Fresh-Register Automata

Nikos Tzevelekos

Oxford University Computing Laboratory
What this talk is about

What is a basic automata-theoretic model of computation with names and fresh-name generation?
Names in computation

```java
int mainClass NameTest {
    public void main(String[] args){
        Object A = new Object();
        Object B = new Object();
        System.out.println(A.equals(B));
    }
}
```
Names in computation

new x=3 in f(); assert(x==3)
Names in computation

new x1, x2, x3, x4, x5 in
  fn x => case x of x1 => x2
  | x2 => x3
  | x3 => x4
  | x4 => x5
  | _  => x1
Motivation and related work

What is a basic automata-theoretic model of computation with names and fresh-name generation?

- Programming languages
  - Operational, denotational models of higher-order computation with names
  - Nominal game semantics
Names in computation (II)

\[ P(a) = \nu b. \bar{a}b. \ P(b) \]
Names in computation (II)

\[ P(a) = \forall b. \bar{a}b. P(b) \]
Names in computation (II)

\[ P(a) = \nu b. \bar{a}b. P(b) \]

\[ P(a) \xrightarrow{\bar{a}b} P(b) \]
Names in computation (II)

\[ P(a) = \nu b. \; \bar{a}b. \; P(b) \]

\[ P(a) \xrightarrow{\bar{a}b} P(b) \xrightarrow{\bar{b}c} P(c) \]
Names in computation (II)

\[ P(a) = \nu b. \bar{a}b. P(b) \]

\[
\begin{align*}
P(a) & \xrightarrow{\bar{a}b} P(b) \\
P(b) & \xrightarrow{\bar{b}c} P(c) \\
P(c) & \xrightarrow{\bar{c}d} P(d)
\end{align*}
\]
Names in computation (II)

\[ P(a) = \nu b. \bar{a}b. P(b) \]
$P(a) = \forall b. \overline{a}b. P(b)$
Motivation and related work

What is a basic automata-theoretic model of computation with names and fresh-name generation?

- Programming languages
  - Operational, denotational models of higher-order computation with names
  - Nominal game semantics
- Process calculi
  - Semantics of mobility
  - History-Dependent automata
Specifications

What is a basic automata-theoretic model of computation with names and fresh-name generation?

- Simple machines of “first principles”
- Infinite alphabet
- Freshness recognition
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


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 model to have a close relationship with other models of the
An appealing paradigm

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

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 model to avoid the use of languages recognizable by interpreting an array of the...
Fresh-Register Automata

- FMA's satisfy the specifications:
  - *Simple machines of “first principles”*
  - *Infinite alphabet*
- but not:
  - *Freshness recognition*

- Extend FMA's with transitions for fresh names.
Do names with registers

- Let $\mathbb{A}$ be an infinite set of *names*
- Let $\mathbb{C}$ be a finite set of *constants*

- Consider finite-state automata over:

$$\mathbb{L}_n = \mathbb{C} \cup \{ i, i^\bullet, i^\ast \mid 1 \leq i \leq n \}$$
Do names with registers

• Let $\mathcal{A}$ be an infinite set of names
• Let $\mathcal{C}$ be a finite set of constants

• Consider finite-state automata over:

$$\mathbb{L}_n = \mathcal{C} \cup \{ i, i^\bullet, i^\ast \mid 1 \leq i \leq n \}$$

• but operate in reality over:

$$\mathcal{C} \cup \mathcal{A}$$
Definition

- Recall: \( \mathbb{L}_n = \mathbb{C} \cup \{ i, i^\bullet, i^\circ \mid 1 \leq i \leq n \} \)

- Define *register assignments* of size \( n \) by:

\[
\text{Reg}_n = \{ \sigma : \{1, \ldots, n\} \rightarrow A \cup \{\#\} \mid \forall i \neq j . \sigma(i) = \sigma(j) \implies \sigma(i) = \# \}
\]
A **fresh-register automaton (FRA)** of \( n \) registers is a quintuple \( \mathcal{A} = \langle Q, q_0, \sigma_0, \delta, F \rangle \) where:

- \( Q \) is a finite set of states,
- \( q_0 \in Q \) is the initial state,
- \( \sigma_0 \in \text{Reg}_n \) is the initial register assignment,
- \( \delta \subseteq Q \times \mathbb{L}_n \times Q \) is the transition relation,
- \( F \subseteq Q \) is the set of final states.

\[
\mathbb{L}_n = \mathbb{C} \cup \{ i, i^\bullet, i^\star \mid 1 \leq i \leq n \}
\]

\[
\text{Reg}_n = \{ \sigma : \{1, \ldots, n\} \rightarrow \mathbb{A} \cup \{\#\} \mid \forall i \neq j. \sigma(i) = \sigma(j) \implies \sigma(i) = \# \}
\]
A fresh-register automaton (FRA) of $n$ registers is a quintuple $\mathcal{A} = \langle Q, q_0, \sigma_0, \delta, F \rangle$ where:

- $Q$ is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\sigma_0 \in \text{Reg}_n$ is the initial register assignment,
- $\delta \subseteq Q \times \mathbb{L}_n \times Q$ is the transition relation,
- $F \subseteq Q$ is the set of final states.
A fresh-register automaton (FRA) of $n$ registers is a quintuple $A = \langle Q, q_0, \sigma_0, \delta, F \rangle$ where:

- $Q$ is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\sigma_0 \in \text{Reg}_n$ is the initial register assignment,
- $\delta \subseteq Q \times \mathbb{L}_n \times Q$ is the transition relation,
- $F \subseteq Q$ is the set of final states.

A configuration is a triple:

$$(q, \sigma, H) \in Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(A)$$

- state
- register assignment
- history
A fresh-register automaton (FRA) of \( n \) registers is a quintuple \( A = \langle Q, q_0, \sigma_0, \delta, F \rangle \) where:

- \( Q \) is a finite set of states,
- \( q_0 \in Q \) is the initial state,
- \( \sigma_0 \in \text{Reg}_n \) is the initial register assignment,
- \( \delta \subseteq Q \times \mathbb{L}_n \times Q \) is the transition relation,
- \( F \subseteq Q \) is the set of final states.

A configuration is a triple:

\[
(q, \sigma, H) \in Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(A)
\]
Configurations

A *fresh-register automaton (FRA)* of $n$ registers is a quintuple $\mathcal{A} = \langle Q, q_0, \sigma_0, \delta, F \rangle$ where:

- $Q$ is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\sigma_0 \in \text{Reg}_n$ is the initial register assignment,
- $\delta \subseteq Q \times \mathbb{L}_n \times Q$ is the transition relation,
- $F \subseteq Q$ is the set of final states.

- A configuration is a triple:

  $$(q, \sigma, H) \in Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(\mathcal{A})$$

- Transitions between config's: the 'true' semantics
Demo: known transitions

$q \xrightarrow{i} q'$
Demo: known transitions

\[ q \xrightarrow{i} q' \]

\[
\begin{array}{|c|c|c|}
\hline
\ldots & a & \ldots \\
\hline
1 & i & n \\
\hline
\end{array}
\]
Demo: \textit{known} transitions

$q \xrightarrow{\text{a}} q'$

\[
\begin{array}{c}
\ldots \ a \ldots \\
1 \ i \ n
\end{array}
\quad
\begin{array}{c}
\ldots \ a \ldots \\
1 \ i \ n
\end{array}
\]
Demo: locally fresh transitions

$q \xrightarrow{\cdot \bullet} q'$
Demo: *locally fresh* transitions

$\text{...}$

$q \xrightarrow{i} q'$

$\text{...}$

$1 \quad i \quad n$
Demo: locally fresh transitions

- Transition from state $q$ to $q'$ labeled with $i$.
Demo: *locally fresh* transitions

\[ q \xrightarrow{i} q' \]

\[ \cdots \mid i \mid \cdots \]

\[ \cdots \mid a \mid \cdots \]

\[ a \text{ fresh} \]
Demo: *globally fresh* transitions

\[ q \xrightarrow{i^*} q' \]
Demo: globally fresh transitions

\[
\begin{array}{c}
\cdots \\
1 \quad i \\
\cdots
\end{array}
\]
Demo: globally fresh transitions
Demo: globally fresh transitions
Demo: globally fresh transitions

An FMA is (equivalent to) an FRA without \((q, i^*, q') \in \delta\)
Demo: constant transitions

