Parameter Synthesis Algorithms for Parametric Interval Markov Chains

Laure Petrucci, Jaco van De Pol

Parameter Synthesis Algorithms for Parametric Interval Markov Chains*

Laure Petrucci[1][0000-0003-3154-5268] and Jaco van de Pol[2][0000-0003-4305-0625]

[1] LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France
[2] Formal Methods and Tools, University of Twente, Enschede, The Netherlands

Abstract. This paper considers the consistency problem for Parametric Interval Markov Chains. In particular, we introduce a co-inductive definition of consistency, which improves and simplifies previous inductive definitions considerably. The equivalence of the inductive and co-inductive definitions has been formally proved in the interactive theorem prover PVS.
These definitions lead to forward and backward algorithms, respectively, for synthesizing an expression for all parameters for which a given PIMC is consistent. We give new complexity results when tackling the consistency problem for IMCs (i.e. without parameters). We provide a sharper upper bound, based on the longest simple path in the IMC. The algorithms are also optimized, using different techniques (dynamic programming cache, polyhedra representation, etc.). They are evaluated on a prototype implementation. For parameter synthesis, we use Constraint Logic Programming and the PARMA library for convex polyhedra.[1]

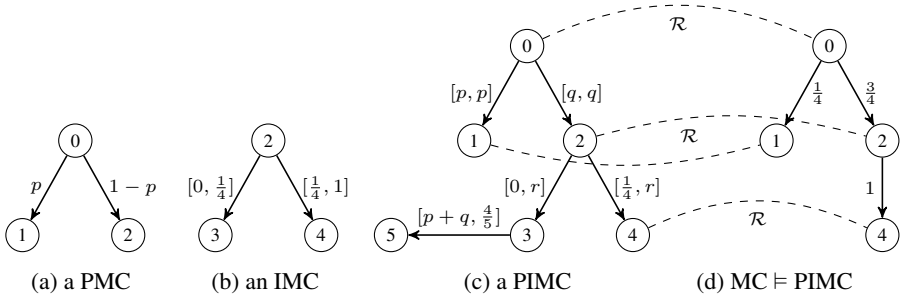
1 Introduction

Markov Chains (MC) are widely used to model stochastic systems, like randomized protocols, failure and risk analysis, and phenomena in molecular biology. Here we focus on discrete time MCs, where transitions between states are governed by a state probability distribution, denoted by $\mu : S \times S \to [0, 1]$. Practical applications are often hindered by the fact that the probabilities $\mu(s, t)$, to go from state s to t, are unknown. Several solutions have been proposed, for instance Parametric Markov Chains (PMC) [7] and Interval Markov Chains (IMC) [14], in which unknown probabilities are replaced by parameters or intervals, respectively, see Figs. 1a and 1b. Following [9], we study their common generalization, Parametric Interval Markov Chains (PIMC, Fig. 1c), which allow intervals with parametric bounds. PIMCs are more expressive than IMCs and PMCs [2]. PIMCs allow to study the boundaries of admissible probability intervals, which is useful in the design exploration phase. This leads to the study of parameter synthesis for PIMCs, started in [12].

IMCs can be viewed as specifications of MCs. An IMC is consistent if there exists an MC that implements it. The main requirements on the implementation are: (1) all behaviour of the MC can be simulated by the IMC, preserving probabilistic information;

* This research was conducted with the support of PHC Van Gogh project PAMPAS.

[1] The complete text of the proofs, their PVS formalisation, Prolog programs, and experimental data can be found at http://fmt.cs.utwente.nl/~vdpol/PIMC2018.zip.



(a) a PMC (b) an IMC (c) a PIMC (d) MC $\models$ PIMC

Fig. 1: Examples of a PMC (a), an IMC (b), and of their generalization PIMC (c). The MC (d) implements the PIMC (c), as shown by the dashed edges, and formalized in Def. 7. We drop self-loops with probability 1 (or $[1, 1]$) in all terminal nodes.

and (2) the outgoing transition probabilities for each state sum up to 1. The consistency synthesis problem for PIMCs is to compute all parameter values leading to a consistent IMC. E.g., PIMC in Fig. 1c[2] is consistent when $q = 0, p = 1$, or when $p+q = 1, r = 1$. The witness MC of Fig. 1d corresponds to $p = \frac{1}{4}, q = \frac{3}{4}, r = 1$.

Contribution. This paper studies the consistency synthesis problem for PIMCs. We improve the theory in [12], which gave an inductive definition of n-consistency. Basically, a state is n-consistent if it has a locally consistent subset of successors, which are in turn all $(n-1)$-consistent. Here local consistency checks that there is a solution within the specified bounds that sums up to 1. That paper provided an expression for n-consistency. There are two drawbacks from an algorithmic perspective: all possible subsets of successors must be enumerated, and a sufficiently large upper bound for n must be provided. It has been shown that taking the number of states for n is sufficient.

We address both problems. First we simplify the inductive definition and show that for IMCs, the enumeration over the subset of successors is not necessary. Instead, we can restrict to a single candidate: the set of *all* $(n-1)$-consistent successors. Second, we provide a smaller upper bound for n. We show that it is sufficient to take the length of the longest simple path in the IMC. However, the length of the longest simple path cannot be efficiently computed (this would solve the Hamiltonian path problem, which is NP-complete).

Our main contribution is to provide an alternative, co-inductive definition of consistency, dropping the need to reason about n altogether. Instead, we define consistency as the largest set, such that a state is consistent if the set of its consistent successors is locally consistent. We have formally proved the equivalence between all these definitions in the interactive theorem prover PVS [17]. The complete PVS proof development is available online.

Based on the simplified inductive definition, we provide a polynomial time *forward* algorithm to check that an IMC is consistent. Based on the new co-inductive defini-

[2] In the following, for the sake of readability, we do not consider linear combinations of parameters as bounds of the intervals. However, allowing them would not change the results.

tion, we provide a polynomial *backward* algorithm. Again, the number of iterations is bounded by the length of the longest simple path, without having to compute it.

Finally, we provide algorithms to compute an expression for *all* parameters for which a PIMC is consistent. Unfortunately, to obtain an expression we must fall back on subset enumeration. The forward algorithm can be implemented as a Constraint Logic Program, so Prolog + CLP(Q) can be used directly to compute a list of all solutions, basically as a disjunction of conjunctions of linear inequations over the parameters. We introduce two optimizations: caching intermediate results and suppressing subsumed solutions. The backward algorithm for IMCs can be viewed as computing the maximal solution of a Boolean Equation System. Generalizing this to PIMCs, we now compute the maximal solution of an equation system over disjunctions of conjunctions of constraints. Such equation systems can be solved by standard iteration, representing the intermediate solutions as powerdomains over convex closed polyhedra. We implemented this using the Parma Polyhedra Library [1].

Related Work. One of the first results on synthesis for PMCs [7] computes the probability of path formulas in PCTL as an expression over the parameters. Since then, the efficiency and numeric stability has been improved considerably [18,8]. On such models, the realizability (or well-defined) property is considered [16,13] which mainly differs from the consistency we address in that they consider consistency of all states, while we have the option to avoid some states (and their successors) by assigning null-probability to some edges. Model checking for IMCs is studied, for instance in [5,6]. For *continuous-time* Markov Chains, precise parameter synthesis is studied as well [4]. However, PIMCs are more expressive (concise) than PMCs and IMCs, as shown in [2]. As far as we know, besides reachability, there are no model checking results for PIMCs.

Other specification formalisms include Constraint Markov Chains [11], with arbitrary constraints on transitions, and Abstract Probabilistic Automata [10], which add non-deterministic transitions. We believe that our work can be extended in a straightforward manner to CMCs with linear constraints. For APA the situation is probably quite different. Another branch of research has investigated the synthesis of parameters for timed systems, but that is out of the scope of this paper.

PIMCs, and the related consistency problem, have been introduced in [9]. Our backward algorithm for IMCs is somewhat similar in spirit to the pruning operator of [9]. We provide a sharper upper bound on the number of required iterations. That paper addresses the *existence* of a consistent parameter valuation for a restricted subclass of PIMCs, where parameters occur only locally. The parameter *synthesis* problem for the full class of PIMCs was considered in [12]. We improved on their theory, as explained before. Our experiments (Section 6) show that our algorithms and optimizations are more efficient than the approach in [12].

