В.А. Маркин

ЯЗЫК ОПИСАНИЯ ГРАФОВЫХ МОДЕЛЕЙ И АЛГОРИТМОВ GRAMAL[1]

ВВЕДЕНИЕ

При построении программных систем различных уровней сложности часто и широко используются графовые модели и различные методы их обработки. Идея использования графовых схем для языков спецификации или в языках высокого уровня обсуждается более 30 лет. Но только недавно начали бурно развиваться различные средства разработки, анализа и тестирования на базе систем переписывания графов. Многие считали, что моделирование графов и использование систем переписывания графов ведет в тупик из-за NP-сложности многих графовых алгоритмов. Ситуация кардинально изменилась с появлением систем переписывания графов или систем на базе графовых грамматик, таких как PROGRES [1], PAGG [2] ,GraphEd [3], или подобных рассмотренным в [4, 5].

Графы, являясь очень удобным инструментом описания структур данных, различных видов связей, информационных потоков, широко используются в теории компиляции, в различных математических задачах. Часто при описании алгоритма на графах требуется наглядно представить, увидеть пошаговые фазы алгоритма и т.д.

Именно для этих целей создана система GRAMAL, преследующая четыре цели: предоставить инструмент для описания графовых моделей; графически представить данную модель, предоставить средства тестирования и отладки методов работы с графами, а также возможность генерации программного кода для последующего применения.

В основе системы GRAMAL лежит разработанный язык, поддерживающий средства описания соответствующих графовых схем и средства описания преобразований на графах. Помимо этого, интегрированная среда разработки, написанная для использования одноименного языка GRAMAL, содержит в себе текстово-графический редактор языка, средства просмотра структур языка, а также средства интерпретации и генерации конечного

[1] Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (грант № 01-01-794) и Министерства образования РФ.

кода на языке С. Используя систему GRAMAL, можно получить прототип независимого приложения на языке С, который в дальнейшем может быть откомпилирован внешним компилятором с языка С. Тем самым спецификация конструируемой системы может быть исполнена или протестирована конечным пользователем без специальных знаний по теории графовых грамматик.

Далее будут даны описания языковых конструкций языка GRAMAL, определены формат описания предопределенных типов языка (вершин, дуг и графа), способы инициализации и методы манипулирования данными структурами. Будет также приведено краткое описание инструментальной системы GRAMAL с кратким же изложением режимов системы, ее целей и решаемых ею задач.

1. ОПИСАНИЕ ЯЗЫКОВЫХ КОНСТРУКЦИЙ ЯЗЫКА GRAMAL

Система GRAMAL предоставляет средства для представления различных атрибутированных графовых моделей (состоящих из атрибутированных вершин и дуг, как ориентированных, так и неориентированных). Атрибуты прописываются как для вершин, так и для дуг, что отличает GRAMAL от многих систем. В атрибутах хранится дополнительная информация, которая напрямую не относится к структуре графа, такая как, например, в таблице идентификаторов. Другими словами, атрибуты содержат информацию, локальную для объекта. Дополнительные связи между вершинами, помимо задания типов инцидентных ребер, можно задавать атрибутами соответствующих типов ребер. Как и во всех различных системах [1–5], связанных с системами описания и манипулирования графовыми моделями bkb системами переписывания графов, выделяются три основных объекта — вершина, ребро (либо дуга) и сам граф. Тем самым в языке GRAMAL определяются три основных класса: node_class, edge_class и graph_class. По умолчанию в системе определены три базовых типа: Node, Edge и Graph, от которых создаются все соответствующие производные типы. Эти базовые типы отвечают за инициализацию и базовые настройки (какие именно, укажем позднее) объектов соответствующих типов. В отношении всех основных классов язык GRAMAL поддерживает принцип наследования. Так, описав один из типов вершин, мы можем создать производный класс от данного типа, наследующий все атрибуты и методы базового типа.

