Undec Problems about CFG's

April 25, 2024

Goals

Def If G is a CFG then $L(G)$ is the language that G generates.

Goals

Def If G is a CFG then $L(G)$ is the language that G generates.

We will do the following:

Goals

Def If G is a CFG then $L(G)$ is the language that G generates.

We will do the following:

1. Show that the following problem is undec:
   Given a CFG G, determine if $L(G) = \Sigma^*$

Goals

Def If G is a CFG then $L(G)$ is the language that G generates.

We will do the following:

1. Show that the following problem is undec:
   Given a CFG G, determine if $L(G) = \Sigma^*$
   (We denote this problem $\mathbf{CFG\Sigma^*}$.)

Goals

Def If G is a CFG then $L(G)$ is the language that G generates.

We will do the following:

1. Show that the following problem is undec:
   Given a CFG G, determine if $L(G) = \Sigma^*$
   (We denote this problem $\mathbf{CFG\Sigma^*}$.)
2. Discuss the exact complexity of that problem.

Goals

Def If G is a CFG then $L(G)$ is the language that G generates.

We will do the following:

1. Show that the following problem is undec:
   Given a CFG G, determine if $L(G) = \Sigma^*$
   (We denote this problem $\mathbf{CFG\Sigma^*}$.)
2. Discuss the exact complexity of that problem.
3. Discuss the following problem: Given a CFG G of size n such that $L(G)$ is regular, bound the size of the DFA for $L(G)$.

The Problem $\mathbf{CFG\Sigma^*}$

April 25, 2024

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

$$\#x(s, \#)\# \cdots \#$$

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

$$\#x(s,\#)\#\cdots\#$$

2) If a string is accepted the final config is

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

$$\#x(s,\#)\#\cdots\#$$

2) If a string is accepted the final config is

$$\#(h,Y)\#\cdots\#$$

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

$$\#x(s, \#)\# \cdots \#$$

2) If a string is accepted the final config is

$$\#(h, Y)\# \cdots \#$$

3) Let C and D be configs.

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

$$\#x(s, \#)\# \cdots \#$$

2) If a string is accepted the final config is

$$\#(h, Y)\# \cdots \#$$

3) Let C and D be configs.
$C \vdash D$ means from C the TM **goes to** D.

Conventions for TM

(This differs from the convention used for the Cook-Levin Thm)

1) Assume the TM M has start state s.

Input x. Head of TM is just to the right of x. Initial Config:

$$\#x(s, \#)\# \cdots \#$$

2) If a string is accepted the final config is

$$\#(h, Y)\# \cdots \#$$

3) Let C and D be configs.
$C \vdash D$ means from C the TM **goes to** D.
$C \not\vdash D$ means from C the TM **does not go to** D.

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse.

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse. $aaba^R = abaa$.

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse. $aaba^R = abaa$.

Let $e, x \in \mathsf{N}$. Consider Turing Machine M_e.

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse. $aaba^R = abaa$.

Let $e, x \in \mathbb{N}$. Consider Turing Machine M_e.
Def $\mathrm{ACC}_{e,x}$ is the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

such that

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse. $aaba^R = abaa$.

Let $e, x \in \mathbb{N}$. Consider Turing Machine M_e.
Def $\mathrm{ACC}_{e,x}$ is the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

such that

- $|C_1| = |C_2| = \cdots = |C_s|$.

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse. $aaba^R = abaa$.

Let $e, x \in \mathbb{N}$. Consider Turing Machine M_e.

Def $\mathrm{ACC}_{e,x}$ is the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

such that

- $|C_1| = |C_2| = \cdots = |C_s|$.
- $C_1, C_2, \ldots, C_s$ represents an accepting computation of $M_e(x)$.

Connect TM's to CFG's

Recall If $w \in \Sigma^*$ then w^R is the reverse. $aaba^R = abaa$.

