SMT-Based Verification of Hybrid Systems

Alessandro Cimatti

FBK-irst, Trento, Italy
cimatti@fbk.eu

ATVA'13, Hanoi, Vietnam, October 2013

Plan for today

Goals of the Tutorial

- Hybrid systems: discrete and continuous dynamics
- Strong verification capabilities required
- Techniques based on Satisfiability Modulo Theories
- Techniques specific to hybrid systems

Joint work with Sergio Mover and Stefano Tonetta

Structure of the Tutorial

1. From Complex Embedded Systems to Hybrid Automata
   - Complex Embedded Systems
   - The formalism: Hybrid Automata
   - Verification of Hybrid Automata
2. Satisfiability Modulo Theories
3. SMT-based Verification of Symbolic Transition Systems
4. SMT-based verification of Hybrid Automata - Reachability
   - Global-time semantic
   - Local-time semantic
   - Shallow-synchronization
5. SMT-based verification of Hybrid Systems - Scenario Feasibility
6. SMT-based Analysis of Requirements for Hybrid Systems
7. Conclusions and future work

Outline

Complex embedded systems

- Designing complex systems
  - Automotive
  - Railways
  - Aerospace
  - Industrial production
- Sources of complexity:
  - Hundreds of functions
  - Networked control
  - Real-time constraints
  - Complex execution model with mixture of real-time and event-based triggers
  - System composed of multiple heterogeneous subsystems
- Critical Functions:
  - ABS, drive-by-wire
  - Operate switches, level crossings, lights
  - Manage on-board power production
- Conflicting objectives:
  - Avoid crashes vs move trains

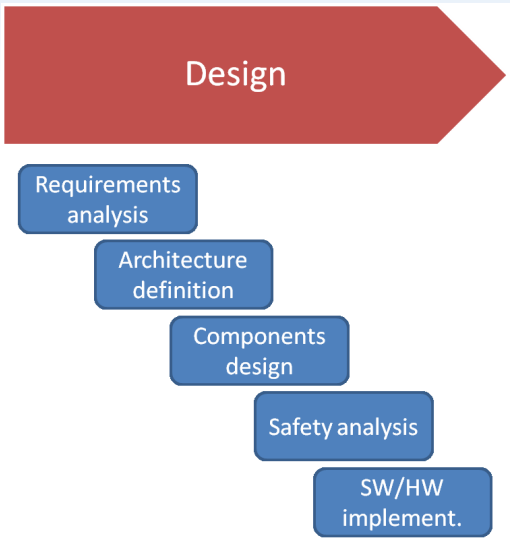
Examples of complex embedded system

- The European Train Control System
  - Supervision of movement of trains
  - Requirements on location and speed
  - Protocols between on-board train systems and track-side systems
  - Communication by radio (radio block centers) or on-track physical devices (balises).
- The framework must be able to express
  - classes of entities and their relationships;
  - integer and real attributes of the objects;
  - constraints on the admitted configurations;
  - constraints on the admitted temporal evolutions:
    - instantaneous changes of the configurations
    - temporal constraints on the movement of objects.

} Hybrid Systems

Supporting the design process

- How do we support the design?
- Requirements validation:
  - Are the requirements flawed?
- Functional correctness
  - Does the system satisfy the requirements?
- Safety assessment
  - Is the system able to deal with faults?



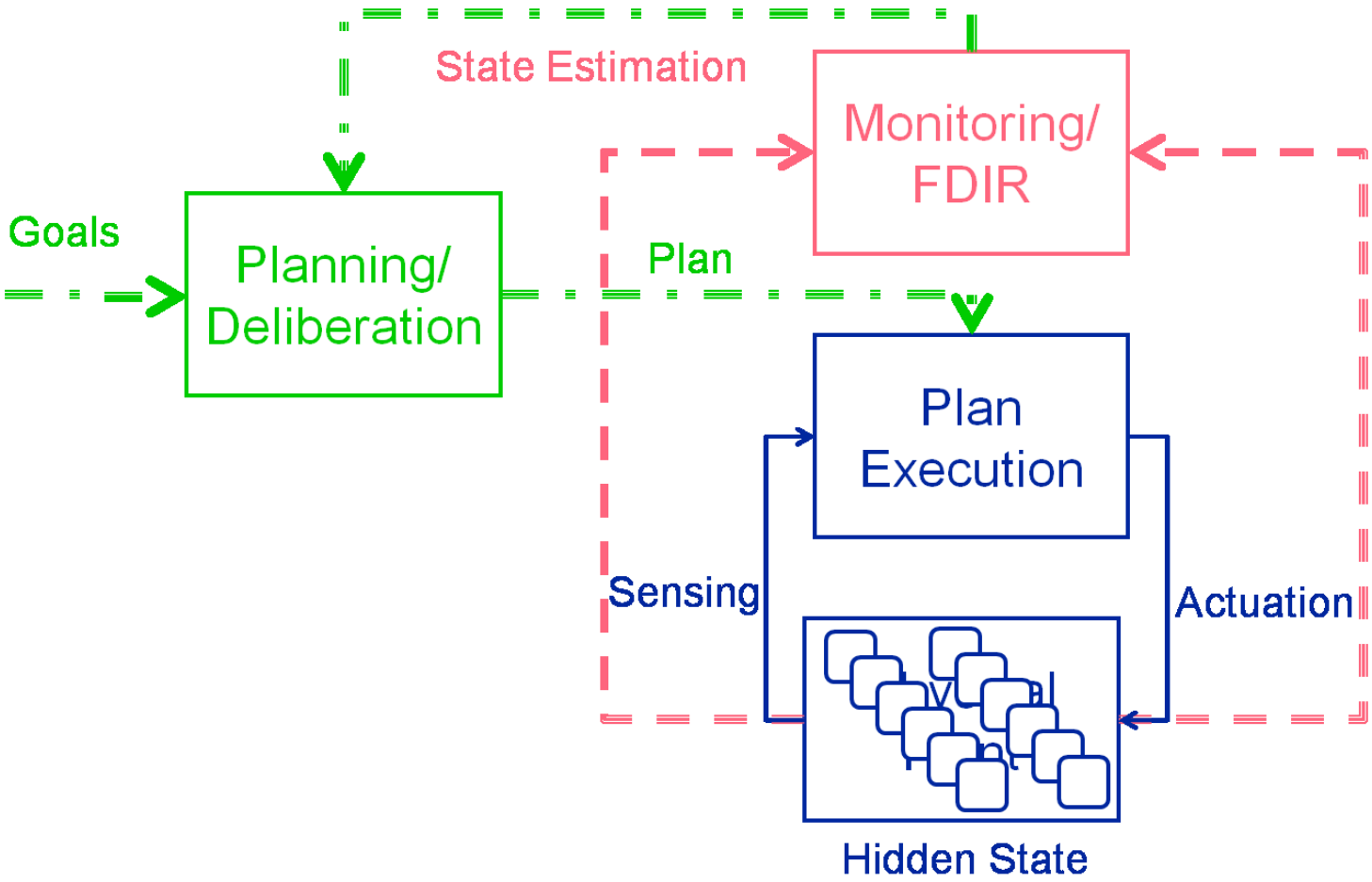
Another example application

- a pipe laying vessel
- effectively a floating factory
- retrieval of pipes from holds, routing through prefabrication stations, welding
- variable time/speed of welding operations
- nondeterminism, faults, operations in degraded modes

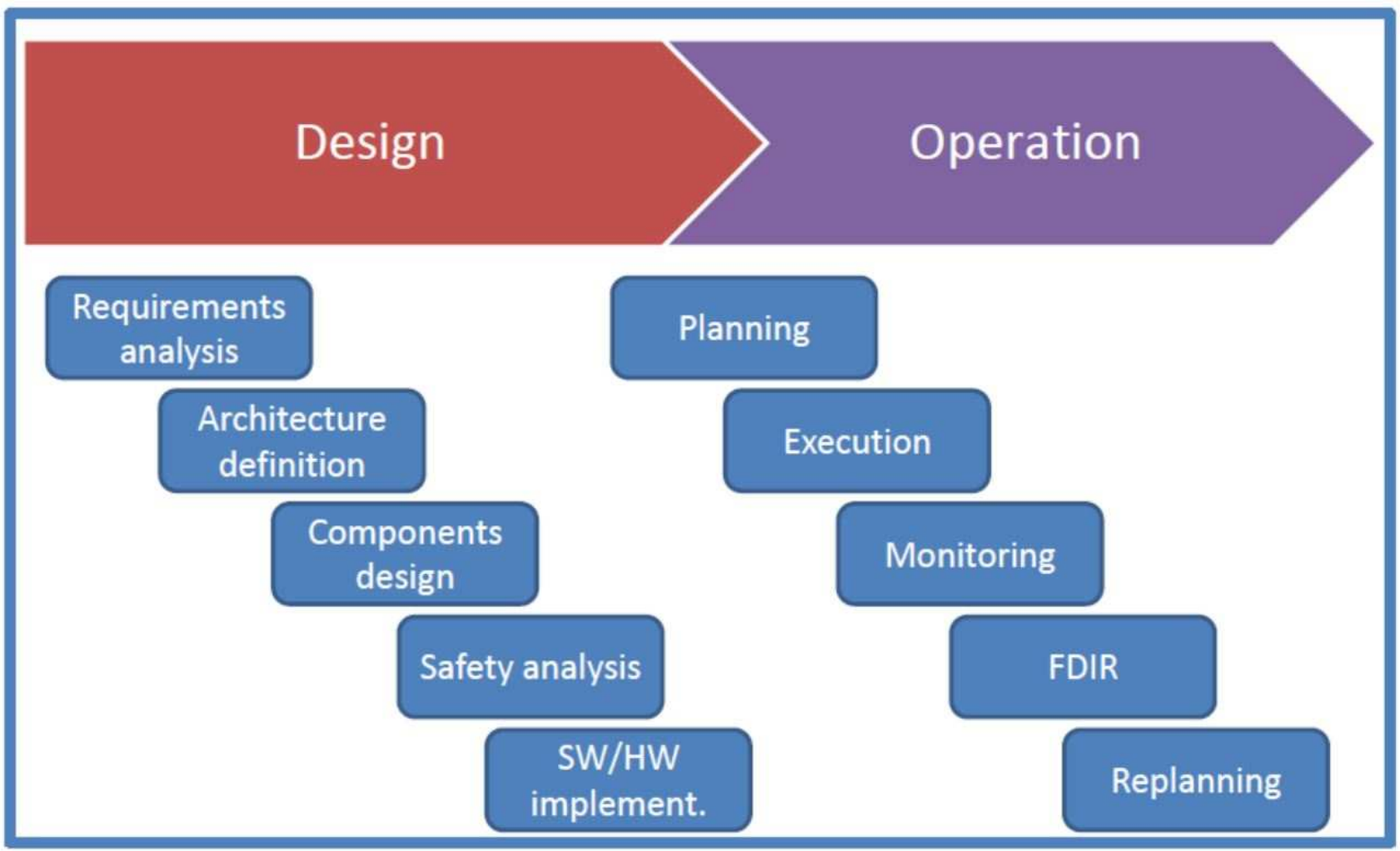
From design to operation

- Planning
  - plan how to achieve desired firing sequence
  - retrieve pipes from holds, pre-weld, send to firing line, final weld
- Execution Monitoring
  - welding may fail, activities can take more time than expected
  - plant may fail
- Fault Detection, Fault Identification/Isolation
  - is there a problem? where is it?
- Fault Recovery
  - put off-line problematic equipment
- Replanning
  - identify alternative course of actions, e.g. reroute pipes

Operation Phases

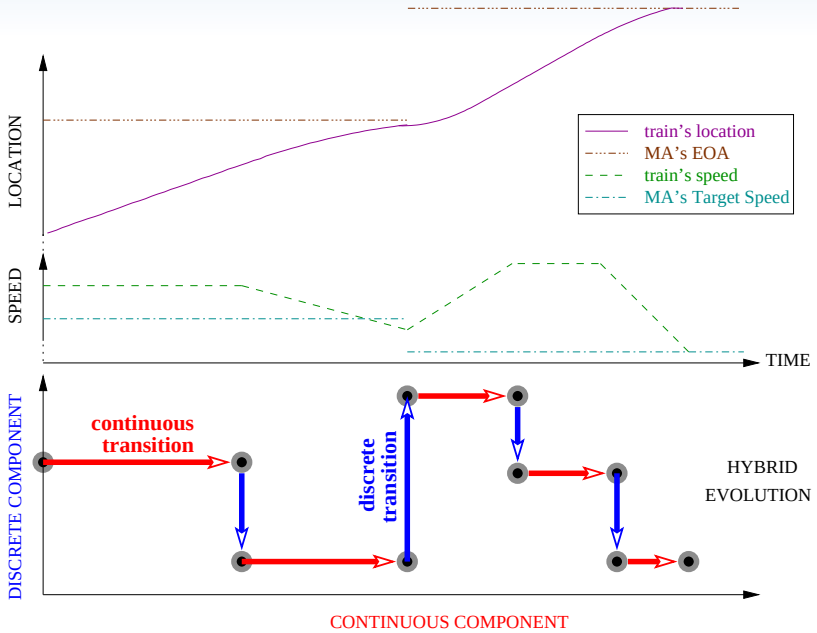


The design-to-operation continuum



Key requirement: reasoning about evolution of hybrid systems

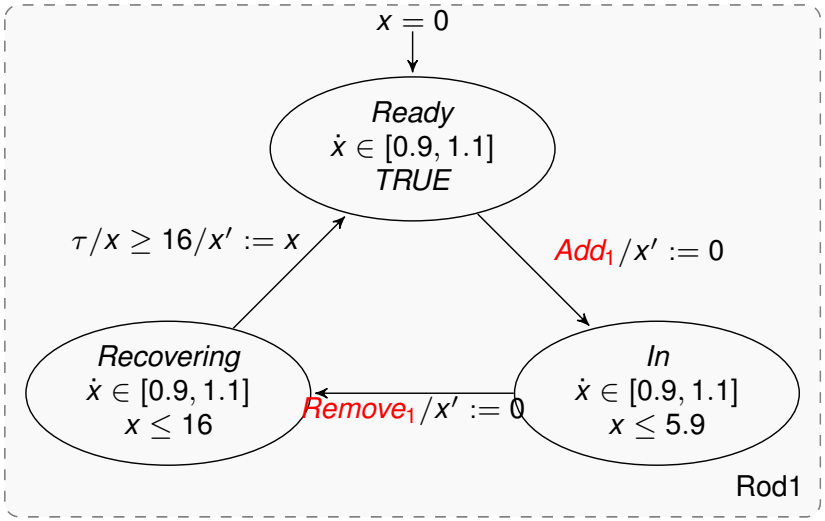
Continuous evolution plus discrete evolution

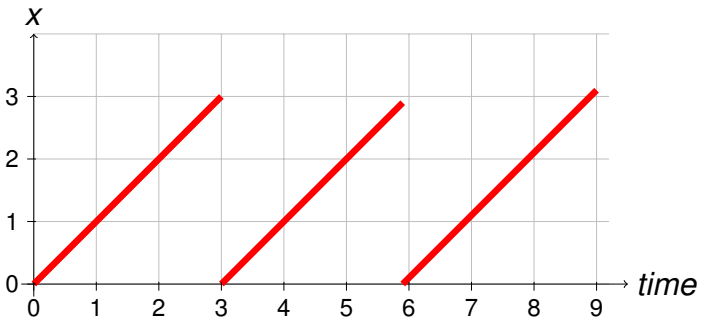


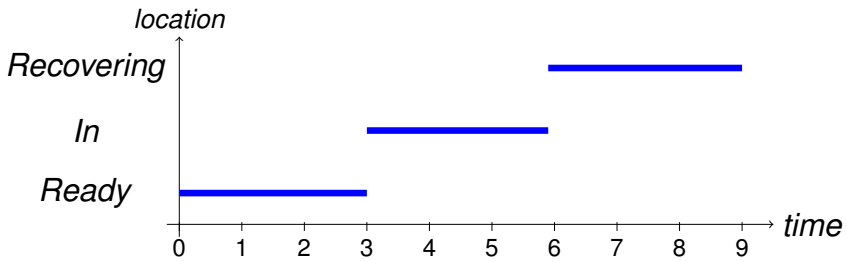
- Discrete transitions:
  - instantaneous change in (discrete) state of the system
- Continuous transitions:
  - continuous variables evolve over time according to specified laws
  - discrete state does not change

Hybrid Automata (HA)

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.
- Discrete instantaneous mode switches.
- Continuous evolution according to flow conditions.







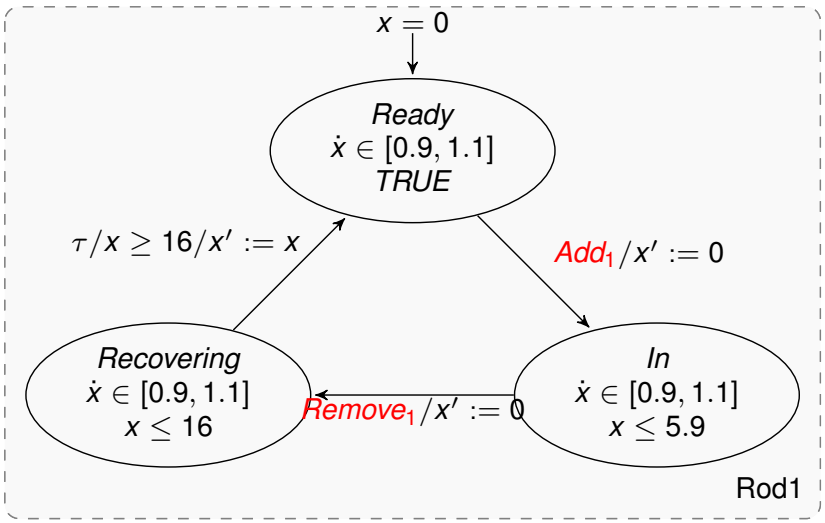
Hybrid Automaton - formal definition

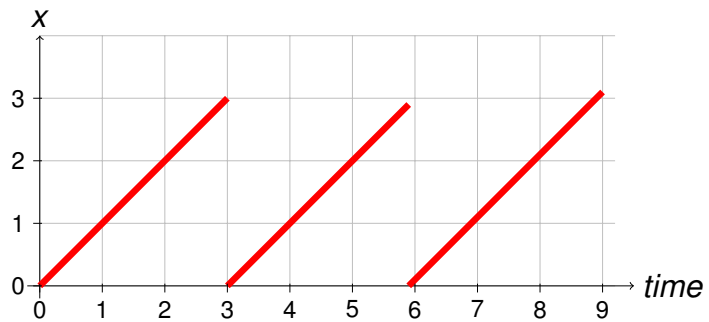
A *Hybrid Automaton* is a tuple $\langle Q, A, Q_0, R, X, \mu, \iota, \xi, \theta \rangle$ where:

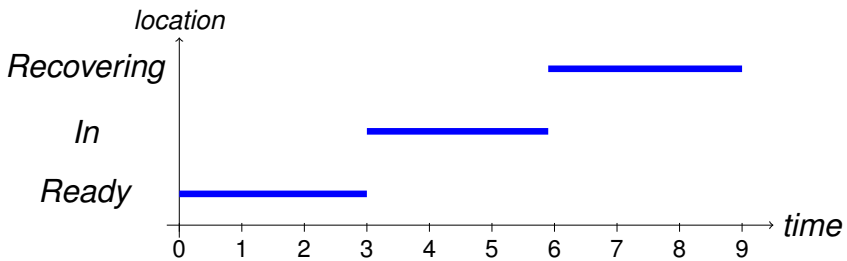
- Q is the set of locations,
- A is the set of events,
- $Q_0 \subseteq Q$ is the set of initial locations,
- $R \subseteq Q \times A \times Q$ is the set of discrete transitions,
- X is the set of continuous variables,
- $\mu : Q \to P(X, \dot{X})$ is the flow condition,
- $\iota : Q \to P(X)$ is the initial condition,
- $\xi : Q \to P(X)$ is the invariant condition,
- $\theta : R \to P(X, X')$ is the jump condition.

Hybrid Automata: evolution

- state in $Q \times (X \to \mathbb{R})$
- discrete transition: location can change, values to continuous variables may change according to jump condition
- time elapse: location does not change, continuous variables evolve according to flow conditions, while satisfying invariants





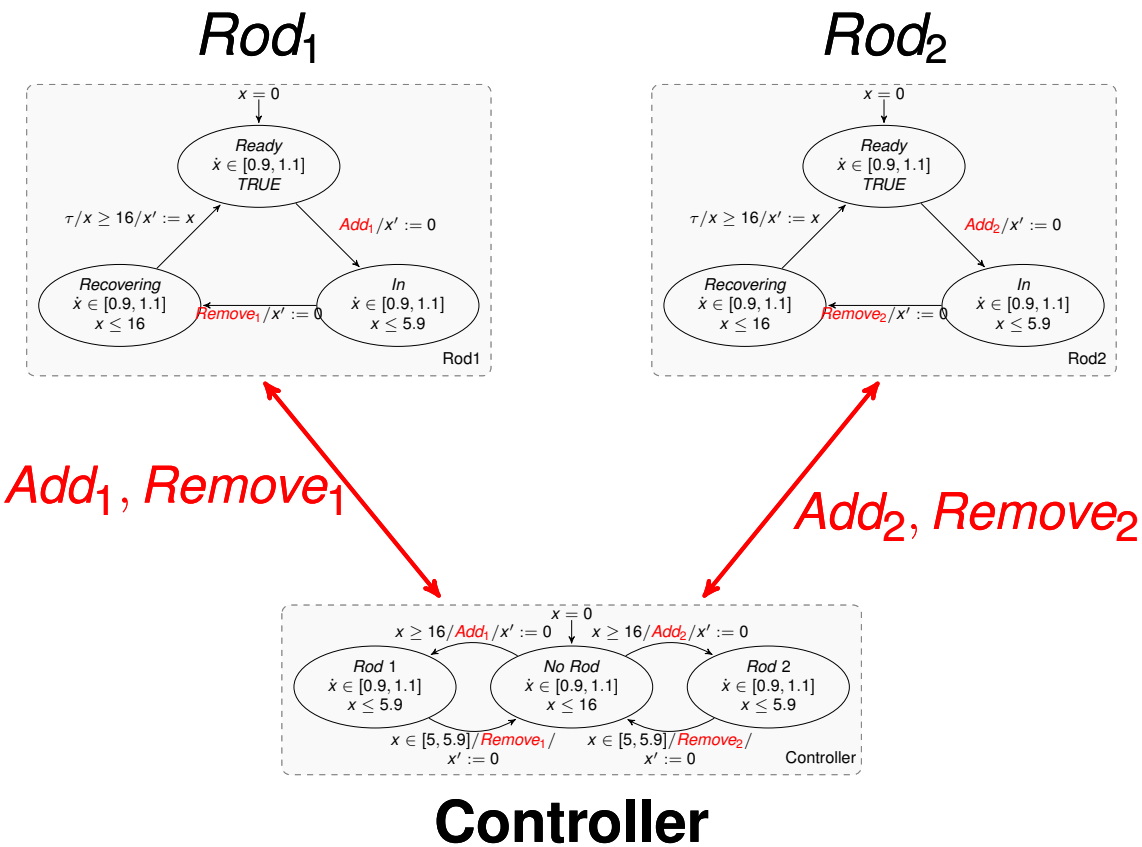


Hybrid Automata: some subclasses

- *Timed Automata*: For all $x \in X$ $\dot{x} = 1$, each $P(X)$ are of the form $x \bowtie y$ or $x \bowtie a$, where $x, y \in X$ and $a \in \mathbb{R}$.
- *Stopwatch Automata*: For all $x \in X$ $\dot{x} = 1$ or $\dot{x} = 0$.
- *Rectangular Automata*: LHA with the restriction that for all $x \in X$, $a \leq \dot{x} \leq b$, $a, b \in \mathbb{R}$.
- *Linear Hybrid Automata* (LHA):
  - $P(X, \dot{X}), P(X), P(X')$ are a Boolean combinations of linear inequalities.
  - The flow condition contains only variables in $\dot{X}$.
  - The invariants are convex.

Hybrid Automata Network (HAN)

- Automata move asynchronously on local events (τ).
- Automata synchronize on shared events.
- The time elapse is a synchronization event (global-time semantic [Henzinger 96]).



HAN - formal definition

$\mathcal{N} = H_1 || \ldots || H_n$ is a *network* of Hybrid automata:

- For all $1 \leq i \leq n$, $H_i = \langle Q_i, A_i, Q_{0i}, R_i, X_i, \mu_i, \iota_i, \xi_i, \theta_i \rangle$.
- $H_1 || \ldots || H_n$ denotes the *parallel composition* of $H_1, \ldots, H_n$.
- In our settings, for all $1 \leq i < j \leq n$ $X_i \cap X_j = \emptyset$ (i.e. the set of continuous variables of the hybrid automata are disjoint)

Notation:

- *Shared events of* H_i: $A_i \cap \bigcup_{j \neq i} A_j$.
- *Local events of* H_i: $\tau_i = A_i \setminus \bigcup_{j \neq i} A_j$.

HAN semantics as Labeled Transition Systems (LTS)

The semantics of a HAN can be provided in terms of a network of Labeled Transition Systems (LTS).

- LTS *network*: parallel composition of two or more LTSs $S_1 || \ldots || S_n$.
- An LTS S is a tuple $\langle Q, A, Q_0, R \rangle$ where:
  - Q is the set of states,
  - A is the set of actions/events (also called alphabet),
  - $Q_0 \subseteq Q$ is the set of initial states,
  - $R \subseteq Q \times A \times Q$ is the set of labeled transitions.

Remark: S may have an infinite number of states, transitions and events.

HAN Semantics as LTS: General idea

The semantic of the HA H is defined by the LTS S.

- A state in S is a location and an assignment to the continuous variables of H.
- A label in S is an event of H (discrete transition) or a duration (continuous evolution).
- We have a transition in S:
  - For every discrete transitions of H (“obvious”).
  - For every possible value of the duration of the continuous evolution (i.e. the continuos evolution is mapped into an infinite number of transitions).
  - Invariants must hold throughout the timed transition. This results in a universally quantified condition, and may affects the existence of a transition for a specific value.

The semantics of a HAN is obtained composing a network of LTSs.

HAN Semantics as LTS

The semantic of $\mathcal{H} = H_1 || \ldots || H_n$ is defined by $\mathcal{N}_{\text{GLTIME}}(\mathcal{H}) = S_1 || \ldots || S_n$ where:

- states
  $Q'_i = \{\langle q, \overline{x} \rangle \mid q \in Q_i, \overline{x} \in \mathbb{R}^{|X_i|}\}$,
- labels
  $A'_i = A_i \cup \{\langle \text{TIME}, \delta \rangle \mid \delta \in \mathbb{R}\}$,
- initial states
  $Q'_{0i} = \{\langle q, \overline{x} \rangle \mid q \in Q_{0i}, \overline{x} \in \iota_i(q)\}$,
- (instantaneous) transitions $R'_i =$
  $\{\langle \langle q, \overline{x} \rangle, a, \langle q', \overline{x}' \rangle \rangle \mid \langle q, a, q' \rangle \in R_i, \langle \overline{x}, \overline{x}' \rangle \in \theta_i(q, a, q'), \overline{x} \in \xi_i(q), \overline{x}' \in \xi_i(q')\}$
- (timed) transitions
  $\{\langle \langle q, \overline{x} \rangle, \langle \text{TIME}, \delta \rangle, \langle q, \overline{x}' \rangle \rangle \mid$ there exists f satisfying $\mu_i(q)$ s.t. $f(0) = \overline{x}, f(\delta) = \overline{x}', f(\epsilon) \in \xi(q), \epsilon \in [0, \delta]\}$.

HAN - some remarks

- networks of timed automata include all the difficulties of asynchronous models (typically explicit state) and real-time models (typically require symbolic reasoning of some form)
- timed transitions require that the invariants hold for all the points within the duration
  - formally, it requires dealing with universal quantification
  - often eliminated by means of convexity arguments (but see [CMT12])

Which focus on hybrid systems?

Two “classes” of hybrid systems:

- component level, detailed model of physical behaviour
  - relatively small discrete space
  - one/few components
  - complex dynamics (nontrivial differential equations)
  - gearbox, steering control algorithm, pacemaker
- system level, “coarse” model
  - large discrete space
  - many components
  - dynamics of limited complexity
  - space exploration rover, satellite, pipe laying vessel

Verification techniques

1. Reachability analysis.
2. Deductive verification.
3. Abstraction.

See also [Alu11] for a survey.

Reachability analysis

Prove that a safety property φ holds by a fixed-point computation of the reachable states.
Representative tools:

- HyTech [HHWT97]:
  - Represents the reachable state space using polyhedra.
  - Handles LHA.
- d/dt [ADM02]:
  - Handles non-linear hybrid automata.
  - Computes an overapproximation of the reachable states using Polyhedra (*flowpipe* approximation).
- Phaver and SpaceEx [Fre08, FGD+11]:
  - Different representation of the state space: polyhedra, support functions (approximating!).
  - Handles non-linear hybrid automata.
- Techniques based on DBM warping [LWM+11]

And many more!

Deductive Verification

- Barrier certificates [PJ04]:
  - Applied to the dynamical system (i.e. a single location).
  - Identifies a function ψ such that:
    - Is non-negative in the initial states.
    - Is negative in the bad states.
    - The sign of ψ is invariant for all the system trajectories.
- KeYMaera [PQ08]:
  - Differential invariants.
  - Uses quantifier elimination procedures on polynomials.
  - Even higher degree of parameterization
  - System under analysis represented in Dynamic logic
  - Deductive techniques, based on interactive theorem proving.
  - Impressive case studies, requires substantial human effort.

Abstraction techniques

- Linear Phase-Portrait Partitioning [HWT95]
  - Abstract complex dynamics splitting each location in multiple locations.
  - Each location has a simple dynamic of the form $a \leq \dot{x} \leq b$, $a, b \in \mathbb{R}$ which approximate the complex dynamic in a specific interval.
- Predicate Abstraction [ADI06, Tiw08]
  - A set of predicates defines a finite-state abstraction of the hybrid systems.
  - The finite-state system can be analyzed using finite-state techniques.
  - Hard to choose the “right” predicates to use and to compute the resulting abstraction.

Here: SMT-based verification of HA

In the rest of this talk:

- focus on systems with large state space, coarse dynamics
- fully symbolic, automated techniques
- from SAT-based to SMT-based algorithms
- dedicated techniques for hybrid systems
  - exploit network structure with local time semantics
  - analyze scenarios expressed as MSC's
  - analysis of requirements for hybrid systems

Outline

Satisfiability Modulo Theories

- Check satisfiability of first order formulae
- with respect to background theory
- Underlying technology:
  - SAT solver behaves as enumerator
  - theory (constraint) solver used to check feasibility
- Status of the research field:
  - a standardized SMT-LIB2.0 language and a library of benchmarks `http://www.smt-lib.org`
  - a yearly competition `http://www.smt-comp.org`
  - a yearly SMT workshop
  - SMT solvers: Yices, OpenSMT, Z3, CVC, iSAT, HySAT, MathSAT
  - A (yearly?) school: Boston (2011), Trento (2012) w/ SAT'12

SMT: useful theories

- Satisfiability of a first order formula...
- where the atoms are interpreted modulo a background theory
- Theories of practical interest
  - Equality and Uninterpreted Functions (EUF)
    - x = f(y), h(x) = g(y)
  - Difference constraints (DL)
    - $x - y \leq 3$
  - Linear Arithmetic
    - $3x - 5y + 7z \leq 1$
    - reals (LRA), integers (LIA)
  - Arrays (Ar)
    - read(write(A, i, v), j)
  - Bit Vectors (BV)
    - $x[15:8] :: (y[7:0] + 0d8\_3) = (z \&\& w)$
  - Their combination

SMT checking

- SMT as an extension of boolean SAT
- Some atoms have non-boolean (theory) content
  A1 : $x - y \leq 3$
  A2 : $y - z = 10$
  A3 : $x - z \geq 15$
- Theory interpretation for individual variables, constants, functions and predicates
  if $x = 0, y = 20, z = 10$
  then A1 = T, A2 = T, A3 = F
- Interpretations of atoms are constrained
- A1, A2 and A3 can not be all true at the same time

- Boolean reasoning + constraint solving
  - SAT solver for boolean reasoning
  - theory solvers to interpret numerical constraints

SMT: search space

- DPPL-style search
  - SAT solver looks for satisfying assignment to boolean abstraction of the formula, ignoring theory content
  - literals on stack identify a conjunction (set) of theory constraints
    - the same atom may be associated to a constaint or its negation, depending o the truth value it is assigned to
  - if set of constraints solvable, then return SAT
  - if set of constraints unsatisfiable, backtrack
  - if search space exhausted, return UNSAT

SMT solvers in practice

- In practice, the integration is very tight
  - SAT solver working as an enumerator
  - Theory solver follows the stack-based search
    - Inconsistent partial assignments are pruned on the fly
    - conflicts clauses learnt from theory reasoning
    - used to drive search at the boolean level
- Satisfiability Modulo Theories: a sweet spot?
  - increase expressiveness
  - retain efficiency of boolean reasoning
  - Trade off between expressiveness and reasoning
  - SAT solvers: boolean case, automated and very efficient
  - theorem provers: general FOL, limited automation

SMT: useful functionalities

Similarly to SAT solvers, SMT solvers expor

- Model construction
- Incremental interface
- Unsatisfiable core
- Proof production
- Interpolation
- AllSMT

AllSMT

$$AR(P,P') \doteq \exists XX'.(R(X,X') \wedge \bigwedge_i (P_i \leftrightarrow \psi_i(X)) \wedge \bigwedge_i (P'_i \leftrightarrow \psi_i(X')))$$

- AllSMT - a particular form of existential quantification
  - Enumerate all satisfying assignments to P_i by generalizing AllSAT to AllSMT [LNO06]
  - Extend BDD-based existential quantification to deal with theory constraints [CCF+07]
    - Build a boolean abstraction of the formula to quantify
    - Interpret each boolean variable as a theory constraint
    - Drive SMT solver while traversing BDD (NOT a theory solver)
  - Structure aware existential quantification [CDJR09]
    Exploit the available problem structure
    - At high level: structure of system being abstracted, modules scope of variables, nature of transitions
    - At low level: structure of quantified formula, reduce scope of quantification

The MathSAT project

- MathSAT is an SMT solver developed in Trento since 2001
- Joint project of Fondazione Bruno Kessler (FBK) and the University of Trento
- `http://mathsat.fbk.eu/`
- Latest available version: MathSAT5
- Current team:
  - Alessandro Cimatti, Alberto Griggio, Bas Schaafsma, Roberto Sebastiani
- Past contributors:
  - Gilles Audemard, Piergiorgio Bertoli, Marco Bozzano, Roberto Bruttomesso, Anders Franzén, Tommi Junttila, Veselin Kirov, Artur Kornilowicz, Jeremy Ridgeway, Peter van Rossum, Alessandro Santuari, Stephan Schulz, Cristian Stenico

MathSAT: features

Supported interaction modes:

- Languages: SMT-LIB 1 and SMT-LIB 2
- In-memory API: C++, Python

Supported theories:

- $\mathcal{EUF}$, $\mathcal{BV}$, $\mathcal{RDL}$, $\mathcal{IDL}$, $\mathcal{LRA}$, $\mathcal{LIA}$, memories ($\mathcal{AR}$)
- Their combination
  - via Delayed Theory Combination
  - via Ackermanization Reduction

Functionalities:

- Incremental Solving
- Model extraction
- AllSMT
- Unsatisfiable core extraction
- Interpolation
- Costs

MathSAT: some highlights

- The “lazy approach” to SMT [ABC+02]
  - SAT solver as model enumerator
  - tight integration between SAT solver and theory solver
- Layering [BBC+05]
  - cheap solvers first
- Delayed Theory Combination [BCF+09]
  - use SAT search to deal with interface equalities
  - superseded by model-based combination
- Unsat core extraction [CGS11]
  - reduction to boolean unsatisfiable core extraction
  - based on reuse of theory lemmas computed during search
- Interpolation [CGS08, CGS09, CGS10]
  - avoid “proof theoretic” reasoning
  - based on information produced by theory solvers
- Bit-vectors [BCF+07, FCN+10]
  - experiments with various approaches
  - rewriting, lazy bit-blasting, underapproximation

MathSAT for Microcode Verification

- Result of long-standing collaboration with Intel Haifa
  - BoWLing (2003-2006)
  - Wolfling GRC CADTS Verification 2009-TJ-1880
  - WOLF (2013-2015)
- Microcode
  - expand complex ISA instructions to native micro-instructions
  - similar to low level assembly, highly optimized
- Perceived as critical problem in practice
  - a flow for the verification of microcode
  - cycle-accurate equivalence checking based on path enumerations
- Bit-precise reasoning required
  - based on boolean SAT solving
  - solving VC's requires significant portion of overall time
- Experiment with word-level reasoning
  - use MathSAT instead of internal SAT solver
  - black-box replacement: no idea on high level algorithm
  - a sequence of verification problems
  - not a nice sequence of "path extension"
  - the correlation between subsequent problems is hidden
- MathSAT now shipped with design environment for microcode
  - More details in award-winning FMCAD'10 paper [FCN+10]

Outline

Symbolic Transition Systems

LTS with logic-based, symbolic representation.
States as assignments to variables ranging over real, integers, bit vectors, arrays, ...
Transitions as pairs of states.
Symbolic representation: use formulae to describe sets of states and transitions

- Vectors of state variables: current state X, next state X'
- Initial condition $I(X)$
- Transition relation $R(X, X')$
- Bad states $B(X)$

Key difference wrt finite state model checking

- X, X' do not range only over boolean variables
- I, R, B are SMT formulae

From SAT-based to SMT-based verification

Same representation, extend algoriths from SAT to SMT:

- Bounded model checking
- Induction
- Interpolation-based verification
- Abstraction/refinement
- IC3

In many cases, no longer guaranteed to converge.
Useful SMT functionalities:

- Incrementality
- Model extraction
- Unsat core extraction
- Interpolation
- Quantifier elimination / AllSMT

SMT-based bounded model checking

- State variables replicated k times
  $X_0, X_1, \ldots, X_{k-1}, X_k$
- Look for bugs of increasing length
  $I(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
- bug if satisfiable
- increase k until ...

SMT-based k-induction

- Prove absence of bugs by induction

  $I(X_0) \wedge B(X_0)$

  $\neg B(X_0) \wedge R(X_0, X_1) \wedge B(X_1)$

  $\ldots$

  $I(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge R(X_{k-1}, X_k) \wedge B(X_k)$

  $\neg B(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge \neg B(X_{k-1}) \wedge R(X_{k-1}, X_k) \wedge B(X_k)$

- Proved correct if unsatisfiable (and no bugs until k)
- Invariant strengthening, simple path condition, ...

SMT-based interpolation

- An interpolant for an unsatisfiable formula

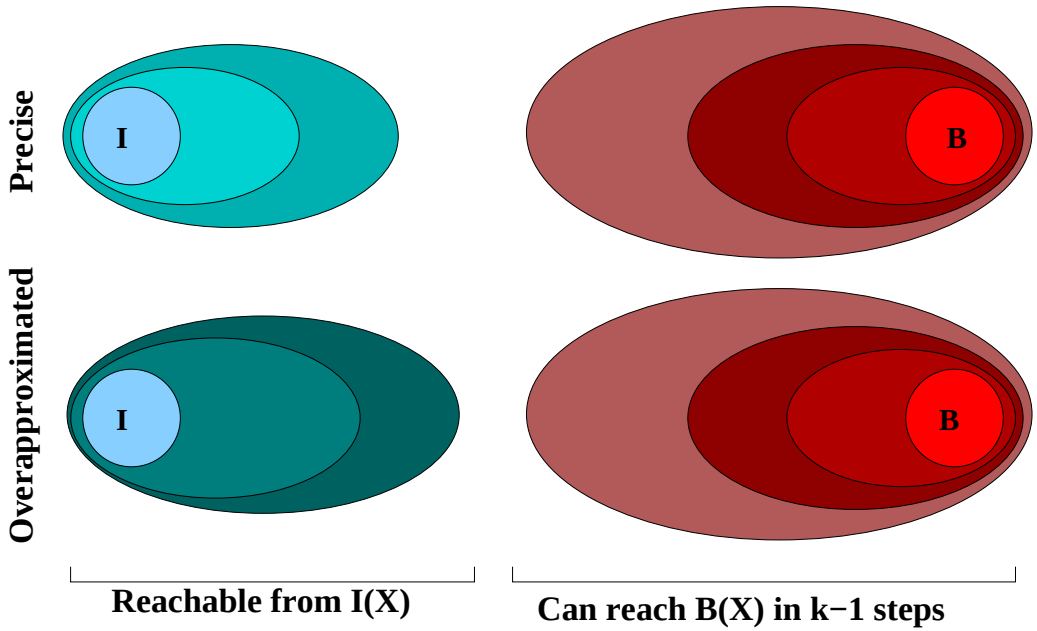
$$\Phi_1(X, Y) \wedge \Phi_2(Y, Z)$$

is a formula $Itp(Y)$ such that:

- $\Phi_1(X, Y) \rightarrow Itp(Y)$
- $Itp(Y) \wedge \Phi_2(Y, Z)$ is unsatisfiable

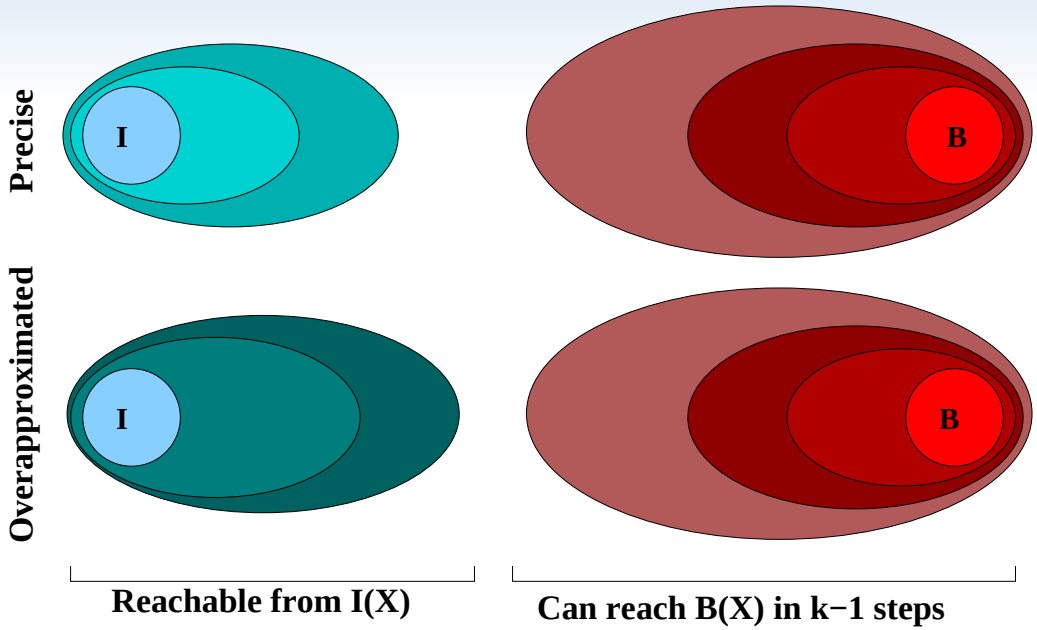
SMT-based interpolation

$$\overbrace{I(X_0) \wedge R(X_0, X_1)}^{\Phi_1(X_0,X_1)} \underbrace{\wedge}_{Itp(X_1)} \overbrace{R(X_1, X_2) \ldots \wedge R(X_{k-1}, X_k) \wedge B(X_k)}^{\Phi_2(X_1,\ldots,X_k)}$$



