university of groningen

Languages and Machines

L12: Non Context-Free Grammars/Languages

Jorge A. Pérez

Bernoulli Institute for Math, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

June 6, 2024

Last Lecture(s)

- Church-Turing's thesis
- A Universal Turing machine
- Undecidability results / problem reducibility
- Acceptance of the empty string (the blank tape problem)
- Incompleteness of arithmetic

Today: Non-CF grammars and Course evaluation.

Outline

From Lecture 2: Context-Free Grammars

A quadruple (V, Σ, P, S) where

- V is a set of **variables** or **nonterminals**
- Σ is an alphabet of **terminals**, disjoint from V
- P is a finite set of **production rules**, taken from $V \times (V \cup \Sigma)^*$.
  We write $A \to w$ instead of (A, w).
- $S \in V$ is the **start symbol**.

From Lecture 2: Context-Free Grammars

A quadruple (V, Σ, P, S) where

- V is a set of **variables** or **nonterminals**
- Σ is an alphabet of **terminals**, disjoint from V
- P is a finite set of **production rules**, taken from $V \times (V \cup \Sigma)^*$.
  We write $A \to w$ instead of (A, w).
- $S \in V$ is the **start symbol**.

Notice:

- $A \to w$ involves rewriting:
  A **symbol** $A \in V$ is rewritten into a **string** in $w \in (V \cup \Sigma)^*$

Today:

- Grammars that rewrite a **string** u into **another string** w

The Chomsky Hierarchy

Grammar	Language	Machine(s)
Type 0	R.e./Semi-decidable	TMs
?	Recursive / Decidable	Always-terminating TMs
Type 1	Context-sensitive	Linear-bounded automata
Type 2	Context-free	Pushdown automata
Type 3	Regular	NϵFSM / NFSM / DFSM

where

- Type 0 = Unrestricted
- Type 1 = Context-sensitive
- Type 2 = Context-free
- Type 3 = Regular

Outline

Rewriting Systems

Let $u, v, \ldots$ range over strings over an alphabet Σ.

- A rewriting rule is a pair (u, v), written $u \to v$
- A (string) rewriting system is a set of rewriting rules

$$u_1 \to v_1, \cdots, u_n \to v_n$$

Rewriting Systems

Let $u, v, \ldots$ range over strings over an alphabet Σ.

- A rewriting rule is a pair (u, v), written $u \to v$
- A (string) rewriting system is a set of rewriting rules

$$u_1 \to v_1, \cdots, u_n \to v_n$$

Given a rewriting system P, we write $x \Rightarrow_P y$ if
$x = uvw$, $(v, z) \in P$, and $y = uzw$.

Relation $\Rightarrow_P^*$ is the reflexive, transitive closure of $\Rightarrow_P$.

Rewriting Systems

Let $u, v, \ldots$ range over strings over an alphabet Σ.

- A rewriting rule is a pair (u, v), written $u \to v$
- A (string) rewriting system is a set of rewriting rules

$$u_1 \to v_1, \cdots, u_n \to v_n$$

Given a rewriting system P, we write $x \Rightarrow_P y$ if $x = uvw$, $(v, z) \in P$, and $y = uzw$.

Relation $\Rightarrow_P^*$ is the reflexive, transitive closure of $\Rightarrow_P$.

The Nullability Problem (NULL):
Given a string w and a rewriting system P, does $w \Rightarrow_P^* \epsilon$ hold?

