OIL代码自动生成技术过程中的部分研究

徐健峰，张正兰，张 明
（上海海事大学 信息工程学院，上海 200135）

摘 要：针对OSEK标准的应用设计了一个从OIL代码到C代码的自动生成系统，该系统允许用户输入OIL配置文件信息，读取用户的输入转换为标准的C程序，返回给用户，在具体应用的时候，文法限制严格。该系统结合代码自动生成的过程，提出了一些具体的解决过程，消弱了对文法输入的限制，提高了对文法的适应能力。

关键词：代码自动生成；OIL；LL(K)
中图分类号：TP311 **文献标识码**：B

Some research on the process of the technology of OIL code automatic generation

XU Jian Feng, ZHANG Zheng Lan, ZHANG Ming
(School of Information Engineering , Shanghai Maritime University, Shanghai 200135, China)

Abstract: This paper aiming at a hot research question— the application of OSEK standard,designs a system which translate a OIL code to a standard C code. This system allows user input some OIL configuration information,then translates the input information to a standard C programme which would return to the user.There is a strict grammar restriction in the real application.This system based on the code automatic generation puts forward some essential solution to minish the grammar restriction and boosts the grammar adapting ability.

Key words: code automatic generation; OIL; LL(K)

代码自动生成是当今自动化程序设计的一个热点，代码自动生成技术就是帮助程序员完成系统底层的、重复性代码的自动生成，减少软件开发中枯燥且重复的编码工作，使得程序员将更多的时间花在系统架构研究、软件工程学习等方面，从而提高软件系统健壮性、可扩展性以及可维护性和生产率，缩短项目开发时间，节约项目的开发成本，降低项目开发风险，提高软件公司的信誉度，赢得市场主导地位，使公司获得最大回报率。OIL配置文件是对OSEK标准的描述文件，OSEK/VDX是应用在模块和静态实时操作系统上的标准，由主要的汽车制造商和供应商、研究机构以及软件开发商发起。在具体的开发过程中，往往要根据OIL文件的描述来进行具体的编码，将代码自动生成技术应用于OIL文件上，可以减少程序员的大量手工开发，节省了大量的人力物力，具有相当广泛的工业应用前景。本文设计的系统接受用户输入的OIL配置文件，然后经过系统的分析生成相应的C代码，实现了从配置文件到具体程序的自动化，节省了大量的人力物力，并且在嵌入式开发的时候可以继承到嵌入式开发环境中，提供了很大的便捷性。

1 OIL代码自动生成系统的设计与实现

1.1 OIL代码自动生成系统的功能描述

本文中代码自动生成系统的设计模块如图1所示。

OIL代码自动生成系统的输入模块主要是提供两种方式让用户输入OIL配置文件：一是用户输入完OIL配置文件后提供保存功能，此时将用户输入的配置文件保存到用户制定的文件夹内；二是提供选择功能，让用

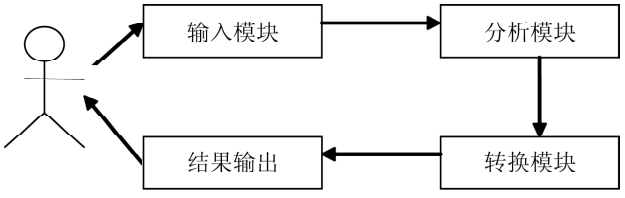

图1 OIL代码自动生成功能模块

户选择已经保存好的配置文件或者是使用其他工具生成的配置文件，将文件读取进系统。

当用户确定输入的配置文件并点击生成按钮后，此时由分析模块对用户输入的配置文件进行分析，系统根据系统规定好的产生式规则进行判定，首先对配置文件进行分词，系统根据输入好的正则表达式提供有穷自动机的功能，对用户的配置文件进行词法分析，将用户输入的字符串分割成符合 OIL 规范的字符集作为下一步语法分析的输入，此时得到的文件应该是具有标记的字符串集合。随后的语法分析模块对词法分析得到的结果进行分析，根据预先设定的正则表达式来判定句子是否符合语法规则，采用 LL(K)进行产生式匹配，并且在匹配后建立相对应的语法树，为后面的 C 代码生成打基础。此后再进行语义分析，通过对语法树进行分析，得到带有注释的语法树，方便后面的转换模块进行遍历。

转换模块的工作主要是收集要生成的 C 程序中必要的数据，例如 CPU 的信息、消息间的相互联系、以及中断和警告的信息等，通过对这些必要信息的记录来实现从配置文件到 C 程序的数据的映射，通过对前面 OIL 语法树的遍历得到这些数据。

