Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 298 (2013) 179–195

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

# A Geometric View of Partial Order Reduction

Eric Goubault Tobias Heindel Samuel Mimram[1]

*CEA, LIST*
*Gif-sur-Yvette, France*

**Abstract**

Verifying that a concurrent program satisfies a given property, such as deadlock-freeness, is computationally difficult. Naive exploration techniques are facing the *state space explosion* problem: they consider an exponential number of interleavings of parallel threads (relative to the program size). Partial order reduction is a standard method to address this difficulty. It is based on the observation that certain sets of instructions, called *persistent sets*, are not affected by other concurrent instructions and can thus always be explored first when searching for deadlocks. More recent models of concurrent processes use directed topological spaces: states are points, computations are paths, and equivalent interleavings are homotopic. This *geometric* approach applies theoretical results of algebraic topology to improve verification. Despite the very different origin of the approaches, the paper compares partial-order reduction with a construction of the geometric approach, the category of future components. The main result, which shows that the two techniques make essentially the same use of persistent transitions, is of foundational interest and aims for cross-fertilization of the two approaches to improve verification methods for concurrent programs.

Verifying concurrent programs is a computationally difficult task because one has to check that the desired safety properties are valid for *any possible scheduling* of the program and, typically, the number of schedulings is exponential in the size of the program. For instance, consider the following concurrent program, consisting of two processes in parallel, each of which is modifying the contents of two memory cells:

```
x:=1;y:=2   |   y:=3;z:=4
```
(1)

In the following, we assume that the execution model is sequentially consistent, so that an execution of the program (1) will interleave the instructions of the two processes and thus corresponds to one of the following six sequential programs.

```
x:=1;y:=2;y:=3;z:=4    x:=1;y:=3;y:=2;z:=4    x:=1;y:=3;z:=4;y:=2
y:=3;x:=1;y:=2;z:=4    y:=3;x:=1;z:=4;y:=2    y:=3;z:=4;x:=1;y:=2
```
(2)

In order to verify that the program (1) is correct, one could thus use traditional verification techniques on the six programs (2), which can also be pictured as the

[1] The authors gratefully acknowledge support from the project PANDA ANR-09-BLAN-0169.


http://dx.doi.org/10.1016/j.entcs.2013.09.013

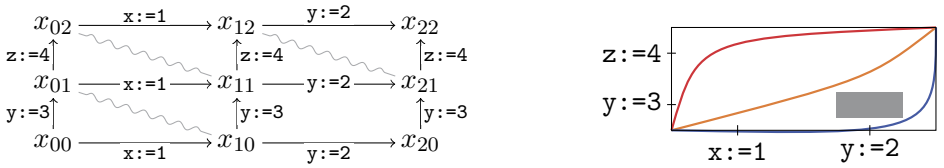



Fig. 1. Asynchronous transition semantics and geometric model of (1).

paths from $x_{00}$ to $x_{22}$ on the left of Figure 1. However, this approach will not scale up because the number of sequential programs corresponding to a concurrent one grows very fast: this phenomenon is called *state space explosion*. In order to avoid it, techniques of *state space reduction* have been invented that exploit the inherent dependencies between instructions. For instance, the order in which `x:=1` and `y:=3` are executed is not relevant for most program properties: both possible executions result in the same memory state in the end. We use *tiles* (marked by ⌒) on the left of Figure 1 to indicate when instructions can be switched in this way, and say that the instructions commute. Formally, the graph with tiles forms an *asynchronous transition system* or ATS for short [19].

Notice that, in this program, the instruction `x:=1` is *persistent* in the sense that it commutes with all possible instructions running in parallel with it (forming the second process). Without loss of generality, we might thus suppose that it is executed first and we are left to verify that the program `x:=1;(y:=2|y:=3;z:=4)` is valid. This program gives rise to only three possible executions, compared to the six of (2): we can avoid examining the paths going through the states $x_{01}$ and $x_{02}$. Of course we can iteratively use this procedure to reduce the program further, which results in removing paths going through the state $x_{12}$. This procedure, introduced by Valmari and developed by Godefroid [7], is called *partial order reduction* (POR) and has lead to a wide variety of successful tools such as SPIN [14] (where the above mentioned persistent sets are complemented with other techniques such as sleep sets). These tools have been originally devised to optimize deadlock detection (and have been extended afterwards in various ways). Thus, we shall focus on deadlock detection, though the intended range of application is wider (full reachability will be detailed in future works); moreover, we shall mainly be concerned with acyclic systems, which is also a common restriction in techniques of partial order reduction (see for example [6]).

Independently of advances of POR, work on *topological semantics* of concurrent programs has lead to new techniques based on similar observations, but formulated in a much different context [8,9]. In this line of research, concurrent programs are modeled as directed topological spaces [4]. For instance, on the right of Figure 1, we see the space associated to the program (1): ignoring the curvy paths at this point, the space is essentially a filled square with a square hole (rendered in gray). Notice the strong resemblance between the space and the asynchronous transition system on the left: in some precise sense, the topological model is actually the geometrical counterpart of the ATS (*viz*, its *geometric realization*): constructions in one setting can be reformulated in the other; avoiding the formal details about topology, we appeal to geometric intuition to illustrate the main ideas.

In the geometric approach, program executions are modeled as paths in a space, as shown in Figure 1. For instance the path that passes underneath the hole corresponds to the first possible scheduling in (2). Notice that paths should always be monotone when projected to a coordinate; this captures progression in time of each thread of a program. This is why topological spaces of the model need to be endowed with a notion of *direction* [13]. In these topological models, two interleavings that can be obtained from each other by repeatedly switching commuting instructions are represented as *dihomotopic* paths, i.e. paths which can be continuously deformed into each other. For instance, on the right of Figure 1, the two paths that pass on top of the hole are dihomotopic, but none of them is dihomotopic to the one that passes underneath; the reason is that the hole is an obstacle to continuous deformation of paths below and above the hole.

The interest of defining a program semantics in terms of topological spaces, is that it allows one to reuse concepts and tools coming from algebraic topology. This has enabled the formulation of state space reduction methods as follows. From a concurrency point of view, a path is *inessential* if it does not change the possible future paths, up to homotopy: such a path corresponds to an execution where neither the program has made a significant choice (such as choosing a branch of a conditional choice) nor the scheduler (such as ordering two concurrent actions which do not commute). Using a suitable formalization, one is naturally lead to consider the category of paths in the topological semantics of a program, quotiented by inessential paths in order to only retain the structure which is relevant for studying the program up to commutation of instructions, i.e. define a notion of reduced state space. This category, introduced in [11], is called the *category of future components* and is used in the tool ALCOOL [4,9], which has been successfully used in an industrial context [2] for deadlock detection.

