EECS 368
Programming Language Paradigms

Dr. Andy Gill

Department of Electrical Engineering & Computer Science
University of Kansas

August 24, 2011

KU

Summary of Last Class

Machine Codes to Modern Languages

Improving levels of abstraction over time.

- The ability to write using mnemonics, not binary codes
- The ability to write formulas
- The ability to structure code in terms of objects
- The ability to create objects, an have them automatically disposed of after you have finished with them

Syntax for Languages

- Different flavors and families of syntax
- Understanding a language starts with understanding its syntax
- There are tools and techniques for expressing and understanding syntax

Class Narrative

Syntax trees are a fundamental data structure used to understand and implement computer languages

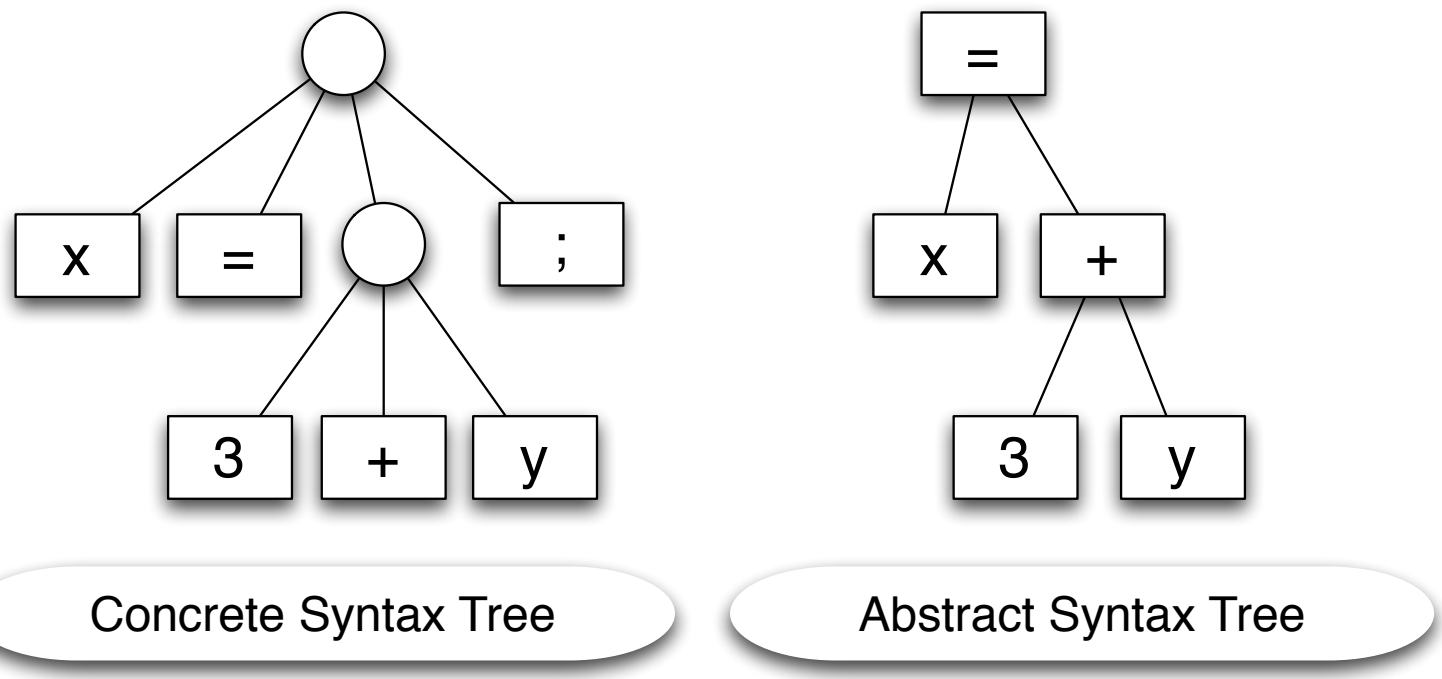
Backus-Naur Form (BNF) is way of expressing valid concrete syntax trees

KU

What is a Syntax Tree?

A tree that represents the syntactical structure of a specific program written in a specific language.