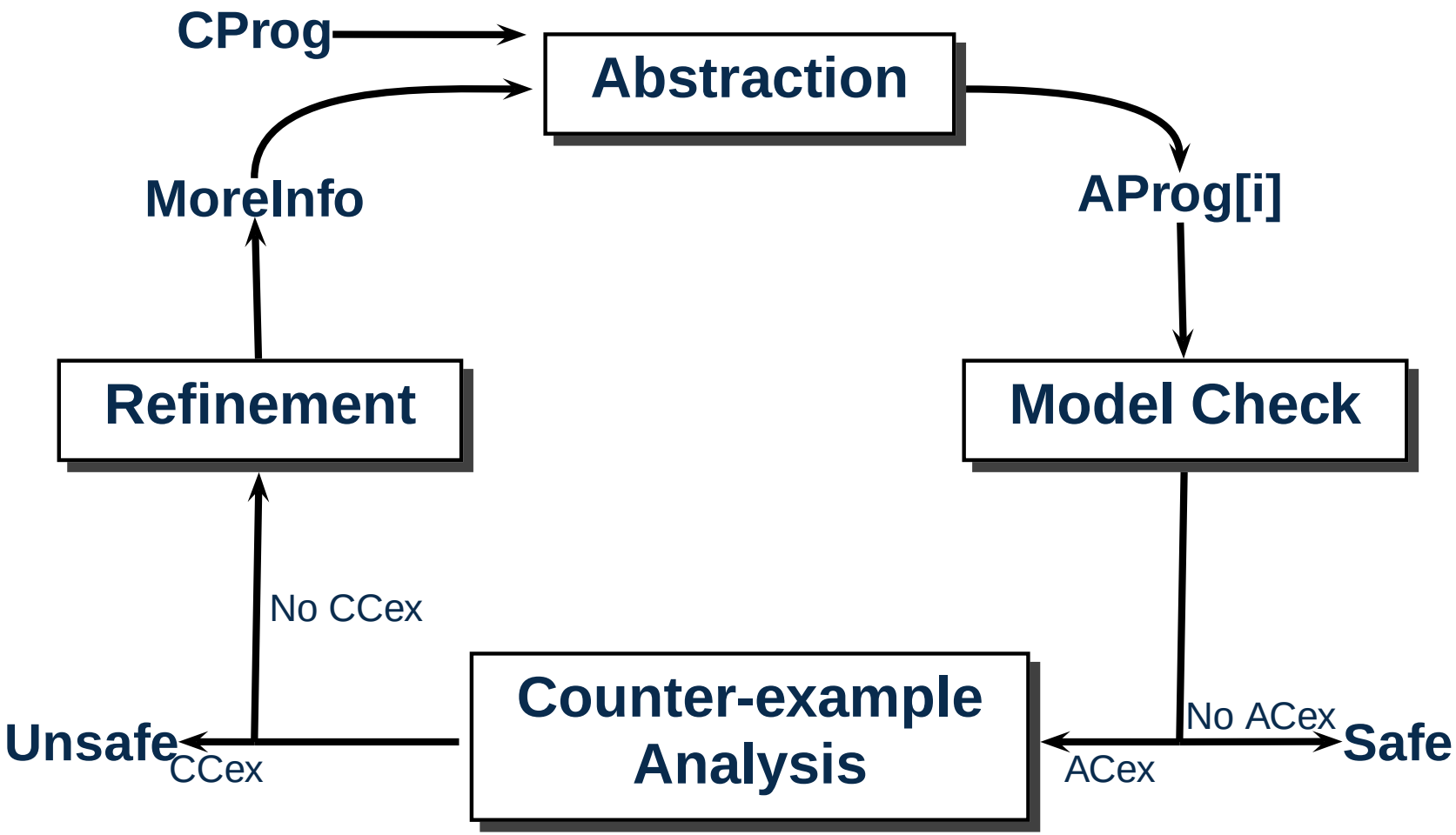
$$Itp(X_1) = Itp(R, I(X_0), k)$$

SMT-based interpolation

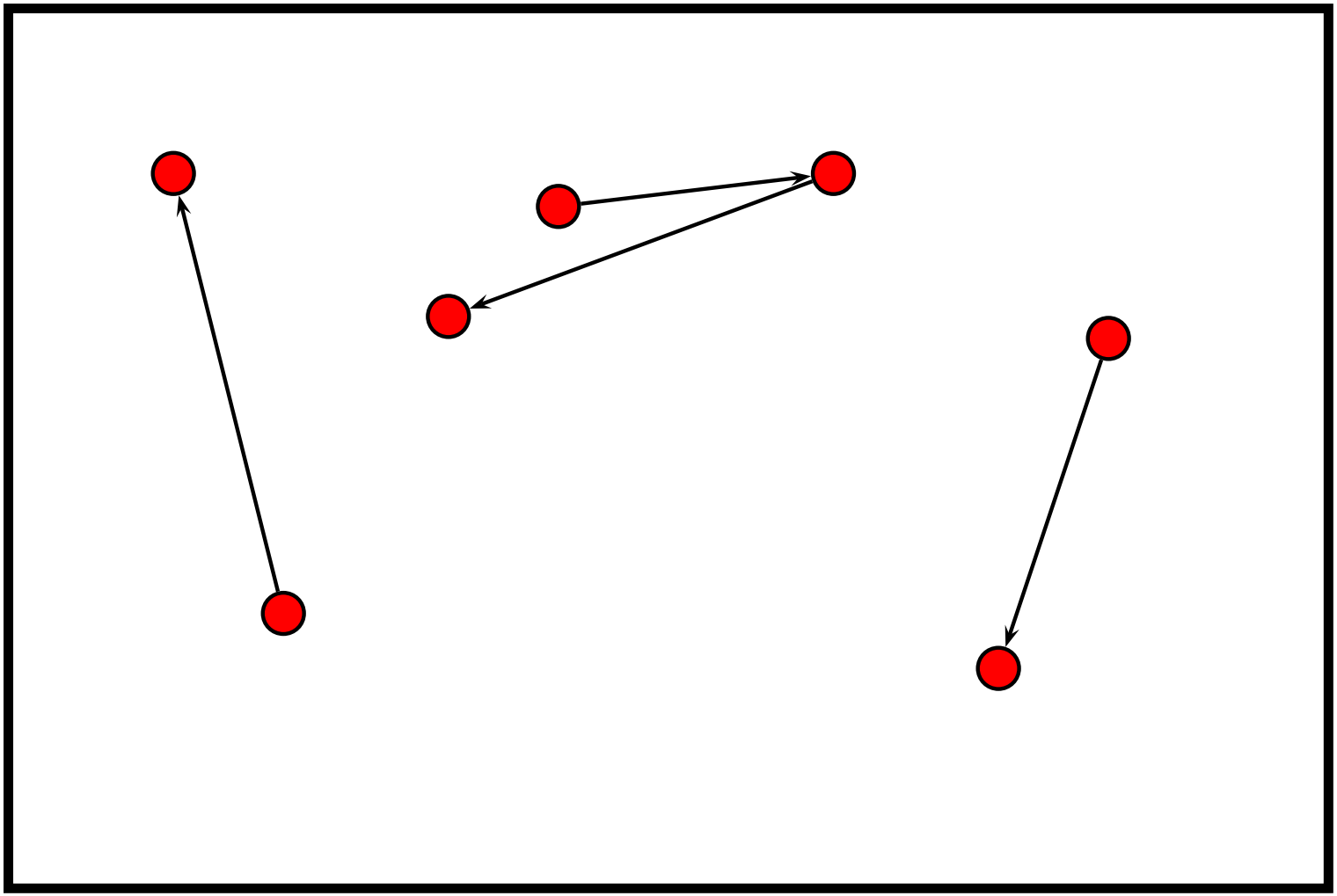


- Precise reachability
  - $\mathcal{R}_0 = I$
  - $\mathcal{R}_i = Img(R, \mathcal{R}_{i-1}) \cup \mathcal{R}_{i-1}$
- Interpolation based reachability
  - $Itp_0 = I(X_1)$
  - $Itp_i = Itp(R, Itp_{i-1}, k) \cup Itp_{i-1}$

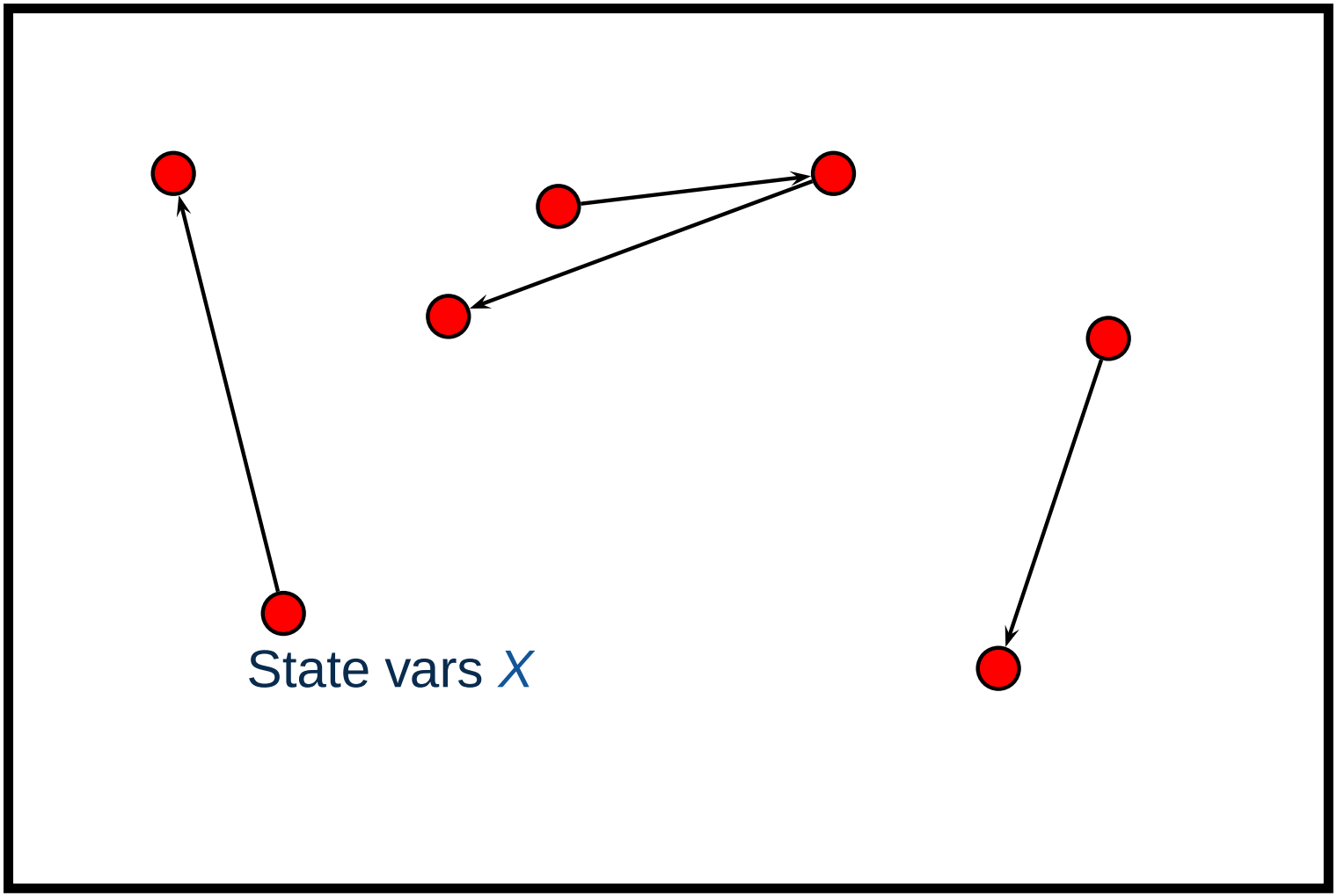
CEGAR loop


CProg
Abstraction
MoreInfo
AProg[i]
Refinement
Model Check
No CCex
Counter-example Analysis
Unsafe
CCex
No ACex
ACex
Safe