Even though there is a striking similarity between the notions of persistent sets and inessential paths, the relationship between the two has never been formally studied and the purpose of this article is to fill this gap, in order for partial order reduction and geometric techniques to improve each other and combine their potential to alleviate the state space explosion problem. As a result, we are able to show that, under fairly reasonable assumptions, persistent transitions are the algebraic counterpart of inessential paths. Despite the fact that the analogy is intuitive, it turns out that the theoretical comparison is sometimes technically involved. On a more practical side, a preliminary comparison, based on experiments, was started in [4].

In Section 1, we begin with a review of the models of computations used here to formalize persistent sets: *labeled transitions systems with independence* (LTSI), which are generalized into *asynchronous transitions systems* (ATS). In Section 2, we conservatively extend the original definition of persistent sets from LTSI to ATS, which are closer to geometric models, and show that they retain their fundamental reduction potential to prune the search space for deadlock detection. Finally, we develop in Section 3 a first conceptual link between partial order reduction and directed algebraic topology; in Theorem 3.10, we make precise in what sense *persistent*

*singletons* are essentially the same as *inessential morphisms*, thus identifying a common concept of geometrical and partial order reduction methods (amongst the host of techniques and heuristics that are used in both approaches). This is further discussed in our summary in Section 4, together with venues for future research.

# 1 Models of concurrent computation

We shall use two models for concurrent computations: labeled transitions systems with independence (LTSI), as traditionally used in the POR technique, and asynchronous transition systems (ATS), which can be seen as algebraic counterparts of the directed topological spaces of the geometric point of view. Both of these formalize the state spaces of programs, such as the example given in Figure 1.

## *1.1 Labelled transition systems with independence*

A labeled transition system (LTS) is a triple $(S, \Lambda, \to)$ where $S$ is a set of *states*, $\Lambda$ is a set of *transition labels*, and $\to \subseteq S \times \Lambda \times S$ is a *transition relation*. We write $s \xrightarrow{t} s'$ whenever $(s, t, s') \in \to$ and $s \xrightarrow{t}$ when $t \in \Lambda$ is *enabled* in $s$, i.e. when there exists a state $s'$ such that $s \xrightarrow{t} s'$. We shall here consider only deterministic transition systems, i.e. $s \xrightarrow{t} s'$ and $s \xrightarrow{t} s''$ imply $s' = s''$.

Two (labels of) transitions are considered to be independent if the relative order in which they are performed does not matter, in the sense that both schedulings are essentially the same computation and in particular yield the same result. This notion of independence motivates the following definition [7].

**Definition 1.1 (Independence)** *A* labeled transition system with independence (LTSI) *consists of an* LTS $(S, \Lambda, \to)$ *with an* independence relation $\|$*, which is a symmetric, irreflexive relation* $\| \subseteq \Lambda \times \Lambda$ *such that for each pair* $(t_1, t_2) \in \|$ *and every reachable state* $s$*,*

(i) *if* $s \xrightarrow{t_1} s'$ *then* $s' \xrightarrow{t_2}$ *iff* $s \xrightarrow{t_2}$*; and*

(ii) *if both* $s \xrightarrow{t_1}$ *and* $s \xrightarrow{t_2}$ *hold, there exists a unique* $s''$ *such that (for a unique pair of states* $s_1, s_2 \in S$*) both* $s \xrightarrow{t_1} s_1 \xrightarrow{t_2} s''$ *and* $s \xrightarrow{t_2} s_2 \xrightarrow{t_1} s''$ *hold.*

## *1.2 Asynchronous transition systems*

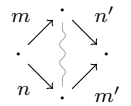
Geometric models for concurrency have been introduced with the idea to apply tools developed in algebraic topology to the analysis of concurrent programs [8]; those programs are considered as directed topological spaces, in which (directed) paths correspond to particular executions of the program, and dihomotopy between paths is an equivalence of executions (relating schedulings of a concurrent program that lead to the same result). For the analysis of concurrent systems, the topological formalization can be replaced by its algebraic counterpart, namely asynchronous transition systems (or more generally precubical sets), which brings us closer to the formalism of LTSI as we recall here, see [12] for a detailed comparison between those

models.

Recall that a *graph* $G = (V, E, \mathrm{src}, \mathrm{tgt})$ consists of a set $V$ of *vertices*, a set $E$ of *edges*, and two functions $\mathrm{src}, \mathrm{tgt} \colon E \to V$, which associate to each edge its *source* and *target*, respectively. A *path* $u$ from vertex $x$ to vertex $y$, denoted by $u \colon x \twoheadrightarrow y$, is a sequence of consecutive edges; the empty path on a vertex $x$ is denoted by $\varepsilon_x \colon x \twoheadrightarrow x$. The concatenation of two paths $u \colon x \twoheadrightarrow y$ and $v \colon y \twoheadrightarrow z$ is denoted by $u \cdot v \colon x \twoheadrightarrow z$.

**Definition 1.2 (Asynchronous transition system)** *An* asynchronous transition system*, or* ATS *for short, is a pair* $(G, \diamond)$ *where* $G$ *is a graph (whose vertices are called* positions *and edges are called* transitions*) and* $\diamond$ *is a relation on the set of paths of length two that have the same source and target, i.e.* $(u, v) \in \diamond$ *only if* $u, v \colon x \twoheadrightarrow y$*. The elements of* $\diamond$ *are called* tiles*. Two transitions* $m \colon x \to y$ *and* $n \colon x \to z$ *are* independent*, written* $m \parallel n$*, if there exist transitions* $m'$ *and* $n'$ *such that* $(m \cdot n') \diamond (n \cdot m')$*, i.e. we have the tile as in* (3)*, on the right below.*

To formalize the fact that we can reschedule independent events in the run of a concurrent system without changing the actual computation that is performed, we use the following definition of trace, which refines Mazurkiewicz's notion and is the algebraic counterpart of homotopy in topological spaces.



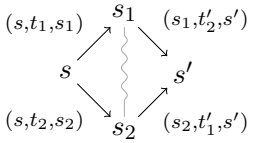
(3)

**Definition 1.3 (Trace)** *Two paths* $u, v \colon x \twoheadrightarrow y$ *are* homotopic*, written* $u \sim v$*, if* $u$ *can be obtained from* $v$ *by repeatedly replacing path fragments* $m \cdot n'$ *by* $m' \cdot n$ *whenever there exists a tile* (3)*, i.e. the relation* $\sim$ *is the smallest congruence w.r.t. path composition that extends* $\diamond$*. A* trace *is an equivalence class w.r.t.* $\sim$ *and we write* $[u]$ *for the equivalence class of a path* $u$*.*

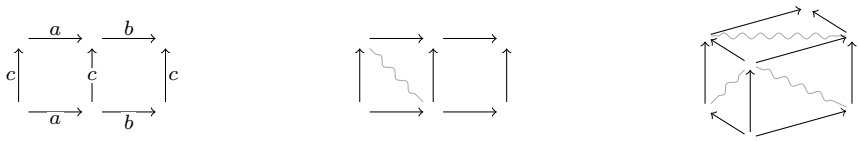
**Definition 1.4 (Trace category)** *The* trace category *associated to an* ATS $(G, \diamond)$ *has the vertices of* $G$ *as objects and traces* $[u]$ *with* $u : x \twoheadrightarrow y$ *as morphisms from* $x$ *to* $y$*. Composition is the operation induced on traces by path concatenation and identities are empty traces* $[\varepsilon_x]$*.*

In order to relate this model with the one introduced in previous section, we should remark that each LTSI (Definition 1.1) naturally induces an ATS as follows.

