CSCI 340: Computational Models

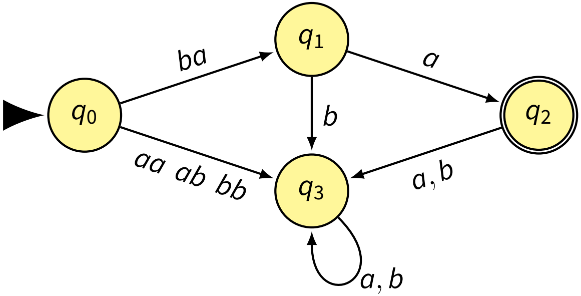
Transition Graphs

Relaxing Restraints on Inputs

- We can build an FA that accepts only the word **baa**!
- 5 states because an FA can only process one letter at a time.
- Can we construct a more powerful machine?

Relaxing Restraints on Inputs

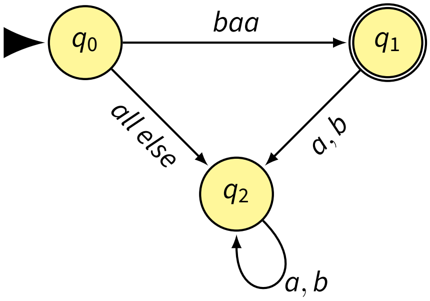
- We can build an FA that accepts only the word **baa**!
- 5 states because an FA can only process one letter at a time.
- Can we construct a more powerful machine?



Only processing one- or two- characters at a time

Relaxing Restraints on Inputs

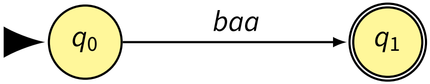
- We can build an FA that accepts only the word **baa**!
- 5 states because an FA can only process one letter at a time.
- Can we construct a more powerful machine?



Processing up to three characters at a time

Relaxing Restraints on Inputs

- We can build an FA that accepts only the word **baa**!
- 5 states because an FA can only process one letter at a time.
- Can we construct a more powerful machine?


$$\begin{tikzcd} {} \arrow[r] & q_0 \arrow[r, "baa"] & \boxed{q_1} \end{tikzcd}$$

The most basic of possible FA-like machines accepting only *baa*

But – we have a problem: what happens with *baabb*?

A Black-Hole State?

Up until this point, we had always specified a transition for every single letter from every single state

- Rules of FAs states we cannot stop reading input until we have no more letters
- Do we want to specify an imaginary hell state for every FA?
- Alternatively introduce a new term to describe what happens

A Black-Hole State?

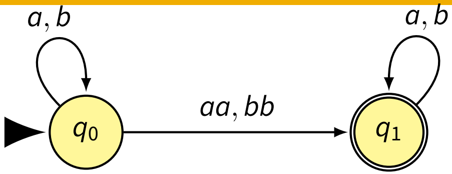
Up until this point, we had always specified a transition for every single letter from every single state

- Rules of FAs states we cannot stop reading input until we have no more letters
- Do we want to specify an imaginary hell state for every FA?
- Alternatively introduce a new term to describe what happens

Definition

When an input string that has not been completely read reaches a state (final or otherwise) that cannot leave because there is no outgoing edge that it may follow, we say that the input (or the machine) **crashes** at that state. Execution terminates and the input *must* be rejected.

Example: A Double-Letter Accepting Machine

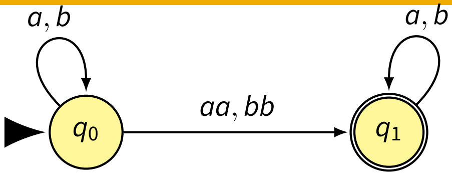


Problem

How many letters should we read at a time?

Discussion

Example: A Double-Letter Accepting Machine


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "{a, b}"] \arrow[rr, "{aa, bb}"] & & \boxed{q_1} \arrow[loop above, "{a, b}"] \end{tikzcd}$$

Problem

How many letters should we read at a time?