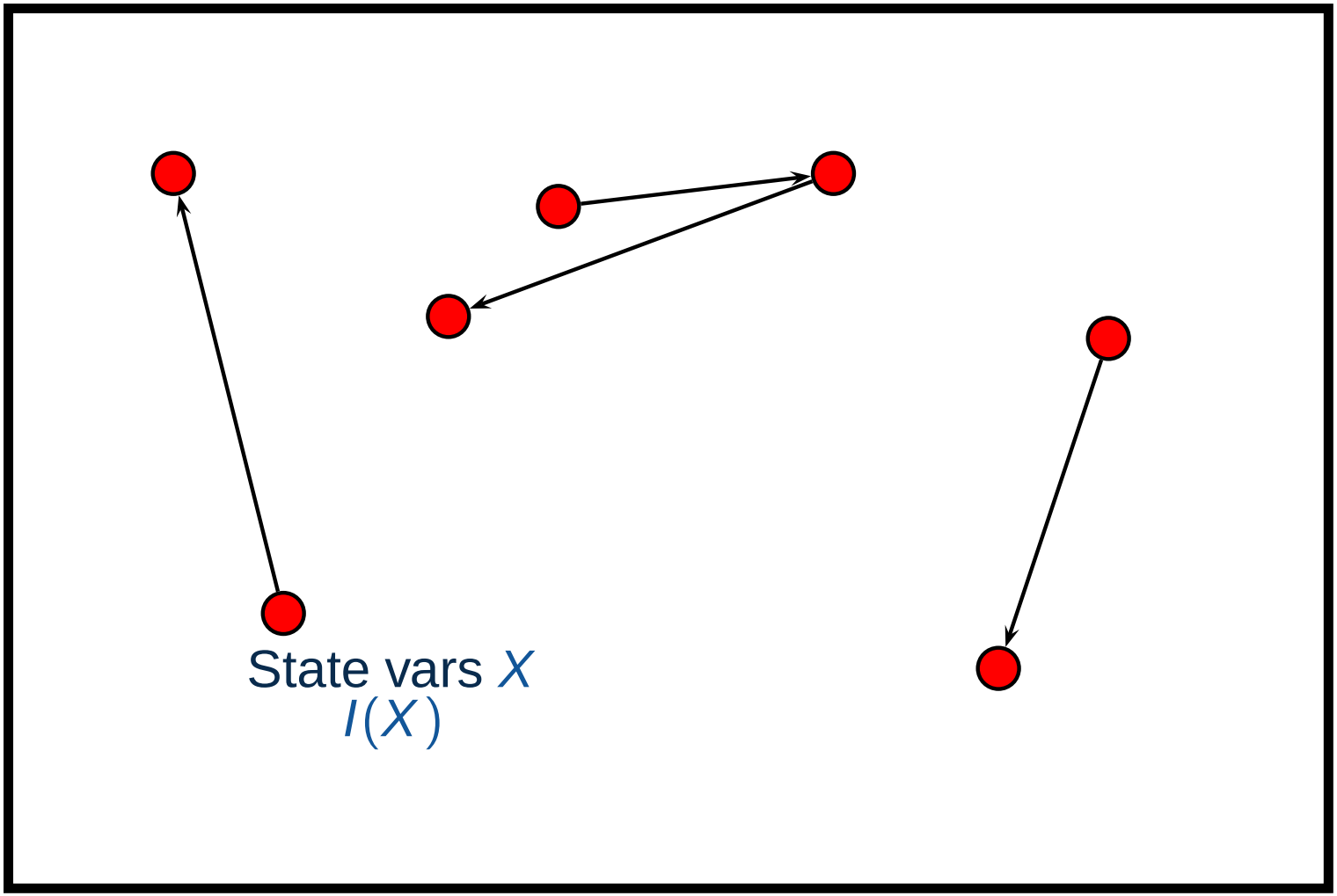
CEGAR based on Predicate Abstraction



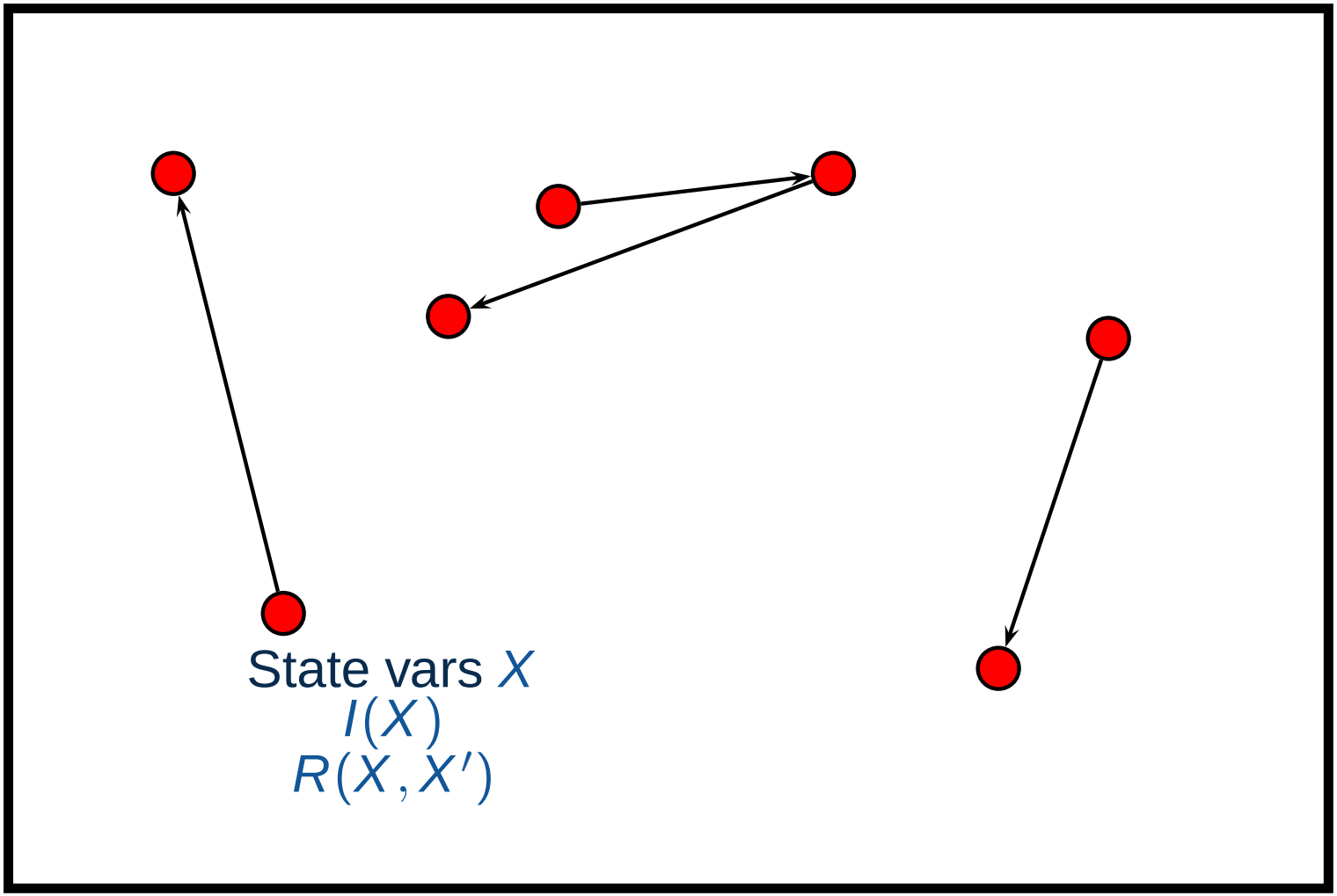
CEGAR based on Predicate Abstraction



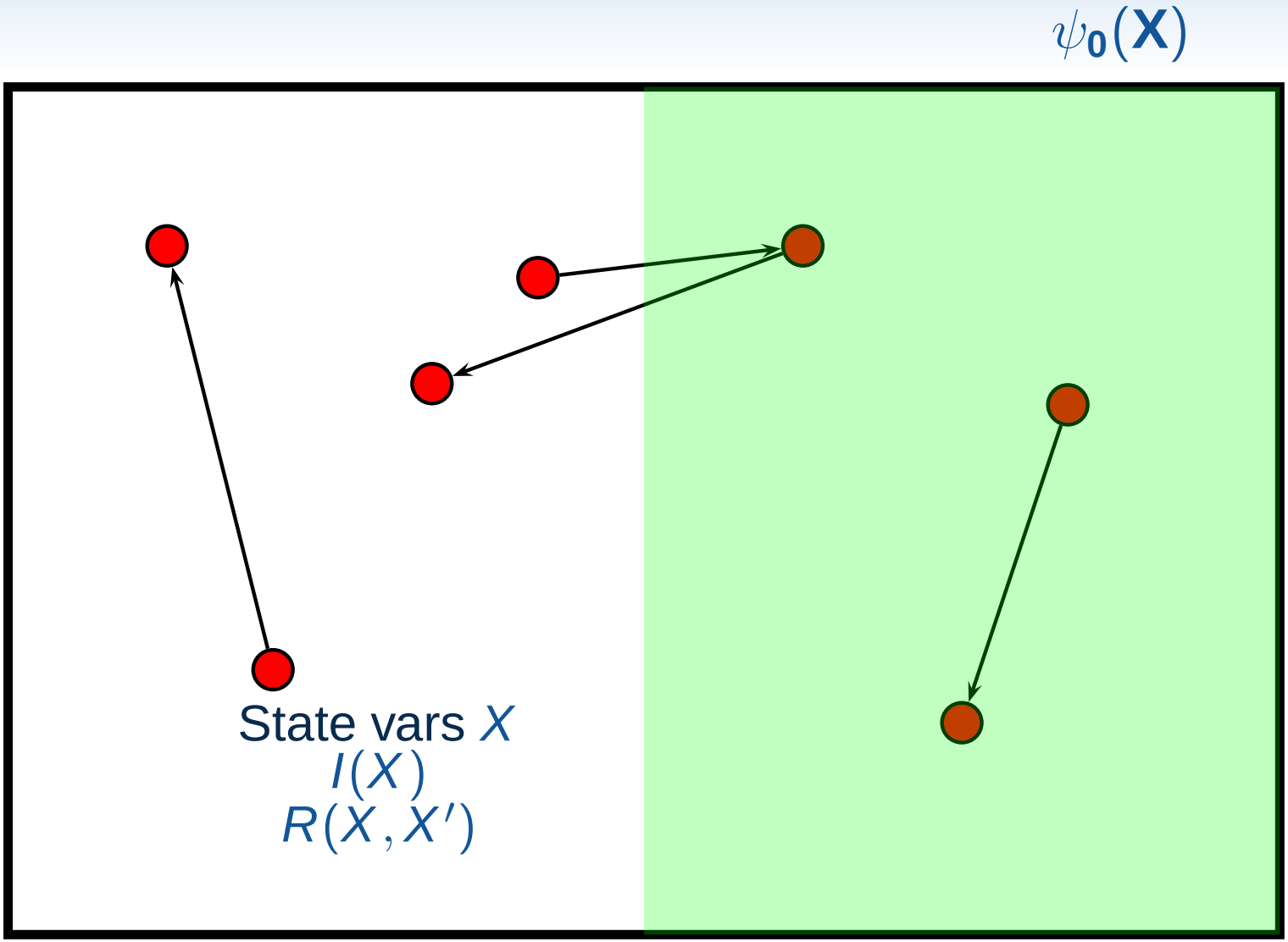
CEGAR based on Predicate Abstraction



CEGAR based on Predicate Abstraction

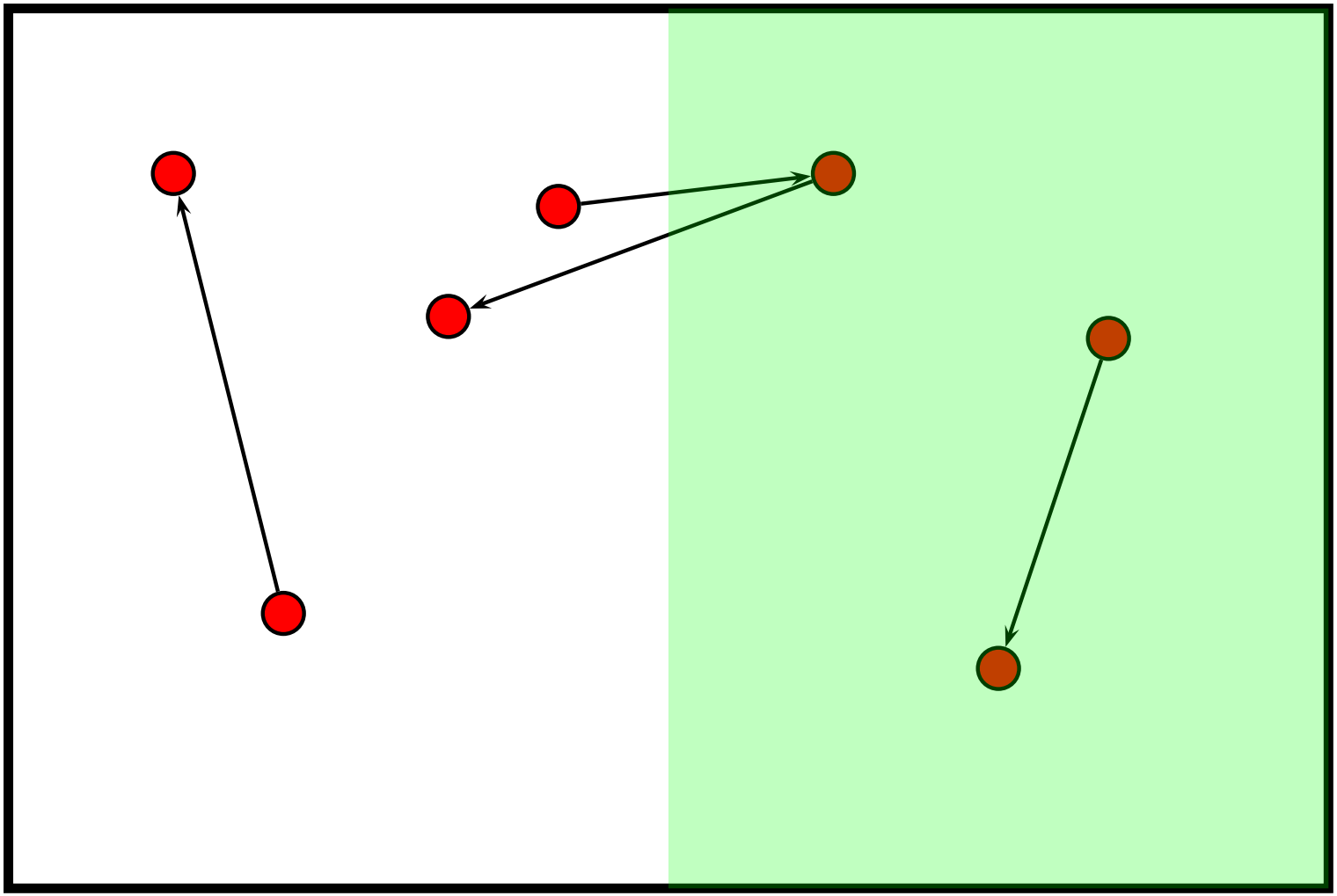


CEGAR based on Predicate Abstraction



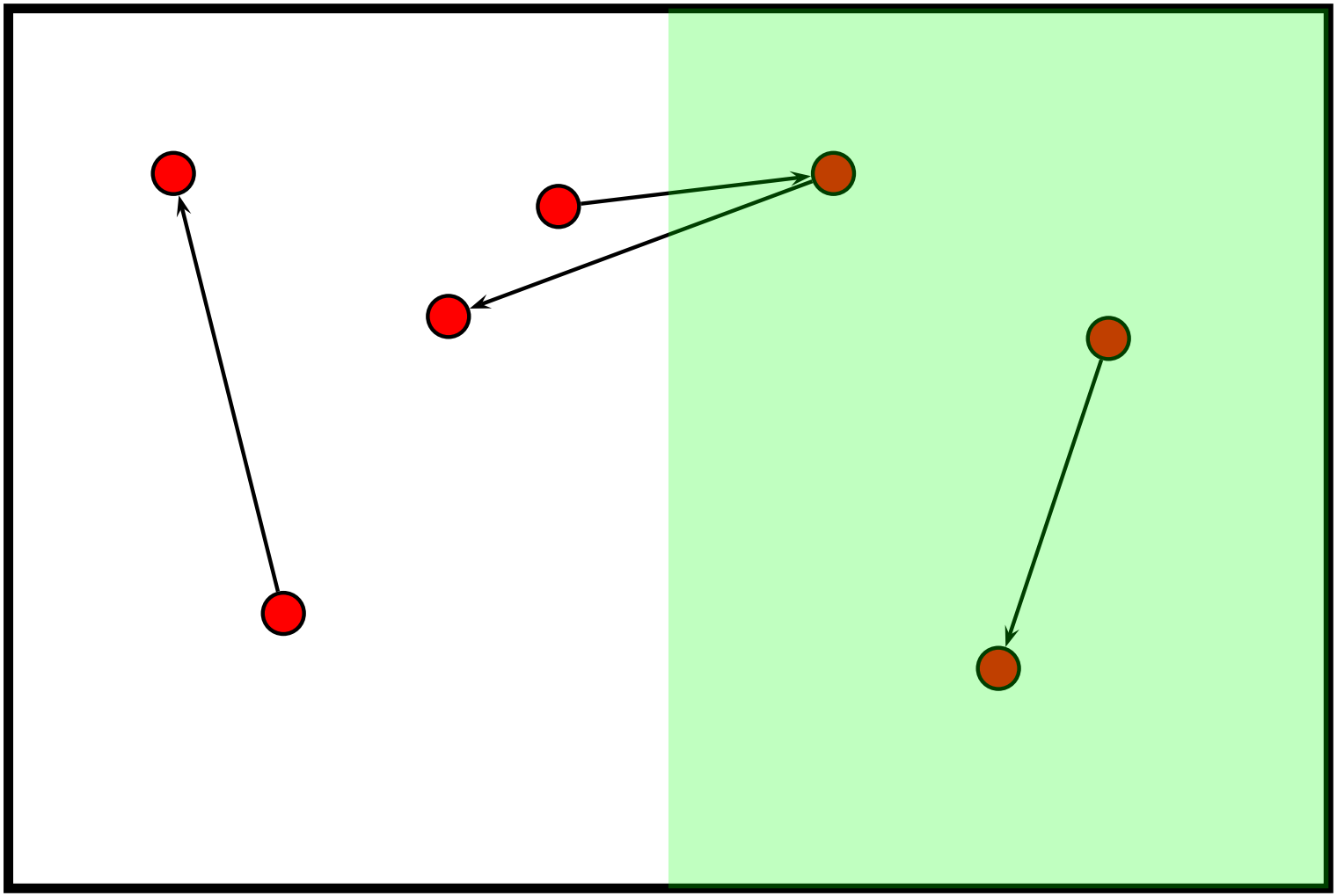
CEGAR based on Predicate Abstraction

$\psi_0(\mathbf{X})$

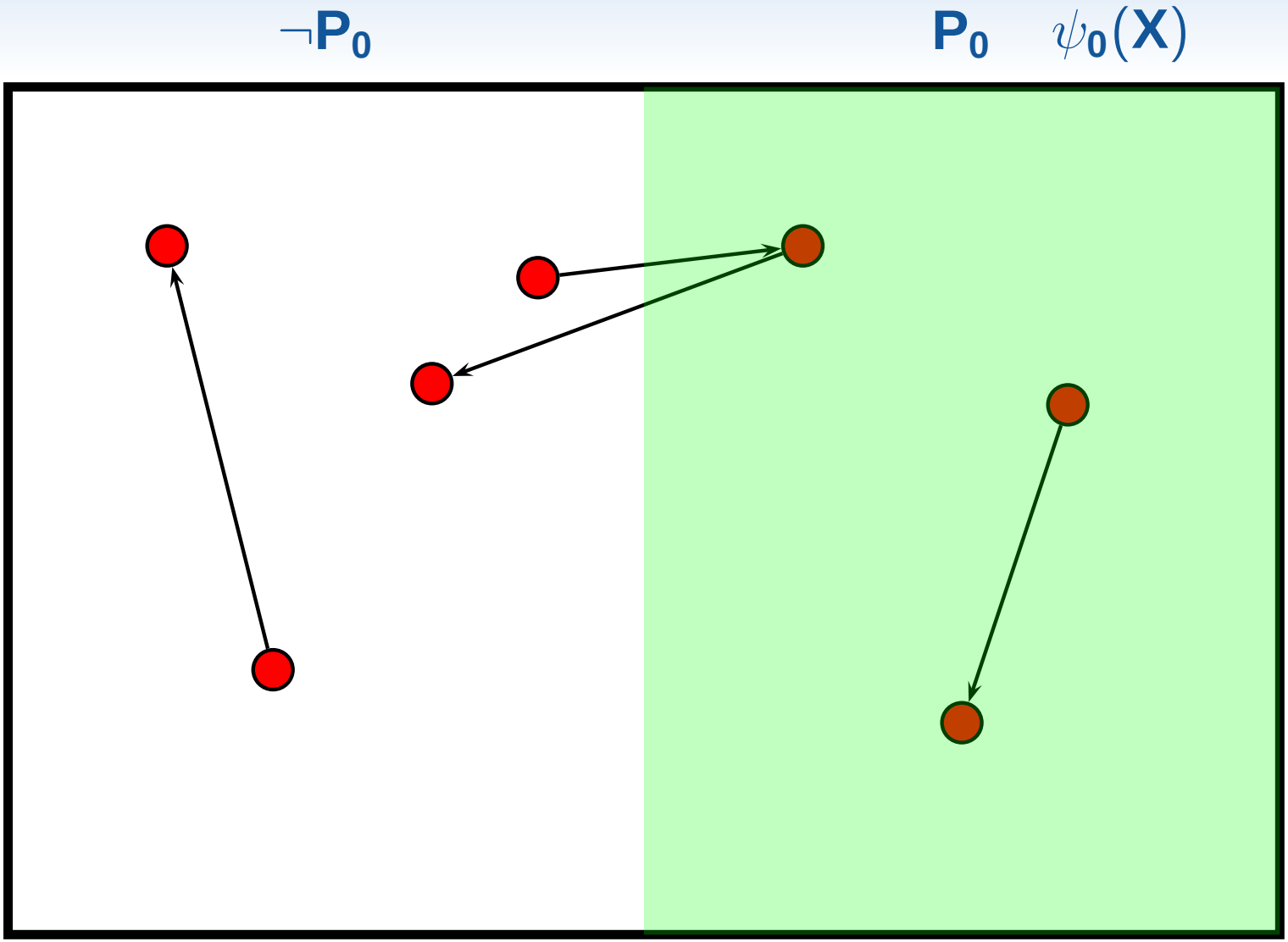


CEGAR based on Predicate Abstraction

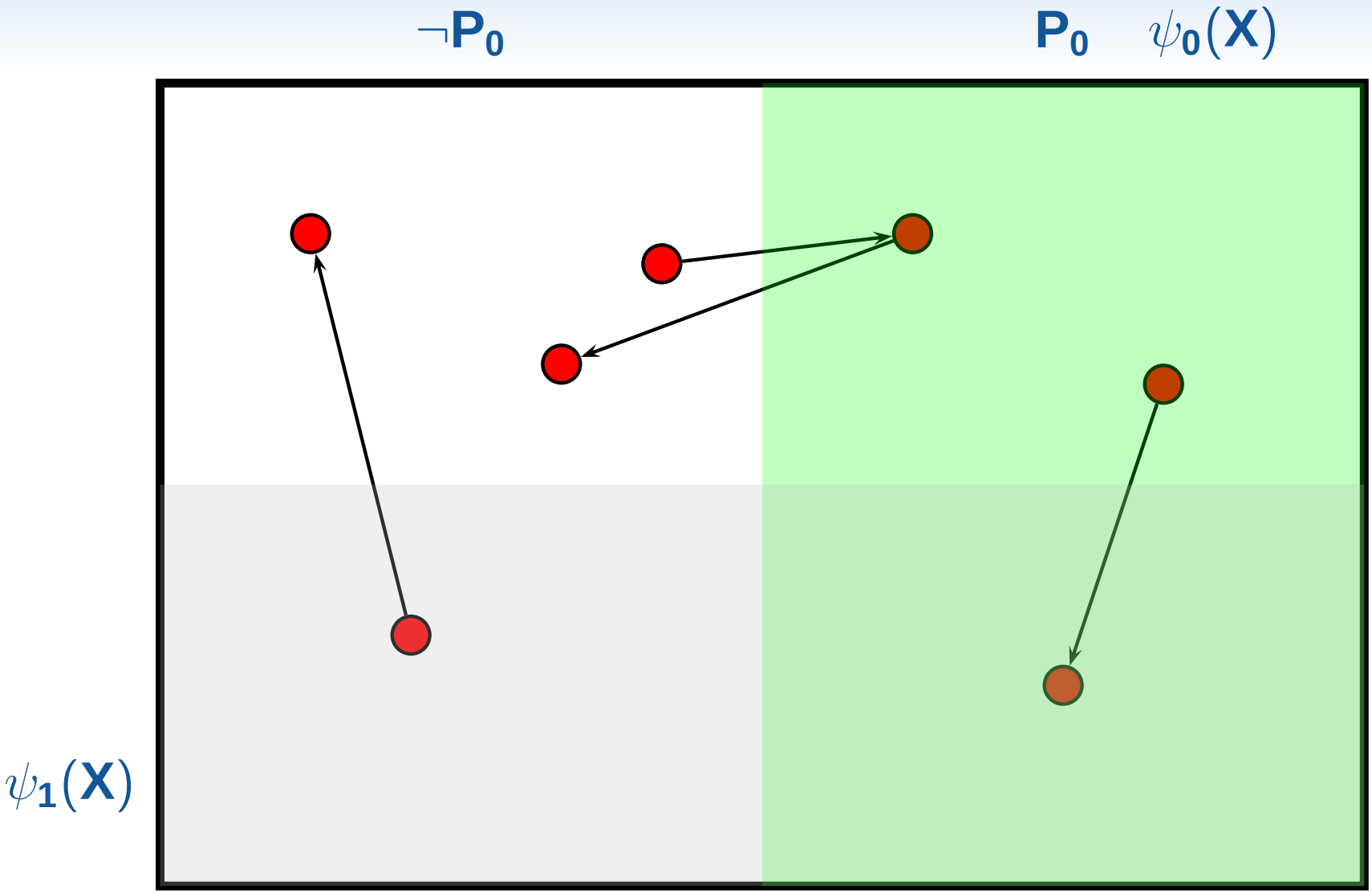
$P_0 \quad \psi_0(X)$



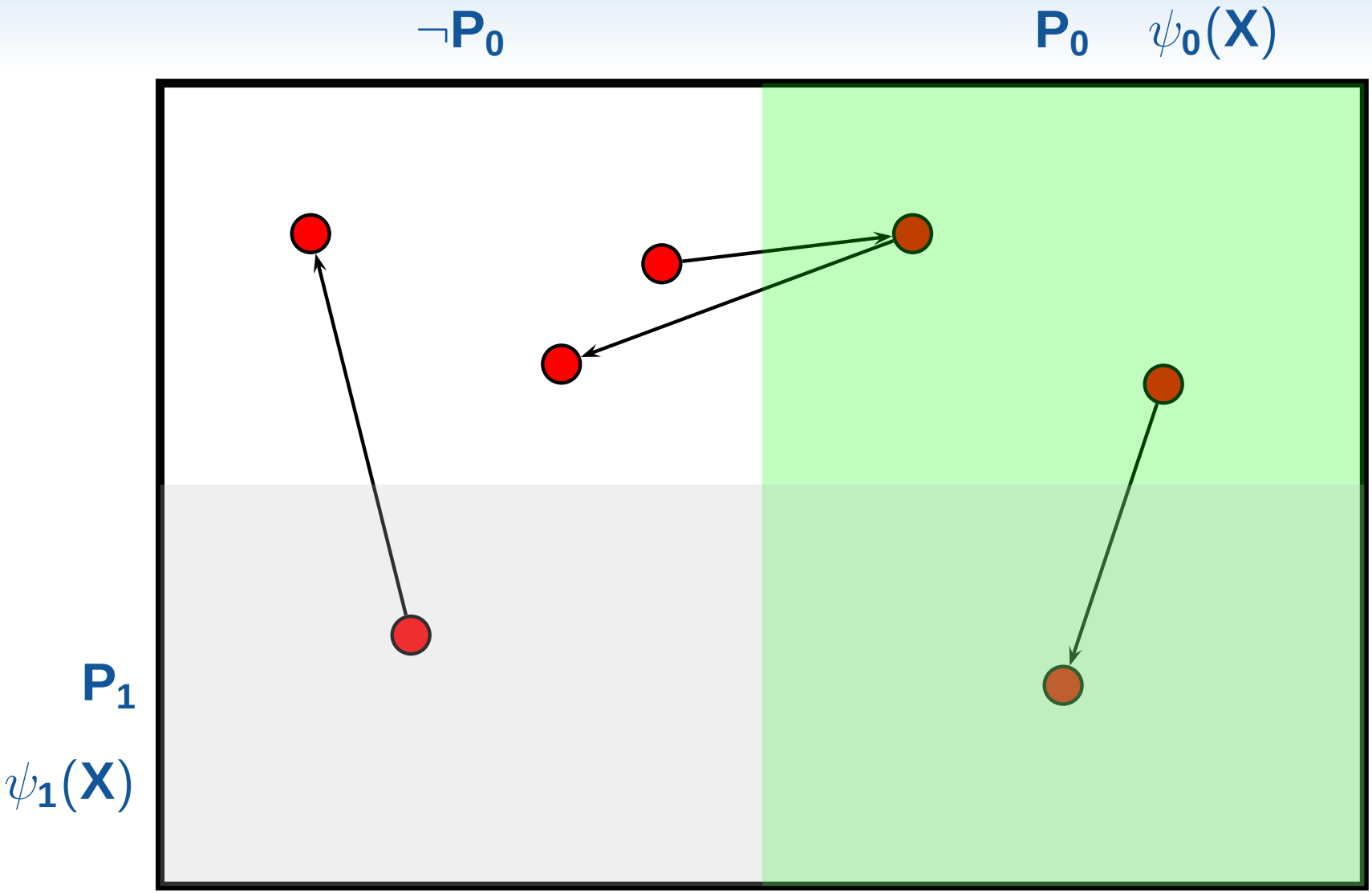
CEGAR based on Predicate Abstraction



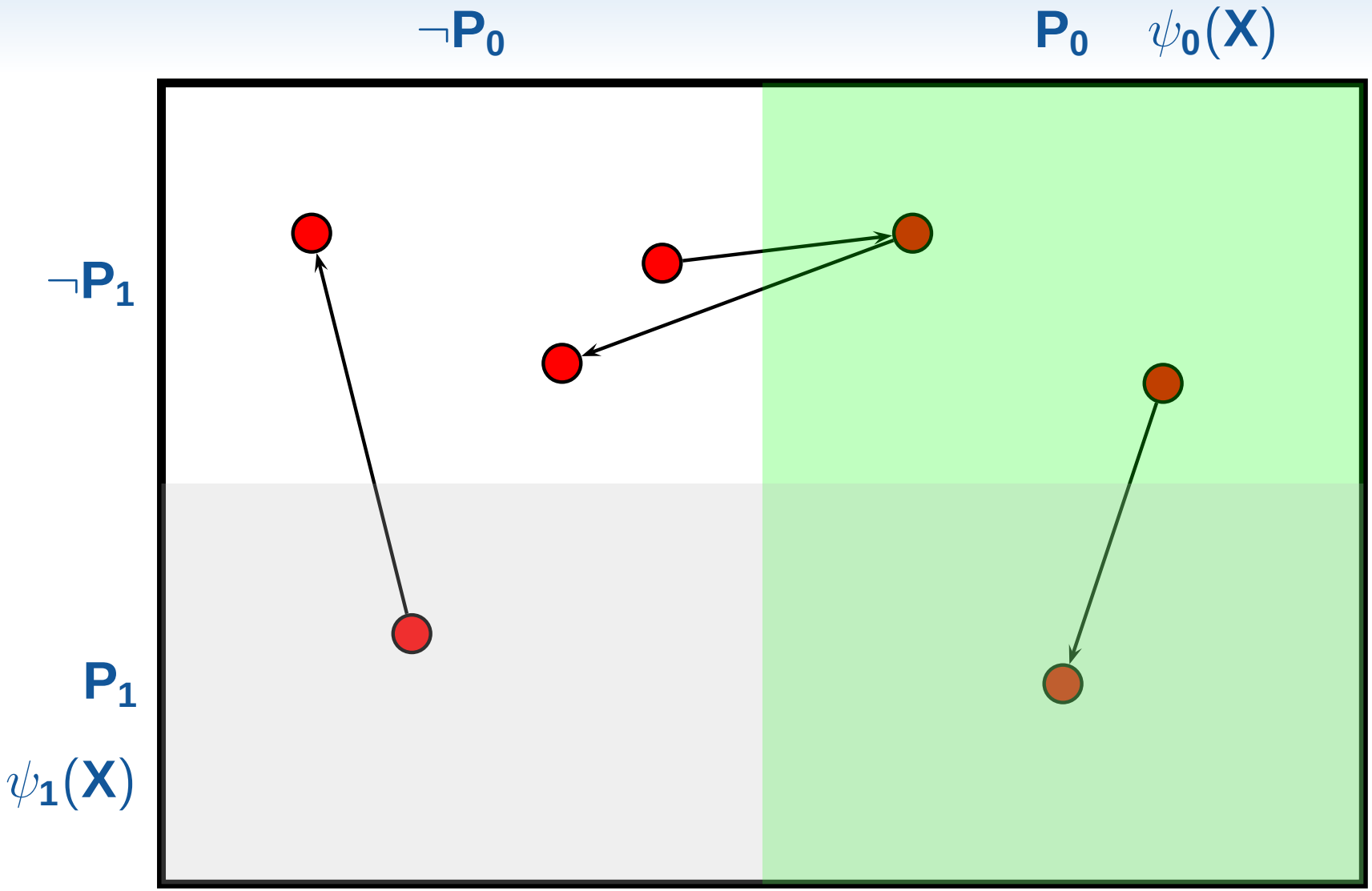
CEGAR based on Predicate Abstraction



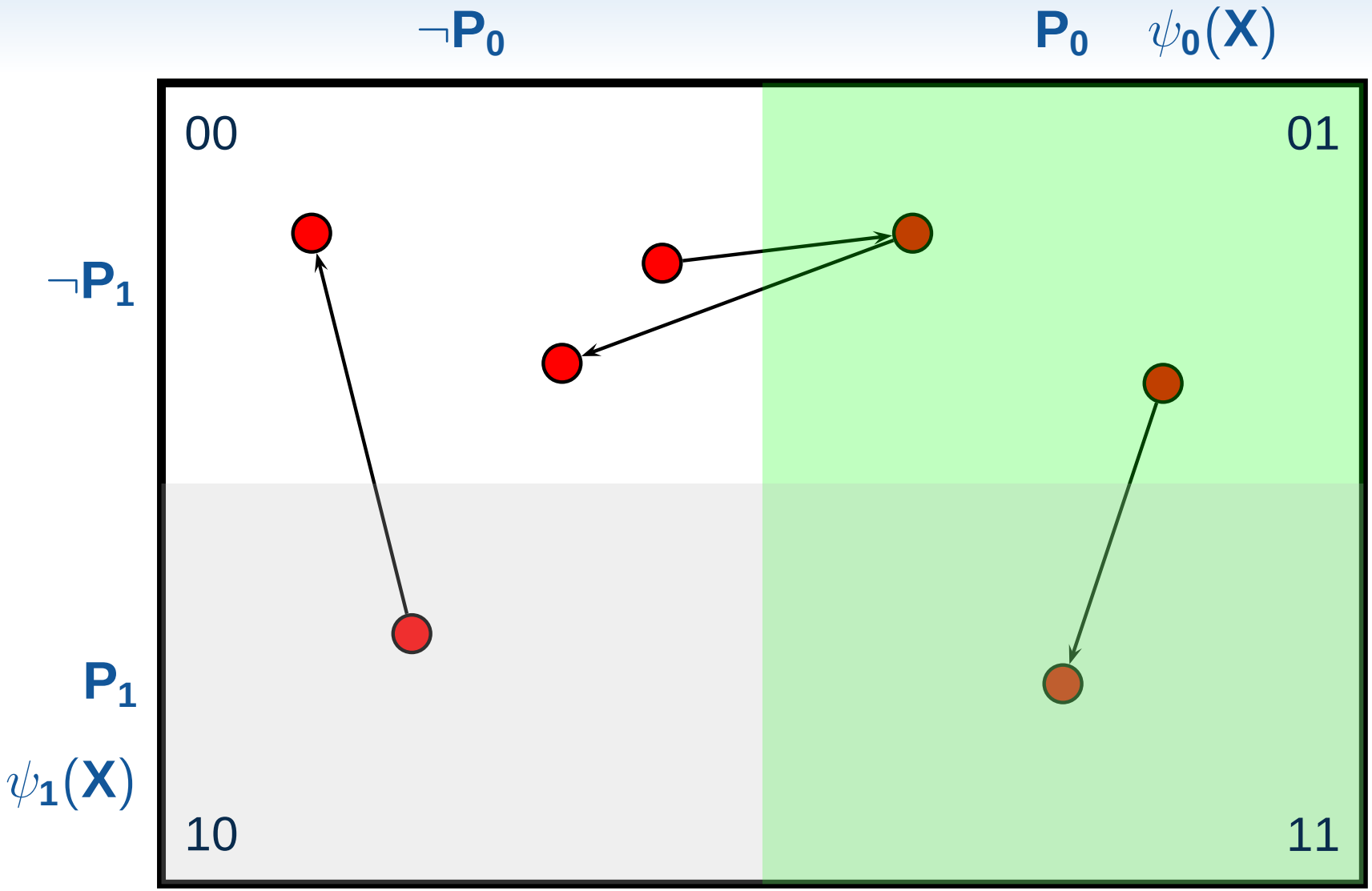
CEGAR based on Predicate Abstraction



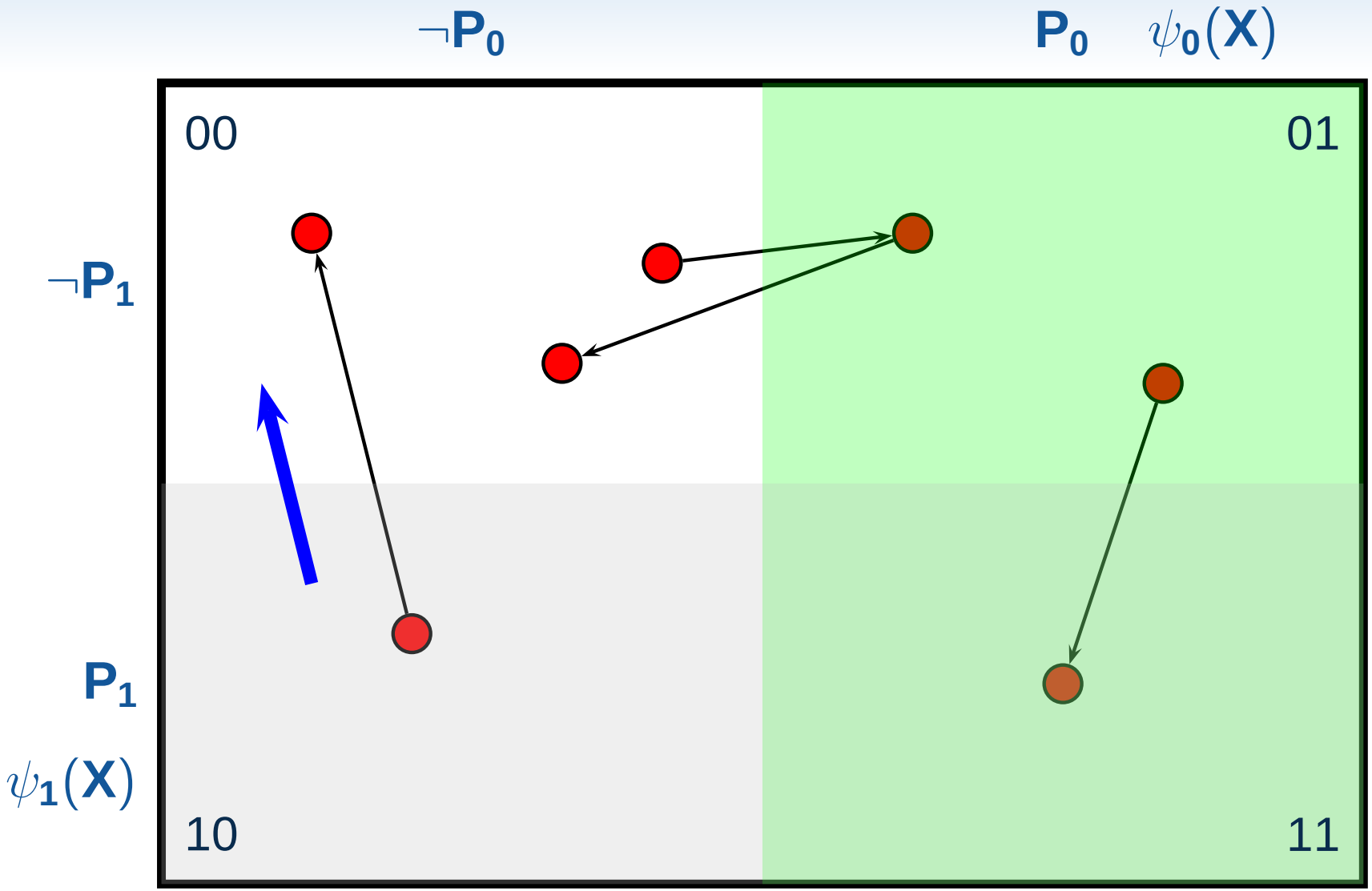
CEGAR based on Predicate Abstraction



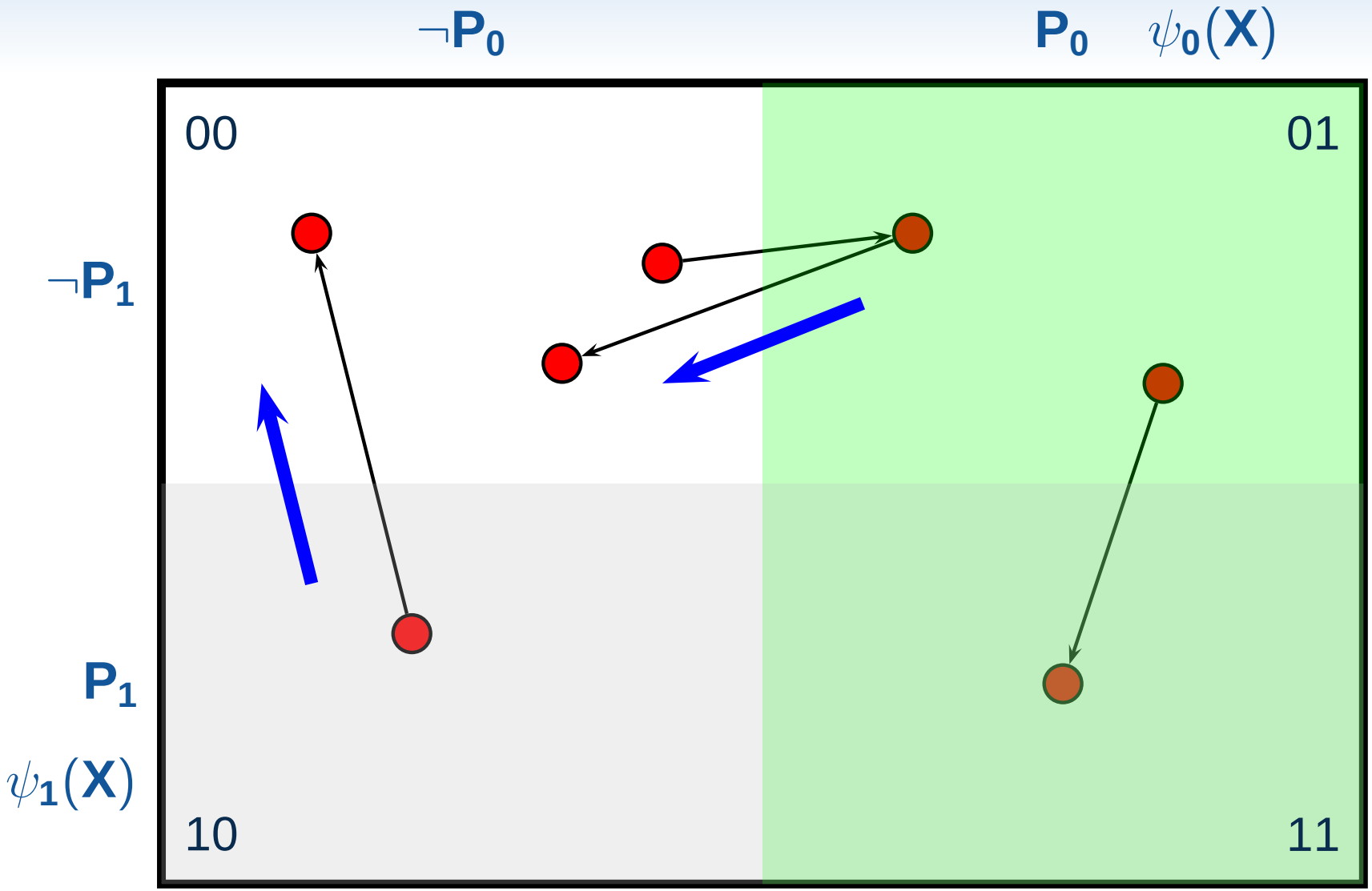
CEGAR based on Predicate Abstraction



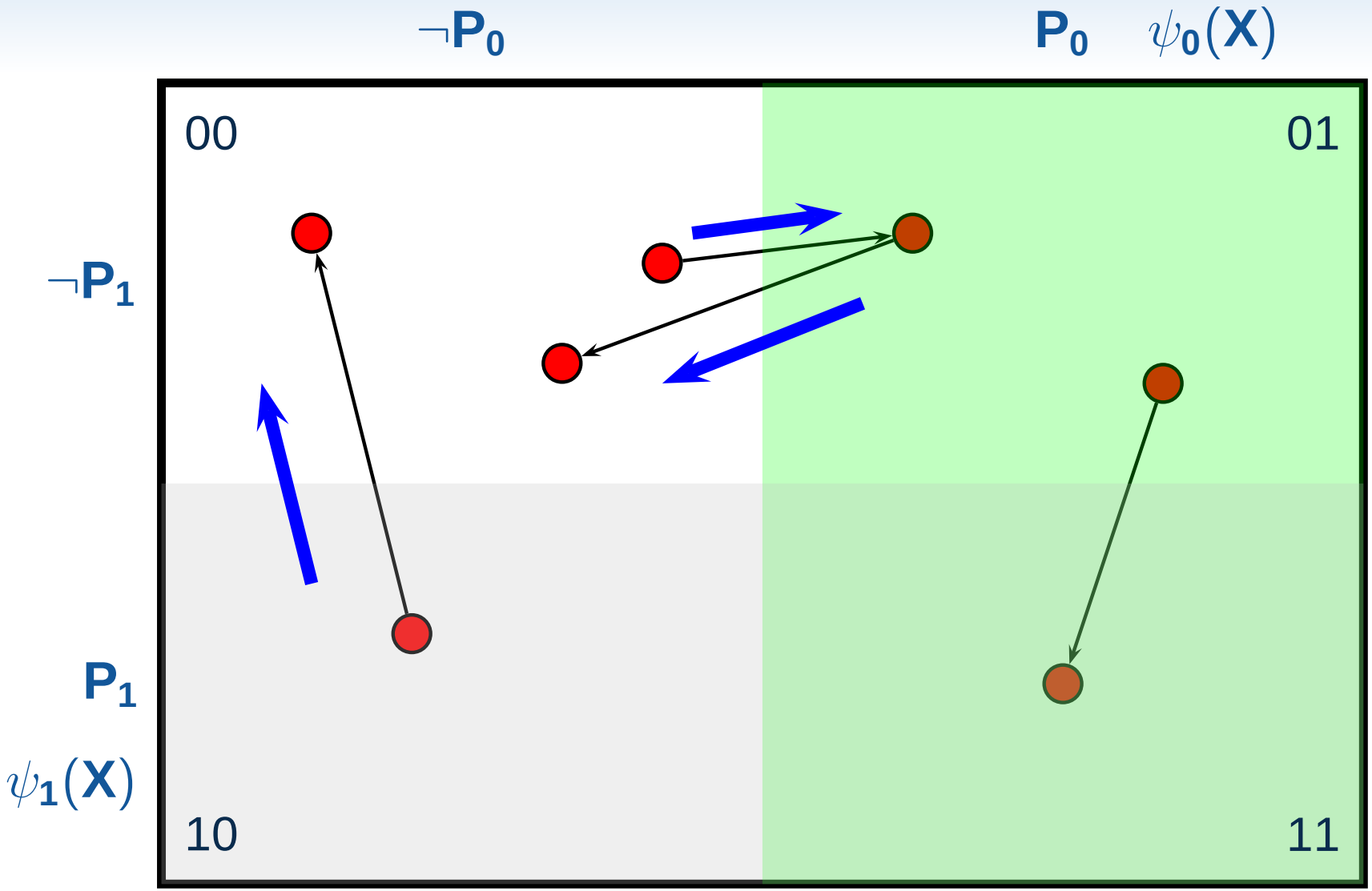
CEGAR based on Predicate Abstraction



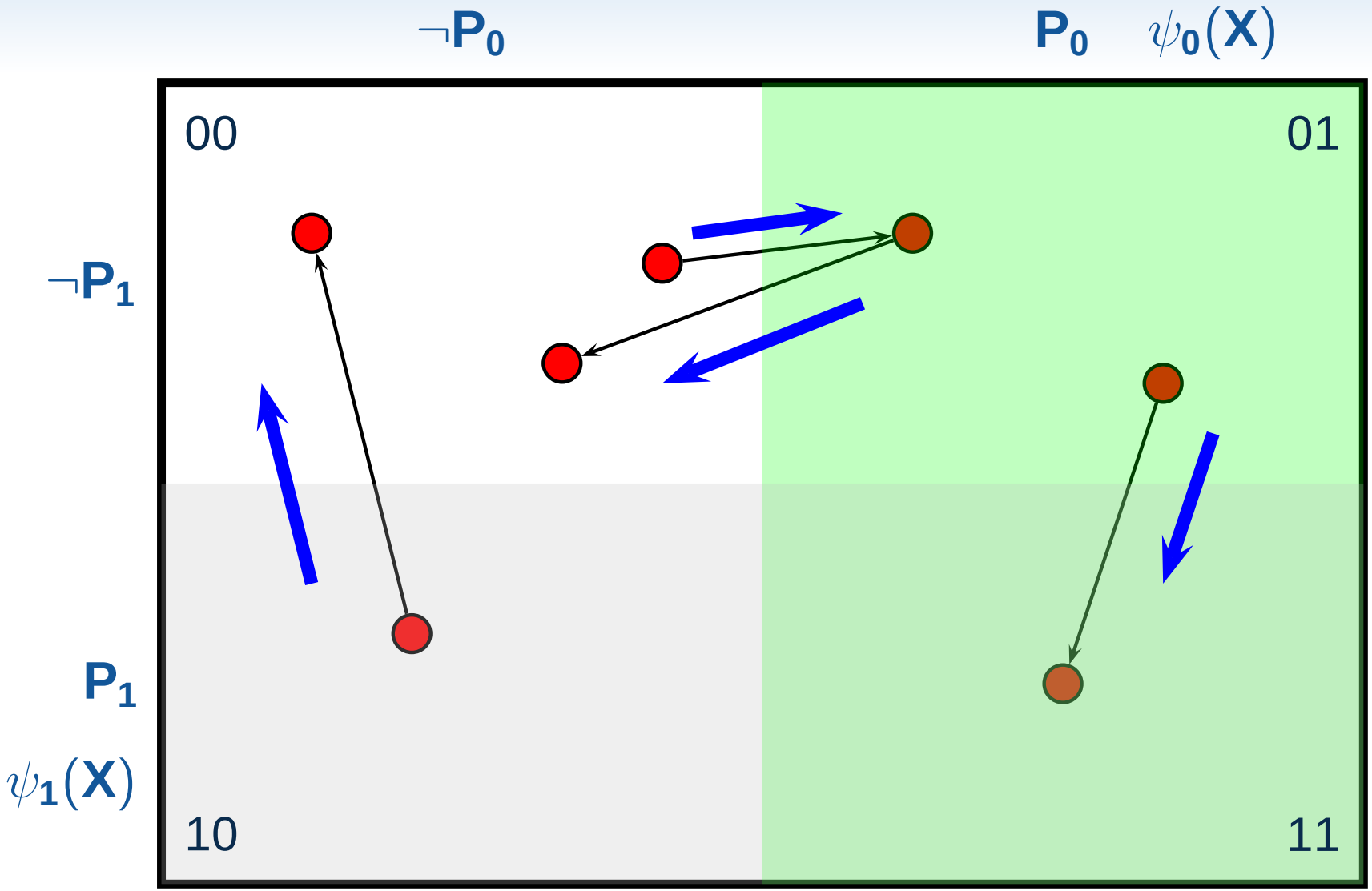
CEGAR based on Predicate Abstraction



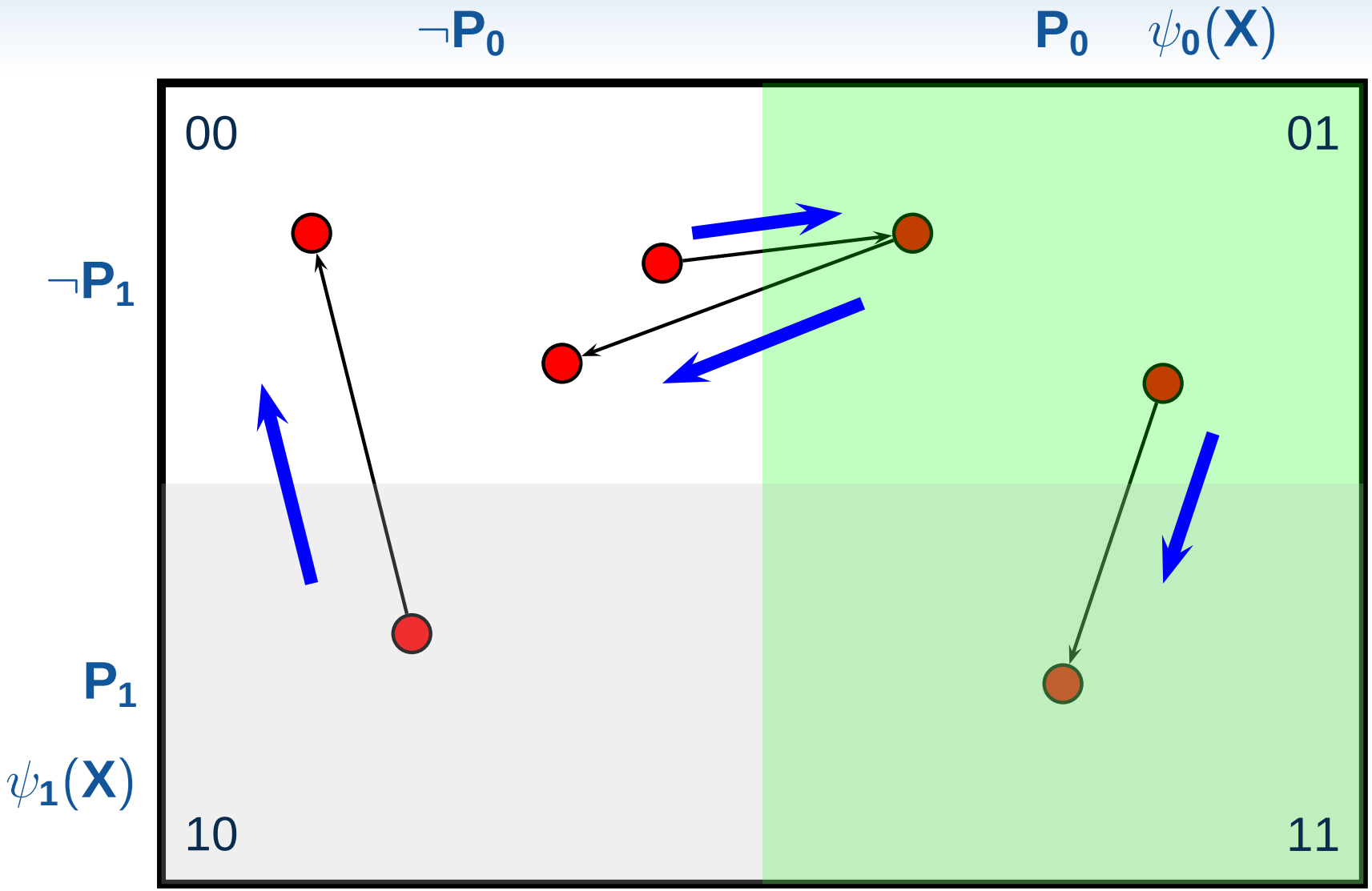
CEGAR based on Predicate Abstraction



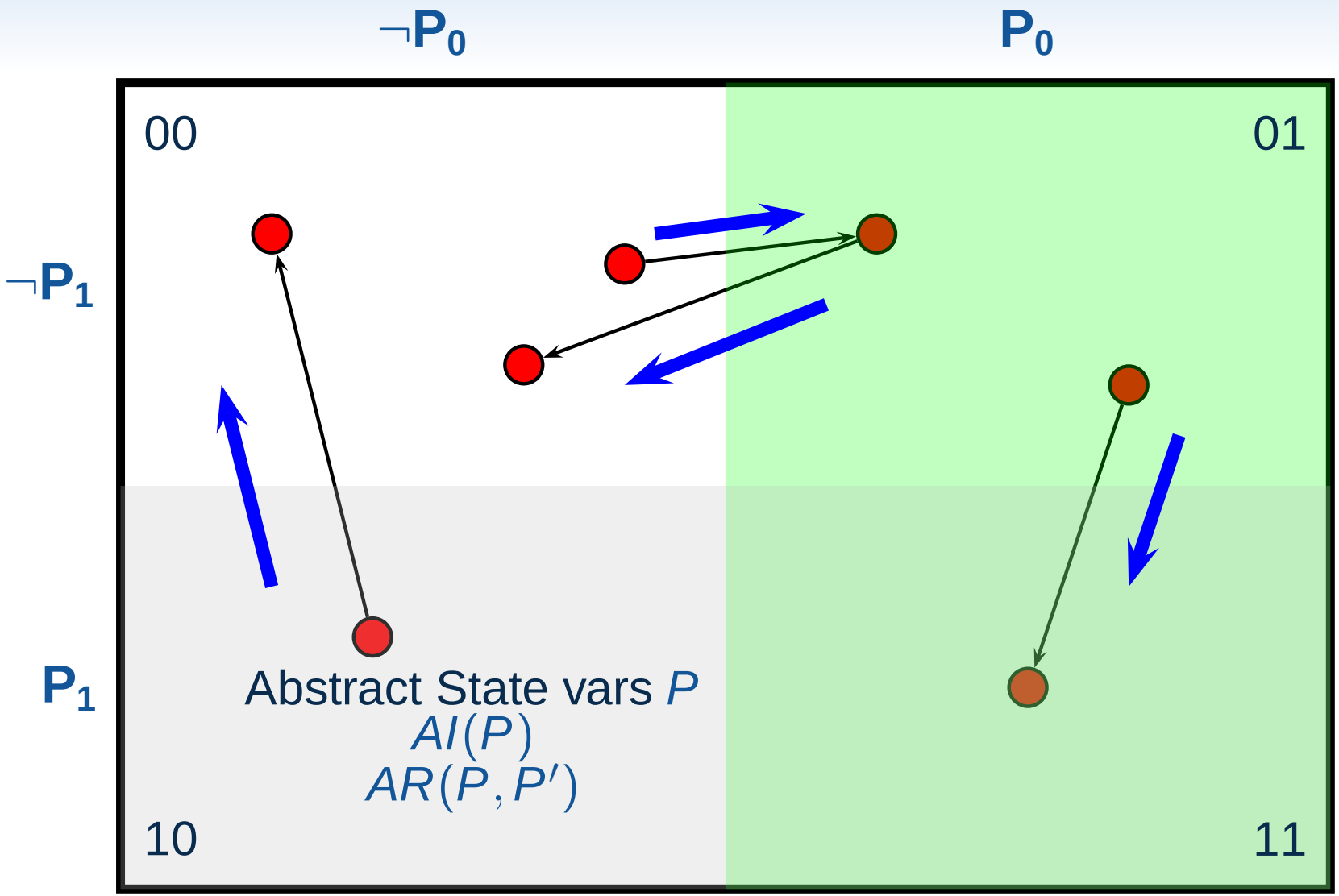
CEGAR based on Predicate Abstraction



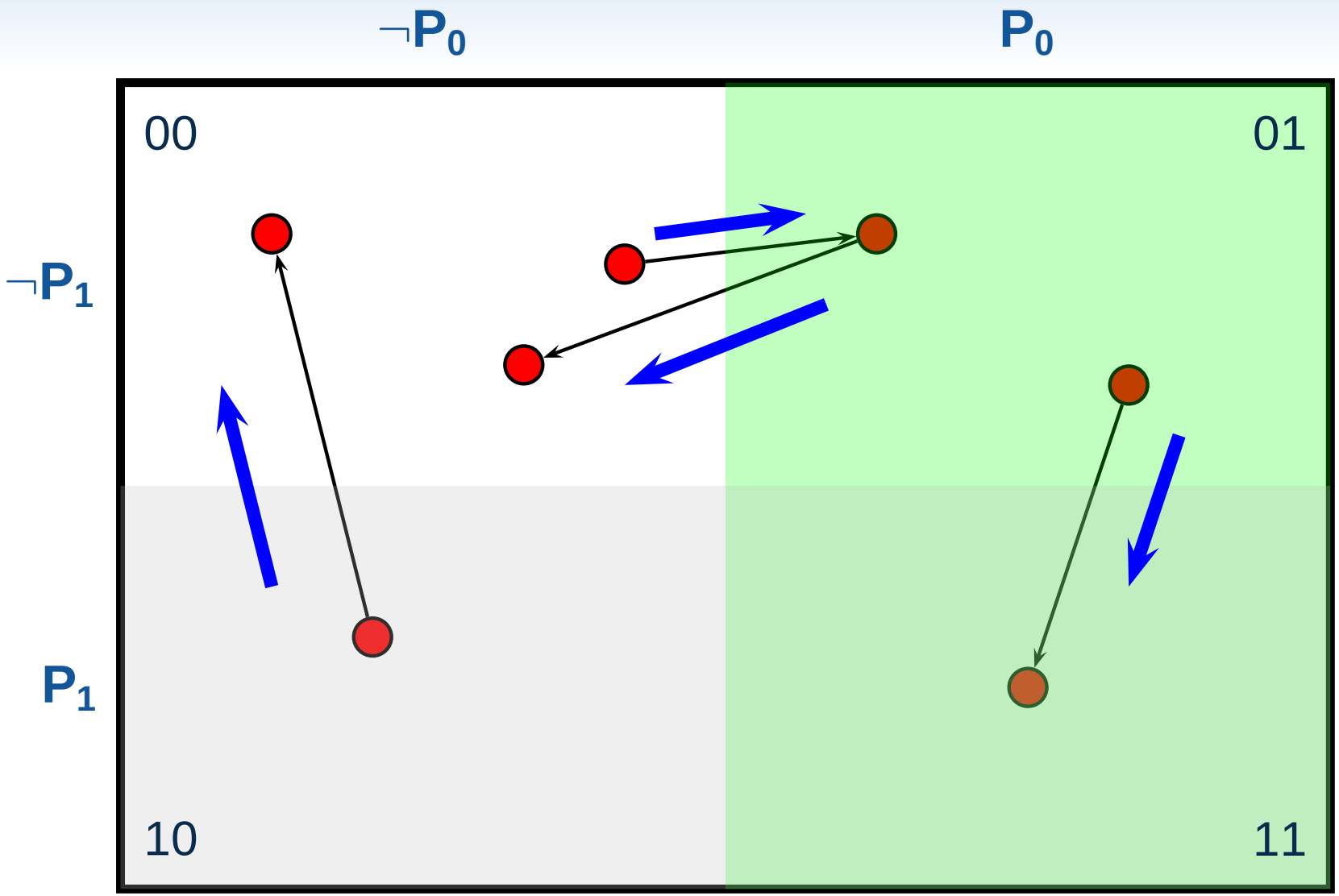
CEGAR based on Predicate Abstraction



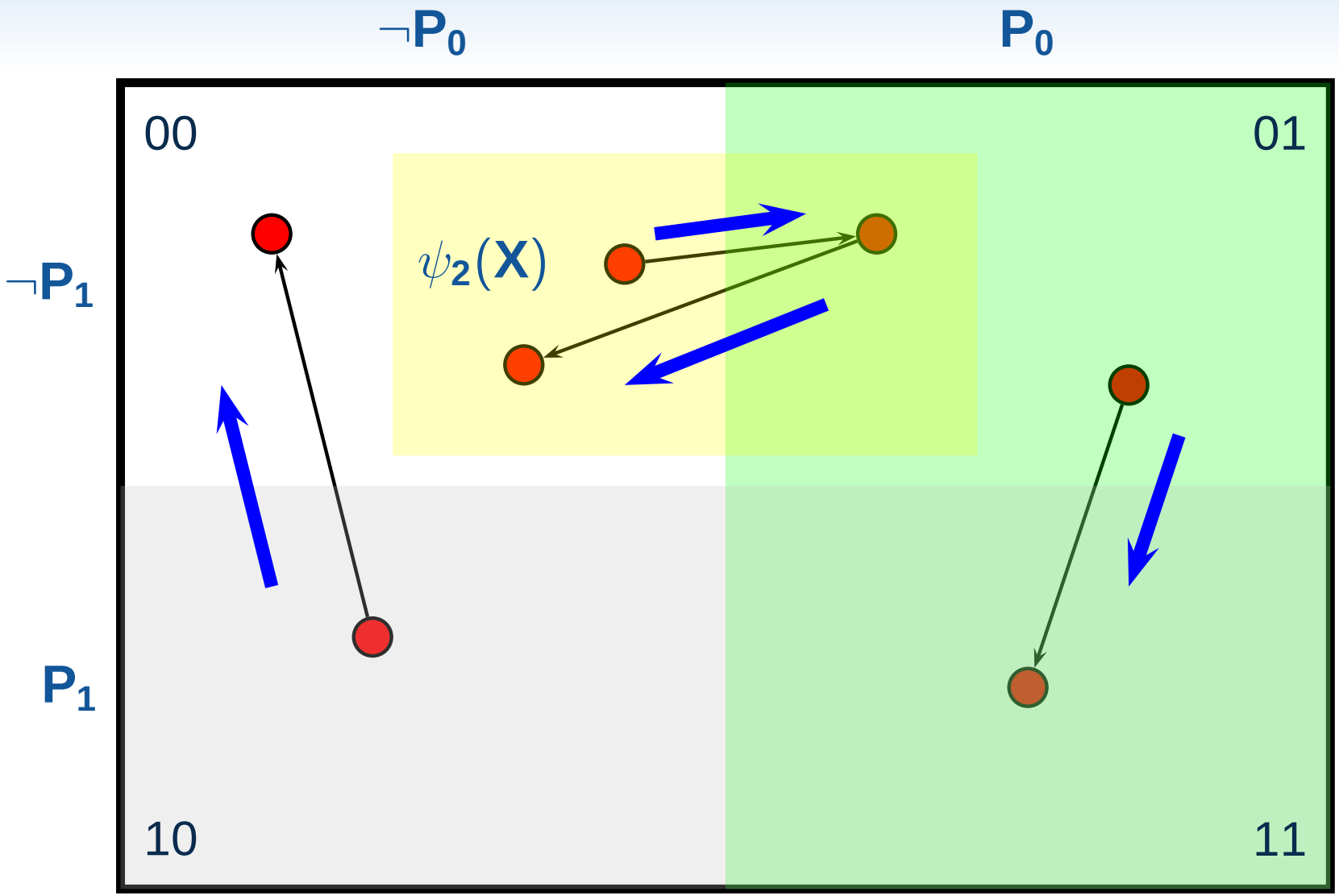
CEGAR based on Predicate Abstraction



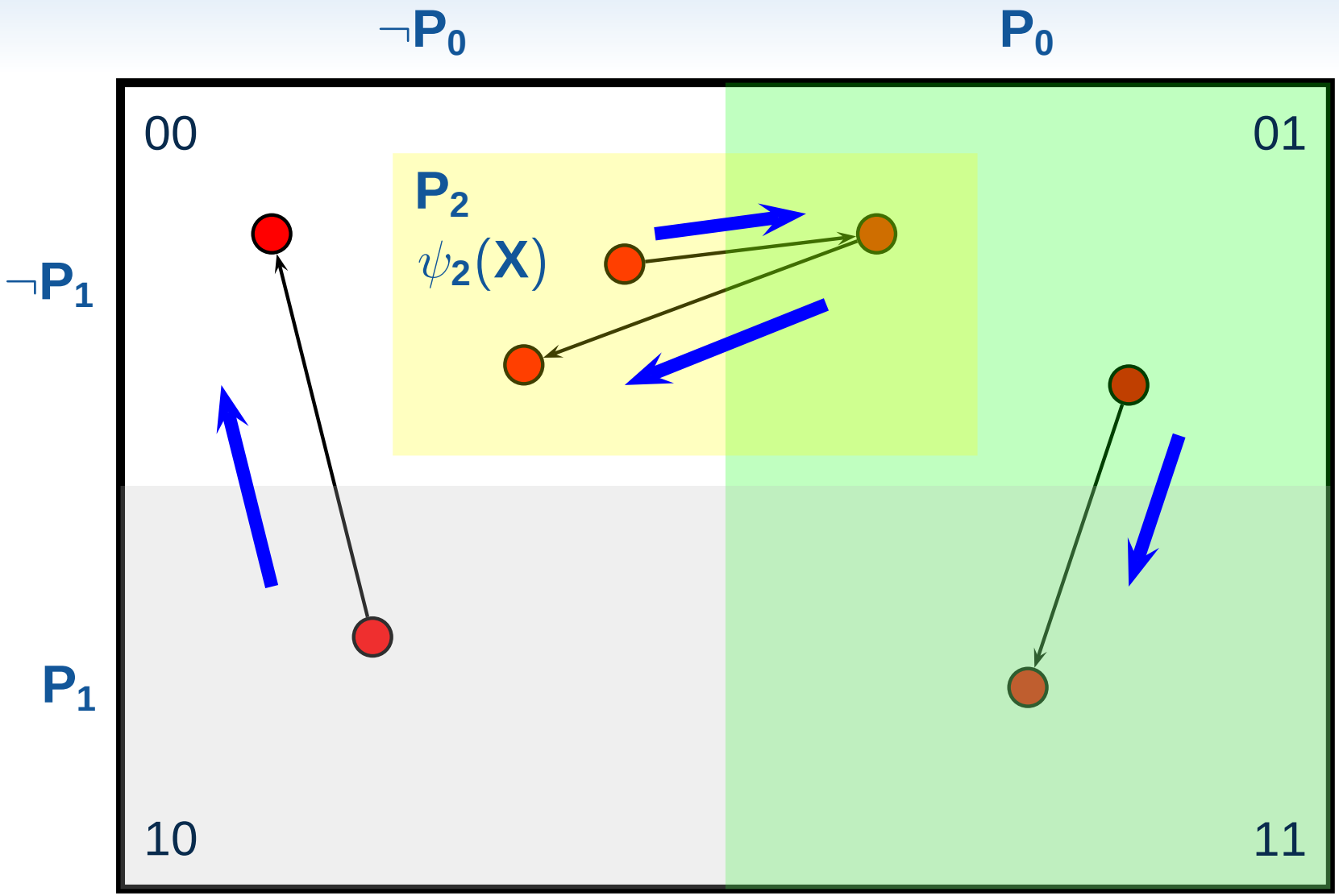
CEGAR based on Predicate Abstraction



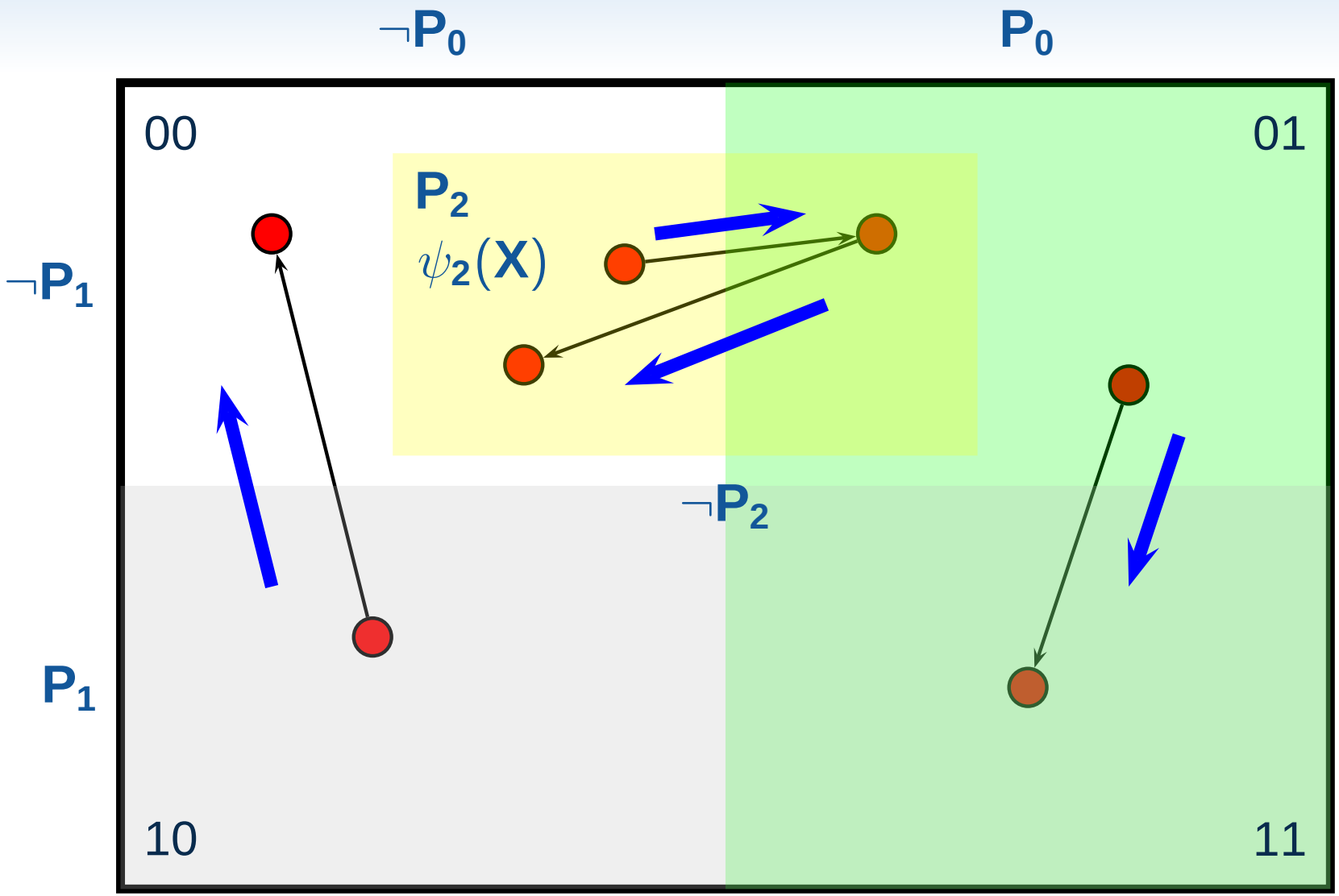
CEGAR based on Predicate Abstraction



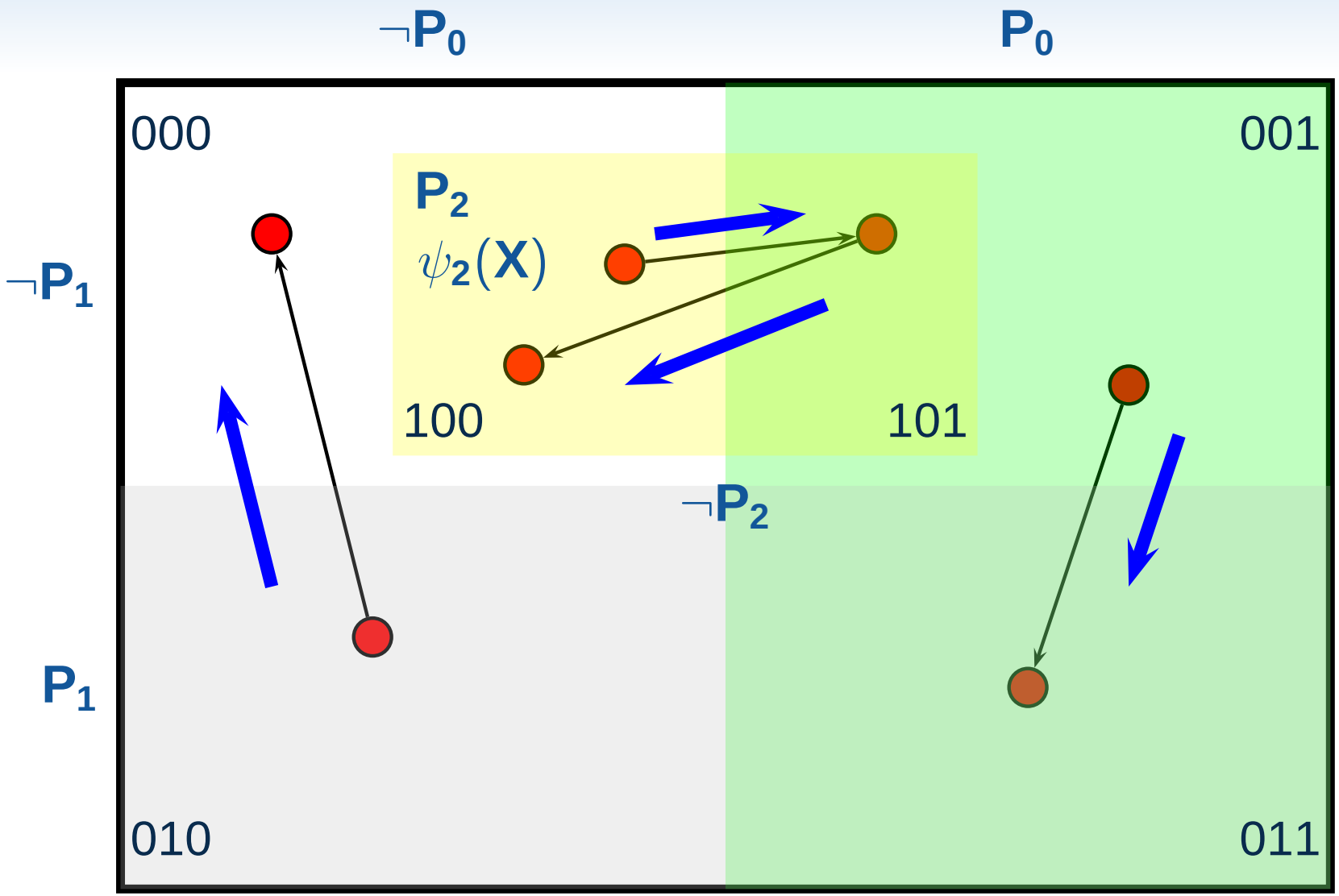
CEGAR based on Predicate Abstraction



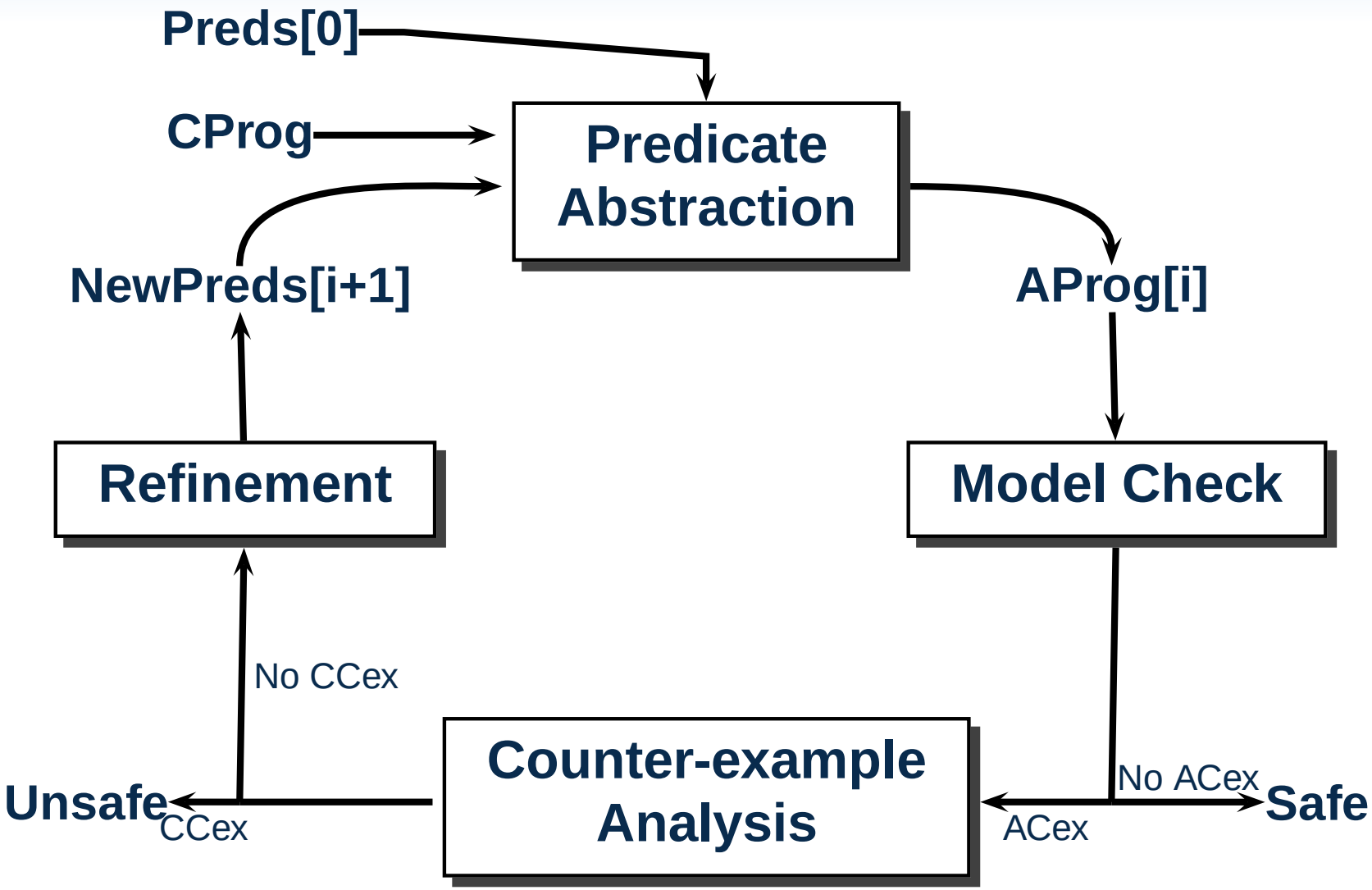
CEGAR based on Predicate Abstraction



CEGAR based on Predicate Abstraction



CEGAR with Predicate Abstraction



CEGAR without AllSMT

- Abstract transition system computed with AllSMT:
  - Exponential in the number of predicates.
  - Major bottleneck of CEGAR.
  - Prevents the analysis of the abstract system.
- Main idea [Ton09]: avoid computing the abstract state space
  - how: embedding the abstraction definition into the BMC/k-induction encodings;
  - abstract transitions implicitly computed by the SMT solver;
  - similar to lazy abstraction but completely symbolic and without any image computation/quantifier elimination.
- Applicable when the abstraction α induces an equivalence relation EQ_α among the concrete states.
  - For predicate abstraction, $EQ_\alpha(X, X') = \bigwedge_{P \in \mathcal{P}} P(X) \leftrightarrow P(X')$.
- Example of application:
  - Concrete unrolling: $\bigwedge_{0 \leq h \leq k-1} R(X_h, X_{h+1})$
  - Abstract unrolling: $\bigwedge_{0 \leq h \leq k-1} R(X_h, X'_h) \wedge EQ_\alpha(X'_h, X_{h+1})$

Other symbolic approaches for STS

- not SMT-based, combines boolean reasoning with constraint solving
- Functionally Reduced AIG's (FRAIGs) [DDH+07]
  - Use And-Inverter-Graphs (AIGs) to represent a set of states of the system.
  - Pre-Image computation via a specialized quantifier-elimination technique.
  - SMT-based fixed-point.
- RED [Wan04]
  - Hybrid-Restriction Diagrams (HRDs): BDD-like data structure to represent the state space.
  - Predicates of the form $\sum a_i \cdot x_j \bowtie a$
  - HRD node: right-hand-side of the predicate ($\sum a_i \cdot x_j$).
  - HRD arc: left-hand-side of the predicate (i.e. a).
  - Representation for the union of a set of convex polyhedra.
  - Non-canonical.