**Definition 1.5 (Induced ATS)** *Let* $(\mathbb{S}, \|)$ *be an* LTSI *where* $\mathbb{S} = (S, \Lambda, \to)$*. Its* induced ATS*, denoted by* $\mathrm{ATS}^{\|}_{\mathbb{S}} = (G, \diamond)$*, has vertices as positions and edges as transitions, i.e.* $V = S$ *and* $E = \to$*; moreover, for each* $e = (s, t, s') \in E$ *we have* $\mathrm{src}(e) = s$*,* $\mathrm{tgt}(e) = s'$*, and* $\diamond$ *contains a tile as pictured on the right whenever* $t_1 \parallel t_2$ *and* $s \xrightarrow{t_1}$ *and* $s \xrightarrow{t_2}$*.*




**Example 1.6** Consider the LTSI on the left below with $\| = \{(a, c), (c, a)\}$; its associated ATS is shown in the middle. Notice that no LTSI can generate the ATS on the right, because to generate its transitions, we would have to generate the "full cube" as stated in Lemma 2.11. In this sense, ATS are more general than LTSI.




A labeled variant of ATS can easily be defined and all the constructions performed on ATS in this article can be generalized to the labeled case (see [12] for the precise relations between those models). However, the labels of transitions are not really needed since a suitable notion of transition label can be recovered abstractly as its associated event:

**Definition 1.7 (Event)** *Let $(G, \diamond)$ be an* ATS*. Two transitions $m$ and $m'$ are* parallel*, written $m \,\square\, m'$, if they form opposite sides of some tile* (3)*, i.e. if there exists a tile $(m \cdot n', n \cdot m') \in \diamond$ for some transitions $n$ and $n'$. An* event *is an equivalence class w.r.t. the least equivalence relation on transitions that contains $\square$.*

For instance, in the middle ATS in Example 1.6, the two vertical transitions on the left are elements of the same event, while the rightmost is not: considering the LTSI on the left, since $b$ and $c$ are not independent, performing $c$ before or after $b$ does not correspond to the same event – even though they carry the same label.

**Remark 1.8** In the sequel, we shall restrict to ATS that have unique ways to "close" tiles, and "switchings" of transitions are uniquely defined. Formally, the former means that $(m \cdot p) \diamond (n \cdot q)$ and $(m \cdot p') \diamond (n \cdot q')$ imply $p = p'$ and $q = q'$ while the latter says that $u \diamond v$ and $u \diamond w$ imply $v = w$. All ATS that are induced by LTSI satisfy this property.

## 2 Persistent sets in asynchronous transition systems

Persistent sets are one of the most well-known techniques to reduce the number of executions to be explored to check that a concurrent program cannot lead to a deadlock state. Here, we generalize their definition to ATS to compare it with geometric reduction techniques in Section 3. We begin by recalling Godefroid's original definition [7].

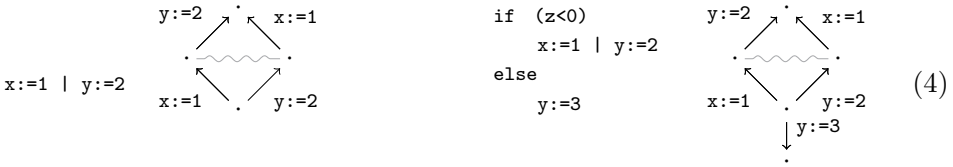
**Definition 2.1 (Persistent set)** *Given an* LTSI $(S, \Lambda, \to, \|)$ *and a state $s \in S$, a set $R \subseteq \Lambda$ of transition labels that are enabled in a state $s \in S$ is* persistent *in $s$, if for all nonempty transition sequences*

$$s = s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots \xrightarrow{t_{n-1}} s_n \xrightarrow{t_n} s_{n+1}$$

*from $s$ that satisfy $t_i \notin R$ $(1 \leq i \leq n)$, $t_n \parallel t$ holds for every transition $t \in R$.*

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

**Example 2.2** Consider the programs below, with their respective LTSI semantics.




(4)

In the LTSI on the left, we have `x:=1` and `y:=2` running completely in parallel; thus, both $\{\texttt{x:=1}\}$ and $\{\texttt{y:=2}\}$ are persistent sets at the initial state. In contrast, in the system on the right, the transitions `x:=1` and `y:=2` are in conflict with `y:=3`; the only persistent set consists of a "monolithic" component with no threads running concurrently, i.e. $\{\texttt{x:=1}, \texttt{y:=2}, \texttt{y:=3}\}$ is the only persistent set at the initial state. As a final example, in Figure 1, $\{\texttt{x:=1}\}$ and $\{\texttt{x:=1}, \texttt{y:=3}\}$ are persistent in the initial state while the set $\{\texttt{y:=3}\}$ is not.

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

**Proposition 2.3 (Persistent deadlock reachability [7, Theorem 4.3])** *Given a choice of a non-empty persistent set $R_s$ at each state $s$ that is not a deadlock, for each path $s_0 \xrightarrow{t_0} s_1 \dots s_n \xrightarrow{t_n} d$ to a deadlock $d$, there exists a path $s_0 \xrightarrow{t'_0} s'_1 \dots s'_n \xrightarrow{t'_n} d$ such that $t'_i \in R_{s_i}$ for all $i \in \{0, \dots, n\}$.*

The proof of the preceding proposition is based on the following observation: for each path $u : s \twoheadrightarrow d$ leading to a deadlock $d$ and persistent set $R$ at $s$, there exists a path $v \in [u]$ (i.e. $v \sim u$) whose initial transition is in $R$. This motivates our generalization of the definition of persistent sets, based on the following definitions:

**Definition 2.4** *Let $(G, \diamond)$ be an* ATS. *Given a path $u\colon x \twoheadrightarrow z$, a transition $m\colon x \to y$ is* initial modulo homotopy *if there exists a path $v\colon y \twoheadrightarrow z$ such that $u \sim m \cdot v$; the set of all transitions that are initial modulo homotopy in $u$ is denoted by $\tilde{\imath}(u)$. Two paths $u\colon x \twoheadrightarrow y$ and $v\colon x \twoheadrightarrow z$ with common source $x$ are* compatible, *written $u \uparrow v$, if there exist paths $w_y\colon y \twoheadrightarrow x'$ and $w_z\colon z \twoheadrightarrow x'$ with common target $x'$ such that $u \cdot w_y \sim v \cdot w_z$.*

