上下文相关图文法分析及其应用初探*)

冉 平 石 兵 马晓星 吕 建
(南京大学计算机软件新技术国家重点实验室 南京大学计算机软件研究所 南京 210093)

摘 要 图文法是一种对可视化语言进行形式化定义的元语言,具有表达自然、能力强大的特点。随着使用可视化语言的最终用户编程技术的广泛应用,图文法分析尤其是上下文相关图文法分析在工程应用中的重要性日益突出。国内外相关文献或着重于纯理论探讨,或局限于特定语法类的特定应用,不利于工程应用人员参考。本文选取简洁明了的符号体系,介绍上下文相关图文法分析的一般性过程,并将其中规则选取关键步骤描述为 CSP 问题,利用已有的针对 CSP 问题的优化方法来优化算法,介绍了现有的优化方法并给出实现算法;同时,结合自身实践,讨论其在一个面向体系结构的 Web 服务集成系统中的应用。

关键词 图文法,上下文相关,分析算法,软件体系结构

Parsing Algorithm of Context-Sensitive Graph Grammar: Overview and Experience

RAN Ping SHI Bing MA Xiao-Xing LU Jian
(National Key Laboratory for Novel Software Technology, Institute of Computer Software, Nanjing University, Nanjing 210093)

Abstract Graph grammars are natural and efficient in formally specifying visual programming languages, which are often the keys to the end user programming. Parsing based on graph grammars, especially more expressive context-sensitive grammars, is increasingly becoming important and prominent in the industrial applications. Existing references either focus only on theoretical aspects or limit to particular uses of some self-defined grammars. In this paper, we overview the general parsing process with concise notations, and abstract the step of selecting rules as a CSP problem in order to use existing optimization methods for CSP to optimize the parsing algorithm. In addition, we present our experience on the development of an architecture-oriented Web services integration system in which a context-sensitive attributed graph grammar is used to describe and check software architectures.

Keywords Graph grammar, Content-sensitive, Parsing algorithm, Software architecture

1 引言

正如大量存在的文本语言一样,也存在大量的可视化的语言,并且越来越多地出现在学术研究以及应用在工业产品中。未来的软件朝着最终用户编程、面向应用以及依靠服务集成开发的方向发展,而为了给这种发展方向提供便利,需要提供一种直观、有效的描述软件结构等机制。幸运的是,可视化的语言就恰恰具有这样的特性,能够提供这样的功能。例如,CASE 提供了大量的图形,用来支持软件需求的说明以及软件设计,未来还应提供软件开发支持。

图文法作为一种对可视化语言进行形式化定义的方法,具有自然以及强大的特点。和文本语言类似,一个图文法也是由一组规则组成,这组规则定义了如何从一个初始图形(开始状态)通过应用这组规则生成一个图语言。每条规则定义了如何转换,由左部和右部组成,都分别是一个图形。而图形是在有向图的基础上,由节点以及描述节点之间的空间关系的边这两种基本元素组成。

有时要求图形满足某些条件,需要检查一个图形是否是该文法所产生的图语言,这也称为一个识别或者分析的过程。通常的基于规则的图文法方法中,一个最常见的操作是将一条图文法规则应用到另一图时,首先找到规则左部在图中的一个匹配(match)。在图理论中,这个问题通常被称为是子图同构问题,并且是 NP 问题[1]。最坏情况的复杂度是图规则的左部或右部元素的指数关系,而且这仅仅是图分析过程中的一个步骤。当前分析算法大都只能处理上下文无关(context-free)的文法和规则,即规则的左部只是一个单独的非终结符号。这对于理论研究是足够的,但是对于具体的工程应用则显得描述能力不足。需要一个允许规则的左部和右部都是任意的图形,并且允许存在一些图形元素用来作为图形转换必须满足的前提条件的文法,这种文法称为上下文相关(context-sensitive)图文法。

本文的工作主要集中在基于一个简单的符号体系给出分析算法要正常终止时图文法需要满足的前提条件,并对讨论的文法做出一些限制;描述图文法分析的一般性过程、深度优先和广度优先算法,以及该分析过程的子图匹配和规则选取两个阶段,并分析导致算法高代价的问题,指出基于这些问题可以进行的优化工作;在此基础上介绍文[2]中利用 CSP 来优化图匹配的过程的优化工作,并延续该思路,将规则选取步骤也抽象描述为 CSP 问题,利用已有的针对 CSP 问题的优化方法来优化分析算法;使用智能回退来部分解决 Thrashing

*)本文工作受到国家自然科学基金(60403014,60273034)、863 计划(200AA116010)和 973 计划(2002CB312002)的资助。冉 平 硕士研究生,主要研究兴趣为 Internet 软件技术;石 兵 硕士研究生,主要研究兴趣为 Internet 软件技术;马晓星 副教授,主要研究兴趣为 Internet 软件技术、软件体系结构;吕 建 博士、博士生导师,主要研究领域为软件自动化、并行程序形式化方法、面向对象语言和环境。

