SMT-Based Verification with MathSAT

Alessandro Cimatti

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

SAT/SMT School, Boston, June 2011

Introduction

- Many interesting talks about the internals of SMT solvers.
- Here:
  - a quick overview of the MathSAT solver
  - focus on the use of MathSAT for formal verification

Outline

Outline

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: MathSAT4
- Soon to be released: 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

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+05b]
  - 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  - superseded by model-based combination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
- 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

Infinite State Transition Systems

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
- IC3?
- Abstraction/refinement

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

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

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

  ...

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

  $\neg B(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge \neg B(X_{k-1}) \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
- Proved correct if unsatisfiable (and no bugs until k)
- Invariant strengthening, simple path condition, ...

SMT-based interpolation

- An interpolant for an unsatisfiable formula

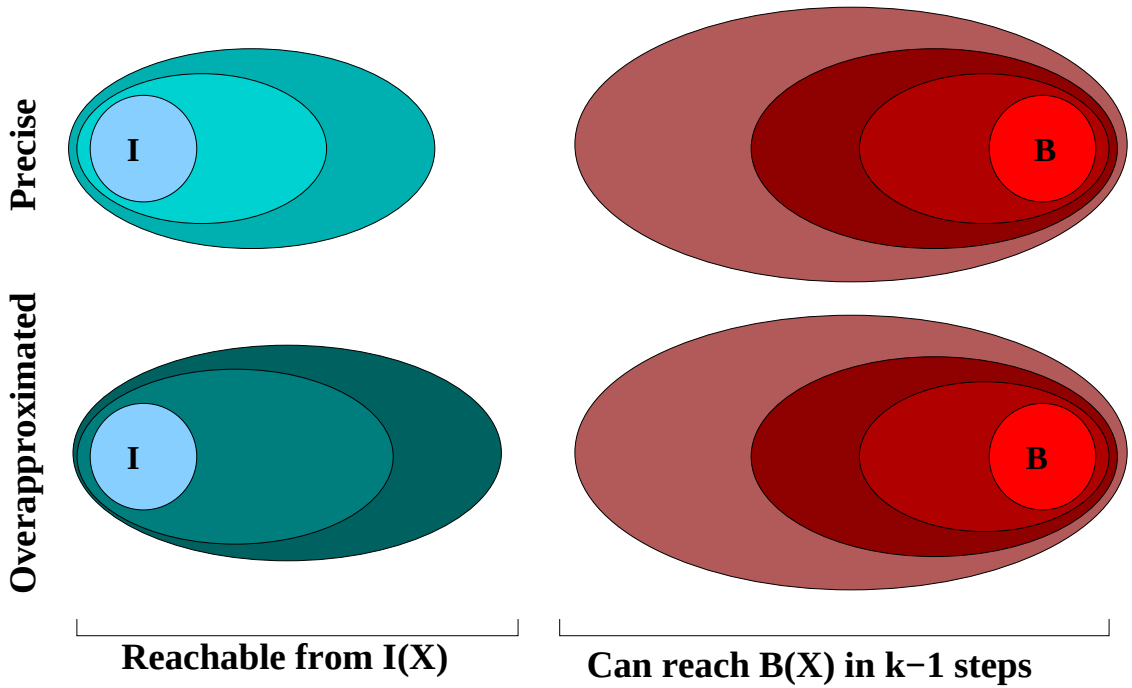
$$\Phi_1(X, Y) \wedge \Phi_2(Y, Z)$$

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

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

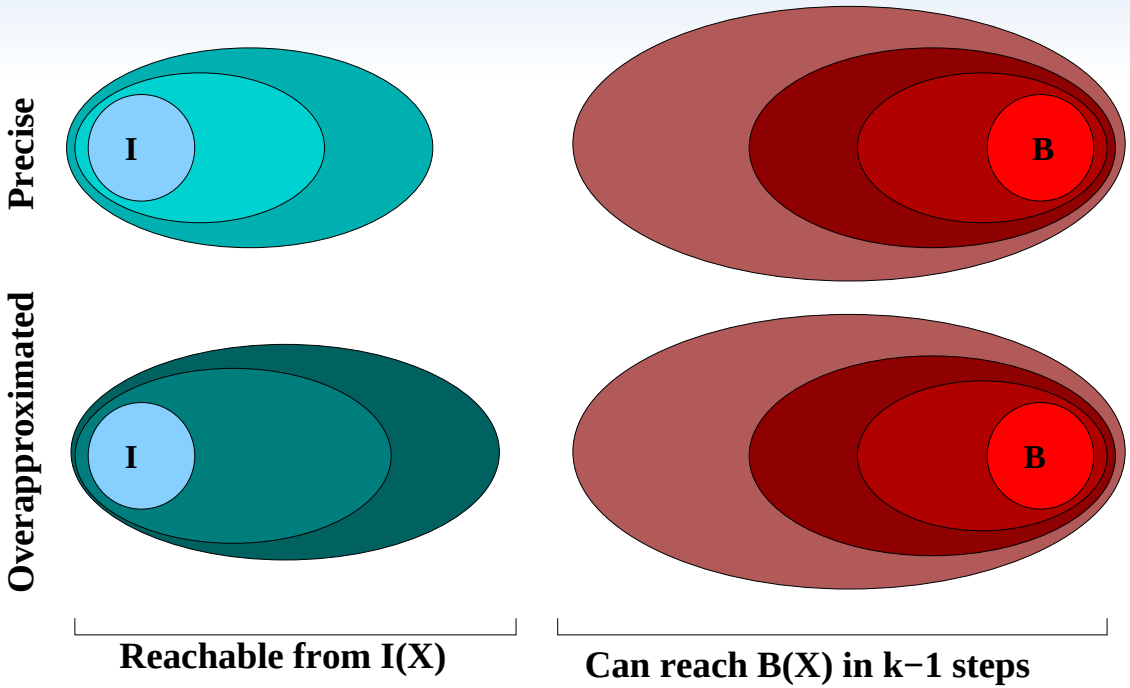
SMT-based interpolation

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



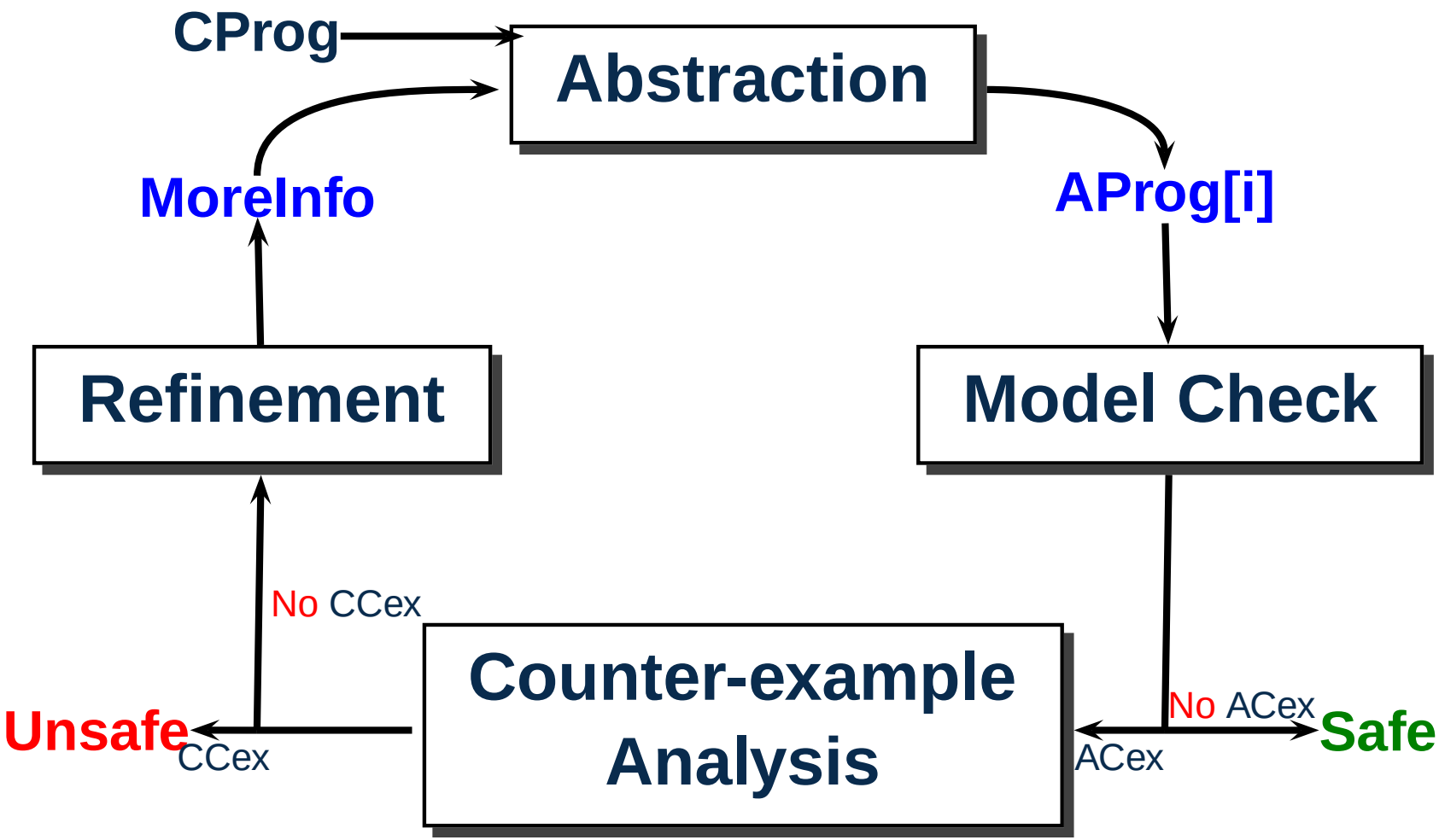
$$Itp(X_1) = Itp(R, I(X_0), k)$$

SMT-based interpolation

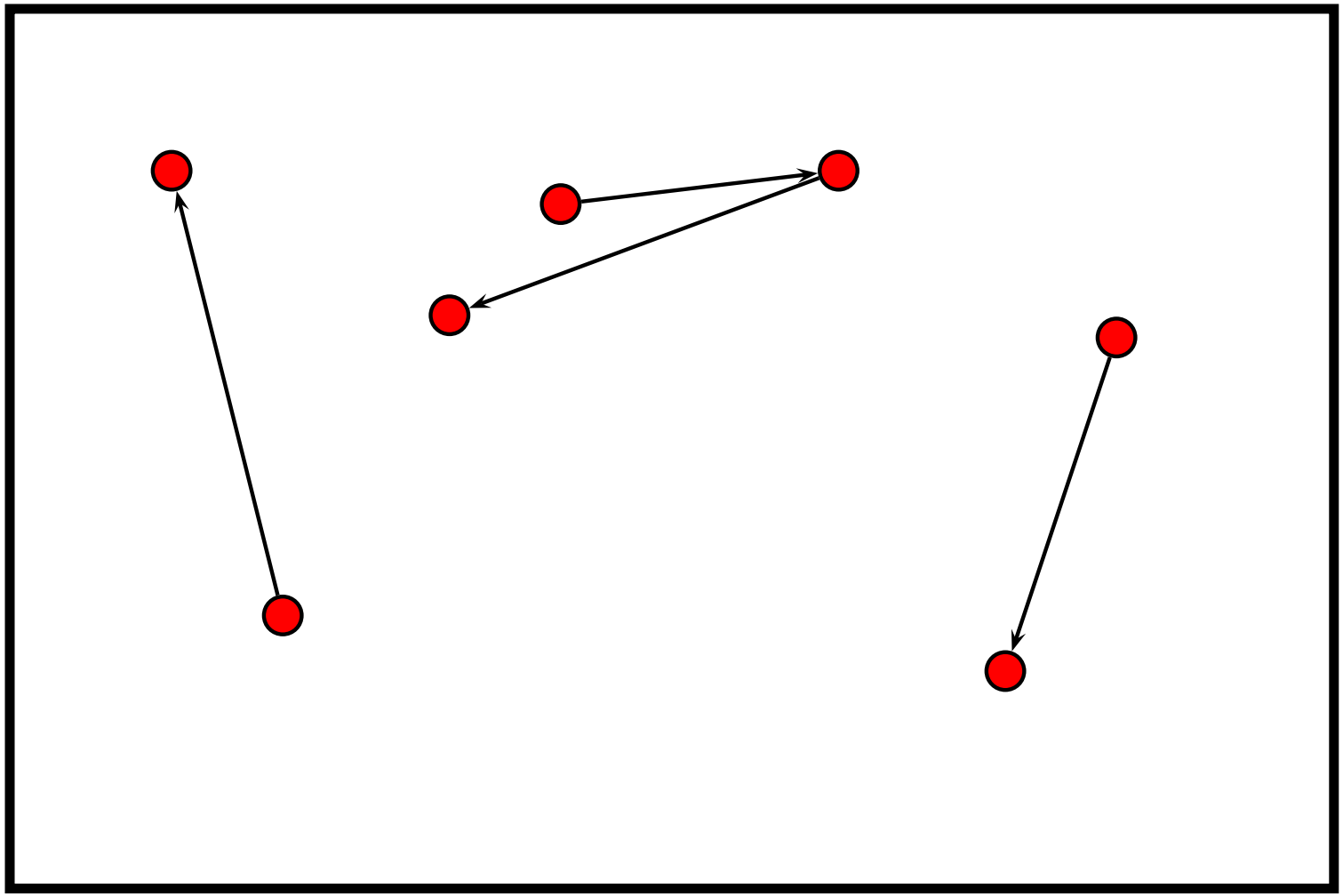


- Precise reachability
  - $\mathcal{R}_0 = I$
  - $\mathcal{R}_i = Img(R, \mathcal{R}_{i-1}) \cup \mathcal{R}_{i-1}$
- Interpolation based reachability
  - $Itp_0 = I(X_1)$
  - $Itp_i = Itp(R, Itp_{i-1}, k) \cup Itp_{i-1}$