Very recently, [2] also introduced a CLP approach for checking the *existence* of a consistent parameter valuation. Their contribution is a CLP program of linear size in the PIMC. Their CLP is quite different from ours: basically, they introduce a Boolean variable for each state and a real variable for each transition probability of the Markov Chain that implements the PIMC. So solving the CLP corresponds to searching for a satisfying implementation.

2 Parametric Interval Markov Chains

As Parametric Interval Markov Chains allow for describing a family of Markov Chains, we first define these.

Definition 1 (Markov Chain). *A* Markov Chain *(MC) is a tuple* (S, s_0, μ, A, V)*, where:*

- S *is a set of* states *and* $s_0 \in S$ *is the* initial state*;*
- $\mu : S \times S \to [0, 1]$ *is the* transition probability distribution *s.t.:* $\forall s \in S : \sum_{s' \in S} \mu(s, s') = 1$*;*
- A *is a set of* labels *and* $V : S \to A$ *is the* labelling function*.*

Notation 2. Let P be a set of *parameters*, i.e. variable names. We denote by $\mathsf{Int}[0, 1](P)$ the set of pairs $[a, b]$ with $a, b \in [0, 1] \cup P$. Given $x \in \mathsf{Int}[0, 1](P)$, we denote by x_ℓ and x_u its left and right components. If x is an interval, this corresponds to its lower and upper bounds. The same notation is used for functions which result in an interval.

Example 3. $[0.3, 0.7], [0, 1], [0.5, 0.5], [p, 0.8], [0.99, q], [p, q]$ are all in $\mathsf{Int}[0, 1](\{p, q\})$.

Definition 4 ((Parametric) Interval Markov Chain). *A* Parametric Interval Markov Chain *(PIMC) is a tuple* $(P, S, s_0, \varphi, A, V)$ *where:*

- P *is a set of* parameters*;*
- S *is a set of* states *and* $s_0 \in S$ *is the* initial state*;*
- $\varphi : S \times S \to \mathsf{Int}[0, 1](P)$ *is the* parametric transition probability constraint*;*
- A *is a set of* labels *and* $V : S \to A$ *is the* labelling function*.*

An Interval Markov Chain *(IMC) is a PIMC with* $P = \varnothing$ *(we drop* P *everywhere).*

Note that Def. 4 and 1 are very similar, but for PIMCs and IMCs the well-formedness of the intervals and of the probability distribution will be part of the consistency property to be checked (see Def. 11).

When ambiguity is possible, we will use a subscript to distinguish the models, e.g. $S_\mathcal{M}$, $S_\mathcal{I}$ and $S_\mathcal{P}$ will denote the set of states of respectively a MC, an IMC and a PIMC.

If all intervals are point intervals of the form $[p, p]$, this PIMC is actually a Parametric Markov Chain [7].

Example 5. Fig. 2a shows our running example of a PIMC with two parameters p and q (taken from [12]). Fig. 2b shows a particular IMC.

Definition 6 (Support). *The* support *of a probability distribution* μ *at a state* $s \in S_\mathcal{M}$ *is the set:* $sup(\mu, s) := \{s' \in S_\mathcal{M} \mid \mu(s, s') > 0\}$.

Similarly, for a parametric transition probability constraint φ *at a state* $s \in S_\mathcal{P}$ *the support is the set:* $sup(\varphi, s) := \{s' \in S_\mathcal{P} \mid \varphi_u(s, s') > 0\}$.

Assumption: From now on, we will assume that $\mathcal{I}$ is finitely branching, i.e. for all $s \in S$, $sup(\varphi, s)$ is a finite set. For the algorithms in Section 4 we will even assume that S is finite.

PIMCs and IMCs can be viewed as specifications of MCs.

Definition 7 (A MC implements an IMC). *Let $\mathcal{M} = (S_\mathcal{M}, s_{\mathcal{M}0}, \mu, A, V_\mathcal{M})$ be a MC and $\mathcal{I} = (S_\mathcal{I}, s_{\mathcal{I}0}, \varphi, A, V_\mathcal{I})$ an IMC.* $\mathcal{M}$ implements $\mathcal{I}$ *($\mathcal{M} \vDash \mathcal{I}$) if there exists a* simulation relation $\mathcal{R} \subseteq S_\mathcal{M} \times S_\mathcal{I}$, *s.t.* $\forall s_\mathcal{M} \in S_\mathcal{M}$ *and* $s_\mathcal{I} \in S_\mathcal{I}$, *if* $s_\mathcal{M} \mathcal{R} s_\mathcal{I}$, *then:*

1. $V_\mathcal{M}(s_\mathcal{M}) = V_\mathcal{I}(s_\mathcal{I})$
   (the source and target states have the same label)
2. *There exists a probabilistic correspondence* $\delta : S_\mathcal{M} \times S_\mathcal{I} \to [0,1]$, *s.t.:*
   (a) $\forall s'_\mathcal{I} \in S_\mathcal{I} : \sum_{s'_\mathcal{M} \in S_\mathcal{M}} \mu(s_\mathcal{M}, s'_\mathcal{M}) \cdot \delta(s'_\mathcal{M}, s'_\mathcal{I}) \in \varphi(s_\mathcal{I}, s'_\mathcal{I})$
   (the total contribution of implementing transitions satisfies the specification)
   (b) $\forall s'_\mathcal{M} \in S_\mathcal{M} : \mu(s_\mathcal{M}, s'_\mathcal{M}) > 0 \Rightarrow \sum_{s'_\mathcal{I} \in S_\mathcal{I}} \delta(s'_\mathcal{M}, s'_\mathcal{I}) = 1$
   (the implementing transitions yield a probability distribution)
   (c) $\forall s'_\mathcal{M} \in S_\mathcal{M}, s'_\mathcal{I} \in S_\mathcal{I} : \delta(s'_\mathcal{M}, s'_\mathcal{I}) > 0 \Rightarrow s'_\mathcal{M} \mathcal{R} s'_\mathcal{I}$
   (corresponding successors are in the simulation relation)
   (d) $\forall s'_\mathcal{M} \in S_\mathcal{M}, s'_\mathcal{I} \in S_\mathcal{I} : \delta(s'_\mathcal{M}, s'_\mathcal{I}) > 0 \Rightarrow \mu(s_\mathcal{M}, s'_\mathcal{M}) > 0 \wedge \varphi_u(s_\mathcal{I}, s'_\mathcal{I}) > 0$
   (δ is only defined on the support of μ and φ)

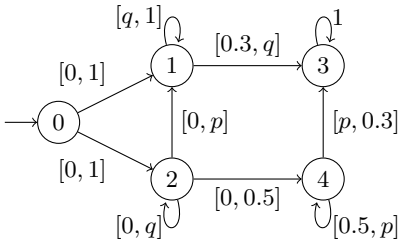
Definition 8 (Consistency).
An IMC $\mathcal{I}$ is consistent *if for some MC $\mathcal{M}$, we have $\mathcal{M} \vDash \mathcal{I}$.*
A PIMC is consistent if there exist parameter values such that the corresponding IMC is consistent.

Intuitively, this definition states that the implementation is an MC, whose behaviour is allowed by the specification IMC, i.e., the IMC can simulate the MC. Clause (2d) was not present in the original definition [9], but it is convenient in proofs. We first show that limiting δ to the support of μ and φ does not alter the implementation relation.

