# Intersection Type Systems Corresponding to Nominal Automata

## UdS Qualifying Exam / MFoCS Dissertation

Dominik Kirst
Supervised by Steven Ramsay and Luke Ong

Lady Margaret Hall
University of Oxford

November 23, 2016

# Outline

# Outline

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

Then we clearly have $w \in \mathcal{L}(\mathcal{A}, q_I)$ for the initial state $q_I$ of $\mathcal{A}$.

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

Then we clearly have $w \in \mathcal{L}(\mathcal{A}, q_I)$ for the initial state $q_I$ of $\mathcal{A}$.

If we translate $\mathcal{A}$ into a type system, we can have more:

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

Then we clearly have $w \in \mathcal{L}(\mathcal{A}, q_I)$ for the initial state $q_I$ of $\mathcal{A}$.

If we translate $\mathcal{A}$ into a type system, we can have more:

- $\vdash w : q_I$ with $w$ interpreted as term of the system language

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

Then we clearly have $w \in \mathcal{L}(\mathcal{A}, q_I)$ for the initial state $q_I$ of $\mathcal{A}$.

If we translate $\mathcal{A}$ into a type system, we can have more:

- $\vdash w : q_I$ with $w$ interpreted as term of the system language
- $\vdash (\lambda x.xbcx)a : q_I$ for a simple program computing $w$

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

Then we clearly have $w \in \mathcal{L}(\mathcal{A}, q_I)$ for the initial state $q_I$ of $\mathcal{A}$.

If we translate $\mathcal{A}$ into a type system, we can have more:

- $\vdash w : q_I$ with $w$ interpreted as term of the system language
- $\vdash (\lambda x.xbcx)a : q_I$ for a simple program computing $w$
- $\vdash K((\lambda x.xbcx)a)\Omega : q_I$ for a more complex program

# Example

Consider the word $w := abca$ over the alphabet $\Sigma := \{a, b, c, d\}$.

Let $\mathcal{A}$ be an NFA over $\Sigma$ that accepts the language:

$$\mathcal{L} := \{w \in \Sigma^* \mid w \text{ contains two } a\}$$

Then we clearly have $w \in \mathcal{L}(\mathcal{A}, q_I)$ for the initial state $q_I$ of $\mathcal{A}$.

If we translate $\mathcal{A}$ into a type system, we can have more:

- $\vdash w : q_I$ with $w$ interpreted as term of the system language
- $\vdash (\lambda x.xbcx)a : q_I$ for a simple program computing $w$
- $\vdash K((\lambda x.xbcx)a)\Omega : q_I$ for a more complex program
- Theorem: $\forall \Gamma, s, q.\ \Gamma \vdash s : q \iff \exists n.\, s \Downarrow n \wedge n \in \mathcal{L}(\mathcal{A}, q)$

# Motivation

# Motivation

...for type systems corresponding to automata:

- First appearance in higher-order model checking[1]
- Acceptance reduces to type checking, allowing new algorithms
- Generalised acceptance of programs evaluating to words/trees

# Motivation

...for type systems corresponding to automata:

- First appearance in higher-order model checking[1]
- Acceptance reduces to type checking, allowing new algorithms
- Generalised acceptance of programs evaluating to words/trees

$\Longrightarrow$ Contribute a self-contained presentation

# Motivation

...for type systems corresponding to automata:

- First appearance in higher-order model checking[1]
- Acceptance reduces to type checking, allowing new algorithms
- Generalised acceptance of programs evaluating to words/trees

$\Longrightarrow$ Contribute a self-contained presentation

...for nominal automata[2]:

- Well-behaved automata over infinite alphabets
- Equivariant properties independent of concrete names

# Motivation

...for type systems corresponding to automata:

- First appearance in higher-order model checking[1]
- Acceptance reduces to type checking, allowing new algorithms
- Generalised acceptance of programs evaluating to words/trees

$\Longrightarrow$ Contribute a self-contained presentation

...for nominal automata[2]:

- Well-behaved automata over infinite alphabets
- Equivariant properties independent of concrete names

$\Longrightarrow$ Contribute new instances for new automaton models

# Outline

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st$

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to$

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$ terminates, we write $(\lambda x.x)y \Downarrow y$ and call $y$ normal

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$ terminates, we write $(\lambda x.x)y \Downarrow y$ and call $y$ normal
$(\lambda x.xx)(\lambda x.xx) \to$

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$ terminates, we write $(\lambda x.x)y \Downarrow y$ and call $y$ normal
$(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) \to \dots$

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$ terminates, we write $(\lambda x.x)y \Downarrow y$ and call $y$ normal
$(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) \to \ldots$ diverges

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$ terminates, we write $(\lambda x.x)y \Downarrow y$ and call $y$ normal
$(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) \to \dots$ diverges

Can encode booleans, natural numbers and recursion, hence:

# Lambda Calculus[3]

Minimal (functional) programming language defined by...

- Term language: $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st$
- Reduction: $(\lambda x.s)t \to s[t/x]$ + syntactic closure

$(\lambda x.x)y \to y$ terminates, we write $(\lambda x.x)y \Downarrow y$ and call $y$ normal
$(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) \to \ldots$ diverges

Can encode booleans, natural numbers and recursion, hence:

**Theorem**

The (untyped) lambda calculus is Turing complete.

