CS 373: Theory of Computation

Gul Agha Mahesh Viswanathan

University of Illinois, Urbana-Champaign

Fall 2010

Grammars

Definition

A grammar is $G = (V, \Sigma, R, S)$, where

- V is a finite set of variables/non-terminals
- Σ is a finite set of terminals
- $S \in V$ is the start symbol
- $R \subseteq (\Sigma \cup V)^* \times (\Sigma \cup V)^*$ is a finite set of rules/productions

Grammars

Definition

A grammar is $G = (V, \Sigma, R, S)$, where

- V is a finite set of variables/non-terminals
- Σ is a finite set of terminals
- $S \in V$ is the start symbol
- $R \subseteq (\Sigma \cup V)^* \times (\Sigma \cup V)^*$ is a finite set of rules/productions

We say $\gamma_1 \alpha \gamma_2 \Rightarrow_G \gamma_1 \beta \gamma_2$ iff $(\alpha \rightarrow \beta) \in R$.

Grammars

Definition

A grammar is $G = (V, \Sigma, R, S)$, where

- V is a finite set of variables/non-terminals
- Σ is a finite set of terminals
- $S \in V$ is the start symbol
- $R \subseteq (\Sigma \cup V)^* \times (\Sigma \cup V)^*$ is a finite set of rules/productions

We say $\gamma_1 \alpha \gamma_2 \Rightarrow_G \gamma_1 \beta \gamma_2$ iff $(\alpha \to \beta) \in R$. And
$L(G) = \{w \in \Sigma^* \mid S \stackrel{*}{\Rightarrow}_G w\}$

Example

Example

Consider the grammar G with $\Sigma = \{a\}$ with

$$
\begin{array}{lll}
S \rightarrow \$Ca\# \mid a \mid \epsilon & Ca \rightarrow aaC & \$D \rightarrow \$C \\
C\# \rightarrow D\# \mid E & aD \rightarrow Da & aE \rightarrow Ea \\
\$E \rightarrow \epsilon & &
\end{array}
$$

The following are derivations in this grammar

$$
S \Rightarrow \$Ca\# \Rightarrow \$aaC\# \Rightarrow \$aaE \Rightarrow \$aEa \Rightarrow \$Eaa \Rightarrow aa
$$

$$
\begin{aligned}
S &\Rightarrow \$Ca\# \Rightarrow \$aaC\# \Rightarrow \$aaD\# \Rightarrow \$aDa\# \Rightarrow \$Daa\# \Rightarrow \$Caa\# \\
&\Rightarrow \$aaCa\# \Rightarrow \$aaaaC\# \Rightarrow \$aaaaE \Rightarrow \$aaaEa \Rightarrow \$aaEaa \\
&\Rightarrow \$aEaaa \Rightarrow \$Eaaaa \Rightarrow aaaa
\end{aligned}
$$

Example

Example

Consider the grammar G with $\Sigma = \{a\}$ with

$$
\begin{array}{lll}
S \rightarrow \$Ca\# \mid a \mid \epsilon & Ca \rightarrow aaC & \$D \rightarrow \$C \\
C\# \rightarrow D\# \mid E & aD \rightarrow Da & aE \rightarrow Ea \\
\$E \rightarrow \epsilon & &
\end{array}
$$

The following are derivations in this grammar

$$S \Rightarrow \$Ca\# \Rightarrow \$aaC\# \Rightarrow \$aaE \Rightarrow \$aEa \Rightarrow \$Eaa \Rightarrow aa$$

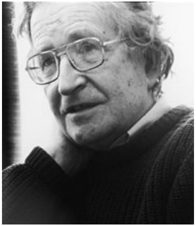
$$
\begin{aligned}
S &\Rightarrow \$Ca\# \Rightarrow \$aaC\# \Rightarrow \$aaD\# \Rightarrow \$aDa\# \Rightarrow \$Daa\# \Rightarrow \$Caa\# \\
&\Rightarrow \$aaCa\# \Rightarrow \$aaaaC\# \Rightarrow \$aaaaE \Rightarrow \$aaaEa \Rightarrow \$aaEaa \\
&\Rightarrow \$aEaaa \Rightarrow \$Eaaaa \Rightarrow aaaa
\end{aligned}
$$

$L(G) = \{a^i \mid i \text{ is a power of } 2\}$

Grammars for each task

- What is the expressive power of these grammars?

