Chapter 4 Lecture Notes (Section 4.1: Decidable Languages)

David Martin
dm@cs.uml.edu

With modifications by Prof. Karen Daniels, Fall2012
Back to $\Sigma_1$

- The fact that $\Sigma_1$ is not closed under complement means that there exists some language $L$ that is not recognizable by any TM.

- By Church-Turing thesis this means that *no imaginable finite computer, even with infinite memory, could recognize this language $L!*
A non-$\Sigma_1$ language

Each point is a language in this Venn diagram
Strategy

- **Goal**: Explore limits of algorithmic solvability.
- We’ll show (later in Section 4.2) that there are more (a lot more) languages in ALL than there are in $\Sigma_1$
  - Namely, that $\Sigma_1$ is countable but ALL isn’t countable
  - Which implies that $\Sigma_1 \neq$ ALL
  - Which implies that there exists some $L$ that is not in $\Sigma_1$
Overview of Section 4.1

- **Decidable Languages** (in $\Sigma_0$): to foster later appreciation of undecidable languages
  - Regular Languages
    - $A_{DFA}$: Acceptance problem for DFAs
    - $A_{NFA}$: Acceptance problem for NFAs
    - $A_{REX}$: Acceptance problem for Regular Expressions
    - $E_{DFA}$: Emptiness testing for DFAs
    - $EQ_{DFA}$: 2 DFAs recognizing the same language
  - Context-Free Languages (see next slide)...
Overview of Section 4.1 (cont.)

- **Decidable Languages** *(in $\Sigma_0$)*: to foster later appreciation of undecidable languages

  - **Context-Free Languages**
    - $A_{\text{CFG}}$: Does a given CFG generate a given string?
    - $E_{\text{CFG}}$: Is the language of a given CFG empty?
    - Every CFL is decidable by a Turing machine.
Overview of Section 4.1

- *Decidable Languages* (*in* \( \Sigma_0 \)): to foster later appreciation of undecidable languages
- Regular Languages
  - \( A_{DFA} \): *Acceptance problem for DFAs*
  - Acceptance problem for NFAs
  - Acceptance problem for Regular Expressions
  - Emptiness testing for DFAs
  - 2 DFAs recognizing the same language
Decidable Problems for Regular Languages: DFAs

- **Acceptance problem for DFAs**
  
  \[ A_{\text{DFA}} = \{ < B, w > \mid B \text{ is a DFA that accepts a given string } w \} \]
  
  - Language includes encodings of all DFAs and strings they accept.
  - Showing language is decidable is same as showing the computational problem is decidable.

- **Theorem 4.1**: \( A_{\text{DFA}} \) is a decidable language.
  
  - **Proof Idea**: Specify a TM \( M \) that decides \( A_{\text{DFA}} \).
    
    \( M = \) “On input \( <B,w> \), where \( B \) is a DFA and \( w \) is a string (implicit legal encoding check too):
    
    1. Simulate \( B \) on input \( w \).
    2. If simulation ends in accept state, accept. If it ends in nonaccepting state, reject.”

  Implementation details??
Overview of Section 4.1

- *Decidable Languages* (in $\Sigma_0$): to foster later appreciation of undecidable languages
  - Regular Languages
    - Acceptance problem for DFAs
    - $A_{NFA}$: *Acceptance problem for NFAs*
    - Acceptance problem for Regular Expressions
    - Emptiness testing for DFAs
    - 2 DFAs recognizing the same language
Decidable Problems for Regular Languages: NFAs

- **Acceptance problem for NFAs**

  \[ A_{\text{NFA}} = \{ <B, w> | B \text{ is an NFA that accepts a given string } w \} \]

- **Theorem 4.2**: \( A_{\text{NFA}} \) is a decidable language.

  - **Proof Idea**: Specify a TM \( N \) that decides \( A_{\text{NFA}} \).

    - \( N \) = “On input \( <B, w> \), where \( B \) is an NFA and \( w \) is a string:
      1. Convert NFA \( B \) to equivalent DFA \( C \) using Theorem 1.39.
      2. Run TM \( M \) from Theorem 4.1 on input \( <C, w> \).
      3. If \( M \) accepts, accept. Otherwise, reject.”