问题，减少搜索空间和回退的次数；介绍使用规则分层的优化方法并给出实现算法、冲突对的分析优化方法等，同时结合自身实践讨论其在一个面向体系结构的 Web 服务集成系统中的应用；使用上述图文法分析技术来描述和检查软件体系结构并探讨在包括程序理解，模式识别等其他领域中的应用。

本文第 2 节给出图文法的基本定义；第 3 节给出分析算法的一般性步骤；第 4 节将分析过程抽象为 CSP 问题，讨论存在的常用的优化策略，并改进部分算法；第 5 节给出图文法分析的具体应用；最后是结束语。

2 基本定义

大量存在的可视化的语言并没有一个统一的形式化的描述方法。每一种语言，其作者都定义了一套自己的标注和描述方法，这对于研究和有效地利用可视化语言都是一个困难，所以需要一个图文法的形式化的定义。这是消除歧义，产生正确的语言分析器和语法制导编辑器的重要前提条件。我们使用一种简单的易于理解的符号体系来表示图文法[2,10]。

定义 2.1 简单图 G(graph)是一个元组，$G=(G_V,G_E,L,s,t,l)$，其中有限集 G_V 表示节点的集合，有限集 G_E 表示边的集合，且 $G_V\cap G_E=\varnothing$。两个全映射 $s,t:G_E\to G_V$(表示的是边的起始和终结)；L 表示的是边和节点标号集合(类别)；一个全映射 $l:G_V\cup G_E\to L$。

定义 2.2 规则 P(production、rule)是定义在同一个类别集合 L 之上的一个元组 $p:=(L,R)$，其中 L 和 R 分别表示规则的左部和右部，L 和 R 可能有一个共同的子图 $K(L\cap R)$，它满足以下的条件：

$\forall e\in E(L)\cap E(R)\Rightarrow s(e)\in V(L)\cap V(R)\wedge t(e)\in V(L)\cap V(R)$

$Vx\in L\cap R\Rightarrow l(L)(x)=l(R)(x)$

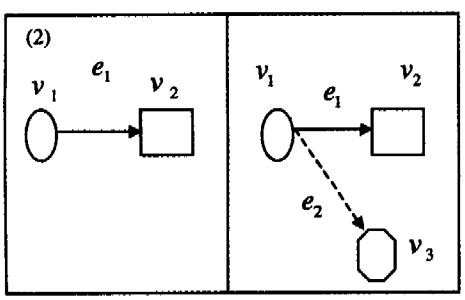

图 1 规则示例

第一个条件表示公共边的起始节点和终结节点都必须是 L 和 R 的公共节点；第二个条件表示公共边和节点的标号相同，也就是类型相同。K 称为上下文元素，在规则的应用中，作为规则需要满足的前提条件，在应用规则时的转换过程中保持不变。同理，我们可以定义 L/R 和 R/L，分别表示规则应用中被删除的节点和创建的节点。在上例中，K 为 v_1,v_2，e_1，L/R 为空，R/L 为 v_3,e_2。

定义 2.3 图文法 GG(graph grammar)是一个元组，由一组规则集合 RS 和一个图形 G 组成(G,RS)，图形定义了开始状态。为了将图文法规则 P 应用到图形 G，需要对规则左部图形 L 到图形 G 的匹配(match)做一个精确的定义。这个匹配是定义在相同标号集合(类型)上的一个映射。

定义 2.4 图映射(graph morphism)是定义在两个简单图 $L=(L_V,L_E,L_L,s_L,t_L,l_L)$ 和 $G=(G_V,G_E,L_G,s_G,t_G,l_G)$ 间的一对全映射 $m=(m_V:L_V\to G_V,m_E:L_E\to G_E)$，满足以下条件：

- $\forall v\in L_V:l_L(v)=l_G(m_V(v))$
- $\forall e\in L_E:l_L(e)=l_G(m_E(e))$
- $\forall e\in L_E$
- $m_V(s_L(e))=s_G(m_E(e))$
- $m_V(t_L(e))=t_G(m_E(e))$

条件 1 表示点的类型相同，条件 2 表示边的类型相同，条件 3 表示边的起点和终点相同。

定义 2.5 规则实例(production instance)是一个元组 $pi:=(p,h,h')$，定义一个图形 G 应用一个规则后到另外一个图形 G'，其中 $h:L\to G,h':R\to G'$，而且满足下列条件：

