Reachability in Pushdown Register Automata

Andrzej Murawski
Steven Ramsay
Nikos Tzevelekos

University of Warwick
Queen Mary University of London

Highlights, Paris, September 2014

What this talk is about

This talk is about automata over infinite alphabets

In particular, we examine automata with two store mechanisms:

finitely many registers and a pushdown stack

We examine their reachability properties and establish complexity bounds

Why automata over *infinite* alphabets?

```
public void foo() {
  // Create new list
  List x = new ArrayList();

  x.add(1); x.add(2);
  Iterator i = x.iterator();
  Iterator j = x.iterator();
  i.next(); i.remove(); j.next();
}
```

finite alphabet is not a satisfactory abstraction here

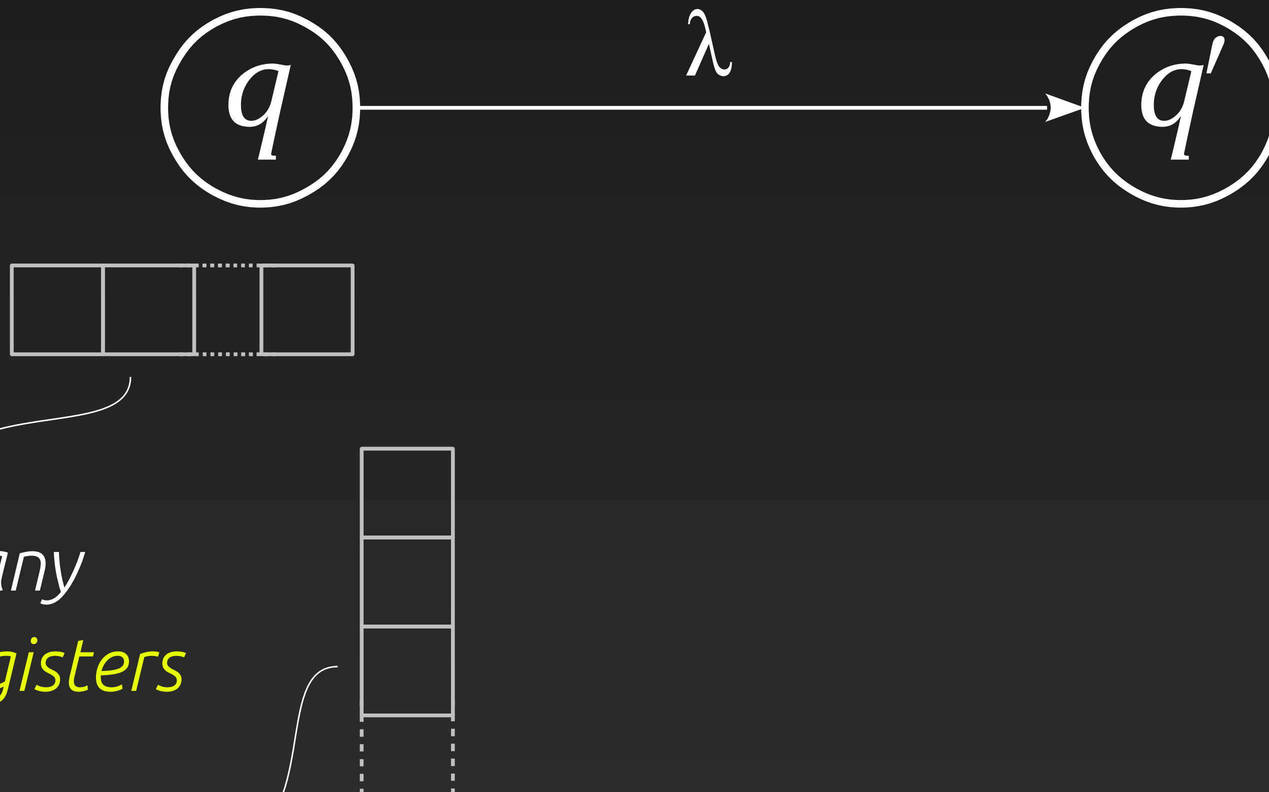
Let $\Sigma = \{a_1, a_2, \ldots, a_n, \ldots\}$ be an infinite alphabet of names

can only be compared for equality

[Cheng & Kaminski '98, Segoufin '06]

Pushdown Register Automata *(PDRA)*

Let $\Sigma = \{a_1, a_2, \ldots, a_n, \ldots\}$ be an infinite alphabet of names



finitely many (say R) registers

pushdown stack

registers & stack store names

[Cheng & Kaminski '98, Segoufin '06]

Pushdown Register Automata *(PDRA)*

Let $\Sigma = \{a_1, a_2, \ldots, a_n, \ldots\}$ be an infinite alphabet of names

finitely many (say R) registers

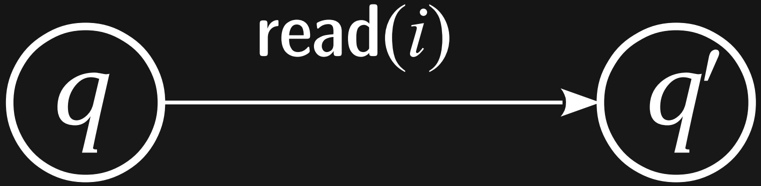
pushdown stack

registers & stack store names

Label λ of the form:

- **read**(i), $i \in \{1,\ldots,R\}$
- **refresh**(i), $i \in \{1,\ldots,R\}$
- **push**(i), $i \in \{1,\ldots,R\}$
- **pop**(i), $i \in \{1,\ldots,R\}$
- **pop-fresh**

Transitions:



Transitions:

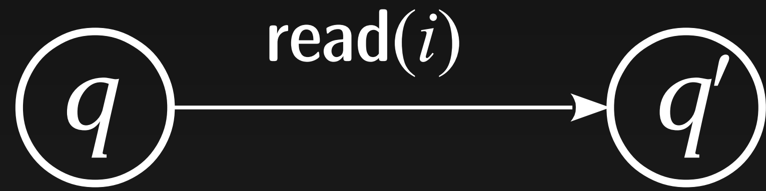
$$q \xrightarrow{\mathbf{read}(i)} q'$$

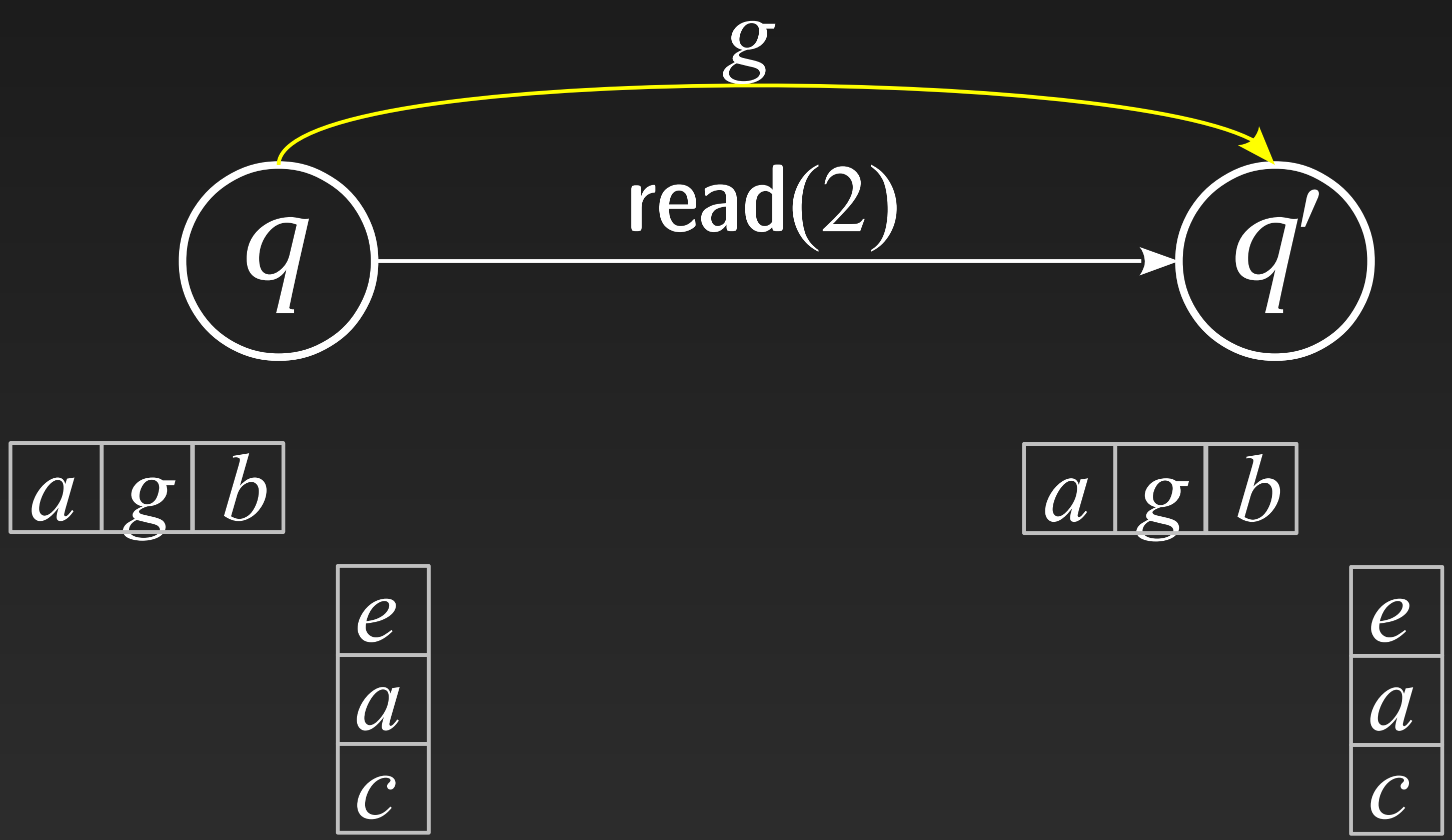
$$q \xrightarrow{\mathbf{read}(2)} q'$$

$$\begin{array}{|c|c|c|}\hline a & g & b \\ \hline\end{array}$$

$$\begin{array}{|c|}\hline e \\ \hline a \\ \hline c \\ \hline\end{array}$$

Transitions:





