



little languages

lecture 20:


Regular Expressions & Automata

Open PolleEv.com/compunc
And have some paper / pencil out.

Regular Expressions Review

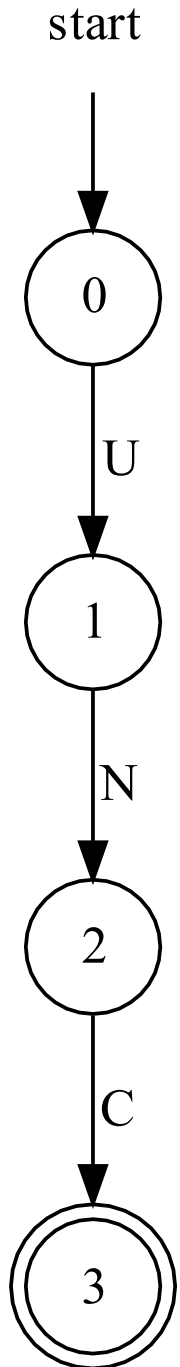
- Little Language to specify textual pattern matching.
- Fundamental Operators:
 - Catenation / Concatenation - "AB" - A and then B
 - Alternation / Union - "A|B" - A or B
 - Closure / Zero or More - "A*" - Zero or more A
- Syntactical Sugar Operators:
 - Zero-or-one: ?
 - One-or-more: +
- You can *compose* Regular Expressions by combining operators.

Goal: Get a CPU to Process a Regular Expression

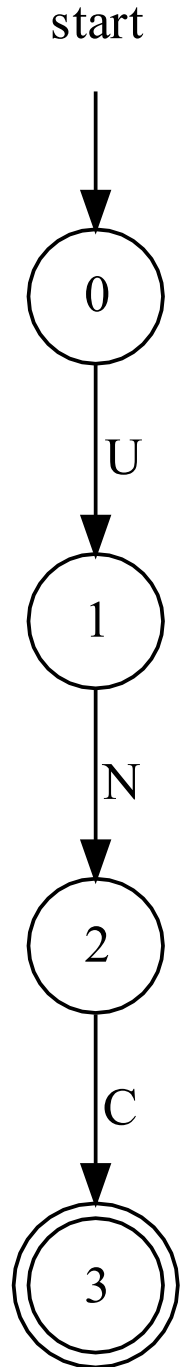
- Regular Expressions are a *language* for *humans* to express textual patterns.
 - Computers inherently know *nothing* about regular expressions.
- Humans are very slow at processing large volumes of text precisely.
 - Computers... 
- How do we translate this *human language* into computational *machinery*?
 - Machinery being loosely defined as data structures + algorithms.
- We need to rely on some formalism from the 1950s...

Transition Diagrams (Directed Graphs)

- You can represent regular expressions as transition diagrams, where:
- States are Nodes
 - Represented as indexed circles
 - A state captures everything you need to know at any point in the process
- Transitions are Directed Edges
 - Represented as labeled, directed edges between nodes
 - A transition bridges current state to the next state when input matches label.
- A Start node (Node 0 right) designates the initial state.
- *Accepting* node(s) (Node 3 right) designates success if, after processing the input, it is the final state. *Accepting* or *Final* nodes are double circles.

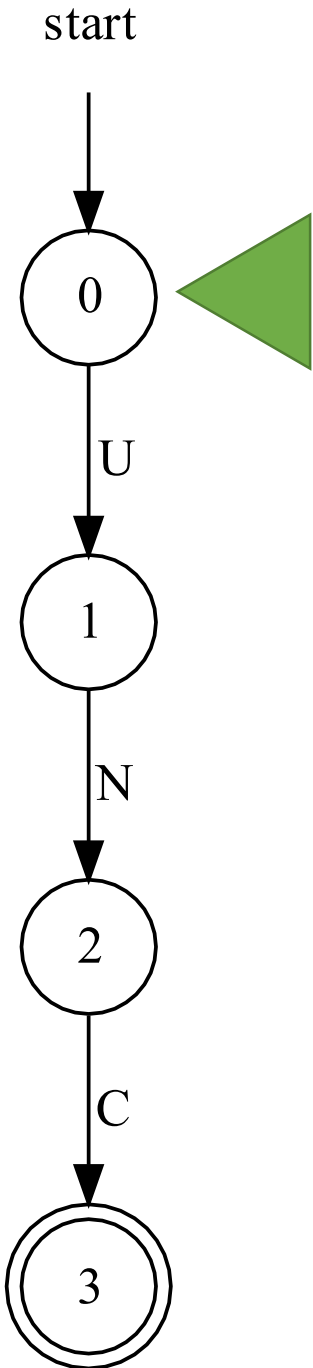


Tracing a Transition Diagram



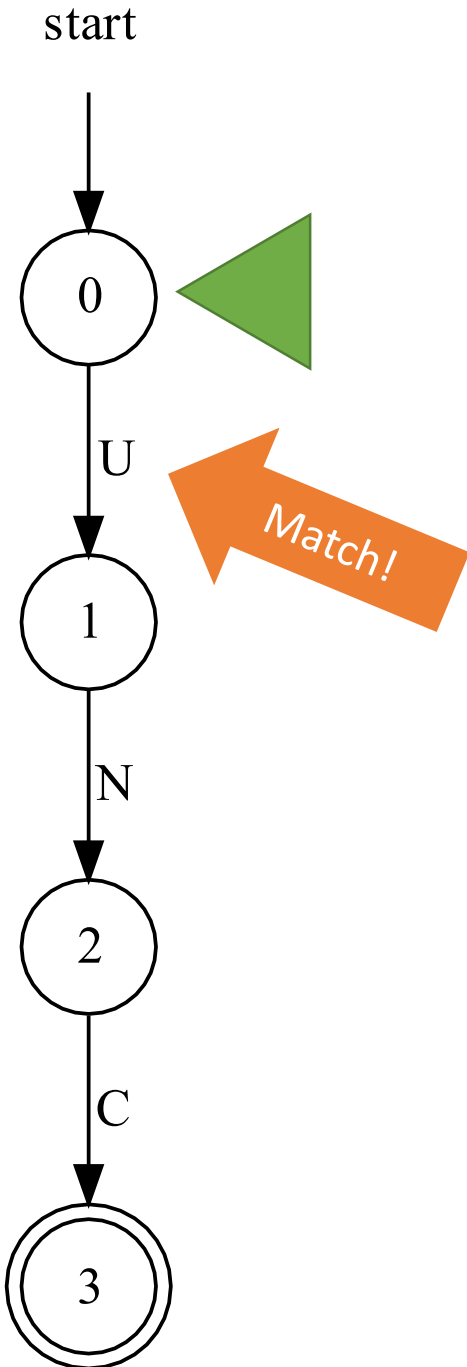
- Transition Diagram: Left
- Input String: "USC"
- When processing the input using the transition diagram, does it end in the accepting state 3?

Tracing a Transition Diagram



Before any input is considered, begin in the starting state.

