Scalable Parameter Synthesis from CTL Hypotheses

Samuel Pastva

Masaryk University
Brno

Bachelor Thesis, 2015

Goals

- Design a distributed memory algorithm that computes the Parameter Synthesis Problem for biochemical models and CTL hypotheses, i.e. states and parameter values of investigated model where given hypothesis is satisfied.

Goals

- Design a distributed memory algorithm that computes the Parameter Synthesis Problem for biochemical models and CTL hypotheses, i.e. states and parameter values of investigated model where given hypothesis is satisfied.
- Implement the algorithm and integrate it with existing modelling frameworks for ODE models and Thomas Networks (Biodivine and Parsybone).

Goals

- Design a distributed memory algorithm that computes the Parameter Synthesis Problem for biochemical models and CTL hypotheses, i.e. states and parameter values of investigated model where given hypothesis is satisfied.
- Implement the algorithm and integrate it with existing modelling frameworks for ODE models and Thomas Networks (Biodivine and Parsybone).
- Test and benchmark the algorithm on existing biochemical models.

Why CTL?

Similar technique already exist for LTL - Why do we need CTL for biochemical models?

- Different expressive power than LTL
  $CTL \not\subseteq LTL \wedge LTL \not\subseteq CTL$

Why CTL?

Similar technique already exist for LTL - Why do we need CTL for biochemical models?

- Different expressive power than LTL
  $CTL \not\subseteq LTL \wedge LTL \not\subseteq CTL$
- We can test for properties that can't be expressed in LTL, such as multistability

Why CTL?

Similar technique already exist for LTL - Why do we need CTL for biochemical models?

- Different expressive power than LTL
  $CTL \not\subseteq LTL \wedge LTL \not\subseteq CTL$
- We can test for properties that can't be expressed in LTL, such as multistability
- Globality - it is easier to obtain global information about whole model using CTL rather than LTL.

Model

As an input to our algorithm, we consider Parametrised Kripke Structure:

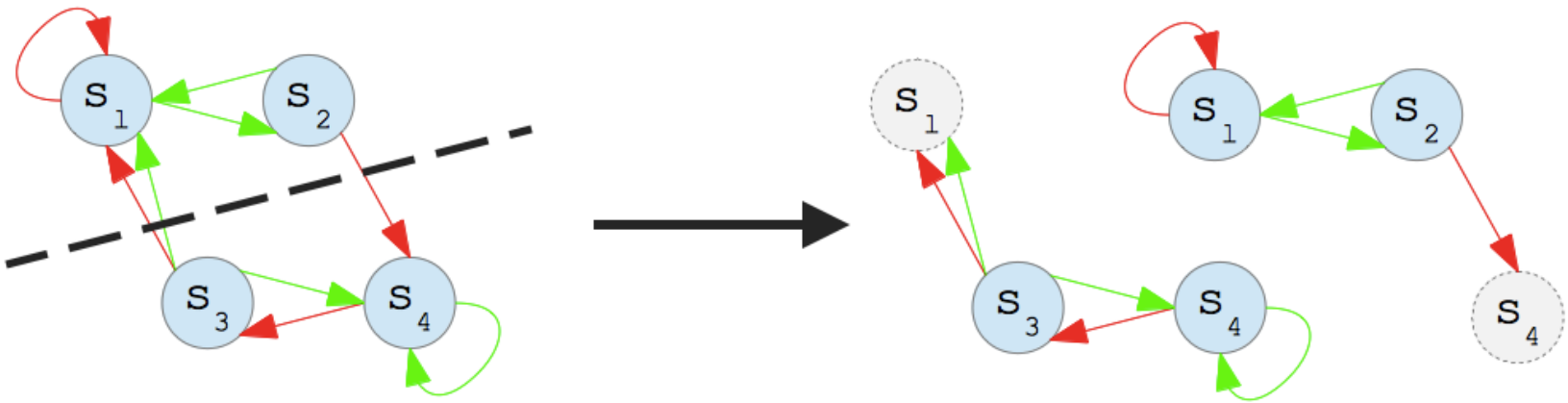
Parametrised Kripke Structure(PKS)

