Connectivity of spaces of directed paths in geometric models for concurrent computation

Raussen, Martin

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

CONNECTIVITY OF SPACES OF DIRECTED PATHS IN GEOMETRIC MODELS FOR CONCURRENT COMPUTATION

MARTIN RAUSSEN

ABSTRACT. Higher Dimensional Automata (HDA) are higher dimensional relatives to transition systems in concurrency theory taking into account to which degree various actions commute. Mathematically, they take the form of labelled cubical complexes. It is important to know, and challenging from a geometric/topological perspective, whether the space of directed paths (executions in the model) between two vertices (states) is connected; more generally, to estimate higher connectedness of these path spaces.

This paper presents an approach for such an estimation for particularly simple HDA modelling the access of a number of processors to a number of resources with given limited capacity each. It defines a spare capacity for a concurrent program with prescribed periods of access of the processors to the resources. It shows that the connectedness of spaces of directed paths can be estimated (from above) by spare capacities. Moreover, spare capacities can also be used to detect deadlocks and critical states in such a HDA.

The key theoretical ingredient is a transition from the calculation of local connectedness bounds (of the upper links of vertices of an HDA) to global ones by applying a version of the nerve lemma due to Anders Björner.

1. PV PROGRAMS AND THEIR GEOMETRIC SEMANTICS

1.1. **Euclidean cubical complexes and path spaces.** In this paper, we consider an old and simple model for concurrent computation, the so-called PV models going back to Dijkstra [4]: In the simplest situation, consider n processors $p_j \in P$ each executing a linear program. During execution, a processor may lock (Pr) and relinquish (Vr) one or several resources r from a pool R of resources; possibly several times. Any resource $r \in R$ has capacity to serve up to $\kappa(r) \in \mathbf{N}$ processors at any given time.

Every execution of a linear program for a given processor corresponds to a directed map $p_j : I = [0,1] \to I_j$; here, I_j is a finite closed interval; and directed means continuous and (not necessarily strictly) increasing. Disregarding conflicting locks in the first place, any concurrent execution corresponds to a (componentwise) directed path $p : I \to \prod_{j=1}^{n} I_j \subseteq \mathbf{R}^n$. Capacity restrictions disallow these d-paths to enter a forbidden region $F \subset I_j \subseteq \mathbf{R}^n$ (at least one of the resources locked by more processors than its capacity allows); the "allowed" state space $X \subset \mathbf{R}^n$ for such a PV program is the complement of the forbidden region F within $\prod_{j=1}^{n} I_j$. For details cf Section 2.2 or the more comprehensive description in Fajstrup etal. [6, ch. 3].

This state space is a very particular simple case of a pre-cubical set or rather its geometric realization (cf [6, ch. 3.4], Ziemiański [21, Sect. 1], [22, Sect. 2.4]) underlying general Higher Dimensional Automata introduced by Pratt [11] and analyzed by van Glabbeek, cf eg [18]). It has a natural embedding into $\mathbf{R}^n$ (a Euclidean cubical complex); as such, it is non-self-linked and proper, cf Ziemiański [21, Sect. 1].

Key words and phrases. Higher Dimensional Automata, directed path, spare capacity, connectivity, nerve lemma, deadlock, critical state.

The space of all executions from an initial state s to a final state t in state space X corresponds to the space $\vec{P}(X)_s^t$ of all *directed* paths $p : I \to X$ with $p(0) = s$ and $p(1) = t$; equipped with the compact-open topology (uniform convergence). Various simplicial models of such spaces have (in far more general situations) been described in the literature (in particular Raussen [14, 15], Fajstrup etal [6] and Ziemiański [21, 22]. The most notable result (Ziemiański [20, Theorem 5.9]) reports that they can be arbitrarily complicated: For every finite simplicial complex C there exists a PV program with a state space X and vertices v and w such that the path space $\vec{P}(X)_v^w$ has a connected component that is homotopy equivalent to that complex C. On the positive side, it was shown that, roughly speaking, path connectivity locally everywhere implies global path connectivity of all path spaces $\vec{P}(X)_s^t$ (cf Raussen [12, Prop. 2.18], Belton etal [1, Theorem 2]). This is quite important: Connectedness of path space means that any two directed paths are d-homotopic (homotopic through a one-parameter family of d-paths), and thus any two executions are equivalent and yield the same result (cf [6]).

Throughout most of this paper, we stick to concurrent executions of *linear* programs. This might seem very restrictive and unrealistic; processors usually execute programs on *directed graphs*, allowing branchings and loops. The space of all such executions between a source and a target decomposes (by unfolding) to a disjoint union of spaces of executions along the linear directed paths within such a directed graph. As a consequence, the space of all executions in a concurrent program splits into spaces of executions along n-tuples of such linear directed paths; for these linear subspaces, the methods developed below apply. For details, consult Section 4.3.

1.2. A short overview.

1.2.1. *The main lines.* The present paper yields a quite simple numerical criterion ensuring local connectivity of path spaces which, by an inductive argument using machinery from combinatorial topology (cf Björner [3, Theorem 6]), implies global connectivity. We call the number in question the *spare capacity* of the program, cf Definition 4.1. The spare capacity allows to estimate not only path connectivity but also *higher connectivity* of spaces of directed paths between vertices in the associated state space.

More precisely, take departure in a PV-program X on n processors $p \in P$ with several shared resources r within a pool R of resources, each with a capacity $\kappa(r) < n$. We determine, by a simple calculation, the spare capacity $\kappa(X)$ of X, a numerical invariant – depending on the individual capacities and on the intersection patterns of intervals on which these resources are jointly called by the processors – that allows to determine the minimal (higher) connectivity of path spaces $P(X)_s^t$ (with a fixed target t that is reachable from a variable source s): There exists a vertex s such that path space $P(X)_s^t$ is exactly $(\kappa(X)-2)$-connected, and for all other vertices s', $P(X)_{s'}^t$ is at least $(\kappa(X)-2)$-connected.

In particular, if this invariant $\kappa(X) \geq 2$, it is guaranteed that all relevant (non-empty) path spaces are path-connected. That means that all executions of the program X (same source, same target, same individual execution, in particular same number of loops traversed) yield the *same result* regardless the order of accesses to shared resources. Such a program can thus *not* be used to solve a decision problem (cf eg Herlihy etal [8]). On the other hand, if $\kappa(X) = 1$, then there exists a vertex such that directed paths starting from that vertex are not dihomotopic, and corresponding executions may lead to different results.

1.2.2. *Content of the paper in more detail.* In Section 2, we describe the point of departure: Given a number of processors, each following a linear program (linearity is not assumed throughout) and participating in a concurrent program with shared resources, each with a given capacity, which are potentially locked and relinquished sequentially by every processor. The geometric semantics corresponds to a state space in the form of a Euclidean cubical complex, a subcomplex of a cubical subdivision of $\mathbf{R}^n$; executions correspond to *directed* paths from a source vertex to a target vertex. The aim is to study the *space* of all such executions as a topological space; in particular to determine its connectivity. This section recapitulates in essence the point of departure in Fajstrup etal [6, ch. 3].

Section 3 focusses on the *local* behaviour of the state space. As already explained by Ziemiański [20] and Belton etal [1, 2], the key information is the topology (in particular, the connectivity) of the *future links* (or past links) of vertices in the state space. It turns out that these future links are *joins* (aka convex combinations) of *skeleta of simplices.* This observation lets us determine the connectivity of the future link of a vertex v in terms of a spare capacity $\kappa(X;V)$ *defined at that vertex*; this spare capacity can be calculated from the syntax of the individual programs by a combinatorial formula; cf Definition 3.7.

In Section 4, we define the spare capacity of a concurrent program (or its state space) as the *minimum* of the spare capacities of all its reachable vertices. Björner's version of the nerve theorem [3, Theorem 6] is then applied to conclude that the connectivity of the state space is bounded below by the connectivities of the future links of all inbetween vertices. The overall result holds also for unfoldings of individual programs modelled on a general digraph. Moreover, the section contains reflections on what happens to spare capacities (and thus connectivities) if processors are allowed to crash.

The final section is devoted to a sketch of algorithmic aspects. Particular care is devoted to deadlock detection – corresponding to spare capacity 0 at a vertex – extending the results of Fajstrup etal [7], and also to vertices with spare capacity 1 indicating potential "splits" of the space of executions into several path components. Throughout, simple examples and illustrations motivate the strategy.

2. Forbidden region and state space

2.1. **Resource consumption.** Let R denote the set of resources and $P = [1:n]$ the set of processors.

2.1.1. *One processor.* A *PV* command line for a single processor $j \in P$ can be encoded by a number of functions $Pr_j, Vr_j : [1:k_j(r)] \to [1:l(j)],\ r \in R$, such that

- $Pr_j(i) < Vr_j(i),\ i \leq k_j(r)$, and $Vr_j(i) < Pr_j(i+1),\ i < k_j(r)$, and
- $\bigcup_{r\in R, i\in[1:k_j(r)]}\{Pr_j(i), Vr_j(i)\} = [1:l(j)]$.

