Combining Refinement of Parametric Models with Goal-Oriented Reduction of Dynamics

Stefan Haar, Juraj Kolčák, Loïc Paulevé

Combining Refinement of Parametric Models with Goal-Oriented Reduction of Dynamics*

Stefan Haar[1], Juraj Kolčák[1,2], Loïc Paulevé[3,4]

[1] LSV, CNRS & ENS Paris-Saclay, Université Paris-Saclay, France
[2] National Institute of Informatics, Tokyo, Japan
[3] LRI UMR 8623, Univ. Paris-Sud – CNRS, Université Paris-Saclay, Orsay, France
[4] Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800, F-33400 Talence, France

Abstract Parametric models abstract part of the specification of dynamical models by integral parameters. They are for example used in computational systems biology, notably with parametric regulatory networks, which specify the global architecture (interactions) of the networks, while parameterising the precise rules for drawing the possible temporal evolutions of the states of the components. A key challenge is then to identify the discrete parameters corresponding to concrete models with desired dynamical properties. This paper addresses the restriction of the abstract execution of parametric regulatory (discrete) networks by the means of static analysis of reachability properties (goal states). Initially defined at the level of concrete parameterised models, the goal-oriented reduction of dynamics is lifted to parametric networks, and is proven to preserve all the minimal traces to the specified goal states. It results that one can jointly perform the refinement of parametric networks (restriction of domain of parameters) while reducing the necessary transitions to explore and preserving reachability properties of interest.

1 Introduction

Various cyber and physical systems are studied by the means of discrete dynamical models which describe the possible temporal evolution of the state of the components of the system. Defining such models requires extensive knowledge on the underlying system for specifying the rules which generate the admissible state transitions over time. Usually, and especially for physical systems, such as biological networks for which discrete models are extensively employed [24,12,8,1,9,18], it is common to lack such precise knowledge, making an accurate specification of discrete models challenging.

With *parametric* models, part of the specification of the rules for generating the discrete transitions is encoded as (integral) parameters. Thus, a parametric

* This work has been partly funded by ANR-FNR project "AlgoReCell" ANR-16-CE12-0034, by Labex DigiCosme (project ANR-11-LABEX-0045-DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02), and by ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER 1603), JST.

model abstracts a set of concrete *parameterised* models, this set being characterised by the domain of parameter values.

In this paper, we focus on *Parametric Regulatory Networks* (PRNs), also known as Thomas Networks [23,4,2,15], which are commonly employed for modelling qualitative dynamics of biological systems. PRNs allow separating biological knowledge on the pairwise interactions (the architecture of the network) from the rules of interplay between the interactions, usually less known.

In the literature, PRNs are mainly used as a basic framework for identifying fully parameterised models (i.e., Boolean and multilevel networks) which satisfy dynamical properties typically generated from experimental data. This identification task, related to so-called model inference and process mining [5,17,16,21], consists in transforming an abstract parametric model into a set of concrete parameterised models verifying desired dynamical properties. For PRNs, state-of-the-art methods rely on parameter enumeration [13], coloured model-checking [14], logic programming and Boolean satisfiability [10,19], and Hoare logic [3].

However, the exhaustive identification of parameters is often limited to small models, as the set of parameterised models can turn out to be too large to be exhaustively enumerated and further analysed.

In [15], we introduced a semantics of PRNs enabling the *refinement* of a PRN by restricting the domain of its parameters without having to enumerate concrete models, keeping them in a compact abstract representation instead. The refinement is performed according to concrete discrete state transitions: the domain of parameters is restricted so that it abstracts all the concrete models in which the state transition is admissible. Essentially, such semantics of PRNs enable efficient exploration of dynamics of a set of parameterised models.

This exploration suffers from the same bottleneck as individual parameterised models: the number of reachable states grows exponentially with the number of components and thus, becomes intractable for large networks. The exploration of the reachable state space is usually performed to verify dynamical properties. Consequently, various *model reduction* methods have been designed on concrete parameterised models to enhance the tractability of their verification [11,22,20,6]: by reducing the transitions to consider, these methods limit the reachable state space to explore while guaranteeing the correctness of the verification.

In this paper, we address the combination of refinement operations on parametric models with model reductions initially defined at the level of concrete individual models. Essentially, the challenge consists in lifting up such model reductions so they can be performed at the abstract level of parametric models, while ensuring the correctness of their refinement.

We focus on reachability properties, i.e., starting in an initial configuration, the ability to eventually reach a given (partial) configuration. On the one hand, we are interested in refining PRNs to accurately identify concrete parameterised discrete network models that verify the reachability property; on the other hand, we want to take advantage of goal-driven exploration of dynamics of parameterised models to ignore transitions which do not influence the reachability of the goal, enhancing the tractability of the analysis.

The refinement of PRNs we consider for reachability properties has been introduced in [15]. It consists in dynamically drawing transitions allowed by at least one concrete model, and subsequently restricts the domain of parameters to exclude models which do not allow the drawn transition. The generation of transitions is done directly from the abstract representation of the set of concrete models, and therefore involves no enumeration of parameterised models.

The goal-oriented model reduction we consider has been introduced in [20] at the level of parameterised network models. Given a reachability property (goal), the method relies on static analysis by abstract interpretation to identify transitions which are not involved in any minimal trace leading to the goal. Here the minimality refers to the absence of a sub-trace. Whereas deciding reachability properties in parametrised models (namely automata networks) is a PSPACE-complete problem [7], the goal-oriented model reduction has a complexity polynomial in the number of components and exponential in the in-degree of components in the networks (components having a direct influence on a single one).

In this article we present a lifting of the goal-oriented reduction from parametrised models to sets of models with shared architecture, represented by parametric models. To this end, we introduce a directed version of PRNs which allow us to efficiently capture model reduction without the need to explicitly enumerate all possible transitions. We conduct the reduction itself on abstract dynamics of PRNs where instead of enumerating all enabled transitions, we only consider the minimal necessary condition for each component to change value.

The introduced reduction method can be applied on-the-fly to speed up reachability checking in parametric models. Thanks to the preservation of *all* minimal traces, it is guaranteed to capture all parametrised models capable of reproducing the coveted behaviour.

Outline Section 2 recalls the definition of parametric regulatory networks, their dynamics, constraints on influences and finally presents a generalised parametrisation set semantics. In Section 3, the goal-oriented model reduction procedure is extended from parametrised models to parametric models. Directed version of PRNs is introduced for this purpose alongside an abstraction of dynamics designed to alleviate the reduction complexity. Section 4 supplies an algorithm for computing a suitable abstraction of PRN dynamics used in the reduction procedure. Finally, Section 5 summarises the results and offers a brief introduction to possible extensions and applications as well as future directions.

Notations We use $\prod$ to build Cartesian products between sets. As the ordering of components matters, $\prod$ is not commutative. Therefore, we write $\prod^{\leq}_{x \in X}$ for the product over elements in X according to a total order $\leq$. To ease notations, when the order is clear from the context, or when either X is a set of integers, or a set of integer vectors, on which we use the lexicographic ordering, we simply write $\prod_{x \in X}$. Given a sequence of n elements $\pi = (\pi_i)_{1 \leq i \leq n}$, we write $\widetilde{\pi} \stackrel{\Delta}{=} \{\pi_i \mid 1 \leq i \leq n\}$ for the set of its elements. Given a vector $v = \langle v_1, \ldots, v_n \rangle$, we write $v_{[i \mapsto y]}$ for the vector equal to v except on the component i, which is equal to y.

2 Parametric Regulatory Networks

Regulatory networks are finite discrete dynamical systems where the components evolve individually with respect to the value of (a few) other components, their regulators. The value of components in regulatory networks ranges in a finite discrete domain, usually represented as $\{0, ..., m\}$ for some $m \in \mathbb{N}$, thus extending Boolean networks [24]. The evolution of components is then defined by discrete functions which associate to the global states of the network the value towards which each component tends.

Thus, defining regulatory networks requires knowledge on which components influence each others, and how the value of each component is computed from the value of its regulators. *Parametric* Regulatory Networks (PRNs) allow to decouple this specification by having on the one hand a fixed architecture of the network, so-called *influence graph*, and on the other hand discrete parameters, which when instantiated specify the functions of the regulatory network.

2.1 Influence graph and constraints

The influence graph encodes the directed interactions between the components of the regulatory networks: a component u having a direct influence on component v means that in some states of the regulatory network, the computation of the value of node v *may* depend on the value of u. Importantly, if the component w has no direct influence of v, then the computation of the value on v never depends on the value of w.

Definition 1 (Influence Graph). *An* influence graph *G is a tuple (V, I) where V is a finite set of n nodes (components) and $I \subseteq V \times V$ is a set of directed edges (influences).*
For each $v \in V$ we denote the set of its regulators *by $n^-(v) \triangleq \{u \in V \mid (u, v) \in I\}$.*

Besides the existence/absence of direct influences between components, it is usual to have some knowledge about the nature of the influences. Two kinds of constraints are generally considered: *signs* and *observability*.

Influence signs are captured by monotonicity constraints. An influence $(u, v) \in I$ is *positive-monotonic*, denoted $+$, if the sole increase of the value of the regulator u cannot cause a decrease of the computed value of the target v. Symmetrically, an influence $(u, v) \in I$ is *negative-monotonic*, denoted $-$, if the sole increase in the value of u cannot cause an increase in the computed value of v. An influence $(u, v) \in I$ is *observable*, denoted o, if there exists a state in which the sole change of the value of u induces a change of the computed value of v. Thus observability enforces that u does have an influence on the value of v, in some states of the regulatory network. Remark that observability does not imply positive/negative monotonicity – e.g., when the value of v is computed as the exclusive disjunction XOR between its own value and the value of u.

