Automata over infinite alphabets: Investigations in Fresh-Register Automata

Nikos Tzevelekos
Queen Mary University of London

Andrzej Murawski & Steven Ramsay, University of Warwick
Radu Grigore, University of Oxford

University of Birmingham, Feb 2015

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();
}
```

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();
}
```

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 and freshness oracles

We look at their expressiveness and algorithmic properties with particular focus on bisimilarity and 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

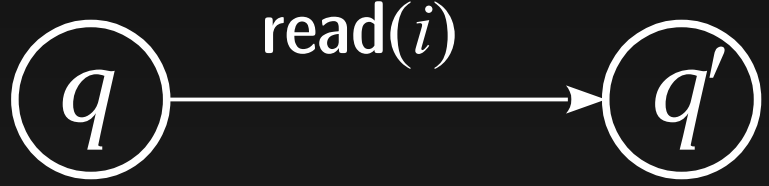
finitely many (say R) registers

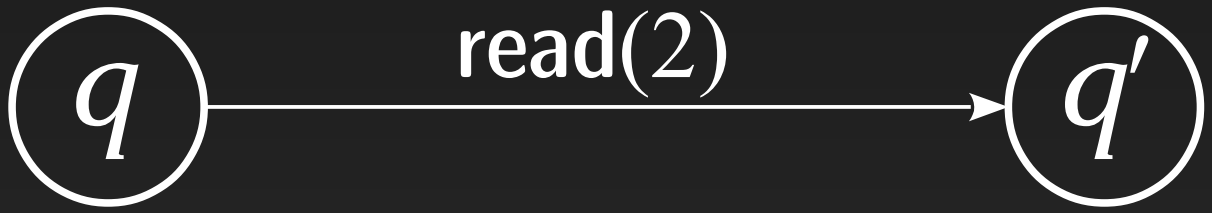
registers store names

Label λ of the form:

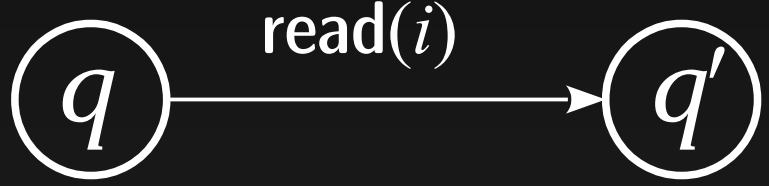
- **read**(i), $i \in \{1,\ldots,R\}$
- **fresh**(i), $i \in \{1,\ldots,R\}$

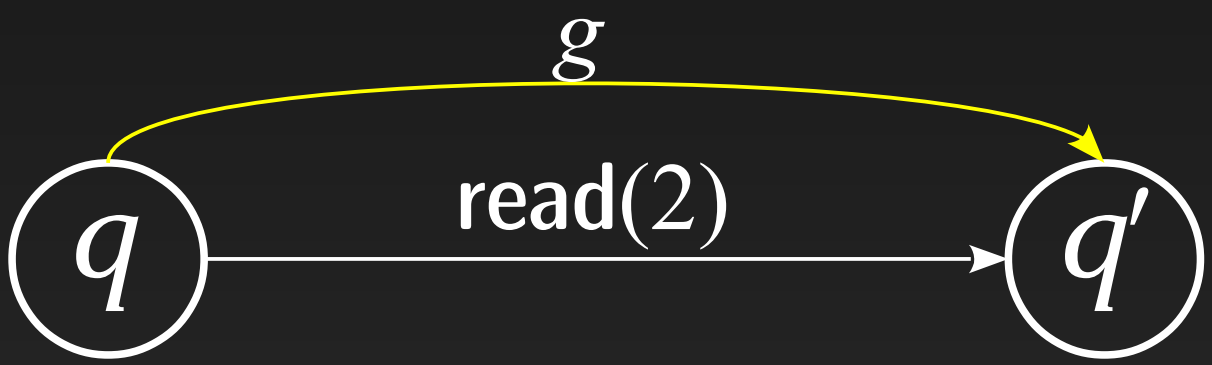
Transitions:



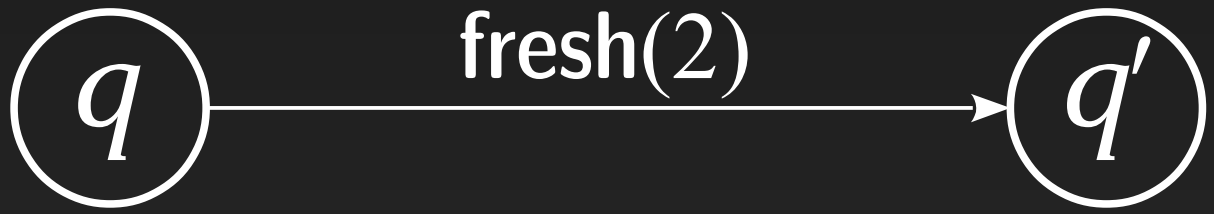


Transitions:



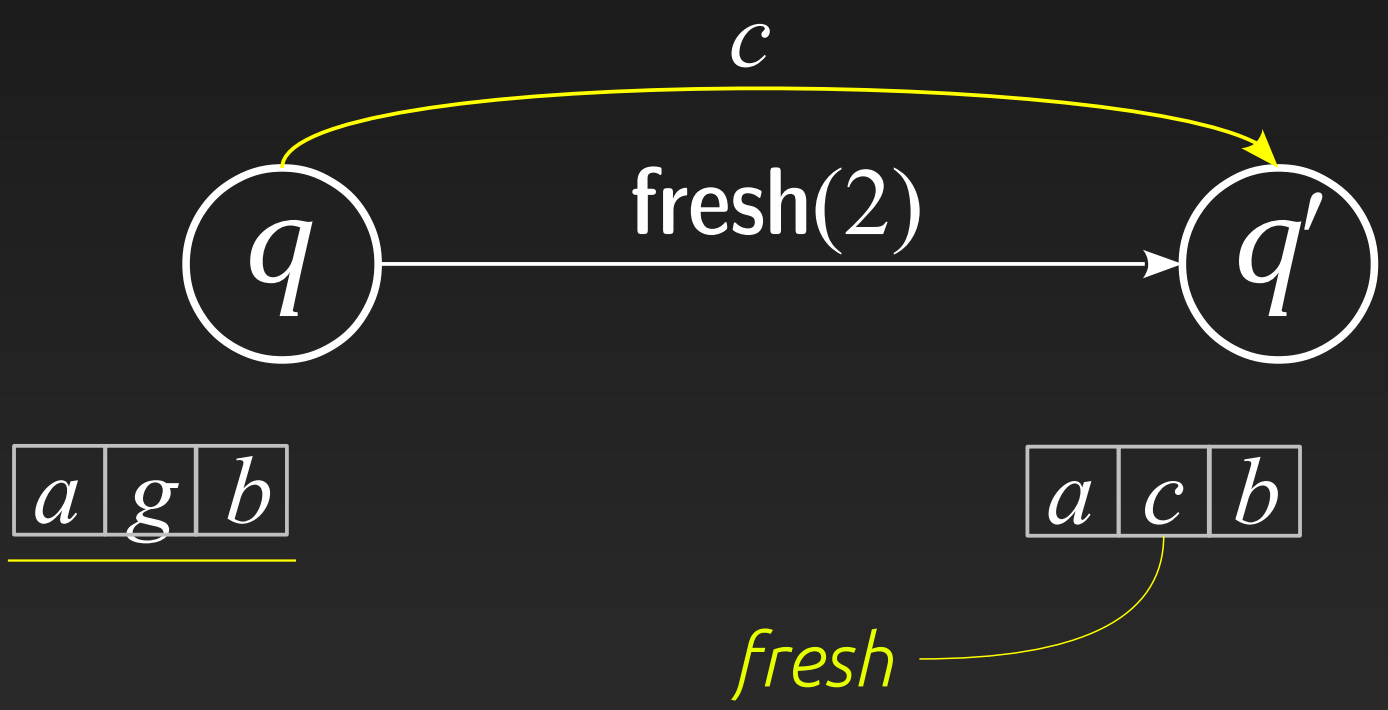


Transitions:



a	g	b

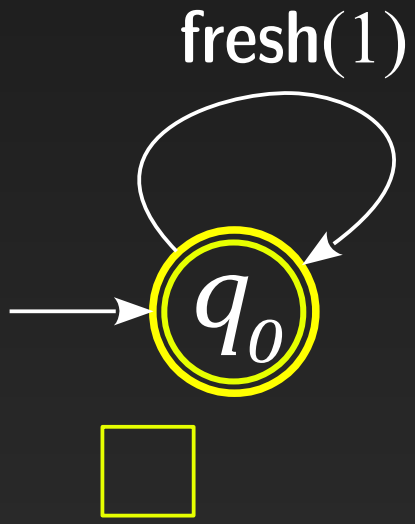
Transitions:



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

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

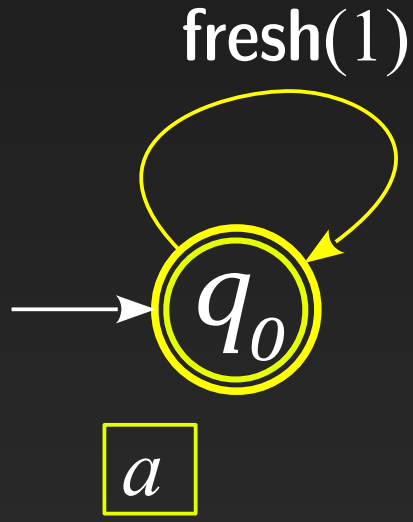
(all strings where each name is distinct from its predecessor)

$a\,b\,c$

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)

