Parameterized Systems

Parosh Aziz Abdulla

Uppsala University

December 16, 2009

(Joint work with Noomene Ben Henda, Giorgio Delzanno, and Ahmed Rezine.)

Model Checking

Model Checking

$Model \models (Safety)\ Property$

Classical Approach

Finite-State Systems

Challenge: Infinite-State Systems

- Unbounded data structure
  - clocks, stacks, queues, counters, etc.
- Parametrization
  - mutual exclusion protocols, cache coherence protocols, multi-threaded programs, etc.

Model Checking

Model Checking

$Model \models (Safety)\ Property$

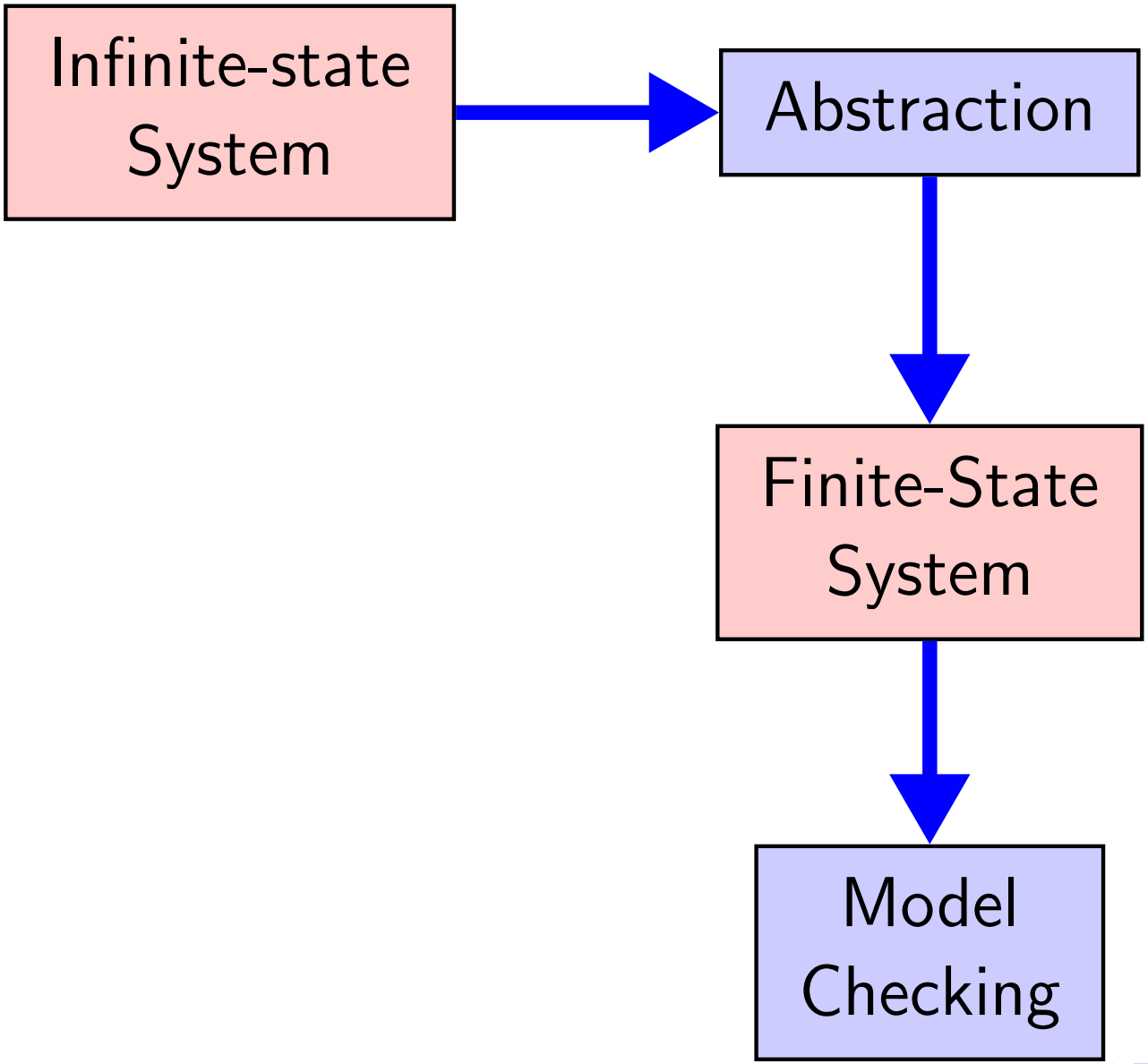
Classical Approach

Finite-State Systems

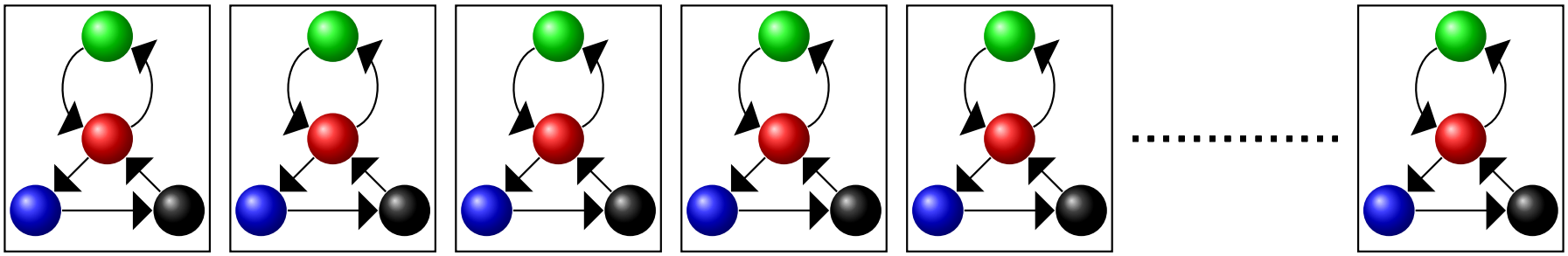
Challenge: Infinite-State Systems

- Unbounded data structure
  - clocks, stacks, queues, counters, etc.
- Parametrization $\longleftarrow$ subject of this talk
  - mutual exclusion protocols, cache coherence protocols, multi-threaded programs, etc.

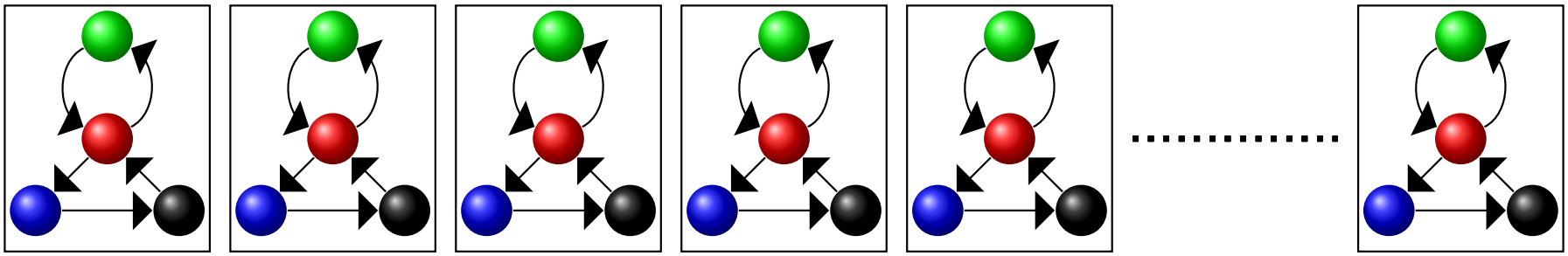
Model Checking+Abstraction

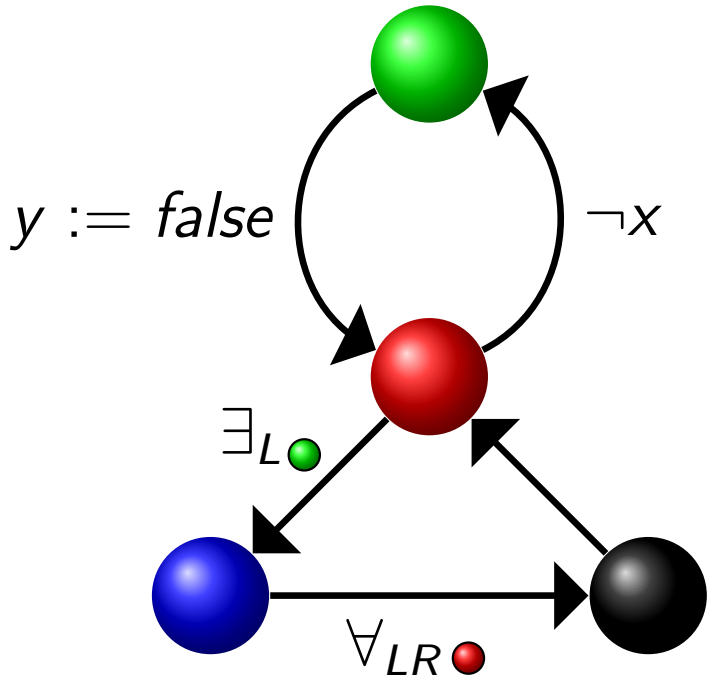


Parameterized Systems (of finite-state processes)

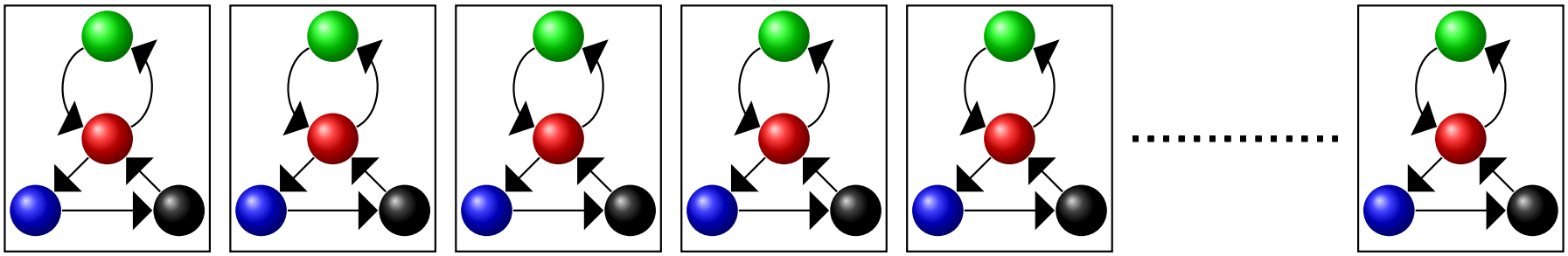


Parameterized Systems (of finite-state processes)





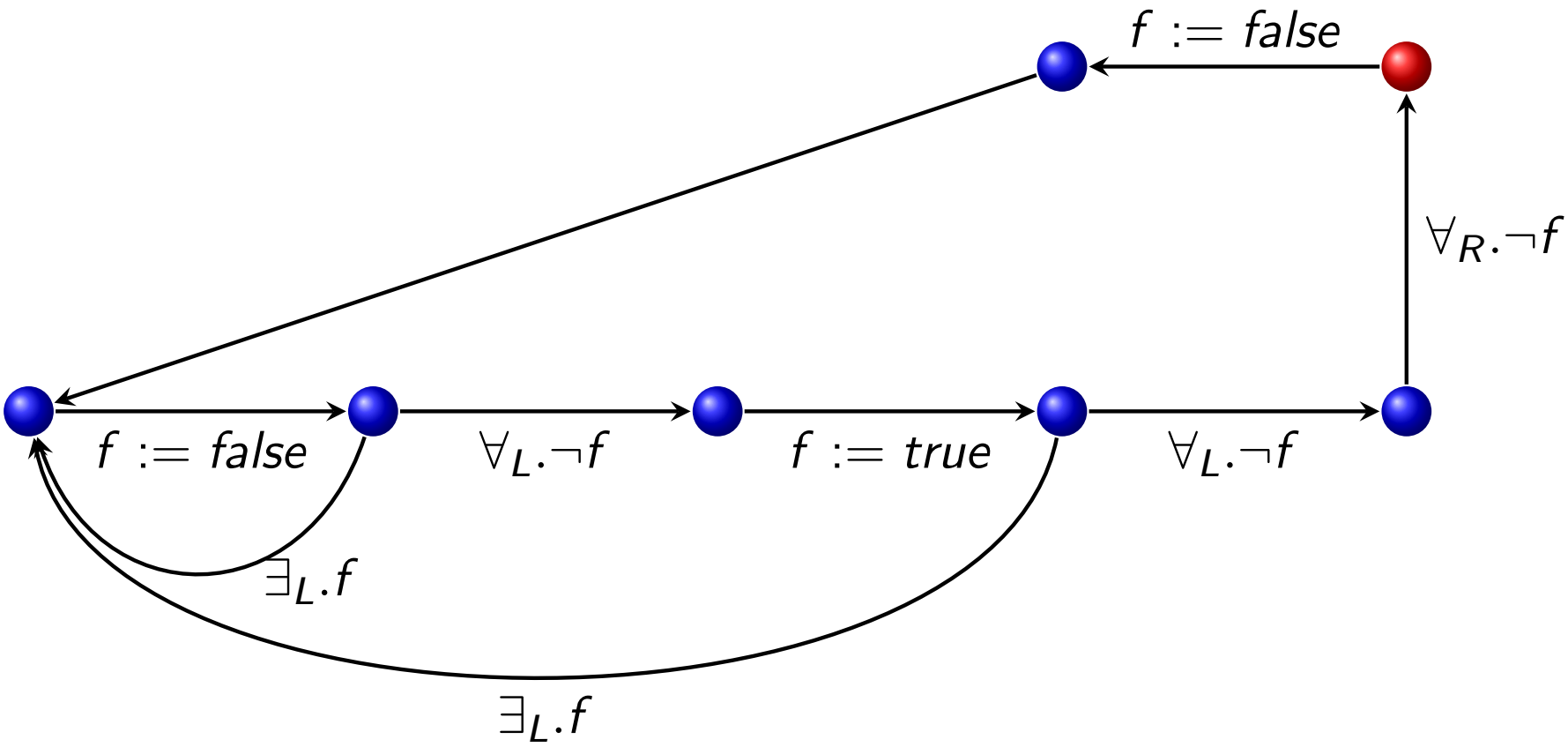
Parameterized Systems (of finite-state processes)

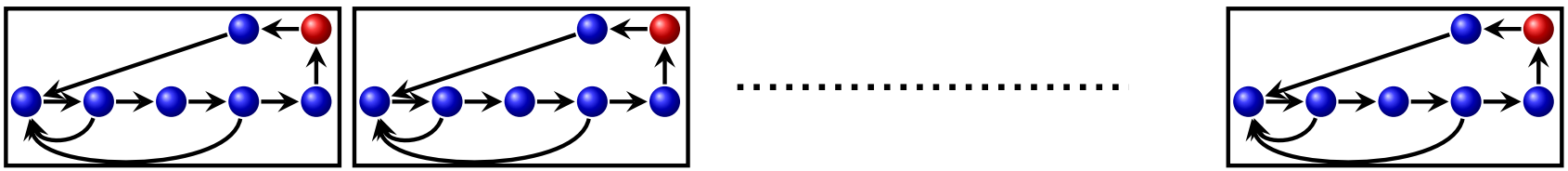


Remarks

- Infinite-state system:
  - unbounded number of processes.
  - Parameterized Verification: verify correctness regardless of the number of processes.
- Problem undecidable in general.
  - Challenge: find abstractions which work often.

Parameterized Burns' Protocol





German Protocol

Instance

Q : q_{inv}, q_{sh}, q_{exc}; q_{all} is any state in Q.

X : $curClt, sLst, iLst \in \mathcal{B}, ch_1 \in \{\epsilon, reqSh, reqExc\}$, $ch_2 \in \{\epsilon, grantSh, grantExc, inval\}$, $ch_3 \in \{\epsilon, invAck\}$

S : $excGran \in \mathcal{B}, curCm \in \{\epsilon, reqSh, reqExc\}$

T:

$$h_0 : \left[\begin{array}{c} q_{all} \\ curCm = reqSh, \neg excGran, ch_2 = \epsilon, curClt \\ \longrightarrow curCm = \epsilon, sLst, ch_2 = grantSh \\ q_{all} \end{array}\right]$$

$$h_1 : \left[\begin{array}{c} q_{all} \\ curCm = reqExc, ch_2 = \epsilon, \forall_{IR} \neg sLst, \neg sLst \\ \longrightarrow curCm = \epsilon, sLst, excGran, ch_2 = grantExc \\ q_{all} \end{array}\right]$$

$$h_2 : \left[\begin{array}{c} q_{all} \\ curCm = \epsilon, ch_1 \neq \epsilon \\ \longrightarrow curCm = ch_1, ch_1 = \epsilon, iLst = sLst, curClt \\ q_{all} \end{array}\right] \left[\begin{array}{c} q_{all} \\ \mathbf{tt} \\ \longrightarrow iLst = sLst, \neg curClt \\ q_{all} \end{array}\right]^*$$

$$h_3 : \left[\begin{array}{c} q_{all} \\ curCm = reqSh, excGran, iLst, ch_2 = \epsilon \\ \longrightarrow \neg iLst, ch_2 = inval \\ q_{all} \end{array}\right]$$

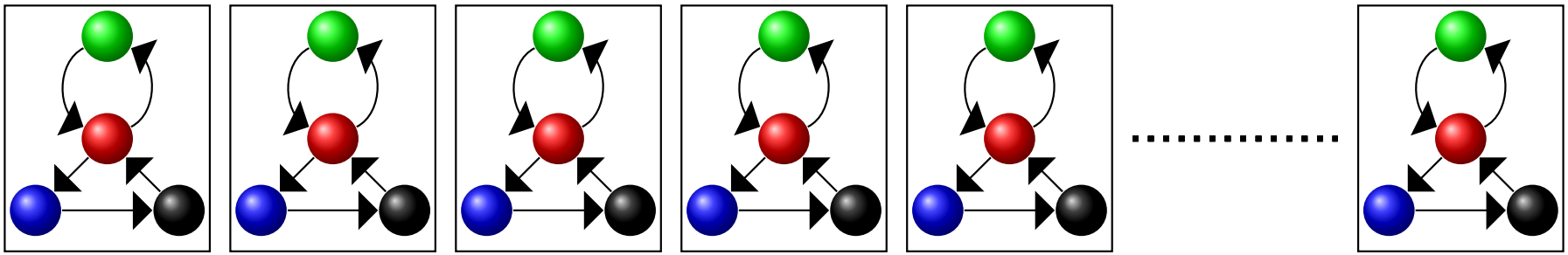
$$h_4 : \left[\begin{array}{c} q_{all} \\ curCm = reqExc, iLst, ch_2 = \epsilon \\ \longrightarrow \neg iLst, ch_2 = inval \\ q_{all} \end{array}\right] \quad h_5 : \left[\begin{array}{c} q_{all} \\ curCm \neq \epsilon, ch_3 = invAck \\ \longrightarrow \neg sLst, \neg excGran, ch_3 = \epsilon \\ q_{all} \end{array}\right]$$

$$p_1 : \left[\begin{array}{c} q_{inv} \\ ch_1 = \epsilon \\ \longrightarrow\ ch_1 = reqSh \\ q_{inv} \end{array}\right] \qquad p_2 : \left[\begin{array}{c} q_{inv} \\ ch_1 = \epsilon \\ \longrightarrow\ ch_1 = reqExc \\ q_{inv} \end{array}\right] \qquad p_3 : \left[\begin{array}{c} q_{sh} \\ ch_1 = \epsilon \\ \longrightarrow\ ch_1 = reqExc \\ q_{sh} \end{array}\right]$$

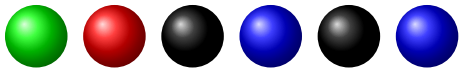
$$p_4 : \left[\begin{array}{c} q_{all} \\ ch_2 = inval, ch_3 = \epsilon \\ \longrightarrow\ ch_2 = \epsilon, ch_3 = invAck \\ q_{inv} \end{array}\right] \qquad p_5 : \left[\begin{array}{c} q_{all} \\ ch_2 = grantSh \\ \longrightarrow\ ch_2 = \epsilon \\ q_{sh} \end{array}\right] \qquad p_6 : \left[\begin{array}{c} q_{all} \\ ch_2 = grantExc \\ \longrightarrow\ ch_2 = \epsilon \\ q_{exc} \end{array}\right]$$

Initial State of Shared Vars: $excGran \mapsto \mathbf{ff}, curCm \mapsto \epsilon$
Initial Process State: u_{init}: $q_{inv}, (ch_1, ch_2, ch_3, curClt, sLst, iLst) \mapsto (\epsilon, \epsilon, \epsilon, \mathbf{ff}, \mathbf{ff}, \mathbf{ff})$
Final Constraints: Φ_F: $q_{exc}\,q_{exc}$, $q_{sh}\,q_{exc}$, $q_{exc}\,q_{sh}$

