Parametric Interval Markov Chains: Synthesis Revisited

Laure Petrucci[1⋆] and Jaco van de Pol[2⋆⋆]

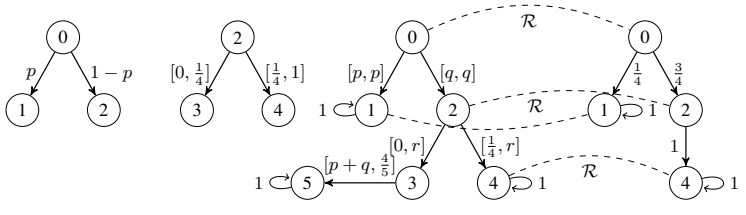
[1] LIPN, CNRS UMR 7030, Université Paris 13, Sorbonne Paris Cité, Villetaneuse, France
[2] Formal Methods and Tools, University of Twente, Enschede, The Netherlands

1 Introduction

Markov Chains (MC) are widely used to model stochastic systems, like randomised protocols, failure and risk analysis, and modelling 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) [3] and Interval Markov Chains (IMC) [7], in which unknown probabilities are replaced by parameters or intervals, respectively; see the diagram (a), (b). Following [4], we study their common generalization, Parametric Interval Markov Chains (PIMC), which allow intervals with parametric bounds. 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 [6].

IMCs can be viewed as specifications of MCs. An IMC is consistent if there exists some MC that implements it. The main requirements of this relation are: (1) all behaviour of the MC can be simulated by the IMC; (2) the transition probabilities out of each state sum up to 1. The consistency synthesis problem for PIMCs computes all parameter values leading to a consistent IMC. E.g., PIMC (c) is consistent when $q = 0, p = 1$, or $p + q = 1, r = 1$. The implementation MC (d) corresponds to $r = 1, p = \frac{1}{4}, q = \frac{3}{4}$.

Our contribution is a simplification of the theory in [6], together with a formal verification in PVS. In particular, we provide a simple co-inductive definition of consistency, and prove that it coincides with the inductive notion of n-consistency in [6]. Finally, based on these definitions, we provide algorithms for the consistency check of IMCs and the consistency parameter synthesis of PIMCs. We implemented the latter as a Constraint Logic Program over the reals, in SWI-prolog with clp(R).



(a) a PMC (b) an IMC (c) a PIMC (d) MC ⊨ PIMC

⋆ This work has been partially supported by project PACS ANR-14-CE28-0002.
⋆⋆ This research was performed when the author was invited professor at Université Paris 13.

2 Consistency of Parametric Interval Markov Chains

We first define (Parametric Interval) Markov Chains and their relation. For PIMCs, P denotes the set of parameters and $\mathsf{Int}[0,1](P)$ denotes intervals $[p,q]$ with $p,q \in P \cup [0,1]$. For IMCs, $P = \varnothing$ and PMCs only have point intervals $[p,p]$.

Definition 1 (MC, PIMC, $\models$, consistency [7,4]). *A* Markov Chain *(MC) is a tuple $\mathcal{M} = (S, s_0, \mu)$, with initial state $s_0 \in S$ and probability distribution $\mu : S \times S \to [0,1]$, i.e., $\forall s \in S : \sum_{s' \in S} \mu(s,s') = 1$. A* Parametric Interval Markov Chain *(PIMC) is a tuple $\mathcal{I} = (T, t_0, P, \varphi)$, with initial state $t_0 \in T$, parameters P, and parametric probability distribution $\varphi : T \times T \to \mathsf{Int}[0,1](P)$.*
$\mathcal{M}$ implements *$\mathcal{I}$ ($\mathcal{M} \models \mathcal{I}$) if there exists a simulation relation $\mathcal{R} \subseteq S \times T$, such that for all $s\mathcal{R}t$, there exists a probabilistic correspondence $\delta : S \times T \to \mathsf{Int}[0,1]$, with:*

1. $\forall t' \in T : \sum_{s' \in S} \mu(s,s') \cdot \delta(s',t') \in \varphi(t,t')$
   (the total contribution of the implementing transitions satisfies the specification)
2. $\forall s' \in S : \mu(s,s') > 0 \Rightarrow \sum_{t' \in T} \delta(s',t') = 1$
   (the implementing transitions yield a probability distribution)
3. $\forall s' \in S, t' \in T : \delta(s',t') > 0 \Rightarrow s'\mathcal{R}t'$
   (corresponding successors are in the simulation relation)

A PIMC $\mathcal{I}$ is consistent if for some parameters P and MC $\mathcal{M}$, we have $\mathcal{M} \models \mathcal{I}(P)$.

We write $\varphi(t)(t')$ for $\varphi(t,t')$. For $\nu = \varphi(t)$, lower/upper bounds are denoted by $\nu(t') = [\nu_\ell(t'), \nu_u(t')]$. The support $sup(\nu) = \{t' \mid \nu_u(t') \neq 0\}$. From now on, we assume that $sup(\varphi(t))$ is a finite set, so all sums below are well-defined.