Discussion

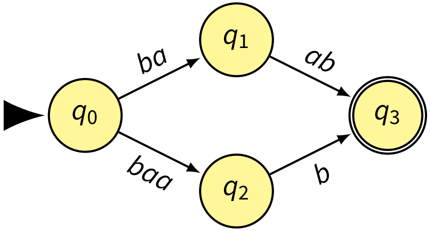
Given **baa** we can tokenize it in the following ways:

- b-a-a
- b-aa
- ba-a

Only one of these yields admission into the final state (q_1)

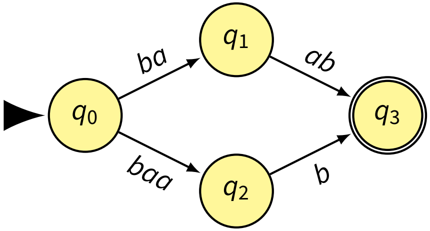
A Potential Problem

A string is *accepted* by a machine if there is **some** way it could be processed so as to arrive at a final state.



A Potential Problem

A string is *accepted* by a machine if there is **some** way it could be processed so as to arrive at a final state.



- We can accept *baab* in two different ways!
- These are no longer Finite Automata
- We shall refer to these new machines as **transition graphs**

Transition Graphs

Definition

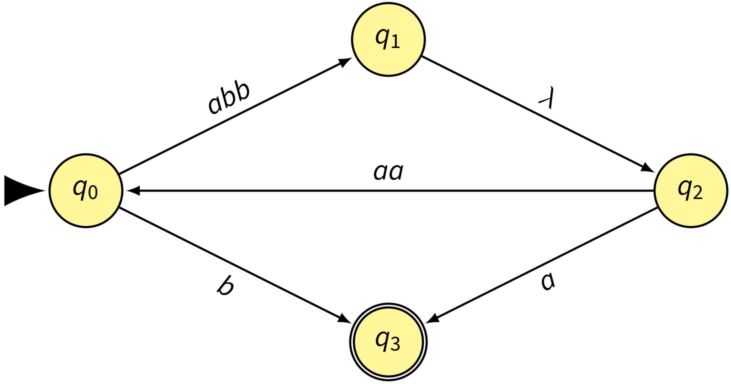
A **transition graph**, abbreviated **TG**, is a collection of three things:

1. A finite set of states, at least one of which is designated as the start state and some (maybe none) of which are designated as final states.
2. An alphabet Σ of possible input letters from which input strings are formed.
3. A finite set of transitions (edge labels) that show how to go from some states to some others, based on reading specified substrings of input letters (possibly even the null string λ).

TGs were invented by John Myhill in 1957

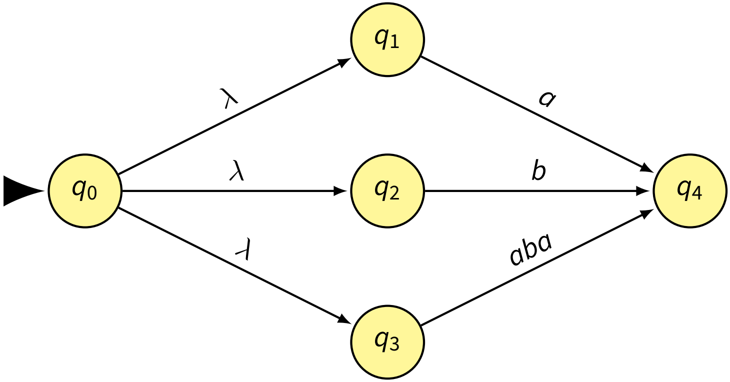
A **successful path** through a transition graph is a series of edges forming a path beginning at some start state and ending at a final state. Concatenating the edges visited will yield the input string.

Example with λ transitions

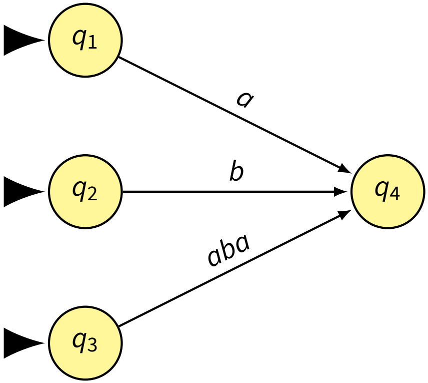


What language is accepted by this **TG**?

