УДК: 004.855.5

Вывод древовидной вспомогательной грамматики как инструмент автоматического обобщения алгоритмов

Л. О. Воробьёв, А. В. Григорьев
Донецкий национальный технический университет, г. Донецк
e-mail: lev.vorobyov@rambler.ru , grigorievalvl@gmail.com

Аннотация.
В статье рассматривается проблема автоматического обобщения алгоритмов в процедурных языках программирования. Предлагается уникальный способ объединения семантики алгоритмов путём представления программы в виде рекуррентной функции в обратной польской нотации и вывод древовидной вспомогательной грамматики (TAG) с образованием И-ИЛИ дерева с помощью алгоритма теоретико-множественных операций над формальными грамматиками. Приводится описание И-ИЛИ графа внутреннего представления алгоритма и основные этапы обобщения программ. В настоящее время алгоритм не апробирован.

Введение

Основы теории алгоритмов и формальных грамматик описаны в [1, 2] соответственно.

В предлагаемой работе рассматривается вопрос синтеза алгоритмов решения общей задачи по примерам решений частных задач путём объединения формальных грамматик. Задача объединения формальных грамматик подробно описана в [3], реализации некоторых алгоритмов обобщения грамматик на языке Python приведены в [4].

При написании программы на императивном языке программирования (ИЯП) иногда возникает задача объединить несколько алгоритмов с одинаковым интерфейсом (набором аргументов). В результате объединения получается обобщённая реализация алгоритма. Инструментами обобщения алгоритмов могут быть условное выполнение операторов и шаблонные функции [5]. Существующие решения задачи автоматического обобщения программ [6, 7] используют вероятностные методы. Детерминированные алгоритмы обобщения программ в литературе освещены слабо, что обуславливает **актуальность** формализации процесса обобщения программ.

Введём следующее определение. **Обобщённая программа** — такая программа, которая решает все задачи из некоторого класса. Например: пусть есть множество задач C и есть два собственных подмножества A и B, при этом может быть $A \cap B \neq \emptyset$ и обычно $A \cup B \neq C$. Задача обобщения состоит в том, чтобы из двух программ, решающих частные задачи из множества A и B соответственно, составить программу, которая решает все задачи из множества C. Это и будет обобщённая программа.

Мы предполагаем, что обобщённые программы относятся к одному классу, т.е. программы, решающие сходную задачу. Таким образом, считается, что обобщённая программа не может включать алгоритмы из различных классов. В таком случае, для каждого класса задач создаётся отдельная обобщённая программа. Классификация задач — это предварительный этап, который нужно выполнять перед обобщением программ.

Цель работы: автоматизировать процесс обобщения алгоритмов.

Исходя из поставленной цели предполагается решить следующие задачи:

- привести справочный материал по выводу грамматик, древовидным грамматикам и автоматическому программированию;
- разработать математическую форму представления обобщённых алгоритмов;
- проверить полноту по Тьюрингу предлагаемого математической формы представления алгоритма;
- описать процедуру обобщения алгоритмов во внутреннем представлении;
- описать процедуру перевода текстов программ во внутреннее представление и обратно.

Краткий анализ существующих решений

Задача обобщения программ относится к более общей задаче автоматического программирования. Вероятностные методы автоматического программирования описаны в [8, 9]. Они основаны на случайной генерации и оценке возможных решений по некоторому критерию, Обычно для оценки возможных решений в автоматическом программировании программу проверяют на наборе примеров входов и выходов. Здесь же рассматривается вывод обобщённой программы по примерам

программ. Поскольку программа состоит из слов (термов), то она является сентенциальной формой из некоторого формального языка. Таким образом, обобщение программ есть частный случай грамматического вывода (grammatical inference, GI).

Индуктивный вывод грамматики - это разновидность задачи машинного обучения по примерам, когда в качестве примеров предоставляется набор синтаксически правильных цепочек термов, и строится распознающий автомат, или продукционные правила порождения этих примеров. Алгоритм синтеза распознающего автомата с помощью обучения рекуррентной нейронной сети [10] относится к вероятностным методам обучения, и позволяет выводить грамматику ограниченной сложность (не более 5 состояний автомата), и подходит только для вывода регулярной грамматики. Вывод формальной грамматики, по сути является сжатием информации. Ведь исходные сентенциальные формы имеют больший размер в совокупности, чем продукционные правила порождения этих форм, потому что сентенциальные формы обычно содержать повторяющиеся фрагменты, которые выявляет алгоритм вывода грамматики.

