An automaton over data words that captures EMSO logic

Benedikt Bollig
LSV, ENS Cachan & CNRS

Automata, Concurrency and Timed Systems (ACTS) III

Chennai Mathematical Institute, January 27–29, 2011

Data words

finite words over $\Sigma \times D^m$

finite

label

infinite

data values

$m \geq 0$

Data words

finite words over $\Sigma \times D^m$

	finite	infinite	$m \geq 0$
	label	data values	
XML	tag	contents	$m = 1$

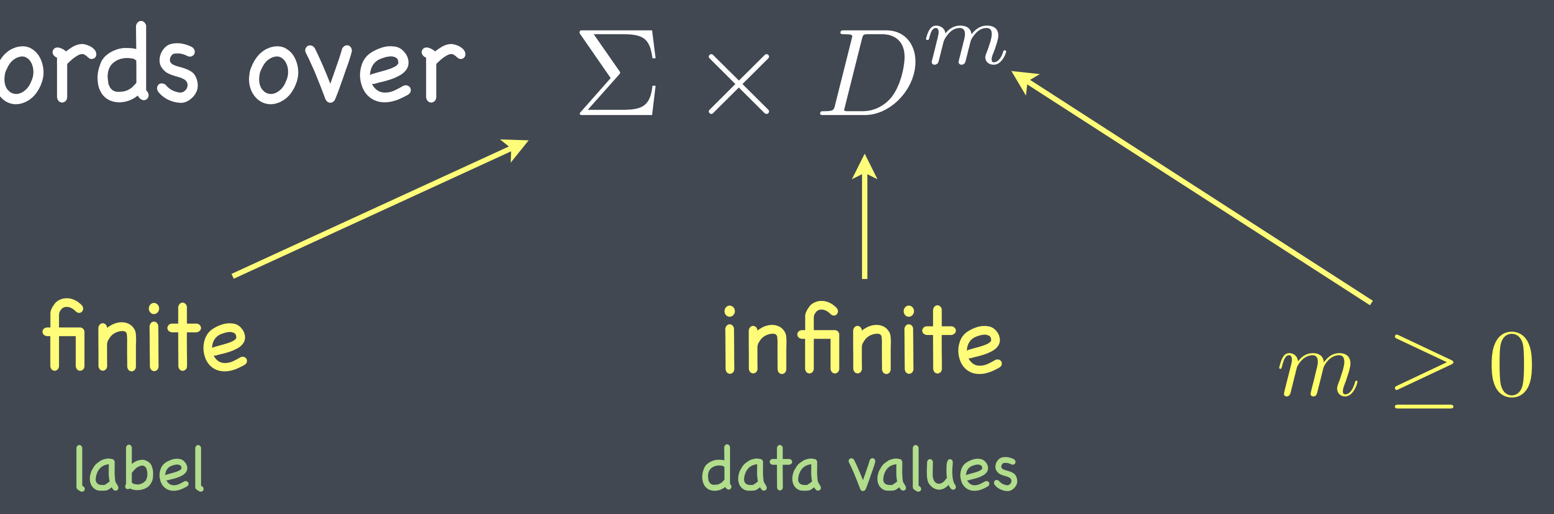
Data words

finite words over $\Sigma \times D^m$

finite	infinite	$m \geq 0$
label	data values	

XML	tag	contents	$m = 1$
time	action	$\mathbb{R}^{\geq 0}$	$m = 1$

Data words



finite words over $\Sigma \times D^m$

finite

label

infinite

data values

$m \geq 0$

XML	tag	contents	$m = 1$
time	action	$\mathbb{R}^{\geq 0}$	$m = 1$
messages	{ ! , ? , fork }	$\mathbb{N}$	$m = 2$

fork	fork	!	?	!	?
1	2	2	1	1	3
2	3	1	2	3	1

Automata vs. Logic

Automata vs. Logic

- Logic
  - Specifying properties
  - Declarative (what should happen)

Automata vs. Logic

- Logic
    - Specifying properties
    - Declarative (what should happen)
- Automata
    - Implementation model (how it should happen)
    - Tool for checking satisfiability

Automata vs. Logic

- Logic
  - Specifying properties
  - Declarative (what should happen)
- Automata
  - Implementation model (how it should happen)
  - Tool for checking satisfiability

Looking for an expressive logic with reasonable implementation model (one-way, non-deterministic)

Logic

MSO logic for data words

- $a(x)$ position x carries $a \in \Sigma$
- $x \prec_{+1} y$ y is direct successor of x
- $\varphi_1 \vee \varphi_2$ $\neg\varphi$ $\exists x \varphi$ $\exists X \varphi$ $x \in X$

MSO logic for data words

- $a(x)$ position x carries $a \in \Sigma$
- $x \prec_{+1} y$ y is direct successor of x
- $\varphi_1 \vee \varphi_2$ $\neg\varphi$ $\exists x \varphi$ $\exists X \varphi$ $x \in X$
- $d^k(x) = d^l(y)$ k-th data value at x equals l-th data value at y

  $k, l \in \{1, \ldots, m\}$

MSO logic for data words

- $a(x)$ position x carries $a \in \Sigma$
- $x \prec_{+1} y$ y is direct successor of x
- $\varphi_1 \vee \varphi_2$ $\neg\varphi$ $\exists x \varphi$ $\exists X \varphi$ $x \in X$
- $d^k(x) = d^l(y)$ ~~k-th data value at x~~ equals l-th data value at y

  ~~$k, l \in \{1, \ldots, m\}$~~

too expressive

=> restrict access to data values

=> relate positions that automaton can access

MSO logic for data words

Signature: finite set of relation symbols $\lhd$ such that

MSO logic for data words

Signature: finite set of relation symbols $\lhd$ such that

- $\lhd^w \subseteq <$

MSO logic for data words

Signature: finite set of relation symbols $\lhd$ such that

- $\lhd^w \subseteq <$

- out-degree at most 1

  for all i there is at most one j such that $i \lhd^w j$

MSO logic for data words

Signature: finite set of relation symbols $\lhd$ such that

- $\lhd^w \subseteq <$
- out-degree at most 1

  for all i there is at most one j such that $i \lhd^w j$
- in-degree at most 1

  for all i there is at most one j such that $j \lhd^w i$

MSO logic for data words

Signature: finite set of relation symbols $\lhd$ such that

- $\lhd^w \subseteq <$
- out-degree at most 1

  for all i there is at most one j such that $i \lhd^w j$
- in-degree at most 1

  for all i there is at most one j such that $j \lhd^w i$
- monotonicity

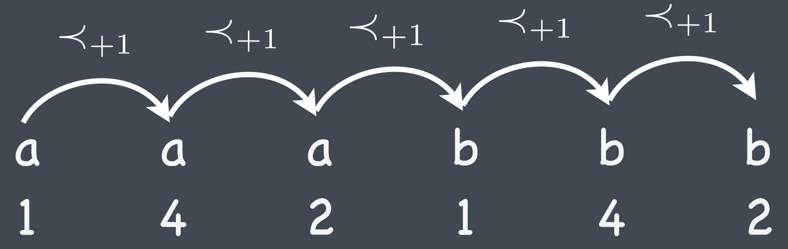
$$i \lhd^w j \;\wedge\; i' \lhd^w j' \;\wedge\; w_i = w_{i'} \;\wedge\; w_j = w_{j'}$$

$$\Longrightarrow \quad i < i' \quad \text{iff} \quad j < j'$$

MSO logic for data words

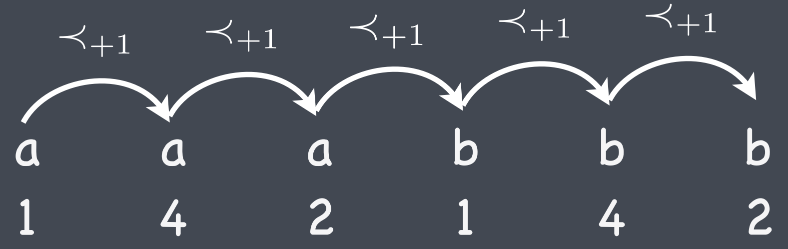
MSO logic for data words

- direct successor relation $\prec_{+1}$

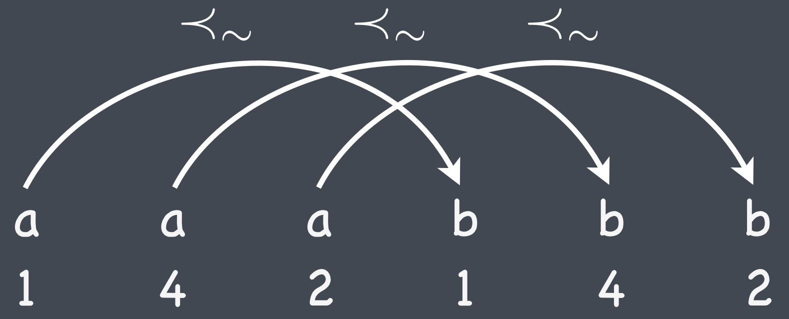


MSO logic for data words

- direct successor relation $\prec_{+1}$

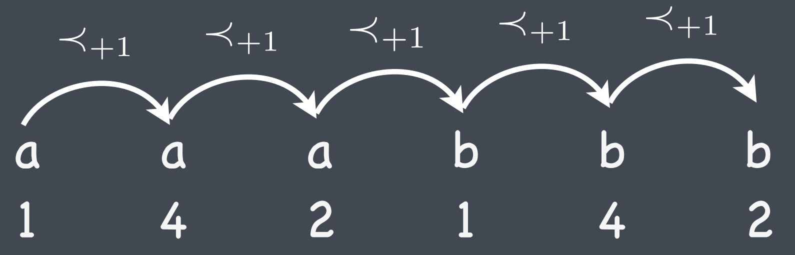


- successive positions with the same data value $\prec_{\sim}$

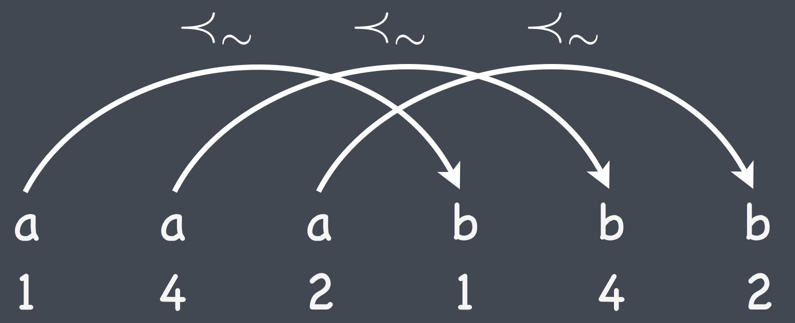


MSO logic for data words

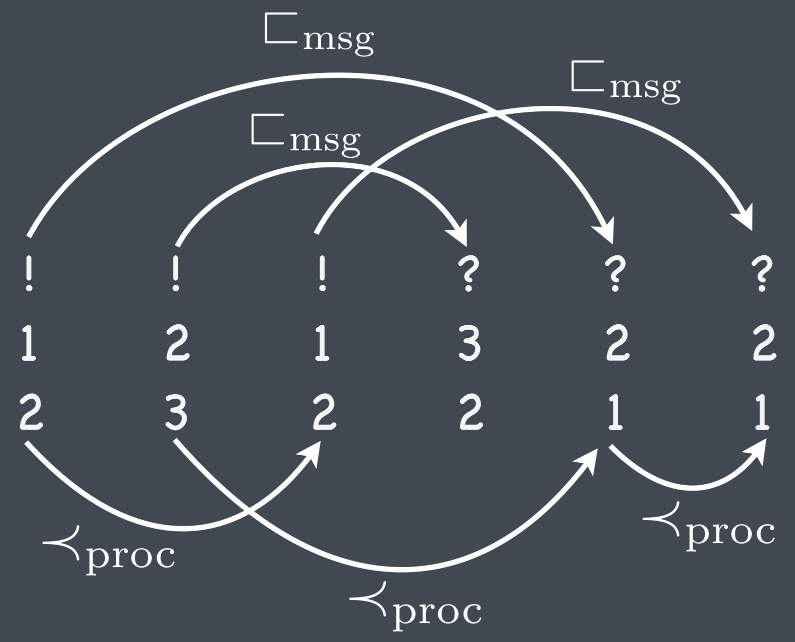
- direct successor relation $\prec_{+1}$



- successive positions with the same data value $\prec_{\sim}$



- message-passing system (m=2)



EMSO(S)

where S is any signature

- $a(x)$ position x carries $a \in \Sigma$
- $x \lhd y$ $\lhd \in$ S
- $d^k(x) = d^l(x)$ local reasoning about data values
- $x = y$ $\varphi_1 \vee \varphi_2$ $\neg\varphi$ $\exists x \varphi$ $x \in X$

EMSO(S)

where S is any signature

- $a(x)$ position x carries $a \in \Sigma$
- $x \lhd y$ $\lhd \in$ S
- $d^k(x) = d^l(x)$ local reasoning about data values
- $x = y$ $\varphi_1 \vee \varphi_2$ $\neg\varphi$ $\exists x \varphi$ $x \in X$

Formula: $\exists X_1 \ldots \exists X_n \varphi$

first-order

Example formulas (m=1)

req	req	req	ack	ack	req
4	4	2	1	4	2

Example formulas (m=1)

- there is a request that is acknowledged

$$\exists x \exists y \, (req(x) \wedge ack(y) \wedge x \precsim y)$$

req	req	req	ack	ack	req
4	4	2	1	4	2

Example formulas (m=1)

- there is a request that is acknowledged

$$\exists x \exists y\, (req(x) \wedge ack(y) \wedge x \precsim y) \quad \Dashv$$

Example formulas (m=1)

- there is a request that is acknowledged

$$\exists x \exists y \, (req(x) \wedge ack(y) \wedge x \precsim y) \quad \Dashv$$

- every request is acknowledged (before next request)

$$\forall x \exists y \, (req(x) \rightarrow ack(y) \wedge x \precsim y)$$

Example formulas (m=1)

- there is a request that is acknowledged

$$\exists x \exists y\, (req(x) \wedge ack(y) \wedge x \prec_{\sim} y) \quad \Dashv$$

- every request is acknowledged (before next request)

$$\forall x \exists y\, (req(x) \rightarrow ack(y) \wedge x \prec_{\sim} y)$$

- two successive requests are acknowledged in the order they were received

$$\forall x \forall y\, (\qquad req(x) \wedge req(y) \wedge x \prec_{+1} y$$
$$\rightarrow \exists x' \exists x'\, (ack(x') \wedge ack(y') \wedge x \prec_{\sim} x' \prec_{+1} y' \wedge y \prec_{\sim} y'))$$

Example formulas (m=1)

- there is a request that is acknowledged

$$\exists x \exists y\, (req(x) \wedge ack(y) \wedge x \prec_{\sim} y) \quad \Dashv$$

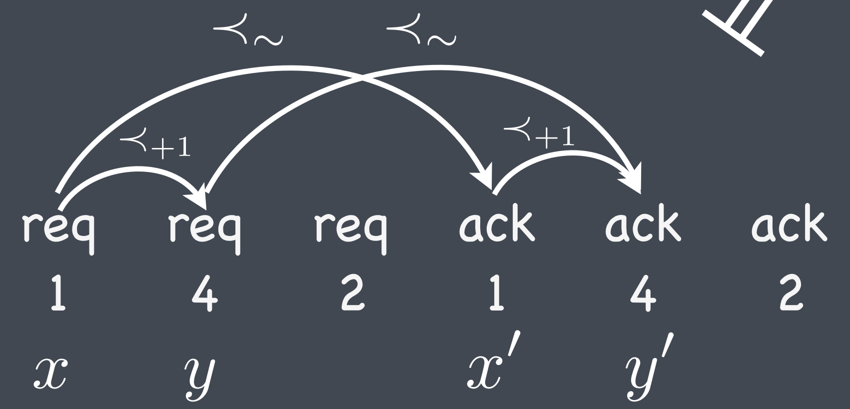
- every request is acknowledged (before next request)

$$\forall x \exists y\, (req(x) \rightarrow ack(y) \wedge x \prec_{\sim} y)$$

- two successive requests are acknowledged in the order they were received

$$\forall x \forall y\, (\qquad req(x) \wedge req(y) \wedge x \prec_{+1} y$$
$$\rightarrow \exists x' \exists x'\, (ack(x') \wedge ack(y') \wedge x \prec_{\sim} x' \prec_{+1} y' \wedge y \prec_{\sim} y'))$$

$\not\models$



Example formulas (m=1)

- there is a request that is acknowledged

$$\exists x \exists y\,(req(x) \wedge ack(y) \wedge x \precsim y) \quad \Dashv$$

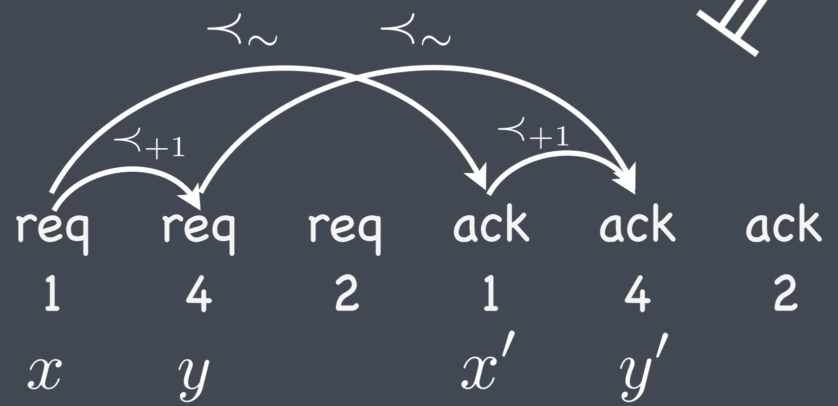
- every request is acknowledged (before next request)

$$\forall x \exists y\,(req(x) \rightarrow ack(y) \wedge x \precsim y)$$

- two successive requests are acknowledged in the order they were received

$$\forall x \forall y\,(\qquad req(x) \wedge req(y) \wedge x \prec_{+1} y$$
$$\rightarrow \exists x' \exists x'\,(ack(x') \wedge ack(y') \wedge x \precsim x' \prec_{+1} y' \wedge y \precsim y'))$$

$\Dashv$



Goal:
non-deterministic
one way automaton for
this kind of property

From Logic to Automata

[related works, most of them for m=1]

From Logic to Automata

[related works, most of them for m=1]

- MSO → register automata (restricted use of variables)
  [Bouyer 2000]

From Logic to Automata

[related works, most of them for m=1]

- MSO → register automata (restricted use of variables)
  [Bouyer 2000]
- MSO vs. two-way and pebble automata
  [Neven, Schwentick, Vianu 2004]

From Logic to Automata

[related works, most of them for m=1]

- MSO → register automata (restricted use of variables)
  [Bouyer 2000]
- MSO vs. two-way and pebble automata
  [Neven, Schwentick, Vianu 2004]
- LTL with freeze quantifier → register automata
  [Demri, Lazic 2006]

From Logic to Automata

[related works, most of them for m=1]

- MSO → register automata (restricted use of variables)
  [Bouyer 2000]
- MSO vs. two-way and pebble automata
  [Neven, Schwentick, Vianu 2004]
- LTL with freeze quantifier → register automata
  [Demri, Lazic 2006]
- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec^*_\sim)$ → data automata/class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]

From Logic to Automata

[related works, most of them for m=1]

- MSO → register automata (restricted use of variables)
  [Bouyer 2000]
- MSO vs. two-way and pebble automata
  [Neven, Schwentick, Vianu 2004]
- LTL with freeze quantifier → register automata
  [Demri, Lazic 2006]
- $\text{EMSO}^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ → data automata/class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- Regular XPath → class automata
  [Bojanczyk, Lasota 2010]

From Logic to Automata

[related works, most of them for m=1]

- MSO → register automata (restricted use of variables)
  [Bouyer 2000]
- MSO vs. two-way and pebble automata
  [Neven, Schwentick, Vianu 2004]
- LTL with freeze quantifier → register automata
  [Demri, Lazic 2006]
- $\text{EMSO}^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ → data automata/class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- Regular XPath → class automata
  [Bojanczyk, Lasota 2010]
- EMSO(S) → ? (non-deterministic, one-way)
  (e.g., $\text{EMSO}(\prec_\sim, \prec_{+1})$)

Automata

Register Automata

[Kaminski & Francez, 1994]

$$\mathcal{A} = (Q, R, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0, F initial state, set of final states

Register Automata

[Kaminski & Francez, 1994]

$$\mathcal{A} = (Q, R, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0, F initial state, set of final states
- R finite set of registers

Register Automata

[Kaminski & Francez, 1994]

$$\mathcal{A} = (Q, R, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0, F initial state, set of final states
- R finite set of registers
- transition relation:

$$(q, guard) \xrightarrow{a} (q', upd)$$

Register Automata

[Kaminski & Francez, 1994]

$$\mathcal{A} = (Q, R, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0, F initial state, set of final states
- R finite set of registers
- transition relation:

$$(q, guard) \xrightarrow{a} (q', upd)$$

$$guard \in \mathcal{B}(R)$$

current value is in 1st but not in 2nd register:

$$r_1 \wedge \neg r_2$$

Register Automata

[Kaminski & Francez, 1994]

$$\mathcal{A} = (Q, R, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0, F initial state, set of final states
- R finite set of registers
- transition relation:

$$(q, guard) \xrightarrow{a} (q', upd)$$

$$guard \in \mathcal{B}(R) \qquad upd \subseteq R$$

current value is in 1st but not in 2nd register:

$$r_1 \wedge \neg r_2$$

write current value in both registers:

$$\{r_1, r_2\}$$

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma =$ {req, ack}

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update
q_0		req , ack	q_0	
q_0		req	q_0	$\{r\}$
q_0	r	ack	q_1	
q_1		req , ack	q_1	

$L(\mathcal{A}) =$ "some request is acknowledged"

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma =$ {req, ack}

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma =$ {req, ack}

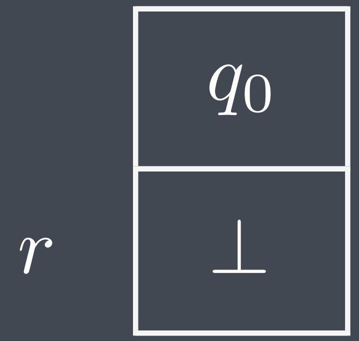
$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7



Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma =$ {req,ack}

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0
r	$\perp$	$\perp$

→

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{$req, ack$\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0
r	$\perp$	$\perp$	5
		→	⬇

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma =$ {req, ack}

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0
r	$\perp$	$\perp$	5	5
		→	⬇	→

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{$req, ack$\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0	q_0
r	$\perp$	$\perp$	5	5	4

→ ⬇ → ⬇

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{\text{req}, \text{ack}\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✓
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0	q_0	q_0
r	$\perp$	$\perp$	5	5	4	4

→ ⬇ → ⬇ →

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{\text{req}, \text{ack}\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0	q_0	q_0	q_1
r	$\perp$	$\perp$	5	5	4	4	4

→ ⬇ → ⬇ → ✔

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma =$ {req, ack}

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✓
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0	q_0	q_0	q_1	q_1
r	$\perp$	$\perp$	5	5	4	4	4	4
		→	⬇	→	⬇	→	✓	⇒

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{\text{req}, \text{ack}\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "some request is acknowledged"

	req	req	ack	req	ack	ack	ack	req
	8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0	q_0	q_0	q_1	q_1	q_1
r	$\perp$	$\perp$	5	5	4	4	4	4	4

→ ⬇ → ⬇ → ✔ ⇒ ⇒

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{$req, ack$\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A}) =$ "~~some~~ request is acknowledged"

every ?

req	req	ack	req	ack	ack	ack	req
8	5	3	4	2	4	5	7

	q_0	q_0	q_0	q_0	q_0	q_0	q_1	q_1	q_1
r	$\perp$	$\perp$	5	5	4	4	4	4	4

→ ⬇ → ⬇ → ✔ ⇒ ⇒

Register Automata

[Kaminski & Francez, 1994]

$\mathcal{A}$

$\Sigma = \{$req, ack$\}$

$D = \mathbb{N}$

$F = \{q_1\}$

source	guard	label	target	update	
q_0		req , ack	q_0		→
q_0		req	q_0	$\{r\}$	⬇
q_0	r	ack	q_1		✔
q_1		req , ack	q_1		⇒

$L(\mathcal{A})$ = "~~some~~ request is acknowledged"

every ?

