Learning Deterministic One-Counter Automata

OL: Polynomial-time active-learning algorithm for DOCA*

Sreejith A V

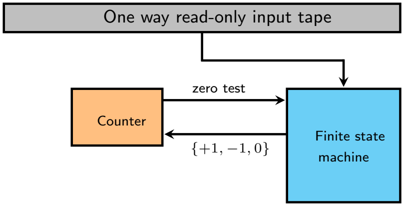
IIT Goa

IARCS, 20th May 2025

Prince Mathew

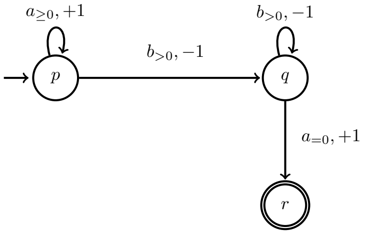
Vincent Penelle

One counter automata



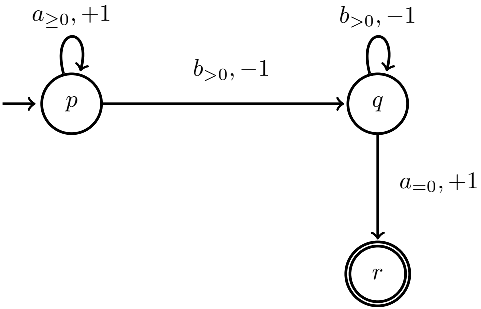
Counter: Can be incremented, decremented or tested for zero.

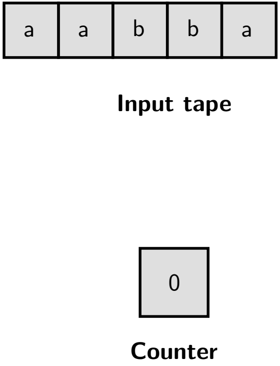
DOCA: Deterministic One Counter Automata.



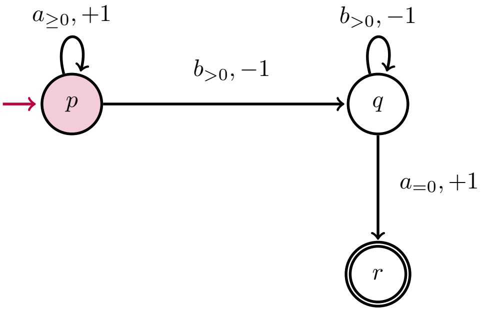
OCA accepting $\{a^n b^n a \mid n \geq 0\}$.

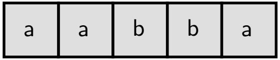
Example: $a^n b^n a$



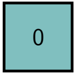


Example: $a^n b^n a$



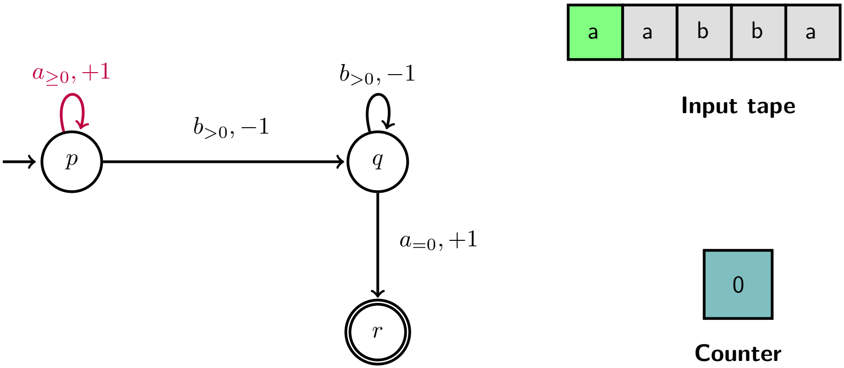


Input tape

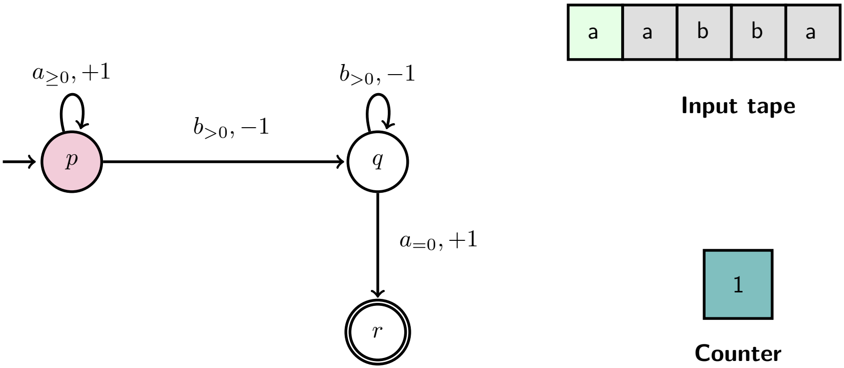


Counter

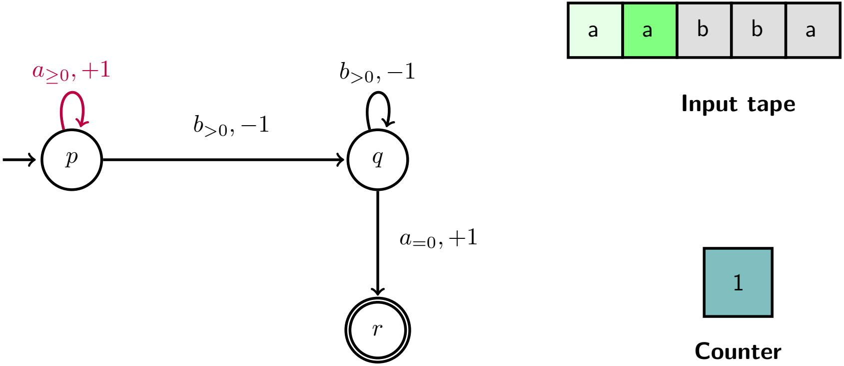
Example: $a^n b^n a$



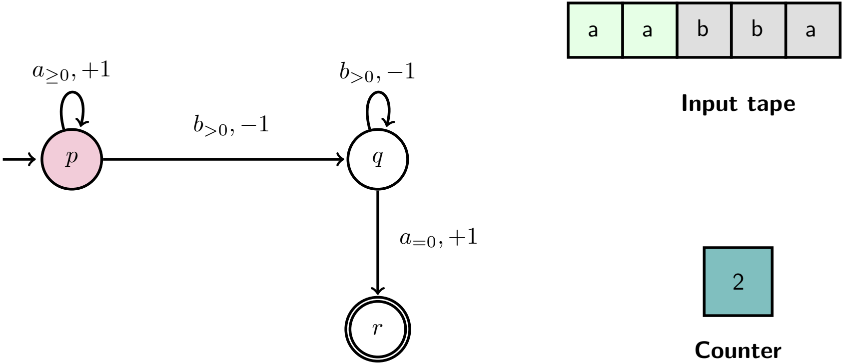
Example: $a^n b^n a$



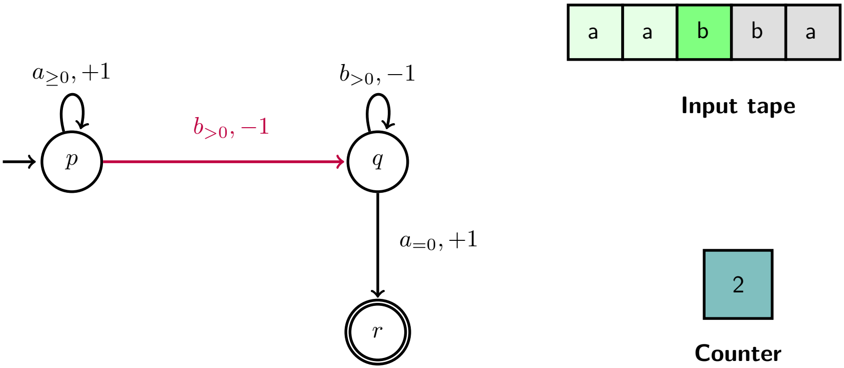
Example: $a^n b^n a$



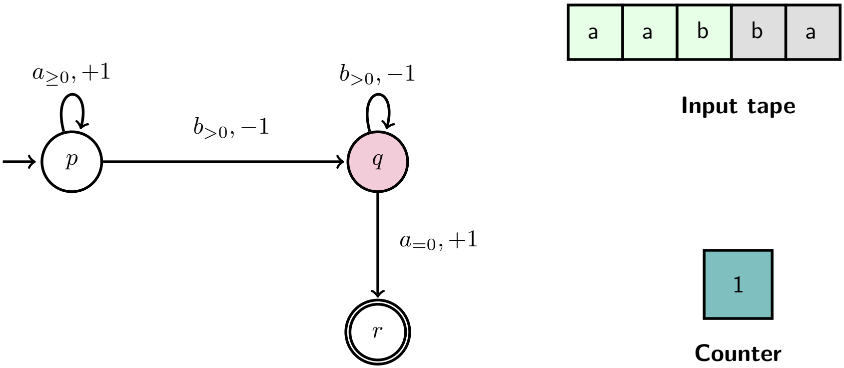
Example: $a^n b^n a$



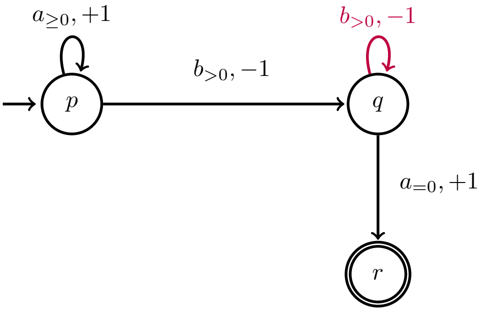
Example: $a^n b^n a$

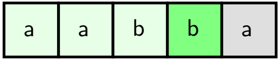


Example: $a^n b^n a$

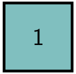


Example: $a^n b^n a$



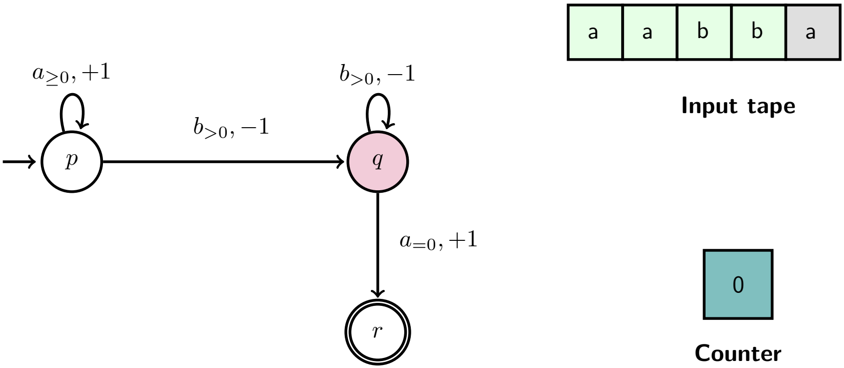


Input tape

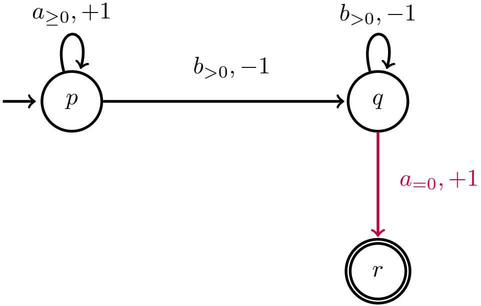


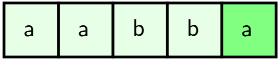
Counter

Example: $a^n b^n a$

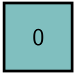


Example: $a^n b^n a$



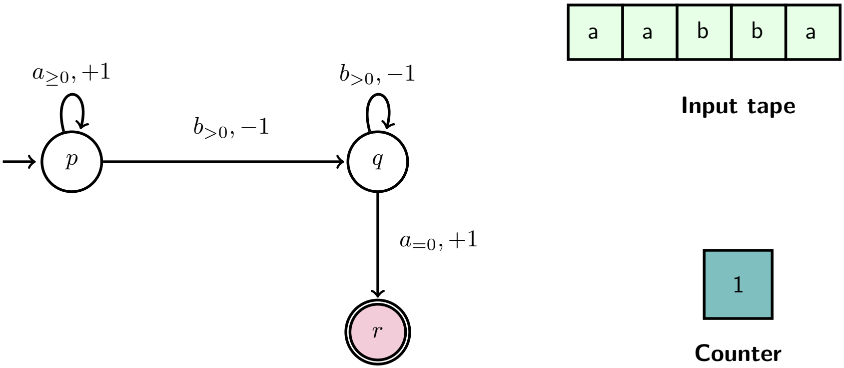


Input tape



Counter

Example: $a^n b^n a$



Motivation

Finite Automata $\subsetneq$ One-Counter Automata (OCA) $\subsetneq$ Pushdown Automata

- Modelling systems
  - Finite automata used extensively - eg. hardware verification.
  - Pushdown automata can model highly complex systems - eg. Softwares.
- Algorithmic complexity
  - Finite automata: Fast, mostly linear.
  - Pushdown automata: Hard, non-elementary to undecidable.
  - One-counter automata: Shows promise, some problems are theoretically good.
- Major challenges in OCA:
  - Equivalence - polynomial but $O(n^{20})$.
  - Active Learning - exponential.