![Diagram showing a transition from q to q']
A fresh-register automaton (FRA) of $n$ registers is a quintuple $\mathcal{A} = \langle Q, q_0, \sigma_0, \delta, F \rangle$ where:

- $Q$ is a finite set of states,
- $q_0 \in Q$ is the initial state,
- $\sigma_0 \in \text{Reg}_n$ is the initial register assignment,
- $\delta \subseteq Q \times \mathbb{L}_n \times Q$ is the transition relation,
- $F \subseteq Q$ is the set of final states.

$$\mathcal{L}(\mathcal{A}) = \{ \vec{\ell} \in (\mathbb{A} \cup \mathbb{C})^* \mid (q_0, \sigma_0, \emptyset) \xrightarrow{\vec{\ell}'} (q, \sigma, H) \land q \in F \}$$
A name generator

\[ q_0 \]

\[ \sigma_0 = \{1 \mapsto \#\} \]
A name generator

$q_0 \xrightarrow{1^*} q_0 \quad \sigma_0 = \{1 \mapsto \#\}$
A name generator

\[ \sigma_0 = \{1 \mapsto \#\} \]
A name generator

\[ q_0 \xrightarrow{1^*} \]

\[ \sigma_0 = \{1 \mapsto \#\} \]

\[
\begin{array}{c}
q_0 \xrightarrow{a_1} q_0 \xrightarrow{a_2} q_0 \\
\# \xrightarrow{a_1} a_1 \xrightarrow{a_2} a_2
\end{array}
\]
A name generator

\[ q_0 \xrightarrow{1^*} \]

\[ \sigma_0 = \{ 1 \mapsto \# \} \]
A name generator

\[ L_1 \rightarrow q_0 \]

\[ \sigma_0 = \{ 1 \mapsto \# \} \]

\[ \mathcal{L}(A) = \{ a_1 \cdots a_k \in \mathbb{A}^* \mid \forall i \neq j. a_i \neq a_j \} \]
Another example
Properties

• Closure under union and intersection.
• Non-closure under concatenation and Kleene star.

E.g. \( L_1 \ast L_1 \) is not FRA-recognisable.
Properties

- Closure under union and intersection.
- Non-closure under concatenation and Kleene star.

E.g. $L_1 * L_1$ is not FRA-recognisable.

*Nominal versions of concatenation and Kleene star?*
Properties

- Closure under union and intersection.
- Non-closure under concatenation and Kleene star.
  E.g. $L_1 \cdot L_1$ is not FRA-recognisable.
- Non-closure under complementation.
  E.g. $\overline{L_1 \cdot L_1}$ is FRA-recognisable.
Bisimulations

• Recall: \( \hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(A) \)

• Let \( A_1, A_2 \) be FRA's. Consider relations \( R \subseteq \hat{Q}_1 \times \hat{Q}_2 \)
Bisimulations

- Recall: $\hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{fn}(\mathbb{A})$
- Let $A_1, A_2$ be FRA's. Consider relations $R \subseteq \hat{Q}_1 \times \hat{Q}_2$
Bisimulations

- Recall:  $\hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(\mathbb{A})$
- Let $\mathcal{A}_1, \mathcal{A}_2$ be FRA's. Consider relations $R \subseteq \hat{Q}_1 \times \hat{Q}_2$
Bisimulations

- Recall: $\hat{Q} = \mathcal{Q} \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(\mathcal{A})$

- Let $\mathcal{A}_1, \mathcal{A}_2$ be FRA's. Consider relations $R \subseteq \hat{Q}_1 \times \hat{Q}_2$
Bisimulations

- Recall: $\hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{fn}(\mathbb{A})$

- Let $\mathcal{A}_1, \mathcal{A}_2$ be FRA's. Consider relations $R \subseteq \hat{Q}_1 \times \hat{Q}_2$
Bisimulations

- Recall: $\hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(\mathcal{A})$
- Let $\mathcal{A}_1, \mathcal{A}_2$ be FRA's. Consider relations $R \subseteq \hat{Q}_1 \times \hat{Q}_2$
Bisimulations

