Towards a
Mechanized Theory of Computation for Education

Tiago Cogumbreiro and Yannick Forster

20–23 June, 2022
University of Nantes, TYPES22
Motivation

- **Context:** Formal languages and automata (FLA) for undergraduate course
- **Goal:** help with math anxiety; rethink FLA to computer scientists

Contributions

- A simple and expressive calculus for decidability/computability
- Used in 3 editions of a course on FLA at UMass Boston
- Formalized multiple textbook results (e.g., halting problem)

Our technique

- Assumptions
- Base calculus
- Results
Math anxiety & Proofs

As a student:

- How do I know which theorems are available to use in a proof?
- How do I know if my steps are correct?
- How can I get more details about a particular proof?
- How can I study autonomously?

Context

- Undergraduate students (3rd, 4th year)
- Compulsory course on FLA/computability/decidability
- No experience with proof assistants
Proof assistants in education

- *(Math anxiety)* Interactive mechanism allows students to step through a proof autonomously (independent study)
- *(Math anxiety)* Proof assistant turns a logic assignment into a programming assignment (great for computer science students)
- *(Courseware)* Machine checked proof scripts help automate grading
UMB-SVL Turing

- Open source software (MIT License)
- Regular languages results (eg, pumping lemma)
- Decidability, undecidability, recognizability results

https://gitlab.com/umb-svl/turing/
Mechanization goals

Develop supplementary material to Michael Sipser's *Introduction to the theory of computation*

- Coq formalism should be **similar to the textbook**
- **Simple proofs and techniques**, expect rudimentary knowledge of Coq (case analysis, induction, polymorphism, and logical connectives).
- **Include alternative proofs**, when there is pedagogical benefit
Use case: Theorem 4.11 $A_{TM}$ is undecidable

AN UNDECIDABLE LANGUAGE

Now we are ready to prove Theorem 4.11, the undecidability of the language

$$A_{TM} = \{ \langle M, w \rangle | M \text{ is a TM and } M \text{ accepts } w \}.$$  

PROOF  We assume that $A_{TM}$ is decidable and obtain a contradiction. Suppose that $H$ is a decider for $A_{TM}$. On input $\langle M, w \rangle$, where $M$ is a TM and $w$ is a string, $H$ halts and accepts if $M$ accepts $w$. Furthermore, $H$ halts and rejects if $M$ fails to accept $w$. In other words, we assume that $H$ is a TM, where

$$H(\langle M, w \rangle) = \begin{cases} 
\text{accept} & \text{if } M \text{ accepts } w \\
\text{reject} & \text{if } M \text{ does not accept } w.
\end{cases}$$  

Now we construct a new Turing machine $D$ with $H$ as a subroutine. This new TM calls $H$ to determine what $M$ does when the input to $M$ is its own description $\langle M \rangle$. Once $D$ has determined this information, it does the opposite. That is, it rejects if $M$ accepts and accepts if $M$ does not accept. The following is a description of $D$.

$$D = \text{"On input } \langle M \rangle, \text{ where } M \text{ is a TM:}$$

1. Run $H$ on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what $H$ outputs. That is, if $H$ accepts, reject; and if $H$ rejects, accept."
Formalizing the language

A language is input → Prop, a function from an input to a proposition.

- Run : prog → bool → Prop runs program (our calculus) and returns acceptance upon termination
- decode_mach_input deconstruct an input as a pair M (Turing machine) and input w
- For any function f:input → prog there exist a machine M that computes f (Axiom)

\[ A_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w \} \]

Definition \( A_{tm} : \text{input} \rightarrow \text{Prop} := \)
\[ \begin{align*}
\text{fun } i \Rightarrow \\
\text{let } (M, w) := \text{decode_mach_input } i \text{ in} \\
\text{Run (Call M w) } \text{true}.
\end{align*} \]
Formalizing “high-level descriptions”

A calculus to invoke and compose abstract Turing machines

\[ D = \text{“On input } \langle M \rangle \text{, where } M \text{ is a TM:} \]
\[ \begin{align*}
1. & \text{ Run } H \text{ on input } \langle M, \langle M \rangle \rangle. \\
2. & \text{ Output the opposite of what } H \text{ outputs. That is, if } H \text{ accepts, reject; and if } H \text{ rejects, accept.”}
\end{align*} \]

**Definition** \( D : \text{input} \to \text{prog} : \text{input} \to \text{prog} := \)

\[
\begin{aligned}
\text{fun} \ (w: \text{input}) & \Rightarrow (\ast w = \langle M \rangle \text{ and decode_mach } w = M \ast) \\
\text{mlet } b & \leftarrow H \langle \text{decode_mach } w, w \rangle \text{ in } (\ast \text{Run } H \text{ on input } \langle M, \langle M \rangle \rangle \ast) \\
\text{if } b & \text{ then Ret false } (\ast \text{If } H \text{ accepts, reject } \ast) \\
\text{else } & \text{ Ret true } (\ast \text{If } H \text{ rejects, accept } \ast)
\end{aligned}
\]
Syntax

\[ p ::= \text{mlet } x = p \text{ in } p \mid \text{call } M \ i \mid \text{return } b \quad \text{where } b \in \{\top, \bot\} \]

Semantics

\[
\begin{align*}
\text{return } b \Downarrow b \\
\text{call } M \ i \Downarrow \top \\
\text{M rejects } i \\
\text{call } M \ i \Downarrow \bot \\
p \Downarrow b \quad p'[x := b] \Downarrow b' \\
m\text{let } x = p \text{ in } p' \Downarrow b'
\end{align*}
\]

We embed Coq functions in our High-level descriptions
Results

- $A_{TM}, \overline{HALT_{TM}} = \{\langle M \rangle \mid M \text{ halts}\}$, and $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid \forall i, M_1 \text{ accepts } i \iff M_2 \text{ accepts } i\}$ are undecidable
- $A$ is decidable iff $A$ is recognizable and co-recognizable
- $EQ_{TM}$ is neither recognizable nor co-recognizable
- Rice’s Theorem (proved by Kleopatra Gjini, \textit{undergrad research project})
- Results include direct proofs and proofs using map-reducibility
Future Work

- **Consistency of axioms**: instantiate our theory with one of the models of the *Coq library of undecidable problems* [CoqPL'20]
- **Report on education insights**: How to teach effectively with Proof assistants
  - step-by-step evaluation in proofs (understanding `simpl`)
  - induction principles and recursive types
  - brute forcing solutions
Assumptions

- Theory parameterized by input type, Turing machine type, and Turing machine \textit{deterministic} semantics
- Any Turing machine either accepts, rejects, or neither (e.g., loops)
- For any Turing function $f$ there exists a machine $M$ that computes $f$ (definable Coq functions are computable, Church's Thesis)