Languages and Machines (Module 7 TCS+IAM)

L&M 8: Turing Machines and Unrestricted Grammars

(Undecidability of the Halting Problem)

Ch8:1-7; Ch11:1; Ch11:4-5; Ch12:1; Ch12:4

Alexander Skopalik

Discrete Mathematics and Mathematical Programming, Applied Mathematics, University of Twente

Lecture 8

Recall Turing Machines

Turing Machines:

- Finite Automaton + Unbounded Tape
  - Read/Modify the tape and move to the Left/Right
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

Recall Turing Machines

Turing Machines:

- Finite Automaton + Unbounded Tape
  - Read/Modify the tape and move to the Left/Right
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- Equivalent Alternatives
  - Standard Turing Machine (1 tape, deterministic)
  - 2-sided tape, multiple tracks, multiple tapes
  - Non-deterministic Turing Machine (backtracking)

Recall Turing Machines

Turing Machines:

- Finite Automaton + Unbounded Tape
  - Read/Modify the tape and move to the Left/Right
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- Equivalent Alternatives
  - Standard Turing Machine (1 tape, deterministic)
  - 2-sided tape, multiple tracks, multiple tapes
  - Non-deterministic Turing Machine (backtracking)
- Language Acceptance by TM
  - TM accepts w if TM halts in a final state
  - L is recursively enumerable: L is recognised by some TM, which then accepts precisely all words $w \in L$.
  - L is recursive: L is decided by some TM, which accepts all words $w \in L$, and moreover it *halts on all inputs in* Σ^* and

Contents

Contents

Unrestricted grammars

Left side contains (besides a variable) *context information*

Grammar:

$$
\begin{aligned}
S &\rightarrow aAbc \mid \lambda \\
A &\rightarrow aAbC \mid \lambda \\
Cb &\rightarrow bC \\
Cc &\rightarrow cc
\end{aligned}
$$

Unrestricted grammars

Left side contains (besides a variable) *context information*

Grammar:

$$
\begin{aligned}
S &\rightarrow aAbc \mid \lambda \\
A &\rightarrow aAbC \mid \lambda \\
Cb &\rightarrow bC \\
Cc &\rightarrow cc
\end{aligned}
$$

Example derivation:

$$
\begin{aligned}
S &\Rightarrow aAbc \Rightarrow aaAbCbc \\
&\Rightarrow aabCbc \Rightarrow aabbCc \\
&\Rightarrow aabbcc
\end{aligned}
$$

Unrestricted grammars

Left side contains (besides a variable) *context information*

Grammar:

$$\begin{aligned} S &\rightarrow aAbc \mid \lambda \\ A &\rightarrow aAbC \mid \lambda \\ Cb &\rightarrow bC \\ Cc &\rightarrow cc \end{aligned}$$

Example derivation:

$$\begin{aligned} S &\Rightarrow aAbc \Rightarrow aaAbCbc \\ &\Rightarrow aabCbc \Rightarrow aabbCc \\ &\Rightarrow aabbcc \end{aligned}$$

Language: $\{a^i b^i c^i \mid i \geq 0\}$ (not context-free!)

Unrestricted grammars

Left side contains (besides a variable) *context information*

Grammar:

$$\begin{aligned} S &\to aAbc \mid \lambda \\ A &\to aAbC \mid \lambda \\ Cb &\to bC \\ Cc &\to cc \end{aligned}$$

Example derivation:

$$\begin{aligned} S &\Rightarrow aAbc \Rightarrow aaAbCbc \\ &\Rightarrow aabCbc \Rightarrow aabbCc \\ &\Rightarrow aabbcc \end{aligned}$$

Language: $\{a^i b^i c^i \mid i \geq 0\}$ (not context-free!)

Definition

An *unrestricted grammar* is a tuple $\langle V, \Sigma, P, S \rangle$ with

- V a set of variables
- Σ the alphabet
- $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ a set of rules
- $S \in V$ the start symbol.

Example: unrestricted grammar for $w[w]$

The following grammar generates the language $\{w[w] \mid w \in \{a,b\}^*\}$:

$$S \to aT[a] \mid bT[b] \mid []$$
$$T[\to aT[A \mid bT[B \mid [$$

$$Ax \to xA \qquad (x \in \{a,b\})$$
$$Bx \to xB \qquad (x \in \{a,b\})$$
$$A] \to a]$$
$$B] \to b]$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language
$\{w[w] \mid w \in \{a,b\}^*\}$:

$$
\begin{aligned}
S &\to aT[a] \mid bT[b] \mid [] \\
T[&\to aT[A \mid bT[B \mid [
\end{aligned}
\qquad
\begin{aligned}
Ax &\to xA \quad (x \in \{a,b\}) \\
Bx &\to xB \quad (x \in \{a,b\}) \\
A] &\to a] \\
B] &\to b]
\end{aligned}
$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language $\{w[w] \mid w \in \{a,b\}^*\}$:

$$S \to aT[a] \mid bT[b] \mid []$$
$$T[\to aT[A \mid bT[B \mid [$$

$$Ax \to xA \qquad (x \in \{a,b\})$$
$$Bx \to xB \qquad (x \in \{a,b\})$$
$$A] \to a]$$
$$B] \to b]$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$S$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language $\{w[w] \mid w \in \{a,b\}^*\}$:

$$
\begin{array}{rcl}
S & \rightarrow & aT[a] \mid bT[b] \mid [] \\
T[& \rightarrow & aT[A \mid bT[B \mid [
\end{array}
\qquad
\begin{array}{rcll}
Ax & \rightarrow & xA & (x \in \{a,b\}) \\
Bx & \rightarrow & xB & (x \in \{a,b\}) \\
A] & \rightarrow & a] & \\
B] & \rightarrow & b] &
\end{array}
$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$S \Rightarrow aT[a]$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language $\{w[w] \mid w \in \{a,b\}^*\}$:

$$S \to aT[a] \mid bT[b] \mid []$$
$$T[\to aT[A \mid bT[B \mid [$$

$$Ax \to xA \qquad (x \in \{a,b\})$$
$$Bx \to xB \qquad (x \in \{a,b\})$$
$$A] \to a]$$
$$B] \to b]$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$S \Rightarrow aT[a]$$
$$\Rightarrow aaT[Aa]$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language
$\{w[w] \mid w \in \{a,b\}^*\}$:

$$\begin{array}{rcl} S & \to & aT[a] \mid bT[b] \mid [] \\ T[& \to & aT[A \mid bT[B \mid [\end{array} \qquad \begin{array}{rcll} Ax & \to & xA & (x \in \{a,b\}) \\ Bx & \to & xB & (x \in \{a,b\}) \\ A] & \to & a] & \\ B] & \to & b] & \end{array}$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$\begin{array}{rl} S & \Rightarrow aT[a] \\ & \Rightarrow aaT[Aa] \\ & \Rightarrow aabT[BAa] \end{array}$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language $\{w[w] \mid w \in \{a,b\}^*\}$:

$$
\begin{aligned}
S &\rightarrow aT[a] \mid bT[b] \mid [] & Ax &\rightarrow xA \quad (x \in \{a,b\}) \\
T[&\rightarrow aT[A \mid bT[B \mid [& Bx &\rightarrow xB \quad (x \in \{a,b\}) \\
& & A] &\rightarrow a] \\
& & B] &\rightarrow b]
\end{aligned}
$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$
\begin{aligned}
S &\Rightarrow aT[a] \\
&\Rightarrow aaT[Aa] \\
&\Rightarrow aabT[BAa] \\
&\Rightarrow aab[BAa]
\end{aligned}
$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language
$\{w[w] \mid w \in \{a,b\}^*\}$:

$$S \to aT[a] \mid bT[b] \mid []$$
$$T[\to aT[A \mid bT[B \mid [$$

$$Ax \to xA \qquad (x \in \{a,b\})$$
$$Bx \to xB \qquad (x \in \{a,b\})$$
$$A] \to a]$$
$$B] \to b]$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$\begin{aligned} S &\Rightarrow aT[a] \\ &\Rightarrow aaT[Aa] \\ &\Rightarrow aabT[BAa] \\ &\Rightarrow aab[BAa] \end{aligned} \qquad aab[BAa] \Rightarrow aab[BaA]$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language
$\{w[w] \mid w \in \{a,b\}^*\}$:

$$S \to aT[a] \mid bT[b] \mid []$$
$$T[\to aT[A \mid bT[B \mid [$$
$$Ax \to xA \qquad (x \in \{a,b\})$$
$$Bx \to xB \qquad (x \in \{a,b\})$$
$$A] \to a]$$
$$B] \to b]$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$\begin{aligned} S &\Rightarrow aT[a] \\ &\Rightarrow aaT[Aa] \\ &\Rightarrow aabT[BAa] \\ &\Rightarrow aab[BAa] \end{aligned} \qquad \begin{aligned} aab[BAa] &\Rightarrow aab[BaA] \\ &\Rightarrow aab[aBA] \end{aligned}$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language
$\{w[w] \mid w \in \{a,b\}^*\}$:

$$S \to aT[a] \mid bT[b] \mid []$$
$$T[\to aT[A \mid bT[B \mid [$$

$$Ax \to xA \quad (x \in \{a,b\})$$
$$Bx \to xB \quad (x \in \{a,b\})$$
$$A] \to a]$$
$$B] \to b]$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$S \Rightarrow aT[a] \Rightarrow aaT[Aa] \Rightarrow aabT[BAa] \Rightarrow aab[BAa]$$
$$aab[BAa] \Rightarrow aab[BaA] \Rightarrow aab[aBA] \Rightarrow aab[aBa]$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language
$\{w[w] \mid w \in \{a,b\}^*\}$:

$$
\begin{array}{rcl}
S & \to & aT[a] \mid bT[b] \mid [] \\
T[& \to & aT[A \mid bT[B \mid [
\end{array}
\qquad
\begin{array}{rcll}
Ax & \to & xA & (x \in \{a,b\}) \\
Bx & \to & xB & (x \in \{a,b\}) \\
A] & \to & a] & \\
B] & \to & b] &
\end{array}
$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$
\begin{array}{rl}
S & \Rightarrow aT[a] \\
 & \Rightarrow aaT[Aa] \\
 & \Rightarrow aabT[BAa] \\
 & \Rightarrow aab[BAa]
\end{array}
\qquad
\begin{array}{rl}
aab[BAa] & \Rightarrow aab[BaA] \\
 & \Rightarrow aab[aBA] \\
 & \Rightarrow aab[aBa] \\
 & \Rightarrow aab[aaB]
\end{array}
$$

Example: unrestricted grammar for $w[w]$

The following grammar generates the language $\{w[w] \mid w \in \{a,b\}^*\}$:

$$
\begin{aligned}
S &\to aT[a] \mid bT[b] \mid [] \\
T[&\to aT[A \mid bT[B \mid [
\end{aligned}
\qquad
\begin{aligned}
Ax &\to xA & (x \in \{a,b\}) \\
Bx &\to xB & (x \in \{a,b\}) \\
A] &\to a] \\
B] &\to b]
\end{aligned}
$$

Functioning (W represents w in uppercase):

- S and T generate $\{xw[W^R x] \mid x \in \{a,b\}, w \in \{a,b\}^*\}$
- A and B then propagate to the right; thus $[W^R x]$ turns into $[xw]$

Example: derivation of $aab[aab]$

$$
\begin{aligned}
S &\Rightarrow aT[a] \\
&\Rightarrow aaT[Aa] \\
&\Rightarrow aabT[BAa] \\
&\Rightarrow aab[BAa]
\end{aligned}
\qquad
\begin{aligned}
aab[BAa] &\Rightarrow aab[BaA] \\
&\Rightarrow aab[aBA] \\
&\Rightarrow aab[aBa] \\
&\Rightarrow aab[aaB] \\
&\Rightarrow aab[aab]
\end{aligned}
$$

Relation unrestricted grammars $\leftrightarrow$ Turing machines

Theorem (Theorem 10.1.2 and 10.1.3)

L recursively enumerable $\Leftrightarrow$ $\exists$ unrestricted grammar G with $L = L(G)$

Relation unrestricted grammars ↔ Turing machines

Theorem (Theorem 10.1.2 and 10.1.3)

L recursively enumerable $\Leftrightarrow \exists$ unrestricted grammar G with $L{=}L(G)$

"If" ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Relation unrestricted grammars ↔ Turing machines

Theorem (Theorem 10.1.2 and 10.1.3)

L recursively enumerable $\Leftrightarrow \exists$ unrestricted grammar G with $L{=}L(G)$

"If" ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

"Only If" ($\Rightarrow$)

Given a Turing machine M, construct an unrestricted grammar G_M such that $L(G_M) = L(M)$.