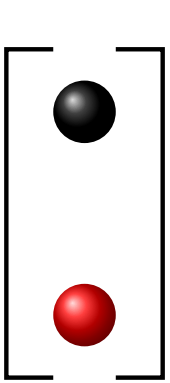
Configurations

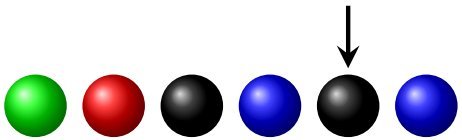


configuration $\longrightarrow$

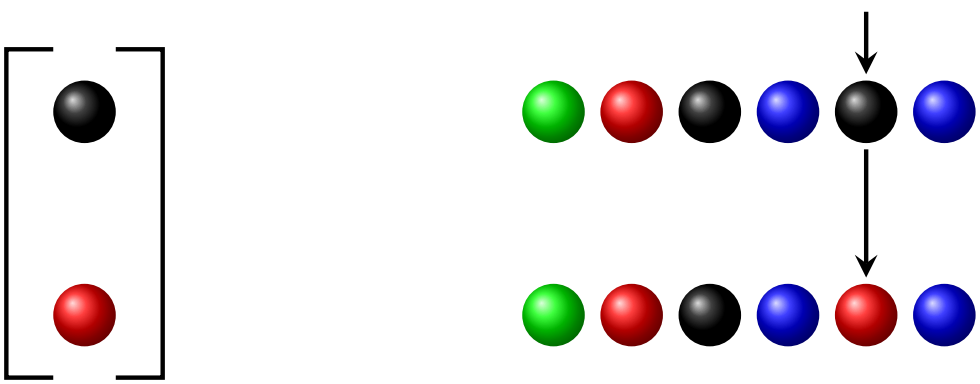


Local Transitions

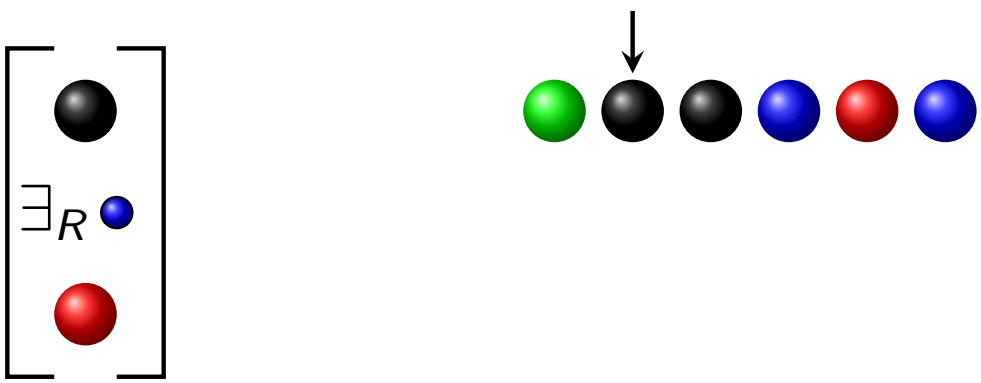




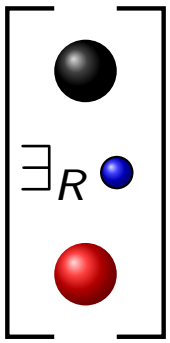
Local Transitions

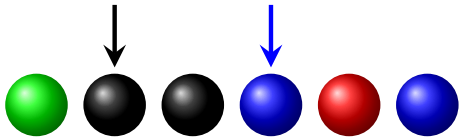


Existential Global Transitions

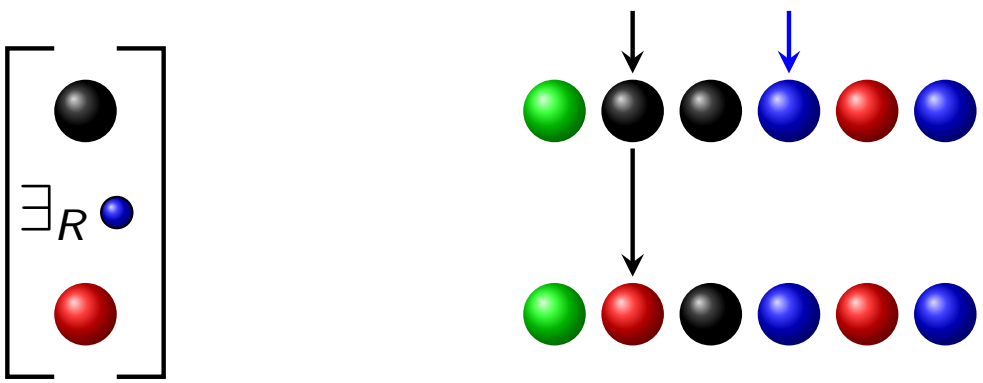


Existential Global Transitions

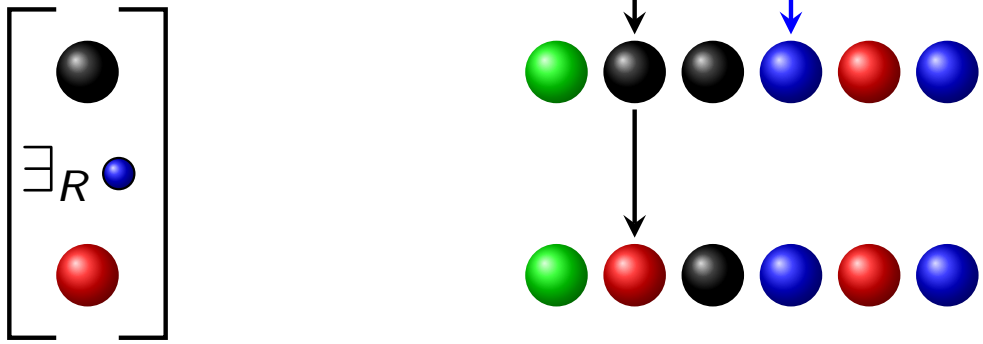


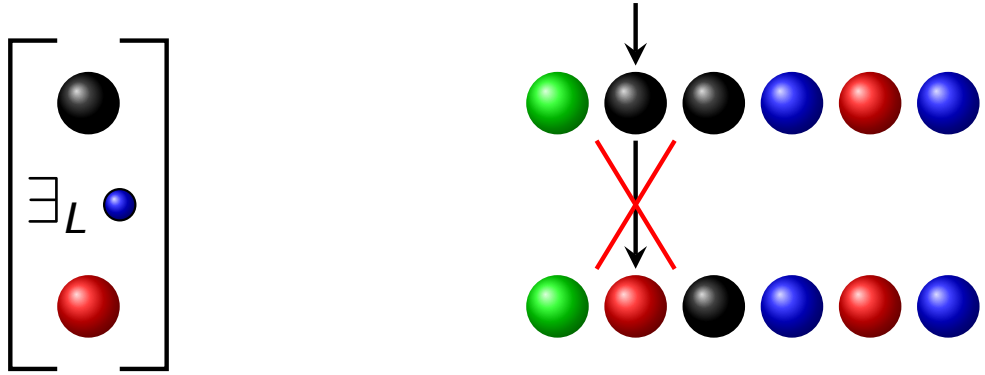


Existential Global Transitions

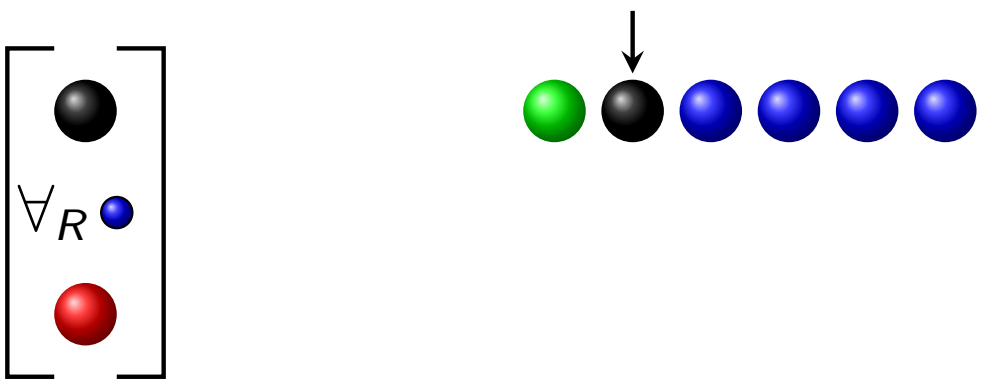


Existential Global Transitions

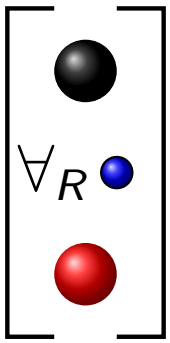


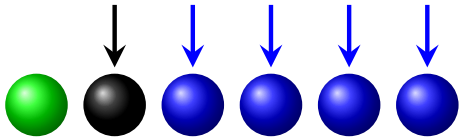


Universal Global Transitions

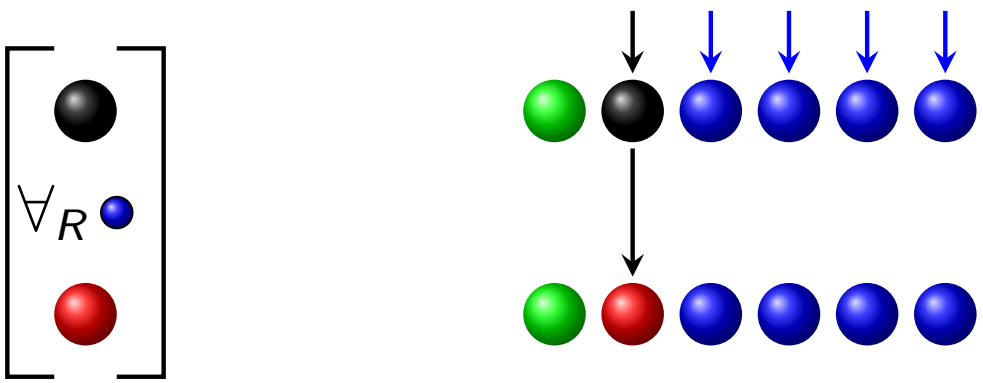


Universal Global Transitions

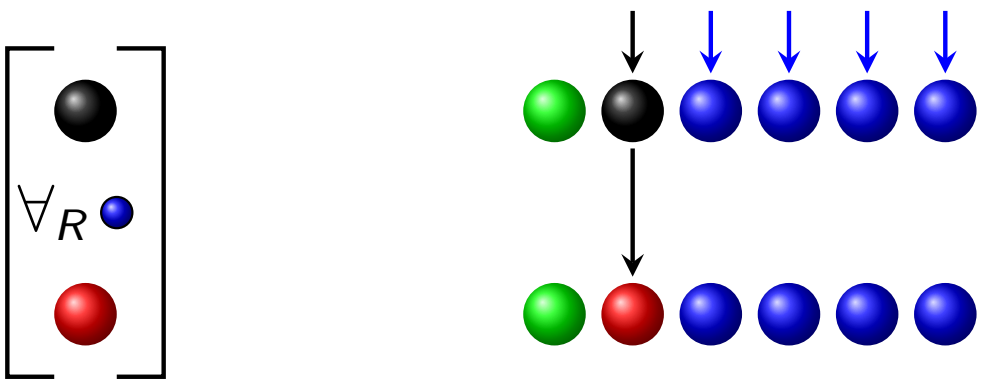


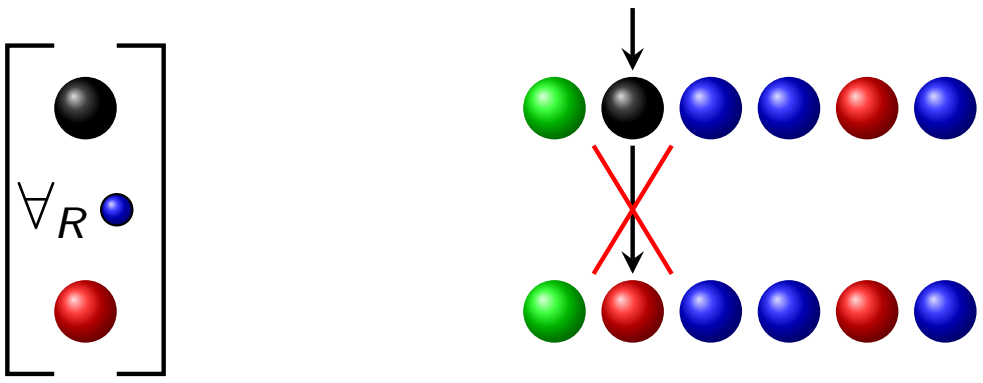


Universal Global Transitions

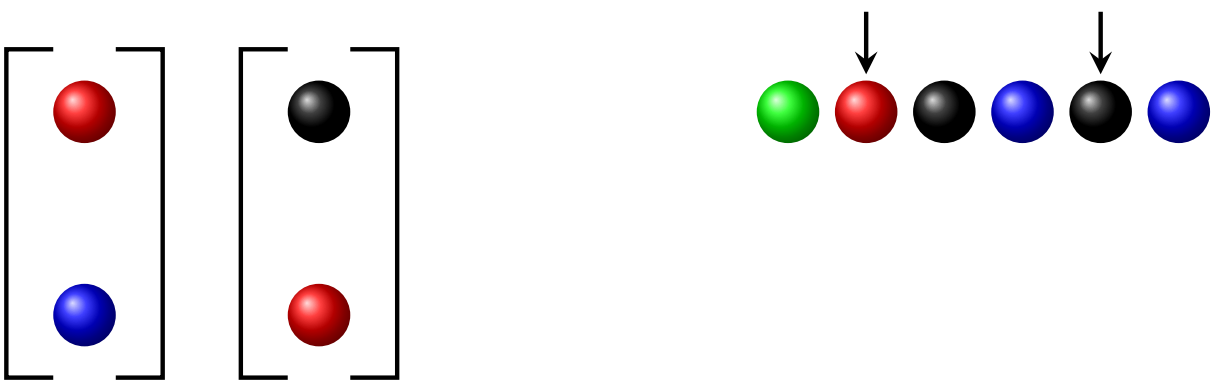


Universal Global Transitions

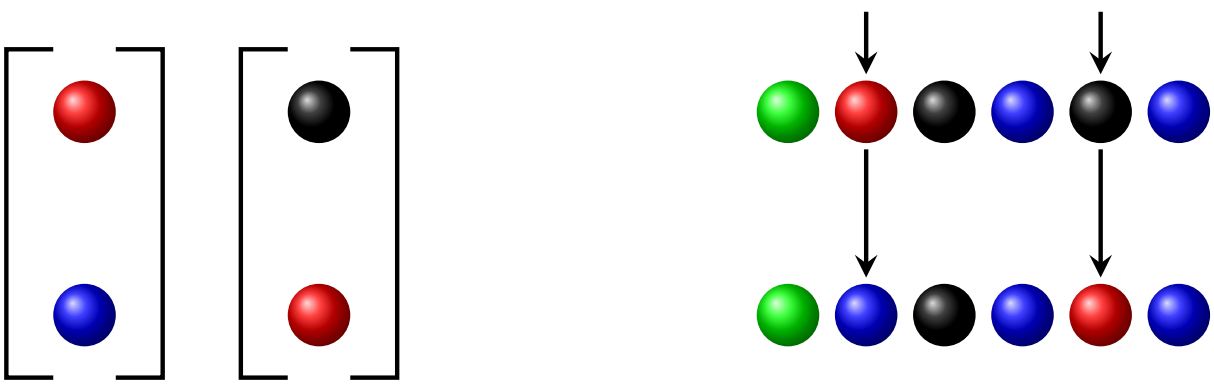




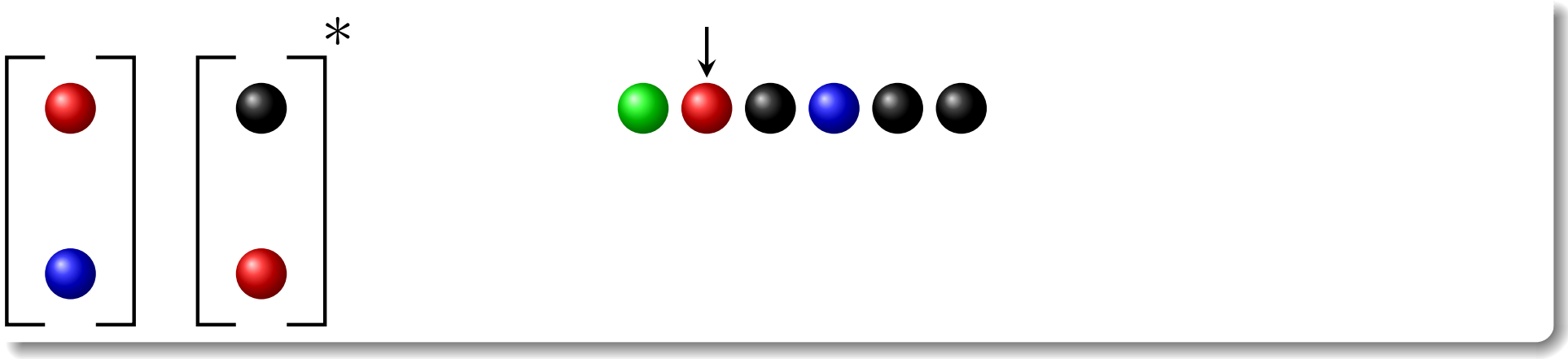
Binary Transitions



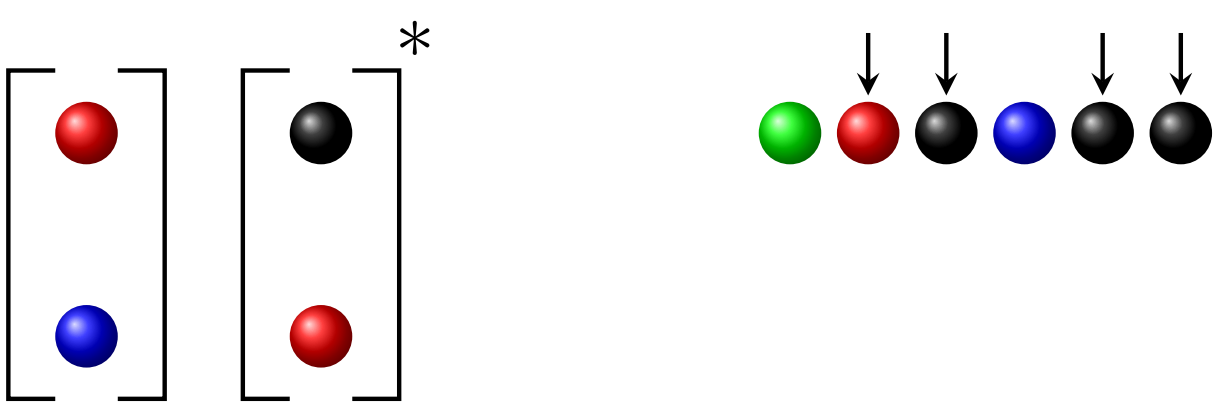
Binary Transitions



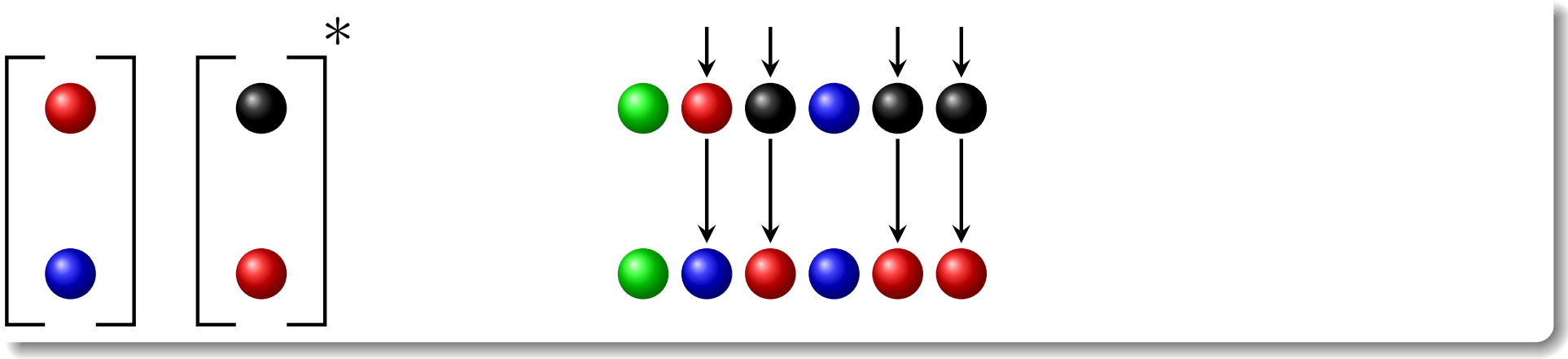
Broadcast Transitions



Broadcast Transitions

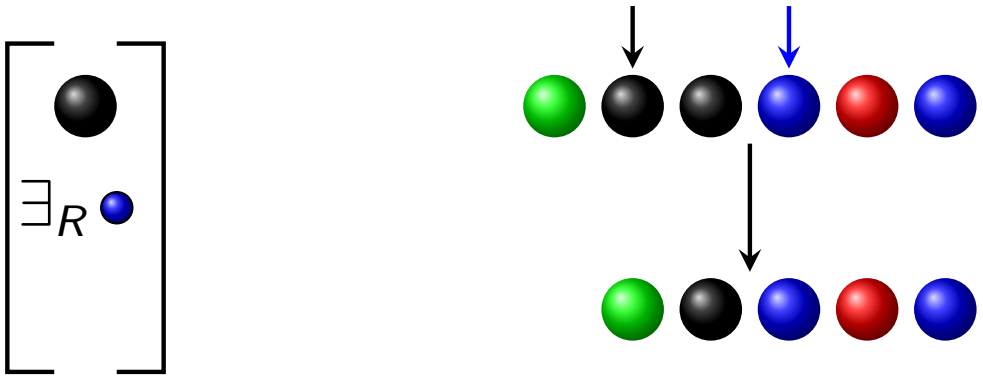


Broadcast Transitions

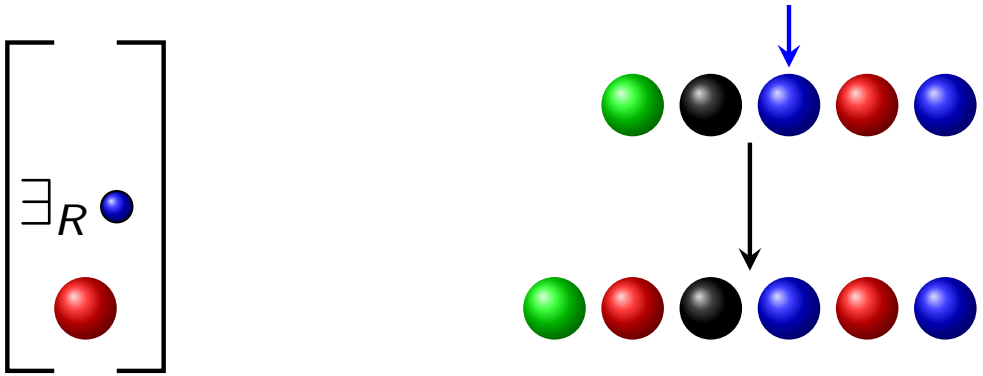


Dynamic Process Deletion and Creation

Process Deletion



Process Creation



German Protocol

Instance

Q : q_{inv}, q_{sh}, q_{exc}; q_{all} is any state in Q.

X : $curClt, sLst, iLst \in \mathcal{B}$, $ch_1 \in \{\epsilon, reqSh, reqExc\}$, $ch_2 \in \{\epsilon, grantSh, grantExc, inval\}$, $ch_3 \in \{\epsilon, invAck\}$

S : $excGran \in \mathcal{B}$, $curCm \in \{\epsilon, reqSh, reqExc\}$

T:

$$h_0 : \left[\begin{array}{c} q_{all} \\ curCm = reqSh, \neg excGran, ch_2 = \epsilon, curClt \\ \longrightarrow curCm = \epsilon, sLst, ch_2 = grantSh \\ q_{all} \end{array}\right]$$

$$h_1 : \left[\begin{array}{c} q_{all} \\ curCm = reqExc, ch_2 = \epsilon, \forall_{LR} \neg sLst, \neg sLst \\ \longrightarrow curCm = \epsilon, sLst, excGran, ch_2 = grantExc \\ q_{all} \end{array}\right] \longleftarrow \text{Universal}$$

$$h_2 : \left[\begin{array}{c} q_{all} \\ curCm = \epsilon, ch_1 \neq \epsilon \\ \longrightarrow curCm = ch_1, ch_1 = \epsilon, iLst = sLst, curClt \\ q_{all} \end{array}\right] \left[\begin{array}{c} q_{all} \\ \mathbf{tt} \\ \longrightarrow iLst = sLst, \neg curClt \\ q_{all} \end{array}\right]^* \longleftarrow \text{Broadcast}$$

$$h_3 : \left[\begin{array}{c} q_{all} \\ curCm = reqSh, excGran, iLst, ch_2 = \epsilon \\ \longrightarrow \neg iLst, ch_2 = inval \\ q_{all} \end{array}\right]$$

$$h_4 : \left[\begin{array}{c} q_{all} \\ curCm = reqExc, iLst, ch_2 = \epsilon \\ \longrightarrow \neg iLst, ch_2 = inval \\ q_{all} \end{array}\right] \quad h_5 : \left[\begin{array}{c} q_{all} \\ curCm \neq \epsilon, ch_3 = invAck \\ \longrightarrow \neg sLst, \neg excGran, ch_3 = \epsilon \\ q_{all} \end{array}\right]$$

$$p_1 : \begin{bmatrix} q_{inv} \\ ch_1 = \epsilon \\ \longrightarrow ch_1 = reqSh \\ q_{inv} \end{bmatrix} \quad p_2 : \begin{bmatrix} q_{inv} \\ ch_1 = \epsilon \\ \longrightarrow ch_1 = reqExc \\ q_{inv} \end{bmatrix} \quad p_3 : \begin{bmatrix} q_{sh} \\ ch_1 = \epsilon \\ \longrightarrow ch_1 = reqExc \\ q_{sh} \end{bmatrix}$$

$$p_4 : \begin{bmatrix} q_{all} \\ ch_2 = inval, ch_3 = \epsilon \\ \longrightarrow ch_2 = \epsilon, ch_3 = invAck \\ q_{inv} \end{bmatrix} \quad p_5 : \begin{bmatrix} q_{all} \\ ch_2 = grantSh \\ \longrightarrow ch_2 = \epsilon \\ q_{sh} \end{bmatrix} \quad p_6 : \begin{bmatrix} q_{all} \\ ch_2 = grantExc \\ \longrightarrow ch_2 = \epsilon \\ q_{exc} \end{bmatrix}$$

Initial State of Shared Vars: $excGran \mapsto \mathbf{ff}, curCm \mapsto \epsilon$
Initial Process State: u_{init}: $q_{inv}, (ch_1, ch_2, ch_3, curClt, sLst, iLst) \mapsto (\epsilon, \epsilon, \epsilon, \mathbf{ff}, \mathbf{ff}, \mathbf{ff})$
Final Constraints: Φ_F: $q_{exc}\,q_{exc}$, $q_{sh}\,q_{exc}$, $q_{exc}\,q_{sh}$

Futurebus+ Protocol

Instance

$Q : q_{pendR}, q_{shU}, q_{pendSU}, q_{pendE}, q_{exclU}, q_{exclM}, q_{pendEW}, q_{pendW}, q_{inv}$

T:

$$t_1 : \begin{bmatrix} q_{inv} \\ \forall_{LR} \neg q_{pendW} \rightarrow \{\} \\ q_{pendR} \end{bmatrix} \begin{bmatrix} q_{exclM} \\ \mathbf{tt} \rightarrow \{\} \\ q_{pendE} \end{bmatrix}^* \begin{bmatrix} q_{shU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{pendSU} \end{bmatrix}^* \begin{bmatrix} q_{exclU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{pendSU} \end{bmatrix}^*$$

$$t_2 : \begin{bmatrix} q_{pendE} \\ \mathbf{tt} \rightarrow \{\} \\ q_{shU} \end{bmatrix} \begin{bmatrix} q_{pendR} \\ \mathbf{tt} \rightarrow \{\} \\ q_{shU} \end{bmatrix}^* \qquad t_3 : \begin{bmatrix} q_{pendSU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{shU} \end{bmatrix} \begin{bmatrix} q_{pendR} \\ \mathbf{tt} \rightarrow \{\} \\ q_{shU} \end{bmatrix}^* \begin{bmatrix} q_{pendSU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{shU} \end{bmatrix}^*$$

$$t_4 : \begin{bmatrix} q_{pendR} \\ (\forall_{LR} \neg (q_{pendSU} \vee q_{pendE}) \wedge (\exists_{LR} q_{pendR}) \rightarrow \{\} \\ q_{shU} \end{bmatrix} \begin{bmatrix} q_{pendR} \\ \rightarrow \{\} \\ q_{shU} \end{bmatrix}^* \longleftarrow \text{Universal} + \text{Broadcast}$$

$$t_5 : \begin{bmatrix} q_{pendR} \\ \forall_{LR} \neg (q_{pendSU} \vee q_{pendE} \vee q_{pendR}) \rightarrow \{\} \\ q_{exclU} \end{bmatrix}$$

$$t_6 : \begin{bmatrix} q_{inv} \\ \forall_{LR} \neg q_{pendW} \rightarrow \{\} \\ q_{pendW} \end{bmatrix} \begin{bmatrix} q_{pendE} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix}^* \begin{bmatrix} q_{shU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix}^* \begin{bmatrix} q_{exclU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix}^*$$

$$\begin{bmatrix} q_{pendSU} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix}^* \begin{bmatrix} q_{pendR} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix}^* \begin{bmatrix} q_{pendEW} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix}^* \begin{bmatrix} q_{exclM} \\ \mathbf{tt} \rightarrow \{\} \\ q_{pendEW} \end{bmatrix}^*$$

$$t_7 : \begin{bmatrix} q_{pendEW} \\ \mathbf{tt} \rightarrow \{\} \\ q_{inv} \end{bmatrix} \begin{bmatrix} q_{pendW} \\ \mathbf{tt} \rightarrow \{\} \\ q_{exclM} \end{bmatrix}^* \qquad t_8 : \begin{bmatrix} q_{pendW} \\ \forall_{LR} \neg q_{pendEW} \rightarrow \{\} \\ q_{exclM} \end{bmatrix} \begin{bmatrix} q_{pendW} \\ \mathbf{tt} \rightarrow \{\} \\ q_{exclM} \end{bmatrix}^*$$

$$t_9 : \begin{bmatrix} q_{exclU} \\ \mathbf{tt} \longrightarrow \{\} \\ q_{exclM} \end{bmatrix} \quad t_{10} : \begin{bmatrix} q_{exclM} \\ \mathbf{tt} \longrightarrow \{\} \\ q_{exclM} \end{bmatrix} \quad t_{11} : \begin{bmatrix} q_{shU} \\ \mathbf{tt} \longrightarrow \{\} \\ q_{exclM} \end{bmatrix} \begin{bmatrix} q_{shU} \\ \longrightarrow \{\} \\ q_{inv} \end{bmatrix}^*$$

Initial Process State: u_{init}: q_{inv}

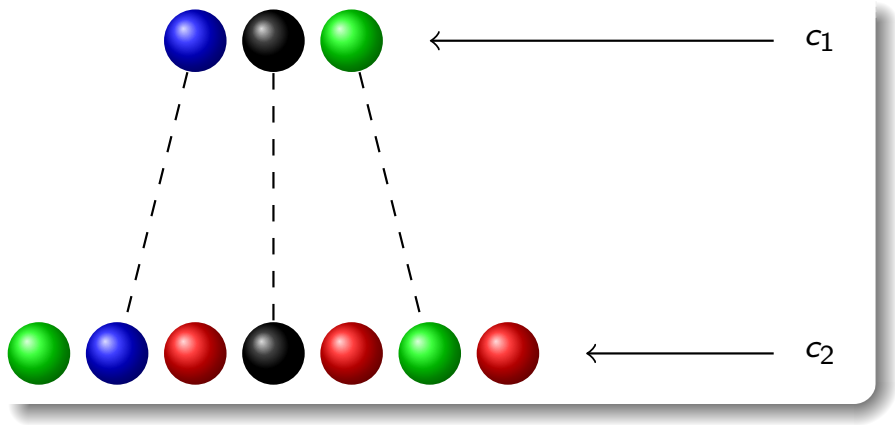
Final Constraints: Φ_F: $q_{shU}(q_{exclM} \vee q_{exclU}), (q_{exclM} \vee q_{exclU})q_{shU},$
$q_{exclU}(q_{exclU} \vee q_{exclM}), q_{exclM}q_{exclU},$
$q_{pendW}(q_{pendW} \vee q_{pendR}), q_{pendR}q_{pendW}$

Ordering on Configurations

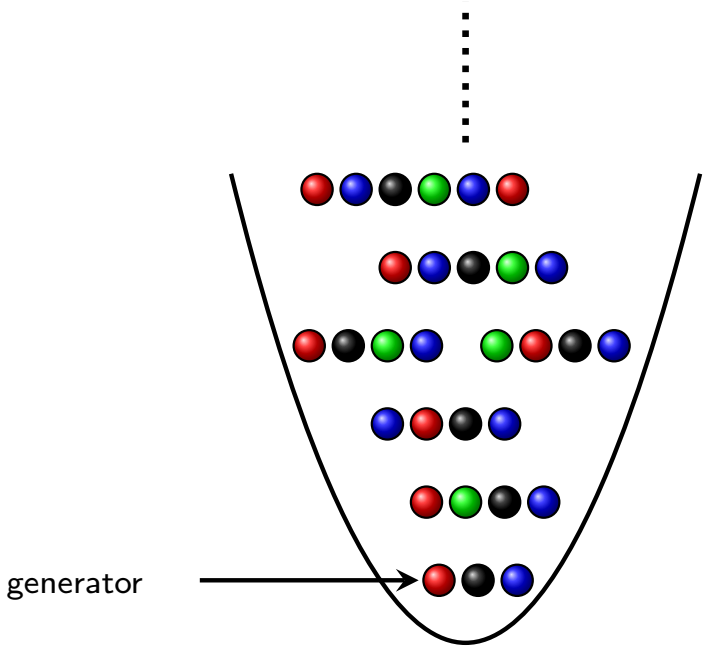
c_1

$\sqsubset$

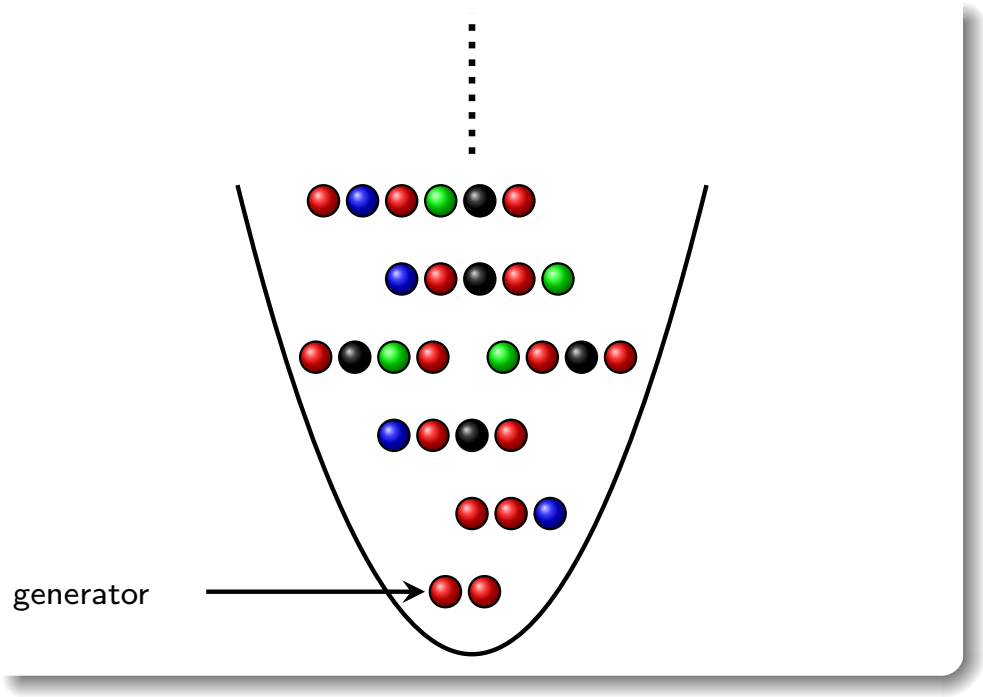
c_2



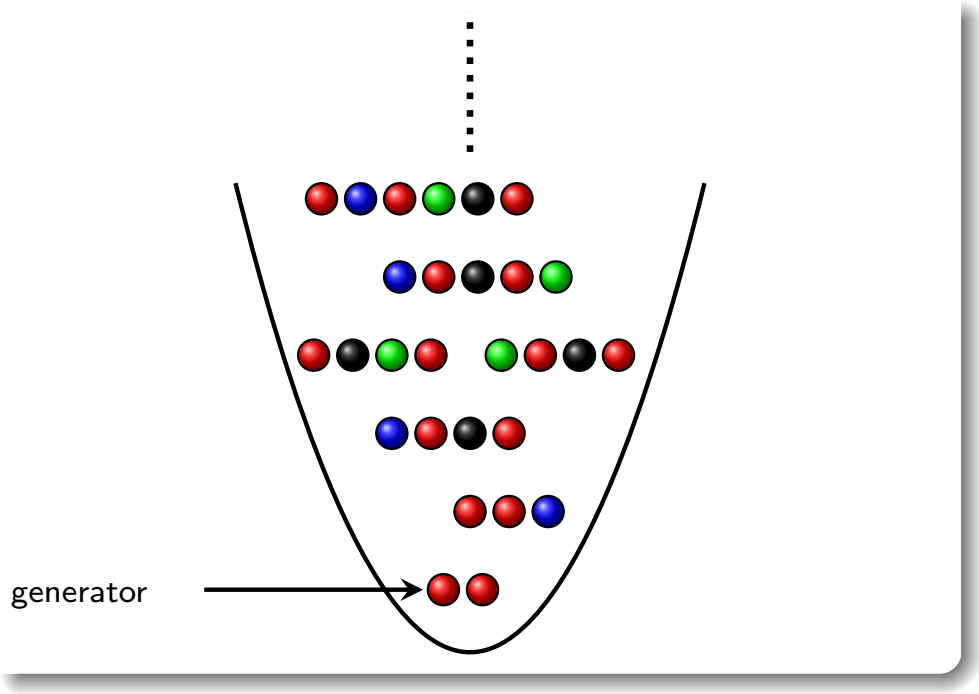
Upward-Closed Sets (UC)



Upward-Closed Sets (UC)



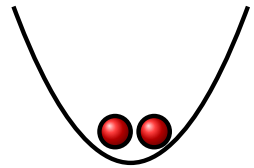
Upward-Closed Sets (UC)



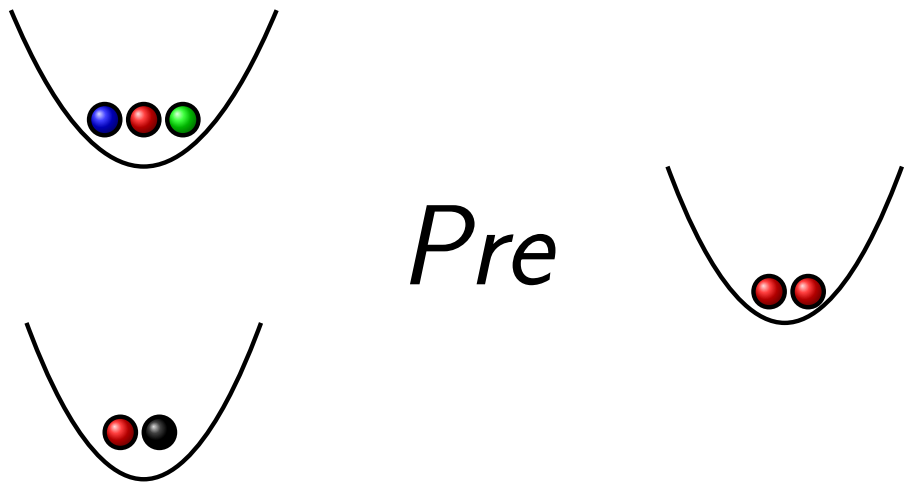
Why UC?

