Distributed Algorithms as Register Automata

Benedikt Bollig Patricia Bouyer Fabian Reiter

LSV, University of Paris-Saclay

5 December 2018 @ LaBRI, Bordeaux

Identifiers in Registers

Describing Network Algorithms with Logic

Benedikt Bollig Patricia Bouyer Fabian Reiter

LSV, University of Paris-Saclay

5 December 2018 @ LaBRI, Bordeaux

Fagin's theorem (1974)

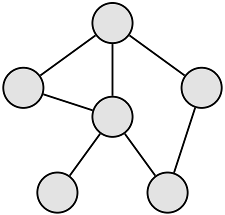
Fagin's theorem (1974)

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

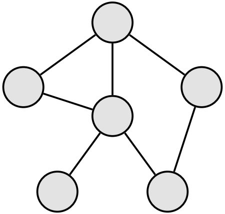
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



Fagin's theorem (1974)

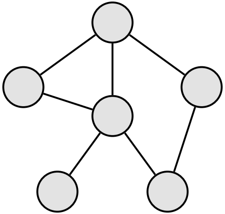
∃ SECOND-ORDER LOGIC



Example: Hamiltonian path

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



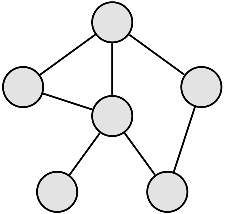
Example: Hamiltonian path

$\exists R\,\big($

$\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



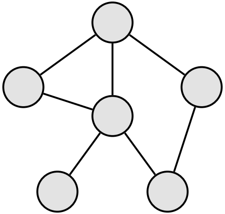
Example: Hamiltonian path

$\exists R \,\big($ "R is a strict total order" $\wedge$

$\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC



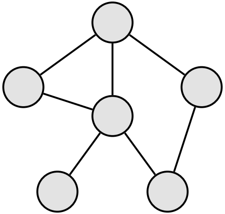
Example: Hamiltonian path

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

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES



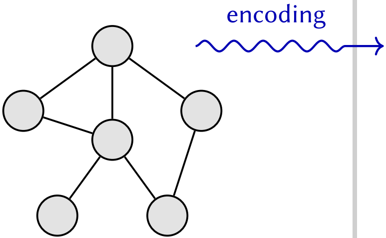
Example: Hamiltonian path

$\exists R\,\big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES



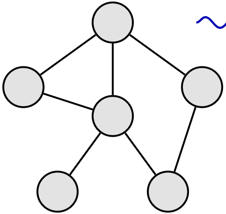
Example: Hamiltonian path

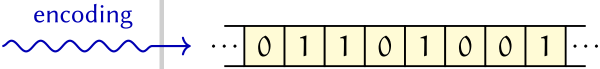
$\exists R \big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES



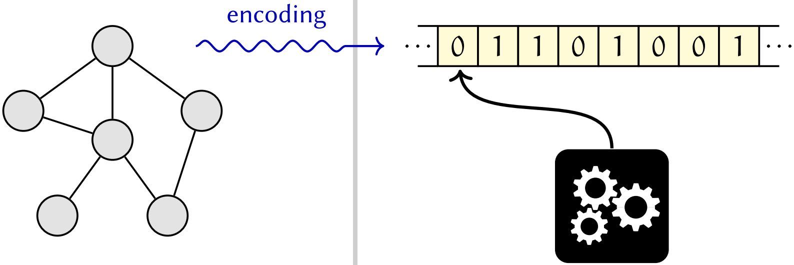


Example: Hamiltonian path

$\exists R\,\big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC | **NP TURING MACHINES**



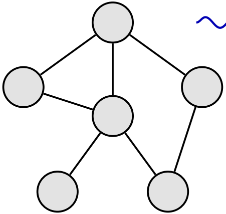
Example: Hamiltonian path

$\exists R \,\big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

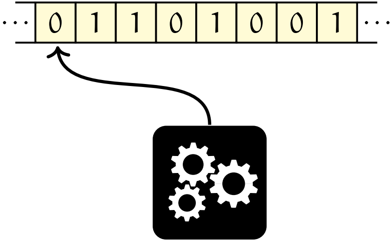
Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES



encoding



Example: Hamiltonian path

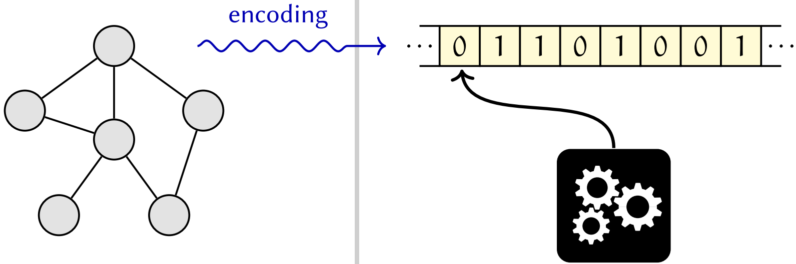
$\exists R\,\big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

- Nondeterministic moves

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC

NP TURING MACHINES



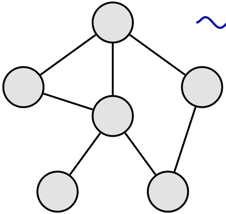
Example: Hamiltonian path

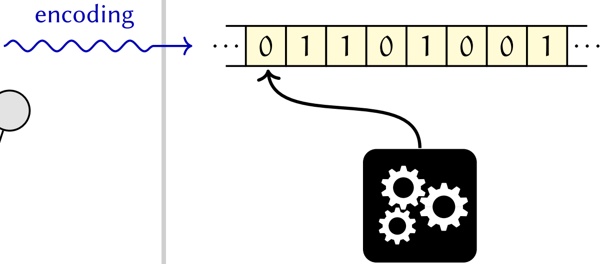
$\exists R\,\big($ "R is a strict total order" $\wedge$
"R-successors are adjacent" $\big)$

- Nondeterministic moves
- Polynomial running time

Fagin's theorem (1974)

∃ SECOND-ORDER LOGIC ⟺ EQUIVALENT ⟺ NP TURING MACHINES



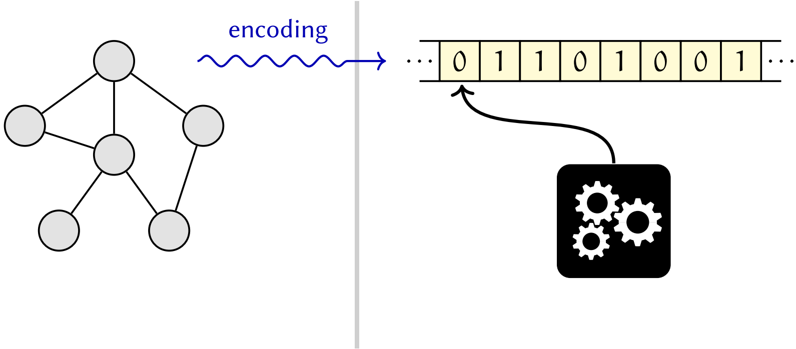


Example: Hamiltonian path

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

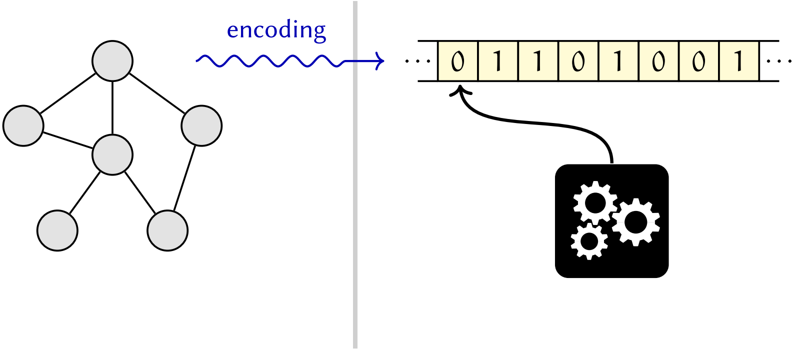
- Nondeterministic moves
- Polynomial running time