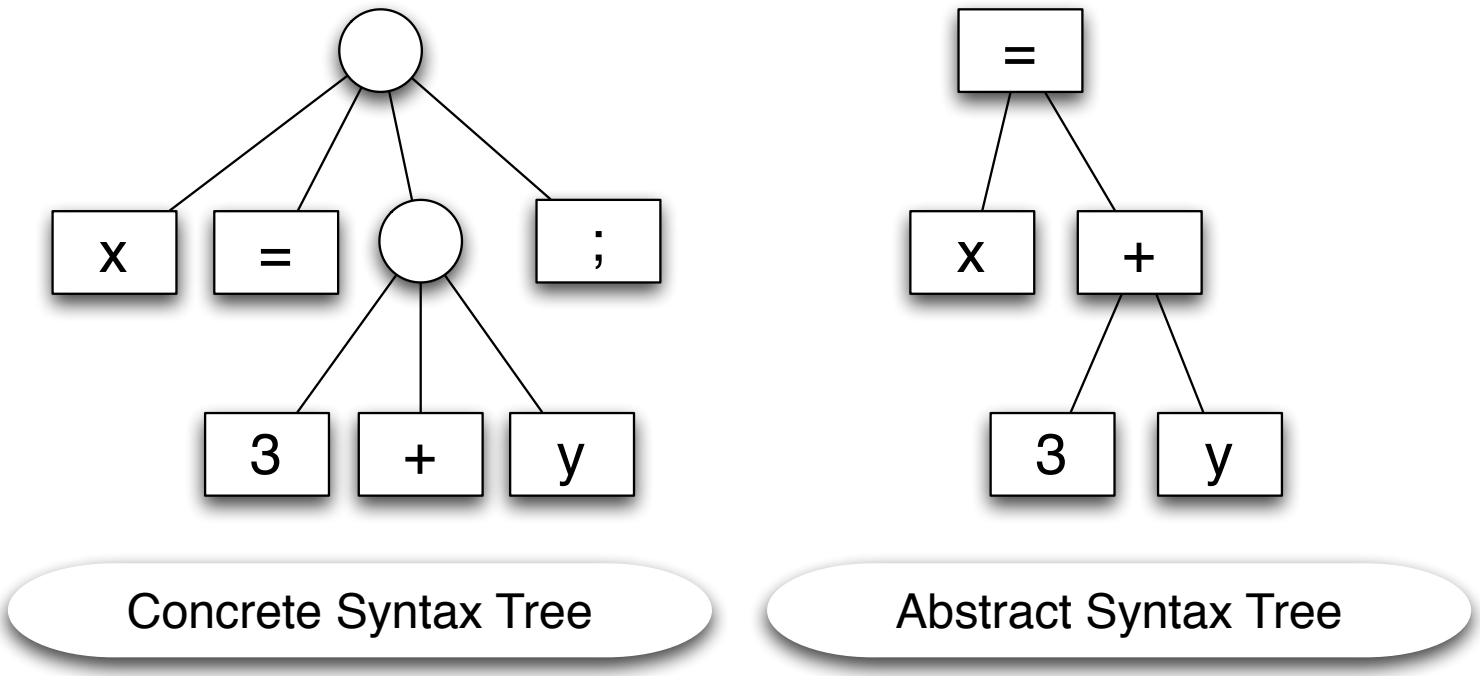
$$x = 3 + y;$$



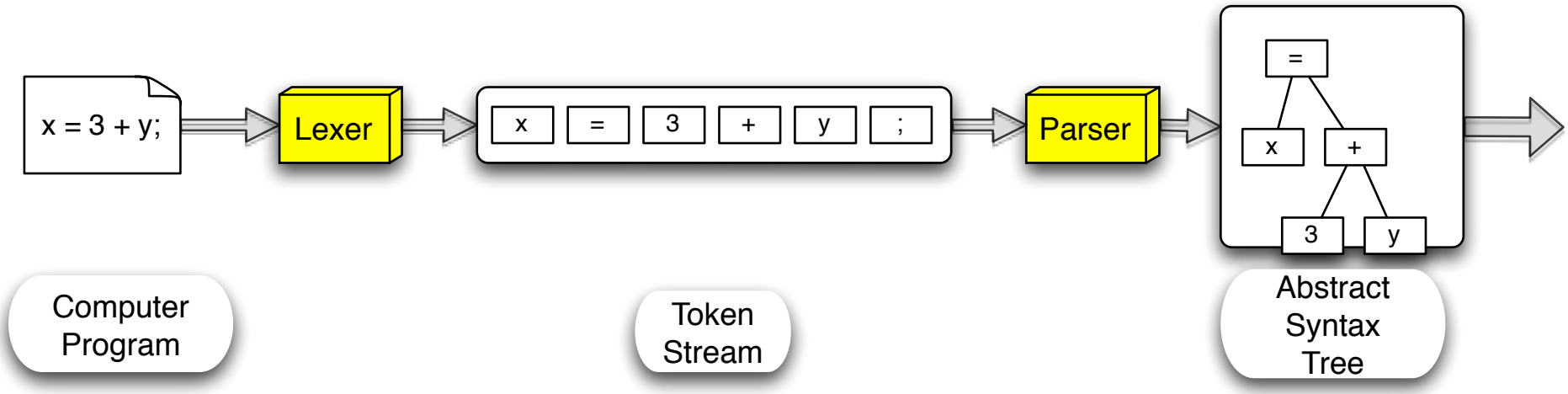
Concrete and Abstract Syntax Trees

