

COMP 442 / 6421

Compiler Design

Tutorial 4

Instructor:

Dr. Joey Paquet

paquet@cse.concordia.ca

TAs:

Haotao Lai

h_lai@encs.concordia.ca

Jashanjot Singh

s_jashan@cs.concordia.ca



Contents

Concrete Parse Tree vs Abstract Syntax Tree

Examples



Concrete Parse Tree vs Abstract Syntax Tree

A parse tree pictorially shows how the start symbol of a grammar derives a string in the language.

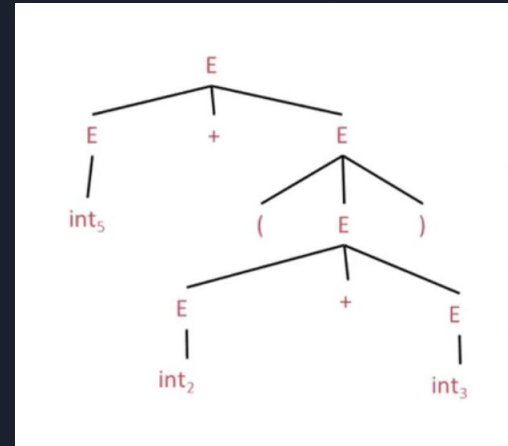
Whereas,

Abstract syntax trees, or simply syntax trees, differ from parse trees because superficial distinctions of form, unimportant for translation, do not appear in syntax trees.

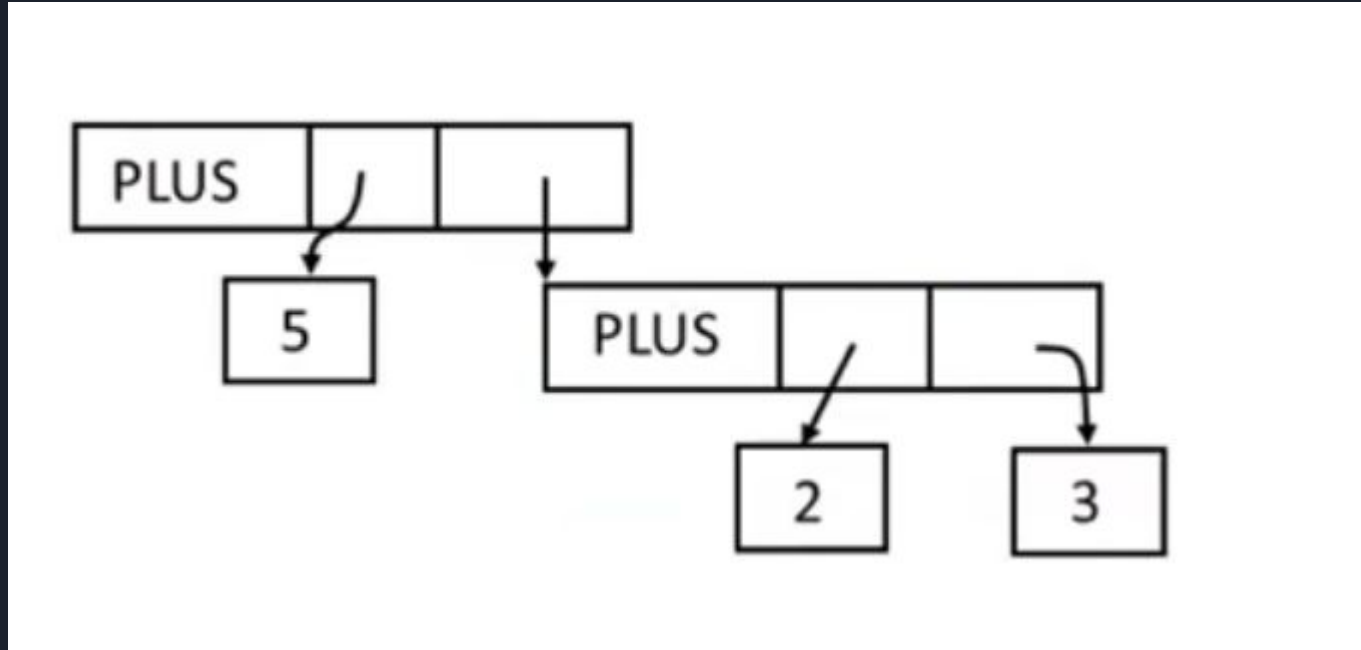
Also, they're most often expressed by the data structures of the language used for implementation.^[1]

Concrete Parse Tree vs Abstract Syntax Tree

- AST's ignore some details as compared to Parse Trees.
- For example:
- $E \rightarrow \text{int} \mid (E) \mid E + E$ and a String `5 + (2+3)`
- After lexical analysis we get, `int5 '+' '(' int2 '+' int3 '`
- But it does contain too much information.