$G/(h(L/R))=G'/(h'(R/L))$

在实际应用中，规则的左部和右部可以是任何的图形，具有强大的描述能力。但是，并不是所有的图文法都是可分析的，即图文法中可能存在着循环，使得分析过程不能够正常地结束。为了使其在有限步骤内结束，借用 Chomsky 对文本文法的分类：

1. $\forall p(|L_p|\leqslant|R_p|\wedge L\neq R)$

对于满足该条件的文法，能够保证分析程序的正常结束，因为规则的右边的元素不比左边少，而且至少创建或修改其中的一个元素(节点或边)，这在 chomsky 文法体系中称为 1 型文法，也就是上下文相关的文法。

在实际的应用中，为了分析算法的高效，要求

2. 图形是连通图。

对于连通图的要求是为了方便将规则的左部或右部线性化。在寻找匹配时，按照边的关系一步一步匹配，这也会在下面的分析中得到。

3 分析算法

为了检查一个图形是否是某个图文法的语言，需要产生一个图文法规则的推导步骤，就需要对图形进行分析。

从原子图形 A 到最终的分析图 G 的推导过程是一组顺序的规则实例：

$$A\xrightarrow{pi_1}G_1\xrightarrow{pi_2}G_2\xrightarrow{Pi_3}\cdots\xrightarrow{pi_n}G_n\equiv G$$

可以看到，分析算法的过程一般分成两个阶段：一是找到规则在图形中的匹配，也即找到一个规则实例；二是对图形应用规则进行转换，并决定如何选取规则。最直观的想法就是使用穷尽搜索法，步骤如下：

算法 1：

```
PROCESS parsingGG,Stop Graph;
BEGIN
a:G1<-Start Graph of GG, G2 <-Stop graph, RS<-rules
  set of GG;ST<-Stack;
LOOP BEGIN
B: if((isomorphism(G1,G2))
        Terminate Correctly;
  initialize a Set S for G1;
c:if
  (∃rule(rule∈RS∧match(rule,G1)∧(match(rule,G1)∉
  S))
  )
      add(G1, match)into ST;
```

```
        add match into S;
        G1<-apply rule to G1
        else
            if(ST=∅)
                Terminate Wrongly
            else
                (G1,S)<-POP(ST);
        END
END
```

这个算法是最简单的没有优化的穷尽搜索算法，而且是深度优先，每次都随机选择一条可以应用的规则(称这个点为决策点)，保存当前状态(图形)并应用规则，重复此过程。如果演化的图形和终结图形匹配，则分析成功，该图形是图文法的一个语言。否则回退到上一个决策点。如果所有的路径都已尝试，则分析失败。

可以看出程序的代价主要体现在：

1. 检查图的同构，找到规则到图形的匹配的代价；
2. 图形的应用规则转换的代价；
3. 大量的回退以及回退所导致的额外的操作。

分析这些情况，可以知道检查图的同构以及找到规则到图形的匹配是两个类似的操作，都是检查两个图之间是否存在一个映射，只不过同构是需要两者完全相同，而匹配只要部分相同。这个过程就是前面提到的子图匹配问题，而且在最坏情况下具有指数级的复杂度。图形的演化的代价是相对固定的，和图形在具体的应用中使用何种数据结构描述的复杂程度以及方便性相关。而第三个部分的大量的回退是造成算法低效的重要原因，因为大量的回退不仅会有额外的栈操作，而且造成额外的图演化以及查找可应用的规则的过程。

上述是一个从开始状态到终结图的推导过程，每次都是寻找规则左部和图形的匹配。还存在一种方法是从右到左的应用规则的规约过程，本质思想和推导类似，也是穷尽回退方法，只需要对上述的算法做出微小的改动，将 a 的开始图和终结图的位置互换，将寻找规则匹配的 c 改成应用规则右部等。但是，通过这个改动，可以发现 b 点检查图同构的过程将变得很简单，因为终结图形现在只是一个节点。所以，通常使用规约过程，在后面的优化措施中使用的也是规约。

为改进算法，虽然不能改变子图同构的最坏情况下的代价，但可以降低图形匹配的平均复杂度，并且可以通过研究如何选取规则来尽可能减少回退次数。

4 算法优化

4.1 子图匹配问题

在文[2]中将子图匹配问题抽象成 CSP 问题，便于直观地理解子图匹配中的代价开销产生的原因，并且充分利用目前 CSP 研究领域中的已有成果来解决部分问题。

CSP 问题由以下三个部分组成：

一组变量集合 $X=\{x_1,\cdots,x_n\}$，每个变量的取值范围组成的一个有限集合 D_i，一组变量的约束条件的集合。一个约束条件是这组变量的 D_i 的笛卡儿乘积上的一个关系。当对一组变量的实例化取值满足所有的约束条件时，称之为一个解决方案。在 CSP 问题研究中，图着色问题是典型的例子。对于 CSP 问题的介绍，请参见文[5]。

