Introduction to Computability Theory

Lecture 12: Decidable Languages
Prof. Amos Israeli

Introduction and Motivation

In this lecture we review some decidable languages related to regular and context free languages.

In the next lecture we will present a undecidable language.

Decidable Languages (rerun)

A language $L$ over $\Sigma^*$ is *decidable* if there exists a Turing machine recognizing $L$, that stops on every input.

Objects as Input

The input for a TM is always a string. If we want to give some other object, e.g. an automaton or a grammar, the object must be encoded as a string. Encoding can be straight forward.

For example: An undirected graph $G$, is encoded by specifying its nodes and its edges. $G$’s encoding is denoted as $\langle G \rangle$. 

Objects as Input
Objects as Input

A TM that gets as input the encoding of an undirected graph $\langle G \rangle$, starts its computation by verifying that the encoding is well formed. If this is not the case, the input is rejected.

In the future we will regard input representations as implementation details and conveniently ignore them.

Example: Connected Graphs

Let $A$ be the language of strings representing undirected connected graphs.

$A = \{ \langle G \rangle \mid G$ is a connected undirected graph $\}$

Note: in this way we form a computational problem as a language recognition problem.

In the next slide we present a high level description of a TM that decides $A$.

Example: Connected Graphs

$M$ = “On input $\langle G \rangle$, the encoding of a graph $G$:
1. Select a node of $G$ and mark it.
2. Repeat until no new nodes are marked:
   3. For every node of $G$, $v$, if $v$ is connected to a marked node, mark it as connected.
4. If all nodes are marked accept otherwise reject.”

Connected Graphs-Representation

Consider a representation of a graph by two lists:
1. List of nodes (natural numbers).
2. List of edges (pairs of nodes).

Note: We do not specify the alphabet which can be binary, decimal or other. The idea is to ignore insignificant implementation details.
Input Verification

Check that the input consists of:
1. A list of nodes (natural numbers) with no repetition.
   To check uniqueness use the machine presented on yesterday’s discussion.
2. A list of edges: A list of pairs of nodes that appear in the previous list.

Implementation

1. Select a node of $G$ and mark it -
   Mark the first node on the list. One way to do it is to “dot” its leftmost digit.
2. Repeat until no new nodes are marked -
   Mark the first un-dotted node on the list by underlining (different from dotting) its leftmost digit.

2. Repeat until no new nodes are marked:
   Underline first un-dotted node on list, $n_1$.
3. For every node of $G$, $v$, if $v$ is connected to a marked node, mark it as connected –
   Underline the first dotted node on the list, $n_2$.
   Search edges for $(n_1, n_2)$ edge.

Implementation

2. Repeat until no new nodes are marked:
   Underline first un-dotted node on list as $n_1$
3. For every node of $G$, $v$, if $v$ is connected to a marked node, mark it as connected –
   Underline the first dotted node on the list $n_2$, search edges for $(n_1, n_2)$ edge.
   If found mark $n_1$ as dotted. Remove underlines go back to 2.
2. Repeat until no new nodes are marked:
   Underline first un-dotted node on list as $n_1$
3. For every node of $G$, $v$, if $v$ is connected to a marked node, mark it as connected –
   Underline the first dotted node on the list $n_2$,
   search edges for ($n_1,n_2$) edge.
   If not underline the next dotted node on the list as $n_2$.

Implementation

13

Decision Problems on Automatons

Now we turn to deal with some problems related to regular and CF Languages. Typical problems are:
• Deciding whether a DFA accepts a language.
• Deciding whether a language is empty.
• Deciding whether two languages are equal, etc.

Finite Automatons are Decidable

Consider the language
$A_{DFA} = \{ B, w \} | B$ is a DFA accepting $w \}$

Note that once again we formulate a computational problem as a membership problem.

