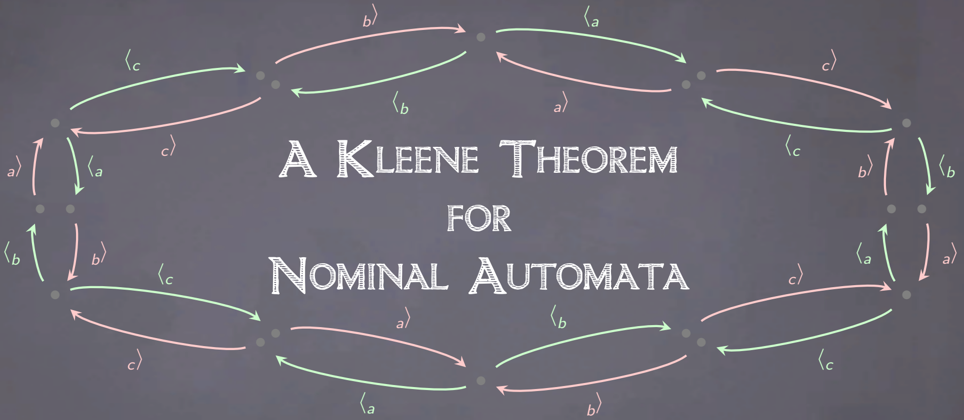


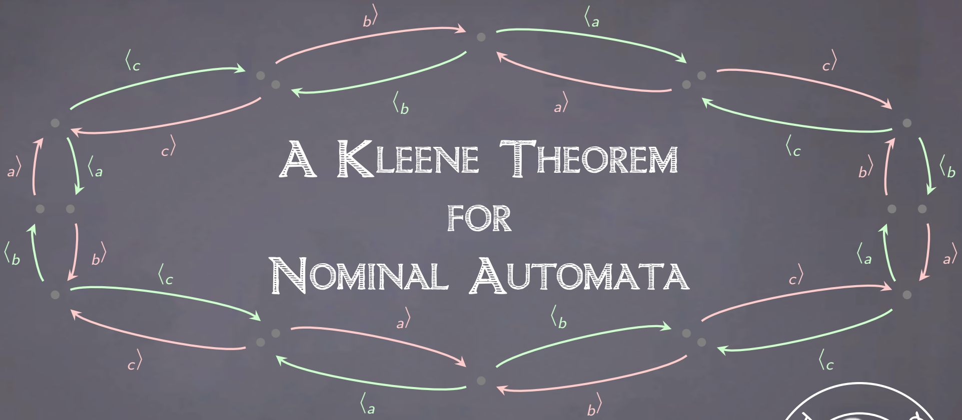
A Kleene Theorem for Nominal Automata

ICALP, July 2019

Paul Brunet, Alexandra Silva

University College London

UCL



A Kleene Theorem for Nominal Automata

ICALP, July 2019

Paul Brunet, Alexandra Silva

University College London

Proved and developped in Coq

Data languages

Sets of words over an infinite alphabet.

- ☞ query-based languages
- ☞ XML processing
- ☞ URLs
- ☞ process-calculi
- ☞ ...

Such an alphabet may be respresented as a Nominal set.

This paper

Data languages

Operational semantics:

testing, algorithms

Syntax:

specification, other algorithms

Nominal automata

This paper

Data languages

Operational semantics:

testing, algorithms

Syntax:

specification, other algorithms

Nominal automata

Regular expressions with brackets

Outline

I. Nominal automata

II. Brackets

III. Kleene Theorem

Nominal sets

Notations

$a, b, c, \dots \in \mathbb{A}$

Transposition $(a\ b) : \mathbb{A} \to \mathbb{A}$

$$c \mapsto \begin{cases} a & \text{if } c = b, \\ b & \text{if } c = a, \\ c & \text{otherwise.} \end{cases}$$

Nominal sets

Notations

$a, b, c, \dots \in \mathbb{A}$

$\pi, \pi', \dots \in \mathfrak{S}_{\mathbb{A}}$

Transposition
$$(a\ b) : \mathbb{A} \to \mathbb{A}$$
$$c \mapsto \begin{cases} a & \textit{if } c = b, \\ b & \textit{if } c = a, \\ c & \textit{otherwise}. \end{cases}$$

(Finitely supported) permutation: composition of transpositions.

