Theory of Computation [a]

Decidable Problems Concerning Context-Free Languages

Teodor Rus

rus@cs.uiowa.edu

The University of Iowa, Department of Computer Science

Topics

- String generation problem by a CFG;
- The problem of determining whether the language of a CFG is empty;
- The problem of determining whether two Context-Free Languages are equal;
- The problem of determining whether a context-free language is decidable.

Methodology

1. We first formulate the problem;
2. Then we transform the problem in to a language;
3. Then we show how to construct a TM that decides the language at (2).

String Generation Problem

Problem:

Describe an algorithms to test whether a CFG $G = (V, \Sigma, R, S)$ generates a particular string $w \in \Sigma^*$.

The Language

$A_{CFG} = \{\langle G, w\rangle | G$ is a CFG that generates $w\}$.

In other words, for a given CFG G and w a string, does $w \in L(G)$?;

Solution

Theorem 4.7: A_{CFG} is a decidable language.

Proof idea: For a CFG G and a string w, consider all derivations generated by G checking whether any is a derivation of w.

Problem: Since there are infinitely many derivations this idea does not work. If G does not generate w the algorithm doesn't halt.

Note:

this idea provides a recognizer but not a decider.

A Better Idea

Make the recognizer a decider. For that we need to ensure that the algorithm tries only finitely many derivations.

Fact 1

If G is a CFG in Chomsky normal form then for any $w \in L(G)$ where $|w| = n$ exactly $2n - 1$ steps are required for any derivation of w.

Proof:

1. Derivation rules of a Chomsky normal form are of the form: $A \rightarrow A_1 A_2 | A \rightarrow a$.
2. The rule $A \rightarrow A_1 A_2$ adds 1 to the length of w. That is, if $|w| = n$ then $S \Rightarrow A_1 A_2 \Rightarrow A_1 A_2 A_3 \Rightarrow A_1 A_2 \ldots A_n$, using $n - 1$ steps.
3. To eliminate A_1,A_2,. . ., A_n by rules of the form $A \rightarrow a$ we need another n steps.

Conclusion: only $2n - 1$ steps are required.

Checking CFG derivations

- Convert G into Chomsky normal form;
- For a string w of length $|w| = n$, $n > 1$, check all derivations with 2n-1 steps to determine whether G generates w;
- For a string w, of length $|w| \leq 1$, check all one-step derivations to determine whether G generates w.

Note: since we can convert G into a Chomsky normal form (see Section 2.1), this is a good idea.

Proof of Theorem 4.7

The TM S that decide A_{CFG} is:

S = "On input $\langle G, w\rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent grammar in Chomsky normal form;
2. List all derivations with $2n-1$ steps, $n = length(w)$ except if $n \leq 1$;for $n \leq 1$ list all derivations with 1 step;
3. If any of the derivations listed above generates w, *accept*; if not *reject*."

Observations

- The problem of testing whether G generates w is actually the parsing problem of parsing programming languages;
- The algorithm performed by S is very inefficient. Early algorithm based on the same idea is $\mathcal{O}(n^3)$;
- Theorem 2.20 proves that CFG are equivalent with PDA and provides a mechanism to convert a CFG into a PDA and vice-versa.

Conclusion: everything about decidability of problems concerning CFG applies equally to the decidability of problems concerning PDAs.

Language Emptiness Problem

Problem:

Describe an algorithm to test whether the language generated by a CFG $G = (V, \Sigma, R, S)$ is empty.

The Language:

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

Solution

Theorem 4.8: E_{CFG} is a decidable language.

Proof idea:

- To test whether $L(G)$ is empty we need to test whether the axiom of G can generate a string of terminals;
- However, we may solve a more general problem, determining *for each variable* whether that variable can generate a string of terminals;
- When the algorithm determines that a variable can generate a string of terminals the algorithm mark that variable;
- The algorithm start by marking first all terminals. Then it marks variables that have on their rhs in some rules only terminals, i.e., marked symbols, and so one.

Proof of theorem 4.8

Construct the TM R:

R = "On input $\langle G \rangle$ where G is a CFG:

1. Mark all terminal symbols of G;
2. Repeat until no new variable get marked:

   Mark any variable A where G has a rule $A \rightarrow u_1 u_2 \ldots u_k$ and each symbol $u_1, u_2, \ldots, u_k$ has already been marked;
3. If the start symbol of G (i.e., the axiom) is not marked, *accept*; otherwise *reject*."

Decidability of CFLs

Problem:

Describe an algorithm to test whether a string is an element of a context free language.

In other words, for a given context-free language A is there an algorithm that determine whether an arbitrary string is an element of A?

The Language:

$L_{CFL} = \{\langle A\rangle | A$ is a context-free language $\}$

Solution

Theorem 4.9 Every context-free language is decidable.