Descriptive complexity



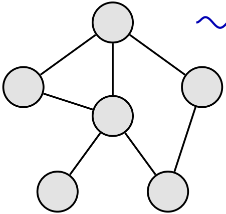
Descriptive complexity

SOME LOGICAL FORMALISM

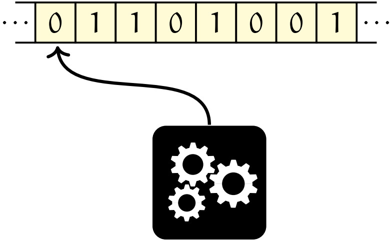


Descriptive complexity

SOME LOGICAL FORMALISM EQUIVALENT

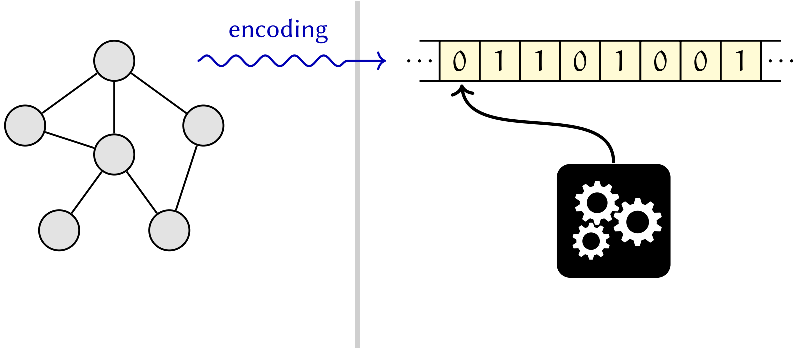


encoding



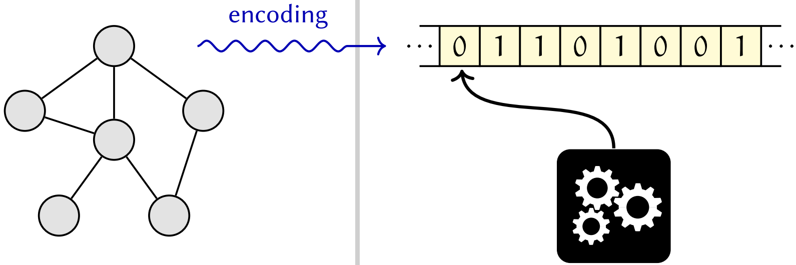
Descriptive complexity

SOME LOGICAL FORMALISM ⟺ EQUIVALENT ⟺ SOME ABSTRACT MACHINES



Descriptive complexity

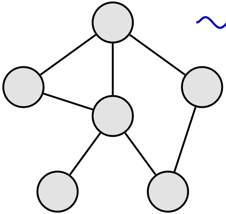
SOME LOGICAL FORMALISM EQUIVALENT SOME ABSTRACT MACHINES

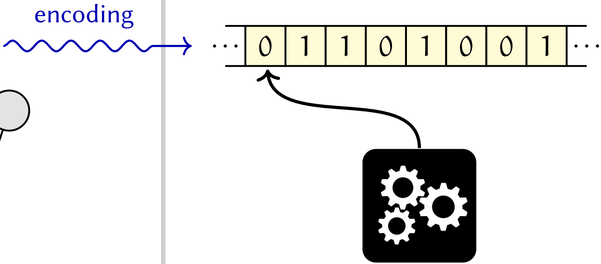


Formula class Φ

Descriptive complexity

SOME LOGICAL FORMALISM EQUIVALENT SOME ABSTRACT MACHINES





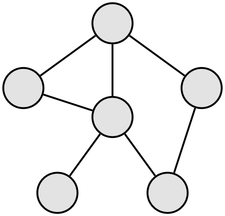
Formula class Φ

Algorithm class $\mathcal{A}$

Descriptive distributed complexity

Descriptive distributed complexity

SOME LOGICAL FORMALISM

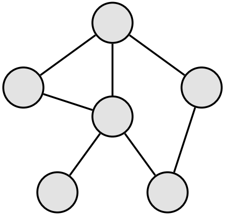


Formula class Φ

Descriptive distributed complexity

SOME LOGICAL FORMALISM

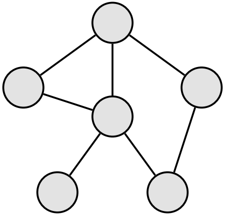
EQUIVALENT



Formula class Φ

Descriptive distributed complexity

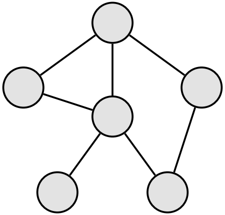
SOME LOGICAL FORMALISM EQUIVALENT COMMUNICATING MACHINES



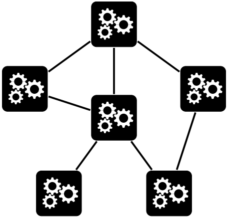
Formula class Φ

Descriptive distributed complexity

SOME LOGICAL FORMALISM — EQUIVALENT — COMMUNICATING MACHINES

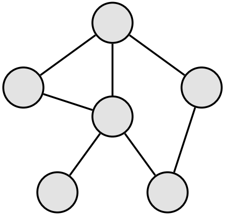


Formula class Φ

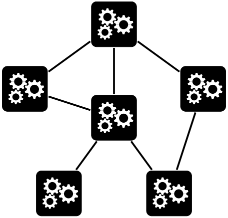


Descriptive distributed complexity

SOME LOGICAL FORMALISM — EQUIVALENT — COMMUNICATING MACHINES

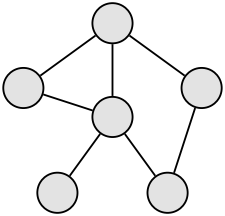


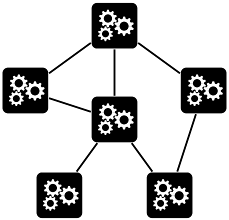
Formula class Φ



Distributed algorithm class $\mathcal{A}$

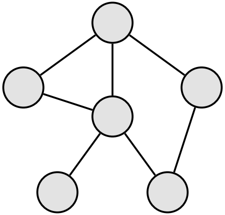
Contribution

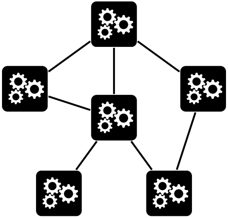




Contribution

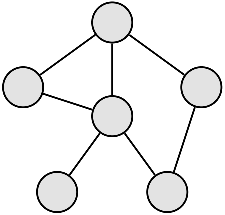
DISTR. REGISTER AUTOMATA

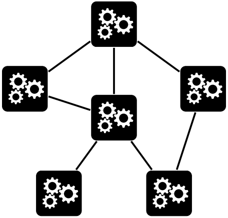




Contribution

DISTR. REGISTER AUTOMATA

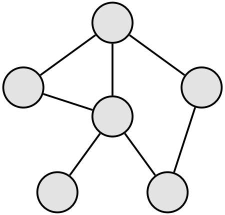


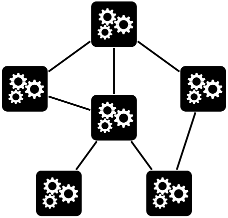


 $: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

Contribution

DISTR. REGISTER AUTOMATA



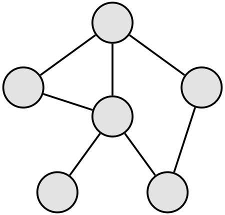


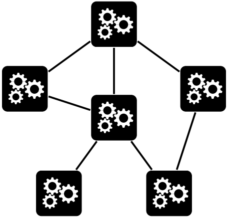
 : $(Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

- Finite-state & registers

Contribution

DISTR. REGISTER AUTOMATA





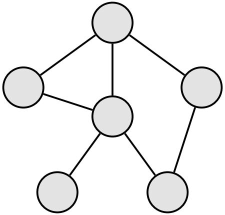
 $: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

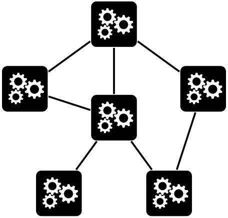
- Finite-state & registers
- Synchronous execution

Contribution

FUNCTIONAL FIXPOINT LOGIC ⟺ EQUIVALENT ⟺ DISTR. REGISTER AUTOMATA

restricted to ordered graphs





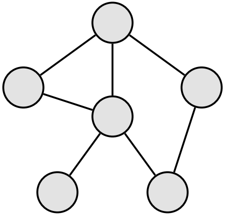
 $: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

