syntax-directed translation:



我们要把token先用grammar转换成parse tree,再把parse tree用sdt转换成abstract syntax tree

有很多英文概念：我们先要搞明白

Syntax-directed translation:语法制导翻译 把Parse Tree变成abstract syntax tree

translation process翻译过程是由syntactic结构负责的，也就是assignment 2 的结构

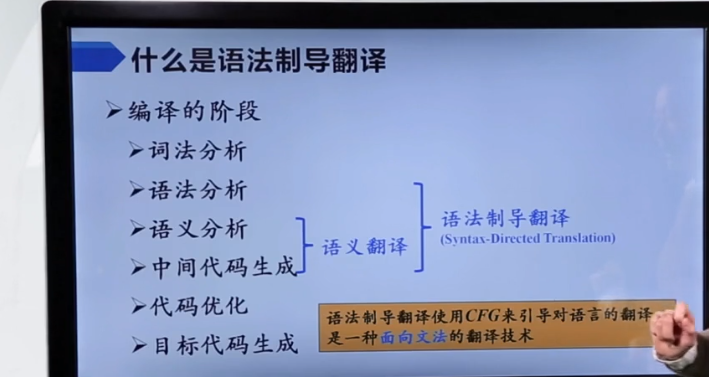
Semantic analysis: assignment 3 的步骤

syntax-directed translation 由程序的syntactic structure所驱动，这个syntactic structure是我们的parser生成的 //就是parser生成的parse tree

在syntax-directed translation中， 分为analysis and translation steps 而他们又能分别切割成两部分

分析部分：analysis(syntactic句法的, semantic语义的)

翻译部分：synthesis (translation and optimization最优化) //synthesis，生成，就是把原来的东西翻译成中间代码的部分



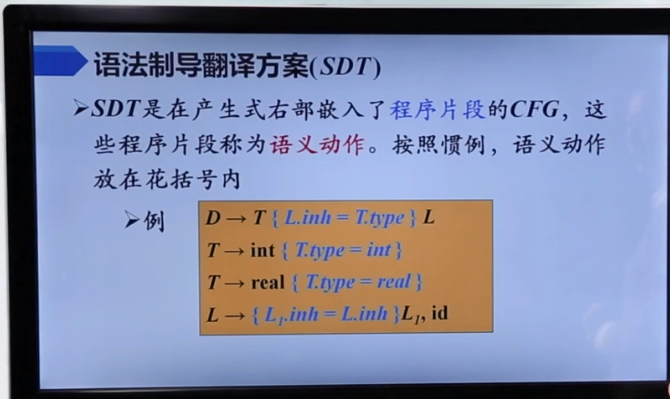
semantic analysis是 analysis与synthesis之间的桥梁, synthesis的translation部分 是积极语义分析的条件之一

Syntax-directed translation 的处理过程目的是让syntax analysis阶段，semantic checking阶段，translation阶段产生非常强的关联性//inducing 耦合， coupling，纽带

在这个project中，我们将用syntax-directed translation把Linear token stream转换成abstract syntax tree，这个tree将在后两个assignment中被使用

语义动作：Semantic actions

Semantic actions:被集成在parsing process中： be integrated in 被集成在



括号内就是语义动作 ,parsing process就是我们原来的构建规则

意思就是当遇到T以后，我们就可以用T的type值计算L的inh属性

遇到T->int后，我们就能让T的type为int

最下面的，在转换L1之前，我们就能根据L的Inh属性计算L1的InhInh属性

语义动作

semantic actions是一组function，这些function的目的是在parsing阶段结束以后能够

accumulate积攒并处理关于程序的semantic information //在Parsing时处理一下信息为semantic analysis做准备

在parsing不同的syntactical词法规则的时候，不同的信息被收集了起来

这些信息被称作semantic values语义信息，被聚集并累计在semantic records中，直到被下一步的语义动作处理

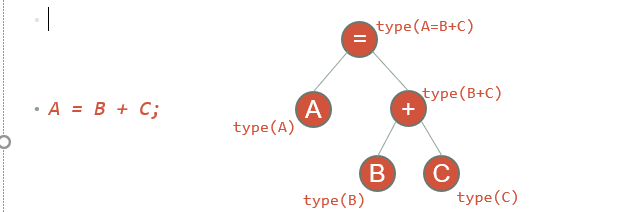
有一些信息使用了多个syntactical rules

这就需要通过解析树迁移这些信息，这个过程称为属性迁移 attribute migration。

在这个project中，我们希望在parsing阶段累计语义信息

并且把他们以subtrees的形式表现出来//例如variable declaration, function declaration, class declaration, statement ,expression等等