  *\( N \) uses \( M \) as a “ subroutine.”*

Alternatively, could we have modified proof of Theorem 4.1 to accommodate NFAs?
Overview of Section 4.1

- Decidable Languages (in $\Sigma_0$): to foster later appreciation of undecidable languages
  - Regular Languages
    - Acceptance problem for DFAs
    - Acceptance problem for NFAs
    - $A_{\text{REX}}$: Acceptance problem for Regular Expressions
    - Emptiness testing for DFAs
    - 2 DFAs recognizing the same language
Decidable Problems for Regular Languages: Regular Expressions

- **Acceptance problem for Regular Expressions**
  
  $A_{REX} = \{ <R, w> | R \text{ is a regular expression that generates string } w \}$

- **Theorem 4.3**: $A_{REX}$ is a decidable language.
  
  **Proof Idea**: Specify a TM $P$ that decides $A_{REX}$.
  
  $P = “On input <R,w>, where $R$ is a regular expression and $w$ is a string:
  
  1. Convert regular expression $R$ to equivalent NFA $A$ using Theorem 1.54.
  2. Run TM $N$ from Theorem 4.2 on input $<A,w>$.
  3. If $N$ accepts, accept. If $N$ rejects, reject.”
Overview of Section 4.1

- **Decidable Languages (in $\Sigma_0$):** to foster later appreciation of undecidable languages
  - Regular Languages
    - Acceptance problem for DFAs
    - Acceptance problem for NFAs
    - Acceptance problem for Regular Expressions
    - $E_{DFA}$: **Emptiness testing for DFAs**
    - 2 DFAs recognizing the same language
Decidable Problems for Regular Languages: DFAs

- **Emptiness problem for DFAs**
  \[ E_{DFA} = \{ <A> | A \text{ is a DFA and } L(A) = \emptyset \} \]

- **Theorem 4.4**: \( E_{DFA} \) is a decidable language.
  
  **Proof Idea**: Specify a TM \( T \) that decides \( E_{DFA} \).
  
  \( T = \) “On input \( <A> \), where \( A \) is a DFA:
  
  1. Mark start state of \( A \).
  2. Repeat until no new states are marked:
  3. Mark any state that has a transition coming into it from any state that is already marked.
  4. If no accept state is marked, *accept*; otherwise, *reject*.”

---

**Example (board work)**
Overview of Section 4.1

- Decidable Languages (in $\Sigma_0$): to foster later appreciation of undecidable languages
  - Regular Languages
    - Acceptance problem for DFAs
    - Acceptance problem for NFAs
    - Acceptance problem for Regular Expressions
    - Emptiness testing for DFAs
    - $\mathbf{EQ}_{\text{DFA}}$: 2 DFAs recognizing the same language
Decidable Problems for Regular Languages: DFAs

- 2 DFAs recognizing the same language
  \[ EQ_{DFA} = \{ <A, B> | A \text{ and } B \text{ are DFAs and } L(A) = L(B) \} \]

- Theorem 4.5: \( EQ_{DFA} \) is a decidable language.

  symmetric difference:
  \[
  L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))
  \]

  Recall regular languages are closed under complement, intersection, union.

  emptiness:
  \[ L(C) = \emptyset \iff L(A) = L(B) \]

Source: Sipser Textbook

F = “On input \( \langle A, B \rangle \), where \( A \) and \( B \) are DFAs:

1. Construct DFA \( C \) as described.
2. Run TM \( T \) from Theorem 4.4 on input \( \langle C \rangle \).
3. If \( T \) accepts, accept. If \( T \) rejects, reject.”

