Towards a Mechanized Theory of Computation for Education

Tiago Cogumbreiro[1] and Yannick Forster[2]

[1] University of Massachussets Boston, Boston, Massachussets, U.S.A.
tiago.cogumbreiro@umb.edu
[2] Inria, France
yannick.forster@inria.fr

Abstract

In this talk we present a mechanization effort of theory of computation in the context of undergraduate education, with a focus on decidability and computability. We introduce a Coq library used to teach 3 sessions of a course on Formal Languages and Automata, at the University of Massachusetts Boston. Our project includes full proofs of results from a textbook, such as the undecidability of the halting problem and Rice's theorem. To this end, we present a simple and expressive calculus that allows us to capture the essence of informal proofs of classic theorems in a mechanized setting. We discuss the assumptions of our formalism and discuss our progress in showing the consistency of our theory.

Introduction. Formal languages and automata (FLA) is in the basis of the curriculum of undergraduate computer science [10]. We report on an open source project written in Coq [3] to mechanize results of classical theory of computation. The first author used this software for 3 semesters to teach decidability, computability, and regular languages at the University of Massachusetts Boston. Proof assistants play a central role in our lectures for three reasons. Firstly, a proof assistant offers an interactive mechanism to allow students to step through a proof autonomously, allowing students to independently browse every detail of a proof at their own pace. Secondly, a proof assistant turns a logic assignment into a programming assignment, which can be more approachable to computer science students. Thirdly, having proof scripts that can be machine checked, lets instructors automatically grade homework assignments. Other works that use proof assistants to aid education include [1, 9, 12].

Mechanization goals. We formalize Sipser's *Introduction to the theory of computation* [16] in Coq. Our design goal is to keep our formalism as close to the textbook as possible, which includes having mechanized proofs that mirror the textbook proofs. Another important design goal is that proofs should only include basic Coq capabilities. The proofs need to be comprehensible to an undergraduate student with rudimentary knowledge of Coq (case analysis, induction, polymorphism, and logical connectives). Further, we include alternative proofs of some theorems when there's a pedagogical benefit, *e.g.*, the proof is simpler, or the intuition is easier to explain. Our approach contrasts many published works on mechanized computability theory [13, 17, 8, 2, 15, 4, 11].

Decidability results. Our mechanization includes the main results of [16, Chapters 4 and 5], on decidability and reducibility. One of our contributions is formalizing Sipser's "high-level descriptions," which is essentially pseudo-code to describe a Turing machine. For instance, consider Theorem 4.11 of [16, Chapter 4], where TM denotes a Turing machine, ranged over by meta-variable M, inputs are ranged over by i. A language, ranged over by A is a set of inputs. Language A is decidable if there exists a Turing machine M that decides A, *i.e.*, M accepts i if, and only if, $i \in A$; and M rejects i if, and only if, $i \notin A$. Additionally, $\langle \cdot \rangle$ denotes a reasonable encoding of one or more objects into a string, *e.g.*, $\langle M, i \rangle$ encodes a Turing machine M and an input i into an input, and $\langle M \rangle$ encodes a Turing machine M into an input.

Theorem 4.11 ([16, pp. 207]). *$A_{TM} = \{\langle M, i\rangle \mid M$ is a TM and M accepts $i\}$ is undecidable.*

The proof of Theorem 4.11 includes the following high-level description of a Turing machine D, parameterized by a Turing machine H which decides A_{TM}: *"The following is the description of D: (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."*

We formalize such high-level description as:

```
Definition D (H:input → prog): input → prog :=
  fun (i:input) ⇒                      (* On input i = <M> *)
    mlet b ← H <[ decode_mach i, i ]> in (* Step 1 *)
    if b then Ret false else Ret true    (* Step 2 *)
```

where input $\texttt{i} = \langle M\rangle$ and $\texttt{decode\_mach i} = M$. We formalize high-level descriptions next. We first give the syntax of high-level descriptions p.

$$p \ ::= \ \texttt{mlet}\ x = p\ \texttt{in}\ p \ \mid\ \texttt{call}\ M\ i \ \mid\ \texttt{return}\ b \qquad \text{where } b \in \{\top, \bot\}$$

Next, we introduce a big-step operational semantics in terms of high-level descriptions:

$$\frac{}{\texttt{return}\ b \Downarrow b} \qquad \frac{M \text{ accepts } i}{\texttt{call}\ M\ i \Downarrow \top} \qquad \frac{M \text{ rejects } i}{\texttt{call}\ M\ i \Downarrow \bot} \qquad \frac{p \Downarrow b \qquad p'[x := b] \Downarrow b'}{\texttt{mlet}\ x = p\ \texttt{in}\ p' \Downarrow b'}$$

We then define that a Turing machine M computes a Turing function f of type $\texttt{input} \to \texttt{prog}$ if for any input i we have $\texttt{call}\ M\ i \Downarrow b$ if and only if $f(i) \Downarrow b$.

Assumptions. Our theory is parameterized by an input type, a type of Turing machines, and the semantics of Turing machines. We assume that their execution is deterministic and that for any machine M and an input i we can obtain M accepts i, or M rejects i, or neither. Centrally, we assume that for any Turing function f there exists a Turing machine M computing f. This assumption is consistent since we work in Coq, where every definable function is computable.