It is allowed that $k_j(r) = 0$, ie that some resources are not called upon by processor j.

For $r \in R$ and $j \in P$, let $cr_j : I_J := [0, l(j)+1] \to \{0,1\}$ denote the characteristic function of the subset $\bigcup_{i\in[1:k_j(r)]}]Pr_j(i), Vr_j(i)[\cup\{l(j)+1\}$ indicating whether j has a lock to r or not (or has arrived at a the final state). Together, they assemble to a binary valued *consumption vector function* $c_j : [0, l(j)] \to \{0,1\}^R$.

Furthermore, we let $dr_j : [0:l(j)] \to \{-1,0,1\}$ denote the difference of the characteristic function of the set of integers $\{Pr_j(i)|\ 1 \leq i \leq k_j(r)\}$ and of the characteristic function of $\{Vr_j(i)|\ 1 \leq i \leq k_j(r)\}$, ie $dr_j(k) = \pm 1$ if $k = Pr_j(i)$, resp. $k = Vr_j(i)$ for some $i \in [1:k_j(r)]$, and 0 else. Then $cr_j(i+t) = cr_j(i) + dr_j(i),\ i \in [1:l(j)-1], t \in]0,1[$.

2.1.2. *Several processors.* Information regarding consumption of resources by *all* processors $j \in P$ is encoded by functions on $\prod_{j\in P}[0, l(j)+1)] \subset \mathbf{R}^P$: In particular, the total consumption vector function $\mathbf{c} : \prod_{j\in P}[0, l(j) + 1)] \to (\mathbf{N}_{\geq 0})^R$ is defined by $\mathbf{c}(x_1, \dots, x_n) = \sum_{j\in P} c_j(x_j)$ measuring how many locks to resources $r \in R$ have been acquired at $(x_1, \dots, x_n)$. It has component functions $cr : \prod_{j\in P}[0, l(j) + 1)] \to (\mathbf{N}_{\geq 0})$ for every $r \in R$.

How does the consumption function change when proceeding from an integer vertex $v = (i_1, \dots, i_n) \in \prod_{j\in P}[0 : l(j)] \cap \mathbf{Z}^n$? The answer is given as

$$cr(i_1 + t_1, \dots, i_n + t_n) = cr(i_1, \dots, i_n) + \sum_{t_j>0} dr_j(i_j),\ 0 \leq t_j < 1.$$

This last sum encodes the number of Pr commands at the vertex $v = (i_1, \dots, i_n)$ on a given set of processors (those j with $t_j > 0$).

2.2. **Forbidden region. State space.**

2.2.1. *A single shared resource.* We start by considering the case of a concurrent program in which processors in a set $P := \{1, \dots, n\}$ compete for a *single* resource r with capacity $\kappa(r)$ called upon (often several times) by programs each of the form $(PrVr)^{k_j},\ 1 \leq j \leq n, k_j \geq 0$; the case $k_j = 0$ is allowed in order to take care of processors that do not call on r at all. Let $N(r) \subseteq [1 : n]$ denote the subset with the property: $j \in N(r) \Leftrightarrow k_j > 0 \Leftrightarrow p_j$ calls upon r at least once.

The corresponding *forbidden* region $F(r)$ is defined as

$$F(r) := \{x = (x_1, \dots, x_n) \in \prod_{j\in P}[0, l(j) + 1)]|\ cr(x) > \kappa := \kappa(r)\}.$$

It can be described as a union of subsets enumerated as follows: Consider any injection $i(r) : [1 : \kappa + 1] \hookrightarrow N(r) \subseteq [1 : n]$ and the dual projection $i(r)^* : \mathbf{R}^n \to \mathbf{R}^{\kappa+1}$. For every such injection consider all $(\kappa + 1)$-tuples $\mathbf{l} := (j_1, \dots, j_{\kappa+1})$ such that $0 < j_l \leq k_{i(r)}(l),\ 1 \leq l \leq \kappa+1$. For each combined choice let $F(i, \mathbf{l}) := (i^*)^{-1}(\prod_{l=1}^{\kappa+1}]Pr_j(i), Vr_j(i)[)$; a product that has $(n-\kappa-1)$ factors consisting of an entire interval $[0, l_j+1]$ corresponding to $j \notin i(r)([1 : \kappa+1])$. The entire forbidden region is then $F(r) := \bigcup_{(i,\mathbf{l})} F(i, \mathbf{l})$; cf Figure 1.

The *state space* $X(r)$ is the complement of the forbidden region:

$$X(r) := \prod_{j=1}^{n}[0, l(k_j) + 1] \setminus F(r).$$

We focus on properties of spaces $\vec{P}(X(r))_s^t$ of directed paths in $X(r)$ (cf Section 1.1) between two vertices (in X) with integer coordinates. It can be shown (cf Raussen-Ziemiański [17]) that this path space is homotopy equivalent to the path space in the κ-skeleton – consisting of all cubes of dimension at most κ – of the cubical grid decomposition of $\prod_1^n[0, l(k_j) + 1]$ with integer vertices.

The common boundary of $F(r)$ and of $X(r)$ consists of those $x \in \prod_{j=1}^n[0, l(k_j)+1]$ satisfying

- $cr(x) \leq \kappa(r)$
- $\exists\mathbf{t} = (t_1, \dots, t_n), -1 < t_j < 1, t_j = 0$ for $x_j \notin \mathbf{Z}$: $cr(x + \mathbf{t}) > \kappa(r)$.

In particular, a *vertex* $v = (i_1, \dots, i_n)$ with integer coordinates is contained in this boundary if and only if

- $cr(v) \leq \kappa(r)$

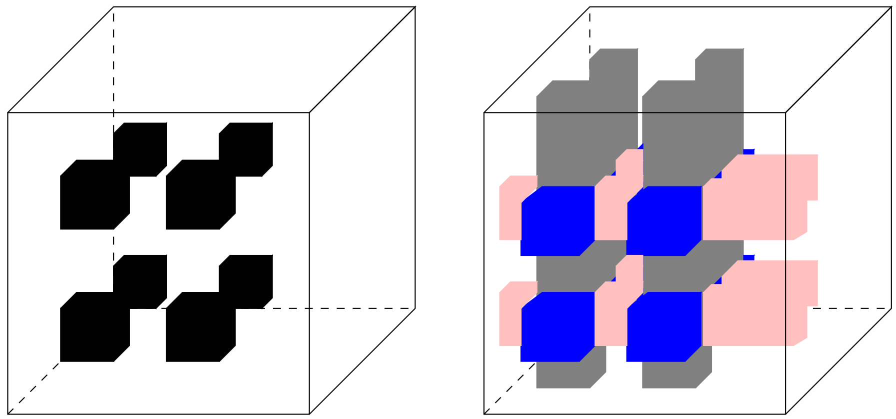

FIGURE 1. Forbidden region and state space (complement of the union of all boxes): One resource r of capacity 2 (left), resp. 1 (right); three processors each executing the program $PrVrPrVr$.

- $cr(v) + |\{j|\ i_j \in Pr_j([0 : k_j(r)]) \cup Vr_j([0 : k_j(r)])\}| > \kappa(r)$.

In the latter case, the point $v + (t_1, \dots, t_n)$ is contained in $F(r)$ if $0 < t_j < 1$ for $i_j \in Pr_j([0 : k_j(r)])$, $-1 < t_j < 0$ for $i_j \in Vr_j([0 : k_j(r)])$ and $t_j = 0$ else.

2.2.2. *Several resources.* The forbidden region $F(R)$ corresponding to a set R of resources is the *union* $F := \bigcup F(r)$ of the forbidden regions $F(r)$, $r \in R$. It agrees with $\{x = (x_1, \dots, x_n) \in \prod_{j=1}^n [0, l(k_j) + 1]|\ \exists r \in R : cr(x) > \kappa(r)\}$.
The state space is its complement: $X(R) := \prod_{j=1}^n [0, l(k_j) + 1] \setminus F = \bigcap_{r\in R} X(r)$.
It agrees with $\{x = (x_1, \dots, x_n) \in \prod_{j=1}^n [0, l(k_j) + 1]|\ \forall r \in R : cr(x) \leq \kappa(r)\}$.

A vertex $v \in X(R)_0$ with integer coordinates in the common boundary $\partial F = \partial X$ is characterized by the following properties:

(1) $cr(v) \leq \kappa(r)$ *for all* $r \in R$
(2) $\exists r \in R : cr(v) + |\{j|\ v_j \in Pr_j([0 : k_j(r)]) \cup Vr_j([0 : k_j(r)])\}| > \kappa(r)$.