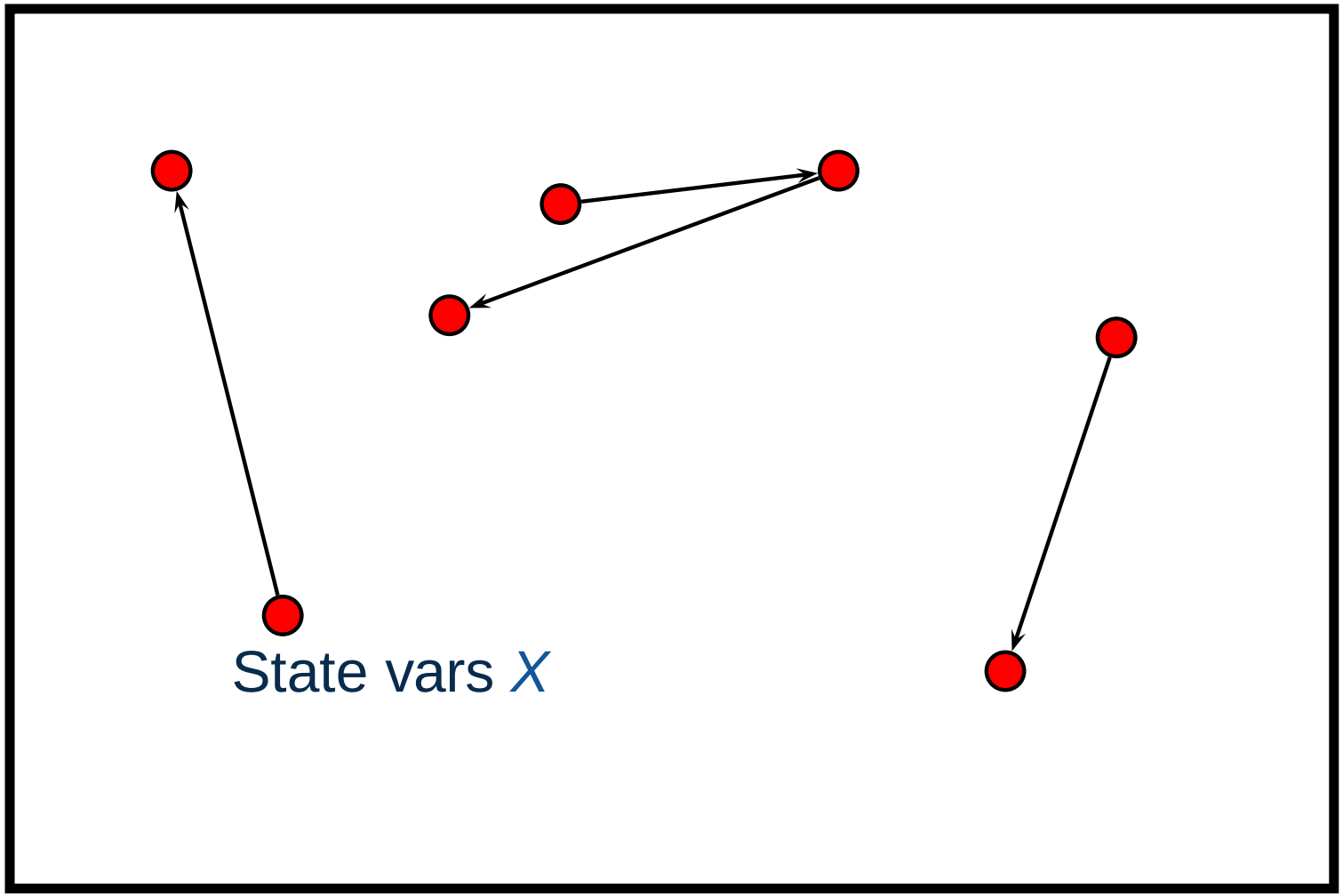
CEGAR loop


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

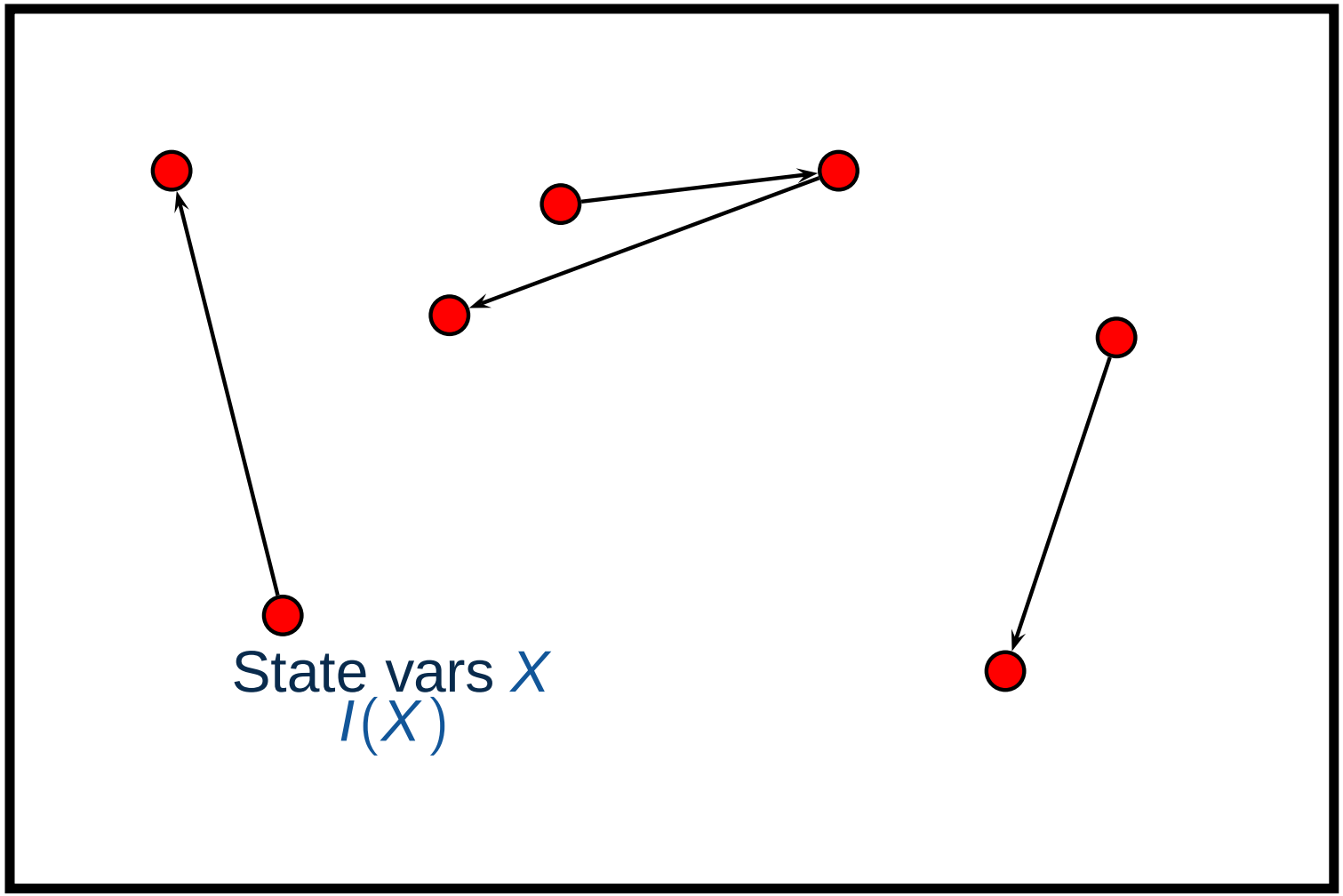
CEGAR based on Predicate Abstraction



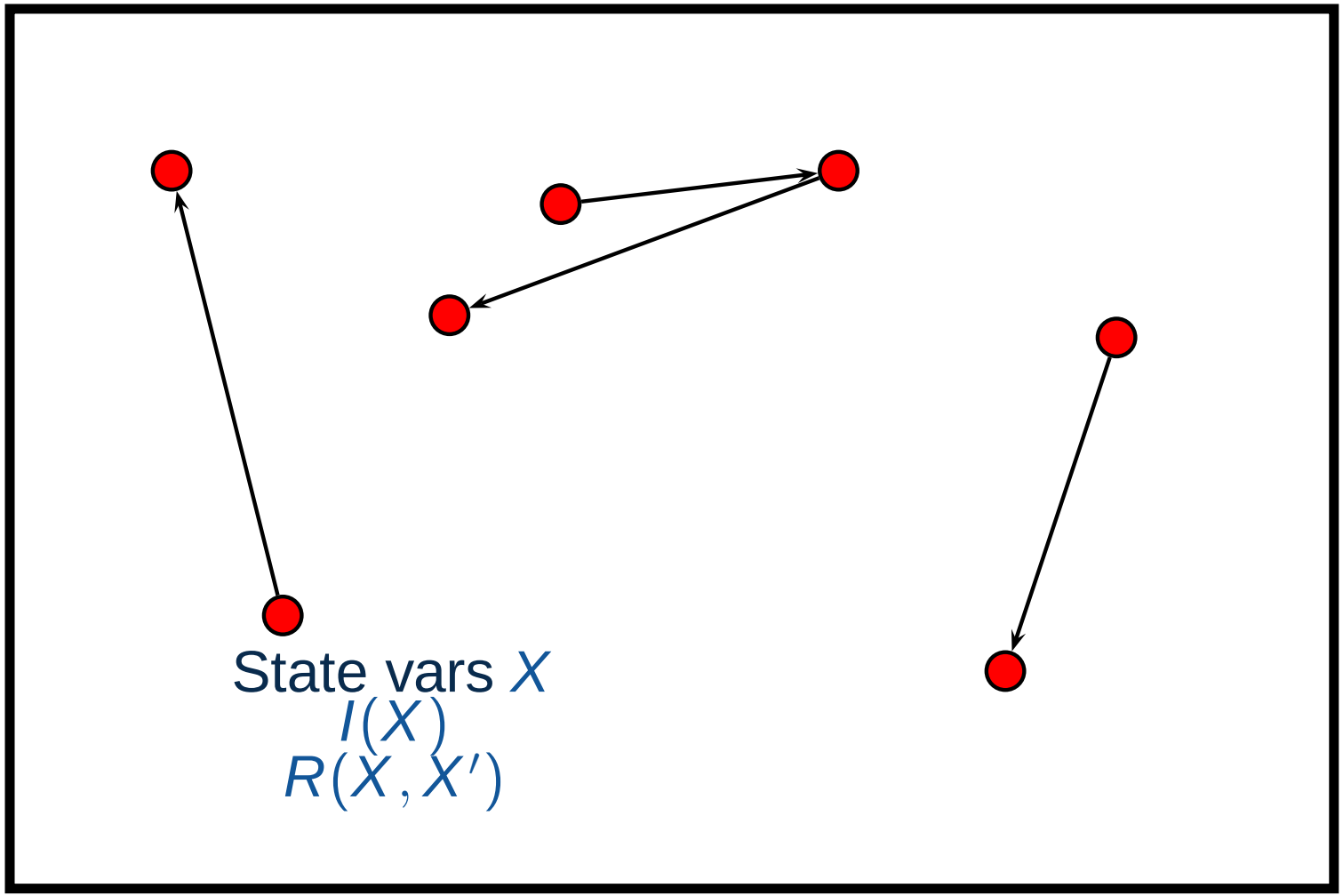
CEGAR based on Predicate Abstraction



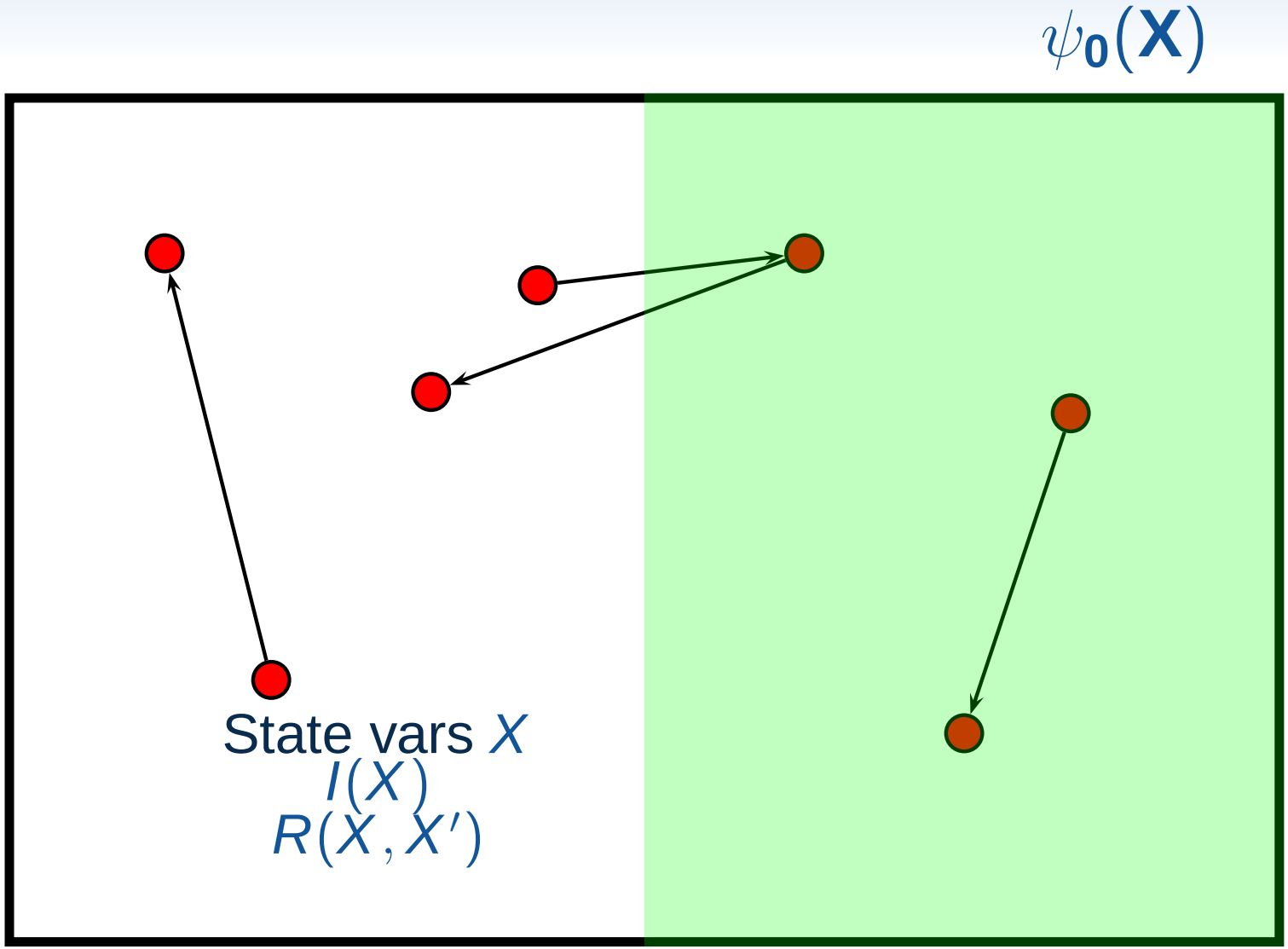
CEGAR based on Predicate Abstraction



CEGAR based on Predicate Abstraction

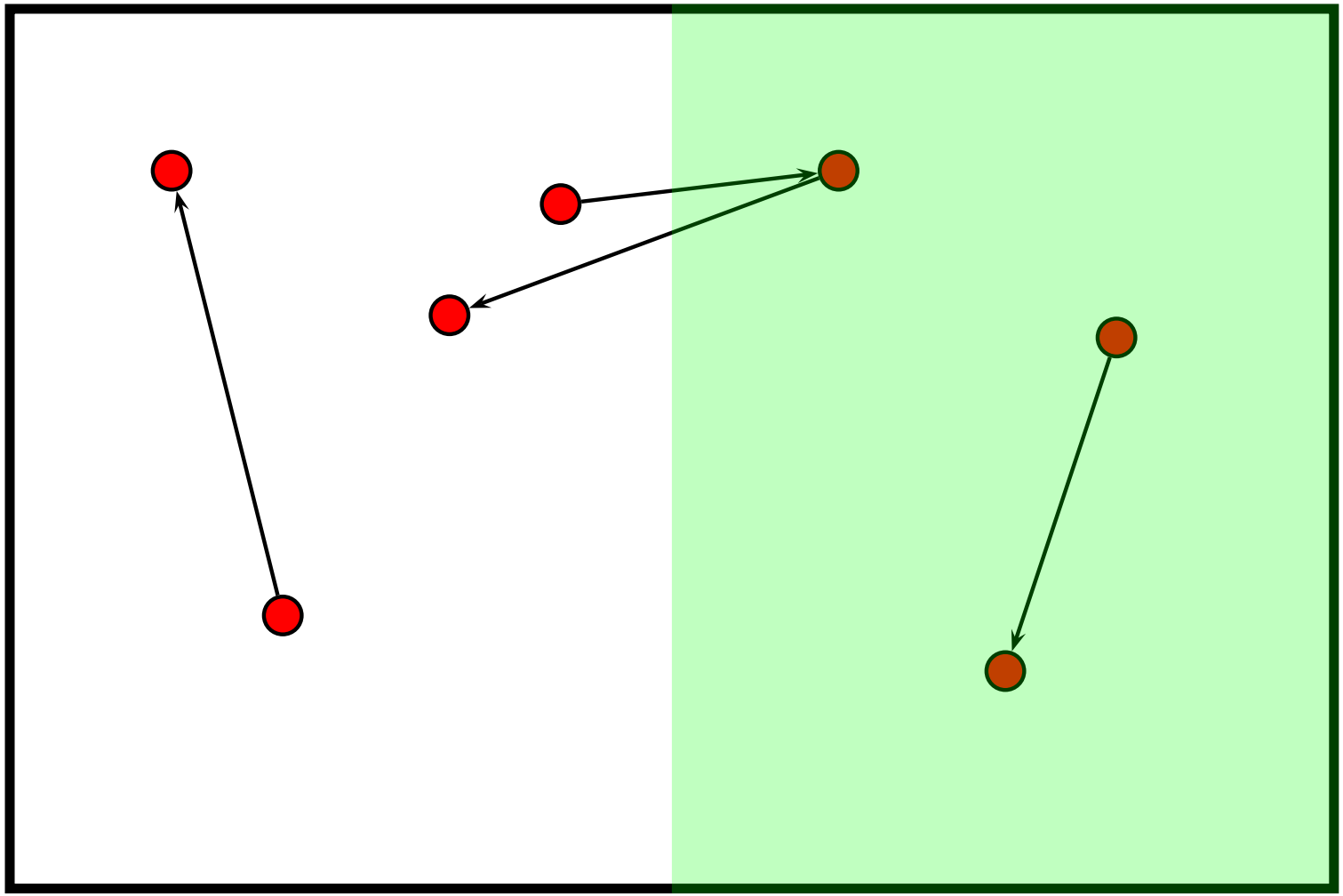


CEGAR based on Predicate Abstraction



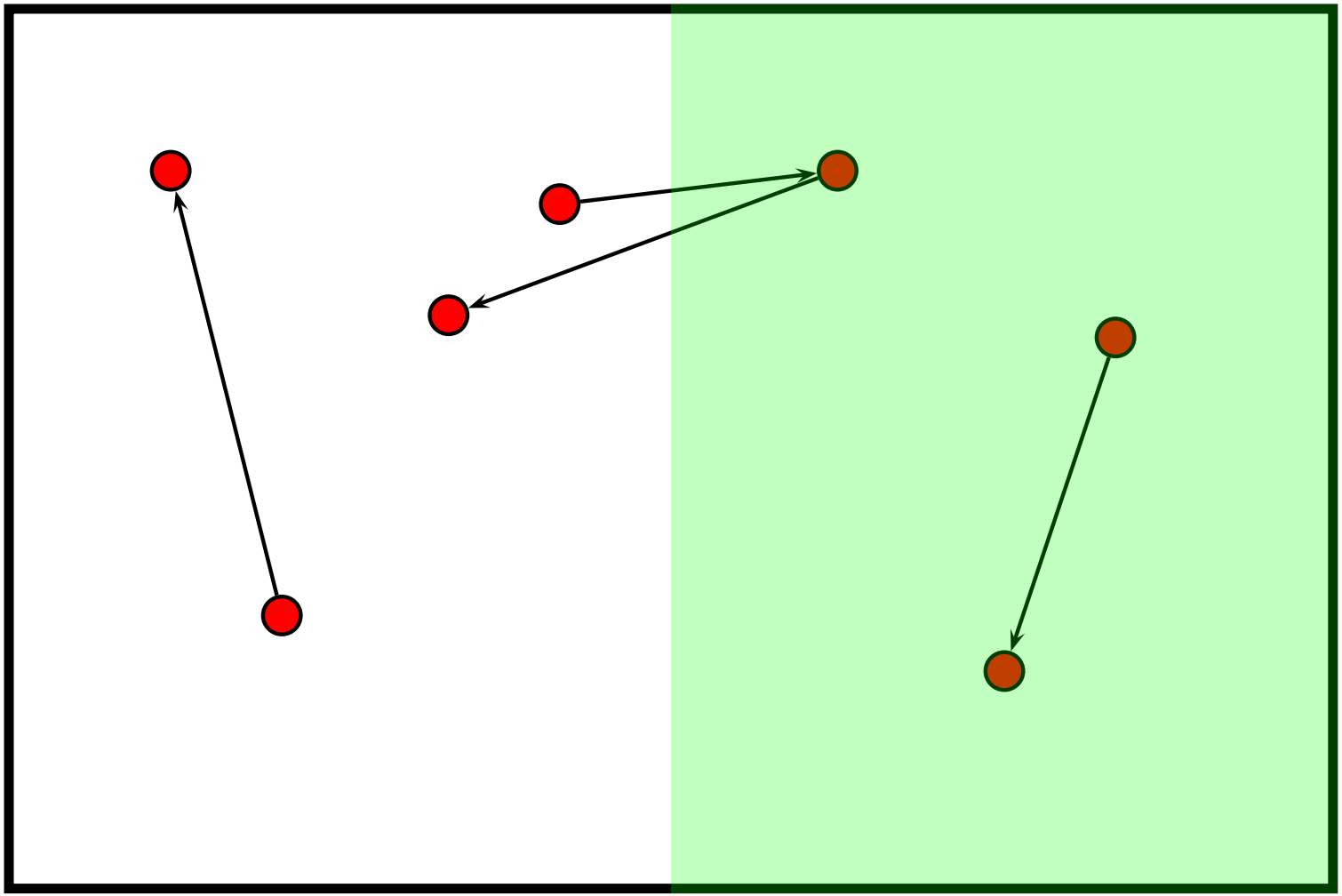
CEGAR based on Predicate Abstraction

$\psi_0(\mathbf{X})$

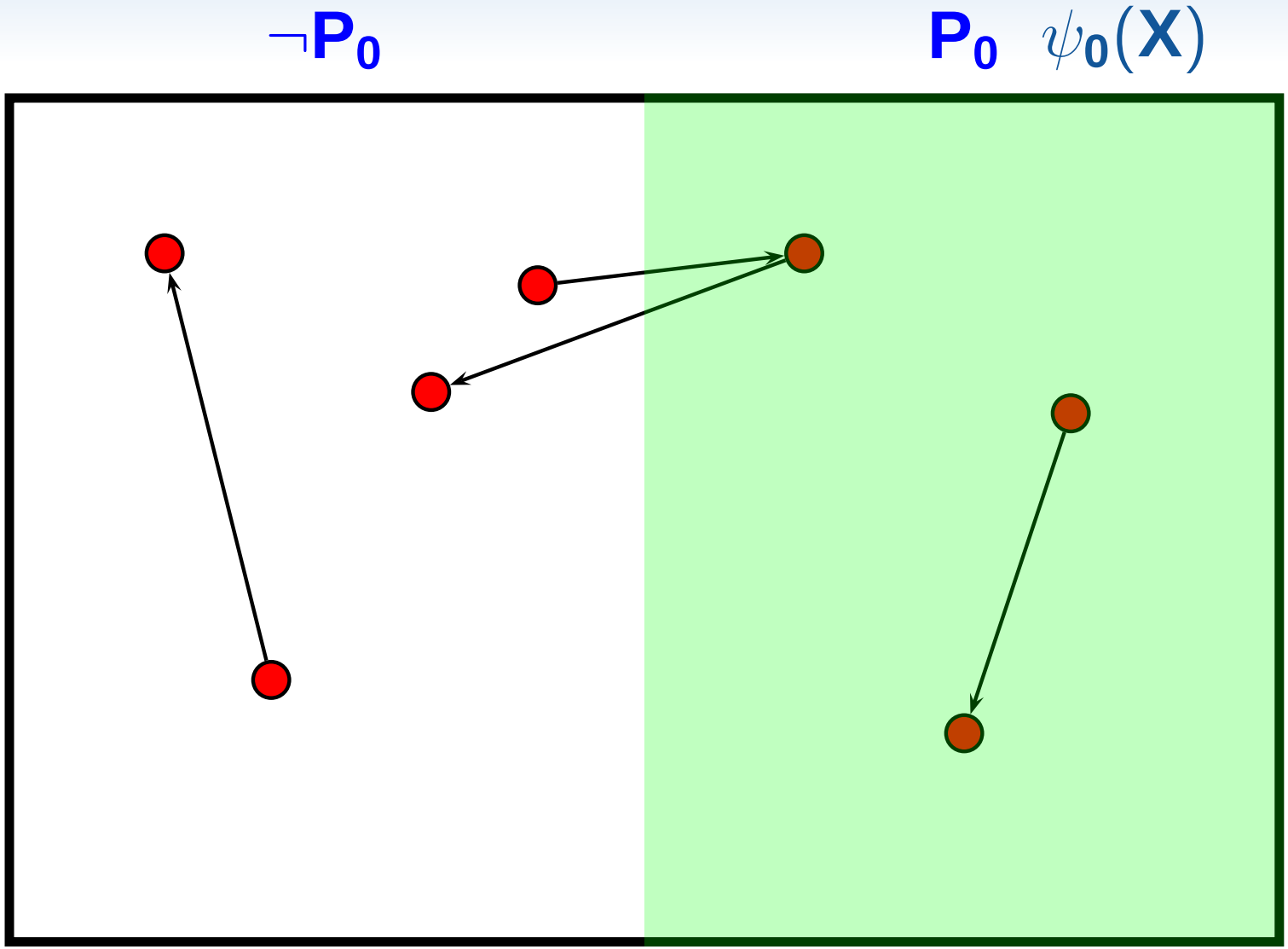


CEGAR based on Predicate Abstraction

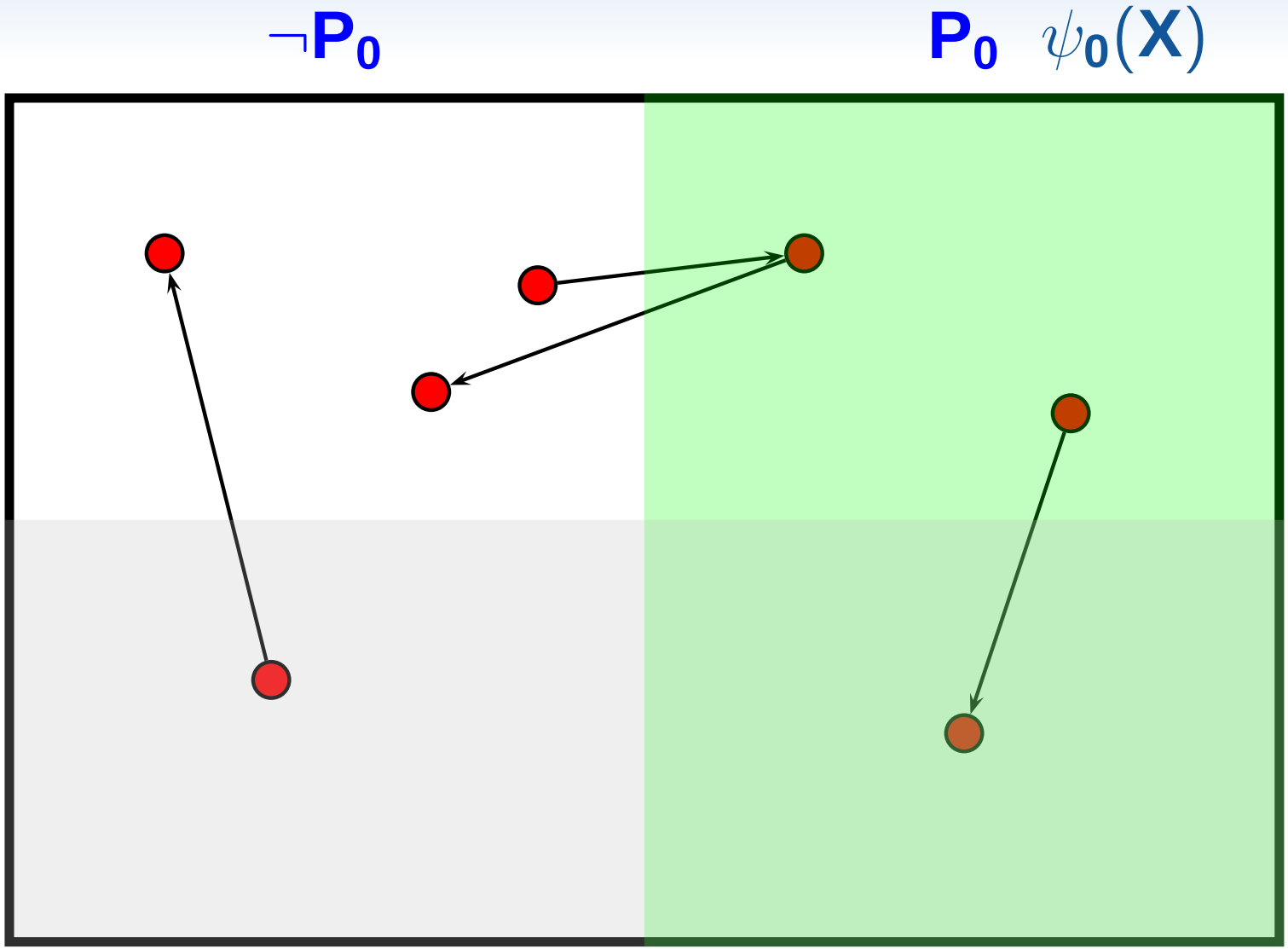
$P_0 \quad \psi_0(X)$



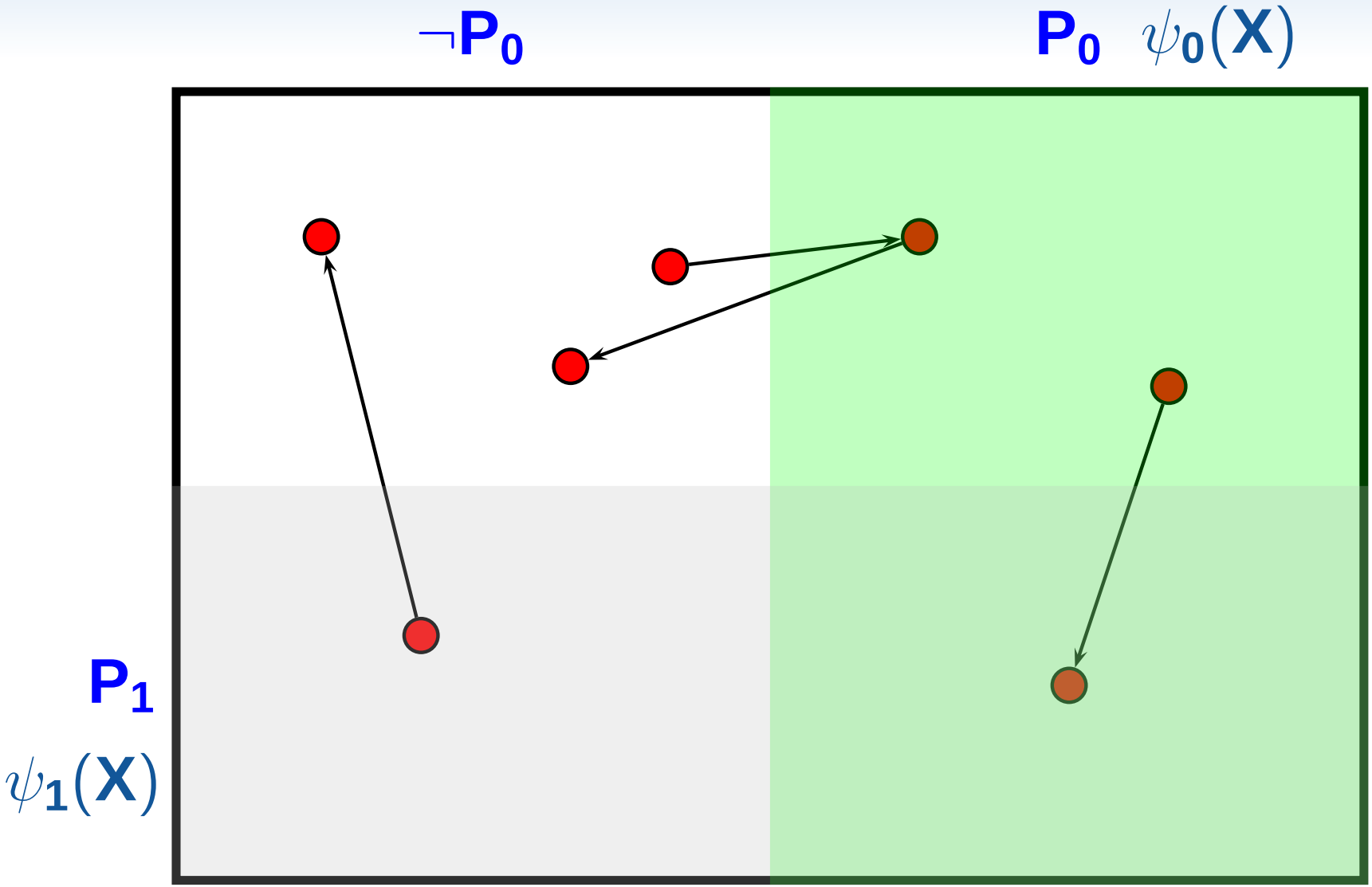
CEGAR based on Predicate Abstraction



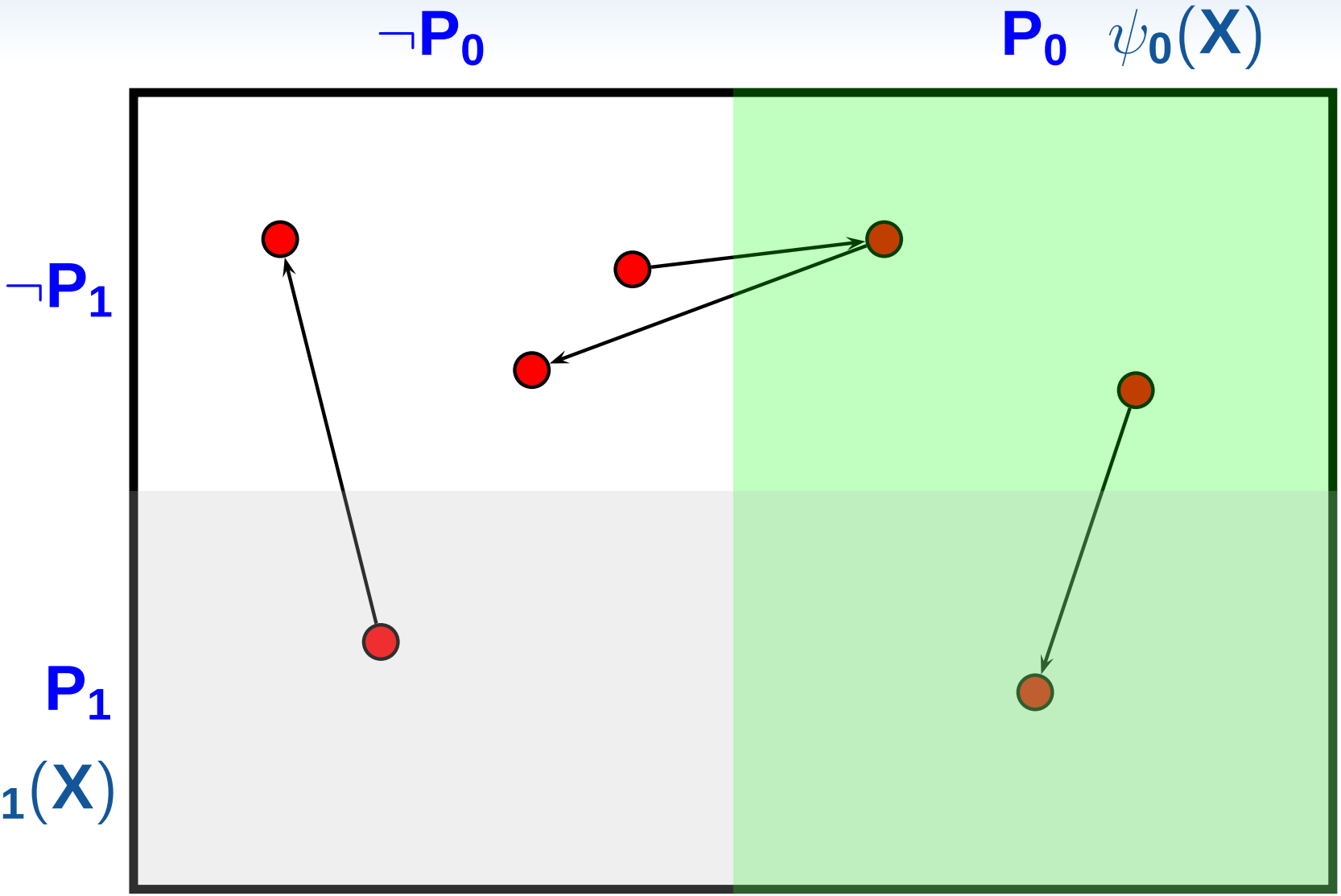
CEGAR based on Predicate Abstraction



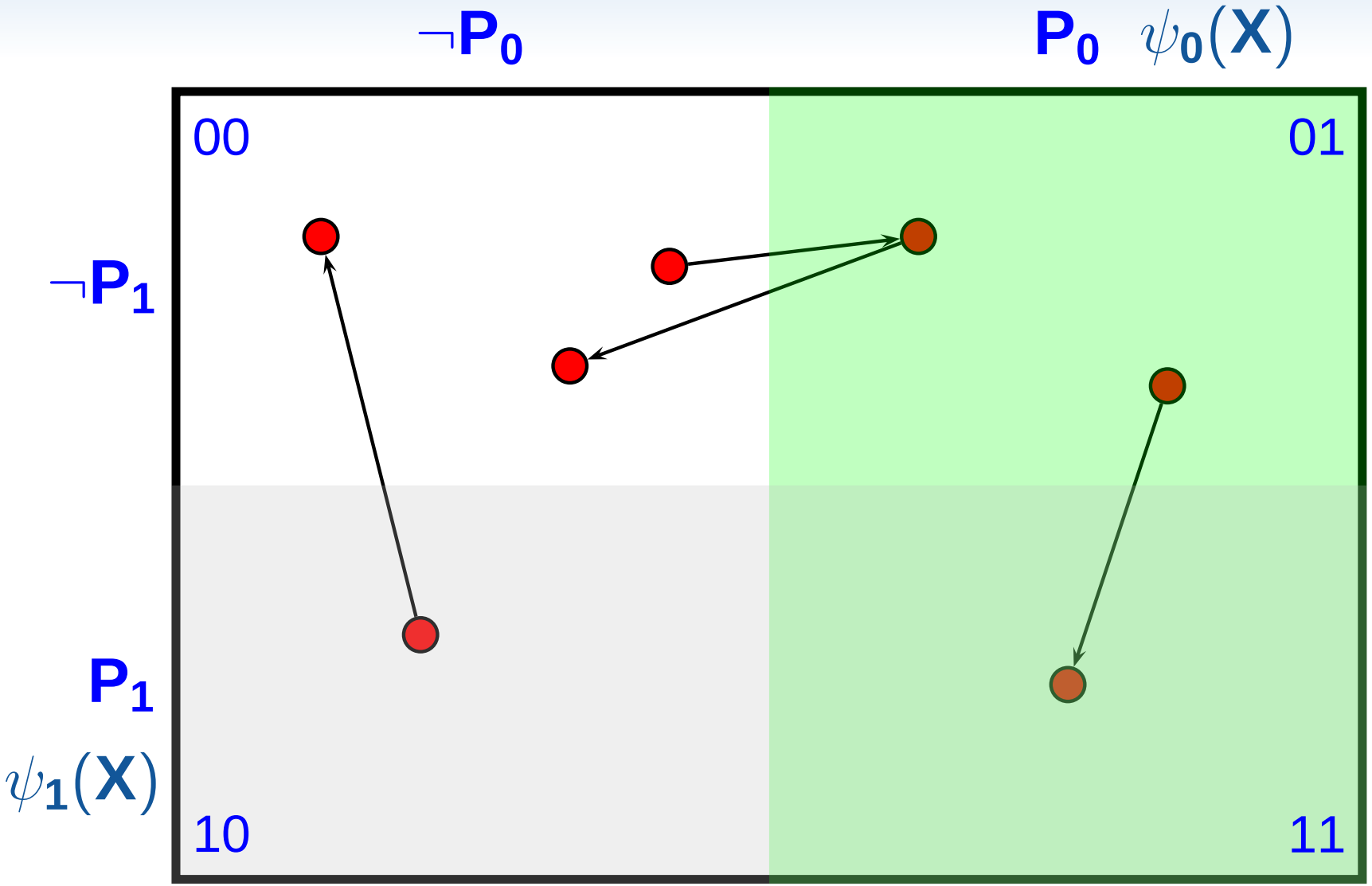
CEGAR based on Predicate Abstraction



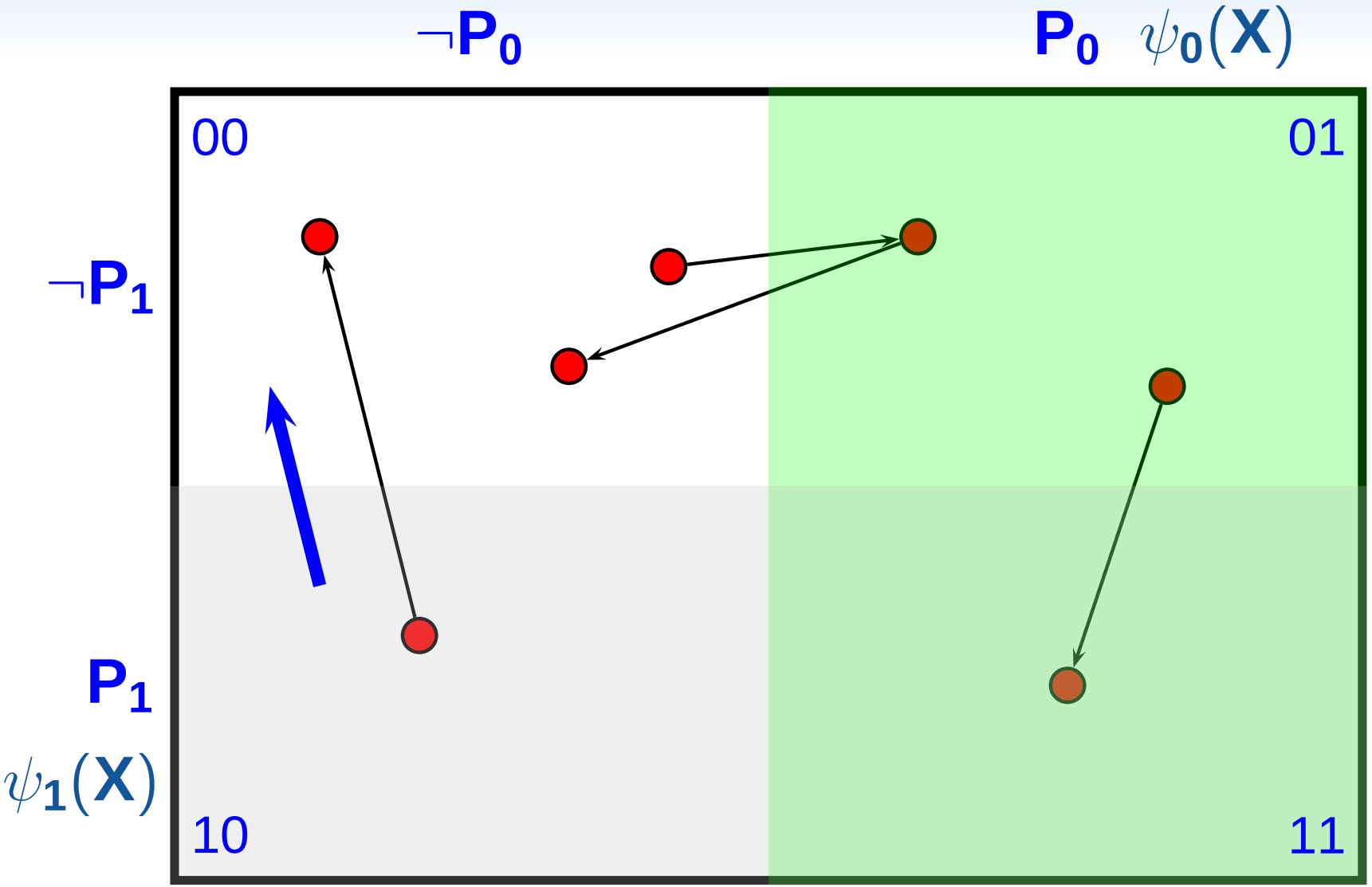
CEGAR based on Predicate Abstraction



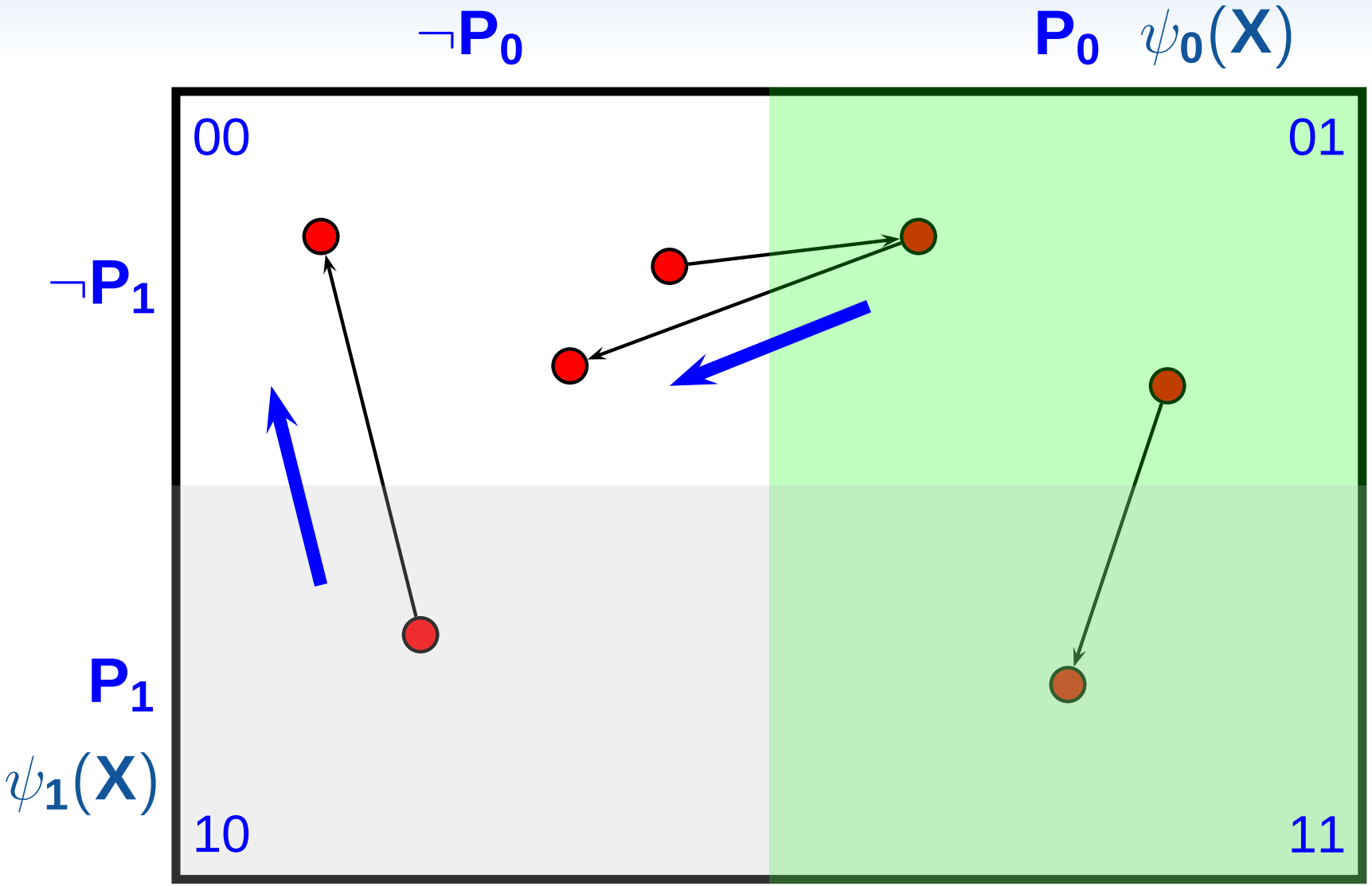
CEGAR based on Predicate Abstraction



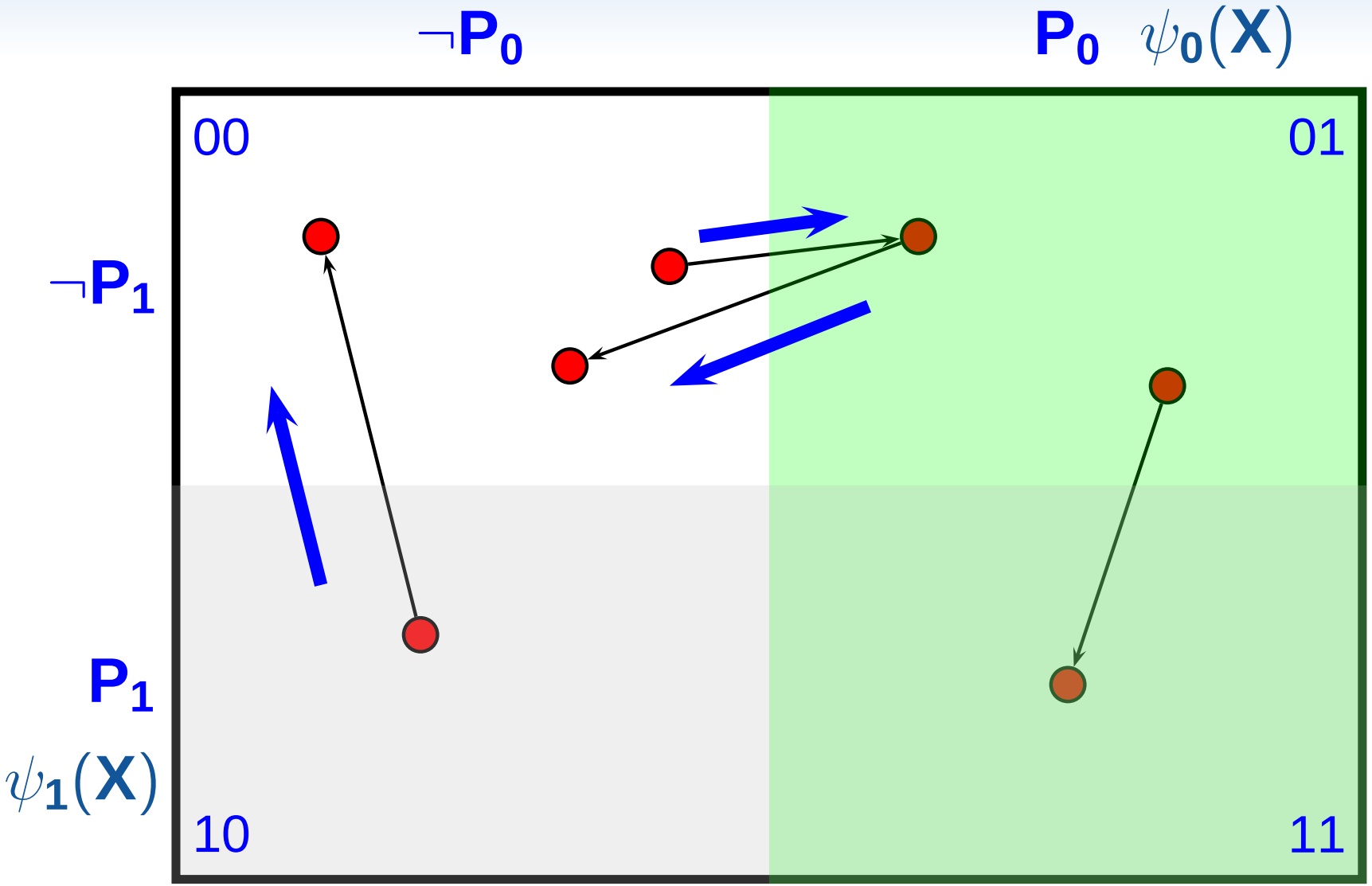
CEGAR based on Predicate Abstraction



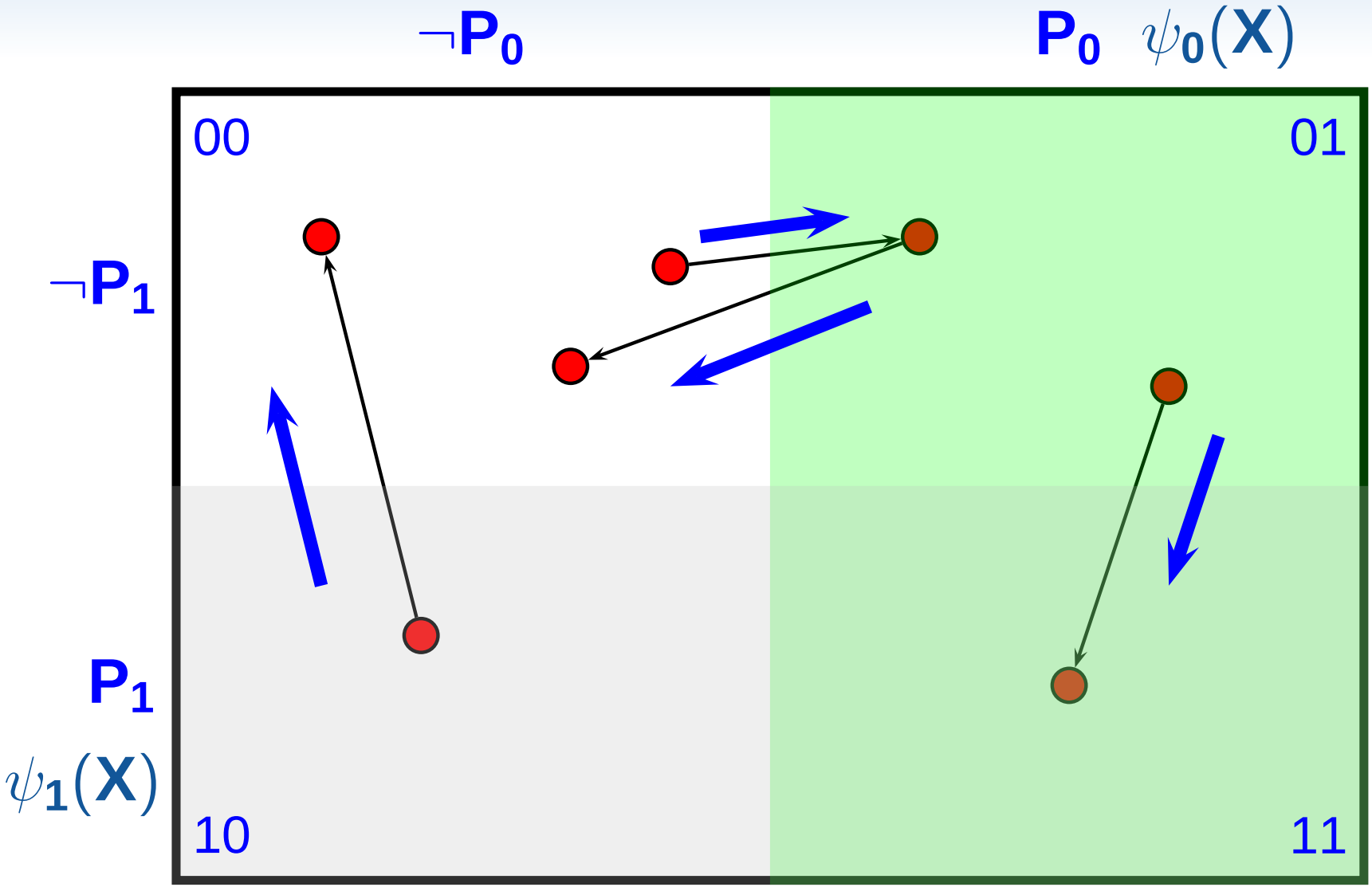
CEGAR based on Predicate Abstraction



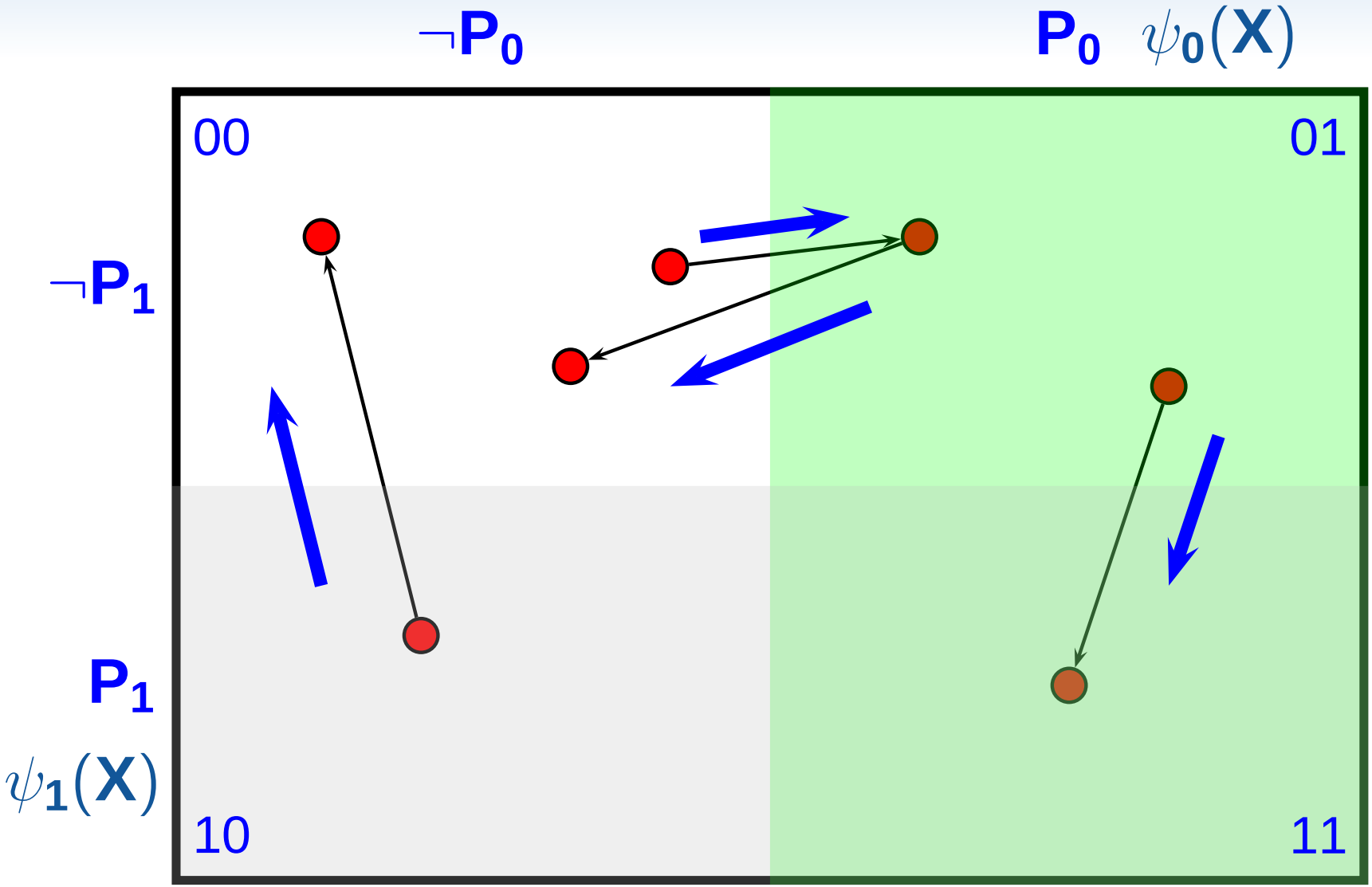
CEGAR based on Predicate Abstraction



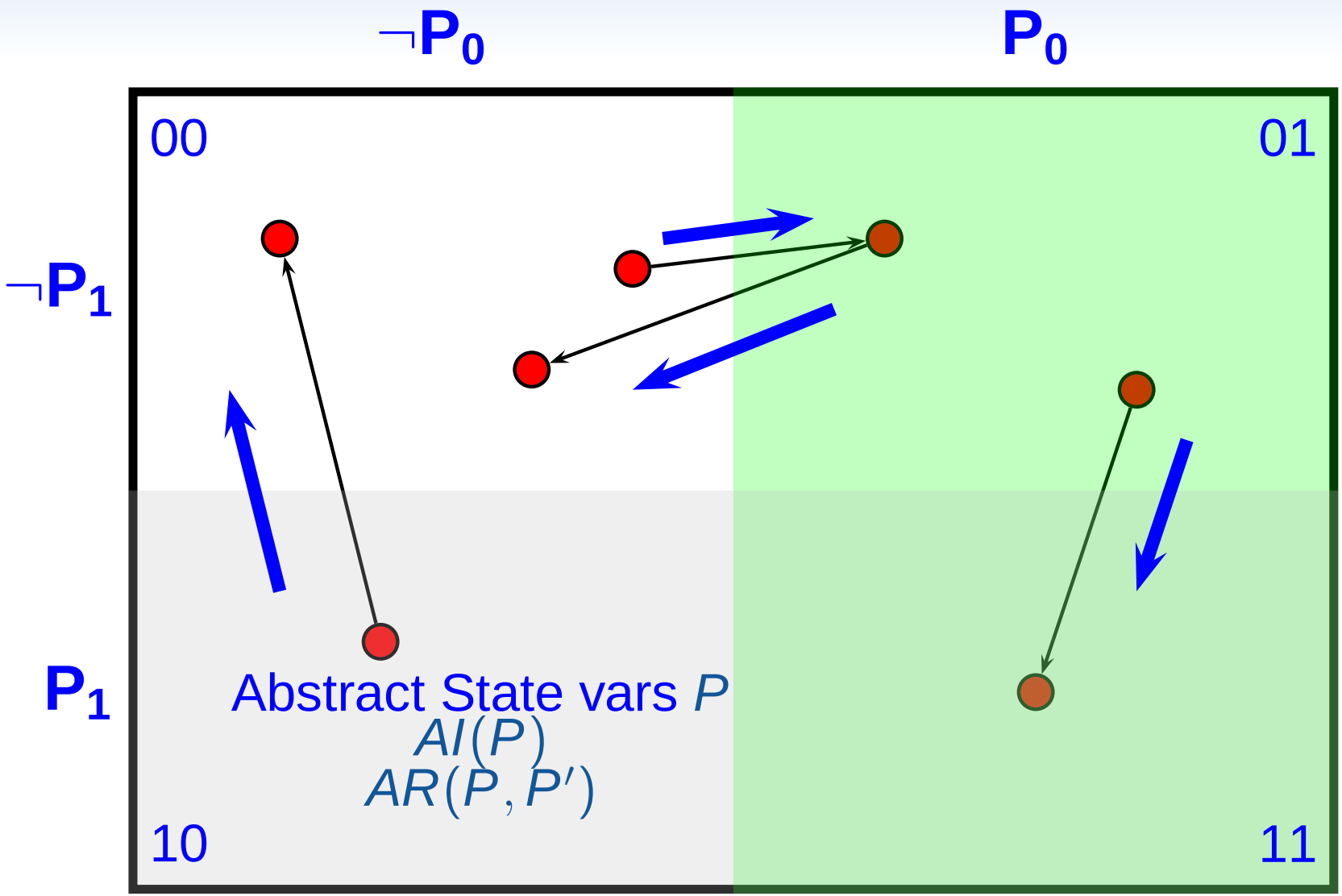
CEGAR based on Predicate Abstraction



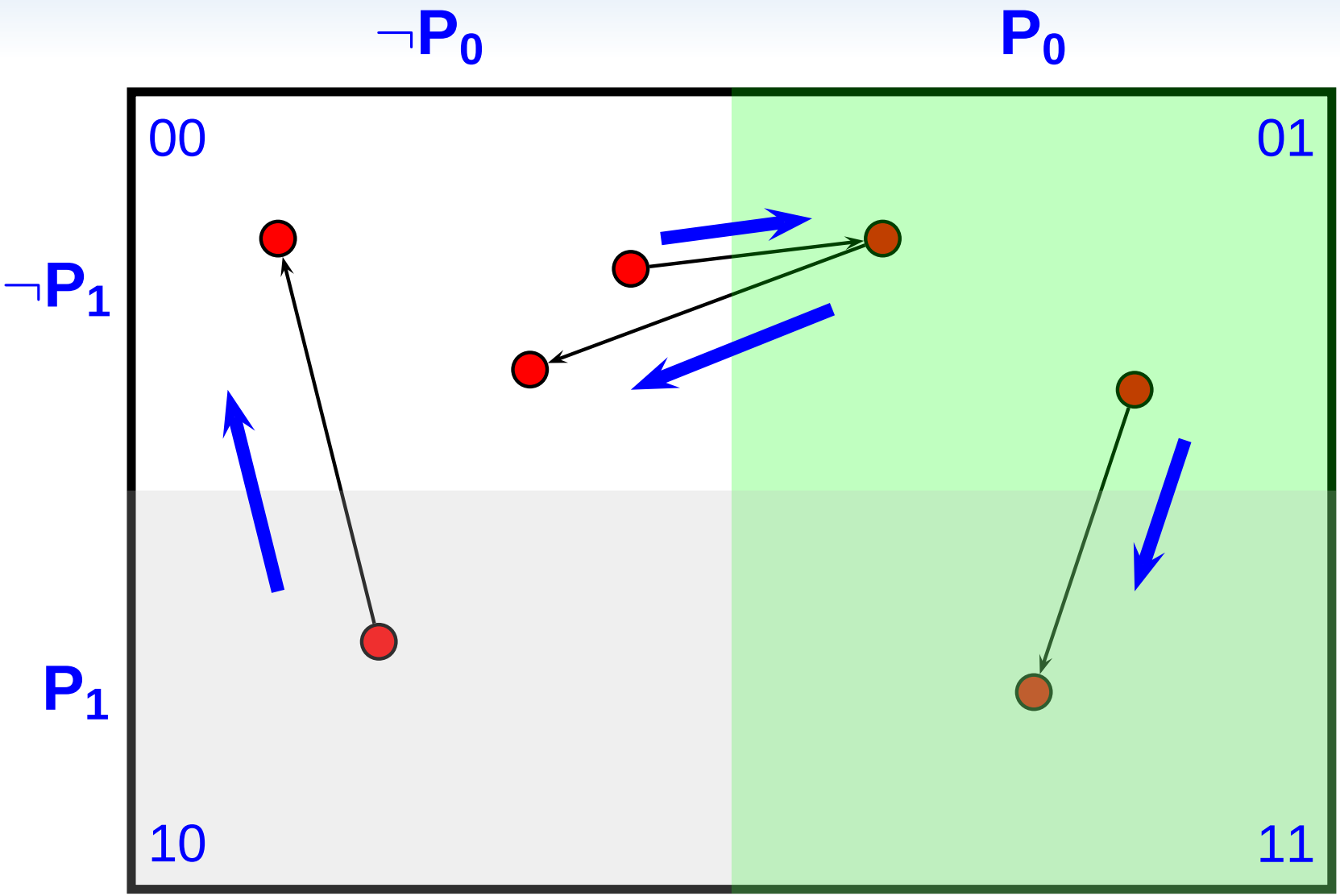
CEGAR based on Predicate Abstraction



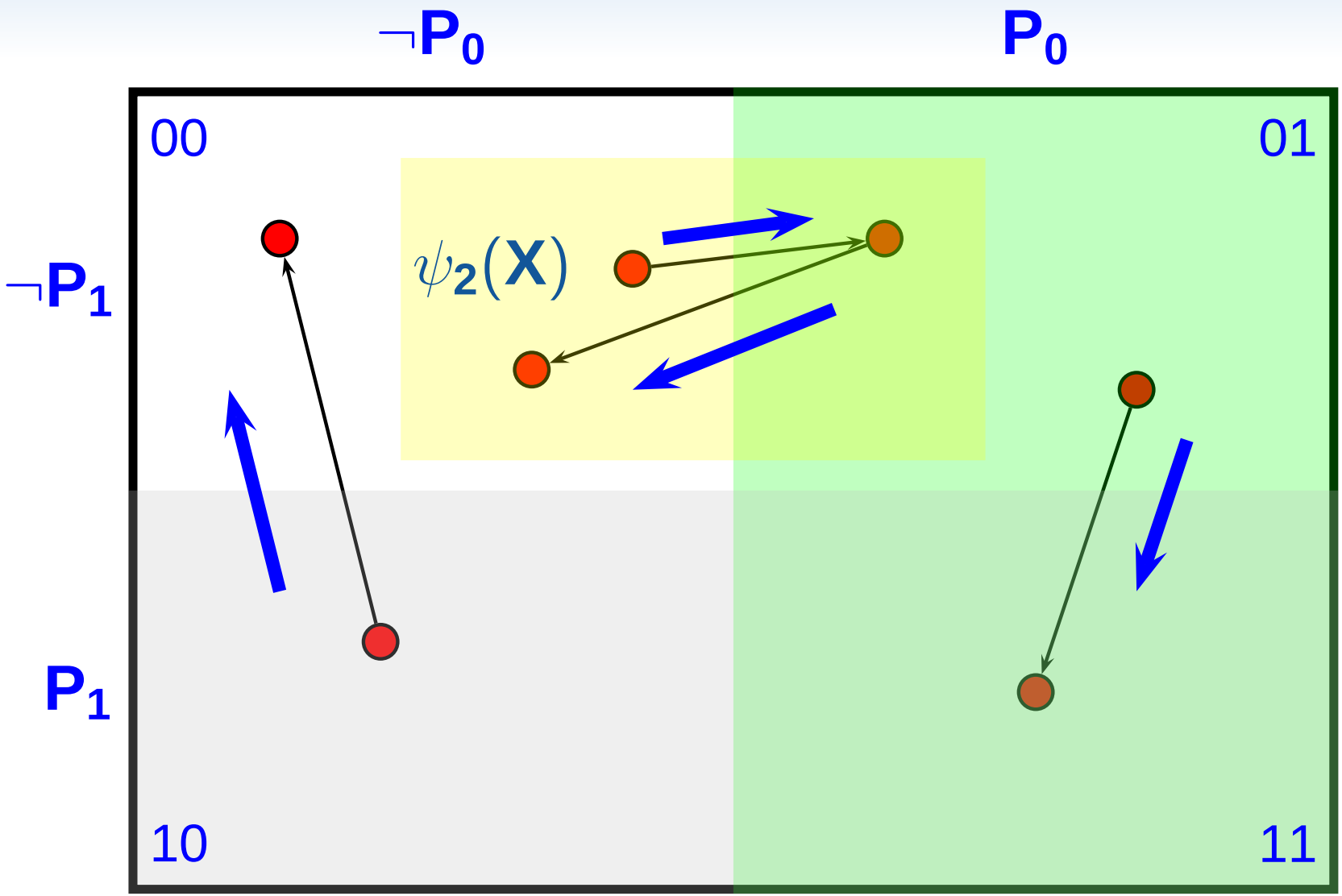
