Gidayu
Visualizing Automaton and Their Computations

Tiago Cogumbreiro and Gregory Blike

UMass Boston

July 11, 2022

ITiCSE

Context

Subject ***Formal Language and Automata*** (FLA)

> An important subject in the curriculum of undergraduate computer science [Computer Science Curricula 2013]

- **Material:** formal languages (regular, context-free, Turing-recognizable), decidability, computability
- **Diagrams defined in automata theory:** (non)deterministic finite automata, (non)deterministic pushdown automata, Turing machines

Context

Subject ***Formal Language and Automata*** (FLA)

> An important subject in the curriculum of undergraduate computer science [Computer Science Curricula 2013]

- **Material:** formal languages (regular, context-free, Turing-recognizable), decidability, computability
- **Diagrams defined in automata theory:** (non)deterministic finite automata, (non)deterministic pushdown automata, Turing machines

Problem

How to depict ***mathematical diagrams*** simply and rigorously?

UMass Boston

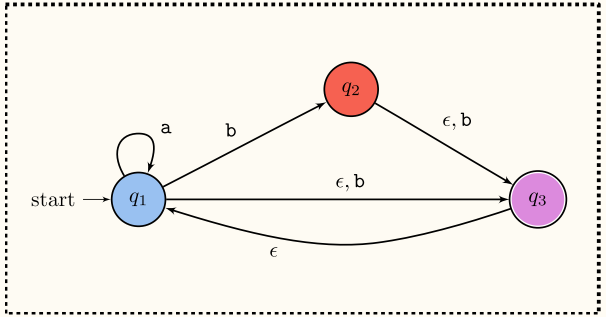
This talk: Gidayu

- Motivation and related work
- State diagrams
- Computation diagrams
- Customization
- In the classroom

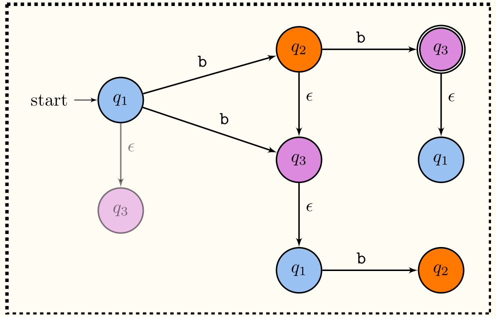
nfa spec

