The Recursive Arrival Problem

Citation for published version:
Webster, T 2023, The Recursive Arrival Problem. in *Proceedings of the 14th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2023.* vol. 390, Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, pp. 168–184, Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, Udine, Italy, 18/09/23. https://doi.org/10.4204/EPTCS.390.11

Digital Object Identifier (DOI):
10.4204/EPTCS.390.11

Document Version:
Peer reviewed version

Published In:
Proceedings of the 14th International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2023

The Recursive Arrival Problem

Thomas Webster
University of Edinburgh, UK
Thomas.Webster@ed.ac.uk

We study an extension of the `Arrival` problem, called `Recursive Arrival`, inspired by Recursive State Machines, which allows for a family of switching graphs that can call each other in a recursive way. We study the computational complexity of deciding whether a Recursive Arrival instance terminates at a given target vertex. We show this problem is contained in $\mathsf{NP} \cap \mathsf{coNP}$, and we show that a search version of the problem lies in UEOPL, and hence in $\mathsf{EOPL} = \mathsf{PLS} \cap \mathsf{PPAD}$. Furthermore, we show P-hardness of the Recursive Arrival decision problem. By contrast, the current best-known hardness result for `Arrival` is PL-hardness.

1 Introduction

`Arrival` is a simply described decision problem defined by Dohrau, Gärtner, Kohler, Matoušek and Welzl [5]. Informally, it asks whether a train moving along the vertices of a given directed graph, with n vertices, will eventually reach a given target vertex, starting at a given start vertex. At each vertex, v, there are (without loss of generality) two outgoing edges and the train moves deterministically, alternately taking the first out-edge, then the second, switching between them if and when it revisits that vertex repeatedly. This process is known as "switching" and can be viewed as a deterministic simulation of a random walk on the directed graph. It can also be viewed as a natural model of a state transition system where a local deterministic cyclic scheduler is provided for repeated transitions out of each state.

Dohrau et al. showed that this `Arrival` decision problem lies in the complexity class $\mathsf{NP} \cap \mathsf{coNP}$, but it is not known to be in P. There has been much recent work showing that a search version of the `Arrival` problem lies in subclasses of TFNP including PLS [17], CLS [13], and UEOPL [12], as well as showing that `Arrival` is in $\mathsf{UP} \cap \mathsf{coUP}$ [13]. There has also been progress on lower bounds, including PL hardness and CC hardness [18]. Further, another recent result by Gärtner et al. [14] gives an algorithm for `Arrival` with running time $2^{\mathcal{O}(\sqrt{n}\log(n))}$, the first known sub-exponential algorithm. In addition, they give a polynomial-time algorithm for "almost acyclic" instances. Auger et al. also give a polynomial-time algorithm for instances on a "tree-like multigraph" [2].

The complexity of `Arrival` is particularly interesting in the context of other games on graphs. For instance, Condon's simple stochastic games, mean-payoff games, and parity games [4, 20, 16], where the two-player variants are known to be in $\mathsf{NP} \cap \mathsf{coNP}$ and the one-player variants have polynomial time algorithms. `Arrival`, however, is a zero-player game that has no known polynomial time algorithm and, furthermore, Fearnley et al. [11] that a one-player generalisation of Arrival is, in fact, NP-complete, in stark contrast to these two-player graph games.

We introduce and consider a new generalisation of Arrival that we call `Recursive Arrival`, in which we are given a finite collection of Arrival instances ("components") with the ability to, from certain nodes, invoke each other in a potentially recursive way. Each component has a set of entries and a set of exits, and we study the complexity of deciding whether the run starting from a given entry of a given component reaches a given exit of that component, which may involve recursive calls to other components.

Submitted to:
GandALF 2023

Our model is inspired by work on recursive transition systems started by Alur et al. [1] with Recursive State Machines (RSMs) modelling sequential imperative programming. These inspired further work on Recursive Markov Chains (RMCs), Recursive Markov Decision Processes (RMDPs), and Recursive Simple Stochastic Games (RSSGs) by Etessami and Yannakakis [8, 9, 10]. RSMs (and RMCs) are essentially "equivalent" (see [9]) to (probabilistic) pushdown systems [3, 6] and have applications in model-checking of procedural programs with recursion.

There is previous work on Arrival generalisations including a variant we call `Succinct Arrival`, where at a vertex v the alternation takes the first outgoing edge of v on the first A_v visits and then the second edge the next B_v visits, repeating this sequence indefinitely. The numbers A_v and B_v are given succinctly in binary as input, and hence may be exponentially larger than the bit encoding size of the instance. Fearnley et al. showed that `Succinct Arrival` is P-hard and in NP$\cap$coNP [11]. However, we do not know any inter-reducibility between `Recursive Arrival` and `Succinct Arrival` variants. In [19], we also defined and studied a generalisation of `Arrival` that allows both switching nodes as well as randomised nodes, and we showed that this results in PP-hardness and containment in PSPACE for (quantitative) reachability problems.

In this paper, we show that the `Recursive Arrival` problem lies in NP$\cap$coNP, like `Arrival`, by giving a generalised witness scheme that efficiently categorises both terminating and non-terminating instances. We also show that the natural search version of `Recursive Arrival` is in both PLS and PPAD and in fact in UEOPL, by giving a reduction to a canonical UEOPL problem. We also show P-hardness for the `Recursive Arrival` problem by reduction from the Circuit Value Problem. This contrasts with the current best-known hardness result for `Arrival`, which is PL-hardness ([18]).

Due to space limitations, many proofs are relegated to the appendix.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \dots\}$ denote the natural numbers, and let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. We assume the usual ordering on elements of $\mathbb{N}_\infty$. For $j \in \mathbb{N}$ and $k \in \mathbb{N}_\infty$, we use the notion $[j \dots k] = \{i \in \mathbb{N} \mid j \leq i \leq k\}$, and we define $[k] = [1 \dots k]$. All propositions of this section follow directly from the cited prior works.

Definition 2.1. A *switch graph* is given by a tuple $G := (V, s^0, s^1)$ where, for each $\sigma \in \{0,1\}$, $s^\sigma : V \to V$ is a function from vertices to vertices. $\square$

Given a Switch Graph G, we define its directed *edges* to be the set $E := \{(v, s^0(v)) \mid v \in V\} \cup \{(v, s^1(v)) \mid v \in V\}$, with self-loops allowed. We write $E_\sigma := \{(v, s^\sigma(v)) \mid v \in V\}$ for $\sigma \in \{0,1\}$ to refer to edges arising specifically from transitions $s^\sigma(v)$, for each vertex v.

Given a switch graph, $G := (V, s^0, s^1)$, we say $q : V \to \{0,1\}$ is a *switch position* on V. We let Q be the set of all switch positions on V and define $q^0 \in Q$ by $q^0(v) = 0$ for all $v \in V$ as the *initial switch position*. Given a switch graph, we say a *state* of the graph is an ordered pair $(v, q) \in V \times Q$ and we let $\Gamma = V \times Q$ be the *state space*. We define the "flip action", *flip* $: V \times Q \to Q$, of a vertex on a switch position, as follows: $\mathit{flip}(v,q)(u) = q(u)$ for all $u \in V \setminus \{v\}$ and $\mathit{flip}(v,q)(v) = 1 - q(v)$, i.e., this action flips the function value of q at v only. We can then define a transition function $\delta : \Gamma \to \Gamma$ on a switch graph as $\delta((v,q)) = (s^{q(v)}(v), \mathit{flip}(v,q))$.

We define the *run* of a switch graph G with initial state $\gamma_0 := (v_0, q_0)$ to be the unique infinite sequence over Γ, $\mathsf{RUN}^\infty(G, \gamma_0) := (\gamma_i)_{i=0}^\infty$ satisfying $\gamma_{i+1} := \delta(\gamma_i)$ for $i \geq 0$. For a vertex $v \in V$, we say a run *terminates at* v if $\exists t \in \mathbb{N}$ such that $\forall i \geq t\ \exists q_i \in Q$ with $\gamma_i = (v, q_i)$. We call $T \in \mathbb{N}_\infty$ the *termination time* defined by $T := \inf\{t \mid \forall i \geq t,\ v_i = v_t\}$, where $\inf \emptyset = \infty$. We denote by $\mathsf{RUN}(G, \gamma_0) := (\gamma_i)_{i=0}^T$ the subsequence of $\mathsf{RUN}^{\infty}(G,\gamma_0)$ up to the termination time T. We say a run *hits* a vertex $v \in V$ if $\exists t \in \mathbb{N}$ and $\exists q_t \in Q$ such that $\gamma_t = (v, q_t)$.

We note that in order to terminate at a vertex, $v \in V$, we must have that $v = s^0(v) = s^1(v)$. We define the set of "Dead Ends" in the instance as $\mathsf{DE} := \{v \in V \mid s^0(v) = s^1(v) = v\}$. From this definition, it is obvious that we either terminate at some unique vertex $v \in \mathsf{DE}$, or we never terminate. We may now define the Arrival Decision problem:

Arrival	
Instance:	A Switch Graph $G := (V, s^0, s^1)$ and vertices $o, d \in V$.
Problem:	Decide whether or not the run of switch graph G with initial state (o, q^0) terminates at vertex d.

Given a switch graph G with directed edges, E, we define the relations $\to^*, \to^+ \subseteq V \times V$ as follows $u \to^* v$ (resp. $\to^+$) for $u, v \in V$ if and only if there is a directed path $w_0, \dots, w_k \in V$ with $(w_i, w_{i+1}) \in E$ for $i \in [k-1]$, with $w_0 = u$ and $w_k = v$ for $k \geq 0$ (resp. $k \geq 1$) from u to v in (V, E). We write $u \not\to^* v$ (resp. $\not\to^+$) whenever we do not have $u \to^* v$ (resp. $u \to^+ v$).

We note that we can view the sequence of vertices visited on a run as a directed path in (V, E), thus if the run with initial state (v, q) hits w then we can conclude $v \to^* w$ and, contrapositively, if $v \not\to^* w$ then for all $(v, q) \in Q$ the run starting at (v, q) does not hit w.

We let $\mathbb{I}\{a = b\}$ be the indicator function of $a = b$, which is equal to 1 if $a = b$ and is equal to 0 otherwise. We now define a switching flow, rephrasing Definition 2 of Dohrau et al. [5]:

Definition 2.2 ([5, Definition 2])**.** Let $G := (V, s^0, s^1)$ be a switch graph, and let $o, d \in V$ be vertices. We define a *switching flow* on G from o to d as a vector $\boldsymbol{x} := (x_e \mid e \in E)$ where $x_e \in \mathbb{N}$ such that the following family of conditions hold for each $v \in V$:

$$\text{Flow Conservation}: \left(\sum_{e=(u,v)\in E} x_e\right) - \left(\sum_{e=(v,w)\in E} x_e\right) = \mathbb{I}\{v = d\} - \mathbb{I}\{v = o\}, \quad \forall v \in V,$$
$$\text{Parity Condition}: \quad x_{(v,s^1(v))} \leq x_{(v,s^0(v))} \leq x_{(v,s^1(v))} + 1, \qquad \forall v \in V. \qquad \square$$

We note that given G, o and a switching flow $\boldsymbol{x}$ from o to some, unknown, vertex $d \in V$, we can compute exactly which d by verifying the equalities. We refer to d as the *current-vertex* of the switching flow. If $o \in V$ is an initial vertex and $t \in \mathbb{N}$ a time, we let $\mathsf{RUN}(G, (o, q^0)) := ((v_i, q_i))_{i=0}^{\infty}$ be the run, and define the *Run Profile* to time t to be the vector $\boldsymbol{run}(o,t) := (|\{i \in [t] \mid (v_{i-1}, v_i) = e\}| \mid e \in E)$. It follows that for any $o \in V$ and $t \in \mathbb{N}$ that $\boldsymbol{run}(o,t)$ is a switching flow from o to some vertex $d \in V$ [5, Observation 1]. We say a switching flow $\boldsymbol{x}$ is *run-like* if there exists some $t \in \mathbb{N}$ such that $\boldsymbol{x} = \boldsymbol{run}(o,t)$.

It follows directly from the results of Dohrau et al.[5] and Gartner et al.[13] that:

Proposition 2.3 ([5, 13])**.** *There exists a polynomial function* $\mathrm{p}: \mathbb{N} \to \mathbb{N}$ *such that for all Switch Graphs* $G := (V, s^0, s^1)$ *and all vertices* $o, d \in V$ *with* $o \neq d$ *and* $d \in \mathsf{DE}$ *the following are equivalent:*

- *The run on* G *from initial state* (o, q^0) *terminates at* d.
- *There exists a run-like switching flow* $\boldsymbol{x}$ *on* G *from* o *to* d *satisfying* $\forall e \in E$, *that* $\log_2(x_e) \leq \mathrm{p}(|G|)$.

Furthermore, for the same polynomial p*, the following are equivalent:*

- *The run on* G *from initial state* (o, q^0) *does not terminate.*
- *There exists a vertex* $d' \in V \setminus \mathsf{DE}$*, a run-like switching flow* $\boldsymbol{x}$ *on* G *from* o *to* d'*, and an edge* $e' = (u, d') \in E$ *which satisfies for all* $e \in E \setminus \{e'\}$ *that* $\log_2(x_e) \leq \mathrm{p}(|G|)$ *and that* $x_{e'} = 2^{\mathrm{p}(|G|)} + 1$.

It follows from these results that `Arrival` is in $\mathsf{NP} \cap \mathsf{coNP}$, as we may non-deterministically guess a vector, $\boldsymbol{x}$, whose coordinate entries are bounded by $2^{\mathrm{p}(|G|)}+1$, and then verify whether or not $\boldsymbol{x}$ is a run-like switching flow. Using [13, Lemma 11] we may verify the run-like condition in polynomial time, on which we will elaborate subsequently. If we find a run-like switching flow to some dead end $d' \in \mathsf{DE}$ we may conclude G terminates at d' and by the first part of Proposition 2.3 we can find such a flow within these bounds. This may be either a flow to the given dead-end d in our input, or to some other dead-end, certifying non-termination at d. The last case of Proposition 2.3 says that when G does not terminate anywhere, we may also find a flow certifying this within our bounds, namely with some coordinate value of the guessed vector $\boldsymbol{x}$ being exactly$2^{\mathrm{p}(|G|)}+1$. In fact, it was shown by [13] that this argument also shows containment of `Arrival` in $\mathsf{UP} \cap \mathsf{coUP}$, by showing there is a unique witness $\boldsymbol{x}$ satisfying just one of these conditions.

Let $G := (V, s^0, s^1)$ be a Switch Graph and let $\boldsymbol{x}$ be a switching flow on G between some vertices $o, d \in V$. We define the *last-used-edge* graph $G^*_{\boldsymbol{x}} := (V, E^*_{\boldsymbol{x}})$ with the following set of edges:

$$E^*_{\boldsymbol{x}} := \{(v, s^0(v)) \mid v \in V \text{ and } x_{(v,s^0(v))} \neq x_{(v,s^1(v))}\} \cup \{(v, s^1(v)) \mid v \in V \text{ and } x_{(v,s^0(v))} = x_{(v,s^1(v))} > 0\}$$

This graph contains at most one of the edges $(v, s^0(v))$ or $(v, s^1(v))$. If $x_{(v,s^0(v))} + x_{(v,s^1(v))} > 0$, then assuming there exists some run on which we visit vertex v a total of $x_{(v,s^0(v))} + x_{(v,s^1(v))}$ times, $E^*_{\boldsymbol{x}}$ contains the edge out of v that our switching order would use the last time v was visited on such a run. If on the other hand $x_{(v,s^0(v))} + x_{(v,s^1(v))} = 0$, then $E^*_{\boldsymbol{x}}$ contains neither edge.

Proposition 2.4 ([13])**.** *Let $G := (V, s^0, s^1)$ be a Switch Graph and let $\boldsymbol{x}$ be a switching flow on G from $o \in V$ to $d \in V$, then there exists a unique $t \in \mathbb{N}$ such that $\boldsymbol{x} = \boldsymbol{run}(o,t)$, if and only if one of the following two (mutually exclusive) conditions hold:*

- *The graph $G^*_{\boldsymbol{x}}$ is acyclic,*
- *The graph $G^*_{\boldsymbol{x}}$ contains exactly one cycle and d is on this cycle,*

Furthermore, given G and any such $\boldsymbol{x}$ whether or not one of these conditions hold can be checked in polynomial time in the size of G and $\boldsymbol{x}$.

Proposition 2.5 ([13, Lemma 16])**.** *Let $G := (V, s^0, s^1)$ be a Switch Graph and let $t \in \mathbb{N}$ with $\boldsymbol{run}(o,t)$ the run profile up to time t, which is a switching flow on G from $o \in V$ to some vertex $d \in V$. Then at least one of the following two conditions hold:*

- *There is a unique edge $(u, d) \in E^*_{\boldsymbol{run}(o,t)}$ incoming to d in the graph $G^*_{\boldsymbol{run}(o,t)}$.*
- *The graph $G^*_{\boldsymbol{run}(o,t)}$ contains exactly one cycle, and that cycle contains exactly one edge of the form $(u, d) \in E^*_{\boldsymbol{run}(o,t)}$ on the cycle.*

Moreover, the edge (u, d) was the edge traversed at time t in the run (i.e., if $\mathsf{RUN}^\infty(G, (o, q^0)) = ((v_i, q_i))_{i=0}^\infty$ then $v_{t-1} = u$ and $v_t = d$). Furthermore, this uniquely determined edge can be computed given G and $\boldsymbol{run}(o,t)$ in time polynomial in the size of G and $\boldsymbol{run}(o,t)$.

Using these results, we are able to efficiently (in P-time) compute a function LUE which takes a switching flow of the form $\boldsymbol{run}(o,t)$ and returns the "last-used-edge", namely the unique edge $(u, d) \in E$ guaranteed by Proposition 2.5, where (u, d) is the edge which was traversed at time t.

