Introduction to Theoretical Computer Science

Lecture 2: Regular Languages

Dr. Liam O'Connor

University of Edinburgh
Semester 1, 2023/2024

Recall..

DFAs, NFAs and ε-NFAs all recognise the same class of languages, called the *regular languages*. They are equal in expressive power, although some representations (NFAs) are more compact than others (DFAs).

Closure Properties

Definition

The *union* of two languages L_1 and L_2, written $L_1 \cup L_2$, is the language that includes all strings of L_1 and all strings of L_2.

Are the regular languages *closed* under union?

Closure Properties

Definition

The *union* of two languages L_1 and L_2, written $L_1 \cup L_2$, is the language that includes all strings of L_1 and all strings of L_2.

Are the regular languages *closed* under union?

That is, if we have two regular languages L_1 and L_2, is $L_1 \cup L_2$ also regular?
Exercise: Prove this.

Closure Properties

Definition

The *sequential composition* of two languages L_1 and L_2, written $L_1 L_2$, is the language of strings that consist of a string in L_1 followed by a string in L_2.

$$L_1 L_2 = \{vw \mid v \in L_1, w \in L_2\}$$

Are the regular languages *closed* under sequential composition?

Closure Properties

Definition

The *sequential composition* of two languages L_1 and L_2, written $L_1 L_2$, is the language of strings that consist of a string in L_1 followed by a string in L_2.

$$L_1 L_2 = \{vw \mid v \in L_1, w \in L_2\}$$

Are the regular languages *closed* under sequential composition?

That is, if we have two regular languages L_1 and L_2, is $L_1 L_2$ also regular?
Exercise: Prove this.

Closure Properties

Notation

Similarly to arithmetic, define L^0 as $\{\varepsilon\}$ and $L^{n+1} = LL^n$.

Definition

The *Kleene closure* of a language L, written L^*, is the language of strings that consist wholly of zero or more strings in L.

$$L^* = \bigcup_{i \in \mathbb{N}} L^i$$

(n.b: in computer science, $0 \in \mathbb{N}$)

Are the regular languages *closed* under Kleene closure?

Exercise: Prove this.

Regular Expressions

Regular expressions are an algebraic notation for regular languages. Many of you will have already used (some variant of) regular expressions in your text editors.

Syntax	Semantics
a	$[\![a]\!] = \{a\} \quad (a \in \Sigma)$
$\varnothing$	$[\![\varnothing]\!] = \varnothing$
ε	$[\![\varepsilon]\!] = \{\varepsilon\}$

Regular Expressions

Regular expressions are an algebraic notation for regular languages. Many of you will have already used (some variant of) regular expressions in your text editors.

Syntax	**Semantics**			
a	$[\![a]\!]$	$=$	$\{a\}$	$(a \in \Sigma)$
$\varnothing$	$[\![\varnothing]\!]$	$=$	$\varnothing$	
ε	$[\![\varepsilon]\!]$	$=$	$\{\varepsilon\}$	
$R_1 \cup R_2$	$[\![R_1 \cup R_2]\!]$	$=$	$[\![R_1]\!] \cup [\![R_2]\!]$	

Regular Expressions

Regular expressions are an algebraic notation for regular languages. Many of you will have already used (some variant of) regular expressions in your text editors.

Syntax	**Semantics**
a	$[\![a]\!] = \{a\} \quad (a \in \Sigma)$
$\varnothing$	$[\![\varnothing]\!] = \varnothing$
ε	$[\![\varepsilon]\!] = \{\varepsilon\}$
$R_1 \cup R_2$	$[\![R_1 \cup R_2]\!] = [\![R_1]\!] \cup [\![R_2]\!]$
$R_1 \circ R_2$	$[\![R_1 \circ R_2]\!] = [\![R_1]\!][\![R_2]\!]$

Regular Expressions

Regular expressions are an algebraic notation for regular languages. Many of you will have already used (some variant of) regular expressions in your text editors.

Syntax	Semantics
a	$[\![a]\!] = \{a\} \quad (a \in \Sigma)$
$\varnothing$	$[\![\varnothing]\!] = \varnothing$
ε	$[\![\varepsilon]\!] = \{\varepsilon\}$
$R_1 \cup R_2$	$[\![R_1 \cup R_2]\!] = [\![R_1]\!] \cup [\![R_2]\!]$
$R_1 \circ R_2$	$[\![R_1 \circ R_2]\!] = [\![R_1]\!][\![R_2]\!]$
$R*$	$[\![R*]\!] = [\![R]\!]^*$

Regular Expressions

The notation used for regexes here may differ from the "regular" expressions you may have seen in text editors. Please note that sometimes these editors contain extensions that recognise non-regular languages, so intuitions from text editors may not apply here.

Questions

- How do we write "at least one `0`"? What about "at least one `0` and at least one `1`?"

Regular Expressions

The notation used for regexes here may differ from the "regular" expressions you may have seen in text editors. Please note that sometimes these editors contain extensions that recognise non-regular languages, so intuitions from text editors may not apply here.

Questions

- How do we write "at least one 0"? What about "at least one 0 and at least one 1?"
- How do we write $R+ = R^1 \cup R^2 \cup R_3 \cup \cdots$ using existing operators?

Regular Expressions

The notation used for regexes here may differ from the "regular" expressions you may have seen in text editors. Please note that sometimes these editors contain extensions that recognise non-regular languages, so intuitions from text editors may not apply here.