Tracing a Transition Diagram



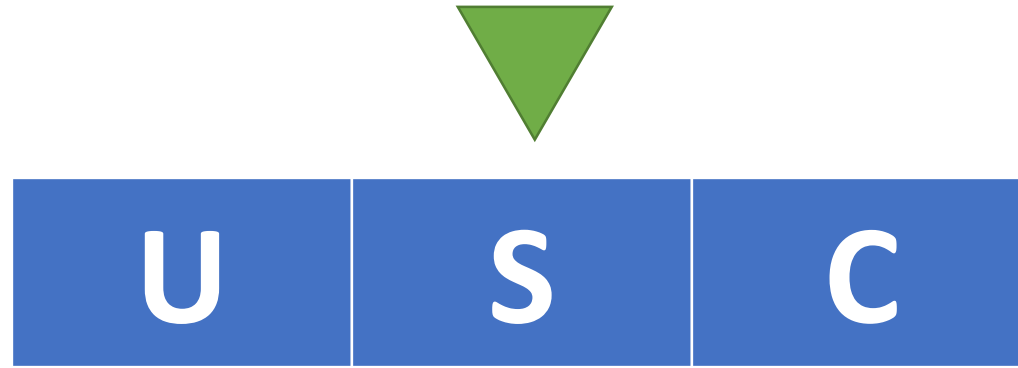
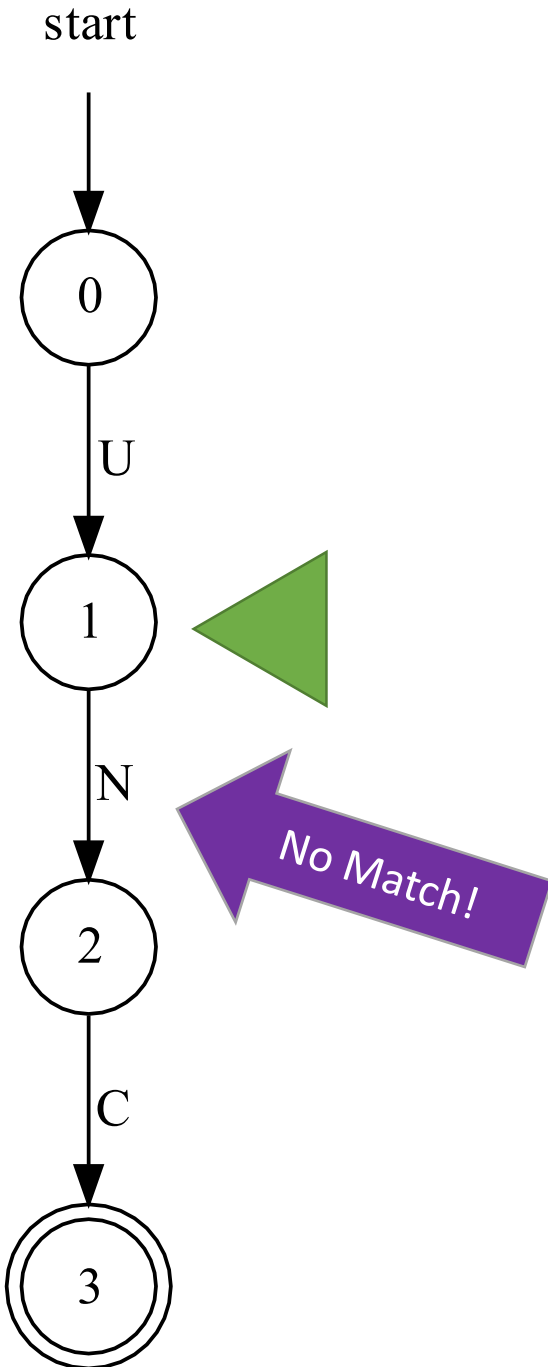
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character?

Yes: Transition to the next state.

No: Not accepting! Failure to match input against diagram.

Tracing a Transition Diagram



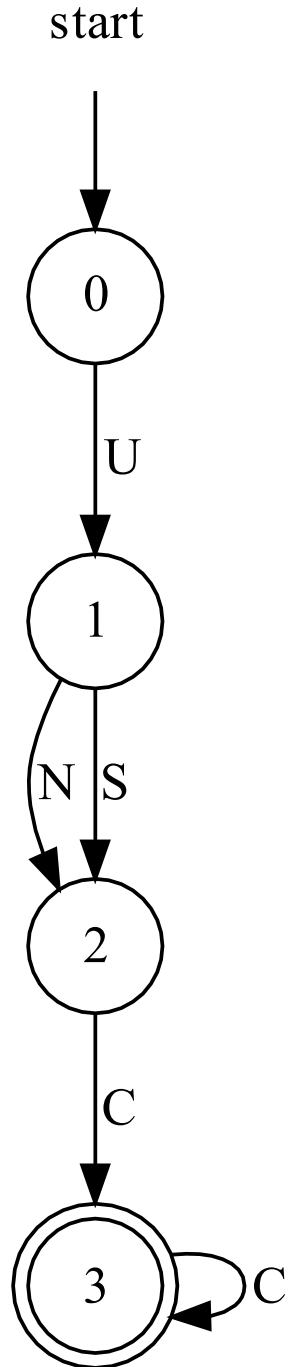
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character?

Yes: Transition to the next state.

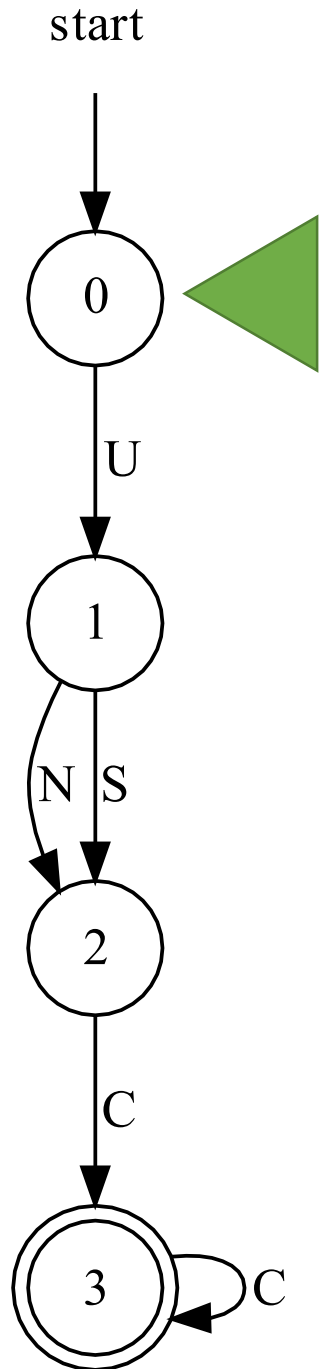
No: Not accepting! Failure to match input against diagram.

Tracing a Transition Diagram



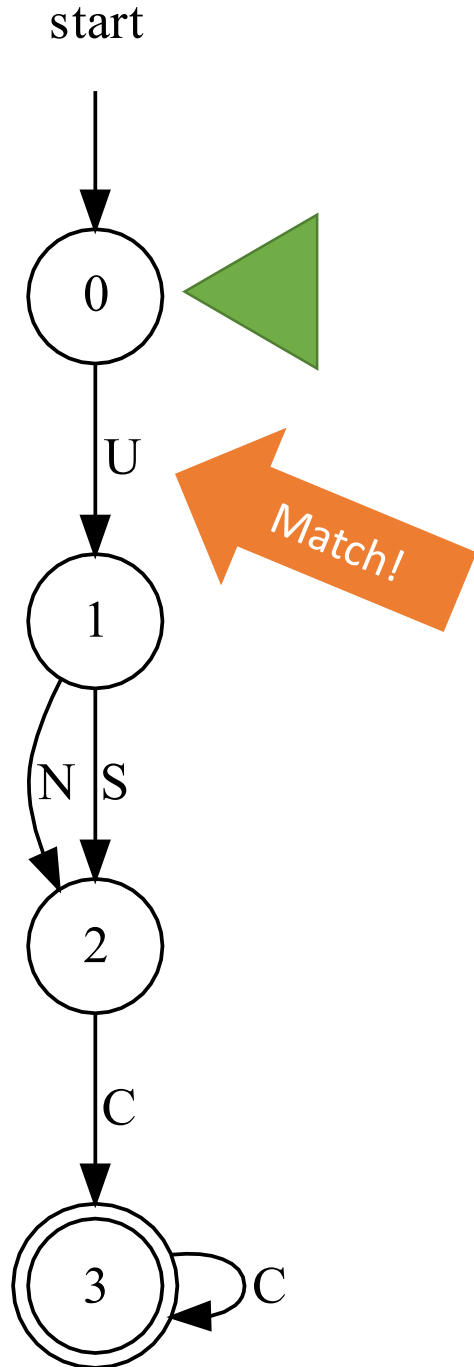
- Transition Diagram: Left
- Input String: "UNCC"
- When processing the input using the transition diagram, does it end in the accepting state 3?