Transitions:

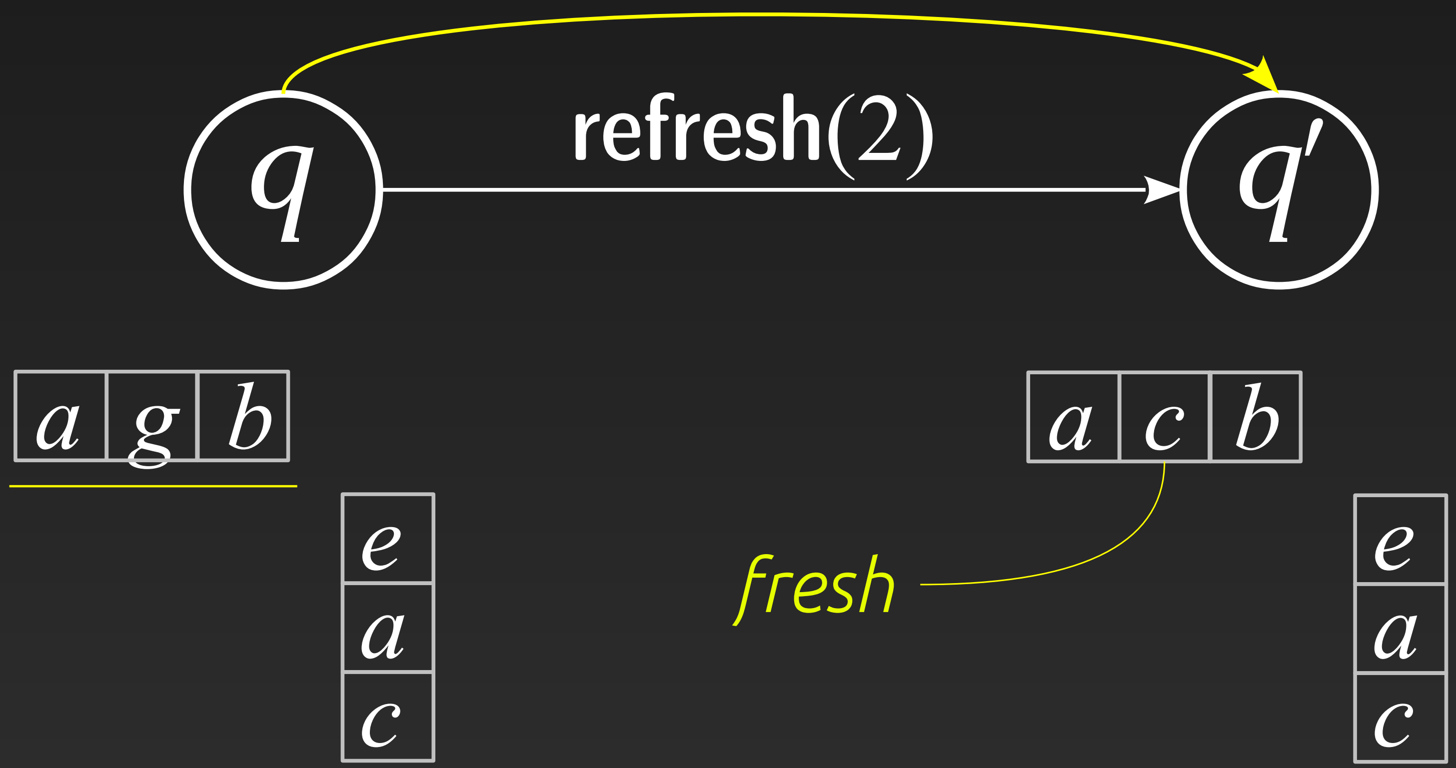
$$q \xrightarrow{\textbf{refresh}(i)} q'$$

$$q \xrightarrow{\textbf{refresh}(2)} q'$$

$$\boxed{a}\boxed{g}\boxed{b}$$

$$\begin{array}{|c|}\hline e \\ \hline a \\ \hline g \\ \hline \end{array}$$

Transitions:



Transitions:

$$q \xrightarrow{\mathsf{push}(i)} q'$$

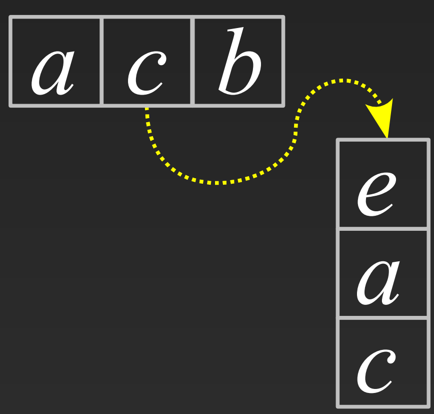
$$q \xrightarrow{\mathsf{push}(2)} q'$$

$$\boxed{a}\boxed{c}\boxed{b}$$

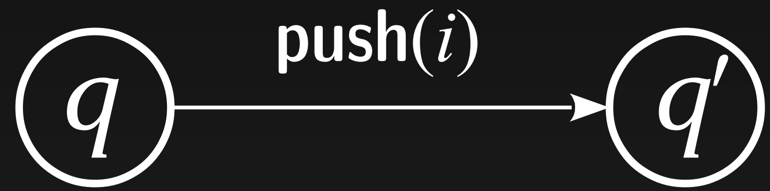
$$\begin{array}{|c|}\hline e \\ \hline a \\ \hline c \\ \hline \end{array}$$

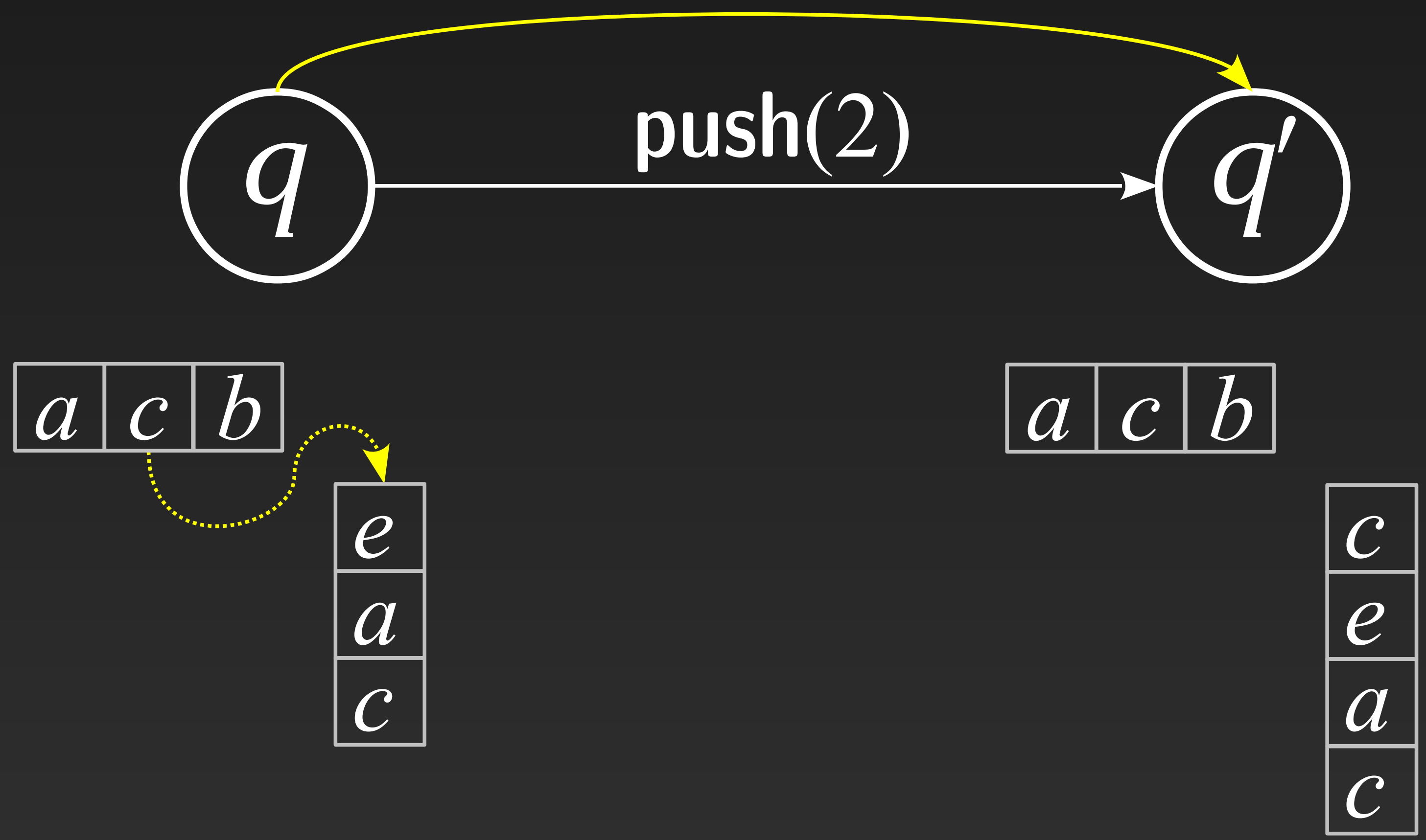
Transitions: $q \xrightarrow{\textsf{push}(i)} q'$

$$q \xrightarrow{\textsf{push}(2)} q'$$



Transitions:





Transitions: $q \xrightarrow{\textbf{pop}(i)} q'$

$$q \xrightarrow{\textbf{pop}(2)} q'$$

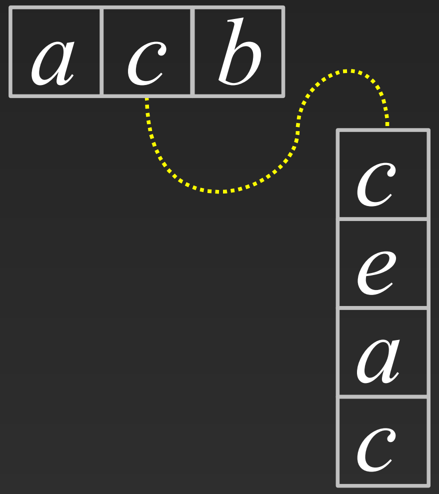
a	c	b

c
e
a
c

Transitions:

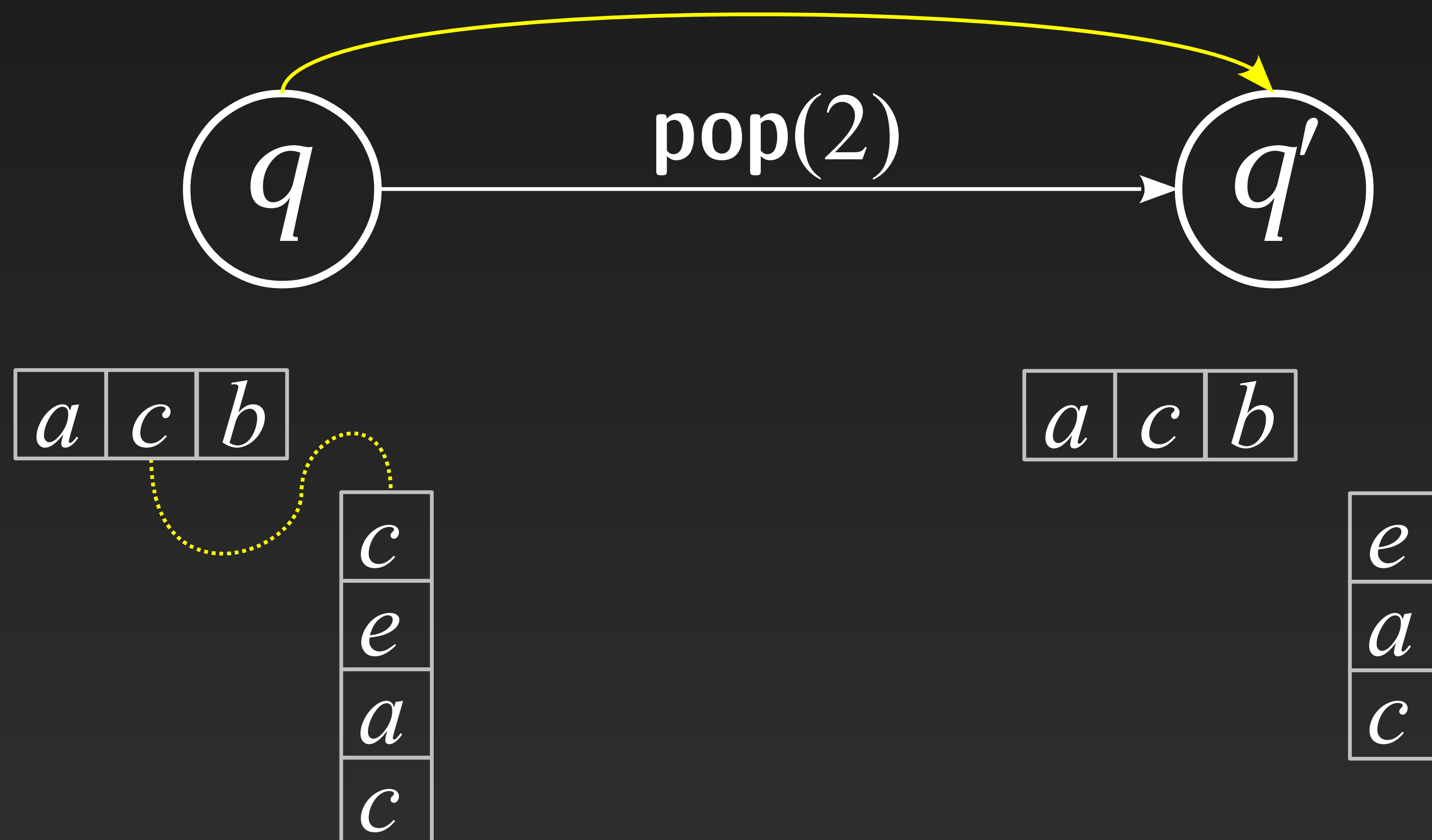
$$q \xrightarrow{\mathbf{pop}(i)} q'$$

$$q \xrightarrow{\mathbf{pop}(2)} q'$$



Transitions:

$$q \xrightarrow{\mathbf{pop}(i)} q'$$



Transitions:

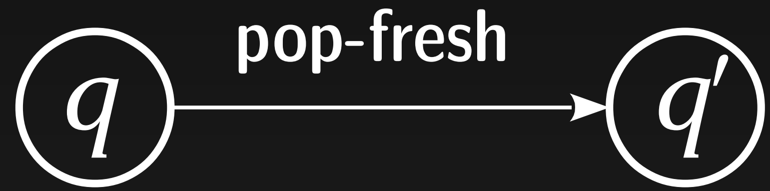
$$q \xrightarrow{\textbf{pop-fresh}} q'$$

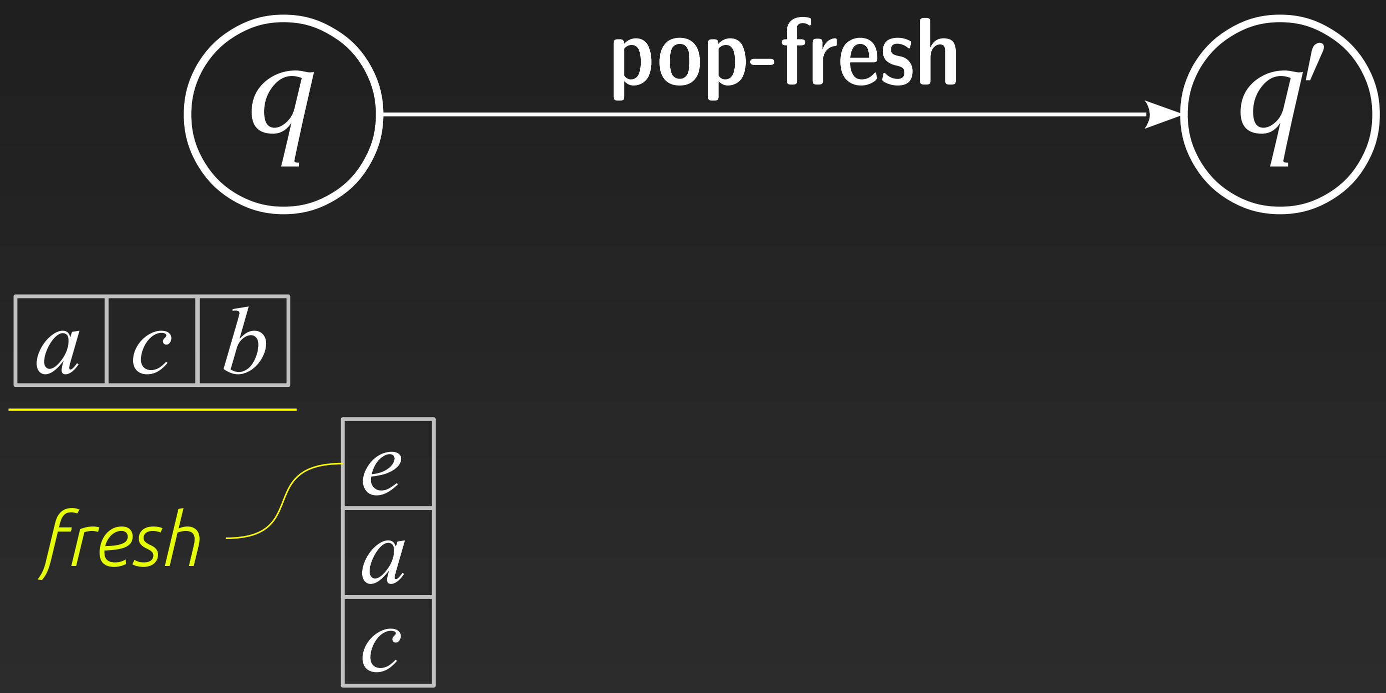
$$q \xrightarrow{\textbf{pop-fresh}} q'$$

