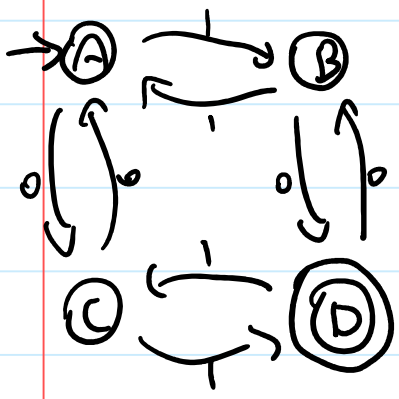
DFA

2020年9月14日　下午 10:16



5 Attributes

Q = set of all states

Σ = inputs

q_0 = start state

F = set of final states

δ = transmisition function from $Q \times \Sigma \to Q$

$Q = \{A, B, C, D\}$

$\Sigma = \{0, 1\}$

$q_0 = A$

$F = \{D\}$

could be more than 1!

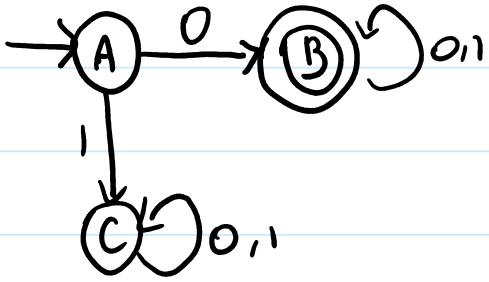
δ

	0	1
A	C	B
B	D	A
C	A	D
D	B	C

example

language

L1 = set of all strings start with '0'



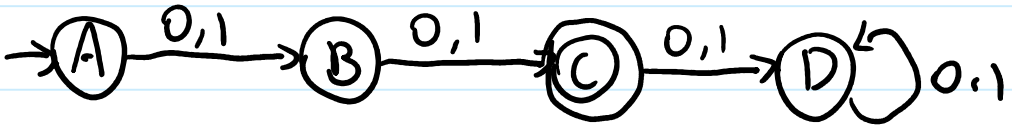
example

Construct a DFA that accepts sets of all strings over $\{0,1\}$ of length 2

solution:

$\Sigma = \{0,1\}$

$L = \{00, 01, 10, 11\}$



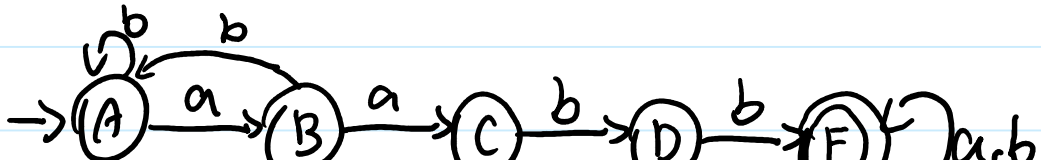
example

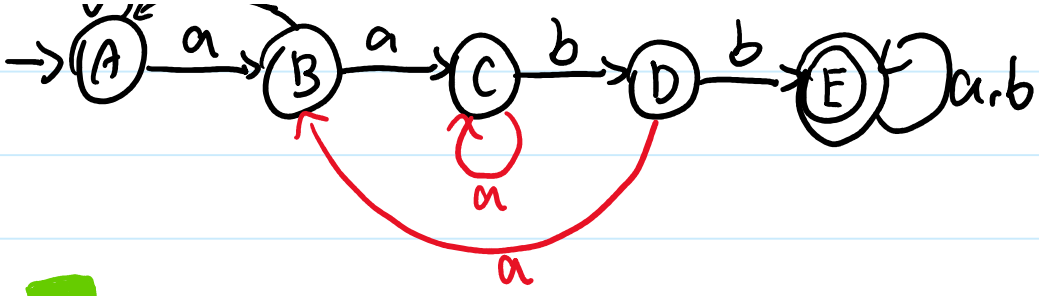
Construct a DFA that accepts all strings over $\{a,b\}$ that contains the string aabb in it.

Construct a DFA that accepts any strings over $\{a,b\}$ that does not contain the string aabb in it.

solution:

$\Sigma = \{a,b\}$

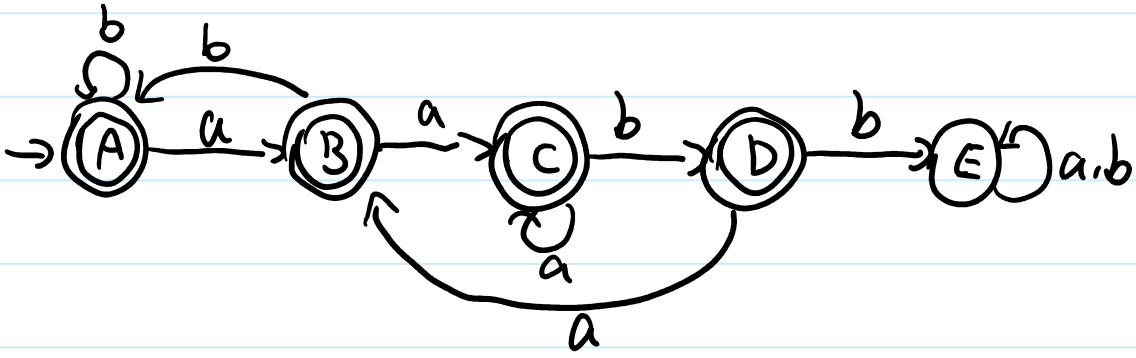




- What about part 2?

Solution: flip the states

- make the final states into non-final states
- make the non-final states into final states



Regular Language

2020年9月15日 上午 11:06

(def)

If and only if some finite state machine recognizes it

Not regular languages

- Not recognized by any FSM
- Requires memory ← memory of FSM is very limited cannot store or count strings

(example) a language that repeats abcabb

not RL because requires to store ababb into memory (?)

(example) $a^N b^N$ (same) like ab, aabb, aaabbb.

not RL because requires to count as and bs.

Operations

Union - $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$

Concatenation - $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

Star - $A^* = \{x_1 x_2 x_3 \cdots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$

(example) $A = \{pq, r\}$ $B = \{t, uv\}$

$A \cup B = \{pq, r, t, uv\}$

$A \circ B = \{pqt, pquv, rt, ruv\}$

$A^* = \{\varepsilon, pq, r, pqr, rpq, pqpq, rr, pqpqpq, rrr, \ldots\}$

ε: empty string

(Theorem)

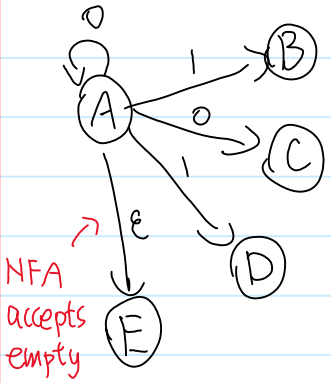
- Regular languages is closed under union

Theorem

- Regular languages is closed under union
  $LR \cup LR = LR$
- Regular languages is closed under concatenation
  $LR \circ LR = LR$

NFA

2020年9月21日　　下午 12:00



NFA accepts empty string, DFA does not.

given the current state, there could be multiple next states

The next state may be chosen at random, or all the next states may be chosen in parallel

5 Attributes : $\{Q, \Sigma, q_0, F, \delta\}$

Q = set of all states

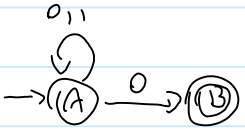
Σ = inputs

q_0 = initial state

F = set of final states

$\delta = Q \times \Sigma \rightarrow p(Q)$ (or 2^Q)

↑ power set



L = {set of strings that ends with 0}

example

$Q = \{A, B\}$

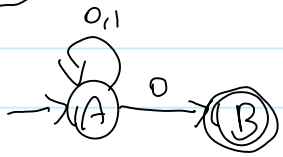
$\Sigma = \{0, 1\}$

$q_0 = A$

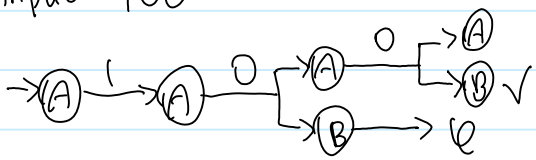
$F = \{B\}$

$A \xrightarrow{1} A, B, AB, \emptyset$ (empty state)

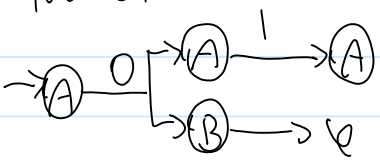
example L = {Set of all strings that end with 0}



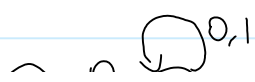
input: 100

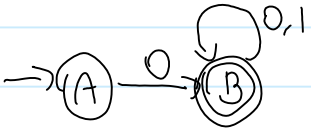


input 01



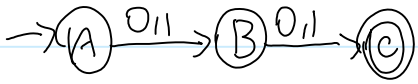
example L = {Set of all strings that start with 0}



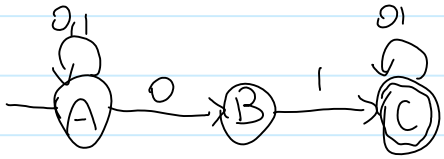


DFA need to define which state it goes to with input 1
or it will be incomplete

example NFA that accepts sets of all strings over {0,1} of length 2



example set of all strings that contain "01"

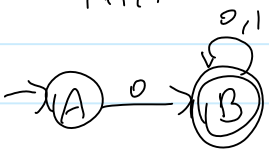


Convert NFA to DFA

2020年10月5日　上午 11:17

example set of all strings over {0,1} that starts with 0

NFA



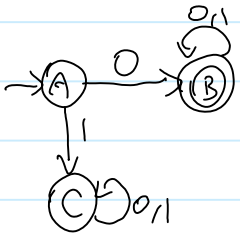
NFA table

	0	1
→ A	{B}	{Ø}
B	{B}	{B}

DFA table

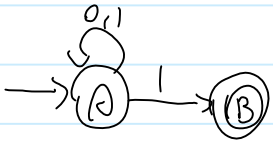
	0	1
→ A	B	C
B	B	B
C	C	C

DFA



example set of all strings over {0,1} that ends with 1

NFA



NFA table

	0	1
→ A	{A}	{A,B}
B	{Ø}	{Ø}

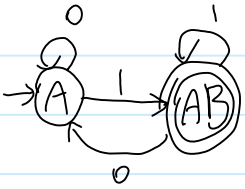
union

DFA table

	0	1
→ A	A	AB
AB	A	AB

single state

already complete



formal def of NFA

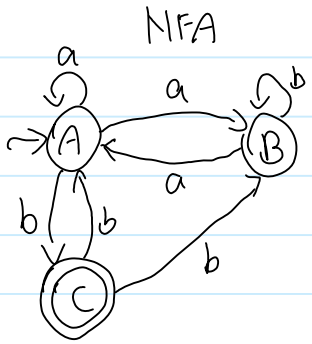
example Find the equivalent DFA for the NFA given by M=[{A,B,C},{a,b},δ,A,{C}]

states　inputs　transition func　starting state　final states

where δ is given by:

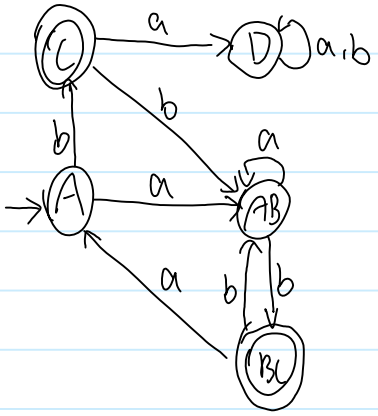
	a	b
→ A	{A,B}	{C}
B	{A}	{B}
C	{Ø}	{A,B}

NFA



DFA table

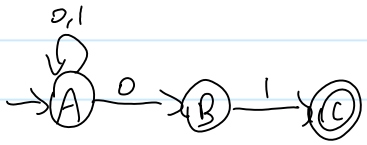
	a	b
→ A	AB	C
AB	AB	BC
C	D	AB
BC	A	AB
D	D	D



Question what does this NFA (DFA) accepts?

