Fundamentele Informatica 2

Formal Languages and Automata

Hendrik Jan Hoogeboom

Bachelor Informatica
Universiteit Leiden

Fall 2019

Universiteit Leiden
Leiden Institute of
Advanced Computer Science

Contents

edit 2020-01-08

lecture based on the book

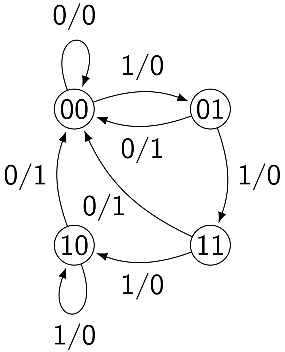
John C. Martin:
Introduction to Languages and the Theory of Computation.
Mcgraw-Hill, 4th [international] edition, 2011

Section 1

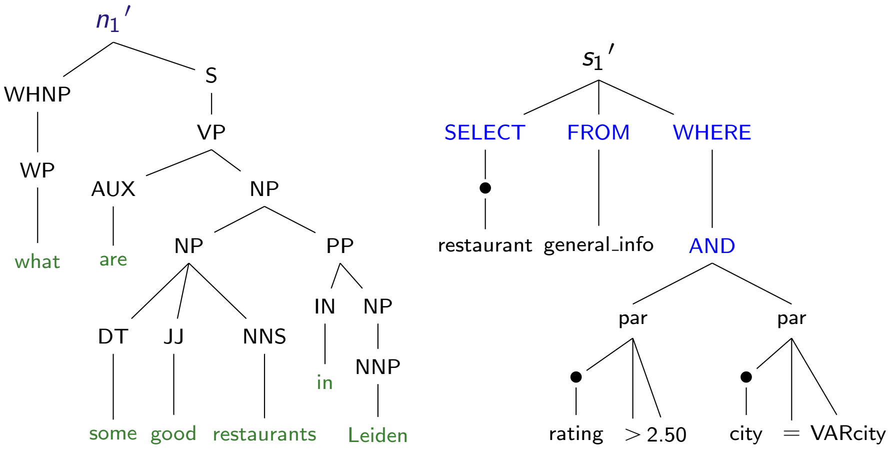
Languages

1. Logic and recursive-function theory   Logica
2. Switching circuit theory and logical design   DiTe
3. Modeling of biological systems, particularly developmental systems and brain activity
4. Mathematical and computational linguistics
5. Computer programming and the design of ALGOL and other problem-oriented languages

S.A. Greibach. Formal Languages: Origins and Directions.
Annals of the History of Computing (1981) doi:10.1109/MAHC.1981.10006



Digital Technique by Todor Stefanov, Leiden University



A. Giordani and A. Moschitti. Corpora for Automatically Learning to Map Natural Language Questions into SQL Queries (LREC 2010)

Inductive definition ⇒ grammar

inductive definition (of set of strings over $\{\ (,\ )\ \}$)

Example

- $\lambda \in \mathit{Balanced}$ — *basis*
- for every $x, y \in \mathit{Balanced}$, also $xy \in \mathit{Balanced}$ — *induction:1*
- for every $x \in \mathit{Balanced}$, also $(x) \in \mathit{Balanced}$ — *:2*
- no other strings in *Balanced* — *closure*

strings

basis λ ind:2 $(\lambda) = ()$ ind:1 $()()$ ind:2 $(())$

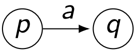
ind:1 $()()()$, $()(())$, $(())()$, ind:2 $(()())$, $((()))$

grammar

rules: $S \to \lambda \mid SS \mid (S)$

rewriting: $S \Rightarrow SS \Rightarrow S(S) \Rightarrow (S)(S) \Rightarrow ()(S) \Rightarrow ()((S)) \Rightarrow ()(())$

[M] E 1.19 see Dyck language, Catalan numbers

TYPE	**grammar**		**automaton**
3		regular	
	right-linear		finite state
	$A \to aB$		
2		context-free	
	$A \to \alpha$		pushdown
			(+lifo stack)
1		context-sensitive	
	$(\beta_\ell, A, \beta_r) \to \alpha$		linear bounded
	$\alpha \to \beta$	$\lvert\beta\rvert \geqslant \lvert\alpha\rvert$	
	monotone		
0		recursively enumerable	
	$\alpha \to \beta$		turing machine

[M] Table 8.21

letter, symbol σ $0, 1$ a, b, c
alphabet Σ $\{a, b, c\}$
(finite, nonempty)

string, word $w \in \Sigma^*$
$w = a_1 a_2 \ldots a_n$, $a_i \in \Sigma$ $abbabb$
empty string λ, Λ, ε

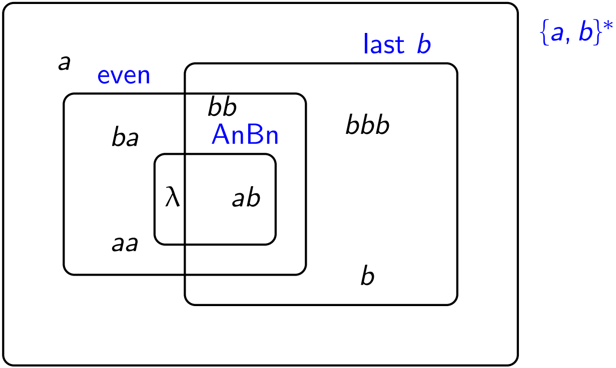
length $|x|$ $|\lambda| = 0$ $|xy| = |x| + |y|$

concatenation $a_1 \ldots a_m \cdot b_1 \ldots b_n$ $ab \cdot babb$
$w\lambda = \lambda w = w$ $(xy)z = x(yz)$

language $L \subseteq \Sigma^*$

Example

- $\{a, b\}^*$ all strings over $\{a, b\}$ λ, *baa*, *aaaaa*
- all strings of even length λ, *babbba*
- all strings with last letter b *bbb*, *aabb*
- $AnBn = \{a^n b^n \mid n \in \mathbb{N}\}$ λ, *aaabbb*



commutativity	$A \cup B = B \cup A$	...
associativity	$(A \cup B) \cup C = A \cup (B \cup C)$	
distributivity	$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	
idempotency	$A \cup A = A$	$A \cap A = A$
De Morgan	$(A \cup B)^c = A^c \cap B^c$	
unit	$A \cup \varnothing = A$	$A \cap U = A$
	$A \cap \varnothing = \varnothing$	$A \cup U = U$
involution	$(A^c)^c = A$	
complement	$A \cap A^c = \varnothing$	
		duality

brackets

priority $\quad ^c$ before $\cup, \cap$

$K \cap L \cup M$??

[M] page 4 DiTe, FI1

Definition

$K \cdot L = KL = \{\, xy \mid x \in K, y \in L \,\}$

$\{a, ab\}\{a, ba\} = \{aa, aba, abba\}$

one	$\{\lambda\}L = L\{\lambda\} = L$
zero	$\varnothing L = L\varnothing = \varnothing$
associative	$(KL)M = K(LM)$

$K^0 = \{\lambda\}$, $K^1 = K$, $K^2 = KK$, $\ldots$
$K^{n+1} = K^n K$.

Definition

$K^* = \bigcup_{n \geqslant 0} K^n$

$$K^n = \underbrace{K \cdot K \cdot \ldots \cdot K}_{n \text{ times}}$$

$$K^n = \{\, w_1 w_2 \ldots w_n \mid w_1, w_2, \ldots, w_n \in L \,\} \quad \text{fixed } n$$

$$K^* = \{\, w_1 w_2 \ldots w_n \mid w_1, w_2, \ldots, w_n \in L, n \in \mathbb{N} \,\}$$

Example

$\{a\}^* \cdot \{b\} = \{\lambda, a, aa, aaa, \ldots\} \cdot \{b\} = \{b, ab, aab, aaab, \ldots\}$

$(\ \{a\}^* \cdot \{b\}\ )^* = \{b, ab, aab, aaab, \ldots\}^* =$
$\{\lambda, b, ab, bb, aab, abb, bab, bbb, aaab, \ldots\}$

$(\ \{a\}^* \cdot \{b\}\ )^* = \{a, b\}^* \{b\} \cup \{\lambda\}$

family all languages that can be defined by
– type of automata
(deterministic) finite state DFA, NFA, pushdown PDA
– type of grammar
context-free grammar CFG, right linear
– certain operations
regular REG

Boolean operations: $\cup$, $\cap$, ^{c}
Regular operations: $\cup$, $\cdot$, *

family F *closed under* operation ∇:
if $K, L \in F$, then $K \nabla L \in F$.

RECOGNIZING, algorithm

$L_2 = \{ x \in \{a, b\}^* \mid n_a(x) > n_b(x) \}$

count a and b

deterministic [finite] automaton

GENERATING, description

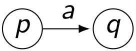
regular expression

$L_1 = (\{ab, bab\}^*\{b\})^*\{ab\} \cup \{b\}\{ba\}^*\{ab\}^*$

recursive definition

↪well-formed formulas

grammar

TYPE	grammar		automaton
3		regular	
	right-linear		finite state
	$A \to aB$		
2		context-free	
	$A \to \alpha$		pushdown
			(+lifo stack)
1		context-sensitive	
	$(\beta_\ell, A, \beta_r) \to \alpha$		linear bounded
	$\alpha \to \beta$	$\lvert\beta\rvert \geqslant \lvert\alpha\rvert$	
	monotone		
0		recursively enumerable	
	$\alpha \to \beta$		turing machine

[M] Table 8.21

– clever idea, intuition

– formal construction, specification

– show it works, e.g., induction

once the idea is understood,
the other parts might be boring

but essential to test intuition

examples help to get the message

L_1, L_2, L_3 are languages over some alphabet Σ.

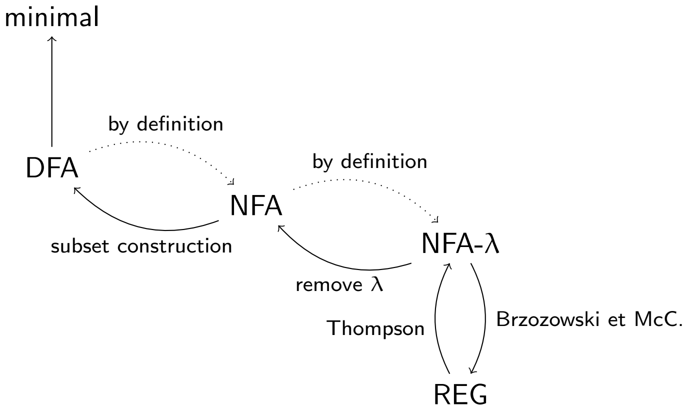
For each pair of languages below, what is their relationship?

Are they always equal? If not, is one always a subset of the other?

1. $L_1(L_2 \cap L_3)$ vs. $L_1L_2 \cap L_1L_3$
2. $L_1^* \cap L_2^*$ vs. $(L_1 \cap L_2)^*$
3. $L_1^*L_2^*$ vs. $(L_1L_2)^*$

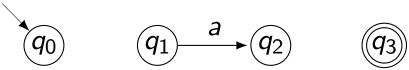
[M] Exercise 1.37

[1] A quiz is a brief assessment used in education to measure growth in knowledge, abilities, and/or skills. Wikipedia


minimal
by definition
DFA
by definition
NFA
subset construction
NFA-λ
remove λ
Thompson
Brzozowski et McC.
REG

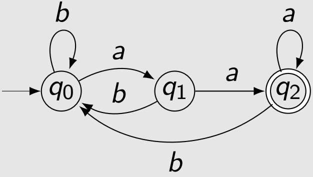
Section 2

Deterministic Finite Automata



Example

$L_1 = \{ x \in \{a, b\}^* \mid x \text{ ends with } aa \}$

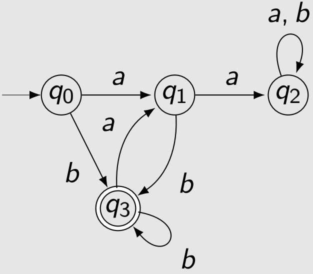


δ	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0

[M] E. 2.1

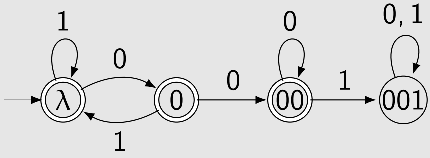
Example

$L_2 = \{\, x \in \{a, b\}^* \mid x \text{ ends with } b \text{ and does not contain } aa \,\}$



[M] E. 2.3

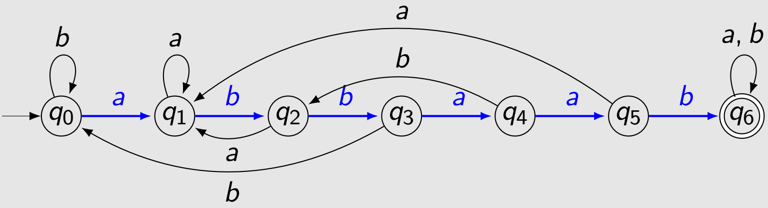
Example (Strings not containing 001)



[L] E 2.4

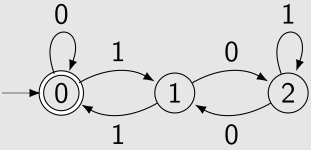
Example (Similar to Knuth-Morris-Pratt string search)

$L_3 = \{\, x \in \{a, b\}^* \mid x \text{ contains the substring } abbaab \,\}$



[M] E. 2.5

Example



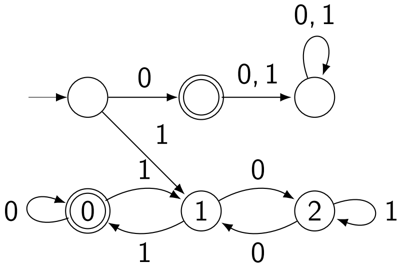
δ	0	1
x	$2x$	$2x+1$
0	0	1
1	2	0
2	1	2

$w \in \{0,1\}^* \longrightarrow val(w) \in \mathbb{N}$

$val(w0) = 2 \cdot val(w)$

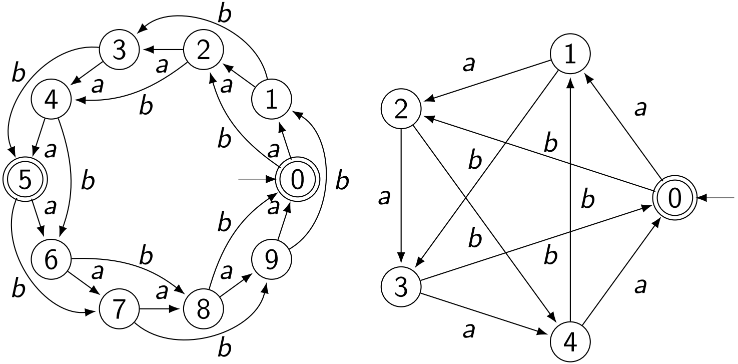
$val(w1) = 2 \cdot val(w) + 1$

states represent $val(w)$ modulo 3



[M] E. 2.7

$\{ x \in \{a, b\}^* \mid n_a(x) + 2n_b(x) \equiv 0 \mod 5 \}$



☒cs.SE Planar regular languages

Een student vroeg of alle automaten zonder kruisende takken getekend konden worden. De automaat rechts heeft de vorm van K_5 (de volledige graaf op vijf knopen) waarvan bekend is dat die niet planair is.

Dezelfde taal kan echter wel met een vlakke automaat verkregen worden (links). Er zijn talen zonder vlakke automaat.

Definition (DFA)

[deterministic] finite automaton 5-tuple $M = (Q, \Sigma, \delta, q_{in}, A)$,
- Q finite set *states*;
- Σ finite *input alphabet*;
- $\delta : Q \times \Sigma \to Q$ *transition function*;
- $q_{in} \in Q$ *initial* state;
- $A \subseteq Q$ *accepting* states.

[M] D 2.11 Finite automaton, slightly different order

[L] D 2.1 Deterministic finite accepter, has 'final' states

DFA $M = (Q, \Sigma, \delta, q_{in}, A)$

Definition

extended transition function $\delta^* : Q \times \Sigma^* \to Q$, such that
– $\delta^*(q, \lambda) = q$ for $q \in Q$
– $\delta^*(q, y\sigma) = \delta(\ \delta^*(q, y), \sigma\ )$ for $q \in Q, y \in \Sigma^*, \sigma \in \Sigma$

[M] D 2.12 [L] p.40/1

Theorem

$q = \delta^*(p, w)$ iff there is a path in [the transition graph of] M from p to q with label w.

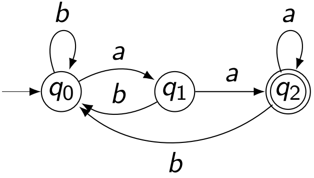
[L] Th 2.1

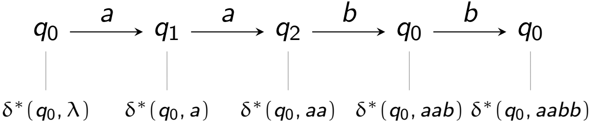
Definition

The *language accepted* by $M = (Q, \Sigma, \delta, q_{in}, A)$ is the set
$L(M) = \{\ x \in \Sigma^* \mid \delta^*(q_{in}, x) \in A\ \}$

[M] D 2.14 [L] D 2.2

Extended transition function





$\delta^*(q_0, aabb) = q_0$:

$\delta^*(q_0, \lambda) = q_0$

$\delta^*(q_0, a) = \delta^*(q_0, \lambda\, a) = \delta(\delta^*(q_0, \lambda), a) = \delta(q_0, a) = q_1$

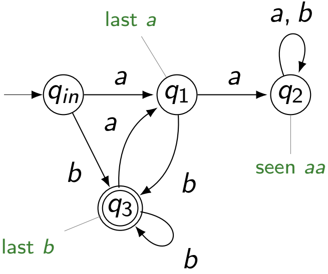
$\delta^*(q_0, aa) = \delta(\delta^*(q_0, a), a) = \delta(q_1, a) = q_2$

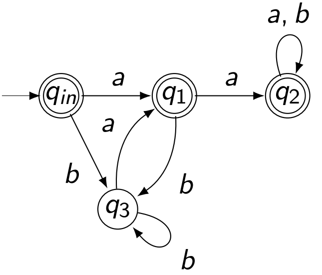
$\delta^*(q_0, aab) = \delta(\delta^*(q_0, aa), b) = \delta(q_2, b) = q_0$

$\delta^*(q_0, aabb) = \delta(\delta^*(q_0, aab), b) = \delta(q_0, b) = q_0$

Intro: complement

$L_2 = \{\ x \in \{a, b\}^* \mid x \text{ ends with } b \text{ and does not contain } aa\ \}$





$$\neg(P \wedge Q) = \neg P \vee \neg Q$$

$L_2^c = \{\ x \in \{a, b\}^* \mid x \text{ does not end with } b \text{ or contains } aa\ \}$

Complement, construction

Construction

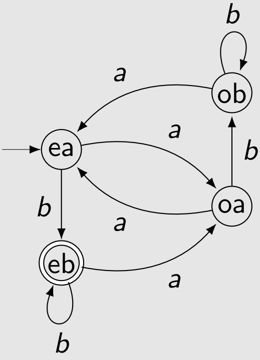
DFA $M = (Q, \Sigma, \delta, q_{in}, A)$,

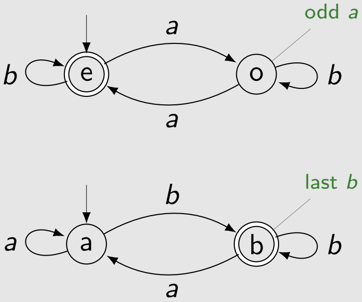
let $M^c = (Q, \Sigma, \delta, q_{in}, Q - A)$

Theorem

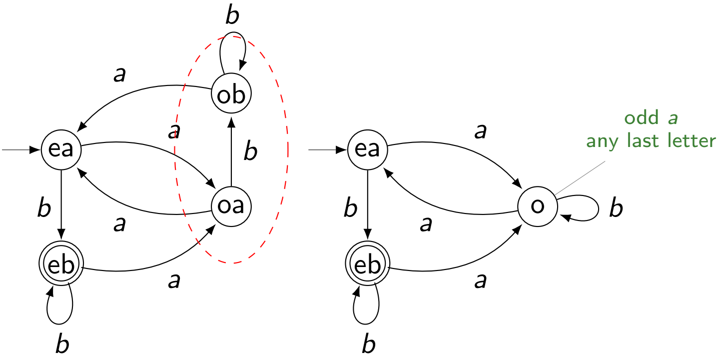
$L(M^c) = \Sigma^* - L(M)$

Example (Even number of *a*, and ending with *b*)





Even number of *a and* ending with *b*



DFA $M_i = (Q_i, \Sigma, \delta_i, q_i, A_i)$

Product construction

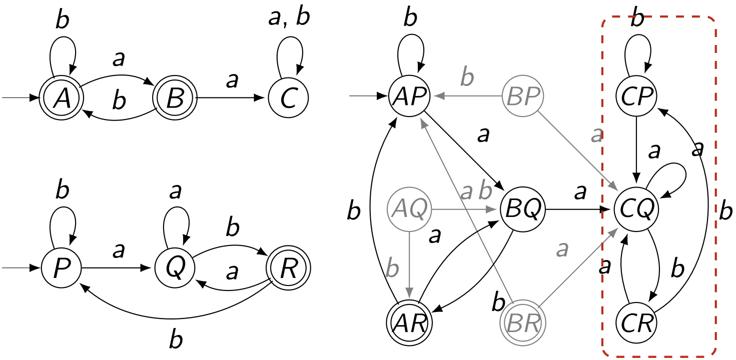
construct DFA $M = (Q, \Sigma, \delta, q_0, A)$ such that
- $Q = Q_1 \times Q_2$
- $q_0 = (q_1, q_2)$
- $\delta(\ (p, q), \sigma) = (\ \delta_1(p, \sigma), \delta_2(q, \sigma)\ )$
- A as needed

Theorem (2.15 Parallel simulation)

- $A = \{(p, q) \mid p \in A_1 \text{ or } q \in A_2\}$, *then* $L(M) = L(M_1) \cup L(M_2)$
- $A = \{(p, q) \mid p \in A_1 \text{ and } q \in A_2\}$, *then* $L(M) = L(M_1) \cap L(M_2)$
- $A = \{(p, q) \mid p \in A_1 \text{ and } q \notin A_2\}$, *then* $L(M) = L(M_1) - L(M_2)$

