On regular languages of λ-terms

Lê Thành Dũng (Tito) Nguyễn — nltd@nguyentito.eu – École normale supérieure de Lyon
joint work with Vincent Moreau (IRIF, Université Paris Cité)

26 April 2024, Journées du GT DAAL

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\; (f_b\; (f_b\; x))$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\; (f_b\; (f_b\; x))$

$$\overline{abb}\;\texttt{not}\;\texttt{id}\;\texttt{true} \longrightarrow^*_\beta \texttt{not}\;(\texttt{id}\;(\texttt{id}\;\texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\ t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\ t)\ u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

Booleans

$\texttt{true} = \lambda x.\ \lambda y.\ x \qquad \texttt{false} = \lambda x.\ \lambda y.\ y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\ \lambda f_b.\ \lambda x.\ f_a\ (f_b\ (f_b\ x))$

$$\overline{abb}\ \texttt{not}\ \texttt{id}\ \texttt{true} \longrightarrow^*_\beta \texttt{not}\ (\texttt{id}\ (\texttt{id}\ \texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\ t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\ t)\ u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)

Booleans

$\texttt{true} = \lambda x.\ \lambda y.\ x \qquad \texttt{false} = \lambda x.\ \lambda y.\ y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\ \lambda f_b.\ \lambda x.\ f_a\ (f_b\ (f_b\ x))$

$$\overline{abb}\ \texttt{not}\ \texttt{id}\ \texttt{true} \longrightarrow_\beta^* \texttt{not}\ (\texttt{id}\ (\texttt{id}\ \texttt{true})) \longrightarrow_\beta^* \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\ t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\ t)\ u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

Booleans

$\texttt{true} = \lambda x.\ \lambda y.\ x \qquad \texttt{false} = \lambda x.\ \lambda y.\ y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\ \lambda f_b.\ \lambda x.\ f_a\ (f_b\ (f_b\ x))$

$$\overline{abb}\ \texttt{not}\ \texttt{id}\ \texttt{true} \longrightarrow_\beta^* \texttt{not}\ (\texttt{id}\ (\texttt{id}\ \texttt{true})) \longrightarrow_\beta^* \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\; (f_b\; (f_b\; x))$

$$\overline{abb}\;\texttt{not}\;\texttt{id}\;\texttt{true} \longrightarrow^*_\beta \texttt{not}\;(\texttt{id}\;(\texttt{id}\;\texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$
$\texttt{Bool} = o \to (o \to o) = o \to o \to o$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\; (f_b\; (f_b\; x))$

$$\overline{abb}\;\texttt{not}\;\texttt{id}\;\texttt{true} \longrightarrow^*_\beta \texttt{not}\;(\texttt{id}\;(\texttt{id}\;\texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

$\texttt{Bool} = o \to (o \to o) = o \to o \to o$

so that $t : \texttt{Bool} \iff t =_\beta \texttt{true}$ or $t =_\beta \texttt{false}$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\; (f_b\; (f_b\; x))$

$$\overline{abb}\;\texttt{not}\;\texttt{id}\;\texttt{true} \longrightarrow^*_\beta \texttt{not}\;(\texttt{id}\;(\texttt{id}\;\texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$
$\texttt{Bool} = o \to (o \to o) = o \to o \to o$
so that $t : \texttt{Bool} \iff t =_\beta \texttt{true}$ or $t =_\beta \texttt{false}$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\ (f_b\ (f_b\ x))$
$\texttt{Str}_{\{a,b\}} = (o \to o) \to (o \to o) \to o \to o$

$$\overline{abb}\ \texttt{not}\ \texttt{id}\ \texttt{true} \longrightarrow^*_\beta \texttt{not}\ (\texttt{id}\ (\texttt{id}\ \texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$
$\texttt{Bool} = o \to (o \to o) = o \to o \to o$
so that $t : \texttt{Bool} \iff t =_\beta \texttt{true}$ or $t =_\beta \texttt{false}$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\; (f_b\; (f_b\; x))$
$\texttt{Str}_{\{a,b\}} = (o \to o) \to (o \to o) \to o \to o$

$\overline{w} : \texttt{Str}_\Sigma$ therefore $\overline{w} : \texttt{Str}_\Sigma\{o := A\}$ for all A

$$\overline{abb}\;\texttt{not}\;\texttt{id}\;\texttt{true} \longrightarrow_\beta^* \texttt{not}\;(\texttt{id}\;(\texttt{id}\;\texttt{true})) \longrightarrow_\beta^* \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$
$\texttt{Bool} = o \to (o \to o) = o \to o \to o$
so that $t : \texttt{Bool} \iff t =_\beta \texttt{true}$ or $t =_\beta \texttt{false}$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\ (f_b\ (f_b\ x))$
$\texttt{Str}_{\{a,b\}} = (o \to o) \to (o \to o) \to o \to o$

$\overline{w} : \texttt{Str}_\Sigma$ therefore $\overline{w} : \texttt{Str}_\Sigma\{o := A\}$ for all A