结果输出模块是主要是进行模板构造，对转换模块中得到的需要的数据和设定的模板相结合，然后输出，得到最后要生成的 C 程序。

1.2 OIL 代码自动生成系统的核心工作流程

OIL 代码自动生成系统的工作流程如图 2 所示，图 2 描述出了整个系统的核心工作流程：从用户输入代码到输入模块，一直到输出 C 代码返回给用户。

(1) 词法扫描。词法扫描程序对源程序进行扫描，从中收集到有意义的字符序列，收集到记号中。

(2) 语法分析。程序依据文法规则，从扫描程序中获取记号形式的源代码，完成程序结构的语法分析，从而确定整个输入串是否构成一个语法上正确的程序，并输出语法树。

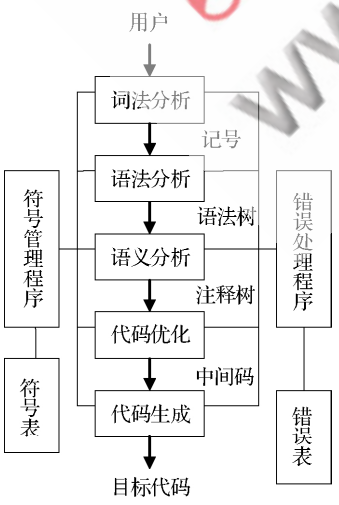

图 2 系统工作流程

(3) 语义分析。审查源程序有无语义错误，并为代码生成收集必要的信息。

(4) 代码优化程序。对于语义分析形成的注释树进行遍历，取得需要的数据。

(5) 代码生成部分。根据前面取得的信息将信息以符合 C 程序的形式组织起来形成 C 代码。

2 OIL 代码自动生成系统中关键技术的研究

本系统采用的是自上而下的 LL(K)分析方法，所以本系统可以接受的文法必须是一个正确的、上下文无关文法，该文法不仅能够正确完整地反映出 OIL 的语法，并且应该符合自顶向下分析的要求，这个就要求该系统能够处理以下几种情况：

(1) 如何处理出现二义性；

(2) 克服左递归弊端；

(3) 如何确定 LL(K)中 K 的值以保证正确识别文法和效率之间的统一。

文法的二义性是指对于同一句子有两种不同的语法树，则称该句子是二义性的，称产生该句子的文法为二义性文法。解决二义性的方法有两种：一种是设置一种规则，该规则指出在二义性的情况下哪种语法树是正确的，例如在 ELSE 问题上面，规定每个 ELSE 和最近的没有分配的 IF 匹配，这种方法的优点是无需修改文法就可以克服文法的二义性，缺点是此时语言的语法结构就不能由文法单独决定了；另外一种方法就是对存在二义性的文法进行改写，如果一个二义产生式右部有非终结符出现一次以上，可以利用产生式引入消除，如产生式 $A\rightarrow \alpha B\beta B\gamma$，可以变换为 $A\rightarrow \alpha B\beta A'$，$A'\rightarrow B\gamma$.如果多候选产生式的右部有一个是二义性的，那么每个右部都要作为这个代换部分移除,例如 $A\rightarrow \alpha A\beta A\gamma|\alpha_1|\alpha_2|\cdots|\alpha_n$,转换为 $A\rightarrow \alpha A'\beta A\gamma|A', A'\rightarrow A'|\alpha_1|\alpha_2|\cdots|\alpha_n$,消去其中的无用产生式后得到，$A\rightarrow \alpha A'\beta A\gamma|A', A'\rightarrow \alpha_1|\alpha_2|\cdots|\alpha_n$。如果一个产生式有多个二义性产生式，可以用上述方法重复变换。

左递归是指当一个上下文无关文法 $G=(V_N,V_T,P,S)$,其中 V_N、V_T、P、S 分别表示非终结符集、终结符集、产生式和开始字符，当文法如下：(1)$A\rightarrow A\alpha|\beta$，其中 $A\in V_N$, $\alpha,\beta\in V^*$，此时认为这是直接左递归,(2)$A\rightarrow B\alpha$，$B\rightarrow A\beta|\gamma$，其中 $A\in V_N$, $\alpha,\beta,\gamma\in V^*$，此时称为间接左递归，当出现左递归的时候，由于本文采用的是 LL(K)文法是采用从左到右的扫描方法，当扫描到(1)中的 A 或者(2)中的 B 时，此时无法确定 LL(K)扫描中的 FIRST 集，会导致扫描失败。对于两步以上的左递归(2)可以转换为直接左递归形式 $A\rightarrow A\beta\alpha|\gamma\alpha$，然后利用下面的算法消

除。此算法可以消除所有无循环推导和空产生式的文法中的左递归：

(1) 以某种顺序排列非终结符 $A_1, A_2, ...A_N$。

(2) For i = 1 to n do begin

For j = 1 to i–1 do begin

用产生式 $A_i \to \delta_1\gamma | \delta_2\gamma | ... | \delta_k\gamma$ 代替每个形如 $A_i \to A_j\gamma$ 的产生式，其中 $A_j \to \delta_1 | \delta_2 | ... | \delta_k$ 是当前 A_j 的所有产生式