Let us denote a set of influence constraints for an influence graph (V, I) as $R \subseteq I \times \{+, -, \mathrm{o}\}$. An example of influence constraint set is given in Figure 1 (a) as labels on edges of the influence graph.

2.2 Parametrisation

Let us consider an influence graph $G = (V, I)$ among n components and a set of influences constraints R. Let us denote by $m \in \mathbb{N}^n$ the vector specifying the maximum discrete value of each component: the states of the regulator networks span $\prod_{v \in V} \{0, \ldots, m_v\}$. The computation of the value of each component of a regulatory network is constrained by G, R and m. In particular, G imposes that the value of a component depends only on its regulators.

A *regulator state* ω of a component $v \in V$ is a vector specifying the value of each regulator of v. We denote the set of all regulator states of a component as $\Omega_v \triangleq \prod_{u \in n^-(v)} \{0, \ldots, m_u\}$. Intuitively, a regulator state of a component v is a projection of a global state of the network (states of all components) to just the regulators of v, that fully determine its evolution.

A *parameter* $\langle v, \omega \rangle$ then represents a target value towards which component $v \in V$ evolves in regulator state $\omega \in \Omega_v$. We denote the set of all parameters as $\Omega \triangleq \bigcup_{v \in V} \{v\} \times \Omega_v$

A *parametrisation* P is a vector assigning a value to each parameter. The set of all parametrisations associated to an influence graph G and a maximum value vector m is therefore given by $\mathbb{P}(G_m) = \prod^{\lhd}_{\langle v,\omega \rangle \in \Omega} \{0, \ldots, m_v\}$ where $\lhd$ is an arbitrary, but fixed total order on parameters. The set of all parametrisations satisfying both the influence graph $G = (V, I)$ and influence constraints R with maximum value vector m is then defined as:

$$\begin{aligned}
\mathbb{P}(G^R_m) \triangleq \{P \in \textstyle\prod^{\lhd}_{\langle v,\omega \rangle \in \Omega} \{0, \ldots, m_v\} \mid \forall u, v \in V,& \\
(u, v, +) \in R \Rightarrow \forall \omega \in \Omega_v, \forall k \in \{1, \ldots, m_u\} : P_{v,\omega_{[u \mapsto k]}} &\geq P_{v,\omega_{[u \mapsto k-1]}} \\
(u, v, -) \in R \Rightarrow \forall \omega \in \Omega_v, \forall k \in \{1, \ldots, m_u\} : P_{v,\omega_{[u \mapsto k]}} &\leq P_{v,\omega_{[u \mapsto k-1]}} \\
(u, v, \mathrm{o}) \in R \Rightarrow \exists \omega \in \Omega_v, \exists k \in \{1, \ldots, m_u\} : P_{v,\omega_{[u \mapsto k]}} &\neq P_{v,\omega_{[u \mapsto k-1]}}\}
\end{aligned}$$

2.3 Parametric Regulatory Networks

A *Parametric Regulatory Network* (PRN) gathers an influence graph G, influence constraints R, and maximum value vector m, to which can then be associated a subset of parametrisations $\mathbb{P}(G^R_m)$. A (*parametrised*) regulatory network can then be defined by a couple (G^R_m, P) where $P \in \mathbb{P}(G^R_m)$.

Definition 2. *A* parametric regulatory network *(PRN) is a tuple (G, m, R), written G^R_m, where G is an influence graph between n components, R is a set of influence constraints, and $m \in \mathbb{N}^n$ is a vector of the maximum values of each component.*

The set of states *of G^R_m is denoted by $S(G^R_m) \triangleq \prod_{v \in V} \{0, \ldots, m_v\}$.*

The set of (local) transitions *of G^R_m is denoted by: $\Delta(G^R_m) \triangleq \{(v_i \to v_j, \omega) \mid v \in V \wedge\ \omega \in \Omega_v \wedge i, j \in \{0, \ldots, m_v\} \wedge |i - j| = 1\}$. We use $V((v_i \to v_j, \omega)) = v$ to denote the component whose value is changed by transition $(v_i \to v_j, \omega) \in \Delta(G^R_m)$. Furthermore, $s((v_i \to v_j, \omega)) = s(v_i \to v_j) = j - i$ denotes the sign of the transition (value change).*

A transition $(v_i \to v_j, \omega) \in \Delta(G_m^R)$ is *enabled* in state $x \in S(G_m^R)$ if $x_v = i$ and $\omega_v(x) = \omega$, where $\omega_v(x)$ is the projection of state x to the regulators of v. Given a state x and a transition t enabled in x, $x \cdot t$ denotes the state $x_{[v \mapsto j]} \in S(G_m^R)$ obtained by firing transition t in x.

Finally, a transition $(v_i \to v_j, \omega) \in \Delta(G_m^R)$ is *enabled* by a parametrisation set $\mathcal{P} \subseteq \mathbb{P}(G_m^R)$ if there exists a parametrisation $P \in \mathcal{P}$ such that the parameter value $P_{v,\omega} = j + a \cdot s(v_i \to v_j)$ for some $a \in \mathbb{N}_0$.

Example 1. An example of a PRN G_m^R composed of an influence graph $G = (V, I)$ and vector $m = \{1\}^{|V|}$ is depicted in Figure 1. Based on the influences in G and maximum values in m, all regulator states of each component, which correspond to parameters of G_m^R, are determined. The table in Figure 1 (b) lists all the parameters alongside an example parametrisation $P \in \mathbb{P}(G_m^R)$. G_m^R combined with P identifies a unique parametrised network (G_m^R, P). The dynamics of (G_m^R, P) are given in Figure 1 (c).

We capture the basic semantics of a PRN by traces, which correspond to different possible behaviours of the network.

Definition 3 (Trace). *Given a PRN G_m^R and set of parametrisations $\mathcal{P} \subseteq \mathbb{P}(G_m^R)$, a finite sequence $\pi = (\pi_1, \ldots, \pi_{|\pi|})$ of transitions in $\Delta(G_m^R)$ is a* trace *of G_m^R starting in state $x \in S(G_m^R)$ iff $\forall i \in \{1, \ldots, |\pi|\} : \pi_i$ is enabled in state $x \cdot \pi_1 \cdot \ldots \cdot \pi_{i-1}$ and by parametrisation set $\mathcal{P}$.*

To simplify notation, we use ${}^\bullet\pi = x$ and $\pi^\bullet = x \cdot \pi_1 \cdot \ldots \cdot \pi_{|\pi|}$. Moreover, let $\pi^{:i} = (\pi_1, \ldots, \pi_i)$, $\pi^{i:} = (\pi_i, \ldots, \pi_{|\pi|})$ and $\pi^{i:j} = (\pi_i, \ldots, \pi_j)$ denote the prefix, suffix *and* infix *sub-traces of π respectively.*

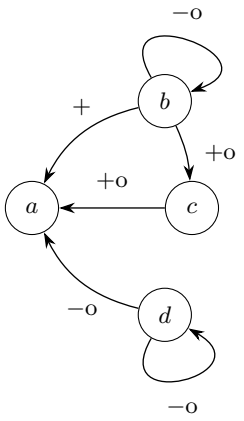
With $P \in \mathbb{P}(G_m^R)$ and $\mathcal{P} = \{P\}$, the above definition gives the traces of the parametrised regulatory network (G_m^R, P). In the general case, each transition is independently enabled with respect to any parametrisation in $\mathcal{P}$.

2.4 Parametrisation Set Semantics

With the basic PRN semantics (Definition 3), two transitions are allowed to fire consequently in a single trace despite no single parametrisation enabling both of them. To forbid such behaviours, semantics have been introduced for PRNs that associate each trace (set of transitions) with a set of parametrisations [15]. Said trace can then be extended only by transitions enabled under some parametrisation from the associated set.

The purpose of parametrisation set semantics is to discriminate transitions based on their causal history. This is done by progressive restriction of the set of parametrisations to *admissible* parametrisations. Following [15], we consider a parametrisation P to be admissible if all transitions in the set (causal history) are enabled under P. However, we allow for a more lenient definition to make room for over-approximation.

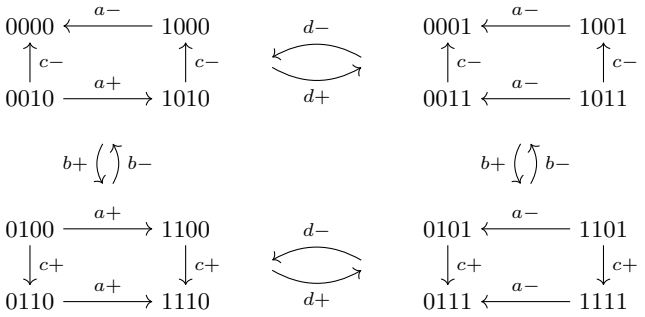
Definition 4 (Parametrisation Set Semantics). *Given a PRN G_m^R, a function $\Psi : 2^{\Delta(G_m^R)} \to 2^{\mathbb{P}(G_m^R)}$ is a* parametrisation set semantics *of G_m^R iff:*



(a) The influence graph G.