Существует также детерминированный алгоритм построения продукционных правил формальной грамматики по примерам порождаемых ею слов, впервые описанный в [11]. Проблема этого алгоритма состоит в том, что он не создаёт грамматику с рекурсивными правилами. Рекурсивные правила — это правила, в которых в левой и правой части содержится один и тот же нетерминальный символ [2 с. 19]. Поскольку количество операторов в одной программе обычно ничем не ограничено то для описания формальной грамматики, сентенциальные формы которой являются программами, нужны рекурсивные правила.

Древовидная вспомогательная грамматика (tree-adjunct grammars, TAG) - это система переписывания деревьев [12], введённая Аравиндом Джоши в 1975 г. для описания естественного языка. Формально TAG является пятёркой (T, V, I, A, S), где: T — множество терминальных символов; V — множество нетерминальных символов; $S \in V$ — начальный символ; I — множество начальных деревьев, корень которых помечен S, листья — терминальными символами, а остальные вершины — нетерминальными символами; A — множество вспомогательных деревьев, листья которых помечены терминальными и нетерминальными символами, а промежуточные узлы - только нетерминальыными символами.

Особенностью вспомогательных деревьев TAG является наличие одного листа, который помечен тем же символом, что и корень дерева. Этот лист называется ногой. Порождение дерева в TAG осуществляется путём замещения поддерева от нетерминального узла вспомогательным деревом, у которого корень соответствует тому же нетерминальному символу. А исходное поддерево присоединяется к ноге вспомогательного дерева. Это позволяет порождать бесконечные языки, что нужно для описания грамматики обобщённых программ. В [12] описывается метод автоматического программирования AntTAG, использующий TAG в качестве способа представления программы, который позволяет представлять дерево программы, как линейную последовательность термов для автоматического программирвоания путём муравьиной оптимизации.

Ещё одной формой линейного представления программы является польская запись. Польская запись - это запись, в которой операторы следуют за или перед операндами. Бывает прямая и обратная польская запись. Примером прямой польской записи является язык Lisp.

Если представить вспомогательное дерево TAG, как рекурсивную функцию, где листья — это операнды, промежуточные узлы — это операции, а корень — это точка входа в программу, и использовать обратную польскую запись, то получим линейную последовательность термов. Это позволяет использовать алгоритм теоретико-множественных операций над формальными грамматиками [11], как способ вывода вспомогательных деревьев TAG для обобщения программ.

Предлагаемый подход

Предлагаемое решение состоит в том, что обобщаемые примеры программ представляются в виде сентенциальной формы некоторой формальной грамматики. Из набора сентенциальных форм выводится эта формальная грамматика, которая затем переводится в набор обобщённых подпрограмм.

Обобщаемые примеры программ назовём прототипами. Прототип имеет номер в базе данных. Из прототипов методом синтаксического и семантического анализа можно извлечь алгоритм. Для внутреннего представления алгоритма используем И-ИЛИ граф, как способ хранения атрибутной грамматики. Узлы И-ИЛИ графа помечены номерами прототипов, а дуги от ИЛИ-узлов — значениями параметров.

Для обобщения программ нужно исходные прототипы различать с помошью некоторого критерия. Для указания, какими параметрами обладает конкретный прототип, используется два И-ИЛИ дерева: дерево выбора прототипов и дерево порождения прототипов.

И-ИЛИ дерево порождения прототипов задаёт множество синтаксически правильных выражений, определённых на алфавите терминальных символов. Подмножеством множества синтаксически правильных выражений является семантически правильные выражения, которые задаются номерами прототипов в ИЛИ-узлах. При нисходящем выводе пользователь выбирает одну из альтернатив в дереве выбора прототипов. ИЛИ-узлы с номерами прототипов, записанными в остальных альтернативах, в дереве порождения прототипов удаляются [13].