3. Future links and their connectivity

Let $I^n = [0, 1]$ denote the standard n-cube. Any subcube (face of) I^n containing the minimal vertex $\mathbf{0}$ is characterized by the set of 1-coordinates of its *maximal* vertex. The poset of the subcubes properly containing $\mathbf{0}$ as minimal vertex is thus in an order-preserving correspondence with the non-empty subsets of $[1 : n]$ and forms an $(n-1)$ dimensional "future" simplex Δ_{n-1}. For a cubical *sub*complex $X \subseteq I^n$ containing $\mathbf{0}$, consider the *future link* $lk^+(X, \mathbf{0})$ in X consisting of those simplices corresponding to the subcubes *contained in* X. This definition extends easily to the *future link* $lk^+(X, v)$ of a vertex v (with interger coordinates; a state) in a Euclidean cubical complex $X \subset \mathbf{R}^n$. By definition, it is a simplicial complex embedded in the future simplex Δ_{n-1} of all cubes in $\mathbf{R}^n$ properly containing v as minimal vertex; cf Ziemiański [20, Def. 5.1] and Belton etal [1, Def. 4], [2, Sect. 2.2] who deal with analogously defined past links.

3.1. **A single resource.** Let $v = (v_1, \dots, v_n) \in X(r) \subseteq \prod_{j=1}^n [0, l(k_j) + 1]$ denote a vertex in the state space $X(r)$ corresponding to a single resource r; cf Section 2.2.1. Let $m(v)$ denote the number of coordinates that are *not* maximal (ie $v_j \neq l(k_j) + 1$). To the maximal cube

$M(v) \subset \prod_{j=1}^{n}[0, l(k_j)+1]$ with lower vertex v – of dimension $m(v)$ – corresponds a future simplex $\Delta^{m(v)-1}$; the future link $lk^+(X(r), v) \subseteq \Delta^{m(v)-1}$ encodes those faces of $M(v)$ that are contained in $X(r)$.

The capacity $\kappa(r)$ of resource r has the following consequence:

Lemma 3.1. *The future link $lk^+(X(r), v)$ is*

(1) *the $(\kappa(r)-1)$-skeleton $\Delta^{m(v)-1}_{(\kappa(r)-1)}$ of the future simplex $\Delta^{m(v)-1}$ if all coordinates v_j are of the form $Pr_j(i)$ or the final $l(k_j)+1$.*
(2) *contractible otherwise.*

Proof. (1) The maximal subcubes in $X(r)$ containing v as minimal vertex are in one-to-one correspondence with subsets of cardinality $\kappa(r)$ among the coordinates in v that are not final.

(2) Suppose v_j is neither of the form $Pr_j(i)$ nor the final $l(k_j)+1$. Consider the subsimplex $\Delta^{m(v)-2} \subset \Delta^{m(v)-1}$ consisting of all subsets *not* containing j. For every subcube $Q \subset X(r)$ with minimal vertex v and future simplex contained in Q, the product $Q \times I$ (in direction j) is also contained in $X(r)$. In other words, the future link $lk^+(X(r), v)$ is a *cone* with apex corresponding to j; and hence contractible.

□

Corollary 3.2. The future link $lk^+(X(r), v)$ is

(1) contractible if either v has at least one coordinate not of the form $Pr_j(i)$ or the final $l(k_j)+1$ or if $m(v) \leq \kappa(r)$
(2) $(\kappa(r)-2)$-connected but not $(\kappa(r)-1)$-connected if all coordinates v_j are of the form $Pr_j(i)$ or the final $l(k_j)+1$ and if $m(v) > c(r)$.

Remark 3.3. For a resource r with given capacity κ, information about the homology of the path space $\vec{P}(X(r))_{\mathbf{0}}^{\mathbf{2k+1}}$, $\mathbf{k} = (k_1, \dots, k_n)$, ie the Betti numbers, can be found in Meshulam-Raussen [10, Corollary 5.2].

3.2. **Several shared resources.** Let us now consider a program on the resource set R and a vertex $v = (v_1, \dots, v_n) \in X_0$ in the corresponding state space $X := X(R)$, cf Section 2.2.2. Every coordinate v_j is either 0 or maximal or in the range of one of the functions Pr, resp. Vr, $r \in R$. It is only relevant to investigate the future links of vertices v with *all* coordinates either maximal or in the range of a function Pr_j, $r \in R, j \in P$, since

Lemma 3.4. *Let one of the coordinates v_j be either 0 or of the form $Vr(i_j)$ for some resource $r \in R$. Then the future $lk^+(X, v)$ is contractible.*

Proof. As in the proof of Lemma 3.1, the future link is then a cone and hence contractible. □

We need some notation to explore future links of the remaining vertices. We let X_0^P denote the set of those vertices $v \in X_0$ with all coordinates maximal or in the range of one of the Pr-functions. For such a vertex v:

- $P(v) \subseteq P$ denotes the subset of *active* processors j such that $v_j \neq l(k_j)+1$ (not a final state); with cardinality $m(v) := |P(v)|$.
- $R(v) \subseteq R$ denotes the subset of *resources* such that there exists a $j \in [1:n]$ with v_j in the range of Pr (a lock to r is requested at v).

- For $r \in R(v)$, we let $Pr(v)$ denote $\{j \in P(v) | v_j \in Pr([1 : k_j(r)])\}$ the set of processors with a call to resource r at v, with cardinality $dr(v) := |Pr(v)|$, ie the number of calls to r issued at v. The sum $\sum_{r \in R(v)} dr(v)$ is equal to $m(v) = |P(v)|$ – if every processor calls exactly one resource at a "time".

With this in place, we can formulate the following crucial simple technical result:

Proposition 3.5. *For $v \in X_0^P$, the future link $lk^+(X, v)$ is homeomorphic to a join (consisting of all convex combinations; notation $*$) of skeleton spaces*

$$lk^+(X, v) \cong \mathop{\ast}_{r \in R(v)} \Delta^{dr(v)-1}_{(\kappa(r)-cr(v)-1)} = \mathop{\ast}_{r \in R(v),\, cr(v) < \kappa(r)} \Delta^{dr(v)-1}_{(\kappa(r)-cr(v)-1)}.$$

Here, join with an empty set – occuring on the left hand side if $cr(v) = \kappa(r)$ – has to be understood as $A * \emptyset = A$.

Proof. Decompose the maximal cube $M(v) \subset \prod_{j=1}^{n} [0, l(k_j) + 1]$ with minimal vertex $v \in X_0^P$ – of dimension $m(v)$ – as a product $\prod_{r \in R(v)} M(r)$ of cubes $M(r)$ in directions $Pr(v)$, each of dimension $dr(v), r \in R(v)$. A (future) subcube $C \subseteq M(v)$ decomposes correspondingly as $C = \prod_{r \in R(v)} C(r)$. For a future cube C in $X(R)$, the capacity constraint regarding $r \in R$ corresponds to $\dim C(r) \leq \kappa(r) - cr(v)$: since $cr(v)$ locks are already active at v, only up to $\kappa(r) - cr(v)$ locks can be acquired at v.

Products of cubes correspond to *unions* of sets of indices, and these correspond to *joins* of the associated future links of v with respect to each of the $C(r)$, $r \in R(v)$. Apply Lemma 3.1. □

Remark 3.6. Proposition 3.5 is only true – on the nose – if one assumes that every processor makes call to resources subsequently, not at the same "time". If this is not the case, one may split up two concurrent calls without changing the homotopy type of the path space, as observed by Ziemiański [20, Sect. 3].

For every vertex $v \in X_0^P$, let $\chi_v : R \to \{0, 1\}$ denote the characteristic function of $R(v) \subset R$.

Definition 3.7. The *spare capacity* $\kappa(X; v)$ of the state space X at the (allowed) vertex $v \in X_0^P$ is defined as

$$\kappa(X; v) := \begin{cases} \infty & dr(v) \leq \kappa(r) - cr(v) > 0 \text{ for at least one } r \in R(v) \\ \sum_{r \in R} \chi_v(r)(\kappa(r) - cr(v)) & \text{else} \end{cases}.$$

Remark 3.8.
1. We remarked in Section 2.2.2 that if a vertex $v \in X_0^P$ is reachable, then $cr(v) \leq \kappa(r)$ for each $r \in R(v)$. Hence $\kappa(X; v) \geq 0$ for every $v \in X_0^P$.
2. For a (critical) vertex $v \in X_0^P$ in the *boundary* of the state space X, it follows from Section 2.2.2 that $\kappa(r) - cr(v) \geq 0$ for all $r \in R$. Moreover, $dr(v) > \kappa(r) - cr(v)$ for all $r \in R(v)$. In particular, $\kappa(X; v)$ is finite for such a vertex.
3. For $v \in X_0^P$, spare capacity $\kappa(X; v)$ is finite if and only if for every resource $r \in R(v)$ either $cr(v) = \kappa(r)$ or $dr(v) > \kappa(r) - cr(v)$. Moreover, $cr(v) \leq \kappa(r)$ for all $r \in R$, and v is contained in the *boundary* of X.
4. For $\kappa(X; v) < \infty$, the definition of spare capacity is equivalent to $\kappa(X; v) = \sum_{r \in R(v)} \kappa(r) - \sum_{r \in R(v)} cr(v)$.

