History-Register Automata

Radu Grigore
University of Kent

Nikos Tzevelekos
Queen Mary Uni. of London

DIMAP seminar, Warwick, 26 Jan 2016

Supported by an RAEng research fellowship

What this talk is about

It is about automata theory over infinite alphabets

used for modelling programs which use dynamic resources (*names*)

History-Register Automata :
a class of such automata capturing fresh name generation, consumption, interleavings, etc.

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();
}
```

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

– but in a *parametric way*

Lots of interest also from XML model-checking community

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

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

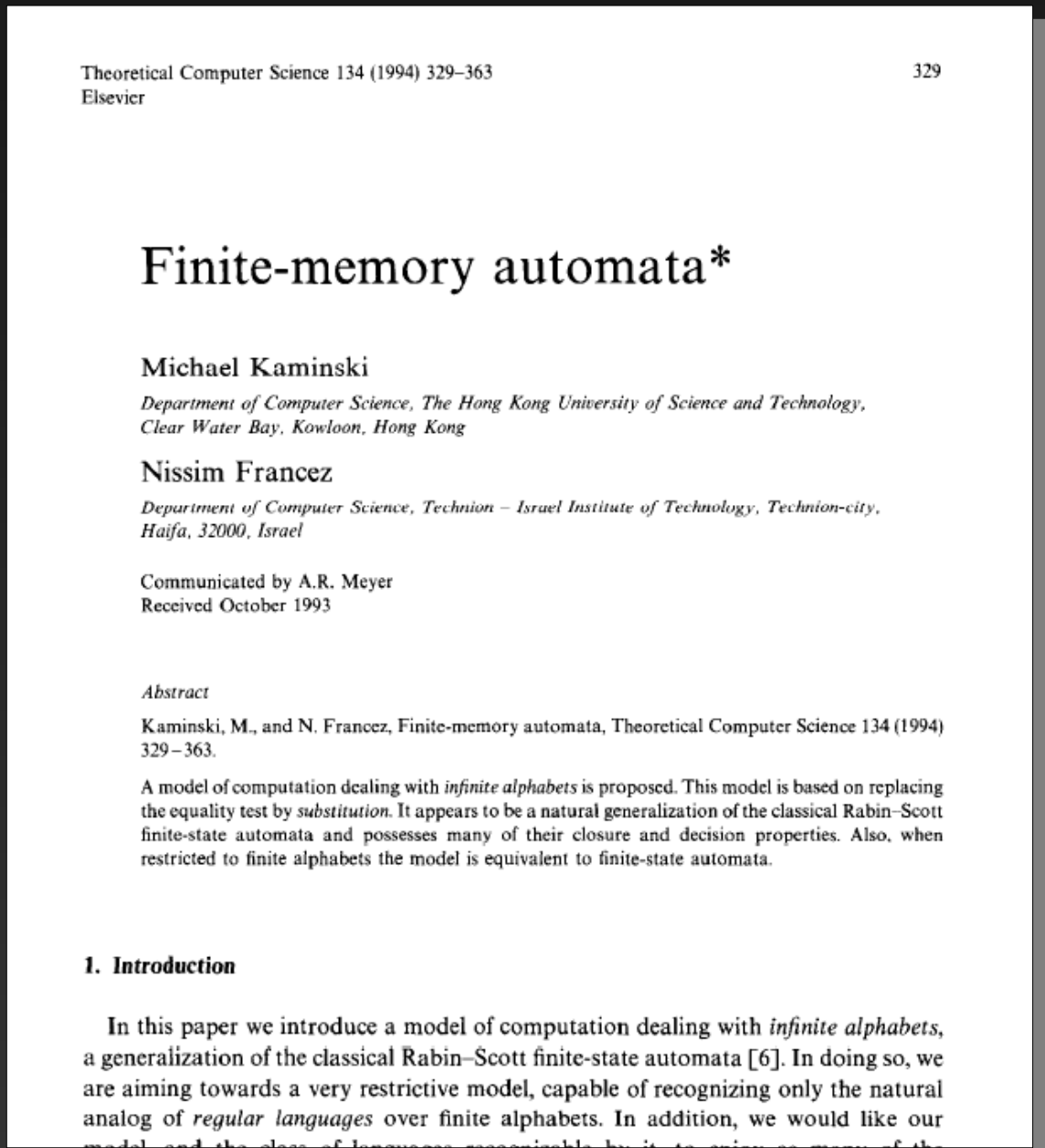
a finite set of constants

- look for notions of regularity, CFGs, etc.
- devise effective algorithms for reachability, membership, etc.
- use e.g. for modelling and verifying code

[Kaminski & Francez '94]

Register Automata (RA)

- finite state automata
- operating on an infinite alphabet of inputs
- and using a finite amount of register memory



Theoretical Computer Science 134 (1994) 329–363
Elsevier

Finite-memory automata*

Michael Kaminski
Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong

Nissim Francez
Department of Computer Science, Technion – Israel Institute of Technology, Technion-city, Haifa, 32000, Israel

Communicated by A.R. Meyer
Received October 1993

Abstract

Kaminski, M., and N. Francez, Finite-memory automata, Theoretical Computer Science 134 (1994) 329–363.

A model of computation dealing with *infinite alphabets* is proposed. This model is based on replacing the equality test by *substitution*. It appears to be a natural generalization of the classical Rabin–Scott finite-state automata and possesses many of their closure and decision properties. Also, when restricted to finite alphabets the model is equivalent to finite-state automata.

1. Introduction

In this paper we introduce a model of computation dealing with *infinite alphabets*, a generalization of the classical Rabin–Scott finite-state automata [6]. In doing so, we are aiming towards a very restrictive model, capable of recognizing only the natural analog of *regular languages* over finite alphabets. In addition, we would like our

[Kaminski & Francez '94]

Register Automata (RA)

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

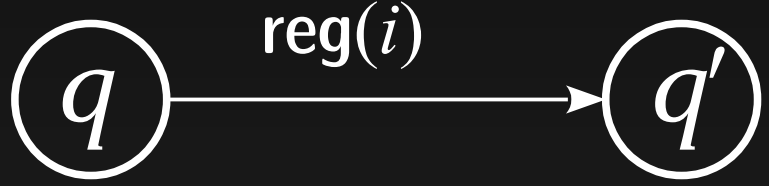
finitely many (say R) registers

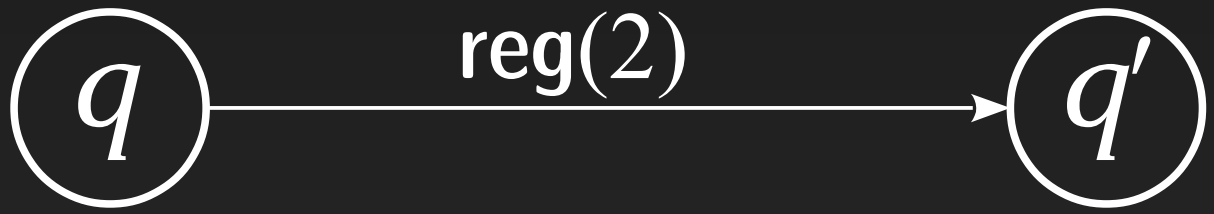
registers store names

Label λ of the form:

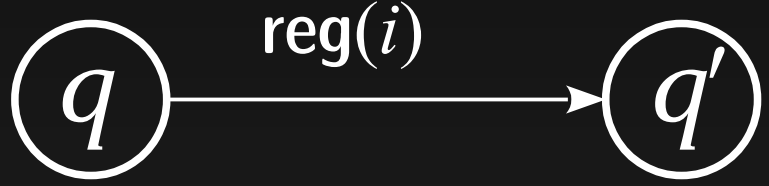
- **reg**(i), $i \in \{1,\ldots,R\}$
- **diff**(i), $i \in \{1,\ldots,R\}$

Transitions:



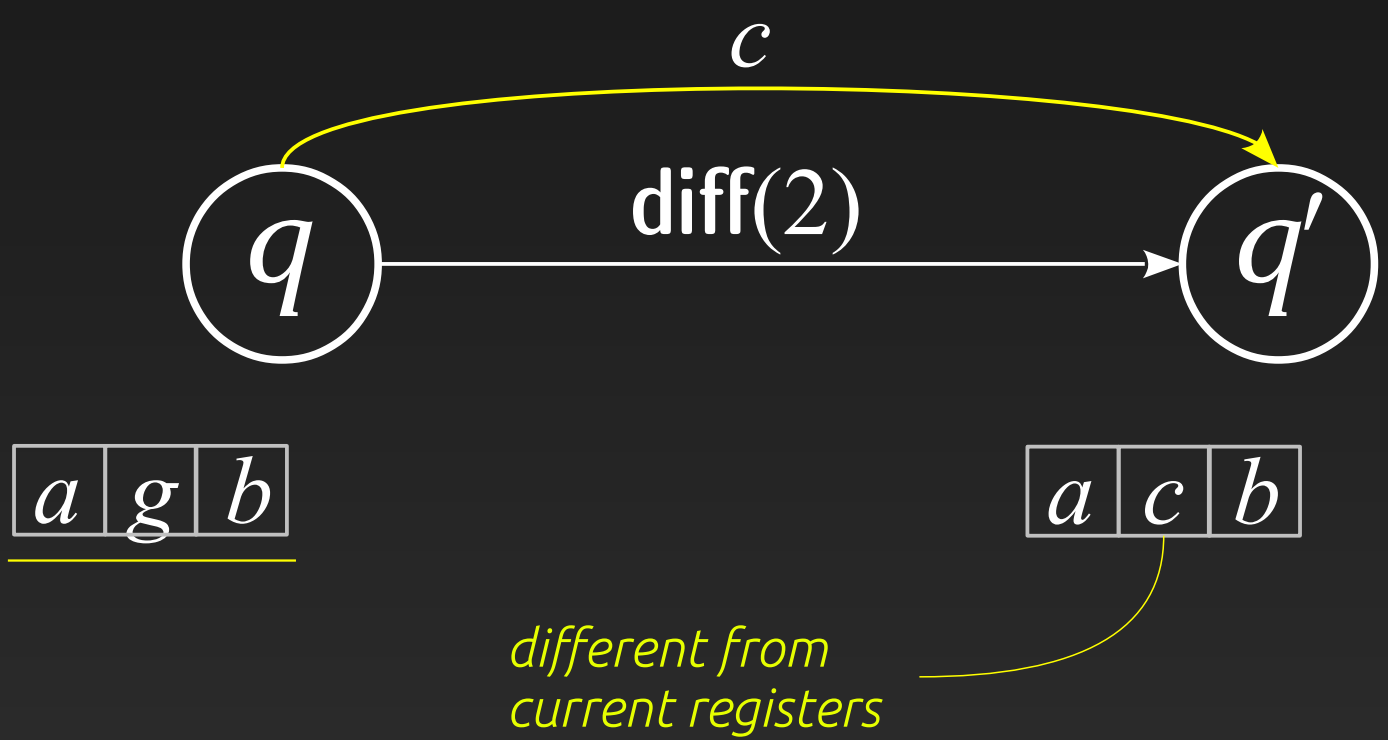


Transitions:



Transitions:

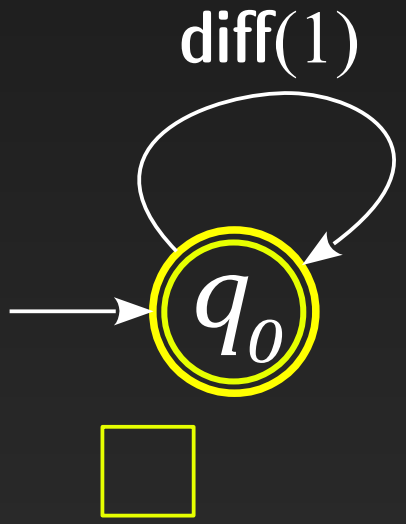
Transitions:



Example