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

1. Write the input on Tape 1

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

1. Write the input on Tape 1
2. Encode the rules of G on Tape 2; for example

   $S \to Abc \mid \lambda \qquad A \to aAbC \mid \lambda \qquad Cb \to bC \qquad Cc \to cc$

   is encoded as

   $BS\#Abc\#\#S\#\#\#A\#aAbC\#\#A\#\#\#Cb\#bC\#\#Cc\#ccB$

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

1. Write the input on Tape 1
2. Encode the rules of G on Tape 2; for example
   $$S \to Abc \mid \lambda \qquad A \to aAbC \mid \lambda \qquad Cb \to bC \qquad Cc \to cc$$
   is encoded as
   $$BS\#Abc\#\#S\#\#\#A\#aAbC\#\#A\#\#\#Cb\#bC\#\#Cc\#ccB$$
3. Write the start symbol S on Tape 3

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

1. Write the input on Tape 1
2. Encode the rules of G on Tape 2; for example
   $$S \to Abc \mid \lambda \qquad A \to aAbC \mid \lambda \qquad Cb \to bC \qquad Cc \to cc$$
   is encoded as
   $$BS\#Abc\#\#S\#\#\#A\#aAbC\#\#A\#\#\#Cb\#bC\#\#Cc\#ccB$$
3. Write the start symbol S on Tape 3

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

1. Write the input on Tape 1
2. Encode the rules of G on Tape 2; for example
   $S \to Abc \mid \lambda \qquad A \to aAbC \mid \lambda \qquad Cb \to bC \qquad Cc \to cc$
   is encoded as
   $BS\#Abc\#\#S\#\#\#A\#aAbC\#\#A\#\#\#Cb\#bC\#\#Cc\#ccB$
3. Write the start symbol S on Tape 3

Iteratively (nondeterministically) apply rules $u \to v$ of Tape 2 on a (nondeterministic) partial word u on Tape 3

From unrestricted grammars to Turing machines

“If” ($\Leftarrow$)

Construct a Turing machine M_G such that $L(M_G) = L(G)$

Construction-idea: Use a 3-tape machine:

1. Write the input on Tape 1
2. Encode the rules of G on Tape 2; for example
   $$S \to Abc \mid \lambda \qquad A \to aAbC \mid \lambda \qquad Cb \to bC \qquad Cc \to cc$$
   is encoded as
   $BS\#Abc\#\#S\#\#\#A\#aAbC\#\#A\#\#\#Cb\#bC\#\#Cc\#ccB$
3. Write the start symbol S on Tape 3

Iteratively (nondeterministically) apply rules $u \to v$ of Tape 2 on a (nondeterministic) partial word u on Tape 3

Terminate if Tapes 1 and 3 are equal

From Turing machines to unrestricted grammars

“Only If” ($\Rightarrow$)

Given a Turing machine M, construct an unrestricted grammar G_M such that $L(G_M) = L(M)$.

From Turing machines to unrestricted grammars

“Only If” ($\Rightarrow$)

Given a Turing machine M, construct an unrestricted grammar G_M such that $L(G_M) = L(M)$.

Idea of the functioning of G_M in three phases:

- Rules to generate any word $w[q_0 Bw]$ from S
  - The first part w is the word that will be recognized
  - The last part $[q_0 Bw]$ represents the configuration of TM M

From Turing machines to unrestricted grammars

"Only If" ($\Rightarrow$)

Given a Turing machine M, construct an unrestricted grammar G_M such that $L(G_M) = L(M)$.

Idea of the functioning of G_M in three phases:

- Rules to generate any word $w[q_0Bw]$ from S
  - The first part w is the word that will be recognized
  - The last part $[q_0Bw]$ represents the configuration of TM M
- Give grammar rules that simulate the functioning of M, for example:
  - For all $q_i \xrightarrow{x/z\ R} q_j$ and all $y \in \Gamma$, we include a rule $q_ixy \to zq_jy$ in the grammar

From Turing machines to unrestricted grammars

"Only If" ($\Rightarrow$)

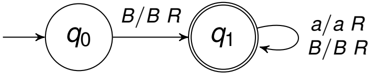
Given a Turing machine M, construct an unrestricted grammar G_M such that $L(G_M) = L(M)$.

Idea of the functioning of G_M in three phases:

- Rules to generate any word $w[q_0 Bw]$ from S
  - The first part w is the word that will be recognized
  - The last part $[q_0 Bw]$ represents the configuration of TM M
- Give grammar rules that simulate the functioning of M, for example:
  - For all $q_i \xrightarrow{x/z\ R} q_j$ and all $y \in \Gamma$, we include a rule $q_i xy \to z q_j y$ in the grammar
- Remove the final part $[\cdots]$ only if a final state is reached
  - This can be done with a number of simple "Erasure" rules

Example: Turing machine to unrestricted grammar

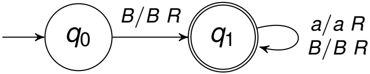
- Turing machine for the (regular) language $a^* b(a \cup b)^*$:



- Symbols of G_M:
  $\Sigma = \{a, b\}, V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^*b(a \cup b)^*$:



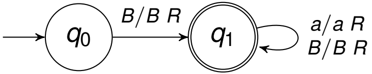
- Symbols of G_M:
  $\Sigma = \{a, b\}$, $V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

Phase 1 ($z \in \{a, b\}$)

$$S \to zT[z] \mid [q_0B]$$
$$T[\to aT[X \mid bT[Y \mid [q_0B$$
$$Xz \to zX$$
$$Yz \to zY$$
$$X] \to a]$$
$$Y] \to b]$$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^* b(a \cup b)^*$:



- Symbols of G_M:
$\Sigma = \{a, b\}$, $V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

Phase 1 ($z \in \{a, b\}$)

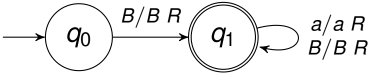
$$S \to zT[z] \mid [q_0 B]$$
$$T[\to aT[X \mid bT[Y \mid [q_0 B$$
$$Xz \to zX$$
$$Yz \to zY$$
$$X] \to a]$$
$$Y] \to b]$$

Phase 2 ($z \in \{a, b, B\}$)

$$q_0 B z \to B q_1 z$$
$$q_0 B] \to B q_1 B]$$
$$q_1 a z \to a q_1 z$$
$$q_1 a] \to a q_1 B]$$
$$q_1 B z \to B q_1 z$$
$$q_1 B] \to B q_1 B]$$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^*b(a \cup b)^*$:



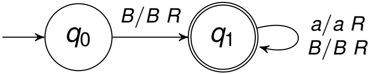
- Symbols of G_M:
  $\Sigma = \{a, b\}$, $V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

Phase 1 $(z \in \{a, b\})$	**Phase 2** $(z \in \{a, b, B\})$	**Phase 3**
$S \to zT[z] \mid [q_0B]$	$q_0Bz \to Bq_1z$	$q_1b \to E_R$
$T[\to aT[X \mid bT[Y \mid [q_0B$	$q_0B] \to Bq_1B]$	$E_Rz \to E_R$
$Xz \to zX$	$q_1az \to aq_1z$	$E_R] \to E_L$
$Yz \to zY$	$q_1a] \to aq_1B]$	$zE_L \to E_L$
$X] \to a]$	$q_1Bz \to Bq_1z$	$[E_L \to \lambda$
$Y] \to b]$	$q_1B] \to Bq_1B]$	

- Example: derivation of *aaba*

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^* b(a \cup b)^*$:



- Symbols of G_M:
  $\Sigma = \{a, b\}$, $V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

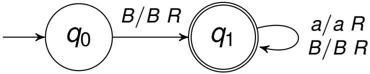
Phase 1 ($z \in \{a, b\}$)	**Phase 2** ($z \in \{a, b, B\}$)	**Phase 3**
$S \to zT[z] \mid [q_0 B]$	$q_0 Bz \to Bq_1 z$	$q_1 b \to E_R$
$T[\to aT[X \mid bT[Y \mid [q_0 B$	$q_0 B] \to Bq_1 B]$	$E_R z \to E_R$
$Xz \to zX$	$q_1 az \to aq_1 z$	$E_R] \to E_L$
$Yz \to zY$	$q_1 a] \to aq_1 B]$	$zE_L \to E_L$
$X] \to a]$	$q_1 Bz \to Bq_1 z$	$[E_L \to \lambda$
$Y] \to b]$	$q_1 B] \to Bq_1 B]$	

- Example: derivation of *aaba*

$$S \Rightarrow^* aaba[q_0 Baaba]$$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^* b(a \cup b)^*$:



- Symbols of G_M:
  $\Sigma = \{a, b\}, V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

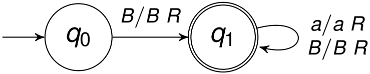
Phase 1 ($z \in \{a, b\}$)	**Phase 2** ($z \in \{a, b, B\}$)	**Phase 3**
$S \to zT[z] \mid [q_0 B]$	$q_0 Bz \to Bq_1 z$	$q_1 b \to E_R$
$T[\to aT[X \mid bT[Y \mid [q_0 B$	$q_0 B] \to Bq_1 B]$	$E_R z \to E_R$
$Xz \to zX$	$q_1 az \to aq_1 z$	$E_R] \to E_L$
$Yz \to zY$	$q_1 a] \to aq_1 B]$	$zE_L \to E_L$
$X] \to a]$	$q_1 Bz \to Bq_1 z$	$[E_L \to \lambda$
$Y] \to b]$	$q_1 B] \to Bq_1 B]$	

- Example: derivation of *aaba*

$$S \Rightarrow^* aaba[q_0 Baaba] \Rightarrow^* aaba[Baaq_1 ba]$$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^*b(a \cup b)^*$:



- Symbols of G_M:
  $\Sigma = \{a, b\}$, $V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

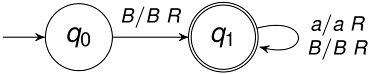
Phase 1 $(z \in \{a, b\})$	**Phase 2** $(z \in \{a, b, B\})$	**Phase 3**
$S \to zT[z] \mid [q_0 B]$	$q_0 Bz \to Bq_1 z$	$q_1 b \to E_R$
$T[\to aT[X \mid bT[Y \mid [q_0 B$	$q_0 B] \to Bq_1 B]$	$E_R z \to E_R$
$Xz \to zX$	$q_1 az \to aq_1 z$	$E_R] \to E_L$
$Yz \to zY$	$q_1 a] \to aq_1 B]$	$zE_L \to E_L$
$X] \to a]$	$q_1 Bz \to Bq_1 z$	$[E_L \to \lambda$
$Y] \to b]$	$q_1 B] \to Bq_1 B]$	

- Example: derivation of *aaba*

$$S \Rightarrow^* aaba[q_0 Baaba] \Rightarrow^* aaba[Baaq_1 ba]$$
$$\Rightarrow aaba[BaaE_R a]$$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^* b(a \cup b)^*$:



- Symbols of G_M:
  $\Sigma = \{a, b\}, V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

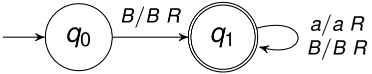
Phase 1 $(z \in \{a, b\})$	**Phase 2** $(z \in \{a, b, B\})$	**Phase 3**
$S \rightarrow zT[z] \mid [q_0 B]$	$q_0 Bz \rightarrow Bq_1 z$	$q_1 b \rightarrow E_R$
$T[\rightarrow aT[X \mid bT[Y \mid [q_0 B$	$q_0 B] \rightarrow Bq_1 B]$	$E_R z \rightarrow E_R$
$Xz \rightarrow zX$	$q_1 az \rightarrow aq_1 z$	$E_R] \rightarrow E_L$
$Yz \rightarrow zY$	$q_1 a] \rightarrow aq_1 B]$	$zE_L \rightarrow E_L$
$X] \rightarrow a]$	$q_1 Bz \rightarrow Bq_1 z$	$[E_L \rightarrow \lambda$
$Y] \rightarrow b]$	$q_1 B] \rightarrow Bq_1 B]$	

- Example: derivation of *aaba*

$$S \Rightarrow^* aaba[q_0 Baaba] \Rightarrow^* aaba[Baaq_1 ba]$$
$$\Rightarrow aaba[BaaE_R a] \Rightarrow aaba[BaaE_L$$

Example: Turing machine to unrestricted grammar

- Turing machine for the (regular) language $a^* b(a \cup b)^*$:



- Symbols of G_M:
  $\Sigma = \{a, b\}$, $V = \{S, T, E_R, E_L, [,], A, X, B, q_0, q_1\}$

Phase 1 $(z \in \{a, b\})$	**Phase 2** $(z \in \{a, b, B\})$	**Phase 3**
$S \to zT[z] \mid [q_0 B]$	$q_0 Bz \to Bq_1 z$	$q_1 b \to E_R$
$T[\to aT[X \mid bT[Y \mid [q_0 B$	$q_0 B] \to Bq_1 B]$	$E_R z \to E_R$
$Xz \to zX$	$q_1 az \to aq_1 z$	$E_R] \to E_L$
$Yz \to zY$	$q_1 a] \to aq_1 B]$	$zE_L \to E_L$
$X] \to a]$	$q_1 Bz \to Bq_1 z$	$[E_L \to \lambda$
$Y] \to b]$	$q_1 B] \to Bq_1 B]$	

- Example: derivation of *aaba*

$$S \Rightarrow^* aaba[q_0 Baaba] \Rightarrow^* aaba[Baaq_1 ba]$$
$$\Rightarrow aaba[BaaE_R a] \Rightarrow aaba[BaaE_L \Rightarrow^* aaba$$

Contents

The Chomsky Hierarchie

Classification of languages (sets of words over Σ^*):

Chomsky Hierarchy:

Type	Languages	Grammars	Machines
Type 0			
Type 2			
Type 3			

- Vertically downwards:
  - formalisms become less powerful, languages become smaller
  - easier to analyze
- Horizontal:
  - Various grammars exactly equal to various automata

The Chomsky Hierarchie

Classification of languages (sets of words over Σ^*):

Chomsky Hierarchy:

Type	**Languages**	**Grammars**	**Machines**
Type 0	Rec. enumerable	Unrestricted	Turing Machine
Type 2			
Type 3			

- Vertically downwards:
  - formalisms become less powerful, languages become smaller
  - easier to analyze
- Horizontal:
  - Various grammars exactly equal to various automata

The Chomsky Hierarchie

Classification of languages (sets of words over Σ^*):

Chomsky Hierarchy:

Type	**Languages**	**Grammars**	**Machines**
Type 0	Rec. enumerable	Unrestricted	Turing Machine
	Recursive		(terminates)
Type 2			
Type 3			

- Vertically downwards:
  - formalisms become less powerful, languages become smaller
  - easier to analyze
- Horizontal:
  - Various grammars exactly equal to various automata

The Chomsky Hierarchie

Classification of languages (sets of words over Σ^*):

Chomsky Hierarchy:

Type	Languages	Grammars	Machines
Type 0	Rec. enumerable	Unrestricted	Turing Machine
	Recursive		(terminates)
Type 2	Context-free	Context-free	Pushdown automaton (PDA)
			Deterministic
Type 3			

- Vertically downwards:
  - formalisms become less powerful, languages become smaller
  - easier to analyze
- Horizontal:
  - Various grammars exactly equal to various automata

The Chomsky Hierarchie

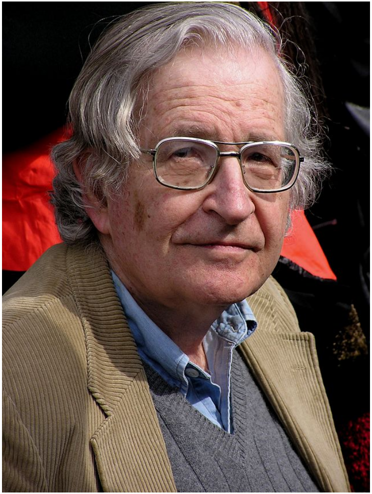
Classification of languages (sets of words over Σ^*):

Chomsky Hierarchy:

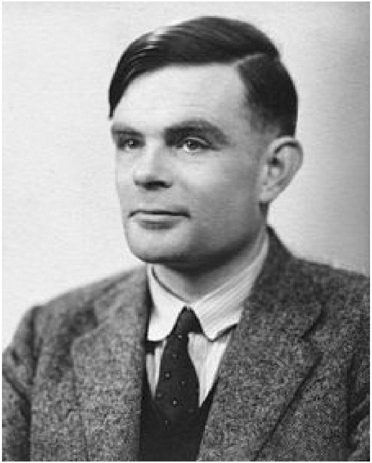
Type	Languages	Grammars	Machines
Type 0	Rec. enumerable	Unrestricted	Turing Machine
	Recursive		(terminates)
Type 2	Context-free	Context-free	Pushdown automaton (PDA)
			Deterministic
Type 3	Regular	Regular	Finite (DFA/NDA)

- Vertically downwards:
  - formalisms become less powerful, languages become smaller
  - easier to analyze
- Horizontal:
  - Various grammars exactly equal to various automata

Heroes and Pioneers



Noam Chomsky (1928)



Alan Turing (1912-1954)

Contents

What is the power of Turing Machines?

We have seen that Turing Machines can simulate:

- Non-deterministic Turing Machines
- Unrestricted Grammars

What is the power of Turing Machines?

We have seen that Turing Machines can simulate:

- Non-deterministic Turing Machines
- Unrestricted Grammars

Challenge

Can we build a single Turing Machine that can simulate all other Turing Machines?

What is the power of Turing Machines?

We have seen that Turing Machines can simulate:

- Non-deterministic Turing Machines
- Unrestricted Grammars

Challenge

Can we build a single Turing Machine that can simulate all other Turing Machines?

What should be the input to that Universal Turing Machine?

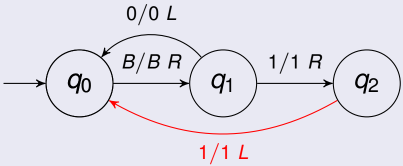
Encode a Turing Machine as input

- A Turing Machine can be encoded in 0's and 1's.
- Let $R(M)$ be the code of Turing Machine M.
- One can then use $R(M)$ as input for another TM M'.

Encode a Turing Machine as input

- A Turing Machine can be encoded in 0's and 1's.
- Let $R(M)$ be the code of Turing Machine M.
- One can then use $R(M)$ as input for another TM M'.

Turing Machine M:

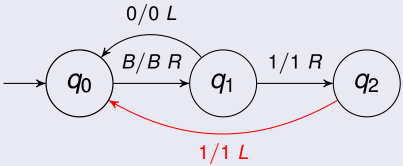


$$\delta(q_2, 1) = [q_0, 1, L]$$

Encode a Turing Machine as input

- A Turing Machine can be encoded in 0's and 1's.
- Let $R(M)$ be the code of Turing Machine M.
- One can then use $R(M)$ as input for another TM M'.

Turing Machine M:



$$\delta(q_2, 1) = [q_0, 1, L]$$

1110110101101

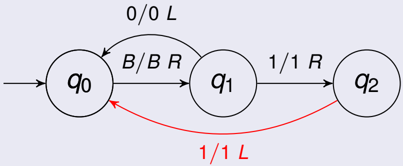
Encoding:

0	1
1	11
B	111
q_0	1
q_1	11
q_2	111
L	1
R	11

Encode a Turing Machine as input

- A Turing Machine can be encoded in 0's and 1's.
- Let $R(M)$ be the code of Turing Machine M.
- One can then use $R(M)$ as input for another TM M'.

Turing Machine M:



$$\delta(q_2, 1) = [q_0, 1, L]$$

1110110101101

Encoding:

0	1
1	11
B	111
q_0	1
q_1	11
q_2	111
L	1
R	11

$R(M) =$ 00010111011011101100110101010100110110111011011001110110101101000

The Universal Turing Machine

We can now build one Universal Turing Machine U:

For every Turing Machine M the following applies:

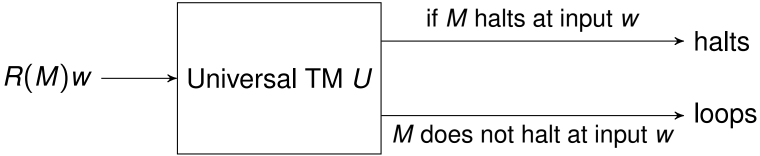
- If M halts for input w: Then U also halts for input $R(M)w$
- If M does not terminate for input w: Then U will also not terminate for input $R(M)w$

The Universal Turing Machine

We can now build one Universal Turing Machine U:

For every Turing Machine M the following applies:

- If M halts for input w: Then U also halts for input $R(M)w$
- If M does not terminate for input w: Then U will also not terminate for input $R(M)w$

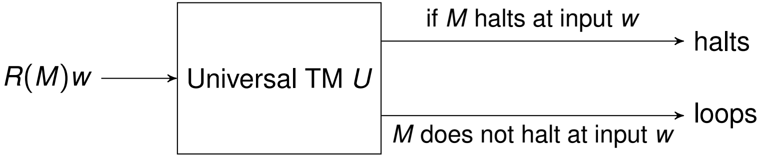


The Universal Turing Machine

We can now build one Universal Turing Machine U:

For every Turing Machine M the following applies:

- If M halts for input w: Then U also halts for input $R(M)w$
- If M does not terminate for input w: Then U will also not terminate for input $R(M)w$



So $\mathcal{L} = \{R(M)w \mid \text{TM } M \text{ halts with input } w\}$ is recursively enumerable.

