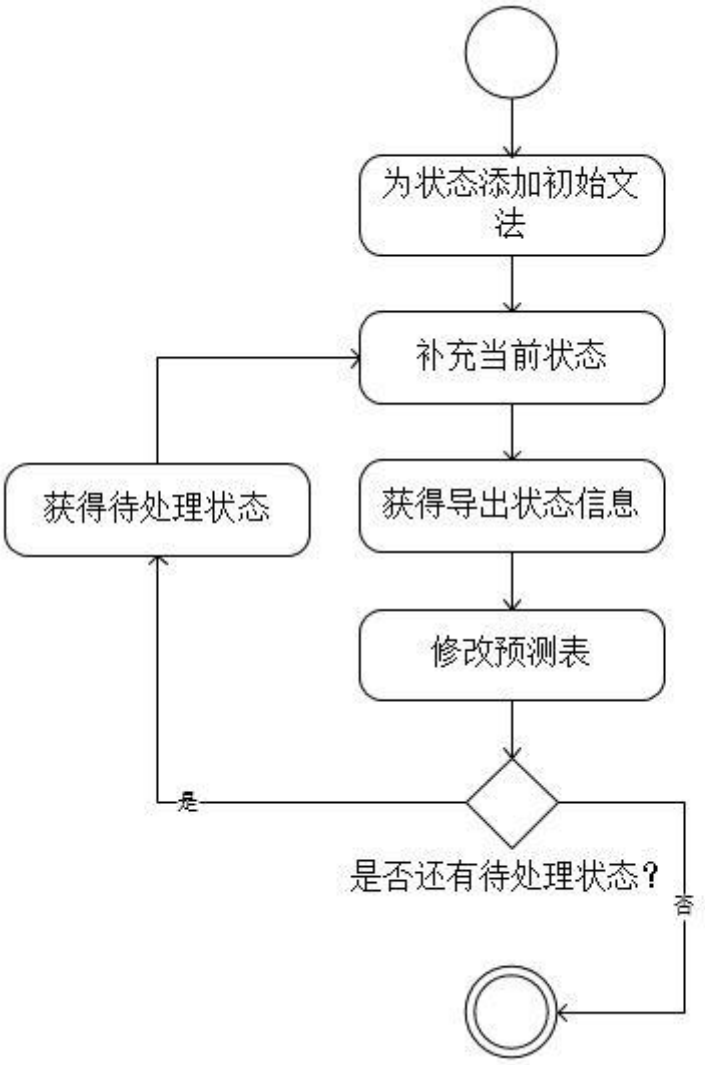
语法分析——LR1

为了完成 LR1 文法分析，首先建立 LR1 文法的状态机，然后利用该状态机建立 LR1 预测表，这样 LR1 语法分析的核心工作就完成了。通过这张表，我们使用栈就能判断字符串是否符合对应的文法了。

在实现的时候，我使用了 Rule 和 RuleSet 两种数据结构。Rule 表示的是文法，例如 S->E 这样一条语句就是用 Rule 来存储，Rule 类内部将上述文法存储为 Left 和 Right 两个部分，同时利用 C#的语言特性，提供了一个 Show 属性，自动将文法左部和右部拼接成一个完整的字符串供使用。RuleSet 存储的是若干个 Rule 的集合，用来表示状态机中的每一个状态。

程序的大致流程如下图所示：



流程图中核心的部分就是补充当前状态、获得导出状态信息和修改预测表这三步，下面分别阐述。

首先是补充当前状态。首先扫描当前状态中已经有的文法，对于每一个文法，根据“.”

（程序中使用 ptr 来表示）的位置，其后是否有非终结符，如果有的话则将左侧是该非终结符的其它文法（状态中不存在的）加入该状态，然后为该文法添加预测符。预测符一般取触发该文法引入的非终结符在文法中位置的后面一个符号的 First 集合。添加完成后我们还要再一次的扫描当前状态中已经有的文法，同样执行上面的操作。需要注意的是，如果在加入状态时产生重复，只需要把新的预测符加入到已有的文法中即可。重复上述操作，知道当前状态不再产生变化为止。程序中的 FillSet 方法实现了这个过程。

当一个状态完整后，我们就能够进行下一步了——获得导出状态信息。导出状态的信息使用 C#中的字典来表示（C++、JAVA 中一般叫 Map），Key 为状态转换条件，Value 为转换后的状态。首先扫描当前状态中的每一条文法，根据“.”（程序中使用 ptr 来表示）后的符号从字典中获取 RuleSet 对象（如果当前键值对还没有建立，需要先创建 RuleSet 对象），然后向 RuleSet 中添加当前文法，并且将“.”（程序中使用 ptr 来表示）向后移动一位。

需要注意一点，由于 Rule 和 RuleStep 的 ID 是自动增长的（当对象是通过 new 创建的，如果调用 Copy 和 Clone 方法是取被复制对象的 ID），所以当你创建一个新的状态时，需要和已有的状态的内容进行一一比对，如果这个状态已经存在，你需要把修改这个状态的 ID 使它和已经存在的那个相等。**（最后试验的时候你会发现产生的状态 ID 不是连续，就是这个原因）**。

每产生一个新的导出状态信息，就能够根据这个信息更新 LR1 预测表。这一部相对来说比较简单，取出导出信息中的每一个键值对，检查 Key 值，如果是非终结符则进行 Goto 操作，如果不是则进行 Shift 操作。程序中我使用 s 代表 shift，g 代表 Goto。在操作类型后面加上 Value 值的 ID 后就能把这个字符串更新到预测表中了。

然后还需要扫描当前状态（不是导出的状态）每一个文法，是否存在“.”（程序中使用 ptr 来表示）已经到文法尾部，如果有则需要进行 Reduce 操作，程序中使用 r 代表 Reduce，在其后加上需要进行 Reduce 操作的文法的 ID。

需要额外注意，如果是包含开始符的文法（如 S->E），在满足 Reduce 条件时表示文法已被接受，而不需要再进行 Reduce 操作了，我实现的时候往表中放了一个“Accept”字符串。

当完成上面的三个方法之后只需要按照流程图的形式组织代码即可。我在实现的时候使用了一个队列来存储待处理的状态集，每次取出队列头的状态集进行上述操作，导出新状态后都把当前不存在的状态加入到队列的尾部，重复上述操作直至队列为空，这样就能得到一张 LR1 预测表。

接下来使用这张预测表来进行分析，分析的过程需要使用到栈。首先向待分析的字符串末尾添加上一个结束符“#”，然后向栈的底部压入初始状态 0。然后循环下列操作，直到无法从预测表中找到对应的动作（不匹配）或者动作为 Accept（匹配）：

1. 取栈顶元素和字符串指针对应的元素（指针从最左侧开始），查表，根据表项进行操作：
   - S?：字符指针右移 1 位，压入状态 ID。
   - G?：字符指针右移 1 位，压入状态 ID。
   - R?：弹出栈顶的 X 个元素，X 为 ID 为?的文法右侧元素的个数，并且向当前字符指针所指的为止前加入 ID 为?的文法的左侧的元素并修改字符指针至这个位置。

测试数据及结果（运行环境 .NET 4.5 VS2013）：