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

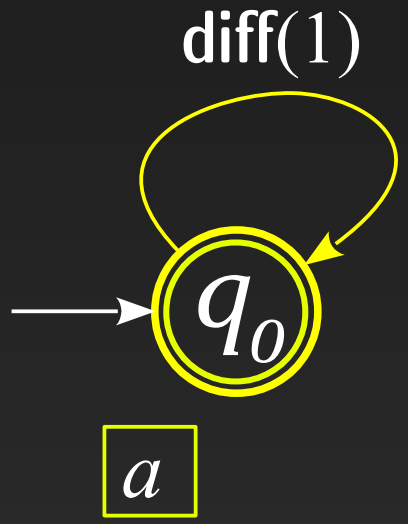
(all strings where each name is distinct from its predecessor)



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)

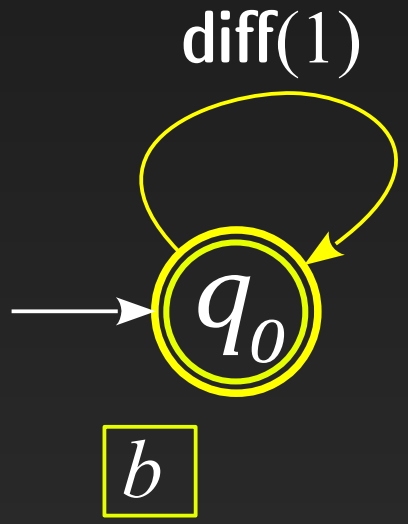


a

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)

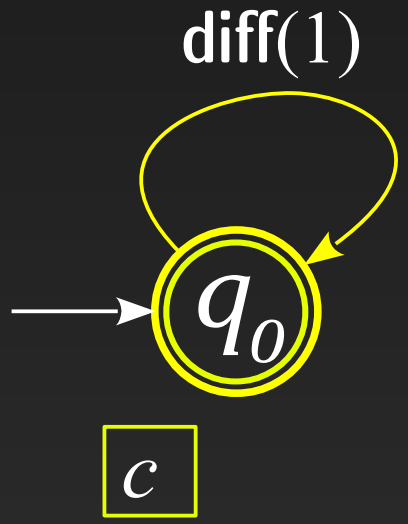


$a\,b$

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)

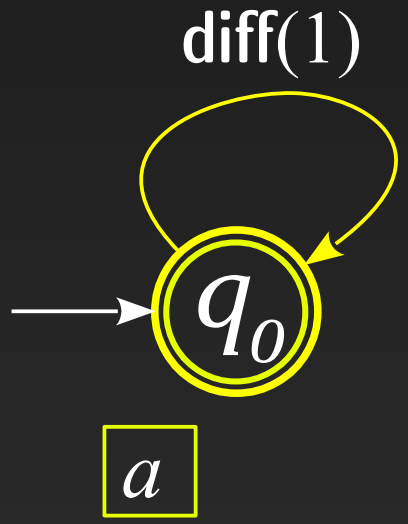


$a\,b\,c$

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)

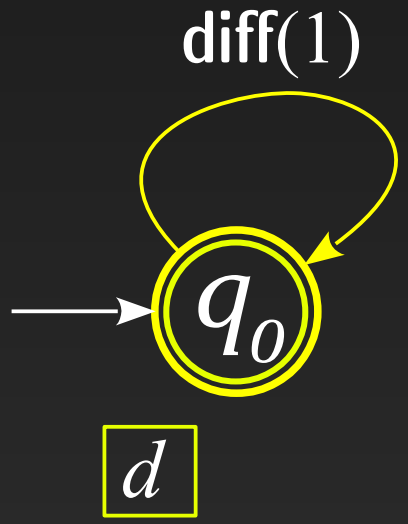


$a\,b\,c\,a$

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)

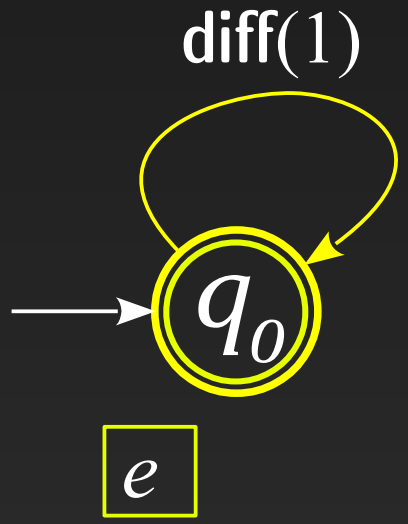


$a\,b\,c\,a\,d$

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)

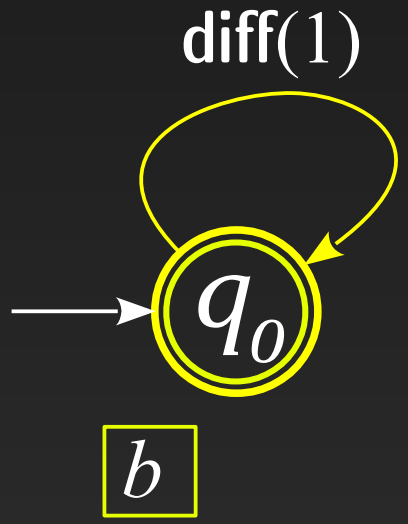


$a\,b\,c\,a\,d\,e$

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)

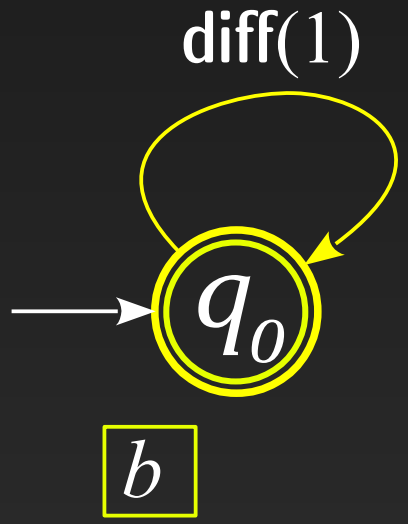


$a\,b\,c\,a\,d\,e\,b$

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)

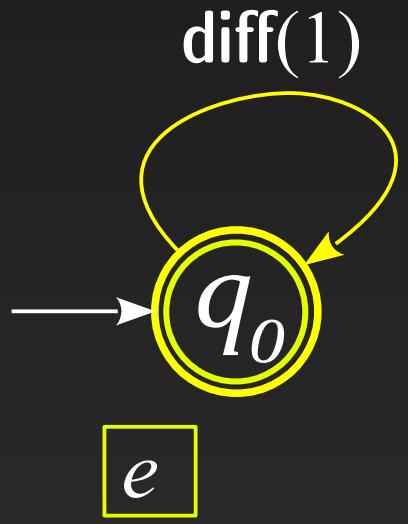


$$a\,b\,c\,a\,d\,e\,b\,a\,g\,c\,a\,b$$

Example

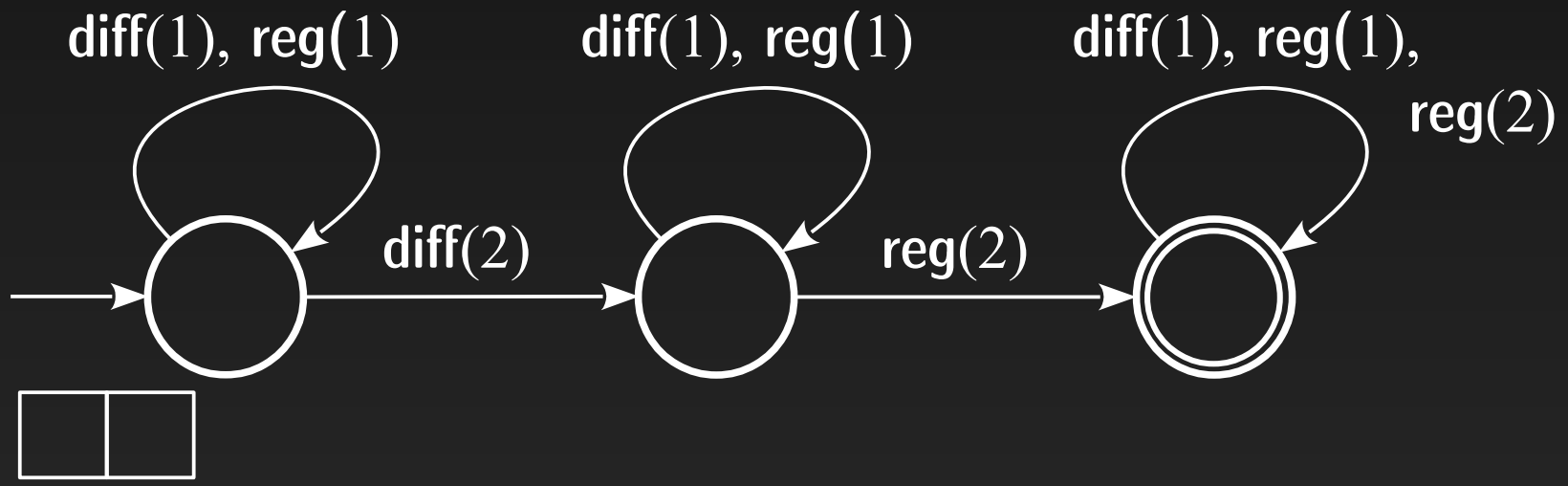
$$L_1 = \{\, a_1 a_2 \ldots a_n \in \Sigma^* \mid n \geq 0, \forall i < n.\ a_i \neq a_{i+1} \,\}$$

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

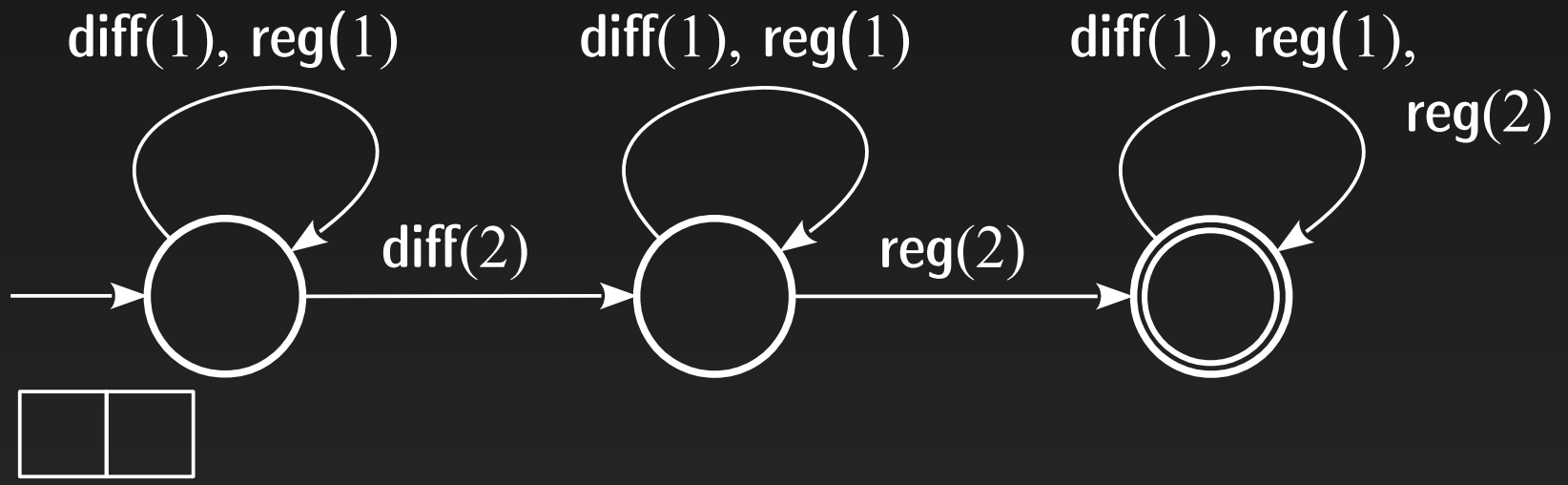


$$a\,b\,c\,a\,d\,e\,b\,a\,g\,c\,a\,b\ a\,n\,d\ w\,e\ l\,o\,v\,e\ c\,a\,k\,e$$

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)

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

(all strings of pairwise distinct names)

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

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

  → cannot capture *freshness*, i.e. name creation

[T. '11]

Fresh-Register Automata (FRA)

Finite state automata
with finitely many
registers
and a freshness oracle

POPL'11

Fresh-Register Automata

Nikos Tzevelekos
Oxford University Computing Laboratory
nikt@comlab.ox.ac.uk

Abstract

What is a basic automata-theoretic model of computation with names and fresh-name generation? We introduce Fresh-Register Automata (FRA), a new class of automata which operate on an infinite alphabet of names and use a finite number of registers to store fresh names, and to compare incoming names with previously stored ones. These finite machines extend Kaminski and Francez's Finite-Memory Automata by being able to recognise globally fresh inputs, that is, names fresh in the whole current run. We examine the expressivity of FRA's both from the aspect of accepted languages and of bisimulation equivalence. We establish primary properties and connections between automata of this kind, and answer key decidability questions. As a demonstrating example, we express the theory of the pi-calculus in FRA's and characterise bisimulation equivalence by an appropriate, and decidable in the finitary case, notion in these automata.

