Секция 4

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ И ПРОГРАММИРОВАНИЯ

УДК 510.52+004.051

FPT-АЛГОРИТМЫ И ИХ КЛАССИФИКАЦИЯ НА ОСНОВЕ ЭЛАСТИЧНОСТИ

В. В. Быкова

Идея параметризированного подхода к оценке сложности алгоритмов и задач была предложена в 90-х годах прошлого столетия [1]. За последние два десятилетия эта идея нашла применение в различных областях компьютерных наук, таких, как криптография, искусственный интеллект, вычислительная биология, теория баз данных [2]. Суть подхода заключается в рассмотрении так называемых параметризированных задач и параметризированных алгоритмов.

Параметризированная задача Π состоит в том, что для заданных языка $L(\Pi) \subseteq \Sigma^* \times \mathbb{N}$, где Σ — некоторый конечный алфавит и $\mathbb{N}$ — множество всех неотрицательных целых чисел, и пары $(I, k) \in \Sigma^* \times \mathbb{N}$ требуется определить, является ли (I, k) элементом $L(\Pi)$. Для алгоритма α, решающего данную задачу, (I, k) называют входом, I — его основной частью, $n = |I|$ — длиной входа и число k — параметром задачи. Алгоритм α называют параметризированным, если его вычислительная сложность, определяемая как количество времени исполнения, оценивается с точки зрения длины входа n и значения параметра k. Таким образом, функция вычислительной сложности параметризированного алгоритма есть некоторая функция двух переменных $t(n, k)$.

Параметризированная задача Π считается разрешимой с фиксированным параметром, или FPT-разрешимой (*Fixed-Parameter Tractable*), если она может быть решена некоторым параметризированным алгоритмом за время $t(n, k) = \mathrm{O}(n^{\mathrm{O}(1)} \cdot f(k))$ для функции f, зависящей только от параметра k. Класс всех разрешимых с фиксированным параметром задач обозначается FPT. Соответствующие параметризированные алгоритмы, решающие такие задачи, называют FPT-алгоритмами. Очевидно, что в FPT лежат все полиномиально разрешимые задачи. Однако наибольший интерес представляют FPT-разрешимые задачи, являющиеся NP-трудными. С теоретической точки зрения все эти задачи могут быть решены за полиномиальное время при каждом фиксированном значении параметра. Уже имеются сборники параметризированных задач, включая FPT-разрешимые задачи [3].

Теория параметризированной сложности развивается по нескольким направлениям: определение иерархии классов сложности параметризированных задач, установление условий FPT-разрешимости, выявление взаимосвязи между параметризированной сложностью и классами приближенных алгоритмов, развитие методов анализа и разработки параметризированных алгоритмов. В многочисленных работах по параметризированной теории сложности для анализа и сравнения алгоритмов применяются методы классической (одномерной) теории сложности [4]. Между тем классическая одномерность ограничивает глубину анализа параметризированных алгоритмов, для которых вычислительная сложность описывается функцией от двух переменных $t(n, k)$.

В настоящей работе предлагается мера вычислительной сложности алгоритма, с помощью которой можно исследовать темп роста функций многих переменных, анализировать степень влияния структуры входных данных параметризированного алгоритма на его вычислительную сложность, сравнивать между собой FPT-алгоритмы. Этой мерой является частная эластичность. Представленные результаты являются обобщением и расширением результатов автора, опубликованных в [5 – 7] применительно к одномерной теории сложности вычислений.

Под частной эластичностью $E_x(z)$ функции $z = z(x, y)$ по аргументу x понимается эластичность переменной z, которая рассматривается как функция только от x и при постоянных значениях y. Частная эластичность $E_x(z)$ связана с частной производной функции $z = z(x, y)$ по x соотношением $E_x(z) = z'_x \cdot x/z$. Аналогично $E_y(z) = z'_y \cdot y/z$.