$\mathcal{K} = (\mathcal{P}, S, S_0, \rightarrow, L)$

- $\mathcal{P}$ is a finite set of parameters (all possible parameter valuations)
- S is a finite set of states
- $S_0 \subseteq S$ is a set of initial states
- $\rightarrow \subseteq S \times \mathcal{P} \times S$ is a transition relation labelled by parameter valuations
- $L : S \rightarrow 2^{AP}$ is a labelling function from states to sets of atomic propositions which are true in such states

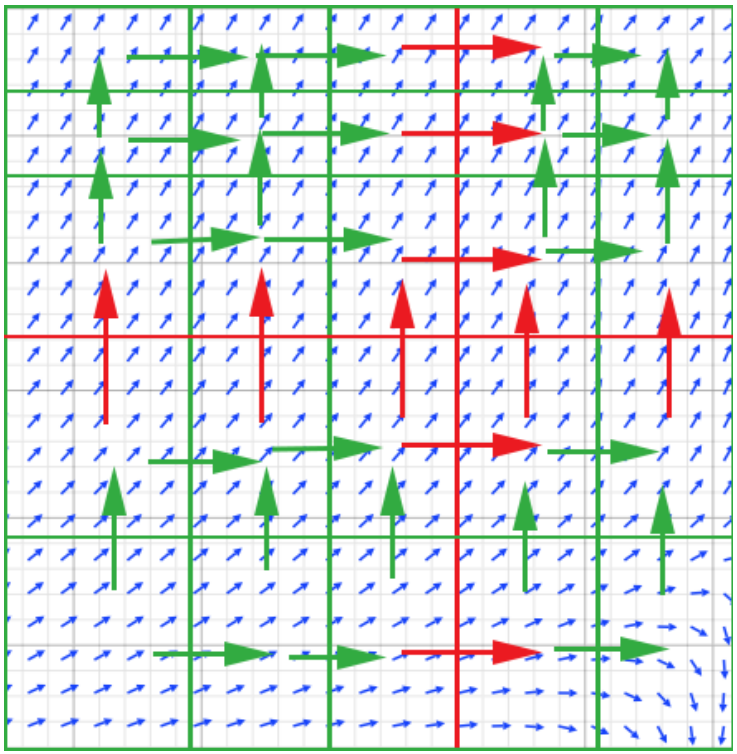
State Space Distribution

In order to distribute the computation across N workstations, we use distribution function $f : S \to \{1, \ldots, N\}$.
The distribution function divides the original PKS into N *Kripke Fragments*.
Each fragment contains states described in distribution function plus all direct successors of these states (border states).
Each border state represents a remaining portion of the state space located at another workstation.



Rectangular partitioning

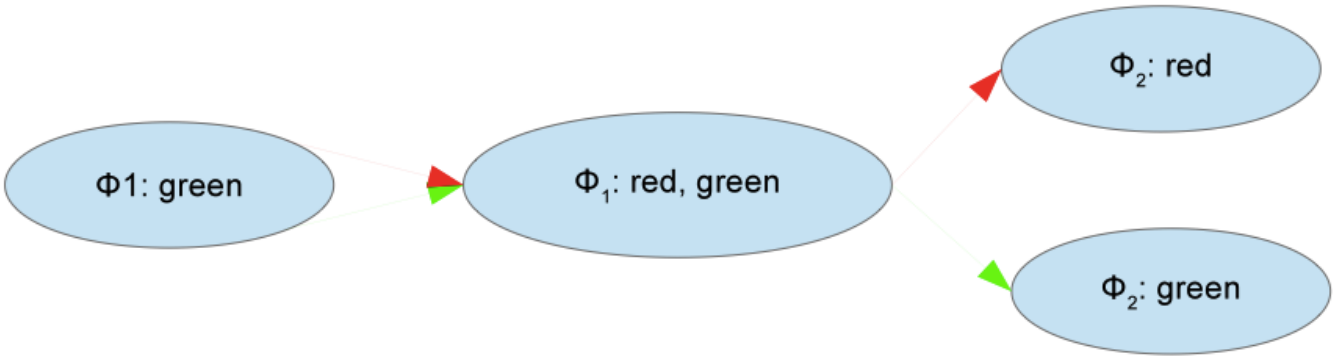
Observation: Biochemical models have rectangular structure with transitions only between adjacent states.
Optimization: Instead of using randomized hash-based partitioning, the state space is partitioned into similarly sized rectangular blocks. This way, we reduce the communication overhead.



