Context Free Grammars and Pushdown Automata

Chomsky Normal Form: simplify, bounds on derivation complexity
A → BC, A → a, (S → ε)

any derivation has 2|w| − 1 steps
e.g., decide A_CFG, try all derivations from S
A_CFG ∈ P, dynamic programming (all terminals generating any substring)

CFG / PDA Equivalence proof:
⇒: nondet. choose rule, push nonterminals, match/pop terminals
⇔: nonterminal for each PDA state pair

Closure properties:
yes: union, concatenation, Kleene star
no: intersection, complement (only DCFL)

Turing Machines, Recognizability, Decidability

A Turing recognizable (by M₁) and A Turing recognizable (by M₂)
⇒ A decidable
run M₁ and M₂ in lockstep, see which halts first

Enumerating
Turing-recognizable ⇔ (recursively) enumerable
run for k steps on s₁, s₂, . . . s_k (dovetailing)
Decidable ⇔ enumerable in lexicographical order

Important: carefully consider language of given problem:
Is is {⟨M, w⟩ | ...} or just {⟨M⟩ | ...}? If the latter, can’t say “run M on w and . . .” (what is the input w?)

Problems for Recognizers

Acceptance:
A_M = {⟨M, w⟩ | M is a machine that accepts w}

Emptiness
E_M = {⟨M⟩ | M is a machine with L(M) = ∅}

Universality
ALL_M = {⟨M⟩ | M is a machine with L(M) = Σ*}

Equivalence
EQ_M = {⟨A, B⟩ | A, B are machines with L(A) = L(B)}

All decidable for DFA/NFA/REX (convert to minimized DFA)
A_CFG, E_CFG decidable, ALL_CFG, EQ_CFG not decidable
A_TM, HALT_TM, E_TM, etc. not decidable

Automata and Regular Languages

Interesting questions (accept, empty, equivalence) decidable
Useful argument: convert to DFA, minize ⇒ unique form.
Representative problem: ALL_NFA

ALL_NFA: in PSPACE
 nondeterministically guess rejected string
 simulate, NFA in ≤ |Q| states at each step (PSPACE)
Does this imply ∈ NP? No. (unknown whether NP, or coNP)
Useful idea: cross-product (shuffle, suffix languages), also use for PDAs.

Pumping Lemmas

Regular: any language string with |s| ≥ p can be divided into three pieces, s = xyz, with the conditions
1. xykz ∈ A for any i ≥ 0
2. |y| > 0
3. |xy| ≤ p

Context-Free: any language string with |s| ≥ p can be divided into five pieces, s = uvxyz, with the conditions
1. uv^iwxz ∈ A for any i ≥ 0
2. |vy| > 0
3. |vxy| ≤ p

Pump up or down (both useful)
Unidirectional:
if pumped string not in language, is not regular/context-free
there are other languages for which pumping holds
e.g., α^nβ^n∪ε^n(a+b)^*, k ≠ 1 nonregular, pumpable

COMPSCI 501: Formal Language Theory
Lecture 39: Review

Marius Minea
marius@cs.umass.edu
University of Massachusetts Amherst

1 May 2019
Proving Undecidability of a Language $L$

**Diagonalization** (directly)

- For proving $A_{TM}$ undecidable
  - Turing-recognizable.

**Reduction via Computation Histories**

- On input $(M, w)$, construct TM $M_1$ that will either have an empty language or not, depending on whether $M$ accepts $w$.
- On input $x$:
  - If $x \neq w$, reject.
  - Otherwise, run $A$ on $w (=x)$, accept if $A$ accepts $w$.
- Thus, $L(M_1) = \{w\}$ if $A$ accepts $w$, $\emptyset$ otherwise.
- Could use $D$ to decide $A_{TM}$.

**More General: Rice’s Theorem**

- Let $P$ be a nontrivial property of a Turing machine:
  - A language property, $L(M_1) = L(M_2) \leftrightarrow (P(M_1) \leftrightarrow P(M_2))$.
  - At least one TM has this property.
- Then $P$ is undecidable.
- Let $MP$ a Turing machine with that property.

**Proving a Language is not Turing-recognizable**

- Similar idea, but reduce from $\overline{A_{TM}}$.