```
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



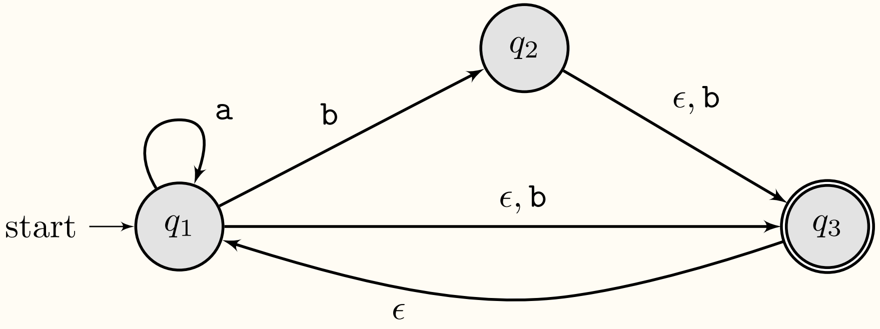
computation diagram



Motivation and related work

Drawing mathematical diagrams

State diagram



Requirements

- How easy is it to create a diagram?
- How to ensure correctness?
- How to customize appearance?

Overview

- depiction is a directed graph
- nodes are **states**
- edges are **transitions**
- labels on edges are **actions**
- starting/final states are visually distinct
- Mathematical diagrams have rigorous constraints
- Author-dependent notations (visual styling)
- Different notations → confusion, harm correctness

UMass Boston

Drawing mathematical diagrams

Requirement	*Rapid Prototyping*	*Correctness*	*Customization*
Image editors (Inkscape)	-laborious, -low reusability	-error prone	+full control
Interactive editors (JFLAP)	+easy to draw 1, -hard to automate	+full support	-limited

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	✗	✗	✗	✗	✗	✗	✓

Insights

- Limited support to depict computation
- Limited support to customize appearance

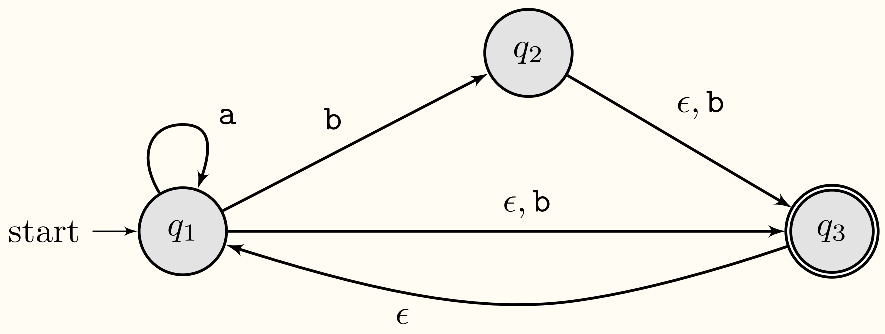
State diagrams

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

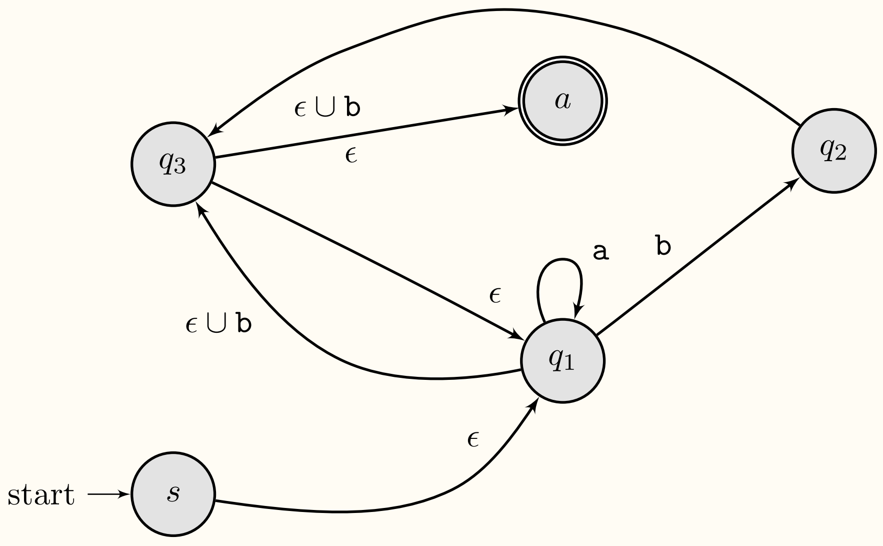


GNFA state diagram

Spec