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \mathsf{TVar} \mid A \to B$

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \mathrm{TVar} \mid A \to B$
- Typing rules:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash st : B}$$

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \mathsf{TVar} \mid A \to B$
- Typing rules:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash st : B}$$

...implying the following properties:

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \text{TVar} \mid A \to B$
- Typing rules:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash st : B}$$

...implying the following properties:

- Subject Reduction: $s \to t \Longrightarrow \Gamma \vdash s : A \Longrightarrow \Gamma \vdash t : A$

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \mathrm{TVar} \mid A \to B$
- Typing rules:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash st : B}$$

...implying the following properties:

- Subject Reduction: $s \to t \Longrightarrow \Gamma \vdash s : A \Longrightarrow \Gamma \vdash t : A$
- Strong Normalisation: $\Gamma \vdash s : A \Longrightarrow$ all reductions terminate

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \mathrm{TVar} \mid A \to B$
- Typing rules:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash st : B}$$

...implying the following properties:

- Subject Reduction: $s \to t \implies \Gamma \vdash s : A \implies \Gamma \vdash t : A$
- Strong Normalisation: $\Gamma \vdash s : A \implies$ all reductions terminate
- Decidability of type checking, typability and type inhabitance

# Simply Typed Lambda Calculus[4]

Introduce typing judgements $\Gamma \vdash s : A$ by...

- Type language: $A, B ::= a \in \mathrm{TVar} \mid A \to B$
- Typing rules:

$$\frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash s : B}{\Gamma \vdash \lambda x.s : A \to B} \qquad \frac{\Gamma \vdash s : A \to B \quad \Gamma \vdash t : A}{\Gamma \vdash st : B}$$

...implying the following properties:

- Subject Reduction: $s \to t \implies \Gamma \vdash s : A \implies \Gamma \vdash t : A$
- Strong Normalisation: $\Gamma \vdash s : A \implies$ all reductions terminate
- Decidability of type checking, typability and type inhabitance

However, some normal forms like $\lambda x.xx$ are untybable...

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \quad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \quad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \qquad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \qquad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

Then the term $\lambda x.xx$ can be assigned a type:

$$\dfrac{\dfrac{[x : \bigwedge \emptyset \to A] \vdash x : \bigwedge \emptyset \to A}{[x : \bigwedge \emptyset \to A] \vdash xx : A}}{\vdash \lambda x.xx : (\bigwedge \emptyset \to A) \to A}$$

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \quad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \quad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

Then the term $\lambda x.xx$ can be assigned a type:

$$\cfrac{\cfrac{[x : \bigwedge \emptyset \to A] \vdash x : \bigwedge \emptyset \to A}{[x : \bigwedge \emptyset \to A] \vdash xx : A}}{\vdash \lambda x.xx : (\bigwedge \emptyset \to A) \to A}$$

In general, the intersection type system satisfies:

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \qquad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \qquad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

Then the term $\lambda x.xx$ can be assigned a type:

$$\cfrac{\cfrac{[x : \bigwedge \emptyset \to A] \vdash x : \bigwedge \emptyset \to A}{[x : \bigwedge \emptyset \to A] \vdash xx : A}}{\vdash \lambda x.xx : (\bigwedge \emptyset \to A) \to A}$$

In general, the intersection type system satisfies:

- Subject Expansion: $s \to t \implies \Gamma \vdash t : A \implies \Gamma \vdash s : A$

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \qquad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \qquad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

Then the term $\lambda x.xx$ can be assigned a type:

$$\dfrac{\dfrac{[x : \bigwedge \emptyset \to A] \vdash x : \bigwedge \emptyset \to A}{[x : \bigwedge \emptyset \to A] \vdash xx : A}}{\vdash \lambda x.xx : (\bigwedge \emptyset \to A) \to A}$$

In general, the intersection type system satisfies:

- Subject Expansion: $s \to t \implies \Gamma \vdash t : A \implies \Gamma \vdash s : A$
- (Weak) Normalisation: $\Gamma \vdash s : A \implies \exists n.\, s \Downarrow n$

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \qquad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \qquad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

Then the term $\lambda x.xx$ can be assigned a type:

$$\frac{\dfrac{[x : \bigwedge \emptyset \to A] \vdash x : \bigwedge \emptyset \to A}{[x : \bigwedge \emptyset \to A] \vdash xx : A}}{\vdash \lambda x.xx : (\bigwedge \emptyset \to A) \to A}$$

In general, the intersection type system satisfies:

- Subject Expansion: $s \to t \implies \Gamma \vdash t : A \implies \Gamma \vdash s : A$
- (Weak) Normalisation: $\Gamma \vdash s : A \implies \exists n.\, s \Downarrow n$
- Typability: $s \Downarrow n \implies \exists \Gamma, A.\, \Gamma \vdash s : A$