**EQ$_{TM}$ not Turing-recognizable nor co-Turing-recognizable.**

- On input $(M, w)$, construct two machines:
  - $M_0$: rejects any input / $M_{all}$: accepts any input.
  - $M_w$: accept all/none, according to run of $M$ on $w$.
- $M_0$ not EQ $M_w$ iff $M$ accepts $w$: $A_{TM} \leq_m EQ_{TM}$, $\overline{A_{TM}} \leq_m EQ_{TM}$.
- $M_{all}$ EQ $M_w$ iff $M$ accepts $w$: $A_{TM} \leq_m EQ_{TM}$, $\overline{A_{TM}} \leq_m EQ_{TM}$.
- Mapping reduction: $f((M, w)) = (M_0, M_w)$ or $(M_{all}, M_w)$.

**Linear Bounded Automata:** $A_{LBA}$ decidable (finite number of configurations), but $E_{LBA}$ is not.

**Reduction via Computation Histories**

- Set of accepting TM computation histories of a TM checkable by an LBA.
- Another use: all strings that are not accepting computation histories on a string $w$.
  - can generate with a PDA / CFG $\implies$ ALL$_{CFG}$= undecidable (deciding $\neq \Sigma^*$ $\implies$ deciding $A_{TM}$).
  - can generate via extended regular expressions
    - use to prove: equivalence of extended regular expressions with exponentiation is EXPSPACE-hard.

**Mapping Reducibility**

- A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on input $w$, halts with just $f(w)$ on tape.

**Def.** A language $A$ is **mapping reducible** to language $B$ (written $A \leq_m B$) if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$ where for every $w, w \in A \iff f(w) \in B$.

**Using Mapping Reducibility**

**Decidability**

- If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.
- If $A \leq_m B$ and $A$ is undecidable then $B$ is undecidable.

**Turing-recognizability**

- If $A \leq_m B$ and $B$ is Turing-recognizable, then $A$ is Turing-recognizable.
- If $A \leq_m B$ and $A$ is not Turing-recognizable then $B$ is not Turing-recognizable.
Recursion Theorem

A TM can obtain and execute its own description.

Use: e.g., in proofs by contradiction (do something else than
description says)

e.g. assume $A_{TM}$ has a decider $H$

Construct a TM $B$:

On input $w$:
1. Obtain own description $⟨B⟩$
2. Run $H$ on input $⟨B, w⟩$
3. Do the opposite of $H$ (accept/reject)

Descriptive Complexity

The minimal description of a binary string $x$ is the shortest string
$⟨M, w⟩$ where $M$ halts on input $w$ with $x$ on tape.

The descriptive complexity (Kolmogorov complexity) is the length
of the minimal description: $K(x) = |d(x)|$

Def.: A string $x$ is $c$-compressible if $K(x) ≤ |x| − c$.

incompressible = not 1-compressible.

Most strings are close to incompressible.

Incompressible strings are undecidable.

Can only enumerate a finite subset.

NP-Completeness

Def. A language $B$ is NP-complete iff
1. $B$ is in NP
2. $B$ is NP-hard: for any $A$ in NP, we have $A ≤_P B$

Important: reduce from:

- to prove $B$ is NP-hard, show $C ≤_P B$ for NP-complete $C$
- reduce known NP-complete problem $C$ to target $B$
- reduce target problem $B$ from NP-complete problem $C$
- If $B$ is NP-complete and $B ∈ P$, then $P = NP$

All NP-complete problems are polynomially reducible to one another
(the hardest problems in NP)

Polynomial Verifiers and NP

Def. A verifier for a language $A$ is an algorithm $V$, where
$A = \{ w | V$ accepts $⟨w, c⟩$ for some string $c \}$

A polynomial-time verifier runs polynomial in the length of $v$.

A language is polynomially verifiable if it has a polynomial time verifier.

NP is the class of languages that have polynomial-time verifiers.

equivalent: A language is in NP iff it is decided by some

nondeterministic polynomial time Turing machine

Assymetry: Proving (witness) ≠ Disproving (no witness?)

Def.: $A$ is in coNP if $\overline{A}$ in NP. $P ⊆ NP ∩ coNP$

Time Complexity

Time complexity class $TIME(t(n))$ = all languages that are
decidable by an $O(t(n))$ (deterministic, single-tape) Turing machine.