Concrete Parse Tree vs Abstract Syntax Tree





How to construct an AST

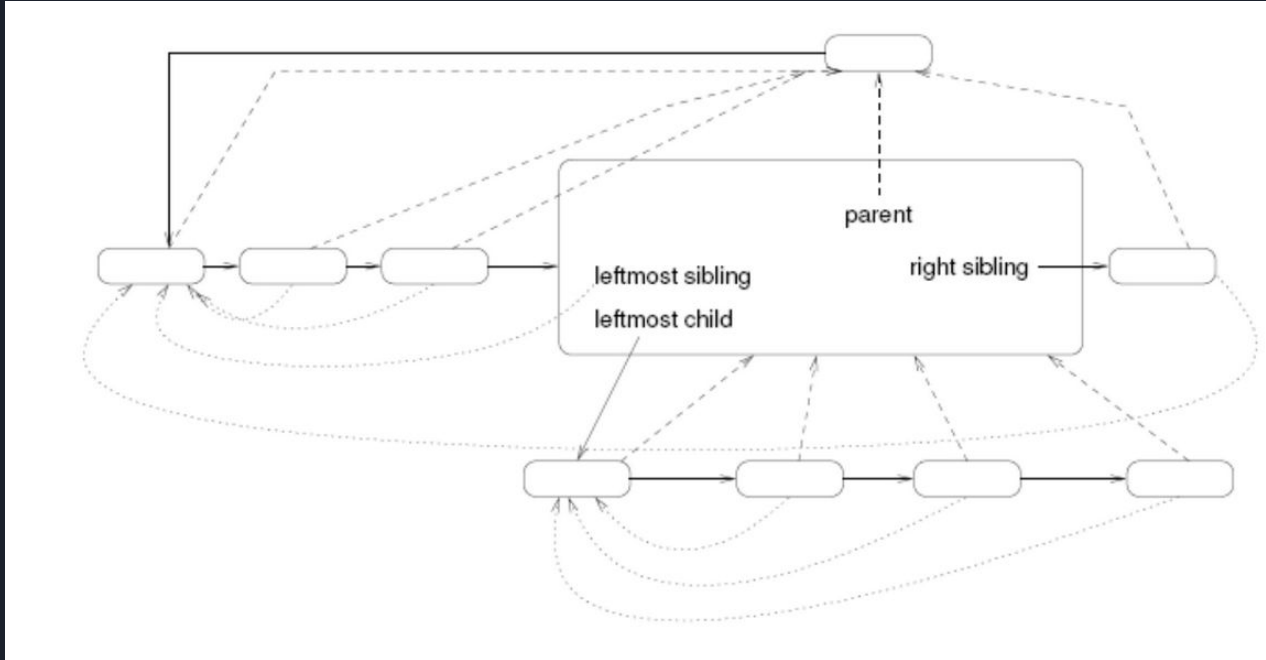
The AST structure is constructed bottom-up where,

- A list of siblings is generated and
- The list is later adopted by a parent.

Each node needs connection to:

- **Parent:** to migrate information upwards in the tree
 - Link to parent
- **Siblings:** to iterate through (1) a list of operands or (2) members of a group, e.g, members of a class, or statements.
 - Link to right sibling (thus creating a single linked list of siblings)
 - Link to leftmost sibling (in case one needs to traverse the list as a sibling is being processed).
- **Children:** to generate/traverse the tree
 - Link to leftmost child (who represents the head of a linked list of children).

Data Structure Design





How to construct an AST

```
class AST
```

- + static makeNode() -> AstNode
- + static makeFamily() -> AstNode
- + root: AstNode

