Automata over infinite alphabets: Investigations in Fresh-Register Automata

Nikos Tzevelekos, Queen Mary University of London

Andrzej Murawski, University of Warwick

Radu Grigore & Steven Ramsay, University of Oxford

Logical Foundations of Data Science, UCL, Nov 2015

Supported by a Royal Academy of Engineering Research Fellowship

infinite alphabets & program behaviour

```
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();
}
```

infinite alphabets & program behaviour

```
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();
}
```

Programs with usage of resources/names can go beyond finite alphabets (cf. modelling/analysis of programs)

– but in a *parametric way*

What this talk is about

This talk is about an automata model over infinite alphabets akin to finite-state automata:

finite-state + registers + freshness oracles

We give an overview of their expressiveness & talk about

- emptiness, closures
- bisimilarity
- extensions (pushdown, classes/histories)

Automata for 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 for infinite alphabets

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

can only be compared for equality

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

a finite set of constants

- look for notions of regularity, CFGs, etc.
- devise effective algorithms for reachability, membership, etc.

many (finitely many) automata models

History-Dependent Automata

- π-calculus models, “named sets”, symmetries, bisimulation

[Montanari & Pistore '98, Pistore '99; Montanari & Pistore '00, Ferrari, Montanari & Pistore '02]

Register Automata (aka FMA)

- FSAs with registers, regularity, data words & XML, extensions

[Kaminski & Francez '94, Neven, Schwentick & Vianu '04]
[Sakamoto & Ikeda '00, Demri & Lazić '09; Libkin, Tan & Vrgoc '15; Jurdzinski & Lazić '11, Figueira '12]
[Cheng & Kaminski '98, Segoufin '06]
[Bojańczyk, Muscholl, Schwentick, Segoufin & David '06 , Bjorklund & Schwentick '10]

Nominal Automata

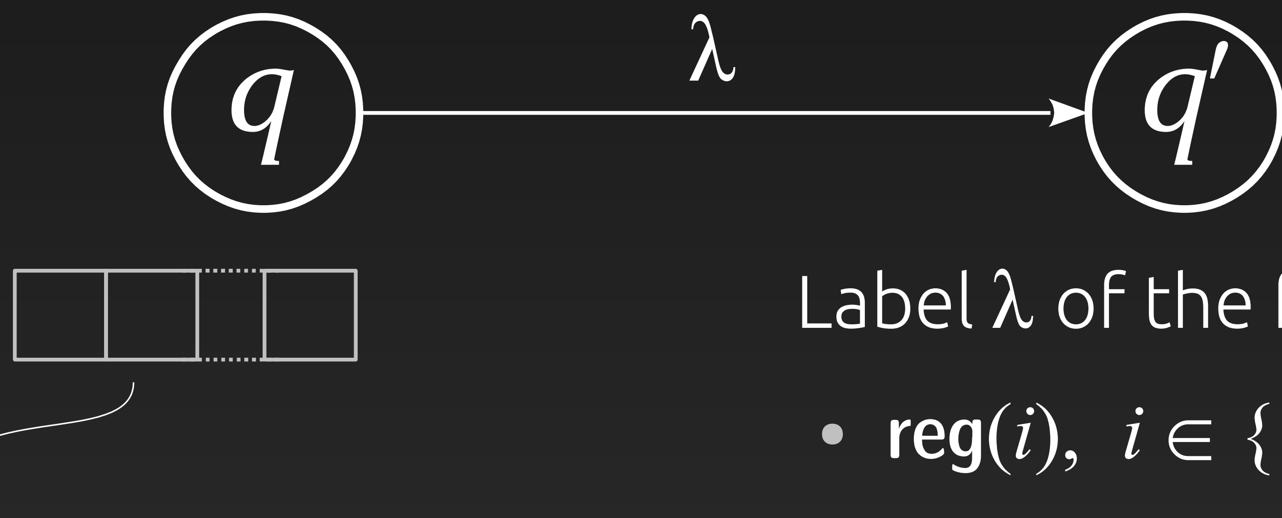
- Finite → finite orbit, used on nominal sets & other group actions

[Bojańczyk, Klin & Lasota '11, '14]

[Kaminski & Francez '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:

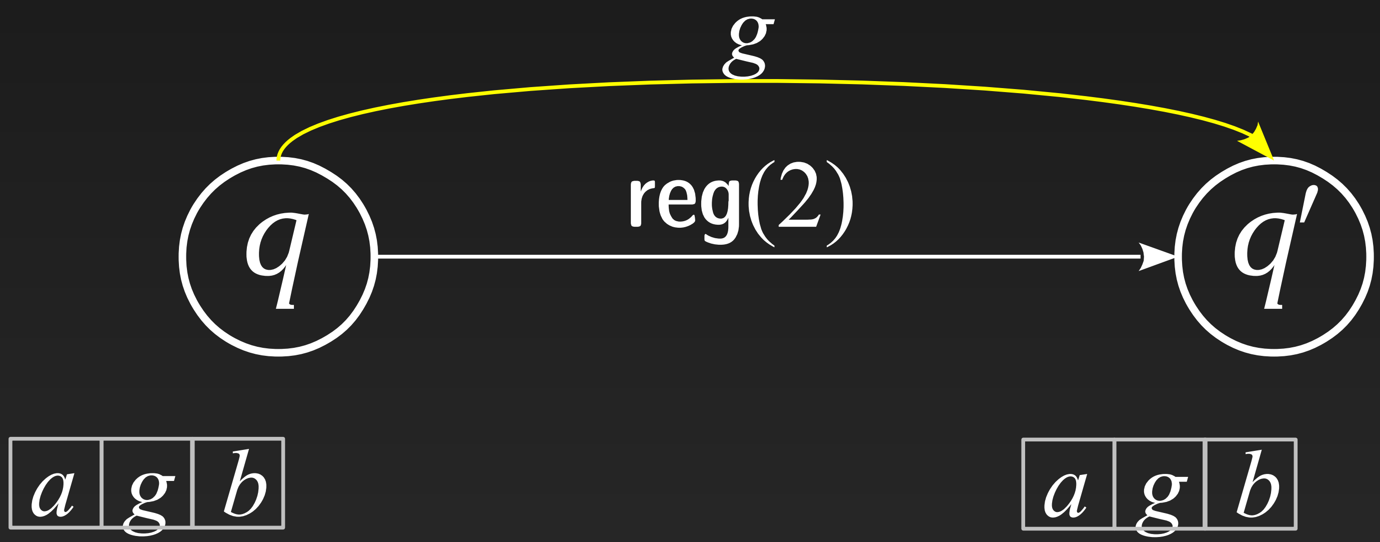
- **reg**(i), $i \in \{1,\ldots,R\}$
- **diff**(i), $i \in \{1,\ldots,R\}$

Transitions: $q \xrightarrow{\mathbf{reg}(i)} q'$

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

a	g	b

Transitions:



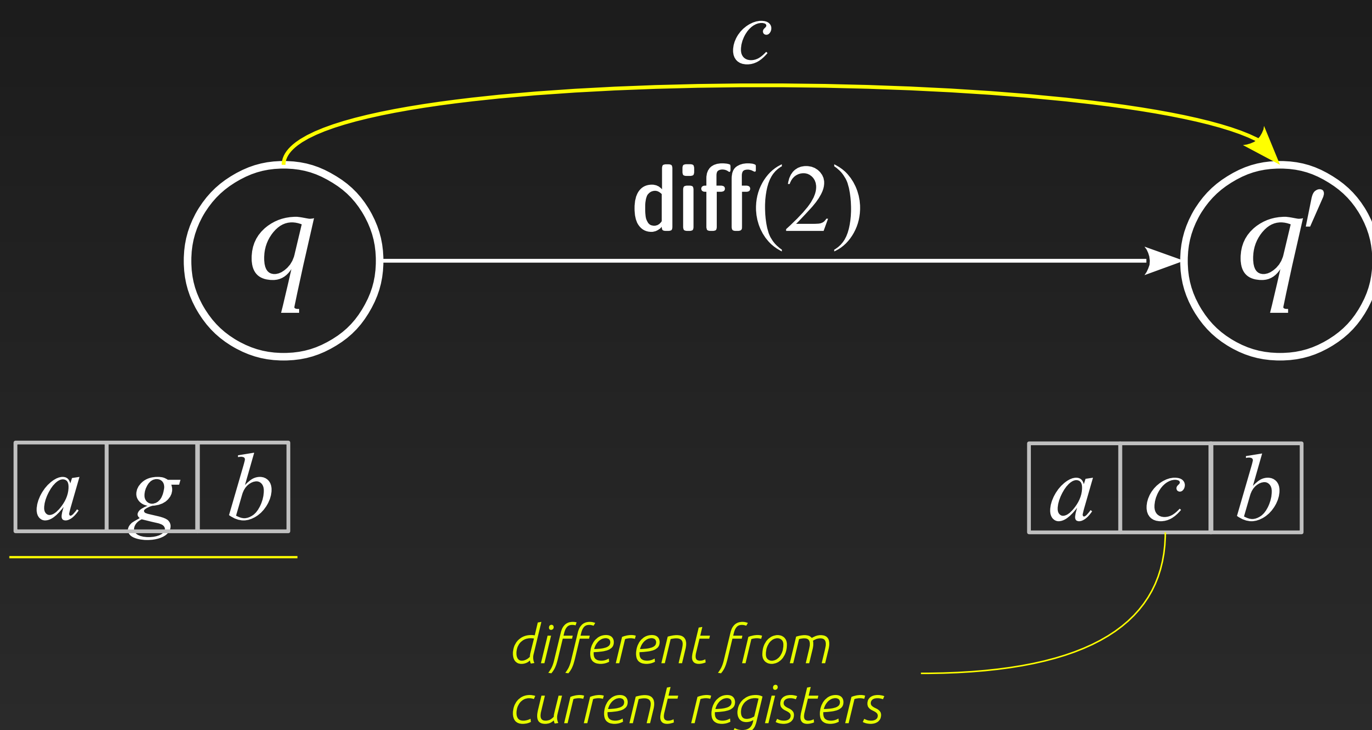
Transitions:

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

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

a	g	b

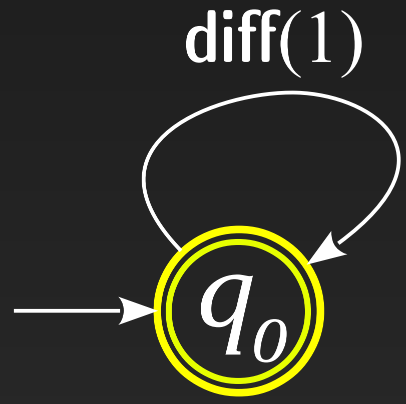
Transitions:



Example

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

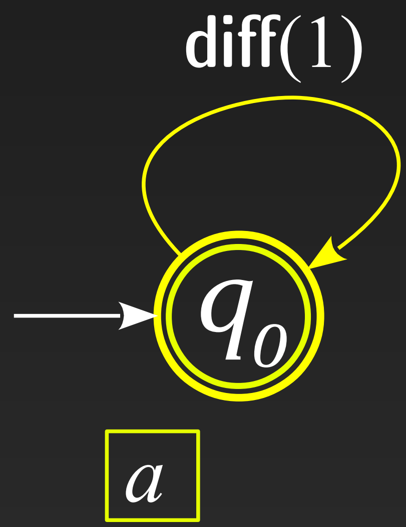
(all strings where each name is distinct from its predecessor)



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)

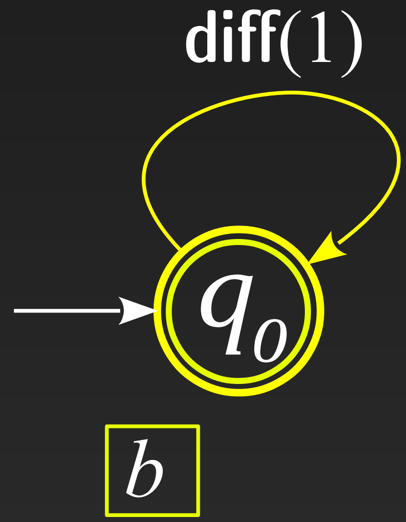


a

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)

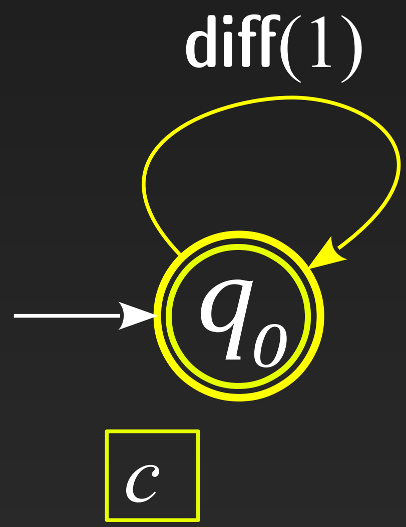


$a\,b$

Example

$$L_1 = \{\ a_1a_2\ldots a_n \in \Sigma^* \mid n \geq 0,\ \forall i < n.\ a_i \neq a_{i+1}\ \}$$

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

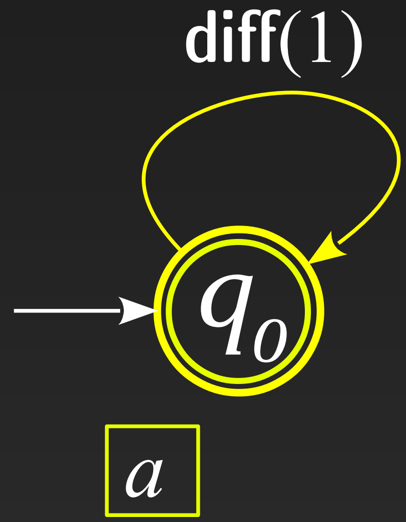


$a\ b\ c$

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

$$L_1 = \{\ a_1a_2 \ldots a_n \in \Sigma^* \mid\ n \geq 0,\ \forall i < n.\ a_i \neq a_{i+1}\ \}$$