A $t(n)$ multtape TM has an equivalent $O(t^2(n))$ single-tape TM.

multi-tape polynomial ⇒ single-tape polynomial

Every $t(n)$ nondeterministic TM has an equivalent $2^{O(t(n))}$
deterministic single-tape TM.

nondeterministic polynomial ⇒ single-tape exponential

Space Complexity

Savitch’s Theorem

For any function $f : N → R^+$, where $f(n) ≥ n$,

$NSPACE(f(n)) ≤ SPACE(f^2(n))$⇒ $NPSPACE = PSPACE$

PSPACE-completeness: Quantified Boolean Formula = valid ?
also admits log-space reducibility
L and NL

Model for sublinear space: read-only input tape, work tape gives space complexity.

log-space: constant number of pointers to input tape

PATH = { (G, s, t) | G is directed graph that has an s → t path } is NL-complete.

NL-coNL: Nondeterministic space complexity classes are closed under complementation (Immerman-Szelepcsényi)

\( \text{PATH} \in \text{NL} \), guess and re-count number of reachable nodes

Traversing \( \log n \)-depth tree in log-space:

pointer to node at each level is \( \log n \) \( \Rightarrow \) \( O(\log^2 n) \); instead:

keep constant-bit (bounded degree?) encoding of branch chosen

Complexity Hierarchies

L \subseteq NL = coNL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME

NL \subseteq PSPACE \quad P \subseteq \text{EXPTIME}

\( f : N \to N \) that is at least \( O(\log n) \) is space constructible if there is a \( O(f(n)) \) space TM that computes \( f(n) \) from \( 1^n \)

Space Hierarchy Theorem: For any space constructible function \( f : N \to N \), there exists a language that is decidable in \( O(f(n)) \) space but not \( o(f(n)) \) space.

\( \text{SPACE}(n^{c_1}) \subseteq \text{SPACE}(n^{c_2}) \) for any real \( c_1, c_2 > 0 \)

\( t : N \to N \) that is at least \( O(n \log n) \) is time constructible if \( t(n) \) is computable in time \( O(t(n)) \) from \( 1^n \).

Time Hierarchy Theorem: For any time constructible function \( t : N \to N \), there exists a language that is decidable in \( O(t(n)) \) time but not in time \( o(t(n)/\log t(n)) \).

\( \text{TIME}(n^{c_1}) \subseteq \text{TIME}(n^{c_2}) \) for any reals \( 1 \leq c_1 < c_2 \)

Circuit Complexity

Parameterized circuit families: uniformity, size-depth complexity

\( \text{NC}^i \): decidable by a uniform family of circuits with polynomial size and \( O(\log^i n) \) depth.

\( \text{AC}^i \): like \( \text{NC} \), but gates have unlimited fan-in (inputs).

\( \text{TC}^i \): like \( \text{AC} \), and also majority gates ("threshold circuits").

\( \text{NC}^i \subseteq \text{AC}^i \subseteq \text{TC}^i \subseteq \text{NC}^{i+1} \)

\( \text{NC} \subseteq \text{P} \) (generate + evaluate in polynomial time)

\emph{CIRCUIT-VALUE} is \text{P}-complete

\( \text{NC}^1 \subseteq \text{L} \): evaluate in log-space (constant-bit trick per level)

\( \text{NL} \subseteq \text{NC}^2 \): \emph{PATH}/transitive closure in \( \log^2 n \) depth

Branching Programs, Arithmetization

Barrington’s theorem:

depth \( d \) circuit \( \implies \) branching program of width 5 and length \( 4^d \).

log-depth circuit \( \implies \) poly-length program

Arithmetization:

Construct polynomials from branching programs / formulas for (dis)proving equivalence (increase probability of finding difference)

\[ f : N \to N \] that is at least \( O(\log n) \) is space constructible if there is a \( O(f(n)) \) space TM that computes \( f(n) \) from \( 1^n \)

Space Hierarchy Theorem: For any space constructible function \( f : N \to N \), there exists a language that is decidable in \( O(f(n)) \) space but not \( o(f(n)) \) space.

