Automata (& programs) over infinite alphabets

Nikos Tzevelekos

Queen Mary University of London

OASIS 10th Anniversary Oxford

Supported by a Royal Academy of Engineering Research Fellowship

Why *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();
}
```

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet of input symbols.

A … automaton A over Σ is a tuple $A = <\ldots>$ such that …

Why *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();
}
```

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet of input symbols.

A … automaton A over Σ is a tuple $A = <\ldots>$ such that …

Finite alphabet not satisfactory for modelling or verifying resourceful code (and computation)

Why *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();
}
```

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be an alphabet of input symbols.

A … automaton A over Σ is a tuple $A = \langle \ldots \rangle$ such that …

Finite alphabet not satisfactory for modelling or verifying resourceful code (and computation)

Lots of interest also from XML model-checking community!

What this talk is about

This talk is about automata over infinite alphabets

We take motivation from program modelling & verification

We concentrate on infinite alphabet extensions of Finite-State Automata and Pushdown Automata obtained by addition of name registers

We study their expressiveness and algorithmic properties with particular focus on reachability

Automata theory in infinite alphabets

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

can only be compared for equality

Automata theory in infinite alphabets

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

- examine languages over Σ^*
  - or, languages over $(F \cup \Sigma)^*$
  - or, languages over $(F \times \Sigma)^*$
    - usually called *data words* (XML)

can only be compared for equality

a finite set of constants

Automata theory in infinite alphabets

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

- examine languages over Σ^*
  - or, languages over $(F \cup \Sigma)^*$
  - or, languages over $(F \times \Sigma)^*$
    - usually called *data words* (XML)
- look for notions of regularity, CFGs, etc.
- devise effective algorithms for reachability, membership, etc.

can only be compared for equality

a finite set of constants

[Kaminski & Fraencez '94]

Register Automata (RA)

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

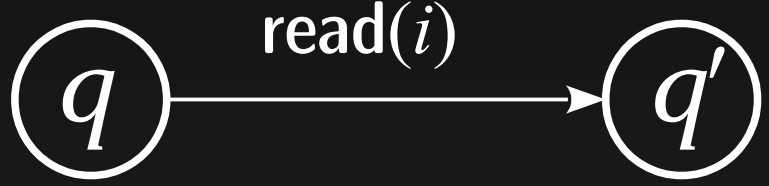
finitely many (say R) registers

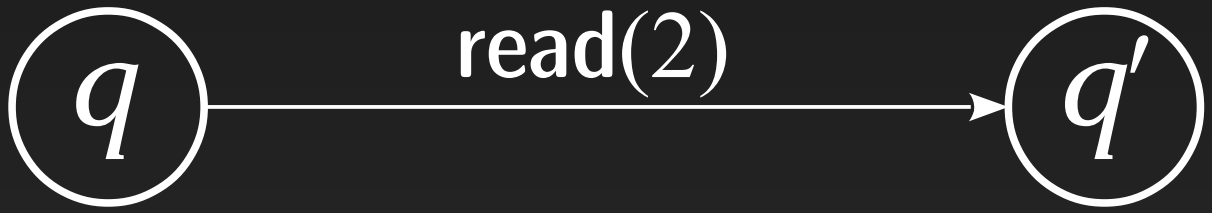
registers store names

Label λ of the form:

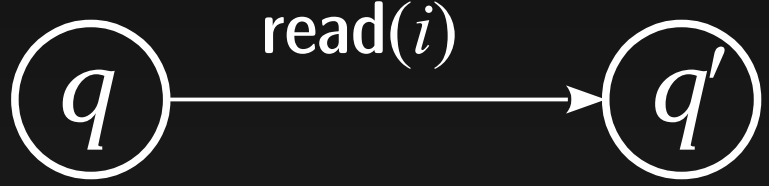
- **read**(i), $i \in \{1,\ldots,R\}$
- **fresh**(i), $i \in \{1,\ldots,R\}$

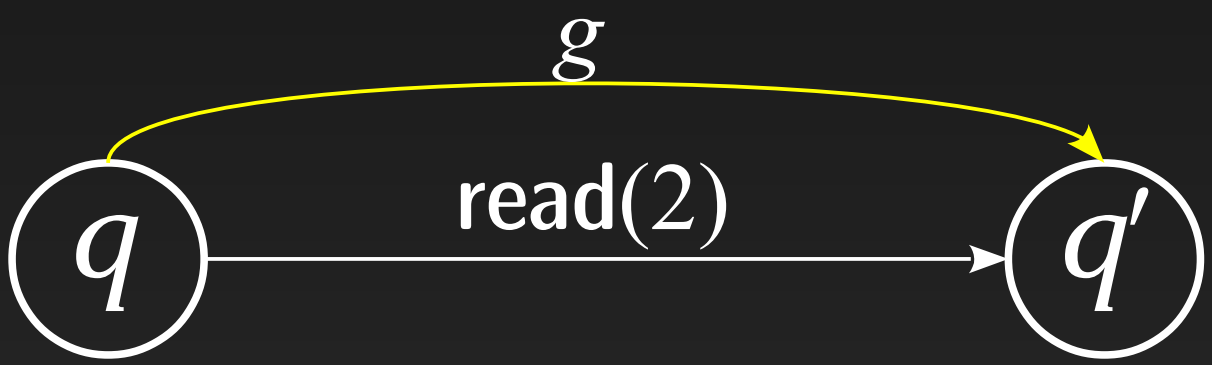
Transitions:



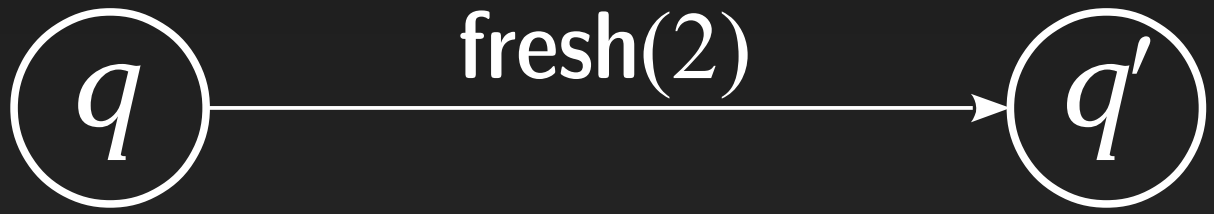


Transitions:



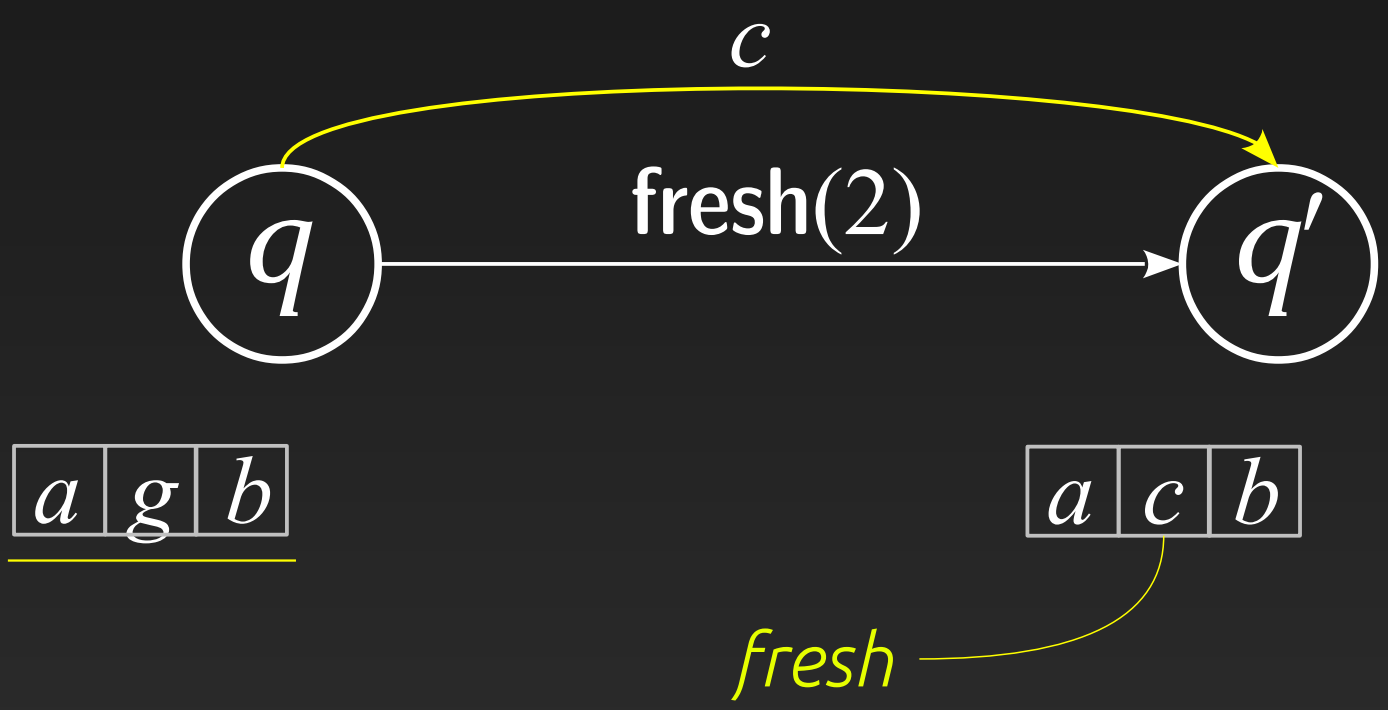


Transitions:



a	g	b

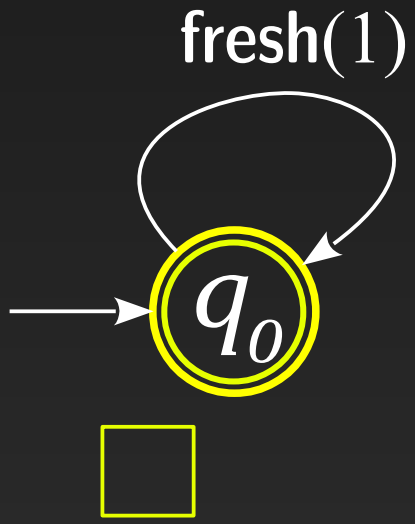
Transitions:



Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)



Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

a

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

$a\,b$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

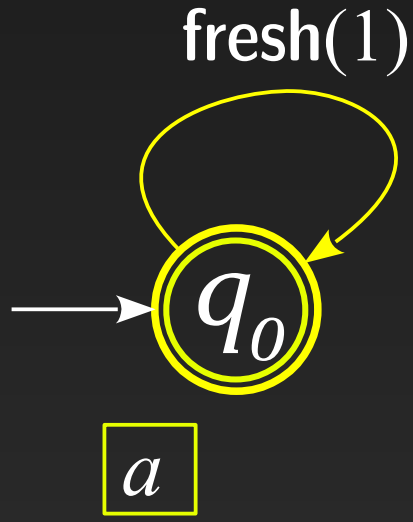
(all strings where each name is distinct from its predecessor)

$a\,b\,c$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

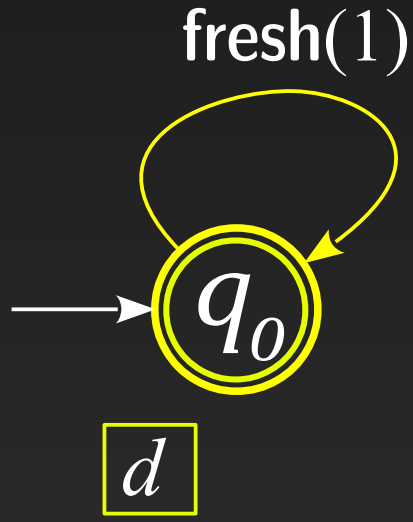


$a\,b\,c\,a$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)



$a\,b\,c\,a\,d$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

$a\,b\,c\,a\,d\,e$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

$a\,b\,c\,a\,d\,e\,b$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

$$a\,b\,c\,a\,d\,e\,b\,a\,g\,c\,a\,b$$

Example

$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

(all strings where each name is distinct from its predecessor)

$$abcadebagcab\ and\ we\ love\ OASIS$$

RA properties

- Capture regularity when Σ restricted to finite
  - Closed under $\cup, \cap, \bullet, *$.