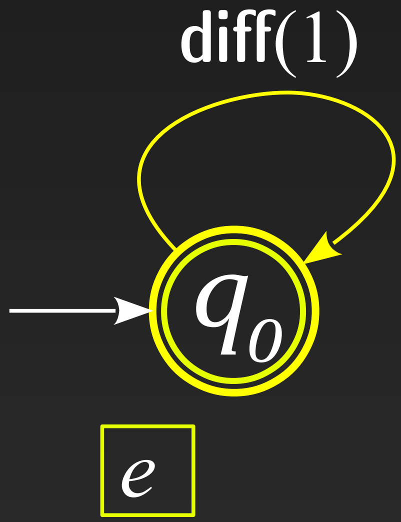
(all strings where each name is distinct from its predecessor)

$a\ b\ c\ a\ d$

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)

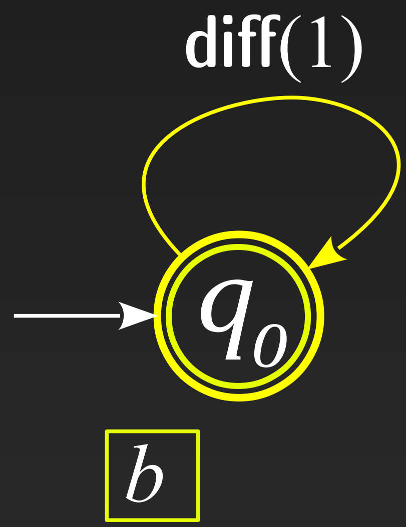


$a\ b\ c\ a\ d\ e$

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)

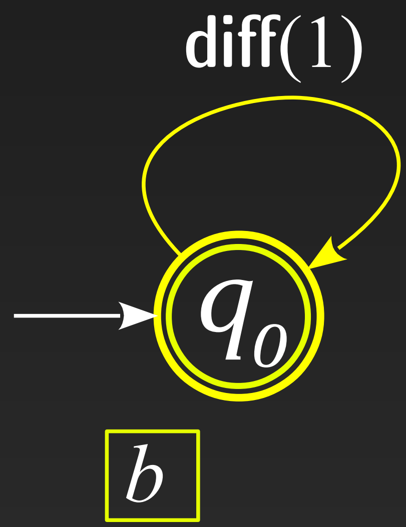


$a\,b\,c\,a\,d\,e\,b$

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)

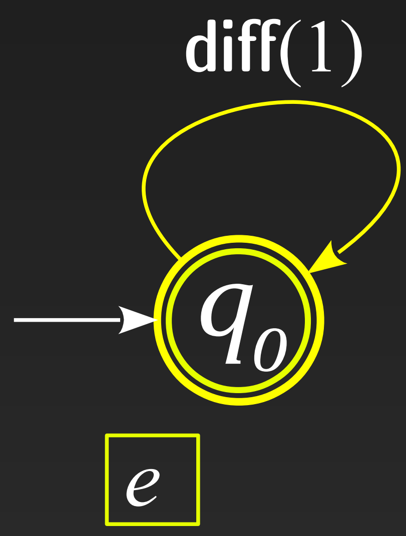


$a\,b\,c\,a\,d\,e\,b\,a\,g\,c\,a\,b$

Example

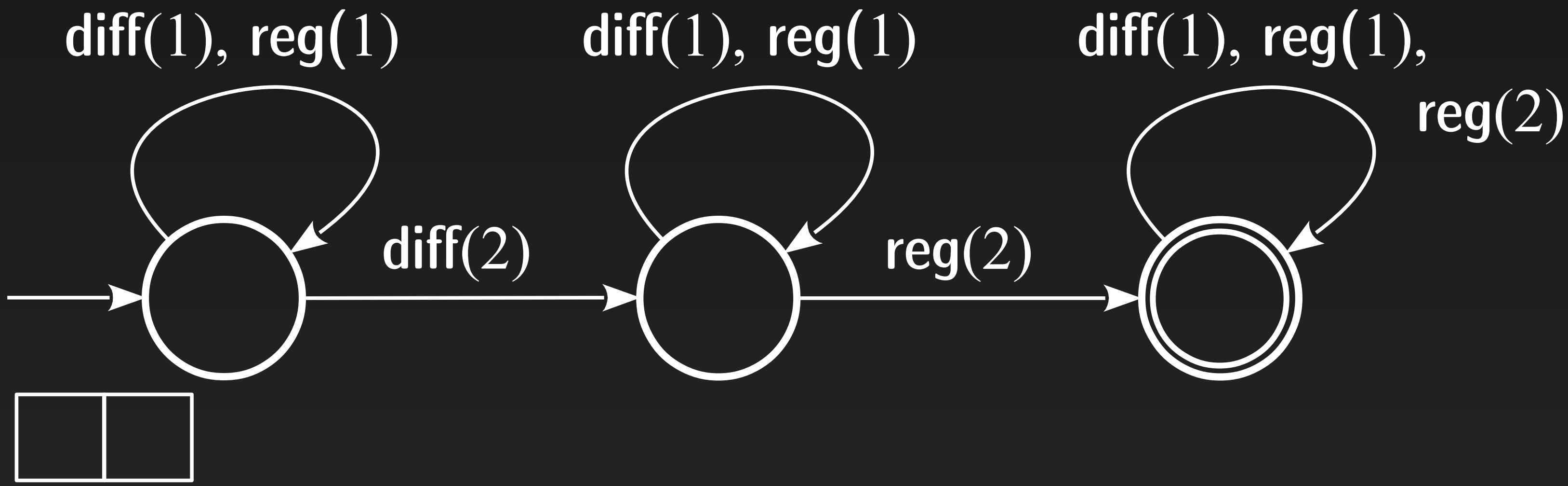
$$L_1 = \{\ a_1 a_2 \ldots a_n \in \Sigma^* \mid\ n \geq 0,\ \forall i < n.\ a_i \neq a_{i+1}\ \}$$

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

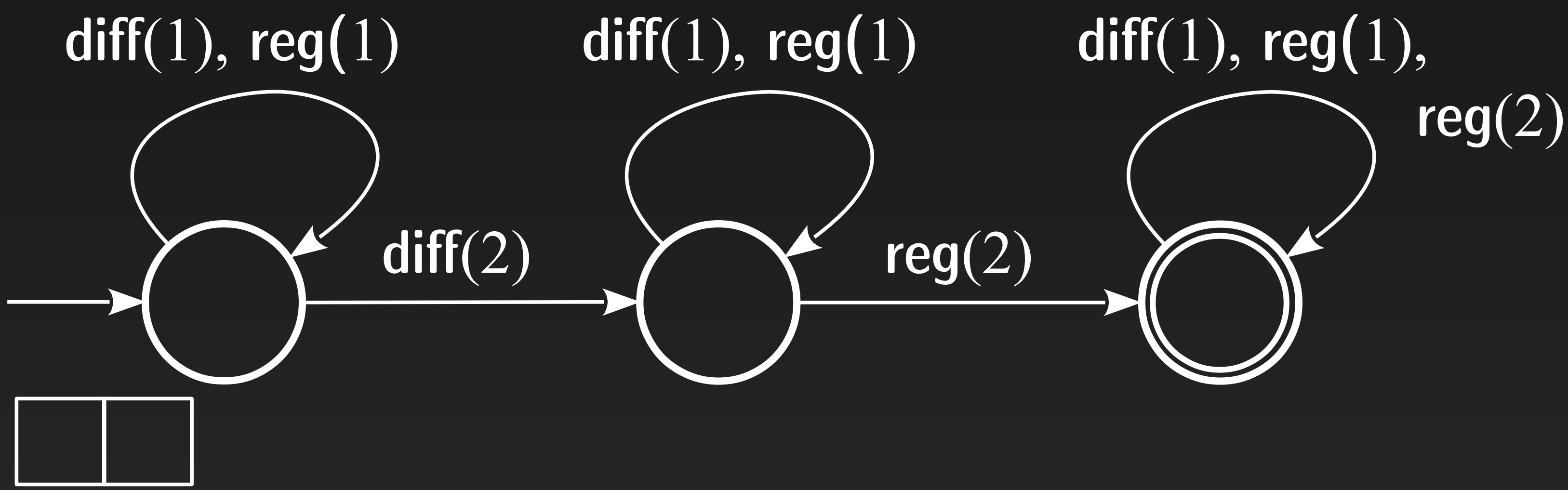


a b c a d e b a g c a b a n d w e l o v e c a k e

Quiz



Quiz



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

(all strings where some name appears twice)

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

(all strings of pairwise distinct names)

– *what about the complement of* L_{fr}*? And that of* $L_{\mathrm{fr}} \bullet L_{\mathrm{fr}}$*?*

RA properties

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

[Kaminski & Francez '94]

- Universality / equivalence undecidable

[Neven, Schwentick & Vianu '04]