=> class memory automata

		req	req	ack	req	ack	ack	ack	req
		8	5	3	4	2	4	5	7
	q_0	q_0	q_0	q_0	q_0	q_0	q_1	q_1	q_1
r	$\perp$	$\perp$	5	5	4	4	4	4	4
		→	⬇	→	⬇	→	✔	⇒	⇒

Class Memory Automata

[Björklund & Schwentick, 2007]

$$\mathcal{A} = (Q, R, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0 initial state

Class Memory Automata

[Björklund & Schwentick, 2007]

$$A = (Q, \xcancel{R}, \longrightarrow, q_0, F)$$

Q	finite set of states
q_0	initial state
~~R~~	~~finite set of registers~~

Class Memory Automata

[Björklund & Schwentick, 2007]

$$A = (Q, \cancel{R}, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0 initial state
- ~~R finite set of registers~~
- transition relation:

$$\begin{tikzcd} p \arrow[rr, bend left, "\precsim"] & q \arrow[r, "a"] & q' \end{tikzcd}$$

q is current state
p is state after last position with same data value

Class Memory Automata

[Björklund & Schwentick, 2007]

$$\mathcal{A} = (Q, \cancel{R}, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0 initial state
- ~~R finite set of registers~~
- transition relation:

$$\begin{tikzcd} p \arrow[rr, bend left, "\precsim"] & q \arrow[r, "a"] & q' \end{tikzcd}$$

q is current state
p is state after last position with same data value

$$q \xrightarrow{a} q'$$

q is current state
data value occurs for the first time

Class Memory Automata

[Björklund & Schwentick, 2007]

$$\mathcal{A} = (Q, \cancel{R}, \longrightarrow, q_0, F)$$

- Q finite set of states
- q_0 initial state
- ~~R finite set of registers~~
- transition relation:

$$\begin{tikzcd} p \arrow[rr, bend left, "\prec_\sim"] & q \arrow[r, "a"] & q' \end{tikzcd}$$

q is current state
p is state after last position with same data value

$$q \xrightarrow{a} q'$$

q is current state
data value occurs for the first time

- $F = (F_\sim, F_{+1})$ sets of final states

Class Memory Automata

[Björklund & Schwentick, 2007]

$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged,
every acknowledgment is preceded by a request, and req*ack* "

Class Memory Automata

[Björklund & Schwentick, 2007]

$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

req	req	req	req	ack	ack	ack	ack
8	5	3	4	3	4	5	8

$\boxed{q_0}$

Class Memory Automata

[Björklund & Schwentick, 2007]

$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

	req	req	req	req	ack	ack	ack	ack
	8	5	3	4	3	4	5	8
q_0	q_0							

Class Memory Automata

[Björklund & Schwentick, 2007]

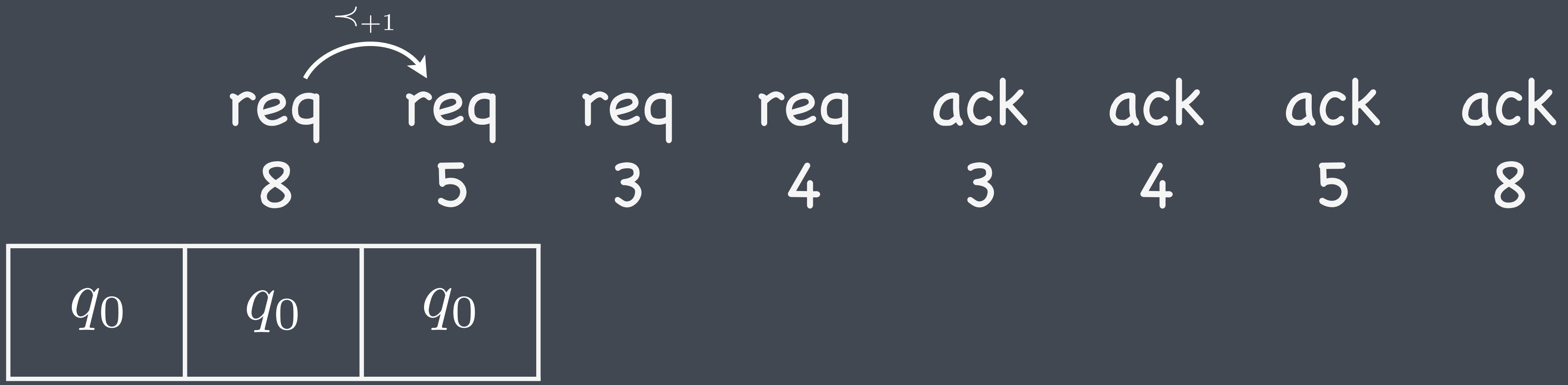
$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "



Class Memory Automata

[Björklund & Schwentick, 2007]

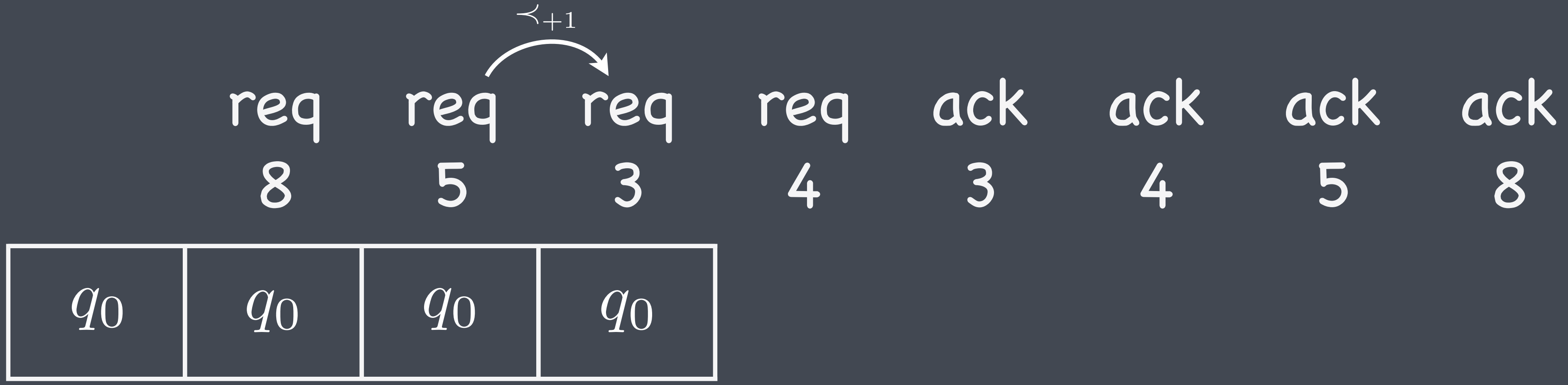
$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "



req	req	req	req	ack	ack	ack	ack
8	5	3	4	3	4	5	8

q_0	q_0	q_0	q_0

Class Memory Automata

[Björklund & Schwentick, 2007]

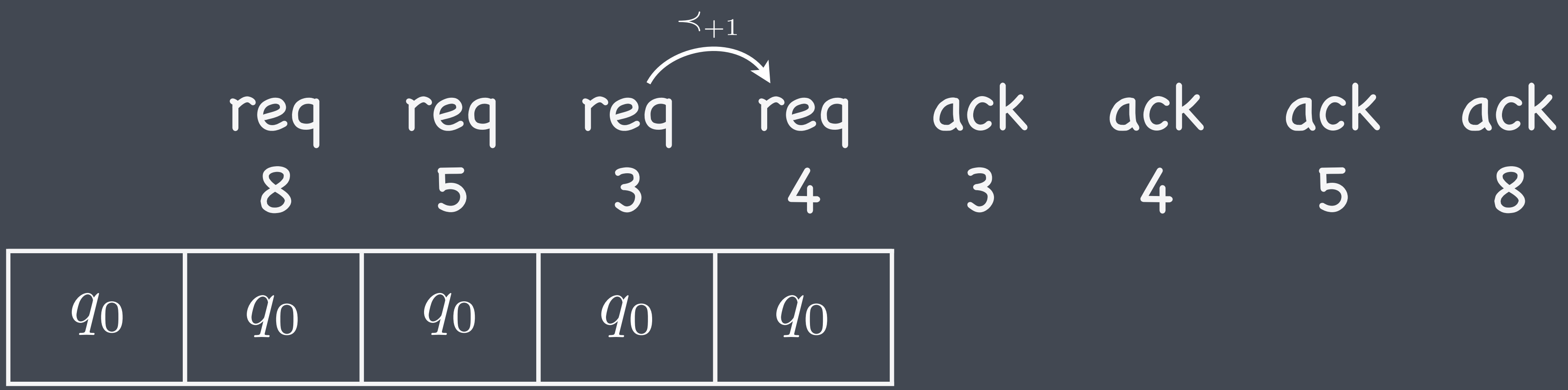
$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "



	req	req	req	req	ack	ack	ack	ack
	8	5	3	4	3	4	5	8
q_0	q_0	q_0	q_0	q_0				

Class Memory Automata

[Björklund & Schwentick, 2007]

$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

req	req	req	req	ack	ack	ack	ack
8	5	3	4	3	4	5	8

q_0	q_0	q_0	q_0	q_0	q_1

Class Memory Automata

[Björklund & Schwentick, 2007]

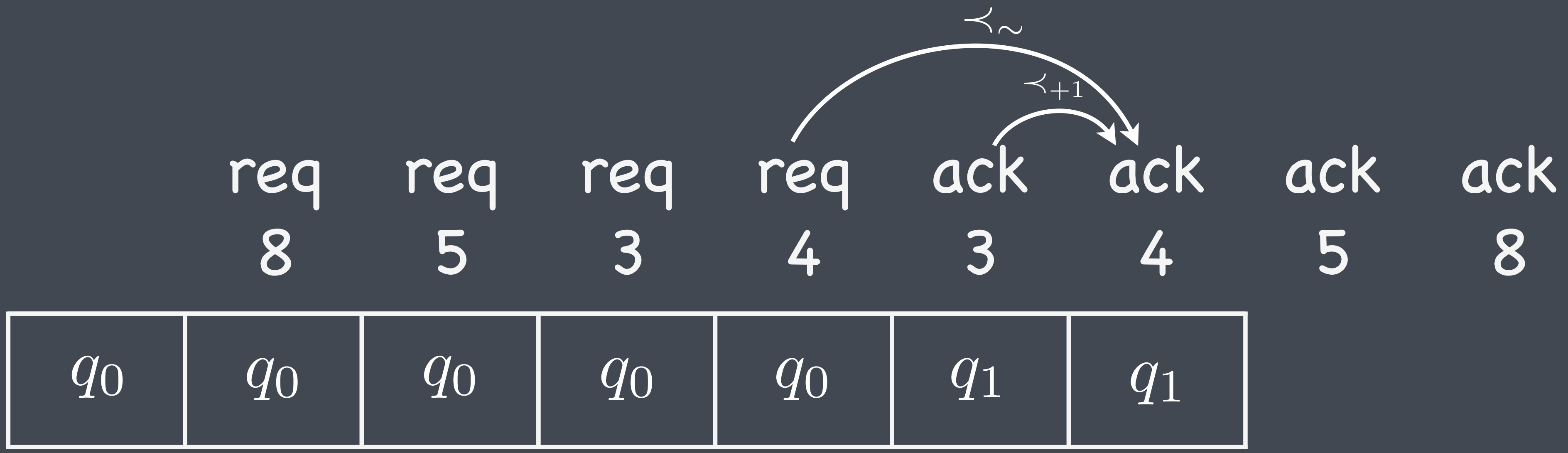
$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "



Class Memory Automata

[Björklund & Schwentick, 2007]

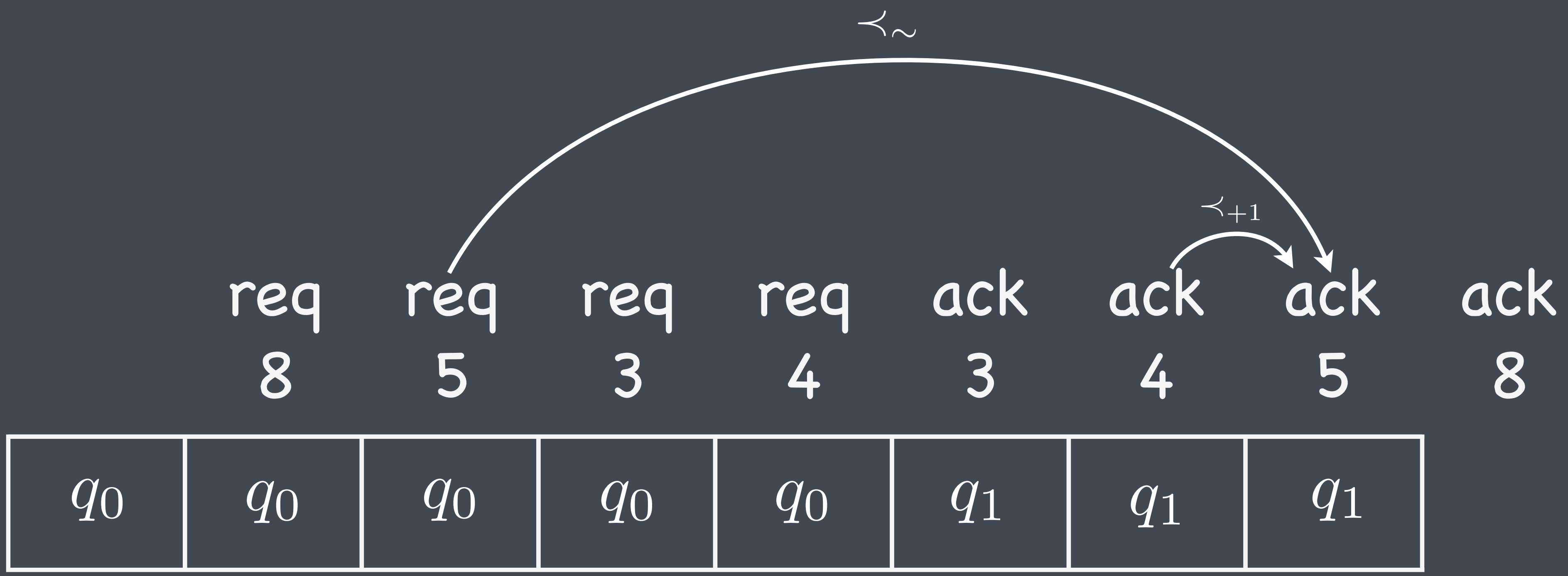
$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "



Class Memory Automata

[Björklund & Schwentick, 2007]

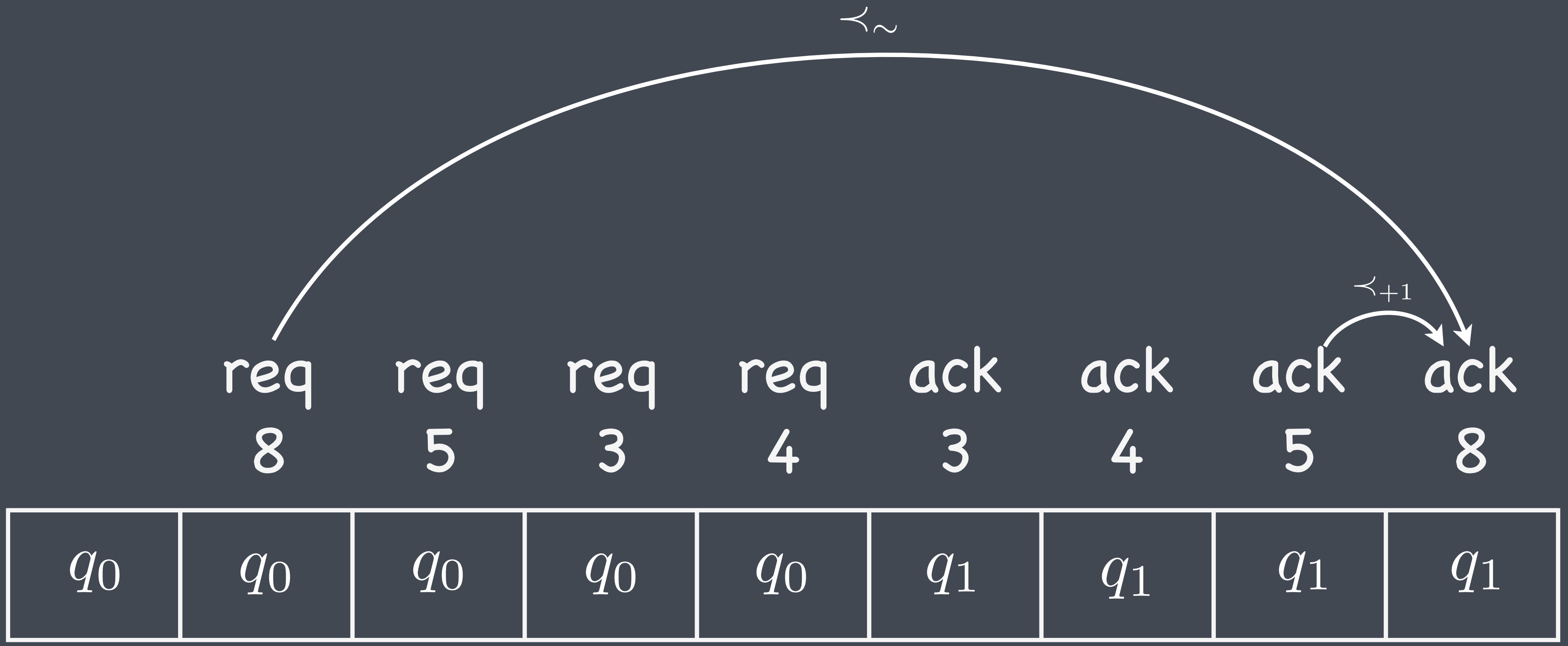
$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "



	req	req	req	req	ack	ack	ack	ack
	8	5	3	4	3	4	5	8
q_0	q_0	q_0	q_0	q_0	q_1	q_1	q_1	q_1

Class Memory Automata

[Björklund & Schwentick, 2007]

$\mathcal{A}$

$\prec_{\sim}$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_{\sim} = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received ?

req req req req ack ack ack ack

8 5 3 4 3 4 5 8

q_0	q_0	q_0	q_0	q_0	q_1	q_1	q_1	q_1

Class Memory Automata

[Björklund & Schwentick, 2007]

$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	~~guard~~	label	target	~~update~~
	q_0		req	q_0	
q_0	q_0		ack	q_1	
q_0	q_1		ack	q_1	

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received ?

=> class register automata

	req	req	req	req	ack	ack	ack	ack
	8	5	3	4	3	4	5	8
q_0	q_0	q_0	q_0	q_0	q_1	q_1	q_1	q_1

Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$A = (Q, \xcancel{R}, \longrightarrow, q_0, F)$$

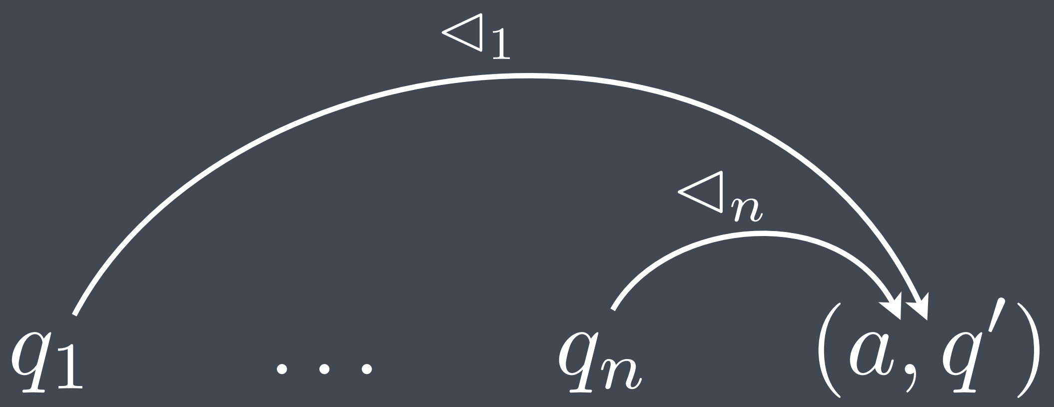
- Q finite set of states
- ~~R finite set of registers~~
- transition relation:

Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$A = (Q, \cancel{R}, \longrightarrow, q_0, F)$$

- Q finite set of states
- ~~R finite set of registers~~
- transition relation:

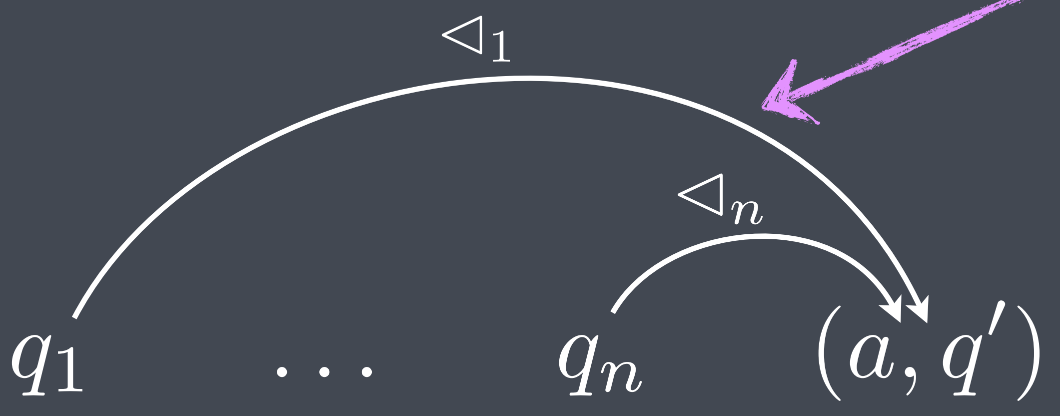


Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$A = (Q, \cancel{R}, \longrightarrow, q_0, F)$$

- Q finite set of states
- ~~R finite set of registers~~
- transition relation:



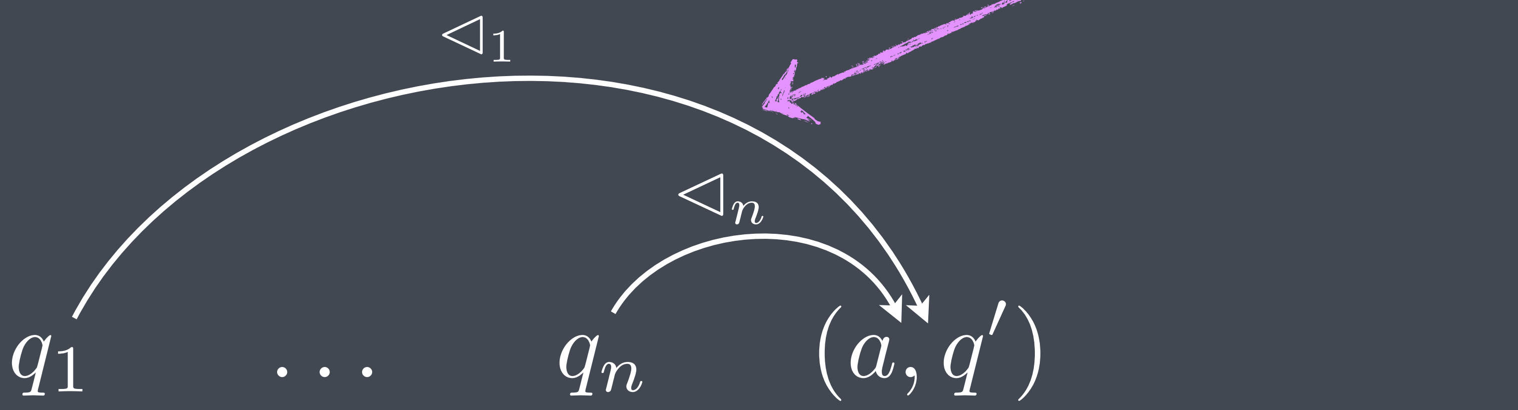
can be missing

Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$A = (Q, \cancel{R}, \longrightarrow, \cancel{q_0}, F)$$

- Q finite set of states
- ~~R finite set of registers~~
- transition relation:



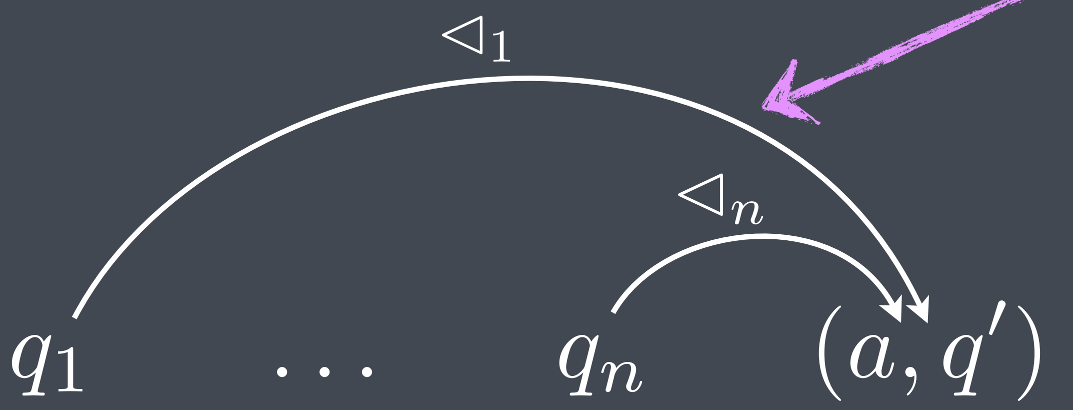
Class Register Automata

over S = $\{\lhd_1, \dots, \lhd_n\}$

$$A = (Q, \cancel{R}, \longrightarrow, \cancel{q_0}, F)$$

- Q finite set of states
- ~~R finite set of registers~~
- transition relation:

- $F = (F_{\lhd_1}, \dots, F_{\lhd_n})$



can be missing

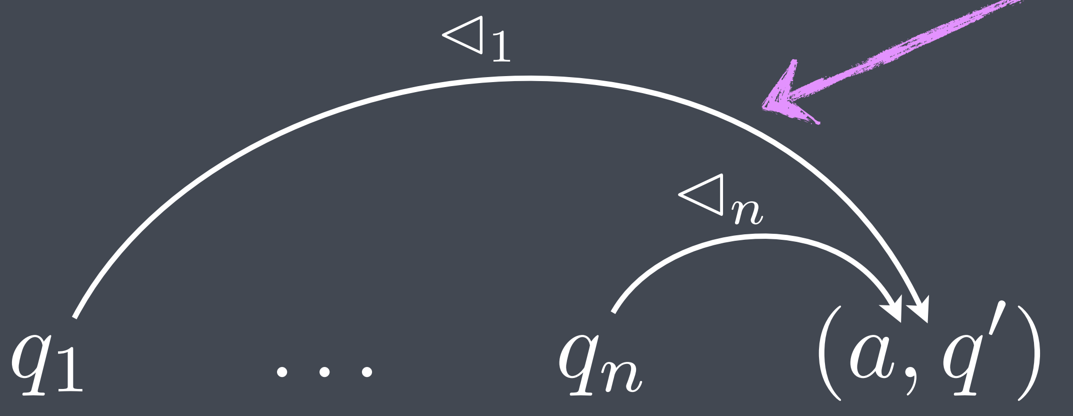
Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$\mathcal{A} = (Q, \cancel{R}, \longrightarrow, \cancel{q_0}, F)\ G$$

- Q finite set of states
- ~~R finite set of registers~~
- transition relation:

- $F = (F_{\lhd_1}, \ldots, F_{\lhd_n})$
- $G \in \mathcal{B}(\text{“}q \leq N\text{”})$