NULL is Undecidable

- Key Idea: Given a simple TM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, define a set of rewriting rules P such that:

$$M \text{ terminates on input } w \text{ iff } [q_0 w] \Rightarrow_P^* \epsilon$$

NULL is Undecidable

- Key Idea: Given a simple TM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, define a set of rewriting rules P such that:

$$M \text{ terminates on input } w \text{ iff } [q_0 w] \Rightarrow_P^* \epsilon$$

- The alphabet for rewriting is $\Gamma \cup Q \cup \{[\,, E\,,]\,\}$, where E ('end') is an auxiliary symbol, to be erased during rewriting

NULL is Undecidable

- Key Idea: Given a simple TM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, define a set of rewriting rules P such that:

$$M \text{ terminates on input } w \text{ iff } [q_0 w] \Rightarrow_P^* \epsilon$$

- The alphabet for rewriting is $\Gamma \cup Q \cup \{[\,, E\,,]\,\}$, where E ('end') is an auxiliary symbol, to be erased during rewriting
- To simulate M, P will enact the rewriting sequence ($x, y \in \Gamma^*$):

$$\underbrace{[q_0 w] \Rightarrow_P^* [xEy]}_{\text{Part I}}$$

NULL is Undecidable

- Key Idea: Given a simple TM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, define a set of rewriting rules P such that:

$$M \text{ terminates on input } w \text{ iff } [q_0 w] \Rightarrow_P^* \epsilon$$

- The alphabet for rewriting is $\Gamma \cup Q \cup \{[\,, E\,,]\,\}$, where E ('end') is an auxiliary symbol, to be erased during rewriting
- To simulate M, P will enact the rewriting sequence ($x, y \in \Gamma^*$):

$$\underbrace{[q_0 w] \Rightarrow_P^* [xEy]}_{\text{Part I}} \text{ followed by } \underbrace{[xEy] \Rightarrow_P^* [xE] \Rightarrow_P^* [E] \Rightarrow_P \epsilon}_{\text{Part II}}$$

NULL is Undecidable

- Key Idea: Given a simple TM $M = (Q, \Sigma, \Gamma, \delta, q_0)$, define a set of rewriting rules P such that:

$$M \text{ terminates on input } w \text{ iff } [q_0 w] \Rightarrow_P^* \epsilon$$

- The alphabet for rewriting is $\Gamma \cup Q \cup \{[\,, E\,,]\,\}$, where E ('end') is an auxiliary symbol, to be erased during rewriting
- To simulate M, P will enact the rewriting sequence ($x, y \in \Gamma^*$):

$$\underbrace{[q_0 w] \Rightarrow_P^* [xEy]}_{\text{Part I}} \text{ followed by } \underbrace{[xEy] \Rightarrow_P^* [xE] \Rightarrow_P^* [E] \Rightarrow_P \epsilon}_{\text{Part II}}$$

Parts I and II rely on different rewriting rules (see next slide).

- A **reduction** of HALT into NULL:
If NULL were decidable, then we could use the rewriting system P to solve HALT.

NULL is Undecidable

$$\underbrace{[q_0 w] \Rightarrow_P^* [xEy]}_{\text{Part I}} \text{ followed by } \underbrace{[xEy] \Rightarrow_P^* [xE] \Rightarrow_P^* [E] \Rightarrow_P \epsilon}_{\text{Part II}}$$

Rewriting rules for Part I (assume $q, r \in Q$ and X, Y, $Z \in \Gamma$):

1. $[\,qX \to [\,BqX$
2. $q\,] \to qB\,]$ ("surround" the configuration with blank symbols)
3. $ZqX \to ZYr$ if $\delta(q, X) = (r, Y, R)$ (moving right)
4. $ZqX \to rZY$ if $\delta(q, X) = (r, Y, L)$ (moving left)
5. $ZqX \to ZEX$ if $\delta(q, X) = \perp$ (termination: add E)

NULL is Undecidable

$$\underbrace{[q_0 w] \Rightarrow_P^* [xEy]}_{\text{Part I}} \text{ followed by } \underbrace{[xEy] \Rightarrow_P^* [xE] \Rightarrow_P^* [E] \Rightarrow_P \epsilon}_{\text{Part II}}$$

Rewriting rules for Part I (assume $q, r \in Q$ and X, Y, $Z \in \Gamma$):