Concrete Syntax Trees contains the original textual information

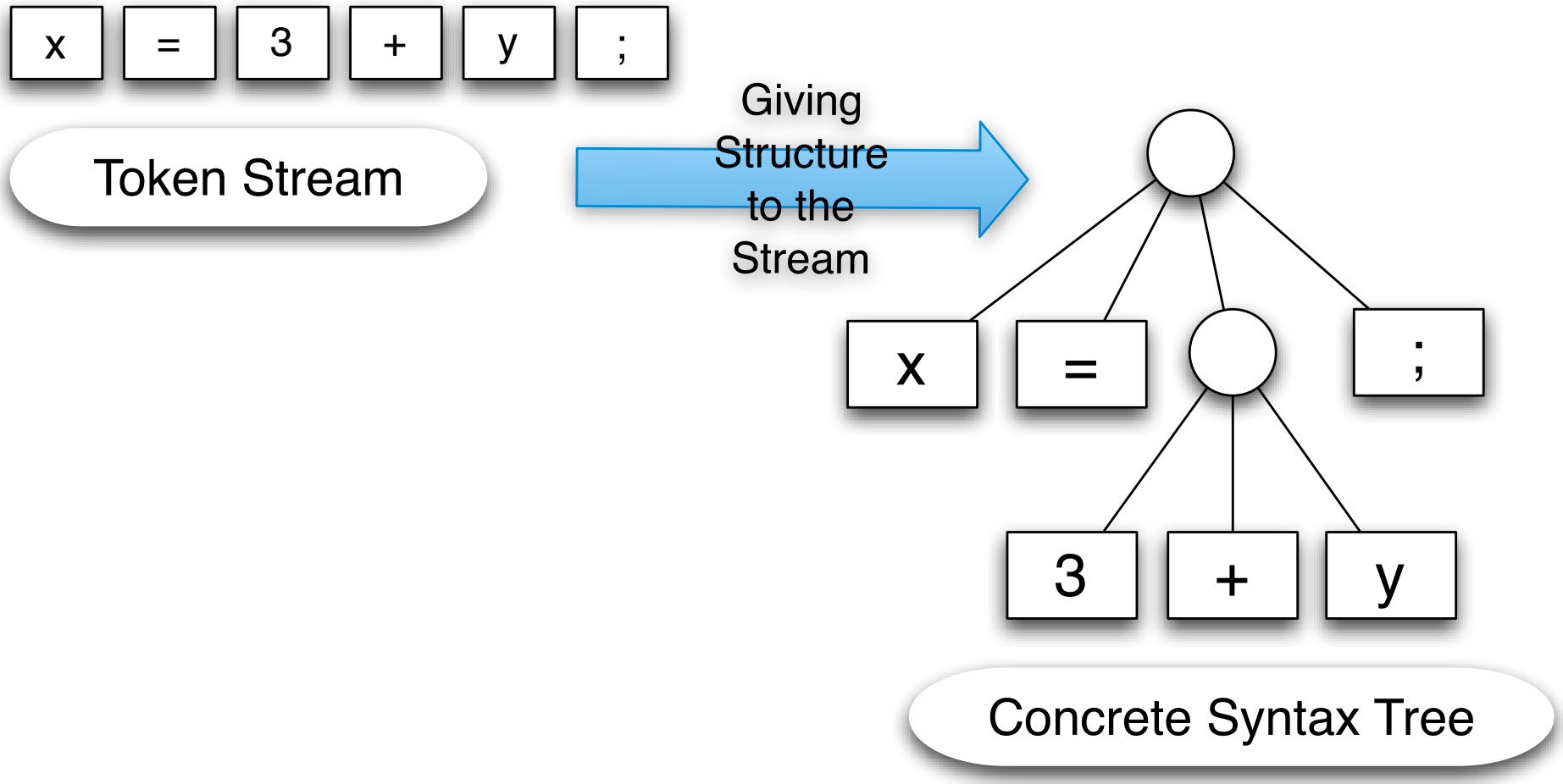
Abstract Syntax Trees have an abstracted, idealized view of a program