Results. To mechanize the proof of Theorem 4.11 we assume a machine M' deciding A_{TM}, *i.e.*, computing a Turing function H. Now using the assumption that every Turing function is computable on $\texttt{D}(H)$ yields M such that $\texttt{call}\ M\ i \Downarrow \texttt{true} \leftrightarrow \texttt{call}\ M\ i \Downarrow \texttt{false}$, a contradiction.

Our further main results include: A is decidable if, and only if A is recognizable and co-recognizable (*i.e.*, its complement is recognizable) (Theorem 4.22); the complement of A_{TM} is not recognizable (Corollary 4.23); $HALT_{TM} = \{\langle M, i\rangle \mid M \text{ accepts or rejects } i\}$ is undecidable (Theorem 5.1); $E_{TM} = \{\langle M\rangle \mid L(M) = \emptyset\}$ is undecidable (Theorem 5.2), where $L(M) = \{i \mid M \text{ accepts } i\}$; $EQ_{TM} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ is neither recognizable nor co-recognizable (Theorem 5.30); EQ_{TM} is undecidable (Theorem 5.4); Rice's Theorem (Problem 5.28). Our proofs of these results are all constructive. Our project contains results on languages and regular languages, *e.g.*, the pumping lemma for regular language inspired by the proof in [14].

Future work. To show the consistency of our axioms, we are working on instantiating our theory with a mechanized formalism of computability from the Coq library of undecidability proofs [7] equivalent to Turing machines [6]. Consistency of the central assumptions then follows from the consistency of the axiom CT in type theory [5]. We are also investigating multiple grading approaches in classes that use proof assistants, *e.g.*, multiple-choice questions, automatic questions about students submissions.

References

[1] Grzegorz Bancerek, Czesław Byliński, Adam Grabowski, Artur Korniłowicz, Roman Matuszewski, Adam Naumowicz, Karol PaźK, and Josef Urban. Mizar: State-of-the-art and beyond. In *CICM*, volume 9150, page 261–279. Springer, 2015.

[2] Mario Carneiro. Formalizing Computability Theory via Partial Recursive Functions. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:17, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[3] Tiago Cogumbreiro. Turing: formalization of introduction to the theory of computation, 2022. https://gitlab.com/umb-svl/turing.

[4] Yannick Forster. Parametric church's thesis: Synthetic computability without choice. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science - International Symposium, LFCS 2022, Deerfield Beach, FL, USA, January 10-13, 2022, Proceedings*, volume 13137 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2022.

[5] Yannick Forster. Parametric Church's Thesis: Synthetic computability without choice. In Sergei N. Artëmov and Anil Nerode, editors, *Logical Foundations of Computer Science - International Symposium, LFCS 2022, Deerfield Beach, FL, USA, January 10-13, 2022, Proceedings*, volume 13137 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2022.

[6] Yannick Forster, Fabian Kunze, and Maximilian Wuttke. Verified programming of Turing machines in Coq. In *CPP*, page 114–128. ACM, 2020.

[7] Yannick Forster, Dominique Larchey-Wendling, Andrej Dudenhefner, Edith Heiter, Dominik Kirst, Fabian Kunze, Gert Smolka, Simon Spies, Dominik Wehr, and Maximilian Wuttke. A Coq library of undecidable problems. In *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020).*, 2020.

[8] Yannick Forster and Gert Smolka. Weak call-by-value lambda calculus as a model of computation in Coq. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving - 8th International Conference, ITP 2017, Brasília, Brazil, September 26-29, 2017, Proceedings*, volume 10499 of *Lecture Notes in Computer Science*, pages 189–206. Springer, 2017.

[9] Asta Halkjær From, Alexander Birch Jensen, Anders Schlichtkrull, and Jørgen Villadsen. Teaching a formalized logical calculus. In *ThEdu*, volume 313 of *EPTCS*, pages 73–92, 2019.

[10] Association for Computing Machinery (ACM) Joint Task Force on Computing Curricula and IEEE Computer Society. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM, 2013.

[11] J Strother Moore. Towards a mechanically checked theory of computation. In *Logic-Based Artificial Intelligence*, pages 547–574. Springer US, 2000.

[12] Tobias Nipkow. Teaching semantics with a proof assistant: No more LSD trip proofs. In Viktor Kuncak and Andrey Rybalchenko, editors, *VMCAI*, volume 7148 of *LNCS*, pages 24–38. Springer, 2012.

[13] Michael Norrish. Mechanised computability theory. In *ITP 2011*, volume 6898 of *LNCS*, pages 297–311. Springer, 2011.

[14] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Software Foundations.* Electronic textbook, 2021. Version 6.1. https://softwarefoundations.cis.upenn.edu/lf-current/.

[15] Thiago Mendonça Ferreira Ramos, Ariane Alves Almeida, and Mauricio Ayala-Rincón. Formalization of the computational theory of a turing complete functional language model. *Journal of Automated Reasoning*, January 2022.

[16] Michael Sipser. *Introduction to the theory of computation.* Cengage Learning, 2012.

[17] Jian Xu, Xingyuan Zhang, and Christian Urban. Mechanising Turing machines and computability theory in Isabelle/HOL. In *International Conference on Interactive Theorem Proving*, pages 147–162. Springer, 2013.