Proof idea Let A be a CFL. This means that A is generated by a context-free grammar G and thus, there is a PDA P_G that is equivalent to G. Hence, if $w \in A$ then P_G recognizes w.

Constructing a Decider

- A bad idea: convert a PDA for A directly into a TM.

  Some branches of a PDA computation may go forever, reading and writing the stack without coming to a halt.

  The TM simulation would then have some non-halting branches in its computation and thus it would not be a decider

- A good idea: *use the TM S that decides string generation problem by converting G into Chomsky normal form*.

Proof of Theorem 4.9

Let G be a CFG for A, i.e. $L(G) = A$. Design a TM M_G that decides A by building a copy of G into M_G:

M_G = "On input w:

1. Run TM S on input $\langle G, w\rangle$;
2. If this machine accepts, *accept*; if it rejects, *rejects*."

Note: TM S converts G to Chomsky normal form, and produces all derivations of length $2n - 1$ where $n = |w|$. Then check if w is among the derived strings.

Fact 2

Class of CF languages is not closed under intersection.

Proof: By construction.

- Consider the CF languages:
  $A = \{a^m b^n c^n | m, n \geq 0\}$ and, $B = \{a^n b^n c^m | m, n \geq 0\}$
  generated by the grammars $G_A = (\{S, R, T\}, \{a, b, c\}, P_A, S)$,
  where $P_A = \{S \rightarrow RT, R \rightarrow aR|\epsilon, T \rightarrow bTc|\epsilon\}$ and
  $G_B = (\{S, R, T\}, \{a, b, c\}, P_B, S)$, where
  $P_B = \{S \rightarrow TR, T \rightarrow aTb|\epsilon, R \rightarrow cR|\epsilon\}$ respectively;
- $L(A) \cap L(B) = \{a^n b^n c^n | n \geq 0\}$ which is not a CFL;
- This establishes Fact 2.

Fact 3

Class of CF languages is not closed under complementation.

Proof: by contradiction.
Assume that CFL is closed under complementation.

- If G_1 and G_2 are two CFG-s then $\overline{L(G_1)}$ and $\overline{L(G_2)}$ are CFL;
- Then $\overline{L(G_1)} \cup \overline{L(G_2)}$ is a CFL. Hence, $\overline{\overline{L(G_1)} \cup \overline{L(G_2)}}$ is a CFL;
- By DeMorgan's law $\overline{\overline{L(G_1)} \cup \overline{L(G_2)}} = L(G_1) \cap L(G_2)$, a contradiction because by Fact 2 we established that class of CFL is not closed under intersections.

Equality of CFLs

Problem:

For two CFLs L_1 and L_2 is $L_1 = L_2$ true?

The Language:

$EQ_{CFL} = \{\langle L_1, L_2\rangle | L_1, L_2$ are CFLs, $L_1 = L_2\}$

Note: since L_1, L_2 are context free languages it means that there exists context free grammars G and H such that $L_1 = L(G)$ and $L_2 = L(H)$. Hence, the language of the problem can also be formulated as follows:

The Language:

$EQ_{CFL} = \{\langle G, H\rangle | G, H$ are CFGs, L(G) = L(H)$\}$

Solution

Theorem 4.10: EQ_{CFL} is not decidable!

Note: Since class of CF languages is not closed under intersection and complementation (as proven by Fact 2 and Fact 3), we cannot use the symmetric difference for the construction of a TM that would decide EQ_{CFL}.

Methodology (review)

To solve decidability problems concerning relations between languages one should proceed as follows:

- Understand the language relationship involved;
- Transform the language relationship into a language expression using closure operators on decidable languages;
- Design a TM that decide the resulting language using as components the TMs that decide the language components;

Example 1

Equivalence of DFA and REX:

- Consider the problem of testing whether a DFA and a regular expression are equivalent.
- Express this problem as a language and show that this language is decidable.

Solution

- The language:
  $EQ_{DFA,REX} = \{\langle A, R\rangle |$ A is a DFA, R is a RE and $L(A) = L(R)\}$
- The following TM E decides $EQ_{DFA,REX}$:
  E = "On input $\langle A, R\rangle$
  1. Convert R to an equivalent DFA B
  2. Use the TM F for deciding EQ_{DFA} on input $\langle A, B\rangle$
  3. If F accepts, *accept*; if F rejects, *reject*."

Note: F constructs C, the DFA that recognizes the symmetric difference of A and B, $L(C) = L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B)$, and test if $L(C)$ is empty!

Example 2

Decidability of Σ^*

- **Problem:**
  For Σ a finite alphabet, is the language Σ^* decidable?
- **Language:**
  $ALL_{DFA} = \{\langle A\rangle | A$ is a DFA that recognize $\Sigma^*\}$

  Show that ALL_{DFA} is decidable

Solution

The TM L that decides ALL_{DFA} uses the fact that $\overline{L(A)}$ is regular.

