基于 PRAM 模型的 CFGs 并行识别与语法分析的扩充算法*)

孙玉强[1,2] 周 蕾[3] 刘三阳[1] 王洪元[2] 张英丽[2]
(西安电子科技大学理学院 西安 710071)[1] (江苏工业学院计算机系 常州 213016)[2]
(郑州经济管理干部学院计算机系 郑州 450052)[3]

摘 要 本文介绍了一种 PRAM 模型上的上下文无关文法的并行识别和改进的并行语法分析方法——金字塔结构，并对该方法进行了修改和补充，使其对非 Chomsky 规范形式，即文法的产生式右部候选式(即规则)有两个以上的非终结符连接的，或者候选式中既有非终结符，又有终结符的情况，扩充的算法也能识别和分析。

关键词 上下文无关文法，语法分析，算法，扩充

Expanded Algorithms of Parallel Context-free Recognition and Parsing on a PRAM Model

SUN Yu-Qiang[1,2] ZHOU Lei[3] LIU San-Yang[1] WANG Hong-Yuan[2] ZHANG Ying-Li[2]
(School of Science, Xidian University, Xi'an 710071)[1]
(Department of Computer, Jiangsu Polytechnic University, Changzhou 213016)[2]
(Department of Computer, Zhengzhou Economic Management Institute, Zhengzhou 450052)[3]

Abstract In this paper a method of parallel context-free recognition and parsing on a PRAM model is presented, which structure is pyramid. And it is revised and mended, in order that this method can also be applied no matter the given grammar G is Chomsky Normal Form or not, that is, this method can also be applied to a grammar G, whose right side of each production (or rule) is consisted of more than two non-terminals or terminals mingled with non-terminals.

Keywords Context-free grammars, Parsing, Algorithm, Expand

1 引言

上下文无关文法的并行识别与语法分析是计算机软件理论研究的热点，有不少文章都对上下文无关文法的并行识别与语法分析进行了不同方位和侧重点的研究，包括按传统的语法分析规则，对一般上下文无关文法的并行识别与语法分析的并行算法[1]；对括号语言的并行识别与语法分析[2]；线性环形拓扑结构中的并行语法分析算法[3]；基于逻辑方法、面向对象方法及联结模型[4~6]等等。文[7]中，介绍了 PRAM 模型上的一种并行语法分析方法，对给出的文法有较高的要求，只可以对 Chomsky 规范形式文法进行语法分析，即对传统的产生式右部候选式为一个终结符或者为两个连接的非终结符有效。本文研究与分析了对任意上下文无关文法(产生式右部的情况较为复杂时)应用该算法的情况和限制，经过对文法的等价变换和算法的改进，使算法的应用范围更广。

2 算法思想

本文提到的文法，如果未加说明都是指上下文无关文法。

定义 1 如果存在推导过程 $S\Rightarrow^{*}\alpha$，则称 α 为 G 的一个句型。如果 $\alpha\in V_T^{*}$，则称 α 为文法 G 的句子。文法 G 的所有句子构成的集合称为文法 G 的语言 $L(G)$，即 $L(G)=\{\alpha|\alpha\in V_T^{*},S\Rightarrow^{*}\alpha\}$，即语言 $L(G)$ 是 V_T 上的一个串集。空字用 ϕ 表示，空字满足：$\phi u=u\phi=u$，并且 $|\phi|=0$。

上下文无关文法的识别问题，就是给定上下文无关文法 G，和一个串 $\omega(\omega\in V_T^{*})$，确定是否存在着推导：$S\Rightarrow^{*}\omega$ 的问题。而语法分析问题，就是给定上文无关文法 G，和一个串 ω ($\omega\in V_T^{*}$)，对由文法的开始符号 S 到串 ω 的推导过程进行分析和研究。

定义 2 对于给定文法 G，和一个输入串 $\omega=a_1\,a_2\cdots a_n$ ($a_i\in V_T$)，我们约定用 $T[i,j]$ 表示如下定义的集合：

$$T[i,j]=\{A\mid A\Rightarrow^{*}a_j\,a_{j+1}\cdots a_{i+j-1}\}$$

显然，$T[i,j]$ 是 V_N 的一个子集，并且对于给定文法 G，和一个输入串 $\omega=a_1\,a_2\cdots a_n(a_i\in V_T)$，$|\omega|=n$，由定义 2，可得如下性质：

