LR(1) 语法分析

– Based on Racket (A Dialect of Lisp)

61519322 杨哲睿

2021 年 11 月 19 日

目录

1 实验要求

1. 手工画出 LR(1) 项目集族及状态图
2. 构造 LR(1) 分析表
3. 在上述过程中，对二义文法进行必要的处理

   表达优先级和结合律

   按照优先级和结合律对预测符及状态转换关系进行必要的取舍
4. 按照 LR(1) 分析表写出语法分析程序

当然，这只是一个最简单的要求。但考虑到 LR(1) 分析器的手工构造难度，该实验使用**程序推导**的方式来构造 LR(1) 项目集族和状态转换图。并且在实验的具体实现上有所变化。

2 设计思路

根据 htdp 中的详细介绍，我们可以将设计一个程序分为以下六个步骤：

1. From Problem to Data Definitions
2. Signature, Purpose Statement, Header
3. Functional Examples
4. Function Template
5. Function Definitions
6. Testing

报告也将从这六个方面展开，对于该语法分析程序的制作进行分析。

2.1 From Problem to Data Definitions

> *Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.*

总的来说，我们 LR 分析程序：

- 输入为：Token 的流 (具体实现为生成器 *generator*)
- 输出为：对应的一颗语法树

一个 LR(1) 语法分析程序所涉及的数据结构如下：

1. `syntax-item` 文法符号 – 指上下文无关文法中的非终结符或终结符单元，拥有名称 `id`，优先级 `proority`、匹配器 (函数)`matcher`
2. `matched-item` 匹配的文法符号 – 指输出时匹配后的终结符单元
3. `production` 产生式 – 上下文无关文法的一个产生式，分为左部和右部两部分

4. `syntax-tree-node` 语法树的一个节点
5. `LRItem` LR(1) 项 – 分为四个部分：产生式的左部、产生式右部在 · 之前的部分、产生式右部在 · 之后的部分、展望 *(Look-Ahead)* 符号

对该数据结构的定义如下：

Listing 1: types.rkt

```
(struct syntax-item
 (id priority matcher) #:transparent)
(struct matched-item
 (stx content))
(struct production
 (left right) #:transparent)
(struct syntax-tree-node
 (head children) #:transparent)
```

当然这些只是基础的定义，为了完成 LR(1) 程序，我们还需要定义：

1. 一个 LR(1) 项目集就是一个 `LRItem` 的列表
2. 一个上下文无关文法包含：终结符号、非终结符号、开始符号、产生式
3. 对于一个增广的上下文无关文法，还需要一个增广文法的新开始符号及其对应的开始符号产生式

2.2 Signature, Purpose Statement, Header and Functional Examples

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

根据这些类型的定义，我们足以定义出整个 LR(1) 语法分析程序的所有函数，如下：

closure

输入 LR(1) 项目集 I

输出 项目集的闭包 CLOSURE(I)

注 1 使用柯里化，这里不需要输入产生式表。

get-closure-function

输入 产生式的列表

输出 CLOSURE 函数

注 2 同理下面几个函数也使用柯里化，降低复杂度。

get-look-ahead

输入 产生式列表

输出 $look - ahead$ 函数

look-ahead

输入 LRItem

输出 当前 LRItem 在获取到下一个非终结符后的部分的 First 集

注 3 若有 LRItem $= [A \to \alpha \cdot B\beta, a]$ 则 $look-ahead(\text{LRItem}) = FIRST(\beta)$。这里不需要原来的 *look-ahead* 是因为在后续计算中加入了 *look-ahead* 的考量。

go

输入 LR(1) 项目集闭包、syntax-item

输出 闭包在获取到该 syntax-item 后产生的新项目集 (非闭包)

look-ahead-and-reduce

输入 LR(1) 项目集闭包、(读头下的非终结符对应的)syntax-item

输出 闭包中可以执行规约的项目

注 4 这是我实现的方法和书上方法主要的不同：在龙书中，*LR* 自动机的一个项目实则对应了一个项目集 (闭包)，所以我们无需对于状态进行编号，而只需要通过闭包的转换即可实现等价的自动机行为。

look-ahead-and-reduce

输入 LR(1) 项目集闭包、(读头下的非终结符对应的)syntax-item

输出 闭包中可以执行规约的项目