

Identifiers in Registers

Describing Network Algorithms with Logic

Benedikt Bollig¹ Patricia Bouyer¹ Fabian Reiter²

¹LSV, University of Paris-Saclay

²Technical University of Munich

3 April 2019 @ TUM, Munich

Fagin's theorem (1974)

Fagin's theorem (1974)



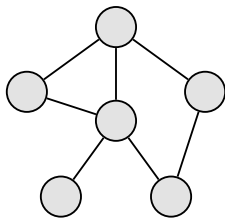
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



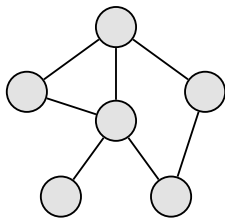
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



Fagin's theorem (1974)

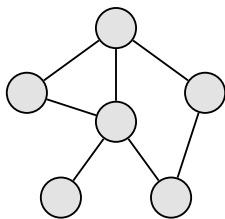
\exists SECOND-ORDER LOGIC



Example: Hamiltonian path

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

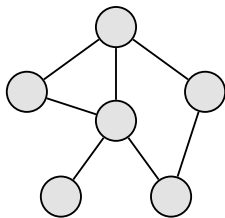


Example: Hamiltonian path

∃R ()

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

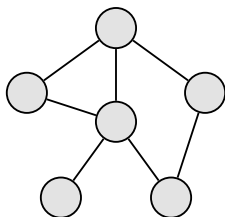


Example: Hamiltonian path

$\exists R (\text{“R is a strict total order”} \wedge)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

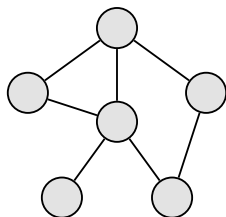


Example: Hamiltonian path

$\exists R (\text{“}R \text{ is a strict total order”} \wedge \text{“}R\text{-successors are adjacent”})$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



Example: Hamiltonian path

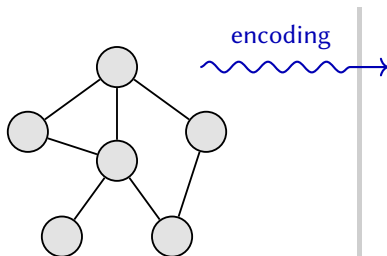
$\exists R (\text{“R is a strict total order”} \wedge \text{“R-successors are adjacent”})$

NP TURING MACHINES

Fagin's theorem (1974)

\exists SECOND-ORDER LOGIC

NP TURING MACHINES

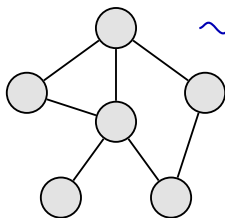


Example: Hamiltonian path

$\exists R (\text{"R is a strict total order"} \wedge \text{"R-successors are adjacent"})$

Fagin's theorem (1974)

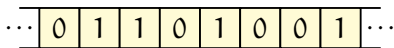
\exists SECOND-ORDER LOGIC



encoding



NP TURING MACHINES

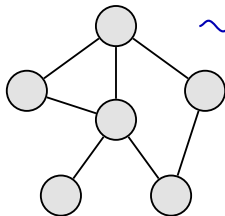


Example: Hamiltonian path

$\exists R (\text{“}R \text{ is a strict total order”} \wedge \text{“}R\text{-successors are adjacent”})$

Fagin's theorem (1974)

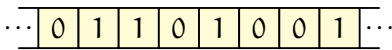
\exists SECOND-ORDER LOGIC



encoding



NP TURING MACHINES

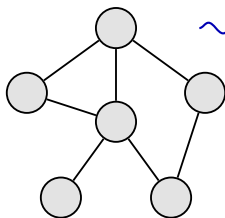


Example: Hamiltonian path

$\exists R (\text{“}R \text{ is a strict total order”} \wedge$
 $\text{“}R\text{-successors are adjacent”})$

Fagin's theorem (1974)

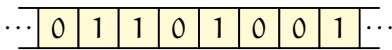
\exists SECOND-ORDER LOGIC



encoding



NP TURING MACHINES



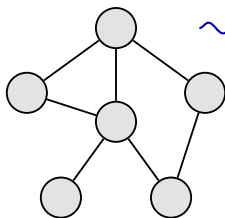
Example: Hamiltonian path

$\exists R (\text{“}R \text{ is a strict total order”} \wedge \text{“}R\text{-successors are adjacent”})$

► Nondeterministic moves

Fagin's theorem (1974)

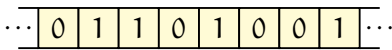
\exists SECOND-ORDER LOGIC



encoding



NP TURING MACHINES



Example: Hamiltonian path

$\exists R (\text{“}R \text{ is a strict total order”} \wedge$
 $\text{“}R\text{-successors are adjacent”})$

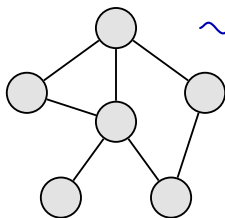
- ▶ Nondeterministic moves
- ▶ Polynomial running time

Fagin's theorem (1974)

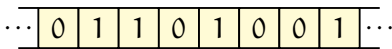
\exists SECOND-ORDER LOGIC



NP TURING MACHINES



encoding

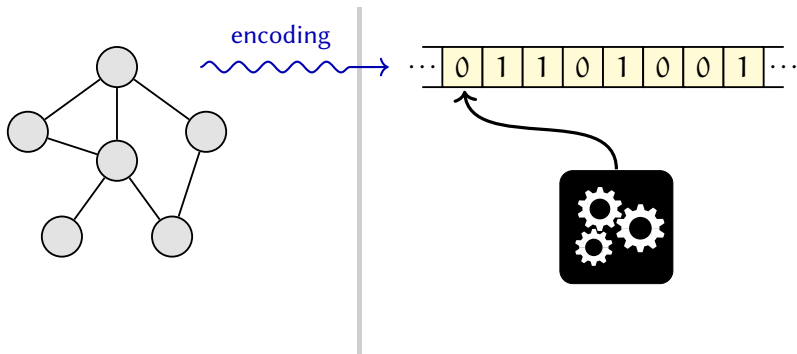


Example: Hamiltonian path

$\exists R$ (“R is a strict total order” \wedge
“R-successors are adjacent”)

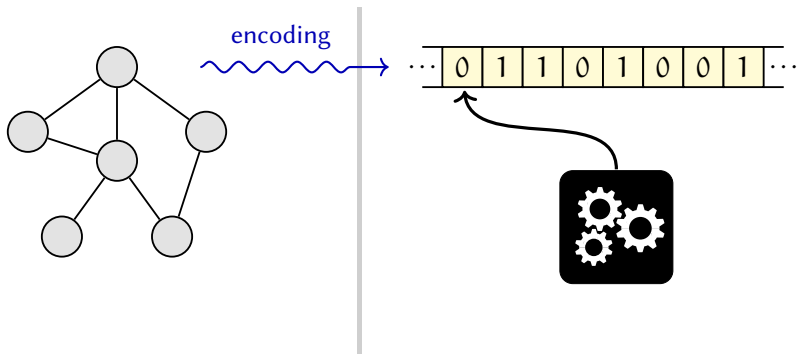
- ▶ Nondeterministic moves
- ▶ Polynomial running time

Descriptive complexity



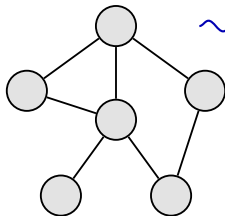
Descriptive complexity

SOME LOGICAL FORMALISM

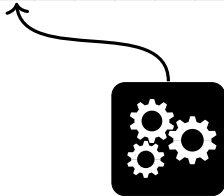
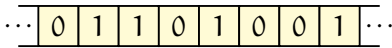