建立 CSP 问题和图形匹配的对应关系，从 $L=(L_V,L_E,L_L,s_L,t_L,l_L)$ 到 $G=(G_V,G_E,L_G,s_G,t_G,l_G)$ 的匹配过程中，将 L 的元素当成是 CSP 中需要匹配的变量，将 G 中元素组成的集合当作变量的取值范围，而约束条件则由一些限制条件转换而来。可以得出约束条件主要是类别相等以及边的约束(空间关系)，这也是前面要求图形是连通图的另一个主要原因。这样，可以更多地建立图形中元素的关系，并将有联系的元素显式表示出来，增加了约束条件，降低了匹配过程的难度，提高了匹配速度。

得出 $X=L_V\cup L_E=\{x_1,\cdots,x_n\}$，$D=G_V or G_E$，约束条件是[2]：

- $x_i=x_k$，

$C^{type}_{x_i}=\{d\in D_i\mid l_L(x_i)=l_G(d)\}$

- $x_i\in L_E, x_k\in L_C, s_L(x_i)=x_k$，

$C^{sr}_{(x_i,x_k)}=\{(d_i,d_k)\in D_i\times D_k\mid s_G(d_i)=d_k\}$

- $x_i\in L_E, x_k\in L_V, t_L(x_i)=x_k$

$C^{tar}_{(x_i,x_k)}=\{(d_i,d_k)\in D_i\times D_k\mid t_G(d_i)=d_k\}$

为了对这组变量进行实例化，CSP 常用的就是穷尽的搜索和回退，通常意义上需要考虑的搜索空间是 $|D|^{|X|}$。如果变量 x_i 之前的变量都已经被赋值时，还需要搜索的空间是 $|D|^{|X|-i}$，而且如果此时 x_i 对于 D_i 中的任何值都不满足约束条件，就可以放弃回退到 x_{i-1}，减少了搜索空间。从直观上想象，还可以减少每个变量的取值范围，这同样可以加快搜索。在关于 CSP 的讨论中，关于简单回退的一个缺点就是存在 thrshing 问题[6]，也就是说当不满足某个约束条件时，每次都是回退到上一个决策点。而造成匹配失败的原因，可能显见是更上层的决策点，也就是回退的过程没有从失败中学习。给一个直观的例子：

对 x_1,x_2,x_3 按照简单的回退方法顺序赋值，它们的取值范围 Di 分别在图中标出。如果此时已经为 x_1,x_2 分别赋值了 v_{11},v_{12}，需要考虑 x_3 的赋值，我们发现不满足约束 $C13$，而且不满足是由于 x_1 的取值造成的，应直接回退到 x_1 的决策点，但是仍继续考虑 x_3,x_2 的各个取值。

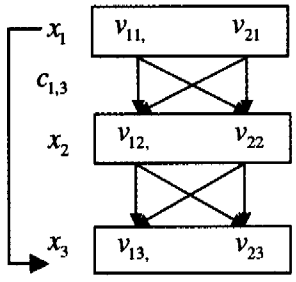

图 2 Thrashing 问题

变量顺序和智能回退：

通过前面的分析，可以看出一个良好的变量排列顺序是非常有必要的。如果将那些经常匹配失效的元素放在最前面，就能迅速排除很多搜索路径，并且通过边的约束条件来减少后续元素需要考虑的空间。这个顺序可以是由用户在编写规则的时候指定并附加在规则上，或者通过将一条规则的左部和其他的规则左部进行一个重叠(overlap)操作来计算得出，可以称为是这条规则的特征元素。

当一个约束条件不满足时，需要从这个失效中学习到错误产生的根源，智能回退到产生问题的决策点，而不是遍历其中所有的变量的所有取值。

4.2 规则的选取

分析算法中的另外一个时间开销相当大的过程是规则的

选取，而且也存在大量的回退过程。这个过程和子图匹配很相似，我们也抽象成 CSP 问题。

从分析过程终结的条件得出，不论是分析成功还是失败，算法都能够在有限的步骤内结束，所以变量是一个有限集合。但是变量集合的具体数量是未知的。取值范围就是规则集合，约束条件是每个步骤都必须有可选的规则，并且最终得到的转换图形等于终止图形。关于子图匹配的优化措施，具体有两种：一是减少搜索的空间 Di，二是减少回退的次数。那么，在这两种思想的基础上，就出现了分层规则。

直观上对分层的理解是规则的应用都是有顺序的，一条规则的应用必然是在另一条规则的前面，底层规则的应用为高层规则的使用创造了条件。那么当应用了一条规则之后，只需要考虑和该规则同层或比该规则层次高的规则，这就减少了 Di 空间。而且，如果规则是分层的话，可以避免高层规则应用到底层规则应用的回退。文[7]中定义了规则分层必须满足的条件，其中将规则按照从右到左使用，并且认为每层都只有一条规则。