  - not closed under complement & not determinisable

[Kaminski & Fraencez '94]

RA properties

- Capture regularity when Σ restricted to finite
  - Closed under $\cup, \cap, \bullet, *$.
  - not closed under complement & not determinisable

[Kaminski & Fraencez '94]

$$L_2 = \{ a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j.\ a_i \neq a_j \}$$

(all strings of pairwise distinct names)

RA properties

- Capture regularity when Σ restricted to finite
  - Closed under $\cup, \cap, \bullet, *$.
  - not closed under complement & not determinisable

[Kaminski & Fraencez '94]

- Universality / equivalence undecidable

[Neven, Schwentick & Vianu '01]

- Decidable emptiness:
  - complexity depends on register “mode” (NL → NP → PSPACE)

[Sakamoto & Ikeda '00; Demri & Lazić '09]

Example revisited

```
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();
}
```

here is a safety property φ:

if an iterator modifies its collection x then other iterators of x become invalid

e.g. the code on the left is bad.

We can express such “chaining” properties using RAs

- and dynamically verify them!

[Grigore, Distefano, Petersen & Tz '13]

Example revisited

```
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();
}
```

here is a safety property φ:

if an iterator modifies its collection x then other iterators of x become invalid

e.g. the code on the left is bad.

We can express such “chaining” properties using RAs

- and dynamically verify them!

[Grigore, Distefano, Petersen & Tz '13]

However, RAs do not capture `new` and hence cannot give abstract models of such programs (e.g. for static analysis)

[Tz '11]

Fresh-Register Automata (FRA)

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

finitely many (say R) registers

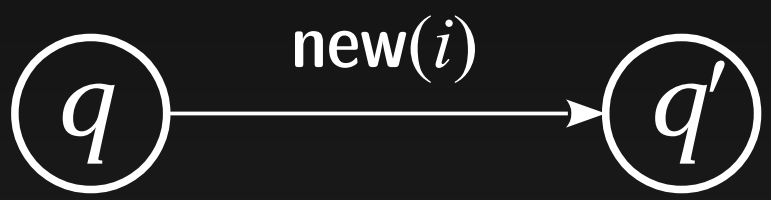
registers store names

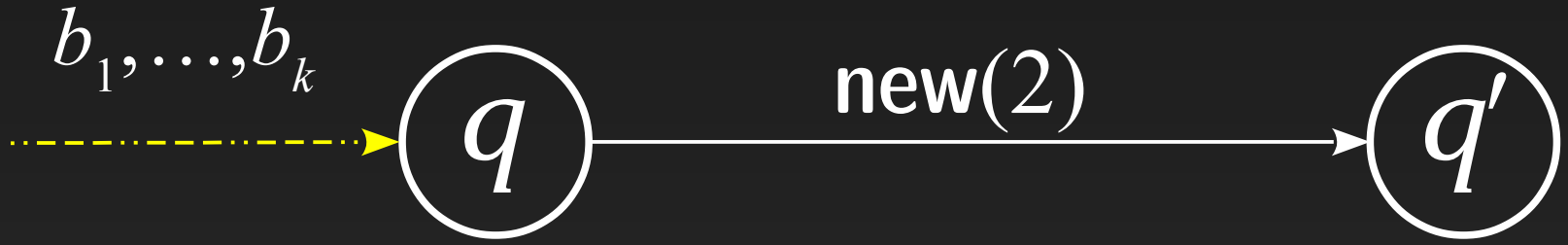
Label λ of the form:

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

global freshness oracle

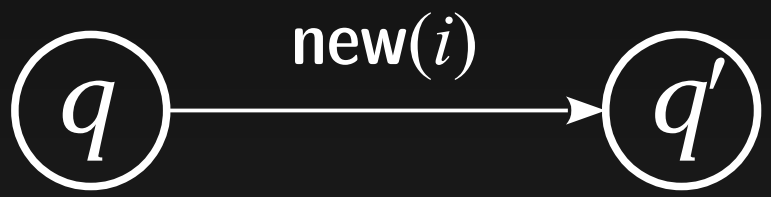
Transitions:

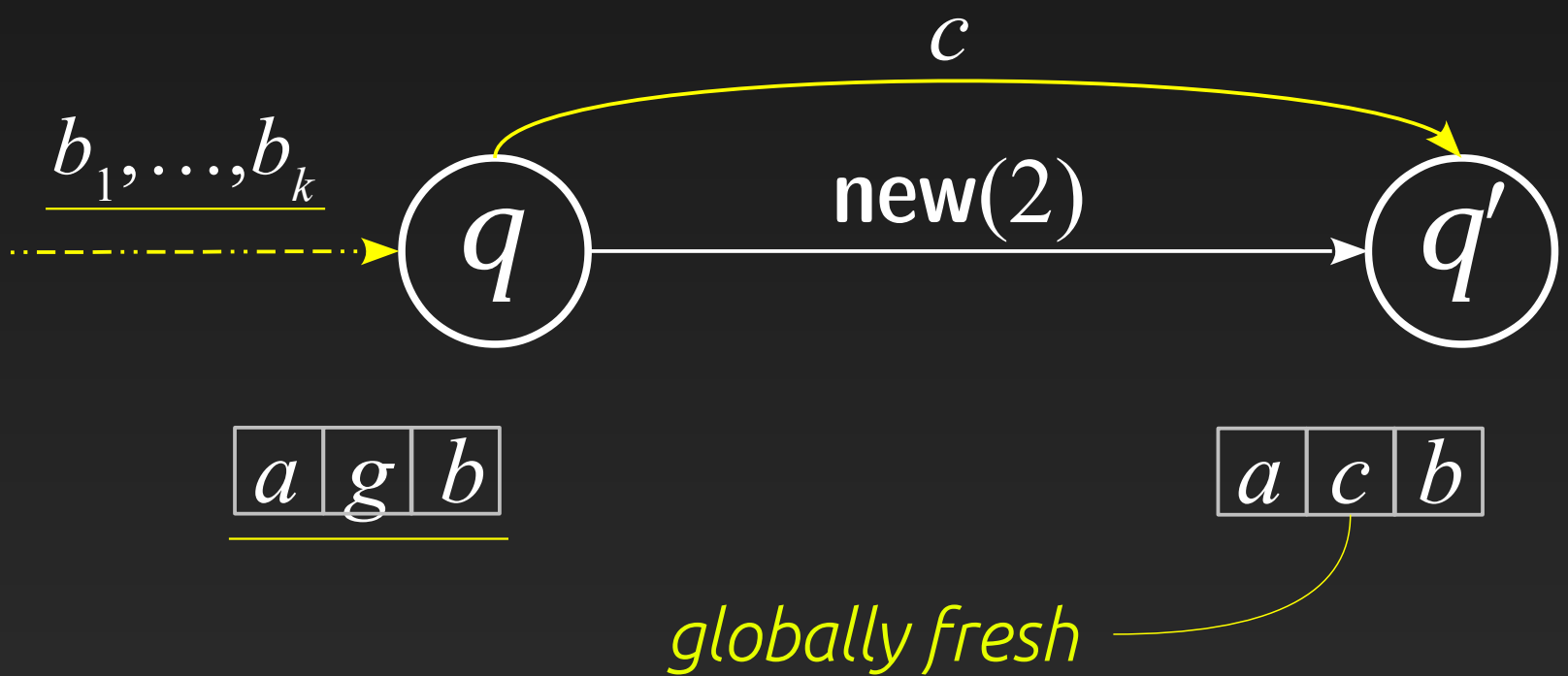




a	g	b

Transitions:

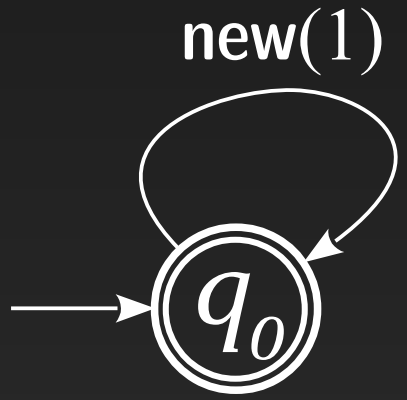




Examples

$$L_2 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i \neq j.\ a_i \neq a_j \,\}$$

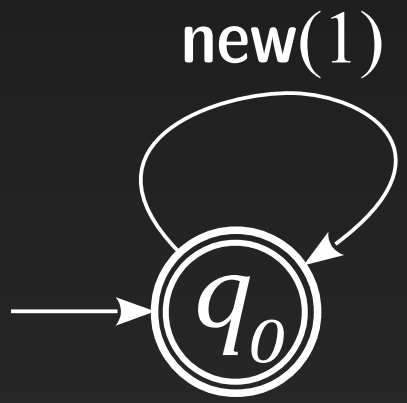
(all strings of pairwise distinct names)

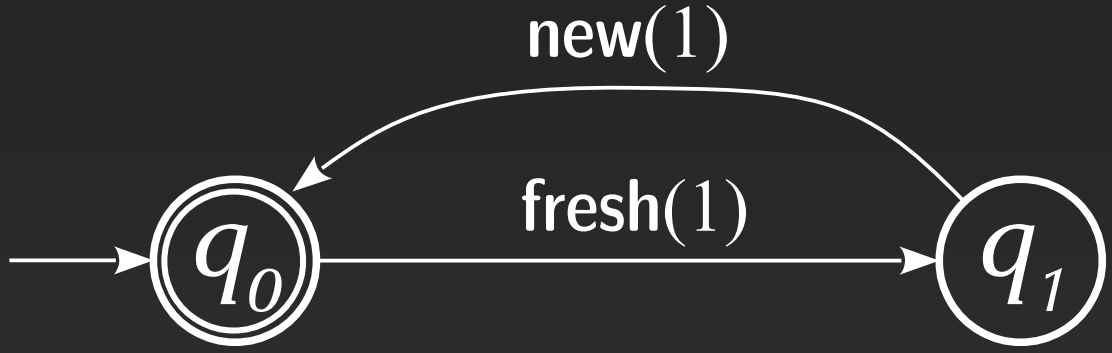


Examples

$$L_2 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i \neq j.\ a_i \neq a_j \,\}$$

(all strings of pairwise distinct names)

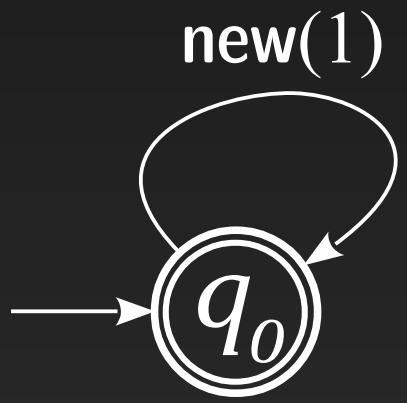


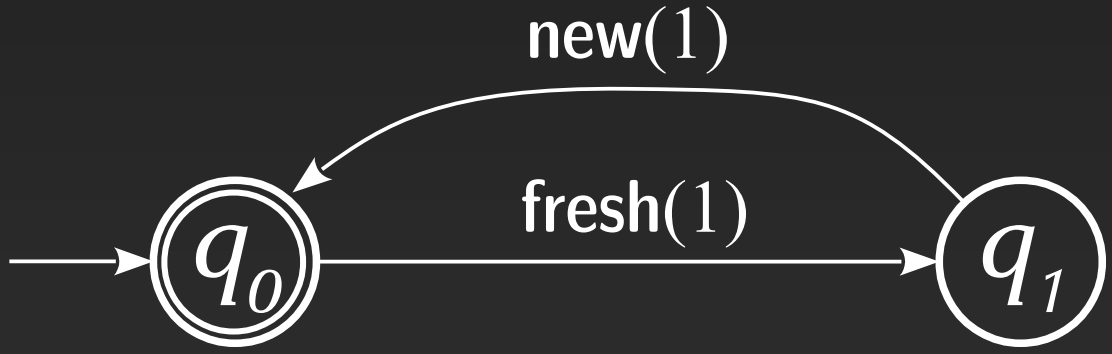


Examples

$$L_2 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i \neq j.\ a_i \neq a_j \,\}$$

(all strings of pairwise distinct names)





