AST 代表了抽象句式syntactic结构

每一个tree的node代表了源码中的一个语法构造syntactic construct

abstract代表着他不会像源码一样展现每一个细节

这一类细节在AST中被移除了因为他们并不传递信息

例如终结符Punctuations 符号例如逗号， 分号 括号 会被移除

又例如if-then-else这个结构会被人做一个node而不是3个

这就是AST与concrete syntax tree的具体区别，有一个更加具体

一旦建立，额外信息被加入AST，通过后续的代码处理步骤例如semantic analysis 或者code generation

目的，

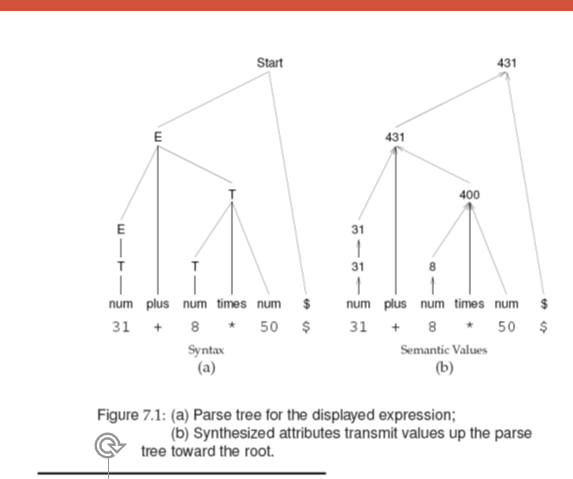
（1）在parsing的时候聚合aggregate信息，以得到更好的理解关于整个语法结构syntactic constructs

（2）用数据结构代表整个程序，在接下来的步骤中可以被重复使用

叶子代表着具体的语法信息

中间node代表着更高层次的概念，通过aggregation子树传达的信息 //就是概括子树信息

ROOT NODE包含了所有信息



左边是当前的ADT，右边是对应的第三次作业的语义部分

AST是从下往上构建的

一连串siblings node被生成然后被push到一个semantic stack语义stack上

这串nodes在这之后会从semantic stack上被pop出来，然后被对应的Parent node领养

AST结构允许在list的两端添加siblings节点

有这些AST node需要固定数量的children

例如运算符号，（binary, unary） if-then-else语句

有些AST nodes可以有零个或多个children

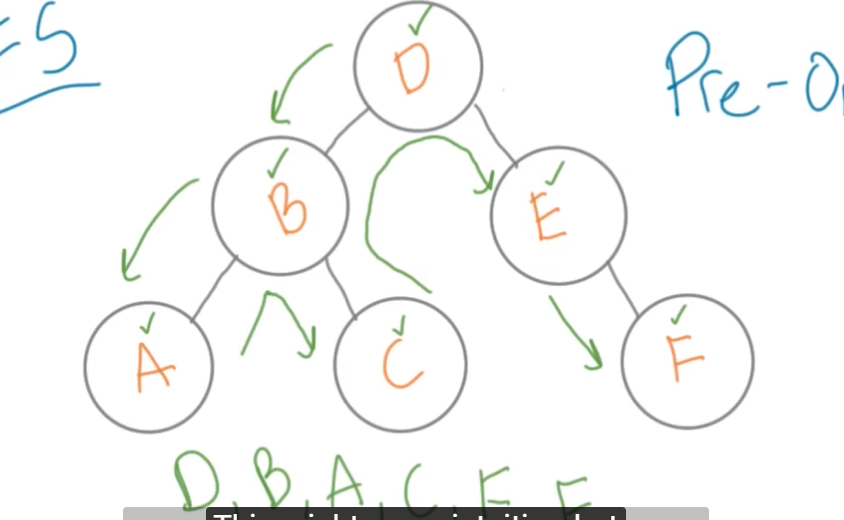
parameter lists(参数信息，可以由多个参数)