В общем случае $E_x(z)$, $E_y(z)$ являются функциями, зависящими от двух аргументов x и y. Однако ситуация упрощается, если учесть тот факт, что для большинства параметризированных алгоритмов время выполнения алгоритма описывается функцией вида $z(x, y) = q(x)f(y)$, где $q(x)$ — количественная компонента, а $f(y)$ — параметрическая компонента этой функции. Для такой формы представления функции $z = z(x, y)$ частные эластичности $E_x(z)$, $E_y(z)$ вырождаются в обычные эластичности функции одного аргумента: $E_x(z) = E_x(q(x))$, $E_y(z) = E_y(f(y))$. Теперь $E_x(z)$ зависит лишь от x, а $E_y(z)$ — от y. Каждая из функций $q(x)$, $f(y)$ принадлежит одному сложностному классу множества $\mathfrak{K} = \{\text{SUBPOLY, POLY, SUBEXP, EXP, HYPEREXP}\}$ по подобающему аргументу [5]. Обозначим через $\mathfrak{F}_x$ класс сложности функции $z(x, y)$ по аргументу x, а через $\mathfrak{F}_y$ — класс сложности по аргументу y. Тогда всякому параметризированному алгоритму с функцией вычислительной сложности $z(x, y) = q(x)f(y)$ соответствует пара $(\mathfrak{F}_x, \mathfrak{F}_y) \in \mathfrak{K} \times \mathfrak{K}$, характеризующая сложность данного алгоритма как по длине входа x, так и значению параметра y. Таким образом, мы приходим к двумерной классификации параметризированных алгоритмов. При двумерном подходе более отчетливую и общую формулировку получает определение FPT-алгоритма: параметризированный алгоритм называется FPT-алгоритмом, если его вычислительная сложность задается функцией $z(x, y) = q(x)f(y)$, для которой $(\mathfrak{F}_x, \mathfrak{F}_y) \in \in \{\text{SUBPOLY, POLY}\} \times \mathfrak{K}$.

Введенная двумерная классификация FPT-алгоритмов не противоречит понятиям, используемым в настоящее время в параметризированной теории сложности применительно к FPT-алгоритмам, а лишь формально их уточняет и расширяет. Допустимо дальнейшее развитие предложенного подхода в части увеличения числа параметров и анализа других форм представления функций сложности.

Подробное изложение представленных результатов можно найти в [8].

ЛИТЕРАТУРА

1. *Downey R. and Fellows M.* Parameterized complexity. New York: Springer Verlag, 1999.
2. *Flum J. and Grohe M.* Parameterized complexity theory. Berlin; Heidelberg: Springer Verlag, 2006.
3. *Cesati M.* Compendium of parameterized problems. `http://bravo.ce.uniroma2.it/home/cesati/research/compendium` — 2006.
4. *Niedermeier R.* Invitation to fixed-parameter algorithms: Oxford Lecture series in mathematics and its applications. Oxford: University Press, 2006.
5. *Быкова В. В.* Метод распознавания классов алгоритмов на основе асимптотики эластичности функций сложности // Журн. СФУ. Математика и физика. 2009. № 2(1). С. 48–61.

6. *Быкова В. В.* Эластичность алгоритмов // Прикладная дискретная математика. 2010. № 2(8). С. 87–95.
7. *Bykova V. V.* Complexity and elasticity of the computation // Proc. of the 3-rd IASTED International Multi-Conference on Automation, Control, and Information Technology (ACIT-CDA 2010). Anaheim-Calgary-Zurich: ACTA Press, 2010. P. 334–340.
8. *Быкова В. В.* FPT-алгоритмы и их классификация на основе эластичности // Прикладная дискретная математика. 2011. № 2. С. 40–48.

УДК 519.682

О СВОЙСТВЕ ФОРМАЛЬНЫХ ЯЗЫКОВ НЕПОСРЕДСТВЕННО СОСТАВЛЯЮЩИХ

К. В. Сафонов, Д. А. Калугин-Балашов

В теории формальных грамматик словарь языка $X = \{x_1, \ldots, x_n\}$ обычно называют терминальным множеством, тогда как нетерминальным называют конечное множество $Z = \{z_1, \ldots, z_m\}$ вспомогательных символов, необходимых для задания грамматических правил (грамматики) данного языка. Для элементов этих множеств определены операции конкатенации и формальной суммы, приводящие к мономам и многочленам [1].

Если мономы и многочлены построены в соответствии с грамматическими правилами данного языка, то они интерпретируются как предложения и совокупности предложений. Рассмотрение совокупности всех грамматически правильных предложений приводит к необходимости изучать формальные степенные ряды от терминальных символов.

Грамматики непосредственно составляющих (нс-грамматики) являются важным подклассом контекстно-зависимых грамматик (кз-грамматик); в то же время контекстно-свободные грамматики (кс-грамматики) являются частным случаем нс-грамматик.