Любой производный тип (тип вершины, ребра или графа) содержит описания атрибутов и методов работы с данным типом. Дадим, например краткое описание графовой модели Бинарного дерева с соответствующими методами работы с ними.

```
// Схема Бинарного дерева

Node_class Node{  // Вершина дерева
        Attrib:
                Key int num;
        Condition:
                Numeration(int n){this.num==n};
                Leave{(not with –R_Rebro-) and (not with -L_Rebro-)};
};

Node_class Root:Node{ //Отдельно выделен тип корень дерева
        Condition:
                Root{(not with -L_Rebro->) and (not with –R_Rebro->) };
};

Edge_class Rebro(node_class Node => node_class Node){ //Соответствующий
//тип ребра дерева
        Attrib:
                String Label;
};

Edge_class L_Rebro:Rebro{ //Левое ребро
        Condition:
                L_Rebro{(<-).num <(->).num};
};

Edge_class R_Rebro:Rebro{ //Правое ребро
        Condition:
                R_Rebro{(<-).num >(->).num};
};
```

```
graph_class BiTree{

        attrib:
        Node Nodes;          //Описание множества вершин
        Root Roots;//Описание корней
        L_Rebro L_Edges;   //Множество левосторонних дуг
        R_Rebro R_Edges;   //Множество правосторонних дуг

        methods:

                path Node Find(int n):Node{
                        ( L_Rebro(n) | R_Rebro(n) )
                        };
                rule Root Create (void)
                        { //Пустая часть инициализации
                        }=>{ //Вторая часть правила
                        Root uzel = new Root; //Создаем вершину типа Root
                        Return uzel; //Возвращаем значение новой вершины
                        };
                rule void Insert ( Node Uzel, int N)
                        {
                                Node tmp;
                                Set Uzel;
                                Tmp=Find(N);
                        }=>{
                                Node tmp2=new Node;
                                Tmp2.num=N;
                                If (tmp.num>N)
                                        New  L_Rebro(tmp,tmp2);
                                Else
                                        New  R_Rebro(tmp,tmp2)
                        }

};
```

Описанная выше схема бинарного дерева практически в полной мере описывает многие аспекты языка GRAMAL.

Как видно из схемы, GRAMAL содержит синтаксические конструкции для определения типизированной графовой схемы. В такой схеме описаны соответствующие типы вершин и дуг, возможности и условия возникающих связей между ними. Раздел TYPES_NODES содержит объявления типов вершин (метки и атрибуты вершин, состоящие из имен и областей видимости). Раздел TYPES_EDGES определяет метки ребер и, что важно, типы вершин, которые данный тип ребер соединяет.

Ниже приведем пример схемы, описывающей систему таблиц при компиляции программ:

```
Section  CLASS_NODES
{
    node_class GRAPH_NODE;
    node_class MODULE: GRAPH_NODE;
    node_class LIST_HEAD: GRAPH_NODE;
}
Section TYPES_NODES
{
    typedef node_type Ident GRAPH_NODE;
    typedef node_type Function_Module MODULE;
    typedef node_type Data_Type_Module MODULE;
    typedef node_type Data_Object_Module MODULE;
    typedef node_type Ident_List LIST_HEAD;
    typedef node_type Proc_List LIST_HEAD;
    typedef node_type Implementation GRPH_NODE;
}

Section TYPES_EDGES
{
    typedef edge_type ToIdent: node_class MODULE => node_class Ident;
    typedef edge_type ToBaseOn: node_class MODULE => node_class
Ident_List;
    typedef edge_type  ToExport: node_class MODULE => node_class Proc_List;
    typedef edge_type  ToContains: node_class MODULE => node_class
Ident_List;
    typedef edge_type ToImport: node_class MODULE => node_class Ident_List;
    typedef edge_type  ToImplementation: node_class MODULE => node_class
Implemetation;
}
```

Разделы TYPES_NODES и TYPES_EDGES необходимы для более упрощенного задания формализма при описании графовой схемы. Очень часто встречается, что при определении графовой схемы в общем случае определяется большое число типов вершин и ребер, отличающихся друг от друга частичными параметрами. Поэтому систему GRAMAL отличает от других возможность задания базовых и порожденных типов объектов, что делает описываемую схему более компактной и легко определяемой. Так, например, во время трансляции можно получить несколько типов таблиц (Function_Module, Data_Type_Module, Data_Object_Module). В нашем примере все они наследуются от типа MODULE, чем мы уменьшили описание допустимых типов ребер в разделе TYPES_EDGES, потому что такие типы ребер, как ToIdent, ToBasedOn, ToExport и т.д., для всех описанных типов модулей определили один раз вместо трех.

Как и в объектно-ориентированных языках GRAMAL, предоставляет возможность строить иерархию типов вершин и ребер.

Наследование позволяет уменьшить размер описания графовой схемы, по сравнению с тем, как это происходило бы при использовании только базовых типов вершин и ребер.

Помимо этого наследование позволяет выделить общие атрибуты объектов. Так, например, если мы определим две таблицы — Типов и Идентификаторов — как производные от типа LIST_ELEM, то общим у них будет атрибут — имя объекта NAME, который мы и можем приписать к множеству атрибутов типа LIST_ELEM.