Corollary 3.9. Let $v \in X_0^P$ denote a vertex.

1. If $\kappa(X; v) = \infty$, then $lk^+(X, v)$ is contractible.

(2) v is a deadlock vertex if and only if $\kappa(X;v)=0$.
(3) If $\kappa(X;v)<\infty$, then $\kappa(X;v)\le n-|R(v)|$.
(4) The future link $lk^+(X,v)$ is disconnected if and only if $\kappa(X;v)=1$.
(5) The future link $lk^+(X,v)$ is path-connected but not simply connected if and only if $\kappa(X;v)=2$.
(6) If $2<\kappa(X;v)<\infty$, then $lk^+(X,v)$ is $(\kappa(X;v)-2)$-connected but not $(\kappa(X;v)-1)$-connected.

Proof. (1) If $dr(v)\le\kappa(r)-cr(v)$ for $r\in R(v)$, then the skeleton corresponding to r in the join decomposition from Proposition 3.5 is the *entire* simplex $\Delta^{dr(v)-1}$ (non-empty since $dr(v)>0$) and hence contractible: All processors locking r at v can proceed independently of each other. Moreover, a join with a contractible space (in Proposition 3.5) is contractible.

(2) Spare capacity $\kappa(X;v)=0$ for $v\in X_0$ if and only if $cr(v)=\kappa(r)$ for all $r\in R(v)$. Hence, none of the processors can acquire an additional lock at v; or more technically, all skeleta $\Delta^{dr(v)-1}_{(\kappa(r)-cr(v)-1)}$, $r\in R(v)$, are empty. If $\kappa(X;v)>0$, then $cr(v)<\kappa(r)$ for some $r\in R(v)$. At least one processor can proceed from v; at least one of the skeleta is non-empty.

(3) For each $r\in R(v)$ we have that $cr(v)-\kappa(r)<dr(v)$ – even if the left hand side is 0. Since we are dealing with integers, this means that $cr(v)-\kappa(r)\le dr(v)-1$. Summing up over all $r\in R(v)$, we get the spare capacity on the left and $n-|R(v)|$ on the right.

(4) The spare capacity $\kappa(X;v)=1$ if and only if $\kappa(r)=cr(v)$ for all $r\in R(v)$ apart from a single $r_0\in R(v)$ with $\kappa(r_0)=cr_0(v)+1$. In that case $lk^+(X,v)$ is the 0-skeleton of $\Delta^{dr_0(v)-1}$ which is not path-connected since $dr_0(v)>\kappa(r_0)-cr_0(v)=1$.

(5) The spare capacity $\kappa(X;v)=2$ if and only if either there is a single resource $r_0\in R(v)$ with $\kappa(r_0)=cr_0(v)+2$ and $\kappa(r)=cr(v)$ for all others or if there are two resources $r_1,r_2\in R(v)$ with $\kappa(r_i)=cr_i(v)+1$ and $\kappa(r)=cr(v)$ for all others. In the first case, $lk^+(X,v)$ is the 1-skeleton of a simplex $\Delta^{dr_0(v)-1}$, which is path-connected, but not simply-connected, since $dr_0(v)>\kappa(r_0)-cr_0(v)=2$. In the second case $lk^+(X,v)$ is the join of two discrete spaces which is path-connected. It cannot be simply-connected: Neither of them is a single point since $dr_i(v)>\kappa(r_i)-cr_i(v)=1$.

(6) If $\kappa(X;v)\ge 3$, then $lk^+(X,v)$ is either the $(\kappa(X;v)-1)$-skeleton of a non-empty simplex or the join of a path-connected space with another space, and therefore simply-connected; connectivity can thus be read off from homology. The Mayer-Vietoris sequence in homology (cf. [9, ch. 2.2] applied to a join $A*B=CA\times B\cup_{A\times B}A\times CB$ shows: If A k-connected and B is l-connected, then $A*B$ is $(k+l+1)$-connected. Inductively, this implies that the join in Proposition 3.5 is trivial in dimensions up to $\kappa(X;v)-2$ and non-trivial in dimension $\kappa(X;v)-1$. The Betti number in that dimension is the sum of the topdimensional Betti numbers of the skeleta of the simplices involved.

□

Remark 3.10. Let us stress: A vertex $v\in X_0^P$ has a future link $lk^+(X;v)$ that is *not path-connected* if and only if $\kappa(X;v)=1$ if and only if $\kappa(r)=cr(v)$ for all $r\in R(v)$ apart from a single resource $r_0\in R(v)$ with $dr_0(v)>\kappa(r_0)-cr_0(v)=1$.
We will call such a vertex v with $\kappa(X;v)=1$ a *critical* vertex (or state)

Example 3.11. (1) Consider two resources r^1, r^2 of capacity three each, and four processors $p_i,\ 1 \leq i \leq 4$.

A: Processors $p_i, i \leq 3$, start with Pr^1Pr^2 and p_4 starts with Pr^2Pr^1. At the vertex $v = (2,2,2,2) \in X_0^P$ with the final lock requests, p_4 cannot proceed whereas two of the processors $p_i,\ i \leq 3$, can proceed concurrently. Hence $lk^+(X;v)$ is the 1-skeleton of a 2-simplex homeomorphic to S^1 (path-connected, but not simply-connected).

B: p_1, p_2 start with Pr^1Pr^2 and p_3, p_4 starts with Pr^2Pr^1. At the vertex $v = (2,2,2,2)$, one out of p_1, p_2 and one out of p_3, p_4 – but not both – can proceed. Hence $lk^+(X;v)$ is the join of two spaces consisting of two points each; the resulting graph is homotopy equivalent to S^1.

In both cases, $\kappa(X;v) = 2$.

(2) Now add a fifth processor p_0 starting with Pr^1Pr^2.

A: The vertex $w = (2,2,2,2,2)$ is no longer reachable: $cr^1(w) = 4 > 3 = \kappa(r^1)$.

B: At w, only one of processors p_0, p_1, p_2 can proceed; p_3 and p_4 have to wait. Hence $lk^+(X,w)$ is a three-point space and not path-connected. This agrees with our observation from Remark 3.10: $\kappa(X;w) = 1$.

Remark 3.12. Is it necessary to analyze the local future of (points on) faces of the state spaes, as well? This is not the case as long as we only consider spaces of d-paths whose target is a *vertex* t. It is shown in Fajstrup [5] that $\vec{P}(X)_v^t$ and $\vec{P}(X)_{v_0}^t$ are homotopy equivalent if v is a point on a face with top vertex v_0.

4. Global connectivity of spaces of d-paths

We fix a PV-program with state space X and target vertex t. All future links have to be understood with respect to that target t. Our aim is to establish connectivity bounds for spaces $\vec{P}(X)_v^t$ of directed paths within X starting at v and ending at t, endowed with the compact-open (aka uniform convergence) topology; in particular, to find out whether such spaces are path-connected, via directed homotopies, ie 1-parameter families of directed paths; cf Fajstrup etal [6, ch. 4.2].

Definition 4.1. The *spare capacity* $\kappa(X)$ of a concurrent PV program with state space X is defined as $\kappa(X) := \min_{v \in X_0^P, v \leq t} \kappa(X;v)$, ie the *minimum* of all spare capacities $\kappa(X;v)$ of (allowed) vertices $v \in X_0$ from which t is reachable.

Remark 4.2. It is possible to remove the doomed region (cf [6, ch. 5]) the set of all elements that cannot be connected to t by a d-path) from the state space in a first algorithmic step. Only the spare capacities of vertices in the new smaller state space have to be estimated; cf Section 5.3 for details.

4.1. **The spare capacity as a connectivity indicator.**

Proposition 4.3. *Given the state space X of a concurrent (linear) PV-program with final state t and a vertex $v_0 \in X_0^P$ from which t is reachable. Then the path space $\vec{P}(X)_{v_0}^t$ is $(\kappa(X) - 2)$-connected.*

Proof. The proof makes use of the future link $lk^+(X,v)$ of a vertex $v_0 \in X_0$ viewed as a *poset category*, with inclusion (of sets; or simplices) as partial order. It proceeds by induction on the L_1 (aka taxicab) distance between vertices v_0 and t in the cubical grid; cf Raussen [13, Sect. 2.2] in this context. It starts with distance 0, ie $v_0 = t$. Then the path space consists of the constant path only, and it is hence contractible.

Assume by induction that $\vec{P}(X)_v^t$ is $(\kappa(X)-2)$-connected for every vertex v with $v_0 < v \le t$, ie $v_0 \ne v$ and there exist d-paths from v_0 to v and from v to t. In Raussen-Ziemiański [17, Sect. 2.3], subsequently exploited in Ziemiański [20] and Belton etal [1, 2][1], it is shown that $\vec{P}(X)_{v_0}^t$ is the colimit (the union) of certain subspaces $F_v\vec{P}(X)_{v_0}^t, v \in lk^+(X, v_0)$ with $F_v\vec{P}(X)_{v_0}^t$ homotopy equivalent to $\vec{P}(X)_v^t$; in particular, $F_v\vec{P}(X)_{v_0}^t$ is $(\kappa(X) - 2)$-connected for each vertex v in the future link $lk^+(X, v_0)$.