Нс-грамматике сопоставима система символьных уравнений

$$\alpha_j^{k_j} z_j \beta_j^{k_j} = p_j^{k_j}(x, z), \quad j = 1, \ldots, m, \quad k_j = 1, \ldots, L_j. \tag{1}$$

Ее решением является совокупность $(\alpha_1^1 z_1^1(x)\beta_1^1, \ldots, \alpha_m^{k_m} z_m^{k_m}(x)\beta_m^{k_m})$ формальных степенных рядов от терминальных переменных $x = (x_1, \ldots, x_n)$, а ряды $\alpha_1^{k_j} z_1^{k_j}(x)\beta_j^{k_j}$ являются нс-языком, определяемым данной грамматикой. Мономы $\alpha_j^{k_j}$ и $\beta_j^{k_j}$ называются контекстом. Рассмотрим *вполне определенные* нс-грамматики, которые сопоставимы системам уравнений вида

$$\alpha_j z_j \beta_j = p_j(x, z), \quad j = 1, \ldots, m. \tag{1'}$$

Эффективным инструментом изучения решений таких систем уравнений является *коммутативный образ* этой системы — новая система уравнений, переменные в которой рассматриваются как коммутативные, например из поля комплексных чисел. Понятно, что формальные степенные ряды $(z_1(x), \ldots, z_m(x))$, являющиеся решениями исходной системы (1′), переходят в степенные ряды, представляющие алгебраические функции. Коммутативный образ некоторого многочлена или ряда r будем обозначать как ci(r) (*commutative image*) [2].

Произведем ряд замен вида $z'_j = \alpha_j z_j \beta_j$. Получим систему уравнений

$$z'_j = p_j(x, z), \quad j = 1, \ldots, m. \tag{2}$$

Запишем систему уравнений (2) в виде

$$q_j(x, z, z') = 0, \quad j = 1, \ldots, m. \tag{3}$$

Решением этой системы назовем выражение символов z'_i в виде формальных степенных рядов от x, подстановка которых в многочлены $q_i(x, z, z')$ обращает их в нуль. Определим условия, при которых система (2) имеет такое решение.

Теорема 1. Если выполняется неравенство

$$J = \det \left(\frac{\partial(\mathrm{ci}(q_i(x, z, z')))}{\partial z'_j} \right)_{(x,z,z')=(0,0,0)} \neq 0,$$

то исходная система имеет решение в виде формальных степенных рядов.

Легко показать, что элементы главной диагонали матрицы Якоби содержат сумму мономов терминального алфавита (обусловленную линейной зависимостью некоторых z'_j от соответствующих z_j) и некоторого скаляра. Таким образом, в силу невырожденности матрицы Якоби в начале координат можно сделать линейную замену переменных z'_i, в результате которой эта матрица становится единичной, что, учитывая ряд проведенных замен, приводит систему (3) к виду

$$z'_j = p'_j(x, z'), \quad j = 1, \ldots, m,$$

в результате чего её можно решать методом последовательных приближений. Искомые ряды равны линейной комбинации получаемых рядов.

Ранее подобное свойство было доказано для контекстно-свободных грамматик [3].

ЛИТЕРАТУРА

1. *Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л.* Алгебра, языки, программирование. Киев: Наукова думка, 1974. 328 с.
2. *Сафонов К. В., Егорушкин О. И.* О синтаксическом анализе и проблеме В. М. Глушкова распознавания контекстно-свободных языков Хомского // Вестник Томского госуниверситета. Приложение. 2006. № 17. С. 63–66.
3. *Сафонов К. В., Калугин-Балашов Д. А.* О представлении контекстно-свободных языков диагоналями линейных языков // Прикладная дискретная математика. Приложение. 2010. № 3. С. 82–83.

УДК 004.423.43

ДЕНОТАЦИОННОЕ ОПИСАНИЕ ЯЗЫКА ASPECTTALK[1]

Д. А. Стефанцов, А. Е. Крюкова

Язык аспектно-ориентированного программирования (АОП) AspectTalk [1] разработан с целью создания защищённых систем обработки информации. На нём могут быть реализованы информационная система и политика её безопаности, а также осуществлена их интеграция с помощью соединительных модулей [2]. Одной из задач в определении языка является задание его семантики. В [3] получено денотационное описание семантики (ДОС) [4] объектно-ориентированного подмножества языка AspectTalk для доказательства семантической эквивалентности последнего языку

[1]Работа выполнена в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг. (гос. контракт № П1010).

Smalltalk. В данной работе приводится ДОС языка АОП AspectTalk, которое заключается в задании тройки объектов (L, S, M).