$$L_3 = \{\, a_1 a_2 \ldots a_{2n} \in \Sigma^* \mid n \geq 0,\ \forall i < 2n.\ a_i \neq a_{i+1}$$
$$\forall i \leq n,\ j < 2i.\ a_j \neq a_{2i} \,\}$$

Application: Nominal AGS

The modelling power of FRAs can be used to model resourceful programs via game semantics

Program → game model → FRA

effectively:

two programs are equivalent ⇔ their FRAs are language equivalent / bisimilar

Application: Nominal AGS

The modelling power of FRAs can be used to model resourceful programs via game semantics

Program → game model → FRA

effectively:

two programs are equivalent ⇔ their FRAs are language equivalent / bisimilar

Nominal Algorithmic Game Semantics:

- decision procedures for ML fragments [Murawski & Tz '11, '12]
- same for Interface Middleweight Java [Murawski, Ramsay & Tz (sub)]

Investigations in RAs

Investigations in RAs

Freshness: from one to many histories

- History RA (~ Petri nets with transfer arcs) [Grigore & Tz '13]

Bisimilarity for RA (complexity)

- Depends on register mode (NP? → PSPACE → EXPTIME)
  - approach uses inverse semigroup theory [Murawski, Ramsay & Tz (sub)]

Investigations in RAs

Freshness: from one to many histories

- History RA (~ Petri nets with transfer arcs) [Grigore & Tz '13]

Context-freeness: Pushdown RA [Cheng & Kaminski '98; Segoufin '06]

- Reachability is EXPTIME-complete
- Global reachability via “saturation” [Murawski, Ramsay & Tz '14]

Bisimilarity for RA (complexity)