Example set of all strings over (0,1) that ends with '01'

NFA

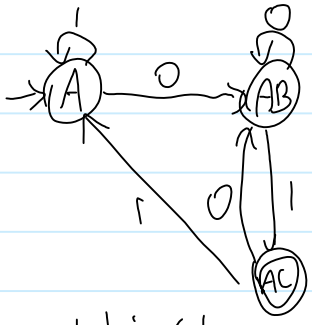


NFA table

	0	1
→ A	{A,B}	{A}
B	{∅}	{C}
(C)	{∅}	{∅}

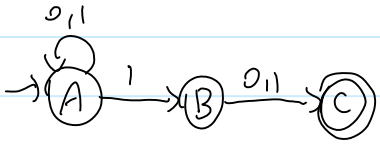
DFA table

	0	1
→ A	AB	A
AB	AB	AC
(AC)	AB	A



Example set of all strings over (0,1) that the second last symbol is '1'

NFA

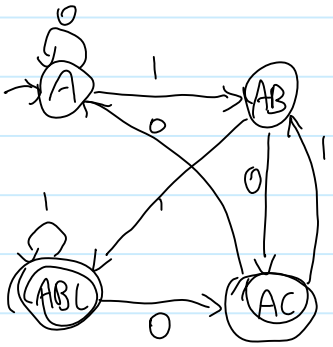


NFA table

	0	1
→ A	{A}	{A,B}
B	{C}	{C}
(C)	{∅}	{∅}

DFA table

	0	1
→ A	A	AB
AB	AC	ABC
(AC)	A	AB
(ABC)	AC	ABC



Regular Expression

2020年10月5日　下午 12:12

represents certain sets of strings in an algebraic fassion

△ Any terminal symbol i.e. symbols $\in \Sigma$ (inputs) including $\wedge$ (empty, ε) and $\emptyset$ (null) are regular expressions

△ The union of 2 regular expressions is also a regular expression $R_1, R_2 \rightarrow (R_1 + R_2)$

△ The concatenation of 2 r.e. is also a r.e. $R_1, R_2 \rightarrow (R_1 \cdot R_2)$

△ The iteration (or closure) of a r.e. is also a r.e $R \rightarrow R^*$ $a^* = \wedge, a, aa, \cdots$

△ The r.e. over Σ are precisely those obtained recursively by the application of the above rules once or several times

Example describe the following sets as r.e.

1) $\{0,1,2\}$ ← 0 or 1 or 2

$R = 0+1+2$

2) $\{\wedge, ab\}$ ← empty symbol not included, more than 1 elements (or); empty symbol along with 1 symbol, no need to +

$R = \wedge + ab$

3) $\{abb, a, b, bba\}$

$R = abb + a + b + bba$

4) $\{\wedge, 0, 00, 000, \cdots\}$ ← closure of 0

$R = 0^*$

5) $\{1, 11, 111, 1111, \cdots\}$ ← closure of 1 excluding $\wedge$

$R = 1^+$

Identities of R.E.

= why

1) $\emptyset + R = R$ (union, r.e.)

7) $RR^* = R^*R = R^+$ (at least 1 word generated by R)

1) $\emptyset + R = R$ (r.e.)

2) $\emptyset R + R\emptyset = \emptyset$ (concatenation)

3) $\varepsilon R + R\varepsilon = R$

4) $\varepsilon^* = \varepsilon$ and $\emptyset^* = \varepsilon$

5) $R + R = R$

6) $R^* R^* = R^*$ (concatenation of the closure of two r.e.)

7) $RR^* = R^*R = R^+$ (R must contain ... by R)

8) $(R^*)^* = R^*$

9) $\varepsilon + RR^* = \varepsilon + R^*R = R^*$

10) $(PQ)^*P = P(QP)^*$

11) $(P+Q)^* = (P^*Q^*)^* = (P^* + Q^*)^* = P^*(Q(P^*))^*$

12) $(P+Q)R = PR + QR$ and $R(P+Q) = RP + RQ$

Example prove that $R = QP^*$ is a solution to $R = Q + RP$

$$R = Q + RP$$
$$= Q + QP^*P$$
$$= Q(\varepsilon + P^*P)$$
$$= QP^*$$

Arden's Theorem
$R = Q + RP$
$\Downarrow$
$R = QP^*$

Example language accepting strings of length exactly 2 over $\{a, b\}$

$L = \{aa, ab, ba, bb\}$
$R = aa + bb + ba + bb$
$= a(a+b) + b(a+b)$
$= (a+b)(a+b)$ $\{a,b\}\{a,b\}$

Example language accepting strings of length atleast 2 over $\{a, b\}$

$L = \{aa, ab, ba, bb, aaa, \dots\}$
$R = (a+b)(a+b)(a+b)^*$ $\{a,b\}\{a,b\}\{a,b\}^*$

Example language accepting strings of length atmost 2 over $\{a, b\}$

$L = \{\varepsilon, a, b, aa, ab, ba, bb\}$

$R = \varepsilon + a + b + aa + ab + ba + bb$

$= (\varepsilon + a + b)(\varepsilon + a + b)$

Properties (from lecture 9)

closed under union $\cup$

L_1 and L_2 are r.l., then $L_1 \cup L_2$ is regular

but not necessarily true when reversed!!

e.g. $L_1 \cup L_2$ is r.l. $\Rightarrow$ L_1 is regular

L_1: $M_1 = (Q_1, A, \sigma_1, i_1, T_1)$

L_2 $M_2 = (Q_2, A, \sigma_2, i_2, T_2)$

$L_1 \cup L_2$: $M = (Q_1 \times Q_2, A, \sigma, (i_1, i_2), Q_1 \times T_2 \cup T_1 \times Q_2)$

$\sigma((p_1, p_2), a) = (\sigma_1(p_1, a), \sigma_2(p_2, a))$

closed under intersection $\cap$

$M = (Q_1 \times Q_2, A, \sigma, (i_1, i_2), T_1 \times T_2)$

$\sigma((i_1, i_2), w) = (\sigma_1(i_1, w), \sigma_2(i_2, w))$

closed under complement 补集

complement of L: $A^* - L$

$\bar{M} = (Q, A, \sigma, i, Q - T)$

closed under difference $-$

$L_1 - L_2$.

$M = (Q_1 \times Q_2, A, \sigma, (i_1, i_2), T_1 \times (Q_2 - T_2))$

$\sigma((i_1, i_2), w) = (\sigma(i_1, w), \sigma(i_2, w))$

also, $L_1 - L_2 = L_1 \cap (A^* - L_2) = L_1 \cap \bar{L_2}$

closed under reversal

$M = (Q, A, \sigma^{-1}, T, i)$

$\sigma^{-1}(p, a) = q$ if $\sigma(q, a) = p$

closed under closure *
closed under concatenation

Apply closure properties to prove a language is not regular

(example) prove that $\{a^m b^n c^{m-n} \mid m \geq n \geq 0\}$ is not regular

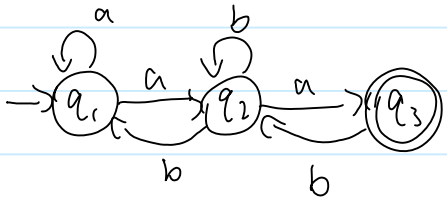
Suppose
$L_1 = \{a^m b^n c^{m-n} \mid m \geq n \geq 0\}$ is regular.
we take another regular language $L_2 = a^* b^*$

then $L_1 \cap L_2 = \{a^m b^m \mid m \geq 0\}$, which is not regular
this contradicts with the assumption that L_1 is regular
therefore, L_1 is not regular.

NFA to Regular Expression

2020年10月5日　下午 8:52

(example) find the r.e. for the following NFA



equations for states

$q_3 = q_2 a$ ①

$q_2 = q_1 a + q_2 b + q_3 b$ ②

$q_1 = \varepsilon + q_1 a + q_2 b$ ③

② substitude to ①

$q_3 = (q_1 a + q_2 b + q_3 b) a$

$q_3 = q_1 aa + q_2 ba + q_3 ba$ ④

① into ②

$q_2 = q_1 a + q_2 b + (q_2 a) b$

$q_2 = q_1 a + q_2 b + q_2 ab$

$q_2 = q_1 a + q_2 (b + ab)$

$q_2 = (q_1 a)(b + ab)^*$ ⑤ - by Arden's Theorem

⑤ into ①

$q_1 = \varepsilon + q_1 a + ((q_1 a)(b + ab)^*) b$

$q_1 = \varepsilon + q_1 (a + a(b + ab)^* b)$

$q_1 = \varepsilon ((a + a(b + ab)^* b)^*$

$q_1 = (a + a(b + ab)^* b)^*$ ⑥

final state (q_3)

$q_3 = q_2 a$

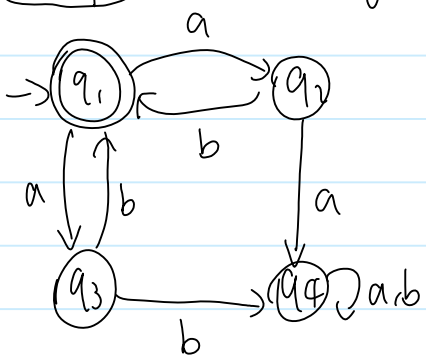
$q_3 = q_1 a (b + ab)^* a$ from ⑤

$q_3 = (a + a(b + ab)^* b)^* a(b + ab)^* a$ ← expressed completely with input symbols ⇒ required r.e.

DFA to Regular Expression

2020年10月5日　下午 9:16

Example find the regular expression for the following DFA



$q_1 = \varepsilon + q_2 b + q_3 a$ ①

$q_2 = q_1 a$ ②

$q_3 = q_1 b$ ③

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$ ④

①: $q_1 = \varepsilon + q_2 b + q_3 a$

$q_1 = \varepsilon + q_1 ab + q_1 ba$ by ② ③

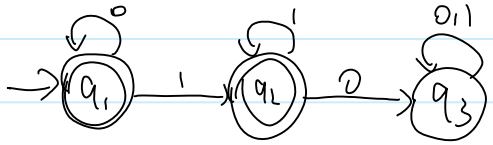
$q_1 = \varepsilon + q_1 (ab + ba)$

$q_1 = \varepsilon (ab + ba)^*$ by Arden's Theorem

$q_1 = (ab + ba)^*$ ④

↖ required regular expression

Example find the r.e. for the following DFA (multiple final states)



$q_1 = \varepsilon + q_1 0$ ①

$q_2 = q_1 1 + q_2 1$ ②

$q_3 = q_2 0 + q_3 0 + q_3 1$ ③

①: $q_1 = \varepsilon + q_1 0$

$q_1 = \varepsilon 0^*$ by Arden's Theorem

$q_1 = 0^*$ ④

② $q_2 = q_1 1 + q_2 1$

$q_2 = 0^* 1 + q_2 1$ by ④

$q_2 = 0^* 1 1^*$ by Arden's Theorem ⑤

R would be the union of both final states

$R = 0^* + 0^* 1 1^*$ by ④ ⑤

$R = 0^* + 0^*1 1^*$ by ④, ⑤

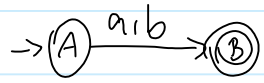
$R = 0^* (\varepsilon + 11^*)$

$R = 0^* 1^*$

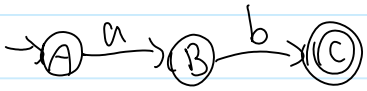
Regular Expression to Finite Automata

2020年10月5日　下午 9:39

△ Union　(a+b)

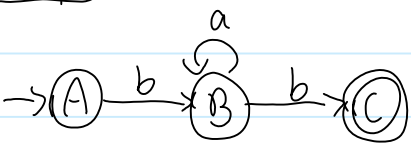


○ Concatenation　(a b)