- Finite-state & registers
- Synchronous execution

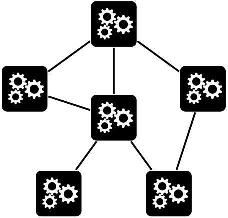
Contribution

FUNCTIONAL FIXPOINT LOGIC ⟺ EQUIVALENT ⟺ **DISTR. REGISTER AUTOMATA**

restricted to ordered graphs



$$\mathbf{pfp}\begin{bmatrix} f_1\colon\ \varphi_1(f_1, f_2, \textsc{in}, \textsc{out}) \\ f_2\colon\ \varphi_2(f_1, f_2, \textsc{in}, \textsc{out}) \end{bmatrix}\psi$$



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

- Finite-state & registers
- Synchronous execution

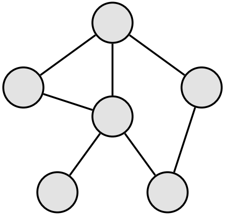
Contribution

FUNCTIONAL FIXPOINT LOGIC

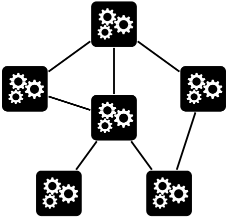
restricted to ordered graphs

EQUIVALENT

DISTR. REGISTER AUTOMATA



$$\mathbf{pfp}\begin{bmatrix} f_1\colon\ \varphi_1(f_1, f_2, \textsc{in}, \textsc{out}) \\ f_2\colon\ \varphi_2(f_1, f_2, \textsc{in}, \textsc{out}) \end{bmatrix}\psi$$

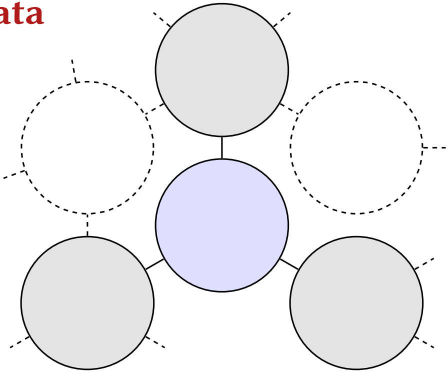


 $: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

- Finite-state & registers
- Synchronous execution

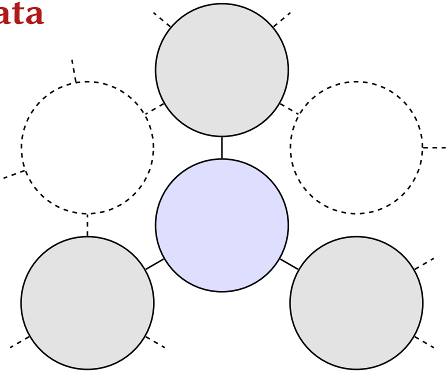
Distributed register automata

Distributed register automata



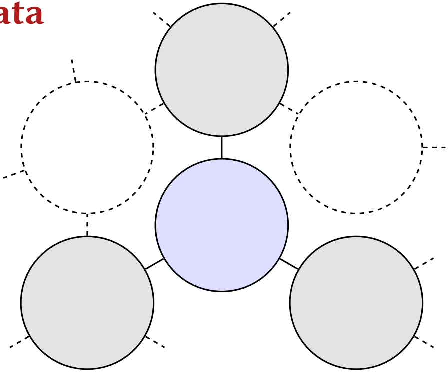
Distributed register automata

- Connected, undirected network



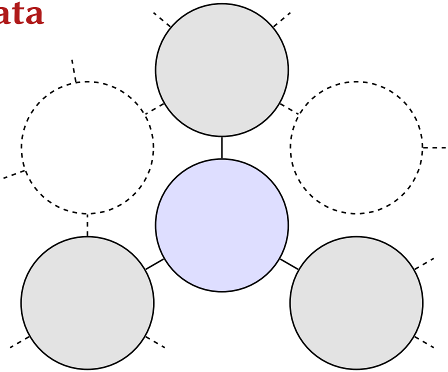
Distributed register automata

- Connected, undirected network
- Synchronous execution



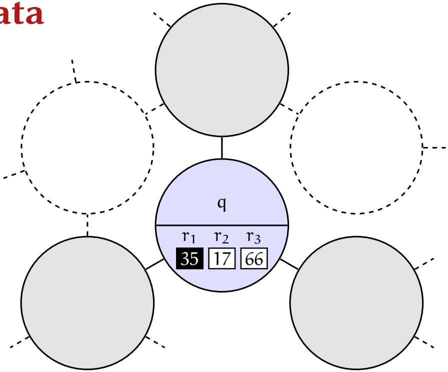
Distributed register automata

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



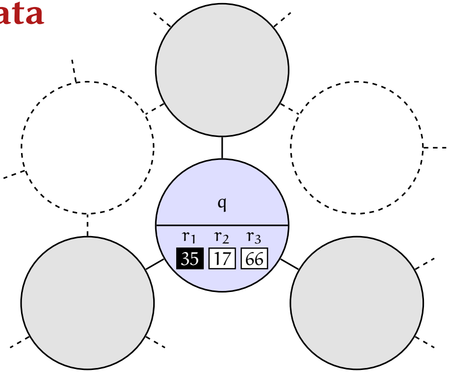
Distributed register automata

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



Distributed register automata

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

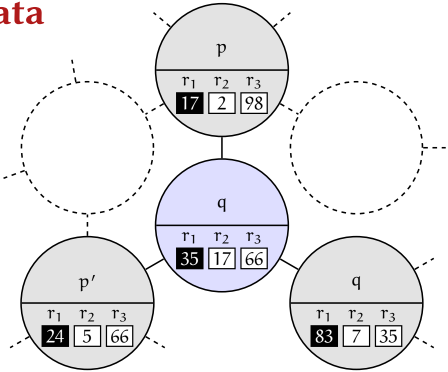


$Q = \{p, \ldots, q'\}$ ⇜ states

$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata

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

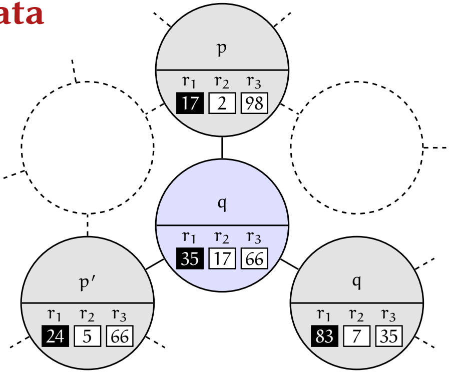


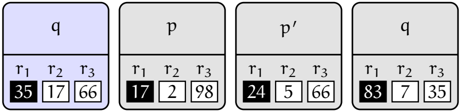
$Q = \{p, \ldots, q'\}$ ⇜ states

$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata

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





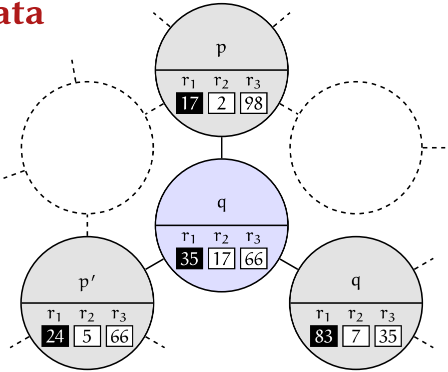
$Q = \{p, \dots, q'\}$ ⇜ states

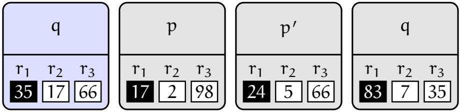
$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata

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

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





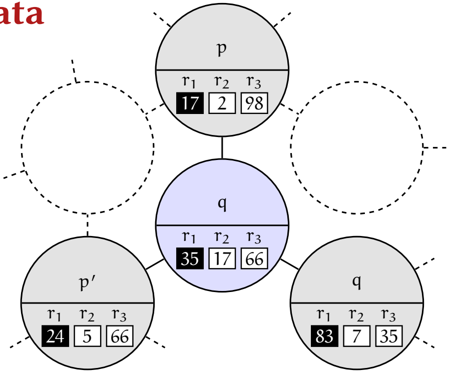
$Q = \{p, \ldots, q'\}$ ⇜ states

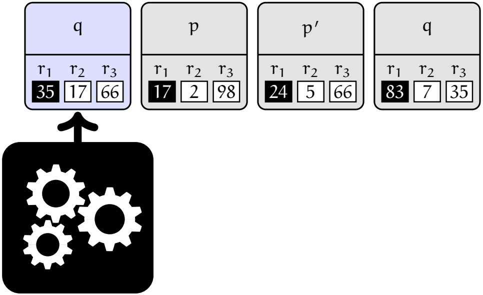
$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata

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

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

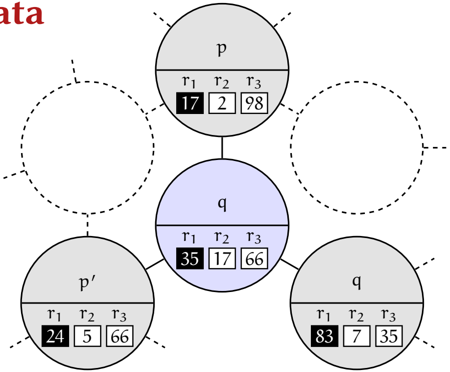




$Q = \{p, \ldots, q'\}$ ⇜ states

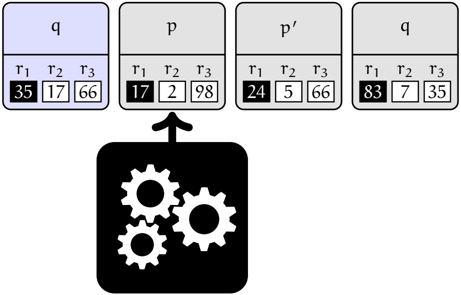
$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata



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

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

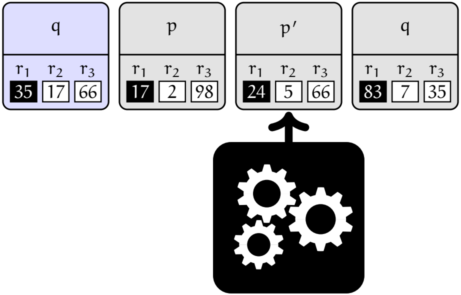


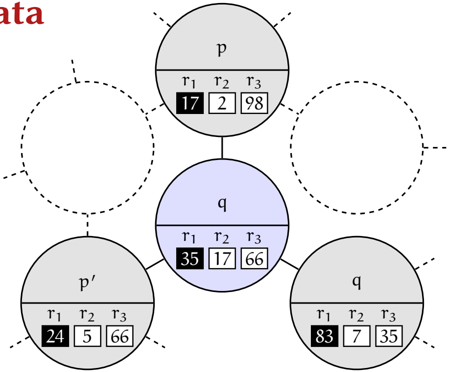
$Q = \{p, \ldots, q'\}$ ⇜ states
$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata

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

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





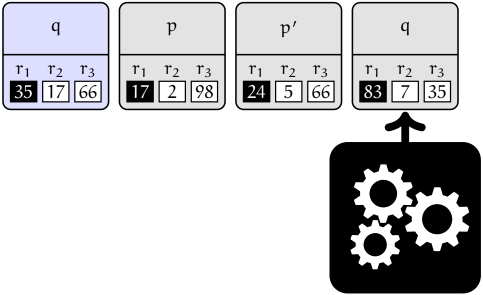
$Q = \{p, \ldots, q'\}$ ⇜ states

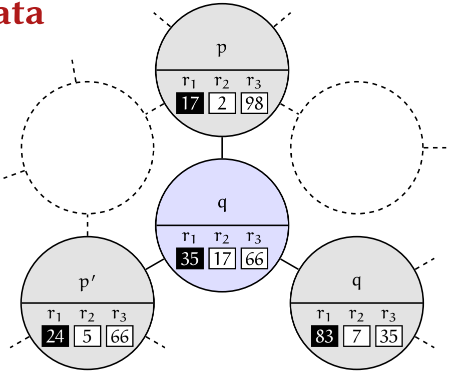
$R = \{r_1, r_2, r_3\}$ ⇜ registers

Distributed register automata

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

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



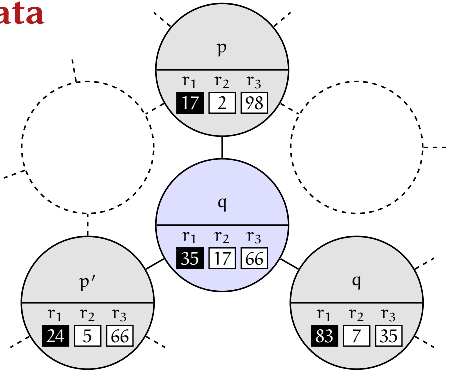


$Q = \{p, \dots, q'\}$ ⇝ states

$R = \{r_1, r_2, r_3\}$ ⇝ registers

Distributed register automata

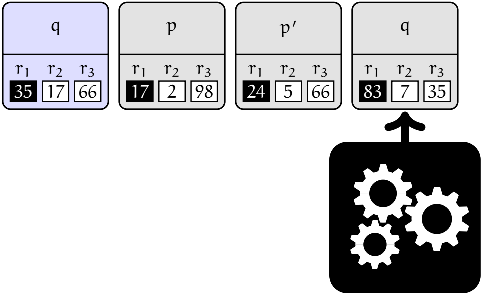
- Connected, undirected network
- Synchronous execution
- Unique identifiers in $\mathbb{N}$



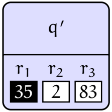
$: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

$Q = \{p, \ldots, q'\}$ ⇜ states

$R = \{r_1, r_2, r_3\}$ ⇜ registers

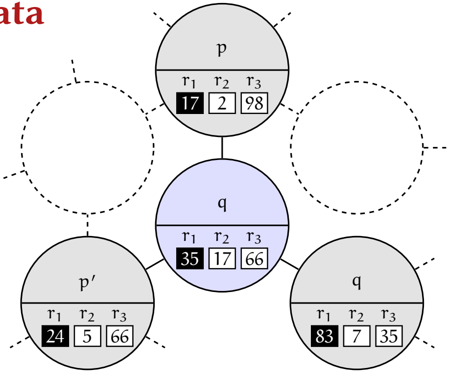


$\mapsto$



Distributed register automata

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

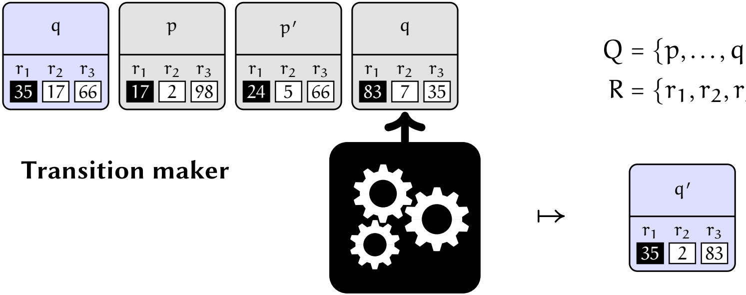


$: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

$Q = \{p, \ldots, q'\}$ ⇜ states

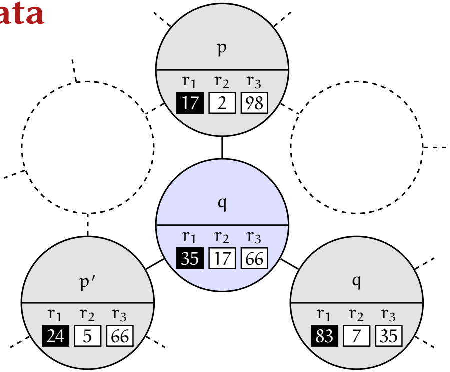
$R = \{r_1, r_2, r_3\}$ ⇜ registers

Transition maker



Distributed register automata

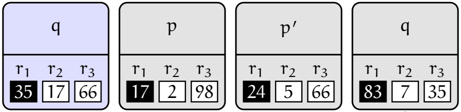
- Connected, undirected network
- Synchronous execution
- Unique identifiers in $\mathbb{N}$



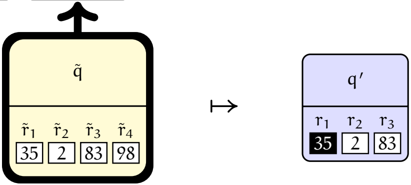
: $(Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