CEGAR based on Predicate Abstraction



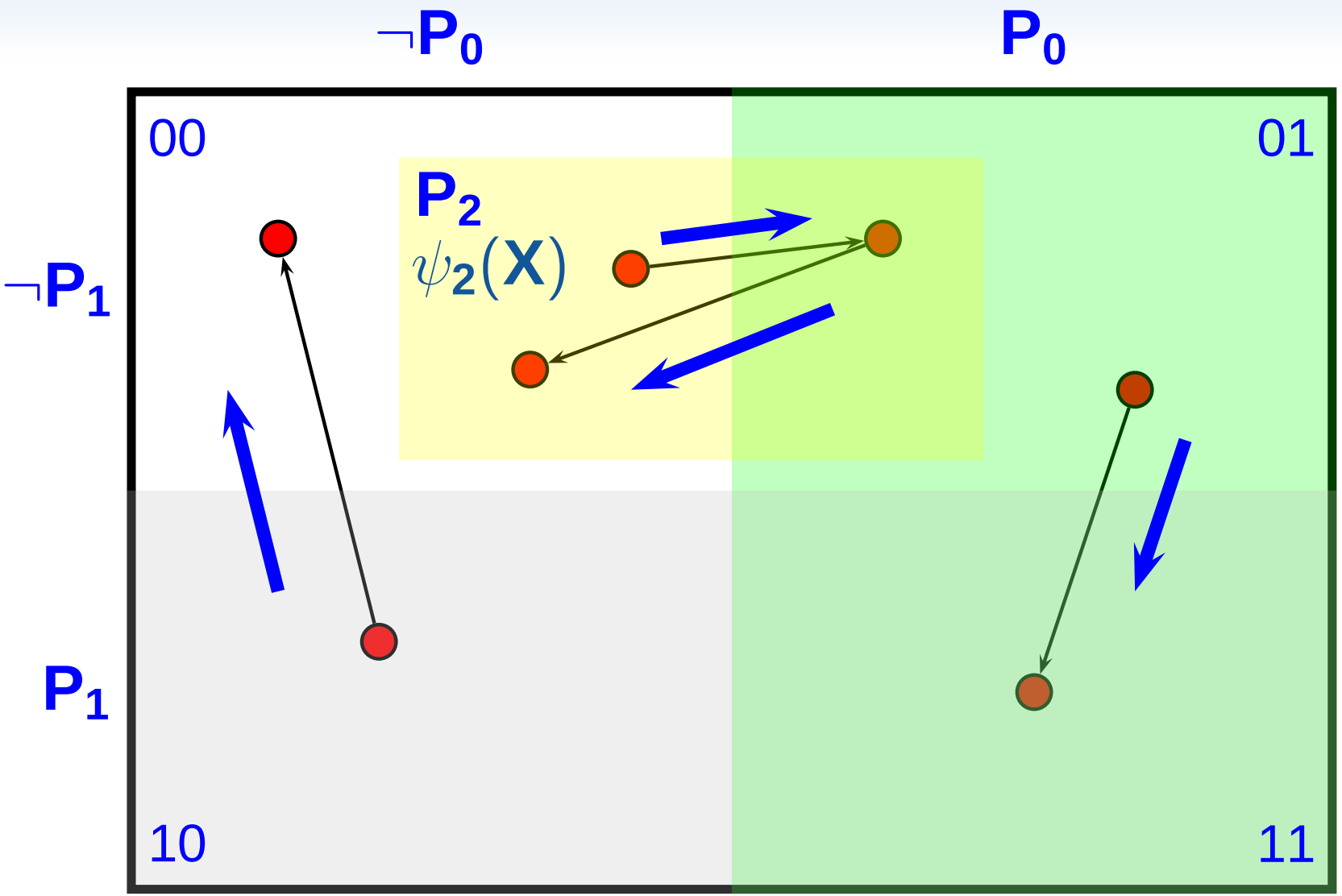
CEGAR based on Predicate Abstraction



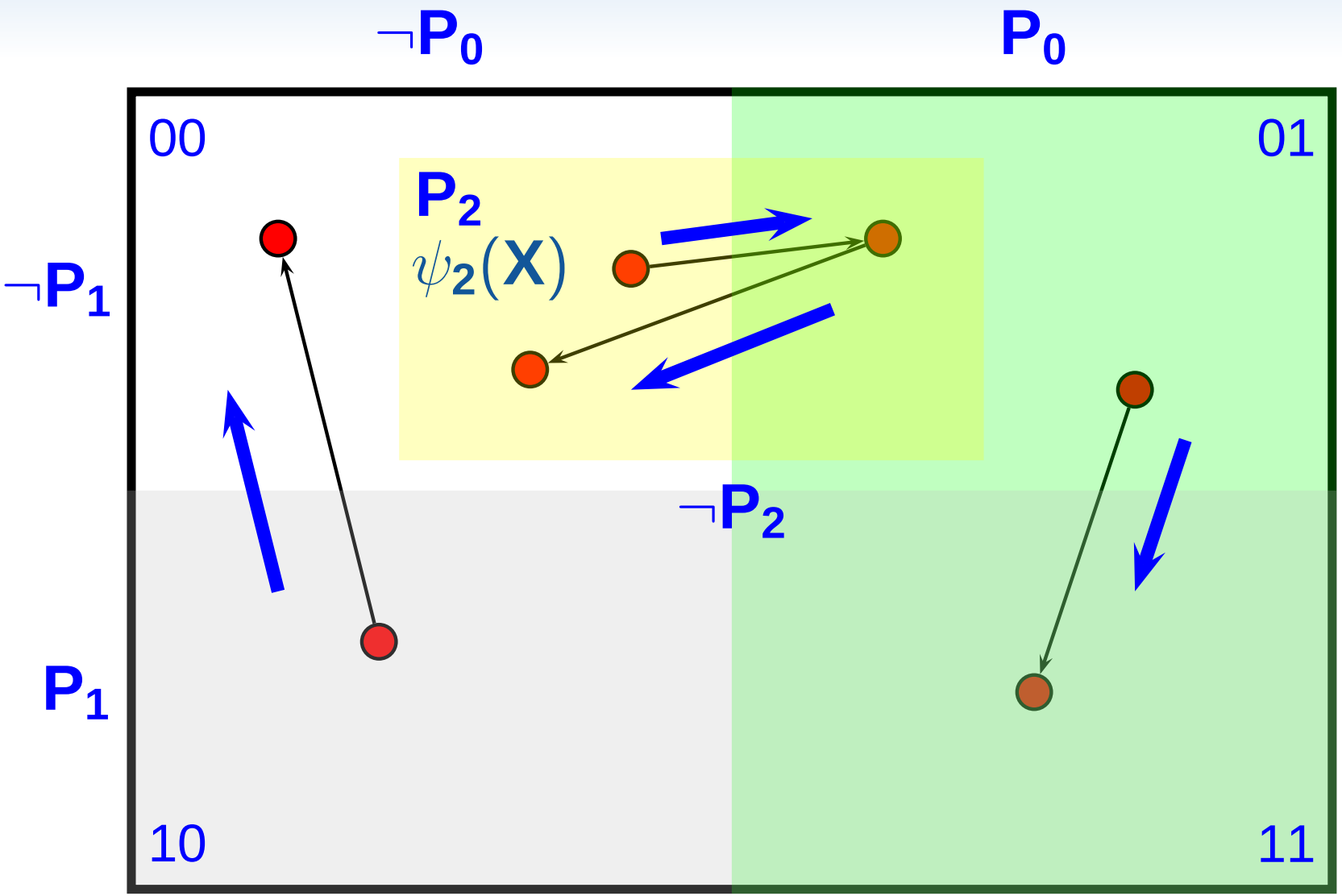
CEGAR based on Predicate Abstraction



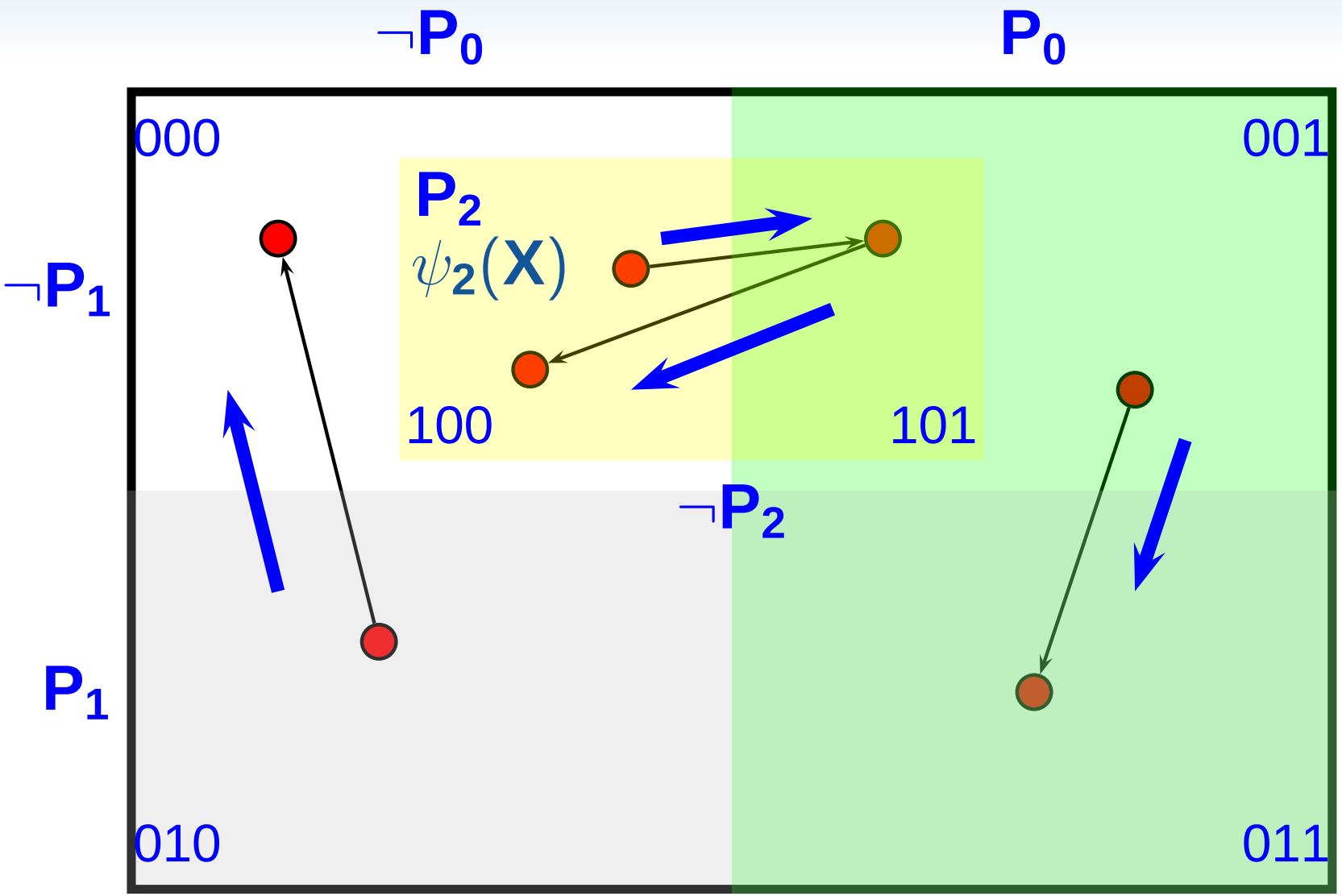
CEGAR based on Predicate Abstraction



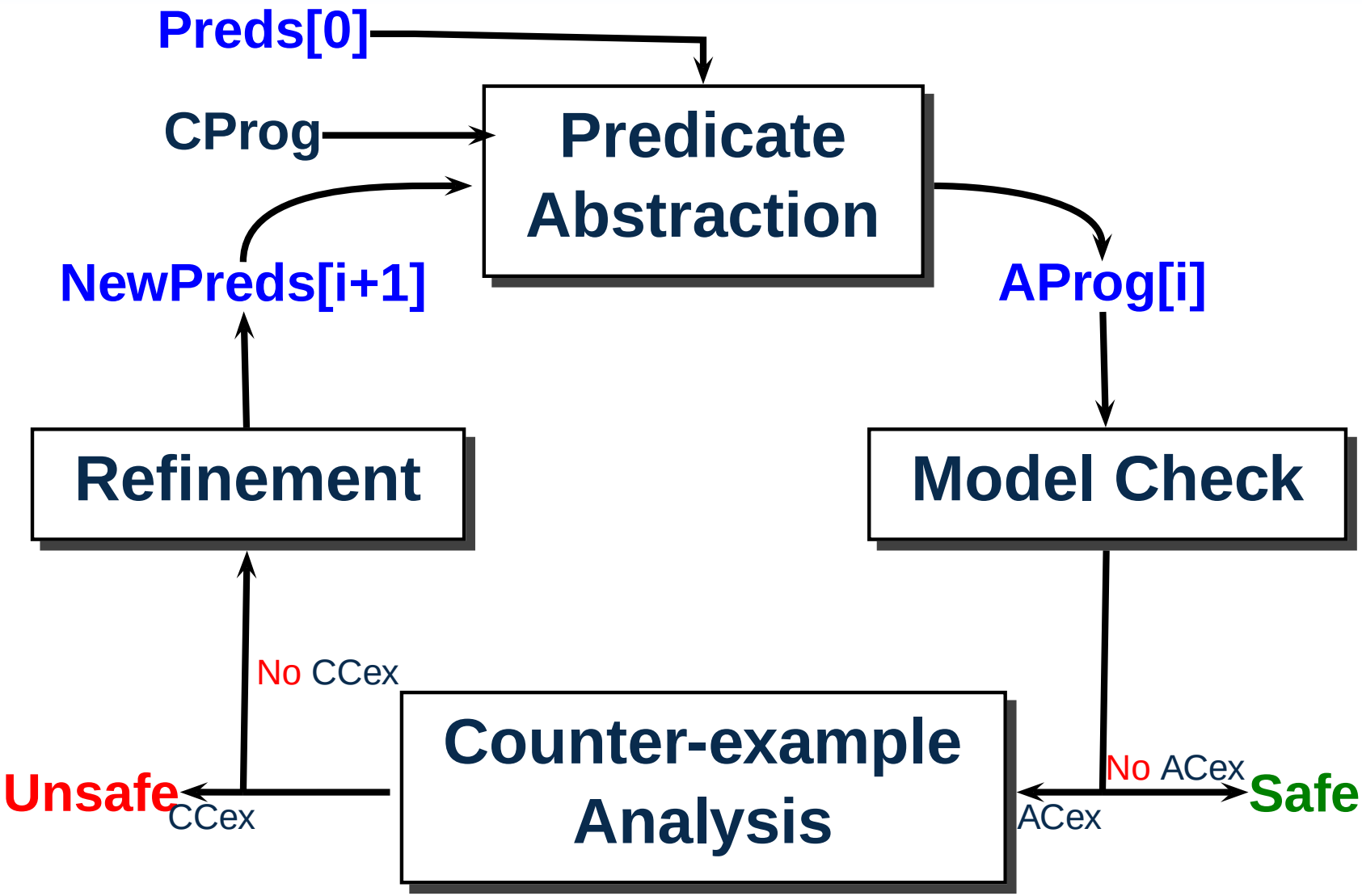
CEGAR based on Predicate Abstraction



CEGAR based on Predicate Abstraction



CEGAR with Predicate Abstraction



SMT-based Predicate Abstraction

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

- 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

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: $\bigwedge_{0 \leq h \leq k-1} R(X_h, X'_h) \wedge EQ_\alpha(X'_h, X_{h+1})$

Beyond NuSMV

NuSMV2

- https://nusmv.fbk.eu/
- since 1997 BDD-based and SAT-based reasoning

Extended NuSMV

- extended types: integers, reals
- actually a new system, using “base NuSMV” as a library
- integrated with MathSAT
- connections to other design languages AADL, Altarica, Simulink
- applied in several projects: FP VII, ESA, ERA
- dedicated language for structured hybrid systems

Outline

Programs as CFAs

Programs are represented as control-flow automata (CFAs).

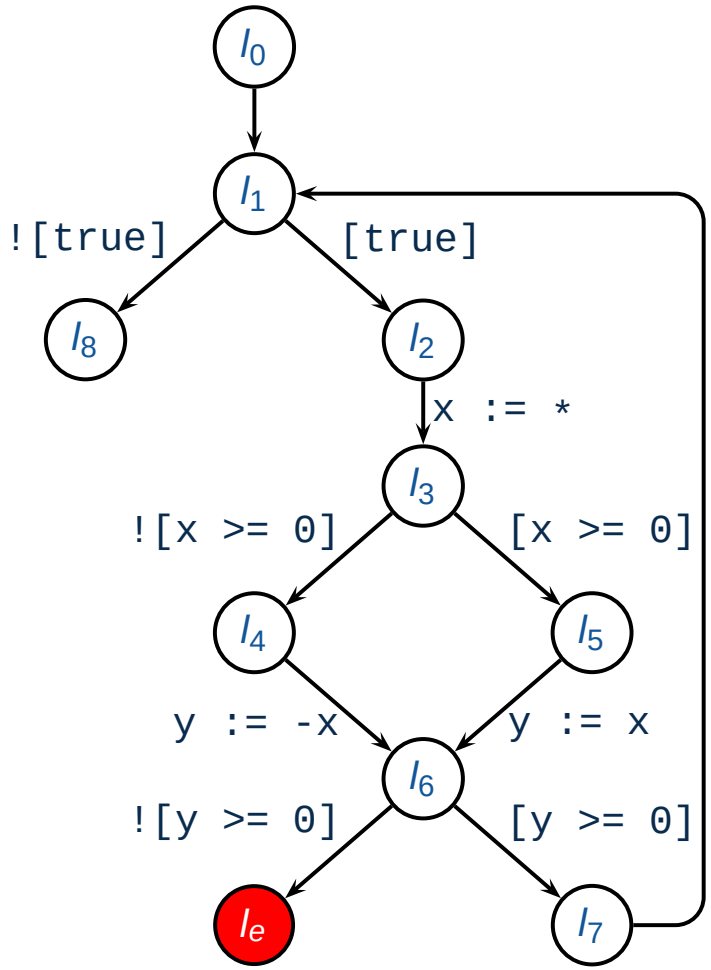
- A CFA for program P is a pair (L, G)
  - L is a set of program locations.
  - $G \subseteq L \times Op \times L$ is a set of edges.
  - l_0 is the unique entry location.
  - l_e is the unique (sink) error location.

Programs as CFAs

Programs are represented as control-flow automata (CFAs).