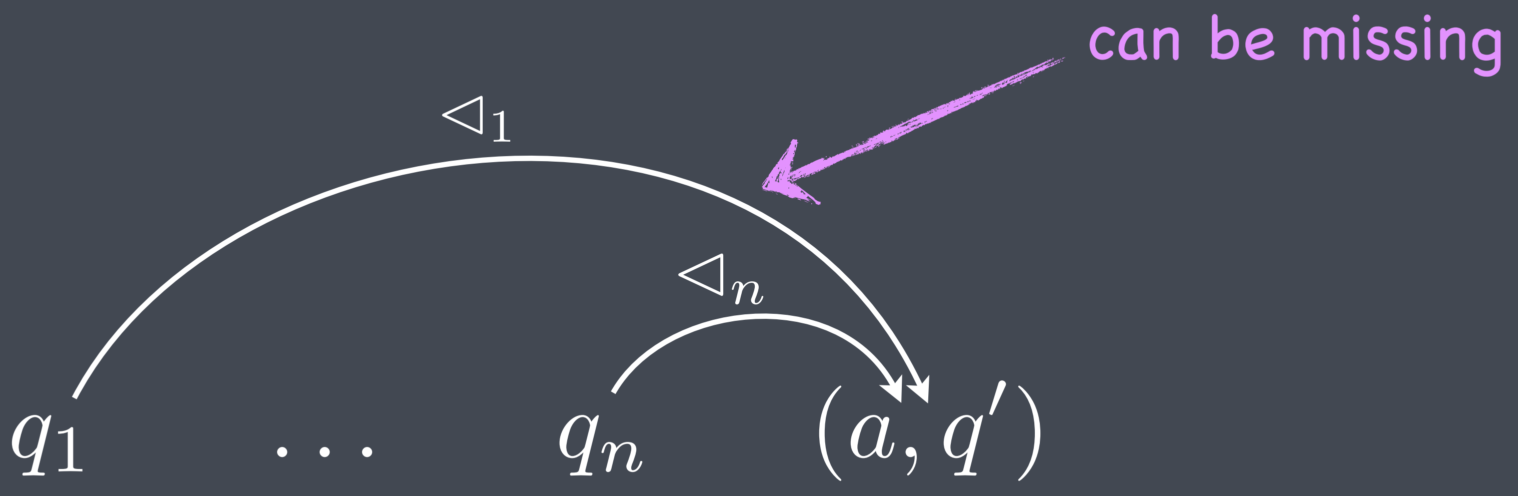
can be missing

Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$\mathcal{A} = (Q, R, \longrightarrow, \xcancel{q_0}, F)\ G$$

- Q finite set of states
- R finite set of registers
- transition relation:



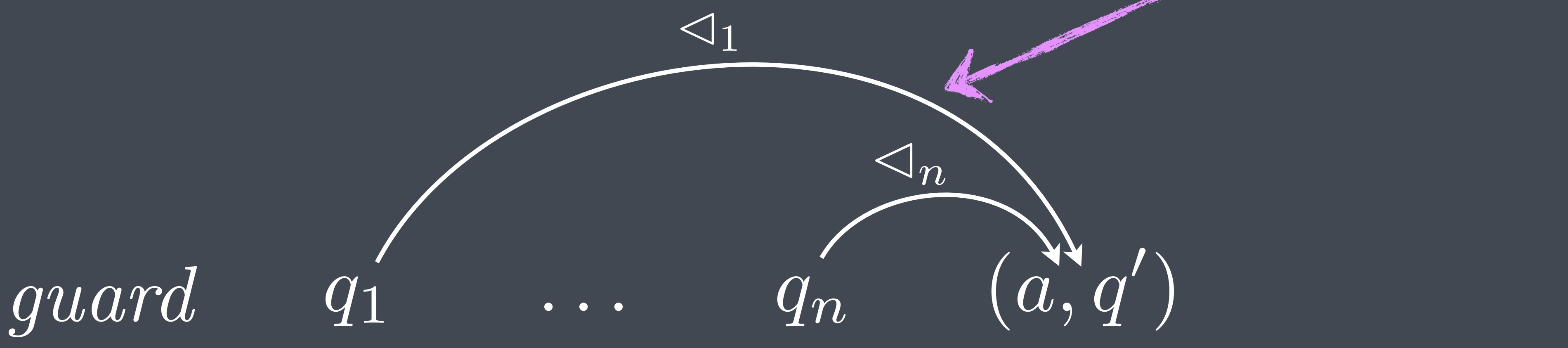
- $F = (F_{\lhd_1}, \ldots, F_{\lhd_n})$
- $G \in \mathcal{B}(\text{“}q \leq N\text{”})$

Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$\mathcal{A} = (Q, R, \longrightarrow, \cancel{q_0}, F)\ G$$

- Q finite set of states
- R finite set of registers
- transition relation:
- $F = (F_{\lhd_1}, \ldots, F_{\lhd_n})$
- $G \in \mathcal{B}(\text{“}q \leq N\text{”})$



can be missing

$guard \in \mathcal{B}(((S \times R) \cup \{1, \ldots, m\})^2)$

contents of r_1 at $\lhd_1$ = contents of r_2 at $\lhd_2$

$(\lhd_1, r_1) = (\lhd_2, r_2)$

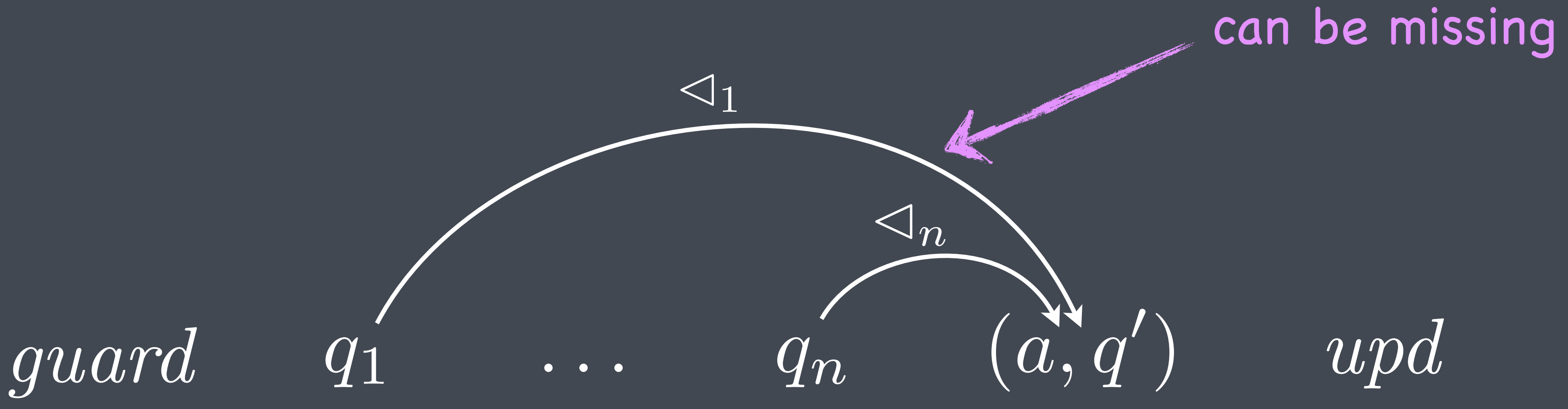
Class Register Automata

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$\mathcal{A} = (Q, R, \longrightarrow, \cancel{q_0}, F)\ G$$

- Q finite set of states
- R finite set of registers
- transition relation:

- $F = (F_{\lhd_1}, \ldots, F_{\lhd_n})$
- $G \in \mathcal{B}(\text{“}q \leq N\text{”})$



$guard \in \mathcal{B}(((S \times R) \cup \{1, \ldots, m\})^2)$

contents of r₁ at $\lhd_1$ = contents of r₂ at $\lhd_2$

$(\lhd_1, r_1) = (\lhd_2, r_2)$

$upd : R \rightharpoonup ((S \times R) \cup \{1, \ldots, m\})$

new value of r₁ := value of r₂ at $\lhd_2$

$upd(r_1) = (\lhd_2, r_2)$

Class Register Automata

$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req , d)	q_0	$r_1 := d$
	q_0		(req , d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \perp$	(ack , d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack , d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received !

req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

Class Register Automata

$\mathcal{A}$

$F_{\sim} = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$\prec_{\sim}$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_{\sim}, r_2) = \bot$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_{\sim}, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged,
every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged
in the order they are received !

req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state
r_1
r_2

Class Register Automata

$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \perp$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged,
every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged
in the order they are received !

req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0
r_1	8
r_2	$\perp$

Class Register Automata

$\mathcal{A}$

$F_\sim = \{q_1\}$

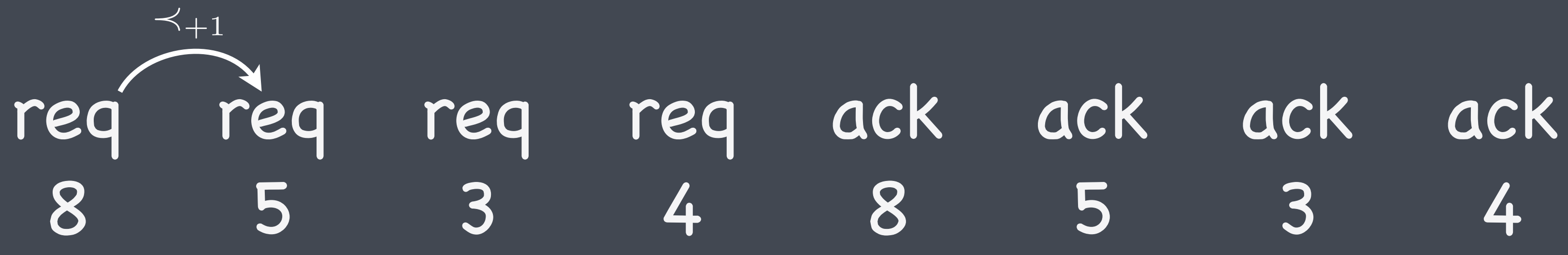
$F_{+1} = \{q_1\}$

$G = true$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d$ $\quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \perp$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged,
every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged
in the order they are received !



req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0
r_1	8	5
r_2	$\perp$	8

Class Register Automata

$\mathcal{A}$

$F_{\sim} = \{q_1\}$

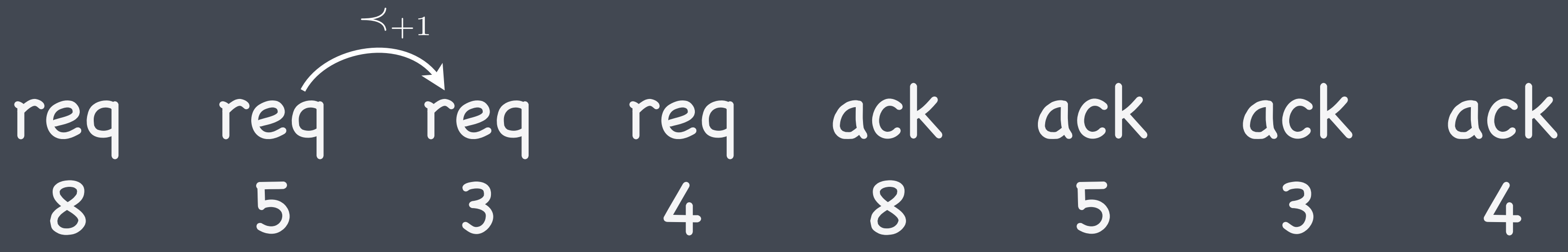
$F_{+1} = \{q_1\}$

$G = true$

$\prec_{\sim}$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_{\sim}, r_2) = \bot$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_{\sim}, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received !



req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0	q_0
r_1	8	5	3
r_2	$\bot$	8	5

Class Register Automata

$\mathcal{A}$

$F_{\sim} = \{q_1\}$

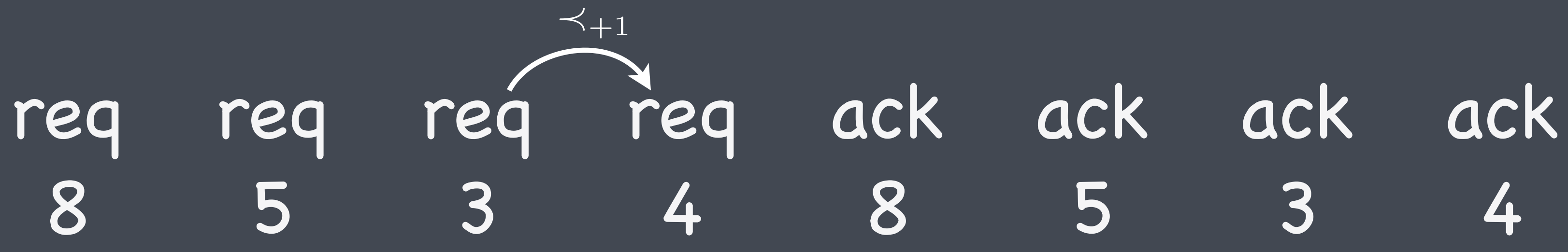
$F_{+1} = \{q_1\}$

$G = true$

$\prec_{\sim}$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_{\sim}, r_2) = \perp$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_{\sim}, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received !



req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0	q_0	q_0
r_1	8	5	3	4
r_2	$\perp$	8	5	3

Class Register Automata

$\mathcal{A}$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \bot$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

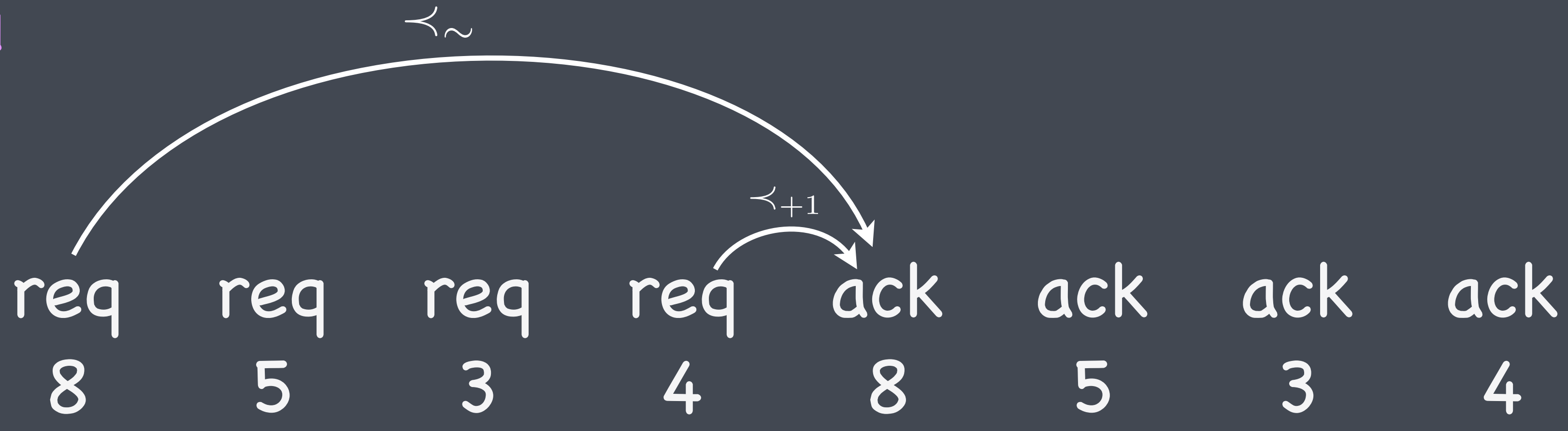
$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged,
every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged
in the order they are received !



req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0	q_0	q_0	q_1
r_1	8	5	3	4	8
r_2	$\bot$	8	5	3	$\bot$

Class Register Automata

$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d$ $\quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \bot$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received !

req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0	q_0	q_0	q_1	q_1
r_1	8	5	3	4	8	5
r_2	$\bot$	8	5	3	$\bot$	$\bot$

Class Register Automata

$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \bot$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged,
every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged
in the order they are received !

req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0	q_0	q_0	q_1	q_1	q_1
r_1	8	5	3	4	8	5	3
r_2	$\bot$	8	5	3	$\bot$	$\bot$	$\bot$

Class Register Automata

$\mathcal{A}$

$F_\sim = \{q_1\}$

$F_{+1} = \{q_1\}$

$G = true$

$\prec_\sim$	$\prec_{+1}$	guard	input	target	update
			(req, d)	q_0	$r_1 := d$
	q_0		(req, d)	q_0	$r_1 := d \quad r_2 := (\prec_{+1}, r_1)$
q_0	q_0	$(\prec_\sim, r_2) = \perp$	(ack, d)	q_1	$r_1 := d$
q_0	q_1	$(\prec_\sim, r_2) = (\prec_{+1}, r_1)$	(ack, d)	q_1	$r_1 := d$

$L(\mathcal{A}) =$ " every process sends one request, which is acknowledged, every acknowledgment is preceded by a request, and req*ack* "

... and requests are acknowledged in the order they are received !

req	req	req	req	ack	ack	ack	ack
8	5	3	4	8	5	3	4

state	q_0	q_0	q_0	q_0	q_1	q_1	q_1	q_1
r_1	8	5	3	4	8	5	3	4
r_2	$\perp$	8	5	3	$\perp$	$\perp$	$\perp$	$\perp$

From Logic to Automata

From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]

From Logic to Automata

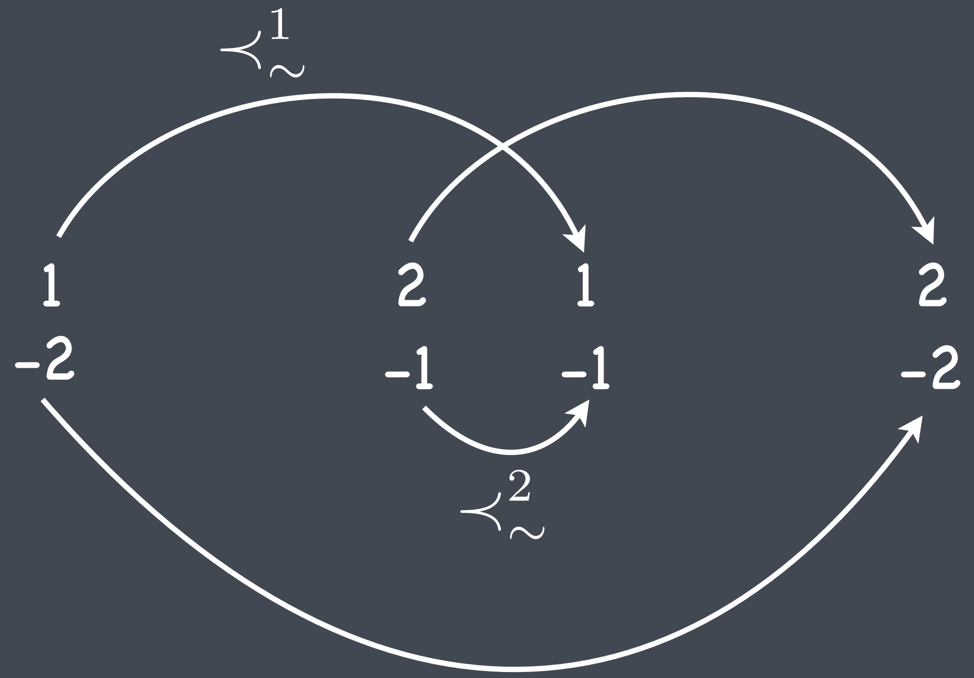
- $\mathrm{EMSO}^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?

From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?
- FO$(\prec_\sim^1, \prec_\sim^2)$ $\not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$

From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?
- FO$(\prec_\sim^1, \prec_\sim^2)$ $\not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$

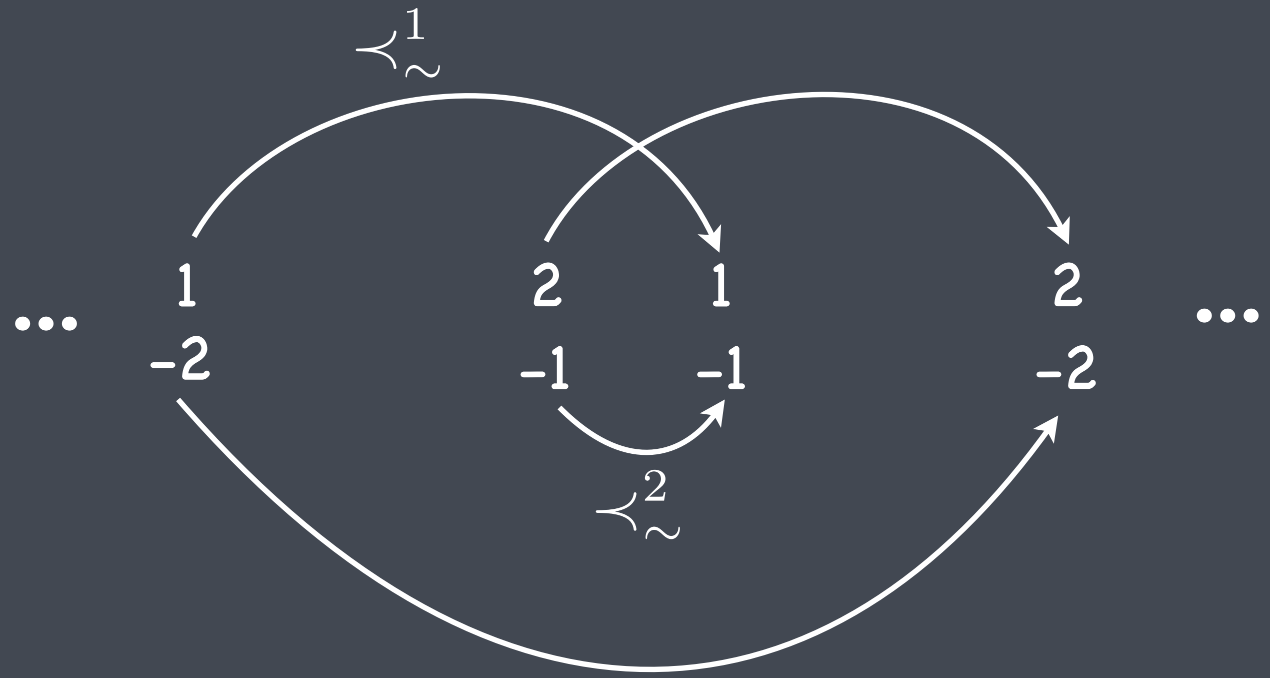


From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]

- EMSO(S) → class register automata ?