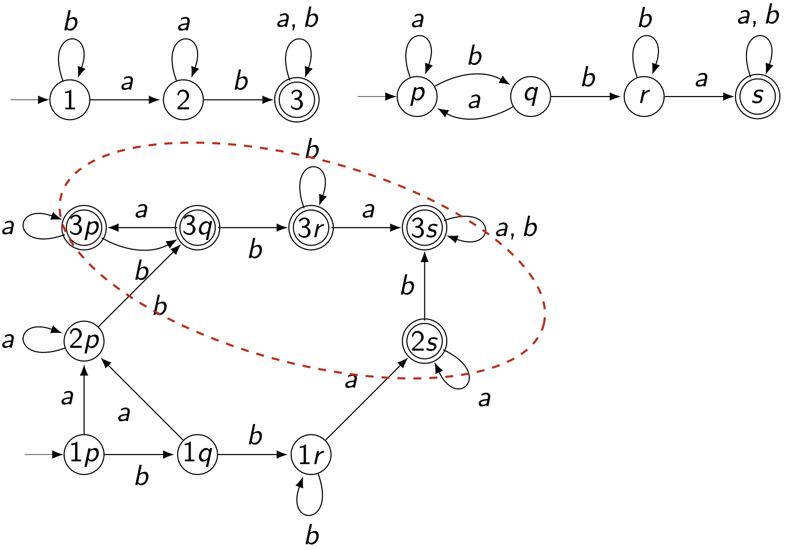
[M] Sect 2.2

not substring *aa*



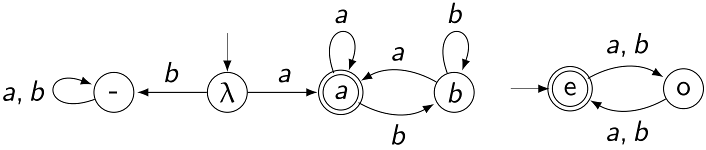
ends with *ab*

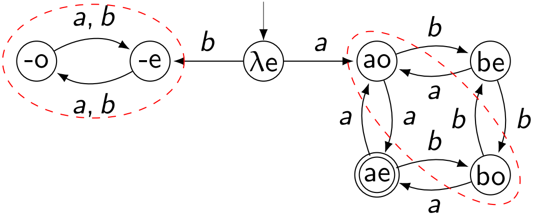
[M] E 2.16



[M] E. 2.18, see also ↪subset construction

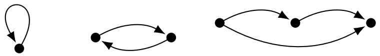
$K = \{\ w \in \{a, b\}^* \mid w$ begint en eindigt met een a, en $|w|$ is even $\}$





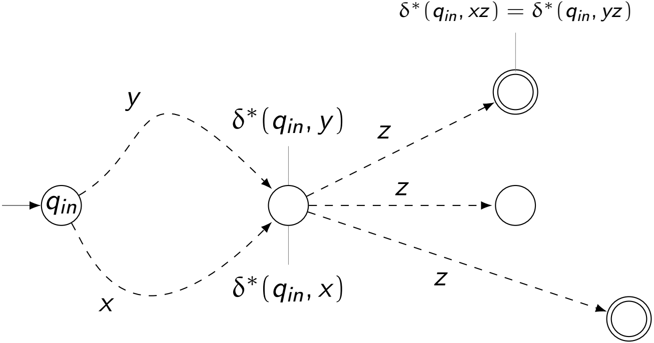
- Give 2-state DFA for each of the languages over $\{a, b\}$
  - strings with even number of a's
  - strings with at least one b
- Use the product construction to obtain a 4-state DFA for the language of strings with even number of a's or at least one b
- Investigate which states can be merged

R equivalence relation on A
– reflexive aRa for all ...
– symmetric aRb then bRa
– transitive aRb and bRc then aRc



equivalence class $[x]_R = \{ y \in A \mid yRx \}$
partition A

[M] Sect. 1.3


$\delta^*(q_{in}, xz) = \delta^*(q_{in}, yz)$
y
$\delta^*(q_{in}, y)$
z
z
q_{in}
z
$\delta^*(q_{in}, x)$
x

arbitrary $L \subseteq \Sigma^*$, $x, y \in \Sigma^*$

x, y *distinguishable* wrt L exists $z \in \Sigma^*$ with

$xz \in L$ and $yz \notin L$ or $xz \notin L$ and $yz \in L$

Definition

$L/x = \{ z \in \Sigma^* \mid xz \in L \}$

x and y are L-*indistinguishable* iff $L/x = L/y$

equivalence relation on Σ^* $x \equiv_L y$

- right invariant $x \equiv_L y$ implies $xz \equiv_L yz$
- L union of equivalence classes
- may exist infinitely many classes

[M] D 2.20 uses I_L

$K = \{ w \in \{a, b\}^* \mid n_a(w)$ is even, or last letter w is $b \}$

Example $L = AnBn = \{ a^n b^n \mid n \geqslant 0 \}$

– prefixes

each a^i

$a^i \not\equiv a^j \quad a^i \cdot b^i \in L$ vs $a^j \cdot b^i \notin L \quad i \neq j$

a^i vs. $x \quad x \cdot abb^i \in L$ iff $x = a^i$

$a^{i+k} b^i \equiv_L a^{j+k} b^j \quad i, j > 0$ at least one b

– non-prefixes $x\,ba\,y$ or $a^i b^j$, $j > i$

all equivalent, cannot be extended

quotients

– $L/a^i = \{ a^k b^{i+k} \mid k \geqslant 0 \}$

– $L/a^{i+k} b^i = \{ b^k \} \quad i > 0$

– $L/a^i b^j = L/xbay = \varnothing \quad j > i$

[M] E 2.37

$L \subseteq \Sigma^*$, $x, y \in \Sigma^*$
$L = L(M)$, M with initial state q_{in}

Theorem

If $\delta^*(q_{in}, x) = \delta^*(q_{in}, y)$, then $x \equiv_L y$.

$x \equiv_M y$ end in same state for M

– right invariant $x \equiv_M y$ implies $xz \equiv_M yz$
– L union of equivalence classes
– finitely many classes 'finite index'

at least as many states as $\equiv_L$-classes

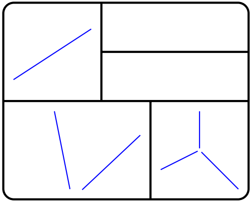
[M] Thm 2.21

$L = L(M)$

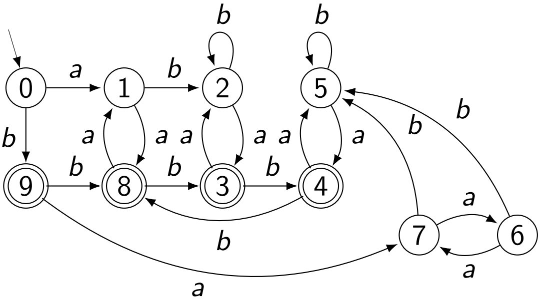
$\equiv_M$ state $\delta^*(q_{in}, x)$

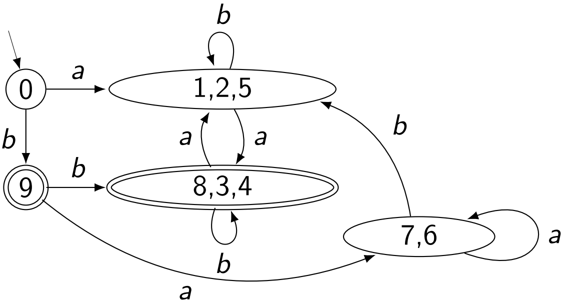
$\equiv_L$ "future" L/x

$x \equiv_M y$, then $x \equiv_L y$.



Σ^*





[M] Fig 2.42

Theorem

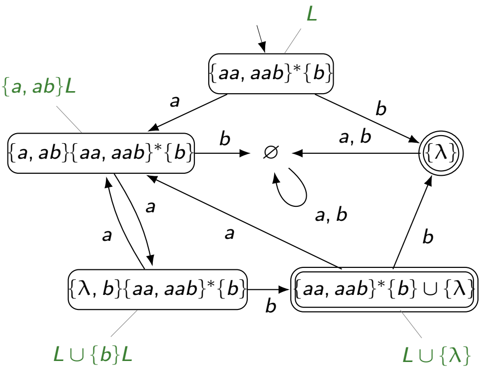
L is regular iff $\equiv_L$ is of finite index

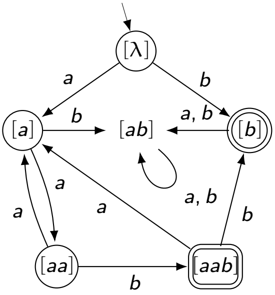
proof sketch.
Use the $\equiv_L$-classes as states.

– $q_{in} = [\lambda]_L$
– final states $[x]_L$ with $x \in L$.
– transition relation $\delta([x]_L, \sigma) = [x\sigma]_L$

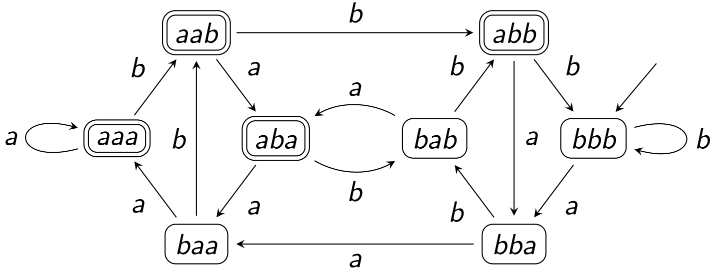
$L = \{aa, aab\}^*\{b\}$

$L/\sigma = \{\ z \in \Sigma^* \mid \sigma z \in L\ \}$ $\quad L \xrightarrow{\sigma} L/\sigma$ $\qquad [x] \xrightarrow{\sigma} [x\sigma]$





[M] E 2.22 see ↪E 3.6

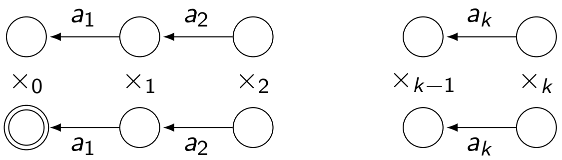


[M] E. 2.24

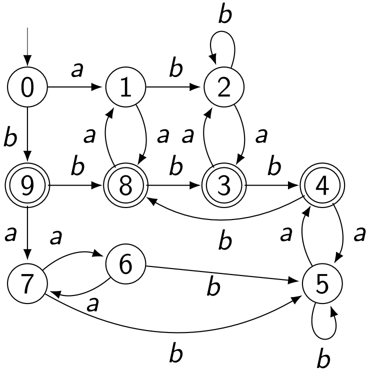
ALGORITHM mark pairs of non-equivalent states

start by marking pairs (p, q) where exactly one p, q in A

repeat
    for each unmarked pair (p, q)
        check whether there is a σ such that $(\,\delta(p,\sigma), \delta(p,\sigma)\,)$ is marked
        then mark (p, q)
until this pass does not mark new pairs

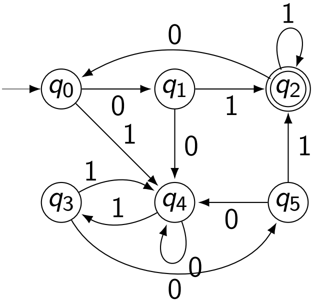


[M] Algo 2.40



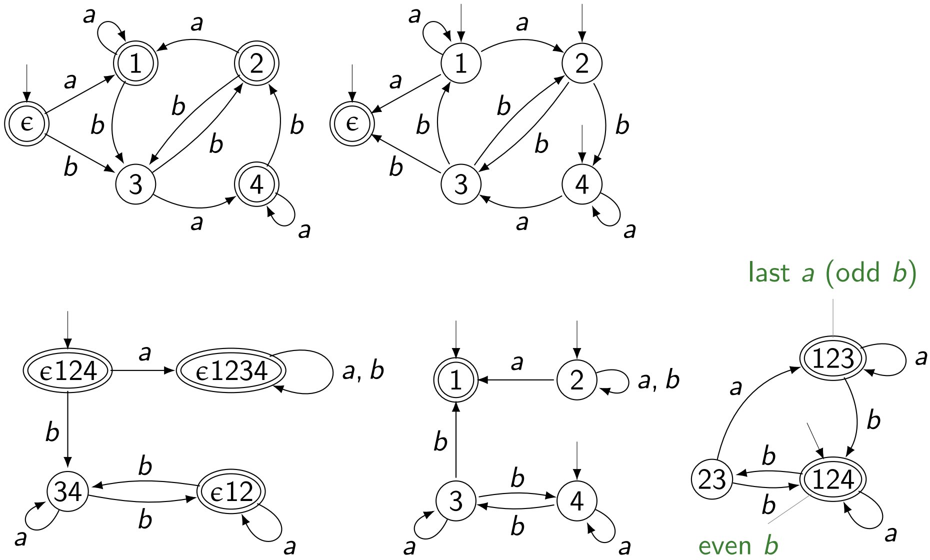
	0	1	2	3	4	5	6	7	8
1	1								
2	1	.							
3	0	0	0						
4	0	0	0	.					
5	1	.	.	0	0				
6	1	1	1	0	0	1			
7	1	1	1	0	0	1	.		
8	0	0	0	.	.	0	0	0	
9	0	0	0	1	2	0	0	0	1

[M] Fig 2.42



Antal Iványi, Algorithms of Informatics

Example: Brzozowski minimization

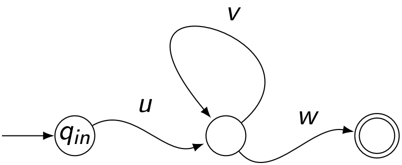


ABOVE

Brzozowski observes that one can minimize a DFA by performing the following operations twice: invert (mirror), then determinize.

It is rather magical that this indeed works.

The method is in theory rather unfavourable, because of the exponentiation when detrminizing, but in practice seems not too slow.



[M] Fig. 2.28

Theorem

$\forall$ *for every regular language* L

$\exists$ *there exists a constant* $n \geqslant 1$

such that

$\forall$ *for every* $z \in L$

with $|z| \geqslant n$

$\exists$ *there exists a decomposition* $z = uvw$

with (1) $|uv| \leqslant n$,

and (2) $|v| \geqslant 1$

such that

$\forall$ *(3) for all* $i \geqslant 0$, $uv^iw \in L$

if $L = L(M)$ then $n = |Q|$.

[M] Thm. 2.29

Example

AnBn is not accepted by DFA.

[M] E 2.30

AeqB same argument, or closure properties

Example

$\{ x \in \{a, b\}^* \mid n_a(x) > n_b(x) \}$ is not accepted by DFA

[M] E 2.31

We prove that the language AnBn is not regular, by contradiction.

Assume that $L =$ AnBn is regular.
Then there exists a constant p for L as in the pumping lemma.
Take $z = a^p b^p$. Then $z \in L$, and $|z| = 2p \geqslant p$.
Thus there exists a decomposition $z = uvw$ such that $|uv| \leqslant n$ with v nonempty, and $uv^i w \in L$ for every i.
Consider $i = 0$. Observe v consists of a's only. deleting v from the string z will delete a number of a's. So $uv^0 w$ is of the form $a^m b^n$ with $m < n$.
This string is not in L; a contradiction.
So, L is not regular.

Exactly the same argument can be used (verbatim) to prove that $L =$ AeqB is not regular.

We can also apply closure properties of REG to see that AeqB is not regular, as follows.

Assume AeqB is regular. Then also AnBn $=$ AeqB $\cap\, a^* b^*$ is regular, as regular languages are closed under intersection.
This is a contradiction, as we just have argued that AnBn is not regular.
Thus, also AeqB is not regular.

$\{ a^i b^j c^k \mid i = 0 \text{ or } j = k \} = \{b\}^*\{c\}^* \cup \{a\}\{ a^i b^j c^j \mid i, j \geqslant 0 \}$

– can be pumped, as in the pumping lemma

– is not accepted by DFA

[M] E 2.39

$L \subseteq \{a\}^*$

Example

$L = \{ a^{i^2} \mid i \geqslant 0 \}$ is not accepted by DFA

[M] E 2.32

Fun fact

$L^4 = \{a\}^*$

Lagrange's four-square theorem

The length of uv^2w cannot be a square: we will show it is strictly in between two consecutive squares.
$|uv^2w| = |z| + |v| > |z| = n^2$.
$|uv^2w| = |z| + |v| \leqslant n^2 + n < (n+1)^2$.

$M = (Q, \Sigma, \delta, q_{in}, A)$
membership problem $x \in L(M)$?

Theorem

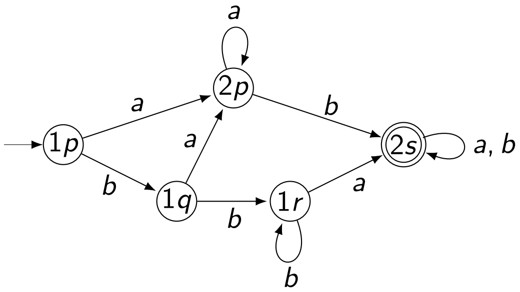
The following two problems are decidable
1. *Given a* DFA *M, is $L(M)$ nonempty?*
2. *Given a* DFA *M, is $L(M)$ infinite?*

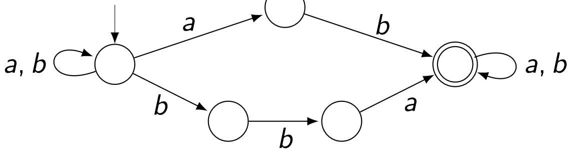
[M] E 2.34

Section 3

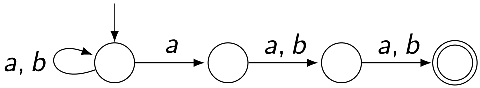
Non-Determinism

Non-determinism:
possibly many computations on given input
accept input when at least one of these computations accepts.

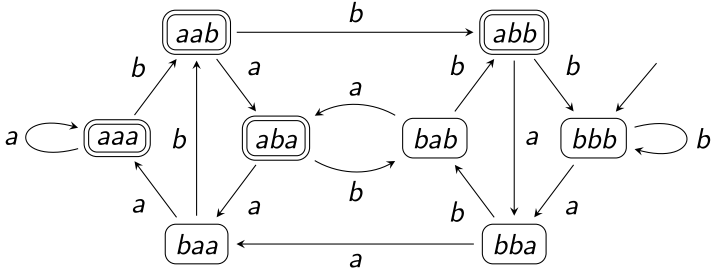




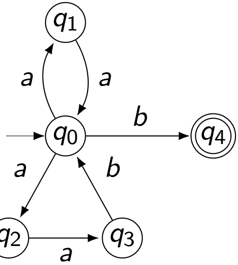
[M] see ↪E.2.18 (product construction)

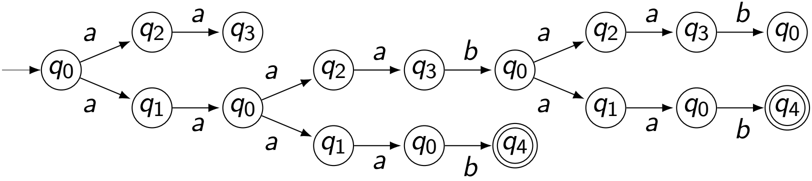


Also ↪deterministic



$n + 1$ versus 2^n states.





[M] E 3.6. also ↪E 2.22

5-tuple $M = (Q, \Sigma, \delta, q_{in}, A)$

Definition (↪DFA)

[deterministic] finite automaton

– $\delta : Q \times \Sigma \to Q$ *transition function*;

Definition (NFA)

non-deterministic finite automaton

– $\delta \subseteq Q \times \Sigma \times Q$ *transition relation*;

Definition (NFA-λ)

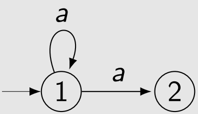
finite automaton with λ-transitions

– $\delta \subseteq Q \times (\Sigma \cup \{\lambda\}) \times Q$ *transition relation*;

$\delta \subseteq Q \times \Sigma \times Q \quad \longleftrightarrow \quad \delta : Q \times \Sigma \to 2^Q$

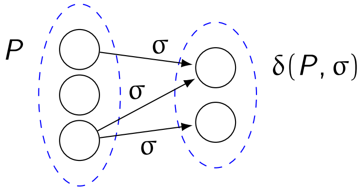
$\delta(p, \sigma) = \{ q \in Q \mid (p, \sigma, q) \in \delta \}.$

Example



δ	a
1	$\{1, 2\}$
2	$\varnothing$

$\delta(P, \sigma) = \bigcup_{p \in P} \delta(p, \sigma) = \{q \in Q \mid (p, \sigma, q) \in \delta, \text{ for some } p \in P\}.$



NFA $M = (Q, \Sigma, \delta, q_{in}, A)$

Definition

extended transition function $\delta^* : Q \times \Sigma^* \to 2^Q$, such that
– $\delta^*(q, \lambda) = \{q\}$ for $q \in Q$
– $\delta^*(q, y\sigma) = \delta(\, \delta^*(q, y), \sigma \,)$ for $q \in Q, y \in \Sigma^*, \sigma \in \Sigma$

$(q, y\sigma, p) \in \delta^*$ if $(q, y, r) \in \delta^*$ and $(r, \sigma, p) \in \delta$ for $q \in Q, y \in \Sigma^*, \sigma \in \Sigma$

Theorem

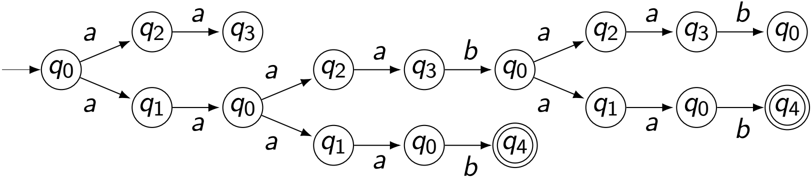
$q \in \delta^*(p, w)$ iff there is a path in [the transition graph of] M from p to q with label w.