性质 1 如果在文法 G 中，存在产生式 $A\rightarrow a_i$ ($1\leqslant i\leqslant n$)，那么 $A\in T[1,i]$；

性质 2 如果存在产生式 $C\rightarrow AB$，并且 $A\in T[m,j]$，$B\in T[i-m,j+m]$ (其中 $1\leqslant i\leqslant n$，$1\leqslant j\leqslant n-i+1$，$1\leqslant m\leqslant i-1$)，那么 $C\in T[i,j]$；

性质 3 对于文法 G 的开始符号 S，如果 $S\in T[n,1]$，那么文法 G 可以识别输入串 ω。

证明 1：当 $1\leqslant i\leqslant n$ 时，由 $A\rightarrow a_i$，得 $A\Rightarrow^{*}a_i$，根据定义 2 易得 $A\in T[1,i]$；

证明 2：由 $A\in T[m,j]$ 和定义 2，可得：

$$A\Rightarrow^{*}a_j\,a_{j+1}\cdots a_{m+j-1} \tag{1}$$

又由 $B\in T[i-m,j+m]$，同理可得

$$B\Rightarrow^{*}a_{j+m}\,a_{j+m+1}\cdots a_{i+j-1} \tag{2}$$

由产生式 $C\rightarrow AB$ 和(1)，(2)可得：

$C\Rightarrow AB\Rightarrow^{*}a_j\,a_{j+1}\cdots a_{m+j-1}\,B\Rightarrow^{*}a_j\,a_{j+1}\cdots a_{m+j-1}\,a_{j+m}\,a_{j+m+1}\cdots a_{i+j-1}$，即 $C\Rightarrow^{*}a_j\,a_{j+1}\cdots a_{i+j-1}$，所以有：$C\in T[i,$

*)基金项目：河南省基础研究(004061800)、自然科学基金(021102600)和(0324220079)资助项目。孙玉强 教授，博士生，主要研究方向为软件工程、并行计算与优化。周 蕾 副教授。刘三阳 教授，博导。

$j]$；

证明 3：由 $S\in T[n,1]$ 和定义 2，可得：$S\Rightarrow^{*} a_1\ a_2\cdots a_n$，所以文法 G 可以识别输入串 ω。

由以上性质，可得：Chomsky 规范形式文法 G 识别输入串 ω 的一种方法，即，计算出集合 $T[n,1]$，只要集合 $T[n,1]$ 中包括有文法 G 的开始符号 S，那么文法 G 就可以识别输入串 ω。

3 并行算法的设计与改进

3.1 Chomsky 规范形式文法并行识别算法

Chomsky 规范形式文法识别输入串的方法分为三个步骤，依次为：

(1)根据性质 1，对文法 G 的产生式右部为终结符的产生式，即对形如 $A\rightarrow a_i$，$a_i\in V_T$ 的产生式，计算出 $T[1,i](1\leqslant i\leqslant n)$；

(2)根据性质 2，计算出 $T[i,j]$(其中 $1\leqslant i\leqslant n, 1\leqslant j\leqslant n-i+1$)；

(3)最后，根据性质 3，由 $S\in T[n,1]$ 是否成立，即 $T[n,1]$ 中是否含有文法 G 的开始符号 S，来确定文法 G 是否可以识别输入串 ω。

该算法其时间复杂度为：$O(n)$，最坏情况下，处理器复杂度为 $O(n^2)$。设在构造的最底层，即 $i=1$ 时，构造时间为 t_1，设在第 i 步迭代($2\leqslant i\leqslant n$)时，每次的迭代时间为 t_2，则整个算法的总时间为：

$$t_1+t_2=\sum_{c=2}^{n}t_1+t_2(n-1)$$

所以，时间复杂度为：$O(n)$。又在第一步迭代中，需要 n 个处理器，在第 i 步迭代($2\leqslant i\leqslant n$)，构造集合 $T[i,j]$ 时，需要 $i-1$ 个处理器，存放按照性质 2 得到的 $T[i,j]$ 中元素，又 $1\leqslant j\leqslant n-i+1$，需要构造 $n-i-1$ 个集合 $T[i,j]$，故在第 i 步迭代($2\leqslant i\leqslant n$)时，共需要：$(n-i-1)(i-1)$ 个处理器[7]。

