INF210 Overview

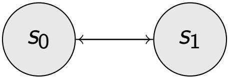
Åsmund Kløvstad

Universitetet i Bergen

December 12, 2020

State Transition Systems

- a tuple (S, R)
- $S = \{s_0, s_1 ...\}$
- $R \subseteq S \times S$
- example: switch $= (\{s_0, s_1\}, \{s_0 \times s_1, s_1 \times s_0\})$



Important Properties of STSs

- if a state has no rules going *from* it, it is a **terminal state**.
- if there is a sequence of steps from state a to state b, b is **reachable** from a. Write $R^*(a, b)$.
- if R is a function (in the set theory sense), then STS is **deterministic**.
- if all paths eventually converge (clarify/formalize?) the STS is **confluent**.
- switch is deterministic and confluent, and has no terminal states.

Labeled Transition Systems

Syntax

- extends STSs with labels
- a 3-tuple (S, L, R).
- $R \subseteq S \times L \times S$, often written $s \xrightarrow{a} s'$

Reachability

- consider the STS (S_L, R_L) with:
  - $S_L = S \times L^*$
  - $s \times l \cdot \mathbf{w} \rightarrow s' \times \mathbf{w} \in R_L$ iff $s \xrightarrow{l} s' \in R$
- s' is reachable from s by $\mathbf{w}$ if $s' \times \lambda$ is reachable from $s \times \mathbf{w}$.
- write $s \times \mathbf{w} \vdash_R^* s' \times \lambda$

Languages

- language over alphabet: $L \subseteq \Sigma^*$
- ex. $\{a^n b^n \mid n \in \mathbb{N}\}$ over $\Sigma = \{a, b\}$
- deciding membership in a language
- classifying classes of languages

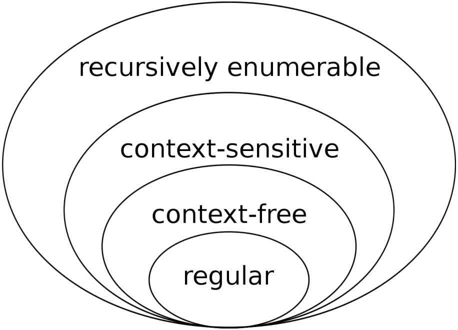


Figure: from https://en.wikipedia.org/wiki/Chomsky_hierarchy

Grammars

- $G = (\Sigma, N, S, \mathcal{R})$
- **generates** a language over Σ
- example: $G = (\{a\}, \{S\}, S, \{S \Rightarrow \lambda, S \Rightarrow Sa\})$ generates a^*
- let $\mathcal{A} = \Sigma \cup N$
- write $\mathbf{u} \Rightarrow_G^* \mathbf{v}$ for $\mathbf{u}, \mathbf{v} \in \mathcal{A}$ for "**u** generates **v**"

Machines

- FA/FSM's, PDA's, TM's
- **accepts** (recognizes) a language

Finite Automata

- $M = (\Sigma, Q, q_0, \Upsilon, F)$
- a finite LTS with initial and final states
- the language accepted by M is
  $L_M = \{\mathbf{w} \mid q_0 \times \mathbf{w} \vdash^*_{\Upsilon} q_i \times \lambda \text{ and } q_i \in F\}$
- example accepting a^+b^+

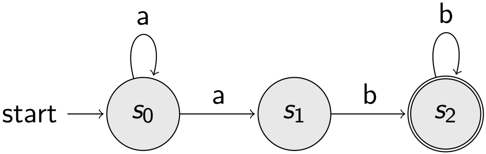


Figure: $M = (\{a, b\}, \{s_0, s_1, s_2\}, s_0, \{(q_0, a, q_0), (q_0, a, q_1), (q_1, b, q_2), (q_2, b, q_2)\}, \{q_2\})$

Regular Languages

- generated by regular grammars where rules are either $A \Rightarrow B$, $A \Rightarrow aB$ or $A \Rightarrow \lambda$ for $A, B \in N, a \in \Sigma$
- **regular** languages are inductively defined by:
  - $\emptyset, \{\lambda\}$ and $\{a\}$ are regular for $a \in \Sigma$
  - if L, L' regular, then $L \cup L', L \cdot L'$ and L^* are regular
- $\overline{L}$ and $L \cap L'$ are also regular

Kleene's Theorem

Theorem

a language is regular $\iff$ *it is accepted by a finite automata*

$\Rightarrow$

Construct FAs to the empty language, $\{\lambda\}$ and singleton languages. Then construct FAs for union, concatenation, and Kleene star.

$\Leftarrow$

Define $R(i, k, j) =$
$\{\mathbf{w} \mid q_j$ is reachable from q_i without visiting q_m with $m \geq k\}$
Then show
$R(i, k+1, j) = R(i, k, j) \cup R(i, k, k) \cdot R(k, k, k)^* \cdot R(k, k, j)$

Pumping Lemma

- if L is regular then there exists some $n > 0$ s.t for any $\mathbf{w} \in L$ longer than n, $\mathbf{w} = \mathbf{xyz}$ and $\mathbf{xy^kz} \in L$ for all $k \geq 0$
- makes sense because some state must be visited twice
- very useful for showing a language is **not** regular

Syntactic Monoid of a Language

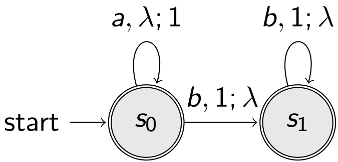
- $LR(\mathbf{w}) = \{\mathbf{x} \times \mathbf{y} \mid \mathbf{xwy} \in L\}$
- $\mathbf{u} \approx \mathbf{v} := LR(\mathbf{u}) = LR(\mathbf{v})$
- this is left-right invariant, so $[\mathbf{u}]_\approx \cdot [\mathbf{v}]_\approx := [\mathbf{uv}]_\approx$ is well defined
- $(\Sigma^*_{/\approx}, \cdot, [\lambda]_\approx)$ is the **syntactic monoid** of L.

