

# CMPT-755

## Compilers

Anoop Sarkar

<http://www.cs.sfu.ca/~anoop>

# Programming Languages and Formal Language Theory

- We ask the question: *Does a particular formal language describe some key aspect of a programming language*
- Then we find out if that language **isn't** in a particular language class

# Programming Languages and Formal Language Theory

- For example, if we abstract some aspect of the programming language structure to the formal language:  
 $\{ww^R \mid \text{where } w \in \{a, b\}^*, w^R \text{ is the reverse of } w\}$  we can then ask if this language is a regular language
- If this is false, i.e. the language is not regular, then we have to go beyond regular languages

# Defining the Set of Regular Languages

- A **regular language** is a set of strings constructed as follows:
  - $\phi$  is a RL
  - $\forall x \in \Sigma \cup \epsilon, \{x\}$  is a RL
  - If  $L_1$  and  $L_2$  are RLs then the following are RLs,
    1.  $L_1 \cdot L_2 = \{xy \mid x \in L_1, y \in L_2\}$
    2.  $L_1 \cup L_2$
    3.  $L_1^*$

# Recursion in Regular Languages

- Consider a regular expression for arithmetic expressions:

$2 + 3 * 4$

$8 * 10 + -24$

$2 + 3 * -2 + 8 + 10$

$$^{\wedge} \backslash s^{*} - ? \backslash s^{*} \backslash d^{+} \backslash s^{*} ( ( \backslash + | \backslash * ) \backslash s^{*} - ? \backslash s^{*} \backslash d^{+} \backslash s^{*} ) ^{*} \$$$

- *Can we compute the meaning of these expressions?*

# Recursion in Regular Languages

- Construct the finite state automata and associate the meaning with the state sequence
- However, this solution is missing something crucial about arithmetic expressions – *what is it?*

# Do Programming Languages belong to Regular Languages

- Consider the following arithmetic expressions
  - $((2) + (3)) * (4)$
  - $((8) * ((10) + (-24)))$
- Map  $(\rightarrow a$  and  $) \rightarrow b$ . Map everything else to  $\epsilon$ .
- This results in strings like *aaababbabb* and *aabaababbb*
- What is a good description of this language? Let's call it  $L$

# Pumping Lemma proofs

- Is  $L$  a regular language?
- To show something is *not* a regular language, we use the **pumping lemma**
- For any infinite set of strings generated by a finite-state machine if you consider a string that is long enough from this set, there has to be a loop which visits the same state at least twice (from *the pigeonhole principle*)
- Thus, in a regular language  $L$ , there are strings  $x, y, z$  such that  $xy^n z \in L$  for  $n \geq 0$  where  $y \neq \epsilon$



## Pumping Lemma proofs

- Let  $L'$  be the intersection of  $L$  with the language  $L_1$  defined by the regular expression  $a^*b^*$
- Intersect the set  $L = \{\epsilon, ab, abab, aabb, \dots\}$  with  $L_1 = \{\epsilon, a, b, aa, ab, aab, abb, bb, \dots\}$
- Recall that RLs are closed under intersection, so  $L'$  must also be a RL. In fact, we can describe  $L'$  as the language  $a^n b^n$  for  $n \geq 0$

## Pumping Lemma proofs

- For any choice of  $y$  (consider  $a^i$  or  $a^i b$  or  $b^i$ ) if we multiply  $y^n$  for  $n \geq 0$  we get strings that are not in  $L'$
- For example, for a string  $aaabbb$  if we pick  $y = ab$  and pick  $n = 2$  we get a string  $aaababbb$  which is not in  $L'$
- Hence, the pumping lemma leads to the conclusion that  $L'$  is **not** regular
- This implies that  $L$  is not regular since RLs are closed under intersection
- What lies beyond the set of regular languages?

