λ-calculus and recognizability

Sylvain Salvati and Igor Walukiewicz

Réunion de démarrage ANR FREC

Outline

Simply typed λ-calculus, free algebra and free monoid

Recognizability in the simply typed λ-calculus

A machine-like characterization of recognizability

Closure properties

Some applications

Perspectives within FREC

Outline

Simply typed λ-calculus

Given a finite set of atomic types $\mathcal{A}$, simple types are:

$$\mathcal{T}_{\mathcal{A}} := \mathcal{A} | (\mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{A}})$$

Simply typed λ-calculus

Given a finite set of atomic types $\mathcal{A}$, simple types are:

$$\mathcal{T}_{\mathcal{A}} := \mathcal{A} | (\mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{A}})$$

$\mathrm{ord}(\alpha) = 1$ and $\mathrm{ord}(\alpha \to \beta) = \max(\mathrm{ord}(\alpha) + 1, \mathrm{ord}(\beta))$

Simply typed λ-calculus

Given a finite set of atomic types $\mathcal{A}$, simple types are:

$$\mathcal{T}_{\mathcal{A}} := \mathcal{A} | (\mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{A}})$$

$\text{ord}(\alpha) = 1$ and $\text{ord}(\alpha \to \beta) = \max(\text{ord}(\alpha) + 1, \text{ord}(\beta))$
A higher order signature is a tuple $\Sigma = (\mathcal{A}, \mathcal{C}, \tau)$ where:

- $\mathcal{A}$ is a finite set of atomic types,
- $\mathcal{C}$ is a finite set of constants,
- τ is a function from $\mathcal{C}$ to $\mathcal{T}_{\mathcal{A}}$.

Simply typed λ-calculus

Given a finite set of atomic types $\mathcal{A}$, simple types are:

$$\mathcal{T}_{\mathcal{A}} := \mathcal{A} | (\mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{A}})$$

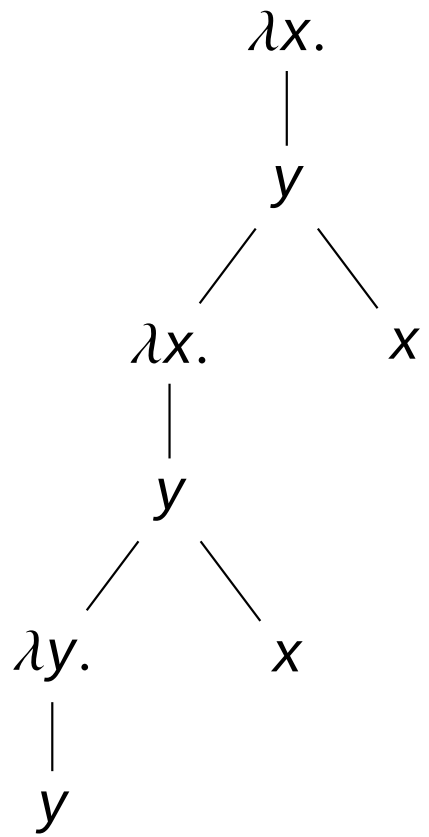
$\text{ord}(\alpha) = 1$ and $\text{ord}(\alpha \to \beta) = \max(\text{ord}(\alpha) + 1, \text{ord}(\beta))$
A higher order signature is a tuple $\Sigma = (\mathcal{A}, \mathcal{C}, \tau)$ where:

- $\mathcal{A}$ is a finite set of atomic types,
- $\mathcal{C}$ is a finite set of constants,
- τ is a function from $\mathcal{C}$ to $\mathcal{T}_{\mathcal{A}}$.

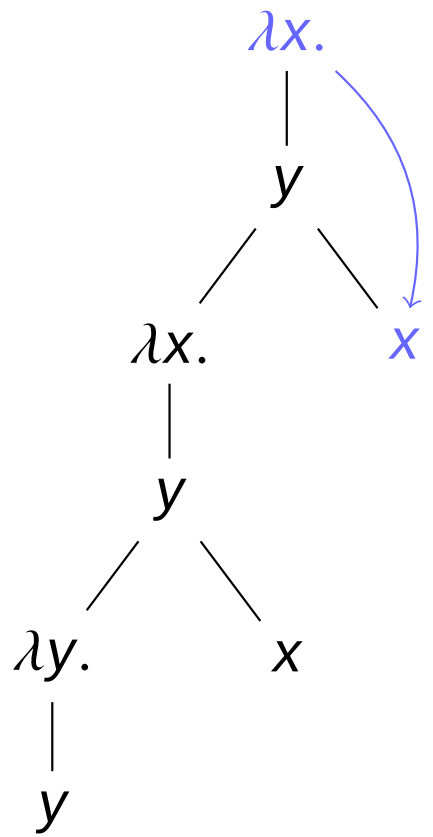
λ-terms built on Σ are defined as:

- for $\alpha \in \mathcal{T}_{\mathcal{A}}$, $x^{\alpha} \in \Lambda_{\Sigma}^{\alpha}$,
- $c \in \Lambda_{\Sigma}^{\tau(c)}$,
- if $M_1 \in \Lambda_{\Sigma}^{\alpha_2 \multimap \alpha_1}$, $M_2 \in \Lambda_{\Sigma}^{\alpha_2}$, then $(M_1 M_2) \in \Lambda_{\Sigma}^{\alpha_1}$,
- if $M \in \Lambda_{\Sigma}^{\alpha_1}$, then $\lambda x^{\alpha_2}.M \in \Lambda_{\Sigma}^{\alpha_2 \to \alpha_1}$.

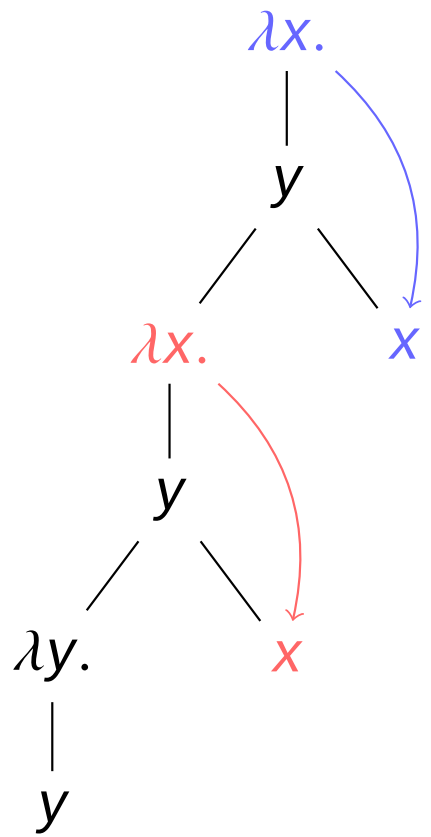
Free and bound variables



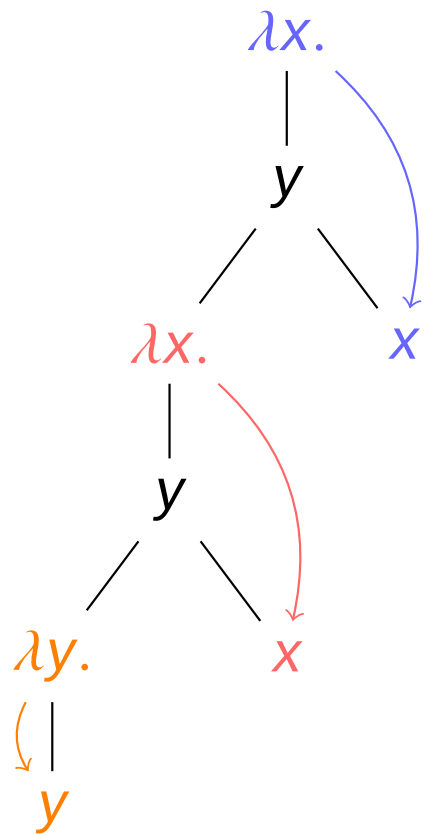
Free and bound variables



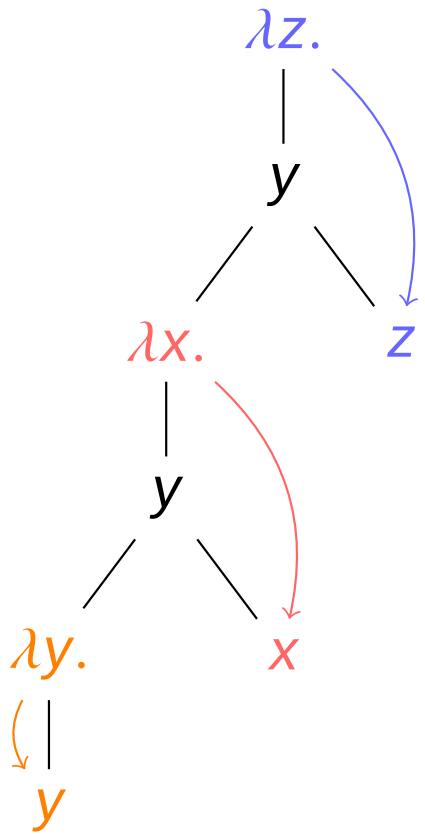
Free and bound variables



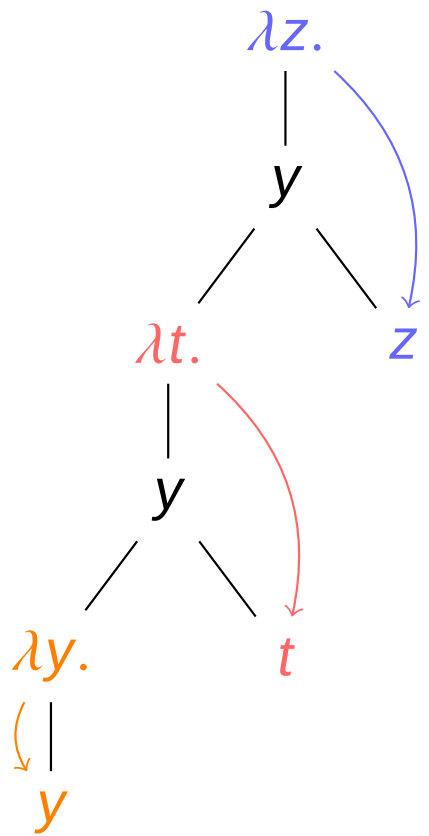
Free and bound variables



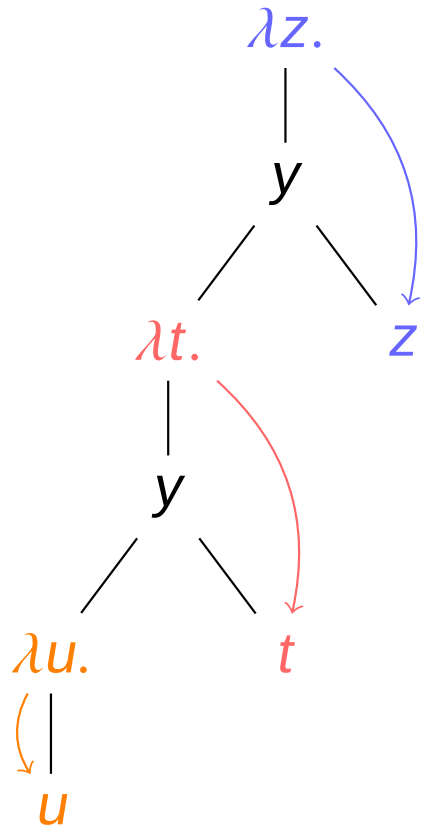
Free and bound variables



Free and bound variables



Free and bound variables



Using λ-terms to represent logical formulae

We use two types:

- e, the type for entities,

Using λ-terms to represent logical formulae

We use two types:

- e, the type for entities,
- t, the type for truth values.

Using λ-terms to represent logical formulae

We use two types:

- e, the type for entities,
- t, the type for truth values.

We represent logical connectives as constants:

- $\wedge, \vee : t \to t \to t$, $\neg : t \to t$

Using λ-terms to represent logical formulae

We use two types:

- e, the type for entities,
- t, the type for truth values.

We represent logical connectives as constants:

- $\wedge, \vee : t \to t \to t$, $\neg : t \to t$
- $\forall : (e \to t) \to t$, $\exists : (e \to t) \to t$

Using λ-terms to represent logical formulae

We use two types:

- e, the type for entities,
- t, the type for truth values.

We represent logical connectives as constants:

- $\wedge, \vee : t \to t \to t, \neg : t \to t$
- $\forall : (e \to t) \to t, \exists : (e \to t) \to t$
- $love : e \to e \to t$,$woman : e \to t$, $man : e \to t$, $mary : e$, $john : e \ldots$

$$\forall(\lambda x^e.\exists(\lambda y^e. \wedge (man\, x^e)(\wedge(woman\, y^e)(love\, x^e\, y^e)))$$

$$\approx$$

$$\forall x.\exists y.man(x) \wedge woman(y) \wedge love(x, y)$$

Operational semantics: β-reduction

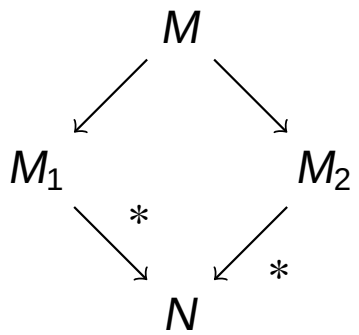
λ-calculus is a theory of function and computation.
Computation is done with the relation of β-contraction ($\rightarrow_\beta$):

$$\frac{(\lambda x.M)N}{(\lambda x.M)N \rightarrow_\beta M[x := N]} \qquad \frac{M_1 \rightarrow_\beta M_2}{(MM_1) \rightarrow_\beta (MM_2)}$$

$$\frac{M_1 \rightarrow_\beta M_2}{(M_1 M) \rightarrow_\beta (M_2 M)} \qquad \frac{M_1 \rightarrow_\beta M_2}{(\lambda x.M_1) \rightarrow_\beta (\lambda x.M_2)}$$

β-reduction ($\overset{*}{\rightarrow}_\beta$): reflexive transitive closure of β-contraction

Theorem (Church-Rosser)



Operational semantics:η-reduction

Mathematicians do not make a difference between the function sin and the function $f : x \to \sin(x)$.
η-contraction is used to reflect this fact in the λ-calculus.

$$\frac{(\lambda x.Mx) \quad x \notin FV(M)}{M} \qquad \frac{M_1 \to_\eta M_2}{(MM_1) \to_\eta (MM_2)}$$

$$\frac{M_1 \to_\eta M_2}{(M_1 M) \to_\eta (M_2 M)} \qquad \frac{M_1 \to_\beta M_2}{(\lambda x.M_1) \to_\eta (\lambda x.M_2)}$$

Operational semantics:η-reduction

Mathematicians do not make a difference between the function sin and the function $f : x \to \sin(x)$.
η-contraction is used to reflect this fact in the λ-calculus.

$$\frac{(\lambda x.Mx) \quad x \notin FV(M)}{M} \qquad \frac{M_1 \to_\eta M_2}{(MM_1) \to_\eta (MM_2)}$$

$$\frac{M_1 \to_\eta M_2}{(M_1 M) \to_\eta (M_2 M)} \qquad \frac{M_1 \to_\beta M_2}{(\lambda x.M_1) \to_\eta (\lambda x.M_2)}$$

η-reduction ($\overset{*}{\to}_\eta$): reflexive transitive closure of $\to_\eta$

Operational semantics:η-reduction

Mathematicians do not make a difference between the function sin and the function $f : x \to \sin(x)$.
η-contraction is used to reflect this fact in the λ-calculus.

$$\frac{(\lambda x.Mx) \quad x \notin FV(M)}{M} \qquad \frac{M_1 \to_\eta M_2}{(MM_1) \to_\eta (MM_2)}$$

$$\frac{M_1 \to_\eta M_2}{(M_1 M) \to_\eta (M_2 M)} \qquad \frac{M_1 \to_\beta M_2}{(\lambda x.M_1) \to_\eta (\lambda x.M_2)}$$

η-reduction ($\overset{*}{\to}_\eta$): reflexive transitive closure of $\to_\eta$
$\beta\eta$-reduction ($\overset{*}{\to}_{\beta\eta}$): $\overset{*}{\to}_\beta \cup \overset{*}{\to}_\eta$
$\beta\eta$-conversion ($=_{\beta\eta}$): commutative closure of $\to_{\beta\eta}$

Operational semantics:η-reduction

Mathematicians do not make a difference between the function sin and the function $f : x \to \sin(x)$.
η-contraction is used to reflect this fact in the λ-calculus.

$$\frac{(\lambda x.Mx) \quad x \notin FV(M)}{M} \qquad \frac{M_1 \to_\eta M_2}{(MM_1) \to_\eta (MM_2)}$$

$$\frac{M_1 \to_\eta M_2}{(M_1 M) \to_\eta (M_2 M)} \qquad \frac{M_1 \to_\beta M_2}{(\lambda x.M_1) \to_\eta (\lambda x.M_2)}$$

η-reduction ($\overset{*}{\to}_\eta$): reflexive transitive closure of $\to_\eta$
$\beta\eta$-reduction ($\overset{*}{\to}_{\beta\eta}$): $\overset{*}{\to}_\beta \cup \overset{*}{\to}_\eta$
$\beta\eta$-conversion ($=_{\beta\eta}$): commutative closure of $\to_{\beta\eta}$
Church-Rosser Theorem holds for all those relations

Strong normalisation

Theorem (strong normalisation)

If M is a simply typed term, then there is no infinite chains of term in relations of $\beta\eta$-contraction starting in M.