$\delta^*(q_{in}, w) = \varnothing$ no path for w from initial state

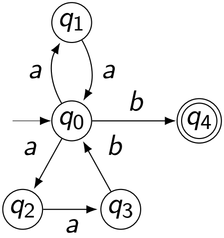
Definition

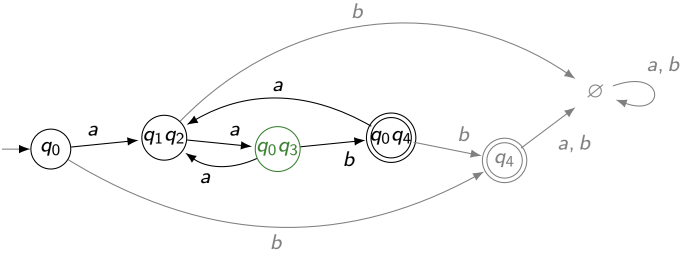
The *language accepted* by $M = (Q, \Sigma, \delta, q_{in}, A)$ is the set
$L(M) = \{\, x \in \Sigma^* \mid \delta^*(q_{in}, x) \cap A \neq \varnothing \,\}$

Folding the computation tree

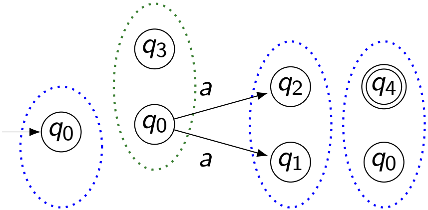


$\{q_0\}$ $\{q_1, q_2\}$ $\{q_0, q_3\}$ $\{q_1, q_2\}$ $\{q_0, q_3\}$ $\{q_0, q_4\}$ $\{q_1, q_2\}$ $\{q_0, q_3\}$ $\{q_0, q_4\}$





[M] E 3.6 ánd E 3.21



Subset construction

NFA $M = (Q, \Sigma, \delta, q_0, A)$

construct DFA $M_1 = (Q_1, \Sigma, \delta_1, q_1, A_1)$

– $Q_1 = 2^Q$

– $q_1 = \{q_0\}$

– $A_1 = \{\ q \in Q_1 \mid q \cap A \neq \varnothing\ \}$

– $\delta_1(q, \sigma) = \bigcup_{p \in q} \delta(p, \sigma)$

[M] Th 3.18

ABOVE

The subset construction (or powerset construction) can be used to transform a non-deterministic finite state automaton (without λ) into an equivalent deterministic automaton. The states of the new automaton consist of sets of states of the original automaton (hence powerset). The set collects all possible states that the original automaton could have ended in with the same input.

Note that the constructed automaton may be exponential in size compared to the nondetereministic one.

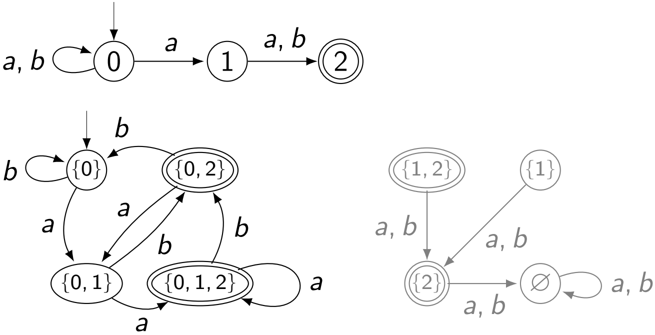
Reference

M.O. Rabin, D. Scott. Finite automata and their decision problems. IBM Journal of Research and Development. 3 (2): 114–125, 1959. doi:10.1147/rd.32.0114

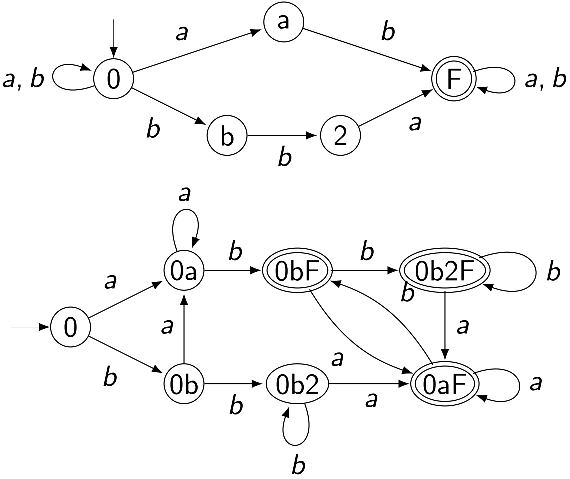
BELOW

Unreachable states can be omitted.

Reachable states



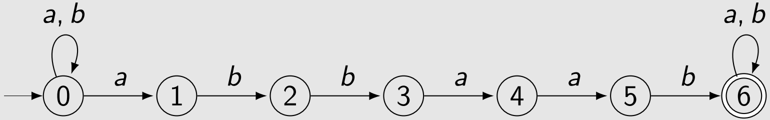
also $\hookrightarrow$3rd from the end



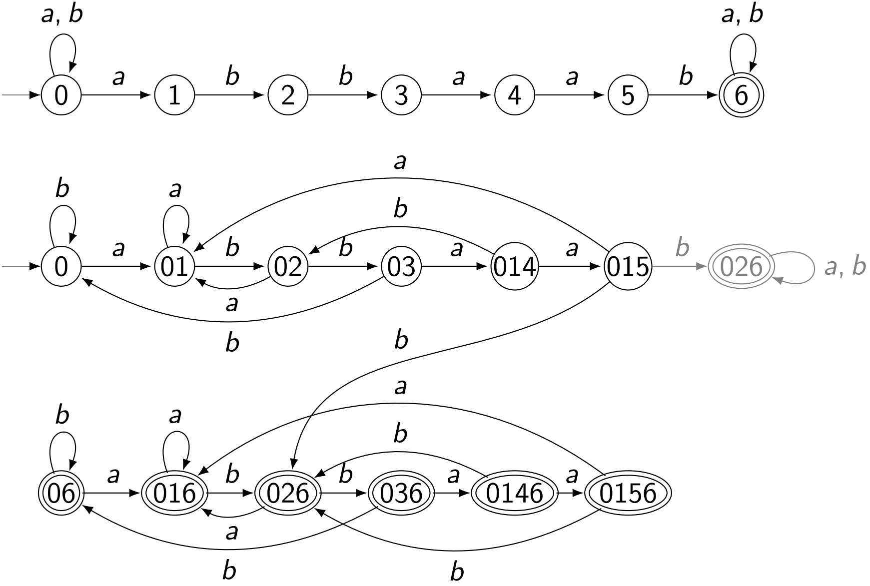
[M] language from ↪E 2.18

Example

$L_3 = \{ x \in \{a, b\}^* \mid x \text{ contains the substring } abbaab \}$



[M] ↪E. 2.5 (deterministic)

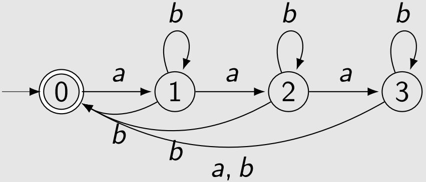

a, b
a
b
b
a
a
b
a, b
0
1
2
3
4
5
6
a
b
a
b
b
a
a
b
0
01
02
03
014
015
026
a, b
a
b
b
a
b
a
b
a
b
b
a
a
06
016
026
036
0146
0156
a
b
b

ABOVE

Illustration.

The determinization algorithm for the nondeterministic automaton for "has substring x" will always generate two copies of x. In the last copy all nodes are accepting, and they can be reduced to one node.

Example ($n = 4$)

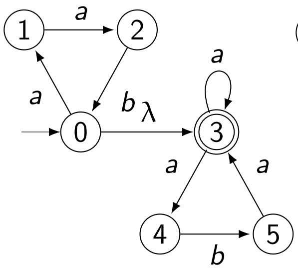


all 2^n subsets are reachable, nonequivalent, states.

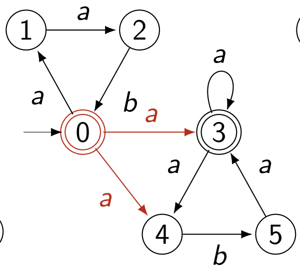
Theoretically, the subset construction used on a set Q with n nodes constructs an automaton with state set 2^Q with 2^n nodes. In practice however, not all nodes are really necessary.
Usually not all nodes are reachable, and we omit those from the construction.
Sometimes nodes can be joined because they are equivalent.

This worst-case example however needs all nodes. So the determinization algorithm applied to a finite state automaton in the worst case will blow-up the original nondeterministic automaton exponentially in size.

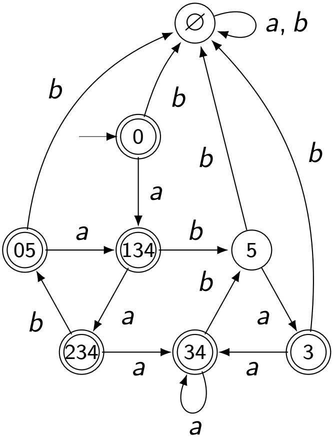
$\{aab\}^*\{a, aba\}^*$



NFA-λ

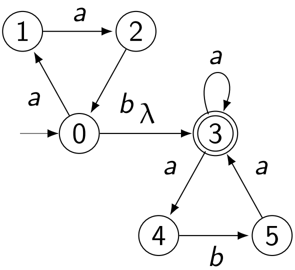


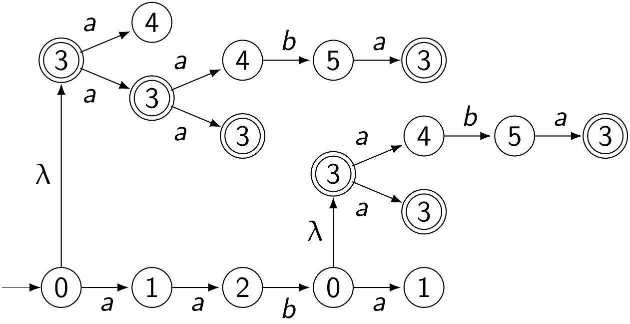
NFA



DFA

Computation tree when λ's are around

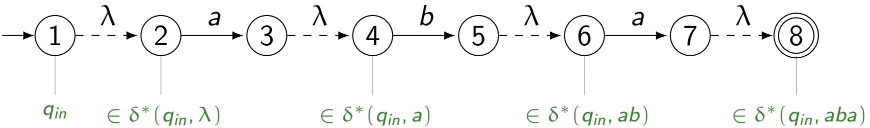




NFA-λ $M = (Q, \Sigma, \delta, q_{in}, A)$ $S \subseteq Q$

Definition

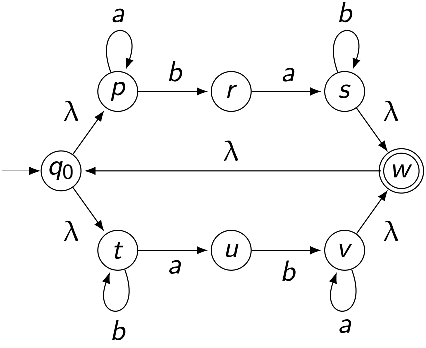
– $S \subseteq \Lambda(S)$
– $p \in \Lambda(S)$ and $(p, \lambda, q) \in \delta$, then $q \in \Lambda(S)$



Definition

– $\delta^*(q, \lambda) = \Lambda(\{q\})$ $q \in Q$
– $\delta^*(q, y\sigma) = \Lambda(\ \delta(\ \delta^*(q, y), \sigma\ )\ )$ $q \in Q,\ y \in \Sigma^*,\ \sigma \in \Sigma$

[M] D 3.13 & 3.14



λ

$\Lambda(\{q_0\}) = \{q_0, p, t\}$

$\delta^*(q_0, \lambda) = \{q_0, p, t\}$

$\lambda \cdot a$

$\delta(\{q_0, p, t\}, a) = \{p, u\}$

$\delta^*(q_0, a) = \Lambda(\{p, u\}) = \{p, u\}$

$a \cdot b$

$\delta(\{p, u\}, b) = \{r, v\}$

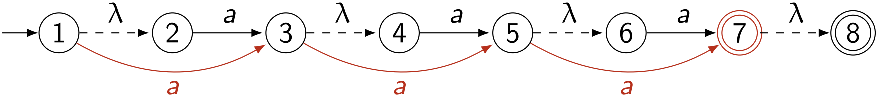
$\delta^*(q_0, ab) = \Lambda(\{r, v\}) =$
$\{r, v, w, q_0, p, t\}$

$ab \cdot a$

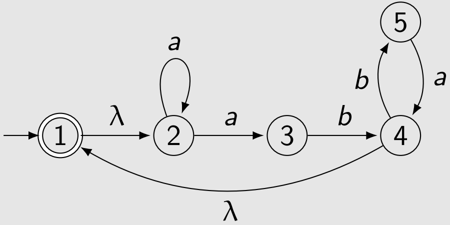
$\delta(\{r, v, w, q_0, p, t\}, a) = \{s, v, p, u\}$

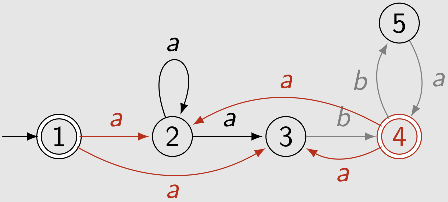
$\delta^*(q_0, aba) = \Lambda(\{s, v, p, u\}) =$
$\{s, w, q_0, p, t, v, u\}$

[M] E 3.15

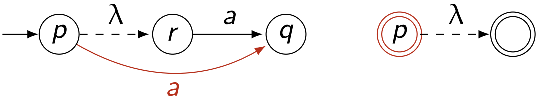


Example





[M] E 3.19, but less edges!



λ-removal

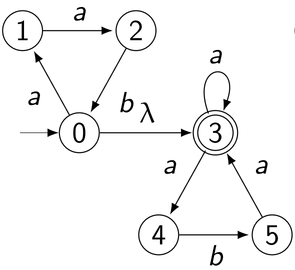
NFA-λ $M = (Q, \Sigma, \delta, q_{in}, A)$

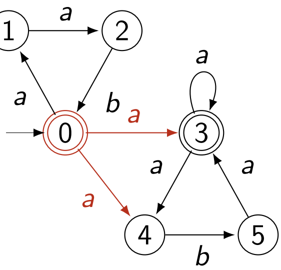
construct NFA $M_1 = (Q, \Sigma, \delta_1, q_{in}, A_1)$

– whenever $r \in \Lambda_M(\{p\})$ and $(r, a, q) \in \delta$, add $(p, a, q) \in \delta_1$

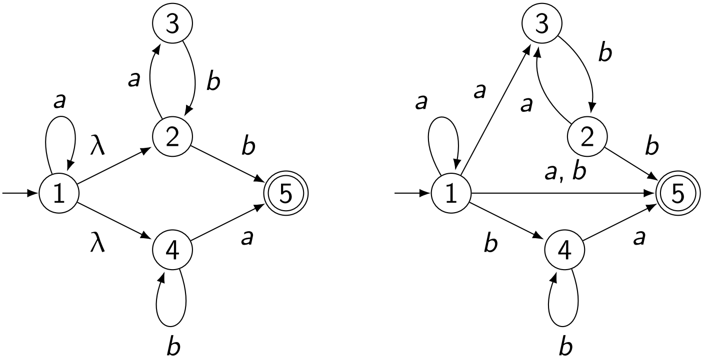
– whenever $\Lambda_M(\{p\}) \cap A \neq \varnothing$, add p to A_1.

$L = \{aab\}^*\{a, aba\}^*$



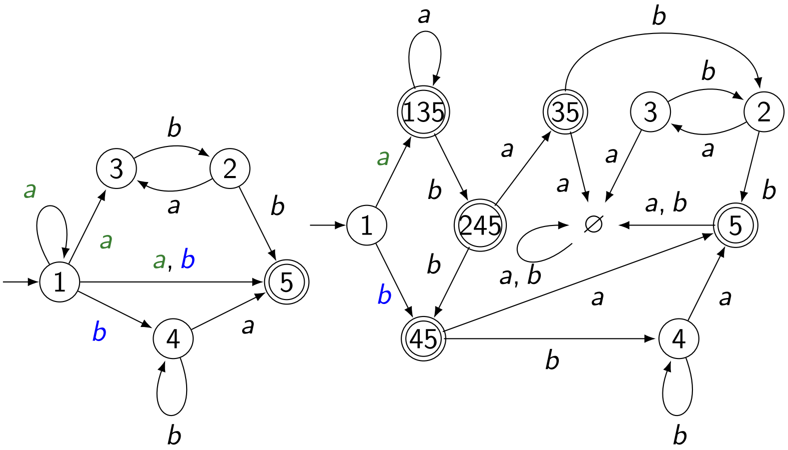


$\{a\}^* [\, \{ab\}^* \{b\} \cup \{b\}^* \{a\} \,]$



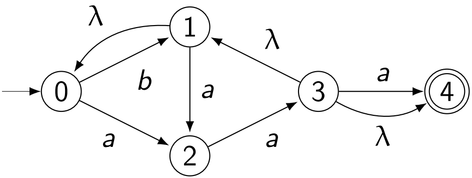
[M] E 3.23

$\{a\}^*[\,\{ab\}^*\{b\} \cup \{b\}^*\{a\}\,]$



[M] E 3.23 ctd.

Construct an equivalent DFA, applying the appropriate algorithms.



Definition (REG)

– $\varnothing$ is in REG.

– $\{a\}$ in REG,   for every $a \in \Sigma$

– if L_1 and L_2 in REG,
then so are $L_1 \cup L_2$, $L_1 \cdot L_2$, and L_1^*.

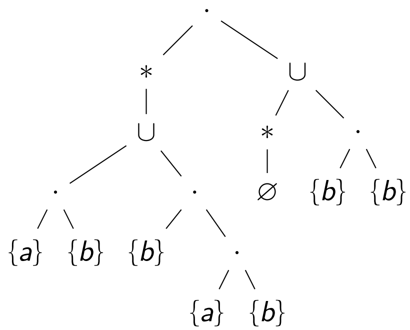
[M] D. 3.1 $\mathcal{R}$

Smallest set[family] of languages that
– contains $\varnothing$ and $\{a\}$ for $a \in \Sigma$, and   basis
– is *closed under* union, concatenation and star.   induction
[M] cf. E 1.20

From elementary components

$\{ab, bab\}^*\{\lambda, bb\}$

$(\ (\{a\} \cdot \{b\}) \cup (\{b\} \cdot \{a\} \cdot \{b\})\ )^* \cdot (\ \varnothing^* \cup (\{b\} \cdot \{b\})\ )$



– $\varnothing$, Λ, and a are RegEx (for all $a \in \Sigma$)
– if E_1 and E_2 are RegEx, then so are E_1^*, $(E_1 + E_2)$, and $(E_1 E_2)$

expression [syntax] vs its language [semantics]

E string	$L(E)$ language
$\varnothing$	$\varnothing$
Λ	$\{\lambda\}$
a	$\{a\}$
$(E_1 + E_2)$	$L(E_1) \cup L(E_2)$
$(E_1 E_2)$	$L(E_1) \cdot L(E_2)$
E_1^*	$L(E_1)^*$

we say

$E_1 = E_2$ iff $L(E_1) = L(E_2)$
$w \in E$ iff $w \in L(E)$

trivial	$E + 0 = 0 + E = E$	where $0 = \varnothing$
	$E \cdot 0 = 0 \cdot E = 0$	
	$E \cdot 1 = 1 \cdot E = E$	where $1 = \Lambda$
associative	$(E_1 + E_2) + E_3 = E_1 + (E_2 + E_3)$	
	$(E_1 \cdot E_2) \cdot E_3 = E_1 \cdot (E_2 \cdot E_3)$	
distributive	$E(E_1 + E_2) = EE_1 + EE_2$	
	$(E_1 + E_2)E = E_1E + E_2E$	
commutative	$E_1 + E_2 = E_2 + E_1$	
unrolling	$E^* = 1 + EE^* = 1 + E^*E$	$*$-rules
denesting	$(E_1 + E_2)^* = E_1^* \cdot (E_2 \cdot E_1^*)^*$	
sliding	$(E_1 \cdot E_2)^* \cdot E_1 = E_1 \cdot (E_2 \cdot E_1)^*$	
cyclic Zn	$E^* = (1 + E + E^2 + \ldots E^{n-1}) \cdot (E^n)^*$	
idempotency	$E + E = E$	
	$(E^*)^* = E^*$	

based on Jacques Sakarovitch: Automata and expressions consider real numbers

motivation $\frac{1}{1-x} = 1 + x + x^2 + x^3 + \dots$

define $x^* = \frac{1}{1-x}$ and check the axioms

$x^* = 1 + xx^*$ $E^* = 1 + EE^*$
$\frac{1}{1-x} = 1 + x\frac{1}{1-x}$ iff $1 = (1-x) + x$

$(x+y)^* = x^*(yx^*)^*$ $(E_1 + E_2)^* = E_1^* \cdot (E_2 \cdot E_1^*)^*$
indeed $1 - (x+y) = (1-x)(1 - \frac{y}{1-x}) = (1-x)(1-yx^*)$

unfortunately, no idempotency: $x + x = x$ nor $(x^*)^* = x^*$

Extras. To illustrate that the rules in the Kleene algebra are more generally applicable than just regular languages.

The infinite sum $\sum x^i$ looks like the expression for Kleene star. Its value as geometric series equals $\frac{1}{1-x}$, for real x, $|x| < 1$.

If we define that fraction as x^* for $x \in \mathbb{R}$, then the rules unrolling and denesting are again valid!

– Odd number of *a*

$bba_0ba_1bbba_2bba_1a_2bb$

$b^*ab^*(ab^*ab^*)^*$ $\quad b^*ab^*(ab^*ab^*)(ab^*ab^*)$

$b^*a(ab^*a+b)^*$ $\quad b^*ab^*(ab^*a)b^*(ab^*a)b^*$

[M] cf. E 3.2

– Ending with *b*, no *aa*

$bb(ab)bbb(ab)(ab)b$

$(b+ab)^*(b+ab)$ at least once

[M] cf. E 3.3, see ↪E. 2.3

– No *aa* may also end in *a*

$(b+ab)^*(\Lambda+a)$

– Even number of both *a* and *b*

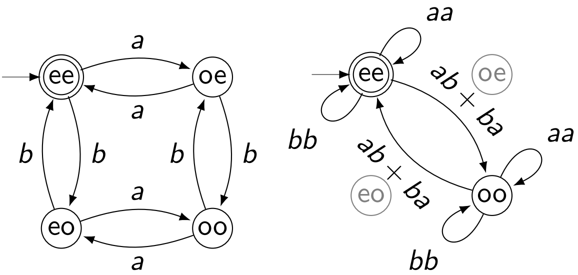
tw letters together

aa and *bb* keep both numbers even [odd]

ab and *ba* switch between even and odd, for both numbers

$$(\ aa + bb + (ab + ba)(aa + bb)^*(ab + ba) \ )^*$$

[M] E 3.4, see ↪Brzozowski *et* McCluskey



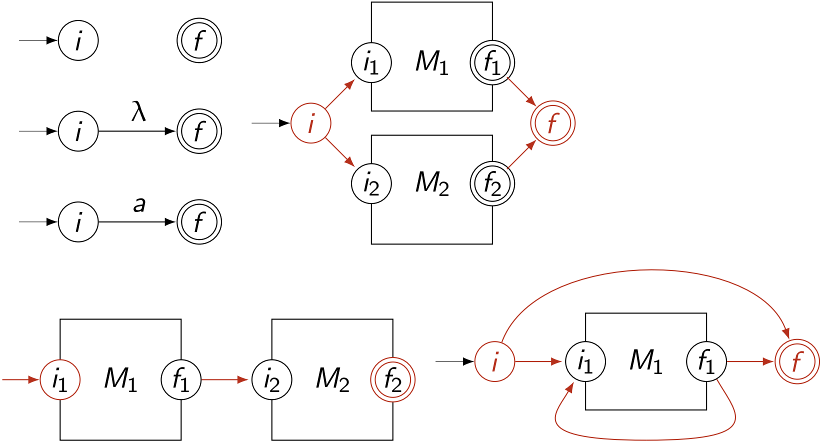
Theorem (Kleene)

Finite state automata and regular expressions specify the same familiy of languages.