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

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

- Can only truly distinguish between $R+1$ names

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 & T. '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();
}
```

but we cannot capture `new`!

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 & T. '13]

[T. '11]

Fresh-Register Automata (FRA)

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

$$q \xrightarrow{\lambda} q'$$

finitely many (say R) registers

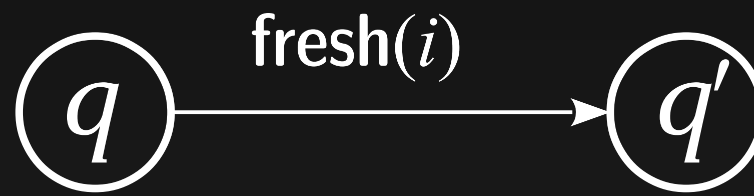
registers store names

Label λ of the form:

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

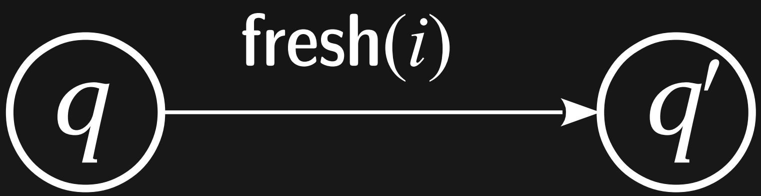
global freshness oracle

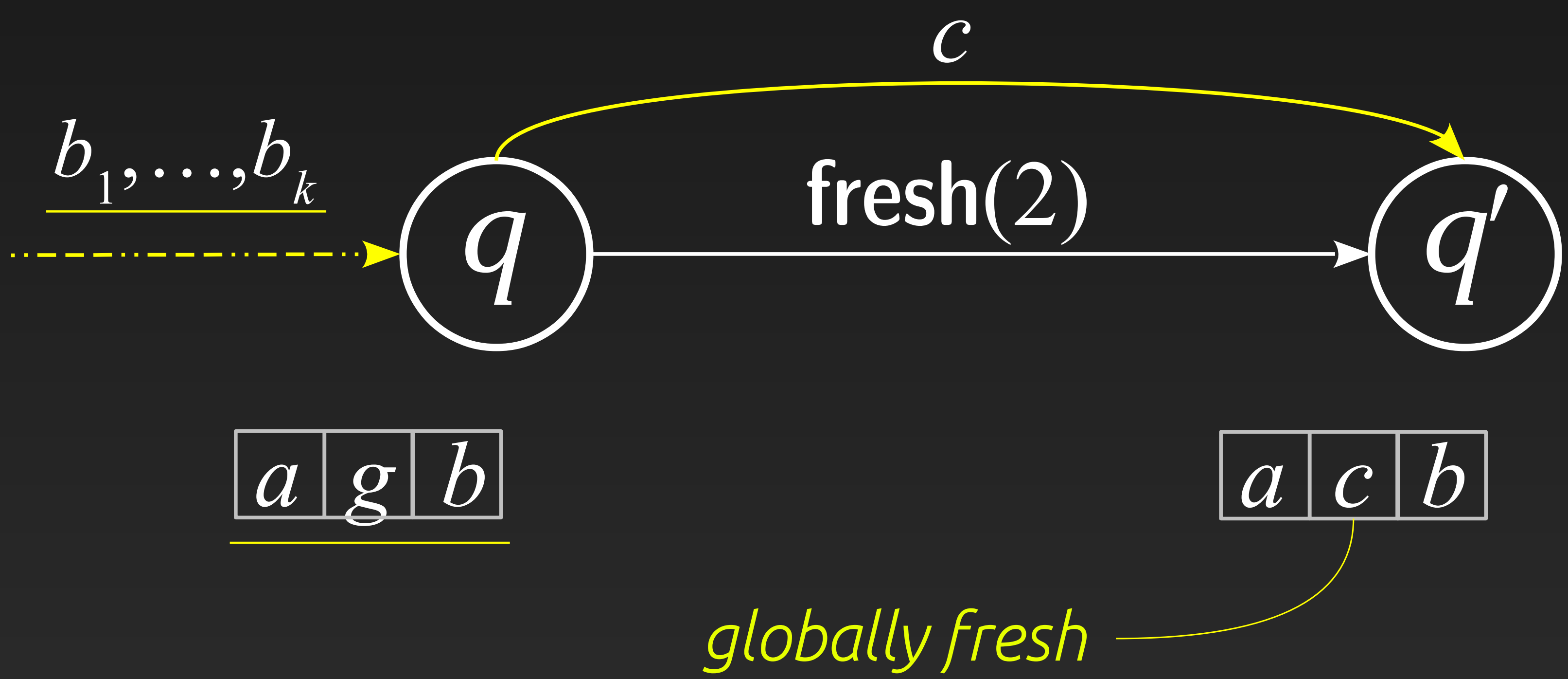
Transitions:



a	g	b

Transitions:

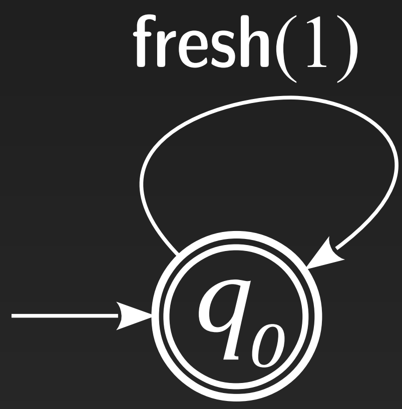


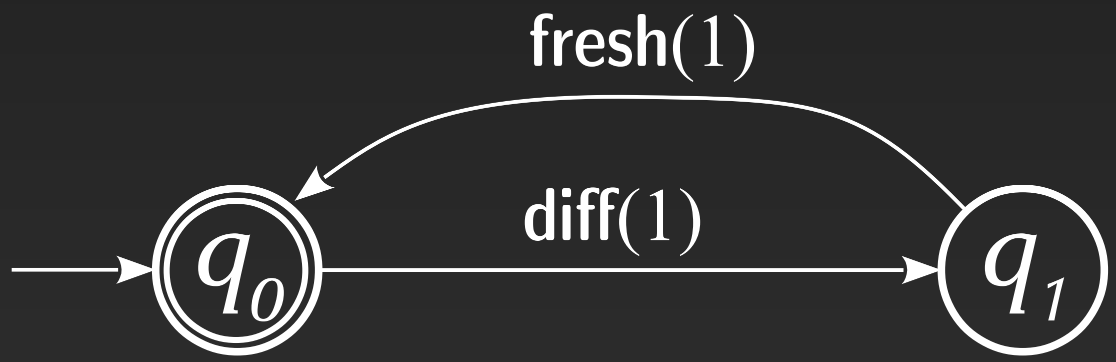


Examples

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

(all strings of pairwise distinct names)





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

FRA properties

- Not closed under complement & not determinisable
  - Closed under $\cup$, $\cap$, but not under $\bullet$, $*$
- Universality / equivalence undecidable (from RAs)
- Decidable emptiness (same as RAs):
  - complexity depends on register "mode" (NL → NP → PSPACE)
- Bisimilarity: decidable [T.11], complexity open

FRAs for program equivalence

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

Program → game model → FRA

effectively:

two programs are equivalent $\Longleftrightarrow$ their FRAs are language equivalent / bisimilar

what we get:

- decision procedures for ML fragments [Murawski & T. '11, '12]
- same for Interface Middleweight Java [Murawski, Ramsay & T. '15]
  http://bitbucket.org/sjr/coneqct/wiki/Home

More applications and variants

History-Dependent Automata

- freshness via “black holes” (histories)