И наконец, в GRAMALe можно определить область определения атрибута. Допускаются два возможности: 1) user тип — атрибут определяется пользователем, его можно инициализировать и задавать значение; 2) derived тип — атрибут задается или вычисляется системой по определенным правилам, заданным при описании типа объекта. Примером первого типа атрибутов может служить Наименование Идентификатора NAME_Ident в заданной таблице идентификаторов, а второй — заданная на графе разметка.

```
Section CLASS_NODES
{
        node_class IDENT:GRAPH_NODE
        {
                user Name: string="";
        }
…….
```

```
        node_class LIST_ELEM:GRAPH_NODE
        {
                derived Num: int = [<=ToNext=.Num +1 | 1];
        }
}
```

Из приведенной в начале главы схемы бинарного дерева видно, что описание любого из видов предопределенных классов состоит из трех частей: раздела атрибутов, раздела условий (правил специального вида) и раздела методов (процедур, связанных с данным классом).

В разделе атрибутов описываются обычные информационные поля, имеющие предопределенные типы данных с дополнительным ключом Key. Данный ключ просто указывает системе, что при создании определенного объекта надо явно указать значение данного поля. При создании, скажем, вершины во множестве вершин данного класса система должна проверить уникальность данного идентификатора (используя терминологию баз данных). Данное поле может быть только одно, а при его отсутствии считается, что существует целочисленный нумератор (еще раз напомним, что производные классы наследуют атрибуты базового класса).

В разделе условий описываются специальные правила, которые, в зависимости от значения условного выражения, меняют состояние (контекст) класса.

Под состоянием класса мы подразумеваем:

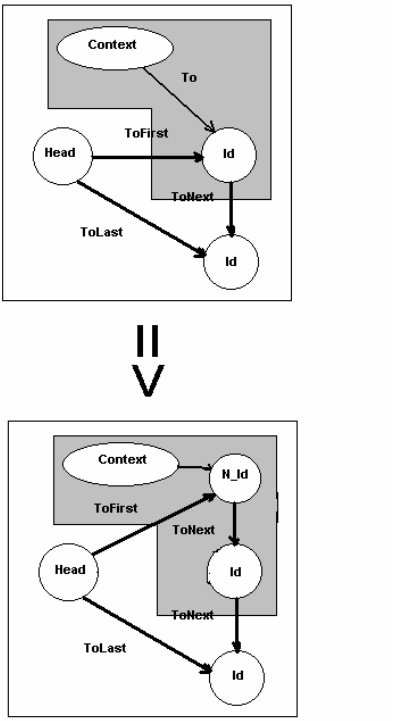
- для типов вершин — указатель на данную вершину;
- для типов дуг — указатели на смежные вершины и указатель на текущую вершину;
- для графов — совокупность состояний всех типов вершин и дуг.

В описанном выше примере два условия L_Rebro и R_Rebro на входе получают целочисленное значение, и если условие верно, то меняют контекст и возвращают указатель на смежную вершину.

В разделе методов, существующих только для графовых классов, описываются правила преобразования графа, где под преобразованием будем понимать изменение контекстов и множеств вершин либо дуг. Все методы работы с графом разделим на три вида: path (изменение контекста), rule (правило преобразования графа), и transform (более сложное преобразование, состоящее из совокупности правил (rules)).

Для методов типа *path* необходимо указать два контекста: входной контекст входной вершины и контекст выходной вершины. Если при вызове данного метода на входе указать явно объект данного типа вершины, то контекст, связанный с данным типом вершин, меняется на входной объект. В приведенном примере при описании метода Find встречаем альтернативный оператор, где перебираются все выполнимые альтернативы.

Основными методами являются правила, методы типа *rule*, каждое из которых состоит из двух частей: выделение подграфа для преобразования и само преобразование. Если первая часть — выделение подграфа для преобразования — не выполняется, то правило отклоняется. В одном из методов используется оператор Set, который устанавливает соответствующий контекст в системе. Графически правило преобразования rule на примере вставки элемента в список приведено на рисунке.



Последний вид методов — *transform* (преобразования) — является совокупностью правил либо преобразований.

Любая программа на языке GRAMAL состоит из последовательности операторов описания типов, объектов и процедур. В языке, помимо описанных выше специализированных типов, переменные могут быть стандартными типами: int, char, float, string и т.д.

Входом в точку программы является процедура *main*.

2. ФУНКЦИОНАЛЬНЫЕ ОСОБЕННОСТИ СИСТЕМЫ GRAMAL

Помимо задачи описания грамматических особенностей специализированного языка вторым этапом работы над данным проектом являлось создание инструментальной системы GRAMAL.