Tracing a Transition Diagram



Before any input is considered, begin in the starting state.

Tracing a Transition Diagram



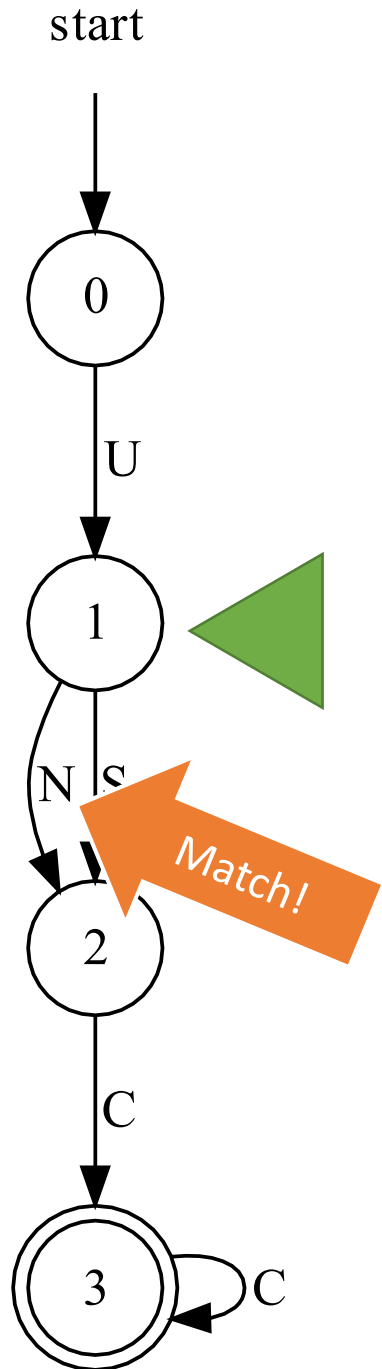
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character?

Yes: Transition to the next state.

No: Not accepting! Failure to match input against diagram.

Tracing a Transition Diagram



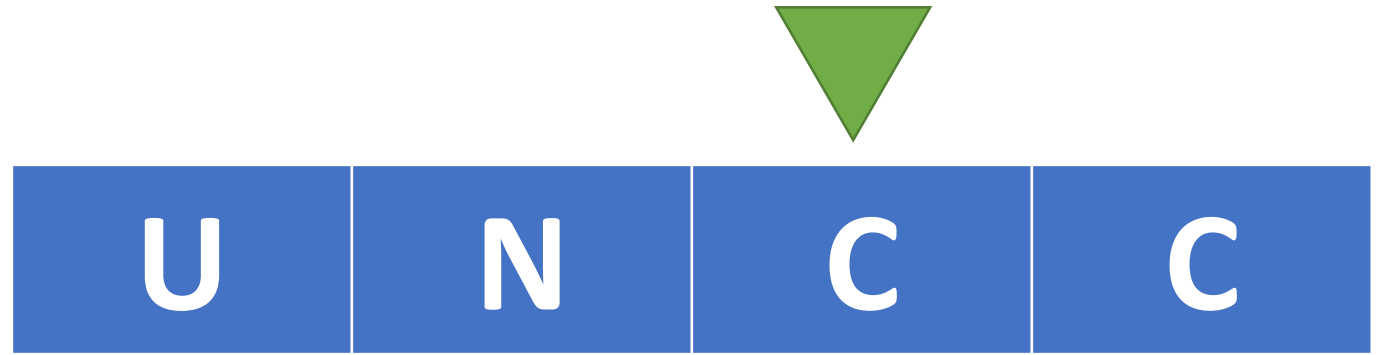
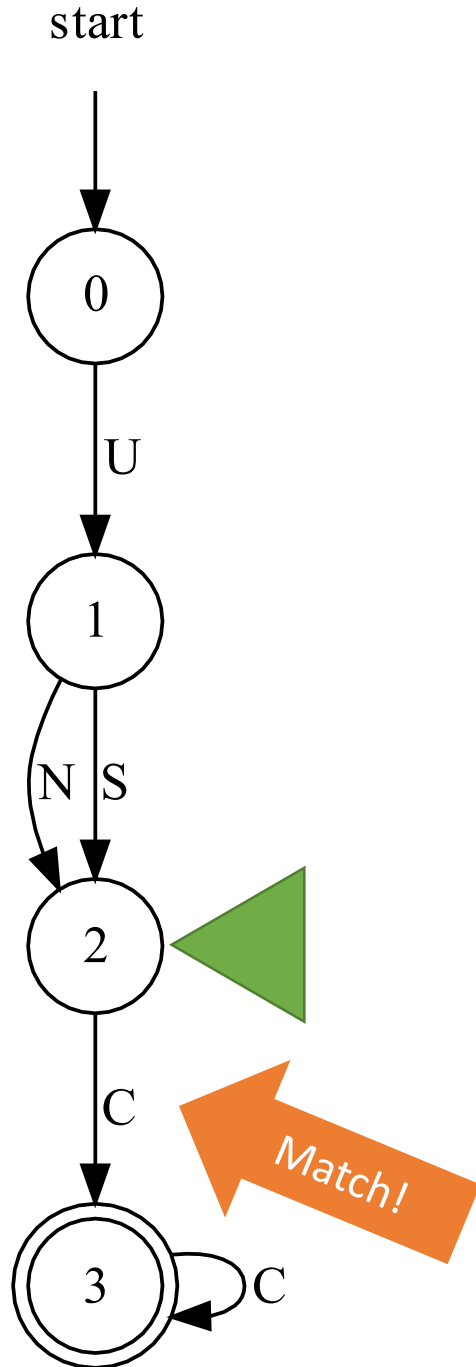
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character?

Yes: Transition to the next state.

No: Not accepting! Failure to match input against diagram.