Theorem (Hillebrand & Kanellakis 1996)

The language $L \subseteq \Sigma^*$ is regular $\iff$
L is defined by some $t : \texttt{Str}_\Sigma\{o := A\} \to \texttt{Bool}$
in the simply typed λ-calculus
(*A: arbitrary simple type, may depend on L*)

$$\overline{abb}\ \texttt{not}\ \texttt{id}\ \texttt{true} \longrightarrow^*_\beta \texttt{not}\ (\texttt{id}\ (\texttt{id}\ \texttt{true})) \longrightarrow^*_\beta \texttt{false}$$

A λ-calculus crash course: Church encodings and simple types

A naive syntactic theory of functions:

$$f\,t \;\approx\; f(t) \qquad \lambda x.\, t \;\approx\; (x \mapsto t)$$

For $f : x \mapsto x^2 + 1$, we have $f(42) = 42^2 + 1$
$\rightsquigarrow$ Execution by rewriting, using substitution

$$(\lambda x.\, t)\; u \longrightarrow_\beta t\{x := u\}$$

No primitive data types $\rightsquigarrow$ encode as functions

Simple types: specifications for λ-terms

- base type o (no constant of type o)
- $A \to B$ = "functions from A to B"

$t : A$ ("t is of type A") defined inductively

Exemple : even number of 'a's

$t = \lambda s.\, s$ `not id true` : $\texttt{Str}_{\{a,b\}}\{o := \texttt{Bool}\} \to \texttt{Bool}$

Booleans

$\texttt{true} = \lambda x.\, \lambda y.\, x \qquad \texttt{false} = \lambda x.\, \lambda y.\, y$

$\texttt{Bool} = o \to (o \to o) = o \to o \to o$

so that $t : \texttt{Bool} \iff t =_\beta \texttt{true}$ or $t =_\beta \texttt{false}$

(two arguments: $x \mapsto (y \mapsto x) \cong (x, y) \mapsto x$)

Church encoding of strings (1930s?)

$\overline{abb} = \lambda f_a.\, \lambda f_b.\, \lambda x.\, f_a\ (f_b\ (f_b\ x))$

$\texttt{Str}_{\{a,b\}} = (o \to o) \to (o \to o) \to o \to o$

$\overline{w} : \texttt{Str}_\Sigma$ therefore $\overline{w} : \texttt{Str}_\Sigma\{o := A\}$ for all A

Theorem (Hillebrand & Kanellakis 1996)

The language $L \subseteq \Sigma^*$ is regular $\iff$
L is defined by some $t : \texttt{Str}_\Sigma\{o := A\} \to \texttt{Bool}$
in the simply typed λ-calculus

(*A: arbitrary simple type, may depend on L*)

$$t\ \overline{abb} \longrightarrow_\beta \overline{abb}\ \texttt{not id true} \longrightarrow_\beta^* \texttt{not}\ (\texttt{id}\ (\texttt{id true})) \longrightarrow_\beta^* \texttt{false}$$

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Naive semantics of simply typed λ-terms:

$t : A \implies [\![t]\!]_Q \in [\![A]\!]_Q$ where

$$[\![o]\!]_Q = Q \text{ (an arbitrary set)}$$

$$[\![A \to B]\!]_Q = [\![A]\!]_Q \to [\![B]\!]_Q = [\![B]\!]_Q^{[\![A]\!]_Q}$$

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Naive semantics of simply typed λ-terms:
$t : A \implies [\![t]\!]_Q \in [\![A]\!]_Q$ where

$$[\![o]\!]_Q = Q \text{ (an arbitrary set)}$$

$$[\![A \to B]\!]_Q = [\![A]\!]_Q \to [\![B]\!]_Q = [\![B]\!]_Q^{[\![A]\!]_Q}$$

- *compositional* by def., e.g. $[\![t\,u]\!]_Q = [\![t]\!]_Q\,([\![u]\!]_Q)$
  + invariant modulo $=_\beta$
- Q finite $\implies$ every $[\![A]\!]_Q$ finite

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

For languages of type $A = \mathtt{Str}_\Sigma$:

usual reg. $\Rightarrow$ **synt. reg.** programming exercise

synt. reg. $\Rightarrow$ **sem. reg.** take $Q = [\![B]\!]_{\{0,1\}}$ ...

sem. reg. $\Rightarrow$ **usual** for $w \in \Sigma^*$, compute $[\![\overline{w}]\!]$ by DFA

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Naive semantics of simply typed λ-terms:

$t : A \implies [\![t]\!]_Q \in [\![A]\!]_Q$ where

$$[\![o]\!]_Q = Q \text{ (an arbitrary set)}$$

$$[\![A \to B]\!]_Q = [\![A]\!]_Q \to [\![B]\!]_Q = [\![B]\!]_Q^{[\![A]\!]_Q}$$

- *compositional* by def., e.g. $[\![t\,u]\!]_Q = [\![t]\!]_Q\,([\![u]\!]_Q)$
  + invariant modulo $=_\beta$
- Q finite $\implies$ every $[\![A]\!]_Q$ finite