2.1 The Recursive Arrival Problem

We consider a recursive generalisation of Arrival in the spirit of Recursive State Machines, etc. ([1, 9, 10]). A Recursive Arrival instance is defined as follows:

Definition 2.6. A *Recursive Arrival* graph is given by a tuple, $(G^1,\dots,G^k)$, where each *component* $G^i := (N_i \cup B_i, Y_i, \mathsf{En}_i, \mathsf{Ex}_i, \delta_i)$ consists of the following pieces:

- A set N_i of *nodes* and a (disjoint) set B_i of *boxes*.
- A labelling $Y_i : B_i \to \{1,\dots,k\}$ that assigns every box an index of one of the components $G^1,\dots,G^k$.
- A set of *entry nodes* $\mathsf{En}_i \subseteq N_i$ and a set of *exit nodes* $\mathsf{Ex}_i \subseteq N_i$.
- To each box $b \in B_i$, for all $i \in [k]$, we associate a set of *call ports*, $\mathsf{Call}_{b,i} = \{(b,o) \mid o \in \mathsf{En}_{Y_i(b)}\}$ corresponding to the entries of the corresponding component, and a set of *return ports*, $\mathsf{Return}_{b,i} = \{(b,d) \mid d \in Ex_{Y_i(b)}\}$ corresponding to the exits of the corresponding component. We define the sets $\mathsf{Call}_i = \cup_{b\in B_i} \mathsf{Call}_{b,i}$ and $\mathsf{Return}_i = \cup_{b\in B_i} \mathsf{Return}_{b,i}$. We will use the term *ports* of G^i to refer to the set $\mathsf{Port}_i = \mathsf{Call}_i \cup \mathsf{Return}_i$, of all call ports and return ports associated with all boxes $b \in B_i$ that occur within the component G^i.
- A *transition relation*, δ_i, where transitions are of the form (u,σ,v) where:
  1. The source u is either a node in $N_i \setminus \mathsf{Ex}_i$ or a return port (b,x) in Return_i. We define $\mathsf{Sor}_i = N_i \setminus \mathsf{Ex}_i \cup \mathsf{Return}_i$ to be the set of all *source vertices*.
  2. The label σ is either 0 or 1.
  3. The destination v is either a node in $N_i \setminus \mathsf{En}_i$ or a call port (b,e) where b is a box in B_i and e is an entry node in En_j for $j = Y_i(b)$; we call this the set Dest_i of *destination vertices*.

  and we require that the relation δ_i has the following properties:

  1. For every vertex $u \in \mathsf{Sor}_i$ and each $\sigma \in \{0,1\}$ there is a unique vertex $v \in \mathsf{Dest}_i$ with $(u,\sigma,v) \in \delta_i$. Thus, for each $i \in [k]$ and $\sigma \in \{0,1\}$, we can define total functions $s_i^\sigma : \mathsf{Sor}_i \to \mathsf{Dest}_i$ by the property that $(u,\sigma,s_i^\sigma(u)) \in \delta_i$, for all $u \in \mathsf{Sor}_i$. □

We will use the term *vertices* of G^i, which we denote by V_i to refer to the union $V_i = N_i \cup \mathsf{Port}_i$ of its set of nodes and its set of ports. For $\sigma \in \{0,1\}$, we let $E_i^\sigma = \{(u,v) \mid (u,\sigma,v) \in \delta_i\}$ be the set of underlying edges of δ_i with label σ, and we define $E_i := E_i^0 \cup E_i^1$. We will often alternatively view components as being equivalently specified by the pair of functions (s_i^0, s_i^1), which define the transition function $\delta_i := \{(u,\sigma,s_i^\sigma(u)) \mid u \in \mathsf{Sor}_i, \sigma \in \{0,1\}\}$.

We can view a box as a "call" to other components, and, as such, it is natural to ask which components "call" other components. Given an instance of Recursive Arrival, $(G^1,\dots,G^k)$, we define its *Call Graph* to be the following directed graph, $C = ([k], E_C)$. Our vertices are component indices and for all $(i,j) \in [k]\times[k]$ let $(i,j) \in E_C$ if and only if there exists some $b \in B_i$ with $j = Y_i(b)$ (i.e., a component G^i can make a call to component G^j). We allow self-loop edges in this directed graph, which correspond to a component making a call to itself.

We are also able to lift some definitions from non-recursive Arrival to analogous definitions about Recursive Arrival instances. Firstly, we define the sets $\mathsf{DE}_i := \{v \in \mathsf{Sor}_i \mid s_i^0(v) = s_i^1(v) = v\} \cup \mathsf{Ex}_i$, of dead-ends of each component. This contains both vertices $v \in \mathsf{Sor}_i$ where both outgoing transitions are to itself and all the exits of the component.

In a given component, G^i, we define a *switch position* on G^i as a function $q : \mathsf{Sor}_i \to \{0,1\}$. We let Q_i be the set of all switch position functions on G^i. We let $q_i^0 \in Q_i$ be the function $q_i^0(v) = 0$ for all

$v \in \mathsf{Sor}_i$ and call this the *initial switch position*. We define the action $\mathit{flip}_i : \mathsf{Sor}_i \times Q_i \to Q_i$ analogously to non-recursive Arrival, which flips the bit corresponding to a given vertex in a given switch position.

A *state* of a Recursive Arrival graph $(G^1, \dots, G^k)$ is given by a tuple $\gamma := ((b_1, q_1) \dots (b_r, q_r), (v, q))$ where the *call stack* $\beta := (b_1, q_1) \dots (b_r, q_r)$ is a string of pairs (b_i, q_i) with each $b_i \in \cup_k B_k$ a box, q_i is a switch position on some component G^{c_i} (i.e. $q_i \in Q_{c_i}$), and the *current position* is the pair (v, q) where $v \in V_{c_{r+1}}$ is a vertex in some component $G^{c_{r+1}}$ and $q \in Q_{c_{r+1}}$ is a switch position on $G^{c_{r+1}}$. We call the sequence $(c_1, \dots, c_r, c_{r+1})$ the *component call-stack* of the state. We say that a state is *well-formed* if:

- For all $i \in [r]$ we have $b_i \in B_{c_i}$.
- The sequence satisfies $Y_{c_i}(b_i) = c_{i+1}$ for $i \in [r]$.

We let Γ be the set of all well-formed states and $\Gamma_S := \{\beta : \exists (v, q),\ (\beta, (v, q)) \in \Gamma\}$ be the set of well-formed stacks β appearing in some state of Γ.

We define the transition function $\delta : \Gamma \to \Gamma$ on a well-formed state $\gamma := ((b_1, q_1) \dots (b_r, q_r), (v, q))$ as:

1. If $v \in \mathsf{Sor}_j$ is a source vertex then we let $v' := s^{q(v)}(v)$ and then we define $\delta(\gamma) := ((b_1, q_1), \dots, (b_r, q_r), (v', \mathit{flip}_j(v, q)))$;
2. If $v = (b, e) \in \mathsf{Call}_j$ then $e \in \mathsf{En}_{j'}$ for $j' = Y_j(b)$. We let $q^0_{j'}$ be the initial switch position on $G^{j'}$ and define $\delta(\gamma) := ((b_1, q_1) \dots (b_r, q_r)(b, q), (e, q^0_{j'}))$;
3. If $v \in \mathsf{Ex}_j$ and $r \geq 1$ then we define $\delta(\gamma) := ((b_1, q_1) \dots (b_{r-1}, q_{r-1}), ((b_r, v), q_r))$;
4. If $v \in \mathsf{Ex}_j$ and $r = 0$ then $\delta(\gamma) := \gamma$;

The function $\delta : \Gamma \to \Gamma$ defines a deterministic transition system on well-formed states. We call the *run* of a Recursive Arrival graph from an initial component index $j \in [k]$, an initial switch position $q_0 \in Q_j$ and a start entrance $o \in \mathsf{En}_j$ the (infinite) sequence $\mathsf{RUN}^\infty(G, (o, q_0)) := (\gamma_i)_{i=0}^\infty$ given by $\gamma_0 := (\varepsilon, (o, q_0))$ and $\gamma_{i+1} := \delta(\gamma_i)$. We say a run *terminates* at an exit $d \in \mathsf{Ex}_j$ if there $\exists t \in \mathbb{N}$ such that $\forall i \geq t$ there $\exists q_i \in Q_j$ such that $\gamma_i = (\varepsilon, (d, q_i))$. We call $T \in \mathbb{N}_\infty$ the termination time defined by $T := \inf\{t \mid \forall i \geq t,\ v_i \in \mathsf{Ex}_j\}$, where $\inf(\emptyset) = \infty$. We denote by $\mathsf{RUN}(G, (o, q)) := (\gamma_i)_{i=0}^T$ the subsequence up to termination. We say a run *hits* a vertex $v \in V$ if there $\exists t \in \mathbb{N}$, $\exists q_t \in Q$ and $\exists \beta \in \Gamma_S$ with $\gamma_t = (\beta, (v, q_t))$.

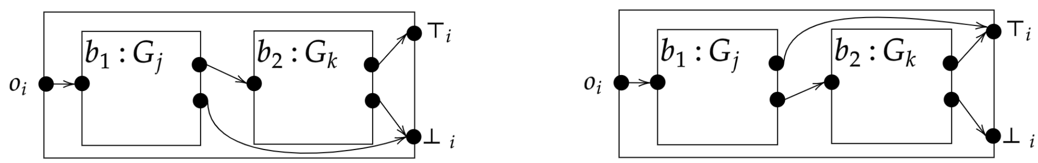
Our decision problem can then be stated as:

`Recursive Arrival`	
Instance:	A Recursive Arrival graph $(G^1, \dots, G^k)$, with $\lvert\mathsf{En}_j\rvert = 1$ for all $j \in [k]$, and a target exit $d \in \mathsf{Ex}_1$
Problem:	Does the run from initial state $(\varepsilon, (o_1, q^0_1))$ terminate at exit d? (Where $o_1 \in \mathsf{En}_1$ is the unique entry of G^1 and $q^0_1 \in Q_1$ is the initial switch position.)

This decision problem covers in full generality any termination decision problem on Recursive Arrival instances, as we may accomplish a change of initial state by renumbering components and relabelling transitions. Also, restricting to models with $\lvert\mathsf{En}_i\rvert = 1$ is without loss of generality, because we can efficiently convert the model into an "equivalent" one where each component has a single entry, by making copies of components (and boxes) with multiple entries, each copy associated with a single entry (single, call port, respectively). This is analogous to the same fact for Recursive Markov Chains, which was noted by Etessami and Yannakakis in [9, p. 16]. Thus, we may assume that in the `Recursive Arrival` problem all components of the instance have a unique entry, i.e., for $i \in [k]$ that $\mathsf{En}_i = \{o_i\}$, and, unless stated otherwise, the run on G refers to the run starting in the state $(\varepsilon, (o_1, q^0_1))$, writing $\mathsf{RUN}(G) := \mathsf{RUN}(G, (o_1, q^0_1))$.

(a) G^1, the constant "true" component. (b) G^2, the constant "false" component.

Figure 1: Initial components corresponding to constant gates "true" and "false".



(a) Component for the AND of gates g_j and g_k. (b) Component for the OR of gates g_j and g_k.

Figure 2: Component G^i, where $j, j' \in [i-1]$ are the indices of the two inputs to the gate g_i. All edges correspond to both s^0 and s^1 transitions.

While, in such an instance, we may make an exponential number of calls to other functions, it turns out we are able to give a polynomial bound on the maximum recursion depth before we can conclude an instance must loop infinitely.

Lemma 2.7. *Let $G := (G^1, \ldots, G^k)$ be an instance of Recursive Arrival and assume the run on G hits some state $(\beta, (v, q))$, with $|\beta| \geq k$. Then the run on G does not terminate.*

3 P-Hardness of Recursive Arrival

Manuell [18] has shown the `Arrival` problem to be PL-hard, which trivially provides the same hardness result for `Recursive Arrival`. This is currently the strongest hardness result known for the `Arrival` problem. By contrast, we now show that the `Recursive Arrival` problem is in fact P-hard.

Theorem 3.1. *The* `2-exit Recursive Arrival` *problem is* P*-hard.*

Proof (Sketch). We show this by reduction from the P-complete Monotone Circuit Value Problem (see e.g., [15]). We construct one component corresponding to each gate of an input boolean circuit. Each component will have two exits, which we refer to as "top", $\top$, and "bottom", $\bot$, (located accordingly in our figures) and we will view these exits as encoding the outputs, "true" and "false" respectively.

Firstly, we show in Figure 1 two components for a constant true and constant false gate of the circuit. Depicted in Figure 2 are two cases corresponding to AND or OR gates. These perform a lazy evaluation of the AND or OR of components G^j and G^k. This process produces a polynomially sized `Recursive Arrival` instance for an input boolean circuit where each component G_j can be shown inductively to reach exit $\top_j$ if and only if it's corresponding gate, g_j, outputs true. $\square$

4 Recursive Arrival is in NP $\cap$ coNP and UEOPL

Recall the notion of Switching Flow for an Arrival instance. For Recursive Arrival, we generalise the notion of a Switching Flow to a tuple of vectors $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k)$, one for each component of the Recursive Arrival instance. We define for each component G^i, $i \in [k]$, and each box $b \in B_i$ the set of *potential edges* $F_{b,i} := \mathsf{Call}_{b,i} \times \mathsf{Return}_{b,i}$, representing the potential ways of crossing the box b, assuming that the box is eventually returned from. We define the sets $F_i := \cup_{b \in B_i} F_{b,i}$. We recall that the set of internal edges of a component G^i is given by $E_i := \{(u,v) \mid u,v \in V_i,\ \exists \sigma \in \{0,1\}, (u,\sigma,v) \in \delta_i\}$. We say the *Flow Space* for component G^i is the set of vectors $\mathscr{F}_i := \mathbb{N}^{|E_i \cup F_i|} := \{(x^i_e \in \mathbb{N} \mid e \in E_i \cup F_i)\}$, where we identify coordinates of these vectors with edges in $E_i \cup F_i$. We define the *Flow Space* of G to be the set $\mathscr{F} := \Pi_{i=1}^{k} \mathscr{F}_i$, a tuple of k vectors, with the i'th vector in the flow space of component G^i. We denote specifically by $\mathbf{0}^i \in \mathscr{F}_i$ the all zero vector, which has $\mathbf{0}^i_e = 0$ for all $e \in E_i \cup F_i$, and $\mathbf{0} \in \mathscr{F}$ the all zero tuple, $\mathbf{0} := (\mathbf{0}^1, \ldots, \mathbf{0}^k)$. We refer to elements of $\mathscr{F}$ (resp. $\mathscr{F}_i$) as flows on G (resp. G^i).

Firstly, we define a switching flow on each component. For a Recursive Arrival instance $G := (G^1, \ldots, G^k)$ and for $l \in [k]$, we call a vector $\boldsymbol{x}^l \in \mathscr{F}_l$ to be a *component switching flow* if the following conditions hold. Firstly, by definition, the all-zero vector $\mathbf{0}^l$ is always considered a component switching flow. Furthermore, by definition, a non-zero vector $\boldsymbol{x}^l \in \mathscr{F}_l \setminus \{\mathbf{0}^l\}$ is called a component switching flow if there exists some *current-vertex* $d^l_{\boldsymbol{x}^l} \in V_l \setminus \{o_l\}$ (which, as we will see, is always uniquely determined when it exists), such that for o_l the unique entry of G^l, $\boldsymbol{x}^l$ satisfies the following family of conditions:

$$\text{Flow Conservation} \quad \begin{cases} \left(\sum_{e=(u,v)\in E_l\cup F_l} x^l_e\right) - \left(\sum_{e=(v,w)\in E_l\cup F_l} x^l_e\right) = 1, & \text{for } v = d^l_{\boldsymbol{x}^l}, \\ \qquad\qquad\qquad\qquad + \left(\sum_{e=(v,w)\in E_l\cup F_l} x^l_e\right) = 1, & \text{for } v = o_l, \\ \left(\sum_{e=(u,v)\in E_l\cup F_l} x^l_e\right) - \left(\sum_{e=(v,w)\in E_l\cup F_l} x^l_e\right) = 0, & \forall v \in V_l \setminus \{o_l, d^l_{\boldsymbol{x}^l}\}, \end{cases}$$

$$\text{Switching Parity Condition} \quad x_{(v,s^1(v))} \le x_{(v,s^0(v))} \le x_{(v,s^1(v))} + 1, \qquad \forall v \in \mathsf{Sor}_l,$$

$$\text{Box Condition} \quad \exists f_b \in F_{b,l} \text{ such that } \forall f \in (F_{b,l} \setminus \{f_b\})\ x^l_f = 0, \qquad \forall b \in B_l$$

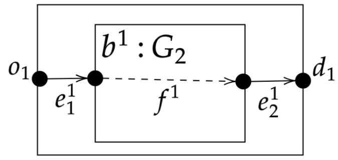
Importantly, note that for any such component switching flow, $\boldsymbol{x}^l$, the current-vertex node $d^l_{x^l}$ is *uniquely* determined. This follows from the fact that the left-hand sides of the Flow Conservation equalities for nodes $v \in V_l \setminus \{o_l\}$ are identical and independent of the specific node v. Hence, if a vector $\boldsymbol{x}^l$ satisfies all of those equalities, there can only be one vertex $v \in V_l \setminus \{o_l\}$ for which the corresponding linear expression on the left-hand side, evaluated over the coordinates of the vector $\boldsymbol{x}^l$, equals 1.