Для каждого нетерминала X из грамматики G языка AspectTalk определяется язык L_X, грамматика которого получается из G заменой аксиомы на X. Множество всех языков L_X обозначается L и называется множеством *синтаксических областей*. Примерами синтаксических областей являются множество записей примитивных операций языка AspectTalk и сам язык AspectTalk.

Элементы множества S являются *доменами* — множествами с завершённым частичным порядком. Сумма доменов, декартово произведение доменов, а также множество отображений из домена в домен являются доменами; порядок на последних определяется с помощью порядков на первых. Допускаются рекурсивные определения доменов. Подробнее о доменах можно прочитать в [5]. Домены ДОС языка AspectTalk подбирались для отражения сущностей языка и включают, например, домен процедур — домен функций из домена состояний в домен состояний — и домен программ — домен функций из домена входных последовательностей в домен выходных последовательностей.

Множество функций M, отображающих из элементов L в элементы S, называется множеством *семантических отображений* и, фактически, задаёт интерпретацию языка: множество M включает функцию, сопоставляющую программам на AspectTalk элементы функционального домена.

ЛИТЕРАТУРА

1. *Стефанцов Д. А.* Реализация политик безопасности в компьютерных системах с помощью аспектно-ориентированного программирования // Прикладная дискретная математика. № 1(1). 2008. С. 94–100.
2. *Стефанцов Д. А.* Технология и инструментальная среда создания защищённых систем обработки информации // Прикладная дискретная математика. Приложение № 1. 2009. С. 55–56.
3. *Стефанцов Д. А., Крюкова А. Е.* Формальное доказательство семантической эквивалентности ядра языка АОП AspectTalk и языка ООП Smalltalk // Прикладная дискретная математика. Приложение № 3. 2010. С. 84–85.
4. *Tennent R. D.* Denotational semantics // Handbook of logic in computer science. Oxford, UK: Oxford University Press, 1994. V. 3. P. 169–322.
5. *Scott D. S.* Data types as lattices // Lecture Notes in Mathematics. 1975. V. 499. P. 579–651.

УДК 004.428

РАЗРАБОТКА И РЕАЛИЗАЦИЯ БИБЛИОТЕКИ ORM НА ЯЗЫКЕ C++[1]

Д. А. Стефанцов, Н. О. Ткаченко, Д. В. Чернов, Р. В. Шмакова

Распространённым способом хранения информации в компьютерных системах является использование реляционных баз данных (БД), при котором информация о сущностях и связях предметной области представляется в виде записей в таблицах. Каждому из полей таблицы соответствует имя и тип хранимых данных. Можно выделить два класса библиотек взаимодействия с системами управления БД (СУБД) для языков программирования (ЯП). Библиотеки первого класса устанавливают соответствие

[1]Работа выполнена в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг. (гос. контракт № П1010).

между данными, хранимыми в полях записей таблиц, и данными элементарных типов в соответствующем ЯП. Библиотеки второго класса характерны для объектно-ориентированных ЯП и устанавливают соответствие между записями БД и объектами в ЯП; при этом поля записей соответствуют член-данным объектов. Эти библиотеки, называемые также объектно-реляционными отображениями (или ORM — Object-Relational Mapping) [1], реализуются с помощью библиотек первого класса и удобнее в использовании, поскольку позволяют абстрагироваться от способа представления информации в БД и использовать только понятия соответствующего ЯП. Создана библиотека ORM для языка C++ [2], рабочее название которой — «C++ ORM on Templates» (COT).

Основной сложностью в реализации ORM является избавление пользователя от необходимости поддержания большого количества служебной информации, нужной для взаимодействия с СУБД. Во-первых, необходимо установить соответствие между таблицами БД и классами объектов в ЯП. При этом от пользователя ORM требуется задание имени класса, а также имен и типов член-данных его экземпляров. Запросы на вставку, удаление и изменение соответствующих записей в таблице БД генерируются ORM. Во-вторых, необходимо задавать запросы на выборку данных из таблиц БД, используя только член-функции классов (называемые в языке C++ *статическими*) и объектов и не прибегая к использованию языка запросов SQL [3]. Это означает необходимость генерации член-функций запросов на выборку данных, не существующих в момент реализации ORM. Традиционно оба перечисленных требования к ORM реализуются с помощью *рефлексии* [4] — способа метапрограммирования, заключающегося в доступе программы к информации о собственной структуре и возможности изменять эту структуру. Под *структурой программы* подразумевается совокупность типов данных и алгоритмов, используемых в ней.