End

消除 A_j 产生式中的直接左递归

End

在使用 LL(K)算法的时候，如何确定步长是一个很关键的问题，如果步长过大，那么每次扫描的时候向前看的单词数过多，会引起编译效率的下降；如果步长过小，当两个非终结符具有相同的 FIRST(K)值会导致识别的失败。一般来说，选取 K 值为 1 的时候能满足通常的识别要求，但是在某些特定的情况下可能导致识别失败，不能保证系统的健壮性，例如在以下的情况下使用 LL(1)就不能满足要求：

(1) 当出现 $A \to \alpha\beta_1 | \alpha\beta_2 | ... | \alpha\beta_n$ 的时候，此时如果要对产生式进行展开的话，采用 LL(1)无法确定展开后应该采用那个产生式。

(2) 当出现左递归的时候或者步长为 K 的时候才能区别的产生式。

(3) 当根据以下规则进行词法分析：

Float：（DIGIT）+'.' +(DIGIT)*+； 浮点型

ARRAY：（DIGIT）+'..' +(DIGIT)+;数组

当在这种情况下，由于两个产生式都无法确定前面的 DIGIT 的个数，只有当扫描到“.”或者“..”的时候才能确定该使用哪个产生式，因此此时无法使用 LL(1)进行确定。

当出现(1)的情况时，此时采取提取公因子的方式对产生式进行改写，例如(1)中的产生式可以改写为如下格式 $A \to \alpha A', A' \to \beta_1 | \beta_2 | ... | \beta_n$ 的形式进行转化，此时采用 LL(1)可以成功进行扫描，如果公因子比较长的话可以采取上述办法进行多重转化。

对于左递归的情况上述已经提到过解决方法了，对于步长为 K 才能区别的情况下，此时可以将步长调整到 K 进行扫描，但是使用固定 K 值采用 LL(K)的方法进行扫描的时候，会要求对终结符的 FIRST 集进行计算，这样对许多无需使用 LL(K)的情况造成了资源的浪费，使得扫描的效率降低。

当出现情况(3)的时候无论将 K 值定为多长都有可能出现 K 值不够大而形成扫描失败的情况，此时应该采取步长不确定的方式来进行扫描：当刚刚开始扫描的时候确定 K 的初始值为 1，当扫描失败的时候，如果确定失败的原因是由以上第三种情况导致的话，此时对 K 的值加 1 进行扫描，如果失败再次加 1 直到扫描成功。根据对 OIL 的语法进行观察，当 K 值定为 3 的时候就能解决 99% 以上的扫描失败问题。对于少数为 4 的情况下可以采取提取公因子的情况进行转化。

采用递归下降分析程序扫描失败后会返回失败的节点，这种返回原节点的方式称为回溯，在编译过程中这样的现象被认为是一种极大降低效率的现象，因此要尽力避免回溯的出现。为了避免回溯的出现，在每次选择产生式的时候采取预测分析的方法，即禁止回溯，当需要确定使用产生式的时候采取预测的方法，使用预测的产生式，如果失败则报错，这样就避免了回溯的出现。预测是使用预测函数来实现的，预测函数就是确定下一个待输入的字符是否在当前产生式 A 的预测函数中 predict(A)中，如果在的话，就选择产生式 A，预测函数就是产生式 A 的向前看 K 个单词，下列产生式 $A \to X_1X_2X_3...X_N$ 如果向前看的单词 K 个数为 1 的话，则此产生式的预测函数的定义为如下：

$$predict(A \to X_1X_2...X_N) = \varepsilon \in FIRST(X_1X_2...X_N)\ ?\ (FIRST(X_1X_2...X_N) - \{\varepsilon\} \cup FOLLOW(A)) : FIRST(X_1X_2...X_N)$$

如果两个右部产生式的预测函数有非空交集的时候，还需要往前看 K 个字符，以上的方法一般表现为一个分析表，表的行表示非终结符，表的列表示终结符，表和列的交叉点就是当非终结符遇到该终结符该使用哪个产生式去进行扩展。

本文探讨了代码自动生成技术的一些步骤，并提供了 OIL 代码自动生成技术的系统模型。文中提出了一些在 OIL 代码自动生成词法分析和语法分析过程中遇到的实际问题加以讨论并提出了实际的解决办法，为下一步语义注入提供了基础。本系统的实际开发遵循了 MVC 开发方式，保证了先进性，并且为代码自动生成技术提供了一些可以参考的思路。

参考文献

[1] LOUNDER K C. 编译原理及实践[M]. 冯博琴, 冯岚, 译. 北京: 机械工业出版社, 2000.

[2] 陈火旺, 刘春林. 程序设计语言编译原理(第 3 版)[M]. 北京：国防工业出版社, 2001.

[3] 吕映芝, 张素琴. 编译原理[M]. 北京：清华大学出版社, 1998.

[4] APPEL A W, PALSBERG J. Modern compiler implementation in java[M]. 高等教育出版社, 2003.

[5] APPEL A W. 现代编译原理C 语言描述[M] . 赵克佳, 黄春, 沈志宇, 译. 北京:人民邮电出版社出版,2006.

（收稿日期：2009-01-05）