In the case where $\boldsymbol{x}^l = \mathbf{0}^l$, i.e., the all zero-vector, we define the current-vertex of the all-zero component switching flow to be $d^l_{\mathbf{0}^l} := o_l$. We say a component switching flow $\boldsymbol{x}^l \in \mathscr{F}_l$ is *complete* if its current vertex $d^l_{\boldsymbol{x}^l}$ is an exit vertex in Ex_l. These conditions follow the same structure as for non-recursive switching flows, with the additional "Box Condition" only allowing at most one potential edge across each box (i.e., an edge in $F_{b,l}$) to be used.

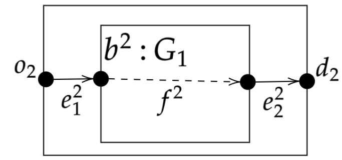
Next, we extend our component switching flows by adding conditions that relate the flows on different components. Consider a tuple $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}$ of vectors, one for each component, such that each $\boldsymbol{x}^i \in \mathscr{F}_i$ is a component switching flow for component G^i. We sometimes write $d^i_{\boldsymbol{X}}$ instead of $d^i_{\boldsymbol{x}^i}$. Let $K_{\boldsymbol{X}} = \{i \in [k] \mid \boldsymbol{x}^i \text{ is complete}\}$ be the subset of indices corresponding to complete component switching flows. We then say the tuple $\boldsymbol{X} \in \mathscr{F}$ is a *recursive switching flow* if for every $l \in [k]$, $b \in B_l$ and $f \in F_{b,l}$, the following holds:

- $\boldsymbol{x}^l \in \mathscr{F}_l$ is a component switching flow for component G^l, and
- if $x^l_f > 0$ then $Y_l(b) \in K_{\boldsymbol{X}}$, and
- if $x^l_f > 0$, then letting $d^{Y_l(b)}_{\boldsymbol{X}} \in \mathsf{Ex}_{Y_l(b)}$ be the current vertex of $\boldsymbol{x}^{Y_l(b)}$, we must have that $f = ((b, o_{Y_l(b)}), (b, d^{Y_l(b)}_{\boldsymbol{X}}))$.

We define $\mathscr{R} \subset \mathscr{F}$ to be the set of all recursive switching flows. These conditions ensure "consistency" in the following way; if we use an edge $f \in F_{b,l}$ then we have a component switching flow on component



(a) Initial component G^1.



(b) Component G^2.

Figure 3: A Recursive Arrival instance G on which there exists a recursive switching flow $(\boldsymbol{x}^1, \boldsymbol{x}^2)$ on G whose current vertex is in G^1 is d_1 however the run on G does not terminate, or even hit the exit d_1.

$G^{Y_l(b)}$ which is complete and reaches the exit matching the edge f, and we are taking that same edge across all boxes with the same label. We note our definition implies $\mathbf{0} \in \mathscr{R}$, thus there is always at least one recursive switching flow. These conditions can be verified in polynomial time.

We will view recursive switching flows as hypothetical partial "runs" on each component, where an edge $e \in E_l \cup F_l$ is used x_e^l times along this "run". It may well be the case no such run actually exists. However, unlike the case of non-recursive switching flows in Arrival, it is no longer the case that any recursive switching flow where the current vertex is $d_{\boldsymbol{X}}^1$ in component G^1, and where $d_{\boldsymbol{X}}^1 \in \mathsf{Ex}_1$ is an exit, necessarily certifies termination at $d_{\boldsymbol{X}}^1$. It need not do so. For example, in the instance depicted in Figure 3 we may give the following flow on G: $\boldsymbol{x}^1 = (1,1,1)$, $\boldsymbol{x}^2 = (1,1,1)$. The instance depicted obviously loops infinitely, alternating calls between components G^1 and G^2, but neither ever reaching an exit. However, the given $(\boldsymbol{x}^1, \boldsymbol{x}^2)$ corresponds to a recursive switching flow for this instance, both of whose component switching flows have an exit as their current vertex.

We need a way to determine whether the recursive switching flow avoids such pathologies. To do this, we need some additional definitions. We describe a component switch flow $\boldsymbol{x}^l$ as *call-pending* if its current vertex $d_{\boldsymbol{x}^l}^l \in \mathsf{Call}_l$ is a call port, we let $J_{\boldsymbol{X}} \subseteq [k]$ be the set of all call-pending components and we let $r_{\boldsymbol{X}} := |J|$. From a recursive switching flow $\boldsymbol{X} := (\boldsymbol{x}^1, \dots, \boldsymbol{x}^k)$ we can compute the *pending-call graph* $C_{\boldsymbol{X}}^{\text{Pen}} := ([k], E_{\boldsymbol{X}}^{\text{Pen}})$ where we have edge $(i,j) \in E_{\boldsymbol{X}}^{\text{Pen}}$ if and only if $i \in J_{\boldsymbol{X}}$, $d_{\boldsymbol{X}}^i = (b,o) \in \mathsf{Call}_i$ is the current vertex of $\boldsymbol{x}^i$ and $j = Y_i(b)$. We can also compute the *completed-call graph*, $C_{\boldsymbol{X}}^{\text{Com}} := ([k], E_{\boldsymbol{X}}^{\text{Com}})$, where we have an edge $(i,j) \in E_{\boldsymbol{X}}^{\text{Com}}$ if and only if $\exists b \in B_i$, $\exists f \in F_{i,b}$ with $x_f^i > 0$ and $Y_i(b) = j$. The pending-call graph represents, from the perspective of an imagined "run" corresponding to the recursive switching flow $\boldsymbol{X}$, which components G^i are currently "paused" at a call port and waiting for component G^j to reach an exit to determine the return port they should move to next. The completed-call graph represents the dependencies in the calls already made in such an imagined run, where an edge from component G^i to component G^j means that inside component G^i the imagined run is making a call to a box labelled by G^j and "using" the fact that component G^j, once called upon, reaches a specific exit. In turn, in order to G^j to reach its exit the imagined run might be "using" the completion of other components to which there are outgoing edges from G^j in the completed-call graph. Thus, any cycle in the completed-call graph represents a series of circular (and hence not well-founded) assumptions about the imagined "run" corresponding to the recursive switching flow $\boldsymbol{X}$. For example, in the case of a 2-cycle between components G^i and G^j, these are: "If G^i reaches exit $d_{\boldsymbol{X}}^i$ then G^j reaches exit $d_{\boldsymbol{X}}^j$"; and "If G^j reaches exit $d_{\boldsymbol{X}}^j$ then G^i reaches exit $d_{\boldsymbol{X}}^i$" (c.f. Figure 3).

Let G be an instance of recursive arrival and let $\mathsf{RUN}^\infty(G, o_1, q_1^0) := (\beta_t, (v_t, q_t))_{t=0}^\infty$ be the run starting at (o_1, q_1^0). We define the times $S_l := \inf\{t \mid v_t = o_l\}$ and $T_l := \inf\{t \mid v_t \in \mathsf{Ex}_l\}$ for each component index $l \in [k]$, with these values being ∞ if the set is empty. If $S_l < \infty$ we define the stack $\beta^l := \beta_{S_l}$. We define the *component run* to be the (potentially finite) subsequence $t_1^l, t_2^l, \dots$ of times which are precisely all times

$t^l_j \in [S_l, \ldots, T_l]$ where $\beta_{t^j_l} = \beta^l$. We define the *Recursive Run Profile* of G up to time t as the sequence of vectors, $\boldsymbol{Run}(G,t) := (\boldsymbol{run}(G^1,t), \ldots, \boldsymbol{run}(G^k,t))$, where for each $l \in [k]$, $\boldsymbol{run}(G^l,t) := (|\{j \in \mathbb{N} \mid t^l_{j+1} \leq t \wedge (v_{t^l_j}, v_{t^l_{j+1}}) = e\}| \mid e \in E_l \cup F_l)$.

In other words, $\boldsymbol{run}(G^l,t)$ is a vector that provides counts of how many times each edge in component G^l has been crossed, up to time t, during one "visit" to component G^l, with some particular call stack. (The specific call stack doesn't matter. This sequence does not depend on the specific calling context β_l in which G^l was initially called.) We note that $\boldsymbol{run}(G^l,0) = \boldsymbol{0}^l$.

Similarly to the non-recursive case, we can define the *last-used-edge* graph for each component G^l as, $G^*_{l,\boldsymbol{x}^l} := (V_l, E^*_{l,\boldsymbol{x}^l})$ who's edge set is defined as:

$$\begin{aligned} E^*_{l,\boldsymbol{x}^l} :=& \{(v, s^0(v)) \mid v \in \mathsf{Sor}_l \text{ and } x^l_{(v,s^0(v))} \neq x^l_{(v,s^1(v))}\} \cup \\ & \{(v, s^1(v)) \mid v \in \mathsf{Sor}_l \text{ and } x^l_{(v,s^0(v))} = x^l_{(v,s^1(v))} > 0\} \cup \{f \in F_l \mid x^l_f > 0\} \end{aligned}$$

We note that for the all-zero vector we have $E^*_{l,\boldsymbol{0}^l} = \emptyset$, and if $\boldsymbol{x}^l \neq \boldsymbol{0}^l$ is non-zero then the current vertex $d^l_{\boldsymbol{x}^l}$ must have at least one incoming edge in $E^*_{l,\boldsymbol{x}^l}$, and thus the set $E^*_{l,\boldsymbol{x}^l}$ isn't empty.

Depending on how our run evolves, there are three possible cases:

- For all $l \in [k]$, if $S_l < \infty$ then $T_l < \infty$. This case corresponds to reaching some exit of G^1, i.e., terminating there.
- There exists some $l \in [k]$ with $S_l < \infty$ and yet with $T_l = \infty$, however, where for all such $l \in [k]$ the subsequence $t^l_1, t^l_2, \ldots$ is of finite length. This case corresponds to blowing up the call stack to arbitrarily large sizes, and as we shall describe, we can detect it by looking for a cycle in $C^{\text{Pen}}_{\boldsymbol{X}}$.
- There exists $l \in [k]$ with $S_l < \infty$ and $T_l = \infty$, where the subsequence $t^l_1, t^l_2, \ldots$ is of infinite length. This case corresponds to getting stuck inside component G^l, and infinitely often revisiting a vertex in a loop with the same call stack. As we shall see, we can detect this case by looking for a sufficiently large entry in some coordinate of $\boldsymbol{x}^l$.

Let G be a Recursive Arrival instance and let $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{R}$ be a recursive switching flow on G, we say $\boldsymbol{X}$ is *run-like* if it satisfies the following conditions:

- For each component index $l \in [k]$ one of the following two conditions hold:
  - The graph $G^*_{l,\boldsymbol{x}^l}$ is acyclic,
  - The graph $G^*_{l,\boldsymbol{x}^l}$ contains exactly one cycle and $d^l_{\boldsymbol{x}^l}$ is on this cycle.
- If the set of call-pending component indexes $J_{\boldsymbol{X}}$ is non-empty, then $1 \in J_{\boldsymbol{X}}$ and there is some total ordering $j_1, \ldots, j_{r_{\boldsymbol{X}}}$ of the set $J_{\boldsymbol{X}}$, with $j_1 = 1$, and a unique $j_{(r_{\boldsymbol{X}}+1)} \in [k]$ such that the edges of the pending-call graph are given by $E^{\text{Pen}}_{\boldsymbol{X}} = \{(j_i, j_{i+1}) \mid i \in [r_{\boldsymbol{X}}]\}$. Note that we may have $j_{(r_{\boldsymbol{X}}+1)} = j_m$ for some $m \in \{1, \ldots, r_{\boldsymbol{X}}\}$, in which case $E^{\text{Pen}}_{\boldsymbol{X}}$ forms not a directed line graph but a "lasso" meaning a directed line ending in one directed cycle. When $J_{\boldsymbol{X}} = \emptyset$ we say that $r_{\boldsymbol{x}} := 0$ and that $j_1 := 1$, thus the sequence is defined for all $\boldsymbol{X}$.
- For any $l \in [k]$ either: $l \in J_{\boldsymbol{X}} \cup K_{\boldsymbol{X}}$, or $\boldsymbol{x}^l = (0, \ldots, 0)$, or $l = j_{(r_{\boldsymbol{X}}+1)}$.
- The completed-call graph $C^{\text{Com}}_{\boldsymbol{X}} := ([k], E^{\text{Com}}_{\boldsymbol{X}})$ is acyclic.
- For any $l \in [k]$, if $\boldsymbol{x}^l \neq \boldsymbol{0}^l$, then in the graph $([k], E^{\text{Pen}}_{\boldsymbol{X}} \cup E^{\text{Com}}_{\boldsymbol{X}})$ we must have $1 \to^* l$, i.e., there must be a path in this graph from component 1 to all components l for which $\boldsymbol{x}^l$ is non-zero.

We denote by $\mathscr{X} \subset \mathscr{R}$ the set of all run-like recursive switching flows on G. We note for any G that we always have $\mathbf{0} \in \mathscr{X}$. We later show $\boldsymbol{X} \in \mathscr{F}$ is run-like if and only if $\exists t \in \mathbb{N}, \boldsymbol{X} = \boldsymbol{Run}(G,t)$ (Lemma C.3).

We now introduce "unit vectors" for this space, we write $\boldsymbol{u}_e^l \in \mathscr{F}_l$ for the vector where $u_e^l = 1$ and for all other $e' \in E_l \cup F_l$ with $e' \neq e$ that $u_e^l = 0$. We then write $\boldsymbol{U}_{i,e} \in \mathscr{F}$ for the sequence of k vectors $(\mathbf{0}^1, \dots, \mathbf{0}^{i-1}, \boldsymbol{u}_e^i, \mathbf{0}^{i+1}, \dots, \mathbf{0}^k)$ where the i'th vector is $\boldsymbol{u}_e^i$ and for $i \neq j \in [k]$ that the j'th vector is the all-zero $\mathbf{0}^j$. We may naturally define the notion of addition on $\mathscr{F}$ and we define the notion of subtraction $\boldsymbol{X} - \boldsymbol{U}_{i,e}$ in the natural way whenever $x_e^i > 0$, i.e., the result of the subtraction remains in $\mathbb{N}$ for every coordinate, subtraction is undefined where this isn't the case. We write $\mathscr{U} := \{\boldsymbol{U}_{i,e} \mid i \in [k], e \in E_i \cup F_i\}$ for the set of all unit vectors.

Given a run-like recursive switching flow, $\boldsymbol{X} := (\boldsymbol{x}^1, \dots, \boldsymbol{x}^k) \in \mathscr{X}$, we say that $\boldsymbol{X}$ is *complete* if it is the case that $1 \in K_{\boldsymbol{X}}$, i.e., the current vertex $d^1_{\boldsymbol{x}^1}$ of $\boldsymbol{x}^1$ is an exit of G^1. We say $\boldsymbol{X}$ is *lassoed* when $E_{\boldsymbol{X}}^{\text{Pen}}$ forms a "lasso", meaning a directed line ending in one directed cycle, as described earlier. We note that being complete and lassoed are mutually exclusive, because either $1 \in K_{\boldsymbol{X}}$ or $1 \in J_{\boldsymbol{X}}$, but not both.

Lemma 4.1. *Let G be an instance of Recursive Arrival, and let $\boldsymbol{X} \in \mathscr{X}$ be a run-like recursive switching flow on G. Then if $\boldsymbol{X}$ is neither complete nor lassoed, then there exists exactly one $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} + \boldsymbol{U}_{i,e})$ is a run-like recursive switching flow. Otherwise, if $\boldsymbol{X}$ is either complete or lassoed, then there exists no such $\boldsymbol{U}_{i,e}$.*

Proof (Sketch). We shall show that for any $\boldsymbol{X}$ which is neither complete nor lassoed, we are able to give unique i and e as a function of $\boldsymbol{X}$. Viewing $\boldsymbol{X}$ as a "hypothetical run" to some time we use $J_{\boldsymbol{X}}$ as our "call stack" at this time and use that to determine the edge to increment.

1. If $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} \in \mathsf{Sor}_{j_{(r_{\boldsymbol{X}}+1)}}$, then the "current component" is at a switching node and we take the edge given by our switching order. We note that this includes the case where $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} = o_{j_{(r_{\boldsymbol{X}}+1)}}$, i.e. there is a call pending to a new component.
2. If $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$, then we can resolve the pending call in component $j_{r_{\boldsymbol{X}}}$ and increment the summary edge in $F_{j_{r_{\boldsymbol{X}}}}$ corresponding to exit $d_{\boldsymbol{X}}^{(j_{\boldsymbol{X}}+1)}$.

We can show that this is the unique choice in these cases through elimination, making use of the definitions of component, recursive, and run-like switching flows. □

We define the *completed call count* as the function $CC : \mathscr{F} \times [k] \to \mathbb{N}$ which counts how many times a given component has been crossed in a given flow, defined for $\boldsymbol{X} \in \mathscr{F}$ and $l \in [k]$ as follows:

$$CC(\boldsymbol{X}, l) := \sum_{i \in [k]} \sum_{\{b \in B_i \mid Y_i(b) = l\}} \sum_{f \in F_{b,i}} x_f^i$$

Lemma 4.2. *Let G be an instance of Recursive Arrival, and let $\boldsymbol{X} \in \mathscr{X}$ be a run-like recursive switching flow on G. If $\boldsymbol{X}$ is non-zero then there exists a unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} - \boldsymbol{U}_{i,e}) \in \mathscr{X}$ is a run-like recursive switching flow. Otherwise, if $\boldsymbol{X}$ is all-zero, then no such $\boldsymbol{U}_{i,e}$ exists.*

Proof (Sketch). We shall show for non-zero $\boldsymbol{X}$ the following choice is the unique value for i, and then e can be determined using the last-used-edge graph in component i, as is the case for non-recursive switching flows. Viewing $\boldsymbol{X}$ as a "hypothetical run" to some time we use $J_{\boldsymbol{X}}$ as our "call stack" at this time and use that to determine the edge to decrement.