$P_{a,\langle b=0,c=0,d=0\rangle}$	$\in \{0,\ldots,m_a\}$	0
$P_{a,\langle b=1,c=0,d=0\rangle}$		1
$P_{a,\langle b=0,c=1,d=0\rangle}$		1
$P_{a,\langle b=1,c=1,d=0\rangle}$		1
$P_{a,\langle b=0,c=0,d=1\rangle}$		0
$P_{a,\langle b=1,c=0,d=1\rangle}$		0
$P_{a,\langle b=0,c=1,d=1\rangle}$		0
$P_{a,\langle b=1,c=1,d=1\rangle}$		0
$P_{b,\langle b=0\rangle}$	$\in \{0,\ldots,m_b\}$	1
$P_{b,\langle b=1\rangle}$		0
$P_{c,\langle b=0\rangle}$	$\in \{0,\ldots,m_c\}$	0
$P_{c,\langle b=1\rangle}$		1
$P_{d,\langle d=0\rangle}$	$\in \{0,\ldots,m_d\}$	1
$P_{d,\langle d=1\rangle}$		0

(b) All the parameters of PRN G_m^R with an example parametrisation P.



(c) States and transitions of (G_m^R, P) depicted as nodes and edges of a state space graph respectively. Since components b and d may update values independently of other components (i.e. in any state), their value changes are only displayed schematically to improve readability.

Figure 1: Influence graph with influence constraints as labels, parameters and dynamics of a possible parametrisation of PRN G_m^R.

1. $\forall T \subseteq \Delta(G_m^R) : \{P \in \mathbb{P}(G_m^R) \mid \forall t \in T : P \textit{ enables } t\} \subseteq \Psi(T)$,
2. $\forall T, T' \subseteq \Delta(G_m^R) : T \subseteq T' \Rightarrow \Psi(T') \subseteq \Psi(T)$.

A trace π of the PRN G_m^R is realisable *according to the parametrisation set semantics if and only if $\Psi(\widetilde{\pi}) \neq \emptyset$.*

A best abstraction Ψ_C producing parametrisation sets of exactly all the parametrisations that allow each transition in the input set has been defined in [15]. To facilitate practical application, as the number of parametrisations may be in the worst case double exponential in the number of components, [15] has

tackled the semantics Ψ_A over-approximating parametrisation sets by convex covers, keeping track of only a maximal and minimal element and thus avoiding the need to enumerate parametrisations explicitly. More formally, let us first reintroduce the *parametrisation order*.

Definition 5. *The* parametrisation order *on vectors of length k is the partial order $\leq$ defined as follows:*

$$a \leq b \overset{A}{\Leftrightarrow} \forall i \in \{0, \ldots, k-1\} : a_i \leq b_i$$

The parametrisation set given by Ψ_A is a couple of parametrisations (L, U) representing the lower and upper bound. Formally, $(L, U) = \{P \in \mathbb{P}(G_m^R) \mid L \leq P \wedge U \geq P\}$ is a bounded convex sublattice of all vectors of length $|\Omega|$ with the parametrisation order. In [15], a method has been provided to compute *the tightest lower and upper bounds* for a given set of transitions and influence constraints without the need to explicitly enumerate the parametrisations.

Naturally, checking whether a particular parametrisation belongs to the abstracted set can be done simply by comparing it with the bounds. Similarly, determining whether a transition is enabled (by any parametrisation) can be done without explicit enumeration of the parametrisations. In fact, it is enough to compare against the corresponding parameter value of the relevant bound, e.g. $U_{v,\omega} \geq k+1$ is the sufficient and necessary condition for the transition $(v_k \to v_{k+1}, \omega)$ to be enabled.

In this article, we consider any parametrisation set semantics compatible with Definition 4, however, special attention is given to Ψ_A as it can be used with the restriction method without the need to enumerate the parametrisations explicitly.

3 Goal-Oriented Reduction

In this section, we extend the goal-oriented model reduction procedure from parametrised models (in particular, automata networks) [20] to the parametric models.

3.1 Minimal Traces

Given a PRN G_m^R and a state $x \in S(G_m^R)$, we say a value $\top \in \{0, \ldots, m_g\}$ of a component $g \in V$ is *reachable* from x iff either $x_g = \top$ or there exists a realisable trace π with ${}^\bullet\pi = x$ and $\pi^\bullet{}_g = \top$.

We are interested in reachability by *minimal* traces. Adapted from [20], a realisable trace is minimal for $g_\top$ reachability if there exists no other realisable trace reaching $g_\top$ with a subsequence of transitions.

Definition 6 (Minimal Trace). *Given a parametrised PRN (G_m^R, P), a trace π of (G_m^R, P) is* minimal *w.r.t. reachability of goal $g_\top$ from state x if and only if*

there exists no other trace ρ *satisfying* $x = {}^\bullet\rho$, ${\rho^\bullet}_g = \top$, $|\rho| < |\pi|$ *and existence of an injection* $\phi : \{1, \ldots, |\rho|\} \to \{1, \ldots, |\pi|\}$ *such that* $\forall i, j \in \{1, \ldots, |\rho|\} : i \leq j \Rightarrow \phi(i) \leq \phi(j)$ *and* $\rho_i = \pi_{\phi(i)}$.

An important property of minimal traces is their independence on the exact parametrisation. More precisely, using parametrisation set semantics, if a trace is minimal for at least one parametrisation, then it is minimal for any other parametrisation under which it is enabled.

Property 1 (Parametrisation Independence of Minimal Traces). Let G^R_m be a PRN and π a realisable trace minimal in (G^R_m, P) for some $P \in \Psi(\widetilde{\pi})$. Then, π is minimal in any (G^R_m, P') where $P' \in \Psi(\widetilde{\pi})$

Proof. $P' \in \Psi(\widetilde{\pi})$ guarantees π is a proper trace of (G^R_m, P'). We conduct the rest of the proof by contradiction. Let thus ρ be a trace in (G^R_m, P') satisfying the conditions in Definition 6. From the existence of the injection ϕ we get $\widetilde{\rho} \subseteq \widetilde{\pi}$ and from the definition of parametrisation set semantics $\Psi(\widetilde{\pi}) \subseteq \Psi(\widetilde{\rho})$. ρ is therefore realisable in (G^R_m, P) meaning that π is not minimal in (G^R_m, P) which is a contradiction.

Property 1 allows us to speak of a realisable trace of a PRN as minimal without the need to explicitly state the parametrisation which is witness to the minimality.

3.2 Directed Parametric Regulatory Networks

The goal-oriented reduction for parametrised models is facilitated by pruning transitions which are guaranteed to not be used by any minimal trace reaching the goal [20]. The PRN definition could be extended to allow for pruning the transition set to subsets $T \subseteq \Delta(G^R_m)$. Unlike the case of general parametrised models, however, the transitions of a PRN only allow to change the value of a component by steps of size 1. As such, if a transition increasing the value of a component $v \in V$ to $k \in \{0, \ldots, m_v\}$ is to be pruned, all transitions increasing the value of v beyond k can surely be pruned as well, and symmetrically for decreasing transitions. Thus, instead of removing individual transitions of PRNs, we disable increasing, respectively decreasing, value of a component in a given regulator state beyond a certain value (or entirely). This is facilitated by keeping record of the activation (increase) and inhibition (decrease) limits for each component in vectors l^A and l^I respectively.

Definition 7 (Directed Parametric Regulatory Network). *A* directed parametric regulatory network *(DPRN) is a tuple* $\mathcal{G} = (G^R_m, l^A, l^I)$*, where* G^R_m *is a parametric regulatory network,* $l^A \in (\mathbb{N} \cup \{-\infty\})^{|\Omega|}$ *is a vector of activation limits for each regulator state* $\omega \in \Omega$ *and* $l^I \in (\mathbb{N}_0 \cup \{\infty\})^{|\Omega|}$ *is a vector of inhibition limits for each regulator state* $\omega \in \Omega$.

The set of states of $\mathcal{G}$ *is equal to the set of states of the underlying PRN:* $S(\mathcal{G}) = S(G^R_m)$.

The set of transitions of $\mathcal{G}$ is a subset of the PRN transitions satisfying the activation and inhibition limits l^A and l^I respectively. Formally, $\Delta(\mathcal{G}) \subseteq \Delta(G_m^R)$ such that:

$$\forall t = (v_i \to v_j, \omega) \in \Delta(G_m^R) : t \in \Delta(\mathcal{G}) \overset{\Delta}{\Leftrightarrow} \begin{cases} i < {l^A}_\omega & \text{if } s(t) = +1 \\ i > {l^I}_\omega & \text{if } s(t) = -1 \end{cases}$$

One may remark that by using parametrisation set semantics, it is already possible to restrict the activation or inhibition of components in individual regulator states while just using PRNs. While it is true that an equivalent set of enabled transitions can be achieved both by restricting the parametrisation set and by DPRN, the semantics of the two restrictions are different.

The parametrisation set semantics serves primarily to keep track of parametrisations capable of reproducing certain behaviour(s), and thus restrict the set of enabled transitions based on their causal history. On the other hand, the l^A and l^I of DPRN mark components whose activation or inhibition (beyond a certain value) is not necessary to reach a given goal (via a minimal trace). A parametrisation that allows changing a component value beyond the limit, thus allowing behaviour which does not lead to the established goal may still allow a different sequence of transitions leading to the goal. We want to retain such parametrisations, thus the "useless" behaviour which does not lead to the goal cannot be restricted in the parametrisation set semantics. Therefore, keeping the information about parametrisations and about the activation and inhibition limits independently is key.

The complete independence of parametrisation set semantics and the limit vectors l^A and l^I allows us to employ both in parallel. The extension of both traces (Definition 3) and parametrisation set semantics (Definition 4) from PRNs to DPRNs is thus natural.

3.3 Objectives

The reduction for parametrised models relies on identifying sub-goals, or *objectives*, local in terms of individual components. We reintroduce the concept of a (local) objective for the parametric model.