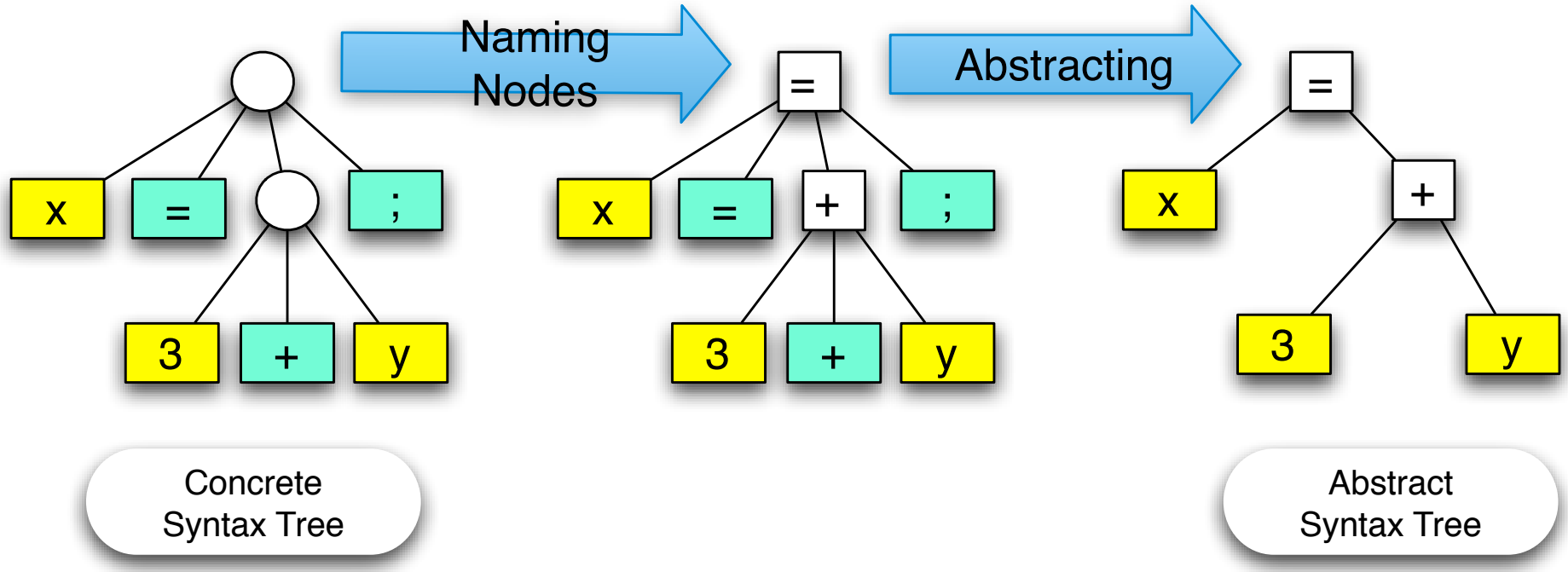
Generating Syntax Trees



Discovering Structure in Tokens



Concrete to Abstract Trees



Reality Check

Inside real compilers, we

- Turn our computer program into a stream of tokens
- Understand the hidden structure of our stream using a notation that expresses valid concrete syntax trees
- When discovering this structure, we build an abstract syntax tree internally

Backus-Naur Form (BNF) is way of expressing valid concrete syntax trees

KU

Backus-Naur Form

- **Non Terminals** are **internal nodes** in a concrete syntax
- `S` and `E` are the **Non Terminals**
- **Terminals** are **leaf nodes** in a concrete syntax
- `id`, `=`, `;`, `num`, `+`, `*`, `(` and `)` are the **Terminals**
- `id` and `num` are special terminals, with a payload

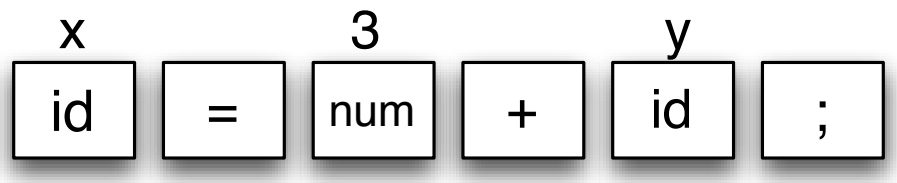
Example BNF

```
S ::= id = E ;
E ::= id
    |  num
    |  E + E
    |  E * E
    |  ( E )
```

Our token stream for our example

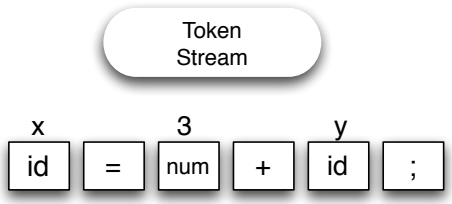
```
x = 3 + y;
```

Actually is a stream of terminals, with optional payloads.