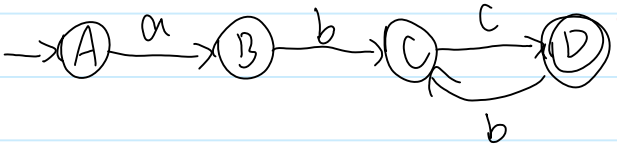
△ Closure　a^*

example　$b\,a^*\,b$

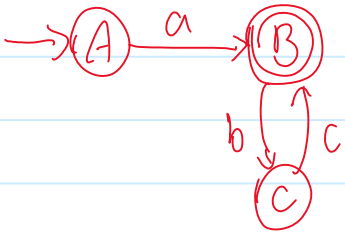


example　$(a+b)\,c$

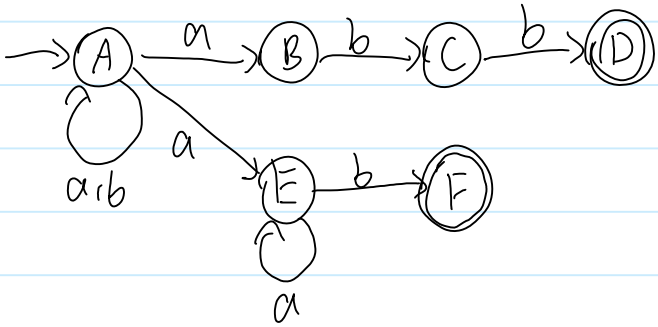
example　$a(bc)^*$



← WRONG: does not accept 'a'

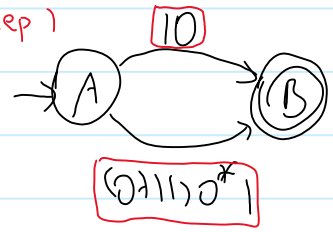


example　$(a \cup b)^*(abb \cup a^+b)$

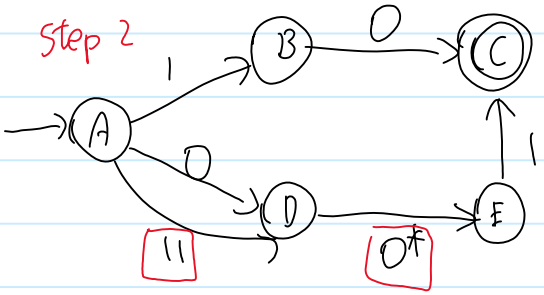


Example $10+(0+11)0^*1$

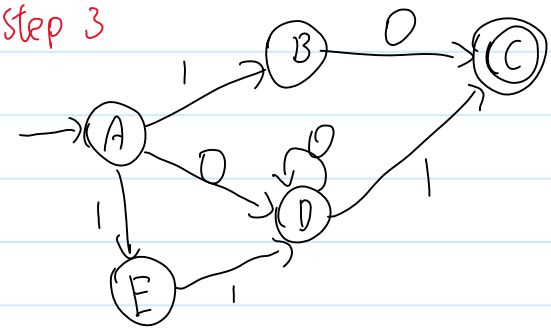
Step 1



Step 2



Step 3



Minimization of DFA

2020年10月8日　上午 11:08

minimize DFAs by combining equivalent states

two states 'A' and 'B' are said to be equivalent if

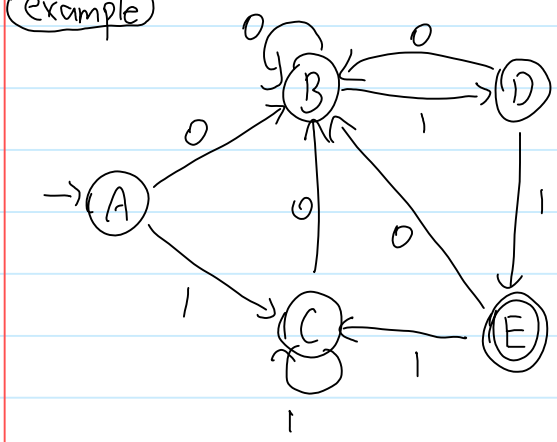
$\delta(A,x) \to F$ and $\delta(B,x) \to F$ or $\delta(A,x) \not\to F$ and $\delta(B,x) \not\to F$ where 'x' is any input string

(↑ does not go to)

(length)

- if $|x| = 0$ then A and B are said to be 0 equivalent
- $|x| = 1$ ... 1 equivalent
- $|x| = n$ ... n equivalent

example



transition table

	0	1
→A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C

0 Equivalence {A, B, C, D} {E} ($\delta(A,x) \not\to F$ and $\delta(A,x) \to F$)

(non-final states) (final states)

① A, B — same, ok / different, check whether 'c and d' are 0-equivalent, which is yes, so ok

	0	1
A	B	C
B	B	D

② C ? (with A or B) — same, ok

	0	1
A	B	C
C	B	C

③ D ? (with A or B or C) — C and E not 0-equivalent. not ok

	0	1
A	B	C
D	B	E

check from 0 equivalence

1 Equivalence {A, B, C} {D} {E}

check from 1 equivalence

	0	1
A	B	C
B	B	D

	0	1
A	B	C
C	B	C

2 Equivalence {A, C} {B} {D} {E}

	0	1
A		

same,

2 Equivalence {A, C} {B} {D} {E}

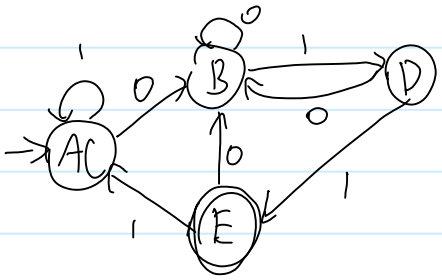
	0	1
A	D	E
C	B	C

same, STOP

3 Equivalence {A, C} {B} {D} {E}

final 4 states

	0	1
→ A	B	C
B	B	D
C	B	C
D	B	E
(E)	B	C



example

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_2	q_6
q_4	q_7	q_5
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

0 Equivalence

$\{q_0, q_1, q_3, q_4, q_5, q_6, q_7\}$ $\{q_2\}$

1 Equivalence

$\{q_0, q_4, q_6\}$

$\{q_1, q_7\}$

$\{q_3, q_5\}$ $\{q_2\}$

2 Equivalence

$\{q_0, q_4\}$ $\{q_6\}$

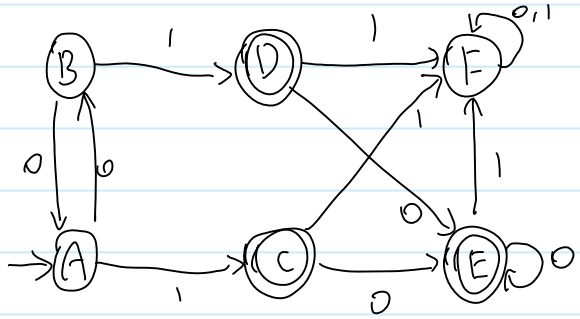
$\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

3 Equivalence

$\{q_0, q_4\}$ $\{q_6\}$ $\{q_1, q_7\}$ $\{q_3, q_5\}$ $\{q_2\}$

	0	1
→ $\{q_0, q_4\}$	$\{q_1, q_7\}$	$\{q_3, q_5\}$
$\{q_6\}$	$\{q_6\}$	$\{q_0, q_4\}$
$\{q_1, q_7\}$	$\{q_6\}$	$\{q_2\}$
$\{q_3, q_5\}$	$\{q_2\}$	$\{q_6\}$
($\{q_2\}$)		

example with more than 1 final states



1 equivalence

{A, B, F} {C, D, E}

2 equivalence

{A, B} {F} {C, D, E}

3 equivalence

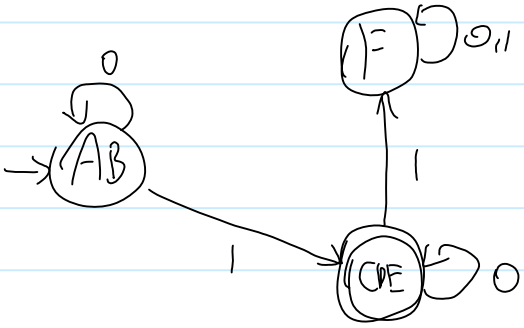
{A, B} {F} {C, D, E}

	0	1

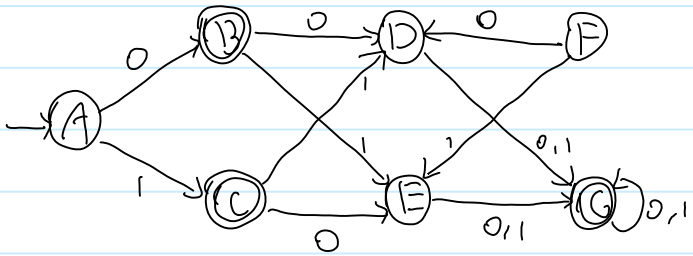
	0	1
→ A	B	C
B	A	D
C	E	F
D	E	F
E	E	F
F	F	F

{A, B} {F} {C, D, E}

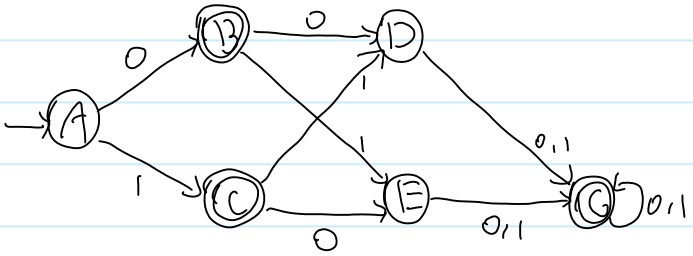
	0	1
→ {A B}	{AB}	{CDE}
{F}	{F}	{F}
{C D E}	{CDE}	{F}



example with unreachable state



step 1 : remove unreachable state



step 2 : proceed as normal

	0	1
→ A	B	C
B	D	E
C	E	D
D	G	G
E	G	G
G	G	G

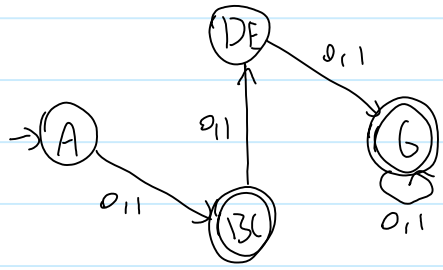
0 equivalence
{A, D, E} {B, C, G}

1 equivalence
{A, D, E} {B, C} {G}

2 equivalence
{A} {D, E} {B, C} {G}

3 equivalence
{A} {D, E} {B, C} {G}

	0	1
→ {A}	{B,C}	{B,C}
{D,E}	{G}	{G}
{B, C}	{D,E}	{D,E}
{G}	{G}	{G}



Pumping Lemma

2020年10月12日 上午 11:10

ref. youtube v=I3FuVKVgLCA

Used to prove that a language is not regular

cannot be used to prove that a language is regular

If A is a regular language, then A has a pumping length 'p' such that any string 'S' where $|S| \geq p$ may be divided into 3 parts $S = xyz$ such that the following conditions must be true:

(1) $xy^iz \in A$ for every $i \geq 0$

(2) $|y| > 0$

(3) $|xy| \leq p$

To prove that a language is not regular using pumping lemma:

proof by contradiction.