- If $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} > \mathbf{0}^{j_{(r_{\boldsymbol{X}}+1)}}$ and $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) = 0$ then we decrement inside the "current component" as the pending-call in component $j_{r_{\boldsymbol{X}}}$ is the only call made.

- Otherwise, we take $i = j_{r_{\boldsymbol{X}}}$. Where, since we have either $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) \geq 1$ or $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} = \boldsymbol{0}^{j_{(r_{\boldsymbol{X}}+1)}}$ the current call from $j_{r_{\boldsymbol{X}}}$ to $j_{(r_{\boldsymbol{X}}+1)}$ is either made elsewhere and thus we cannot alter the component flow in $j_{(r_{\boldsymbol{X}}+1)}$ without affecting the edge traversed on these other calls or the flow in $j_{(r_{\boldsymbol{X}}+1)}$ is zero, in which case we step back from the final pending-call to it.

This can be shown to be the unique choice in each case through elimination. □

We define the function $Val : \mathscr{F} \to \mathbb{N}$ as: $Val((\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k)) := \sum_{i \in [k]} \sum_{e \in E_i \cup F_i} x_e^i$. This function sums all values across all vectors of the tuple. We note that for any flow $\boldsymbol{X} \in \mathscr{F}$ and any $i \in [k]$ and $e \in E_i \cup F_i$ that we have $Val(\boldsymbol{X} + \boldsymbol{U}_{i,e}) = Val(\boldsymbol{X}) + 1$ and that when defined (i.e. $x_e^i > 0$) that $Val(\boldsymbol{X} - \boldsymbol{U}_{i,e}) = Val(\boldsymbol{X}) - 1$.

Recall Proposition 2.3 regarding non-recursive Arrival switching graphs, and in particular the fixed polynomial p which that proposition asserts the existence of. We say a recursive switching flow $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{X}$ is *finished* if it satisfies one of the following conditions:

1. $\boldsymbol{X}$ is complete, i.e, $1 \in K_{\boldsymbol{X}}$, or, the current vertex $d_{\boldsymbol{X}}^1$ of $\boldsymbol{x}^1$ is an exit in Ex_1.
2. $\boldsymbol{X}$ is lassoed, i.e., $1 \notin K_{\boldsymbol{X}}$ and $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$, or, the edges of $E_{\boldsymbol{X}}^{\text{Pen}}$ form a lasso.
3. $\boldsymbol{X}$ is *just-overflowing*, which we define as follows: $1 \notin K_{\boldsymbol{X}}$, and there exists some unique $l \in [k]$, and unique $e = (u, d_{\boldsymbol{X}}^l) \in E_l \cup F_l$ with $x_e^l = 2^{\mathrm{p}(|V_l|)} + 1$, i.e., there is some unique component, l, and edge, e, incoming to its current vertex, $d_{\boldsymbol{X}}^l$, with a "just-excessively large" value in the flow $\boldsymbol{X}$.

We say the flow is *post-overflowing* if $1 \notin K_{\boldsymbol{X}}$, and there exists some $l \in [k]$, with $d_{\boldsymbol{X}}^l$ the current-vertex of $\boldsymbol{x}^l$, and some $e = (u, v) \in E_l \cup F_l$ satisfying at least one of: A) $x_e^l = 2^{\mathrm{p}(|V_l|)} + 1$ and $v \neq d_{\boldsymbol{X}}^l$; B) $x_e^l > 2^{\mathrm{p}(|V_l|)} + 1$. We note that by repeatedly applying Lemma 4.2 to a post-overflowing run-like recursive switching flow we must eventually find some finished just-overflowing run-like recursive switching flow.

We introduce the notation $\mathscr{F}^N \subseteq \mathscr{F}$ to be the restriction to tuples in which in every vector each coordinate is less than or equal to some $N \in \mathbb{N}$. Thus $\mathscr{F}^N$ is finite, and any element $\boldsymbol{X} \in \mathscr{F}^N$ can be represented using at most $(\sum_{i=1}^k |E_i \cup F_i|) \cdot \log_2(N)$ bits. For all our subsequent results taking $N := 2^{\mathrm{p}(\max_l |V_l|)} + 1$ will be sufficient, noting this means elements of $\mathscr{F}^N$ are represented using a polynomial number of bits in our input size.

Theorem 4.3. *The* `Recursive Arrival` *problem is in* $\mathsf{NP} \cap \mathsf{coNP}$ *and* $\mathsf{UP} \cap \mathsf{coUP}$.

Proof (Sketch). The proof follows from a series of lemmas. Lemma C.4: For any instance of `Recursive Arrival`, G, there is a (unique) $\boldsymbol{X} \in \mathscr{F}^N$ which is a finished run-like recursive switching flow; Lemma C.5: Given any $\boldsymbol{X} \in \mathscr{F}^N$ we can verify whether or not $\boldsymbol{X}$ is a finished run-like recursive switching flow in P-time; Lemma C.6: Given any $\boldsymbol{X} \in \mathscr{F}^N$ which is a finished run-like recursive switching flow, we can determine whether or not G terminates and if it does terminate at which exit in Ex_1 it does so. □

4.1 Containment in UEOPL

Given the previous results, we may consider a search version of `Recursive Arrival` as follows:

`Search Recursive Arrival`	
Instance:	A Recursive Arrival graph $(G^1, \ldots, G^k)$
Problem:	Compute the unique finished run-like recursive switching flow $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}$ on G

By Lemma C.4 we know that this problem is total and hence lies in TFNP. We show containment in the total search complexity class UEOPL defined by Fearnley et al. [12], as problems polynomial time many-one search reducible to `UniqueEOPL`, which is defined as follows:

`UniqueEOPL` [12]

Instance: Given boolean circuits $S, P : \{0,1\}^n \to \{0,1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$ and a boolean circuit $V : \{0,1\}^n \to \{0,1,\ldots,2^m-1\}$ such that $V(0^n) = 0$

Problem: Compute one of the following:

(U1) A point $x \in \{0,1\}^n$ such that $P(S(x)) \neq x$.

(UV1) A point $x \in \{0,1\}^n$ such that $x \neq S(x)$, $P(S(x)) = x$, and $V(S(x)) \leq V(x)$.

(UV2) A point $x \in \{0,1\}^n$ such that $S(P(x)) \neq x \neq 0^n$.

(UV3) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < V(S(x))$.

We may interpret an instance of `UniqueEOPL` as describing an exponentially large directed graph in which our vertices are points $x \in \{0,1\}^n$ and each vertex has both in-degree and out-degree bounded by at most one. Edges are described by the circuits S, P, for a fixed vertex $x \in \{0,1\}^n$ there is an outgoing edge from x to $S(x)$ if and only if $P(S(x)) = x$ and an incoming edge to x from $P(x)$ if and only if $S(P(x)) = x$. We are given that 0^n is a point with an outgoing edge but no incoming edge or the "start of the line". We also have an "odometer" function, V, which has a minimal value at 0^n. We assume our graph has the set-up of a single line $0^n, S(0^n), S(S(0^n)), \ldots$ along which the function V strictly increases, with some "isolated points" where $x = S(x) = P(x)$. There are four types of solutions that can be returned, representing:

(U1) a point which is an "end of the line", with an incoming edge but no outgoing edge.

(UV1) a violation of the assumption that valuation V strictly increases along a line, since $V(x) \not< V(S(x))$.

(UV2) a violation of the assumption there is a single line, since x is the start of a line, but it is not 0^n, thus it starts a distinct line.

(UV3) a violation of one of the assumptions, however, in a more nuanced way. We can assume that $P(S(x)) = x$ and $P(S(y)) = y$, else they'd constitute a (UV1) example too, thus neither x nor y is isolated and both have an outgoing edge. If x and y were on the same line, then either $S(\ldots S(S(x))) = y$ or $S(\ldots S(y)) = x$ by doing this iteration we'd eventually find some $z \in \{0,1\}^n$ where $V(z) \not< V(S(z))$, violating (UV1). However, if x and y are on different lines, then that would imply the existence of two distinct lines, violating (UV2). Thus, a (UV3) violation is a short proof of existence of a (UV1) or (UV2) violation elsewhere in the instance.

For our reduction, our space will be made up of all possible flows $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}^N$ and our line will be made up of those arising from distinct $\boldsymbol{Run}(G,t)$'s, each step increasing in t until we reach a finished flow, with all other vectors being isolated. A type (U1) solution will correspond to a finished run-like recursive switching flow, and we will show our instance has no (UV1-3) solutions, thus our computed solution to `UniqueEOPL` will be a solution to `Search Recursive Arrival`.

Given any flow $\boldsymbol{X} \in \mathscr{F}$ we can verify whether or not $\boldsymbol{X}$ is a run-like recursive switching flow (i.e. $\boldsymbol{X} \in \mathscr{X} \subset \mathscr{F}$). We will use this fact in our definitions of functions $Adv : \mathscr{F} \to \mathscr{F}$ and $Prev : \mathscr{F} \to \mathscr{F}$.

Our function Adv on some value $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}$ is defined by the following sequence:

1. If $\boldsymbol{X} \notin \mathscr{X}$ then we take $Adv(\boldsymbol{X}) = \boldsymbol{X}$.
2. Else if $\boldsymbol{X} \in \mathscr{X}$ is either finished or post-overflowing then we take $Adv(\boldsymbol{X}) = \boldsymbol{X}$.
3. Otherwise, take $Adv(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{U}_{i,e}$, for the unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $\boldsymbol{X} + \boldsymbol{U}_{i,e} \in \mathscr{X}$ (Lemma 4.1).

We note by this process that if $Adv(\boldsymbol{X}) \neq \boldsymbol{X}$, then $Val(Adv(\boldsymbol{X})) = Val(\boldsymbol{X})+1$, since we have incremented exactly one edge in exactly one vector. Hence, this is consistent with our odometer. We may also define the operation $Prev : \mathscr{F} \to \mathscr{F}$ analogously on some value $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}$. Taking $Prev(\boldsymbol{X}) = \boldsymbol{X}$ whenever: $\boldsymbol{X} \notin \mathscr{X}$; $\boldsymbol{X} = \boldsymbol{0}$, or; $\boldsymbol{X}$ is post-overflowing. Otherwise, taking $Prev(\boldsymbol{X}) = \boldsymbol{X} - \boldsymbol{U}_{i,e}$, for the unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $\boldsymbol{X} - \boldsymbol{U}_{i,e} \in \mathscr{X}$ (Lemma 4.2). Observe that, for any non-zero $\boldsymbol{X} \in \mathscr{X}$, that $Adv(Prev(\boldsymbol{X})) = \boldsymbol{X}$, and, for any $\boldsymbol{X} \in \mathscr{X}$, if we have $Prev(Adv(\boldsymbol{X})) \neq \boldsymbol{X}$, then $\boldsymbol{X}$ must be finished.

Theorem 4.4. *The* `Search-Recursive Arrival` *is in* UEOPL.

Proof (Sketch). We will give a polynomial-time search reduction from `Search Recursive Arrival` to the `UniqueEOPL` problem. We compute boolean circuits S, P and V which will be given by the restriction of the functions *Adv*, *Prev*, and *Val* to the domain $\mathscr{F}^N$. This process involves computing membership of $\mathscr{X}$ and then computing the unique values i and e given by Lemmas 4.1 and 4.2 for *Adv* and *Prev* respectively. We can then show using Lemma C.4 that the only UEOPL solution is of type (U1) and is a run-like recursive switching flow, which is a solution we are looking for. □

5 Conclusions

We have shown that `Recursive Arrival` is contained in many of the same classes as the standard `Arrival` problem. While we have shown P-hardness for `Recursive Arrival`, whether or not `Arrival` is P-hard remains open.

Let us note that the way we have chosen to generalise Arrival to the recursive setting uses one of two possible natural choices for its semantics. Namely, it assumes a “local” semantics, meaning that the current switch position for each component on the call stack is maintained as part of the current state. An alternative “global” semantics would instead consider the switch position of each component as a “global variable”. In such a model all switch positions would start in an initial position, and as the run progresses the switch positions would persist between, and be updated during, different calls to the same component. It is possible to show (a result we have not included in this paper) that such a “global" formulation immediately results in PSPACE-hardness of reachability and termination problems.

As mentioned in the introduction, a stochastic version of `Arrival`, in which some nodes are switching nodes whereas other nodes are chance (probabilistic) nodes with probabilities on outgoing transitions, has already been studied in [19], building on the work of [12] which generalises `Arrival` by allowing switching and player-controlled nodes. There is extensive prior work on RMCs and RMDPs, with many known decidability/complexity results (see, e.g., [9, 10]). It would be natural to ask similar computational questions for the generalisation of RMCs and RMDPs to a recursive Arrival model combining switching nodes with chance (probabilistic) nodes and controlled/player nodes.

Finally, we note that Fearnley et al. also defined a P-hard generalisation of `Arrival` in [12] which uses “succinct switching orders” to succinctly encode an exponentially larger switch graph. We will refer to this problem as `Succinct Arrival`. We don’t know whether there are any P-time reduction, in either direction, between `Recursive Arrival` and `Succinct Arrival`. It has been observed[1] that the results of [14] imply that both `Arrival` and `Succinct Arrival` are P-time reducible to the `Tarski` problem defined in [7]. `Succinct Arrival` is also contained in UEOPL by the same arguments as for `Arrival`. We do not currently know whether `Recursive Arrival` is P-time reducible to `Tarski`.

Acknowledgements. Thanks to my PhD supervisor Kousha Etessami for his support.

[1] Personal communication from Kousha Etessami and Mihalis Yannakakis.

References

[1] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2001): *Analysis of Recursive State Machines*. In Gérard Berry, Hubert Comon & Alain Finkel, editors: *Computer Aided Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 207–220, doi:10.5555/647770.734260.

[2] David Auger, Pierre Coucheney & Loric Duhaze (2022): *Polynomial Time Algorithm for ARRIVAL on Tree-like Multigraphs*. In: *International Symposium on Mathematical Foundations of Computer Science*, doi:10.48550/arXiv.2204.13151.

[3] Ahmed Bouajjani, Javier Esparza & Oded Maler (1997): *Reachability analysis of pushdown automata: Application to model-checking*. In Antoni Mazurkiewicz & Józef Winkowski, editors: *CONCUR '97: Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 135–150, doi:10.1007/3-540-63141-0_10.

[4] Anne Condon (1992): *The Complexity of Stochastic Games*. *Inf. Comput.* 96(2), pp. 203–224, doi:10.1016/0890-5401(92)90048-K.

[5] Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jíri Matoušek & Emo Welzl (2017): *A Journey through Discrete Mathematics: A Tribute to Jiri Matousek*, chapter Arrival: A zero-player graph game in $\mathsf{NP} \cap \mathsf{coNP}$, pp. 367–374. Springer, doi:10.1007/978-3-319-44479-6_14.

[6] Javier Esparza, Antonin Kucera & Richard Mayr (2006): *Model Checking Probabilistic Pushdown Automata*. *Logical Methods in Computer Science* Volume 2, Issue 1, doi:10.2168/LMCS-2(1:2)2006.

[7] Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein & Mihalis Yannakakis (2020): *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*. In Thomas Vidick, editor: *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, *LIPIcs* 151, pp. 18:1–18:19, doi:10.4230/LIPIcs.ITCS.2020.18.

[8] Kousha Etessami & Mihalis Yannakakis (2008): *Recursive Concurrent Stochastic Games*. *CoRR* abs/0810.3581, doi:10.48550/arXiv.0810.3581. arXiv:0810.3581.

[9] Kousha Etessami & Mihalis Yannakakis (2009): *Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations*. *J. ACM* 56(1), pp. 1:1–1:66, doi:10.1145/1462153.1462154.

[10] Kousha Etessami & Mihalis Yannakakis (2015): *Recursive Markov Decision Processes and Recursive Stochastic Games*. *J. ACM* 62(2), pp. 11:1–11:69, doi:10.1145/2699431.

[11] John Fearnley, Martin Gairing, Matthias Mnich & Rahul Savani (2021): *Reachability Switching Games*. *Log. Methods Comput. Sci.* 17(2), doi:10.23638/LMCS-17(2:10)2021.

[12] John Fearnley, Spencer Gordon, Ruta Mehta & Rahul Savani (2019): *Unique End of Potential Line*. In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, *LIPIcs* 132, pp. 56:1–56:15, doi:10.4230/LIPIcs.ICALP.2019.56.

[13] Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubácek, Karel Král, Hagar Mosaad & Veronika Slívová (2018): *ARRIVAL: Next Stop in CLS*. In: *45th International Colloquium on Automata, Languages, and Programming*, *LIPIcs* 107, pp. 60:1–60:13, doi:10.4230/LIPIcs.ICALP.2018.60.

[14] Bernd Gärtner, Sebastian Haslebacher & Hung P. Hoang (2021): *A Subexponential Algorithm for ARRIVAL*. In: *48th International Colloquium on Automata, Languages, and Programming*, *LIPIcs* 198, pp. 69:1–69:14, doi:10.4230/LIPIcs.ICALP.2021.69.

[15] Raymond Greenlaw, H. James Hoover & Walter L. Ruzzo (1995): *Limits to Parallel Computation*. Oxford University Press, doi:10.1093/oso/9780195085914.001.0001.

[16] Marcin Jurdzinski (1998): *Deciding the Winner in Parity Games is in* $\mathsf{UP} \cap \mathsf{coUP}$. *Inf. Process. Lett.* 68(3), pp. 119–124, doi:10.1016/S0020-0190(98)00150-1.

[17] C. S. Karthik (2017): *Did the train reach its destination: The complexity of finding a witness*. *Information Processing Letters* 121, pp. 17–21, doi:10.1016/j.ip1.2017.01.004.

[18] Graham Manuell (2021): *A simple lower bound for ARRIVAL*. *CoRR* abs/2108.06273, doi:10.48550/arXiv.2108.06273.