- from RegEx to FSA
  ↪Thompson's construction
  ↪Brozowski derivatives (☒ example only)
- from FSA to RegEx
  ↪McNaughton and Yamada
  State elimination ↪Brzozowski et McCluskey
  Solving linear ↪language equations

Theorem

If L is a regular language, then there exists a NFA-λ *that accepts L.*



[M] Th 3.25 [L] Th 3.1

ABOVE

The construction we use is such that each automaton has exactly one accepting node. The initial state has no ingoing edges, the accepting state has no outgoing edges. This makes the construction rather "safe". Usually the automaton can be optimized.

⊠ DFA directly from RegEx

$L \xrightarrow{\sigma} L/\sigma$ see ↪Distinguishing States

$\varnothing/\sigma = \varnothing$
$\Lambda/\sigma = \varnothing$
$a/\sigma = \lambda \quad a = \sigma$
$b/\sigma = \varnothing \quad b \neq \sigma$
$(E_1 + E_2)/\sigma = E_1/\sigma + E_2/\sigma$
$(E_1 E_2)/\sigma = E_1/\sigma\ E_2$
$(E_1 E_2)/\sigma = E_1/\sigma\ E_2 + E_2/\sigma \quad \text{if } \lambda \in E_1$
$E^*/\sigma = E/\sigma\ E^*$

can be extended to intersection and negation

problem: checking equality

$L = \{aab\}^*\{a, aba\}^*$

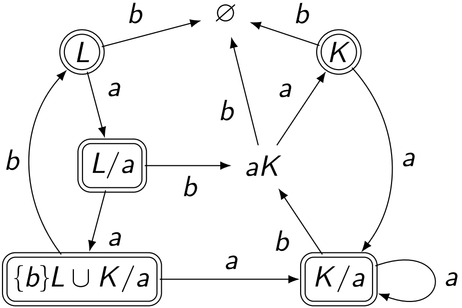
$K = \{a, aba\}^*$

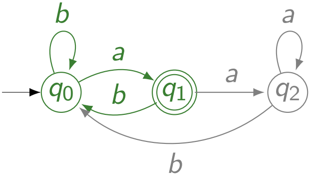
$K/a = \{a, aba\}/a \cdot K = \{\lambda, ba\}K \stackrel{\text{def}}{=} K_1$
$K/b = \{a, aba\}/b \cdot K = \varnothing \cdot K = \varnothing$
$K_1/a = \{\lambda, ba\}/a\ K \cup K/a = K/a = K_1$
$K_1/b = \{\lambda, ba\}/b\ K \cup K/b = \{a\}K \stackrel{\text{def}}{=} K_2$
$K_2/a = \{a\}/a\ K = \{\lambda\}\ K = K$
$K_2/b = \varnothing$

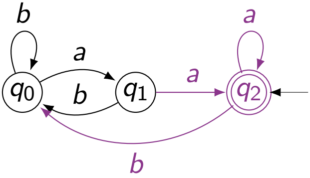
$L/a = \{ab\}L \cup \{\lambda, ba\}K \stackrel{\text{def}}{=} L_1$
$L/b = \varnothing$

$L_1/a = \{ab\}/a\ L \cup \{\lambda, ba\}/a\ K \cup K/a = \{b\}L \cup K/a \stackrel{\text{def}}{=} L_2$
$L_1/b = \{ab\}/b\ L \cup \{\lambda, ba\}/b\ K \cup K/b = \{a\}K = K_2$
$L_2/a = \{b\}/a\ L \cup K_1/a = K_1$
$L_2/b = \{b\}/b\ L \cup K_1/b = L \cup \{a\}K = L$

$L = \{aab\}^*\{a, aba\}^*$
$K = \{a, aba\}^*$







$\underbrace{(b+ab)^*}_{\text{loop on } q_0}\, a$

$\underbrace{b\,[(b+ab)^*a]\,a + a}_{\text{single loop on } q_2}$

$$\underbrace{[(b+ab)^*aa]}_{\text{from } q_0 \text{ to } q_2}\ \underbrace{[b(b+ab)^*aa + a]^*}_{\text{loop on } q_2}$$

short answer $\quad (a+b)^*aa \quad$ see ↪DFA example

ABOVE

It is possible to construct an expression for a small automaton "by hand" by starting with a restricted version of the automaton, and slowly adding nodes and edges.

BELOW

Next a formal proof how this can be done generally, referred to as the McNaughton–Yamada algorithm.

The expression is built iteratively. First we consider only paths in the automaton that can not pass any node: we only consider single edges. Then we add the nodes one by one. Regular expression $R^{(k)}(i,j)$ includes all strings from paths from i to j where only the paths visiting nodes from 1 to k. (We always may exit or enter any other node, but only as first or last node of the path.)

LATER

The method of Brzozowski below "implements" this proof, using a generalized automaton. It features graphs with edges that carry regular expressions.

Theorem

If M is an NFA, then $L(M)$ is regular.

PROOF

$M = (Q, \Sigma, \delta, q_{in}, A)$ assume $Q = \{1, 2, \ldots, n\}$ $q_{in} = 1$

$R^{(k)}(i,j)$ only paths $i, p_1, \ldots, p_\ell, j$ with $1 \leqslant p_\ell \leqslant k$

$R^{(0)}(i,j) = \{a \mid (i,a,j) \in \delta\}$ $i \neq j$ basis

$R^{(0)}(i,j) = \{a \mid (i,a,j) \in \delta\} \cup \{\lambda\}$ $i = j$

one by one add nodes, k from 1 to n:

$$R^{(k)}(i,j) = \underbrace{R^{(k-1)}(i,k)}_{\text{from } i \text{ to } k} \cdot \underbrace{\left(R^{(k-1)}(k,k) \right)^*}_{\text{loop from } k \text{ to } k} \cdot \underbrace{R^{(k-1)}(k,j)}_{\text{from } k \text{ to } j} + R^{(k-1)}(i,j)$$

$L(M) = \bigcup_{j \in A} R^{(n)}(1,j)$ full language, all nodes

[M] Th 3.30

BELOW The *state elimination method* by Brzozowski et McCluskey constructs a regular expression for a given automaton, by iteratively removing the states. The edges of the automaton do not just contain symbols (or λ) but regular expressions themselves. Thus the graphs are a hybrid form of finite automata and regular expressions. It is rather clear however what they express.

Start by adding a new initial and final state; connect the initial state to the old initial state, and connect the old final states to the new final state, using as label the expression Λ (representing the empty word).

Whenever during this construction two parallel edges (p, r_1, q) and (p, r_2, q) appear, we replace them with a single edge $(p, r_1 + r_2, q)$

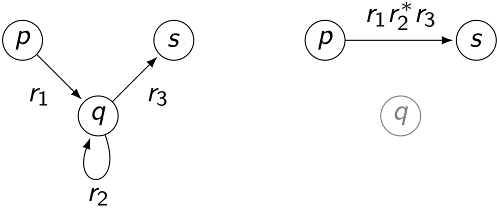
Choose any node q to be removed. Let r_2 be the expression on the loop for q. (If there is no loop we consider this expression to be $\varnothing$.)

For any incoming edge (p, r_1, q) and outgoing edge (q, r_3, s) we add the edge $(p, r_1 r_2^* r_3, s)$ which replace the path from p to s via q.

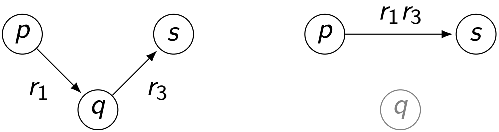
Remove q. Repeat.

When all original nodes are removed, we obtain a graph with single edge; its label represents the language of the original automaton.

join parallel edges



reduce node q



special case: $r_2 = \varnothing$

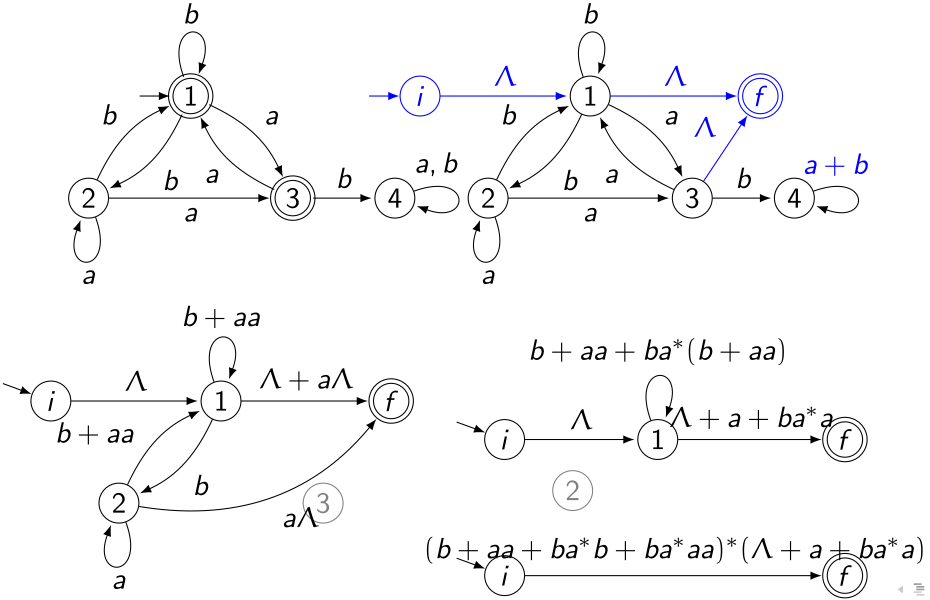
[M] Exercise 3.54

References

R. McNaughton and H. Yamada, Regular expressions and state graphs for automata, IRE Trans. Electronic Computers, vol. 9 (1960), 39–47.
S.C. Kleene. Representation of Events in Nerve Nets and Finite Automata. Automata Studies, Annals of Math. Studies. Princeton Univ. Press. 34 (1956)

state elimination method:

J.A. Brzozowski et E.J. McCluskey, Signal Flow Graph Techniques for Sequential Circuit State Diagrams, IEEE Transactions on Electronic Computers, Institute of Electrical & Electronics Engineers (IEEE), vol. EC-12, no 2, avril 1963, p. 67–76. doi:10.1109/pgec.1963.263416

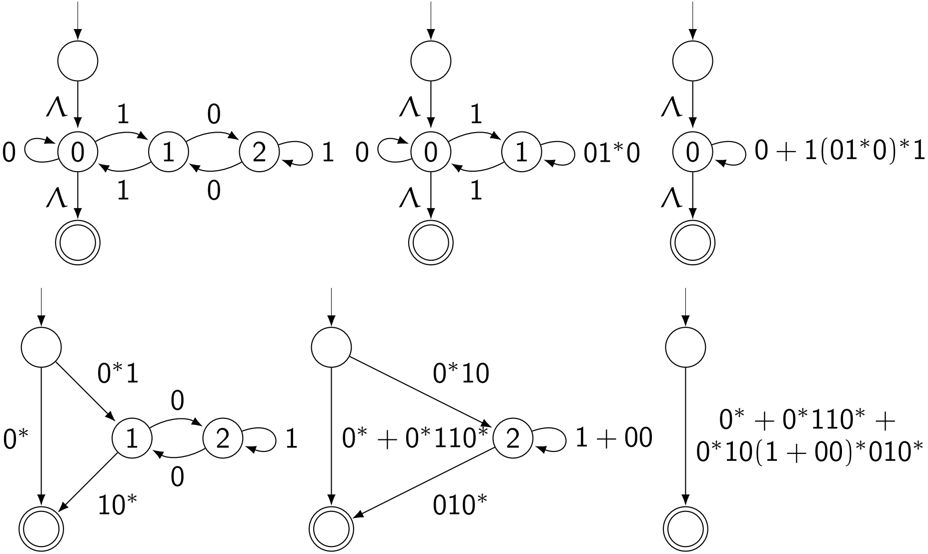

b
1
a
b
a, b
2
b
a
3
b
4
a
a
i
Λ
Λ
f
Λ
a + b
b + aa
Λ
Λ + aΛ
b + aa
b
aΛ
3
b + aa + ba* (b + aa)
Λ
Λ + a + ba* a
2
(b + aa + ba* b + ba* aa)* (Λ + a + ba* a)

ABOVE

Start by adding new initial and final states i and f. Connect these to the original initial and final states by edges with the expression Λ. Note we also replaced the parallel edges a, b (loops on node 4) with the expression $a + b$.

The first node that is eliminated is 4. The proces is not visible here, as there are no pairs (i, j) such that there are edges $(i, R_1, 4)$ and $(4, R_2, j)$, because there are no outgoing edges from 4. Thus no edges are constructed.

The secon node eliminated is 3, as shown.

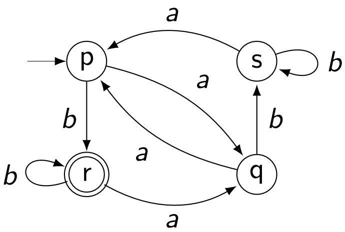

Λ
1
0
0
0
1
2
1
1
0
Λ
Λ
1
0
0
1
01*0
1
Λ
Λ
0
0 + 1(01*0)*1
Λ
0*1
0
0*
1
2
1
0
10*
0*10
0* + 0*110*
2
1 + 00
010*
0* + 0*110* +
0*10(1 + 00)*010*

ABOVE

We compute a regular expression fro the given automaton in two different reduction orders.

The first example reduces nodes in the order $\mathsf{2, 1, 0}$. The result is $(\ \mathsf{0} + \mathsf{1}(\mathsf{0}\mathsf{1}^*\mathsf{0})^*\mathsf{1}\ )^*$
(The last loop was not removed, due to space restrictions.)

The second example in the order $\mathsf{0, 1, 2}$. The result $\mathsf{0}^* + \mathsf{0}^*\mathsf{11}\mathsf{0}^* + \mathsf{0}^*\mathsf{10}(\mathsf{1} + \mathsf{00})^*\mathsf{01}\mathsf{0}^*$
The result differs in structure and size.



X strings starting in x

$$P = aQ + bR$$
$$Q = aP + bS$$
$$R = aQ + bR + \lambda$$
$$S = aP + bS$$

Lemma (Arden's rule)

$L = RL + S$ then $L = R^*S$ provided $\lambda \notin R$

$$P = aQ + bR \qquad P = (a + bb^*a)Q + bb^* = b^*aQ + bb^*$$
$$Q = aP + bS \qquad Q = (a + bb^*a)P = b^*aP$$
$$R = b^*(aQ + \lambda)$$
$$S = b^*aP \qquad P = b^*ab^*aP + bb^* \qquad P = (b^*ab^*a)^*bb^*$$

ABOVE

Start with the four equations for the states and their languages. Strings starting in p either start with an a and continue in Q, or start with a b and continue in R. Etc.

$P = aQ + bR$
$Q = aP + bS$
$R = aQ + bR + \lambda$
$S = aP + bS$

Solve (R) and (S) using Arden's rule

$R = bR + (aQ + \lambda) \quad R = b^*(aQ + \lambda)$
$S = bS + aP \quad S = b^*aP$

Substitute (R) and (S) into (P) and (Q)

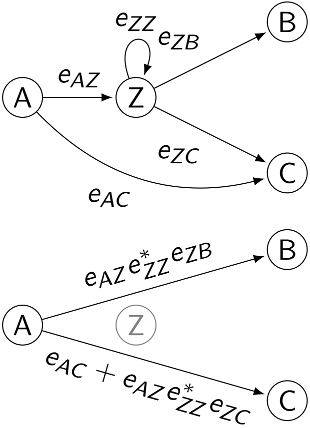
$P = aQ + bR = aQ + b(b^*(aQ + \lambda)) = b^*aQ + bb^*$
$Q = aP + bS = aP + b(b^*aP) = (a + bb^*a)P = b^*aP$

Here we used the simplification $a + bb^*a = b^*a$

Substitute (Q) in (P) and solve using Arden.

$P = b^*ab^*aP + bb^* \quad P = (b^*ab^*a)^*bb^*$

(1) state elimination Z



(2) algebraic method

$$A = e_{AZ}Z + e_{AC}C$$
$$Z = e_{ZZ}Z + e_{ZB}B + e_{ZC}C$$

solve Z using Arden,
then substitute Z into A:

$$Z = a^*_{ZZ}(e_{ZB}B + e_{ZC}C)$$
$$A = e_{AZ}a^*_{ZZ}(e_{ZB}B + e_{ZC}C) + e_{AC}C$$
$$= e_{AZ}a^*_{ZZ}e_{ZB}B + \left(e_{AZ}a^*_{ZZ}e_{ZC} + e_{AC}\right)C$$

ABOVE

The third method based on the solution of a system of linear equations, is in fact a more algebraic presentation of the second method.
If the reductions are done in the same order then the expressions computed in both methods are rather similar.

Sakarovitch paper: Proposition 1 ([8]). The state elimination method and the solution (by Gaussian elimination) of a system of linear equations taken from an automaton give the same regular expression (assuming that the same order in elimination is used in both cases).

$h : \Sigma \to \Delta^*$ letter-to-string map

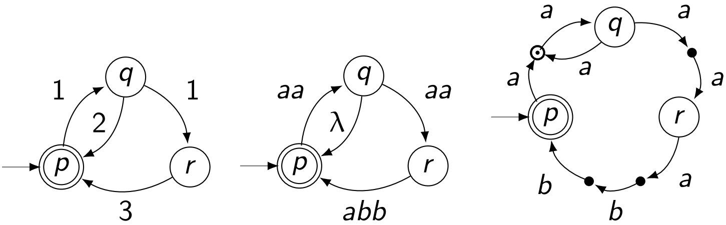
$$h : \begin{array}{ccc} 1 & \mapsto & aa \\ 2 & \mapsto & \lambda \\ 3 & \mapsto & abb \end{array}$$

$h : \Sigma^* \to \Delta^*$ string-to-string map

$h(\sigma_1\sigma_2 \ldots \sigma_k) = h(\sigma_1)h(\sigma_2) \ldots h(\sigma_k)$ $\quad h(1213) = aa \cdot \lambda \cdot aa \cdot abb$

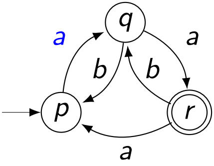
$K \subseteq \Sigma^*$ language-to-language map

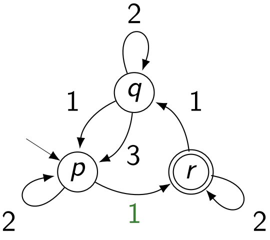
$h(K) = \{ h(x) \mid x \in K \}$



$h : \Sigma \to \Delta^*,\ L \subseteq \Delta^* \qquad h^{-1}(L) = \{\, x \in \Sigma^* \mid h(x) \in L \,\}$

$$h : \begin{array}{ccc} 1 & \mapsto & aa \\ 2 & \mapsto & \lambda \\ 3 & \mapsto & abb \end{array} \qquad \begin{array}{lllllll} \Sigma^* & K & \ni & 1 & 2 & 1 \ \ 3 & 1 \\ \downarrow h & & & & & & \\ \Delta^* & L & \ni & aa & \lambda & aa \ \ abb & aa \end{array}$$

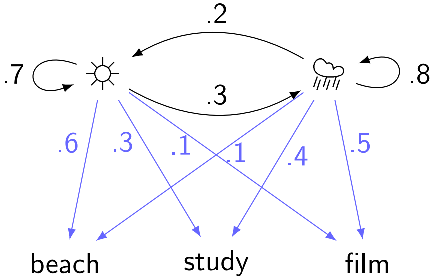




Regular languages are closed under

– Boolean operations (complement, union, intersection)
– Regular operations (union, concatenation, star)
– Reverse (mirror)
– [inverse] Homomorphism
– [fair] Shuffle

- automata on infinite strings
- automata on trees
- automata on grids
- automata and logic
- automata and probability


.2
.7
.3
.8
.6
.3
.1
.1
.4
.5
beach
study
film

Section 4

Context-Free Languages

4 Context-Free Languages
- Examples: recursion
- Regular operations
- Right-linear grammars
- Expression, ambiguity
- Normalform
- Chomsky normalform
- Attribute grammars
- Pumping Lemma
- Decision problems

$\langle$assignment$\rangle$ ::= $\langle$variable$\rangle$ = $\langle$expression$\rangle$

$\langle$statement$\rangle$ ::= $\langle$assignment$\rangle$ |
  $\langle$compound-statement$\rangle$ |
  $\langle$if-statement$\rangle$ |
  $\langle$while-statement$\rangle$ | ...