# The Chomsky Hierarchy

- **unrestricted** or **type-0** grammars, generate the *recursively enumerable* languages, automata equals *Turing machines*
- **context-sensitive** or **type-1** grammars, generate the *context-sensitive* languages, automata equals *Linear Bounded Automata*
- **context-free** or **type-2** grammars, generate the *context-free* languages, automata equals *Pushdown Automata*
- **regular** or **type-3** grammars, generate the *regular* languages, automata equals *Finite-State Automata*

# The Chomsky Hierarchy

## A system of grammars $G = (N, T, P, S)$

- $T$  is a set of symbols called terminal symbols.  
Also called the alphabet  $\Sigma$
- $N$  is a set of non-terminals, where  $N \cap T = \emptyset$   
Some notation:  $\alpha, \beta, \gamma \in (N \cup T)^*$   
 $N$  is sometimes called the set of variables  $V$
- $P$  is a set of production rules that provide a finite description of an infinite set of strings (a language)
- $S$  is the start non-terminal symbol (similar to the start state in a FSA)

# Languages

- Language defined by  $G$ :  $L(G)$ 
  - $L(G)$ : set of strings  $w \in T^*$  derived from  $S$
  - $S \Rightarrow^+ w$  (derives in 1 or more steps using rules in  $P$ )
  - $w$  is a sentence of  $G$
  - Sentential form:  $S \Rightarrow^+ \alpha$  and  $\alpha$  contains a mix of terminals and non-terminals
- Two grammars  $G_1$  and  $G_2$  are equivalent if  $L(G_1) = L(G_2)$

The Chomsky Hierarchy:  
 $G = (N, T, P, S)$  where,  $\alpha, \beta, \gamma \in (N \cup T)^*$

- **unrestricted** or **type-0** grammars:  $\alpha \rightarrow \gamma$ , such that  $\alpha \neq \epsilon$
- **context-sensitive** or **type-1** grammars:  $\alpha \rightarrow \gamma$ , where  $|\gamma| \geq |\alpha|$   
CSG Normal Form:  $\alpha A \beta \rightarrow \alpha \gamma \beta$ , such that  $\gamma \neq \epsilon$  and  $S \rightarrow \epsilon$  if  $\epsilon \in L(G)$
- **context-free** or **type-2** grammars:  $A \rightarrow \gamma$
- **regular** or **type-3** grammars:  $A \rightarrow a B$  or  $A \rightarrow a$

Regular grammars: **right-linear CFG**:  $L(G) = L(a^*b^*)$

$$A \rightarrow a A \quad (1)$$

$$A \rightarrow \epsilon \quad (2)$$

$$A \rightarrow b B \quad (3)$$

$$B \rightarrow b B \quad (4)$$

$$B \rightarrow \epsilon \quad (5)$$

- Input:  $bb$
- Derivation using sentential forms:  $A \Rightarrow bB \Rightarrow bbB \Rightarrow bb\epsilon = bb$

Context-free grammars:  $L(G) = \{a^n b^n \mid n \geq 0\}$

$$S \rightarrow a S b$$

$$S \rightarrow \epsilon$$

- Input:  $aabb$
- Derivation using sentential forms:  
 $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aa\epsilon bb = aabb$



Context-free grammars:  $L(G) = \{a^n \mid n \geq 0\}$

$$S \rightarrow S S$$

$$S \rightarrow a$$

- Input:  $aaaa$

- Derivation using sentential forms:

$$S \Rightarrow SS \Rightarrow aS \Rightarrow aSS \Rightarrow aaS \Rightarrow aaSS \Rightarrow aaaS \Rightarrow aaaa$$

- But what about another derivation:

$$S \Rightarrow SS \Rightarrow SSS \Rightarrow SSSS \Rightarrow aSSS \Rightarrow \dots \Rightarrow aaaa$$