Church-Turing thesis

Computer = Universal Turing Machine

U = hardware, $R(M)$ = software (!)

Church-Turing thesis

Computer = Universal Turing Machine

U = hardware, $R(M)$ = software (!)

Definition: Effective procedure (= algorithm)

A procedure is effective if it has the following properties:

- **Complete** The procedure always produces the correct answer
- **Mechanistic** Carrying out the procedure requires no intelligence
- **Deterministic** The procedure always returns the same answer

Church-Turing thesis

Computer = Universal Turing Machine

U = hardware, $R(M)$ = software (!)

Definition: Effective procedure (= algorithm)

A procedure is effective if it has the following properties:

- **Complete** The procedure always produces the correct answer
- **Mechanistic** Carrying out the procedure requires no intelligence
- **Deterministic** The procedure always returns the same answer

Church-Turing thesis

Every effective procedure can be encoded as a Turing Machine.

But then it becomes interesting to investigate which problems a Turing machine can not solve!

Contents

Decision problems and solutions

Definition: Decision problem

A *decision problem* is a question that takes some input and provides a yes/no answer

Example:

- Is a given number a square?
- Is there a path between node a and b in graph G?

A decision problem can be *encoded* as word on a Turing machine

- For example: by a series of digits, or even a row of ones

Decision problems and solutions

Definition: Decision problem

A *decision problem* is a question that takes some input and provides a yes/no answer

Example:

- Is a given number a square?
- Is there a path between node a and b in graph G?

A decision problem can be *encoded* as word on a Turing machine

- For example: by a series of digits, or even a row of ones

Definition: solution to a decision problem by a TM

A Turing machine M solves a decision problem if M always

- Halts in a final state if the answer is “yes”
- Halts in a non-final state if the answer is “no”
- *Always terminates!*

Decision problems and solutions

Definition: Decision problem

A *decision problem* is a question that takes some input and provides a yes/no answer

Example:

- Is a given number a square?
- Is there a path between node a and b in graph G?

A decision problem can be *encoded* as word on a Turing machine

- For example: by a series of digits, or even a row of ones

Definition: solution to a decision problem by a TM

A Turing machine M solves a decision problem if M always

- Halts in a final state if the answer is "yes"
- Halts in a non-final state if the answer is "no"
- *Always terminates!*

Church' thesis: A problem is decidable $\iff$ a TM exists for it

Decision Problems as Language Recognition

Using a suitable representation, we have encoded decision problems as input to a Turing Machine.

Decidability of Decision Problems

A (decision) problem is *decidable* iff it is solved by some TM.

Decision Problems as Language Recognition

Using a suitable representation, we have encoded decision problems as input to a Turing Machine.

Decidability of Decision Problems

A (decision) problem is *decidable* iff it is solved by some TM.

So to every decision problem we associate the language
$\mathcal{L}$ = inputs for which the answer to the problem is “yes”

A problem is *decidable* if its corresponding language is *recursive*

Decision Problems as Language Recognition

Using a suitable representation, we have encoded decision problems as input to a Turing Machine.

Decidability of Decision Problems

A (decision) problem is *decidable* iff it is solved by some TM.

So to every decision problem we associate the language
$\mathcal{L}$ = inputs for which the answer to the problem is "yes"

A problem is *decidable* if its corresponding language is *recursive*

- Question: Do undecidable problems exist?
- Equivalent question: Do non-recursive languages exist?

Decision Problems as Language Recognition

Using a suitable representation, we have encoded decision problems as input to a Turing Machine.

Decidability of Decision Problems

A (decision) problem is *decidable* iff it is solved by some TM.

So to every decision problem we associate the language
$\mathcal{L}$ = inputs for which the answer to the problem is "yes"

A problem is *decidable* if its corresponding language is *recursive*

- Question: Do undecidable problems exist?
- Equivalent question: Do non-recursive languages exist?
- Answer *yes!* And this puts a limit to what computers can do

Problem Reduction

A problem can be solved directly by providing a TM that decides it.
Alternative, we could reduce it an already solved problem:

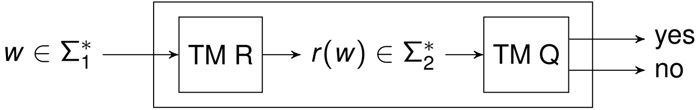
Problem Reduction

A problem can be solved directly by providing a TM that decides it. Alternative, we could reduce it an already solved problem:

Effective Problem Reduction (many-to-one)

$L \subseteq \Sigma_1^*$ is reducible to $Q \subseteq \Sigma_2^*$ if there exists a Turing Machine M that implements a function $r : \Sigma_1^* \to \Sigma_2^*$, such that for all w:

$$w \in L \text{ if and only if } r(w) \in Q$$



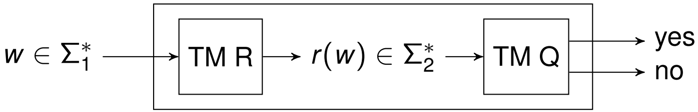
Problem Reduction

A problem can be solved directly by providing a TM that decides it.
Alternative, we could reduce it an already solved problem:

Effective Problem Reduction (many-to-one)

$L \subseteq \Sigma_1^*$ is reducible to $Q \subseteq \Sigma_2^*$ if there exists a Turing Machine M that implements a function $r : \Sigma_1^* \to \Sigma_2^*$, such that for all w:

$$w \in L \text{ if and only if } r(w) \in Q$$



Problem Reduction and Decidability

- If L is reducible to Q and Q is decidable, then L is decidable
- If L is reducible to Q and L is not decidable, then Q is also not decidable

Contents

The *Halting* problem

Definition: Halting problem

Will a given Turing machine halt when provided a given input?