```yaml
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

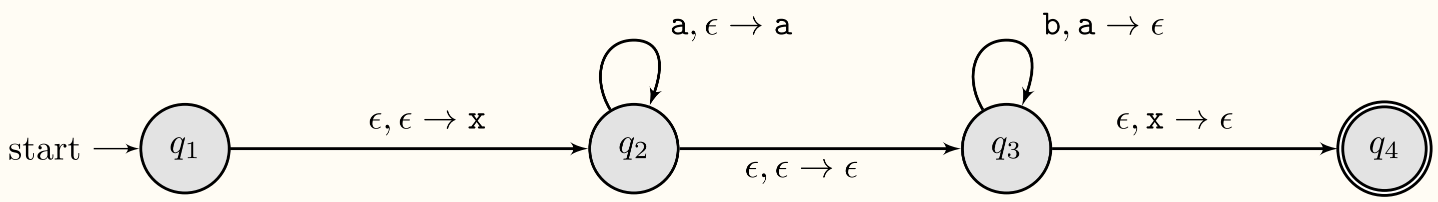
PDA state diagram

Spec

```
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}
```

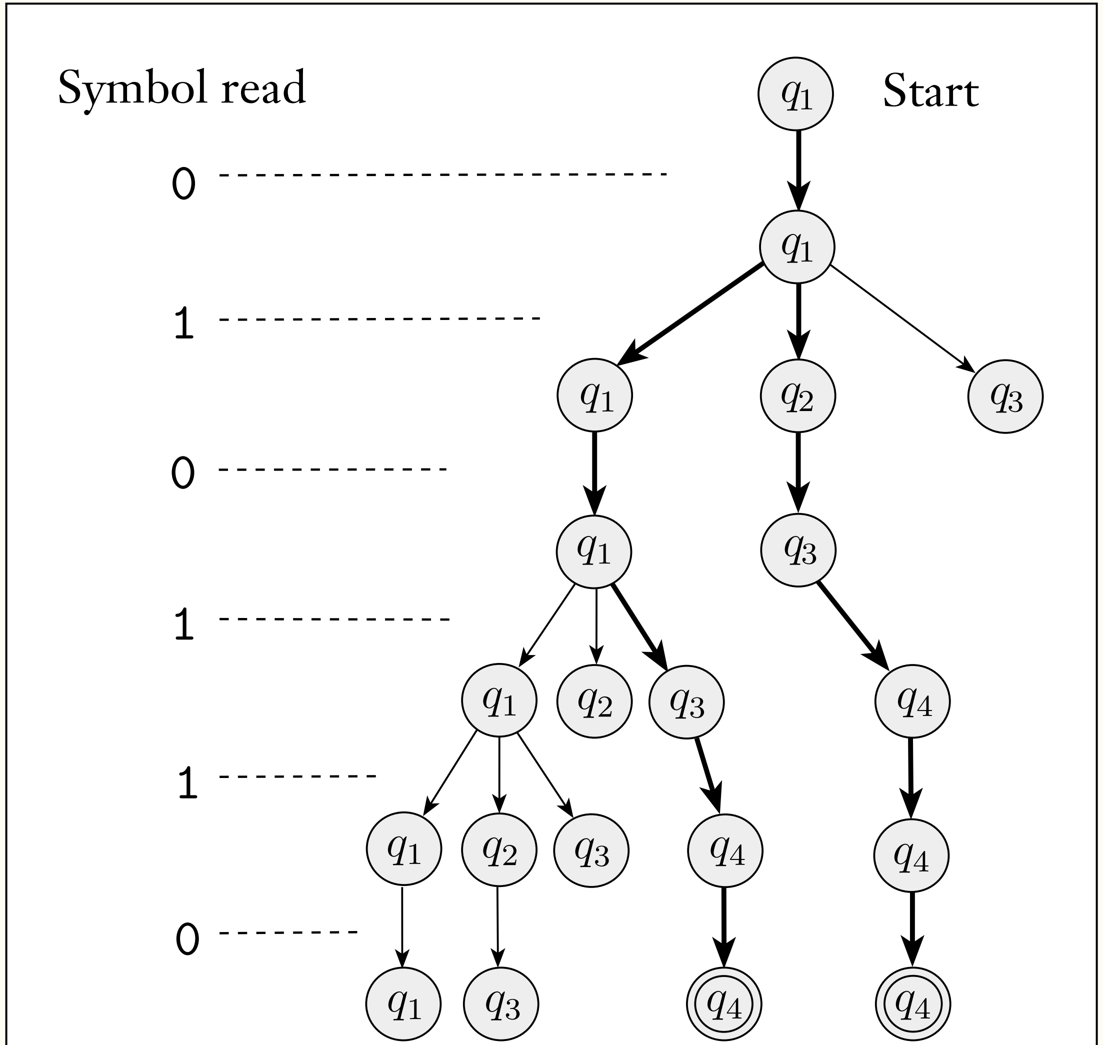
State diagram



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

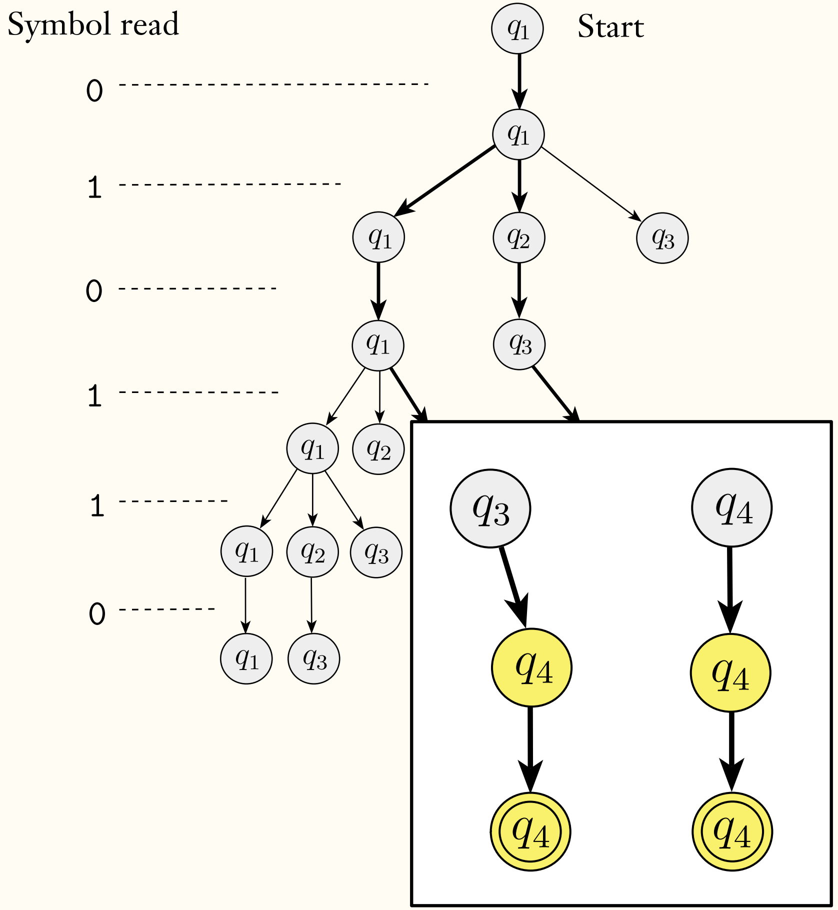
Computation diagrams



Computation diagrams

