Finitary semantics and regular languages of $\lambda$-terms

Vincent Moreau, joint work with Tito Nguyễn

ASV day 2023
December the 6th, 2023

IRIF, Université Paris Cité, Inria Paris
Let $\Sigma = \{a, b\}$ be a two-letter alphabet. We write the word $aab \in \Sigma^*$ as

$$a, b \vdash aab.$$ 

Words can be encodes as finite ranked trees: each letter of $\Sigma$ has arity 1, and we add a constant of arity 0.

We write the ranked tree associated to $aab$ as

$$a : 1, b : 1, c : 0 \vdash b(a(a(c))).$$
Finite trees

Let $\Sigma$ be the ranked alphabet $\{a : 2, b : 1, c : 0\}$. Then, the tree

```
               a
              / \  \
             a   b
            /   /  \
           c   b   b
          /   /   /
         c   c   c
```

is represented by the judgment

$$a : 2, b : 1, c : 0 \vdash a(a(c, b(c)), b(b(c))) .$$
Adding simple types

The ranked alphabet \( \{a : 2, b : 1, c : 0\} \) can be seen as the typed context

\[
a : \varnothing^2 \Rightarrow \varnothing, \quad b : \varnothing \Rightarrow \varnothing, \quad c : \varnothing.
\]

With that in mind, the tree

\[
\begin{array}{c}
  a \\
  \quad a \\
  \quad \quad a \\
  \quad \quad \quad b \\
  \quad \quad \quad \quad c \\
  \quad \quad \quad \quad \quad b \\
  \quad \quad \quad \quad \quad \quad c \\
  \quad \quad \quad \quad \quad \quad \quad c \\
\end{array}
\]

is represented by the typed judgment

\[
a : \varnothing^2 \Rightarrow \varnothing, \quad b : \varnothing \Rightarrow \varnothing, \quad c : \varnothing \vdash a(a(c, b(c)), b(b(c))) : \varnothing.
\]

→ The simply typed \( \lambda \)-calculus generalizes finite ranked trees.
The $\lambda$-abstraction

Consider the typed context

$$\rightarrow : \varnothing^2 \Rightarrow \varnothing, \ \forall : (\varnothing \Rightarrow \varnothing) \Rightarrow \varnothing, \ \bot : \varnothing.$$ 

The $\forall$ takes a function as an argument. They can be built using the $\lambda$-abstraction.

The following $\lambda$-terms

$$\forall (\lambda(\varphi : \varnothing). \varphi \rightarrow \varphi)$$
$$\forall (\lambda(\varphi : \varnothing). ((\varphi \rightarrow \bot) \rightarrow \bot) \rightarrow \varphi)$$
$$\forall (\lambda(\varphi : \varnothing). \forall (\lambda(\psi : \varnothing). ((\varphi \rightarrow \psi) \rightarrow \varphi) \rightarrow \varphi)$$

are all of type $\varnothing$ in the fixed context.
Closing the $\lambda$-terms

The $\lambda$-abstraction moves variables from the context into the $\lambda$-term:

$$a : A, b : B, c : C, d : D \vdash t : T$$

\[ \downarrow \]

$$a : A, b : B, c : C \vdash \lambda(d : D). t : D \Rightarrow T.$$ 

Repeating this on trees, we obtain closed $\lambda$-terms like

$$a : \circ^2 \Rightarrow \circ, b : \circ \Rightarrow \circ, c : \circ \vdash a(a(c, b(c)), b(b(c))) : \circ$$

\[ \downarrow \downarrow \downarrow \]

$$\vdash \lambda(a : \circ^2 \Rightarrow \circ). \lambda(b : \circ \Rightarrow \circ). \lambda(c : \circ). a(a(c, b(c)), b(b(c))) : \text{Church}_{2,1,0}$$

with no free variable and which are of type

$$\text{Church}_{2,1,0} \ := \ (\circ^2 \Rightarrow \circ) \Rightarrow (\circ \Rightarrow \circ) \Rightarrow \circ \Rightarrow \circ.$$
Two notions of recognition

finite words

finite trees

\( \lambda \)-terms

semantical recognition

syntactic recognition

They are the same languages.
Language of $\lambda$-terms: the semantic side

If $Q$ is a finite set, then any tree $t : \text{Church}_{[2,1,0]}$ can be interpreted as

$$[t]_Q \in (Q^2 \to Q) \Rightarrow (Q \Rightarrow Q) \Rightarrow Q \Rightarrow Q.$$ 

For all $\delta_a : Q^2 \to Q$, $\delta_b : Q \to Q$, $\delta_c \in Q$, we have

$$[\lambda a. \lambda b. \lambda c. a(a(c, b(c)), b(b(c)))]_Q(\delta_a, \delta_b, \delta_c) = \delta_a(\delta_a(\delta_c, \delta_b(\delta_c)), \delta_b(\delta_b(\delta_c))).$$