有些语义动作 (用semantic routine实现）通过执行semantic aggregation语义聚合或检查需要语义聚合的生产动作来完成额外的分析阶段， 这取决于语义规则



语义动作把信息整合，并且最终生成一个意义(translation) ，例如这里，最上面的点aggregate了所有的子节点，并且有了意义

因此semantic routine是compiler的核心

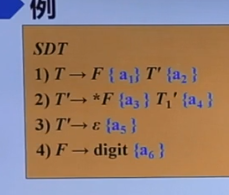
概念上来说，语义动作应该绑定在parse tree的各个节点

.//规则的左边就是父节点，规则的右边就是子节点们

Leave: 在树的叶子部分的语义动作通常收集语义信息，这些信息通常来自节点表示的token，或许查找symbol表

中间node:在中间的语义动作通常聚合aggregate,验证validate或者使用use这些语义信息，然后在树之间传递这些信息 （attribute migration属性迁移）

因为parse tree本身代表的是grammar rules，所以semantic actions也同样如此，**语义动作一般写在语法的右侧**



在recursive-descent predictive parsing中，实现语义动作的方式还是function , 在常规的parsing function中插入他们的call

实现竖线迁移attribute migration的方式是向parsing function传递参数，并且让给他return

在table-driven parser中，所有右侧的semantic action被当做像terminal nonterminal一样的东西 一起推入栈中。popping一个语义动作符就会进行一个对应的语义动作，就这样我们完成整个aggribute migration。有些Parser必须还有独立的semantic stack

recursive-descent prective parser也可以使用语义stack

semantic routines可以用attribute grammar来标准化·

attribute grammar拓展了传统的context-free grammar通过 attributes, 这些attributes代表着语义方面的一些性质例如type, value，correctness

在编写semantic routine语义程序的过程中，我们应该把semantic checking与semantic translation（code generation）区别开而不是混在一起写

semantic checking 是machine-independent而code generation不是，所以把它们分开写更好

Attribute，编程语言构建的性质，从最简单的leaf开始，慢慢迁移，直到最顶上的attribute收集全部信息， 也叫做semantic records

binding: 给一个attribute的值计算或赋值的过程

Static binding: 写代码的时候给他赋值

Dynamic binding: runtime的时候给一个attribute赋值

我们的Project只用关注static binding

Attributes migration

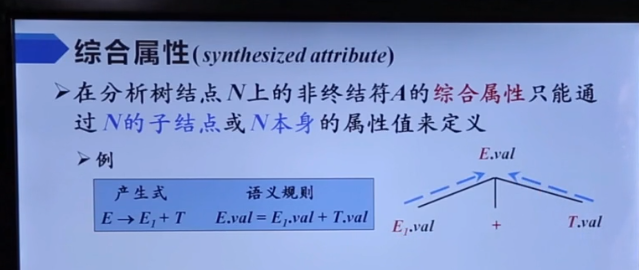
Static attribute binding通过在遍历parse tree时gathering收集（从底部收集信息）, propagating传播, aggregating聚合 所完成

Attribute在tree leaves处被gather，在tree nodes之间传播propagate，在一些父类node处被aggregated（当额外信息被提供）

Static attribute binding可以在parsing阶段完成

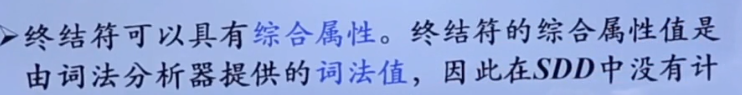
Synthetized attributes:综合属性 从child中gather到的attributes