# Intersection Types[5]

Introduce (finite) type intersections $A \wedge B$ with rules:

$$\frac{\Gamma \vdash s : A \quad \Gamma \vdash s : B}{\Gamma \vdash s : A \wedge B} \qquad \frac{\Gamma \vdash s : \bigwedge_i A_i}{\Gamma \vdash s : A_i} \qquad \frac{\Gamma \vdash s : (\bigwedge_i A_i) \to B \quad \Gamma \vdash t : A_i}{\Gamma \vdash st : B}$$

Then the term $\lambda x.xx$ can be assigned a type:

$$\cfrac{\cfrac{[x : \bigwedge \emptyset \to A] \vdash x : \bigwedge \emptyset \to A}{[x : \bigwedge \emptyset \to A] \vdash xx : A}}{\vdash \lambda x.xx : (\bigwedge \emptyset \to A) \to A}$$

In general, the intersection type system satisfies:

- Subject Expansion: $s \to t \implies \Gamma \vdash t : A \implies \Gamma \vdash s : A$
- (Weak) Normalisation: $\Gamma \vdash s : A \implies \exists n.\, s \Downarrow n$
- Typability: $s \Downarrow n \implies \exists \Gamma, A.\, \Gamma \vdash s : A$

However, general type checking and typability become undecidable...

# Outline

# Nominal Sets[6]

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

We define the following:

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

We define the following:

- $x \in X$ has finite support if it is fixed by a finite $A \subset \mathbb{A}$, that is, $\pi \cdot x = x$ whenever $\pi|_A = \mathrm{id}_A$

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

We define the following:

- $x \in X$ has finite support if it is fixed by a finite $A \subset \mathbb{A}$, that is, $\pi \cdot x = x$ whenever $\pi|_A = \mathrm{id}_A$
- $X$ is nominal if every $x \in X$ is finitely supported

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

We define the following:

- $x \in X$ has finite support if it is fixed by a finite $A \subset \mathbb{A}$, that is, $\pi \cdot x = x$ whenever $\pi|_A = \mathrm{id}_A$
- $X$ is nominal if every $x \in X$ is finitely supported
- $X$ is orbit-finite if there exist only finitely many $\mathrm{Perm}(\mathbb{A}) \cdot x$

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

We define the following:

- $x \in X$ has finite support if it is fixed by a finite $A \subset \mathbb{A}$, that is, $\pi \cdot x = x$ whenever $\pi|_A = \mathrm{id}_A$
- $X$ is nominal if every $x \in X$ is finitely supported
- $X$ is orbit-finite if there exist only finitely many $\mathrm{Perm}(\mathbb{A}) \cdot x$
- Subsets $Y \subseteq X$ are equivariant if $\mathrm{Perm}(\mathbb{A}) \cdot Y = Y$

# Nominal Sets[6]

Let $\mathbb{A}$ be a countable set of atomic names.
Then consider sets $X$ with actions $\cdot : \mathrm{Perm}(\mathbb{A}) \times X \to X$ such that

$$\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x \qquad \mathrm{id} \cdot x = x$$

We define the following:

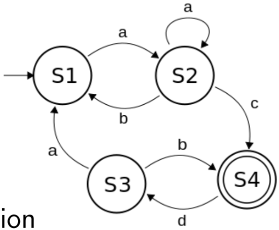
- $x \in X$ has finite support if it is fixed by a finite $A \subset \mathbb{A}$, that is, $\pi \cdot x = x$ whenever $\pi|_A = \mathrm{id}_A$
- $X$ is nominal if every $x \in X$ is finitely supported
- $X$ is orbit-finite if there exist only finitely many $\mathrm{Perm}(\mathbb{A}) \cdot x$
- Subsets $Y \subseteq X$ are equivariant if $\mathrm{Perm}(\mathbb{A}) \cdot Y = Y$

Examples: $\mathbb{A}$ itself, (finite) syntax over $\mathbb{A}$, singleton sets etc.

# Finite Automata

A finite automaton $\mathcal{A}$ consists of:

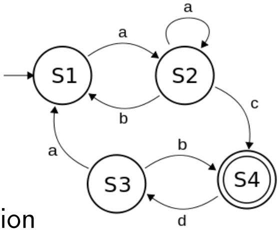
- $\Sigma$, a finite alphabet
- $Q$, a finite set of states
- $I \subseteq Q$, a finite subset of initial states
- $F \subseteq Q$, a finite subset of final states
- $\delta \subseteq Q \times A \times Q$, a finite transition relation



# Finite Automata

A finite automaton $\mathcal{A}$ consists of:

- $\Sigma$, a finite alphabet
- $Q$, a finite set of states
- $I \subseteq Q$, a finite subset of initial states
- $F \subseteq Q$, a finite subset of final states
- $\delta \subseteq Q \times A \times Q$, a finite transition relation



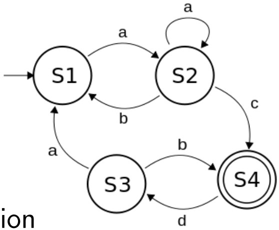

We write:

- $q \xrightarrow{a} q'$ for $a \in \Sigma$, $q, q' \in Q$ and $(q, a, q') \in \delta$
- $q \xrightarrow{w} q'$ for $w \in \Sigma^*$ and the reflexive-transitive closure of $\delta$
- $w \in \mathcal{L}(\mathcal{A})$ for $q \xrightarrow{w} q'$ with $q \in I$ and $q' \in F$

