

Lumping Simulation

Quantitative Logics

12 June 2013

David N. Jansen

Simulation and bisimulation

- general notions on behaviour of many automata-like models.
- Given two states s and t in a model:
 - t simulates s $s \preceq t$
:= t can do everything that s does.
 - s and t are bisimilar $s \sim t$
:= s simulates t and t simulates s ,
through the same relation.
 - s and t are simulation-equivalent $s \approx t$
:= s simulates t and t simulates s , possibly through different relations.

Logical characterisation of (bi)simulation

- States s and t are bisimilar ($s \sim t$)



they satisfy the same formulas

- t simulates s ($s \preceq t$)



t satisfies all liveness formulas that s satisfies



s satisfies all safety formulas that t satisfies

Bisimilar states in a Markov chain

- Assume given a labelled MC (S, \mathbf{P}, L) .
- An equivalence relation $R \subseteq S \times S$ is a **bisimulation** if for all states $s, t \in S$,
 $s R t$ implies
 - $L(s) = L(t)$
 - For every R -equivalence class $C \in S/R$,
 $\mathbf{P}(s, C) = \mathbf{P}(t, C)$
- States s and t are **bisimilar** or **lumping equivalent** if \exists bisimulation relation R with $s R t$.

Logical characterisation of lumping equivalence

- States s and t are lumping equivalent ($s \sim t$)

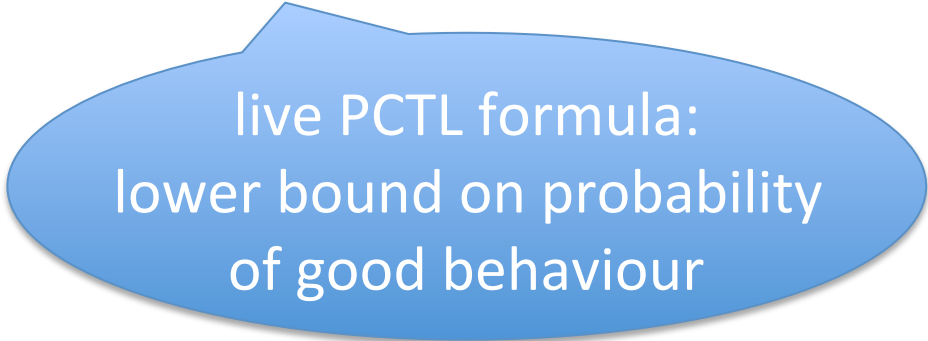


they satisfy the same PCTL formulas

- t simulates s ($s \preceq t$)



t satisfies all live PCTL formulas that s satisfies



live PCTL formula:
lower bound on probability
of good behaviour

Bisimulation minimisation (= Lumping)

- Bisimulation is an equivalence relation
- bisimulation quotient
= smallest model
that is bisimilar to original model
- Fast bisimulation minimisation makes
model checking faster
(and is guaranteed to be correct)

Partition refinement

- standard algorithm to find bisimulation equivalence classes
- maintains a partition of the states,
i.e. a division of states into disjoint subsets
 - an overapproximation to the equivalence classes

Partition refinement

- ① start with a coarse partition Π_0
- ② refine partition until fixpoint is reached
 - S_p is splitter for $B \in \Pi$ if the probability to enter S_p is not the same for every state $\in B$.
 - In that case, split B into subparts that have equal probability.
- ③ the resulting partition contains the bisimulation equivalence classes

Example

- Example on the blackboard.
(See Baier/Katoen, p. 811: Craps)

More Efficient Lumping

- not every new set in the partition becomes a (potential) splitter
- keep a list of potential splitters
- if a non-splitter block is split,
all **but one** subblock are potential splitters
 - works because total probability is constant

Partition refinement algorithm

Initialisation

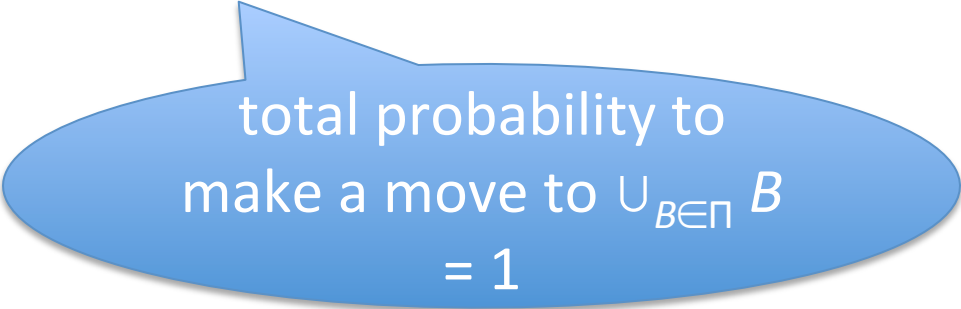
- /* Create a block for every possible label */

$$\Pi := \{ \{ s \mid L(s) = A \} \mid A \subseteq AP \} \setminus \{ \emptyset \}$$

- /* Create a list of potential splitters */

$$\mathcal{PS} := \Pi$$

Delete the largest element from \mathcal{PS} .



total probability to
make a move to $\cup_{B \in \Pi} B$
= 1

Partition refinement algorithm

Main loop

- while $\mathcal{PS} \neq \emptyset$
 - Pick any $Sp \in \mathcal{PS}$ and delete it from \mathcal{PS}
 - Split Π according to Sp
(This may change \mathcal{PS}).