$\langle$if-statement$\rangle$ ::=
  if $\langle$test$\rangle$ then $\langle$statement$\rangle$ |
  if $\langle$test$\rangle$ then $\langle$statement$\rangle$ else $\langle$statement$\rangle$

$\langle$while-statement$\rangle$ ::=
  while $\langle$test$\rangle$ do $\langle$statement$\rangle$

Definition (well-formed formulas)

. . . by using the construction rules below, and only those, finitely many times:

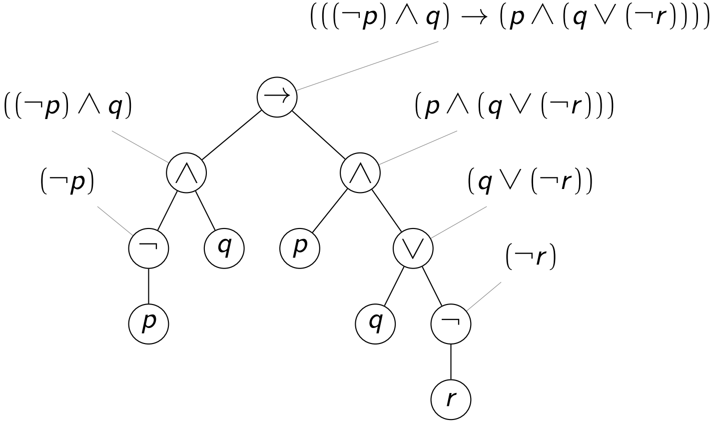
– every propositional atom $p, q, r, \ldots$ is a wff
– if ϕ is a wff, then so is $(\neg\phi)$
– if ϕ and ψ are wff, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$,

BNF   Backus Naur form

$\psi ::= p \mid (\neg\psi) \mid (\psi \wedge \psi) \mid (\psi \vee \psi) \mid (\psi \rightarrow \psi)$

M.Huet & M.Ryan, Logic in Computer Science

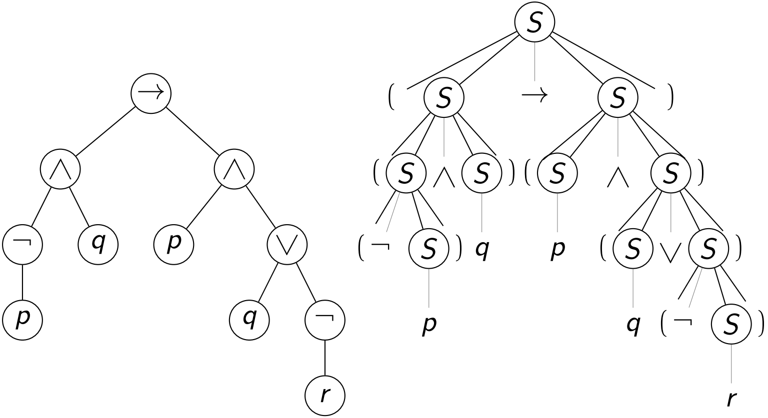
$\psi ::= p \mid (\neg\psi) \mid (\psi \wedge \psi) \mid (\psi \vee \psi) \mid (\psi \rightarrow \psi)$



[H&R] Fig 1.3

$S ::= p \mid q \mid r \mid (\neg S) \mid (S \wedge S) \mid (S \vee S) \mid (S \rightarrow S)$

parse tree vs. derivation tree[2]



[2]with all brackets explicit

$AnBn, Pal \subseteq \{a, b\}^*$, $Balanced \subseteq \{(,)\}^*$

Example

– $\lambda \in AnBn$
– for every $x \in AnBn$, also $axb \in AnBn$

Example

– $\lambda, a, b \in Pal$
– for every $x \in Pal$, also $axa, bxb \in Pal$

Example

– $\lambda \in Balanced$
– for every $x, y \in Balanced$, also $xy \in Balanced$
– for every $x \in Balanced$, also $(x) \in Balanced$

[M] E 1.18, E 1.19

Example

– $\lambda \in AnBn$ (basis)
– for every $x \in AnBn$, also $axb \in AnBn$ (induction)

$S \to \lambda$
$S \to aSb$

$$S \;\Rightarrow\; aSb \;\Rightarrow\; aaSbb \;\Rightarrow\; aa\,bb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaa\,bbb$$

if $S \Rightarrow^* x$ then also $S \Rightarrow^* axb$

$AnBn, Pal \subseteq \{a, b\}^*$, $Balanced \subseteq \{(,)\}^*$

Example

– $\lambda, a, b \in Pal$

– for every $x \in Pal$, also $axa, bxb \in Pal$

$S \to \lambda \mid a \mid b$
$S \to aSa$
$S \to bSb$

$S \Rightarrow aSa \Rightarrow aaSaa \Rightarrow aabSbaa \Rightarrow aababaa$

$AnBn = \{ a^n b^n \mid n \geqslant 0 \}$

variants

$\{ a^n b^{n+1} \mid n \geqslant 0 \}$

$S \to b$ (end with extra b)
$S \to aSb$

$\{ a^i b^j \mid i \leqslant j \}$

$S \to \lambda$
$S \to aSb \mid b$ (free b's)

$\{ a^i b^j \mid i \neq j \}$

$S \to A \mid B$ (choice!)
$A \to aAb \mid aA \mid a$ $(i > j)$
$B \to aBb \mid Bb \mid b$ $(i < j)$

NonPal $\subseteq \{a, b\}^*$

Example

– for every $A \in \{a, b\}^*$, aAb and bAa are elements of NonPal
– for every S in NonPal, aSa and bSb are in NonPal

$A \to \lambda \mid aA \mid bA$
$S \to aAb \mid bAa \mid aSa \mid bSb$
[M] E 4.3

Coin exchange language

alphabet $\{ 1, 2, 5, = \}$

$\{ x{=}y \mid x \in \{1, 2\}^*, y \in \{5\}^*, n_1(x) + 2n_2(x) = 5n_5(y) \}$

$n_\sigma(x)$ number of σ occurrences in x

$212{=}5 \quad 22222{=}55 \quad 12(122)^3 2{=}5^4$

The problem with most solutions is that when read from left to right the initial string over $\{0, 1\}$ cannot always be chopped into part with exact value 5, without chopping the symbol 2.
The solution is like a finite state automaton, which reads $1, 2$ and 'saves' the values until the value 5 is reached, then we write a 5 to the right.

$\Sigma = \{ 1, 2, 5, = \}$

variables S_i, $0 \leqslant i \leqslant 4$

axiom S_0

productions

$S_0 \rightarrow 1S_1 \mid 2S_2$
$S_1 \rightarrow 1S_2 \mid 2S_3$
$S_2 \rightarrow 1S_3 \mid 2S_4$
$S_3 \rightarrow 1S_4 \mid 2S_0 5$
$S_4 \rightarrow 1S_0 5 \mid 2S_1 5$
$S_0 \rightarrow =$

Definition

context-free grammar (CFG) 4-tuple $G = (V, \Sigma, P, S)$
– V alphabet *variables*
– Σ alphabet *terminals* disjoint $V \cap \Sigma = \varnothing$
– $S \in V$ *axiom*, start symbol
– P finite set rules, *productions*
of the form $A \to \alpha$, $A \in V$, $\alpha \in (V \cup \Sigma)^*$

derivation step $\alpha = \alpha_1 A \alpha_2 \Rightarrow_G \alpha_1 \gamma \alpha_2 = \beta$ for $A \to \gamma \in P$

Definition

language generated by G
$L(G) = \{ x \in \Sigma^* \mid S \Rightarrow_G^* x \}$

[M] Def 4.6 & 4.7

NonPal, its grammar components

$$A \to \lambda \mid aA \mid bA$$
$$S \to aAb \mid bAa \mid aSa \mid bSb$$

variables $V = \{ S, A \}$
terminals $\Sigma = \{ a, b \}$
axiom S
productions
$P = \{A \to \lambda, A \to aA, A \to bA, S \to aAb, S \to bAa, S \to aSa, S \to bSb\}$

$\Rightarrow_G^*$ is the *transitive and reflexive closure* of $\Rightarrow_G$

zero, one or more steps

general case $\quad \alpha = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = \beta$

$\alpha \Rightarrow_G^* \beta$ iff there are strings $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that
- $\alpha_0 = \alpha$
- $\alpha_n = \beta$
- $\alpha_i \Rightarrow \alpha_{i+1} \quad$ for $0 \leqslant i < n$.

special case $\quad n = 0 \quad \alpha = \alpha_0 = \beta$

Lemma

If $u_1 \Rightarrow^* v_1$ *and* $u_2 \Rightarrow^* v_2$*, then* $u_1 u_2 \Rightarrow^* v_1 v_2$*.*

Lemma

If $u \Rightarrow^* v_1 v v_2$ *and* $v \Rightarrow^* w$*, then* $u \Rightarrow^* v_1 w v_2$*.*

Lemma

If $u \Rightarrow^* v$ *and* $u = u_1 u_2$*,*
then $v = v_1 v_2$ *such that* $u_1 \Rightarrow^* v_1$ *and* $u_2 \Rightarrow^* v_2$*.*

$AeqB = \{ x \in \{a, b\}^* \mid n_a(x) = n_b(x) \}$

$aaabbb, ababab, aababb, \ldots$

$$\begin{aligned} S &\to \lambda \mid aB \mid bA \\ A &\to aS \mid bAA \\ B &\to bS \mid aBB \end{aligned}$$

A generates $n_a(x) = n_b(x) + 1$
B generates $n_a(x) + 1 = n_b(x)$

$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow \ldots$ (different options)

(1) $aabB \Rightarrow aabaBB \Rightarrow aababSB \Rightarrow aababB \Rightarrow aababbS \Rightarrow aababb$
(2) $aaba\underline{B}B \Rightarrow aabab\underline{S}B \Rightarrow aabab\underline{B} \Rightarrow aababb\underline{S} \Rightarrow aababb$
(2') $aabaB\underline{B} \Rightarrow aaba\underline{B}bS \Rightarrow aababSb\underline{S} \Rightarrow aabab\underline{S}b \Rightarrow aababb$

[M] E 4.8

ABOVE

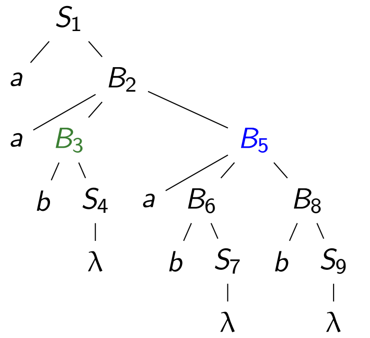
When a string has multiple variables, like $aabSB$ in the above example, then we are not forced to rewrite the first variable, we can as well rewrite another one.

Thus we can do $aab\underline{S}B \Rightarrow aabB$, but also $aabS\underline{B} \Rightarrow aabSaBB$, for instance.

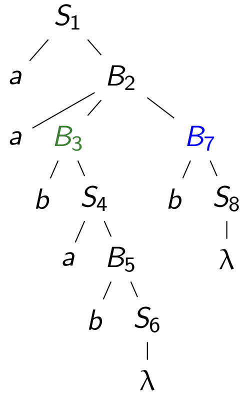
BELOW

In detail, two different derivation trees for the same string, corresponding to derivations (1) and (2,2') respectively, together with two associated leftmost derivations.

Given these to trees we conclude the grammar is ambibuous.



$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow$
$aabB \Rightarrow aabaBB \Rightarrow aababSB \Rightarrow$
$aababB \Rightarrow aababbS \Rightarrow aababb$



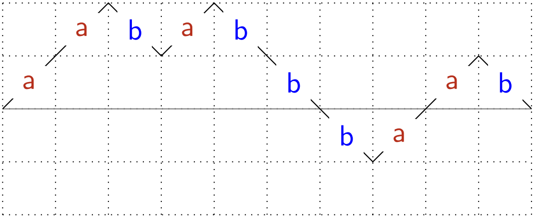
$S \Rightarrow aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow$
$aabaBB \Rightarrow aababSB \Rightarrow aababB \Rightarrow$
$aababbS \Rightarrow aababb$

derivation step $\alpha = \alpha_1 A \alpha_2 \Rightarrow_G \alpha_1 \gamma \alpha_2 = \beta$ for $A \to \gamma \in P$

The derivation step is *leftmost* iff $\alpha_1 \in \Sigma^*$

We write $\alpha \overset{\ell}{\Rightarrow} \beta$

$AeqB = \{ x \in \{a, b\}^* \mid n_a(x) = n_b(x) \}$



$S \to \lambda \mid aSb \mid bSa \mid SS$

$S \Rightarrow SS \Rightarrow a_1Sb_6S \Rightarrow a_1a_2Sb_3Sb_6S \Rightarrow \ldots$

$S \Rightarrow a_1Sb_{10} \Rightarrow \ldots$

[M] Exercise 1.66

TODO

D_2 nested, two pairs of brackets $\Sigma = \{ \, (, \,), [,] \, \}$

$S \to (\, S \,)\, S \mid [\, S \,]\, S \mid \lambda$

$L_1 = \{ a^i b^j c^k \mid i = j + k \}$ $\quad aaa\,b\,cc$

generate as $\quad a^{k+j}\ b^j\ c^k = \underbrace{a^k\ \underbrace{a^j\ b^j}\ c^k}$

$S \to aSc \mid T$
$T \to aTb \mid \lambda$

$S \Rightarrow aSc \Rightarrow aaScc \Rightarrow aaTcc \Rightarrow aaaTbcc \Rightarrow aaabcc$

$L_2 = \{ a^i b^j c^k \mid j = i + k \}$ $\quad a\,bbb\,cc$

generate as $\quad a^i\ b^{i+k}\ c^k = \underbrace{a^i\ b^i}\ \underbrace{b^k\ c^k}$

$S \to AC$ (concatenate)
$A \to aAb \mid \lambda$
$C \to bCc \mid \lambda$

$S \Rightarrow \underline{A}\,C \Rightarrow a\underline{A}b\,C \Rightarrow ab\,\underline{C} \Rightarrow ab\,b\underline{C}c \Rightarrow ab\,bb\underline{C}cc \Rightarrow abbbcc$
$S \Rightarrow A\,\underline{C} \Rightarrow \underline{A}\,bCc \Rightarrow aAb\,b\underline{C}c \Rightarrow a\underline{A}b\,bbCcc \Rightarrow ab\,bb\underline{C}cc \Rightarrow abbbcc$

(a priori there is no prescribed order rewriting A or C)

Using building blocks

Theorem

If L_1, L_2 are CFL, then so are $L_1 \cup L_2$, $L_1 L_2$ and L_1^*.

$G_i = (V_i, \Sigma, P_i, S_i)$, having no variables in common.

Construction

$G = (V_1 \cup V_2 \cup \{S\}, \Sigma, P, S)$, new axiom S

– $P = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$ $L(G) = L(G_1) \cup L(G_2)$

– $P = P_1 \cup P_2 \cup \{S \to S_1 S_2\}$ $L(G) = L(G_1) L(G_2)$

$G = (V_1 \cup \{S\}, \Sigma, P, S)$, new axiom S

– $P = P_1 \cup \{S \to S_1 S, S \to \lambda\}$ $L(G) = L(G_1)^*$

[M] Thm 4.9

$$L_0 = \{\ a^i b^j c^k \mid j = i + k\ \} = \{\ a^i b^{i+k} c^k \mid j = i + k\ \}$$
$$= \{\ \underbrace{a^i b^i}\, \underbrace{b^k c^k} \mid j = i + k\ \}$$

$$S_0 \to XY \qquad X \to aXb \mid \lambda \qquad Y \to bYc \mid \lambda$$

$L = \{\ a^i b^j c^k \mid j \neq i + k\ \} = L_1 \cup L_2$

$S \to S_1 \mid S_2$

$L_1 = \{\ a^i b^j c^k \mid j > i + k\ \}$

$$S_1 \to X_1 b Y_1$$
$$X_1 \to aX_1 b \mid X_1 b \mid \lambda$$
$$Y_1 \to bY_1 c \mid bY_1 \mid \lambda$$

$L_2 = \{\ a^i b^j c^k \mid j < i + k\ \}$

$$S_2 \to aX_2 Y_2 \mid X_2 Y_2 c$$
$$X_2 \to aX_2 b \mid aX_2 \mid \lambda$$
$$Y_2 \to bY_2 c \mid Y_2 c \mid \lambda$$

[M] E 4.10

ABOVE

De uitwerking uit het boek is wat te ingewikkeld, dat hebben we hier wat ingekort.

Example

$\{ \, x\,y \mid x, y \in \{a, b\}^*, |x| = |y|, x \neq y \, \}$

[M] can't find it

Fact, proof follows $\hookrightarrow$later

Theorem

the languages
– $AnBnCn = \{ a^n b^n c^n \mid n \geqslant 0 \}$ *and*
– $XX = \{xx \mid x \in \{a, b\}^* \}$
are not context-free

[M] E 6.3, E 6.4

AnBnCn is the intersection of two context-free languages
[M] E 6.10

The complement of both *AnBnCn* and *XX* is context-free.
[M] E 6.11

Example

$L_1 = \{\ a^{2n} b^n \mid n \geqslant 1\ \}^*$

$a^{16} b^8 a^8 b^4 a^4 b^2 a^2 b^1$

$L_2 = a^* \{\ b^n a^n \mid n \geqslant 1\ \}^* \{\ b\ \}$

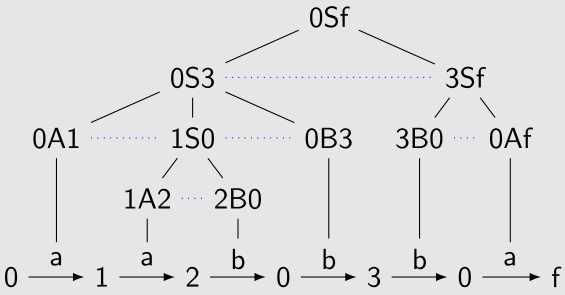
Theorem

If L is a CFL, and R in REG, then L is CFL.

$AeqB \cap \{aab, bb\}^*\{a\}$

$S \to ASB \mid BSA \mid AB \mid BA \mid SS \quad A \to a \quad B \to b$

Example (☒)



[M] Th 6.13, Via ↪PDA

ABOVE

The intersection of a CFL and a regular language is a again context-free.

The example shows how this can be proved asing grammars. The derivation tree matches that of the original CFG, while it also guesses a computation of the finite state automaton for the regular language. The states are in the leafs, but have to be transferred up, to check the states of consecutive steps in the leafs actually match.

This is example-only. We will discuss a slightly simpler proof, when we consider pushdown automata, which are a machine model for context-free languages.

$S \rightarrow S_1 \mid S_2$	union
$S \rightarrow S_1 S_2$	concatenation
$S \rightarrow SS_1 \mid \lambda$	star

Example

$L = bba(ab)^* + (ab + ba^* b)^* ba$

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow S_1 ab \mid bba$

$S_2 \rightarrow CS_2 \mid ba \qquad C \rightarrow abC \mid bEbC \mid \lambda \qquad E \rightarrow Ea \mid \lambda$

[M] E 4.11

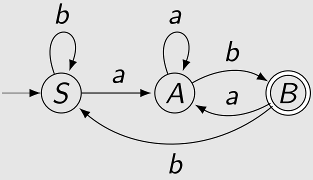
ABOVE

We have seen constructions to apply the regular operations (union, concatenation and star) to context-free grammars. These we can now use to build CFG for regular expressions.

There is a better way to build CFG for regular languages. Use finite state automata, and simulate these using a very simple type of context-free grammar. These simple grammars are called right-linear.

systematic approach

Example



axiom S	initial state
$S \to aA \mid bS$	transitions
$A \to aA \mid bB$	
$B \to aA \mid bS$	
$B \to \lambda$	accepting state

Definition

right-linear grammar

productions are of the form

– $A \to \sigma B$ variables A, B, terminal σ

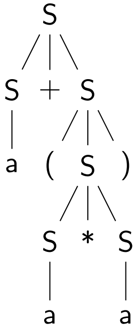
– $A \to \lambda$ variable A

do *not* use 'regular grammar'

Theorem

language L is regular iff there is a right-linear grammar generating L.

[M] Def 4.13, Thm 4.14



$S \to a \mid S+S \mid S*S \mid (S) \qquad \Sigma = \{a, +, *, (,)\}$

$S \Rightarrow S+\underline{S} \Rightarrow S+(\underline{S}) \Rightarrow S+(\underline{S}*S) \Rightarrow \underline{S}+(a*S) \Rightarrow a+(a*\underline{S}) \Rightarrow a+(a*a)$

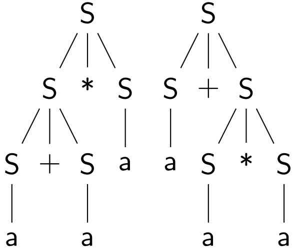
$S \stackrel{\ell}{\Rightarrow} \underline{S}+S \stackrel{\ell}{\Rightarrow} a+S \stackrel{\ell}{\Rightarrow} a+(\underline{S}) \stackrel{\ell}{\Rightarrow} a+(\underline{S}*S) \stackrel{\ell}{\Rightarrow} a+(a*\underline{S}) \stackrel{\ell}{\Rightarrow} a+(a*a)$

leftmost derivation $\longleftrightarrow$ derivation tree

[M] E 4.2, Fig 4.15

TODO [M] Thm 4.17

Ambiguity (1)



$\Sigma = \{a, +, *, (,)\}$

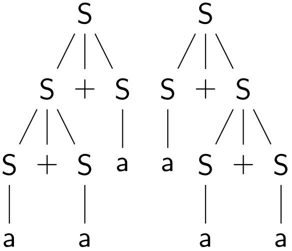
$S \to a \mid S + S \mid S * S \mid (S)$

$a + a * a$

$S \overset{\ell}{\Rightarrow} \underline{S} * S \overset{\ell}{\Rightarrow} S + S * S \overset{\ell}{\Rightarrow} a + S * S \overset{\ell}{\Rightarrow}$
$a + a * S \overset{\ell}{\Rightarrow} a + a * a$

$S \overset{\ell}{\Rightarrow} \underline{S}{+}S \overset{\ell}{\Rightarrow} a{+}S \overset{\ell}{\Rightarrow} a{+}S{*}S \overset{\ell}{\Rightarrow} a{+}a{*}S \overset{\ell}{\Rightarrow}$
$a + a * a$

leftmost derivation $\longleftrightarrow$ derivation tree



$\Sigma = \{a, +, *, (,)\}$

$S \to a \mid S + S \mid S * S \mid (S)$

$a + a + a$

$S \overset{\ell}{\Rightarrow} \underline{S} + S \overset{\ell}{\Rightarrow} S + S + S \overset{\ell}{\Rightarrow} a + S + S \overset{\ell}{\Rightarrow}$
$a + a + S \overset{\ell}{\Rightarrow} a + a + a$

$S \Rightarrow S + \underline{S} \Rightarrow S + S + S \Rightarrow a + S + S \Rightarrow$
$a + a + S \Rightarrow a + a + a$

$S \overset{\ell}{\Rightarrow} \underline{S} + S \overset{\ell}{\Rightarrow} a + S \overset{\ell}{\Rightarrow} a + S + S \overset{\ell}{\Rightarrow}$
$a + a + S \overset{\ell}{\Rightarrow} a + a + a$

leftmost derivation $\longleftrightarrow$ derivation tree

ABOVE

This example is a little weird. In the derivation step $S{+}S \Rightarrow S{+}S{+}S$ we cannot really see which S has been rewritten. In general we will assume that we know (without introducing extra notation to our definitions).