Descriptive complexity

SOME LOGICAL FORMALISM



encoding

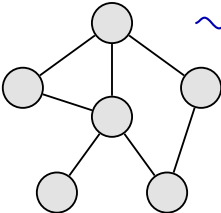


Descriptive complexity

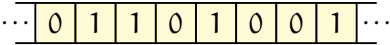
SOME LOGICAL FORMALISM



SOME ABSTRACT MACHINES



encoding

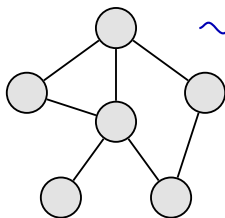


Descriptive complexity

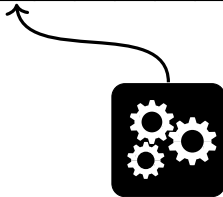
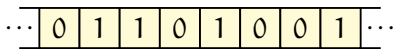
SOME LOGICAL FORMALISM



SOME ABSTRACT MACHINES



encoding



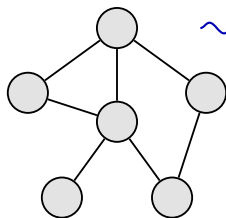
Formula class Φ

Descriptive complexity

SOME LOGICAL FORMALISM

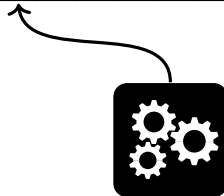
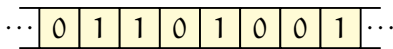


SOME ABSTRACT MACHINES



Formula class Φ

encoding



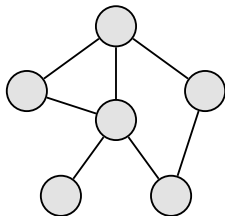
Algorithm class \mathcal{A}

Descriptive distributed complexity



Descriptive distributed complexity

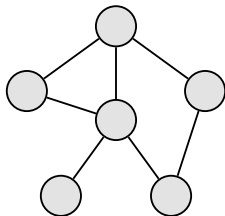
SOME LOGICAL FORMALISM



Formula class Φ

Descriptive distributed complexity

SOME LOGICAL FORMALISM



Formula class Φ

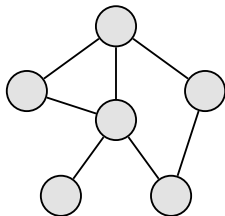


Descriptive distributed complexity

SOME LOGICAL FORMALISM



COMMUNICATING MACHINES



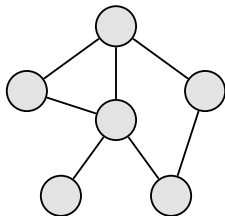
Formula class Φ

Descriptive distributed complexity

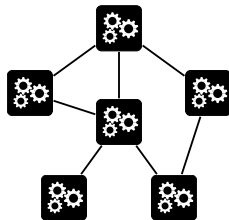
SOME LOGICAL FORMALISM



COMMUNICATING MACHINES



Formula class Φ

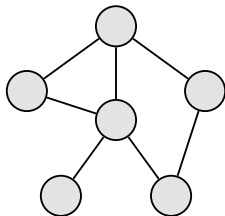


Descriptive distributed complexity

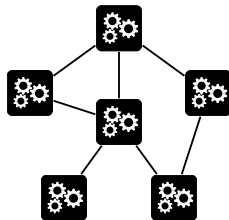
SOME LOGICAL FORMALISM



COMMUNICATING MACHINES

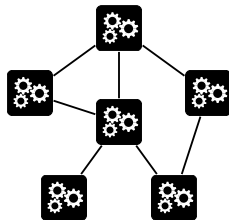
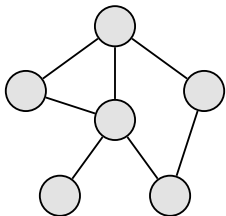


Formula class Φ

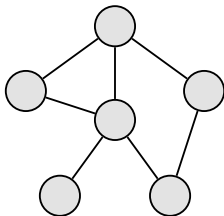


Distributed algorithm class \mathcal{A}

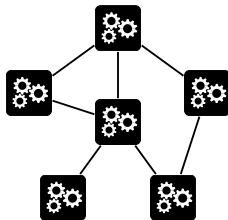
Contribution



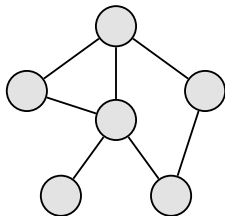
Contribution



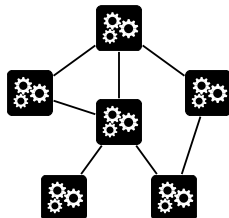
DISTR. REGISTER AUTOMATA



Contribution

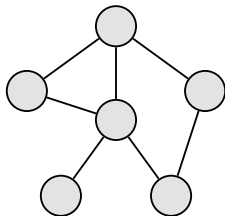


DISTR. REGISTER AUTOMATA

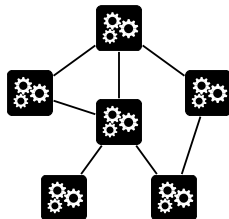


$$\text{Gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

Contribution



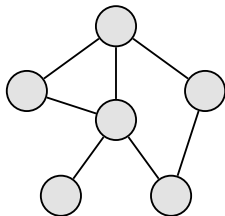
DISTR. REGISTER AUTOMATA



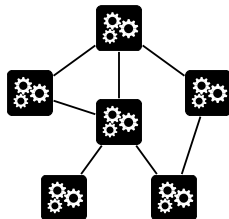
$$\text{Gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

- ▶ Finite-state & registers

Contribution



DISTR. REGISTER AUTOMATA

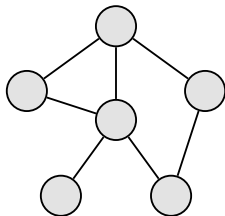


$$\text{Gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

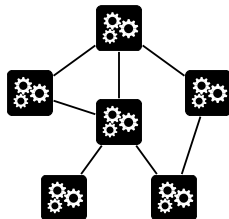
- ▶ Finite-state & registers
- ▶ Synchronous execution

Contribution

FUNCTIONAL FIXPOINT LOGIC
restricted to ordered graphs



DISTR. REGISTER AUTOMATA



$$\text{Gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

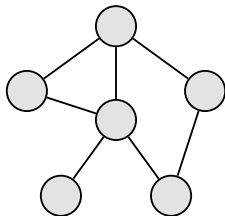
- ▶ Finite-state & registers
- ▶ Synchronous execution

Contribution

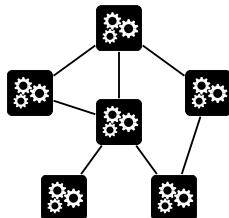
FUNCTIONAL FIXPOINT LOGIC
restricted to ordered graphs



DISTR. REGISTER AUTOMATA



$$\text{pfp} \left[\begin{array}{l} f_1: \varphi_1(f_1, f_2, \text{IN}, \text{OUT}) \\ f_2: \varphi_2(f_1, f_2, \text{IN}, \text{OUT}) \end{array} \right] \psi$$

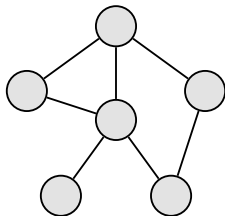


$$\text{gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

- ▶ Finite-state & registers
- ▶ Synchronous execution

Contribution

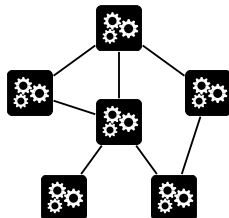
FUNCTIONAL FIXPOINT LOGIC
restricted to ordered graphs



$$\text{pfp} \left[\begin{array}{l} f_1: \varphi_1(f_1, f_2, \text{IN}, \text{OUT}) \\ f_2: \varphi_2(f_1, f_2, \text{IN}, \text{OUT}) \end{array} \right] \psi$$