Thus, in particular all transitions that are initial modulo homotopy in some path are pairwise compatible. These notions enable us to generalize persistence to ATS as follows:

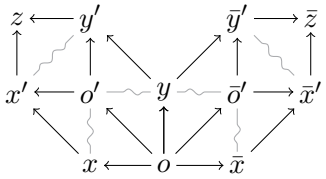
**Definition 2.5 (Homotopy persistent set)** *Let $R$ be a set of transitions in an* ATS *$(G, \diamond)$ that share a state $x$ as common source. The set $R$ is* homotopy persistent, *if each path $u\colon x \twoheadrightarrow z$ is compatible with all transitions in $R$ provided that no transition in $R$ is amongst its homotopy initial ones, i.e.*

$$\forall u\colon x \twoheadrightarrow z, \qquad R \cap \tilde{\imath}(u) = \varnothing \qquad \Rightarrow \qquad \forall m \in R.\ u \uparrow m.$$

**Remark 2.6** A *persistent singleton* (a persistent set with a single element) corresponds to a transition that is compatible with all paths with the same source.

**Example 2.7** The singleton $\{o \to y\}$ is an homotopy persistent set in the ATS below in (5). Note that the transition $o \to y$ is compatible with the transition $o \to x$ even though the two transitions are not "independent".

Situations like this typically arise in consumer producer problems where $n$-ary semaphores are used to ensure that an exhausted resource is not used, such as when implementing a queue with limited size. For instance, suppose that we have a queue in which we can `put` at most two elements (if there are already two elements, the `put` operation blocks until an element is `taken` from the queue).




(5)

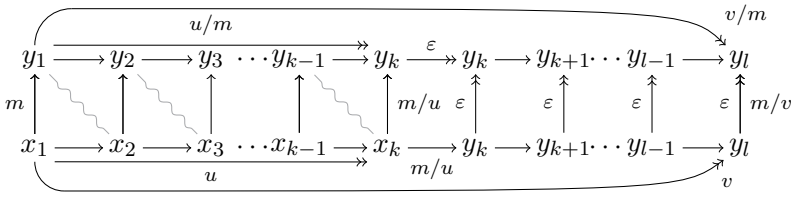
The following program generates the above ATS (the arrow subscripts indicate the direction of the corresponding transitions).

$$\texttt{put}_{\uparrow} \quad | \quad \texttt{if (...)} \ \{\texttt{take}_{\nearrow} \ | \ \texttt{put}_{\rightarrow}\} \ \texttt{else} \ \{\texttt{take}_{\nwarrow} \ | \ \texttt{put}_{\leftarrow}\}$$

It remains to show that the notion of homotopy persistent set is in fact a conservative extension of the original one (Definition 2.1). The proof is based on the observation that in each ATS that is induced by an LTSI, a transition $m : x \to y$ has a unique residual path $u/m : y \twoheadrightarrow z'$ after a compatible path $u : x \twoheadrightarrow z$; we say this kind of ATS *has compatible residuals*. The residual $u/m$ of $m$ after $u$ has intuitively the "same effect" as $u$, once $m$ has been performed. The assumption made in Remark 1.8 is necessary for the following definition to be sound.

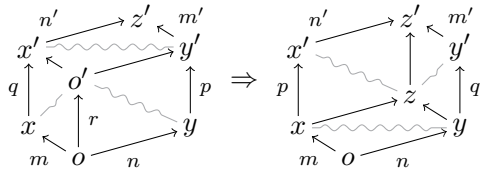
**Definition 2.8 (Residual)** *Let $m\colon x \to y$ be a transition, let $u : x \twoheadrightarrow z$ be a path. The* residual *of $u$ after $m$, denoted by $u/m$, and the* residual *of $m$ after $u$, denoted by $m/u$, are defined by induction on the length of $u$ as follows.*

- $\varepsilon_x/m = \varepsilon_y$ *and* $m/\varepsilon_x = m$
- $(m \cdot u')/m = u'$ *and* $m/(m \cdot u') = \varepsilon$
- *If* $n \neq m$, $(n \cdot u)/m = n' \cdot (u/m')$ *where $m'$ and $n'$ are transitions such that $m \cdot n' \diamond n \cdot m'$ as in (3) (where $m'$ is uniquely determined by Remark 1.8).*




**Remark 2.9** The residual $m/u$ of a transition $m$ after a path $u$ (when it exists) is either a transition or empty (as illustrated above). It is straightforward to extend the definition to the residual $v/u$ of a path $v\colon x \twoheadrightarrow z$ after a path $u\colon x \twoheadrightarrow y$.

The fact that every ATS that is induced by an LTSI has compatible residuals can be deduced from the fact that they satisfy a particular diagrammatic property; it can be expressed as follows [17].




(6)

**Definition 2.10 (Forward Cube Property)**
*An* ATS *has the* Forward Cube Property *(or* FCP*) if for every three tiles as shown on the left in* (6) *there exist three matching tiles as shown on the right in* (6).

**Lemma 2.11** *The induced* ATS *of an* LTSI *has the* FCP *property.*

Our main interest in this property is the following property [16]:

**Proposition 2.12 (Residuation)** *An* ATS *with the* FCP *has compatible residuals.*

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

**Proposition 2.13 (Homotopy persistent is persistent)** *Let* $(S, \Lambda, \to, \|)$ *be an* LTSI*, let* $x$ *be a position, and let* $R \subseteq \Lambda$ *be a set of transitions that are all enabled at* $x$*. The set* $R$ *is persistent at* $x$ *if and only if the set* $R' = \{(x, t, y) \mid t \in R, x \xrightarrow{t} y\}$ *is homotopy persistent at* $x$ *in* $\mathrm{ATS}^{\|}_{\mathbb{S}}$.

**Proof.** If $R$ is persistent, it is easy to show that $R'$ is homotopy persistent since if a transition label in $R$ is independent with all transitions labels on the path, it is in particular compatible with the corresponding transitions (by Definition 2.8). Conversely, assume that $R'$ is homotopy persistent. Since $\mathrm{ATS}^{\|}_{\mathbb{S}}$ has the FCP by Lemma 2.11, for every path $u\colon x \twoheadrightarrow z$, we can use Proposition 2.12 to show that either some residual of a transition in $R'$ occurs in $u$ (if $\tilde{\iota}(u) \cap R' \neq \varnothing$), or we have residuals of all transitions in $R'$ after $u$ (if $u$ is compatible with all transitions in $R'$). It can easily be checked that the residual of any transition in $R'$ always carries the same label as the "original" in $R'$. □

With this result at hand, we refer to homotopy persistent sets in ATS as persistent ones from now on. Moreover, we have a further successful "soundness check" for our generalization of persistent sets, namely, the fundamental fact about reachability of deadlocks "along" persistent sets, namely Proposition 2.3, lifts to any ATS.