- FO$(\prec_\sim^1, \prec_\sim^2)$ $\not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$

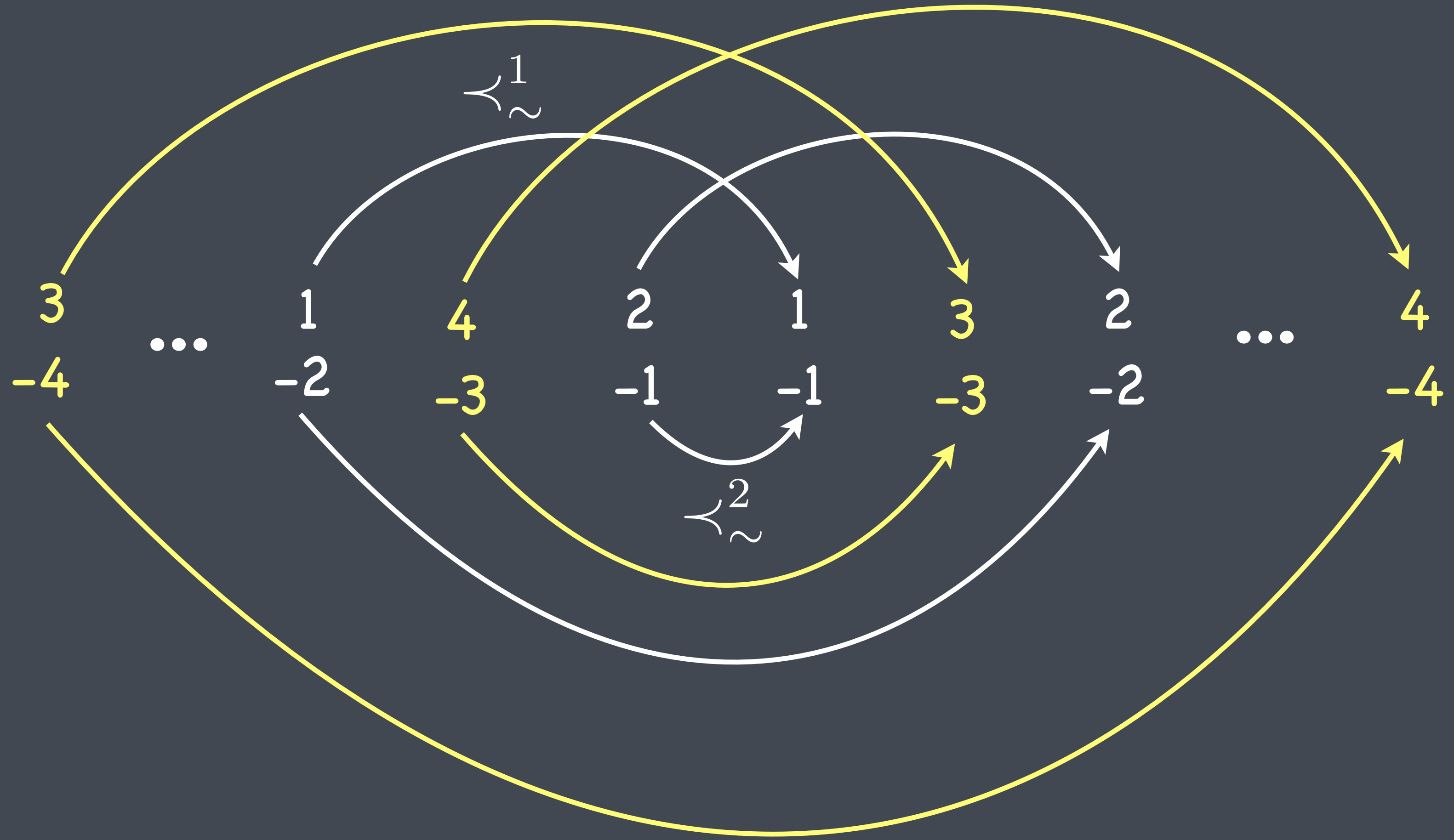


From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]

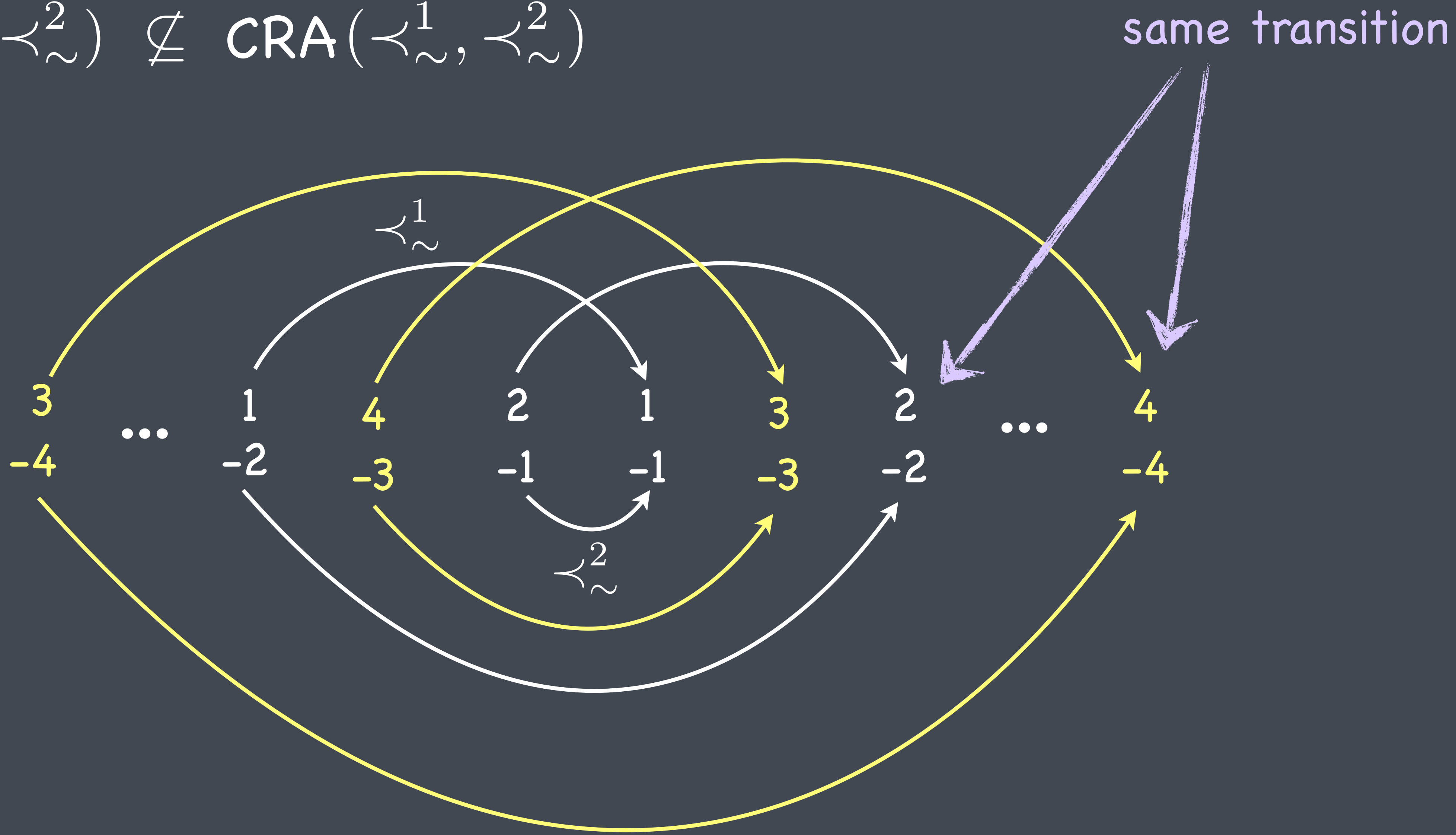
- EMSO(S) → class register automata ?

- FO$(\prec_\sim^1, \prec_\sim^2) \not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$



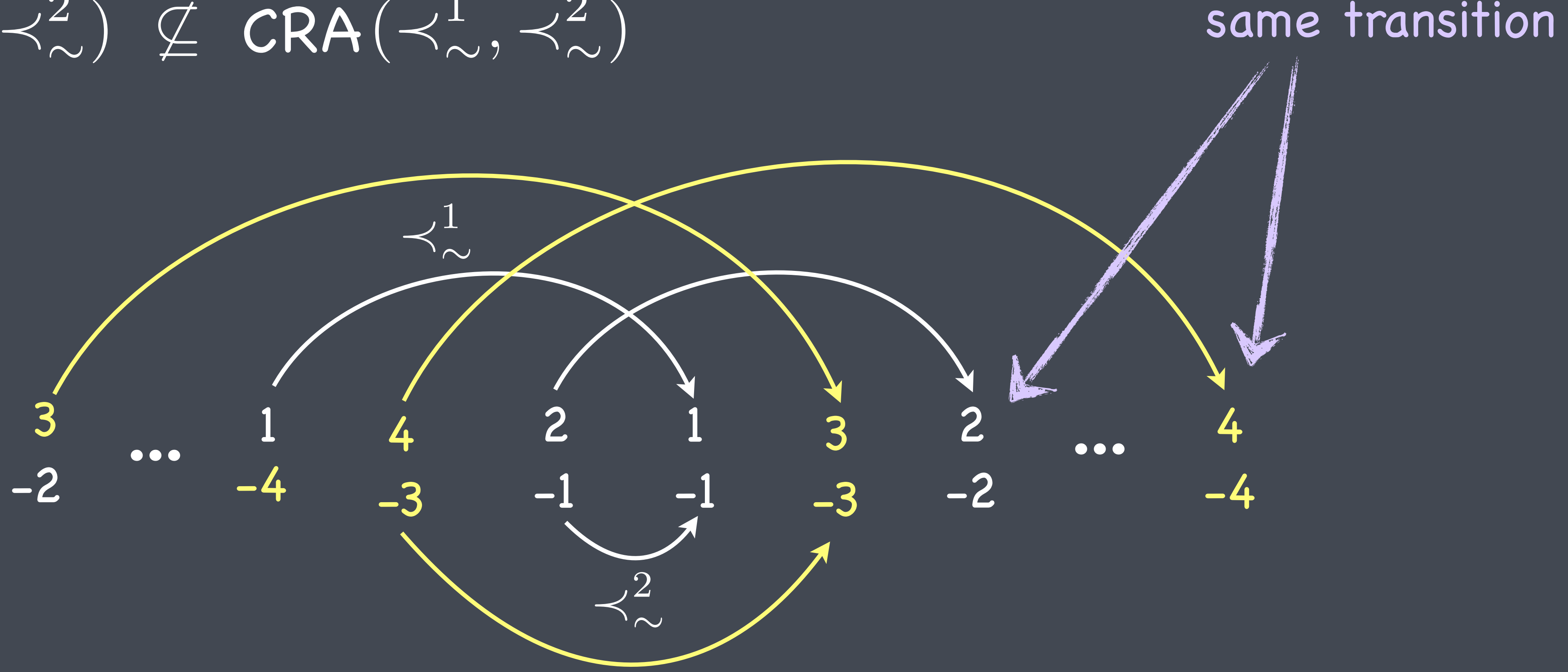
From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?
- FO$(\prec_\sim^1, \prec_\sim^2) \not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$



From Logic to Automata

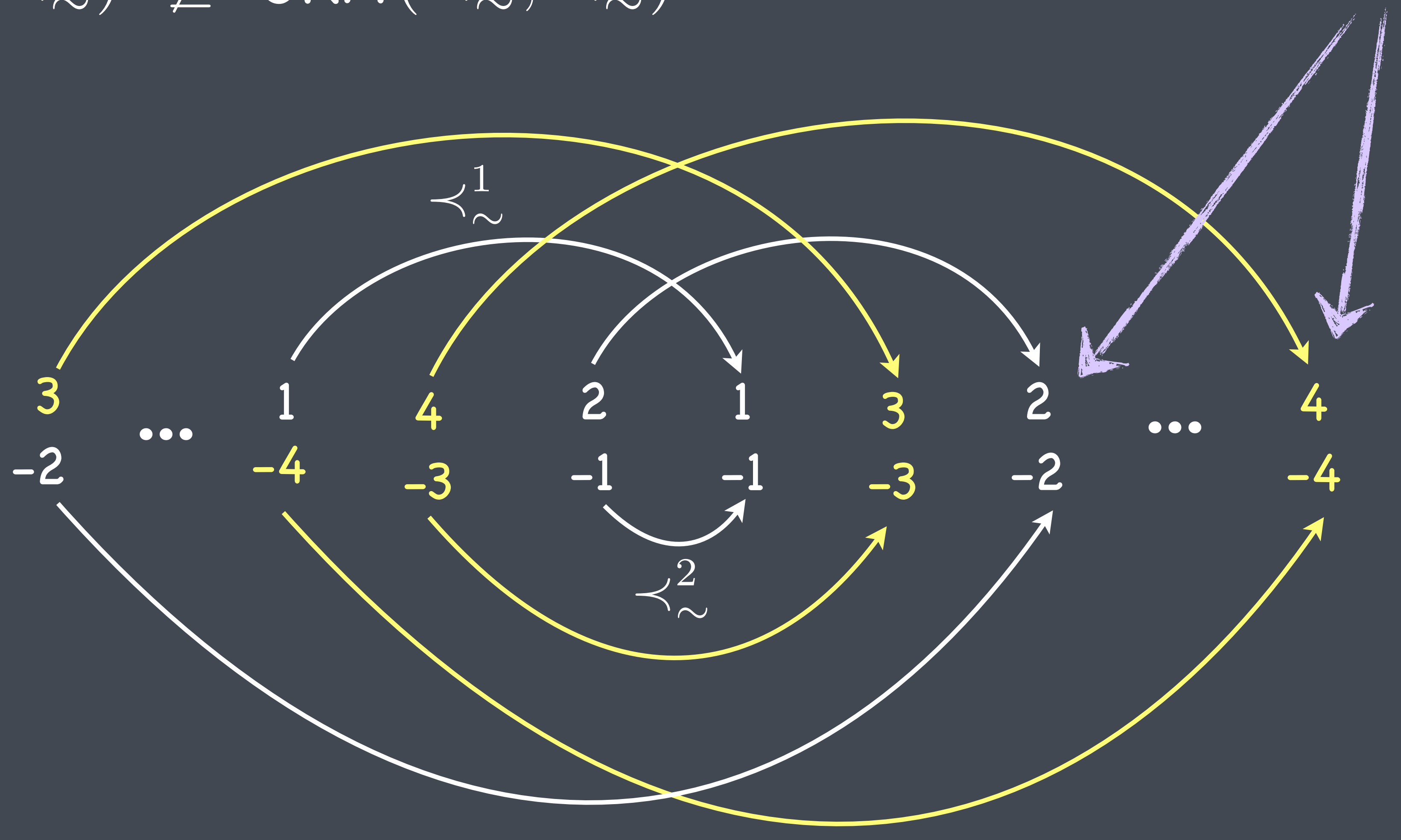
- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec^*_\sim)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?
- FO$(\prec^1_\sim, \prec^2_\sim) \not\subseteq$ CRA$(\prec^1_\sim, \prec^2_\sim)$



From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?
- FO$(\prec_\sim^1, \prec_\sim^2) \not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$

same transition

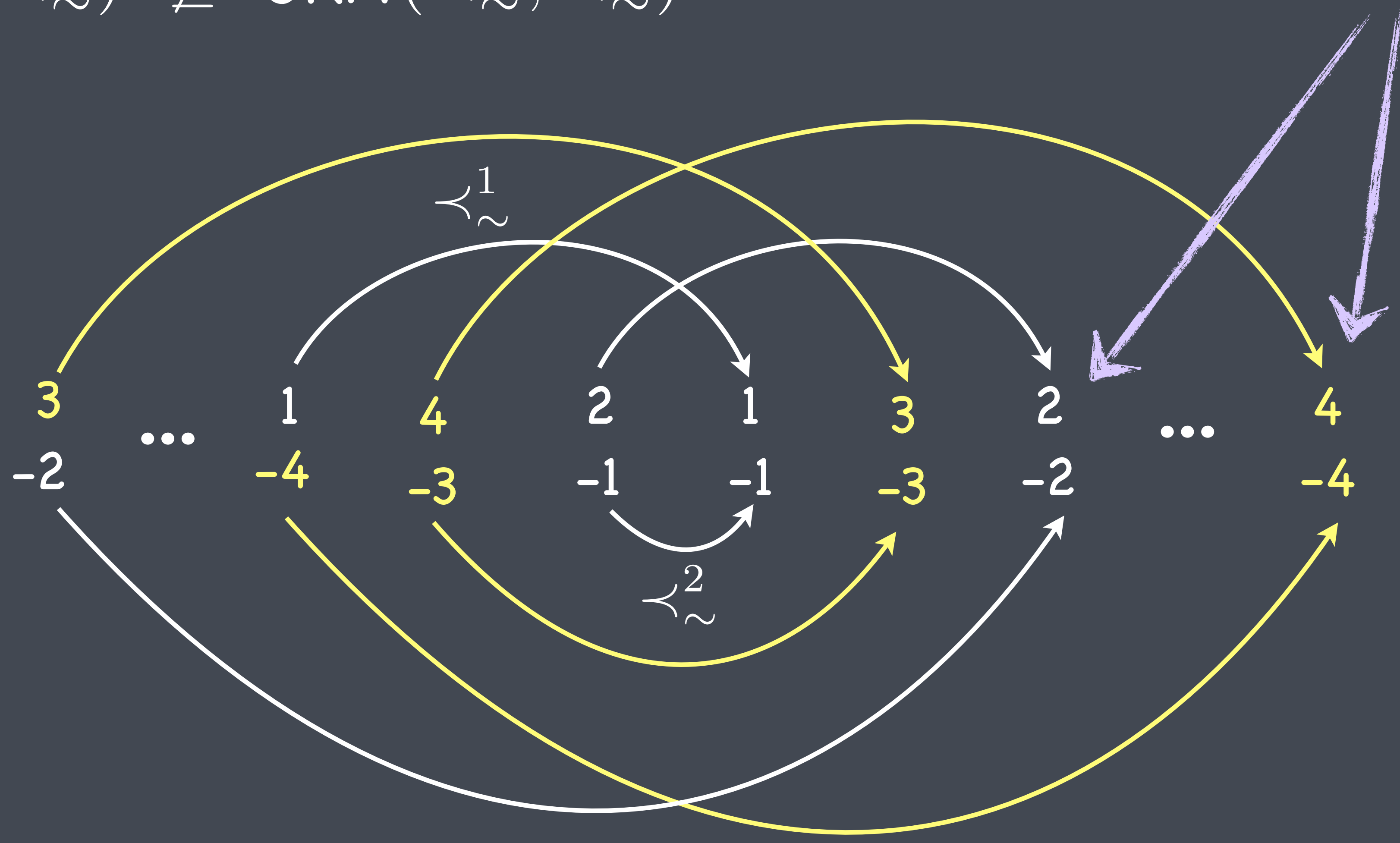


From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]

- EMSO(S) → class register automata ?

- FO$(\prec_\sim^1, \prec_\sim^2)$ $\not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$

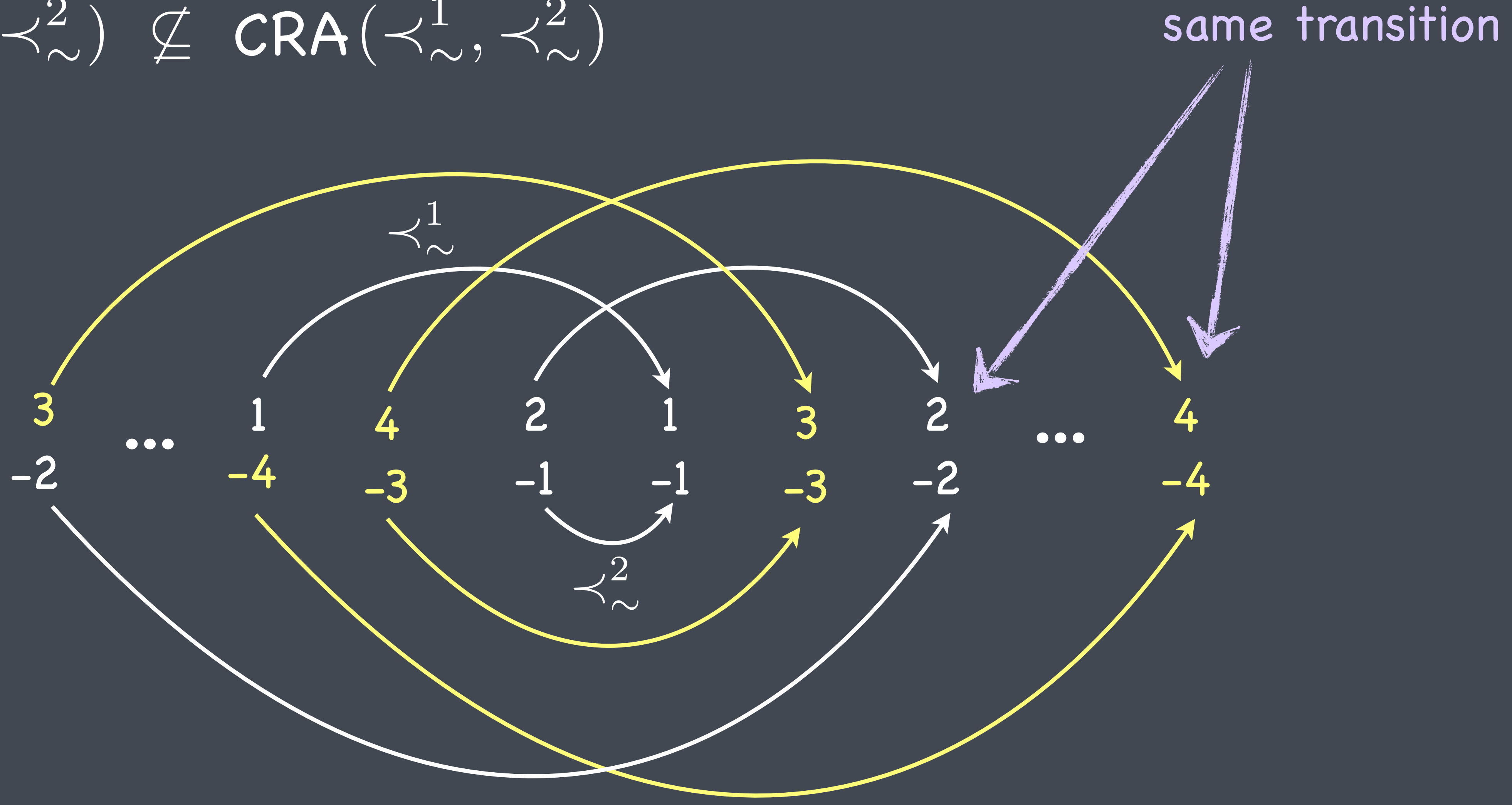


same transition

=> CRA cannot detect cycles

From Logic to Automata

- EMSO$^2(\prec_\sim, \prec_{+1}, <, \prec_\sim^*)$ = class memory automata
  [Bojanczyk, David, Muscholl, Schwentick, Segoufin 2006]
  [Björklund, Schwentick 2007]
- EMSO(S) → class register automata ?
- FO$(\prec_\sim^1, \prec_\sim^2)$ $\not\subseteq$ CRA$(\prec_\sim^1, \prec_\sim^2)$



=> CRA cannot detect cycles

=> guess data values!

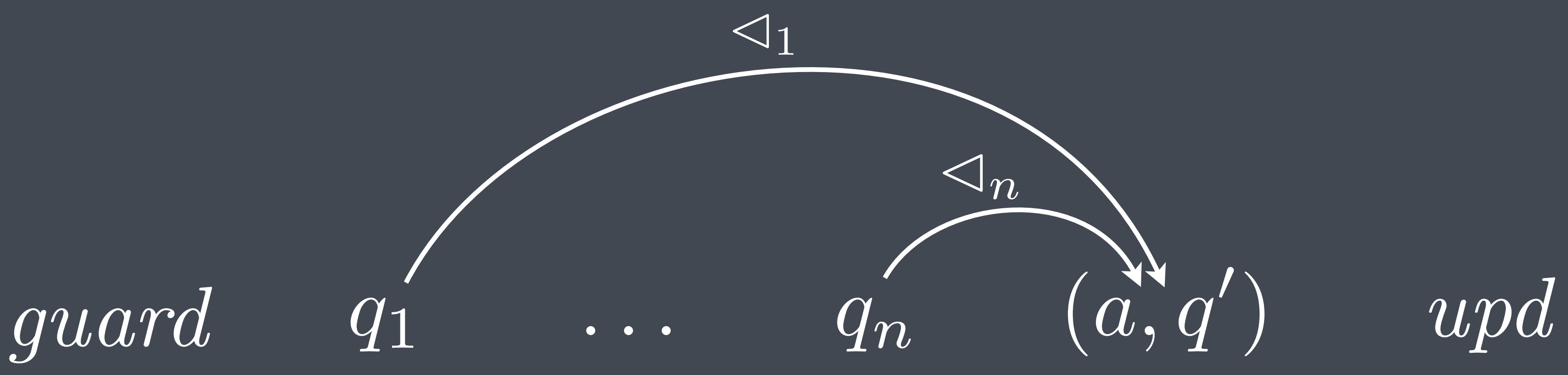
Class Register Automata (with guess)

over S = $\{\lhd_1, \ldots, \lhd_n\}$

$$\mathcal{A} = (Q, R, \longrightarrow, F, G)$$

- Q finite set of states
- R finite set of registers
- transition relation:

- $F = (F_{\lhd_1}, \ldots, F_{\lhd_n})$
- $G \in \mathcal{B}(\text{“}q \leq N\text{”})$



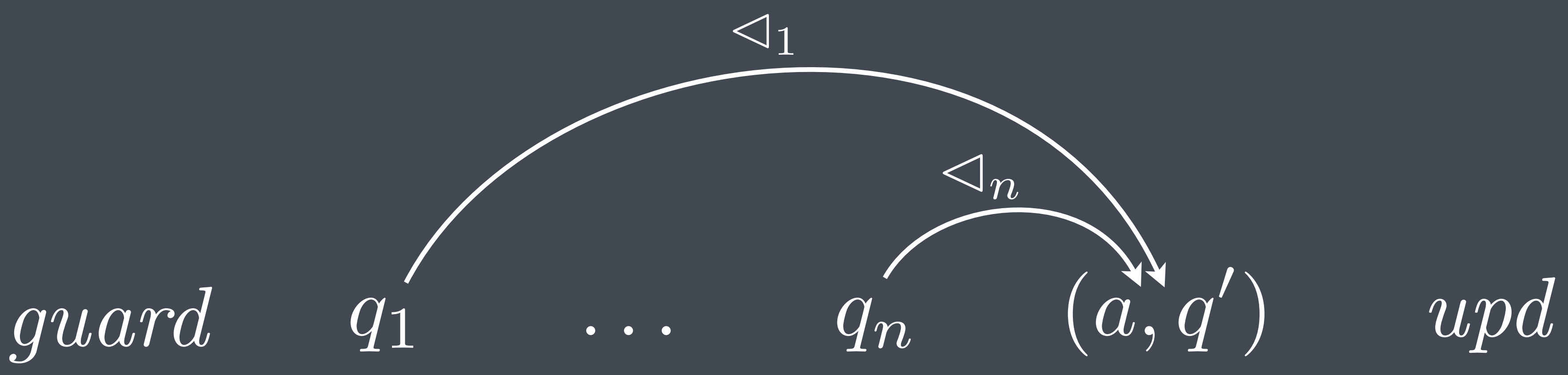
$upd : R \rightharpoonup ((S \times R) \cup \{1, \ldots, m\})$

Class Register Automata (with guess)

over S = $\{\lhd_1, \dots, \lhd_n\}$

$$\mathcal{A} = (Q, R, \longrightarrow, F, G)$$

- Q finite set of states
- R finite set of registers
- transition relation:



$upd : R \rightharpoonup ((S \times R) \cup \{1, \dots, m\})$

$\cup$ {guess}

- $F = (F_{\lhd_1}, \dots, F_{\lhd_n})$
- $G \in \mathcal{B}(\text{“}q \leq N\text{”})$

guessing register automata:
[Kaminski, Zeitlin 2010]

Class Register Automata (with guess)

Theorem: For every signature S, EMSO(S) $\subseteq$ ${}_{g}$CRA(S).

Class Register Automata (with guess)

Theorem: For every signature S, EMSO(S) $\subseteq$ $_g$CRA(S).

Proof:

- Use Hanf`s Theorem (1965):
  normal form of first-order formulas

Class Register Automata (with guess)

Theorem: For every signature S, EMSO(S) $\subseteq$ $_g$CRA(S).