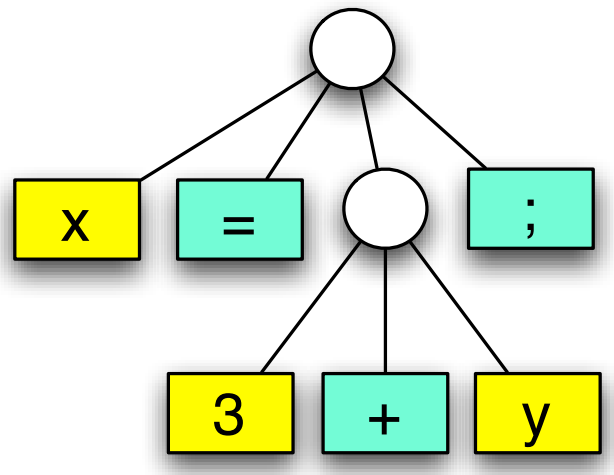
KU

Discovering Structure



BNF for Statements

```
S ::= id = E ;
E ::= id
    |  num
    |  E + E
    |  E * E
    |  ( E )
```



Concrete
Syntax Tree

How do we take take a token stream, and turn it into a concrete syntax tree?

Production Rules

$$S \rightarrow \texttt{id = E ;}$$

$$E \rightarrow \texttt{id}$$
$$E \rightarrow \texttt{num}$$
$$E \rightarrow \texttt{E + E}$$
$$E \rightarrow \texttt{E * E}$$
$$E \rightarrow \texttt{(E)}$$

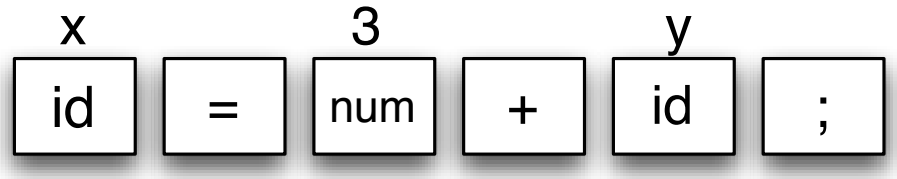
BNF for Statements

```
S ::= id = E ;
E ::= id
    |  num
    |  E + E
    |  E * E
    |  ( E )
```

Finding Structure

S $\rightarrow$ **id** = **E** ;

E $\rightarrow$ id
E $\rightarrow$ num
E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)

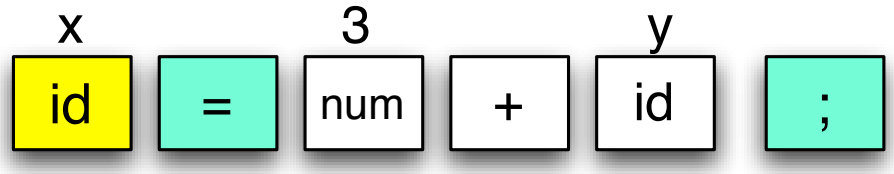


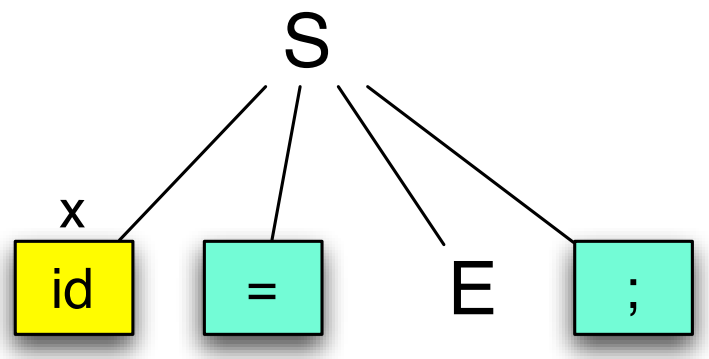
S

Finding Structure

S $\rightarrow$ **id** = **E** ;

E $\rightarrow$ id
E $\rightarrow$ num
E $\rightarrow$ E + E
E $\rightarrow$ E * E
E $\rightarrow$ (E)