**Lemma 2.14** *Let* $G = (G, \diamond)$ *be an* ATS*, let* $u\colon x \twoheadrightarrow d$ *be a path such that* $d$ *is a deadlock and let* $R$ *be a non-empty persistent set at* $x$*. Then* $R$ *contains some of the initial transitions of* $u$ *(i.e.* $u \sim m \cdot v$ *for some* $m \in R$ *and a suitable path* $v$*).*

**Proof.** Consider a path $u\colon x \twoheadrightarrow d$ leading to a deadlock $d$. Suppose that $u$ is compatible with all transitions in $R$. Because $R$ is non-empty, there exists some transition $m\colon x \to y$ in $R$ and paths $u'\colon d \twoheadrightarrow z$ and $v\colon y \twoheadrightarrow z$ such that $u \cdot u' \sim m \cdot v$. Since $d$ is supposed to be a deadlock, we necessarily have $u' = \varepsilon_d$ and $z = d$.

Therefore $u \sim m \cdot v$, i.e. $m \in \tilde{\imath}(u)$ and we conclude. Otherwise, $u$ is incompatible with some transition $m\colon x \to y$ of $R$. Since $R$ is homotopy persistent, $R \cap \tilde{\imath}(u) \neq \varnothing$.□

**Corollary 2.15 (Persistent deadlock reachability)** *Let $(G, \diamond)$ be an* ATS *with $V$ and $E$ as set of vertices and edges respectively, let $R_{\_}\colon V \to \wp(E)$ be a function such that for all non-deadlocking states $x \in V$, the set $R_x$ is a non-empty persistent set at $x$. Given a deadlock $d \in V$ reachable by a path $u : x_0 \twoheadrightarrow d$, there exists a path $v : x_0 \twoheadrightarrow d$ such that $u \sim v$ and every transition $m : x \to y$ occurring in $v$ is persistent at $x$, i.e. $m \in R_x$.*

Note that for arbitrary safety properties, persistent sets alone do not suffice, and one has to use extra techniques similar to the sleep sets of [7].

Finally, the following technical lemma will be useful in the following:

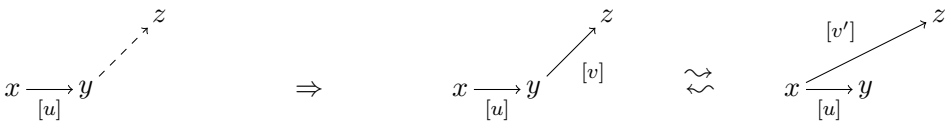
**Lemma 2.16** *In the trace category associated to an* ATS *which satisfies the* FCP *every morphism is epi, that is for every paths $u : x \twoheadrightarrow y$ and $v, w : y \twoheadrightarrow z$, $[u \cdot v] = [u \cdot w]$ implies $[v] = [w]$.*

## 3 Comparing POR and categories of future components

In this section, we relate the notion of persistent set with the construction of the *category of future components* [10], which gives a condensed representation of (geometric models of) concurrent programs by eliminating states that enable only "inessential" transitions. We first reformulate this construction in the setting of ATS, as well as related properties: we introduce the notion of future-reflecting trace, and the category of future components is then defined as the quotient of the category of traces by a consistent set of future-reflecting traces. After that, the crucial point is to make precise which transitions are considered as inessential. Intuitively, one might expect that all persistent transitions are inessential. However, the general definition of ATS allows peculiar situations which do not occur in ATS that are generated by usual programs (for instance persistent transitions might not be stable under residuation). Nevertheless, we will show, for suitably "well-behaved" ATS, that inessential transitions are the same as persistent ones and that the category of future components does only contain states that do not enable persistent transitions.

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

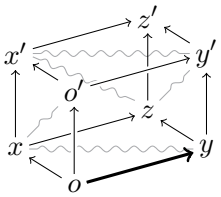
**Definition 3.1 (Future-reflecting trace)** *A trace $[u]\colon x \twoheadrightarrow y$ is* future-reflecting *if for each position $z$ reachable from $y$ (i.e. there exists a path $y \twoheadrightarrow z$), precomposition with $[u]$ induces a bijection between traces from $y$ to $z$ and traces from $x$ to $z$.*



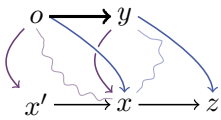
**Example 3.2 (Future-reflecting transition)** Reconsider the ATS in Figure 1. The only important choice for scheduling the transitions concerns the relative order of the instructions `y:=3` and `y:=2`. Namely, the transition $x_{00} \to x_{10}$ (with label `x:=1`) does not influence the choice and thus is intuitively "inessential". In fact, it reflects all futures according to the definition. For example, the two traces from $x_{10}$ to $x_{22}$ factor uniquely through $x_{00} \to x_{10}$. In contrast, the transition $x_{10} \to x_{20}$ (with label `y:=2`) does not reflect futures as there is only one trace from $x_{20}$ to $x_{22}$ while there are two traces from $x_{10}$ to $x_{22}$.

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

**Example 3.3** Consider the cube on the right. All pairs of transitions are independent except for $o \to o'$ and $o \to y$ (the front face of the cube is not a tile). Now, both transitions $o \to y$ and $o \to o'$ are persistent since they are compatible with all other transitions from $o$. However neither of them is a future-reflecting trace. To see that $o \to y$ is not a future-reflecting trace, consider the trace $[y \to y']$. There is only one trace from $y$ to $y'$ but two from $o$ to $y'$. The argument for $o \to o'$ is symmetric. The important point to notice in this example, is that the associated trace category is not a poset (it might have more than one morphism between two objects). In fact, it can be shown that when the trace category is a poset (this is the case for event structures), all persistent transitions are future-reflecting.




In the ATS on the right, the transition $o \to y$ is persistent. However, it is not a future-reflecting trace since the trace $[o \to x]$ does not factor through $[o \to y]$. Also note that this ATS is actually induced by an LTSI.




The basic idea of state space reduction used in the category of future components is that future-reflecting transitions are not informative from a concurrency point of view and thus need not be represented explicitly. So, starting from an ATS, one might be tempted to consider the associated trace category (Definition 1.4) and quotient it by all the future-reflecting traces, which amounts to formally turn them into identities. It turns out that this crushes too much information about traces (see [5], in particular there is no equivalence between the quotient and the fraction category and no compositionality result via Van Kampen theorems). A suitable solution is to quotient wrt a subset of all future-reflecting traces, namely those that are closed under composition and "residuals"; the formal details are as follows.