EQUIVALENT

DISTR. REGISTER AUTOMATA

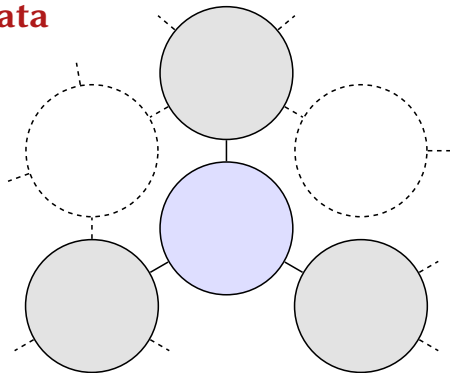


$$\text{gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

- ▶ Finite-state & registers
- ▶ Synchronous execution

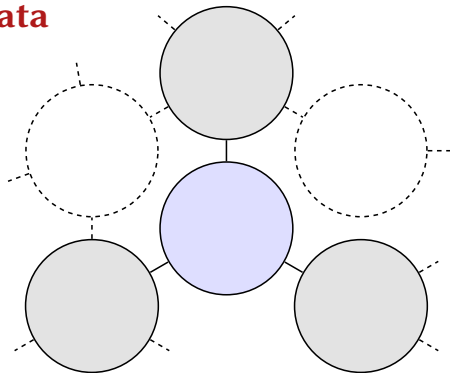
Distributed register automata

Distributed register automata



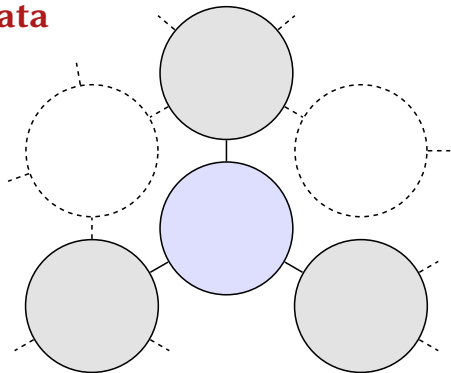
Distributed register automata

- ▶ Connected, undirected network



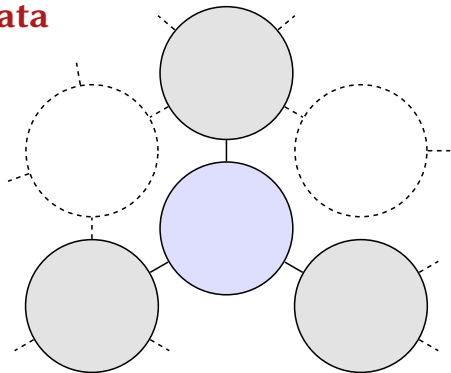
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution



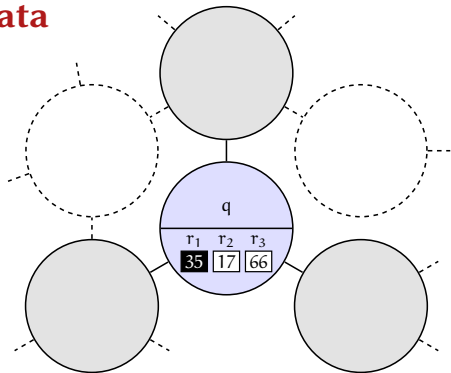
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ **Unique identifiers** in \mathbb{N}



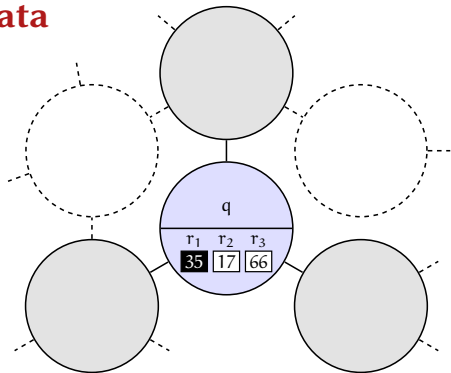
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ **Unique identifiers** in \mathbb{N}



Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ **Unique identifiers** in \mathbb{N}

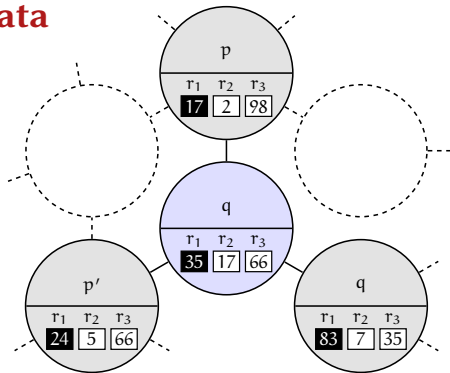


$Q = \{p, \dots, q'\}$ \leftarrow states

$R = \{r_1, r_2, r_3\}$ \leftarrow registers

Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ **Unique identifiers** in \mathbb{N}

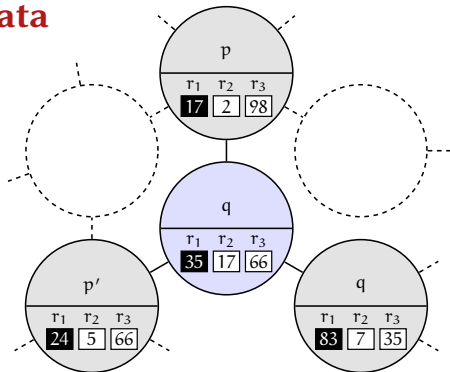


$Q = \{p, \dots, q'\} \leftarrow$ states

$R = \{r_1, r_2, r_3\} \leftarrow$ registers

Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ **Unique identifiers** in \mathbb{N}



q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

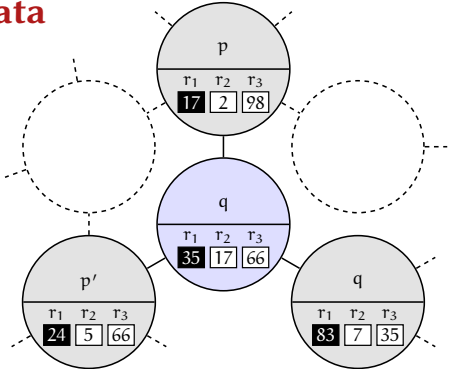
$Q = \{p, \dots, q'\} \leftarrow$ states

$R = \{r_1, r_2, r_3\} \leftarrow$ registers

Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}

 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$



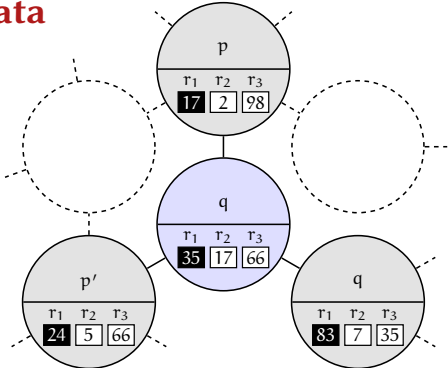
q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

$Q = \{p, \dots, q'\} \leftarrow \text{states}$
 $R = \{r_1, r_2, r_3\} \leftarrow \text{registers}$

Distributed register automata

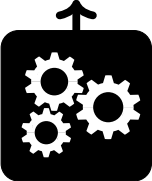
- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}

 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$



q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

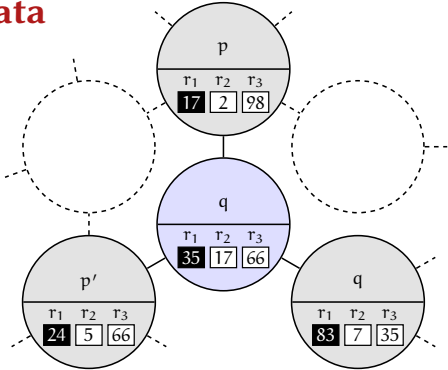
$Q = \{p, \dots, q'\} \leftarrow$ states
 $R = \{r_1, r_2, r_3\} \leftarrow$ registers



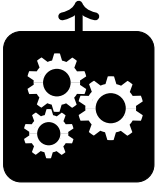
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}

