Emptiness Problems for Distributed Automata

Fabian Reiter

IRIF, Université Paris Diderot

September 21, 2017

Joint work with Antti Kuusisto
Distributed automata

<table>
<thead>
<tr>
<th>Transition function: $\delta$</th>
<th>$Q \times 2^Q \to Q$</th>
</tr>
</thead>
</table>

$S$: set of received states

Synchronous run:

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Distributed automata

Transition function:

\[ \delta : Q \times 2^Q \rightarrow Q \]

\( Q \) is the set of states.

- If \( S \cap \{2, 3\} \neq \emptyset \), the next state is always 0.
- If \( S \subseteq \{1\} \), the next state is always the empty set.
- Otherwise, the next state is always 1.

Synchronous run:

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
2 & 0 & 0 & 2 \\
1 & 0 & 0 & 1 \\
2 & 0 & 0 & 2 \\
3 & 0 & 0 & 3 \\
\end{array}
\]
Distributed automata

\[
\delta : Q \times 2^Q \to Q
\]

\(Q\): set of states

\[S: \text{set of received states}\]

Synchronous run:

\[\begin{array}{ccc}
0 & 0 & 0 \\
2 & 0 & 2 \\
1 & 2 & 1 \\
2 & 2 & 3 \\
2 & 2 & 1 \\
\end{array}\]

\[\cdot \cdot \cdot\]
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]

- \( Q \): set of states

S: set of received states

Synchronous run:

\[
\begin{array}{cccc}
0 & 0 & 0 & 2 \\
2 & 1 & 1 & 2 \\
2 & 3 & 0 & 2 \\
\end{array}
\]
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
\((Q: \text{set of states})\)
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
(Q: set of states)
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
\[(Q: \text{set of states}) \]
Distributed automata

Transition function:
\[ \delta: Q \times 2^Q \to Q \]
(Q: set of states)

\[
\begin{align*}
\text{if } S \subseteq \{1\} \\
\text{if } S \subseteq \{2, 3\} \\
\text{otherwise}
\end{align*}
\]

\[ S: \text{ set of received states} \]
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
(Q: set of states)

Synchronous run:

\[ S: \text{set of received states} \]
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
(Q: set of states)
Distributed automata

Transition function:
\( \delta: Q \times 2^Q \rightarrow Q \)  
\((Q: \text{set of states})\)

Synchronous run:
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
(Q: set of states)

Synchronous run:
Distributed automata

Transition function:

\[ \delta : Q \times 2^Q \rightarrow Q \]

\( Q \): set of states

Synchronous run:

\[ 0 \rightarrow 0 \rightarrow 0 \rightarrow 2 \rightarrow 0 \rightarrow 0 \rightarrow 2 \]

\[ 1 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 1 \]
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]  
(Q: set of states)

Synchronous run:
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
\( (Q: \text{set of states}) \)

Synchronous run:

\begin{align*}
\text{S: set of received states} \\
\text{always} \quad \text{always}
\end{align*}
Distributed automata

Transition function:
\[ \delta : Q \times 2^Q \rightarrow Q \]
(Q: set of states)

Synchronous run:
Emptiness problem

Automaton $A$ accepts digraph $G$ on node $v \in G$ iff $v$ visits an accepting state at some time $t \in \mathbb{N}$. 

Emptiness problem: Does automaton $A$ accept on some node in some digraph?

Simple reduction: undecidable on dipaths $\Rightarrow$ undecidable on digraphs.
Emptiness problem

Automaton $A$ accepts digraph $G$ on node $v \in G$
iff $v$ visits an accepting state at some time $t \in \mathbb{N}$.
Emptiness problem

Automaton $A$ accepts digraph $G$ on node $v \in G$ iff $v$ visits an accepting state at some time $t \in \mathbb{N}$.

Emptiness problem:

Does automaton $A$ accept on some node in some digraph?
Emptiness problem

Automaton $A$ accepts digraph $G$ on node $v \in G$ iff $v$ visits an accepting state at some time $t \in \mathbb{N}$.

Emptiness problem:

Does automaton $A$ accept on some node in some digraph?