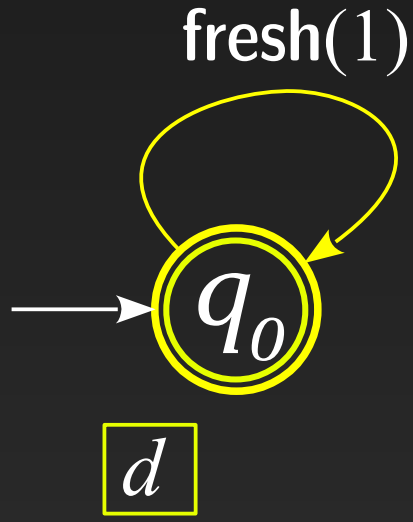


$a\,b\,c\,a$

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)

$$a\,b\,c\,a\,d\,e\,b\,a\,g\,c\,a\,b\ a\,n\,d\ w\,e\ l\,o\,v\,e\ c\,a\,k\,e$$

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

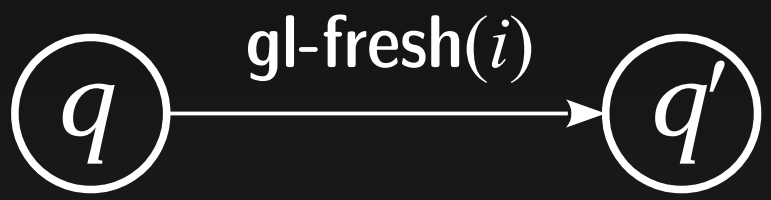
registers store names

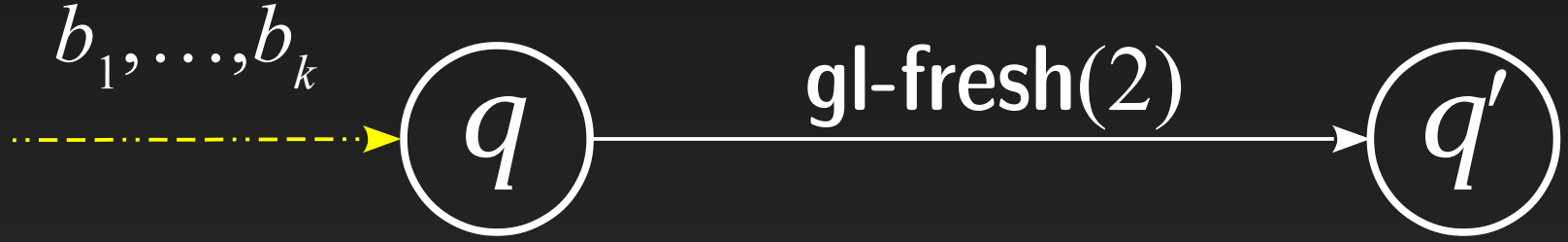
Label λ of the form:

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

global freshness oracle

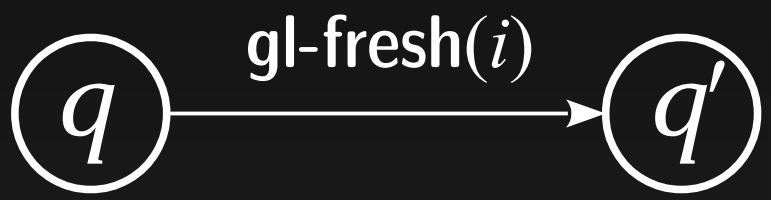
Transitions:

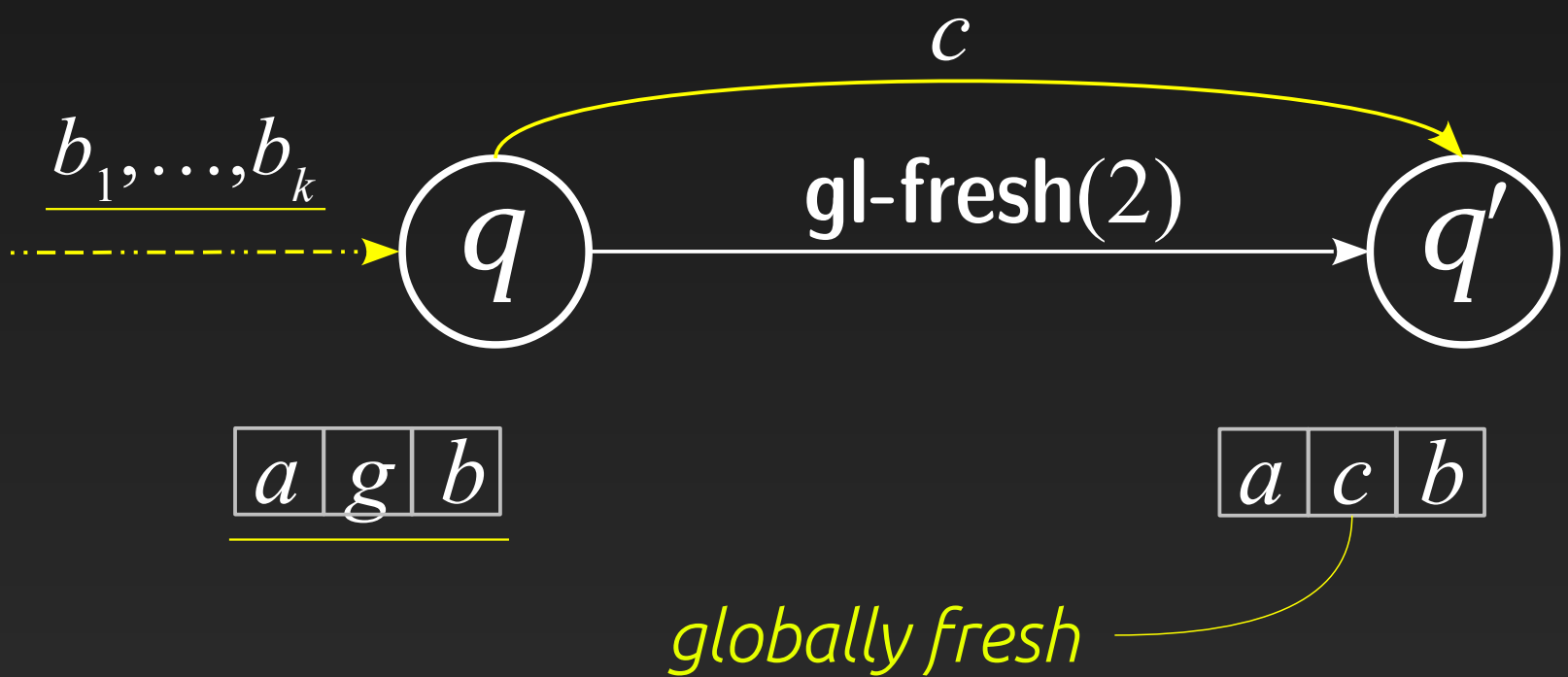




a	g	b

Transitions:

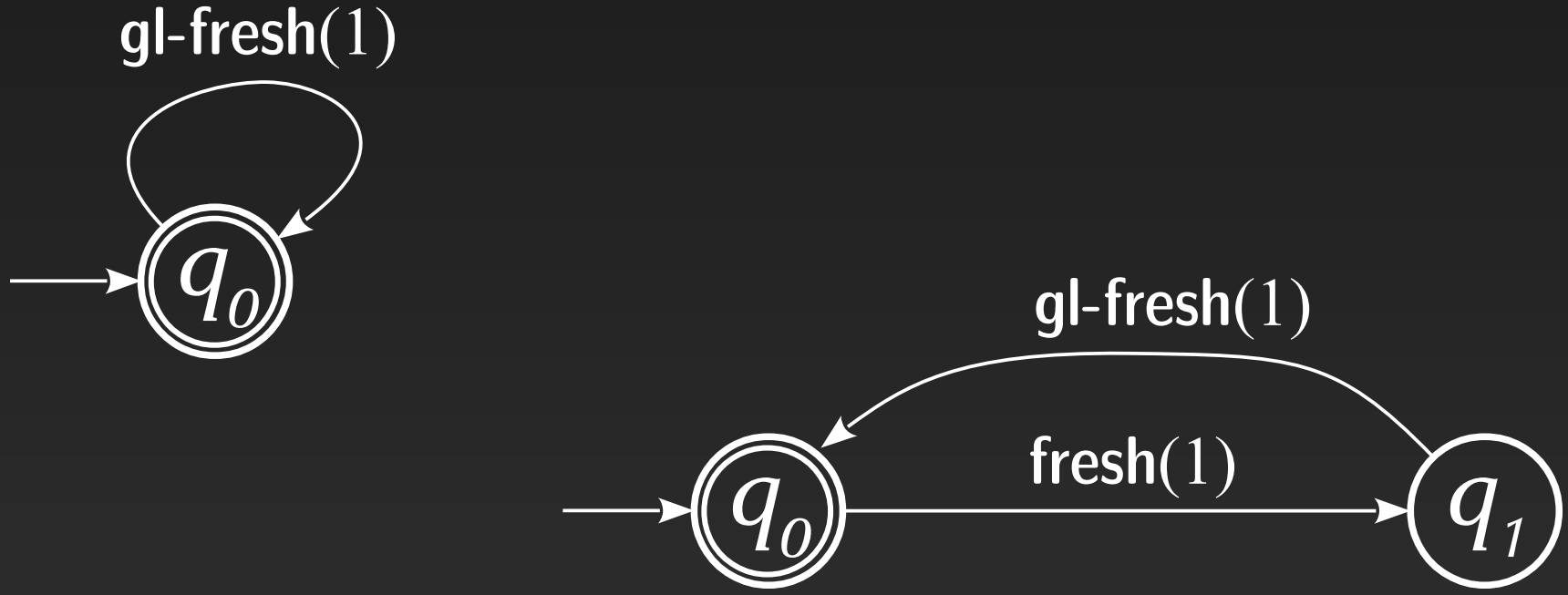




Examples

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

(all strings of pairwise distinct names)



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

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]

Investigations in FRAs

Investigations in FRAs

Freshness: from one to many histories

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

Bisimilarity for FRA (complexity)

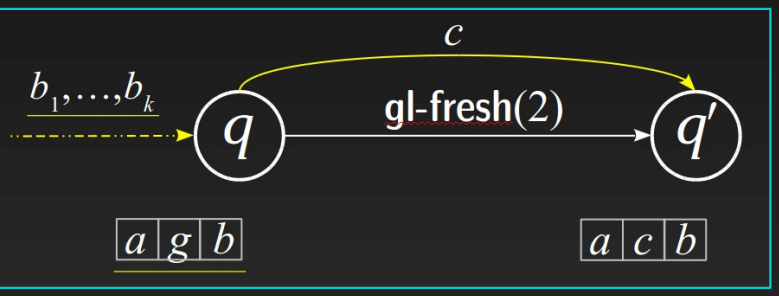
- Depends on register mode (NP? → PSPACE → EXPTIME)
  - approach uses permutation group theory [Murawski, Ramsay & Tz]