$\Rightarrow$ Interpreting the $\lambda$-calculus in finite sets specializes to runs in deterministic finite bottom-up tree automata.
Semantic languages of $\lambda$-terms

In the case of words, any homomorphism $\varphi : \Sigma^* \to M$ into a finite monoid, together with a subset $F \subseteq M$, induces the regular language of finite words

$$L_F := \{ w \in \Sigma^* \mid \varphi(w) \in F \}.$$ 

The notion of regular language of $\lambda$-terms has been introduced by Salvati. Let $A$ be a simple type. For any finite set $Q$ and any subset $F \subseteq \llbracket A \rrbracket_Q$, we define

$$L_F := \{ t : A \mid \llbracket t \rrbracket_c \in F \}.$$ 

**Definition.** A language of $\lambda$-terms is **semantically recognizable** if it is of the form $L_F$. 
Quantified boolean formulas

We have seen that \( \lambda \)-terms of the type

\[
\text{QBF} \quad := \quad \mathcal{O}^2 \Rightarrow \mathcal{O} \Rightarrow (\mathcal{O} \Rightarrow \mathcal{O}) \Rightarrow \mathcal{O} \Rightarrow \mathcal{O} \Rightarrow \mathcal{O} \Rightarrow \mathcal{O}
\]

By taking \( Q = \{0, 1\} \) with the transitions

\[
\delta_\rightarrow \in Q^2 \Rightarrow Q \quad \quad \delta_\forall \in (Q \Rightarrow Q) \Rightarrow Q \quad \quad \delta_\bot \in Q,
\]

for any \( \lambda \)-term \( t : \text{QBF} \), we have

\[
t \text{ represents a true formula} \quad \iff \quad \llbracket t \rrbracket_Q(\delta_\rightarrow, \delta_\forall, \delta_\bot) = 1.
\]

which shows that true formula form a semantically recognizable language.
Language of $\lambda$-terms: the syntactic side

We consider the type

$$\text{Bool} := \o \Rightarrow \o \Rightarrow \o$$

whose only inhabitants, up to $\beta\eta$-conversion, are

$$\text{true} := \lambda(a : \o).\lambda(b : \o). a \quad \text{and} \quad \text{false} := \lambda(a : \o).\lambda(b : \o). b .$$

An automaton can be encoded as a closed $\lambda$-term of type

$$r : ((B^2 \Rightarrow B) \Rightarrow (B \Rightarrow B) \Rightarrow B \Rightarrow B) \Rightarrow \text{Bool}$$

for some simple type $B$ representing the set of finite states.

$\rightarrow$ Regular languages can be recovered syntactically from the $\lambda$-calculus.
Syntactic languages of $\lambda$-terms

If $A$ is a simple type, then $A[B]$ is the substitution of $B$ for $\circ$ in $A$.

If $t$ is $\lambda$-term of type $A$, then $t[B]$ is a $\lambda$-term of type $t[A]$, defined inductively as

$$
(\lambda (d : D). t)[B] = \lambda (d : D[B]). t[B] \quad (t \ u)[B] = t[B] \ u[B] \quad d[B] = d
$$

For any simple type $B$, any $\lambda$-term $r : A[B] \Rightarrow \text{Bool}$ induces a language

$$
L_r := \{t : A \mid r \ t[B] =_{\beta\eta} \text{true}\}.
$$

**Definition.** A language of $\lambda$-terms is **syntactically regular** if it is of the form $L_r$. 
Our theorem

It was known that, at type Church\(_\Sigma\) and \(L\) a language of \(\lambda\)-terms of type Church\(_\Sigma\),

\[
L \text{ is semantically recognizable } \iff L \text{ is syntactically regular}
\]

which is also equivalent to being regular in the usual sense.

**Theorem (M., Nguyen).** For every type \(A\) and \(L\) a language of \(\lambda\)-terms of type \(A\),

\[
L \text{ is semantically recognizable } \iff L \text{ is syntactically regular.}
\]

Actually stronger: still hold when replacing finite sets by other sufficiently well-behaved finitary model of the \(\lambda\)-calculus! For example:

- deterministic models (finite sets and functions),
- nondeterministic models (finite Scott domains).
Conclusion

Future work:

- Finitary intensional models of the simply typed $\lambda$-calculus, e.g. sequential algorithms, some qualitative models of linear logic.
- Study different calculi, e.g. linear, polymorphic, with effects.
- Study which languages of formulas are regular languages of $\lambda$-terms.
Conclusion

Future work:

- Finitary intensional models of the simply typed $\lambda$-calculus, e.g. sequential algorithms, some qualitative models of linear logic.
- Study different calculi, e.g. linear, polymorphic, with effects.
- Study which languages of formulas are regular languages of $\lambda$-terms.

Thank you for your attention!

Any questions?