# Finite Automata

A finite automaton $\mathcal{A}$ consists of:

- $\Sigma$, a finite alphabet
- $Q$, a finite set of states
- $I \subseteq Q$, a finite subset of initial states
- $F \subseteq Q$, a finite subset of final states
- $\delta \subseteq Q \times A \times Q$, a finite transition relation




We write:

- $q \xrightarrow{a} q'$ for $a \in \Sigma$, $q, q' \in Q$ and $(q, a, q') \in \delta$
- $q \xrightarrow{w} q'$ for $w \in \Sigma^*$ and the reflexive-transitive closure of $\delta$
- $w \in \mathcal{L}(\mathcal{A})$ for $q \xrightarrow{w} q'$ with $q \in I$ and $q' \in F$

Example language: all words that contain at least two $a \in \Sigma$

# Nominal Automata[2]

A nominal automaton $\mathcal{A}$ over names $\mathbb{A}$ consists of:

# Nominal Automata[2]

A nominal automaton $\mathcal{A}$ over names $\mathbb{A}$ consists of:

- $\Sigma$, an orbit-finite nominal alphabet
- $Q$, an orbit-finite nominal set of states
- $I \subseteq Q$, an equivariant subset of initial states
- $F \subseteq Q$, an equivariant subset of final states
- $\delta \subseteq Q \times A \times Q$, an equivariant transition relation

# Nominal Automata[2]

A nominal automaton $\mathcal{A}$ over names $\mathbb{A}$ consists of:

- $\Sigma$, an orbit-finite nominal alphabet
- $Q$, an orbit-finite nominal set of states
- $I \subseteq Q$, an equivariant subset of initial states
- $F \subseteq Q$, an equivariant subset of final states
- $\delta \subseteq Q \times A \times Q$, an equivariant transition relation

We write:

- $q \xrightarrow{a} q'$ for $(q, a, q') \in \delta$
- $q \xrightarrow{w} q'$ for $w \in \Sigma^*$ and the reflexive-transitive closure of $\delta$
- $w \in \mathcal{L}(\mathcal{A})$ for $q \xrightarrow{w} q'$ with $q \in I$ and $q' \in F$

# Nominal Automata[2]

A nominal automaton $\mathcal{A}$ over names $\mathbb{A}$ consists of:

- $\Sigma$, an orbit-finite nominal alphabet
- $Q$, an orbit-finite nominal set of states
- $I \subseteq Q$, an equivariant subset of initial states
- $F \subseteq Q$, an equivariant subset of final states
- $\delta \subseteq Q \times A \times Q$, an equivariant transition relation

We write:

- $q \xrightarrow{a} q'$ for $(q, a, q') \in \delta$
- $q \xrightarrow{w} q'$ for $w \in \Sigma^*$ and the reflexive-transitive closure of $\delta$
- $w \in \mathcal{L}(\mathcal{A})$ for $q \xrightarrow{w} q'$ with $q \in I$ and $q' \in F$

Example language: all words containing their initial letter twice

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).
Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \dots n_k \mid \nu a_k . n$$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).
Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \ldots n_k \mid \nu a_k . n$$

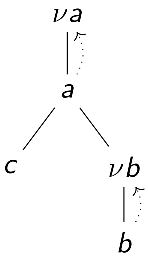
Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]). Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \ldots n_k \mid \nu a_k . n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.

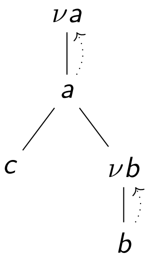


# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).
Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \ldots n_k \mid \nu a_k . n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

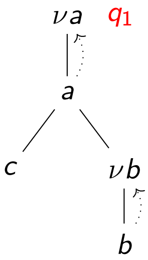
1. $(q, a_k) \Rightarrow (q_1, \ldots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \ldots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]). Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \dots n_k \mid \nu a_k . n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

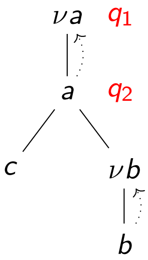
1. $(q, a_k) \Rightarrow (q_1, \dots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \dots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]). Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \ldots n_k \mid \nu a_k.n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

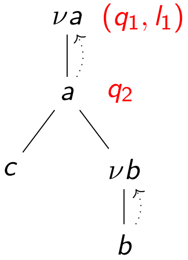
1. $(q, a_k) \Rightarrow (q_1, \ldots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \ldots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).
Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \ldots n_k \mid \nu a_k.n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

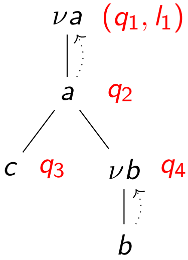
1. $(q, a_k) \Rightarrow (q_1, \ldots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \ldots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]). Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \dots n_k \mid \nu a_k.n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

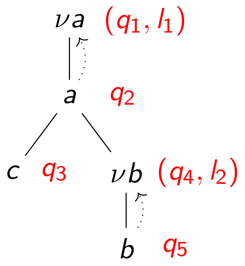
1. $(q, a_k) \Rightarrow (q_1, \dots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \dots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).
Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \dots n_k \mid \nu a_k . n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

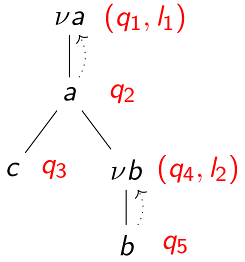
1. $(q, a_k) \Rightarrow (q_1, \dots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \dots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

# $\nu$-Tree Automata

Work building on Pitts/Stark ($\nu$-calculus[7]) and Stirling (NDTA[8]).
Consider a ranked finite alphabet $\Sigma \subset \mathbb{A}$ and $\nu$-trees constructed by

$$n ::= a_k n_1 \ldots n_k \mid \nu a_k . n$$

Think of $\nu$ as binding new names, so $\nu$-trees denote sets of $\mathbb{A}$-trees.




