COMPSCI 501: Formal Language Theory

Lecture 14: Decidable Problems

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

22 February 2019

Turing-Recognizable and Decidable Languages

Recall: **Recursively Enumerable = Recognizable**

- Enumerate all strings in language
- Compare with given input; if equal, accept

Recall: **Decidable** $\subsetneq$ **Recognizable**

Both: will accept all strings *in* language

Decide: will reject all strings *not in* language

Recognize: **may loop** forever on strings *not in* language

Can't we just wait some time and then report "no" ?
might not be able to bound potentially good answers

First-Order Logic: Recognizable

Consistency: every formula that can be **proved** is **valid**

Completeness: every **valid** formula can be **proved**

Caveat: only says something about **valid** formulas:
if not valid, can't be proved (consistency – good!)
but might not be able to **disprove** it

Connection:
can *enumerate* all proofs (of anything), and check none matches
can't tell when to stop

Can recognize language (**valid** formulas)
Can't recognize **complement** (formulas that are not valid)
not **decidable**

Acceptance Problem for DFA

Will a given DFA accept a given string ?

Is this the same as asking "is a regular language decidable"?
No. Language of strings vs. language of pairs (DFA, string)

$A_{\text{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA that accepts string } w\}$

TM with control of *given* DFA (just scans tape) vs.
TM that can **simulate any** DFA *description*
check if valid DFA encoding
store automaton state and input position on tape
update state/position at each step according to DFA.

Acceptance Problem for NFA

$A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA that accepts string } w\}$

How would we simulate a **nondeterministic** automaton ?

- With a nondeterministic Turing Machine
  when processing transition list, noindeterministically choose current one, or skip to next
- By simulating NFA computations
  breadth-first traversal of computation tree
  keep *list* with current state for each execution branch
- By converting NFA to DFA (subset construction) – step 1
  and then using TM for DFS acceptance problem – step 2

Same way for string membership in regular expression

$A_{\text{REX}} = \{\langle R, w\rangle \mid R \text{ is a regular expression that generates string } w\}$

Testing Regular Language Emptiness

Why test for an empty language ?
incompatible constraints: $L_1 \cap L_2 = \emptyset$
check language inclusion $L_1 \subseteq L_2 \leftrightarrow L_1 \cap \overline{L_2} = \emptyset$

$E_{\text{DFA}} = \{\langle A\rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

traverse DFA states from start (BFS/DFS/any)
until accept state reached or no new states marked

Equivalence of DFAs

$EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A, B$ are DFAs and $L(A) = L(B)\}$

$L(A) = L(B) \leftrightarrow L(A)\Delta L(B) = \emptyset$

Construct automaton for $L(A)\Delta L(B)$
Check emptiness

CFL Membership Test

$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G$ is a CFG that generates string $w\}$

Could we enumerate all derivations and check them in turn ?
this works for strings *in* the language (*recognizer*)
will never stop if $w \notin L(G)$ and G has infinitely many derivations (usually the case)

Can we bound the number of derivations required for w ?

Chomsky normal form: $A \to BC$, $A \to a$ (or $S \to \varepsilon$)
one derivation to get each terminal (n)
$n - 1$ derivations to get from S to n nonterminals

Generate all derivations of $2n - 1$ steps, check if one generates w

CFL Emptiness Test

$E_{\mathsf{CFG}} = \{\langle G\rangle \mid G$ is a CFG and $L(G) = \emptyset\}$

When would the language of a grammar be empty?
when derivations can't reach any nonterminal strings

Check for *each* variable whether it can generate string of terminals
$=$ compute a boolean $t(V)$; answer given by $t(S)$
rule $A \to U_1U_2 \dots U_k$ tells us value $t(A) = t(U_1) \wedge \dots \wedge t(U_k)$
if all U_i marked, mark A
repeat until stable (no new symbols marked)

Can use for lots of other problems:
does variable generate empty string?
What terminals can strings from V contain/start/end with?

Least fixpoint computation: always terminates if set of computed values is monotone and bounded.

CFG Equivalence ?

$EQ_{\mathsf{CFG}} = \{\langle G, H\rangle \mid G, H$ are CFGs and $L(G) = L(H)$

Idea from DFAs: $L(G) = L(H) \leftrightarrow L(G)\Delta L(H) = \emptyset$

$L(G)\Delta L(H) = (L(G) \setminus L(H)) \cup (L(H) \setminus L(G))$

But CFGs are not closed under complement! $\Rightarrow$ can't use!

In fact, CFG equivalence is **undecidable** (we'll see).

CFLs are Decidable

Theorem: Every context-free language is decidable

Why can't we just simulate the PDA for L with a TM (more powerful)?
DFAs/NFAs always stop (finite string), but PDAs may not
TM "clone" of PDA may also not stop $\Rightarrow$ *recognizer, not decider*

But we already have a Turing-machine that for *any* (G, w) checks whether $w \in L(G)$
particularize it for given G (build G in)
with input w, will decide $L(G)$

regular $\subset$ **context-free** $\subset$ **decidable** $\subset$ Turing-**recognizable**