- verification of LTL + allocation

[Pistore '99, Distefano, Rensink & Katoen '02, '04]

Session automata and learning

- freshness, but no **diff**
- canonical forms, decide equivalence

[Bollig, Habermehl, Leucker & Monmege '14]

Kleene algebras for languages with binders

- NKA: KA with ν-binder → match with automata

[Gabbay & Ciancia '11; Kozen, Mamouras, Petrisan & Silva '15]

Investigations in FRAs

Bisimilarity for FRAs (complexity)

- Depends on register mode (NP → PSPACE → EXPTIME)
  - approach uses permutation group theory [Murawski, Ramsay & T. '15]

Context-freeness: Pushdown FRA

[Cheng & Kaminski '98; Segoufin '06]
[Murawski & T. '12]

- Reachability EXPTIME-complete
- Global reachability via "saturation" [Murawski, Ramsay & T. '14]

Freshness oracle: from one to many histories

- History Register Automata (cf. DA/CMA) [Grigore & T. '13]

Semantics formally: configurations

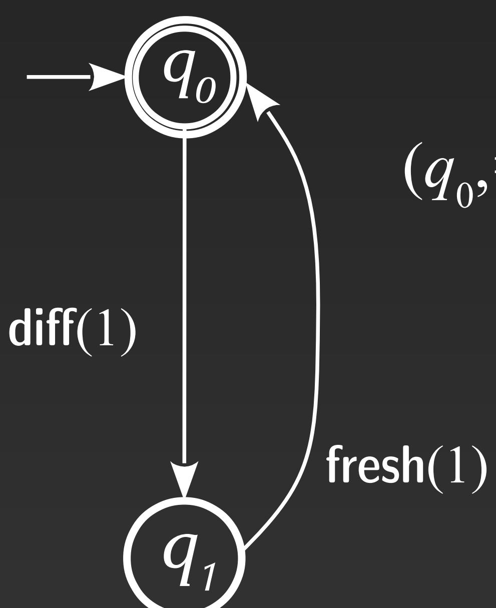
Semantics of FRAs given by configuration graphs:

$$\underbrace{(q,\rho,H)}_{\text{configuration}} \xrightarrow{a} (q',\rho',H')$$

state

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

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



$$(q_0,\#,\emptyset) \xrightarrow{a_1} (q_1,a_1,\{a_1\}) \xrightarrow{b_1} (q_0,b_1,\{a_1,b_1\}) \xrightarrow{c_1} (q_1,c_1,\{a_1,b$$

$$(q_0,\#,\emptyset) \xrightarrow{a_2} (q_1,a_2,\{a_2\}) \xrightarrow{b_2} (q_0,b_2,\{a_2,b_2\}) \xrightarrow{c_2} (q_1,c_2,\{a_1,b$$

$\ldots \qquad \ldots \qquad \ldots$

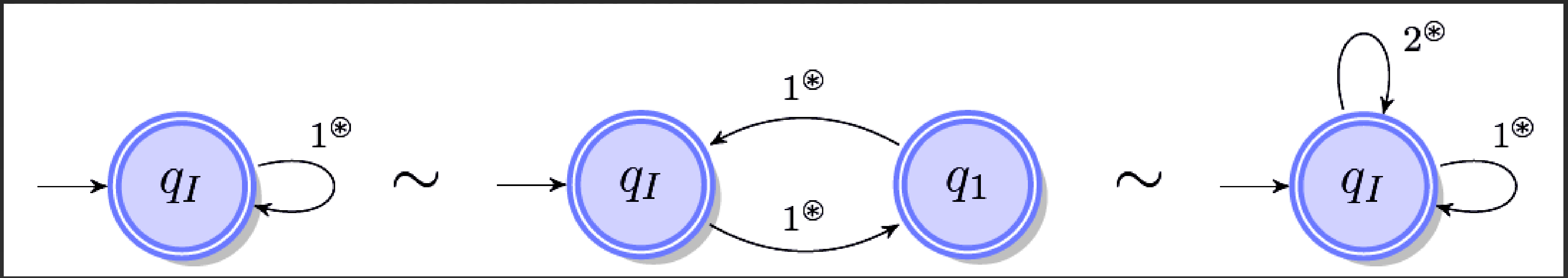
Bisimilarity

A behavioural notion of equivalence:

two configurations κ_1,κ_2 are bisimilar ($\kappa_1 \sim \kappa_2$)
if they can simulate one another name-by-name

We say that two FRAs are bisimilar if their initial configurations are (in the combined conf. graph).

e.g. (writing $1^{\circledast}$ for **fresh**(1)):

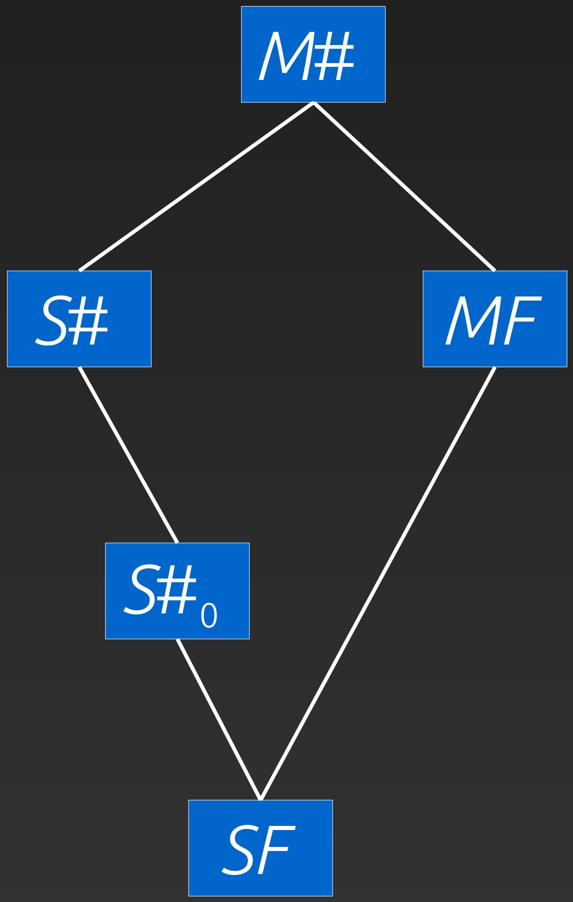


A small detail: register modes