$\nu$-tree automaton (NTA) $\mathcal{A}$ consists of finite sets $Q$ and $L$ of states and labels together with transition rules of the form:

1. $(q, a_k) \Rightarrow (q_1, \ldots q_k)$
2. $((q, l), a_k) \Rightarrow (q_1, \ldots q_k)$
3. $(q, \nu a_k) \Rightarrow (q', l)$

$\Rightarrow$ closed under union, product + decidable acceptance, emptiness

# Outline

# General Procedure

## General Procedure

1. Fix an automaton of a certain model for words/trees

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types
3. Define $s \Downarrow n$ if $n$ is normal and a pure word/tree

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types
3. Define $s \Downarrow n$ if $n$ is normal and a pure word/tree
4. Derive typing rules for constants from automaton transitions

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types
3. Define $s \Downarrow n$ if $n$ is normal and a pure word/tree
4. Derive typing rules for constants from automaton transitions
5. Show that the type system is still well-behaved and captures acceptance: (Lemma) $\forall\, n, q.\ \vdash n : q \iff n \in \mathcal{L}(\mathcal{A}, q)$

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types
3. Define $s \Downarrow n$ if $n$ is normal and a pure word/tree
4. Derive typing rules for constants from automaton transitions
5. Show that the type system is still well-behaved and captures acceptance: (Lemma) $\forall n, q.\ \vdash n : q \iff n \in \mathcal{L}(\mathcal{A}, q)$
6. Prove a correspondence theorem of the shape:

**Theorem**

$$\forall \Gamma, s, q.\ \ \Gamma \vdash s : q \iff \exists n.\, s \Downarrow n \wedge n \in \mathcal{L}(\mathcal{A}, q)$$

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types
3. Define $s \Downarrow n$ if $n$ is normal and a pure word/tree
4. Derive typing rules for constants from automaton transitions
5. Show that the type system is still well-behaved and captures acceptance: (Lemma) $\forall\, n, q.\ \vdash n : q \iff n \in \mathcal{L}(\mathcal{A}, q)$
6. Prove a correspondence theorem of the shape:

**Theorem**

$$\forall\, \Gamma, s, q.\ \ \Gamma \vdash s : q \iff \exists n.\, s \Downarrow n \wedge n \in \mathcal{L}(\mathcal{A}, q)$$

"$\Longrightarrow$" by Normalisation, Subject Reduction and Lemma

# General Procedure

1. Fix an automaton of a certain model for words/trees
2. Interpret alphabet as constants and states as base types
3. Define $s \Downarrow n$ if $n$ is normal and a pure word/tree
4. Derive typing rules for constants from automaton transitions
5. Show that the type system is still well-behaved and captures acceptance: (Lemma) $\forall\, n, q.\ \vdash n : q \iff n \in \mathcal{L}(\mathcal{A}, q)$
6. Prove a correspondence theorem of the shape:

**Theorem**

$$\forall\, \Gamma, s, q.\ \ \Gamma \vdash s : q \iff \exists n.\, s \Downarrow n \wedge n \in \mathcal{L}(\mathcal{A}, q)$$

"$\Longrightarrow$" by Normalisation, Subject Reduction and Lemma
"$\Longleftarrow$" by Lemma and Subject Expansion

# ...for Finite Automata

# ...for Finite Automata

1. Let $\mathcal{A}$ be a finite automaton for words over finite $\Sigma$

# ...for Finite Automata

1. Let $\mathcal{A}$ be a finite automaton for words over finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$

# ...for Finite Automata

1. Let $\mathcal{A}$ be a finite automaton for words over finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\dots))$ is a pure word

# ...for Finite Automata

1. Let $\mathcal{A}$ be a finite automaton for words over finite $\Sigma$
2. Consider terms $s, t ::= x \in \text{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\dots))$ is a pure word
4. Add the following typing rules:

$$\frac{(q, a, q') \in \delta}{\Gamma \vdash a : q' \to q} \qquad \frac{(q, a, q') \in \delta \quad q' \in F}{\Gamma \vdash a : q}$$

# ...for Finite Automata

1. Let $\mathcal{A}$ be a finite automaton for words over finite $\Sigma$
2. Consider terms $s, t ::= x \in \text{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\ldots))$ is a pure word
4. Add the following typing rules:

$$\frac{(q, a, q') \in \delta}{\Gamma \vdash a : q' \to q} \qquad \frac{(q, a, q') \in \delta \quad q' \in F}{\Gamma \vdash a : q}$$