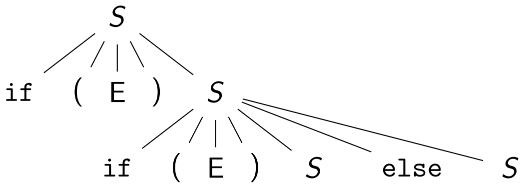
Definition

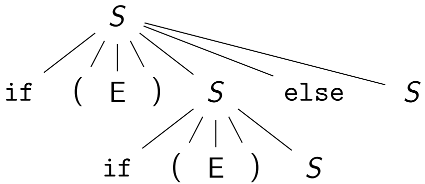
CFG G is *unambiguous* if each $x \in L(G)$ has exactly one derivation tree.

iff each x has exactly one leftmost derivation

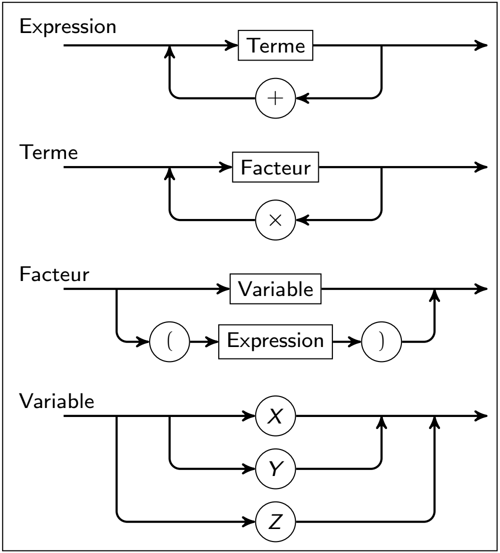
[M] D 4.18

$S \rightarrow \texttt{if}\ (\ E\ )\ S \ \mid\ \texttt{if}\ (\ E\ )\ S\ \texttt{else}\ S\ \mid\ \ldots$





[M] D 4.19



http://math.et.info.free.fr/TikZ/index.html Chapitre 7

Expr

$S \to a \mid S + S \mid S * S \mid (S)$

[M] E 4.20

$S \to S + T \mid T$

$T \to T * F \mid F$

$F \to a \mid (S)$

is unambiguous

[M] Thm 4.25

Balanced

$S \to SS \mid (S) \mid \lambda$ (more or less the definition of balanced)

$S \to (S)S \mid \lambda$

is unambiguous

[M] Exercise 4.45

unwanted in CFG:
– variables not used in successful derivations $S \Rightarrow^* x \in \Sigma^*$
– $A \to \lambda$ A variable λ-productions
– $A \to B$ A, B variables unit productions [chain rules]

restricted CFG, with 'nice' from
Chomsky normalform $A \to BC$, $A \to \sigma$
Greibach normalform (⊠) $A \to \sigma B_1 \dots B_k$

CFG $G = (V, \Sigma, S, P)$

Definition

variable A is *live* if $A \Rightarrow^* x$ for some $x \in \Sigma^*$.

variable A is *reachable* if $S \Rightarrow^* \alpha A \beta$ for some $\alpha, \beta \in (\Sigma \cup V^*)$.

variable A is *useful* if there is a derivation of the form $S \Rightarrow^* \alpha A \beta \Rightarrow^* x$ for some string $x \in \Sigma^*$.

useful implies live and reachable.
For $S \to AB$ and $A \to a$, variable A is live and reachable, not useful.
[M] Exercise 4.51, 4.52, 4.53

Construction

- $N_0 = \varnothing$
- $N_{i+1} = N_i \cup \{ A \in V \mid A \to \alpha \text{ in } P, \text{ with } \alpha \in (N_i \cup \Sigma)^* \}$

$N_1 = \{ A \in V \mid A \to x \text{ in } P, \text{ with } x \in \Sigma^* \}$
$N_0 \subseteq N_1 \subseteq N_2 \subseteq \cdots \subseteq V$
there exists a k such that $N_k = N_{k+1}$
A is live iff $A \in \bigcup_{i \geqslant 0} N_i = N_k$ depth of derivation tree

Construction

– $N_0 = \{S\}$
– $N_{i+1} = N_i \cup \{\, A \in V \mid B \to \alpha_1 A \alpha_2 \text{ in } P, \text{ with } B \in N_i \,\}$

$N_0 \subseteq N_1 \subseteq N_2 \subseteq \cdots \subseteq V$
there exists a k such that $N_k = N_{k+1}$
A is reachable iff $A \in \bigcup_{i \geqslant 0} N_i = N_k$ length of derivation

– remove all non-live variables (and productions that contain them)
– remove all unreachable variables (and productions)

then all variables are useful

does not work the other way around . . .

Definition

variable A is *nullable* iff $A \Rightarrow^* \lambda$

Theorem

– *if* $A \to \lambda$ *then* A *is nullable*
– *if* $A \to B_1 B_2 \dots B_k$ *and all* B_i *are nullable, then* A *is nullable*

[M] Def 4.26

Construction

– $N_0 = \varnothing$
– $N_{i+1} = N_i \cup \{ A \in V \mid A \to \alpha \text{ in } P, \text{ with } \alpha \in N_i^* \}$

$N_1 = \{ A \in V \mid A \to \lambda \text{ in } P \}$
$N_0 \subseteq N_1 \subseteq N_2 \subseteq \cdots \subseteq V$
there exists a k such that $N_k = N_{k+1}$
A is nullable iff $A \in \bigcup_{i \geqslant 0} N_i = N_k$

Construction

– identify nullable variables
– for every production $A \to \alpha$ add $A \to \beta$,
  where β is obtained from α by removing one or more nullable variables
– remove all λ-productions

Theorem

For every CFG G *there is* CFG G_1 *without* λ*-productions such that* $L(G_1) = L(G) - \{\lambda\}$.

[M] Thm 4.27

Grammar for $\{\ a^i b^j c^k \mid i = j \text{ or } i = k\ \}$

$S \to TU \mid V$
$T \to aTb \mid \lambda$
$U \to cU \mid \lambda$
$V \to aVc \mid W$
$W \to bW \mid \lambda$

$N_1 = \{T, U, W\}$, variables with λ at right-hand side productions

$N_2 = \{T, U, W\} \cup \{S, V\}$, variables with $\{T, U, W\}^*$ at rhs productions

$N_3 = N_2 = \{T, U, W, S, V\}$, all productions found, no new add all productions, where (any number of) nullable variables are removed

$S \to TU \mid V \quad S \to T \mid U \mid \lambda$
$T \to aTb \mid \lambda \quad T \to ab$
$U \to cU \mid \lambda \quad U \to c$
$V \to aVc \mid W \quad V \to ac \mid \lambda$
$W \to bW \mid \lambda \quad W \to b$

remove all λ-productions

$S \to TU \mid V \mid T \mid U$
$T \to aTb \mid ab$
$U \to cU \mid c$
$V \to aVc \mid W \mid ac$
$W \to bW \mid b$

[M] Ex. 4.31

Construction

If $A \to B$, and $B \to \beta$, add $A \to \beta$
repeat until no new rules can be added

If $A \to B$, $B \to C$ and $C \to \gamma$, then we get $B \to \gamma$, and $A \to \gamma$

$S \to TU \mid V \mid T \mid U$
$T \to aTb \mid ab$
$U \to cU \mid c$
$V \to aVc \mid W \mid ac$
$W \to bW \mid b$

Chain rules: $S \to V \mid T \mid U$, $V \to W$

New rules:

$S \to aTb \mid ab \quad S \to cU \mid c \quad S \to aVc \mid W \mid ac \quad S \to bW \mid b$
$V \to bW \mid b$

Remove chain rules:

$S \to TU \mid aTb \mid ab \mid cU \mid c \mid aVc \mid ac \mid bW \mid b$
$T \to aTb \mid ab$
$U \to cU \mid c$
$V \to aVc \mid ac \mid bW \mid b$
$W \to bW \mid b$

Definition

CFG in *Chomsky normal form*

productions are of the form

– $A \to BC$ variables A, B, C

– $A \to \sigma$ variable A, terminal σ

Theorem

For every CFG G *there is* CFG G_1 *in CNF such that* $L(G_1) = L(G) - \{\lambda\}$.

[M] Def 4.29, Thm 4.30

Construction

1. remove λ-productions
2. remove chain productions
3. introduce variables for terminals $X_\sigma \rightarrow \sigma$
4. split long rules

$$A \rightarrow aBabA$$

is replaced by

$$X_a \rightarrow a \quad X_b \rightarrow b \quad A \rightarrow X_aBX_aX_bA$$

$$A \rightarrow ACBA$$

is replaced by

$$A \rightarrow AY_1 \quad Y_1 \rightarrow CY_2 \quad Y_2 \rightarrow BA$$

Grammar for $\{ a^i b^j c^k \mid i = j \text{ or } i = k \}$

$S \to TU \mid V$
$T \to aTb \mid \lambda \quad U \to cU \mid \lambda$
$V \to aVc \mid W \quad W \to bW \mid \lambda$

After removing λ-rules, and chain rules, we obtain (see before)

$S \to TU \mid aTb \mid ab \mid cU \mid c \mid aVc \mid ac \mid bW \mid b$
$T \to aTb \mid ab \quad U \to cU \mid c$
$V \to aVc \mid ac \mid bW \mid b \quad W \to bW \mid b$

Now introduce rules for the terminals:

$X_a \to a \quad X_b \to b \quad X_c \to c$
$S \to TU \mid X_a T X_b \mid X_a X_b \mid X_c U \mid c \mid X_a V X_c \mid X_a X_c \mid X_b W \mid b$
$T \to X_a T X_b \mid X_a X_b$
$U \to X_c U \mid c$
$V \to X_a V X_c \mid X_a X_c \mid X_b W \mid b$
$W \to X_b W \mid X_b$

Only a few non-Chomsky productions:

$S \to X_a T X_b \mid X_a V X_c \mid X_a X_c$
$T \to X_a T X_b$
$V \to X_a V X_c$

Split these long rules:

$S \to X_a Z_1 \mid X_a Z_2$
$Z_1 \to T X_b \quad Z_2 \to V X_c$
$T \to X_a Z_1$
$V \to X_a Z_2$

Note that we can reuse Z_1, Z_2 for two rules

$\text{even}(L) = \{ w \in L \mid |w| \text{ even } \}$

idea: new variables for even/odd length strings
Chomsky normalform to reduce number of possibilities.

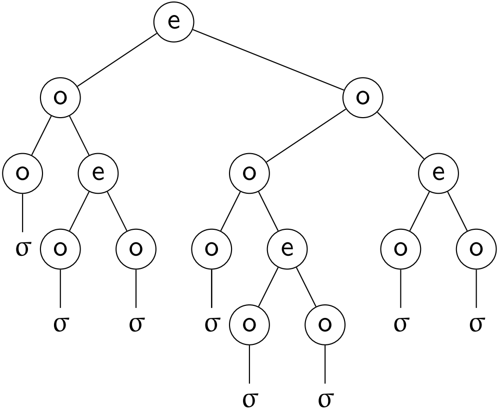
grammar $G = (V, \Sigma, P, S)$ for L, in ChNF
new grammar $G = (V', \Sigma, P', S')$ for $\text{even}(L)$
variables: $V' = \{X_e, X_o \mid X \in V\}$
axiom: $S' = S_e$
productions: – for every $A \to BC$ in P we have in P':

$$A_e \to B_e C_e \mid B_o C_o \qquad A_o \to B_e C_o \mid B_o C_e$$

– for every $A \to \sigma$ in P we have in P': $A_o \to \sigma$

ABOVE

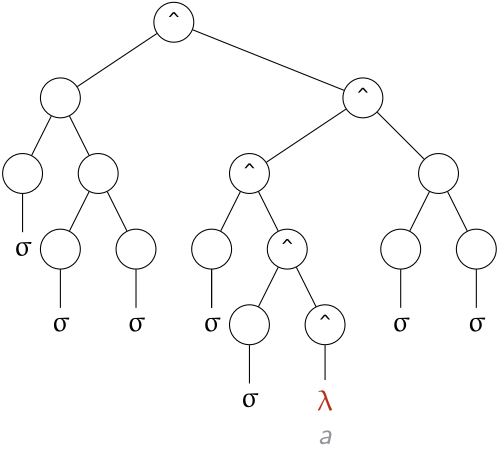
We consider closure properties: given an operation X show that whenever L is regular/context-free, then also $X(L)$ is regular/context-free. This is done as follows: if L is regular/context-free, then we know there is a right-linear/context-free grammar G for L, and we show how to construct a new grammar G' (of the same type) for $X(L)$, in terms of the original grammar G.


e
o
o
o
e
o
e
σ
o
o
o
e
o
o
σ
σ
σ
o
o
σ
σ
σ
σ

$L \subseteq \{a, b\}^*$, $\text{chop}(L) = \{ xy \mid xay \in L\}$ remove some a in each string

idea: new variables for the task of removing letter a

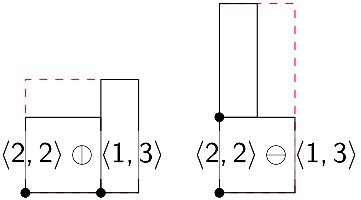
grammar $G = (V, \{a, b\}, P, S)$ for L, in ChNF
new grammar $G = (V', \{a, b\}, P', S')$ for $\text{chop}(L)$
variables: $V' = V \cup \{\hat{X} \mid X \in V\}$
axiom: $S' = \hat{S}$
productions: keep all productions from P, and
– for every $A \to BC$ add $\hat{A} \to \hat{B}C \mid B\hat{C}$
– for every $A \to a$ add $\hat{A} \to \lambda$

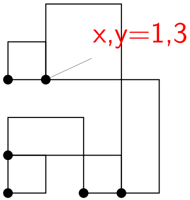

^
^
^
^
^
σ
σ
σ
σ
σ
σ
σ
λ
a

$$E \to T + E \mid T$$
$$T \to F * T \mid F$$
$$F \to (\ E\ ) \mid int$$

$$\begin{array}{ll}
E \to T_1 + E_1 & E.\mathrm{val} = \mathrm{T_1.val} + \mathrm{E_1.val} \\
E \to T_1 & E.\mathrm{val} = \mathrm{T_1.val} + \mathrm{E_1.val} \\
T \to F_1 * T_1 & T.\mathrm{val} = \mathrm{F_1.val} \cdot \mathrm{T_1.val} \\
T \to F_1 & T.\mathrm{val} = \mathrm{F_1.val} \\
F \to (\ E_1\ ) & E.\mathrm{val} = \mathrm{T_1.val} \\
F \to int & F.\mathrm{val} = \mathrm{IntVal}(\ \mathrm{int}\ )
\end{array}$$

D.E. Knuth. Semantics of Context-Free Languages.
Math. Systems Theory (1968) 127–145 doi:10.1007/BF01692511



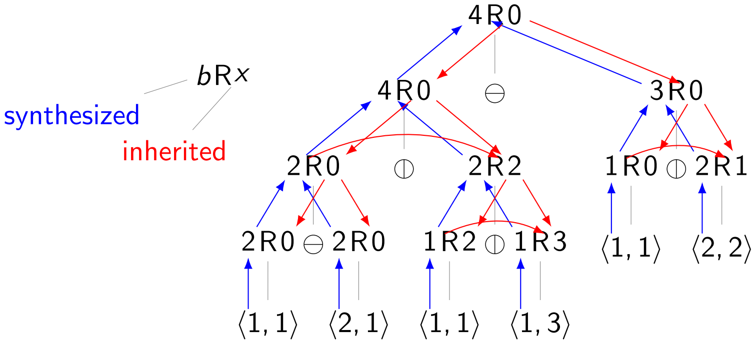


$$(\ ((\langle 1,1\rangle \ominus \langle 2,1\rangle) \obar (\langle 1,1\rangle \obar \langle 1,3\rangle)\ ) \ominus (\langle 1,1\rangle \obar \langle 2,2\rangle)$$

production	semantic rule
$R \to \langle E_1, E_2\rangle$	$R.b = E_1.\mathrm{val} \quad \mathrm{R.h} = \mathrm{E_2.val}$
$R \to (R_1 \obar R_2)$	$R.b = R_1.b + R_2.b$
	$R.h = \max\{R_1.h, R_2.h\}$
	$R_1.x = R.x \quad R_2.x = R.x + R_1.b$
	$R_1.y = R.y \quad R_2.y = R.y$
$R \to (R_1 \ominus R_2)$	$R.b = \max\{R_1.b, R_2.b\}$
	$R.h = R_1.h + R_2.h$
	$R_1.x = R.x \quad R_2.x = R.x$
	$R_1.y = R.y \quad R_2.y = R.y + R_1.h$

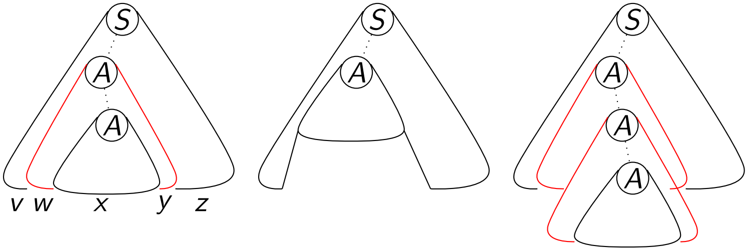
⊠Evaluating attributes

$$R \to (R_1 \oplus R_2) \quad R.b = R_1.b + R_2.b$$
$$R_1.x = R.x \quad R_2.x = R.x + R_1.b$$
$$R \to (R_1 \ominus R_2) \quad R.b = \max\{R_1.b, R_2.b\}$$
$$R_1.x = R.x \quad R_2.x = R.x$$



$$S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz,\ v, w, x, y, z \in \Sigma^*$$

$$S \underset{(1)}{\Rightarrow^*} vAz,\ A \underset{(2)}{\Rightarrow^*} wAy,\ A \underset{(3)}{\Rightarrow^*} x$$



$$S \underset{(1)}{\Rightarrow^*} vAz \underset{(3)}{\Rightarrow^*} vxz$$

$$S \underset{(1)}{\Rightarrow^*} vAz \underset{(2)}{\Rightarrow^*} vwAyz \underset{(2)}{\Rightarrow^*} vwwAyyz \underset{(3)}{\Rightarrow^*} vwwxyyz$$

Theorem (Pumping Lemma for context-free languages)

$\forall$ *for every context-free language* L

$\exists$ *there exists a constant* $n \geqslant 1$

such that

$\forall$ *for every* $u \in L$

with $|u| \geqslant n$

$\exists$ *there exists a decomposition* $u = vwxyz$

with (1) $|wy| \geqslant 1$

and (2) $|wxy| \leqslant n$,

such that

$\forall$ *(3) for all* $i \geqslant 0$, $vw^ixy^iz \in L$

if $L = L(G)$ then $n = 2^{|V|+1}$.

[M] Thm. 6.1

Example

AnBnCn is not context-free.

[M] E 6.3

$u = a^n b^n c^n$

$\{ x \in \{a, b, c\}^* \mid n_a(x) = n_b(x) = n_c(x) \}$

Example

XX is not context-free.

[M] E 6.4

$u = a^n b^n a^n b^n$

$\{ a^i b^j a^i b^j \mid i, j \geqslant 0 \}$

Example

$\{ x \in \{a, b, c\}^* \mid n_a(x) < n_b(x)$ and $n_a(x) < n_c(x) \}$ is not context-free.

[M] E 6.5

ABOVE

$L = \{ x \in \{a, b, c\}^* \mid n_a(x) < n_b(x) \text{ and } n_a(x) < n_c(x) \}$ is not context-free.

Proof by contradiction.

Assume L is context-free, then there exists a pumping constant n for L.

Choose $u = a^n b^{n+1} c^{n+1}$. Then $u \in L$, and $|u| \geqslant n$.

This means that we can pump u within the language L.

Consider a subdivision $u = vwxyz$ that satisfies the pumping lemma, in particular $|wxy| \leqslant n$.

Case 1: wy contains a letter a. Then wy cannot contain letter c (otherwise $|wxy| > n$). Now $u_2 = vw^2xy^2z$ contains more a's than u, so at least $n + 1$, while u_2 still contains $n + 1$ c's. Hence $u_2 \notin L$.

Case 2: wy contains no a. Then wy contains at least one b or one c (or both). Then $u_0 = vw^0xy^0z = vxz$ has still n a's, but less than $n + 1$ b's or less than $n + 1$ c's (depending on which letter is in wy). Hence $u_0 \notin L$.

These are two possibilities for the division $vwxyz$, in both cases we see that pumping leads out of the language L.

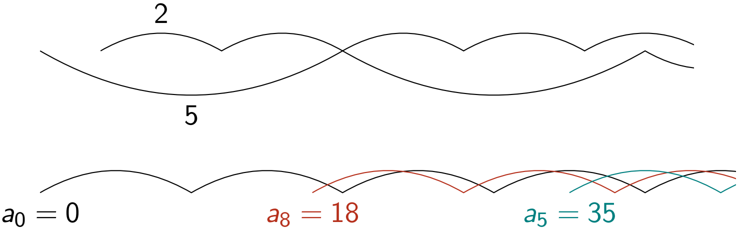
Hence u cannot be pumped.

Contradiction; so L is not context-free.

Lemma (☒)

$L \subseteq \{a\}^*$ context-free, then L regular.

[M] Exercise 6.23



. . . de Karl d'Maria em Peter de Hans laat hälfe lärne schwüme

. . . Charles lets Mary help Peter to teach John to swim

$\{ ww \mid w \in \Sigma^* \}$

$\{ a^n b^m a^n b^m \mid m, n \geqslant 0 \}$

https://en.wikipedia.org/wiki/Cross-serial_dependencies

"given a CFL L, does it have property ... ?" yes/no
input CFG G

Given CFG G [G_1 and G_2]
– and given a string x, is $x \in L(G)$? membership problem
Cocke, Younger, and Kasami (1967)
Earley (1970)
– is $L(G) = \varnothing$? emptiness
– is $L(G)$ infinite?
pumping lemma
– is $L(G_1) \cap L(G_2)$ nonempty?
– is $L(G_1) \subseteq L(G_2)$?
– is $L(G) = \Sigma^*$?

Accepts:
– Given a TM T and a string w, is $w \in L(T)$?
Halts:
– Given a TM T and a string w, does T halt on input w?

Theorem

Both Accepts and Halts are undecidable.