→ Assume A is regular

→ It has to have a pumping length p

→ All strings longer than p can be pumped $|S| \geq p$

→ find a string S in A such that $|S| \geq p$

→ Look at every decomposition of S into xyz such that $|y| \geq 1$ and $|xy| \leq p$

→ Find one i for each decomposition such that $xy^iz \notin A$

→ S cannot be pumped == CONTRADICTION

(example) prove that $A = \{0^n 1^n \mid n \geq 0\}$ is not regular

↖ not regular as it requires counting

Proof: Assume that A is a regular language

then A has a pumping length p

We choose a string $S = 0^p 1^p$ so that $|S| \geq p$

then, we look at every decomposition of S into xyz such that $|y| \geq 1$ and $|xy| \leq p$:

$\begin{cases} x = 0^\alpha, \alpha \geq 0 \\ y = 0^\beta, \beta \geq 1 \\ z = 0^{p-\alpha-\beta}1^p \end{cases}$; when $i=2$, $xy^iz = 0^\alpha 0^\beta 0^\beta 0^{p-\alpha-\beta}1^p$

$= 0^{p+\beta}1^p \notin A$

therefore, S cannot be pumped, which contradicts with the assumption

hence A is not regular. ■

therefore, S cannot be pumped, which contradicts with the assumption
hence A is not regular. ■

(example) prove that $L=\{0^i1^j \mid i>j\}$ is not regular

Proof: Suppose L is regular
then there exists a pumping length P for L
We choose a string $S=0^{p+1}1^p$ so that $|S|\geq P$
then, we look at all decompositions of S into xyz such that
$|y|\geq 1$ and $|xy|\leq P$:

$x=0^\alpha$, $\alpha\geq 0$
$y=0^\beta$, $\beta\geq 1$
$z=0^{p+1-\alpha-\beta}1^p$

when $i=0$, $xy^iz=0^\alpha 0^{p+1-\alpha-\beta}1^p$
$=0^{p+1-\beta}1^p$
since $\beta\geq 1$, $p+1-\beta\leq p$
so $xy^iz\notin L$

therefore S cannot be pumped, which contradicts with the assumption.
hence L is not regular. ■

(example) prove that $L=\{0^{n^2} \mid n\in N\}$ is not regular

Proof: Suppose L is regular
then there exists a pumping length P for L
we choose a string $S=0^{P^2}$ so that $|S|\geq P$
then, for every decompositions of S into x y z such that
$|y|\geq 1$ and $|xy|\leq P$:
when $i=2$, $xy^iz=xy^2z$
$|xy^2z|=p^2+|y|$
since $|y|\geq 1$, $p^2+|y|>p^2$
since $|xy|\leq P$, $p^2+|y|\leq p^2+p$
and we have $(p+1)^2=p^2+2p+1$
therefore: $p^2<|xy^2z|<(p+1)^2$, so $xy^2z\notin L$
therefore S cannot be pumped, which contradicts with the assumption
hence L is not regular. ■

(example) prove that $L=\{0^n \mid n$ is a prime number$\}$ is not regular
Proof: Suppose L is regular

- cannot choose $w=0^p$

Example: prove that $L = \{0^p \mid p \text{ is a prime number}\}$ is not regular

Proof: Suppose L is regular

then there exists a pumping length P for L

we choose a string $w = 0^R$ where R is a prime and $|z| \geq P + 2$

cannot choose $w = 0^P$ as P may not be a prime

Equivalence of Two Finite Automata

2020年10月19日　下午 12:16

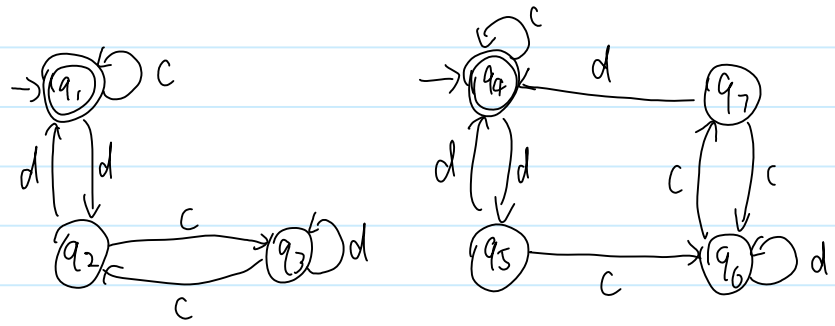
Steps to identify equivalence

1) For any pair of states $\{q_i, q_j\}$ the transition for input $a \in \Sigma$ is defined by $\{q_a, q_b\}$ where $\delta\{q_i, a\} = q_a$, $\delta\{q_j, a\} = q_b$

   The two automata are not equivalent if for a pair $\{q_a, q_b\}$ one is intermediate state and one is final state.

2) If the initial state is the final state for one automation, then the initial state must also be the final state for the second automation.

Example

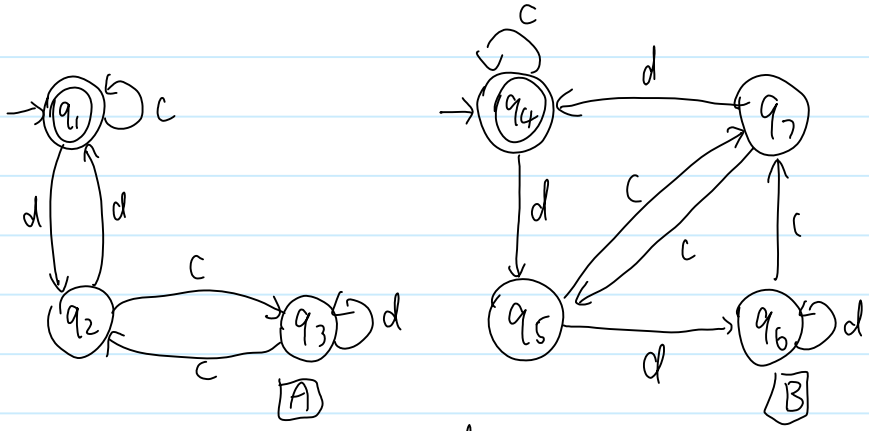


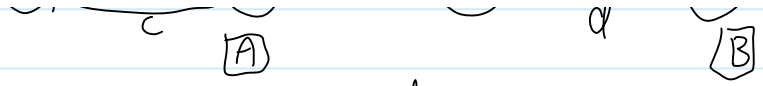
States	c	d
$\{q_1, q_4\}$	$\{q_1, q_4\}$	$\{q_2, q_5\}$
discuss any new states	fs fs	is is
$\{q_2, q_5\}$	$\{q_3^{is}, q_6^{is}\}$	$\{q_1^{fs}, q_4^{fs}\}$
$\{q_3, q_6\}$	$\{q_2^{is}, q_7^{is}\}$	$\{q_3^{is}, q_6^{is}\}$
$\{q_2, q_7\}$	$\{q_3^{is}, q_6^{is}\}$	$\{q_1^{fs}, q_4^{fs}\}$

both final states or both intermediate states, ok.

therefore, the two finite automatas are equivalent

Example





States	C	d
$\{q_1, q_4\}$	$\{q_1^f, q_4^f\}$	$\{q_2^i, q_5^i\}$
$\{q_2, q_5\}$	$\{q_3^i, q_7^i\}$	$\{q_1^i, q_6^f\}$ — final state + intermediate state ⇒ not equivalent

Since q_1 and q_6 are not of the same type of states
therefore, A and B are not equivalent.

Regular Grammar

2020年10月26日　上午 10:43

Noam Chomsky: mathematical model of grammar in 4 types:

Grammar Type	Grammar Accepted	Language Accepted	Automation
Type 0	Unrestricted Grammar	Recursively Enumerable Language	Turing Machine
Type 1	Context Sensitive Grammar	Context Sensitive Language	Linear Bounded Automation
Type 2	Context Free Grammar	Context Free Language	Pushdown Automata
Type 3	Regular Grammar	Regular Language	Finite State Automation

Grammar:

$G = (V, T, S, P)$, where

V = Set of variables or non-terminal states

T = Set of terminal states

S = Start symbol

P = Production rules for terminals and non-terminals

↑ has the form of $\alpha \to \beta$ where α and β are strings on $V \cup T$ and at least one symbol of α belongs to V.

(example) $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

$S \to AB$

$\to aB$

$\to ab$

Regular Grammar

– can be divided into two types:

Right Linear Grammar

- if all productions are of the form

$A \to xB$

$A \to x$

where $A, B \in V$ and $x \in T$

Left Linear Grammar

- if all productions are of the form

$A \to Bx$

$A \to x$

where $A, B \in V$ and $x \in T$

Example $S \to abS \mid b$ - right linear
$S \to Sbb \mid b$ - left linear

Derivations from a Grammar
- the set of strings that can be derived from a grammar is said to be the LANGUAGE generated from that grammar

Example $G_1 = (\{S, A\}, \{a, b\}, S, \{S \to aAb, aA \to aaAb, A \to \varepsilon\})$

$S \to aAb$ [by $S \to aAb$]
$\to aaAbb$ [by $aA \to aaAb$]
$\to aabb$ [by $A \to \varepsilon$]

↖ a derivation from G_1
↙ the language generated from G_1

$L(G_1) = \{a^m b^m \mid m \geq 1\}$

Example $G_2 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

$S \to AB$
$\to ab$

$L(G_2) = \{ab\}$

Example $G_3 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

$S \to AB$ by $S \to AB$
$\to aB$ by $A \to a$
$\to ab$ by $B \to b$

$S \to AB$ by $S \to AB$
$\to aAbB$ by $A \to aA, B \to bB$
$\to aabb$ by $A \to a, B \to b$

$S \to AB$ by $S \to AB$
$\to aAb$ by $A \to aA, B \to b$
$\to aaab$ by $A \to a$

$L(G_3) = \{ab, a^2b, ab^2, a^2b^2, \ldots\}$
$= \{a^+b^+$

Context Free Grammar & Context Free Language

2020年10月26日　上午 11:56

Context Free Grammar

- defined by 4 tuples $G = \{V, \Sigma, S, P\}$

V : set of variables or non-terminal symbols

Σ : set of terminal symbols

S : Start symbol

P : Production Rule → in the form of $A \to \alpha$ where $\alpha = \{V \cup \Sigma\}^*$ and $A \in V$

↑ can be ε too

(example) a language in the form of $a^n b^n$

$G = (\{S, A\}, \{a, b\}, S, \{S \to aAb, A \to aAb \mid \varepsilon\})$

$S \to aAb$

$\to aaAbb$

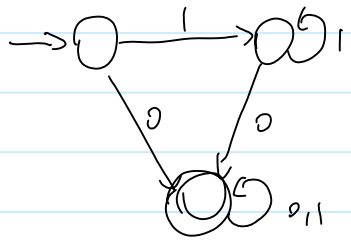
$\to aaaAbbb$

...

Regular Languages & Finite Automata

2020年10月26日　下午 12:32

Example Consider the DFA given below

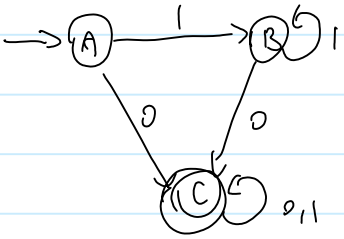


which of the following statements are true?

1. Complement of L(A) is context-free
   ↑ 补集
2. $L(A) = L((11^*0+0)(0+1)^*0^*1^*)$
3. For the language accepted by A, A is the minimal DFA.
4. A accepts all strings over {0,1} of length at least 2.

1. A is a DFA → L(A) is regular language → complement of L(A) is regular language → complement of L(A) is context-free

2. Convert the DFA to regular expression:



$C = A0 + B0 + C0 + C1$ ①

$B = A1 + B1$ ②

$A = \varepsilon$ ③

② $B = A1 + B1$

$B = 1 + B1$ by ③

$B = 11^*$ ④, by Arden's Theorem

① $C = A0 + B0 + C0 + C1$

$C = 0 + 11^*0 + C(0+1)$ by ③, ④

$C = (0+11^*0)(0+1)^*$ ← equivalent

same, same

provided r.e. $(0+11^*0)(0+11^*0^*1^*)$

∴ true

3.

	0	1
→A	C	B
B	C	B
(C)	C	C

0-equivalence
{AB} {C}

1-equivalence
{AB} {C}

minimized:

	0	1
→AB	C	AB
(C)	C	C

∴ False

4. from 3, we see that the r.e. of the DFA is:
$(0+11^*0)(0+1)^*$
therefore, 0 is accepted by the DFA, and its length is 1
∴ false

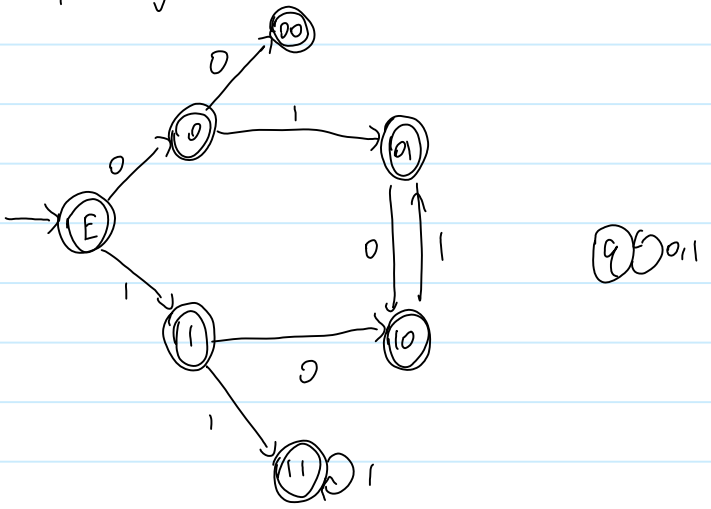
example $L_1 = \emptyset$, $L_2 = \{a\}$, what is $L_1L_2 \cup L_1^*$?