Multiple Start States



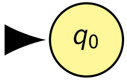
Multiple Start States



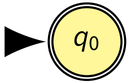
- *language-acceptor* equivalent (the TG on the prior slide is functionally equivalent as the TG on this slide)
- Important note: every FA is a TG, however every TG is not an FA

Looking at Simple Transition Graphs

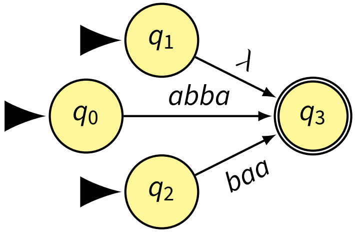
❶



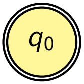
❷



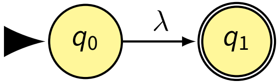
❸



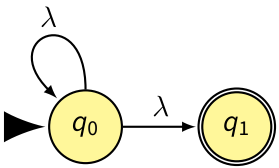
❹



❺

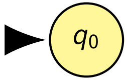


❻

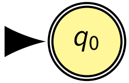


Looking at Simple Transition Graphs

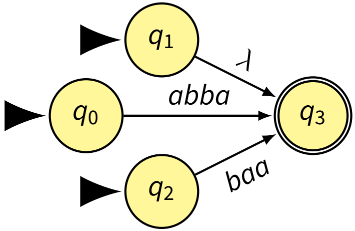
1. accepts nothing (no final)



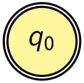
2. accepts only λ



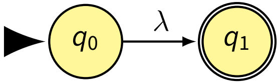
3. accepts only λ, *abba*, *baa*



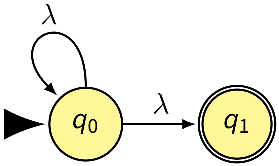
4. accepts nothing (no start)



5. accepts only λ

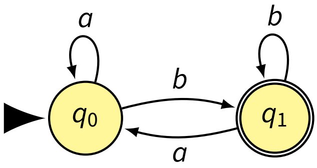


6. accepts only λ

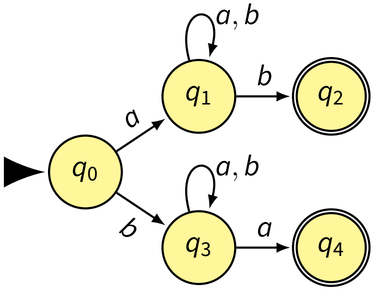


Examples – What do they do?

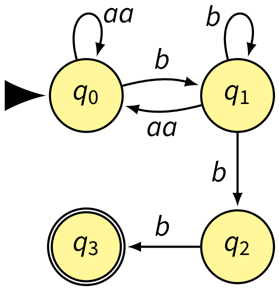
TG1:



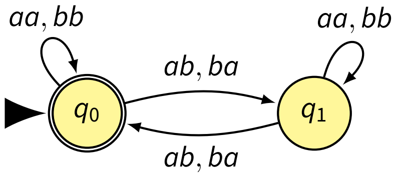
TG2:



TG3:



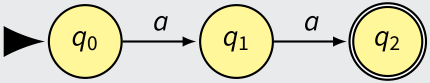
TG4:



Infinite Paths?

Question

Can we construct a **TG** which has infinitely many accept paths for a finite-length string?

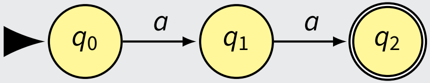


Solution

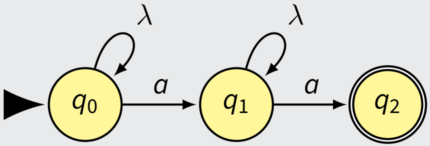
Infinite Paths?

Question

Can we construct a **TG** which has infinitely many accept paths for a finite-length string?



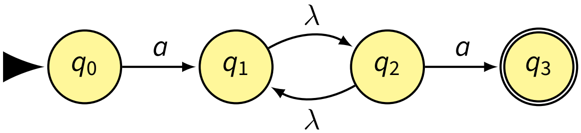
Solution



λ Circuits

Question

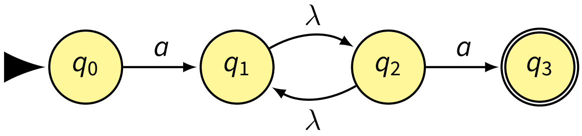
How can we remove λ-transitions?