Motivation

Finite Automata $\subsetneq$ One-Counter Automata (OCA) $\subsetneq$ Pushdown Automata

- Modelling systems
  - Finite automata used extensively - eg. hardware verification.
  - Pushdown automata can model highly complex systems - eg. Softwares.
- Algorithmic complexity
  - Finite automata: Fast, mostly linear.
  - Pushdown automata: Hard, non-elementary to undecidable.
  - One-counter automata: Shows promise, some problems are theoretically good.
- Major challenges in OCA:
  - Equivalence - polynomial but $O(n^{20})$.
  - Active Learning - exponential.

Motivation

Finite Automata $\subsetneq$ One-Counter Automata (OCA) $\subsetneq$ Pushdown Automata

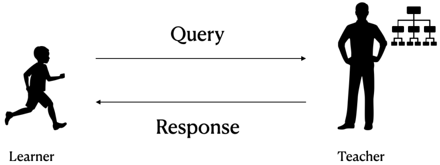
- Modelling systems
  - Finite automata used extensively - eg. hardware verification.
  - Pushdown automata can model highly complex systems - eg. Softwares.
- Algorithmic complexity
  - Finite automata: Fast, mostly linear.
  - Pushdown automata: Hard, non-elementary to undecidable.
  - One-counter automata: Shows promise, some problems are theoretically good.
- Major challenges in OCA:
  - Equivalence - polynomial but $O(n^{20})$.
  - Active Learning - exponential.

Motivation

Finite Automata $\subsetneq$ One-Counter Automata (OCA) $\subsetneq$ Pushdown Automata

- Modelling systems
  - Finite automata used extensively - eg. hardware verification.
  - Pushdown automata can model highly complex systems - eg. Softwares.
- Algorithmic complexity
  - Finite automata: Fast, mostly linear.
  - Pushdown automata: Hard, non-elementary to undecidable.
  - One-counter automata: Shows promise, some problems are theoretically good.
- Major challenges in OCA:
  - Equivalence - polynomial but $O(n^{20})$.
  - Active Learning - exponential.

Active Learning Framework



- There are two parties: Learner and Teacher.
- The teacher knows the language of a doca $\mathcal{T}$.
- The learner wants to learn a doca $\mathcal{L}$ such that $\mathcal{T}$ and $\mathcal{L}$ accept the same language.
- The learner can ask the teacher questions about the language of $\mathcal{T}$.
- The teacher answers the questions.
- The learner use the answers to learn the doca $\mathcal{L}$.

Queries

Membership query

Learner: Is w in the language of $\mathcal{T}$?

Teacher: Yes or No.

Equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a counter example w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Minimal-equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a minimal word w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Counter value query

Learner: What is the value of the counter in $\mathcal{T}$ after reading w?

Teacher: Counter value reached on w.

Queries

Membership query

Learner: Is w in the language of $\mathcal{T}$?

Teacher: Yes or No.

Equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a counter example w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Minimal-equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a minimal word w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Counter value query

Learner: What is the value of the counter in $\mathcal{T}$ after reading w?

Teacher: Counter value reached on w.

Queries

Membership query

Learner: Is w in the language of $\mathcal{T}$?

Teacher: Yes or No.

Equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a counter example w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Minimal-equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a minimal word w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Counter value query

Learner: What is the value of the counter in $\mathcal{T}$ after reading w?

Teacher: Counter value reached on w.

Queries

Membership query

Learner: Is w in the language of $\mathcal{T}$?

Teacher: Yes or No.

Equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a counter example w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Minimal-equivalence query

Learner: Is a doca $\mathcal{L}$ equivalent to $\mathcal{T}$?

Teacher: Yes or "No and a minimal word w that distinguishes $\mathcal{L}$ and $\mathcal{T}$".

Counter value query

Learner: What is the value of the counter in $\mathcal{T}$ after reading w?

Teacher: Counter value reached on w.

OL* - Active learning of doca[1]

Theorem (OL* in P)

- *Let teacher know a doca language.*
- *Let $\mathcal{T}$ be a minimal doca that accepts the language.*
- *Let $n = |\mathcal{T}|$ be the number of states in $\mathcal{T}$.*
- *The OL* algorithm learns a doca $\mathcal{L}$ that is equivalent to $\mathcal{T}$ in time polynomial in n, using membership and minimal-equivalence queries.*

[1] P. Mathew, V. Penelle, S. Learning deterministic one-counter automata in polynomial time, LICS 2025.

Literature review: Active learning of doca

- Angluin, 1987 → DFA – Membership, Equivalence
  P
- Fahmy & Roos, 1995 → DOCA^1 – Membership, Min. Equivalence
  $\mathrm{EXPTIME}$
- Neider & Löding, 2010 → VOCA^2 – Membership, Min. Equivalence
  $\mathrm{EXPTIME}$
- Bruyère et. al, 2022 → DOCA^1 – Membership, Min. Equivalence, Counter Value
  $\mathrm{EXPTIME}$
- Mathew et. al, 2025 → DOCA^1 – Membership, Min. Equivalence, Counter Value
  $\mathrm{EXPTIME}$, Polynomial queries

[1] realtime doca: strict subclass of doca,

[2] voca: visibly oca

The Configuration graph of a DOCA

Configuration graph of a doca

- Configuration: A pair (p, i) where p is a state and i is a counter value.
- Configuration graph:
  - States: all configurations (p, i).
  - Transitions: $(p, i) \xrightarrow{a} (q, j)$ if there is a transition from p to q on letter a and the counter value changes from i to j.
  - Final states: (p, i) where p is a final state.
  - Initial state: $(s, 0)$ where s is the start state.
- The configuration graph is infinite, if the oca is not a finite automata.

Configuration graph of a doca

- Configuration: A pair (p, i) where p is a state and i is a counter value.
- Configuration graph:
  - States: all configurations (p, i).
  - Transitions: $(p, i) \xrightarrow{a} (q, j)$ if there is a transition from p to q on letter a and the counter value changes from i to j.
  - Final states: (p, i) where p is a final state.
  - Initial state: $(s, 0)$ where s is the start state.
- The configuration graph is infinite, if the oca is not a finite automata.

Configuration graph of a doca

- Configuration: A pair (p, i) where p is a state and i is a counter value.
- Configuration graph:
  - States: all configurations (p, i).
  - Transitions: $(p, i) \xrightarrow{a} (q, j)$ if there is a transition from p to q on letter a and the counter value changes from i to j.
  - Final states: (p, i) where p is a final state.
  - Initial state: $(s, 0)$ where s is the start state.
- The configuration graph is infinite, if the oca is not a finite automata.

Configuration graph of a doca

- Configuration: A pair (p, i) where p is a state and i is a counter value.
- Configuration graph:
  - States: all configurations (p, i).
  - Transitions: $(p, i) \xrightarrow{a} (q, j)$ if there is a transition from p to q on letter a and the counter value changes from i to j.
  - Final states: (p, i) where p is a final state.
  - Initial state: $(s, 0)$ where s is the start state.
- The configuration graph is infinite, if the oca is not a finite automata.

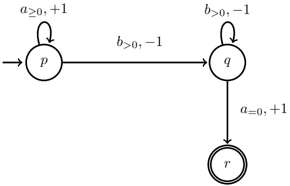
Configuration graph of a doca

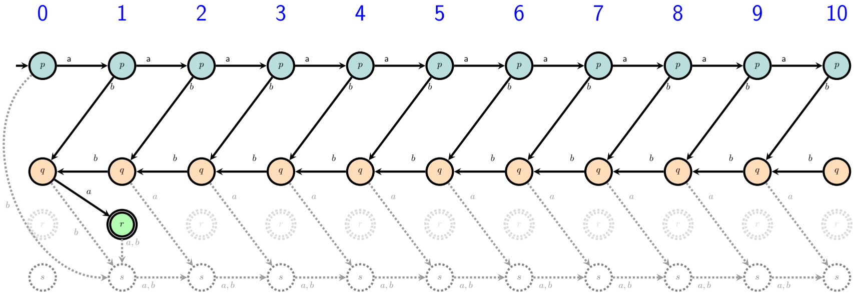
- Configuration: A pair (p, i) where p is a state and i is a counter value.
- Configuration graph:
  - States: all configurations (p, i).
  - Transitions: $(p, i) \xrightarrow{a} (q, j)$ if there is a transition from p to q on letter a and the counter value changes from i to j.
  - Final states: (p, i) where p is a final state.
  - Initial state: $(s, 0)$ where s is the start state.
- The configuration graph is infinite, if the oca is not a finite automata.

Configuration graph of a doca

- Configuration: A pair (p, i) where p is a state and i is a counter value.
- Configuration graph:
  - States: all configurations (p, i).
  - Transitions: $(p, i) \xrightarrow{a} (q, j)$ if there is a transition from p to q on letter a and the counter value changes from i to j.
  - Final states: (p, i) where p is a final state.
  - Initial state: $(s, 0)$ where s is the start state.
- The configuration graph is infinite, if the oca is not a finite automata.

Configuration graph - Example





Configuration sequences

- Consider a doca with n states.
- Let p be a state, and integers $d \leq n^2$, and $i > n^3$.
- Consider sequence of configurations

$$(p, i),\ (p, i+d),\ (p, i+2d) \quad \ldots$$

- On taking letter a from these configurations, we get sequence

$$(q, i+c),\ (q, i+d+c),\ (q, i+2d+c) \quad \ldots$$

for some state q and $c \in \{-1, 0, +1\}$.

- For any word w, where $|w| \leq n^3$, there is a state r and counter j

$$(p, i),\ (p, i+d), \quad \ldots \quad \xrightarrow{w} \quad (r, j),\ (r, j+d), \quad \ldots$$

- What is unique about each sequence? state, counter value (mod d) pair

Configuration sequences

- Consider a doca with n states.
- Let p be a state, and integers $d \leq n^2$, and $i > n^3$.
- Consider sequence of configurations

$$(p, i),\ (p, i+d),\ (p, i+2d) \quad \dots$$

- On taking letter a from these configurations, we get sequence

$$(q, i+c),\ (q, i+d+c),\ (q, i+2d+c) \quad \dots$$

  for some state q and $c \in \{-1, 0, +1\}$.
- For any word w, where $|w| \leq n^3$, there is a state r and counter j

$$(p, i),\ (p, i+d), \quad \dots \quad \xrightarrow{w} \quad (r, j),\ (r, j+d), \quad \dots$$

- What is unique about each sequence? state, counter value (mod d) pair

Configuration sequences

- Consider a doca with n states.
- Let p be a state, and integers $d \leq n^2$, and $i > n^3$.
- Consider sequence of configurations

$$(p, i),\ (p, i+d),\ (p, i+2d) \quad \ldots$$

- On taking letter a from these configurations, we get sequence

$$(q, i+c),\ (q, i+d+c),\ (q, i+2d+c) \quad \ldots$$

  for some state q and $c \in \{-1, 0, +1\}$.
- For any word w, where $|w| \leq n^3$, there is a state r and counter j

$$(p, i),\ (p, i+d), \quad \ldots \quad \xrightarrow{w} \quad (r, j),\ (r, j+d), \quad \ldots$$

- What is unique about each sequence? state, counter value (mod d) pair

Configuration sequences

- Consider a doca with n states.
- Let p be a state, and integers $d \leq n^2$, and $i > n^3$.
- Consider sequence of configurations

$$(p, i),\ (p, i+d),\ (p, i+2d) \quad \ldots$$

- On taking letter a from these configurations, we get sequence

$$(q, i+c),\ (q, i+d+c),\ (q, i+2d+c) \quad \ldots$$

  for some state q and $c \in \{-1, 0, +1\}$.
- For any word w, where $|w| \leq n^3$, there is a state r and counter j

$$(p, i),\ (p, i+d), \quad \ldots \quad \xrightarrow{w} \quad (r, j),\ (r, j+d), \quad \ldots$$

- What is unique about each sequence? state, counter value (mod d) pair

Configuration sequences contd.