The geometric realization of the future link category (aka its nerve) is the future link space $lk^+(X, v_0)$ that is (at least) $(\kappa(X) - 2)$-connected by Corollary 3.9. We apply Björner's theorem [3, Theorem 6] on a colimit of spaces whose connectivity is limited below in a certain pattern. These connectivity conditions are certainly (more than) met, and hence the colimit of the spaces $F_v\vec{P}(X)_{v_0}^t$, and therefore the path space $\vec{P}(X)_{v_0}^t$, has homotopy groups isomorphic to those of the nerve of the (future link) category up to dimension $\kappa(X) - 2$. In particular, the path space $\vec{P}(X)_{v_0}^t$ is also $(\kappa(X) - 2)$-connected.

Björner's theorem is formulated for simplicial complexes and their subcomplexes. All occuring path spaces have the homotopy type of CW-complexes, cf Raussen [13, Prop. 3.15], and hence of simplicial complexes, cf Hatcher [9, Theorem 2C.5]. □

As a special case, we obtain

Corollary 4.4. Let X denote a Euclidean cubical complex with spare capacity $\kappa(X) \ge 2$ and vertices $v_0, t \in X_0^P$. Then $\vec{P}(X)_{v_0}^t$ is either empty or path-connected.

Remark 4.5.
(1) Corollary 4.4 has the following interpretation, under the stated assumptions: Every concurrent execution of individual programs on processors $j \in P$ starting at v_0 and ending at t yields the same result, regardless of the order of access to shared resources.
(2) The condition $\kappa(X) \ge 2$ in Corollary 4.4 is satisfied if and only if for *each* vertex $v \in X_0^P$ one of the following conditions is met:
- $\exists r \in R(v)$ with $dr(v) \le \kappa(r) - cr(v)$ (v is not contained in the boundary of X_0^P);
- $\exists r_1, r_2 \in R(v)$ with $\kappa(r_i) \ge c(r_i)+1$ (two resources with non-exhausted capacity);
- $\exists r \in R(v)$ with $\kappa(r) \ge c(r) + 2$ (a resource that can admit two further locks).

Proposition 4.3 and Corollary 4.4 are strict in the following sense:

Proposition 4.6. *Given the state space X of a PV-program with final state t and a vertex $v_0 \in X_0^P$ from which t is reachable. Assume that $\kappa(X; v_0) \le \kappa(X; v)$ for all vertices $v_0 \le v \le t$. Then the path space $\vec{P}(X)_{v_0}^t$ is $(\kappa(X; v_0) - 2)$-connected but not $(\kappa(X; v_0) - 1)$-connected.*

Proof. As in the proof of Proposition 4.3, we may assume inductively that all path spaces $\vec{P}(X)_v^t$ from vertices $v \in lk^+(v_0)$ are at least $(\kappa(X; v_0) - 2)$-connected. Hence, using [3, Theorem 6], $\vec{P}(X)_{v_0}^t$ has the same *non-trivial* homotopy in dimension $\kappa(X; v_0)-1$ as $lk^+(X; v_0)$. □

Corollary 4.7. Given the state space X of a concurrent program and a vertex $v_0 \in X$ with *minimal* spare capacity, ie $\kappa(X) = \kappa(X; v_0)$. Then the path space $\vec{P}(X)_{v_0}^t$ is $(\kappa(X) - 2)$-connected but not $(\kappa(X) - 1)$-connected.

[1] formulated for past links

Corollary 4.8. Given the state space X of a concurrent program and a vertex $v_0 \in X_0^P$ with spare capacity $\kappa(X; v_0) = 1$. Then the path space $\vec{P}(X)_{v_0}^t$ is not path-connected, ie potentially giving rise to *different* results depending on access to shared resources.

See Figure 2 for an illustration. We have more to say on vertices $v \in X_0^P$ "below v_0" with non path-connected path spaces $\vec{P}(X)_v^t$ in Section 5.4.2.

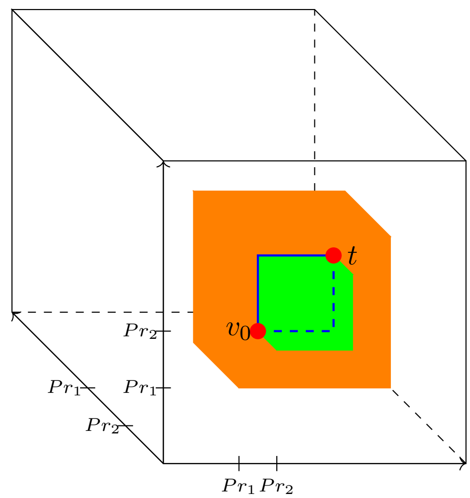

FIGURE 2. A disconnected path space arises for $n = 3, R = \{r_1, r_2\}, \kappa(r_i) = 2$, and PV-programs p_1, p_3 starting both by Pr_1Pr_2 and p_2 by Pr_2Pr_1. The forbidden regions are indicated as $F(r_1)$ in orange, resp. $F(r_2)$ in green. For the indicated vertex $v_0 = (2, 2, 2)$, we have $dr_2(v_0) = 2, cr_2(v_0) = 1$, and $cr_1(v_0) = 2$, and hence $\kappa(X; v_0) = 1$. Path space $\vec{P}(X)_{v_0}^{v_1}$ is in fact disconnected for every vertex $t \leq v_1$.

Example 4.9. Consider $l < n$ positive integers $\kappa_j < n, 1 \leq j \leq l$, such that $\sum_1^l \kappa_j > (l-1)n$. We construct a PV program on n threads and l resources r_j with capacities $\kappa(r_j) = \kappa_j$ and corresponding state space X and two vertices $v_0, v_1 \in X$ such that $\vec{P}(X)_{v_0}^{v_1}$ has first non-trivial homology in dimension $\kappa(X) - 1 = \sum_1^l \kappa_j - (l-1)n - 1$. In particular, $\vec{P}(X)_{v_0}^{v_1}$ is not path-connected for $\sum_1^l \kappa_j = (l-1)n + 1$:

Define $\bar{c}_k = \sum_1^k \kappa_j - (k-1)n, \bar{d}_k = n - \bar{c}_k = kn - \sum_1^k \kappa_j$. Remark that

$$\bar{c}_k + \bar{d}_{k-1} = \kappa_k. \tag{4.10}$$

Consider a PV program where all n threads are of the form $P_{m_1} \dots P_{m_l} V_{m_l} \dots V_{m_1}$ with $(m_1, \dots, m_l)$ a permutation in Σ_l, and P_i is short for Pr_i. More specifically, they are chosen according to the following pattern (column j corresponds to the j-th P command): $P_1, \dots, P_{l-1}$ are successively filled into the columns until the capacity κ_j is exhausted; the remaining slots in column $l-1$ are occupied by P_l. The slots in the last column l are occupied by those P commands that do not occur in the particular thread in previous columns, cf Figure 3.

Let $v_0 = (l-1, \dots, l-1)$ denote the vertex corresponding to the P commands after column $l-1$. Then $cr_i(v_0) = \kappa(r_i), i < l$: these capacities are exhausted. Moreover, $cr_l(v_0) = \bar{d}_{l-1}$,

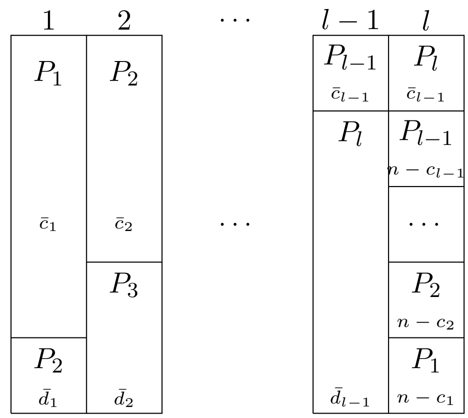

FIGURE 3. Start of a PV program

and hence $\kappa(X; v_0) = \kappa_l - \bar{d}_{l-1} = \bar{c}_l = \sum_1^l \kappa_l - (l-1)n$ by (4.10). The result follows from Proposition 4.6 and Corollary 4.7 since future links of all vertices above v_0 until the final vertex $v_1 = (2l+1, \ldots, 2l+1)$ are contractible, cf Lemma 3.4.