1. $[\,qX \to [\,BqX$
2. $q\,] \to qB\,]$ ("surround" the configuration with blank symbols)
3. $ZqX \to ZYr$ if $\delta(q, X) = (r, Y, R)$ (moving right)
4. $ZqX \to rZY$ if $\delta(q, X) = (r, Y, L)$ (moving left)
5. $ZqX \to ZEX$ if $\delta(q, X) = \perp$ (termination: add E)

Rewriting rules for Part II:

6. $EX \to E$
7. $XE] \to E\,]$
8. $[\,E\,] \to \epsilon$

NULL is Undecidable

$$\underbrace{[q_0 w] \Rightarrow_P^* [xEy]}_{\text{Part I}} \text{ followed by } \underbrace{[xEy] \Rightarrow_P^* [xE] \Rightarrow_P^* [E] \Rightarrow_P \epsilon}_{\text{Part II}}$$

Rewriting rules for Part I (assume $q, r \in Q$ and X, Y, $Z \in \Gamma$):

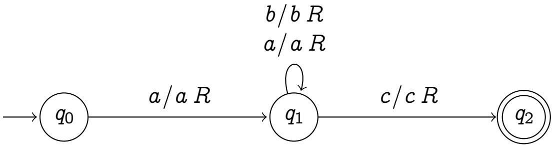
1. $[\,qX \rightarrow [\,BqX$
2. $q\,] \rightarrow qB\,]$ ("surround" the configuration with blank symbols)
3. $ZqX \rightarrow ZYr$ if $\delta(q, X) = (r, Y, R)$ (moving right)
4. $ZqX \rightarrow rZY$ if $\delta(q, X) = (r, Y, L)$ (moving left)
5. $ZqX \rightarrow ZEX$ if $\delta(q, X) = \perp$ (termination: add E)

Rewriting rules for Part II:

6. $EX \rightarrow E$
7. $XE] \rightarrow E\,]$
8. $[\,E\,] \rightarrow \epsilon$

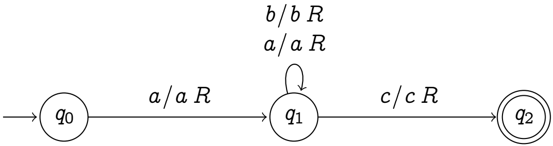
Since $[xEy] \Rightarrow_P^* \epsilon$, we have that $[q_0 w] \Rightarrow_P^* \epsilon$ if $w \in L(M)$.
If $w \notin L(M)$, symbol E is never generated - ϵ is not reached.

Example



$$
\begin{aligned}
& q_0\, aac \\
\vdash\ & a q_1\, ac \\
\vdash\ & aa q_1\, c \\
\vdash\ & aac q_2
\end{aligned}
$$

Example



$$
\begin{array}{ll}
 & [q_0\, aac] \\
q_0\, aac & \Rightarrow_1 [Bq_0\, aac] \\
\vdash\ aq_1\, ac & \Rightarrow_3 [Baq_1\, ac] \\
\vdash\ aaq_1\, c & \Rightarrow_3 [Baaq_1\, c] \\
\vdash\ aacq_2 & \Rightarrow_3 [Baacq_2] \\
 & \Rightarrow_2 [Baacq_2\, B] \\
 & \Rightarrow_5 [BaacEB] \Rightarrow_6 [BaacE] \Rightarrow_7^* [E] \Rightarrow_8 \epsilon
\end{array}
$$

Outline

Chomsky's Hiearchy

Rewriting Systems

Unrestricted Grammars

Context-Sensitive Grammars

Unrestricted (Type 0) Grammars

The most powerful grammars in Chomsky's hierarchy

- A grammar $G = (V, \Sigma, P, S)$ in which P is a **rewriting system**
- Derivations, sentential forms, sentences: as before

Unrestricted (Type 0) Grammars

The most powerful grammars in Chomsky's hierarchy

- A grammar $G = (V, \Sigma, P, S)$ in which P is a **rewriting system**
- Derivations, sentential forms, sentences: as before
- **Example**: Consider $V = \{S, A, C\}$, $\Sigma = \{a, b, c\}$ and