5. Correspondence Lemma for words by induction on length

# ...for Finite Automata

1. Let $\mathcal{A}$ be a finite automaton for words over finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\dots))$ is a pure word
4. Add the following typing rules:

$$\frac{(q, a, q') \in \delta}{\Gamma \vdash a : q' \to q} \qquad \frac{(q, a, q') \in \delta \quad q' \in F}{\Gamma \vdash a : q}$$

5. Correspondence Lemma for words by induction on length
6. Theorem by steps as outlined above

# ...for Nominal Automata

# ...for Nominal Automata

1. Let $\mathcal{A}$ be a nominal automaton for words over orbit-finite $\Sigma$

# ...for Nominal Automata

1. Let $\mathcal{A}$ be a nominal automaton for words over orbit-finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$

# ...for Nominal Automata

1. Let $\mathcal{A}$ be a nominal automaton for words over orbit-finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \rightarrow^* n$ and $n = a(b(\dots))$ is a pure word

# ...for Nominal Automata

1. Let $\mathcal{A}$ be a nominal automaton for words over orbit-finite $\Sigma$
2. Consider terms $s, t ::= x \in \text{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\dots))$ is a pure word
4. Add the following typing rules:

$$\frac{(q', a, q) \in \delta}{\Gamma \vdash a : q \to q'} \qquad \frac{(q, a, q') \in \delta \quad q' \in F}{\Gamma \vdash a : q}$$

# ...for Nominal Automata

1. Let $\mathcal{A}$ be a nominal automaton for words over orbit-finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\ldots))$ is a pure word
4. Add the following typing rules:

$$\frac{(q', a, q) \in \delta}{\Gamma \vdash a : q \to q'} \qquad \frac{(q, a, q') \in \delta \quad q' \in F}{\Gamma \vdash a : q}$$

5. Correspondence Lemma for words by induction on length

# ...for Nominal Automata

1. Let $\mathcal{A}$ be a nominal automaton for words over orbit-finite $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a \in \Sigma$
3. Define $s \Downarrow n$ if $s \to^* n$ and $n = a(b(\dots))$ is a pure word
4. Add the following typing rules:

$$\frac{(q', a, q) \in \delta}{\Gamma \vdash a : q \to q'} \qquad \frac{(q, a, q') \in \delta \quad q' \in F}{\Gamma \vdash a : q}$$

5. Correspondence Lemma for words by induction on length
6. Theorem by steps as outlined above

# ...for $\nu$-Tree Automata

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$
2. Consider terms $s, t ::= x \in \mathsf{Var} \mid \lambda x.s \mid st \mid a_k \in \Sigma \mid \nu a_k.s$

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a_k \in \Sigma \mid \nu a_k.s$
3. Add reduction rule $\nu a_k.\lambda x.s \to \lambda x.\nu a_k.s$ and abbreviate with $s \Downarrow n$ if $s \to^* n$ and $n$ is a well-ranked $\nu$-tree

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a_k \in \Sigma \mid \nu a_k.s$
3. Add reduction rule $\nu a_k.\lambda x.s \to \lambda x.\nu a_k.s$ and abbreviate with $s \Downarrow n$ if $s \to^* n$ and $n$ is a well-ranked $\nu$-tree
4. Add the following typing rules:

$$\frac{(q, a_k) \Rightarrow (q_1, \ldots, q_k) \quad a_k \notin \mathrm{dom}(\varphi)}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q} \qquad \frac{((q, l), a_k) \Rightarrow (q_1, \ldots, q_k) \quad \varphi(a_k) = l}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q}$$

$$\frac{(q, \nu a_k) \Rightarrow (q', l) \quad \Gamma, \varphi[a_k := l] \vdash s : q'}{\Gamma, \varphi \vdash \nu a_k.s : q} \qquad \frac{\Gamma, \varphi \vdash \lambda x.\nu a_k.s : \sigma \to \tau}{\Gamma, \varphi \vdash \nu a_k.\lambda x.s : \sigma \to \tau}$$

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a_k \in \Sigma \mid \nu a_k.s$
3. Add reduction rule $\nu a_k.\lambda x.s \to \lambda x.\nu a_k.s$ and abbreviate with $s \Downarrow n$ if $s \to^* n$ and $n$ is a well-ranked $\nu$-tree
4. Add the following typing rules:

$$\frac{(q, a_k) \Rightarrow (q_1, \ldots, q_k) \quad a_k \notin \mathrm{dom}(\varphi)}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q} \qquad \frac{((q, l), a_k) \Rightarrow (q_1, \ldots, q_k) \quad \varphi(a_k) = l}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q}$$

$$\frac{(q, \nu a_k) \Rightarrow (q', l) \quad \Gamma, \varphi[a_k := l] \vdash s : q'}{\Gamma, \varphi \vdash \nu a_k.s : q} \qquad \frac{\Gamma, \varphi \vdash \lambda x.\nu a_k.s : \sigma \to \tau}{\Gamma, \varphi \vdash \nu a_k.\lambda x.s : \sigma \to \tau}$$