- Key problem with CFGs: **ambiguity**

Context-sensitive grammars:  $L(G) = \{a^n b^n \mid n \geq 1\}$

$$S \rightarrow S B C$$

$$S \rightarrow a C$$

$$a B \rightarrow a a$$

$$C B \rightarrow B C$$

$$B a \rightarrow a a$$

$$C \rightarrow b$$

Context-sensitive grammars:  $L(G) = \{a^n b^n \mid n \geq 1\}$

$$\begin{array}{ccccccc}
 & & & & & & S_1 \\
 & & & & & S_2 & B_1 & C_1 \\
 & & & S_3 & B_2 & C_2 & B_1 & C_1 \\
 a_3 & C_3 & B_2 & C_2 & B_1 & C_1 \\
 a_3 & B_2 & C_3 & C_2 & B_1 & C_1 \\
 a_3 & a_2 & C_3 & C_2 & B_1 & C_1 \\
 a_3 & a_2 & C_3 & B_1 & C_2 & C_1 \\
 a_3 & a_2 & B_1 & C_3 & C_2 & C_1 \\
 a_3 & a_2 & a_1 & C_3 & C_2 & C_1 \\
 a_3 & a_2 & a_1 & b_3 & b_2 & b_1
 \end{array}$$

Unrestricted grammars:  $L(G) = \{a^{2i} \mid i \geq 1\}$

$$S \rightarrow A C a B$$

$$C a \rightarrow a a C$$

$$C B \rightarrow D B$$

$$\mathbf{C B} \rightarrow \mathbf{E}$$

$$a D \rightarrow D a$$

$$A D \rightarrow A C$$

$$a E \rightarrow E a$$

$$\mathbf{A E} \rightarrow \epsilon$$

Unrestricted grammars:  $L(G) = \{a^{2i} \mid i \geq 1\}$

$S$   
 $A C a B$   
 $A a a C B$   
 $A a a E$   
 $A a E a$   
 $A E a a$   
 $a a$

Unrestricted grammars:  $L(G) = \{a^{2i} \mid i \geq 1\}$

- A and B serve as left and right end-markers for sentential forms (derivation of each string)
- C is a marker that moves through the string of  $a$ 's between A and B, doubling their number using  $C a \rightarrow a a C$
- When C hits right end-marker B, it becomes a D or E by  $C B \rightarrow D B$  or  $C B \rightarrow E$
- If a D is chosen, that D migrates left using  $a D \rightarrow D a$  until left end-marker A is reached





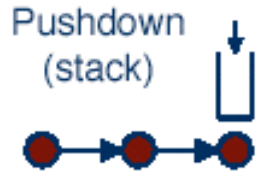





Unrestricted grammars:  $L(G) = \{a^{2i} \mid i \geq 1\}$

- At that point D becomes C using  $A D \rightarrow A C$  and the process starts over
- Finally, E migrates left until it hits left end-marker A using  $a E \rightarrow E a$
- Note that  $L(G) = \{a^{2i} \mid i \geq 1\}$  can also be written as a context-sensitive grammar

# Examples of Languages in the Chomsky Hierarchy

- **context-sensitive** grammars:  $0^i$ ,  $i$  is not a prime number and  $i > 0$
- **indexed** grammars:  $0^n 1^n 2^n \dots m^n$ , for any fixed  $m$  and  $n \geq 0$
- **context-free** grammars:  $0^n 1^n$  for  $n \geq 0$
- **deterministic context-free** grammars:  $S' \rightarrow S c, S \rightarrow S A \mid A, A \rightarrow a S b \mid ab$ : the language of "balanced parentheses"
- **regular** grammars:  $(0|1)^* 00(0|1)^*$



<i>Language</i>	<i>Automaton</i>	<i>Grammar</i>	<i>Recognition</i>	<i>Dependency</i>
Recursively Enumerable Languages	Turing Machine 	Unrestricted $Baa \rightarrow A$	Undecidable	Arbitrary
Context-Sensitive Languages	Linear-Bounded 	Context-Sensitive $At \rightarrow aA$	NP-Complete 	Crossing 
Context-Free Languages	Pushdown (stack) 	Context-Free $S \rightarrow gSc$	Polynomial 	Nested 
Regular Languages	Finite-State Machine 	Regular $A \rightarrow cA$	Linear 	Strictly Local 

# Complexity of Parsing Algorithms

- Given grammar  $G$  and input  $x$ , provide algorithm for: Is  $x \in L(G)$ ?
  - **unrestricted**: undecidable
  - **context-sensitive**: NSPACE( $n$ ) – linear non-deterministic space
  - **indexed** grammars: NP-Complete
  - **context-free**:  $O(n^3)$
  - **deterministic context-free**:  $O(n)$
  - **regular** grammars:  $O(n)$