Categories and Subject Descriptors F.1.1 [*Computation by Abstract Devices*]: Models of Computation; D.3.1 [*Programming Languages*]: Formal Definitions and Theory—Semantics

General Terms Theory, Languages, Verification

1. Introduction

One of the most common and useful abstractions in programming is the assumption that entities of specific kinds can be created at will and, moreover, in such a manner that newly created entities are always *fresh* — distinct from any other such created thus far. This is, for example, the case with mutable reference cells, exceptions user-declared datatypes, *etc.* in languages like Standard ML [15]. Following a long tradition in computer science [20], we call these entities *names* and specify them as follows.

> Names can be created fresh dynamically and locally, compared for equality and communicated between agents or subroutines.

Apart from the uses mentioned above, names form the basis of calculi of mobile processes (e.g. the π-calculus [14]); appear in network protocols and secure transactions; and are generally essential in programming for identifying variables, channels, threads, objects, codes, and many other sorts of name in disguise. To our knowledge, there has not been in the literature a proposal of a basic automata-theoretic model of names, providing abstract machines

Our model is based on the successful paradigm of ***Finite-Memory Automata (FMA)***, introduced by Kaminski and Francez in the early 90's [11]. Motivated by real-world problems (where codes, addresses, identifiers, *etc.* may have unbounded domains), those automata address a demand for a "natural" finite-state machine model over infinite alphabets. An FMA $\mathcal{A}$ is an automaton attached with a finite number of name-storing registers. Its structure looks identical to that of an ordinary finite-state automaton over a finite set of labels generated by indices in the range $1, \ldots, n$, where n is the number of registers. However, $\mathcal{A}$ truly operates on the infinite set of inputs $\mathbb{A}$ (the set of names), with indices i referring to the names stored in the i-th register of $\mathcal{A}$. This simple idea lifts the automaton from finite to infinite alphabet.

There are two ways in which an FMA can access its registers: either by comparing an input name to a stored one, or by storing an input name in one of its registers but only in case it is *locally fresh*, that is, it does not already appear in any of them. Thus, FMA's are history-free: their computational steps rely solely on their current registers. Here we introduce ***Fresh-Register Automata (FRA)***, a finite-register automaton model which extends FMA's by *global freshness* recognition: an automaton can now accept (and store) an input name just in case it is fresh in the whole run. For example, a transition

$$q \xrightarrow{i^{\textcircled{\#}}} q'$$

means that if $\mathcal{A}$ is at state q and the set of names that have appeared in its registers so far is H, then $\mathcal{A}$ can accept any name $a \notin H$, store it in its i-th register and proceed to q'. This history-sensitive feature precisely captures fresh-name creation.[1] Thus, e.g. the following language (not recognised by FMA's [11]) is recognised by a single-state FRA with one register.

$$\mathcal{L}_1 = \{\, a_1 \cdots a_k \in \mathbb{A}^* \mid \forall i \neq j.\, a_i \neq a_j \,\}$$

An intuitive way to view $\mathcal{L}_1$ is as the trace of a fresh-name generator: one which returns reference cells in SML, objects in Java, memory addresses in C, *etc.*

Research in FMA's and their formal languages has been extensive [2, 6, 11, 21, 25, 27]. It has been shown [11, 21] that FMA-recognisable languages are closed under union, intersection, concatenation and Kleene star; they are not closed under complement; emptiness of FMA's is decidable; and universality is undecidable. Our first contribution is to answer this series of questions

[T. '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

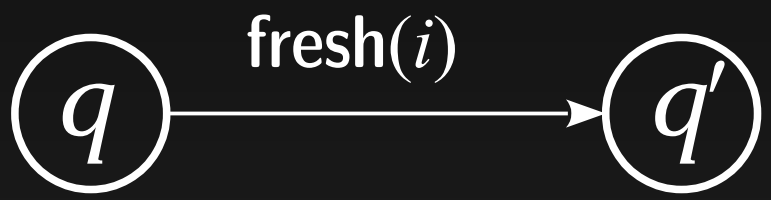
registers store names

Label λ of the form:

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

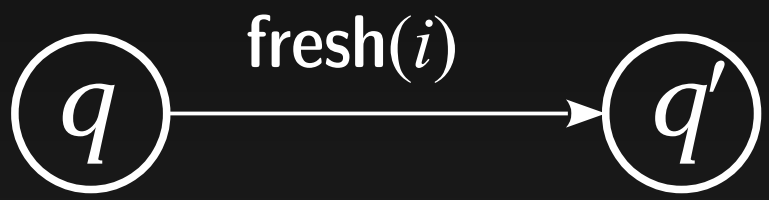
global freshness oracle

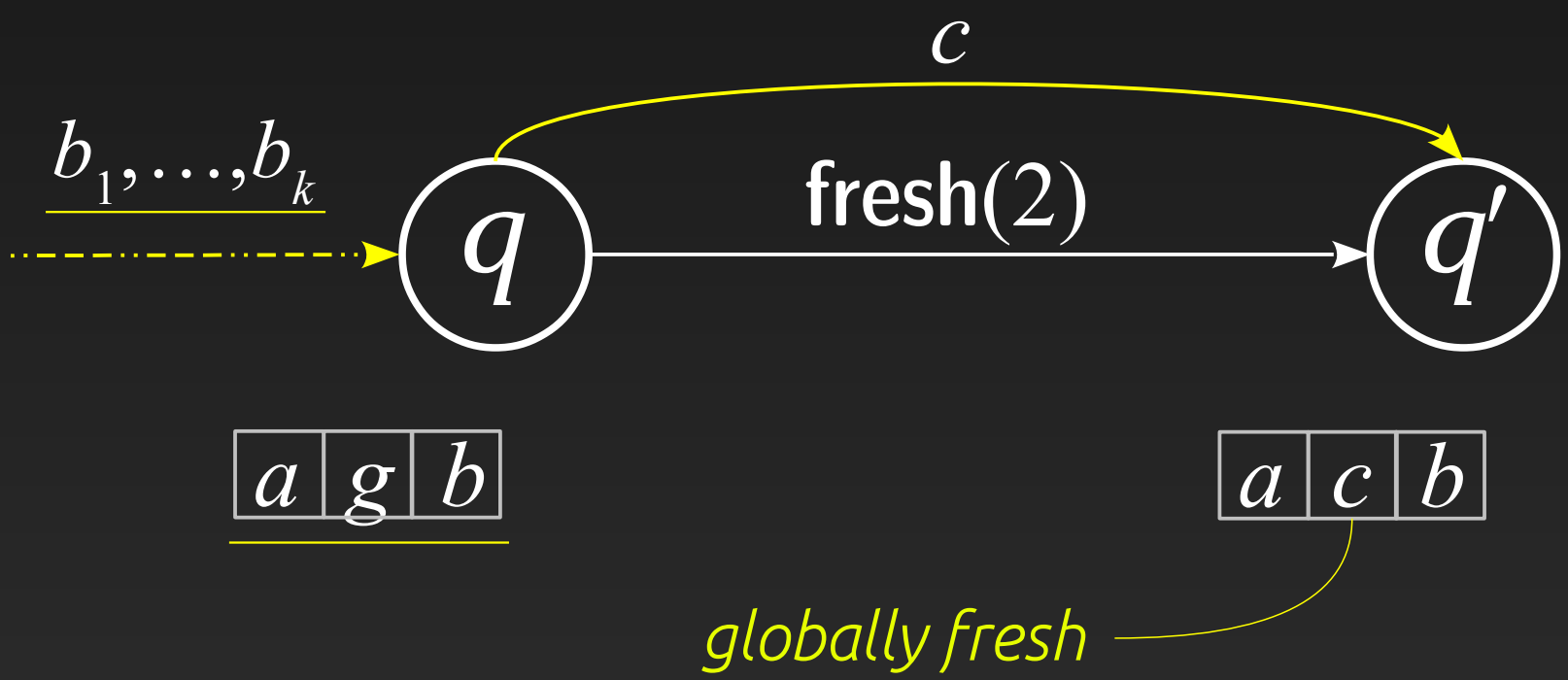
Transitions:



a	g	b

Transitions:

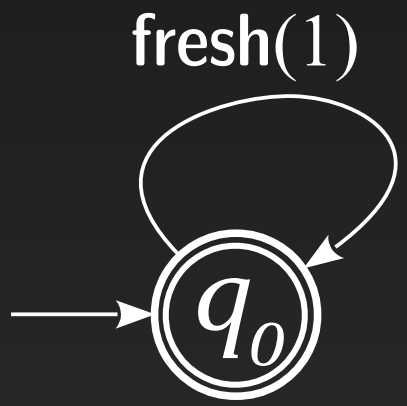


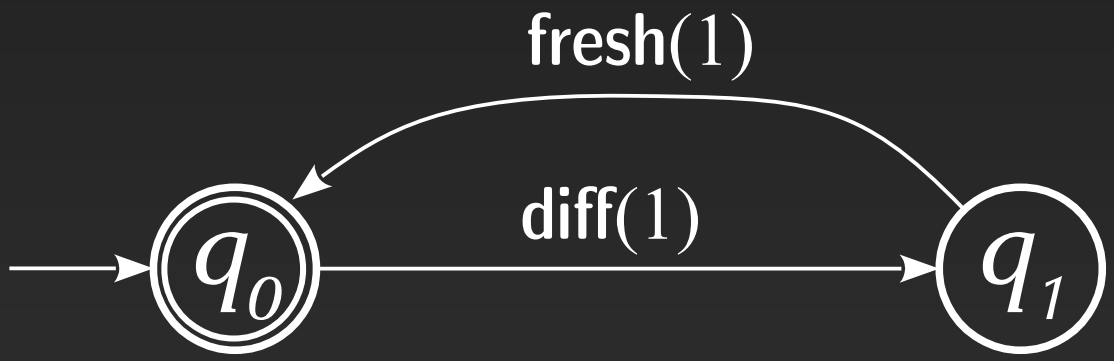


Examples

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

(all strings of pairwise distinct names)





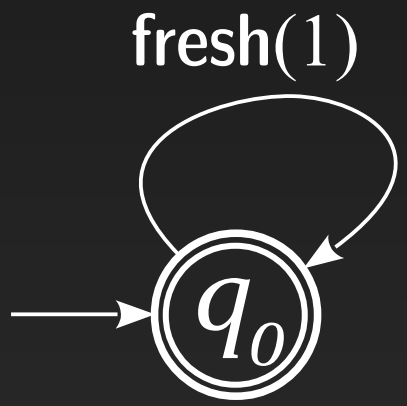
$$L_3 = \{\, a_1 a_2 \ldots a_{2n} \in \Sigma^* \mid n \geq 0,\ \forall i < 2n.\ a_i \neq a_{i+1}$$
$$\forall i \leq n,\ j < 2i.\ a_j \neq a_{2i} \,\}$$

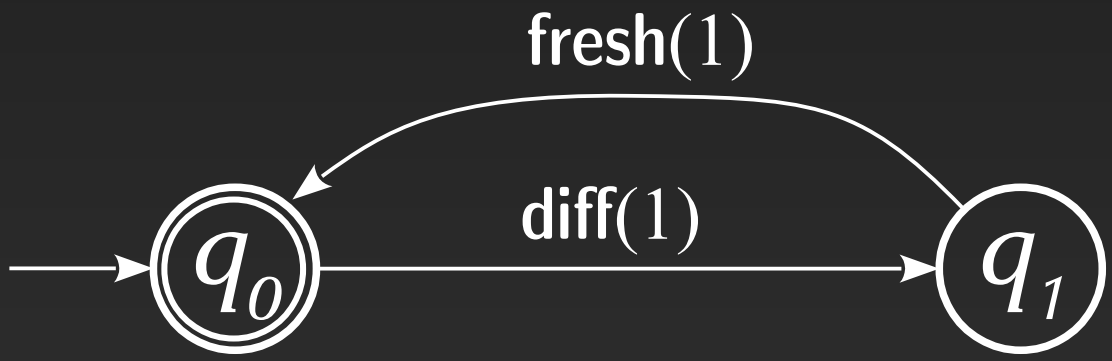
Examples

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

(all strings of pairwise distinct names)

– *what about* $L_{\text{fr}} \bullet L_{\text{fr}}$ *?*





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

FRA properties

- Closed under $\cup$, $\cap$, but not under $\bullet$, *
- Not closed under complement & not determinisable
- Universality / equivalence undecidable (from RAs)
- Decidable emptiness (same as RAs):
  - complexity depends on register “mode” (NL → NP → PSPACE)
- Great for program modelling and verification
  - Nominal algorithmic game semantics (Murawski, Ramsay, T.)