Основными целями данной системы являются средства отладки и визуализации входных задач, написанных на языке GRAMAL, а также генерация программного кода (например, на языке C) для дальнейшего использования наработанной библиотеки в других программных проектах.

Система, таким образом, находится в одном из трех состояний: отладки, трассировки (либо визуализации) и генератора. Соответственно среда GRAMAL состоит из трех основных частей: редактора, интерпретатора и back-end на языке C.

В режиме отладки пользователь находится в окне редактирования, редактируя программный код. По возможности пользователю предоставляется графический контекстный интерфейс с выделением специализированных конструкций языка. Используя средства графического представления графовых объектов, пользователь может быстро создать прототип графовой модели своей задачи либо описать входные данные для нее. Спецификация в системе GRAMAL создается в две стадии. Первая стадия описывает саму графовую схему, а во второй создаются все операции над этой схемой. Виды данных операций можно найти в описании грамматики языка. Типы узлов в описании графовой схемы могут рассматриваться как абстрактные классы в объектно-ориентированных языках, которые включают в себя необходимые атрибуты данного типа узла. Все типы узлов обозначаются прямоугольниками. Если между типами вершин существуют связи наследования, то они указываются жирными стрелками. Для обозначения связей между вершинами или ребрами используются одинарные стрелки, двойной щелчок на которых вызывает дополнительное окно, содержащее атрибуты данного ребра. Построив схему, пользователь переходит к построению пра-

вил преобразований над ней. Каждое такое правило состоит из двух частей: левой и правой (смотри описание грамматики). Для данной задачи пользователь вызывает дополнительное окно, в котором можно описать данное правило как графически, так и через текстовое представление. Нужно отметить, что в любой момент, пользователь для любого объекта схемы может получить back-end на языке C.

По окончании редактирования пользователь должен оттранслировать программу, получив необходимую информацию об ошибках, для дальнейшей отладки.

Как только программа будет отлажена, пользователь может перейти в режим интерпретации программы для анализа запрограммированной графовой задачи (описанных графовых моделей и методов). Данный режим называется еще режимом визуализации, в котором пользователю должен быть предоставлен графический вид графовой задачи с соответствующими методами печати, просмотра и сохранения в графических форматах. При интерпретации программы пользователь с помощью расставленных брейкпойнтов либо пошаговой отладки вызывает дополнительные окна, в которых показано текущее состояние схемы, а цветом указаны текущие контексты вершин и ребер.

И последний режим генератора переводит программу на языке GRAMAL в конечный язык программирования C, где описаны соответствующие структуры и методы, а точкой входа является процедура GRAMAL. В дальнейшим пользователь может использовать сгенерированную процедуру в иных программных проектах.

ЗАКЛЮЧЕНИЕ

Целью системы GRAMAL является достижение простоты в реализации и анализе различных алгоритмов на графовых моделях. С помощью данной системы пользователь может быстро и без каких-либо дополнительных знаний в программировании создать прототип своей задачи, если она описывается графовыми схемами. При необходимости пользователь сможет получить прототип своей задачи, написанный на языке C++, для дальнейшего использования в каких-либо других внешних программных системах.

Развитие системы рассматривается в двух направлениях. Первое включает введение в систему дополнительных функциональных инструментов, таких как анализатор корректности правил трансформации схемы, создание библиотек графовых схем, экспорт в язык Java. Помимо этого рассматрива-

ется исследование по явному описанию и внедрению в систему средств объектно-ориентированного программирования. Второе направление включает реализацию многоплатформенности инструментальной системы, т.е. написание графических компонент системы под Windows и Linux.

СПИСОК ЛИТЕРАТУРЫ

1. **Engels G. et al.** Building integrated software development environments. Part 1: Tool specification // ACM Trans. on Software Engineering and Methodology. — 1992. — Vol. 1, N 2. — P. 135–167.
2. **Gottler H., Gunther J., Nieskens. G.** Use graph grammars to design CAD-systems // Lect. Notes Comput. Sci. — 1991. — Vol. 532. — P. 396–410.
3. **Himsolt M.** GraphEd: An interactive graph editor // Lect. Notes Comput. Sci. — 1989. — Vol. 349. — P. 532–533.
4. **Nagl M., Schurr A., Munch M.** Applications of graph transformations with industrial relevance (International Workshop, AGTIVE 99) // Lect. Notes Comput. Sci. — Vol. 1779.
5. **Theory** and application of graph transformations / Ed. By Ehrig H. et al. // Proc. 6th Internat. Workshop TAGT 98. — Berlin a.o.: Springer-Verlag, 1998. — (Lect. Notes Comput. Sci.; Vol. 1764).