- Depends on register mode (NP? → PSPACE → EXPTIME)
  - approach uses inverse semigroup theory [Murawski, Ramsay & Tz (sub)]

[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

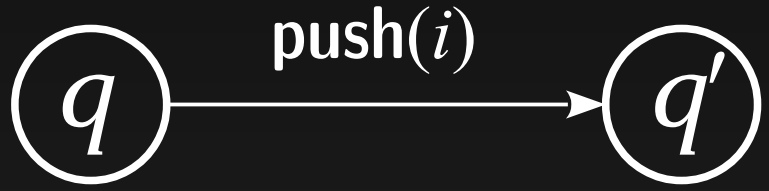
pushdown stack

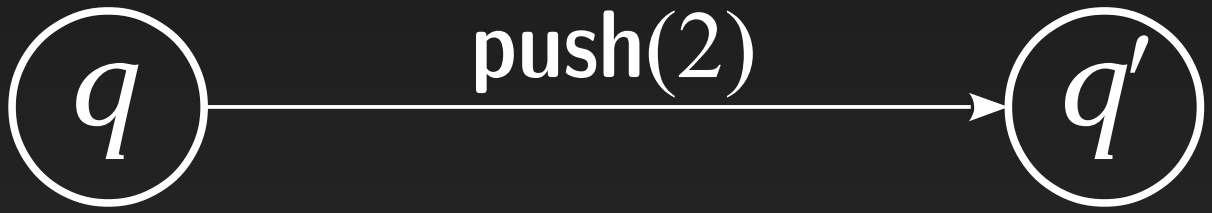
registers & stack store names

Label λ of the form:

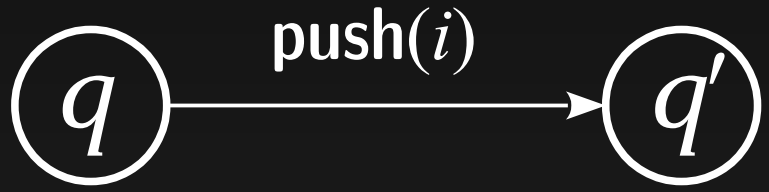
- **read**(i), $i \in \{1,\ldots,R\}$
- **fresh**(i), $i \in \{1,\ldots,R\}$
- **push**(i), $i \in \{1,\ldots,R\}$
- **pop**(i), $i \in \{1,\ldots,R\}$
- **pop-fresh**

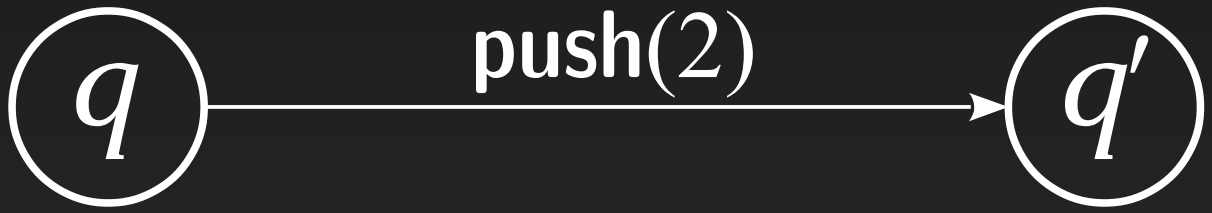
Transitions:

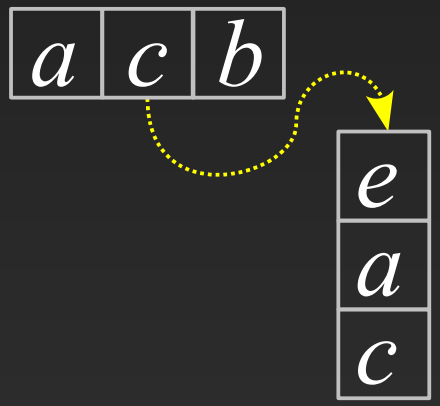




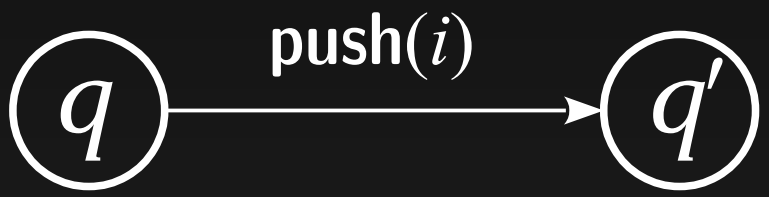
Transitions:

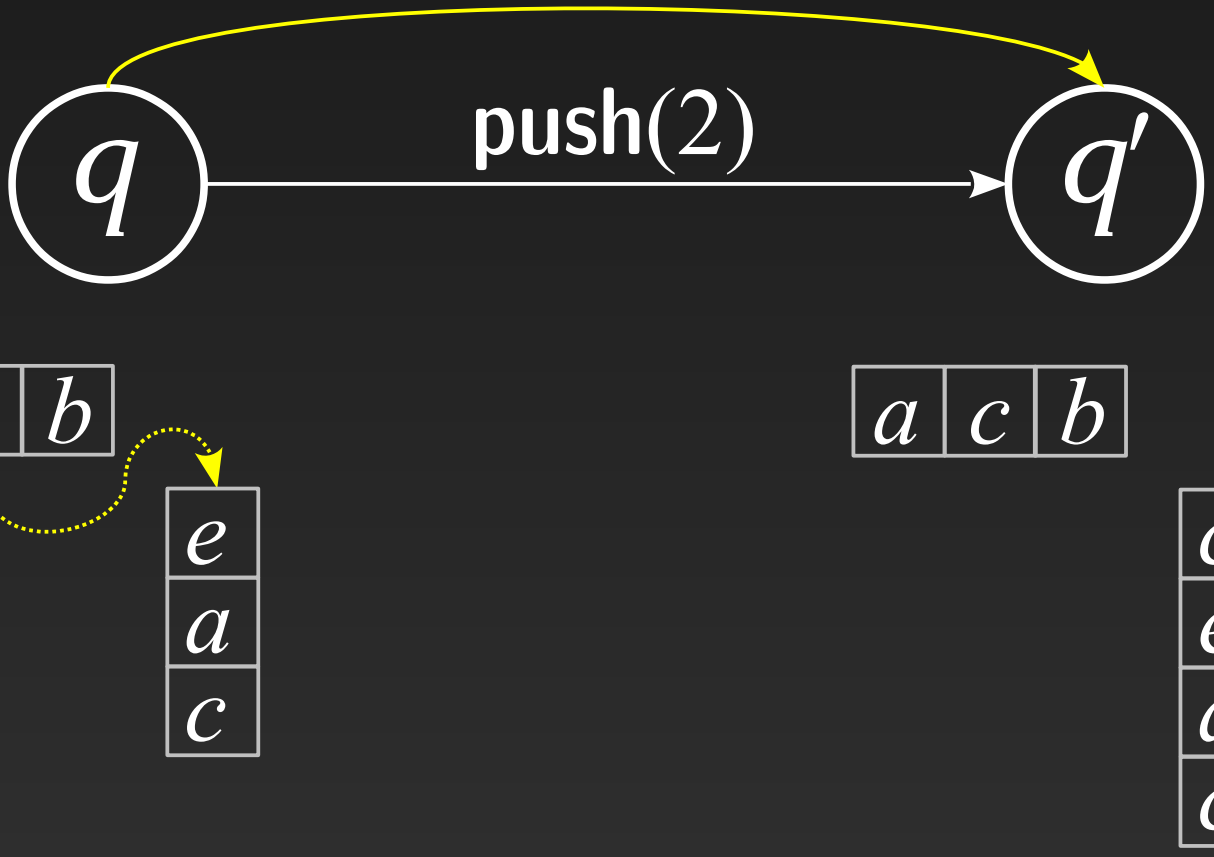




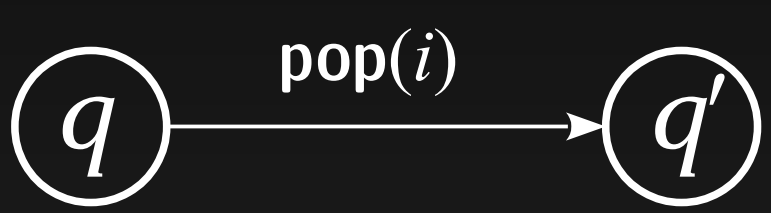


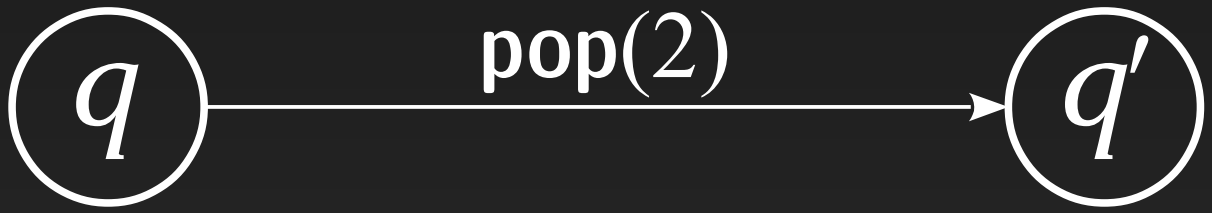
Transitions:



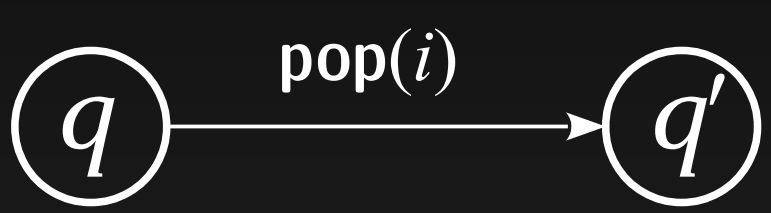


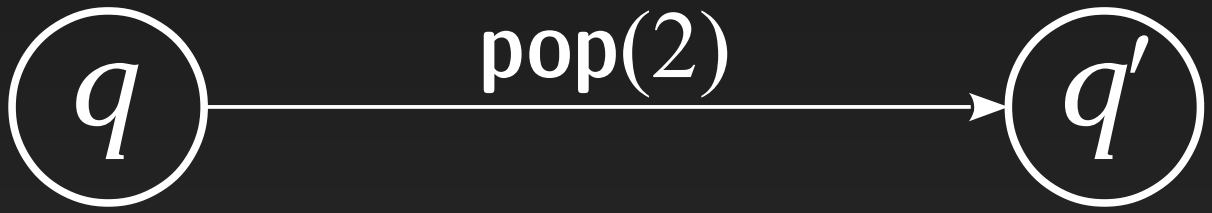
Transitions:

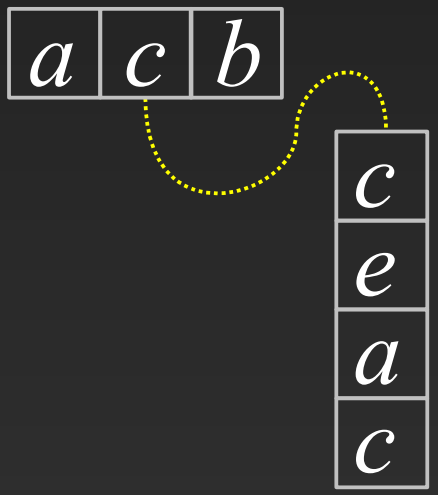




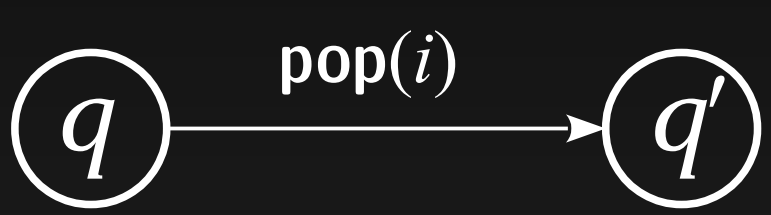
Transitions:

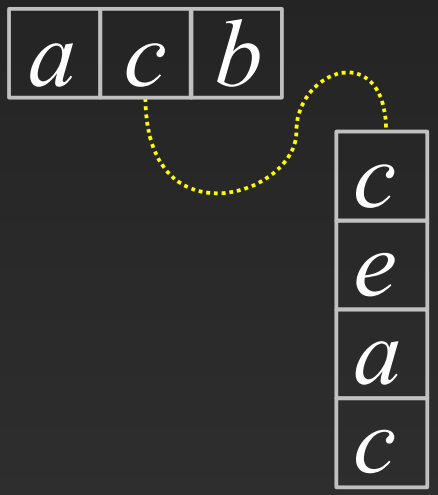


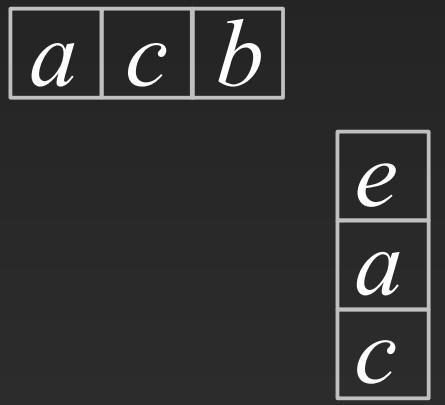




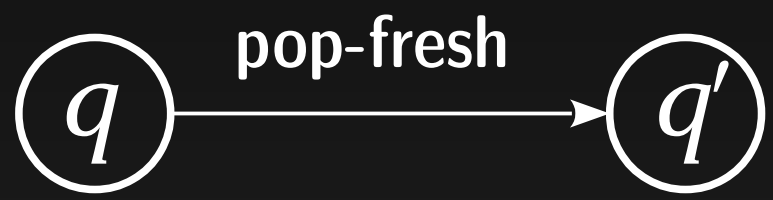
Transitions:

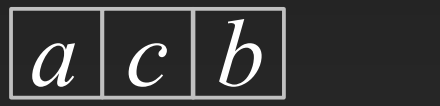




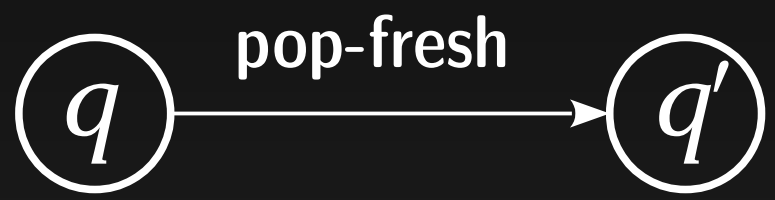


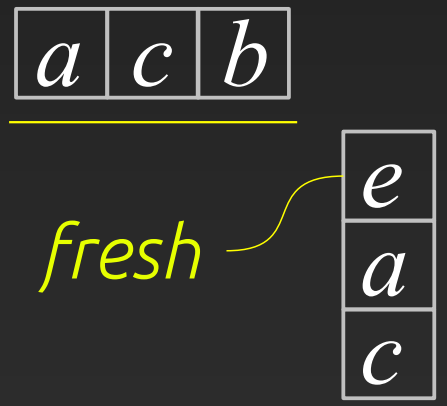
Transitions:



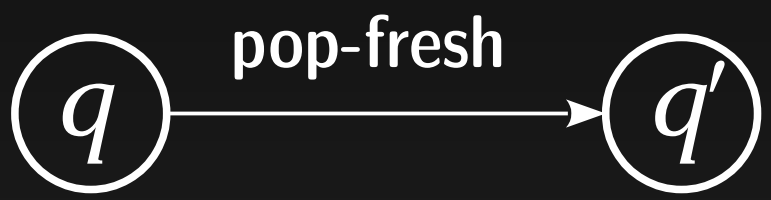