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

Name multiplicity

- (S) single
- (M) multiple



Register fullness

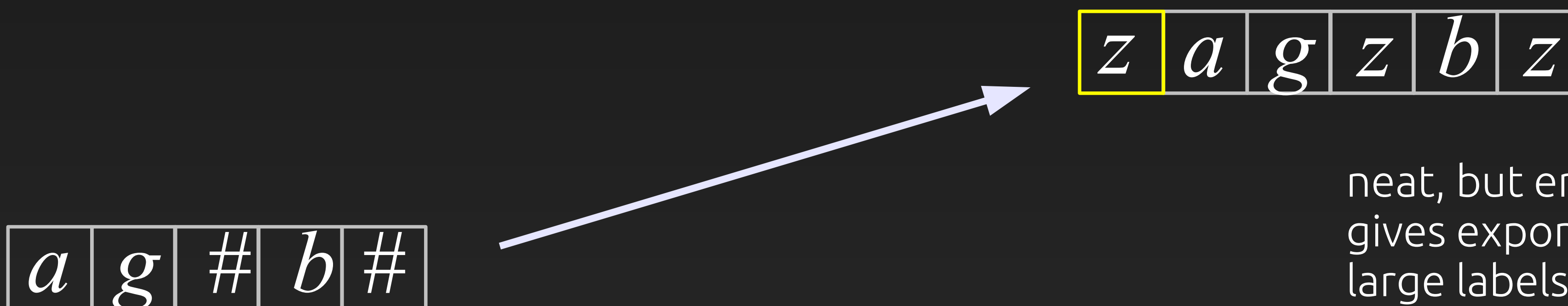
- (F) full
- ($\#_0$) initially empty
- (#) eraseable

is for empty register content

$(S\#) \rightarrow (MF)$

no multiplicities,
but erasing allowed

multiplicities, but
no empty registers

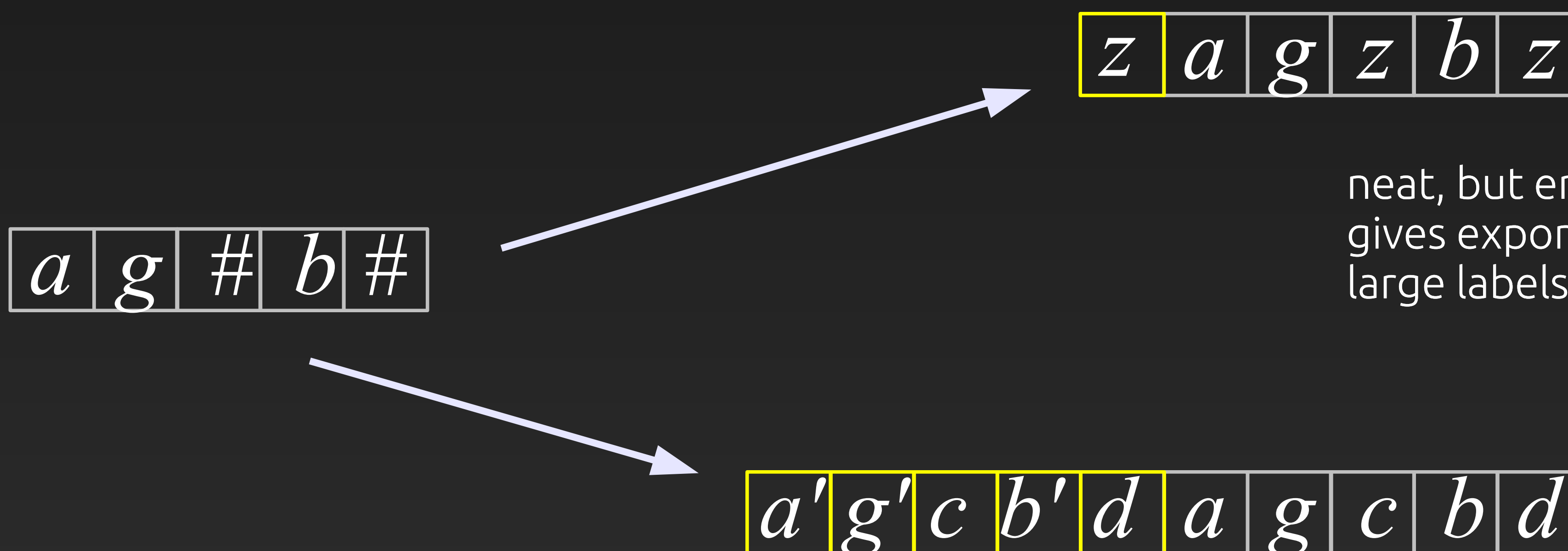


neat, but erasing
gives exponentially
large labels

$(S\#) \rightarrow (MF)$

no multiplicities,
but erasing allowed

multiplicities, but
no empty registers



neat, but erasing
gives exponentially
large labels

concise, as each
name appears
at most twice

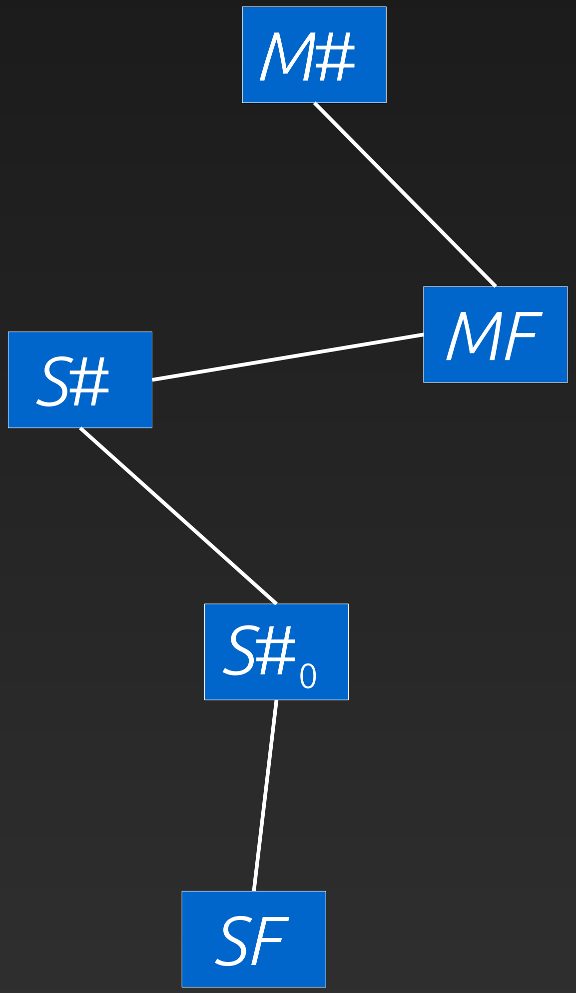
Complexity Picture

Multiplicity:

- (S) single
- (M) multiple

Fullness:

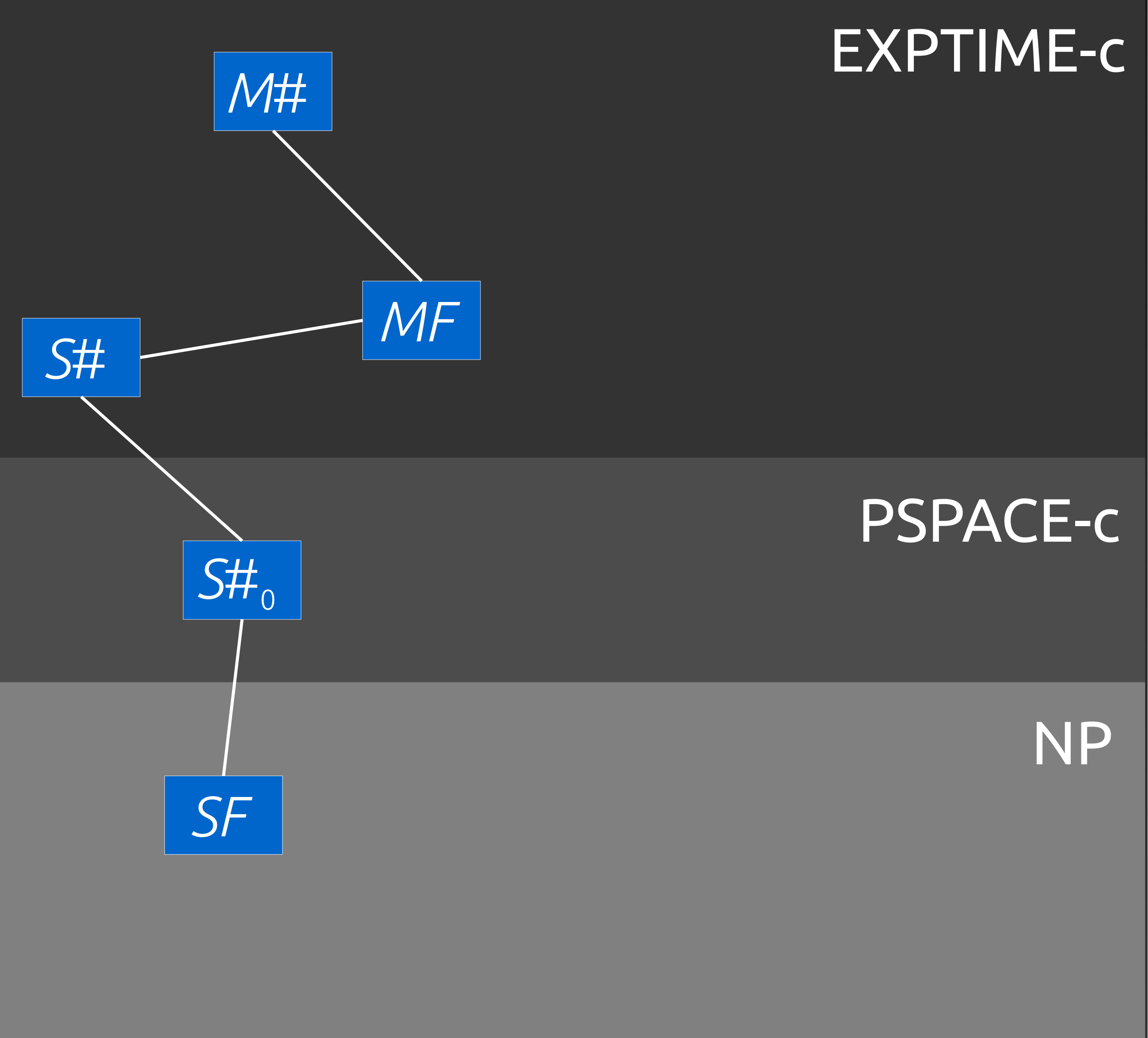
- (F) full
- ($\#_0$) initially empty
- ($\#$) eraseable



Complexity Picture

Multiplicity:
- (S) single
- (M) multiple

Fullness:
- (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 "different"
- and another one for "fresh"

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

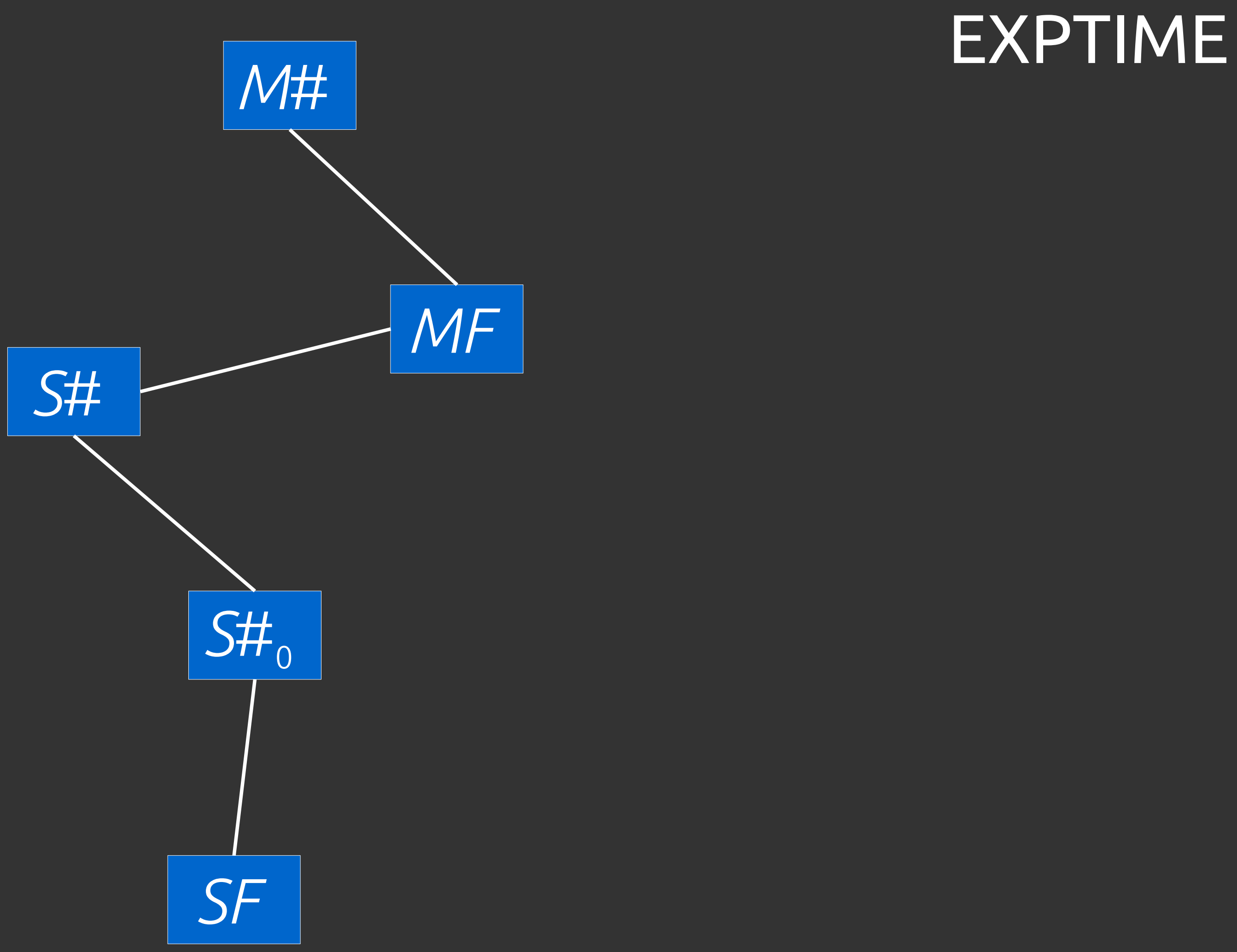
$$Q \longrightarrow Q \times (2R+2)^R$$

(bisimilarity for finite-state automata is in PTIME)

Complexity Picture

Multiplicity:
- (S) single
- (M) multiple

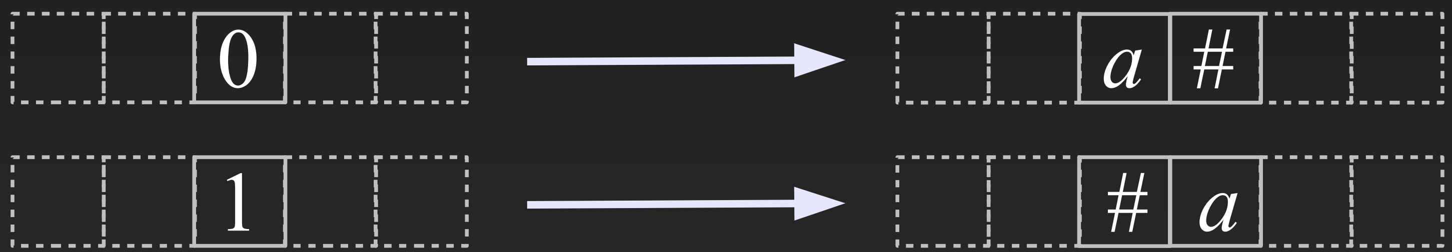
Fullness:
- (F) full
- ($\#_0$) initially empty
- ($\#$) eraseable



EXPTIME hardness

The ($S\#$) case is EXPTIME-hard:

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

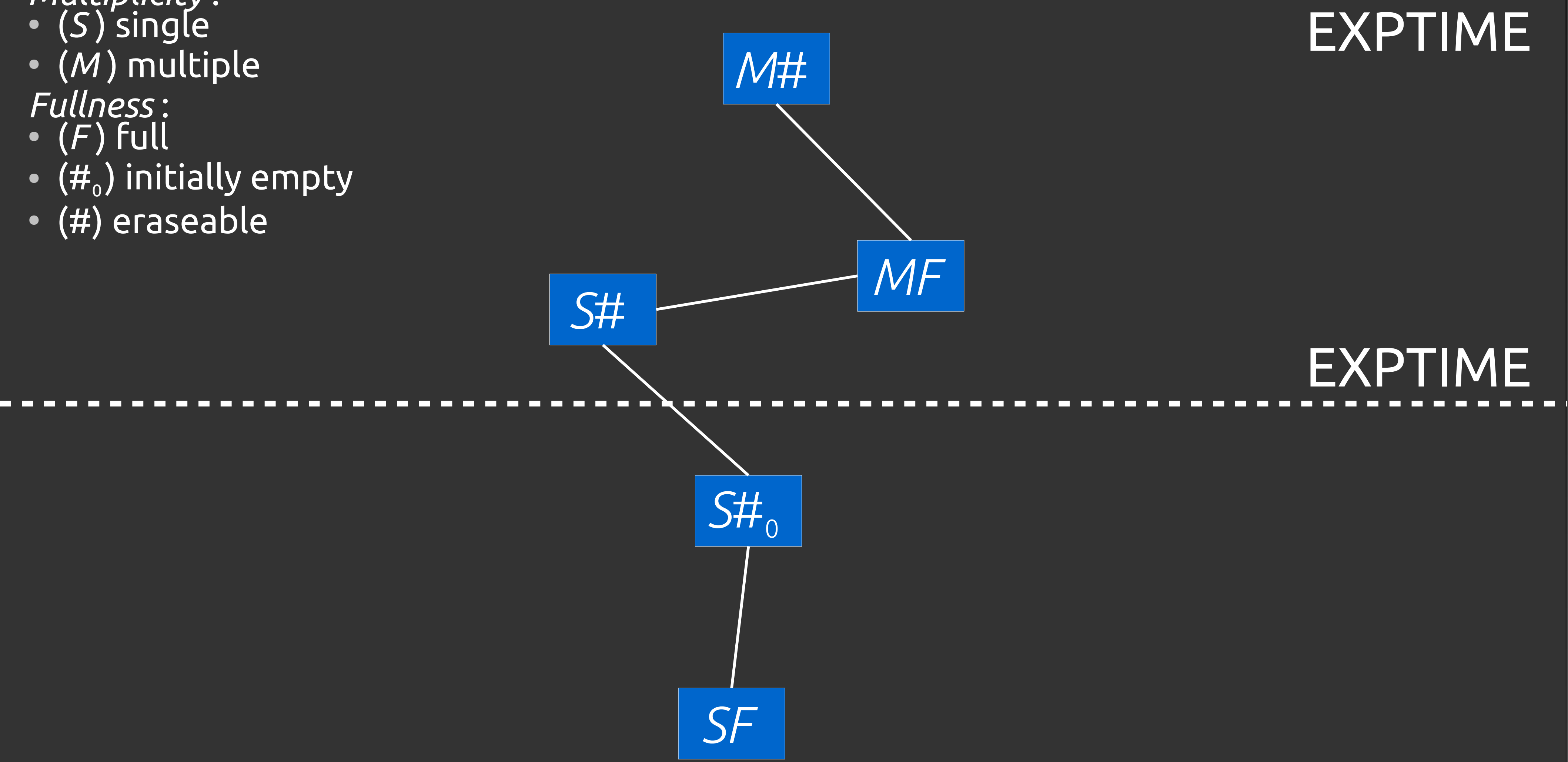


- arrange for non-bisimilarity at rejecting final states
- Bisimulation game (Attacker (A) vs Defender (D)):
  - A controls universal states, D controls existential ones
  - use *Defender forcing* [Jancar & Srba '08]

Complexity Picture

Multiplicity:
- (S) single
- (M) multiple

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



The original case ($S\#_0$)