Context-freeness: Pushdown FRA [Cheng & Kaminski '98; Segoufin '06] [Murawski & Tz '12]

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

Bisimilarity

Running FRAs gives rise to configuration graphs:



configuration

$$(q,\rho,H) \xrightarrow{a} (q',\rho',H')$$

state

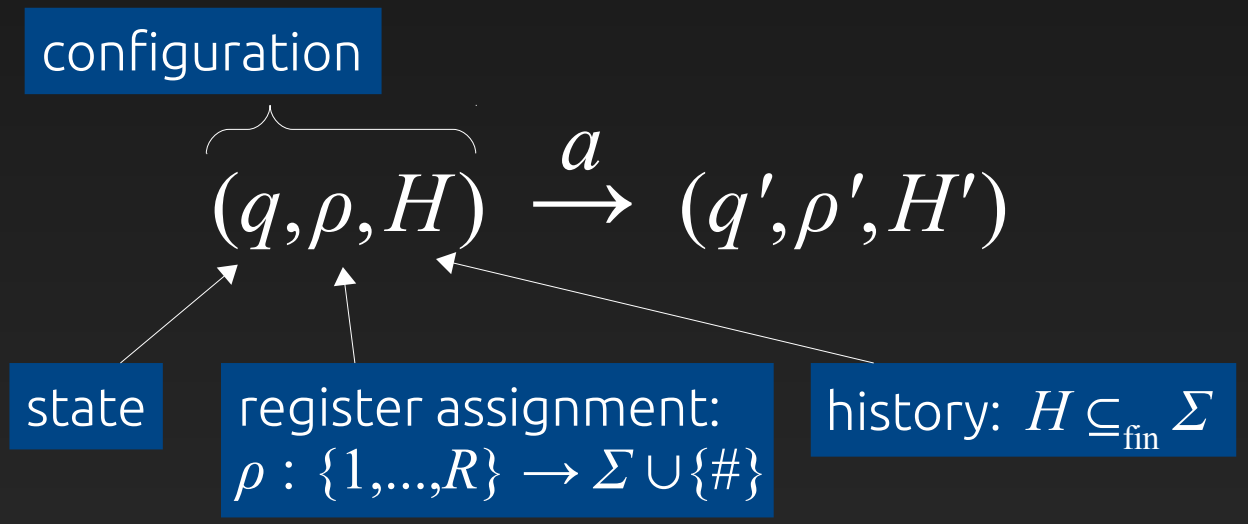
register assignment:
$\rho : \{1,\ldots,R\} \to \Sigma \cup \{\#\}$

history: $H \subseteq_{\text{fin}} \Sigma$

this is how we define
e.g. language acceptance

Bisimilarity

Running FRAs gives rise to configuration graphs:



Bisimilarity (~):

- standard notion of equivalence (e.g. in concurrency)
- specifies that two configurations are locally "indistinguishable" (⇒ trace equivalence)
- was shown decidable in [Tz '11] but no complexity

Bisimulation Game

Two players, an Attacker (A) and a Defender (D), play a game on a configuration graph, starting from configurations κ_1 and κ_2

- A wants to show them not bisimilar, D the opposite
- A picks a configuration, say κ_1, and an edge out of it, say with label a
- D must then pick an a-edge out of κ_2 – or he loses!
- now we are at κ_1' and κ_2' – and the game continues

Bisimulation Game

Two players, an Attacker (A) and a Defender (D), play a game on a configuration graph, starting from configurations κ_1 and κ_2

- A wants to show them not bisimilar, D the opposite
- A picks a configuration, say κ_1, and an edge out of it, say with label a
- D must then pick an a-edge out of κ_2 – or he loses!
- now we are at κ_1' and κ_2' – and the game continues

κ_1 and κ_2 are called *bisimilar*, write $\kappa_1 \sim \kappa_2$, if D has a strategy for not losing the game

A small detail: register modes

So far we assumed: registers initially empty, not possible to erase them, or have name duplicates.
We now generalise:

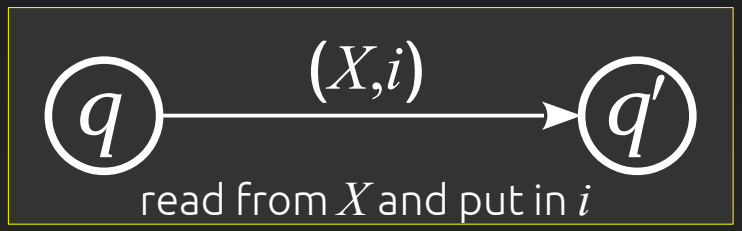
- *Duplicates* :
  - (*S*) single
  - (*M*) multiple
- *Erasure* :
  - (*F*) full
  - ($\#_0$) initially empty
  - (#) eraseable

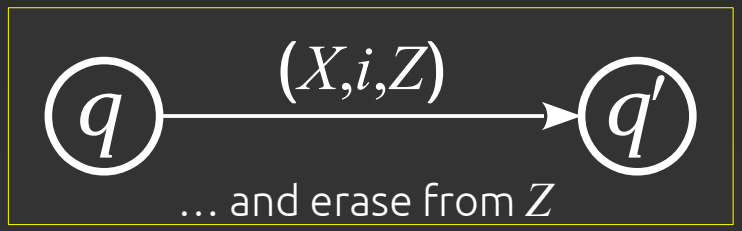
remember: # means "empty register content"

A small detail: register modes

So far we assumed: registers initially empty, not possible to erase them, or have name duplicates.
We now generalise:

- *Duplicates* :
  - (*S*) single
  - (*M*) multiple
- *Erasure* :
  - (*F*) full
  - ($\#_0$) initially empty
  - (#) eraseable



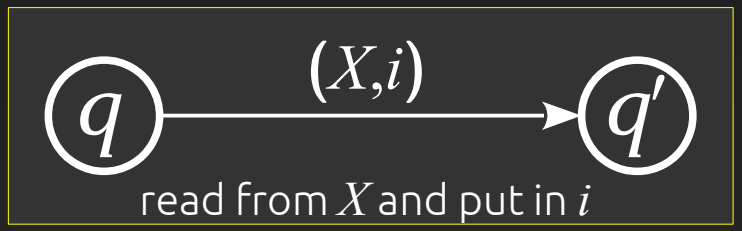


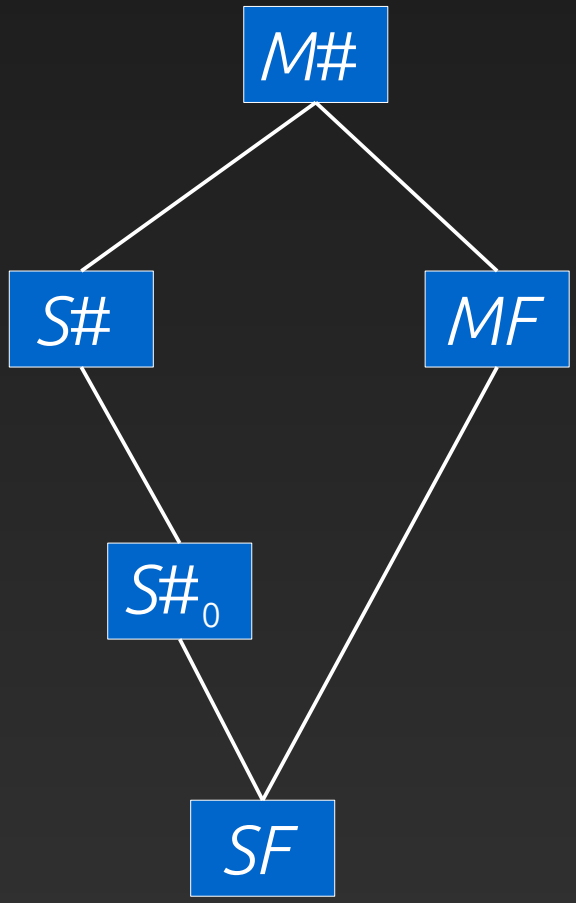
remember: # means "empty register content"

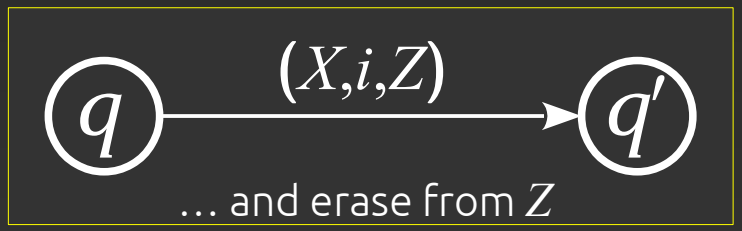
A small detail: register modes

So far we assumed: registers initially empty, not possible to erase them, or have name duplicates.
We now generalise:

- *Duplicates* :
  - (*S*) single
  - (*M*) multiple
- *Erasure* :
  - (*F*) full
  - ($\#_0$) initially empty
  - (#) eraseable