- A CFA for program P is a pair (L, G)
  - L is a set of program locations.
  - $G \subseteq L \times Op \times L$ is a set of edges.
  - l_0 is the unique entry location.
  - l_e is the unique (sink) error location.

```
while (1) {
  x = *;
  if (x >= 0) y = x;
  else y = -x;
  assert(y >= 0);
}
```



Lazy ART Construction

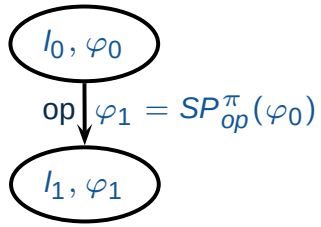
On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

1. Pick an ART node.

Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

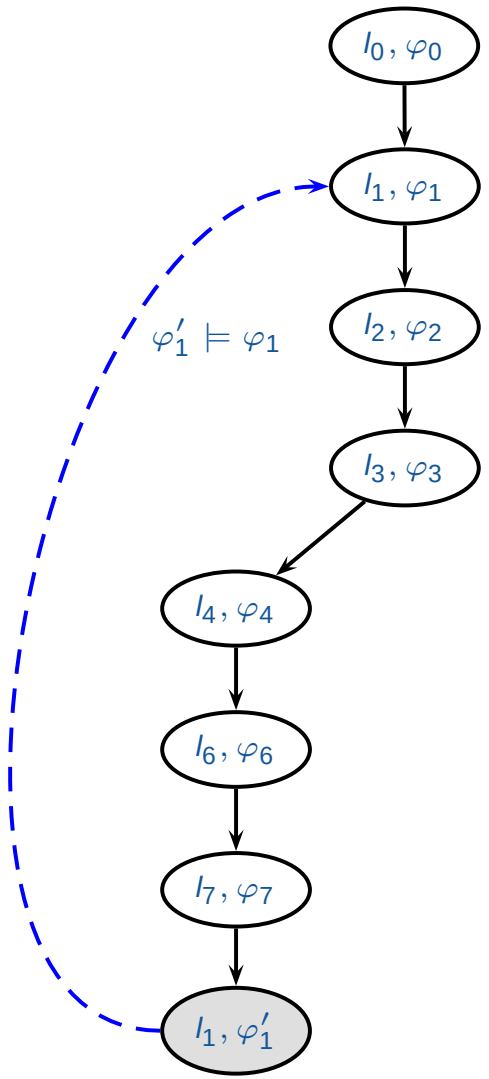
1. Pick an ART node.
2. Compute abstract successors,



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

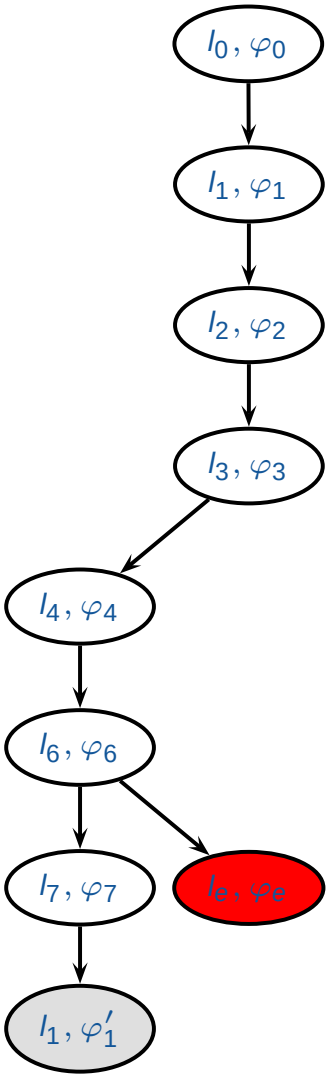
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

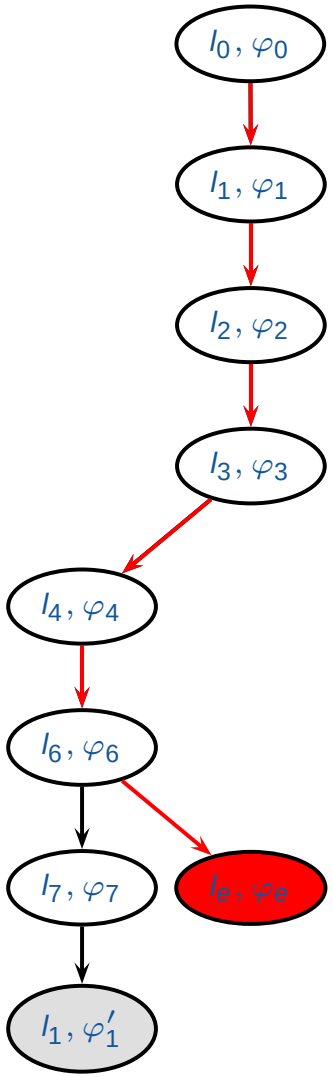
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

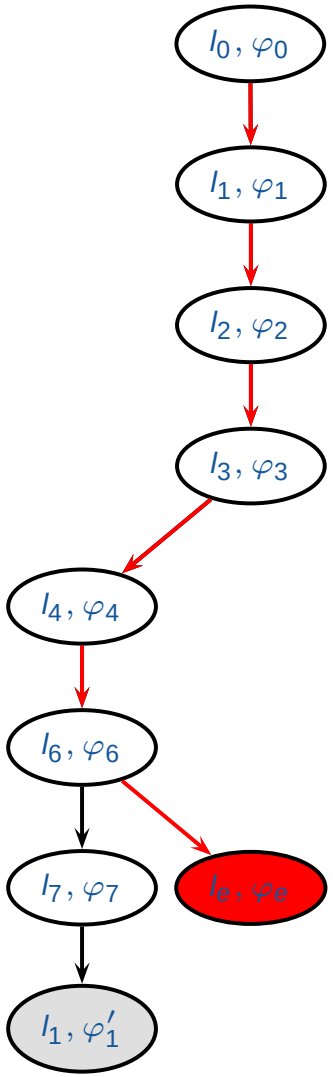
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

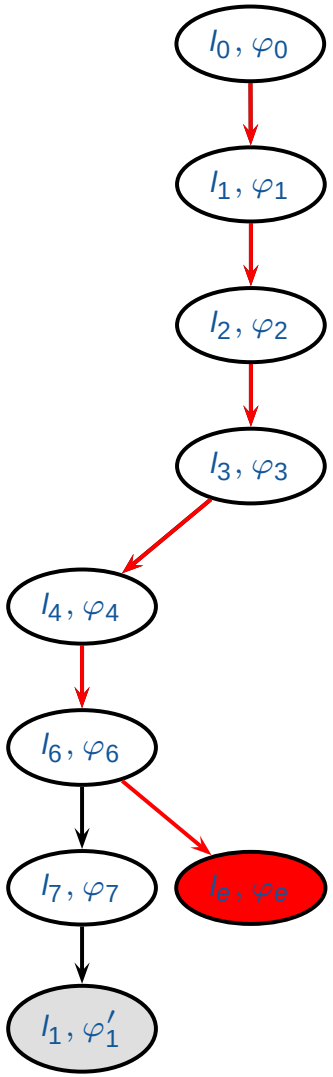
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

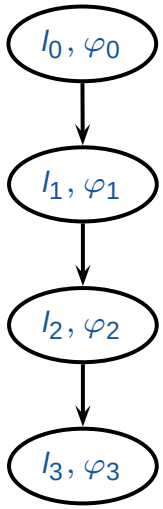
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

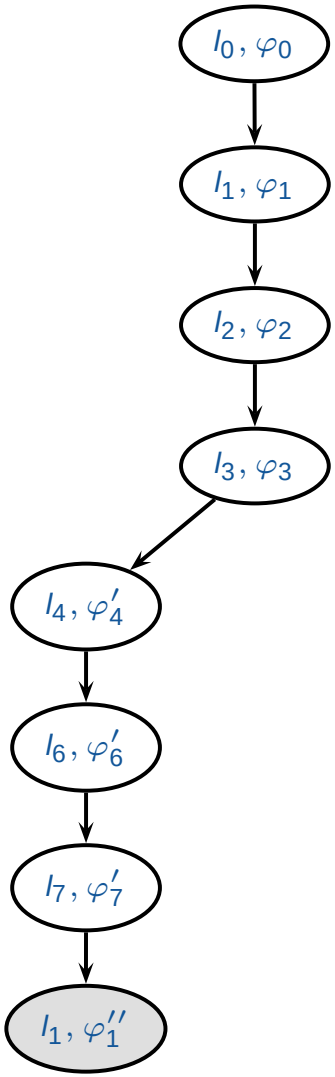
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

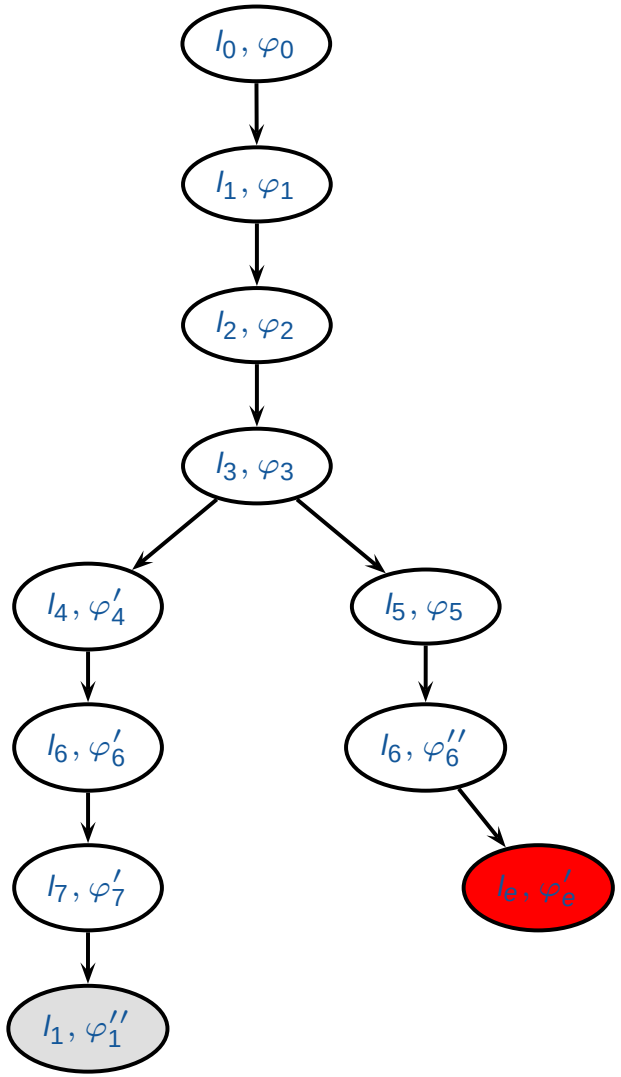
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.
     - Goto 1 to reconstruct subtree.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

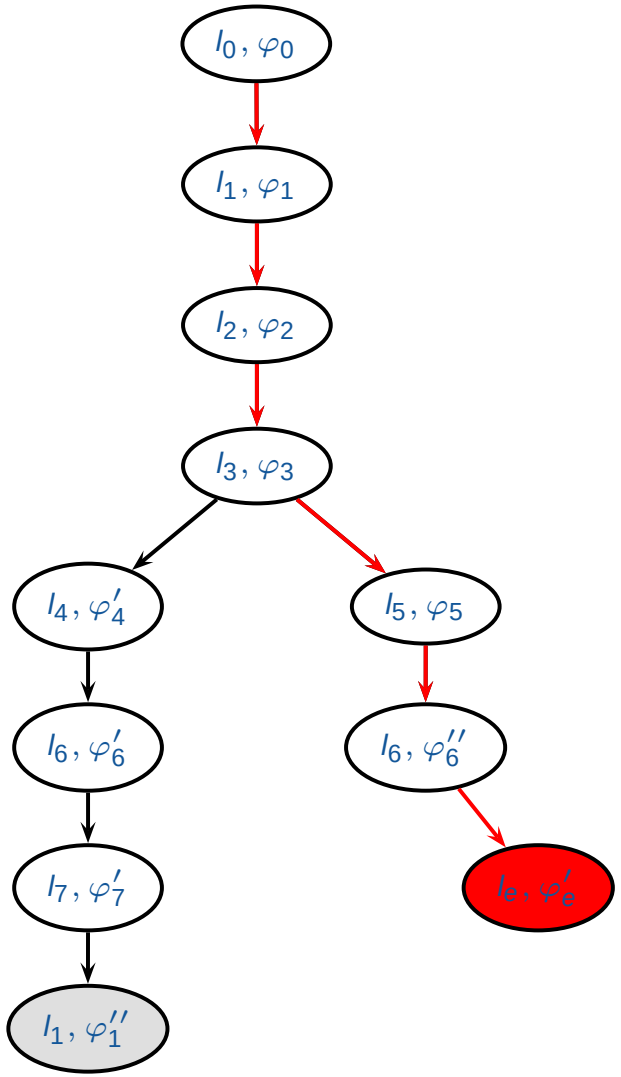
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.
     - Goto 1 to reconstruct subtree.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

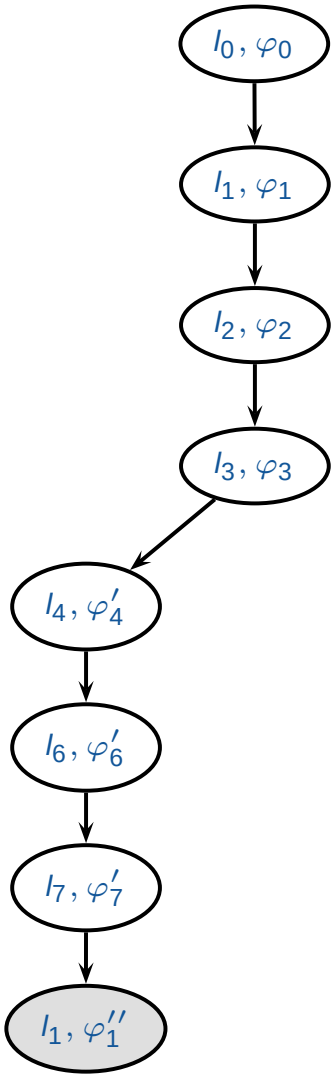
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.
     - Goto 1 to reconstruct subtree.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

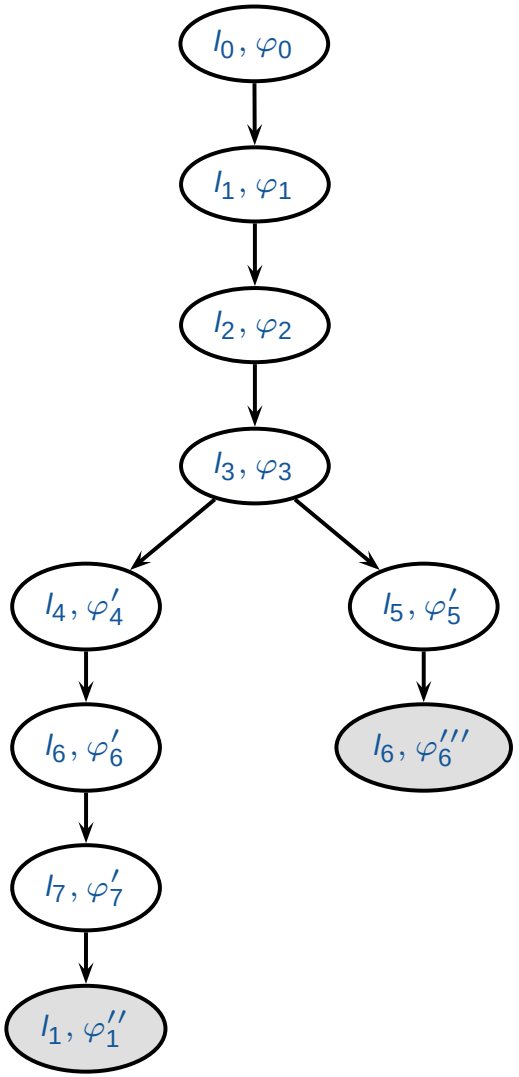
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.
     - Goto 1 to reconstruct subtree.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

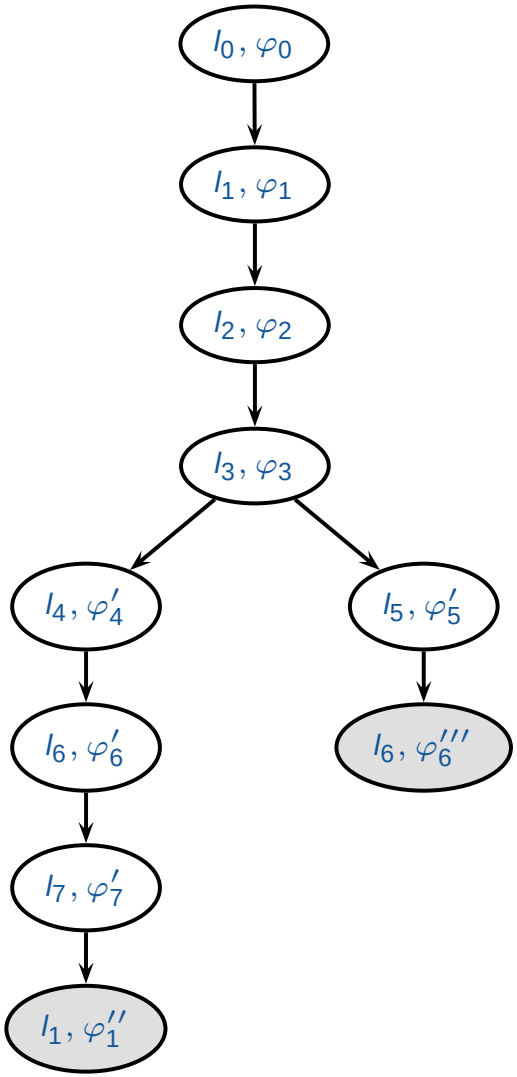
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.
     - Goto 1 to reconstruct subtree.



Lazy ART Construction

On-the-fly ART constuction with counterexample-guided abstraction refinement (CEGAR).

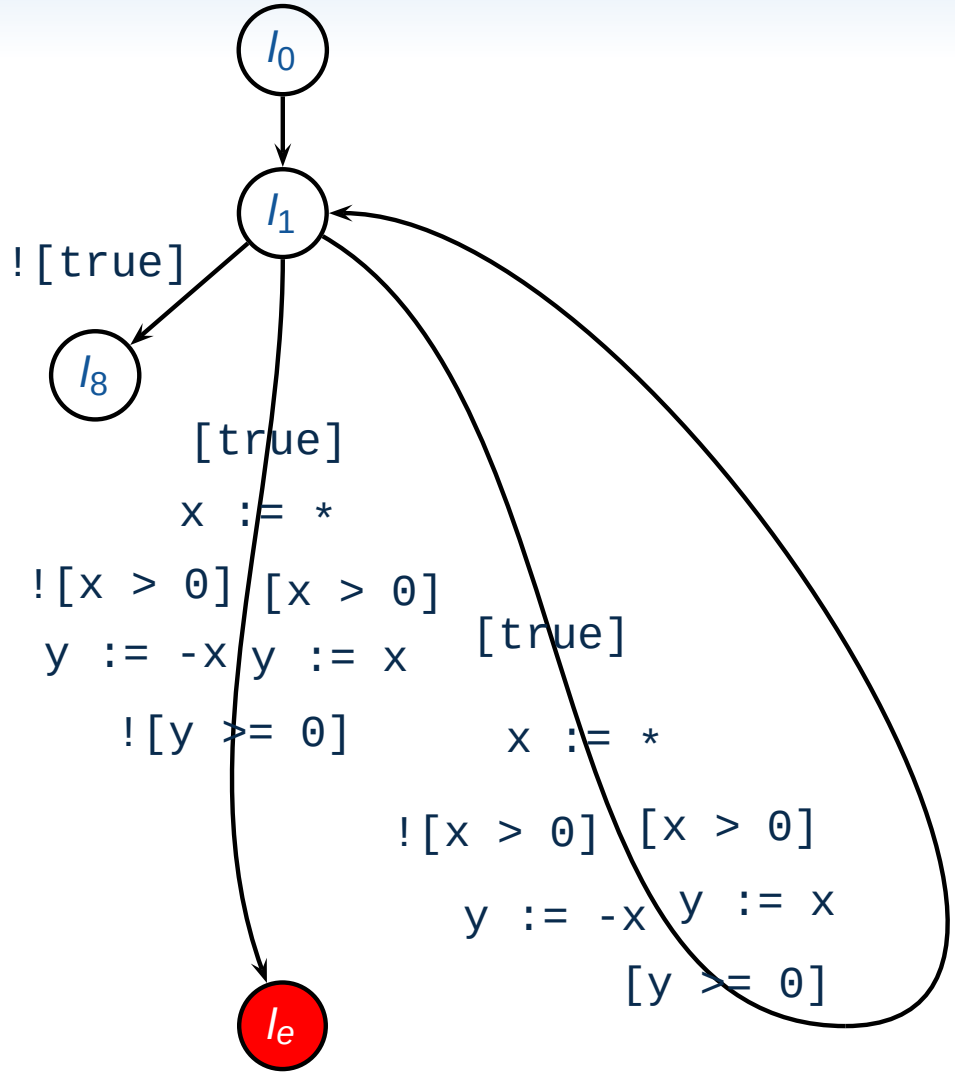
1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. If reach the error location: analyze path.
   - If path is feasible: program is unsafe.
   - If path is spurious:
     - Discover predicates to refine abstraction.
     - Undo part of ART.
     - Goto 1 to reconstruct subtree.
4. ART is safe $\Rightarrow$ program is safe.



Large Block Encoding (LBE)

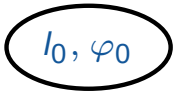
Example program:

```
while (1) {
   x = *;
   if (x >= 0) y = x;
   else y = -x;
   assert(y >= 0);
}
```



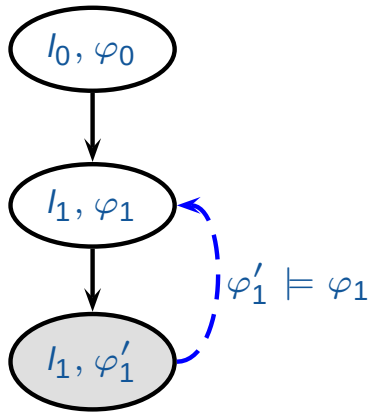
Lazy ART Construction with LBE

1. Pick an ART node.



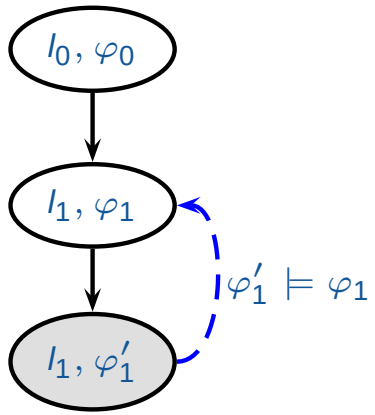
Lazy ART Construction with LBE

1. Pick an ART node.
2. Compute abstract successors, until node gets covered.



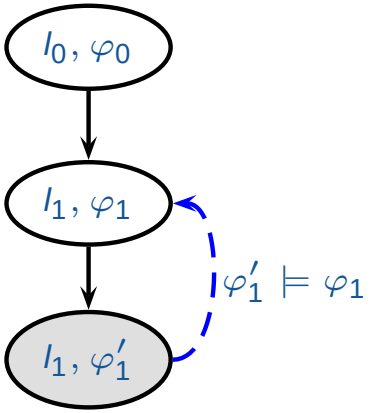
Lazy ART Construction with LBE

1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. Error location is not reachable by abstract strongest post operator.



Lazy ART Construction with LBE

1. Pick an ART node.
2. Compute abstract successors, until node gets covered.
3. Error location is not reachable by abstract strongest post operator.
4. ART is safe $\Rightarrow$ program is safe.



Beyond the Sequential Case

- Architecture in many application domains
  - one scheduler runs mutually exclusive, cooperative threads
  - SystemC, PLC, AADL, railways control software, . . .
- Key idea: do not analyze scheduler + threads;
  instead, run scheduler while analyzing threads
  - ESST: explicit scheduler + symbolic threads [CMNR10]
  - Partial order reduction within the explicit scheduler [CNR11]
- ESST implemented in Kratos
  `http://es.fbk.eu/tools/kratos` (see [CGM+11])

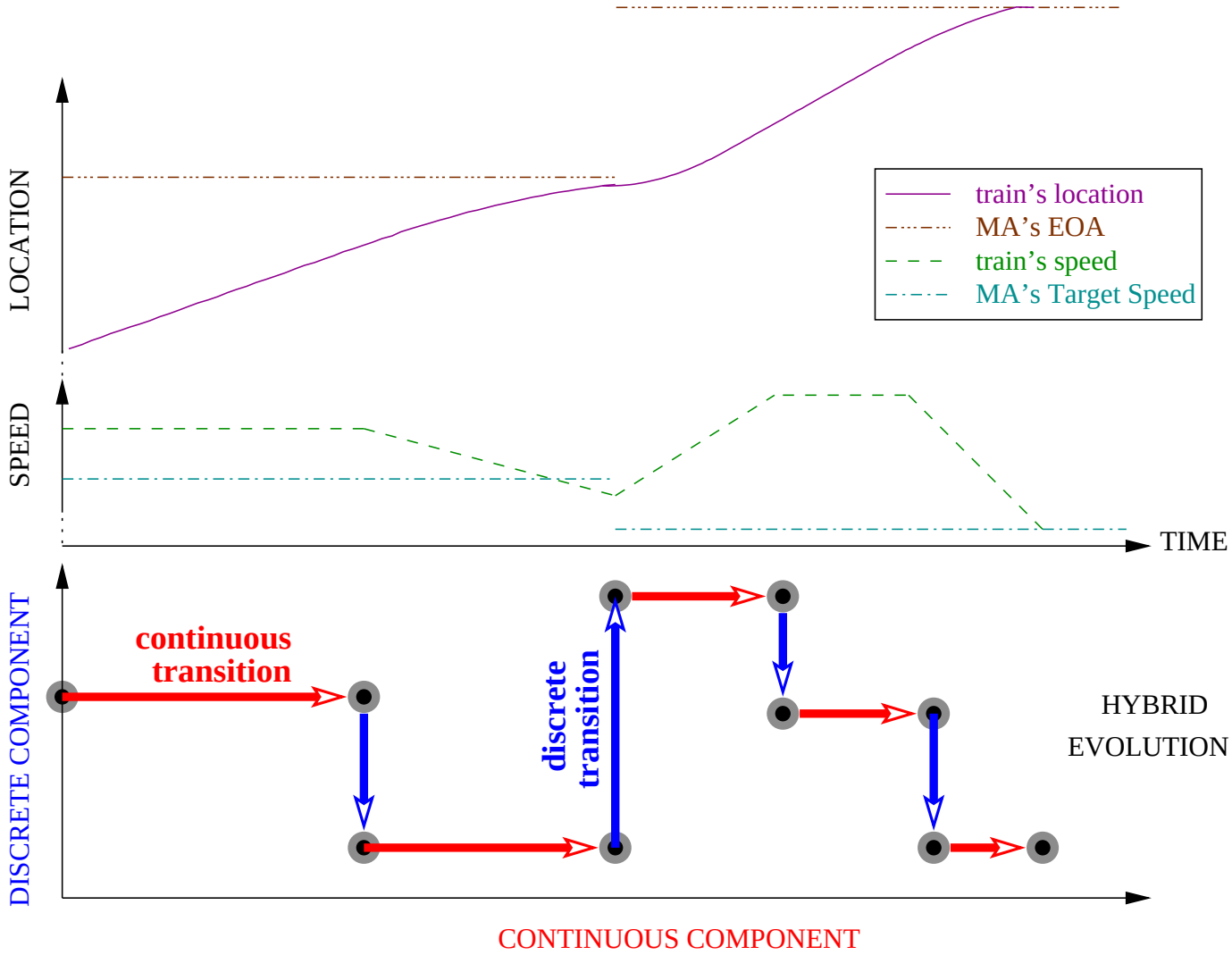
Outline

Complex embedded systems

- Embedded software is software which is part of a larger system whose purpose may be not computational.
- Example: 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.

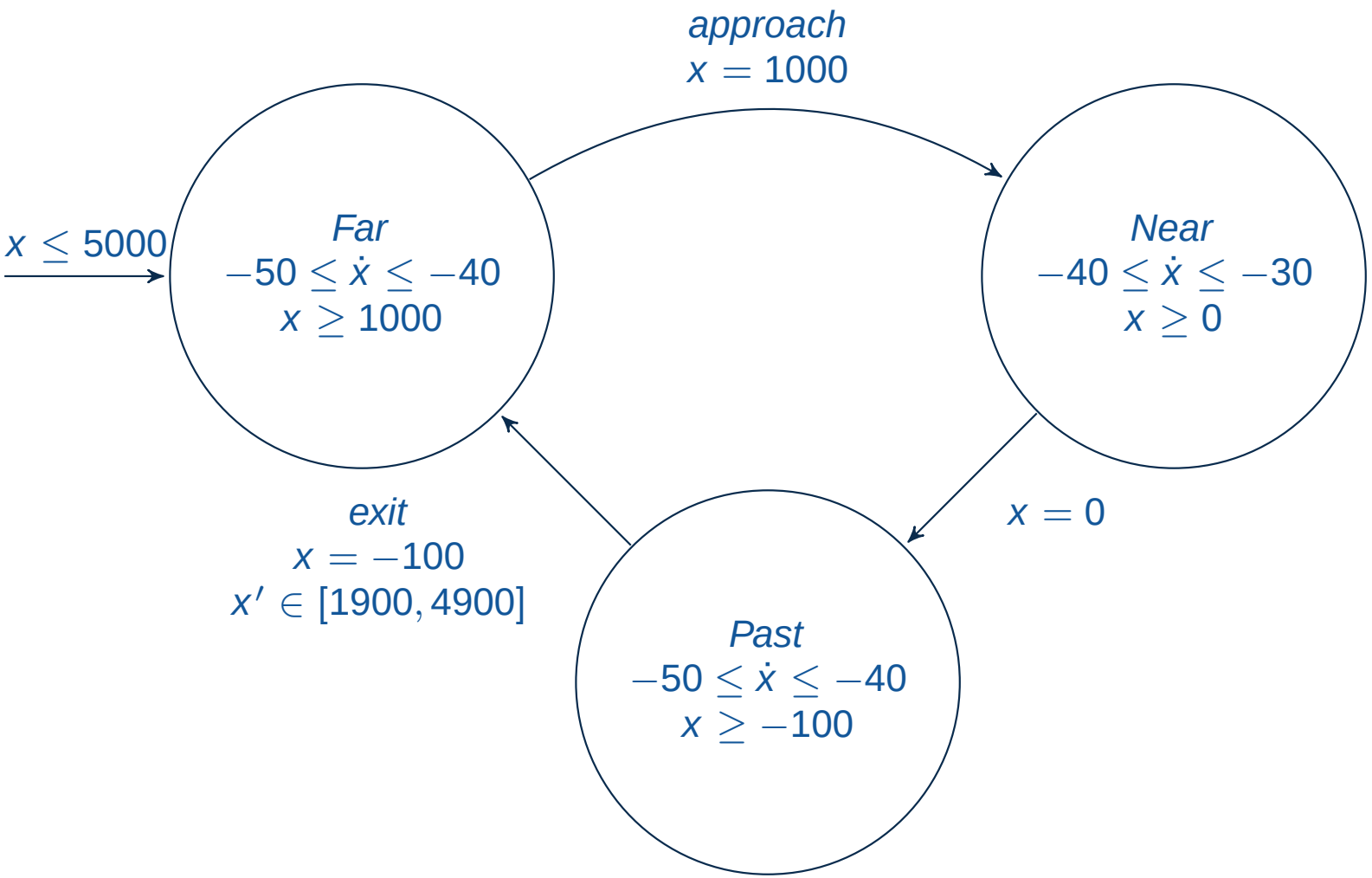
} Hybrid Systems

Continuous vs. discrete changes



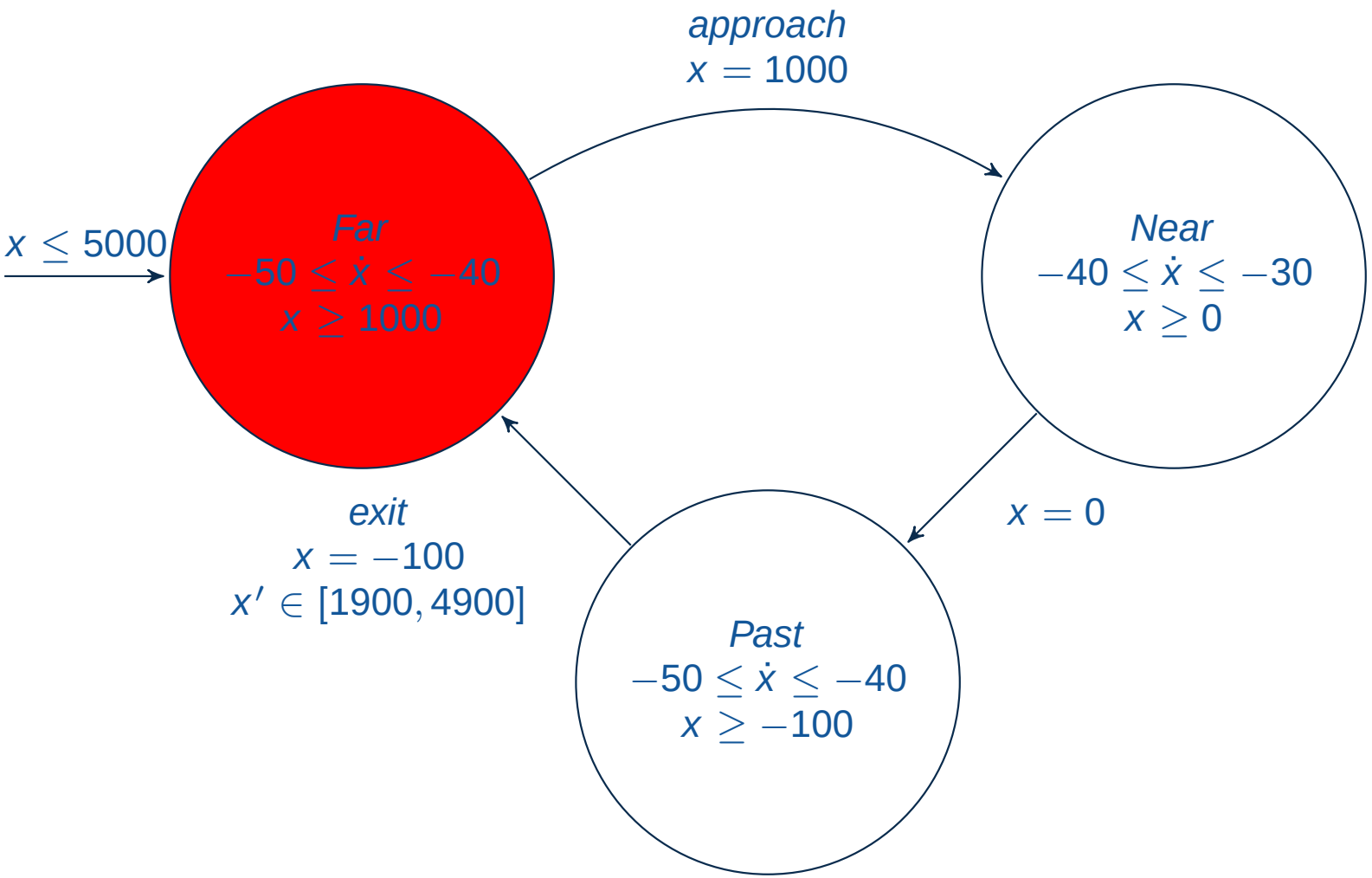
Hybrid automata

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.