- Number of “non-intersecting” sequences are

$$|\text{ number of states }| \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i + nd) \xrightarrow{v} (q, k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p, i), (p, i + d), \quad \dots \quad \xrightarrow{w} \quad (q, j), (q, j + d), \quad \dots, (q, k = j + td), \dots$$

  - *Proof*: Let $(p, i + nd) \xrightarrow{v} (q, k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i + nd) \xrightarrow{u} (r, l) \xrightarrow{y} (r, l + cd) \xrightarrow{v} (q, k).$$

  - Then,

$$(p, i + cd) \xrightarrow{u} (r, l + cd) \xrightarrow{v} (q, k).$$

Configuration sequences contd.

- Number of "non-intersecting" sequences are

$$|\text{ number of states }| \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i+nd) \xrightarrow{v} (q,k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p,i),(p,i+d), \quad \dots \quad \xrightarrow{w} \quad (q,j),(q,j+d), \quad \dots,(q,k=j+td),\dots$$

  - *Proof*: Let $(p, i+nd) \xrightarrow{v} (q,k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i+nd) \xrightarrow{u} (r,l) \xrightarrow{y} (r, l+cd) \xrightarrow{v} (q,k).$$

  - Then,

$$(p, i+cd) \xrightarrow{u} (r, l+cd) \xrightarrow{v} (q,k).$$

Configuration sequences contd.

- Number of "non-intersecting" sequences are

$$|\text{ number of states }| \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i + nd) \xrightarrow{v} (q, k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p, i), (p, i + d), \quad \ldots \quad \xrightarrow{w} \quad (q, j), (q, j + d), \quad \ldots, (q, k = j + td), \ldots$$

  - *Proof*: Let $(p, i + nd) \xrightarrow{v} (q, k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i + nd) \xrightarrow{u} (r, l) \xrightarrow{y} (r, l + cd) \xrightarrow{v} (q, k).$$

  - Then,

$$(p, i + cd) \xrightarrow{u} (r, l + cd) \xrightarrow{v} (q, k).$$

Configuration sequences contd.

- Number of "non-intersecting" sequences are

$$|\text{ number of states } | \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i + nd) \xrightarrow{v} (q, k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p, i), (p, i + d), \quad \dots \quad \xrightarrow{w} \quad (q, j), (q, j + d), \quad \dots, (q, k = j + td), \dots$$

  - *Proof*: Let $(p, i + nd) \xrightarrow{v} (q, k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i + nd) \xrightarrow{u} (r, l) \xrightarrow{y} (r, l + cd) \xrightarrow{v} (q, k).$$

  - Then,

$$(p, i + cd) \xrightarrow{u} (r, l + cd) \xrightarrow{v} (q, k).$$

Configuration sequences contd.

- Number of "non-intersecting" sequences are

$$|\text{ number of states }| \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i + nd) \xrightarrow{v} (q, k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p, i), (p, i + d), \quad \ldots \quad \xrightarrow{\quad w \quad} \quad (q, j), (q, j + d), \quad \ldots, (q, k = j + td), \ldots$$

  - *Proof*: Let $(p, i + nd) \xrightarrow{v} (q, k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i + nd) \xrightarrow{\quad u \quad} (r, l) \xrightarrow{\quad y \quad} (r, l + cd) \xrightarrow{\quad v \quad} (q, k).$$

  - Then,

$$(p, i + cd) \xrightarrow{\quad u \quad} (r, l + cd) \xrightarrow{\quad v \quad} (q, k).$$

Configuration sequences contd.

- Number of "non-intersecting" sequences are

$$|\text{ number of states }| \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i + nd) \xrightarrow{v} (q, k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p, i), (p, i + d), \quad \ldots \quad \xrightarrow{\quad w \quad} \quad (q, j), (q, j + d), \quad \ldots, (q, k = j + td), \ldots$$

  - *Proof*: Let $(p, i + nd) \xrightarrow{v} (q, k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i + nd) \xrightarrow{\quad u \quad} (r, l) \xrightarrow{\quad y \quad} (r, l + cd) \xrightarrow{\quad v \quad} (q, k).$$

  - Then,

$$(p, i + cd) \xrightarrow{\quad u \quad} (r, l + cd) \xrightarrow{\quad v \quad} (q, k).$$

Configuration sequences contd.

- Number of "non-intersecting" sequences are

$$|\text{ number of states }| \times d = nd \quad \leq n^3 \quad (\text{since } d \leq n^2)$$

- Let $(p, i + nd) \xrightarrow{v} (q, k)$ and the run do not touch a configuration with zero counter.
  - Then there is a w where $|w| \leq n^3$ such that

$$(p, i), (p, i + d), \quad \ldots \quad \xrightarrow{\quad w \quad} \quad (q, j), (q, j + d), \quad \ldots, (q, k = j + td), \ldots$$

  - *Proof*: Let $(p, i + nd) \xrightarrow{v} (q, k)$ where $|v| > n^3$.
  - Hence, there is a $c \geq -n$ such that

$$(p, i + nd) \xrightarrow{\quad u \quad} (r, l) \xrightarrow{\quad y \quad} (r, l + cd) \xrightarrow{\quad v \quad} (q, k).$$

  - Then,

$$(p, i + cd) \xrightarrow{\quad u \quad} (r, l + cd) \xrightarrow{\quad v \quad} (q, k).$$

Parallel Breadth First Search

- Consider the sequence

$$(p, i),\ (p, i + d),\ (p, i + 2d) \quad \ldots$$

- A parallel breadth first search (PBFS) will generate all sequences reachable without touching a zero configuration.
- The PBFS depth will be at most n^3.
- For a polynomially bounded sequence,

$$(p, i),\ (p, i + d),\ (p, i + 2d) \quad \ldots \quad (p, i + Kd)$$

PBFS will run in polynomial time.

Parallel Breadth First Search

- Consider the sequence

$$(p, i),\ (p, i+d),\ (p, i+2d) \quad \dots$$

- A parallel breadth first search (PBFS) will generate all sequences reachable without touching a zero configuration.
- The PBFS depth will be at most n^3.
- For a polynomially bounded sequence,

$$(p, i),\ (p, i+d),\ (p, i+2d) \quad \dots \quad (p, i+Kd)$$

PBFS will run in polynomial time.

Parallel Breadth First Search

- Consider the sequence

$$(p, i),\ (p, i + d),\ (p, i + 2d) \quad \ldots$$

- A parallel breadth first search (PBFS) will generate all sequences reachable without touching a zero configuration.
- The PBFS depth will be at most n^3.
- For a polynomially bounded sequence,

$$(p, i),\ (p, i + d),\ (p, i + 2d) \quad \ldots \quad (p, i + Kd)$$

PBFS will run in polynomial time.

The OL* Algorithm

Assumptions on DOCA

- The teacher knows a language accepted by a doca.
  - $\mathcal{T}$ is a minimal doca equivalent to teacher's doca language.
  - We denote by $n = |\mathcal{T}|$, the number of states.
- To make the presentation simpler, we assume the following about $\mathcal{T}$:
  - There are no ε transitions.
  - In a transition, the counter is incremented or decremented at most by one.
- Learner wants to learn a doca $\mathcal{L}$ equivalent to $\mathcal{T}$.

Assumptions on DOCA

- The teacher knows a language accepted by a doca.
  - $\mathcal{T}$ is a minimal doca equivalent to teacher's doca language.
  - We denote by $n = |\mathcal{T}|$, the number of states.
- To make the presentation simpler, we assume the following about $\mathcal{T}$:
  - There are no ε transitions.
  - In a transition, the counter is incremented or decremented at most by one.
- Learner wants to learn a doca $\mathcal{L}$ equivalent to $\mathcal{T}$.

Assumptions on DOCA

- The teacher knows a language accepted by a doca.
  - $\mathcal{T}$ is a minimal doca equivalent to teacher's doca language.
  - We denote by $n = |\mathcal{T}|$, the number of states.
- To make the presentation simpler, we assume the following about $\mathcal{T}$:
  - There are no ε transitions.
  - In a transition, the counter is incremented or decremented at most by one.
- Learner wants to learn a doca $\mathcal{L}$ equivalent to $\mathcal{T}$.

Assumptions on DOCA

- The teacher knows a language accepted by a doca.
  - $\mathcal{T}$ is a minimal doca equivalent to teacher's doca language.
  - We denote by $n = |\mathcal{T}|$, the number of states.
- To make the presentation simpler, we assume the following about $\mathcal{T}$:
  - There are no ε transitions.
  - In a transition, the counter is incremented or decremented at most by one.
- Learner wants to learn a doca $\mathcal{L}$ equivalent to $\mathcal{T}$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL* - Algorithm

- OL* first assumes $n = 1$.
- It learns a doca $\mathcal{L}$ that is checked for equivalence with teacher.
- If teacher says $\mathcal{L}$ is not equivalent to $\mathcal{T}$, then n is incremented.
- Process continues with incremented n.
- If teacher says $\mathcal{L}$ is equivalent to $\mathcal{T}$, then OL* terminates.

- For proof of correctness, it suffices to show the following
  - For every n, OL* runs in time polynomial in n.
  - OL* learns an equivalent doca, when $n = |\mathcal{T}|$.

OL*: Step 1. Learning Behaviour DFA

- Learner do not have access to the configuration graph of $\mathcal{T}$.
- $\mathcal{A}$ is a k-behaviour dfa if $\mathcal{A}$ is k-equivalent to $\mathcal{T}$. That is,

$$w \text{ is accepted by } \mathcal{A} \quad \text{iff} \quad w \text{ is accepted by } \mathcal{T}, \quad \text{for all } |w| \leq k.$$

- Angluin's L^* algorithm can learn a k-behaviour dfa in time polynomial in k and n.
  - This is where minimal-equivalence is used.
- Step 1. of learner is to learn a $poly(n)$-behaviour dfa.
- We will fix $poly(n)$ later.

OL*: Step 1. Learning Behaviour DFA

- Learner do not have access to the configuration graph of $\mathcal{T}$.
- $\mathcal{A}$ is a k-behaviour dfa if $\mathcal{A}$ is k-equivalent to $\mathcal{T}$. That is,

$$w \text{ is accepted by } \mathcal{A} \quad \text{iff} \quad w \text{ is accepted by } \mathcal{T}, \quad \text{for all } |w| \leq k.$$

- Angluin's L^* algorithm can learn a k-behaviour dfa in time polynomial in k and n.
  - This is where minimal-equivalence is used.
- Step 1. of learner is to learn a $poly(n)$-behaviour dfa.
- We will fix $poly(n)$ later.

OL*: Step 1. Learning Behaviour DFA

- Learner do not have access to the configuration graph of $\mathcal{T}$.
- $\mathcal{A}$ is a k-behaviour dfa if $\mathcal{A}$ is k-equivalent to $\mathcal{T}$. That is,

$$w \text{ is accepted by } \mathcal{A} \quad \text{iff} \quad w \text{ is accepted by } \mathcal{T}, \quad \text{for all } |w| \leq k.$$

- Angluin's L^* algorithm can learn a k-behaviour dfa in time polynomial in k and n.
  - This is where minimal-equivalence is used.
- Step 1. of learner is to learn a $poly(n)$-behaviour dfa.
- We will fix $poly(n)$ later.