Let $e, x \in \mathbb{N}$. Consider Turing Machine M_e.
Def $\mathrm{ACC}_{e,x}$ is the set of all sequences of config's represented by

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

such that

- $|C_1| = |C_2| = \cdots = |C_s|$.
- $C_1, C_2, \ldots, C_s$ represents an accepting computation of $M_e(x)$.
- We will later see why we do this funny thing with reversals.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.
4. $w \in \Sigma^*\$C\$D\$\Sigma^*$ where $C, D \in \{a, b, \#\}^*$ and $|C| \neq |D|$.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.
4. $w \in \Sigma^*\$C\$D\$\Sigma^*$ where $C, D \in \{a, b, \#\}^*$ and $|C| \neq |D|$.
   Two configs of different lengths.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.
4. $w \in \Sigma^*\$C\$D\$\Sigma^*$ where $C, D \in \{a, b, \#\}^*$ and $|C| \neq |D|$.
   Two configs of different lengths. HW CFL.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.
4. $w \in \Sigma^*\$C\$D\$\Sigma^*$ where $C, D \in \{a, b, \#\}^*$ and $|C| \neq |D|$.
   Two configs of different lengths. HW CFL.
5. $w \in \Sigma^*\$C\$D^R\$\Sigma^*$ where $C \not\vdash D$.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.
4. $w \in \Sigma^*\$C\$D\$\Sigma^*$ where $C, D \in \{a, b, \#\}^*$ and $|C| \neq |D|$.
   Two configs of different lengths. HW CFL.
5. $w \in \Sigma^*\$C\$D^R\$\Sigma^*$ where $C \not\vdash D$.
   Sequence is not a valid computation.

$\overline{\mathrm{ACC}_{e,x}}$ is a CFG!

M_e's alphabet: $\{a, b, Y, N, \#\}$. Y and N only used in final config.
Our CFG will use alphabet
$\Sigma = \{a, b, Y, N, \#, \$\} \cup Q \times \{a, b, Y, N, \#\}$.
If $w \notin \mathrm{ACC}_{e,x}$ then one of the following happens:

1. w's prefix **is not** $\#x(s, \#)\#^*\$$.
   Initial config is not what you get if the input is x. Regular.
2. w's suffix **is not** $\$\#(h, Y)\#^*\$$.
   Final configuration does not accepts. Regular.
3. $w \in \Sigma^*\{Y, N\}\Sigma^*\$\Sigma^*\$$.
   Y or N appears before final configuration. Regular.
4. $w \in \Sigma^*\$C\$D\$\Sigma^*$ where $C, D \in \{a, b, \#\}^*$ and $|C| \neq |D|$.
   Two configs of different lengths. HW CFL.
5. $w \in \Sigma^*\$C\$D^R\$\Sigma^*$ where $C \not\vdash D$.
   Sequence is not a valid computation. In these slides.

An Example

Want CFG that accepts a string with $\$C\$D^R\$$ where $C \not\vdash D$.

An Example

Want CFG that accepts a string with $\$C\$D^R\$$ where $C \not\vdash D$.
If $\delta(q, b) = (p, a)$

ρ	(q, b)	η
ρ	(p, a)	η

An Example

Want CFG that accepts a string with $\$C\$D^R\$$ where $C \not\vdash D$.
If $\delta(q, b) = (p, a)$

ρ	(q, b)	η
ρ	(p, a)	η

We want a CFG that will generate a string where the (q, b) in C does not lead to a (p, a). For all $\sigma \neq (p, a)$ we produce a CFG that will put σ in the right place.

An Example

Want CFG that accepts a string with $\$C\$D^R\$$ where $C \not\vdash D$.
If $\delta(q, b) = (p, a)$

ρ	(q, b)	η
ρ	(p, a)	η

We want a CFG that will generate a string where the (q, b) in C does not lead to a (p, a). For all $\sigma \neq (p, a)$ we produce a CFG that will put σ in the right place.
We use I to denote the instruction $\delta(q, b) = (p, a)$

An Example

Want CFG that accepts a string with $\$C\$D^R\$$ where $C \not\vdash D$.
If $\delta(q, b) = (p, a)$

ρ	(q, b)	η
ρ	(p, a)	η

We want a CFG that will generate a string where the (q, b) in C does not lead to a (p, a). For all $\sigma \neq (p, a)$ we produce a CFG that will put σ in the right place.
We use I to denote the instruction $\delta(q, b) = (p, a)$

Continued on the next slides.

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We first generate the substrings.