\( \text{SPACE}(n^{c_1}) \subseteq \text{SPACE}(n^{c_2}) \) for any real \( c_1, c_2 > 0 \)

\( t : N \to N \) that is at least \( O(n \log n) \) is time constructible if \( t(n) \) is computable in time \( O(t(n)) \) from \( 1^n \).

Time Hierarchy Theorem: For any time constructible function \( t : N \to N \), there exists a language that is decidable in \( O(t(n)) \) time but not in time \( o(t(n)/\log t(n)) \).

\( \text{TIME}(n^{c_1}) \subseteq \text{TIME}(n^{c_2}) \) for any reals \( 1 \leq c_1 < c_2 \)

Probabilistic Complexity Classes

\text{BPP} \) (bounded error): accepts/rejects with error probability \( \epsilon < 1/2 \)

Amplification lemma: can make error \( 2^{-p(n)} \) for any polynomial

\text{RP} \) (randomized poly-time): always rejects when it should re-runs make acceptance error arbitrarily small

\text{coRP} \) : always accepts when it should

Clearly \( \text{RP} \subseteq \text{BPP} \) (reject error is zero), likewise \( \text{coRP} \subseteq \text{BPP} \)

\( \text{P} \subseteq \text{RP} \) : no nondeterminism, always right answer

\( \text{RP} \subseteq \text{NP} \) : NTM needs no coin, guesses correct path

\text{BPP} ? \text{NP} \) (no relation is known). It it believed that \( \text{P} = \text{BPP} \).

Polynomial identity testing is in \text{coRP}, unknown if in \text{P}.

Alternation

universal states \( \text{(AND, \wedge) accepts if all successors do} \)

existential states \( \text{(OR, \vee) accepts if some successor does} \)

Example: \emph{MIN-FORMULA} in AP:

universally select all formulas \( \psi \) shorter than \( \phi \)

existentially select a truth assignment, eval \( \psi \) and \( \phi \)
Alternation and Space/Time Complexity Connections

1. $\text{ATIME}(f(n)) \subseteq \text{SPACE}(f(n)) \subseteq \text{ATIME}(f^2(n))$ for $f(n) \geq n$
2. $\text{ASPACE}(f(n)) = \text{TIME}(2^{O(f(n))})$ for $f(n) \geq \log n$

Consequences:
- $\text{AL} = \text{P}$
- $\text{AP} = \text{PSPACE}$
- $\text{APSPACE} = \text{EXPTIME}$

Polynomial Time Hierarchy

- bound number of alternations between $\land$ and $\lor$
- hierarchy within $\text{AP} = \text{PSPACE}$

- $\Sigma_i$ alternating TM: at most $i$ runs of existential or universal steps, starting with existential steps.
- $\Pi_i$ alternating TM: at most $i$ runs of existential or universal steps, starting with universal steps.

- $\Sigma_i \text{P} = \bigcup_k \Sigma_i \text{TIME}(n^k)$
- $\Pi_i \text{P} = \bigcup_k \Pi_i \text{TIME}(n^k)$

- $\text{PH} = \bigcup_i \Sigma_i \text{P} = \bigcup_i \Pi_i \text{P}$
- $\text{P} = \Sigma_0 \text{P} = \Pi_0 \text{P}$ (no nondeterminism, no alternation)
- $\text{NP} = \Sigma_1 \text{P}$, $\text{coNP} = \Pi_1 \text{P}$

Interactive Proofs

- $\text{IP}$ a verifier (polynomially computable function) $V$ exists such that for every string $w$
  1. for some function $P$, $w \in A \implies \Pr[V \leftrightarrow P \text{ accepts } w] \geq \frac{2}{3}$
  2. for any function $\tilde{P}$, $w \notin A \implies \Pr[V \leftrightarrow \tilde{P} \text{ accepts } w] \leq \frac{1}{3}$
- some (honest) prover $P$ can produce likely correct accept
- no (dishonest?) prover $\tilde{P}$ can produce likely incorrect accept

- Can use amplification to make error probability arbitrarily small.
- $\text{BPP} \subseteq \text{IP}$ (need no $\text{P}$/ignore)
- $\text{NP} \subseteq \text{IP}$ (never wrongly accepts, try often enough)
- $\text{IP} = \text{PSPACE}$ (Shamir’s Theorem)