Strong normalisation

Theorem (strong normalisation)

If M is a simply typed term, then there is no infinite chains of term in relations of $\beta\eta$-contraction starting in M.

- No matter how a term is reduced, it will always output a result, and the same result.

Simply typed λ-calculus generalizes trees

The ranked alphabet $\{e; g; f\}$ where $\mathrm{rank}(e) = 0$, $\mathrm{rank}(g) = 1$, $\mathrm{rank}(f) = 2$ can be represented by the following second order constants:

$$e : o,\ g : o \to o,\ f : o \to o \to o$$

Simply typed λ-calculus generalizes trees

The ranked alphabet $\{e; g; f\}$ where $\mathrm{rank}(e) = 0$, $\mathrm{rank}(g) = 1$, $\mathrm{rank}(f) = 2$ can be represented by the following second order constants:

$$e : o,\ g : o \to o,\ f : o \to o \to o$$

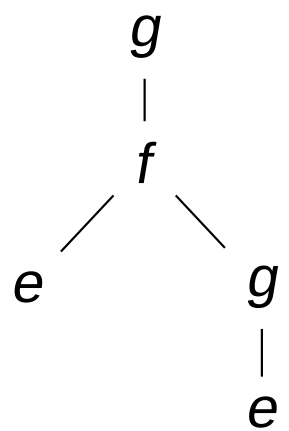
the term $g(f(e, g(e)))$ is represented by the λ-term $g(f\, e\, (g\, e))$

Simply typed λ-calculus generalizes trees

The ranked alphabet $\{e; g; f\}$ where $\mathrm{rank}(e) = 0$, $\mathrm{rank}(g) = 1$, $\mathrm{rank}(f) = 2$ can be represented by the following second order constants:

$$e : o,\ g : o \to o,\ f : o \to o \to o$$

the term $g(f(e, g(e)))$ is represented by the λ-term $g(f\, e\, (g\, e))$
The Böhm tree of the λ-term is the same as the graphic representation of the term:



Simply typed λ-calculus generalizes strings

The elements of $\{a; b\}^*$ can be represented with the constants:

$$a : o \to o,\ b : o \to o$$

Strings are represented by terms of type $o \to o$:

the string aba is represented by $/aba/ = \lambda x^o.a(b(a\, x^o))$

Simply typed λ-calculus generalizes strings

The elements of $\{a; b\}^*$ can be represented with the constants:

$$a : o \to o,\ b : o \to o$$

Strings are represented by terms of type $o \to o$:

the string aba is represented by $/aba/ = \lambda x^o.a(b(a\,x^o))$

Concatenation is then $s_1 + s_2 = \lambda x^o.s_1(s_2(x^o))$:

$$\begin{array}{rcl} /ab/ + /bb/ & = & \lambda x^o.a(b(x^o)) + \lambda x^o.b(b(x^o)) \\ & = & \lambda x^o.(\lambda y^o.a(b\,y^o))((\lambda z^o.b(b\,z^o))x^o) \\ & =_{\beta\eta} & \lambda x^o.a(b(b(b\,z^o))) \end{array}$$

and the empty string is $\lambda x^o.x^o$

Formal language theory and λ-calculus

- tree generated by a ranked alphabet: free algebra
- string generated by an alphabet: free monoid
- λ-terms of a given type generated by a higher order signature are also freely generated

Formal language theory and λ-calculus

- tree generated by a ranked alphabet: free algebra
- string generated by an alphabet: free monoid
- λ-terms of a given type generated by a higher order signature are also freely generated

Languages of λ-terms:

- Abstract Categorial Grammars
- Sets of semantic representations in Montague semantics
- Solutions of higher-order matching equations
- Finer grained languages than those obtained in substructural logic may be necessary in linguistics for: traces, extractions, ellipsis...

Recognizability

- For strings and trees, it corresponds to one of the simplest classes of languages,
- It naturally has several characterisation:
  - finite state automaton,
  - regular expressions,
  - finite congruences (Myhill-Nerode),
  - logic (W1S, WkS) (Büchi, Thatcher and Right, Döner),
- It is used in formal language theory to study classes of languages (AFL, Greibach),
- It is used, via MSO, for model checking.

Recognizability

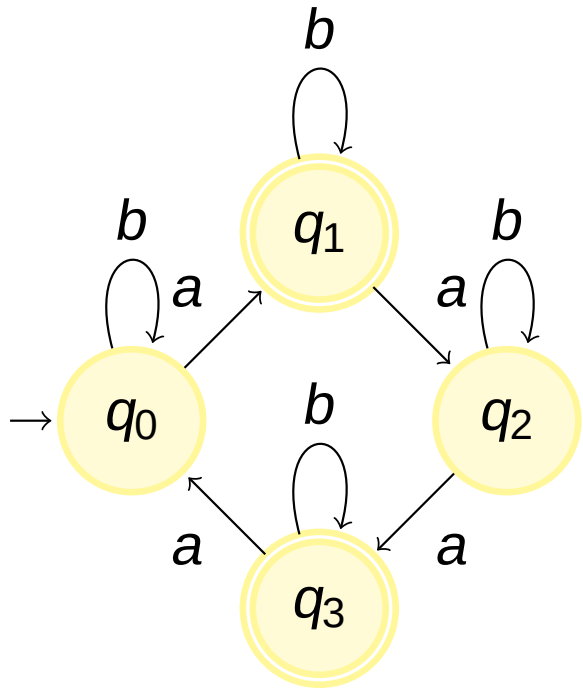
- For strings and trees, it corresponds to one of the simplest classes of languages,
- It naturally has several characterisation:
  - finite state automaton,
  - regular expressions,
  - finite congruences (Myhill-Nerode),
  - logic (W1S, WkS) (Büchi, Thatcher and Right, Döner),
- It is used in formal language theory to study classes of languages (AFL, Greibach),
- It is used, via MSO, for model checking.

We here present a generalization of recognizability for trees and strings to the simply typed λ-calculus.

Outline

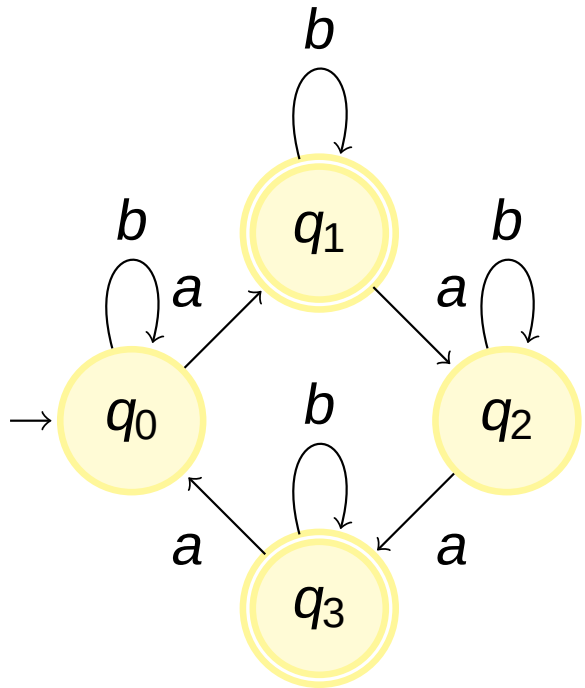
String recognizability and finite semigroup

Let's consider a deterministic total finite state automaton:



String recognizability and finite semigroup

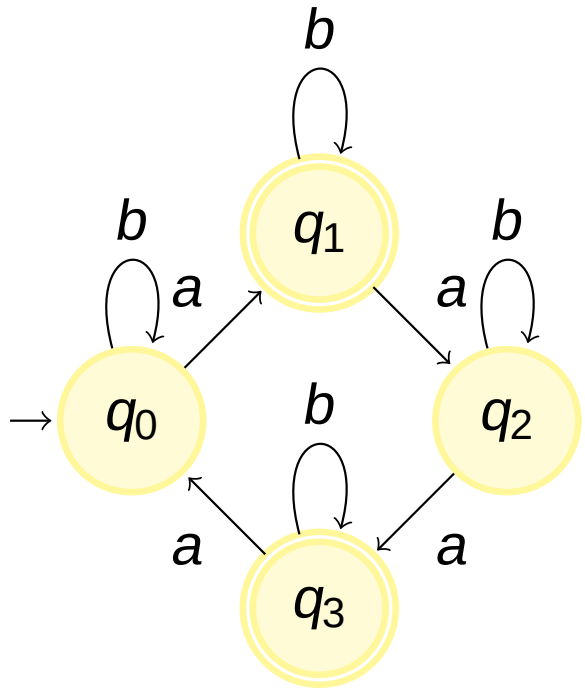
Let's consider a deterministic total finite state automaton:



- Each letter, a or b, can be interpreted as a function f_a, f_b, from states to states such that $f_\alpha(q) = q'$ iff (q, α, q') is a transition.

String recognizability and finite semigroup

Let's consider a deterministic total finite state automaton:



- Each letter, a or b, can be interpreted as a function f_a, f_b, from states to states such that $f_\alpha(q) = q'$ iff (q, α, q') is a transition.
- A word $\alpha_1 \ldots \alpha_n$ is in the language iff $h_{\alpha_1} \circ \ldots \circ h_{\alpha_n}(q_0) \in \{q_1; q_3\}$.

Tree recognizability and finite algebra

Let's consider the deterministic tree automata $\mathcal{A}$ with the states q_0, q_1, q_2 whose final states are $\{q_0; q_1\}$ and the rules:

$$f(q_i, q_j) \longrightarrow q_0$$
$$g(q_0) \longrightarrow q_1$$
$$g(q_1) \longrightarrow q_2$$
$$a \longrightarrow q_0$$

Tree recognizability and finite algebra

Let's consider the deterministic tree automata $\mathcal{A}$ with the states q_0, q_1, q_2 whose final states are $\{q_0; q_1\}$ and the rules:

$$f(q_i, q_j) \longrightarrow q_0 \qquad f(q, q') \longrightarrow \perp \text{ when } q = \perp \text{ or } q' = \perp$$
$$g(q_0) \longrightarrow q_1 \qquad g(q_2) \longrightarrow \perp$$
$$g(q_1) \longrightarrow q_2 \qquad g(\perp) \longrightarrow \perp$$
$$a \longrightarrow q_0$$

Tree recognizability and finite algebra

Let's consider the deterministic tree automata $\mathcal{A}$ with the states q_0, q_1, q_2 whose final states are $\{q_0; q_1\}$ and the rules:

$$
\begin{array}{ll}
f(q_i, q_j) \longrightarrow q_0 & f(q, q') \longrightarrow \bot \text{ when } q = \bot \text{ or } q' = \bot \\
g(q_0) \longrightarrow q_1 & g(q_2) \longrightarrow \bot \\
g(q_1) \longrightarrow q_2 & g(\bot) \longrightarrow \bot \\
a \longrightarrow q_0 &
\end{array}
$$

Then the automaton is computing the operations in the finite algebra $\mathbb{A}$, $\{q_0; q_1; q_2; \bot\}$ where $f(q_i, q_j) =_{\mathbb{A}} q_0$, $f(q, q') =_{\mathbb{A}} \bot$ when $q = \bot$ or $q' = \bot \ldots$ and

$$L(\mathcal{A}) = \{t \mid t =_{\mathbb{A}} q_0 \vee t =_{\mathbb{A}} q_1\}$$

Tree recognizability and finite algebra

Let's consider the deterministic tree automata $\mathcal{A}$ with the states q_0, q_1, q_2 whose final states are $\{q_0; q_1\}$ and the rules:

$$
\begin{array}{ll}
f(q_i, q_j) \longrightarrow q_0 & f(q, q') \longrightarrow \bot \text{ when } q = \bot \text{ or } q' = \bot \\
g(q_0) \longrightarrow q_1 & g(q_2) \longrightarrow \bot \\
g(q_1) \longrightarrow q_2 & g(\bot) \longrightarrow \bot \\
a \longrightarrow q_0 &
\end{array}
$$

Then the automaton is computing the operations in the finite algebra $\mathbb{A}$, $\{q_0; q_1; q_2; \bot\}$ where $f(q_i, q_j) =_{\mathbb{A}} q_0$, $f(q, q') =_{\mathbb{A}} \bot$ when $q = \bot$ or $q' = \bot \ldots$ and

$$L(\mathcal{A}) = \{t \mid t =_{\mathbb{A}} q_0 \vee t =_{\mathbb{A}} q_1\}$$

More generally this relation is formalized by Myhill-Nerode Theorem or in [Mezei, Wright 67].

Tree recognizability and finite algebra

Let's consider the deterministic tree automata $\mathcal{A}$ with the states q_0, q_1, q_2 whose final states are $\{q_0; q_1\}$ and the rules:

$$
\begin{array}{ll}
f(q_i, q_j) \longrightarrow q_0 & f(q, q') \longrightarrow \bot \text{ when } q = \bot \text{ or } q' = \bot \\
g(q_0) \longrightarrow q_1 & g(q_2) \longrightarrow \bot \\
g(q_1) \longrightarrow q_2 & g(\bot) \longrightarrow \bot \\
a \longrightarrow q_0 &
\end{array}
$$

Then the automaton is computing the operations in the finite algebra $\mathbb{A}$, $\{q_0; q_1; q_2; \bot\}$ where $f(q_i, q_j) =_{\mathbb{A}} q_0$, $f(q, q') =_{\mathbb{A}} \bot$ when $q = \bot$ or $q' = \bot \ldots$ and

$$L(\mathcal{A}) = \{t \mid t =_{\mathbb{A}} q_0 \vee t =_{\mathbb{A}} q_1\}$$

More generally this relation is formalized by Myhill-Nerode Theorem or in [Mezei, Wright 67].
Finite algebras and finite semi-groups are particular cases of finite models of the simply typed λ-calculus.

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.
- ι maps constants of type α to $\mathcal{M}^\alpha$

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.
- ι maps constants of type α to $\mathcal{M}^\alpha$

A variable assignment $\chi : V \to \bigcup_{\alpha \in \mathcal{T}(\Sigma)} \mathcal{M}^\alpha$ so that $\chi(x^\alpha) \in \mathcal{M}^\alpha$.