Definition 8 (Objective). *Given an DPRN $\mathcal{G}$, an* objective $v_i \rightsquigarrow v_j$ *is a pair of values $i, j \in \{0, \ldots, m_v\}$ of a component $v \in V$.*

An objective $v_i \rightsquigarrow v_j$ is valid *in a starting state $x \in S(\mathcal{G})$ iff $i = j$ or a realisable trace π of the parametrised DPRN exists, such that ${}^\bullet\pi = x$, ${\pi^\bullet}_v = j$ and $\exists k \in \{0, \ldots, |\pi| - 1\} : {}^\bullet\pi_{kv} = i$.*

$i \rightsquigarrow j$ is used to denote $v_i \rightsquigarrow v_j$ if the component $v \in V$ is obvious from the context.

Each objective $v_i \rightsquigarrow v_j$ captures either increase or decrease of the value of the component. Formally, the sign of an objective $s(v_i \rightsquigarrow v_j) \overset{\Delta}{=} \text{SIGN}(j - i)$.

By requiring the witness of objective validity to be a realisable trace instead of just a trace of enabled transitions, we retain only behaviours which are present in at least one parametrised model.

The objective represents a change of value of only one component $v \in V$. A realisable trace reproducing such a change may, however, require to also change value of other components, namely the regulators of v. Each objective is thus associated to a set of transitions which may be used to complete it, and from which the required regulator values can be obtained.

3.4 Regulation Cover Sets

Depending on the parametrisation set semantics, it may be a common occurrence for a particular value change to be enabled by numerous regulator states (recall that enabling is existential w.r.t. parametrisations). Such cases lead to a substantial redundancy in individual transition enumeration as the value of only a subset of regulators may be enough to determine whether a value change is enabled or not. To this end we introduce a definition of a partial regulator state, which is used to represent a (minimal) condition for a value change to be enabled.

Definition 9 (Partial Regulator State). *A* partial regulator state $\aleph$ *of component* $v \in V$ *is a vector* $\aleph \in \prod_{u \in n^-(v)} \{0, \ldots, m_u\} \cup \{*\}$ *assigning a value or a wildcard character* $*$ *to each regulator* u *of* v*. By abuse of notation,* $\aleph$ *is also a set of regulator states, more precisely* $\aleph \subseteq \Omega_v$ *such that for all* $\omega \in \Omega_v$*:*

$$\omega \in \aleph \overset{\Delta}{\Leftrightarrow} \forall u \in n^-(v) : \omega_u = \aleph_u \vee \aleph_u = *$$

The set of all partial regulator states of $v \in V$ *is denoted as* $\mathcal{A}_v$*.*

Partial regulator states can be utilised to abstract the DPRN dynamics while minimising the number of repetitions of each value of each regulator. We capture these abstractions by the means of sets of partial regulator states, called *regulation cover sets*, representing the enabling condition of a given value change. We impose two conditions on regulation cover set of value change c. First, the set has to cover all regulator states ω such that (c, ω) is enabled. In other words, for each such regulator state there must exist one or more partial regulator states which specify the value of each regulator in ω. Second, no bad regulator state ω such that (c, ω) is not enabled is subsumed by any of the partial regulator states in the cover set. These two conditions not only guarantee that the abstract dynamics enable exactly the same value changes as the concrete dynamics, but also preserve the regulator information, i.e. each value of each regulator that appears in the enabling conditions. The regulator information is necessary to accurately determine which regulator values are necessary to complete an objective.

Definition 10 (Regulation Cover Set). *Let* $\mathcal{G}$ *be a DPRN and* $\mathcal{P}$ *a parametrisation set from the parametrisation set semantics, and let* $c = v_i \to v_j$ *be an arbitrary value change of a component* $v \in V$*. A set of partial regulator states* $\mathcal{A}_c \subseteq \mathcal{A}_v$ *is a cover set of* c *iff the following is satisfied:*

– $\forall \omega \in \Omega_v : (c, \omega)$ *is enabled under* $\mathcal{P}$*:* $\forall u \in n^-(v) : \exists \aleph \in \mathcal{A}_c : \omega \in \aleph \wedge \omega_u = \aleph_u$.
– $\forall \omega \in \Omega_v : (c, \omega)$ *is not enabled:* $\forall \aleph \in \mathcal{A}_c : \omega \notin \aleph$.

Any regulation cover set, including the concrete regulation cover set $\{\omega \mid \omega \in \Omega_v : (c, \omega)$ is enabled$\}$, may be used for the purposes of the reduction procedure. The aim of the regulation cover set being to minimise the number of individual regulator values which appear across all of the partial regulator states, an algorithm that computes regulator cover sets with no more regulator value specifications than the concrete regulation cover set is introduced in Section 4.

3.5 Reduction of Directed Parametric Regulatory Networks

Our reduction procedure essentially relies on associating to objectives the set of (partial) transitions which are necessary to realise the objective within the corresponding components of the PRN. Starting from the final (goal) objective, the procedure then recursively collects objectives related to the identified transitions.

Since PRNs allows only unitary value changes, the realisation of an objective $v_i \rightsquigarrow v_j$ involves a monotonic change of value of component v from i to j, where each change of value depends on specific (partial) regulator state. This coupling of a value change with a corresponding partial regulator state is referred to as a *partial transition.*

Definition 11 (Objective Transition Set). *Let $\mathcal{G}$ be an DPRN parametrised by $\mathcal{P}$, and let $v_i \rightsquigarrow v_j$ be an objective for $v \in V$. The* objective transition set *$\tau(v_i \rightsquigarrow v_j)$ is defined as $\tau(v_i \rightsquigarrow v_j) \triangleq \emptyset$ whenever $i = j$, otherwise,*

$$\tau(v_i \rightsquigarrow v_j) \triangleq \{(v_k \to v_q, \aleph) \mid s(v_k \to v_q) = s(v_i \rightsquigarrow v_j) \wedge \aleph \in \mathcal{A}_{v_k \to v_q} \\ \wedge \text{MAX}\{k, q\} \leq \text{MAX}\{i, j\} \wedge \text{MIN}\{k, q\} \geq \text{MIN}\{i, j\}\}$$

Given an initial state $x \in S(\mathcal{G})$, the valid objective transition set *of an objective $v_i \rightsquigarrow v_j$ in state x is a subset of the objective transition set $\tau_x(v_i \rightsquigarrow v_j) \subseteq \tau(v_i \rightsquigarrow v_j)$ such that: $(c, \aleph) \in \tau_x(v_i \rightsquigarrow v_j) \overset{\Delta}{\Leftrightarrow} \forall u \in n^-(v) : \aleph_u \neq * \Rightarrow x_u \rightsquigarrow \aleph_u$ is valid in state x.*
The (valid) objective transition sets extend to sets of objectives in the natural manner: $\tau(\mathcal{O}) = \bigcup_{O \in \mathcal{O}} \tau(O)$.

Remark that the definition of a valid objective transition set benefits from the use of partial regulator states. Indeed, instead of having to check validity of an objective for each regulator, only the minimal necessary subset of regulators is considered. Checking objective validity consists of searching for a realisable trace, which translates to finding all possible extensions (enabled transitions) of a trace. As enabled transitions can be retrieved using Ψ_A without explicitly enumerating the parametrisations, the validity check is compatible with Ψ_A.

The goal-oriented reduction of DPRNs can then be defined by recursively collecting objectives from partial transitions ($\mathcal{B}$) and refining the component activation and inhibition limits accordingly.

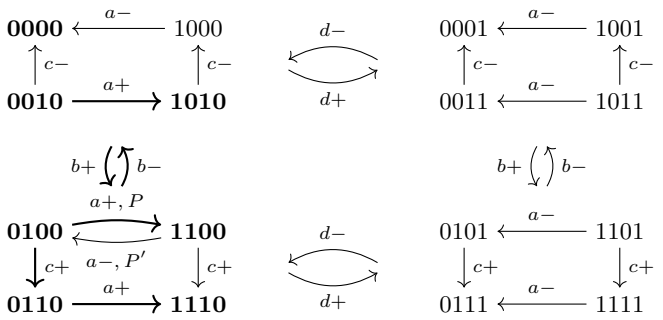


Figure 2: States and transitions of $(\mathcal{G}, \{P, P'\})$ depicted as nodes and edges of a state space graph respectively. Transitions changing the value of b and d are displayed schematically. Transitions only enabled by a single of the parametrisations are marked accordingly. Bold font and lines indicate states and transitions used by at least one minimal trace from the initial state to the goal.

Definition 12 (Reduction Procedure). *The goal-oriented reduction of a DPRN $\mathcal{G} = (G_m^R, l^A, l^I)$ for an initial state $x \in S(\mathcal{G})$ and a goal $g_\top$ is the DPRN $\mathcal{G}' = (G_m^R, l^{A'}, l^{I'})$ with $l^{A'}$ and $l^{I'}$ being defined as follows, $\forall v \in V, \forall \omega \in \Omega_v$:*

$$l^{A'}{}_\omega = \text{MAX}(\{k \in \{0, \ldots, m_v\} \mid \exists (v_{k-1} \to v_k, \aleph) \in \tau_x(\mathcal{B}) : \omega \in \aleph\} \cup \{-\infty\})$$
$$l^{I'}{}_\omega = \text{MIN}(\{k \in \{0, \ldots, m_v\} \mid \exists (v_{k+1} \to v_k, \aleph) \in \tau_x(\mathcal{B}) : \omega \in \aleph\} \cup \{\infty\})$$

where $\mathcal{B}$ is the smallest set of objectives satisfying the following:

1. $x_g \rightsquigarrow \top \in \mathcal{B}$
2. $\forall O \in \mathcal{B} : \forall (v_k \to v_q, \aleph) \in \tau_x(O) : \forall u \in n^-(v) \setminus \{v\} : \aleph_u \neq * \Rightarrow x_u \rightsquigarrow \aleph_u \in \mathcal{B}$
3. $\forall O \in \mathcal{B} : \forall (v_k \to v_q, \aleph) \in \tau_x(O) : \forall v_i \rightsquigarrow v_j \neq O \in \mathcal{B} : v_q \rightsquigarrow v_j \in \mathcal{B}$.

