动态分析之力：自适应$A(daptive)LL(*)$解析算法

摘要

尽管如PEG，$LL(*)$，GLR和GLL等现代解析策略取得了不小的进步，但解析本身并不是一个完全被解决了的问题。现有的方式存在许多缺陷，如：难以支持增强式嵌入动作(action)，性能低下或者说不可预测的解析速度，以及有悖直觉的解析策略。本文介绍的$ALL(*)$解析策略结合了传统的自上而下$LL(k)$解析器的简单、高效、可预测能力，以及类GLR机制的强大解析决策功能。其关键的创新之处在于将语法分析推迟到了解析运行时，这使得$ALL(*)$能够处理任何非左递归的上下文无关文法。理论上$ALL(*)$的解析时间复杂度是$O(n^4)$，但在实际的文法分析中却始终呈线性复杂度，性能比GLL和GLR等一般的解析策略高出好几个数量级。ANTLR4生成的$ALL(*)$解析器，可以通过重写文法来支持直接左递归。ANTLR4的广泛使用（2013 年的下载量为 5000 次/月）表明，$ALL(*)$对各种应用程序都卓有成效。

1. 导论

尽管现代语法解析策略已经非常先进成熟，并且在学术研究上历史悠久，但在实践中，计算机语言的解析仍然未能被彻底解决。迫使程序员修改文法以匹配确定性（注：确定性一词，指的是确定性有限自动机DFA，与之对应的时非确定性有限自动机NFA）$LALR(k)$或$LL(k)$解析器生成器的规范来提升解析性能，在硬件资源稀缺时是无可厚非的。但随着硬件资源成本的不断降低，研究人员开发出了功能更加强大、但解析成本更高的非确定性解析策略，这些策略遵循"自下而上"（LR风格）或"自上而下"（LL风格）的解析方式。这些策略包括GLR，解析器表达式语法(PEG Parser Expression Grammar)，来自ANTLR 3的$LL(*)$以及最近提出的一种全通用自上而下的GLL解析策略。

虽然这些策略在使用上比$LALR(k)$和$LL(k)$解析器生成器更加便捷，但它们也存在各种短板。

首先，非确定性解析器有时会出现未预料的行为。由于GLL和GLR是专为处理自然语言语法设计的，而通常情况下，这种语法会潜在地包含二义性，故GLL和GLR会为存在二义性的语法返回多种解析树（森林）。对于计算机语言来说，二义性几乎就是一种异常。当然为处理这种不常见的情况，我们可以对构建出的解析森林进行遍历来消除此等二义性，但这往往意味着需要消耗额外的时间、空间和硬件资源。根据语法定义，PEG是一种无二义性的文法，但存在一种特殊情况，即规则$A \rightarrow a \mid ab$（表示非终结符A要么匹配a要么匹配ab）永远无法匹配ab，因为只要输入的前缀匹配a，PEG文法将会选择第一种产生式。存在嵌套的回溯时，调试PEG将变得举步维艰。

其次， 像打印语句这种由程序员提供的能够对解析过程产生影响的动作(actions)或者是修改器(mutators)都应当在持续预测（PEG）或支持多种解释（GLL和GLR）的解析策略中避免，因为这些动作或修改器可能并不应当执行（虽然DParser支持'final'动作，即如果程序员确定某次归约是一个二义性语法的最终产物之一，那么这个动作或者修改器将得到执行）。在没有副作用的情况下，可能触发的动作必须在某个不可变的数据结构被缓存下来，或者是解析本身提供撤销操作。前一种机制受到内存大小的限制，而后一种机制实现则相对复杂并且有可能无法实现。而避免上述副作用的一种典型解决方式是构建一个解析树，用于解析后（post-parse）处理，但这种刻意产物从根本上限制了对输入大小的限制，因为要将整个输入的解析树放入内存中（译者注：要将整个输入整成一个解析树放内存里，可不得要求输入尽量小点嘛）。构建解析树的解析器是无法为大文件输入或者是无边界流数据提供解析能力，除非可以讲这些数据按照逻辑分块(logical chunks)进行处理。