Questions

- How do we write "at least one 0"? What about "at least one 0 and at least one 1?"
- How do we write $R+ = R^1 \cup R^2 \cup R_3 \cup \cdots$ using existing operators?
- How do we write $R?$, the *optional* R, using existing operators?

Regular Expressions vs Finite Automata

Regular expressions *exactly characterise* the regular languages, just as finite automata do. This means that every regular language can be represented as a regular expression.

Regular Expressions vs Finite Automata

Regular expressions *exactly characterise* the regular languages, just as finite automata do. This means that every regular language can be represented as a regular expression.

How do we prove this?

Regular Expressions vs Finite Automata

Regular expressions *exactly characterise* the regular languages, just as finite automata do. This means that every regular language can be represented as a regular expression.

How do we prove this?

- **RE** $\rightarrow$ **DFA** – apply the constructions used in our closure proofs, then the subset construction.

Regular Expressions vs Finite Automata

Regular expressions *exactly characterise* the regular languages, just as finite automata do. This means that every regular language can be represented as a regular expression.

How do we prove this?

- **RE** $\rightarrow$ **DFA** – apply the constructions used in our closure proofs, then the subset construction.
- **DFA** $\rightarrow$ **RE** – convert to a *generalised NFA*, then reduce to a single transition.

Regular Expressions vs Finite Automata

Regular expressions *exactly characterise* the regular languages, just as finite automata do. This means that every regular language can be represented as a regular expression.

How do we prove this?

- **RE** $\rightarrow$ **DFA** – apply the constructions used in our closure proofs, then the subset construction.
- **DFA** $\rightarrow$ **RE** – convert to a *generalised NFA*, then reduce to a single transition.

A note

The DFAs we get from our **RE** $\rightarrow$ **DFA** translation are not very space-efficient. Most implementations use more advanced techniques to minimise the DFA.

Generalised NFAs

Definition

A *generalised NFA*, or GNFA, is an NFA where:

- Transitions have regular expressions on them instead of symbols.

Generalised NFAs

Definition

A *generalised NFA*, or GNFA, is an NFA where:

- Transitions have regular expressions on them instead of symbols.
- There is only one unique final state.

Generalised NFAs

Definition

A *generalised NFA*, or GNFA, is an NFA where:

- Transitions have regular expressions on them instead of symbols.
- There is only one unique final state.
- The transition relation is *full*, except that the initial state has no incoming transitions, and the final state has no outgoing transitions.

Generalised NFAs

Definition

A *generalised NFA*, or GNFA, is an NFA where:

- Transitions have regular expressions on them instead of symbols.
- There is only one unique final state.
- The transition relation is *full*, except that the initial state has no incoming transitions, and the final state has no outgoing transitions.

(n.b: transitions can be labelled with $\emptyset$)

What do we need to do to convert a DFA to a GNFA?

DFA to GNFA

1. Add a new start state, connect via ε moves to the old one.

DFA to GNFA

1. Add a new start state, connect via ε moves to the old one.
2. Add a new final state, connect via ε moves from the old final state(s).

DFA to GNFA

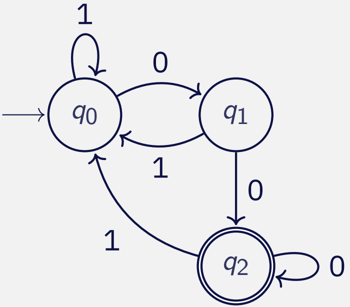
1. Add a new start state, connect via ε moves to the old one.
2. Add a new final state, connect via ε moves from the old final state(s).
3. If two states q_0 and q_1 have two transitions between them $q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{b} q_1$, replace them with $q_0 \xrightarrow{a \cup b} q_1$.

DFA to GNFA

1. Add a new start state, connect via ε moves to the old one.
2. Add a new final state, connect via ε moves from the old final state(s).
3. If two states q_0 and q_1 have two transitions between them $q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{b} q_1$, replace them with $q_0 \xrightarrow{a \cup b} q_1$.
4. Introduce $\varnothing$-labelled transitions where needed to make the transition relation full.

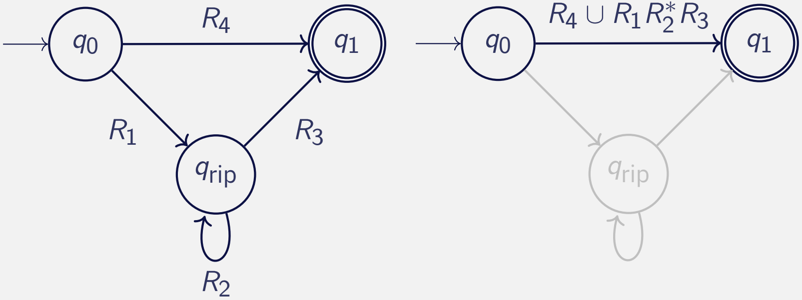
DFA to GNFA

1. Add a new start state, connect via ε moves to the old one.
2. Add a new final state, connect via ε moves from the old final state(s).
3. If two states q_0 and q_1 have two transitions between them $q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{b} q_1$, replace them with $q_0 \xrightarrow{a \cup b} q_1$.
4. Introduce $\varnothing$-labelled transitions where needed to make the transition relation full.



GNFA to RE

We will eliminate each of the inner states of the GNFA one by one. When all of them are gone, only the initial and final state will remain, with one transition between them. The label on this transition will be our regular expression.



Exercise: Let's reduce our example to a single RE.