Example 2. Consider the parametric regulatory network G_m^R introduced in Example 1 converted to a DPRN $\mathcal{G} = (G_m^R, l^A, l^I)$ in an unrestrictive manner ($l^A = \{1\}^V$ and $l^I = \{0\}^V$), and a parametrisation set containing only two parametrisations $\mathcal{P} = \{P, P'\}$, where P is the parametrisation from Example 1 and P' differs from P only in value of ${P'}_{a,\langle b=1,c=0,d=0\rangle} = 0$. Furthermore, let $a = 1$ be a goal and $x = \langle a = 0, b = 0, c = 0, d = 0\rangle$ an initial state.

In Figure 2 we recall the dynamics of $\mathcal{G}$ given as a state space graph. Note that the second parametrisation P' is also shown within the graph as opposed to the one in Example 1.

In our example, there are three minimal traces from the initial state x reaching the goal $a = 1$:

$$\langle 0000\rangle \xrightarrow{b+,\langle 0\rangle} \langle 0100\rangle \xrightarrow{a+,\langle 100\rangle} \langle 1100\rangle$$
$$\langle 0000\rangle \xrightarrow{b+,\langle 0\rangle} \langle 0100\rangle \xrightarrow{c+,\langle 1\rangle} \langle 0110\rangle \xrightarrow{a+,\langle 110\rangle} \langle 1110\rangle$$
$$\langle 0000\rangle \xrightarrow{b+,\langle 0\rangle} \langle 0100\rangle \xrightarrow{c+,\langle 1\rangle} \langle 0110\rangle \xrightarrow{b-,\langle 1\rangle} \langle 0010\rangle \xrightarrow{a+,\langle 010\rangle} \langle 1010\rangle$$

All the listed traces share a common prefix, however, they are all minimal as a different regulator state is used to activate a each time, thus each of the traces has at least one unique transition. One may further remark that the first (shortest) minimal path is only available under parametrisation P, however, thanks to Property 1 this has no impact on the reduction procedure itself.

Observe that node d never activates in any of the minimal traces. This follows from the fact that a is never allowed to activate while d is active. Thus, if d activates it has to deactivate again before the goal can be reached. As d has no impact on the value of the other components besides a, such an activation and deactivation loop can always be stripped from the trace to obtain a smaller trace, unlike the loop by b in the third (longest) trace, which is necessary for the activation of c. One might thus expect the activation of d to be pruned during the reduction procedure, which is, indeed the case:

We start with $\mathcal{B} := \{a_0 \rightsquigarrow a_1\}$ according to rule (1) of Definition 12.

Inference of the regulator cover set used for $\tau_x(a_0 \rightsquigarrow a_1) = \{(a_0 \to a_1, \aleph) \mid \aleph \in \mathcal{A}_{a_0 \to a_1}\} = \{(a_0 \to a_1, \langle 100\rangle), (a_0 \to a_1, \langle 010\rangle), (a_0 \to a_1, \langle 110\rangle)\}$ is illustrated in Example 3. Then, by rule (2) of Definition 12, the following objectives are included in $\mathcal{B} := \mathcal{B} \cup \{b_0 \rightsquigarrow b_0, b_0 \rightsquigarrow b_1, c_0 \rightsquigarrow c_0, c_0 \rightsquigarrow c_1, d_0 \rightsquigarrow d_0\}$.

For arbitrary component v, the objective $v_0 \rightsquigarrow v_0$ has an empty valid transition set $\tau_x(v_0 \rightsquigarrow v_0) = \emptyset$ and thus neither of rules (2) or (3) are applicable. For the the remaining $b_0 \rightsquigarrow b_1$ and $c_0 \rightsquigarrow c_1$ rule (2) produces only duplicate objectives ($b_0 \rightsquigarrow b_0$ and $b_0 \rightsquigarrow b_1$, respectively). Rule (3), however, may be applied to $b_0 \rightsquigarrow b_1$ and $c_0 \rightsquigarrow c_1$ to bridge them to $b_0 \rightsquigarrow b_0$ and $c_0 \rightsquigarrow c_0$, respectively, to include objectives $\mathcal{B} := \mathcal{B} \cup \{b_1 \rightsquigarrow b_0, c_1 \rightsquigarrow c_0\}$.

Only duplicate objectives are obtained by application of either rule (2) or (3) on the newly added $b_1 \rightsquigarrow b_0$ and $c_1 \rightsquigarrow c_0$. Thus, the reduction concludes with $\mathcal{B} = \{a_0 \rightsquigarrow a_1, b_0 \rightsquigarrow b_0, b_0 \rightsquigarrow b_1, b_1 \rightsquigarrow b_0, c_0 \rightsquigarrow c_0, c_0 \rightsquigarrow c_1, c_1 \rightsquigarrow c_0, d_0 \rightsquigarrow d_0\}$, with valid transition set $\tau_x(\mathcal{B}) = \{(a_0 \to a_1, \langle 100\rangle), (a_0 \to a_1, \langle 010\rangle), (a_0 \to a_1, \langle 110\rangle), (b_0 \to b_1, \langle 0\rangle), (b_1 \to b_0, \langle 1\rangle), (c_0 \to c_1, \langle 1\rangle), (c_1 \to c_0, \langle 0\rangle)\}$. One may observe that the computed transition set indeed covers all the transitions used by any of the minimal traces (thick edges in Figure 2).

Finally, the limit vectors for the new DPRN $\mathcal{G}' = (G_m^R, l^{A'}, l^{I'})$ are computed as follows:

$$l^{A'} = \langle a = 1, b = 1, c = 1, d = -\infty\rangle$$
$$l^{I'} = \langle a = \infty, b = 0, c = 0, d = \infty\rangle$$

Observe that component d is indeed completely forbidden from acting in the reduced model, considerably decreasing the reachable state space that has to be explored. Notice that deactivation of a is also disabled, however, in our Boolean case this has no practical effect w.r.t. reachability of $a = 1$.

3.6 Correctness

Following the interpretation of the reduction procedure and thanks to the monotonicity of value updating, a transition (c, ω) remains enabled in $\mathcal{G}'$ iff at least one partial transition $(c, \aleph)$ exists in $\tau_x(\mathcal{B})$ with $\omega \in \aleph$. This leads us to formulate the soundness theorem of the reduction procedure, guaranteeing that all transitions of all minimal traces are preserved and thus, in turn, all minimal traces are preserved.

Theorem 1. *Let $\mathcal{G}$ be a DPRN, and let a realisable trace π of $\mathcal{G}$ be minimal for an initial state $x \in S(\mathcal{G})$ and goal $g_\top$. Then, for any transition $(c, \omega) \in \widetilde{\pi}$ there exists at least one partial transition $(c, \aleph) \in \tau(\mathcal{B})$ such that $\omega \in \aleph$, where $\mathcal{B}$ is constructed according to Definition 12.*

The proof of the theorem relies on showing that any transition which is not preserved is part of a cycle on any trace leading to the goal, and as a consequence does not belong to any minimal trace. The formal proof is given in Appendix A.

4 Regulation Cover Set Inference

In this section we introduce a heuristic for construction of regulation cover sets whose size, w.r.t. specified regulator values across all partial regulator states, does not exceed the size of the concrete regulation cover set.

Let $\mathcal{A}_{\text{ENA}} = \{\aleph \in \mathcal{A}_v \mid \forall \omega \in \aleph : (c, \omega) \text{ is enabled}\}$ be the set of all partial regulators states which contain no bad regulator states. For each $i \in \{0, \ldots, |n^-(v)|\}$ let $\mathcal{A}_i = \{\aleph \in \mathcal{A}_v \mid |\{u \in n^-(v) \mid \aleph_u = *\}| = i\}$ to be the set of all partial regulator states with exactly i regulator values equal to $*$.

The algorithm consists of choosing partial regulator state set, $\mathcal{A}_{\text{EXT}}$, to cover each (concrete) regulator state enabling the value change. This is done separately for each regulator state in an increasing order of a weight function. The weight function represents flexibility of covering the regulator state, i.e. there are more partial regulator states in $\mathcal{A}_{\text{ENA}}$ containing a regulator state with a larger weight than the ones covering regulator state with smaller weight. The weights are dynamic as the partial regulator states get removed ($\mathcal{A}_{\text{RMV}}$) throughout the algorithm. The $\mathcal{A}_{\text{EXT}}$ for each regulator state is computed by testing candidate sets of partial regulator states from $\mathcal{A}_i$ in decreasing order on i. A cover set for each regulator state is guaranteed to exist as for $i = 0$ the candidate set is a singleton set containing the regulator state itself. Once a suitable cover set is found for a particular regulator state, it is included in the regulation cover set $\mathcal{A}_c$ and all partial regulator states containing the regulator state are excluded from further computation.

Algorithm 1 Pseudocode of the algorithm computing regulation cover set.