Tracing a Transition Diagram



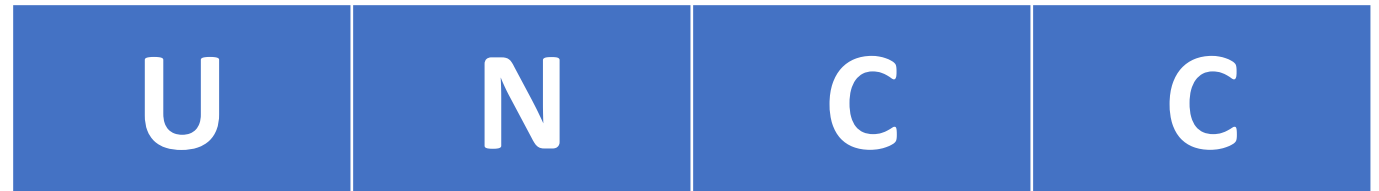
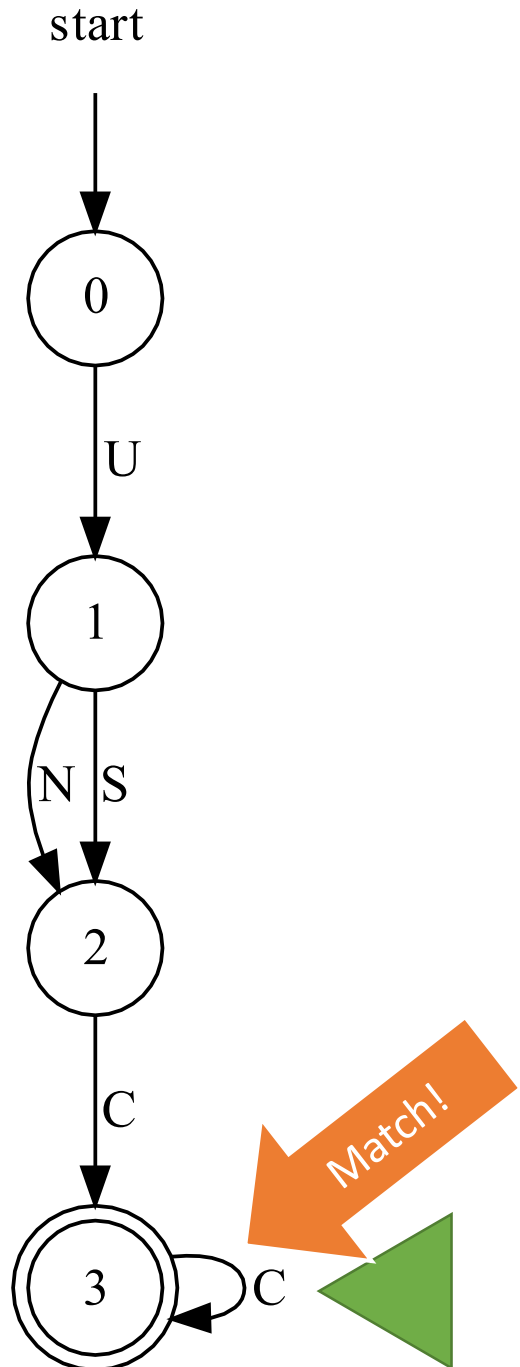
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character?

Yes: Transition to the next state.

No: Not accepting! Failure to match input against diagram.

Tracing a Transition Diagram




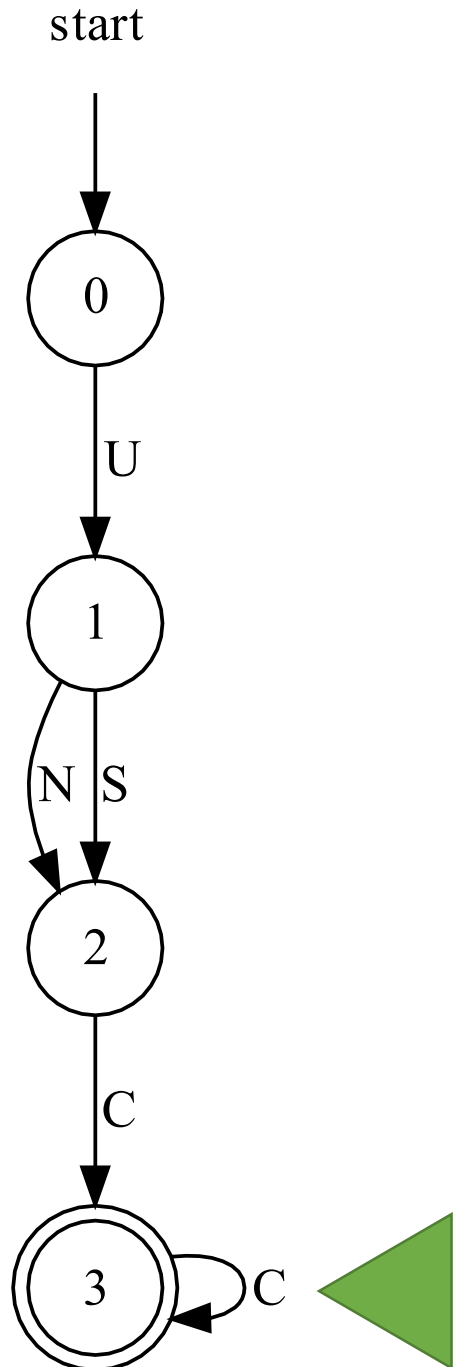
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character?

Yes: Transition to the next state. (Can be same state!)

No: Not accepting! Failure to match input against diagram.

Tracing a Transition Diagram



U	N	C	C
---	---	---	---

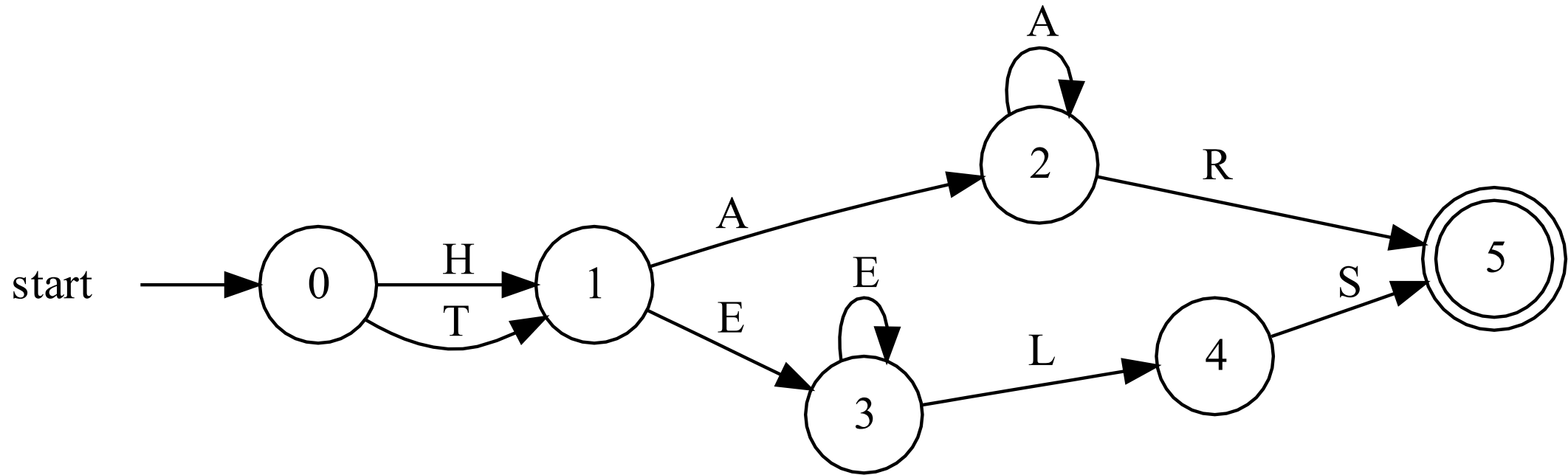
End of input?

Is the current state a *final / accepting* state?

Yes: Match!