Tree representation

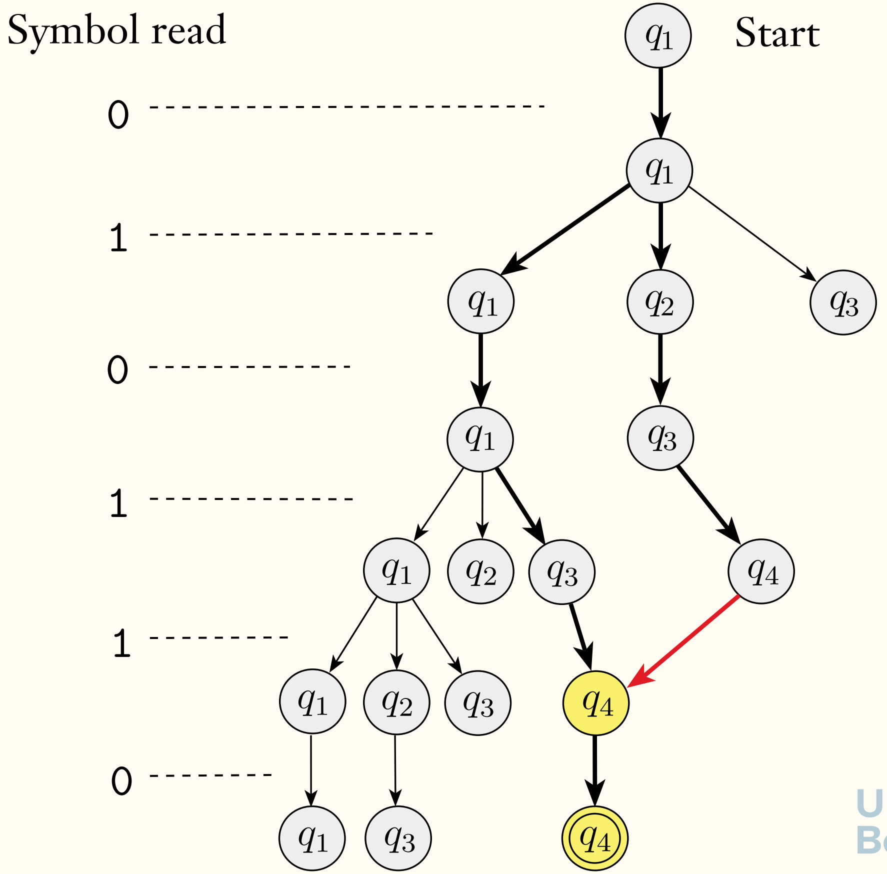
- Tree-based visualization **does not scale**
- the same configuration may appear multiple times in the same level
- redundancy leads to student confusion



Computation diagrams

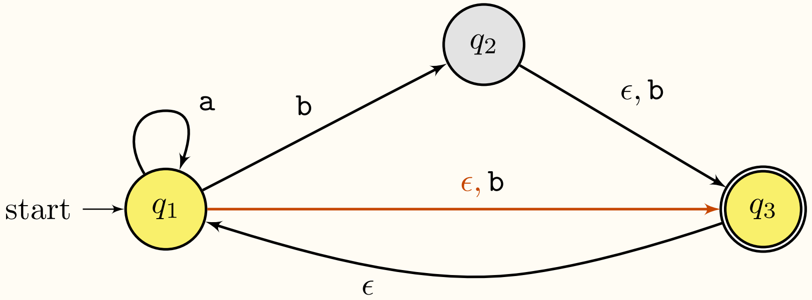
Graph representation

- Graph-based avoids node redundancy
- Our tool defaults to an **acyclic** rendering