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.
- ι maps constants of type α to $\mathcal{M}^\alpha$

A variable assignment $\chi : V \to \bigcup_{\alpha \in \mathcal{T}(\Sigma)} \mathcal{M}^\alpha$ so that $\chi(x^\alpha) \in \mathcal{M}^\alpha$.
The semantics of λ-terms in $\mathbb{M}$ is inductively defined by:

- $[\![c]\!]^{\mathbb{M}}_\chi = \iota(c)$,

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.
- ι maps constants of type α to $\mathcal{M}^\alpha$

A variable assignment $\chi : V \to \bigcup_{\alpha \in \mathcal{T}(\Sigma)} \mathcal{M}^\alpha$ so that $\chi(x^\alpha) \in \mathcal{M}^\alpha$.
The semantics of λ-terms in $\mathbb{M}$ is inductively defined by:

- $[\![c]\!]^{\mathbb{M}}_\chi = \iota(c)$,
- $[\![x^\alpha]\!]^{\mathbb{M}}_\chi = \chi(x^\alpha)$,

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.
- ι maps constants of type α to $\mathcal{M}^\alpha$

A variable assignment $\chi : V \to \bigcup_{\alpha \in \mathcal{T}(\Sigma)} \mathcal{M}^\alpha$ so that $\chi(x^\alpha) \in \mathcal{M}^\alpha$.
The semantics of λ-terms in $\mathbb{M}$ is inductively defined by:

- $[\![c]\!]^{\mathbb{M}}_\chi = \iota(c)$,
- $[\![x^\alpha]\!]^{\mathbb{M}}_\chi = \chi(x^\alpha)$,
- $[\![MN]\!]^{\mathbb{M}}_\chi = [\![M]\!]^{\mathbb{M}}_\chi([\![N]\!]^{\mathbb{M}}_\chi)$,

Finite models for recognizability in the simply typed λ-calculus

Let Σ be a signature. $\mathbb{M} = ((\mathcal{M}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ is a finite model of Σ if:

- The sets $\mathcal{M}^\alpha$ are finite and pairwise disjoint.
- $\mathcal{M}^{\alpha \to \beta}$ is the set of all functions from $\mathcal{M}^\alpha$ to $\mathcal{M}^\beta$.
- ι maps constants of type α to $\mathcal{M}^\alpha$

A variable assignment $\chi : V \to \bigcup_{\alpha \in \mathcal{T}(\Sigma)} \mathcal{M}^\alpha$ so that $\chi(x^\alpha) \in \mathcal{M}^\alpha$.
The semantics of λ-terms in $\mathbb{M}$ is inductively defined by:

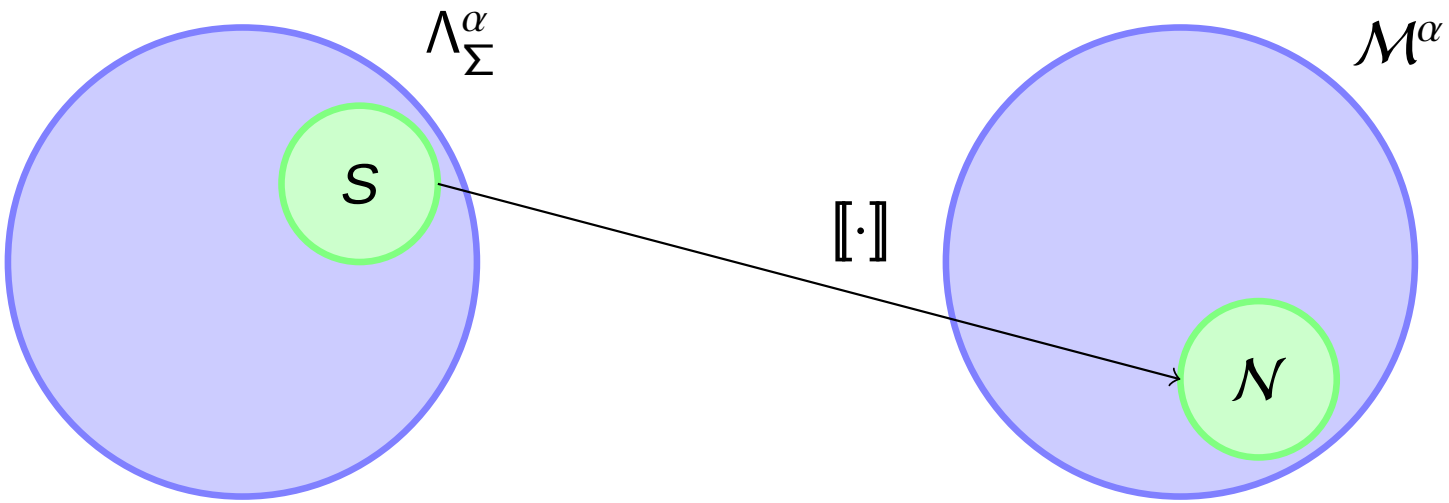
- $[\![c]\!]^{\mathbb{M}}_\chi = \iota(c)$,
- $[\![x^\alpha]\!]^{\mathbb{M}}_\chi = \chi(x^\alpha)$,
- $[\![MN]\!]^{\mathbb{M}}_\chi = [\![M]\!]^{\mathbb{M}}_\chi([\![N]\!]^{\mathbb{M}}_\chi)$,
- $[\![\lambda x^\alpha.M]\!]^{\mathbb{M}}_\chi = a \in \mathcal{M}^\alpha \to [\![M]\!]^{\mathbb{M}}_{\chi \leftarrow [x^\alpha := a]}$

Finite models for recognizability in the simply typed λ-calculus

Definition:

A set of λ-terms $S \subseteq \Lambda^{\alpha}_{\Sigma}$ is recognizable iff there is a finite set of variables W, a finite full model $\mathbb{M} = ((\mathcal{M}^{\alpha})_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$, $\mathcal{N} \subseteq \mathcal{M}^{\alpha}$ and a variable assignment χ such that:

$$S = \{M | FV(M) \subseteq W \wedge [\![M]\!]^{\mathbb{M}}_{\chi} \in \mathcal{N}\}$$

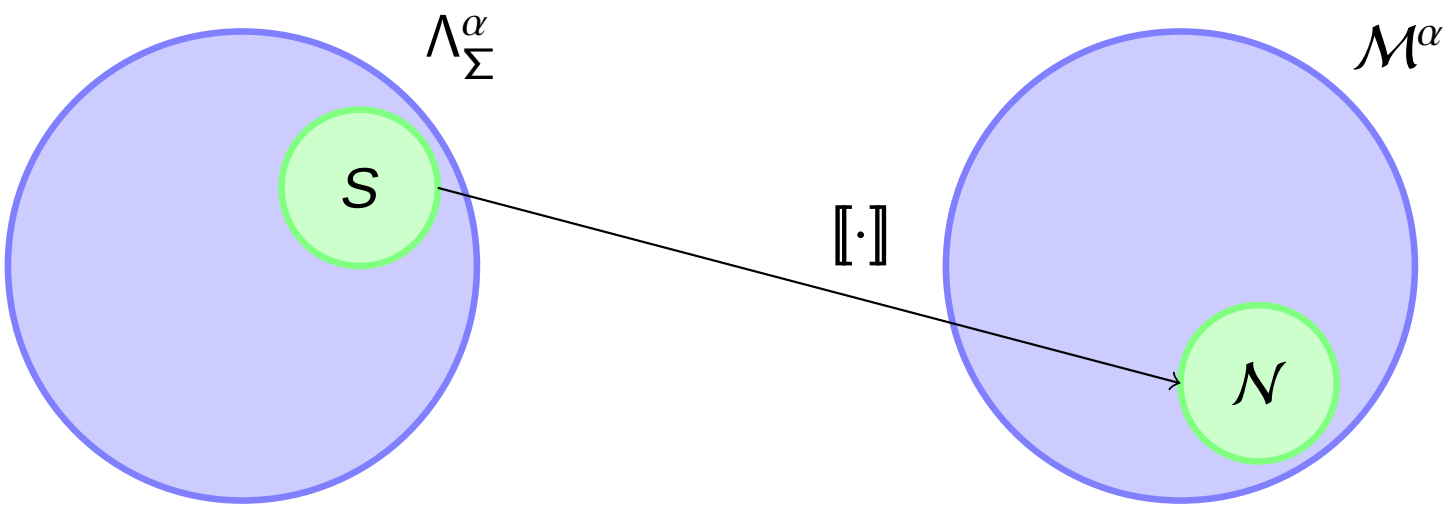


Finite models for recognizability in the simply typed λ-calculus

Definition:

A set of λ-terms $S \subseteq \Lambda^{\alpha}_{\Sigma}$ is recognizable iff there is a finite set of variables W, a finite full model $\mathbb{M} = ((\mathcal{M}^{\alpha})_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$, $\mathcal{N} \subseteq \mathcal{M}^{\alpha}$ and a variable assignment χ such that:

$$S = \{M | FV(M) \subseteq W \wedge [\![M]\!]^{\mathbb{M}}_{\chi} \in \mathcal{N}\}$$



Note:

- recognizable sets are closed under $=_{\beta\eta}$
- the emptiness of recognizable sets subsumes λ-definability which is undecidable (Loader 1993).

An example: Quantified Boolean Formulae (QBF)

We use the signature with the constants: $\wedge : p \to p \to p$, $\vee : p \to p \to p$, $\neg : p \to p$, $\forall^2 : (p \to p) \to p$, $\exists^2 : (p \to p) \to p$
Then the QBF formula $\forall^2 x.x \vee \neg x$ is represented as:

$$\forall^2(\lambda x^p. \vee\ x^p\ (\neg x^p)$$

An example: Quantified Boolean Formulae (QBF)

We use the signature with the constants: $\wedge : p \to p \to p$, $\vee : p \to p \to p$, $\neg : p \to p$, $\forall^2 : (p \to p) \to p$, $\exists^2 : (p \to p) \to p$
Then the QBF formula $\forall^2 x.x \vee \neg x$ is represented as:

$$\forall^2(\lambda x^p. \vee\ x^p\,(\neg x^p)$$

We take the model $\mathbb{B} = ((\mathcal{B}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ such that:

- $\mathcal{B}^p = \{0, 1\}$,
- $\wedge$, $\vee$, $\neg$ are interpreted as usual by ι
- $\iota(\forall^2)(f)$ is equal to 1 iff $f(1) = f(0) = 1$,
- $\iota(\exists^2)(f)$ is equal to 1 iff $f(0) = 1$ or $f(1) = 1$

An example: Quantified Boolean Formulae (QBF)

We use the signature with the constants: $\wedge : p \to p \to p$, $\vee : p \to p \to p$, $\neg : p \to p$, $\forall^2 : (p \to p) \to p$, $\exists^2 : (p \to p) \to p$
Then the QBF formula $\forall^2 x.x \vee \neg x$ is represented as:

$$\forall^2(\lambda x^p. \vee\ x^p\,(\neg x^p))$$

We take the model $\mathbb{B} = ((\mathcal{B}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ such that:

- $\mathcal{B}^p = \{0, 1\}$,
- $\wedge$, $\vee$, $\neg$ are interpreted as usual by ι
- $\iota(\forall^2)(f)$ is equal to 1 iff $f(1) = f(0) = 1$,
- $\iota(\exists^2)(f)$ is equal to 1 iff $f(0) = 1$ or $f(1) = 1$

If M represents the QBF formula F, then $[\![M]\!]^{\mathbb{B}}_{\emptyset} = 1$ iff F is a tautology.

An example: Quantified Boolean Formulae (QBF)

We use the signature with the constants: $\wedge : p \to p \to p$, $\vee : p \to p \to p$, $\neg : p \to p$, $\forall^2 : (p \to p) \to p$, $\exists^2 : (p \to p) \to p$
Then the QBF formula $\forall^2 x.x \vee \neg x$ is represented as:

$$\forall^2(\lambda x^p. \vee\ x^p\,(\neg x^p)$$

We take the model $\mathbb{B} = ((\mathcal{B}^\alpha)_{\alpha \in \mathcal{T}(\Sigma)}, \iota)$ such that:

- $\mathcal{B}^p = \{0, 1\}$,
- $\wedge$, $\vee$, $\neg$ are interpreted as usual by ι
- $\iota(\forall^2)(f)$ is equal to 1 iff $f(1) = f(0) = 1$,
- $\iota(\exists^2)(f)$ is equal to 1 iff $f(0) = 1$ or $f(1) = 1$

If M represents the QBF formula F, then $[\![M]\!]^{\mathbb{B}}_{\emptyset} = 1$ iff F is a tautology.
Thus the set of closed terms that represent QBF tautologies is recognizable.

Outline

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

- Given a BUTA $\mathcal{A}$ whose final state is q, we want to find a type system such that $\vdash t : q$ iff t is recognized by $\mathcal{A}$,

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

- Given a BUTA $\mathcal{A}$ whose final state is q, we want to find a type system such that $\vdash t : q$ iff t is recognized by $\mathcal{A}$,
- we need to associate types to constants in a certain way.

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

- Given a BUTA $\mathcal{A}$ whose final state is q, we want to find a type system such that $\vdash t : q$ iff t is recognized by $\mathcal{A}$,
- we need to associate types to constants in a certain way.

An automaton have rules of the form:

$$\begin{array}{lcl} a(q_1^1, \ldots, q_n^1) & \longrightarrow & q^1 \\ & \vdots & \\ a(q_1^p, \ldots, q_n^p) & \longrightarrow & q^p \end{array}$$

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

- Given a BUTA $\mathcal{A}$ whose final state is q, we want to find a type system such that $\vdash t : q$ iff t is recognized by $\mathcal{A}$,
- we need to associate types to constants in a certain way.

An automaton have rules of the form:

$$\begin{array}{llll} a(q_1^1,\ldots,q_n^1) & \longrightarrow & q^1 & a \;:\; q_1^1 \to \cdots \to q_n^1 \to q^1 \\ & \vdots & & \\ a(q_1^p,\ldots,q_n^p) & \longrightarrow & q^p & \end{array}$$

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

- Given a BUTA $\mathcal{A}$ whose final state is q, we want to find a type system such that $\vdash t : q$ iff t is recognized by $\mathcal{A}$,
- we need to associate types to constants in a certain way.

An automaton have rules of the form:

$$\begin{array}{llllll} a(q_1^1,\ldots,q_n^1) & \longrightarrow & q^1 & a & : & q_1^1 \to \cdots \to q_n^1 \to q^1 \\ & \vdots & & & \vdots & \\ a(q_1^p,\ldots,q_n^p) & \longrightarrow & q^p & a & : & q_1^p \to \cdots \to q_n^p \to q^p \end{array}$$

Obtaining recognizable sets of trees with types

Recognizability is invariant under $=_{\beta\eta}$, and the best way to capture such invariants in the λ-calculus is to use types.
Regular sets of trees may be defined as the sets of trees that are recognized by a bottom-up tree automaton (BUTA).

- Given a BUTA $\mathcal{A}$ whose final state is q, we want to find a type system such that $\vdash t : q$ iff t is recognized by $\mathcal{A}$,
- we need to associate types to constants in a certain way.

An automaton have rules of the form:

$$\begin{array}{llllll} a(q_1^1,\ldots,q_n^1) & \longrightarrow & q^1 & a & : & q_1^1 \to \cdots \to q_n^1 \to q^1 \\ & \vdots & & & \vdots & \\ a(q_1^p,\ldots,q_n^p) & \longrightarrow & q^p & a & : & q_1^p \to \cdots \to q_n^p \to q^p \end{array}$$

It is as if a should have at the same time the type $q_1^1 \to \cdots \to q_n^1 \to q^1, \ldots,$ and the type $q_1^p \to \cdots \to q_n^p \to q^p$.