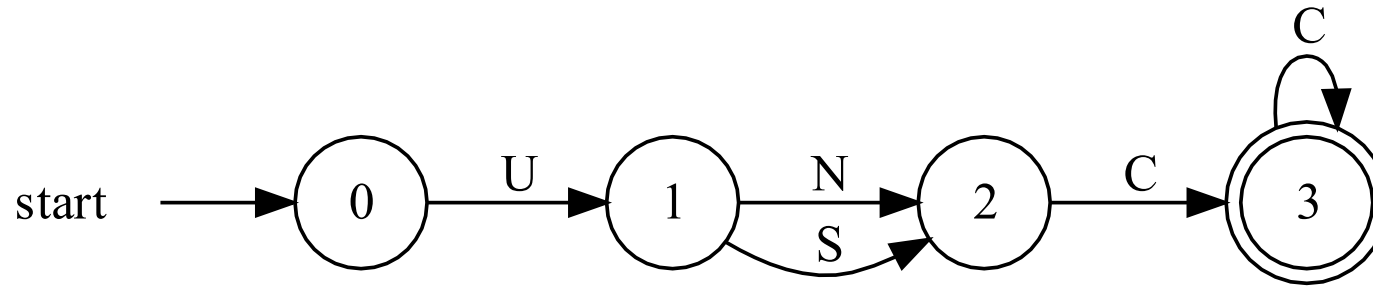
No: Not a match :(





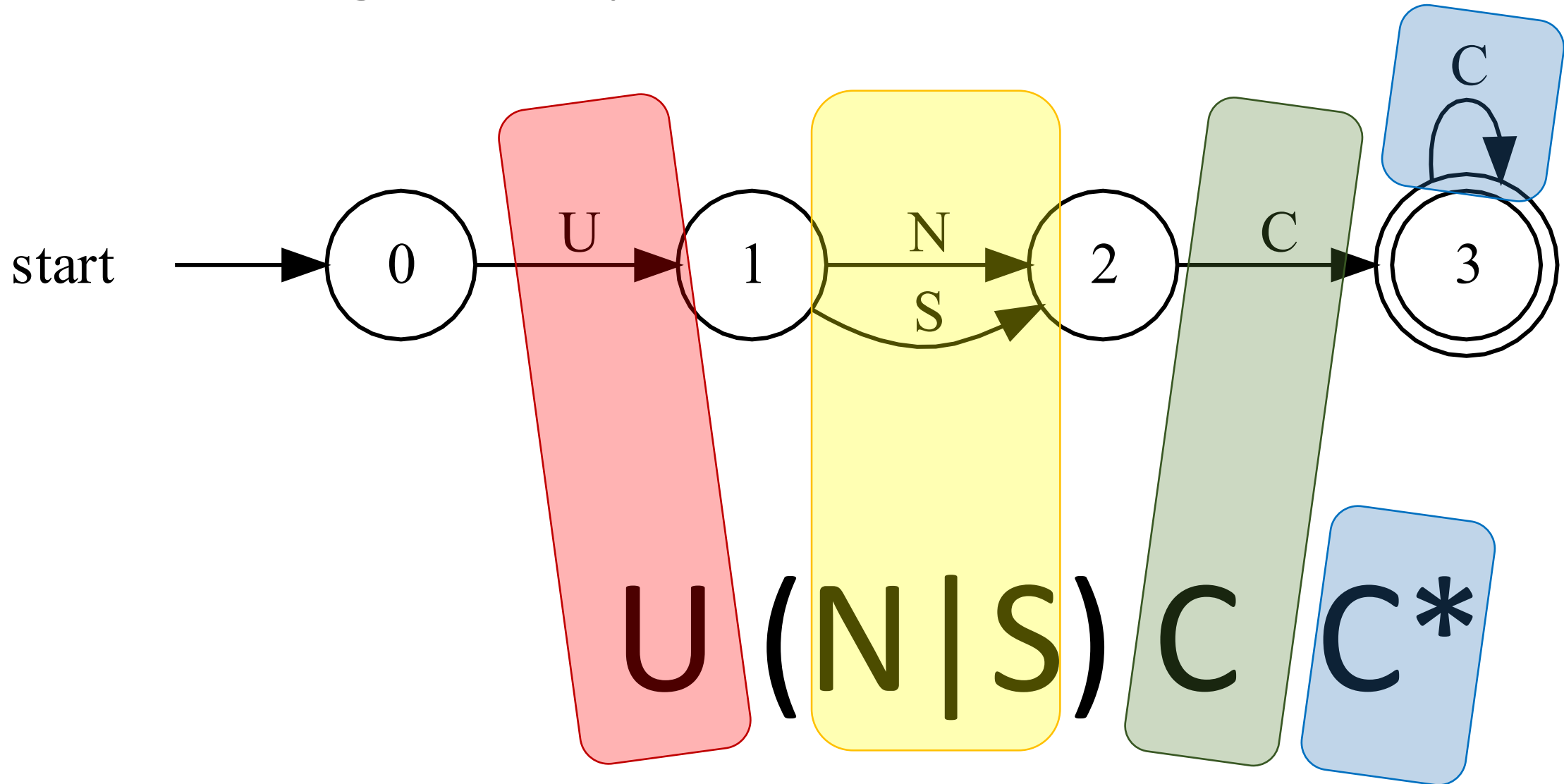
- Generate two strings that fail to match the DFA above.
 1. The first string should be at least length 3 and fail in state number 2.
 2. The second string should be at least length 4 and fail in state number 4.
- Submit your strings on [PollEv.com/compunc](https://pollev.com/compunc)

Transition Diagrams are Visual Representations of Deterministic Finite Automata (DFA) - *Machines!*



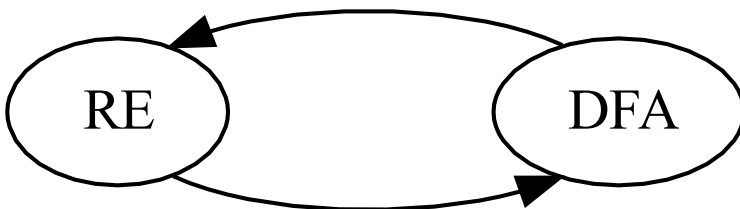
- DFA are formally specified with a set theoretical 5-tuple $M = (Q, \Sigma, \Delta, q_0, F)$
 - Q is a set of states (above 0, 1, 2, 3) - our nodes
 - Σ is the alphabet (above U, N, S, C) - set of all labels
 - Δ is a transition function $(Q, \Sigma) \rightarrow Q$ - labeled edges
 - q_0 is the start state (above 0)
 - F is the set of accepting states (above 3) - double circled nodes
- Since DFA are specified in discrete terms (sets, functions, alphabets, strings) you're:
 1. Able to represent the states and transition functions as a graph data structure in memory
 2. Able to process input string algorithmically as a program
 3. Able to rigorously prove soundness and properties (COMP455 - Models of Language)

DFA to Regular Expression



Equivalence of Regular Expression (RE) and DFA

- Stephen Kleene proved in 1951 that Regular Expressions and Deterministic Finite Automata are in the same equivalence class.
 - Everything you can express in one you can express in the other and vice-versa!
 - Formal proofs such as this are the emphasis of COMP455.