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We first generate the substrings.
$S \rightarrow (q, b)T\sigma$

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$ \cdots \$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We first generate the substrings.
$S \rightarrow (q, b)T\sigma$
$T \rightarrow \tau T \tau$ for all $\tau \in \{a, b, \#\}$.

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We first generate the substrings.
$S \rightarrow (q, b)T\sigma$
$T \rightarrow \tau T \tau$ for all $\tau \in \{a, b, \#\}$.
$T \rightarrow \$$

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We first generate the substrings.
$S \rightarrow (q, b)T\sigma$
$T \rightarrow \tau T \tau \quad$ for all $\tau \in \{a, b, \#\}$.
$T \rightarrow \$$

We call this grammar $G_{I,\sigma}$.

We use that Weird $\$C_1\$C_2^R\$$ Thing

Recall that our strings are of the form:

$$\$C_1\$C_2^R\$C_3\$C_4^R\$\cdots\$C_s^R\$$$

Let $\sigma \neq (p, a)$. We want strings that have this substring:

$(q, b)w_1\$w_2\sigma$ where $|w_1| = |w_2|$ This is where use funny R thing.

We first generate the substrings.
$S \rightarrow (q, b)T\sigma$
$T \rightarrow \tau T \tau$ for all $\tau \in \{a, b, \#\}$.
$T \rightarrow \$$

We call this grammar $G_{I,\sigma}$.
Next slide to finish this up.

Final CFG for this one instruction

$\delta(q, b) = (p, a)$. Recall that this is instruction I.

Final CFG for this one instruction

$\delta(q, b) = (p, a)$. Recall that this is instruction I.

ρ	(q, b)	η
ρ	(p, a)	η

Final CFG for this one instruction

$\delta(q, b) = (p, a)$. Recall that this is instruction I.

ρ	(q, b)	η
ρ	(p, a)	η

For every $\sigma \neq (p, a)$ we have grammar $G_{I,\sigma}$.

Final CFG for this one instruction

$\delta(q, b) = (p, a)$. Recall that this is instruction I.

ρ	(q, b)	η
ρ	(p, a)	η

For every $\sigma \neq (p, a)$ we have grammar $G_{I,\sigma}$.
Let $G^1_{I,\sigma}$ be the CFG for $\Sigma^* L(G_{I,\sigma})\Sigma^*$.

Final CFG for this one instruction

$\delta(q, b) = (p, a)$. Recall that this is instruction I.

ρ	(q, b)	η
ρ	(p, a)	η

For every $\sigma \neq (p, a)$ we have grammar $G_{I,\sigma}$.
Let $G^1_{I,\sigma}$ be the CFG for $\Sigma^* L(G_{I,\sigma})\Sigma^*$.
Let G_I be the CFG for $\bigcup_{\sigma \neq (p,a)} L(G^1_{I,\sigma})$.

Final CFG for this one instruction

$\delta(q,b) = (p,a)$. Recall that this is instruction I.

ρ	(q,b)	η
ρ	(p,a)	η

For every $\sigma \neq (p,a)$ we have grammar $G_{I,\sigma}$.
Let $G^1_{I,\sigma}$ be the CFG for $\Sigma^* L(G_{I,\sigma})\Sigma^*$.
Let G_I be the CFG for $\bigcup_{\sigma \neq (p,a)} L(G^1_{I,\sigma})$.

What about the other instructions?

Final CFG for this one instruction

$\delta(q, b) = (p, a)$. Recall that this is instruction I.

ρ	(q, b)	η
ρ	(p, a)	η

For every $\sigma \neq (p, a)$ we have grammar $G_{I,\sigma}$.
Let $G^1_{I,\sigma}$ be the CFG for $\Sigma^* L(G_{I,\sigma})\Sigma^*$.
Let G_I be the CFG for $\bigcup_{\sigma \neq (p,a)} L(G^1_{I,\sigma})$.

What about the other instructions?
Next slide

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, …, G_{I_m}. (HW)

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, …, G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, …, G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

$$\bigcup_{i=1}^{m} G_{I_i}$$

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, ..., G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

$$\bigcup_{i=1}^{m} G_{I_i}$$

Three points about G_I for any instruction I.

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, ..., G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

$$\bigcup_{i=1}^{m} G_{I_i}$$

Three points about G_I for any instruction I.