花括号内的语句（任意个句子）

一个class的members

所以AST node data应该允许任意数量的children node

使用的是depth-first-search tree traversal



对于任何一个节点，如果有左节点，call左节点， 然后自己，然后call右节点

每个node都需要链接：

1.parent， 为了把信息上移

2.siblings: 为了迭代（1）一串operands加减左右的数字（b）一个group的members

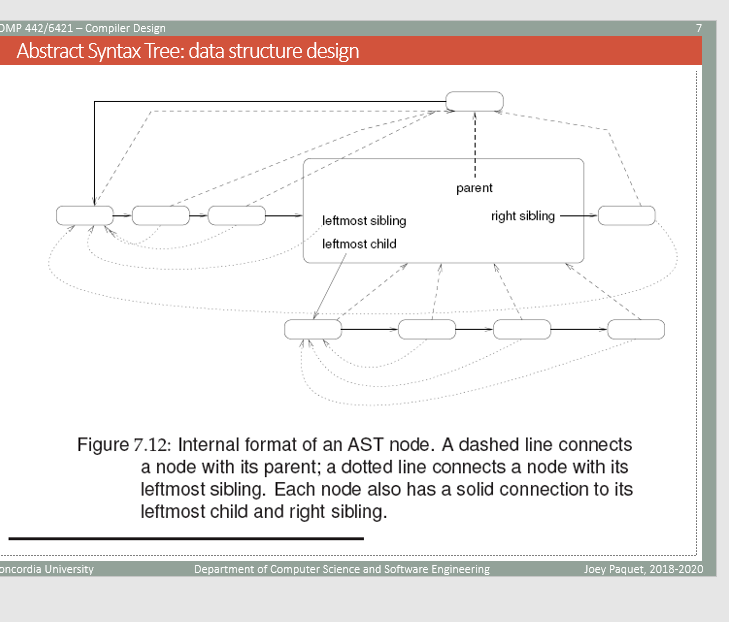
例如class的members，一个statement block的stattements

与右边的sibling相连（这样就能创造一个Linked list）

或者与**最**左边sibling相连（如果有一个node需要遍历整个list）

3.children:为了生成或遍历整个树

与最左的child相连，（代表着children Linked list的head）

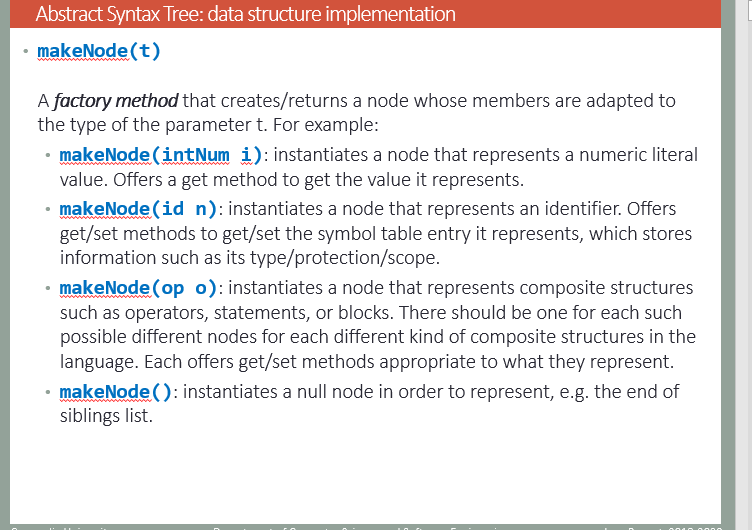


这是一个node的内部结构（因为一个node是抽象的，并不代表所有信息）

虚线连向父母， 点线与最左sibling相连，实现是最左child以及右边sibling

makenode(t)