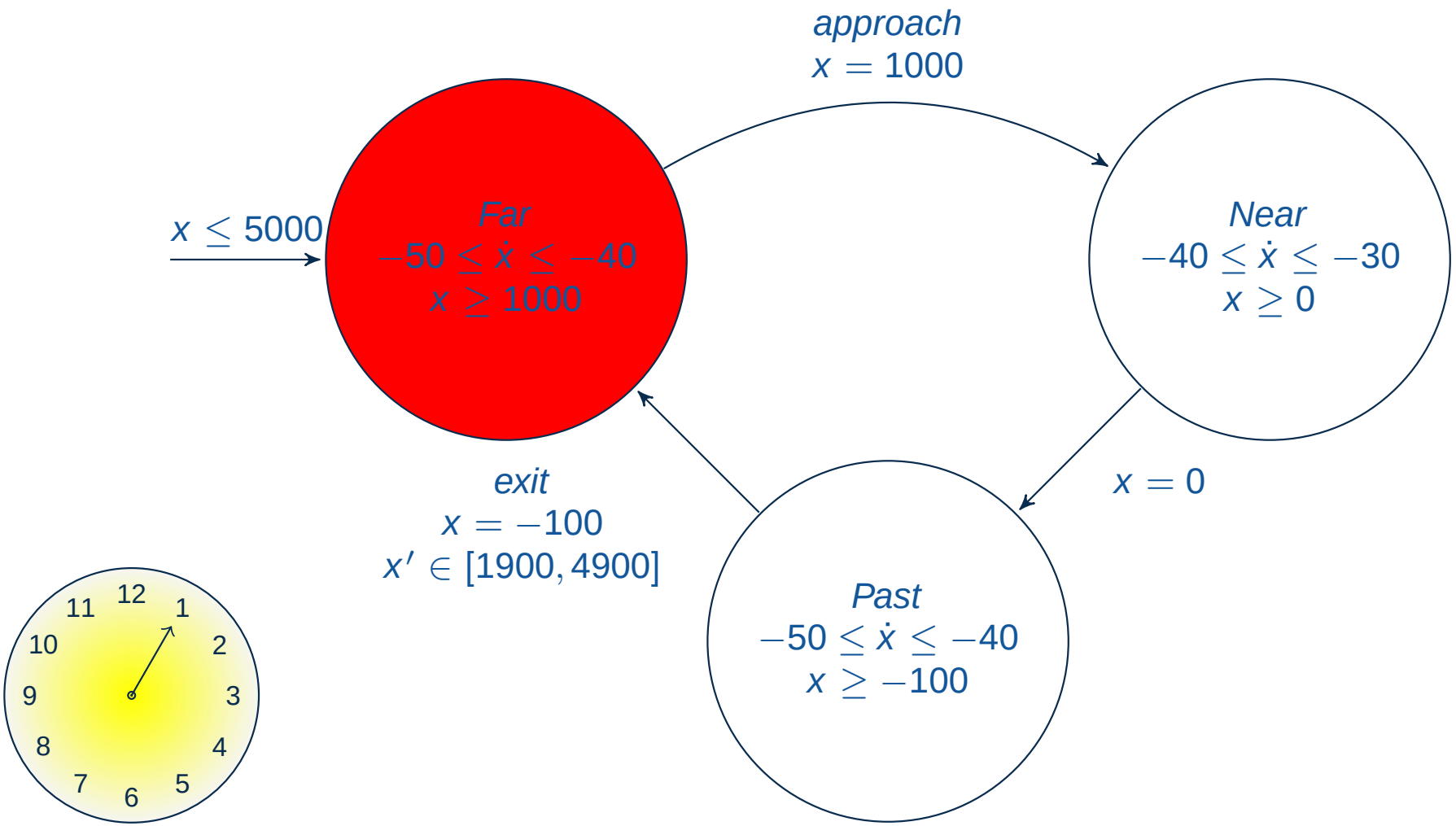
Hybrid automata

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.



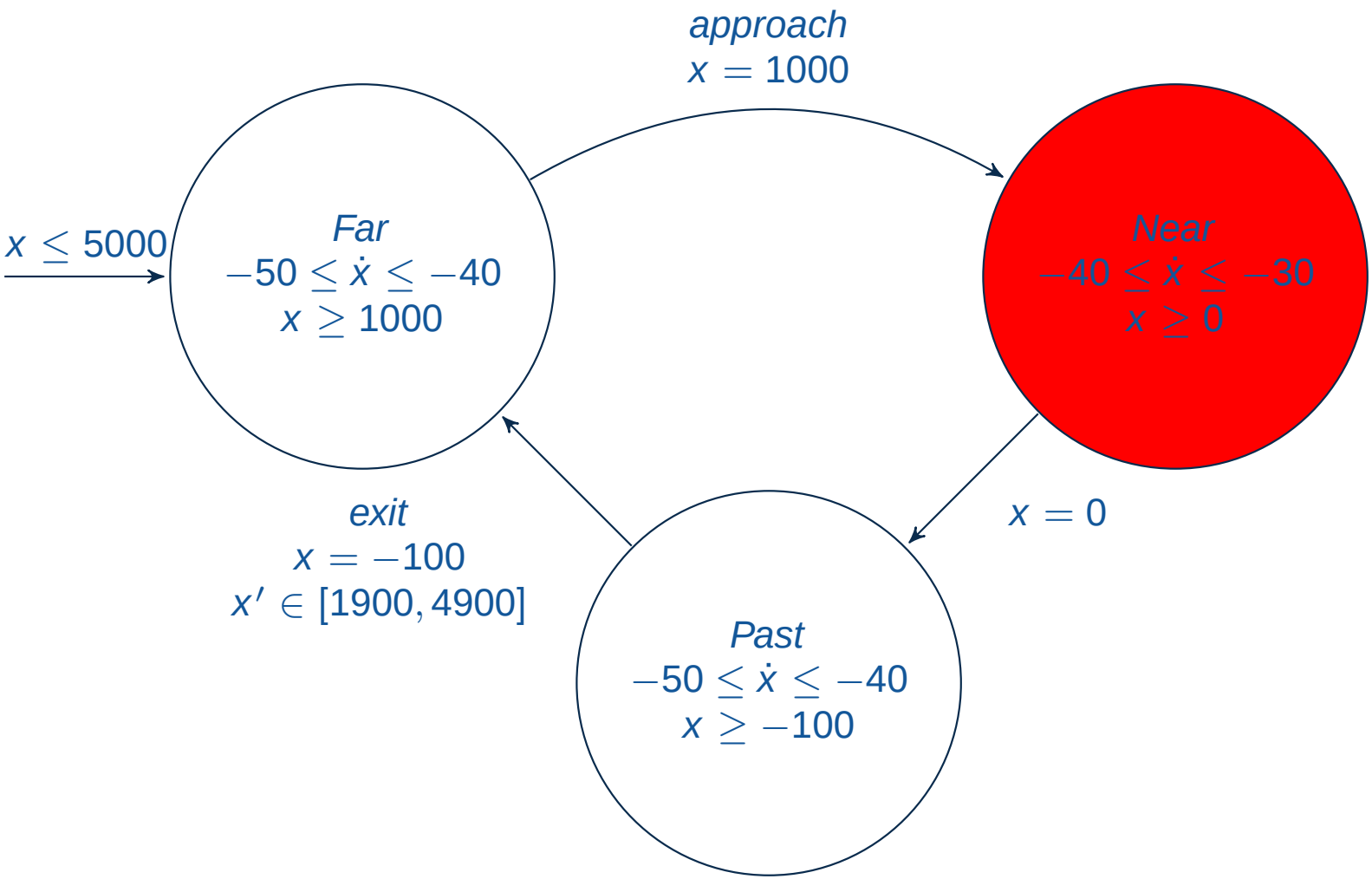
Hybrid automata

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.



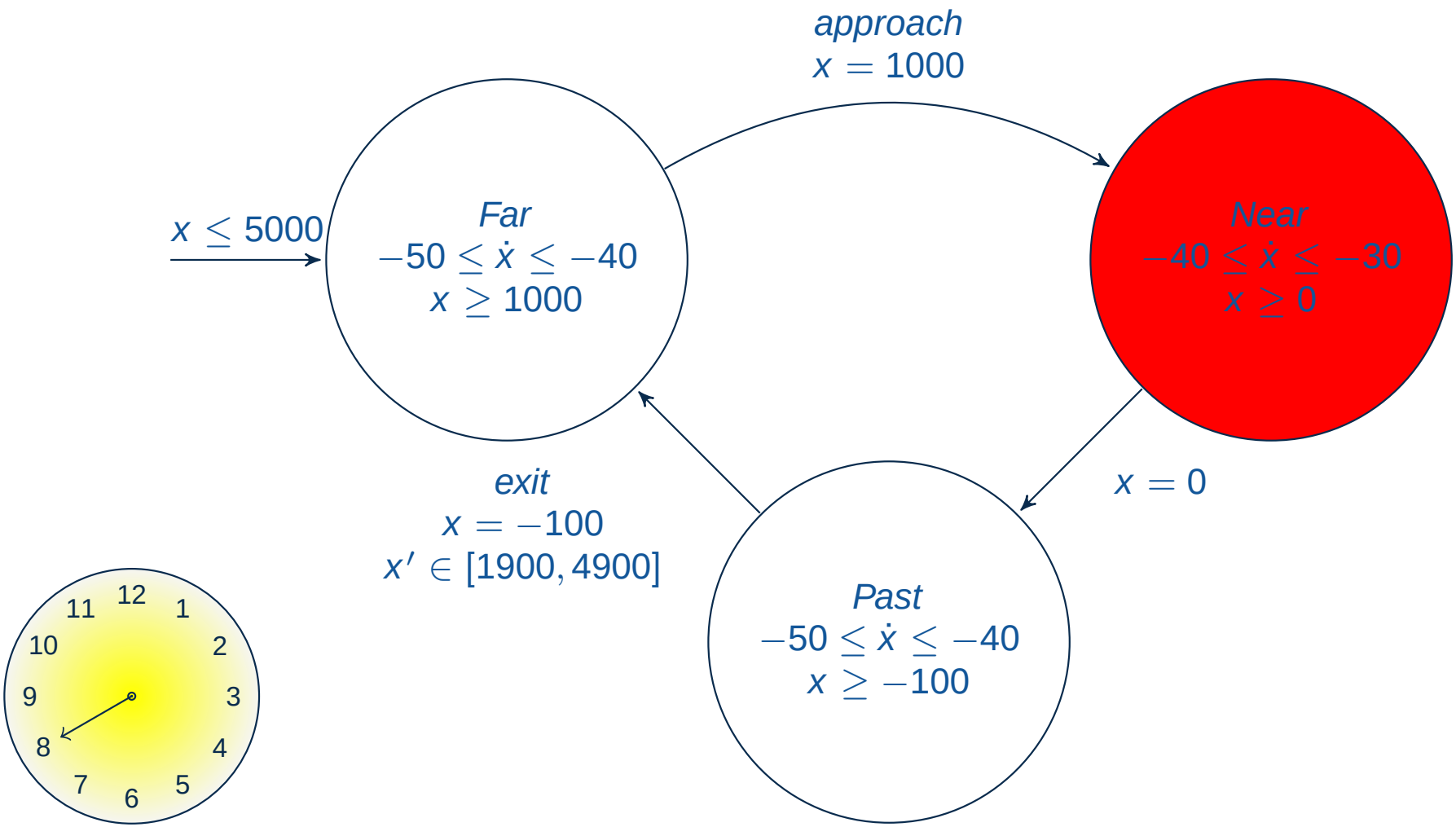
Hybrid automata

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.



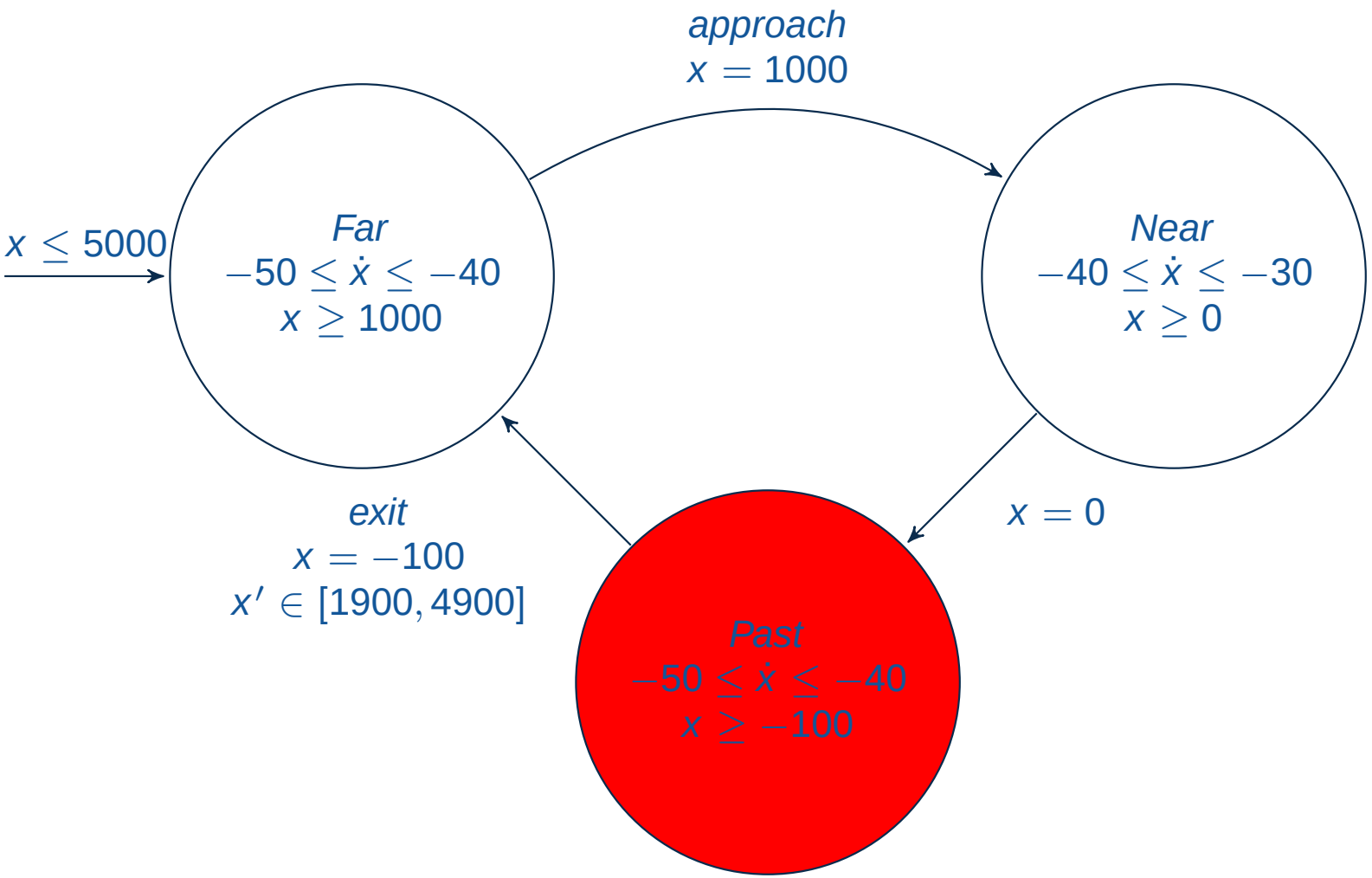
Hybrid automata

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.



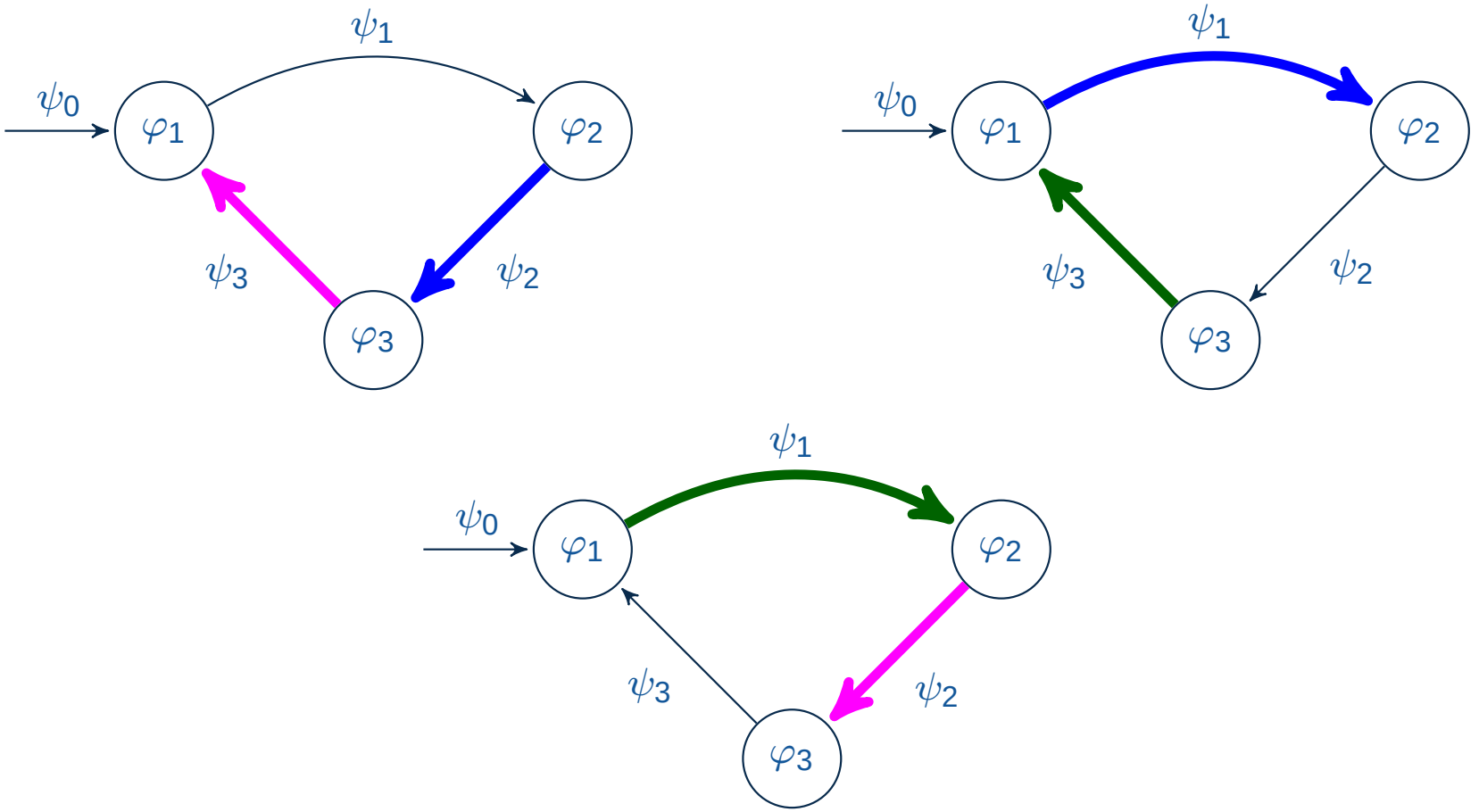
Hybrid automata

- Hybrid automata are a widely accepted modeling framework for systems with discrete and continuous variables.



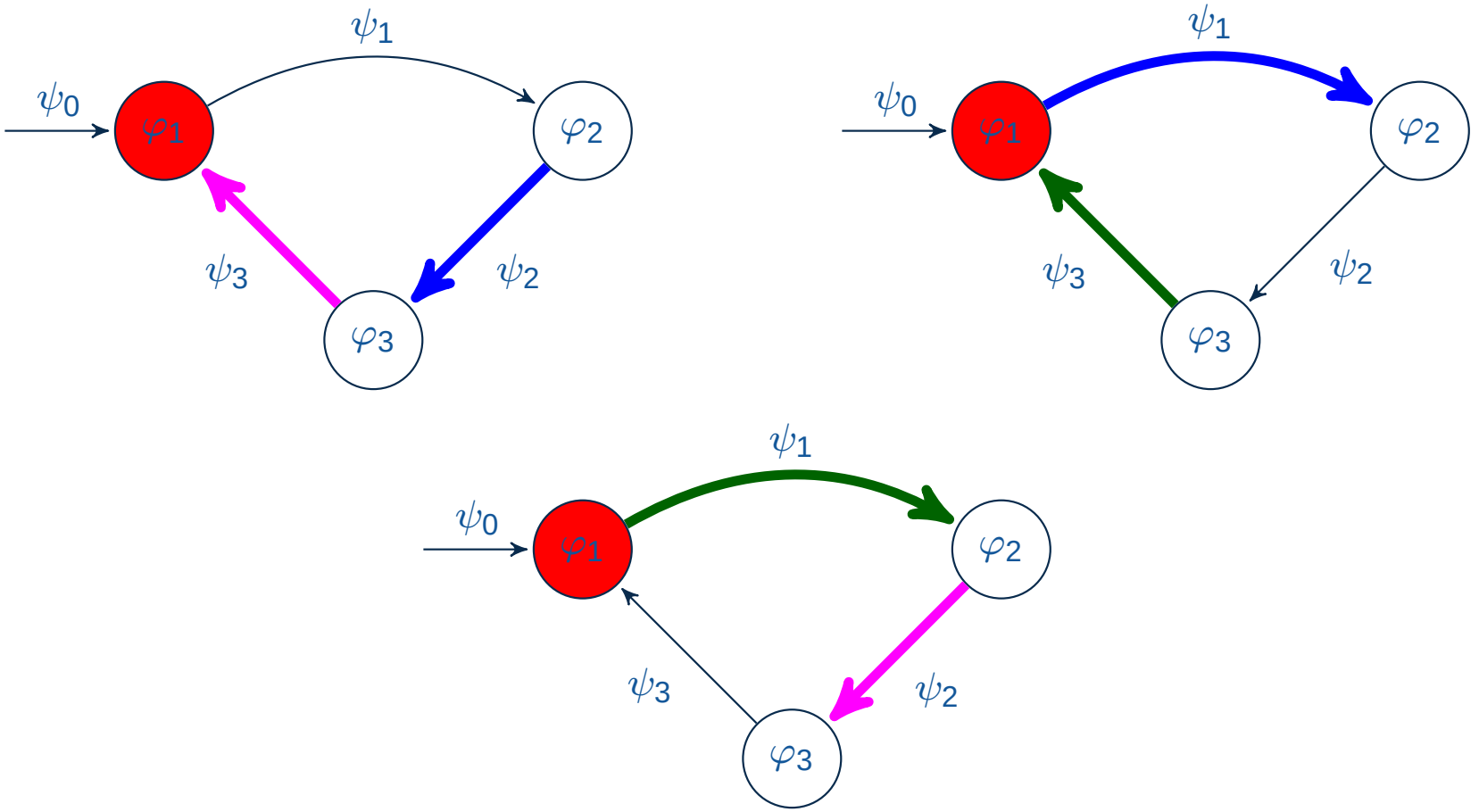
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



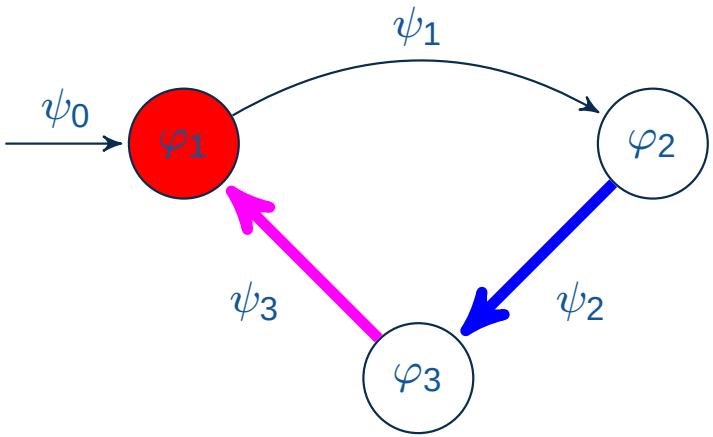
Network of Hybrid automata

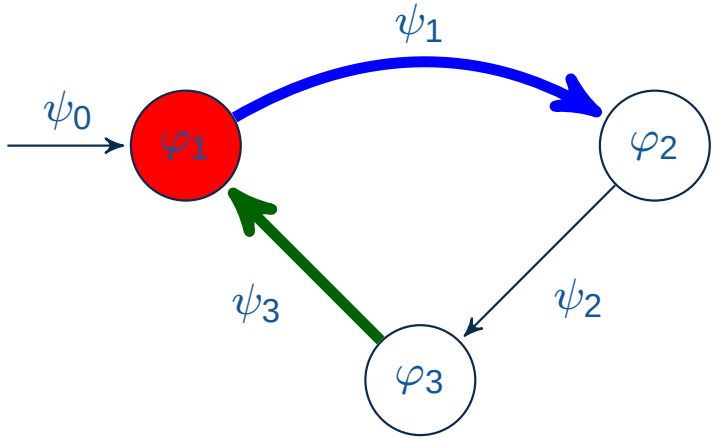
- Networks are partially synchronous composition of hybrid automata.

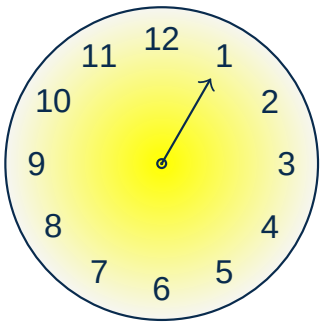


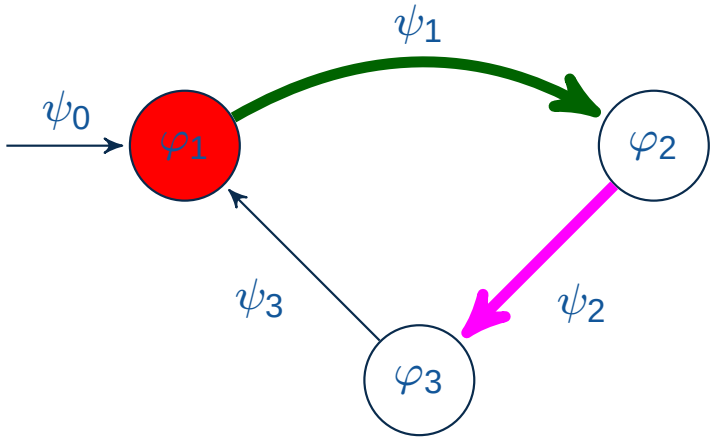
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



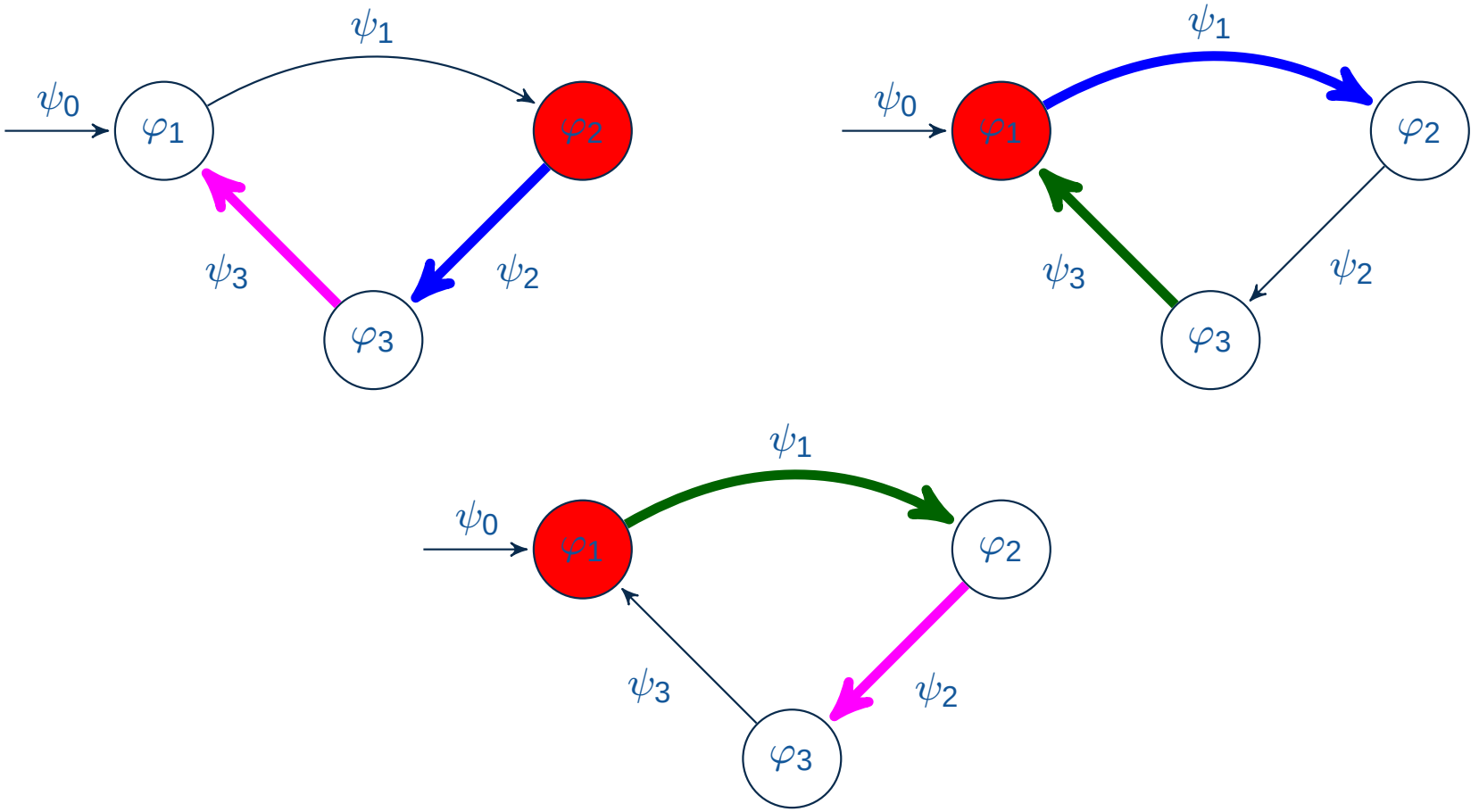






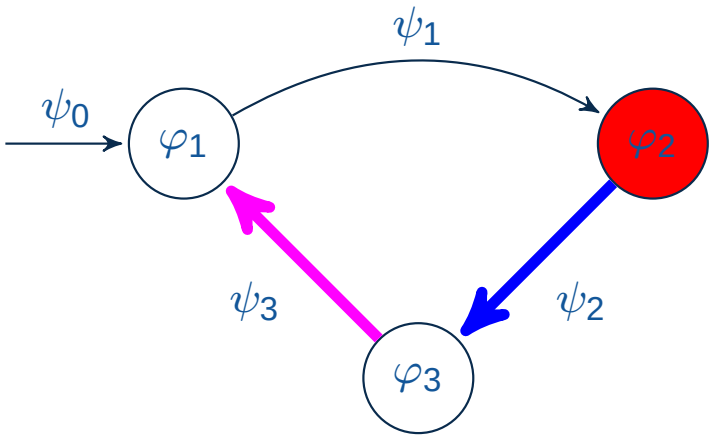
Network of Hybrid automata

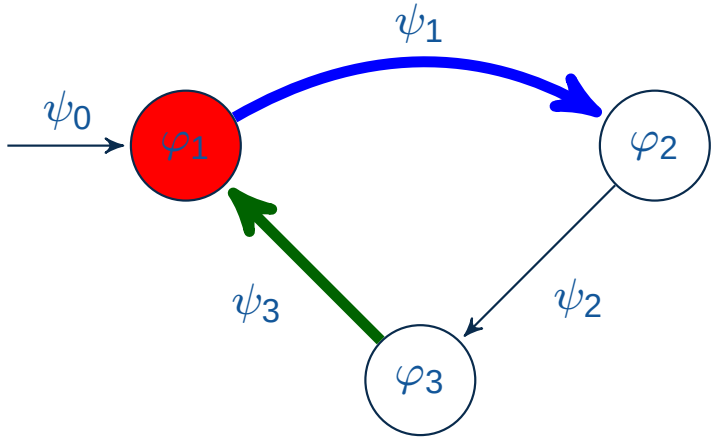
- Networks are partially synchronous composition of hybrid automata.

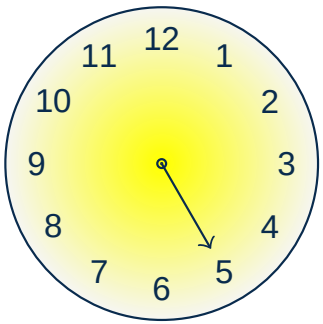


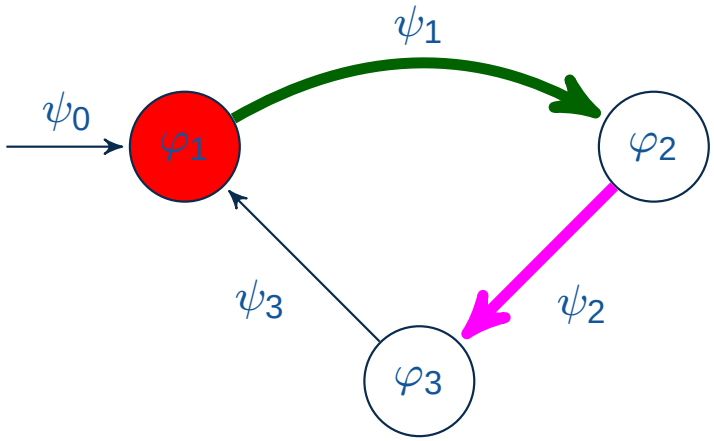
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



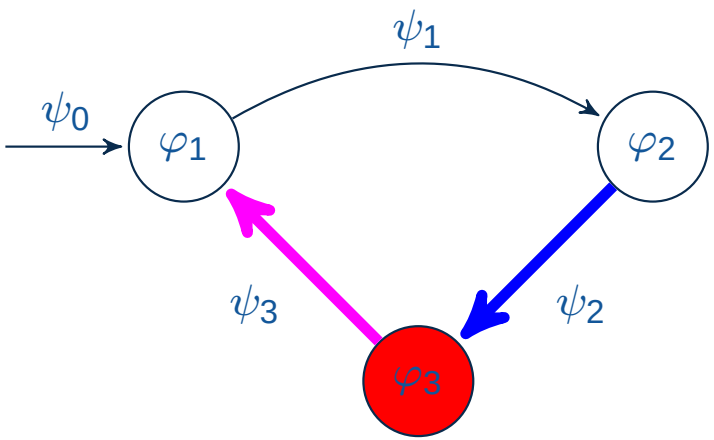


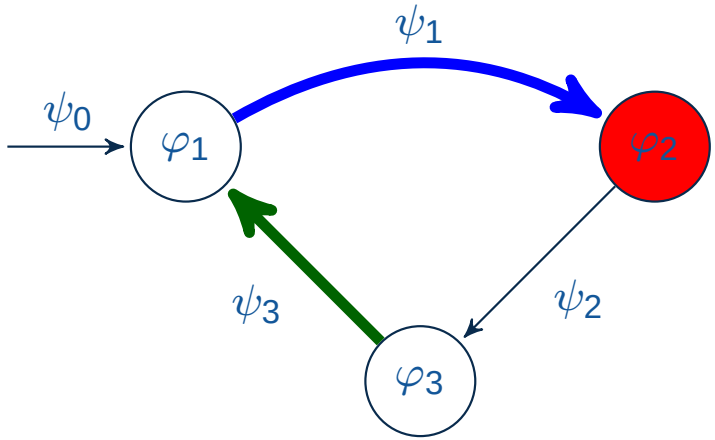


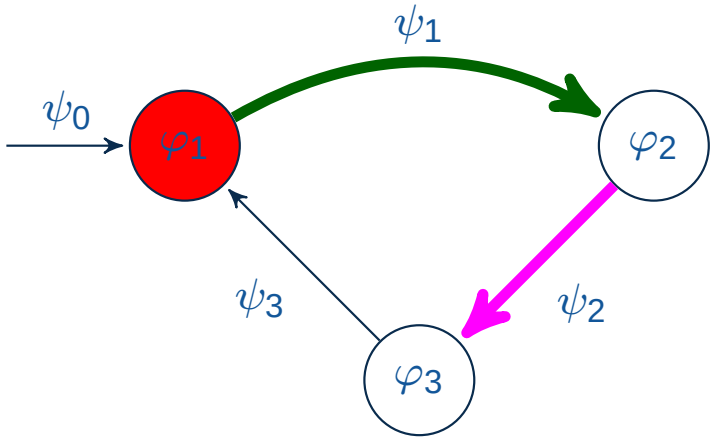


Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.