Finding Structure

$S \rightarrow \texttt{id} = E\ ;$

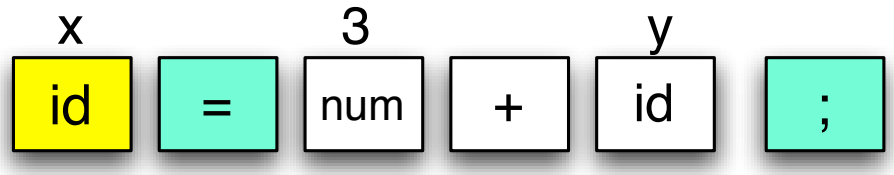
$E \rightarrow \texttt{id}$

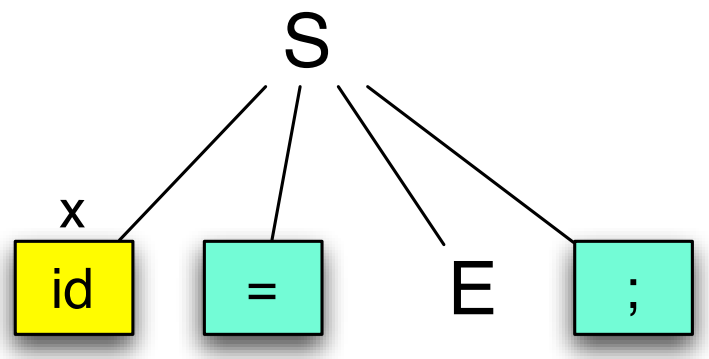
$E \rightarrow \texttt{num}$

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

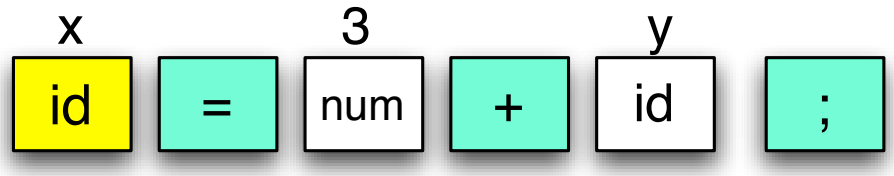
$E \rightarrow E * E$

$E \rightarrow (\ E\ )$





Finding Structure



$S \rightarrow \texttt{id} = E\ ;$

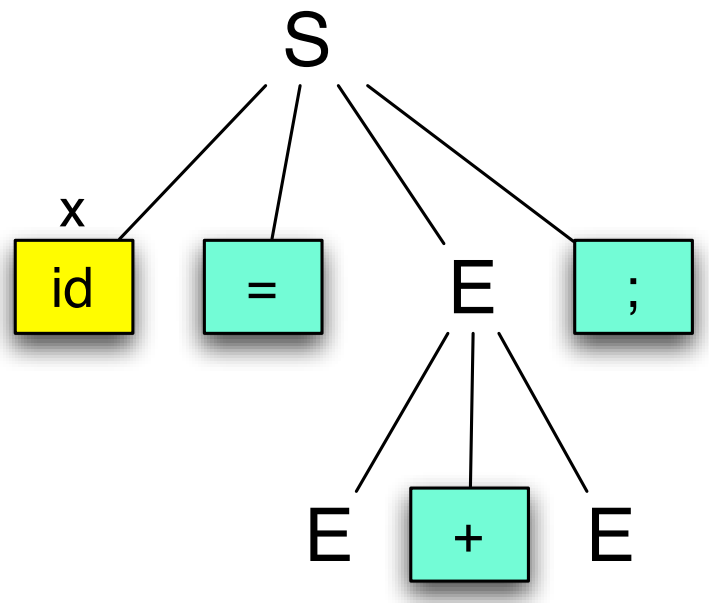
$E \rightarrow \texttt{id}$

$E \rightarrow \texttt{num}$

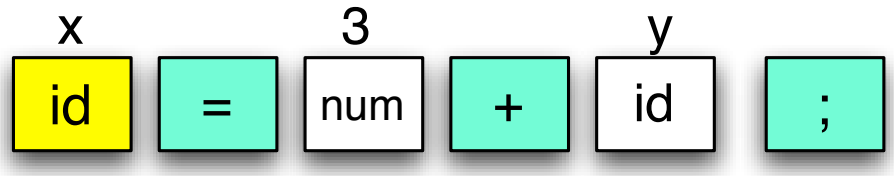