$Q = \{p, \ldots, q'\}$ ⇜ states

$R = \{r_1, r_2, r_3\}$ ⇜ registers

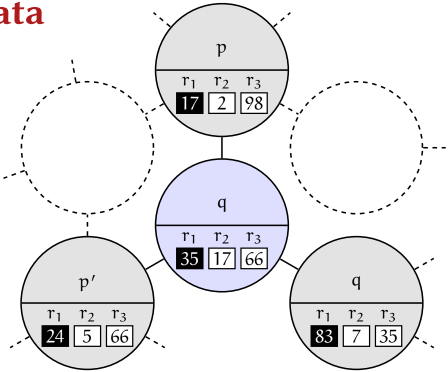


Transition maker



Distributed register automata

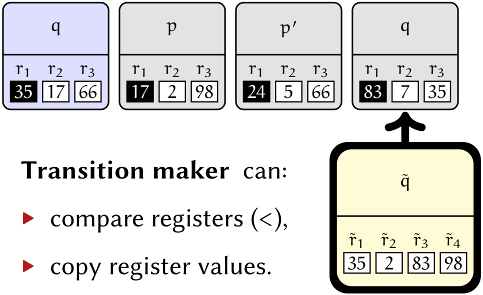
- Connected, undirected network
- Synchronous execution
- Unique identifiers in $\mathbb{N}$



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

$Q = \{p, \ldots, q'\}$ ⇜ states

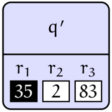
$R = \{r_1, r_2, r_3\}$ ⇜ registers



Transition maker can:

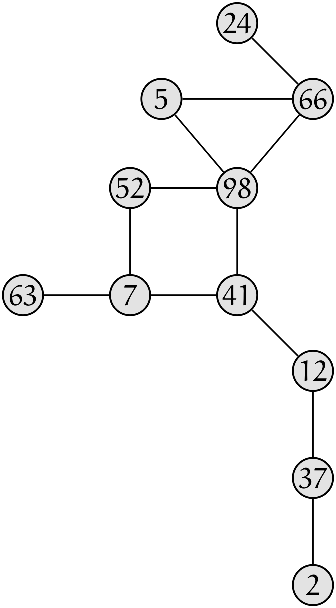
- compare registers (<),
- copy register values.