a	c	b

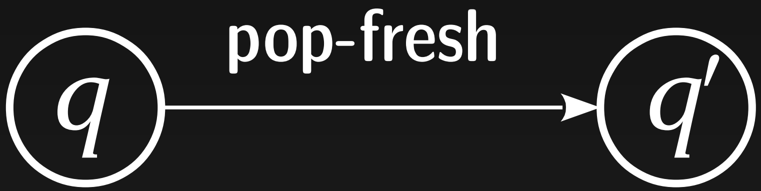
e
a
c

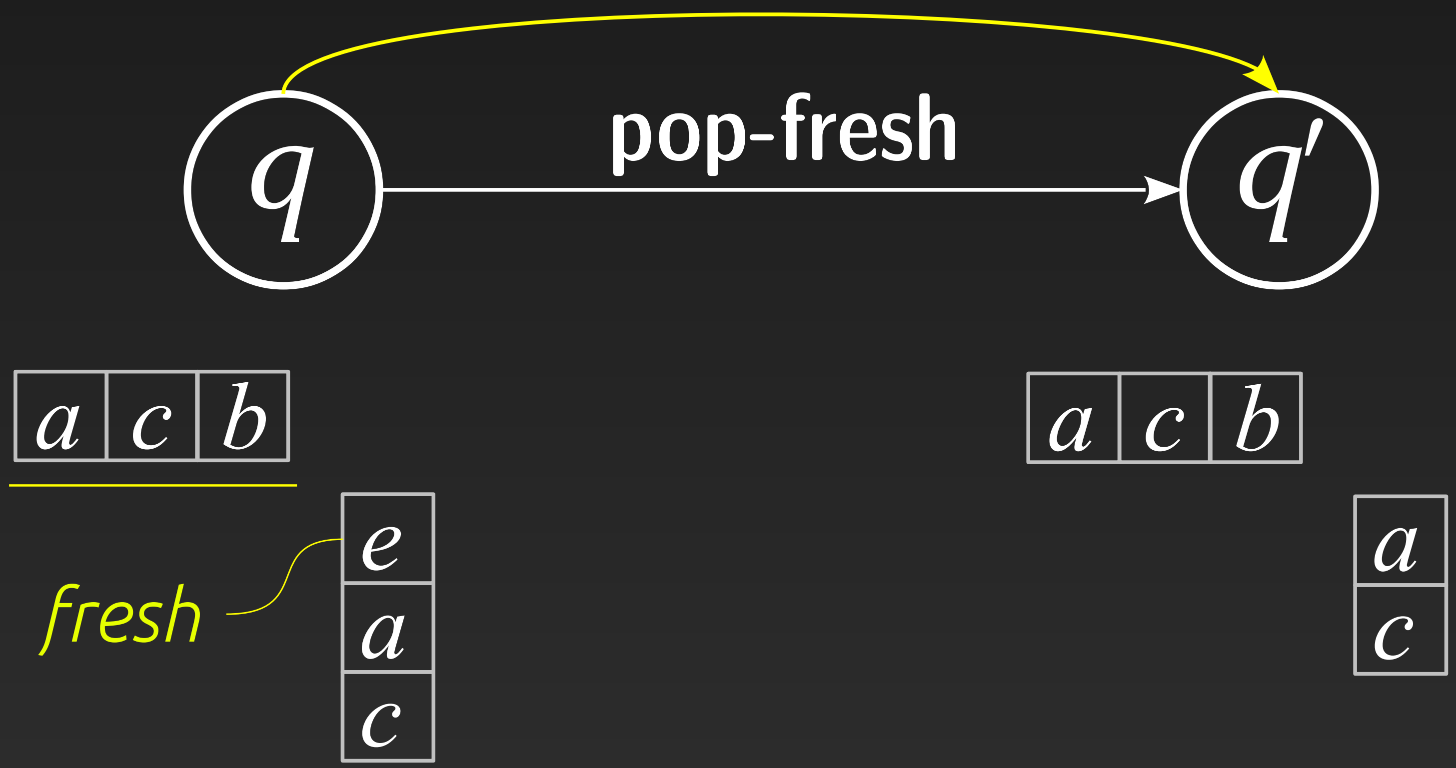
Transitions:





Transitions:

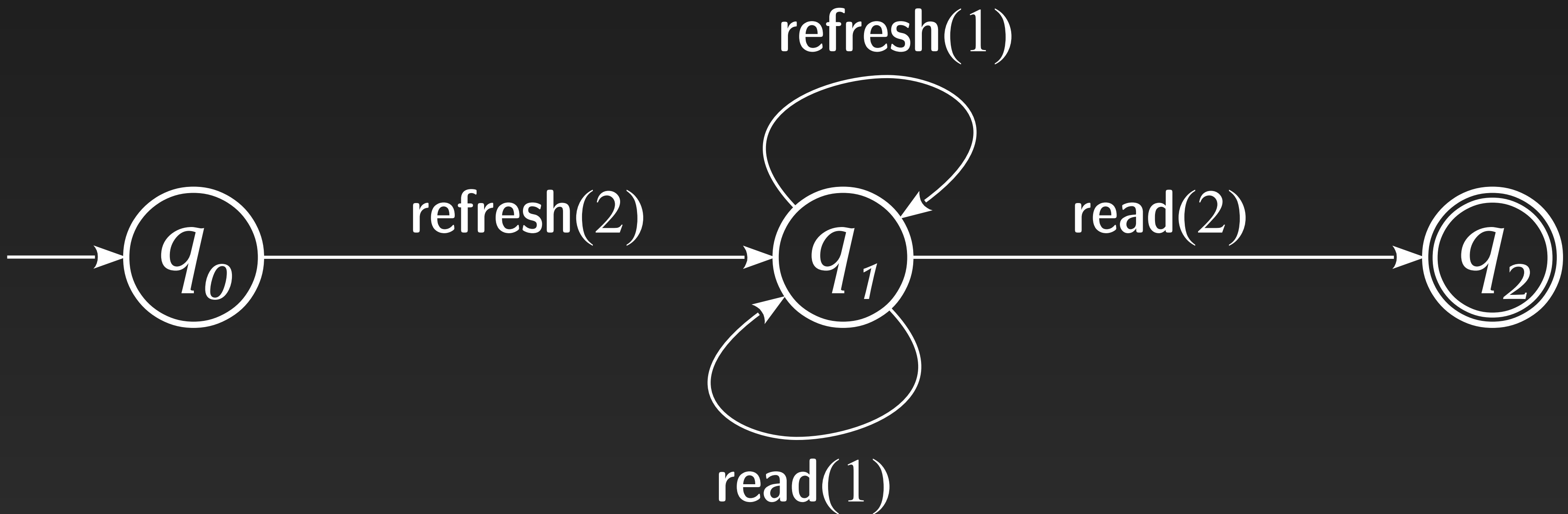




Example

$$L = \{ a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b \}$$

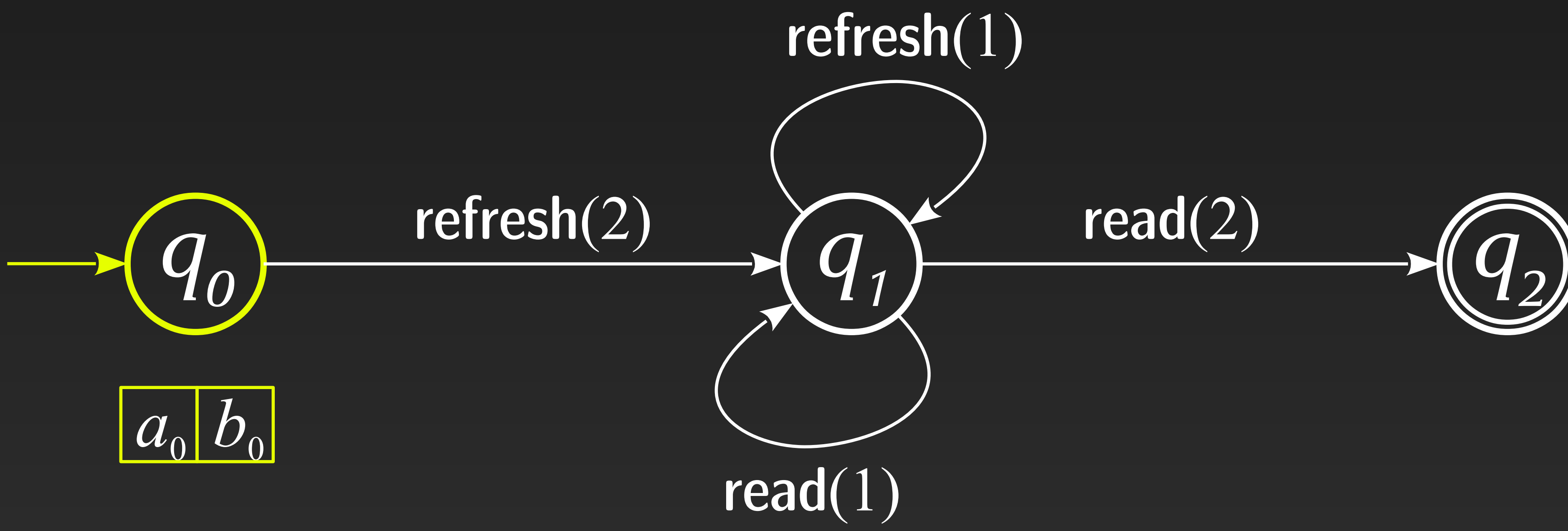
(all strings where last name is distinct from all previous ones)



Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b\ \}$$

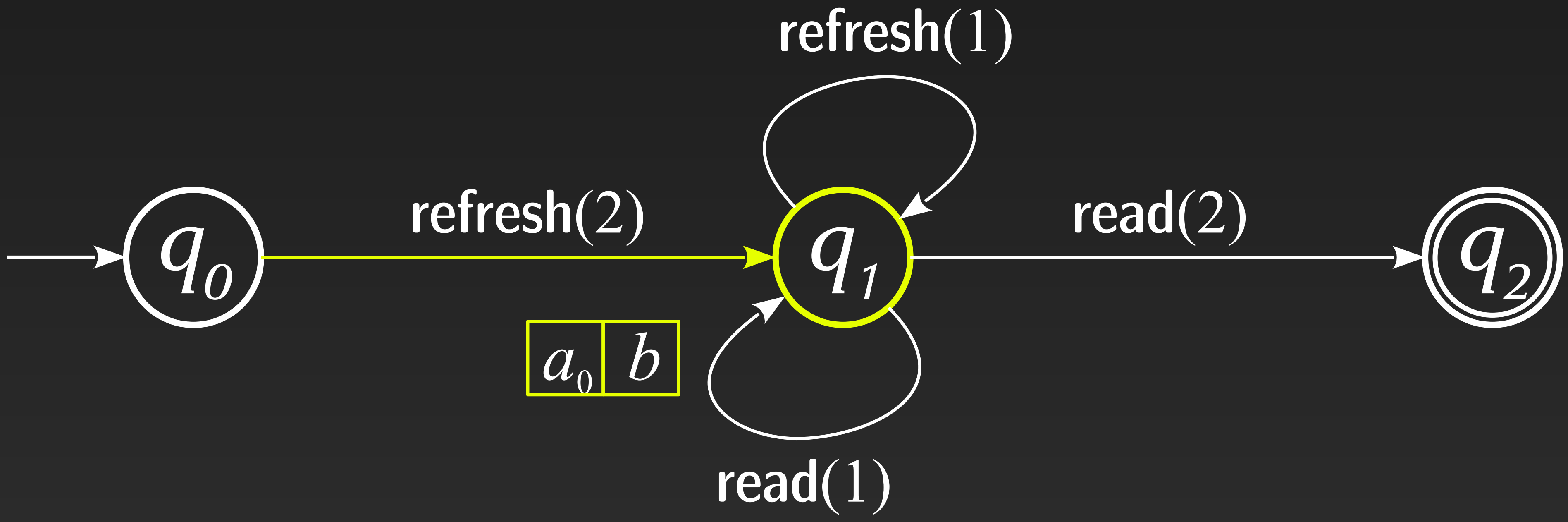
(all strings where last name is distinct from all previous ones)



Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid\ n \geq 0,\ \forall i \leq n.\ a_i \neq b\ \}$$

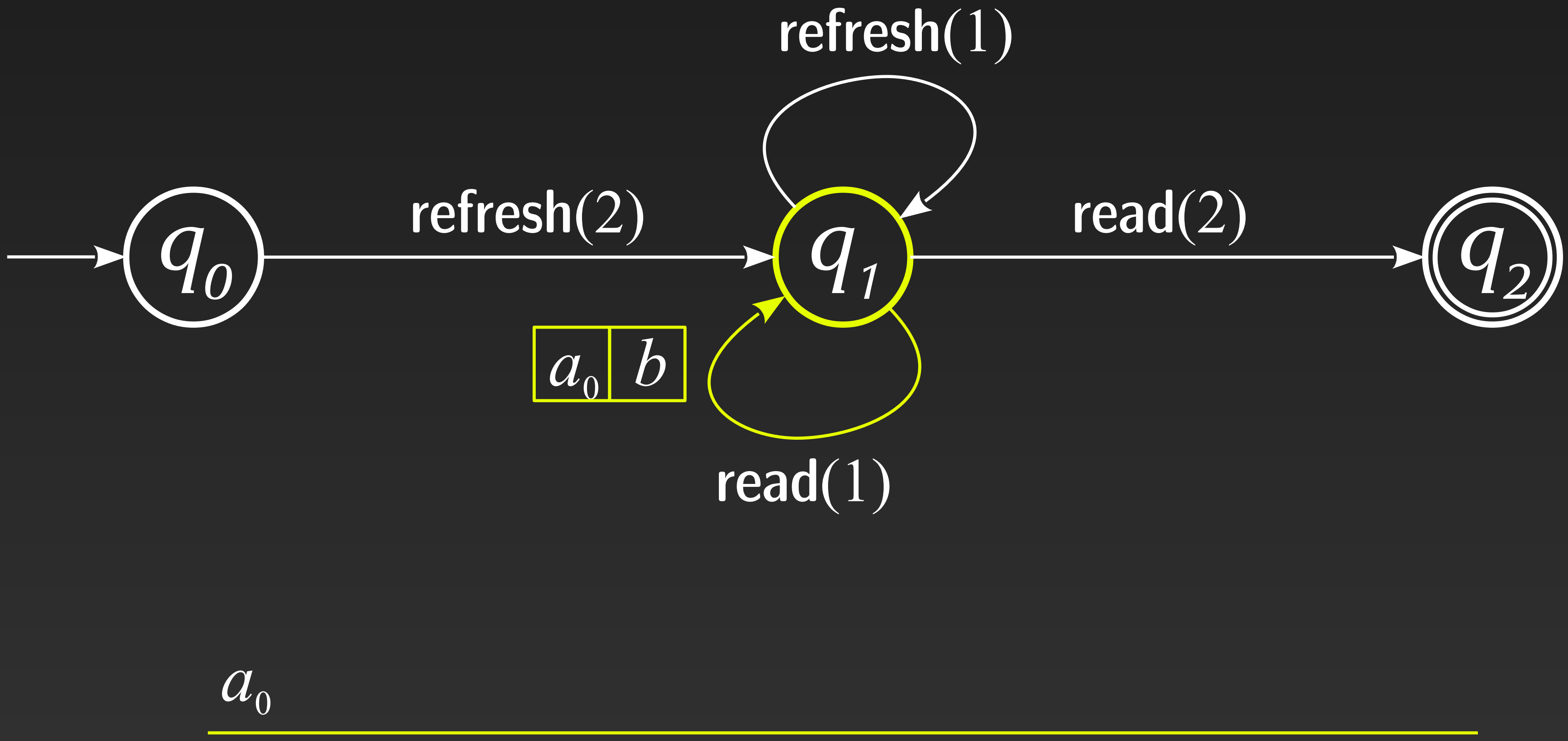
(all strings where last name is distinct from all previous ones)



Example

$$L = \{\, a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0,\ \forall i \leq n.\ a_i \neq b \,\}$$

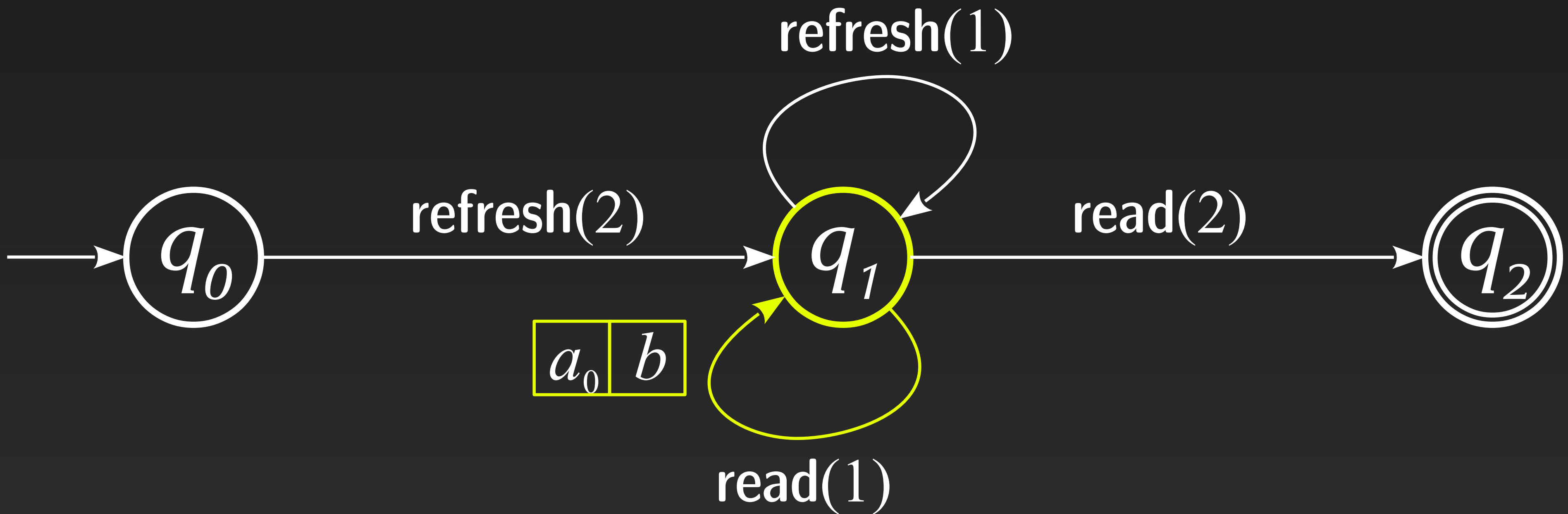
(all strings where last name is distinct from all previous ones)



Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid\ n \geq 0,\ \forall i \leq n.\ a_i \neq b\ \}$$

(all strings where last name is distinct from all previous ones)