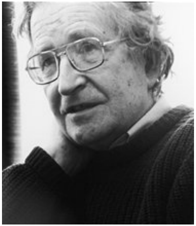
Grammars for each task



Noam Chomsky

- What is the expressive power of these grammars?
- Restricting the types of rules, allows one to describe different aspects of natural languages

Grammars for each task



Noam Chomsky

- What is the expressive power of these grammars?
- Restricting the types of rules, allows one to describe different aspects of natural languages
- These grammars form a hierarchy

Type 0 Grammars

Definition

Type 0 grammars are those where the rules are of the form

$$\alpha \to \beta$$

where $\alpha, \beta \in (\Sigma \cup V)^*$

Example

Consider the grammar G with $\Sigma = \{a\}$ with

$$\begin{array}{lll} S \to \$Ca\# \mid a \mid \epsilon & Ca \to aaC & \$D \to \$C \\ C\# \to D\# \mid E & aD \to Da & aE \to Ea \\ \$E \to \epsilon & & \end{array}$$

Expressive Power of Type 0 Grammars

Theorem

L is recursively enumerable iff there is a type 0 grammar G such that $L = L(G)$.

Expressive Power of Type 0 Grammars

Theorem

L is recursively enumerable iff there is a type 0 grammar G such that $L = L(G)$.

Thus, type 0 grammars are as powerful as Turing machines.

Recognizing Type 0 languages

Proposition

If $G = (V, \Sigma, R, S)$ is a type 0 grammar then $L(G)$ is recursively enumerable.

Recognizing Type 0 languages

Proposition

If $G = (V, \Sigma, R, S)$ is a type 0 grammar then $L(G)$ is recursively enumerable.

Proof.

We will show that $L(G)$ is recognized by a 2-tape non-deterministic Turing machine M, with tape 1 storing the input w, and tape 2 used to construct a derivation of w from S.

Recognizing Type 0 Grammars

Proof (contd).

Recognizing Type 0 Grammars

Proof (contd).

- At any given time tape 2, stores the current string of the derivation; initial tape contains S.

Recognizing Type 0 Grammars

Proof (contd).

- At any given time tape 2, stores the current string of the derivation; initial tape contains S.
- To simulate the next derivation step, M will (nondeterministically) choose a rule to apply, scan from left to right and choose (nondeterministically) a position to apply the rule, replace the substring matching the LHS of the rule with the RHS to get the string at the next step of derivation.

Recognizing Type 0 Grammars

Proof (contd).

- At any given time tape 2, stores the current string of the derivation; initial tape contains S.
- To simulate the next derivation step, M will (nondeterministically) choose a rule to apply, scan from left to right and choose (nondeterministically) a position to apply the rule, replace the substring matching the LHS of the rule with the RHS to get the string at the next step of derivation.
- If tape 2 contains only terminal symbols, then M will check to see if it matches tape 1. If so, the input is accepted, else it is rejected. $\square$

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Proof.

Let M be a Turing machine recognizing L. The grammar G will simulate M "backwards" starting from an accepting configuration.

$\square$

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Proof.

Let M be a Turing machine recognizing L. The grammar G will simulate M "backwards" starting from an accepting configuration.

- A string γ in the derivation will encode a configuration of M

$\square$

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Proof.

Let M be a Turing machine recognizing L. The grammar G will simulate M "backwards" starting from an accepting configuration.

- A string γ in the derivation will encode a configuration of M
- G has rules such that $\gamma_1 \Rightarrow \gamma_2$ iff $\gamma_2 \vdash_M \gamma_1$

□

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Proof.

Let M be a Turing machine recognizing L. The grammar G will simulate M "backwards" starting from an accepting configuration.

- A string γ in the derivation will encode a configuration of M
- G has rules such that $\gamma_1 \Rightarrow \gamma_2$ iff $\gamma_2 \vdash_M \gamma_1$
- The rules of S will generate an accepting configuration of M

$\square$

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Proof.

Let M be a Turing machine recognizing L. The grammar G will simulate M "backwards" starting from an accepting configuration.

- A string γ in the derivation will encode a configuration of M
- G has rules such that $\gamma_1 \Rightarrow \gamma_2$ iff $\gamma_2 \vdash_M \gamma_1$
- The rules of S will generate an accepting configuration of M
- Once (some) initial configuration $q_0 w$ is generated, rules in G will erase symbols to produce the terminal w.

□

Describing R.E. Languages

Proposition

If L is recursively enumerable, then there is a type 0 grammar G such that $L = L(G)$.