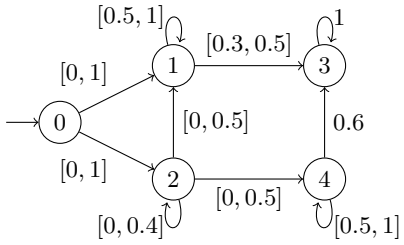
Lemma 9. *Def. 7 with clauses (2a)–(2c) is equivalent to Def. 7 with (2a)–(2d).*

Proof. Assume, there exist $\mathcal{R}$, $s_\mathcal{M} \mathcal{R} s_\mathcal{I}$ and δ that satisfy conditions (2a)– (2c) of Def. 7. Define:

$$\delta'(s'_\mathcal{M}, s'_\mathcal{I}) := \begin{cases} \delta(s'_\mathcal{M}, s'_\mathcal{I}) & \text{if } \mu(s_\mathcal{M}, s'_\mathcal{M}) > 0 \text{ and } \varphi_u(s_\mathcal{I}, s'_\mathcal{I}) > 0; \\ 0 & \text{otherwise.} \end{cases}$$



(a) A Parametric Interval Markov Chain



(b) An Interval Markov Chain

Fig. 2: Running Examples

Note that if $\mu(s_{\mathcal{M}}, s'_{\mathcal{M}}) > 0$ and $\varphi_u(s_{\mathcal{I}}, s'_{\mathcal{I}}) = 0$ then $\delta(s'_{\mathcal{M}}, s'_{\mathcal{I}}) = 0$ by (2a). Now properties (2a)–(2d) can be easily checked for δ'. □

Example 10. For Fig. 3, checking condition (2a) for t_1 boils down to $0.1{\cdot}0.3{+}0.6{\cdot}0.2 = 0.15 \in [0.1, 0.2]$. Checking condition (2b) for s_1 requires $0.7 + 0.3 = 1$.

3 Consistency of Interval Markov Chains

In this section we study consistency of Interval Markov Chains; we will return to Parametric IMCs in Section 5. Intuitively, an IMC is consistent if it can be implemented by at least one MC. From now on, we will drop the labelling V, since it plays no role in the discussion on consistency, and thus consider an arbitrary IMC $\mathcal{I} = (S, s_0, \varphi)$.

3.1 Local Consistency

Local consistency of a state $s \in S$ is defined with respect to a set $X \subseteq S$ of its successors, and its probability constraint $\varphi(s, \cdot)$. It ensures the existence of some probability distribution satisfying the interval constraints on transitions from s to X: the collective upper bound should be greater than or equal to 1 (condition $up(\varphi, s, X)$), the collective lower bound less than or equal to 1 ($low(\varphi, s, X)$). Moreover, each lower bound should be smaller than the corresponding upper bound, and the states outside X should be avoidable, in the sense that they admit probability 0 ($local(\varphi, s, X)$).

Definition 11 (Local consistency). *The* local consistency constraints *for a state $s \in S$ and a set $X \subseteq S$ is $LC(\varphi, s, X)$ s.t.:*

$$
\begin{aligned}
LC(\varphi, s, X) &:= up(\varphi, s, X) \wedge low(\varphi, s, X) \wedge local(\varphi, s, X), \ \textit{where} \\
up(\varphi, s, X) &:= \textstyle\sum_{s' \in X} \varphi_u(s, s') \geq 1 \\
low(\varphi, s, X) &:= \textstyle\sum_{s' \in X} \varphi_\ell(s, s') \leq 1 \\
local(\varphi, s, X) &:= (\forall s' \in X : \varphi_\ell(s, s') \leq \varphi_u(s, s')) \wedge (\forall s' \notin X : \varphi_\ell(s, s') = 0)
\end{aligned}
$$

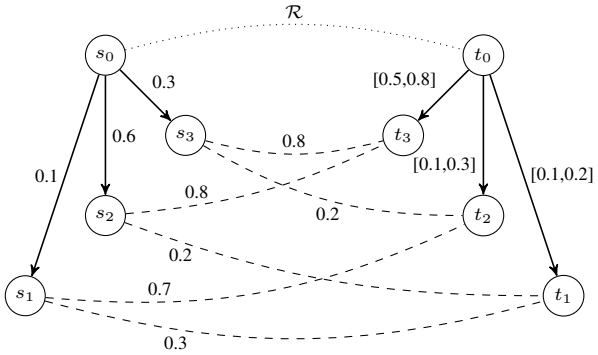


Fig. 3: The Markov Chain (left) implements the Interval Markov Chain (right)

We obtain the following facts, which can be directly checked from the definitions. Note that from Lemma 12(1) and (2) it follows that we may always restrict attention to the support of (φ, s): $LC(\varphi, s, X) \equiv LC(\varphi, s, X \cap sup(\varphi, s))$.

Lemma 12. *For $X, Y \subseteq S$:*

1. *If $X \subseteq Y$ and $LC(\varphi, s, X)$ then $LC(\varphi, s, Y)$.*
2. *If $LC(\varphi, s, X)$ then also $LC(\varphi, s, X \cap sup(\varphi, s))$.*

Proof. 1. Assume $X \subseteq Y$ and $LC(\varphi, s, X)$, hence $up(\varphi, s, X)$, $low(\varphi, s, X)$ and $local(\varphi, s, X)$.
From $up(\varphi, s, X)$, we have $\sum_{s' \in X} \varphi_u(s, s') \geq 1$, so we get $up(\varphi, s, Y)$:

$$\sum_{s' \in Y} \varphi_u(s, s') = (\sum_{s' \in X} \varphi_u(s, s') + \sum_{s' \in Y \setminus X} \varphi_u(s, s')) \geq (1 + \sum_{s' \in Y \setminus X} \varphi_u(s, s')) \geq 1$$

From $local(\varphi, s, X)$, we have $\forall s' \in Y \setminus X : \varphi_\ell(s, s') = 0$, and from $low(\varphi, s, X)$, we have $\sum_{s' \in X} \varphi_\ell(s, s') \leq 1$, so we get $low(\varphi, s, Y)$:

$$\sum_{s' \in Y} \varphi_\ell(s, s') = (\sum_{s' \in X} \varphi_\ell(s, s') + \sum_{s' \in Y \setminus X} \varphi_\ell(s, s')) = \sum_{s' \in X} \varphi_\ell(s, s') + 0 \leq 1$$

Finally, from $local(\varphi, s, X)$, it holds that for $s' \in Y$, if $s' \in X$ then $\varphi_\ell(s, s') \leq \varphi_u(s, s')$, else $\varphi_\ell(s, s') = 0$, which also implies $\varphi_\ell(s, s') \leq \varphi_u(s, s')$. If $s' \notin Y$ then $s' \notin X$, so $\varphi_\ell(s, s') = 0$. So we get $local(\varphi, s, Y)$. This proves that $LC(\varphi, s, Y)$.

2. Assume $LC(\varphi, s, X)$, hence $up(\varphi, s, X)$, $low(\varphi, s, X)$ and $local(\varphi, s, X)$. Note that if $s' \in X \setminus sup(\varphi, s)$, by definition of sup, we obtain $\varphi_u(s, s') = 0$ and by $local(\varphi, s, X)$, we obtain $\varphi_\ell(s, s') = 0$.

$$\begin{aligned} \textstyle\sum_{s' \in X \cap sup(\varphi, s)} \varphi_u(s, s') &= \textstyle\sum_{s' \in X} \varphi_u(s, s') - \sum_{t \in X \setminus sup(\varphi, s)} \varphi_u(s, s') \\ &= \textstyle\sum_{s' \in X} \varphi_u(s, s') - 0 \geq 1 \end{aligned}$$

$$\begin{aligned} \textstyle\sum_{s' \in X \cap sup(\varphi, s)} \varphi_\ell(s, s') &= \textstyle\sum_{s' \in X} \varphi_\ell(s, s') - \sum_{s' \in X \setminus sup(\varphi, s)} \varphi_\ell(s, s') \\ &= \textstyle\sum_{s' \in X} \varphi_\ell(s, s') - 0 \leq 1 \end{aligned}$$

Finally, if $s' \in X \cap sup(\varphi, s)$ then $s' \in X$, so $\varphi_\ell(s, s') \leq \varphi_u(s, s')$.
Otherwise, $s' \notin X$ or $s' \in X \setminus sup(\varphi, s)$, but in both cases $\varphi_\ell(s, s') = 0$. □

3.2 Co-inductive Definition of Global Consistency

Global consistency of (P)IMCs can be defined in several ways, e.g. co-inductively and inductively. Here, we introduce a new co-inductive definition of global consistency, as a greatest fixed point (*gfp*). We first introduce an abbreviation for the set of locally consistent states w.r.t. a set X:

Notation 13. $LC_\varphi(X) := \{s \mid LC(\varphi, s, X)\}$

Next, we define $\mathcal{C}ons$ as the greatest fixed point of LC_φ. Intuitively, from consistent states one can keep taking locally consistent steps to other consistent states.