[19] Thomas Webster (2022): *The Stochastic Arrival Problem*. In Anthony W. Lin, Georg Zetzsche & Igor Potapov, editors: *Reachability Problems*, Springer, pp. 93–107, doi:https://doi.org/10.1007/978-3-031-19135-0_7.

[20] Uri Zwick & Mike Paterson (1996): *The Complexity of Mean Payoff Games on Graphs*. *Theor. Comput. Sci.* 158(1&2), pp. 343–359, doi:10.1016/0304-3975(95)00188-3.

A Missing proofs for Section 2

The following results follow completely from prior work. The proofs are included for completeness to account for the slight modifications made to definitions and statements in this work.

Proposition 2.3 ([5, 13])**.** *There exists a polynomial function* $\mathrm{p} : \mathbb{N} \to \mathbb{N}$ *such that for all Switch Graphs* $G := (V, s^0, s^1)$ *and all vertices* $o, d \in V$ *with* $o \neq d$ *and* $d \in \mathsf{DE}$ *the following are equivalent:*

- *The run on* G *from initial state* (o, q^0) *terminates at* d*.*
- *There exists a run-like switching flow* $\boldsymbol{x}$ *on* G *from* o *to* d *satisfying* $\forall e \in E$*, that* $\log_2(x_e) \leq \mathrm{p}(|G|)$*.*

Furthermore, for the same polynomial p*, the following are equivalent:*

- *The run on* G *from initial state* (o, q^0) *does not terminate.*
- *There exists a vertex* $d' \in V \setminus \mathsf{DE}$*, a run-like switching flow* $\boldsymbol{x}$ *on* G *from* o *to* d'*, and an edge* $e' = (u, d') \in E$ *which satisfies for all* $e \in E \setminus \{e'\}$ *that* $\log_2(x_e) \leq \mathrm{p}(|G|)$ *and that* $x_{e'} = 2^{\mathrm{p}(|G|)} + 1$*.*

Proof. We may take p as any polynomial satisfying $\mathrm{p}(|G|) \geq \log_2(|V|) \cdot |V|$, for instance, $\mathrm{p}(|G|) = |V|^2$. For the first pair of equivalences:

- If the run on G from initial state (o, q^0) terminates at d then the termination time $T < \infty$. We take the run profile $\boldsymbol{run}(o, T)$ up to the termination time T. By [5, Observation 1] the run profile is a switching flow on G from o to d, and, by [5, Theorem 2], this has for all $e \in E$ that $\mathsf{run}(o, T)_e \leq |V| \cdot 2^{|V|}$. Such a flow is also obviously run-like. Hence, $\boldsymbol{run}(o, T)$ is a switching flow as required.
- Assume that there exists some (run-like) switching flow $\boldsymbol{x}$ on G from o to d. By [5, Lemma 1], any switching flow provides an upper bound on the run profile to termination $\boldsymbol{run}(o, T)$, thus we must have $T < \infty$. We note by the definition of DE that for all edges $(v, d') \in E$ with $v \in V$, $d' \in \mathsf{DE}$, and $d' \neq d$, we must have $\mathsf{run}(o, T)_{(v, d')} = 0$ in any switching flow on G from o to d, by flow conservation. Thus, in any run profile, we can not traverse any such edge. Hence, the run can not terminate at d' (as we would need to traverse such an edge) and, since G terminates at some vertex in DE, it must be d.

For the second pair:

- Assume that the run on G from initial state (o, q^0) does not terminate. Consider the run profiles to times $t \in [T]$ given by $\boldsymbol{run}(o, t)$. By [5, Observation 1] we observe the partial-run profile is a switching flow to some current vertex v_t at every time t. We also note that the functions monotonically increase as follows; for any t we have $\forall e \in E \setminus \{(v_t, v_{t+1})\}$ that $\mathsf{run}(o, t)_e = \mathsf{run}(o, t+1)_e$ and that $\mathsf{run}(o, t+1)_{(v_t, v_{t+1})} = \mathsf{run}(o, t)_{(v_t, v_{t+1})} + 1$. Since G does not terminate $T = \infty$. Thus, since the sequence $\boldsymbol{run}(o, t)$ increases monotonically by 1 in exactly one coordinate, there exists some $t \in [T]$ and $e \in E$ at which $\mathsf{run}(o, t)_e = 2^{P(|G|)} + 1$ and we may consider the minimum value t' over all such t. We note that $\boldsymbol{run}(o, t')$ is then such a run-like switching flow taking $e' = (v_{t'-1}, v_{t'})$ and we must have $v_{t'} \notin \mathsf{DE}$, else this would contradict our non-termination assumption.

- Assuming there exists some run-like switching flow $\boldsymbol{x}$ on G to some vertex $d' \in V \setminus \mathsf{DE}$, then there is some $t \in \mathbb{N}$ such that $\boldsymbol{x} = \boldsymbol{run}(o,t)$. We consider the sequence of states $\gamma_0, \ldots, \gamma_t$ visited along $\mathsf{RUN}^\infty(G,(o,q^0))$, where $\gamma_i := (v_i, q_i)$. We consider the subsequence $q_{j_1}, \ldots, q_{j_k}$ consisting of all values $j_i \in [0 \ldots t]$ such that $v_{j_i} = u$. We know that $k \geq x_{e'} = 2^{\mathrm{p}(|G|)} + 1$, since we make one visit to u for every traversal of edge e'. We know each $q_{j_i} \in Q$ and that $|Q| = 2^{|V|} \leq 2^{\mathrm{p}(|G|)}$ by our choice of p. Thus, by the pigeonhole principle, we can find two indices $j, j' \in [0 \ldots t]$ with $j < j'$ from the subsequence of j_i's such that $v_j = v_{j'} = u$ and that $q_j = q_{j'}$. Since we are working in a deterministic state transition system, and, we have found a transition from a state to itself, the system obviously cycles infinitely. We can see evidently since $u, d' \notin \mathsf{DE}$ that we can not visit any vertex $d \in \mathsf{DE}$ since we must visit u infinitely often, and we can never visit u after visiting any such d.

□

Proposition 2.4 ([13])**.** *Let $G := (V, s^0, s^1)$ be a Switch Graph and let $\boldsymbol{x}$ be a switching flow on G from $o \in V$ to $d \in V$, then there exists a unique $t \in \mathbb{N}$ such that $\boldsymbol{x} = \boldsymbol{run}(o,t)$, if and only if one of the following two (mutually exclusive) conditions hold:*

- *The graph $G^*_{\boldsymbol{x}}$ is acyclic,*
- *The graph $G^*_{\boldsymbol{x}}$ contains exactly one cycle and d is on this cycle,*

Furthermore, given G and any such $\boldsymbol{x}$ whether or not one of these conditions hold can be checked in polynomial time in the size of G and $\boldsymbol{x}$.

Proof. This follows exactly from the results of Gartner et al. [13] cf. Observation 8, Lemma 9, Observation 10 and Lemma 11.

As observed by [5] any switching flow that is not run-like looks like the sum of a run-like switching flow and a series of (possibly multiple) "circulations". We can determine the last such circulation in the series by looking at the cycles in the last-used-edge graph, which will contain the edges along this circulation. However, we must distinguish the case where the circulation could be a part of the run, which occurs only if the current vertex is on the cycle ([13, Observation 10]). Together, these lead to the conditions as shown in [13, Lemma 9]. With these conditions, we can see we may evidently compute the graph $G^*_{\boldsymbol{x}}$ in polynomial time by determining, for each vertex, which (if any) of its outgoing edges satisfy the condition ([13, Observation 8]). We can then see, by [13, Lemma 11], this is verifiable in polynomial time. □

Proposition 2.5 ([13, Lemma 16])**.** *Let $G := (V, s^0, s^1)$ be a Switch Graph and let $t \in \mathbb{N}$ with $\boldsymbol{run}(o,t)$ the run profile up to time t, which is a switching flow on G from $o \in V$ to some vertex $d \in V$. Then at least one of the following two conditions hold:*

- *There is a unique edge $(u,d) \in E^*_{\boldsymbol{run}(o,t)}$ incoming to d in the graph $G^*_{\boldsymbol{run}(o,t)}$.*
- *The graph $G^*_{\boldsymbol{run}(o,t)}$ contains exactly one cycle, and that cycle contains exactly one edge of the form $(u,d) \in E^*_{\boldsymbol{run}(o,t)}$ on the cycle.*

Moreover, the edge (u,d) was the edge traversed at time t in the run (i.e., if $\mathsf{RUN}^\infty(G,(o,q^0)) = ((v_i,q_i))_{i=0}^\infty$ then $v_{t-1} = u$ and $v_t = d$). Furthermore, this uniquely determined edge can be computed given G and $\boldsymbol{run}(o,t)$ in time polynomial in the size of G and $\boldsymbol{run}(o,t)$.

Proof. We may obviously compute the edge following the verification of the conditions, as in the proof of Proposition 2.4. It follows by [13, Lemma 16] that this edge is the one used. □

A.1 Recursive Arrival

Lemma A.1. *Let $G := (G^1,\dots,G^k)$ be an instance of Recursive Arrival, and assume the run on G hits some state $(\beta,(v,q))$. If there exists indices $1 \le i < i' \le |\beta|+1$ of the component call stack of β with $c_i = c_{i'}$, then the run on G does not terminate (at any exit of G^1).*

Proof. Suppose $\beta = ((b_1,q_1)\dots(b_r,q_r))$. We are working with a deterministic transition system. We let $j = c_i = c_{i'}$. We then claim that, for any $\beta' \in \Gamma_S$, and for any well-formed state $(\beta',(o_j,q_j^0)) \in \Gamma$, there is a run starting from $(\beta',(o_j,q_j^0))$ which hits the state $(\beta'(b_{i+1},q_{i+1})\dots(b_{i'},q_{i'}),(o_j,q_j^0))$, and furthermore along this run β' is always a prefix of the stack, or equivalently, the size of the stack is bounded below by $|\beta'|$. We also claim that since state $(\beta,(v,q))$ is reached by assumption in the run on G starting at $(\varepsilon,(o_1,q_1^0))$, the state $((b_1,q_1)(b_2,q_2)\dots(b_i,q_i),(o_j,q_j^0))$ is also reached in the same run. Thus taking $\beta' = (b_1,q_1)\dots(b_i,q_i)$ we can prove inductively that for all $l \in \mathbb{N}$ the run on G starting from $(\varepsilon,(o_1,q_1^0))$ reaches $(\beta'[(b_{i+1},q_{i+1})\dots(b_{i'},q_{i'})]^l,(o_j,q_j^0))$, and thereafter $\beta'[(b_{i+1},q_{i+1})\dots(b_{i'},q_{i'})]^l$ remains a prefix of the call stack. Hence the stack size not only remains bounded away from zero, but actually becomes unbounded along the run. Thus the run cannot terminate. ☐

Lemma 2.7. *Let $G := (G^1,\dots,G^k)$ be an instance of Recursive Arrival and assume the run on G hits some state $(\beta,(v,q))$, with $|\beta| \ge k$. Then the run on G does not terminate.*

Proof. We know by the assumption that the size of the component call stack of β satisfies $|\beta|+1 \ge k+1$. Thus, by the pigeonhole principle, since the component call stack $c_1 \dots c_{|\beta|+1}$ is a sequence over $[k]$, there must be two distinct indices $i < i'$ where $c_i = c_{i'}$. Thus, the result follows by Lemma A.1. ☐

B Missing proofs for Section 3

Theorem 3.1. *The* `2-exit Recursive Arrival` *problem is* P*-hard.*

Proof. We proceed to show this by reduction from the Monotone Circuit Value Problem which is P-complete (e.g., [15]). We assume that we are given as input an encoding of a boolean circuit α, as a straight-line program, consisting of AND and OR gates and two constant gates corresponding to values "true" and "false", plus a designated output gate y. The problem is to decide whether or not the output y of α is true. We may assume that the gates of α are $g_1,\dots,g_n$ in this ordering, i.e., for $i,j \in [n]$ if $j \ge i$ then the output of gate g_j is not an input to gate g_i, further we may assume $y = g_n$, that $g_1 = “true''$ and $g_2 = “false''$ are our constant gates, and that all other gates are either the AND or the OR of exactly two inputs.

Our reduction works as follows: given such a boolean circuit, α we will construct a recursive arrival instance, G, such that G has n components, with, for $i \in [n]$, a component G^i corresponding to the gate g_i of the circuit α.

Each component G^i will have a single entry, o_i, and two exits, $\top_i$ and $\bot_i$. We will refer to the two exits as "top", $\top$, and "bottom", $\bot$, and they are located accordingly in our figures. Thus for any box $b \in B_i$ with $Y_i(b) = j$ its entry port will always be (b,o_j) and it's return ports always $(b,\top_j)$ and $(b,\bot_j)$. We will view these exits as encoding the outputs, "true" and "false" respectively, from the component G^i.

We will establish that for each component G^i, the exit $\top_i$ is reached exactly when the output of the corresponding gate g_i is "true" and that the exit $\bot_i$ is reached exactly when the output of the corresponding gate is "false". We do so by induction on the index $i \in [n]$, using our topological ordering of our boolean circuit α.

Firstly, we construct components G^1 and G^2 for the constant gates g_1 and g_2, which form the base case of our induction. These components are depicted in Figure 1. It is obvious that these satisfy the inductive hypothesis, as all transitions go directly to the correct exit.

We now take our strong inductive hypothesis at the i'th step, $i \in [n-1]$. We assume there exist components $G^1, \ldots, G^{i-1}$ such that for all $j \in [i-1]$ component G^j reaches the exit $\top_j$ if gate g_j's output is "true" and otherwise reach the exit $\bot_j$. We now construct a component G^i corresponding to gate g_i. We know g_i has exactly two inputs from gates $j, j' \in [i-1]$. By induction, we can construct components G^j and $G^{j'}$ which will be used inside G^i. We depict in Figure 2 two cases corresponding to whether g_i is an AND or an OR gate:

- AND Gate - The component G^i, depicted in Figure 2a, can be used to compute the AND. We can observe that to hit, $\top_i$ we must hit both $(b_2, \top_{j'})$ and $(b_1, \top_j)$. By induction, this happens when the output of both g_j and $g_{j'}$ is "true", thus the output of g_i should also be "true". Also by induction, we must hit the return ports of both b_1 and b_2, and we observe hitting either $(b_2, \bot_{j'})$ or $(b_1, \bot_j)$ results in hitting exit $\bot_\wedge$, this happens when the output of one of g_j or $g_{j'}$ is "false" and thus the output of g_{i+1} should also be "false".
- OR Gate - The component $\mathrm{OR}_{j,j'}$, depicted in Figure 2b, can be used to compute the OR. The proof of which follows analogously to that for AND.

One can see that these components perform a lazy evaluation over the respective gate inputs. We note that the size of each G^i is constant (independent of i or n). Following this process, we eventually construct components $G^1, \ldots, G^n$. By our induction, G^n reaches exit $\top_n$ if and only if the output from $y = g_n$ is "true".

Thus, we have constructed a `Recursive Arrival` instance G, of polynomial size in n, the number of gates of α, and shown a many-one reduction hence the `Recursive Arrival` problem is P-hard. $\square$

C Missing proofs for Section 4

Lemma C.1. *Let G be an instance of Recursive Arrival. For any $t \in \mathbb{N}$ the Recursive Run Profile $\boldsymbol{Run}(G,t)$ is a run-like recursive switching flow, i.e., $\{\boldsymbol{Run}(G,t) \mid t \in \mathbb{N}\} \subseteq \mathscr{X}$.*

Proof. This fact follows trivially by induction. For $t = 0$, the recursive run profile consists of only all-zero vectors, so trivially satisfies the conditions. Assuming for some time t, we note if there is a change in the recursive run profile at time $t+1$, then it must be in some unique component G^l, for $l \in [k]$, some unique edge e's coordinate has been incremented by 1. In this case, we must have $1 \in J$ and that $l = j_r$ or $l = j_{r+1}$. It also follows as the subsequences just look like regular Arrival runs where we wait at call ports until things are completed. $\square$

Lemma 4.1. *Let G be an instance of Recursive Arrival, and let $\boldsymbol{X} \in \mathscr{X}$ be a run-like recursive switching flow on G. Then if $\boldsymbol{X}$ is neither complete nor lassoed, then there exists exactly one $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} + \boldsymbol{U}_{i,e})$ is a run-like recursive switching flow. Otherwise, if $\boldsymbol{X}$ is either complete or lassoed, then there exists no such $\boldsymbol{U}_{i,e}$.*

Proof. We shall show that for any $\boldsymbol{X}$ the following cases are total and mutually exclusive and from each case we have that $\boldsymbol{X}$ is either complete, lassoed or we are able to give unique i and e as a function of $\boldsymbol{X}$.

1. $\boldsymbol{X}$ is complete, i.e., $1 \in K_{\boldsymbol{X}}$.
2. $\boldsymbol{X}$ is lassoed, i.e., $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$.
3. $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} \in \mathsf{Sor}_{j_{(r_{\boldsymbol{X}}+1)}}$. In which case we shall show that $i = j_{(r_{\boldsymbol{X}}+1)}$ and e can be determined by checking the switch parity condition at $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}}$.
4. If $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$. In which case we shall show that $i = j_{r_{\boldsymbol{X}}}$ and e can be determined by vertex $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} \in \mathsf{Ex}_{j_{(r_{\boldsymbol{X}}+1)}}$.

We note these cases are mutually exclusive, as exactly one of $1 \in K_{\boldsymbol{X}}$ or $1 \in J_{\boldsymbol{X}}$ holds. Also, exactly one of $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$, $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$ or $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} \in \mathsf{Sor}_{j_{(r_{\boldsymbol{X}}+1)}}$ holds as $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}}$ must be exactly one of an exit, call-port or source vertex (including possibly an entry, return-port or switching node).