- Bad sets of states are UC
  - safety properties = reachability of UC
- Uniquely characterized by generator
  - simple representation = finite word

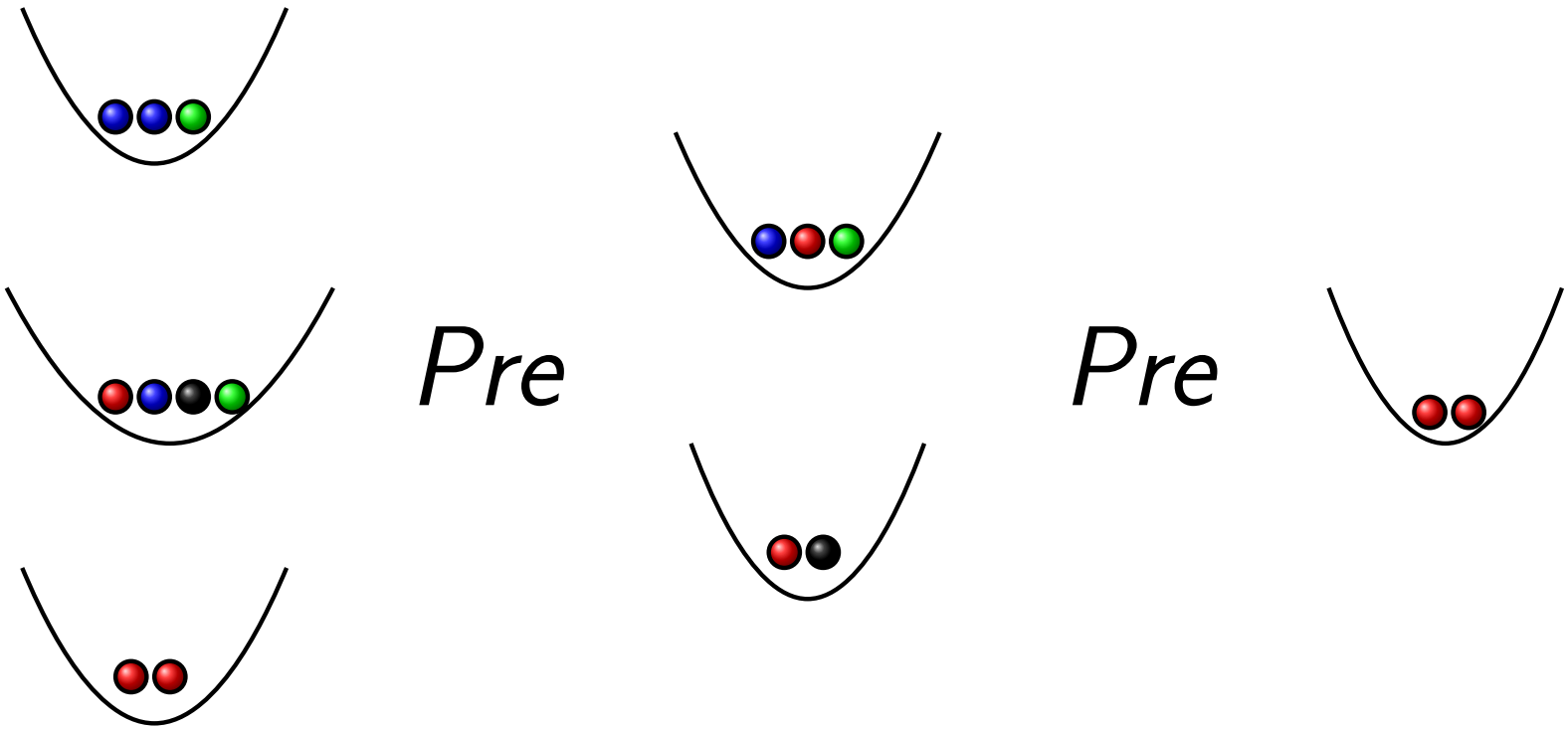
Backward Reachability Analysis (on UC)



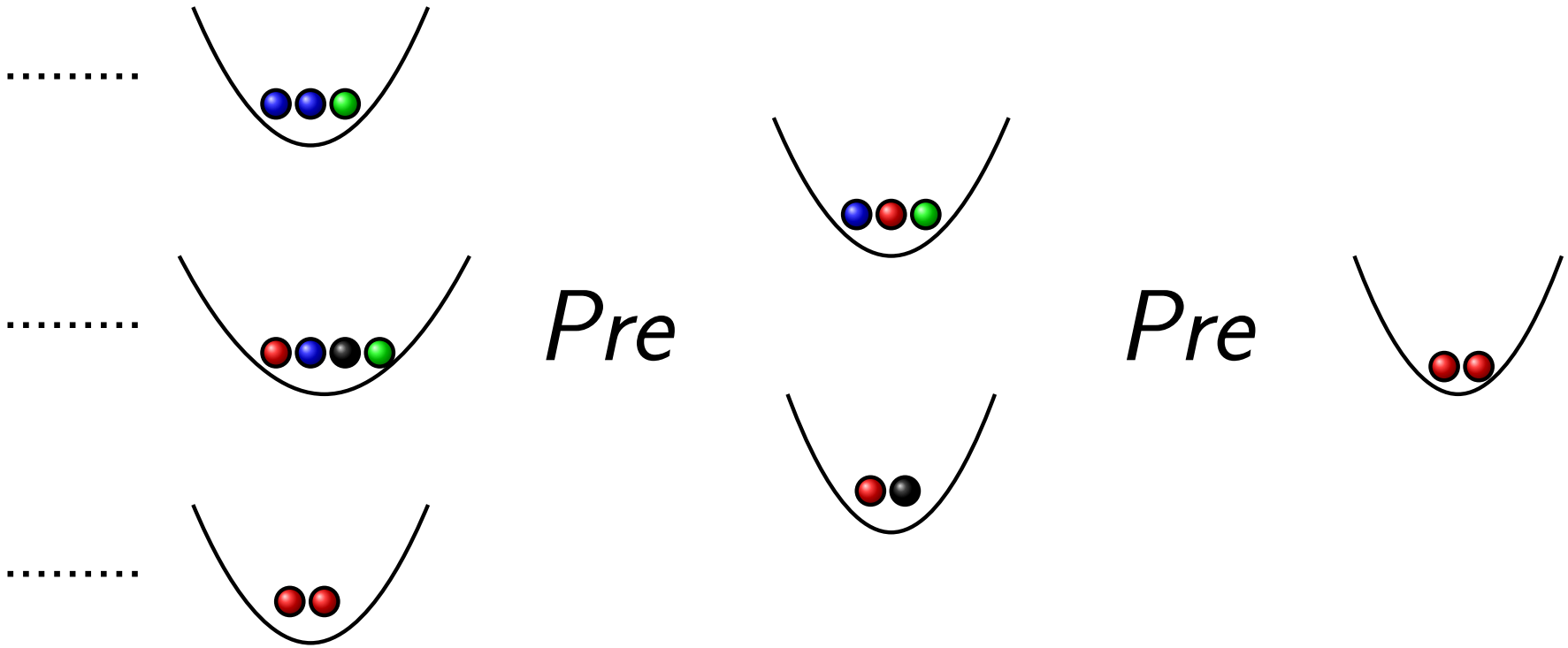
Backward Reachability Analysis (on UC)



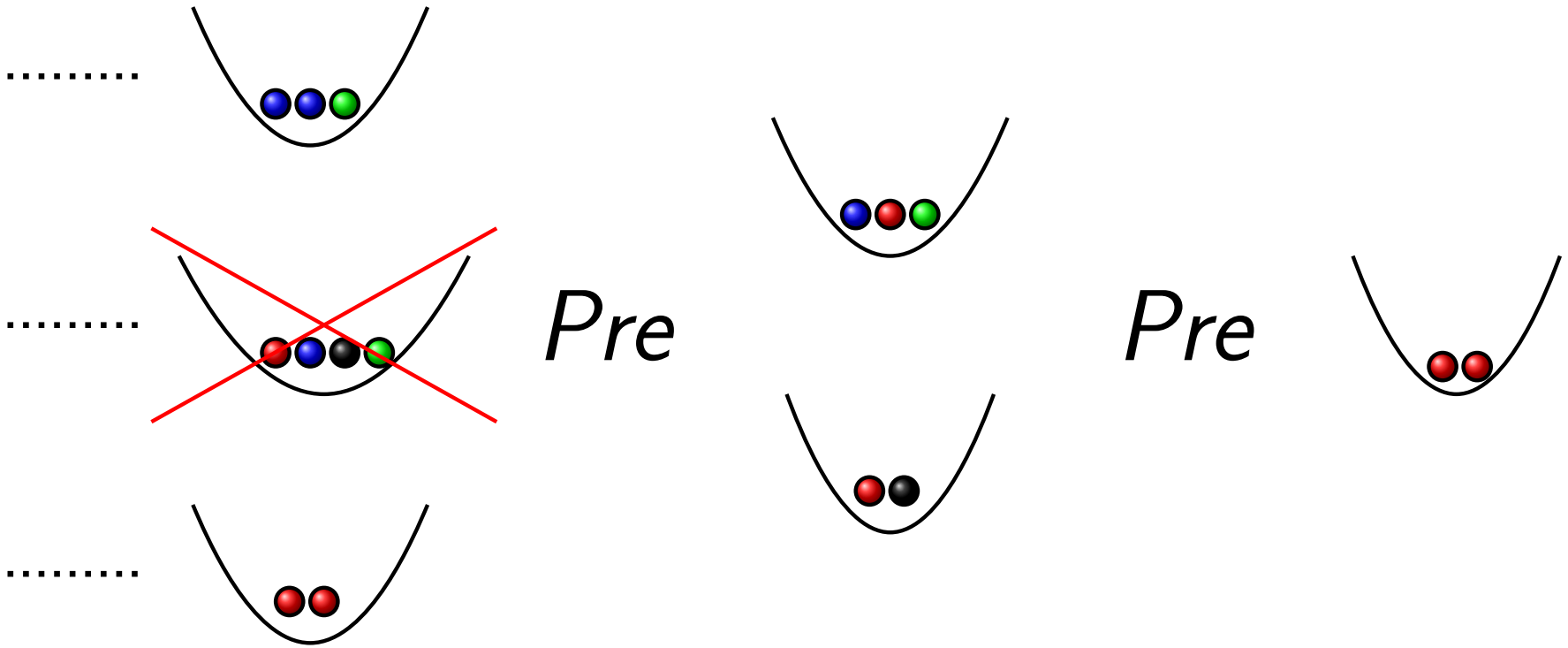
Backward Reachability Analysis (on UC)



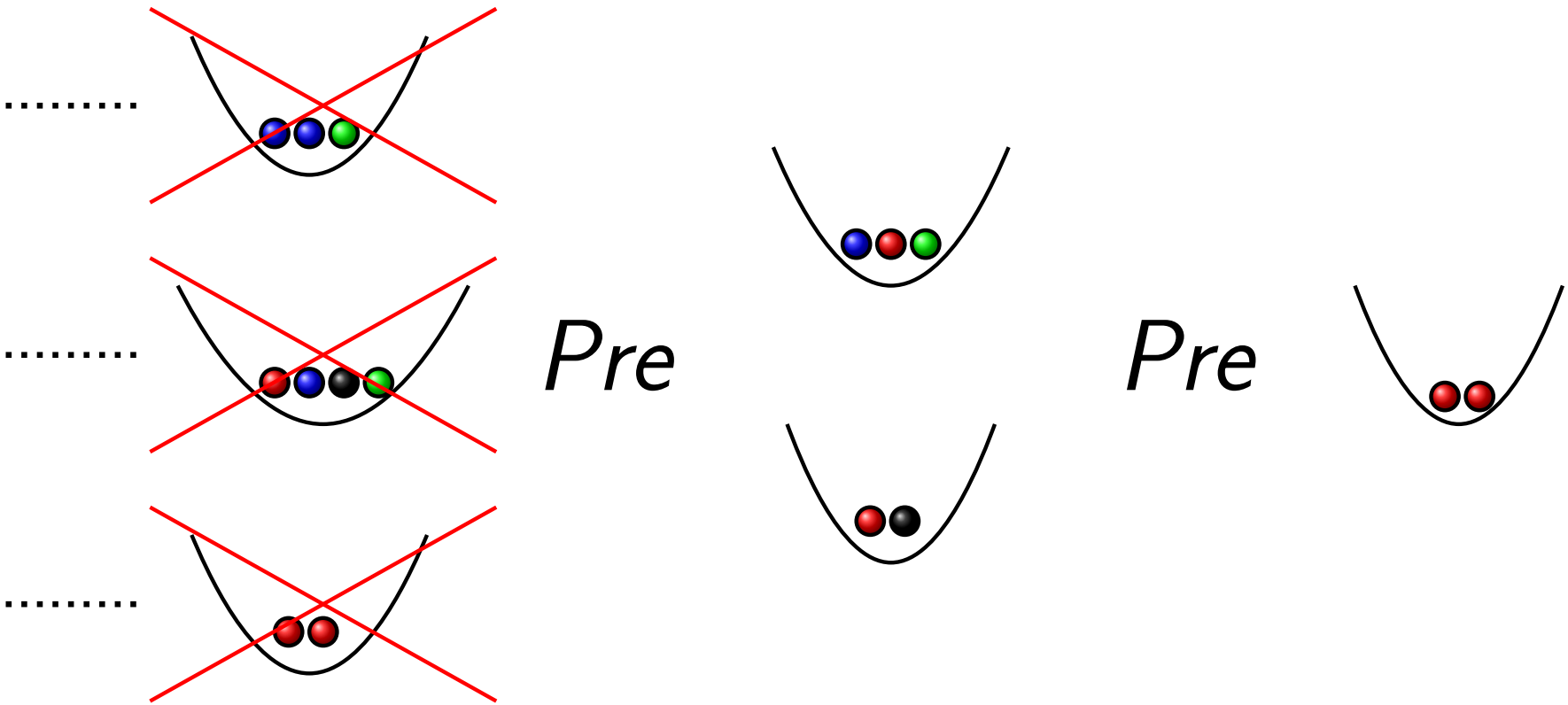
Backward Reachability Analysis (on UC)



Backward Reachability Analysis (on UC)

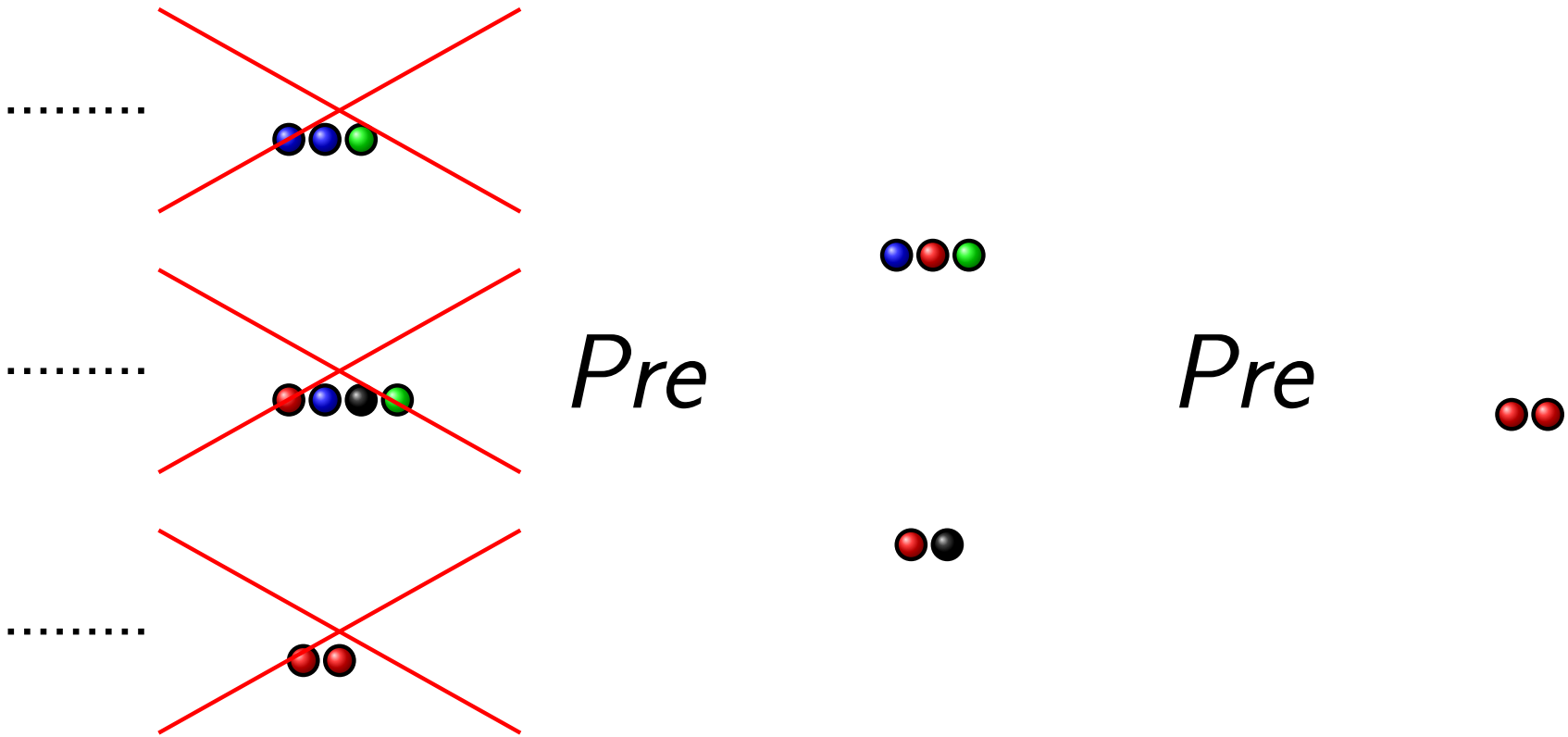


Backward Reachability Analysis (on UC)



Backward Reachability Analysis (on UC)

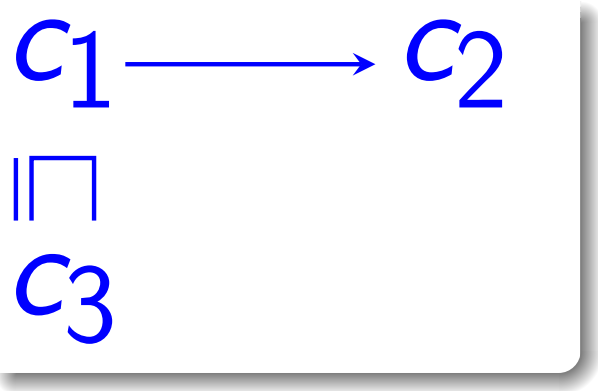
symbolic representation = finite words.



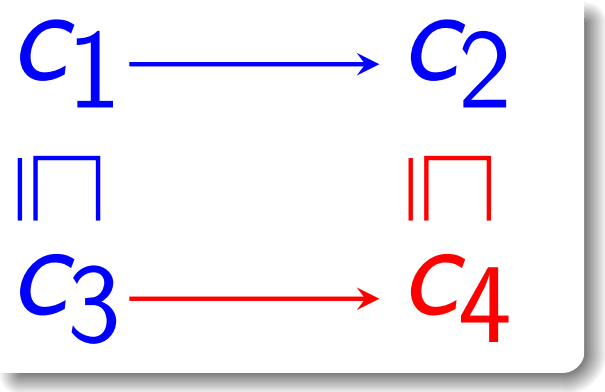
Required:

UC closed under Pre !!

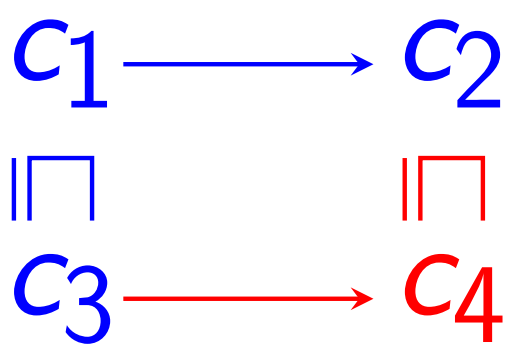
Monotonicity



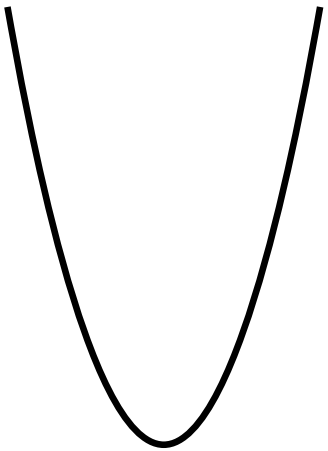
Monotonicity



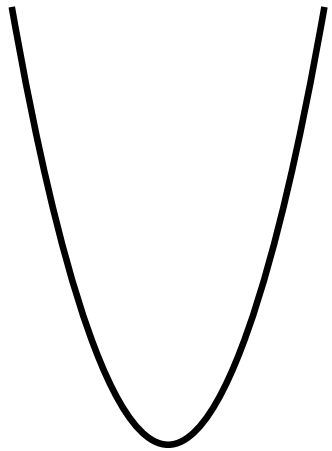
Monotonicity



Monotonicity implies UC is closed under *Pre*

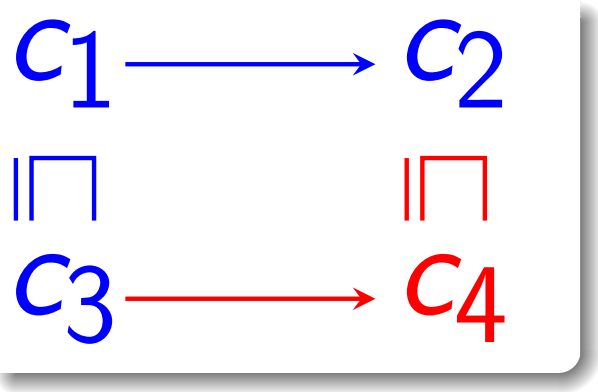


$Pre(U)$:Upward Closed?

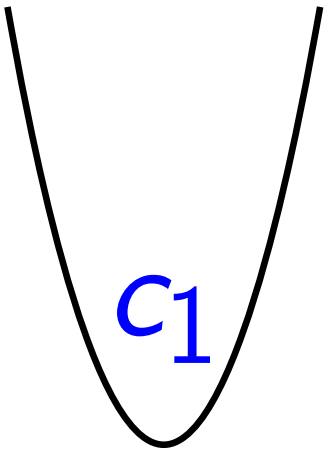


U:Upward Closed

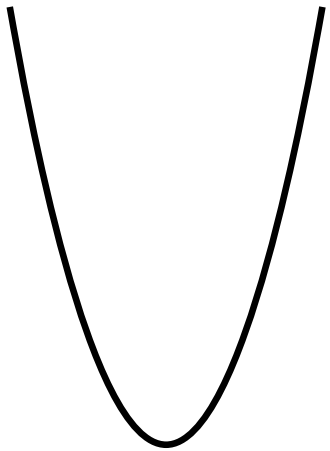
Monotonicity



Monotonicity implies UC is closed under *Pre*

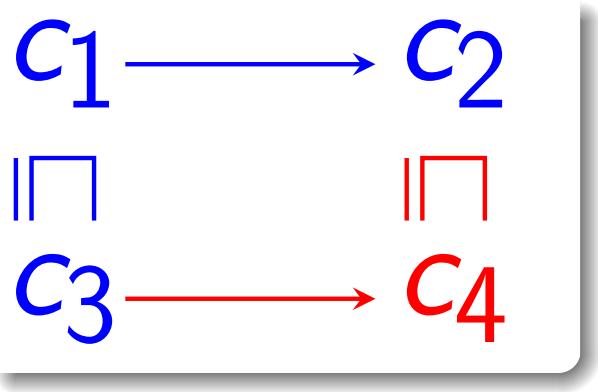


$Pre(U)$:Upward Closed?

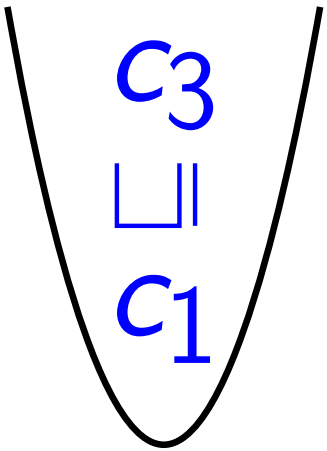


U:Upward Closed

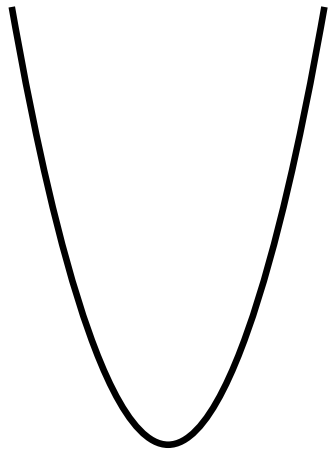
Monotonicity



Monotonicity implies UC is closed under *Pre*

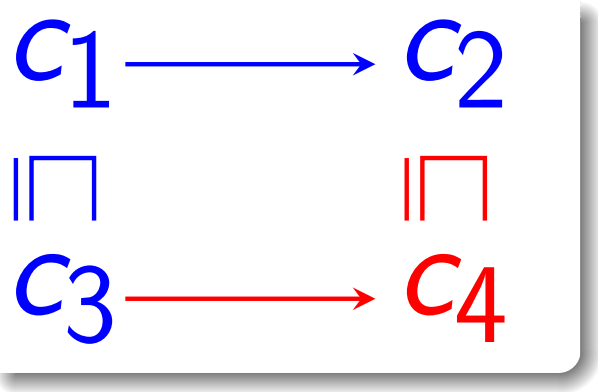


$Pre(U)$:Upward Closed?