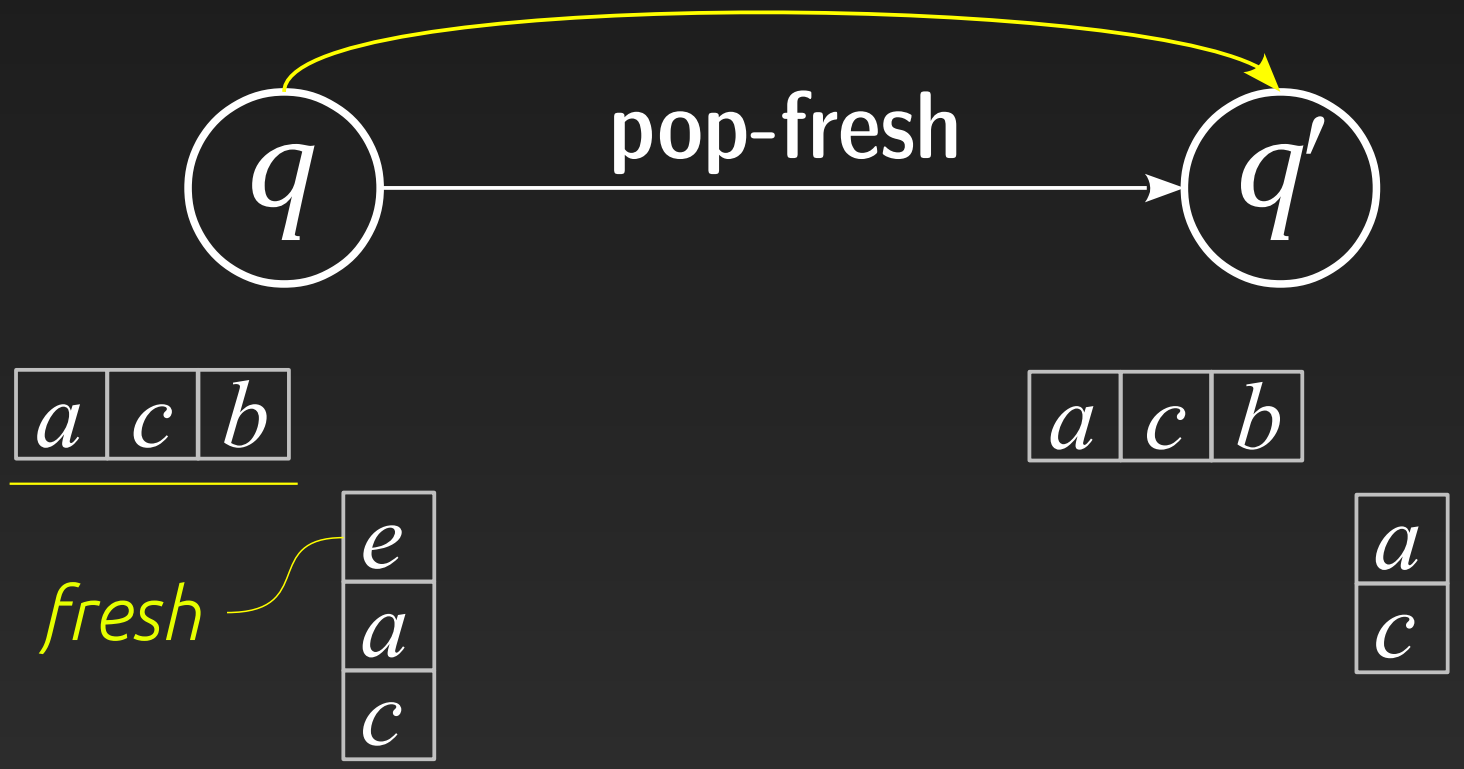
Transitions:





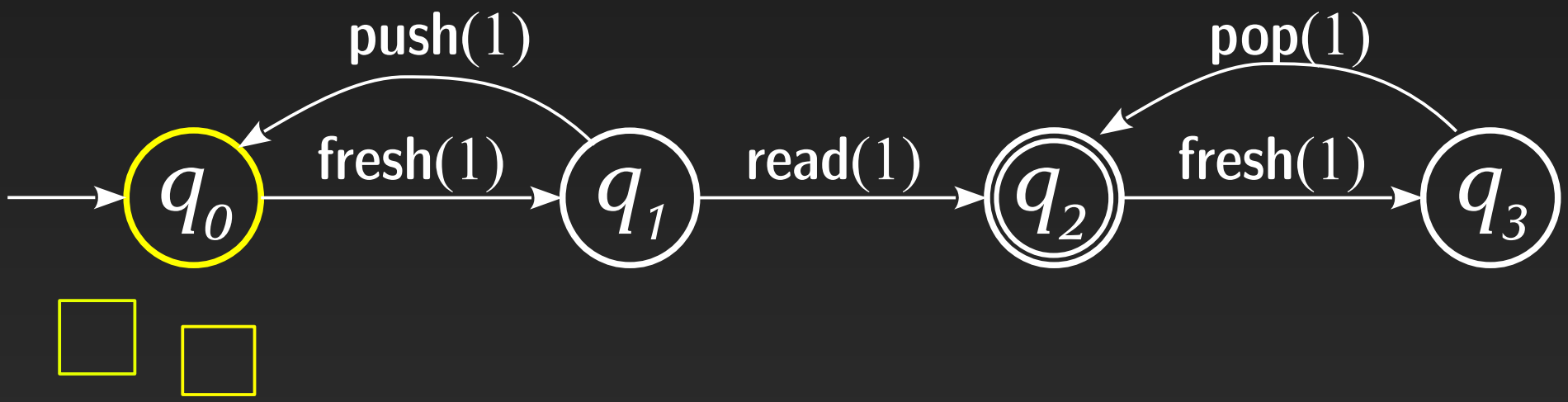
Transitions:





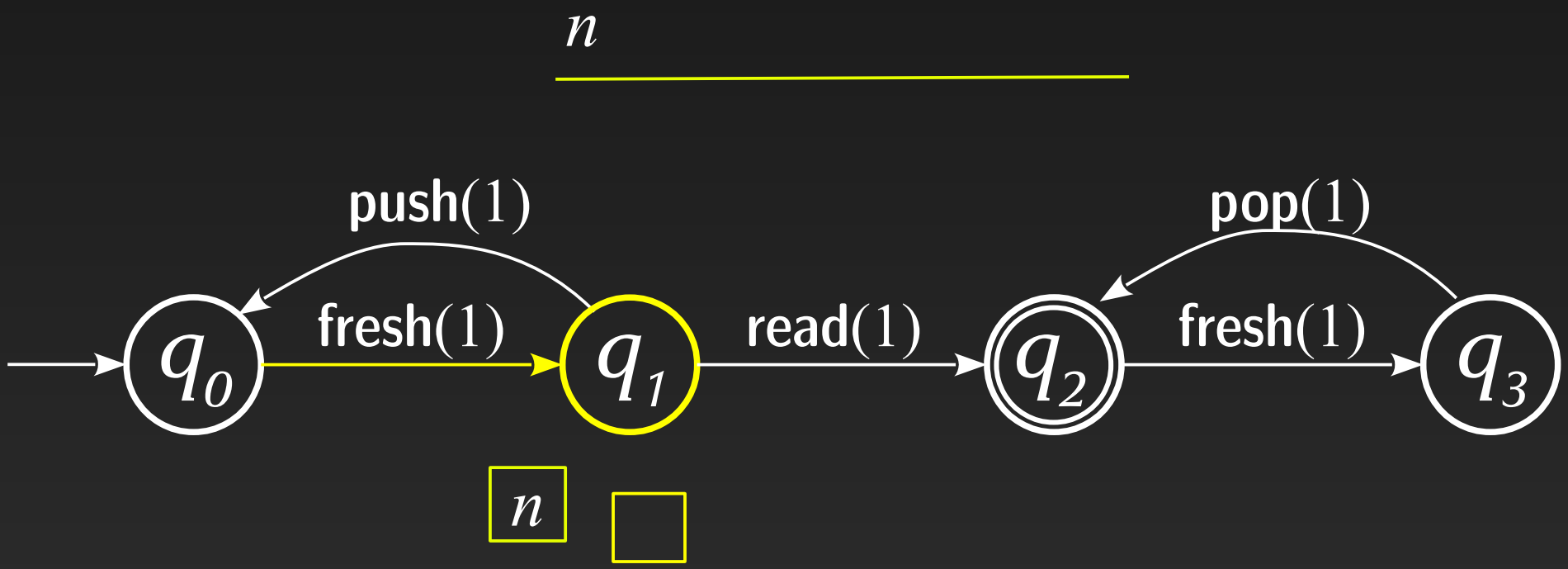
Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$



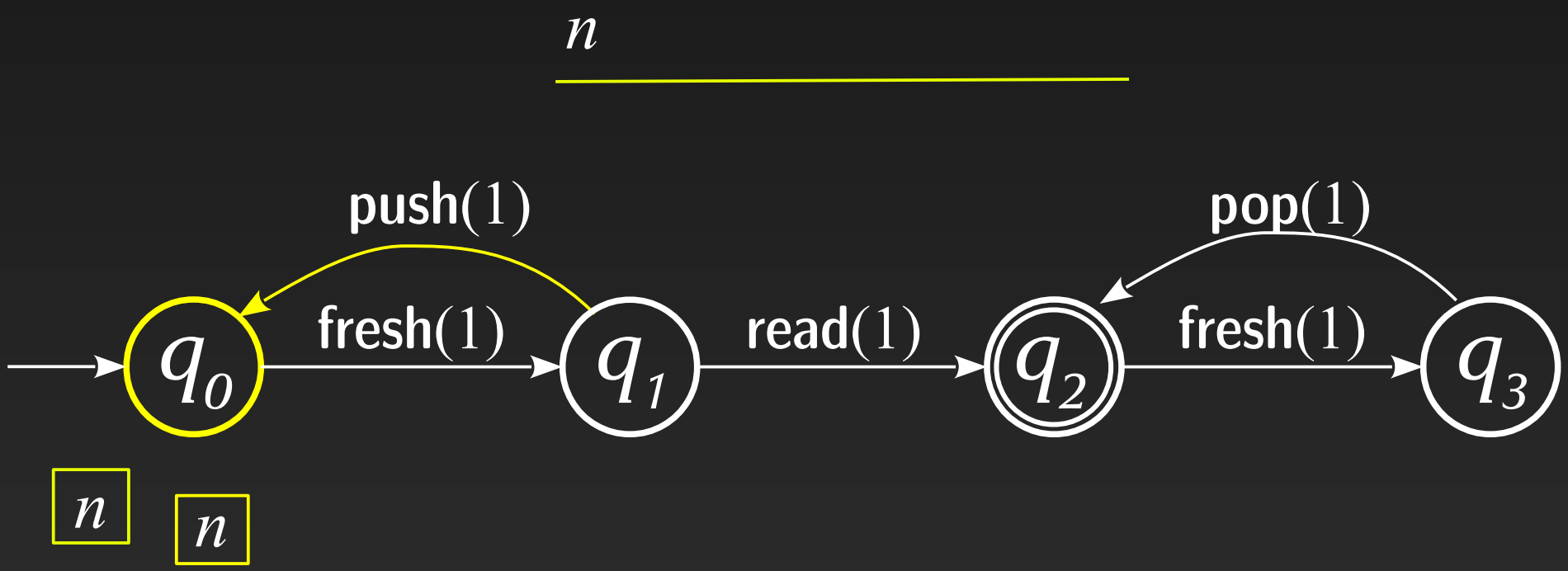
Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$



Example

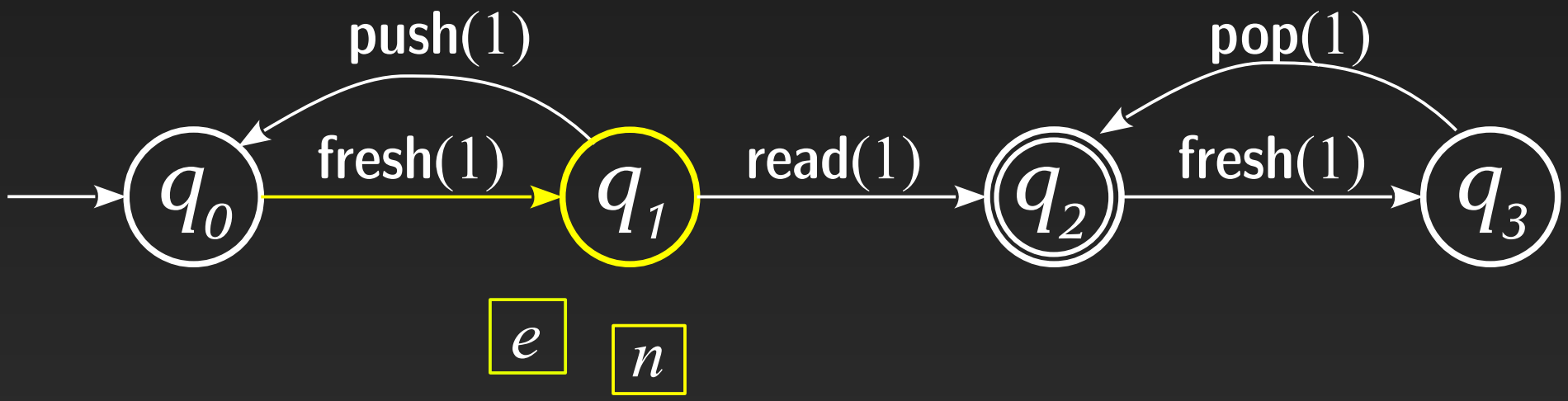
$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

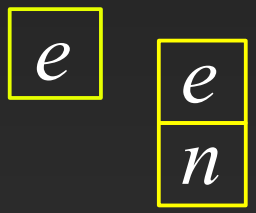
$n\,e$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

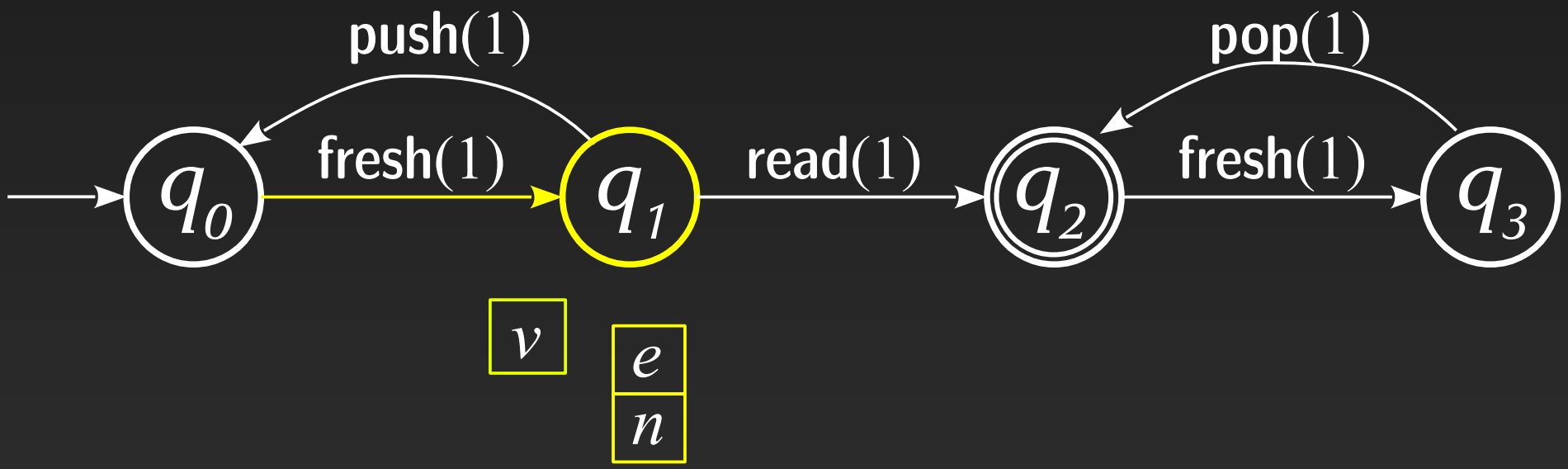
$n\,e$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

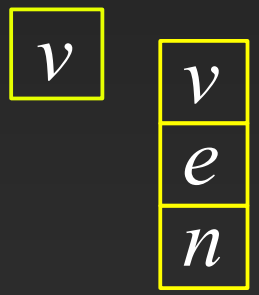
$n\ e\ v$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

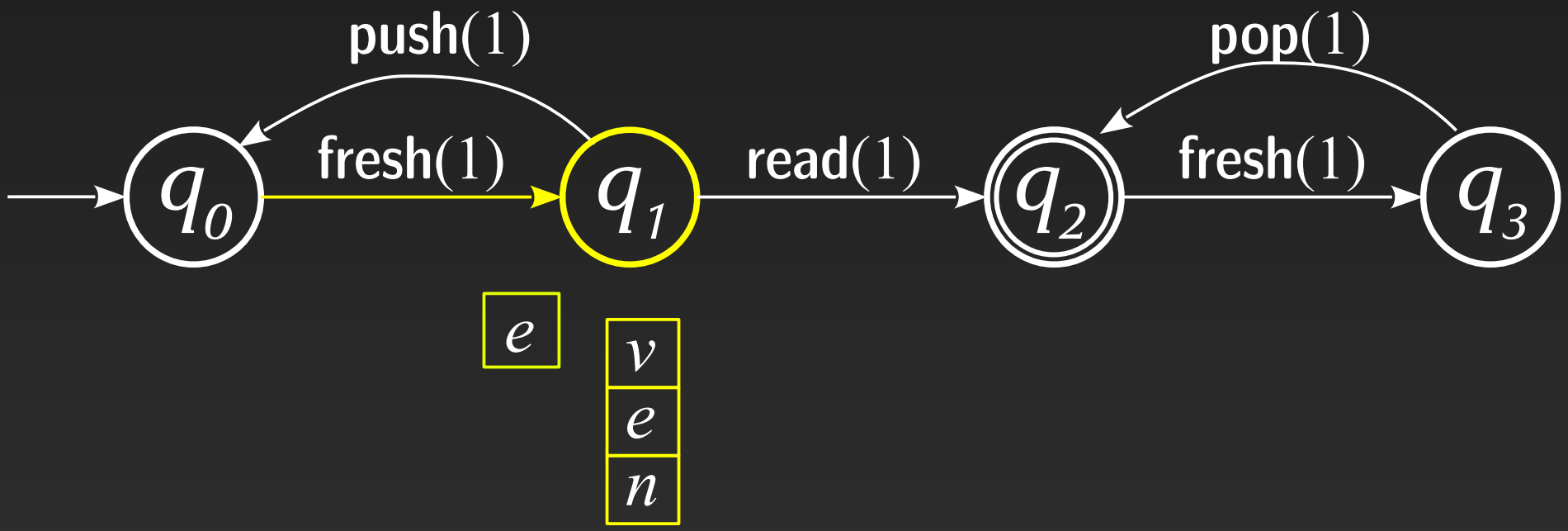
nev



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

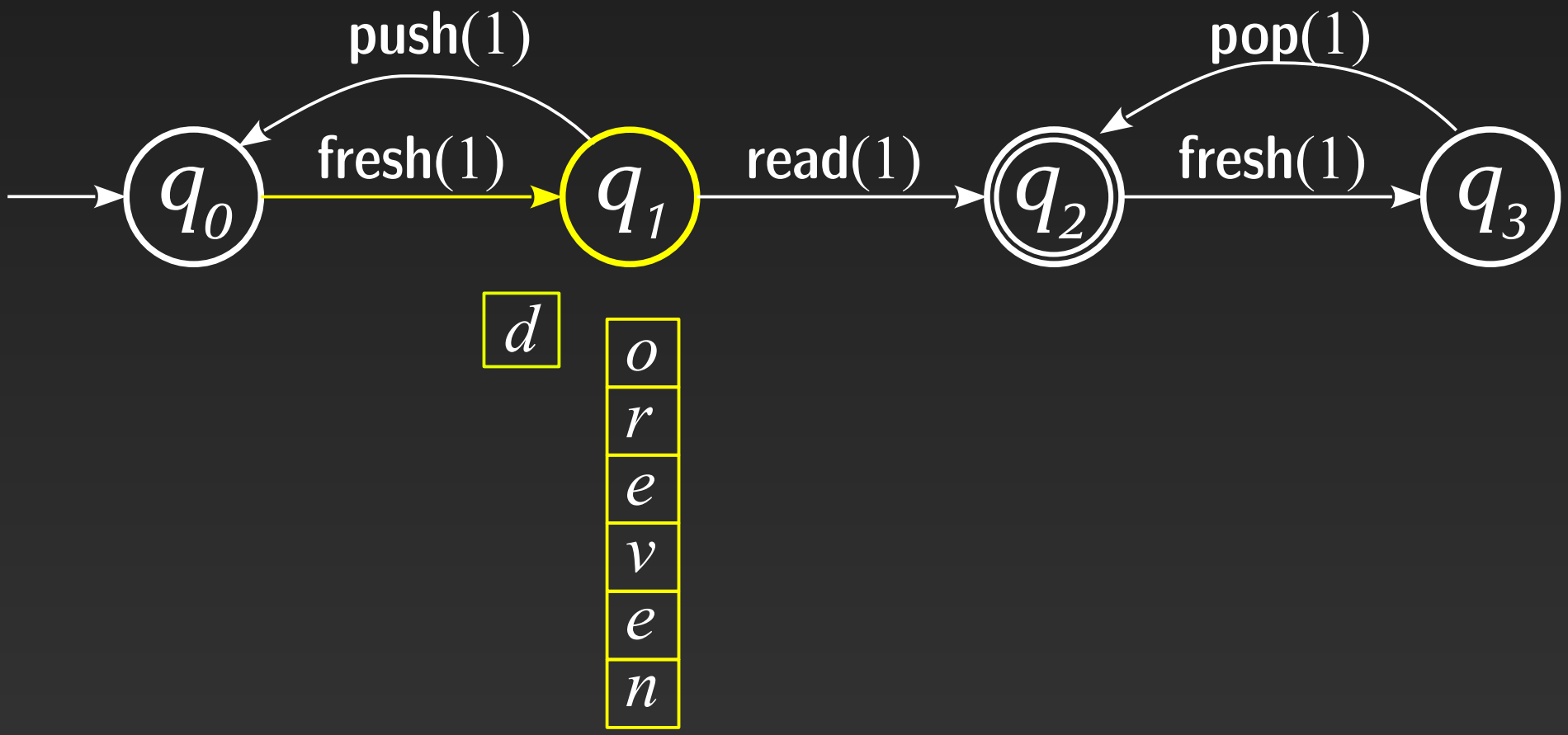
neve



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

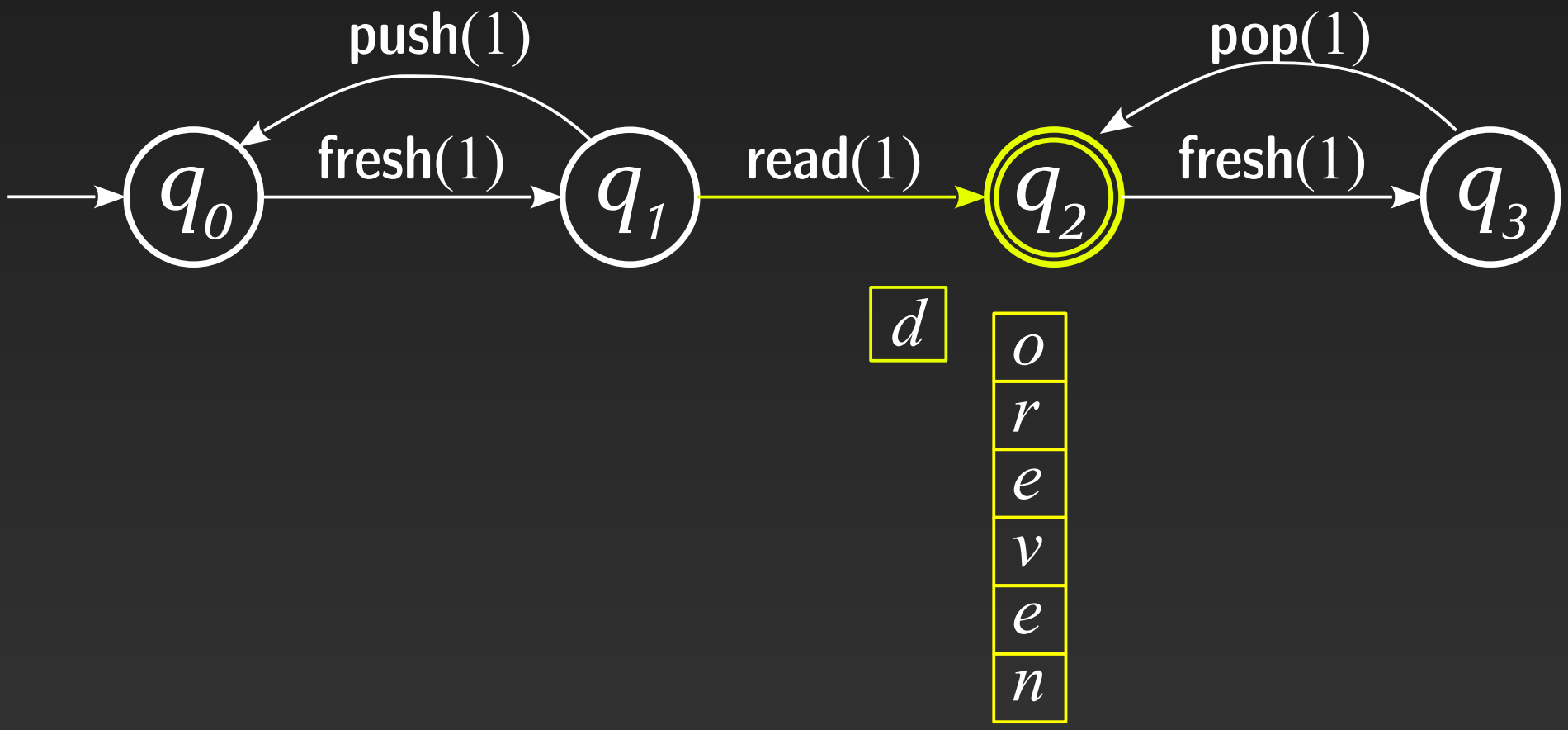
$neverod$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

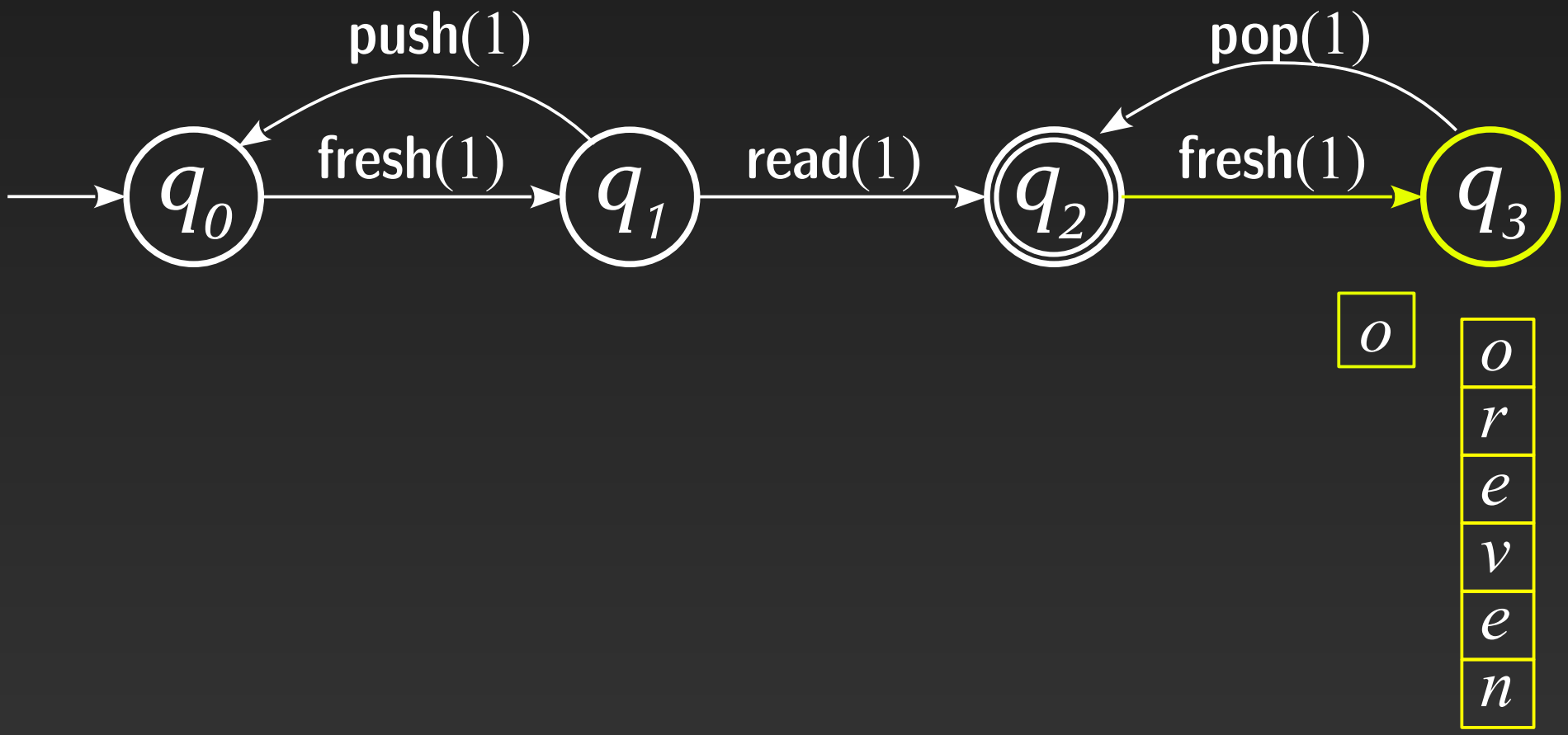
$neverodd$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

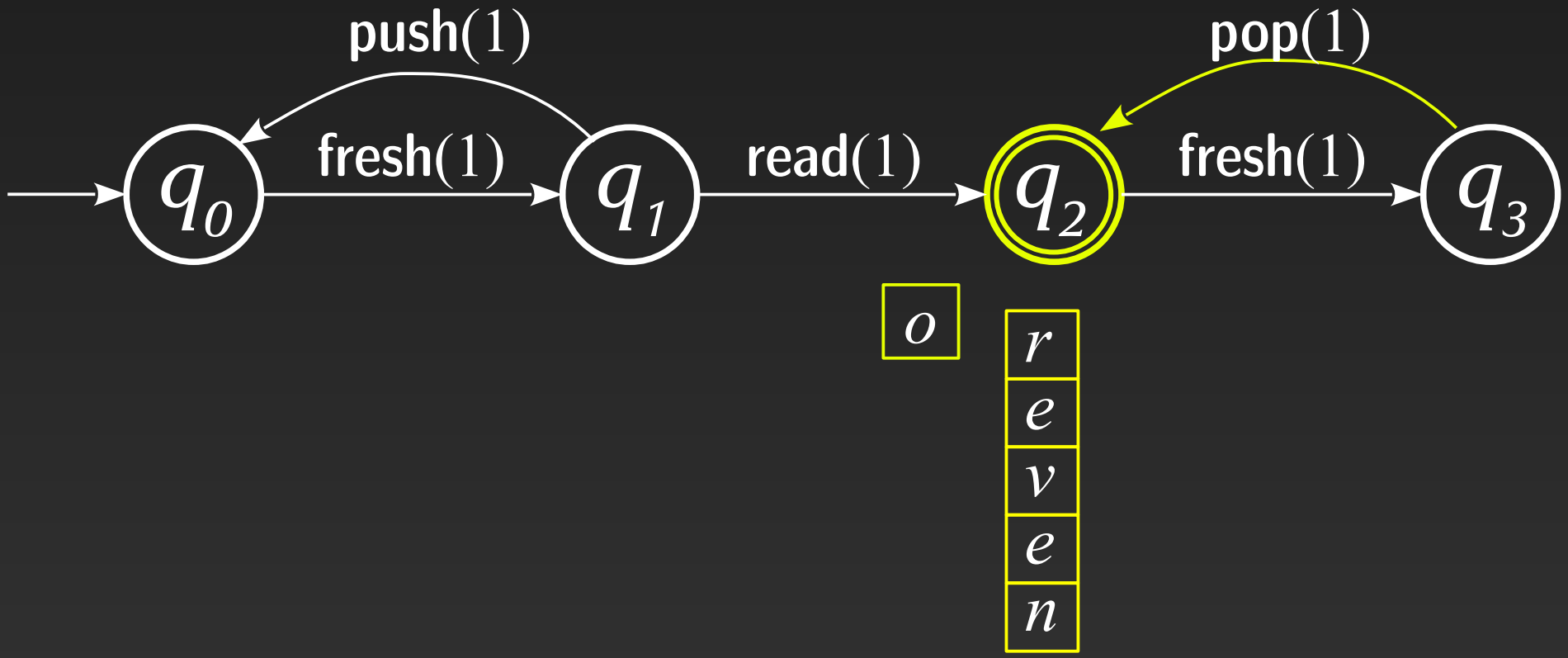
neveroddo



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

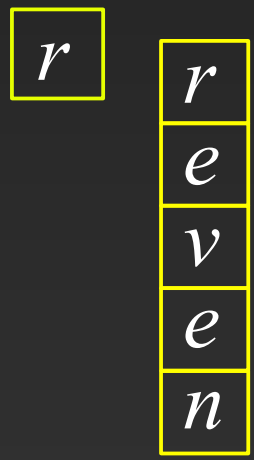
neveroddo



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

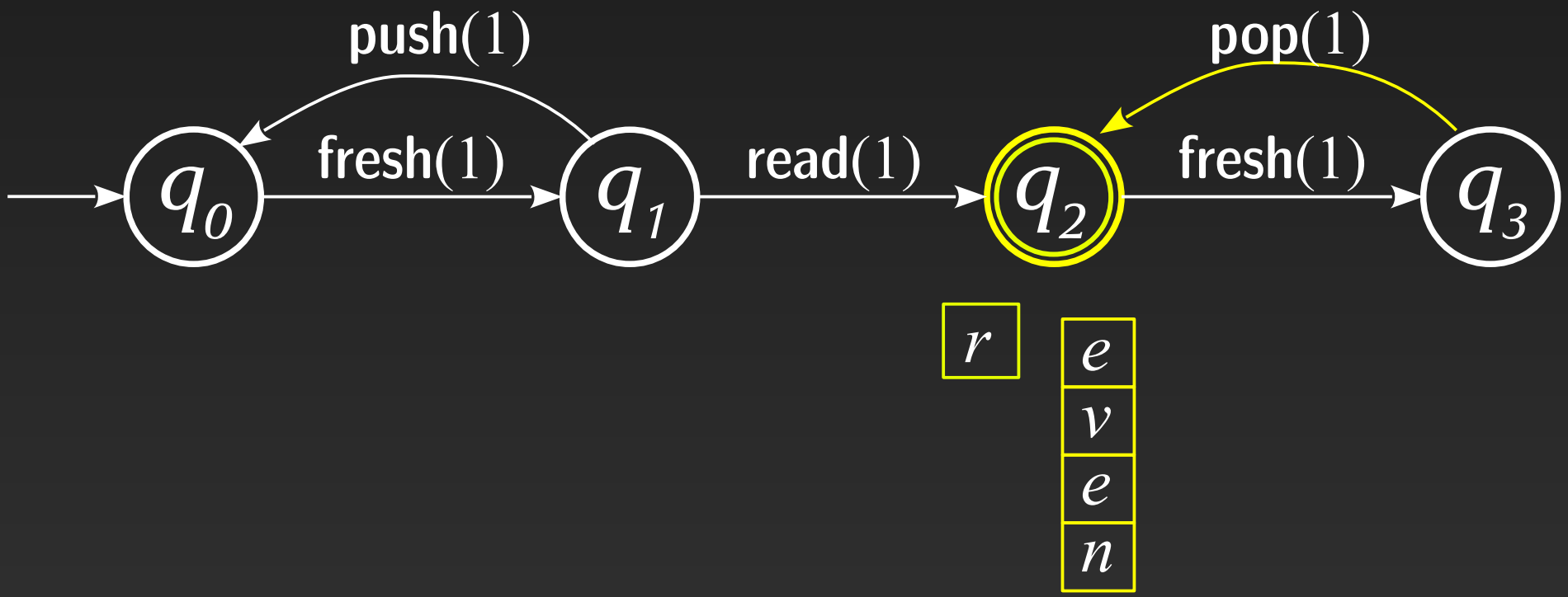