4.2. **Spare capacity in the presence of crashes.** What happens to the spare capacity if one (or several) of the processors crashes during a computation? For simplicity of notation, we assume that processor n crashes. First of all, state space is reduced: The interval $[0, l(k_n)+1]$ is reduced to $[0, C]$ with $0 < C < l(k_n)+1$, and $X_C = \pi_n^{-1}([0, C])$. This makes all vertices v with $v_n > C$ irrelevant. On the other hand, new vertices on the upper boundary $x_j = C$ appear. We let B denote the largest integer strictly smaller than C and such that $B \in Pr_n([1 : k_n])$ for some $r \in R$ – ie the command issued at $x_n = B$ is Pr. We compare the connectivities of future links at vertices $v_C = (v_1, \ldots v_{n-1}, C) \in (X_C)_0^P$ and $v_B = (v_1, \ldots, v_{n-1}, B) \in X_0^P$.

Lemma 4.11. *If $lk^+(X, v_B)$ is k-connected, then $lk^+(X_C, v_C)$ is at least $(k-1)$-connected. It is not k-connected if and only if $lk^+(X, v_B)$ is not $(k+1)$-connected and if the crash happens before r was relinquished again.*

Proof. Comparing the resource calls and resource consumptions, we have: $dr(v_C) = dr(v_B) - 1$ and $dr'(v_C) = dr'(v_B)$ for $r \neq r'$ and $cr(v_B) \leq cr(v_C) \leq cr(v_B) + 1$ (depending on whether r was relinquished between B and C) and $cr'(v_C) = cr'(v_B),\ r \neq r'$. Hence $\kappa(r) - cr(v_B) \geq \kappa(r) - cr(v_C) \geq \kappa(r) - cr(v_B) - 1$ whereas for $r' \neq r$, $\kappa(r') - cr'(v_C) = \kappa(r') - cr'(v_B)$. Apply Definition 3.7. The spare capacity decreases by one if and only if $cr(v_C) = cr(v_B) + 1$. □

This result allows us to compare the spare capacity $\kappa(X)$ of a concurrent program without crash with the spare capacity $\kappa(X_C)$ of that program with a crash at $x_n = C$ (used as estimate for connectivities of path spaces, cf Section 4.1):

Proposition 4.12. *The spare capacity of X_C is at least $\kappa(X_C) \geq \kappa(X) - 1$.*
It is less than $\kappa(X)$ if and only if the last command issued at $x_n = B$ before the crash is of type Pr for some $r \in R$ and *if there exists a vertex $v_B = (v_1, \ldots, v_{n-1}, B) \in X_0^P$ (with last coordinate B) of minimal spare capacity, ie $\kappa(X) = \kappa(X; v_B)$.*
The spare capacity of X_C is larger than $\kappa(X_C) > \kappa(X)$ if and only if the minimal spare capacity $\kappa(X; v)$ occurs only at vertices v after *the crash (ie $v_n > C$).*

Proof. For the last claim, remark that every vertex $v \in X_0^P$ with last coordinate less than C is contained in $(X_C)_0^P$, as well. □

Example 4.13. Consider the following simple example resulting in a drop of spare capacity and hence connectivity after a crash: Three processors compete for a single use of one resource r of capacity $\kappa(r) = 2$ resulting in spare capacity $\kappa(X) = \kappa(X; v_B) = 2 - 0 = 2$. If one of the processors crashes while having acquired a lock, the remaining two processors compete at v_C with $\kappa(X_C) = \kappa(X_C; v_C) = 2 - 1 = 1$, cf Figure 4.

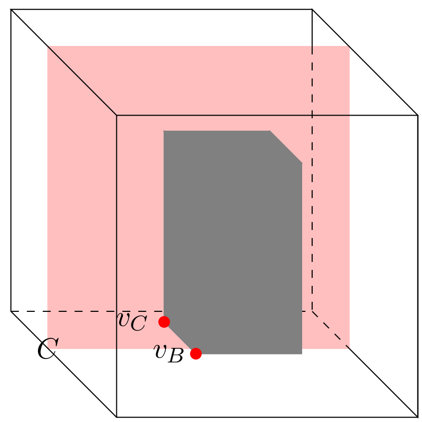

FIGURE 4. spare capacities – with X_C in front ot the crash wall (in pink): $\kappa(X; v_B) = 2, \kappa(X_C; v_C) = 1$

4.3. **Programs with branches and loops.** Allowing branches and loops changes the picture, but not dramatically. Every single processor p_i proceeds then along (the geometric realization of) a *directed graph* G_i instead of along an interval. A concurrent execution corresponds to a directed path in the product $\prod_1^n G_i$ of graphs from which certain forbidden regions have to be removed.

The space of directed paths in any of the graphs G_i from source to target is homotopy discrete: Every connected component of the path space can be represented by a constant speed path p_i with a directed interval within G_i as range; other directed paths in this component are reparametrizations of the given one. The space of directed paths in $\prod_1^n G_i$ (without considering conflicting resource consumption) is a disjoint union of the spaces of directed paths corresponding to n-tuples of such components, represented by n-tuples of paths $(p_1, \dots, p_n)$ – an "unfolding". The space of directed paths corresponding to a particular unfolding can then be described via directed paths in a *product of intervals*, and analyzed as in the previous sections. We are led to capacities and connectivity bounds that will often differ among the unfoldings, ie among the homotopy classes of directed paths in $\prod G_i$.

5. Algorithmics

5.1. **Notation.** The start data consist of a PV program on n threads and l resources $r^i \in R$. For a non-empty subset $R' \subseteq R$, let

- $P^j_{R'}(X) := \{l \in [1 : l(j)] |\ \exists r \in R', i \in [1 : k_j(r)] |\ l = Pr_j(i)\} \cup \{l(j)\} \subset [1 : l(j)]$ denote the subset of P-calls from $j \in P$ to a resource $r \in R'$ including the final position; with predecessor function $p^j_{R'} : P^j_{R'} \to P^j_{R'} \cup \{0\}$, $p^j_{R'}(v) = \max\{l \in P^j_{R'} \cup \{0\} |\ l < v\}$;

- $V^j_{R'}(X) := \{l \in [1 : l(j)] | \exists r \in R', i \in [1 : k_j(r)] | l = Vr_j(i)\} \subset [1 : l(j)]$ denote the subset of V-calls from $j \in P$ to a resource $r \in R'$; with successor function $s^j_{R'} : P^j_{R'} \to V^j_{R'},\ s^j_{R'}(v) = \min\{l \in V^j_{R'} | v < l\}$.

We introduce the following integer (sub-)lattices in $\mathbf{R}^P = \mathbf{R}^n$:

- $L(X) := \prod_{j\in P}[0 : l(j) + 1]$;
- $P_{R'}(X) := \prod_{j\in P} P^j_{R'}(X) \subset \bar{P}_{R'}(X) := \prod_{j\in P} P^j_{R'}(X) \cup \{0\} \subset L(X)$; with predecessor function $p_{R'} : P_{R'}(X) \to \bar{P}_{R'}(X),\ p_{R'}([v_1, \dots, v_n]) = [p^1_{R'}(v_1), \dots, p^n_{R'}(v_n)]$;
- $V_{R'}(X) := \prod_{j\in P} V^j_{R'}(X) \subset L(X)$; with successor function $s_{R'} : P_{R'}(X) \to V_{R'}(X)$, $s_{R'}([v_1, \dots, v_n]) = [s^1_{R'}(v_1), \dots, s^n_{R'}(v_n)]$;

Capacities $\kappa(r^i)$ of individual resources $r^i \in R$ are collected in an $l = |R|$-dimensional *capacity vector* $\boldsymbol{\kappa} = [\kappa r^1, \dots, \kappa r^l] \in \mathbf{N}^l$. To a grid vector $v \in P(X) = P_R(X)$, associate resource *consumption vectors*

- $\mathbf{c}(v) = [cr^1(v), \dots, cr^l(v)] \in \mathbf{N}^l_{\geq 0}$ (resource consumption "at/just before" v)
- $\mathbf{d}(v) = [dr^1(v), \dots, dr^l(v)] \in \mathbf{N}^l_{\geq 0}$ ("new" locks asked for at v; all $dr^i(v) \geq 0$; often $\sum_1^l dr^i(v) = n$).
- $\mathbf{l}(v) = [l^1(v), \dots, l^l(v)] = \mathbf{c}(v) + \mathbf{d}(v)$ (resource consumption "right after" v)

For a non-empty subset $R' \subset R$, the projection $\mathbf{Z}^R \to \mathbf{Z}^{R'}$ map $\boldsymbol{\kappa}, \mathbf{c}(v), \mathbf{d}(v)$ to $\boldsymbol{\kappa}_{R'}, \mathbf{c}_{R'}(v), \mathbf{d}_{R'}(v)$. This is particularly relevant for the subset $R' = R(v) := \{r \in R | dr(v) > 0\} \subseteq R$, the set of resources actually called for at a vertex v.

5.2. **Determining spare capacities algorithmically.** In this section, we will only need an implementation of the grid/array $P(X) := P_R(X) \subset L(X)$. We assume throughout that vectors, including vector addition and dot product, are implemented on that entire array.

