Towards a Mechanized Theory of Computation for Education

Tiago Cogumbreiro

July 4, 2022

Royal Holloway, University of London

Overview

- Motivation
- A theory of decidability and regular languages
- Proof assistants in education
- Visualizing automata and computations

UMass Boston

Motivation

CS 420

Introduction to the Theory of Computation

University of Massachusetts Boston

CS 420

Context

- On formal languages, automata, computability, decidability
- Compulsory course
- Undergraduate students (3rd, 4th year)
- Students have **no experience** with proof assistants
- Cohorts of around 40 students

UMass Boston

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?

UMass Boston

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?

Rethink Theory of Computation for computer scientists!

UMass Boston

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 with:
  - automate grading
  - remote learning

UMass Boston

Related work

- Proof assistants in education: Software Foundations series, Xena, [ThEdu'20], [VMCAI'12]
- Computability results: [LBAI'00], [ITP'11], [ITP'13],[JAR'13],[ITP'17],[ITP'19],[LFCS'22]

Summary

- Proof assistants for education is being used more on programming language semantics, and preparatory mathematics courses
- Mechanization of computability is not applied to education

What about using proof assistants to teach computability?

UMass Boston

CS 420

Syllabus

- 3.5 weeks of Coq principles
  (🐓 induction, recursive data structures, polymorphism)
- 3.5 weeks of Regular languages
  (🐓 formal language connectives, regular expressions)
- 3.5 weeks on Context-free Languages
  (🐓 grammars, pumping lemma and non-regular lang)
- 3.5 weeks on Undecidability/map reducibility
  (🐓 decidability proofs, map-reducibility, undecidability)

UMass Boston

The rest of this talk...

- A theory of decidability and regular languages
- Proof assistants in education
- Visualizing automata and computations

UMass Boston

A theory of decidability and regular languages

TYPES'22

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.

UMB-SVL Turing

- Open source software (MIT License)
- Regular languages results (eg, pumping lemma)
- Decidability, undecidability, recognizability results
- **Joint work with:** Yannick Forster, Inria

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

UMass Boston

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

UMass Boston

Decidability results

A language to compose (abstract) Turing machines

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

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

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

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

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 = "On input $\langle M\rangle$, where M 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*."

UMass Boston

Formalizing a language

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

```
Definition A_tm : input → Prop :=
  fun i ⇒
    let (M, w) := decode_mach_input i in
    Run (Call M w) true.
```

Language: a function from an input to a proposition, `input → Prop`.

Assumptions

- `Run : prog → bool → Prop` runs `prog`ram (our calculus) and returns acceptance upon termination
- `decode_match_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**)

UMass Boston

Formalizing "high-level descriptions"

A calculus to invoke and compose abstract Turing machines

$D =$ "On input $\langle M \rangle$, where M 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*."

```
Definition D (H:input → prog) : input → prog :=
  fun (w:input) ⇒ (* w = <M> and decode_mach w = M *)
    mlet b ← H <[decode_mach w, w]> in (* Run H on input <M, <M>> *)
    if b then Ret false  (* If H accepts, reject *)
         else Ret true   (* If H rejects, accept *)
  .
```

UMass Boston

Syntax

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

Semantics

$$\frac{}{\texttt{return}\ b \Downarrow b} \qquad \frac{M \text{ accepts } i}{\texttt{call}\ M\ i \Downarrow \top}$$

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

We embed Coq functions in our High-level descriptions

UMass Boston

Results

- $A_{TM}, HALT_{TM} = \{\langle M \rangle \mid M \text{ halts}\}$, 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, *undergrad research project*)
- Results include direct proofs and proofs using map-reducibility

Chapter 1: `Regular.v`

- [x] Theorem 1.70: Pumping lemma for regular languages
- [x] Example 1.73: $\{0^n1^n \mid n \geq 0\}$ is not regular

Chapter 4: `LangDec.v`

- [x] Theorem 4.11: A_{TM} is undecidable.
- [x] Corollary 4.18: Some languages are not recognizable.
- [x] Theorem 4.22: L is decidable iff L is recognizable and co-recognizable
- [x] Theorem 4.23: $\overline{A}_{\text{TM}}$ is not recognizable.

