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
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
infinite alphabets & program behaviour

```java
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*
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, which 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

- examine languages over $\Sigma^*$
- 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.
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]
Let $\Sigma = \{a_1, a_2, \ldots, a_n, \ldots\}$ be an infinite alphabet of names.

Label $\lambda$ of the form:

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

Finitely many (say $R$) registers store names.
Transitions:

\[ q \xrightarrow{\text{reg}(i)} q' \]

\[ q \xrightarrow{\text{reg}(2)} q' \]

\[ a \quad g \quad b \]
Transitions:

$q \xrightarrow{\text{reg}(i)} q'$

$q \xrightarrow{g} q'$

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

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

$\begin{array}{c|c|c}
\text{a} & \text{g} & \text{b} \\
\end{array}$
Transitions:

\[ q \xrightarrow{\text{diff}(i)} q' \]

\[ q \xrightarrow{\text{diff}(2)} q' \]

\[ a \ g \ b \]
Transitions:

$\text{diff}(i)$

$c$

$\text{diff}(2)$

different from current registers
Example

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

(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)
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

\[ L_1 = \{ a_1 a_2 \ldots a_n \in \Sigma^* | \ n \geq 0, \ \forall i < n. \ a_i \neq a_{i+1} \} \]
(all strings where each name is distinct from its predecessor)
Example

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

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

Diagram:

$q_0 \xrightarrow{diff(1)}$

$\downarrow$

$a$

$abaaca$
Example

\[ L_1 = \{ a_1 a_2 ... a_n \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \} \]
(all strings where each name is distinct from its predecessor)

\[ \text{diff}(1) \]

\[ \text{q}_0 \]

\[ d \]

\[ abcd \]
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)

\[ \text{diff}(1) \]

\[ q_0 \]

\[ e \]

\[ abc a d e \]
Example

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

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

\[\text{diff}(1)\]

\[q_0\]

\[b\]

\[abcadeb\]
Example

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

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

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

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

$abcadebagcab and we love cake$
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)

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

(all strings of pairwise distinct names)

– what about the complement of $L_{fr}$? And that of $L_{fr} \cdot L_{fr}$?
RA properties

- Capture regularity when $\Sigma$ restricted to finite
  - Closed under $\cup, \cap, \cdot, \cdot^*$
  - Not closed under complement & not determinisable

- Universality / equivalence undecidable

- Decidable emptiness:
  - Complexity depends on register “mode” ($\text{NL} \rightarrow \text{NP} \rightarrow \text{PSPACE}$)

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

[Refs]
- [Kaminski & Francez '94]
- [Neven, Schwentick & Vianu '04]
- [Sakamoto & Ikeda '00; Demri & Lazić '09]
Example revisited

```java
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 $\phi$: 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

```java
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 $\varphi$:

> 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]

but we cannot capture new!
Let $\Sigma = \{a_1, a_2, \ldots, a_n, \ldots\}$ be an infinite alphabet of names.

Label $\lambda$ of the form:

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

Finitely many (say $R$) registers

Registers store names

Global freshness oracle
Transitions:

\[ q \xrightarrow{\text{fresh}(i)} q' \]

\[ b_1, \ldots, b_k \]

\[ a \quad g \quad b \]
Transitions:

\[ q \xrightarrow{fresh(i)} q' \]

\[ q \xrightarrow{b_1, \ldots, b_k} q \xrightarrow{c} q' \]

\[ a \quad g \quad b \]

\[ a \quad c \quad b \]

\textit{globally fresh}
Examples

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

*(all strings of pairwise distinct names)*

\[ L_3 = \{ a_1 a_2 ... a_{2n} \in \Sigma^* \mid n \geq 0, \forall i < 2n. \ a_i \neq a_{i+1} \]

\[ \forall i \leq n, j < 2i. \ a_j \neq a_{2i} \} \]
FRA properties

- Not closed under complement & not determinisalbe
  - Closed under \( \cup, \cap \), but not under \( \cdot, * \)
- Universality / equivalence undecidable (from RAs)
- Decidable emptiness (same as RAs):
  - complexity depends on register “mode” (NL \( \rightarrow \) NP \( \rightarrow \) 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 $\rightarrow$ game model $\rightarrow$ FRA

effectively:

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

what we get:

- decision procedures for ML fragments
- same for Interface Middleweight Java

http://bitbucket.org/sjr/coneqct/wiki/Home

[Murawski & T. '11, '12]
[Murawski, Ramsay & T. '15]
More applications and variants

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

Session automata and learning
- freshness, but no diff
- canonical forms, decide equivalence

Kleene algebras for languages with binders
- NKA: KA with ν-binder → match with automata
Investigations in FRAs

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

Context-freeness: Pushdown FRA
- Reachability EXPTIME-complete
- Global reachability via “saturation”

Freshness oracle: from one to many histories
- History Register Automata (cf. DA/CMA)
Semantics formally: configurations

Semantics of FRAs given by configuration graphs:

Configuration:

\[ (q, \rho, H) \rightarrow (q', \rho', H') \]

State:

Register assignment:
\[ \rho : \{1, \ldots, R\} \rightarrow \Sigma \cup \{\#\} \]

History:
\[ H \subseteq_{\text{fin}} \Sigma \]
A *behavioural* notion of equivalence:

Two configurations $\kappa_1, \kappa_2$ 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 $1^*$ 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
\((S\#) \rightarrow (MF)\)

- \((S\#)\): no multiplicities, but erasing allowed
- \((MF)\): multiplicities, but no empty registers

\(a\ g\ #\ b\ #\)

\(z\ a\ g\ z\ b\ z\)

Neat, but erasing gives exponentially large labels
\[(S\#) \rightarrow (\textit{MF})\]

- 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
**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”

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

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

(bisimilarity for finite-state automata is in PTIME)
**Multiplicity:**
- (S) single
- (M) multiple

**Fullness:**
- (F) full
- (#₀) initially empty
- (#) eraseable

EXPTIME

