



# 抽象语法树的自动生成

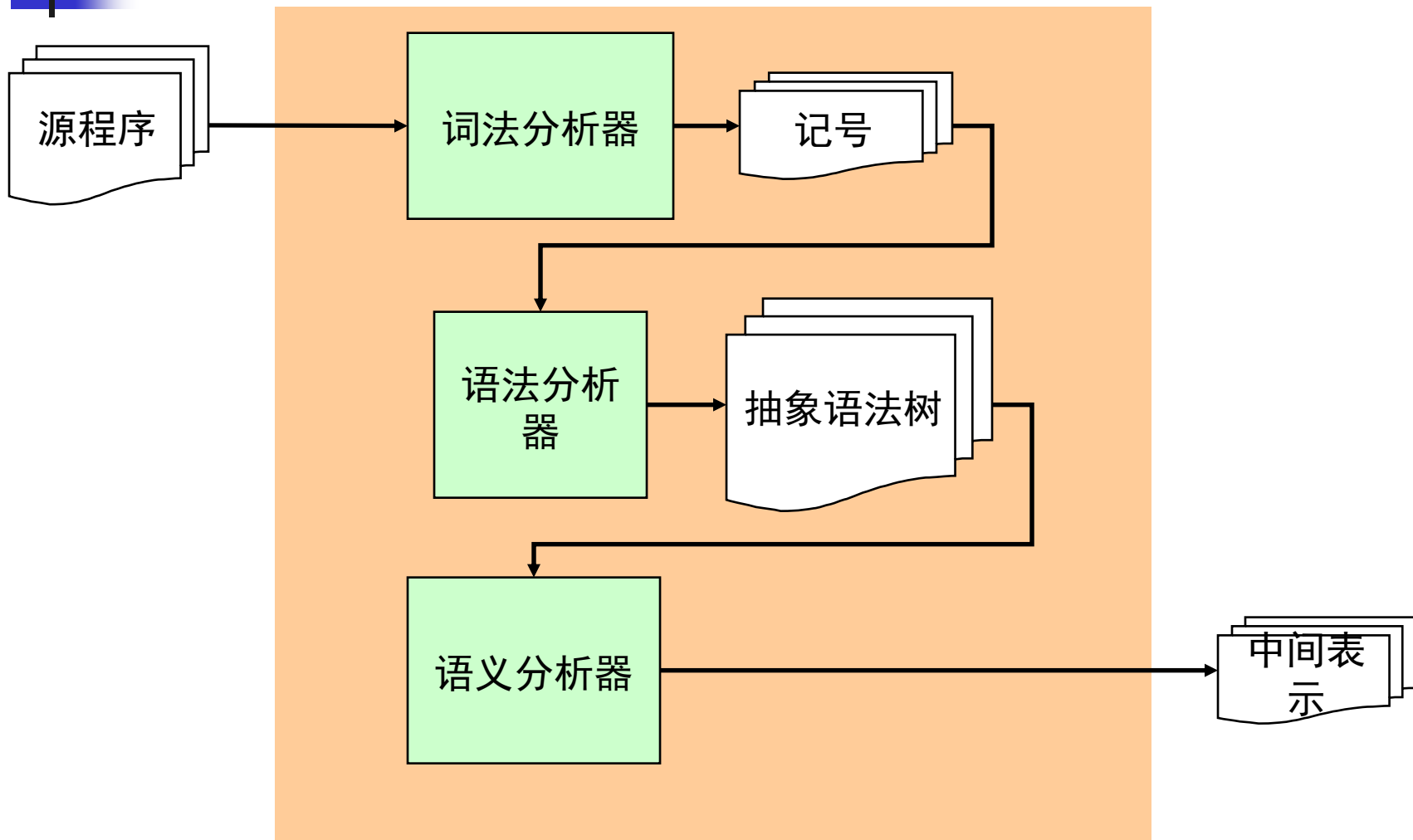
---

编译原理

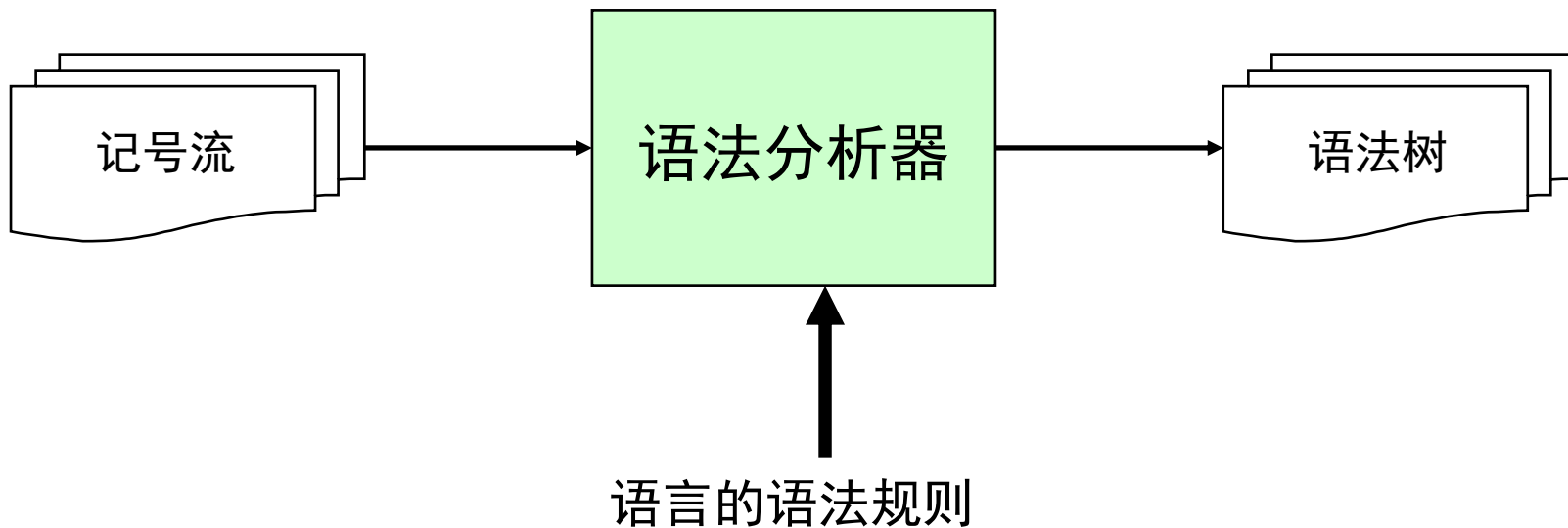
华保健

[bjhua@ustc.edu.cn](mailto:bjhua@ustc.edu.cn)

# 前端



# 语法分析器的任务





# LR分析中生成抽象语法树

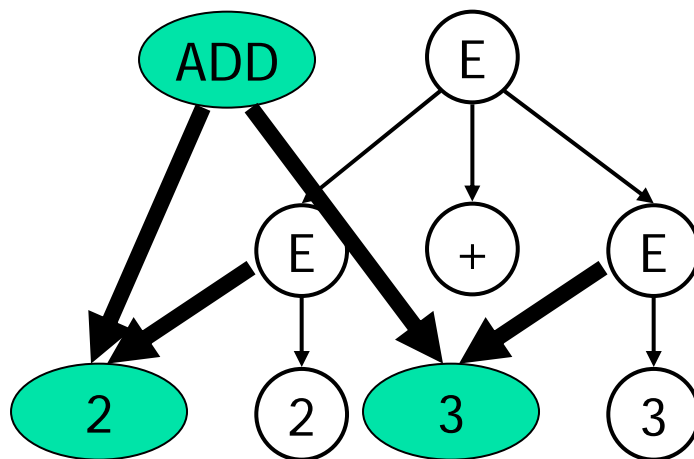
---

- 在语法动作中，加入生成语法树的代码片段
  - 片段一般是语法树的“构造函数”
- 在产生式归约的时候，会自底向上构造整棵树
  - 从叶子到根

# 示例：LR分析中生成抽象语法树

$E \rightarrow E + E \quad \{\$ \$ = \text{Exp\_Add\_new} (\$1, \$3); \}$   
 $\quad \quad \quad | E * E \quad \{\$ \$ = \text{Exp\_Times\_new} (\$1, \$3); \}$   
 $\quad \quad \quad | n \quad \quad \quad \{\$ \$ = \text{Exp\_Int\_new} (\$1); \}$

	2 + 3 + 4
2	● + 3 + 4