5. Lemma for $\nu$-trees by inductive reformulation of acceptance

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a_k \in \Sigma \mid \nu a_k.s$
3. Add reduction rule $\nu a_k.\lambda x.s \to \lambda x.\nu a_k.s$ and abbreviate with $s \Downarrow n$ if $s \to^* n$ and $n$ is a well-ranked $\nu$-tree
4. Add the following typing rules:

$$\frac{(q, a_k) \Rightarrow (q_1, \ldots, q_k) \quad a_k \notin \mathrm{dom}(\varphi)}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q} \qquad \frac{((q, l), a_k) \Rightarrow (q_1, \ldots, q_k) \quad \varphi(a_k) = l}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q}$$

$$\frac{(q, \nu a_k) \Rightarrow (q', l) \quad \Gamma, \varphi[a_k := l] \vdash s : q'}{\Gamma, \varphi \vdash \nu a_k.s : q} \qquad \frac{\Gamma, \varphi \vdash \lambda x.\nu a_k.s : \sigma \to \tau}{\Gamma, \varphi \vdash \nu a_k.\lambda x.s : \sigma \to \tau}$$

5. Lemma for $\nu$-trees by inductive reformulation of acceptance
6. Theorem by ingredients as above

# ...for $\nu$-Tree Automata

1. Let $\mathcal{A}$ be an NTA for $\nu$-trees over finite ranked $\Sigma$
2. Consider terms $s, t ::= x \in \mathrm{Var} \mid \lambda x.s \mid st \mid a_k \in \Sigma \mid \nu a_k.s$
3. Add reduction rule $\nu a_k.\lambda x.s \to \lambda x.\nu a_k.s$ and abbreviate with $s \Downarrow n$ if $s \to^* n$ and $n$ is a well-ranked $\nu$-tree
4. Add the following typing rules:

$$\frac{(q, a_k) \Rightarrow (q_1, \ldots, q_k) \quad a_k \notin \mathrm{dom}(\varphi)}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q} \qquad \frac{((q, l), a_k) \Rightarrow (q_1, \ldots, q_k) \quad \varphi(a_k) = l}{\Gamma, \varphi \vdash a_k : q_1 \to \cdots \to q_k \to q}$$

$$\frac{(q, \nu a_k) \Rightarrow (q', l) \quad \Gamma, \varphi[a_k := l] \vdash s : q'}{\Gamma, \varphi \vdash \nu a_k.s : q} \qquad \frac{\Gamma, \varphi \vdash \lambda x.\nu a_k.s : \sigma \to \tau}{\Gamma, \varphi \vdash \nu a_k.\lambda x.s : \sigma \to \tau}$$

5. Lemma for $\nu$-trees by inductive reformulation of acceptance
6. Theorem by ingredients as above

Type checking, typability, inhabitance all decidable for base types and normal forms!

# Outline

# Possible Next Directions

# Possible Next Directions

- **Develope unranked $\nu$-trees and their automata:**
  Generalisation of Stirling's dependency tree automata

# Possible Next Directions

- **Develope unranked $\nu$-trees and their automata:**
  Generalisation of Stirling's dependency tree automata
- **Consider simply typed $\lambda Y$-terms as base language:**
  Restriction potentially allowing for general decidability

# Possible Next Directions

- **Develope unranked $\nu$-trees and their automata:**
  Generalisation of Stirling's dependency tree automata
- **Consider simply typed $\lambda Y$-terms as base language:**
  Restriction potentially allowing for general decidability
- **Relate the work to nominal type theory (Cheney 2009)[9]:**
  Based on nominal set of type variables similar to NNA

# References I

[1] Naoki Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *Proceedings of the 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2009, Savannah, GA, USA, January 21-23, 2009*, pages 416–428, 2009.

[2] Mikolaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014.

[3] H.P. Barendregt. *The lambda calculus: its syntax and semantics*. Studies in logic and the foundations of mathematics. North-Holland, 1984.

# References II

[4] H. Barendregt, W. Dekkers, and R. Statman. *Lambda Calculus with Types*. Lambda Calculus with Types. Cambridge University Press, 2013.

[5] J. Roger Hindley. Types with intersection: An introduction. *Formal Aspects of Computing*, 4(5):470–486, 1992.

[6] A. M. Pitts. *Nominal Sets: Names and Symmetry in Computer Science*, volume 57 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2013.

[7] Ian Stark. Names, equations, relations: Practical ways to reason about *new*. *Fundamenta Informaticae*, 33(4):369–396, April 1998.

# References III

[8] Colin Stirling. *Foundations of Software Science and Computational Structures: 12th International Conference, FOSSACS 2009, York, UK, March 22-29. Proceedings*, chapter Dependency Tree Automata, pages 92–106. Springer, Berlin, Heidelberg, 2009.

[9] James Cheney. A simple nominal type theory. *Electr. Notes Theor. Comput. Sci.*, 228:37–52, 2009.