 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$



q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

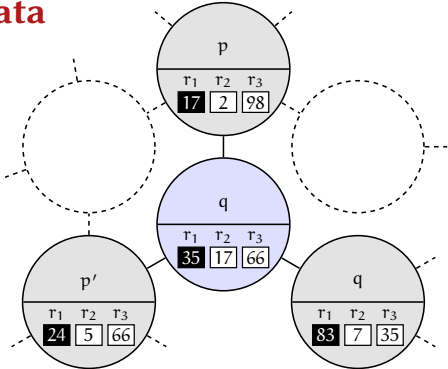


$Q = \{p, \dots, q'\} \leftarrow \text{states}$
 $R = \{r_1, r_2, r_3\} \leftarrow \text{registers}$

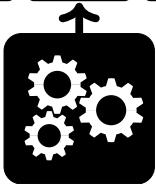
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}

 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$



q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

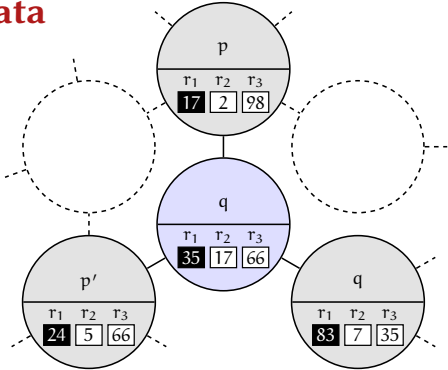


$Q = \{p, \dots, q'\} \leftarrow \text{states}$
 $R = \{r_1, r_2, r_3\} \leftarrow \text{registers}$

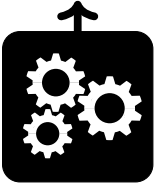
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}

 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$



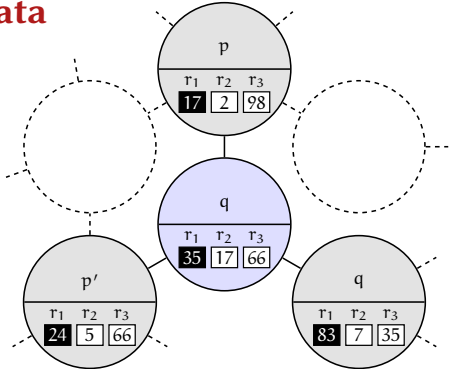
q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35



$Q = \{p, \dots, q'\} \leftarrow$ states
 $R = \{r_1, r_2, r_3\} \leftarrow$ registers

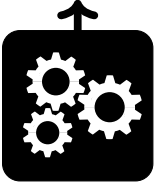
Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}



 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$

q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35



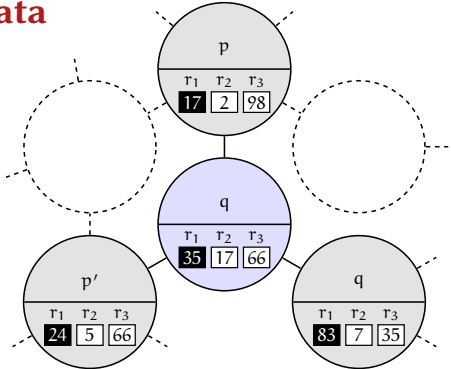
→

q'
r ₁ r ₂ r ₃
35 2 83

$Q = \{p, \dots, q'\} \leftarrow \text{states}$
 $R = \{r_1, r_2, r_3\} \leftarrow \text{registers}$

Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}



 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$

q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

$Q = \{p, \dots, q'\}$ \leftarrow states
 $R = \{r_1, r_2, r_3\}$ \leftarrow registers

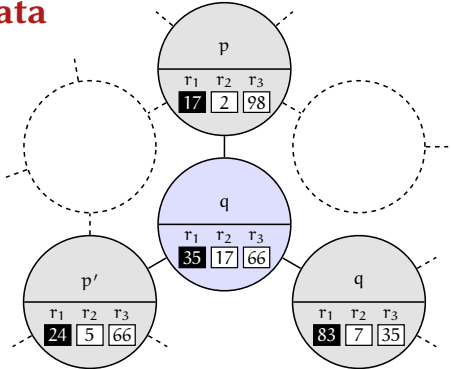
Transition maker



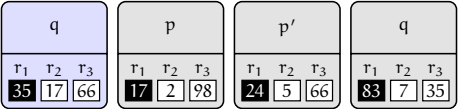
q'
r ₁ r ₂ r ₃
35 2 83

Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}

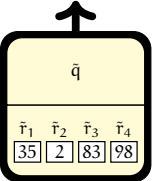


 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$

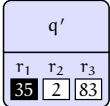


$Q = \{p, \dots, q'\} \leftarrow \text{states}$
 $R = \{r_1, r_2, r_3\} \leftarrow \text{registers}$

Transition maker

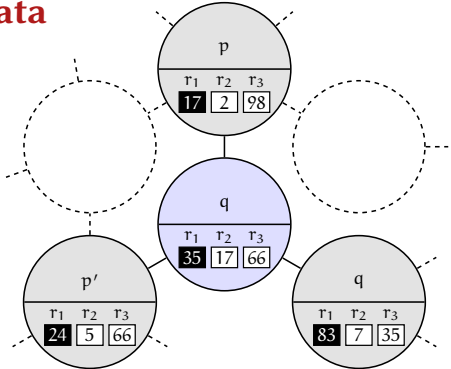


\mapsto



Distributed register automata

- ▶ Connected, undirected network
- ▶ Synchronous execution
- ▶ Unique identifiers in \mathbb{N}



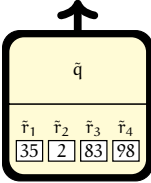
 : $(Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$

q	p	p'	q
r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃	r ₁ r ₂ r ₃
35 17 66	17 2 98	24 5 66	83 7 35

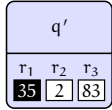
$Q = \{p, \dots, q'\} \leftarrow \text{states}$
 $R = \{r_1, r_2, r_3\} \leftarrow \text{registers}$

Transition maker can:

- ▶ compare registers (<),
- ▶ copy register values.