1. G_I will generates all sequences of configs which have adjacent C and D that should use instruction I but do not.

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, …, G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

$$\bigcup_{i=1}^{m} G_{I_i}$$

Three points about G_I for any instruction I.

1. G_I will generates all sequences of configs which have adjacent C and D that should use instruction I but do not.
2. G_I will generate **many other strings** that are in $\overline{ACC_{e,x}}$.

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, …, G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

$$\bigcup_{i=1}^{m} G_{I_i}$$

Three points about G_I for any instruction I.

1. G_I will generates all sequences of configs which have adjacent C and D that should use instruction I but do not.
2. G_I will generate **many other strings** that are in $\overline{ACC_{e,x}}$.Thats fine.

Final CFG for all instruction

Let the instructions be $I_1, \ldots, I_m$.

By similar methods you can get G_{I_2}, G_{I_3}, ..., G_{I_m}. (HW)

SO our final grammar for $C \not\vdash D$ is

$$\bigcup_{i=1}^{m} G_{I_i}$$

Three points about G_I for any instruction I.

1. G_I will generates all sequences of configs which have adjacent C and D that should use instruction I but do not.
2. G_I will generate **many other strings** that are in $\overline{ACC_{e,x}}$.Thats fine.
3. We are not quite done yet. Next slide.

Another Way for $C \not\vdash D$

$\delta(q, b) = (p, a)$.

Another Way for $C \not\vdash D$

$\delta(q, b) = (p, a)$.

a	a	b	b	(q, b)
b	a	b	b	(p, a)

Another Way for $C \nvdash D$

$\delta(q, b) = (p, a)$.

a	a	b	b	(q, b)
b	a	b	b	(p, a)

Its possible that around the head it looks like $C \vdash D$ but away from the head is where you see $C \nvdash D$.

Another Way for $C \not\vdash D$

$\delta(q,b) = (p,a)$.

a	a	b	b	(q,b)
b	a	b	b	(p,a)

Its possible that around the head it looks like $C \vdash D$ but away from the head is where you see $C \not\vdash D$.

A CFG for this case is similar to $G_{I,\sigma}$. We omit it. (HW)

WAKE UP. No more Low Level TM Stuff

The last few slides established the following:

WAKE UP. No more Low Level TM Stuff

The last few slides established the following:

$\exists$ an algorithm: given e, x, create a CFG G such that

$$L(G) = \overline{ACC_{e,x}}$$

WAKE UP. No more Low Level TM Stuff

The last few slides established the following:

$\exists$ an algorithm: given e, x, create a CFG G such that

$$L(G) = \overline{ACC_{e,x}}$$

We use this algorithm and to not need to know its details.

$\mathrm{HALT} \leq_T \mathrm{CFG}\Sigma^*$: What does it Mean

Recall $\mathrm{CFG}\Sigma^*$: Given a CFG G determine if $L(G) = \Sigma^*$.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$: What does it Mean

Recall $\mathrm{CFG}\Sigma^*$: Given a CFG G determine if $L(G) = \Sigma^*$.

On the next slide we will present an algorithm for HALT that makes calls to $\mathrm{CFG}\Sigma^*$

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$: What does it Mean

Recall $\mathrm{CFG}\Sigma^*$: Given a CFG G determine if $L(G) = \Sigma^*$.

On the next slide we will present an algorithm for HALT that makes calls to $\mathrm{CFG}\Sigma^*$

We denote this $\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$.

$\mathrm{HALT} \leq_T \mathrm{CFG}\Sigma^*$: What does it Mean

Recall $\mathrm{CFG}\Sigma^*$: Given a CFG G determine if $L(G) = \Sigma^*$.

On the next slide we will present an algorithm for HALT that makes calls to $\mathrm{CFG}\Sigma^*$

We denote this $\mathrm{HALT} \leq_T \mathrm{CFG}\Sigma^*$.

We will not define $\leq_T$ formally.

$\mathrm{HALT} \leq_T \mathrm{CFG}\Sigma^*$: What does it Mean

Recall $\mathrm{CFG}\Sigma^*$: Given a CFG G determine if $L(G) = \Sigma^*$.

On the next slide we will present an algorithm for HALT that makes calls to $\mathrm{CFG}\Sigma^*$

We denote this $\mathrm{HALT} \leq_T \mathrm{CFG}\Sigma^*$.