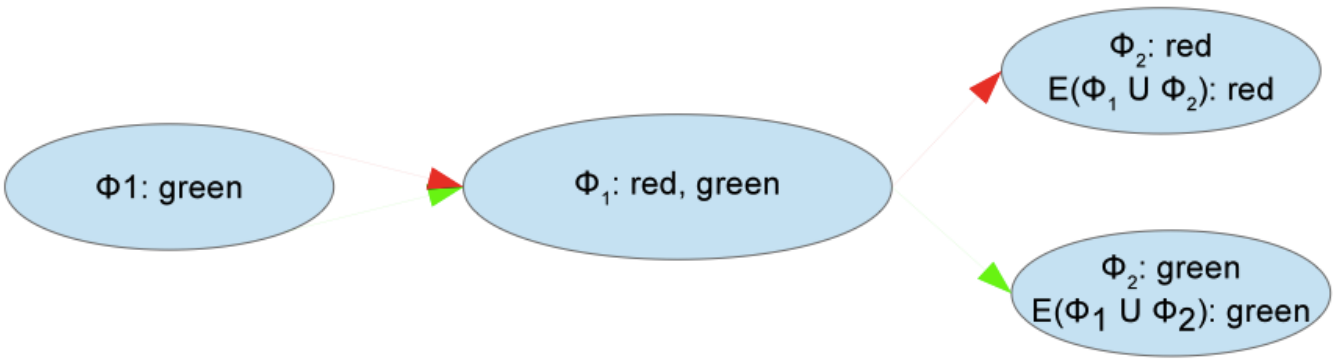
Algorithm

1. Based on distributed CTL model checking algorithm.
2. Uses coloured model checking heuristic to reduce runtime for big number of parameters.
3. Heuristic is based on two assumptions:
   1. Operations on parameter sets are cheaper than state space traversal.
   2. Small change in parameter leads to small change in transition system.
4. We combine all possible transition systems into one and mark the transitions with parameter values for which they are valid.
5. This way, we reduce the number of required traversals significantly, since part of the traversal can be shared by multiple parameters.