Consider any run-like recursive switching flow $\boldsymbol{X}$ on G. For any $i \in [k]$ and $e \in E_i \cup F_i$, define $\boldsymbol{X}_{i,e} := (\boldsymbol{X} + \boldsymbol{U}_{i,e}) = (\boldsymbol{x}_{i,e}^1, \ldots, \boldsymbol{x}_{i,e}^k)$.

In any component $i \in [k]$, the only edges $e \in E_i \cup F_i$, such that $\boldsymbol{x}^i + \boldsymbol{u}_e^i$ is a component switching flow, are those edges e outgoing from $d_{\boldsymbol{X}}^i$ or where e is a self-loop edge $e = (v,v) \in E_i$. Thus, for any other e, $\boldsymbol{X}_{i,e}$ is not a recursive switching flow, because its i'th component is not a component switching flow on G^i.

If it were the case that $e = (v,v)$ is a self-loop, then $\boldsymbol{X}_{i,e}$ the last-used-edge graph, $G^*_{i,\boldsymbol{x}_{i,e}^i}$, must contain a cycle consisting of the edge (v,v). If $v \neq d_{\boldsymbol{X}}^i$, then $\boldsymbol{X}_{i,e}$ cannot be run-like because, by definition of being run-like, $d_{\boldsymbol{X}}^i$ must be on any cycle in $G^*_{i,\boldsymbol{x}_{i,e}^i}$. Thus, we only need to consider $\boldsymbol{X}_{i,e}$ where e is an outgoing edge of $d_{\boldsymbol{X}}^i$. There are three cases to consider, based on whether $d_{\boldsymbol{X}}^i$ is an exit, call-port, or source vertex:

- If $d_{\boldsymbol{X}}^i \in \mathsf{Ex}_i$, i.e. $i \in K_{\boldsymbol{X}}$, then there are no outgoing edges e from $d_{\boldsymbol{X}}^i$. Thus, for this i, there does not exist any edge $e \in E_i \cup F_i$ such that $\boldsymbol{X}_{i,e}$ is a run-like recursive switching flow.
- If $d_{\boldsymbol{X}}^i \in \mathsf{Sor}_i$. Then there are two outgoing edges of $d_{\boldsymbol{X}}^i \in \mathsf{Sor}_i$ which we call $e_0 := (d_{\boldsymbol{X}}^i, s^0(d_{\boldsymbol{X}}^i))$ and $e_1 := (d_{\boldsymbol{X}}^i, s^1(d_{\boldsymbol{X}}^i))$. By the Switching Parity Condition $\boldsymbol{x}_{i,e_0}^i$ is a component switching flow if and only if $x_{e_0}^i = x_{e_1}^i$. Similarly, $\boldsymbol{x}_{i,e_1}^i$ is a component switching flow if and only if $x_{e_0}^i \neq x_{e_1}^i$. Since these are mutually exclusive, exactly one of $\boldsymbol{x}_{i,e_0}^i$ and $\boldsymbol{x}_{i,e_1}^i$ is a component switching flow. Thus, for such an i, there is some unique e such that $\boldsymbol{X}_{i,e}$ is a recursive (not necs. run-like) switching flow.

  Now, considering this $\boldsymbol{X}_{i,e}$, we determine for which choices of i, with $d_{\boldsymbol{X}}^i \in \mathsf{Sor}_i$, that $\boldsymbol{X}_{i,e}$ can be run-like. Note that, $K_{\boldsymbol{X}} = K_{\boldsymbol{X}_{i,e}}$ and $J_{\boldsymbol{X}} = J_{\boldsymbol{X}_{i,e}}$. Thus, since $\boldsymbol{x}_{i,e}^i \neq \boldsymbol{0}^i$, we must have $i = j_{r_{\boldsymbol{X}_{i,e}}+1} = j_{(r_{\boldsymbol{X}}+1)}$ for $\boldsymbol{X}_{i,e}$ to be run-like.
- If $d_{\boldsymbol{X}}^i \in \mathsf{Call}_i$, i.e., $i \in J_{\boldsymbol{X}}$ and $d_{\boldsymbol{X}}^i = (b, o_{Y_i(b)})$. Then all outgoing edges of $d_{\boldsymbol{X}}^i$ are $e \in F_{i,b}$. In $\boldsymbol{x}_{i,e}^i$ we must have that the e'th coordinate is strictly positive, thus where $\boldsymbol{X}_{i,e}$ run-like, we must have that $Y_i(b) \in K_{\boldsymbol{X}_{i,e}}$. It must further be the case that $K_{\boldsymbol{X}_{i,e}} = K_{\boldsymbol{X}}$ and that $d_{\boldsymbol{X}}^{Y_i(b)} = d_{\boldsymbol{X}_{i,e}}^{Y_i(b)} \in \mathsf{Ex}_{Y_i(b)}$, since the only component switching flow that has changed is that for component i and its new current vertex, $d_{\boldsymbol{X}_{i,e}}^i$, is a return-port of box b. Thus, for such an i, there is some unique e, namely $((b, o_{Y_i(b)}), (b, d_{\boldsymbol{X}}^{Y_i(b)}))$, for which $\boldsymbol{X}_{i,e}$ is a recursive (not necs. run-like) switching flow.

  Now, considering this $\boldsymbol{X}_{i,e}$, we determine for which i, with $i \in J_{\boldsymbol{X}}$, it can be run-like. We note $K_{\boldsymbol{X}}$ and $J_{\boldsymbol{X}}$ are disjoint and for the ordering $j_1, \ldots, j_{r_{\boldsymbol{X}}}$ of $J_{\boldsymbol{X}}$ and where $d_{\boldsymbol{X}}^{j_k} = (b_k, o)$ that for all $k \in [r_{\boldsymbol{X}}]$ $Y_{j_k}(b_k) = j_{k+1}$. Since $i = j_k$ for some $k \in [r_{\boldsymbol{X}}]$ it must be that $b = b_k$ and $Y_i(b) = j_{k+1} \in K_{\boldsymbol{X}}$

and $j_{k+1} \notin J_{\boldsymbol{X}}$. Hence, $k = r_{\boldsymbol{X}}$ as this is the only $k \in [r_{\boldsymbol{X}}]$ where this can be true. Further, by our conditions on component flows $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}_{i,e}} = K_{\boldsymbol{X}}$.

From these necessary conditions on i and e for $\boldsymbol{X}_{i,e}$ to be run-like we can examine the cases when $\boldsymbol{X}$ is complete or lassoed.

- If $\boldsymbol{X}$ is complete, then for each component, $i \in [k]$, either $i \in K_{\boldsymbol{X}}$ or $d^i_{\boldsymbol{X}} = o_i \in \mathsf{Sor}_i$. We also note that $r_{\boldsymbol{X}} = 0$ and $j_{(r_{\boldsymbol{X}}+1)} = 1 \in K_{\boldsymbol{X}}$. It follows from the above, no $\boldsymbol{X}_{i,e}$ can be run-like. It cannot be the case that $i \in K_{\boldsymbol{X}}$ as no such e exists where $\boldsymbol{X}_{i,e}$ is run-like. It also cannot be the case that both $i \notin K_{\boldsymbol{X}}$ and $d^i_{\boldsymbol{X}} = o_i \in \mathsf{Sor}_i$, as we have shown for $\boldsymbol{X}_{i,e}$ run-like it is necessary for $i = j_{(r_{\boldsymbol{X}}+1)} = 1$, however, this implies $i \in K_{\boldsymbol{X}}$, a contradiction. Thus, no $i \in [k]$ leads to a run-like flow.
- $\boldsymbol{X}$ is lassoed, then for each component, $i \in [k]$, either $i \in K_{\boldsymbol{X}}$, $i \in J_{\boldsymbol{X}}$ or $d^i_{\boldsymbol{X}} = o_i \in \mathsf{Sor}_i$. Since $\boldsymbol{X}$ is lassoed, we know that $j_{r_{\boldsymbol{X}}} \in J_{\boldsymbol{X}}$ and thus $j_{r_{\boldsymbol{X}}} \notin K_{\boldsymbol{X}}$. It follows from the above, no $\boldsymbol{X}_{i,e}$ can be run-like. It cannot be the case that $i \in K_{\boldsymbol{X}}$ as no such e exists where $\boldsymbol{X}_{i,e}$ is run-like. It also cannot be the case that both $i \in J_{\boldsymbol{X}}$, as we have shown for $\boldsymbol{X}_{i,e}$ run-like it is necessary for $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$, but since $\boldsymbol{X}$ is lassoed $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$ and $J_{\boldsymbol{X}}$ and $K_{\boldsymbol{X}}$ are disjoint, a contradiction. It also cannot be the case that both $i \notin K_{\boldsymbol{X}} \cup J_{\boldsymbol{X}}$ and $d^i_{\boldsymbol{X}} = o_i \in \mathsf{Sor}_i$, as we have shown for $\boldsymbol{X}_{i,e}$ run-like it is necessary for $i = j_{(r_{\boldsymbol{X}}+1)}$, however, this implies $i \in J_{\boldsymbol{X}}$, a contradiction.

Thus the only remaining case is that in which $\boldsymbol{X}$ is neither complete nor lassoed in which we wish to show i and e are uniquely determined. We shall use the above necessary conditions for $\boldsymbol{X}_{i,e}$ to be run-like to show at most one pair exists, then verify this pair is in fact run-like. For $\boldsymbol{X}_{i,e}$ to be run-like it must be that $i \notin K_{\boldsymbol{X}}$ and one of:

- $d^i_{\boldsymbol{X}} \in \mathsf{Sor}_i$ and $i = j_{(r_{\boldsymbol{X}}+1)}$ (our initial third case), or,
- $i \in J_{\boldsymbol{X}}$, $i = j_{r_{\boldsymbol{X}}}$ and $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$ (our initial fourth case)

As argued initially these are mutually exclusive and any $\boldsymbol{X}$ which is neither complete nor lassoed (cases 1 & 2) satisfies exactly one of these cases. Thus, there is at most one i and e for such $\boldsymbol{X}$. We check that for whichever case $\boldsymbol{X}$ satisfies, the resulting $\boldsymbol{X}_{i,e}$ is actually run-like. The edge e was chosen so that $\boldsymbol{x}^i_{i,e}$ was a component switching flow and $\boldsymbol{X}_{i,e}$ was a recursive switching flow. Thus we check each item of our run-like definition.

- For each component index $l \in [k] \setminus \{i\}$ the graph $G^*_{l,\boldsymbol{x}^l_{i,e}} = G^*_{l,\boldsymbol{x}^l}$. Thus, we only need to confirm that the condition holds for $G^*_{l,\boldsymbol{x}^i_{i,e}}$. If $e \in E_l$ this follows by our non-recursive result Proposition 2.5. Otherwise, if $e = ((b,o),(b,x)) \in F_l$ it follows as if $G^*_{l,\boldsymbol{x}^i_{i,e}}$ contained a cycle with (b,o) on the cycle this cycle must also contain the edge e and if adding e forms a cycle then there must already be some outgoing edge from (b,x) in $G^*_{l,\boldsymbol{x}^i}$, however then e must also be in $G^*_{l,\boldsymbol{x}^i}$, thus this cannot be a new cycle.
- Here we distinguish between cases 3 and 4. In case 3: there is no change to $J_{\boldsymbol{X}}$ and the ordering remains the same. In case 4: the value $r_{\boldsymbol{X}_{i,e}} = r_{\boldsymbol{X}} - 1$ as $j_{r_{\boldsymbol{X}}} \notin J_{\boldsymbol{X}_{i,e}}$, the ordering remains the same however, as it is just the maximal element removed.
- Here we distinguish between cases 3 and 4. In case 3: we only increase the value in component $j_{(r_{\boldsymbol{X}}+1)} = j_{r_{\boldsymbol{X}_{i,e}}+1}$, thus never violating the condition. In case 4: it must be the case $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}} = K_{\boldsymbol{X}_{i,e}}$, thus while $j_{r_{\boldsymbol{X}}} \notin J_{\boldsymbol{X}_{i,e}}$ we have $j_{r_{\boldsymbol{X}}} = j_{r_{\boldsymbol{X}_{i,e}}+1}$.
- We can not introduce a cycle into $C^{\mathrm{Com}}_{\boldsymbol{X}'}$ as if adding edge (i,j) to $C^{\mathrm{Com}}_{\boldsymbol{X}}$ forms a cycle then this set of edges from j to i in $C^{\mathrm{Com}}_{\boldsymbol{X}}$ implies $i \in K_{\boldsymbol{X}}$ but then we know $i \notin K_{\boldsymbol{X}}$ for $\boldsymbol{X}_{i,e}$ to be run-like.

- We know that $C_{\boldsymbol{X}}^{\text{Pen}}$ did not contain a cycle, since by assumption $\boldsymbol{X}$ was not lassoed. If $C_{\boldsymbol{X}'}^{\text{Pen}}$ contains a cycle, we must be in the case where we have $C_{\boldsymbol{X}}^{\text{Pen}} \subset C_{\boldsymbol{X}'}^{\text{Pen}}$, thus the newly added edge forms the cycle as required. Note in other cases the graph either does not change or the final edge is removed, not introducing any cycles.
- In the graph $([k], E_{\boldsymbol{X}'}^{\text{Pen}} \cup E_{\boldsymbol{X}'}^{\text{Com}})$ we only need to consider the case when $\boldsymbol{x}^l = \boldsymbol{0}^l$ but $\boldsymbol{x}'^l \neq \boldsymbol{0}^l$, we note, this only occurs in the case $l = j_{(r_{\boldsymbol{X}}+1)}$ in which case $(E_{\boldsymbol{X}'}^{\text{Pen}} \cup E_{\boldsymbol{X}'}^{\text{Com}}) = (E_{\boldsymbol{X}}^{\text{Pen}} \cup E_{\boldsymbol{X}}^{\text{Com}})$ and that both $(j_{r_{\boldsymbol{X}}}, j_{(r_{\boldsymbol{X}}+1)}) \in E_{\boldsymbol{X}}^{\text{Pen}}$ and $1 \to^* j_{r_{\boldsymbol{X}}}$, since $\boldsymbol{x}^{(j_{r_{\boldsymbol{X}}})} \neq \boldsymbol{0}^{(j_{r_{\boldsymbol{X}}})}$. Thus also $1 \to^* j_{(r_{\boldsymbol{X}}+1)}$

$\square$

Lemma 4.2. *Let G be an instance of Recursive Arrival, and let $\boldsymbol{X} \in \mathscr{X}$ be a run-like recursive switching flow on G. If $\boldsymbol{X}$ is non-zero then there exists a unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} - \boldsymbol{U}_{i,e}) \in \mathscr{X}$ is a run-like recursive switching flow. Otherwise, if $\boldsymbol{X}$ is all-zero, then no such $\boldsymbol{U}_{i,e}$ exists.*

Proof. We shall show for non-zero $\boldsymbol{X}$ the following choice is the unique value for i, and e can be determined using $E^*_{i,\boldsymbol{x}^i}$. We take:

- If $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} > \boldsymbol{0}^{j_{(r_{\boldsymbol{X}}+1)}}$ and $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) = 0$ then take $i = j_{(r_{\boldsymbol{X}}+1)}$.
- Otherwise, take $i = j_{r_{\boldsymbol{X}}}$. Where, necessarily, that either $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) \geq 1$ or $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} = \boldsymbol{0}^{j_{(r_{\boldsymbol{X}}+1)}}$.

For pairs $i \in [k]$ and $e \in E_i \cup F_i$ we define $\boldsymbol{X}_{i,e} := (\boldsymbol{X} - U_{i,e}) = (\boldsymbol{x}^1_{i,e}, \ldots, \boldsymbol{x}^k_{i,e})$ where this subtraction is defined and shall just say $\boldsymbol{X}_{i,e}$ is undefined otherwise.

The case for $\boldsymbol{X} = \boldsymbol{0}$ is trivial, as for any choice of i and e $\boldsymbol{X}_{i,e}$ is undefined. Thus, we show for $\boldsymbol{X} > \boldsymbol{0}$ that there is a unique $\boldsymbol{X}_{i,e}$ which is defined and a run-like recursive switching flow. Since $\boldsymbol{X}$ is run-like, we know that $\boldsymbol{x}^1 > \boldsymbol{0}^1$. It is also evident that for any $i \in [k]$ with $\boldsymbol{x}^i = \boldsymbol{0}^i$ that $\boldsymbol{X}_{i,e}$ is undefined for all $e \in E_i \cup F_i$.

We know by our results on non-recursive switching flows, proposition 2.5, that for any fixed $i \in [k]$ with $\boldsymbol{x}^i \neq \boldsymbol{0}^i$ that there is a unique edge $e_i \in E_i \cup F_i$ such that $\boldsymbol{X}_{i,e_i}$ is defined and that $\boldsymbol{x}^i_{i,e_i}$ is a component switching flow. It follows this e_i is the unique edge in $E^*_{i,\boldsymbol{x}^i}$ such that either:

- e_i is the unique edge into $d^i_{\boldsymbol{X}}$,
- e_i is on the unique cycle of $E^*_{i,\boldsymbol{x}^i}$ containing $d^i_{\boldsymbol{X}}$ and is the incoming edge to $d^i_{\boldsymbol{X}}$ on this cycle.