We will not define $\leq_T$ formally.

The T stands for Turing.

$\mathbf{HALT} \leq_T \mathbf{CFG\Sigma^*}$

HALT $\le_T$ CFGΣ^*

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

HALT $\le_T$ CFGΣ^*

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \neq \Sigma^*$.

HALT $\le_T$ CFGΣ^*

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \neq \Sigma^*$.

$(e, x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \neq \Sigma^*$.

$(e, x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

Thm $L(G) = \Sigma^*$ is undec.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e,x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \ne \Sigma^*$.

$(e,x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

Thm $L(G) = \Sigma^*$ is undec.
Assume, BWOC that $L(G) = \Sigma^*$ is dec. Can use to solve HALT.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \neq \Sigma^*$.

$(e, x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

Thm $L(G) = \Sigma^*$ is undec.
Assume, BWOC that $L(G) = \Sigma^*$ is dec. Can use to solve HALT.
Given e, x, create CFG G for $\overline{ACC_{e,x}}$.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e,x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \ne \Sigma^*$.

$(e,x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

Thm $L(G) = \Sigma^*$ is undec.
Assume, BWOC that $L(G) = \Sigma^*$ is dec. Can use to solve HALT.
Given e, x, create CFG G for $\overline{ACC_{e,x}}$.
Test if $L(G) = \Sigma^*$.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \ne \Sigma^*$.

$(e, x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

Thm $L(G) = \Sigma^*$ is undec.
Assume, BWOC that $L(G) = \Sigma^*$ is dec. Can use to solve HALT.
Given e, x, create CFG G for $\overline{ACC_{e,x}}$.
Test if $L(G) = \Sigma^*$.
If NO then $(e, x) \in \mathrm{HALT}$.

$\mathrm{HALT} \le_T \mathrm{CFG}\Sigma^*$

Recall $\exists$ algorithm: given e, x, produce CFG for $\overline{ACC_{e,x}}$.

$(e,x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1 \rightarrow \overline{ACC_{e,x}} \ne \Sigma^*$.

$(e,x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset \rightarrow \overline{ACC_{e,x}} = \Sigma^*$.

Thm $L(G) = \Sigma^*$ is undec.
Assume, BWOC that $L(G) = \Sigma^*$ is dec. Can use to solve HALT.
Given e, x, create CFG G for $\overline{ACC_{e,x}}$.
Test if $L(G) = \Sigma^*$.
If NO then $(e,x) \in \mathrm{HALT}$.
If YES then $(e,x) \notin \mathrm{HALT}$.

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \leq_T \mathbf{CFG\Sigma^*}$

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

How Hard is $\mathrm{CFG}\Sigma^*$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

Vote

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

Vote

1. Yes and this is known and people care.

How Hard is $\mathrm{CFG}\Sigma^*$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG}\Sigma^*$

Is the following true? $\mathbf{CFG}\Sigma^* \le_T \mathbf{HALT}$

Vote

1. Yes and this is known and people care.
2. No and this is known and people care.

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

Vote

1. Yes and this is known and people care.
2. No and this is known and people care.
3. Only Bill cares and its unknown.

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

Vote

1. Yes and this is known and people care.
2. No and this is known and people care.
3. Only Bill cares and its unknown.
4. Only Bill cares and he showed YES.

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

Vote

1. Yes and this is known and people care.
2. No and this is known and people care.
3. Only Bill cares and its unknown.
4. Only Bill cares and he showed YES.
5. Only Bill cares and he showed NO.

How Hard is $\mathbf{CFG\Sigma^*}$?

Valiant (1976) proved $\mathbf{HALT} \le_T \mathbf{CFG\Sigma^*}$

Is the following true? $\mathbf{CFG\Sigma^*} \le_T \mathbf{HALT}$

Vote

1. Yes and this is known and people care.
2. No and this is known and people care.
3. Only Bill cares and its unknown.
4. Only Bill cares and he showed YES.
5. Only Bill cares and he showed NO.

Answer on next slide.