Theorem
$A_{DFA}$ is a decidable language.
**Proof**

Consider $M=\langle \text{DFA, string} \rangle$ input $\langle B, w \rangle$:

1. Simulate $B$ on input $w$.
2. If $B$ accepts - accept. Otherwise - reject.

**Implementation**

The encoding of $\langle B, w \rangle$ can be straightforward:

The five components of $B$ are listed on $M$'s tape one after the other.

TM $M$ starts its computation by verifying that the encoding is well-formed.

If this is not the case, the input is rejected.

**Implementation**

Following that $M$ simulates on $B$'s computation on $w$ in a way, very similar to the way a computer program will do.

**Simulation of a DFA Initialization**

Assume that the DFA is encoded by a list of its components. TM $M$ should first verify that the string representing $B$ is well-formed.

Than it should use a “state dot” to mark $B$'s initial state as its current state and an “input dot” to mark $w$'s first symbol as the current input symbol.
Simulation of a DFA

Following that, $M$ scans the substring representing $B$'s transition function to find the transition that should take place for the current state and the current input symbol. Once the right transition is found, the new current state is known.

Finite Automatons are Decidable

Now we turn to consider the language

$$A_{NFA} = \{ \langle B, w \rangle | B \text{ is a NFA accepting } w \}$$

and prove:

**Theorem**

$A_{NFA}$ is a decidable language.

Proof

Consider

$N =$ “On a $<$NFA, string$>$ input $\langle B, w \rangle$:

1. Convert NFA $B$ to an equivalent DFA $C$.
2. Simulate $C$ on input $w$ using TM $M$ (See previous proof).
3. If $C$ accepts - accept. Otherwise - reject.”
**Implementation**

Item 2 $N$’s high level description says: “Simulate $C$ on input $w$ using TM $M$. ”

Here $M$ is used by $N$ as a **procedure**. This can be done as follows:

1. $N$ is equipped with an additional tape on which $M$’s input will be written.

2. At the point in which $M$ is called, a section in which $M$’s input is written on the additional tape is added to $N$.

3. Following this section we add to $N$ a complete copy of $M$, using its input tape.

4. This procedure is repeated on each call to $M$.

---

**Regular Expressions are Decidable**

An additional way to describe Regular Languages is by use of regular expressions. Now we consider the language

$$A_{REX} = \{(R,w) \mid R \text{ is an RE generating } w\}$$

and prove:

**Theorem**

$A_{REX}$ is a decidable language.

---

**Proof**

Consider

$P$=“On a <RE, string> input $(R,w)$:

1. Convert RE $R$ to an equivalent NFA $A$.

2. Run TM $N$ (See previous proof) on $w$.

3. If $N$ accepts - *accept*. Otherwise – *reject.*”
The Emptiness Problem for DFA-s

In this problem it is required to compute whether a given DFA accepts at least one string: Consider the language
\[ E_{DFA} = \{ A \mid A \text{ is a DFA and } L(A) = \emptyset \} \]
and prove:
**Theorem**
\[ E_{DFA} \] is a decidable language.

Proof
Consider
\[ T = \text{“On a DFA input } <A> : \]
1. Mark the start state of \( A \).
2. Repeat until no new states are marked:
   3. Mark any state that has an incoming transition from an already marked state.
   4. If no accept state is marked - accept. Otherwise – reject.”

DFA Equivalence is Decidable

Our survey of decidability problems for regular languages is completed by considering
\[ EQ_{DFA} = \{ A, B \mid A \text{ and } B \text{ are DFA-s and } L(A) = L(B) \} \]
and proving:
**Theorem**
\[ EQ_{DFA} \] is a decidable language.

Proof
In order to prove this theorem we use the TM \( T \) of the previous proof. The input for \( T \) is a DFA \( C \) satisfying:
\[ L(C) = \left( L(A) \cap \overline{L(B)} \right) \cup \left( \overline{L(A)} \cap L(B) \right) \]
This expression is called The symmetric difference of \( L(A) \) and \( L(B) \) and it can be proved that \( L(C) = \emptyset \) iff \( L(A) = L(B) \).
**Proof**

DFA $C$ is constructed using the algorithms for constructing **Union**, **Intersection**, and **Complementation**, of Chapter 1.

