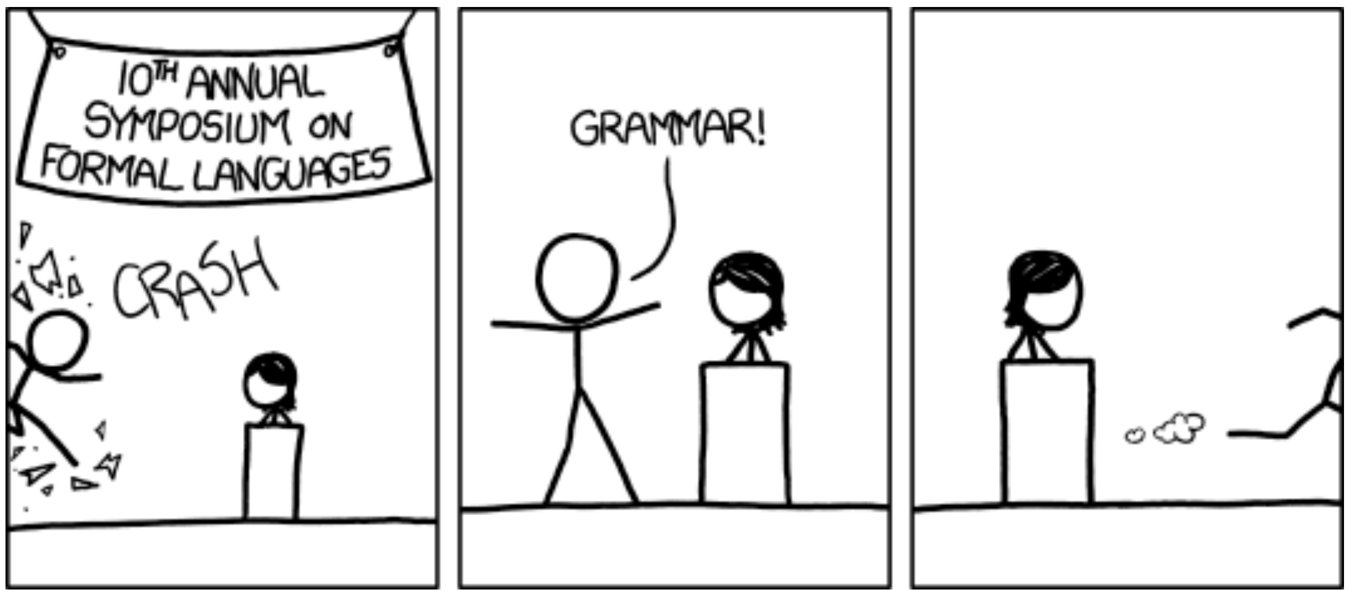
CSE 311: Foundations of Computing

Lecture 19: Regular Expressions & Context-Free Grammars



[Audience looks around]
“What is going on? There must be some context we’re missing”

Administrivia

- HW7 out tomorrow
  - longer than usual

    Problem 1 may be the hardest proof so far (we'll see)

- Part due next Friday
  Rest due Monday after
  - won't get through all the material until Wed
  - start early!

Review: Context-Free Grammars

- A Context-Free Grammar (CFG) is given by a finite set of substitution rules involving
  - A finite set **V** of *variables* that can be replaced
  - Alphabet Σ of *terminal symbols* that can't be replaced
  - One variable, usually **S**, is called the *start symbol*

- The substitution rules involving a variable **A**, written as

$$\mathbf{A} \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$$

where each w_i is a string of variables and terminals
– that is $w_i \in (\mathbf{V} \cup \Sigma)^*$

Review: How CFGs generate strings

- Begin with start symbol **S**

- If there is some variable **A** in the current string you can replace it by one of the w's in the rules for **A**
  - $\mathbf{A} \rightarrow w_1 \mid w_2 \mid \cdots \mid w_k$
  - Write this as $x\mathbf{A}y \Rightarrow xwy$
  - Repeat until no variables left

- The set of strings the CFG generates are all strings produced in this way (after a finite number of steps) that have no variables

Example Context-Free Grammars

Example: $\mathbf{S} \rightarrow 0\mathbf{S}0 \mid 1\mathbf{S}1 \mid 0 \mid 1 \mid \varepsilon$

The set of all binary palindromes.