Chapter 5: `LangRed.v`

- [x] Theorem 5.1: $HALT_{\text{TM}}$ is undecidable.
- [x] Theorem 5.2: E_{TM} is undecidable.
- [x] Theorem 5.4: EQ_{TM} is undedicable.
- [x] Theorem 5.22: If $A \leq_{\text{m}} B$ and A decidable, then B decidable.
- [x] Theorem 5.28: If $A \leq_{\text{m}} B$ and A recognizable, then B recognizable.
- [x] Corollary 5.29: If $A \leq_{\text{m}} B$ and B is undecidable, then A is undecidable.
- [x] Corollary 5.30: EQ_{TM} unrecognizable and co-unrecognizable.

UMass Boston

Proof assistants in education

Challenges, opportunities, and insights

On proof proficiency

- Students master proof assistant quickly
  - **Few proof tactics** help students master majority of the course
- Students can prove harder theorems with proof assistants
  - As long as they know a statement is provable, it becomes just a matter of time
  - They have (proof) context to better understand where they are stuck
  - They can query the validity of their progress

UMass Boston

On students brute-forcing solutions

- Students may have mastered the "mechanics" of proving
- Exercise may be too simple
- Higher-level proofs (eg, decidability) are harder to brute force

UMass Boston

On understanding the mechanics, but missing the idea

- Instead of "prove P", ask "prove either P, or prove not P"
- Test students in writing about the lemmas proved
- Assignment idea: formalize a paper (harder to check automatically)

UMass Boston

On low solution variability

- Example: solution is difficult, but consists of 3 lines of code
- Harder to identify plagiarism
- Ask students to comment their code
  - Students may "translate tactics to English"
- Consider exercises that demand longer solutions
- Make the style count
- Keep a submission history (submission site versions, or git history)
  - Students can still choose to submit a single version

UMass Boston

On struggling with magic

- Focus on evaluation of terms early on
  Test students on why a term evaluates to another term with `simpl`
- Consider simplifying the nomenclature
  - Students struggle to keep track of all the verbs, adjectives, and nouns
    (eg, accepts, decides, recognizes, decidable, recognizable, decider, recognizer, co-, etc)
  - Names are non-intuitive
    (a decider? a program accepts? language of a program?)
  - How to relate to textbooks?
- Focus on recursive data-types and induction schemes early on
  Test students on deriving the induction scheme.
- Avoid Coq notations and Unicode
- Avoid Coq coercions
- Decidability proofs more trivial than ones in regular languages
  (consider going from Turing machines down to NFAs)

UMass Boston

Distributing Coq assignments

- Favor assignments that only depend on the standard-library
  - fail an assignment *because* of installation problems
- Chocolatey works really well in Windows
  - If you rely on ASCII art, remind students to enable Mono-spaced fonts
  - CI for Windows (eg, AppVeyour) works really well
- Setup in macOS is the most challenging:
  - students install multiple versions of Coq (homebrew, web site installation)
  - compilation environment unavailable (inside AppImage),
  - difficult to distribute binaries
  - CoqIDE does not integrate well with the OS (slow, visual glitches)
- CoqJS might be a useful alternative
- Automated proof checking is trivial

UMass Boston

Visualizing automata and computations

Gidayu: Visualizing Automaton and Their Computations

Tiago Cogumbreiro
tiago.cogumbreiro@umb.edu
University of Massachusetts Boston
Boston, Massachusetts, USA

Gregory Blike
gregory.blike001@umb.edu
University of Massachusetts Boston
Boston, Massachusetts, USA

ABSTRACT

Generating visualizations of Formal Languages and Automata (FLA) is often a laborious and error prone task. Existing tools either offer the ability to fully customize the appearance of the artifacts, or offer reusability and abstraction, but do not offer both at the same time. In this paper, we introduce a system called Gidayu for creating mathematical diagrams of automata, of their computations, and of their transformations. Many kinds of automata are supported: (non)deterministic finite automata, generalized nondeterministic finite automata, and (non)-deterministic pushdown automata. Gidayu fosters experimentation and rapid prototyping, as diagrams are generated automatically. The specification language includes directives to fine tune the presentation of each element; and, users can customize the visual notation used by our system. We discuss various parameters Gidayu offers to visualize diagrams and their importance in the instruction of FLA.

notation, so reusing such artifacts in an educational context can be challenging. The visual notation cannot be adapted to match the artifacts of a textbook or of the instructor's slides, which can introduce confusion for students and inconsistencies in production. Educators need to devote time to clarify the different conventions found in the artifacts produced by interactive editors.