The *Halting* problem

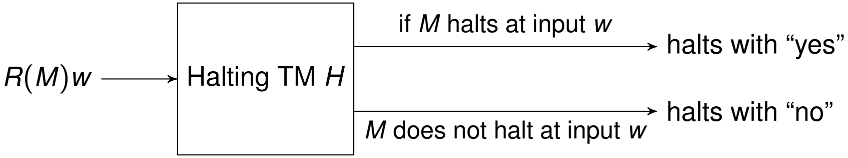
Definition: Halting problem

Will a given Turing machine halt when provided a given input?

The *Halting* problem

Definition: Halting problem

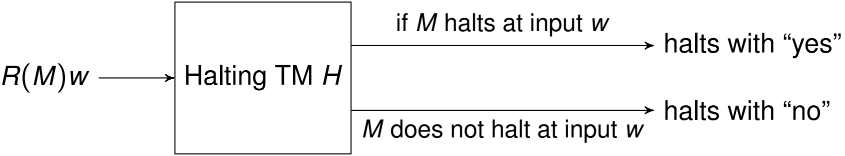
Will a given Turing machine halt when provided a given input?



The *Halting* problem

Definition: Halting problem

Will a given Turing machine halt when provided a given input?



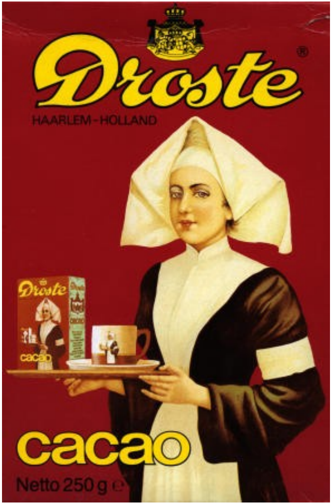
- Turing machine and input word are parameters of this problem
- Required: encoding of Turing machine M: $R(M) \in \Sigma^*$

Theorem

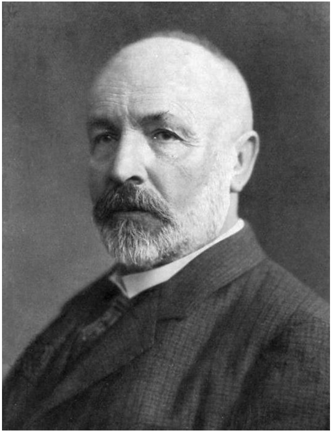
The halting problem is undecidable.

Proof: by contradiction and self-reference (diagonalisation).

Self reference



The Droste effect

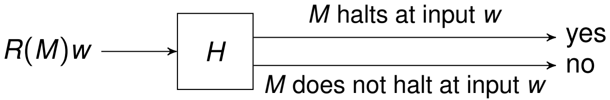


Georg Cantor (1845-1918)

(Remember Cantor's diagonal argument?)

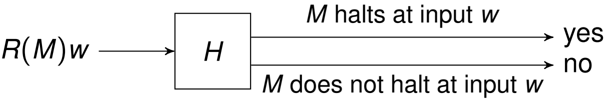
Undecidability of the *halting* problem

1. Suppose we have a solution H of the halting problem

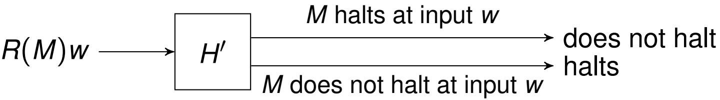


Undecidability of the *halting* problem

1. Suppose we have a solution H of the halting problem

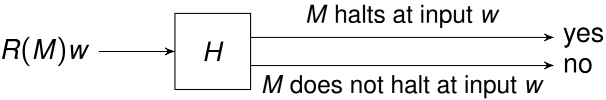


2. Transform this into H' by not terminating in every successful state

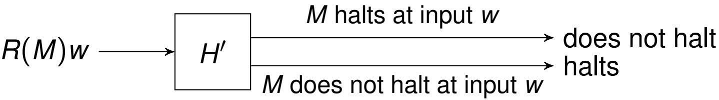


Undecidability of the *halting* problem

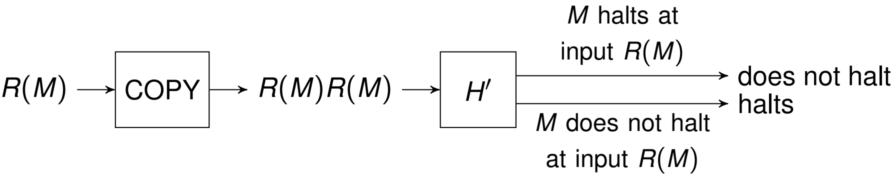
1. Suppose we have a solution H of the halting problem



2. Transform this into H' by not terminating in every successful state

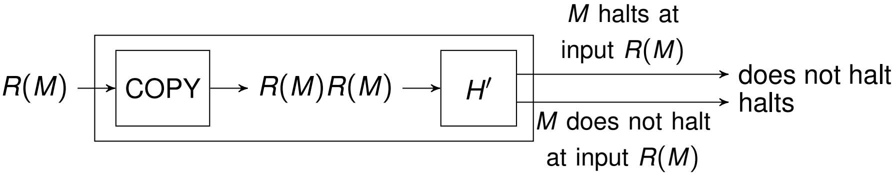


3. Use the encoding $R(M)$ as input w



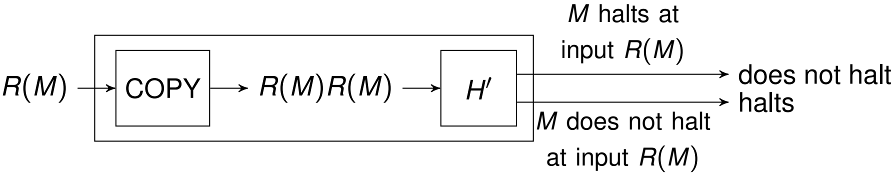
Undecidability of the *halting* problem, continuation