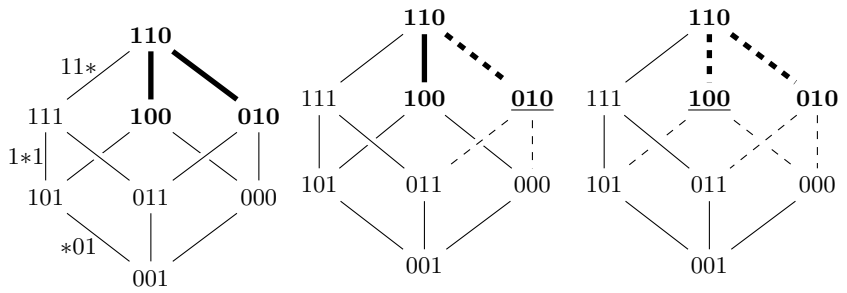
function WEIGHT(ω)
  return $|\{\aleph \in (\mathcal{A}_1 \cap \mathcal{A}_{\text{ENA}}) \setminus \mathcal{A}_{\text{RMV}} \mid \omega \in \aleph\}| + \frac{|\{\aleph \in \mathcal{A}_1 \cap \mathcal{A}_{\text{ENA}} \mid \omega \in \aleph\}|}{|n^-(v)|+1}$
end function

function COMPUTECOVERSET($c = v_k \to v_q$)
  $\mathcal{A}_c \leftarrow \emptyset$
  $\mathcal{A}_{\text{RMV}} \leftarrow \emptyset$
  while $\mathcal{A}_0 \neq \emptyset$ **do**
    $\omega \leftarrow \omega' \in (\mathcal{A}_0 \cap \mathcal{A}_{\text{ENA}}) \setminus \mathcal{A}_{\text{RMV}}$
      with WEIGHT(ω') = MIN$\{$WEIGHT(ω'') $\mid \omega'' \in (\mathcal{A}_0 \cap \mathcal{A}_{\text{ENA}}) \setminus \mathcal{A}_{\text{RMV}}\}$
    $\mathcal{A}_{\text{EXT}} \leftarrow \emptyset$
    $i \leftarrow |n^-(v)| - 1$
    while ω is not covered by $\mathcal{A}_c \cup \mathcal{A}_{\text{EXT}}$ **do**
      $\mathcal{A}_{\text{EXT}} \leftarrow (\mathcal{A}_i \cap \mathcal{A}_{\text{ENA}}) \setminus \mathcal{A}_{\text{RMV}}$
      $i \leftarrow i - 1$
    end while
    $\mathcal{A}_c \leftarrow \mathcal{A}_c \cup \mathcal{A}_{\text{EXT}}$
    $\mathcal{A}_{\text{RMV}} \leftarrow \mathcal{A}_{\text{RMV}} \cup \{\aleph \in \mathcal{A}_v \mid \omega \in \aleph\}$
  end while
  return $\mathcal{A}_c$
end function

As the weight function gives only a partial order on the regulator states, the algorithm is forced to make nondeterministic choices. This occurs, however, only in cases when the choices are isomorphic. As such, the partial order given by weights can be extended to a total order arbitrarily, e.g. by underlying lexicographic order. The pseudocode of the algorithm to construct regulation cover sets is given in Algorithm 1.

The correctness of the algorithm comes directly from the construction. No bad states may be included as the algorithm works only with the set of partial regulator states which include no bad states. On the other hand, all regulator states which enable the value change are fully covered as the algorithm ensures this for each of them individually.

The resulting cover set computed by Algorithm 1 contains no more explicit regulator value specifications than the concrete regulation cover set. This is a consequence of the order of regulator states covering. Suppose a regulator state ω is covered by several partial regulator states which contain more regulator value specifications than ω itself. Each partial regulator state $\aleph \in \mathcal{A}_1$ with $\aleph_u = *$ is shared with exactly $m_u - 1$ other regulator states. Thus, the partial regulator states included to cover ω can be utilised while covering $m_u - 1$ other regulator states. Finally, since WEIGHT(ω) ≥ 2 is the smallest weight among all uncovered regulator states, all the other uncovered regulator states are also sharing partial regulator states among themselves, thus closing the loop and guaranteeing the regulator value specification debt eventually gets "payed off".



(a) Initial configuration (b) Configuration after one iteration. (c) Configuration after two iterations.

Figure 3: Regulator states of component a during computation of regulation cover set for value change $a_0 \to a_1$. Only the leftmost edges in (a) are labelled by the corresponding partial regulator states $11*$, $1*1$ and $*01$ for the sake of readability. Bold text and lines indicate (partial) regulator states which enable the value change ($\mathcal{A}_{\text{ENA}}$). Underlined regulator state is the state covered in the respective iteration and dashed lines represent removed partial regulator states ($\mathcal{A}_{\text{RMV}}$).

The fractional part of the weight function is included to introduce bias towards states that have less partial regulator states in the beginning (due to sharing with more bad states). If there are two regulator states ω and ω' such that $\lfloor \textsc{Weight}(\omega) \rfloor = \lfloor \textsc{Weight}(\omega') \rfloor$ but $\textsc{Weight}(\omega) < \textsc{Weight}(\omega')$, we know that both of them have equally many partial regulator states to choose from for their respective cover sets. However, more of the partial regulator states containing ω' have been removed and thus, quite possibly included in the regulation cover set $\mathcal{A}_c$. ω' is therefore in all likelihood already covered to a higher degree than ω and possibly, has more covering options. The bias thus ensures ω is covered first in order to avoid introducing potentially redundant partial regulator states into the regulation cover set.

Both principles making up the weight function are illustrated in Example 3.

Example 3. Consider the same directed parametric regulatory network $\mathcal{G}$ as in Example 2.

We now show the regulation cover set computation for value changes of component a. Let us start with $a_0 \to a_1$. The initial configuration and first two iterations, consisting of covering of the first two regulator states, of the algorithm are schematically depicted in Figure 3.

Figure 3 lists all regulator states of component a as nodes in a graph. Bold font indicates the three regulator states which enable the increase of a. The partial regulator states from $\mathcal{A}_1$ correspond to edges in the graph, connecting contained regulator states. Thick edges indicate partial regulator states which contain no bad regulator states. Partial regulator states from $\mathcal{A}_2$ could in turn be viewed as squares in the diagram, all of them containing at least one bad

regulator state in our case. In the graphical representation of regulator states, a partial regulator state belonging to $\mathcal{A}_i$ is a i-dimensional hypercube in the Boolean case, or a i-dimensional hyper-rectangular cuboid in the general case.

The graph representation in Figure 3 allows for easy visualisation of the weight function. The weight corresponds to number of thick, non-dashed edges plus, the number of thick edges divided by $|n^-(a)| + 1$, in our case 4. Consequently, in the initial configuration (Figure 3 (a)) the regulator states $\langle 100 \rangle$ and $\langle 010 \rangle$ have equal (minimal) weight. This is justified by their perfectly symmetrical position.

Figure 3 illustrates the run of the algorithm assuming lexicographic order is used to distinguish between regulator states with equal weights. In the first iteration $\langle 010 \rangle$ is covered using itself for the extension set $\mathcal{A}_{\text{EXT}} = \{\langle 010 \rangle\}$ as the only partial regulator state with more unspecified regulator values, $\langle *10 \rangle$, alone does not fully cover $\langle 010 \rangle$. Figure 3 (b) depicts the situation after the first iteration, including the removed partial regulator states (dashed lines).

In the second iteration $\langle 100 \rangle$ is covered in the exact same fashion, owning to the symmetric position w.r.t. $\langle 010 \rangle$. The result is shown in Figure 3 (c).

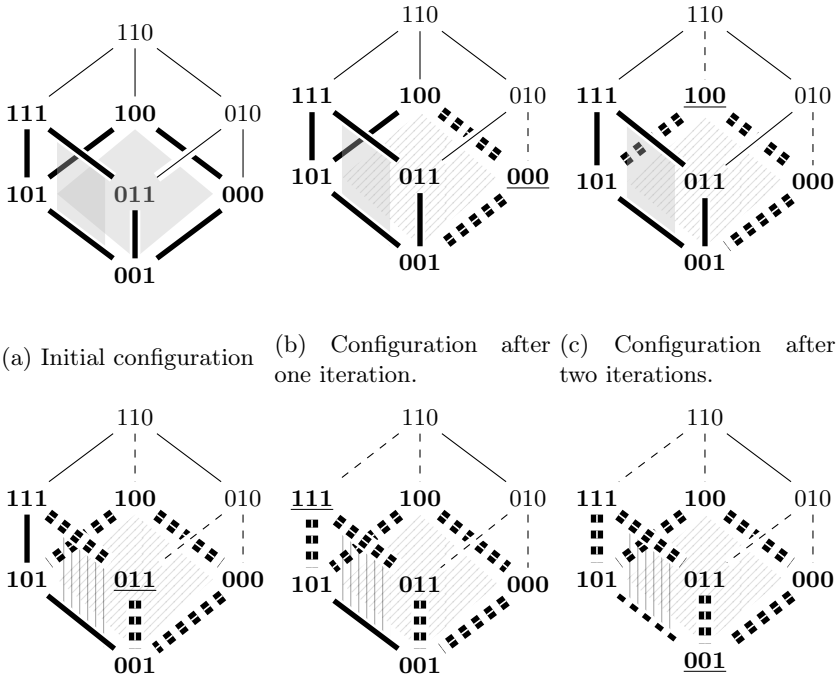
No partial regulator states remain for the last regulator state $\langle 110 \rangle$ except the regulator state itself. Thus, $\langle 110 \rangle$ also gets covered explicitly. The algorithm therefore concludes with the concrete regulation cover set $\mathcal{A}_{a_0 \to a_1} = \{\langle 010 \rangle, \langle 100 \rangle, \langle 110 \rangle\}$, which, in fact, is the optimal solution in our case.