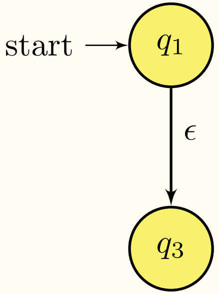
Visualizing computation stepwise

Specification

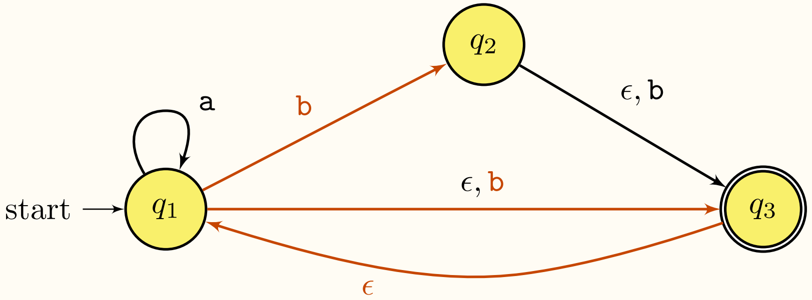


Compute bb

Initial configurations

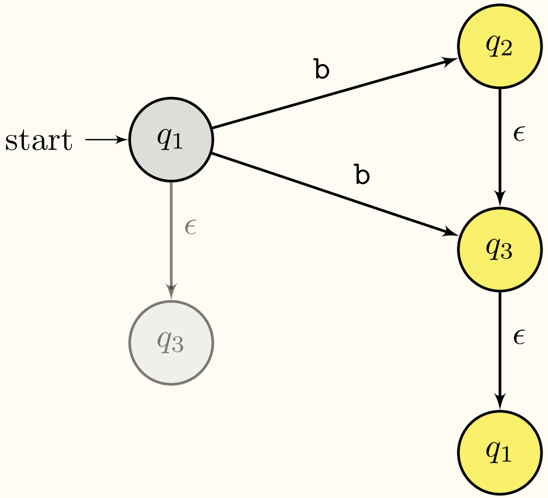


Specification

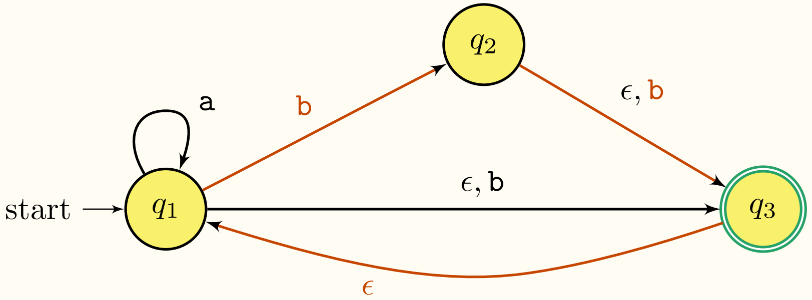


Compute bb

Process b

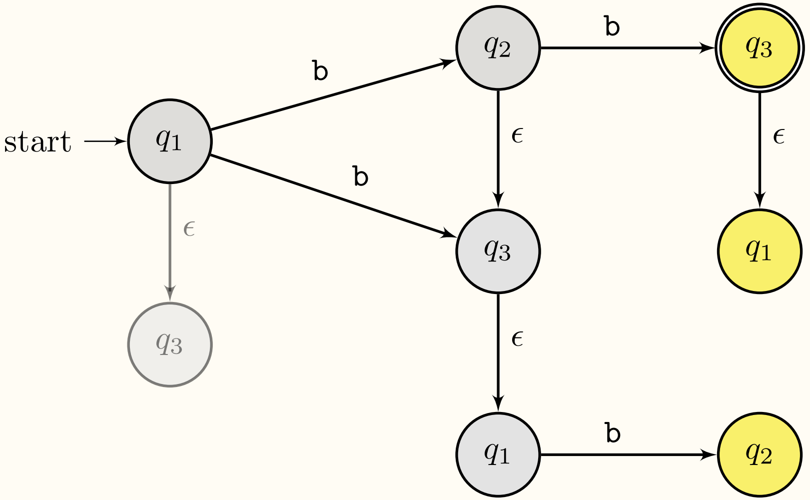