E.g., to see that 001101100 is in the language:

$S \Rightarrow 0S0$
$\Rightarrow 00S00$
$\Rightarrow 001S100$
$\Rightarrow 0011S1100$
$\Rightarrow 00110100$

Example Context-Free Grammars

Grammar for $\{0^n 1^n : n \geq 0\}$

(all strings with same # of 0's and 1's with all 0's before 1's)

$$S \rightarrow 0S1 \mid \varepsilon$$

HW7 Problem 1:

binary strings with an equal number of 0s and 1s

Simple Arithmetic Expressions

$$E \rightarrow E+E \mid E*E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Generate $(2*x) + y$

Simple Arithmetic Expressions

$$E \rightarrow E+E \mid E*E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

Generate $(2*x) + y$

$$E \Rightarrow E+E \Rightarrow (E)+E \Rightarrow (E*E)+E \Rightarrow (2*E)+E \Rightarrow (2*x)+E \Rightarrow (2*x)+y$$

Six different ways to do

$$(E*E)+E \Rightarrow \ldots \Rightarrow (2*x)+y$$

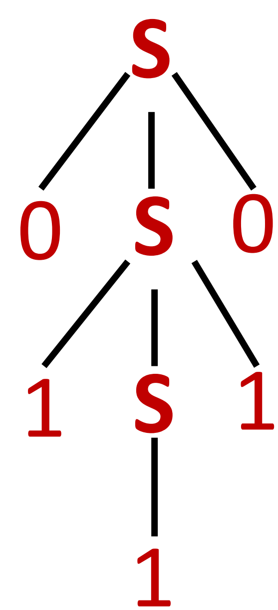
Parse Trees

Parse tree shows the substitutions but not the exact order in which they were performed

Suppose that grammar **G** generates a string **x**

- A *parse tree* of **x** for **G** has
  - Root labeled **S** (start symbol of **G**)
  - The children of any node labeled **A** are labeled by symbols of **w** left-to-right for some rule $A \rightarrow w$
  - The symbols of **x** label the leaves ordered left-to-right

$$S \rightarrow 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon$$



Parse tree of 01110

Simple Arithmetic Expressions

$$E \rightarrow E+E \mid E*E \mid (E) \mid x \mid y \mid z \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

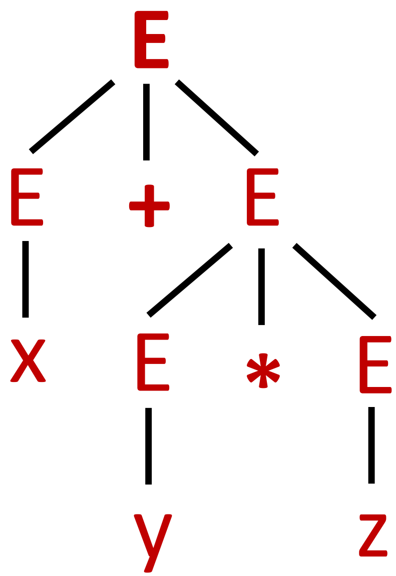
Generate x+y∗z in two ways that give two *different* parse trees

Simple Arithmetic Express

Different parse trees give different **meanings** (here, order of operations)

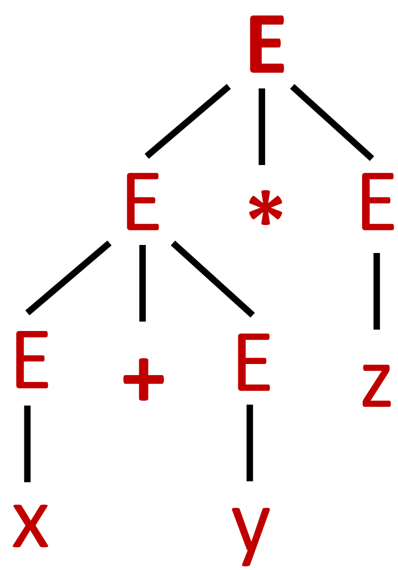
E→ E+E | E∗E | (E) | x | y | z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Generate x+y∗z in ways that give two *different* parse trees