## Verifying that $L = L(G)$

- Let's say we have a context-free grammar  $G$  and a description of a language  $L$
- How can we say for sure that  $L = L(G)$ ?
- By verifying the statement in two directions:
  - $\Rightarrow$  All strings generated by  $G$  are in  $L$
  - $\Leftarrow$  All strings  $w \in L$  can be generated by  $G$

## Verifying that $L = L(G)$

- Example:  $T = \{a, b\}$ . Consider language  $L$  to be “all strings with same number of  $as$  and  $bs$ ”
- Consider  $G$  to be a CFG:  $S \rightarrow \epsilon \mid a S b S \mid b S a S$
- To verify that  $L = L(G)$ , prove that
  - $\Rightarrow$  All strings generated by  $G$  are in  $L$
  - $\Leftarrow$  All strings  $w \in L$  can be generated by  $G$

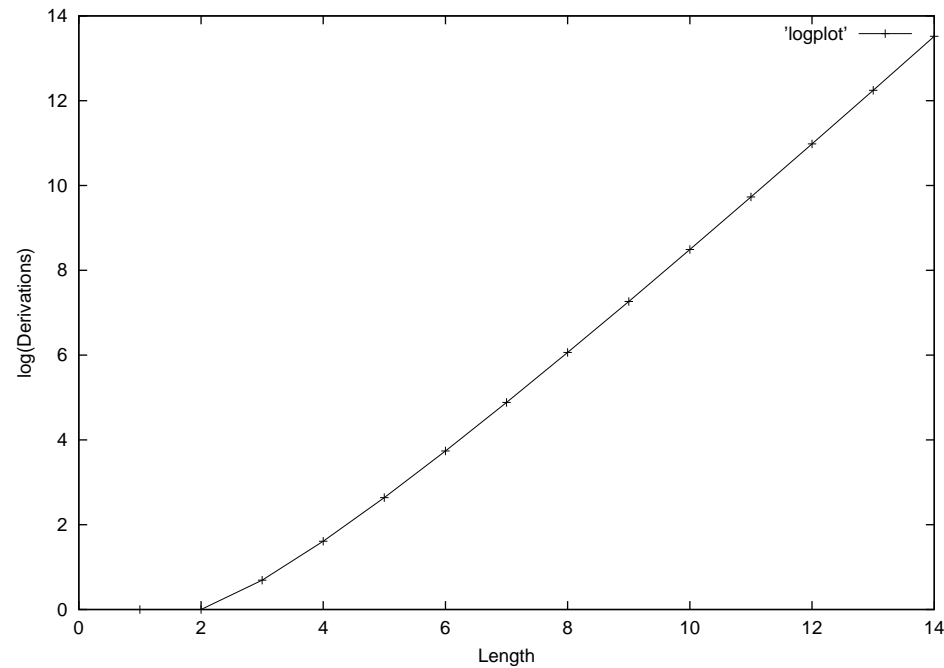
## Proof ( $\Rightarrow$ ): All strings generated by $G$ are in $L$

- Proof by induction:
  - **Base case:**  $\epsilon$  is in  $L$  (trivial)
  - **Inductive hypothesis:** Assume  $u \in L$  and  $v \in L$ . Let  $w$  be generated by  $G$  with  $|u| < |w|$  and  $|v| < |w|$ 
    - \* Because  $w$  is generated by  $G$  then either  $w \Rightarrow a u b v$  or  $w \Rightarrow b u a v$ , where  $u$  and  $v$  are generated by  $G$
    - \* Since  $|u| < |w|$  and  $|v| < |w|$  and  $u, v \in L$  then since we only added a single matching  $a, b$  pair, we can conclude that  $w$  is in  $L$