Definition 14 (Global consistency, co-inductive). $\mathcal{Cons} := gfp(LC_\varphi)$.

Lemma 15. *From the definition of greatest fixed point, $\mathcal{Cons}$ is the largest set C s.t. $C \subseteq LC_\varphi(C)$:*

1. $s \in \mathcal{Cons} \equiv s \in LC_\varphi(\mathcal{Cons}) \equiv s \in LC_\varphi(\mathcal{Cons} \cap sup(\varphi, s))$
2. *If $C \subseteq LC_\varphi(C)$ then $C \subseteq \mathcal{Cons}$.*

Proof. (1) holds because $\mathcal{Cons}$ is a fixed point; the second equation uses Lemma 12(2); (2) holds because $\mathcal{Cons}$ is the *greatest* fixed point. (Tarski) □

We motivate the definition of consistency by the following two theorems:

Theorem 16. *Let $\mathcal{M} = (S_\mathcal{M}, s_{\mathcal{M}0}, \mu)$ be a MC, $\mathcal{I} = (S_\mathcal{I}, s_{\mathcal{I}0}, \varphi)$ an IMC, and assume $\mathcal{M} \models \mathcal{I}$. Then $s_{\mathcal{I}0} \in \mathcal{Cons}$.*

Proof. Since $\mathcal{M} \models \mathcal{I}$, there is a simulation relation $\mathcal{R}$, with $s_{\mathcal{M}0}\mathcal{R}s_{\mathcal{I}0}$, and for all $s_\mathcal{M} \in S_\mathcal{M}, s_\mathcal{I} \in S_\mathcal{I}$, if $s_\mathcal{M}\mathcal{R}s_\mathcal{I}$, there is a correspondence δ, satisfying properties (2a)–(2d) of Def. 7. We will prove that $\{s_\mathcal{I} \mid \exists s_\mathcal{M} : s_\mathcal{M}\mathcal{R}s_\mathcal{I}\} \subseteq \mathcal{Cons}$. Since $s_{\mathcal{M}0}\mathcal{R}s_{\mathcal{I}0}$, it will follow that $s_{\mathcal{I}0} \in \mathcal{Cons}$.
We proceed using the *gfp*-property (Lemma 15(2)), so it is sufficient to prove:
$\{s_\mathcal{I} \in S_\mathcal{I} \mid \exists s_\mathcal{M} \in S_\mathcal{M} : s_\mathcal{M}\mathcal{R}s_\mathcal{I}\} \subseteq LC_\varphi(\{s_\mathcal{I} \in S_\mathcal{I} \mid \exists s_\mathcal{M} \in S_\mathcal{M} : s_\mathcal{M}\mathcal{R}s_\mathcal{I}\})$.
Let $s_\mathcal{I} \in S_\mathcal{I}$ be given with $s_\mathcal{M} \in S_\mathcal{M}$ s.t. $s_\mathcal{M}\mathcal{R}s_\mathcal{I}$. Define $X := \{s'_\mathcal{I} \in S_\mathcal{I} \mid \exists s'_\mathcal{M} \in S_\mathcal{M} : \delta(s'_\mathcal{M}, s'_\mathcal{I}) > 0\}$. Clearly, if $s'_\mathcal{I} \in X$, then for some $s'_\mathcal{M} \in S_\mathcal{M}$, $\delta(s'_\mathcal{M}, s'_\mathcal{I}) > 0$ and by the correspondence property of Def. 7(2c), $s'_\mathcal{M}\mathcal{R}s'_\mathcal{I}$. So $X \subseteq \{s_\mathcal{I} \in S_\mathcal{I} \mid \exists s_\mathcal{M} \in S_\mathcal{M} : s_\mathcal{M}\mathcal{R}s_\mathcal{I}\}$. Thus, by monotonicity, Lemma 12(1), it is sufficient to show that $s_\mathcal{I} \in LC_\varphi(X)$.
To check that $s_\mathcal{I} \in LC_\varphi(X)$, we first check that the corresponding transitions yield a probability distribution:

$$
\begin{aligned}
&\textstyle\sum_{s'_\mathcal{I} \in X} \sum_{s'_\mathcal{M} \in S_\mathcal{M}} \mu(s_\mathcal{M}, s'_\mathcal{M}) \cdot \delta(s'_\mathcal{M}, s'_\mathcal{I}) \\
&= \textstyle\sum_{s'_\mathcal{I} \in S_\mathcal{I}} \sum_{s'_\mathcal{M} \in S_\mathcal{M}} \mu(s_\mathcal{M}, s'_\mathcal{M}) \cdot \delta(s'_\mathcal{M}, s'_\mathcal{I}) && \text{(if } s'_\mathcal{I} \notin X, \delta(s'_\mathcal{M}, s'_\mathcal{I}) = 0 \text{ by def. of } X) \\
&= \textstyle\sum_{s'_\mathcal{M} \in S_\mathcal{M}} \mu(s_\mathcal{M}, s'_\mathcal{M}) \cdot (\sum_{s'_\mathcal{I} \in S_\mathcal{I}} \delta(s'_\mathcal{M}, s'_\mathcal{I})) && \text{(by } \textstyle\sum\text{-manipulation)} \\
&= \textstyle\sum_{s'_\mathcal{M} \in S_\mathcal{M}} \mu(s_\mathcal{M}, s'_\mathcal{M}) \cdot 1 && (\delta \text{ is a prob. distrib. by Def. 7(2b))} \\
&= 1 && (\mu \text{ is a prob. dist. by Def. 1 of MC})
\end{aligned}
$$

By Def. 7(2a), $\forall s'_\mathcal{I} \in X, \varphi_\ell(s_\mathcal{I}, s'_\mathcal{I}) \leq \sum_{s'_\mathcal{M} \in S_\mathcal{M}} \mu(s_\mathcal{M}, s'_\mathcal{M}) \cdot \delta(s'_\mathcal{M}, s'_\mathcal{I}) \leq \varphi_u(s_\mathcal{I}, s'_\mathcal{I})$.
By the computation above, $\sum_{s'_\mathcal{I} \in X} \varphi_\ell(s_\mathcal{I}, s'_\mathcal{I}) \leq 1$ and $\sum_{s'_\mathcal{I} \in X} \varphi_u(s_\mathcal{I}, s'_\mathcal{I}) \geq 1$, proving $low(\varphi, s_\mathcal{I}, X)$ and $up(\varphi, s_\mathcal{I}, X)$, respectively. For $s'_\mathcal{I} \in X$ we already established $\varphi_\ell(s_\mathcal{I}, s'_\mathcal{I}) \leq \varphi_u(s_\mathcal{I}, s'_\mathcal{I})$. For $s'_\mathcal{I} \notin X$, by definition of X: $\forall s'_\mathcal{M} : \delta(s'_\mathcal{M}, s'_\mathcal{I}) = 0$. Thus, $\varphi_\ell(s_\mathcal{I}, s'_\mathcal{I}) \leq 0$ by the computation above, proving $local(\varphi, s_\mathcal{I}, X)$. This proves $s_\mathcal{I} \in LC_\varphi(X)$. □

Conversely, we prove that a consistent IMC can be implemented by at least one MC. Note that the proof of Theorem 17 provides the construction of a concrete MC.

Theorem 17. *For IMC $\mathcal{I} = (S, s_0, \varphi)$, if $s_0 \in Cons$, then there exist a probability distribution μ and a MC $\mathcal{M} = (Cons, s_0, \mu)$ such that $\mathcal{M} \models \mathcal{I}$.*

Proof. Assume $\mathcal{I}$ is consistent. Consider an arbitrary state $s \in Cons$. We will define $\mu(s, s')$ in between $\varphi_\ell(s, s')$ and $\varphi_u(s, s')$, scaling it by a factor p such that $\mu(s)$ sums up to 1. Define $L := \sum_{s' \in Cons} \varphi_\ell(s, s')$ and $U := \sum_{s' \in Cons} \varphi_u(s, s')$. Set $p := \frac{1-L}{U-L}$ (or 0 if $L = U$). Finally, we define $\mu(s, s') := (1 - p) \cdot \varphi_\ell(s, s') + p \cdot \varphi_u(s, s'))$.