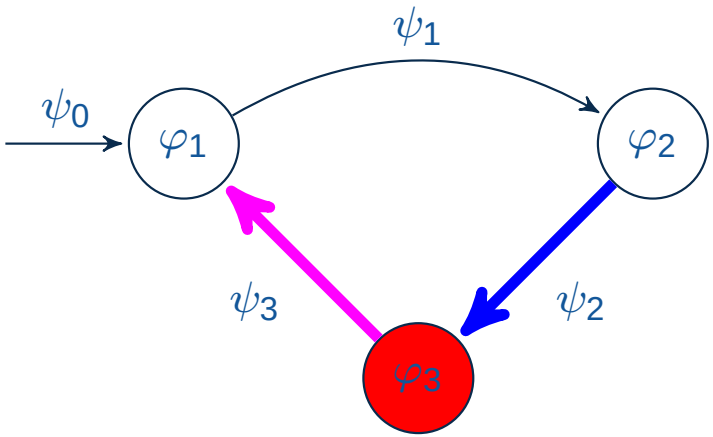


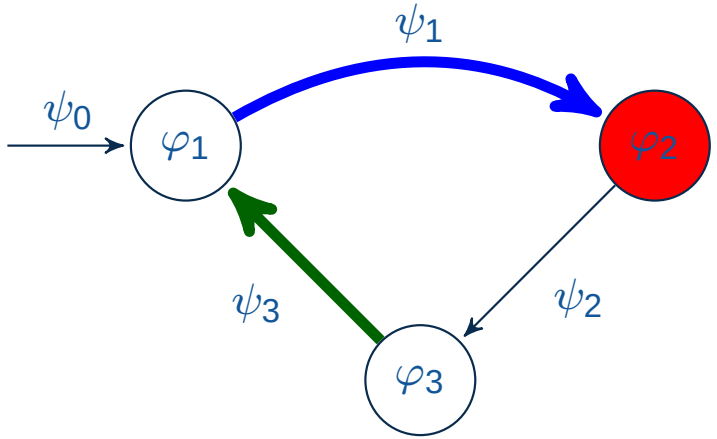


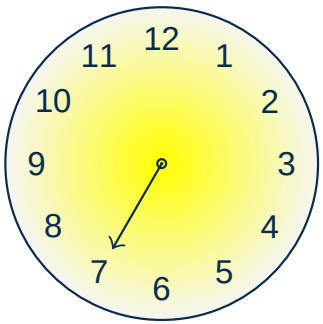


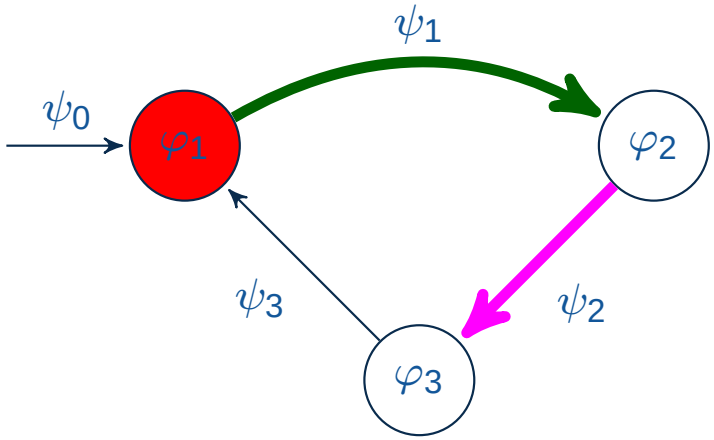
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



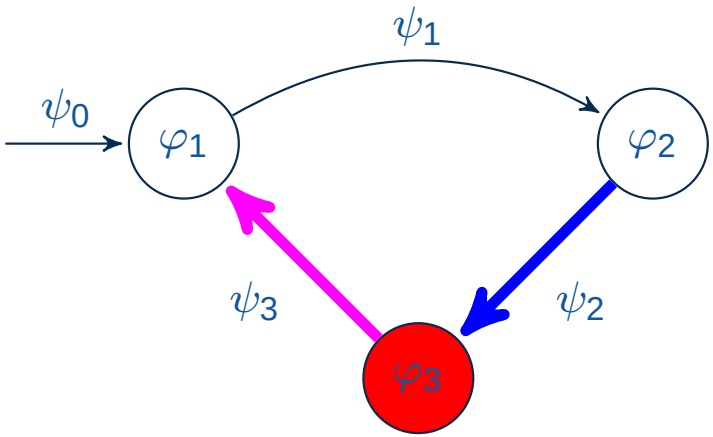


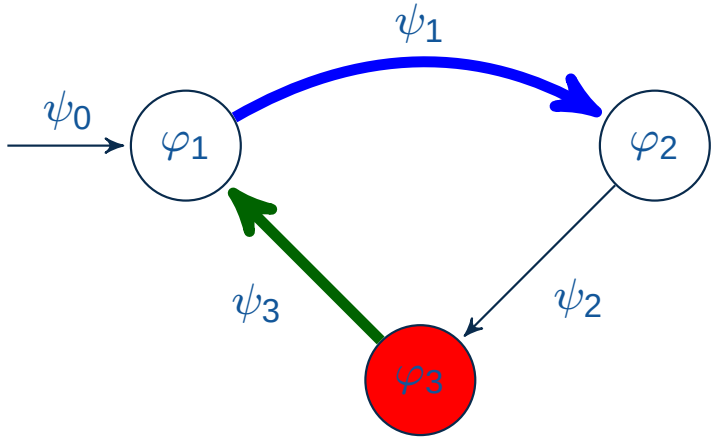


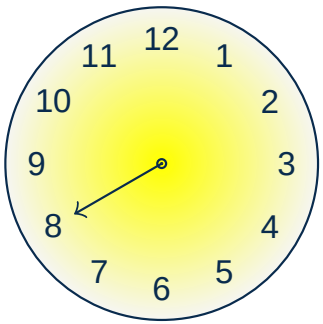


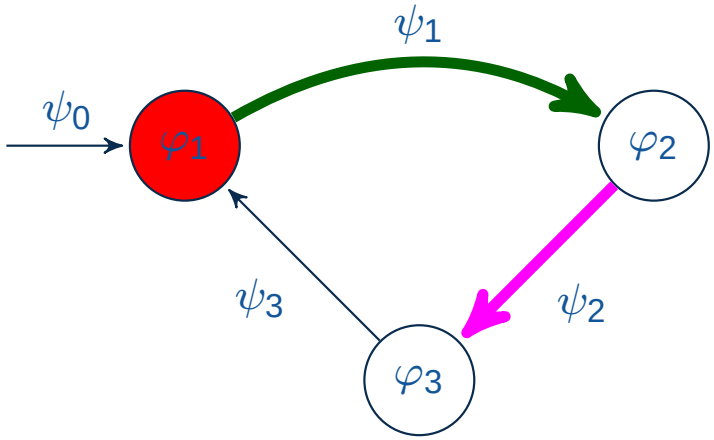
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



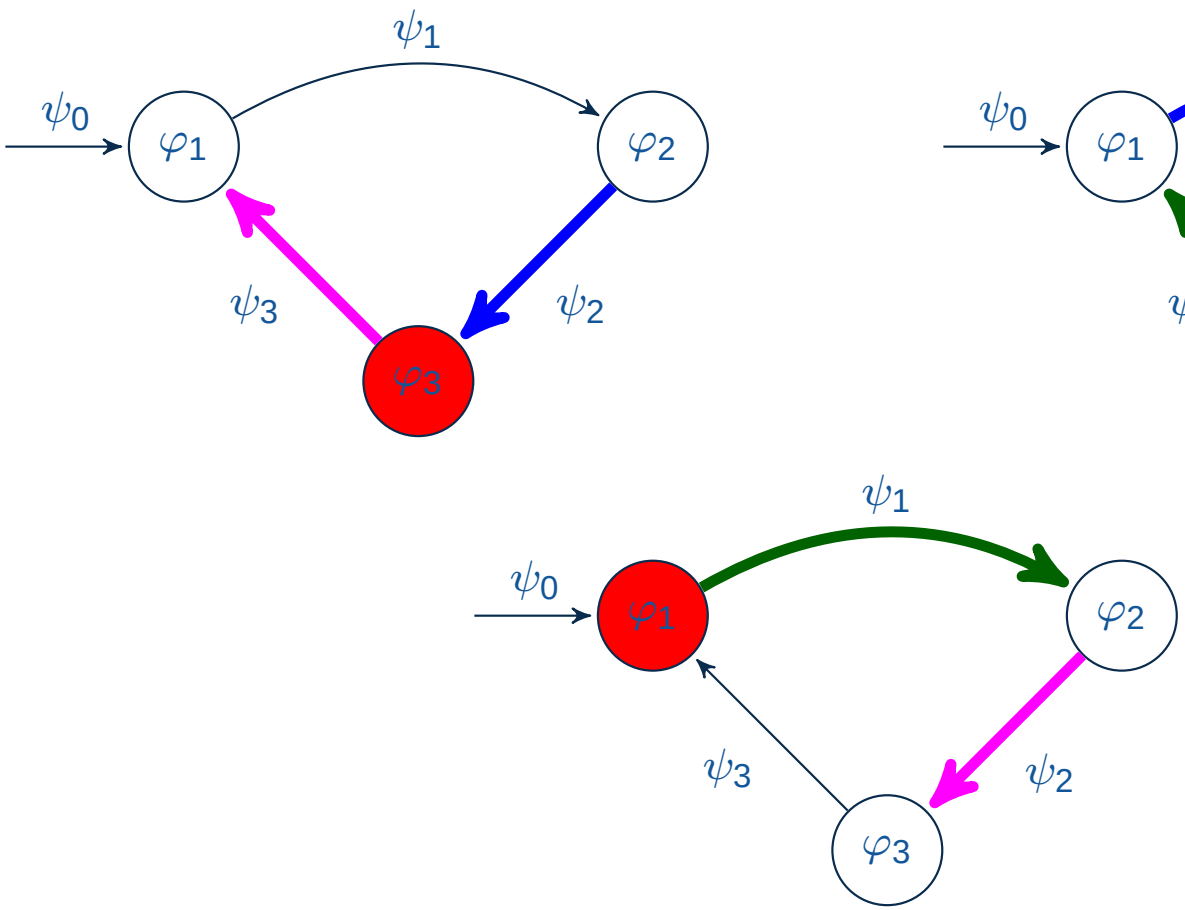






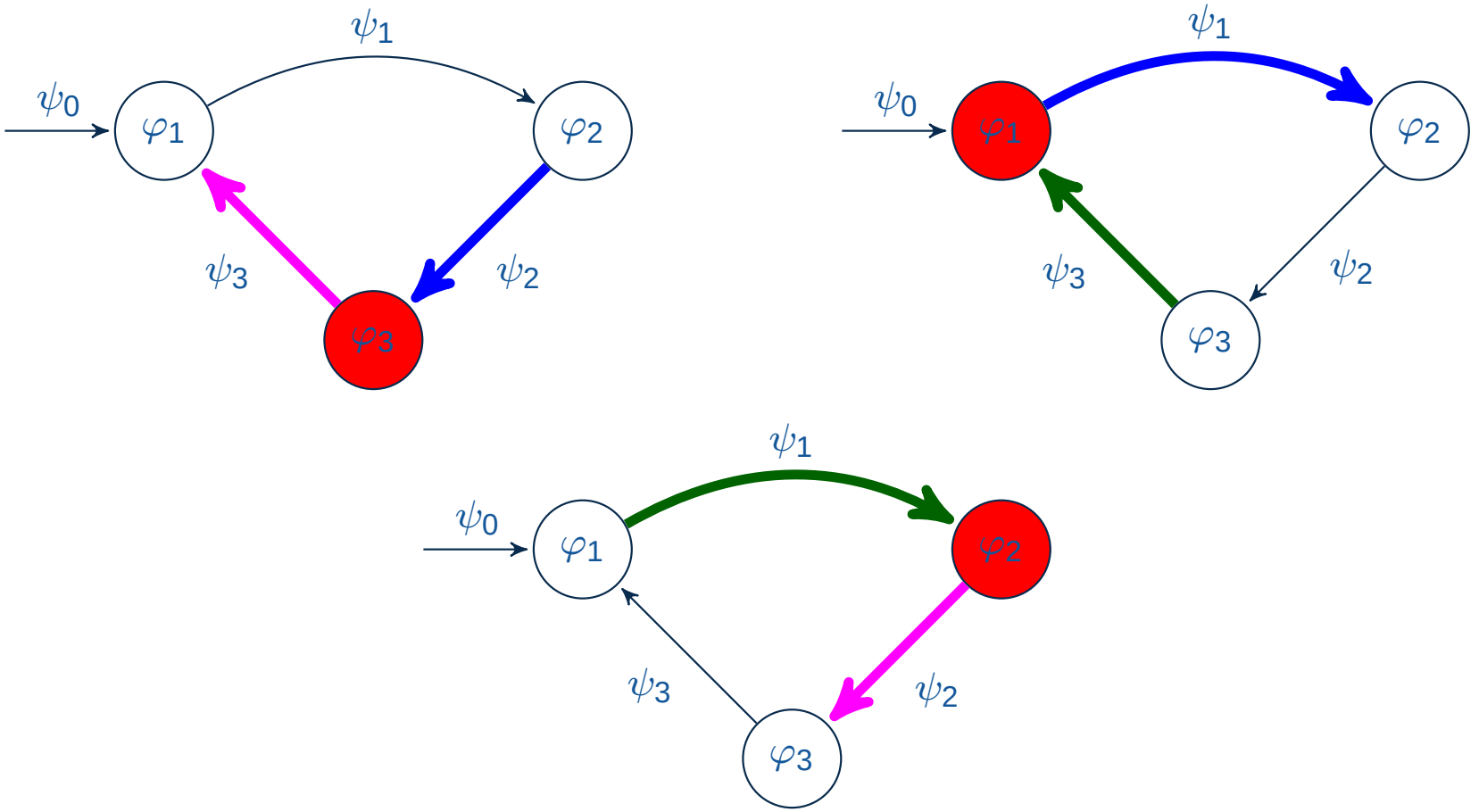
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



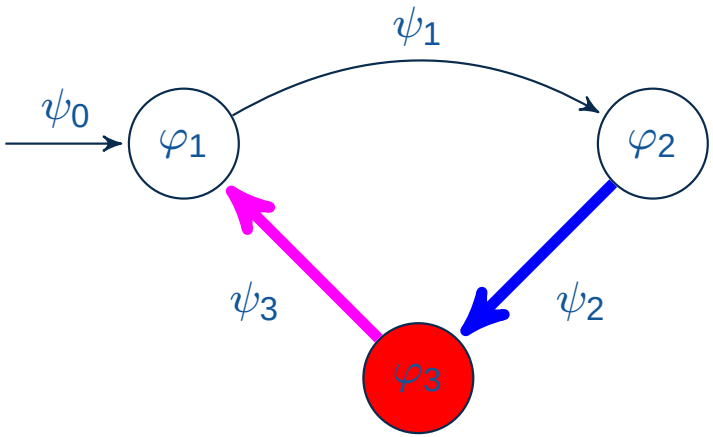
Network of Hybrid automata

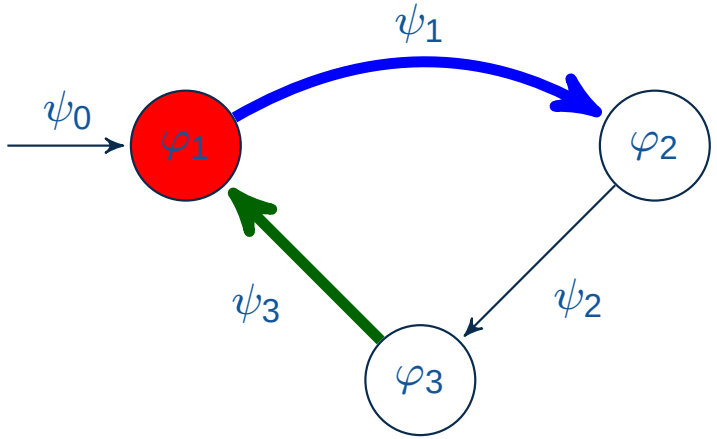
- Networks are partially synchronous composition of hybrid automata.

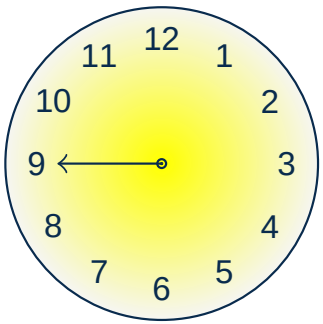


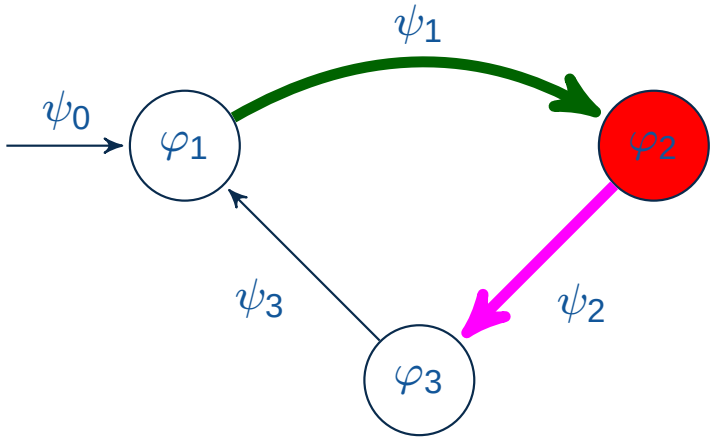
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



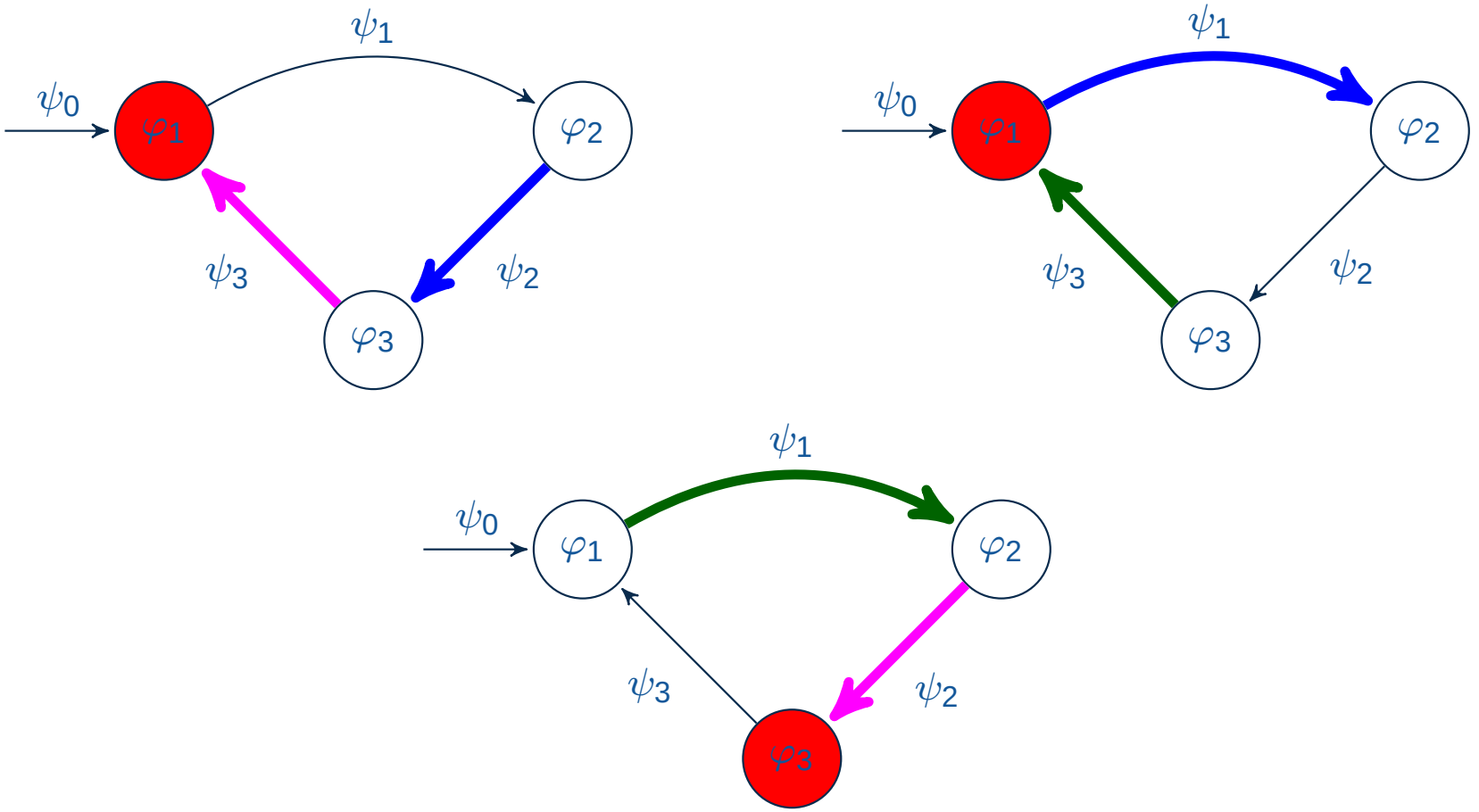






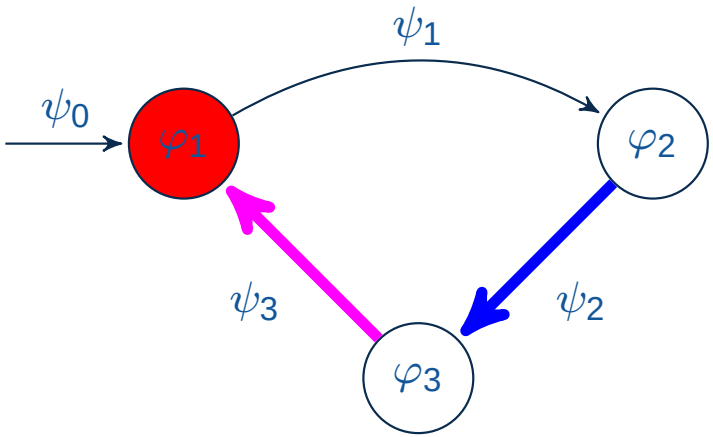
Network of Hybrid automata

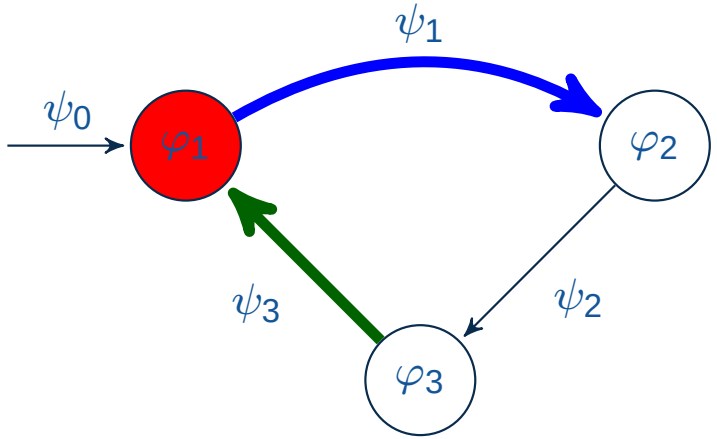
- Networks are partially synchronous composition of hybrid automata.

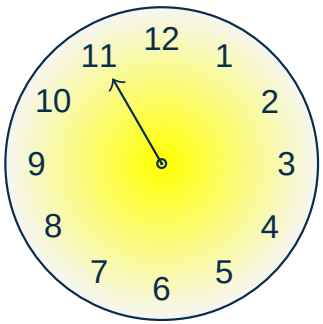


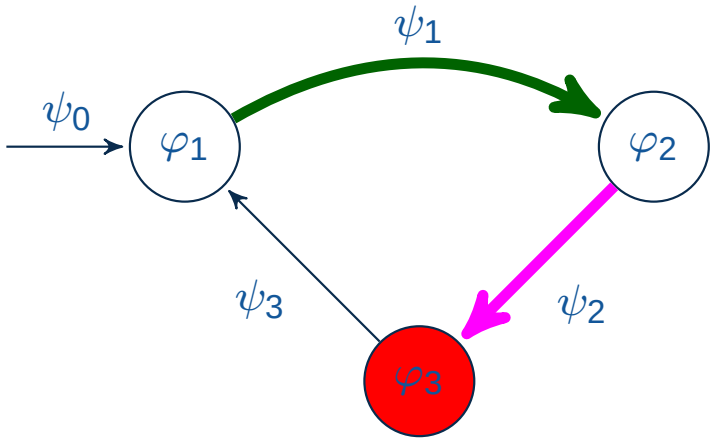
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



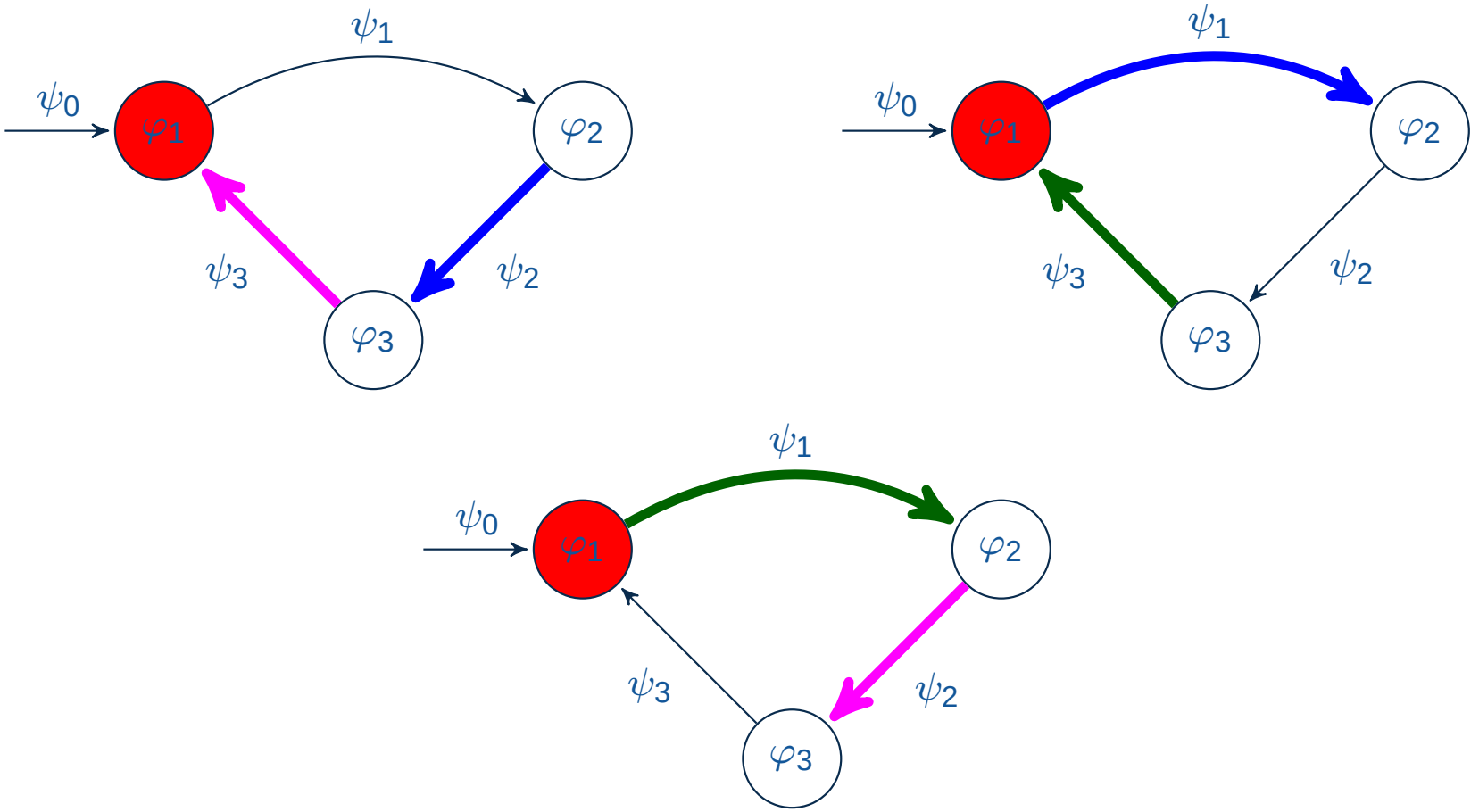






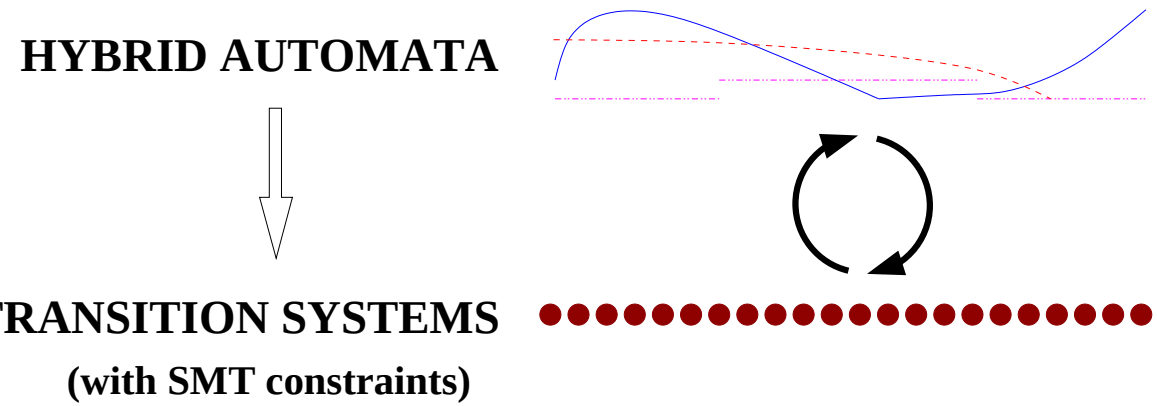
Network of Hybrid automata

- Networks are partially synchronous composition of hybrid automata.



Symbolic bounded reachability (BMC)

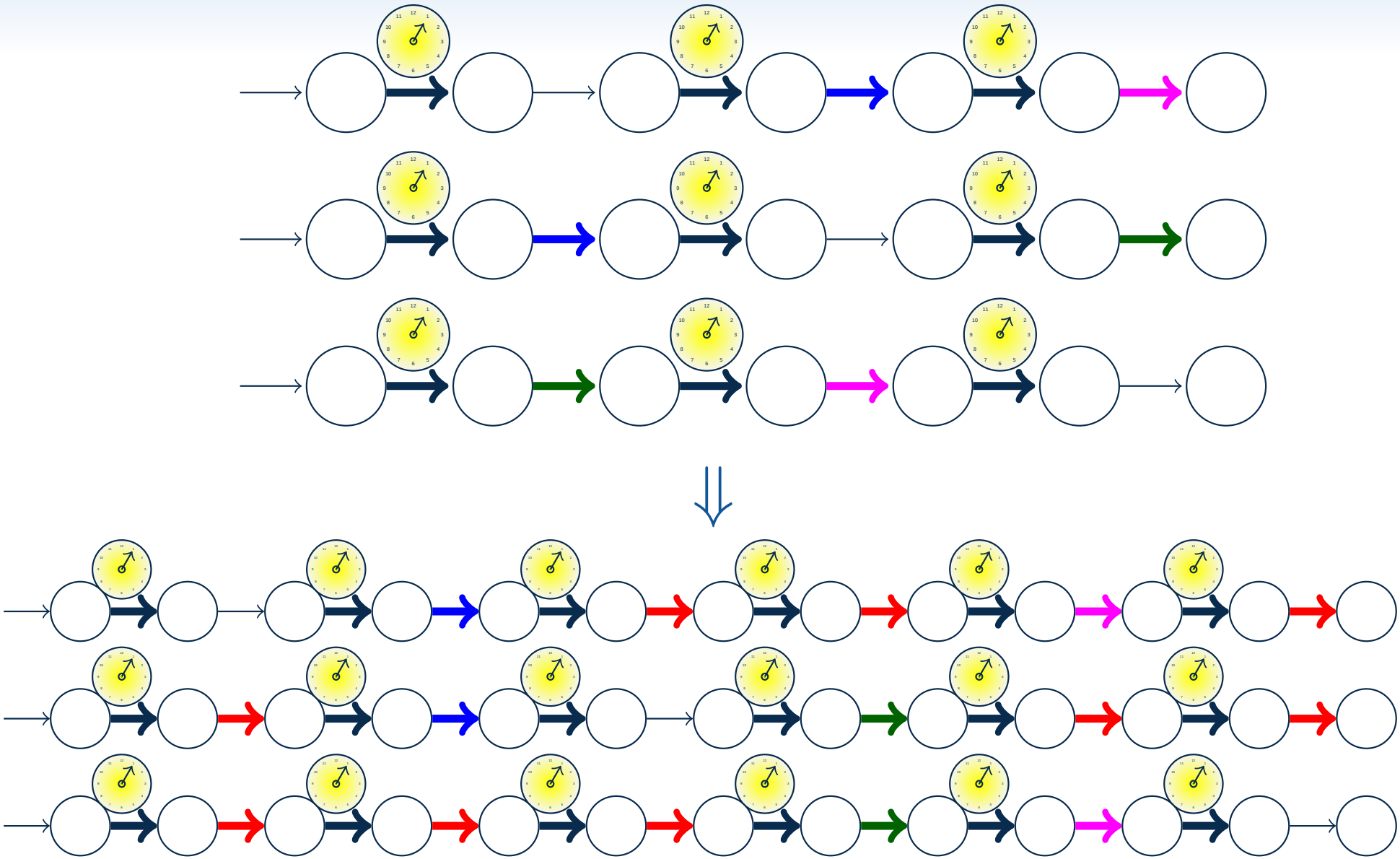
- Look for path up to *k* steps.
- Encode the bounded reachability problem into a formula:
  - the formula is satisfiable iff the target is reachable.
- Encoding unrolls the transition relation *k* times.
- *k* is critical for performance:
  - number of possible paths is exponential in *k*!
- Necessary *k* depends on the semantics of the composition of systems.
- Baseline: compilation into symbolic transition system.