在节点和边的标号集合 L（具有 n 个类型）上的规则集合 R，定义如下的分层函数：

• $cl,dl:L->N$ 是一个满射函数，为每一个标号定义了一个唯一的创建和删除层数，其中 $\forall l\in L:0\leqslant cl(l)\leqslant dl(l)\leqslant n$；

• $rl:R->N$ 也是一个满射函数，为每一条规则定义了唯一的标号。

如果对一个图文法存在一个分层函数，满足如下的条件，就称其是一个分层的图文法。

对于 R 的所有规则 r，$rl(r)=k$：

• r 至少删除一个节点或者边；

• r 仅仅删除那些 $dl(l)\leqslant k$ 的节点或边；

• r 仅仅创建那些 $cl(l)>k$ 的节点或边；

• r 使用的上下文元素必须满足：$dl(l)>k$，$cl(l)\leqslant k$。

直观上理解，该规则删除比该规则层次低或同层的节点或边，只创建比该层次高的节点或边，使用的上下文节点或边是被底层规则创建的且被高层规则删除。要满足上述的条件，显然一条规则不能既删除一个节点又创建同一类节点。

我们给出一个算法来计算分层函数并给每个元素赋值。为 L 中的每个元素赋值并对规则的层次赋值，使其满足上述条件。

算法 2：

```
BEGIN
∀l∈L,cl(l)=dl(l)=0 ∀l∈RS,rl(l)=0,
  change=false
  DO
    BEGIN
      FOR every rule in RS
      BEGIN
      ∀element∈rule
      if(element∈R\L)
        if (dl(L(Element))>rl(rule))
          rl(rule)=dl(L(element)),change=true;
      if(element∈L\R)
        if(cl(L(element))<=rl(rule))
          cl(L(element))=dl(element)=rl(rule)+1
          change=true;
      if(element∈L∩R)
        if(dl(L(element))<=rl(rule))
          dl(element)=rl(rule)+1,change=true;
        if(cl(L(element))>rl(rule))
          rl(rule)=cl(L(element))change=true;
      END
    END
  WHILE(change)
END
```

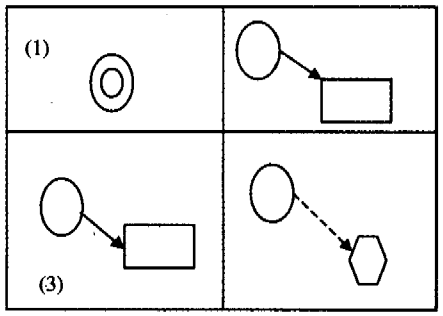

图 3　可分层的规则示例

上面的两条规则和前面的一条规则组成一个图文法，可以看出具有一个层次关系。当从右向左使用时，都删除了节点，并且满足上述的其他条件。所以当推导的时候，规则 1 肯定最先使用，然后是 2，最后是 3，其中绝无交叉。在规约的时候，按照相反的顺序。利用上面的算法得出的结论是完全相同的。

上面给出的例子是特例，每层都只有一条规则，要满足上述的严格的条件的文法较少而且描述能力不强，在实际的使用中，可以进行调整，允许同一层具有多条规则，例如将所有右部都是终结符号的规则规定为一层，在推导过程中，在应用一条终结规则之后，不会再使用前面层次的规则。但是，由于每层可能具有多个规则，而且对该层规则使用的次数是无法确定的，同层规则间还是具有回退。

冲突对的分析：

Critical Pair Analysis 最初出现在重写规则中，后被应用在分析算法中。主要思想是将有冲突的规则尽量推迟使用，也即是如果两条规则之间存在冲突，使用了其中的一条之后，另一条不会出现在推导（规约）路径中；如何定义冲突？可以认为如果一条规则删除了一个节点或边，而这个元素是另一条规则的 match 的上下文元素，文[7,8]中给出了详细的定义。但是使用的是静态分析，也即是对规则进行分析，认为两条冲突规则不出现在一条路径中。对于需要转换的图，仅仅是两条规则的左部或右部的叠加产生的最小图是成立的。而在实际的使用中，两条具有冲突的规则仍然能够同时出现在一条路径中。因为虽然规则 A 删除了规则 B 的 match 的一个上下文元素，但是在图形中还可能存在 B 的另一个 match。

冲突对的思想仍然是有用的，但冲突分析是在规则实例之间进行的。因为规则实例才定义了共同使用的元素，才会出现一个规则实例的使用阻止了另一个的使用，所以文[9,10]中定义了一种广度优先的分析算法并给出证明。