Hence, for any other edge $e \in E_i \cup F_i \setminus \{e_i\}$ we have that $\boldsymbol{X}_{i,e}$ is either undefined or, when defined, that $\boldsymbol{x}^i_{i,e}$ is not a component switching flow, thus $\boldsymbol{X}_{i,e}$ cannot be a recursive switching flow. We now show there is exactly one such $i \in [k]$ such that $\boldsymbol{X}_{i,e_i}$ is defined and a recursive switching flow by considering the conditions on the pending-call graph. Consider some fixed $i \in [k]$ with $\boldsymbol{x}^i > \boldsymbol{0}^i$, so that we know $\boldsymbol{X}_{i,e_i}$ is defined. We observe that for $j \in [k] \setminus \{i\}$ that the current vertex in component j is unchanged between $\boldsymbol{X}$ and $\boldsymbol{X}_{i,e_i}$, i.e., $d^j_{\boldsymbol{X}} = d^j_{\boldsymbol{X}_{i,e_i}}$. Thus, for such j we have $j \in J_{\boldsymbol{X}}$ if and only if $j \in J_{\boldsymbol{X}_{i,e_i}}$ and similarly for $K_{\boldsymbol{X}}$ and $K_{\boldsymbol{X}_{i,e_i}}$. We also note that if $j_1, \ldots, j_{r_{\boldsymbol{X}}}, j_{(r_{\boldsymbol{X}}+1)}$ is the component call stack of $\boldsymbol{X}$ and for $l \in [r_{\boldsymbol{X}}]$ that if $j_l \neq i$ then $(j_l, j_{l+1}) \in E^{\text{Pen}}_{\boldsymbol{X}_{i,e_i}}$. We note for i that if $i \in J_{\boldsymbol{X}}$ then $i \notin J_{\boldsymbol{X}_{i,e_i}}$ and if $i \in K_{\boldsymbol{X}}$ then $i \notin K_{\boldsymbol{X}_{i,e_i}}$.

We may consider the three cases of the state of component i to determine the necessary conditions for $\boldsymbol{X}_{i,e_i}$ to be run-like:

- If $i \notin K_{\boldsymbol{X}} \cup J_{\boldsymbol{X}}$, since $\boldsymbol{X}$ is also run-like we must have that either $\boldsymbol{x}^i = \boldsymbol{0}^i$ or that $i = j_{(r_{\boldsymbol{X}}+1)}$. By our assumption, it must be the later case, i.e., $i = j_{(r_{\boldsymbol{X}}+1)}$. It must also be the case that $CC(\boldsymbol{X}, i) = 0$ as $CC(\boldsymbol{X}, i) > 0$ implies $i \in K_{\boldsymbol{X}}$.

- If $i \in J_{\boldsymbol{X}}$ then it must be the case that $i = j_{r_{\boldsymbol{X}}}$, since we know other edges in the pending-call graph are unchanged we must remove the last index in the sequence. Then it must be the case that $J_{\boldsymbol{X}_{i,e_i}} = J_{\boldsymbol{X}} \setminus \{i\}$ and that $r_{\boldsymbol{X}_{i,e_i}} = r_{\boldsymbol{X}} - 1$. Considering $j_{(r_{\boldsymbol{X}}+1)}$ we note we must have one of $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$, $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$ or $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} = \boldsymbol{0}^{j_{(r_{\boldsymbol{X}}+1)}}$, since by $\boldsymbol{X}_{i,e_i}$ run-like we must have one of $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}_{i,e_i}} \cup K_{\boldsymbol{X}_{i,e_i}}$ or $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} = \boldsymbol{0}^{j_{(r_{\boldsymbol{X}}+1)}}$ since $j_{(r_{\boldsymbol{X}}+1)} \neq j_{r_{\boldsymbol{X}}}$. We also note that edge $(j_{r_{\boldsymbol{X}}}, j_{(r_{\boldsymbol{X}}+1)})$ is removed from the graph $([k], E^{\text{Pen}}_{\boldsymbol{X}} \cup E^{\text{Com}}_{\boldsymbol{X}})$ to $([k], E^{\text{Pen}}_{\boldsymbol{X}_{i,e_i}} \cup E^{\text{Com}}_{\boldsymbol{X}_{i,e_i}})$, thus, since both are run-like, there must be some other incoming edge to i in $([k], E^{\text{Pen}}_{\boldsymbol{X}} \cup E^{\text{Com}}_{\boldsymbol{X}})$. We can see this happens if and only if $CC(\boldsymbol{X}, j_{r+1}) \geq 1$. Any such edge must be in $E^{\text{Com}}_{\boldsymbol{X}_{i,e_i}} = E^{\text{Com}}_{\boldsymbol{X}}$ and $CC(\boldsymbol{X}, j_{r+1})$ counts, with positive multiplicity, the number of such edges.
- If $i \in K_{\boldsymbol{X}}$ then we know that $i \notin K_{\boldsymbol{X}_{i,e_i}}$. Since the only successors of call-ports are return-ports (and thus not exits) we also have that $i \notin J_{\boldsymbol{X}_{i,e_i}}$ and thus that $J_{\boldsymbol{X}_{i,e_i}} = J_{\boldsymbol{X}}$. Thus, by our run-like conditions, it must be that either $\boldsymbol{x}^i_{i,e_i} = \boldsymbol{0}^i$ or $i = j_{r_{\boldsymbol{X}+1}}$. In either case, since $i \notin K_{\boldsymbol{X}_{i,e_i}}$ we must have that for every $l \in [k]$, $b \in B_l$ and $f \in F_{b,l}$ that if $Y_l(b) = i$ then $x^l_{i,e_i}(f) = 0$ by our recursive switching flow definition. Evidently, this holds if and only if $CC(\boldsymbol{X}_{i,e_i}, i) = 0$, and since $\boldsymbol{X}$ agrees on all edges bar e, that $CC(\boldsymbol{X}), i) = 0$. Finally, considering the case when $\boldsymbol{x}^i_{i,e_i} = \boldsymbol{0}^i$, we know that $CC(\boldsymbol{X}, i) = 0$, thus since there is a path from 1 to i in $([k], E^{\text{Pen}}_{\boldsymbol{X}} \cup E^{\text{Com}}_{\boldsymbol{X}})$ this must use an edge in $E^{\text{Pen}}_{\boldsymbol{X}}$, since there are no incoming edges to i in $E^{\text{Com}}_{\boldsymbol{X}}$. Thus, since $i \notin J_{\boldsymbol{X}}$ this edge must be $(j_{r_{\boldsymbol{X}}}, j_{(r_{\boldsymbol{X}}+1)})$ and $i = j_{(r_{\boldsymbol{X}}+1)}$.

We note by these implications only one of the cases may occur, thus we may uniquely determine i as according to Appendix C. We note $\boldsymbol{X}_{i,e_i}$ is evidently run-like as all the conditions follow from it being recursive and bounded by $\boldsymbol{X}$. □

Lemma C.2. *Let G be an instance of Recursive Arrival and $\boldsymbol{X}, \boldsymbol{Y} \in \mathscr{X}$ be run-like recursive switching flows on G. If $Val(\boldsymbol{X}) = Val(\boldsymbol{Y})$ then $\boldsymbol{X} = \boldsymbol{Y}$.*

Proof. We show for each $\ell \in \mathbb{N}$ there exists at most 1 choice of $\boldsymbol{X} \in \mathscr{X}$ with $Val(\boldsymbol{X}) = \ell$. For $\ell = 0$ it is trivial that $\boldsymbol{0} \in \mathscr{X}$ is the unique vector with $Val(\boldsymbol{0}) = 0$. Assume the hypothesis holds for some fixed $\ell \in \mathbb{N}$ with $\ell > 0$. Assume there exists $\boldsymbol{X}, \boldsymbol{Y} \in \mathscr{X}$ with $Val(\boldsymbol{X}) = \ell + 1 = Val(\boldsymbol{Y})$, we will then show that $\boldsymbol{X} = \boldsymbol{Y}$. By Lemma 4.2 there exists unique $\boldsymbol{U}_{i,e}, \boldsymbol{U}_{i',e'} \in \mathscr{U}$ such that both $(\boldsymbol{X} - \boldsymbol{U}_{i,e}), (\boldsymbol{Y} - \boldsymbol{U}_{i',e'}) \in \mathscr{X}$, since we must have $Val(\boldsymbol{X} - \boldsymbol{U}_{i,e}) = Val(\boldsymbol{Y} - \boldsymbol{U}_{i',e'}) = \ell$ by our inductive hypothesis $(\boldsymbol{X} - \boldsymbol{U}_{i,e}) = (\boldsymbol{Y} - \boldsymbol{U}_{i',e'}) = \boldsymbol{Z}$. Using Lemma 4.1 on $\boldsymbol{Z}$ we know there is at most one vector $\boldsymbol{U}' \in \mathscr{U}$ such that $(\boldsymbol{Z} + \boldsymbol{U}') \in \mathscr{X}$, however $\boldsymbol{Z} + \boldsymbol{U}_{i,e} = \boldsymbol{X} \in \mathscr{X}$ and $\boldsymbol{Z} + \boldsymbol{U}_{i',e'} = \boldsymbol{Y} \in \mathscr{X}$, hence we must have that $\boldsymbol{U}_{i,e} = \boldsymbol{U}_{i',e'}$ and thus $\boldsymbol{X} = \boldsymbol{Y}$. □

Lemma C.3. *Let G be an instance of Recursive Arrival. For any run-like recursive switching flow, $\boldsymbol{X} \in \mathscr{X}$, there exists some $t \in \mathbb{N}$ such that $\boldsymbol{Run}(G,t) = \boldsymbol{X}$.*

Proof. We observe firstly that $\boldsymbol{Run}(G,0) = \boldsymbol{0}$ and this is obviously the unique run-like recursive switching flow with $Val(\boldsymbol{X}) = 0$. We then consider some value $t \in \mathbb{N}$ with $\boldsymbol{Run}(G,t) \neq \boldsymbol{Run}(G,t+1)$, then it is evident from the definition that $Val(\boldsymbol{Run}(G,t)) + 1 = Val(\boldsymbol{Run}(G,t+1))$. We assume, for contradiction, that $\boldsymbol{X}$ is the Val-minimal run-like recursive switching flow such that, for all $t \in \mathbb{N}$, $\boldsymbol{Run}(G,t) \neq \boldsymbol{X}$. By Lemma C.2 we know that if such a minimal $\boldsymbol{X}$ exists it is unique. By Lemma 4.2 there is some unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} - \boldsymbol{U}_{i,e})$ is a run-like recursive switching flow. We must therefore have some $t \in \mathbb{N}$ such that $(\boldsymbol{X} - \boldsymbol{U}_{i,e}) = \boldsymbol{Run}(G,t)$, because $Val(\boldsymbol{X} - \boldsymbol{U}_{i,e}) = Val(\boldsymbol{X}) - 1$ and we have assumed $\boldsymbol{X}$ is Val-minimal. We want to show that there exists a time $t' > t$ such that $\boldsymbol{X} = \boldsymbol{Run}(G,t')$. There are two cases to consider:

- If there $\exists t' > t$ such that $\boldsymbol{Run}(G,t') \neq \boldsymbol{Run}(G,t)$, then there is some minimal t' which satisfies $Val(\boldsymbol{Run}(G,t')) = 1 + Val(\boldsymbol{Run}(G,t)) = Val(\boldsymbol{X})$. Using Lemma C.2 we see this would imply that $\boldsymbol{X} = \boldsymbol{Run}(G,t')$.
- Otherwise, $\forall t' > t$ we have that $\boldsymbol{Run}(G,t') = \boldsymbol{Run}(G,t)$. We then show that no run-like recursive switching flow can have value greater than $Val(\boldsymbol{Run}(G,t)) = Val(\boldsymbol{X}) - 1$, thus contradicting that $\boldsymbol{X}$ is a run-like recursive switching flow. We consider the sequences $S_1,\dots,S_k$ (first entry times), $T_1,\dots,T_k$ (first exit times), for each $l \in [k]$ with $S_l < \infty$ the first-encounter stack β^l (the call-stack at the first entry time) and, finally, the component runs, the (potentially-finite) subsequence $t_1^l, t_2^l, \dots$ of all times $t_j^l \in [S_l \dots T_l]$ where our component call stack is equal to β^l. Our assertion, that $\forall t' > t$ we have that $\boldsymbol{Run}(G,t') = \boldsymbol{Run}(G,t)$, implies that for each $l \in [k]$ that $\sup\{t_1^l, t_l^2, \dots\} \leq t$, where we take the supremum of the empty sequence to be 0. Thus, for each $l \in [k]$, the sequence $t_1^l, t_l^2, \dots$ must be of finite length. We let $d^l_{\boldsymbol{Run}(G,t)} \in V_l$ be the current-vertex of the flow $\boldsymbol{Run}(G,t)$ in component l. Then we must, for each $l \in [k]$, have that $d^l_{\boldsymbol{Run}(G,t)} \in \{o_l\} \cup \mathsf{Call}_l \cup \mathsf{Ex}_l$. Those at o_l have empty sequences $t_1^l, \dots$ and thus $\boldsymbol{run}(G^l,t) = \mathbf{0}^l$. Those in Ex_l have $T_l \leq t$, thus T_l finite.

  We do so by showing that $\boldsymbol{Run}(G,t)$ must be finished. Considering our run profile at time t it must be that after time t, for any component $l \in [k]$, we cross no more edges in the calling context (i.e., with the call stack) β_l, which is our first ever calling context when encountering component l. The two ways in which it can occur that $\forall t' > t$ we have $\boldsymbol{Run}(G,t') = \boldsymbol{Run}(G,t)$ are as follows:

  1. The run has completed at time t, and thus $1 \in K_{\boldsymbol{Run}(G,t)} = K_{\boldsymbol{Run}(G,t')}$. Hence this run profile is finished.
  2. We never return to any of the β_l calling contexts, thus the stack is bounded away from 0 for the remainder of the run. In this case, we know by Lemma 2.7 that we must make a call to some previously visited component at time t. Thus $\boldsymbol{Run}(G,t)$ is call-pending with $j_{(r_{\boldsymbol{Run}(G,t)}+1)} \in J_{\boldsymbol{Run}(G,t)}$, thus it is also finished.

  We note in both these cases there are no possible post-overflowing flows, as any flow which is strictly greater can not be run-like as it must contradict the last-used edge graph's acyclicity condition.

We note that if we have a finished switching flow of the final type, i.e., where $x_e^i > 2^{\mathrm{p}(N)}$, then there are run-like recursive switching flows for every $t \in \mathbb{N}$. Any run-like switching flows greater than a finished run-like switching flow one must be post-overflowing by definition. □

Lemma C.4. *Let G be an instance of Recursive Arrival. There exists a unique finished run-like recursive switching flow, $\boldsymbol{X} \in \mathscr{F}$, on G. Furthermore, $\boldsymbol{X} \in \mathscr{F}^N$ where $N = 2^{\max_i |V_i|} + 1$.*

Proof. Consider the sequence $\boldsymbol{Run}(G,t)$'s, for all $t \in \mathbb{N}$, which are all run-like recursive switching flows. Note moreover that $\boldsymbol{Run}(G,t) \leq \boldsymbol{Run}(G,t+1)$ for all $t \in \mathbb{N}$. Consider the set $\{\boldsymbol{Run}(G,t) \mid t \in \mathbb{N}\}$, which by Lemma C.3 is precisely equal to $\mathscr{X}$. Either the set $\mathscr{X} = \{\boldsymbol{Run}(G,t) \mid t \in \mathbb{N}\}$ is finite, in which case it contains some coordinate-wise maximum vector corresponding to the unique finished recursive run-like switching flow which is either completed or lassoed, or else, the set $\mathscr{X}$ is infinite in which case it must contain vectors with arbitrarily large coordinate values. Thus we can pick the minimum vector in $\mathscr{X}$ with an "excessively large" coordinate value equal to $2^{\mathrm{p}(N)} + 1$, and this uniquely determined minimum vector will correspond to $\boldsymbol{Run}(G,t')$ for some specific first-time t' when we one of the edge counters overflows.

We know such a flow is bounded by $2^{\mathrm{p}(N)} + 1$, since any component flow which terminates is bounded by this value (Proposition 2.3). □

Lemma C.5. *There is a polynomial-time algorithm that, given G, an instance of Recursive Arrival, and given $\boldsymbol{X} \in \mathscr{F}$, decides whether $\boldsymbol{X}$ is a finished run-like recursive switching flow on G.*

Proof. We need only to check each of the conditions which define each of being a: "component switching flow"; "recursive switching flow"; "run-like recursive switching flow"; and "finished", including checking which of the (mutually exclusive) conditions "complete", "lassoed" and "just-overflowing" is satisfied. Each condition resolves to a check on some graph of either an inequality relating coordinates of $\boldsymbol{X}$ or, a reachability question on graphs like $C^{\text{Pen}}_{\boldsymbol{X}}$ or $C^{\text{Com}}_{\boldsymbol{X}}$, both of which are computable in polynomial time from a given $\boldsymbol{X}$ and reachability is similarly computable. □

Lemma C.6. *There is a polynomial time algorithm that, given G, an instance of Recursive Arrival, given $\boldsymbol{X} \in \mathscr{X}$, a finished run-like recursive switching flow on G, and given an exit, $d \in \mathsf{Ex}_1$, decides whether or not the run on G terminates at d.*

Proof. We simply need to check under which condition $\boldsymbol{X}$ is finished. If it is "complete", we decide "yes" whenever the current vertex $d^1_{\boldsymbol{x}^1}$ of $\boldsymbol{x}^1$ satisfies $d^1_{\boldsymbol{x}^1} = d$. In all other cases, we decide "no". Correctness follows from the fact that there is a unique finished run-like recursive switching flow which corresponds to corresponds to some $\boldsymbol{Run}(G,t)$, and which either indicates that the run is complete and terminates at d (the only "yes" case), or else it terminates elsewhere, or it doesn't terminate. □

Theorem 4.3. *The* `Recursive Arrival` *problem is in* $\mathsf{NP} \cap \mathsf{coNP}$ *and* $\mathsf{UP} \cap \mathsf{coUP}$.