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

Unrestricted (Type 0) Grammars

The most powerful grammars in Chomsky's hierarchy

- A grammar $G = (V, \Sigma, P, S)$ in which P is a **rewriting system**
- Derivations, sentential forms, sentences: as before
- **Example**: Consider $V = \{S, A, C\}$, $\Sigma = \{a, b, c\}$ and

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

We have $S \Rightarrow \epsilon$ and $S \Rightarrow a^{i+1}b^{i+1}c^{i+1}$ (not a CFL):

$$\begin{aligned} S \Rightarrow\ & aAbc \\ \Rightarrow\ & aaAbCbc \Rightarrow aaaAbCbCbc \Rightarrow \cdots \Rightarrow a^{i+1}A(bC)^i bc \\ \Rightarrow\ & a^{i+1}(bC)^i bc \Rightarrow a^{i+1}b^{i+1}C^i c \Rightarrow a^{i+1}b^{i+1}c^{i+1} \end{aligned}$$

Unrestricted (Type 0) Grammars

Grammar G with alphabet $\{a, b, [,]\}$
and productions:

$$
\begin{aligned}
S &\rightarrow aT[a] \mid bT[b] \mid [\,] \\
T[&\rightarrow aT[A \mid bT[B \mid [\\
Aa &\rightarrow aA \\
Ab &\rightarrow bA \\
Ba &\rightarrow aB \\
Bb &\rightarrow bB \\
A] &\rightarrow a] \\
B] &\rightarrow b]
\end{aligned}
$$

Unrestricted (Type 0) Grammars

Grammar G with alphabet $\{a, b, [,]\}$ and productions:

$$
\begin{aligned}
S &\rightarrow aT[a] \mid bT[b] \mid [\,] \\
T[&\rightarrow aT[A \mid bT[B \mid [\\
Aa &\rightarrow aA \\
Ab &\rightarrow bA \\
Ba &\rightarrow aB \\
Bb &\rightarrow bB \\
A] &\rightarrow a] \\
B] &\rightarrow b]
\end{aligned}
$$

$L(G) = \{u[u] \mid u \in \{a, b\}^*\}$.
For instance:

$$
\begin{aligned}
S &\Rightarrow a\,\underline{T\,[}\,a\,] \\
&\Rightarrow a\,a\,T\,[\,\underline{A\,a}\,] \\
&\Rightarrow a\,a\,T\,[\,a\,\underline{A\,]} \\
&\Rightarrow a\,a\,\underline{T\,[}\,a\,a\,] \\
&\Rightarrow a\,a\,b\,T\,[\,\underline{B\,a}\,a\,] \\
&\Rightarrow a\,a\,b\,T\,[\,a\,\underline{B\,a}\,] \\
&\Rightarrow a\,a\,b\,T\,[\,a\,a\,\underline{B\,]} \\
&\Rightarrow a\,a\,b\,\underline{T\,[}\,a\,a\,b\,] \\
&\Rightarrow a\,a\,b\,[\,a\,a\,b\,]
\end{aligned}
$$

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3
2. Choose a rule $u \to v$ from T2

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3
2. Choose a rule $u \to v$ from T2
3. An instance of u on T3 is chosen, if one exists; otherwise, then halt in a rejecting state.

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \rightarrow v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3
2. Choose a rule $u \rightarrow v$ from T2
3. An instance of u on T3 is chosen, if one exists; otherwise, then halt in a rejecting state.
4. The string u is replaced by v on T3

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3
2. Choose a rule $u \to v$ from T2
3. An instance of u on T3 is chosen, if one exists; otherwise, then halt in a rejecting state.
4. The string u is replaced by v on T3
5. If T1 and T3 match, then halt in an accepting state

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3
2. Choose a rule $u \to v$ from T2
3. An instance of u on T3 is chosen, if one exists; otherwise, then halt in a rejecting state.
4. The string u is replaced by v on T3
5. If T1 and T3 match, then halt in an accepting state
6. To apply some other rule, repeat steps 2-5.