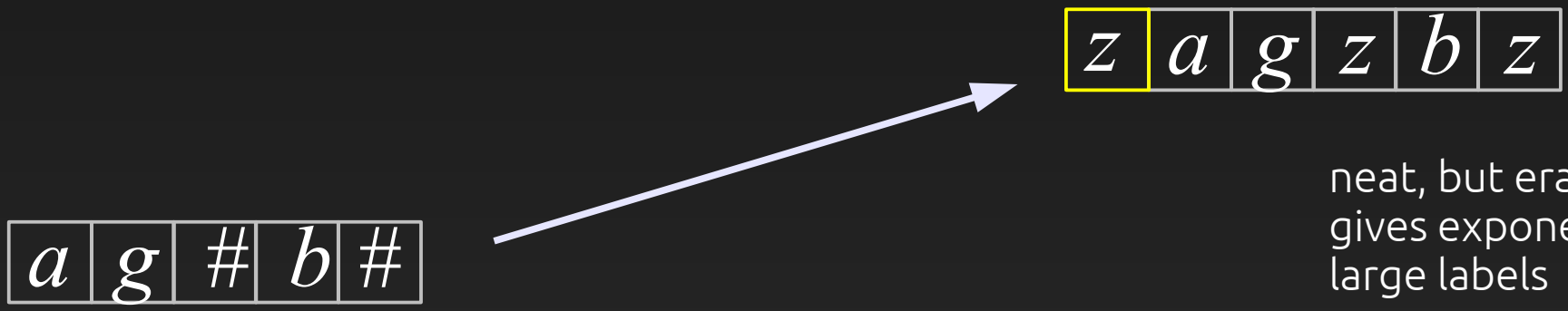






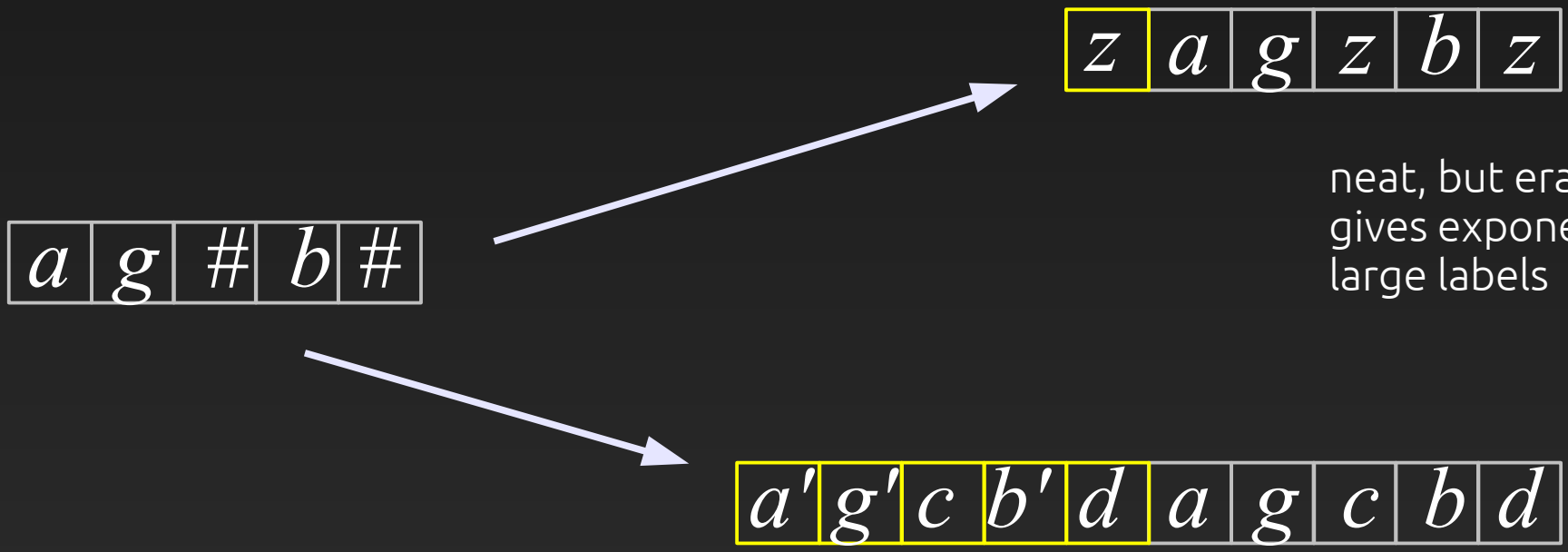
remember: # means "empty register content"

Example: (*S#*) → (*MF*)



neat, but erasing gives exponentially large labels

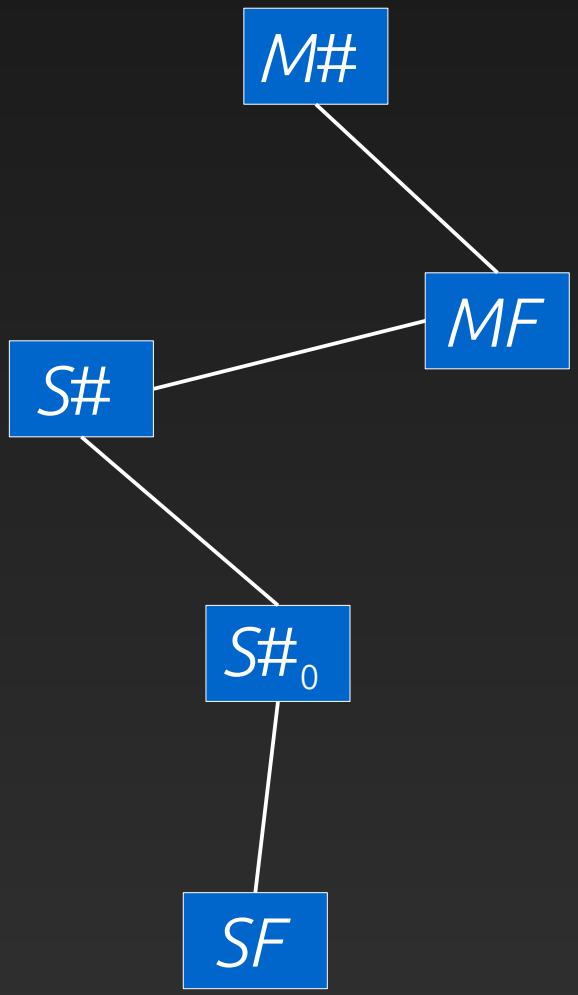
Example: (*S#*) → (*MF*)


a g # b #
z a g z b z
neat, but erasing
gives exponentially
large labels
a' g' c b' d a g c b d
concise, as each
name appears
at most twice

Complexity Picture

Duplicates :
- (*S*) single
- (*M*) multiple

Erasure :
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable



EXPTIME solvability

To decide bisimilarity of two configurations of size R:

- we need $2R$ names to represent all possible name matchings between them
- plus one name that stands for “locally fresh”
- and another one for “globally fresh”

EXPTIME solvability

To decide bisimilarity of two configurations of size R:

- we need $2R$ names to represent all possible name matchings between them
- plus one name that stands for “locally fresh”
- and another one for “globally fresh”

→ $2R+2$ names, that we can encode inside states:

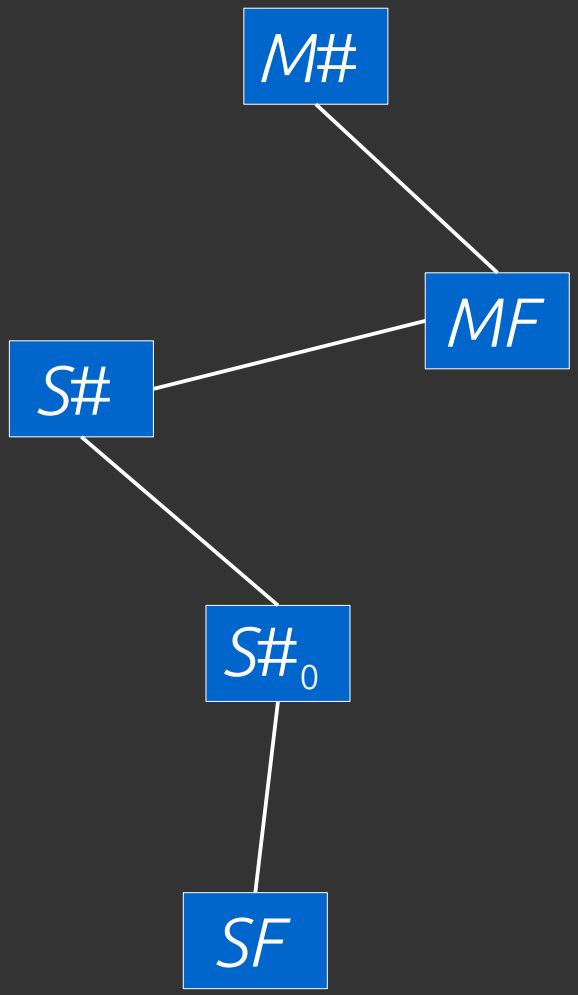
$$Q \longrightarrow Q \times (2R+2)^R$$

and bisimilarity for finite-state automata is in PTIME

Complexity Picture

Duplicates:
- (*S*) single
- (*M*) multiple

Erasure:
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable

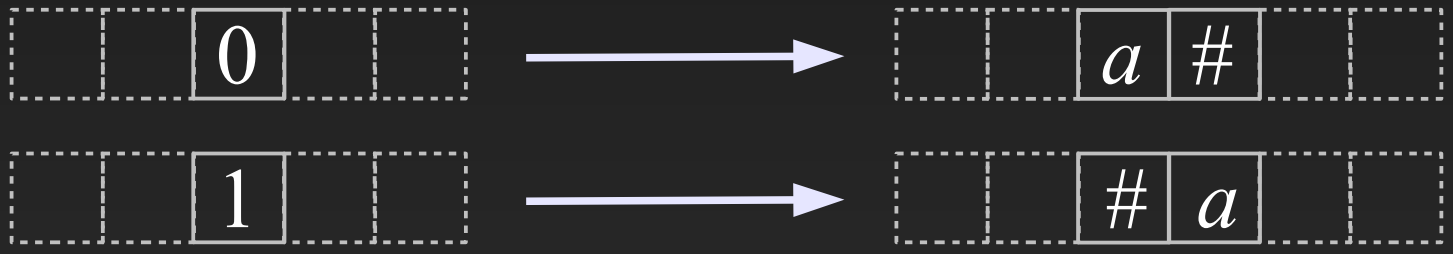


EXPTIME

EXPTIME hardness

The (*S*#) case is EXPTIME-hard:

- reduce from alternating TMs with linear-size tape (ALBA)
- model each cell by two registers:

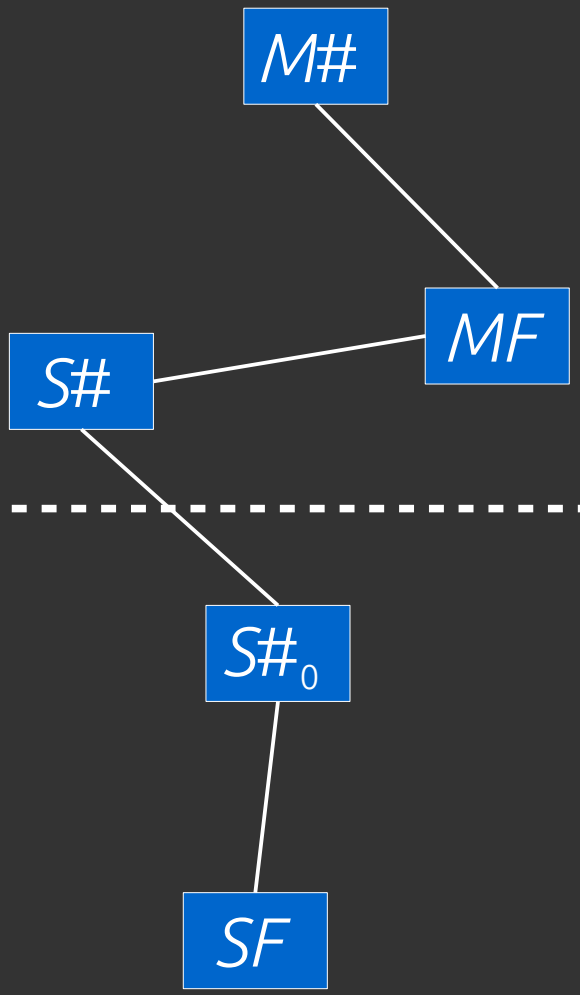


- arrange for non-bisimilarity at rejecting final states
- use Defender forcing for existential states

Complexity Picture

Duplicates:
- (*S*) single
- (*M*) multiple

Erasure:
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable



EXPTIME

EXPTIME

The original case ($S\#_0$)

Disallowing erasures makes impossible our modelling of a linear-size tape...