OL*: Step 1. Learning Behaviour DFA

- Learner do not have access to the configuration graph of $\mathcal{T}$.
- $\mathcal{A}$ is a k-behaviour dfa if $\mathcal{A}$ is k-equivalent to $\mathcal{T}$. That is,

$$w \text{ is accepted by } \mathcal{A} \quad \text{iff} \quad w \text{ is accepted by } \mathcal{T}, \quad \text{for all } |w| \leq k.$$

- Angluin's L^* algorithm can learn a k-behaviour dfa in time polynomial in k and n.
  - This is where minimal-equivalence is used.
- Step 1. of learner is to learn a $poly(n)$-behaviour dfa.
- We will fix $poly(n)$ later.

OL*: Step 1. Learning Behaviour DFA

- Learner do not have access to the configuration graph of $\mathcal{T}$.
- $\mathcal{A}$ is a k-behaviour dfa if $\mathcal{A}$ is k-equivalent to $\mathcal{T}$. That is,

$$w \text{ is accepted by } \mathcal{A} \quad \text{iff} \quad w \text{ is accepted by } \mathcal{T}, \quad \text{for all } |w| \leq k.$$

- Angluin's L^* algorithm can learn a k-behaviour dfa in time polynomial in k and n.
  - This is where minimal-equivalence is used.
- Step 1. of learner is to learn a $poly(n)$-behaviour dfa.
- We will fix $poly(n)$ later.

OL*: Step 1. Learning Behaviour DFA

- Learner do not have access to the configuration graph of $\mathcal{T}$.
- $\mathcal{A}$ is a k-behaviour dfa if $\mathcal{A}$ is k-equivalent to $\mathcal{T}$. That is,

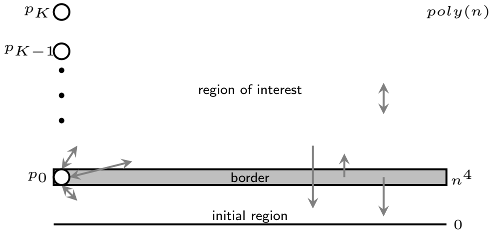
$$w \text{ is accepted by } \mathcal{A} \quad \text{iff} \quad w \text{ is accepted by } \mathcal{T}, \quad \text{for all } |w| \leq k.$$

- Angluin's L^* algorithm can learn a k-behaviour dfa in time polynomial in k and n.
  - This is where minimal-equivalence is used.
- Step 1. of learner is to learn a $poly(n)$-behaviour dfa.
- We will fix $poly(n)$ later.

OL*: Step 2. Partitioning the behaviour DFA

The dfa $\mathcal{A}$ is partitioned into:

- Initial region: States reachable by words of length $< n^4$.
- Border region: States reachable by words of length n^4 but not less.
- Region of interest: Remaining states.

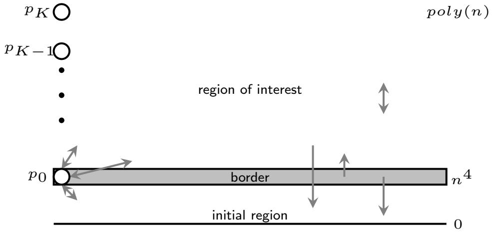


- A path from initial region to region of interest should traverse via some border state.
- Partial OCA construction
  - Pick a border state p_0.
  - DFA $\mathcal{A}_{p_0}$: Remove all states other than p_0 from border.
  - Learner constructs a partial OCA, $\mathcal{L}_{p_0}$ that is $poly(n)$-equivalent to $\mathcal{A}_{p_0}$.

OL*: Step 2. Partitioning the behaviour DFA

The dfa $\mathcal{A}$ is partitioned into:

- Initial region: States reachable by words of length $< n^4$.
- Border region: States reachable by words of length n^4 but not less.
- Region of interest: Remaining states.

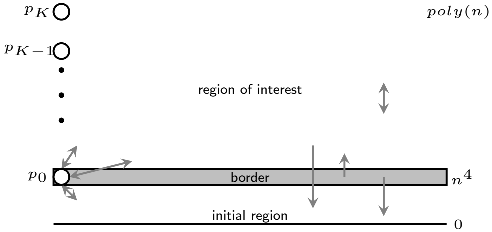


- A path from initial region to region of interest should traverse via some border state.
- Partial OCA construction
  - Pick a border state p_0.
  - DFA $\mathcal{A}_{p_0}$: Remove all states other than p_0 from border.
  - Learner constructs a partial OCA, $\mathcal{L}_{p_0}$ that is $poly(n)$-equivalent to $\mathcal{A}_{p_0}$.

OL*: Step 2. Partitioning the behaviour DFA

The dfa $\mathcal{A}$ is partitioned into:

- Initial region: States reachable by words of length $< n^4$.
- Border region: States reachable by words of length n^4 but not less.
- Region of interest: Remaining states.

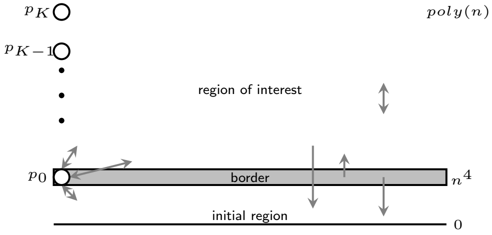


- A path from initial region to region of interest should traverse via some border state.
- Partial OCA construction
  - Pick a border state p_0.
  - DFA $\mathcal{A}_{p_0}$: Remove all states other than p_0 from border.
  - Learner constructs a partial OCA, $\mathcal{L}_{p_0}$ that is $poly(n)$-equivalent to $\mathcal{A}_{p_0}$.

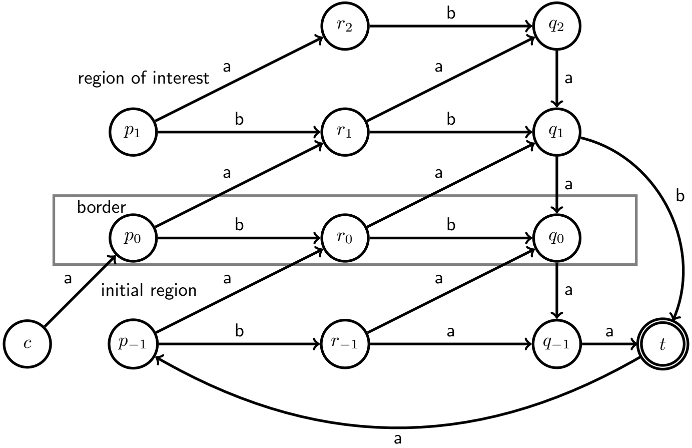
OL*: Step 2. Partitioning the behaviour DFA

The dfa $\mathcal{A}$ is partitioned into:

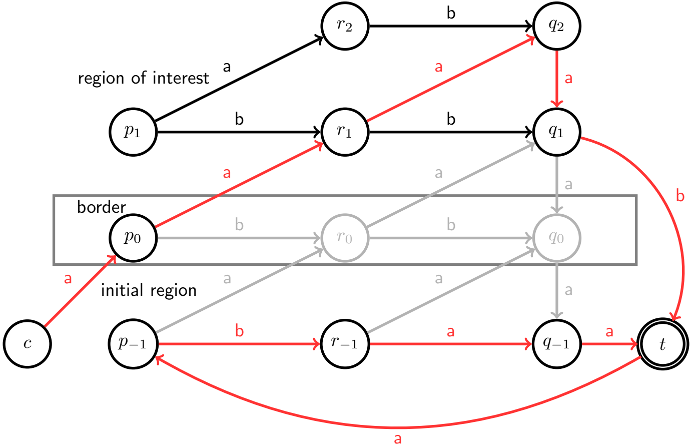
- Initial region: States reachable by words of length $< n^4$.
- Border region: States reachable by words of length n^4 but not less.
- Region of interest: Remaining states.



- A path from initial region to region of interest should traverse via some border state.
- Partial OCA construction
  - Pick a border state p_0.
  - DFA $\mathcal{A}_{p_0}$: Remove all states other than p_0 from border.
  - Learner constructs a partial OCA, $\mathcal{L}_{p_0}$ that is $poly(n)$-equivalent to $\mathcal{A}_{p_0}$.



r_2
b
q_2
a
region of interest
a
a
p_1
b
r_1
b
q_1
a
a
a
b
border
p_0
b
r_0
b
q_0
a
initial region
a
a
a
c
p_{-1}
b
r_{-1}
a
q_{-1}
a
t
a


region of interest
border
initial region
r_2
q_2
p_1
r_1
q_1
p_0
r_0
q_0
c
p_{-1}
r_{-1}
q_{-1}
t
a
b

OL*: Step 3. Finding a winning sequence

Definition

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

is a winning sequence if the run of these words on $\mathcal{T}$ reach configurations

$$(p, i),\ (p, i+d),\ (p, i+2d), \quad \ldots, \quad (p, i+Kd)$$

respectively, for some state p, and $d \leq n^2$ and $i > n^3$.

Lemma (Winning sequence lemma)

For any state p_0 in behaviour dfa $\mathcal{A}$, a winning sequence

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

can be found in polynomial time, such that the run of w_0 on $\mathcal{A}$ reaches state p_0.

OL*: Step 3. Finding a winning sequence

Definition

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

is a winning sequence if the run of these words on $\mathcal{T}$ reach configurations

$$(p, i),\ (p, i+d),\ (p, i+2d), \quad \ldots, \quad (p, i+Kd)$$

respectively, for some state p, and $d \leq n^2$ and $i > n^3$.

Lemma (Winning sequence lemma)

For any state p_0 in behaviour dfa $\mathcal{A}$, a winning sequence

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

can be found in polynomial time, such that the run of w_0 on $\mathcal{A}$ reaches state p_0.

OL*: Step 4. Parallel BFS on $\mathcal{A}$

- Consider a *winning sequence*

$$w_0, w_1, w_2, \quad \dots, \quad w_K$$

- Run these words on the behaviour dfa. We reach state sequence

$$p_0, p_1, p_2, \quad \dots, \quad p_K$$

- Run parallel BFS (depth at most n^3) from this sequence.
  - All distinct sequences identified.
  - At most n^3 distinct sequences.
  - These sequences are the states of doca $\mathcal{L}_{p_0}$.

OL*: Step 4. Parallel BFS on $\mathcal{A}$

- Consider a *winning sequence*

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

- Run these words on the behaviour dfa. We reach state sequence

$$p_0, p_1, p_2, \quad \ldots, \quad p_K$$

- Run parallel BFS (depth at most n^3) from this sequence.
  - All distinct sequences identified.
  - At most n^3 distinct sequences.
  - These sequences are the states of doca $\mathcal{L}_{p_0}$.

OL*: Step 4. Parallel BFS on $\mathcal{A}$

- Consider a *winning sequence*

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

- Run these words on the behaviour dfa. We reach state sequence

$$p_0, p_1, p_2, \quad \ldots, \quad p_K$$

- Run parallel BFS (depth at most n^3) from this sequence.
  - All distinct sequences identified.
  - At most n^3 distinct sequences.
  - These sequences are the states of doca $\mathcal{L}_{p_0}$.

OL*: Step 4. Parallel BFS on $\mathcal{A}$

- Consider a *winning sequence*

$$w_0, w_1, w_2, \quad \dots, \quad w_K$$

- Run these words on the behaviour dfa. We reach state sequence

$$p_0, p_1, p_2, \quad \dots, \quad p_K$$

- Run parallel BFS (depth at most n^3) from this sequence.
  - All distinct sequences identified.
  - At most n^3 distinct sequences.
  - These sequences are the states of doca $\mathcal{L}_{p_0}$.

OL*: Step 4. Parallel BFS on $\mathcal{A}$

- Consider a *winning sequence*

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

- Run these words on the behaviour dfa. We reach state sequence

$$p_0, p_1, p_2, \quad \ldots, \quad p_K$$

- Run parallel BFS (depth at most n^3) from this sequence.
  - All distinct sequences identified.
  - At most n^3 distinct sequences.
  - These sequences are the states of doca $\mathcal{L}_{p_0}$.