Limitations of FRAs

- Not closed under concatenation, repetition, interleaving and complementation

- Expressivity:
  - Cannot express non-freshness of names
  - What about consuming names
  - or classifying between fresh names

Interleaving

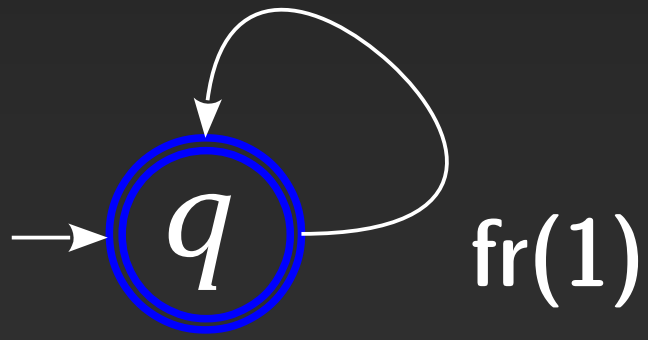
$$P(x) = va.\overline{x}a.P(x)$$ (name generator)

Interleaving

$P(x) = va.\overline{x}a.P(x)$ (name generator)

$\overline{x}a\ \overline{x}a'\ \overline{x}a''\ \ldots$

$L = \{a\ a'\ a''\ \ldots \mid \text{all names distinct}\}$



Interleaving

$$P(x) = va.\overline{x}a.P(x)$$ (name generator)

$$P(x) \mid * \mid P(y)$$

Name producer – produces and sends names on x

Name producer – produces and sends (possibly the same) names on y

Consumption

Name producer – produces and sends names on x

$$P(x) \parallel C(x,y)$$

Name consumer – Receives names on x and consumes them on y

Classification

Name producers – produce and send names on x and y

$$P(x) \parallel P(y) \parallel Q(x,y)$$

Receives names on x and y and uses them for different purposes

Register Automata

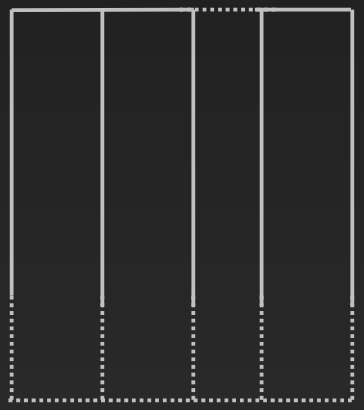
finitely many (say R) registers

registers store names

Label λ of the form:

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

History-Register Automata (HRAs)