Proof:

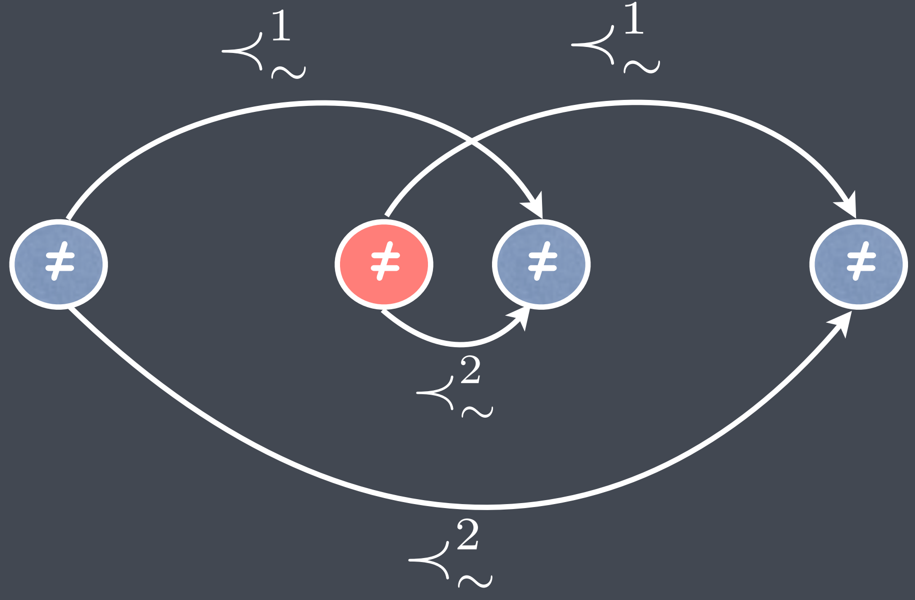
- Use Hanf's Theorem (1965): normal form of first-order formulas
- Sufficient to detect local patterns and count them up to some threshold

Class Register Automata (with guess)

Theorem: For every signature S, EMSO(S) ⊆ $_{g}$CRA(S).

Proof:

- Use Hanf's Theorem (1965): normal form of first-order formulas
- Sufficient to detect local patterns and count them up to some threshold
- Local pattern:

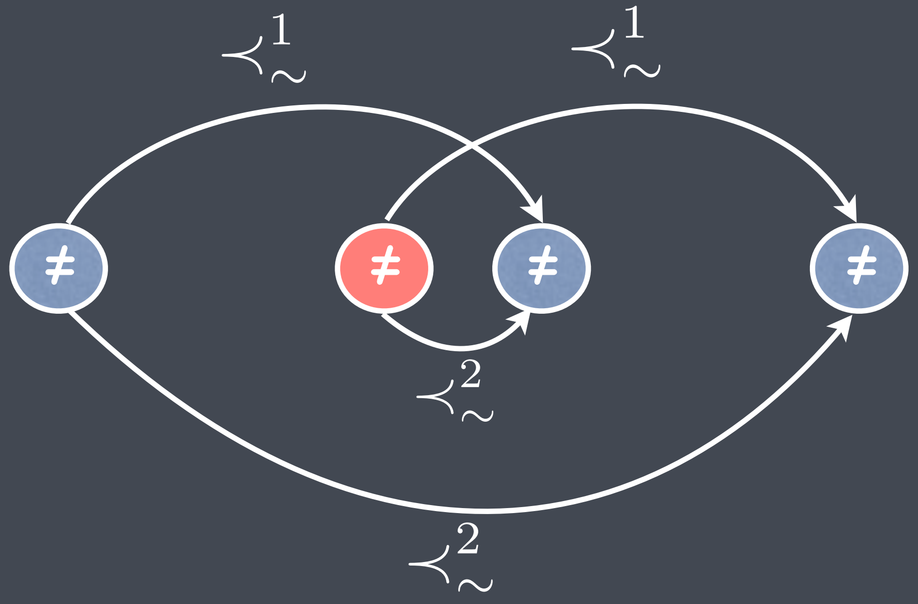


Class Register Automata (with guess)

Theorem: For every signature S, EMSO(S) $\subseteq$ ${}_g$CRA(S).

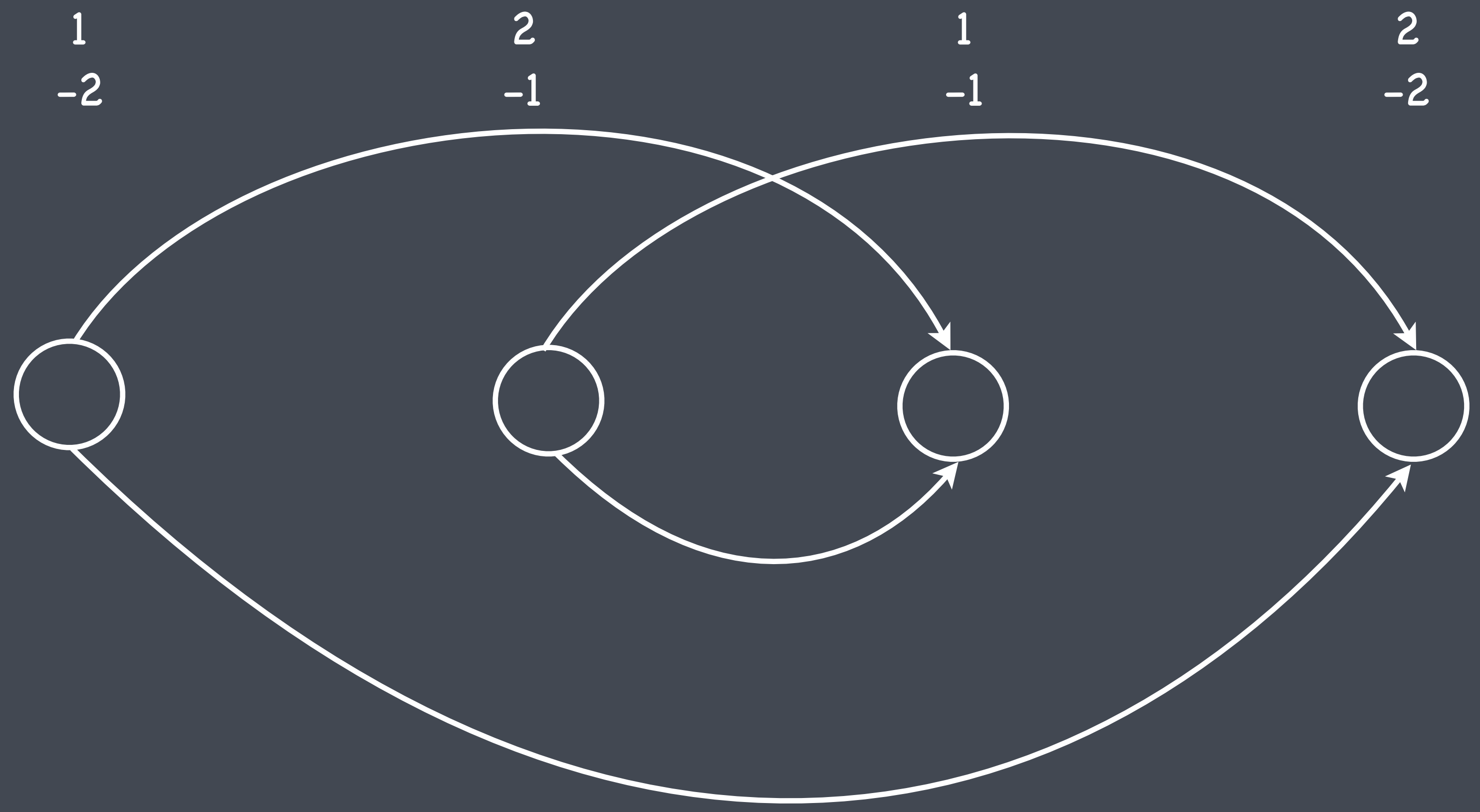
Proof:

- Use Hanf's Theorem (1965): normal form of first-order formulas
- Sufficient to detect local patterns and count them up to some threshold
- Local pattern:

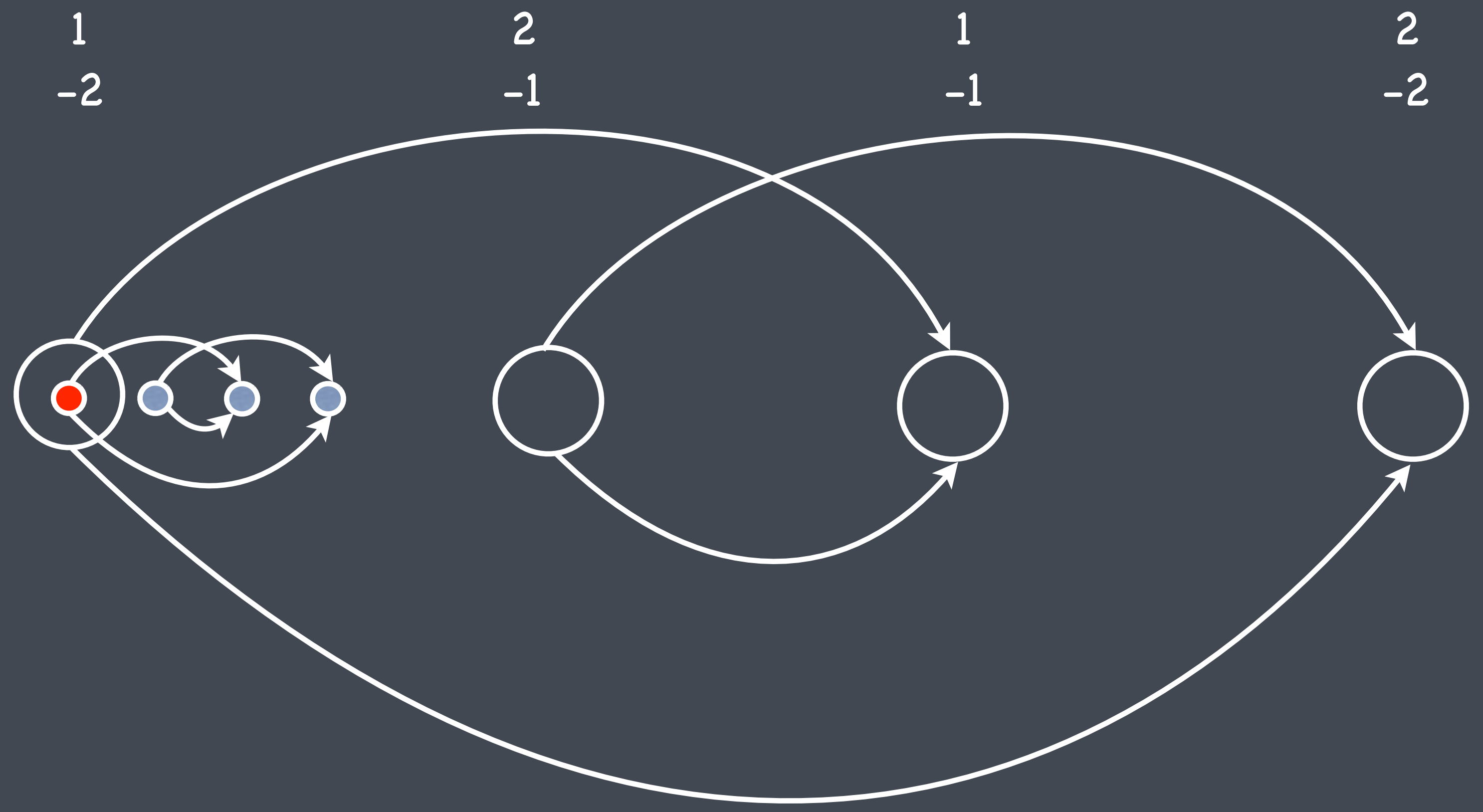


- => Build class register automaton that detects spheres

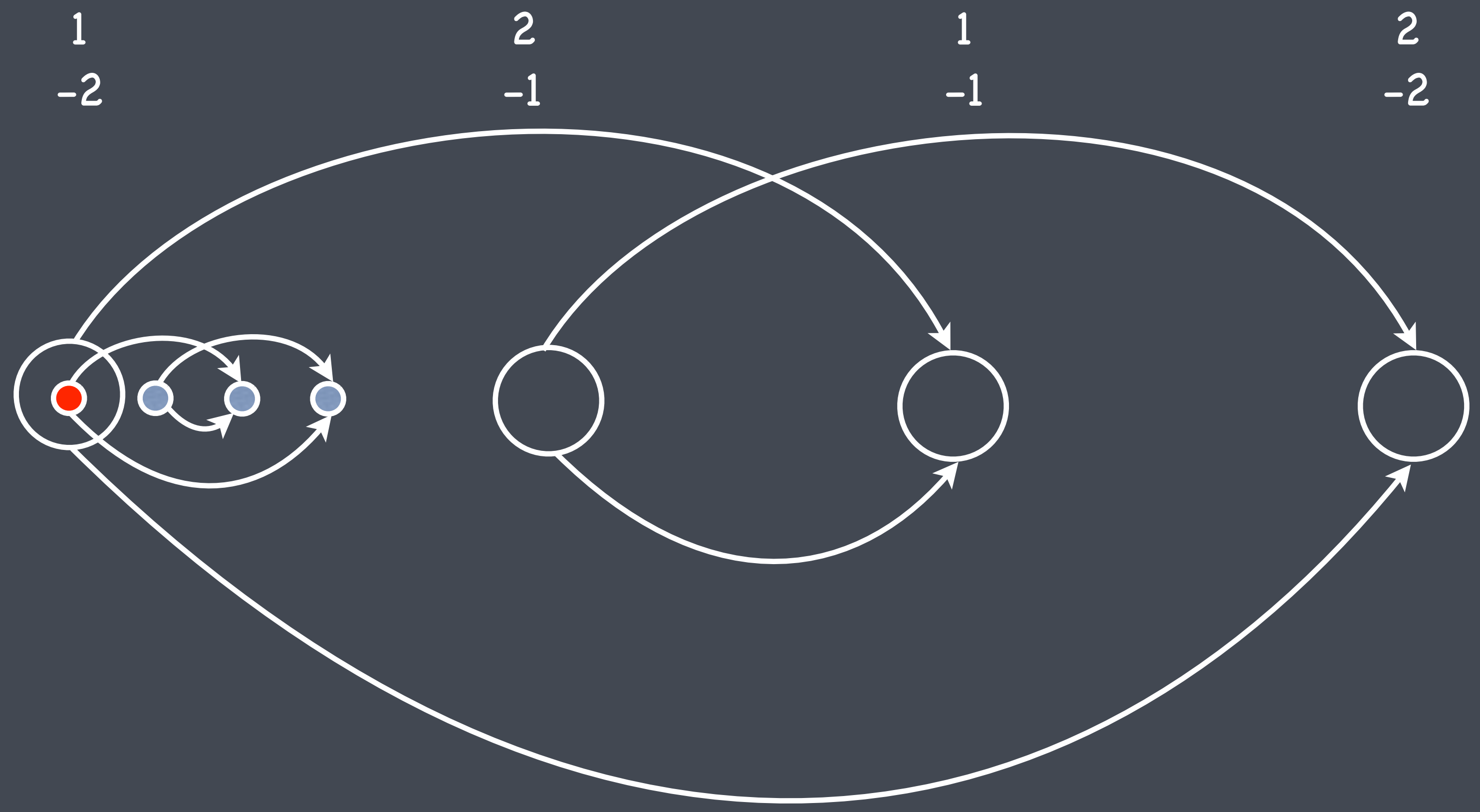
Sphere Automaton



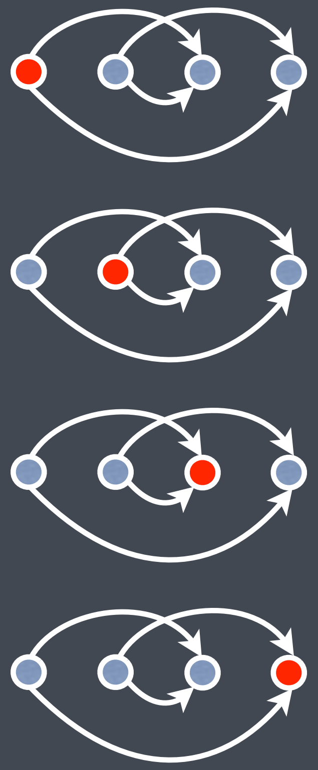
Sphere Automaton



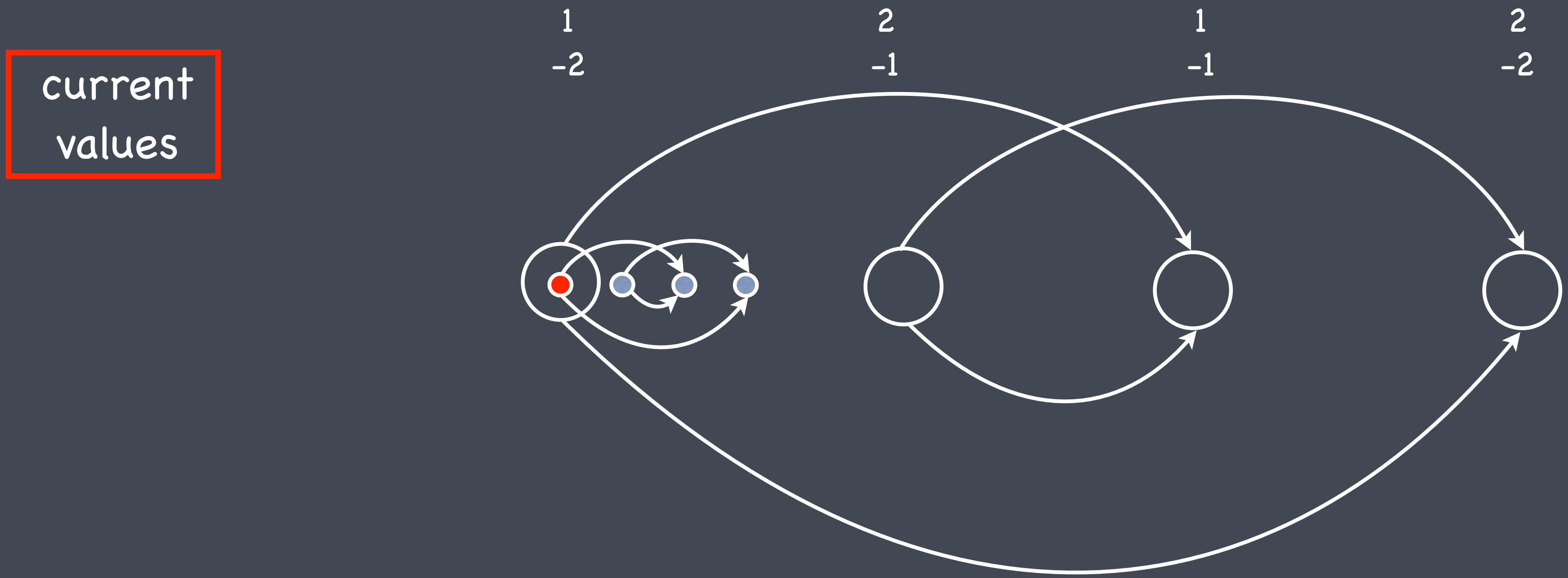
Sphere Automaton



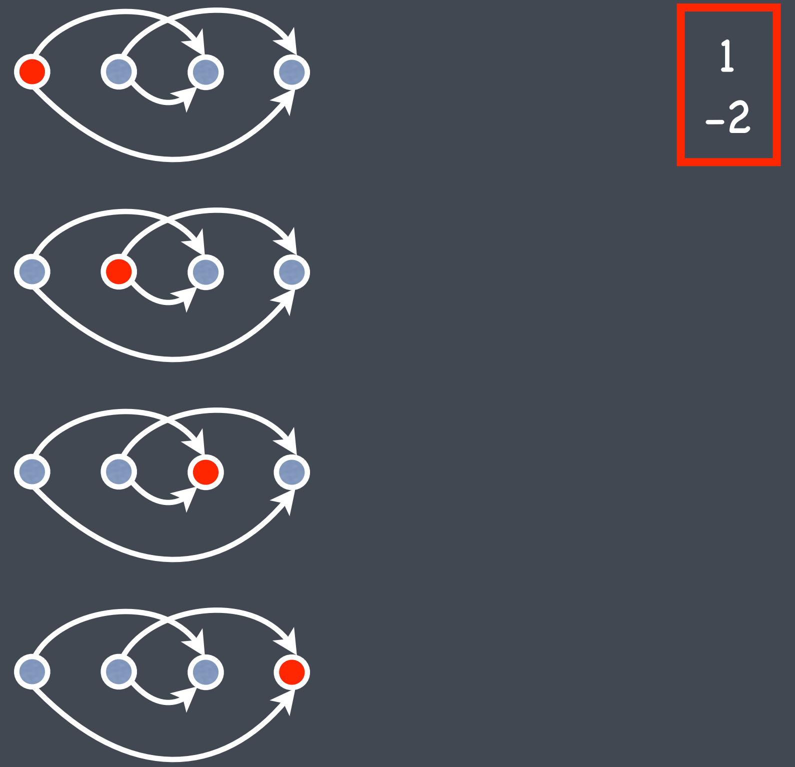
Registers:



Sphere Automaton



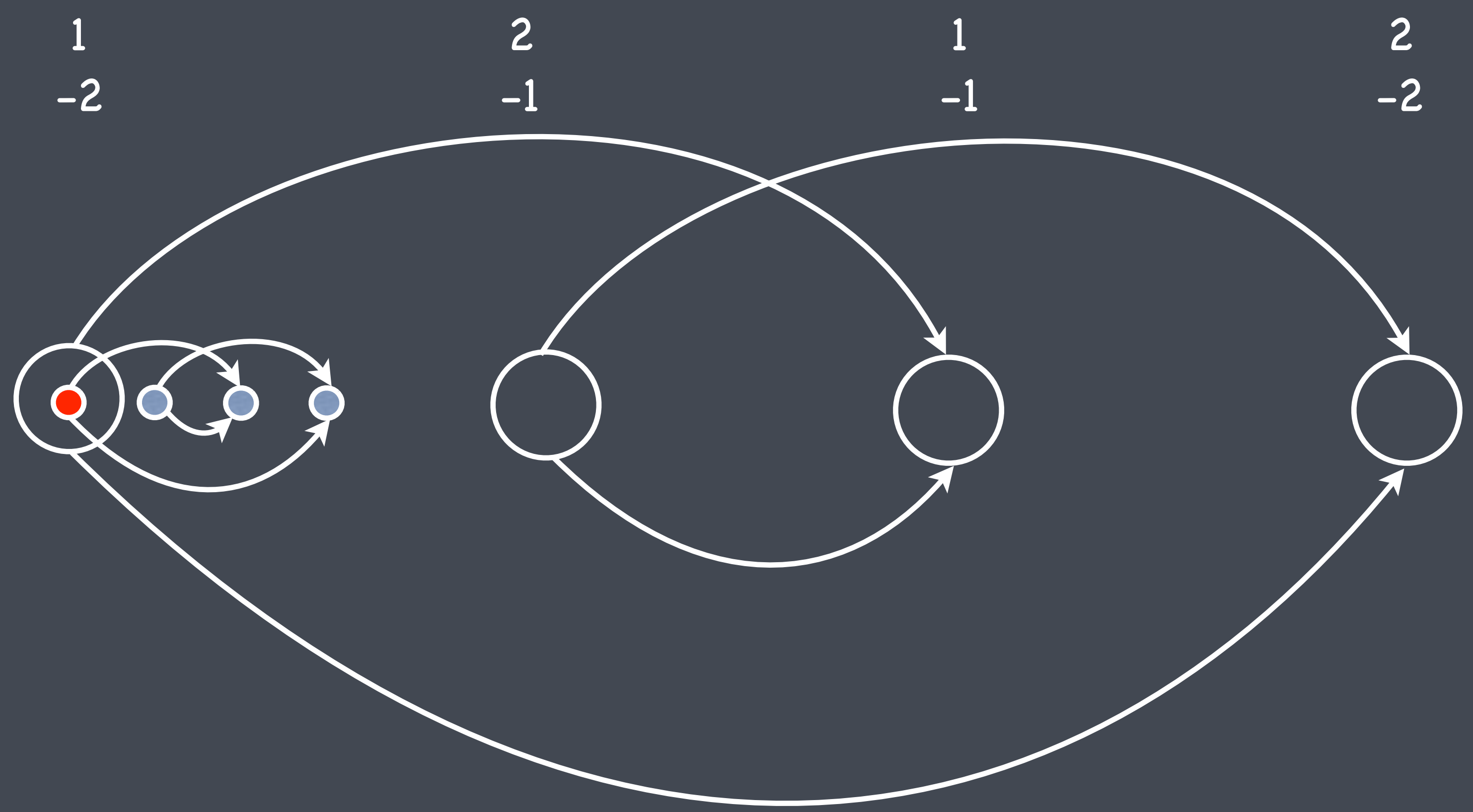
Registers:



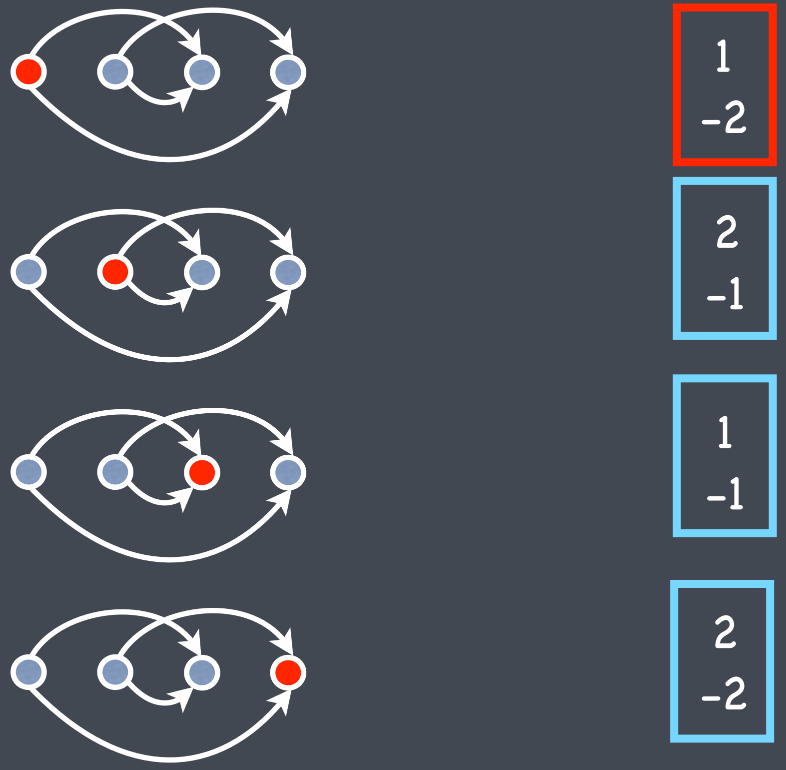
Sphere Automaton

current values

guessed values



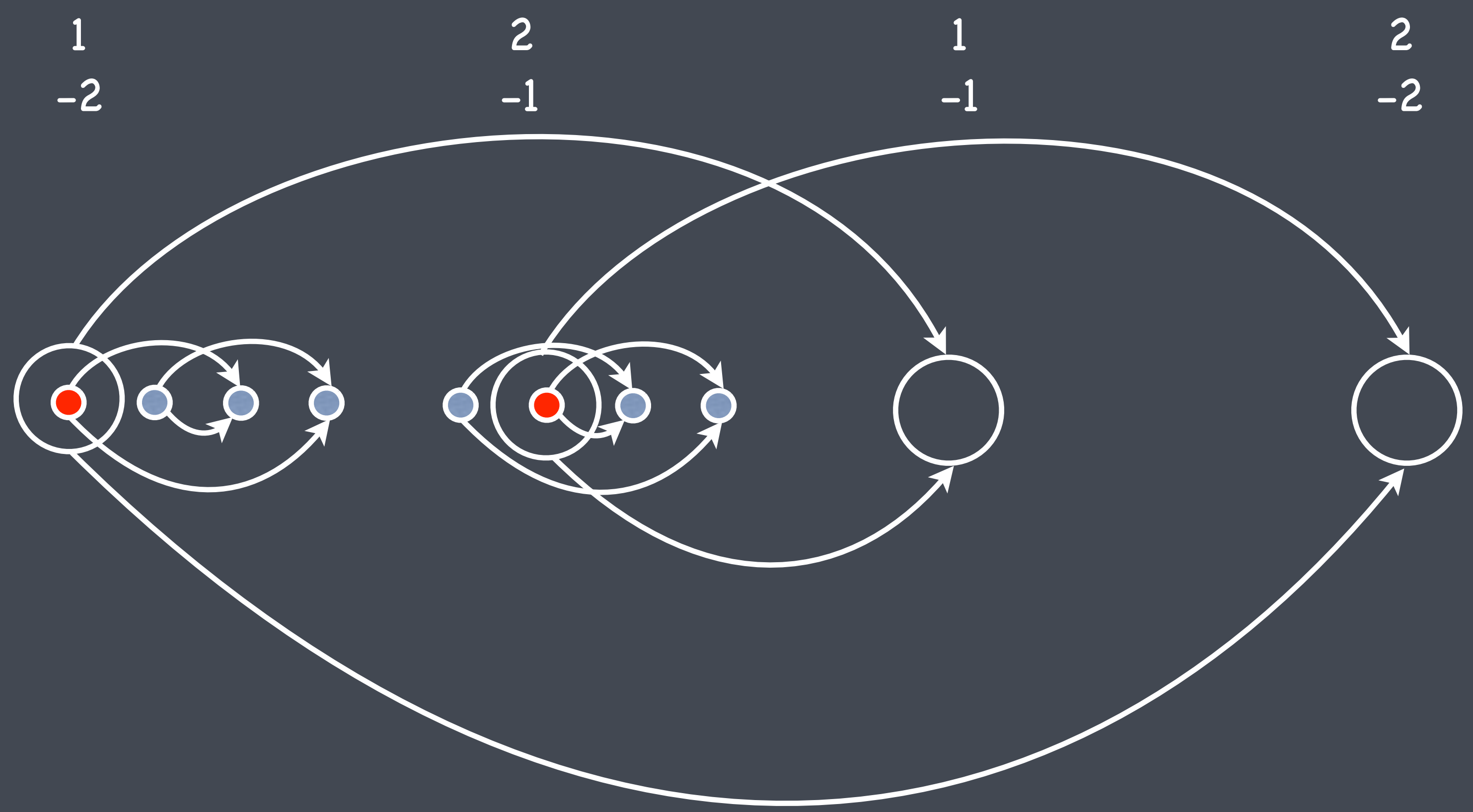
Registers:



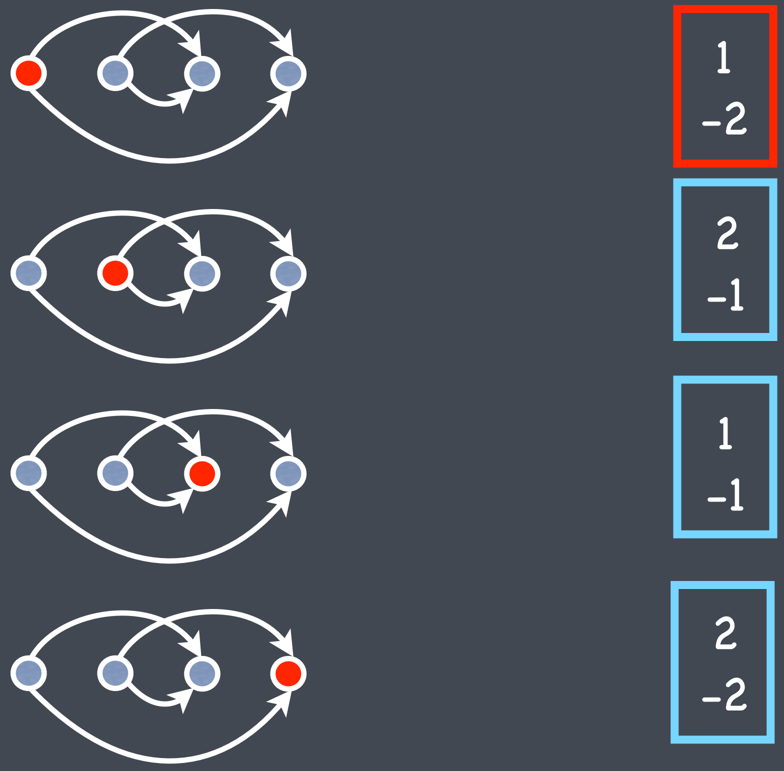
Sphere Automaton

current values

guessed values



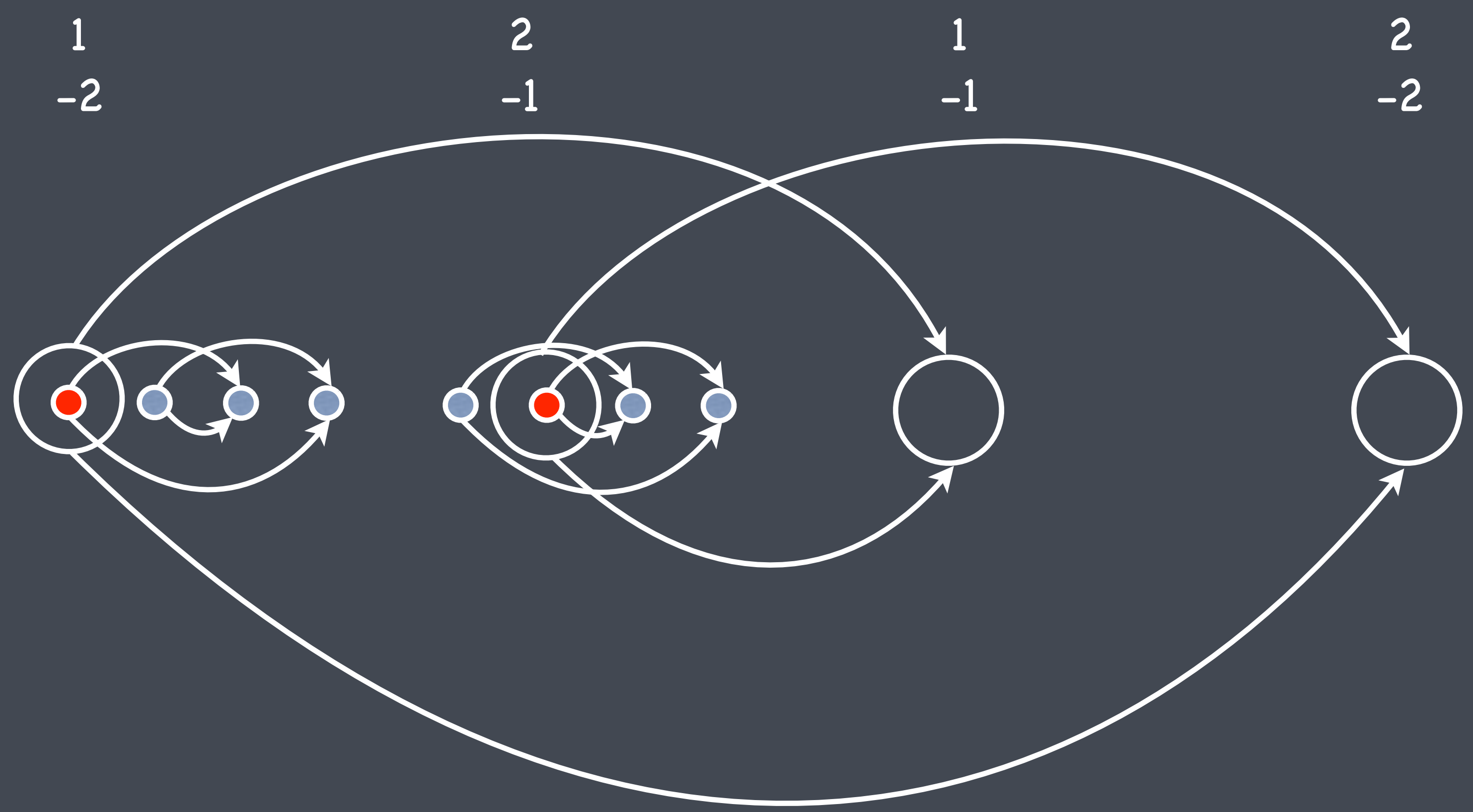
Registers:



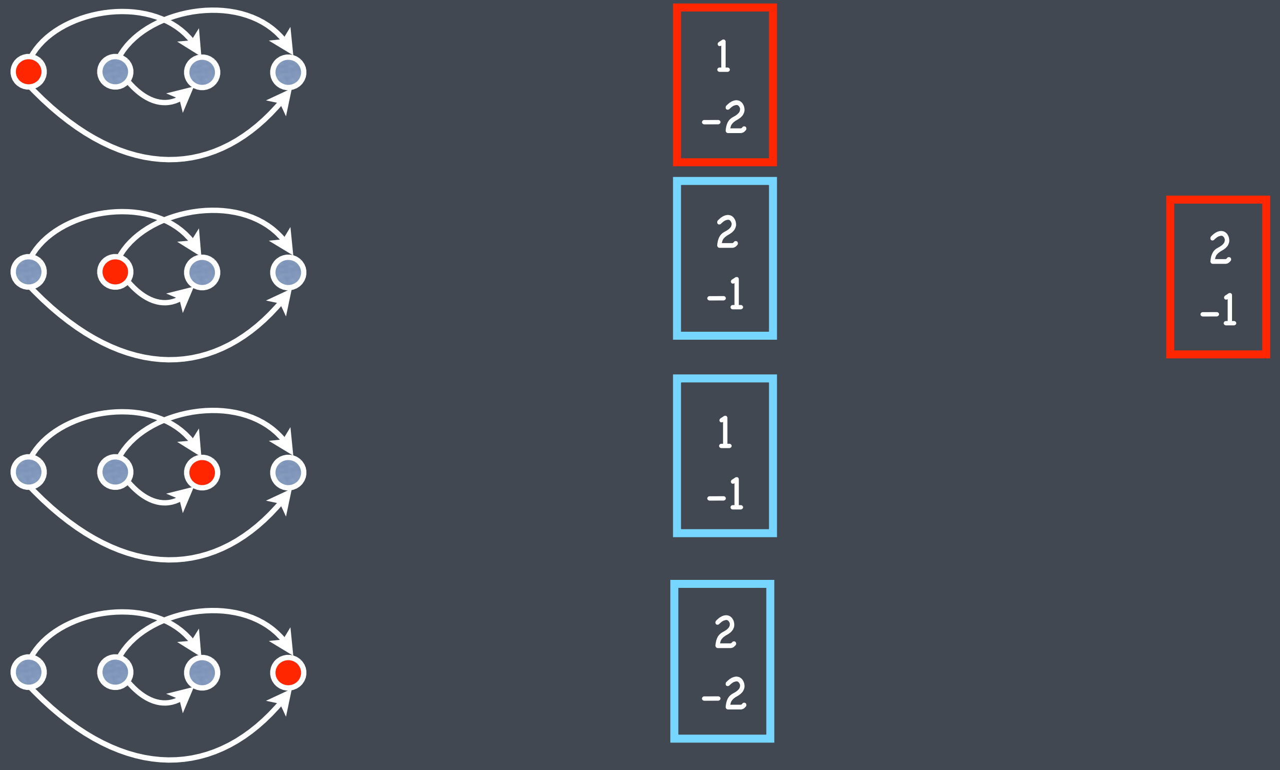
Sphere Automaton

current values

guessed values



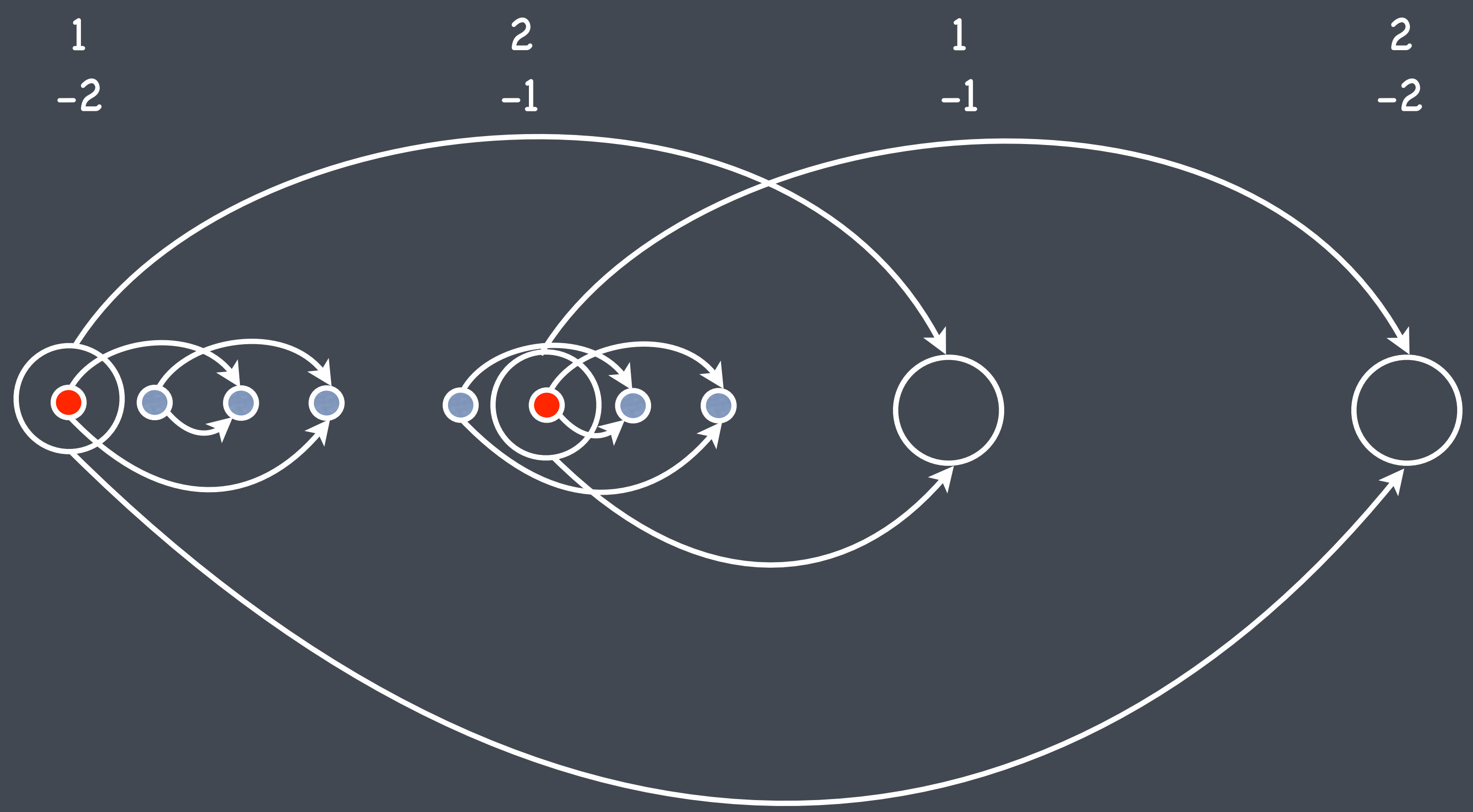
Registers:



Sphere Automaton

current values

guessed values

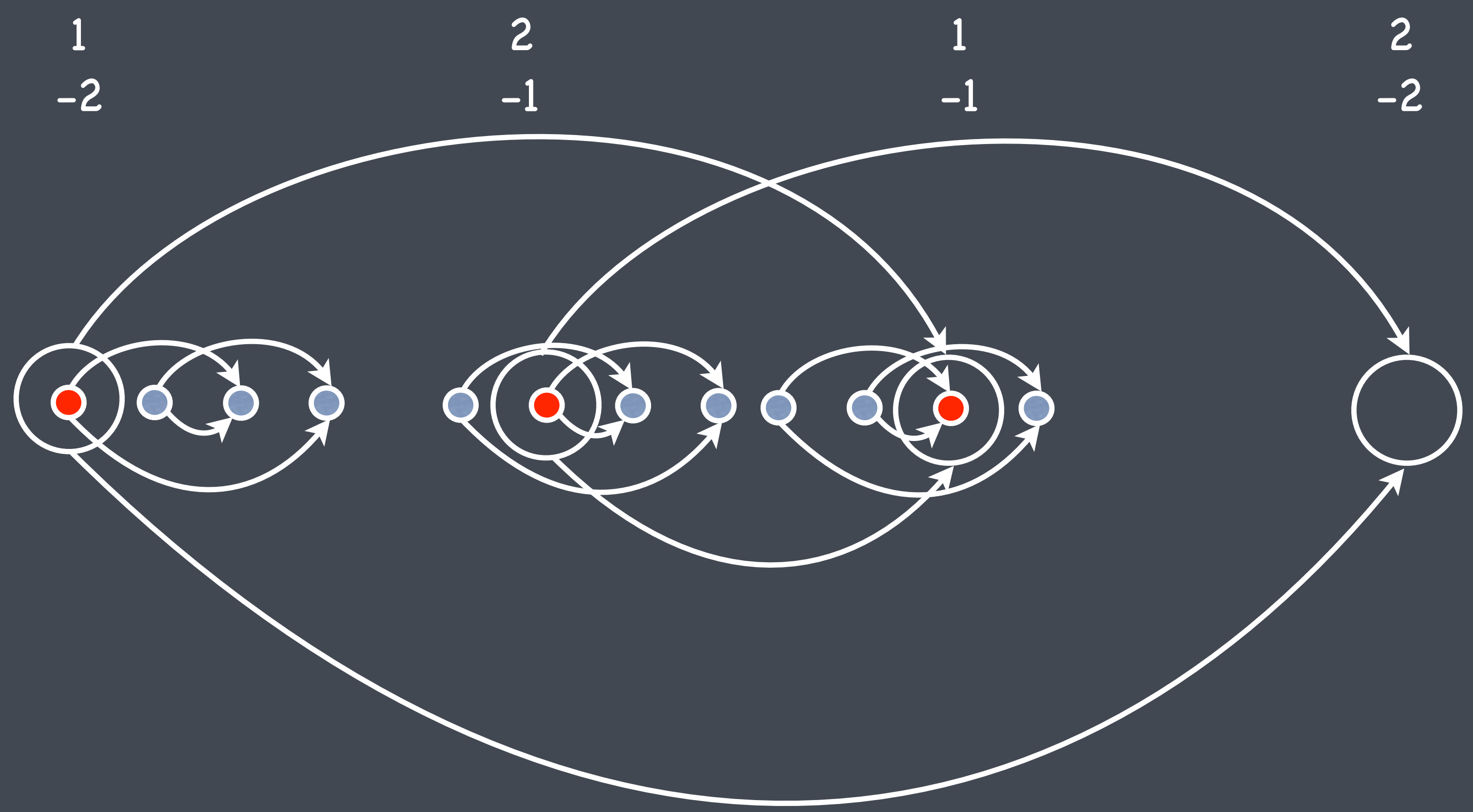


Registers:

Sphere Automaton

current values

guessed values

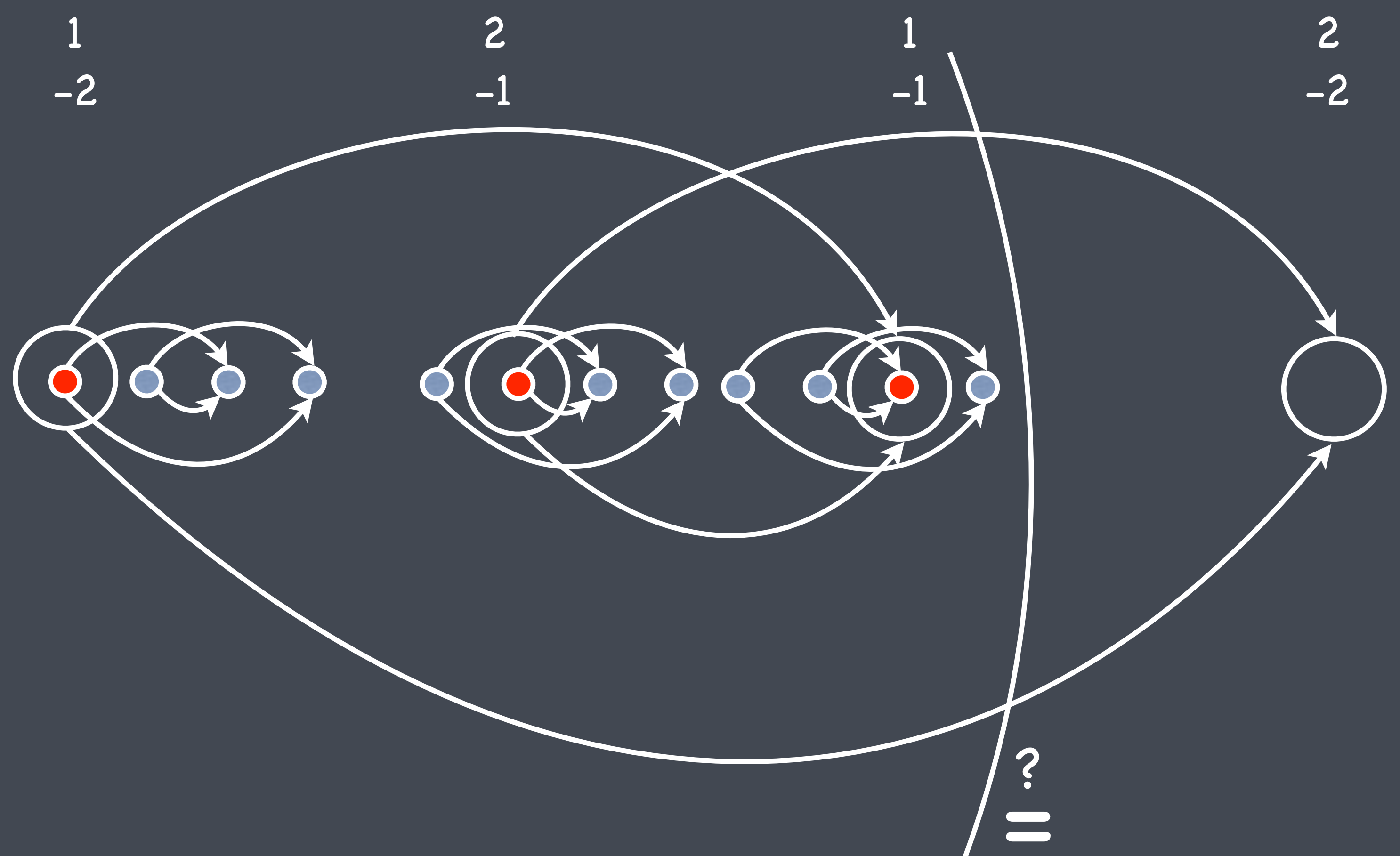


Registers:

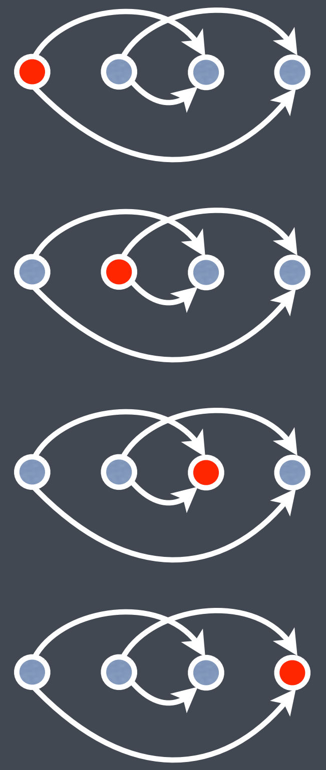
Sphere Automaton

current values

guessed values



Registers:



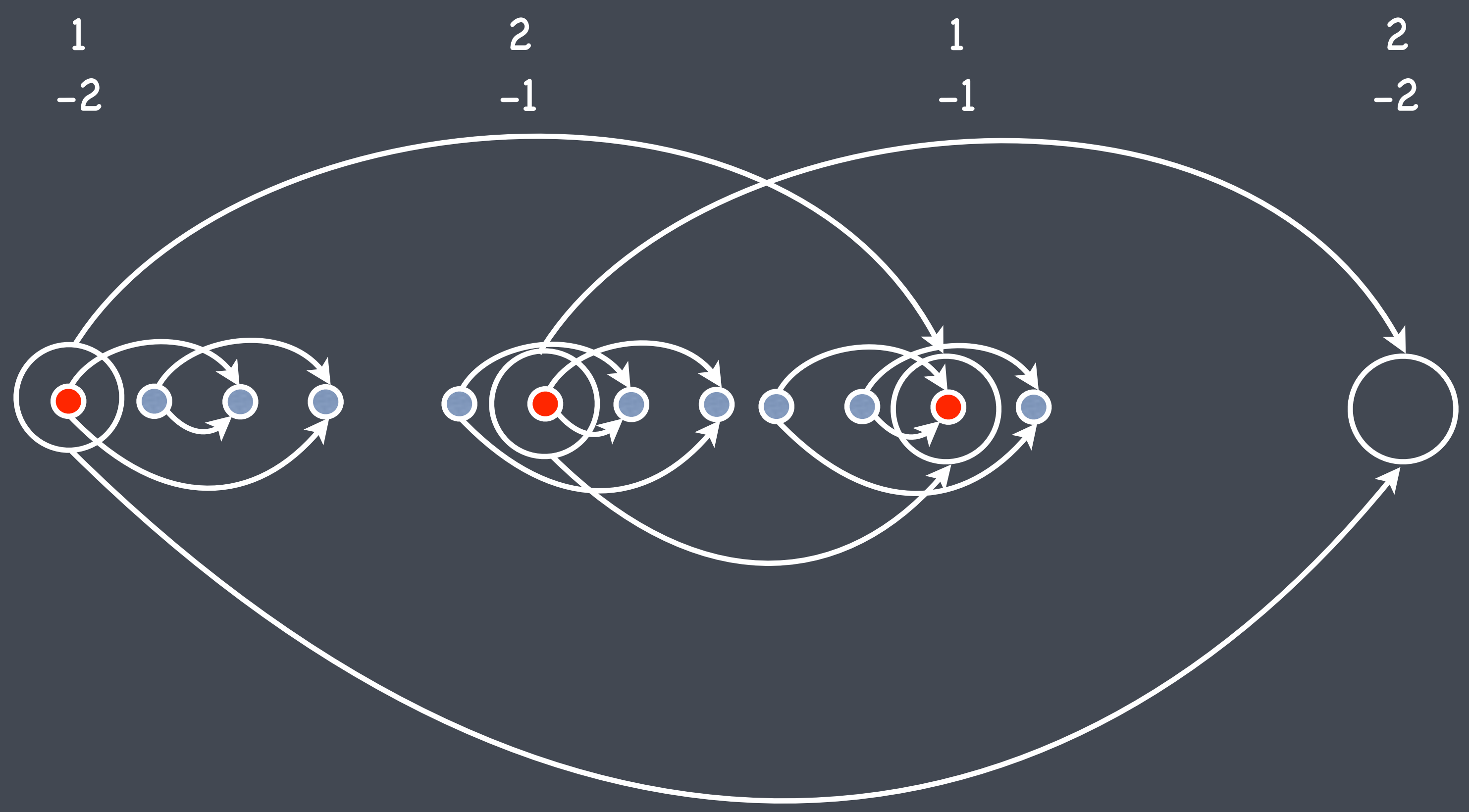
1 -2	$\overset{?}{=}$	1 -2
2 -1	$\overset{?}{=}$	2 -1
1 -1	$\overset{?}{=}$	1 -1
2 -2	$\overset{?}{=}$	2 -2

Sphere Automaton

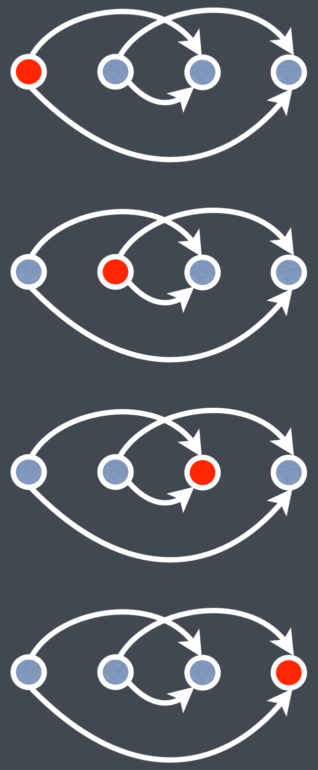
current values

guessed values

previous values



Registers:



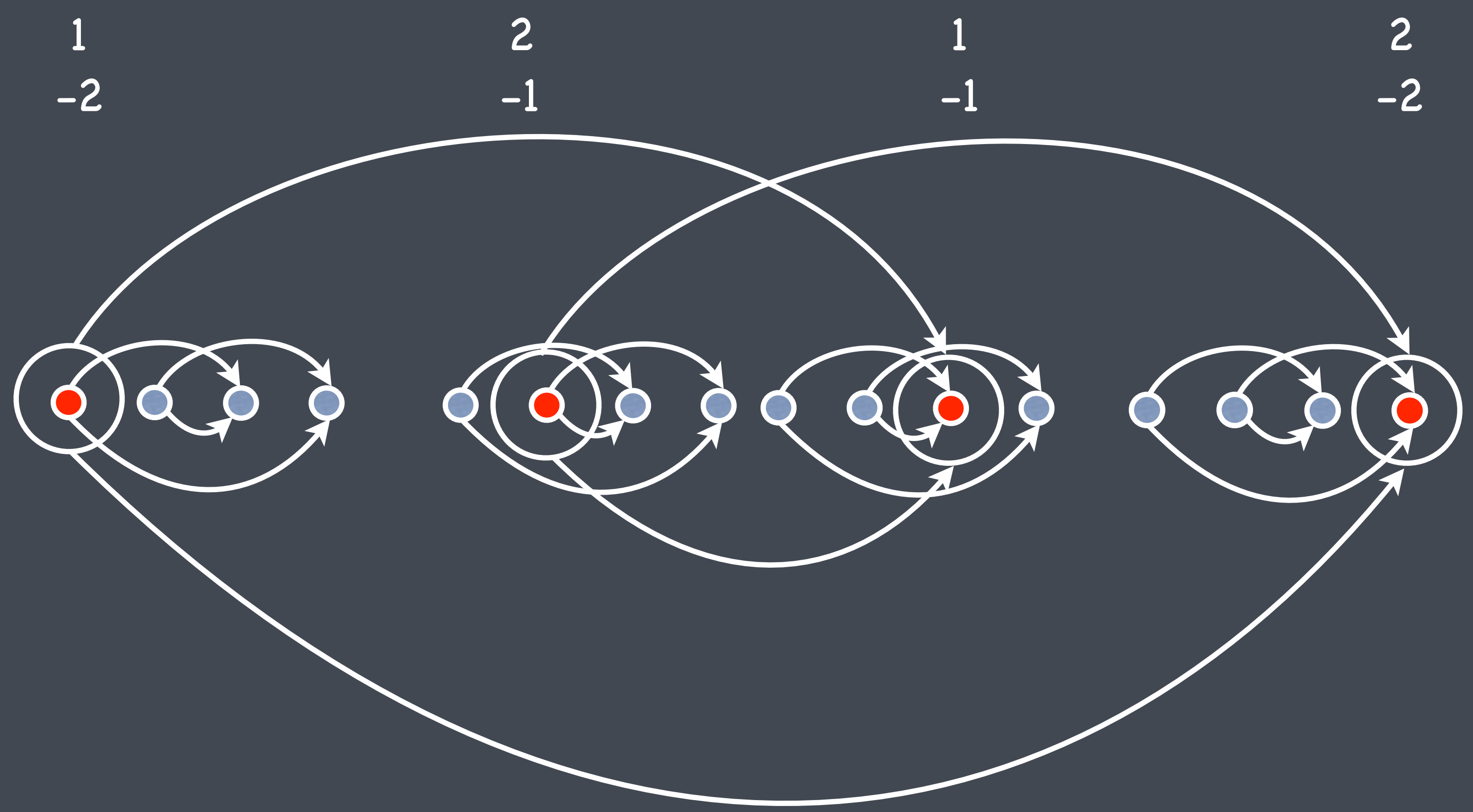
1 -2	1 -2	update →	1 -2
2 -1	2 -1	update →	2 -1
1 -1	1 -1	update →	1 -1
2 -2	2 -2	update →	2 -2

Sphere Automaton

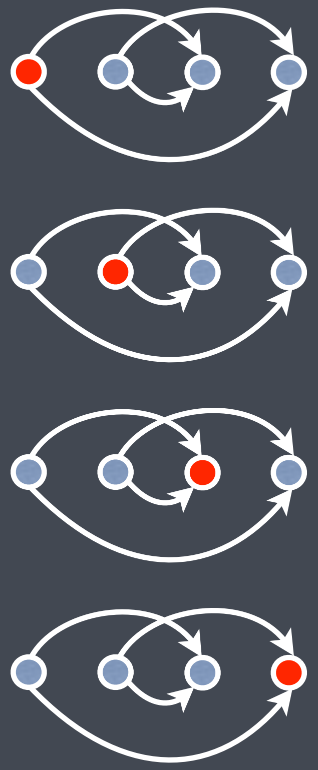
current values

guessed values

previous values



Registers:



1 -2	1 -2	1 -2
2 -1	2 -1	2 -1
1 -1	1 -1	1 -1
2 -2	2 -2	2 -2

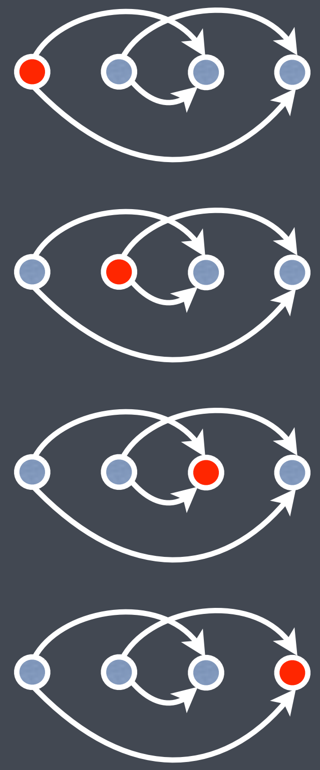
Sphere Automaton

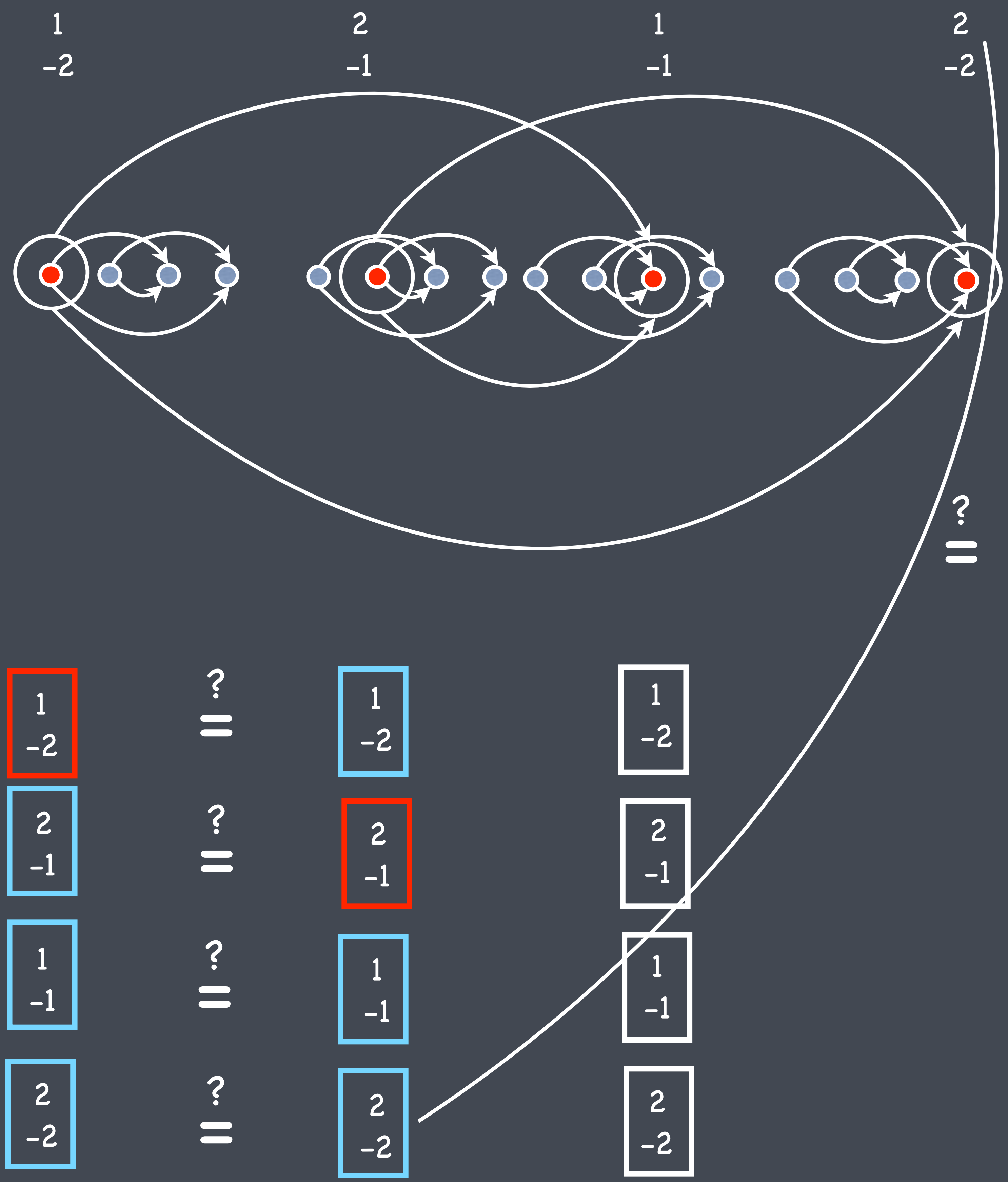
current values

guessed values

previous values

Registers:



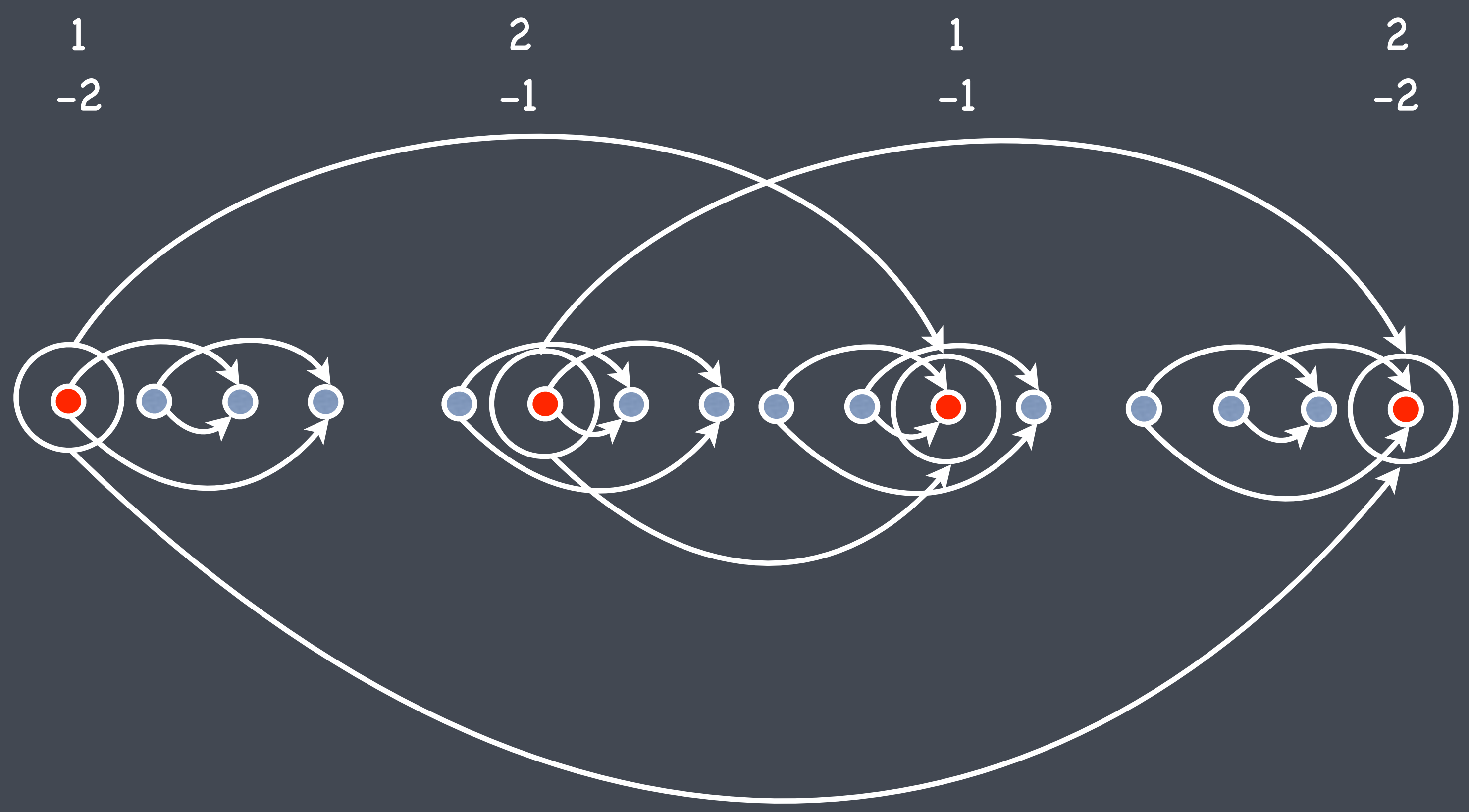


Sphere Automaton

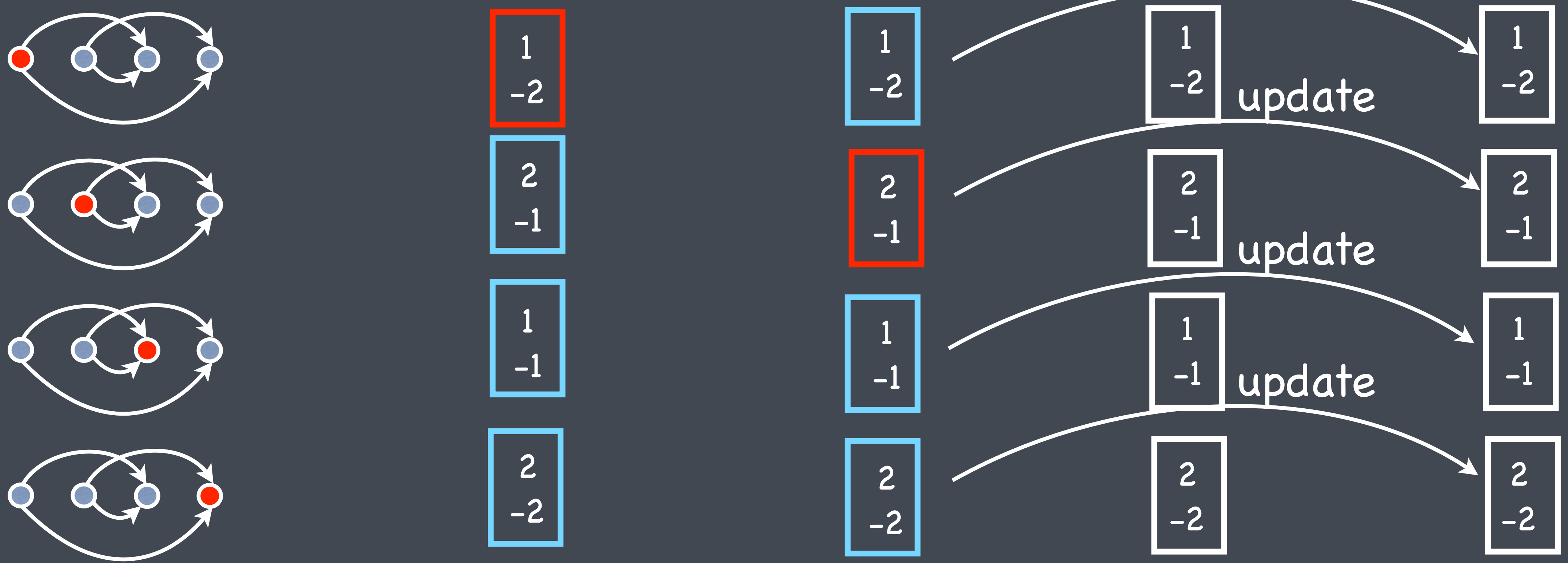
current values

guessed values

previous values



Registers:

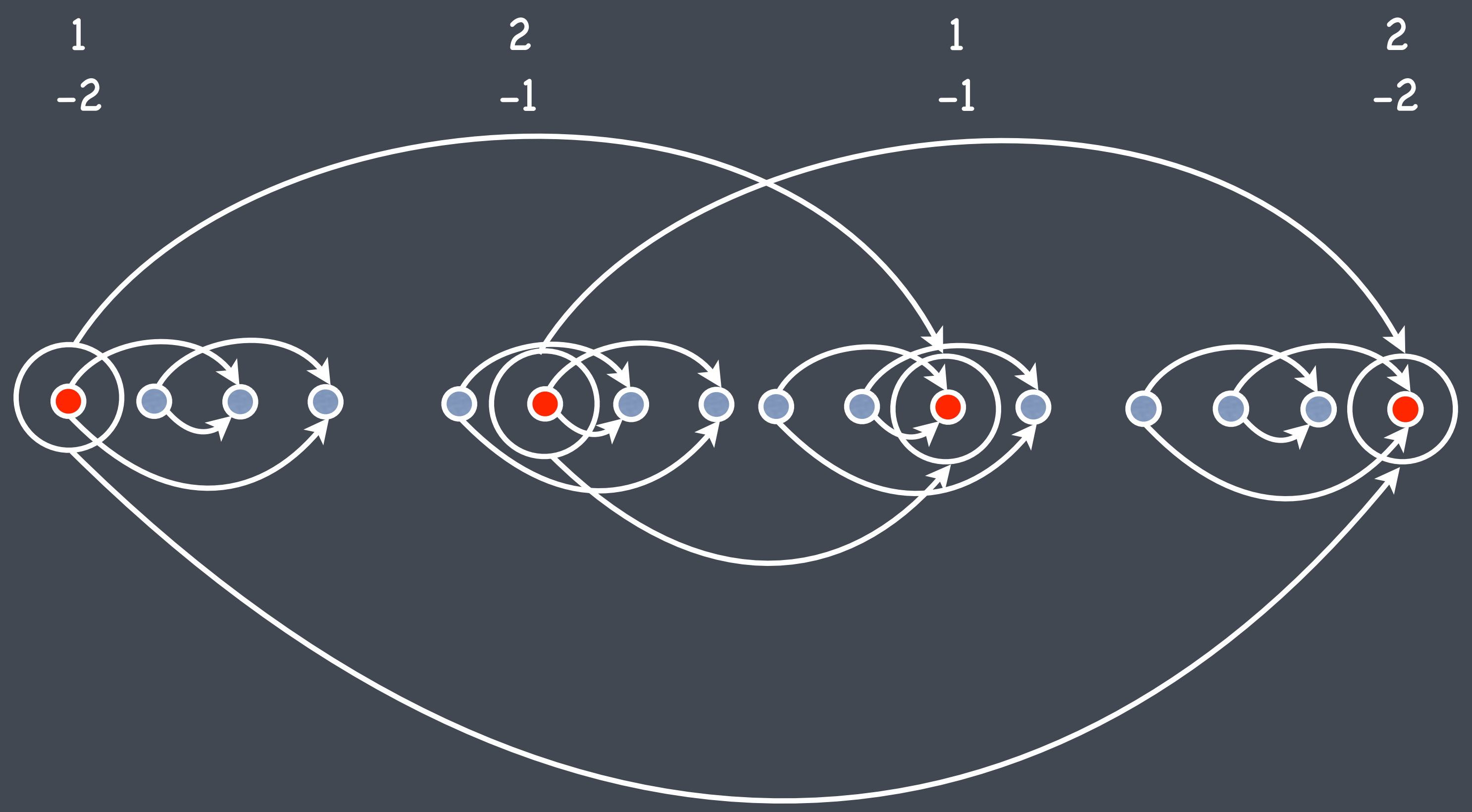


Sphere Automaton

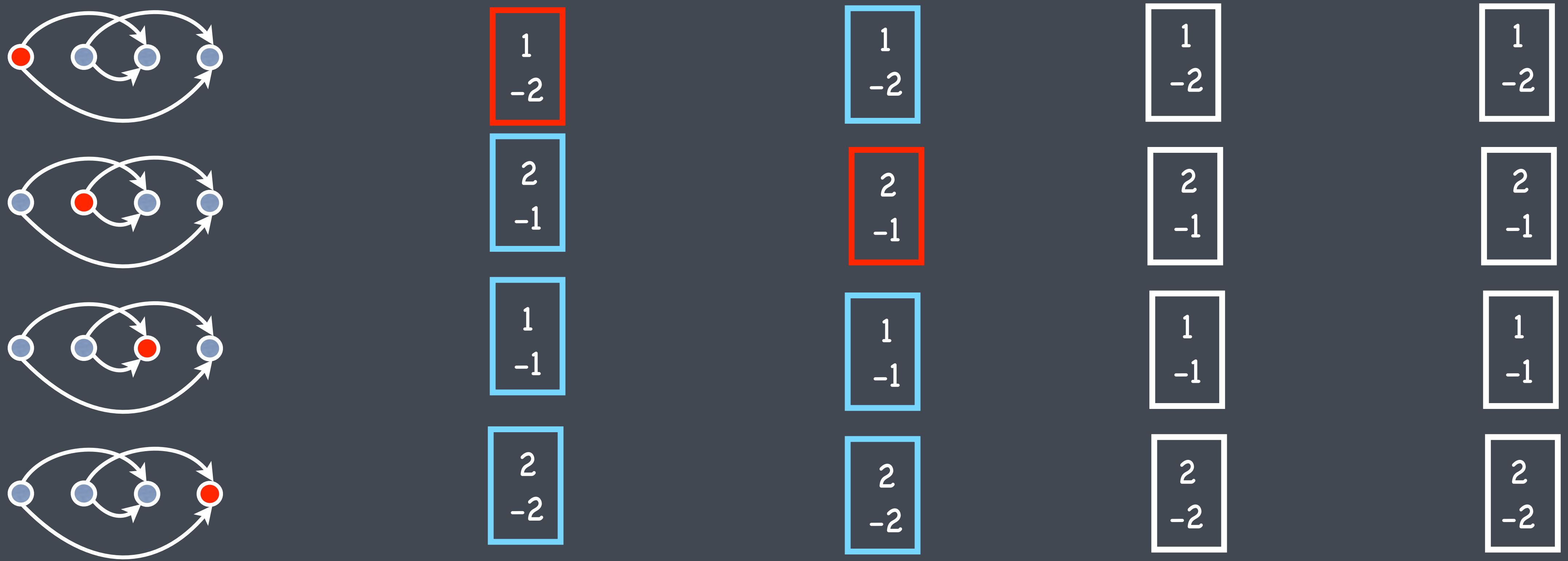
current values

guessed values

previous values

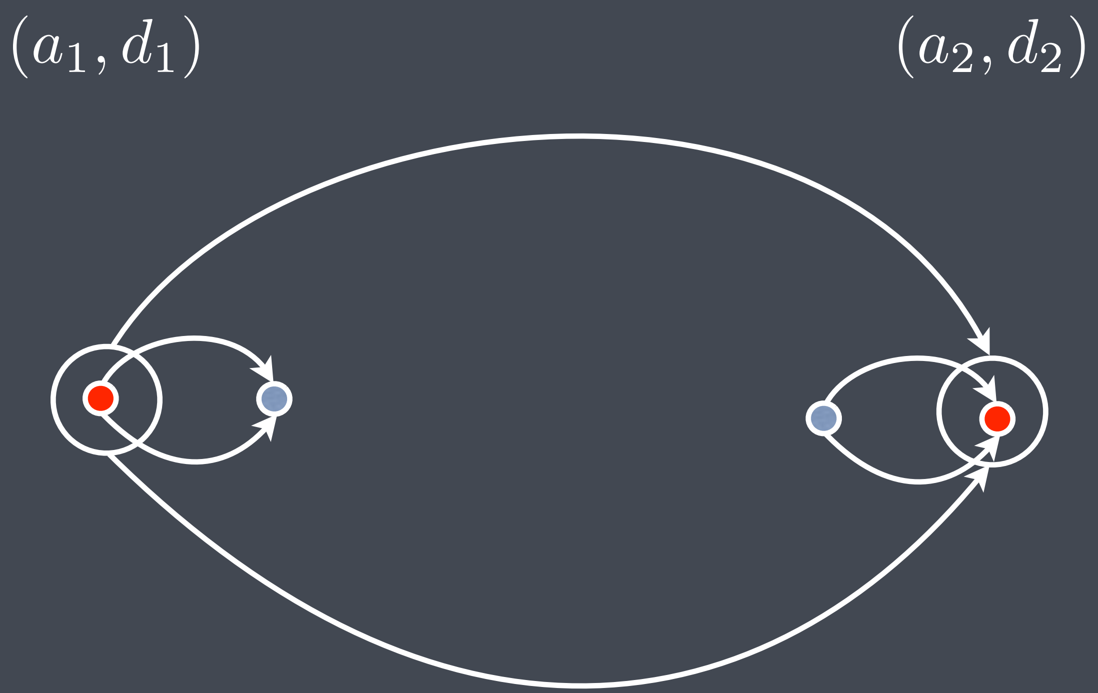


Registers:



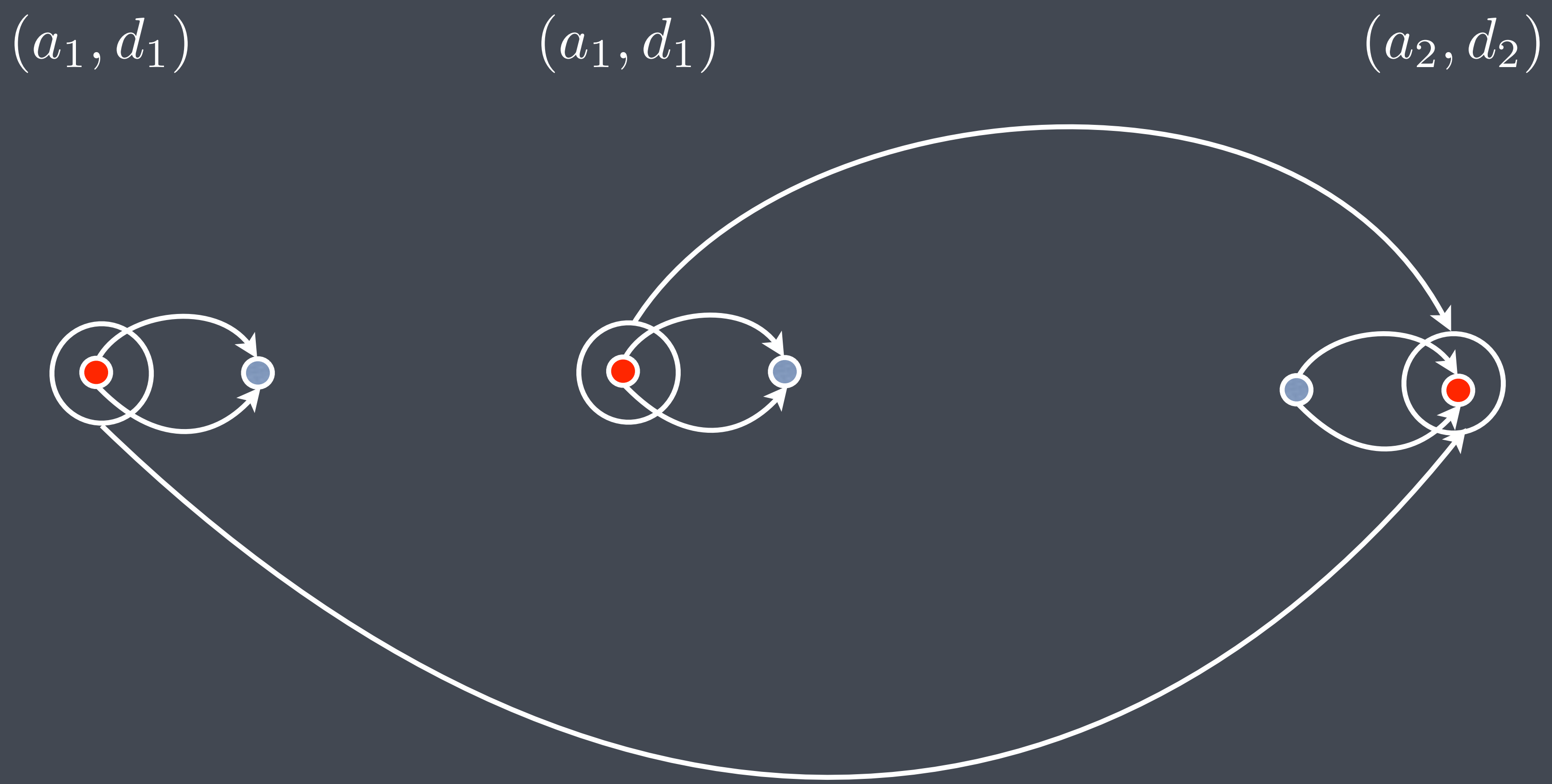
1 -2	1 -2	1 -2	1 -2
2 -1	2 -1	2 -1	2 -1
1 -1	1 -1	1 -1	1 -1
2 -2	2 -2	2 -2	2 -2

Sphere Automaton



How can we be sure that a cycle is closed?

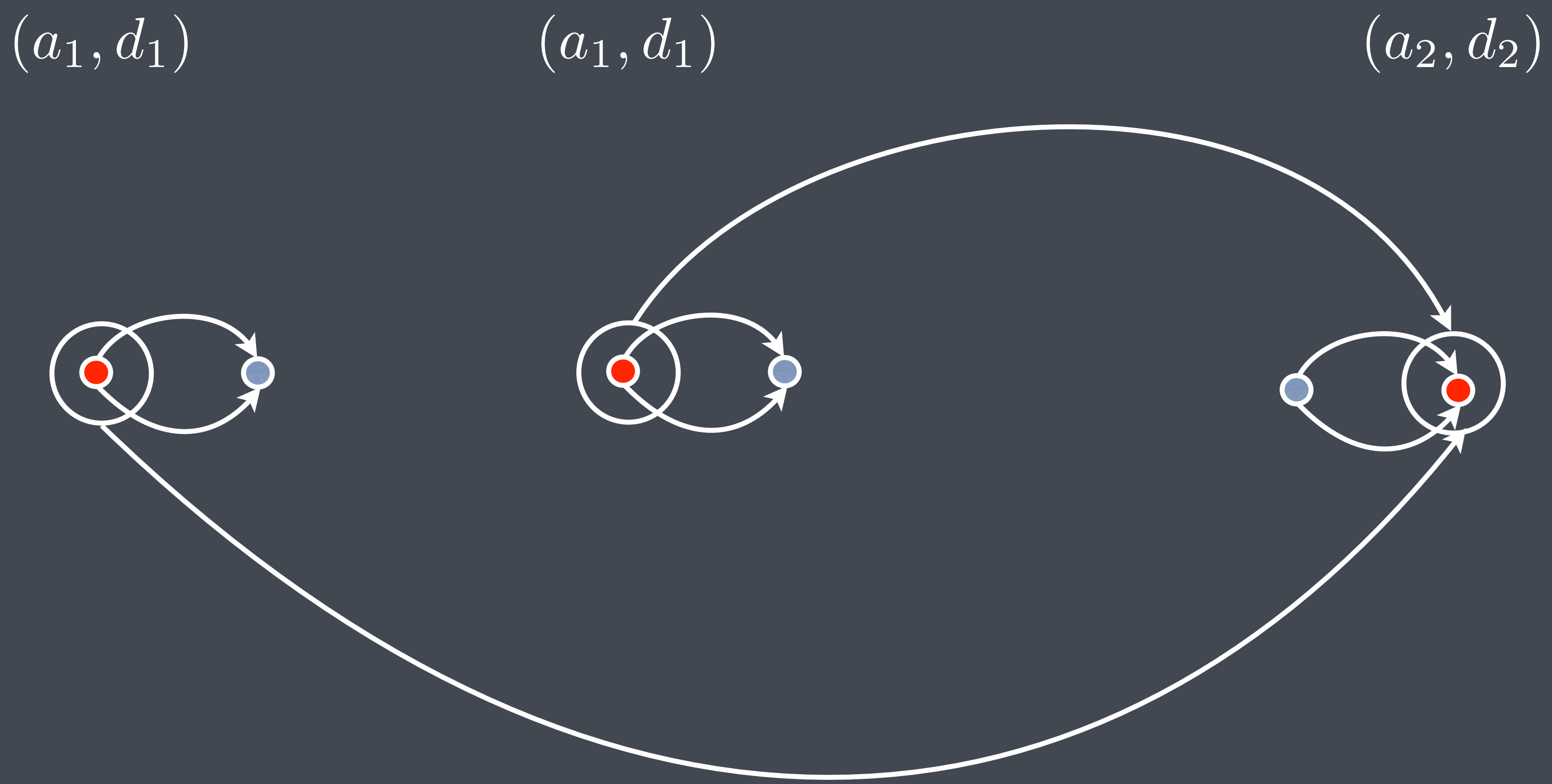
Sphere Automaton



How can we be sure that a cycle is closed?

Suppose it is not closed.

Sphere Automaton

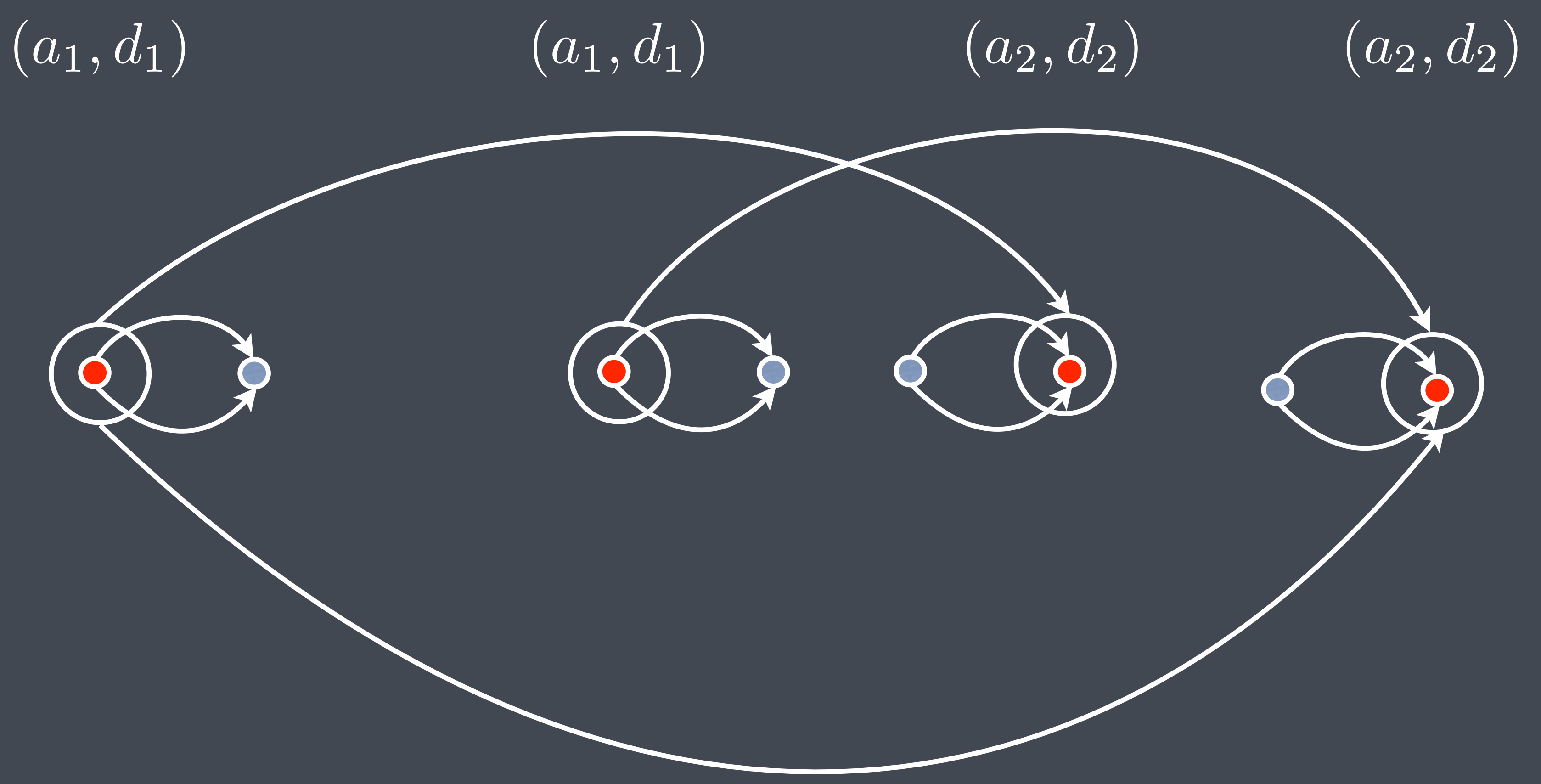


How can we be sure that a cycle is closed?

Suppose it is not closed.

1) we carry along labels and data values
2) relations are monotone

Sphere Automaton



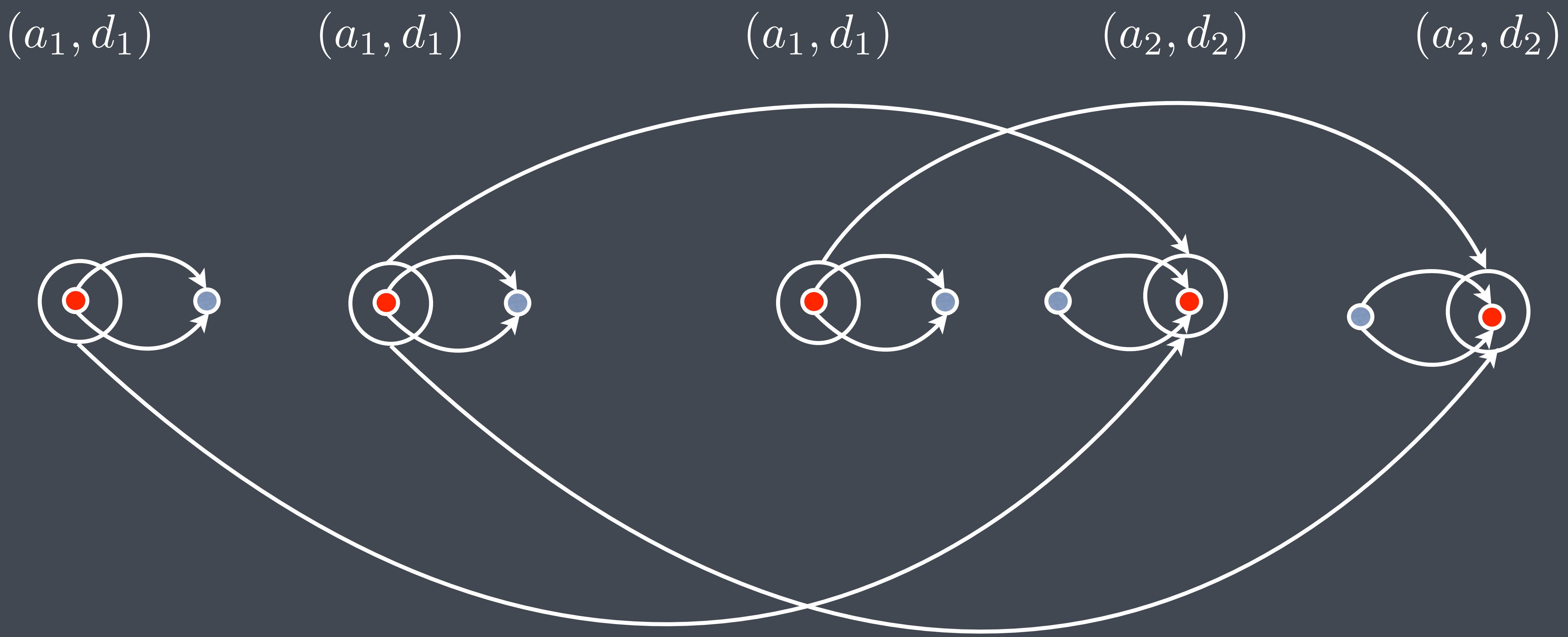
How can we be sure that a cycle is closed?

Suppose it is not closed.

1) we carry along labels and data values

2) relations are monotone

Sphere Automaton

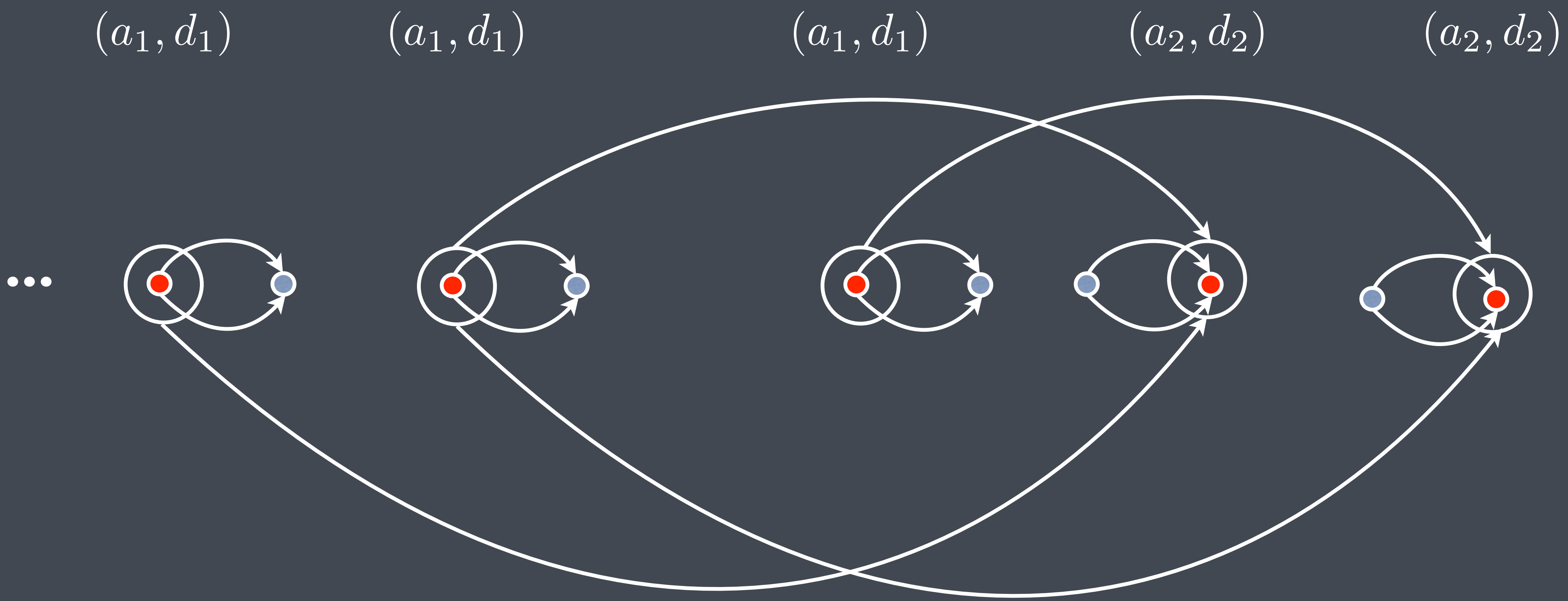


How can we be sure that a cycle is closed?

Suppose it is not closed.

1) we carry along labels and data values
2) relations are monotone

Sphere Automaton



How can we be sure that a cycle is closed?

Suppose it is not closed.

1) we carry along labels and data values
2) relations are monotone

=> infinite descending chain ↯

From Automata to Logic

(m=1)

From Automata to Logic

(m=1)

- MSO$(\prec_{+1}) \subsetneq$ RA

From Automata to Logic

(m=1)

- MSO$(\prec_{+1}) \subsetneq$ RA
- CRA$(\prec_{+1}, \prec_{\sim}) \subseteq$ MSO$(\prec_{+1}, \prec_{\sim})$

From Automata to Logic

(m=1)

- $\mathsf{MSO}(\prec_{+1}) \subsetneq \mathsf{RA}$
- $\mathsf{CRA}(\prec_{+1}, \prec_{\sim}) \subseteq \mathsf{MSO}(\prec_{+1}, \prec_{\sim})$
- $\mathsf{gCRA}(\prec_{+1}, \prec_{\sim}) \subseteq \mathsf{MSO}(\prec_{+1}, \prec_{\sim})$ **?**

From Automata to Logic

(m=1)

- MSO$(\prec_{+1}) \subsetneq$ RA
- CRA$(\prec_{+1}, \prec_{\sim}) \subseteq$ MSO$(\prec_{+1}, \prec_{\sim})$
- gCRA$(\prec_{+1}, \prec_{\sim}) \subseteq$ MSO$(\prec_{+1}, \prec_{\sim})$ **?**
- MSO$(\prec_{+1}, \prec_{\sim}) \not\subseteq$ gCRA$(\prec_{+1}, \prec_{\sim})$

Applications

Applications

- Synthesis of dynamic communicating systems

Applications

- Synthesis of dynamic communicating systems
- Class register automata subsume
  dynamic communicating automata [B., Hélouët, 2010]

Applications

- Synthesis of dynamic communicating systems
- Class register automata subsume dynamic communicating automata [B., Hélouët, 2010]
  - $\mathrm{upd}(r) = (\sqsubset_{\mathrm{msg}}, r')$ receive process id

Applications

- Synthesis of dynamic communicating systems
- Class register automata subsume dynamic communicating automata [B., Hélouët, 2010]
  - $\mathrm{upd}(r) = (\sqsubset_{\mathrm{msg}}, r')$ receive process id
  - $(\prec_{\mathrm{proc}}, r) = (\sqsubset_{\mathrm{msg}}, r_0)$ receive from r

Applications

- Synthesis of dynamic communicating systems
- Class register automata subsume
  dynamic communicating automata [B., Hélouët, 2010]
  - $\mathrm{upd}(r) = (\sqsubset_{\mathrm{msg}}, r')$ receive process id
  - $(\prec_{\mathrm{proc}}, r) = (\sqsubset_{\mathrm{msg}}, r_0)$ receive from r
- Finitely branching transition system
  => partial satisfiability checking

Applications

- Synthesis of dynamic communicating systems
- Class register automata subsume
  dynamic communicating automata [B., Hélouët, 2010]
  - $\mathrm{upd}(r) = (\sqsubset_{\mathrm{msg}}, r')$ receive process id
  - $(\prec_{\mathrm{proc}}, r) = (\sqsubset_{\mathrm{msg}}, r_0)$ receive from r
- Finitely branching transition system
  => partial satisfiability checking
- Tool to show that a property is not EMSO-definable

Applications

- Synthesis of dynamic communicating systems
- Class register automata subsume dynamic communicating automata [B., Hélouët, 2010]
  - $\mathrm{upd}(r) = (\sqsubset_{\mathrm{msg}}, r')$ receive process id
  - $(\prec_{\mathrm{proc}}, r) = (\sqsubset_{\mathrm{msg}}, r_0)$ receive from r
- Finitely branching transition system => partial satisfiability checking
- Tool to show that a property is not EMSO-definable
- Study of combined expressive power of existing concepts

Conclusion

Conclusion

- General framework for specification and implementation of data-word languages

Conclusion

- General framework for specification and implementation of data-word languages
- Synthesis of dynamic communicating systems

Conclusion

- General framework for specification and implementation of data-word languages
- Synthesis of dynamic communicating systems
- Next: synthesize more practical automata
  => restricted specification languages
  => temporal logics