This paper introduces the first tool to visualize the computation of automata. The state of the art has limited support to explain the computation of an automaton, rendering the execution as a series of tables. Computation diagrams, *e.g.*, [16, pp. 50], visualize all reachable runtime states (configurations) of an automaton for a particular input. Computation diagrams are useful pedagogical device to understand automata: students can visualize in one diagram the various possible configurations and the relationships among them. In such diagrams, nondeterminism is visually evident as a form of branching, and accepting an input can be explained as a search algorithm (*i.e* reaching an accepting configuration). Furthermore

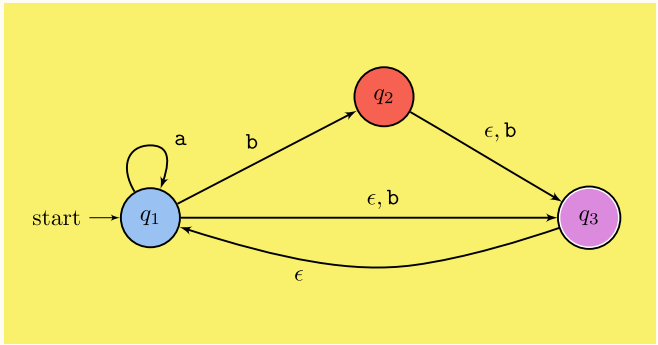
Gidayu

- Diagrams as a powerful education device to teach automata theory
- Domain-specific language to specify automata

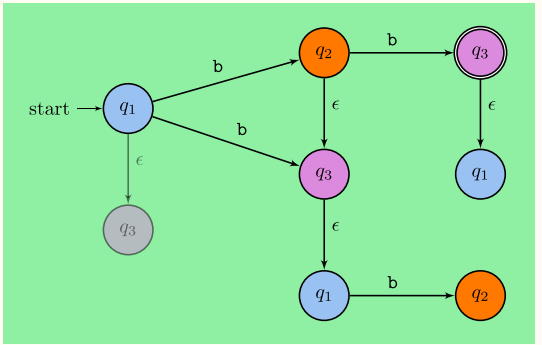
nfa spec

```
type: nfa
states:
  q1: {label: q_1, initial: true}
  q2: {label: q_2}
  q3: {label: q_3, final: true}
transitions:
- {src: q1, actions: [b], dst: q2}
- {src: q1, actions: [a], dst: q1}
- {src: q1, actions: [null,b], dst: q3}
- {src: q2, actions: [null,b], dst: q3}
- {src: q3, actions: [null], dst: q1}
```

state diagram



computation diagram



Related work

- **GUI:**
  - JFLAP [14]
  - OpenFLAP [9]
  - GUItar [1]
- **API:**
  - PyFormlang [15]
  - VisualAutomata [7]
- **Lang:**
  - Penrose [19]

Table 1: Feature comparison, where G represents GIDAYU, "Custom. viz." means customizable visualization, "Comp. diagram" means computation diagram.

	[14]	[9]	[1]	[15]	[7]	[19]	G
UI	GUI	GUI	GUI	API	API	DSL	DSL
Custom. viz.	✗	✗	✓	✗	✗	✓	✓
State diagram	✓	✓	✓	✓	✓	✗	✓
Computation	✓	✗	✗	✓	✓	✗	✓
Comp. diagram	✗	✗	✗	✗	✗	✗	✓