3. Name the new machine D

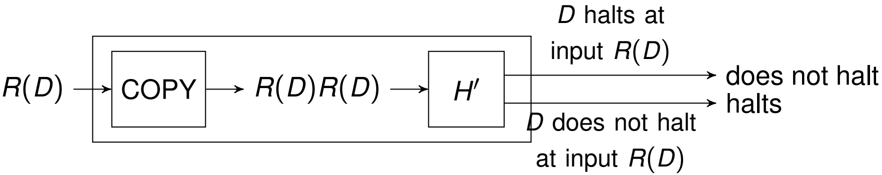


Undecidability of the *halting* problem, continuation

3. Name the new machine D

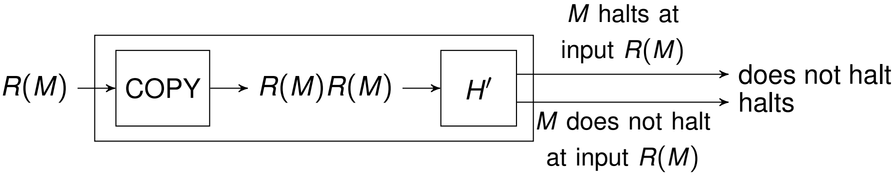


4. Give the encoding of D to itself as input

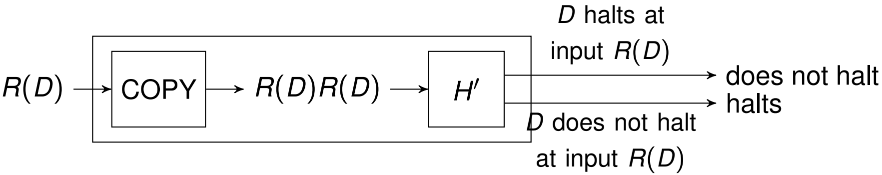


Undecidability of the *halting* problem, continuation

3. Name the new machine D



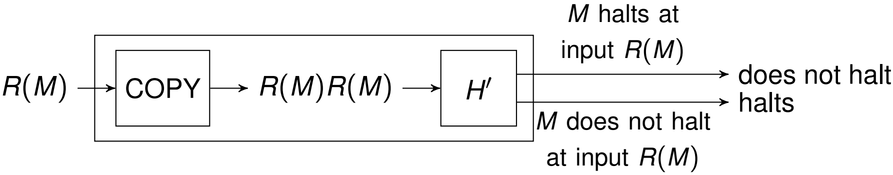
4. Give the encoding of D to itself as input



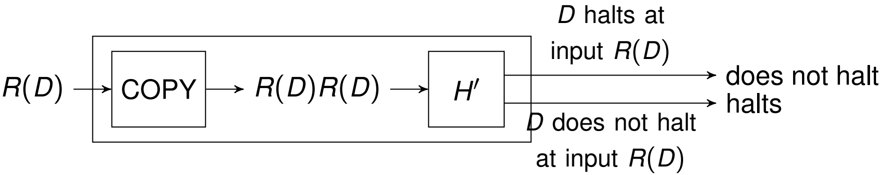
5. Contradiction: the computation halts $\iff$ the computation does not terminate

Undecidability of the *halting* problem, continuation

3. Name the new machine D



4. Give the encoding of D to itself as input



5. Contradiction: the computation halts $\iff$ the computation does not terminate
6. Conclusion: the original machine H cannot exist

Direct consequences of the Halting Problem

- There is no program that can check the termination of all other programs, which (given unbounded resources) always terminates with the correct answer.
- This has not stopped scientists to organise termination competitions!

Direct consequences of the Halting Problem

- There is no program that can check the termination of all other programs, which (given unbounded resources) always terminates with the correct answer.
- This has not stopped scientists to organise termination competitions!
- The class of recursive languages is *strictly smaller* than the class of recursively enumerable languages:
  - The Halting Problem is not a recursive language
  - The Halting Problem is recursively enumerable (universal TM)

Direct consequences of the Halting Problem

- There is no program that can check the termination of all other programs, which (given unbounded resources) always terminates with the correct answer.
- This has not stopped scientists to organise termination competitions!
- The class of recursive languages is *strictly smaller* than the class of recursively enumerable languages:
  - The Halting Problem is not a recursive language
  - The Halting Problem is recursively enumerable (universal TM)
- Example:
  - The language of all syntactically correct Python programs is recursive
  - The language of all *terminating* Pyton programs is recursively enumerable, but not recursive!

More Consequences

By Problem Reduction: many other problems are also undecidable

- Blank Tape Problem – does M halt with an empty tape as input?
- Strategy: Reduce Halting Problem to Blank Tape Problem (!)
- Input for Halting Problem: $R(M)w$.
- Transform this input to: $R(M')$, where M' first writes word w and then executes M on w.
- Blank Tape for $R(M') \iff$ Halting for $R(M)w$.

More Consequences

By Problem Reduction: many other problems are also undecidable

- Blank Tape Problem – does M halt with an empty tape as input?
- Strategy: Reduce Halting Problem to Blank Tape Problem (!)
- Input for Halting Problem: $R(M)w$.
- Transform this input to: $R(M')$, where M' first writes word w and then executes M on w.
- Blank Tape for $R(M') \iff$ Halting for $R(M)w$.

Rice's Theorem

Every non-trivial property of recursively enumerable languages of the form $\mathcal{L}(M)$ is undecidable

Examples:

- Is $\mathcal{L}(M)$ empty? context-free? regular? Is $\lambda \in \mathcal{L}(M)$?
- Are $\mathcal{L}(M)$ and $\mathcal{L}(M')$ equivalent?

Other Undecidable Problems

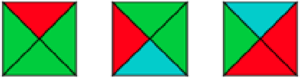
Decidable:

- Given CFG G and word w: is $w \in \mathcal{L}(G)$?

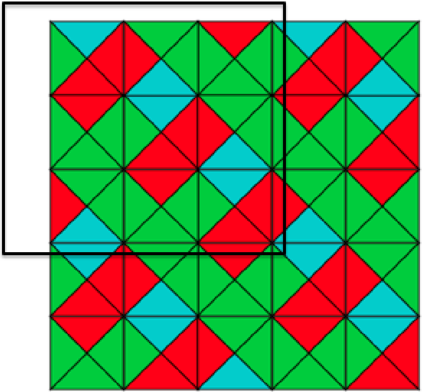
Undecidable:

- Given CFG G_1 and G_2. Is $\mathcal{L}(G_1) = \mathcal{L}(G_2)$?
- Post's Correspondence Problem:
  - Input: two finite sequences of words ("dominos") $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$.
  - Question: is there a finite sequence of dominos (with copies) such that $x_{f(1)}, \ldots, x_{f(m)} = y_{f(1)}, \ldots y_{f(m)}$?
- Tiling Problem: Given n tiles, can we tile any m x m area?

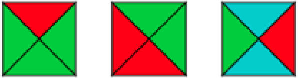
Positive Tiling Instance



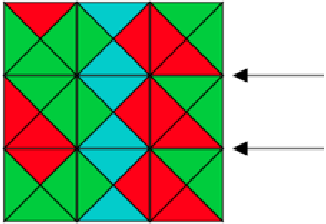
YES



Negative Tiling Instance



NO



Seen this week

Chomsky Hierarchy

- Turing Machines
  - Multi-tape, Non-deterministic – it does not matter
- Unrestricted grammars
  - Equivalent to Turing machines (recursively enumerable languages)
- Seen before: Context-free grammars
  - Equivalent to pushdown automata
- Regular grammars
  - Equivalent to finite automata and regular expressions

Undecidability

- Not all problems can be solved efficiently (P vs NP)
- Some problems cannot be solved at all! (halting problem)

Watch this great animation:

https://www.youtube.com/watch?v=92WHN-pAFCs