# Abstract Syntax

Mooly Sagiv

html://www.cs.tau.ac.il/~msagiv/courses/wcc20.html

# Abstract Syntax

- Intermediate program representation
- Defines a tree - Preserves program hierarchy
- Generated by the parser
- Declared using an (ambiguous) context free grammar (relatively flat)
  - Not meant for parsing
- Keywords and punctuation symbols are not stored (Not relevant once the tree exists)

# Topics

- Inductive Definitions
- Context Free Languages
- Example: Expressions

# Inductive Definitions

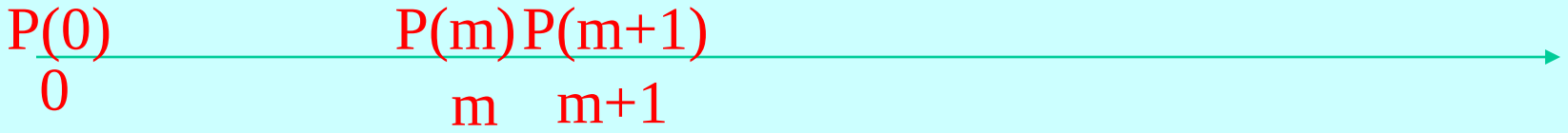
- Many programming langue features are defined by induction

# Induction in Programming Languages

- The syntax of programing languages is inductively defined
  - A number is expression
  - If $e_1$ and $e_2$ are expressions to so is $e_1 + e_2$
- Types are inductively defined
  - int is a type
  - if $t_1, t_2, \ldots, t_k$ are types then
    struct { $t_1$ $i_1$; $t_2$ $i_2$; …, $t_k$ $i_k$;} is a type where $i_1, i_2, \ldots, i_k$ are identifiers
- Recursive functions
  - fac(n) = if n = 1 then 1 else n * fac(n-1)
    $fac_1=1$, $fac_n=n*fac_{n-1}$

# Mathematical Induction

- P(n) is a property of natural number n
- To show that P(n) holds for every n, it suffices to show that:
  - P(0) is true
  - If P(m) is true then P(m+1) is true for every number m
- In logic
  - $(P(0) \wedge \forall m \in N.\ P(m) \Rightarrow P(m+1)) \Rightarrow \forall n \in N.\ P(n)$



# Course of values Induction

- P(n) is a property of natural number n
- To show that P(n) holds for every n, it suffices to show that for all m if for all k <m, P(k) holds then P(m)
- In logic
  - $\forall m \in N.\ (\forall k < m.\ P(k)) \Rightarrow P(m)) \Rightarrow \forall n \in N.\ P(n)$

# Context Free Languages

- Inductively define a set of words
- Terminals
- Non-Terminals
  - Start Non-terminal
- Rules
  - Non-Terminal → $Var_1$ $Var_2$ … $Var_n$