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

In fact, the problem is PSPACE complete

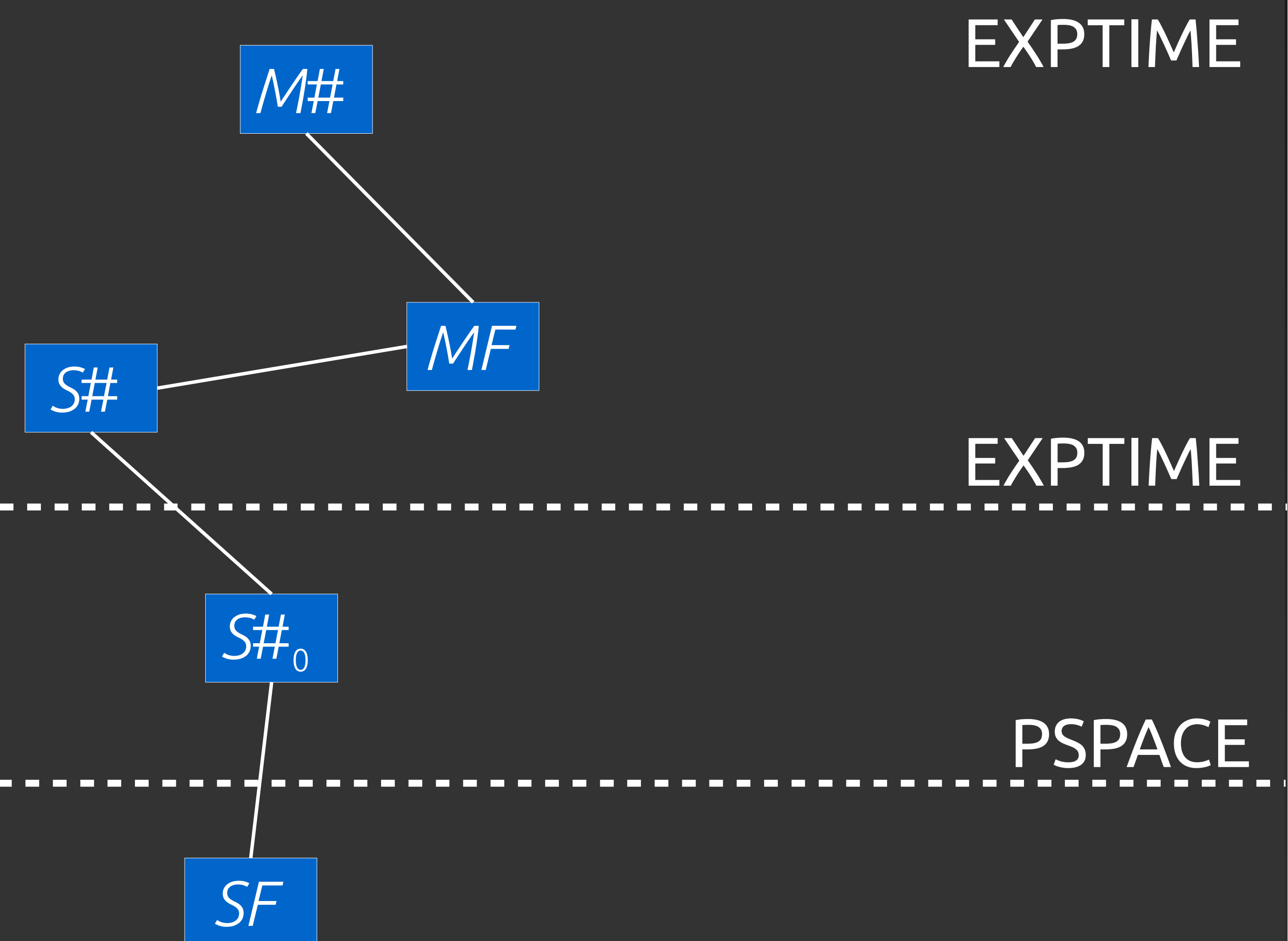
First, we can model boolean assignments (cf. write-once tape), which are enough for PSPACE-hardness:

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

Complexity Picture

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

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



PSPACE solvability: difficult

Our best bet is APTIME = PSPACE

- problem: while we cannot simulate a linear tape, we still have a lot of configurations!

We look into internal symmetries of FRAs:

- symbolic reasoning: we are only look at 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 $2R$

PSPACE solvability

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

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

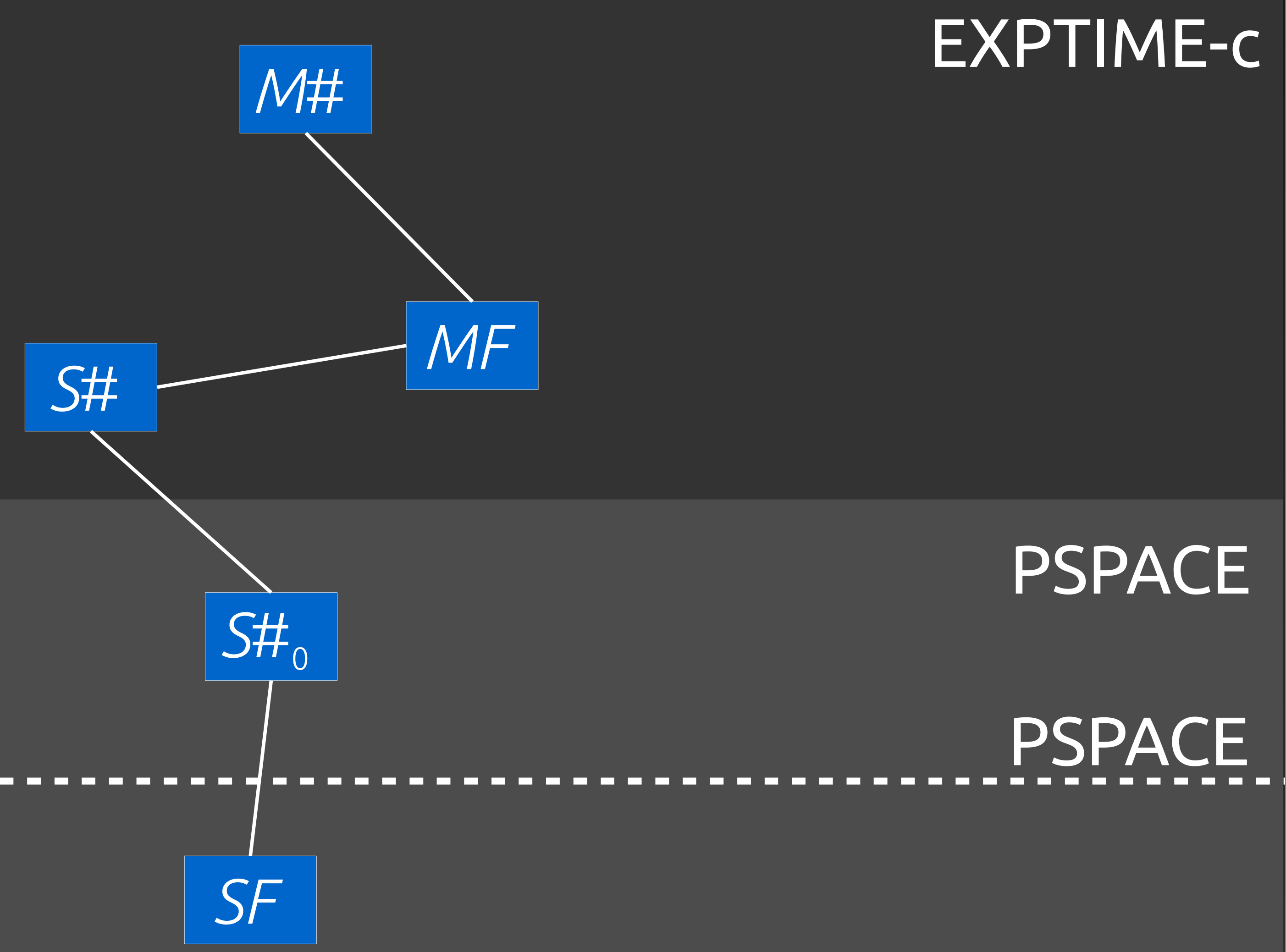
This means there is a final polynomial-size i

- polynomial bound for bisimulation game → APTIME

Complexity Picture

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

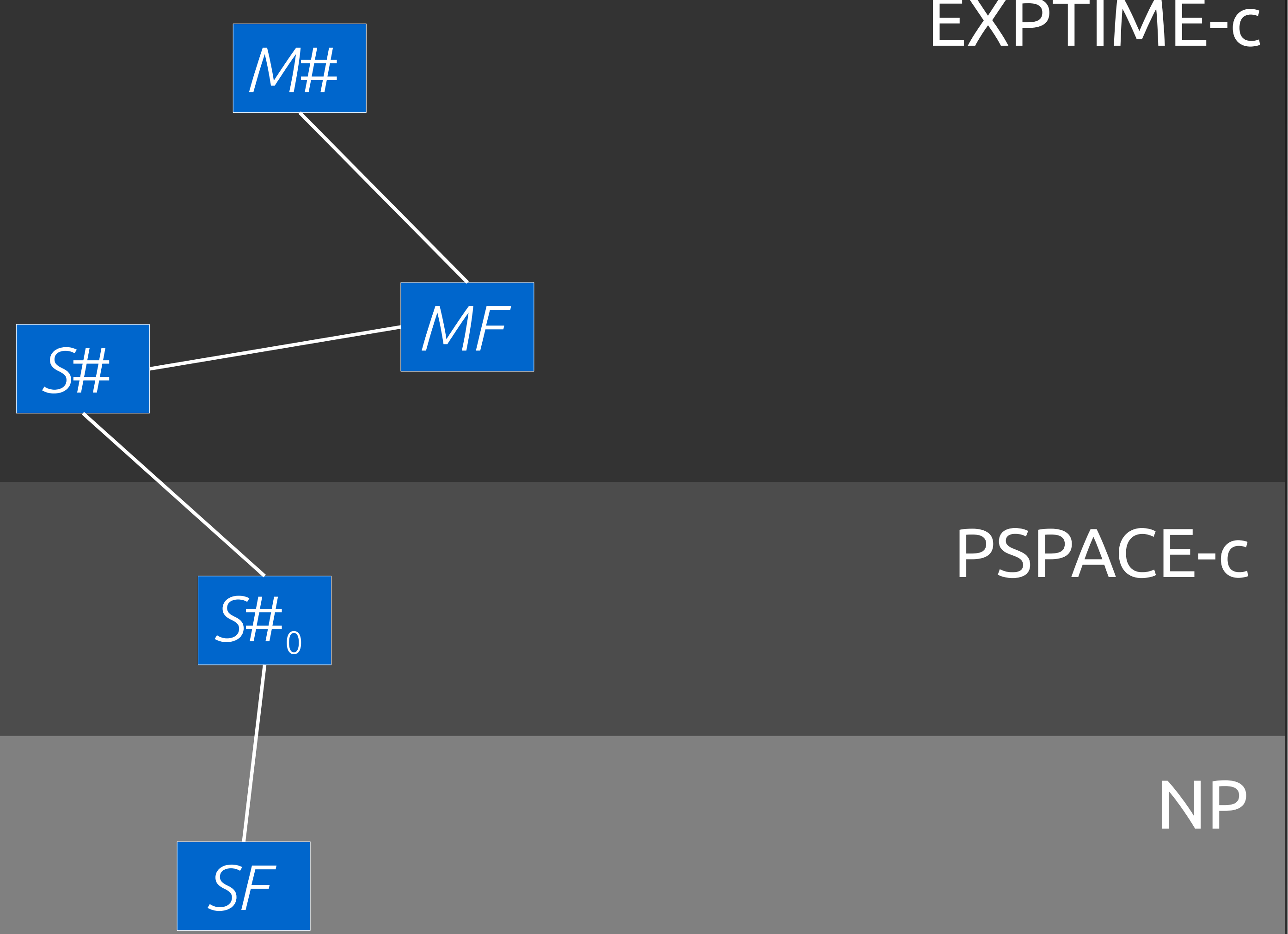
Fullness:
- (*F*) full
- ($\#_0$) initially empty
- (#) eraseable



Bisimilarity for (F)RAs

Multiplicity:
- (S) single
- (M) multiple

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



Investigations in FRAs

Bisimilarity for FRAs (complexity)

- Depends on register mode (NP → PSPACE → EXPTIME)
  - approach uses permutation group theory

[Murawski, Ramsay & T. '15]

Context-freeness: Pushdown FRA

[Cheng & Kaminski '98; Segoufin '06]
[Murawski & T. '12]

- Reachability EXPTIME-complete
- Global reachability via "saturation"

[Murawski, Ramsay & T. '14]

Freshness oracle: from one to many histories

- History Register Automata (cf. DA/CMA)

[Grigore & T. '13]

[Cheng & Kaminski '98, Segoufin '06; Murawski & T. 12]

Pushdown FRAs (FPDRAs)

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