OL*: Step 4. Parallel BFS on $\mathcal{A}$

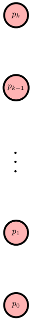
- Consider a *winning sequence*

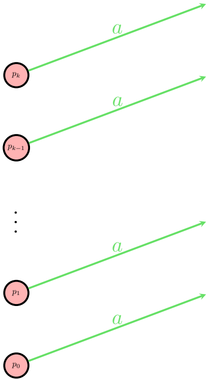
$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

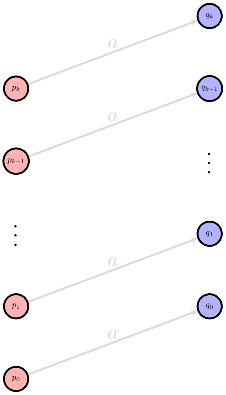
- Run these words on the behaviour dfa. We reach state sequence

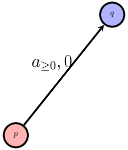
$$p_0, p_1, p_2, \quad \ldots, \quad p_K$$

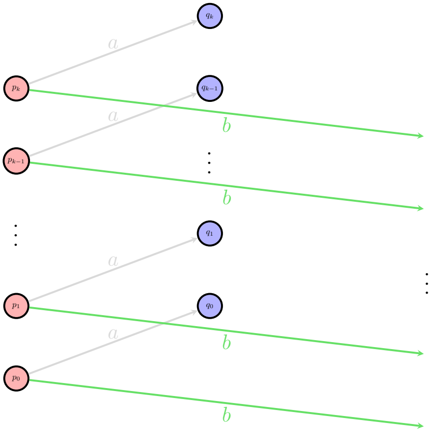
- Run parallel BFS (depth at most n^3) from this sequence.
  - All distinct sequences identified.
  - At most n^3 distinct sequences.
  - These sequences are the states of doca $\mathcal{L}_{p_0}$.

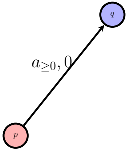

p_k
p_{k-1}
p_1
p_0

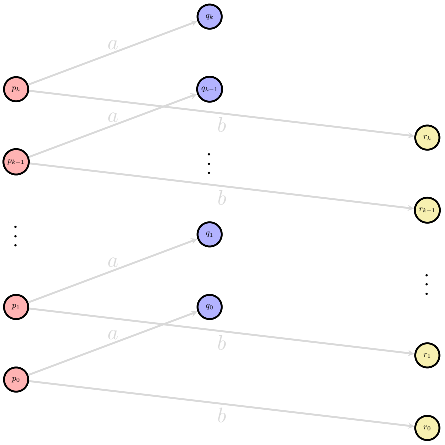

p_k
a
p_{k-1}
a
p_1
a
p_0
a

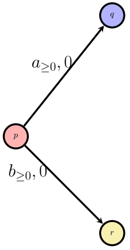

q_k
a
p_k
q_{k-1}
a
p_{k-1}
q_1
a
p_1
q_0
a
p_0

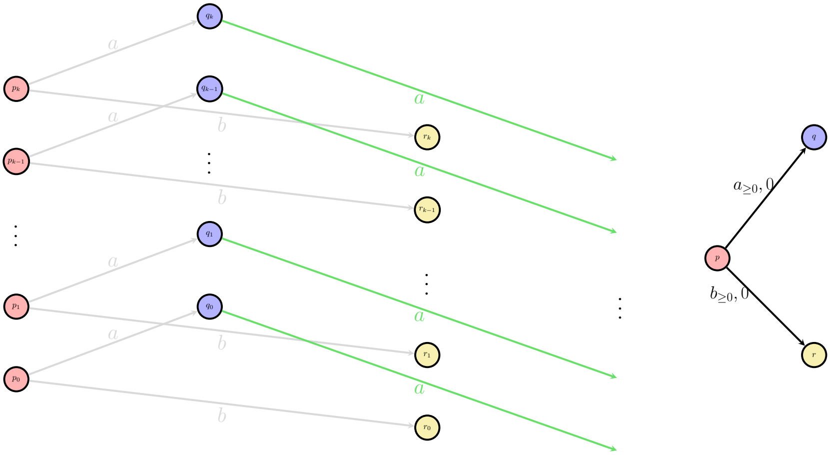

q
$a_{\geq 0}, 0$
p

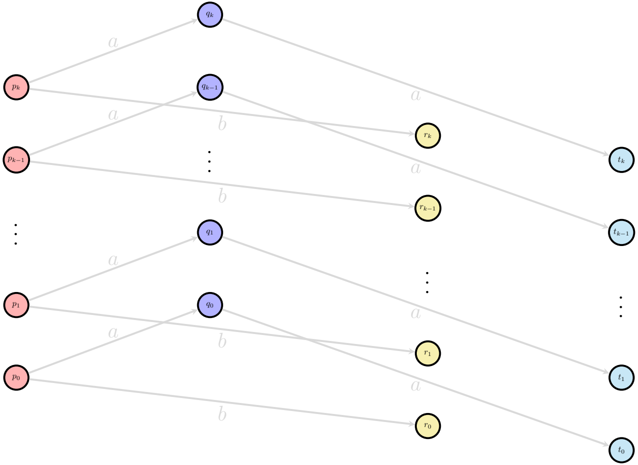

q_k
a
p_k
q_{k-1}
a
b
p_{k-1}
b
q_1
a
q_0
p_1
a
b
p_0
b

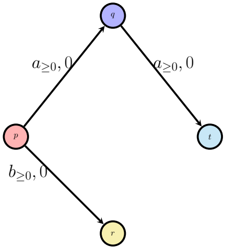

q
$a_{\geq 0}, 0$
p

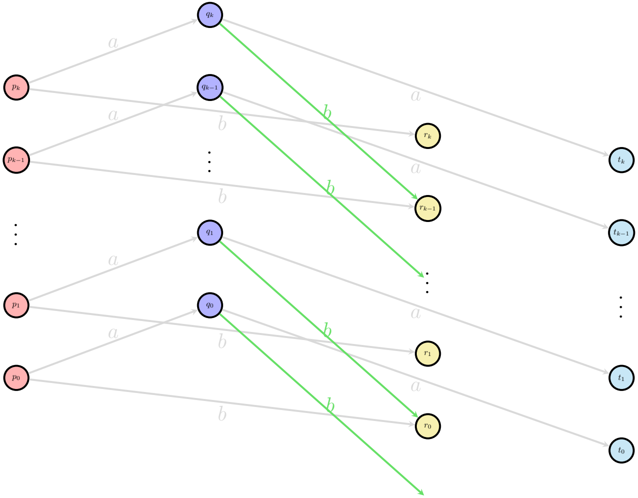

q_k
q_{k-1}
q_1
q_0
p_k
p_{k-1}
p_1
p_0
r_k
r_{k-1}
r_1
r_0
a
b

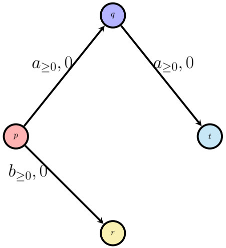

p
q
r
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$

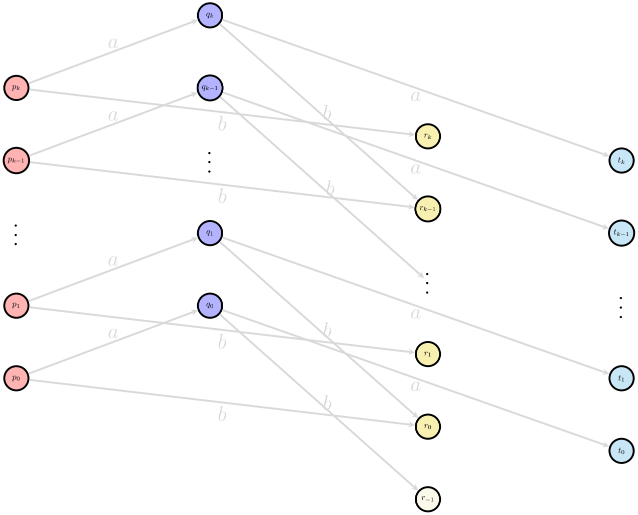

q_k
q_{k-1}
q_1
q_0
p_k
p_{k-1}
p_1
p_0
r_k
r_{k-1}
r_1
r_0
a
b
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
p
q
r

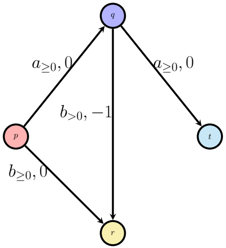

p_k
p_{k-1}
p_1
p_0
q_k
q_{k-1}
q_1
q_0
r_k
r_{k-1}
r_1
r_0
t_k
t_{k-1}
t_1
t_0
a
b

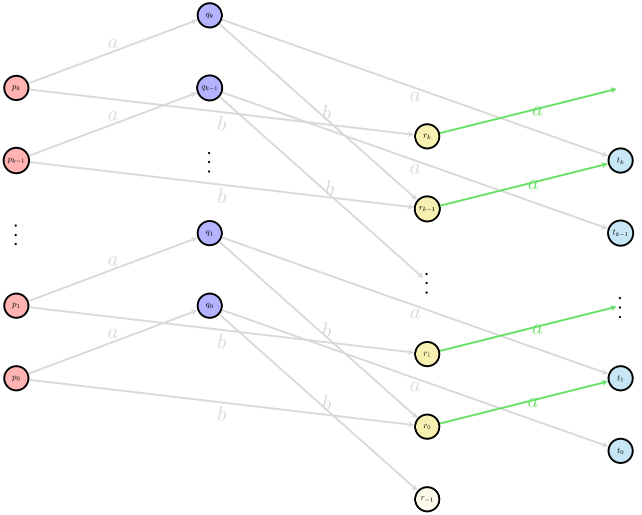

p
q
r
t
$a_{\geq 0}, 0$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$

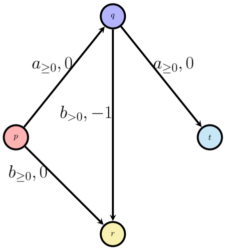

q_k
p_k
q_{k-1}
p_{k-1}
r_k
t_k
r_{k-1}
t_{k-1}
q_1
p_1
q_0
p_0
r_1
t_1
r_0
t_0
a
b

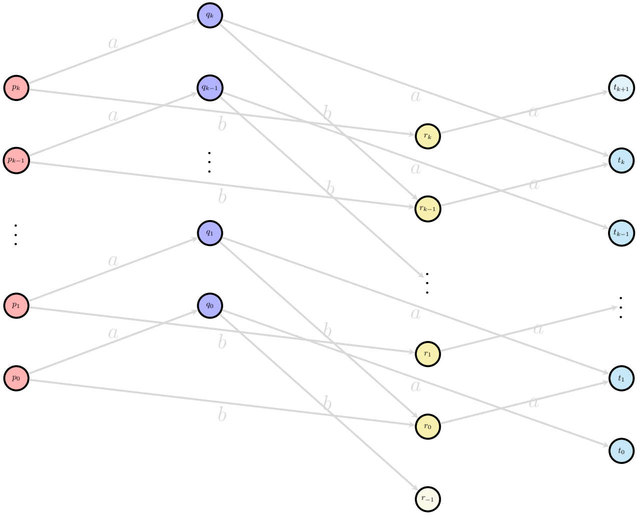

q
p
t
r
$a_{\geq 0}, 0$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$

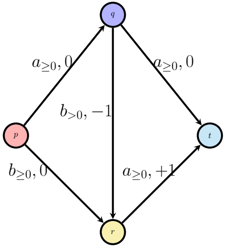

p_k
p_{k-1}
p_1
p_0
q_k
q_{k-1}
q_1
q_0
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_k
t_{k-1}
t_1
t_0
a
b