U. S. AIR FORCE
PROJECT RAND
RESEARCH MEMORANDUM

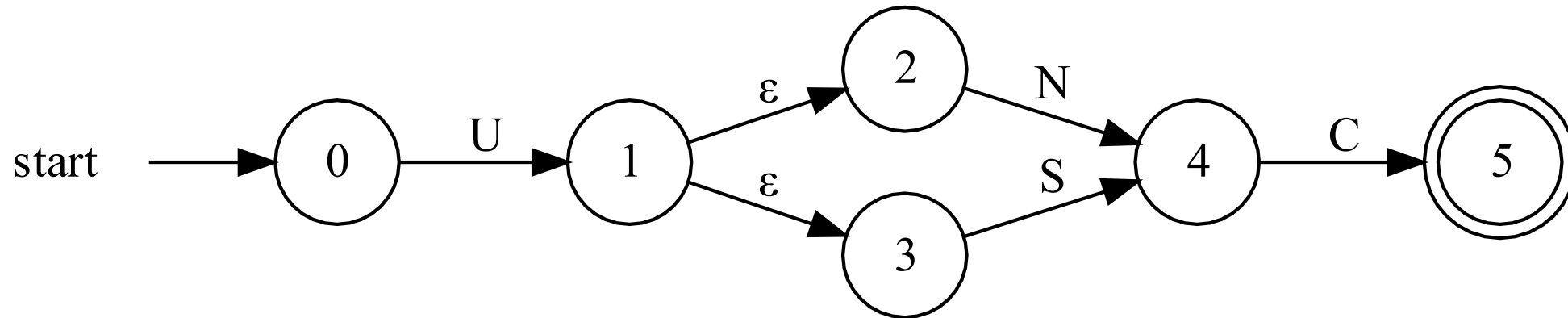
**REPRESENTATION OF EVENTS IN NERVE NETS AND
FINITE AUTOMATA**

S. C. Kleene

RM-704

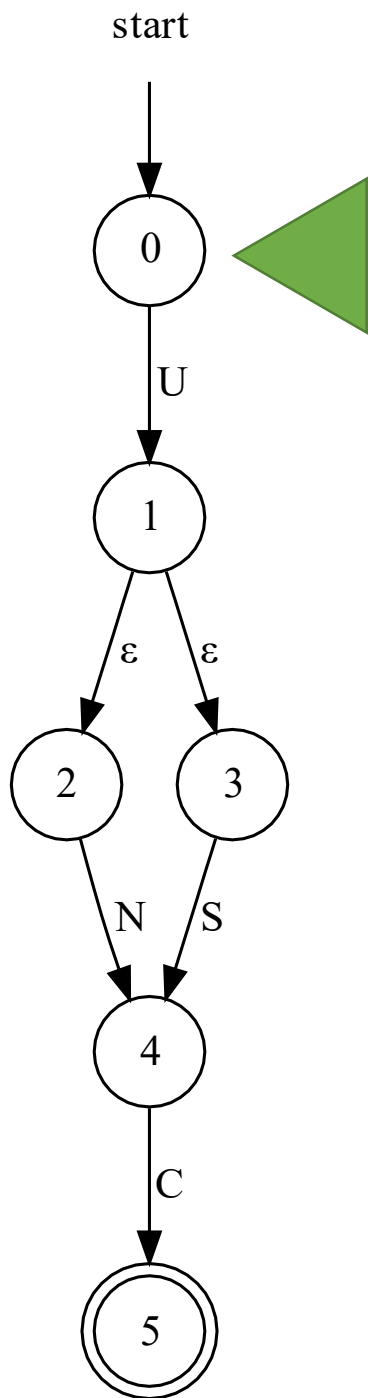
15 December 1951

ϵ -Transition Diagrams



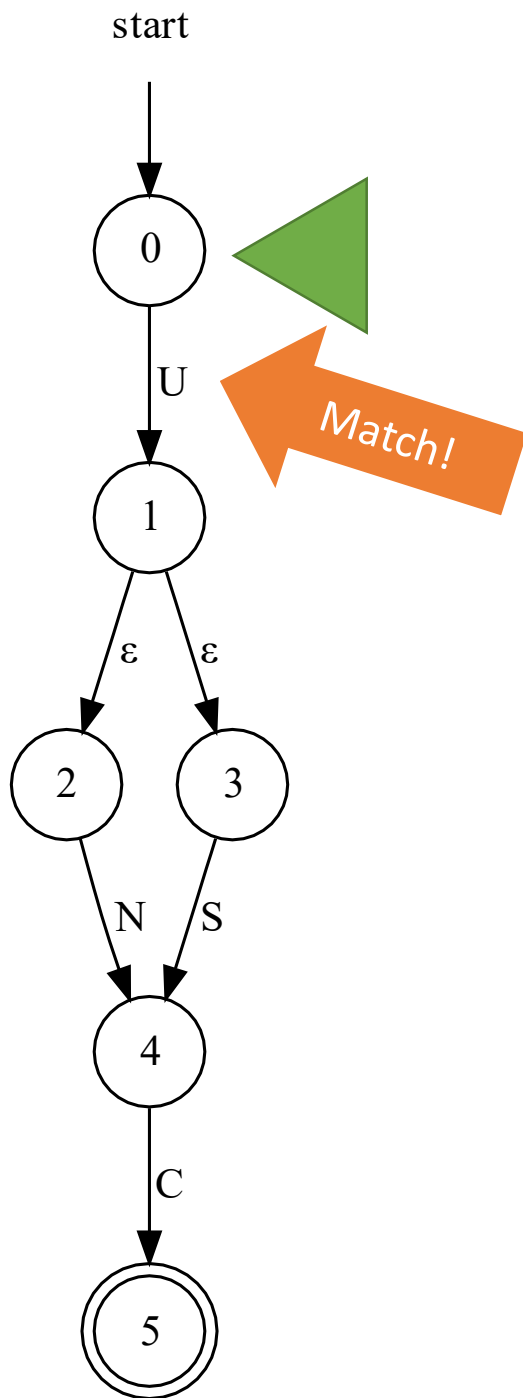
- ϵ -transitions are edges that match the *empty string*
- This is a funny concept that simply means as soon as you reach a state with an ϵ -transition you follow it immediately
- Thus, as you're processing input you may be in multiple states *at the same time!*
- By introducing **epsilon**-transitions our state machines become non-deterministic

Tracing an ϵ -Transition Diagram



Before any input is considered, begin in the starting state.

Tracing an ϵ -Transition Diagram



Consider the next input character.

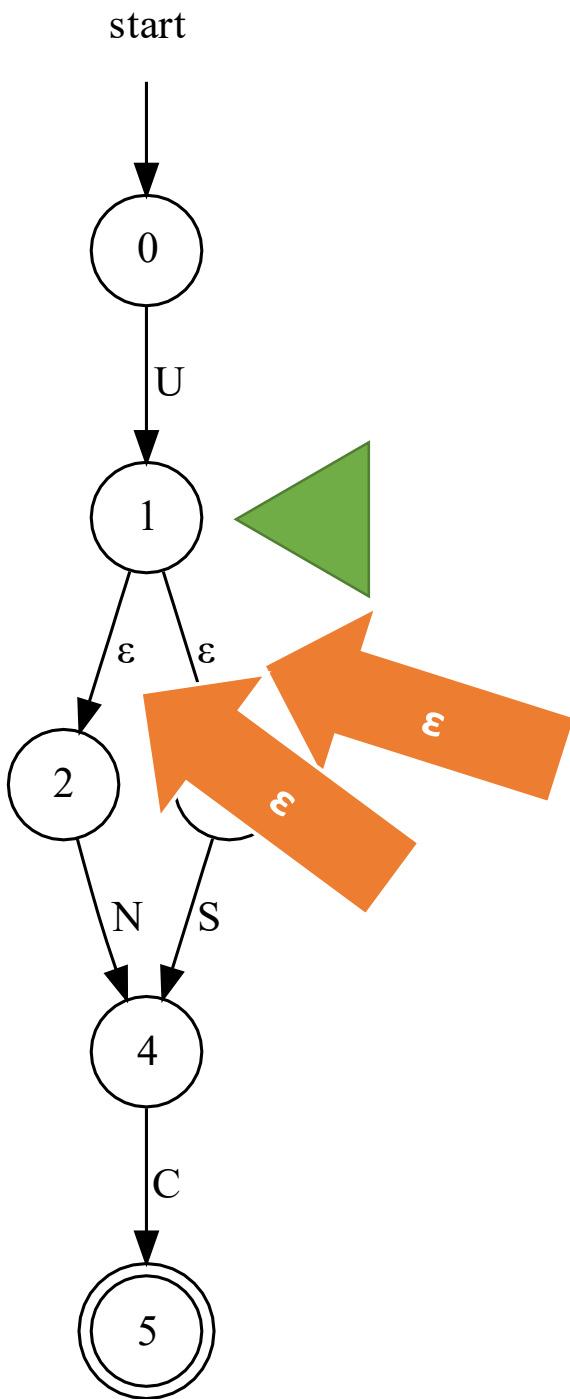
Is there a directed edge from the current state that is labelled with this character?

Or is there an ϵ -transition?

Yes: Transition to the next state.