- Recall: \( \hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{fn}(\mathcal{A}) \)
- Let \( \mathcal{A}_1, \mathcal{A}_2 \) be FRA's. Consider relations \( R \subseteq \hat{Q}_1 \times \hat{Q}_2 \)
Bisimulations

- Recall: \( \hat{Q} = Q \times \text{Reg}_n \times \mathcal{P}_{\text{fn}}(\Delta) \)
- Let \( A_1, A_2 \) be FRA's. Consider relations \( R \subseteq \hat{Q}_1 \times \hat{Q}_2 \)

Lemma. Bisimilarity implies language equivalence.
Bisimulations formally

$R \subseteq \hat{Q}_1 \times \hat{Q}_2$ is called a \textit{simulation} on $A_1$ and $A_2$ if, whenever $(q_1, \sigma_1, H_1) R (q_2, \sigma_2, H_2)$,

- if $q_1 \in F_1$ then $q_2 \in F_2$,

- if $(q_1, \sigma_1, H_1) \xrightarrow{\ell} (q'_1, \sigma'_1, H'_1)$ then there is $(q'_2, \sigma'_2, H'_2)$ with:

  $$(q_2, \sigma_2, H_2) \xrightarrow{\ell} (q'_2, \sigma'_2, H'_2)$$

  and

  $$(q'_1, \sigma'_1, H'_1) R (q'_2, \sigma'_2, H'_2).$$

$R$ is called a \textit{bisimulation} if both $R$ and $R^{-1}$ are simulations.
Bisimulations formally

$R \subseteq \hat{\mathcal{Q}}_1 \times \hat{\mathcal{Q}}_2$ is called a simulation on $\mathcal{A}_1$ and $\mathcal{A}_2$ if, whenever $(q_1, \sigma_1, H_1) R (q_2, \sigma_2, H_2)$,

- if $q_1 \in F_1$ then $q_2 \in F_2$,

- if $(q_1, \sigma_1, H_1) \xrightarrow{\ell} (q'_1, \sigma'_1, H'_1)$ then there is $(q'_2, \sigma'_2, H'_2)$ with:
  
  $(q_2, \sigma_2, H_2) \xrightarrow{\ell} (q'_2, \sigma'_2, H'_2)$ and $(q'_1, \sigma'_1, H'_1) R (q'_2, \sigma'_2, H'_2)$.

$R$ is called a bisimulation if both $R$ and $R^{-1}$ are simulations.

We say that $\mathcal{A}_1$ and $\mathcal{A}_2$ are bisimilar, written $\mathcal{A}_1 \sim \mathcal{A}_2$, if there is a bisimulation $R$ such that:

$(q_{01}, \sigma_{01}, \emptyset) R (q_{02}, \sigma_{02}, \emptyset)$
Example

\[ \sigma_0 = \{1 \mapsto \#\} \]

\[ \sigma_0 = \{1 \mapsto \#\} \]
Example

\[ \sigma_0 = \{ 1 \mapsto \# \} \]

\[ R = \{ ((q_0, \sigma_0, \emptyset), (q_0, \sigma_0, \emptyset)) \} \cup \{ ((q_0, \sigma_1, H), (q_1, \sigma_2, H)) \} \]
Another example
Closed FRA's

- \( A \) is closed if it has no blocking transitions:

\[
(q_0, \sigma_0, \emptyset) \xrightarrow{\ell} (q, \sigma, H) \land (q, i, q') \in \delta \implies \sigma(i) \neq \# 
\]
Closed FRA's

• $A$ is closed if it has no blocking transitions:

$$ (q_0, \sigma_0, \emptyset) \xrightarrow{\ell} (q, \sigma, H) \land (q, i, q') \in \delta \implies \sigma(i) \neq \# $$

**Lemma.** For any FRA $A$ we can effectively construct a closed FRA $\overline{A} \sim A$.

**Corollary.** FRA-emptiness is decidable.
Symbolic bisimulations

• Symbolic reasoning can be used for bisimulations too:
  • We can define a notion of symbolic bisimulation:

\[
R \subseteq Q_1 \times \{0, \ldots, n_1 + n_2\} \times (n_1 \models n_2) \times Q_2
\]

• capturing (actual) bisimilarity.
Symbolic bisimulations

- Symbolic reasoning can be used for bisimulations too:
  - We can define a notion of symbolic bisimulation:
    \[ R \subseteq Q_1 \times \{0, \ldots, n_1 + n_2\} \times (n_1 \leq n_2) \times Q_2 \]
  - Capturing (actual) bisimilarity.