Let us now consider also the decreasing case $a_1 \to a_0$. Again, we illustrate the running of the algorithm using graph representation of the regulator states of a. All iterations up to the final one of the algorithm using lexicographic order on regulator states of equal weight are given in Figure 4.

The algorithm begins with covering the regulator state $\langle 000 \rangle$. Unlike in the case of increasing a, a nonempty candidate extension set exists for partial regulator states on level $\mathcal{A}_2$ containing a single element $\{\langle *0* \rangle\}$. This partial regulator state alone, however, does not suffice to cover $\langle 000 \rangle$ and extension set $\{\langle 00* \rangle, \langle *00 \rangle\}$ is used instead as indicated by double lines in Figure 4 (b). Notice that in this case, the node $\langle 000 \rangle$ gets covered by two partial regulator states having one more regulator value specification (a total of 4 specifications against the explicit 3).

According to the weight function, $\langle 100 \rangle$ gets covered next. $\langle *0* \rangle$ is no longer available, thus the first nonempty candidate extension set is $\{\langle 10* \rangle\}$. Although $\langle 10* \rangle$ alone is not enough to fully cover $\langle 100 \rangle$, the cover set $\mathcal{A}_{a_1 \to a_0}$ already contains $\langle *00 \rangle$ which covers $\langle 100 \rangle$ completely in combination with $\langle 10* \rangle$. Thus, $\langle 100 \rangle$ gets covered by including only 2 additional regulator value specifications, effectively "paying-off" the depth incurred while covering $\langle 000 \rangle$.

Covering $\langle 011 \rangle$ and subsequently $\langle 111 \rangle$ is identical to that of $\langle 000 \rangle$ and $\langle 100 \rangle$. Both of them thus get covered by three partial regulator states $\langle 0*1 \rangle$, $\langle *11 \rangle$ and $\langle 1*1 \rangle$ as shown in Figure 4 (d) and (e). Furthermore, $\langle 00* \rangle$ and $\langle 0*1 \rangle$, fully cover $\langle 001 \rangle$ and $\langle 10* \rangle$, $\langle 1*1 \rangle$ fully covers $\langle 101 \rangle$. As such, the remaining two regulator states are covered with empty extension sets and the final solution uses



(a) Initial configuration (b) Configuration after one iteration. (c) Configuration after two iterations.

(d) Configuration after three iterations. (e) Configuration after four iterations. (f) Configuration after five iterations.

Figure 4: Regulator states of component a during computation of regulation cover set for value change $a_1 \to a_0$. Bold text, lines and shaded areas indicate (partial) regulator states which enable the value change ($\mathcal{A}_{\text{ENA}}$). The underlined regulator state is the state covered in the respective iteration. Dashes represent removed partial regulator states ($\mathcal{A}_{\text{RMV}}$) and double lines represent partial regulator states included in the regulation cover set ($\mathcal{A}_{a_1 \to a_0}$).

12 regulator value specifications as opposed to the 18 required by the explicit representation.

The fractional part of the weight function is crucial to distinguish between $\langle 001 \rangle$, $\langle 101 \rangle$ and $\langle 011 \rangle$, $\langle 111 \rangle$ after the second iteration (Figure 4 (c)). Covering $\langle 001 \rangle$ or $\langle 101 \rangle$ before $\langle 011 \rangle$ and $\langle 111 \rangle$ would include either $\langle **1 \rangle$ or $\langle *01 \rangle$, depending on the exact order, in the final regulation cover set. As both of them are redundant, this would lead to a suboptimal solution.

Algorithm 1 is quasilinear in the number of regulator states and quadratic in the number of regulators. Its main complexity comes from computing the extension sets $\mathcal{A}_{\text{EXT}}$. Whether a regulator state $\omega \in \Omega_v$ is covered by some $\mathcal{A}_c \cup \mathcal{A}_{\text{EXT}}$ can be checked in $\mathcal{O}(|n^-(v)|)$. Each ω requires at most $|n^-(v)|$ such

tests (but usually much less). As such, the extension set can be computed in $\mathcal{O}(|n^-(v)|^2)$ and thus, for all the regulator states: $\mathcal{O}(|\Omega_v| \cdot |n^-(v)|^2)$. Finally, the quasilinear complexity comes from the need to keep the regulator states in a priority queue giving us the final complexity of $\mathcal{O}(|\Omega_v| \cdot (\text{LOG}(|\Omega_v|) + |n^-(v)|^2))$.

Algorithm 1 does not require explicit enumeration of parametrisations when coupled with the parametrisation set semantics Ψ_A. The parametrisation set is only used to determine which regulator states enable the value change (queries to $\mathcal{A}_{\text{ENA}}$). This information is readily available using Ψ_A in the form of parameter values of the relevant bound.

5 Discussion

The goal-oriented model reduction procedure for parametrised models has been extended to parametric regulatory networks. The parametric reduction procedure is compatible with a large family of parametrisation set semantics functions, including the over-approximating semantics introduced for PRNs in [15], without the need to enumerate the parametrisations explicitly.

The reduction method can be applied alongside the model refinement procedure based on unfolding [15]. The parametric reduction can be applied on-the-fly within PRN unfolding in the same fashion the reduction procedure for parametrised networks is applied in Petri net unfoldings [6]. The application to PRN unfoldings suffers from the same challenge with cut-off events as the parametrised version with Petri net unfoldings. The challenge arises from the need to keep track of the transition set as the model evolves (transitions are pruned) by the reduction procedure along the unfolding process. Moreover, a similar challenge is already present in PRN unfoldings due to parametrisation sets [15]. Two different methods are used to tackle the issue. In [6], if more transitions are encountered during the unfolding, the respective branch is reiterated with the new transition set. In [15], a new branch is introduced into the unfolding for the new parametrisation set instead. Both of the methods are applicable for transitions (respectively, l^A and l^I) in PRN unfoldings with model reduction.

The parametric reduction is an independent procedure and can be applied in any other setting besides the mentioned coupling with model refinement. Moreover, should complexity be a concern, several possibilities to abstract the procedure exist. The regulation cover set allows for a different algorithm, or even to relax the definition itself. Or, the condition for a trace to be realisable can be dropped from the validity criterion for objectives to avoid having to check against parametrisation sets. Both of the suggested approximations are sound as adding new transitions has no effect on minimal traces.

Future work includes the refinement of the interplay between parametric model reduction and model refinement, further applications and extensions of the parametric model reduction itself and application of goal-oriented reduction to a wider variety of parametric models.

References

1. Ezio Bartocci and Pietro Lió. Computational modeling, formal analysis, and tools for systems biology. *PLOS Computational Biology*, 12(1):1–22, 2016. doi:10.1371/journal.pcbi.1004591.
2. G. Bernot, J.-P. Comet, and Z. Khalis. Gene regulatory networks with multiplexes. In *European Simulation and Modelling Conference Proceedings*, pages 423–432, 2008.
3. G. Bernot, J.-P. Comet, Z. Khalis, A. Richard, and O. Roux. A genetically modified hoare logic. *Theoretical Computer Science*, 2018. doi:10.1016/j.tcs.2018.02.003.
4. Gilles Bernot, Franck Cassez, Jean-Paul Comet, Franck Delaplace, Céline Müller, and Olivier Roux. Semantics of biological regulatory networks. *Electronic Notes in Theoretical Computer Science*, 180(3):3 – 14, 2007. doi:10.1016/j.entcs.2004.01.038.
5. Josep Carmona and Thomas Chatain. Anti-alignments in conformance checking – the dark side of process models. In Fabrice Kordon and Daniel Moldt, editors, *Proceedings of the 37th International Conference on Applications and Theory of Petri Nets (PETRI NETS'16)*, volume 9698 of *Lecture Notes in Computer Science*, pages 240–258, Torún, Poland, 2016. Springer. doi:10.1007/978-3-319-39086-4_15.
6. Thomas Chatain and Loïc Paulevé. Goal-Driven Unfolding of Petri Nets. In Roland Meyer and Uwe Nestmann, editors, *28th International Conference on Concurrency Theory (CONCUR 2017)*, volume 85 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2017.18.
7. Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1&2):117–136, 1995. doi:10.1016/0304-3975(94)00231-7.
8. David P. A. Cohen, Loredana Martignetti, Sylvie Robine, Emmanuel Barillot, Andrei Zinovyev, and Laurence Calzone. Mathematical modelling of molecular pathways enabling tumour cell invasion and migration. *PLoS Comput. Biol.*, 11(11):e1004571, 2015. doi:10.1371/journal.pcbi.1004571.
9. Samuel Collombet, Chris van Oevelen, Jose Luis Sardina Ortega, Wassim Abou-Jaoudé, Bruno Di Stefano, Morgane Thomas-Chollier, Thomas Graf, and Denis Thieffry. Logical modeling of lymphoid and myeloid cell specification and transdifferentiation. *Proc. Natl. Acad. Sci.*, 114(23):5792–5799, 2017. doi:10.1073/pnas.1610622114.
10. Fabien Corblin, Eric Fanchon, Laurent Trilling, Claudine Chaouiya, and Denis Thieffry. *Automatic Inference of Regulatory and Dynamical Properties from Incomplete Gene Interaction and Expression Data*, pages 25–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-28792-3_4.
11. Serge Haddad and Jean-François Pradat-Peyre. New efficient Petri nets reductions for parallel programs verification. *Parallel Processing Letters*, 16(1):101–116, March 2006. doi:10.1142/S0129626406002502.
12. Tomáš Helikar, Bryan Kowal, Sean McClenathan, Mitchell Bruckner, Thaine Rowley, Alex Madrahimov, Ben Wicks, Manish Shrestha, Kahani Limbu, and Jim A Rogers. The Cell Collective: toward an open and collaborative approach to systems biology. *BMC Syst. Biol.*, 6:96, 2012. doi:10.1186/1752-0509-6-96.