No: Not accepting! Remove possible state from diagram and if no states remain, fail to match.

Tracing an ϵ -Transition Diagram

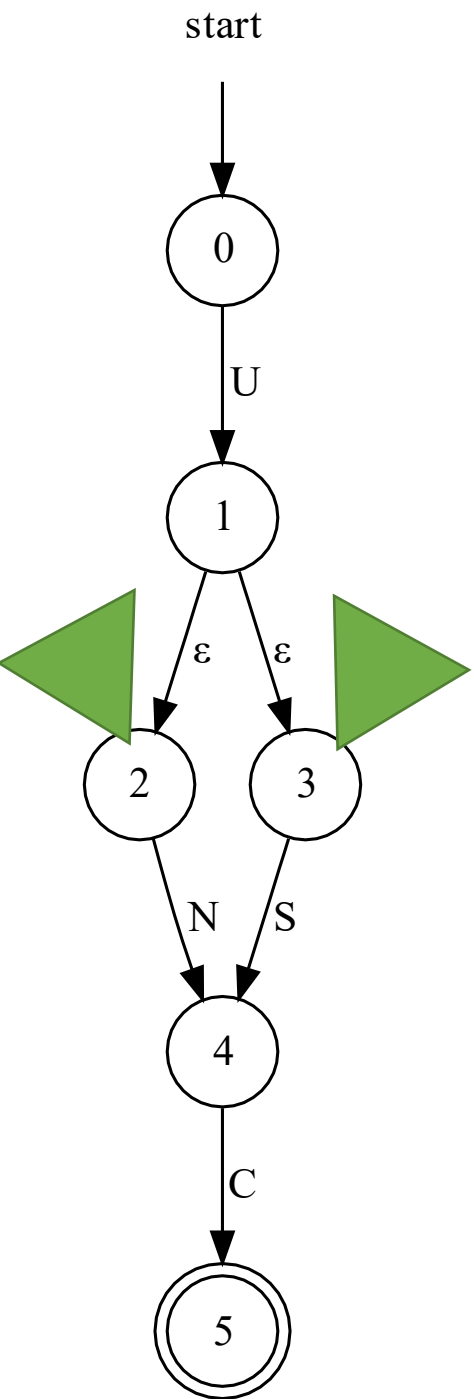


Different from tracing a DFA:

After a transition, are there any **ϵ -transitions** from the current state to other states?

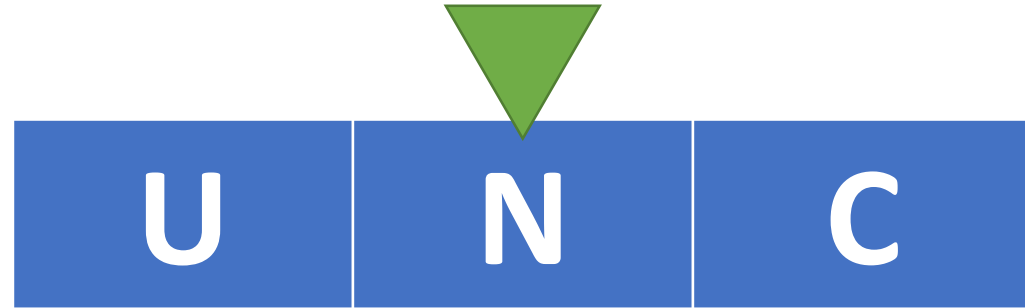
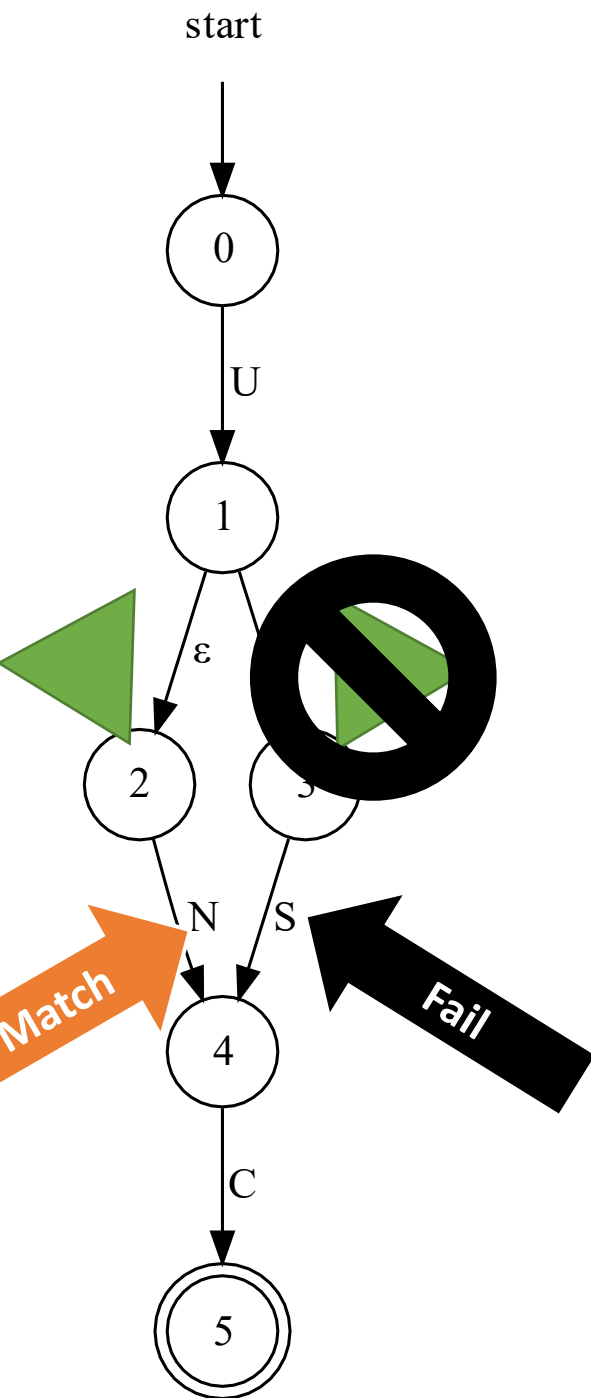
Yes! Follow each of those states

Tracing an ϵ -Transition Diagram



- Notice you're in *two states simultaneously!*
 - Currently both 2 and 3.
- This is where *non-determinism* arises.
 - You can't actually know which (if any) are leading you toward an accepting state and your machine is in multiple states at the same time.
- From here we resume normal algo *from all active states*.

Tracing an ϵ -Transition Diagram



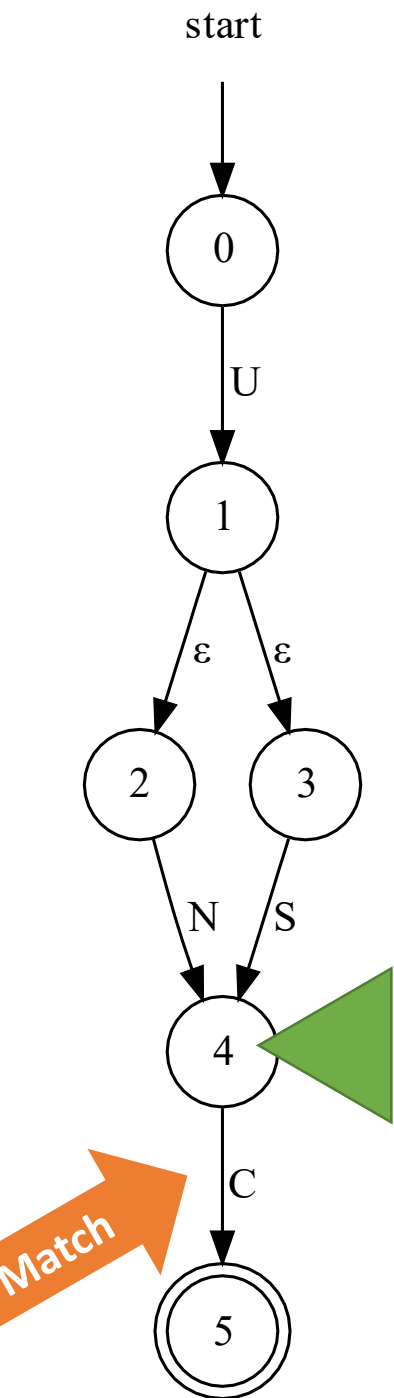
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character? Or is there an ϵ -transition?