$\mapsto$

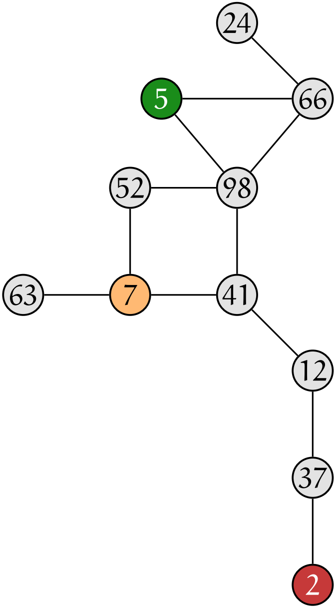


Computing a spanning tree

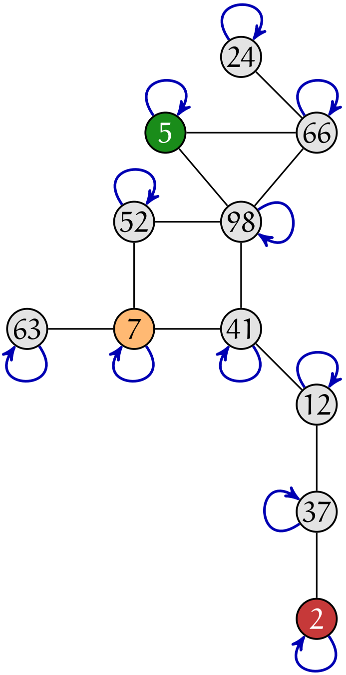
Computing a spanning tree



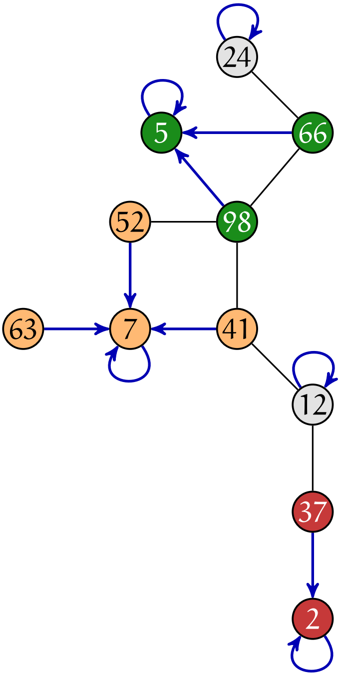
Computing a spanning tree



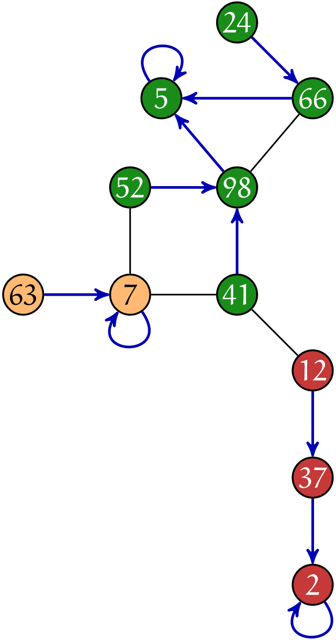
Computing a spanning tree



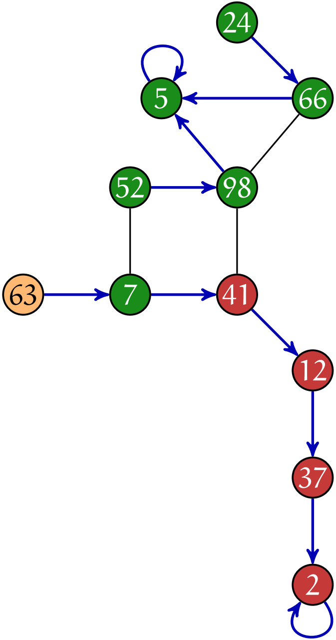
Computing a spanning tree



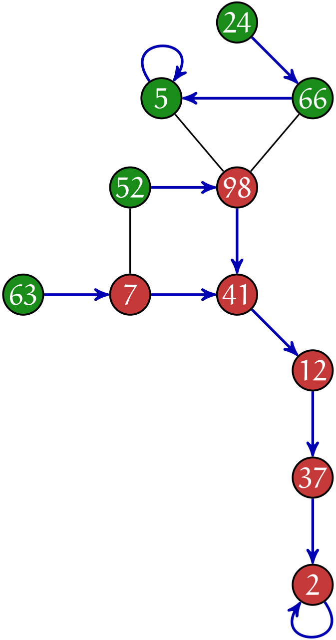
Computing a spanning tree



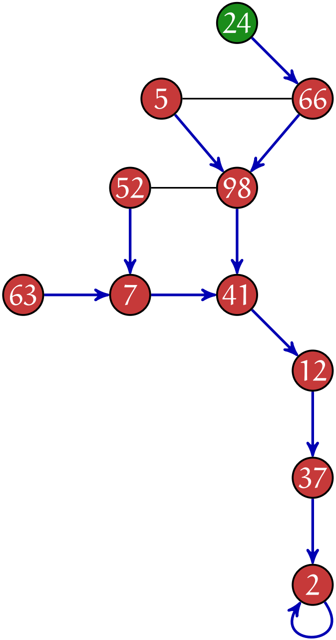
Computing a spanning tree



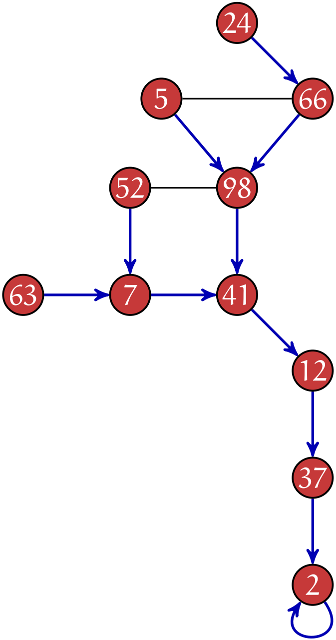
Computing a spanning tree



Computing a spanning tree

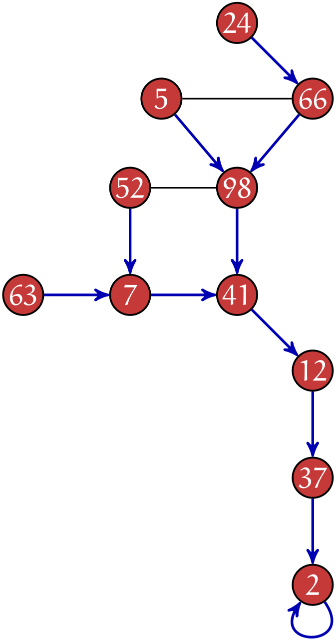


Computing a spanning tree



Computing a spanning tree

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



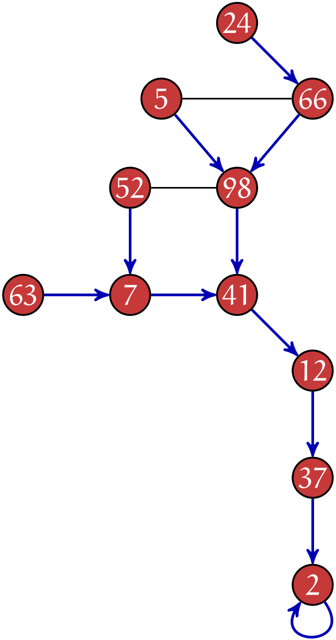
Computing a spanning tree

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

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

MY.parent ← NB.self

MY.root ← NB.root



Computing a spanning tree

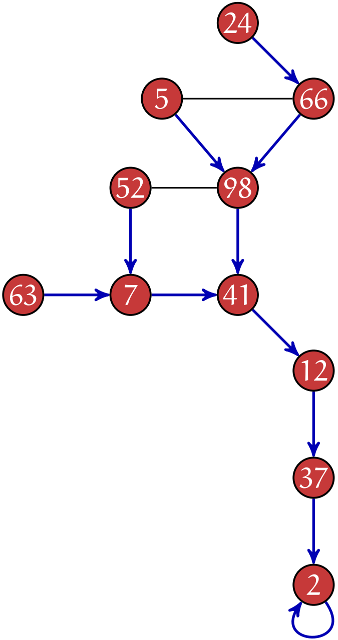
$Q = \{a, b, c\}$ with a initial

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

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

MY.parent $\leftarrow$ NB.self

MY.root $\leftarrow$ NB.root



Computing a spanning tree

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

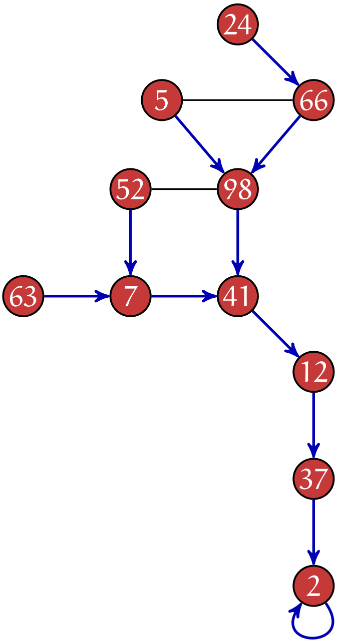
$R = \{\text{self}, \text{parent}, \text{root}\}$

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

MY.parent ← NB.self

MY.root ← NB.root

MY.state ← a



Computing a spanning tree

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

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

A. If $\exists$ neighbor $\textsc{nb}$ $(\textsc{nb}.\text{root} < \textsc{my}.\text{root})$:

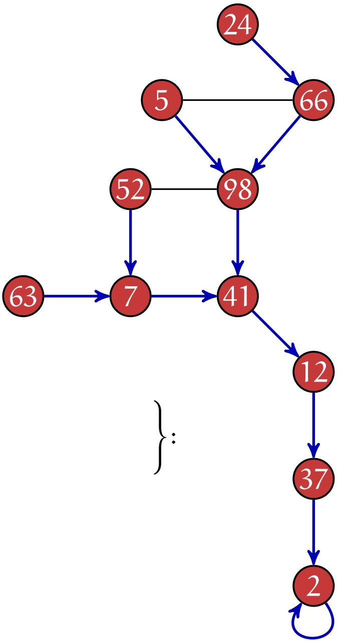
$\textsc{my}.\text{parent} \leftarrow \textsc{nb}.\text{self}$

$\textsc{my}.\text{root} \leftarrow \textsc{nb}.\text{root}$

$\textsc{my}.\text{state} \leftarrow a$

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

$\textsc{my}.\text{state} \leftarrow b$



Computing a spanning tree

$Q = \{a, b, c\}$ with a initial
$R = \{\text{self}, \text{parent}, \text{root}\}$

A. If $\exists$ neighbor $\textsc{nb}$ $(\textsc{nb}.\text{root} < \textsc{my}.\text{root})$:

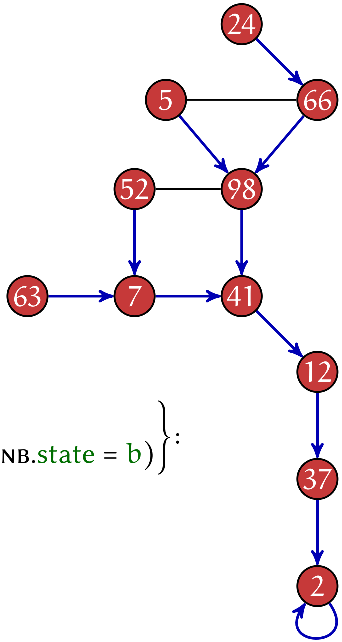
$\textsc{my}.\text{parent} \leftarrow \textsc{nb}.\text{self}$

$\textsc{my}.\text{root} \leftarrow \textsc{nb}.\text{root}$

$\textsc{my}.\text{state} \leftarrow a$

B. If $\forall$ neighbor $\textsc{nb}$ $\left\{\begin{matrix}\textsc{nb}.\text{root} = \textsc{my}.\text{root} \;\wedge \\ (\textsc{nb}.\text{parent} \neq \textsc{my}.\text{self} \vee \textsc{nb}.\text{state} = b)\end{matrix}\right\}$:

$\textsc{my}.\text{state} \leftarrow b$



Computing a spanning tree

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

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

A. If $\exists$ neighbor $\textsc{nb}$ $(\textsc{nb}.\text{root} < \textsc{my}.\text{root})$:

$\textsc{my}.\text{parent} \leftarrow \textsc{nb}.\text{self}$

$\textsc{my}.\text{root} \leftarrow \textsc{nb}.\text{root}$

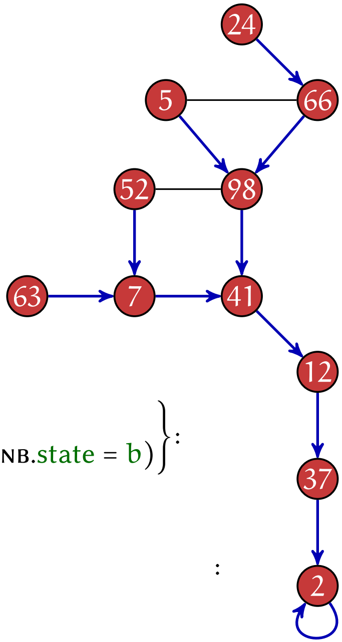
$\textsc{my}.\text{state} \leftarrow a$

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

$\textsc{my}.\text{state} \leftarrow b$

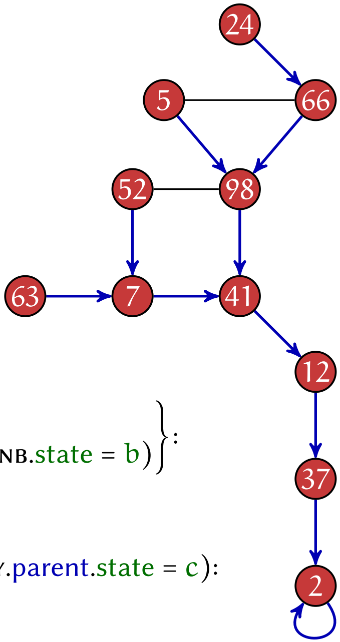
C. If $(\textsc{my}.\text{root} = \textsc{my}.\text{self} \wedge \textsc{my}.\text{state} = b)$:

$\textsc{my}.\text{state} \leftarrow c$



Computing a spanning tree

$$Q = \{a, b, c\} \quad \text{with } a \text{ initial}$$
$$R = \{\text{self}, \text{parent}, \text{root}\}$$



A. If $\exists$ neighbor $\textsc{nb}$ $(\textsc{nb}.\text{root} < \textsc{my}.\text{root})$:

$\textsc{my}.\text{parent} \leftarrow \textsc{nb}.\text{self}$

$\textsc{my}.\text{root} \leftarrow \textsc{nb}.\text{root}$

$\textsc{my}.\text{state} \leftarrow a$

B. If $\forall$ neighbor $\textsc{nb}$ $\left\{\begin{matrix}\textsc{nb}.\text{root} = \textsc{my}.\text{root} \ \wedge \\ (\textsc{nb}.\text{parent} \neq \textsc{my}.\text{self} \vee \textsc{nb}.\text{state} = b)\end{matrix}\right\}$:

$\textsc{my}.\text{state} \leftarrow b$

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

$\textsc{my}.\text{state} \leftarrow c$

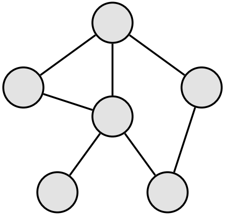
Contribution

FUNCTIONAL FIXPOINT LOGIC

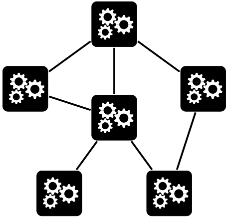
restricted to ordered graphs

EQUIVALENT

DISTR. REGISTER AUTOMATA



$$\mathbf{pfp}\begin{bmatrix} f_1\colon\ \varphi_1(f_1, f_2, \textsc{in}, \textsc{out}) \\ f_2\colon\ \varphi_2(f_1, f_2, \textsc{in}, \textsc{out}) \end{bmatrix}\psi$$

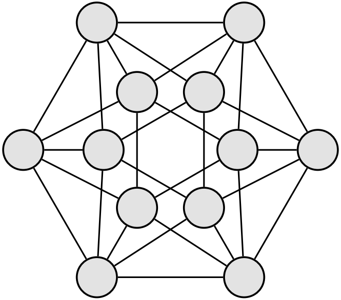


 $: (Q \times \mathbb{N}^R)^+ \to Q \times \mathbb{N}^R$

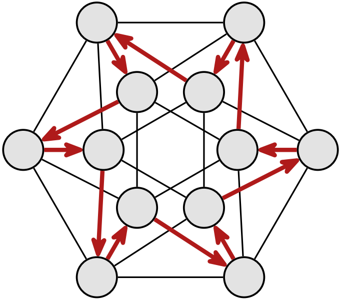
- Finite-state & registers
- Synchronous execution

Hamiltonian cycle

Hamiltonian cycle

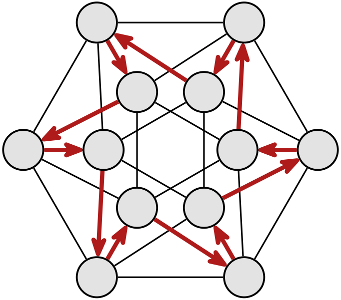


Hamiltonian cycle



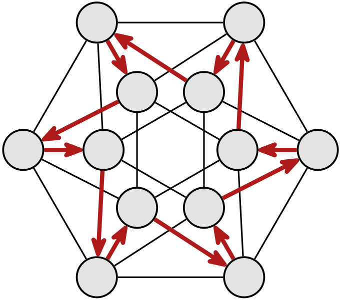
Hamiltonian cycle

$$\exists f \left(\qquad\qquad \wedge \qquad\qquad \wedge \qquad\qquad \right)$$



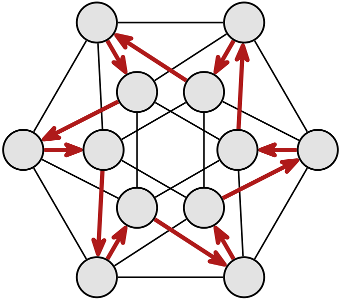
Hamiltonian cycle

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



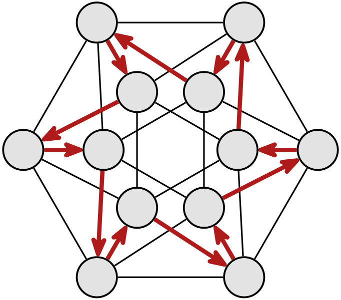
Hamiltonian cycle

$$\exists f \left(\overbrace{\forall x \big(f(x) \rightharpoondown x\big)}^{\text{f follows edges}} \quad \wedge \qquad\qquad \wedge \qquad\qquad \right)$$



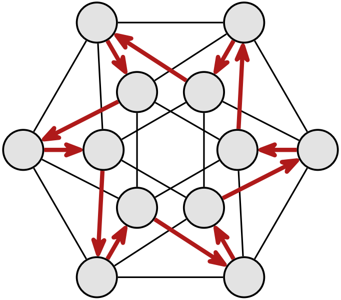
Hamiltonian cycle

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



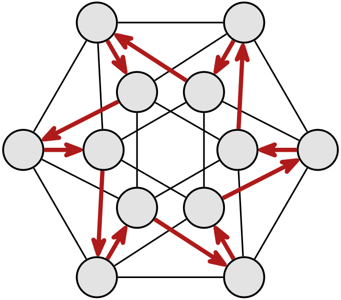
Hamiltonian cycle

$$\exists \mathbf{f} \left(\overbrace{\forall x \big(\mathbf{f}(x) \leftrightharpoondownup x\big)}^{\text{f follows edges}} \quad \wedge \quad \overbrace{\forall y \, \exists x \big(\mathbf{f}(x) = y\big)}^{\text{f is surjective}} \quad \wedge \qquad \right)$$



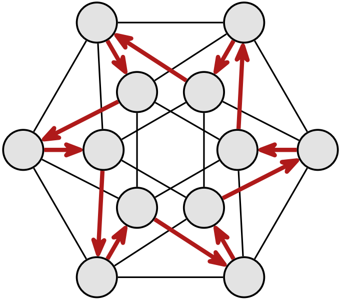
Hamiltonian cycle

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



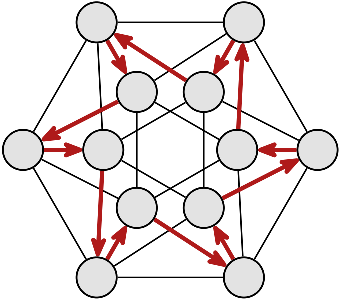
Hamiltonian cycle

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



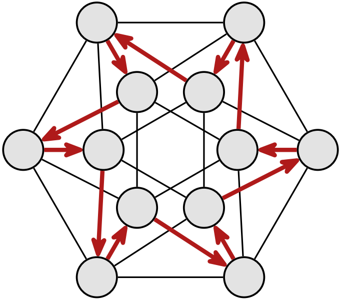
Hamiltonian cycle

$$\exists f \left(\begin{array}{l} \overbrace{\forall x\big(f(x) \mathrel{\text{⥊}} x\big)}^{f \text{ follows edges}} \quad \wedge \quad \overbrace{\forall y\, \exists x\big(f(x) = y\big)}^{f \text{ is surjective}} \quad \wedge \\ \forall S\Big(\underbrace{\qquad\qquad\qquad\qquad}_{S \text{ is nonempty and closed under } f} \Rightarrow \underbrace{\qquad\qquad}_{S \text{ covers all}} \Big) \end{array} \right)$$



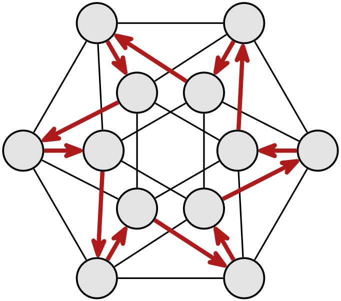
Hamiltonian cycle

$$\exists f \left(\begin{array}{l} \overbrace{\forall x \big(f(x) \leftharpoondown x\big)}^{f \text{ follows edges}} \quad \wedge \quad \overbrace{\forall y\, \exists x \big(f(x) = y\big)}^{f \text{ is surjective}} \quad \wedge \\ \forall S \Big(\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{S \text{ is nonempty and closed under } f} \Rightarrow \underbrace{\forall x (x \in S)}_{S \text{ covers all}} \Big) \end{array} \right)$$