Proof.

Let M be a Turing machine recognizing L. The grammar G will simulate M "backwards" starting from an accepting configuration.

- A string γ in the derivation will encode a configuration of M
- G has rules such that $\gamma_1 \Rightarrow \gamma_2$ iff $\gamma_2 \vdash_M \gamma_1$
- The rules of S will generate an accepting configuration of M
- Once (some) initial configuration $q_0 w$ is generated, rules in G will erase symbols to produce the terminal w.

Details in the notes. □

Type 1 Grammars

The rules in a type 1 grammar are of the form

$$\alpha \to \beta$$

where $\alpha, \beta \in (\Sigma \cup V)^*$ and $|\alpha| \leq |\beta|$.

Type 1 Grammars

The rules in a type 1 grammar are of the form

$$\alpha \to \beta$$

where $\alpha, \beta \in (\Sigma \cup V)^*$ and $|\alpha| \leq |\beta|$.
In every derivation, the length of the string never decreases.

Type 1 Grammars

The rules in a type 1 grammar are of the form

$$\alpha \to \beta$$

where $\alpha, \beta \in (\Sigma \cup V)^*$ and $|\alpha| \leq |\beta|$.
In every derivation, the length of the string never decreases.

Example

Consider the grammar G with $\Sigma = \{a, b, c\}$, $V = \{S, B, C, H\}$ and

$S \to aSBC \mid aBC$	$CB \to HB$	$HB \to HC$
$HC \to BC$	$aB \to ab$	$bB \to bb$
$bC \to bc$	$cC \to cc$	

Type 1 Grammars

The rules in a type 1 grammar are of the form

$$\alpha \to \beta$$

where $\alpha, \beta \in (\Sigma \cup V)^*$ and $|\alpha| \leq |\beta|$.
In every derivation, the length of the string never decreases.

Example

Consider the grammar G with $\Sigma = \{a, b, c\}$, $V = \{S, B, C, H\}$ and

$S \to aSBC \mid aBC$	$CB \to HB$	$HB \to HC$
$HC \to BC$	$aB \to ab$	$bB \to bb$
$bC \to bc$	$cC \to cc$	

$L(G) = \{a^n b^n c^n \mid n \geq 0\}$

Context Sensitivity

Normal Form for Type 1 grammars

For every Type 1 grammar G, there is a grammar (in normal form) G' such that $L(G) = L(G')$ and all the rules of G' are of the form

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$

Context Sensitivity

Normal Form for Type 1 grammars

For every Type 1 grammar G, there is a grammar (in normal form) G' such that $L(G) = L(G')$ and all the rules of G' are of the form

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$
So, rules of G' replace a variable A by β in the context $\alpha_1 \Box \alpha_2$.

Context Sensitivity

Normal Form for Type 1 grammars

For every Type 1 grammar G, there is a grammar (in normal form) G' such that $L(G) = L(G')$ and all the rules of G' are of the form

$$\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$
So, rules of G' replace a variable A by β in the context $\alpha_1 \Box \alpha_2$.
Thus, the class of language described by Type 1 grammars are called context-sensitive languages.

Expressive Power of Context Sensitive Languages

What languages can be described by Type 1 grammars?

Expressive Power of Context Sensitive Languages

What languages can be described by Type 1 grammars?

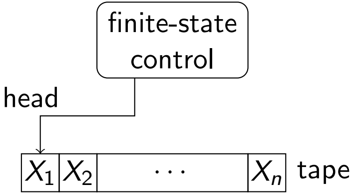
- It turns out to be quite a lot!

Expressive Power of Context Sensitive Languages

What languages can be described by Type 1 grammars?

- It turns out to be quite a lot!
- To say exactly, we need to define a new class of machines . . .

Linear Bounded Automata



Definition

A linear bounded automaton is a restricted Turing machine where the tape head is not permitted to move beyond the portion of the tape containing the input.

- If the machine tries to move the head off either end of the input, the head stays where it is.

LBA and Type 1 Grammars

Theorem

If G is a Type 1 grammar then there is a linear bounded automaton M such that $L(G) = L(M)$.

LBA and Type 1 Grammars

Theorem

If G is a Type 1 grammar then there is a linear bounded automaton M such that $L(G) = L(M)$.
If M is a linear bounded automaton then there is a Type 1 grammar G such that $L(M) = L(G)$.