5.2.1. *Calculations required for a single processor.* To initialize, consider a single processor $j \in P$ at a time: For every processor $r \in R$, we defined in Section 2.1 the call function $dr_j : [0 : l(j) + 1] \to \{0, 1, -1\}$. Taken over all resources $r \in R$, they define a call vector function $\mathbf{d}_j : [0 : l(j)] + 1 \to \{0, 1, -1\}^R$. It is directly read off from the command line for processor j in $l(j)$ steps resulting in $|R|$-dimensional vectors. If only one call is issued at every node, then $\mathbf{d}_j$ takes signed unit vectors as values; apart from at the ends 0 and $l(j) + 1$ with fixed value $\mathbf{0}$. Moreover, we determine the vector function $\bar{\mathbf{d}}_j : [0 : l(j) + 1] \to \{0, 1\}^R$ with $\bar{d}r_j(l) = 0$ if and only if $dr_j(l) = 1$ in another $l(j)$ steps.

Resource consumption $cr_j : [0 : l(j)] \to \{0, 1\}$ can be calculated inductively: We set $cr_j(0) = 0$ and update by $cr_j(l) = 0$ if $dr_j(l) = 1$, and $cr_j(l) = cr_j(l-1) + dr_j(l-1) + dr_j(l)$ otherwise; which is equivalent to $cr_j(l) = cr_j(l-1) + \bar{d}r_j(l)(dr_j(l-1) + dr_j(l))$: Check case by case'! Again, taken together over all resources $r \in R$, they calculate the resource consumption function $\mathbf{c}_j : [0 : l(j)] \to \{0, 1\}^R$.

Its determination affords another $2l(j)$ (vector addition) steps. Establishing the resource consumption function $\mathbf{c}_j$ for each processor $j \in P$, requires in total $4\sum_1^n l(j)$ steps.

5.2.2. *Array calculations: spare capacities of vertices.* In the next steps, calculate the vector function $\mathbf{d} : P(X) \to \{0, 1\}^R$ – with information on "new calls" – by $\mathbf{d}(v) = \sum_{j\in P} \mathbf{d}_j(v_j)$. Likewise the vector function $\mathbf{c} : P(X) \to (\mathbf{N}_{\geq 0})^R$ – encapsulating information of previous resource consumption – by $\mathbf{c}(v) = \sum_{j\in P} \mathbf{c}_j(v_j)$. Each of these calculations requires $n + 1 =$

$|P|+1$ steps: After initializing the entire array with 0-vectors, the *same* vector functions $\mathbf{d}_j$, resp. $\mathbf{c}_j$, $j \in P$, are added to all cells (in different "directions" $j \in P$, of course).

Next, calculate the difference $\boldsymbol{\kappa} - \mathbf{c} : P(X) \to \mathbf{Z}^R$ of the fixed capacity vector $\boldsymbol{\kappa} \in \mathbf{N}^R$ and the capacity function $\mathbf{c}$. Every vertex v for which $\boldsymbol{\kappa} - \mathbf{c}(v)$ has at least one negative component ($|R|$ comparison steps) belongs to the forbidden region and is flagged.

Finally, spare capacities for (non-flagged) vertices v are determined by one dot product operation per cell: $\boldsymbol{\kappa}(X)(v) = \mathbf{d}(v) \cdot (\boldsymbol{\kappa} - \mathbf{c}(v)) \in \mathbf{Z}$ on the entire array. Under our assumptions, the total number of steps is thus linear in the number of PV-steps on each of the processors in P and on the number of resources in R.

5.2.3. *Determining the spare capacity of a complex.* In a final round, probe successively equations $\boldsymbol{\kappa}(X)(v) = k$ starting with and increasing from $k = 0$ on the (non-flagged) vertices of the entire array. The minimal k for which an equation $\boldsymbol{\kappa}(X)(v)(v) = k$ yields the answer `true` for some vertex $v \in P(X)$ corresponds to the spare capacity $\kappa(X)$ of the state space. The number of steps needed is at most $|P| - 1$ for a (non-void) PV program on n processors.

Well, this answer is not always true; it has to be modified if deadlocks (with spare capacity 0) arise. In that case, one has to flag the entire "doomed regions" (no d-path from there to the top vertex) associated with them in a first step. This is explained in detail in Section 5.3.2.

5.3. **Deadlocks and doomed regions.** In the remaining sections, particular consideration is given to vertices v with spare capacity $\kappa(X; v) = 0$ (ie deadlocks) and those with $\kappa(X; v) = 1$ (ie with disconnected future link $lk^+(X, v)$).

5.3.1. *Deadlock detection.* Deadlock detection in Fajstrup etal. [7] was only provided in detail if all participating resources $r \in R$ have capacity $\kappa(r) = n-1$. In a way, the general case, with other and variable capacities, was still included, since the forbidden region $F(r)$ associated to a resource r of smaller capacity can be modelled as the union of many resources of capacity $n-1$. But it is preferable to give a formulation for deadlocks in the general case, with resources of various capacities participating: In view of Corollay 3.9(1), a vertex $v \in P(X)$ is a deadlock, if

- $\mathbf{c}(v) - \boldsymbol{\kappa} \geq \mathbf{0}$ (ie v is not contained in the forbidden region)
- $\mathbf{c}_{R(v)}(v) = \boldsymbol{\kappa}_{R(v)}$ (All resources asked for at v have been locked already before up to full capacity; none of the processors can advance).

A deadlock at v – detected as in Section 5.2.2 – comes thus with the following data:

- A subset $R' \subseteq R$ of resources (ie $R' = R(v)$);
- For each $r \in R'$, a subset $C(r) \subseteq P$ of locking processors of *cardinality* $|C(r)| = \kappa(r)$ and a non-empty subset $D(r) \subseteq P \setminus C(r)$ of its complement such that $\bigsqcup_{r \in R'} D(r) = P = [1:n]$ is a partition of P (ie the $D(r)$ are disjoint);
- Every processor $j \in C(r)$, $r \in R'$ has delivered a call Pr in front of and still active at v_j (ie $r \in R', j \in C(r) \Rightarrow cr_j(v_j) = 1$, hence $cr(v) \geq \kappa(r)$);
- For $j \in D(r)$, a call Pr is issued at v_j (ie $dr_j(v) = 1$);
- No other call $Pr, r \in R'$, is (still) active at v ($\Rightarrow cr(v) = \kappa(r)$);
- For all $r' \in R \setminus R'$, at most $\kappa(r')$ calls Pr' are active at v (ie $cr'(v) \leq \kappa(r')$).

The last two requirements make sure that v is not a forbidden vertex.

5.3.2. *Doomed regions.* Compare with Fajstrup etal [7] (where these regions are called "unsafe") and Fajstrup etal [6].

Let v denote a deadlock vertex with resource calls to $R(v)\subseteq R$ and predecessor vertex $w:=p_{R(v)}(v)$, cf Section 5.1. Remark that $\mathbf{l}_{R(v)}(w)=\mathbf{c}_{R(v)}(w)+\mathbf{d}_{R(v)}(w)=\mathbf{c}_{R(v)}(v)=\boldsymbol{\kappa}_{R(v)}$. Hence, every resource $r\in R(v)$ is locked by $\kappa(r)$ processors within the hyperrectangle $D(v):=]w,v]=\prod_{j=1}^{n}]v_j,w_j]$ spanned by w and v, and no directed path can leave $D(v)$.

One may eliminate this "primary" *doomed region* $D(v)$ from the state space X by a modification of the original PV-program: Add an extra resource $\bar{r}$ of capacity $n-1$ and, for each $j\in P$, calls $P\bar{r}$ at predecessors $w_j=p^j_{R(v)}(v_j)$ to be relinquished by $V\bar{r}$ at the successor $s^j_{R(v)}(v_j)$. Then $D(v)=F(\bar{r}):=]w,x],\ x=s_{R(v)}(v)$, becomes part of the forbidden region of the modified program – but path spaces with target not included in $D(v)$ remain unchanged!

Including $F(\bar{r})=]w,x]$ as an additional forbidden region, one can, in the same way as described in [7], inductively define *higher order* doomed regions: With the updated capacity consumption, new deadlocks may arise at the intersection of the boundaries of the new forbidden region $F(\bar{r})$ and the previous forbidden region F. Modifying the recursive algorithm in [7], one obtains a program that is *deadlockfree and with literally the same path spaces as before* - if just the target is not contained in any of the doomed regions (from which it cannot terminate correctly).

5.4. **Disconnected futures.**

5.4.1. *Vertices with disconnected future links.* As a consequence of Corollary 3.9(4), a vertex v has a disconnected non-empty future link $lk^+(X;v)$ if and only if

(1) $\mathbf{c}(v)-\boldsymbol{\kappa}\geq\mathbf{0}$ (ie v is not contained in the forbidden region);
(2) $\mathbf{c}_{R(v)}(v)-\boldsymbol{\kappa}_{R(v)}$ is a standard unit vector $\mathbf{e}_r,\ r\in R(v)$ (with a single coordinate 1, all others 0);
(3) $dr(v)>1$.