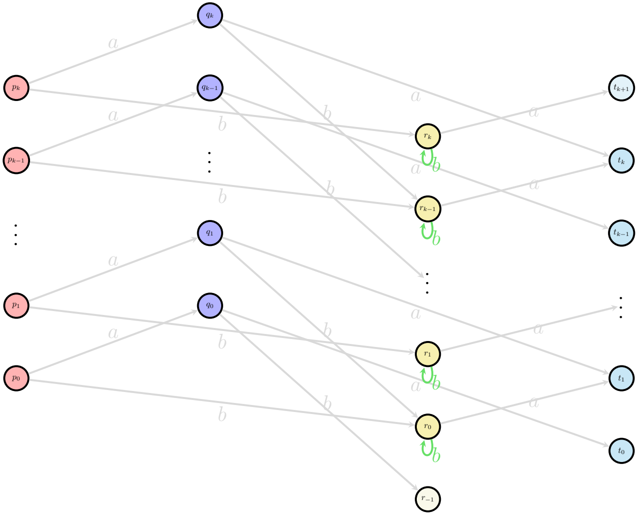

p
q
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$

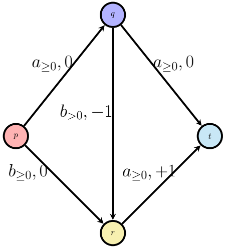

q_k
p_k
q_{k-1}
p_{k-1}
r_k
t_k
r_{k-1}
t_{k-1}
q_1
p_1
q_0
p_0
r_1
t_1
r_0
t_0
r_{-1}
a
b

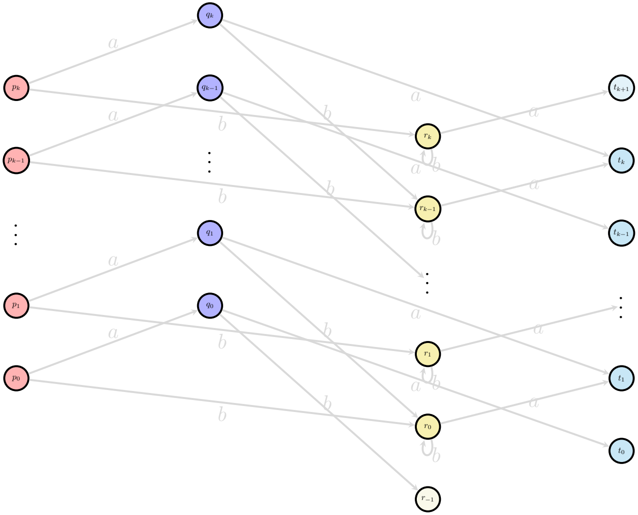

q
p
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$b_{\geq 0}, 0$

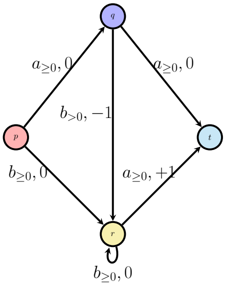

p_k
p_{k-1}
p_1
p_0
q_k
q_{k-1}
q_1
q_0
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b

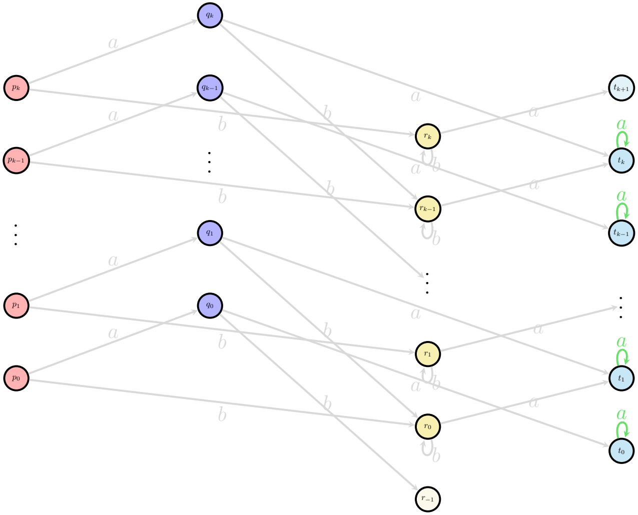

p
q
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$

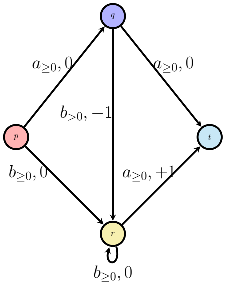

q_k
q_{k-1}
q_1
q_0
p_k
p_{k-1}
p_1
p_0
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0

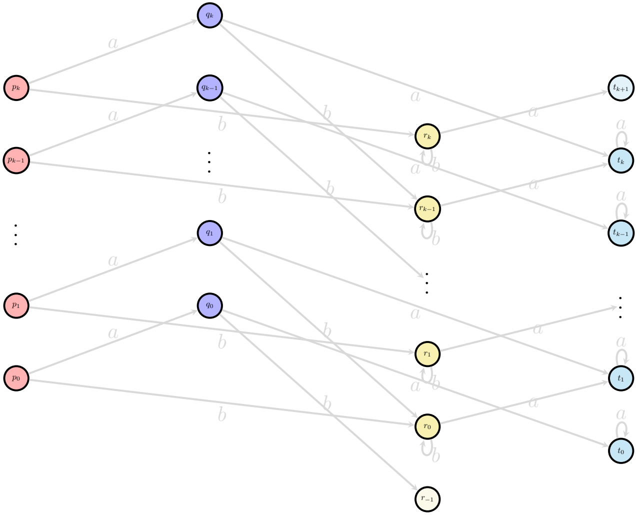

q
p
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$

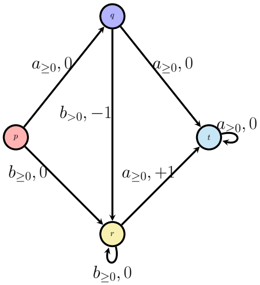

p_k
p_{k-1}
p_1
p_0
q_k
q_{k-1}
q_1
q_0
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b

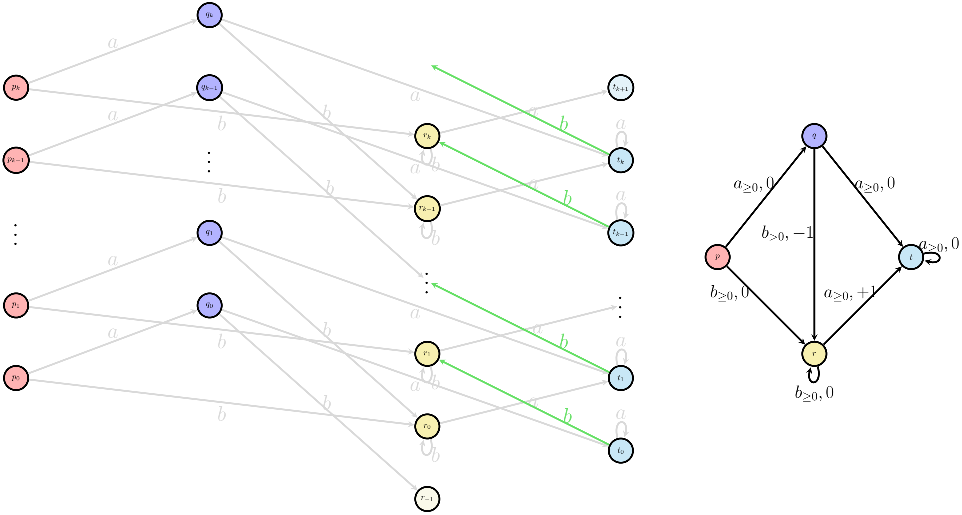

p
q
r
t
$a_{\geq 0}, 0$
$a_{\geq 0}, 0$
$b_{>0}, -1$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$
$b_{\geq 0}, 0$

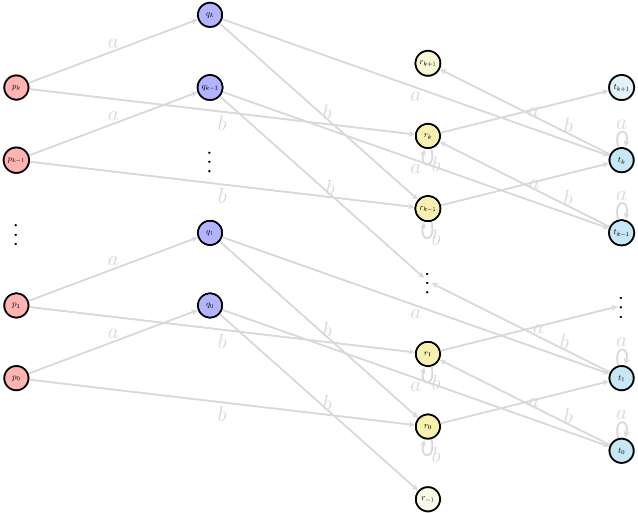

p_k
p_{k-1}
p_1
p_0
q_k
q_{k-1}
q_1
q_0
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b

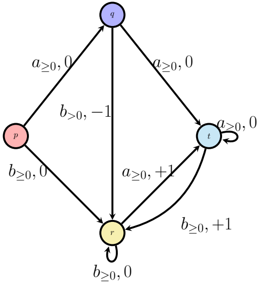

p
q
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$
$b_{\geq 0}, 0$


p_k
p_{k-1}
p_1
p_0
q_k
q_{k-1}
q_1
q_0
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b


p
q
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$
$b_{\geq 0}, 0$


q_k
p_k
q_{k-1}
t_{k+1}
r_k
p_{k-1}
t_k
r_{k-1}
t_{k-1}
q_1
p_1
q_0
r_1
t_1
p_0
r_0
t_0
r_{-1}
a
b
q
p
r
t
$a_{\geq 0}, 0$
$b_{>0}, -1$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$


q_k
q_{k-1}
q_1
q_0
p_k
p_{k-1}
p_1
p_0
r_{k+1}
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b


p
q
r
t
$a_{\geq 0}, 0$
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$
$b_{\geq 0}, +1$
$b_{\geq 0}, 0$

OL*: Step 5. Constructing $\mathcal{L}_{p_0}$

- The states of the partial OCA are the sequences.
  - Let (p_0, p_K) be one sequence - call this Red sequence.
  - Let (q_0, q_K) be another sequence - call this $Blue$ sequence.
  - Then Red and $Blue$ are states of partial OCA.
- Transitions:
  - If $(p_0, p_K) \xrightarrow{a} (q_0, q_K)$, add transition $Red \xrightarrow{a \geq 0, 0} Blue$.
  - If $(p_0, p_{K-1}) \xrightarrow{a} (q_1, q_K)$, add transition $Red \xrightarrow{a \geq 0, +1} Blue$.[1]
  - If $(p_1, p_K) \xrightarrow{a} (q_0, q_{K-1})$, add transition $Red \xrightarrow{a > 0, -1} Blue$.[1]
- Hence:
  - $p_i \xrightarrow{a} q_j$ if and only if $(Red, i) \xrightarrow{a} (Blue, j)$.
  - If $(Red, i) \xrightarrow{w} (Blue, j)$, then $p_i \xrightarrow{w} q_j$.
  - If $p_i \xrightarrow{a} r_k \xrightarrow{b} s_l \ldots \xrightarrow{a} q_j$, then $(Red, i) \xrightarrow{ab\ldots a} (Blue, j)$.

[1] $p_n \xrightarrow{a} q_\ell$ for $0 \leq \ell \leq 2n$ is possible.

OL*: Step 5. Constructing $\mathcal{L}_{p_0}$

- The states of the partial OCA are the sequences.
  - Let (p_0, p_K) be one sequence - call this Red sequence.
  - Let (q_0, q_K) be another sequence - call this $Blue$ sequence.
  - Then Red and $Blue$ are states of partial OCA.
- Transitions:
  - If $(p_0, p_K) \xrightarrow{a} (q_0, q_K)$, add transition $Red \xrightarrow{a_{\geq 0}, 0} Blue$.
  - If $(p_0, p_{K-1}) \xrightarrow{a} (q_1, q_K)$, add transition $Red \xrightarrow{a_{\geq 0}, +1} Blue$.[1]
  - If $(p_1, p_K) \xrightarrow{a} (q_0, q_{K-1})$, add transition $Red \xrightarrow{a_{>0}, -1} Blue$.[1]
- Hence:
  - $p_i \xrightarrow{a} q_j$ if and only if $(Red, i) \xrightarrow{a} (Blue, j)$.
  - If $(Red, i) \xrightarrow{w} (Blue, j)$, then $p_i \xrightarrow{w} q_j$.
  - If $p_i \xrightarrow{a} r_k \xrightarrow{b} s_l \ldots \xrightarrow{a} q_j$, then $(Red, i) \xrightarrow{ab \ldots a} (Blue, j)$.