Label λ of the form:

- $X/Y,\ X,Y \subseteq \{1,...,R\}$
- $\mathsf{rs}(i),\ i \in \{1,...,R\}$

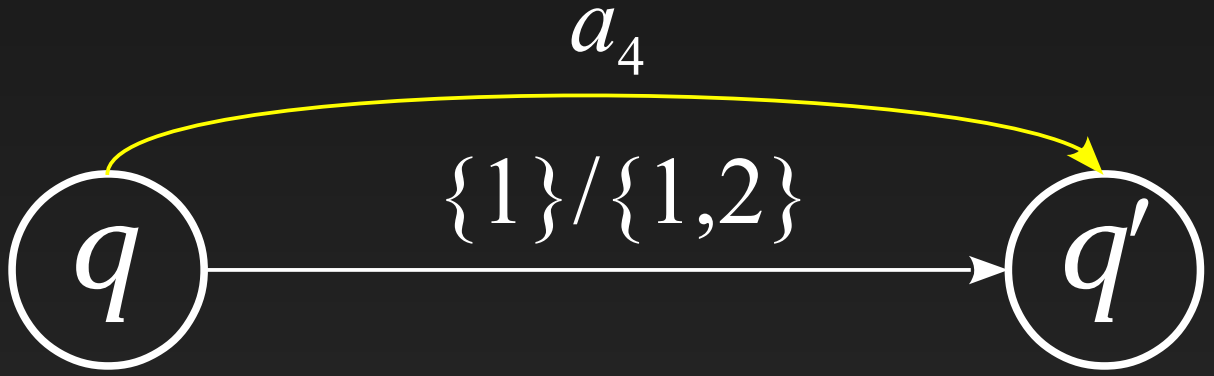
finitely many (say R) histories

histories store names

Transitions:

a_1	a_2	a_1
a_3	a_3	a_3
a_4		

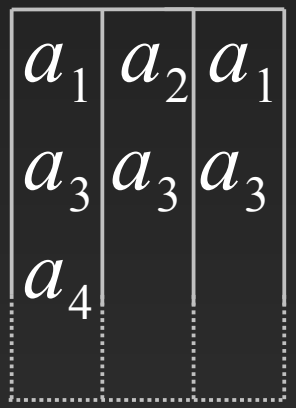
Transitions:



a_1	a_2	a_1
a_3	a_3	a_3
a_4		

a_1	a_2	a_1
a_3	a_3	a_3
a_4	a_4	

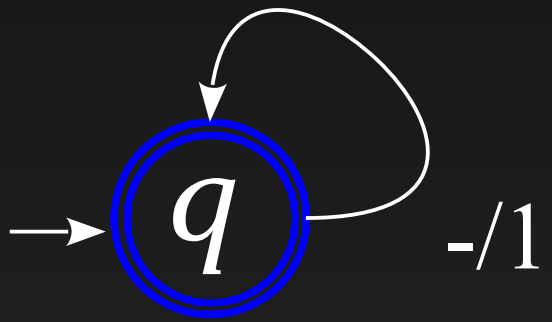
Transitions:



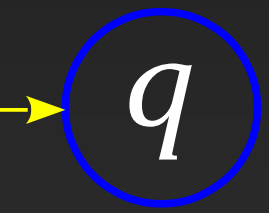
Transitions:

Name generator

$$P(x) = va.\bar{x}a.P(x)$$

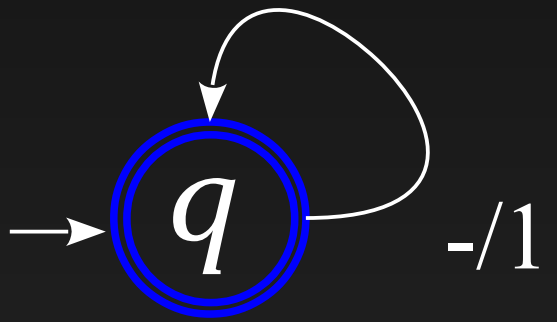


$$L = \{a\ a'\ a''\ \ldots \mid \text{all names distinct}\}$$



Name generator

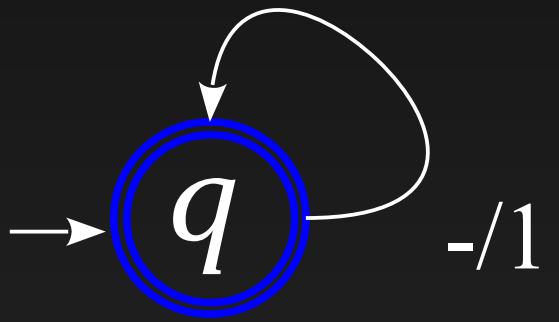
$$P(x) = va.\bar{x}a.P(x)$$



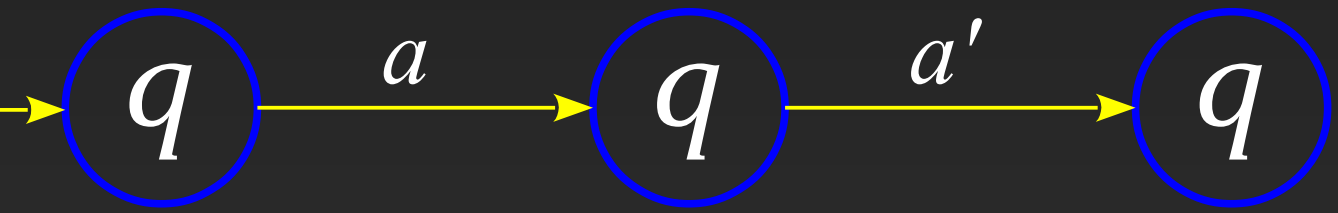
$$L = \{a\ a'\ a''\ \ldots \mid \text{all names distinct}\}$$

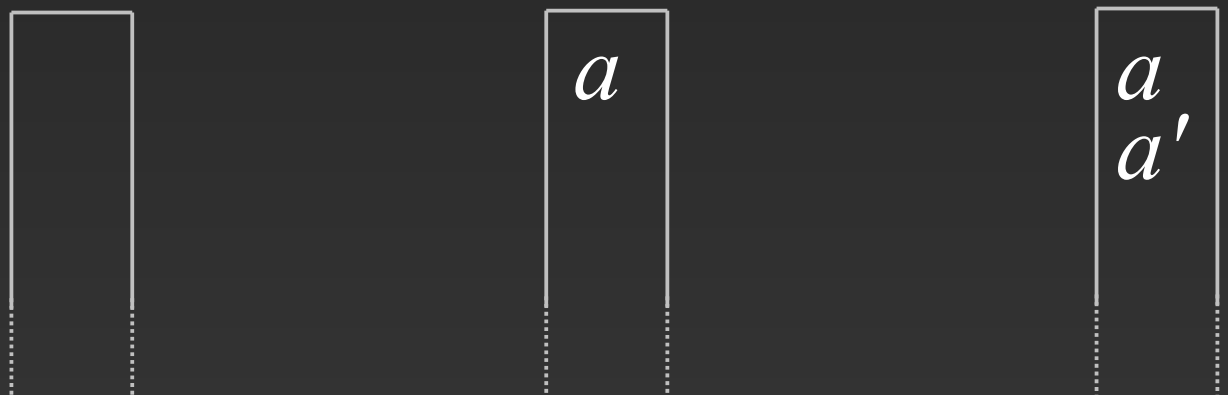
Name generator

$$P(x) = va.\bar{x}a.P(x)$$



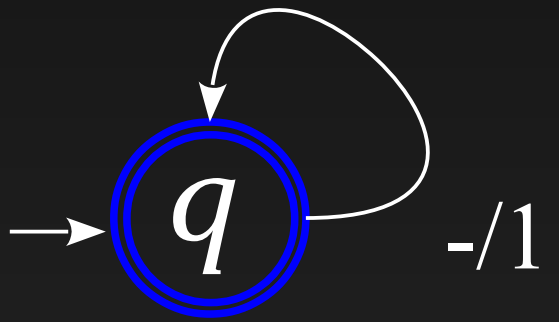
$$L = \{a\ a'\ a''\ \ldots \mid \text{all names distinct}\}$$



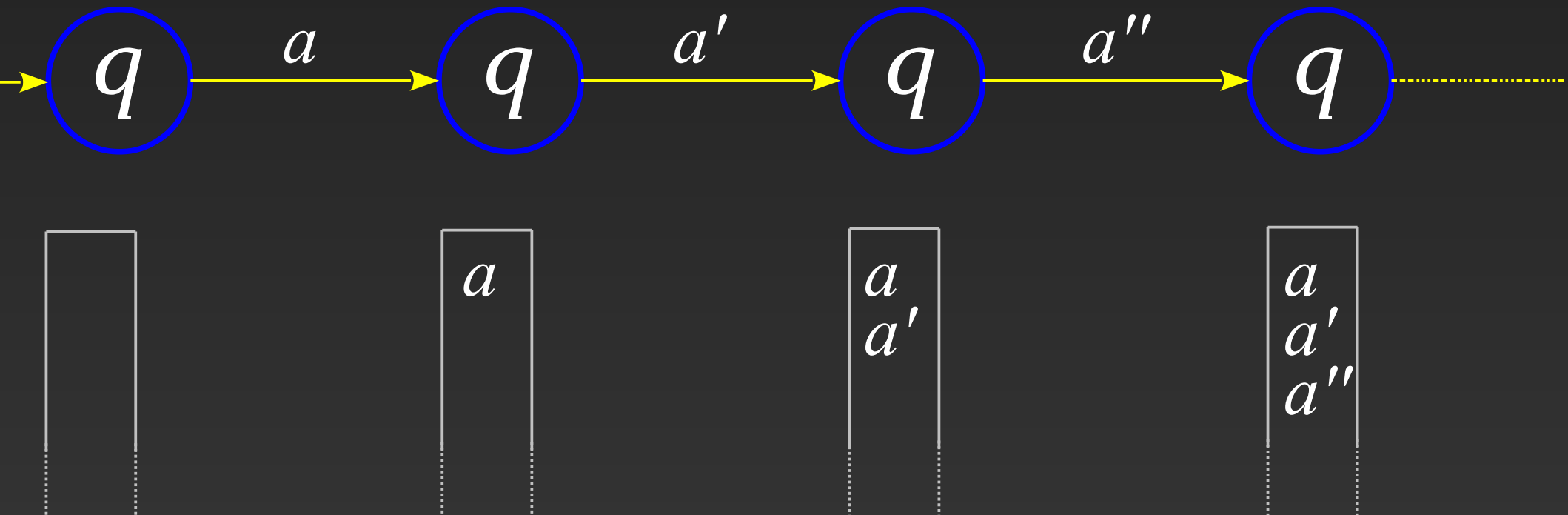


Name generator

$$P(x) = va.\bar{x}a.P(x)$$

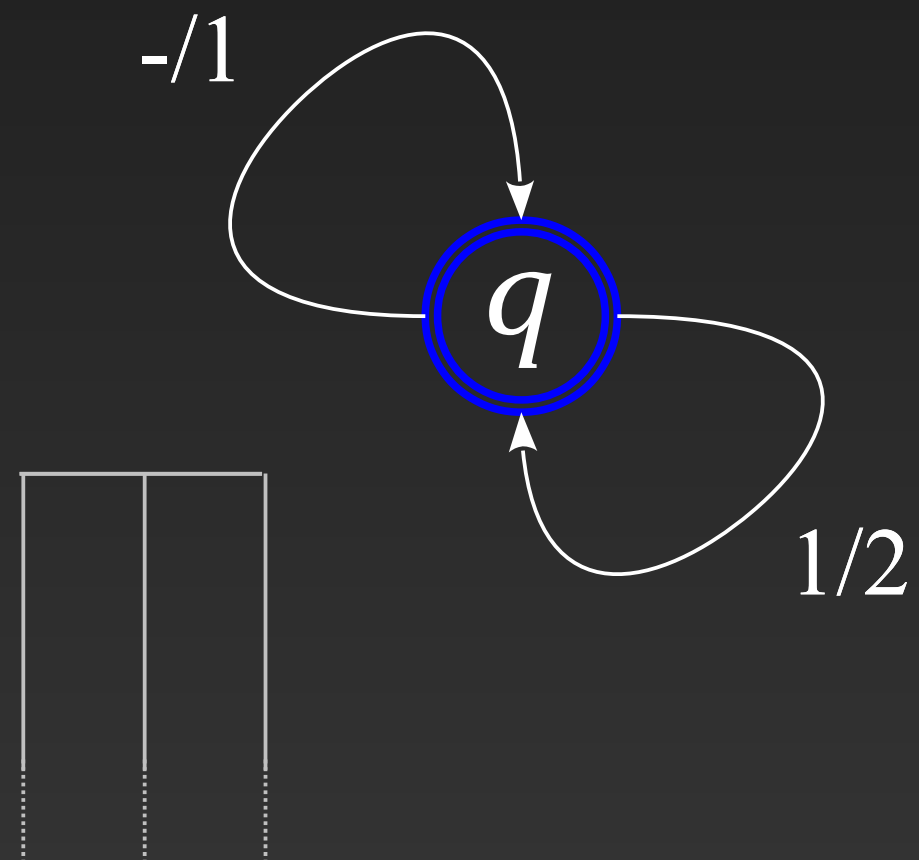


$$L = \{a\ a'\ a''\ \ldots \mid \text{all names distinct}\}$$



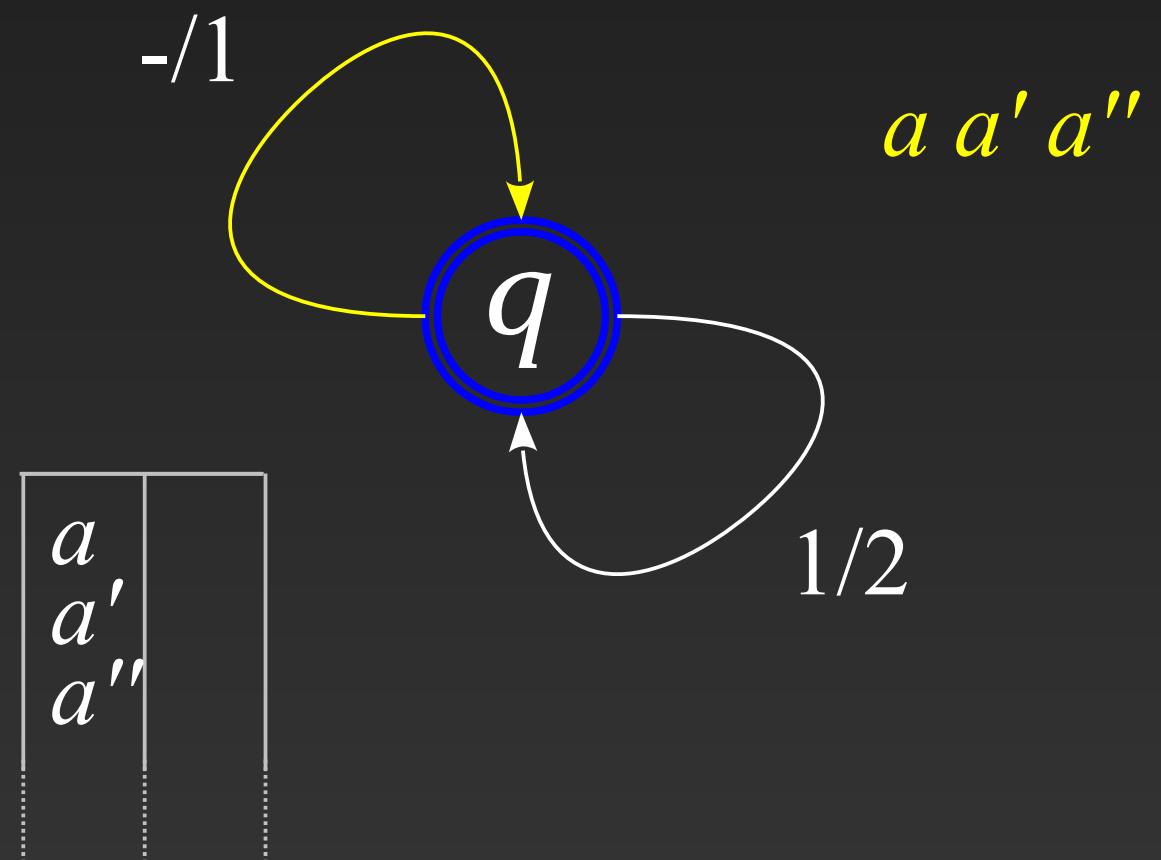
Consumption