```
M#

S#

S#₀

SF
```
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 (\textbf{Attacker (A) vs Defender (D)}):
  - A controls universal states, D controls existential ones
  - use \textit{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

---

Graph:
- **S#**
- **M#**
- **MF**
- **S#₀**
- **SF**

**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!

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

\[
\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \ldots \supseteq \sim_i \supseteq \ldots \quad \text{and} \quad \sim_s = \bigcup_{i \in \omega} \sim_i
\]

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 \( \rightarrow \) APTIME
**Complexity Picture**

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

**Fullness:**
- (F) full
- (#₀) initially empty
- (#) eraseable

Diagram:
- EXPTIME-c
- PSPACE
- PSPACE

Nodes:
- S#
- M#
- MF
- S#₀
- SF
Bisimilarity for (F)RAs

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

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

---

EXPTIME-c

PSPACE-c

NP
Investigations in FRAs

Bisimilarity for FRAs (complexity)
- Depends on register mode ($\text{NP} \rightarrow \text{PSPACE} \rightarrow \text{EXPTIME}$)
  - approach uses permutation group theory

Context-freeness: Pushdown FRA
- Reachability EXPTIME-complete
- Global reachability via “saturation”

Freshness oracle: from one to many histories
- History Register Automata (cf. DA/CMA)
Let $\Sigma = \{a_1, a_2, \ldots, a_n, \ldots\}$ be an infinite alphabet of names.

Label $\lambda$ of the form:

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

finitely many (say $R$) registers

pushdown stack

registers & stack store names
Transitions:

\[
q \xrightarrow{\text{push}(i)} q'
\]

\[
q \xrightarrow{\text{push}(2)} q'
\]

\[
\begin{array}{ccc}
a & c & b \\
e & a & c \\
\end{array}
\]
Transitions:

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

push(2)

$a \quad c \quad b$

$e \quad a \quad c$
Transitions:

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

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

$abc$ $abc$

$eac$ $ceac$
Transitions:

\[ q \xrightarrow{\text{pop}(i)} q' \]

\[ q \xrightarrow{\text{pop}(2)} q' \]

\[ a \ c \ b \]

\[ c \ e \ a \]

\[ c \]
Transitions:

1. $q \xrightarrow{\text{pop}(i)} q'$
2. $q \xrightarrow{\text{pop}(2)} q'$

States: $a \quad c \quad b$

Pushed Symbols: $c \quad e \quad a \quad c$
Transitions:

\[
q \xrightarrow{\text{pop}(i)} q'
\]

\[
q \xrightarrow{\text{pop}(2)} q'
\]
Transitions:

\(q\overset{\text{pop-diff}}{\rightarrow}q'\)

\(q\overset{\text{pop-diff}}{\rightarrow}q'\)

\[a \quad c \quad b\]

\[e \quad a \quad c\]
Transitions:

\[ q \xrightarrow{\text{pop-diff}} q' \]

\[ q \xrightarrow{\text{pop-diff}} q' \]

different from current registers
Transitions:

\[ q \xrightarrow{\text{pop-diff}} q' \]

\[ q \xrightarrow{\text{pop-diff}} q' \]

Different from current registers
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

$L_4 = \{ a_1a_2...a_n a_n...a_2a_1 \in \Sigma^* \mid n \geq 0, \forall i < n. a_i \neq a_{i+1} \}$

\text{neveroddoreven}
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

\[ L_{fr} = \{ a_1a_2...a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. \ a_i \neq a_j \} \]

(all strings of distinct names)
**Limited distinguish-ability**

\[ L_{fr} = \{ a_1a_2...a_n \in \Sigma^* \mid n \geq 0, \forall i \neq j. a_i \neq a_j \} \]

(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 3\( R \) names.

Conversely, there is a PDRA with \( R \) registers whose runs to a designated state involve exactly 3\( R \) 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

Context-freeness: Pushdown FRA
- Reachability EXPTIME-complete
- Global reachability via “saturation”

Freshness oracle: from one to many histories
- History Register Automata (cf. DA/CMA)
HRAs: from registers to histories

histories = registers with unboundedly many equivalent elements
Expressivity

Histories simulate registers $\rightarrow$ HRAs extend FRAs

Histories can be reset:
$\rightarrow$ Closure under Kleene* and concatenation

Several histories $\rightarrow$ 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!)