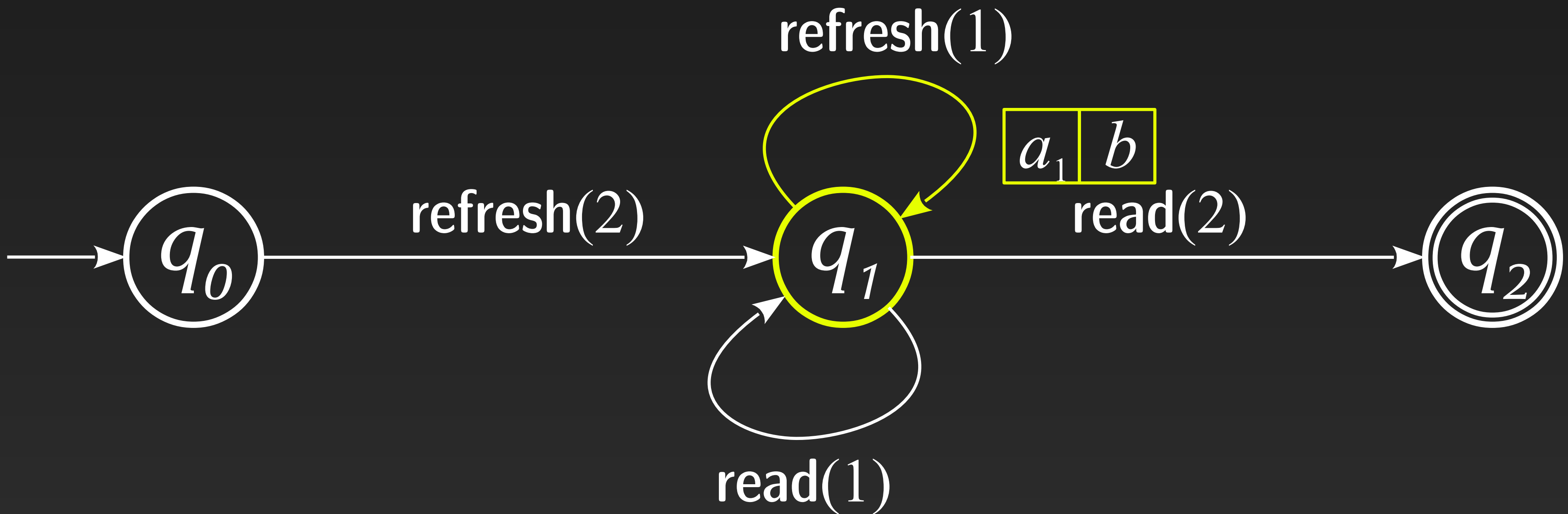


$a_0\, a_0$

Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b\ \}$$

(all strings where last name is distinct from all previous ones)

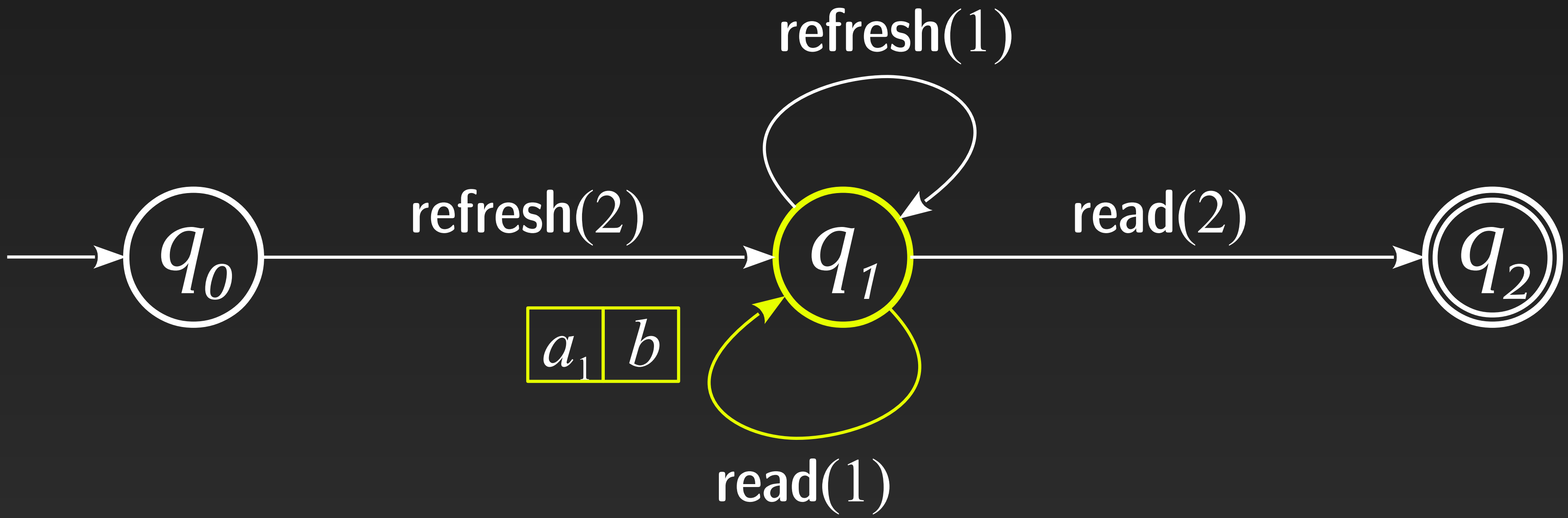


$a_0\, a_0$

Example

$$L = \{\, a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b \,\}$$

(all strings where last name is distinct from all previous ones)

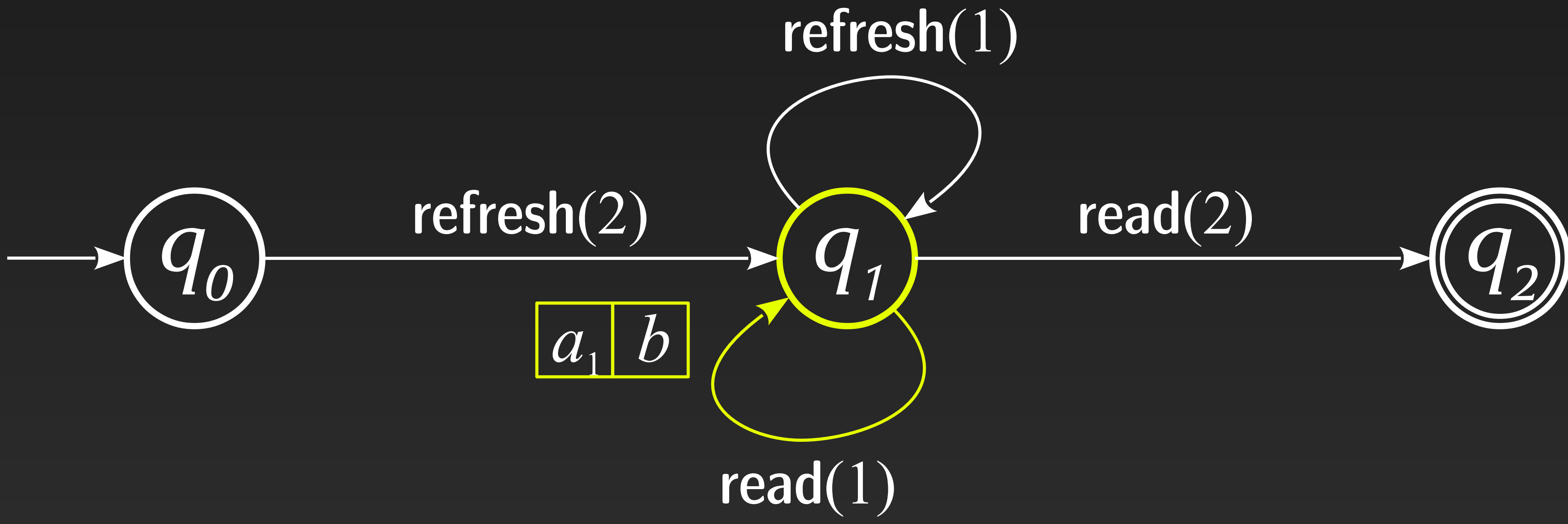


$a_0\, a_0\, a_1$

Example

$$L = \{\, a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0,\ \forall i \leq n.\ a_i \neq b \,\}$$

(all strings where last name is distinct from all previous ones)

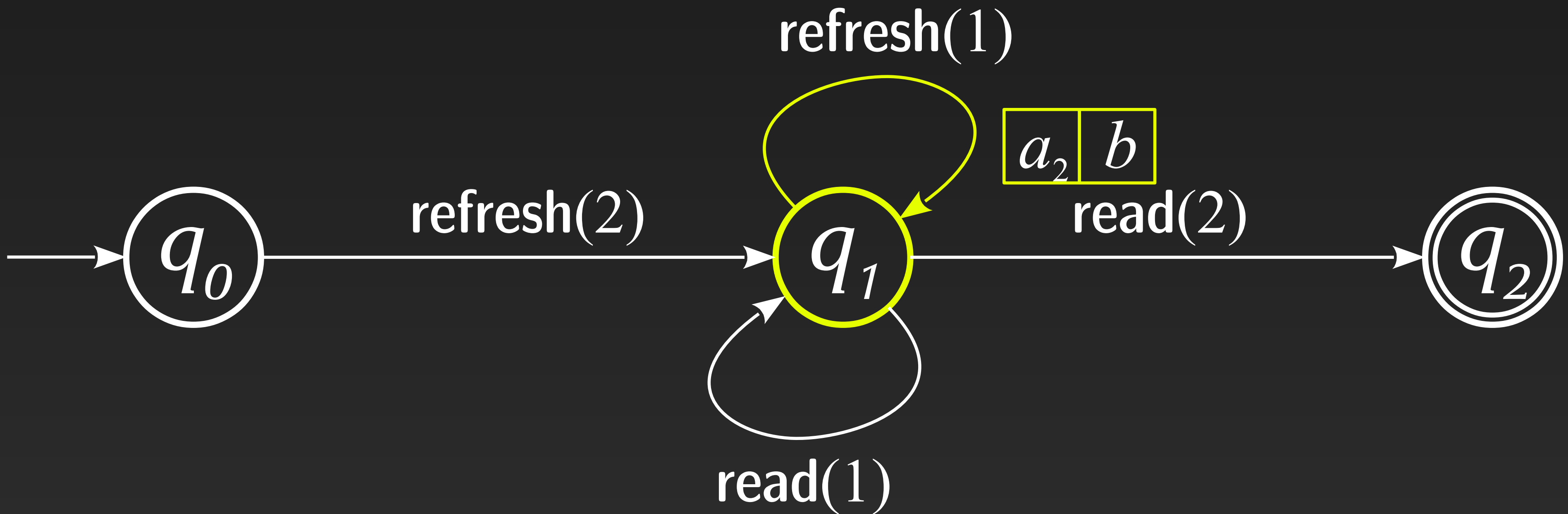


$a_0\, a_0\, a_1\, a_1\, a_1$

Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b\ \}$$

(all strings where last name is distinct from all previous ones)