Specification

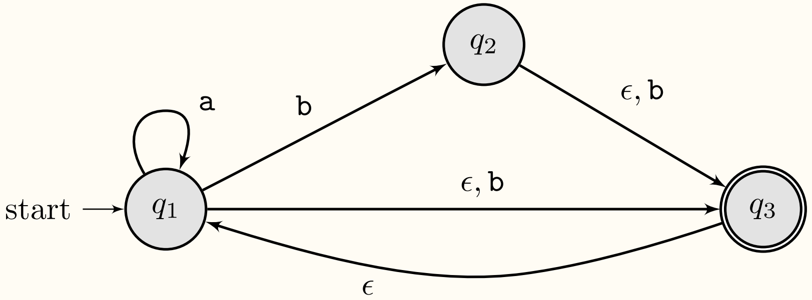


Compute bb

Process bb

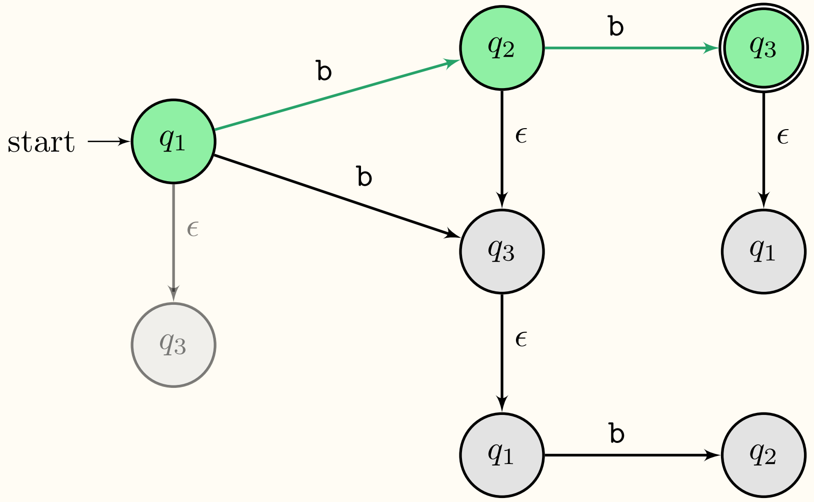


Specification



Compute bb

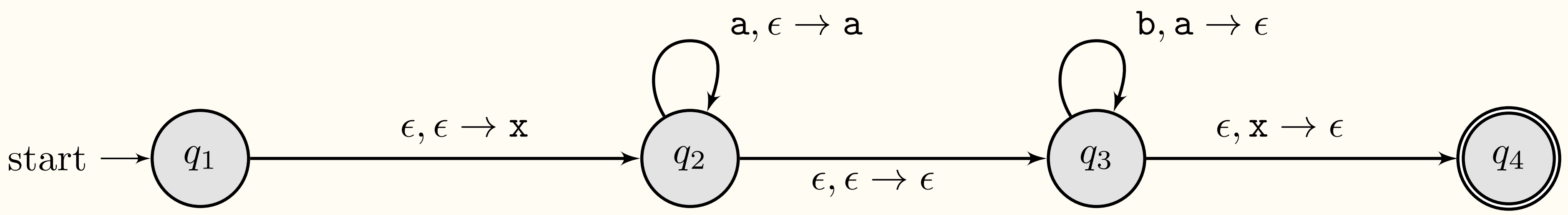
Accepting configuration



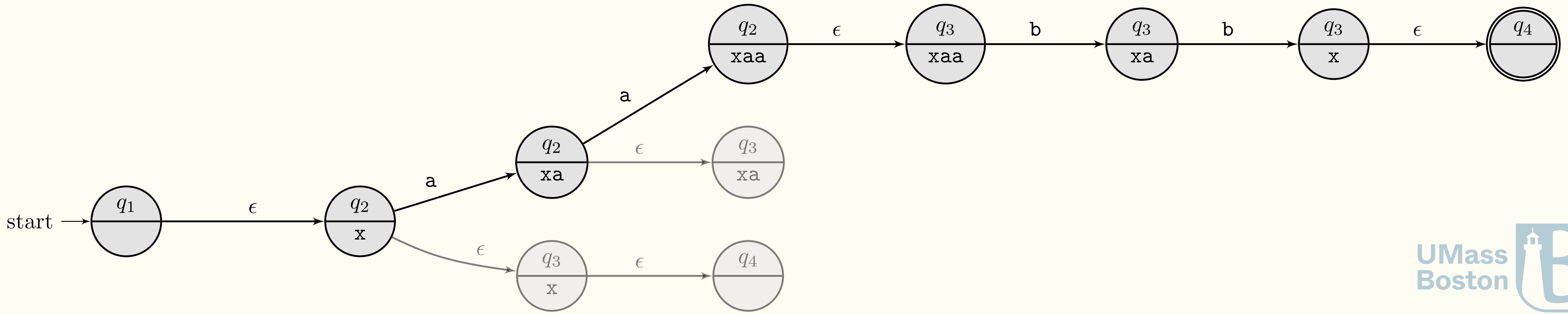
Visualizing computations of pushdown automata