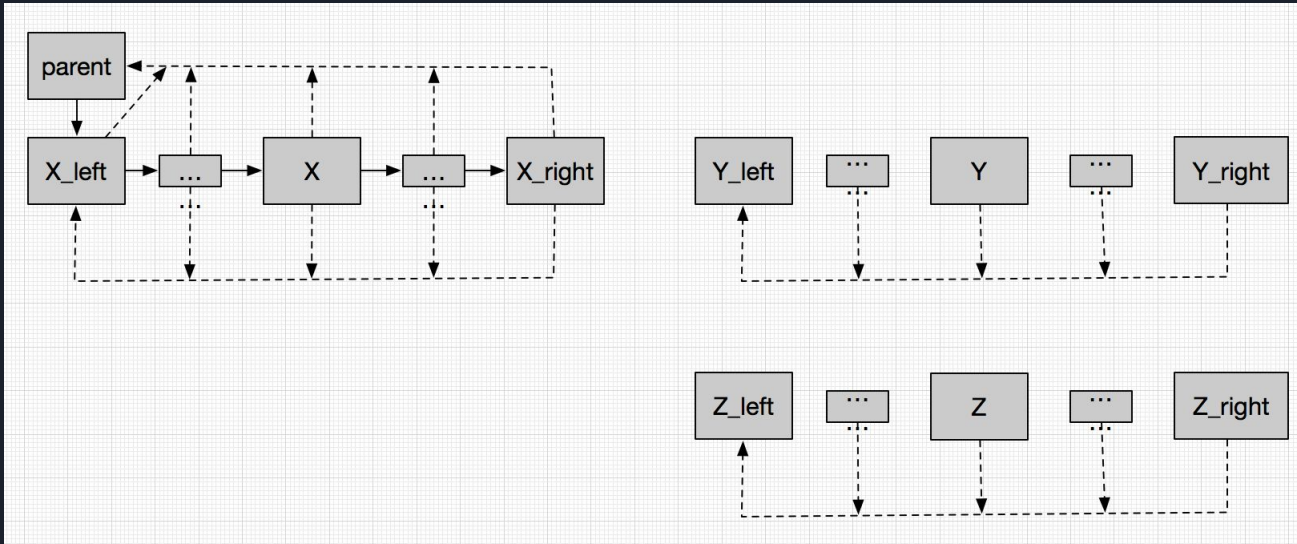
```
class AstNode
```

- + makeSibling()
- + adoptChildren()

** note: only the key fields and methods were shown here, you may need extra fields or methods*

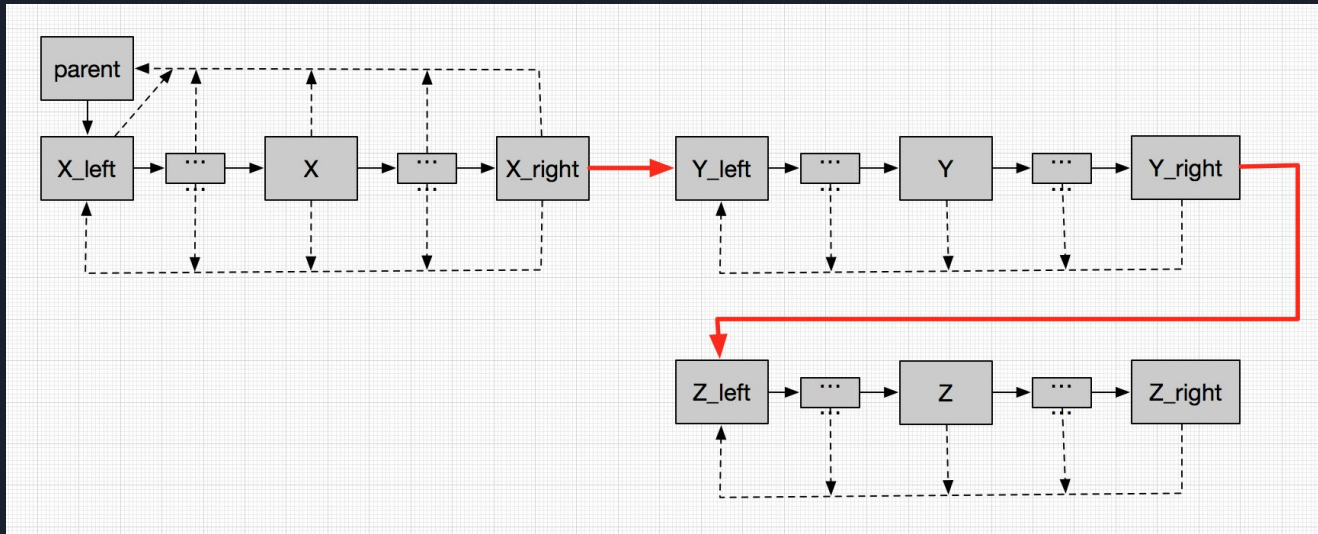
makeSibling()