Simple reduction:
Emptiness problem

Automaton $A$ accepts digraph $G$ on node $v \in G$ iff $v$ visits an accepting state at some time $t \in \mathbb{N}$.

Emptiness problem:

Does automaton $A$ accept on some node in some digraph?

Simple reduction:

undecidable on dipaths

![Diagram](image-url)
Emptiness problem

Automaton $A$ accepts digraph $G$ on node $v \in G$ iff $v$ visits an accepting state at some time $t \in \mathbb{N}$.

Emptiness problem:

*Does automaton $A$ accept on some node in some digraph?*

Simple reduction:

undecidable on dipaths $\implies$ undecidable on digraphs
Simulating a Turing machine

Would be easy on doubly linked dipaths:

- Turing machine with alphabet \{ , \} and state set \{ 0, 1, 2, 3 \}
- Distributed automaton with state set \{ , \} × \{ ϵ, 0, 1, 2, 3 \}

But not on simple dipaths:
Simulating a Turing machine

Would be easy on doubly linked dipaths:
Simulating a Turing machine

Would be easy on doubly linked dipaths:

**Turing machine**
with alphabet \( \{ \text{□}, \text{□} \} \)
and state set \( \{ 0, 1, 2, 3 \} \)
Simulating a Turing machine

Would be easy on doubly linked dipaths:

space →

Turing machine

with alphabet \{◻, □\}

and state set \{0, 1, 2, 3\}
Simulating a Turing machine

Would be easy on doubly linked dipaths:

Turing machine
with alphabet \{0, 1\}
and state set \{0, 1, 2, 3\}
Simulating a Turing machine

Would be easy on doubly linked dipaths:

Turing machine
with alphabet \{□, □\}
and state set \{0, 1, 2, 3\}

Distributed automaton
with state set
\{□, □\} × \{ε, 0, 1, 2, 3\}
Simulating a Turing machine

Would be easy on doubly linked dipaths:

Turing machine
with alphabet \{\text{□}, \text{□}\}
and state set \{0, 1, 2, 3\}

Distributed automaton
with state set
\{\text{□}, \text{□}\} \times \{\epsilon, 0, 1, 2, 3\}

But not on simple dipaths:
Simulating a Turing machine

Would be easy on doubly linked dipaths:

Turing machine
with alphabet \{□, □\}
and state set \{0, 1, 2, 3\}

Distributed automaton
with state set
\{□, □\} × \{ε, 0, 1, 2, 3\}

But not on simple dipaths:
Simulating a Turing machine

Would be easy on doubly linked dipaths:

Turing machine
with alphabet \{\bigcirc, \square\}
and state set \{0, 1, 2, 3\}

Distributed automaton
with state set
\{\bigcirc, \square\} \times \{\epsilon, 0, 1, 2, 3\}

But not on simple dipaths:
Simulating a Turing machine

Would be easy on doubly linked dipaths:

Turing machine
with alphabet \{□, □\}
and state set \{0, 1, 2, 3\}

Distributed automaton
with state set
\{□, □\} \times \{ε, 0, 1, 2, 3\}

But not on simple dipaths:

One-way communication 😞
Exchanging space and time

Delay of 2 time steps between neighbors.

Turing machine

Distributed automaton

alphabet: {0, 1, 2, 3}

state set: {0, 1, 2, 3}
Delay of 2 time steps between neighbors.

Turing machine

alphabet : \{□, □\}
state set : \{0, 1, 2, 3\}
Exchanging space and time

Turing machine

space →

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}
Exchanging space and time

Turing machine

space →

0

0

0

0

time

alphabet : \{□, □\}

state set : \{0, 1, 2, 3\}
Exchanging space and time

**Turing machine**

- **space** →
- **time**

<table>
<thead>
<tr>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

alphabet: \{□, ■\}

state set: \{0, 1, 2, 3\}

Delay of 2 time steps between neighbors.
Exchanging space and time

**Turing machine**

- **space**

```
0 1 2 3 ...
```

- **time**

```
0 1 2 3 ...
```

- **alphabet**: \{□, □\}
- **state set**: \{0, 1, 2, 3\}

Delay of 2 time steps between neighbors.
Exchanging space and time

**Turing machine**

space $\rightarrow$

time

- 0
- 1
- 2
- 1

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

Delay of 2 time steps between neighbors.
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- **alphabet**: \{□, □\}
- **state set**: \{0, 1, 2, 3\}