$neveroddor$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

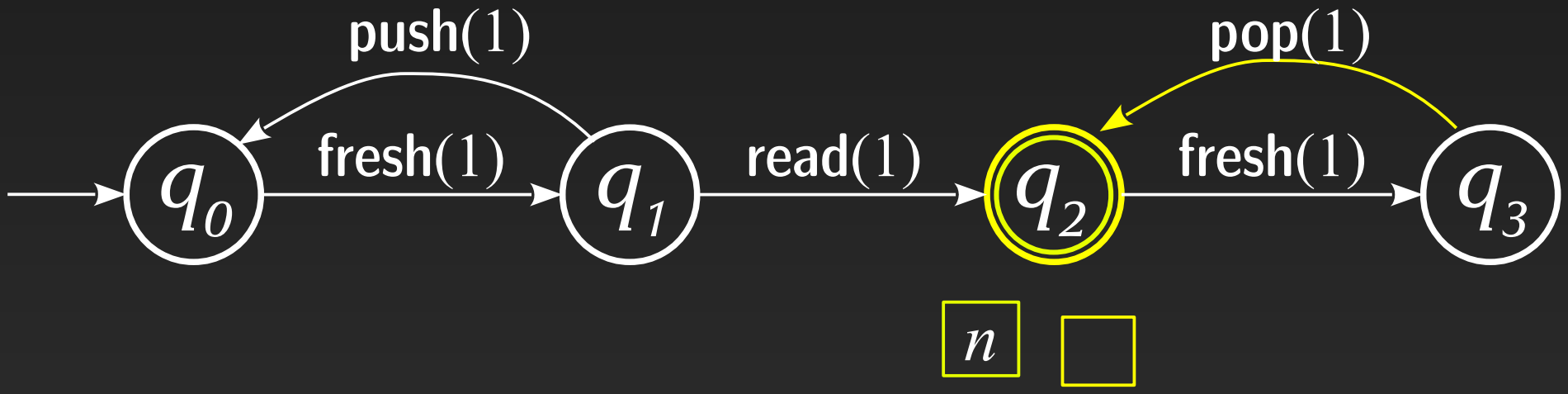
$neveroddor$



Example

$$L_4 = \{\, a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

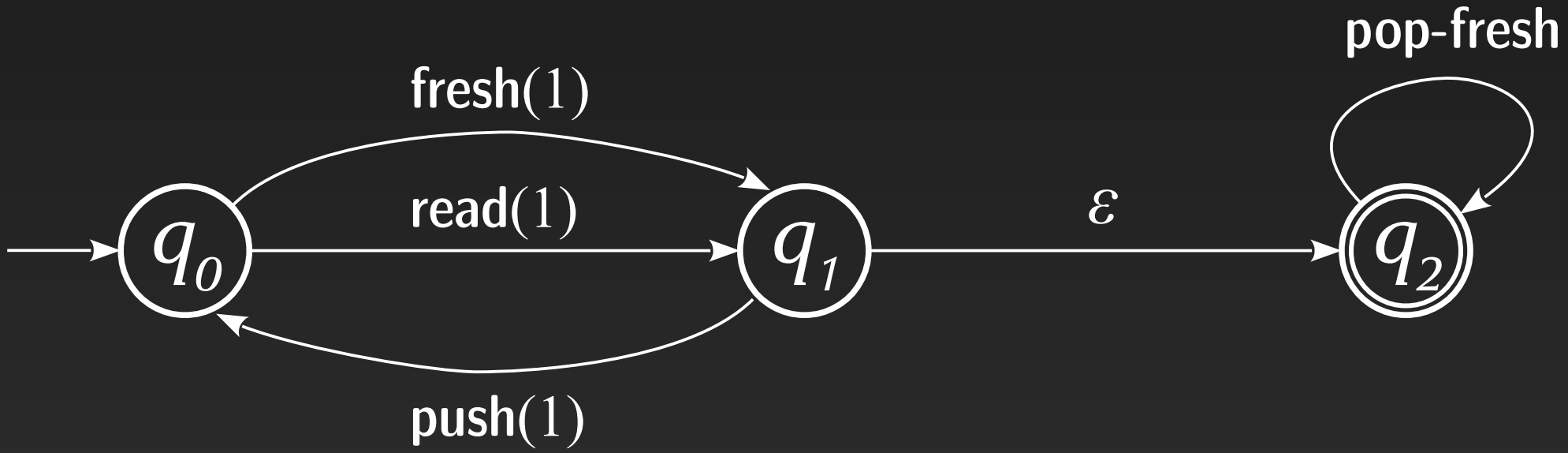
$neveroddoreven$



Another example

$$L_5 = \{\, a_1 a_2 \ldots a_n b \in \Sigma^* \mid n \geq 0,\ \forall i \leq n.\ a_i \neq b \,\}$$

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



Limited distinguish-ability

$$L_2 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j.\ a_i \neq a_j \,\}$$

(all strings of distinct names)

Limited distinguish-ability

$$L_2 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i \neq j.\ a_i \neq a_j \,\}$$

(all strings of distinct names)

Lemma: Let A be a PDRA 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 of same length involving at most $3R$ names.

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

Limited distinguish-ability

$$L_2 = \{ a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j.\ a_i \neq a_j \}$$

(all strings of distinct names)

Lemma: Let A be a PDRA 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 of same length involving at most $3R$ names.

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

Reachability / non-emptiness

R-PRDA Reach: *Given a PDRA A with R registers and a state q, is there a run of A to q?*