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

For languages of type $A = \mathtt{Str}_\Sigma$:

usual reg. $\Rightarrow$ **synt. reg.** programming exercise

synt. reg. $\Rightarrow$ **sem. reg.** take $Q = [\![B]\!]_{\{0,1\}}$ ...

sem. reg. $\Rightarrow$ **usual** for $w \in \Sigma^*$, compute $[\![\overline{w}]\!]$ by DFA

Theorem (Moreau & N., CSL'24)

$\forall A$, *syntactically regular* $\iff$ *semantically regular*

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Naive semantics of simply typed λ-terms:

$t : A \implies [\![t]\!]_Q \in [\![A]\!]_Q$ where

$$[\![o]\!]_Q = Q \text{ (an arbitrary set)}$$

$$[\![A \to B]\!]_Q = [\![A]\!]_Q \to [\![B]\!]_Q = [\![B]\!]_Q^{[\![A]\!]_Q}$$

- *compositional* by def., e.g. $[\![t\,u]\!]_Q = [\![t]\!]_Q\,([\![u]\!]_Q)$
  + invariant modulo $=_\beta$
- Q finite $\implies$ every $[\![A]\!]_Q$ finite

Syntactically vs semantically regular languages of λ-terms

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular*
when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

For languages of type $A = \mathtt{Str}_\Sigma$:

usual reg. $\Rightarrow$ **synt. reg.** programming exercise

synt. reg. $\Rightarrow$ **sem. reg.** take $Q = [\![B]\!]_{\{0,1\}}$ ...

sem. reg. $\Rightarrow$ **usual** for $w \in \Sigma^*$, compute $[\![\overline{w}]\!]$ by DFA

Theorem (Moreau & N., CSL'24)

$\forall A$, *syntactically regular* $\iff$ *semantically regular*

($\Rightarrow$) same proof as before!
($\Leftarrow$) "represent" elements of $[\![A]\!]_Q$ by λ-terms of type $A\{o := (o^{|Q|} \to o)\}$ using *logical relations*

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Naive semantics of simply typed λ-terms:
$t : A \implies [\![t]\!]_Q \in [\![A]\!]_Q$ where

$$[\![o]\!]_Q = Q \text{ (an arbitrary set)}$$

$$[\![A \to B]\!]_Q = [\![A]\!]_Q \to [\![B]\!]_Q = [\![B]\!]_Q^{[\![A]\!]_Q}$$

- *compositional* by def., e.g. $[\![t\,u]\!]_Q = [\![t]\!]_Q\,([\![u]\!]_Q)$ + invariant modulo $=_\beta$
- Q finite $\implies$ every $[\![A]\!]_Q$ finite

Syntactically vs semantically regular languages of λ-terms (Thanks for your attention!)

Rephrasing of [Hillebrand & Kanellakis 1996]
regular languages over Σ (usual sense) $\cong$
syntactically regular lang. of λ-terms of type $\mathtt{Str}_\Sigma$

Definition

$L \subseteq \{u \mid u : A\}/(=_\beta)$ is *syntactically regular* when it is defined by some $t : A\{o := B\} \to \mathtt{Bool}$

For languages of type $A = \mathtt{Str}_\Sigma$:

usual reg. $\Rightarrow$ **synt. reg.** programming exercise

synt. reg. $\Rightarrow$ **sem. reg.** take $Q = [\![B]\!]_{\{0,1\}}$ ...

sem. reg. $\Rightarrow$ **usual** for $w \in \Sigma^*$, compute $[\![\overline{w}]\!]$ by DFA

Theorem (Moreau & N., CSL'24)

$\forall A$, *syntactically regular* $\iff$ *semantically regular*

($\Rightarrow$) same proof as before!
($\Leftarrow$) "represent" elements of $[\![A]\!]_Q$ by λ-terms of type $A\{o := (o^{|Q|} \to o)\}$ using *logical relations*

Algebraic recognition of regular languages:

$$\Sigma^* \xrightarrow{\text{morphism}} \text{finite monoid} \to \{\text{yes}, \text{no}\}$$

Definition (Salvati 2009)

Semantically regular language of λ-terms:

$$\{u \mid u : A\}/(=_\beta) \xrightarrow{[\![-]\!]_Q} [\![A]\!]_Q \to \{\text{yes}, \text{no}\}$$

Naive semantics of simply typed λ-terms:
$t : A \implies [\![t]\!]_Q \in [\![A]\!]_Q$ where

$$[\![o]\!]_Q = Q \text{ (an arbitrary set)}$$

$$[\![A \to B]\!]_Q = [\![A]\!]_Q \to [\![B]\!]_Q = [\![B]\!]_Q^{[\![A]\!]_Q}$$

- *compositional* by def., e.g. $[\![t\,u]\!]_Q = [\![t]\!]_Q\,([\![u]\!]_Q)$ + invariant modulo $=_\beta$
- Q finite $\implies$ every $[\![A]\!]_Q$ finite