Рассмотрим пример обобщённого алгоритма сложения n чисел, выраженного в виде рекурсивной функции f от переменного числа аргументов:

$$\begin{Bmatrix} f(x_1) = x_1; \\ f(x_1, x_2, \dots, x_n) = x_1 + f(x_2, \dots, x_n), \end{Bmatrix}. \quad (1)$$

Здесь параметром, от которого зависит выбор вида функции, является количество аргументов x_i. Таким образом, одна из исходящих из корневого ИЛИ-узла дерева на рис. 1 дуг будет помечена значениями параметра n = 1, а вторая не будет помечена, и соответствует всем остальным значениям этого параметра.

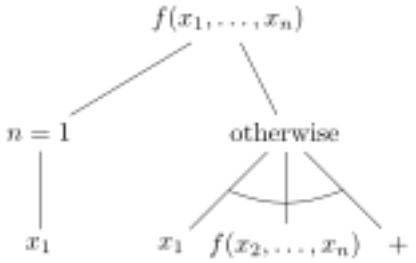

Рисунок 1 — Представление алгоритма в виде И-ИЛИ дерева

Здесь циклы устраняется путём добавления новых символов операций в алфавит и превращения узла, создающего циклическую связь в лист. Впоследствии, при генерации программы этот лист заменяется на рекурсивный вызов подпрограммы. Таким образом, на рис. 1 изображен пример вспомогательного дерева для TAG, полученной в результате вывода грамматики из И-ИЛИ дерева прототипов.

ИЛИ-узлы транслируются в блоки условного выполнения операторов, а И-узлы — в последовательность выполнения операторов в обратной польской нотации.

И-ИЛИ граф есть структурный граф программы [14], т.е. связный помеченный ориентированный граф, такой, что у него есть вершина, из которой достижима любая вершина; и есть хотя бы одна вершина, которая имеют нулевую степень исхода. Каждая вершина И-ИЛИ графа, с ненулевой степенью исхода, является либо И-вершиной, либо ИЛИ вершиной. Для представления алгоритмов применяется И-ИЛИ граф со следующими особенностями:

— узлы графа с нулевой степенью исхода помечаются терминальными символами языка записи рекуррентных формул в обратной польской нотации, т.е. переменные и операции;

— исходящие из И-узла дуги связаны с вершинами, которые помечены порядковым номером;

— порядок следования подчинённых узлов по И определяет порядок записи выражений в рекуррентной формуле;

— исходящие из ИЛИ-узла дуги связаны с узлами, помеченными списком номеров прототипов, и называются ИЛИ-альтернативами [13];

— ИЛИ-узел трактуется как рекуррентная формула с набором альтернативных случаев;

— если из И-узла исходят дуги, связанные с узлами с нулевой степенью исхода, то этот И-узел трактуется как рекуррентная формула, иначе это последовательность рекуррентных формул;

— циклические связи между узлами означают рекуррентное вычисление формул.

Между ИЛИ-узлами разных деревьев установлены отношения соответствия с помощью номеров прототипов. При выборе альтернативы в одном дереве все альтернативы в другом дереве, не связанные с выбранной альтернативой номерами прототипов, удаляются.

Это позволяет определить отношение ЕСЛИ-ТО между двумя языками: языком спецификаций и языком прототипов.

По факту, продукционная база знаний, то есть набор продукций ЕСЛИ-ТО реализуется на алгоритмическом языке программирования с помощью конструкции if-else.

Каждому ИЛИ-узлу в дереве спецификаций соответствует хотя бы один узел в дереве прототипов.

Проверка полноты И-ИЛИ графа для представления алгоритмов

Теорема 1. Представленная выше нотация записи алгоритмов является полной по Тьюрингу.

Доказательство: рекуррентные формулы являются полными по Тьюрингу вариантами записи алгоритмов. Следовательно, И-ИЛИ дерево, задающее в компактном виде набор рекуррентных формул, является полным по Тьюрингу.

Следствие. Программу на универсальном языке программирования можно перевести в описанное выше И-ИЛИ дерево и наоборот.

Описанное выше задание алгоритма соответствует принципам, которые положены в основу функционального программирования и языка Haskell.