UMass Boston

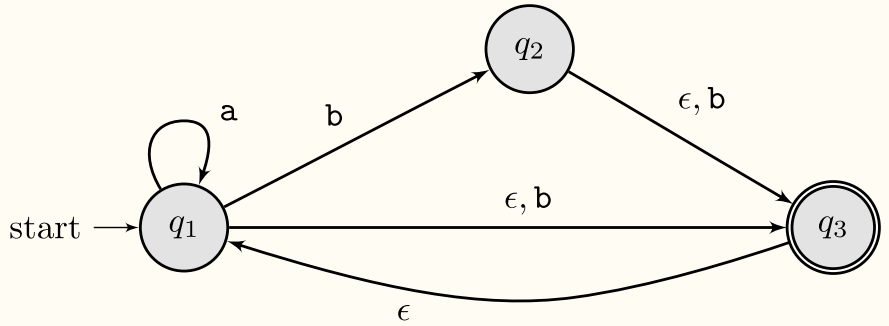
State diagrams

NFA state diagram

Spec

```yaml
type: nfa
states:
  q1: {label: q_1, initial: true}
  q2: {label: q_2}
  q3: {label: q_3, final: true}
transitions:
- {src: q1, actions: [b], dst: q2}
- {src: q1, actions: [a], dst: q1}
- {src: q1, actions: [null,b], dst: q3}
- {src: q2, actions: [null,b], dst: q3}
- {src: q3, actions: [null], dst: q1}
```

State diagram

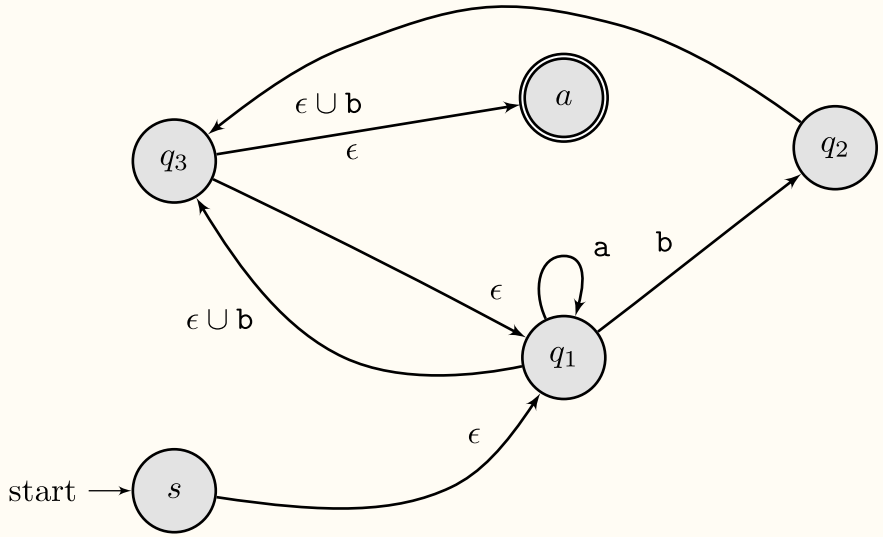


GNFA state diagram

Spec

```
type: gnfa
states:
  s: {label: s, initial: true}
  q1: {label: q_1}
  q2: {label: q_2}
  q3: {label: q_3}
  a: {label: a, final: true}
transitions:
- {src: s, dst: q1, actions: [[]]}
- {src: q1, actions:[{char: b}], dst: q2}
- {src: q1, actions:[{char: a}], dst: q1}
- {src: q1, actions:[{union: {left:[], right:{char: b}}}], dst:
    q3}
- {src: q2, actions:[{union: {left:[], right:{char: b}}}], dst:
    q3}
- {src: q3, actions:[[]], dst: q1}
- {src: q3, dst: a, actions: [ [] ] }
```

State diagram



The specification can be generated directly from the NFA spec

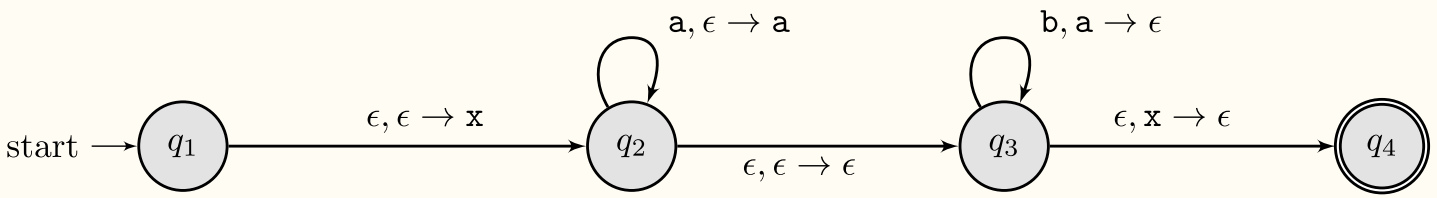
PDA state diagram

Spec

```yaml
type: pda
states:
  q1: {initial: true, label: q_1}
  q2: {label: q_2}
  q3: {label: q_3}
  q4: {label: q_4, final: true}

transitions:
 - {src: q1, push: x, dst: q2}
 - {src: q2, read: a, push: a, dst: q2}
 - {src: q2, dst: q3}
 - {src: q3, read: b, pop: a, dst: q3}
 - {src: q3, pop: x, dst: q4}
```