Only Bill cares and he showed NO

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

$$\mathrm{INF} = \{e : (\forall y)(\exists x \geq y)(\exists s)[M_{e,s}(x) \downarrow]\}.$$

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

$$\mathrm{INF} = \{e : (\forall y)(\exists x \geq y)(\exists s)[M_{e,s}(x) \downarrow]\}.$$

Known $\mathrm{HALT} <_T \mathrm{INF}$.

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

$$\mathrm{INF} = \{e : (\forall y)(\exists x \geq y)(\exists s)[M_{e,s}(x)\downarrow]\}.$$

Known $\mathrm{HALT} <_T \mathrm{INF}$.
Hence $\mathrm{HALT} <_T \mathrm{CFG}\Sigma^*$.

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

$$\mathrm{INF} = \{e : (\forall y)(\exists x \geq y)(\exists s)[M_{e,s}(x) \downarrow]\}.$$

Known $\mathrm{HALT} <_T \mathrm{INF}$.
Hence $\mathrm{HALT} <_T \mathrm{CFG}\Sigma^*$.

How do we know nobody else cares?

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

$$\mathrm{INF} = \{e : (\forall y)(\exists x \geq y)(\exists s)[M_{e,s}(x) \downarrow]\}.$$

Known $\mathrm{HALT} <_T \mathrm{INF}$.
Hence $\mathrm{HALT} <_T \mathrm{CFG}\Sigma^*$.

How do we know nobody else cares?

Valiant's paper was 1976. Bill's was 2015.

Only Bill cares and he showed NO

Bill (2015) showed NO. Bill showed that

$$\mathrm{CFG}\Sigma^* \equiv_T \mathrm{INF}$$

$$\mathrm{INF} = \{e : (\forall y)(\exists x \ge y)(\exists s)[M_{e,s}(x) \downarrow]\}.$$

Known $\mathrm{HALT} <_T \mathrm{INF}$.
Hence $\mathrm{HALT} <_T \mathrm{CFG}\Sigma^*$.

How do we know nobody else cares?

Valiant's paper was 1976. Bill's was 2015.

So nobody worked on it between 1976 and 2014.

Bounding Functions

April 25, 2024

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $\mathbf{(DFA, CFG)}$** is a function f such that the following holds:

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $(\mathbf{DFA}, \mathbf{CFG})$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $(\mathrm{DFA}, \mathrm{CFG})$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $\mathbf{(DFA, CFG)}$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $\mathbf{(DFA, CFG)}$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Vote

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $(\mathbf{DFA}, \mathbf{CFG})$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Vote

1. $(|G| \le n \wedge L(G)\ \mathrm{Reg}\ ) \rightarrow (\exists M, |M| \le \mathbf{2^n})[L(M) = L(G)]$.

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $\mathbf{(DFA, CFG)}$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Vote

1. $(|G| \le n \wedge L(G) \text{ Reg }) \rightarrow (\exists M, |M| \le \mathbf{2^n})[L(M) = L(G)]$.
2. $(|G| \le n \wedge L(G) \text{ Reg }) \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for** $\mathbf{(DFA, CFG)}$ is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Vote

1. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{2^n})[L(M) = L(G)]$.
2. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.
3. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{ACK(n)})[L(M) = L(G)]$.

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $(\mathbf{DFA, CFG})$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Vote

1. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{2^n})[L(M) = L(G)]$.
2. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.
3. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{ACK(n)})[L(M) = L(G)]$.
4. There is no computable f such that $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{f(n)})[L(M) = L(G)]$.

Bounding Function

G is CFG, $|G|$ is size, M is DFA, $|M|$ is numb of states.

A **bounding function for $(\mathbf{DFA}, \mathbf{CFG})$** is a function f such that the following holds:

$(\forall n)(\forall G)[$

$$(|G| \le n \wedge L(G) \in \mathrm{REG}) \rightarrow (\exists \mathrm{DFA}\ M)[L(G) = L(M) \wedge |M| \le f(n)]$$

$]$

Vote

1. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{2^n})[L(M) = L(G)]$.
2. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.
3. $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{ACK(n)})[L(M) = L(G)]$.
4. There is no computable f such that $(|G| \le n \wedge L(G)$ Reg $) \rightarrow (\exists M, |M| \le \mathbf{f(n)})[L(M) = L(G)]$.

Answer on the next slide.

Bdding Funct for $(\mathbf{DFA}, \mathbf{CFG})$ is Not Computable

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Computable

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1.$

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Computable

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1$.