Outline

Linear Hybrid Automata as STS (1/1)

$H := \langle Q, A, Q_0, R, X, \mu, \iota, \xi, \theta \rangle$ is encoded in the STS $S = \langle V, \mathcal{W}, \textsc{Init}, \textsc{Invar}, \textsc{Trans} \rangle$, where:

- $V := \{\text{boolean encoding of } Q\} \cup X \cup t$,
  - $t \in \mathbb{R}$ counts the amount of time elapsed.
  - Shorthand: $loc = q$ is the encoding of the location q.
- $W := \{\text{boolean encoding of the set } A \cup \{\textsc{t}, \textsc{s}\}\}$,
  - $\textsc{t}$ is the "time elapse" event.
  - $\textsc{s}$ is the "stutter" event.
  - Shorthand: $\varepsilon = a$ is the encoding of the event a.
- $\textsc{Init} := t = 0 \wedge \bigwedge_{q \in Q}(loc = q \rightarrow \iota(q)(X))$;
- $\textsc{Invar} := \bigwedge_{q \in Q}(loc = q \rightarrow \xi(q)(X))$;
- $\textsc{Trans} := \bigwedge_{q \in Q}(loc = q \rightarrow (\textsc{Timed}_q \vee \bigvee_{\langle q,a,q' \rangle \in R} \textsc{Untimed}_{\langle q,a,q' \rangle})) \vee \textsc{Stutter}$

Linear Hybrid Automata as STS - (2/2)

- STUTTER := all the variables do not change.
- UNTIMED$_{\langle q,a,q' \rangle} := \varepsilon = a \wedge loc' = q' \wedge t' = t \wedge \theta(\langle q, a, q' \rangle)(X, X')$.
- TIMED$_q := \varepsilon = \top \wedge loc' = loc \wedge t' > t \wedge \overline{\mu(q)}^{(t'-t)}(X, X')$,
  where $\overline{\mu(q)}^{(t'-t)}$ encodes the time elapse step:
  - Note: $\mu(q) := \sum a \cdot \dot{x} \bowtie b, a, b \in \mathbb{R}$.
  - Time elapse step:
    - We replace each $\dot{x}$ in $\mu(q)$ by $\frac{x'-x}{t'-t}$.
    - Since $t' > t$: $\overline{\mu(q)}^{(t'-t)} := \sum a \cdot (x' - x) \bowtie (t' - t) \cdot b$.

Remark: The encoding may be extended to a subclasses of non-linear Hybrid Automata. In that case, we need a special handling of invariants to get rid of a universal quantifier.

Encoding of HAN - Global-time semantics

- We encode the network of LHAs $\mathcal{N} = H_1 || \ldots || H_n$ in the STS $S_{\mathcal{N}}$.
- $S_{\mathcal{N}}$ is the composition of:
  - The STSs $S_1, \ldots, S_n$ which encodes $H_1, \ldots, H_n$.
  - The synchronization constraints $SYNC$:

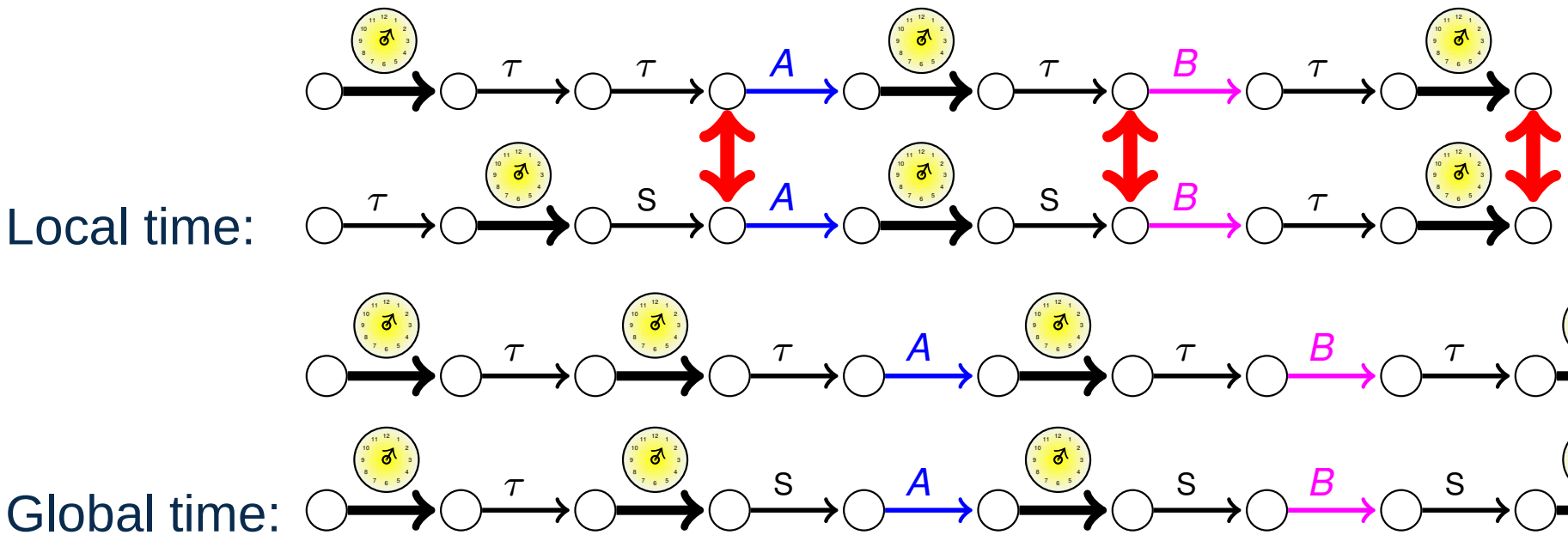
$$\textsc{Sync} := \bigwedge_{1 \le j < h \le n} \bigwedge_{a \in A_j \cap A_h \cup \{\tau\}} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a)$$
$$\wedge \bigwedge_{a \in (A_j \setminus A_h)} (\varepsilon_j = a \rightarrow \varepsilon_h = \mathsf{s})$$
$$\wedge \bigwedge_{a \in (A_h \setminus A_j)} (\varepsilon_h = a \rightarrow \varepsilon_j = \mathsf{s})$$

SYNC encodes the global-time semantics.

In the actual encoding, the variable TIME is replaced by a variable δ, which represents the time elapsed during the continuous transition ($t' = t$).

Local-Time semantics

- The time evolves independently in each automaton:
  - Local time scale.
  - The continuous evolution is a local transition.
- Automata agree on the time elapsed:
  - On synchronizations.
  - At the end of a run.



S = stutter event. τ = local event (no stutter or time).

Equivalence of the two semantics

Definition (Synchronized state)

A state is synchronized if all local clocks agree on the amount of time elapsed.

Theorem ([BJLY98])

A state s is reachable in the global-time semantics iff there exists a synchronized state s' reachable in the local-time semantic.

Encoding of HAN - local-time semantics

The local-time and the global-time semantics differ for the synchronization constraint SYNC:

$$\begin{aligned}\textsc{Sync} := \bigwedge_{1\le j<h\le n} & \bigwedge_{a\in A_j\cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \wedge (\varepsilon_j = a \rightarrow t_j = t_h) \\ & \wedge \bigwedge_{a\in (A_j\setminus A_h)\cup\{\tau\}} (\varepsilon_j = a \rightarrow \varepsilon_h = \mathsf{S}) \\ & \wedge \bigwedge_{a\in (A_h\setminus A_j)\cup\{\tau\}} (\varepsilon_h = a \rightarrow \varepsilon_j = \mathsf{S})\end{aligned}$$

SYNC encodes the local-time semantics.

Step semantics

Step semantics:

- Exploit the independence of the local transitions.
- Relax the *SYNC* constraint such that independent transitions are executed in parallel.

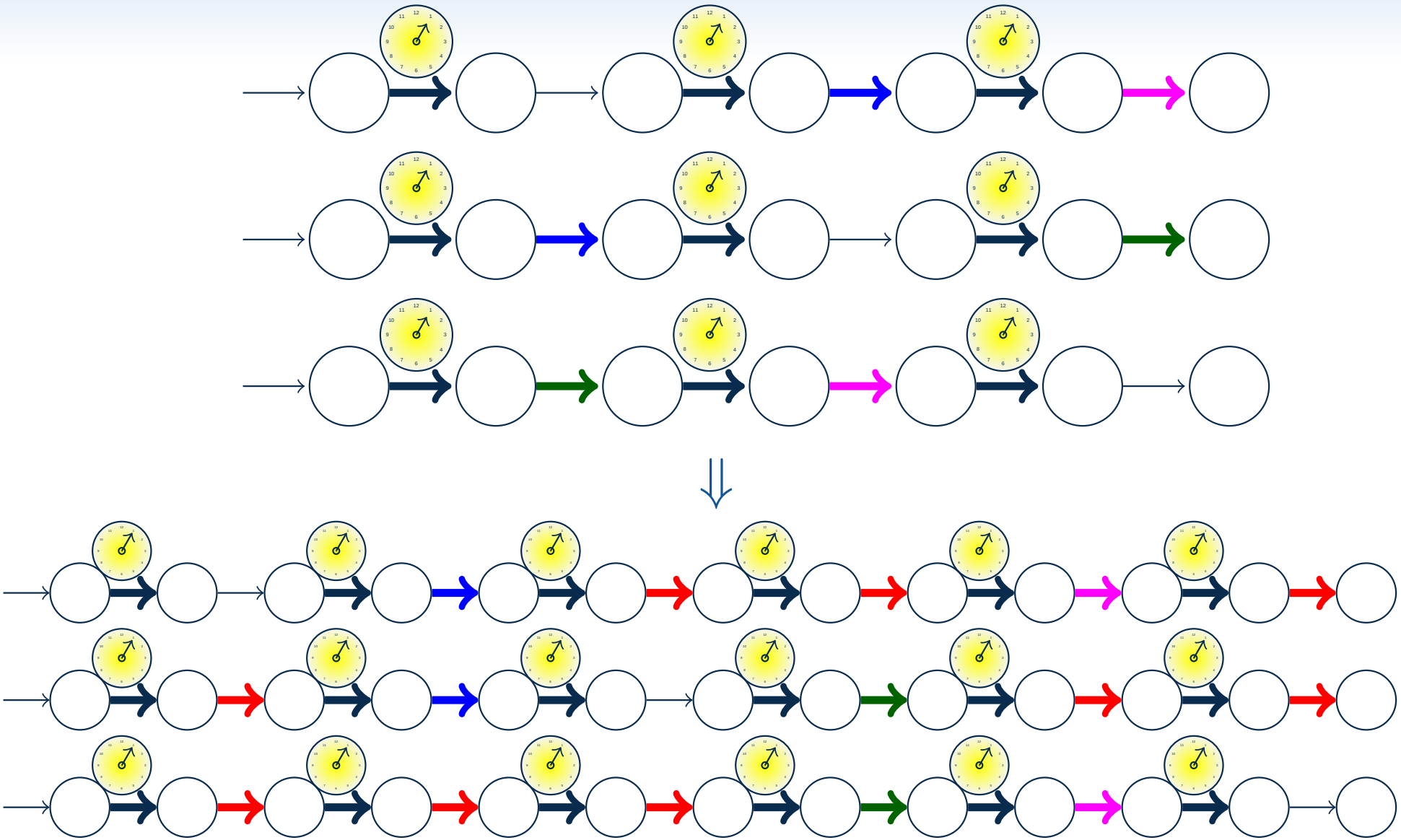
$$\textsc{Sync}_{step} := \bigwedge_{1 \le j < h \le n} \bigwedge_{a \in A_j \cap A_h} (\varepsilon_j = a \leftrightarrow \varepsilon_h = a) \wedge (\varepsilon_j = a \rightarrow t_j = t_h)$$

- Orthogonal to global-time or local-time semantics.

Traditional composition

- Traditional semantics of a network of systems is based on interleaving.
- Required construction of a monolithic hybrid automaton based on the composition of the systems.
- Destroyed structure of the network and results in a loss of efficiency, especially using bounded model checking techniques.

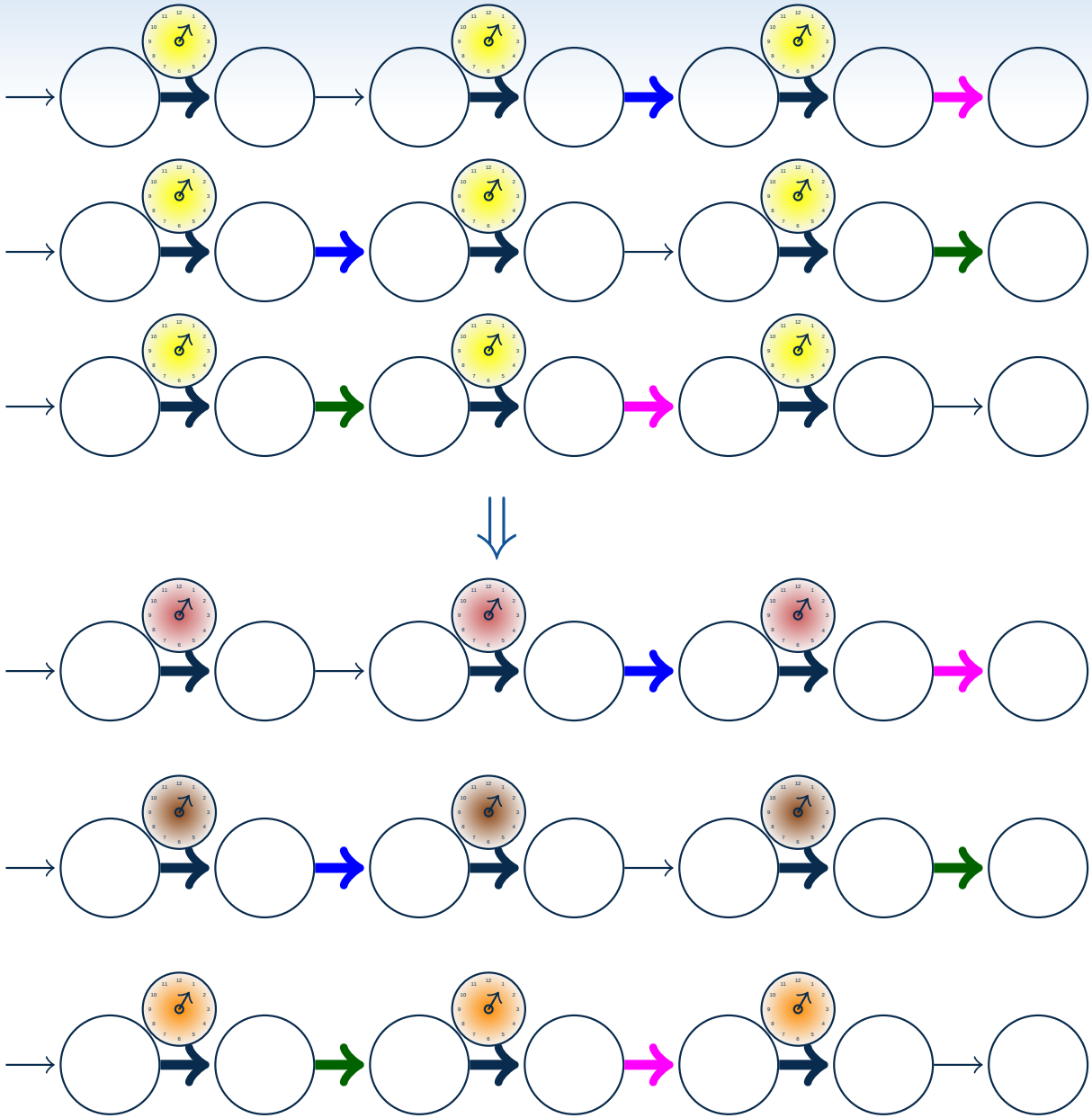
Interleaving effect



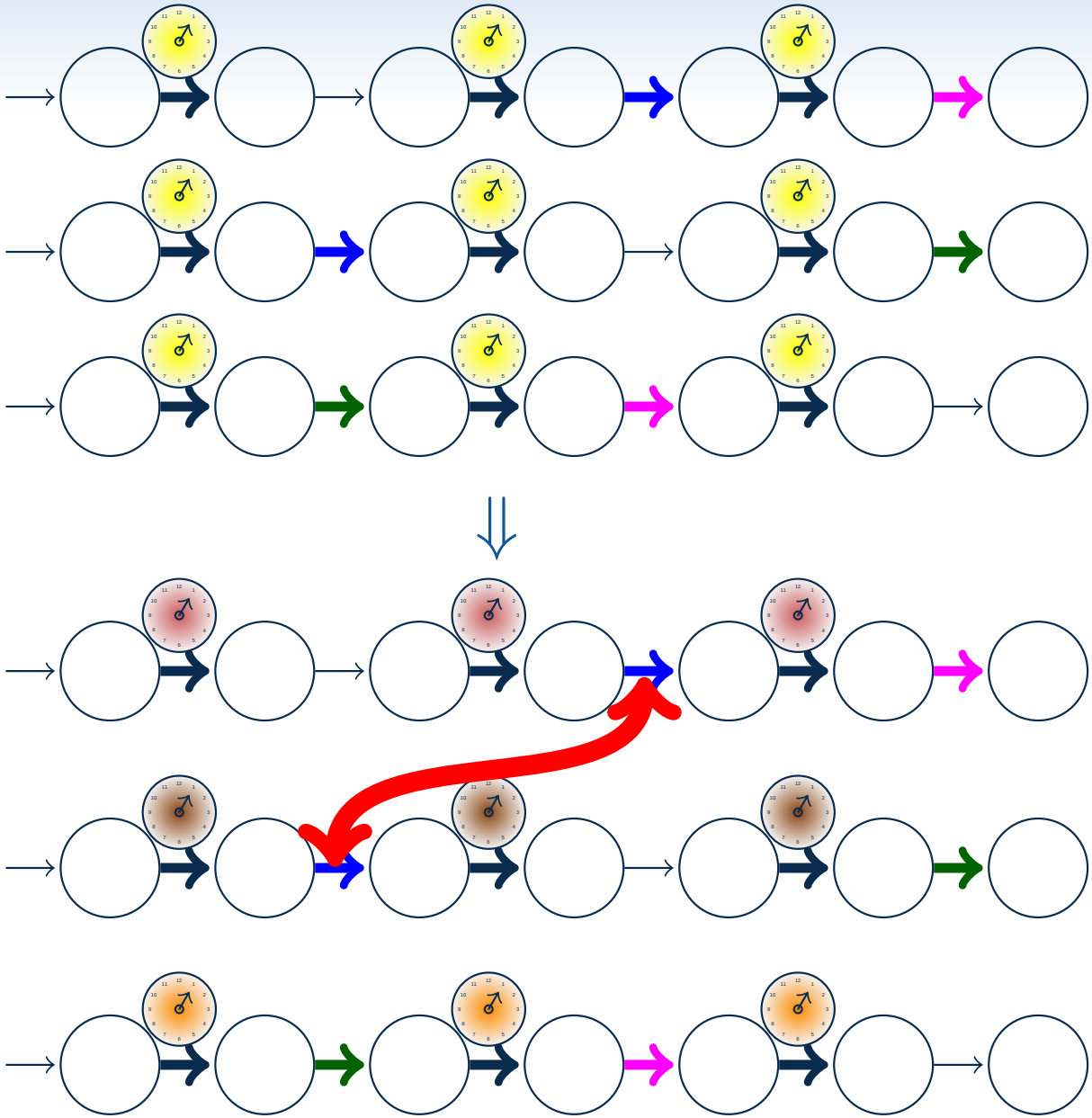
Alternative composition

- Idea:
  - shallow synchronization to improve reachability encoding.
- Shallow synchronized runs:
  - set of local traces compatible wrt synchronization and time.
- Exploiting local clocks:
  - Independent evolution of time.
  - Time is synchronized only on shared events.

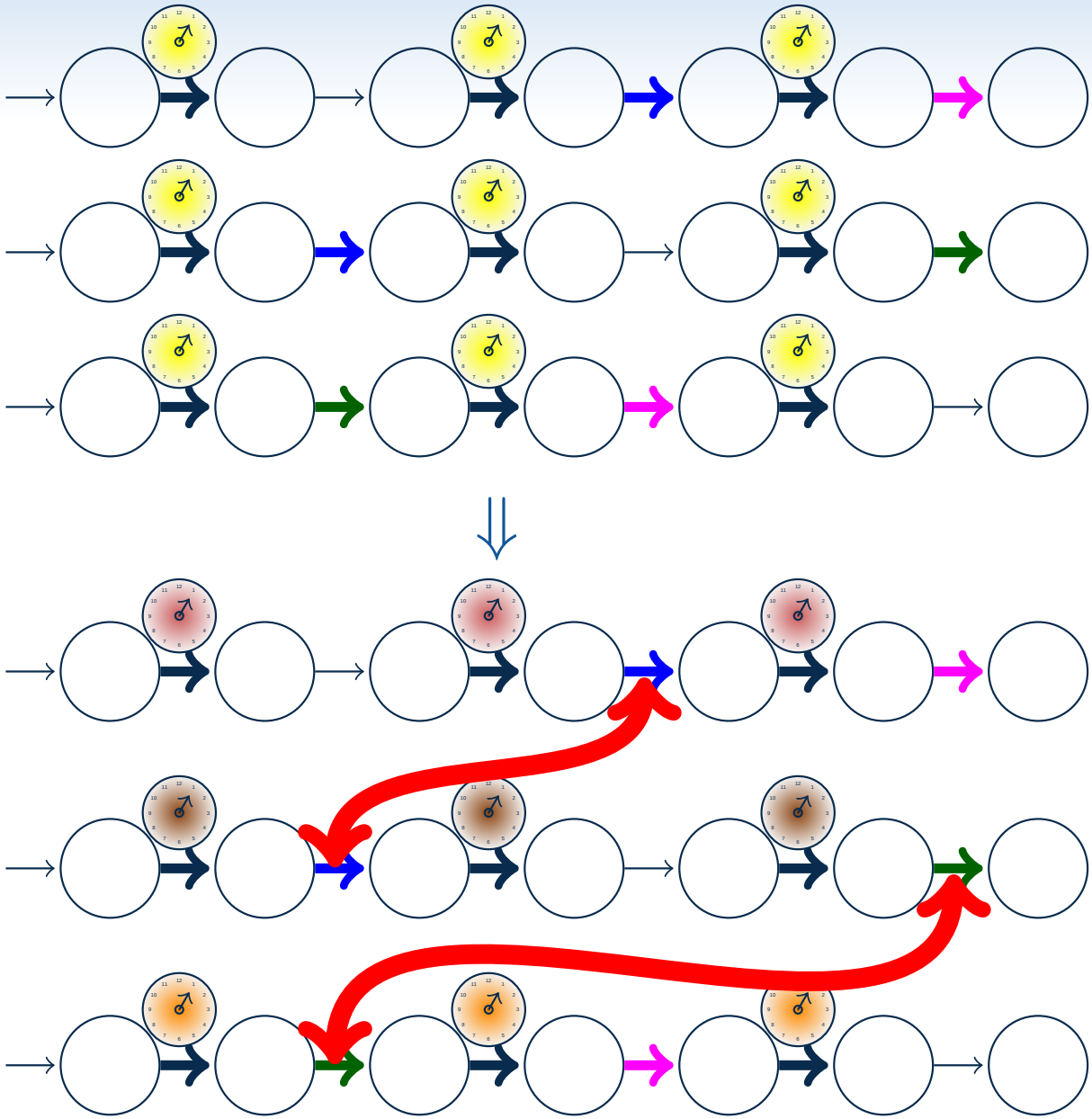
Local time effect



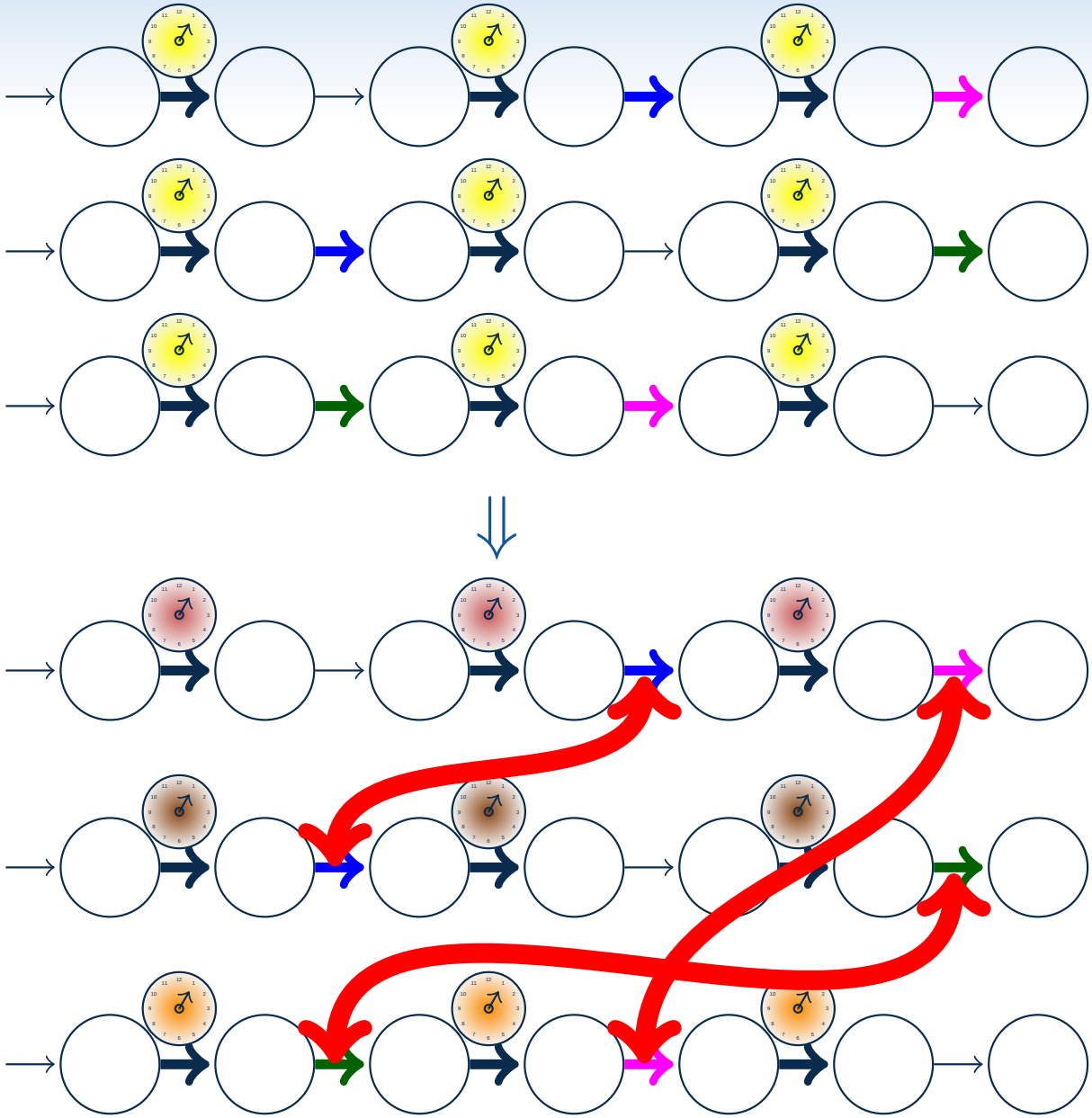
Local time effect



Local time effect



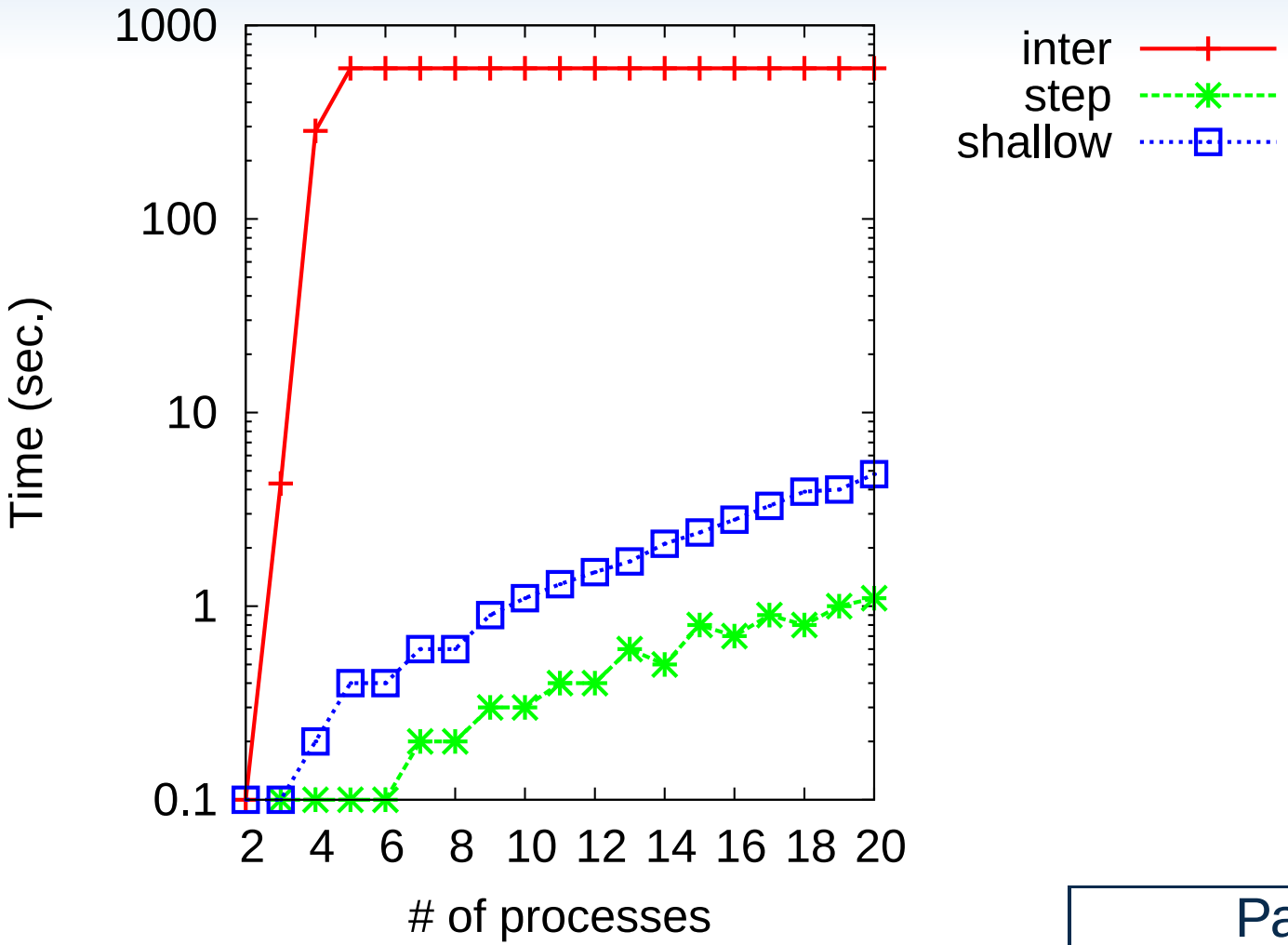
Local time effect



Shallow synchronization

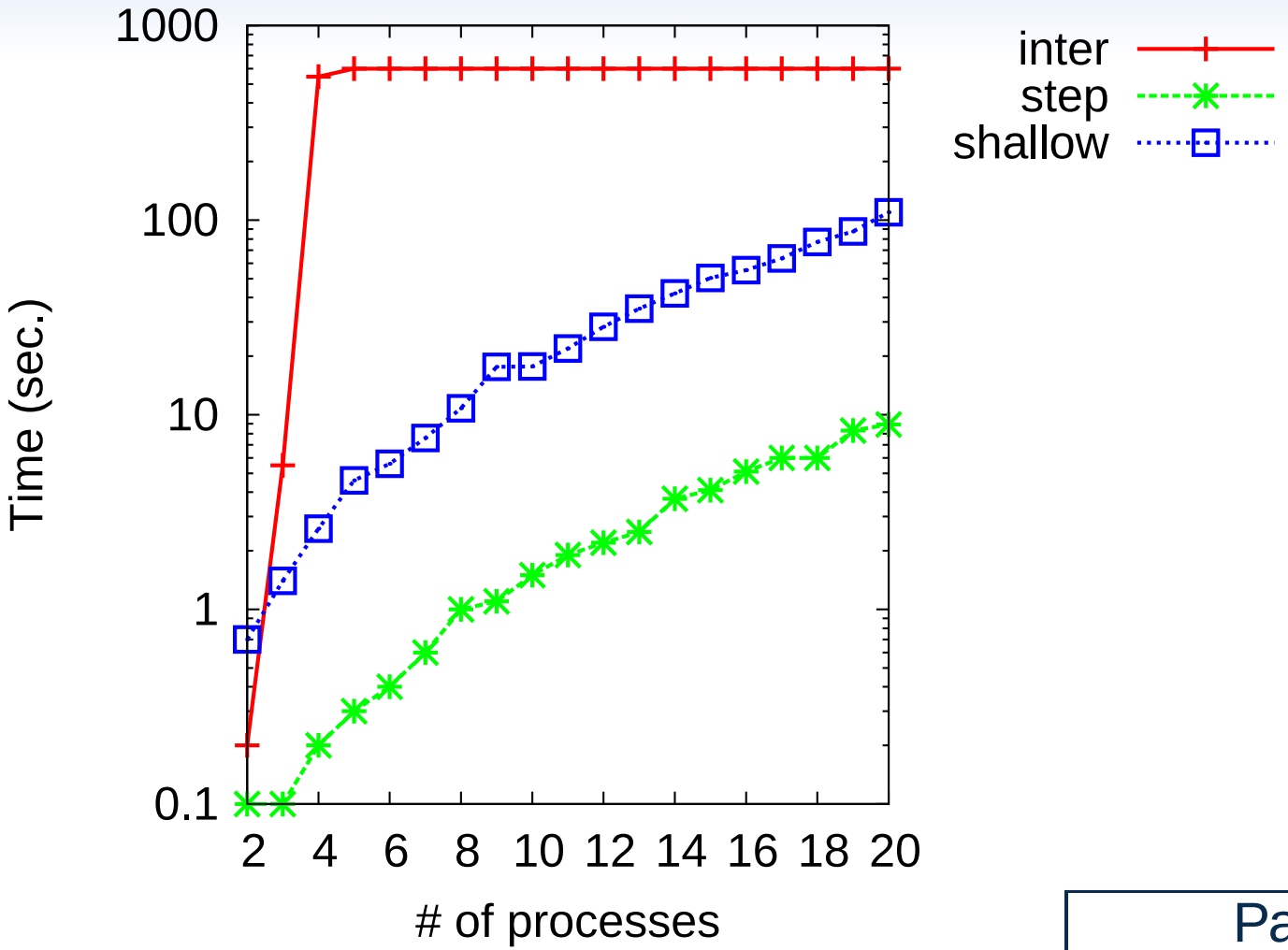
- Shallow synchronization:
  - for all systems S_j and S_h, the sequence of shared events performed by S_j and S_h is the same;
  - for all systems S_j and S_h, for all events a shared by S_j and S_h, S_j performs the i-th occurrence of a at the same time S_h performs the i-th occurrence of a;
  - for all systems S_j and S_h, the time in the last step of S_j is the same to the time in the last step of S_h.
- Different variants of the encoding:
  - Enumerating all possible combinations of occurrences.
  - Exploiting uninterpreted functions.
- Different interaction with the solver:
  - Adding sync while unrolling vs after unrolling.
  - Depth-first search vs. breadth-first search.

Ring



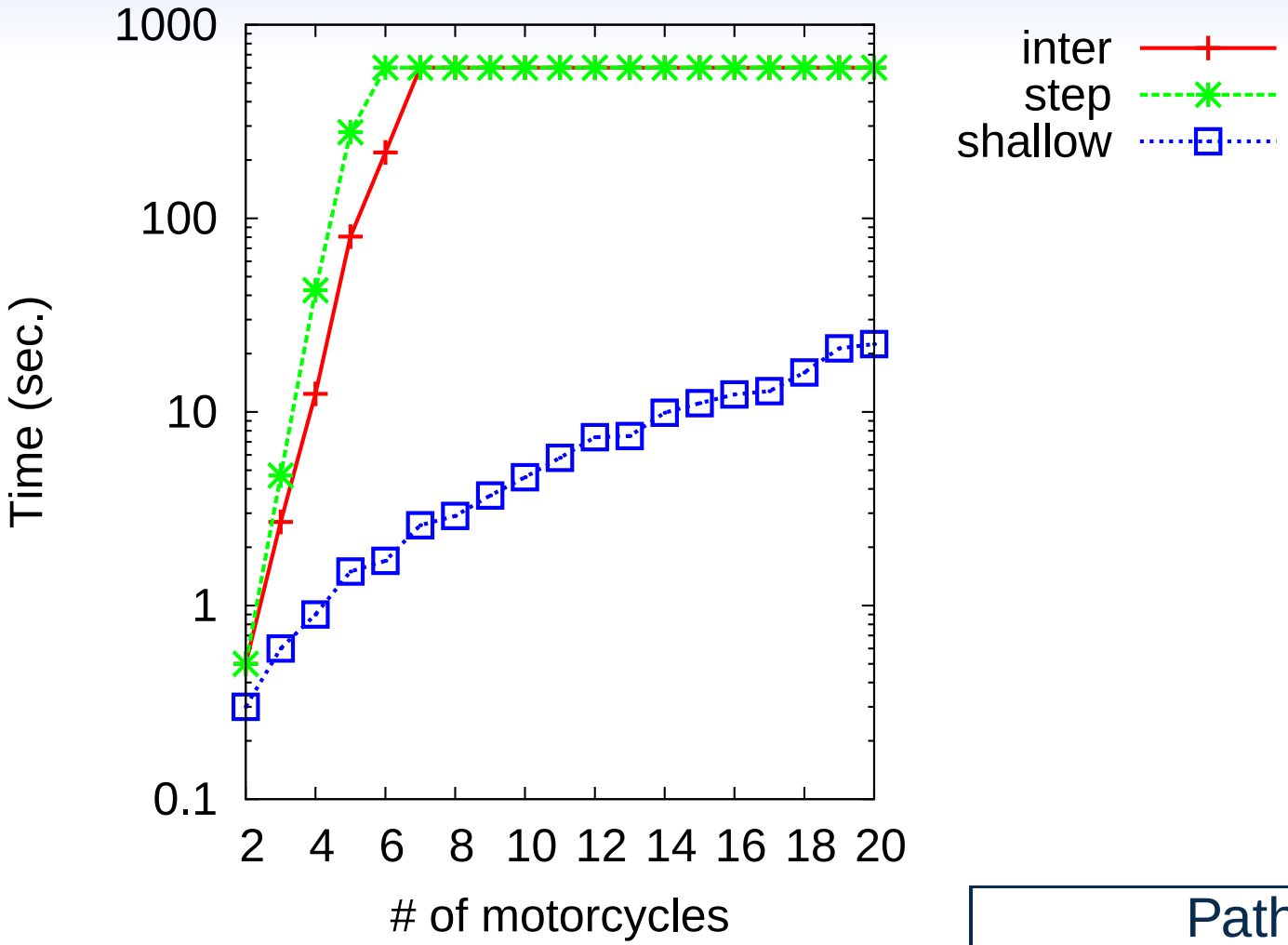
Path length		
Inter	Step	Shallow
5*n*	6	6

Ring-shape Fischer



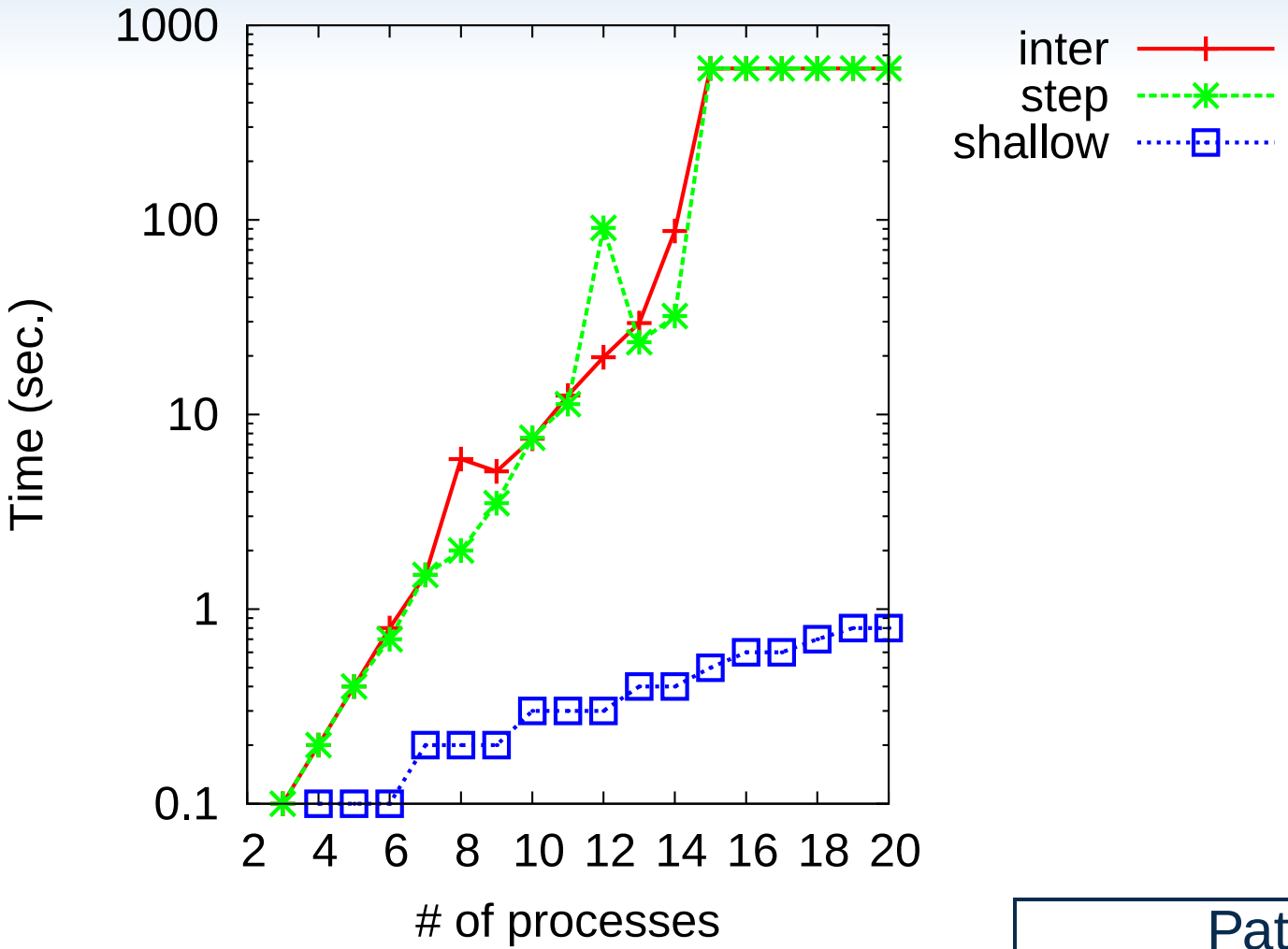
Path length		
Inter	Step	Shallow
$7n$	7	7

Motorcycle



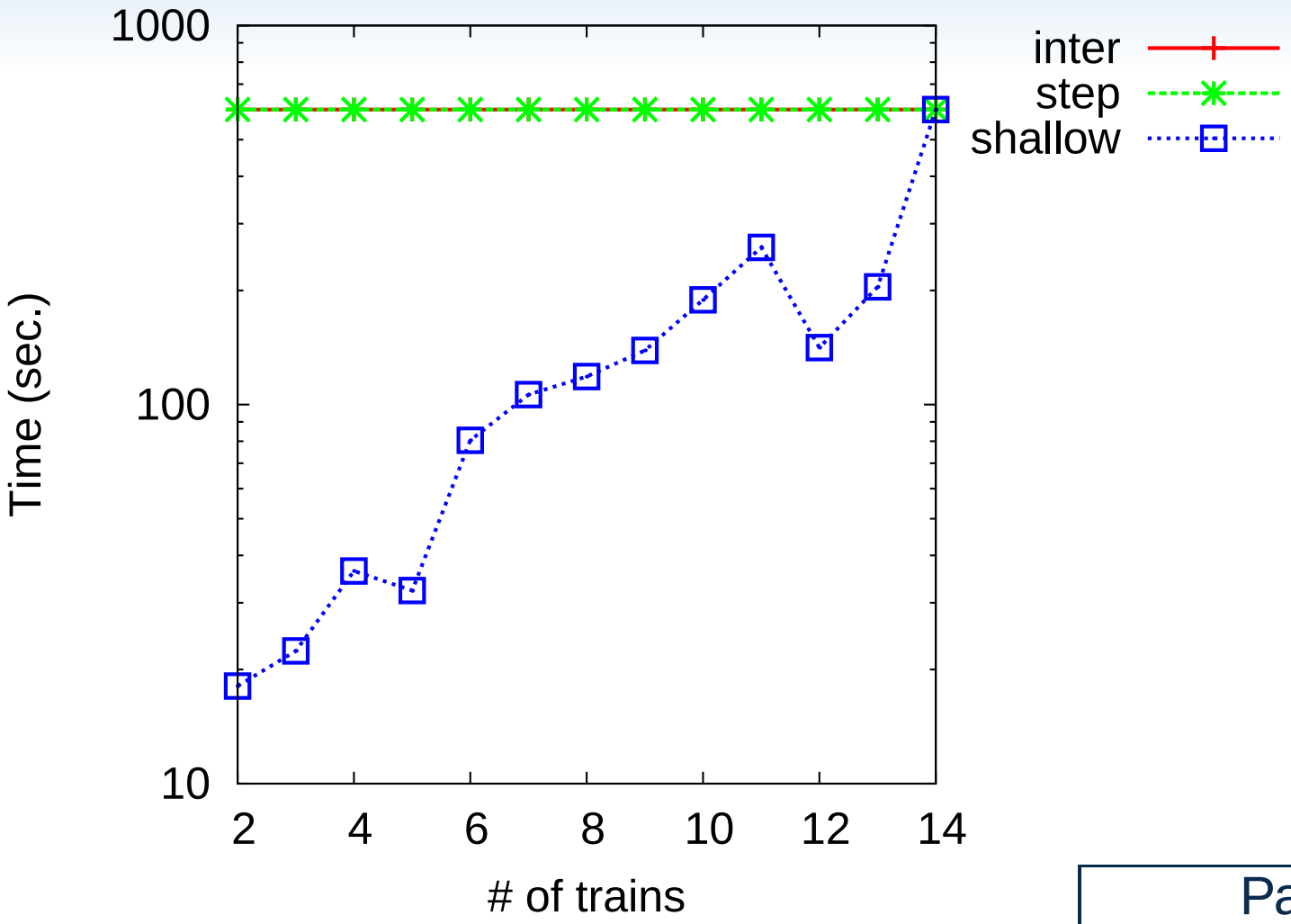
Path length		
Inter	Step	Shallow
$4n + 3$	$4n + 3$	7..9