一个工厂模式用来create一个node， 取决于参数type

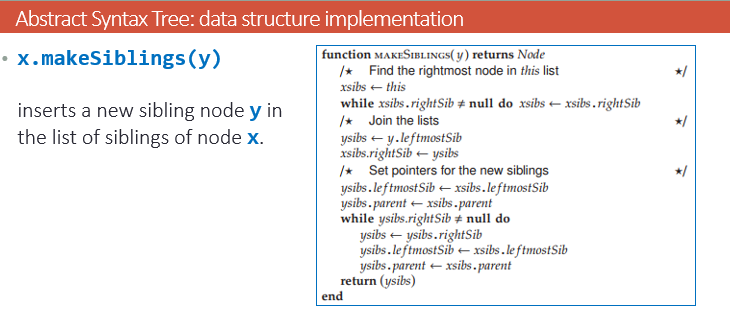


num会有一个get method来得到数字

id 需要get set method

op 创造一个node代表着组合结构， 例如operator，，block，等等， 提供get set

普通的make node， null, 在siblings list的末尾



make siblings

插入一个新sibling node，

先让当前X node作为xsibs

然后用while，如果右边不为空，就一直往右，一直到最右端，让x所在List的最右端作为被插node

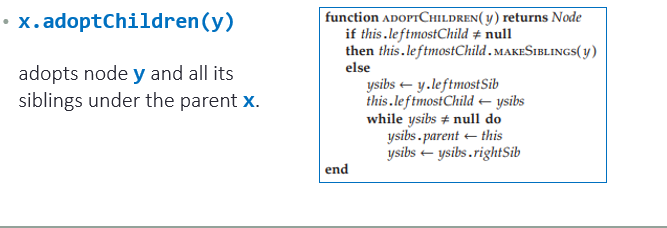
**注意插入ynode是指把他所在的list与x所在的list对接**

因此我们不能直接插入Y，而是要让Y的leftmostsib作为插入node

然后让最右端的右端被Y链接

pointer设置：让Y的最左边sib被x的最左边sib覆盖

y的parent被x的parent覆盖

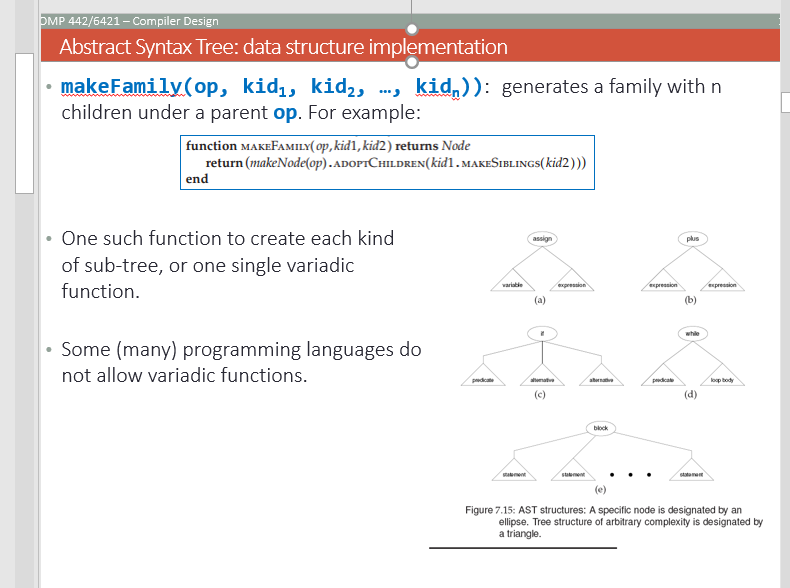


adoptChildren， 把node y以及所有sibling作为 children

如果这个X的最左child不为null(有child),然后让这个child make siblings

不然就先找到Y的最左边，让这个的mostchild等于这个node

然后设置pointer

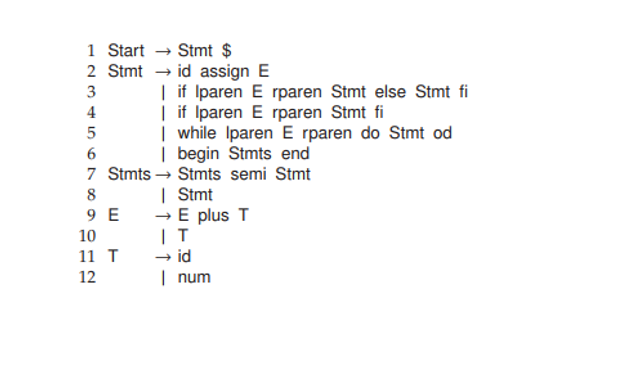


生成一个family，曹组符号为op，子为Kid

实现方法：

kid1.makesiblings

makenode(op),adopt children(kid1)



stmt:something

semi:semicolon

用syntax-directed translation生成AST

Language Semantic concept （语言语义概念），是构建程序的基础部件

值，function描述,数据结构

在AST中，每一个concept都是一个node

Atomic Concept（CA）是 简单concept结构， 由leaf node表示(CAN)

composite concept(CC)复杂concept，代表的是subordinate concept(CS)的聚合

composite concept由一个AST子树表示（这个子树还有多个子树作为children）

常规步骤，

实现Ca 原子结构

从最底端的parse tree leaf开始（存储语序信息的地方），使用makeNode() ,来生成AST node CaN来表示原子结构

把semantic information加入Can

把Can push进semantic stack

实现Cc复合结构

只要parsing subtree收集到了所有需要的semantic information(对于这个Cc节点来说)