将分析算法分成两个步骤：第一个阶段是 bottom－up（规约），从要分析的图 G 出发，每一步使用规则右部，考虑当前可以匹配的所有规则，产生一个规则实例，并且将规则应用到图形中，但在应用中并不删除任何节点，只把规则的左部加

到 G 上;循环该步骤,直到图文法的开始符号出现在图形中,此时的图形 G1 是具有很多元素的复杂的图形,要分析的图是 G1 的子图。

第二阶段 Top-down 的过程就是从开始符号出发,从在第一阶段产生的每个步骤的所有规则应用中找到一条到分析图的路径,这个过程并不仅仅是简单的搜索和回退过程。在第一阶段产生了很多有用的信息,并在规则实例之间定义了三种关系:above,exclude,consequence,分别是一个规则实例必须在另一个规则实例之前;一条规则实例的应用排除了另一个规则实例的使用可能;一个规则实例是另一个规则实例的结果。

这种方法比较好地利用了冲突对的思想,并且算法的效率也较好。但是可以看出,该算法的效率代价主要是在第一阶段,由于不删除元素,所以到后面节点具有大量的节点或边,匹配是一个很复杂的问题。

冲突分析的方法还是可以和分层的方法结合起来使用,只需要在同层规则之间分析冲突。

thrashing 问题:正如前面所讨论的,thrashing 问题在这个阶段仍然存在,只能是通过智能回退的办法来解决。如果分析过程失效了,发现是一条规则创建了一个元素造成的,并且该元素不会在后续规则的应用中被删除,那么就可以直接回退到该规则处。实际应用较复杂,造成 thrashing 问题的原因很多。但可以使用优化措施,减少回退的数量。

在图形分析失败的情况下,需要对所有的路径遍历完才能判断。这个过程有大量的回退,而且大量的回退是在同一条规则上的,也即是说两次使用同一条规则产生的两个规则实例是等价的,即以前已经有过应用这条规则并且失效,后来回退到该决策点仍然对同一规则的不同实例进行尝试。不再应用这条规则,但这不是绝对的,因为同一条规则的应用并不一定产生相同的效果,也即转化得到的图形可能不同。如果能够区分这两种情况,那么就能减少搜索的路径。

定义(等价规则实例) 如两个规则应用得到的图形相同,那么可以称这两个规则实例等价。这可以是同一条规则产生。也可以由不同的规则产生。但是在一个不存在二义性的文法中(即不存在两条推导路径),只会出现同一条规则的两个实例等价。

4.3 关于算法优化的相关工作

在文[9,11]中对现有的算法进行了比较。在很多的文献中都定义了自己的文法、各自定义规则描述、重写方法等。虽然分析算法的代价是线性的或多项式的,但大都是上下文无关的,且描述能力不强。例如文[11]在前面 LGG 的基础上定义了一种 RGG 的文法,认为规则使用没有顺序,也即规则的应用可以互换,不影响推导的结果。分析算法的代价也是多项式的,但只对部分文法有效。

此外,还存在其他的优化措施,如 Type Graph 方法,即类型图,就是关于图的图。对某些应用,我们可以定义期望从规则集推导产生的图语言,类似于根据某种文本文法所产生的语言,如:language(s)=$\{a^i b^i\}$。可以定义待检查的图中可以出现的元素的类型、每种类型元素出现的次数,以及连接的对应关系(1∶1 或 1∶n)等。当只是需要判断一个图是不是某个图文法的语言时,就可以用这种方法进行快速的检查,不需要进行搜索匹配。但这种方法也有局限的地方:有些复杂结构的图难以用 Type Graph 来表示。

属性的扩展:前面描述的图只是节点和边的组合。为了增加图的描述能力,可以对图形中的元素进行扩展,使其具有属性。例如一个节点表示网络上的一个计算服务,则节点可以具有一个属性,用来表示点的计算能力等。属性同样可以用来定义元素之间的联系和约束。对具有属性的图,其分析过程是基于上述分析过程的,即每一步首先检查图形的逻辑结构是否满足,再检查属性的约束是否满足。

5 具体应用

5.1 软件体系结构的描述和检查

在文[14]中,使用预先定义的软件体系结构来描述和集成 Internet 上的 Web 服务以及其他构件,开发面向 Internet 的应用。这个预先定义的软件体系结构定义了一类应用的拓扑结构以及该结构需要满足的一些约束条件。在开放的环境下,为了使系统灵活地动态演化,以适应底层环境的变化或用户需求的变化,同时需要对描述该应用的体系结构进行调整和演化。通常情况下,体系结构约束的破坏就是系统演化的驱动因素。我们的系统 Artemis-ARC 包含了一个基于 Eclipse 通用平台开发的可视化的服务(构件)集成环境,具有面向服务的应用的集成开发支持、运行支撑支持和运行监控支持等几个方面的功能,所有这些功能都可以在集成环境上可视化地完成。