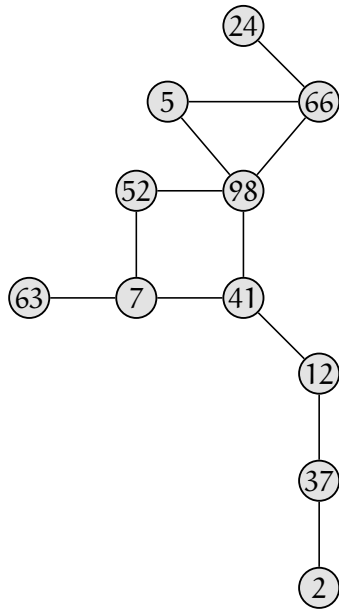


\mapsto

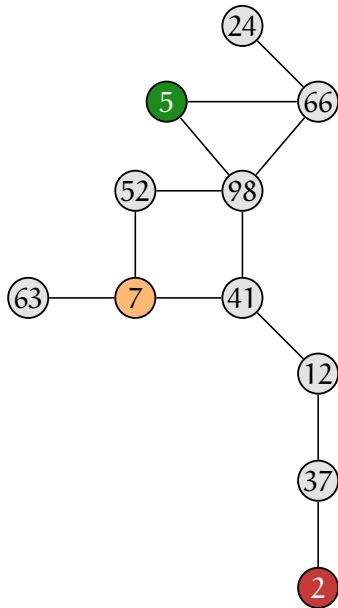


Computing a spanning tree

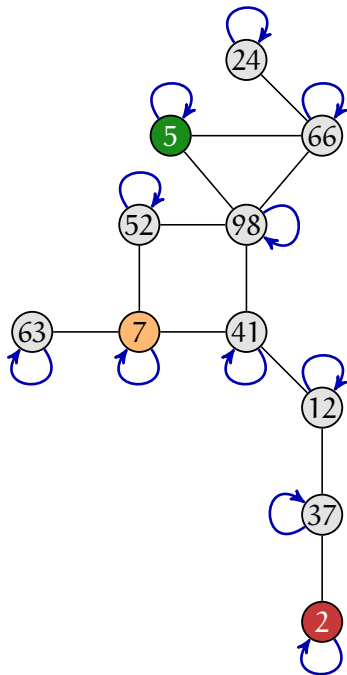
Computing a spanning tree



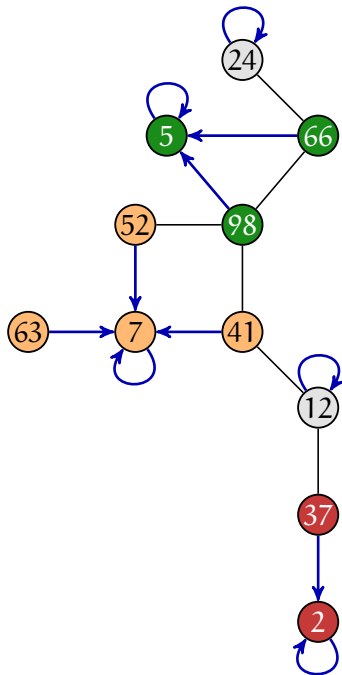
Computing a spanning tree



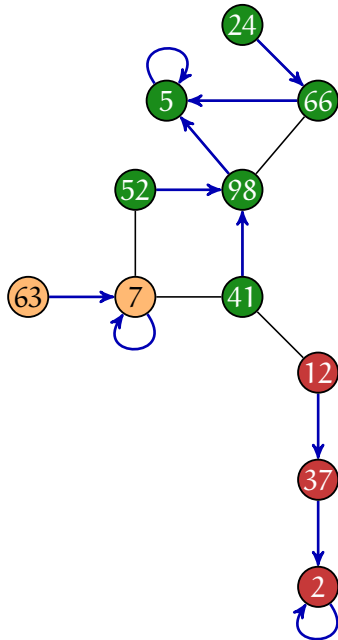
Computing a spanning tree



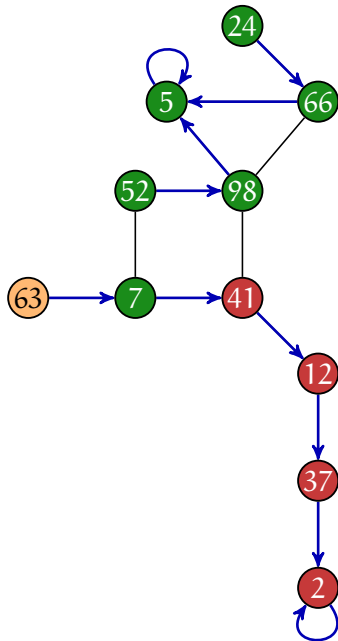
Computing a spanning tree



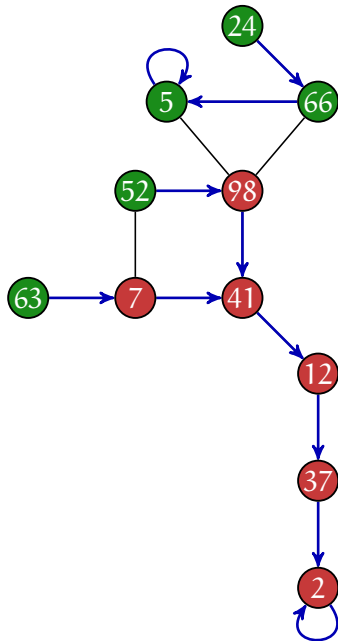
Computing a spanning tree



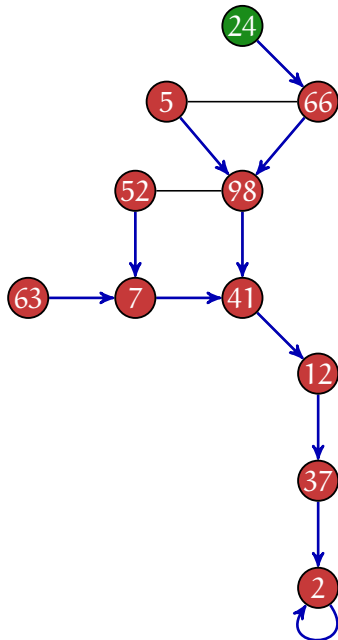
Computing a spanning tree



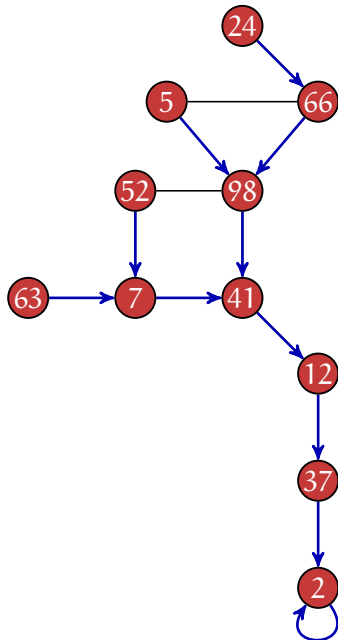
Computing a spanning tree



Computing a spanning tree

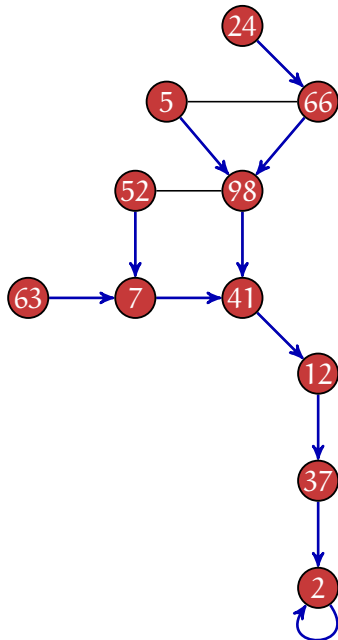


Computing a spanning tree



Computing a spanning tree

$R = \{\text{self, parent, root}\}$



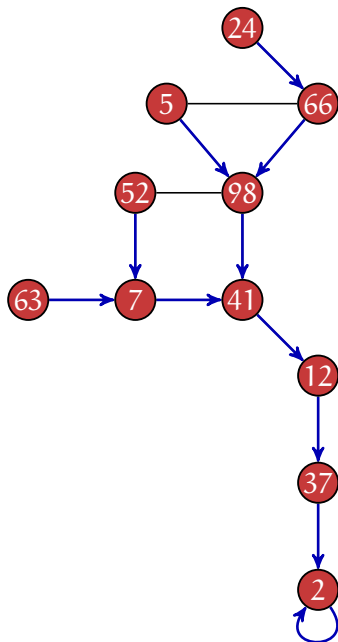
Computing a spanning tree

$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If \exists neighbor NB ($\text{NB.root} < \text{MY.root}$):

$\text{MY.parent} \leftarrow \text{NB.self}$

$\text{MY.root} \leftarrow \text{NB.root}$



Computing a spanning tree

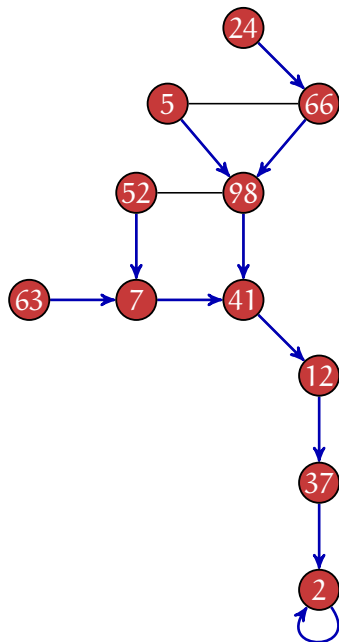
$Q = \{a, b, c\}$ with a initial

$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If \exists neighbor NB ($NB.\text{root} < MY.\text{root}$):