Задача состоит в том, чтобы исходные примеры программ преобразовать в И-ИЛИ деревья синтаксического разбора, разделить их на три И-ИЛИ дерева для внешней границы, состава блоков и связей, затем обобщить полученные деревья, получив обобщённые три И-ИЛИ дерева, которые затем объединить в одно И-ИЛИ дерево алгоритма и перевести его в программу.

Этапы обобщения примеров программ:

— кластеризация примеров по признаку наличия и отсутствия параметров.

— синтаксический и семантический анализ примеров с построением набора рекурсивных функций в обратной польской нотации;

— поиск одинаковых фрагментов в цепочках термов, и создание для каждого из них нового продукционного правила до тех пор, пока не получится один нетерминальный символ;

— представление грамматики в виде И-ИЛИ дерева, выявление повторяющихся поддеревьев и вспомогательных деревьев TAG;

— перевод вспомогательных деревьев полученной TAG в рекурсивные функции и генерация программы.

Пример обобщения набора программ

Алгоритм обобщения функций получает набор программ в виде цепочек терминальных символов, и значения некоторого параметра, от которого зависит вид этой программы. Затем алгоритм производит выведение общего алгоритма и возвращает полученный алгоритм в форме рекуррентного выражения. Например, пусть необходимо вывести общий алгоритм сложения *N* чисел (не важно, какого типа). В исходном виде функции представляют собой цепочки из элементов массива и знаков сложения, как показано в табл. 1.

Таблица 1 - Входной набор прототипов и их свойств

№	Выражение для *f(X,N)*	Свойство *N*
1	x_1	1
2	$x_1 + x_2$	2
3	$x_1 + x_2 + x_3$	3
4	$x_1 + x_2 + x_3 + x_4$	4
5	$x_1 + x_2 + x_3 + x_4 + x_5$	5

Здесь в качестве параметра, от которого зависит программа, выступает N — количество слагаемых. В результате обобщения нужно получить рекуррентную функцию, форма которой не зависит от количества слагаемых. Эта функция имеет вид:

$$f(\{x_i\}_i^n, n) = \begin{cases} f(\{x_i\}_i^{n-1}, n-1) + x_n, n > 1; \\ x_1, n = 1. \end{cases} \quad (2)$$

Сначала свойства прототипов используются для построения дерева решений по алгоритму C4.5. В результате получаем следующее дерево, в листьях которого находятся номера прототипов, ИЛИ-узлы помечены свойствами, в дуги от ИЛИ-узлов — значениями свойств.

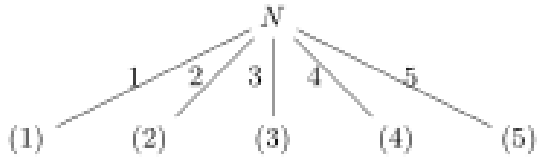

Рисунок 2 — Дерево решения для выбора прототипов

Параллельно исходные прототипы переводятся в обратную польскую запись:

Таблица 2 - Промежуточное представление прототипов

№ прототипа	Обратная польская запись
1	x_1
2	$x_1, x_2, +$
3	$x_1, x_2, x_3, +, +$
4	$x_1, x_2, x_3, x_4, +, +, +$
5	$x_1, x_2, x_3, x_4, x_5, +, +, +, +$

На основе данных примеров строится грамматика в виде И-ИЛИ дерева (рис. 3).

Это недоообученное И-ИЛИ дерево, потому что оно включает только те примеры, которым его обучили, а нужно создать такую модель, которые включают все возможные варианты задач данного класса.. Алгоритм унификации должен выявить сходство и построить более простую порождающую грамматику.

Индуцирование проводится с помощью метода, описанного в [11]. Алгоритм индуцирует регулярную грамматику с не более чем 5 состояниями. В данном случае прототипы в обратной польской записи определяются следующей КЗ грамматикой G = (VN,VT,P,S) (3).

Это атрибутная грамматика. Здесь каждому нетерминальному символу соответствует атрибута *N*, и альтернативные правила применяются при наличии определённого значения атрибута.

Конечный автомат, распознающий эту грамматику, содержит ровно пять состояний.