Yes: Transition to the next state.

No: Not accepting! Remove possible state from diagram and if no states remain, fail to match.

Tracing an ϵ -Transition Diagram



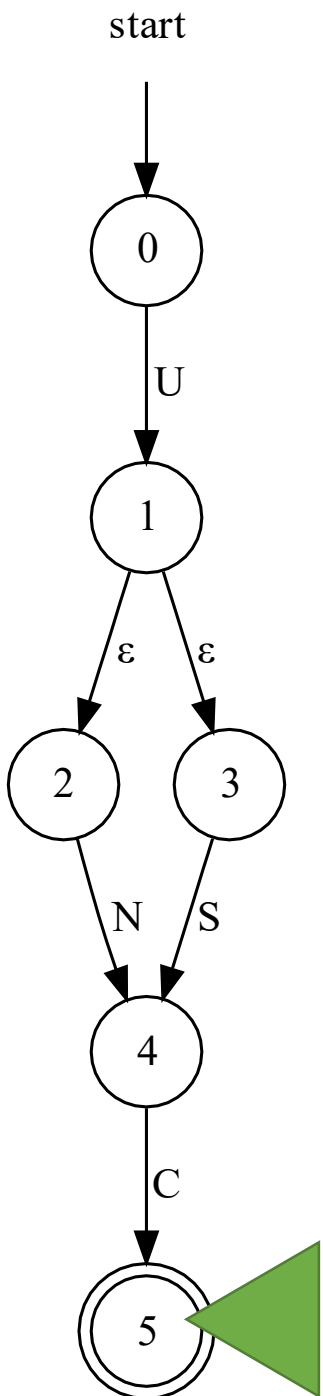
Consider the next input character.

Is there a directed edge from the current state that is labelled with this character? Or is there an ϵ -transition?

Yes: Transition to the next state.

No: Not accepting! Failure to match input against diagram.

Tracing an ϵ -Transition Diagram



U N C

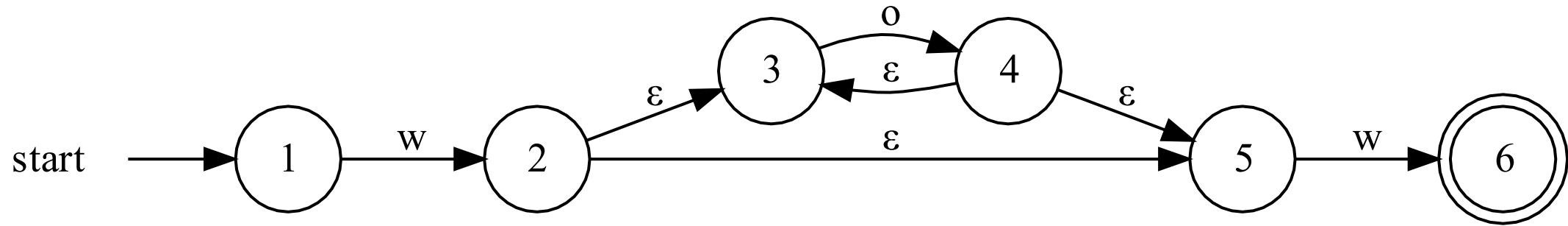
End of input?

Is *any* state in a *final / accepting* state?

Yes: Match!

No: Not a match :(

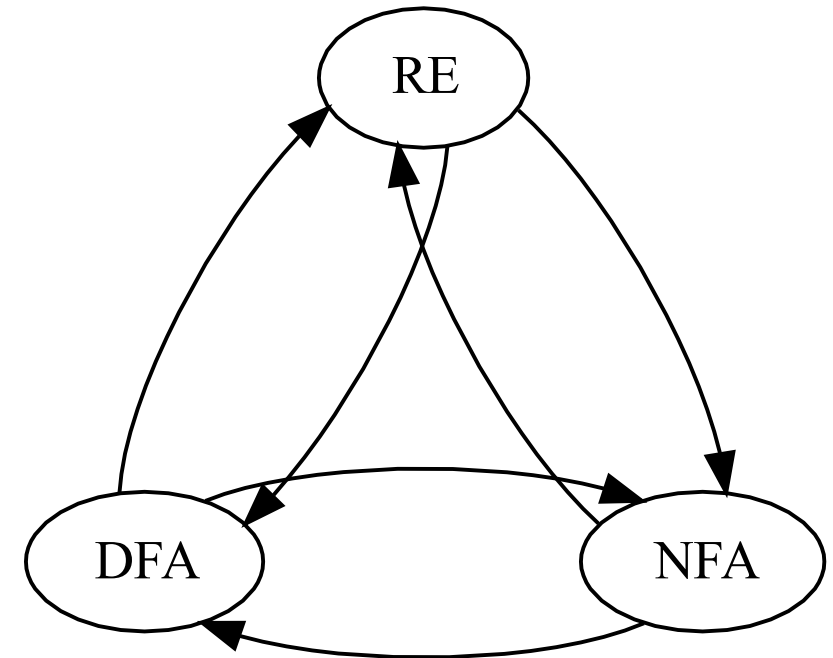
Generate a string of length 4 accepted
by this **ϵ -Transition** Diagram



Done? Submit it on [PollEv.com/compunc](https://pollev.com/compunc)

Equivalence of DFA and NFA ... *and RE*

- Rabin and Scott introduced and proved non-deterministic finite automata are equivalent to deterministic finite automata in 1959.
 - Everything you can express in one you can express in the other and vice-versa!
 - Therefore regular expressions and non-deterministic finite automata are equivalent, as well!
- This is important because NFAs are easily constructed from Regular Expressions



Thompson's RegEx -> NFA Construction Algo

- Ken Thompson, (aka Gandolf the Grey) published the foundational ideas behind **grep** in 1968.

regular expressions. The second stage converts the regular expression to reverse Polish form. The third stage is the object code producer. The first two stages are straightforward and are not discussed. The third stage expects a syntactically correct, reverse Polish regular expression.

The regular expression $a(b \mid c)^*d$ will be carried through as an example. This expression is translated into $abc \mid * \cdot d \cdot$ by the first two stages. A functional description of the

The heart of the third stage is a pushdown stack. Each entry in the pushdown stack is a pointer to the compiled code of an operand. When a binary operator (“|” or “.”) is compiled, the top (most recent) two entries on the stack are combined and a resultant pointer for the operation replaces the two stack entries. The result of the binary operator is then available as an operand in another operation. Similarly, a unary operator (“*”) operates on the top entry of the stack and creates an operand to replace that entry. When the entire regular expression is compiled, there is just one entry in the stack, and that is a pointer to the code for the regular expression.

Thompson's Construction: Preview

$a(b \mid c)*d \longrightarrow \boxed{abc \mid * \cdot d \cdot}$

Classic Time-Space Trade-off of DFA vs. NFA

	Representation (Space)	Simulation (Time)
NFA	$O(m)$	$O(m^2n)$
DFA	$O(2^m)$	$O(n)$

Where **m** is number of states in an NFA representation and **n** is length of input string.