L = "On input $\langle A \rangle$ where A is a DFA:

1. Construct DFA B that recognizes $\overline{L(A)}$ by swapping accept and unaccept states in A;
2. Run the TM T that decides the emptiness E_{DFA} on B;
3. If T accepts, *accept*; if T rejects *reject*."

Note: if T accepts it means that $L(B) = \emptyset$. But $L(B) = \overline{L(A)}$.

That is, $\Sigma^* \setminus L(A) = \emptyset$, i.e. $L(A) = \Sigma^*$.

Example 3

Using CFG and REX

- **Problem:**
  show that the problem of testing whether a CFG generates some string in 1^* is decidable.
- **Language:**
  $A = \{\langle G\rangle | G$ is a CFG and $1^* \cap L(G) \neq \emptyset\}$

Solution

Assume that G is over $\{0,1\}^*$. Then we need to show that the language $A = \{\langle G\rangle | G$ is a CFG over $\{0,1\}^*$ and $1^* \cap L(G) \neq \emptyset\}$ is decidable.

Since 1^* is regular and $L(G)$ is CFL then $1^* \cap L(G)$ is a CFL.

Hence the TM X that decides A is:

X = “On input $\langle G\rangle$ where G is a CFG:

1. Construct CFG H such that $L(H) = 1^* \cap L(G)$;
2. Run the TM R that decides the language E_{CFG} on $\langle H\rangle$;
3. If R accepts, *reject*; if R rejects, *accept*."

Note: if R accepts it means that $L(H) = 1^* \cap L(G) = \emptyset$.

That is, $\forall w \in 1^*$, $w \notin L(G)$, hence, X should reject.

Example 4

Example regular expressions

- **Problem:**
  Is the language generated by a particular regular expression decidable? For example, is the language of regular expressions that contain at least one string that has the pattern “111" as a substring decidable?
- **Language:**
  $A = \{\langle R\rangle | R$ is a regular expression describing a language that contain at least one string w that has “111" as a substring (i.e., $w = x111y$ where x and y are strings$\}$

Solution

The language A is decidable. The reason is that language A can be expressed using regular operators as $L(\Sigma^* \circ 111 \circ \Sigma^*) \cap L(R)$.

Hence, the TM X that decides A is:

X = "On input $\langle R \rangle$ where R is a regular expression:

1. Construct the DFA E that accepts $\Sigma^* 111 \Sigma^*$;
2. Construct the DFA B that accepts $L(B) = L(R) \cap L(E)$;
3. Run TM T that decide E_{DFA} on input $\langle B \rangle$;
4. If T accepts *reject*; if T rejects *accept*."

Note: if T accepts, it means that $L(R) \cap L(E) = \emptyset$, i.e., $\forall x, y \in \Sigma^*$, $x111y \notin L(R)$.

The Halting Problem

Problem:

Develop an algorithm to test whether a Turing machine accepts a given input string.

By analogy with A_{DFA} and A_{CFG} we call the corresponding language A_{TM}.

Language:

$$A_{TM} = \{\langle M, w\rangle | M\ is\ a\ TM\ and\ M\ accepts\ w\}$$

Solution

Theorem 4.11 A_{TM} is undecidable.

Contrasting A_{DFA} and A_{CFG}, which are decidable,A_{TM} is not decidable. Since A_{TM} is recognizable this result shows the relationship among the classes of languages considered so far:

$$RL \subseteq CFL \subseteq DecidableL \subseteq RecognizableL$$

Observations

- A_{TM} is however Turing-recognizable;
- Hence, Theorem 4.11 shows that recognizers are more powerful than deciders;
- Requiring a TM to halt on all inputs restricts the kind of languages that it can recognize.

A recognizer for A_{TM}

The following TM U recognizes A_{TM}:

U = "On input $\langle M, w\rangle$, where M is a TM and w is a string:

1. Simulate M on input w;
2. If M ever enters its accept state, *accept*;
   if M ever enters its reject state, *reject*."

Observations

1. Machine U loops on the input $\langle M, w\rangle$ if M loops on w. This is why U does not decide A_{TM}.
2. If the algorithm had some way to determine that M was not halting on w, it could reject. This is why it is called the halting problem.
3. However, as we demonstrate Theorem 4.11, an algorithm has no way to make this determination.

Observations

1. The TM U is interesting in its own right because it is an example of the *Universal Turing Machine*, first proposed by Turing;
2. U is called universal because it is capable to simulate any other Turing machine from the description of that machine;
3. The universal TM played an important role in the theory of computation by stimulating the development of stored-program computers.

Fact 4

The algorithm implemented by a real computer processor while executing a program is described by:

```
while ((PC).opcode is not halt)
     Execute PC;
     PC := Next(PC);
```

and thus behaves like U.