Figure 4.6
The symmetric difference of \( L(A) \) and \( L(B) \)
Overview of Section 4.1

- Decidable Languages (in $\Sigma_0$): to foster later appreciation of undecidable languages
  - Context-Free Languages
    - $A_{\text{CFG}}$: Does a given CFG generate a given string?
    - Is the language of a given CFG empty?
    - Every CFL is decidable by a Turing machine.
Decidable Problems for Context-Free Languages: CFGs

- **Does a given CFG generate a given string?**
  \[ A_{\text{CFG}} = \{ < G, w > | G \text{ is a CFG that generates string } w \} \]

- **Theorem 4.7**: \( A_{\text{CFG}} \) is a decidable language.
  - Why is this unproductive: use \( G \) to go through all derivations to determine if any yields \( w \)?
  - Better Idea... **Proof Idea**: Specify a TM \( S \) that decides \( A_{\text{CFG}} \).
    - \( S = \) “On input \( < G, w > \), where \( G \) is a CFG and \( w \) is a string:
      1. Convert \( G \) to equivalent Chomsky normal form grammar.
      2. List all derivations with \( 2n-1 \) steps (**why?**), where \( n = \) length of \( w \). (Except if \( n=0 \), only list derivations with 1 step.)
      3. If any of these derivations yield \( w \), *accept*; otherwise, *reject*.”
Overview of Section 4.1

- **Decidable Languages (in \( \Sigma_0 \)):** To foster later appreciation of undecidable languages

  - **Context-Free Languages**
    - Does a given CFG generate a given string?
    - \( E_{\text{CFG}}: \text{Is the language of a given CFG empty?} \)
    - Every CFL is decidable by a Turing machine.
Decidable Problems for Context-Free Languages: CFGs

- **Is the language of a given CFG empty?**
  
  \[ E_{\text{CFG}} = \{ < G > | G \text{ is a CFG and } L(G) = \emptyset \} \]

- **Theorem 4.8**: \( E_{\text{CFG}} \) is a decidable language.

  **Proof Idea**: Specify a TM \( R \) that decides \( E_{\text{CFG}} \).

  \( R = \) “On input \( < G > \), where \( G \) is a CFG:

  1. Mark all terminal symbols in \( G \).
  2. Repeat until no new variables get marked:
     3. Mark any variable \( A \) where \( G \) has rule \( A \rightarrow U_1 U_2 \ldots U_k \)
        and each symbol \( U_1 U_2 \ldots U_k \) has already been marked.
  1. If start variable is not marked, *accept*; otherwise, *reject*."

---

20
Decidable (?) Problems for Context-Free Languages: CFGs

- **Check if 2 CFGs generate the same language.**

\[ EQ_{\text{CFG}} = \{ < G, H > | G \text{ and } H \text{ are CFGs and } L(G) = L(H) \} \]

- **Not decidable! (see Chapter 5)**

- Why is this possible? Why is this problem not in \( \Sigma_0 \) if CFL is in \( \Sigma_0 \)?
Recall: Closure properties of CFL

- Reminder: closure properties can help us measure whether a computation model is reasonable or not
- CFL is closed under
  - Union, concatenation
  - Thus, exponentiation and *
- CFL is not closed under
  - Intersection
  - Complement
- Weak intersection:

If $A \in \text{CFL}$ and $R \in \text{REG}$, then $A \cap R \in \text{CFL}$
Overview of Section 4.1

- **Decidable Languages (in \( \Sigma_0 \))**: to foster later appreciation of undecidable languages
  - Context-Free Languages
    - Does a given CFG generate a given string?
    - Is the language of a given CFG empty?
    - **Every CFL is decidable by a Turing machine.**
Decidable Problems for Context-Free Languages: CFLs

- Every CFL is decidable by a Turing machine.
- Bad Idea: Convert PDA for CFL into TM
- Theorem 4.9: Every context-free language is decidable.

- Let $A$ be a CFL and $G$ be a CFG for $A$. (Where does $G$ come from?)
- Design TM $M_G$ that decides $A$.
- $M_G$ = “On input $w$, where $w$ is a string:
  1. Run TM $S$ from Theorem 4.7 on input $<G,w>$.
  2. If $S$ accepts, accept. If $S$ rejects, reject.”
Summary: Some problems (languages) related to languages in $\Sigma_0$ have been shown in this lecture to be in $\Sigma_0$.  

Remember that just because a language is in $\Sigma_0$ does not mean that every problem (language) related to members of its class is also in $\Sigma_0$!