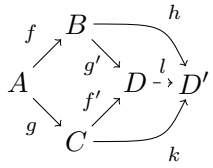
**Definition 3.4 (System of inessentials)** *Let $(G, \diamond)$ be an* ATS*, and let $\Sigma_f$ be a set of traces. The set $\Sigma_f$ is a* system of inessentials (SOI) *if*

(i) *each element of $\Sigma_f$ is a future-reflecting trace;*

(ii) *$\Sigma_f$ contains all empty traces and is closed under composition;*

(iii) $\Sigma_f$ *is stable under pushout, i.e. for every trace* $\sigma : x \twoheadrightarrow z \in \Sigma_f$ *and for any trace* $[u] : x \twoheadrightarrow y$*, there exists a pushout* $z \xrightarrow{[u']} x' \xleftarrow{\sigma'} y$ *of* $z \xleftarrow{\sigma} x \xrightarrow{[u]} y$ *such that* $\sigma' \in \Sigma_f$*.*

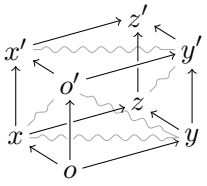
We recall that the *pushout* of two morphisms $f : A \to B$ and $g : A \to C$ in a category $\mathcal{C}$ is a pair of morphisms $g' : B \to D$ and $f' : C \to D$ such that $f' \circ g = g' \circ f$ and moreover for any other pair of morphisms $h : B \to D'$ and $k : C \to D'$ that satisfy $k \circ g = h \circ f$, there exists a unique morphism $l : D \to D'$ such that both $k = l \circ f'$ and $h = l \circ g'$ (as illustrated to the right).




**Definition 3.5 (Inessential trace)** *A trace is* inessential *if it belongs to the union of all systems of inessentials of an* ATS *(which is a non-empty* SOI *which is maximal wrt inclusion).*

In fact, in most of the following examples, the pushout of an inessential trace along another one is just its residual. The following example shows that the set of future-reflecting transitions need not be closed under residuals and thus the maximal SOI does not contain all future-reflecting traces.

**Example 3.6** Consider the ATS on the right, in which all faces but the back face are tiles. The transition $o \to o'$ is future-reflecting. Its residual after $o \to x$, namely $x \to x'$, is not future-reflecting (and not even persistent). In fact, the largest SOI is the closure of $\{o' \to x', o' \to y', y \to z, y \to y'\}$ by residuals, composition, and identities.




Note that the ATS of this example does not satisfy the Forward Cube Property. Nevertheless, the category of future components is well-defined.

**Definition 3.7 (Category of future components)** *The category of future components of an* ATS *is its trace category quotiented by inessential traces.*

This construction amounts to forgetting inessential transitions by considering them as identities. Another point of view, formalized by the following proposition, is that this construction removes states which enable inessential transitions: informally, passing through them does not bring any new information about possible future traces (as no important choice can be made by the program or the scheduler). This agrees with the informal explanations of the introductory example in Figure 1; the state space reduction removes states $x_{01}$, $x_{02}$ and $x_{12}$.

**Proposition 3.8** *Let* $(G, \diamond)$ *be an* ATS*, let* $\Sigma$ *be the maximal* SOI*. If* $\Sigma$ *is finite, the category of future components is the full subcategory of the category of traces that is induced by all states that do not enable any transition in* $\Sigma$*.*

Finally, we give sufficient conditions which imply that the category of future components yields the expected state space. The final condition is the absence of déjà vus (cf. Example 3.3).

**Definition 3.9 (Déjà vu)** *A* déjà vu *is a transition* $m : x \to y$ *such that there exists a path* $u : y' \twoheadrightarrow x$ *and a transition* $m' : x' \to y'$ *in* $u$ *that is the same event as*

*$m$. The* ATS *is* déjà vu free *if none of its transitions are déjà vus.*

In other words, an ATS is déjà vu free if none of its paths contains two transitions that are instances of the same event. For example, in the second ATS of Example 3.3, the transition $x \to z$ is a déjà vu since the path $o \to x' \to x \to z$ contains two occurrences of the same event: $(x \to z)\,\Box(o \to y)\,\Box(x \to x')$. Similarly, any cyclic ATS has déjà vus though déjà vus need not necessarily be cycles. With this final proviso, we obtain a formal correspondence between inessential and persistent transitions (i.e. transitions $m : x \to y$ such that $\{m\}$ is a persistent set at $x$).

**Theorem 3.10 (Inessential vs. persistent transitions)** *In any déjà vu free* ATS *that has the forward cube property, a transition is persistent iff it belongs to some* SOI *(and in particular the maximal one). If the trace category of the* ATS *is finite, the category of future components is the full subcategory containing all objects that do not enable persistent transitions.*

**Proof.** Assume that $\Sigma_f$ is a SOI and $m \in \Sigma_f$ a transition. We want to show that $\{m\}$ is a persistent set. For any path coinitial with $m$, we have a pushout of $m$ along $u$ because $\Sigma_f$ is closed under pushouts. Thus $m \uparrow u$ and $m$ is persistent by Remark 2.6.

Conversely, let $\Sigma'$ be the set of traces that consist only of persistent transitions. It suffices to show that $\Sigma'$ is a SOI. To show that $\Sigma'$ is stable under pushouts, let $m_1 \cdots m_k \in \Sigma'$ and let $u$ be a path coinitial with $m_1$. We successively take residuals of $m_i$ along $u$ using the FCP, Remark 2.6, and Proposition 2.12, and verify that the composite of the residuals is a pushout. Next, we show that persistent transitions are future-reflecting. Let $m\colon x \to y$ be a persistent transition and $z$ a state reachable from $y$. Since the ATS is supposed to have the FCP, the transition $m$ is an epimorphic trace by Lemma 2.16, i.e. for every paths $u$ and $v$ such that $[m \cdot u] = [m \cdot v]$ we have $[u] = [v]$. Namely, in this case we have $u = (m \cdot u)/m \sim (m \cdot v)/m = v$, the middle homotopy being justified by the fact that $m \cdot u \sim m \cdot v$ and residuation is compatible with homotopy [16]. Precomposition with $m$ with traces from $y$ to $z$ is thus injective and it remains to show that it is surjective. Given a path $u\colon x \twoheadrightarrow z$, we have to show that $u$ factors through $m$. The transition $m$ is compatible with $u$ by Remark 2.6, and thus we can pushout $[m]$ along $[u]$, which is just the residual of $m$ after $u$; it must be the identity as we would obtain a déjà vu otherwise and thus, in fact, $[u]$ factors as $[m] \cdot [u']$ where $[u']$ is the pushout of $[u]$ along $[m]$. □

Thus, in a large class of common systems, including those generated by event structures or Petri nets with acyclic causality, persistent singletons are in fact the same as inessential morphisms. Thus, despite the huge gap between the origins of the geometric approach and the POR technique, in a large class of systems, we do not have only "obvious" similarities, but in fact, a formal argument that inessential transitions are exploited in the same way.