Nominal sets

Notations
$a, b, c, \dots \in \mathbb{A}$
$\pi, \pi', \dots \in \mathfrak{S}_{\mathbb{A}}$
$x, y, z, \dots \in \mathbb{X}$

Transposition

$$(a\ b) : \mathbb{A} \to \mathbb{A}$$
$$c \mapsto \begin{cases} a & \text{if } c = b, \\ b & \text{if } c = a, \\ c & \text{otherwise.} \end{cases}$$

(Finitely supported) permutation: composition of transpositions.

Nominal set

A nominal set is a set $\mathbb{X}$ with two functions

$$\cdot : \mathfrak{S}_{\mathbb{A}} \times \mathbb{X} \to \mathbb{X} \text{ and } \mathbf{supp}(\_) : \mathbb{X} \to \mathcal{P}_f(\mathbb{A})$$

such that:

1) $\pi \cdot (\pi' \cdot x) = \pi \circ \pi' \cdot x$;
2) $\mathbf{supp}(\pi \cdot x) = \pi \cdot \mathbf{supp}(x)$;
3) $(\forall a \in \mathbf{supp}(x), \pi(a) = a) \Rightarrow \pi \cdot x = x$.

Orbit-finite nominal sets

Orbits

- ☞ $x \sim_{\mathcal{O}} y$ iff $\exists \pi, \pi \cdot x = y$.
- ☞ Orbit of x $:= \{\pi \cdot x \mid \pi \in \mathfrak{S}_{\mathbb{A}}\} = [x]_{\sim_{\mathcal{O}}}$.

Orbit-finite nominal sets

Orbits

- $x \sim_{\mathcal{O}} y$ iff $\exists \pi, \pi \cdot x = y$.
- Orbit of x $:= \{\pi \cdot x \mid \pi \in \mathfrak{S}_{\mathbb{A}}\} = [x]_{\sim_{\mathcal{O}}}$.

A nominal set is orbit-finite if it has finitely many orbits, i.e. $\sim_{\mathcal{O}}$ has finite index.

Orbit-finite nominal sets

Orbits

- $x \sim_{\mathcal{O}} y$ iff $\exists \pi, \pi \cdot x = y$.
- Orbit of x $:= \{\pi \cdot x \mid \pi \in \mathfrak{S}_{\mathbb{A}}\} = [x]_{\sim_{\mathcal{O}}}$.

A nominal set is orbit-finite if it has finitely many orbits, i.e. $\sim_{\mathcal{O}}$ has finite index.

Tractable subsets

$A \subseteq \mathbb{X}$ is tractable if

1) A intersects finitely many orbits
2) A is supported by a finite set $S \subseteq \mathbb{A}$, meaning

$$(\forall a \in S, \pi(a) = a) \Rightarrow \forall x, x \in A \Leftrightarrow \pi \cdot x \in A.$$

NFA
Non-deterministic Finite Automata

$$\mathscr{A} := \langle Q, \Sigma, \Delta, I, F \rangle$$

Where:

- ☞ Q finite set of states
- ☞ Σ finite alphabet
- ☞ $\Delta \subseteq Q \times \Sigma \times Q$ finite transition relation
- ☞ $I, F \subseteq Q$ finite sets of initial/final states

NOFA
Non-deterministic Orbit-Finite Automata

$$\mathscr{A} := \langle Q, \Sigma, \Delta, I, F\rangle$$

Where:

- ☞ Q tractable set of states
- ☞ Σ tractable alphabet
- ☞ $\Delta \subseteq Q \times \Sigma \times Q$ tractable transition relation
- ☞ $I, F \subseteq Q$ tractable sets of initial/final states

Bojańczyk, Klin & Lasota, "Automata theory in nominal sets", LMCS 2014

Deterministic vs. Non-deterministic

Theorem

NOFA are stricIty more expressive than their deterministic counterparts.