assume we are calling: `X.makeSibling(Y, Z)`



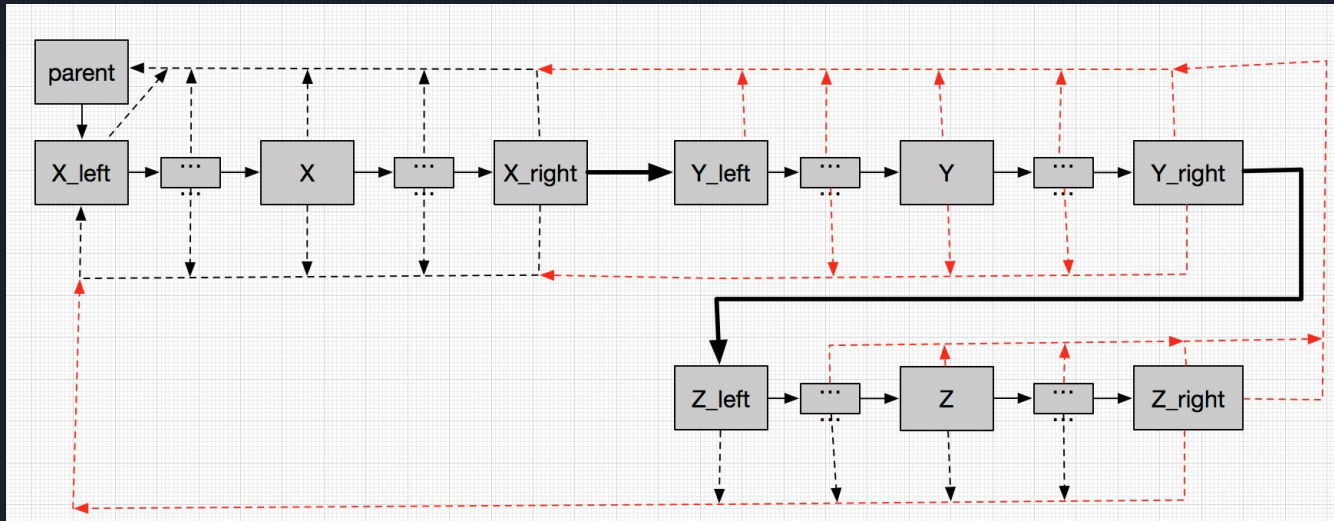
makeSibling()

find X's rightmost sibling, link it with the Y's leftmost sibling, treat them as a whole and link to Z's leftmost sibling (repeat until no more new unlinked)



makeSibling()

set all new added elements' parent to X's parent and all their leftmost sibling pointer to the X's leftmost sibling



adpotChildren()

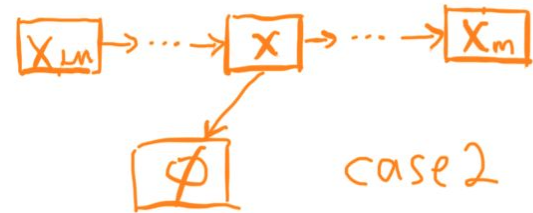
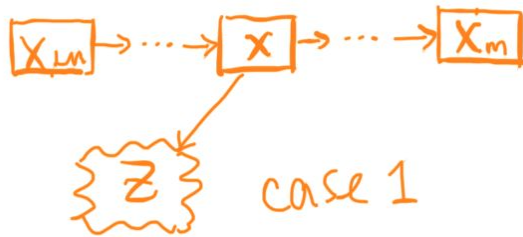
assume we are calling `X.adoptChildren(Y)`



adpotChildren()

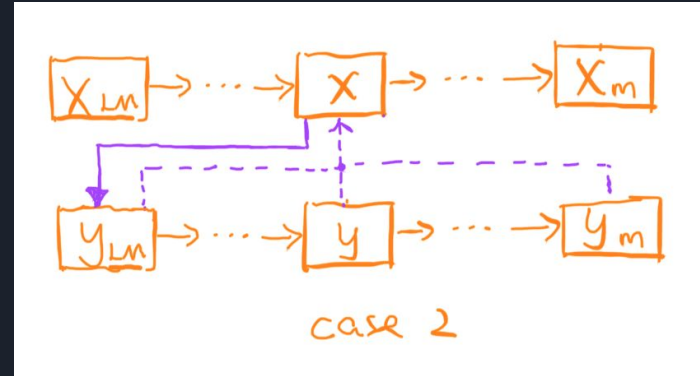
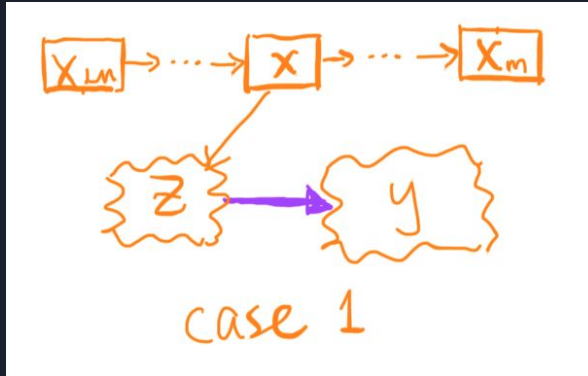
There will be two possibilities:

1. X already has a leftmost child
2. X don't have any leftmost child



adpotChildren()

1. just link Z to Y, do what we do in makeSibling()
2. link new element's parent to X and set X's leftmost child pointer to Y's leftmost sibling





References

[1] Compilers: Principles, Techniques, and Tools" by Aho, Sethi and Ullman

[2] <https://eli.thegreenplace.net/2009/02/16/abstract-vs-concrete-syntax-trees/>

[3] C.N. Fischer, R.K. Cytron, R.J. LeBlanc Jr., Crafting a Compiler, Adison-Wesley,
2009. Chapter 7.