Delay of 2 time steps between neighbors.
Exchanging space and time

Turing machine

space  →

time

0

1

2

1

0

3

···

···

···

···

···

···

···

···

alphabet: \{\ bluesquare, \ whitesquare\}

state set: \{0, 1, 2, 3\}
Exchanging space and time

Turing machine

Distributed automaton

space →

time

2

 Delay of 2 time steps

between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Distributed automaton

alphabet: \{\square, \blacksquare\}

state set: \{0, 1, 2, 3\}
Exchanging space and time

Turing machine

Distributed automaton

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}
Exchanging space and time

Turing machine
delay of 2 time steps between neighbors.

Distributed automaton

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\[ \bullet \] : waiting node
Exchanging space and time

Turing machine

![Turing machine diagram]

Distributed automaton

- alphabet: \{□, □\}
- state set: \{0, 1, 2, 3\}
- \(\bullet\) : waiting node
- \(\bigcirc\) : nodes “visiting” a Turing cell

Delay of 2 time steps between neighbors.
Exchanging space and time

Turing machine

- space →

- time

- alphabet: \{□, □\}
- state set: \{0, 1, 2, 3\}

Distributed automaton

- time →

- space

- waiting node

- nodes “visiting” a Turing cell
Exchanging space and time

**Turing machine**

- Space: 0, 1, 2, 3, ...
- Time: 0, 1, 2, 3, ...

**Distributed automaton**

- Time: 0, 1, 2, 3, ...
- Space: 0, 1, 2, 3, ...

**Alphabet:** \{□, □\}

**State set:** \{0, 1, 2, 3\}

- □: waiting node
- ○: nodes “visiting” a Turing cell

Delay of 2 time steps between neighbors.
Exchanging space and time

**Turing machine**

<table>
<thead>
<tr>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

**Distributed automaton**

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **alphabet**: \{□, □\}
- **state set**: \{0, 1, 2, 3\}
- **Waiting node**: ●
- **Nodes “visiting” a Turing cell**: ○
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

• : waiting node
○ : nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

Distributed automaton

alphabet: \{□, □\}
state set: \{0, 1, 2, 3\}

black: waiting node
blue 1 yellow: nodes “visiting” a Turing cell

Delay of 2 time steps between neighbors.
Exchanging space and time

Turing machine

space \[\rightarrow\]

time

0
1
2
1
0
3
···
···
···
···
···
···

Distributed automaton

time \[\rightarrow\]

space

0
1
2
1
0
3

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\(\bullet\) : waiting node

\(\bigcirc\) : nodes “visiting” a Turing cell

Delay of 2 time steps between neighbors.
Exchanging space and time

Turing machine

space →

<table>
<thead>
<tr>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

time ↓

Distributed automaton

time →

<table>
<thead>
<tr>
<th>0</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>...</th>
</tr>
</thead>
</table>

space ↓

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

• : waiting node

○: nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

- **space →**
  - 
  - 
  - 
  - 
  - 

- **time ↓**
  - 0
  - 1
  - 2
  - 1
  - 0
  - 3

Distributed automaton

- **time →**
  - 0
  - 1
  - 2
  - 1
  - 0

- **space ↓**
  - 0
  - 1
  - 2
  - 1
  - 0

**alphabet:** \{□, □\}

**state set:** \{0, 1, 2, 3\}

- **black dot:** waiting node
- **blue dot:** nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

Distributed automaton

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\(\blacktriangleright\) waiting node

\(\bluehexagon\) nodes “visiting” a Turing cell

Delay of 2 time steps between neighbors.
Exchanging space and time

Turing machine

Distributed automaton

alphabet: \{\square, \square\}

state set: \{0, 1, 2, 3\}

\(\text{Delay of 2 time steps between neighbors.}\)

\(\text{Turing machine:}\)

\(\text{Distributed automaton:}\)

\(\text{waiting node}\)

\(\text{nodes “visiting” a Turing cell}\)
Exchanging space and time

Turing machine

- Space →
- Time:
  - 0
  - 1
  - 2
  - 3

Distributed automaton

- Time →
- Space:
  - 0
  - 1
  - 2
  - 3

alphabet: \{□, □\}
state set: \{0, 1, 2, 3\}

• Waiting node
• Nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{\square, \blacksquare\}

state set: \{0, 1, 2, 3\}

\( \blacksquare \) : waiting node

\( \textcircled{1} \textcircled{0} \) : nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

- Space →
- Time:
  - 0
  - 1
  - 2
  - 3
  - ... 