State diagram



Operations on specs

- convert NFA into Generalized-NFA
- remove a state from a Generalized-NFA (intermediate step on converting to REGEX)
- convert NFA into DFA
- union, intersection, concatenation, Kleene-star of NFAs

UMass Boston

Computation diagrams

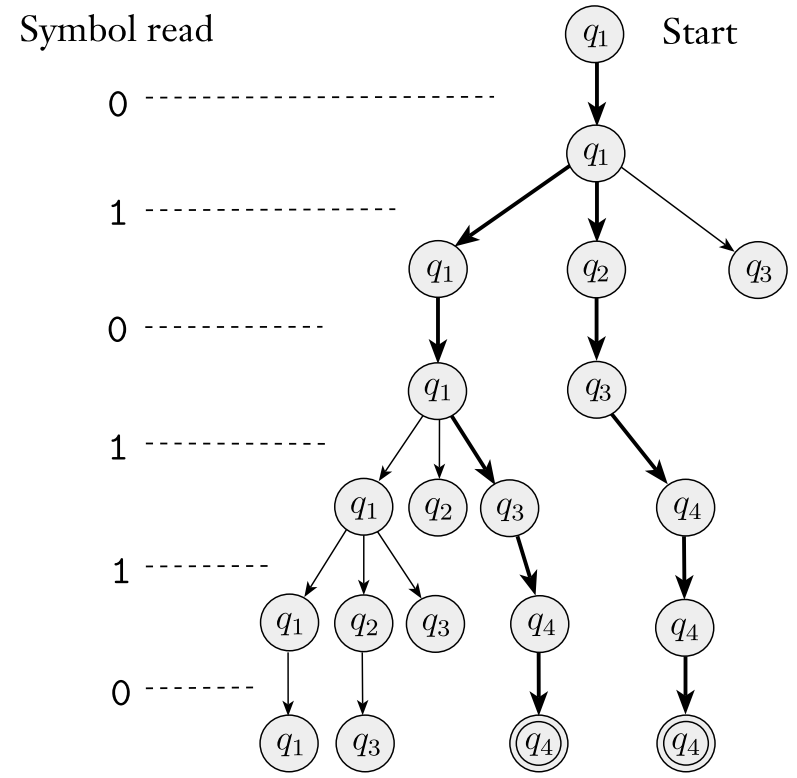


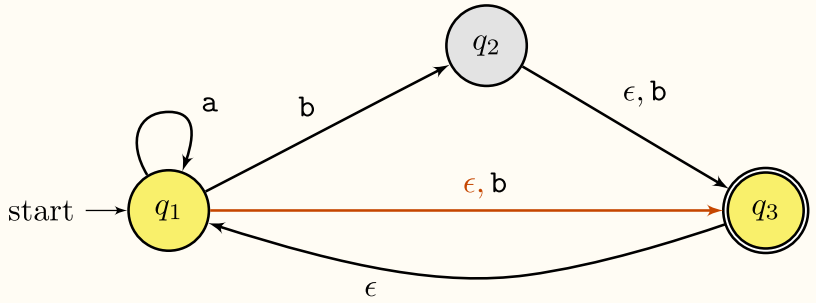
FIGURE 1.29
The computation of N_1 on input 010110

Computation diagrams

- Visual representation of the yields relation $C_1 \models C_2$
  - $(q, cw) \models (r, w)$ if $r \in \delta(q, c)$
  - $(q, w) \models (q, w)$ if $r \in \delta(q, \epsilon)$
- Tree-based visualization **does not scale**
  - the same configuration may appear multiple times in the same level
  - redundancy leads to student confusion
- We opt for a directed-acyclic graph diagram