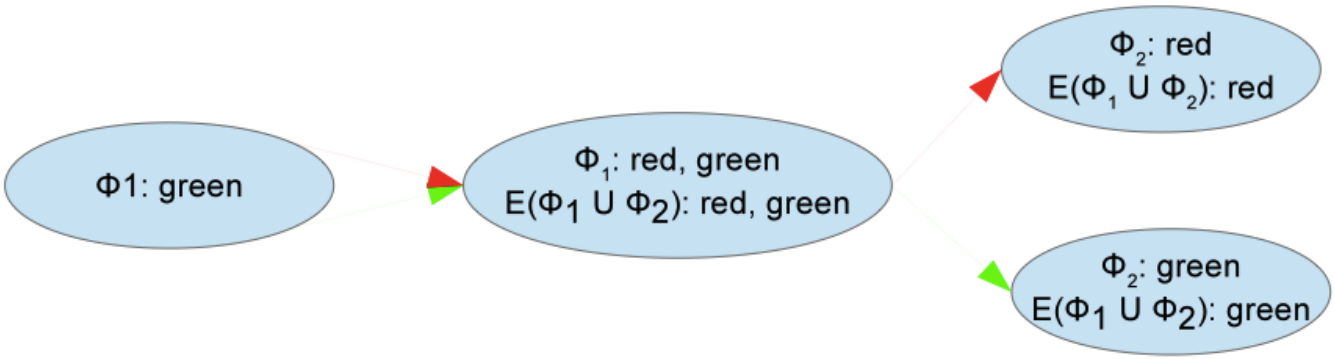
Example



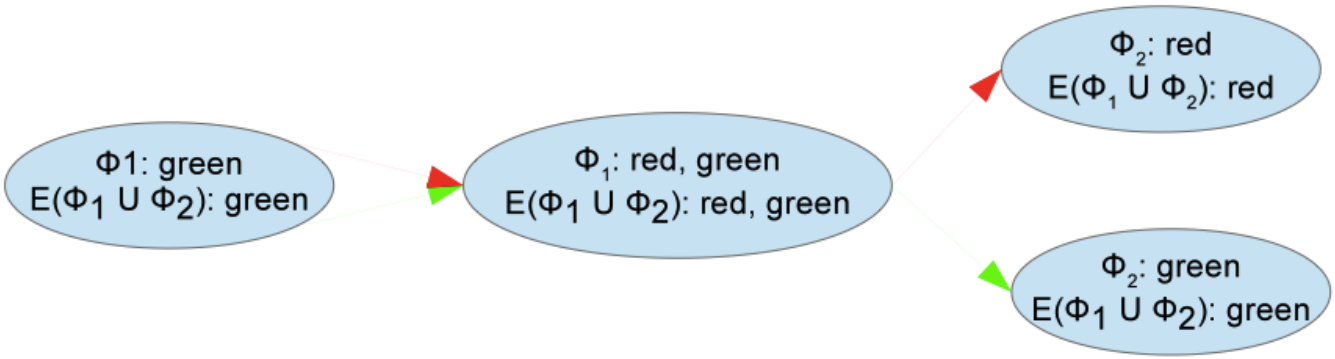
Example



Example

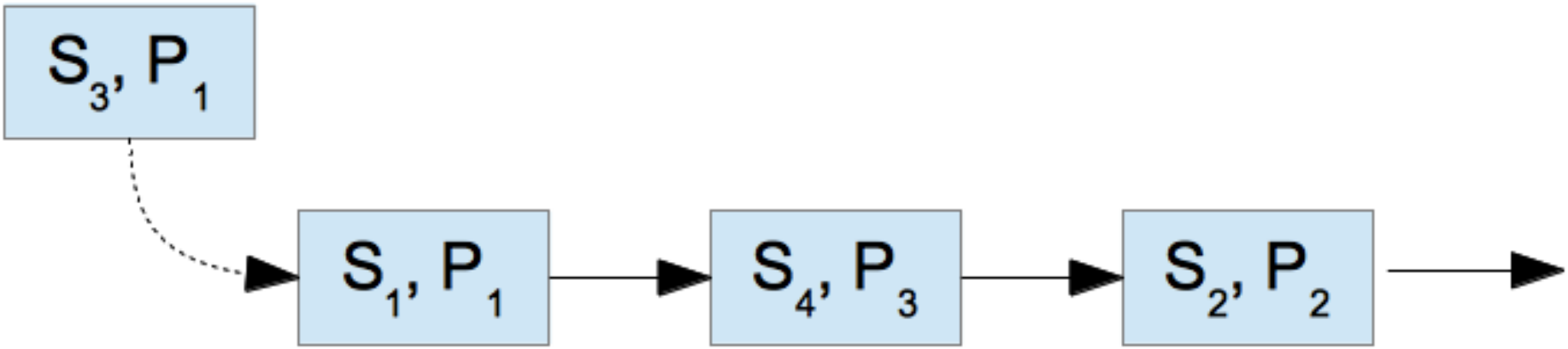

Φ1: green
Φ1: red, green
E(Φ1 U Φ2): red, green
Φ2: red
E(Φ1 U Φ2): red
Φ2: green
E(Φ1 U Φ2): green