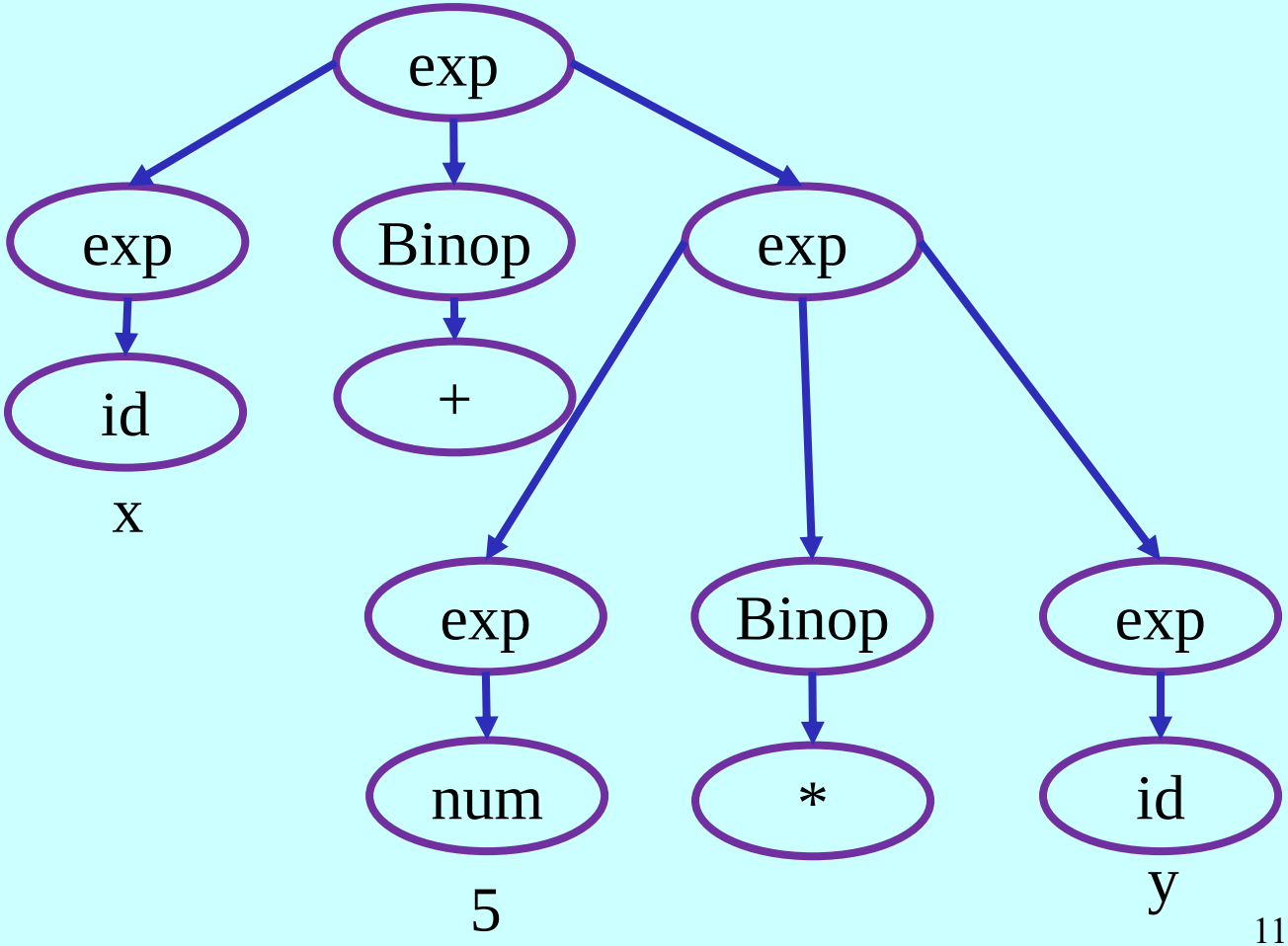
# Expression Definition (take 1)

- exp → **id**
- exp → **num**
- exp → exp Binop exp // binary expression
- exp →Unop exp //  unary expression
- Binop → +
- Binop → -
- Binop → *
- Unop → -

# Questions

- How to show that “x + 5 * y” is an expression
- How to show that “v v” is not an expression for any v?