# 4 Conclusion

We have developed a conservative extension of persistent sets to asynchronous transition systems (Definition 2.5) that coincides with the original concept on LTSI (Proposition 2.13). This extension forms the base of our comparison of the partial order reduction technique and the geometric approach to state space reduction, since ATS are the algebraic counterpart of geometrical models. In particular, we have shown that our definition retains the main application of persistent sets which consists in pruning the search for deadlocks (Corollary 2.15).

These preliminaries are crucial for our main contribution, which demonstrates the practical relevance of the theoretical construction of the category of future components and further results in [10], where state space reduction is performed for general directed topological spaces. Glossing over details, Theorem 3.10 says that inessential transitions are the same as persistent singletons. As a direct consequence, the construction of the category of future components roughly amounts to the application of the POR technique when we use only persistent *singletons*. Thus, we have found a common core of the geometric approach and the POR technique, while both approaches have additional heuristics and methods to improve performance. There are favorable examples for the geometric approach [4], which motivates future research.

In theory, a fully general correspondence between persistent singletons and inessential transitions is impossible (as witnessed by Example 3.3), which is not surprising given the difference of origins. To gauge the advantages of each of the approaches in practice, a systematic practical comparison of POR and the geometric approach is called for. We further plan extensions of the geometric approach using methods and tools from Petri nets [3,15]. This is motivated by the fact that Petri nets are closer to *both* – the geometric approach and the POR techniques – as witnessed by the study of so-called stubborn sets in Petri nets [18], which are a particular case of persistent sets [7]. The addition of Petri net inspired techniques is based on the observation that the category of future components bears similarities with the facet abstraction for occurrence nets [1] and that recent advances of the geometric approach in [4] use a notion of "weak causality", which is tightly related to classical models such as event structures and Petri net processes [19]. In the end, we expect to obtain representative experimental results, which will complement the theoretical results of the present paper.

# References

[1] S. Balaguer, T. Chatain, and S. Haar. Building tight occurrence nets from reveals relations. In *ACSD*, pages 44–53. IEEE, 2011.

[2] R. Bonichon and al. Rigorous evidence of freedom from concurrency faults in industrial control software. *Computer Safety, Reliability, and Security*, pages 85–98, 2011.

[3] J.-M. Couvreur, D. Poitrenaud, and P. Weil. Branching processes of general petri nets. In *Applications and Theory of Petri Nets*, volume 6709 of *LNCS*, pages 129–148. Springer, 2011.

[4] L. Fajstrup, E. Goubault, E. Haucourt, S. Mimram, and M. Raußen. Trace spaces: An efficient new technique for state-space reduction. In *ESOP*, pages 274–294, 2012.

[5] L. Fajstrup, E. Goubault, E. Haucourt, and M. Raußen. Components of the fundamental category. *Applied Categorical Structures*, 12(1):81–108, 2004.

[6] C. Flanagan and P. Godefroid. Dynamic partial-order reduction for model checking software. In *POPL'05*, pages 110–121, 2005.

[7] P. Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[8] E. Goubault. Geometry and concurrency: a user's guide. *Mathematical Structures in Computer Science*, 10(4):411–425, 2000.

[9] E. Goubault and E. Haucourt. A practical application of geometric semantics to static analysis of concurrent programs. In *Proc. of CONCUR'05*, volume 3653 of *LNCS*, pages 503–517. Springer, 2005.

[10] E. Goubault and E. Haucourt. Components of the Fundamental Category II. *Applied Categorical Structures*, 15(4):387–414, 2007.

[11] E. Goubault, E. Haucourt, and S. Krishnan. Future path-components in directed topology. In *MFPS*, volume 265 of *ENTCS*, pages 325–335, sep 2010.

[12] E. Goubault and S. Mimram. Formal relationships between geometrical and classical models for concurrency. *ENTCS*, 2012.

[13] M. Grandis. *Directed Algebraic Topology: Models of Non-Reversible Worlds*, volume 13 of *New Mathematical Monographs*. Cambridge University Press, 2009.

[14] G. Holzmann. The Model Checker SPIN. *IEEE Trans. Soft. Eng.*, 23(5):279–295, 1997.

[15] V. Khomenko and A. Mokhov. An Algorithm for Direct Construction of Complete Merged Processes. In *Applications and Theory of Petri Nets*, volume 6709 of *LNCS*, pages 89–108. Springer Berlin, 2011.

[16] S. Mimram. *Sémantique des jeux asynchrones et réécriture 2-dimensionnelle*. PhD thesis, Université Paris Diderot, UFR d'Informatique, 2008.

[17] R. Morin. Concurrent Automata vs. Asynchronous Systems. In *MFCS*, volume 3618 of *LNCS*, pages 686–698. Springer, 2005.

[18] A. Valmari. Stubborn sets for reduced state space generation. In *Applications and Theory of Petri Nets*, pages 491–515, 1989.

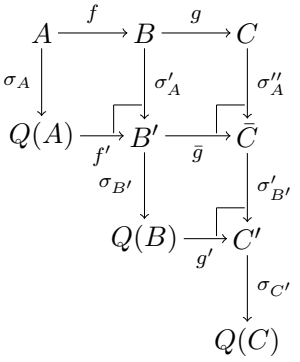
[19] G. Winskel. *Handbook of Logic in Computer Science*, volume 4: Semantic Modelling, chapter 1: Models for concurrency. Oxford University Press, 1995.

# A Characterizing the category of future components

**Proof.** Let $\mathcal{D} \subseteq \mathcal{C}$ be the maximal full subcategory such that for all $\sigma\colon A \to B \in \Sigma$, if $A \in \mathcal{D}$ then $\sigma = \mathrm{id}_A$.