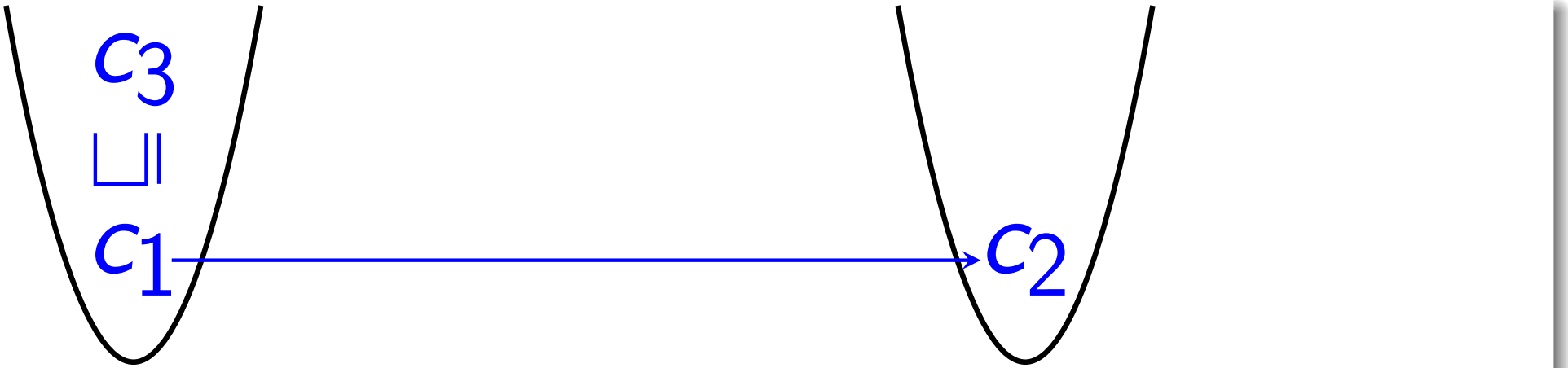


U:Upward Closed

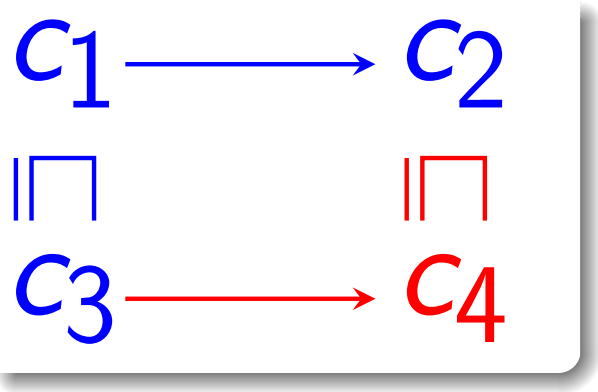
Monotonicity



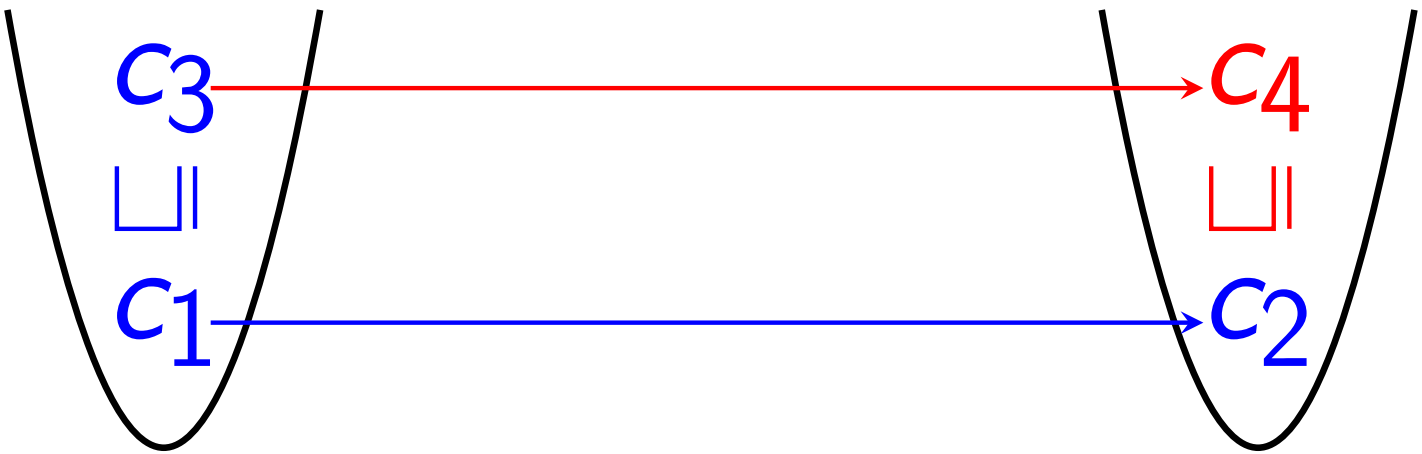
Monotonicity implies UC is closed under *Pre*



Monotonicity



Monotonicity implies UC is closed under *Pre*

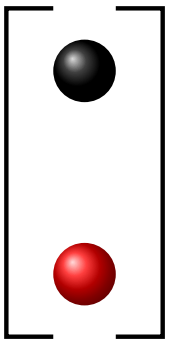


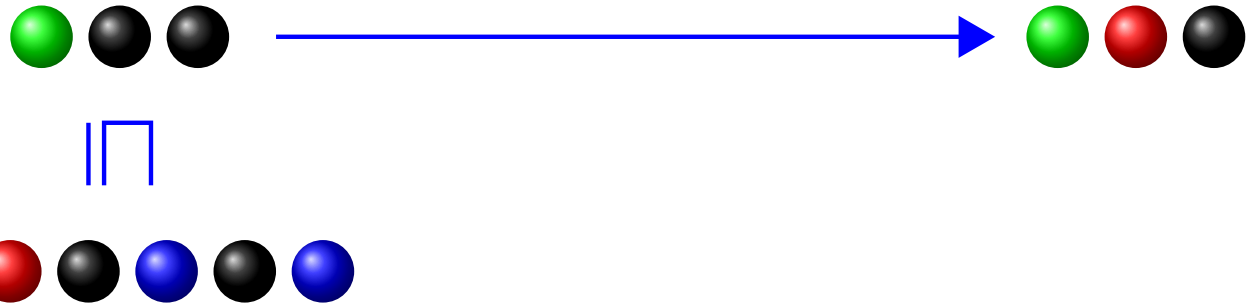
$Pre(U)$:Upward Closed? Yes

U:Upward Closed

Which transitions are monotonic?

Local Transitions

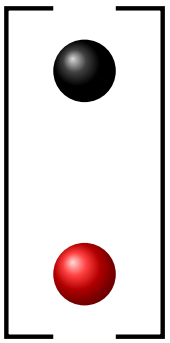


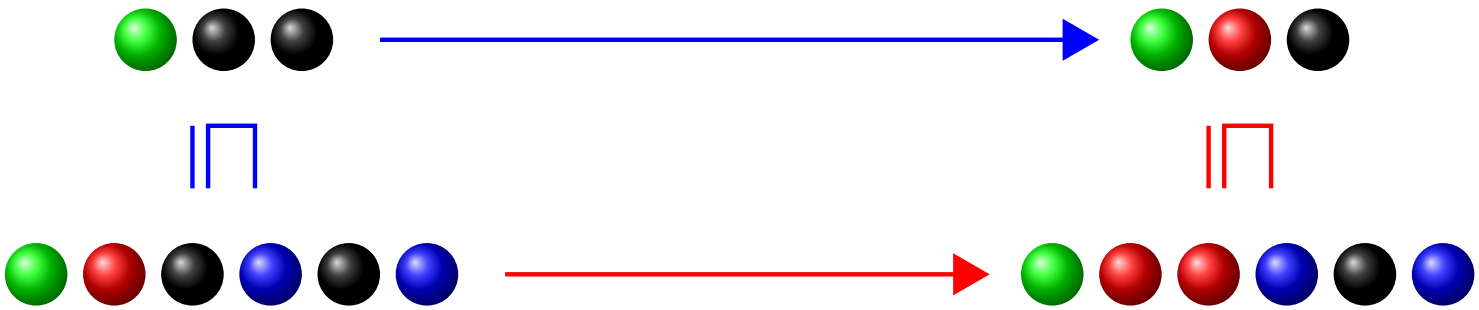


Which transitions are monotonic?

Local Transitions

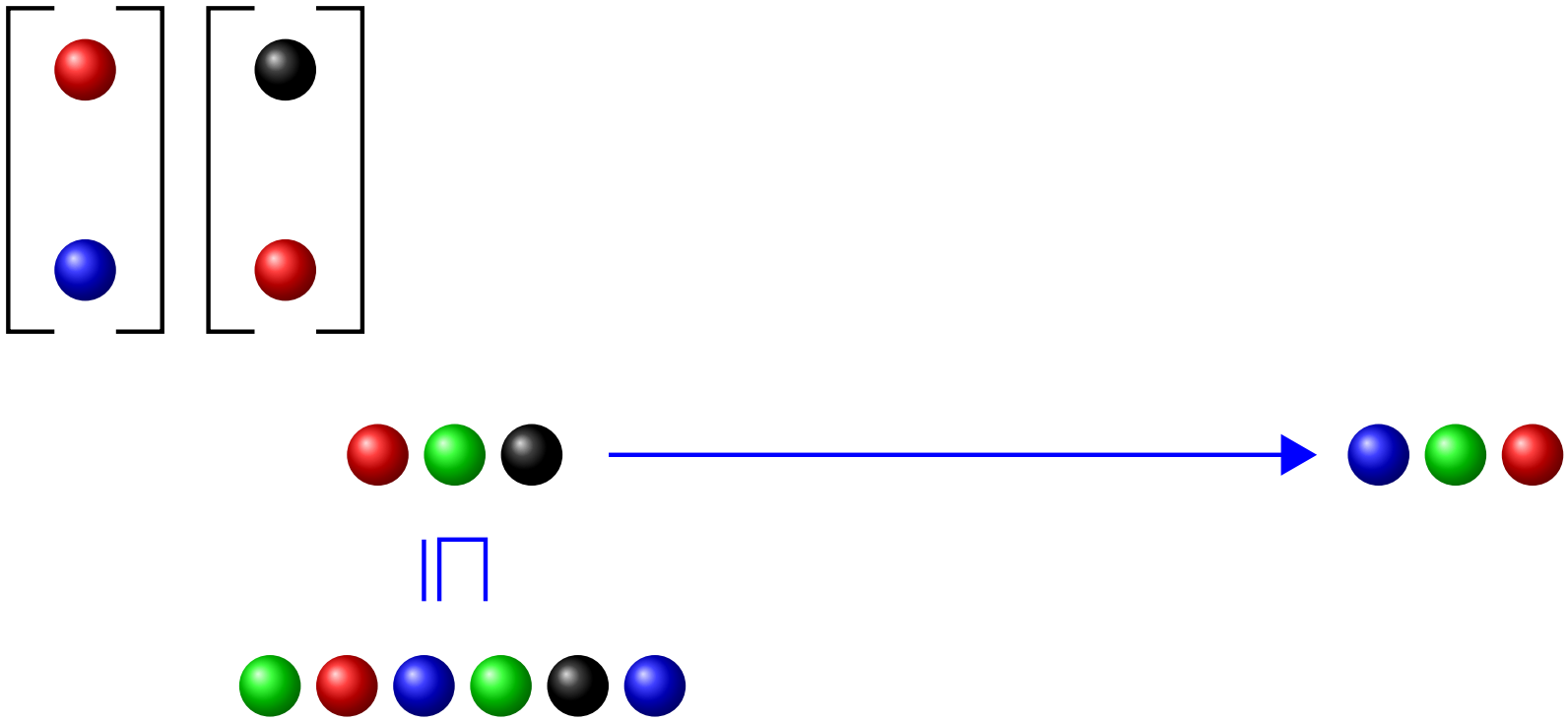
Yes





Which transitions are monotonic?

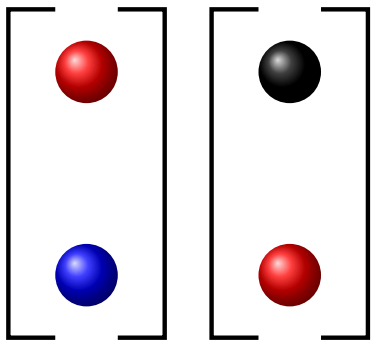
Binary Transitions

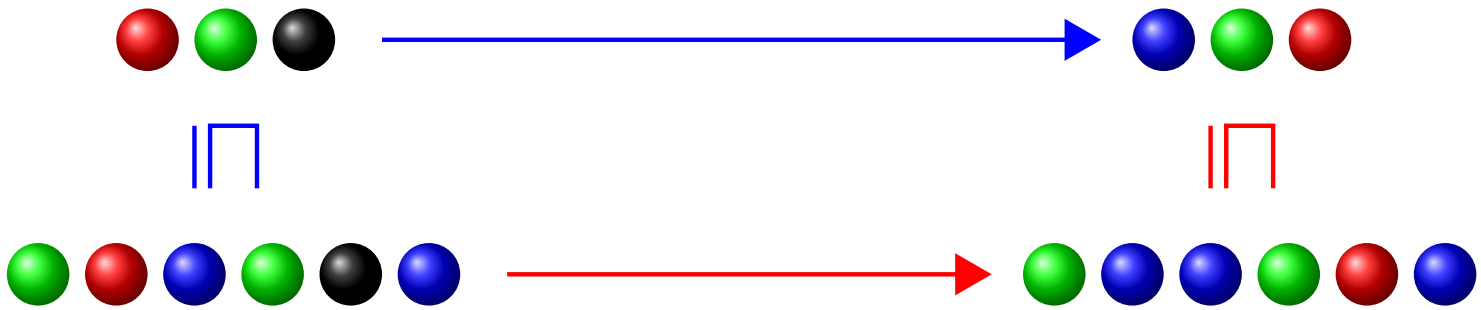


Which transitions are monotonic?

Binary Transitions

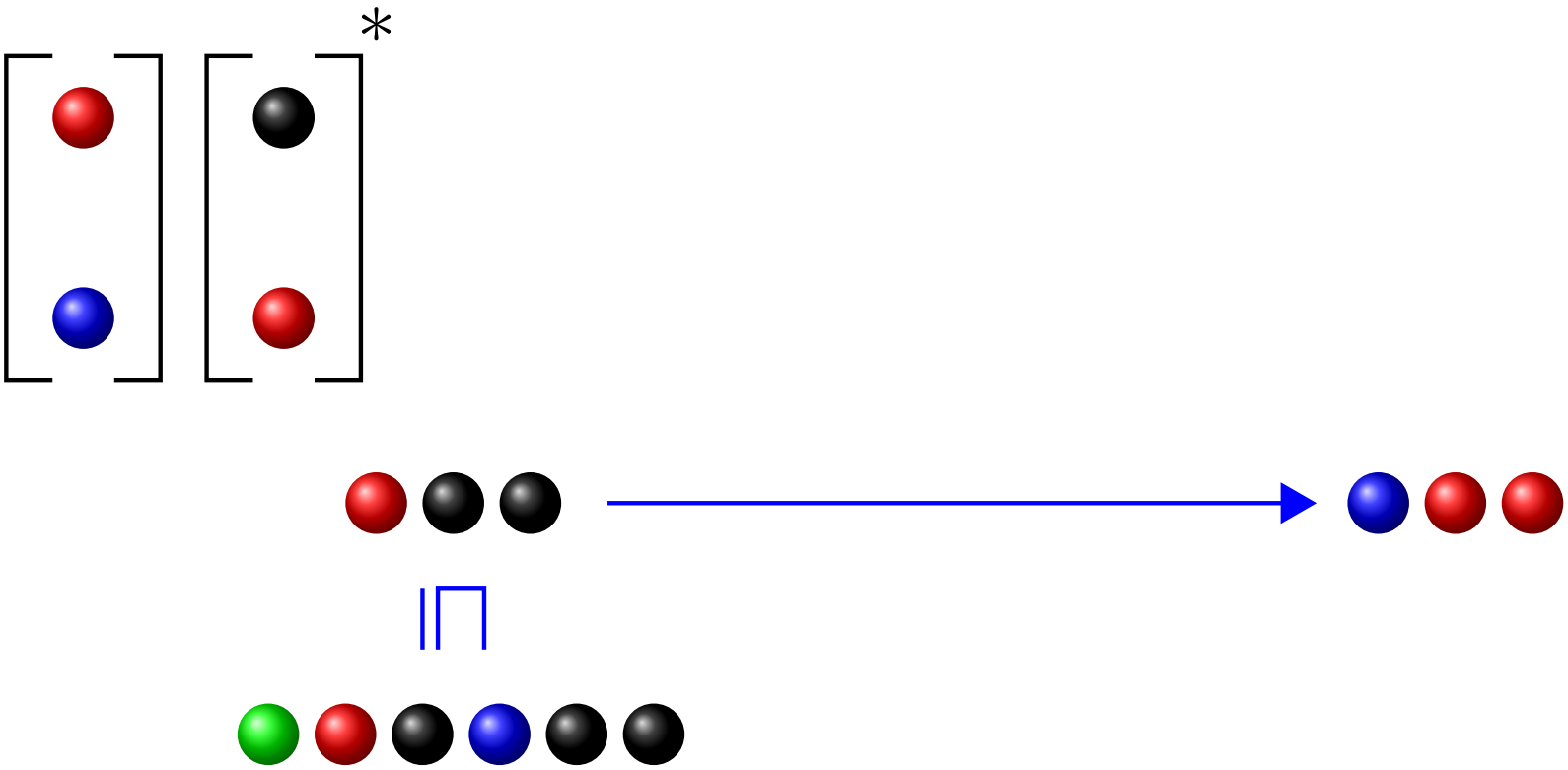
Yes





Which transitions are monotonic?

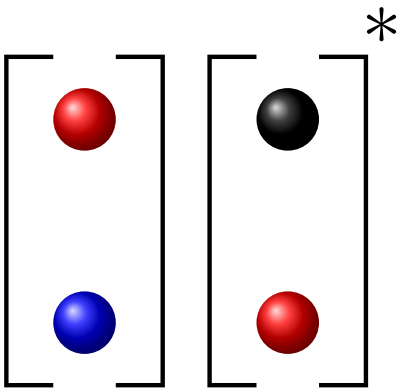
Broadcast Transitions

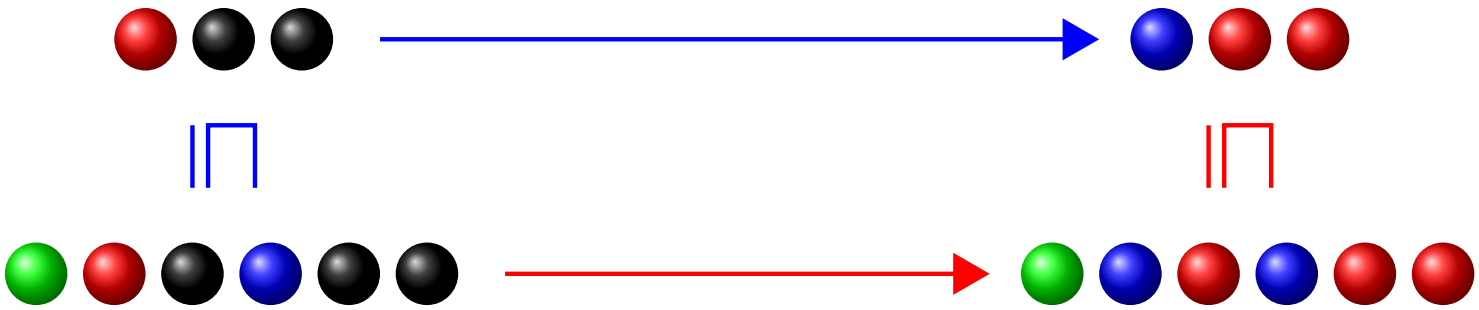


Which transitions are monotonic?

Broadcast Transitions

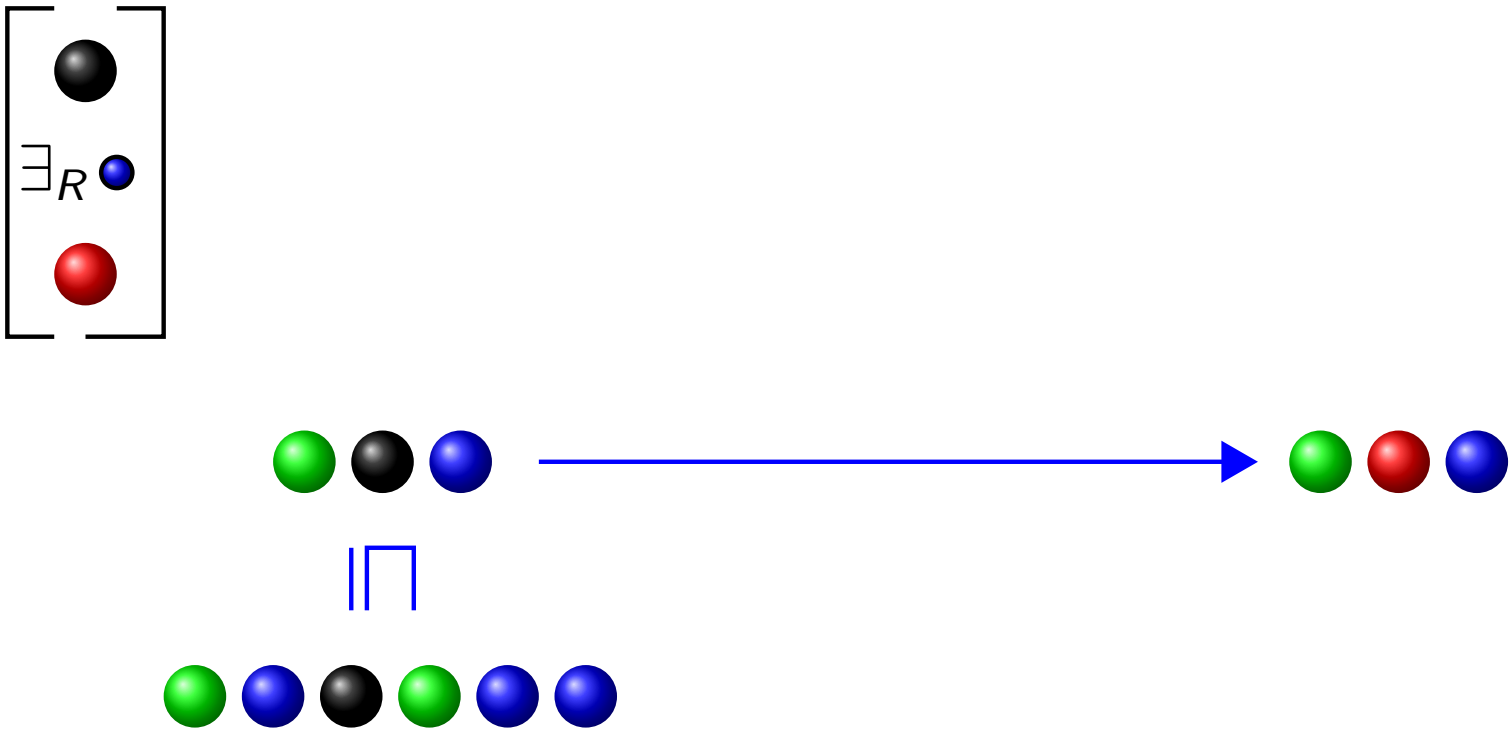
Yes





Which transitions are monotonic?

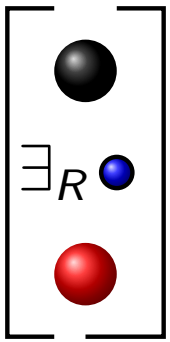
Existential Global Transitions

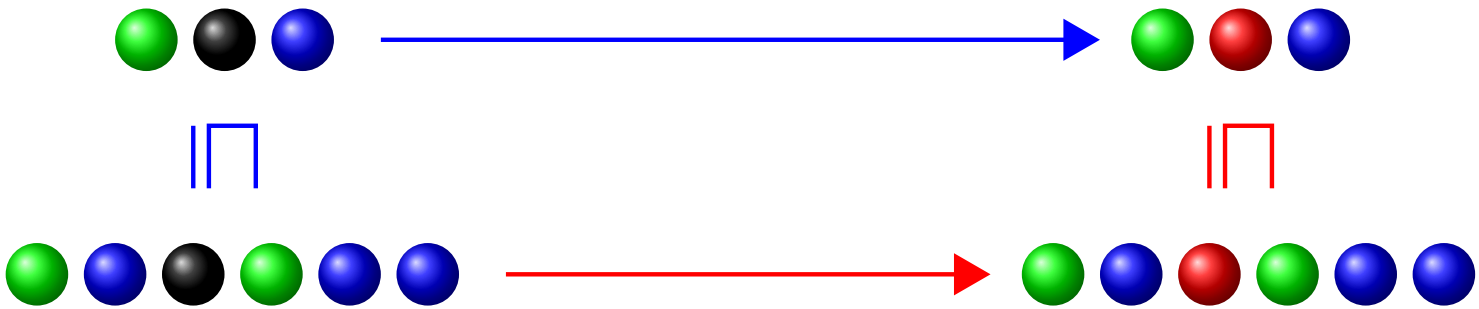


Which transitions are monotonic?

Existential Global Transitions

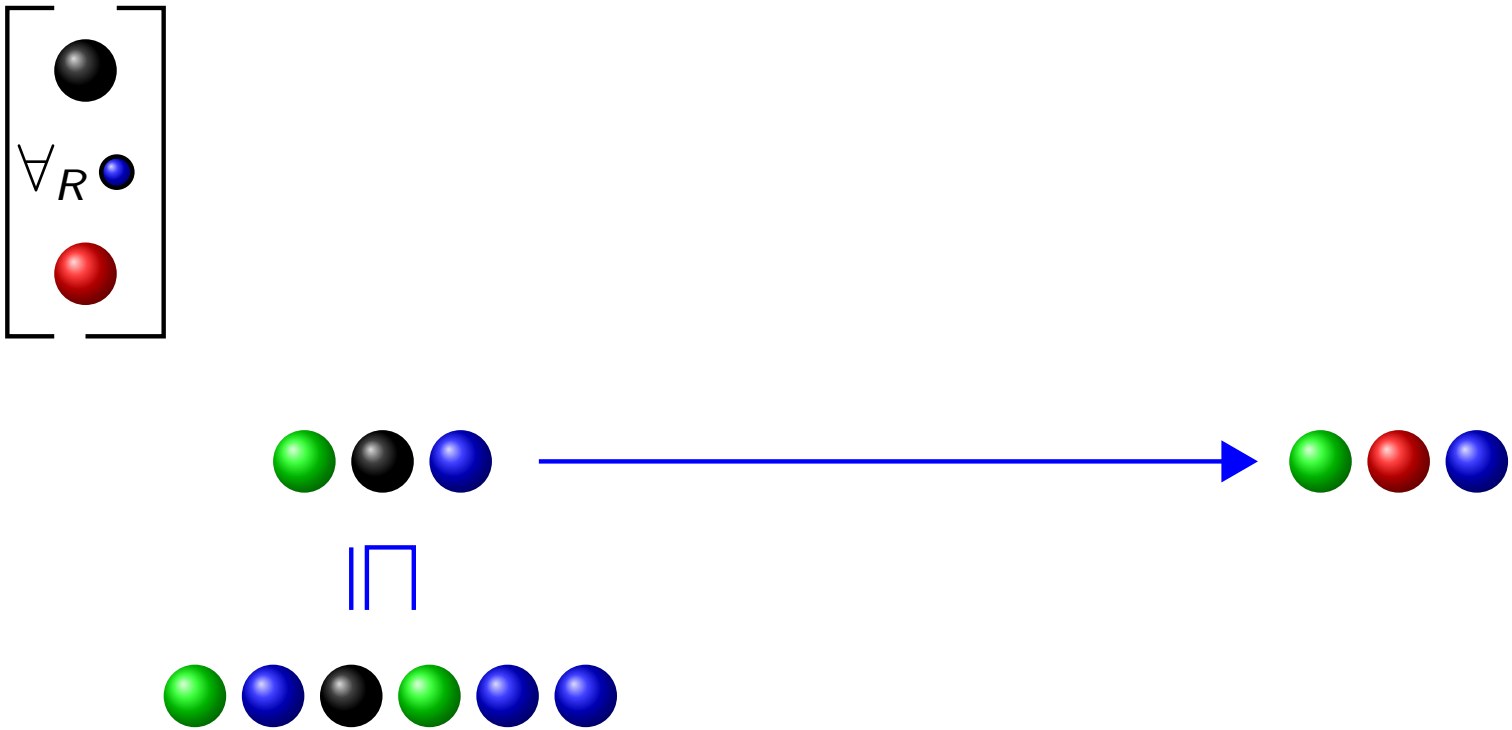
Yes





Which transitions are monotonic?