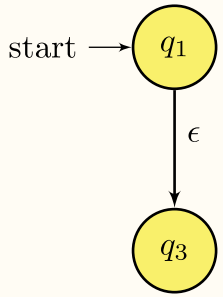
UMass Boston

Specification

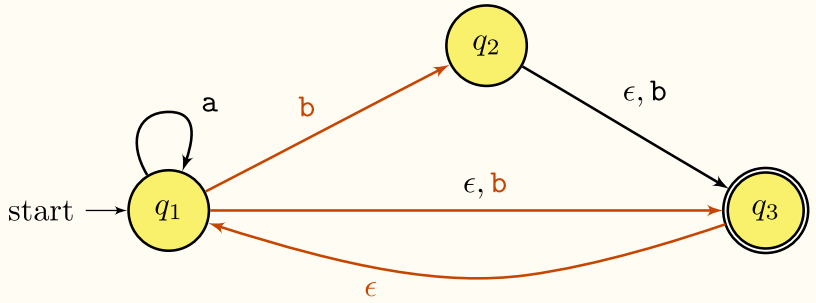


Compute `bb`

Initial configurations

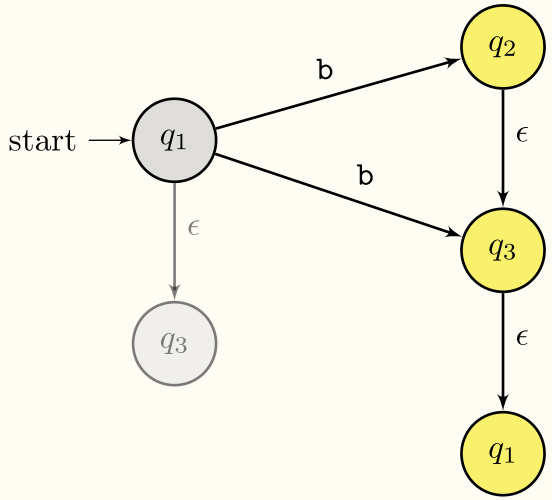


Specification

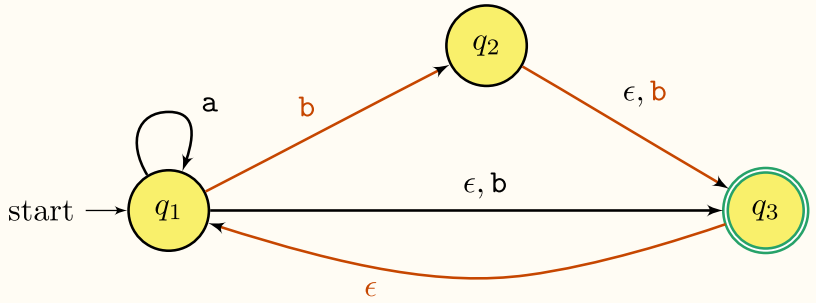


Compute bb

Process b

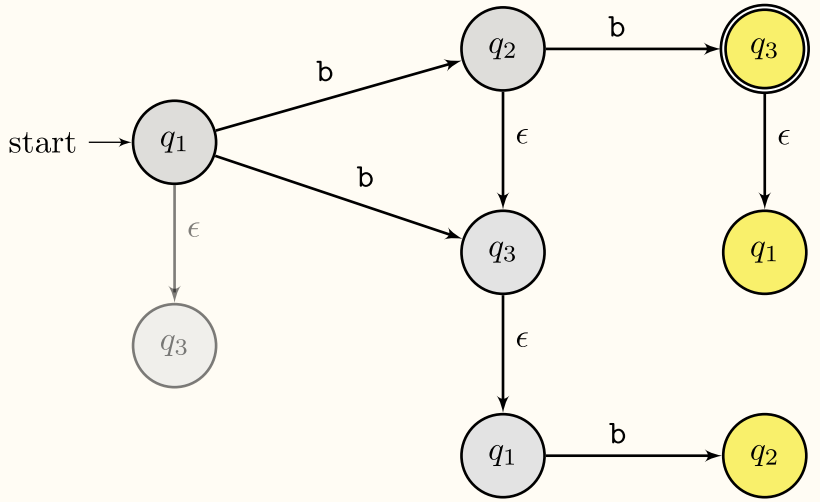


Specification

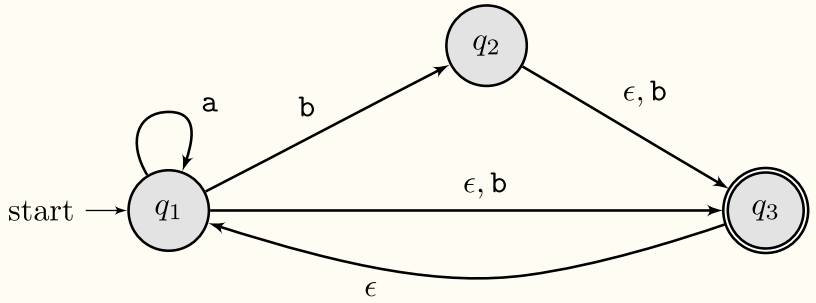


Compute bb

Process bb

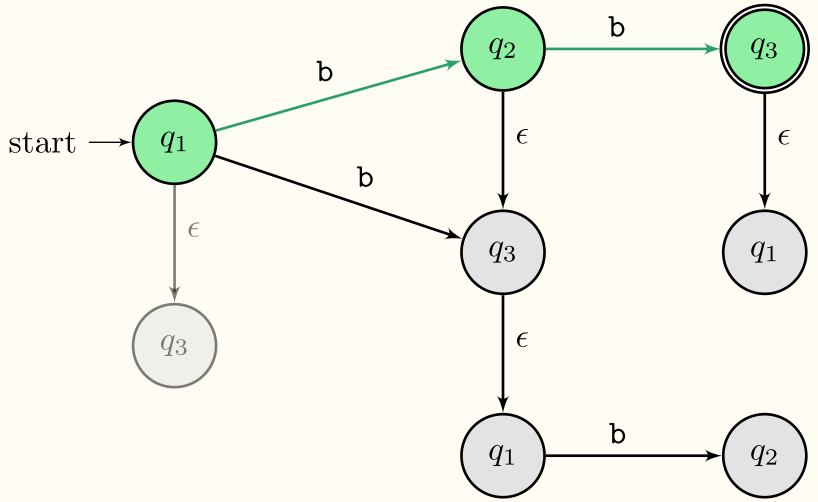


Specification



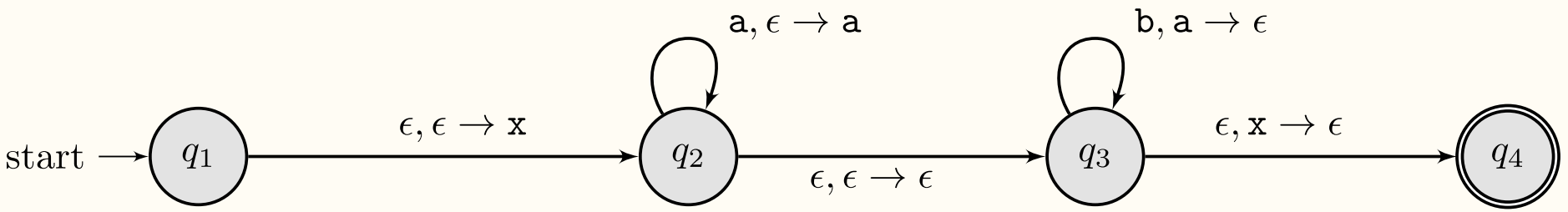
Compute bb

Accepting configuration

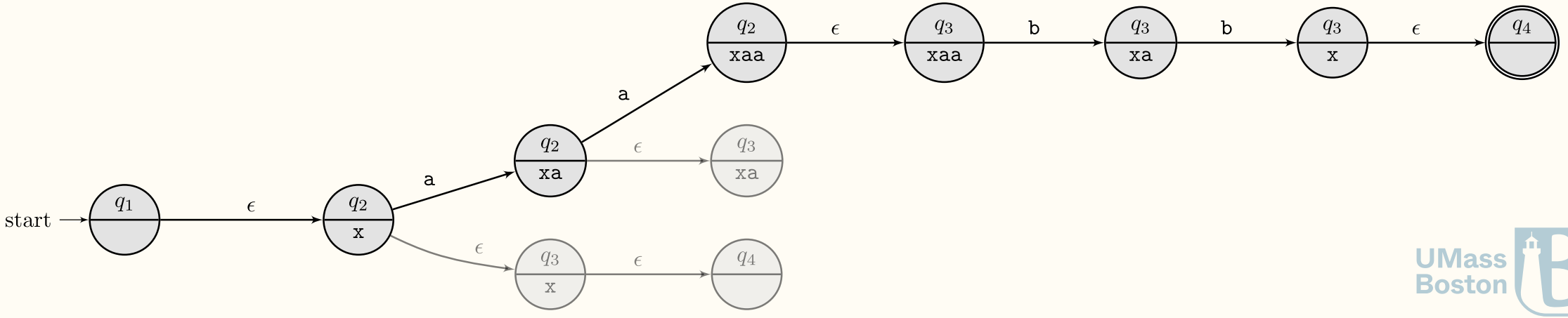


PDA $a^n b^n$ computing $a^2 b^2$