$L_1L_2^* \cup L_1^*$
$= \emptyset \cdot a^* \cup \emptyset^*$
$= \emptyset \cup \varepsilon$
$= \varepsilon$

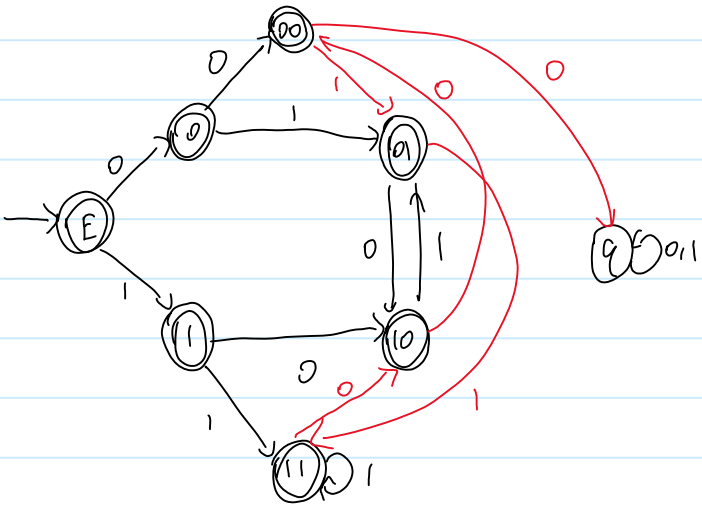
∅ is empty set!
$\emptyset R = R\emptyset = \emptyset$
$\emptyset^* = \varepsilon$

ε is empty symbol!
$\varepsilon R = R\varepsilon = R$
$\varepsilon^* = \varepsilon$

example Consider a set of strings on {0,1} in which every substring of 3 symbols has at most two zeros, e.g. 00110 and 01001 are in the language but 10010 is not. All strings of length less than 3 is also accepted. a partially completed DFA that accepts this language is shown below



What are the missing arcs in the DFA ?

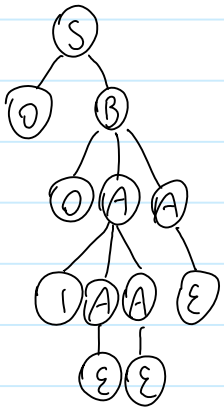

E
0
1
00
01
10
11
q
0,1

Derivation Tree

2020年10月29日　上午 11:36

A derivation tree (or a parse tree) is an ordered rooted tree that graphically represents the semantic information of strings derived from a context-free grammar.

Example For the grammar $G = \{V, T, P, S\}$ where $S \to 0B$, $A \to 1AA \mid \varepsilon$, $B \to 0AA$

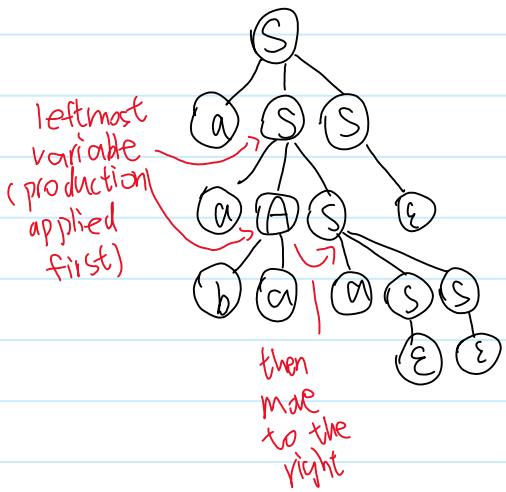


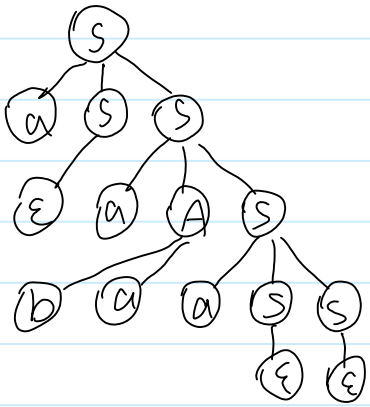
Root vertex: Must be labelled by the start symbol
vertex: Labelled by non-terminal symbols
leaves: Labelled by terminal symbols or ε.

Left derivation tree
- obtained by applying production to the leftmost variable in each step

Right derivation tree
- obtained by applying production to the rightmost variable in each step

Example For generating the string aabaa from the grammar $S \to aAS \mid aSS \mid \varepsilon$, $A \to SbA \mid ba$





Ambiguous Grammar

2020年10月29日　下午 12:13

- there exists two or more left derivation trees for a string w

example $G = (\{S\}, \{a+b, +, *\}, S, P)$ where P consists of

$S \to S+S \mid S*S \mid a \mid b$

the string $a+a*b$ can be generated as

$S \to S+S$
$\to a+S$
$\to a+S*S$
$\to a+a*S$
$\to a+a*b$

or

$S \to S*S$
$\to S+S*S$
$\to a+S*S$
$\to a+a*S$
$\to a+a*b$

thus, this grammar is ambiguous.

Removing Ambiguity　Ref: https://www.gatevidyalay.com/removing-ambiguity-grammar-ambiguity/

Note: It is not always possible to convert an ambiguous grammar into an unambiguous grammar.

An ambiguous grammar may be converted into an unambiguous grammar by implementing
- Precedence Constraints
- Associativity Constraints

Precedence Constraints
- The level at which the production is present defines the priority of the operator contained in it
- The higher the level of the production, the lower the priority of operator
- The lower the level of the production, the higher the priority of operator

Associativity Constraints
- If the operator is left associative (-, /, ...), induce left recursion on its production
- If the operator is right associative (^, ...), induce right recursion on its production

(example) Convert the following ambiguous grammar into unambiguous grammar:

$E \rightarrow a \mid E+E \mid E*E$

where + is addition and * is multiplication

Solution: The given grammar consists of the following operators:

+, *

The given grammar consists of the following operands:

a

The priority order is:

a > * > +

Where:

the + and * operators can be seen as left assaiative

Using the precedence and associativity rules, we write the corresponding unambiguous grammar as:

$E \rightarrow E + T \mid T$

$T \rightarrow T * a \mid a$

Simplification of CFG

2020年11月9日　下午 1:56

Ref: https://www.youtube.com/watch?v=lEsDll4Um7Y
https://www.youtube.com/watch?v=eUlUzH9fmXQ
↑null and unit productions　useless symbols↓
https://www.geeksforgeeks.org/simplifying-context-free-grammars/
https://www.sanfoundry.com/automata-theory-cfg-eliminating-useless-symbols/

3 steps
1. Remove useless productions
2. Remove unit productions
3. Remove null productions

Remove useless productions
1. Eliminate non-producing symbols
2. Eliminate unreachable symbols
⚠ Productions in which the removed symbol occurs are removed

(example) Simplify:
P. S → abS | abA | abB
A → cd
B → aB
C → dc

Solution: non-producing symbol: B
after removal:
P: S → abS | abA
A → cd
C → dc
unreachable symbol: C
after removal:
P: S → abS | abA
A → cd

Remove null productions
- a non-terminal symbol A is nullable if there is a production $A \to \varepsilon$ or $A \to \underbrace{X_1 X_2 \cdots X_n}_{\text{variables}}$ and each X_i is nullable

(Procedure)
o for each rule $A \to \alpha B \beta$ where α, β is any mix of vars or terminals, and

Procedure

- for each rule $A \to \alpha B \beta$ where α, β is any mix of vars or terminals, and B is a single var, add $A \to \alpha\beta$
- if S' is nullable, add $S' \to \varepsilon$

Example remove null productions from the following grammar:
$P: S \to ABAC,\ A \to aA \mid \varepsilon,\ B \to bB \mid \varepsilon,\ C \to c$

null productions: $A \to \varepsilon,\ B \to \varepsilon$

for $A \to \varepsilon$:
$S \to ABAC \mid ABC \mid BAC \mid BC$
$A \to aA \mid a$
therefore, $P: S \to ABAC \mid ABC \mid BAC \mid BC$
$A \to aA \mid a,\ B \to bB \mid \varepsilon,\ C \to c$

for $B \to \varepsilon$:
$S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$
$B \to bB \mid b$
therefore, $P: S \to ABAC \mid ABC \mid BAC \mid BC \mid AAC \mid AC \mid C$
$A \to aA \mid a,\ B \to bB \mid b,\ C \to c$

Remove unit productions
- Any production rule of the form $A \to B$ where $A, B \in$ non-terminals is called unit production

Procedure

- To remove $A \to B$, add production $A \to x$ to the grammar rule whenever $B \to x$ occurs in the grammar. [$x \in$ terminal, x can be
- Delete $A \to B$ from the grammar
- Repeat until all unit productions are removed.

Example Remove unit productions from the grammar whose production rule is given by
$P: S \to X,\ X \to a,\ Y \to Z \mid b,\ Z \to M,\ M \to N,\ N \to a$

Unit productions: $Y \to Z$, $Z \to M$, $M \to N$

for $M \to N$:
since $N \to a$, we add $M \to a$, and remove $M \to N$
P: $S \to XY$, $X \to a$, $Y \to Z|b$, $Z \to M$, $M \to a$, $N \to a$

for $Z \to M$:
since $M \to a$, we add $Z \to a$ and remove $Z \to M$
P: $S \to XY$, $X \to a$, $Y \to Z|b$, $Z \to a$, $M \to a$, $N \to a$

for $Y \to Z$:
since $Z \to a$, we add $Y \to a$ and remove $Y \to Z$
P: $S \to XY$, $X \to a$, $Y \to a|b$, $Z \to a$, $M \to a$, $N \to a$

Chomsky Normal Form

Ref: https://www.youtube.com/watch?v=lEsDll4Um7Y
https://www.youtube.com/watch?v=eUlUzH9fmXQ

2020年11月9日 下午 3:34

- restrictions on the RHS

(rules)

1. only rule that can make ε is $S \to \varepsilon$
2. $A \to a$
3. $A \to BC$

} $A, B, C \in V$, $a \in \Sigma$, $B, C \neq S$

(steps)

1. Ensure start variable S not on RHS of any rule.

$S \to \dots$, $A \to SB$ $\Rightarrow$ $S' \to S$ (new start variable), $S \to \dots$, $A \to SB$

2. Remove null productions
3. Remove unit productions
4. Break up mix of vars and terminals

$A \to \alpha a \beta$ (a: mix of vars and terminals) $\Rightarrow$ $U_a \to a$, $A \to \alpha U_a \beta$

5. Break up RHS

$A = BCDE \Rightarrow A = Y_1 DE, Y_1 = BC \Rightarrow A = Y_2 E, Y_1 = BC, Y_2 = Y_1 D$

order matters

(example) P: $S \to AaB \mid b$, $A \to S \mid \varepsilon \mid AB$, $B \to bbb \mid ASA$

[1] add new start state $S' \to S$

P: $S' \to S$, $S \to AaB \mid b$, $A \to S \mid \varepsilon \mid AB$, $B \to bbb \mid ASA$

[2] nullable variable: A

add $S \to aB$, $A \to B$, $B \to SA \mid AS \mid S$

P: $S' \to S$, $S \to AaB \mid b \mid aB$, $A \to S \mid AB \mid B$, $B \to bbb \mid ASA \mid SA \mid AS \mid S$

P: S'→S , S→AaB|b|aB , A→S|AB|B, B→bbb|ASA|SA|AS|S

3 unit productions

S'→S

S → AaB|b|aB

A → S|AB|B

B → bbb|ASA|SA|AS|S

□: unit productions

⇓

S'→ AaB|b|aB

S → AaB|b|aB

A → AaB|b|aB|AB|bbb|ASA|SA|AS

B → bbb|ASA|SA|AS|AaB|b|aB

□: replaced

4 replace terminals

$U_a \to a$

$U_b \to b$

single terminal are safe to be left

S' → AUaB|b|UaB

S → AUaB|b|UaB

A → AUaB|b|UaB|AB|UbUbUb|ASA|SA|AS

B → Ub|UbUb|ASA|SA|AS|AUaB|b|UaB

5 break up RHS

$Y_1 \to AU_a$

S' → Y1B|b|UaB

S → Y1B|b|UaB

$Y_2 \to U_bU_b$

$Y_3 \to AS$

A → Y1B|b|UaB|AB|Y2Ub|Y3A|SA|AS

B → Y2Ub|Y3A|SA|AS|Y1B|b|UaB

$U_a \to a$

$U_b \to b$

therefore, the CNF of the CFG is:

P: $S' \to Y_1B \mid b \mid U_aB$

$S \to Y_1B \mid b \mid U_aB$

$A \to Y_1B \mid b \mid U_aB \mid AB \mid Y_2U_b \mid Y_3A \mid SA \mid AS$

$B \to Y_2U_b \mid Y_3A \mid SA \mid AS \mid Y_1B \mid b \mid U_aB$

$U_a \to a$

$U_b \to b$

$Y_1 \to AU_a$

$Y_2 \to U_bU_b$

$Y_3 \to AS$

Greibach Normal Form

2020年11月9日　下午 6:29

in the form of

$A \to b$

$A \to bC_1C_2C_3 \cdots C_n$

where b is terminal and $A, C_1, \ldots, C_n$ are variables

(steps)

- △ convert to CNF
- △ change the names of variables into some A_i in ascending order of i
- △ if a rule is in the form $A_i \to A_j x$, then alter the rule so that $i < j$
- △ remove left recursion

$A_1 \to A_2 A_3$, OK ($1<2$)

$A_1 \to A_4 A_4$, OK ($1<4$)

$A_4 \to A_1 A_4$, not OK → substitute A_1 ($4 \geq 1$)

$A_4 \to A_4 A_4$, left recursion, not ok. → remove left recursion ($4 \geq 4$)

(example)

$S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$

first: convert to CNF

1. add new start state $S' \to S$

unecessary here
the reason why we don't allow S in RHS is S is allowed to give ε while other variables are not, so it may implicitly add ε to the rules.
However here S does not give ε, therefore adding S0 is not necessary.

2. nullable variables: none

3. unit productions: none

4. remove terminal / variable mix

P: $S' \to S$, $S \to CA \mid BB$, $B \to b \mid SB$, $C \to b$, $A \to a$ ⇒ P = $S' \to CA \mid BB$, $S \to CA \mid BB$, $B \to b \mid SB$, $C \to b$, $A \to a$

5. break up long RHS: nothing to break.

CNF: P = $S' \to CA \mid BB$, $S \to CA \mid BB$, $B \to b \mid SB$

$S \to CA \mid ISD$
$B \to b \mid SB$
$C \to b$
$A \to a$

change variable name in ascending order:

$S' \to A_1$,
$S \to A_2$,
$C \to A_3$,
$A \to A_4$,
$B \to A_5$.

← ascending order which they appear ⇒

$A_1 \to A_3A_4 \mid A_5A_5$
$A_2 \to A_3A_4 \mid A_5A_5$
$A_5 \to b \mid A_2A_5$
$A_3 \to b$
$A_4 \to a$

alter rules so that $i < j$ for $A_i \to A_j\alpha$

$A_5 \to b \mid A_2A_5$
$A_5 \to b \mid A_3A_4A_5 \mid A_5A_5A_5$
$A_5 \to b \mid bA_4A_5 \mid A_5A_5A_5$ ← left recursion: rule in the form of $A \to A\alpha$ for $A \in V$

remove left recursion:
- introduce new variable

$A_5 \to b \mid bA_4A_5 \mid A_5A_5A_5$
let $Z \to A_5A_5Z \mid A_5A_5$ ← problematic, remove the rest: write with Z once, write alone once
then, $A_5 \to b \mid bA_4A_5 \mid bZ \mid bA_4A_5Z$
add Z

now the grammar is:

replace with A_3
replace with A_5

$A_1 \to A_3A_4 \mid A_5A_5$
$A_2 \to A_3A_4 \mid A_5A_5$
$A_5 \to b \mid bA_4A_5 \mid bZ \mid bA_4A_5Z$
$Z \to A_5A_5Z \mid A_5A_5$
$A_3 \to b$
$A_4 \to a$

↓

$A_1 \to bA_4 \mid bA_5 \mid bA_4A_5A_5 \mid bZA_5 \mid bA_4A_5ZA_5$
$A_2 \to bA_4 \mid bA_5 \mid bA_4A_5A_5 \mid bZA_5 \mid bA_4A_5ZA_5$
$A_5 \to b \mid bA_4A_5 \mid bZ \mid bA_4A_5Z$
$Z \to bA_5Z \mid bA_4A_5A_5Z \mid bZA_5Z \mid bA_4A_5ZA_5Z \mid$
$bA_5 \mid bA_4A_5A_5 \mid bZA_5 \mid bA_4A_5ZA_5$

$Z \to bA_5Z \mid bA_4A_5A_5Z \mid bZA_5Z \mid bA_4A_5ZA_5Z \mid$
$bA_5 \mid bA_4A_5A_5 \mid bZA_5 \mid bA_4A_4ZA_5$

$A_3 = b$

$A_4 \to a$

Pumping Lemma for CFL

2020年11月16日　下午 2:57

let L be a CFL

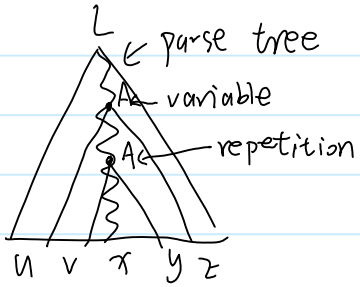
then there exists a "pumping constant" p for L (p: number of vars)

then for all $w \in L$ with $|w| \geq p$,

there exists u, v, x, y, z such that

$w = uvxyz$ such that

1. $uv^i xy^i z \in L$ for all $i \geq 0$
2. $|vy| \geq 1$ (or $vy \neq \varepsilon$)
3. $|vxy| \leq p$



Prove steps

1. Assume L is a CFL
2. Exists a pumping constant p for L
3. Pick some string $w \in L$ with $|w| \geq p$
4. Look at all decompositions of w into $uvxyz$ such that:
   a) $|vy| \geq 1$
   b) $|vxy| \leq p$
5. find one i such that $uv^i xy^i z \notin L$

(example) prove that $\{a^n b^n c^n : n \geq 0\}$ is not CFL

Proof: Suppose L_1 is a CFL

then there exists a pumping length p for L_1

we choose a string $w = a^p b^p c^p$ such that $w \in L_1$ and $|w| \geq p$

we then look at all the decompositions of w into $uvxyz$

such that $|vy| \geq 1$ and $|vxy| \leq p$:

case 1: v and y only contain a's

$\underbrace{a\,a\,a\,a \cdots a}_{u\ v\ x\ y}\ \underbrace{b \cdots b\, c \cdots c}_{z}$

then when $i = 2$, $uv^i xy^i z = a^{p+2} b^p c^p \notin L$

case 2: v and y only contain b's or c's

same as case 1

case 3: v and y only contain a's and b's

$\underbrace{a \cdots\cdots a}\ \underbrace{b \cdots\cdots b}\ \underbrace{c \cdots c}$

case 3: v and y only contain b's and c's

$a \cdots\cdots ab \cdots\cdots bc \cdots c$

u | v, x, y | z

then when $i=2$, the number of a and b will be larger than p, while the number of c is p

so $uv^2xy^2z \notin L$

case 4: v and y only contain b's and c's

same as case 3

therefore, w cannot be pumped under all cases

L_1 is not CFL.

example $L_2 = \{a^i b^j c^k \mid i \le j \le k\}$

suppose L_2 is CFL, then there is a pumping length p for L_2.

we choose $w = a^p b^p c^p$

we decompose w into uvxyz such that $|vy| \ge 1$ and $|vxy| \le p$

case 1: v+y are only in a's

⇒ when $i=2$, there will be more a's than b's

case 2: v+y are only in b's

⇒ when $i=2$, then there will be more b's than c's

case 3: v+y are only in c's

⇒ when $i=0$, then there will be more b's than c's

case 4: vxy has a's and b's

⇒ when $i=2$, then there will be more b's than c's

case 5: vxy has b's and c's

⇒ when $i=0$, then there will be more a's than b's

therefore, w cannot be pumped under all cases

L_2 is not CFL

example $L_3 = \{ww \mid w \in \{0,1\}^*\}$

suppose L_3 is a CFL, then there exists a pumping length p for L

we choose $S = 0^p 1^p 0^p 1^p$ and decompose S into uvxyz such that $|vy| \ge 1$ and $|vxy| \le p$

case 1: vxy in the first $0^p 1^p$:

$0 \cdots\cdots 01 \cdots\cdots 10 \cdots 01 \cdots 1$

case 1: vxy in the first 0^p1^p:

$\underbrace{0\cdots}_{u}\underbrace{\cdots 01\cdots}_{vxy}\underbrace{\cdots 10\cdots 01\cdots 1}_{z}$

when i=2, the first part is not the same as the second part.

case 2: vxy in the second 0^p1^p

similar to case 1.

case 3: vxy in 1^p0^p

$\underbrace{0\cdots 01\cdots}_{u}\underbrace{10\cdots}_{vxy}\underbrace{01\cdots 1}_{z}$

when i=2, the first part will have more 1 than the second part, hence not the same.

therefore, S cannot be pumped under all cases.

L_3 is not CFL.

example $L_4 = \{w \in \{a,b,c,d\}^* : \#\text{ of } c\text{'s} > \#\text{ of } a\text{'s}, b\text{'s}, d\text{'s}\}$

Assume L_4 is a CFL, so there exists a pumping length p for L_4.

we choose a string $w = c^{p+1}a^pb^pd^p$, and decompose it into uvxyz such that $|vy| \geq 1$ and $|vxy| \leq p$

case 1: vxy in c's

when i=0, # of c's $\leq$ # of a's

case 2: vxy only in a's

when i=2, # of a's $\geq$ # of c's

case 3: vxy only in b's or d's

similar to case 2

case 4: vxy in a's and b's

when i=2, # of a's or # of b's $\geq$ # of c's

case 5: vxy in b's and c's

similar to case 4

case 6: vxy in c's and a's

when i=0, # of c's $\leq$ # of b's

or # of c's $\leq$ # of a's

therefore, w cannot be pumped under all cases

L_4 is not CFL.

Pushdown Automata

2020年11月30日　下午 12:37

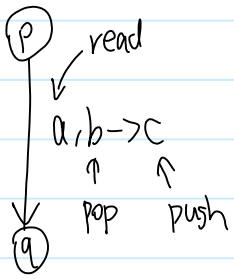
- machine model for CFGs

similar to NFA, but on each transition, we can:

- push (or not)
- pop (or not)

} to the stack ← starts empty, ↖ can end non-empty

transition:

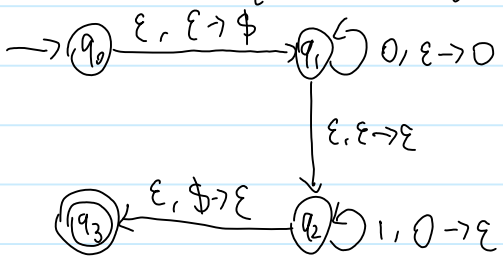


Can take transition if:

1. we can read a
   and
2. we can pop b

example $L = \{0^n 1^n : n \geq 0\}$

push a dummy char first: end of stack



$PDA = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \{\varepsilon, \$, 0, 1\}, \delta, q_0, \{q_3\})$