inherited attributes: 从sibling中gather到的attribute

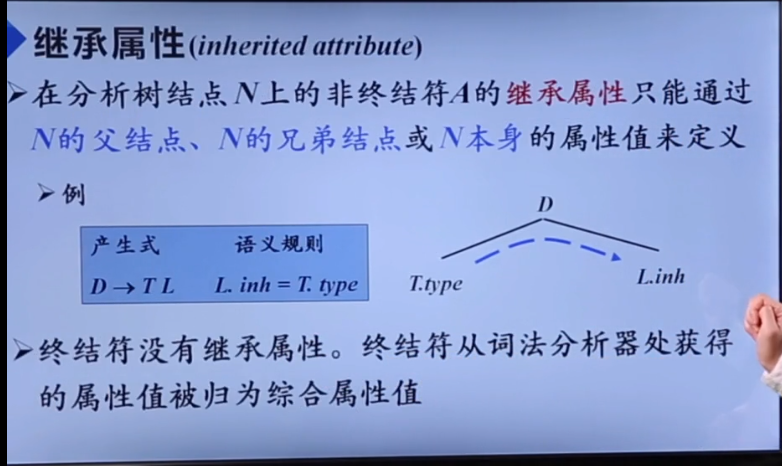


树节点的非终结符的synthesized attribute只能通过他的子节点或者他本身的属性来定义

E1就是E，为了防止混淆写成E1

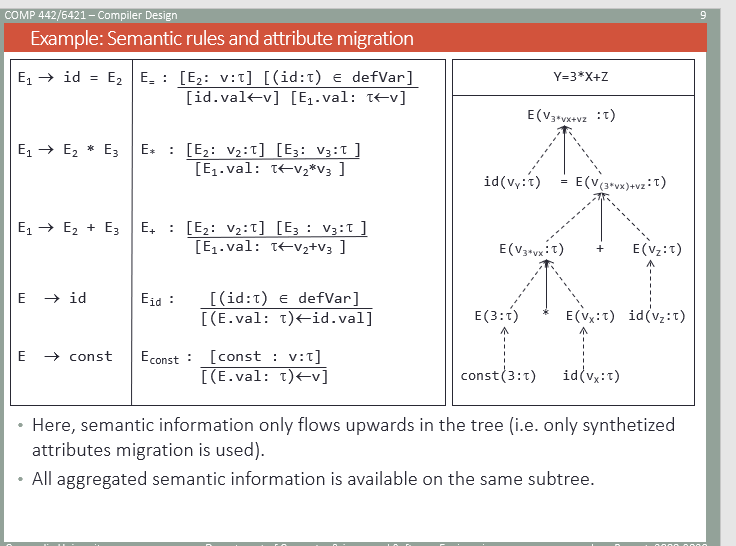


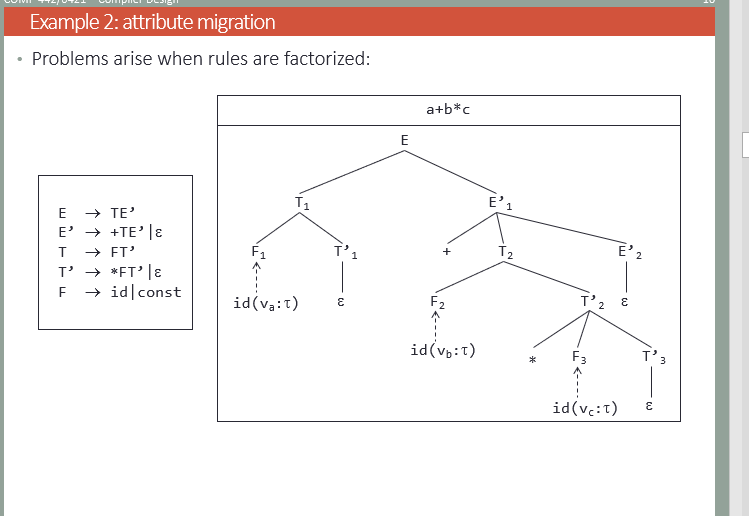
终结符的综合属性就是他自己