State diagram



Computation diagram



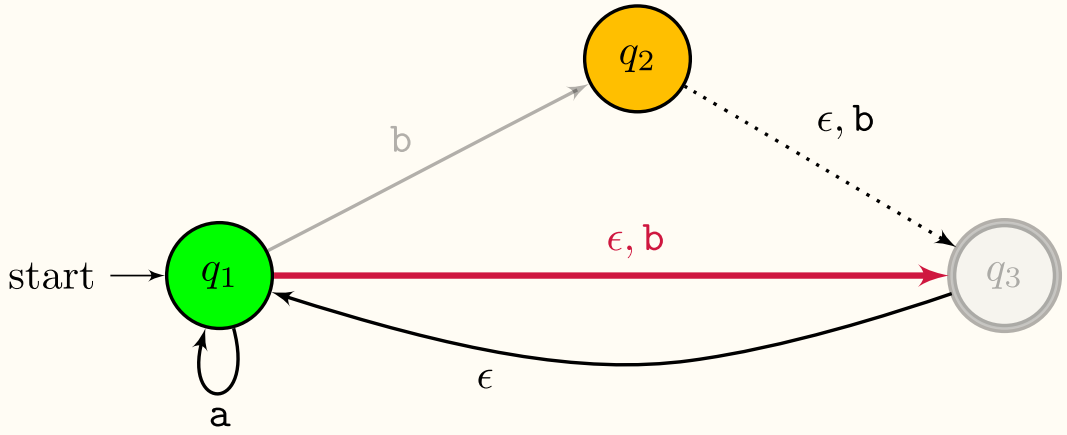
Customization

Customization

Styling state diagrams

Gidayu is powered by Graphviz, LaTeX, and TiKz

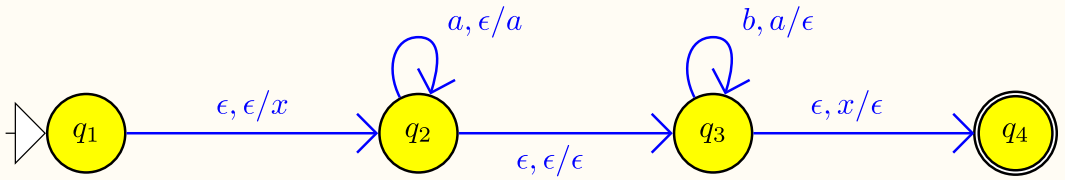
```
type: nfa
states:
  q1: {label: q_1, initial: true, style: [fill=green]}
  q2: {label: q_2, highlight: true}
  q3: {label: q_3, final: true, hide: true}
transitions:
- {src: q1, actions:[b], dst: q2, hide: true}
- {src: q1, actions:[a], dst: q1, topath: [loop below]}
- {src: q1, actions:[null, b], dst: q3, highlight: true}
- {src: q2, actions:[null, b], dst: q3, style: [dotted]}
- {src: q3, actions:[null], dst: q1}
```



Customization

Styling the visual template

```
state:
  default: [fill=yellow]
transition:
  default: [blue]
  format: |
    {% for x in actions %}
      {{ x.read_char }},{{ x.pop_char }}/{{ x.push_char }}
    {% endfor %}
```



Conclusion & future work

Conclusion

- Turing: a Coq library to teach FLA
- an experience report on using proof assistants to teach FLA
- Gidayu: visualize automata and computations

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**

UMass Boston

Assumptions

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

UMass Boston