$\mathrm{ACC}_{e,x}$ is **regular** so $\overline{\mathrm{ACC}_{e,x}}$ is regular.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Computable

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1$.

$\mathrm{ACC}_{e,x}$ is **regular** so $\overline{\mathrm{ACC}_{e,x}}$ is regular.

$(e, x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Computable

$(e, x) \in \mathrm{HALT} \rightarrow |\mathrm{ACC}_{e,x}| = 1$.

$\mathrm{ACC}_{e,x}$ is **regular** so $\overline{\mathrm{ACC}_{e,x}}$ is regular.

$(e, x) \notin \mathrm{HALT} \rightarrow \mathrm{ACC}_{e,x} = \emptyset$.

$\mathrm{ACC}_{e,x}$ is **regular** so $\overline{\mathrm{ACC}_{e,x}}$ is regular.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\leq f(n)$ states.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\leq f(n)$ states.
   Key The DFA for $\overline{\mathrm{ACC}_{e,x}}$ has $\leq f(n)$ states so the DFA for $\mathrm{ACC}_{e,x}$ has $\leq f(n)$ states.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\leq f(n)$ states.
   Key The DFA for $\overline{\mathrm{ACC}_{e,x}}$ has $\leq f(n)$ states so the DFA for $\mathrm{ACC}_{e,x}$ has $\leq f(n)$ states.
   So the DFA for $\mathrm{ACC}_{e,x}$ is one of $D_1, \ldots, D_N$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\le f(n)$ states.
   Key The DFA for $\overline{\mathrm{ACC}_{e,x}}$ has $\le f(n)$ states so the DFA for $\mathrm{ACC}_{e,x}$ has $\le f(n)$ states.
   So the DFA for $\mathrm{ACC}_{e,x}$ is one of $D_1, \ldots, D_N$.
4. Find all D_i's that accept only one string: $D_{i_1}, \ldots, D_{i_M}$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\leq f(n)$ states.
   Key The DFA for $\overline{\mathrm{ACC}_{e,x}}$ has $\leq f(n)$ states so the DFA for $\mathrm{ACC}_{e,x}$ has $\leq f(n)$ states.
   So the DFA for $\mathrm{ACC}_{e,x}$ is one of $D_1, \ldots, D_N$.
4. Find all D_i's that accept only one string: $D_{i_1}, \ldots, D_{i_M}$.
   For $1 \leq j \leq M$ let $L(D_{i_j}) = w_j$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\le f(n)$ states.
   Key The DFA for $\overline{\mathrm{ACC}_{e,x}}$ has $\le f(n)$ states so the DFA for $\mathrm{ACC}_{e,x}$ has $\le f(n)$ states.
   So the DFA for $\mathrm{ACC}_{e,x}$ is one of $D_1, \ldots, D_N$.
4. Find all D_i's that accept only one string: $D_{i_1}, \ldots, D_{i_M}$.
   For $1 \le j \le M$ let $L(D_{i_j}) = w_j$.
   $(\exists j)[w_j$ is accepting comp for $M_e(x)] \rightarrow (e, x) \in \mathrm{HALT}$.

Bdding Funct for $(\mathrm{DFA}, \mathrm{CFG})$ is Not Comp (cont)

Assume there is a computable bdding function f for $(\mathrm{DFA}, \mathrm{CFG})$.

1. Input (e, x). Create CFG G for $\overline{\mathrm{ACC}_{e,x}}$.
2. Let n be the size of G. Compute $f(n)$.
3. Let $D_1, \ldots, D_N$ be all DFA's with $\leq f(n)$ states.
   Key The DFA for $\overline{\mathrm{ACC}_{e,x}}$ has $\leq f(n)$ states so the DFA for $\mathrm{ACC}_{e,x}$ has $\leq f(n)$ states.
   So the DFA for $\mathrm{ACC}_{e,x}$ is one of $D_1, \ldots, D_N$.
4. Find all D_i's that accept only one string: $D_{i_1}, \ldots, D_{i_M}$.
   For $1 \leq j \leq M$ let $L(D_{i_j}) = w_j$.
   $(\exists j)[w_j$ is accepting comp for $M_e(x)] \rightarrow (e, x) \in \mathrm{HALT}$.
   If not then $(e, x) \notin \mathrm{HALT}$.