E ⇒ E+E ⇒ x+E ⇒ x+E*E ⇒ x+y*E ⇒ x+y*z

(multiply y with z and then add to x)



E ⇒ E*E ⇒ E+E*E ⇒ x+E*E

⇒ x+y*E ⇒ x+y*z

(add x to y, then multiply by z)

Building precedence in simple arithmetic expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

$$\mathbf{E} \rightarrow \mathbf{T} \mid \mathbf{E{+}T}$$

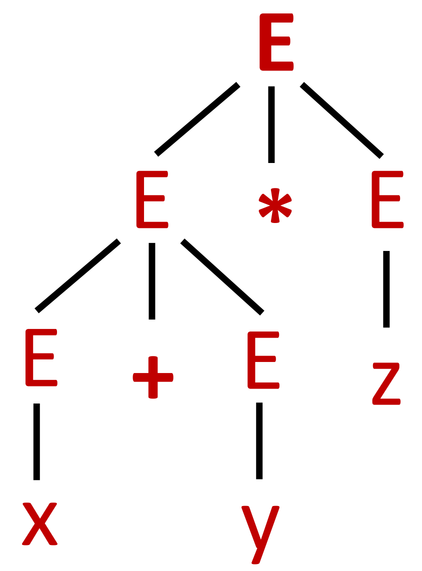
$$\mathbf{T} \rightarrow \mathbf{F} \mid \mathbf{F{*}T}$$

$$\mathbf{F} \rightarrow (\mathbf{E}) \mid \mathbf{I} \mid \mathbf{N}$$

$$\mathbf{I} \rightarrow x \mid y \mid z$$

$$\mathbf{N} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

No longer allows:



Building precedence in simple arithmetic expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

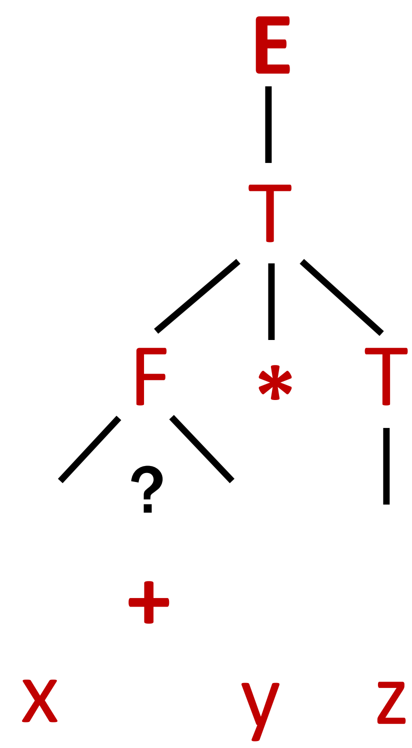
$$\mathbf{E} \rightarrow \mathbf{T} \mid \mathbf{E}+\mathbf{T}$$

$$\mathbf{T} \rightarrow \mathbf{F} \mid \mathbf{F}*\mathbf{T}$$

$$\mathbf{F} \rightarrow (\mathbf{E}) \mid \mathbf{I} \mid \mathbf{N}$$

$$\mathbf{I} \rightarrow x \mid y \mid z$$

$$\mathbf{N} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$



Building precedence in simple arithmetic expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

E $\rightarrow$ **T** | **E+T**

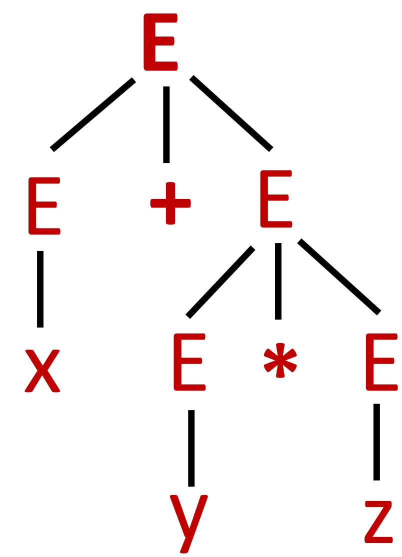
T $\rightarrow$ **F** | **F$*$T**

F $\rightarrow$ (**E**) | **I** | **N**

I $\rightarrow$ x | y | z

N $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Still allows:



Building precedence in simple arithmetic expressions

- **E** – expression (start symbol)
- **T** – term **F** – factor **I** – identifier **N** - number

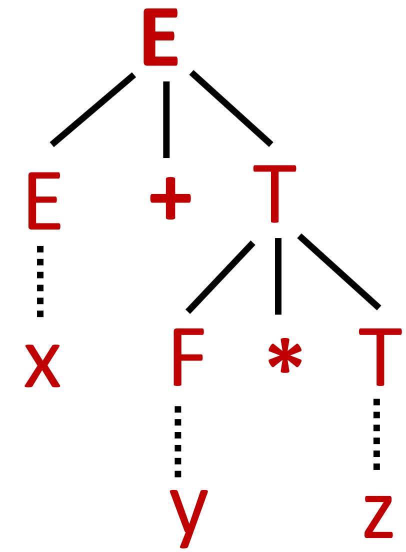
$$\mathbf{E} \rightarrow \mathbf{T} \mid \mathbf{E}+\mathbf{T}$$

$$\mathbf{T} \rightarrow \mathbf{F} \mid \mathbf{F}*\mathbf{T}$$

$$\mathbf{F} \rightarrow (\mathbf{E}) \mid \mathbf{I} \mid \mathbf{N}$$

$$\mathbf{I} \rightarrow x \mid y \mid z$$

$$\mathbf{N} \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$



CFGs and recursively-defined sets of strings

- A CFG with the start symbol **S** as its only variable recursively defines the set of strings of terminals that **S** can generate
  - i.e., CFGs translate into recursively defined sets of strings
  - set of parse trees is also a recursively defined set

- A CFG with more than one variable is a simultaneous recursive definition of the sets of strings generated by *each* of its variables
  - sometimes necessary to use more than one

CFGs and regular expressions

Theorem: For any set of strings (language) A described by a regular expression, there is a CFG that recognizes A.

Proof idea: Structural induction based on the recursive definition of regular expressions...

Regular Expressions over Σ

- **Basis:**
  - ε is a regular expression
  - ***a*** is a regular expression for any $a \in \Sigma$
- **Recursive step:**
  - If **A** and **B** are regular expressions then so are:

    $(A \cup B)$

    (AB)

    A^*

CFGs are more general than REs

- CFG to match RE **ε**

  S $\rightarrow \varepsilon$

- CFG to match RE ***a*** (for any $a \in \Sigma$)

  S $\rightarrow$ a

CFGs are more general than REs

Suppose CFG with start symbol $\mathbf{S_1}$ matches RE **A**
CFG with start symbol $\mathbf{S_2}$ matches RE **B**

- CFG to match RE **A** $\cup$ **B**

$$\mathbf{S} \rightarrow \mathbf{S_1} \mid \mathbf{S_2}$$

- CFG to match RE **AB**

$$\mathbf{S} \rightarrow \mathbf{S}_1\, \mathbf{S}_2$$

CFGs are more general than REs

Suppose CFG with start symbol $\mathbf{S_1}$ matches RE **A**

- CFG to match RE $\mathbf{A}^*$ (= $\varepsilon \cup \mathbf{A} \cup \mathbf{AA} \cup \mathbf{AAA} \cup \dots$)

$$\mathbf{S} \rightarrow \mathbf{S_1}\,\mathbf{S} \mid \varepsilon$$

Backus-Naur Form (The same thing...)

BNF (Backus-Naur Form) grammars

- Originally used to define programming languages
- Variables denoted by long names in angle brackets, e.g.