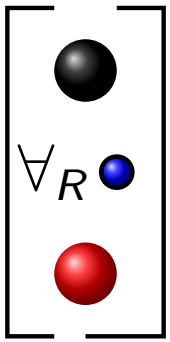
Universal Global Transitions

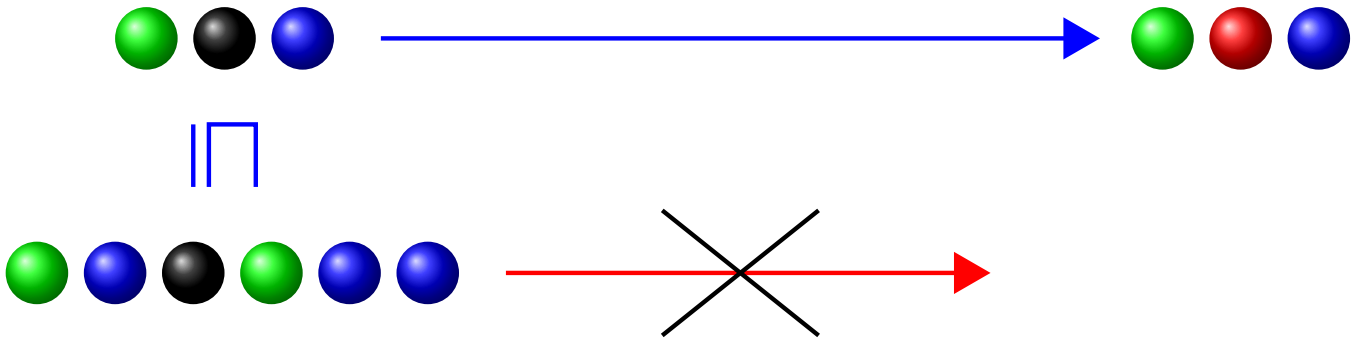


Which transitions are monotonic?

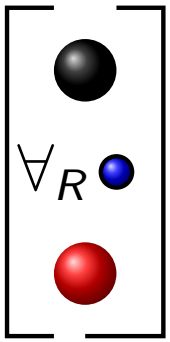
Universal Global Transitions

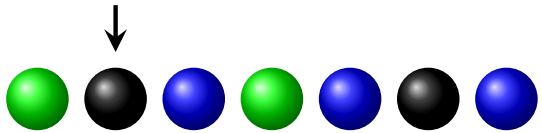
No



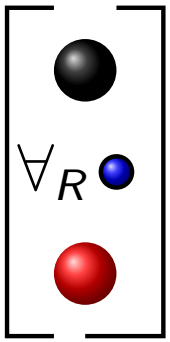


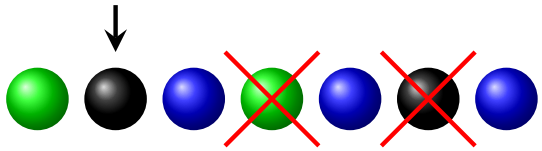
Abstraction (Over-Approximation)



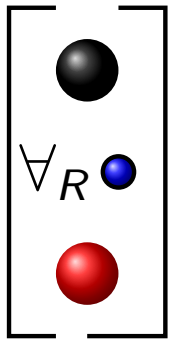


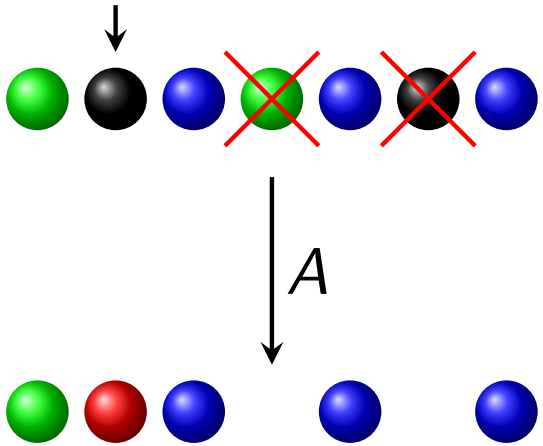
Abstraction (Over-Approximation)



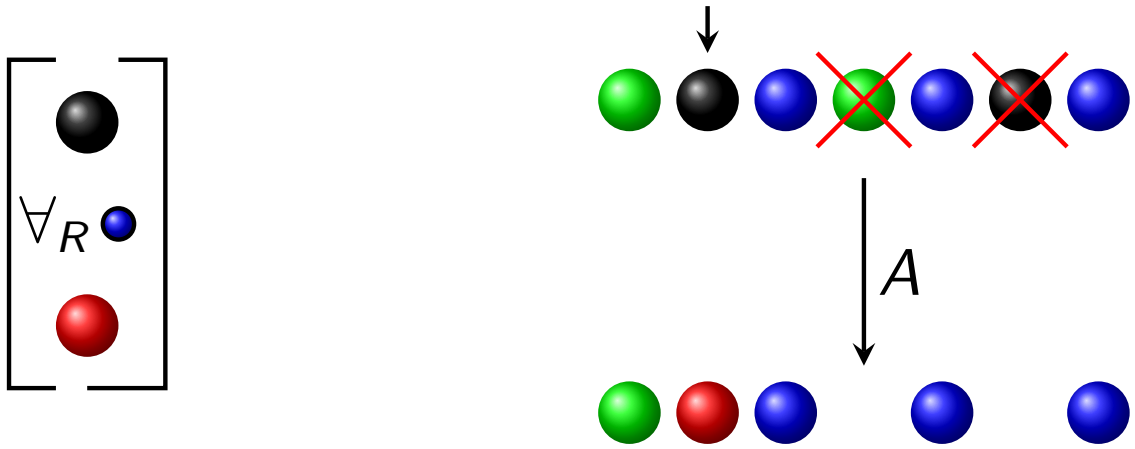


Abstraction (Over-Approximation)

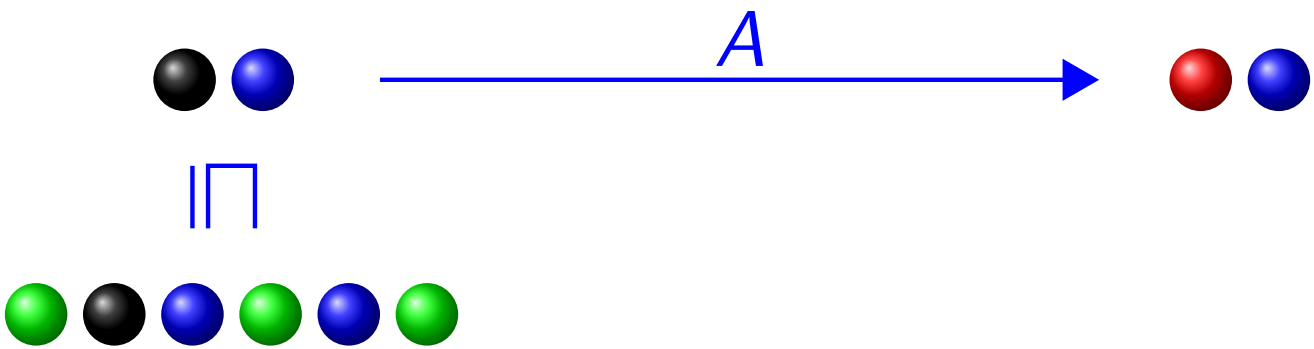




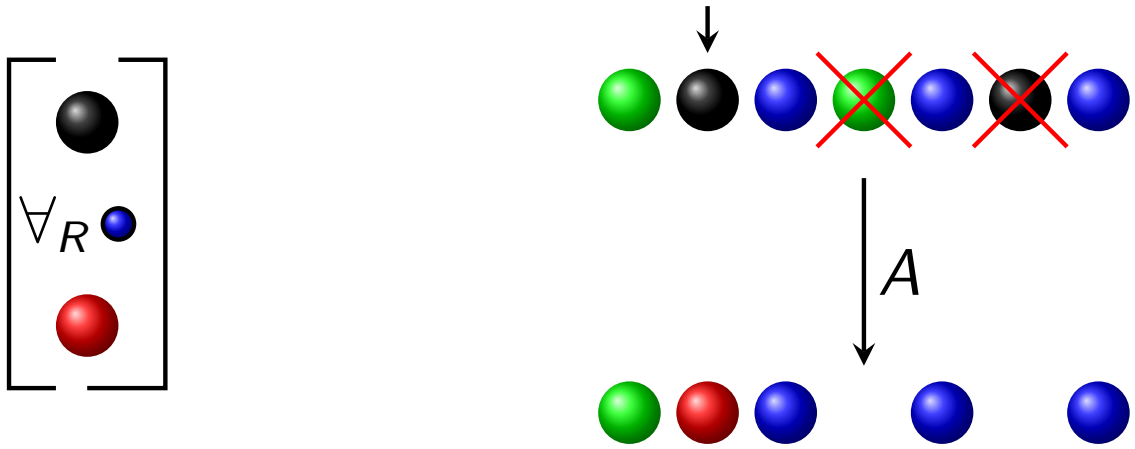
Abstraction (Over-Approximation)



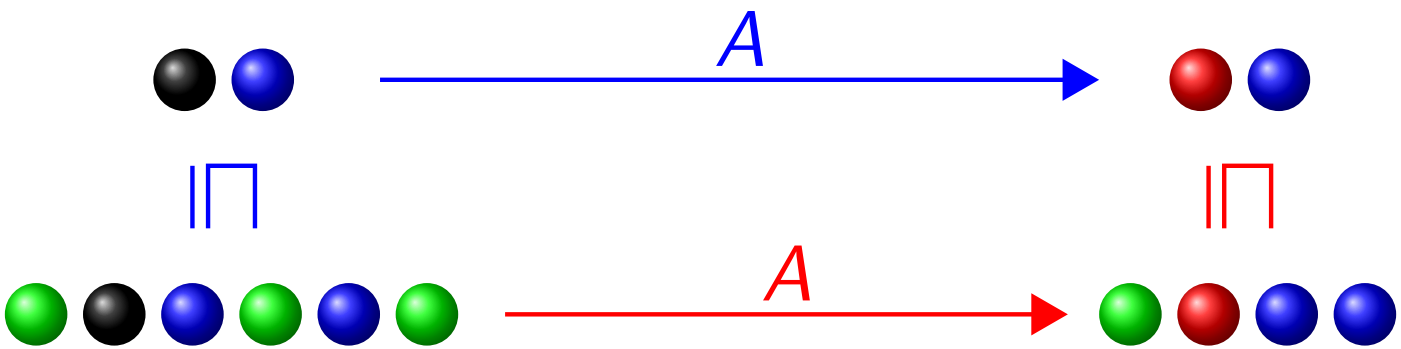
Monotonic?



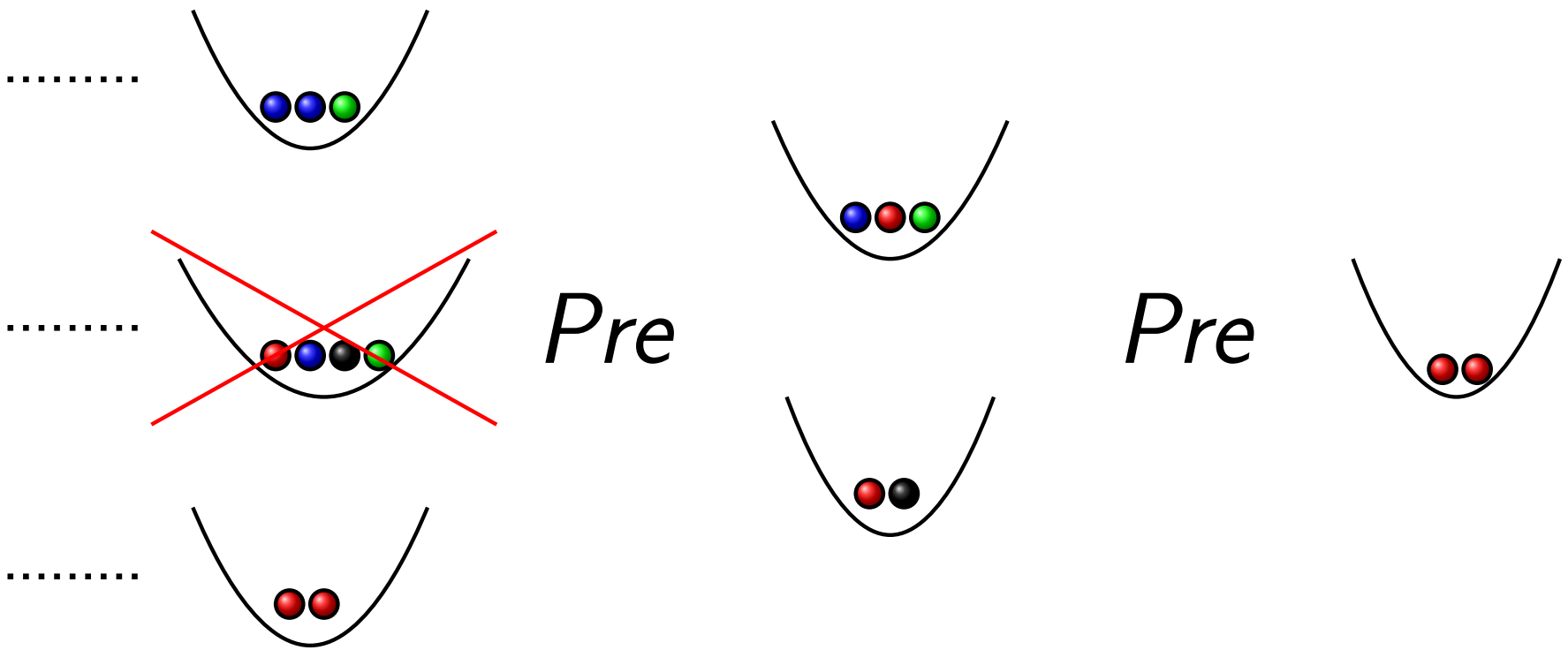
Abstraction (Over-Approximation)



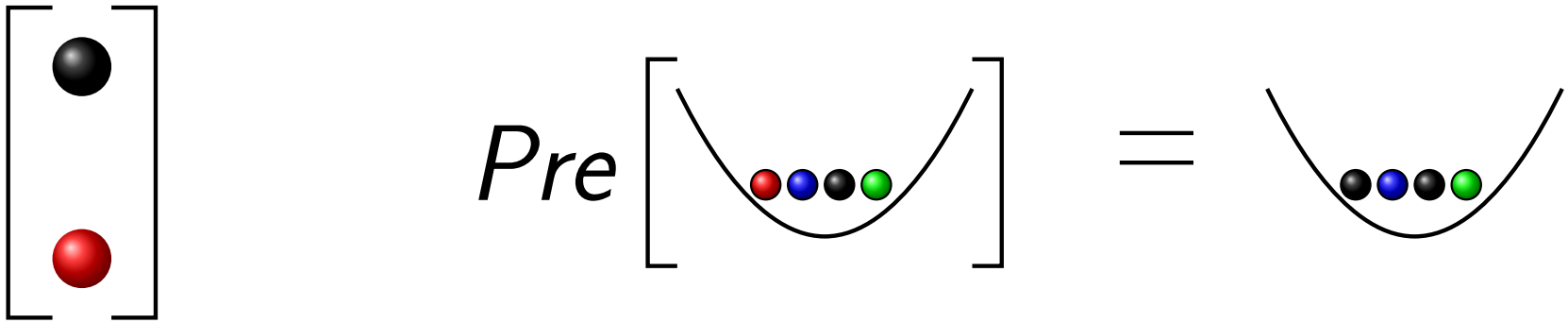
Monotonic? Yes



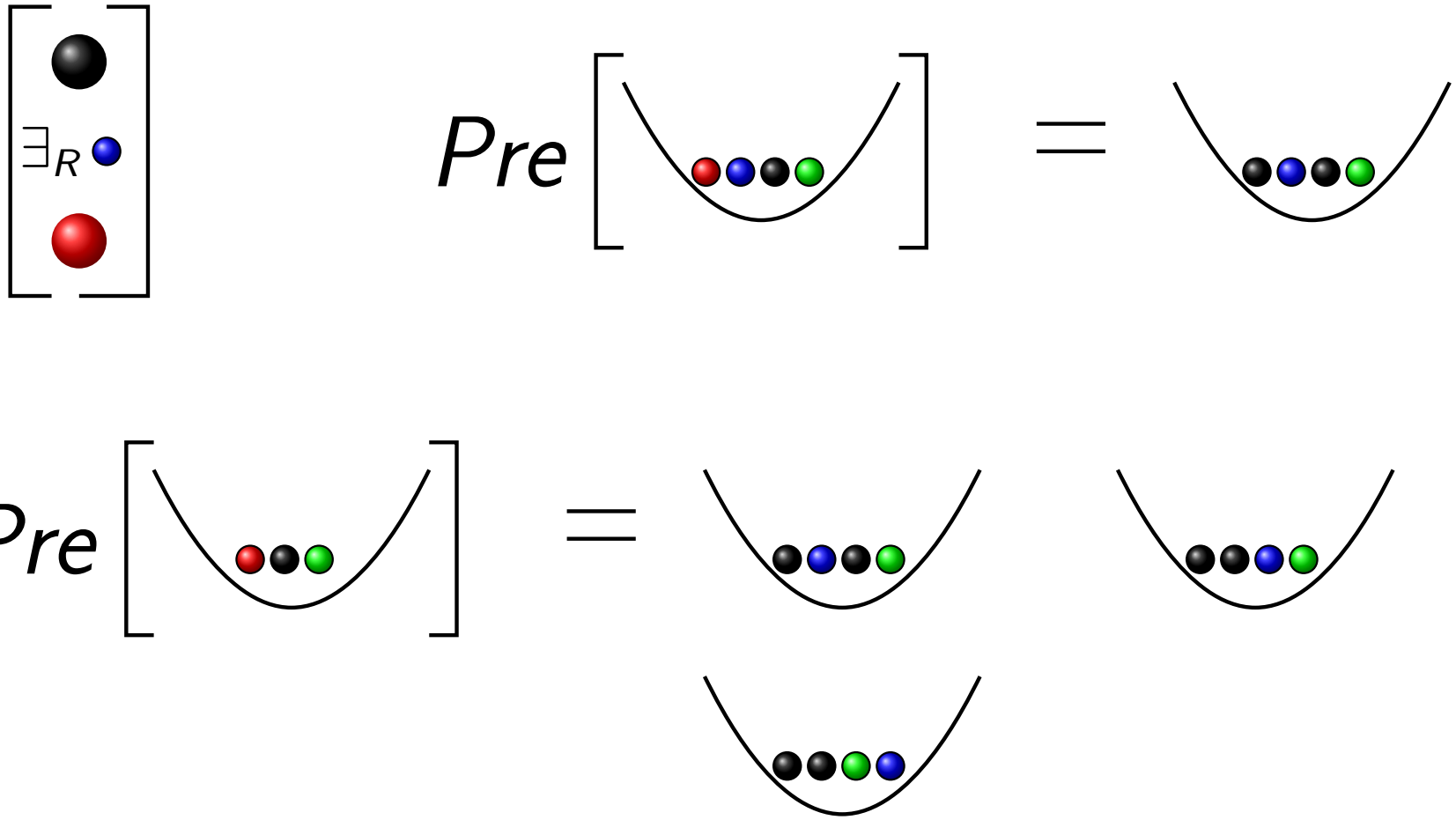
Backward Reachability Analysis on Abstract System



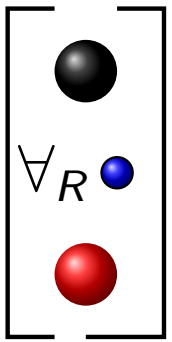
Pre - Local Transitions



Pre - Existential Global Transitions



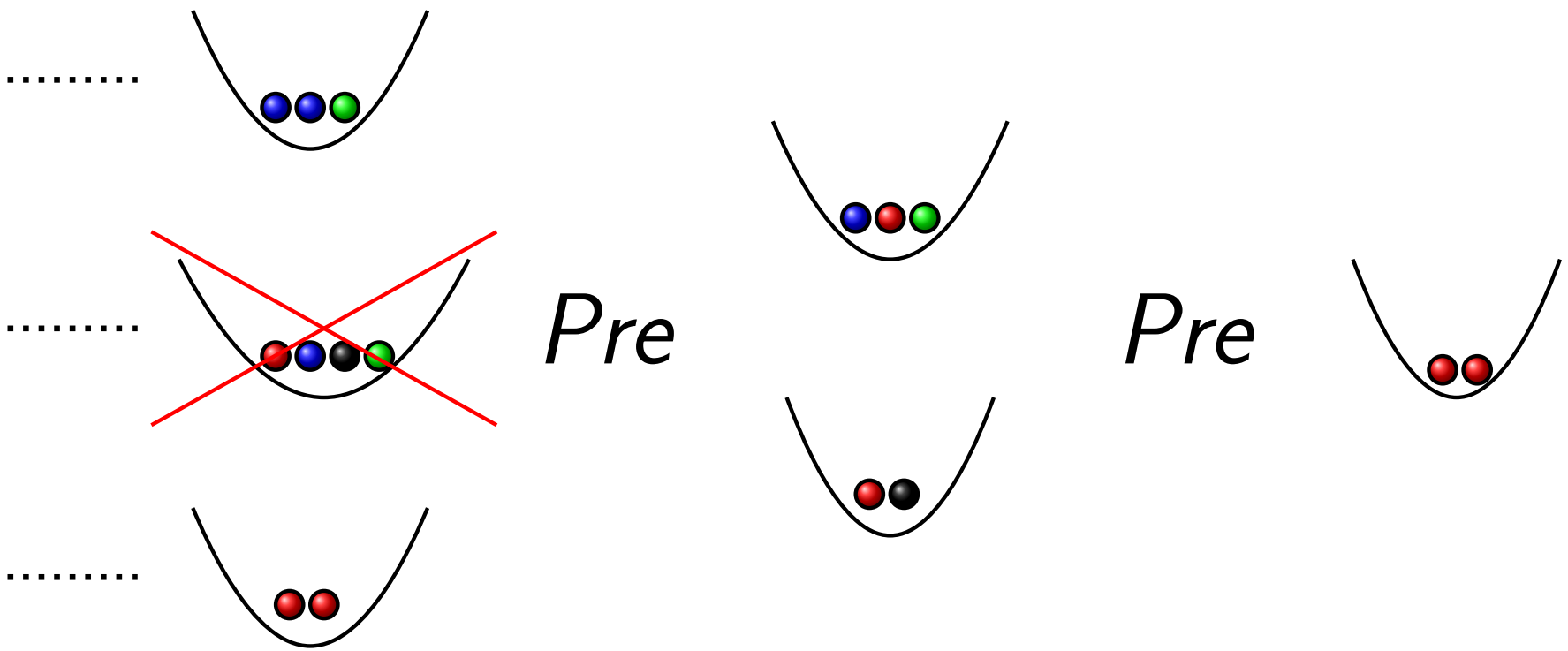
Pre - Universal Global Transitions



$$Pre\left[\quad\right] = \quad$$

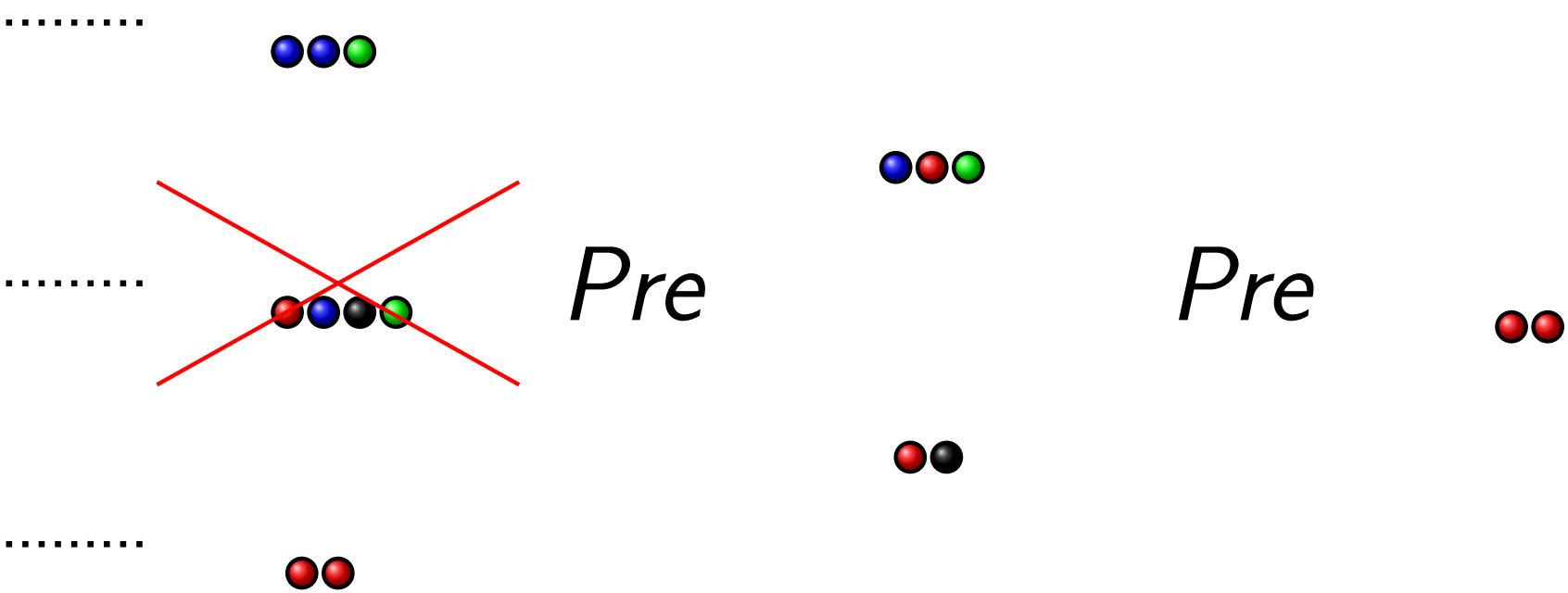
$$Pre\left[\quad\right] = \emptyset$$

Backward Reachability Analysis on Abstract System



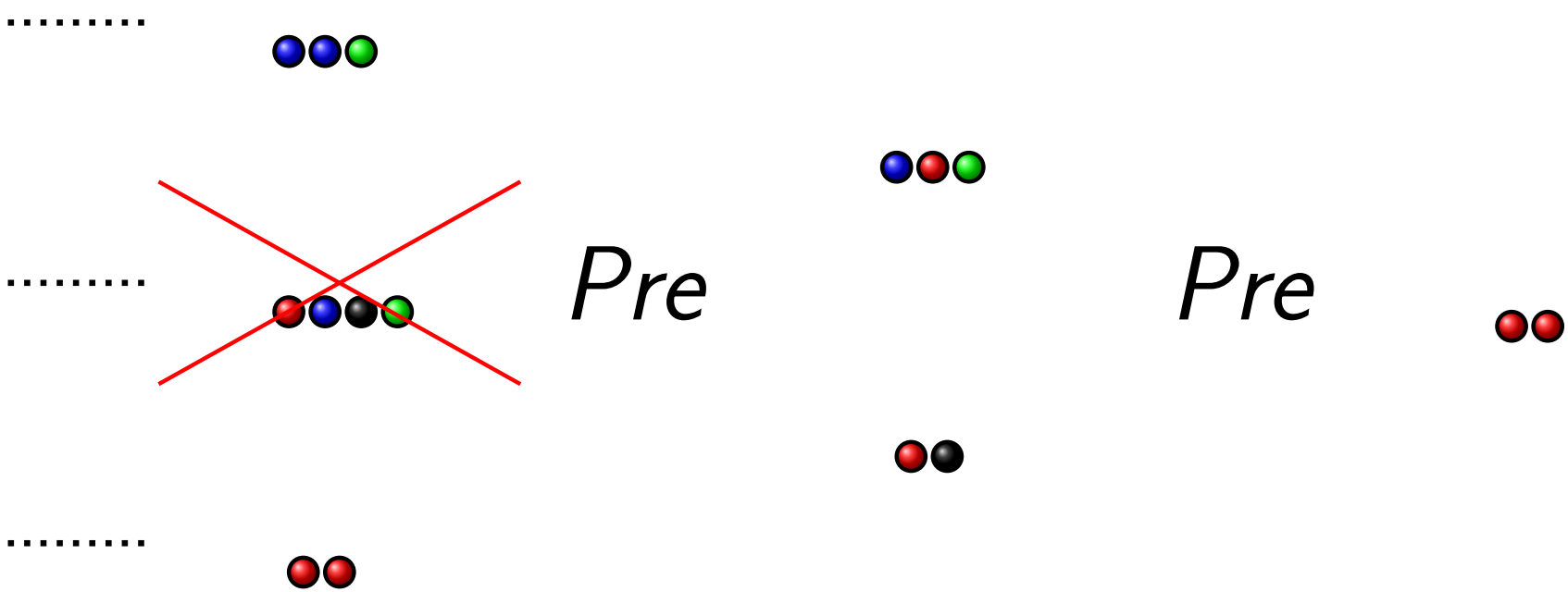
Backward Reachability Analysis on Abstract System

symbolic representation = finite words



Backward Reachability Analysis on Abstract System

symbolic representation = finite words



Termination

- Subword relation is a well quasi-ordering.
- Reachability algorithm guaranteed to terminate.