$\delta(q_0, \varepsilon, \varepsilon) = \{(q_1, \$)\}$

$\delta(q_1, 0, \varepsilon) = \{(q_1, 0)\}$

$\delta(q_1, \varepsilon, \varepsilon) = \{(q_2, \varepsilon)\}$

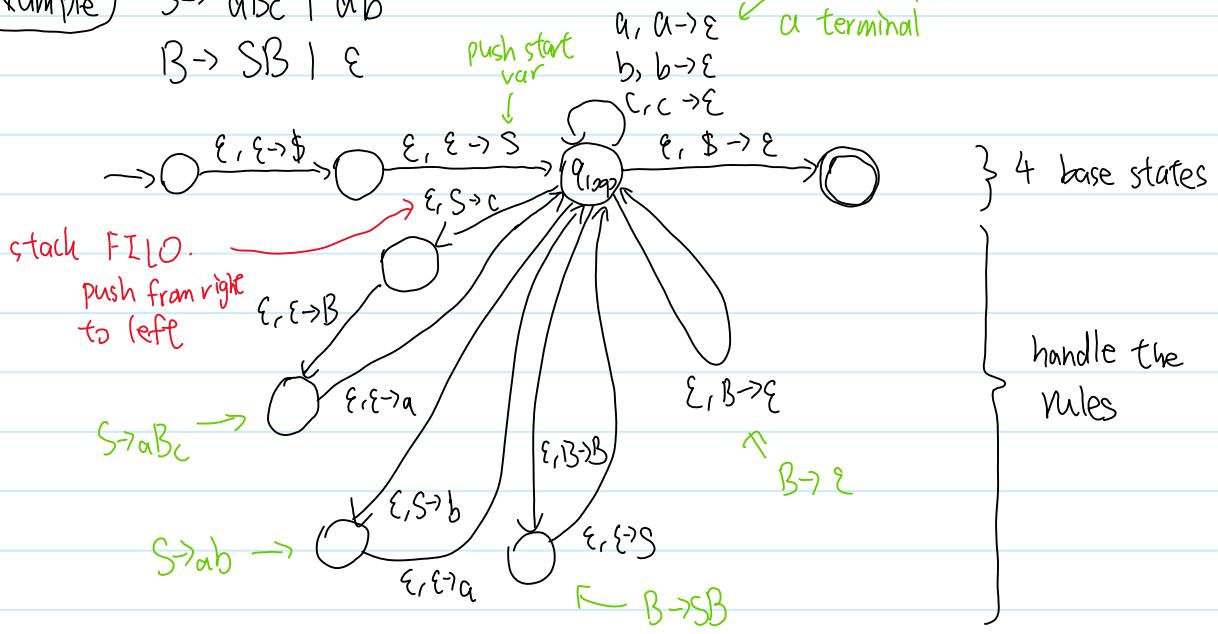
$\delta(q_2, 1, 0) = \{(q_2, \varepsilon)\}$

$\delta(q_2, \varepsilon, \$) = \{(q_3, \varepsilon)\}$

CFG → PDA

example $S \to aBc \mid ab$

$B \to SB \mid \varepsilon$

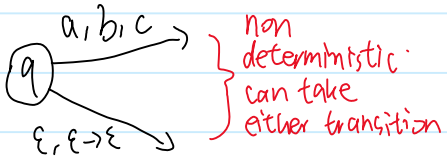


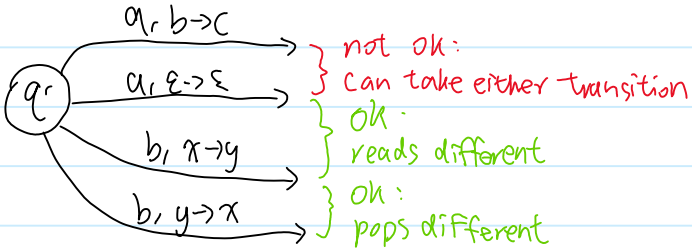
Simplification of PDA

1. Force stack to end empty
2. Every transition stores or ends empty

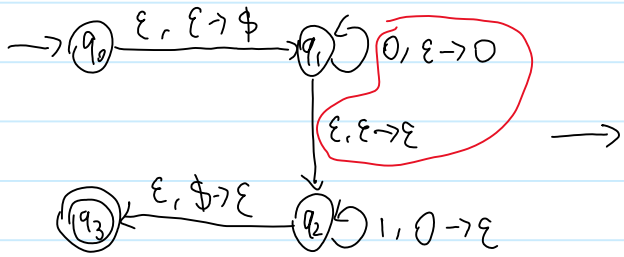
(pass)

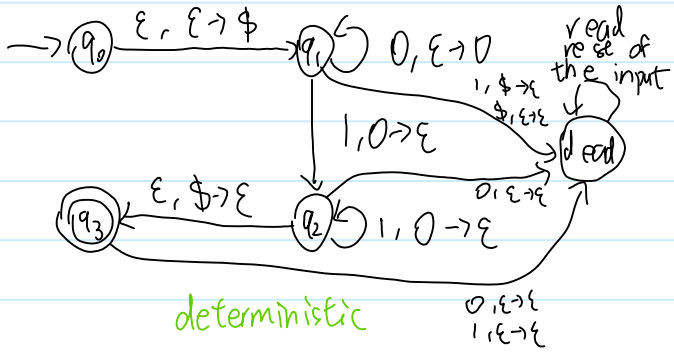
Deterministic CFLs → DPDAs





Example $L = \{0^n 1^n : n \geq 0\}$





DCFL ≠ CFL

DCFL properties

1. Closed under complement
2. NOT closed under union / intersection
3. NOT closed under concatenation

Neso

Formal definition

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Q = Finite set of states

A → Σ = Finite set of input symbols

M → Γ = ...

→ Takes arguments as a triple $\delta(q, a, X)$ where

1. q is a state in Q
2. a is either an input symbol in Σ or ε

A → Σ = Finite set of input symbols

m → Γ = A finite stack alphabet

δ = The transition function

i → q_0 = The start state

Z_0 = The start stack symbol ($)

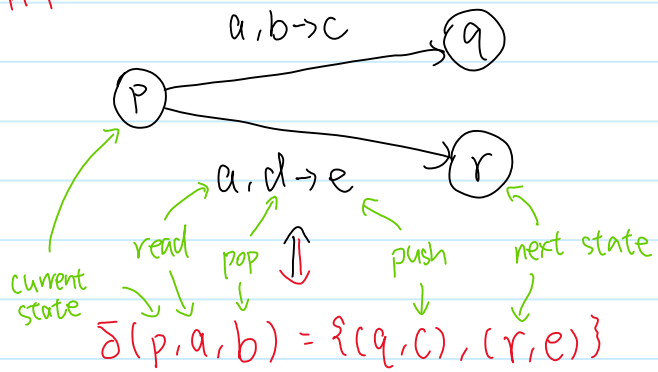
F = The set of final / Accepting states

missing in lecture ppt ↑ ↑

2. α is either an input symbol in Σ or ε
3. X is a stack symbol in Γ

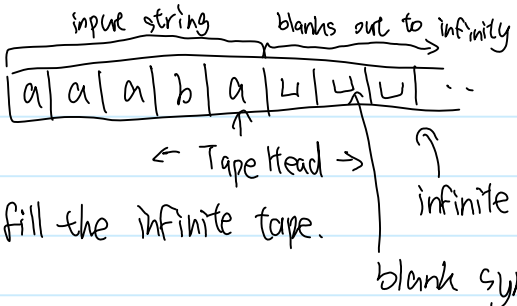
The output is finite set of pair (p, γ) where
- p is a new state
- γ is a string of stack symbols that replaces X at the top of the stack

e.g. if $\gamma = \varepsilon$ then X is popped
if $\gamma = X$ then stack is unchanged
if $\gamma = ZY$ then X replaced by Y and Z pushed



Turing Machine

2020年11月30日　下午 3:08



data structure: tape

Tape alphabets: $\Sigma = \{0, 1, a, b, X, Z_0\}$

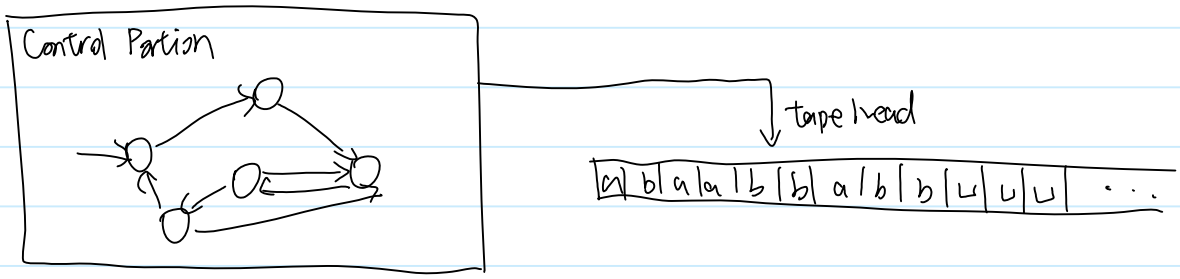
The blank ␣ is a special symbol used to fill the infinite tape.

we denote ␣ as ß

operations:

1. read the symbol below the tape head
2. update / write a symbol below the tape head
3. move the tape head one step left / right

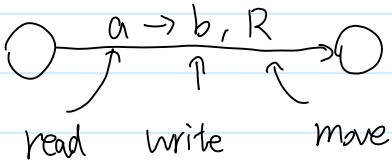
Structure



↑ similiar to FSM or PDA

is deterministic

Operation rules



△ if on the left end and move Left, the tape head will not move.

△ if updating the cell is not desired, write the same symbol.

△ Control is with a sort of FSM
- initial state
- final states : either accept or reject
- computations : halt and accept
  halt and reject
  loop (failed to halt)

Formal definition

Formal definition

$P = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$

Q = Non empty set of states

Σ = Non empty set of symbols ($\Sigma \subseteq \Gamma$)

Γ = Non empty set of tape symbols

δ = Transition function → defined as $Q \times \Sigma \rightarrow \Gamma \times (R/L) \times Q$

- Q: in current state
- Σ: given particular symbol
- Γ: write sth. into the tape sequence
- (R/L): move either right or left
- Q: go to the next state

q_0 = Initial state

b = Blank symbol

F = Set of Final states (accept state q_{accept} and reject state q_{reject})

the production rule of Turing Machine will be written as

$\delta(q_0, a_1) \rightarrow (q_1, y, R)$

- q_0: at current state
- a_1: input symbol
- q_1: go to new state
- y: write symbol
- R: move left/right

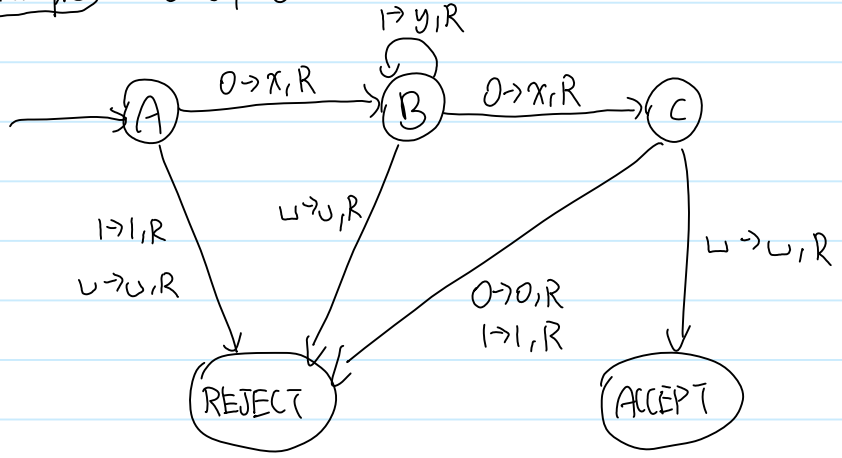
Turing Thesis

any computation that can be carried out by mechanical means can be performed by some Turing Machine

Recursively Enumerable Language

A language L and Σ is said to be recursively enumerable if there exists a Turing Machine that accepts it.