By Lemma 15, $s \in LC_\varphi(Cons)$, thus $L \leq 1 \leq U$. Hence $0 \leq p \leq 1$, and indeed $\varphi_\ell(s, s') \leq \mu(s, s') \leq \varphi_u(s, s')$. We check that $\mu(s)$ is a probability distribution:

$$\begin{aligned}\sum_{s' \in Cons} \mu(s, s') &= \sum_{s' \in Cons} \big((1 - p) \cdot \varphi_\ell(s, s') + p \cdot \varphi_u(s, s')\big) \\ &= (1 - p)L + pU = L + p(U - L) = L + \tfrac{1-L}{U-L}(U - L) = 1\end{aligned}$$

Finally, we show that $\mathcal{M}$ implements $\mathcal{I}$. Define $\mathcal{R} := \{(s, s) \mid s \in Cons\}$. Define $\delta(s', s') := 1$ and $\delta(s', s'') := 0$ for $s' \neq s''$. Properties (2a)– (2c) of Def. 7 follow directly from the definition of δ, so by Lemma 9, $\mathcal{M} \vDash \mathcal{I}$. □

3.3 Inductive n-consistency

Next, we define n-consistency for a state $s \in S$ in an IMC, inductively. This rephrases the definition from [12]. Intuitively, n-consistent states can perform n consistent steps in a row. That is, they can evolve to $(n - 1)$-consistent states.

Definition 18 (Consistency, inductive). *Define sets $Cons_n \subseteq S$ by recursion on n:*

$$\begin{aligned}Cons_0 &= S, \\ Cons_{n+1} &= LC_\varphi(Cons_n).\end{aligned}$$

Lemma 19. *We have the following basic facts on local consistency:*

1. $Cons_{n+1} \subseteq Cons_n$
2. *If $m \leq n$ then $Cons_n \subseteq Cons_m$.*
3. *If $Cons_{n+1} = Cons_n$ and $s \in Cons_n$ then $\forall m : s \in Cons_m$.*

Proof.

1. Induction on n. $n = 0$ is trivial. Let $s' \in Cons_{n+2} = LC_\varphi(Cons_{n+1})$. By induction hypothesis, $Cons_{n+1} \subseteq Cons_n$, so by the monotonicity lemma 12(1, $s' \in LC_\varphi(Cons_n) = Cons_{n+1}$, indeed.
2. Induction on $n - m$, using (1).
3. If $m > n$, we prove the result with induction on $m-n$. Otherwise the result follows from (2). □

Next, we show that universal n-consistency coincides with global consistency. Note that this depends on the assumption that the system is finitely branching.

Theorem 20. *Let $sup(\varphi, s)$ be finite for all s. Then $s \in \mathcal{C}ons \equiv \forall n : s \in Cons_n$.*

Proof. $\Rightarrow$: Induction on n. The base case is trivial. Assume $s \in \mathcal{C}ons$. Then $s \in LC_\varphi(\mathcal{C}ons)$. By induction hypothesis, $\mathcal{C}ons \subseteq Cons_n$. So, by monotonicity, Lemma 12(1), $s \in LC(Cons_n) = Cons_{n+1}$.

$\Leftarrow$: Assume that $s \in \bigcap_{n\geq 0} Cons_n$. Define $Y_n := Cons_n \cap sup(\varphi, s)$. By Lemma 19(1), $s \in Cons_{n+1} = LC(Cons_n) = LC_\varphi(Y_n)$ and Y_n is a decreasing sequence of finite sets (since φ has finite support). Hence it contains a smallest member, say Y_m. For Y_m, we have $s \in LC_\varphi(Y_m)$ and $Y_m \subseteq \bigcap_{n\geq 0} Cons_n$. By monotonicity, Lemma 12(1), $s \in LC_\varphi(\bigcap_{n\geq 0} Cons_n)$. So we found another fixed point, and by Lemma 15(2), $\bigcap_{n\geq 0} Cons_n \subseteq \mathcal{C}ons$, so indeed $s \in \mathcal{C}ons$. □

The following example shows that the condition on finite branching is essential. The situation is similar to the equivalence of the projective limit model with the bisimulation model in process algebras [3].

Example 21. Let $t \xrightarrow{[0,1]} t_i, \forall i$ and $t_{i+1} \xrightarrow{[0,1]} t_i$, see Fig. 4. Then $\forall i : t_i$ is i-consistent, but not $(i+1)$-consistent (since no transition exits t_0). So t is n-consistent for all n. However, no t_i is globally consistent, and $LC(t, \varnothing) = \bot$, so t is not globally consistent either.

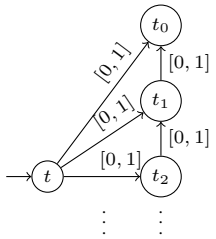


Fig. 4: Infinitely-branching IMC

Finally, we would like to limit the n that we need to check. It has been shown before [12] that when the number of states is finite, $n = |S|$ is sufficient. We now show that we can further bound the computation to the length of the longest simple path.

Definition 22 (Paths properties). *We define a (reverse)* path *of length n as a sequence $s_0, \ldots, s_{n-1}$, such that for all $0 \leq i < n$, $\varphi_u(s_{i+1}, s_i) > 0$. The path is* simple *if for all i, j, with $0 \leq i < j < n$, we have $s_i \neq s_j$. This path is* bounded *by m if $n \leq m$.*

Note that, for instance, a path (s_0, s_1, s_2) has length 3 according to this definition. The essential argument is provided by the following lemma:

Lemma 23. *If $s \in Cons_{n+1} \setminus Cons_{n+2}$, then there exists a s', with $\varphi_u(s, s') > 0$, such that $s' \in Cons_n \setminus Cons_{n+1}$.*

Proof. Assume $s \in Cons_{n+1}$ and $s \notin Cons_{n+2}$. By Lemma 12(2): $s \in LC_\varphi(Cons_n \cap sup(\varphi, s))$. Now if $Cons_n \cap sup(\varphi, s) \subseteq Cons_{n+1}$, by monotonicity we would have $s \in LC_\varphi(Cons_n \cap sup(\varphi, s)) \subseteq LC_\varphi(Cons_{n+1}) = Cons_{n+2}$, which contradicts the assumption. So there must be some $s' \in Cons_n$ with $\varphi_u(s, s') > 0$ and $s' \notin Cons_{n+1}$. □

Since $Cons_{n+1}(s)$ depends on $Cons_n$ of its direct successors only, consistency information propagates along paths. In particular, it propagates no longer than the longest simple path. This can be made more formal as follows:

Theorem 24. *If all simple paths from s are bounded by m, then:*

$$Cons_m(s) \equiv \forall n : Cons_n(s)$$

Proof. We first prove the following statement by induction on n:

(*) If $s \in Cons_n \setminus Cons_{n+1}$, then there exists a simple path from s of length $n+1$, $s_0, \ldots, s_n = s$, such that for all $0 \leq i \leq n$, $s_i \notin Cons_{i+1}$.

Case $n = 0$: we take the path $[s]$. Case $n+1$: Let $s \in Cons_{n+1}$ but $s \notin Cons_{n+2}$. By Lemma 23, we obtain some s' with $\varphi_u(s, s') > 0$ and $s' \in Cons_n \setminus Cons_{n+1}$. By induction hypothesis, we get a simple path $s_0, \ldots, s_n = s'$, with for all $0 \leq i \leq n$, $s_i \notin Cons_{i+1}$. Extend this path with $s_{n+1} := s$. We must show that the extended path is still simple, i.e. $s_i \neq s$ for $i \leq n$. Since $s \in Cons_{n+1}$, by Lemma 19 (2), $s \in Cons_{i+1}$ but $s_i \notin Cons_{i+1}$, so $s \neq s_i$.
Finally, to prove the theorem, assume $s \in Cons_m$, with all simple paths from s bounded by m. We prove $s \in Cons_n$ by induction on n. If $n \leq m$, $s \in Cons_n$ by Lemma 19(2). Otherwise, assume as induction hypothesis $s \in Cons_n$. Note that there is no simple path from s of length $n+1$ since $n+1 > m$. By the statement (*), we obtain $s \in Cons_{n+1}$. So $\forall n : s \in Cons_n$. □

To summarize, if the longest simple path from s_0 in $\mathcal{I} = (S, s_0, \varphi)$ is of length m, we can compute $s_0 \in Cons_m$. From Theorem 24, it follows that $\forall n : s_0 \in Cons_n$. By Theorem 20, we then obtain $s_0 \in \mathcal{C}ons$. By Theorem 17 we then know that there exists a Markov Chain $\mathcal{M} = (S, s_0, \mu)$, s.t. $\mathcal{M} \models \mathcal{I}$. Conversely, if there exists any $\mathcal{M}' = (S', s'_0, \mu')$ s.t. $\mathcal{M}' \models \mathcal{I}$, we know by Theorem 16 that $\mathcal{I}$ is consistent.