当用户通过界面使用某种体系结构定义了一个应用后,在实施和部署之前需要分析该应用是否满足某个体系结构类型,包括应用的逻辑结构是否和体系结构相同,并且是否满足体系结构定义的约束条件。当系统演化后,体系结构发生了改变,那么我们的结构是否还满足该类型呢?所以需要对其进行检查。

为了有效和显式地描述软件体系结构及其动态行为,需要形式化的手段[3,4],我们使用图的方式来描述,因为描述软件体系结构的拓扑结构本身就是一个图,而且图能直观地描述软件构件之间的控制流和数据流。图文法作为定义可视化语言的元语言,一方面自然容易理解,另一方面具有强大的功能,并且提供自动化的语法检查和转化。能可视化地描述出软件体系结构,并且利用图文法的元素以及节点的属性来实现体系结构类型的约束条件,其中节点和边定义的关系是显式的,而变量定义的约束条件是隐式的,利用图的转化规则刻画体系结构的动态演化。例如系统中预先定义了一种 Master/Slave 结构,Master 将工作分配给 Slave 来完成。所以控制流方向是 Master 到 Slave,结构是一个 Master 对应多个 Slave,其中定义的约束是 Master 的工作量小于所有 Slave 的计算能力。当应用发生改变之后,例如 Master 接受的工作的负担更重了或某个节点的计算能力下降,并且违背了预先定义的约束条件,那么系统就需要动态地演化,增加一个 Slave 节点。此时的系统是否还是 M/S 类型,是否仍然满足约束条件?

在 ARTEMIS—ARC 系统中,我们利用图文法分析算法的技术来实现一个检查器,用来检查一个系统的结构图是否满足某种体系结构的类型,是否满足其约束条件。以上面的 Master/Slave 为例,利用图文法定义该体系结构的规则,一个较简单的文法就是由图 1 和图 3 中三条规则组成,其中空间关系表示了逻辑结构,内在的节点的属性定义了约束条件。

$G_{\text{Master/slave}}=(G_V,G_E,L,s,t,l)$

$G_V=\{S_{\text{start}},master,Middle,slave\}$

$G_E=\{L\text{-}m,l\}$

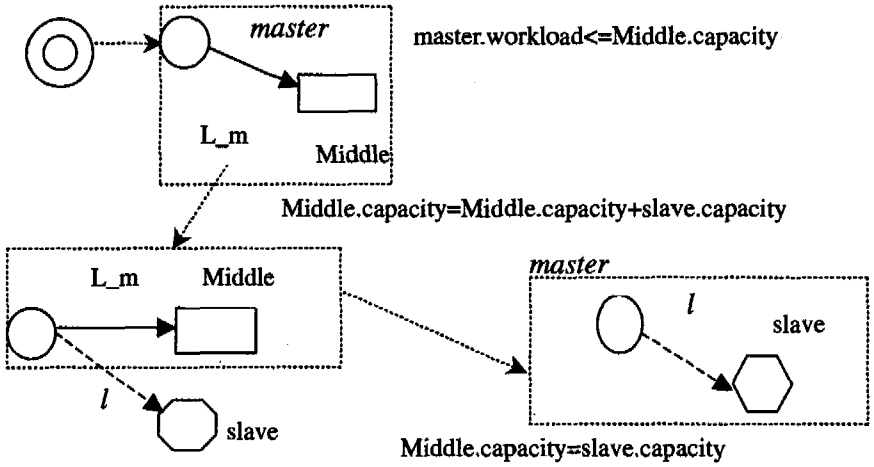

图 4

利用前述的优化措施，我们使用规约的过程，使用分层的规则，利用同构规则实例的判断，减少搜索的路径等方法来开发一个分析算法，称为检查器。在文[19]中类似地也使用了图文法来表示动态的软件体系结构及完成动态配置下的正确性验证。

5.2 分析 xml 语言

在文[15]中使用了图文法来直观地描述 XML 的结构，用分析算法来验证 XML 文件的结构的正确性，利用图的转换代替 XSL，完成将一个 XML 文件从一种结构转换到另一种结构的功能。

5.3 软件工程中的应用

从已存在的软件中识别出公共的、常用的结构称为程序识别，这些常用的结构可能是某个算法（如二叉搜索）或常用的数据结构（如列表）等。程序识别提供了对一个程序的良好的理解，这是逆向工程中软件维护、排错以及重用的重要前提。通常的识别方法都是基于文本的，一是难以理解，二是复杂度比较高，难以实现。为了能够有效地理解程序，希望能自动地识别，在文[16]中使用了图的方法来实现这个过程。把希望识别的通用的结构（结构具有某些约束）预先定义成一组规则，放在一个库里。分析时将面向对象的程序的代码首先转换成有向图，再利用图文法的分析算法来识别出通用结构。