Theorem

Equivalence of NOFA is undecidable.

Outline

I. Nominal automata

II. Brackets

III. Kleene Theorem

Traces with brackets

Given a nominal alphabet $\mathbb{X}$, we build a nominal set of symbols:

$$\Sigma := \mathbb{X} \cup \{\langle_a \mid a \in \mathbb{A}\} \cup \{{}_a\rangle \mid a \in \mathbb{A}\}.$$

Traces with brackets

Given a nominal alphabet $\mathbb{X}$, we build a nominal set of symbols:

$$\Sigma := \mathbb{X} \cup \{\langle_a \mid a \in \mathbb{A}\} \cup \{{}_a\rangle \mid a \in \mathbb{A}\}.$$

We generate words over $\mathbb{X}$ from traces, i.e. words over Σ.

Transducer

$s, s' \in (\mathbb{A} \times \mathbb{A})^\star$

$$\begin{aligned}
s &-[\langle_a /\varepsilon]\rightarrow_{\mathcal{T}} s :: (a,b) && \text{(if } b \notin \pi_2 s\text{)}\\
s &-[x/\pi \cdot x]\rightarrow_{\mathcal{T}} s && \text{(if } \forall a \in \mathbf{supp}(x), \pi(a) = s(a)\text{)}\\
s :: (a,b) :: s' &-[{}_a\rangle / {}_b\rangle]\rightarrow_{\mathcal{T}} s :: s' && \text{(if } a \notin \pi_1 s' \wedge b \notin \pi_2 s'\text{)}
\end{aligned}$$

$$\mathcal{L}(u) = \{v \mid \bot -[u/v]\rightarrow_{\mathcal{T}} \bot\}.$$

First Languages from words with brackets

Example

Trace:

$\langle_a\, a \langle_b\, b\ {}_a\rangle \langle_a\, a\ {}_b\rangle\ {}_a\rangle$

Stack:

$\perp$

Output:

First Languages from words with brackets

Example

Trace:

$\langle_a\, a \langle_b\, b\, {}_a\rangle \langle_a\, a\, {}_b\rangle\, {}_a\rangle$

Stack:

$a \mapsto x$

$\perp$

Output:

First Languages from words with brackets

Example

Trace:

$\underline{\langle_a\, a}\langle_b\, b\, {}_a\rangle \langle_a\, a\, {}_b\rangle\, {}_a\rangle$

Stack:

$a \mapsto x$

$\perp$

Output:

x

First Languages from words with brackets

Example

Trace:

$\underline{\langle_a\, a \langle_b}\, b\, {}_a\rangle \langle_a\, a\, {}_b\rangle\, {}_a\rangle$

Stack:

$b \mapsto y$

$a \mapsto x$

$\perp$

Output:

x

First Languages from words with brackets

Example

Trace:

$\underline{\langle_a a \langle_b b}\ {}_a\rangle \langle_a a\ {}_b\rangle\ {}_a\rangle$

Stack:

$b \mapsto y$

$a \mapsto x$

$\perp$

Output:

xy

First Languages from words with brackets

Example

Trace:

$\underline{\langle_a\, a \langle_b\, b\ {}_a\rangle}\, \langle_a\, a\ {}_b\rangle\ {}_a\rangle$

Stack:

$\boxed{b \mapsto y}$

$\perp$

Output:

xy

First Languages from words with brackets

Example

Trace:

$\underline{\langle_a\, a \langle_b\, b\, {}_a\rangle \langle_a}\, a\, {}_b\rangle\, {}_a\rangle$

Stack:

$a \mapsto z$

$b \mapsto y$

$\perp$

Output:

xy

First Languages from words with brackets

Example

Trace:

$$\underline{\langle_a\, a \langle_b\, b\, {}_a\rangle \langle_a\, a}\; {}_b\rangle\; {}_a\rangle$$

Stack:

$a \mapsto z$

$b \mapsto y$

$\perp$

Output:

xyz

First Languages from words with brackets

Example

Trace:

$\underline{\langle_a\, a \langle_b\, b\; {}_a\rangle \langle_a\, a\; {}_b\rangle}\; {}_a\rangle$

Stack:

$\boxed{a \mapsto z}$

$\perp$

Output:

xyz

First Languages from words with brackets

Example

Trace:

$\langle_a\, a \langle_b\, b\, {}_a\rangle \langle_a\, a\, {}_b\rangle\, {}_a\rangle$

Stack:

$\perp$

Output:

xyz

Languages from words with brackets

Second Example

Trace:

$\langle_a\, a \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle\, {}_a\rangle$

Stack:

$\perp$

Output:

Languages from words with brackets
Second Example

Trace:

$\underline{\langle_a}\, a \langle_a a \,{}_a\rangle \langle_a a \,{}_a\rangle \langle_a a \,{}_a\rangle \,{}_a\rangle$

Stack:

$a \mapsto x$

$\perp$

Output:

Languages from words with brackets

Second Example

Trace:

$\langle_a\, a \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle\, {}_a\rangle$

Stack:

$a \mapsto x$

$\perp$

Output:

x

Languages from words with brackets

Second Example

Trace:

$\underline{\langle_a\, a \langle_a}\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle\, {}_a\rangle$

Stack:

$a \mapsto y$

$a \mapsto x$

$\perp$

Output:

x

Languages from words with brackets
Second Example

Trace:

$\underline{\langle_a\, a \langle_a\, a}\ {}_a\rangle \langle_a\, a\ {}_a\rangle \langle_a\, a\ {}_a\rangle\ {}_a\rangle$

Stack:

$a \mapsto y$

$a \mapsto x$

$\perp$

Output:

xy

Languages from words with brackets

Second Example

Trace:

$$\underline{\langle_a a \langle_a a \,{}_a\rangle} \langle_a a \,{}_a\rangle \langle_a a \,{}_a\rangle \,{}_a\rangle$$

Stack:

$\boxed{a \mapsto x}$

$\perp$

Output:

xy

Languages from words with brackets

Second Example

Trace:

$\langle_a\, a \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle\, {}_a\rangle$

Stack:

$a \mapsto y$

$a \mapsto x$

$\perp$

Output:

xy

Languages from words with brackets

Second Example

Trace:

$$\underline{\langle_a\, a \langle_a\, a\; {}_a\rangle \langle_a\, a}\; {}_a\rangle \langle_a\, a\; {}_a\rangle\; {}_a\rangle$$

Stack:

$a \mapsto y$

$a \mapsto x$

$\perp$

Output:

xyy

Languages from words with brackets
Second Example

Trace:

$\underline{\langle_a\, a \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle} \langle_a\, a\, {}_a\rangle\, {}_a\rangle$

Stack:

$a \mapsto x$

$\perp$

Output:

xyy

Languages from words with brackets
Second Example

Trace:

$$\underline{\langle_a a \langle_a a\ {}_a\rangle \langle_a a\ {}_a\rangle \langle_a} a\ {}_a\rangle\ {}_a\rangle$$

Stack:

$a \mapsto z$

$a \mapsto x$

$\perp$

Output:

xyy

Languages from words with brackets
Second Example

Trace:

$$\underline{\langle_a a \langle_a a\ {}_a\rangle \langle_a a\ {}_a\rangle \langle_a a}\ {}_a\rangle\ {}_a\rangle$$

Stack:

$\boxed{a \mapsto z}$

$\boxed{a \mapsto x}$

$\bot$

Output:

$xyyz$

Languages from words with brackets
Second Example

Trace:

$$\underline{\langle_a\, a \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle}\ {}_a\rangle$$

Stack:

$a \mapsto x$

$\perp$

Output:

$xyyz$

Languages from words with brackets

Second Example

Trace:

$\langle_a\, a \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle \langle_a\, a\, {}_a\rangle\, {}_a\rangle$

Stack:

$\perp$

Output:

$xyyz$

Regular expressions with brackets

$e \in \mathrm{Reg}\langle\Sigma\rangle$,

$$\mathcal{L}(e) := \bigcup_{u \in [\![e]\!]} \mathcal{L}(u).$$

where $[\![e]\!]$ is the regular language (in the classical sense) associated with e.