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

## Definition

We define functions $[\![-]\!]_A : \nu\text{-Tree} \to \mathcal{P}(\mathbb{A}\text{-Tree})$ for $A \in \mathcal{P}_{\mathrm{fin}}(\mathbb{A})$:

$$\frac{m_i \in [\![n_i]\!]_{A \cup \{a_k\}}}{a_k m_1 \ldots m_k \in [\![a_k n_1 \ldots n_k]\!]_A} \qquad \frac{m \in [\![(a_k\ b_k) \cdot n]\!]_{A \cup \{b_k\}} \quad b_k \notin A \cup \mathrm{FN}(\nu a_k.n)}{m \in [\![\nu a_k.n]\!]_A}$$

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

**Definition**

We define functions $[\![-]\!]_A : \nu\text{-Tree} \to \mathcal{P}(\mathbb{A}\text{-Tree})$ for $A \in \mathcal{P}_{\mathrm{fin}}(\mathbb{A})$:

$$\frac{m_i \in [\![n_i]\!]_{A \cup \{a_k\}}}{a_k m_1 \ldots m_k \in [\![a_k n_1 \ldots n_k]\!]_A} \qquad \frac{m \in [\![(a_k\ b_k) \cdot n]\!]_{A \cup \{b_k\}} \quad b_k \notin A \cup \mathrm{FN}(\nu a_k.n)}{m \in [\![\nu a_k.n]\!]_A}$$

Properties we could establish:

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

**Definition**

We define functions $[\![-]\!]_A : \nu\text{-Tree} \to \mathcal{P}(\mathbb{A}\text{-Tree})$ for $A \in \mathcal{P}_{\text{fin}}(\mathbb{A})$:

$$\frac{m_i \in [\![n_i]\!]_{A \cup \{a_k\}}}{a_k m_1 \ldots m_k \in [\![a_k n_1 \ldots n_k]\!]_A} \qquad \frac{m \in [\![(a_k\ b_k) \cdot n]\!]_{A \cup \{b_k\}} \quad b_k \notin A \cup \mathrm{FN}(\nu a_k.n)}{m \in [\![\nu a_k.n]\!]_A}$$

Properties we could establish:

- The function $[\![-]\!]_{\_}$ is equivariant (hence morphism in **Nom**)

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

## Definition

We define functions $[\![-]\!]_A : \nu\text{-Tree} \to \mathcal{P}(\mathbb{A}\text{-Tree})$ for $A \in \mathcal{P}_{\mathrm{fin}}(\mathbb{A})$:

$$\frac{m_i \in [\![n_i]\!]_{A \cup \{a_k\}}}{a_k m_1 \ldots m_k \in [\![a_k n_1 \ldots n_k]\!]_A} \qquad \frac{m \in [\![(a_k\ b_k) \cdot n]\!]_{A \cup \{b_k\}} \quad b_k \notin A \cup \mathrm{FN}(\nu a_k.n)}{m \in [\![\nu a_k.n]\!]_A}$$

Properties we could establish:

- The function $[\![-]\!]_{\_}$ is equivariant (hence morphism in **Nom**)
- If $\pi$ fixes the free names of $n$ we have $[\![\pi \cdot n]\!]_A = [\![n]\!]_A$

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

## Definition

We define functions $[\![-]\!]_A : \nu\text{-Tree} \to \mathcal{P}(\mathbb{A}\text{-Tree})$ for $A \in \mathcal{P}_{\mathrm{fin}}(\mathbb{A})$:

$$\frac{m_i \in [\![n_i]\!]_{A \cup \{a_k\}}}{a_k m_1 \ldots m_k \in [\![a_k n_1 \ldots n_k]\!]_A} \qquad \frac{m \in [\![(a_k\ b_k) \cdot n]\!]_{A \cup \{b_k\}} \quad b_k \notin A \cup \mathrm{FN}(\nu a_k.n)}{m \in [\![\nu a_k.n]\!]_A}$$

Properties we could establish:

- The function $[\![-]\!]_{\_}$ is equivariant (hence morphism in **Nom**)
- If $\pi$ fixes the free names of $n$ we have $[\![\pi \cdot n]\!]_A = [\![n]\!]_A$
- $[\![n]\!]_A = [\![n']\!]_A$ iff both are $\alpha$-equivalent

# On the Denotation of $\nu$-Trees

We want a denotation with e.g. $[\![\nu a.ab]\!] = \{ab \mid a \in \mathbb{A} \setminus \{b\}\}$:

## Definition

We define functions $[\![-]\!]_A : \nu\text{-Tree} \to \mathcal{P}(\mathbb{A}\text{-Tree})$ for $A \in \mathcal{P}_{\text{fin}}(\mathbb{A})$:

$$\frac{m_i \in [\![n_i]\!]_{A \cup \{a_k\}}}{a_k m_1 \ldots m_k \in [\![a_k n_1 \ldots n_k]\!]_A} \qquad \frac{m \in [\![(a_k\ b_k) \cdot n]\!]_{A \cup \{b_k\}} \quad b_k \notin A \cup \mathrm{FN}(\nu a_k.n)}{m \in [\![\nu a_k.n]\!]_A}$$

Properties we could establish:

- The function $[\![-]\!]_{\_}$ is equivariant (hence morphism in **Nom**)
- If $\pi$ fixes the free names of $n$ we have $[\![\pi \cdot n]\!]_A = [\![n]\!]_A$
- $[\![n]\!]_A = [\![n']\!]_A$ iff both are $\alpha$-equivalent

Moreover, our treatment of $\nu$ is related *name abstraction*[6].