文[17,18]中使用类似的方法来研究软件工程中理解和识别的问题，识别的基本单位不仅仅局限于基本的数据结构，而已经扩展到模式。

结束语 图文法是定义可视化语言或定义图形转换系统的非常强大的工具，具有广阔的应用前景和理论研究价值。本文所使用的文法，虽然也是上下文相关的，并且具有上下文元素，但是其描述上下文的能力不足够，规则并不能有效地描述图中被规则匹配的节点已经存在的关系。如该节点和图中其他的节点已存在很多联系，但是在规则左部或右部中并不能完全表示出来。这样，在转换过程中，如果该节点被删除，则可能造成悬空边，所以在应用规则时对这些关系的处理就显得非常重要。而 edNCE[12,13] 文法在定义上下文元素时就很强大，但是对应的分析算法就很复杂。

未来的研究工作放在将各异的图文法定义统一起来，建立一套描述能力强且可有效分析的文法，并制定图形的重写机制，在此基础上共同研究开发有效的分析算法，降低规则匹配和规则选择的搜索空间。为满足实际的需要并将图文法技术应用到更广泛的领域，需要扩展其分析算法，能够进行增量的检查，这是开发语法制导的图形编辑器的关键所在。需要将前面分析步骤中的信息保存下来，并用来指导后面的分析，并且能够像文本语言的分析器那样，能够找到导致错误的图形语言的图样元素的最小集合，并且给用户提出反馈，指导用户修改。

参 考 文 献

1 Mehlhorn K. Graph Algorithm and NP-Completeness. In: Data Structures and Algorithms. volume 2. Heidelberg, Germany: Springer－Verlag, 1984
2 Rudolf M. Utilizing Constraint Satisfaction Techniques for Efficient Graph Pattern Matching. 6th International Workshop on Theory and Application of Graph Transformations, Paderborn (Germany), 1998
3 Ma X. Research on Software Coordination on Internet: [Ph D Dissertation]. Nanjing, Nanjing University, 2003
4 Ma X, Cao J, Lu J. Architecting Distributed Web Applications: A Graph-Oriented Approach. Chinese Journal of Computers, 2003, 26(9): 1104～1115
5 Prosser P. Hybird Algorithms for the Constraint Satisfaction Problem. Computational Intelligence, 1993, 9(3): 268～299
6 Dechter R. Backtracking algorithms for constraint satisfaction problems-a survey: [Technical Report]. University of California, Irvine, 1997
7 Bottoni P, Taentzer G, Schurr A. Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation. Proc. IEEE Symp Visual Languages, Seattle, Washington, 2000
8 Heckel R, Küster J M, Taentzer G. Confluence of Typed Attributed Graph Transformation Systems. In: Proc. of ICGT'2002, 2002. 161～176
9 Rekers J, Schürr A. A Graph Grammar Approach to Graphical Parsing. Proc VL'95 11th Int IEEE Symp on Visual Languages, Darmstadt, 1995
10 Rekers J, Schürr A. A Parsing Algorithm for Context-sensitive Graph Grammars: [Technical Report]. Leiden Univerisity, 1995
11 Zhang Da-qian, Zhang Kang, Cao Jiannong. A Context-Sensitive Graph Grammar Formalism for the Specification of Visual Language. Computer Journal, 2001, 44(3)
12 Ehrig H, Engels G, Kreowski H J, et al. Handbook of Graph Grammars and Computing by Graph Transformation. World Scientic, 1997
13 Adachi Y, Kobayashi S, Tsuchida K, et al. An NCE Context-Sensitive Graph Grammar for Visual Design Languages. In: Proc. IEEE Symposium on Visual Languages, 1999. 228～235
14 马晓星，余萍，陶先平，等. 一种面向服务的动态协同架构及其支撑平台. 计算机学报，2005(4)
15 Zhang Kang, Zhang Da-Qian. XML Transformations Through Graph Grammars. 2001 IEEE International Conference on Multimedia and Expo, Tokyo, Japan, 2001
16 Wills L M. Using Attributed Flow Graph Parsing to Recognize Cliches in Programs. In: Proc. 5th Int Workshop on Graph Grammars and their Application to Computer Science, 1996, 1073: 170～184
17 Irwin W, Cook C, Churcher N. Parsing and Semantic Modelling for Software Engineering Applications. 2005 Australian Software Engineering Conference, 2005
18 Costagliola G, De Lucia A, Deufemia V, et al. Design Pattern Recovery by Visual Language Parsing. Ninth European Conference on Software Maintenance and Reengineering, 2005
19 Baresi L, Heckel R, Sebastian Thones, et al. Style-Based Refinement of Dynamic Software Architectures. Fourth Working IEEE/IFIP Conference on Software Architecture, 2004