Intersection types

There is a type system, intersection types (Coppo, Dezani 1980), that assignes several types to λ-terms:

- it is used to type the untyped λ-caculus and grasp dynamic properties of terms via typing (strong/weak normalization, solvability...),

Intersection types

There is a type system, intersection types (Coppo, Dezani 1980), that assignes several types to λ-terms:

- it is used to type the untyped λ-caculus and grasp dynamic properties of terms via typing (strong/weak normalization, solvability...),
- it is used to build models of the λ-calculi,

Intersection types

There is a type system, intersection types (Coppo, Dezani 1980), that assignes several types to λ-terms:

- it is used to type the untyped λ-caculus and grasp dynamic properties of terms via typing (strong/weak normalization, solvability...),
- it is used to build models of the λ-calculi,
- typing judgements are closed under $=_{\beta\eta}$.

Intersection types a brief introduction

In the untyped λ-calculus, intersection type systems are of the form:

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash M : \Omega} \qquad \frac{\Gamma \vdash M : A \quad \Gamma \vdash M : B}{\Gamma \vdash M : A \cap B}$$

$$\frac{\Gamma \vdash M : A_1 \cap A_2 \quad i \in \{1; 2\}}{\Gamma \vdash M : A_i}$$

$$\frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

Let $\mathbf{S}(\Gamma, A) = \{M | \Gamma \vdash M : A\}$, one can easily see that:

- $\mathbf{S}(\Gamma, A \cap A) = \mathbf{S}(\Gamma, A)$ and
  $\mathbf{S}(\Gamma, \Omega \cap A) = \mathbf{S}(\Gamma, A \cap \Omega) = \mathbf{S}(\Gamma, A)$,
- $\mathbf{S}(\Gamma, A \cap B) = \mathbf{S}(\Gamma, B \cap A)$ and
  $\mathbf{S}(\Gamma, (A \cap (B \cap C))) = \mathbf{S}(\Gamma, ((A \cap B) \cap C))$

Higher order intersection signature

We type simply typed terms with intersection types.

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms
- ρ is a mapping from I to $\mathscr{A}$

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms
- ρ is a mapping from I to $\mathcal{A}$
- f maps constants of type α to a subset of $\cap^{\alpha}_{\Pi}$ where:

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms
- ρ is a mapping from I to $\mathcal{A}$
- f maps constants of type α to a subset of $\cap^{\alpha}_{\Pi}$ where:
  - If $\alpha \in \mathcal{A}$ then $\cap^{\alpha}_{\Pi} = \rho^{-1}(\alpha)$

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms
- ρ is a mapping from I to $\mathcal{A}$
- f maps constants of type α to a subset of $\cap^{\alpha}_{\Pi}$ where:
  - If $\alpha \in \mathcal{A}$ then $\cap^{\alpha}_{\Pi} = \rho^{-1}(\alpha)$
  - $\cap^{\alpha\to\beta}_{\Pi} = 2^{\cap^{\alpha}_{\Pi}} \times \{\alpha\} \times \cap^{\beta}_{\Pi}$.

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms
- ρ is a mapping from I to $\mathcal{A}$
- f maps constants of type α to a subset of $\cap_\Pi^\alpha$ where:
  - If $\alpha \in \mathcal{A}$ then $\cap_\Pi^\alpha = \rho^{-1}(\alpha)$
  - $\cap_\Pi^{\alpha\to\beta} = 2^{\cap_\Pi^\alpha} \times \{\alpha\} \times \cap_\Pi^\beta$.

Given $(S, \alpha, p) \in \cap_\Pi^{\alpha\to\beta}$, it may be written:

- $p_1 \cap \ldots \cap p_n \to p$ when $S \neq \emptyset$ and $S = \{p_1; \ldots; p_n\}$.
- $\Omega_\alpha \to p$ when $S = \emptyset$.

Higher order intersection signature

We type simply typed terms with intersection types.
Given a signature Σ, a higher order intersection signature over Σ is $\Pi = (\Sigma, I, \rho, f)$:

- I is a finite set of atoms
- ρ is a mapping from I to $\mathcal{A}$
- f maps constants of type α to a subset of $\cap_\Pi^\alpha$ where:
  - If $\alpha \in \mathcal{A}$ then $\cap_\Pi^\alpha = \rho^{-1}(\alpha)$
  - $\cap_\Pi^{\alpha\to\beta} = 2^{\cap_\Pi^\alpha} \times \{\alpha\} \times \cap_\Pi^\beta$.

Given $(S, \alpha, p) \in \cap_\Pi^{\alpha\to\beta}$, it may be written:

- $p_1 \cap \ldots \cap p_n \to p$ when $S \neq \emptyset$ and $S = \{p_1; \ldots; p_n\}$.
- $\Omega_\alpha \to p$ when $S = \emptyset$.

N.B.: $|\cap_\Pi^\alpha|$ is finite.

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.
- Typing contexts are partial functions Γ from variables to sets of intersection types such that $\Gamma(x^\alpha) \subseteq \cap^\alpha_\Pi$.

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.
- Typing contexts are partial functions Γ from variables to sets of intersection types such that $\Gamma(x^\alpha) \subseteq \cap^\alpha_\Pi$.

$$\frac{\Gamma, x^\alpha : S \text{ is a context} \quad p \in S}{\Gamma, x^\alpha : S \vdash_\Pi x^\alpha : p}$$

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.
- Typing contexts are partial functions Γ from variables to sets of intersection types such that $\Gamma(x^\alpha) \subseteq \cap^\alpha_\Pi$.

$$\frac{\Gamma, x^\alpha : S \text{ is a context} \quad p \in S}{\Gamma, x^\alpha : S \vdash_\Pi x^\alpha : p}$$

$$\frac{\Gamma, x^\alpha : S \vdash_\Pi M : p}{\Gamma \vdash_\Pi \lambda x^\alpha . M : (S, \alpha, p)}$$

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.
- Typing contexts are partial functions Γ from variables to sets of intersection types such that $\Gamma(x^\alpha) \subseteq \cap^\alpha_\Pi$.

$$\frac{\Gamma, x^\alpha : S \text{ is a context} \quad p \in S}{\Gamma, x^\alpha : S \vdash_\Pi x^\alpha : p}$$

$$\frac{\Gamma, x^\alpha : S \vdash_\Pi M : p}{\Gamma \vdash_\Pi \lambda x^\alpha . M : (S, \alpha, p)}$$

$$\frac{\Gamma \vdash_\Pi M : (S, \alpha, p) \quad \forall q \in S.\, \Gamma \vdash_\Pi N : q \quad N \in \Lambda^\alpha_\Sigma}{\Gamma \vdash_\Pi (MN) : p}$$

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.
- Typing contexts are partial functions Γ from variables to sets of intersection types such that $\Gamma(x^\alpha) \subseteq \cap^\alpha_\Pi$.

$$\frac{\Gamma, x^\alpha : S \text{ is a context} \quad p \in S}{\Gamma, x^\alpha : S \vdash_\Pi x^\alpha : p} \qquad \frac{\Gamma \vdash_\Pi M : p \quad p \sqsubseteq^\alpha_\Pi q}{\Gamma \vdash_\Pi M : q}$$

$$\frac{\Gamma, x^\alpha : S \vdash_\Pi M : p}{\Gamma \vdash_\Pi \lambda x^\alpha . M : (S, \alpha, p)}$$

$$\frac{\Gamma \vdash_\Pi M : (S, \alpha, p) \quad \forall q \in S. \, \Gamma \vdash_\Pi N : q \quad N \in \Lambda^\alpha_\Sigma}{\Gamma \vdash_\Pi (MN) : p}$$

The derivation system

- We suppose we are given a HOS Σ and $\Pi = (\Sigma, I, \rho, f)$ a HOIS over Σ.
- We derive typing judgements with intersection types on *simply typed λ-terms (à la Church)*.
- Typing contexts are partial functions Γ from variables to sets of intersection types such that $\Gamma(x^\alpha) \subseteq \cap^\alpha_\Pi$.

$$\frac{\Gamma, x^\alpha : S \text{ is a context} \quad p \in S}{\Gamma, x^\alpha : S \vdash_\Pi x^\alpha : p} \qquad \frac{\Gamma \vdash_\Pi M : p \quad p \sqsubseteq^\alpha_\Pi q}{\Gamma \vdash_\Pi M : q}$$

$$\frac{\Gamma, x^\alpha : S \vdash_\Pi M : p}{\Gamma \vdash_\Pi \lambda x^\alpha . M : (S, \alpha, p)}$$

$$\frac{\Gamma \vdash_\Pi M : (S, \alpha, p) \quad \forall q \in S.\, \Gamma \vdash_\Pi N : q \quad N \in \Lambda^\alpha_\Sigma}{\Gamma \vdash_\Pi (MN) : p}$$

In general we write $\Gamma \vdash_\Pi M : S$ instead of $\forall p \in S.\, \Gamma \vdash_\Pi M : S$.

Subsumption relation on types

$$\frac{\iota \in \rho(\alpha)}{\iota \sqsubseteq^{\alpha}_{\Pi} \iota} \qquad \frac{S \subseteq \cap^{\alpha}_{\Pi} \quad \forall p \in T. \exists q \in S.\ q \sqsubseteq^{\alpha}_{\Pi} p}{S \trianglelefteq^{\alpha}_{\Pi} T}$$

$$\frac{S \trianglelefteq^{\alpha}_{\Pi} T \quad q \sqsubseteq^{\beta}_{\Pi} p}{(T, \alpha, q) \sqsubseteq^{\alpha \to \beta}_{\Pi} (S, \alpha, p)}$$

Properties

- **Type correctness:** $\Gamma \vdash_{\Pi} M : p \Rightarrow \exists \alpha.\, M \in \Lambda^{\alpha}_{\Sigma} \wedge p \in \cap^{\alpha}_{\Pi}$

Properties

- **Type correctness:** $\Gamma \vdash_{\Pi} M : p \Rightarrow \exists \alpha.\, M \in \Lambda^{\alpha}_{\Sigma} \wedge p \in \cap^{\alpha}_{\Pi}$
- **Typability is decidable:** Given M, it is decidable whether $\Gamma \vdash_{\Pi} M : p$ holds.

Properties

- **Type correctness:** $\Gamma \vdash_{\Pi} M : p \Rightarrow \exists \alpha.\ M \in \Lambda^{\alpha}_{\Sigma} \wedge p \in \cap^{\alpha}_{\Pi}$
- **Typability is decidable:** Given M, it is decidable whether $\Gamma \vdash_{\Pi} M : p$ holds.
- **Subject conversion:**
  $M,\ N \in \Lambda^{\alpha}_{\Sigma} \wedge M =_{\beta\eta} N \Rightarrow \Gamma \vdash_{\Pi} N : p$ iff $\Gamma \vdash_{\Pi} M : p$

Properties

- **Type correctness:** $\Gamma \vdash_\Pi M : p \Rightarrow \exists \alpha.\ M \in \Lambda^\alpha_\Sigma \wedge p \in \cap^\alpha_\Pi$
- **Typability is decidable:** Given M, it is decidable whether $\Gamma \vdash_\Pi M : p$ holds.
- **Subject conversion:**
  $M,\ N \in \Lambda^\alpha_\Sigma \wedge M =_{\beta\eta} N \Rightarrow \Gamma \vdash_\Pi N : p$ iff $\Gamma \vdash_\Pi M : p$
- **Singleton set:** for every $M \in \Lambda^\alpha_\Sigma$ there are Π, Γ and p such that $\Gamma \vdash_\Pi N : p$ iff $N \in \Lambda^\alpha_\Sigma$ and $M =_{\beta\eta} N$.

On the singleton theorem

- The proof of the singleton theorem is inspired from the proof of a coherence Theorem [Mints 79, Babaev Soloviev 82],

On the singleton theorem

- The proof of the singleton theorem is inspired from the proof of a coherence Theorem [Mints 79, Babaev Soloviev 82],
- as for coherence theorem, the types used in the singleton theorems can be interpreted as addresses in the term: types generalize the notion of positions in strings used in parsing,

On the singleton theorem

- The proof of the singleton theorem is inspired from the proof of a coherence Theorem [Mints 79, Babaev Soloviev 82],
- as for coherence theorem, the types used in the singleton theorems can be interpreted as addresses in the term: types generalize the notion of positions in strings used in parsing,
- extensions of the coherence Theorems such as [Aoto, Ono 99] lead to a different proof of the singleton theorem using shorter types,

On the singleton theorem

- The proof of the singleton theorem is inspired from the proof of a coherence Theorem [Mints 79, Babaev Soloviev 82],
- as for coherence theorem, the types used in the singleton theorems can be interpreted as addresses in the term: types generalize the notion of positions in strings used in parsing,
- extensions of the coherence Theorems such as [Aoto, Ono 99] lead to a different proof of the singleton theorem using shorter types,
- this theorem is tightly related to the finite completeness Theorem by [Statman 82]

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$$\frac{}{\vdash \sqcap \quad : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$\vdash_\Pi \lambda f^{*\to *} x^*.$ $: (0 \to 1 \cap 1 \to 2) \to 0 \to 2$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$$\frac{f^{*\to *} : 0 \to 1 \cap 1 \to 2 \vdash_\Pi \lambda x^*. \quad : 0 \to 2}{\vdash_\Pi \lambda f^{*\to *} x^*. \quad : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$$\dfrac{\dfrac{\dfrac{}{f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0 \vdash_\Pi : 2}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2 \vdash_\Pi \lambda x^*. : 0 \to 2}}{\vdash_\Pi \lambda f^{*\to *} x^*. : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Pi f^{*\to *} : 1 \to 2}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0 \vdash_\Pi f^{*\to *} : 2}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2 \vdash_\Pi \lambda x^*.f^{*\to *} : 0 \to 2}}{\vdash_\Pi \lambda f^{*\to *} x^*.f^{*\to *} : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

$\Gamma = f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Pi f^{*\to *} : 1 \to 2} \quad \dfrac{}{\Gamma \vdash_\Pi \quad : 1}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0 \vdash_\Pi f^{*\to *} \quad : 2}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2 \vdash_\Pi \lambda x^*.f^{*\to *} \quad : 0 \to 2}}{\vdash_\Pi \lambda f^{*\to *} x^*.f^{*\to *} \quad : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

$\Gamma = f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Pi f^{*\to *} : 1 \to 2} \quad \dfrac{\dfrac{}{\Gamma \vdash_\Pi f^{*\to *} : 0 \to 1}}{\Gamma \vdash_\Pi f^{*\to *} : 1}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0 \vdash_\Pi f^{*\to *}(f^{*\to *}) : 2}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2 \vdash_\Pi \lambda x^*.f^{*\to *}(f^{*\to *}) : 0 \to 2}}{\vdash_\Pi \lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}) : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

$\Gamma = f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0$

Example of the singleton construction

The unique term that can be typed with $(0 \to 1 \cap 1 \to 2) \to 0 \to 2$ is $\lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}(x^*))$:

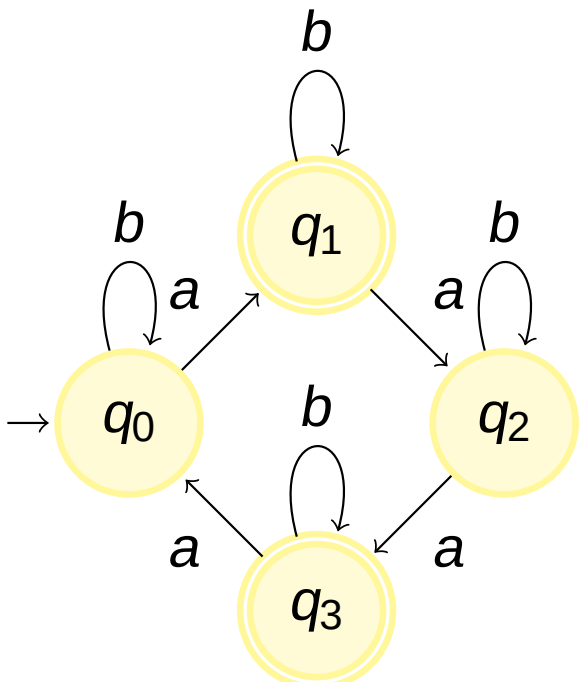
$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_\Pi f^{*\to *} : 1 \to 2} \quad \dfrac{\dfrac{}{\Gamma \vdash_\Pi f^{*\to *} : 0 \to 1} \quad \dfrac{}{\Gamma \vdash_\Pi x^* : 0}}{\Gamma \vdash_\Pi f^{*\to *}\, x^* : 1}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0 \vdash_\Pi f^{*\to *}(f^{*\to *}\, x^*) : 2}}{f^{*\to *} : 0 \to 1 \cap 1 \to 2 \vdash_\Pi \lambda x^*.f^{*\to *}(f^{*\to *}\, x^*) : 0 \to 2}}{\vdash_\Pi \lambda f^{*\to *} x^*.f^{*\to *}(f^{*\to *}\, x^*) : (0 \to 1 \cap 1 \to 2) \to 0 \to 2}$$

$\Gamma = f^{*\to *} : 0 \to 1 \cap 1 \to 2, x^* : 0$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$

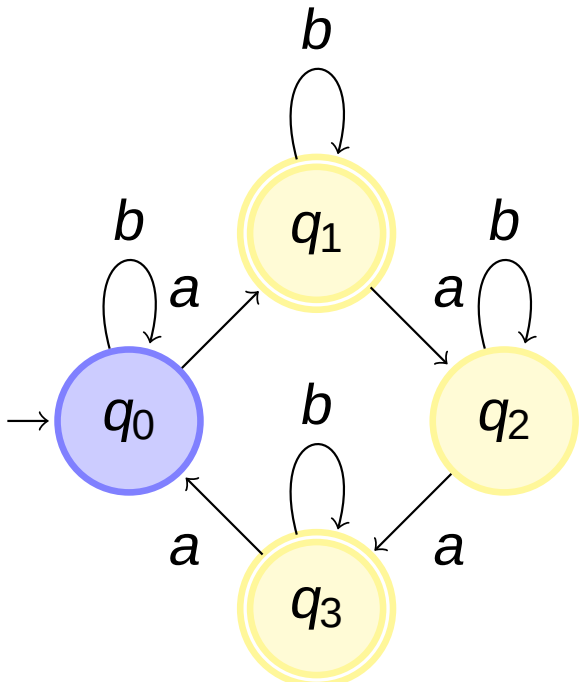


$$\vdash_{\sqcap} \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$

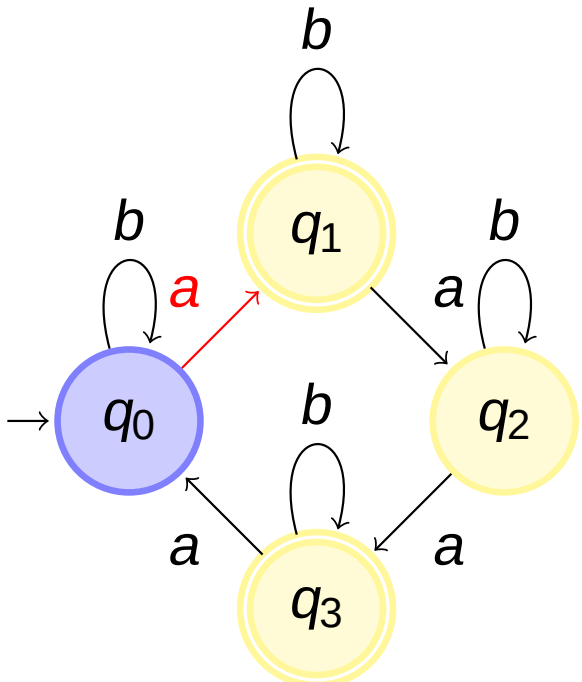


$$\frac{}{x^o : ? \vdash_\Pi a(b(a(a\,x^o))) : q_0}$$
$$\frac{}{\vdash_\Pi \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0}$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$

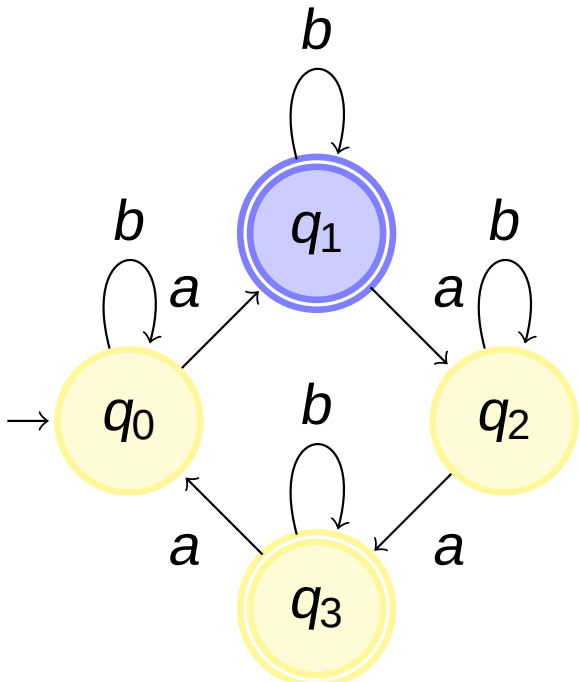


$$\dfrac{\dfrac{}{\vdash_\Pi a : q_1 \to q_0}}{x^o : ? \vdash_\Pi a(b(a(a\,x^o))) : q_0}$$
$$\vdash_\Pi \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$

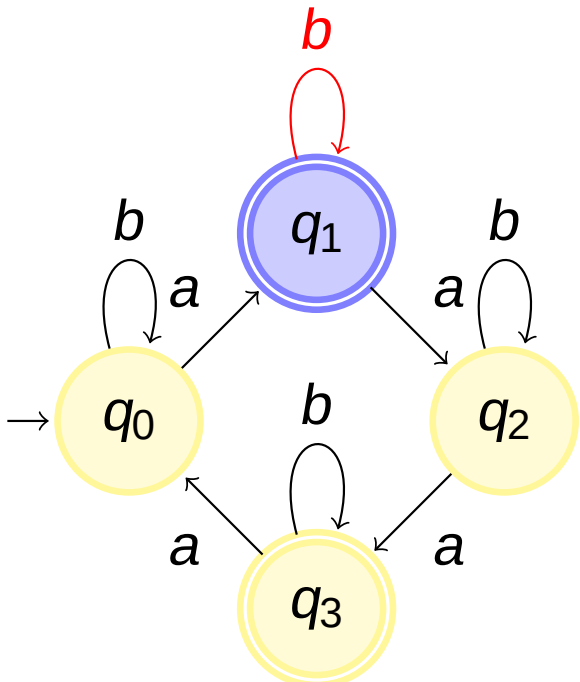


$$\dfrac{\dfrac{}{\vdash_\sqcap a : q_1 \to q_0} \quad \dfrac{}{x^o : ? \vdash_\sqcap b(a(a\,x^o)) : q_1}}{\dfrac{x^o : ? \vdash_\sqcap a(b(a(a\,x^o))) : q_0}{\vdash_\sqcap \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0}}$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$

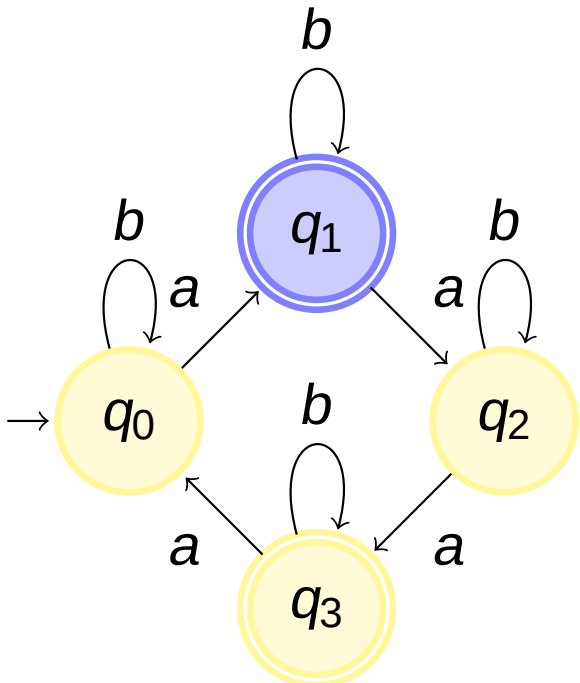


$$\dfrac{\dfrac{}{\vdash_\sqcap a : q_1 \to q_0} \quad \dfrac{}{x^o : ? \vdash_\sqcap b(a(a\,x^o)) : q_1}}{\dfrac{x^o : ? \vdash_\sqcap a(b(a(a\,x^o))) : q_0}{\vdash_\sqcap \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0}}$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$

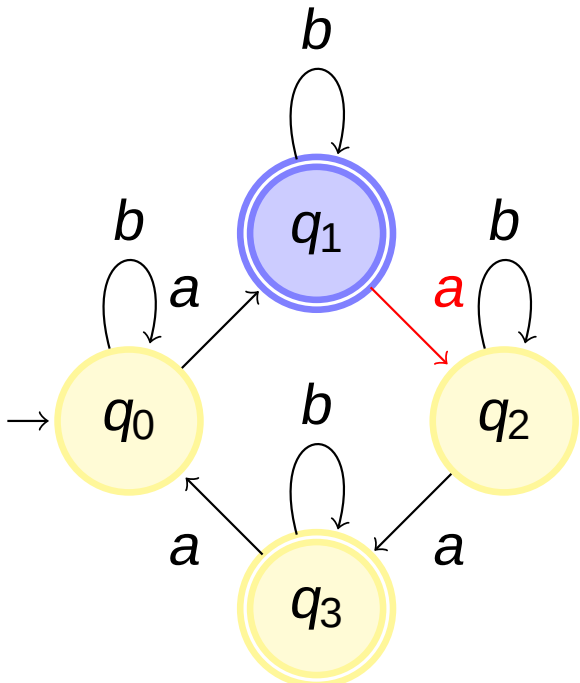


$$\dfrac{\dfrac{}{\vdash_\Pi a : q_1 \to q_0} \quad \dfrac{\dfrac{}{x^o : ? \vdash_\Pi a(a\,x^o) : q_1}}{x^o : ? \vdash_\Pi b(a(a\,x^o)) : q_1}}{x^o : ? \vdash_\Pi a(b(a(a\,x^o))) : q_0}$$
$$\vdash_\Pi \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

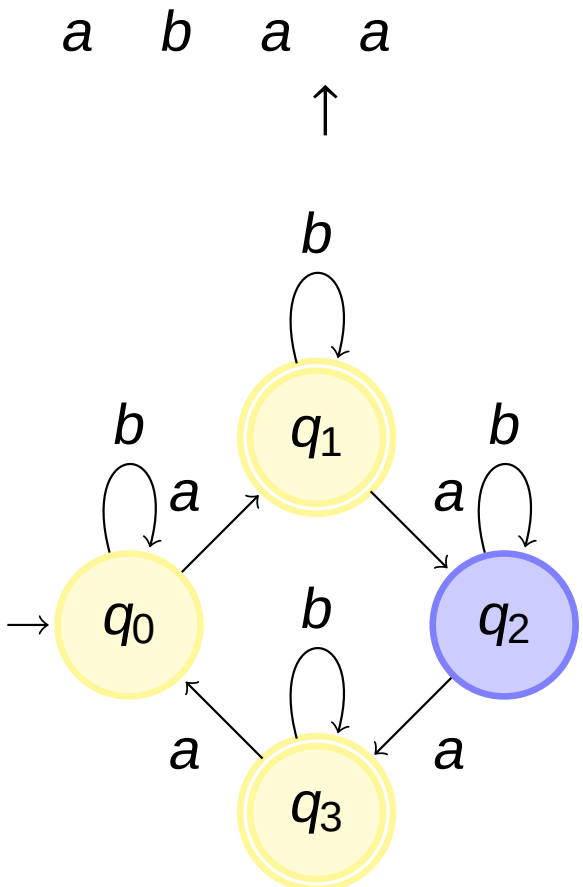
$a \quad b \quad a \quad a$
$\uparrow$



$$\dfrac{\dfrac{}{\vdash_\Pi a : q_1 \to q_0} \quad \dfrac{\dfrac{}{x^o : ? \vdash_\Pi a(a\,x^o) : q_1}}{x^o : ? \vdash_\Pi b(a(a\,x^o)) : q_1}}{x^o : ? \vdash_\Pi a(b(a(a\,x^o))) : q_0}$$
$$\vdash_\Pi \lambda x^o.a(b(a(a\,x^o))) : ? \to q_0$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

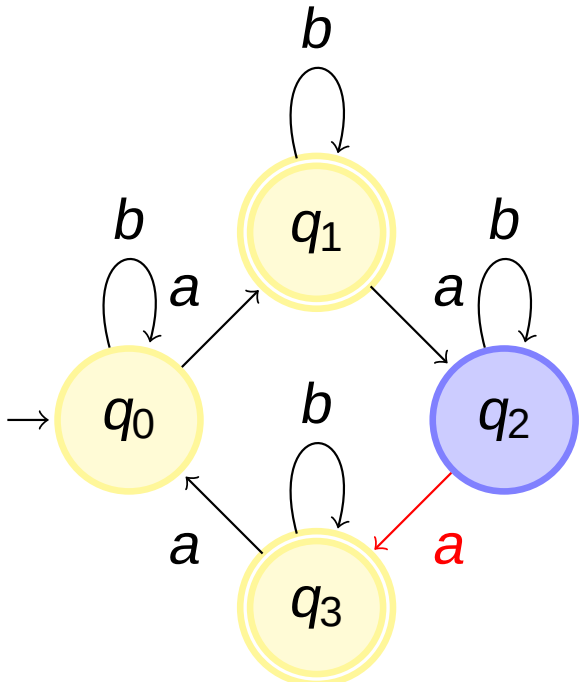


$$\dfrac{\dfrac{\dfrac{}{\vdash_\Pi a : q_1 \to q_0} \quad \dfrac{\dfrac{\dfrac{}{x^o : ? \vdash_\Pi x^o : q_2}}{x^o : ? \vdash_\Pi a\, x^o : q_1}}{x^o : ? \vdash_\Pi a(a\, x^o) : q_1}}{x^o : ? \vdash_\Pi b(a(a\, x^o)) : q_1}}{x^o : ? \vdash_\Pi a(b(a(a\, x^o))) : q_0}}{\vdash_\Pi \lambda x^o.a(b(a(a\, x^o))) : ? \to q_0}$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$