Regular expressions with brackets

$e \in \mathrm{Reg}\langle\Sigma\rangle$,

$$\mathcal{L}(e) := \bigcup_{u \in [\![e]\!]} \mathcal{L}(u).$$

where $[\![e]\!]$ is the regular language (in the classical sense) associated with e.

Memory finiteness

An expression e is memory finite if there is a bound $N \in \mathbb{N}$ such that if $uv \in [\![e]\!]$, u has at most N unmatched $\langle_a$.

Outline

I. Nominal automata

II. Brackets

III. Kleene Theorem

From expressions to automata

Theorem

For every memory-finite expression e there is a NOFA $\mathscr{A}$ such that $\mathcal{L}(e) = \mathcal{L}(\mathscr{A})$.

Idea of the proof: compose the NFA for e with the tranducer $\mathcal{T}$.

From automata to expressions

Theorem

For every NOFA $\mathscr{A}$ there is a memory-finite expression e such that $\mathcal{L}(e) = \mathcal{L}(\mathscr{A})$.

Idea of the proof:

From automata to expressions

Theorem

For every NOFA $\mathscr{A}$ there is a memory-finite expression e such that $\mathcal{L}(e) = \mathcal{L}(\mathscr{A})$.

Idea of the proof:

☞ pick a finite representative (NFA) of $\mathscr{A}$;

From automata to expressions

Theorem

For every NOFA $\mathscr{A}$ there is a memory-finite expression e such that $\mathcal{L}(e) = \mathcal{L}(\mathscr{A})$.

Idea of the proof:

- pick a finite representative (NFA) of $\mathscr{A}$;
- transform transitions:

$$p \xrightarrow{\quad x \quad} q$$

where
$$\begin{aligned}\mathbf{supp}(p) \setminus \mathbf{supp}(q) &= \{a_1 \dots a_n\}\\ \mathbf{supp}(q) \setminus \mathbf{supp}(p) &= \{b_1 \dots b_m\}.\end{aligned}$$

From automata to expressions

Theorem

For every NOFA $\mathscr{A}$ there is a memory-finite expression e such that $\mathcal{L}(e) = \mathcal{L}(\mathscr{A})$.

Idea of the proof:

- pick a finite representative (NFA) of $\mathscr{A}$;
- transform transitions:

$$(p) \xrightarrow{\quad x \quad} (q)$$

$$(p) \xrightarrow{\langle_{b_1} \cdots \langle_{b_m} x\ {}_{a_1}\rangle \cdots {}_{a_n}\rangle} (q)$$

where
$$\begin{aligned} \mathbf{supp}(p) \setminus \mathbf{supp}(q) &= \{a_1 \dots a_n\} \\ \mathbf{supp}(q) \setminus \mathbf{supp}(p) &= \{b_1 \dots b_m\}. \end{aligned}$$

From automata to expressions

Theorem

For every NOFA $\mathscr{A}$ there is a memory-finite expression e such that $\mathcal{L}(e) = \mathcal{L}(\mathscr{A})$.

Idea of the proof:

- ☞ pick a finite representative (NFA) of $\mathscr{A}$;
- ☞ transform transitions:

$$p \xrightarrow{\qquad\qquad x \qquad\qquad} q$$

$$p \xrightarrow{\qquad \langle_{b_1} \cdots \langle_{b_m} x\ {}_{a_1}\rangle \cdots {}_{a_n}\rangle \qquad} q$$

where
$$\begin{aligned} \mathbf{supp}(p) \setminus \mathbf{supp}(q) &= \{a_1 \ldots a_n\} \\ \mathbf{supp}(q) \setminus \mathbf{supp}(p) &= \{b_1 \ldots b_m\}. \end{aligned}$$

- ☞ extract an expression from the NFA.

That's all folks!

Thank you!

See more at:
http://paul.brunet-zamansky.fr