E	● + 3 + 4
E +	● 3 + 4
E + 3	● + 3 + 4
E + E	● 3 + 4
E	● + 4
E +	● 4
E + 4	●
E + E	●





# 源代码信息的保留和传播

---

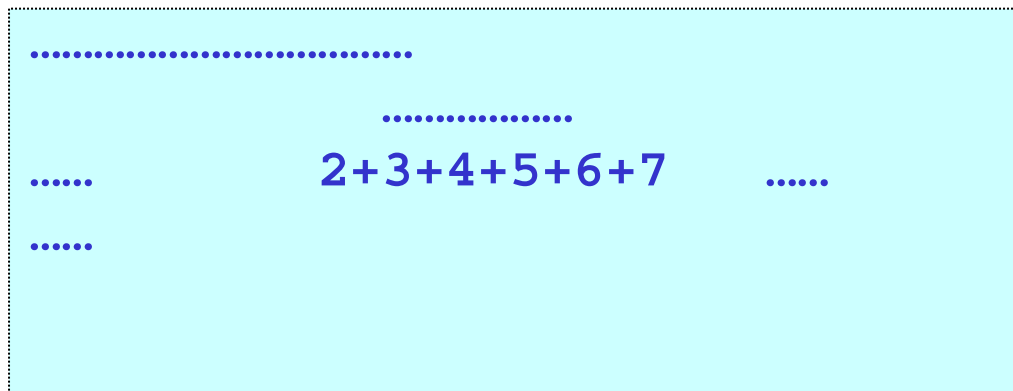
- 抽象语法树是编译器前端和后端的接口
  - 程序一旦被转换成抽象语法树，则源代码即被丢弃
  - 后续的阶段只处理抽象语法树
- 所以抽象语法树必须编码足够多的源代码信息
  - 例如，它必须编码每个语法结构在源代码中的位置（文件、行号、列号等）
    - 这样，后续的检查阶段才能精确的报错
    - 或者获取程序的执行剖面
- 抽象语法树必须仔细设计！



# 示例：位置信息

---

```
struct position_t{
    char *file;
    int line;
    int column;
};
struct Exp_Add{
    enum kind kind;
    Exp *left;
    Exp *right;
    struct position_t from;
    struct position_t to;
};
```



.....  
.....  
..... 2+3+4+5+6+7 .....  
.....