CALL makeNode() 生成AST node CCN 来表示复合结构

对于每个Cc的从属结构CS subordiante concept

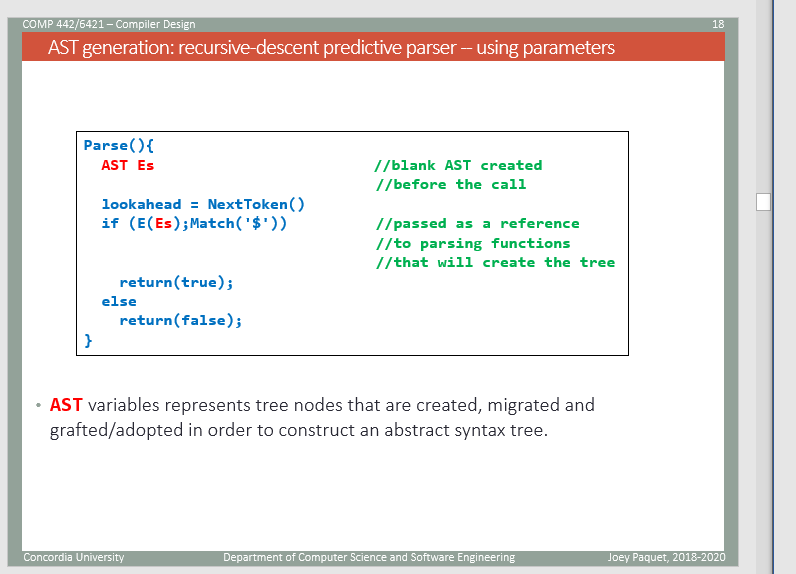
pop semantic stack的顶端，生态产一个CsN node代表Cs

让CsN成为CCN的child

把CcN push回semantic stack

当parsh 结束后， 整个semantic stack应该只有一个node，代表了整个AST

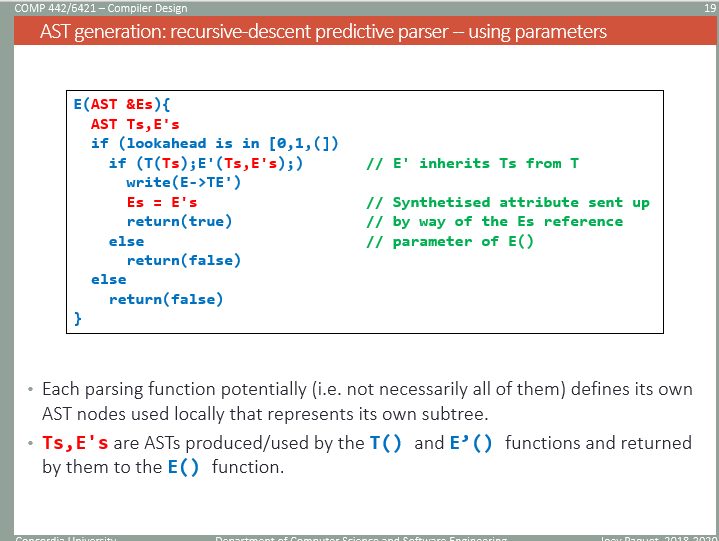