Distributed automaton

- Time →
- Space:
  - 0
  - 1
  - 2
  - 3
  - ... 

Delay of 2 time steps between neighbors.

Alphabet: {□, □}
State set: {0, 1, 2, 3}

- □: waiting node
- □: nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{\text{□}, \text{□}\}

state set: \{0, 1, 2, 3\}

\(\bullet\) : waiting node

\(\circ\) : nodes “visiting” a Turing cell
Exchanging space and time

**Turing machine**

- Space →
- Time →

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Distributed automaton**

- Time →
- Space →

- Delay of 2 time steps between neighbors.

- Alphabet: \{□, □\}
- State set: \{0, 1, 2, 3\}

- Black circle: waiting node
- Blue circle: nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

\[
\begin{align*}
\text{time} & \quad \rightarrow \\
0 & \quad 1 & \quad 2 & \quad 3 & \quad \cdots \\
1 & \quad 2 & \quad 1 & \quad 0 & \quad \cdots \\
2 & \quad 1 & \quad 0 & \quad 1 & \quad \cdots \\
3 & \quad 0 & \quad 1 & \quad 2 & \quad \cdots \\
& \quad & \quad & \quad & \quad \\
\end{align*}
\]

Distributed automaton

\[
\begin{align*}
\text{time} & \quad \rightarrow \\
0 & \quad 1 & \quad 2 & \quad 3 & \quad \cdots \\
1 & \quad 2 & \quad 1 & \quad 0 & \quad \cdots \\
2 & \quad 1 & \quad 0 & \quad 1 & \quad \cdots \\
3 & \quad 0 & \quad 1 & \quad 2 & \quad \cdots \\
& \quad & \quad & \quad & \quad \\
\end{align*}
\]

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\[\begin{align*}
\text{black dot} & : \text{waiting node} \\
\text{blue dot} & : \text{nodes “visiting” a Turing cell}
\end{align*}\]
Exchanging space and time

Turing machine

- Space →
- Time ↓

Distributed automaton

- Time →
- Space ↓

Delay of 2 time steps between neighbors.

alphabet: \( \{ \square, \square \} \)

state set: \( \{ 0, 1, 2, 3 \} \)

- \( \bullet \): waiting node
- \( \circ \): nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

Delay of 2 time steps between neighbors.

Distributed automaton

alphabet: \{ \square, \blacksquare \}

state set: \{0, 1, 2, 3\}

\( \blacklozenge 1 \) : waiting node

\( \bigcirc 1 \) : nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

Delay of 2 time steps between neighbors.

Distributed automaton

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

•: waiting node

○○○○: nodes “visiting” a Turing cell

4 / 5
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>space</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{\[\], [\] \}

state set: \{0, 1, 2, 3\}

- \[\] : waiting node
- [\] : nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\(\bullet\): waiting node

\(\circ\): nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

- \(\bullet\): waiting node
- \(\bigcirc\): nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

time

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Delay of 2 time steps between neighbors.

Distributed automaton

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

time

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

space

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

- □: waiting node
- ○: nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

Delay of 2 time steps between neighbors.

Distributed automaton

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

- **□**: waiting node
- **○**: nodes “visiting” a Turing cell

4 / 5
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>space →</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time ←</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time →</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>space ←</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

• : waiting node

○: nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

space →

time

0

1

2

3

Delay of 2 time steps between neighbors.