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

State diagram



Computation diagram



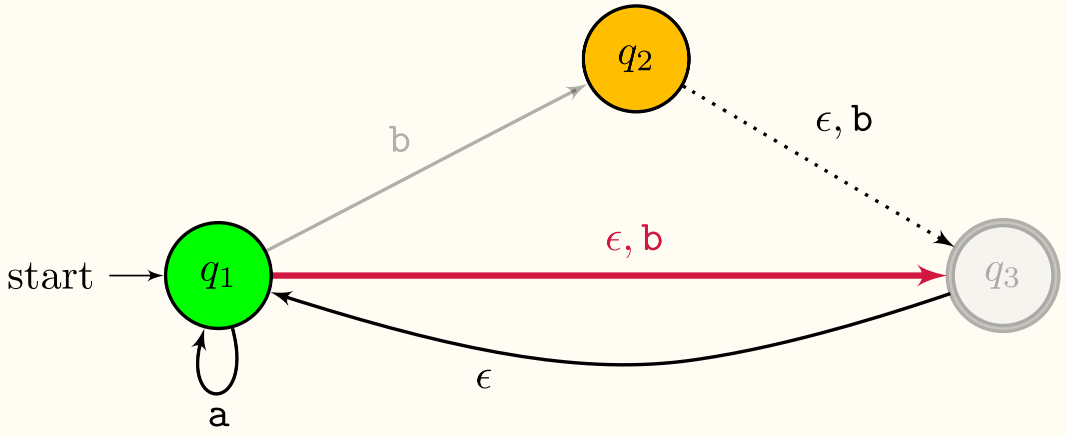
Customization

Customization

Styling state diagrams

Gidayu is powered by Graphviz, LaTeX, and TiKz

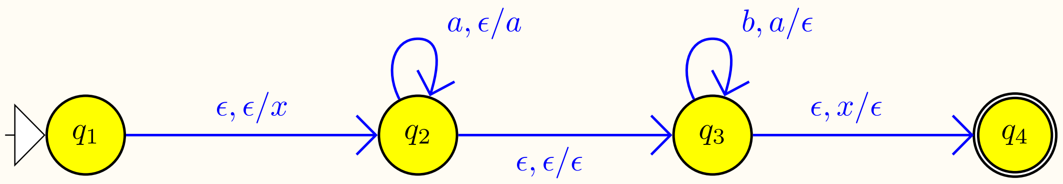
```
type: nfa
states:
  q1: {label: q_1, initial: true, style: [fill=green]}
  q2: {label: q_2, highlight: true}
  q3: {label: q_3, final: true, hide: true}
transitions:
- {src: q1, actions:[b], dst: q2, hide: true}
- {src: q1, actions:[a], dst: q1, topath: [loop below]}
- {src: q1, actions:[null, b],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

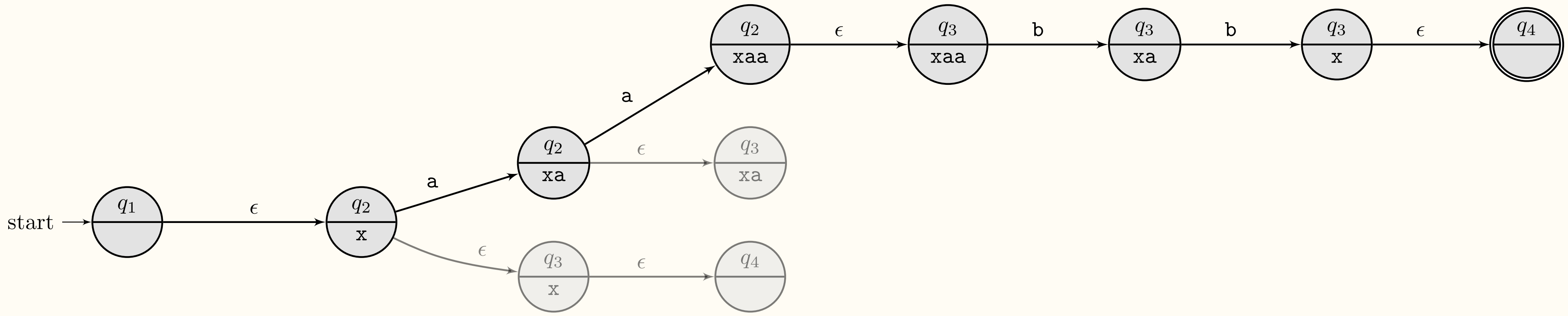


Gidayu in the classroom

Computation diagrams as visual proofs

Existential quantifier

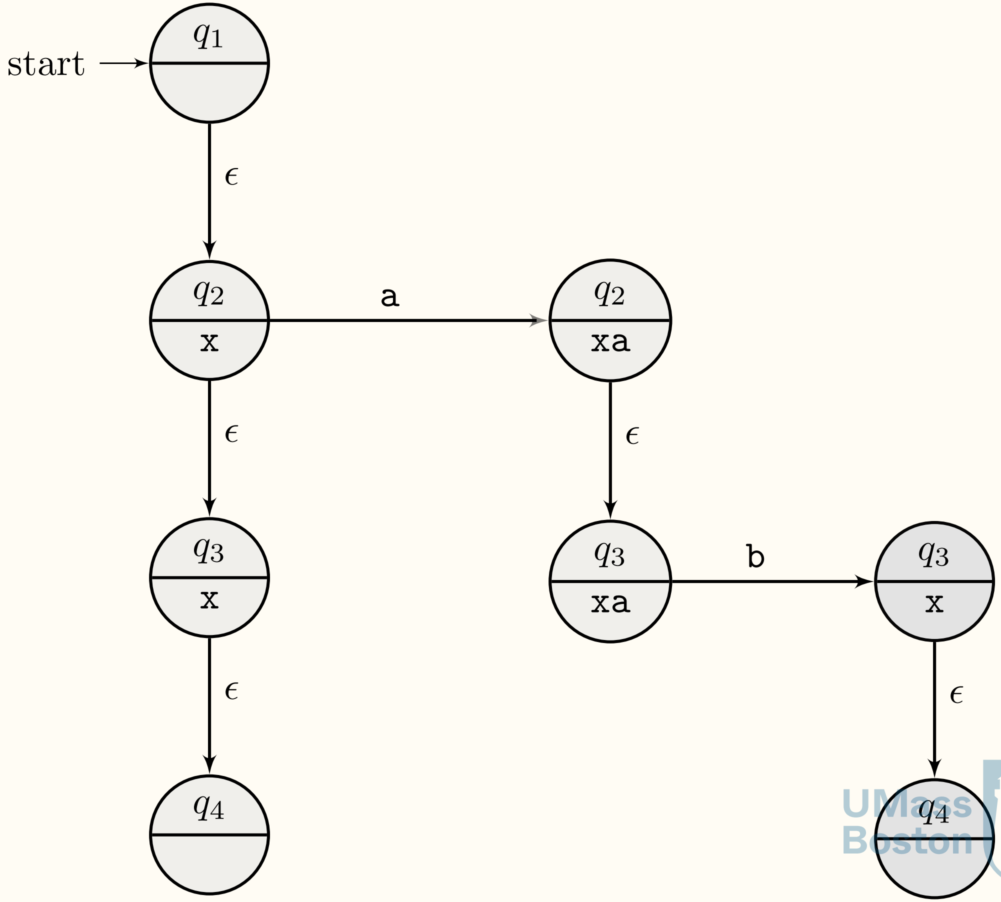
- **Statement:** show that an NFA accepts an input
- **Proof:** give a path that reaches an accepting configuration (or a computation diagram)



Computation diagrams and visual proofs

Universal

- Statement: show that an NFA rejects an input
- Proof: give a computation diagram that shows no accepting configuration



Conclusion & future work

Gidayu: Visualizing Automaton and Their Computations

Tiago Cogumbreiro & Gregory Blike

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

Conclusion

- Automaton types: NFA, DFA, GNFA, PDA, DPDA
- Operations: NFA to REGEX, union, concatenate, intersect, Kleene-star
- Used at UMass Boston during 3 semesters
- Progressively display computation
- Computation diagrams as visual proofs
- Fine-tune diagrams with SVG editors

Future Work

- Add support for other diagrams (eg, Turing machines)
- Visualizing large computations / PDAs with infinite computations