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



Symbolic encoding of a network

- Components represented with symbolic transition systems.
- Local input variable ε enumerating the events including:
  - Shared timed event T.
  - Local stutter event S.
- Shared (global) input variable δ to represent the elapsed time.
- BMC encoding of the network obtained by conjoining the BMC encodings of the components:
  $\textsc{BmcInt}_{\mathcal{N}}(k) := \bigwedge_{1 \leq j \leq n} \textsc{Bmc}_{H_j}(k) \wedge \textsc{Sync}_{\mathcal{N}}(k)$
- $\textsc{Sync}$ encodes the synchronization of the local runs:
  - Strict synchronization:

    $\textsc{StrictSync}^k_{\mathcal{N}} := \bigwedge_{1 \leq j < h \leq n} \bigwedge_{0 \leq i < k} \bigwedge_{a \in U_j \cap U_h \cup \{\mathsf{T}\}} (l^i_j = a \leftrightarrow l^i_h = a)$

    $\wedge \bigwedge_{a \in U_j \setminus U_h} (l^i_j = a \rightarrow l^i_h = \mathsf{S}) \wedge \bigwedge_{a \in U_h \setminus U_j} (l^i_h = a \rightarrow l^i_j = \mathsf{S})$
  - Step semantics (exploiting independence of local transitions):

Symbolic transition system (Global time)

$$\textsc{Init} := \bigwedge_{q \in Q} (loc = q \rightarrow I_q(X))$$

$$\textsc{Invar} := \bigwedge_{q \in Q} (loc = q \rightarrow Z_q(X))$$

$$\textsc{Trans} := \bigwedge_{q \in Q} (loc = q \rightarrow (\textsc{Stutter} \vee \textsc{Timed}_q \vee \bigvee_{(q,p) \in E} \textsc{Untimed}_{q,p}))$$

$$\textsc{Stutter} := \varepsilon = \mathsf{s} \wedge \delta = 0 \wedge loc' = loc \wedge X' = X$$

$$\textsc{Timed}_q := \varepsilon = \mathsf{t} \wedge \delta > 0 \wedge loc' = loc \wedge F_q(\frac{X' - X}{\delta})$$

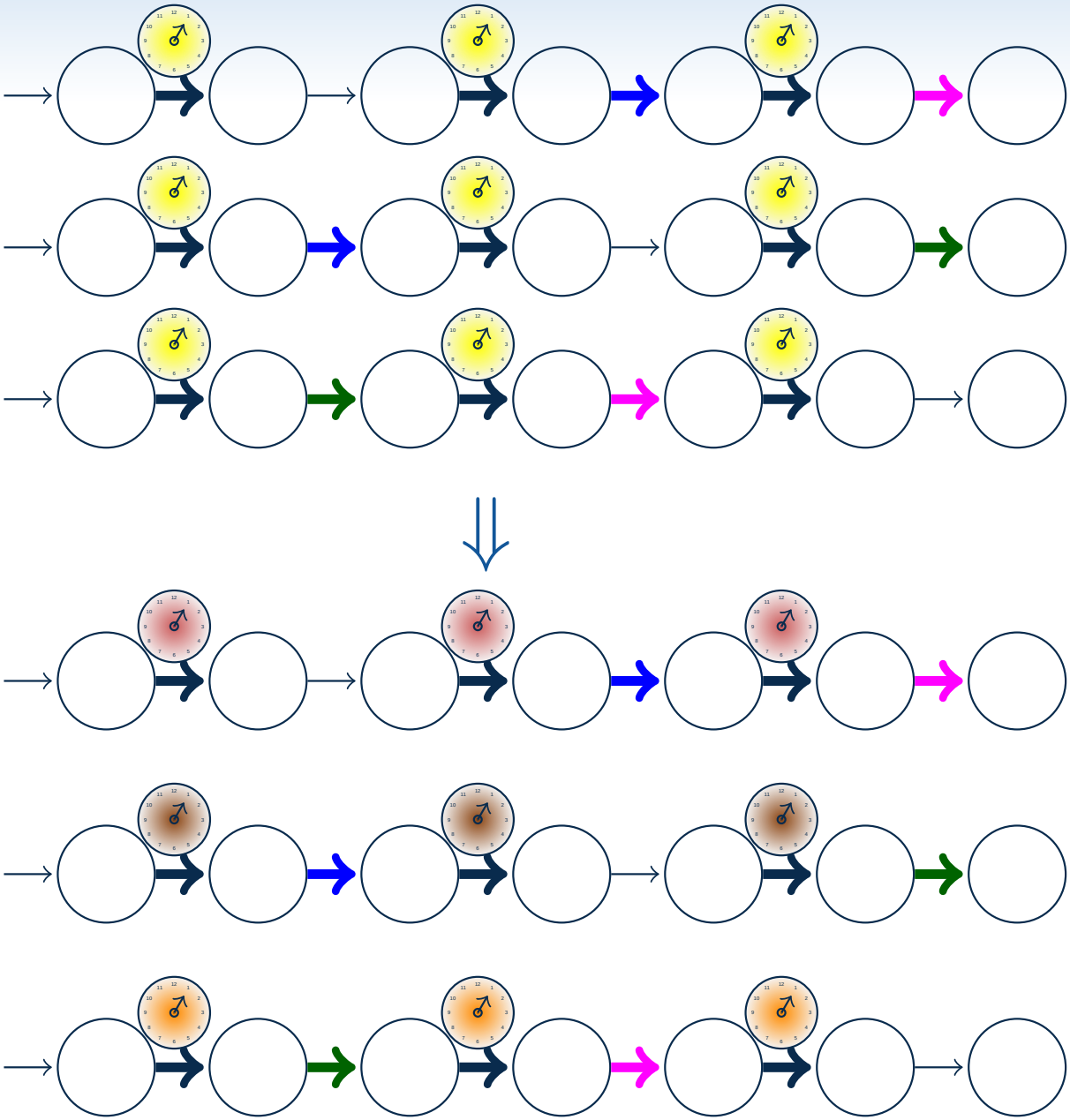
$$\textsc{Untimed}_{q,p} := \varepsilon = L_{q,p} \wedge \delta = 0 \wedge loc' = p \wedge J_{q,p}(X, X')$$

δ is a global shared variable

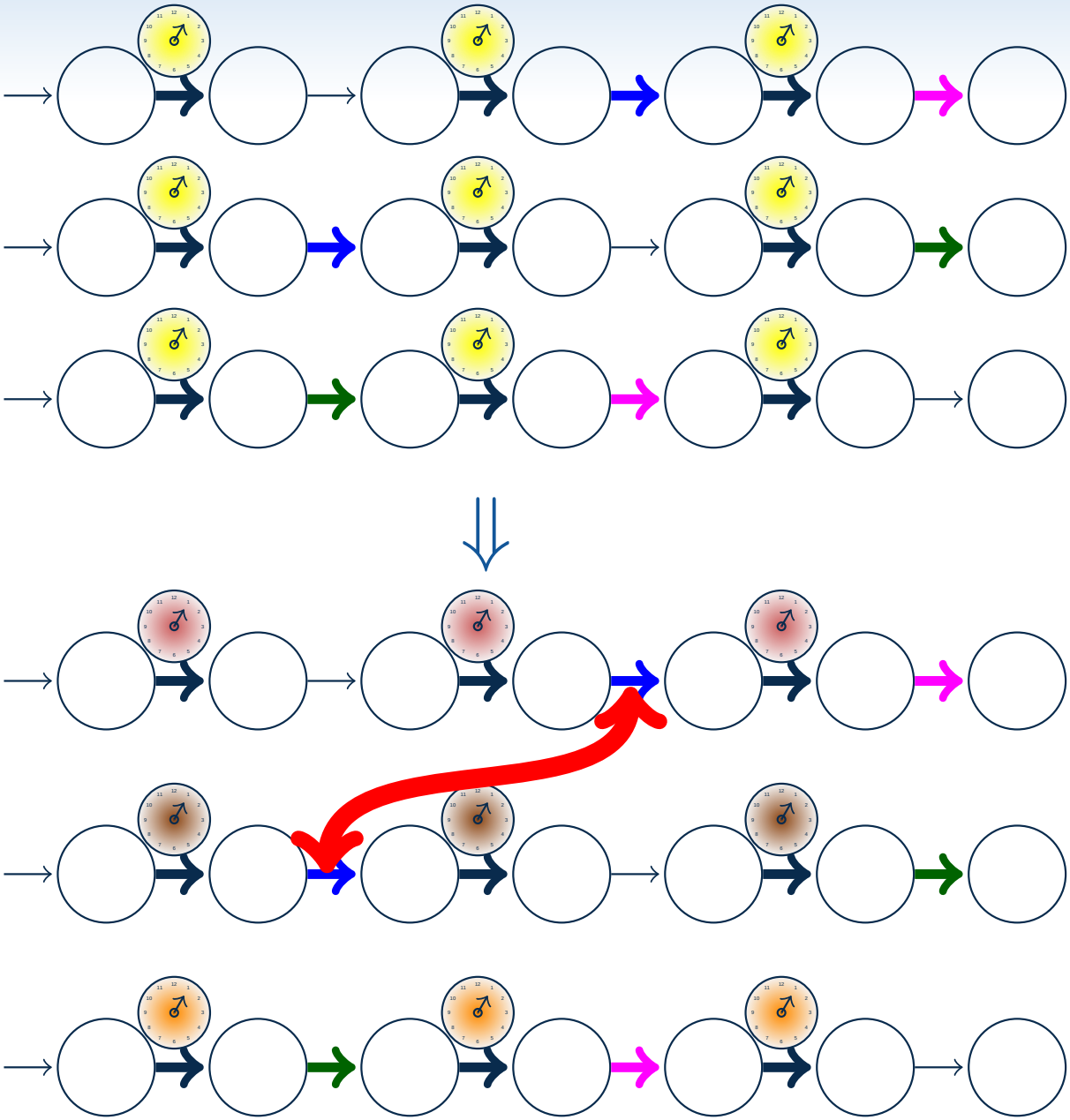
Alternative composition

- 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.

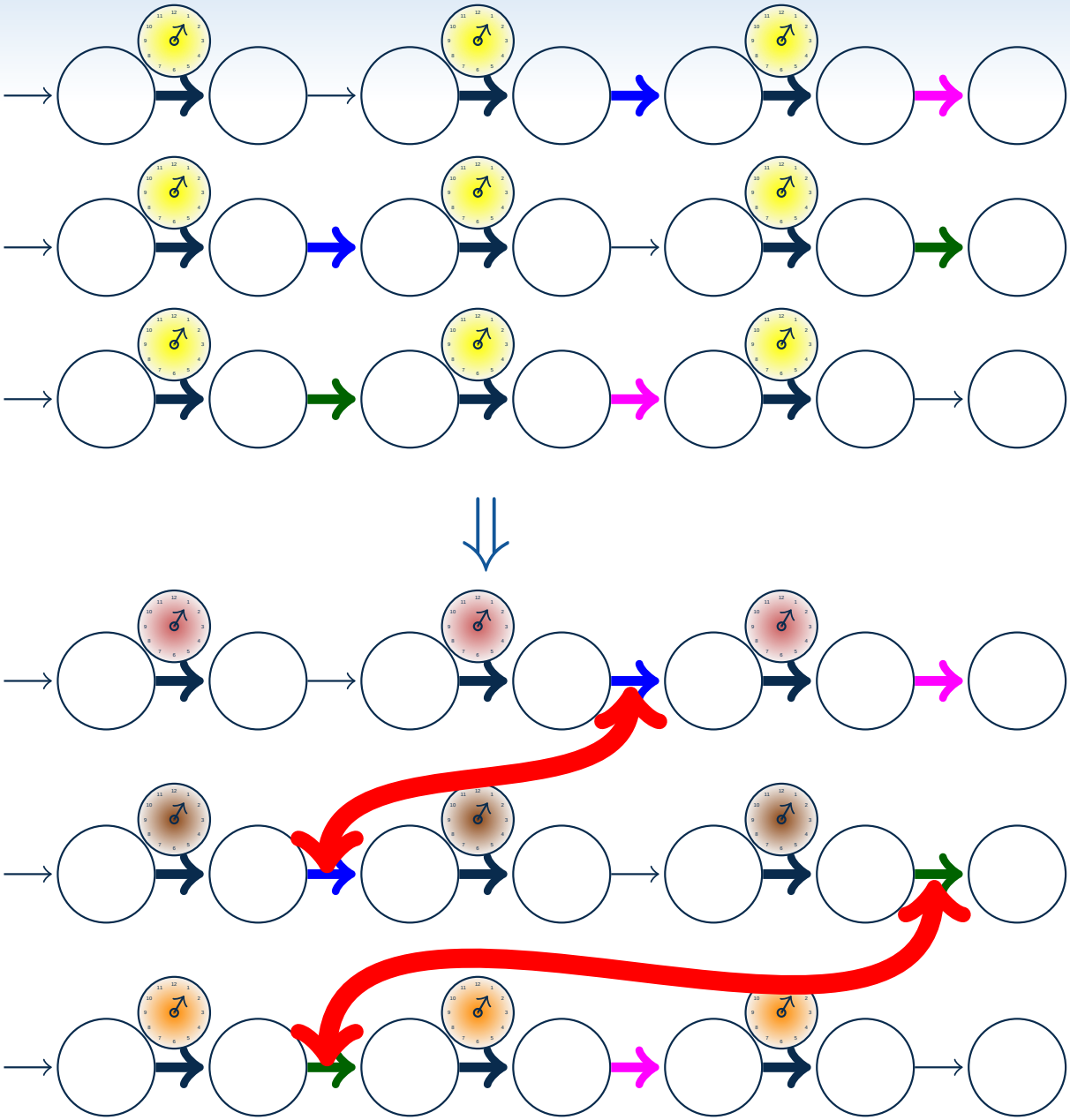
Local time effect



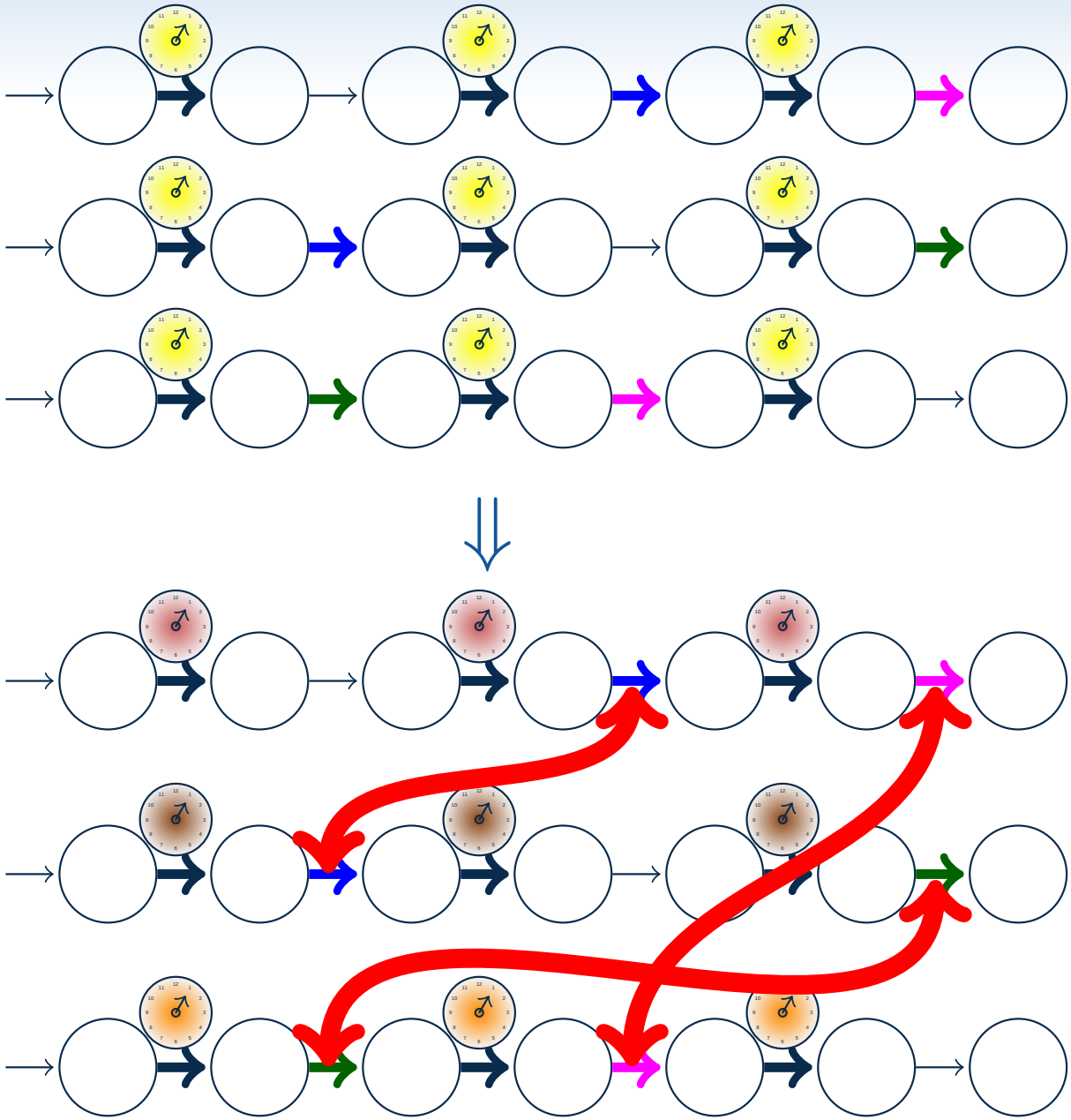
Local time effect



Local time effect



Local time effect



Symbolic transition system (Local time)

$$\textsc{Init} := \bigwedge_{q \in Q} (loc = q \rightarrow I_q(X)) \wedge t = 0$$

$$\textsc{Invar} := \bigwedge_{q \in Q} (loc = q \rightarrow Z_q(X))$$

$$\textsc{Trans} := \bigwedge_{q \in Q} (loc = q \rightarrow (\textsc{Stutter} \vee \textsc{Timed}_q \vee \bigvee_{(q,p) \in E} \textsc{Untimed}_{q,p}))$$

$$\textsc{Stutter} := \varepsilon = \mathsf{s} \wedge \delta = 0 \wedge loc' = loc \wedge X' = X \wedge t' = t$$

$$\textsc{Timed}_q := \varepsilon = \mathsf{t} \wedge \delta > 0 \wedge loc' = loc \wedge F_q(\frac{X' - X}{\delta}) \wedge t' = t + \delta$$

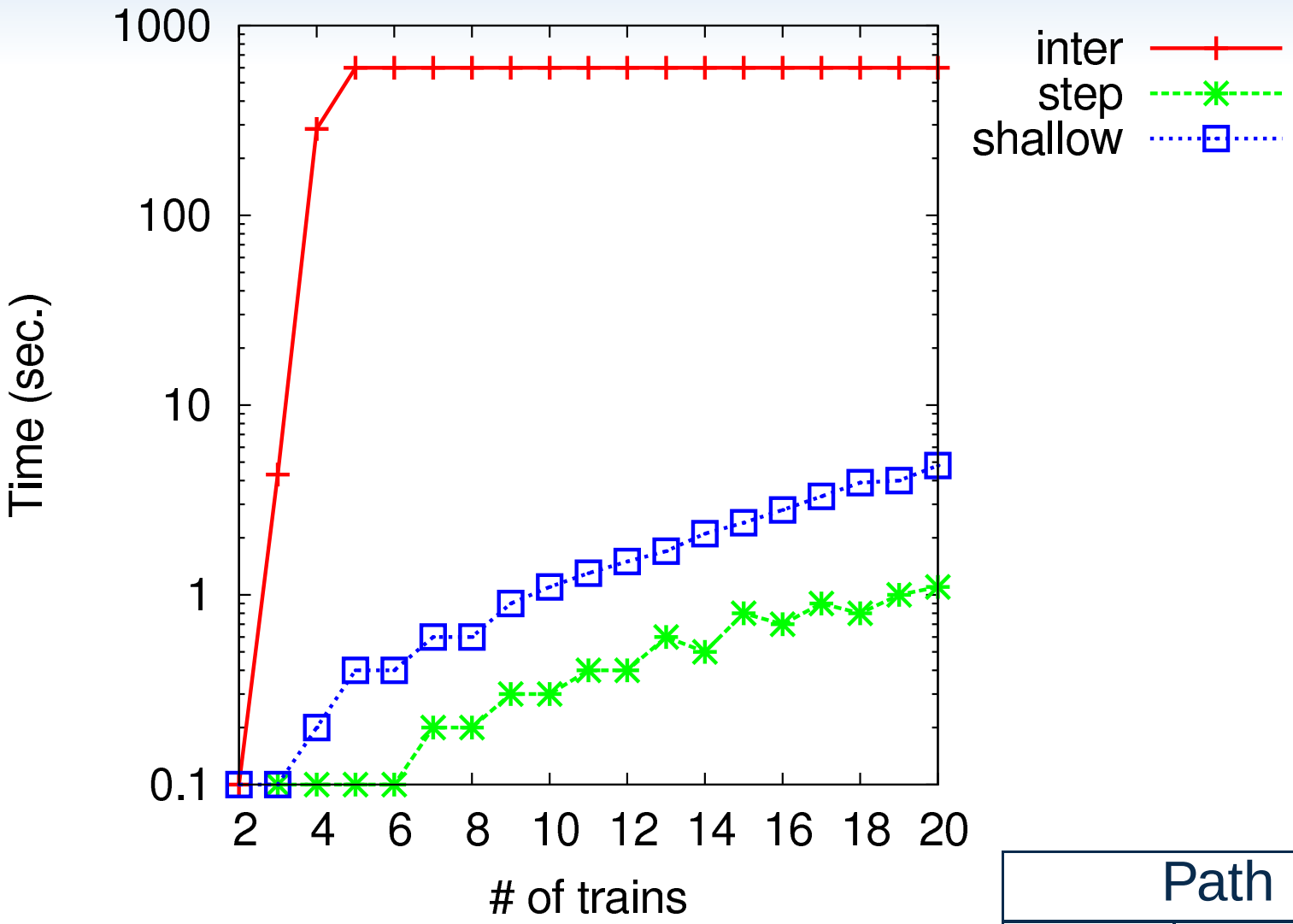
$$\textsc{Untimed}_{q,p} := \varepsilon = L_{q,p} \wedge \delta = 0 \wedge loc' = p \wedge J_{q,p}(X, X') \wedge t' = t$$

δ and t are local

Shallow synchronization

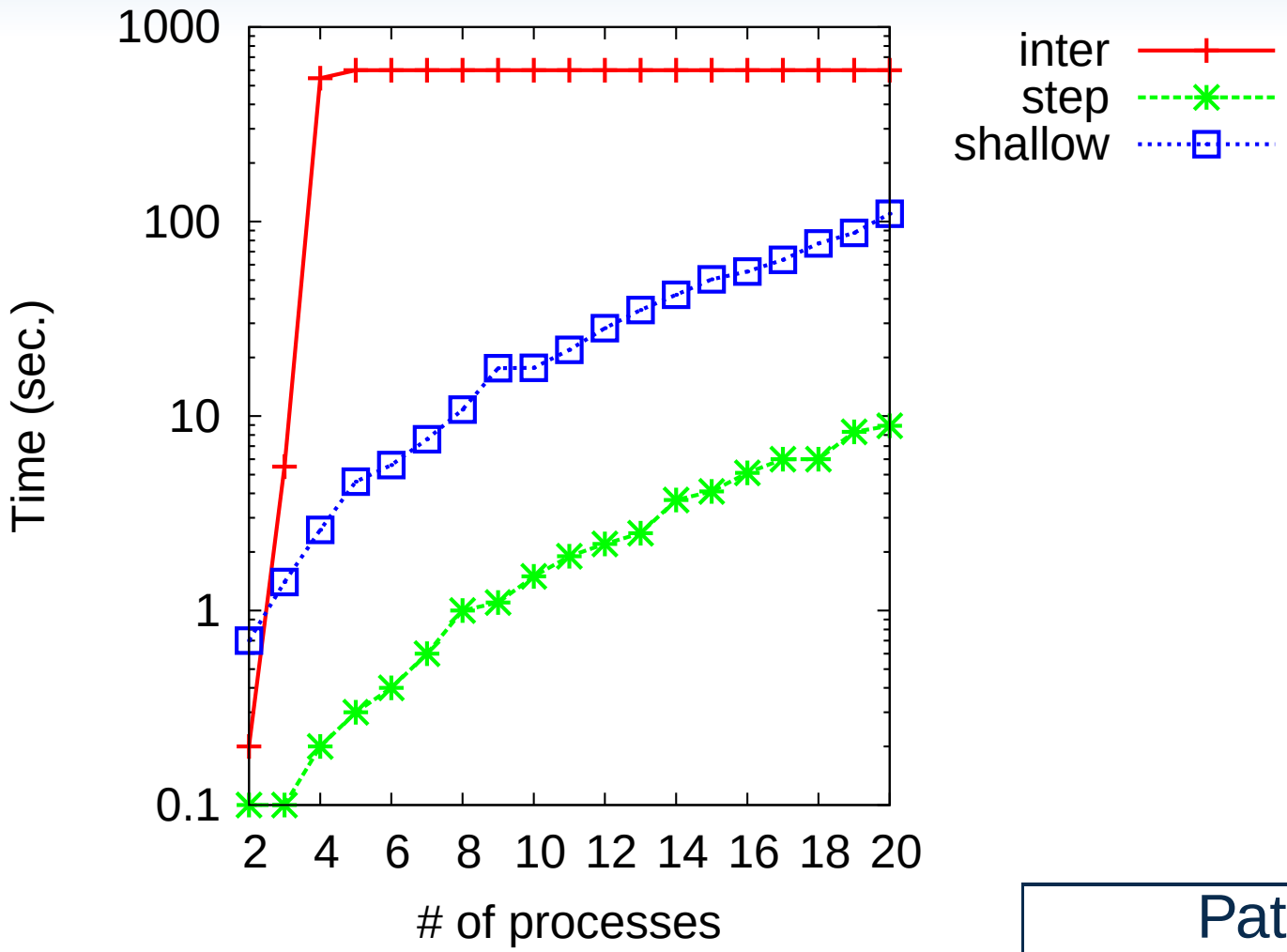
- Shallow synchronization:
  - for all systems S_j and S_h, the sequence of shared events performed by S_j and S_h is the same;
  - for all systems S_j and S_h, for all events a shared by S_j and S_h, S_j performs the i-th occurrence of a at the same time S_h performs the i-th occurrence of a;
  - for 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



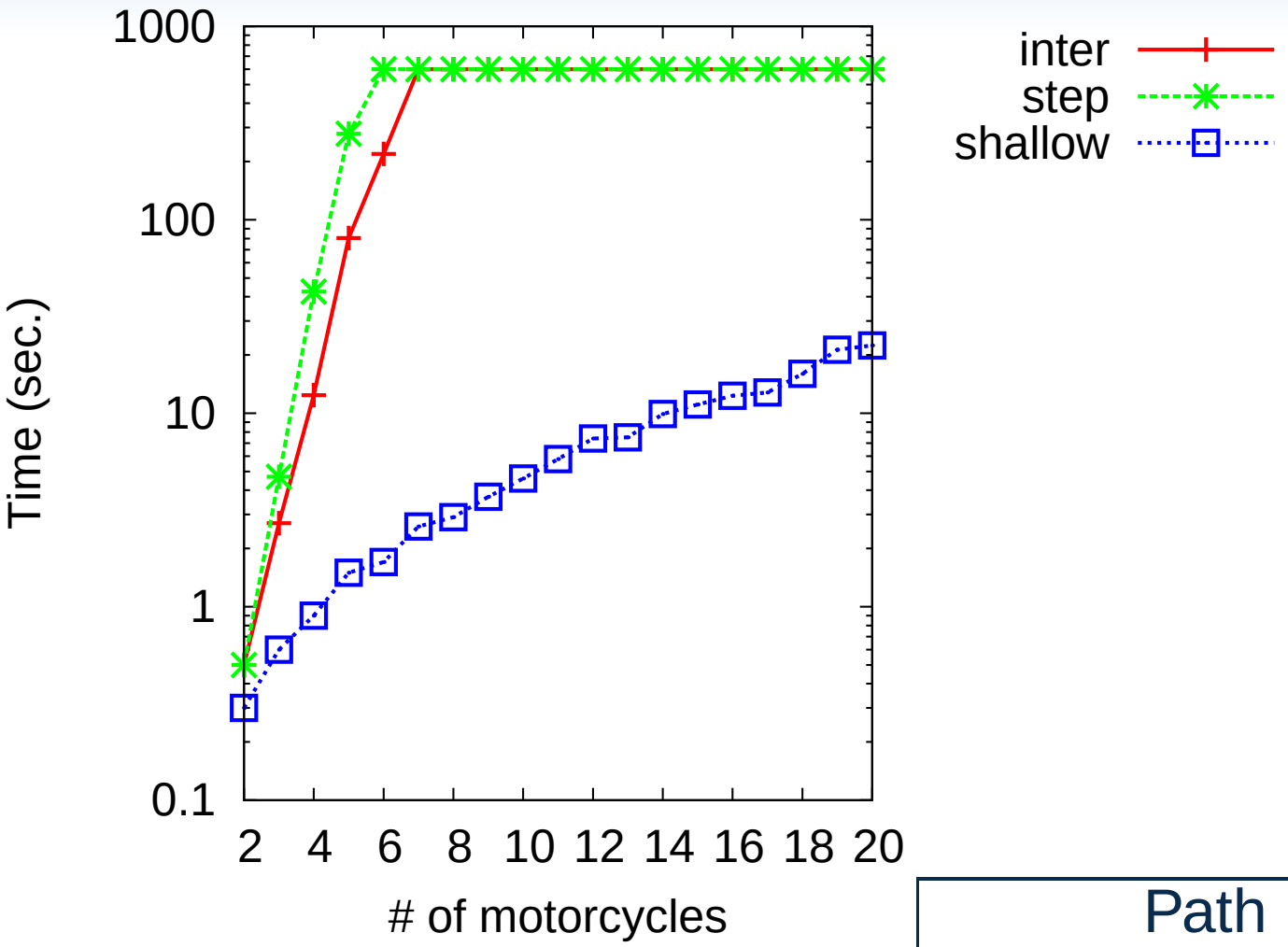
Path length		
Inter	Step	Shallow
5*n*	6	6

Ring-shape Fischer



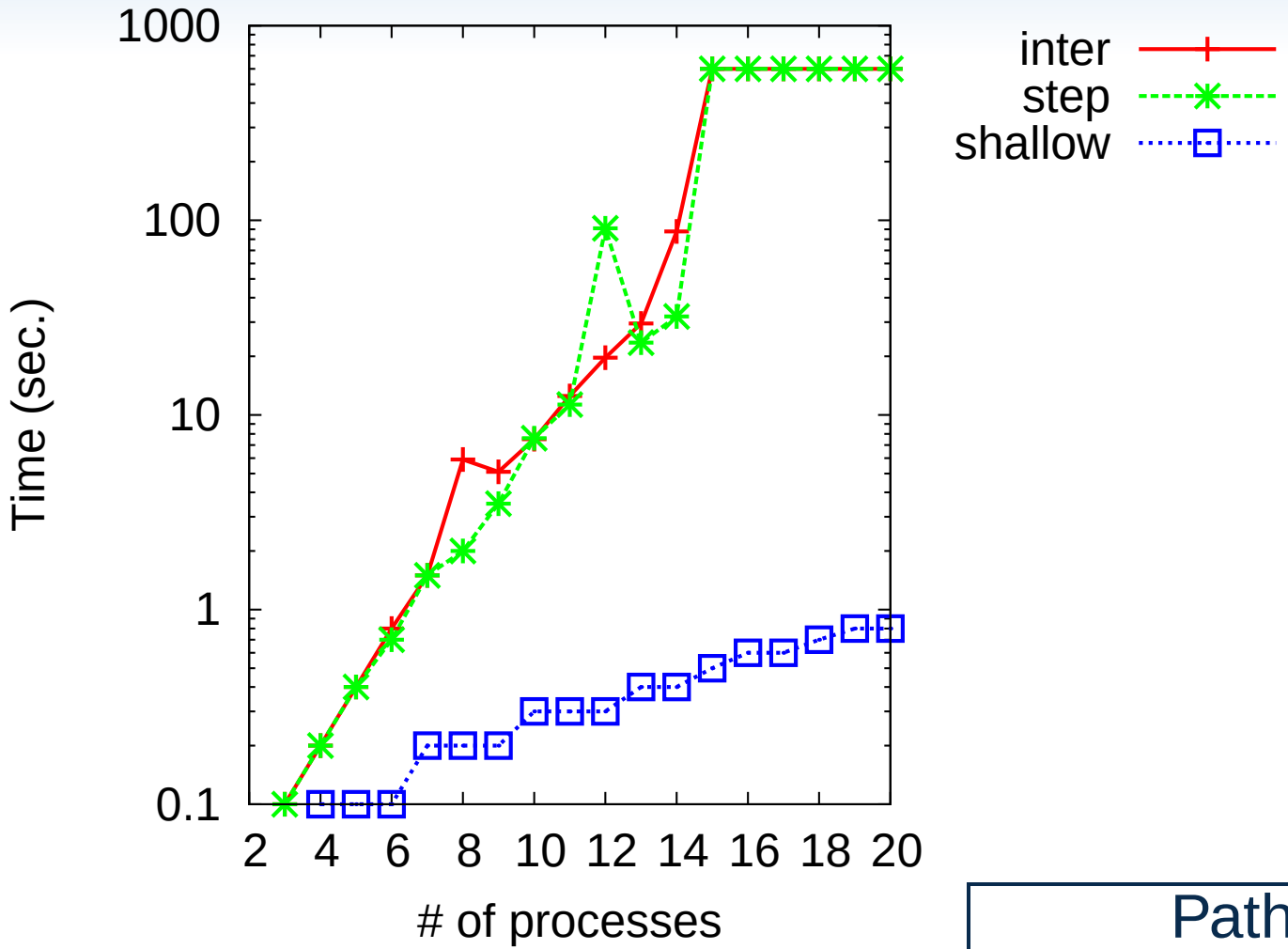
Path length		
Inter	Step	Shallow
$7n$	7	7