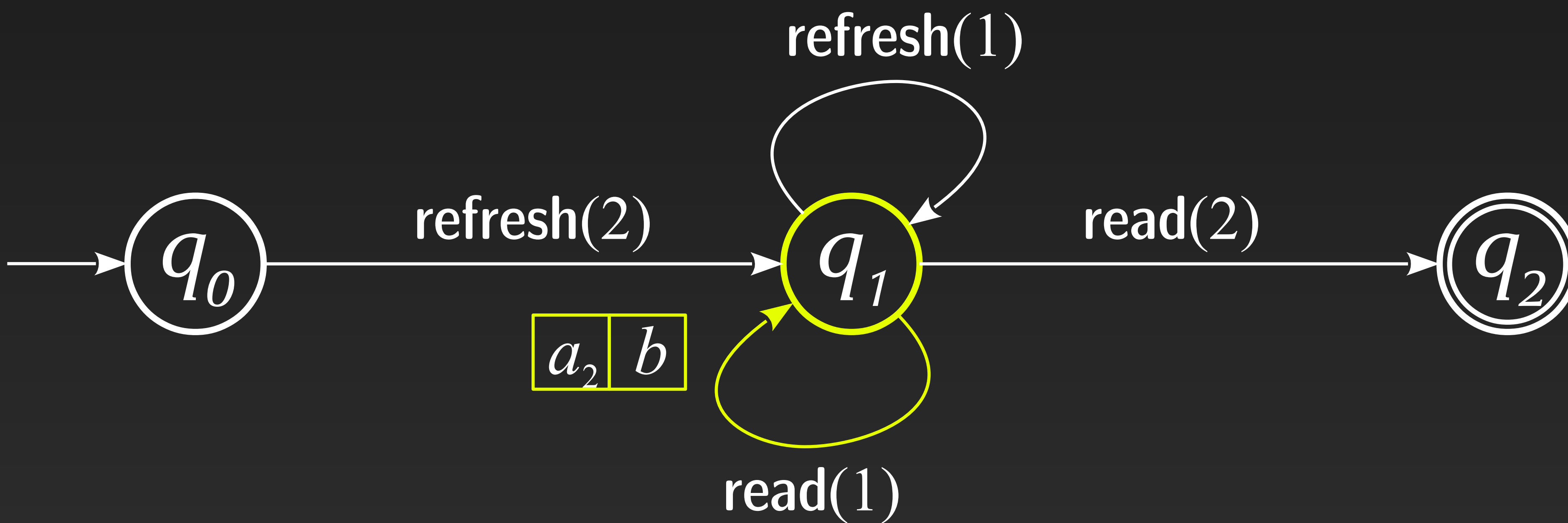


$a_0\, a_0\, a_1\, a_1\, a_1$

Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b\ \}$$

(all strings where last name is distinct from all previous ones)

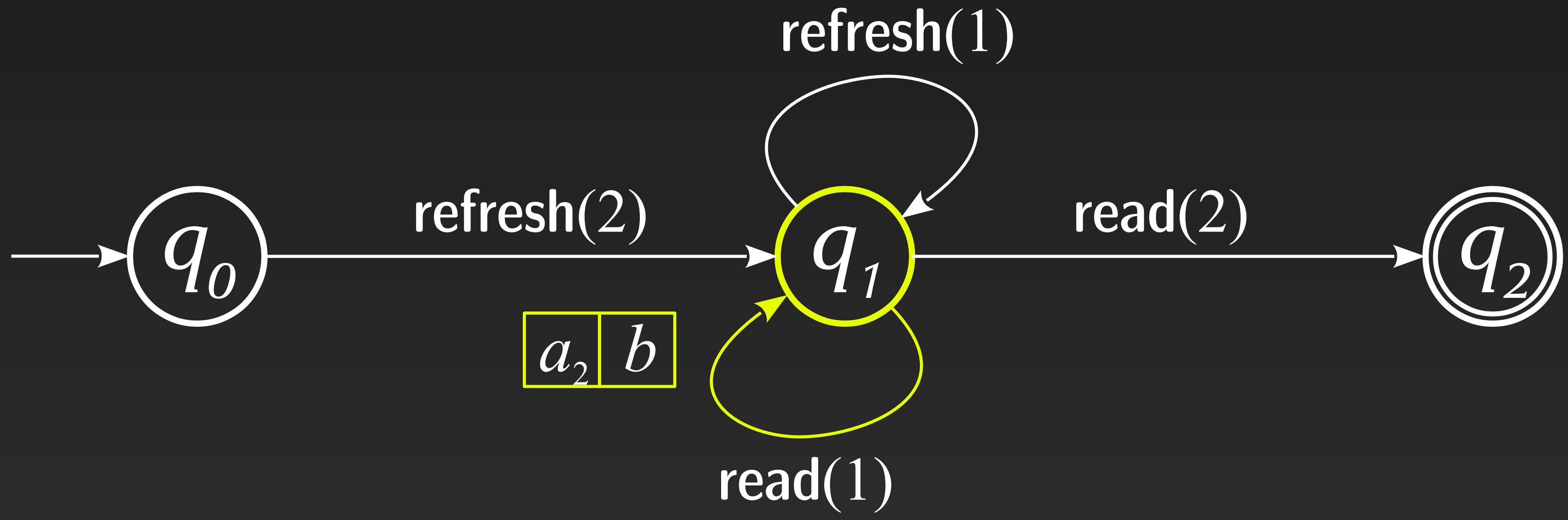


$a_0\, a_0\, a_1\, a_1\, a_1\, a_2$

Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid\ n \geq 0,\ \forall i \leq n.\ a_i \neq b\ \}$$

(all strings where last name is distinct from all previous ones)

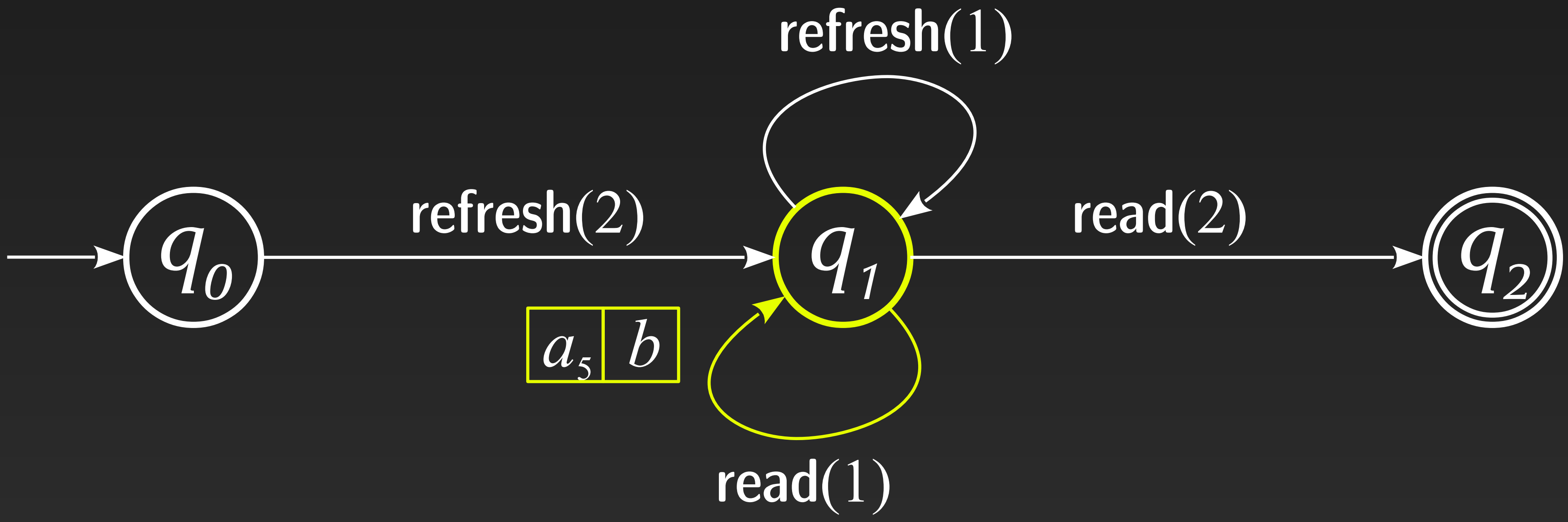


$a_0\, a_0\, a_1\, a_1\, a_1\, a_2\, a_2\, a_2\, a_2\, a_2$

Example

$$L = \{\ a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0,\ \forall i \leq n.\ a_i \neq b\ \}$$

(all strings where last name is distinct from all previous ones)