Proof ( $\Leftarrow$ ): All strings  $w \in L$  can be generated by  $G$

- Proof by induction (show that  $S \Rightarrow^+ w$ ):
  - **Base case:**  $w = \epsilon$  (trivial:  $S \rightarrow \epsilon$ )
  - **Inductive hypothesis:** For a given  $w \in L$ , assume that for all  $u, v \in L$  where  $|u| < |w|$  and  $|v| < |w|$  we have  $S \Rightarrow^+ u$  and  $S \Rightarrow^+ v$ 
    - \* **Case 1 –  $w$  starts with  $a$ :** Find the first  $b$  from the right so that  $w = a u b v$  and  $v$  has the same number of  $a$ s and  $b$ s  
Because  $w \in L$  it has to be true that  $u, v \in L$  and by the inductive hypothesis  $S \Rightarrow^+ u$  and  $S \Rightarrow^+ v$   
Using rule  $S \rightarrow a S b S$  and the above step we get  $S \Rightarrow^+ w$
    - \* **Case 2 –  $w$  starts with  $b$ :** (analogous to Case 1)

# CFG Ambiguity: Number of derivations grows exponentially



$L(G) = a^+$  using CFG rules  $\{ S \rightarrow S S, S \rightarrow a \}$

# CFG Ambiguity

- Algebraic character of parse derivations
- Power Series for grammar for the (simplified) arithmetic expression CFG:  
 $E \rightarrow \text{digit} \mid E \text{ binop } E$
- Write it down as an equation with coefficients equal to number of different analyses possible:

$$\begin{aligned} E &= \text{digit} + \text{digit binop digit} \\ &+ 2(\text{digit binop digit binop digit}) \\ &+ 5(\text{digit binop digit binop digit binop digit}) \\ &+ 14 \dots \end{aligned}$$



# CFG Ambiguity

- Coefficients in previous equation equal the number of parses for each string derived from  $E$
- These ambiguity coefficients are Catalan numbers:

$$Cat(n) = \frac{1}{n+1} \binom{2n}{n}$$

- $\binom{a}{b}$  is the *binomial coefficient*

$$\binom{a}{b} = \frac{a!}{(b!(a-b)!)}$$

# Catalan numbers

- Why Catalan numbers?  $\text{Cat}(n)$  is the number of ways to parenthesize an expression of length  $n$  with two conditions:
  1. there must be equal numbers of open and close parens
  2. they must be properly nested so that an open precedes a close