Transformation Monoid of a FA

- given $M = (\Sigma, Q, q_0, \Upsilon, F)$
- each $a \in \Sigma$ induces a function $\overline{a} : Q \to Q$ by $q \mapsto \Upsilon(q, a)$
- with $\overline{\lambda} = id_Q$ and $\overline{\mathbf{w}a} = \overline{\mathbf{w}} \circ \overline{a}$ this extends to words
- $\overline{\mathbf{vw}} = \overline{\mathbf{v}} \circ \overline{\mathbf{w}}$, so $: \Sigma^* \to (Q \to Q)$ is a (monoid) homomorphism
- call the image $\overline{\Sigma^*} \subseteq (Q \to Q)$ the **transformation monoid** of M
- the syntactic monoid of $L \cong$ the transformation monoid of M_L (intrinsic/minimal FA)

Pushdown Automata

- finite automaton with a stack:
  - Σ: an alphabet
  - Γ: a stack alphabet
  - Q: a finite set of states
  - $q_0 \in Q$: an initial state
  - $\Delta \subseteq Q \times (\Sigma_\lambda \times \Gamma_\lambda \times \Gamma_\lambda) \times Q$: a transition relation
  - $F \subseteq Q$: a set of final states
- accepts in final state with empty stack
- example:



Context Free Grammars

- left hand side is a single non-terminal
- generate a superset of regular languages
- call this class **context free** languages
- ex. $S \Rightarrow \lambda, S \Rightarrow aSb$ generates $a^n b^n$

Context Free Languages

Theorem

a language is context free $\iff$ it is accepted by a pushdown automata

$\Leftarrow$

Given PDA $M = (\Sigma, \Gamma, \{s_0\}, s_0, !, \Delta)$ construct $G' = (\Sigma, \Gamma, !, \mathcal{R})$ with $\mathcal{R}$:

- $A \Rightarrow a\mathbf{W}$ for each $(a, A; \mathbf{W}) \in \Delta$
- $A \Rightarrow \mathbf{W}$ for each $(\lambda, A; \mathbf{W}) \in \Delta$

note: empty stack acceptance, start with ! on stack, allow pushing words

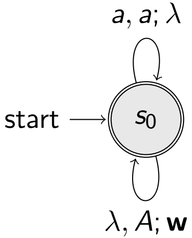
Context Free Languages

Theorem

a language is context free $\iff$ it is accepted by a pushdown automata

$\Rightarrow$

Given CFG $G = (\Sigma, N, S, \mathcal{R})$ construct:



for each $a \in \Sigma$ and $A \Rightarrow \mathbf{w} \in \mathcal{R}$

Pumping Lemma for CFLs

- given CFG $G = (\Sigma, N, S, \mathcal{R})$
- any $\mathbf{w} \in L(G)$ longer than $2^{|N|-1}$ can be written as $\mathbf{w} = \mathbf{uvwxy}$ s.t $\mathbf{uv}^n\mathbf{wx}^n\mathbf{y} \in L(G)$ for all $n \in \mathbb{N}$

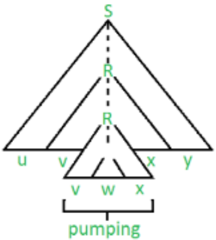


Figure: from https://www.geeksforgeeks.org/pumping-lemma-in-theory-of-computation/

Parsing

normal forms?

Turing Machine

- $M = (Q, \Sigma, q_0, \delta, H)$ with:
  - Q: a finite set of states
  - Σ: a finite alphabet which includes the blank symbol $\#$
  - $q_0 \in Q$: a starting state
  - $\delta : (Q \setminus H) \times \Sigma \to Q \times \Sigma \times \{-1, 0, 1\}$: a (possibly partial) transition function
  - $H = \{h_0, h_1\}$: a rejecting and an accepting halting state
- a head with state reading an infinite tape
- accepts a word **w** iff it halts in h_1 when started with **w** on the tape
- **decides** a language only if h_0 is reached for $\mathbf{w} \notin L$

Phrase-Structure Grammars

- grammars with no restrictions
- as powerful as Turing Machines

Halting Problem

Theorem

Standardize and enumerate all Turing Machines $\{T_i\}$. *Let*
$L = \{1^i01^j \mid T_i \text{ halts on input } 1^j\}$
Then there is no Turing Machine which decides L

- assume M decides L
- construct M' s.t it acts on input 1^i:
  1. change tape to 1^i01^i
  2. move to first cell
  3. emulate M on the tape
  4. if M rejects, loop infinitely. if M accepts, reject
- enumerate $M' = T_k$ and run M with 1^k01^k as input

Other stuff that might be nice

- diagrams for FA's in Kleene's theorem
- diagram from R(i, k, j) for Kleene's theorem
- procedure for making deterministic STS/LTS/FA
- what exactly is the intrinsic FA of a language
- regular grammar ¡-¿ FSM correspondence
- PSG ¡-¿ TM correspondence
- splicing languages (yikes)