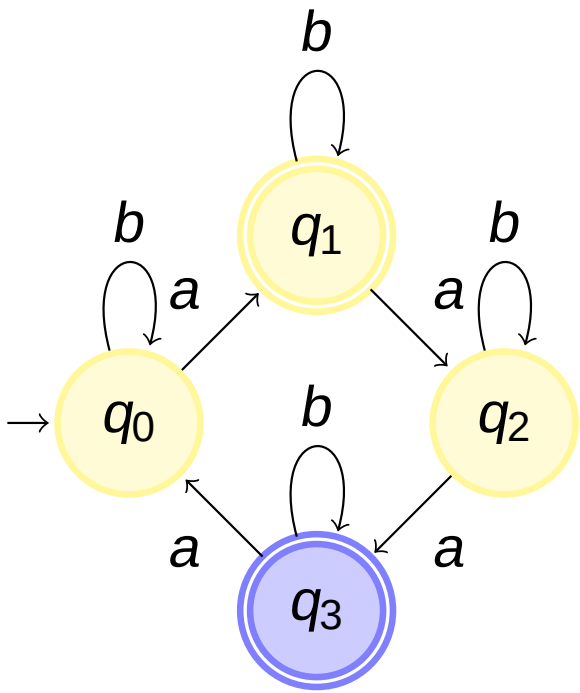
$$\dfrac{\dfrac{\dfrac{}{x^o : ? \vdash_\Pi a\, x^o : q_2}}{x^o : ? \vdash_\Pi a(a\, x^o) : q_1} \quad \dfrac{}{\vdash_\Pi a : q_1 \to q_0} \qquad \dfrac{}{x^o : ? \vdash_\Pi b(a(a\, x^o)) : q_1}}{x^o : ? \vdash_\Pi a(b(a(a\, x^o))) : q_0}$$

$$\vdash_\Pi \lambda x^o.a(b(a(a\, x^o))) : ? \to q_0$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$



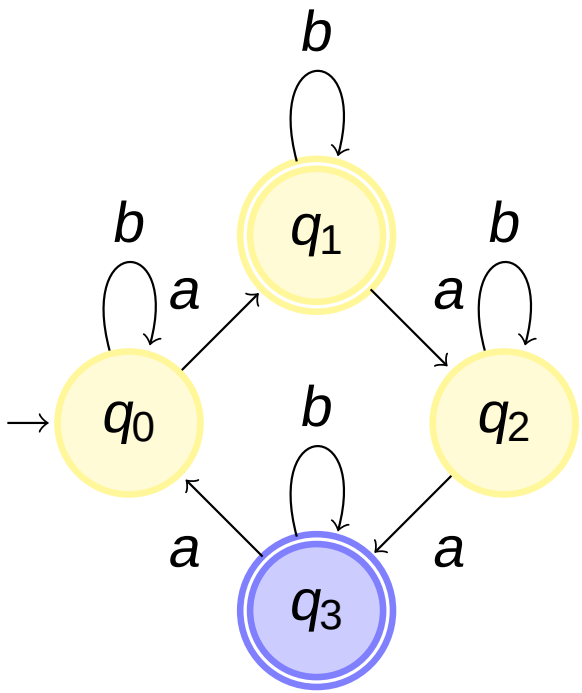
$$\dfrac{\dfrac{\dfrac{\dfrac{}{x^o : ? \vdash_\Pi x^o : q_3}}{x^o : ? \vdash_\Pi a\, x^o : q_2}}{x^o : ? \vdash_\Pi a(a\, x^o) : q_1}}{\dfrac{\vdash_\Pi a : q_1 \to q_0 \qquad x^o : ? \vdash_\Pi b(a(a\, x^o)) : q_1}{x^o : ? \vdash_\Pi a(b(a(a\, x^o))) : q_0}}$$

$$\vdash_\Pi \lambda x^o.a(b(a(a\, x^o))) : ? \to q_0$$

Example: a finite state automaton represented with intersection types

$a : (q_1 \to q_0) \cap (q_2 \to q_1) \cap (q_3 \to q_2) \cap (q_0 \to q_3)$
$b : (q_0 \to q_0) \cap (q_1 \to q_1) \cap (q_2 \to q_2) \cap (q_3 \to q_3)$
accepting types: $q_3 \to q_0$ or $q_1 \to q_0$

$a \quad b \quad a \quad a$
$\uparrow$



$$\dfrac{\dfrac{\dfrac{\dfrac{}{x^o : q_3 \vdash_\sqcap x^o : q_3}}{x^o : q_3 \vdash_\sqcap a\, x^o : q_2}}{x^o : q_3 \vdash_\sqcap a(a\, x^o) : q_1} \qquad \dfrac{}{\vdash_\sqcap a : q_1 \to q_0}}{\dfrac{x^o : q_3 \vdash_\sqcap b(a(a\, x^o)) : q_1}{x^o : q_3 \vdash_\sqcap a(b(a(a\, x^o))) : q_0}}$$
$$\vdash_\sqcap \lambda x^o.a(b(a(a\, x^o))) : q_3 \to q_0$$

HOIS and finite models

We have the two following effective theorems:

- **Theorem:** Given a model $\mathbb{M}$ and h in $\mathbb{M}$ there is a HOIS Π and a context Γ and a set of intersection types P such that $[\![M]\!]^{\mathbb{M}}_{\rho} = h$ iff the sequent $\Gamma \vdash_{\Pi} M : P$ is derivable.

HOIS and finite models

We have the two following effective theorems:

- **Theorem:** Given a model $\mathbb{M}$ and h in $\mathbb{M}$ there is a HOIS Π and a context Γ and a set of intersection types P such that $[\![M]\!]^{\mathbb{M}}_{\rho} = h$ iff the sequent $\Gamma \vdash_{\Pi} M : P$ is derivable.
- **Theorem:** Given Γ and p the set $\{M | \Gamma \vdash_{\Pi} M : p\}$ is recognizable.

HOIS and finite models

We have the two following effective theorems:

- **Theorem:** Given a model $\mathbb{M}$ and h in $\mathbb{M}$ there is a HOIS Π and a context Γ and a set of intersection types P such that $[\![M]\!]^{\mathbb{M}}_{\rho} = h$ iff the sequent $\Gamma \vdash_{\Pi} M : P$ is derivable.
- **Theorem:** Given Γ and p the set $\{M | \Gamma \vdash_{\Pi} M : p\}$ is recognizable.

Note: the first Theorem can be use to reduce the emptiness of intersection types to λ-definability and give a simple proof of the undecidability result by [Urzyczyn 1999].

Intersection types as automata

The two previous theorems imply that S is a recognizable set of λ-terms iff there is a HOIS Π, a context Γ and $\{S_1; \ldots; S_n\}$ such that for every $M \in S$ there is $k \in [1; n]$ so that $\Gamma \vdash_\Pi M : S_k$ is derivable.

Intersection types as automata

The two previous theorems imply that S is a recognizable set of λ-terms iff there is a HOIS Π, a context Γ and $\{S_1; \ldots; S_n\}$ such that for every $M \in S$ there is $k \in [1; n]$ so that $\Gamma \vdash_\Pi M : S_k$ is derivable.
A *type-automaton* is then defined as a tuple $\mathcal{A} = (\Pi, \Gamma, \{S_1; \ldots; S_n\})$ and:

$$L(\mathcal{A}) = \{M \mid \exists i \in [n].\Gamma \vdash_\Pi M : S_i\}$$

Intersection types as automata

The two previous theorems imply that S is a recognizable set of λ-terms iff there is a HOIS Π, a context Γ and $\{S_1; \ldots; S_n\}$ such that for every $M \in S$ there is $k \in [1; n]$ so that $\Gamma \vdash_\Pi M : S_k$ is derivable.
A *type-automaton* is then defined as a tuple $\mathcal{A} = (\Pi, \Gamma, \{S_1; \ldots; S_n\})$ and:

$$L(\mathcal{A}) = \{M \mid \exists i \in [n].\Gamma \vdash_\Pi M : S_i\}$$

Type-automata and finite state automata in a nutshell:

TA	**FSA**
types	states
typing axioms and typing rules	transitions
finite model	finite algebra/semigroup

Outline

Boolean closure properties

We can generalize the traditional constructions using automata to type-automata:

- intersection and union closure can be obtained by taking a generalization of the product of type-automata,

Boolean closure properties

We can generalize the traditional constructions using automata to type-automata:

- intersection and union closure can be obtained by taking a generalization of the product of type-automata,
- complement uses a generalized notion of determinism or finite models.

Homomorphism

Given two signature Σ_1 and Σ_2, $\mathcal{H}$ homomorphism from Σ_1 to Σ_2 if $\mathcal{H}$ is a pair of functions (g, h) such that:

- g maps types of Σ_1 to types of Σ_2 and $g(\alpha \to \beta) = g(\alpha) \to g(\beta)$

Homomorphism

Given two signature Σ_1 and Σ_2, $\mathcal{H}$ homomorphism from Σ_1 to Σ_2 if $\mathcal{H}$ is a pair of functions (g, h) such that:

- g maps types of Σ_1 to types of Σ_2 and $g(\alpha \to \beta) = g(\alpha) \to g(\beta)$
- h maps terms of Σ_1 to terms of Σ_2 and:
  - $h(x^\alpha) = x^{g(\alpha)}$

Homomorphism

Given two signature Σ_1 and Σ_2, $\mathcal{H}$ homomorphism from Σ_1 to Σ_2 if $\mathcal{H}$ is a pair of functions (g, h) such that:

- g maps types of Σ_1 to types of Σ_2 and $g(\alpha \to \beta) = g(\alpha) \to g(\beta)$
- h maps terms of Σ_1 to terms of Σ_2 and:
  - $h(x^\alpha) = x^{g(\alpha)}$
  - $h(c)$ is closed term of Σ_2 of type $g(\alpha)$ when c has type α

Homomorphism

Given two signature Σ_1 and Σ_2, $\mathcal{H}$ homomorphism from Σ_1 to Σ_2 if $\mathcal{H}$ is a pair of functions (g, h) such that:

- g maps types of Σ_1 to types of Σ_2 and $g(\alpha \to \beta) = g(\alpha) \to g(\beta)$
- h maps terms of Σ_1 to terms of Σ_2 and:
  - $h(x^\alpha) = x^{g(\alpha)}$
  - $h(c)$ is closed term of Σ_2 of type $g(\alpha)$ when c has type α
  - $h(MN) = h(M)h(N)$

Homomorphism

Given two signature Σ_1 and Σ_2, $\mathcal{H}$ homomorphism from Σ_1 to Σ_2 if $\mathcal{H}$ is a pair of functions (g, h) such that:

- g maps types of Σ_1 to types of Σ_2 and $g(\alpha \to \beta) = g(\alpha) \to g(\beta)$
- h maps terms of Σ_1 to terms of Σ_2 and:
  - $h(x^\alpha) = x^{g(\alpha)}$
  - $h(c)$ is closed term of Σ_2 of type $g(\alpha)$ when c has type α
  - $h(MN) = h(M)h(N)$
  - $h(\lambda x^\alpha.M) = \lambda x^{g(\alpha)}.h(M)$

Non-closure under homomorphism

As for trees, recognizability is not closed under homomorphism and:

- it is not closed under linear homomorphism,

Non-closure under homomorphism

As for trees, recognizability is not closed under homomorphism and:

- it is not closed under linear homomorphism,
- it is not closed under first order linear homomorphism. A counter-example is:

$$R = \{a(\lambda x.bx\, x)(c^n e) | \in \mathbb{N}\},\ \mathcal{H}(R) = \{b(c^n e)(c^n e) | n \in \mathbb{N}\}$$

and $\mathcal{H}(a) = \lambda fx.fx$ and $\mathcal{H}$ is the identy otherwise.

Non-closure under homomorphism

As for trees, recognizability is not closed under homomorphism and:

- it is not closed under linear homomorphism,
- it is not closed under first order linear homomorphism. A counter-example is:

$$R = \{a(\lambda x.bx\, x)(c^n e) | \in \mathbb{N}\},\ \mathcal{H}(R) = \{b(c^n e)(c^n e) | n \in \mathbb{N}\}$$

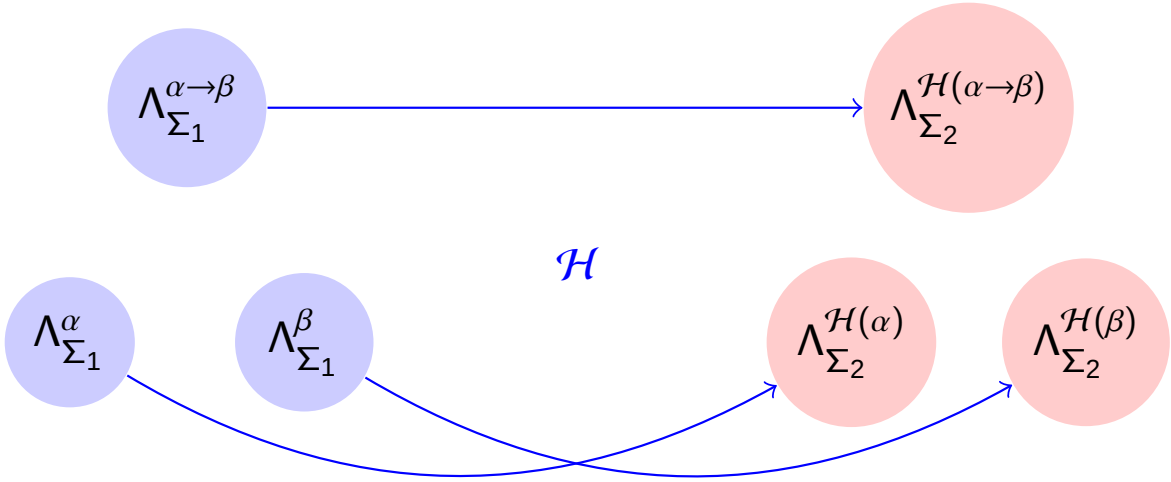
  and $\mathcal{H}(a) = \lambda fx.fx$ and $\mathcal{H}$ is the identy otherwise.
- it is not even closed under relabeling.

Closure under inverse homomorphism

Theorem: Given Σ_1, Σ_2 two HOS and $\mathcal{H}$ a homomorphism from Σ_1 to Σ_2, if R is a recognizable set of Σ_2 then $\mathcal{H}^{-1}(R)$ is also recognizable.