Proof. Our algorithm proceeds to guess some flow $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k) \in \mathscr{F}^N$, where $N = 2^{\max\{|V_i|\}} + 1$, with for $l \in [k]$ each vector $\boldsymbol{x}^l = (x^l_e \in [0\ldots N] \mid e \in E_l \cup F_l)$. We verify, in polynomial time, whether or not $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k)$ is a finished run-like recursive switching flow using Lemma C.5. We know for any G there is a unique choice of $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k)$ satisfying this property by Lemma C.4. Using Lemma C.6 we are able to determine from $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k)$, in polynomial time, whether or not the run on G terminates and if so which exit in Ex_1 it terminates at. Since these options are mutually exclusive and include all possibilities, the value $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k)$ effectively certifies the entire dynamics of the run on G. □

C.1 Containment in UEOPL

Theorem 4.4. *The* `Search-Recursive Arrival` *is in* UEOPL.

Proof. We will give a polynomial-time search reduction from `Search Recursive Arrival` to the `UniqueEOPL` problem.

We will consider the space $\mathscr{F}^N$, where $N = 2^{\max\{|V_i|\}} + 1$, the elements of which are vector sequences, $(\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k)$, where for $l \in [k]$ we have $\boldsymbol{x}^l = (x^l_e \in [0\ldots N] \mid e \in E_l \cup F_l)$. We denote by $\mathscr{X}^N := \mathscr{X} \cap \mathscr{F}^N$, i.e., the set of run-like Recursive Switching flows which are also bounded in each coordinate by N. Elements of this space may be described using $(\sum_{l\in[k]}|E_l \cup F_l|) \cdot (\log_2(N))$ bits, which is polynomial in the size of the given `Search Recursive Arrival` instance. Our boolean circuits S,P, and V are then given by the restriction of the functions Adv, $Prev$, and Val to the domain $\mathscr{F}^N$, where we view elements of this space as bit sequences.

We show that there is a polynomial time algorithm that, given an instance, G, of `Search Recursive Arrival` computes an instance of `UniqueEOPL` by outputting boolean circuits representing S,P, and V. Therefore, we need to show that we can efficiently compute a boolean circuit for each of Adv, $Prev$, and Val. To do so, we show how to efficiently compute boolean circuits (i.e., straight line programmes with boolean operations) for each of the following tasks, given $\boldsymbol{X} \in \mathscr{F}^N$:

1. Decide whether or not $\boldsymbol{X}$ is all zero (i.e., the boolean circuit outputs 1 or 0, depending on whether or not $\boldsymbol{X}$ is all zero, respectively).
2. Decide whether or not $\boldsymbol{X} \in \mathscr{X}^N$.
3. Decide whether or not $\boldsymbol{X}$ is finished.
4. Decide whether or not $\boldsymbol{X}$ is post-overflowing.
5. Compute the vector $\boldsymbol{U}_{i,e} \in \mathscr{U}$ given by Lemma 4.1.
6. Compute the vector $\boldsymbol{U}_{i,e} \in \mathscr{U}$ given by Lemma 4.2.

From these and standard straight line programmes for addition and subtraction, it is straightforward to compute a boolean circuit for each of Adv, $Prev$, and Val. We will define the multi-input boolean gate ONE, which evaluates to true if one and only if exactly one of its arguments is true, and evaluates to false if either none or two or more of the arguments are true. We now proceed to show how to compute the above boolean circuits. Determining whether the input is all-zero is trivial. Next, we show how to decide whether or not $\boldsymbol{X} \in \mathscr{X}^N$. We define the following boolean-valued outputs for each $l \in [k]$ indexed by vertices $v \in V_l$ and boxes $b \in B_l$:

- $\alpha^l_{F1}[v]$ which is true if and only if $\left(\sum_{e=(u,v)\in E_l \cup F_l} x^l_e\right) - \left(\sum_{e=(v,w)\in E_l \cup F_l} x^l_e\right) = 1$
- $\alpha^l_{F0}[v]$ which for $v \in V_l \setminus \{o_l\}$ is true if and only if $\left(\sum_{e=(u,v)\in E_l \cup F_l} x^l_e\right) - \left(\sum_{e=(v,w)\in E_l \cup F_l} x^l_e\right) = 0$.
- $\alpha^l_{F0}[o_l]$ which is true if and only if $\left(\sum_{e=(o_l,w)\in E_l \cup F_l} x^l_e\right) = 1$
- $\alpha^l_{S0}[v]$ which for $v \in \mathsf{Sor}_l$ is true if and only if $x_{(v,s^1(v))} = x_{(v,s^0(v))}$ (we don't require the boolean value $\alpha^l_{S0}[v]$ to be defined when $v \notin \mathsf{Sor}_l$).
- $\alpha^l_{S1}[v]$ which for $v \in \mathsf{Sor}_l$ is true if and only if $x_{(v,s^0(v))} = x_{(v,s^1(v))} + 1$ (we don't require the boolean value $\alpha^l_{S1}[v]$ to be defined when $v \notin \mathsf{Sor}_l$).
- $\alpha^l_B[b]$ which is true if and only if for at most one $f \in F_{b,l}$ we have $x^f_l > 0$.

Clearly, each of the boolean values in the above list can be obtained as the output of an efficiently computable boolean circuit, given $\boldsymbol{X} \in \mathscr{F}^N$. Thus, given $\boldsymbol{X} = (\boldsymbol{x}^1, \dots, \boldsymbol{x}^k) \in \mathscr{F}^N$, $\boldsymbol{x}^l$ is a component switching flow if and only if all of the following conditions hold:

- For every $v \in V_l$, $(\alpha^l_{F1}[v] \wedge \alpha^l_{F0}[v])$ is true. In other words, $\bigwedge_{v\in V_l}(\alpha^l_{F1}[v] \wedge \alpha^l_{F0}[v])$ is true.
- There is exactly one vertex, $d_l \in V_l$, with $\alpha^l_{F1}[d_l]$ true, i.e., $ONE_{d_l \in V_l}(\alpha^l_{F1}[d_l])$.
- $\bigwedge_{v\in \mathsf{Sor}_l}(\alpha^l_{S1}[v] \vee \alpha^l_{S0}[v])$ is true.
- $\bigwedge_{b\in B_l} \alpha^l_B[b]$ is true.

Assuming $\boldsymbol{x}^l$ is a component switching flow we know that d_l is the current vertex $d^l_{\boldsymbol{x}^l}$. We may then determine whether $\boldsymbol{x}^l$ is complete, which holds if and only if $OR(\alpha^l_{F1}[v] \mid v \in \mathsf{Ex}_l)$ is true. Thus, our recursive switching conditions ask that if for any edge $f = ((b, o_{Y_i(b)}), (b, d_{Y_i(b)}))$ in F_l we have $x^l_f > 0$, then $\alpha^l_{F1}[d_{Y_i(b)}]$ must be true. This condition can be written as $\bigwedge_{f\in F_l}((x^l_f > 0) \implies (\alpha^l_{F1}[d_{Y_i(b)}]))$. Thus, we may determine whether $\boldsymbol{X}$ is a recursive switching flow using an AND of each of these conditions, confirming that for all $l \in [k]$ each x^l is a component switching flow and x^l satisfies the recursive switching conditions. Note that each of these conditions is well-defined if all the previous conditions are true and, otherwise, any previous false condition determines the output already.

We may evidently use these to then determine whether or not a given edge is in the pending-call graph, $C^{\mathrm{Pen}}_{\boldsymbol{X}}$, completed-call graph, $C^{\mathrm{Com}}_{\boldsymbol{X}}$, and last-used edge graphs, $G^*_{l,\boldsymbol{x}^l}$, as follows:

- $(i,j) \in E^{\text{Pen}}_{\boldsymbol{X}}$ if and only if $\exists b \in B_i$ with $Y_i(b) = j$ such that $\alpha^i_{F1}[(b,o_j)]$ is true. I.e., $\bigvee_{b\in B_I}((Y_i(b) = j) \wedge (\alpha^i_{F1}[(b,o_{Y_i(b)})]))$.
- $(i,j) \in E^{\text{Com}}_{\boldsymbol{X}}$ if and only if $\exists b \in B_i, \exists f \in F_{b,i}$, such that $((Y_i(b) = j) \wedge (x^i_f > 0))$.
- For each $i \in [k]$, for the last-used-edge graph, $G^*_{i,\boldsymbol{x}^i}$, we need to decide whether the last-used edge from each $v \in \mathsf{Sor}_i$ exists and if it does whether it is $(v,s^0(v))$ or $(v,s^1(v))$. We can do so as follows:
  - There is no last-used edge when $x^i_{(v,s^0(v))} = 0$.
  - $(v,s^1(v))$ is the last-used edge when $(x^i_{(v,s^0(v))} > 0) \wedge (\alpha^i_{S0}[v])$.
  - $(v,s^0(v))$ is the last-used edge when $\alpha^i_{S1}[v]$ is true. Note that for $\boldsymbol{x}^i$ a component switching flow, $\alpha^i_{S1}[v]$ being true implies $(x^i_{(v,s^0(v))} > 0) \wedge \neg\alpha^i_{S0}[v]$ is also true, and hence these three cases are complete and mutually exclusive.

This process also allows us to determine the membership of sets $J_{\boldsymbol{X}}$ and $K_{\boldsymbol{X}}$. To then determine whether on not our sequence is run-like we need to verify all the following defining conditions for being "run-like":

- For each $l \in [k]$, $G^*_{l,\boldsymbol{x}^l}$ is either acyclic or has exactly one cycle containing d_l.
- The edges of the pending-call graph, $E^{\text{Pen}}_{\boldsymbol{X}}$, on the set $J_{\boldsymbol{X}}$ form either a directed line, or a "lasso" consisting of a directed line ending at a directed cycle.
- Every $i \in [k]$, $\boldsymbol{x}^i$ is either complete, call-pending, in-progress, or all-zero.
- The completed-call graph, $C^{\text{Com}}_{\boldsymbol{X}}$, is acyclic.
- For any $l \in [k]$, if $\boldsymbol{x}^l \neq \boldsymbol{0}^l$, then in the graph $([k], E^{\text{Pen}}_{\boldsymbol{X}} \cup E^{\text{Com}}_{\boldsymbol{X}})$ we must have $1 \to^* l$, i.e., there must be a path in this graph from component 1 to all components l for which $\boldsymbol{x}^l$ is non-zero.

Using these defining conditions for being run-like, we can also determine whether or not a run-like recursive switching flow is finished, post-overflowing or neither.

For the circuit for S, we wish to compute the values $i \in [k]$ and $e \in E_i \cup F_i$ such that $\boldsymbol{U}_{i,e} \in \mathscr{U}$ is the unique vector given by Lemma 4.1. By the proof of Lemma 4.1 we have that either $i = j_{r_{\boldsymbol{X}}}$ or $i = j_{(r_{\boldsymbol{X}}+1)}$. We know $j_{r_{\boldsymbol{X}}}$ is the unique sink of the pending-call graph, and thus can be determined by examining $E^{\text{Pen}}_{\boldsymbol{X}}$. We have $i = j_{r_{\boldsymbol{X}}}$ if and only if $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$, and otherwise, we have $i = j_{(r_{\boldsymbol{X}}+1)}$. Since we can determine membership of $K_{\boldsymbol{X}}$, we can determine the component i associated with the unique vector $\boldsymbol{U}_{i,e}$. We can then also determine e. Namely, e is $(d_i,s^0(d_i))$ in case $\alpha^i_{S0}[d_i]$, and otherwise it is $(d_i,s^1(d_i))$ in case $\alpha^i_{S1}[d_i]$, and these conditions are complete and mutually exclusive. Here d_i is the current vertex of $\boldsymbol{x}^i$, $d^i_{\boldsymbol{x}^i}$, which, as we have seen before, we can compute efficiently using a boolean circuit as the unique vertex $v \in \mathsf{Sor}_l$ at which $\alpha^l_{F1}[v]$ outputs true.

Similarly, for the circuit P, we wish to compute $i \in [k]$ and $e \in E_i \cup F_i$ such that $\boldsymbol{U}_{i,e} \in \mathscr{U}$ is the unique vector given by Lemma 4.2. By the proof of Lemma 4.2 we have that either $i = j_{r_{\boldsymbol{X}}}$ or $i = j_{(r_{\boldsymbol{X}}+1)}$. We again know $j_{r_{\boldsymbol{X}}}$ is the unique sink of the pending-call graph and thus can be determined. The function $CC(\boldsymbol{X},l)$ for $l \in [k]$ is obviously computable efficiently, thus we can determine whether or not $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) > 0$. We similarly see by the proof of Lemma 4.2 that we can determine whether $i = j_{r_{\boldsymbol{X}}}$ or $i = j_{(r_{\boldsymbol{X}}+1)}$, and then using the last-used edge graph, $G^*_{i,\boldsymbol{x}^i}$, we can determine the edge, e, as either the unique incoming edge to d_i or the unique incoming edge on the unique cycle containing d_i.

These circuits allow us to compute S (resp. P). To see this, note that we are able to use the output of gates we have computed which determine, for a given input $\boldsymbol{X}$, (a) whether $\boldsymbol{X} \in \mathscr{X}^N$, and (b) whether $\boldsymbol{X}$ is all-zero, (c) whether $\boldsymbol{X}$ is finished, or (d) whether $\boldsymbol{X}$ is post-overflowing. In each case, for the successor (and predecessor) circuits S (and P), we can appropriately output the result of incrementing (resp. decrementing) $\boldsymbol{X}$ by $\boldsymbol{U}_{i,e}$ for the computed pair (i,e) and otherwise, we output $\boldsymbol{X}$ itself (meaning there is a self-loop at $\boldsymbol{X}$). As these combine polynomial-sized circuits, this construction gives circuits for both S and P which have a polynomial number of gates in the size of our instance G.

Thus, we have constructed an instance of `UniqueEOPL`. Through our construction of the circuits, we have the following properties:

(A) If $\boldsymbol{X} \in \mathscr{F}^N \setminus \mathscr{X}^N$ then $P(\boldsymbol{X}) = \boldsymbol{X} = S(\boldsymbol{X})$, i.e., such points $\boldsymbol{X}$ are isolated self-loops.

(B) If $\boldsymbol{X} \in \mathscr{X}^N$ and $\boldsymbol{X} \neq S(\boldsymbol{X})$, then $V(\boldsymbol{X}) + 1 = V(S(\boldsymbol{X}))$, i.e., if $\boldsymbol{X}$ has a successor, then the value increases by 1 from $\boldsymbol{X}$ to $S(\boldsymbol{X})$.

(C) If instead $\boldsymbol{X} \in \mathscr{X}^N$ and $S(P(\boldsymbol{X})) \neq \boldsymbol{X}$, then $\boldsymbol{X} = \mathbf{0}$, i.e., $\mathbf{0}$ is the only start of a line.

(D) For $\boldsymbol{X}, \boldsymbol{Y} \in \mathscr{X}^N$, if $V(\boldsymbol{X}) = V(\boldsymbol{Y})$ then $\boldsymbol{X} = \boldsymbol{Y}$ by Lemma C.2, i.e. V gives each run-like recursive switching flow a unique value.

(E) If $\boldsymbol{X} \in \mathscr{X}^N$ and $P(S(\boldsymbol{X})) \neq \boldsymbol{X}$, then $\boldsymbol{X}$ is finished, i.e., any end of a line is a finished run-like switching flow (which is what we want to compute).

We now consider the points $\boldsymbol{X} \in \mathscr{F}^N$ (or the pair $\boldsymbol{X}, \boldsymbol{Y}$ in the (UV3) solution case) which can be returned as a valid `UniqueEOPL` solution. By (A) we know any point in a returned solution, any of (U1) or (UV1-3), must be in $\mathscr{X}^N$, since no isolated point can be in a solution. Thus, we can only consider points $\boldsymbol{X}, \boldsymbol{Y} \in \mathscr{X}^N$ which are in this set.

By (B) we know for $\boldsymbol{X} \in \mathscr{X}^N$ that when $\boldsymbol{X} \neq S(\boldsymbol{X})$ that $V(S(\boldsymbol{X})) = 1 + V(\boldsymbol{X}) \not\leq V(\boldsymbol{X})$, thus, $\mathscr{X}^N$ contains no (UV1) solutions. By (C) we know if $\boldsymbol{X} \in \mathscr{X}^N$ has $S(P(\boldsymbol{X})) \neq \boldsymbol{X}$ then $\boldsymbol{X} = \mathbf{0}$, thus there are no (UV2) solutions in $\mathscr{X}^N$. We now consider the existence of a (UV3) solution pair $\boldsymbol{X}, \boldsymbol{Y} \in \mathscr{X}^N$. We consider all such pairs satisfying $\boldsymbol{X} \neq \boldsymbol{Y}$, $\boldsymbol{X} \neq S(\boldsymbol{X})$ and $\boldsymbol{Y} \neq S(\boldsymbol{Y})$ and show neither one of $V(\boldsymbol{X}) = V(\boldsymbol{Y})$ or $V(\boldsymbol{X}) < V(\boldsymbol{Y}) < V(S(\boldsymbol{X}))$ can be satisfied. Were it the case that $V(\boldsymbol{X}) = V(\boldsymbol{Y})$ we know by (D) that $\boldsymbol{X} = \boldsymbol{Y}$, which is a contradiction. Instead, if it were the case $V(\boldsymbol{X}) < V(\boldsymbol{Y}) < V(S(\boldsymbol{X}))$ then we know by (B) that $V(S(\boldsymbol{X})) = V(\boldsymbol{X}) + 1$, however, there are no integers strictly between $V(\boldsymbol{X})$ and $V(\boldsymbol{X}) + 1$, thus $V(\boldsymbol{Y})$ can not satisfy this inequality. Thus, there can be no (UV3) solution pairs.

Since `UniqueEOPL` is a total search problem returning one of either a (U1) solution or a (UV1-3) solution, and we have demonstrated no (UV1-3) solutions exist, we know the result of our search must be a solution of type (U1). By (E) this solution must be a finished run-like recursive switching flow. Thus, it is also a valid solution of our original `Search Recursive Arrival` instance. $\square$