[1] $p_n \xrightarrow{a} q_\ell$ for $0 \leq \ell \leq 2n$ is possible.

OL*: Step 5. Constructing $\mathcal{L}_{p_0}$

- The states of the partial OCA are the sequences.
  - Let (p_0, p_K) be one sequence - call this Red sequence.
  - Let (q_0, q_K) be another sequence - call this $Blue$ sequence.
  - Then Red and $Blue$ are states of partial OCA.
- Transitions:
  - If $(p_0, p_K) \xrightarrow{a} (q_0, q_K)$, add transition $Red \xrightarrow{a_{\geq 0}, 0} Blue$.
  - If $(p_0, p_{K-1}) \xrightarrow{a} (q_1, q_K)$, add transition $Red \xrightarrow{a_{\geq 0}, +1} Blue$.[1]
  - If $(p_1, p_K) \xrightarrow{a} (q_0, q_{K-1})$, add transition $Red \xrightarrow{a_{> 0}, -1} Blue$.[1]
- Hence:
  - $p_i \xrightarrow{a} q_j$ if and only if $(Red, i) \xrightarrow{a} (Blue, j)$.
  - If $(Red, i) \xrightarrow{w} (Blue, j)$, then $p_i \xrightarrow{w} q_j$.
  - If $p_i \xrightarrow{a} r_k \xrightarrow{b} s_l \ldots \xrightarrow{a} q_j$, then $(Red, i) \xrightarrow{ab\ldots a} (Blue, j)$.

[1] $p_n \xrightarrow{a} q_\ell$ for $0 \leq \ell \leq 2n$ is possible.

Patch work - Missing states

- The parallel BFS colors "most" of the reachable states from border state p.
- However, upto n^3 number of states are not colored (called Neg states)
  - eg. the state r_{-1} in the example, and some states reachable from r_{-1}.
- The Neg states are added to the partial OCA.
- Neg states are always with zero counter value.
- Transitions between Neg states do not increment or decrement counter.

Patch work - Missing states

- The parallel BFS colors "most" of the reachable states from border state p.
- However, upto n^3 number of states are not colored (called Neg states)
  - eg. the state r_{-1} in the example, and some states reachable from r_{-1}.
- The Neg states are added to the partial OCA.
- Neg states are always with zero counter value.
- Transitions between Neg states do not increment or decrement counter.

Patch work - Missing states

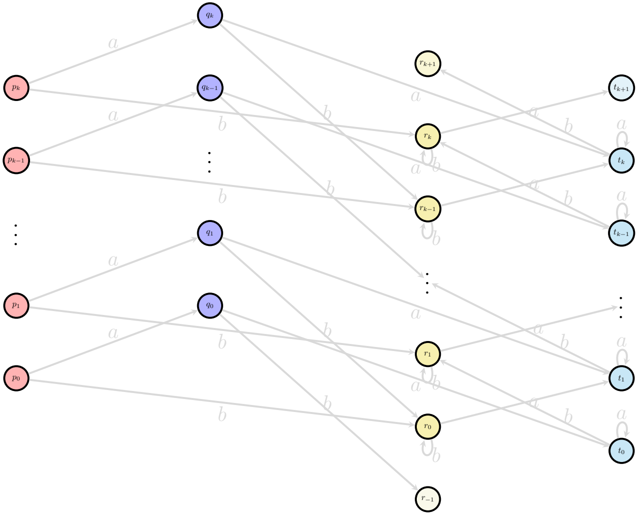
- The parallel BFS colors "most" of the reachable states from border state p.
- However, upto n^3 number of states are not colored (called Neg states)
  - eg. the state r_{-1} in the example, and some states reachable from r_{-1}.
- The Neg states are added to the partial OCA.
- Neg states are always with zero counter value.
- Transitions between Neg states do not increment or decrement counter.

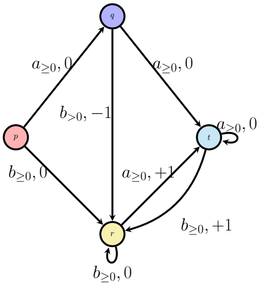
Patch work - Missing states

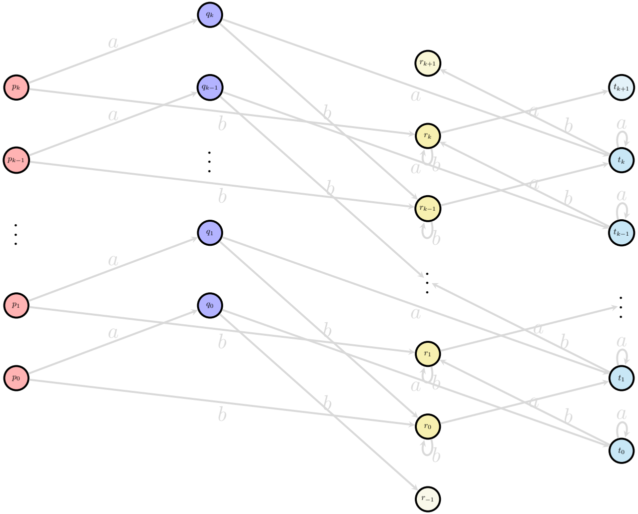
- The parallel BFS colors “most” of the reachable states from border state p.
- However, upto n^3 number of states are not colored (called Neg states)
  - eg. the state r_{-1} in the example, and some states reachable from r_{-1}.
- The Neg states are added to the partial OCA.
- Neg states are always with zero counter value.
- Transitions between Neg states do not increment or decrement counter.

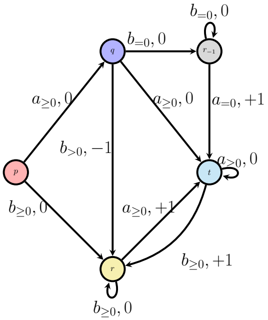
Patch work - Missing states

- The parallel BFS colors “most” of the reachable states from border state p.
- However, upto n^3 number of states are not colored (called Neg states)
  - eg. the state r_{-1} in the example, and some states reachable from r_{-1}.
- The Neg states are added to the partial OCA.
- Neg states are always with zero counter value.
- Transitions between Neg states do not increment or decrement counter.


q_k
q_{k-1}
q_1
q_0
p_k
p_{k-1}
p_1
p_0
r_{k+1}
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b

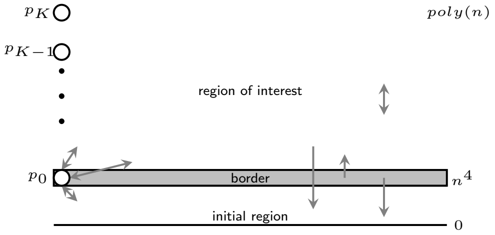

p
q
r
t
$a_{\geq 0}, 0$
$a_{\geq 0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$a_{\geq 0}, +1$
$b_{\geq 0}, +1$
$b_{\geq 0}, 0$


q_k
q_{k-1}
q_1
q_0
p_k
p_{k-1}
p_1
p_0
r_{k+1}
r_k
r_{k-1}
r_1
r_0
r_{-1}
t_{k+1}
t_k
t_{k-1}
t_1
t_0
a
b


p
q
r
t
r_{-1}
$a_{\geq 0}, 0$
$b_{\geq 0}, 0$
$b_{=0}, 0$
$b_{=0}, 0$
$b_{>0}, -1$
$a_{\geq 0}, 0$
$a_{=0}, +1$
$a_{\geq 0}, 0$
$a_{\geq 0}, +1$
$b_{\geq 0}, +1$
$b_{\geq 0}, 0$

Adding initial region

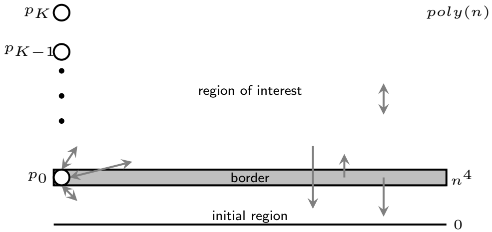
- The initial-region is added to the partial OCA.
- Like Neg states, initial region do not increment or decrement counter.



This concludes the construction of partial OCA $\mathcal{L}_{p_0}$.

Adding initial region

- The initial-region is added to the partial OCA.
- Like Neg states, initial region do not increment or decrement counter.



This concludes the construction of partial OCA $\mathcal{L}_{p_0}$.

Partial OCA $\mathcal{L}_{p_0}$ - Properties

Theorem

Let w be a word of length $\leq poly(n)$. Then one of the following holds:

- *Either*

$$\mathcal{L}_p \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

- *or there is a prefix u such that the run of u on $\mathcal{A}$ reaches a border state $q \neq p$.*

OL*: Step 6. Constructing $\mathcal{L}$

- The final OCA $\mathcal{L}$ is union of partial OCAs $\mathcal{L}_p, \mathcal{L}_q, \ldots, \mathcal{L}_r$ where $border = \{p, q, \ldots, r\}$
- From, construction of $\mathcal{L}_p$: for any word w where $|w| \leq poly(n)$:

$$\mathcal{L}_p \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

  or there is a prefix u such that the run of u on $\mathcal{A}$ reaches a border state $q \neq p$.
- Hence, for any word w where $|w| \leq poly(n)$:

$$\mathcal{L} \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

- [Böhm et. al., 2013] There is a polynomial $poly(n)$ such that if $\mathcal{L}$ is $poly(n)$-equivalent to $\mathcal{T}$, then $\mathcal{L}$ is equivalent to $\mathcal{T}$.
- Hence:

$$\mathcal{L} \quad \text{is equivalent to} \quad \mathcal{T}$$

OL*: Step 6. Constructing $\mathcal{L}$

- The final OCA $\mathcal{L}$ is union of partial OCAs $\mathcal{L}_p, \mathcal{L}_q, \ldots, \mathcal{L}_r$ where $border = \{p, q, \ldots, r\}$
- From, construction of $\mathcal{L}_p$: for any word w where $|w| \leq poly(n)$:

$$\mathcal{L}_p \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

or there is a prefix u such that the run of u on $\mathcal{A}$ reaches a border state $q \neq p$.

- Hence, for any word w where $|w| \leq poly(n)$:

$$\mathcal{L} \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

- [Böhm et. al., 2013] There is a polynomial $poly(n)$ such that if $\mathcal{L}$ is $poly(n)$-equivalent to $\mathcal{T}$, then $\mathcal{L}$ is equivalent to $\mathcal{T}$.
- Hence:

$$\mathcal{L} \quad \text{is equivalent to} \quad \mathcal{T}$$

OL*: Step 6. Constructing $\mathcal{L}$

- The final OCA $\mathcal{L}$ is union of partial OCAs $\mathcal{L}_p, \mathcal{L}_q, \ldots, \mathcal{L}_r$ where $border = \{p, q, \ldots, r\}$
- From, construction of $\mathcal{L}_p$: for any word w where $|w| \leq poly(n)$:

$$\mathcal{L}_p \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

  or there is a prefix u such that the run of u on $\mathcal{A}$ reaches a border state $q \neq p$.
- Hence, for any word w where $|w| \leq poly(n)$:

$$\mathcal{L} \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

- [Böhm et. al., 2013] There is a polynomial $poly(n)$ such that if $\mathcal{L}$ is $poly(n)$-equivalent to $\mathcal{T}$, then $\mathcal{L}$ is equivalent to $\mathcal{T}$.
- Hence:

$$\mathcal{L} \quad \text{is equivalent to} \quad \mathcal{T}$$

OL*: Step 6. Constructing $\mathcal{L}$

- The final OCA $\mathcal{L}$ is union of partial OCAs $\mathcal{L}_p, \mathcal{L}_q, \ldots, \mathcal{L}_r$ where $border = \{p, q, \ldots, r\}$
- From, construction of $\mathcal{L}_p$: for any word w where $|w| \leq poly(n)$:

$$\mathcal{L}_p \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

  or there is a prefix u such that the run of u on $\mathcal{A}$ reaches a border state $q \neq p$.
- Hence, for any word w where $|w| \leq poly(n)$:

$$\mathcal{L} \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

- [Böhm et. al., 2013] There is a polynomial $poly(n)$ such that if $\mathcal{L}$ is $poly(n)$-equivalent to $\mathcal{T}$, then $\mathcal{L}$ is equivalent to $\mathcal{T}$.
- Hence:

$$\mathcal{L} \quad \text{is equivalent to} \quad \mathcal{T}$$

OL*: Step 6. Constructing $\mathcal{L}$

- The final OCA $\mathcal{L}$ is union of partial OCAs $\mathcal{L}_p, \mathcal{L}_q, \ldots, \mathcal{L}_r$ where $border = \{p, q, \ldots, r\}$
- From, construction of $\mathcal{L}_p$: for any word w where $|w| \leq poly(n)$:

  $$\mathcal{L}_p \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

  or there is a prefix u such that the run of u on $\mathcal{A}$ reaches a border state $q \neq p$.
- Hence, for any word w where $|w| \leq poly(n)$:

  $$\mathcal{L} \text{ accepts } w \quad \text{iff} \quad \mathcal{A} \text{ accepts } w$$