FDDI



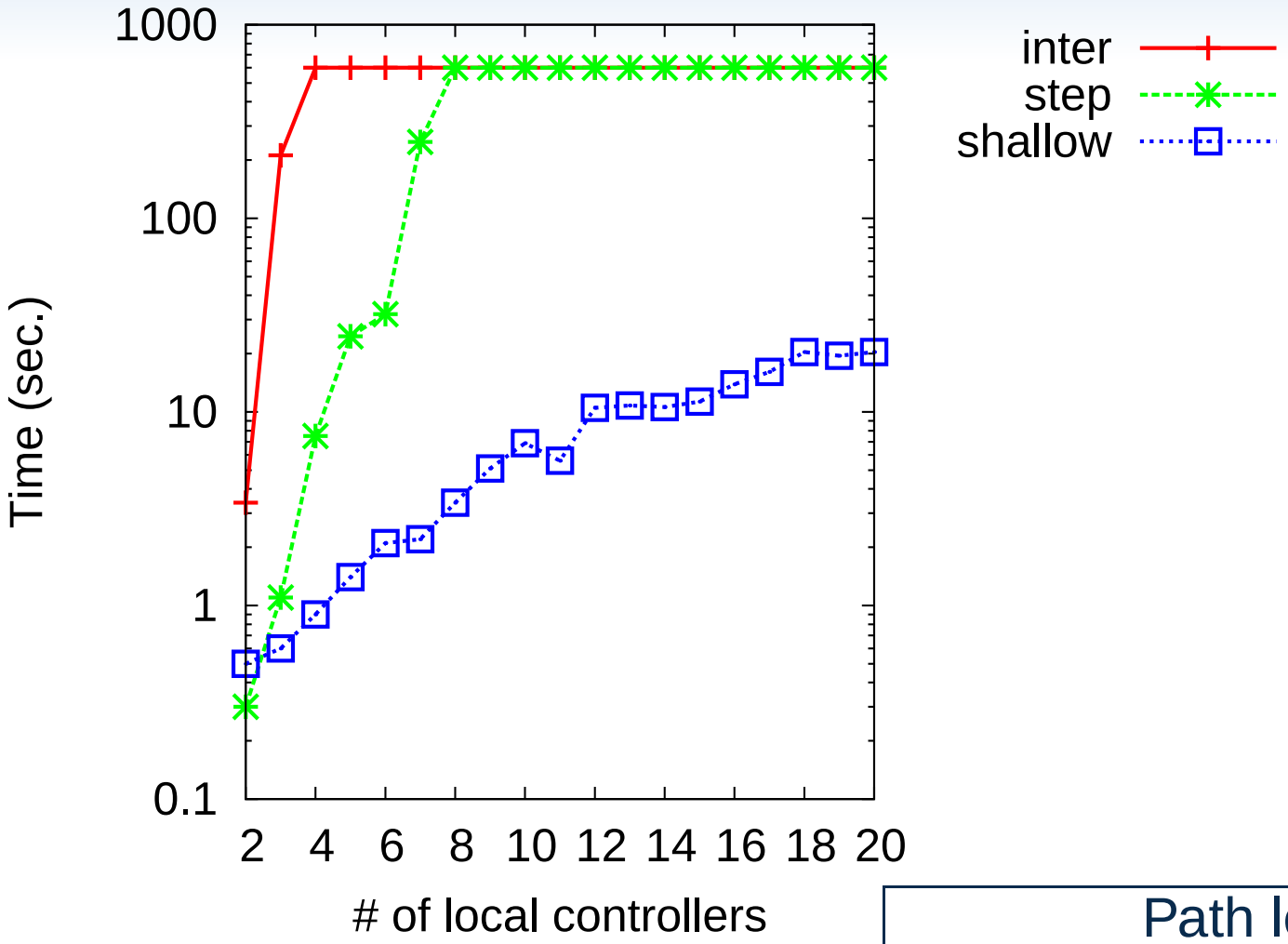
Path length		
Inter	Step	Shallow
$2n+1$	5	3..5

ETCS



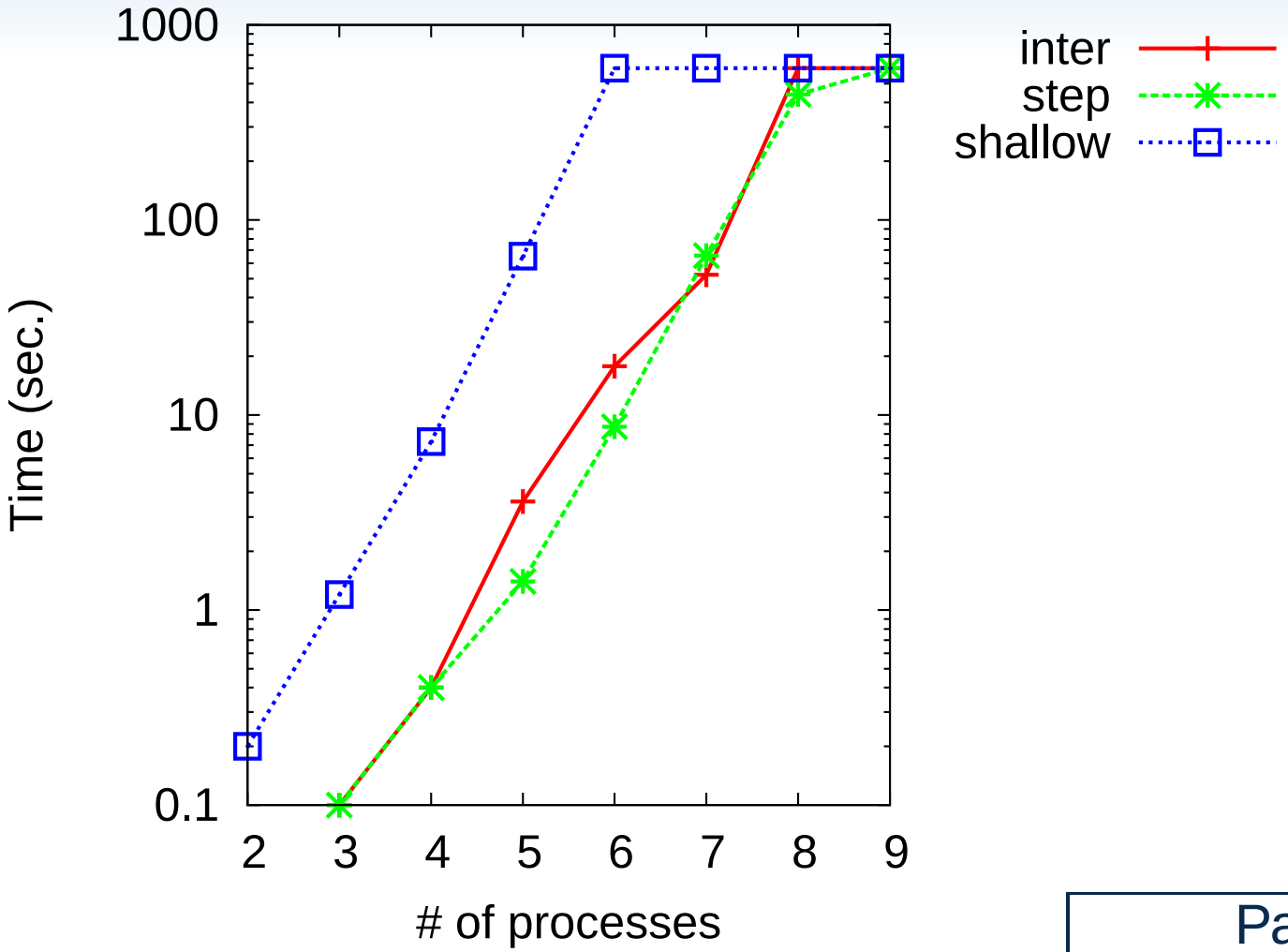
Path length		
Inter	Step	Shallow
NA	NA	17

Multi-Frequency



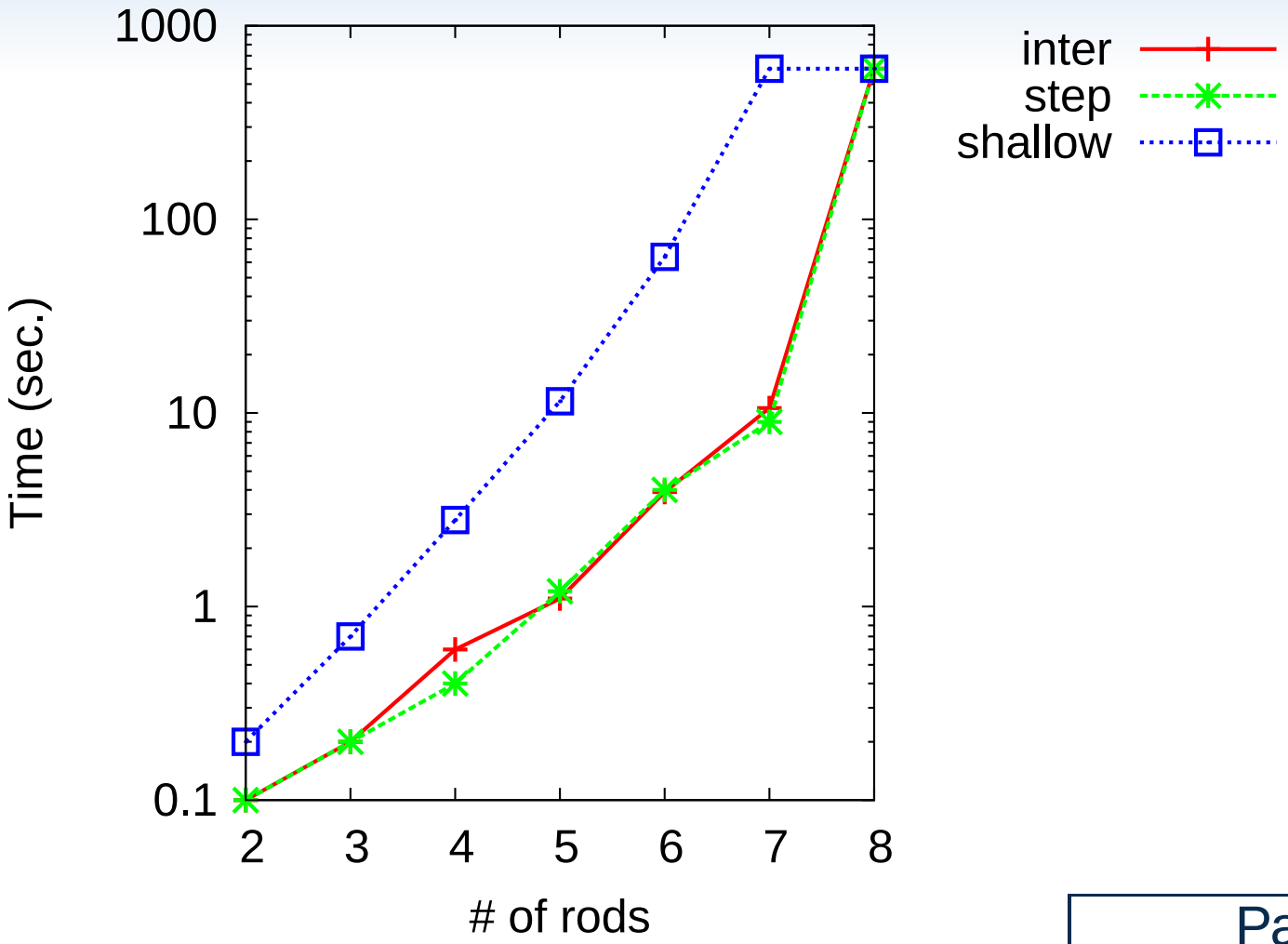
Path length		
Inter	Step	Shallow
NA	$3(n-1)..3n$	9

Star-shape Fischer



Path length		
Inter	Step	Shallow
$3n$	$3n$	$3n$

Nuclear reactor



Path length		
Inter	Step	Shallow
$4n$	$4n$	$4n$

Hybrid Systems in HyCOMP

nuXmv:

- Model checker based on:
  - the NuSMV model checker
  - the MathSat SMT solver.
- Modeling of symbolic (finite- and infinite-state) transition systems
- Functional verification, requirement analysis
- safety analysis (e.g. fault-trees and FMEA tables generation), diagnosability.

Hybrid Systems in HyCOMP:

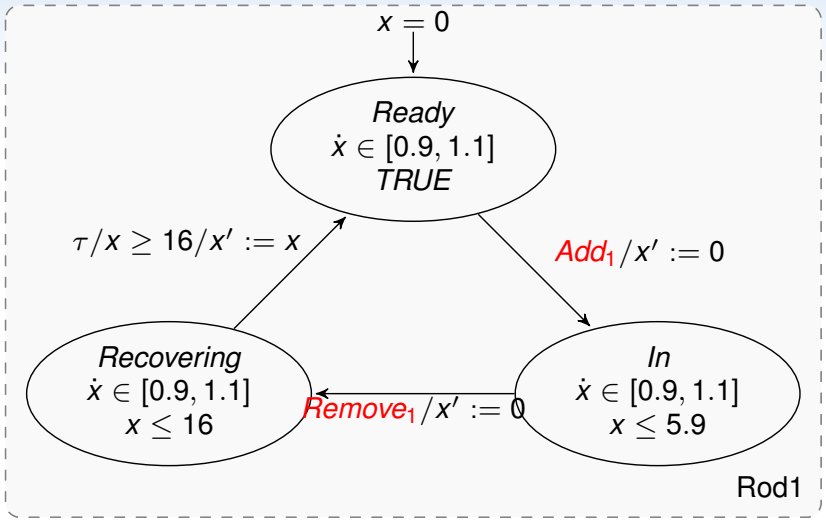
- HyDI language.
- Automatic translation from HyDI to the nuXmv language (i.e. symbolic transition systems).
- Scenario-verification.

HyDI: (HYbrid systems with Discrete Interaction)

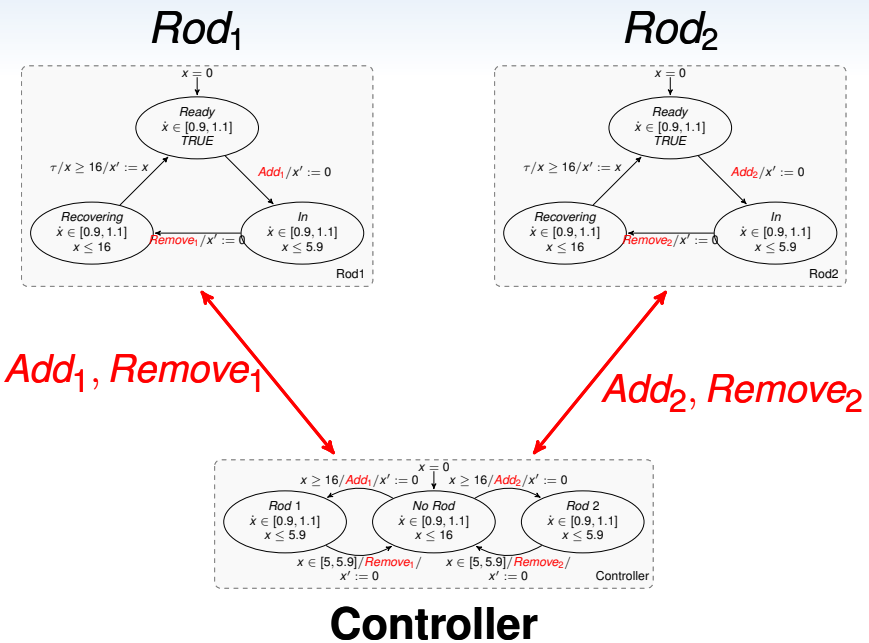
- Symbolic representation of HAN.
- nuXmv language can represent:
  - Discrete systems.
  - Synchronous components.
- HyDI language is extended to represent [SEAA11]:
  - *Continuous features:*
    - *Continuous variables*:
      ```
      VAR x : continuous;
      ```
    - *Flow condition*.
      ```
      FLOW location = Ready -> der(x) <= 1.1;
      ```
  - *Network of automata:*
    - *Instantiation of the automata of the network*.
      ```
      MODULE main
      VAR rod1 : Rod;
      ```
    - *Synchronization constraints:*
      ```
      SYNC rod1, controller EVENT add, add1;
      ```

Hybrid Automata in HyDI

```
MODULE RodType
VAR
location : {Ready,Recovering,In};
x : continuous;
EVENT tau,Remove,Add;
-- Initial conditions
INIT location = Ready & x=0;
INVAR -- Invariants
(location = In -> x<=5.9) &
(location = Recovering -> x<=16)
-- Flow conditions
FLOW 0.9<=der(x) & der(x)<=1.1
TRANS -- Discrete transitions
case
  EVENT=Add : location=Ready & next(location)=In & next(x)=0;
  EVENT=Remove : location=In & next(location)=Recovering &
                 next(x)=0;
  EVENT=tau : location=Recovering & next(location)=Ready &
              next(x)=x;
esac;
```



Hybrid Automata Network in HyDI



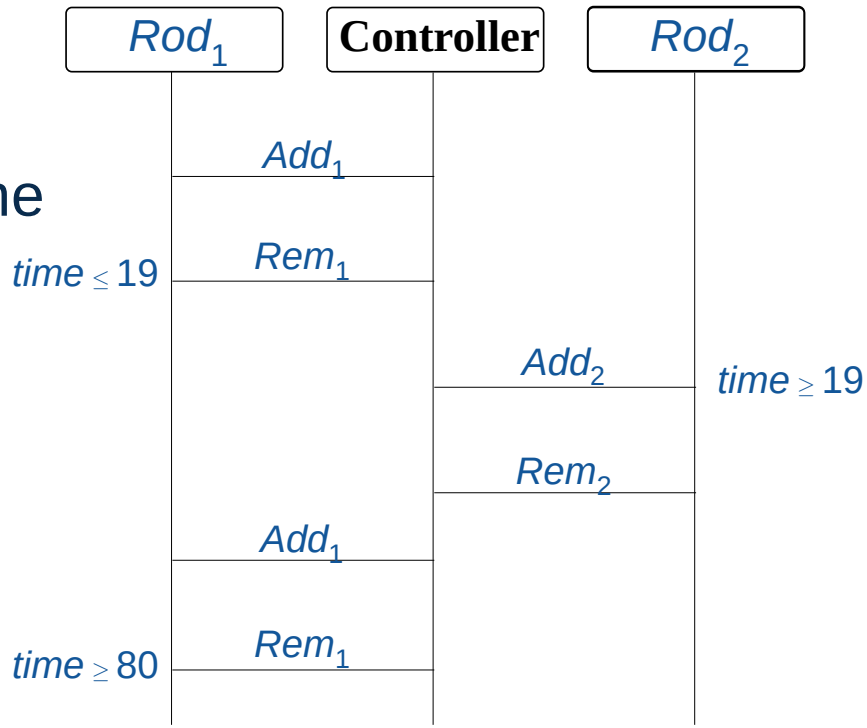
Controller

```
-- Instantiation of the automata of the network
VAR rod1 : RodType;
VAR rod2 : RodType;
VAR controller : ControllerType;
-- Pairwise synchronization constraints
SYNC rod1, controller EVENT add, add1;
SYNC rod2, controller EVENT add, add2;
SYNC rod1, controller EVENT remove, remove1;
SYNC rod2, controller EVENT remove, remove2;
```

Outline

Scenario feasibility

$\langle m, \varphi \rangle$: Message sequence chart (MSC). m with constraints φ.



m: parallel composition of instances.

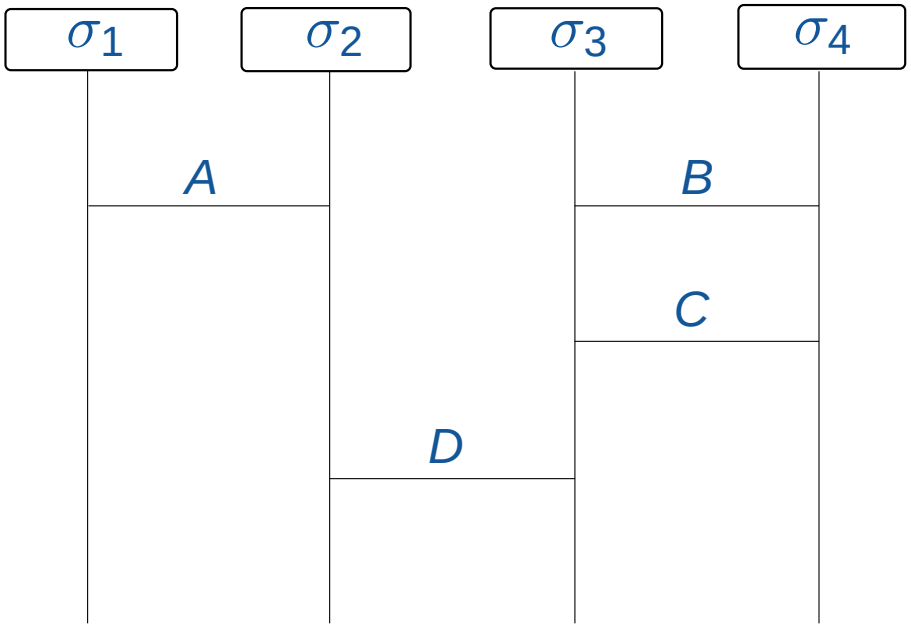
$\varphi = \varphi_g \wedge \varphi_1 \wedge \ldots \wedge \varphi_n$: formulas over the network variables on synchronization.

- Global (φ_g): over all the network variables.
- Local φ_i: over variable of H_i.

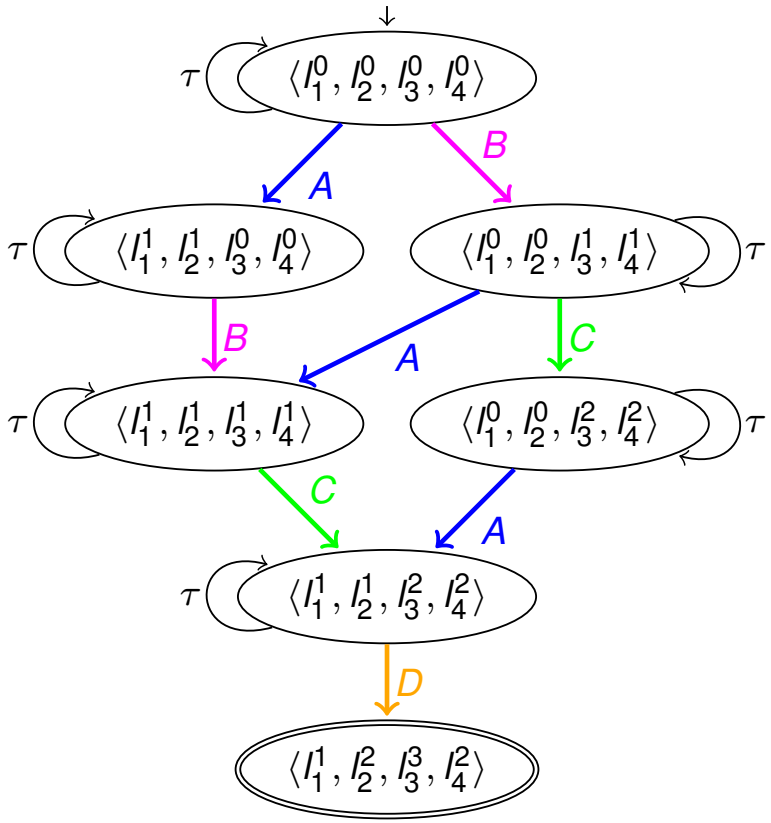
MSC feasibility: the MSC $\langle m, \varphi \rangle$ is feasible in the network $\mathcal{H}$ iff there exist a run in $\mathcal{H}$ compatible with $\langle m, \varphi \rangle$.

MSC verification via reachability

- The MSC is translated in a monitor automaton S_m.
- The automaton is composed with the network.
- Enables off-the-shelf verification techniques:
  - BMC: feasibility.
  - k-induction: unfeasibility.

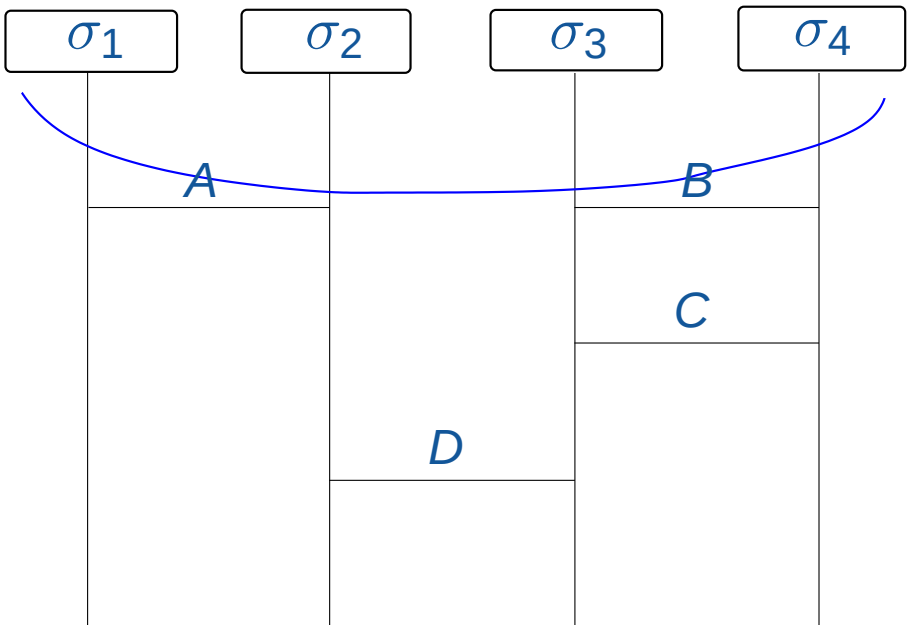


$$m = \sigma_1 || \sigma_2 || \sigma_3 || \sigma_4$$

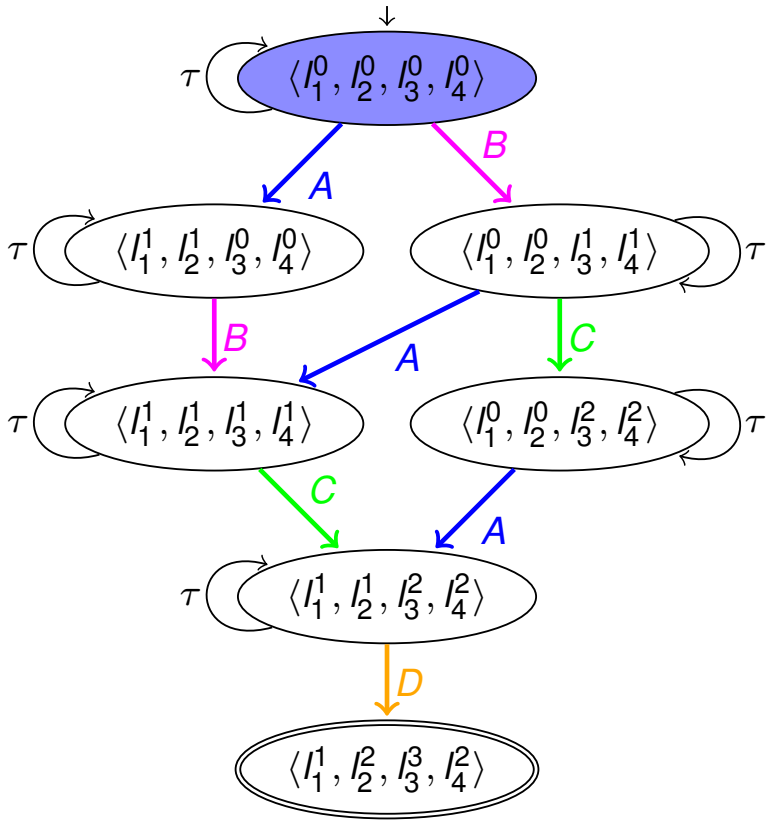


MSC verification via reachability

- The MSC is translated in a monitor automaton S_m.
- The automaton is composed with the network.
- Enables off-the-shelf verification techniques:
  - BMC: feasibility.
  - k-induction: unfeasibility.

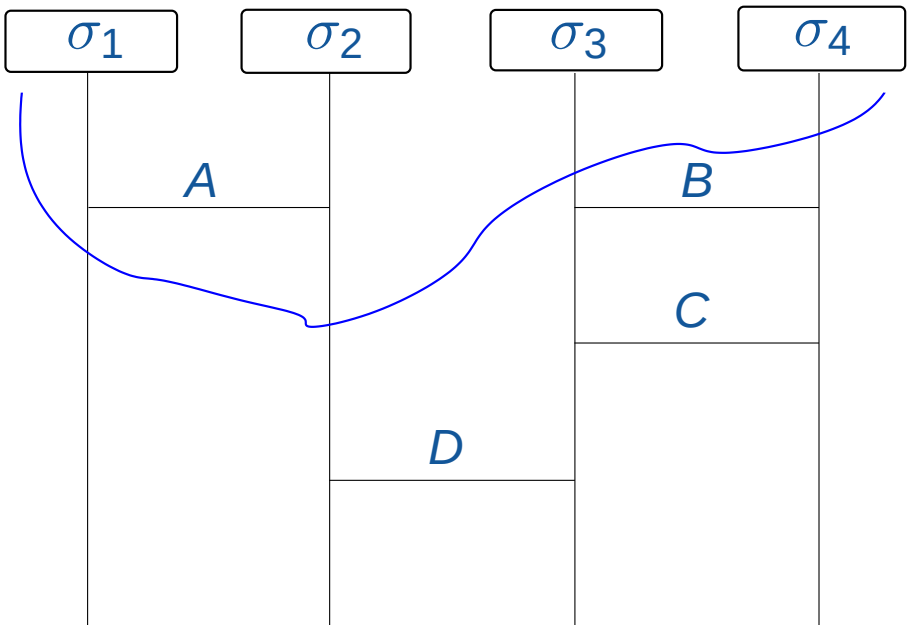


Cut: $\langle l_1^0, l_2^0, l_3^0, l_4^0 \rangle$

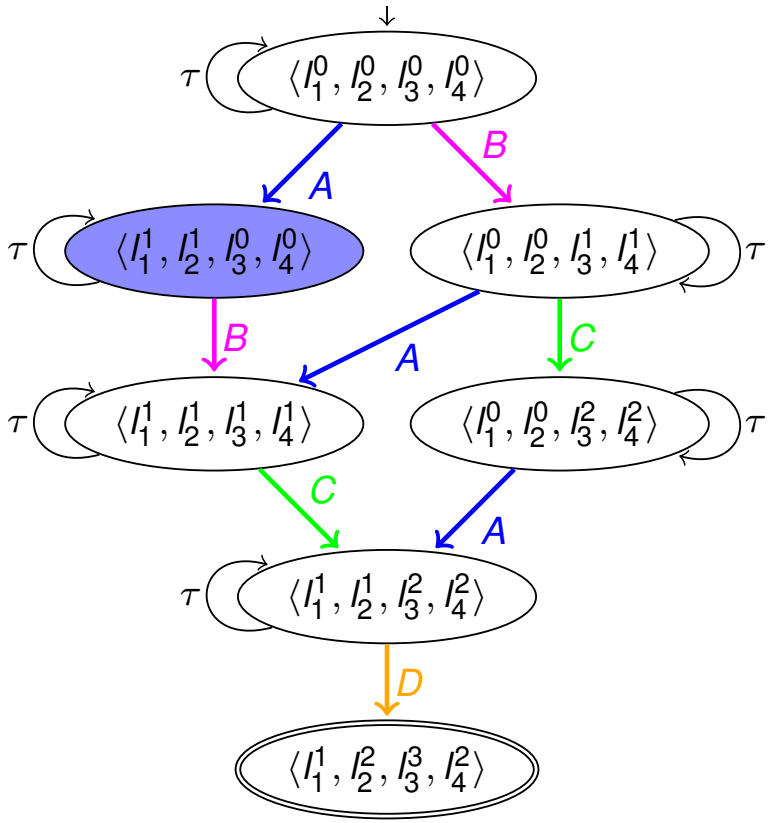


MSC verification via reachability

- The MSC is translated in a monitor automaton S_m.
- The automaton is composed with the network.
- Enables off-the-shelf verification techniques:
  - BMC: feasibility.
  - k-induction: unfeasibility.

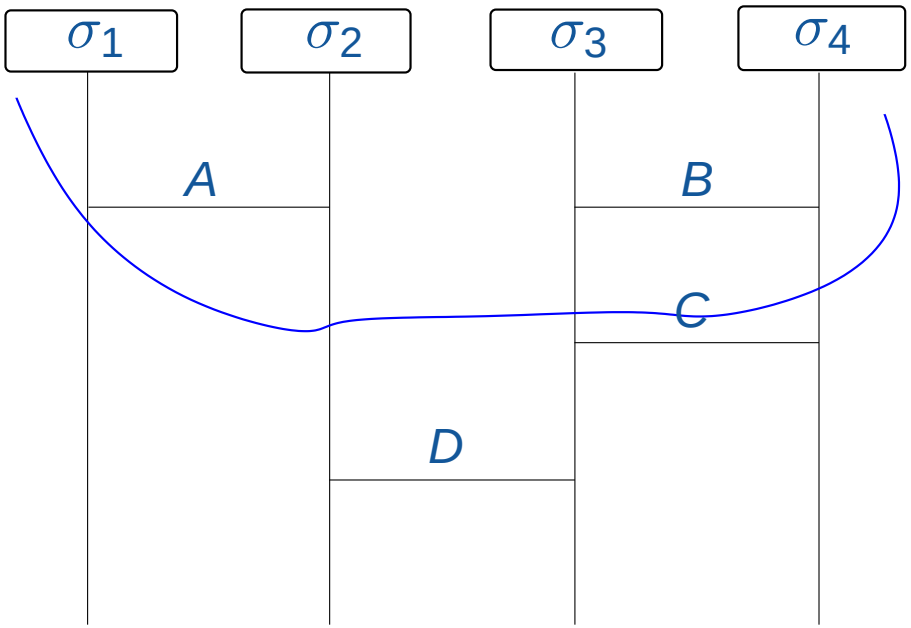


Cut: $\langle l_1^1, l_2^1, l_3^0, l_4^0 \rangle$

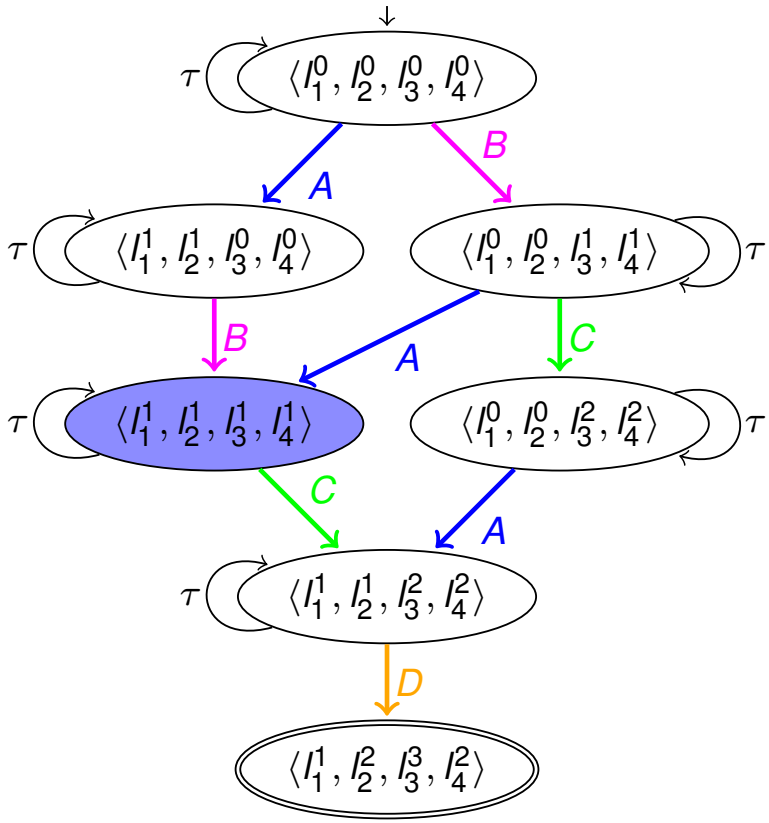


MSC verification via reachability

- The MSC is translated in a monitor automaton S_m.
- The automaton is composed with the network.
- Enables off-the-shelf verification techniques:
  - BMC: feasibility.
  - k-induction: unfeasibility.

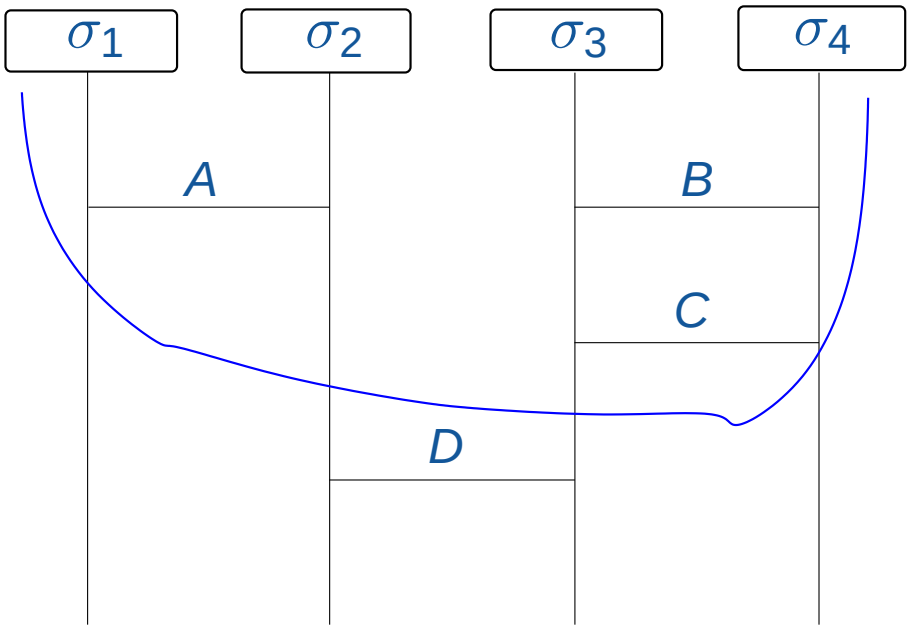


Cut: $\langle l_1^1, l_2^1, l_3^1, l_4^1 \rangle$

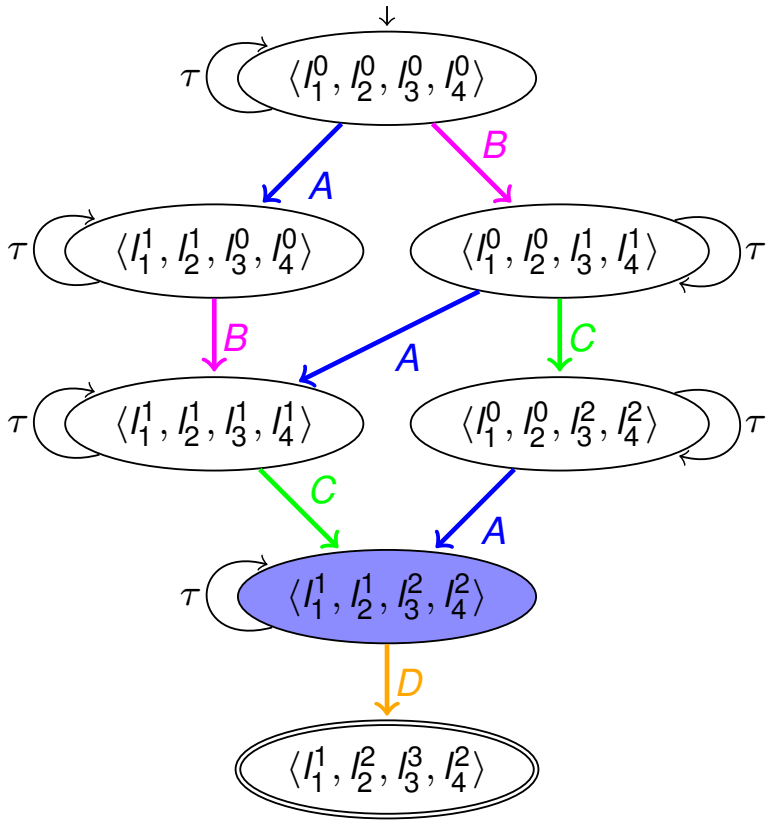


MSC verification via reachability

- The MSC is translated in a monitor automaton S_m.
- The automaton is composed with the network.
- Enables off-the-shelf verification techniques:
  - BMC: feasibility.
  - k-induction: unfeasibility.

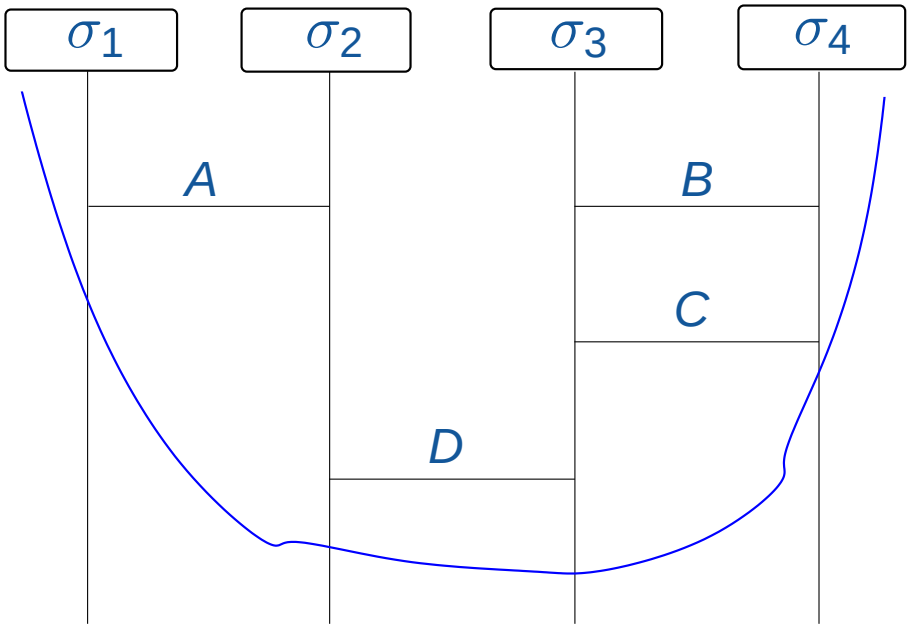


Cut: $\langle l_1^1, l_2^1, l_3^2, l_4^2 \rangle$

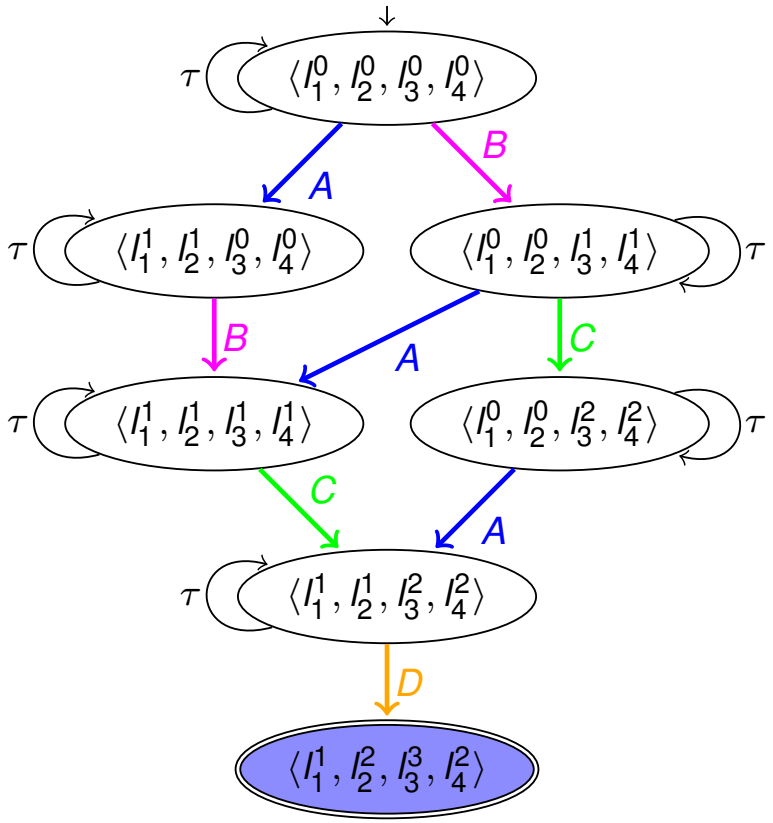


MSC verification via reachability

- The MSC is translated in a monitor automaton S_m.
- The automaton is composed with the network.
- Enables off-the-shelf verification techniques:
  - BMC: feasibility.
  - k-induction: unfeasibility.