Partition refinement algorithm

Split partition according to (potential) splitter Sp

- */* Calculate $\mathbf{P}(s, Sp)$ */*

Set all $\text{sum}(s)$ to 0.

For every state $t \in Sp$

– For every transition $s \rightarrow t$

- $\text{sum}(s) := \text{sum}(s) + \mathbf{P}(s, t)$
- Mark state s

- */* Refine partition */*

For each block $B \in \Pi$ containing marked states

– Split B into subblocks,

each containing all states with a specific $\mathbf{P}(s, Sp)$

Partition refinement algorithm

Split partition according to (potential) splitter S_p

- /* Calculate $\mathbf{P}(s, S_p)$ */
- /* Refine partition */

For each block $B \in \Pi$ containing marked states

- Split B into subblocks,
each containing all states with a specific $\mathbf{P}(s, S_p)$
- Replace B in Π by these subblocks
- If $B \in \mathcal{PS}$, replace B in \mathcal{PS} by these subblocks;
otherwise, add all subblocks of B to \mathcal{PS} ,
except the largest one

Time complexity

- Take only the necessary potential splitters
→ each state is at most $\log |S|$ times in a splitter
- “Split B into subblocks”
requires sorting with key $\mathbf{P}(s, Sp)$
- total complexity therefore $|\mathbf{P}| \cdot (\log |S|)^2$
- can be improved to $|\mathbf{P}| \cdot \log |S|$
by using a sort algorithm for equal keys

Revisiting Weak Simulation for Substochastic Markov Chains

David N. Jansen



Lei Song



Lijun Zhang



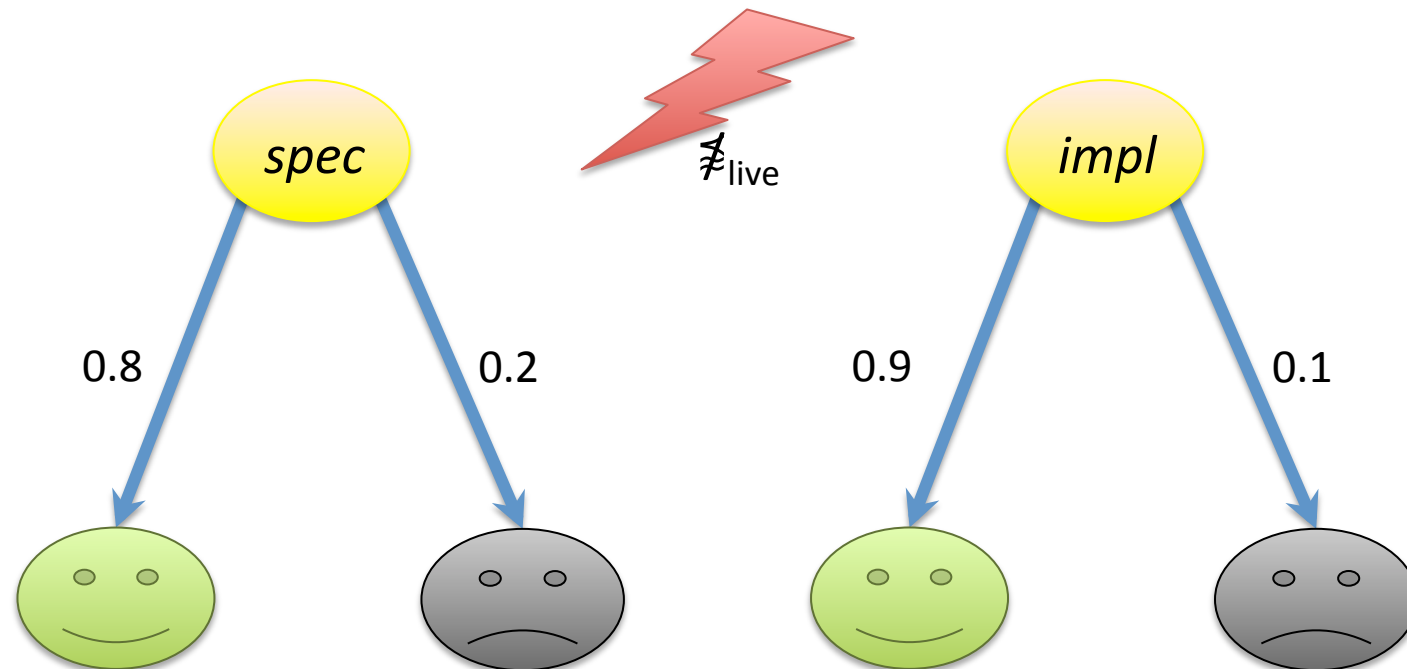
QEST 2013

substochastic DTMC

- A Markov chain consists of:
 - S finite set of states
(often $S = \{1, 2, \dots, n\}$)
 - $\mathbf{P}: S \times S \rightarrow [0,1]$ transition probability matrix
(with row sums ≤ 1)
 - $\pi_0: S \rightarrow [0,1]$ initial state distribution
(sometimes)
 - $L: S \rightarrow 2^{AP}$ labelling with
atomic propositions

Why substochastic?

The system works correctly with probability ≥ 0.8 .



Why substochastic?

The system works correctly with probability ≥ 0.8 .