$$P(x) \parallel C(x,y)$$



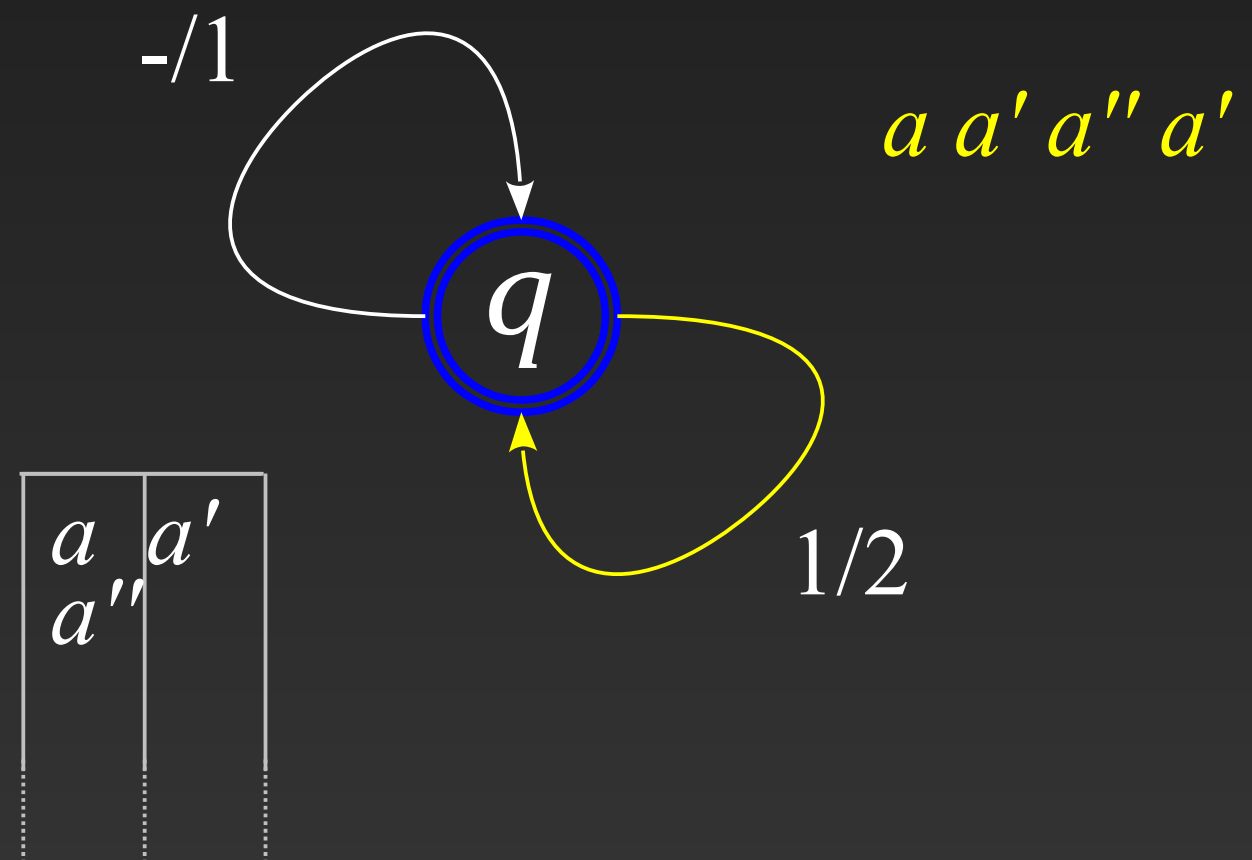
Consumption

$$P(x) \parallel C(x,y)$$



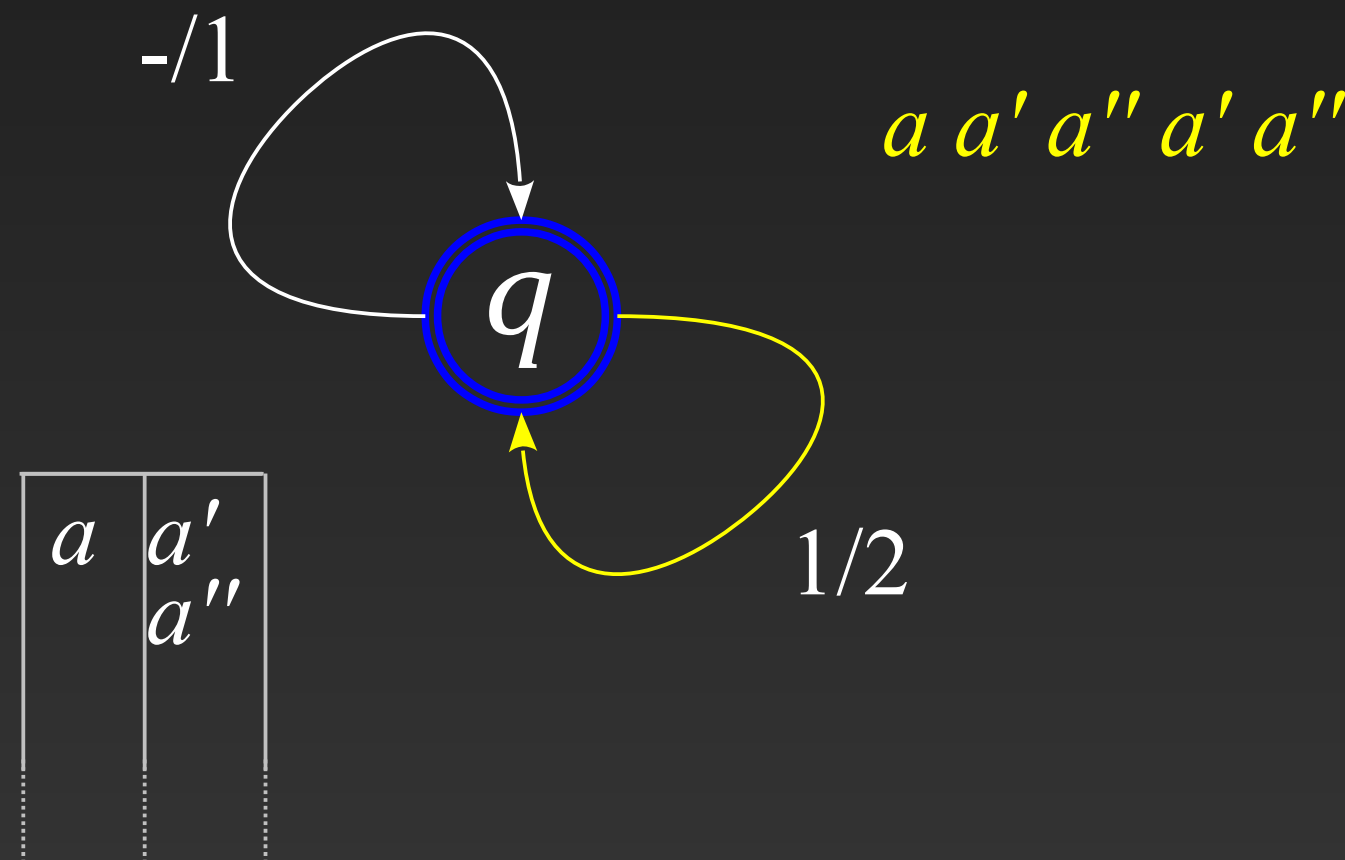
Consumption

$$P(x) \parallel C(x,y)$$



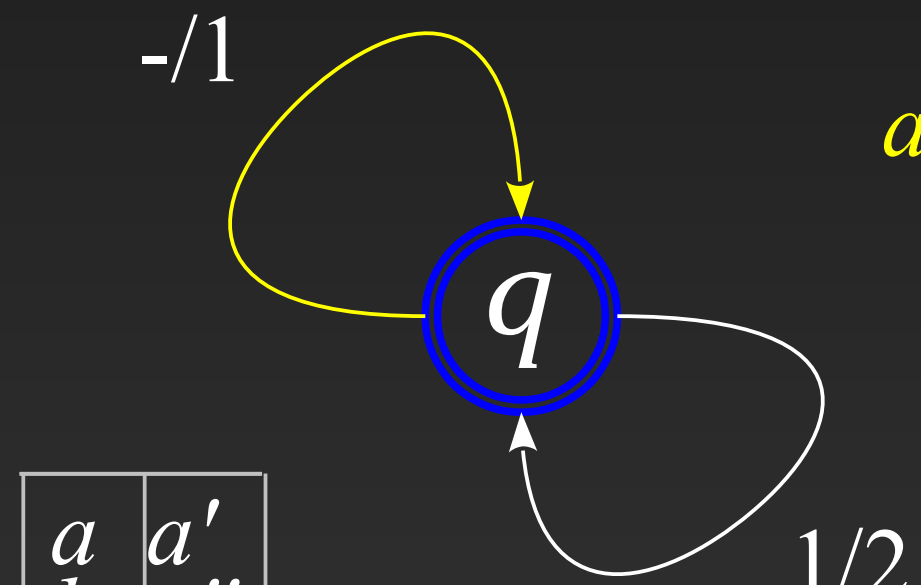
Consumption

$$P(x) \parallel C(x,y)$$

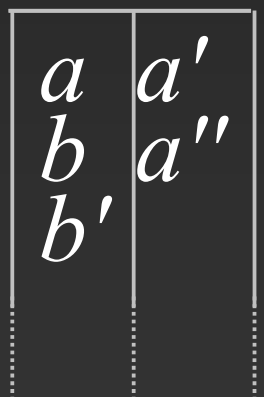


Consumption

$P(x) \parallel C(x,y)$

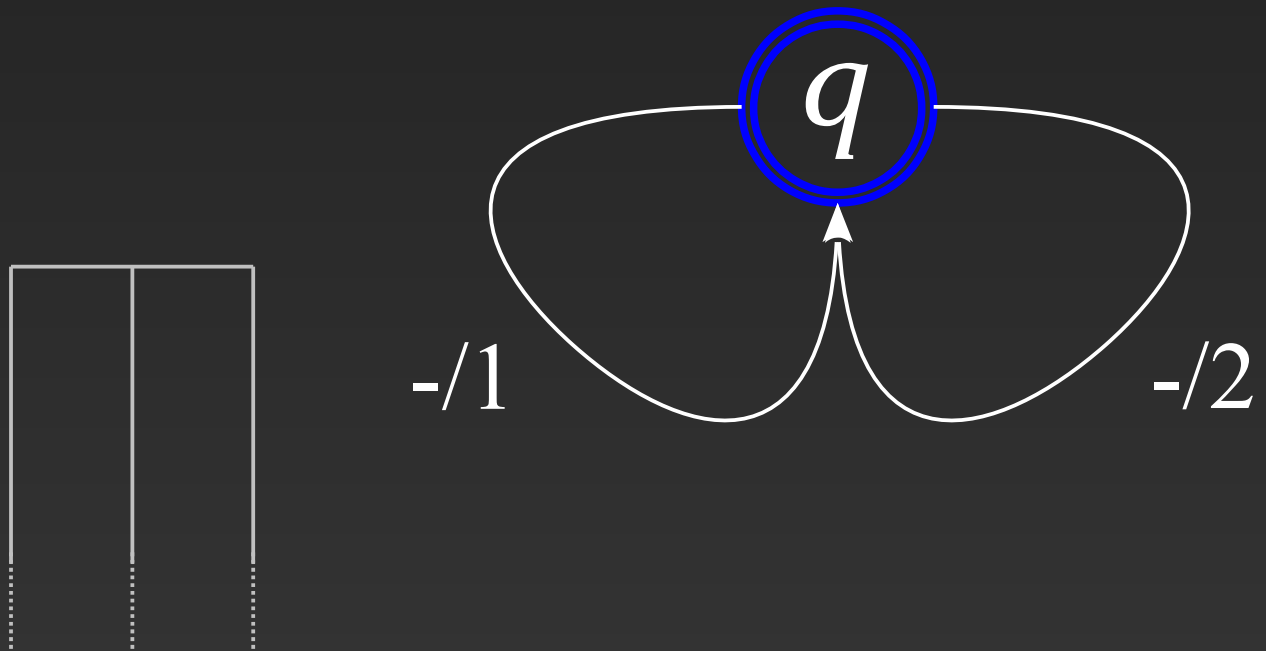


$a\ a'\ a''\ a'\ a''\ b\ b'$



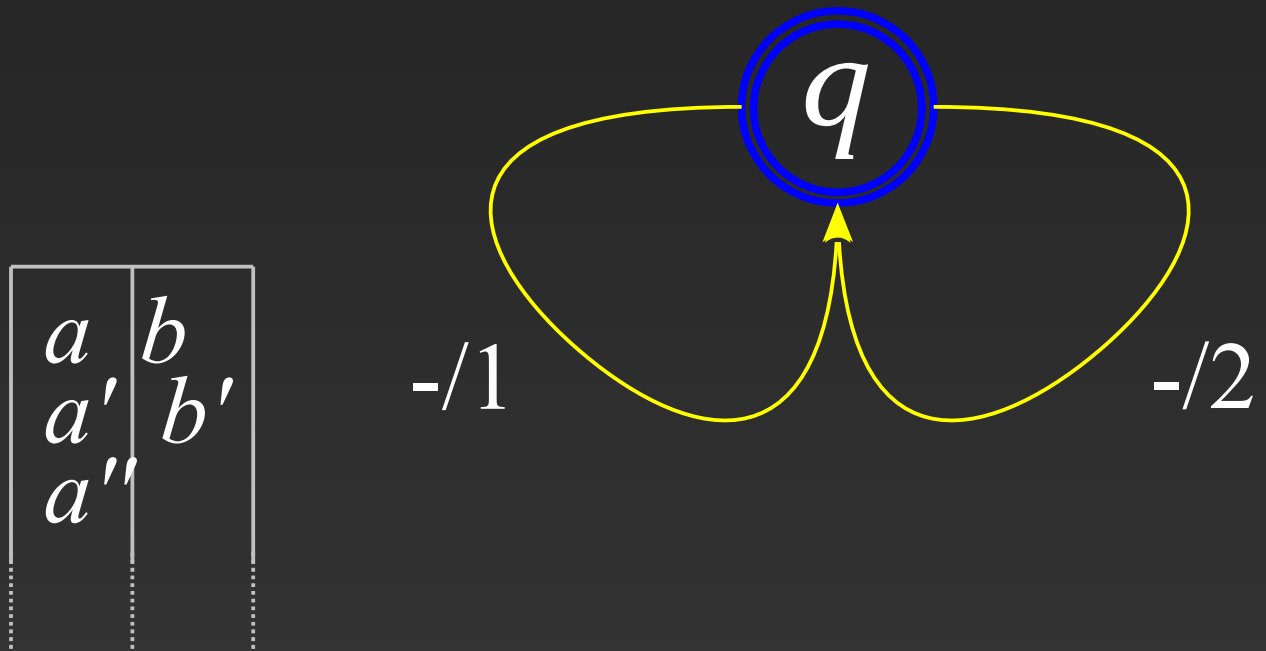
Interleaving

$$P(x) \mid * \mid P(y)$$



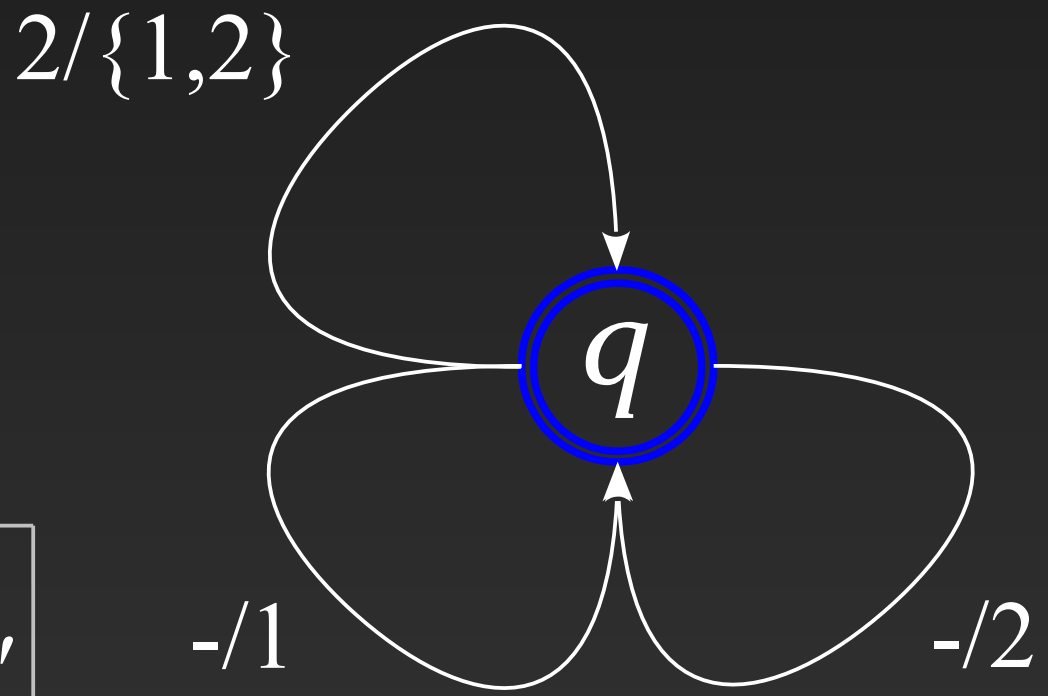
Interleaving

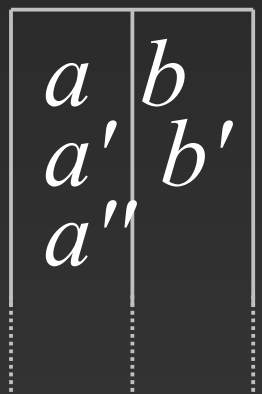
$$P(x) \mid * \mid P(y)$$



Interleaving

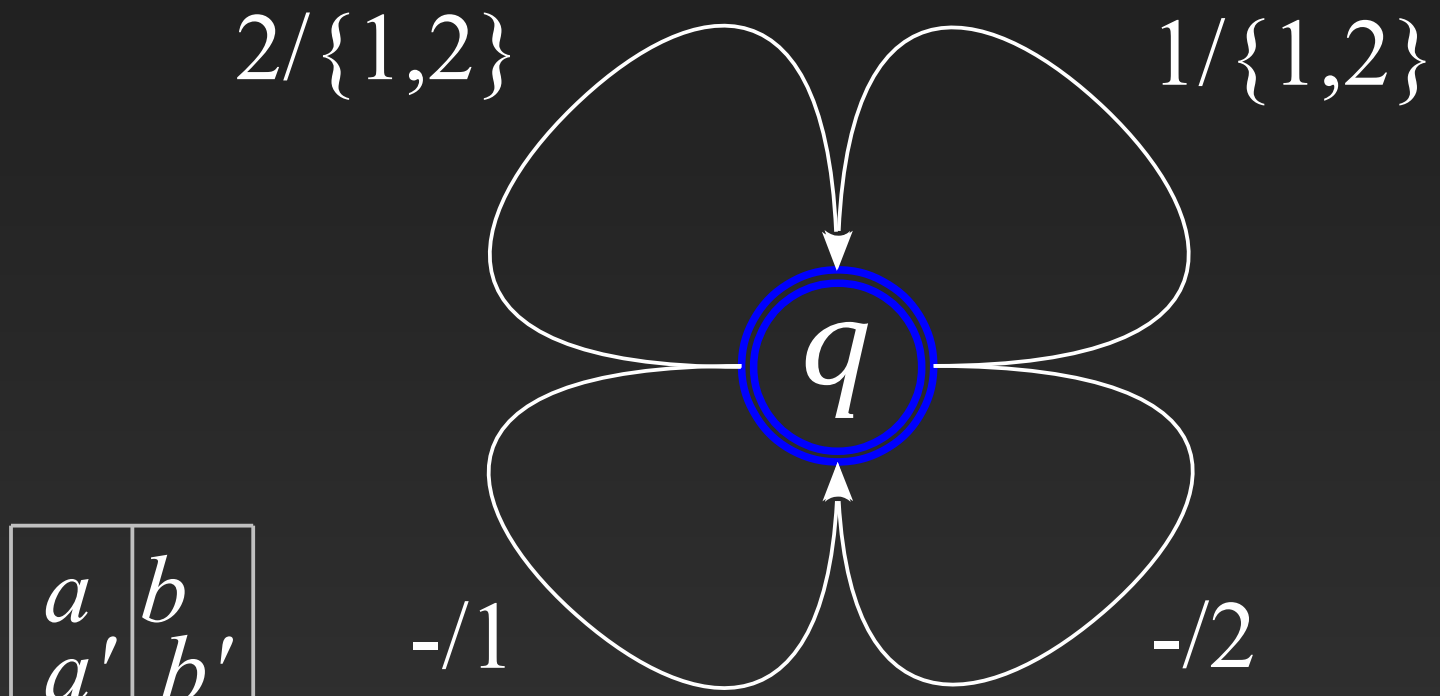
$$P(x) \mid * \mid P(y)$$

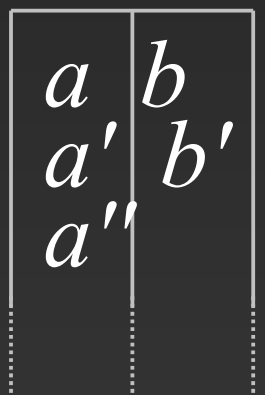




Interleaving

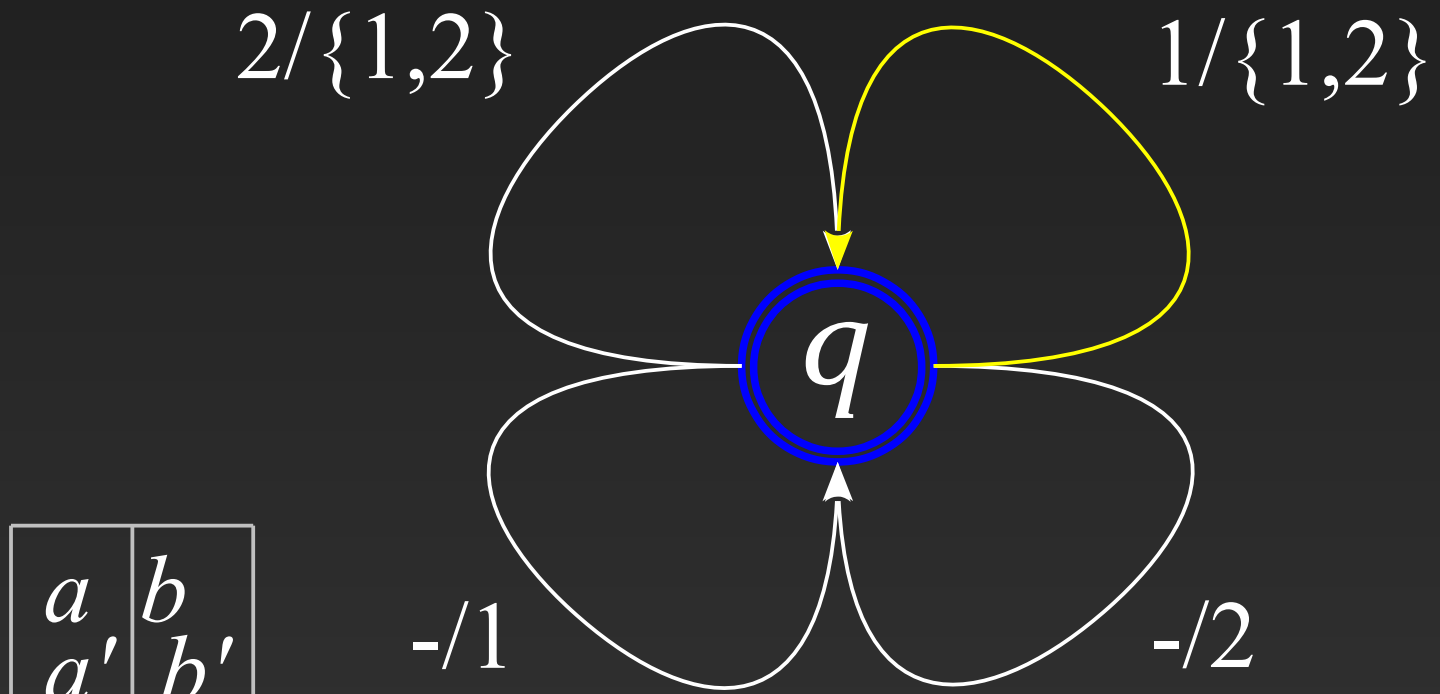
$$P(x) \mid * \mid P(y)$$

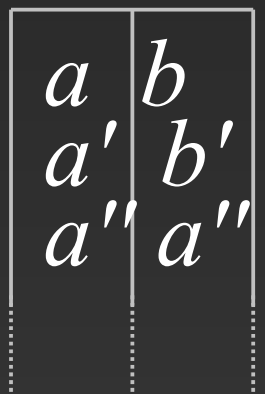




Interleaving

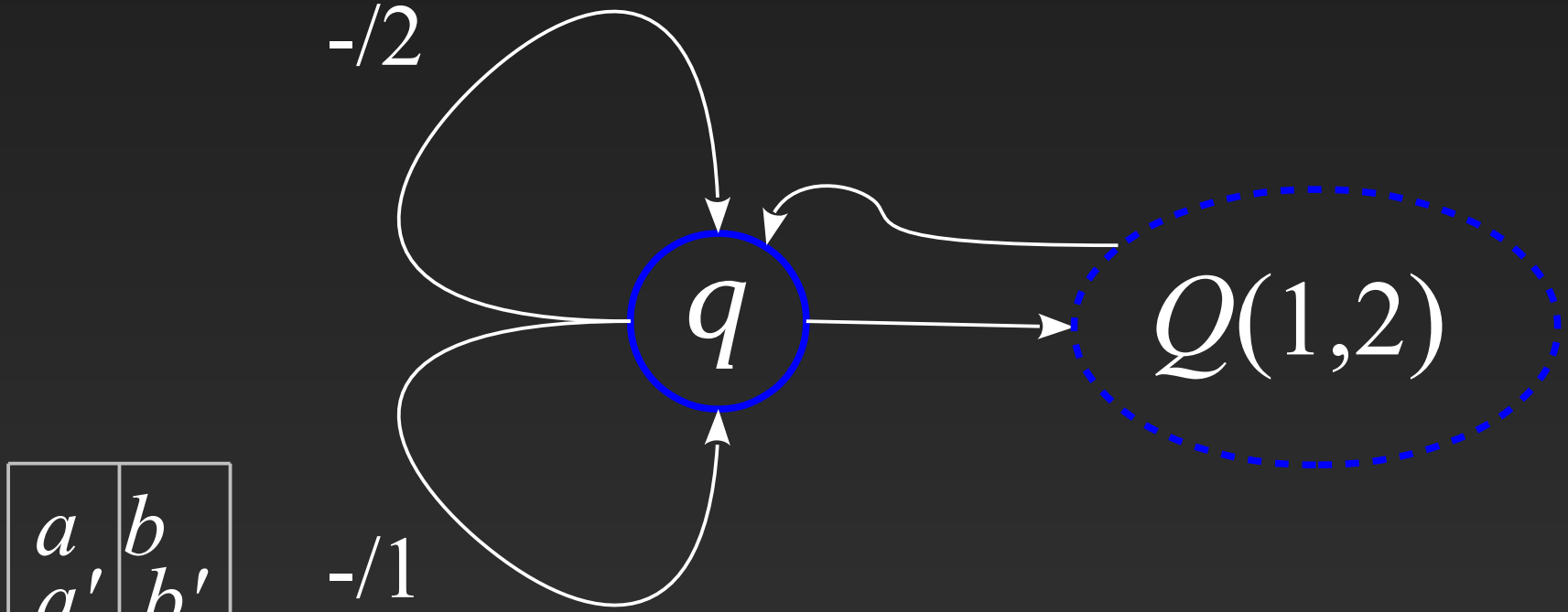
$$P(x) \mid * \mid P(y)$$

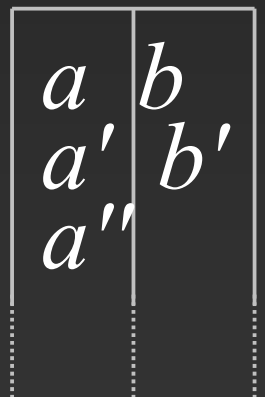




Classification

$$P(x) \parallel P(y) \parallel Q(x,y)$$





Properties

- Cleanly extend RAs and FRAs
  - even without using resets
- Closed under regular operations apart from complementation
- Closed under interleaving
- Universality undecidable (from RAs)
- Emptiness decidable, non-primitive recursive complexity (~transfer/reset Petri nets)

Restrictions

- Just one history (unary HRAs)
  - Lose intersection and interleaving
  - Emptiness in cubic space
- No resets (non-reset HRAs)
  - Lose closure under Kleene star
  - Emptiness EXPSPACE-complete

Automata for XML

LICS'06

Two-Variable Logic on Words with Data*

Mikołaj Bojańczyk, Warsaw University
Anca Muscholl, LIAFA, Paris VII
Thomas Schwentick, Dortmund University
Luc Segoufin, INRIA, Paris XI
Claire David, LIAFA, Paris VII

***Abstract*— In a *data word* each position carries a label from a finite alphabet and a data value from some infinite domain. These models have been already considered in the realm of semistructured data, timed automata and extended temporal logics.**

It is shown that satisfiability for the two-variable first-order logic $\mathrm{FO}^2(\sim,<,+1)$ is decidable over finite and over infinite data words, where $\sim$ is a binary predicate testing the data value equality and $+1$,$<$ are the usual successor and order predicates. The complexity of the problem is at least as hard as Petri net reachability. Several extensions of the logic are considered, some remain decidable while some are undecidable.