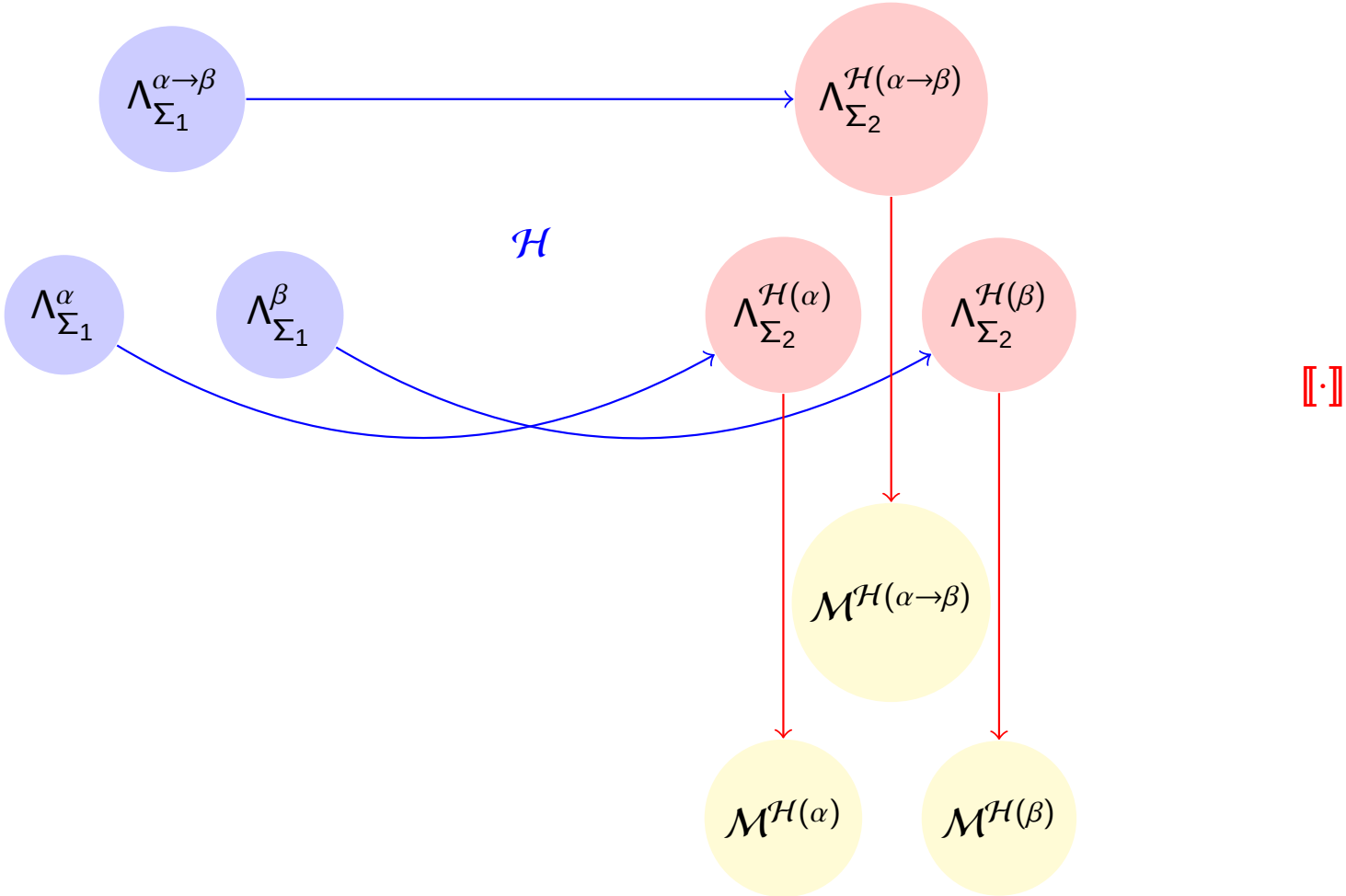
Closure under inverse homomorphism

Theorem: Given Σ_1, Σ_2 two HOS and $\mathcal{H}$ a homomorphism from Σ_1 to Σ_2, if R is a recognizable set of Σ_2 then $\mathcal{H}^{-1}(R)$ is also recognizable.



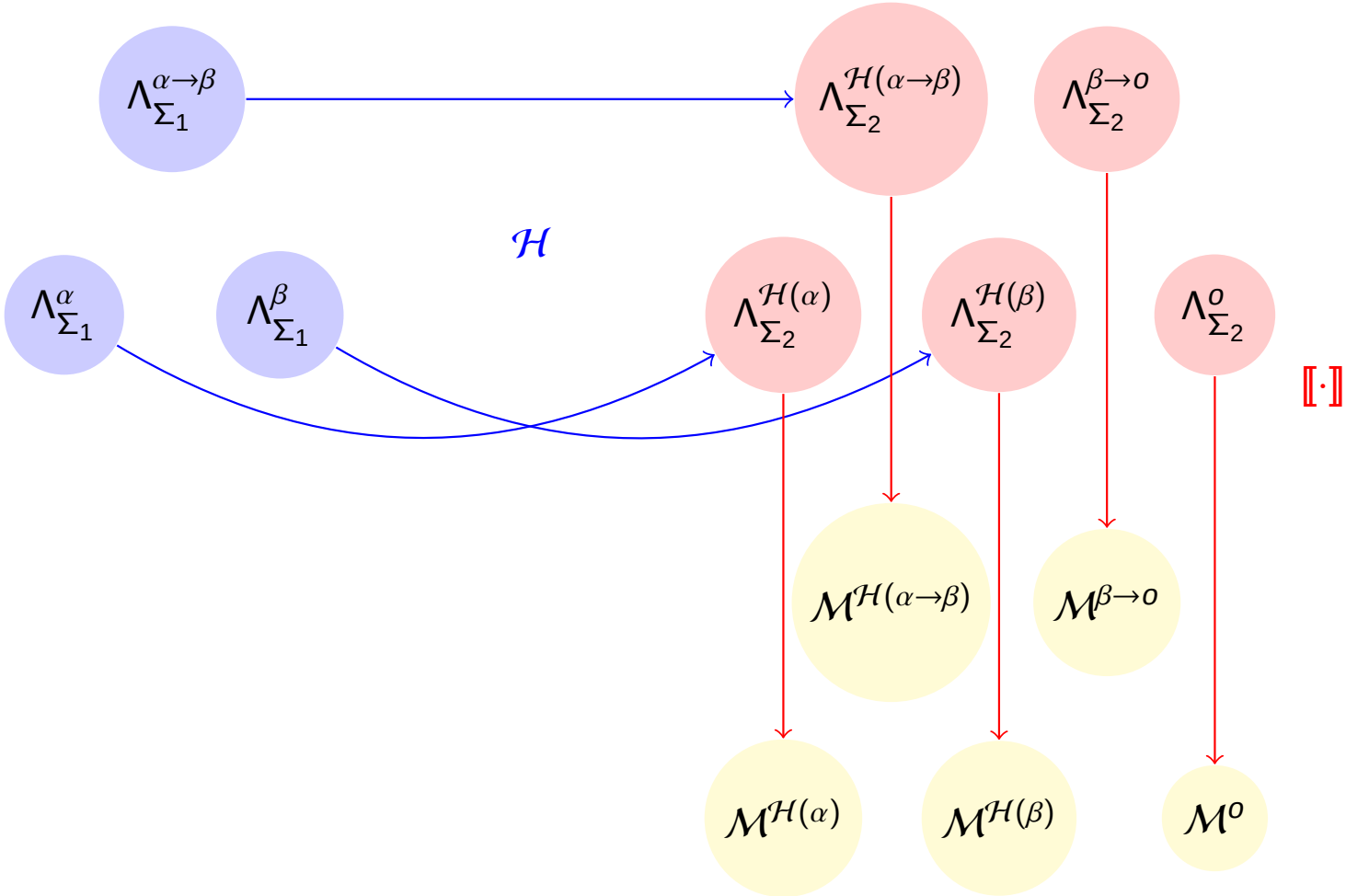
Closure under inverse homomorphism

Theorem: Given Σ_1, Σ_2 two HOS and $\mathcal{H}$ a homomorphism from Σ_1 to Σ_2, if R is a recognizable set of Σ_2 then $\mathcal{H}^{-1}(R)$ is also recognizable.



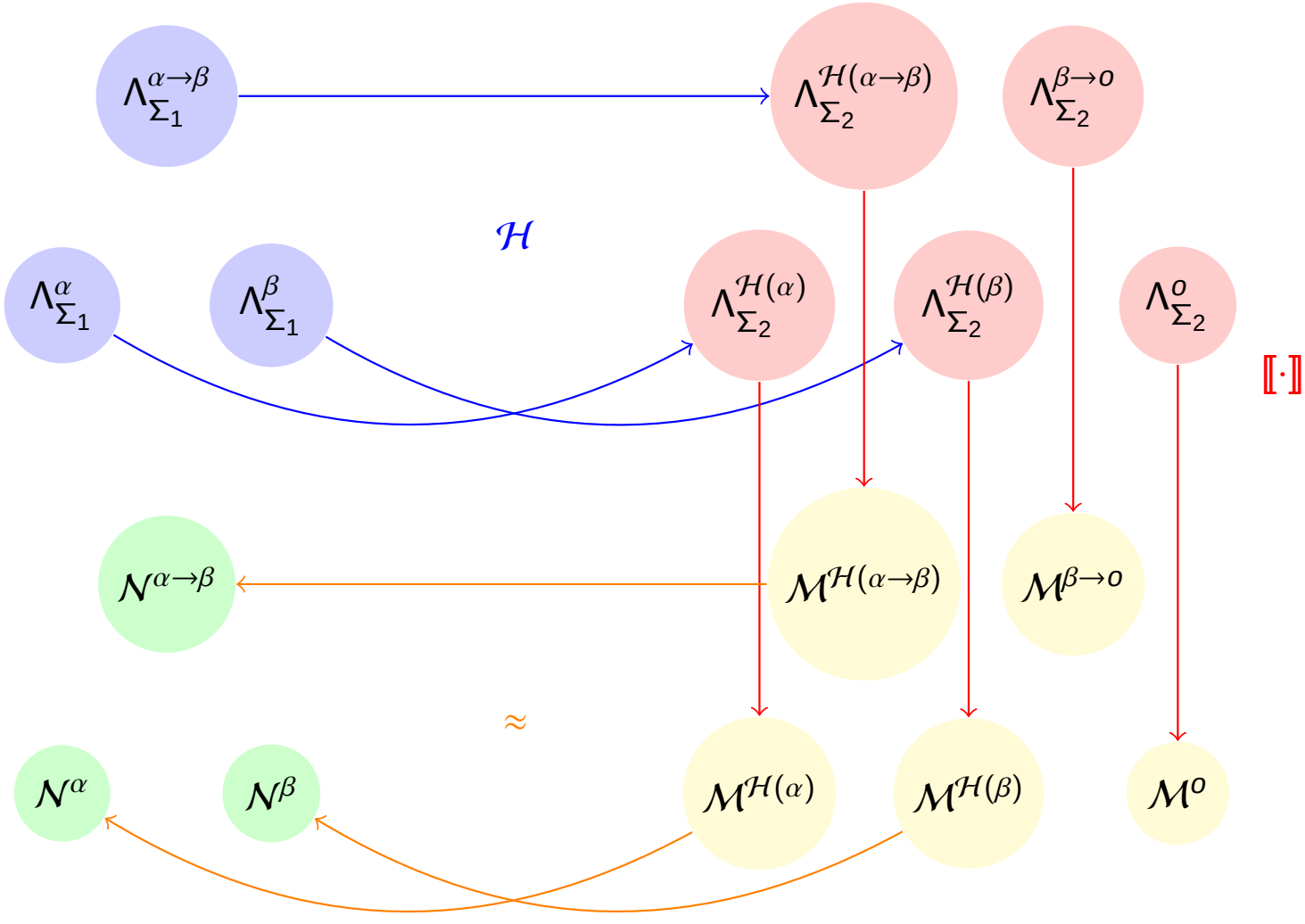
Closure under inverse homomorphism

Theorem: Given Σ_1, Σ_2 two HOS and $\mathcal{H}$ a homomorphism from Σ_1 to Σ_2, if R is a recognizable set of Σ_2 then $\mathcal{H}^{-1}(R)$ is also recognizable.



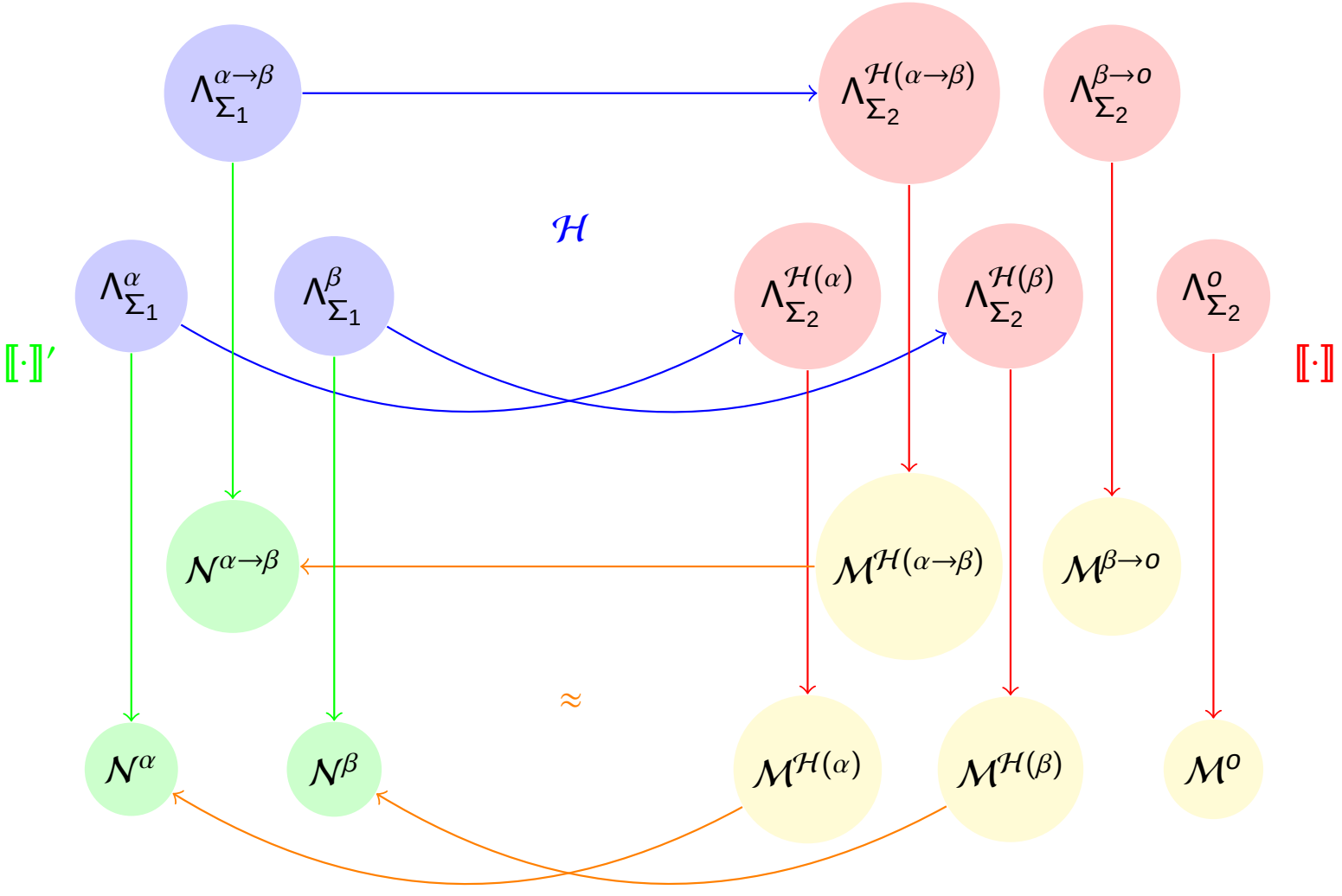
Closure under inverse homomorphism

Theorem: Given Σ_1, Σ_2 two HOS and $\mathcal{H}$ a homomorphism from Σ_1 to Σ_2, if R is a recognizable set of Σ_2 then $\mathcal{H}^{-1}(R)$ is also recognizable.



Closure under inverse homomorphism

Theorem: Given Σ_1, Σ_2 two HOS and $\mathcal{H}$ a homomorphism from Σ_1 to Σ_2, if R is a recognizable set of Σ_2 then $\mathcal{H}^{-1}(R)$ is also recognizable.



Outline

λ-context-free grammars

A context free grammar of lambda-term is a 4-tuple $(\Sigma_1, \Sigma_2, \mathcal{H}, S)$ where:

- Σ_1 is a second order signature, the *rule signature*

λ-context-free grammars

A context free grammar of lambda-term is a 4-tuple $(\Sigma_1, \Sigma_2, \mathcal{H}, S)$ where:

- Σ_1 is a second order signature, the *rule signature*
- Σ_2 is a signature, *the object signature*

λ-context-free grammars

A context free grammar of lambda-term is a 4-tuple $(\Sigma_1, \Sigma_2, \mathcal{H}, S)$ where:

- Σ_1 is a second order signature, the *rule signature*
- Σ_2 is a signature, *the object signature*
- $\mathcal{H}$ is a homomorphism from Σ_1 to Σ_2

λ-context-free grammars

A context free grammar of lambda-term is a 4-tuple $(\Sigma_1, \Sigma_2, \mathcal{H}, S)$ where:

- Σ_1 is a second order signature, the *rule signature*
- Σ_2 is a signature, *the object signature*
- $\mathcal{H}$ is a homomorphism from Σ_1 to Σ_2
- S is an atomic type of Σ_1, the type of valid derivations.