Theorem: R-PRDA 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:

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:

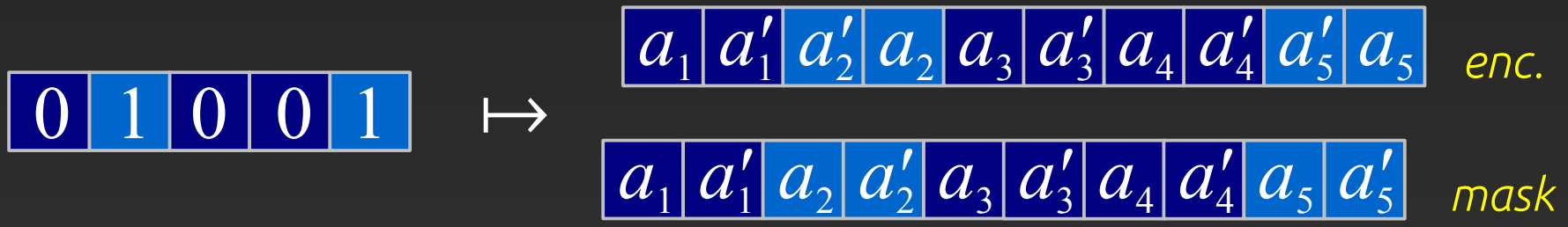
  - now, instead, we use *mask encodings* + the stack:

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:

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



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

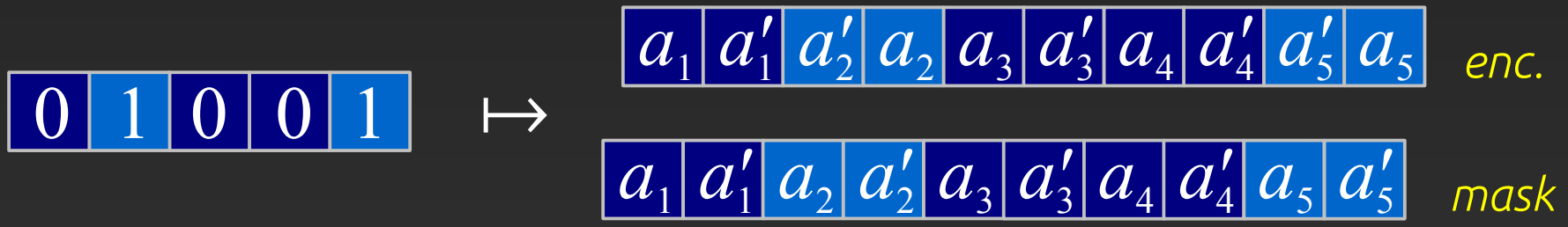
EXPTIME-hardness

without the stack, repetitions give a huge complexity gap!

For hardness, the argument is harder...

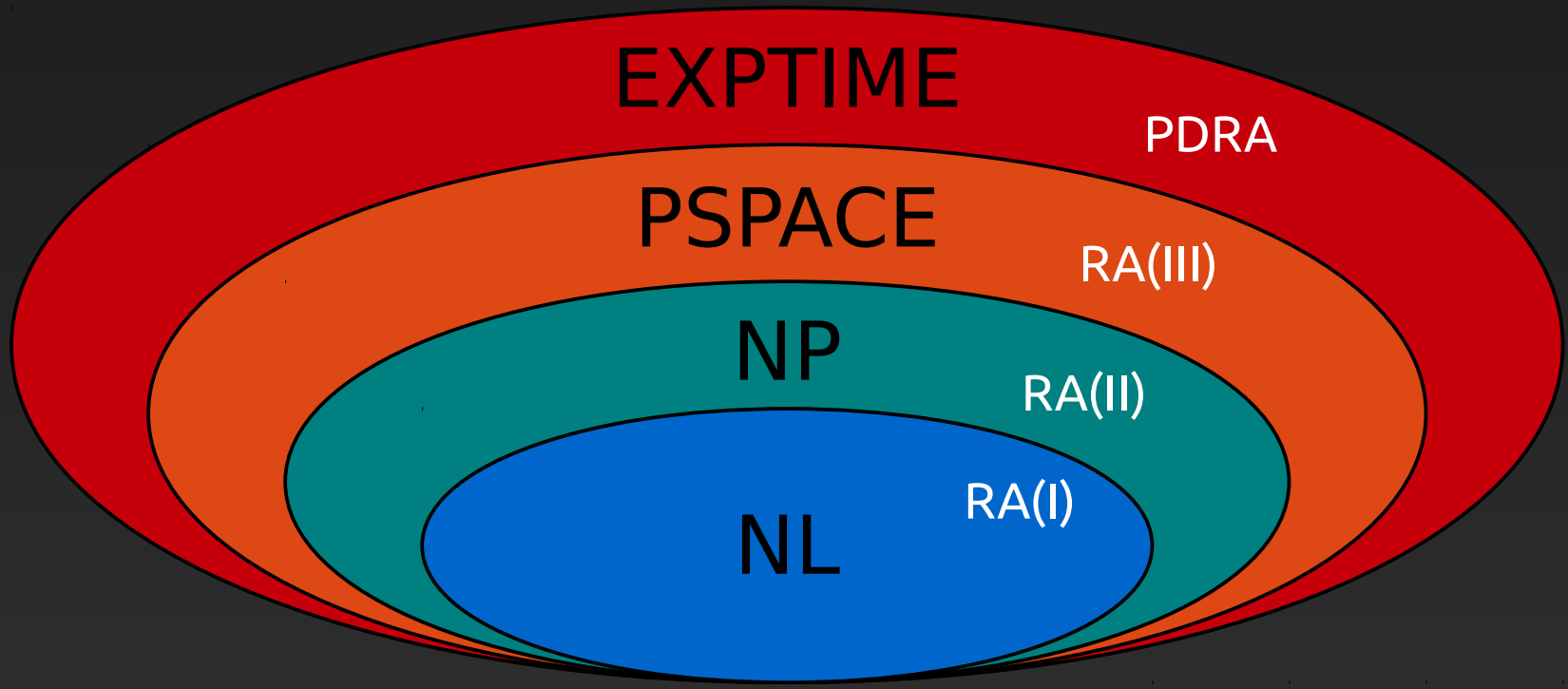
- We reduce from PSPACE Turing machines with stack
- crux of the reduction is the simulation of the tape
  - if we allowed name repetitions in registers then easy:

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



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

Reachability/non-emptiness picture



Concluding

Automata over infinite alphabets:

- natural for some tasks in verification
- new landscape of algorithms and results

Things to do:

- algorithm implementations (an AIA toolkit!)
- more theory, e.g. automata learning
- more applications

Concluding

thanks

Automata over infinite alphabets:

- natural for some tasks in verification
- new landscape of algorithms and results

Things to do:

- algorithm implementations (an AIA toolkit!)
- more theory, e.g. automata learning
- more applications