Example 25. Let us consider again the IMC in Fig. 2b. The longest simple paths from state 0 are $[0, 2, 1, 3]$ and $[0, 2, 4, 3]$, of length 4. Hence to check consistency of the IMC, it suffices to check that state 0 is in $Cons_4$.

4 Algorithms for Consistency Checking of IMCs

In this section, the developed theory is used to provide algorithms to check the consistency of a finite IMC (S, s_0, φ) (i.e. without parameters). In the next section, we will synthesize parameters for PIMCs that guarantee their consistency. For IMCs, we present a forward algorithm and a backward algorithm, which are both polynomial. The justification of these algorithms is provided by the following Corollary to Lemma 12 and Def. 18 and 14.

Corollary 26. *Let IMC (S, s_0, φ) be given, let $s \in S$ and $n \in \mathbb{N}$.*

1. $s \in Cons_{n+1} \equiv s \in LC_\varphi(Cons_n \cap sup(\varphi, s))$.
2. $s \in \mathcal{C}ons \equiv s \in LC_\varphi(\mathcal{C}ons \cap sup(\varphi, s))$.

The backward variant (Alg. 1) follows our simple co-inductive definition, rephrased as Corollary 26 (2). Initially, it is assumed that all states are consistent (**true**), which will be actually checked by putting them in the work list Q. When some state s is not locally consistent (LC, according to Def. 11), it is marked **false** (l. 5) and all predecessors t of s that are still considered consistent, are put back in the work list Q (l. 7–9).

The forward Alg. 2 is based on the inductive definition of consistency, rephrased in Corollary 26(1). It is called with a consistency level m and a state s to check if

Algorithm 1 Consistency (backward) $consistent(s)$

Require: $t.cons = \textbf{true}\ (\forall t \in S)$
1: $Q := S$
2: **while** $Q \neq \varnothing$ **do**
3: pick r from Q; $Q := Q \setminus \{r\}$
4: $X := \{t \in sup(\varphi, r) \mid t.cons = \textbf{true}\}$
5: **if** $\neg LC(\varphi, r, X)$ **then**
6: $r.cons := \textbf{false}$
7: **for all** t s.t. $r \in sup(\varphi, t)$ **do**
8: **if** $t.cons = \textbf{true}$ **then**
9: $Q := Q \cup \{t\}$
10: **return** $s.cons$

Co-inductive (backward) and inductive (forward) algorithms for checking consistency of IMCs.

Algorithm 2 Consistency (forward) $consistent(m)(s)$

Require: $t.pc = 0, t.nc = \infty\ (\forall t \in S)$
1: **if** $m \leq s.pc$ **then**
2: **return true**
3: **else if** $m \geq s.nc$ **then**
4: **return false**
5: $C := \varnothing$
6: **for all** $t \in sup(\varphi, s)$ **do**
7: **if** $consistent(m-1)(t)$ **then**
8: $C := C \cup \{t\}$
9: **if** $LC(\varphi, s, C)$ **then**
10: $s.pc := m$
11: **return true**
12: **else**
13: $s.nc := m$
14: **return false**

$s \in Cons_m$. It stores and reuses previously computed results, to avoid unnecessary computations. In particular, for each state, we store both the *maximal* level of consistency demonstrated so far (in field pc, positive consistency, initially 0), and the *minimal* level of inconsistency (in field nc, negative consistency, initially ∞). We exploit monotonicity to reuse these cached results: By Lemma 19(2), if a state is n-consistent for some $n \geq m$, then it is m-consistent as well (l. 1-2 of Alg. 2). By contraposition, if a state is not n-consistent for $n \leq m$, then it cannot be m-consistent (l. 3-4). In all other cases, all successors $t \in \sup(\varphi, s)$ are recursively checked and the $(m-1)$-consistent ones are collected in C (l. 5-8). Finally, we check the local consistency (LC) of C (l. 9), and store and return the result of the computation in l. 9-14.

Lemma 27. *Let $|S|$ be the number of states of the IMC and let $|\varphi| := |\{(s,t) \mid \varphi_u(s,t) > 0\}|$ be the number of edges. Let d be the maximal degree $d := max_s|sup(\varphi, s)|$.*

1. *Algorithm 1 has worst-case time complexity $\mathcal{O}(|\varphi| \cdot d)$ and space complexity $\mathcal{O}(|S|)$.*
2. *Algorithm 2 has worst-case time complexity $\mathcal{O}(|S| \cdot |\varphi|)$ and space complexity $\mathcal{O}(|S| \cdot \log_2 |S|)$.*

Proof. Note that in Alg. 1 every state is set to $\bot$ at most once. So every state s is added to Q at most $|\varphi_u(s)| + 1 = \mathcal{O}(d)$ times. Handling a state requires checking its incoming and outgoing edges, leading to $\mathcal{O}(|\varphi| \cdot d)$ time complexity. Only one bit per state is stored. Alg. 2 is called at most $m \leq |S|$ times per state. Every call inspects the outgoing edges a constant number of times, leading to time complexity $\mathcal{O}(|S| \cdot |\varphi|)$. It stores two integers, which are at most $|S|$, which requires $2\lfloor \log_2 |S| \rfloor + 1$ bits. □

Alg. 2 could start at m equal to the length of the longest simple path, which cannot be efficiently computed in general. Alg. 1 does not require computing any bound.

5 Parameter Synthesis for Parametric IMCs

In this section, we reconsider intervals with parameters from P. In particular, we present algorithms to synthesize the exact constraints on the parameters for which a PIMC is consistent. Note that given a PIMC and concrete values for all its parameters, we actually obtain the corresponding IMC, by just replacing the parameters by their values. This allows us to reuse all theoretical results on consistency from Section 3 on IMCs.

In particular, we can view parameters as "logical variables", and view a PIMC as a collection of IMCs. For instance, consider a PIMC (P, S, s_0, φ) with parameters $P = \{p, q\}$. Given a state s and a finite set of states $X \subseteq S$, the expression $LC(\varphi, s, X)$ can be viewed as a predicate over the logical variables p and q (appearing as parameters in φ). Also $\mathcal{C}ons(s)$ and $\mathcal{C}ons_n(s)$ can be viewed as predicates over p and q. In PVS, we can use universal quantification to lift results from IMCs to PIMCs. For instance, for PIMCs over P, Lemma 19.1 reads: $\forall p, q : \forall s \in S : \mathcal{C}ons_{n+1}(s) \Rightarrow \mathcal{C}ons_n(s)$.