Example


Φ1: green
E(Φ1 U Φ2): green
Φ1: red, green
E(Φ1 U Φ2): red, green
Φ2: red
E(Φ1 U Φ2): red
Φ2: green
E(Φ1 U Φ2): green

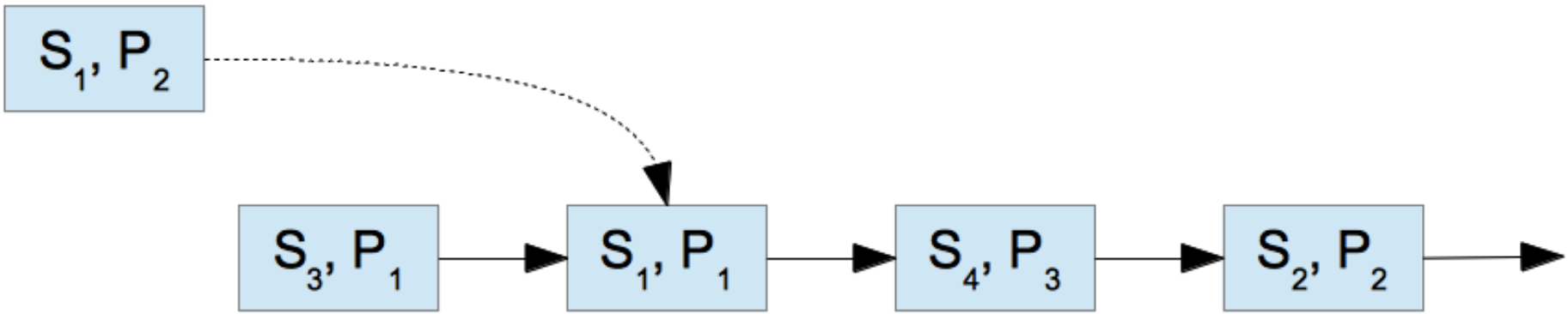
Merge Message Buffer

Observation: Number of messages can grow large when parameter space gets fragmented.
Solution: Representing message buffer as an iterable hash table. Two messages with same destination state can be merged into one.



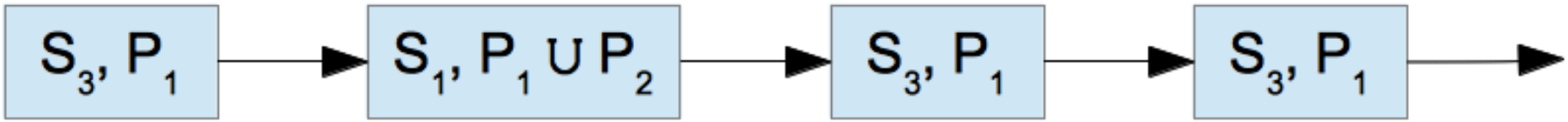
Merge Message Buffer

Observation: Number of messages can grow large when parameter space gets fragmented.
Solution: Representing message buffer as an iterable hash table. Two messages with same destination state can be merged into one.