Local consistency of state s w.r.t. a selected subset $X \subseteq T$ indicates compatibility with the associated $\nu = \varphi(s)$: The lower and upper bounds of the selected transitions are properly ordered and summing them up admits the probability mass 1. The unselected transitions should allow probability 0.

Definition 2. *We define* local consistency constraints *$LC(\nu, X)$ as a conjunction of:*

$$up(\nu, X) = \textstyle\sum_{t \in X} \nu_u(t) \geq 1$$
$$low(\nu, X) = \textstyle\sum_{t \in X} \nu_\ell(t) \leq 1$$
$$local(\nu, X) = (\forall t \in X : \nu_\ell(t) \leq \nu_u(t)) \wedge (\forall t \notin X : \nu_\ell(t) = 0)$$

We now provide three characterisations of consistency for IMCs: $Cons$ contains those states whose set of $Cons$-successors are locally consistent. States in $Cons_n$ can perform n consistent steps in a row. The third definition will be needed for PIMCs.

Definition 3. *Let $\mathcal{I} = (T, t_0, \varphi)$ be an IMC. We define consistency as:*
co-inductive: *the largest set $Cons \subseteq T$ satisfying $\forall t : t \in Cons \equiv LC(\varphi(t), Cons)$.*
n-recursive: $t \in Cons_n \equiv n > 0 \Rightarrow LC(\varphi(t), Cons_{n-1})$
n-recursive (X): $t \in \mathcal{C}ons_n^X \equiv n > 0 \Rightarrow \exists X : LC(\varphi(t), X) \wedge X \subseteq \mathcal{C}ons_{n-1}^X$

Theorem 1. *Let $\mathcal{I} = (T, t_0, \varphi)$ be an IMC.*

1. *$t_0 \in \mathcal{C}ons$ if and only if $\mathcal{M} \models \mathcal{I}$ for some MC $\mathcal{M} = (S, s_0, \mu)$.*
2. *For all $t \in T$, $t \in Cons$ if and only if $\forall n \in \mathbb{N} : t \in Cons_n$.*
3. *For all $t \in T$, $t \in Cons_n$ if and only if $t \in \mathcal{C}ons_n^X$.*
4. *If all simple paths from t have length $\leq m$, then $Cons_m(t) \equiv \forall n : Cons_n(t)$.*

The full version of this paper will contain the proofs, which have been checked in the interactive theorem prover PVS, based on higher-order classical logic. Here we remark that Theorem 1(2) essentially requires the assumption that $\varphi(t)$ has finite support ($\forall t$).

3 Algorithms for Consistency Checking and Synthesis

Theorem 1(1) leads to the backward Algorithm 1 to check IMC consistency (cf. Appendix A), while Thm. 1(2) justifies the forward Algorithm 2. Thm. 1(3) uses finite quantifications only, so it provides a basis to generate a consistency constraint over the parameters as a nested conjunction/disjunction over linear inequalities, as in Alg. 3. Finally, Thm. 1(4) lowers the upper bound on n from $|T|$ [6] to the length of the longest simple path. For instance, in binary trees the length of longest path is $\log(|T|)$.

Theorem 2. *Let (S, s_0, φ) be an IMC, with maximal outdegree d.*

1. *Algorithm 1 checks consistency in time $\mathcal{O}(|\varphi| \cdot d)$.*
2. *Algorithm 2 checks consistency in time $\mathcal{O}(|S| \cdot |\varphi|)$.*
3. *Algorithm 3 synthesises consistency constraints in time $\mathcal{O}(|S|^3 \cdot 2^d)$.*

We prototyped Algorithm 3 as a Constraint Logic Program (linear arithmetic over reals). Appendix A contains a direct implementation in SWI-prolog over clp(R). It uses backtracking to enumerate all subsets X. The constraint solver is used to prune inconsistent computations early. However, note that this corresponds to unfolding the constraints to a disjunctive normal form, so the polynomial time bound will be lost.

4 Conclusion

The algorithms can be further improved by storing intermediate results, pruning irrelevant parts of the computation earlier, and treating strongly connected components in isolation. We are implementing the algorithms, in order to run them on realistic PMCs from Prism [1] and IMCs from Tulip [2]. We would like to study the scalability of parallel synthesis algorithms as well. Finally, it would be interesting to generalise our results to Parametric Interval Markov Decision Processes, thus generalising results for APA [5].