- [Böhm et. al., 2013] There is a polynomial $poly(n)$ such that if $\mathcal{L}$ is $poly(n)$-equivalent to $\mathcal{T}$, then $\mathcal{L}$ is equivalent to $\mathcal{T}$.
- Hence:

  $$\mathcal{L} \quad \text{is equivalent to} \quad \mathcal{T}$$

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Summary of OL*

- Construct $poly(n)$-behaviour DFA using L* algorithm.
- Partition the behaviour DFA into *initial region*, *border*, and *region of interest*.
- For each border state:
  - Generate a winning sequence of words: $w_0, w_1, \ldots, w_K$.
  - Run these words on the DFA to get sequence of states: $p_0, p_1, \ldots, p_K$.
  - Run parallel BFS from this sequence.
  - All reachable sequences of parallel BFS form states of partial OCA.
  - Counter values are incremented / decremented based on sequence shift.
  - Add Neg and initial region to get partial OCA.
- Construct final OCA by combining partial OCAs.

Proof of winning sequence lemma

Winning sequence lemma

Definition

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

is a winning sequence if the run of these words on $\mathcal{T}$ reach configurations

$$(p, i),\ (p, i+d),\ (p, i+2d), \quad \ldots, \quad (p, i+Kd)$$

respectively, for some state p, and $d \leq n^2$ and $i > n^3$.

Lemma (Winning sequence lemma)

For any state p_0 in behaviour dfa $\mathcal{A}$, a winning sequence

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

can be found in polynomial time, such that the run of w_0 on $\mathcal{A}$ reaches state p_0.

Proof of winning sequence lemma

- Let $(s, 0)$ be the start configuration of a doca.
- For a configuration (p, i), we say

$$w = llex(p, i)$$

  if w is the lexicographically minimal word that takes $(s, 0)$ to (p, i).
- That is,

$$(s, 0) \xrightarrow{w} (p, i), \quad \text{and}$$

$$(s, 0) \xrightarrow{u} (p, i) \implies (|w|, w) \leq (|u|, u), \quad \text{for all } u.$$

Proof of winning sequence lemma contd.

Lemma

- *Let (p, i) be a configuration where $i > n^3$. Then,*

$$llex(p, i) = xy^r z, \quad \textit{where } |x|, |y|, |z| \leq n^3, \textit{ and } y \textit{ increases counter by } \leq n^2.$$

- *Furthermore,*

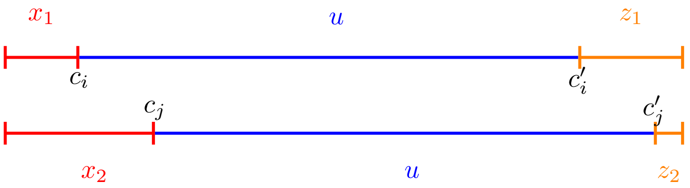
$$(s, 0) \xrightarrow{xy^{r+j}z} (p, i + jd), \quad \textit{for all } j \geq 0, \textit{ and } d \leq n^2.$$

Proof:

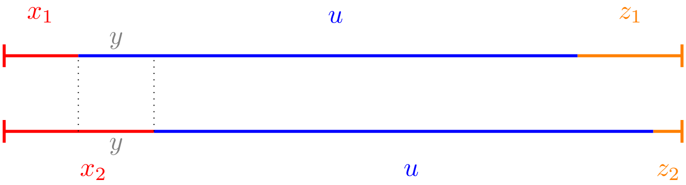
- Let $w = llex(p, i)$.
- Let c_i be the last configuration where counter value i is seen for the last time.

Repeating Factor

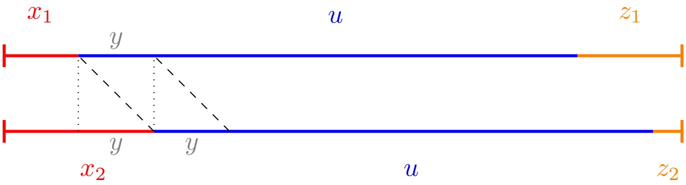
- c_i is the configuration where counter value i is seen for the last time.
- there are c_i and c_j with same state and c'_i and c'_j with same state.



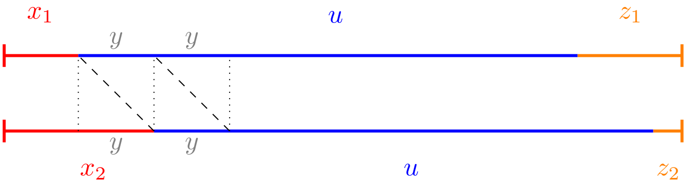
Repeating Factor



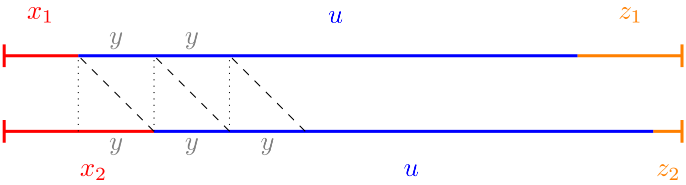
Repeating Factor



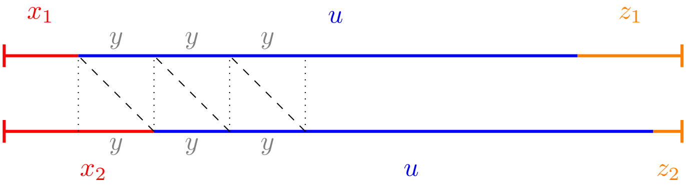
Repeating Factor



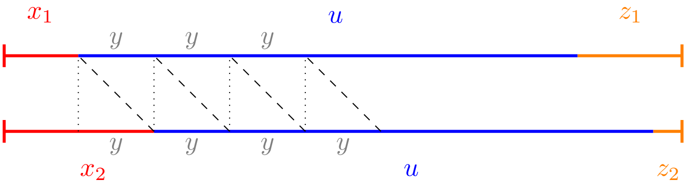
Repeating Factor



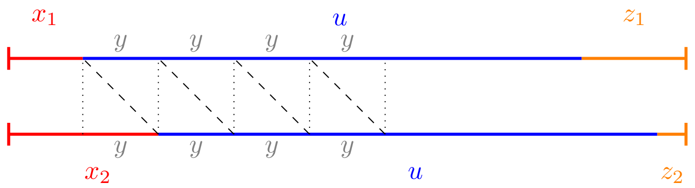
Repeating Factor



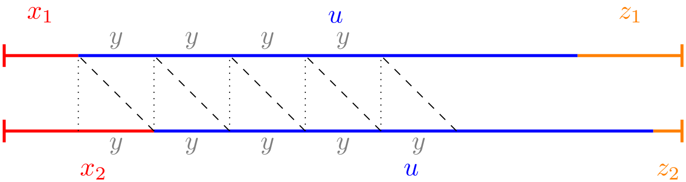
Repeating Factor



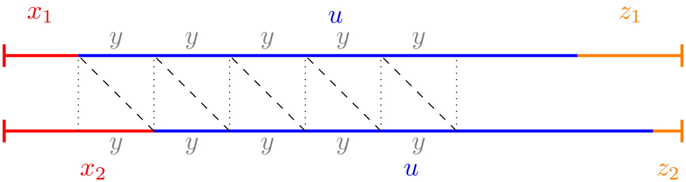
Repeating Factor



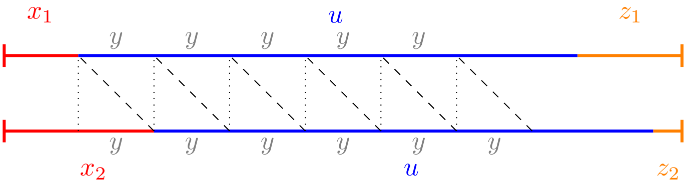
Repeating Factor



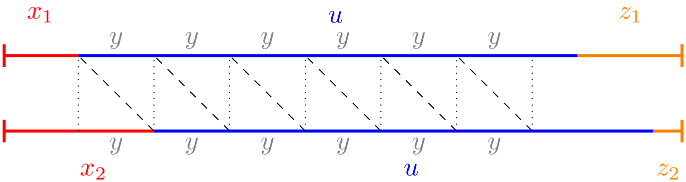
Repeating Factor



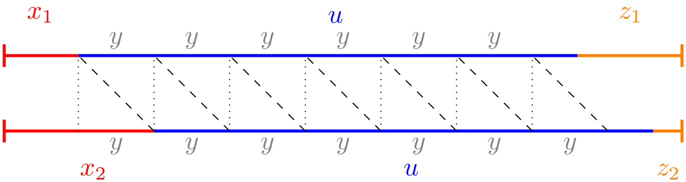
Repeating Factor



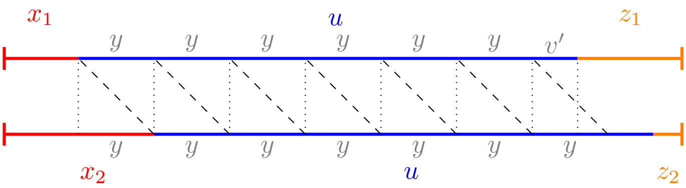
Repeating Factor



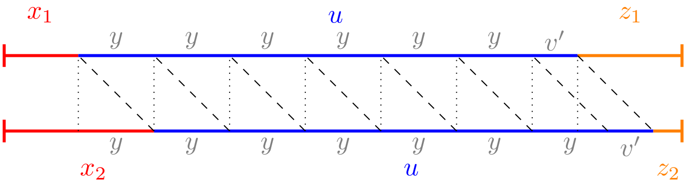
Repeating Factor



Repeating Factor



Repeating Factor



Proof of winning sequence lemma contd.

Lemma

- *Let (p, i) be a configuration where $i > n^3$. Then,*

$$llex(p,i) = xy^r z, \quad \text{where } |x|, |y|, |z| \leq n^3, \text{ and } y \text{ increases counter by } \leq n^2.$$

- *Furthermore,*

$$(s, 0) \xrightarrow{xy^{r+j}z} (p, i + jd), \quad \text{for all } j \geq 0, \text{ and } d \leq n^2.$$

Lemma (Winning sequence lemma)

For any state p_0 in behaviour dfa $\mathcal{A}$, a winning sequence

$$w_0, w_1, w_2, \quad \ldots, \quad w_K$$

can be found in polynomial time, such that the run of w_0 on $\mathcal{A}$ reaches state p_0.

Conclusion

Theorem

OL learns a doca equivalent to the teacher's doca using membership and minimal-equivalence queries, and in time polynomial in the size of a smallest doca recognizing the language.*

In the talk we skipped ε transitions in the doca. However that can also be done using the same technique.

Corollary

Polynomial approximation for minimization of doca.

Conclusion

Theorem

OL learns a doca equivalent to the teacher's doca using membership and minimal-equivalence queries, and in time polynomial in the size of a smallest doca recognizing the language.*

In the talk we skipped ε transitions in the doca. However that can also be done using the same technique.

Corollary

Polynomial approximation for minimization of doca.

Future work

- Replacing minimal-equivalence with equivalence query.
- Practical OL* algorithm.
- Improving running time of equivalence.
- Learning weighted models (like visibly OCA).

Thank You!