**Corollary.** FRA-bisimilarity is decidable.
Results on FRA's

- As language acceptors:
  - Closure under union and intersection.
  - Non-closure under concatenation and Kleene star.
  - Non-closure under complementation.
  - Emptiness is decidable.
  - Containment, universality are undecidable.

- Bisimilarity is decidable by symbolic means.
Application: the pi-calculus
Application: the pi-calculus

| **INP1** | $\sigma \vdash a(b).P \xrightarrow{i} \sigma \vdash (b).P$ |
| **INP2A** | $\sigma \vdash (b).P \xrightarrow{i} \sigma \vdash P\{a/b\}$ |
| **OUT1** | $\sigma \vdash \tilde{a}b.P \xrightarrow{i} \sigma \vdash b.P$ |
| **RES** | $(\sigma + a) \vdash \hat{P} \xrightarrow{\alpha} (\sigma' + a) \vdash \hat{P}'$  |
| **PAR1** | $\sigma \vdash \hat{P} \xrightarrow{\alpha} \sigma \vdash \hat{P}'$  |
| **MATCH** | $\sigma \vdash P \xrightarrow{i} \sigma \vdash \hat{P'}$  |
| **SUM** | $\sigma \vdash P \xrightarrow{\alpha} \sigma \vdash \hat{P'}$  |
| **INP2B** | $\sigma \vdash (a = b).P \xrightarrow{i} \sigma \vdash P\{a/b\}$  |
| **OUT2** | $\sigma \vdash b.P \xrightarrow{i} \sigma \vdash P$  |
| **REC** | $\sigma \vdash P\{\tilde{a}/\tilde{b}\} \xrightarrow{i} \sigma \vdash \hat{P}'$  |
| **OPEN** | $\sigma \vdash a(\hat{P}\{\tilde{a}/\tilde{b}\}) \xrightarrow{i} \sigma \vdash P$  |
| **PAR2** | $\sigma \vdash \nu a.\hat{P} \xrightarrow{i} \sigma \vdash \nu a.\hat{P}'$  |
| **CLOSE** | $\sigma \vdash P \xrightarrow{ij} (b + \sigma) \vdash P'$  |
| **DBLOUT** | $\sigma \vdash P \xrightarrow{i} \sigma \vdash P_{\text{out}} \xrightarrow{ij} \sigma' \vdash P'$  |
| **DBLINP** | $\sigma \vdash P \xrightarrow{i} \sigma \vdash P_{\text{inp}} \xrightarrow{ij} \sigma' \vdash P'$  |
| **DUBLINP** | $\sigma \vdash P \xrightarrow{ij} \sigma \vdash P'$  |

| **Table 1.** The transition relation for the $x\pi$-calculus (symmetric counterparts of SUM, PAR, COMM, CLOSE omitted).
Application: the pi-calculus

\[ \sigma \vdash a(b).P \xrightarrow{i} \sigma \vdash (b).P \]

\[ \sigma \vdash (b).P \xrightarrow{i} \sigma \vdash P\{a/b\} \]

\[ \sigma \vdash (b).P \xrightarrow{i} \sigma[i \mapsto b] \vdash P \]

Table 1. The transition relation for the $\pi$-calculus (symmetric counterparts of SUM, PAR, COMM, CLOSE omitted).
Application: the pi-calculus

Table 1. The transition relation for the $\pi\pi$-calculus (symmetric counterparts of SUM, PAR, COMM, CLOSE omitted).
Application: the pi-calculus

\[ P(a) = \nu b. \overline{a}b. P(b) \]
Application: the pi-calculus

- Algorithmic description which is:
  - name-free
  - finitely-branching
- Bisimilarity can be captured symbolically
  - In the finitary case: decide bisimilarity
Interesting directions

• *Algorithmic game semantics*
  – e.g. (finitary) Reduced ML
• Connections to HD-automata
• Nominal notions of concatenation, Kleene closure
• Variations:
  – Labels
  – Stores
  – Stack