# Catalan numbers

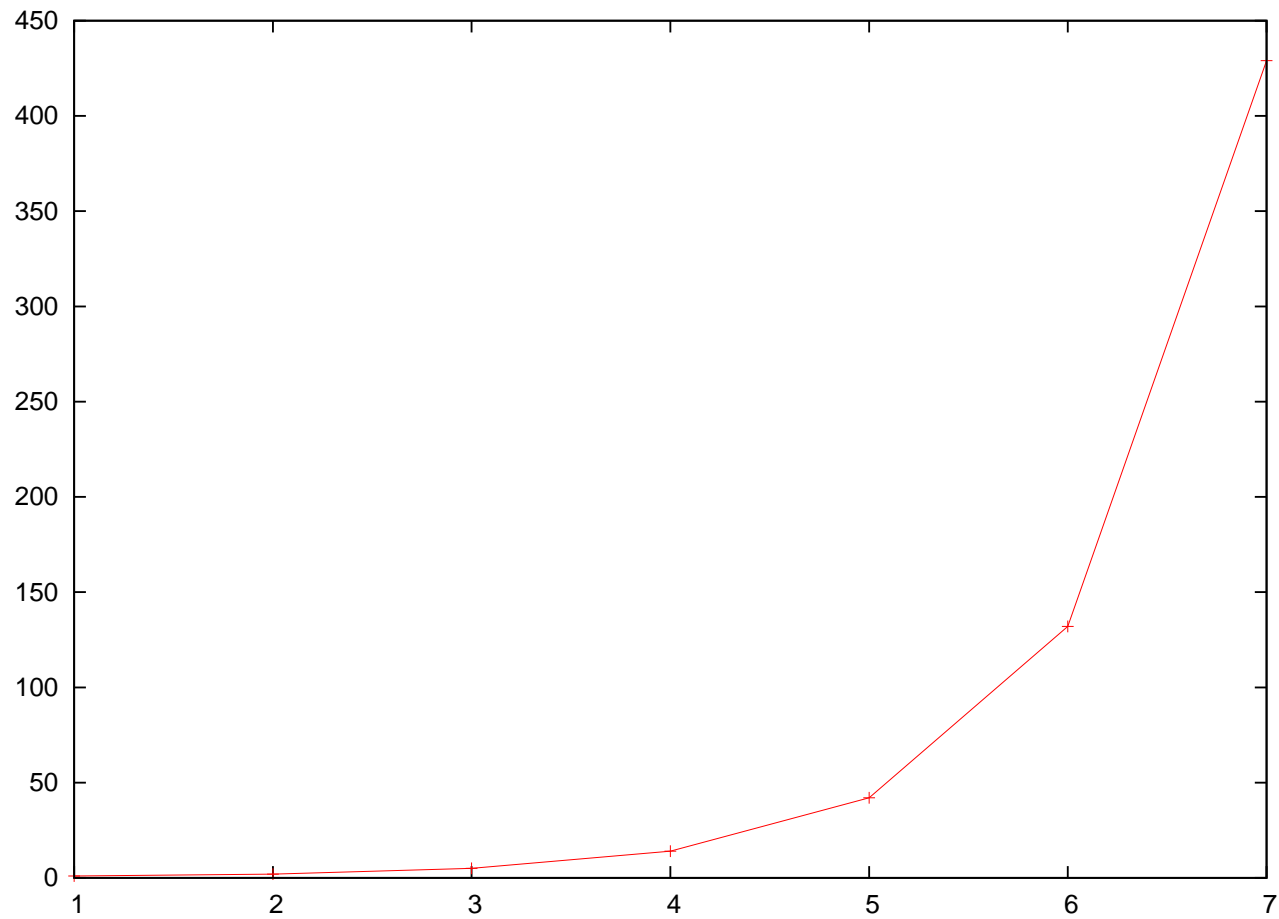
- For an expression of length  $n$  there are a total of  $2n$  choose  $n$  parenthesis pairs. But  $n + 1$  of them have the right parenthesis to the left of its matching left parenthesis  $()()$ .
- So we divide  $2n$  choose  $n$  by  $n + 1$ :