Inspecting the expression $LC(\varphi, s, X)$ in Def. 11, one realizes that it is actually a constraint over p and q in linear arithmetic. However, due to the recursive/inductive nature of Def. 14 and 18, $\mathcal{C}ons$ and $\mathcal{C}ons_n$ are not immediately in the form of a Boolean combination over linear arithmetic constraints. We can rephrase the definition using an enumeration over all subsets of successors, similar to [12]. Since we consider finite PIMCs, the enumeration $\exists X \subseteq S$ corresponds to a finite disjunction. Doing this we get the following variation of the inductive and co-inductive consistency definition:

Corollary 28. *Let IMC (S, s_0, φ) be given, and let $s \in S$, $n \in \mathbb{N}$*

1. $s \in \mathit{Cons}_{n+1} \equiv \exists X \subseteq \sup(\varphi, s) : s \in LC_\varphi(X) \wedge X \subseteq \mathit{Cons}_n$
2. $s \in \mathcal{C}ons \equiv \exists X \subseteq \sup(\varphi, s) : s \in LC_\varphi(X) \wedge X \subseteq \mathcal{C}ons$

Proof. We only prove (1), since (2) is similar.
$\Rightarrow$: Choosing $X := \mathit{Cons}_n$ is sufficient by Lemma 19(1).
$\Leftarrow$: If for some X, $s \in LC_\varphi(X \cap \mathit{sup}(\varphi, s))$ and $X \subseteq \mathit{Cons}_n$, then by monotonicity, Lemma 12(1), $s \in LC_\varphi(\mathit{Cons}_n \cap \mathit{sup}(\varphi, s))$, so $s \in \mathit{Cons}_{n+1}$ by Lemma 19(1). □

It becomes clear that the synthesised parameter constraints will be Boolean combinations over linear arithmetic constraints. In particular, expressions in DNF (disjunctive normal form) provide clear insight in all possible parameter combinations. The number of combinations can be quite large, but we noticed that in many examples, most of them are subsumed by only a few maximal solutions. In particular, we will use the following notations for operators on DNFs:

- $\top$, $\bot$ denote true and false (universe and empty set)
- $\sqcup$ for their union (including subsumption simplification)
- $\sqcap$ for their intersection (including transformation to DNF and simplification)
- $\sqsubseteq$ for their (semantic) subsumption relation

We have experimented with two prototype realizations of our algorithms. One approach is based on CLP (constraint logic programming) in SWI-Prolog [19] + CLP($\mathbb{Q}$). We wrote a small meta-program for the subsumption check. The other approach is based on Parma Polyhedra Library [1]. In particular, its Pointset Powerset Closed Polyhedra provide efficient implementations of the operations on DNFs.

5.1 Inductive Approach

In the inductive approach, Corollary 28(1) gives rise to a set of equations on variables $v_{n,s}$ (state s is n-consistent):

$$\textbf{let } v_{0,s} = \top$$
$$\textbf{let } v_{n+1,s} = \bigsqcup_{X \subseteq \sup(\varphi,s)} \left(LC_\varphi(X) \sqcap \bigsqcap_{t \in X} v_{n,t} \right)$$

Starting from the initial state $v_{|S|,s_0}$, we only need to generate the reachable equations that are not pruned away by inconsistent $LC_\varphi(X)$ constraints, and we need to compute each equation only once. Note that there will be at most $|S|^2$ reachable equations. The number of conjunctions per equation is bounded by 2^d, the number of $X \subseteq sup(\varphi, s)$, and for each X we build a conjunction of length $O(d)$. So, the size of the whole set of equations is bounded by $O(|S|^2.d.2^d)$. In general, however, there is no polynomial upper bound on the size of the corresponding solution. Also, note that by Theorem 24, we could replace $|S|$ by the length of the longest simple path.

The set of equations can be interpreted directly as a Constraint Logic Program in Prolog, with predicates `cons(N,S)`. Prolog will compute the dependency tree for $s \in Cons_n$, by backtracking over all choices for X. Along the way, all encountered LC_φ predicates are asserted as constraints to the CLP solver. This has the advantage that locally inconsistent branches will be pruned directly, without ever generating their successors. By enumerating all feasible solutions, we obtain the complete parameter space as a disjunction of conjunctive constraints.

However, the computation tree has a lot of duplicates, and the number of returned results is very high, since we start out with a deeply nested and-or-expression. We provide a more efficient version in Alg. 3. Here recomputations are avoided by caching all intermediate results in a *Table* (see l. 3 and 12). For each enumerated subset of successors X (l. 6), the algorithm checks $(n-1)$-consistency. Note that we "shortcut" this computation as soon as we find that either X is not locally consistent, or some $t \in X$ is not $(n-1)$-consistent (l. 8). The final optimization is to suppress all subsumed results in the resulting conjunctions and disjunctions. We show this by using $\sqcap$ and $\sqcup$. This drastically reduces the number of returned disjunctions.

We have implemented Alg. 3 in Prolog+CLP, using meta-programming techniques to suppress subsumed results. Alternatively, one could implement the algorithm directly on top of the Parma Polyhedra Library.

5.2 Co-inductive approach

Next, we show how to encode co-inductive consistency in a Boolean Equation System (BES) (over sets of polyhedra). Here, the equations will be recursive. The largest solution for variable v_i indicates that state s_i is consistent. This solution provides then a description of the set of all parameters for which the PIMC is consistent.

Definition 29 (BES in DNF). *Given a PIMC $\mathcal{P} = (P, S, s_0, \varphi)$, we define the BES as the following set of equations, for each formal variable v_s, $s \in S$:*

$$\left\{ v_s = \bigsqcup_{X \subseteq \sup(\varphi,s)} \left(LC_\varphi(X) \sqcap \bigsqcap_{t \in X} v_t \right) \;\middle|\; s \in S \right\}$$

We can bound the size of this BES and the number of iterations for its solution. Again, the size of the final solution cannot be bounded polynomially.

Lemma 30. *For each PIMC $\mathcal{P} = (P, S, s_0, \varphi)$, with out-degree bounded by d, the corresponding BES has size $O(|S|.d.2^d)$. The BES can be solved in $O(\ell)$ iterations, where ℓ is the length of the longest simple path in $\mathcal{P}$.*

Proof. We have $|S|$ equations of size at most $O(d.2^d)$ each. The BES can be solved by value iteration. Let F_s denote the right hand side of the equation for v_s. We can compute the largest solution by iteration and substitution as follows:

$$\sigma_0 = \lambda s.\top$$
$$\sigma_{n+1} = \lambda s.F_s[v_t \mapsto \sigma_n(t) \mid t \in S]$$

By monotonicity, $\sigma_n \supseteq \sigma_{n+1}$ (pointwise). We can terminate whenever $\sigma_n \subseteq \sigma_{n+1}$. Since it can be proved by induction that $\sigma_n \equiv Cons_n$, the process terminates within ℓ steps by Lemma 24. □

Solving this BES can be done by straightforward value iteration, see Lemma 30. The intermediate expressions can be viewed as collections of polyhedra, represented e.g. by the Parma Polyhedra Library as Powersets of Convex Polyhedra [1]. Alg. 4 provides a variant of the iteration in Lemma 30. Here we only update the polyhedra for nodes whose successors have been modified. To this end, we maintain a worklist Q of states that we must check initially (l. 1) or when their successors are updated (l. 9). This algorithm can be viewed as the parametric variant of the backward Alg. 1.

Example 31. On the example in Fig 2a, Def. 29 gives rise to the following equation system (simplifying trivial arithmetic inequalities and global bounds $0 \leq p, q \leq 1$).

Algorithm 3 Inductive Parameter Synthesis Algorithm $Cons(s, n)$

Require: Initialize: $Table := \varnothing$
1: **if** $n = 0$ **then**
2: **return** $\top$
3: **if** $\exists R : (s, n, R) \in Table$ **then**
4: **return** R
5: $D := \bot$
6: **for all** $X \subseteq sup(\varphi, s)$ **do**
7: $C := LC_\varphi(X)$
8: **while** $X \neq \varnothing \wedge C \neq \bot$ **do**
9: pick t from X; $X := X \setminus \{t\}$
10: $C := C \sqcap Cons(t, n-1)$
11: $D := D \sqcup C$
12: add (s, n, D) to $Table$
13: **return** D

Algorithm 4 Co-inductive Parameter Synthesis Algorithm

Require: Initialize: $s.sol := \top$ $(\forall s \in S)$
1: $Q := S$
2: **while** $Q \neq \varnothing$ **do**
3: pick s from Q; $Q := Q \setminus \{s\}$
4: $sol := \bigsqcup_{X \subseteq \mathrm{sup}(\varphi, s)} (LC_\varphi(X) \sqcap \bigsqcap_{t \in X} t.sol)$
5: **if** $sol \sqsubset s.sol$ **then**
6: $s.sol := sol$
7: **for all** t s.t. $s \in sup(\varphi, t)$ **do**
8: **if** $t.sol \neq \bot$ **then**
9: $Q := Q \cup \{t\}$
10: **return** $s_0.sol$