$MY.\text{parent} \leftarrow NB.\text{self}$

$MY.\text{root} \leftarrow NB.\text{root}$



Computing a spanning tree

$Q = \{a, b, c\}$ with a initial

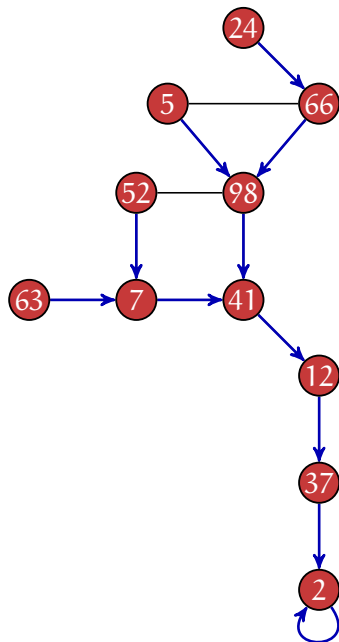
$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If \exists neighbor NB ($\text{NB.root} < \text{MY.root}$):

$\text{MY.parent} \leftarrow \text{NB.self}$

$\text{MY.root} \leftarrow \text{NB.root}$

$\text{MY.state} \leftarrow a$



Computing a spanning tree

$Q = \{a, b, c\}$ with a initial

$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If \exists neighbor NB ($NB.\text{root} < MY.\text{root}$):

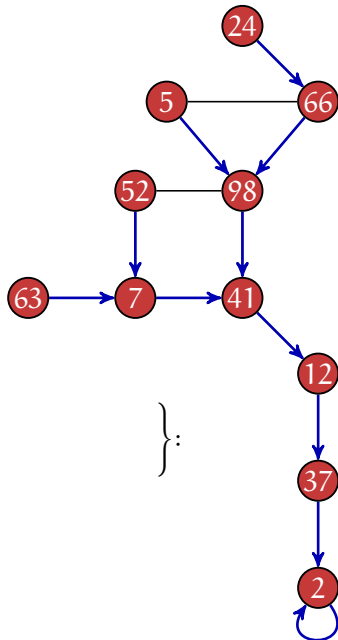
$MY.\text{parent} \leftarrow NB.\text{self}$

$MY.\text{root} \leftarrow NB.\text{root}$

$MY.\text{state} \leftarrow a$

B. If \forall neighbor NB $\left\{ \begin{array}{l} NB.\text{root} = MY.\text{root} \wedge \\ NB.\text{parent} \neq MY.\text{self} \end{array} \right\}$:

$MY.\text{state} \leftarrow b$



Computing a spanning tree

$Q = \{a, b, c\}$ with a initial

$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If \exists neighbor NB ($\text{NB.root} < \text{MY.root}$):

$\text{MY.parent} \leftarrow \text{NB.self}$

$\text{MY.root} \leftarrow \text{NB.root}$

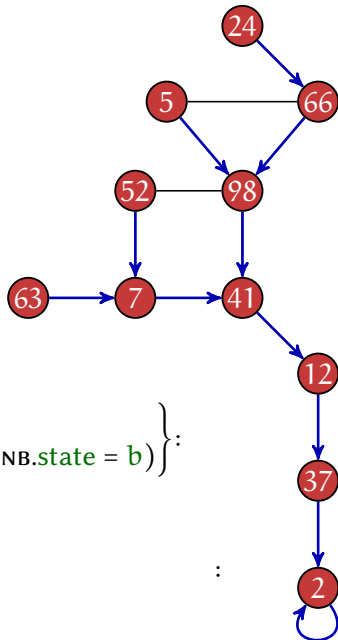
$\text{MY.state} \leftarrow a$

B. If \forall neighbor NB $\left\{ \begin{array}{l} \text{NB.root} = \text{MY.root} \wedge \\ (\text{NB.parent} \neq \text{MY.self} \vee \text{NB.state} = b) \end{array} \right\}$:

$\text{MY.state} \leftarrow b$

C. If $(\text{MY.root} = \text{MY.self} \wedge \text{MY.state} = b)$

$\text{MY.state} \leftarrow c$



Computing a spanning tree

$Q = \{a, b, c\}$ with a initial

$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If \exists neighbor NB ($\text{NB.root} < \text{MY.root}$):

$\text{MY.parent} \leftarrow \text{NB.self}$

$\text{MY.root} \leftarrow \text{NB.root}$

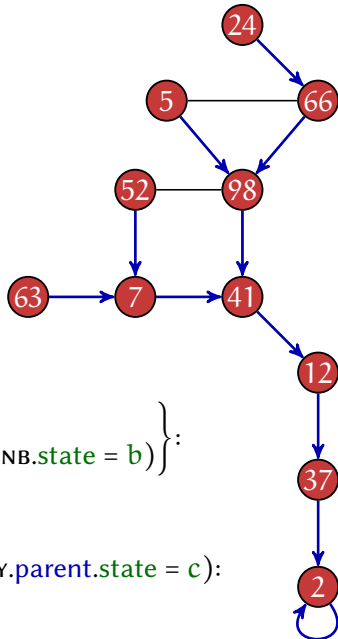
$\text{MY.state} \leftarrow a$

B. If \forall neighbor NB $\left\{ \begin{array}{l} \text{NB.root} = \text{MY.root} \wedge \\ (\text{NB.parent} \neq \text{MY.self} \vee \text{NB.state} = b) \end{array} \right\}$:

$\text{MY.state} \leftarrow b$

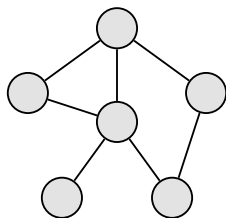
C. If $(\text{MY.root} = \text{MY.self} \wedge \text{MY.state} = b) \vee (\text{MY.parent.state} = c)$:

$\text{MY.state} \leftarrow c$



Contribution

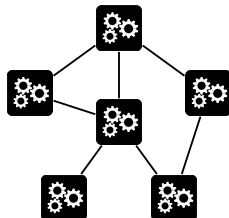
FUNCTIONAL FIXPOINT LOGIC
restricted to ordered graphs



$$\text{pfp} \left[\begin{array}{l} f_1: \varphi_1(f_1, f_2, \text{IN}, \text{OUT}) \\ f_2: \varphi_2(f_1, f_2, \text{IN}, \text{OUT}) \end{array} \right] \psi$$

EQUIVALENT

DISTR. REGISTER AUTOMATA

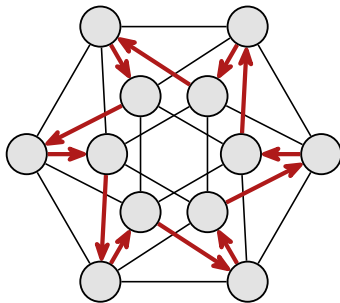


$$\text{gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

- ▶ Finite-state & registers
- ▶ Synchronous execution

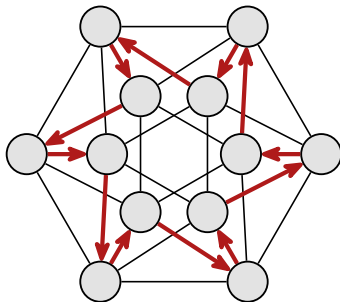
Hamiltonian cycle

Hamiltonian cycle



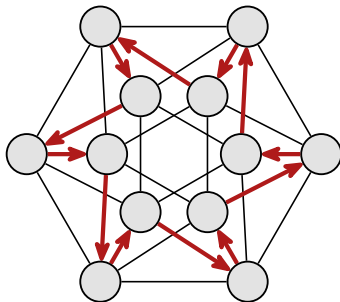
Hamiltonian cycle

$\exists f$ \wedge \wedge



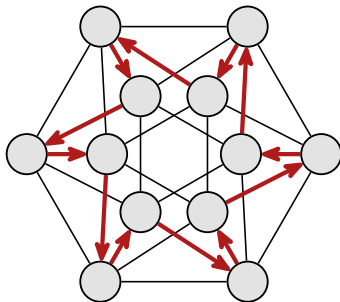
Hamiltonian cycle

$\exists f$ $\left(\overbrace{\quad \quad \quad}^{\text{f follows edges}} \quad \wedge \quad \wedge \quad \right)$