References

1. Prism – probabilistic model checker. `http://www.prismmodelchecker.org`
2. Tulip – Interval Markov Chains. `http://tulip.lenhardt.co.uk/index.jsp`
3. Daws, C.: Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In: ICTAC'04. pp. 280–294 (2004)
4. Delahaye, B.: Consistency for Parametric Interval Markov Chains. In: SynCoP'15. pp. 17–32 (2015)
5. Delahaye, B., Katoen, J., Larsen, K.G., Legay, A., Pedersen, M.L., Sher, F., Wasowski, A.: Abstract probabilistic automata. Information and Computation 232, 66–116 (2013)
6. Delahaye, B., Lime, D., Petrucci, L.: Parameter Synthesis for Parametric Interval Markov Chains. In: VMCAI'16. pp. 372–390 (2016)
7. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: LICS'91. pp. 266–277 (1991)

A Algorithms and SWI Prolog-clp(R) implementation

Algorithm 1 $ConsBwd(s)$

Require: Initialise $t.cons := \top$

$Q := S$
while $Q \neq \varnothing$ **do**
  pick s from Q
  $X := \{t \in sup(\varphi(s)) | t.cons = \top\}$
  if $\neg LC(\varphi(s), X)$ **then**
    $s.cons := \bot$
    for all t s.t. $\varphi(t, s) > 0$ **do**
      if $t.cons = \top$ **then**
        $Q := Q \cup \{t\}$
      end if
    end for
  end if
end while

Algorithm 2 $ConsFwd(m)(s)$

Require: Initialise $t.cons := 0$

if $m \leq s.cons$ **then**
  return true
end if
$C := \varnothing$
for all $t \in sup(\varphi(s))$ **do**
  if $ConsFwd(m-1)(t)$ **then**
    $C := C \cup \{t\}$
  end if
end for
if $LC(\varphi(s), C)$ **then**
  $s.cons := m$
  return true
else
  return false
end if

Algorithm 3 $Synth(m)(s)$

Require: Initialise $t.cons[m] := \bot$

if $s.cons[m] \neq \bot$ **then**
  return $s.cons[m]$
else
  $L := sup(\varphi(s))$
  $D :=$ **false**
  for all $X \subseteq L$ **do**
    $C := LC(\varphi(s), X)$
    for all $t \in X$ **do**
      $C := C \wedge Synth(m-1)(t)$
    end for
    $D := D \vee C$
  end for
  $s.cons[m] := D$
  return D
end if

```
% IMC is a list of state distributions, each state distribution
% is of the form state(source,[ trans(low,high,target), ...])

:- use_module(library(clpr)).
distribution(S,Dist,IMC) :- member(state(S,Dist),IMC).

lc(Dist,Set) :- low(Dist,Set), up(Dist,Set), local(Dist,Set).
low(Dist,Set) :- low(Dist,Set,Expr), {Expr =< 1}.
low([],_,0).
low([trans(A,_,State)|T],Set,A+B) :-
    member(State,Set),!,low(T,Set,B).
low([_|T],Set,B) :- low(T,Set,B).
up(Dist,Set) :-
    up(Dist,Set,Expr), {Expr >= 1}.
up([],_,0).
up([trans(_,A,State)|T],Set,A+B) :-
    member(State,Set),!,up(T,Set,B).
up([_|T],Set,B) :- up(T,Set,B).
local([],_).
local([trans(L,U,State)|T],Set) :-
    member(State,Set),!, {0=<L,L=<U,U=<1}, local(T,Set).
local([trans(L,_,_)|T],Set) :-
    {L=0}, local(T,Set).

subsets([],[]).
subsets(L,[_|K]) :- subsets(L,K).
subsets([A|L],[trans(_,_,A)|K]) :- subsets(L,K).
cons(0,S,IMC) :- !.
cons(N,S,IMC) :- distribution(S,Dist,IMC), subsets(Succ,Dist),
                 lc(Dist,Succ),N1 is N-1,consSucc(N1,IMC,Succ).
consSucc(_,_,[]).
consSucc(N,IMC,[S|Succ]) :- cons(N,S,IMC), consSucc(N,IMC,Succ).
```

Parameter Synthesis: Algorithm 3 in SWI-prolog with CLP(R).

```
example([P,Q,R],
 [ state(0,[trans(P,P,1),trans(Q,Q,2)]),
   state(1,[trans(0,1,1)]),
   state(2,[trans(0,R,3),trans(0.25,R,4)]),
   state(3,[trans(P+Q,0.8,5)]),
   state(4,[trans(0,1,4)]),
   state(5,[trans(0,1,5)])
 ]).

?- example([P,Q,R],PIMC),cons(3,0,PIMC).
P=0, Q=R=1 ;
P=1, Q=0   ;
P+Q=1, R=1 ;
no.
```

Example run, computing $Cons_3(0)$ on the PIMC (c) of the running example. Prolog synthesizes three answers. Note that the first answer is subsumed by the third answer. So the complete parameter space for which (c) is consistent is:

$$(p = 1 \wedge q = 0) \vee (p + q = 1 \wedge r = 1)$$