Motorcycle



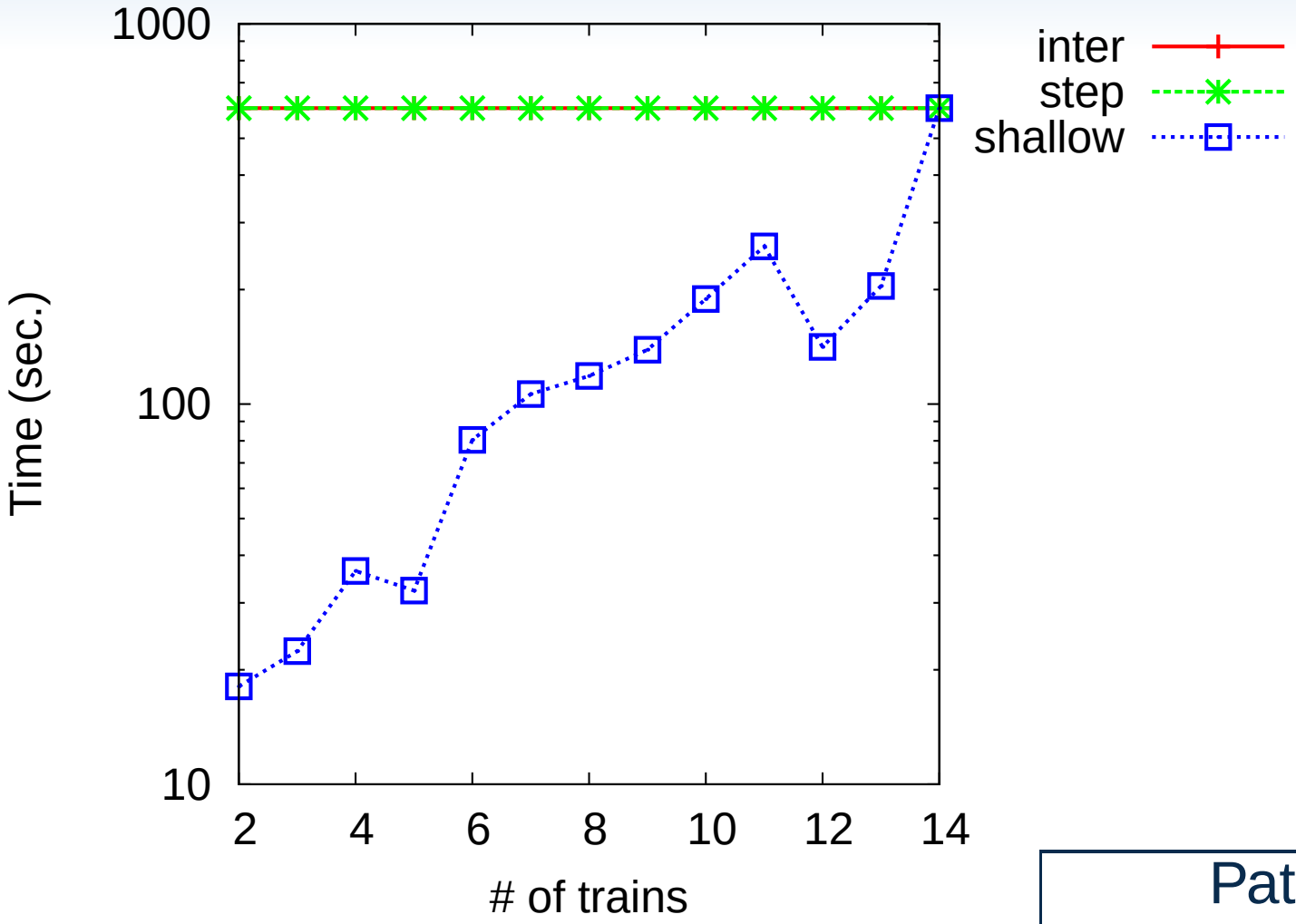
Path length		
Inter	Step	Shallow
$4n + 3$	$4n + 3$	7..9

FDDI



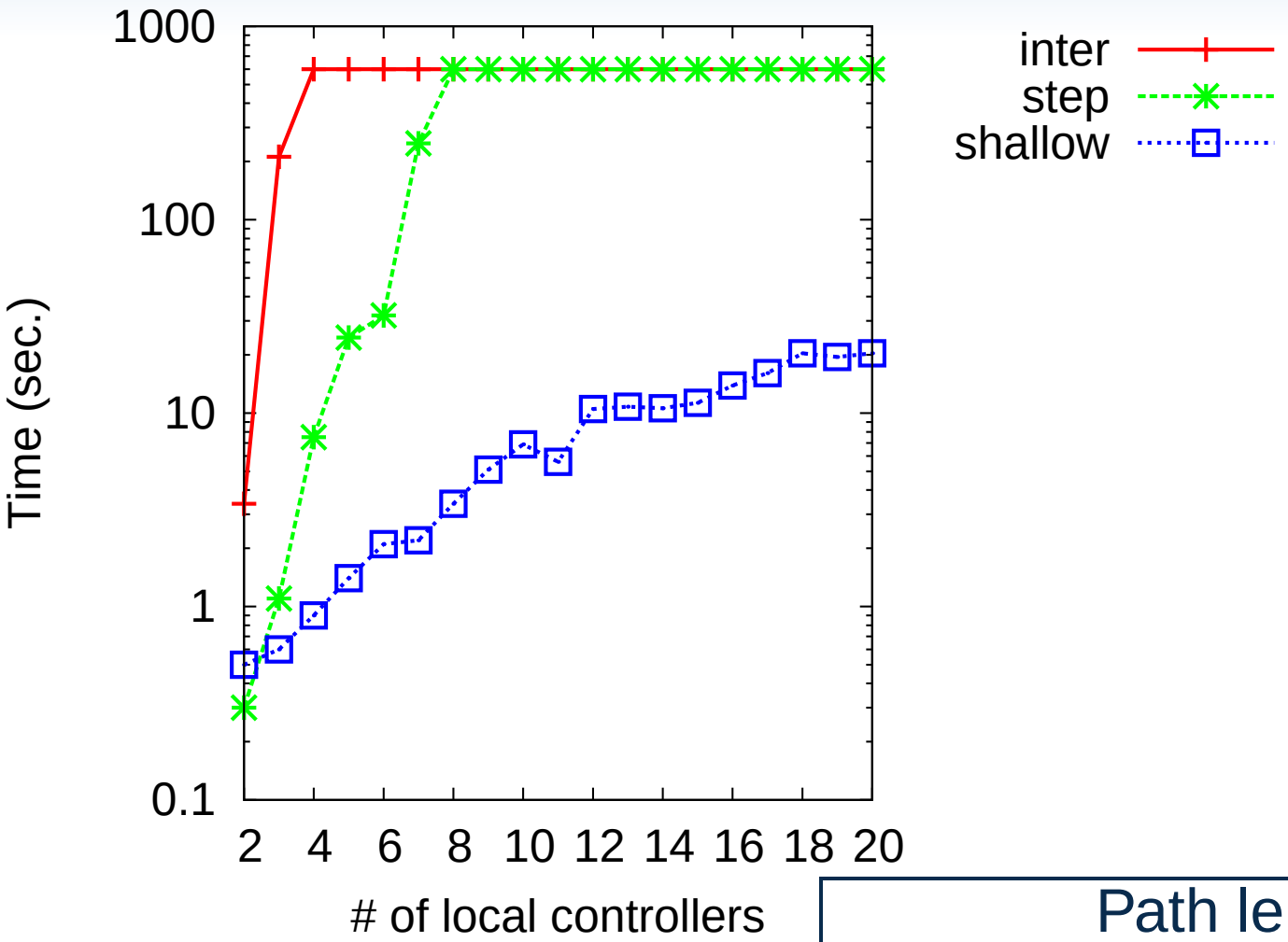
Path length		
Inter	Step	Shallow
$2n + 1$	5	3..5

ETCS



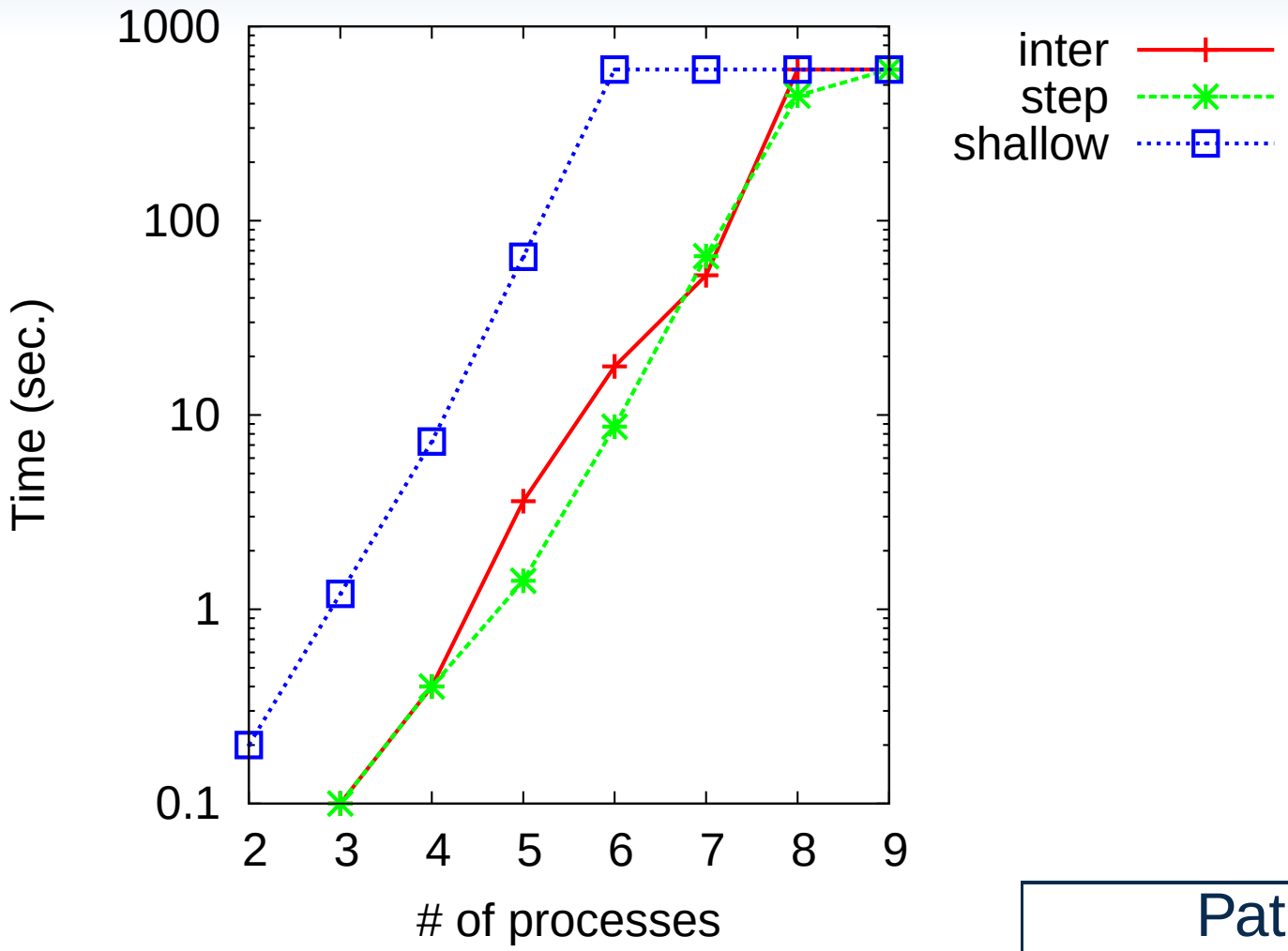
Path length		
Inter	Step	Shallow
NA	NA	17

Multi-Frequency



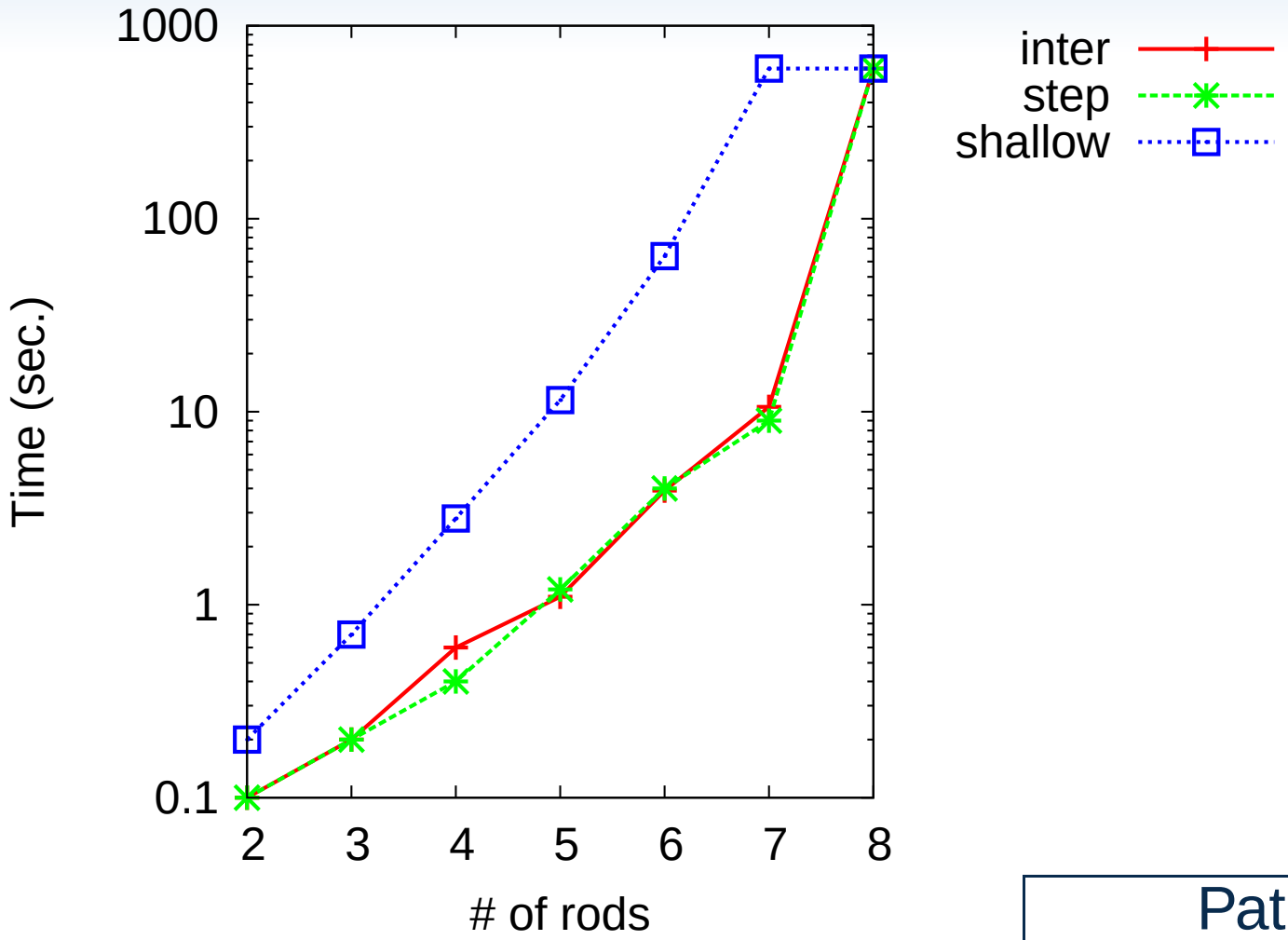
Path length		
Inter	Step	Shallow
NA	$3(n-1)..3n$	9

Star-shape Fischer



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

Nuclear reactor

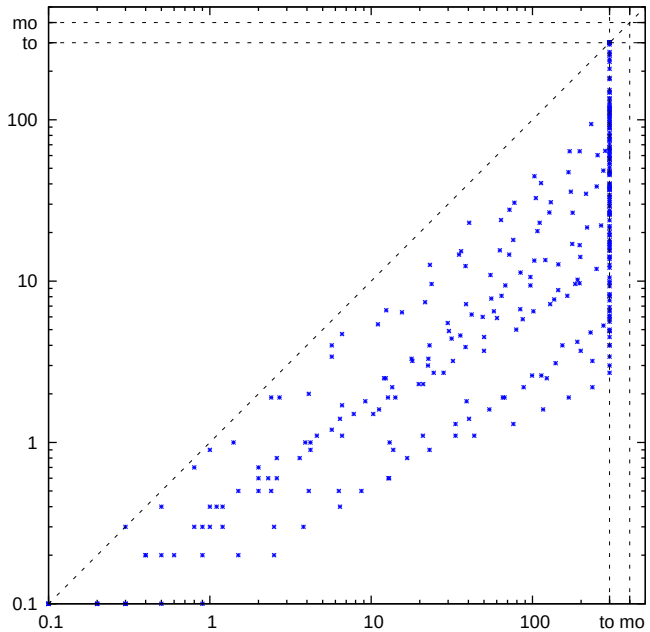


Path length		
Inter	Step	Shallow
$4n$	$4n$	$4n$

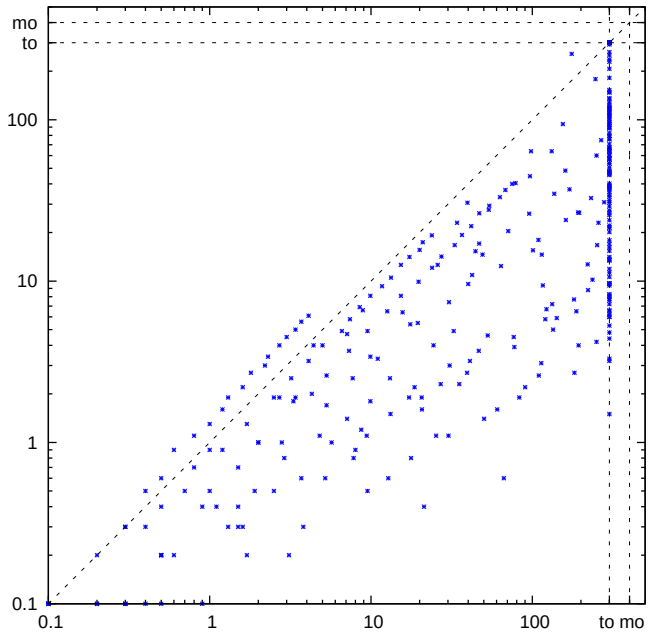
Scenario feasibility

- Scenarios are fundamental for early validation.
- Message Sequence Charts (MSCs) are at the core of many scenario languages (hierarchical MSC, LSCs, UML, ...).
- An MSC fixes a (partial-order) sequence of events that should be feasible in the network.
- Bottlenecks:
  - BMC dies in searching the right positioning of the MSC events in the k steps.
  - Interleaving and global time break the locality of the paths between two synchronizing events.
- Main idea of [CMT11]:
  - Exploit local-time encoding.
  - Improve incrementality by fixing the sequence of events and varying only the local path encoding.
  - Learn invariants on the local path based on the structure of the scenario.
- Dramatic improvement wrt. automata-based encoding.

Scatter Plots



(a) SCENARIO (y axes) vs. GLOBAL (x axes)



(b) SCENARIO (y axes) vs. DISTRIBLOCAL (x axes)

Proving scenario unfeasibility

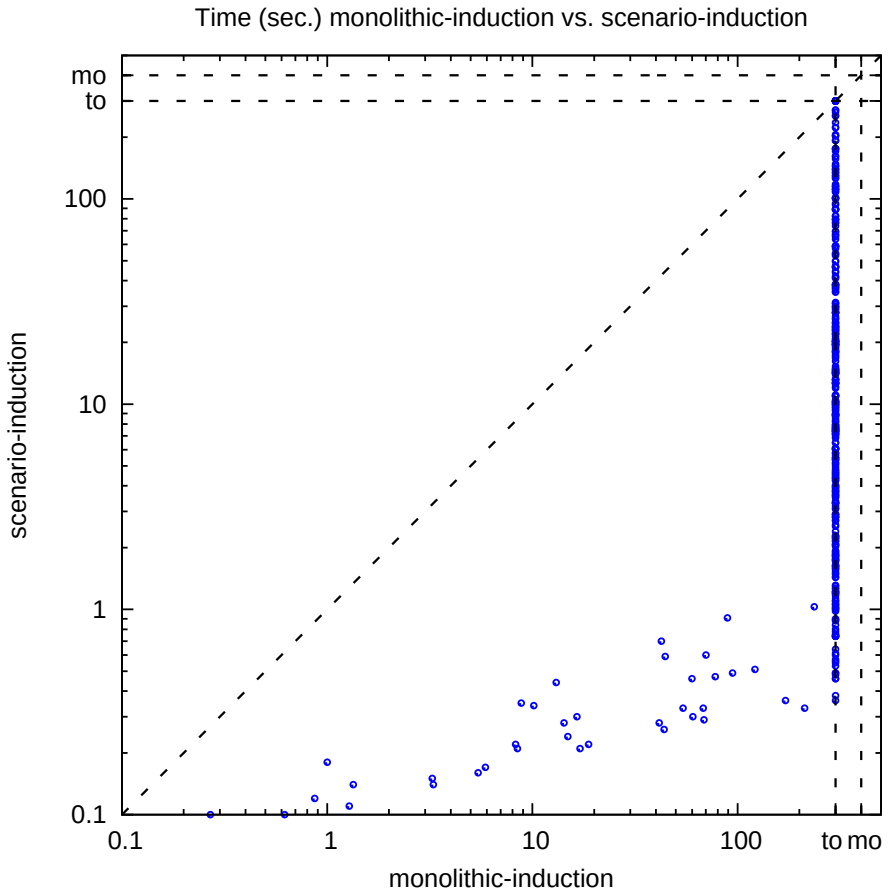
Partitioned k-induction:

- we fix the length of the local paths applying k-induction;
- details:
  - base case: encoding of the local path;
  - step case (localized simple path): every local path cannot reach new states.

k-induction for hybrid automata:

- we alternate discrete and timed transitions;
- in the case of loops (not so frequent on local paths) we apply abstraction techniques.

Scenario-based vs. automata-based k-induction



Run times (sec.): monolithic induction (x axes) vs. scenario-induction (y axes)

Explaining the unfeasibility

Kinds of explanations for the unfeasibility of the MSC m with additional constraints φ:

1. which parts of m and φ cannot be executed by the network?
2. why the network is inconsitent with φ?
3. why the i-th component is consistent with its instance but not with the rest of the scenario?

Explaining the unfeasibility

Kinds of explanations for the unfeasibility of the MSC m with additional constraints φ:

1. which parts of m and φ cannot be executed by the network? Extraction using **unsat cores**.
2. why the network is inconsitent with φ?
3. why the i-th component is consistent with its instance but not with the rest of the scenario?

Explaining the unfeasibility

Kinds of explanations for the unfeasibility of the MSC m with additional constraints φ:

1. which parts of m and φ cannot be executed by the network?
2. why the network is inconsitent with φ?
   Extraction via **interpolation**:
   - A = encoding of the network along m;
   - B = φ;
3. why the i-th component is consistent with its instance but not with the rest of the scenario?

Explaining the unfeasibility

Kinds of explanations for the unfeasibility of the MSC m with additional constraints φ:

1. which parts of m and φ cannot be executed by the network?
2. why the network is inconsitent with φ?
3. why the i-th component is consistent with its instance but not with the rest of the scenario?
   Extraction via **interpolation**:
   - A = encoding of the i-th component along its instance;
   - B = the remaining encoding;

Outline

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

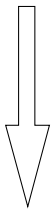
- Property-based approach:
  - One requirement, one formula.
  - Easy traceability.
  - Validation based on series of satisfiability problems:
    - consistent, i.e. if they do not contain some contradiction (sat of $\bigwedge_{1 \leq i \leq n} \varphi_i^{req}$)
    - not too strict, i.e. if they do 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 embedded systems, necessary to predicate over:
  - integer and real variables,
  - continuous quantities,
  - instantaneous changes,
  - continuous evolutions (constraints over derivatives).
- Our solution is HRELTL:
  - RELTL with the addition of:
    - continuous variables
    - arithmetic predicates with next and derivatives
  - Interpreted over hybrid traces.

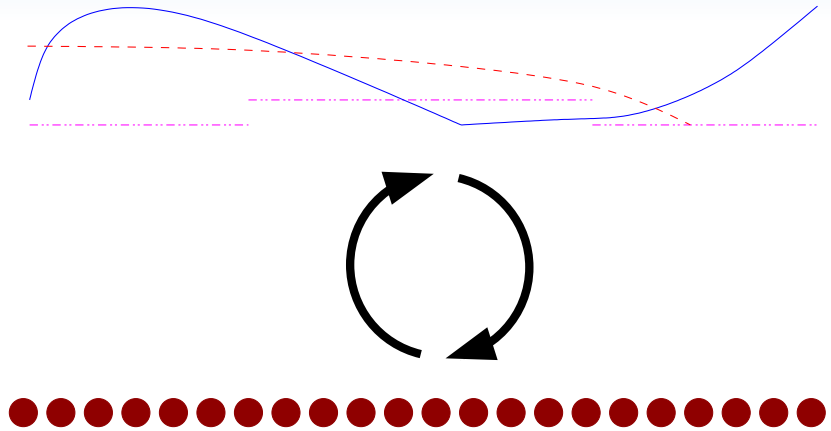
Reduction to discrete semantics

HRELTL



RELTL

(with SMT constraints)



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

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

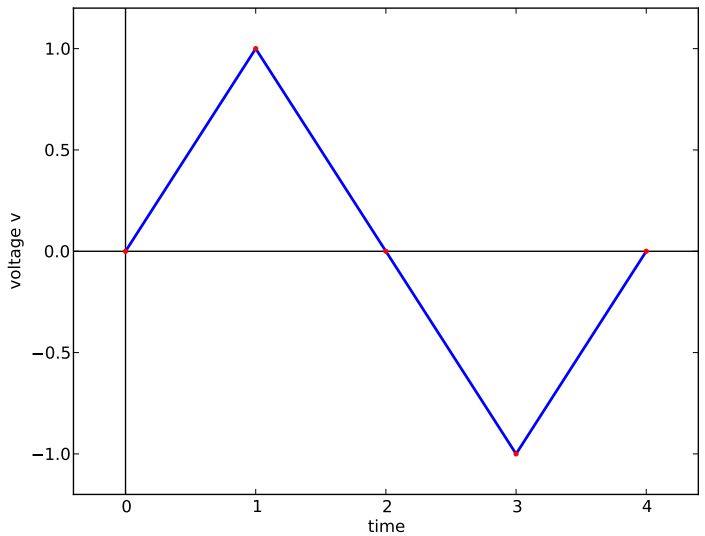
Theorem

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.

Satisfiability procedure

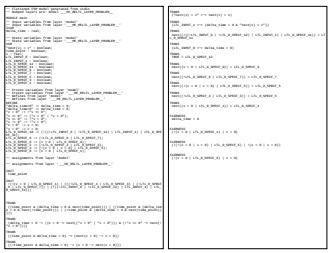
1. convert hybrid formula into discrete φ
2. build a fair transition system S_φ
3. check whether the language accepted by S_φ is not empty.

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)
```

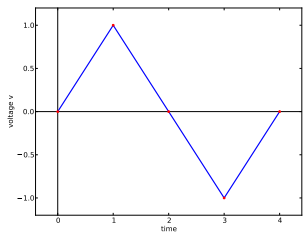
⇒

11 boolean variables
2 real variables
4 fairness conditions



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

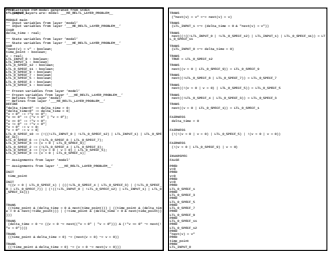
⇒ SAT




```
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

- Strong potential for SMT-based verification
- Different problems at different levels of abstraction
- Lifting SAT-based verification to SMT-based verification
- Many opportunities for different perspective (e.g. implicit abstraction, local time semantics) leveraging SMT-based verification

Future directions

- MathSAT: interpolation for BV, arrays, non-linear arithmetic
- Tighten integration with NuSMV core engines
- Shallow synchronization for non-linear hybrid automata
- Verification of embedded software and hybrid systems
- Formal requirements engineering: how to automate the formalization?

Next June, SAT'12 in Trento

Thanks for your attention

Bibliography I

[ABC+02] Gilles Audemard, Piergiorgio Bertoli, Alessandro Cimatti, Artur Kornilowicz, and Roberto Sebastiani.
A sat based approach for solving formulas over boolean and linear mathematical propositions.
In Voronkov [Vor02], pages 195–210.

[BBC+05a] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani.
Efficient satisfiability modulo theories via delayed theory combination.
In Etessami and Rajamani [ER05], pages 335–349.

[BBC+05b] 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.
In Halbwachs and Zuck [HZ05], pages 317–333.

[BBC+05c] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Peter van Rossum, Stephan Schulz, and Roberto Sebastiani.
Mathsat: Tight integration of sat and mathematical decision procedures.
J. Autom. Reasoning, 35(1-3):265–293, 2005.

[BCF+06a] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, Alessandro Santuari, and Roberto Sebastiani.
To ackermann-ize or not to ackermann-ize? on efficiently handling uninterpreted function symbols in *mt*(*uf* è).
In Miki Hermann and Andrei Voronkov, editors, *LPAR*, volume 4246 of *Lecture Notes in Computer Science*, pages 557–571. Springer, 2006.

[BCF+06b] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani.
Delayed theory combination vs. nelson-oppen for satisfiability modulo theories: A comparative analysis.
In Miki Hermann and Andrei Voronkov, editors, *LPAR*, volume 4246 of *Lecture Notes in Computer Science*, pages 527–541. Springer, 2006.

Bibliography II

[BCF+07] 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.
In Werner Damm and Holger Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 547–560. Springer, 2007.

[BCF+09] 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(1-2):63–99, 2009.

[BCG+09] Dirk Beyer, Alessandro Cimatti, Alberto Griggio, M. Erkan Keremoglu, and Roberto Sebastiani.
Software model checking via large-block encoding.
In *FMCAD*, pages 25–32. IEEE, 2009.

[BCL+10] Lei Bu, Alessandro Cimatti, Xuandong Li, Sergio Mover, and Stefano Tonetta.
Model checking of hybrid systems using shallow synchronization.
In John Hatcliff and Elena Zucca, editors, *FMOODS/FORTE*, volume 6117 of *Lecture Notes in Computer Science*, pages 155–169. Springer, 2010.

[CCF+07] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar.
Computing predicate abstractions by integrating bdds and smt solvers.
In *FMCAD*, pages 69–76. IEEE Computer Society, 2007.

[CCM+09] Roberto Cavada, Alessandro Cimatti, Alessandro Mariotti, Cristian Mattarei, Andrea Micheli, Sergio Mover, Marco Pensallorto, Marco Roveri, Angelo Susi, and Stefano Tonetta.
Supporting requirements validation: The eurailcheck tool.
In *ASE*, pages 665–667. IEEE Computer Society, 2009.

Bibliography III

[CCM+10] Angelo Chiappini, Alessandro Cimatti, Luca Macchi, Oscar Rebollo, Marco Roveri, Angelo Susi, Stefano Tonetta, and Berardino Vittorini.
Formalization and validation of a subset of the european train control system.
In Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel, editors, *ICSE (2)*, pages 109–118. ACM, 2010.

[CDJR09] Alessandro Cimatti, Jori Dubrovin, Tommi A. Junttila, and Marco Roveri.
Structure-aware computation of predicate abstraction.
In *FMCAD*, pages 9–16. IEEE, 2009.

[CFG+10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico.
Satisfiability modulo the theory of costs: Foundations and applications.
In Javier Esparza and Rupak Majumdar, editors, *TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 99–113. Springer, 2010.

[CGM+11] A. Cimatti, A. Griggio, A. Micheli, I. Narasamdya, and M. Roveri.
Kratos: A Software Model Checker for SystemC.
In *CAV*, 2011.
To appear.

[CGS07] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani.
A simple and flexible way of computing small unsatisfiable cores in sat modulo theories.
In João Marques-Silva and Karem A. Sakallah, editors, *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 334–339. Springer, 2007.

[CGS08] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani.
Efficient interpolant generation in satisfiability modulo theories.
In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 397–412. Springer, 2008.

[CGS09] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani.
Interpolant generation for utvpi.
In Renate A. Schmidt, editor, *CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2009.

Bibliography IV

[CGS10] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani.
Efficient generation of craig interpolants in satisfiability modulo theories.
ACM Trans. Comput. Log., 12(1):7, 2010.

[CGS11] Alessandro Cimatti, Alberto Griggio, and Roberto Sebastiani.
Computing small unsatisfiable cores in satisfiability modulo theories.
J. Artif. Intell. Res. (JAIR), 40:701–728, 2011.

[CMNR10] Alessandro Cimatti, Andrea Micheli, Iman Narasamdya, and Marco Roveri.
Verifying systemc: A software model checking approach.
In *FMCAD*, pages 51–59. IEEE, 2010.

[CMT11] A. Cimatti, S. Mover, and S. Tonetta.
Efficent Scenario Verification for Hybrid Automata.
In *CAV*, 2011.

[CNR11] Alessandro Cimatti, Iman Narasamdya, and Marco Roveri.
Boosting lazy abstraction for systemc with partial order reduction.
In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 341–356. Springer, 2011.

[CRST10] Alessandro Cimatti, Marco Roveri, Angelo Susi, and Stefano Tonetta.
Formalization and validation of safety-critical requirements.
Electronic Proceedings in Theoretical Computer Science (EPTCS), 20:68–75, 2010.
Edited by: Manuela Bujorianu and Michael Fisher. Proceedings FM-09 Workshop on Formal Methods for Aerospace Eindhoven, The Netherlands, 3rd November 2009.

[CRST11a] Alessandro Cimatti, Marco Roveri, Angelo Susi, and Stefano Tonetta.
Formalizing requirements with object models and temporal constraints.
Software and System Modeling, 10(2):147–160, 2011.

Bibliography V

[CRST11b] Alessandro Cimatti, Marco Roveri, Angelo Susi, and Stefano Tonetta.
Validation of Requirements for Hybrid Systems: a Formal Approach.
ACM Trans. Softw. Eng. Methodol., 2011.
To appear.

[ER05] Kousha Etessami and Sriram K. Rajamani, editors.
Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings, volume 3576 of *Lecture Notes in Computer Science*. Springer, 2005.

[FCN+10] Anders Franzén, Alessandro Cimatti, Alexander Nadel, Roberto Sebastiani, and Jonathan Shalev.
Applying smt in symbolic execution of microcode.
In *FMCAD*, pages 121–128. IEEE, 2010.

[GLS11] Alberto Griggio, Thi Thieu Hoa Le, and Roberto Sebastiani.
Efficient interpolant generation in satisfiability modulo linear integer arithmetic.
In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *TACAS*, volume 6605 of *Lecture Notes in Computer Science*, pages 143–157. Springer, 2011.

[HZ05] Nicolas Halbwachs and Lenore D. Zuck, editors.
Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings, volume 3440 of *Lecture Notes in Computer Science*. Springer, 2005.

[LNO06] Shuvendu K. Lahiri, Robert Nieuwenhuis, and Albert Oliveras.
Smt techniques for fast predicate abstraction.
In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 2006.

[Ton09] Stefano Tonetta.
Abstract model checking without computing the abstraction.
In Ana Cavalcanti and Dennis Dams, editors, *FM*, volume 5850 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2009.

Bibliography VI

[Vor02] Andrei Voronkov, editor.
Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings, volume 2392 of *Lecture Notes in Computer Science*. Springer, 2002.