$\mathbf{E \rightarrow E + E}$

$E \rightarrow E * E$

$E \rightarrow (E)$

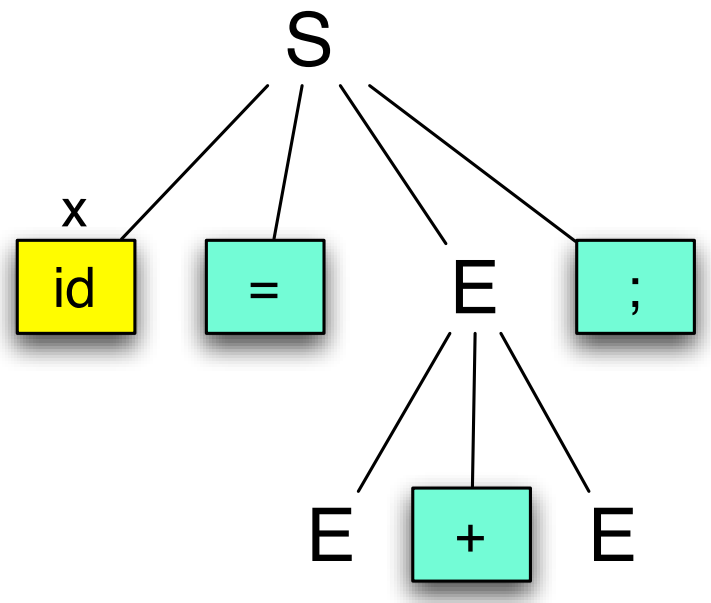


Finding Structure

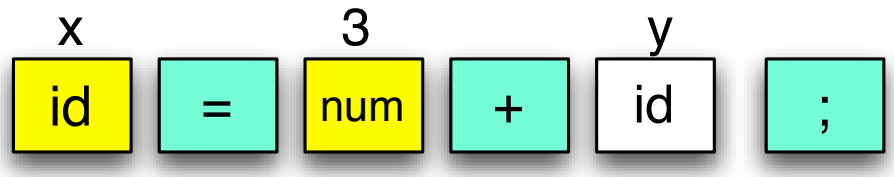


S → id = E ;

E → id
E → num
E → E + E
E → E * E
E → (E)



Finding Structure



$S \rightarrow$ id = E ;

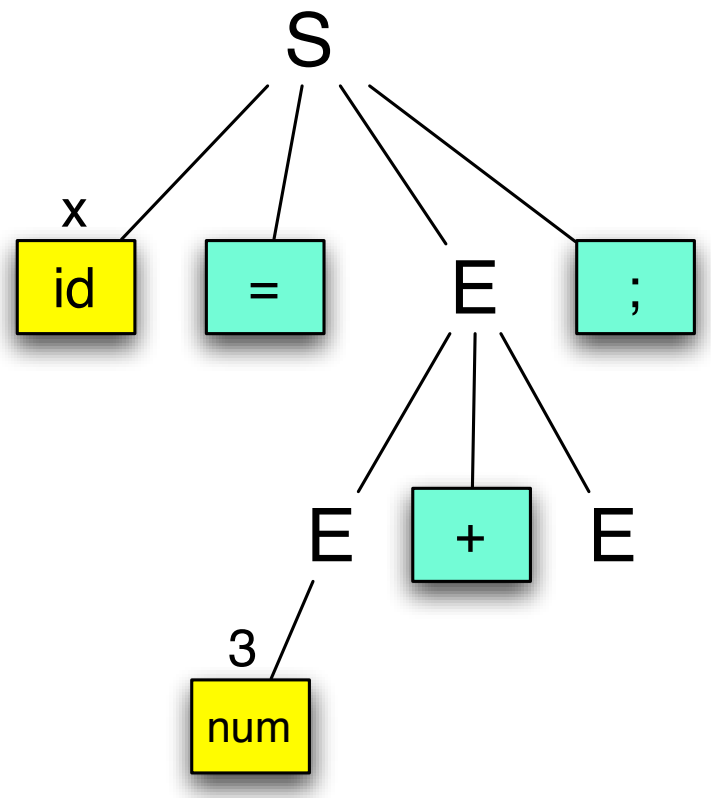
$E \rightarrow$ id

E → num

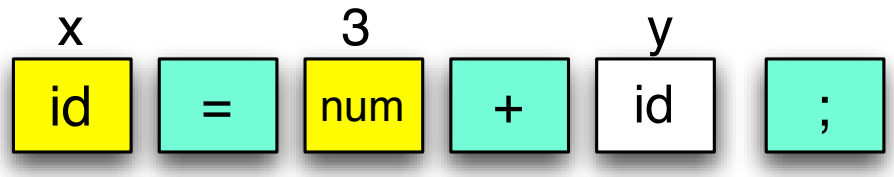
$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$



Finding Structure



$S \rightarrow$ id = E ;