Distributed automaton

time →

space

0

1

2

3

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\[\text{black} : \text{waiting node}\]

\[\text{blue} \rightarrow \text{blue} : \text{nodes “visiting” a Turing cell}\]
Exchanging space and time

Turing machine

space →

Distributed automaton

time →

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

: waiting node

: nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

\[
\begin{array}{c}
\text{space} \\
\hline
\text{time} \\
0 & 1 & 2 & 1 & 0 & \cdots \\
\hline
1 & 2 & 1 & 0 & 3 & \cdots \\
2 & 1 & 0 & 3 & \cdots \\
3 & \cdots & \cdots & \cdots & \cdots & \cdots \\
\end{array}
\]

Distributed automaton

\[
\begin{array}{c}
\text{time} \\
\hline
\text{space} \\
0 & 1 & 2 & 1 & 0 & \cdots \\
\hline
1 & 2 & 1 & 0 & 3 & \cdots \\
2 & 1 & 0 & 3 & \cdots \\
3 & \cdots & \cdots & \cdots & \cdots & \cdots \\
\end{array}
\]

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

\(\blacklozenge\): waiting node

\(\bigcirc\ 1\ \bigcirc\): nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>space</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>time</th>
<th>space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{■, □\}

state set: \{0, 1, 2, 3\}

\(\bullet\): waiting node

\(\bigcirc\): nodes “visiting” a Turing cell
Exchanging space and time

Turing machine

<table>
<thead>
<tr>
<th>Space</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Distributed automaton

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Delay of 2 time steps between neighbors.

alphabet: \{□, □\}

state set: \{0, 1, 2, 3\}

- \(\text{□}\) : waiting node
- \(\text{○}1\) : nodes “visiting” a Turing cell
Results

Emptiness problem undecidable:

- In general.
- For quasicyclic automata.

Emptiness problem decidable in logspace:

- For forgetful automata.

On words: MSO logic = forgetful automata

On trees: ——— ⊊ ———

On digraphs: ——— ⊈ ⊉ ———

State diagrams acyclic except for self-loops.

Nodes cannot remember their own state.
Results

Emptiness problem undecidable:
Results

Emptiness problem undecidable:
  ▶ In general.
Results

Emptiness problem undecidable:
- In general.
- For quasi-acyclic automata.
Results

Emptiness problem undecidable:

- In general.
- For quasi-acyclic automata.

State diagrams acyclic except for self-loops.
Results

Emptiness problem undecidable:
- In general.
- For quasi-acyclic automata.

State diagrams acyclic except for self-loops.

Emptiness problem decidable in LOGSPACE:
Results

Emptiness problem undecidable:
- In general.
- For quasi-acyclic automata.

State diagrams acyclic except for self-loops.

Emptiness problem decidable in LOGSPACE:
- For forgetful automata.
Results

Emptiness problem **undecidable**:

- In general.
- For *quasi-acyclic* automata.

  State diagrams acyclic except for self-loops.

Emptiness problem **decidable in LOGSPACE**:

- For *forgetful* automata.

  Nodes cannot remember their own state.
Results

Emptiness problem **undecidable**:  
- In general.
- For *quasi-acyclic* automata.  
  State diagrams acyclic except for self-loops.

Emptiness problem **decidable in LOGSPACE**:  
- For *forgetful* automata.  
  Nodes cannot remember their own state.

---

On words:  \( \text{MSO logic} = \text{forgetful automata} \)
Results

Emptiness problem undecidable:

- In general.
- For quasi-acyclic automata.

State diagrams acyclic except for self-loops.

Emptiness problem decidable in \textsc{logspace}:

- For forgetful automata.

Nodes cannot remember their own state.

<table>
<thead>
<tr>
<th>On words:</th>
<th>MSO logic</th>
<th>=</th>
<th>forgetful automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>On trees:</td>
<td></td>
<td>∅</td>
<td></td>
</tr>
</tbody>
</table>
Results

Emptiness problem undecidable:
- In general.
- For quasi-acyclic automata.

State diagrams acyclic except for self-loops.

Emptiness problem decidable in LOGSPACE:
- For forgetful automata.

Nodes cannot remember their own state.

<table>
<thead>
<tr>
<th>On words:</th>
<th>MSO logic</th>
<th>= forgetful automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>On trees:</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>On digraphs:</td>
<td>$\not\subset$</td>
<td>$\not\supset$</td>
</tr>
</tbody>
</table>
Thank you!