13. Zohra Khalis, Jean-Paul Comet, Adrien Richard, and Gilles Bernot. The SMBioNet method for discovering models of gene regulatory networks. *Genes, Genomes and Genomics*, 3(1):15–22, 2009. URL: `http://www.globalsciencebooks.info/Online/GSBOnline/OnlineGGG_3_SI1.html`.

14. Hannes Klarner, Adam Streck, David Šafránek, Juraj Kolčák, and Heike Siebert. Parameter identification and model ranking of thomas networks. In David Gilbert and Monika Heiner, editors, *Computational Methods in Systems Biology*, pages 207–226, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. `doi:10.1007/978-3-642-33636-2_13`.

15. Juraj Kolčák, David Šafránek, Stefan Haar, and Loïc Paulevé. Parameter Space Abstraction and Unfolding Semantics of Discrete Regulatory Networks. *Theoretical Computer Science*, 2018. `doi:10.1016/j.tcs.2018.03.009`.

16. Maciej Koutny, Jörg Desel, and Jetty Kleijn, editors. *Transactions on Petri Nets and Other Models of Concurrency XI*, volume 9930 of *Lecture Notes in Computer Science*. Springer, 2016. `doi:10.1007/978-3-662-53401-4`.

17. Andrey Mokhov, Josep Carmona, and Jonathan Beaumont. *Mining Conditional Partial Order Graphs from Event Logs*, pages 114–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. `doi:10.1007/978-3-662-53401-4_6`.

18. Aurélien Naldi, Céline Hernandez, Nicolas Levy, Gautier Stoll, Pedro T. Monteiro, Claudine Chaouiya, Tomáš Helikar, Andrei Zinovyev, Laurence Calzone, Sarah Cohen-Boulakia, Denis Thieffry, and Loïc Paulevé. The CoLoMoTo Interactive Notebook: Accessible and Reproducible Computational Analyses for Qualitative Biological Networks. *Frontiers in Physiology*, 9:680, 2018. `doi:10.3389/fphys.2018.00680`.

19. Max Ostrowski, Loïc Paulevé, Torsten Schaub, Anne Siegel, and Carito Guziolowski. Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. *Biosystems*, 149:139 – 153, 2016. `doi:10.1016/j.biosystems.2016.07.009`.

20. Loïc Paulevé. Reduction of Qualitative Models of Biological Networks for Transient Dynamics Analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2017. `doi:10.1109/TCBB.2017.2749225`.

21. Hernán Ponce de León, César Rodríguez, Josep Carmona, Keijo Heljanko, and Stefan Haar. Unfolding-based process discovery. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA'15)*, volume 9364 of *Lecture Notes in Computer Science*, Shanghai, China, 2015. Springer. `doi:10.1007/978-3-319-24953-7_4`.

22. Carolyn Talcott and David L. Dill. Multiple representations of biological processes. In *Transactions on Computational Systems Biology VI*, pages 221–245. Springer Science Business Media, 2006. `doi:10.1007/11880646_10`.

23. Denis Thieffry and René Thomas. Dynamical behaviour of biological regulatory networks—ii. immunity control in bacteriophage lambda. *Bulletin of Mathematical Biology*, 57:277–297, 1995. `doi:10.1007/BF02460619`.

24. René Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, 42(3):563 – 585, 1973. `doi:10.1016/0022-5193(73)90247-6`.

A Proof of Theorem 1

Here we conduct the proof of Theorem 1 stating that by conducting reduction of a DPRN $\mathcal{G}$ w.r.t. an initial state x and goal $g_\top$, all transitions of all minimal traces are preserved by the means of compatible partial transitions in $\tau_x(\mathcal{B})$.

We first show that an existence of an objective O that covers a transition $t = (v_k \to v_q, \omega)$ of a realisable trace, formally $O = v_{k-a\cdot s(t)} \rightsquigarrow v_{q+b\cdot s(t)}$ for some $a, b \in \mathbb{N}_0$, in $\mathcal{B}$ is enough to guarantee existence of a compatible partial transition and thus preservation of the transition.

Lemma 1. *Let a realisable trace π of an DPRN $\mathcal{G}$ reach goal $g_\top$ from initial state x and let $\mathcal{B}$ be the objective set constructed by Definition 12 for the given goal and initial state. Then for any $\pi_i = (v_k \to v_q, \omega)$ covered by an objective $O \in \mathcal{B}$ there exists a partial transition $\delta = (v_k \to v_q, \aleph) \in \tau_x(\mathcal{B})$ such that $\omega \in \aleph$.*

Proof. Let $\aleph \in \mathcal{A}_{v_k \to v_q}$ be arbitrary such that $\omega \in \aleph$. We know at least one such $\aleph$ exists by definition of regulation cover set (Definition 10).

Then, by Definition 11, the corresponding partial transition $\delta = (v_k \to v_q, \aleph) \in \tau(O) \subseteq \tau(\mathcal{B})$. Finally, since π itself is a witness of the validity of objectives for all regulators required by δ, $\delta \in \tau_x(O) \subseteq \tau_x(\mathcal{B})$.

We now show that if a transition of a realisable trace π is not covered by an objective in $\mathcal{B}$, the trace is not minimal.

Let thus π_i with $V(\pi_i) = v$ be such a transition, and let $t = \pi_h$ be the last transition covered by an objective in $\mathcal{B}$ such that $h < i$ and $V(t) = v$, if it exists. Finally, let $t' = \pi_j$ be the first transition such that $i < j$, $V(t') = v$ and ${}^\bullet\pi^{j:}{}_v = \pi^{:h}{}^\bullet{}_v$ (respectively, ${}^\bullet\pi^{j:}{}_v = x_v$ if t does not exist), if it exists.

We now construct a trace π' by removing all transitions in $\pi^{h+1:j-1}$ which change the value of v from π, where $h = 0$ if t does not exist and $j = |\pi|$ if t' does not exist. Since the removed transitions form a loop on the value of v, respectively, have no causal successors modifying the value of v if t' does not exist, the evolution of v along π' remains valid.

Moreover, if any transition π_κ covered by $O \in \mathcal{B}$ such that $h < \kappa < j$ and $V(\pi_\kappa) = v$ exists, then there exists another transition $\pi_\iota \in \pi^{:h}$ with the same value change as π_κ.

Let thus $v_\alpha \rightsquigarrow v_q \in \mathcal{B}$ cover at least one transition modifying the value of v in $\pi^{h+1:j-1}$. For such an objective to be included there must exist a covered transition requiring value q of component v somewhere along the trace and thus, by point (2) of Definition 12, $x_v \rightsquigarrow q \in \mathcal{B}$. The transition t therefore has to exist, meaning an objective $\beta \rightsquigarrow \pi^{:h}{}^\bullet{}_v \in \mathcal{B}$ also exists. By point (3) of Definition 12, we get $\pi^{:h}{}^\bullet{}_v \rightsquigarrow q, q \rightsquigarrow \pi^{:h}{}^\bullet{}_v \in \mathcal{B}$. Then, since π_i is not covered, we have $s(\beta \rightsquigarrow \pi^{:h}{}^\bullet{}_v) = s(v_\alpha \rightsquigarrow v_q)$ and moreover α is an intermediate value in the objective $\beta \rightsquigarrow \pi^{:h}{}^\bullet{}_v$. Thus, the same value change as the covered transition in $\pi^{h+1:j-1}$ had to occur in $\pi^{:h}$ in order to reach $\pi^{:h}{}^\bullet{}_v$.

Since $x_g \rightsquigarrow \top \in \mathcal{B}$ by point (1) of Definition 12, π' surely reaches the goal from the initial state. Furthermore, thanks to properties of the parametrisation

set semantics and since π is realisable, also $\Psi(\widetilde{\pi'}) \neq \emptyset$. If π' is valid (regulator state of each transition matches the source state), π is not minimal and we are done. Let us therefore assume there exists a transition π_k with $V(\pi_k) \neq v$ which causally depends on one of the removed transitions. We first show such π_k cannot be covered by an objective in $\mathcal{B}$ by contradiction.

Let thus $O \in \mathcal{B}$ cover π_k and let q be the value of v π_k depends on. Surely $q \neq {\pi^{:h}}^{\bullet}{}_v$, as otherwise the removal of $\pi^{h+1:j-1}$ has no effect on π_k. As such, since $x_v \rightsquigarrow q \in \mathcal{B}$, t must exist. Then also objective $\alpha \rightsquigarrow {\pi^{:h}}^{\bullet}{}_v \in \mathcal{B}$ and by point (3) of Definition 12 ${\pi^{:h}}^{\bullet}{}_v \rightsquigarrow q \in \mathcal{B}$. Therefore the first transition reaching the value q of v in $\pi^{h:k}$ is covered, which is a contradiction with π_h being the last covered transition.

We can thus repeat the reasoning for the uncovered π_k and remove the respective loop or unused tail of evolution of component $V(\pi_k)$. Since π is finite, all invalid transitions are ultimately removed while retaining realisability and reachability of the goal. π is thus not minimal.