Such a *critical vertex* v can be characterized by the following data (just a small variation compared to the characterization of deadlocks in Section 5.3.1):

- A subset $R'\subseteq R$ of resources including a particular element $r^0\in R'$ (ie $R'=R(v)$);
- For each $r\in R'$ a subset $C(r)\subseteq P$ of processors such that $|C(r^0)|=\kappa(r^0)-1, |C(r)|=\kappa(r), r\in R'\setminus\{r^0\}$, and a non-empty subset $D(r)\subseteq P\setminus C(r)$ of its complement such that $|D(r^0)|\geq 2$ and $\bigsqcup_{r\in R'}D(r)=P=[1:n]$ is a partition of P (ie the $D(r)$ are disjoint)
- Every processor $j\in C(r),\ r\in R'$, has delivered a call Pr in front of and still active at v_j (ie $r\in R'\Rightarrow cr_j(v_j)=1, cr(v)\geq\kappa(r), r\neq r^0, cr^0(v)\geq\kappa(r^0)-1$);
- For $j\in D(r)$, a call Pr is issued at v_j (ie $dr_j(v)=1$);
- No other call $Pr, r\in R'$, is (still) active at v ($\Rightarrow cr^0(v)=\kappa(r^0)-1,\ cr(v)=\kappa(r),\ r\neq r^0$);
- For all $r'\in R\setminus R'$, at most $\kappa(r')$ calls Pr' are active at v (ie $cr'(v)\leq\kappa(r')$).

5.4.2. *Doomed region for disconnectivity.* Analogous to the doomed region $D(v)$ below a deadlock vertex v from Section 5.3.2, there is a *critical* region $D^1(v)$ below a vertex v with disconnected future link such that path spaces $\vec{P}(X)^t_y$ are *disconnected* for $y\in D^1(v)$:

Let v denote a vertex satisfying the conditions in Section 5.4.1, with resource calls to $R(v)\subseteq R$ and predecessor vertex $w=p_{R(v)}(v)$. Define $D^1(v):=]w,v]$. Note that a d-path can leave this hyperrectangle only through a hyperplane $x_i = v_i$ for $i \in C(r^0)$; at all other upper boundary hyperplanes $x_j = v_j, j \notin C(r^0)$, it would enter the forbidden region. Moreover, such a d-path can enter $x_i > v_i$ for *only one* $i \in C(r^0)$ without entering the forbidden region.

Let $x = s_{R(v)}(v) \in V_{R(v)}(X) \subset L(X)$ denote the successor vertex of v with respect to $R' = R(v)$; cf Section 5.1. The intersection of the state space X with $]w, x[\setminus]w, v]$ – that every d-path from $D^1(v)$ needs to enter – has the form $\coprod_{i\in C(r^0)}]v_i, x_i] \times \prod_{j\neq i}]w_j, v_j[$. Remark that the subspaces in that disjoint union are not connected to each other: exactly one coordinate is larger than v_i.

As for deadlocks, one can eliminate $]w, x[$ – and hence the critical region $D^1(v)$ – from the state space by adding an additional resource of capacity $n-1$ locked, for each processor $j \in P$, at w_j and relinquished at x_j. The arising new state space may contain further deadlocks at the intersection of old and new forbidden regions: the associated doomed regions consist of those points u such that every d-path starting at u needs to pass through the critical region $D^1(v)$ – with disconnected path spaces (with source u) as a consequence; these new doomed regions are *higher order critical regions* with respect to the vertex v. Eliminating *all* critical regions and associated higher order critical regions regions results in a state space $\tilde{X} \subseteq X$ with all spaces of d-paths between vertices being path-connected.

5.4.3. *Estimation of the number of path components.* Mutually reachable critical vertices (with spare capacity 1), or rather their future links allow estimating the number of path components of the space of d-paths between vertices: For every critical vertex $c \in X_0$, consider the connected components c_i of its future link (there are at most n of them) and the partial order relation $\preceq$ given by reachability (within X) between components of various critical vertices. A (possibly empty) chain of components of future links (between a given source s and target vertex t) can be realized by a d-path since reachability was assumed. It is known that every d-path in $\vec{P}(X)_s^t$ is d-homotopic to a *tame* d-path with source s and target t (Ziemiański [19, Prop. 6.28],[21, Theorem 5.6], Raussen [16, Theorem 2.6]) that can only transit from one cube to another at a vertex. If a tame path enters a critical region $D^1(v)$, it has to leave it at its top vertex c and along one of the 1-cubes c_i.

Moreover, two d-paths realizing a chain of components (no other critical vertices and critical regions involved!) are d-homotopic to each other. This can be seen by a minor modification of the proof of Corollary 4.4: At every critical vertex, only one of the possible future components is allowed. Excluding deadlocks and non-selected components of future links, all remaining vertices have a spare capacity at least 2.

Hence, the number of path components of $\vec{P}(X)_s^t$ can be estimated (from above) by the number of chains described above. If this number is not too large, the possible outcomes of *all* executions can thus be determined by running one execution along every such chain.

References

[1] R. Belton etal., *Towards Directed Collapsibility*, In: Advances in Mathematical Sciences: AWM Research Symposium, Springer (2020), 255 -271. 2, 3, 5, 10

[2] R. Belton etal., *Combinatorial Conditions for Directed Collapsing*, arXiv:2106.01524. 3, 5, 10

[3] A. Björner, *Nerves, fibers and homotopy groups*, J. Combin. Theory Ser. A **102** (2003), no. 1, 88 – 93. 2, 3, 10

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

[5] L. Fajstrup, *Dipaths and dihomotopies in a cubical complex*, Adv. Appl. Math. **35**, no. 2, 188 – 206. 9

[6] L. Fajstrup, É. Goubault, E. Haucourt, S. Mimram, M. Raussen, Directed Algebraic Topology and Concurrency, Springer, Cham, 2016. 1, 2, 3, 9, 16
[7] L. Fajstrup, É. Goubault and M. Raussen, *Detecting deadlocks in concurrent systems*. CONCUR 98, Lect. Notes Comput. Sci. (Springer) **1466** (1998), 332 – 347. 3, 15, 16
[8] M. Herlihy, D. Kozlov and S. Rajsbaum, Distributed Computing through Combinatorial Topology, Morgan Kaufman, Waltham, MA, USA, 2014. 2
[9] A. Hatcher, Algebraic Topology, Cambridge Univ. Press, 2002. 8, 10
[10] R. Meshulam and M. Raussen, *Homology of spaces of directed paths in Euclidean pattern spaces*. in: A journey through discrete mathematics, 593 – 614, Springer, Cham, 2017. 6
[11] V. Pratt, *Modelling concurrency with geometry*, Proc. of the 18th ACM Symposium on Principles of Programming Languages. (1991), 311–322. 1
[12] M. Raussen, *On the classification of dipaths in geometric models for concurrency*, Math. Struct. Comput. Sci. **10** (2000), no. 4, 427 – 457. 2
[13] M. Raussen, *Trace spaces in a pre-cubical complex*, Topology Appl. **156** (2009), no. 9, 1717 – 1728. 10
[14] M. Raussen, *Simplicial models of trace spaces*, Algebr. Geom. Topol. **10** (2010), no. 3, 1683 – 1714. 2
[15] M. Raussen, *Simplicial models of trace spaces II: General Higher Dimensional Automata*, Algebr. Geom. Topol. **12**, no. 3 (2012), 1745 – 1765. 2
[16] M. Raussen, *Strictifying and taming directed paths in Higher Dimensional Automata*, arXiv:2006.05797; to appear in Math. Struct. Comput. Sci. 17
[17] M. Raussen and K. Ziemiański, *Homology of spaces of directed paths on Euclidean cubical complexes*, J. Homotopy Relat. Struct. **9**, no. 1 (2014), 67 – 84. 4, 10
[18] R.J. van Glabbeek, *On the Expressiveness of Higher Dimensional Automata*, Theor. Comput. Sci. **368** (2006), no. 1-2, 168 – 194. 1
[19] K. Ziemiański, *A cubical model for path spaces in d-simplicial complexes*, Topology Appl. **159**, no. 8, 2127– 2145. 17
[20] K. Ziemiański, *On execution spaces of PV-programs*, Theoret. Comput. Sci. **619** (2016), 87 – 98. 2, 3, 5, 7, 10
[21] K. Ziemiański, *Spaces of directed paths on pre-cubical sets*, Appl. Algebra Eng. Commun. Comput. **28** (2017), 497 – 525. 1, 2, 17
[22] K. Ziemiański, *Spaces of directed paths on pre-cubical sets II*, J. Appl. Comput. Topol. **4** (2020), 45 – 78. 1, 2

Department of Mathematical Sciences, Aalborg University, Skjernvej 4A, DK-9220 Aalborg Øst, Denmark
Email address: raussen@math.aau.dk