Inductive and co-inductive algorithms for parameter synthesis in PIMCs.

```
v0 = v1                              v2 = v1 & p=1
   | v2                                 | v2 & q=1
   | v1 & v2                            | v1 & v2 & p+q>=1
v1 = v1 & v3 & q>=0.3 & q=<0.7          | v1 & v4 & p>=0.5
v3 = v3                                | v2 & v4 & q>=0.5
v4 = false                             | v1 & v2 & v4 & p+q>=0.5
```

We solve the simplified BES by value iteration as follows. In Approximation 0, each `vi=true`. We show approximations 1–3, which is the stable solution. From the final result, we can conclude that the initial state in Fig. 2a is consistent, if and only if $0.3 \leq q \leq 0.7 \vee q = 1$.

```
Approximation: 1       Approximation: 2       Approximation: 3
v0 = true              v0 = p+q>=0.5          v0 = q>=0.3 & q=<0.7
                          | q>=0.3 & q=<0.7      | q=1
v1 = q=<0.7 & q>=0.3   v1 = q>=0.3 & q=<0.7            v1: idem
v2 = p+q>=0.5          v2 = p+q>=1 & q>=0.3 & q=<0.7   v2: idem
                          | q=1
v3 = true              v3 = true                      v3 = true
v4 = false             v4 = false                     v4 = false
```

6 Experiments

To get an indication of the effectiveness of our algorithms and optimizations, we performed some experiments in a Prolog prototype implementation on Lehmann and Rabin's randomized dining philosophers from Prism [15]. First, we modified that DTMC to an IMC by replacing probabilities like 0.1666 by intervals $[0.1, 0.2]$. This made the consistency analysis numerically more stable. Subsequently, we replaced some probabilities by parameters to get a PIMC, and experimented with different intervals as featured in the first column of Table 1, for one parameter P or two parameters P and Q. The number of edges with these parameters is given in the third column of the table. We compared the inductive algorithm for increasing n-consistency with the co-inductive algorithm. Column CLP shows our Prolog implementation of the inductive algorithm from [12]. CLP+opt corresponds to the inductive Alg. 3, including our optimizations, i.e. caching and subsumption. PPL shows our co-inductive Alg. 4. We carried out the experiments on the model for 3 philosophers (956 states, 3625 edges) and 4 philosophers (9440 states, 46843 edges).

Table 1 shows the results; all times are measured with the SWI-Prolog library `statistics`. Increasing n-consistency highly impacts the performance. It is clear that caching and subsumption provide useful optimizations, since we can now compute consistency for much higher n. The co-inductive variant with PPL wins, since it is always faster than CLP+opt for $n = 50$. We also observe that the increase time from 3 to 4 philosophers is considerable. This is consistent with the complexity result (note that for phil4, $d \approx 5$ so $d \cdot 2^d \approx 150$).

Note that PPL computes consistency constraints for *all* states and for *all* n, whereas CLP only computes this for small n and the initial state. As an example, the constraint computed by PPL for 3 philosophers, 2 parameters, intervals $[0, P]$ and $[0.3, Q]$ is $(3P \geq 0.5 \wedge Q \geq 0.34) \vee (2P + Q \geq 1 \wedge p \geq 0.25 \wedge 3Q \geq 1)$.

interval(s)	model	param. edges	CLP n-inductive		CLP+opt n-inductive		PPL co-inductive
			$n=2$	$n=3$	$n=10$	$n=50$	$n=\infty$
$[P, P{+}0.1]$	phils3	723	0.01	0.02	3.82	82.01	5.08
	phils4	14016	0.03	0.14	51.84	2506.61	360.02
$[0, P]$	phils3	723	0.29	timeout	5.45	123.42	8.78
	phils4	14016	181.15	timeout	82.18	timeout	1605.94
$[P, 1]$	phils3	723	0.21	timeout	5.02	102.36	8.67
	phils4	14016	100.57	timeout	77.62	3300.53	1016.90
$[P, Q]$	phils3	723	0.35	timeout	27.35	timeout	10.33
	phils4	14016	472.79	timeout	118.64	timeout	1649.00
$[0, P]$, $[0.3, Q]$	phils3	723 + 1416	0.30	timeout	122.66	timeout	18.33
	phils4	14016 + 688	318.91	timeout	834.77	timeout	2575.11
$[P, 1]$, $[0.3, Q]$	phils3	723 + 1416	0.22	timeout	13.53	893.27	13.52
	phils4	14016 + 688	161.69	timeout	84.61	timeout	1853.24

Table 1: Experimental results. All entries are time in seconds. Timeout was 1 hour.

7 Perspectives

Using inductive and co-inductive definitions of consistency, we provided forward and backward algorithms to synthesize an expression for all parameters for which a PIMC is consistent. The co-inductive variant, combined with a representation based on convex polyhedra, provides the most efficient algorithm. We believe that our work extends straightforwardly when the intervals are replaced by arbitrary linear constraints, since both CLP and PPL can handle linear constraints. The resulting formalism would generalize (linear) Constraint Markov Chains [11] by adding global parameters. Also, our approach should apply directly to parameter synthesis for consistent reachability [12].

We also plan to experiment with other case studies, e.g. the benchmarks of [2], with an implementation that is more elaborate than our initial prototype. This would give insight on how the algorithms scale up w.r.t. the number of parameters and of parametric edges. Finally, [2] gives a polynomial size CLP with extra variables, but doesn't address the parameter synthesis problem. Polynomial size CLP programs without extra variables can be obtained using if-then-else, e.g. $(v_0?1:0)+(v_1?1:0)+(v_2?p:0)\leq 1$. However, we don't know of a method to solve equation systems with conditionals without expanding them to large intermediate expressions.

Acknowledgement. The authors would like to thank the reviewers for their extensive comments, which helped them to improve the paper. They acknowledge the support of University Paris 13 and of the Van Gogh project PAMPAS, that covered their mutual research visits.

References

1. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
2. Anicet Bart, Benoît Delahaye, Didier Lime, Eric Monfroy, and Charlotte Truchet. Reachability in Parametric Interval Markov Chains Using Constraints. In *Quantitative Evaluation of Systems (QEST'17)*, pages 173–189, 2017.
3. Jan A. Bergstra and Jan Willem Klop. The algebra of recursively defined processes and the algebra of regular processes. In *ICALP'84*, pages 82–94, 1984.
4. Milan Ceska, Petr Pilar, Nicola Paoletti, Lubos Brim, and Marta Z. Kwiatkowska. PRISM-PSY: precise GPU-accelerated parameter synthesis for stochastic systems. In *TACAS'16*, pages 367–384, 2016.
5. Souymodip Chakraborty and Joost-Pieter Katoen. Model checking of open Interval Markov Chains. In *Analytical and Stochastic Modelling Techniques and Applications - 22nd International Conference, ASMTA 2015, Albena, Bulgaria, May 26-29, 2015. Proceedings*, pages 30–42, 2015.
6. Taolue Chen, Tingting Han, and Marta Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time Markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.
7. Conrado Daws. Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In *ICTAC'04*, pages 280–294, 2004.
8. Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. Prophesy: A probabilistic parameter synthesis tool. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 214–231, 2015.
9. Benoît Delahaye. Consistency for Parametric Interval Markov Chains. In *SynCoP'15*, pages 17–32, 2015.
10. Benoît Delahaye, Joost-Pieter Katoen, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, Falak Sher, and Andrzej Wasowski. Abstract probabilistic automata. *Information and Computation*, 232:66–116, 2013.
11. Benoît Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. New results for Constraint Markov Chains. *Perform. Eval.*, 69(7-8):379–401, 2012.
12. Benoît Delahaye, Didier Lime, and Laure Petrucci. Parameter Synthesis for Parametric Interval Markov Chains. In *VMCAI'16*, pages 372–390, 2016.
13. Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric markov models. *STTT*, 13(1):3–19, 2011.
14. Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *LICS'91*, pages 266–277, 1991.
15. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011.
16. Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Asp. Comput.*, 19(1):93–109, 2007.
17. S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th IC on Automated Deduction (CADE)*, LNAI 607, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.
18. Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Parameter synthesis for Markov models: Faster than ever. In *Automated Technology for Verification and Analysis - ATVA 2016*, pages 50–67, 2016.
19. Jan Wielemakers. SWI-prolog version 7 extensions. *WLPE-2014*, July 2014.