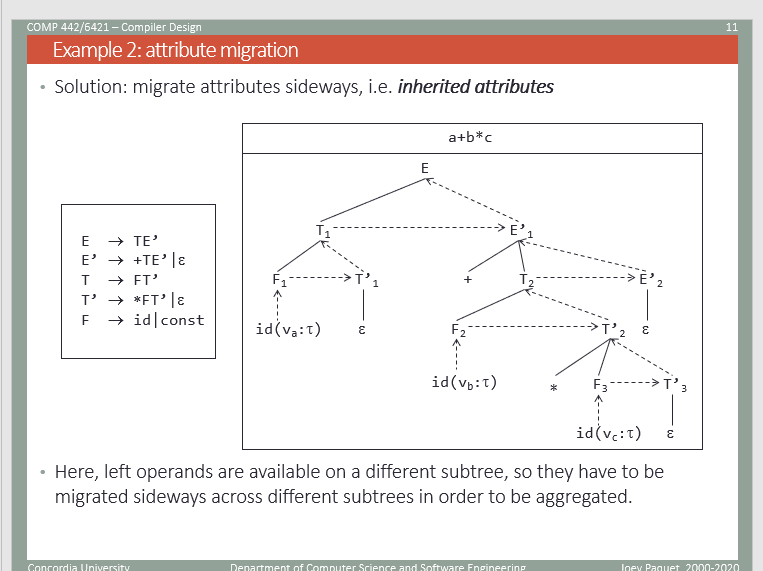
.继承属性：从兄弟节点与父节点获得的属性

终结符的属性是由词法分析器给的，算作综合属性

.这是一个自下而上的没有sibling传递的简单树

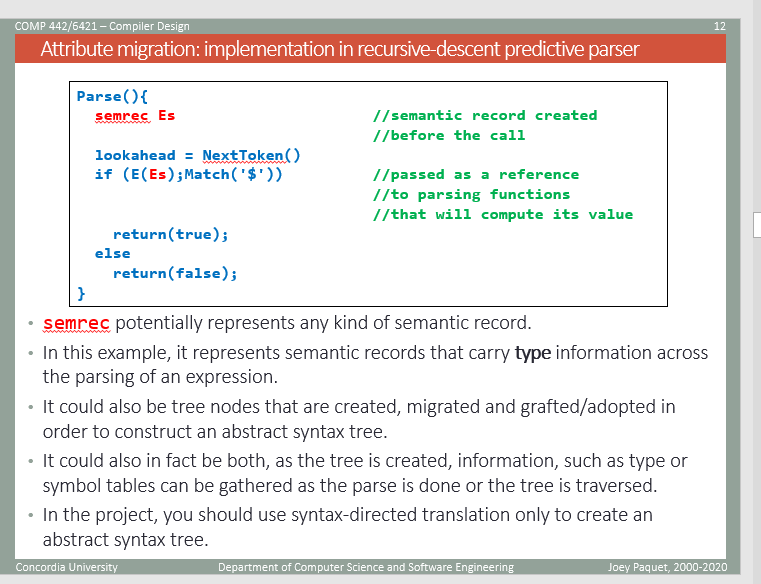


当按照grammar一一分解会出现新的问题



比如说这张图

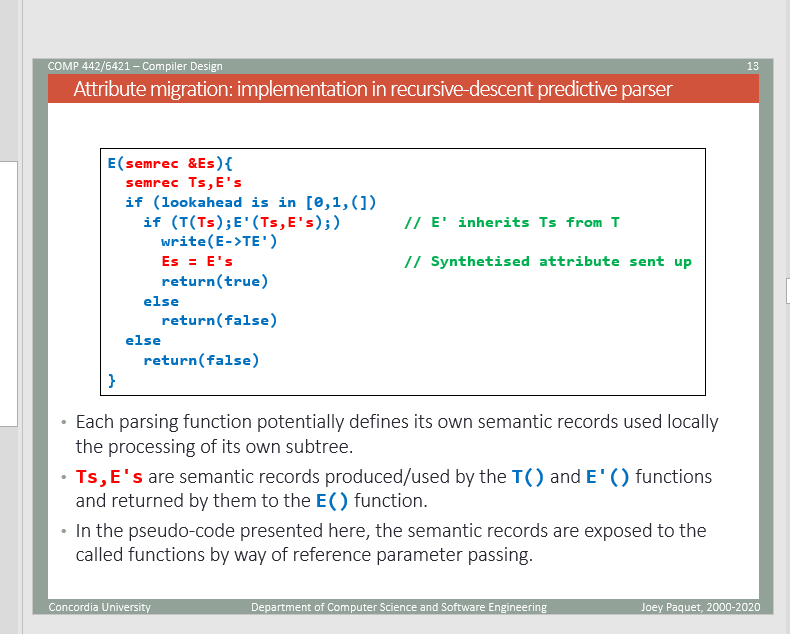
+号与乘号是一个binary 符号， 但是左边的operand 在另外一个subtree上，所以我们要进行属性迁移 //SIBLING传递