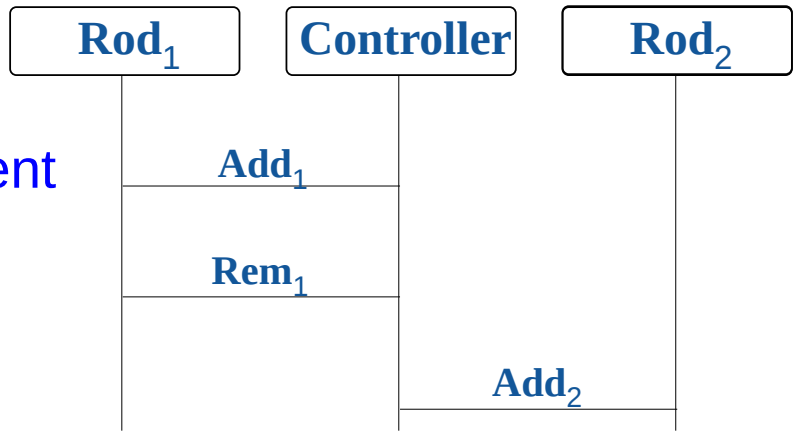


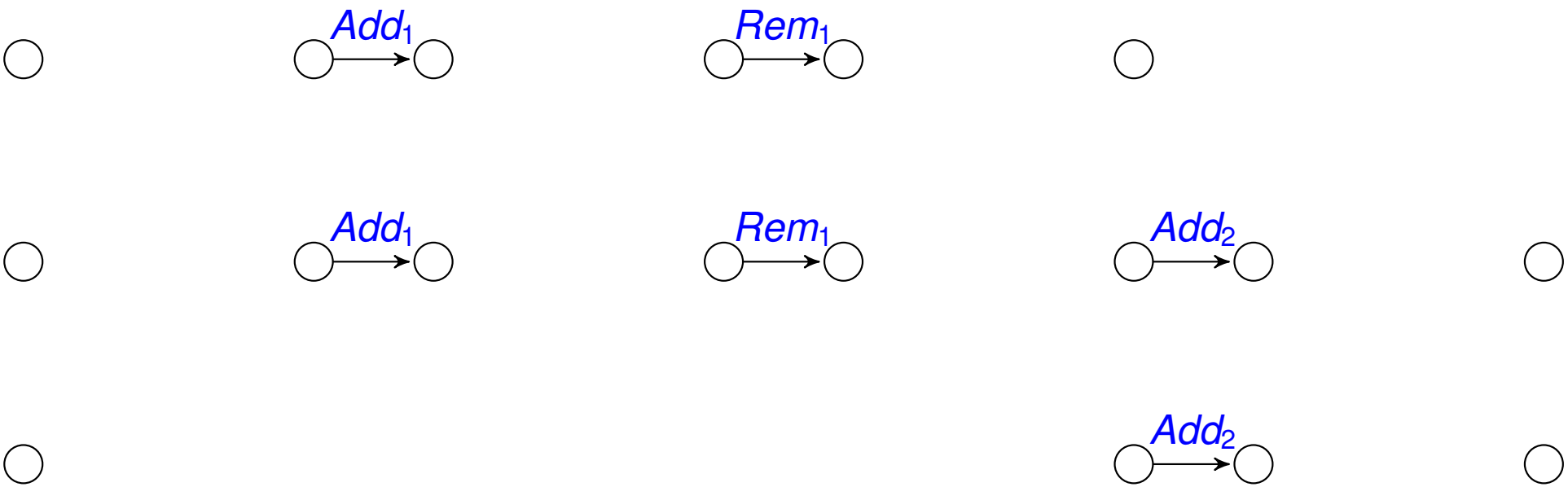
Cut: $\langle l_1^1, l_2^3, l_3^3, l_4^2 \rangle$



Scenario-based encoding

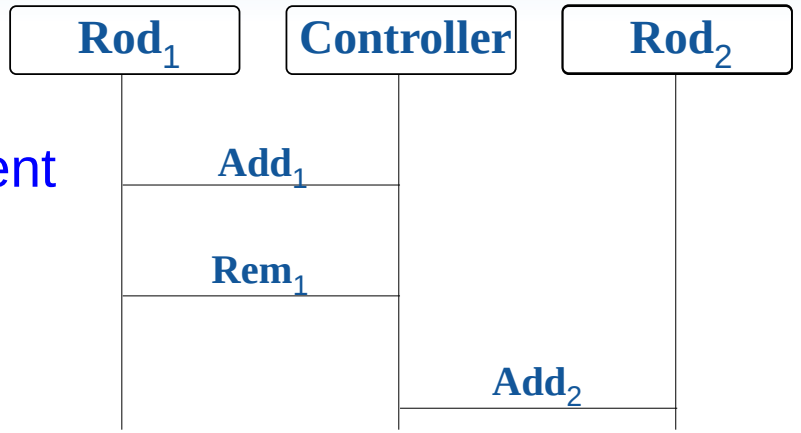
- For all the automata:
  - Fix the position of the shared events.
    transitions are simplified wrt shared event

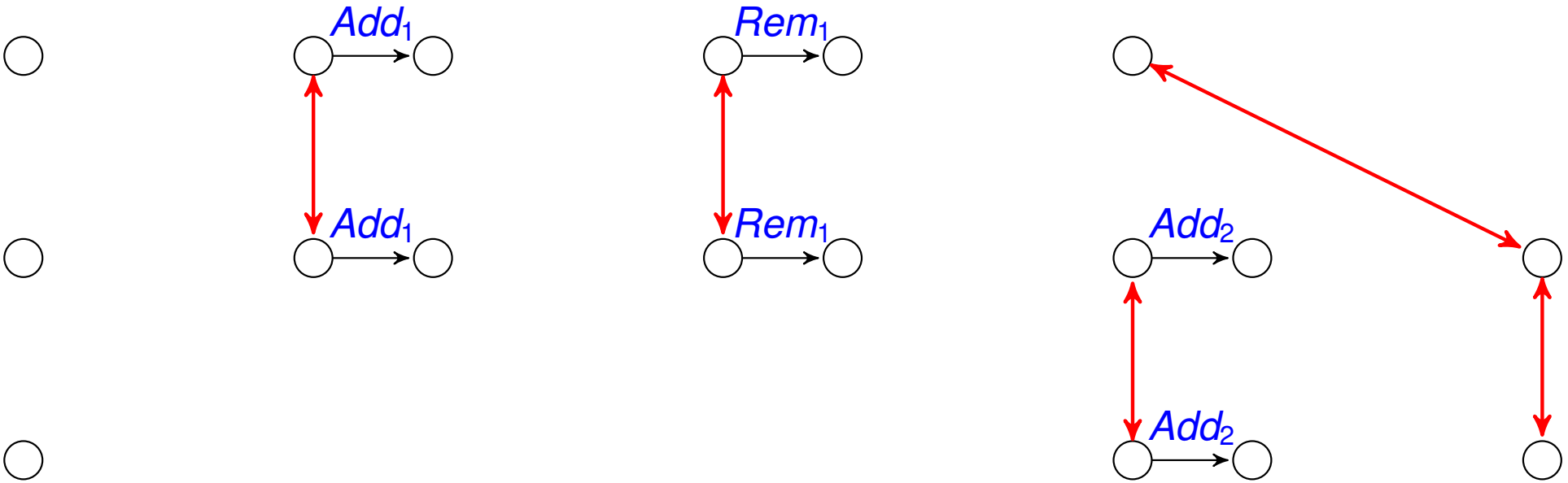




Scenario-based encoding

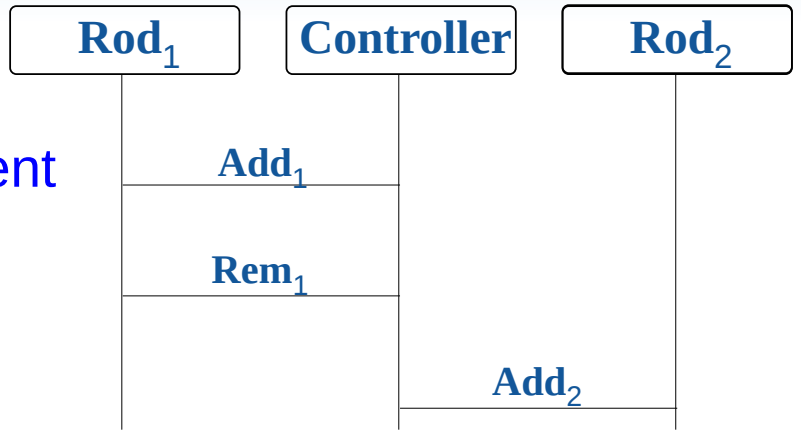
- For all the automata:
  - Fix the position of the shared events.
    transitions are simplified wrt shared event
  - Add the synchronization constraints.

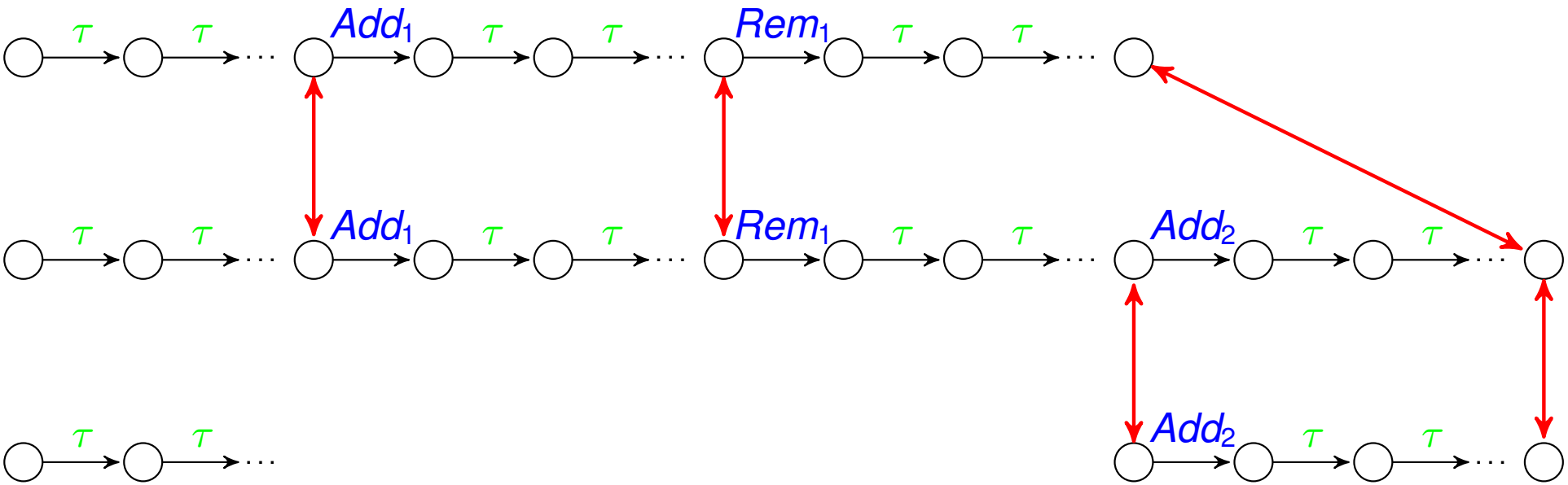




Scenario-based encoding

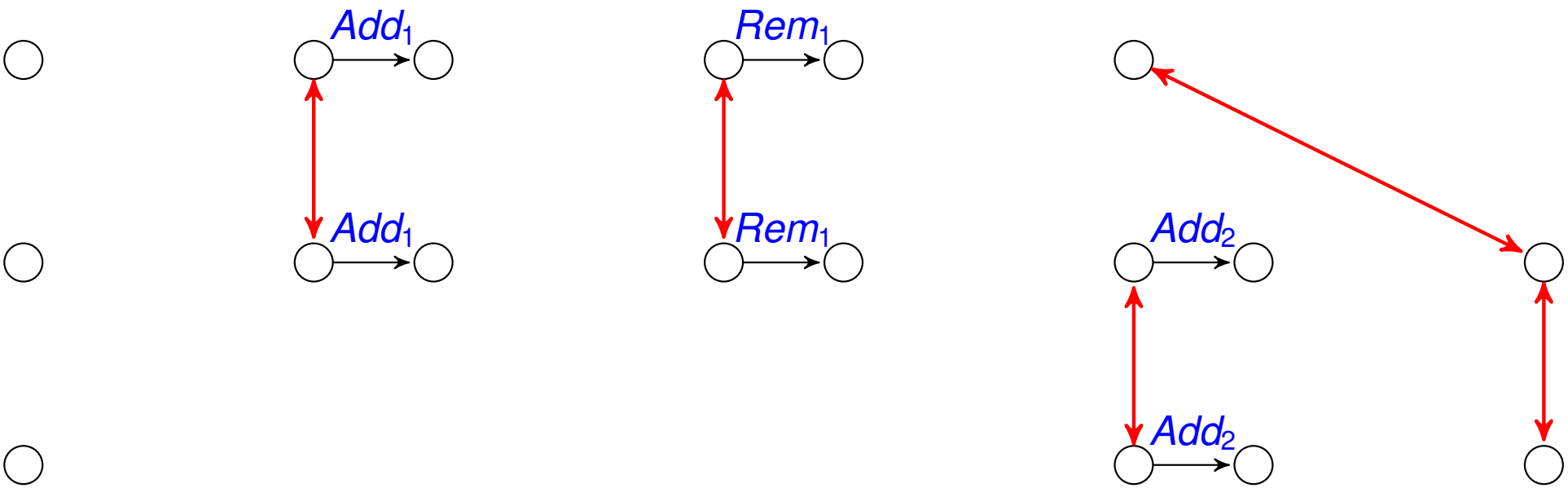
- For all the automata:
  - Fix the position of the shared events.
    transitions are simplified wrt shared event
  - Add the synchronization constraints.
  - Encode the “local segments”.
    transitions are simplified wrt τ





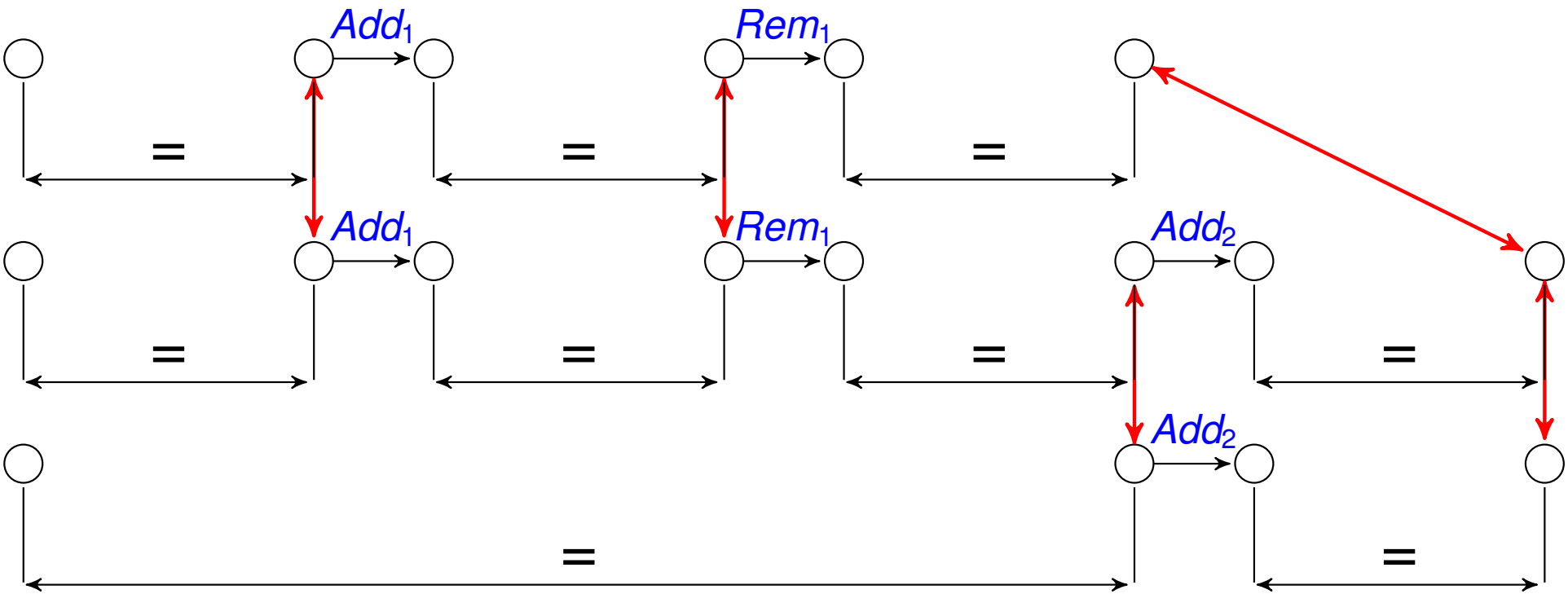
Exploit the solver incrementalty

- Initial encoding of shared events and synchronizations.
- Different pieces are connected via equalities over state variables.
- Local transition are added incrementally.
  - Pop equalities/Assert new local transitions/Push new equalities.



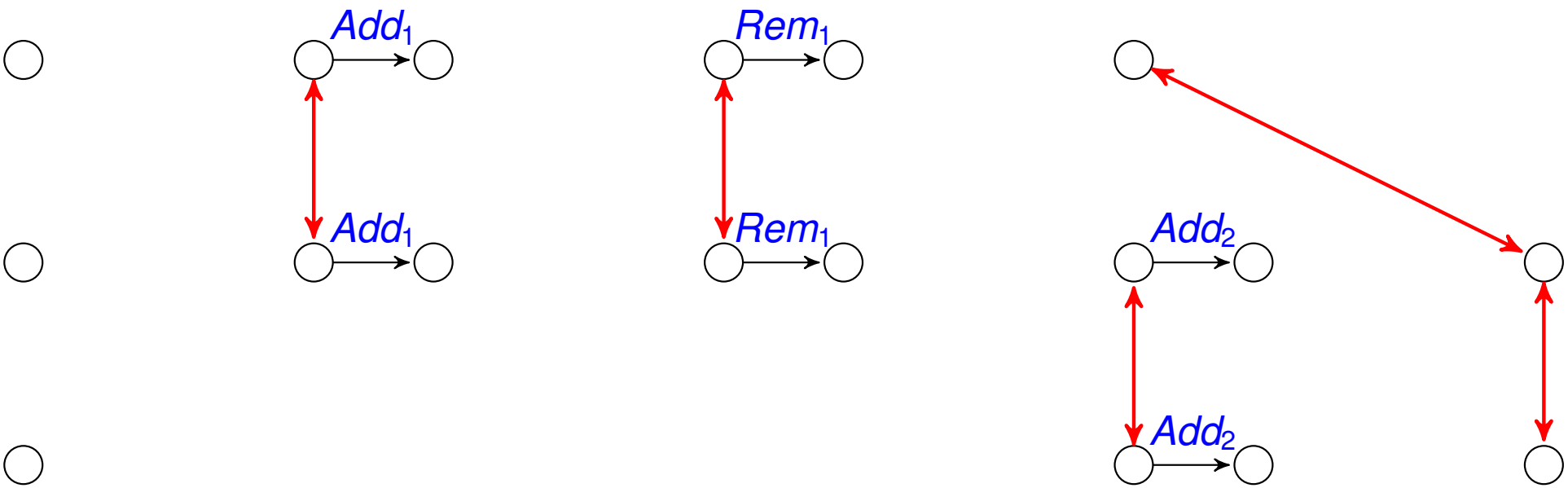
Exploit the solver incrementalty

- Initial encoding of shared events and synchronizations.
- Different pieces are connected via equalities over state variables.
- Local transition are added incrementally.
  - Pop equalities/Assert new local transitions/Push new equalities.



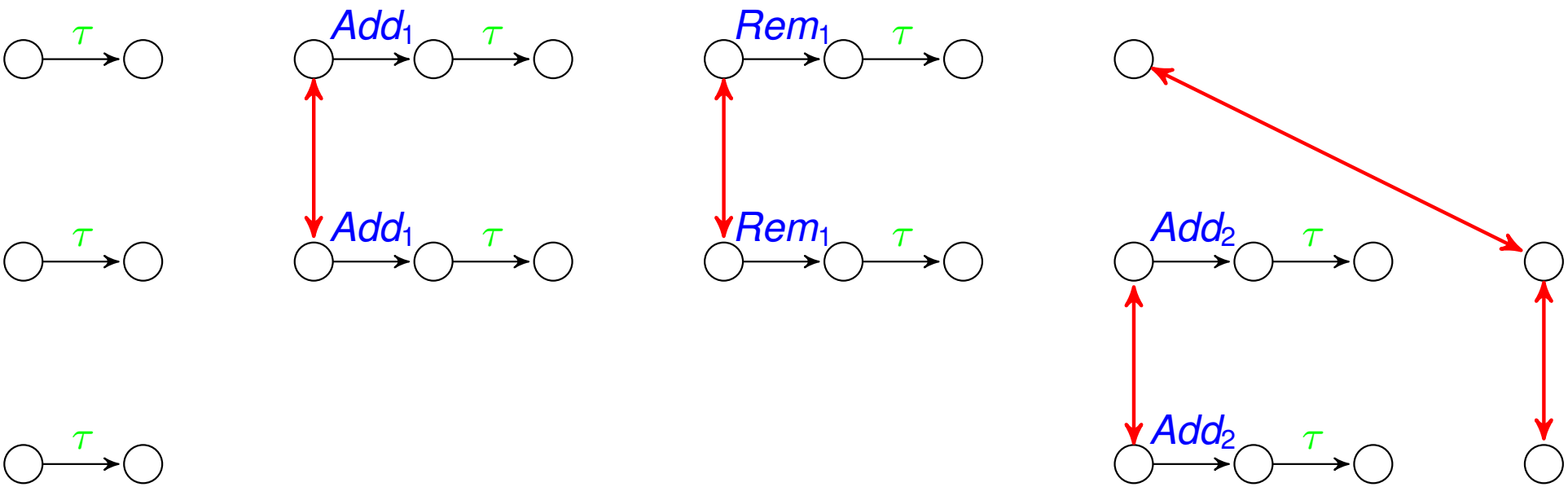
Exploit the solver incrementalty

- Initial encoding of shared events and synchronizations.
- Different pieces are connected via equalities over state variables.
- Local transition are added incrementally.
  - Pop equalities/Assert new local transitions/Push new equalities.



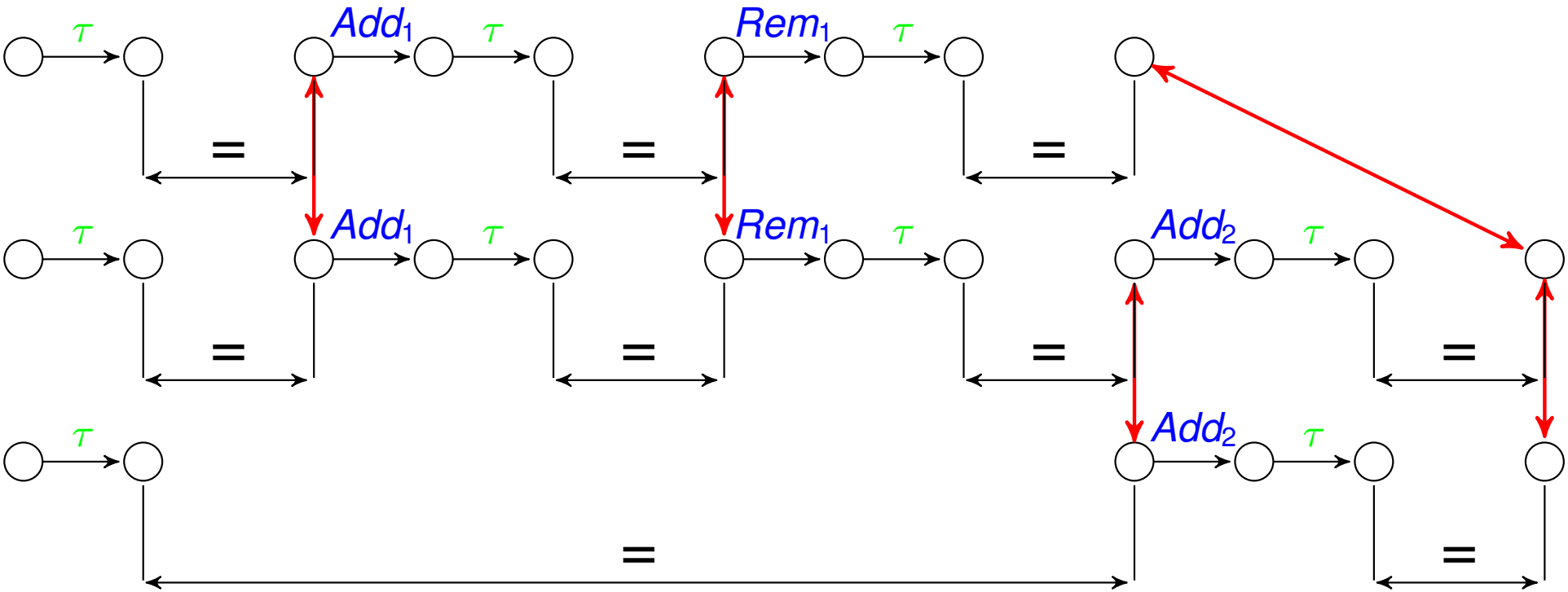
Exploit the solver incrementalty

- Initial encoding of shared events and synchronizations.
- Different pieces are connected via equalities over state variables.
- Local transition are added incrementally.
  - Pop equalities/Assert new local transitions/Push new equalities.



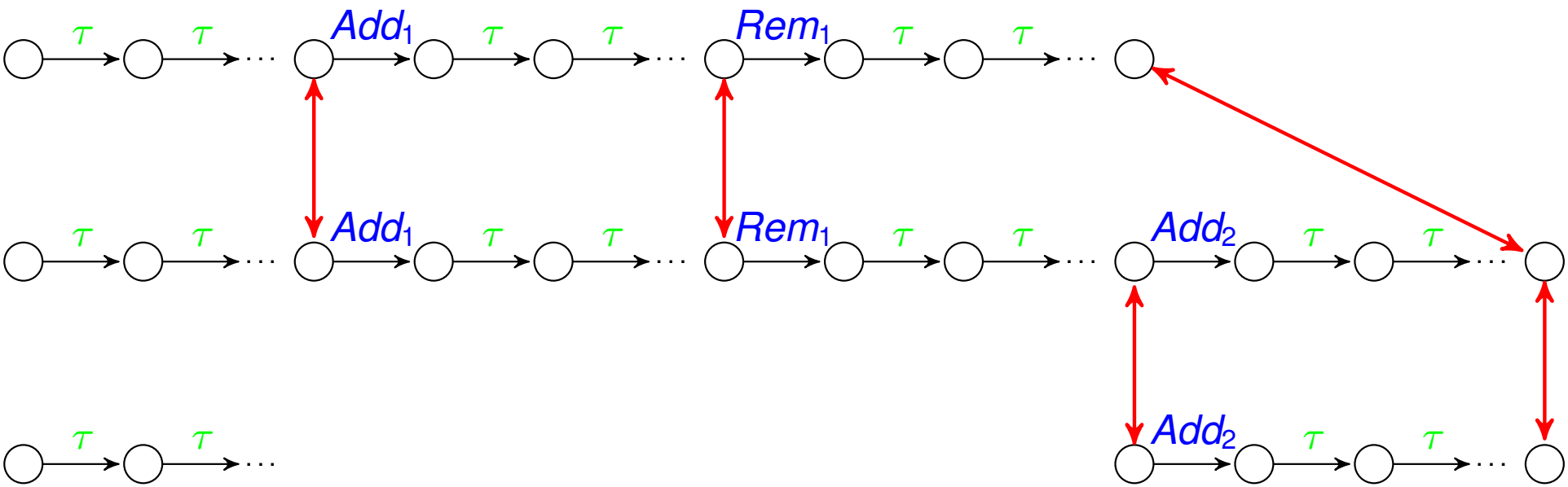
Exploit the solver incrementalty

- Initial encoding of shared events and synchronizations.
- Different pieces are connected via equalities over state variables.
- Local transition are added incrementally.
  - Pop equalities/Assert new local transitions/Push new equalities.



Exploit the solver incrementalty

- Initial encoding of shared events and synchronizations.
- Different pieces are connected via equalities over state variables.
- Local transition are added incrementally.
  - Pop equalities/Assert new local transitions/Push new equalities.



Experimental evaluation

Implementation:

- approach implemented on top of the NuSMV model checker.
- we use the SMT solver MathSat.

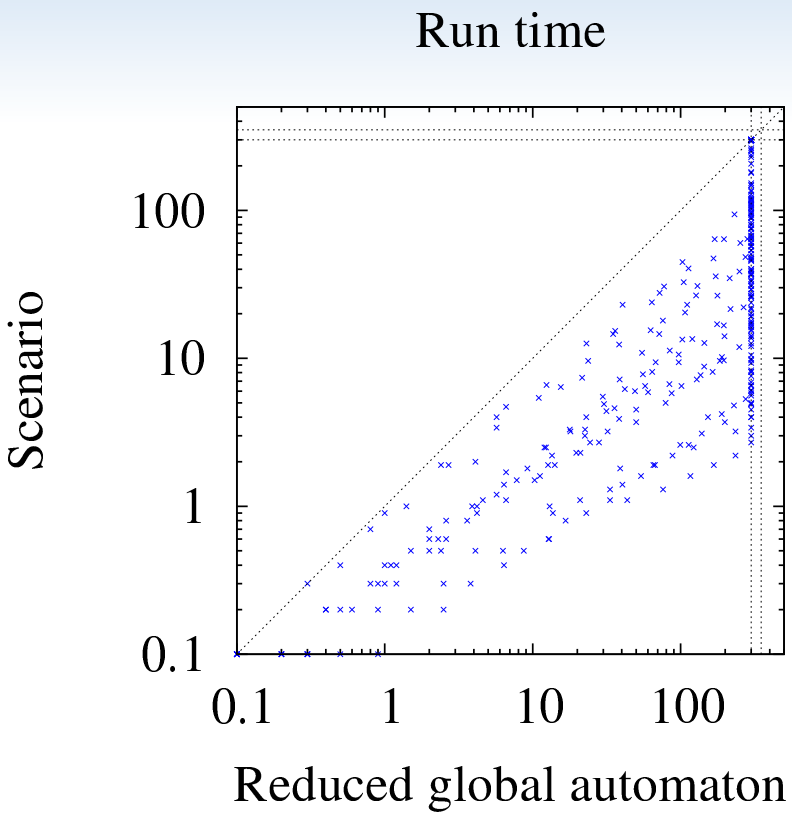
Settings:

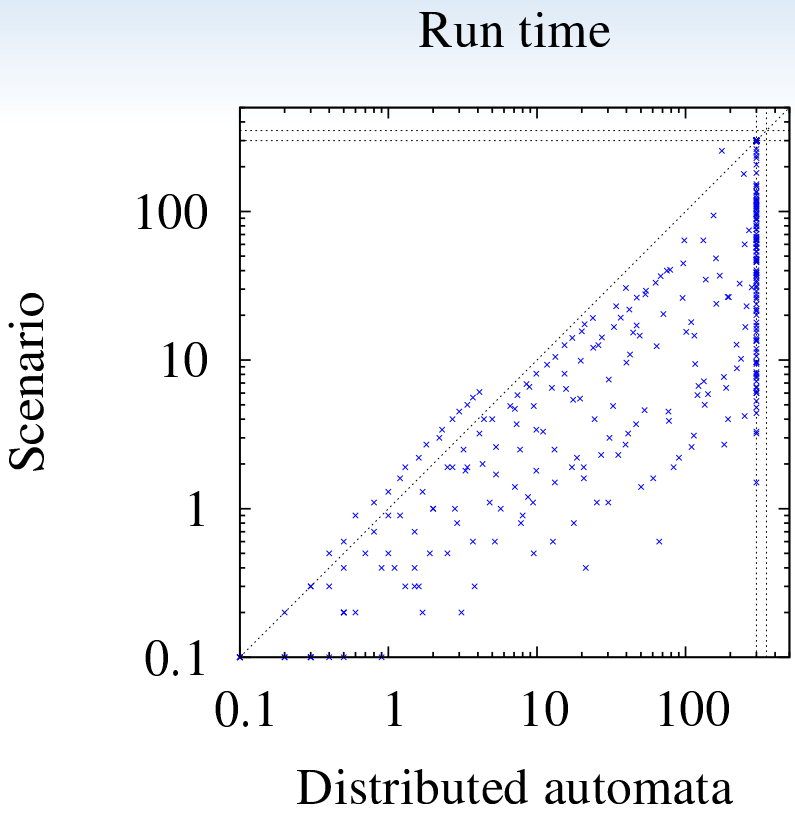
- linear hybrid automata benchmarks.
- several handcrafted MSCs.

Comparision:

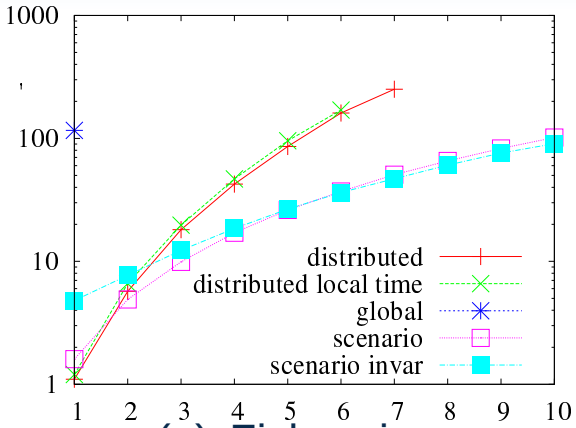
- scenario-based encoding (with and without invariants), reduced automaton, global automaton.
- we scale the dimension of the benchmarks (number of automata, length of the msc).

Scenario vs. automata (running times)

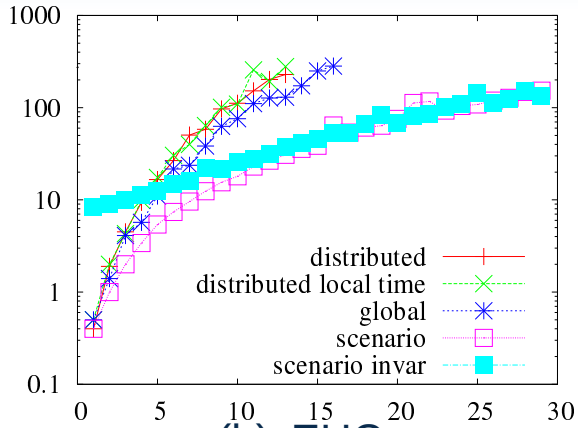




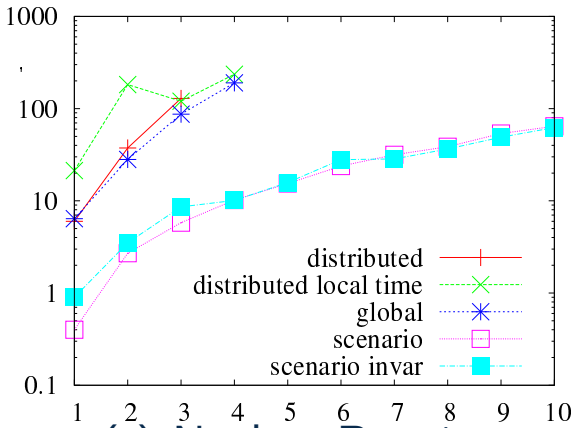
Run times repeating the scenario *n* times



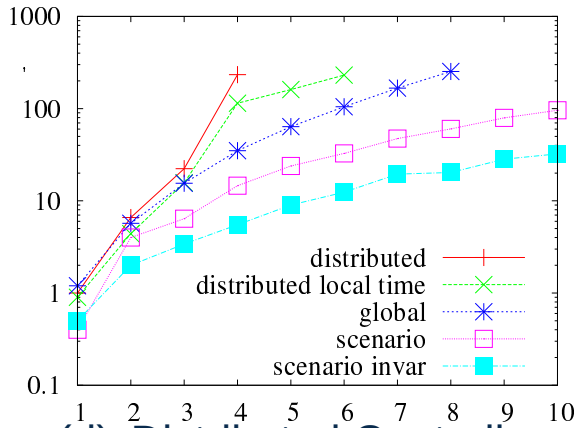
(a) Fisher ring



(b) EHC



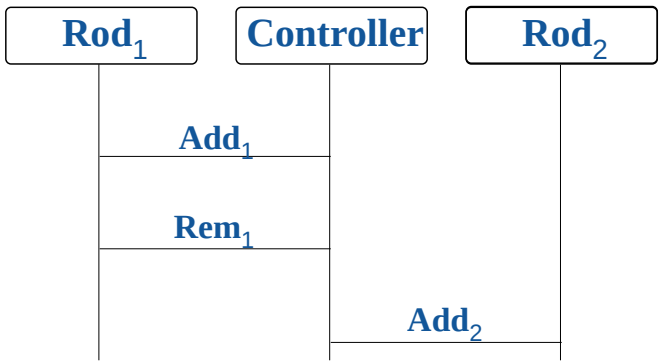
(c) Nuclear Reactor



(d) Distributed Controller

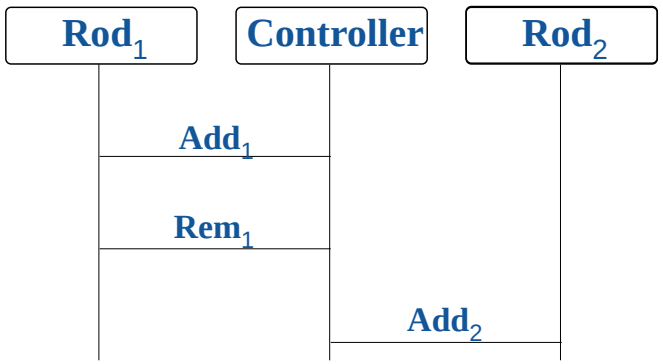
Partitioned K-induction - Algorithm

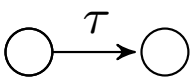
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



Partitioned K-induction - Algorithm

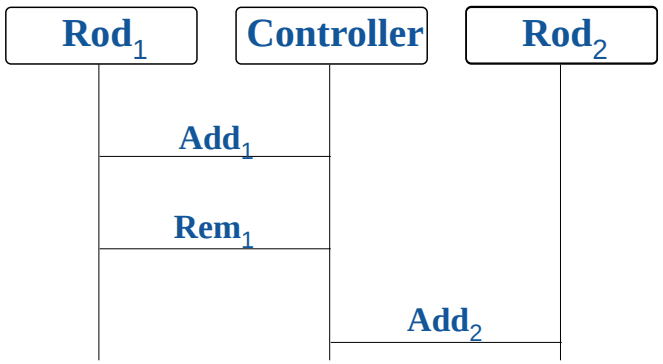
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

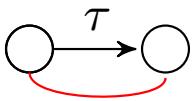




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



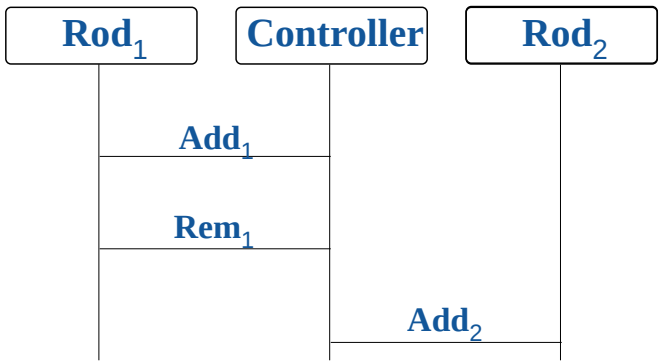


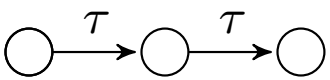
simple path

SAT - new states are reachable

Partitioned K-induction - Algorithm

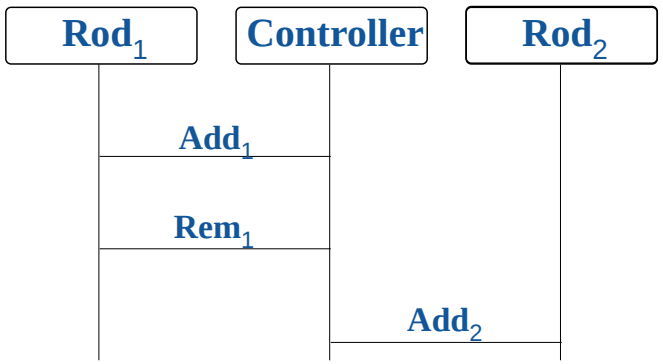
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

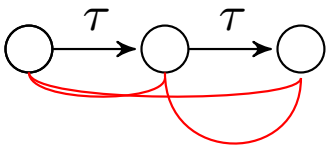




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



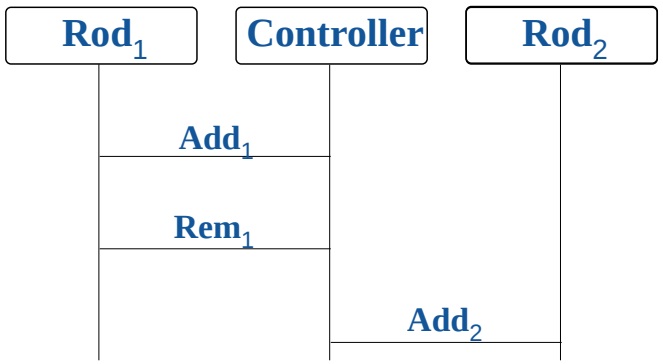


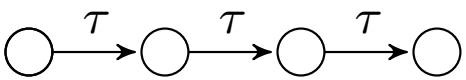
simple path

SAT - new states are reachable

Partitioned K-induction - Algorithm

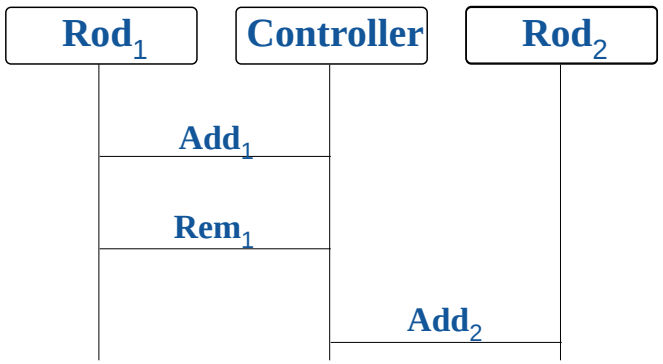
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

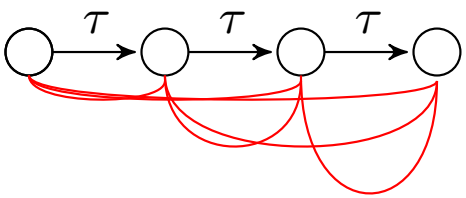




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



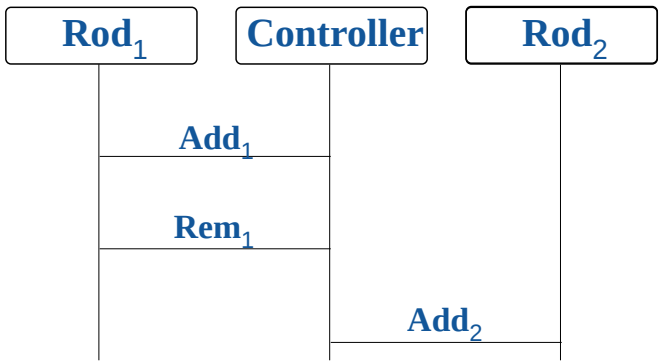


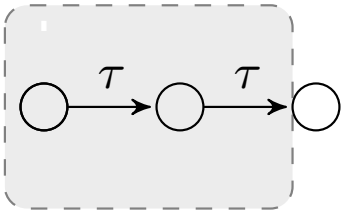
simple path

UNSAT - no new states are reachable

Partitioned K-induction - Algorithm

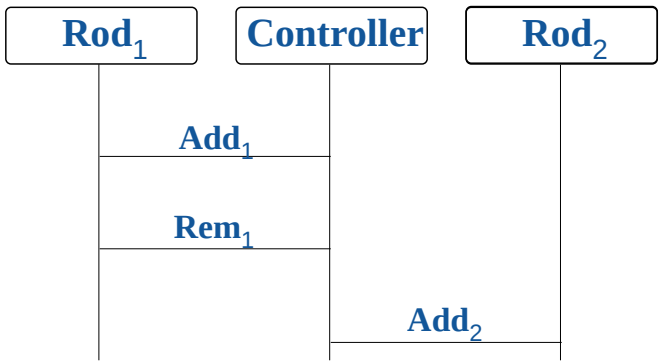
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

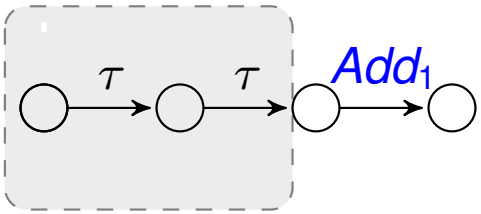




Partitioned K-induction - Algorithm

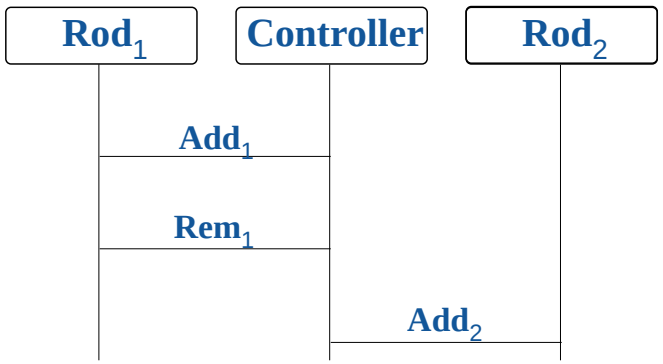
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

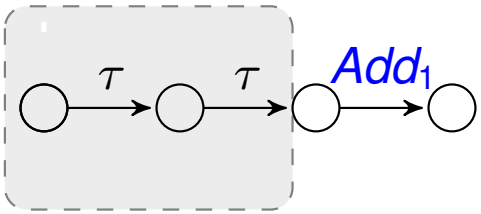


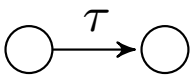


Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

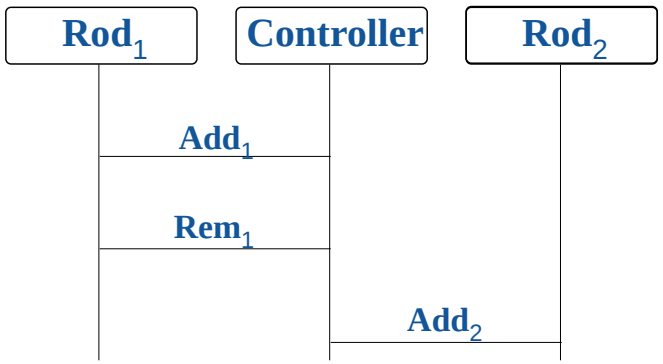


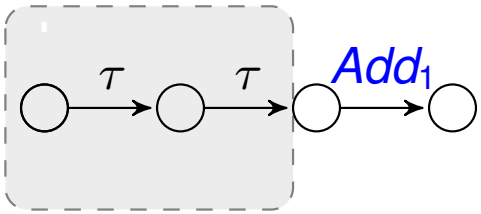


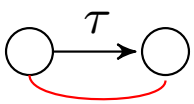


Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.





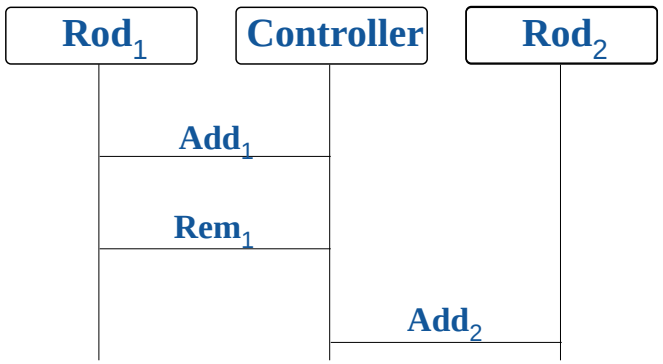


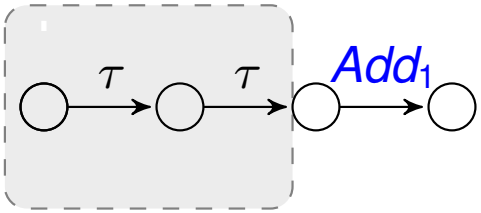
SAT - new states are reachable

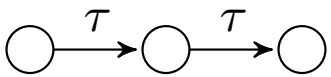
simple path

Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

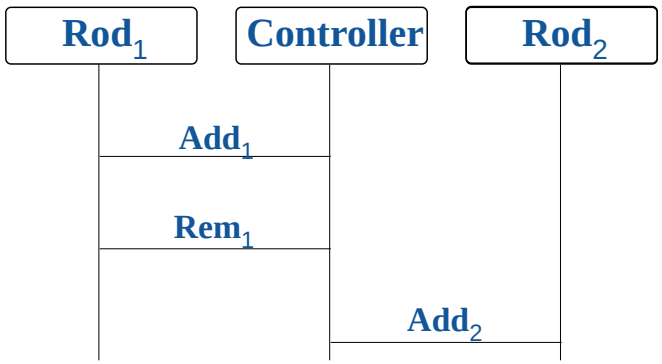


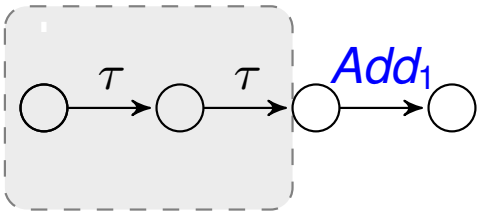


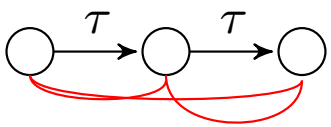


Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.