# “x + 5 * y” ∈L(exp)

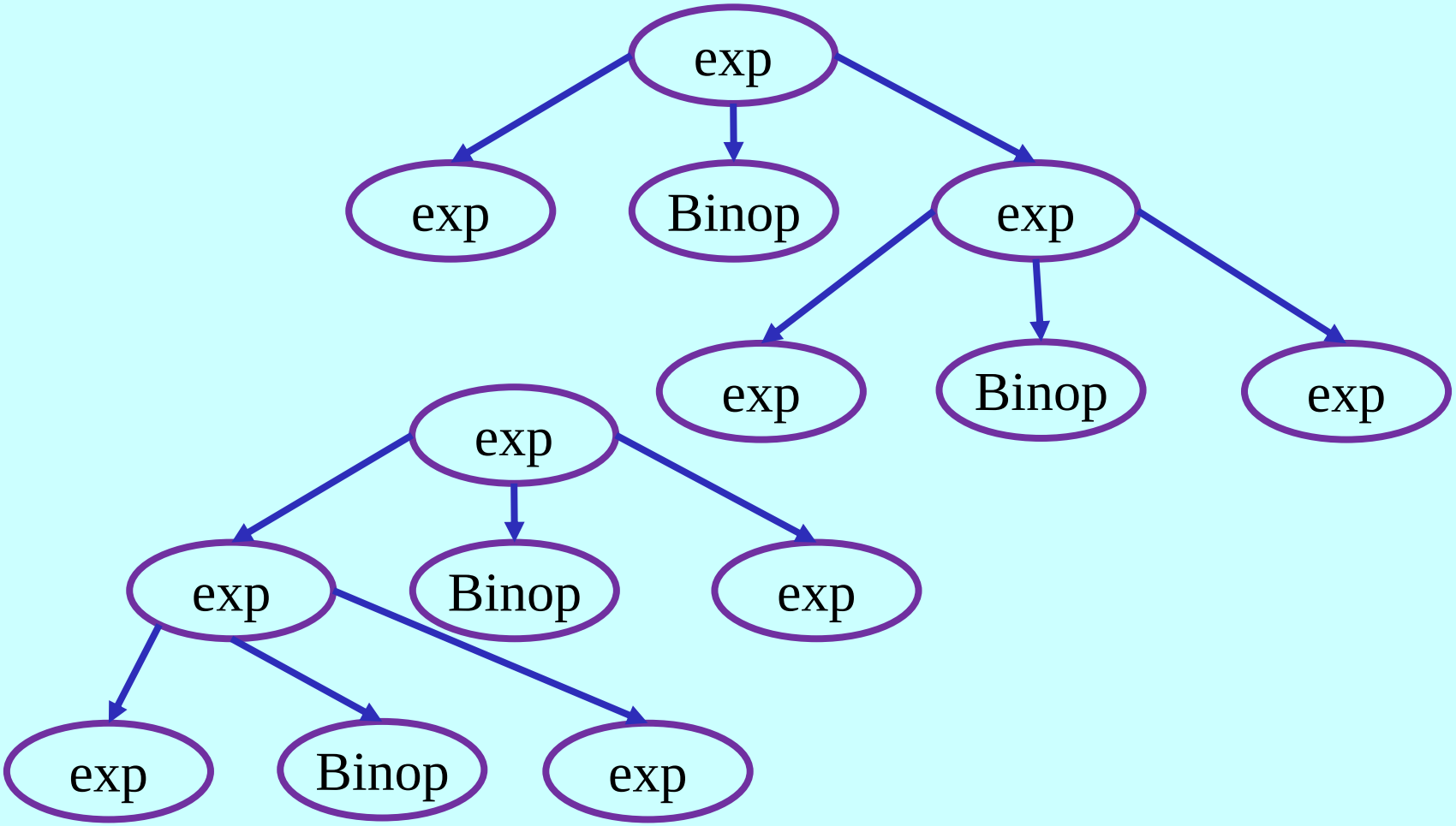


$v\ v \notin L(exp)$

# Ambiguous Context Free Grammars

- Two syntax trees
- [Two leftmost/rightmost derivations]

# Ambiguity in Expressions



# Non-Ambiguous Expression Grammar

- exp → term
- exp → exp + term
- exp → exp - term

- term → factor
- term → term * factor

- factor → -factor
- factor → id
- factor → num
- factor → **(**exp**)**

# Non-Ambiguous Expression Grammar (BNF)

- exp → term | exp + term | exp - term
- term → factor | term → term * factor
- factor → -factor | id | num | **(**exp**)**

# Abstract Syntax Tree

- Intermediate program representation
- Not meant for parsing
- Hides irrelevant details
- Defined by an ambiguous grammar

# Abstract Syntax for Arithmetic Expressions

| | |
|---|---|
| Exp → **id** | (IdExp) |
| Exp → **num** | (NumExp) |
| Exp → Exp Binop Exp | (BinOpExp) |
| Exp → Unop Exp | (UnOpExp) |
| Binop → **+** | (Plus) |
| Binop → **-** | (Minus) |
| Binop → ***** | (Times) |
| Binop → **/** | (Div) |
| Unop → **-** | (UnMin) |

```
package Absyn;
abstract public class Absyn { public int pos ;}
Exp extends Absyn {} ;
class IdExp extends Exp { String rep ;
        IdExp(r) { rep = r ;}
}
class NumExp extends Exp { int number ;
        NumExp(int n) { number = n ;}
}
class OpExp {
    public final static int PLUS=1;  public final static int Minus=2;
    public final static int  Times=3;  public final static int  Div=4;
}
final static int OpExp.PLUS, OpExp.Minus, OpExp.Times, OpExp.Div;
class BinExp extends Exp {
        Exp left, right;    OpExp op ;
        BinExp(Exp l, OpExp o, Bin Exp r) {
             left = l ;  op = o; right = r ;
            }
}
```

# Summary

- Abstract syntax provides a clear interface with other compiler phases
  - Supports general programming languages
  - Can be stored for large programs
- Automatically generated during parsing
- But the design of an abstract syntax for a given PL may take some time