Concrete Thoughts

The following is **false**:

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Hence the following is **true**: There exists n,
$(\exists G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\forall M, |M| \le \mathbf{2^{2^n}})[L(M) \ne L(G)]$.

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \leq n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \leq \mathbf{2^{2^n}})[L(M) = L(G)]$.

Hence the following is **true**: There exists n,
$(\exists G)[|G| \leq n \wedge L(G) \text{ Reg }] \rightarrow (\forall M, |M| \leq \mathbf{2^{2^n}})[L(M) \neq L(G)]$.

This means that any DFA for M has $\geq 2^{2^n}$ states.

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Hence the following is **true**: There exists n,
$(\exists G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\forall M, |M| \le \mathbf{2^{2^n}})[L(M) \ne L(G)]$.

This means that any DFA for M has $\ge 2^{2^n}$ states.

So there is a regular language where the DFA is **much smaller** than the CFG.

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Hence the following is **true**: There exists n,
$(\exists G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\forall M, |M| \le \mathbf{2^{2^n}})[L(M) \neq L(G)]$.

This means that any DFA for M has $\ge 2^{2^n}$ states.

So there is a regular language where the DFA is **much smaller** than the CFG.

You can replace 2^{2^n} with any computable function.

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Hence the following is **true**: There exists n,
$(\exists G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\forall M, |M| \le \mathbf{2^{2^n}})[L(M) \ne L(G)]$.

This means that any DFA for M has $\ge 2^{2^n}$ states.

So there is a regular language where the DFA is **much smaller** than the CFG.

You can replace 2^{2^n} with any computable function.
More is know.

Concrete Thoughts

The following is **false**:

For all n,
$(\forall G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\exists M, |M| \le \mathbf{2^{2^n}})[L(M) = L(G)]$.

Hence the following is **true**: There exists n,
$(\exists G)[|G| \le n \wedge L(G) \text{ Reg }] \rightarrow (\forall M, |M| \le \mathbf{2^{2^n}})[L(M) \ne L(G)]$.

This means that any DFA for M has $\ge 2^{2^n}$ states.

So there is a regular language where the DFA is **much smaller** than the CFG.

You can replace 2^{2^n} with any computable function.
More is know.
Next slide.

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.
2. Any DFA for $L(G_n)$ is of size $\geq 2^{2^n}$.

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.
2. Any DFA for $L(G_n)$ is of size $\geq 2^{2^n}$.

2^{2^n} can be replaced by any computable function.

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.
2. Any DFA for $L(G_n)$ is of size $\geq 2^{2^n}$.

2^{2^n} can be replaced by any computable function.

Open Bill Question can you replace

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.
2. Any DFA for $L(G_n)$ is of size $\geq 2^{2^n}$.

2^{2^n} can be replaced by any computable function.

Open Bill Question can you replace
There are inf number of n

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.
2. Any DFA for $L(G_n)$ is of size $\geq 2^{2^n}$.

2^{2^n} can be replaced by any computable function.

Open Bill Question can you replace
There are inf number of n
with

Concrete Thoughts (cont)

One can show the following:
There are inf number of n such that there exists CFG G_n with:

1. G_n has size n and $L(G_n)$ is regular.
2. Any DFA for $L(G_n)$ is of size $\geq 2^{2^n}$.

2^{2^n} can be replaced by any computable function.

Open Bill Question can you replace
There are inf number of n
with
For all but a finite number of n

Final Notes

Final Notes

1. Hay (1981) proved that the bounding function for $(\mathrm{DFA}, \mathrm{CFG})$ can compute HALT. Note that *HALT* is Σ_1. I showed you her proof.

Final Notes

1. Hay (1981) proved that the bounding function for $(\mathrm{DFA}, \mathrm{CFG})$ can compute HALT. Note that *HALT* is Σ_1. I showed you her proof.
2. Gasarch (2015) proved that the bounding function for $(\mathrm{DFA}, \mathrm{CFG})$ can compute INF. Note that INF is Π_2. He also showed there is a bounding function for $(\mathrm{DFA}, \mathrm{CFG})$ of the same complexity as INF. Hence the complexity is solved.