Experimental Results

Mutual Exclusion Protocols

	# iter	# constr	t(ms)
Bakery	2	2	4
Bakery*	2	2	4
Burns	14	71	230
Burns*	9	21	32
Java M-lock	5	24	30
Java M-lock*	5	17	30
Dijkstra	13	150	1700
Dijkstra*	8	57	168
Szymanski	17	334	3880
Szymanski*	17	334	4080

Experimental Results

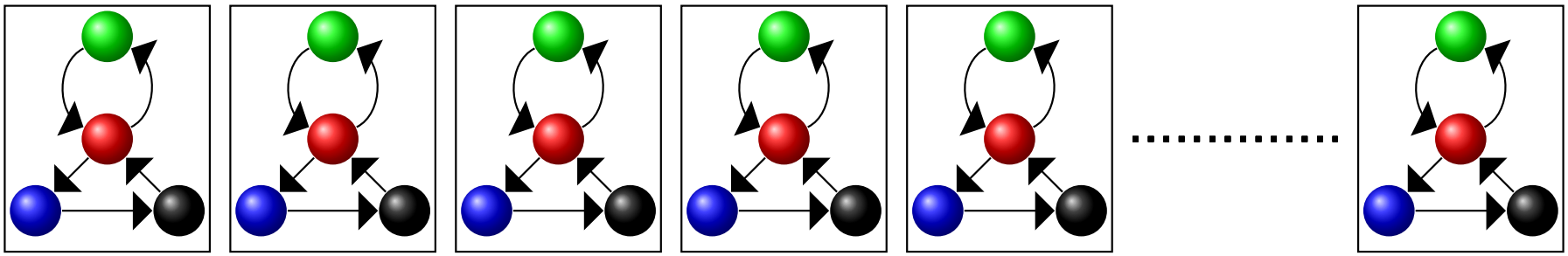
Cache Coherence Protocols

	# iter	# constr	t(ms)
Synapse	3	3	4
Berkeley	2	6	8
Mesi	3	8	8
Moesi	1	12	12
Dec Firefly	3	11	16
Xerox P.D	3	20	52
Illinois	5	33	80
Futurebus	7	153	300
German	44	14475	3h45mn

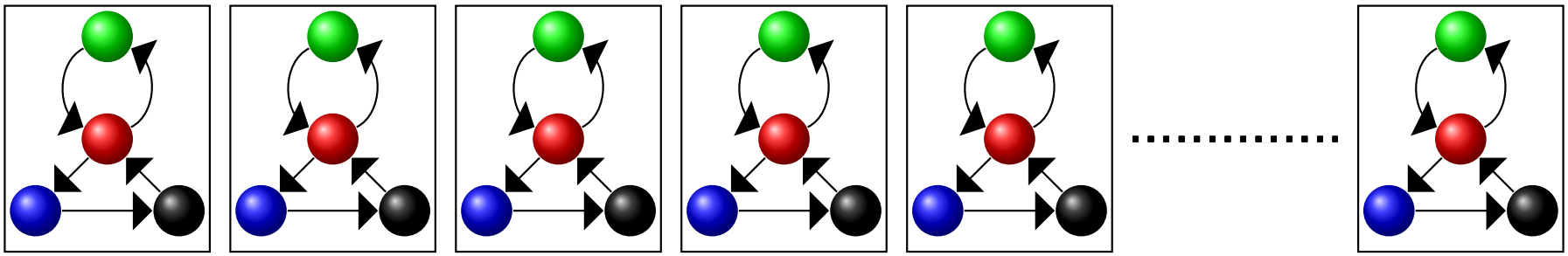
Summary- Parameterized systems of finite-state processes

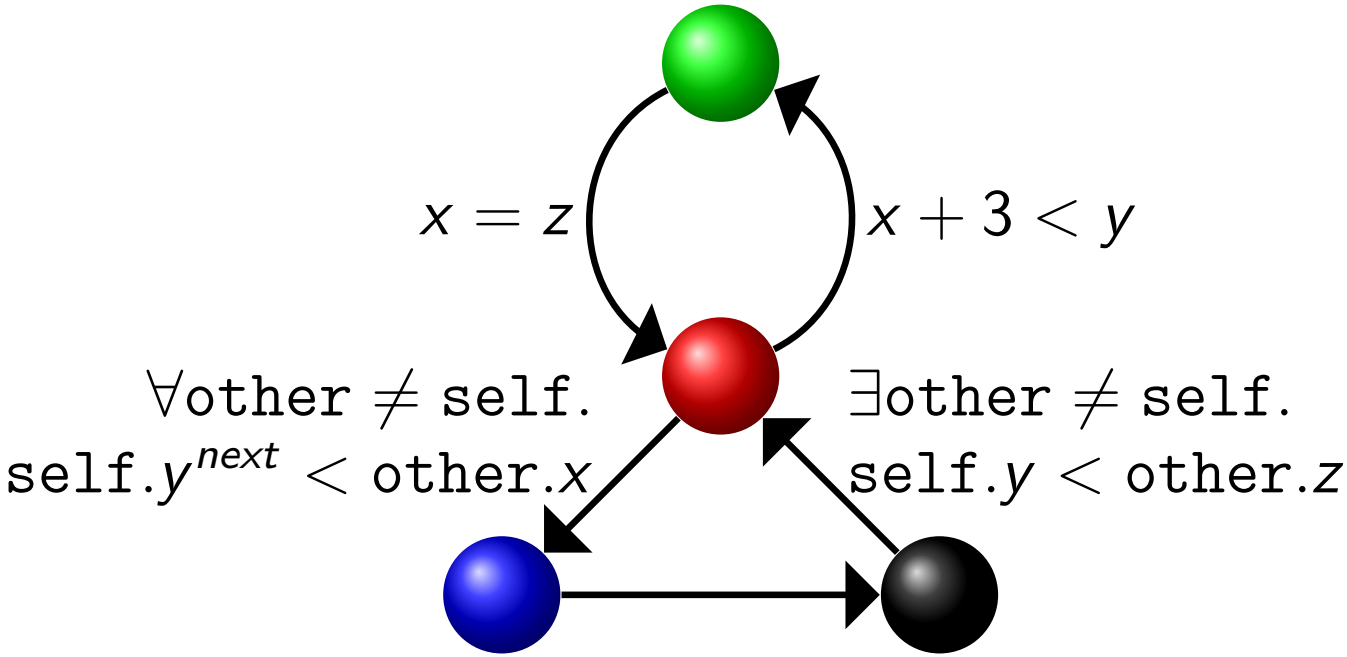
- Monotonicity allows working with upward closed sets
- Symbolic representation = words:
  - More powerful than finite-state abstraction
  - More powerful than counter abstraction
  - Less heavy than general regular expressions (transducer-based methods, e.g., regular model checking)
- Simple abstraction gives monotonicity
- Works on difficult examples !!

Parameterized Systems

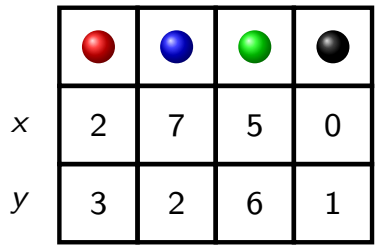


Parameterized Systems

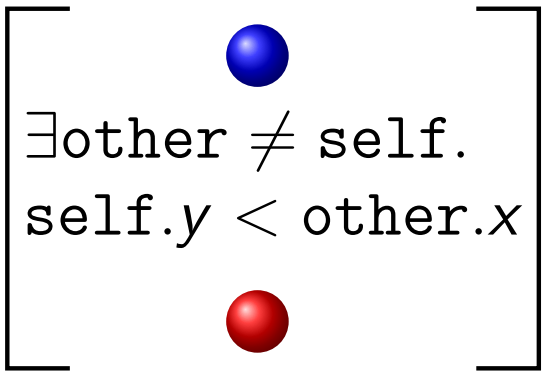


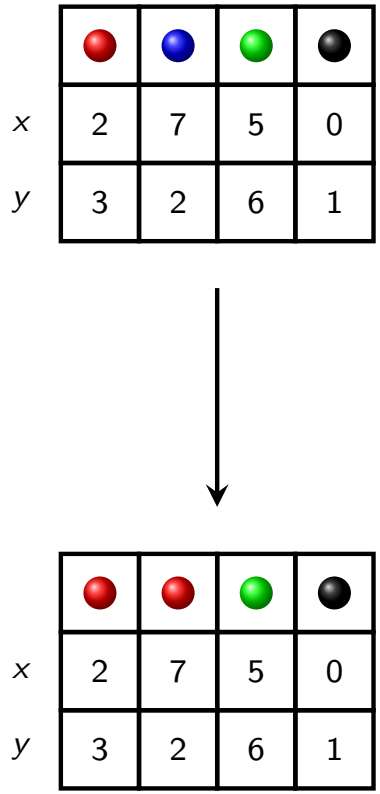


Configurations

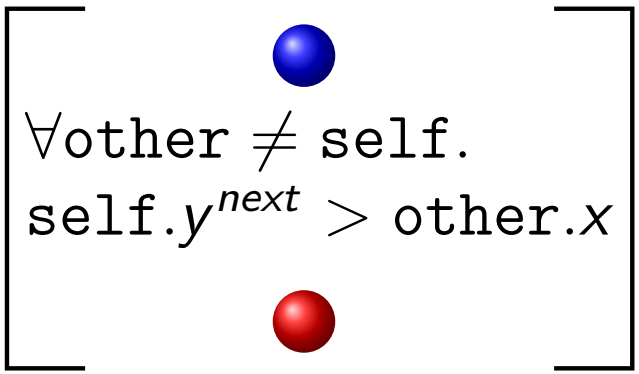


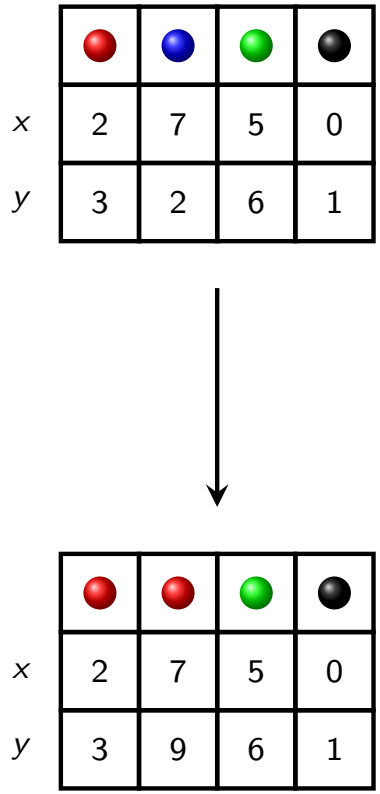
Transitions





Transitions





Ricart-Agrawala's Algorithm

Instance

$Q_P : q_{idle}, q_{wait}, q_{use}$
$Q_C : q_{empty}, q_{req}, q_{ack}, q_{ok}$
$X_P : \{id, clock, last \in \mathcal{N}\}$
$X_C : \{s\_id, r\_id, ts \in \mathcal{N}\}$

Transitions (Part I): Request, Entry, and Exit

$$t_1 : \left[q_{idle} \longrightarrow q_{wait} \ \rhd \left(\begin{array}{l} clock' > clock \\ \wedge \\ \quad \forall\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \left(\begin{array}{l} \texttt{other} \cdot state = empty \ \wedge \ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = req \ \wedge \ \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{array} \right) \end{array} \right) \right]$$

$$t_2 : \left[q_{wait} \longrightarrow q_{wait} \ \rhd \left(\begin{array}{l} clock' > clock \\ \wedge \\ \quad \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \left(\begin{array}{l} \texttt{other} \cdot state = ack \ \wedge \ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = ok \ \wedge \ \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad \wedge \ \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{array} \right) \end{array} \right) \right]$$

$$t_3 : \left[q_{wait} \longrightarrow q_{use} \ \triangleright \ \left(\begin{array}{l} \forall\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \texttt{other} \cdot s\_id = \texttt{self} \cdot id \ \supset \ \texttt{other} \cdot state = ok \end{array} \right) \right]$$

$$t_4 : \left[q_{use} \longrightarrow q_{idle} \ \triangleright \ \left(\begin{array}{l} clock' > clock \\ \wedge \\ \forall\, \texttt{other} \neq \texttt{self} \cdot \\ \left(\begin{array}{l} \texttt{other} \cdot state = ok \ \wedge \ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = empty \end{array} \right) \\ \wedge \\ \forall\, \texttt{other} \neq \texttt{self} \cdot \\ \left(\begin{array}{l} \texttt{other} \cdot state = req \ \wedge \ \texttt{other} \cdot r\_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = ack \ \wedge \ \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad \wedge \ \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{array} \right) \end{array} \right) \right]$$

Ricart-Agrawala's Algorithm, Part II: Reply

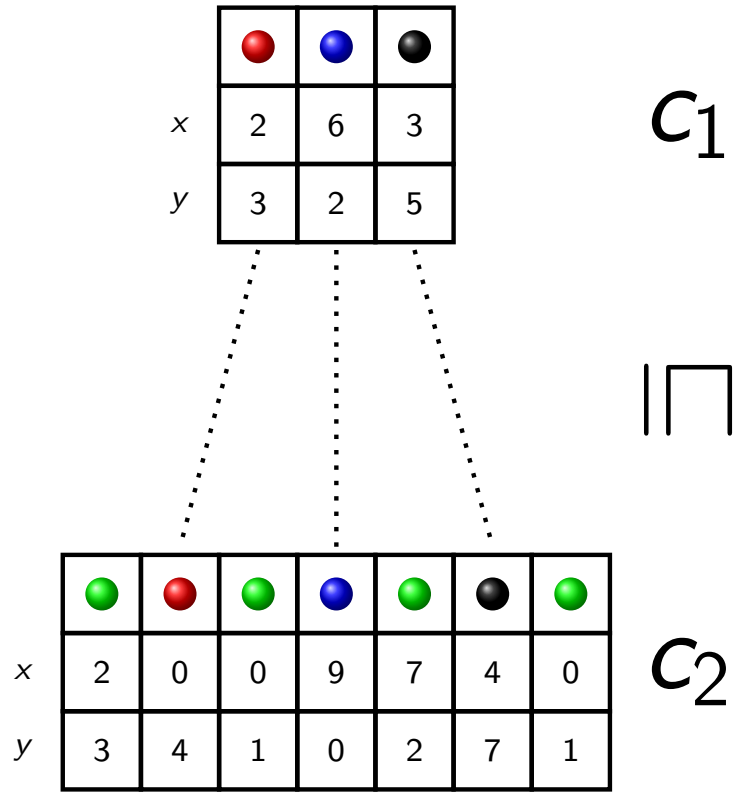
$$t_5 : \left[q_{idle} \longrightarrow q_{idle} \;\rhd\; \left(\begin{array}{l} clock' > clock \\ \wedge \\ \quad \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad\quad \left(\begin{array}{l} \texttt{other} \cdot state = req \;\wedge\; \texttt{other} \cdot r\_id = \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = ack \;\wedge\; \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad\quad \wedge\; \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{array} \right) \end{array} \right) \right]$$

$$t_6 : \left[q_{wait} \longrightarrow q_{wait} \;\rhd\; \left(\begin{array}{l} clock' > clock \\ \wedge \\ \quad \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad\quad \left(\begin{array}{l} \texttt{other} \cdot state = req \;\wedge\; \texttt{other} \cdot r\_id = \texttt{self} \cdot id \\ \quad\quad \wedge\; \texttt{other} \cdot ts < \texttt{self} \cdot last \\ \supset \\ \texttt{other} \cdot state' = ack \;\wedge\; \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad\quad \wedge\; \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{array} \right) \end{array} \right) \right]$$

$$t_7 : \left[q_{wait} \longrightarrow q_{wait} \;\rhd\; \left(\begin{array}{l} clock' > clock \\ \wedge \\ \quad \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad\quad \left(\begin{array}{l} \texttt{other} \cdot state = req \;\wedge\; \texttt{other} \cdot r\_id = \texttt{self} \cdot id \\ \quad\quad \wedge\; \texttt{other} \cdot ts = \texttt{self} \cdot last \;\wedge \\ \quad\quad\quad \wedge\; \texttt{other} \cdot s\_id < \texttt{self} \cdot id \\ \supset \\ \texttt{other} \cdot state' = ack \;\wedge\; \texttt{self} \cdot clock' > \texttt{other} \cdot ts \\ \quad\quad \wedge\; \texttt{other} \cdot ts' = \texttt{self} \cdot clock' \end{array} \right) \end{array} \right) \right]$$

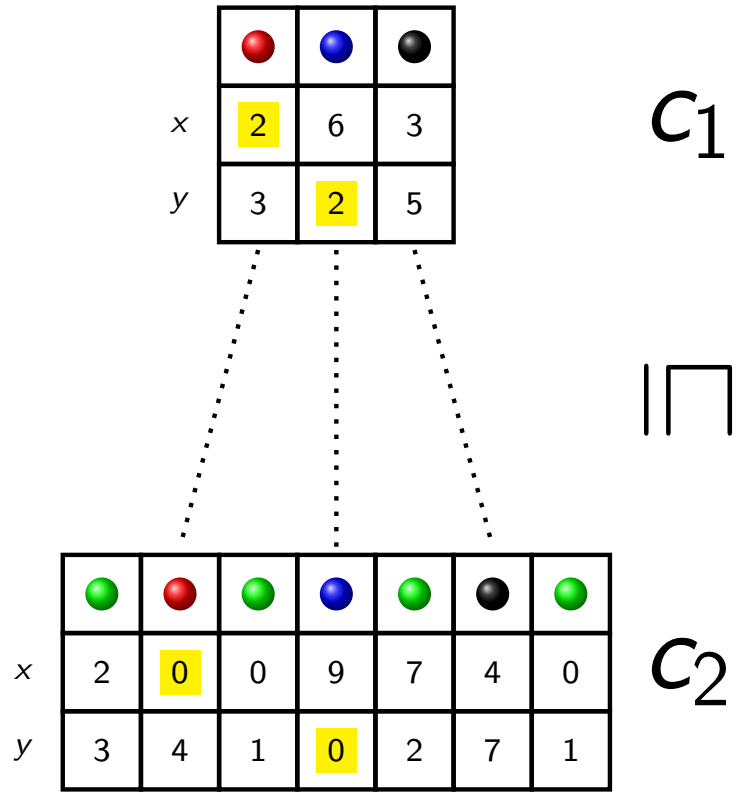
Ordering on Configurations (gap-order)

- Identical control states
- Preserves equality
- Gaps in c_1 $\leq$ Gaps in c_2



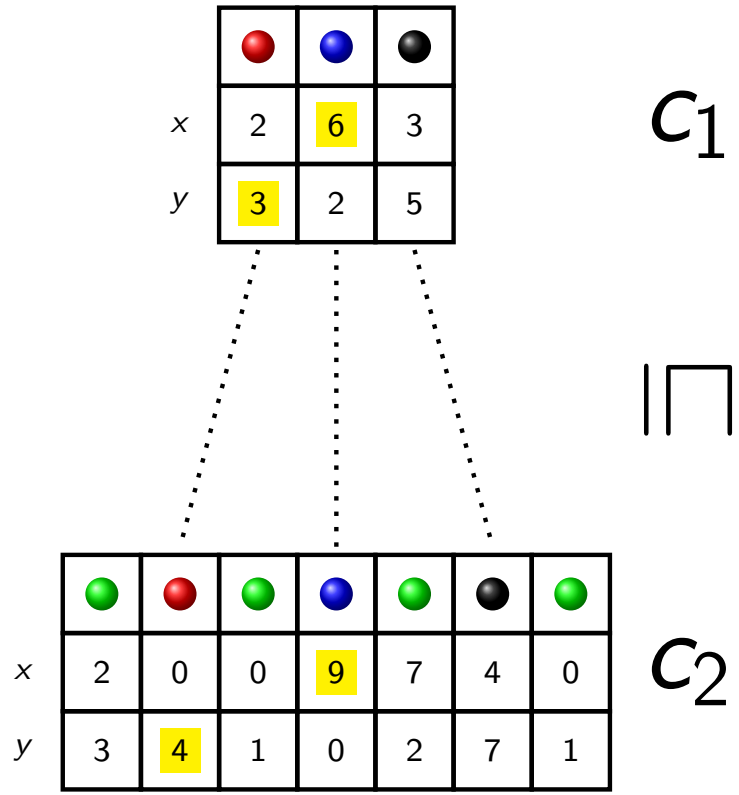
Ordering on Configurations (gap-order)

- Identical control states.
- Preserves equality
- Gaps in c_1 $\leq$ Gaps in c_2

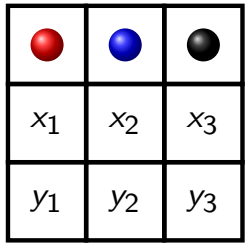


Ordering on Configurations (gap-order)

- Identical control states.
- Preserves equality
- Gaps in c_1 $\leq$ Gaps in c_2

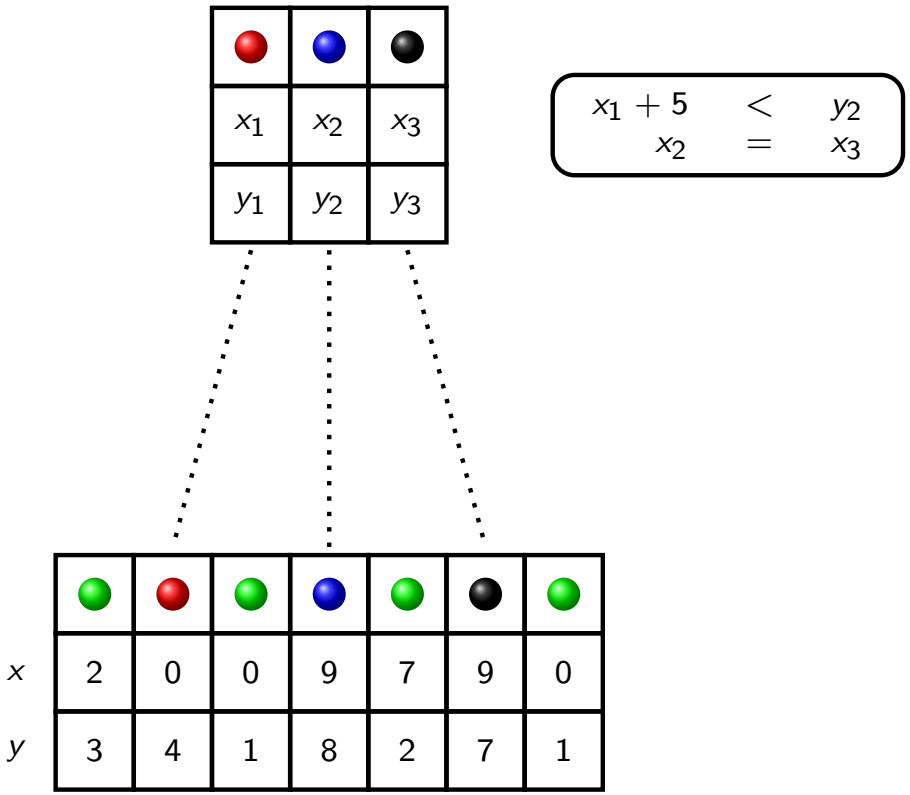


Gap-Order Constraints

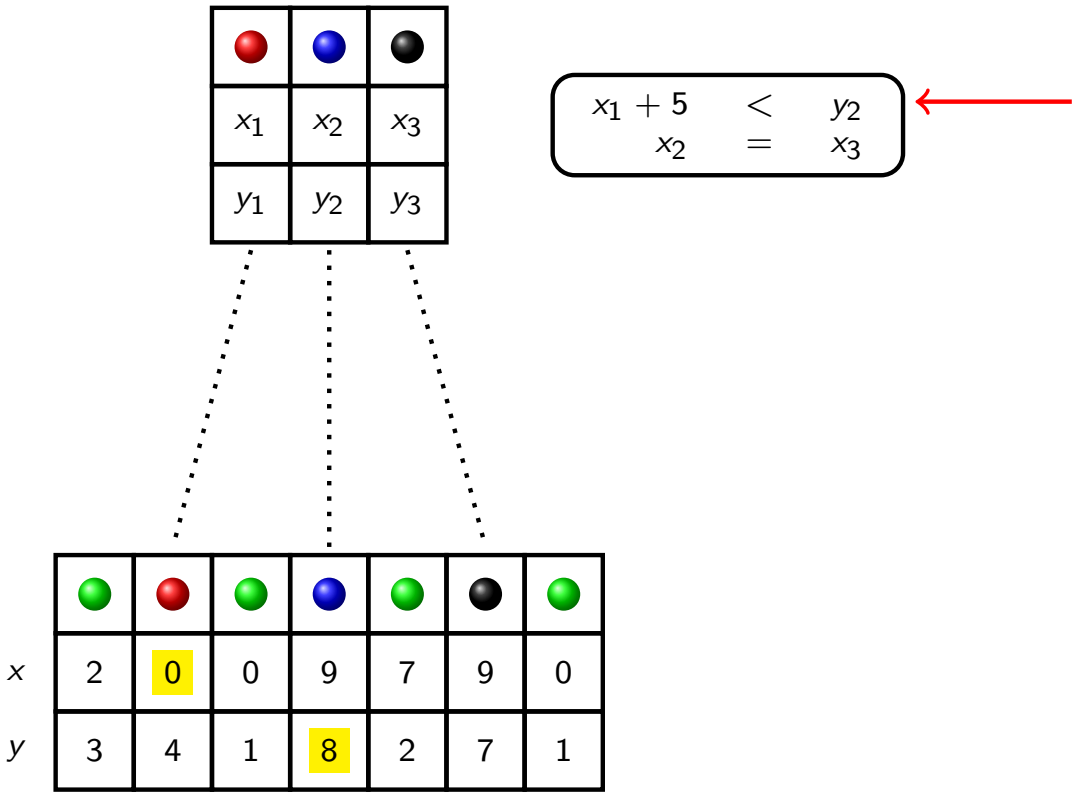


$$\begin{array}{rcl} x_1 + 5 & < & y_2 \\ x_2 & = & x_3 \end{array}$$

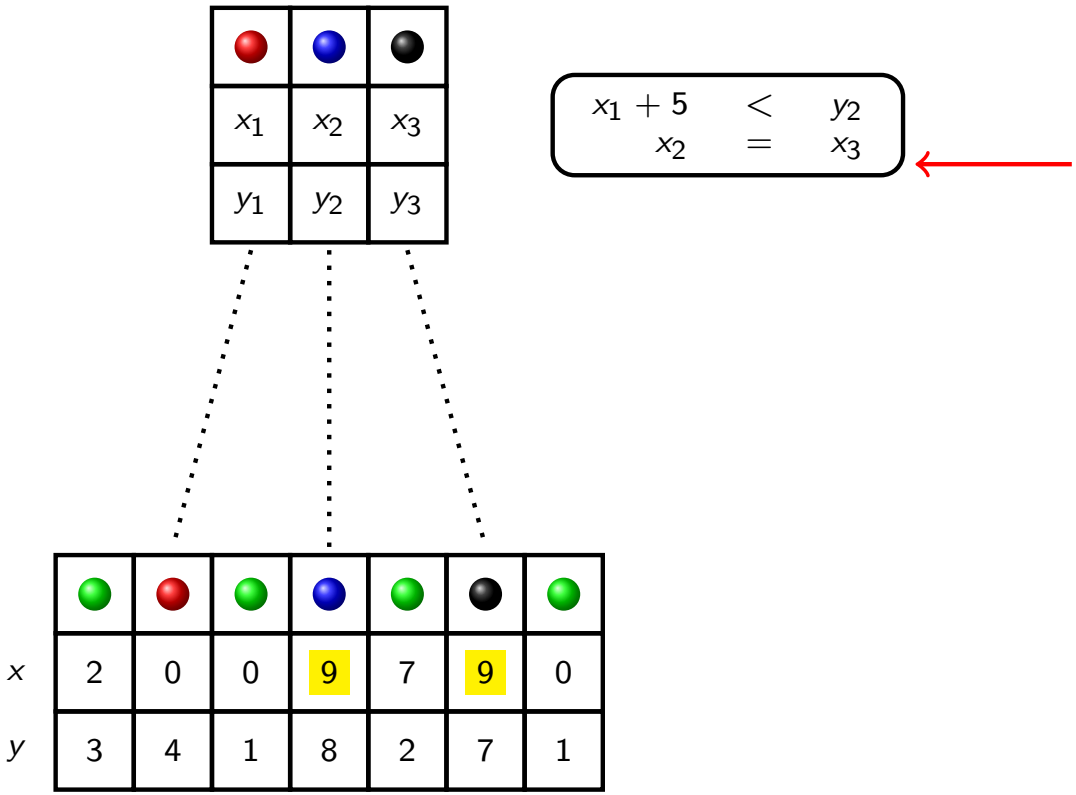
Gap-Order Constraints



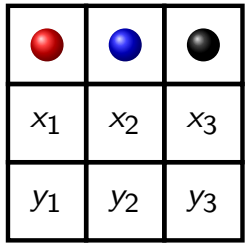
Gap-Order Constraints



Gap-Order Constraints



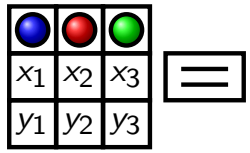
Gap-Order Constraints



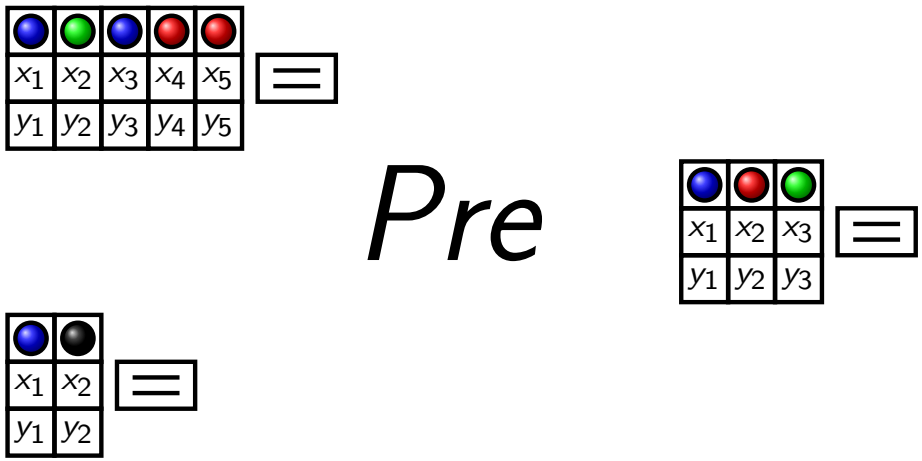
$$\begin{array}{rcl} x_1 + 5 & < & y_2 \\ x_2 & = & x_3 \end{array}$$