LBA and Type 1 Grammars

Theorem

If G is a Type 1 grammar then there is a linear bounded automaton M such that $L(G) = L(M)$.
If M is a linear bounded automaton then there is a Type 1 grammar G such that $L(M) = L(G)$.

Proof.

Translations between TMs and Type 0 grammars, when carried out on Type 1 grammars and LBAs, prove this theorem. $\square$

Decidability of LBAs

Theorem

If M is a linear bounded automaton, then $L(M)$ is decidable.

Proof.

- The number of configurations of M on an input of length n is at most snt^n, where s is the number of states of M and t is the size of the tape alphabet

Decidability of LBAs

Theorem

If M is a linear bounded automaton, then $L(M)$ is decidable.

Proof.

- The number of configurations of M on an input of length n is at most snt^n, where s is the number of states of M and t is the size of the tape alphabet
- If M accepts w of length n then M does so within snt^n steps.

Decidability of LBAs

Theorem

If M is a linear bounded automaton, then $L(M)$ is decidable.

Proof.

- The number of configurations of M on an input of length n is at most snt^n, where s is the number of states of M and t is the size of the tape alphabet
- If M accepts w of length n then M does so within snt^n steps.
  - Any computation of length more than snt^n is "cycling" and so cannot accept w

Decidability of LBAs

Theorem

If M is a linear bounded automaton, then $L(M)$ is decidable.

Proof.

- The number of configurations of M on an input of length n is at most snt^n, where s is the number of states of M and t is the size of the tape alphabet
  - Any configuration is state + head position + contents of the tape. The observation follows since the tape has at most n symbols.
- If M accepts w of length n then M does so within snt^n steps.
  - Any computation of length more than snt^n is "cycling" and so cannot accept w

Decidability of LBAs

Proof (contd).

Consider the following TM D that always halts and decides $L(M)$

```
On input w
    Run M on w for s|w|t^|w| steps
    If M accepts w then accept else reject
```

∎

Model for Decidability?

Do LBAs recognize all decidable languages?

Model for Decidability?

Do LBAs recognize all decidable languages?

- LBAs recognize many but not all decidable languages.

Model for Decidability?

Do LBAs recognize all decidable languages?

- LBAs recognize many but not all decidable languages.
- Decidable languages not recognized by LBAs can be found by diagonalization.

Diagonal LBA Language

Recall that every LBA can be coded as a binary string, and every binary string can be thought of as an LBA. We now consider only LBAs whose input alphabet is $\{0, 1\}$.

Theorem

$L_{d,\mathrm{LBA}} = \{M \mid M$ *is a LBA and* $M \notin L(M)\}$ *is decidable but not context sensitive, i.e., recognized by an LBA.*

$L_{d,\mathrm{LBA}}$ is decidable

$L_{d,\mathrm{LBA}}$ is decidable

The following program M decides $L_{d,\mathrm{LBA}}$

```
On input x
    Check if x accepts x
    If x accepts x then reject else accept
```

$L_{d,\mathrm{LBA}}$ is decidable

The following program M decides $L_{d,\mathrm{LBA}}$

```
On input x
    Check if x accepts x
    If x accepts x then reject else accept
```

Since languages recognized by LBAs are decidable, the step to check if x accepts x will halt.

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.
- Now, if $M \in L_{d,\mathrm{LBA}} = L(M)$

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.
- Now, if $M \in L_{d,\mathrm{LBA}} = L(M)$ then M is accepted by M,

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.
- Now, if $M \in L_{d,\mathrm{LBA}} = L(M)$ then M is accepted by M, which means $M \notin L_{d,\mathrm{LBA}}$!

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.
- Now, if $M \in L_{d,\mathrm{LBA}} = L(M)$ then M is accepted by M, which means $M \notin L_{d,\mathrm{LBA}}$!
- Conversely, if $M \notin L_{d,\mathrm{LBA}} = L(M)$

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.
- Now, if $M \in L_{d,\mathrm{LBA}} = L(M)$ then M is accepted by M, which means $M \notin L_{d,\mathrm{LBA}}$!
- Conversely, if $M \notin L_{d,\mathrm{LBA}} = L(M)$ then M is not accepted by M

$L_{d,\mathrm{LBA}}$ is not context sensitive

- Suppose $L_{d,\mathrm{LBA}}$ were recognized by a LBA, say M.
- Now, if $M \in L_{d,\mathrm{LBA}} = L(M)$ then M is accepted by M, which means $M \notin L_{d,\mathrm{LBA}}$!
- Conversely, if $M \notin L_{d,\mathrm{LBA}} = L(M)$ then M is not accepted by M which means $M \in L_{d,\mathrm{LBA}}$!