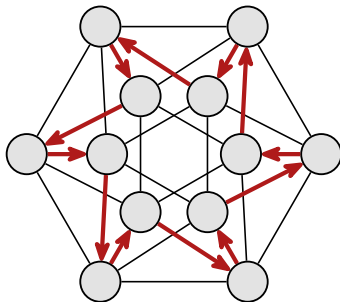
Hamiltonian cycle

$$\exists \mathbf{f} \left(\overbrace{\forall x (\mathbf{f}(x) \leftrightarrow x)}^{\text{f follows edges}} \wedge \quad \wedge \right)$$



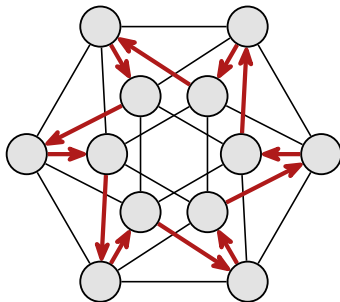
Hamiltonian cycle

$$\exists f \left(\overbrace{\forall x (f(x) \leftrightarrow x)}^{\text{f follows edges}} \wedge \overbrace{\quad \quad \quad}^{\text{f is surjective}} \wedge \quad \quad \quad \right)$$



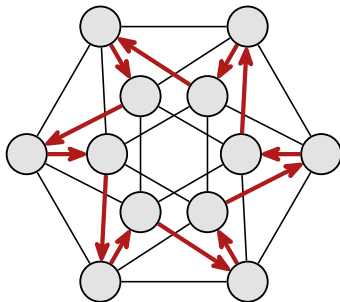
Hamiltonian cycle

$$\exists f \left(\overbrace{\forall x (f(x) \leftrightarrow x)}^{\text{f follows edges}} \wedge \overbrace{\forall y \exists x (f(x) = y)}^{\text{f is surjective}} \wedge \right)$$



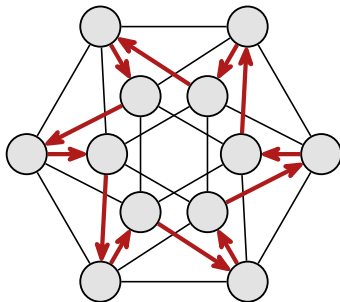
Hamiltonian cycle

$$\exists f \left(\begin{array}{l} \overbrace{\forall x (f(x) \leftrightarrow x)}^{\text{f follows edges}} \quad \wedge \quad \overbrace{\forall y \exists x (f(x) = y)}^{\text{f is surjective}} \quad \wedge \\ \forall S (\quad \Rightarrow \quad) \end{array} \right)$$



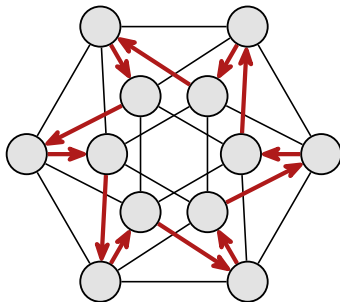
Hamiltonian cycle

$$\exists f \left(\underbrace{\forall x (f(x) \leftrightarrow x)}_{\text{f follows edges}} \wedge \underbrace{\forall y \exists x (f(x) = y)}_{\text{f is surjective}} \wedge \underbrace{\forall S \left(\begin{array}{l} S \text{ is nonempty and closed under } f \end{array} \right)}_{\Rightarrow} \right)$$



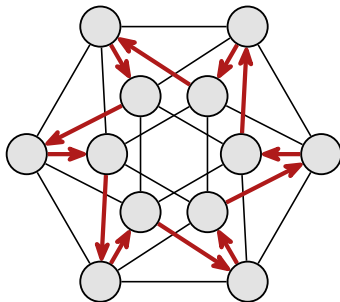
Hamiltonian cycle

$$\exists f \left(\underbrace{\forall x (f(x) \leftrightarrow x)}_{\text{f follows edges}} \wedge \underbrace{\forall y \exists x (f(x) = y)}_{\text{f is surjective}} \wedge \underbrace{\forall S \left(\begin{array}{l} S \text{ is nonempty and closed under } f \\ \Rightarrow S \text{ covers all} \end{array} \right)} \right)$$



Hamiltonian cycle

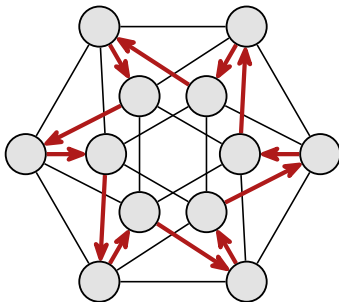
$$\exists f \left(\underbrace{\forall x (f(x) \leftrightarrow x)}_{\text{f follows edges}} \wedge \underbrace{\forall y \exists x (f(x) = y)}_{\text{f is surjective}} \wedge \underbrace{\forall S \left(\begin{array}{l} S \text{ is nonempty and closed under } f \\ \Rightarrow \forall x (x \in S) \end{array} \right)}_{\text{S covers all}} \right)$$



Hamiltonian cycle

$$\exists f \left(\underbrace{\forall x (f(x) \leftrightarrow x)}_{f \text{ follows edges}} \wedge \underbrace{\forall y \exists x (f(x) = y)}_{f \text{ is surjective}} \wedge \underbrace{\forall S \left(\left[\exists x (x \in S) \right] \Rightarrow \forall x (x \in S) \right)}_{S \text{ is nonempty and closed under } f} \right)$$

$\underbrace{\qquad\qquad\qquad}_{S \text{ covers all}}$



Functional fixpoint logic

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

$$\mathbf{pfp} \left[\begin{array}{c} f_1 \\ \vdots \\ f_n \end{array} \right] \psi$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

Binds the
function
variables
 f_1, \dots, f_n .

$$\text{pfp} \left[\begin{array}{c} f_1 \\ \vdots \\ f_n \end{array} \right] \psi$$

Functional fixpoint logic

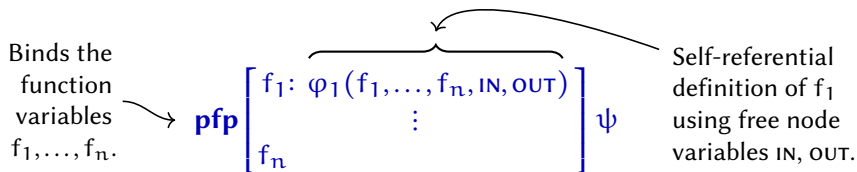
Extends first-order logic with a **partial fixpoint** operator:

Binds the
function
variables
 f_1, \dots, f_n .

$$\mathbf{pfp} \left[\begin{array}{l} f_1: \varphi_1(f_1, \dots, f_n, \text{IN}, \text{OUT}) \\ \vdots \\ f_n \end{array} \right] \psi$$

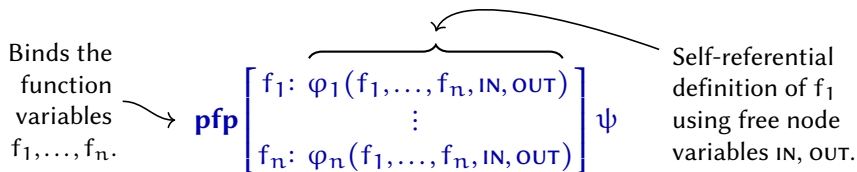
Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:



Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:



Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

Binds the
function
variables
 f_1, \dots, f_n .

$$\mathbf{pfp} \left[\begin{array}{c} f_1: \varphi_1(f_1, \dots, f_n, \text{IN}, \text{OUT}) \\ \vdots \\ f_n: \varphi_n(f_1, \dots, f_n, \text{IN}, \text{OUT}) \end{array} \right] \psi$$
A diagram showing the pfp operator. The text "Binds the function variables f1, ..., fn." has an arrow pointing to the pfp symbol. The pfp symbol is followed by a large square bracket containing a list of function definitions: f1: phi1(f1, ..., fn, IN, OUT), a vertical ellipsis, and fn: phi_n(f1, ..., fn, IN, OUT). To the right of the bracket is the Greek letter psi. A curved arrow points from the psi symbol back to the first function definition, indicating a self-referential definition.