算法对 Chomsky 规范形式，即文法的产生式右部的候选式，仅有两个非终结符连接，或者为一个终结符的情况有效。但是对产生式右部的候选式含有其它各种情况时，要想应用这种并行语法分析方法，就需要改进。

3.2 算法的改进与扩充

对于较一般的非 Chomsky 规范形式文法，即候选式(规则)右部有两个以上的非终结符连接的，或者候选式中既有非终结符，又有终结符的情况，要进行语法分析和识别，算法 1 只能进行到第一步，所以要对文法和算法 1 进行相应的修改。修改的目的是使得这种并行语法分析和识别方法，对一般的文法(包括 Chomsky 规范形式和非 Chomsky 规范形式)都能适用，使其更具有一般性。

对一般文法进行并行语法分析和识别，可以先把非 Chomsky 规范形式的文法转换成 Chomsky 规范形式文法，然后继续进行并行语法分析和识别，例如：对于非 Chomsky 规范形式的文法，有规则：$S\rightarrow ABC$ 和 $S\rightarrow A+C$，我们可以把它们转换成 Chomsky 规范形式，成为候选式右边是两个非终结符连接或者一个终结符的情况，转换方法按照候选式右边符号的类型，有两种转换模式，其一是：

对于规则右式中既有非终结符又有终结符的情况，即，规则为：

$$P\rightarrow A_1\ A_2\cdots A_k^{(1)}\ a_{11}\ a_{12}\cdots\ a_{1m}{}^{(1)}\ A_{k+1}^{(1)}\cdots A_k^{(2)}\ a_{21}\ a_{22}\cdots\ a_{2m}{}^{(2)}\ A_{k+1}^{(2)}\cdots A_k^{(i)}\ a_{i1}\ a_{i2}\cdots\ a_{im}{}^{(i)}\ A_{k+1}^{(i)}\cdots A_k^{(s)}\ a_{s1}\ a_{s2}\cdots\ a_{sm}{}^{(s)}\ A_n$$

其中：$a_i\in V_N$　$A_i\in V_N$。那么，我们可以把该规则改写成如下的形式：

$$P\rightarrow A_1\ A_2\cdots A_k^{(1)}\ B_1\ A_{k+1}^{(1)}\cdots A_k^{(2)}\ B_2\ A_{k+1}^{(2)}\cdots A_k^{(i)}\ B_i\ A_{k+1}^{(i)}\cdots A_k^{(s)}\ B_s\ A_n$$
$$B_1\rightarrow a_{11}\ a_{12}\cdots\ a_{1m}{}^{(1)}$$
$$B_2\rightarrow a_{21}\ a_{22}\cdots\ a_{2m}{}^{(2)}$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$B_i\rightarrow a_{i1}\ a_{i2}\cdots\ a_{im}{}^{(i)}$$
$$\cdots\cdots\cdots\cdots\cdots$$
$$B_s\rightarrow a_{s1}\ a_{s2}\cdots\ a_{sm}{}^{(s)}$$

这样，就转换成了第一种情况，然后，再按照第一种情况的转换方法进行转换，最终得到的文法是 Chomsky 规范形式的，并且形式和原来的形式也是等价的。

由两种转换模式，可以对非 Chomsky 规范形式的文法进行转换，通过改写文法，把该文法转换成 Chomsky 规范形式的，按照改写后的文法，就能继续进行上述算法的第二步和第三步，顺利完成语法分析和识别，所以对上述算法进行修改，可得：