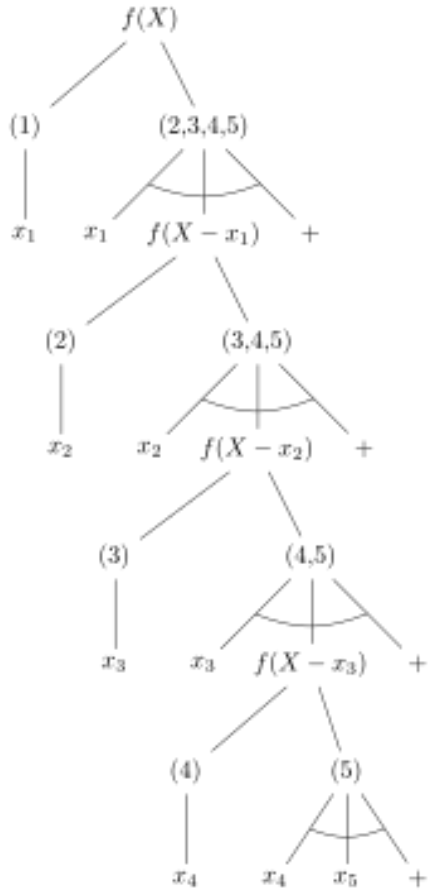

Рисунок 3 — Недообученное И-ИЛИ дерево прототипов

$$VN = \{S, A, B, C, D\};$$
$$VT = \{x_1, x_2, x_3, x_4, x_5, +\};$$
$$P = \begin{cases} S \to x_1 \vee x_1 A; \\ A \to x_2 + x_2 B+; \\ B \to x_3 + x_3 C+; \\ C \to x_4 + x_4 D+; \\ C \to x_5+. \end{cases}. \quad (3)$$

Тот же метод, предложенный в [10], вполне способен индуцировать такую грамматику, но он пригоден только для грамматики регулярного типа с числом состояний не больше пяти.

В результате выполнения алгоритма унификации получается И-ИЛИ дерево:

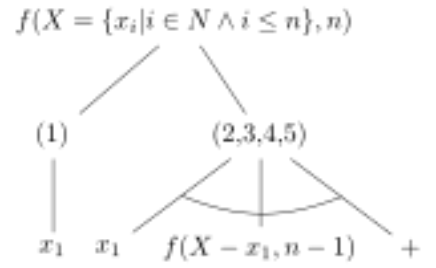

Рисунок 4 — Дообученное И-ИЛИ дерево прототипов

В скобках указывается номера прототипов. Это И-ИЛИ дерево соответствует алгоритму, выраженному в форме рекуррентной функции в (1).

Выражение в блоке if вычисляется исходя из свойств прототипов, номерами которых помечены ИЛИ-альтернативы.

Обобщение программы для всего класса задач, а не только для объединения подмножеств решаемых примерами задач, достигается путём анализа отрицательных примеров.

Заключение

Авторами предложен новый способ обобщения функций в алгоритмических языках программирования. Предлагаемый алгоритм опирается на метод вывода регулярной грамматики.

Отличия состоят с вследующем:

– сентанциальные формы являются рекуррентными функциями в обратной польской нотации;

– алгоритм обрабатывает исходные линейные примеры и строит грамматику;

– для устранения рекуррентных связей программы представляются в виде вспомогательных деревьев TAG.

Данные отличия позволяют использовать алгоритм вывода грамматики, рассчитанный на вывод нерекурсивных правил.

В настоящее время этот алгоритм не был апробирован. Планируется проверить предлагаемый подход в программной реализации на языках Perl и PL/SQL. Грамматический анализатор примеров планируется реализовать с помощью инструментов lex и yacc.

Литература

1. Поляков, В. И. Основы теории алгоритмов / В. И. Поляков, В. И. Скорубский. – СПб : СПб НИУ ИТМО, 2012.

2. Лаздин, А. В. Формальные языки, грамматики, автоматы. / А. В. Лаздин. – СПб : Университет ИТМО, 2019.

3. de la Higuera C. Grammatical Inference: Learning Automata and Grammars. / C. de la Higuera. – Cambridge : Cambridge University Press, 2010.

4. Wojciech, W. Grammatical Inference. Т. 673 / W. Wojciech. – 2017.

5. Вандевурд, Д. Шаблоны C++. Справочник разработчика. / Д. Вандевурд, Н. М. Джонаттис, Д. Грегор. – Изд. 2. – СПб. : ООО «Альфа-книга», 2018. – 848 с.