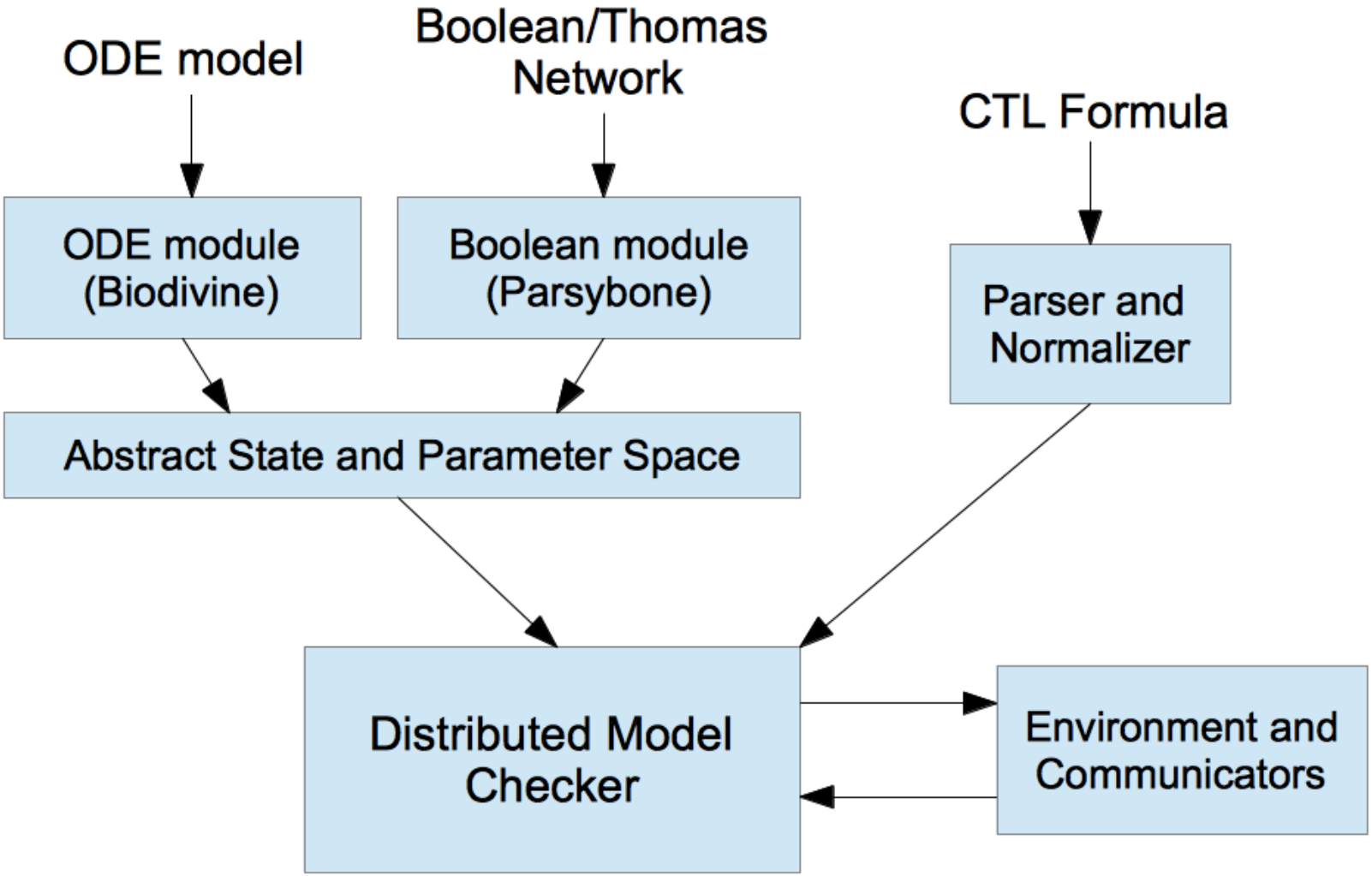


Merge Message Buffer

Observation: Number of messages can grow large when parameter space gets fragmented.
Solution: Representing message buffer as an iterable hash table. Two messages with same destination state can be merged into one.