The TM machine $F$ for Deciding $EQ_{DFA}$ gets as input to DFA-s $A$ and $B$. This machine first activates the algorithms of Chapter 1 to construct DFA $C$ and then it calls TM $T$ to check whether $L(C) = \phi$. If $T$ accepts so does $F$. Otherwise $F$ rejects.

---

**Decidable CFG Related Languages**

Now we consider the language

$$A_{CFG} = \{ (G, w) \mid G \text{ is a CFG generating } w \}$$

and prove:

**Theorem**

$A_{CFG}$ is a decidable language.

If we try to prove this theorem by checking all possible derivations of $G$, we may run into trouble on grammars containing **cycles**, e.g.

$$A \rightarrow B$$
$$B \rightarrow A$$

This grammar has an infinite derivation which is hard to deal with.
Decidable CFG Related Languages

The way to solve this problem is by using the following definition and theorem:

**Definition**
A context-free grammar is in **Chomsky normal form** (CNF) if every rule is of the form:

\[ A \rightarrow BC \]
\[ A \rightarrow a \]

**Theorem**
Every context-free grammar has an equivalent grammar in Chomsky normal form.

**Note:** In a CNF grammar every non-final derivation extends the current string.

**Proof of Theorem**
Consider

\( S = \text{“On <CFG, string> input } \langle G, w \rangle : \)

1. Convert \( G \) to an equivalent CNF grammar.
2. List all derivations with \( 2n-1 \) productions.
3. If \( w \) is generated by one of these derivations - accept. Otherwise – reject.”
Consider the language
\[ E_{CFG} = \{ G \mid G \text{ is a CFG and } L(G) = \phi \} \]

**Theorem**

\( E_{DFA} \) is a decidable language.

How can this be proved?

---

Consider the language
\[ E_{CFG} = \{ G \mid G \text{ is a CFG and } L(G) = \phi \} \]

**Theorem**

\( E_{CFG} \) is a decidable language.

---

Consider the language
\[ E_{CFG} = \{ G, H \mid G \text{ and } H \text{ are CFG-s and } L(G) = L(H) \} \]

Unlike DFA-s CFL-s are not closed under intersection and complementation. Therefore we cannot decide \( E_{CFG} \) using the method we used for \( E_{CFG} \). In the near future we will prove:

**Theorem**

\( E_{CFG} \) is an undecidable language.

---

\( M = \{ \text{On input } \langle G \rangle, \text{ where } G \text{ is a CFG:} \}

1. Mark all terminal symbols in \( G \).
2. Repeat until no new variables are marked:
   3. Mark any variable \( A \) where \( G \) has a rule of the form
      \[ A \rightarrow U_1 U_2 \cdots U_k \]
      and \( U_1, U_2, \ldots, U_k \) are all marked.
   4. If the start variable is not marked accept
      otherwise reject.”

---

Our survey of decidability problems for CFL-s/CFG-s is completed by considering the decision problem for context Free Languages, namely: Given a context free language \( L \), does there exist a TM that decides \( L \)?
CFL-s are Decidable

Recall that every CFL is recognized by some PDA. It is not hard to prove that a TM can simulate a PDA, but this is not enough for many CFL-s the PDA recognizing them is nondeterministic, which may cause the following problem:

Let \( L \) be a CFL, let \( p \) be a PDA recognizing \( L \) and let \( w \) be a string such that \( w \notin L \). PDA \( P \) may never stop on \( w \) and a TM simulating \( P \) may loop while a decider should stop on every input. Nevertheless there is a solution:

Theorem

Let \( L \) be a CFL. There exists a TM deciding \( L \).

Proof

Since \( L \) is a CFL, there exist a CFG \( G \) that generates \( L \). The problem is solved by TM \( M_G \) that contains a copy of \( G \), as follows.

1. RUN TM \( S \) on input \( \langle G,w \rangle \).
2. If \( S \) accepts - accept. Otherwise - reject.