主程序



先开头就是创造一个AST

把Es作为参数，参数传递给第一个起始字母

AST就代表着 tree nodes

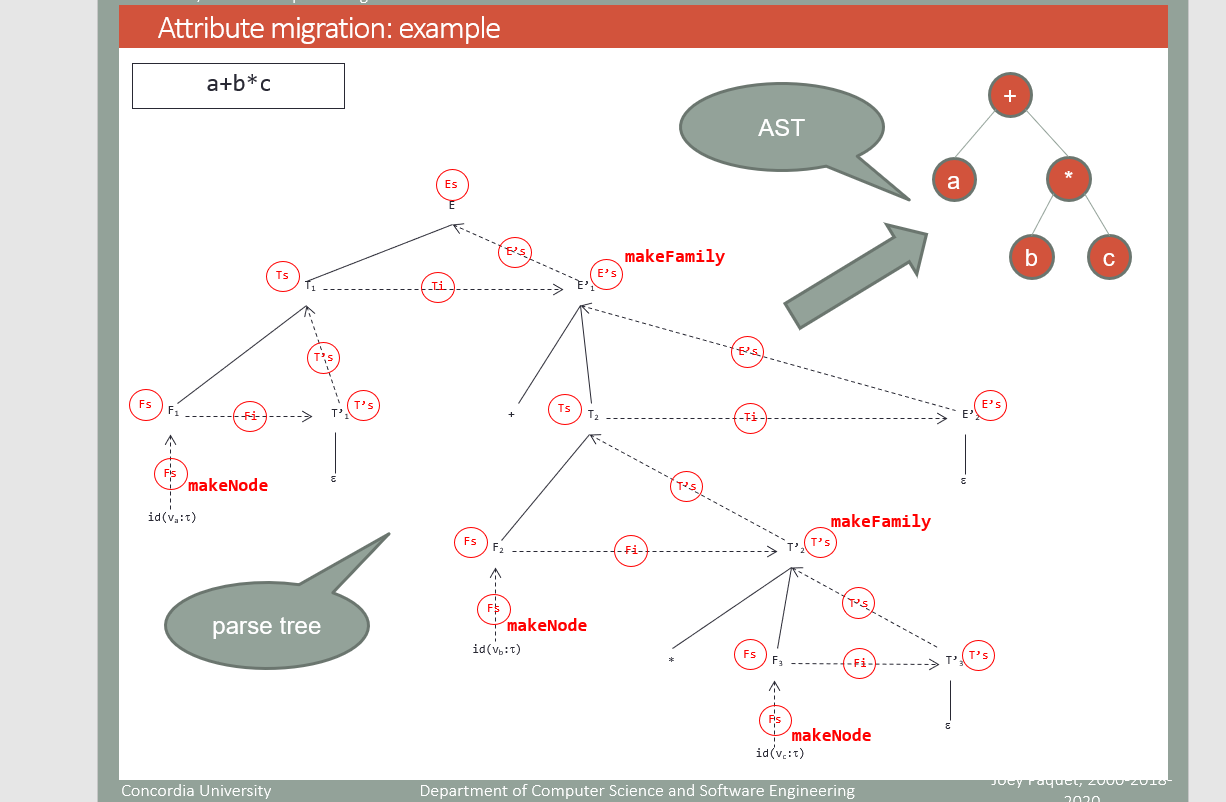


每个都要创造对应AST

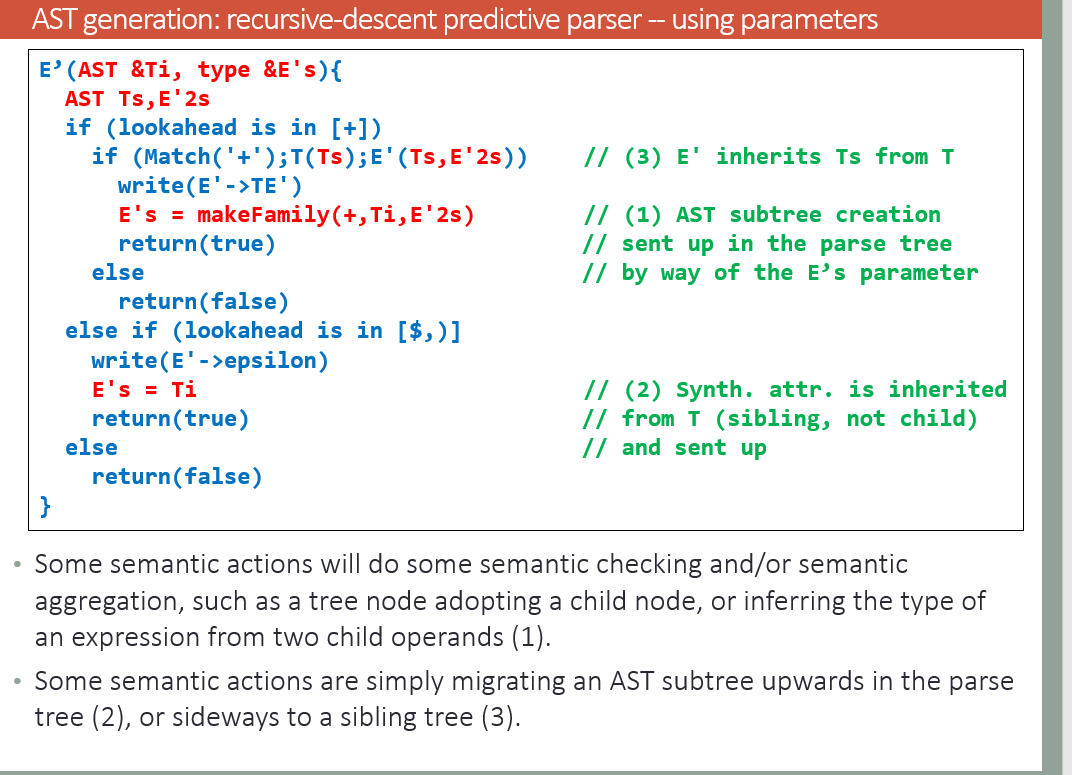
创造子AST

参数就是要从哪里得到，要改变那里

E's要从Ts得到信息， Es要从E‘s得到信息’



下标用的是同一个产生式，为了区别开，

这里新的东西出现了，有Operator的需要 makefamily