[M] D 9.8, Th 9.16

Definition (PCP)

instance: sequence of pairs of strings over Σ $\{ (\alpha_1, \beta_1), \ldots, (\alpha_n, \beta_n) \}$
question: solution, sequence of indices $i_1, i_2, \ldots, i_k$, $k \geqslant 1$, such that $\alpha_{i_1} \alpha_{i_2} \ldots \alpha_{i_k} = \beta_{i_1} \beta_{i_2} \ldots \beta_{i_k}$

Accepts $\leqslant$ PCP

Theorem

Post's correspondence problem is undecidable.

[M] D 9.14, Th 9.17
[H&R] Logic in CS 2.5 Undecidability of predicate logic

i	1	2	3
α_i	1	10	011
β_i	101	00	11

solution:

1	3	2	3
1	011	10	011
101	11	00	11

http://webdocs.cs.ualberta.ca/~games/PCP/list.htm

i	1	2	3
α_i	100	0	1
β_i	1	100	0

Length of shortest solution: 75

10011100100 ...

10011 ...

symbols $c_1, \dots, c_n, \#$ $\quad \# \notin \Sigma$

$\alpha = (\ \alpha_1, \alpha_2, \dots, \alpha_n\ )$ $\quad \alpha_i \in \Sigma^*$

CFG G_α

productions $\quad S_\alpha \to \alpha_1 S_\alpha c_1 \mid \cdots \mid \alpha_n S_\alpha c_n \mid \alpha_1 \# c_1 \mid \cdots \mid \alpha_n \# c_n$

$$L(G_\alpha) \quad \underbrace{\alpha_{i_1} \alpha_{i_2} \dots \alpha_{i_k}}_{\text{string}} \# \underbrace{c_{i_k} \dots c_{i_2} c_{i_1}}_{\text{indices}}$$

Theorem (Undecidable problems)

Disjointness:
– Given two CFG G_1 and G_2, is $L(G_1) \cap L(G_2)$ nonempty?
Ambiguity:
– Given a CFG G, is G ambiguous?

[M] D Th 9.20

Given context-free L and regular R

– is $R \subseteq L$?

– is $L \subseteq R$?

ABOVE

$R \subseteq L$?

Special case $R = \Sigma^*$

$\Sigma^* \subseteq L$ iff $L = \Sigma^*$ undecidable

$L \subseteq R$?

iff $L \cap R^c = \varnothing$

regular languages are closed under complement

CFL closed under intersection with regular languages

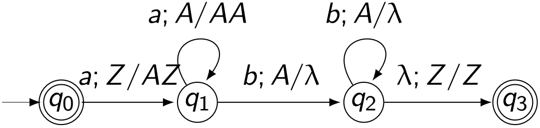
emptiness context-free decidable

Section 5

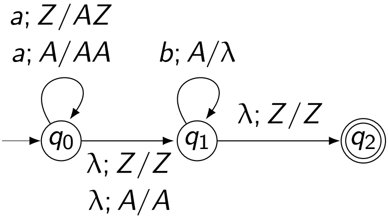
Pushdown Automata

$AnBn = \{ a^n b^n \mid n \geqslant 0 \}$

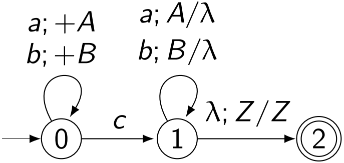
initial q_0, Z, accept $A = \{q_0, q_3\}$

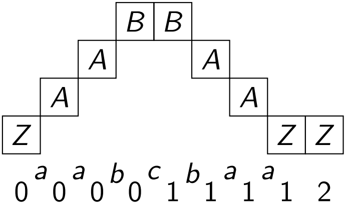


[M] E 5.3



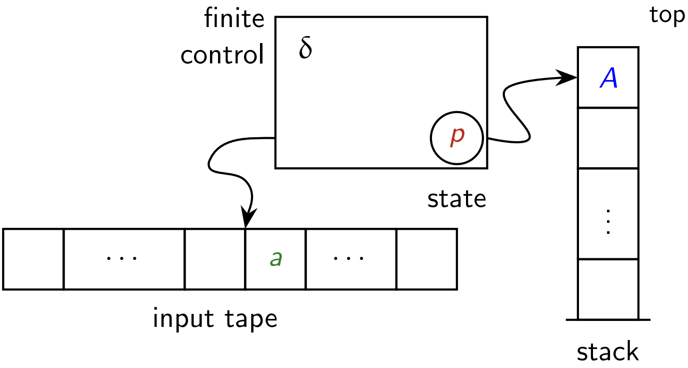
$SimplePal = \{ \, xcx^R \mid x \in \{a, b\}^* \, \}$





$(0, aabcbaa, Z) \vdash$
$(0, abcbaa, AZ) \vdash$
$(0, bcbaa, AAZ) \vdash$
$(0, cbaa, BAAZ) \vdash$
$(1, baa, BAAZ) \vdash$
$(1, aa, AAZ) \vdash$
$(1, a, AZ) \vdash$
$(1, \lambda, Z) \vdash$
$(2, \lambda, Z)$

[M] Fig 5.5


finite
control
δ
p
state
a
. . .
. . .
input tape
top
A
stack

Definition

PDA 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$

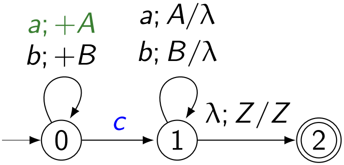
Q	*states*	p, q	
$q_{in} \in Q$	*initial state*		
$A \subseteq Q$	*accepting states*		
Σ	*input alphabet*	a, b	w, x
Γ	*stack alphabet*	A, B	α
$Z_{in} \in \Gamma$	*initial stack symbol*		

$\delta \subseteq Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \times Q \times \Gamma^*$

transition relation (finite)

$$\underbrace{(\overset{\text{from}}{p} \quad \underset{\text{read}}{a} \quad \underset{\text{pop}}{A}}_{\text{before}} \quad \underbrace{\overset{\text{to}}{q} \quad \underset{\text{push}}{\alpha})}_{\text{after}}$$

$SimplePal =$
$\{ xcx^R \mid x \in \{a, b\}^* \}$



$Q = \{0, 1, 2\}$
$\Sigma = \{a, b, c\}$
$\Gamma = \{A, B, Z\}$
$q_{in} = 0$
$Z_{in} = Z$
$A = \{2\}$

δ contains

$(0, a, Z, 0, AZ)$
$(0, a, A, 0, AA)$
$(0, a, B, 0, AB)$

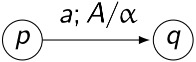
$(0, b, Z, 0, BZ)$
$(0, b, A, 0, BA)$
$(0, b, B, 0, BB)$

$(0, c, Z, 1, Z)$
$(0, c, A, 1, A)$
$(0, c, B, 1, B)$

$(1, a, A, 1, \lambda)$
$(1, b, B, 1, \lambda)$

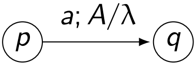
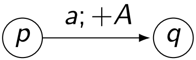
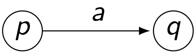
$(1, \lambda, Z, 2, Z)$

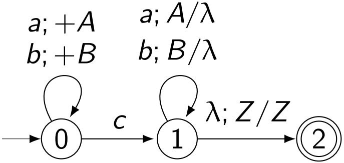
instruction (p, a, A, q, α)

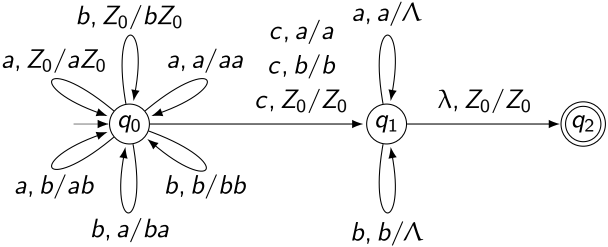


$(p, a, A) \mapsto (q, \alpha)$

$p, q \in Q,\ a \in \Sigma \cup \{\lambda\},\ A \in \Gamma,\ \alpha \in \Gamma^*$

intuitive	formalized as		*my* convention
pop A	(p, a, A, q, λ)	$\alpha = \lambda$	
push A	(p, a, X, q, AX)	for all $X \in \Gamma$	
read a	(p, a, X, q, X)	for all $X \in \Gamma$	


a; +A
b; +B
a; A/λ
b; B/λ
c
λ; Z/Z
0
1
2


b, Z_0/bZ_0
a, Z_0/aZ_0
a, a/aa
c, a/a
c, b/b
c, Z_0/Z_0
a, a/Λ
λ, Z_0/Z_0
q_0
q_1
q_2
a, b/ab
b, b/bb
b, a/ba
b, b/Λ

[M] Fig 5.5

ABOVE

The 'same' PDA twice. First in the version of this lecture where we allow some shortcuts in notation.

Second as depicted in the book. Note Martin happily pushes terminals like a, b on the stack. Formally that is OK, but I am not used to that.

$M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$

configuration (q, x, α) $q \in Q,\ x \in \Sigma^*,\ \alpha \in \Gamma^*$

state, input not yet read, stack with top left

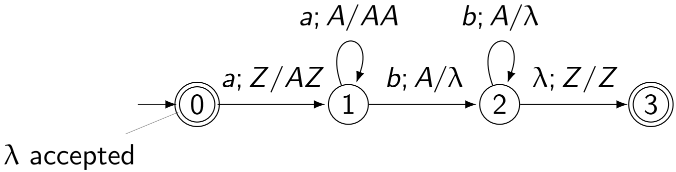
step $(p, ax, B\alpha) \vdash (q, x, \beta\alpha)$ when $(p, a, B, q, \beta) \in \delta$

Definition

Language accepted by M by *final state* $L(M) =$
$\{\ x \in \Sigma^* \mid (q_0, x, Z_0) \vdash^* (q, \lambda, \alpha)$ for some $q \in A$, and some $\alpha \in \Gamma^*\ \}$

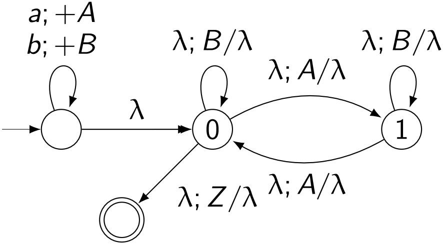
read complete input, end in accepting state

[M] D 5.2



nr	state	input	stack	move	
1	q_0	a	Z	q_1	AZ
2	q_1	a	A	q_1	AA
3	q_1	b	A	q_2	λ
4	q_2	b	A	q_2	λ
5	q_2	λ	Z	q_3	Z

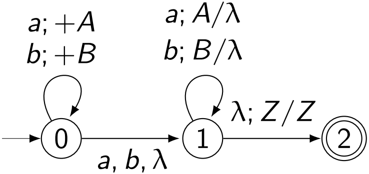
[M] E 5.3, Tab 5.4


$a; +A$
$b; +B$
λ
$\lambda; B/\lambda$
$\lambda; A/\lambda$
$\lambda; B/\lambda$
0
1
$\lambda; Z/\lambda$
$\lambda; A/\lambda$

ABOVE

λ-computations can be very long in PDA, they can even loop.

In the example the input is read and stored on the tape, and at the end of the input it is verified that the string contains an even number of a's.

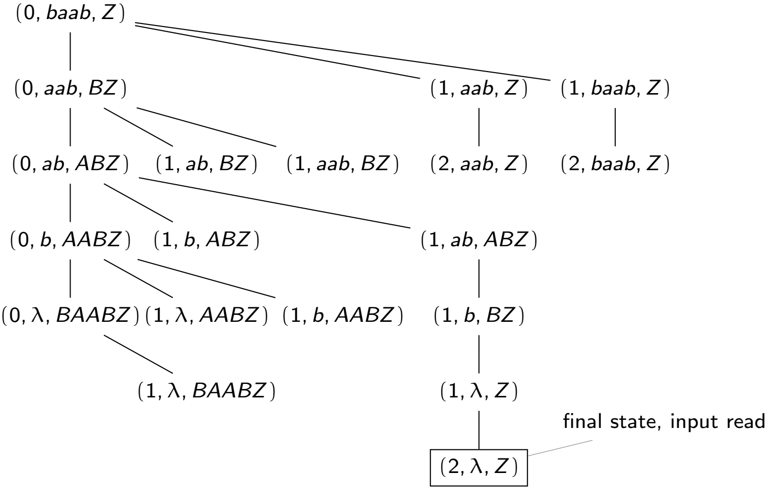


$Q = \{0, 1, 2\}$
$\Sigma = \{a, b, c\}$
$\Gamma = \{A, B, Z\}$
$q_{in} = 0$
$Z_{in} = Z$
$A = \{2\}$

δ contains

$(0, a, Z, 0, AZ)$ $(0, a, Z, 1, Z)$
$(0, a, A, 0, AA)$ $(0, a, A, 1, A)$
$(0, a, B, 0, AB)$ $(0, a, B, 1, B)$
$(0, b, Z, 0, BZ)$ $(0, b, Z, 1, Z)$
$(0, b, A, 0, AZ)$ $(0, b, A, 1, Z)$
$(0, b, B, 0, BB)$ $(0, b, B, 1, B)$
$(0, \lambda, Z, 1, Z)$
$(0, \lambda, A, 1, A)$
$(0, \lambda, B, 1, B)$
$(1, a, A, 1, \lambda)$
$(1, b, B, 1, \lambda)$
$(1, \lambda, Z, 2, Z)$

Computation tree



[M] Fig 5.9

ABOVE

Non-determinism at work. The PDA for palindromes cannot see what is the middle of the input string, and has to guess. Only one of the guesses leads to an accepting computation.

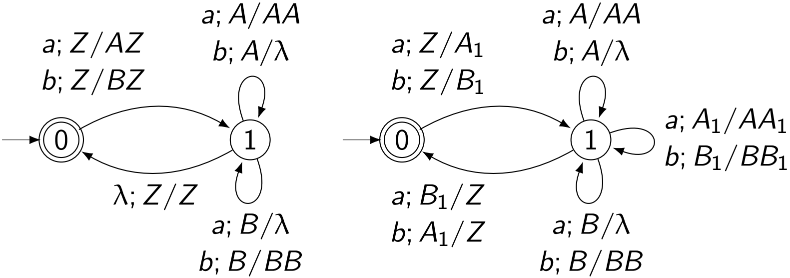
for each state and stack symbol
– on each symbol/λ at most one instruction
– not both symbol and λ-instruction

notation $\delta(q, \sigma, X) = \{ (p, \alpha) \mid (q, \sigma, X, p, \alpha) \in \delta \}$

Definition

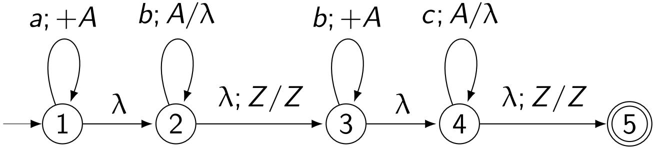
$\delta(q, \sigma, X) \cup \delta(q, \lambda, X)$ at most one element   for each $q \in Q, \sigma \in \Sigma, X \in \Gamma$

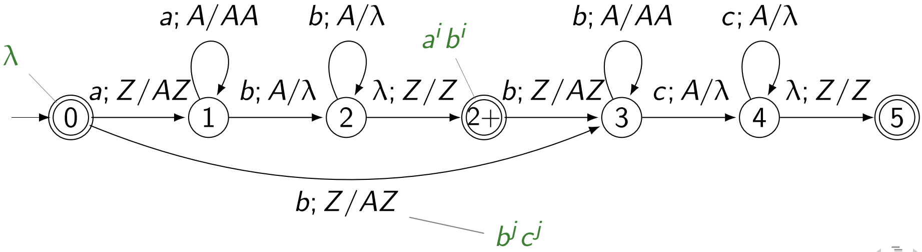
[M] Def 5.10



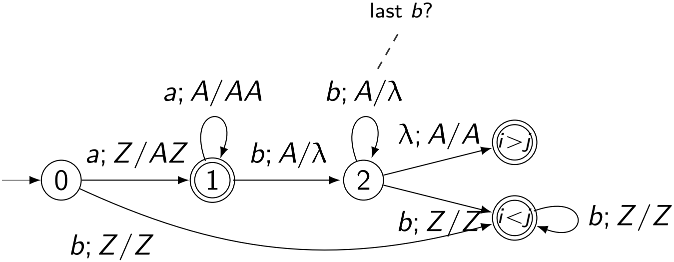
[M] based on E 5.13

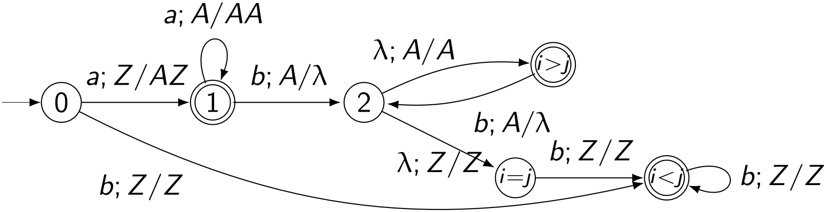
$S \to AB$
$A \to aAb \mid \lambda$
$B \to bBc \mid \lambda$

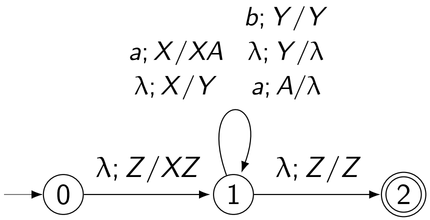


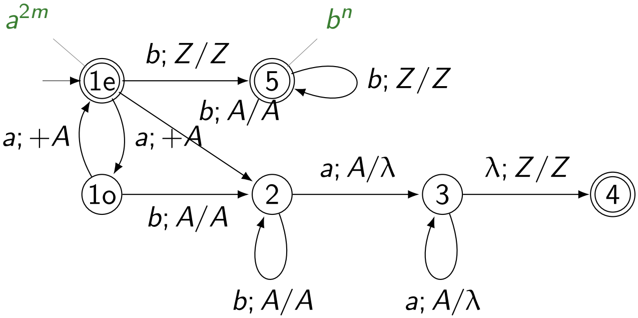


$\{ a^i b^j \mid i \neq j \}$






b; Y/Y
a; X/XA
λ; Y/λ
λ; X/Y
a; A/λ
λ; Z/XZ
λ; Z/Z
0
1
2


a^2m
b^n
b; Z/Z
b; Z/Z
b; A/A
a; +A
a; +A
1e
5
1o
2
3
4
a; A/λ
λ; Z/Z
b; A/A
b; A/A
a; A/λ

ABOVE

The first PDA is not deterministic. Actually it is working like a grammar: in state 1 the following productions are simulated:

$X \rightarrow aXA \mid Y$

$Y \rightarrow bY \mid \lambda$

$A \rightarrow a$

The second automaton is deterministic. We have to distinguish the cases where $m = 0$ (state 5) and $n = 0$ (states $1e$ and $1o$).

$pre(L) = \{ xy \mid x \in L \text{ and } xy \in L \}$

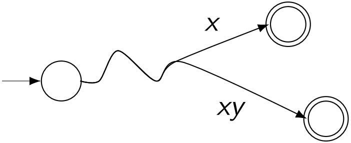
CFL not closed under *pre*
DCFL *is* closed under *pre*

[M] Exercise 5.20 & 6.23

CFL not closed under complement
DCFL is closed under complement ⊠
(the obvious proof does not work)

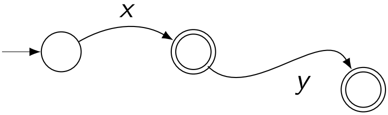
CFL is closed under regular operations $\cup, \cdot, *$
DCFL is not closed under either of these ⊠

language L $\quad x \in L$, $xy \in L$



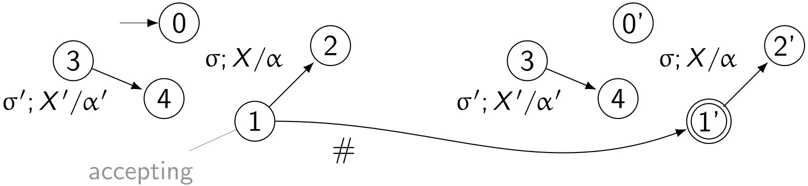
$L_0 = \{ a^n b^n \mid n \geqslant 1 \} \cup \{ a^n b^m c^n \mid m, n \geqslant 1 \}$

$\underline{a^n}\ \underline{b^n} \quad \underline{a^n}\ b^m\ \underline{c^n}$ different behaviour on b's



DCFL is closed under *pre*

$pre(L) = \{ x\#y \mid x, xy \in L \}$



$M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$ with $L = L(M)$
construct $M_1 = (Q_1, \Sigma \cup \{\#\}, \Gamma, \delta_1, q_1, Z_1, A_1)$ with $L(M_1) = pre(L)$

- $Q_1 = Q \cup Q'$ where $Q' = \{ q' \mid q \in Q \}$ primed copy
- $q_1 = q_{in}$, – $A_1 = A' = \{ q' \mid q \in A \}$
- $\delta_1 = \delta \cup \{ (p', \sigma, X, q', \alpha) \mid (p, \sigma, X, q, \alpha) \in \delta \} \cup$ two copies
  $\{ (p, \#, X, p', X) \mid p \in A, X \in \Gamma \}$ move to primed copy

ABOVE
For $K = \{a^n b^n \mid n \geqslant 1\} \cup a^* \cdot \{b^n c^n \mid n \geqslant 1\}$
we have $pre(K) = K\# \cup \{a^n b^n \# b^k c^{n+k} \mid n \geqslant 1\}$.
This language is not context-free, but K is, and thus the context-free languages are not closed under pre.

Again, this construction works because (for deterministic automata) the computation on uv *must* extend the computation on u.

Note the construction might not be deterministic at final states in original Q (like node 1 in the diagram), if that node has an outgoing λ-instruction.
There is however a method that avoids λ-instructions at accepting states.
Whenever the accepting state p has an outgoing instruction $(p, \lambda, A, q, \alpha)$, just predict the next letter σ read, and replace by all transitions $(p, \sigma, A, (q, \sigma), \alpha)$, where (q, σ) is a new state. Then keep simulating λ instructions, until σ is read.