扩充算法：SIMD-CREW 模型上的并行识别算法

输入：给定一个文法 G，和一个输入字符串 $\omega=a_1\ a_2\cdots\ a_n(a_i\in V_T)$

输出：文法 G 是否可以识别输入字符串 ω

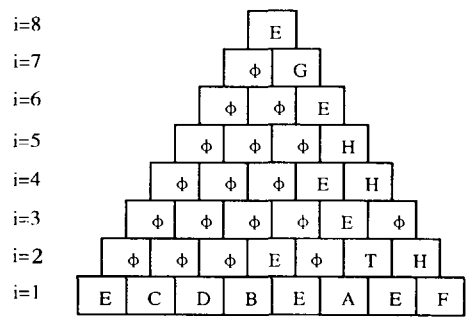
```
begin
  (1)if 文法不是 Chomsky 规范形式
      then 对文法进行改写 else goto(2)
  (2)for all i (1≤i≤n)
        if A→a_i, a_i∈V_T*
          then A∈T[1,i]
        end if
      end for
  (3) for i=1 to n par-do
        for j=1 to n-i+1 par-do
        for m=1 to i-1 par-do
          T[i,j]={C | C→AB ∈ P, 且 A∈T[m,j],
                  B∈T[i-m,j+m]}
        end for
        end for
      end for
  (4)if S∈T[n,1] then return yes
                 else return no
      end if
end
```

下面，我们举例来说明，利用该并行算法来进行识别和语法分析的过程：

例：对于给定的非 Chomsky 规范形式的文法

$G=\{V_N, V_T, \Sigma, P, S\}$，$P=\{E|E\rightarrow E+E\mid E*E\mid(-E)\mid -E\mid i\}$，$V_N=\{E\}$，$V_T=\{+,*,-,(,),i\}$，和输入字符串 $\omega=i*(-i+i)=a_1\ a_2\cdots a_7$，$|\omega|=7$。

计算的结果，可以表示为金字塔的形状：



由图可得：$S\in T[8,1]$，即 $T[8,1]$ 中含有文法 G 的开始符号 E，故能够确定：文法 G 可以识别输入的字符串 $\omega=i*(-i+i)$。

(下转第 208 页)

$\xrightarrow{t2} W_{OK}$

$W_{LO} = water\text{-}lo \xrightarrow{t2} W_{LO} \square water\text{-}ok \xrightarrow{t2} W_{OK}$

$W_{HI} = water\text{-}hi \xrightarrow{t2} W_{HI} \square water\text{-}ok \xrightarrow{t2} W_{OK}$

其中：$t_2 = 20ms$，

$water\text{-}lo \overset{def}{=} WaterSensor.GetLevel() = LOW$，$water\text{-}hi \overset{def}{=} WaterSensor.GetLevel() = HIGH$，

$water\text{-}ok \overset{def}{=} WaterSensor.GetLeval = OK$

Ⅳ）监控模块进程

$M = water\text{-}hi \xrightarrow{\tau 1} turn\text{-}on\text{-}pump \xrightarrow{\tau 2} M \square water\text{-}lo \xrightarrow{\tau 1} turn\text{-}off\text{-}pump \xrightarrow{\tau 2} M \square water\text{-}ok \xrightarrow{\tau 1} no\text{-}change \xrightarrow{\tau 2} M$

其中：$\tau 1 + \tau 2 = 20ms$，*water-hi*、*water-lo*、*water-ok*、*turn-on-pump*、*turn-off-pump*、*no-change* 定义同上。

对模块进程进行同步并行组装，即可完成系统的行为描述：

$PUMP = PC |[I_P]| P$，$I_P = \{pump\text{-}on,\ pump\text{-}off\}$

$WATER = S |[I_w]| M$，$I_w = \{water\text{-}lo,\ water\text{-}ok,\ water\text{-}hi\}$

$SYSTEM = PUMP |[I_{pw}]| WATER$，$I_{pw} = \alpha P \cap \alpha W = \{turn\text{-}on\text{-}pump,\ turn\text{-}off\text{-}pump,\ no\text{-}change\}$

通过分析行为描述，可以生成在特定硬件环境中执行的调度构件，例如每个模块进程可以处理为一个操作系统线程，模块进程间的同步并行组装通过线程间同步实现。又如外部选择组装可以处理为条件选择语句，上文中的监控模块进程可以翻译为：

```
while(true){
        switch(WaterSensor.GetLevel()){
        case OK: PumpController.KeepStat(); break;
        case HIGH: PumpController.TurnOn(); break;
        case LOW: PumpController.TurnOff(); break;
        }
        ...
}
```

实时构件描述的主要目的之一是使调度构件的生成自动化，下一节讨论构件描述在实时 CORBA 框架内的应用。

4 在实时 CORBA 中的应用

在 RT-CORBA 体系结构中，客户端程序在使用一个对象之前，必须先获得对象的引用，通常是客户程序先直接获取一个工厂对象引用，以后的对象引用通过工厂对象获得。工厂对象是指那些能够返回其他对象引用的对象，它可能返回一个新的对象，也可能返回一个已有的对象。