upward closed set of configurations

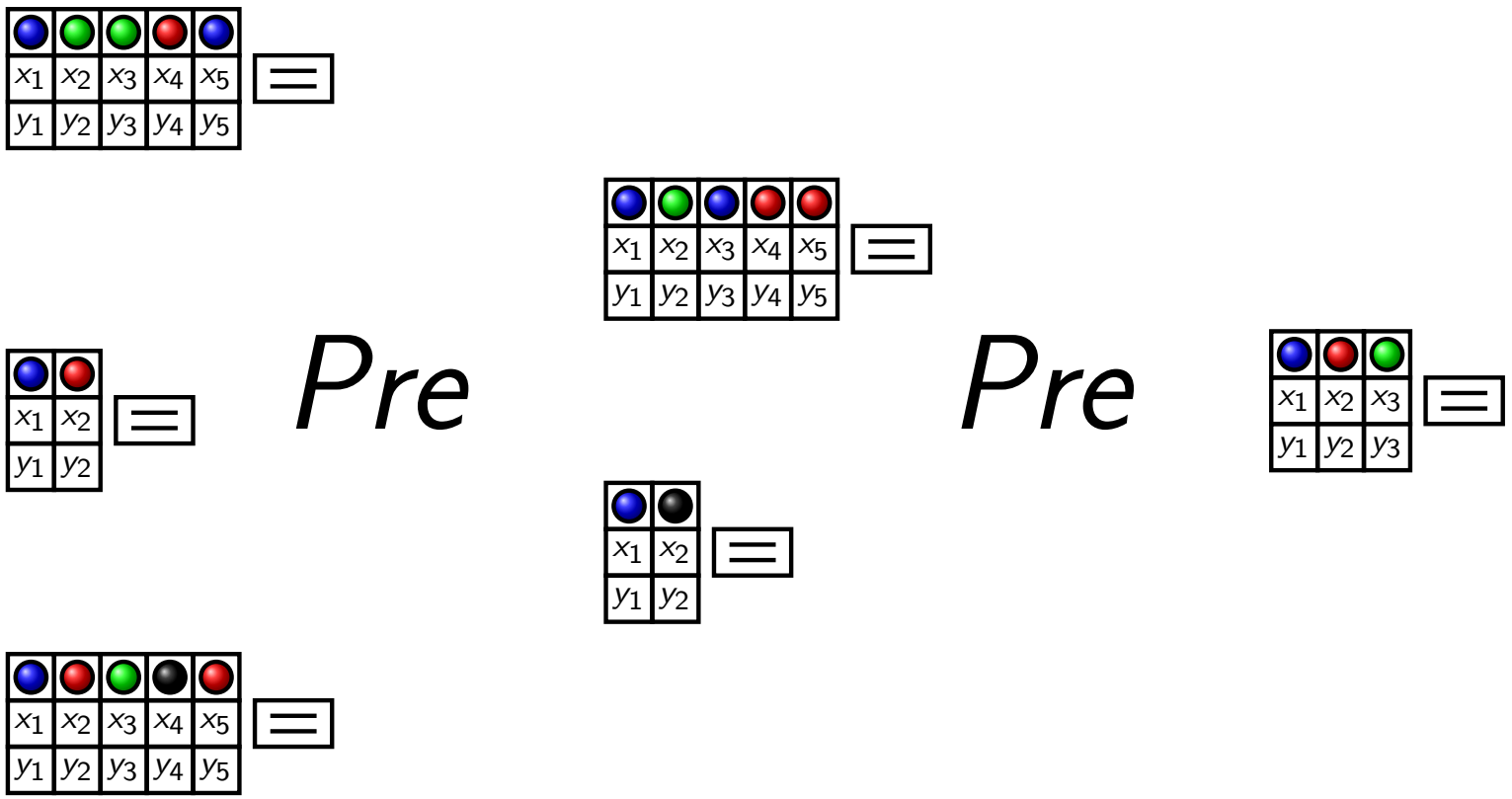
Backward Reachability Analysis



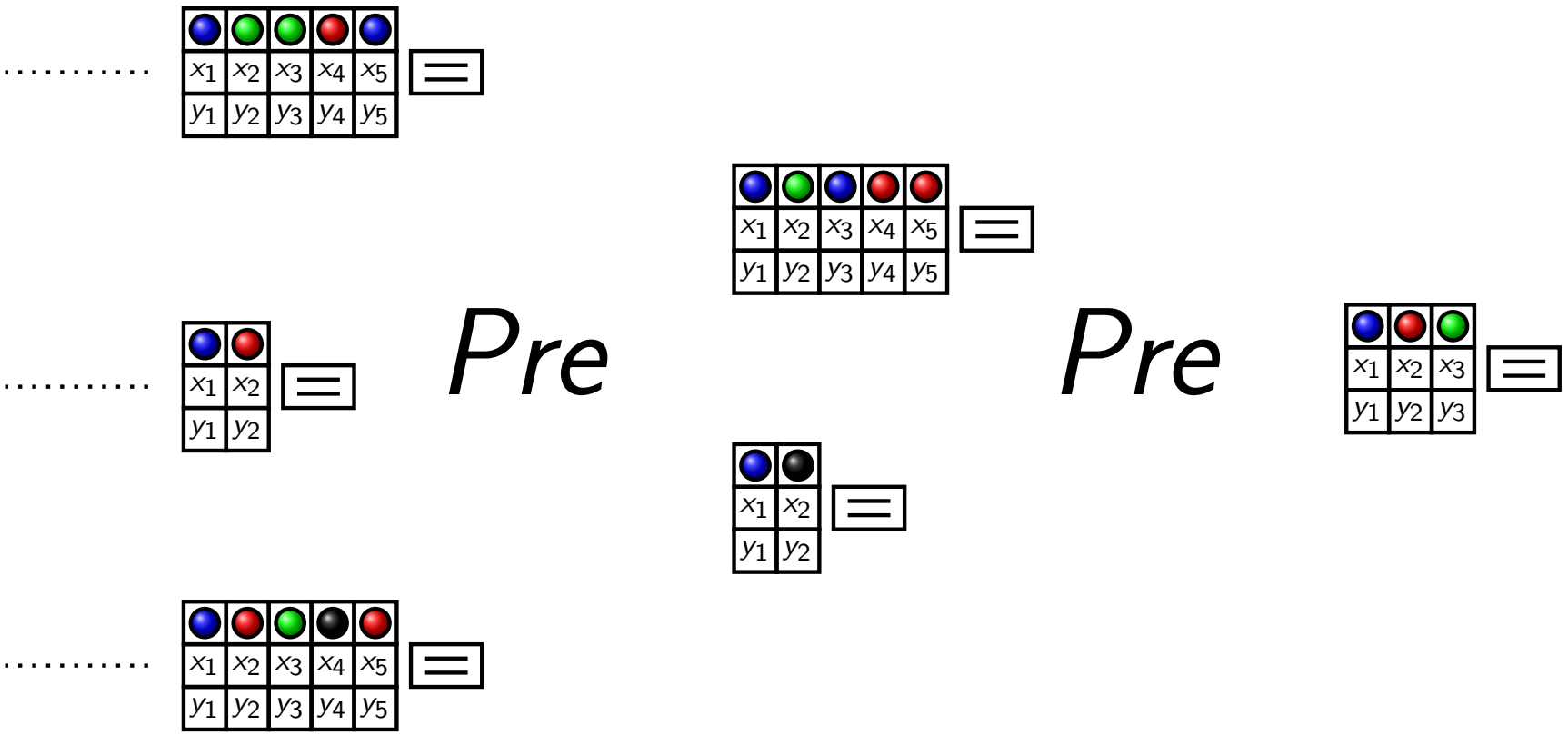
Backward Reachability Analysis



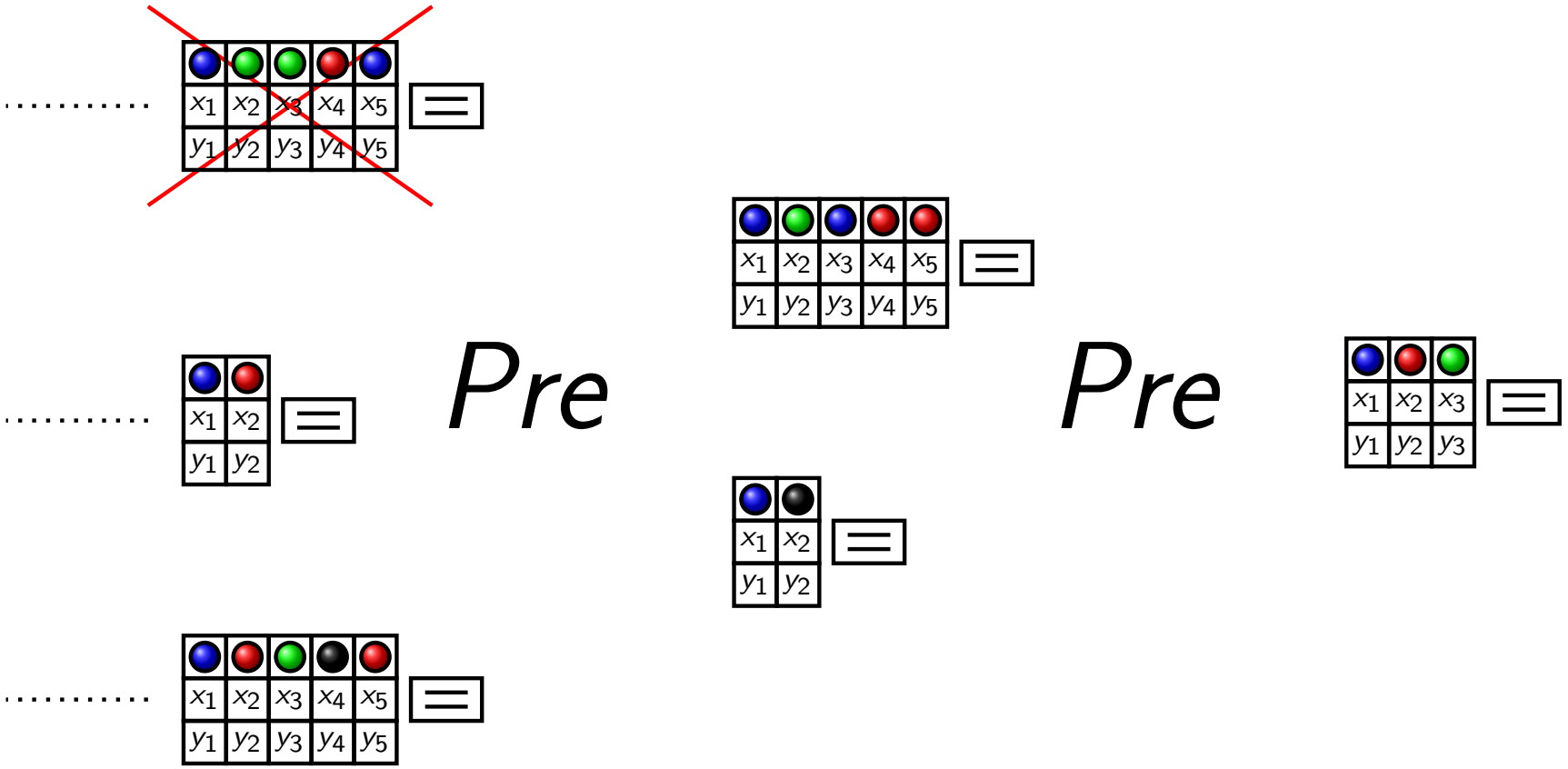
Backward Reachability Analysis



Backward Reachability Analysis

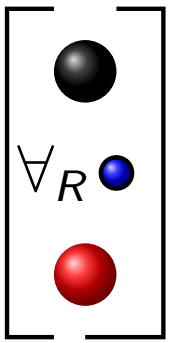


Backward Reachability Analysis



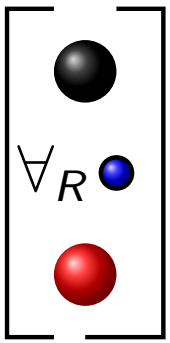
Non-Atomic Global Conditions

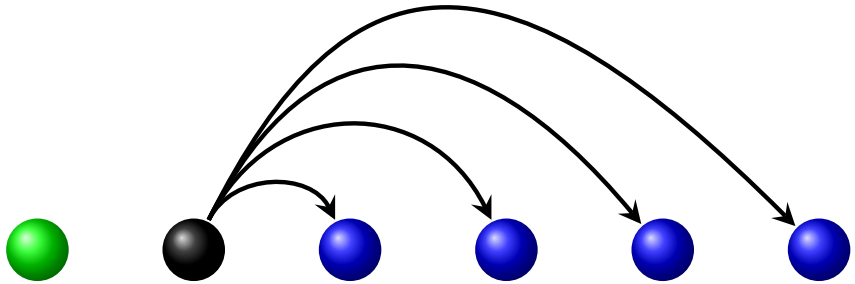
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received



Non-Atomic Global Conditions

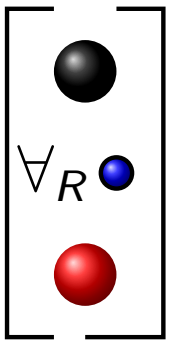
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received

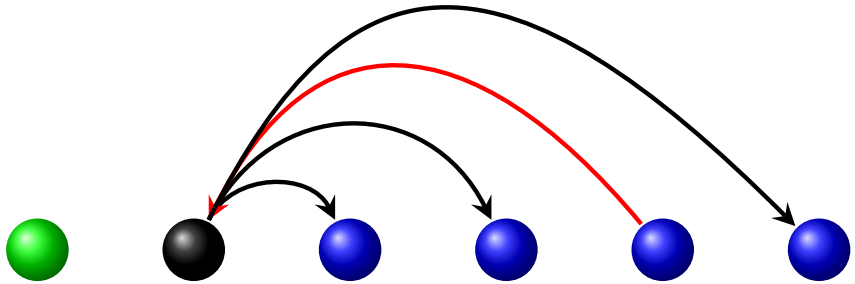




Non-Atomic Global Conditions

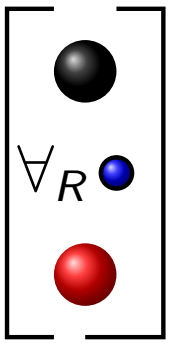
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received

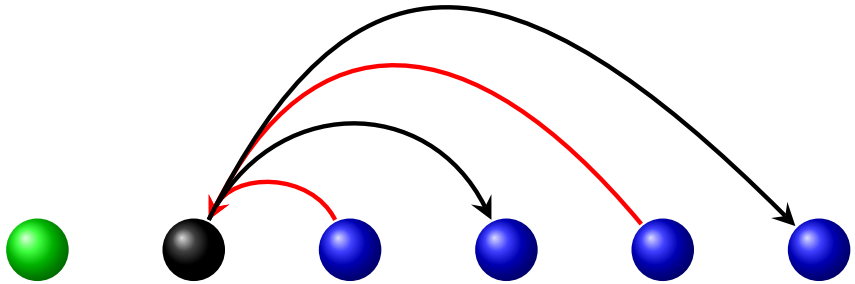




Non-Atomic Global Conditions

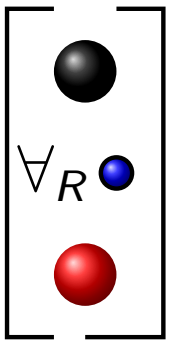
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received

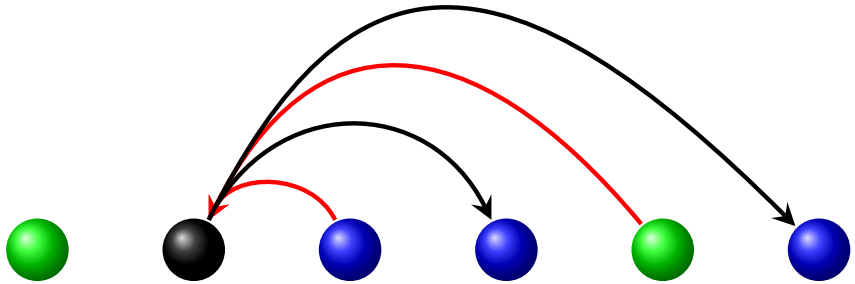




Non-Atomic Global Conditions

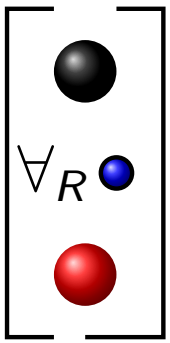
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received

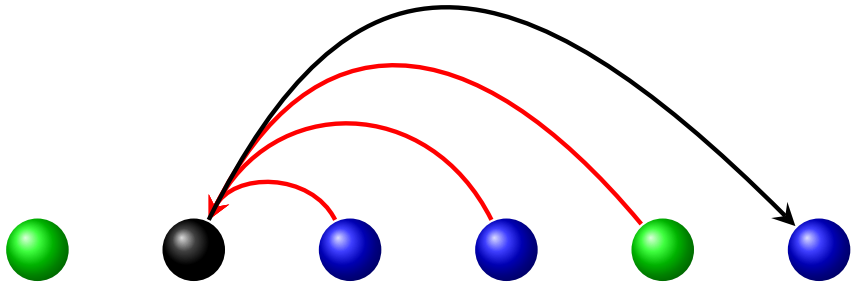




Non-Atomic Global Conditions

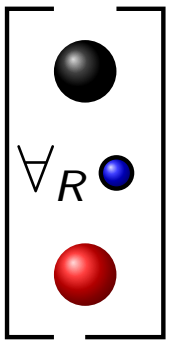
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received

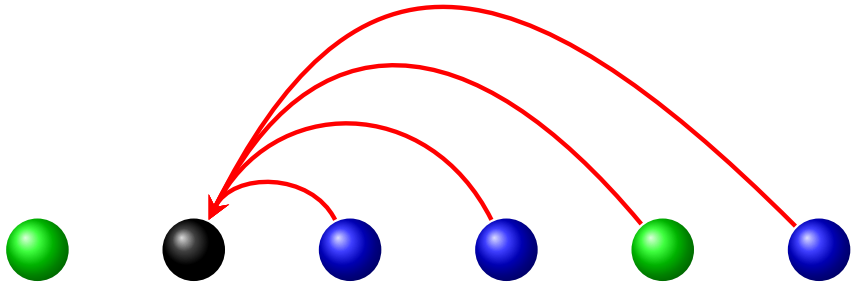




Non-Atomic Global Conditions

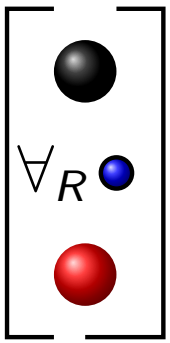
- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received





Non-Atomic Global Conditions

- Replace global condition with protocol:
  - Send request
  - Acks sent successively
  - Perform transition when all acks received



Lamport's Distributed Mut-Ex

$Q_P : q_{idle}, q_{wait}, q_{use}$
$Q_C : q_{empty}, q_{req_1}, q_{ack_1}, q_{ok_1}, q_{req_2}, q_{ack_2}, q_{ok_2}$
$X_P : \{id, num, aux \in \mathcal{N}\}$
$X_C : \{s\_id, r\_id, v \in \mathcal{N}\}$

Part I: Distributed Computation of Number

$$t_1 : \left[q_{idle} \longrightarrow q_{choose} \ \triangleright \left(\begin{array}{l} aux' = num \ \wedge \\ \ \forall\, \mathtt{other} \neq \mathtt{self} \cdot \\ \ \left(\begin{array}{l} \mathtt{other} \cdot state = empty \ \wedge \ \mathtt{other} \cdot s\_id = \mathtt{self} \cdot id \\ \supset \\ \mathtt{other} \cdot state' = req_1 \ \wedge \ \mathtt{other} \cdot v' = \mathtt{self} \cdot num \end{array} \right) \end{array} \right) \right]$$

$$t_2 : \left[q_{choose} \longrightarrow q_{choose} \ \triangleright \left(\begin{array}{l} \exists\, \mathtt{other} \neq \mathtt{self} \cdot \\ \left(\begin{array}{l} \mathtt{other} \cdot state = ack_1 \ \wedge \ \mathtt{other} \cdot s\_id = \mathtt{self} \cdot id \\ \quad \wedge \ \mathtt{other} \cdot v > \mathtt{self} \cdot aux \\ \supset \\ \mathtt{other} \cdot state' = ok_1 \ \wedge \ \mathtt{self} \cdot aux' = \mathtt{other} \cdot v \end{array} \right) \end{array} \right) \right]$$

$$t_3 : \left[q_{choose} \longrightarrow q_{choose} \ \triangleright \left(\begin{array}{l} \exists\, \mathtt{other} \neq \mathtt{self} \cdot \\ \left(\begin{array}{l} \mathtt{other} \cdot state = ack_1 \ \wedge \ \mathtt{other} \cdot s\_id = \mathtt{self} \cdot id \\ \quad \wedge \ \mathtt{other} \cdot v \leq \mathtt{self} \cdot aux \\ \supset \\ \mathtt{other} \cdot state' = ok_1 \end{array} \right) \end{array} \right) \right]$$

$$t_4 : \left[q_{choose} \longrightarrow q_{wait} \ \triangleright \left(\begin{array}{l} num' > aux \\ \wedge \\ \ \forall\, \mathtt{other} \neq \mathtt{self} \cdot \\ \ \mathtt{other} \cdot s\_id = \mathtt{self} \cdot id \ \supset \ \mathtt{other} \cdot state = ok_1 \end{array} \right) \right]$$

Lamport's Distributed Mut-Ex, Part II: Reply

$$t_5: \left[q_s \longrightarrow q_s \;\; \triangleright \left(\begin{array}{l} \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \texttt{other} \cdot state = req_1 \;\wedge\; \texttt{other} \cdot r\_id = \texttt{self} \cdot id \\ \quad \supset \\ \quad \texttt{other} \cdot state' = ack_1 \;\wedge\; \texttt{other} \cdot v' = \texttt{self} \cdot num \end{array} \right) \right]$$

for any $s \in \{idle, choose, wait, use\}$

$$t_6: \left[q_s \longrightarrow q_s \;\; \triangleright \left(\begin{array}{l} \exists\, \texttt{other} \neq \texttt{self} \cdot \\ \quad \texttt{other} \cdot state = req_2 \;\wedge\; \texttt{other} \cdot r\_id = \texttt{self} \cdot id \\ \quad \supset \\ \quad \texttt{other} \cdot state' = ack_2 \;\wedge\; \texttt{other} \cdot v' = \texttt{self} \cdot num \end{array} \right) \right]$$

for any $s \in \{idle, wait, use\}$

Lamport's Distributed Mut-Ex, Part III: Entry and Exit

$$t_7 : \left[q_{wait} \longrightarrow q_{wait} \ \triangleright \left(\begin{array}{l} \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \texttt{other} \cdot state = ok_1 \ \wedge\ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \\ \quad \supset \\ \quad \texttt{other} \cdot state' = req_2 \end{array} \right) \right]$$

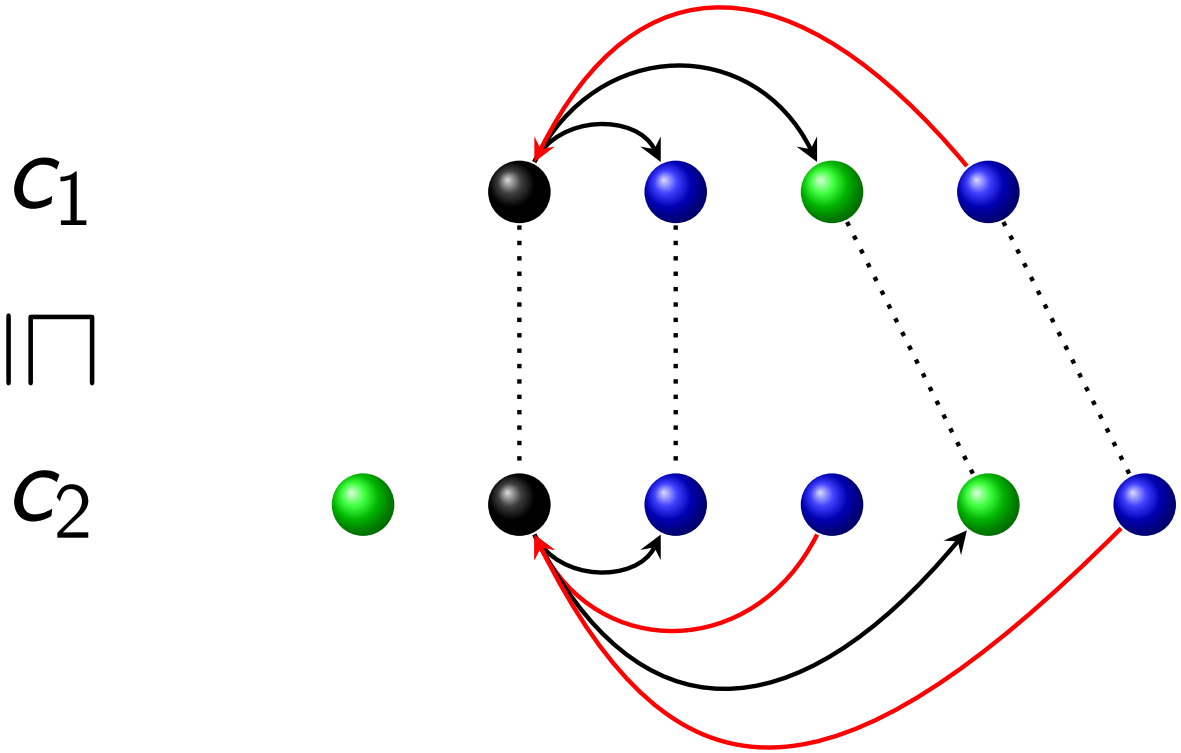
$$t_8 : \left[q_{wait} \longrightarrow q_{wait} \ \triangleright \left(\begin{array}{l} \exists\ \texttt{other} \neq \texttt{self} \cdot \\ \quad \left(\begin{array}{l} \texttt{other} \cdot state = ack_2 \ \wedge \\ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \ \wedge\ \texttt{other} \cdot v > 0 \ \wedge \\ \left(\begin{array}{l} \texttt{self} \cdot num > \texttt{other} \cdot v \ \vee \\ (\ \texttt{other} \cdot v = \texttt{self} \cdot num \ \wedge\ \texttt{self} \cdot id > r\_id\ ) \end{array} \right) \end{array} \right) \\ \quad \supset \\ \quad \texttt{other} \cdot state' = req_2 \end{array} \right) \right]$$

$$t_9 : \left[q_{wait} \longrightarrow q_{wait} \ \triangleright \left(\begin{array}{l} \exists\ \texttt{other} \neq \texttt{self} \cdot \\ \left(\begin{array}{l} \texttt{other} \cdot state = ack_2 \ \wedge \\ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \ \wedge \\ \left(\begin{array}{l} \texttt{other} \cdot v = 0 \ \vee \\ \texttt{self} \cdot num < \texttt{other} \cdot v \ \vee \\ (\texttt{other} \cdot v = \texttt{self} \cdot num \ \wedge\ \texttt{self} \cdot id < r\_id) \end{array} \right) \end{array} \right) \\ \quad \supset \\ \quad \texttt{other} \cdot state' = ok_2 \end{array} \right) \right]$$