Type 3 Grammars

The rules in a type 3 grammar are of the form

$$A \to aB \qquad \text{or} \qquad A \to a$$

where $A, B \in V$ and $a \in \Sigma \cup \{\epsilon\}$.

Type 3 Grammars

The rules in a type 3 grammar are of the form

$$A \to aB \qquad \text{or} \qquad A \to a$$

where $A, B \in V$ and $a \in \Sigma \cup \{\epsilon\}$.

Example

Consider the grammar over $\Sigma = \{0, 1\}$ with rules

$$S \to 1S \mid 0A \qquad A \to \epsilon \mid 1A \mid 0S$$

Type 3 Grammars

The rules in a type 3 grammar are of the form

$$A \to aB \qquad \text{or} \qquad A \to a$$

where $A, B \in V$ and $a \in \Sigma \cup \{\epsilon\}$.

Example

Consider the grammar over $\Sigma = \{0, 1\}$ with rules

$$S \to 1S \mid 0A \qquad A \to \epsilon \mid 1A \mid 0S$$

$L(G) = \{w \in \{0, 1\}^* \mid w \text{ has an odd number of 0s}\}$

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Proof.

Let $G = (V, \Sigma, R, S)$ be a type 3 grammar. Consider the NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

$\cdot\cdot\longrightarrow$

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Proof.

Let $G = (V, \Sigma, R, S)$ be a type 3 grammar. Consider the NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = V \cup \{q_F\}$, where $q_F \notin V$

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Proof.

Let $G = (V, \Sigma, R, S)$ be a type 3 grammar. Consider the NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = V \cup \{q_F\}$, where $q_F \notin V$
- $q_0 = S$

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Proof.

Let $G = (V, \Sigma, R, S)$ be a type 3 grammar. Consider the NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = V \cup \{q_F\}$, where $q_F \notin V$
- $q_0 = S$
- $F = \{q_F\}$

$\cdot\cdot\longrightarrow$

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Proof.

Let $G = (V, \Sigma, R, S)$ be a type 3 grammar. Consider the NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = V \cup \{q_F\}$, where $q_F \notin V$
- $q_0 = S$
- $F = \{q_F\}$
- $\delta(A, a) = \{B \mid \text{if } A \to aB \in R\} \cup \{q_F \mid \text{if } A \to a \in R\}$ for $A \in V$. And $\delta(q_F, a) = \emptyset$ for all a.

Type 3 Grammars and Regularity

Proposition

L is regular iff there is a Type 3 grammar G such that $L = L(G)$.

Proof.

Let $G = (V, \Sigma, R, S)$ be a type 3 grammar. Consider the NFA $M = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = V \cup \{q_F\}$, where $q_F \notin V$
- $q_0 = S$
- $F = \{q_F\}$
- $\delta(A, a) = \{B \mid \text{if } A \rightarrow aB \in R\} \cup \{q_F \mid \text{if } A \rightarrow a \in R\}$ for $A \in V$. And $\delta(q_F, a) = \emptyset$ for all a.

$L(M) = L(G)$ as $\forall A \in V$, $\forall w \in \Sigma^*$, $A \stackrel{*}{\Rightarrow}_G w$ iff $q_F \in \hat{\Delta}(A, w)$.

Type 3 Grammars and Regularity

NFA to Grammars

Proof (contd).

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing L. Consider $G = (V, \Sigma, R, S)$ where

$\square$

Type 3 Grammars and Regularity

NFA to Grammars

Proof (contd).

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing L. Consider $G = (V, \Sigma, R, S)$ where

- $V = Q$

$\square$

Type 3 Grammars and Regularity

NFA to Grammars

Proof (contd).

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing L. Consider $G = (V, \Sigma, R, S)$ where

- $V = Q$
- $S = q_0$

$\square$

Type 3 Grammars and Regularity

NFA to Grammars

Proof (contd).

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing L. Consider $G = (V, \Sigma, R, S)$ where

- $V = Q$
- $S = q_0$
- $q_1 \rightarrow aq_2 \in R$ iff $q_2 \in \delta(q_1, a)$ and $q \rightarrow \epsilon \in R$ iff $q \in F$.

□

Type 3 Grammars and Regularity

NFA to Grammars

Proof (contd).