为实现调度构件的自动生成，可在服务端 ORB 实现一个特殊的工厂对象 SchedulerFactory，专门负责解析构件描述并生成调度构件实例。当客户需要一个新的调度构件时，先获取 SchedulerFactory 引用，然后向 SchedulerFactory 发送构件描述，请求创建调度对象，得到对象引用后，即可实现对实时任务的监控。

通过增加 SchedulerFactory 工厂对象，可实现调度构件的自动生成，很大程度上提高了实时系统的构件复用程度。传输调度构件描述会加重网络负担，但仅在创建时需要，如采用适当的压缩算法，不会带来过大开销。

结束语 本文提出的实时构件描述机制具有精确的语义，可以准确表达构件的功能和 QoS 特性，这对于实时构件复用是必须的。调度构件描述的自包含特性使调度构件的自动生成成为可能。我们将进一步开展相应支撑技术的研究，如构件描述生成工具、行为描述的分析和转换算法等，并在实时 CORBA 的框架内开发一个实用模型。

参 考 文 献

1 Nielsen B, Agha G. Towards Reusable Real-Time Objects. Annals of Software Engineering, 1999.7: 257～282

2 Lau K-K, Ornaghi M. A Formal Approach to Software Component Specification. In: Proc. of Specification and Verification of Component-based Systems Workshop at OOPSLA 2001

3 张涌，王渊峰，钱乐秋．一个集成式的软件构件描述框架．计算机学报，2002，25(5)

4 Zaremski A M, Wing J M. Specification Matching of Software Components. ACM Trans. Softw. Eng. Method. Oct. 1997, 6 (4): 333～369

5 李阳，吴朝晖．一种形式化构件集成语义的研究．浙江大学学报（工学版），2004，38(2)

6 Fröhlund S, Koistinen J. Quality of Service Aware Distributed Object Systems. 5th USENIX Conference on Object-Oriented Technologies and Systems(COOTS '99)

7 Zaremski A M, Wing J M. Signature Matching: a Tool for Using Software Libraries. ACM TOSEM, Apr. 1995

8 Hoare C A R. Communicating Sequential Process. Prentice-Hall international, 1985

9 Davies J. Specification and Proof in Real-Time CSP. Cambridge University Press, 1993

10 Joseph M. Real-time Systems Specification, Verification and Analysis. Prentice Hall International, 1996

（上接第 204 页）

结论 PRAM 模型上的上下文无关文法的并行识别和改进的并行语法分析方法——金字塔结构，能较好地应用于符合 CNF 的规范产生式集合环境。本文对该方法进行了修改和补充，对产生式右部的候选式（即规则）有两个以上的非终结符连接的、或者候选式中既有非终结符又有终结符的情况作了等价变换，并对转换后的文法的等价性、适应性、效率以及识别分析过程做了分析和讨论。在不增加系统太多开销的情况下，算法能较有效地进行识别和分析。

参 考 文 献

1 Gibbons A, Rytter W. Efficient parallel algorithms. Cambridge University Press, 1990

2 Rytter W, Giancarlo R. Optimal parallel parsing of bracket language. Theoretical Computer Science, 1987

3 Ra Dong-Yul, Kim Jong-Hyun. A parallel parsing algorithm for arbitrary contexe-free grammars. Information Processing Letters, 1996, 58: 87～96

4 Matsumoto Y. A parallel parsing system for natural language analysis, New Generation Comput, 1987, 15: 63～78

5 Yonezawa A, Ohsawa I. Object-oriented parallel parsing for context-free grammars. In: proc. of 12th Int. Conf on Computational Linguistics(COLING-88), 1988. 773～778

6 Wyard P J, Ninghtingale C. A single layer higher order neural net and its application to context-free grammar recognition, In: N. Sharkey. ed. Connectionists Natural Language Processing. Chapter, 1992, 8: 139～162

7 Chandwani M, Chaudhari N S. Formulation and analysis of parallel context-free recognition and parsing on a PRAM model. Parallel computing, 1996, 22: 845～868