Now we define a candidate for a "quotient" functor $Q\colon \mathcal{C} \to \mathcal{D}$. For each object $A \in \mathcal{C}$, consider all arrows in $\Sigma$ with domain $A$. Since $\Sigma$ is finite, we can take the colimit of the corresponding diagram (using successive pushouts) to obtain the colimit object $Q(A)$; moreover, we have a unique arrow $\sigma_A\colon A \to Q(A) \in \Sigma$ for each $A \in \mathcal{C}$. For each arrow $f\colon A \to B$, we know how to construct the arrow $\sigma_A\colon A \to Q(A)$; to define $Q(f)$, let $Q(A) \xrightarrow{f'} B' \xleftarrow{\sigma'} B$ be the pushout of $Q(A) \xleftarrow{\sigma_A} A \xrightarrow{f} B$. By pushout-stability $\sigma' \in \Sigma$. Moreover, we have the arrow $\sigma_{B'}\colon B' \to Q(B') = Q(B)$ (as $\Sigma$ is "confluent"). Define $Q(f) := \sigma B' \circ f'$. It easily be verified that $Q$ is actually

a functor: the relevant diagram for composition is




where we have $Q(f) = \sigma_{B'} \circ f'$ and $Q(g) = \sigma_{C'} \circ g'$, and $Q(g \circ f) = (\sigma_{C'} \circ \sigma'_{B'}) \circ (\bar{g} \circ f')$, by definition of $Q$; the latter implies $Q(g) \circ Q(g) = Q(g \circ f)$. Moreover the functor $Q$ is epi and in fact a section of the inclusion $\mathcal{D} \subseteq \mathcal{C}$. This implies that $\mathcal{D}$ satisfies the universal property of the quotient category $\mathcal{C}/_\Sigma$ and thus $\mathcal{D} \cong \mathcal{C}/_\Sigma$. □

# B Residuals, Pushouts and Epimorphic Transitions

This section mainly concerns ATS that are induced by LTSI. Thus, we will assume that we have *normal* tiles, i.e. in all ATSs that we consider, switchings are unique and pairs of co-final and co-initial transitions in each tile have different transitions as components.

**Definition B.1 (Switching Distance)** *Let $s, t \colon x \twoheadrightarrow y$ be parallel paths that are homotopic $s \sim t$. The* switching distance *of $s$ and $t$, written $\|s, t\|$, is the minimal number $i$ such that $s \ (\overset{1}{\sim})^i \ t$ where $r \overset{1}{\sim} r'$ if $r = u \cdot v \cdot w$ and $r' = u \cdot v' \cdot w$ for some $(v, v') \in \diamond$.*

Clearly, we have $\|s, t\| = \|t, s\|$.

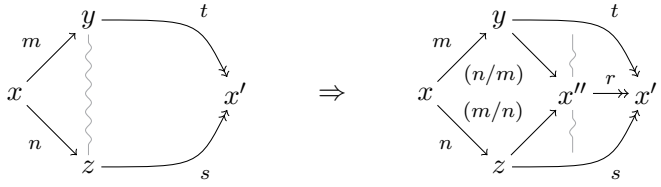



Fig. B.1. Illustration of Lemma B.2

**Lemma B.2 (Residuals of Compatible Transitions)** *Let $(G, \diamond)$ be an* ATS *with the* FCP. *Let $m \colon x \to y$ and $n \colon x \to z$ be two different co-initial transitions; further let $t \colon y \twoheadrightarrow x'$ and $s \colon z \twoheadrightarrow x'$ be co-final paths such that $m \cdot t \sim n \cdot s$. In this situation, $m \parallel n$ and there exists a path $r \colon x'' \twoheadrightarrow x'$ such that $t \sim (n/m) \cdot r$, $(m/n) \cdot r \sim s$, and*

$$\|m \cdot t, n \cdot s\| = \|t, (n/m) \cdot r\| + 1 + \|(m/n) \cdot r, s\|. \tag{B.1}$$

**Proof.** The proof is by induction on the distance $\|m\cdot t, n\cdot s\|$, which is not zero as we have $m \neq n$ by assumption and the convention that in each ATS, the pairs of co-final and co-initial transitions in each tile have different transitions as components.

$\|m \cdot t, n \cdot s\| = 1$**:** In this case, again as $m \neq n$, necessarily $m \parallel n$. Thus there exists a path $r\colon x'' \to x'$ such that $t = (n/m) \cdot r = (m/n) \cdot r = s$, which implies (B.1).

$\|m \cdot t, n \cdot s\| > 1$**:** Now there are paths $u_0, u_1, \ldots, u_i$ with $i = \|m \cdot t, n \cdot s\|$ such that $u_0 = m \cdot t$ and $u_i = n \cdot s$, and $u_j \overset{1}{\sim} u_{j+1}$ for all $j \in \{0, i-1\}$. Now we distinguish the following two cases.

(i) There is some $k \geq 0$ such that for all $j > k$, there exists $u'_j$ such that $u_j = n \cdot u'_j$ and for all $j \leq k$, there exists $u'_j$ such that $u_j = m \cdot u'_j$. Thus, there exists a path $r\colon x'' \to x'$ such that

$$u_k = m \cdot (n/m) \cdot r \overset{1}{\sim} n \cdot (m/n) \cdot r = u_{k+1}.$$

This already implies (B.1).

(ii) For some $i > j > 0$, the first transition of $u_j$ is neither $m$ nor $n$. Thus let $j$ be the maximal index such that $u_{j+1}$ factors as $n \cdot u'_{j+1}$. Thus, there exists $o$ such that $o \parallel n$, and a path $\bar{r}\colon \bar{x} \twoheadrightarrow x'$ such that $u_j = o \cdot (n/o) \cdot \bar{r}$ and $u_{j+1} = n \cdot (o/n) \cdot \bar{r}$.

At this point, we have $m \cdot t \sim o \cdot (n/o) \cdot \bar{r}$ and, in fact, we can use the induction hypothesis to derive that $m \parallel o$. Thus, there exists $r'$ such that $t \sim (o/m) \cdot r'$ and $(m/o) \cdot r' \sim (n/o) \cdot \bar{r}$. Using the convention of unique switchings in every ATS, we see that $(m/o) \neq (n/o)$. Thus, using the induction hypothesis once more, we obtain $r\colon \tilde{x} \twoheadrightarrow x'$ such that $r' \sim ((n/o)/(o/m)) \cdot r$ and $((o/m)/(n/o)) \cdot r \sim \bar{r}$. Finally, we apply the FCP property to the relevant cube starting with transitions $n$, $m$ and $o$. From this, we can conclude that $r$ is in fact a suitable path as the "two ways to go around a cube" take the same number of switchings, namely three.

□

This lemma is already enough to show that any ATS that is induced by an LTSI has compatible residuals (using a straightforward induction on the length of paths). Moreover, we have the following two corollaries.

**Corollary B.3** *Every transition is epimorphic.*

**Proof.** Suppose that $m \cdot t \sim m \cdot s$. We have to show that $t \sim s$. Now there are paths $u_0, u_1, \ldots, u_i$ with $i \geq 0$ such that $u_0 = m \cdot t$ and $u_i = m \cdot s$. If all $u_j$ have $m$ as initial transition, we obtain the desired from the assumption of *normal* tiles (cf. Remark 1.8), and in particular there is no tile $(m \cdot p, m \cdot q)$. Thus, suppose there is at least one $u_j$ such that $u_j = n \cdot u'_j$. Now we apply Lemma B.2 twice to obtain $t \sim (n/m) \cdot r$, $(m/n) \cdot r \sim (m/n) \cdot r'$ and $(n/m) \cdot r' \sim s$. Thus $t \sim s$. □

**Corollary B.4** *Residuals are pushouts.*