Example $L = 01^*0$

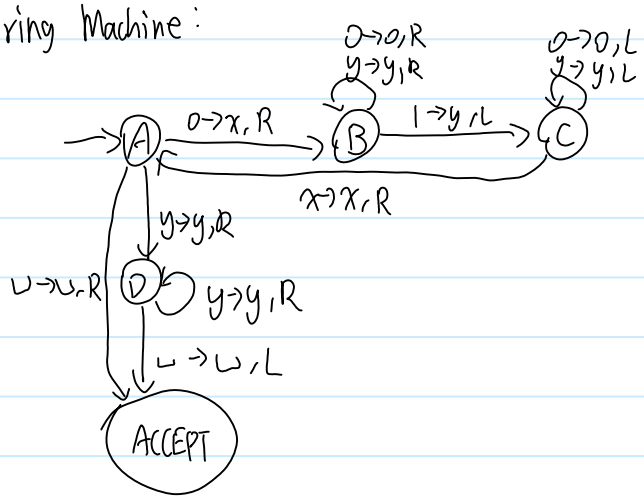


Example $L = 0^N 1^N$

Example $L = 0^N 1^N$

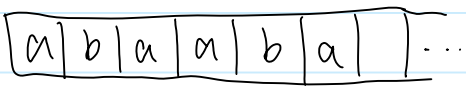
Algorithm:
- change leftmost 0 to x
- move right to the first 1 — reject if none
- change the 1 to y
- move left to the leftmost 0
- repeat until no 0 is left
- check that no 1 is left

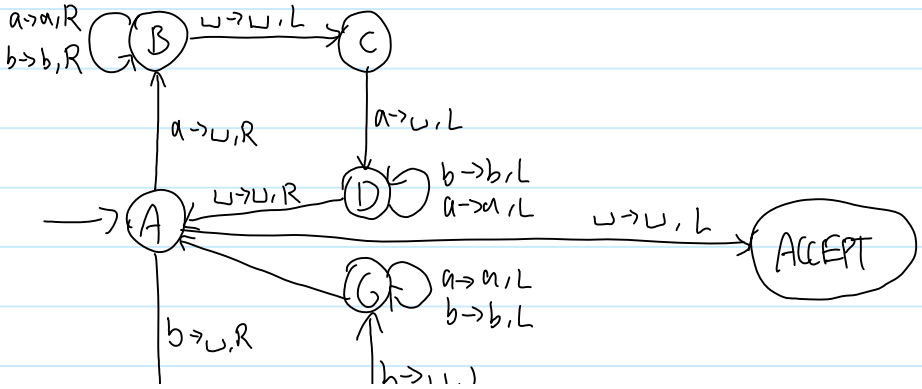
Turing Machine:

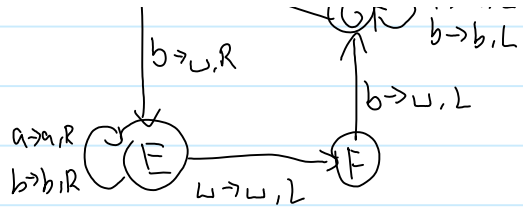


Church - Turing Thesis
- What does computable mean?
  - Alonzo Church : lambda calculus
  - Alan Turing : Turing Machine

Example Turing Machine for even palindromes



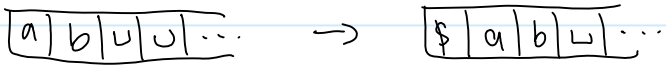


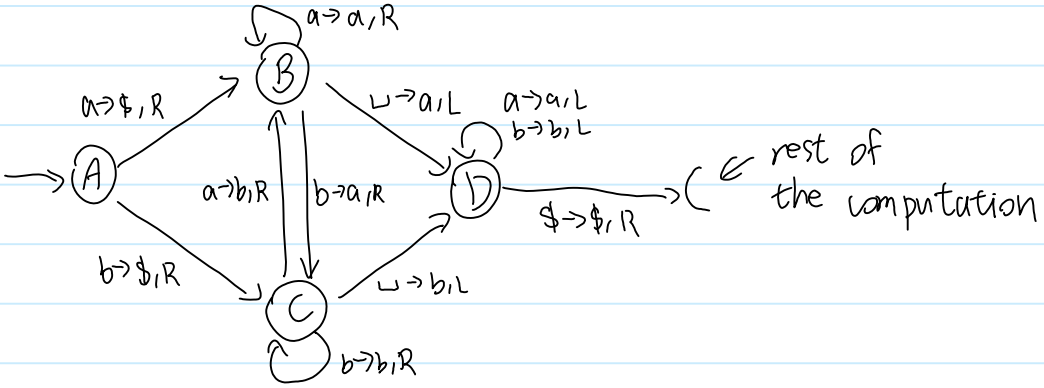


Turing Machine Techniques

- How do we recognize the left end of the tape?

Sol: Put a special symbol $ on the left end of the tape and shift the input over one cell to the right.





- Build a Turing Machine to recognize the language $0^{N}1^{N}0^{N}$

Idea: ↑ in a previous example we built a Turing Machine that recognizes $0^{N}1^{N}$ and turn it into $x^{N}y^{N}$

Therefore,

1. recognize $0^{N}1^{N}$ and turn in into $x^{N}y^{N}$
2. build another Turing Machine that accepts $y^{N}0^{N}$
3. combine the two Turing Machines

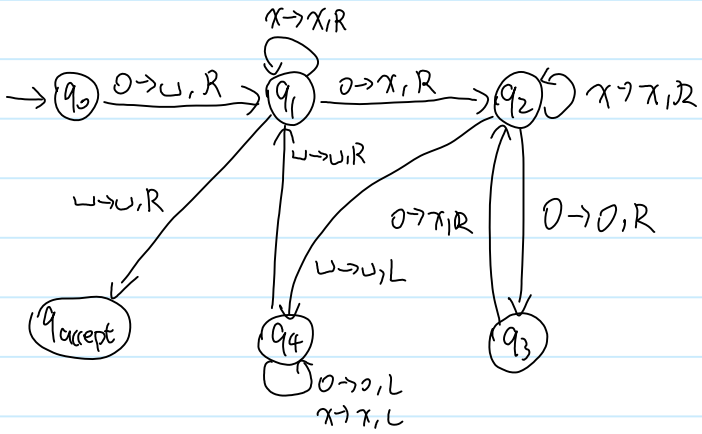
- Comparing two strings: construct a Turing Machine that accepts $\{w\#w \mid w \in \{a,b,c\}^{*}\}$

Idea: 1. use a new symbol such as X

2. examine the leftmost symbol of each part that is not X, replace the symbol into x after it is examined

Easy Theory

Example Check whether the given Turing machine accepts 0000



(All unwritten transitions go to q_{eject})

Input: 0000 ⇒ ␣000 ⇒ ␣x00 ⇒ ␣x00 ⇒ ␣x0x␣ ⇒ ␣x0x␣

State: q_0 q_1 q_2 q_3 q_2 q_4

⇒ ␣x0x␣ ⇒ ␣x0x␣ ⇒ ␣x0x␣ ⇒ ␣x0x␣ ⇒ ␣xxx␣

q_4 q_4 q_4 q_1 q_1 q_2

⇒ ␣xxx␣ ⇒ ␣xxx␣ ⇒ ... ⇒ ␣xxx␣ ⇒ ␣xxx␣ ⇒ ... ⇒

q_2 q_4 q_4 q_1

␣xxx␣ ⇒ ␣xxx␣␣

q_1 q_{accept}

therefore it accepts 4 zeros

Decidability

2020年12月6日　下午 2:41

Recursive language
- L is said to be recursive if there exists a Turing machine that accepts strings in L and rejects strings not in L
- Will always halt and give accept / reject

↖ Turing decidable

↙ Turing recognizable

Recursive Enumerable Language
- L is said to be recursive if there exists a Turing machine that accepts strings in L
- Will always halt if input string is in L, but may either reject or loop otherwise

⇓

Decidable Language
- A language is decidable if it is a recursive language

Partially Decidable Language
- A language is partially decidable if it is a recursive enumerable language

Undecidable Language
- A language is undecidable if there exists no Turing machine that accepts it.
- May sometimes be partially decidabl

Universal Turing Machine
- The Turing machine that takes another Turing machine M and a string w as input tries to determine whether M accepts w.

The language
$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w\}$
Is Turing Recognizable ← not decidable, as it may loop

The halting problem

The halting problem

- Given a Turing Machine, will it halt when run on some particular input string?
  Undecidable

(example) Given a Java code, does it always terminate?
Ans: In general we don't know.
What we can do is perhaps run it and see
For many programs they may terminate or sometimes loop.

(proof) Prove that the halting problem is undecidable
Proof: Assume it is decidable, then we can
define Halt(Program, Input)

```
if Program halts given Input
    return HALT
else
    return NOT_HALT
```

This allows us to write another function:

```
define A(X)
  if Halt(X,X) == HALT
      loop forever
  else
      return none
```

Then if we run A on itself:
A(A)
The result is:
If Halt(A,A) == HALT, A will not halt.
If Halt(A,A) == NOT_HALT, A will halt.

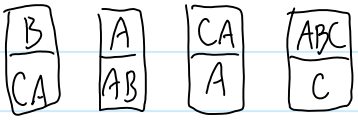
This contradicts with the assumption that the halting problem is decidable, therefore it is undecidable. ■

The Post Correspondence Problem

- Given some dominos

The Post Correspondence Problem

- Given some dominos



Each domino can be used many times

Problem: find a sequence of dominos such that the top and bottom strings are the same

Sample Solution:

A	B	CA	A	ABC
AB	CA	A	AB	C

∴ the process of enumerating may not stop

example

① ② ③

first, we can only apply ①

10

101

then, we can only apply ③

10101

10111

again, we can only apply ③

1010101

1011011

and it goes on and does not terminate.

Reduction Ref: textbook P392

If we have an algorithm to convert instances of a problem P_1 to instances of a problem P_2, then we say that P_1 reduces to P_2. We can use this proof to show that P_2 is at least hard as P_1. Therefore, if P_2 is recursive, then P_1 is recursive If P_1 is undecidable, then P_2 is undecidable.

Example Show how the halting problem can be reduced to the blank tape halting problem

Ref: https://cs.stackexchange.com/questions/41239/halting-problem-reducing-to-the-blank-tape-halting-problem

Solution: we have two languages:

Ref: https://cs.stackexchange.com/questions/41239/halting-problem-reducing-to-the-blank-tape-halting-problem

Solution: we have two languages:

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine that halts on input } w\}$

$BLANKHALT_{TM} = \{\langle M\rangle \mid M \text{ is a Turing machine that halts on input } \varepsilon\}$

We use proof by contradiction to prove that $BLANKHALT_{TM}$ is undecidable:

Suppose $BLANKHALT_{TM}$ is decidable, then there is a Turing machine T that decides it. Then, using T, we can construct a Turing machine R that decides $HALT_{TM}$ where R is:

On input $\langle M, w\rangle$:

1. Construct a new Turing machine M_w that starts with a blank tape, and write w onto the tape, then move its head back to the start and behave the same as M
2. Run $T(\langle M_w\rangle)$ If T accepts, accept. Otherwise reject.

Then this will decide $HALT_{TM}$ However, we know that $HALT_{TM}$ is not decidable, therefore T can't exist and $BLANKHALT_{TM}$ is undecidable.

(example) Show how the halting problem can be reduced to the uniformed halting problem

Ref: https://cs.stackexchange.com/questions/41243/halting-problem-reduction-to-halting-for-all-inputs

Solution: we have two languages:

$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine that halts on input } w\}$

$UNIFORMHALT_{TM} = \{\langle M, Y\rangle \mid M \text{ is a Turing machine and } Y \text{ is the collection of all inputs of } M, M \text{ halts on every elements of } Y\}$

We use proof by contradiction to prove that $UNIFORMHALT_{TM}$ is undecidable:

Suppose $UNIFORMHALT_{TM}$ is decidable, then there is a Turing machine T that decides it. Then, using T, we can construct a Turing machine R that decides $HALT_{TM}$ where R is:

On input $\langle M, w\rangle$:

1. Construct a new Turing machine M_w that starts with M and Y, replace every element of Y to w, and move its head back to the start and behave the same as M
2. Run $T(\langle M, Y\rangle)$ If T accepts, accept. Otherwise reject

behave the same as M

2. Run T(<Mw, Y>) If T accepts, accept. Otherwise reject.

Then this will decide $HALT_{TM}$. However, we know that $HALT_{TM}$ is not decidable, therefore T can't exist and $UNIFORMHALT_{TM}$ is undecidable.

(example) a recursive enumerable language

Answer: the language corresponds to the halting problem is a recursively enumerable language, namely:

$L_{halt} = \{<M,w> \mid M \text{ is a Turing machine that halts on input } w\}$

(example) a not recursive enumerable language

Answer: the language corresponds to the complement of the halting problem is not recursive enumerable, namely:

$\overline{L}_{halt} = \{<M,w> \mid M \text{ is a Turing machine that doesn't halt on input } w\}$