  <identifier>, <if-then-else-statement>, <assignment-statement>, <condition>

  $::=$ used instead of $\rightarrow$

BNF for C

```
statement:
  ((identifier | "case" constant-expression | "default") ":")*
  (expression? ";" |
   block |
   "if" "(" expression ")" statement |
   "if" "(" expression ")" statement "else" statement |
   "switch" "(" expression ")" statement |
   "while" "(" expression ")" statement |
   "do" statement "while" "(" expression ")" ";" |
   "for" "(" expression? ";" expression? ";" expression? ")" statement |
   "goto" identifier ";" |
   "continue" ";" |
   "break" ";" |
   "return" expression? ";"
  )

block: "{" declaration* statement* "}"

expression:
  assignment-expression%

assignment-expression: (
    unary-expression (
      "=" | "*=" | "/=" | "%=" | "+=" | "-=" | "<<=" | ">>=" | "&=" |
      "^=" | "|="
    )
  )* conditional-expression

conditional-expression:
  logical-OR-expression ( "?" expression ":" conditional-expression )?
```

Parse Trees

Back to middle school:

<sentence>::=<noun phrase><verb phrase>

<noun phrase>::==<article><adjective><noun>

<verb phrase>::=<verb><adverb>|<verb><object>

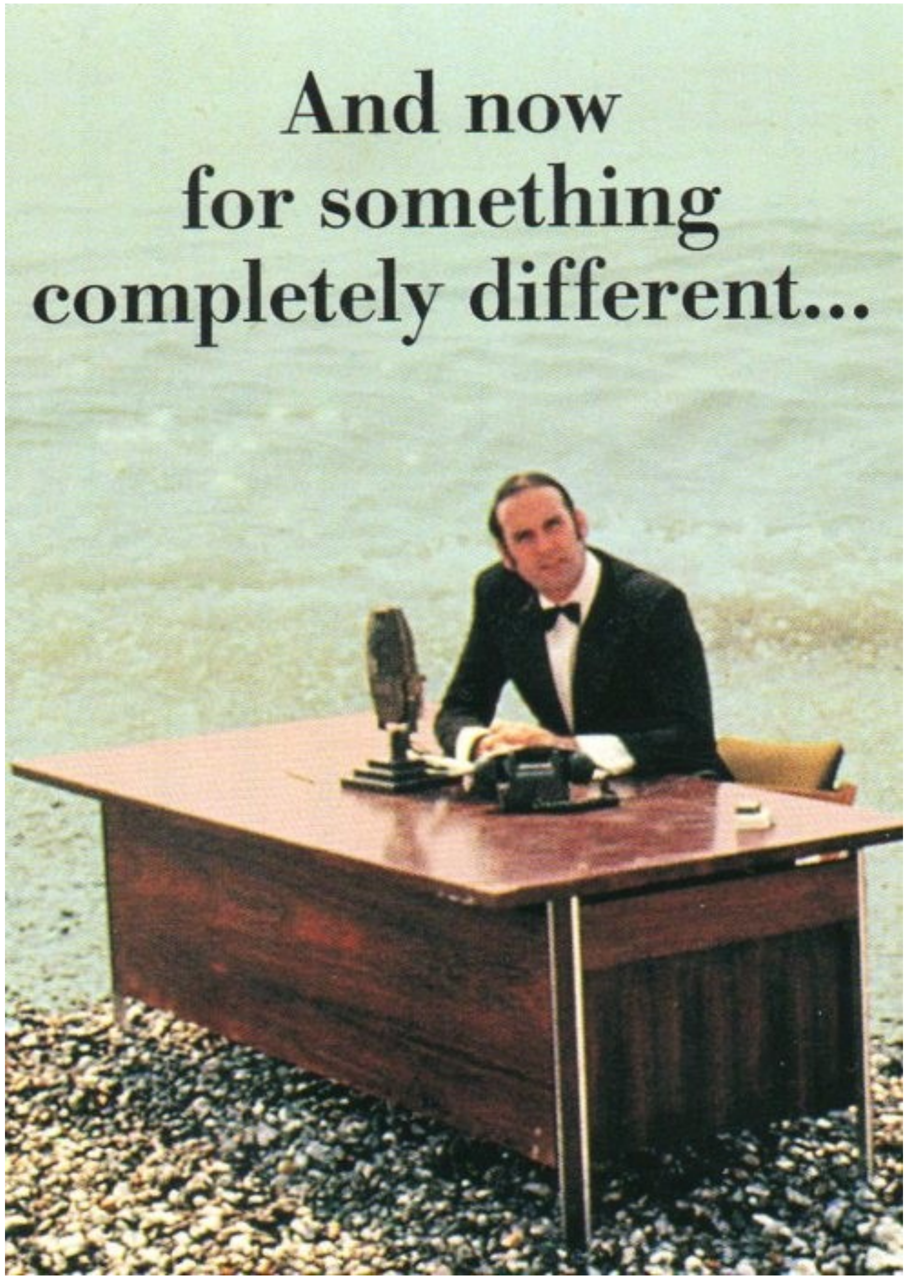
<object>::=<noun phrase>

Parse:

The yellow duck squeaked loudly

The red truck hit a parked car

Relations and Directed Graphs



Relations

Let A and B be sets,

A **binary relation from** A **to** B is a subset of $A \times B$

Let A be a set,

A **binary relation on** A is a subset of $A \times A$

Relations You Already Know!

$\geq$ on $\mathbb{N}$

That is: $\{(x,y) : x \geq y \text{ and } x, y \in \mathbb{N}\}$

$<$ on $\mathbb{R}$

That is: $\{(x,y) : x < y \text{ and } x, y \in \mathbb{R}\}$

$=$ on Σ^*

That is: $\{(x,y) : x = y \text{ and } x, y \in \Sigma^*\}$

$\subseteq$ on $\mathcal{P}(U)$ for universe U

That is: $\{(A,B) : A \subseteq B \text{ and } A, B \in \mathcal{P}(U)\}$