6. Helmuth, T. Improving generalization of evolved programs through automatic simplification / T. Helmuth, N. Mcphee, E. Pantridge, L. Spector. – 2017. – С. 937-944.

7. Xu D. Neural Task Programming: Learning to Generalize Across Hierarchical Tasks / D. Xu, S. Nair, Y. Zhu [и др.]. – 2017.

8. Elleuch, S. From Metaheuristics to Automatic Programming / S. Elleuch, B. Jarboui, P. Siarry. – 2022. – С. 3-38.

9. O'Neill, M. Automatic programming: The open issue? / M. O'Neill, L. Spector // Genetic Programming and Evolvable Machines. – 2020. – Т. 21.

10. Грачев, П. Г. Генерация автоматов на основе рекуррентных нейросетей и автоматического выбора кластеризации [Текст: электронный] / П. Г. Грачев, С. Б. Муравьев, А. А. Фильченков, А. А. Шалыто// Информационно-управляющие системы, 2020. – № 1 (104). – URL: https://cyberleninka.ru/article/n/generatsiya-avtomatov-na-osnove-rekurrentnyh-neyrosetey-i-avtomaticheskogo-vybora-klasterizatsii

11. Григорьев, А. В. Алгоритм выполнения теоретико-множественных операций над грамматиками в среде специализированной оболочки для создания интеллектуальных САПР / А. В. Григорьев // Наукові праці Донецького національного технічного університету : Проблеми моделювання та автоматизації проектування. – Донецьк : ДонНТУ, 2002. – С. 83-93.

12. Abbass, H. AntTAG: a new method to compose computer programs using colonies of ants / H. Abbass, N. Hoai, R. McKay. – 2002. – Т. 2. – С. 1654-1659.

13. Григорьев, А. В. Комплекс средств и методов работы с формальными грамматиками в семиотической концептуальной модели предметной области интеллектуальных САПР / А. В. Григорьев // Информатика и кибернетика. – Донецк: ДонНТУ, 2017. – № 1(7). – С. 46-72.

14. Крицкий, С. П. Реализация оптимизирующих преобразований программ с помощью структурных предикативных грамматик / С. П. Крицкий, Б. Ю. Тапкинов. – Текст: электронный // Известия вузов. Северо-Кавказский регион. Серия: Естественные науки. – 2006. – № S1. – URL: https://cyberleninka.ru/article/n/realizatsiya-optimiziruyuschih-preobrazovaniy-programm-s-pomoschyu-strukturnyh-predikativnyh-grammatik.

Воробьёв Л. О., Григорьев А. В. Вывод древовидной вспомогательной грамматики как инструмент автоматического обобщения алгоритмов. *В статье рассматривается проблема автоматического обобщения алгоритмов в процедурных языках программирования. Предлагается уникальный способ объединения семантики алгоритмов путём представления программы в виде рекуррентной функции в обратной польской нотации и вывод древовидной вспомогательной грамматики (TAG) с образованием И-ИЛИ дерева с помощью алгоритма теоретико-множественных операций над формальными грамматиками. Приводится описание И-ИЛИ графа внутреннего представления алгоритма и основные этапы обобщения программ. В настоящее время алгоритм не апробирован.*

Ключевые слова*: древовидная вспомогательная грамматика, вывод грамматики, автоматическое программирование, И-ИЛИ дерево.*

Vorobyov L. O., Grigoriev A. V. Tree adjunct grammar inference as a tool for automatic generalization of algorithms. *The article considers the problem of automatic generalization of algorithms in procedural programming languages. A unique way of combining the semantics of algorithms is proposed by representing the program as a recurrent function in reverse Polish notation and deriving a tree adjunct grammar (TAG) with the formation of an AND-OR tree using the algorithm of set-theoretic operations on formal grammars. A description of the AND-OR graph of the internal representation of the algorithm and the main stages of program generalization are given. At present, the algorithm has not been tested.*

Keywords*: tree adjunct grammar, grammar inference, automatic programming, AND-OR tree.*

Статья поступила в редакцию 22.02.2023
Рекомендована к публикации профессором Зори С.А.