所以我们现在的parse第一步要先创造个semantic record 语义记录，

在这个例子里，semantic records在parsing一个表达式的时候存储了type信息

semrec也可以是被创造的tree nodes， 被migration转移已建立一个abstract syntax tree



每个parsing function 都可能自己定义semantic records 通过locally处理自己的子树

TS 与E' S是与T() 与E'() 有关的semrec(由他们才产生或被他们使用)，然后你要用E（） function来return 他们 //节点A当地创建自己子节点的record，继承父节点的record

在这里的伪代码 semantic records作为参数传递，function会改变参数的值

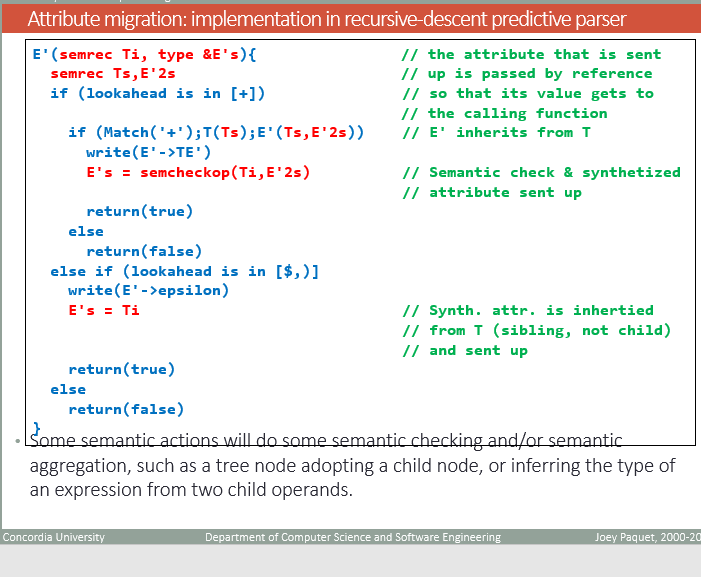
代码分析：

function的参数要有父节点的参数传递，这样得到数据（基础），还可以改变父节点数据（上传）

首先在绝大多数function中，你要创造子节点的semrec

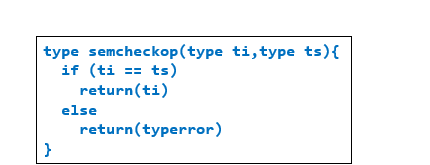
function的参数也不一定只有父节点，例如E'这个function，我们可以通过两个参数传递，在E'内让T1给E'1传递（inherit attribute要让接收数据方的function有两个参数）

Es=E's 向父节点从传递信息，也就是绝大多数synthesis attribute传递方法

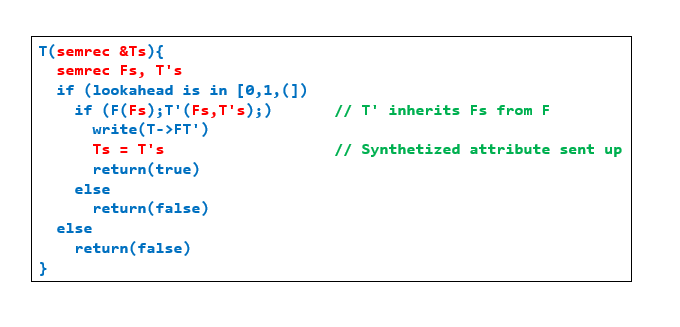


需要从sibling那里得到attribute，所以也需要sibling的semerec //总而言之你这个function要从自己子节点以外得到信息就必须要在function参数额外添加 ，不仅限于semrec， type也是信息

semcheckop（）

如果两个type相等，就return 那个type，不然报错 //type也是 attribute之一

有些semantic action会进行semantic checking或者semantic aggregation.



你可以看见光看图有很多个T，会产生混乱，我们的function永远是基于规则的！

这里在描述T->FT'这个规则， 这里的T是基于第二个function的，左边树，子节点无疑有F1,T1因此创造对应semrec ，