$$q \xrightarrow{\lambda} q'$$

finitely many (say R) registers

pushdown stack

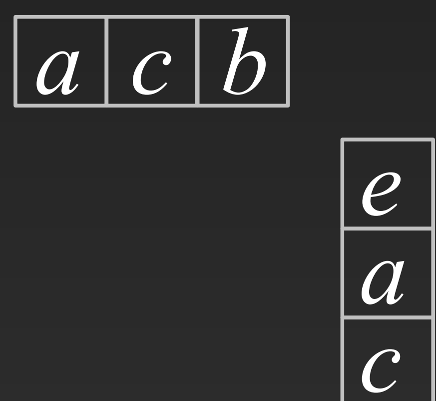
registers & stack store names

Label λ of the form:

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

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

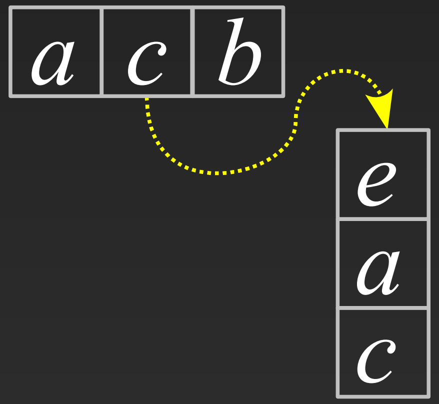
$$q \xrightarrow{\textbf{push}(2)} q'$$



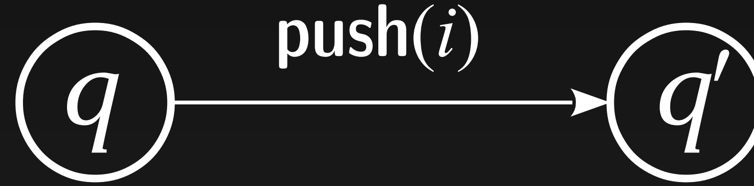
Transitions:

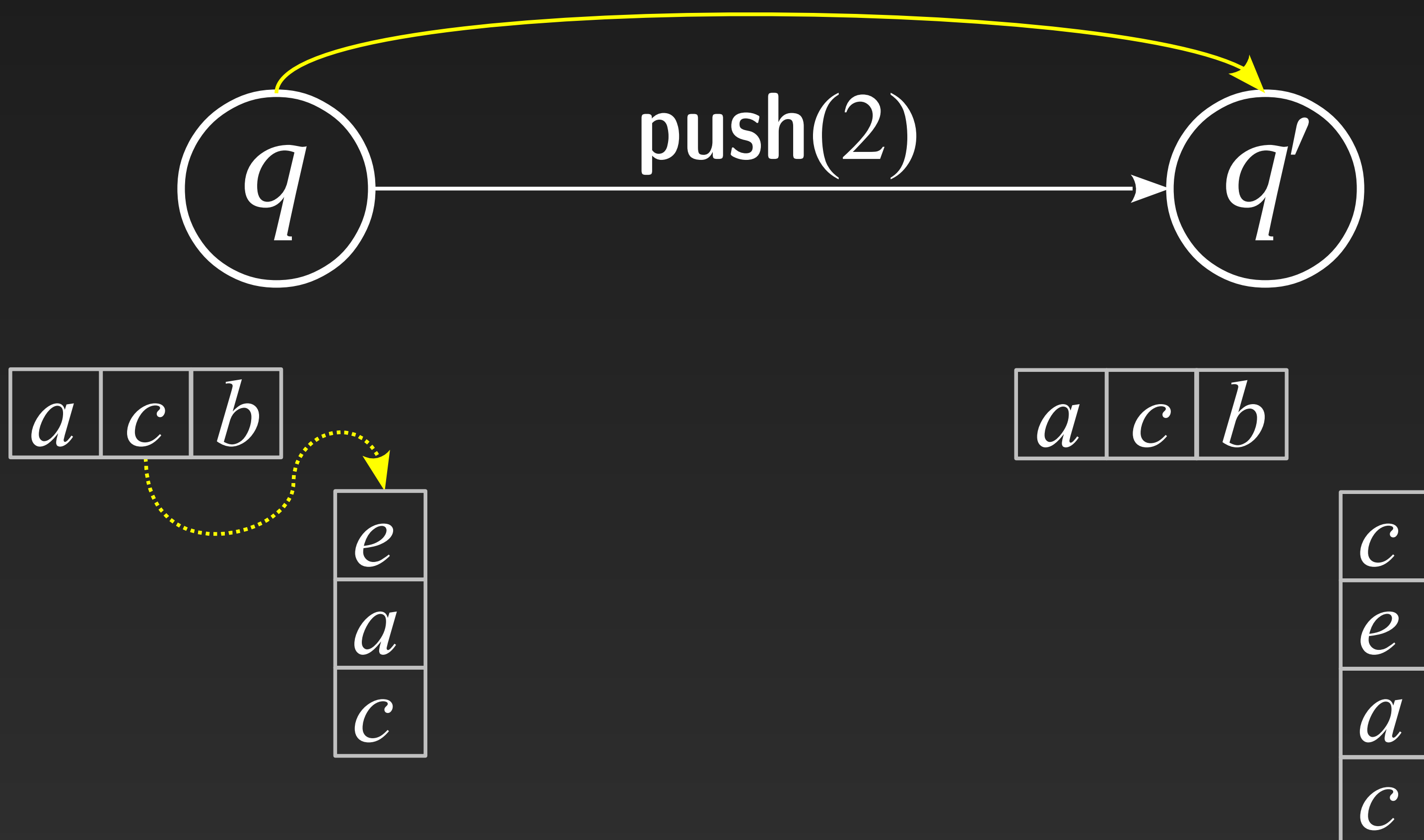
$$q \xrightarrow{\textbf{push}(i)} q'$$

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

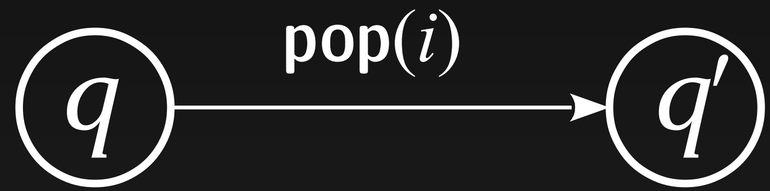


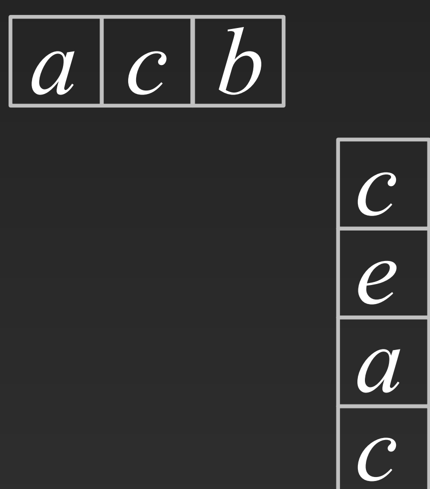
Transitions:





Transitions:

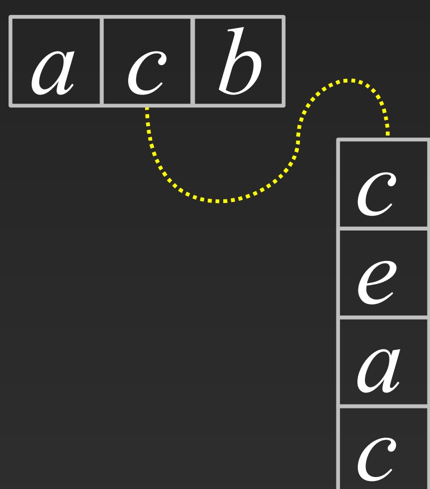




Transitions:

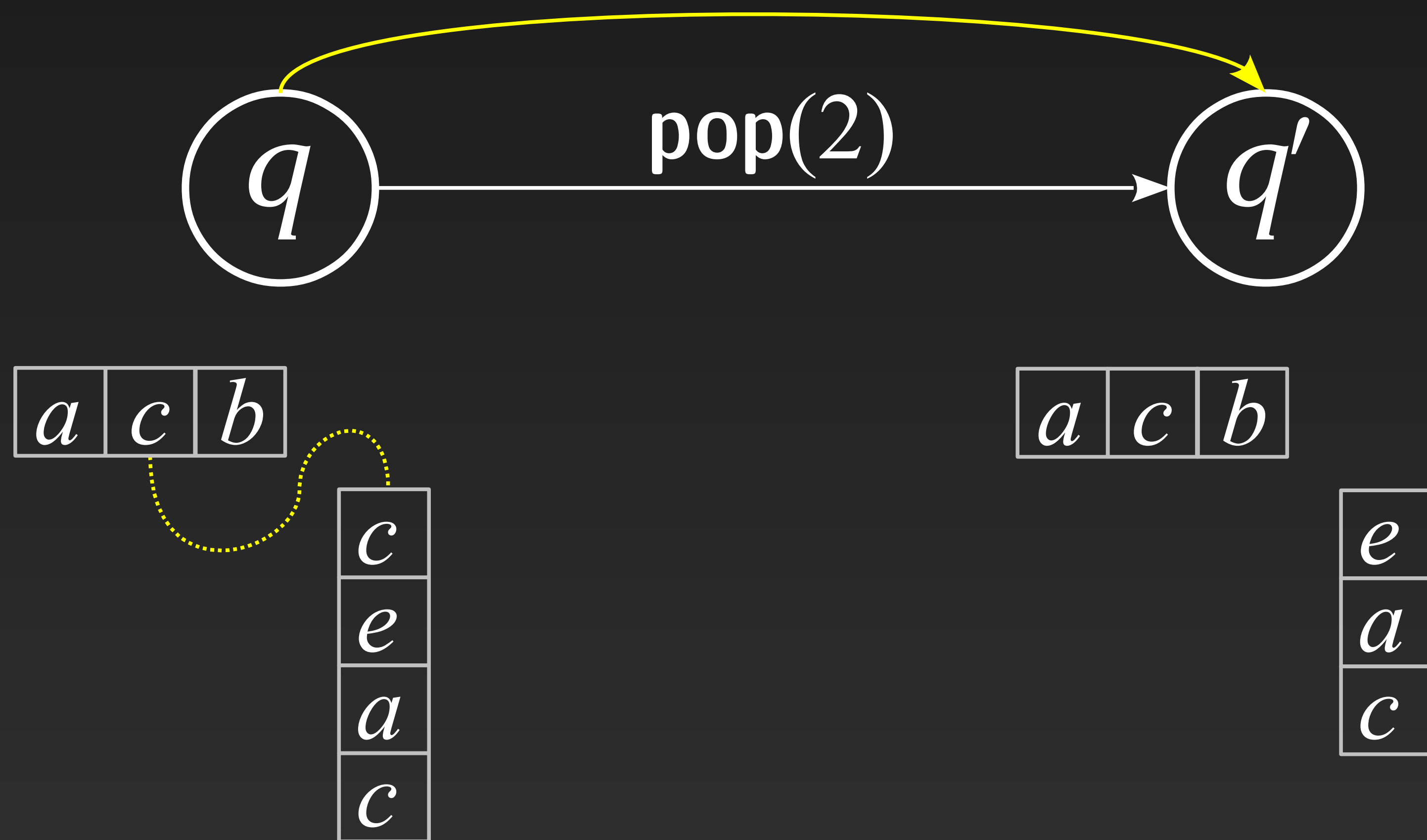
$$q \xrightarrow{\mathbf{pop}(i)} q'$$

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

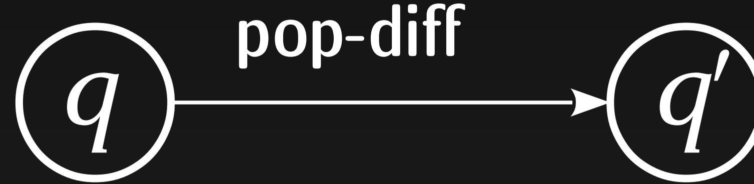


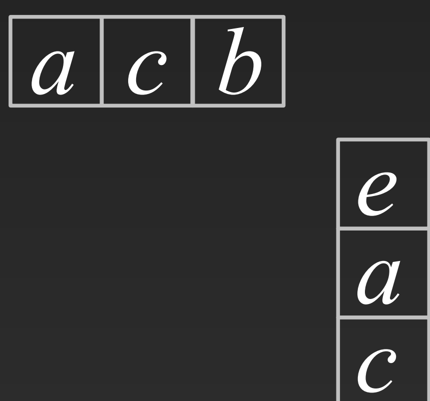
Transitions:

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

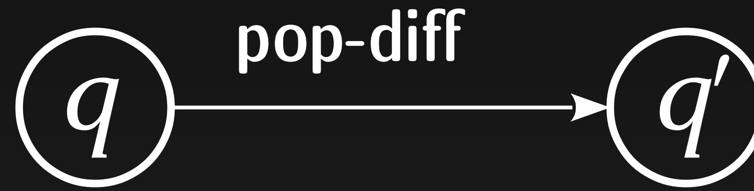


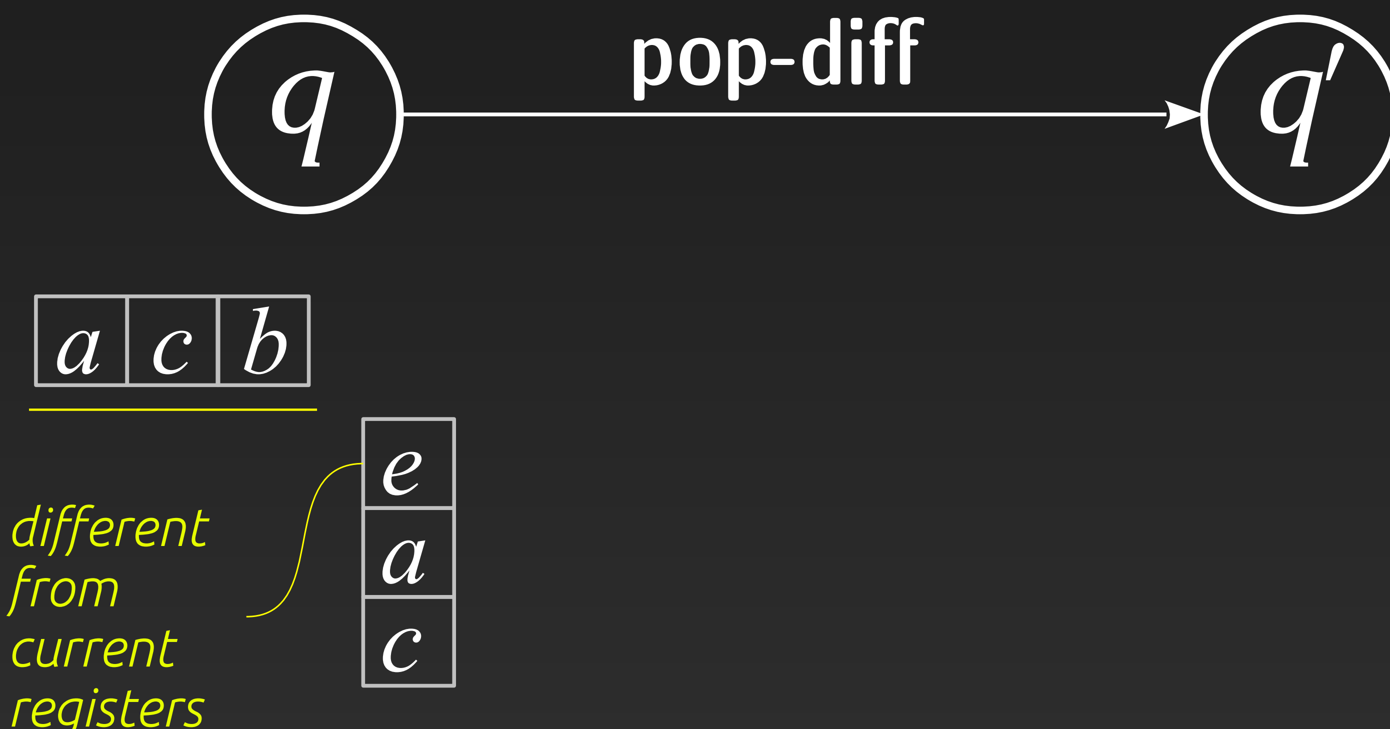
Transitions:



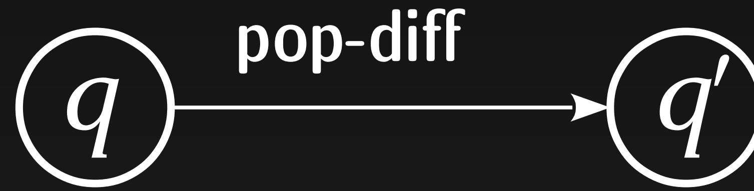


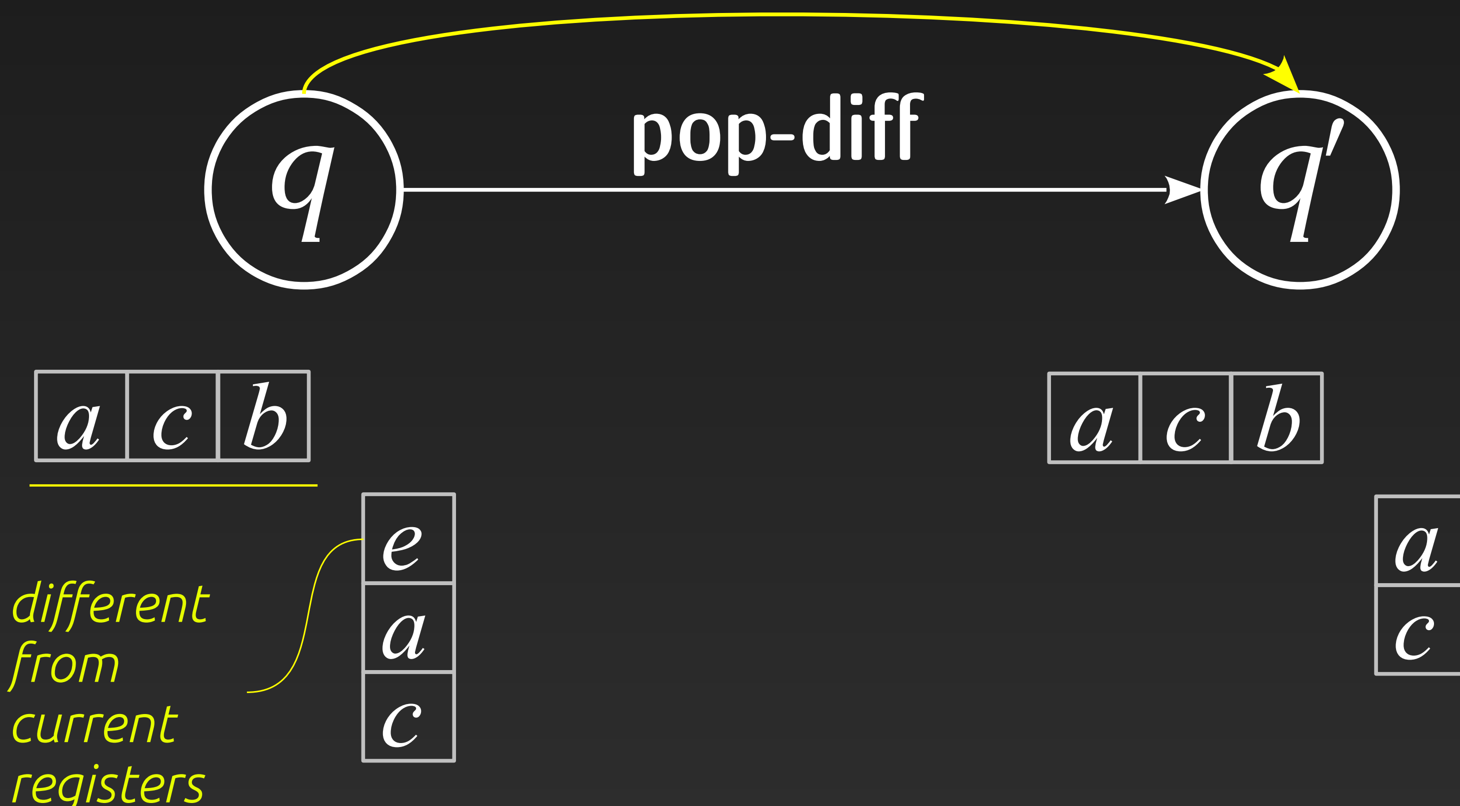
Transitions:





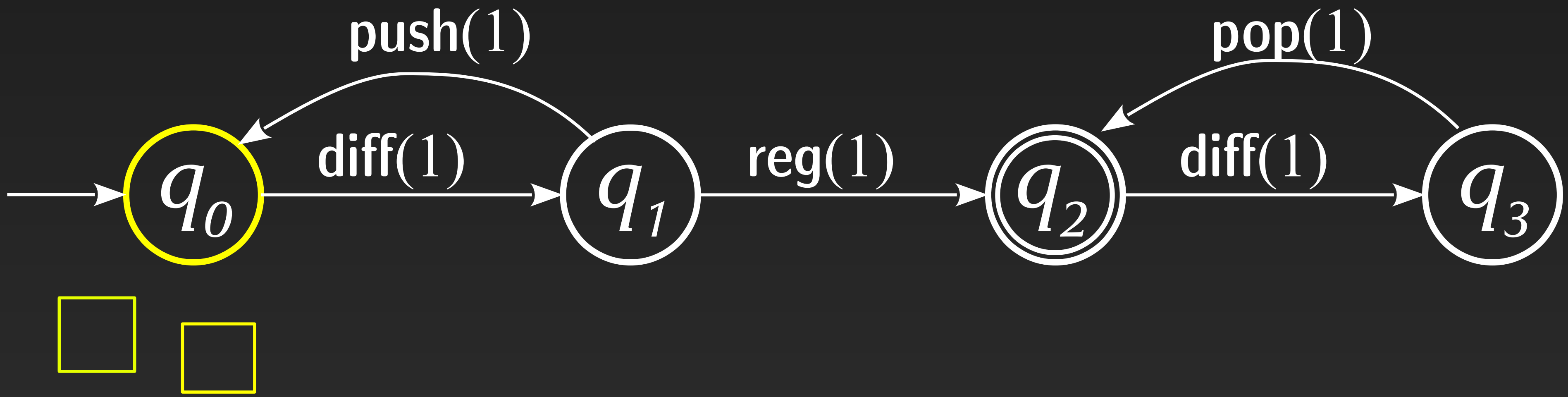
Transitions:





Example

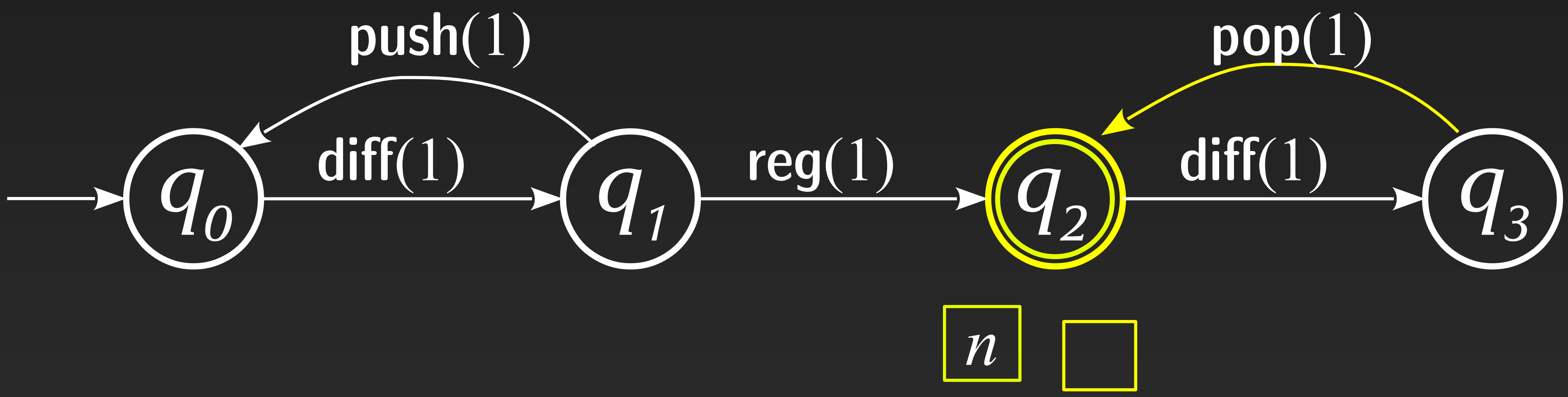
$$L_4 = \{\ a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \in \Sigma^* \mid n \geq 0,\ \forall i < n.\ a_i \neq a_{i+1} \}$$



Example

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

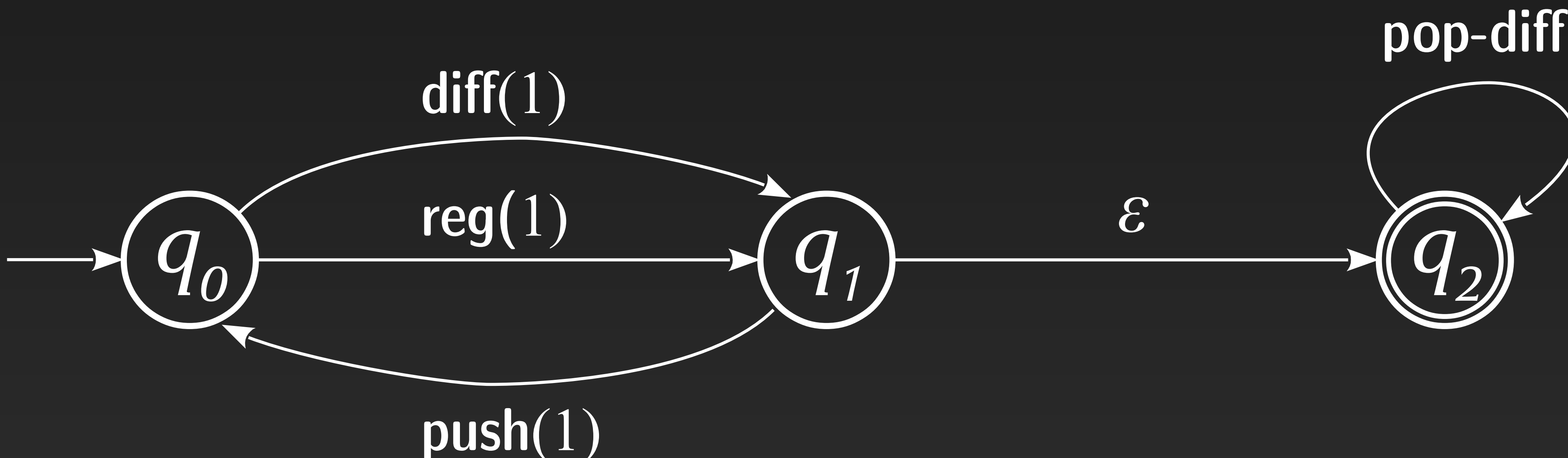
n e v e r o d d o r e v e n



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

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

(all strings of distinct names)

Limited distinguish-ability

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

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

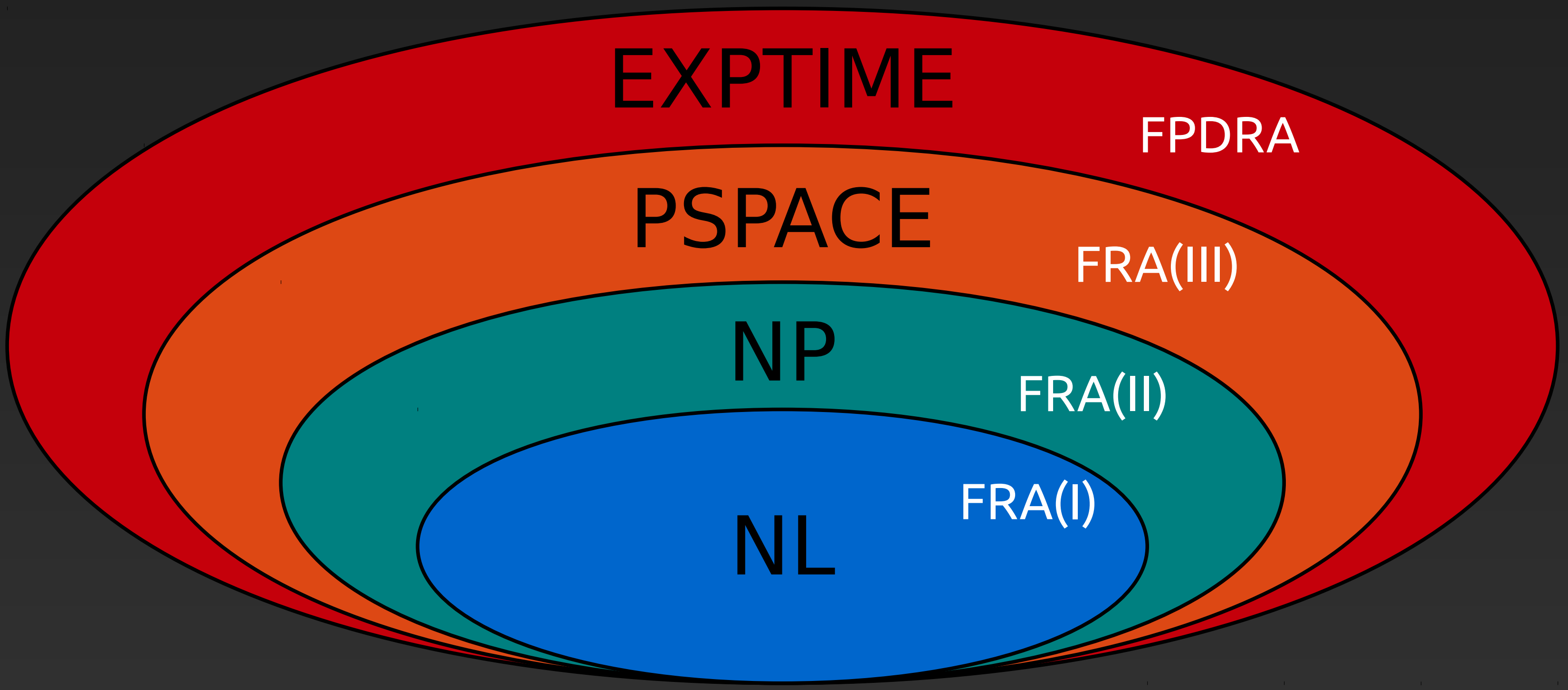
$3R$ property:
Given a PDRA (no **fresh**) with R registers with states q_1, q_2, any run between them (from empty stack to empty stack) can be taken with at most $3R$ names.

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

Reachability/non-emptiness for (F)RAs

→ *R*-FPRDA Reachability is EXPTIME-complete

- upper bound by $3R$ + *freshness simulation* [Murawski & T. 12]
- hardness by reduction from TMs with stack (SF, no **fresh**)



Investigations in FRAs

Bisimilarity for FRAs (complexity)

- Depends on register mode (NP → PSPACE → EXPTIME)
  - approach uses permutation group theory [Murawski, Ramsay & T. '15]

Context-freeness: Pushdown FRA

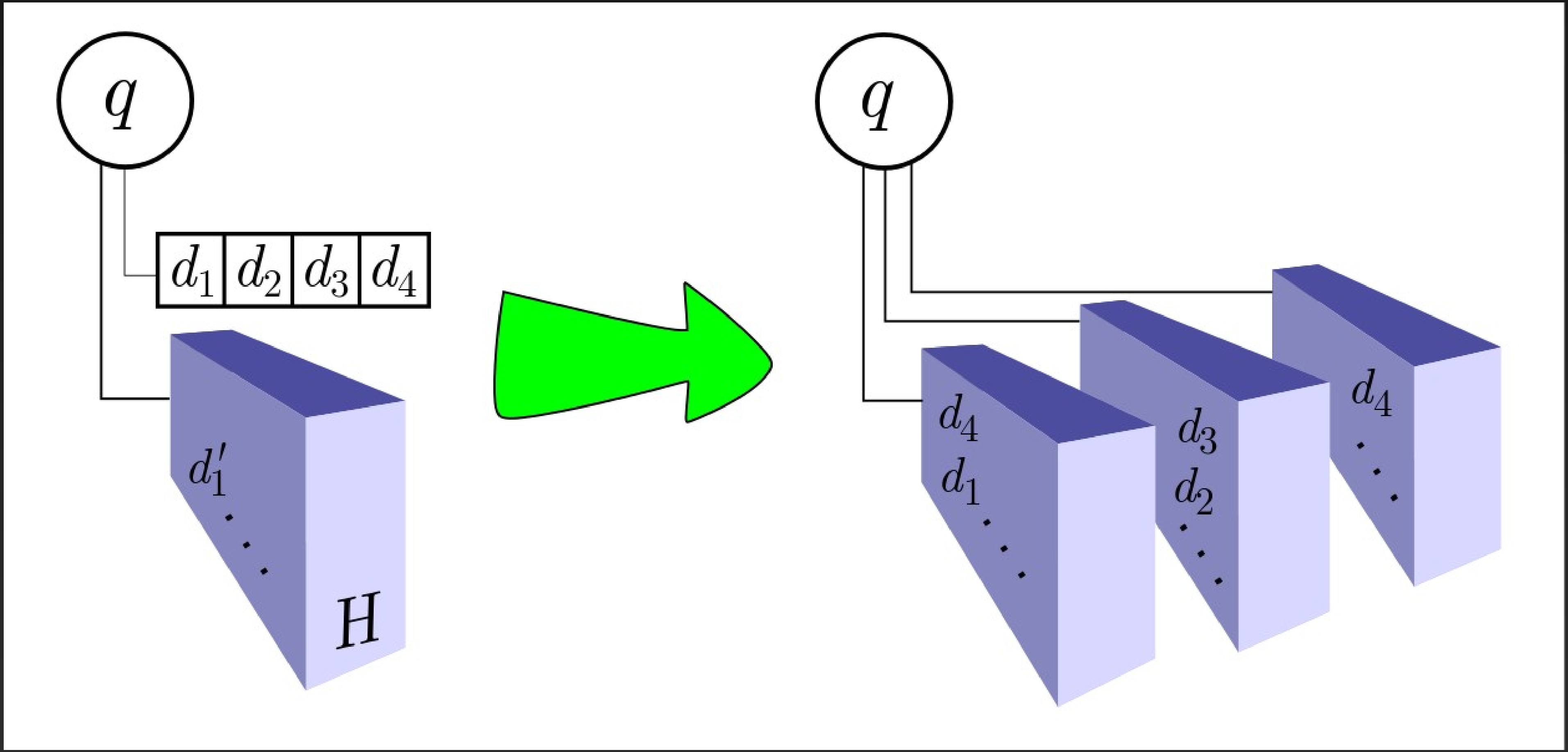
[Cheng & Kaminski '98; Segoufin '06]
[Murawski & T. '12]

- Reachability EXPTIME-complete
- Global reachability via "saturation" [Murawski, Ramsay & T. '14]

Freshness oracle: from one to many histories

- History Register Automata (cf. DA/CMA) [Grigore & T. '13]

HRAs: from registers to histories



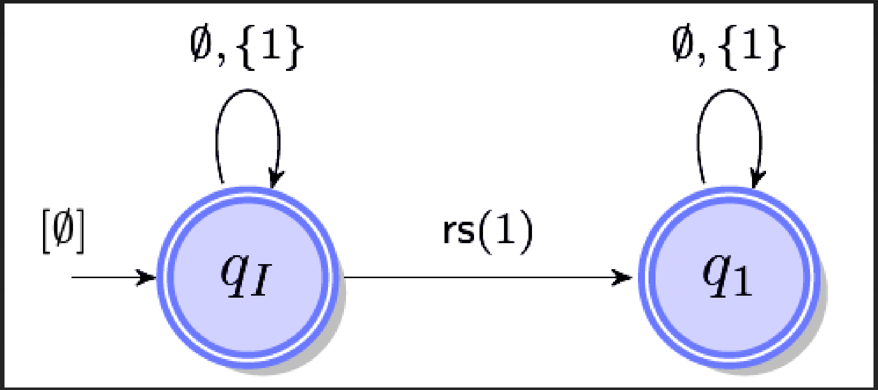
histories = registers with unboundedly many equivalent elements

Expressivity

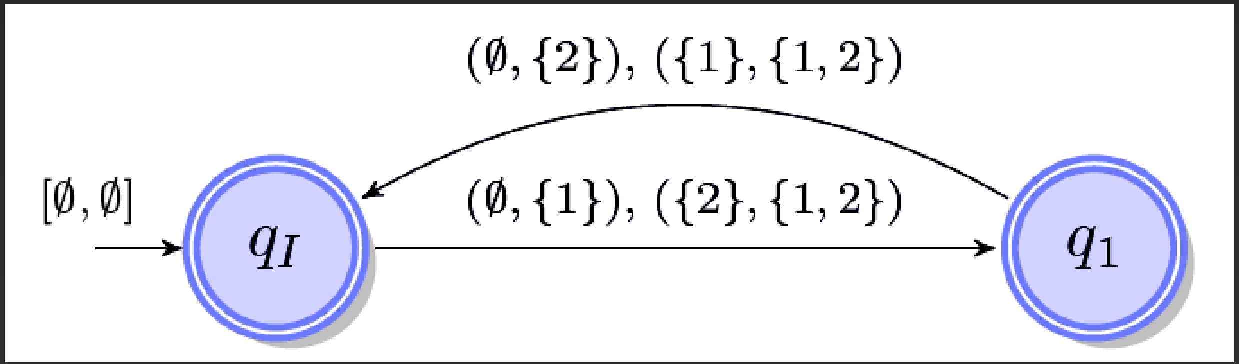
Histories simulate registers → HRAs extend FRAs

Histories can be reset:

→ Closure under Kleene* and concatenation



Several histories → Closure wrt interleavings



HRA properties

- Cleanly extend RAs and FRAs
- Closed under all regular operations apart from complementation
- Closed under interleaving
- Universality undecidable (from RAs)
- Emptiness decidable, non-primitive recursive complexity (~transfer/reset Petri nets)
- Closely related to Data / Class Memory Automata

[Bojańczyk, Muscholl, Schwentick, Segoufin & David '06 , Bjorklund & Schwentick '10]

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

Further on:

- open/working problems: automata learning, regular expressions, infinite words, verification logics
- algorithm implementations (an FRA toolkit!)