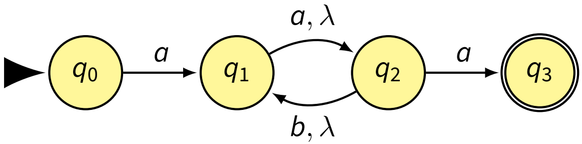

$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[r, "a"] & q_1 \arrow[r, bend left, "\lambda"] & q_2 \arrow[l, bend left, "\lambda"] \arrow[r, "a"] & \boxed{q_3}
\end{tikzcd}$$

λ Circuits

Question

How can we remove λ-transitions?





Generalized Transition Graphs (**GTG**s)

- We want to liberate! state-to-state transitions
- Allow the input to progress from one state to state
  - Not with sequences of characters
  - But with languages! $L_1, L_2, \ldots, L_n$
  - How do we want to represent the languages?

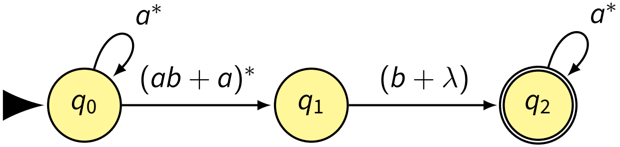
Definition

A **generalized transition graph** (**GTG**) is a collection of 3 things:

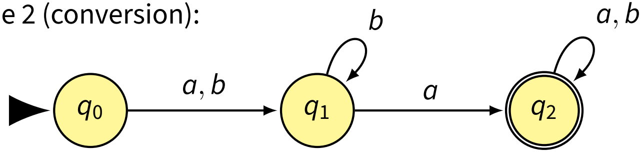
1. A finite set of states, of which as least one is a start state and some (maybe none) are final states.
2. An alphabet Σ of input letters.
3. Directed edges connecting some pairs of states, each labeled with a *regular expression*.

Examples of a GTG

Example 1 (demonstration):



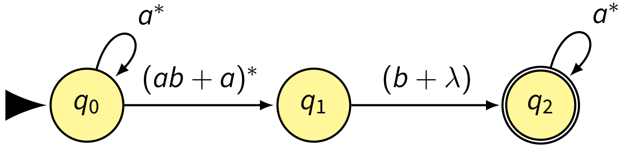
Example 2 (conversion):



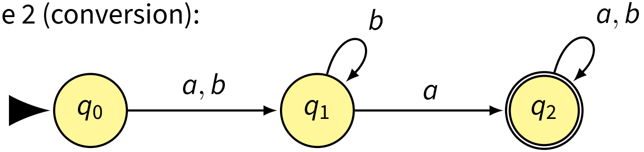
Loops == Kleene Star $a, b == (a + b)$

Examples of a GTG

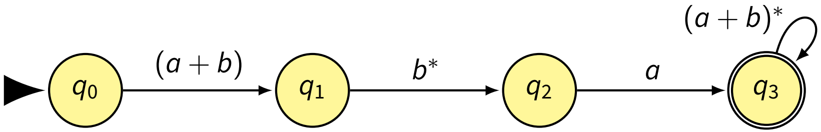
Example 1 (demonstration):



Example 2 (conversion):



Loops == Kleene Star $a, b == (a+b)$



Non-Determinism

Or, how I learned to stopped worrying and love GTGs

- GTGs force us to face a deep, subtle, and disturbing fact:
  - Just as $*$ and $+$ in a regular expression represent a potential multiplicity of choices, so does the possible multiplicity of paths to be selected from a TG.
  - In a GTG, the choices are **static and dynamic**
  - We often have choices of edges at each state, each labeled with an *infinite language* of alternatives
  - The number of ways to transition from Q_i to Q_j might be ∞
- We can't forbid it ("Dread It. Run From It. Destiny Still Arrives.")
- GTGs are **non-deterministic**. Human choice becomes a factor in selecting the path; the machine doesn't make all its own determinations.