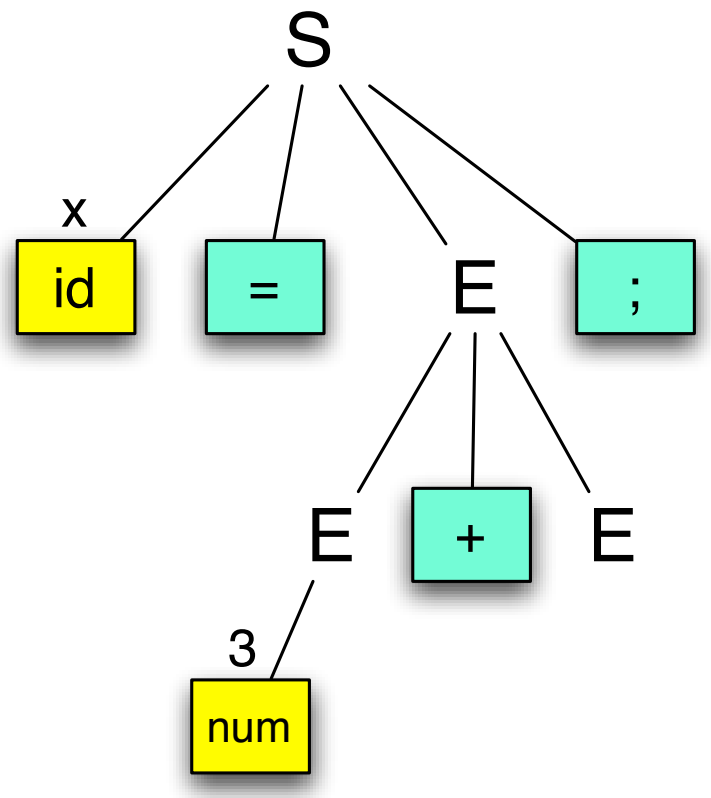
E $\rightarrow$ id

E $\rightarrow$ num

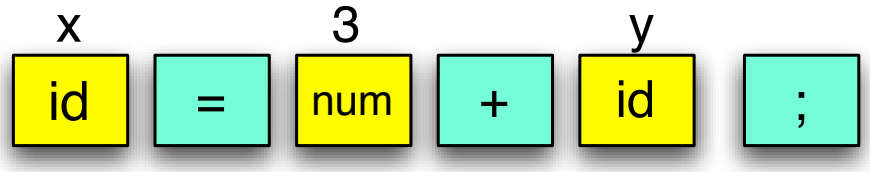
E $\rightarrow$ E + E

E $\rightarrow$ E * E

E $\rightarrow$ (E)



Finding Structure



S → id = E ;

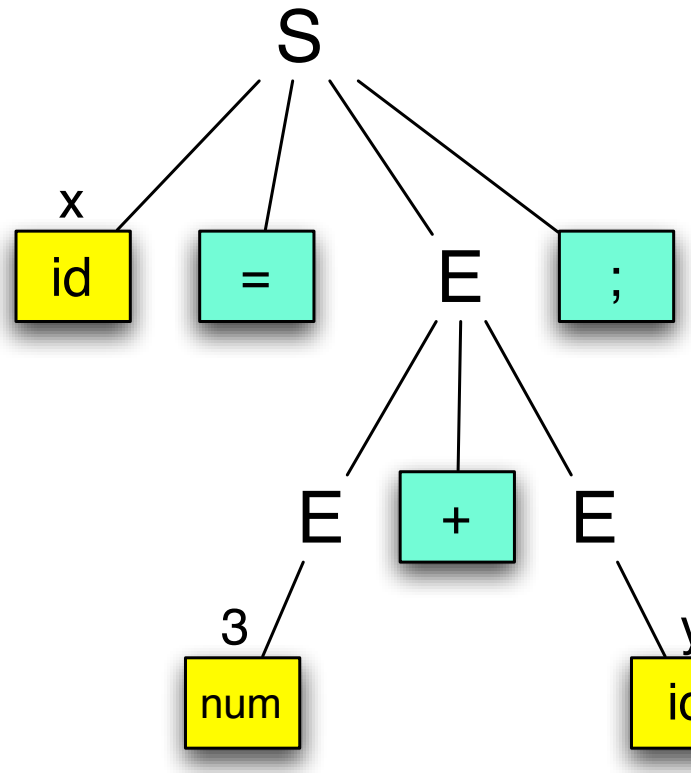
E → id

E → num

E → E + E

E → E * E

E → (E)



Summary

- Syntax Trees are used to represent programs
- BNF is way of expressing possible concrete syntax trees
- BNF can be used to guide turning a token stream into a concrete syntax tree

Exercise: Use our BNF example to find the concrete syntax tree

```
x = ( 2 * ( 3 * y ) ) ;
```

Example BNF

```
S ::= id = E ;
E ::= id
    |  num
    |  E + E
    |  E * E
    |  ( E )
```

KU