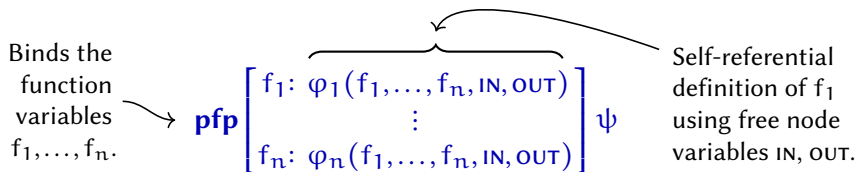
Self-referential
definition of f_1
using free node
variables IN, OUT.

To compute the partial fixpoint:

$$\left(\begin{array}{l} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{array} \right)$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

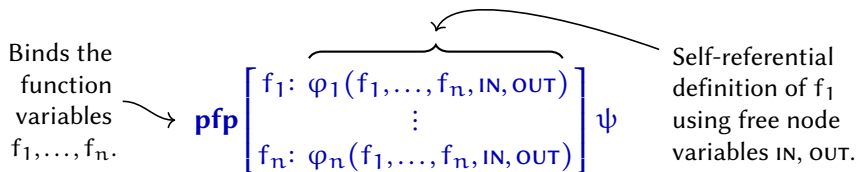


To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix}$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

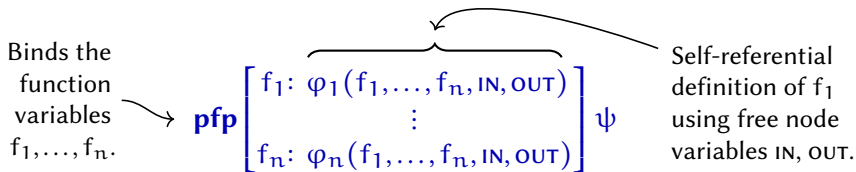


To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix}$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

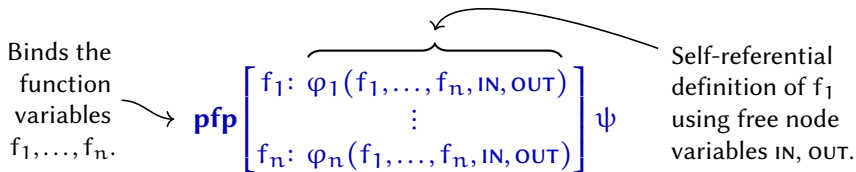


To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \dots$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:



To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \dots \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix}$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

Binds the
function
variables
 f_1, \dots, f_n .

$$\mathbf{pfp} \left[\begin{array}{c} f_1: \varphi_1(f_1, \dots, f_n, \text{IN}, \text{OUT}) \\ \vdots \\ f_n: \varphi_n(f_1, \dots, f_n, \text{IN}, \text{OUT}) \end{array} \right] \psi$$

Self-referential
definition of f_1
using free node
variables IN, OUT.

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \dots \quad \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix} = \begin{cases} \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} & \text{if } \exists k: \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} = \begin{pmatrix} f_1^{k+1} \\ \vdots \\ f_n^{k+1} \end{pmatrix} \end{cases}$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

Binds the
function
variables
 f_1, \dots, f_n .

$$\mathbf{pfp} \left[\begin{array}{c} f_1: \varphi_1(f_1, \dots, f_n, \text{IN}, \text{OUT}) \\ \vdots \\ f_n: \varphi_n(f_1, \dots, f_n, \text{IN}, \text{OUT}) \end{array} \right] \psi$$

Self-referential
definition of f_1
using free node
variables IN, OUT.

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \dots$$

$$\begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix} = \begin{cases} \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} & \text{if } \exists k: \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} = \begin{pmatrix} f_1^{k+1} \\ \vdots \\ f_n^{k+1} \end{pmatrix} \\ \begin{pmatrix} \text{id} \\ \vdots \\ \text{id} \end{pmatrix} & \text{otherwise} \end{cases}$$

Functional fixpoint logic

Extends first-order logic with a **partial fixpoint** operator:

Binds the function variables f_1, \dots, f_n .

$$\mathbf{pfp} \left[\begin{array}{c} f_1: \varphi_1(f_1, \dots, f_n, \text{IN}, \text{OUT}) \\ \vdots \\ f_n: \varphi_n(f_1, \dots, f_n, \text{IN}, \text{OUT}) \end{array} \right] \psi$$

Self-referential definition of f_1 using free node variables IN, OUT.

To compute the partial fixpoint:

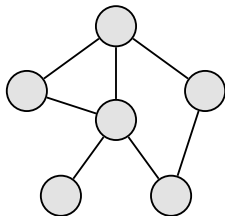
$$\begin{pmatrix} f_1^0 = \text{id} \\ \vdots \\ f_n^0 = \text{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \dots \quad \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix} = \begin{cases} \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} & \text{if } \exists k: \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} = \begin{pmatrix} f_1^{k+1} \\ \vdots \\ f_n^{k+1} \end{pmatrix} \\ \begin{pmatrix} \text{id} \\ \vdots \\ \text{id} \end{pmatrix} & \text{otherwise} \end{cases}$$

On ordered graphs:

pfp can express **quantification** over **functions** and **sets**.

Contribution

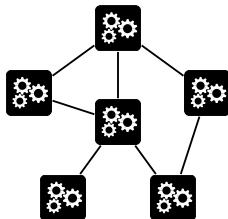
FUNCTIONAL FIXPOINT LOGIC
restricted to ordered graphs



$$\text{pfp} \left[\begin{array}{l} f_1: \varphi_1(f_1, f_2, \text{IN}, \text{OUT}) \\ f_2: \varphi_2(f_1, f_2, \text{IN}, \text{OUT}) \end{array} \right] \psi$$



DISTR. REGISTER AUTOMATA



$$\text{gear icon} : (Q \times \mathbb{N}^R)^+ \rightarrow Q \times \mathbb{N}^R$$

- ▶ Finite-state & registers
- ▶ Synchronous execution

Perspectives

Perspectives

LOGICAL DESCRIPTIONS:

- ▶ A tool to specify and synthesize distributed algorithms?

Perspectives

LOGICAL DESCRIPTIONS:

- ▶ A tool to specify and synthesize distributed algorithms?
- ▶ The key to a **complexity theory** for distributed computing?

Perspectives

LOGICAL DESCRIPTIONS:

- ▶ A tool to specify and synthesize distributed algorithms?
- ▶ The key to a **complexity theory** for distributed computing?
 - ▶ Forces us to formalize our models of computation.

Perspectives

LOGICAL DESCRIPTIONS:

- ▶ A tool to specify and synthesize distributed algorithms?
- ▶ The key to a **complexity theory** for distributed computing?
 - ▶ Forces us to formalize our models of computation.
 - ▶ Can help to identify natural and robust classes of algorithms.

Perspectives

LOGICAL DESCRIPTIONS:

- ▶ A tool to specify and synthesize distributed algorithms?
- ▶ The key to a **complexity theory** for distributed computing?
 - ▶ Forces us to formalize our models of computation.
 - ▶ Can help to identify natural and robust classes of algorithms.
 - ▶ Transfers classical complexity theory to the distributed setting.

Perspectives

LOGICAL DESCRIPTIONS:

- ▶ A tool to specify and synthesize distributed algorithms?
- ▶ The key to a **complexity theory** for distributed computing?
 - ▶ Forces us to formalize our models of computation.
 - ▶ Can help to identify natural and robust classes of algorithms.
 - ▶ Transfers classical complexity theory to the distributed setting.

Thanks!