CFG $G = (V, \Sigma, S, P)$

Definition (Nondeterministic Top-Down PDA)

$NT(G) = (Q, \Sigma, \Gamma, \delta, q_0, Z, A)$, as follows:

- $Q = \{q_0, q_1, q_2\}$
- $A = \{q_2\}$
- $\Gamma = V \cup \{Z\}$

- start $(q_0, \lambda, Z) \mapsto (q_1, SZ)$
- *expand* $(q_1, \lambda, A) \mapsto (q_1, \alpha)$ for $A \to \alpha$ in P
- *match* $(q_1, \sigma, \sigma) \mapsto (q_1, \lambda)$ for $\sigma \in \Sigma$
- finish $(q_1, \lambda, Z) \mapsto (q_2, Z)$ check empty stack

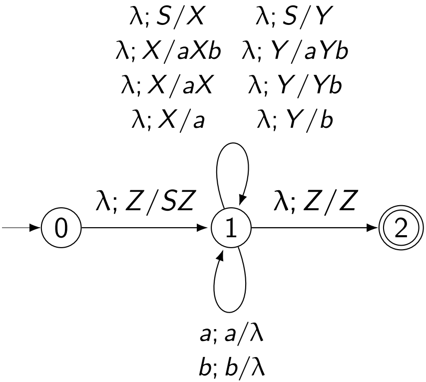
[M] Def 5.17

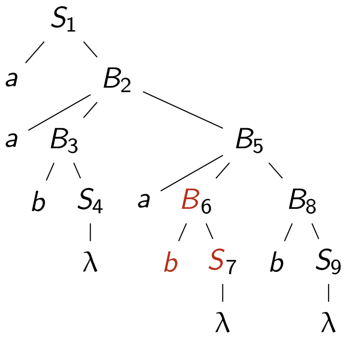
$L = \{ a^i b^j \mid i \neq j \}$

$S \to X \mid Y$ (choice!)

$X \to aXb \mid aX \mid a$ $(i > j)$

$Y \to aYb \mid Yb \mid b$ $(i < j)$

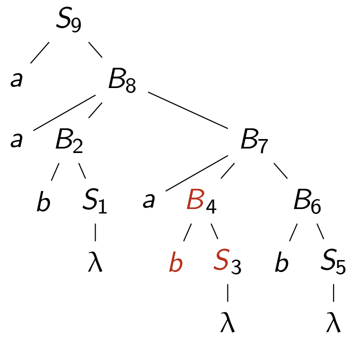




preorder: leftmost

$S \overset{\ell}{\Rightarrow} aB \Rightarrow aaBB \Rightarrow aabSB \Rightarrow$
$aabB \Rightarrow aabaBB \Rightarrow aababSB \Rightarrow$
$aababB \Rightarrow aababbS \Rightarrow aababb$

q_0	$aababb$	Z	
q_1	$aababb$	$S\ Z$	$1: S \to aB$
q_1	$aababb$	$aB\ Z$	
q_1	$a\ ababb$	$B\ Z$	$2: B \to aBB$
q_1	$a\ ababb$	$aBB\ Z$	
q_1	$aa\ babb$	$BB\ Z$	$3: B \to bS$
q_1	$aa\ babb$	$bSB\ Z$	
q_1	$aab\ abb$	$SB\ Z$	$4: S \to \lambda$
q_1	$aab\ abb$	$B\ Z$	$5: B \to aBB$
q_1	$aab\ abb$	$aBB\ Z$	
q_1	$aaba\ bb$	$BB\ Z$	$6: B \to bS$
q_1	$aaba\ bb$	$bSB\ Z$	
q_1	$aabab\ b$	$SB\ Z$	$7: S \to \lambda$
q_1	$aabab\ b$	$B\ Z$	$8: B \to bS$
q_1	$aabab\ b$	$bS\ Z$	
q_1	$aababb$	$S\ Z$	$9: S \to \lambda$
q_1	$aababb$	$\lambda\ Z$	
q_2	$aababb$	Z	



postorder: rightmost, in reverse

$S \overset{r}{\Rightarrow}_9 aB \Rightarrow_8 aaBB \Rightarrow_7 aaBaBB \Rightarrow_6 aaBaBbS \Rightarrow_5 aaBaBb \Rightarrow_4 aaBabSb \Rightarrow_3 aaBabb \Rightarrow_2 aabSabb \Rightarrow_1 aababb$

q_0	$aababb$	Z	
q_0	$a\ ababb$	$Z\ a$	
q_0	$aa\ babb$	$Z\ aa$	
q_0	$aab\ abb$	$Z\ aab$	
q_0	$aab\ abb$	$Z\ aabS$	$1: S \to \lambda$
q_0	$aab\ abb$	$Z\ aaB$	$2: B \to bS$
q_0	$aaba\ bb$	$Z\ aaBa$	
q_0	$aabab\ b$	$Z\ aaBab$	
q_0	$aabab\ b$	$Z\ aaBabS$	$3: S \to \lambda$
q_0	$aabab\ b$	$Z\ aaBaB$	$4: B \to bS$
q_0	$aababb$	$Z\ aaBaBb$	
q_0	$aababb$	$Z\ aaBaBbS$	$5: S \to \lambda$
q_0	$aababb$	$Z\ aaBaBB$	$6: B \to bS$
q_0	$aababb$	$Z\ aaBB$	$7: B \to aBB$
q_0	$aababb$	$Z\ aB$	$8: B \to aBB$
q_0	$aababb$	$Z\ S$	$9: S \to aB$
q_1	$aababb$	Z	
q_2	$aababb$	Z	

ABOVE

To write down the construction of the shift-reduce PDA for a given CFG, we have two technical problems.

Consider a production $A \to \alpha$

First the stack (in standard notation) now contains the string α in reverse.

Second, we pop α, that is, several symbols, rather than exactly one. This can be simulated by popping the symbols one-by-one, using separate instructions.

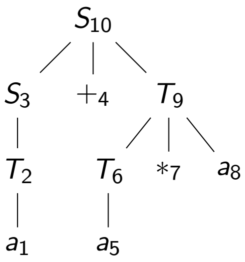
shift $(q_0, \sigma, X) \mapsto (q_0, \sigma X)$ for $\sigma \in \Sigma$, $X \in \Gamma$

reduce $(q_0, \lambda, \alpha^R) \mapsto (q_0, A)$ for $A \to \alpha$ in P

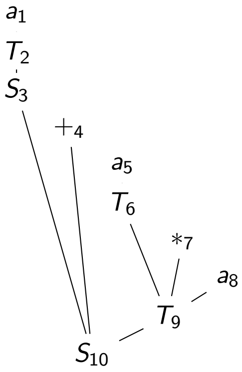
Example: algebraic expressions

shift-reduce

post-order reduction $\equiv$ rightmost derivation, bottom-up



stack [reverse]	input
Z	$a + a * a$
$Z\ a_1$	$+a * a$
$Z\ T_2$	$+a * a$
$Z\ S_3$	$+a * a$
$Z\ S_3\ +_4$	$a * a$
$Z\ S_3\ +_4\ a_5$	$*a$
$Z\ S_3\ +_4\ T_6$	$*a$
$Z\ S_3\ +_4\ T_6\ *_7$	a
$Z\ S_3\ +_4\ T_6\ *_7\ a_8$	
$Z\ S_3\ +_4\ T_9$	
$Z\ S_{10}$	
$-$	



Due to Chomsky, Evey, and Schützenberger (1962/3).

Theorem

Context-free grammars and Pushdown automata are equivalent.

$\hookrightarrow$(1) PDA acceptance by empty stack

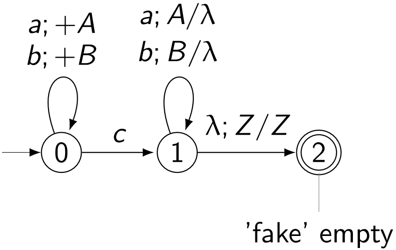
$\hookrightarrow$(2) triplet construction, CFG nonterminals $[p, A, q]$ for PDA computations

References

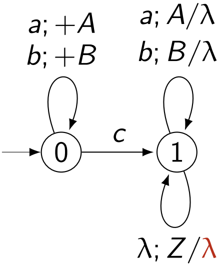
N. Chomsky. Context-free grammars and push-down storage. Quarterly Progress Report No. 65, Research Laboratory of Electronics, M.I.T., Princeton, New Jersey (1962)

R.J. Evey. The theory and application of pushdown store machines. In Mathematical Linguistics and Automatic Translation, NSF-IO, pages 217–255. Harvard University, May 1963,
and

M. P. Schützenberger. On context-free languages and pushdown automata. Inform. and Control, 6:217–255, 1963. doi:10.1016/S0019-9958(63)90306-1



check state



check stack

ABOVE

On many cases the PDA moves to the accepting state after checking that the stack is empty, when the topmost symbol is a special Z that always has been at the bottom of the stack.

It is more natural to accept directly by looking at the stack rather than by looking at the state. This leads to the notion of the *empty stack* language of a PDA.

$M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$

Definition

Language accepted by M by *empty stack* $L_e(M) =$
$\{ x \in \Sigma^* \mid (q_{in}, x, Z_{in}) \vdash^* (q, \lambda, \lambda)$ for some state $q \in Q \}$

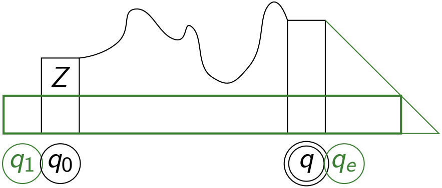
[M] D 5.27

Theorem

If M is a PDA then there is a PDA M_1 such that $L_e(M_1) = L(M)$.

[M] Th 5.28

Simulate $M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$
– empty pushdown 'at' final state
– prohibit early empty pushdown

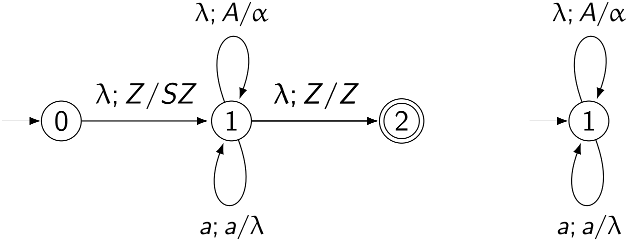


Construction PDA M_1 such that $L_e(M_1) = L(M)$

- $Q_1 = Q \cup \{q_1, q_e\}$
- $\Gamma_1 = \Gamma \cup \{\Box\}$
- new instructions:

  $(q_1, \lambda, \Box) \mapsto (q_0, Z_0\Box)$
  $(q, \lambda, X) \mapsto (q_e, \lambda)$ for $q \in A$, and $X \in \Gamma_1$
  $(q_e, \lambda, X) \mapsto (q_e, \lambda)$ for $X \in \Gamma_1$

$A \to \alpha \in P$, $a \in \Sigma$



Theorem

For every CFL *L there exists a single state* PDA *M such that* $L_e(M) = L$.

ABOVE

Now that we have empty stack acceptance we can reconsider the expand-match technique. In fact we do not need two extra states to introduce a bottom of stack symbol, and can make a single state PDA.

BELOW

The expand-match method can be used for any CFG. If we slightly restrict the grammars, we can combine each match with the expand step just before, that introduced the terminal. This gives a very direct translation between grammar and its leftmost derivation, and a single state PDA and its computation.

On this normal form each production is of the form $A \to a\alpha$, where $a \in \Sigma \cup \{\lambda\}$ can be the only terminal at the right. That means that any terminal pushed on the stack will be on top, and immediately will be matched.

cfg G	$\Longleftrightarrow$	1-pda M	
$A \to \alpha$		$(-, \lambda, A, -, \alpha)$	expand
		$(-, a, a, -, \lambda)$	match

normal form $\alpha \in (\Sigma \cup \{\lambda\}) \cdot V^*$

$A \to a\alpha$ $\quad (-, a, A, -, \alpha)$ combined

leftmost derivation $\Longleftrightarrow$ computation

	S		$(-, abbcbba, S)$
$\Rightarrow$	$a\,SA$	$\vdash$	$(-, bbcbba, SA)$
$\Rightarrow$	$ab\,SBA$	$\vdash$	$(-, bcbba, SBA)$
$\Rightarrow$	$abb\,SBBA$	$\vdash$	$(-, cbba, SBBA)$
$\Rightarrow$	$abbc\,BBA$	$\vdash$	$(-, bba, BBA)$
$\Rightarrow$	$abbcb\,BA$	$\vdash$	$(-, ba, BA)$
$\Rightarrow$	$abbcbb\,A$	$\vdash$	$(-, a, A)$
$\Rightarrow$	$abbcbba$	$\vdash$	$(-, \lambda, \lambda)$

Theorem

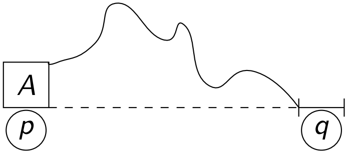
If $L = L_e(M)$ is the empty stack language of PDA *M, then there exists a* CFG *G such that $L = L(G)$.*

$M = (Q, \Sigma, \Gamma, \delta, q_{in}, Z_{in}, A)$

triplet construction

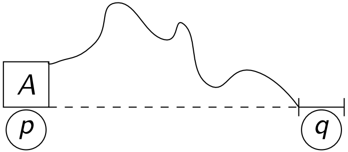
nonterminals $[p, A, q]$ $\quad p, q \in Q,\ A \in \Gamma$

$[p, A, q] \Rightarrow_G^* w \quad$ iff $\quad (p, w, A) \vdash_M^* (q, \lambda, \lambda)$

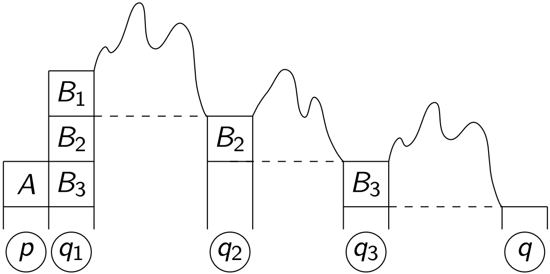


– nonterminals $[p, A, q]$ $p, q \in Q,\ A \in \Gamma$

$[p, A, q] \Rightarrow_G^* w$ iff $(p, w, A) \vdash_M^* (q, \lambda, \lambda)$



– productions $S \to [q_{in}, Z_{in}, q]$ for all $q \in Q$



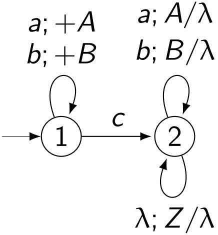
$[p, A, q] \to a\,[q_1, B_1, q_2][q_2, B_2, q_3] \cdots [q_n, B_n, q]$

for $(p, a, A) \mapsto (q_1, B_1 \cdots B_n) \in \delta$, and $q, q_2, \ldots, q_n \in Q$

$[p, A, q] \to a$ for $(p, a, A) \mapsto (q, \lambda) \in \delta$

$$L_e(M) = \{\ wcw^R \mid w \in \{a,b\}^*\ \}$$

twelve transitions $\Rightarrow$ 33 productions (!)



$X \in \{A, B, Z\}$

$(1, a, X, 1, AX)$	$[1,X,1] \to a\,[1,A,1][1,X,1]$
	$[1,X,1] \to a\,[1,A,2][2,X,1]$
	$[1,X,2] \to a\,[1,A,1][1,X,2]$
	$[1,X,2] \to a\,[1,A,2][2,X,2]$
$(1, b, X, 1, BX)$	...
$(1, c, X, 2, X)$	$[1,X,1] \to c\,[2,X,1]$
	$[1,X,2] \to c\,[2,X,2]$
$(2, a, A, 2, \lambda)$	$[2,A,2] \to a$
$(2, b, B, 2, \lambda)$	$[2,B,2] \to b$
$(2, \lambda, Z, 2, \lambda)$	$[2,Z,2] \to \lambda$
	not 'live'

Theorem

If L is a CFL, and R in REG, then L is CFL.

[M] Thm 6.13

product construction

PDA $M_1 = (Q_1, \Sigma, \Gamma, \delta_1, q_1, Z_{in}, A_1)$
NFA $M_2 = (Q_2, \Sigma, \delta_2, q_2, A_2)$

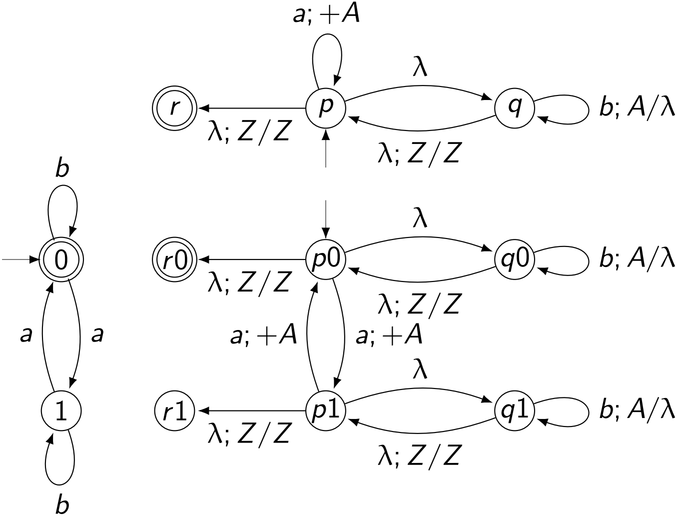
$Q = Q_1 \times Q_2 \qquad q_{in} = \langle q_1, q_2 \rangle \qquad A = A_1 \times A_2$

$(\langle p, q \rangle, \sigma, A) \mapsto_M (\langle p', q' \rangle, \alpha)$
whenever $(p, \sigma, A) \mapsto_{M_1} (p', \alpha)$ and $(q, \sigma, q') \in \delta_2$

$(\langle p, q \rangle, \lambda, A) \mapsto_M (\langle p', q \rangle, \alpha)$
whenever $(p, \lambda, A) \mapsto_{M_1} (p', \alpha)$ and $q \in Q_2$

Also $\hookrightarrow$CFG proof

$\{ a^n b^n \mid n \geqslant 1 \}^* \cap \{ w \in \{a, b\}^* \mid n_a(x) \text{ even} \}$



transition table

	E	Z	X	T
a	$E \to TZ$			$T \to a$
$+$		$Z \to X$	$X \to +TZ$	
$[$	$E \to TZ$			$T \to [E]$
$]$		$Z \to \lambda$		
λ		$Z \to \lambda$		

$a + [a + a]$:

input	stack	production
$a + [a+a]$	$E \perp$	$E \to TZ$
$a + [a+a]$	$TZ \perp$	$T \to a$
$a + [a+a]$	$aZ \perp$	—
$a + [a+a]$	$Z \perp$	$Z \to X$
$a + [a+a]$	$X \perp$	$X \to +TZ$
$a + [a+a]$	$+TZ \perp$	—
$a+ [a+a]$	$TZ \perp$	$T \to [E]$
$a+ [a+a]$	$[E]Z \perp$	—
$a+ [a +a]$	$E]Z \perp$	$E \to TZ$
$a+ [a +a]$	$TZ]Z \perp$	$T \to a$
$a+ [a +a]$	$aZ]Z \perp$	—
$a+ [a + a]$	$Z]Z \perp$	$Z \to X$
$a+ [a + a]$	$X]Z \perp$	$X \to +TZ$
$a+ [a + a]$	$+TZ]Z \perp$	—
$a+ [a+ a]$	$TZ]Z \perp$	$T \to a$
$a+ [a+ a]$	$aZ]Z \perp$	—
$a+ [a+a]$	$Z]Z \perp$	$Z \to \lambda$
$a+ [a+a]$	$]Z \perp$	—
$a+ [a+a] \lambda$	$Z \perp$	$Z \to \lambda$
$a+ [a+a] \lambda$	$\perp$	—

Section 6

Larger Families

6 Larger Families

⊠ (This section contains extras)

$G = (V, \Sigma, P, S)$
$P \subseteq V^* \times V^*$ finite
production $\alpha \to \beta$
derivation step $\gamma_1 \alpha \gamma_2 \Rightarrow \gamma_1 \beta \gamma_2$
language $L(G) = \{ w \in \Sigma^* \mid S \Rightarrow^* w \}$

noncontracting grammar $\alpha \to \beta, |\alpha| \leqslant |\beta|$

context-sensitive grammar $\alpha A \beta \Rightarrow \alpha \gamma \beta, |\gamma| > 0$
$A \to \lambda$ not allowed ignoring λ in languages

Example

$AnBnCn = \{ a^n b^n c^n \mid n \geqslant 1 \}$

noncontracting grammar

$S \to aSBc \mid abc$

$cB \to Bc \quad bB \to bb$

eg. $S \Rightarrow^* a^n S(Bc)^n \Rightarrow^* a^n abc(Bc)^n \Rightarrow^* a^n abB^n cc^n \Rightarrow^* a^{n+1} b^{n+1} c^{n+1}$

context-sensitive grammar

$S \to aSBc \mid abc$

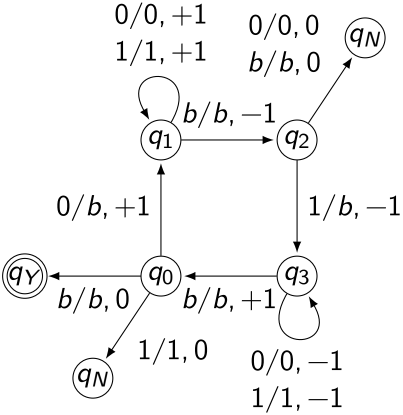
$CB \to CZ,\ CZ \to WZ,\ WZ \to WC,\ WC \to BC$

$aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc$

Example

$XX = \{ ww \mid w \in \{a, b\}^+ \}$

$$S \to L_a R_a \mid L_b R_b$$
$$L_\sigma \to a L_\sigma A \mid b L_\sigma B \qquad \sigma \in \{a, b\}$$
$$A\sigma \to \sigma A \qquad B\sigma \to \sigma B$$
$$AR_\sigma \to R_\sigma a \qquad BR_\sigma \to R_\sigma b$$
$$L_\sigma \to \sigma \qquad R_\sigma \to \sigma$$



q_0 op eerste symbool links, schrap 0

q_1 ga naar rechts, tot b

q_2 op laatste symbool rechts, schrap 1

q_3 ga naar links, tot b

	0	1	b
q_0	$q_1, b, +1$	$q_N, 1, 0$	$q_Y, b, 0$
q_1	$q_1, 0, +1$	$q_1, 1, +1$	$q_2, b, -1$
q_2	$q_N, 0, 0$	$q_3, b, -1$	$q_N, b, 0$
q_3	$q_3, 0, -1$	$q_3, 1, -1$	$q_0, b, +1$

deterministisch: functie

$$\delta : Q \times \{0, 1, b\} \mapsto Q \times \{0, 1, b\} \times \{-1, 0, +1\}$$

– automaton with two stacks   Turing Machine
– automaton with a queue (instead of stack)
– . . . with (several) ‘blind’ counters   Petri nets

	depricated
λ	Λ
DFA, NFA, NFA-λ	FA, NFA
$(Q, \Sigma, \delta, q_{in}, A)$	$(Q, \Sigma, q_0, A, \delta)$
$\equiv_L$	I_L
right-linear grammar	regular grammar

END.