In fact, the problem is PSPACE complete

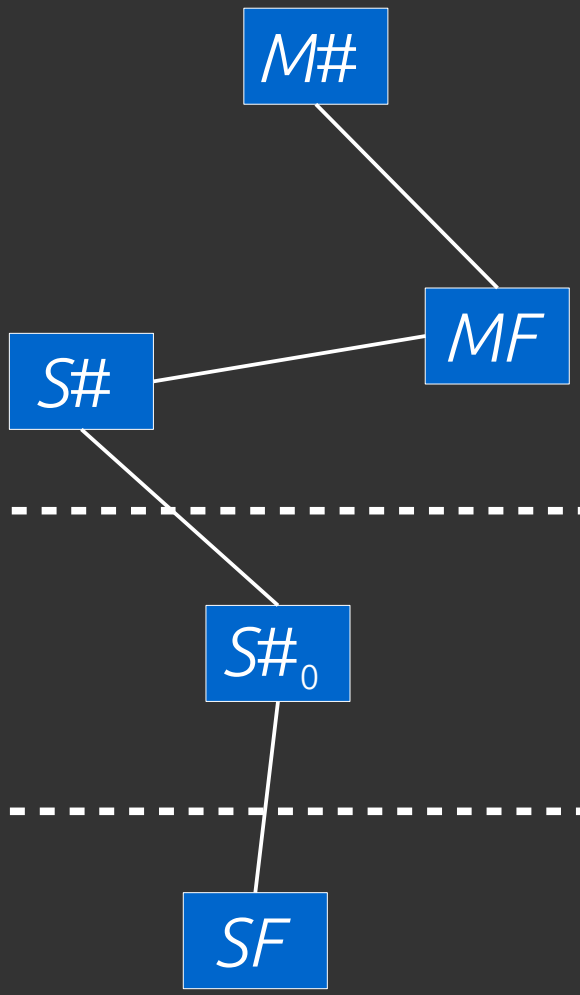
First, we can model boolean assignments (cf. write-once tape), which are enough for PSPACE-hardness:

- we reduce from QBF
- Attacker chooses universal variables
- Defender chooses existential ones (via forcing)

Complexity Picture

Duplicates :
- (*S*) single
- (*M*) multiple

Erasure :
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable



EXPTIME

EXPTIME

PSPACE

PSPACE solvability: difficult

Our best bet is APTIME = PSPACE

- problem: while we cannot simulate a linear tape, we still have a lot of configurations!
  - even for RAs: exponentially many

PSPACE solvability: difficult

Our best bet is APTIME = PSPACE

- problem: while we cannot simulate a linear tape, we still have a lot of configurations!
  - even for RAs: exponentially many

We look into internal symmetries of FRAs:

- symbolic reasoning: we are only interested in how configurations are related, not their actual content
- group representations: we express these interrelations compactly via permutation groups
- bounded history: it suffices to consider histories of size up to $2R$

Indexed Bisimilarity

For each $i \in \omega$, we write $\kappa_1 \overset{i}{\sim} \kappa_2$ if **D** has a strategy for not losing the game within i rounds

We can show that:

$$\overset{0}{\sim} \supseteq \overset{1}{\sim} \supseteq \overset{2}{\sim} \supseteq \ldots \supseteq \overset{i}{\sim} \supseteq \ldots \quad \text{and} \quad \sim \; = \bigcup_{i \in \omega} \overset{i}{\sim}$$

PSPACE solvability

$$\overset{0}{\sim} \supseteq \overset{1}{\sim} \supseteq \overset{2}{\sim} \supseteq \ldots \supseteq \overset{i}{\sim} \supseteq \ldots \quad \text{and} \quad \sim \; = \bigcup_{i \in \omega} \overset{i}{\sim}$$

Reasoning symbolically:

- each decrease in the indexed chain can be traced back to one of polynomially many factors!

use the fact that strict subgroup chains have bounded length

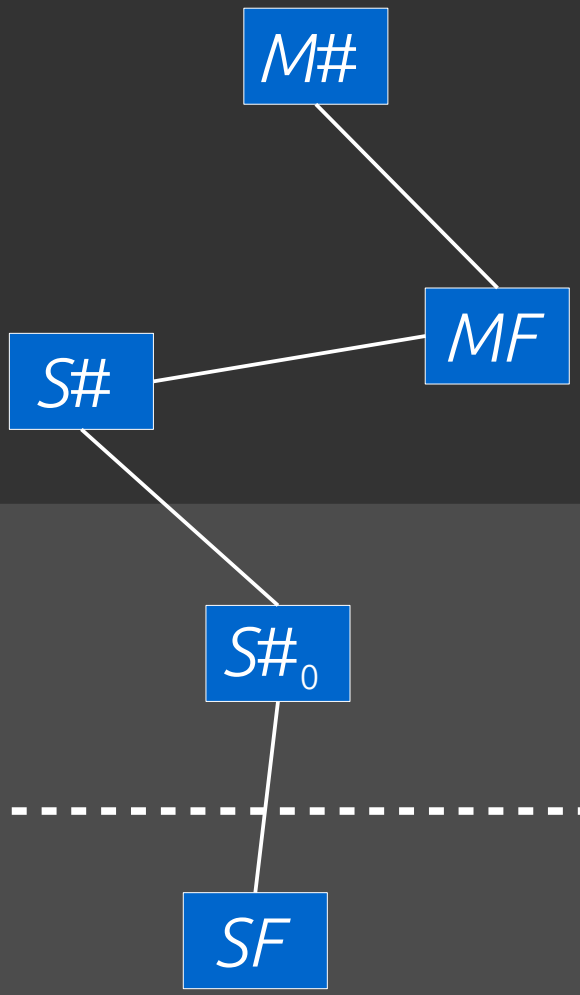
This means there is a final polynomial-size i

- gives a polynomial bound for the bisimulation game, hence APTIME

Complexity Picture

Duplicates :
- (*S*) single
- (*M*) multiple

Erasure :
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable



EXPTIME

PSPACE

PSPACE

The (*SF*) case: NP solvability

Here, reasoning symbolically we can:

- guess and verify an actual symbolic bisimulation
- using compactness of group representations, achieve this in polynomial time

Complexity Picture

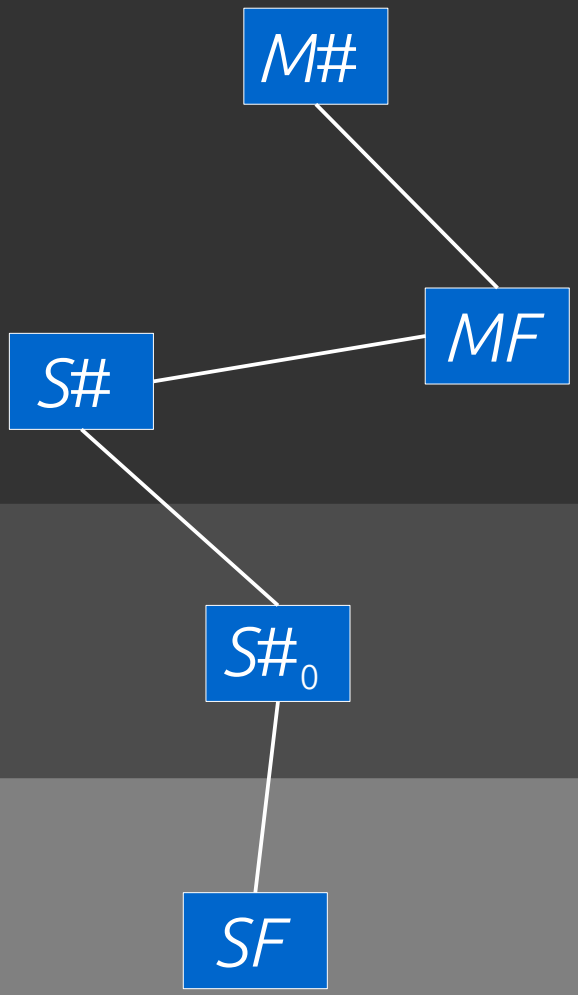
Duplicates :
- (*S*) single
- (*M*) multiple

Erasure :
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable

EXPTIME

PSPACE

NP



[Cheng & Kaminski '98, Segoufin '06]

Pushdown Register Automata (PDRA)

[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

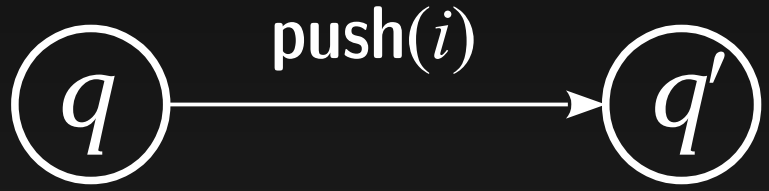
pushdown stack

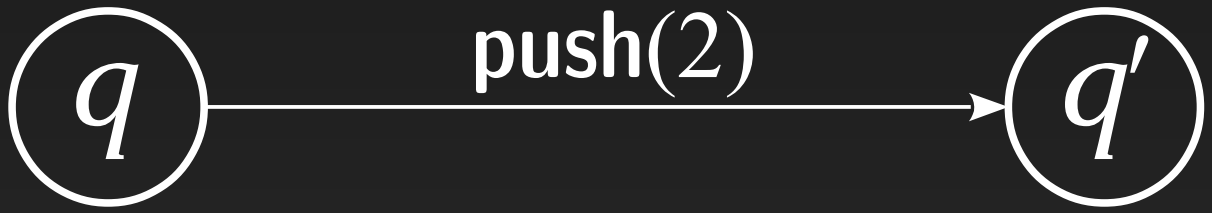
registers & stack store names

Label λ of the form:

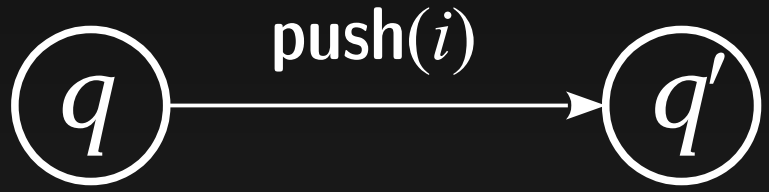
- **read**(i), $i \in \{1,\ldots,R\}$
- **fresh**(i), $i \in \{1,\ldots,R\}$
- **push**(i), $i \in \{1,\ldots,R\}$
- **pop**(i), $i \in \{1,\ldots,R\}$
- **pop-fresh**

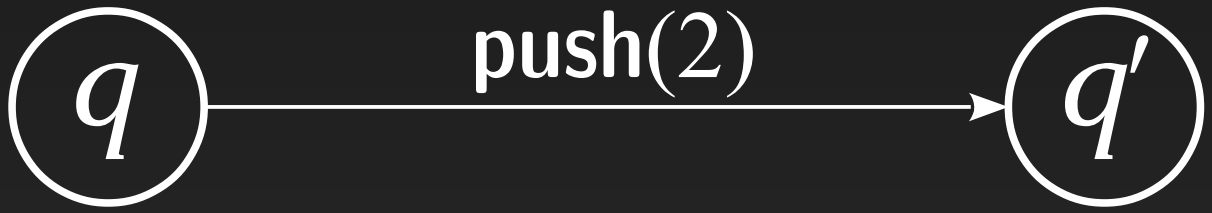
Transitions:

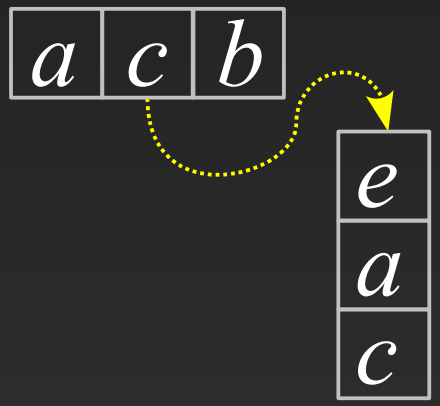




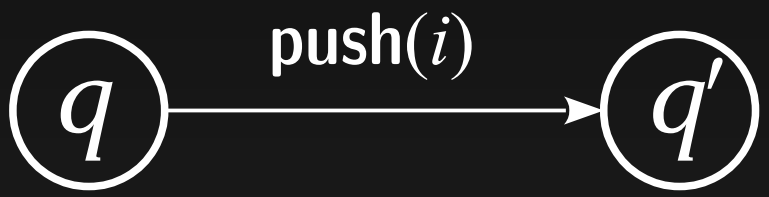
Transitions:

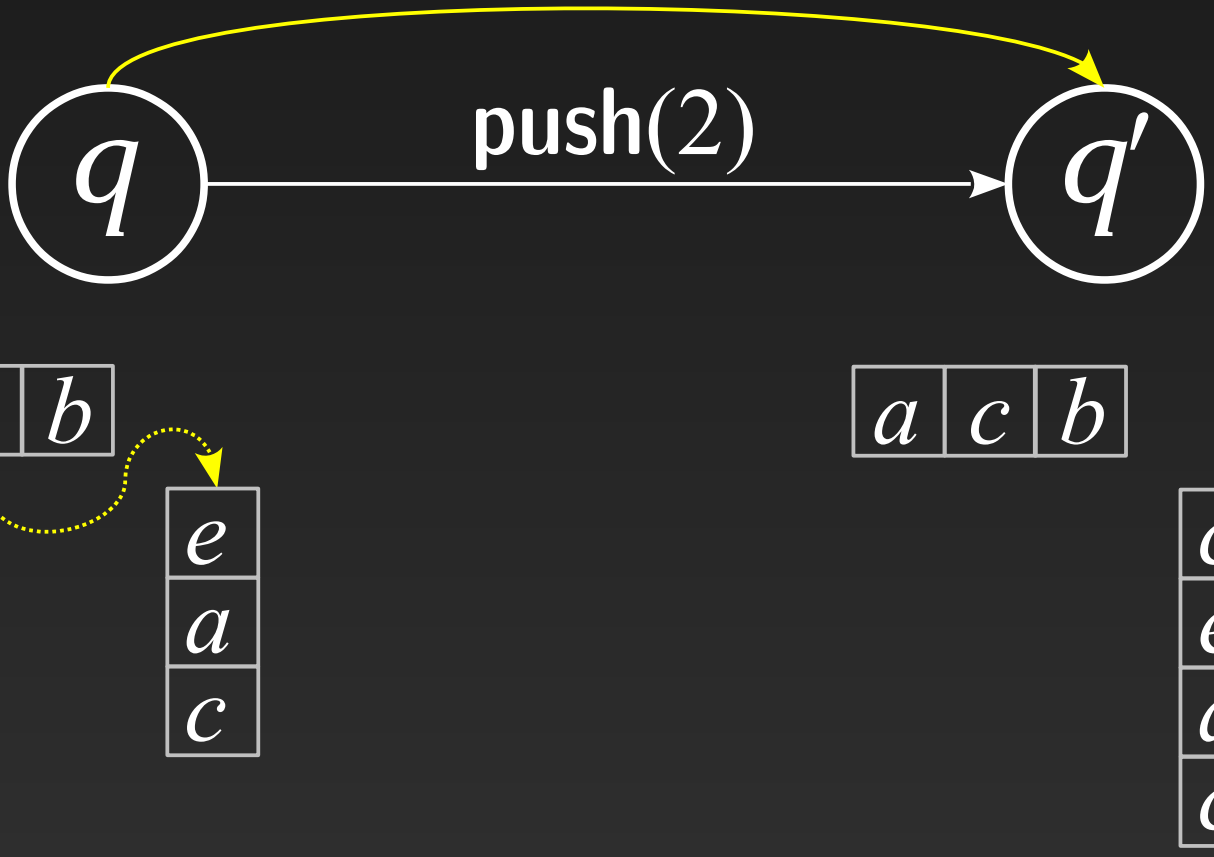




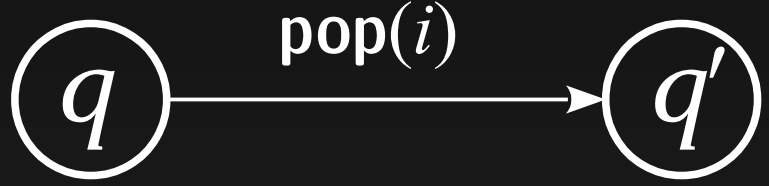


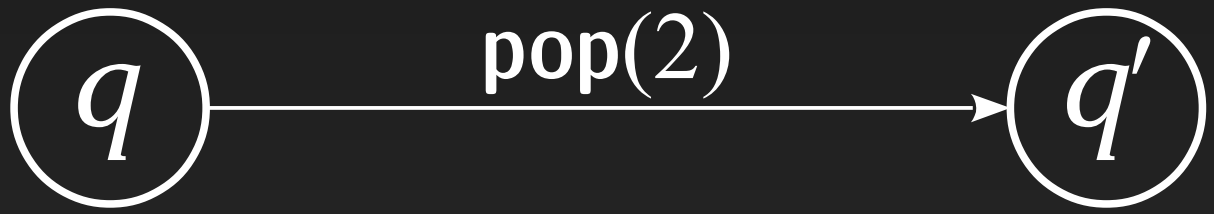
Transitions:



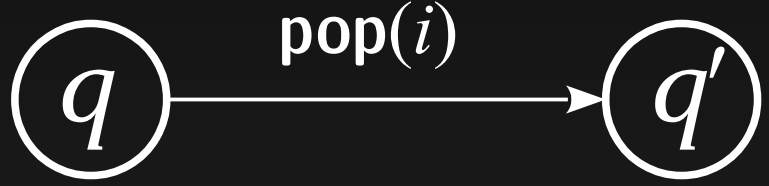


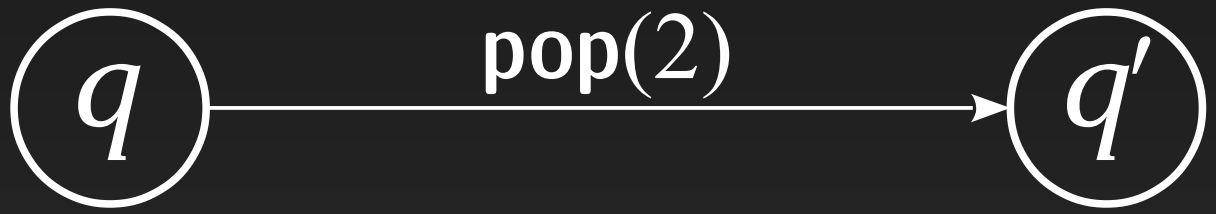
Transitions:

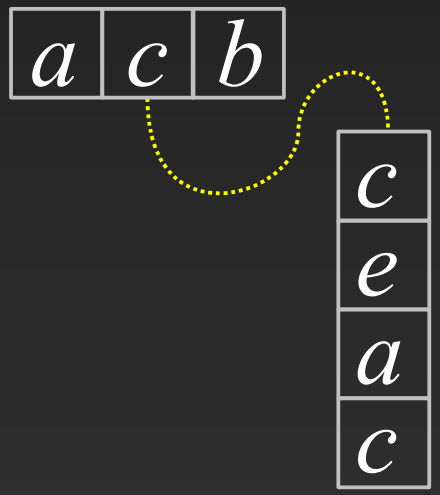




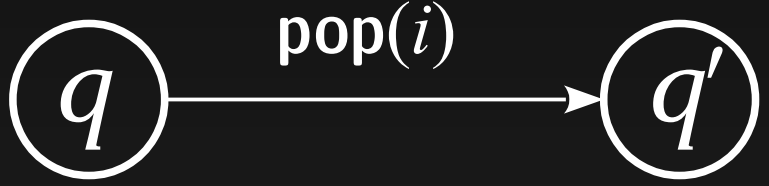
Transitions:

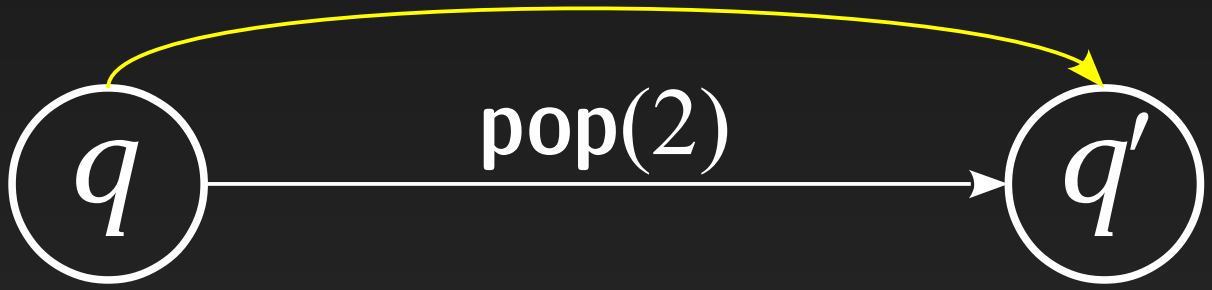


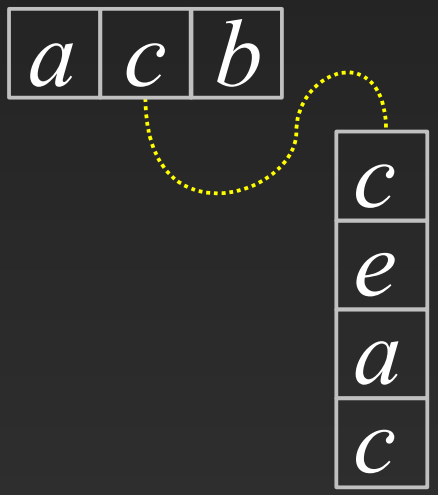


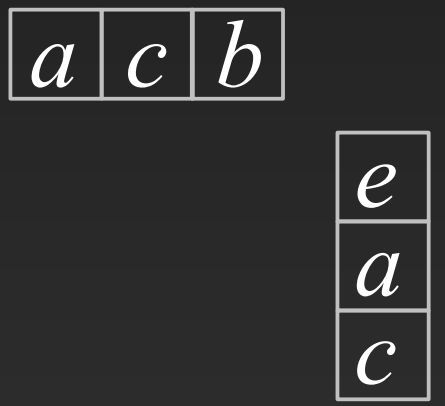


Transitions:

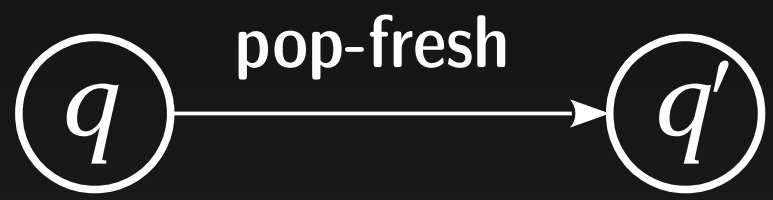


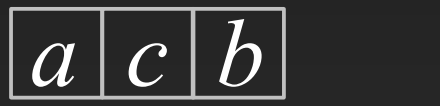




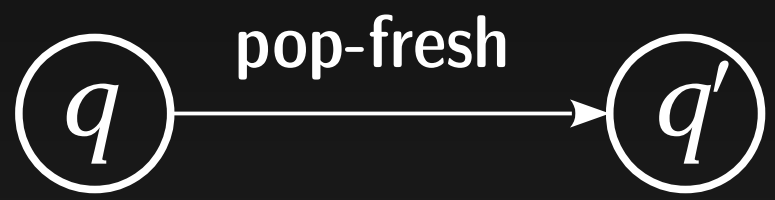


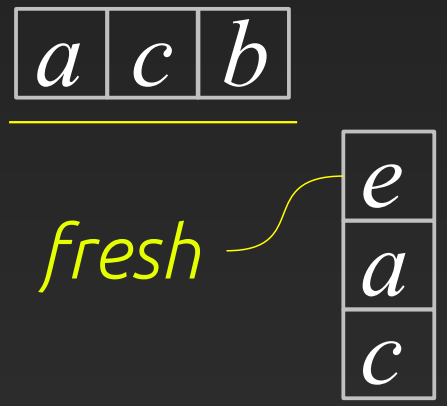
Transitions:



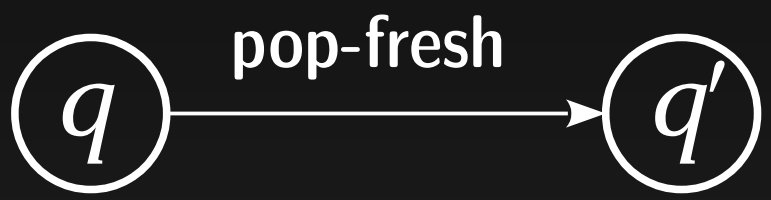


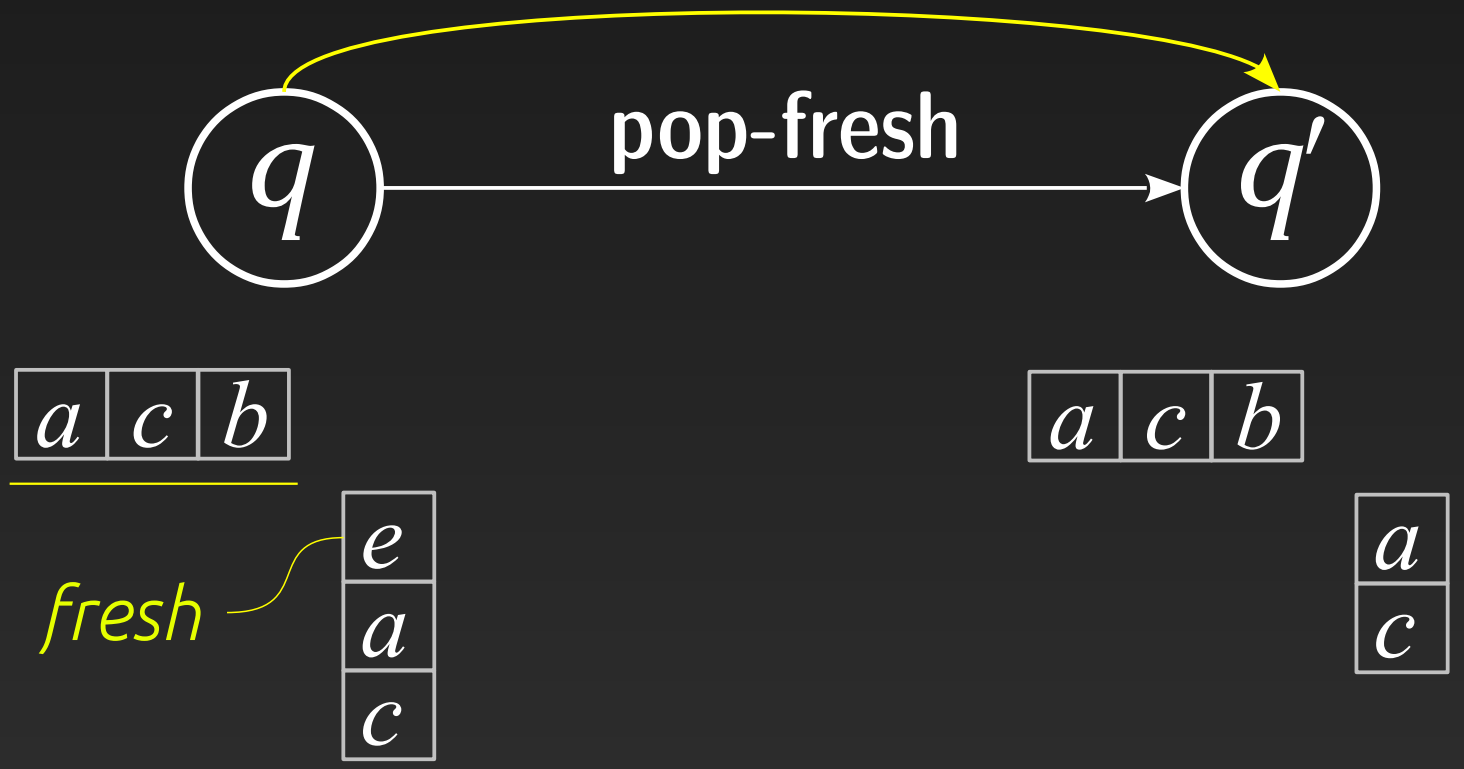
Transitions:





Transitions:

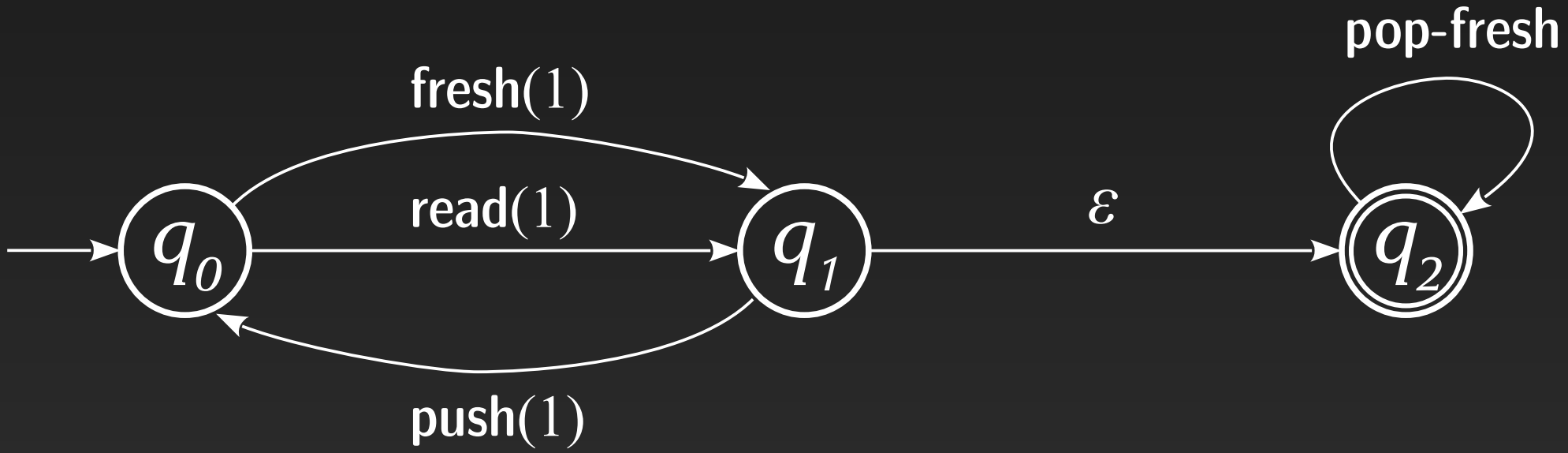




Example

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

(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.