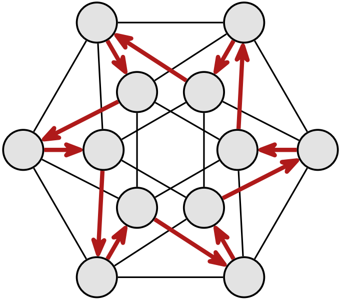
Hamiltonian cycle

$$\exists f \left(\begin{array}{l} \overbrace{\forall x\big(f(x) \rightharpoonup x\big)}^{\text{f follows edges}} \quad \wedge \quad \overbrace{\forall y\, \exists x\big(f(x) = y\big)}^{\text{f is surjective}} \quad \wedge \\ \forall S\Big(\underbrace{\big[\exists x(x \in S) \qquad\qquad\qquad\qquad \big]}_{S \text{ is nonempty and closed under f}} \Rightarrow \underbrace{\forall x(x \in S)}_{S \text{ covers all}} \Big) \end{array} \right)$$



Hamiltonian cycle

$$\exists f \left(\begin{array}{l} \overbrace{\forall x\big(f(x) \leftharpoondown x\big)}^{\text{f follows edges}} \quad \wedge \quad \overbrace{\forall y\, \exists x\big(f(x) = y\big)}^{\text{f is surjective}} \quad \wedge \\ \forall S\Big(\underbrace{\big[\exists x(x \in S) \wedge \forall y\big(y \in S \Rightarrow f(y) \in S\big)\big]}_{S \text{ is nonempty and closed under f}} \Rightarrow \underbrace{\forall x(x \in S)}_{S \text{ covers all}} \Big) \end{array} \right)$$

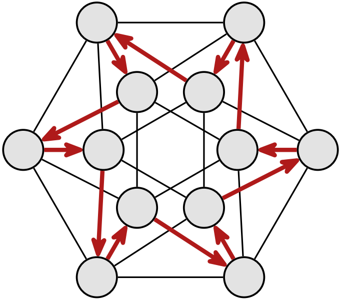


Hamiltonian cycle

$$\exists f \left(\overbrace{\forall x \big(f(x) \rightleftharpoons x \big)}^{f \text{ follows edges}} \;\wedge\; \overbrace{\forall y\, \exists x \big(f(x) = y \big)}^{f \text{ is surjective}} \;\wedge \\ \forall S \Big(\underbrace{\big[\exists x (x \in S) \wedge \forall y \big(y \in S \Rightarrow f(y) \in S \big) \big]}_{S \text{ is nonempty and closed under } f} \;\Rightarrow\; \underbrace{\forall x (x \in S)}_{S \text{ covers all}} \Big) \right)$$

function

set



Functional fixpoint logic

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

$$\mathbf{pfp}\begin{bmatrix} f_1 & & \\ & \vdots & \\ f_n & & \end{bmatrix}\psi$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \dots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1 & & \\ & \vdots & \\ f_n & & \end{bmatrix}\psi$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \ \varphi_1(f_1,\ldots,f_n,\textsc{in},\textsc{out}) \\ \vdots \\ f_n \end{bmatrix} \psi$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \overbrace{\varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out})} \\ \vdots \\ f_n \end{bmatrix} \psi$$

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

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \overbrace{\varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out})} \\ \vdots \\ f_n\colon \varphi_n(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \dots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \overbrace{\varphi_1(f_1, \dots, f_n, \mathrm{IN}, \mathrm{OUT})} \\ \vdots \\ f_n\colon \varphi_n(f_1, \dots, f_n, \mathrm{IN}, \mathrm{OUT}) \end{bmatrix} \psi$$

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

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \mathrm{id} \\ \vdots \\ f_n^0 = \mathrm{id} \end{pmatrix}$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \ \varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \\ \vdots \\ f_n\colon \ \varphi_n(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

To compute the partial fixpoint:

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

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \overbrace{\varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out})} \\ \vdots \\ f_n\colon \varphi_n(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

To compute the partial fixpoint:

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

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1 \colon & \overbrace{\varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out})} \\ & \vdots \\ f_n \colon & \varphi_n(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

To compute the partial fixpoint:

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

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon & \overbrace{\varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out})} \\ & \vdots \\ f_n\colon & \varphi_n(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \mathrm{id} \\ \vdots \\ f_n^0 = \mathrm{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \cdots \qquad \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix}$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \ldots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon \overbrace{\varphi_1(f_1, \ldots, f_n, \textsc{in}, \textsc{out})} \\ \vdots \\ f_n\colon \varphi_n(f_1, \ldots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \mathrm{id} \\ \vdots \\ f_n^0 = \mathrm{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \cdots \qquad \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix} = \begin{cases} \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} & \text{if } \exists k\colon \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} = \begin{pmatrix} f_1^{k+1} \\ \vdots \\ f_n^{k+1} \end{pmatrix} \\ & \end{cases}$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \dots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon & \overbrace{\varphi_1(f_1, \dots, f_n, \textsc{in}, \textsc{out})} \\ & \vdots \\ f_n\colon & \varphi_n(f_1, \dots, f_n, \textsc{in}, \textsc{out}) \end{bmatrix} \psi$$

Self-referential definition of f_1 using free node variables $\textsc{in}$, $\textsc{out}$.

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \mathrm{id} \\ \vdots \\ f_n^0 = \mathrm{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \cdots \qquad \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix} = \begin{cases} \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} & \text{if } \exists k\colon \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} = \begin{pmatrix} f_1^{k+1} \\ \vdots \\ f_n^{k+1} \end{pmatrix} \\ \begin{pmatrix} \mathrm{id} \\ \vdots \\ \mathrm{id} \end{pmatrix} & \text{otherwise} \end{cases}$$

Functional fixpoint logic

Extends first-order logic with a partial fixpoint operator:

Binds the function variables $f_1, \dots, f_n$.

$$\mathbf{pfp}\begin{bmatrix} f_1\colon & \overbrace{\varphi_1(f_1,\dots,f_n,\mathrm{IN},\mathrm{OUT})} \\ & \vdots \\ f_n\colon & \varphi_n(f_1,\dots,f_n,\mathrm{IN},\mathrm{OUT}) \end{bmatrix}\psi$$

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

To compute the partial fixpoint:

$$\begin{pmatrix} f_1^0 = \mathrm{id} \\ \vdots \\ f_n^0 = \mathrm{id} \end{pmatrix} \mapsto \begin{pmatrix} f_1^1 \\ \vdots \\ f_n^1 \end{pmatrix} \mapsto \begin{pmatrix} f_1^2 \\ \vdots \\ f_n^2 \end{pmatrix} \mapsto \cdots \qquad \begin{pmatrix} f_1^\infty \\ \vdots \\ f_n^\infty \end{pmatrix} = \begin{cases} \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} & \text{if } \exists k\colon \begin{pmatrix} f_1^k \\ \vdots \\ f_n^k \end{pmatrix} = \begin{pmatrix} f_1^{k+1} \\ \vdots \\ f_n^{k+1} \end{pmatrix} \\ \begin{pmatrix} \mathrm{id} \\ \vdots \\ \mathrm{id} \end{pmatrix} & \text{otherwise} \end{cases}$$

On ordered graphs:

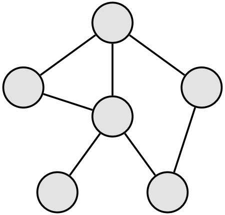
pfp can express quantification over functions and sets.

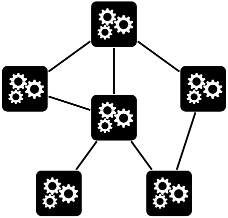
Contribution

FUNCTIONAL FIXPOINT LOGIC
restricted to ordered graphs

EQUIVALENT

DISTR. REGISTER AUTOMATA





$$\mathbf{pfp}\begin{bmatrix} f_1\colon\ \varphi_1(f_1, f_2, \textsc{in}, \textsc{out}) \\ f_2\colon\ \varphi_2(f_1, f_2, \textsc{in}, \textsc{out}) \end{bmatrix}\psi$$

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

- 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!