Undecidable Problems for CFGs

CSCI 3130 Formal Languages and Automata Theory

Siu On CHAN

Fall 2020

Chinese University of Hong Kong

Decidable vs undecidable

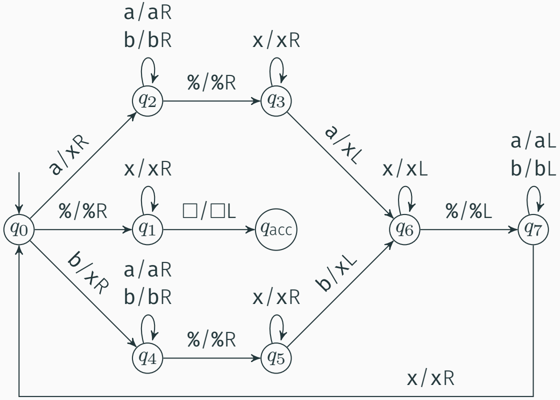
Decidable	Undecidable
DFA D accepts w	TM M accepts w
CFG G generates w	TM M halts on w
DFAs D and D' accept the same inputs	TM M accepts some input
	TM M and M' accept the same inputs

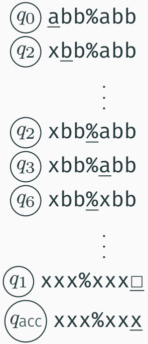
CFG G generates all inputs?

CFG G is ambiguous?

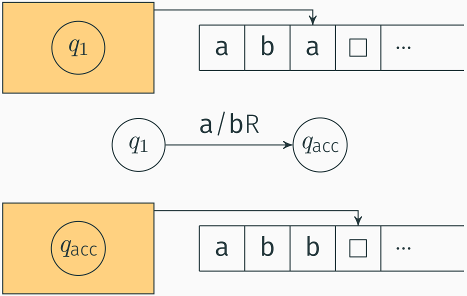
Representing computation

$L_1 = \{w\%w \mid w \in \{\mathsf{a}, \mathsf{b}\}^*\}$





A configuration consists of current state, head position, and tape contents

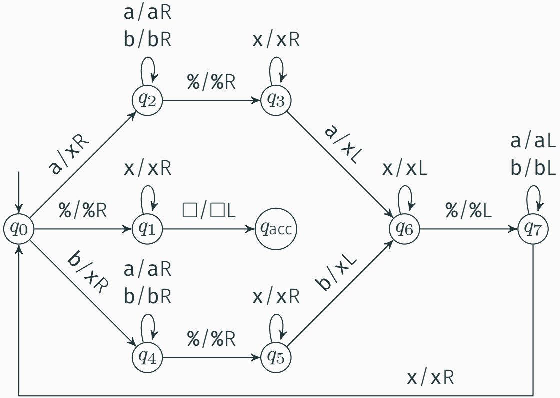


Configuration (abbreviation)

ab q_1 a

abb q_{acc}

Computation history



q_0 abb%abb
x q_2 bb%abb
⋮
xbb q_2 %abb
xbb% q_3 abb
xbb q_2 %xbb
⋮
xxx%xxx q_1
xxx%xx q_{acc} x

computation history

Computation histories as strings

If M halts on w, the computation history of (M, w) is the sequence of configurations $C_1, \ldots, C_k$ that M goes through on input w

q_0 ab%ab
x q_2 b%ab
⋮
xx%xx q_1
xx%x q_{acc} x

$$\#\underbrace{q_0\,\text{ab\%ab}}_{C_1}\#\underbrace{\text{x}\,q_1\,\text{b\%ab}}_{C_2}\#\ldots\#\underbrace{\text{xx\%x}\,q_{\text{acc}}\,\text{x}}_{C_k}\#$$

The computation history can be written as a string h over alphabet $\Gamma \cup Q \cup \{\#\}$

accepting history: M accepts w $\Leftrightarrow$ q_{acc} appears in h

rejecting history: M rejects w $\Leftrightarrow$ q_{rej} appears in h

Undecidable problems for CFGs

$\text{ALL}_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG that generates all strings}\}$