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.

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.

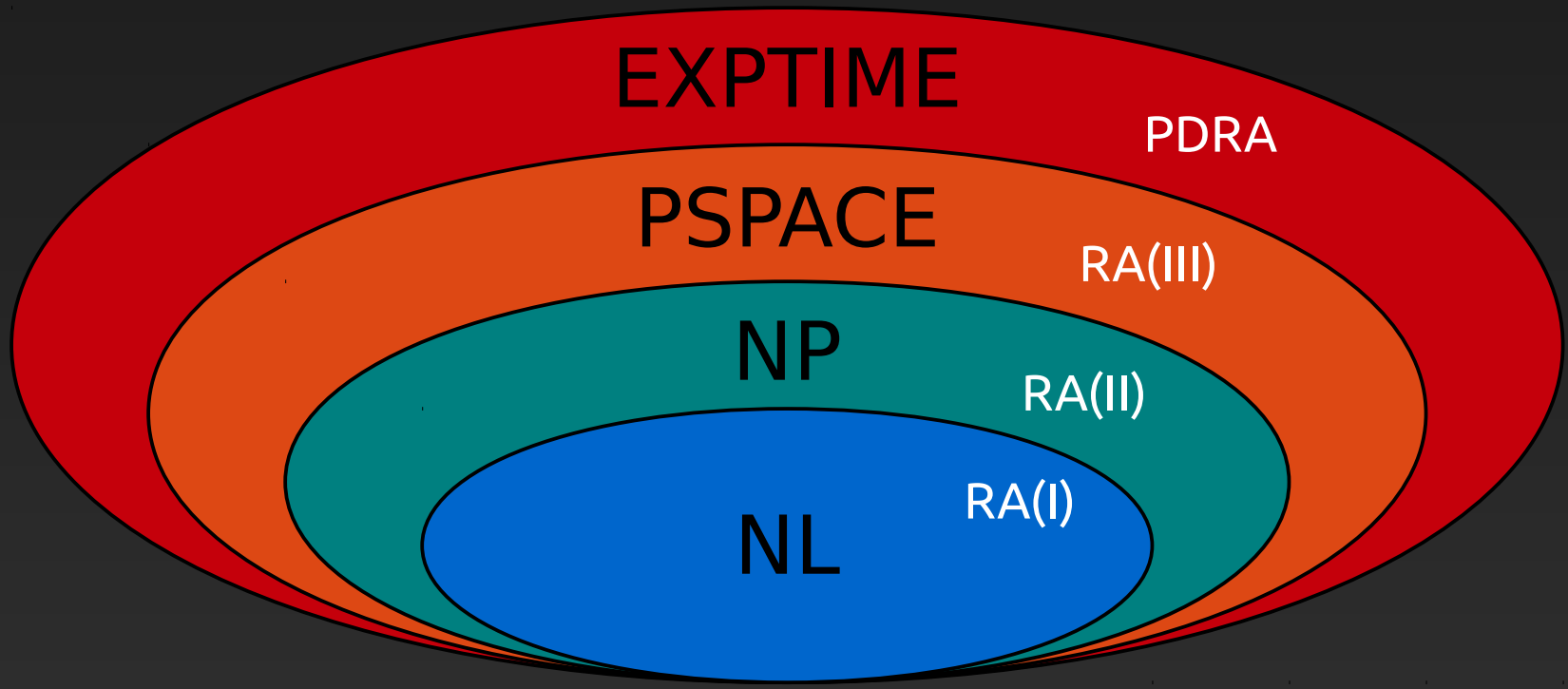
EXPTIME solvability for (*M#*):

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

EXPTIME hardness for (*SF*):

- Reduction from TMs with stack – more difficult

Reachability/non-emptiness picture



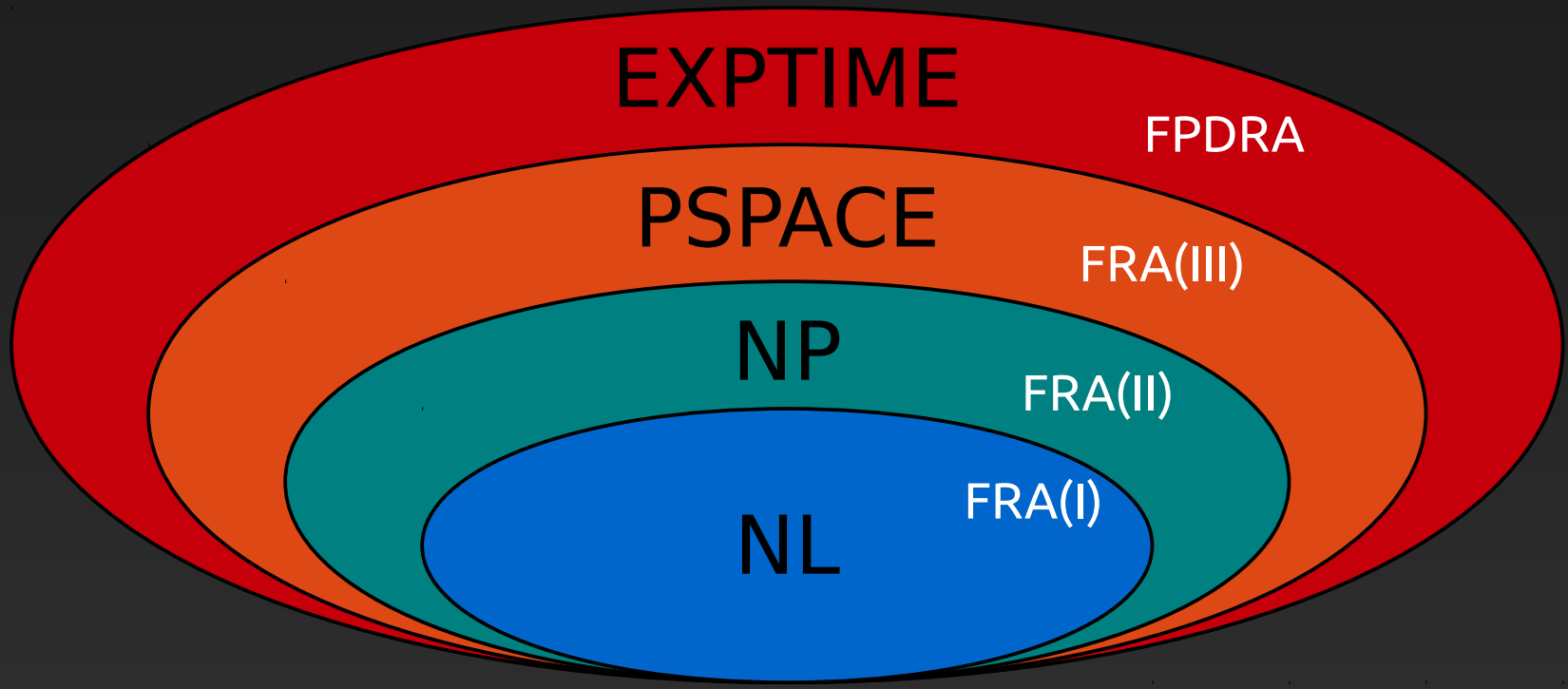
[Murawski & Tz '12]

Adding global freshness

Same as the RA → FRA case: add **gl-fresh** transitions

- No more $3R$ result!
- However, for reachability we can simulate global fresh:
  - by local fresh + tags on registers and stack
  - the tags ensure distinctness in every comparison
  - the simulation is exponential: $Q' = Q \times 2^R$
    - summing up: $Q'' = Q \times 2^R \times 3R^R$

Reachability/non-emptiness for FRAs



Concluding

Fresh-Register Automata:

- Class of automata over infinite alphabets
  “natural” for computation with names/resources
- new landscape of algorithms and results
- applications in verification

Things to do:

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

Concluding

thanks

Fresh-Register Automata:

- Class of automata over infinite alphabets
  “natural” for computation with names/resources
- new landscape of algorithms and results
- applications in verification

Things to do:

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

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:

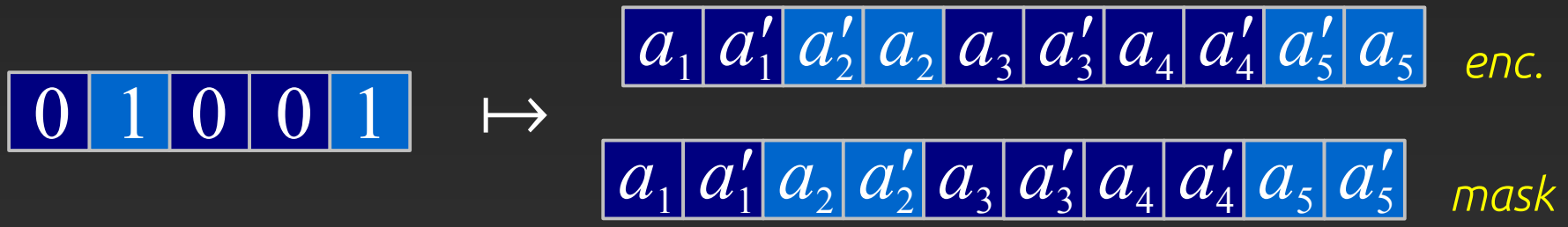
  - now, instead, we use *mask encodings* + the stack:

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...

More results!

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

Theorem: Register automata capture configurations that can reach "regular" sets.

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

Higher-order PDRA: Extensions with stacks of stacks

Theorem: Reachability is undecidable at order 2.

- *reduce from Pebble automata language emptiness*