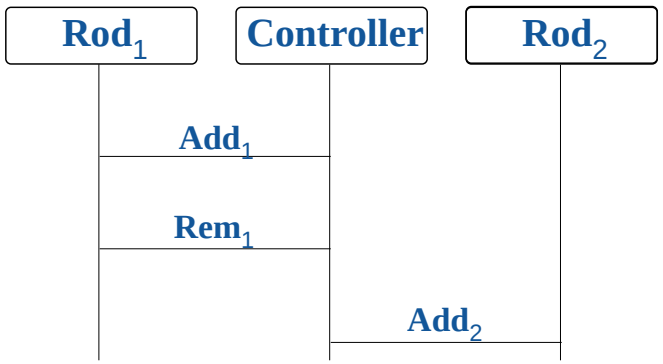


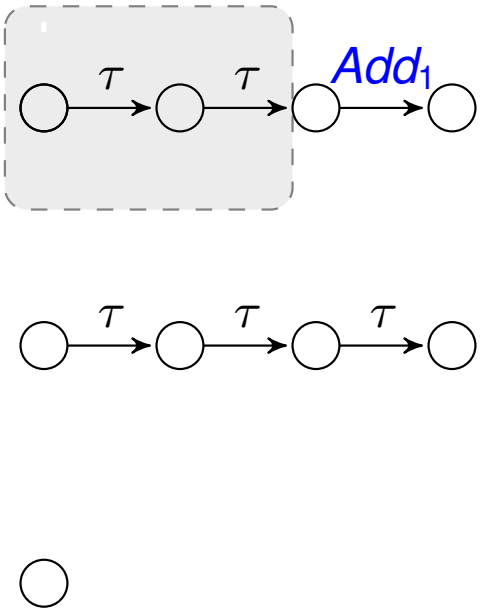
SAT - new states are reachable

simple path

Partitioned K-induction - Algorithm

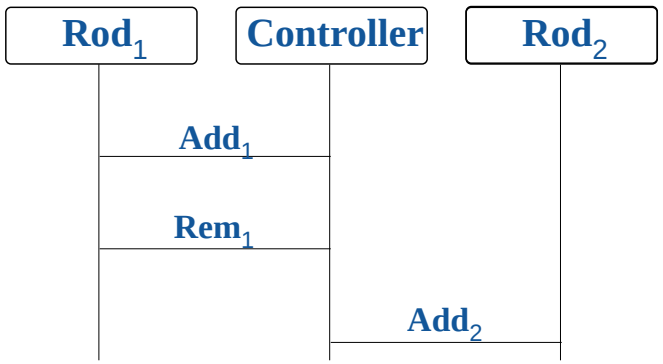
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

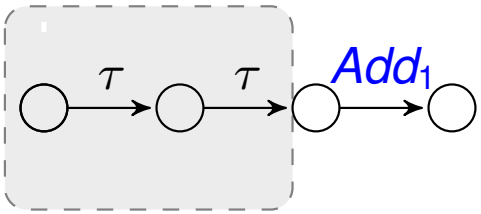


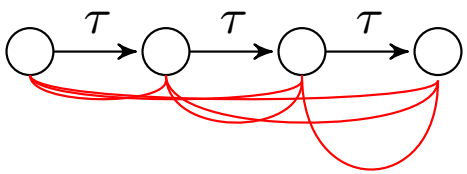


Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.





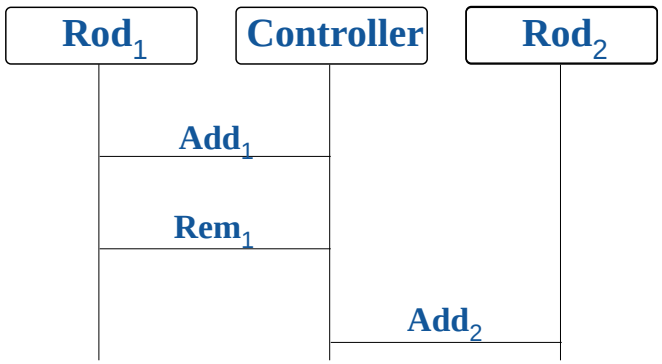


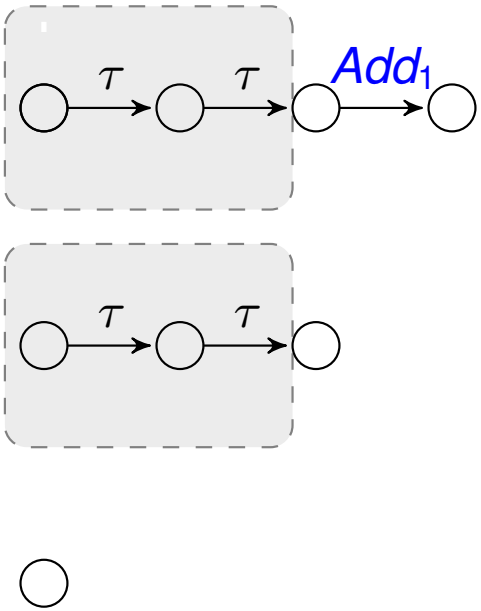
UNSAT - no new states are reachable

simple path

Partitioned K-induction - Algorithm

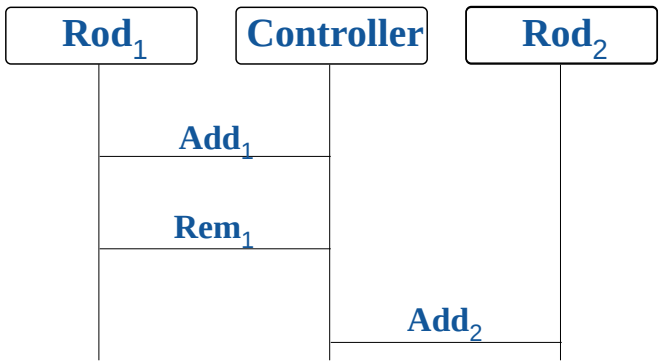
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

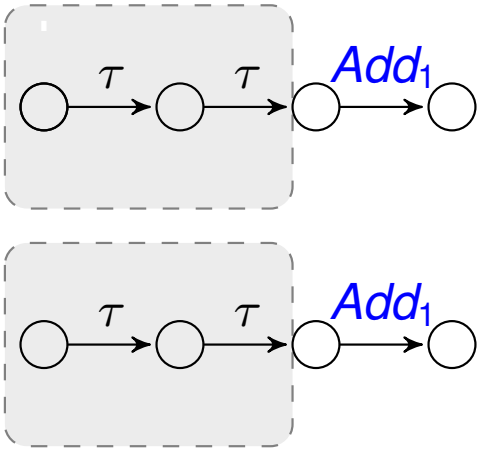




Partitioned K-induction - Algorithm

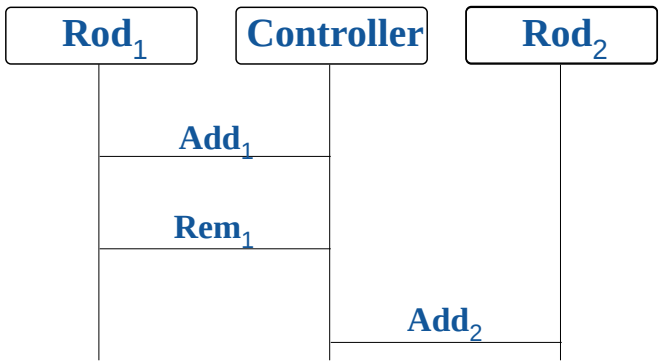
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

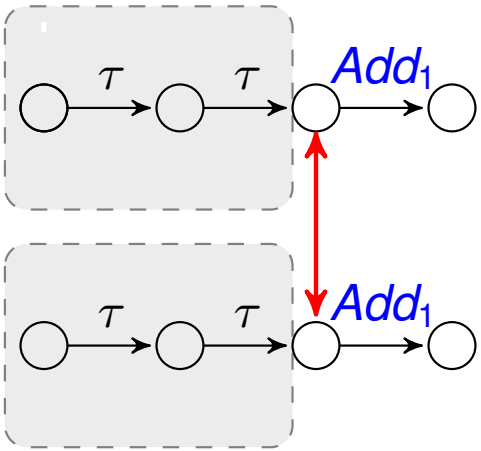




Partitioned K-induction - Algorithm

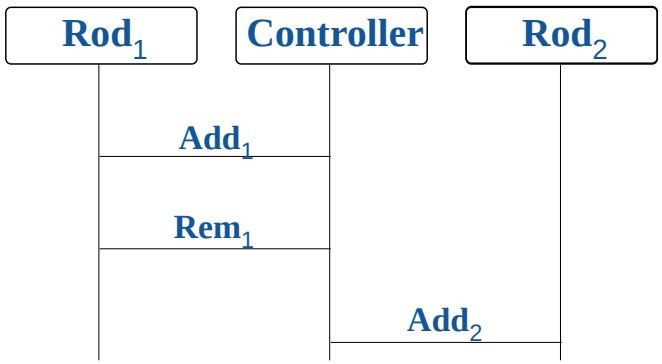
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

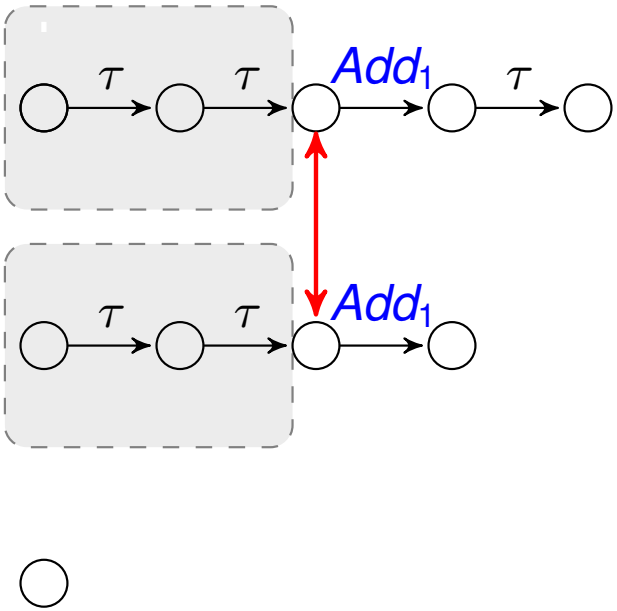




Partitioned K-induction - Algorithm

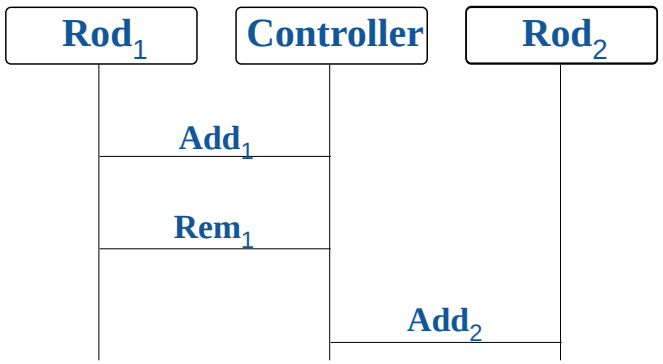
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

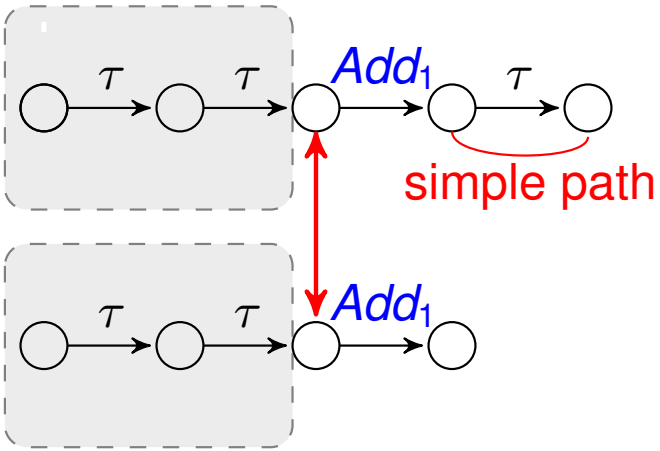




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

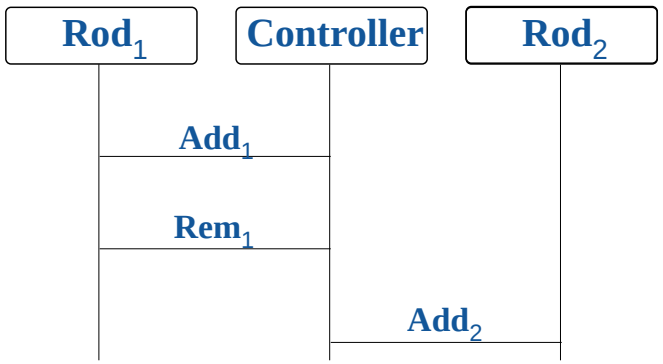


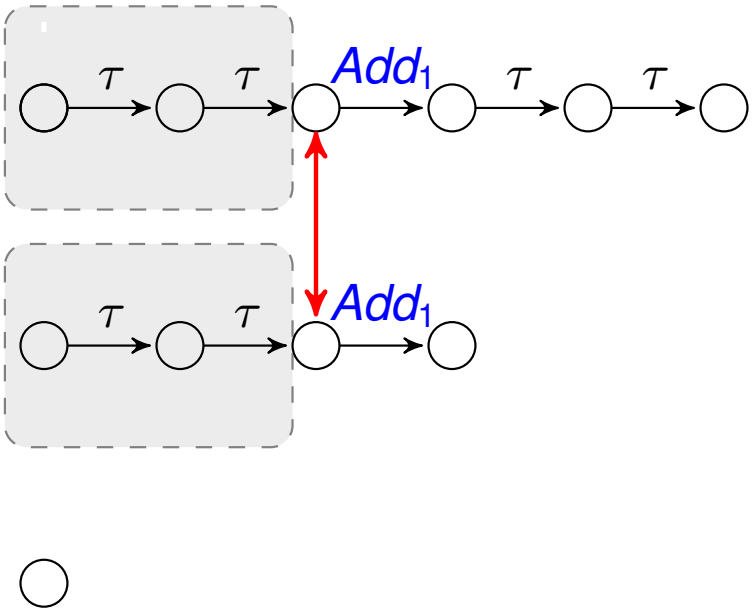


SAT - new states are reachable

Partitioned K-induction - Algorithm

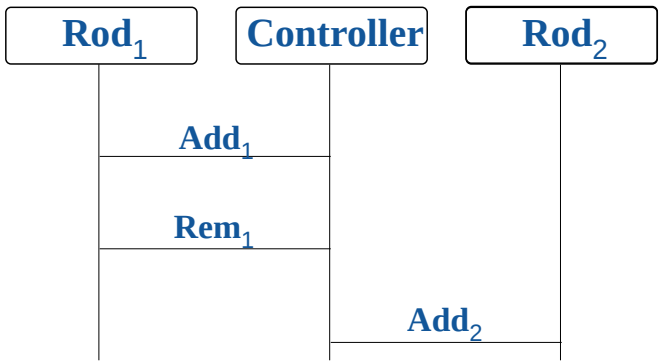
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

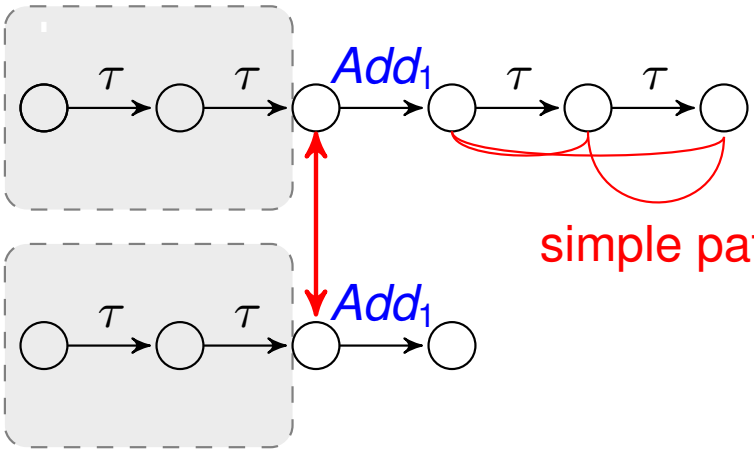




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

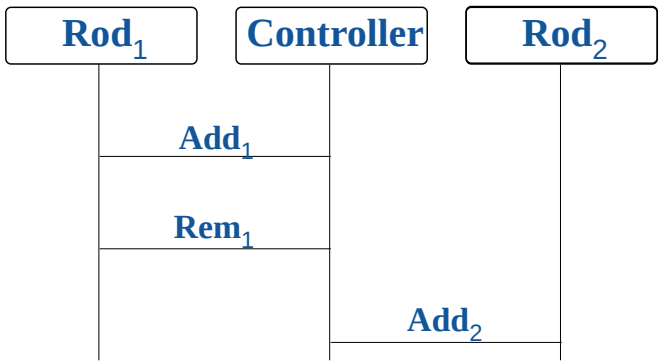


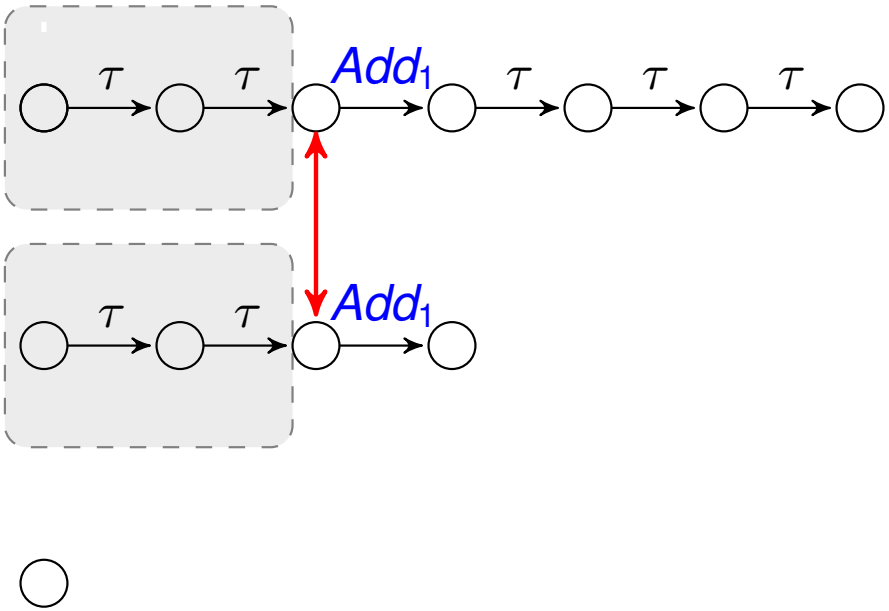


SAT - new states are reachable

Partitioned K-induction - Algorithm

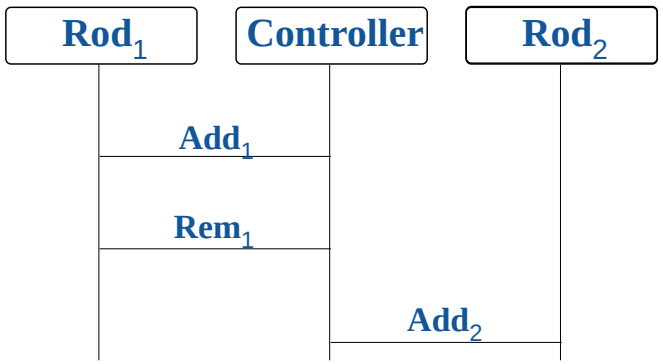
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

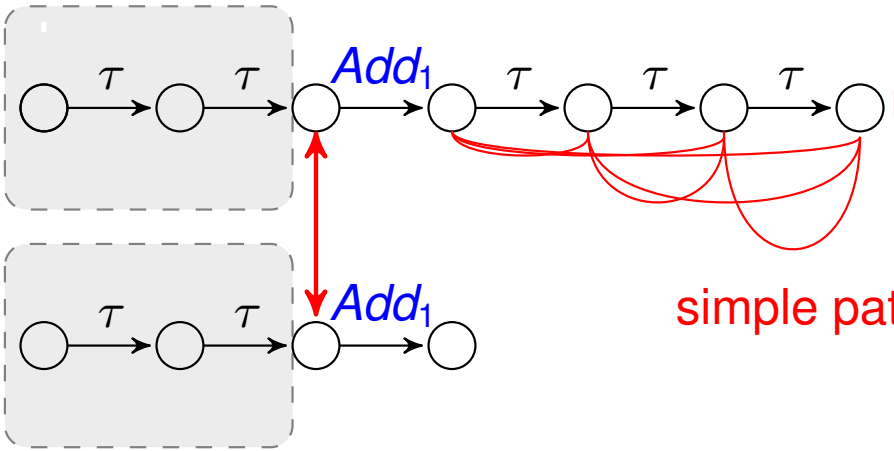




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



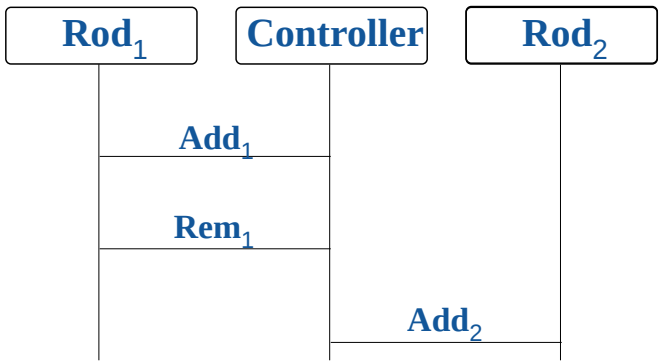


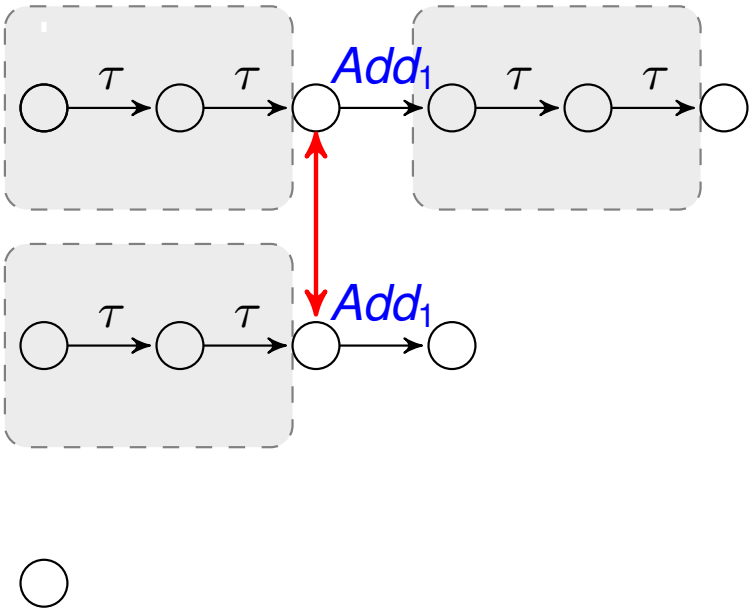
UNSAT - no new states are reachable

simple path

Partitioned K-induction - Algorithm

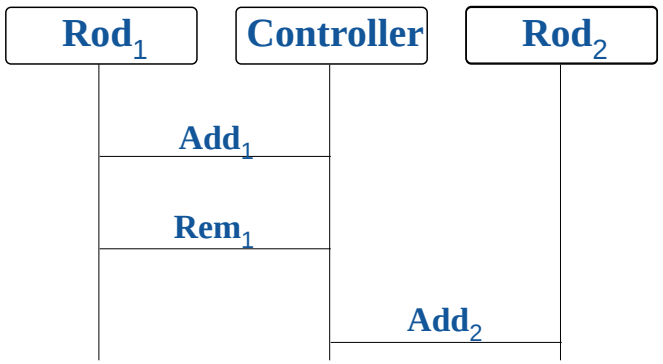
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

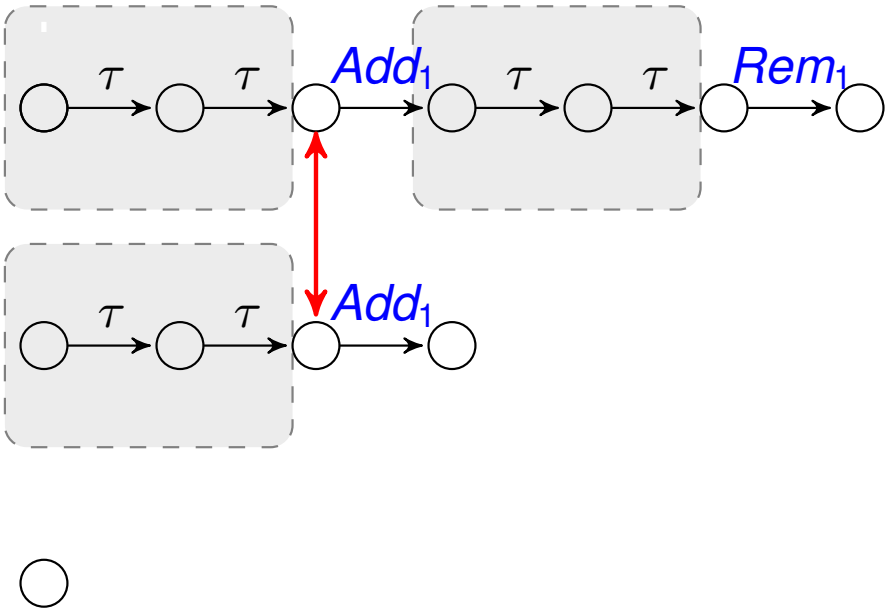




Partitioned K-induction - Algorithm

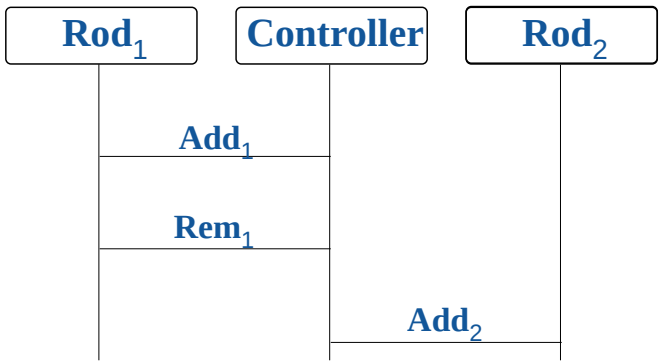
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

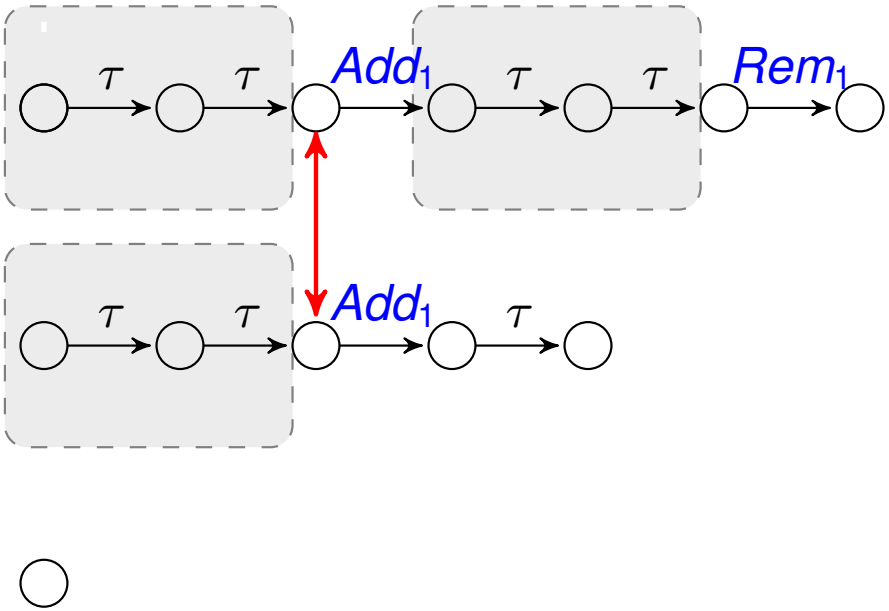




Partitioned K-induction - Algorithm

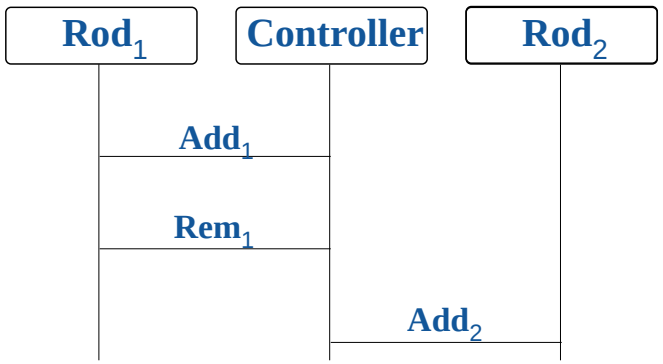
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

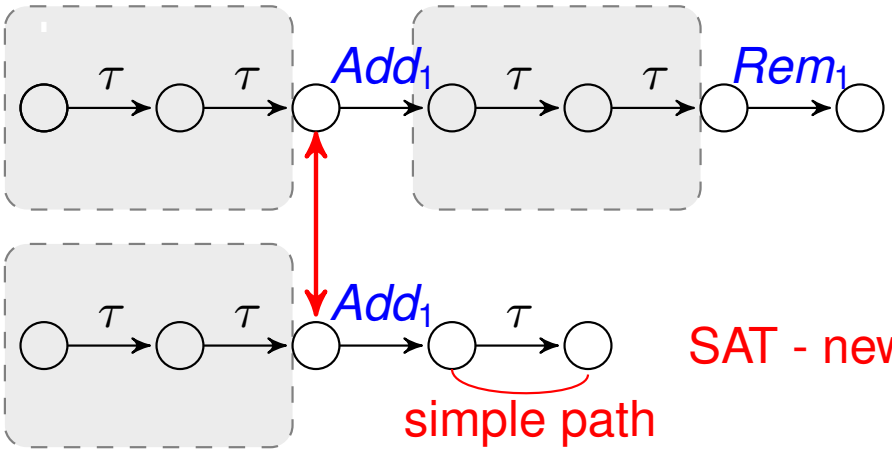




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

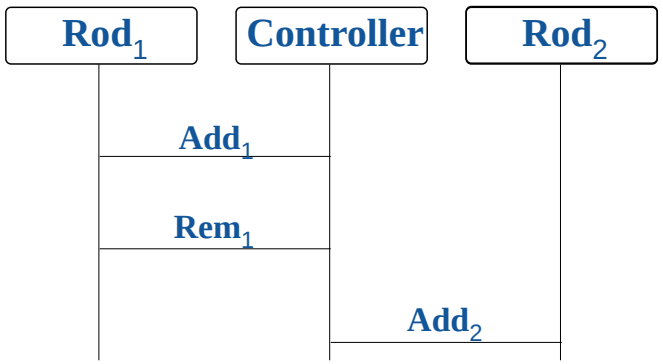


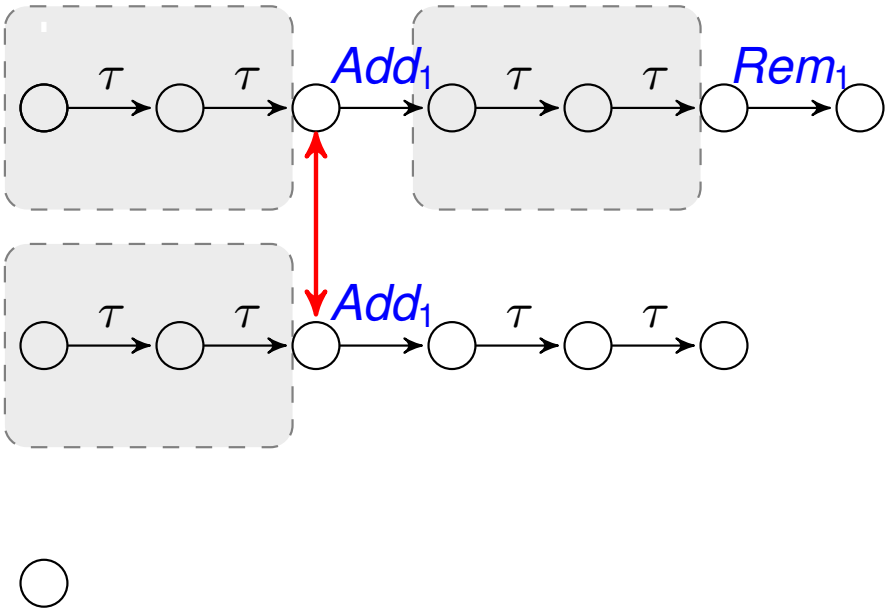


SAT - new states are reachable

Partitioned K-induction - Algorithm

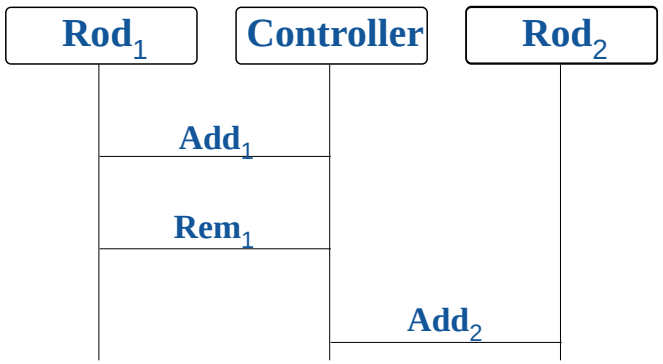
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

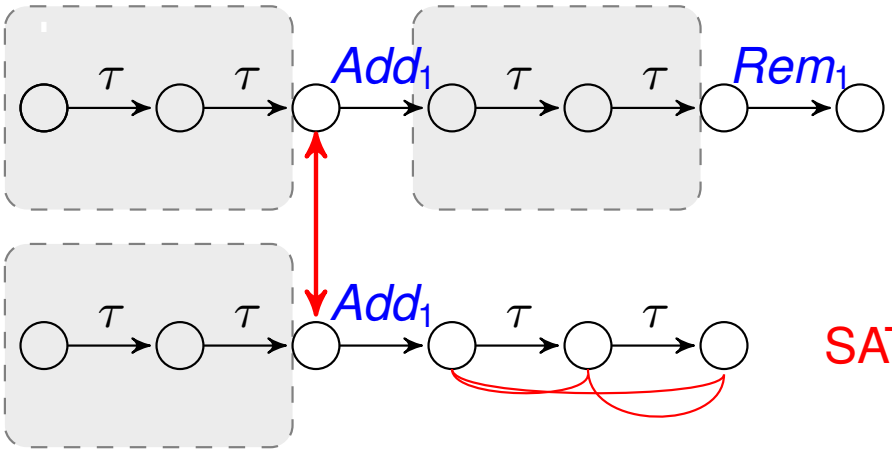




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



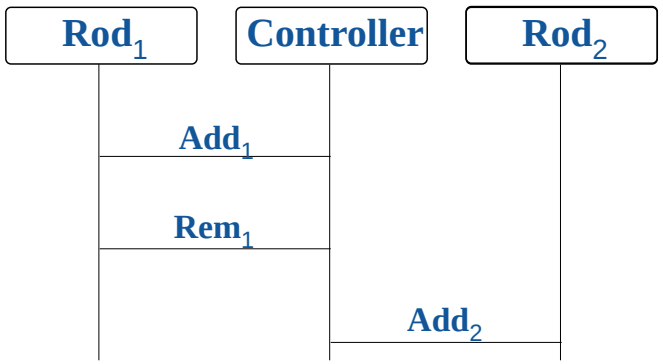


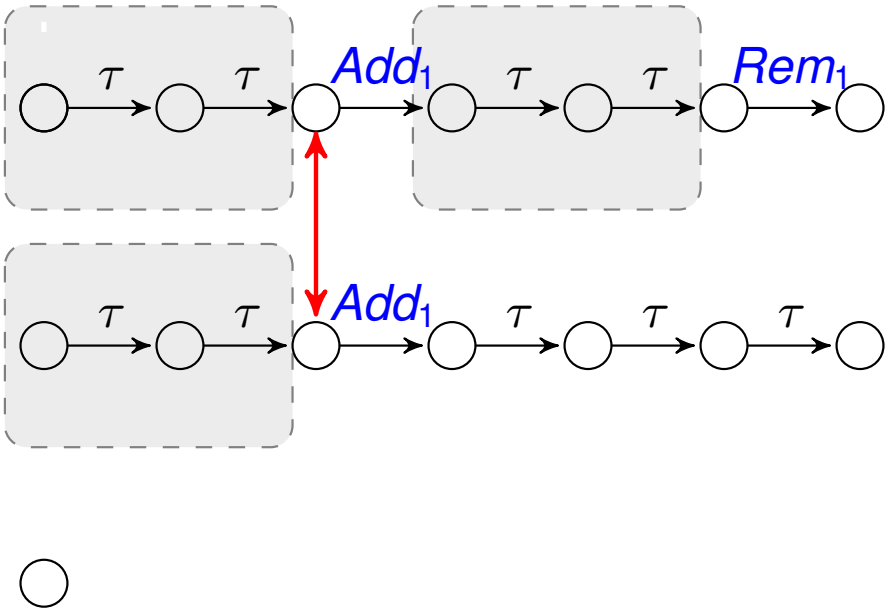
SAT - new states are reachable

simple path

Partitioned K-induction - Algorithm

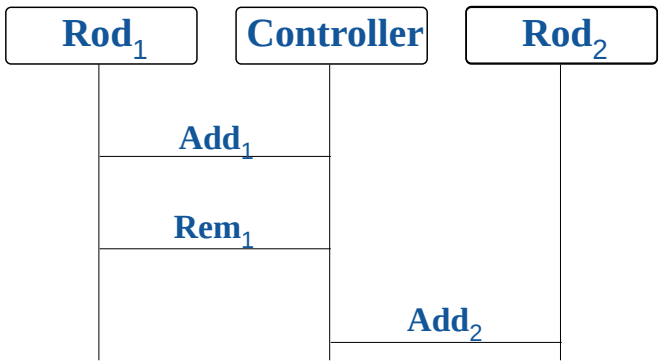
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

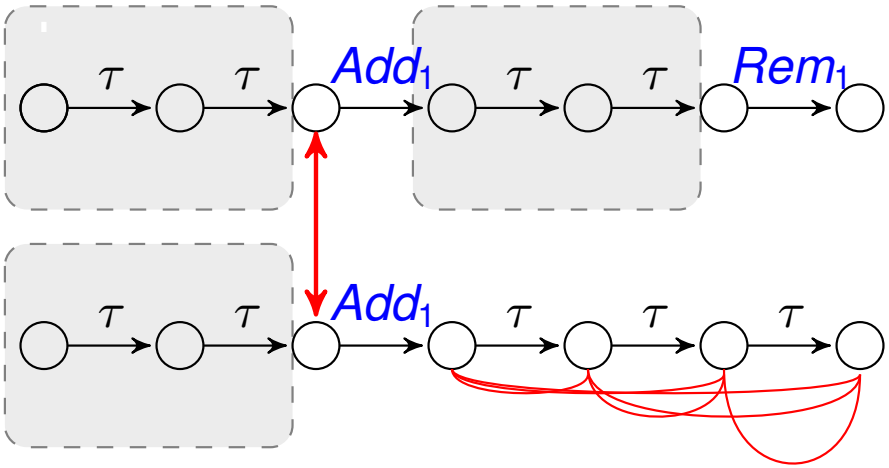




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.



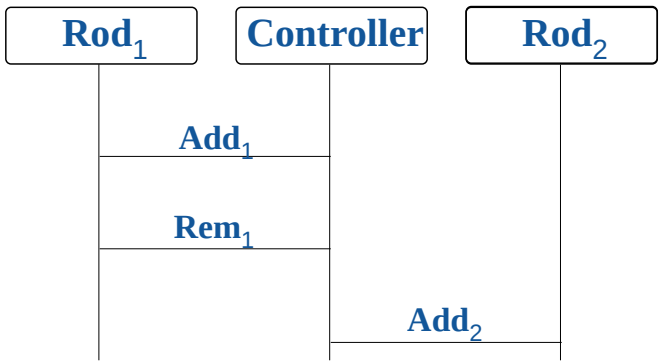


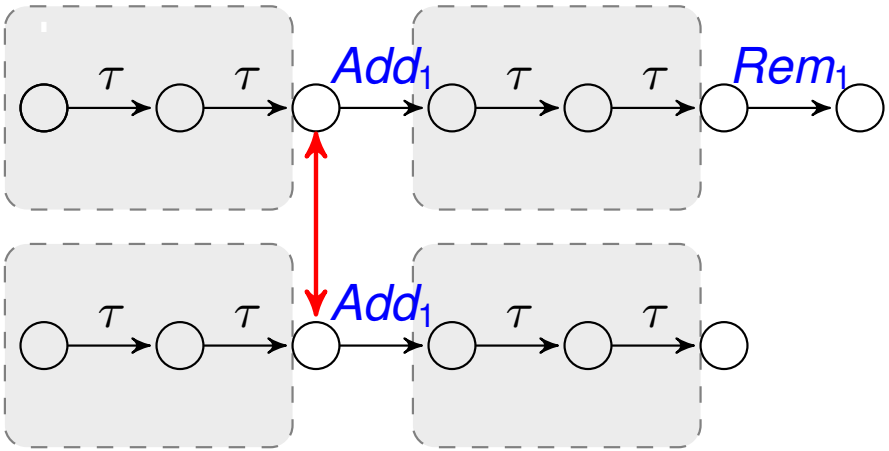
UNSAT - no new states are reachable

simple path

Partitioned K-induction - Algorithm

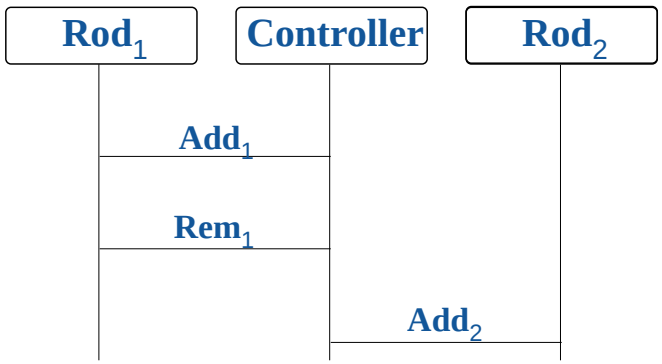
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

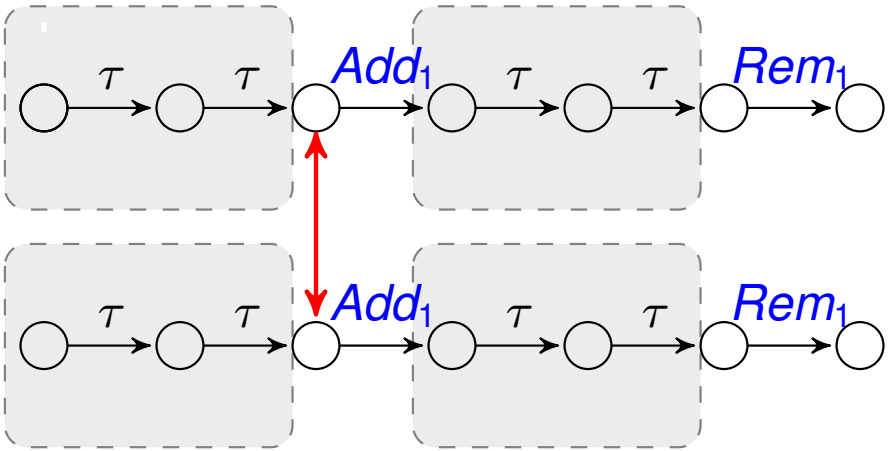




Partitioned K-induction - Algorithm

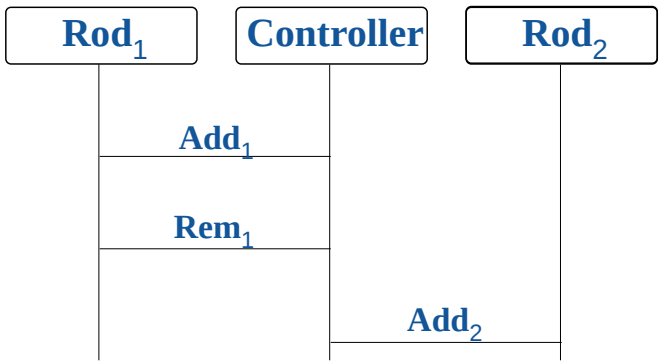
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

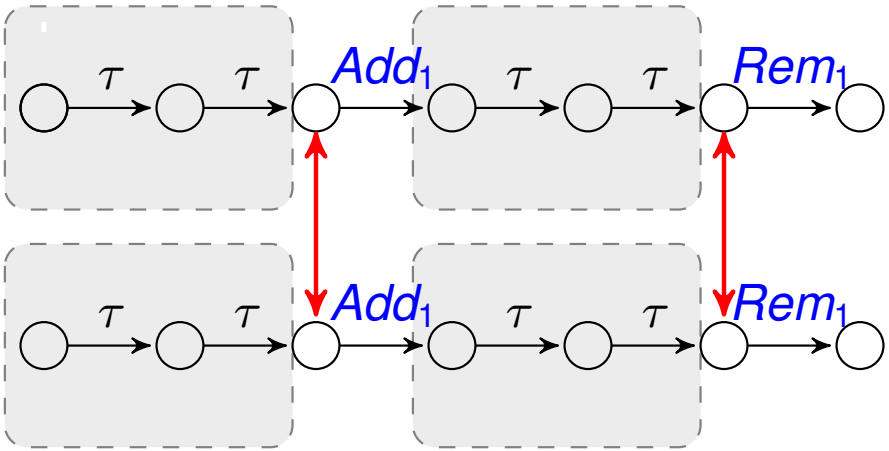




Partitioned K-induction - Algorithm

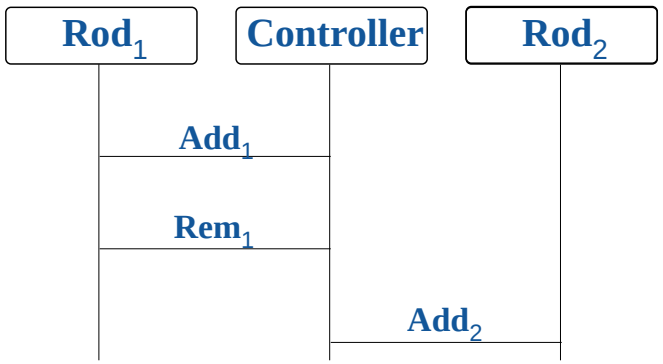
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

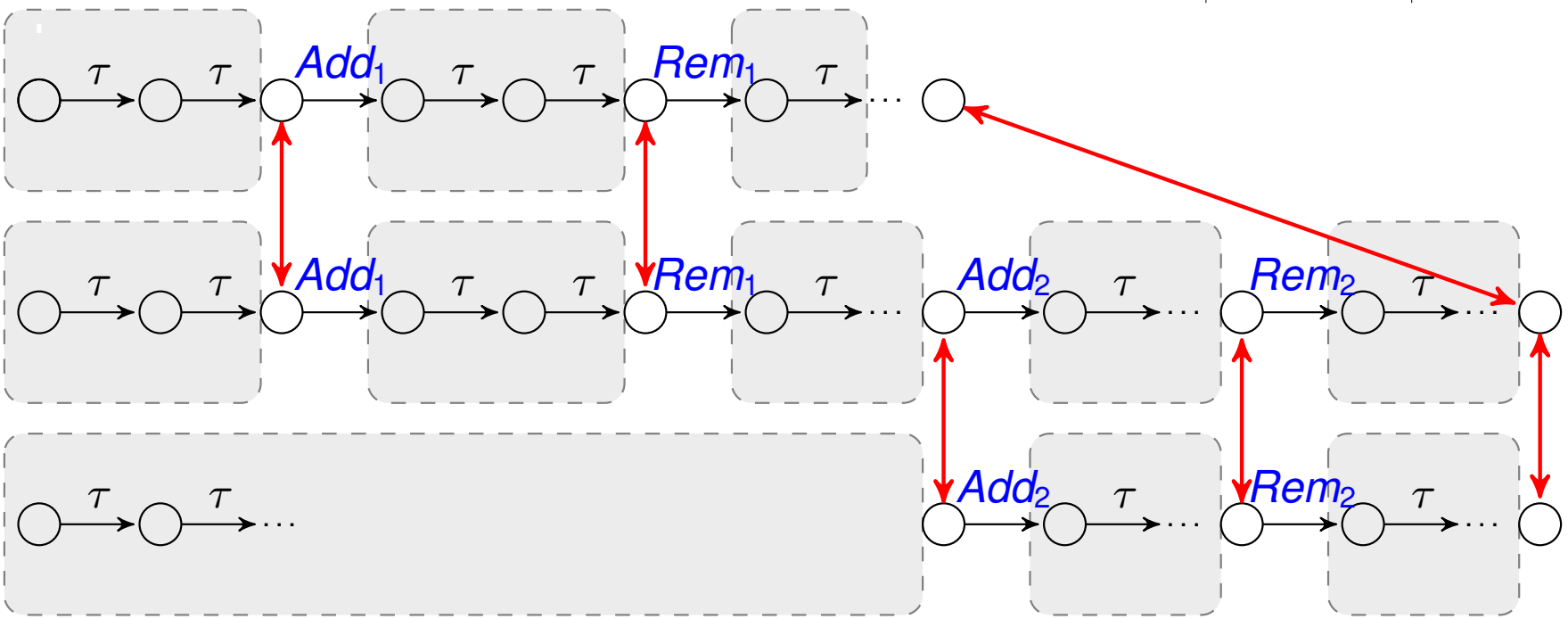




Partitioned K-induction - Algorithm

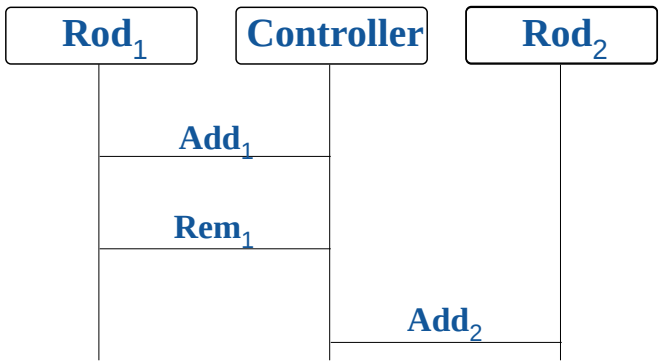
- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.