λ-context-free grammars

A context free grammar of lambda-term is a 4-tuple $(\Sigma_1, \Sigma_2, \mathcal{H}, S)$ where:

- Σ_1 is a second order signature, the *rule signature*
- Σ_2 is a signature, *the object signature*
- $\mathcal{H}$ is a homomorphism from Σ_1 to Σ_2
- S is an atomic type of Σ_1, the type of valid derivations.

Context free grammars of λ-terms subsume many formalisms like CFG, TAG, MCFG, PMCFG, MGs, IO-grammars,...

Parsing of λ-context-free grammars

Given a λ-CFG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, S)$:

- given w, $w \in \mathcal{L}(\mathcal{G})$ iff $\mathcal{H}^{-1}(\{w\}) \neq \emptyset$,

Parsing of λ-context-free grammars

Given a λ-CFG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, S)$:

- given w, $w \in \mathcal{L}(\mathcal{G})$ iff $\mathcal{H}^{-1}(\{w\}) \neq \emptyset$,
- $\mathcal{H}^{-1}(\{w\})$ is a recognizable set of trees and its emptyness is decidable,

Parsing of λ-context-free grammars

Given a λ-CFG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, S)$:

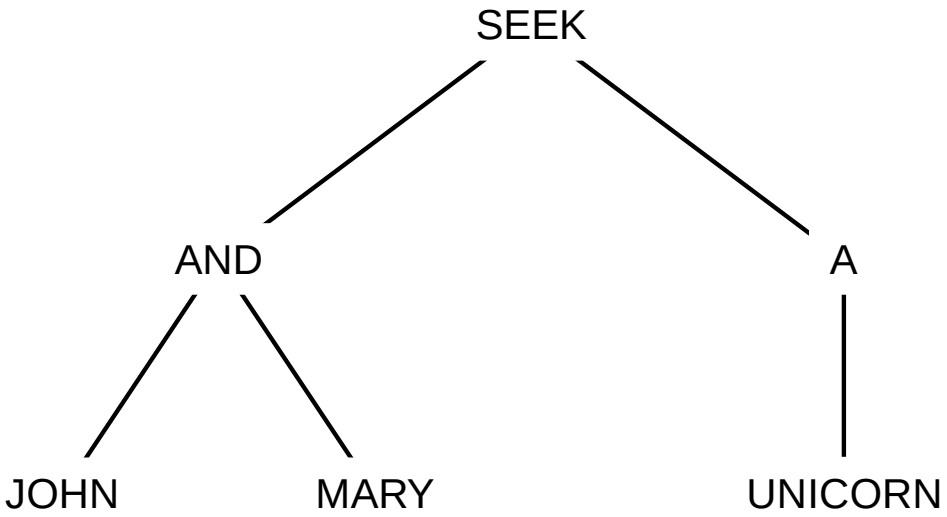
- given w, $w \in \mathcal{L}(\mathcal{G})$ iff $\mathcal{H}^{-1}(\{w\}) \neq \emptyset$,
- $\mathcal{H}^{-1}(\{w\})$ is a recognizable set of trees and its emptyness is decidable,
- this gives a simple generalization of Thatcher Theorem on the derivations of context free formalisms,

Parsing of λ-context-free grammars

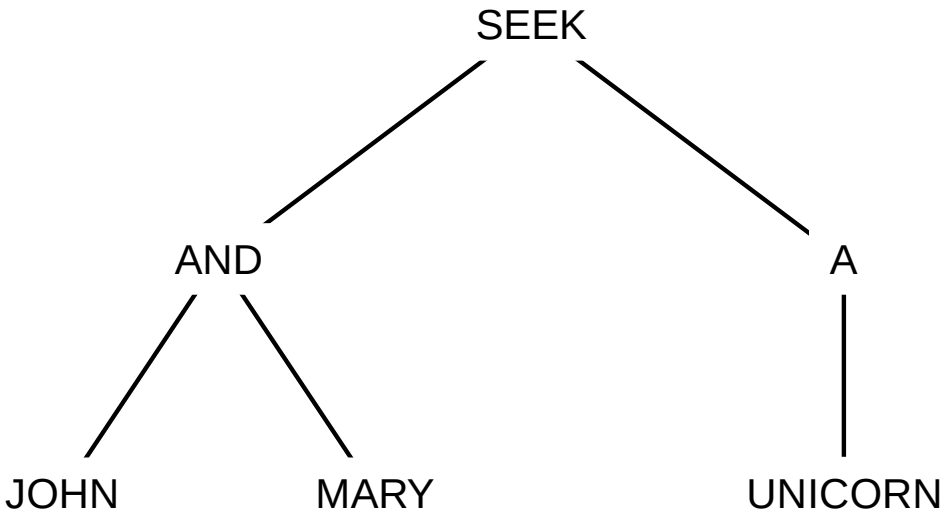
Given a λ-CFG $\mathcal{G} = (\Sigma_1, \Sigma_2, \mathcal{H}, S)$:

- given w, $w \in \mathcal{L}(\mathcal{G})$ iff $\mathcal{H}^{-1}(\{w\}) \neq \emptyset$,
- $\mathcal{H}^{-1}(\{w\})$ is a recognizable set of trees and its emptyness is decidable,
- this gives a simple generalization of Thatcher Theorem on the derivations of context free formalisms,
- furthermore this technique allows to show that parsing is a particular case of finding the inverse image of recognizable sets. The result extends to parsing recognizable sets.

Generation in Montague semantics



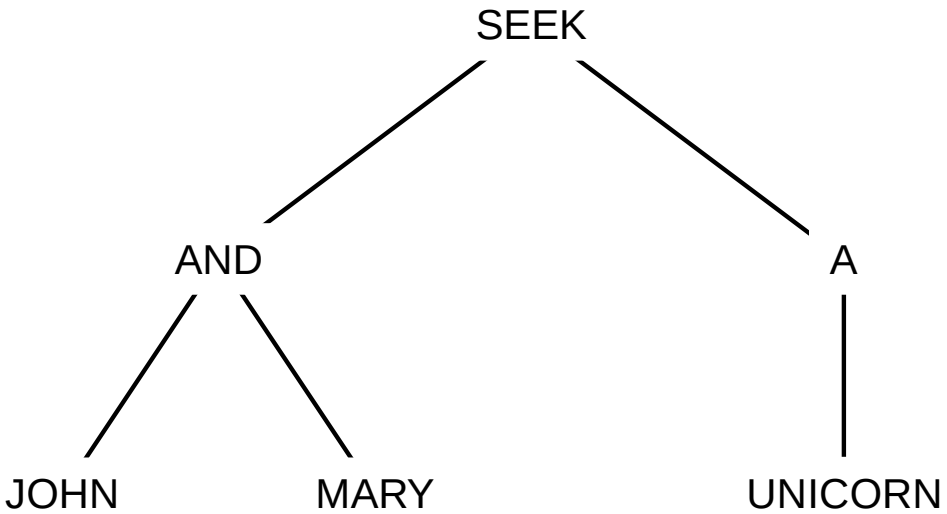
Generation in Montague semantics



Jean et Marie cherchent une licorne

$$
\begin{aligned}
\text{SEEK} &= \lambda xy.x \text{ cherchent } y : np \to np \to s \\
\text{AND} &= \lambda xy.x \text{ et } y : np \to np \to np \\
\text{JOHN} &= \text{Jean} : np \\
\text{MARY} &= \text{Marie} : np \\
\text{A} &= \lambda x.\text{une } x : n \to np \\
\text{UNICORN} &= \text{licorne} : n
\end{aligned}
$$

Generation in Montague semantics



$$\exists(\lambda x.\ \wedge\ (\textsc{unicorn}\ x)(\wedge(\textsc{seek}\ \mathbf{j}\ x)(\textsc{seek}\ \mathbf{m}\ x)))$$
$$\approx$$
$$\exists x.\textsc{unicorn}(x) \wedge \textsc{seek}(\mathbf{j}, x) \wedge \textsc{seek}(\mathbf{m}, x)$$

$$\begin{aligned}
\text{SEEK} &= \lambda S\, O.O(\lambda x.S(\lambda y.\textsc{seek}\ y\ x)) : np \to np \to s \\
\text{AND} &= \lambda N_1 N_2 P.\ \wedge\ (N_1 P)(N_2 P) : np \to np \to np \\
\text{JOHN} &= \lambda P.P\,\mathbf{j} : np \\
\text{MARY} &= \lambda P.P\,\mathbf{m} : np \\
\text{A} &= \lambda P\, Q.\exists(\lambda x.\ \wedge\ (P\ x)(Q\ x)) : n \to np \\
\text{UNICORN} &= \lambda x.\textsc{unicorn}\ x : n
\end{aligned}$$

Another corollary: decidability of 4th order matching

- From finite completeness we know that there is a model $\mathbb{M}$ and an element e of that model such that whenever $[\![t]\!]_{\mathbb{M}} = e$ then $t =_{\beta\eta} u$. Solving the equation in the model entails that the solution of a matching problem form a recognizable set.

Another corollary: decidability of 4th order matching

- From finite completeness we know that there is a model $\mathbb{M}$ and an element e of that model such that whenever $[\![t]\!]_{\mathbb{M}} = e$ then $t =_{\beta\eta} u$. Solving the equation in the model entails that the solution of a matching problem form a recognizable set.
- **Lemma (Schmidt-Schauss)**: if a fourth order equation has a solution then it has a solution such that all its subterm contain a bounded number of free variables.

Another corollary: decidability of 4th order matching

- From finite completeness we know that there is a model $\mathbb{M}$ and an element e of that model such that whenever $[\![t]\!]_{\mathbb{M}} = e$ then $t =_{\beta\eta} u$. Solving the equation in the model entails that the solution of a matching problem form a recognizable set.
- **Lemma (Schmidt-Schauss)**: if a fourth order equation has a solution then it has a solution such that all its subterm contain a bounded number of free variables.
- **Lemma** The set of terms whose subterms only contain a bounded number of free variables can be described with a λ-context context free grammar.

Another corollary: decidability of 4th order matching

- From finite completeness we know that there is a model $\mathbb{M}$ and an element e of that model such that whenever $[\![t]\!]_{\mathbb{M}} = e$ then $t =_{\beta\eta} u$. Solving the equation in the model entails that the solution of a matching problem form a recognizable set.
- **Lemma (Schmidt-Schauss)**: if a fourth order equation has a solution then it has a solution such that all its subterm contain a bounded number of free variables.
- **Lemma** The set of terms whose subterms only contain a bounded number of free variables can be described with a λ-context context free grammar.
- Decidability of fourth order matching becomes then a corollary of the closure of λ-context context free grammars under intersection with recognizable sets of λ-terms.

Outline

The other views on recognizability

Presently: λ-REC is defined with finite models (algebraic view) and intersection types (automaton view). The other views on recognizability are still to be explored.

- Congurential view: having canonical models that are only related to a language. The difficulty comes from the fact that standard models of the λ-calculus are not so well-behaved. Maybe we will need to turn to locally finite Cartesian Closed Categories.

The other views on recognizability

Presently: λ-REC is defined with finite models (algebraic view) and intersection types (automaton view). The other views on recognizability are still to be explored.

- Congurential view: having canonical models that are only related to a language. The difficulty comes from the fact that standard models of the λ-calculus are not so well-behaved. Maybe we will need to turn to locally finite Cartesian Closed Categories.
- The logical view:
  - recognizable sets of λ-terms do not seem to be definable with a taylor-made MSOL (recognizable sets are not closed under relabeling)
  - a possibility is that modal μ-calculus is more adapted.

The other views on recognizability

Presently: λ-REC is defined with finite models (algebraic view) and intersection types (automaton view). The other views on recognizability are still to be explored.

- Congurential view: having canonical models that are only related to a language. The difficulty comes from the fact that standard models of the λ-calculus are not so well-behaved. Maybe we will need to turn to locally finite Cartesian Closed Categories.
- The logical view:
  - recognizable sets of λ-terms do not seem to be definable with a taylor-made MSOL (recognizable sets are not closed under relabeling)
  - a possibility is that modal μ-calculus is more adapted.
- Regular expressions: defining regular expressions of λ-terms is challenging since it requires to handle an unbounded number of free variables.

λ-calculus, higher-order programming schemes, and higher-order (collapsible) pushdown automata

Recognizable sets of λ-terms are well-behaved with context-free definitions and as such they have a neat connection with higher-order programming schemes.

- We already have a simple proof of Kobayashi's results for model-checking higher-order schemes.

λ-calculus, higher-order programming schemes, and higher-order (collapsible) pushdown automata

Recognizable sets of λ-terms are well-behaved with context-free definitions and as such they have a neat connection with higher-order programming schemes.

- We already have a simple proof of Kobayashi's results for model-checking higher-order schemes.
- It remains to understand Kobayashi's and Ong's result in this setting.

λ-calculus, higher-order programming schemes, and higher-order (collapsible) pushdown automata

Recognizable sets of λ-terms are well-behaved with context-free definitions and as such they have a neat connection with higher-order programming schemes.

- We already have a simple proof of Kobayashi's results for model-checking higher-order schemes.
- It remains to understand Kobayashi's and Ong's result in this setting.
- Maybe a generalization can be achieved for schemes generating Böhm trees (with bindings) rather than trees.

λ-calculus, higher-order programming schemes, and higher-order (collapsible) pushdown automata

Recognizable sets of λ-terms are well-behaved with context-free definitions and as such they have a neat connection with higher-order programming schemes.

- We already have a simple proof of Kobayashi's results for model-checking higher-order schemes.
- It remains to understand Kobayashi's and Ong's result in this setting.
- Maybe a generalization can be achieved for schemes generating Böhm trees (with bindings) rather than trees.
- We also need to articulate the invariants of recognizability with higher-order pushdown automata (*c.f.* Fratani/Senizergues's and Carayol's notions of recognizable HO-stacks).

λ-calculus, higher-order programming schemes, and higher-order (collapsible) pushdown automata

Recognizable sets of λ-terms are well-behaved with context-free definitions and as such they have a neat connection with higher-order programming schemes.

- We already have a simple proof of Kobayashi's results for model-checking higher-order schemes.
- It remains to understand Kobayashi's and Ong's result in this setting.
- Maybe a generalization can be achieved for schemes generating Böhm trees (with bindings) rather than trees.
- We also need to articulate the invariants of recognizability with higher-order pushdown automata (*c.f.* Fratani/Senizergues's and Carayol's notions of recognizable HO-stacks).
- An on-going work establishing the connection of Krivine machines with higher-order collapsible automata, may help along this line, it may also connect games on Krivine machines with games on higher-order (collapsible) automata.

Recognizability in the λ-calculus and profinite topology

In the literature on intersection types, Stone duality is a classical tool to construct models of the λ-calculus.

- A first line of research consists in trying to extend Gehrke, Grigorieff and Pin's approach to recognizability to recognizability for the λ-calculus.

Recognizability in the λ-calculus and profinite topology

In the literature on intersection types, Stone duality is a classical tool to construct models of the λ-calculus.

- A first line of research consists in trying to extend Gehrke, Grigorieff and Pin's approach to recognizability to recognizability for the λ-calculus.
- Another one consists in strudying Weil's pre-clones within the settled framework.

Recognizability in the λ-calculus and profinite topology

In the literature on intersection types, Stone duality is a classical tool to construct models of the λ-calculus.

- A first line of research consists in trying to extend Gehrke, Grigorieff and Pin's approach to recognizability to recognizability for the λ-calculus.
- Another one consists in strudying Weil's pre-clones within the settled framework.
- Finally we may try to generalize the notion of implicit operations as defined by Almeida by using higher-order operations as definable by λ-terms.