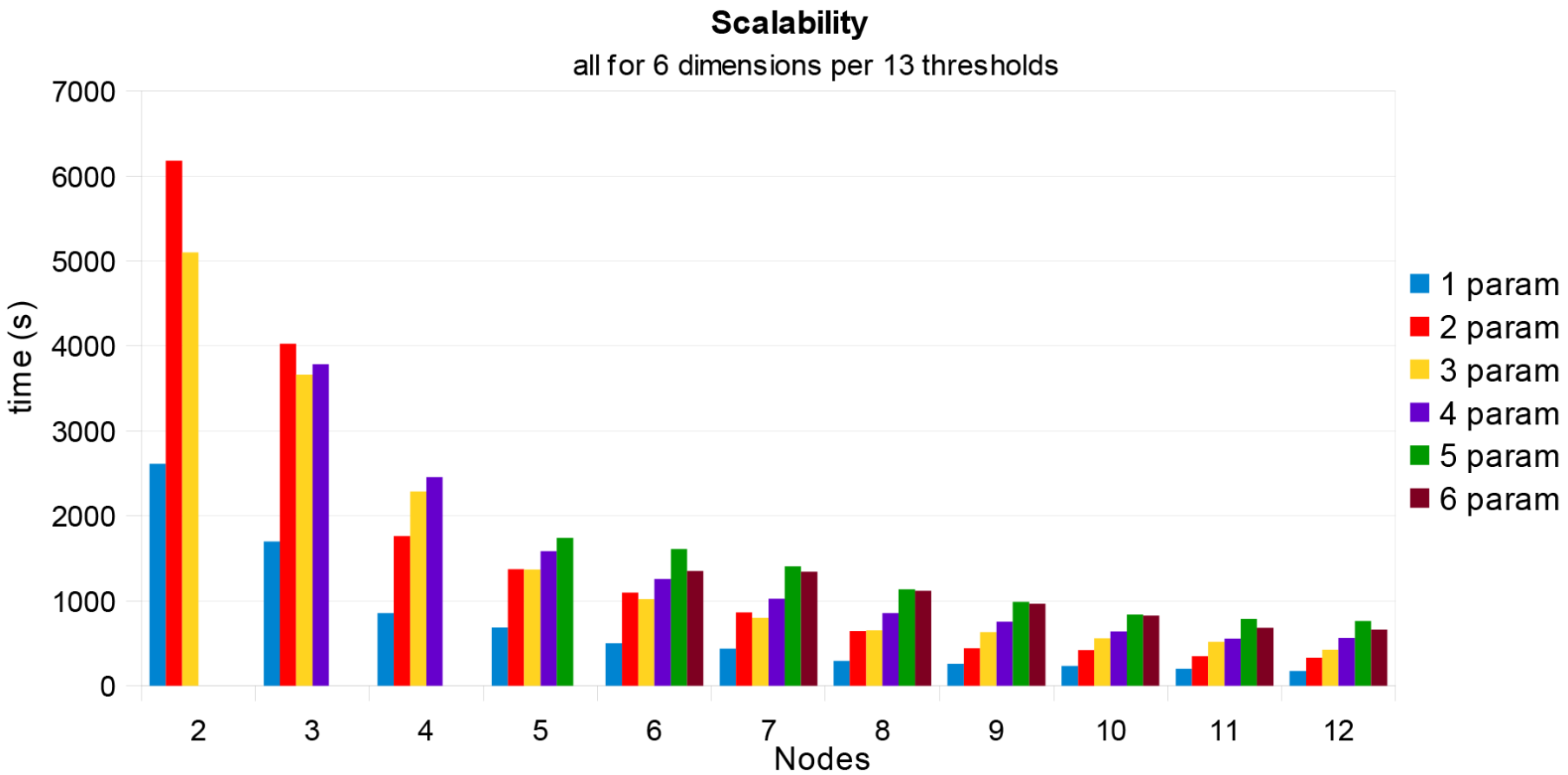


Tool Architecture



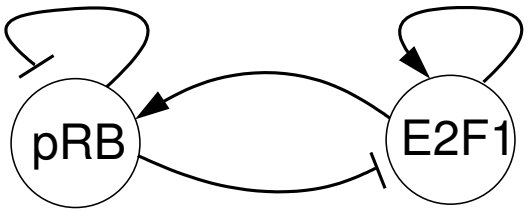
Scalability

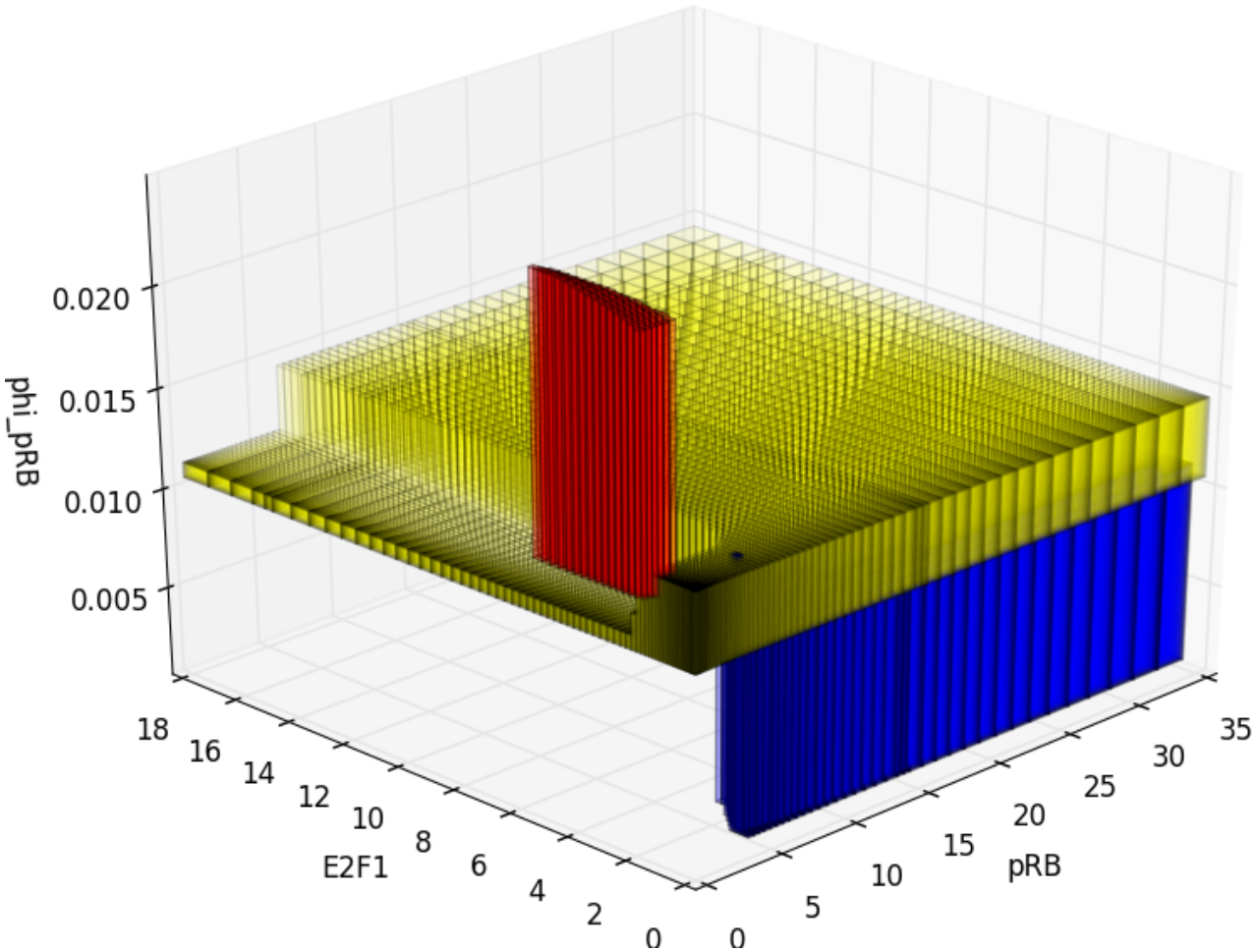
Testing reachability on enzyme-substrate reversible catalytic reaction



Case Study

Investigating bistability of a well known two gene regulatory network.





Question Time