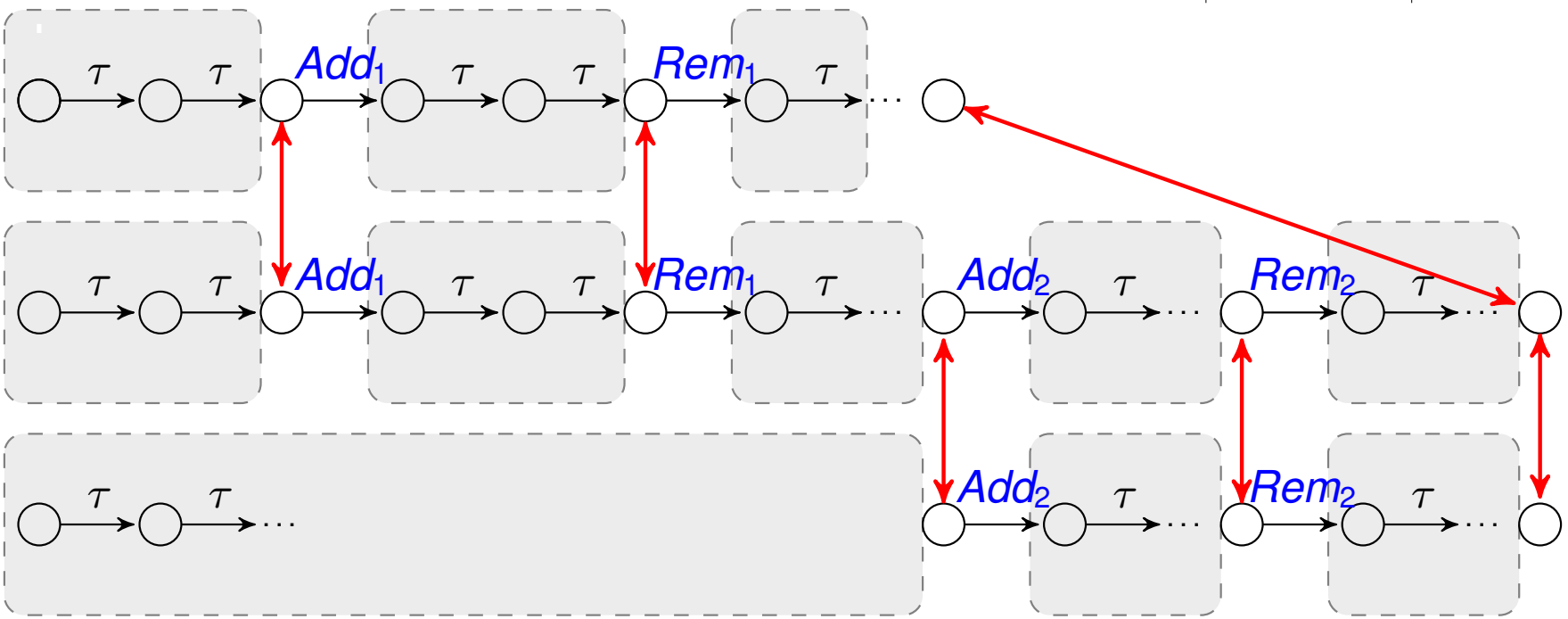




Partitioned K-induction - Algorithm

- Inductive step: proved incrementally following the partial order of the MSC.
- Base case: bounded feasibility check.





Unfeasible iff UNSAT

Experimental Evaluation

Implementation:

- Approach implemented on top of the NuSMV model checker.
- We use the MathSat SMT solver.

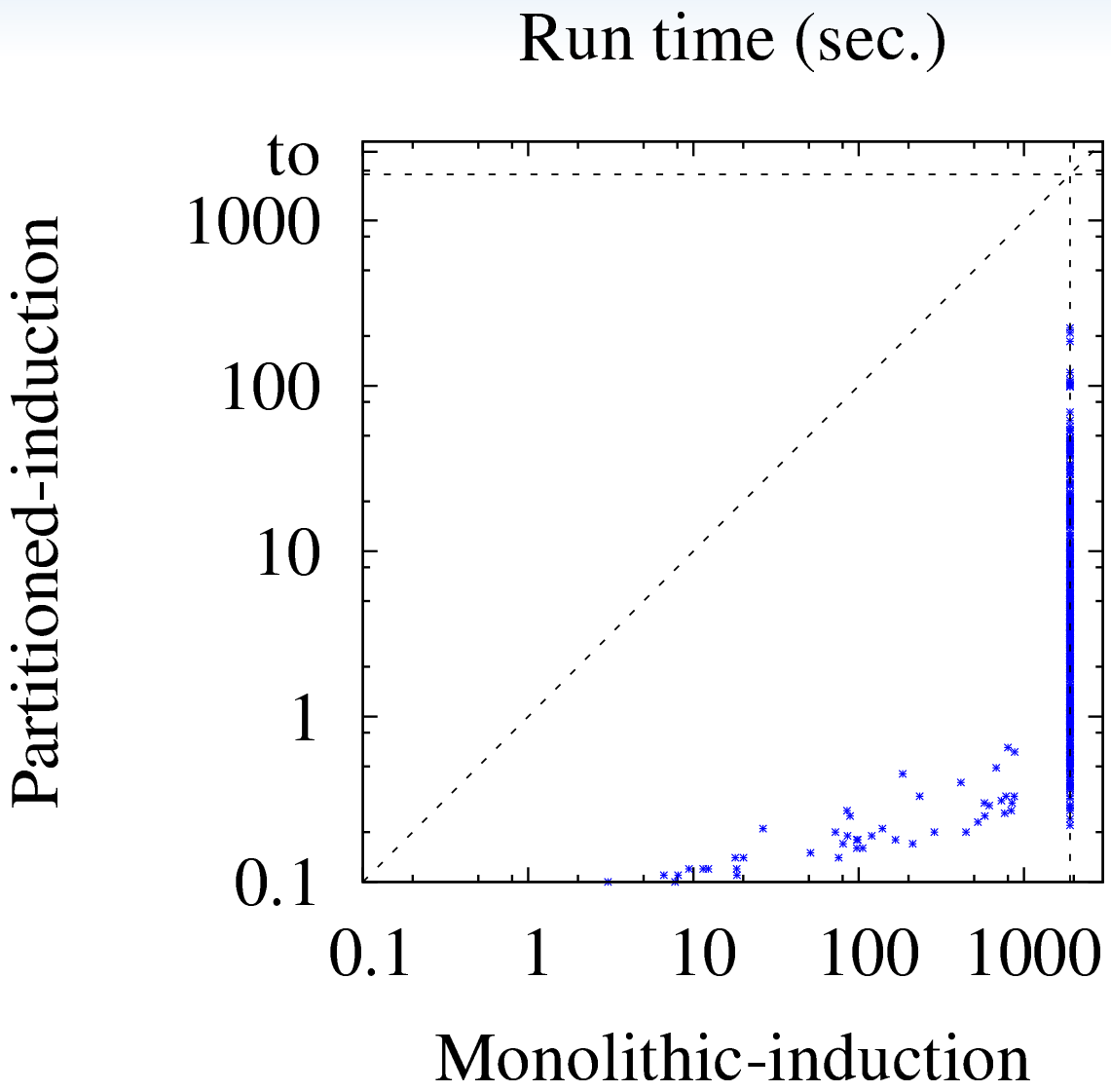
Settings:

- Linear hybrid automata benchmarks.
- Several handcrafted (unsatisfiable) MSCs.
- We scaled the dimension of the benchmarks (number of automata, length of the MSCs).

Comparison:

- MSC partitioned k-induction.
- Monolithic k-induction on the system composed with the monitor automata.

Scenario-based vs. automata-based k-induction



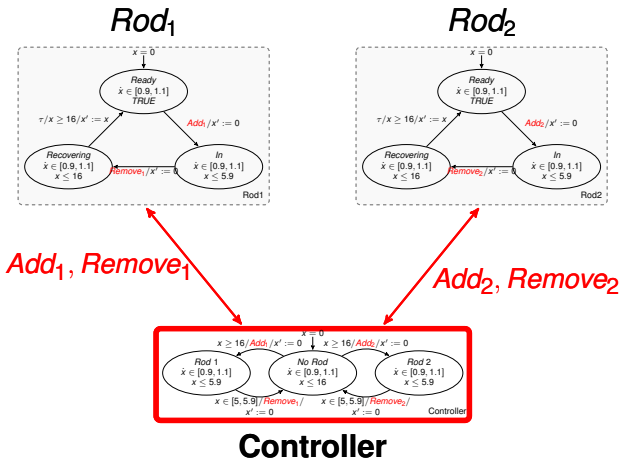
Explaining the unfeasibility

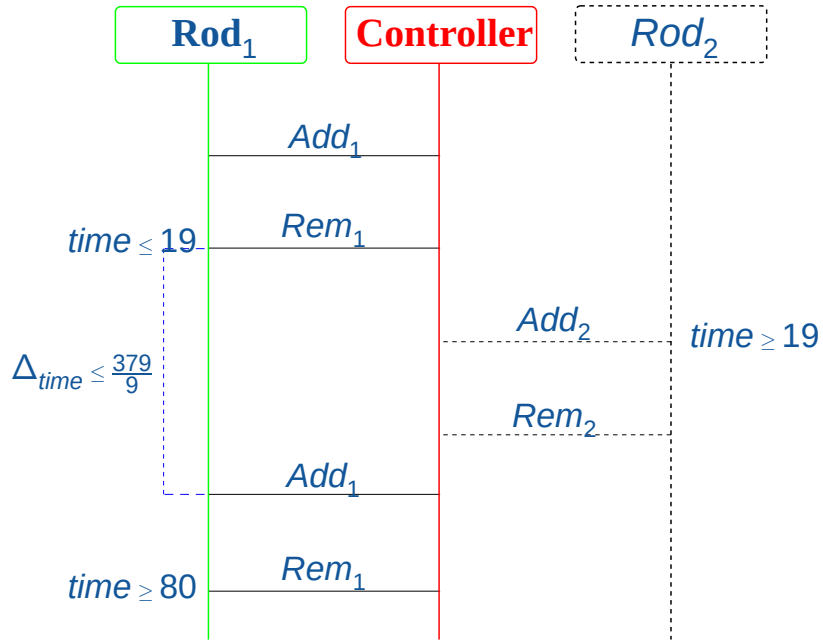
- Typical use case:
  - We expect that a scenario is feasible.
  - The analysis proves that the scenario is unfeasible in the network.
  - How do we explain the unfeasibility?
- We extract three types of explanations for the unfeasibility.

Unfeasibility due to a component

Explained with a formula that:

- Is required by the component when simulating its MSC events.
- Is not consistent with the other components when they simulate the events of the MSC.





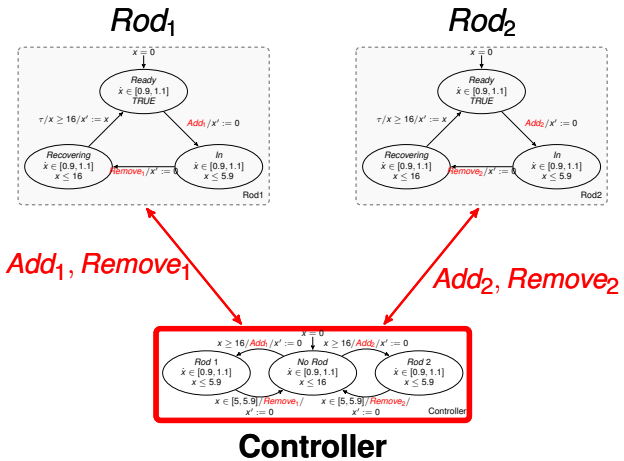
Unfeasibility due to a component

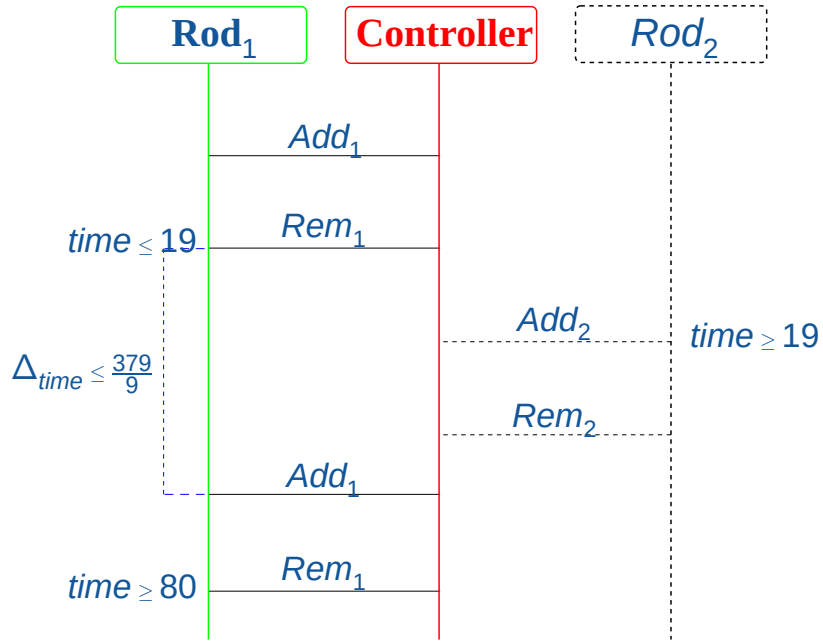
Explained with a formula that:

- Is required by the component when simulating its MSC events.
- Is not consistent with the other components when they simulate the events of the MSC.

It is the interpolant of A and B:

- A is the encoding of the component and its MSC events.
- B is the encoding of the other components and their MSC events.

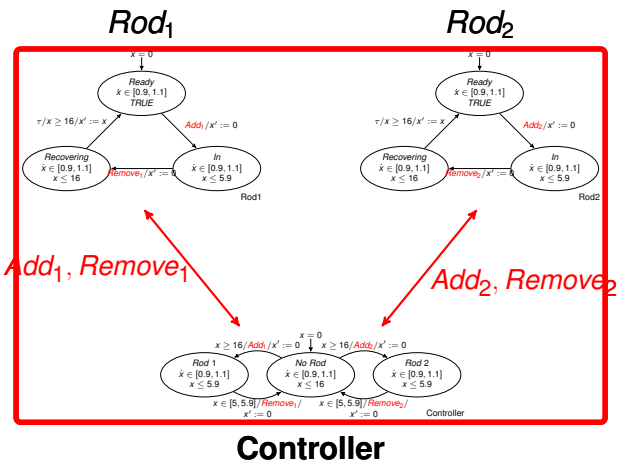


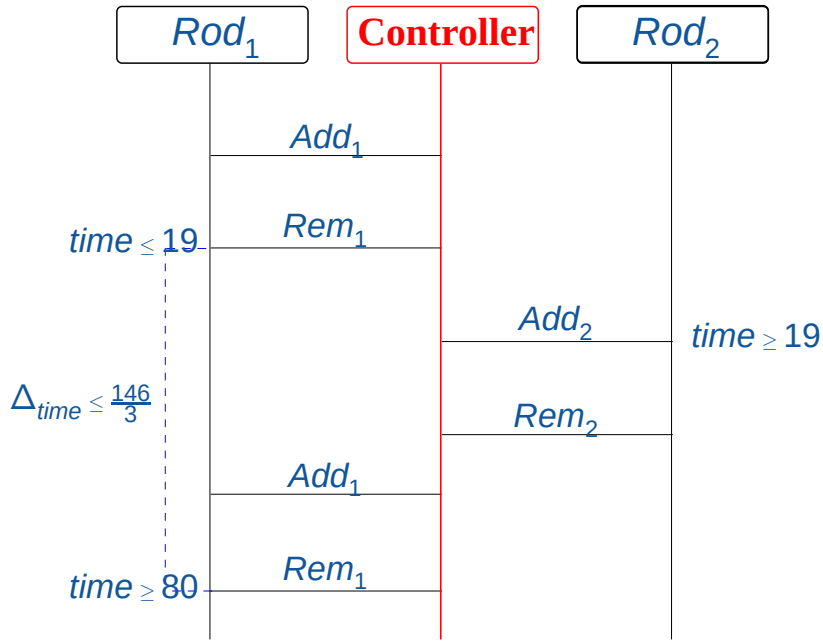


Unfeasibility due the network

Explained with a formula that:

- Is required by the network when simulating the MSC.
- Is not consistent with the additional constraints of the MSC.





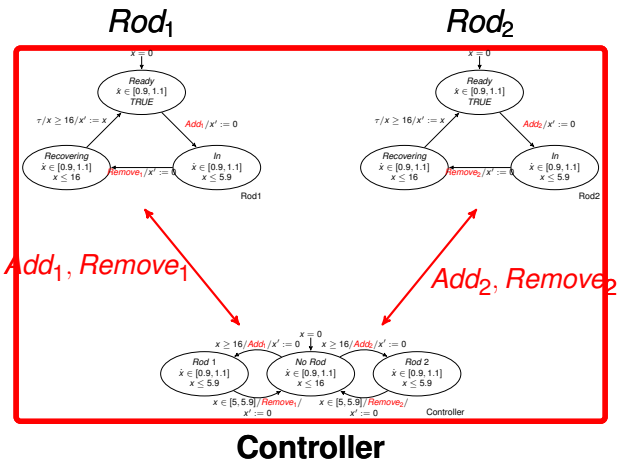
Unfeasibility due the network

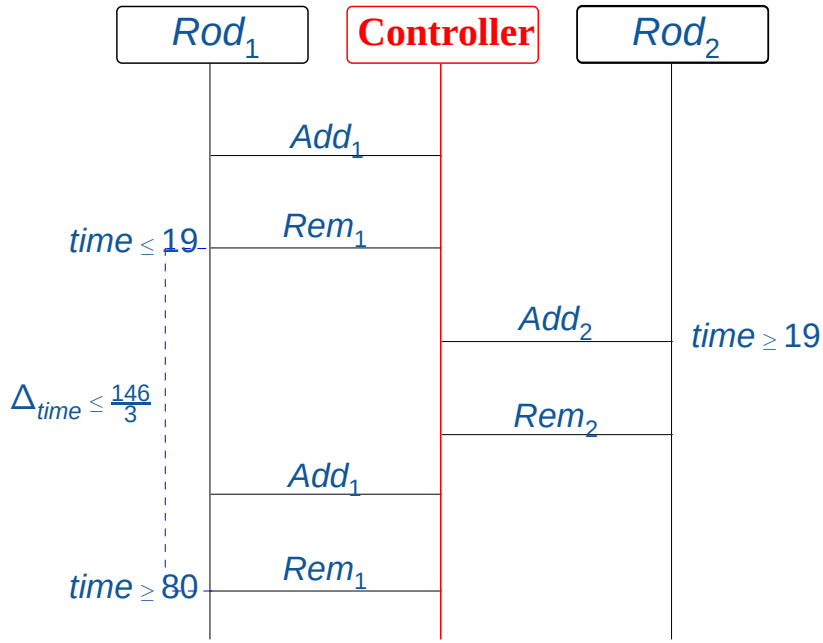
Explained with a formula that:

- Is required by the network when simulating the MSC.
- Is not consistent with the additional constraints of the MSC.

It is the interpolant of A and B:

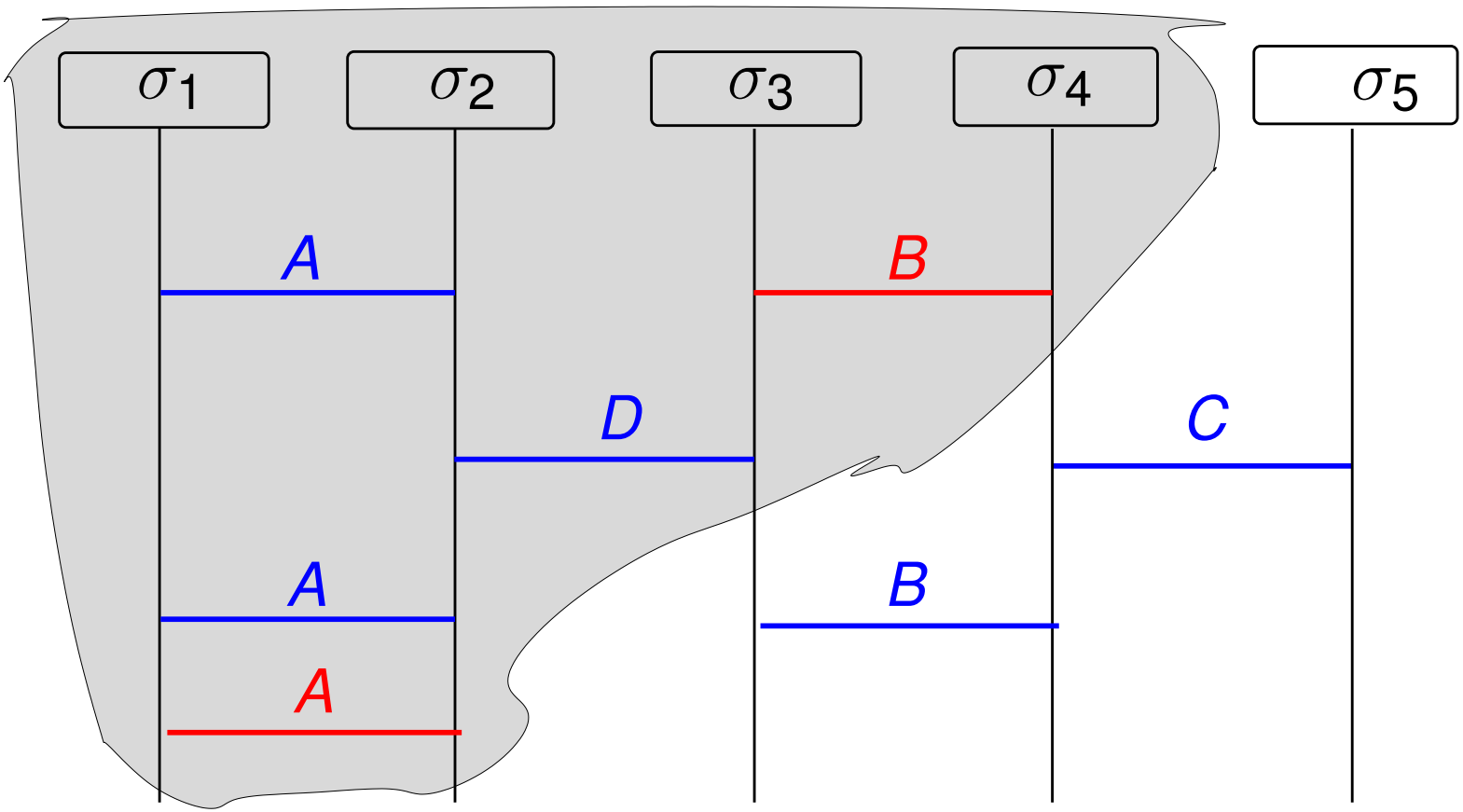
- A is the encoding of the network and the MSC.
- B are the CMSC constraints.





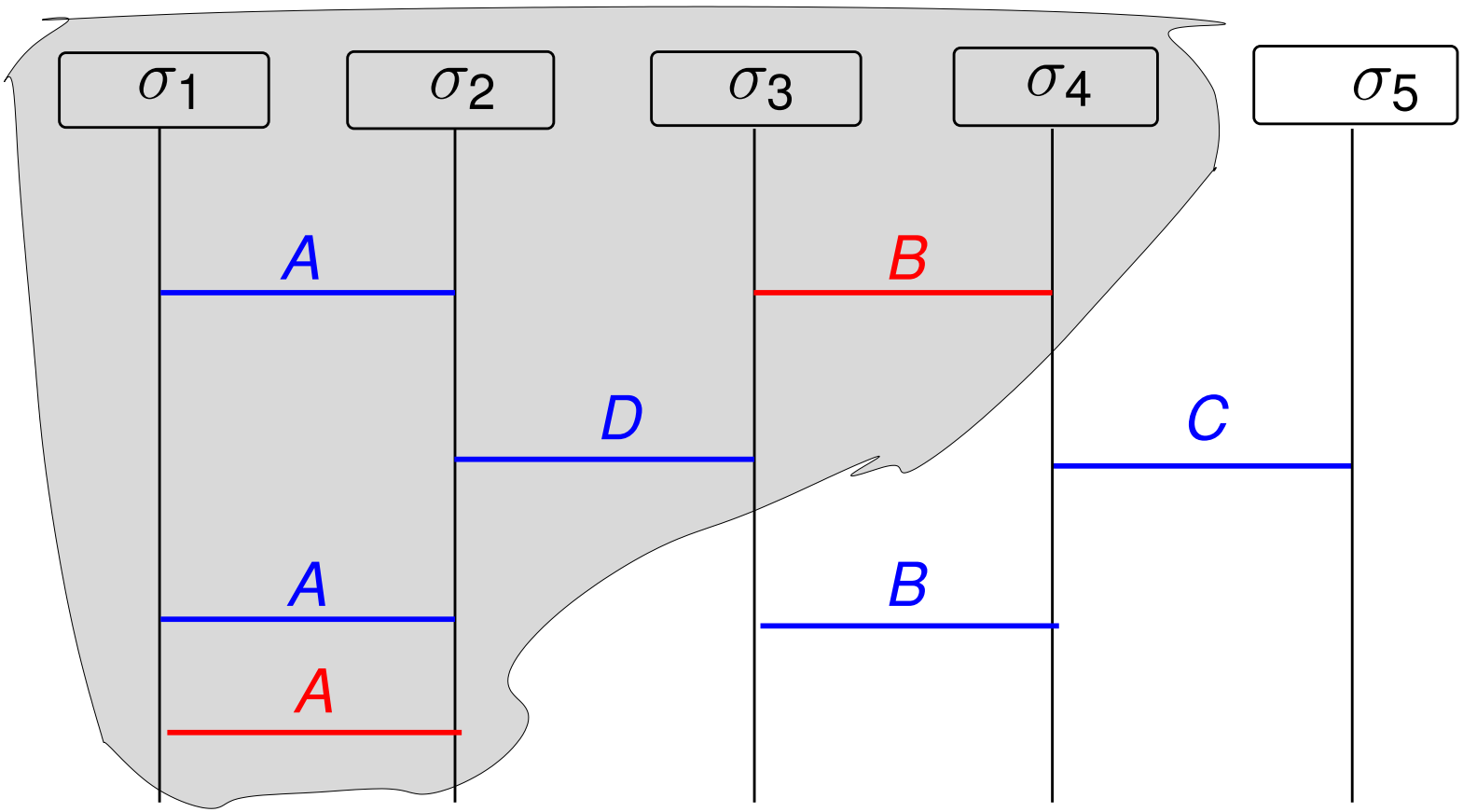
Inconsistent subset of the CMSC

Subset of the original CMSC that is still unfeasible with the network.



Inconsistent subset of the CMSC

Subset of the original CMSC that is still unfeasible with the network.
Extracted from the unsatisfiable core of the encoding.



Outline

Verification of More complex Requirements

- Beyond reachability - motivation to define more complex language.
- Contract based design - need property-based language.
- Even more important issue (different from verification).

Requirements are flawed

- The bugs are not in the system, but in the requirements!
  - The systems often implement correctly wrong/incomplete requirements.
  - Software system errors caused by requirements errors
- Not just a slogan, but a real user need.
- Considered as major problem of software development process by most European companies (EPRITI survey).
- Confirmed by NASA studies on Voyager and the Galileo software errors
  - Primary cause (62% on Voyager, 79% on Galileo): mis-understanding the requirements.
- Confirmed by the ESA and ERA recent calls on requirements.
- Widely acknowledged from industry across domains (IAI, RCF, Intecs, ...).

Requirements validation

- Requirements: descriptions of the functions provided by the system and its operational constraints.
- Requirements validation: checking if the requirements are correct, complete, consistent, and compliant with what the stakeholders have in mind.
- Target requirements errors:
  - Incomplete (e.g., incomplete description of a function),
  - Missing (e.g., missing assumption on lower levels),
  - Incorrect (e.g., wrong value in condition used to trigger some event),
  - Inconsistent (i.e., pair-wise incompatible),
  - Over-specified (e.g., more restrictive than necessary).

Cover 89% of faults examined in NASA projects.

Formal checks and feedback

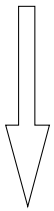
- Property-based approach:
  - One requirement, one formula.
  - Easy traceability.
  - Validation based on series of satisfiability problems:
    - consistent, i.e. if they do not contain some contradiction (sat of $\bigwedge_{1 \leq i \leq n} \varphi_i^{req}$)
    - not too strict, i.e. if they do allow some desired behavior (sat of $\bigwedge_{1 \leq i \leq n} \varphi_i^{req} \wedge \varphi^{des}$)
    - not too weak, i.e. if they rule out some undesired behavior (sat of $\bigwedge_{1 \leq i \leq n} \varphi_i^{req} \wedge \varphi^{und}$)
- Formal feedback:
  - Traces: witnesses of consistency, compatibility, property violation
  - Cores: subset of inconsistent, incompatible, property- entailing formulas

HRELTL: hybrid RELTL

- For hardware specification, standardized languages based on temporal logic + regular expressions (RELTL)
- For hybrid systems, necessary to predicate over:
  - integer and real variables,
  - continuous quantities,
  - instantaneous changes,
  - continuous evolutions (constraints over derivatives).
- Hybrid RELTL:
  - RELTL with the addition of:
    - continuous variables
    - arithmetic predicates with next and derivatives
  - Interpreted over hybrid traces.

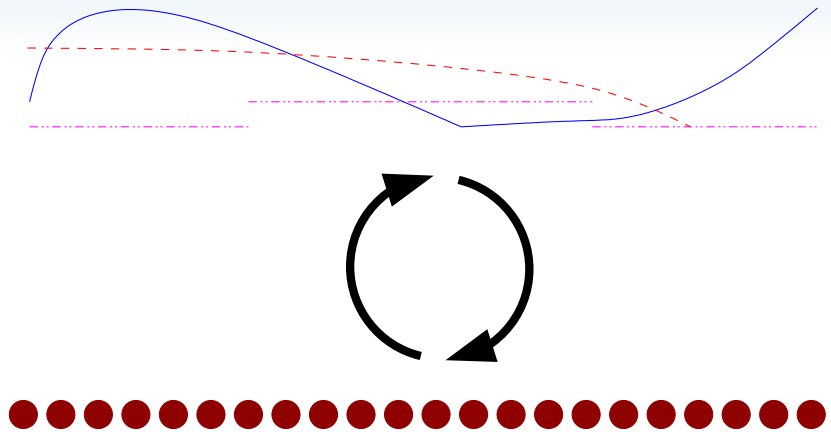
Reduction to discrete semantics

HRELTL



RELTL

(with SMT constraints)



- The translation τ of a generic HRELTL formula is defined as:

$$\tau(\varphi) := \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{PRED_\varphi} \wedge \psi_{V_D} \wedge \tau'(\varphi).$$

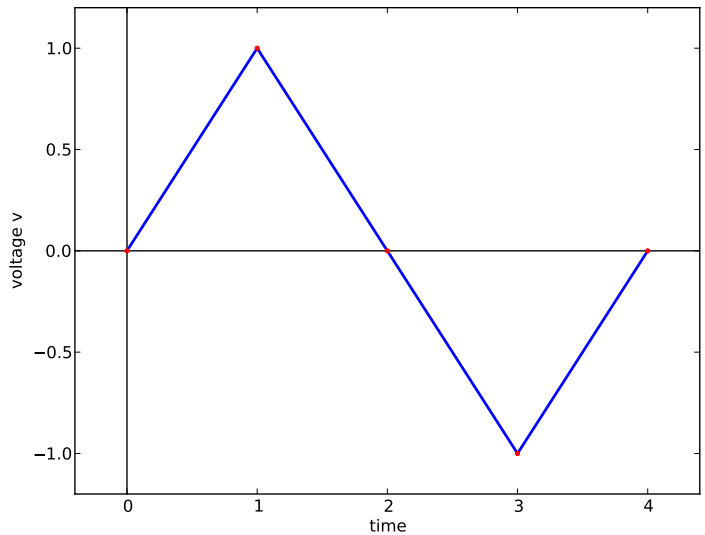
Theorem

φ and $\tau(\varphi)$ are equi-satisfiable.

Predicates over continuous evolution

- Example: a continuous oscillating signal.

```
-- v is a continuous variable
VAR v: continuous;
-- v does not jump
-- during discrete changes
CONSTRAINT
G ( STEP -> next(v)=v)
-- oscillating behavior
CONSTRAINT
G F ( v>0 ) & G F (v<0)
-- inconsistent scenario
CONSTRAINT
G (v!=0)
```



- Predicates may observe the value of continuous variable during continuous evolution.
- Discretization not as easy as in the automata case (where we have only invariants or urgent conditions).

Additional variables and formulas

- New variables:
  - δ_t tracks the elapsing of time;
  - ι tracks if the sampled interval is open or closed;
  - ζ is a parameter used to avoid the Zeno paradox;
  - $\dot{v}_l$ and $\dot{v}_r$ track the left and right derivative of v.
- ψ_ι models the represented sequence of intervals to be compliant with assumptions;
  - E.g., two consecutive singular intervals if and only if $\delta_t = 0$.
- ψ_{DER} encodes the relation among continuous variables and their derivatives in open intervals;
- ψ_{PRED_φ} constrain the set of predicates occurring in φ to model the continuity of represented functions;
  - if $p_=$ holds in an open interval, then $p_=$ holds in adjacent points;
  - we cannot move from $p_<$ to $p_>$ without passing through a state where $p_=$ holds.
- ψ_{V_D} encodes that discrete variables do not change value during a continuous evolution.

Equi-satisfiable RELTL formula

- The translation τ of a generic HRELTL formula is defined as:

$$\tau(\varphi) := \psi_\iota \wedge \psi_{\text{DER}} \wedge \psi_{PRED_\varphi} \wedge \psi_{V_D} \wedge \tau'(\varphi).$$

Theorem

φ and $\tau(\varphi)$ are equi-satisfiable.

- Possible non-linear constraints:
  - from input predicates over discrete variables;
  - from constraints on derivatives.
- if φ does not contain SEREs, then the translation is linear in the size of φ.

Mappings

- From hybrid to discrete:
  - $t_i = t$ if if $I_i = [t, t]$, and $t_i = (t + t')/2$ if $I_i = (t, t')$;
  - $s_i(v) = f_i^v(t_i)$;
  - if $I_i = (t, t')$, $s_i(\dot{v}_l)$ and $s_i(\dot{v}_r)$ are the mean value over $(t_i - t_{i-1})$ and (t_{i+1}, t_i); if $I_i = [t, t]$ then $s_i(\dot{v}_l) = \dot{f}_i^v(t)_-$, $s_i(\dot{v}_r) = \dot{f}_i^v(t)_+$;
  - $s_i(\iota) = \top$ if $I_i = [t, t]$, and $s_i(\iota) = \bot$ if $I_i = (t, t')$;
  - $s_i(\delta_t) = t_{i+1} - t_i$;
  - $s_i(\zeta) = \alpha$ as in the Cauchy's condition.
- From discrete to hybrid:
  - $t_i = \sum_{0 \leq j < i-1} s_j(\delta_t)$,
  - if $s_i(\iota) = \top$ then $I_i = [t_i, t_i]$ else $I_i = (t_{i-1}, t_{i+1})$,
  - f_i is the piecewise linear function defined as $f_i^v(t) = s_i(v) - s_i(\dot{v}_l) \times (t_i - t)$ if $t \leq t_i$ and as $f_i^v(t) = s_i(v) + s_i(\dot{v}_r) \times (t - t_i)$ if $t > t_i$.

Satisfiability procedure

The approach:

- convert hybrid formula into discrete φ
- build a fair transition system S_φ
- check whether the language accepted by S_φ is not empty.

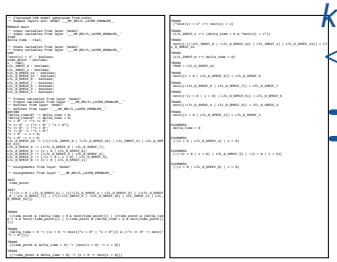
Underlying reasoning engine:

- Bounded Model Checking
  - maximum bound not guaranteed
  - only lasso-shape traces
- Model Checking with Predicate Abstraction and automatic refinement
  - set of predicates not guaranteed to exist
  - abstraction refinement not always guaranteed to find the right set of predicates.

Satisfiability procedure - example

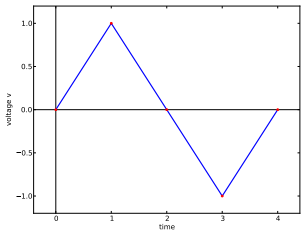
```
VAR v: continuous;
CONSTRAINT
G ( STEP -> next(v)=v)
CONSTRAINT
G F ( v>0 ) & G F (v<0)
-- consistent scenario
CONSTRAINT
! G (v!=0)
```

11 boolean variables
2 real variables
4 fairness conditions



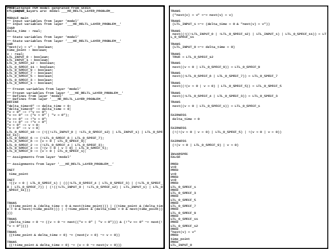
BMC (with fairness)
$k = 4$
$<$ 1 second

SAT




```
VAR v: continuous;
CONSTRAINT
G ( STEP -> next(v)=v)
CONSTRAINT
G F ( v>0 ) & G F (v<0)
-- inconsistent scenario
CONSTRAINT
G (v!=0)
```

11 boolean variables
2 real variables
3 fairness conditions



K-induction + predicate abs.
$k = 6$, 14 predicates
$<$ 1 second

UNSAT

OTHELLO specification language

- OTHELLO = Object Temporal Hybrid expressions Linear-time temporal Logic
- Example:

  The train trip shall issue an emergency brake command, which shall not be revoked until the train has reached standstill and the driver has acknowledged the trip (ETCS SRS Sec. 3.13.8.2)

  for all t **of type** *Train* **(**$t.trip$ **implies**
  ($t.emergency\_brake$ **until (**$t.speed = 0$ **and** $t.driver.ack$**)))**
- Result of industrial project EuRailCheck.
- Base of MSR award winner project OthelloPlay.

Outline

Conclusions

- the role of hybrid automata to model complex embedded systems
- cover all the phase of the lifecycle
  - design (off-line)
  - operation (on-line)
- focus on large discrete state space
- SMT supports a comprehensive modeling framework
- provides for powerful analysis capabilities

Future Challenges

- dealing with nonlinear dynamics in SMT setting
  - integrating local time with nonlinear dynamics
  - integrating more complex dynamics within SMT-based backends
- specialized techniques for analog/mixed signals
  - composition of discrete controller and physical plant
  - fixed-rate sampling of physical plant
  - discrete controller as software
- analysis of parameterized systems, and synthesis of interesting parameter spaces
- modeling in activity-based/transaction-based view [CMR12a,CMRb]

Outline

References I

- Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani, *A sat based approach for solving formulas over boolean and linear mathematical propositions*, CADE (Andrei Voronkov, ed.), Lecture Notes in Computer Science, vol. 2392, Springer, 2002, pp. 195–210.
- Rajeev Alur, Thao Dang, and Franjo Ivancic, *Counterexample-guided predicate abstraction of hybrid systems*, Theor. Comput. Sci. **354** (2006), no. 2, 250–271.
- Eugene Asarin, Thao Dang, and Oded Maler, *The d/dt tool for verification of hybrid systems*, CAV, 2002, pp. 365–370.
- R. Alur, *Formal verification of hybrid systems*, EMSOFT, 2011, pp. 273–278.

References II

Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani, *An incremental and layered procedure for the satisfiability of linear arithmetic logic*, TACAS (Nicolas Halbwachs and Lenore D. Zuck, eds.), Lecture Notes in Computer Science, vol. 3440, Springer, 2005, pp. 317–333.

Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Ziyad Hanna, Alexander Nadel, Amit Palti, and Roberto Sebastiani, *A lazy and layered smt(bv) solver for hard industrial verification problems*, CAV (Werner Damm and Holger Hermanns, eds.), Lecture Notes in Computer Science, vol. 4590, Springer, 2007, pp. 547–560.

Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani, *Delayed theory combination vs. nelson-oppen for satisfiability modulo theories: a comparative analysis*, Ann. Math. Artif. Intell. **55** (2009), no. 1-2, 63–99.

References III

- J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, *Partial Order Reductions for Timed Systems*, CONCUR, 1998, pp. 485–500.
- Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar, *Computing predicate abstractions by integrating bdds and smt solvers*, FMCAD, IEEE Computer Society, 2007, pp. 69–76.
- Alessandro Cimatti, Jori Dubrovin, Tommi A. Junttila, and Marco Roveri, *Structure-aware computation of predicate abstraction*, FMCAD, IEEE, 2009, pp. 9–16.
- Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani, *Efficient interpolant generation in satisfiability modulo theories*, TACAS (C. R. Ramakrishnan and Jakob Rehof, eds.), Lecture Notes in Computer Science, vol. 4963, Springer, 2008, pp. 397–412.

References IV


_____, *Interpolant generation for utvpi*, CADE (Renate A. Schmidt, ed.), Lecture Notes in Computer Science, vol. 5663, Springer, 2009, pp. 167–182.


_____, *Efficient generation of craig interpolants in satisfiability modulo theories*, ACM Trans. Comput. Log. **12** (2010), no. 1, 7.


_____, *Computing small unsatisfiable cores in satisfiability modulo theories*, J. Artif. Intell. Res. (JAIR) **40** (2011), 701–728.


Alessandro Cimatti, Sergio Mover, and Stefano Tonetta, *Quantifier-free smt encoding of non-linear hybrid automata*, FMCAD, 2012.


Werner Damm, Stefan Disch, Hardi Hungar, Swen Jacobs, Jun Pang, Florian Pigorsch, Christoph Scholl, Uwe Waldmann, and Boris Wirtz, *Exact state set representations in the verification of linear hybrid systems with large discrete state space*, ATVA, 2007, pp. 425–440.

References V

- Anders Franzén, Alessandro Cimatti, Alexander Nadel, Roberto Sebastiani, and Jonathan Shalev, *Applying smt in symbolic execution of microcode*, FMCAD, IEEE, 2010, pp. 121–128.
- Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler, *Spaceex: Scalable verification of hybrid systems*, CAV, 2011, pp. 379–395.
- Goran Frehse, *Phaver: algorithmic verification of hybrid systems past hytech*, STTT **10** (2008), no. 3, 263–279.
- Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi, *Hytech: A model checker for hybrid systems*, CAV, 1997, pp. 460–463.
- Thomas A. Henzinger and Howard Wong-Toi, *Linear phase-portrait approximations for nonlinear hybrid systems*, Hybrid Systems, 1995, pp. 377–388.

References VI

- Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras, *Smt techniques for fast predicate abstraction*, CAV (Thomas Ball and Robert B. Jones, eds.), Lecture Notes in Computer Science, vol. 4144, Springer, 2006, pp. 424–437.
- Scott Little, David Walter, Chris J. Myers, Robert A. Thacker, Satish Batchu, and Tomohiro Yoneda, *Verification of analog/mixed-signal circuits using labeled hybrid petri nets*, IEEE Trans. on CAD of Integrated Circuits and Systems **30** (2011), no. 4, 617–630.
- Stephen Prajna and Ali Jadbabaie, *Safety verification of hybrid systems using barrier certificates*, HSCC, 2004, pp. 477–492.
- André Platzer and Jan-David Quesel, *Keymaera: A hybrid theorem prover for hybrid systems (system description)*, IJCAR, 2008, pp. 171–178.
- Ashish Tiwari, *Abstractions for hybrid systems*, Formal Methods in System Design **32** (2008), no. 1, 57–83.

References VII

Stefano Tonetta, *Abstract model checking without computing the abstraction*, FM (Ana Cavalcanti and Dennis Dams, eds.), Lecture Notes in Computer Science, vol. 5850, Springer, 2009, pp. 89–105.

Farn Wang, *Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures*, IEEE TRANS. SOFTW. ENG, Springer-Verlag, 2004, pp. 38–51.