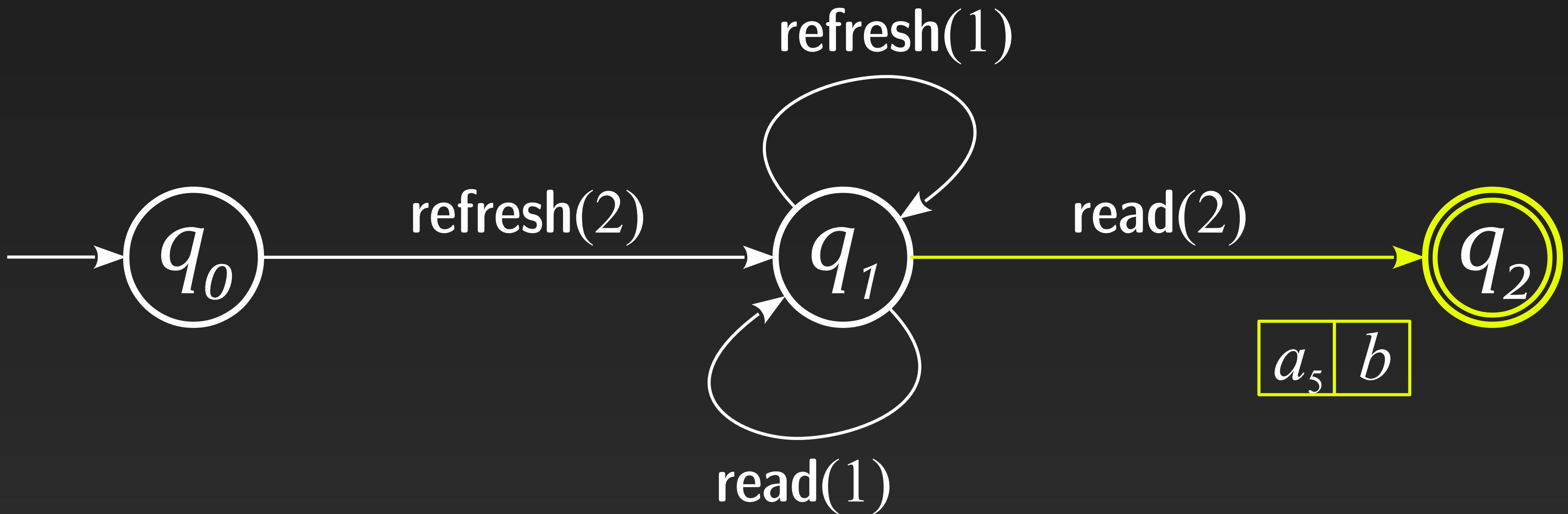


$a_0\, a_0\, a_1\, a_1\, a_1\, a_2\, a_2\, a_2\, a_2\, a_2\, a_3\, a_3\, a_4\, a_4\, a_5\, a_5\, a_5\, a_5\, a_5\, a_5$

Example

$$L = \{\, a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0, \forall i \leq n.\ a_i \neq b \,\}$$

(all strings where last name is distinct from all previous ones)



$$a_0\, a_0\, a_1\, a_1\, a_1\, a_2\, a_2\, a_2\, a_2\, a_2\, a_3\, a_3\, a_4\, a_4\, a_5\, a_5\, a_5\, a_5\, a_5\, a_5\, b$$

Limited distinguish-ability

$$L = \{ a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i{\neq}j.\ a_i{\neq}a_j \}$$

(all strings of distinct names)

Limited distinguish-ability

$$L = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i{\neq}j.\ a_i{\neq}a_j \,\}$$

(all strings of distinct names)

A *PDRS (PDR System)* is a PDRA without **read** transitions

Lemma: Let S be a PDRS with R-many registers.
For any pair of states q_1 and q_2, if there is a run between them (from empty stack to empty stack) then there is one involving at most $3R$ names.

Limited distinguish-ability

$$\cancel{L = \{ a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j.\ a_i \neq a_j \}}$$

~~*(all strings of distinct names)*~~

A *PDRS (PDR System)* is a PDRA without **read** transitions

Lemma: Let S be a PDRS with R-many registers.
For any pair of states q_1 and q_2, if there is a run between them (from empty stack to empty stack) then there is one involving at most $3R$ names.

Limited distinguish-ability

$$\cancel{L = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i \neq j.\ a_i \neq a_j \,\}}$$

~~*(all strings of distinct names)*~~

A *PDRS (PDR System)* is a PDRA without **read** transitions

Lemma: Let S be a PDRS with R-many registers.
For any pair of states q_1 and q_2, if there is a run between them (from empty stack to empty stack) then there is one involving at most $3R$ names.

Conversely, there is a PDRS with R registers whose runs to a designated state involve exactly $3R$ names.

Reachability

R-PRDS Reach: *Given a PDRS S with R registers and a state q, is there a run of S to q?*

Reachability

R-PRDS Reach: *Given a PDRS S with R registers and a state q, is there a run of S to q?*

Theorem: R-PRDS Reach is EXPTIME-complete.

For EXPTIME solvability:

- By previous Lemma, $3R$ names suffice: $\Sigma' = \{a_1, \ldots, a_{3R}\}$
- so, registers can be encoded inside states: $Q' = Q \times R^{3R}$
- and we reduce to PDA reachability (PTIME)

EXPTIME-hardness

For hardness, the argument is harder...

- We reduce from PSPACE Turing machines with stack
- crux of the reduction is the simulation of the tape
    - if we allowed name repetitions in registers then easy:

$$\boxed{0}\boxed{1}\boxed{0}\boxed{0}\boxed{1} \quad \mapsto \quad \boxed{a_1}\boxed{a_1}\boxed{a_2}\boxed{a_2'}\boxed{a_3}\boxed{a_3}\boxed{a_4}\boxed{a_4}\boxed{a_5}\boxed{a_5'}$$

EXPTIME-hardness

For hardness, the argument is harder…

- We reduce from PSPACE Turing machines with stack
- crux of the reduction is the simulation of the tape
  - if we allowed name repetitions in registers then easy:

$$0\ 1\ 0\ 0\ 1 \mapsto a_1\ a_1\ a_2\ a_2'\ a_3\ a_3\ a_4\ a_4\ a_5\ a_5'$$

  - now, instead, we use *mask encodings* + the stack:

$$0\ 1\ 0\ 0\ 1 \mapsto a_1\ a_1'\ a_2'\ a_2\ a_3\ a_3'\ a_4\ a_4'\ a_5'\ a_5$$

EXPTIME-hardness

For hardness, the argument is harder…

- We reduce from PSPACE Turing machines with stack
- crux of the reduction is the simulation of the tape
  - if we allowed name repetitions in registers then easy:

$$\boxed{0 \mid 1 \mid 0 \mid 0 \mid 1} \mapsto \boxed{a_1 \mid a_1 \mid a_2 \mid a_2' \mid a_3 \mid a_3 \mid a_4 \mid a_4 \mid a_5 \mid a_5'}$$

  - now, instead, we use *mask encodings* + the stack:

$$\boxed{0 \mid 1 \mid 0 \mid 0 \mid 1} \mapsto \begin{array}{ll} \boxed{a_1 \mid a_1' \mid a_2' \mid a_2 \mid a_3 \mid a_3' \mid a_4 \mid a_4' \mid a_5' \mid a_5} & \textit{enc.} \\ \boxed{a_1 \mid a_1' \mid a_2 \mid a_2' \mid a_3 \mid a_3' \mid a_4 \mid a_4' \mid a_5 \mid a_5'} & \textit{mask} \end{array}$$

hygiene to ensure that the masks are soundly applied and the stack is not messed up…

EXPTIME-hardness

without the stack, repetitions give a huge complexity gap!

For hardness, the argument is harder…

- We reduce from PSPACE Turing machines with stack
- crux of the reduction is the simulation of the tape
  - if we allowed <u>name repetitions</u> in registers then easy:

$$0\ 1\ 0\ 0\ 1 \mapsto a_1\ a_1\ a_2\ a_2'\ a_3\ a_3\ a_4\ a_4\ a_5\ a_5'$$

  - now, instead, we use *mask encodings* + the stack:

$$0\ 1\ 0\ 0\ 1 \mapsto \begin{array}{ll} a_1\ a_1'\ a_2'\ a_2\ a_3\ a_3'\ a_4\ a_4'\ a_5'\ a_5 & \textit{enc.} \\ a_1\ a_1'\ a_2\ a_2'\ a_3\ a_3'\ a_4\ a_4'\ a_5\ a_5' & \textit{mask} \end{array}$$

hygiene to ensure that the masks are soundly applied and the stack is not messed up…

More results!

Global reachability: For a PDRS S, capture all configurations from which S can reach a specified set of configurations

Theorem: Register automata (i.e. PDRA but no stack) capture configurations that can reach "regular" sets.

- *cf. "saturation" technique of Bouajjani, Esparza and Maler*

Higher-order PDRS: Extensions with stacks of stacks

Theorem: Reachability is undecidable at order 2.

- *reduce from Pebble automata language emptiness*

Concluding

Reachability analysis for infinite-alphabet systems with:

- finitely many registers & a pushdown stack

paper appears at MFCS 2014

Further directions include:

- Implementation & use in automata-base verification
  - e.g *algorithmic game semantics*
- Bisimulation analysis
- dPDRA equivalence

Concluding

thanks

Reachability analysis for infinite-alphabet systems with:

- finitely many registers & a pushdown stack

paper appears at MFCS 2014

Further directions include:

- Implementation & use in automata-base verification
  - e.g *algorithmic game semantics*
- Bisimulation analysis
- dPDRA equivalence