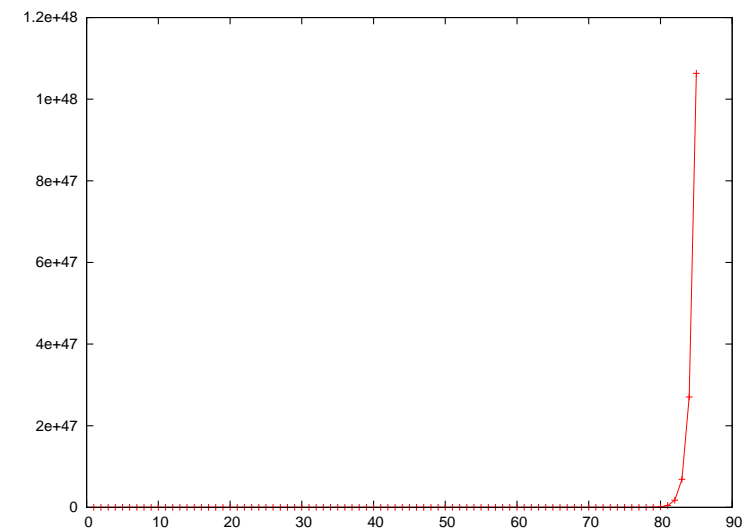
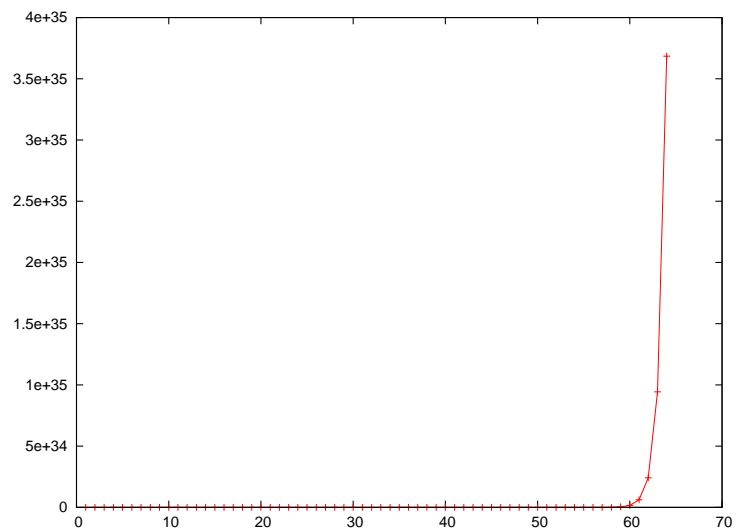
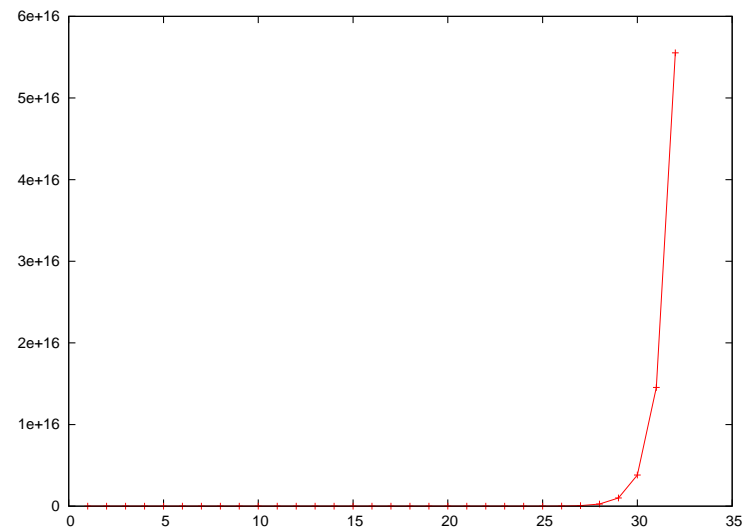
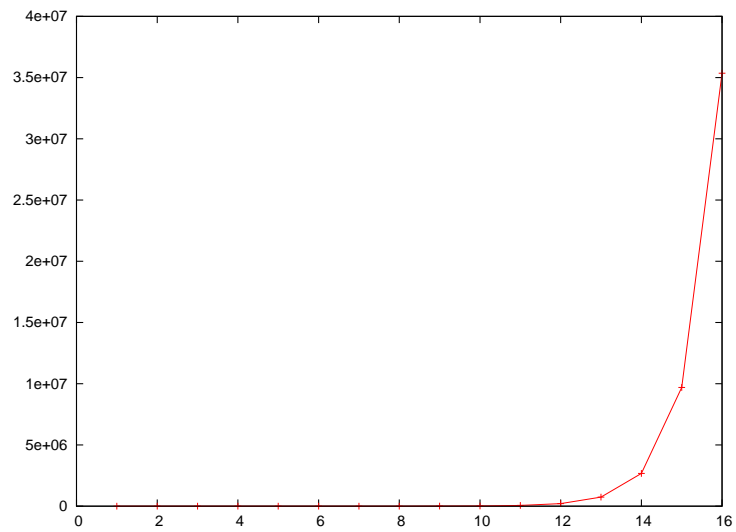
$$Cat(n) = \frac{1}{n + 1} \binom{2n}{n}$$

## Catalan numbers

$n$	catalan( $n$ )
1	1
2	2
3	5
4	14
5	42
6	132
7	429
8	1430
9	4862
10	16796

# Catalan numbers





# Summary

- Aspects of PL structure cannot be represented by FSAs
- Pumping lemma proofs for proving a language is not regular
- Chomsky hierarchy: from FSAs to Turing machines
- Verifying that a particular language is generated by a grammar  $G$
- Context-free grammars (seems sufficient for PLs) but problems with ambiguity