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing L. Consider $G = (V, \Sigma, R, S)$ where

- $V = Q$
- $S = q_0$
- $q_1 \to aq_2 \in R$ iff $q_2 \in \delta(q_1, a)$ and $q \to \epsilon \in R$ iff $q \in F$.

We can show, for any $q, q' \in Q$ and $w \in \Sigma^*$, $q' \in \hat{\Delta}(q, w)$ iff $q \overset{*}{\Rightarrow}_G wq'$. Thus, $L(M) = L(G)$. $\square$

Type 2 Grammars

The rules in a type 2 grammar are of the form

$$A \rightarrow \beta$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$.

Type 2 Grammars

The rules in a type 2 grammar are of the form

$$A \to \beta$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$.
Type 2 grammars describe context-free languages, which we will study next in this class.

Type 2 Grammars

The rules in a type 2 grammar are of the form

$$A \to \beta$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$.
Type 2 grammars describe context-free languages, which we will study next in this class.

Example

Consider G over $\Sigma = \{0, 1\}$ with rules

$$S \to \epsilon \mid 0S1$$

Type 2 Grammars

The rules in a type 2 grammar are of the form

$$A \to \beta$$

where $A \in V$ and $\beta \in (\Sigma \cup V)^*$.
Type 2 grammars describe context-free languages, which we will study next in this class.

Example

Consider G over $\Sigma = \{0, 1\}$ with rules

$$S \to \epsilon \mid 0S1$$

$L(G) = \{0^n 1^n \mid n \geq 0\}$

Grammars and their Languages

Grammar	Rules	Languages
Type 3	$A \to aB$ or $A \to a$	Regular
Type 2	$A \to \alpha$	Context Free
Type 1	$\alpha \to \beta$ with $\lvert\alpha\rvert \leq \lvert\beta\rvert$	Context Sensitive
Type 0	$\alpha \to \beta$	Recursively Enumerable

In the above table, $\alpha, \beta \in (\Sigma \cup V)^*$, $A, B \in V$ and $a \in \Sigma \cup \{\epsilon\}$.

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Proof.

Clearly a Type 3 grammar is a special Type 2 grammar, a Type 2 grammar is a special Type 1 grammar, and a Type 1 grammar is special Type 0 grammar.

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Proof.

Clearly a Type 3 grammar is a special Type 2 grammar, a Type 2 grammar is a special Type 1 grammar, and a Type 1 grammar is special Type 0 grammar.
Moreover, there is a language that has a Type 2 grammar but no Type 3 grammar

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Proof.

Clearly a Type 3 grammar is a special Type 2 grammar, a Type 2 grammar is a special Type 1 grammar, and a Type 1 grammar is special Type 0 grammar.
Moreover, there is a language that has a Type 2 grammar but no Type 3 grammar ($L = \{0^n 1^n \mid n \geq 0\}$),

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Proof.

Clearly a Type 3 grammar is a special Type 2 grammar, a Type 2 grammar is a special Type 1 grammar, and a Type 1 grammar is special Type 0 grammar.
Moreover, there is a language that has a Type 2 grammar but no Type 3 grammar ($L = \{0^n 1^n \mid n \geq 0\}$), a language that has a Type 1 grammar but no Type 2 grammar

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Proof.

Clearly a Type 3 grammar is a special Type 2 grammar, a Type 2 grammar is a special Type 1 grammar, and a Type 1 grammar is special Type 0 grammar.
Moreover, there is a language that has a Type 2 grammar but no Type 3 grammar ($L = \{0^n 1^n \mid n \geq 0\}$), a language that has a Type 1 grammar but no Type 2 grammar ($L = \{a^n b^n c^n \mid n \geq 0\}$),

Chomsky Hierarchy

Theorem

Type 0, Type 1, Type 2, and Type 3 grammars define a strict *hierarchy of formal languages.*

Proof.

Clearly a Type 3 grammar is a special Type 2 grammar, a Type 2 grammar is a special Type 1 grammar, and a Type 1 grammar is special Type 0 grammar.
Moreover, there is a language that has a Type 2 grammar but no Type 3 grammar ($L = \{0^n 1^n \mid n \geq 0\}$), a language that has a Type 1 grammar but no Type 2 grammar ($L = \{a^n b^n c^n \mid n \geq 0\}$), and a language with a Type 0 grammar but no Type 1 grammar. $\square$

Overview of Languages