I. Introduction

Finding decidable logics for models that handle data values is an important problem in several areas that need algorithmic procedures for property validation. Examples can be found in both program verification and database management. In this paper we reconsider a data model that was investigated both in verification (related to timed languages [1] and extended temporal logics [4]) and in XML reasoning [16]. As in these papers, data values are modeled by an infinite alphabet, consisting of a finite and an infinite part. The logic can address the finite part directly, while the infinite part can only be tested for equality. As a first step, this paper considers simple models: words, both finite and infinite.

Our main result is that the satisfiability problem for two variable first-order logic extended by equality tests for data values – $\mathrm{FO}^2(\sim,<,+1)$ for short – is decidable over word models. When more variables are permitted, or when a linear order on the data values is available, or when more than navigational predicates on strings: the linear order $<$ and the successor relation $+1$. (With only two variables, the successor $+1$ cannot be defined in terms of the order $<$.) As usual we also have a unary predicate corresponding to each letter of the finite alphabet.

Perhaps surprisingly, we show that the satisfiability problem for $\mathrm{FO}^2(\sim,<,+1)$ is closely related to the well known problem of reachability for Petri nets. More precisely we show that languages formed by the projection onto the finite alphabet of word models definable by an $\mathrm{FO}^2(\sim,<,+1)$ sentence are recognized by multicounter automata (which are equivalent to Petri nets [7]). The converse is also true, modulo an erasing inverse morphism. Moreover, the correspondences are effective. We give a 2ExpTime reduction of satisfiability for $\mathrm{FO}^2(\sim,<,+1)$ to emptiness for multicounter automata which is known to be decidable [11], [14]. For the opposite direction we provide a PTime reduction from emptiness for multicounter automata to satisfiability for $\mathrm{FO}^2(\sim,<,+1)$. Since there is no known elementary upper bound for emptiness for multicounter automata (see e.g. [5]), finding the exact complexity of satisfiability for $\mathrm{FO}^2(\sim,<,+1)$ formulas is a hard question.

The decidability of $\mathrm{FO}^2(\sim,<,+1)$ immediately implies the decidability of $\mathrm{EMSO}^2(\sim,<,+1)$. Here EMSO^2 stands for the closure of FO^2 under existential quantification of sets of nodes. Without data values, $\mathrm{EMSO}^2(+1)$ has the same expressive power as MSO. In this sense the decidability of $\mathrm{EMSO}^2(\sim,<,+1)$ can be seen as an extension of the classical decidability result of monadic second-order logic over strings.

Automata for XML

LICS'06

Two-Variable Logic on Words with Data*

Mikołaj Bojańczyk (Warsaw University) Anca Muscholl (LIAFA, Paris VII) Thomas Schwentick (Dortmund University) Luc Segoufin (INRIA, Paris XI) Claire David (LIAFA, Paris VII)

***Abstract*— In a *data word* each position carries a label from a finite alphabet and a data value from some infinite domain. These models have been already considered in the realm of semistructured data, timed automata and extended temporal logics.**

It is shown that satisfiability for the two-variable first-order logic $\mathrm{FO}^2(\sim,<,+1)$ is decidable over finite and over infinite data words, where $\sim$ is a binary predicate testing the data value equality and $+1$,$<$ are the usual successor and order predicates. The complexity of the problem is at least as hard as Petri net reachability. Several extensions of the logic are considered, some remain decidable while some are undecidable.

I. Introduction

Finding decidable logics for models that handle data values is an important problem in several areas that need algorithmic procedures for property validation. Examples can be found in both program verification and database management. In this paper we reconsider a data model that was investigated both in verification (related to timed languages [1] and extended temporal logics [4]) and in XML reasoning [16]. As in these papers, data values are modeled by an infinite alphabet, consisting of a finite and an infinite part. The logic can address the finite part directly, while the infinite part can only be tested for equality. As a first step, this paper considers simple models: words, both finite and infinite.

Our main result is that the satisfiability problem for two variable first-order logic extended by equality tests for data values – $\mathrm{FO}^2(\sim,<,+1)$ for short – is decidable over word models. When more variables are permitted, or when a linear order on the data values is available, or when more than

navigation
$<$ and the
variables,
terms of
unary pred
finite alph

Perhaps
bility prob
to the w
Petri nets.
formed by
of word
sentence
(which ar
verse is al
phism. M
tive. We
bility for
counter au
[11], [14].
PTIME re
automata
there is
emptiness
finding th
$\mathrm{FO}^2(\sim,<$

The de
implies t
Here EMS
existential
data value
power as
$\mathrm{EMSO}^2(\sim$
the classic

TCS'10

On notions of regularity for data languages☆

Henrik Björklund*,1, Thomas Schwentick
Technische Universität Dortmund, Computer Science Chair 1, D-44227 Dortmund, Germany

ABSTRACT

With motivation from considerations in XML database theory and model checking, data strings have been introduced as an extension of finite alphabet strings which carry, at each position, a symbol *and* a data value from an infinite domain. Previous work has shown that it is difficult to come up with an expressive yet decidable automaton model for data languages. Recently, such a model, *data automata*, was introduced. This paper introduces a simpler but equivalent model and investigates its expressive power, algorithmic and closure properties, and some extensions.

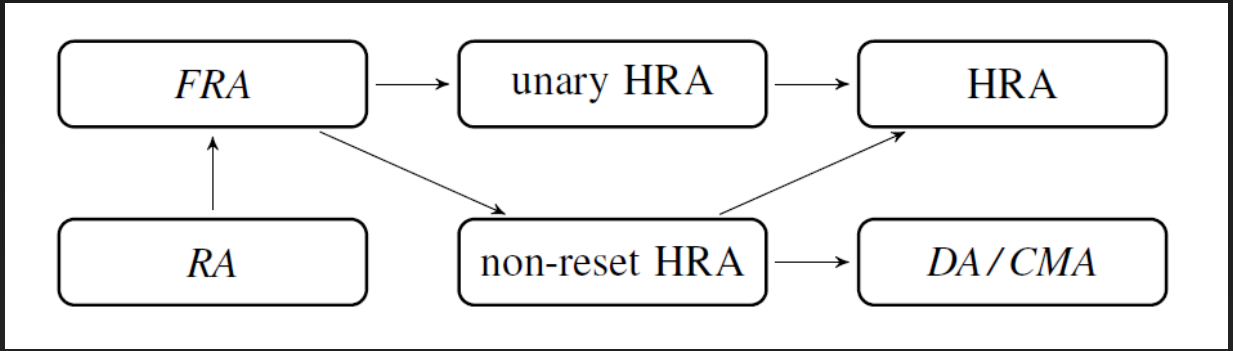
1. Introduction

The concept of regular string languages is clearly one of the most fundamental concepts in (theoretical) computer science. It has applications in basically all branches of computer science. It can be argued that the following properties were crucial to its success: (1) Expressiveness, (2) decidability, (3) efficiency, (4) closure properties, and (5) robustness. The notion of regularity has been successfully generalized to other kinds of structures, including infinite strings and finite or infinite, ranked or unranked trees. More recent applications of regular languages (on infinite strings and finite, unranked trees, respectively) can be found in model checking and XML processing.

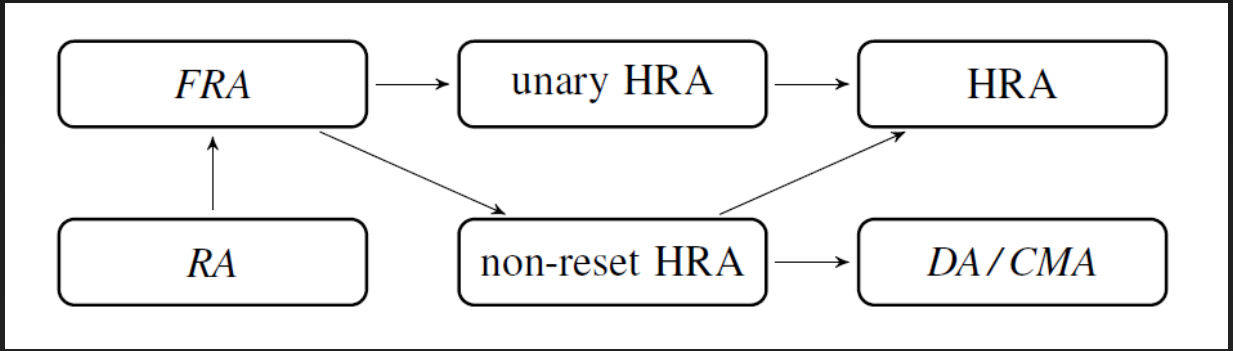
In model checking, a system is given as a finite state model, and properties are specified in a logic like LTL. The step from the "real" system to its finite state representation usually involves many abstractions, especially with respect to data values (variables, process numbers, etc.). Often their range is restricted to a (small) finite domain. Even though this approach has been successful and found its way into large scale industrial applications, the finite abstractions have some inherent shortcomings. As an example, n identical processes with m states each give rise to an overall model size of m^n. (Symbolic model checking partially addresses this problem by "compressing" redundant states.) If the number of processes is unbounded and/or unknown in advance, the finite state approach fails. Previous work has shown that even then, decidability can be obtained by restricting the problem in various ways [9,1].

In XML document processing, regular concepts occur in various contexts. First, most applications restrict the structure of the allowed documents to conforming to a certain schema, which can be modeled as a regular tree language. Second, navigation (XPath) and transformation (XSLT) languages have tight connections to tree automata models and other regular

Picture and Further Directions



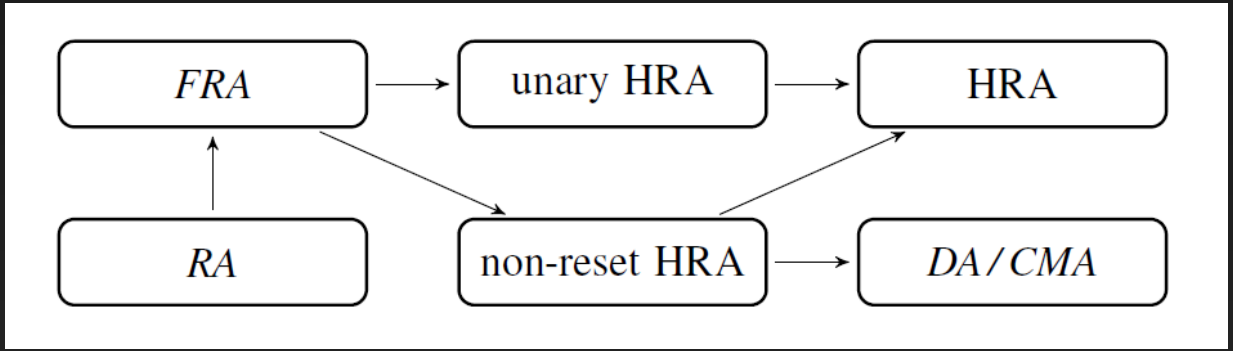
Picture and Further Directions



- Use in applications:
  - Runtime verification
  - Model checking
- Explore variants:
  - Emptiness checks
  - Deterministic

Picture and Further Directions

thank you!



- Use in applications:
  - Runtime verification
  - Model checking
- Explore variants:
  - Emptiness checks
  - Deterministic