More Relation Examples

$\mathbf{R_1}$ = {(a, 1), (a, 2), (b, 1), (b, 3), (c, 3)}

$\mathbf{R_2}$ = {(x, y) | x ≡ y (mod 5) }

$\mathbf{R_3}$ = {(c_1, c_2) | c_1 is a prerequisite of c_2 }

$\mathbf{R_4}$ = {(s, c) | student s has taken course c }

Properties of Relations

Let R be a relation on A.

R is **reflexive** iff $(a,a) \in R$ for every $a \in A$

R is **symmetric** iff $(a,b) \in R$ implies $(b,a) \in R$

R is **antisymmetric** iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$

R is **transitive** iff $(a,b) \in R$ and $(b,c) \in R$ implies $(a,c) \in R$

Which relations have which properties?

$\geq$ **on** $\mathbb{N}$ **:**

$<$ **on** $\mathbb{R}$ **:**

$=$ **on** Σ^* **:**

$\subseteq$ **on** $\mathcal{P}$**(U):**

R_2 = $\{(x, y) \mid x \equiv y \pmod 5\}$:

R_3 = $\{(c_1, c_2) \mid c_1$ is a prerequisite of $c_2\}$:

R is **reflexive** iff $(a,a) \in R$ for every $a \in A$
R is **symmetric** iff $(a,b) \in R$ implies $(b, a) \in R$
R is **antisymmetric** iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$
R is **transitive** iff $(a,b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$

Which relations have which properties?

$\geq$ on $\mathbb{N}$: Reflexive, Antisymmetric, Transitive

$<$ on $\mathbb{R}$: Antisymmetric, Transitive

$=$ on Σ^* : Reflexive, Symmetric, Antisymmetric, Transitive

$\subseteq$ on $\mathcal{P}(U)$: Reflexive, Antisymmetric, Transitive

$R_2 = \{(x, y) \mid x \equiv y \pmod 5\}$: Reflexive, Symmetric, Transitive

$R_3 = \{(c_1, c_2) \mid c_1$ is a prerequisite of $c_2\}$: Antisymmetric

R is **reflexive** iff $(a,a) \in R$ for every $a \in A$
R is **symmetric** iff $(a,b) \in R$ implies $(b, a) \in R$
R is **antisymmetric** iff $(a,b) \in R$ and $a \neq b$ implies $(b,a) \notin R$
R is **transitive** iff $(a,b) \in R$ and $(b, c) \in R$ implies $(a, c) \in R$