The language ALL_{CFG} is undecidable

We will argue that

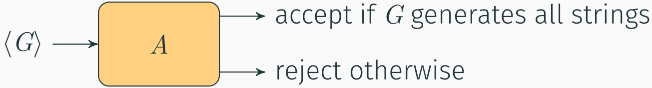
If ALL_{CFG} can be decided, so can $\overline{A_{\text{TM}}}$

$\overline{A_{\text{TM}}} = \{\langle M, w \rangle \mid M \text{ is a TM that rejects or loops on } w\}$

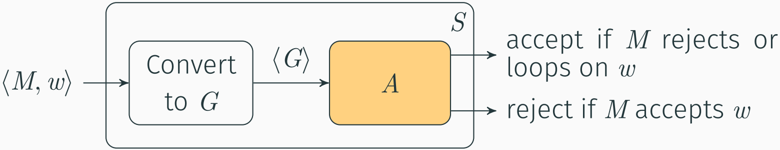
Undecidable problems for CFGs

Proof by contradiction

Suppose some Turing machine A decides ALL_{CFG}



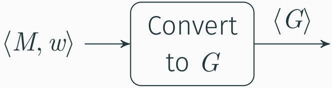
We want to construct a Turing machine S that decides $\overline{A_{\text{TM}}}$



G generates all strings if M rejects or loops on w

G fails to generate some string if M accepts w

Undecidable problems for CFGs



G fails to generate some string

$$\Updownarrow$$

M accepts w

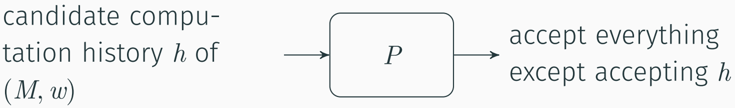
The alphabet of G will be $\Gamma \cup Q \cup \{\#\}$

G will generate all strings except

accepting computation history of (M, w)

First we construct a PDA P, then convert it to CFG G

Undecidablility via computation histories



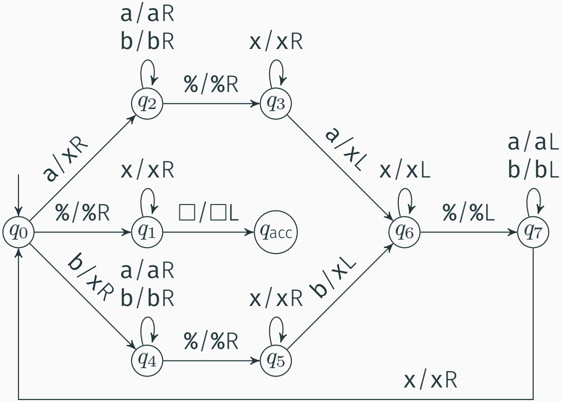
$$\texttt{\#}q_0\texttt{ab\%ab\#x}q_1\texttt{b\%ab\#...\#xx\%x}q_{\texttt{acc}}\texttt{x\#} \quad \Rightarrow \quad \text{Reject}$$

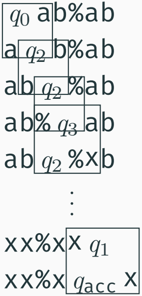
$P =$ on input h (try to spot a mistake in h)

- If h is not of the form $\#w_1\#w_2\#...\#w_k\#$, accept
- If $w_1 \neq q_0 w$ or w_k does not contain q_{acc}, accept
- If two consecutive blocks $w_i\#w_{i+1}$ do not follow from the transitions of M, accept

Otherwise, h must be an accepting history, reject

Computation is local

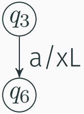




Changes between configurations always occur around the head

Legal and illegal transitions windows

legal windows	illegal windows
… abx … … abx …	… $q_3$ab … … abq_3 …
… aq_3a … … $q_6$ax …	… $q_3 q_3$a … … $q_3 q_3$x …
… aba … … abq_6 …	… aq_3a … … $q_6$ab …
… aa□ … … xa□ …	… aq_3a … … aq_6x …



Implementing P

If two consecutive blocks w_i#w_{i+1} do not follow from the transitions of M, accept

#xb%$q_3$ab
#xbq_5%xb

For every position of w_i:

- Remember offset from # in w_i on stack
- Remember first row of window in state

After reaching the next #:

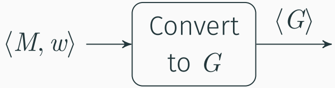
- Pop offset from # from stack as you consume input
- Remember second row of window in state

If window is illegal, accept; Otherwise reject

The computation history method

$\text{ALL}_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG that generates all strings}\}$

If ALL_{CFG} can be decided, so can $\overline{A_{\text{TM}}}$



G accepts all strings except accepting computation history of (M, w)

We first construct a PDA P, then convert it to CFG G

Post Correspondence Problem

Input: A fixed set of tiles, each containing a pair of strings

bab	c	a	baa	a	bab
cc	ab	ab	a	baba	ε

Given an infinite supply of tiles from a particular set, can you match top and bottom?

a	baa	bab	c	c	bab	a
ab	a	ε	ab	ab	cc	baba

Top and bottom are both abaababccbaba

Undecidability of PCP

$$\text{PCP} = \{\langle C \rangle \mid C \text{ is a collection of tiles that contains a top-bottom match}\}$$

Next lecture we will show (using computation history method)

The language PCP is undecidable

Ambiguity of CFGs

$$\text{AMB} = \{\langle G\rangle \mid G \text{ is an ambiguous CFG}\}$$

The language AMB is undecidable

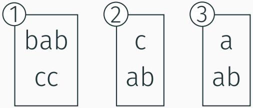
We will argue that

If AMB can be decided, then so can PCP

C (collection of tiles) $\longmapsto$ G (CFG)

If C can be matched, then G is ambiguous
If C cannot be matched, then G is unambiguous

First, let's number the tiles



Encoding the tiles in a grammar

C (collection of tiles) $\longmapsto$ G (CFG)

Example:

①	②	③
bab	c	a
cc	ab	ab

Terminals: a, b, c, 1, 2, 3

Variables: S, T, B

Productions:

$$S \to T \mid B$$

$$\begin{array}{lll} T \to \text{bab}\,T1 & T \to \text{c}\,T2 & T \to \text{a}\,T3 \\ B \to \text{cc}\,B1 & B \to \text{ab}\,B2 & B \to \text{ab}\,B3 \\ T \to \text{bab1} & T \to \text{c2} & T \to \text{a3} \\ B \to \text{cc1} & B \to \text{ab2} & B \to \text{ab3} \end{array}$$

In general:

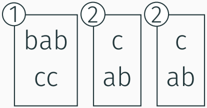
(n)
α
β

yields

$$\begin{array}{ll} T \to \alpha\, T n & T \to \alpha n \\ B \to \beta\, B n & B \to \beta n \end{array}$$

Matching sequence implies ambiguous grammar

Each sequence of tiles gives a pair of derivations



$$S \Rightarrow T \Rightarrow \text{bab}\,T1 \Rightarrow \text{babc}\,T21 \Rightarrow \text{babcc221}$$

$$S \Rightarrow B \Rightarrow \text{cc}B1 \Rightarrow \text{ccab}B21 \Rightarrow \text{ccabab221}$$

If the tiles match, these two derive the same string

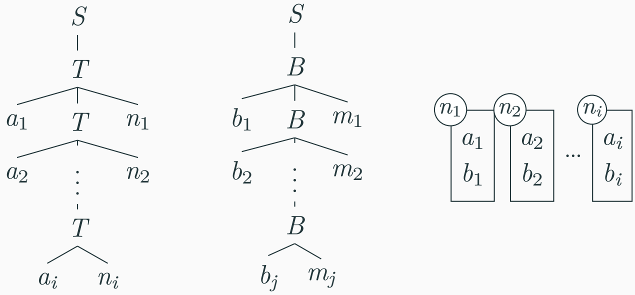
(with different parse trees)

Ambiguous grammar implies matching sequence

C (collection of tiles) $\longmapsto$ G (CFG)

If T can be matched, then G is ambiguous ✓
If T cannot be matched, then G is unambiguous ✓

If G is ambiguous, then the two parse trees look like



Therefore $n_1 n_2 \dots n_i = m_1 m_2 \dots m_j$, and there is a match