Unrestricted (Type 0) Grammars and TMs

Theorem 7.2: Simulate a Type 0 grammar G using a three-tape non-deterministic TM M such that $L(M) = L(G)$.

- T1: the input string x
- T2: rules of G (e.g. $u \to v$ encoded as $u\#v$)
- T3: the derivations of G

Computation proceeds as follows:

1. Write the start symbol S on T3
2. Choose a rule $u \to v$ from T2
3. An instance of u on T3 is chosen, if one exists; otherwise, then halt in a rejecting state.
4. The string u is replaced by v on T3
5. If T1 and T3 match, then halt in an accepting state
6. To apply some other rule, repeat steps 2-5.

Hence, if G is Type 0 then $L(G)$ is recursively enumerable

Unrestricted (Type 0) Grammars and TMs

Theorem 7.3: Conversely, Type 0 grammars can simulate TMs:

- Give a grammar G such that $L(G)$ is the reversal of $L(M)$
  This can be done in two phases (Lemma 7.2):
  1. $S \Rightarrow^* w^R[q_0 w]$, for some $w \in \Sigma^*$
  2. $w^R[q_0 w] \Rightarrow^* w^R$, if $w \in L(M)$.
- Given M, construct a machine M' that
  - Reverts the input string w
  - Applies M to w^R, so that $w_i \in L(M') \iff w_i^R \in L(M)$
  - Since $L(M') = \{w \mid w^R \in L(M)\}$, use Lemma 7.2 to obtain a G such that

$$L(G) = \{w_i^R \mid w_i \in L(M')\} = L(M)$$

Alternatively... (1/2)

- Let L be a r.e. language, recognized by $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$.
- $G = (V, \Sigma, P, S)$ is designed to simulate the computations of M
- The effect of transition $\delta(q_i, x) = (q_j, y, R)$ on a configuration encoded as uq_ixvB is the derivation

$$u\ q_i\ x\ v\ B \Rightarrow u\ q_j\ y\ v\ B$$

- Deriving a terminal string in G in three phases:
  a) Generate a string $u[q_0Bu]$, with $u \in \Sigma^*$
  b) Simulate M on the string $u[q_0Bu]$
  c) Remove the simulation substring
- Let $\Sigma = \{a_1, \dots, a_n\}$ and
  $V = \{S, T, E_R, E_L, [,], A_1, \dots, A_n\} \cup Q$
  The rules (next slide) ensure that a derivation that begins by generating $u\,[\,q_0\ B\ u\,]$ terminates with u whenever $u \in L(M)$; otherwise, the derivation doesn't produce a terminal

Alternatively... (2/2)

The first four rules are similar to the example given before:

1. $S \to a_i\ T\,[\,a_i\,] \mid [\,q_0\ B\,]$ for $1 \leq i \leq n$
2. $A_i\ a_j \to a_j\ A_i$ for $1 \leq i, j \leq n$
3. $A_i\,] \to a_i\,]$ for $1 \leq i \leq n$
4. $T\,[\to a_i\ T\,[\,A_i \mid [\,q_0\ B$

Alternatively... (2/2)

The first four rules are similar to the example given before:

1. $S \to a_i\, T\,[\,a_i\,]\,|\,[\,q_0\, B\,]$ for $1 \leq i \leq n$
2. $A_i\, a_j \to a_j\, A_i$ for $1 \leq i, j \leq n$
3. $A_i\,] \to a_i\,]$ for $1 \leq i \leq n$
4. $T\,[\to a_i\, T\,[\,A_i\,|\,[\,q_0\, B$

The following rules follow δ:

5. $q_i\, x\, y \to z\, q_j\, y$ whenever $\delta(q_i, x) = (q_j, z, R)$ and $y \in \Gamma$
6. $q_i\, x\,] \to z\, q_j\, B\,]$ whenever $\delta(q_i, x) = (q_j, z, R)$
7. $y\, q_i\, x \to q_j\, y\, z$ whenever $\delta(q_i, x) = (q_j, z, L)$ and $y \in \Gamma$

Alternatively... (2/2)

The first four rules are similar to the example given before:

1. $S \to a_i \, T \, [\, a_i \,] \mid [\, q_0 \, B \,]$ for $1 \leq i \leq n$
2. $A_i \, a_j \to a_j \, A_i$ for $1 \leq i, j \leq n$
3. $A_i \,] \to a_i \,]$ for $1 \leq i \leq n$
4. $T \,[\to a_i \, T \,[\, A_i \mid [\, q_0 \, B$

The following rules follow δ:

5. $q_i \, x \, y \to z \, q_j \, y$ whenever $\delta(q_i, x) = (q_j, z, R)$ and $y \in \Gamma$
6. $q_i \, x \,] \to z \, q_j \, B \,]$ whenever $\delta(q_i, x) = (q_j, z, R)$
7. $y \, q_i \, x \to q_j \, y \, z$ whenever $\delta(q_i, x) = (q_j, z, L)$ and $y \in \Gamma$

If an accepting state is reached, erase the string within brackets:

8. $q_i \, x \to E_R$ whenever $\delta(q_i, x)$ is undefined and $q_i \in F$
9. $E_R \, x \to E_R$ for $x \in \Gamma$
10. $E_R \,] \to E_L$ for $x \in \Gamma$
11. $x \, E_L \to E_L$ for $x \in \Gamma$
12. $[\, E_L \to \epsilon$

Outline

Context-Sensitive (Type 1) Grammars

A Type 0 grammar is context sensitive if every $u \to v$ satisfies

- S doesn't occur in v
- If $u \neq S$ then $0 < |u| \leq |v|$

Thus, the length of the derived string remains the same or increases with each rule application (a monotonicity property).

We can see that every context-free language is context-sensitive:

- A context-free grammar is context-sensitive iff it is essentially non-contracting.
- Every context-free grammar is equivalent with an essentially non-contracting context-free grammar

Context-sensitive languages are accepted by always-terminating TMs. Hence, they are recursive.

Context-Sensitive (Type 1) Grammars

Let $V = \{S, A, C\}$, $\Sigma = \{a, b, c\}$ and the unrestricted grammar that generates the language $\{a^i b^i c^i \mid i > 0\}$:

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

An equivalent context-sensitive grammar:

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

Context-Sensitive (Type 1) Grammars

- A linear-bounded automaton is a non-deterministic, single-tape TM whose transitions never replace a blank symbol B.
- That is, the input string determines the length of the available tape. This effectively decreases the expressivity of TMs.
- L is accepted by a linear-bounded automaton iff L is context-sensitive.
- Simulating the derivations of a context-sensitive grammar requires some effort.

The Chomsky Hierarchy

Grammar	Language	Machine(s)
Type 0	R.e./Semi-decidable	TMs
?	Recursive / Decidable	Always-terminating TMs
Type 1	Context-sensitive	Linear-bounded automata
Type 2	Context-free	Pushdown automata
Type 3	Regular	NϵFSM / NFSM / DFSM

where

- Type 0 = Unrestricted
- Type 1 = Context-sensitive
- Type 2 = Context-free
- Type 3 = Regular

Course Evaluation

- Lectures:
  Useful, understandable, too fast, too slow ...?
- Content:
  Too much, too little, useful for your (professional) life, ...?
- Tutorials:
  Helpful, interesting, ...?
- Homeworks:
  Easy, difficult, ...?
- Material (Reader and slides):
  Clear, complete, helpful, ...?

Feel free to send me an email with your constructive criticism!

Languages and Machines

L12: Non Context-Free Grammars/Languages

Jorge A. Pérez

Bernoulli Institute for Math, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

June 6, 2024