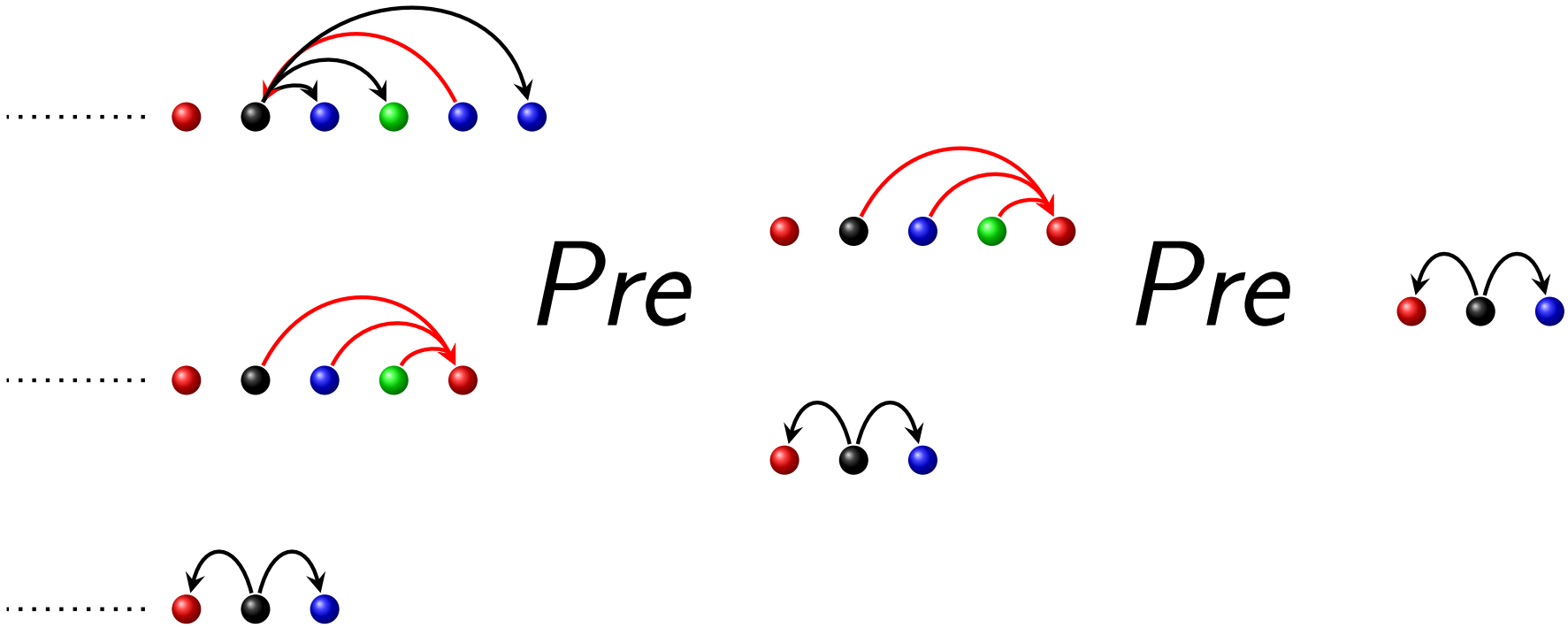
$$t_{10} : \left[q_{wait} \longrightarrow q_{use} \ \triangleright \left(\begin{array}{l} \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad s\_id = \texttt{self} \cdot id \ \supset\ \texttt{other} \cdot state = ok_2 \end{array} \right) \right]$$

$$t_{11} : \left[q_{use} \longrightarrow q_{idle} \ \triangleright \left(\begin{array}{l} num' = 0 \\ \wedge \\ \quad \forall\ \texttt{other} \neq \texttt{self} \cdot \\ \quad\quad \texttt{other} \cdot state = ok_2 \ \wedge\ \texttt{other} \cdot s\_id = \texttt{self} \cdot id \\ \quad\quad \supset \\ \quad\quad \texttt{other} \cdot state' = empty \end{array} \right) \right]$$

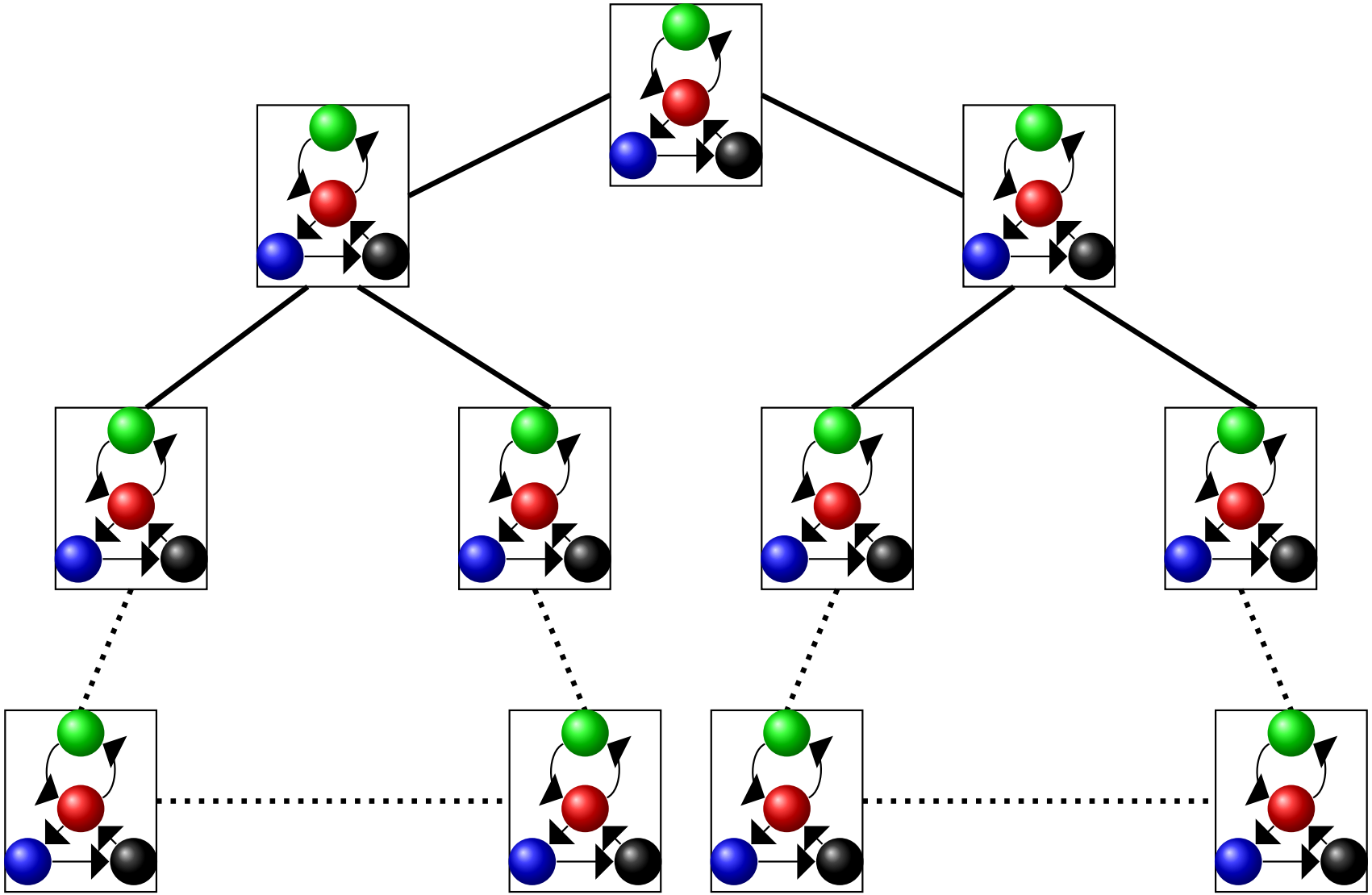
Ordering on Configurations



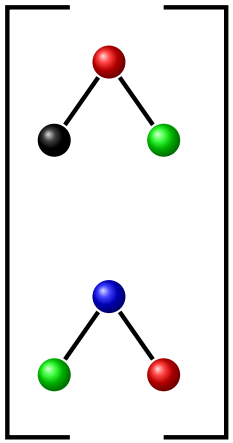
Ordering on Configurations

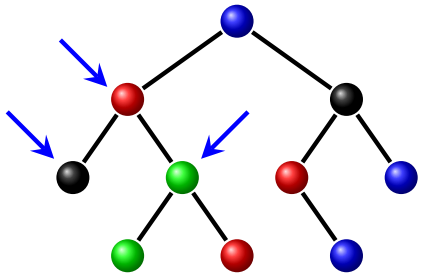


Tree Topologies

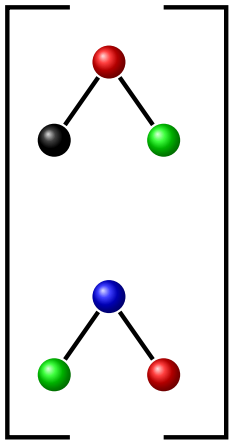


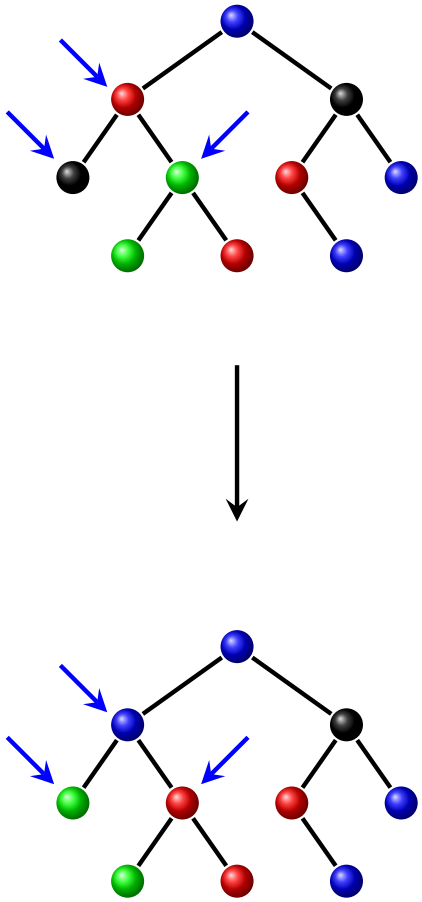
Transitions



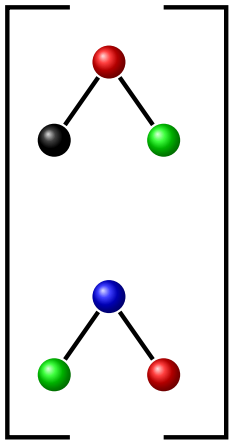


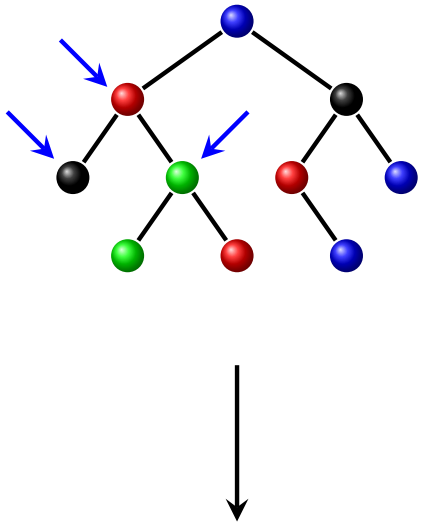
Transitions



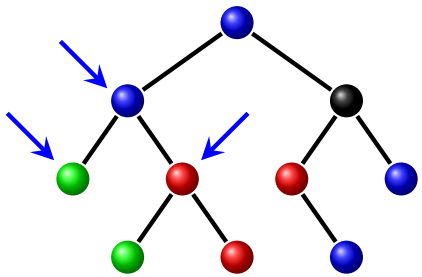


Transitions

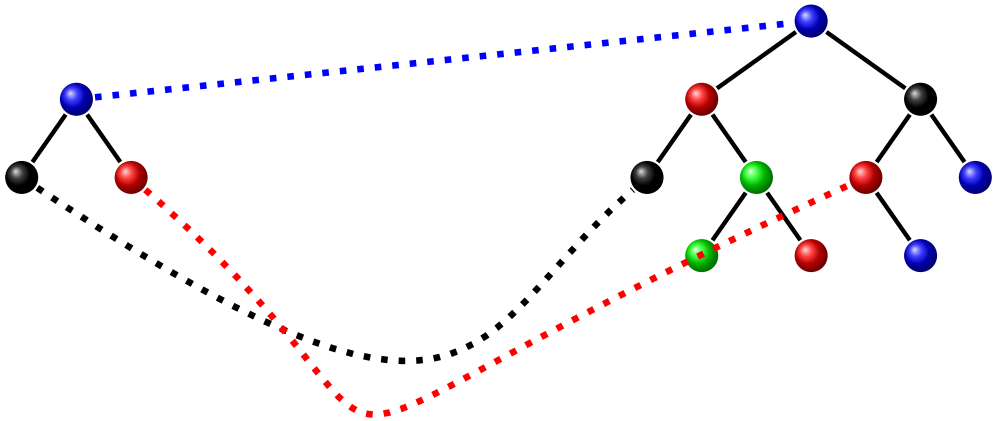




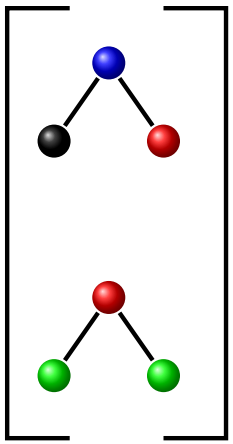
- Tree Arbiter Protocols
- The Arbiter Protocol
- Leader Election Protocols
- Distributed Protocols

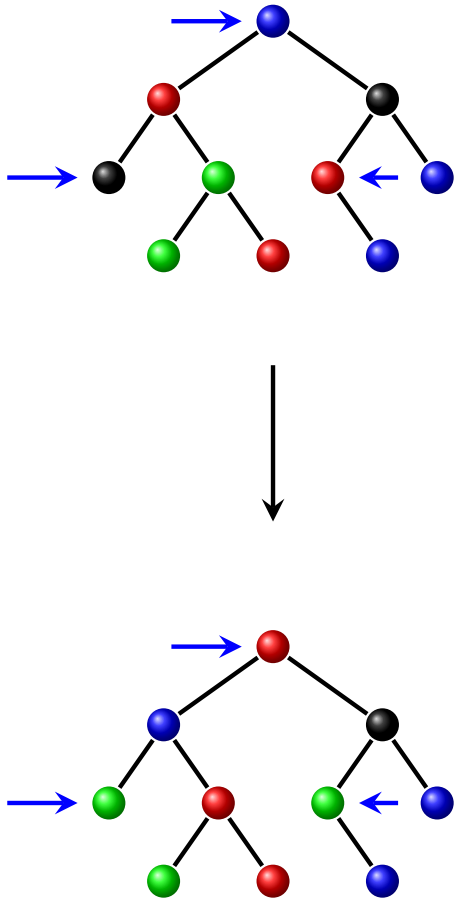


Ordering on Configurations

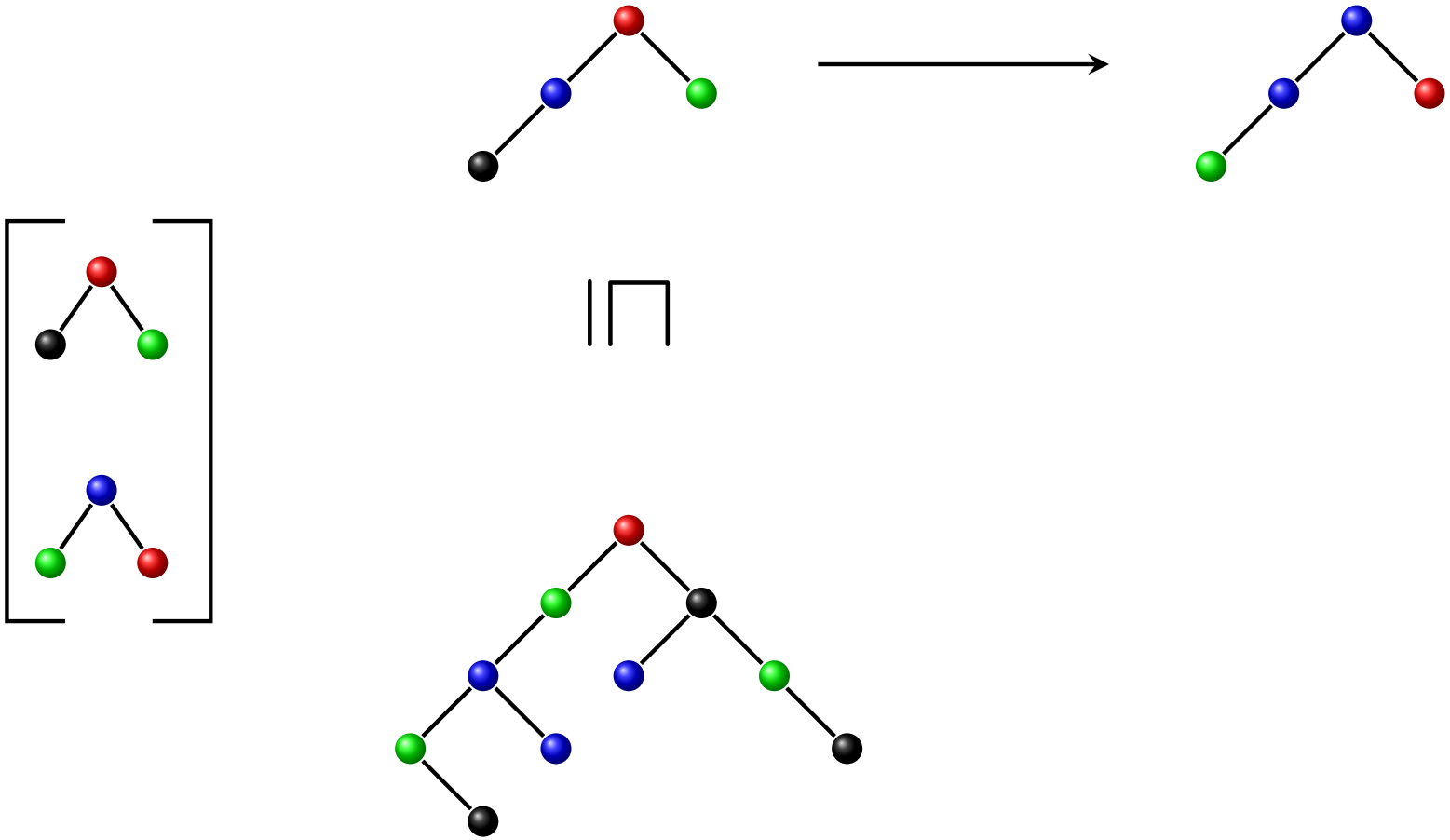


Abstraction (Over-approximation)

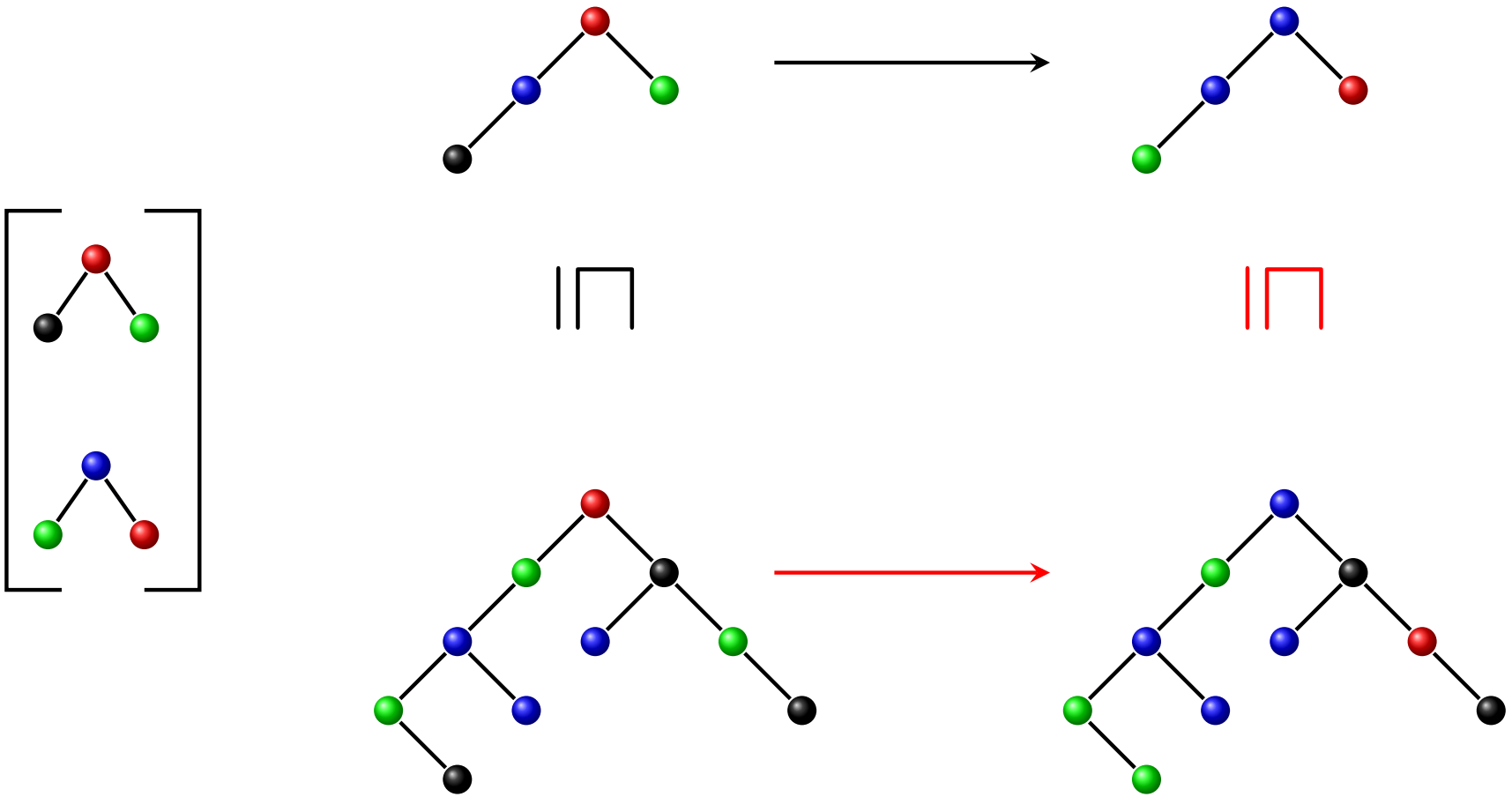




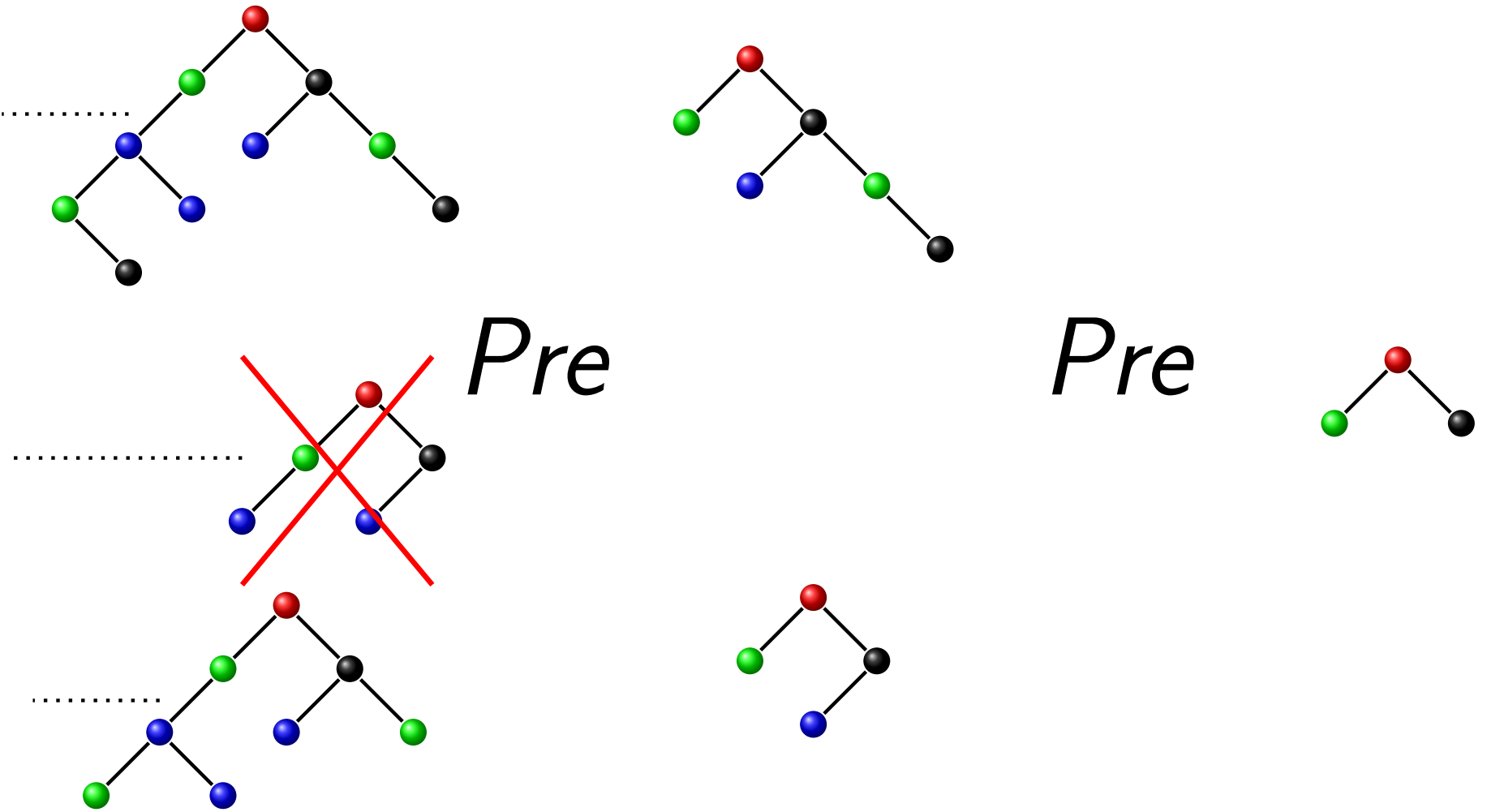
Monotonicity



Monotonicity



Backward Reachability on Trees



Conclusions

- Safety properties, ordering, and monotonicity
- Abstraction=over-approximation of universal quantifiers
- Automatic Verification of complicated protocols

Future Work

- Shape Analysis
- Parameterized Timed Systems
- Multi-threaded programs
- More advanced abstraction scheme: CEGAR