Язык C++ не обладает встроенной возможностью рефлексии, поэтому реализациям ORM необходимо поддерживать метаданные (данные о данных) самостоятельно. Например, библиотека ODB [5] подразумевает использование директив типа `#pragma` в описании классов, соответствующих таблицам БД, и обработку этих описаний внешним транслятором, расширяющим структуру программы. Очевидны недостатки такого способа реализации ORM: наличие дополнительного транслятора, что затрудняет использование библиотеки на всех программно-аппаратных платформах, имеющих трансляторы с C++, а также невозможность автоматической проверки корректности программы до этапа трансляции. Другая библиотека, Wt::Dbo [6], требует поддержания метаданных от своих пользователей; кроме того, в член-функциях, осуществляющих запросы на выборку данных, необходимо использование частей строк запроса `SELECT` на языке SQL, что, во-первых, делает необходимым знание SQL пользователями Wt::Dbo и, во-вторых, может привести к появлению в программе SQL-инъекций [7].

В данной работе используется метапрограммирование на шаблонах в C++, позволяющее расширять структуру программы на этапе её трансляции. Современные трансляторы имеют необходимые алгоритмы оптимизации программ, что делает использование метапрограммирования на шаблонах возможным не только в качестве эксперимента. Популярные библиотеки Boost [8] являются примером использования этого способа написания программ на C++.

В языке C++ возможно рекурсивное описание типов с помощью конструкций `typedef` и `typename`. Библиотека COT использует для хранения информации о типах параметров и результатах выполнения запросов односвязные списки классов, построенные этим способом.

Для описания *моделей* — классов в ЯП, соответствующих таблицам в БД, — в COT используются макросы `BEGIN_MODEL`, `END_MODEL` и `FIELD` (см. строки 1–4 листинга 1). Данные макросы заменяются препроцессором на описание соответствующего класса, которое включает определения: 1) член-данных его экземляров; 2) классов, содержащих информацию об этих член-данных и их связях с полями таблицы в БД; 3) статических член-функций, выполняющих запросы к СУБД, и другую служебную информацию.

Запрос на выборку данных приведён в строках 6–8 листинга 1 — это статическая член-функция `filter` класса модели, принимающая переменное количество параметров. Шаблонные параметры заключены в угловые скобки и представляют собой связанные булевыми функциями условия выборки — шаблонные классы сравнения `Lt`, `Gt` или `Eq`, осуществляющие проверку отношения «меньше», «больше» или «равно» соответственно. Параметром шаблонных классов сравнения является имя поля модели; значения, с которыми сравниваются эти поля, передаются в член-функцию `filter` в качестве параметров в круглых скобках (при этом первым должно быть указано количество сравнений).

```
1 BEGIN_MODEL(Author)
2     FIELD(name, StringValue<256>)
3     FIELD(age, IntValue)
4 END_MODEL
5
6 authors = Author::filter<
7         And< Lt<Author::_age_>, Eq<Author::_name_> >
8     >::with(2, 40, "John Smith");
```

Листинг 1. Пример описания модели и запроса на выборку данных в COT

Предоставляемые библиотекой COT средства разрабатывались по аналогии с ORM из библиотеки Django [9] для ЯП Python [10] — одной из самых известных реализаций ORM. Описание модели и запроса на выборку данных, реализованные с помощью Django ORM, аналогичные представленным на листинге 1, можно найти в строках 1–3 и 5–6 листинга 2 соответственно.

```
1 class Author(models.Model):
2     name = models.CharField(max_length=256)
3     age = models.IntegerField()
4
5 authors = \
6     Author.objects.filter(age__lt=40, name="John Smith")
```

Листинг 2. Пример описания модели и запроса на выборку данных в Django ORM

Заметим, что в Django ORM условия выборки объединяются логической связкой «и», при этом использование связок «или» и «не» затруднено. В COT шаблонные классы `And`, `Or` и `Not` равноправны.

Для связи с СУБД использован метод подготовленных запросов, позволяющий единожды произвести синтаксический разбор строк на языке SQL и впоследствии только подставлять параметры при выполнении запроса. Это делает невозможным изменение синтаксической структуры запроса после его подготовки, что исключает SQL-инъекции. Для передачи результатов выполнения запросов из процедур использованы

указатели из библиотеки Boost, отслеживающие количество ссылок на объекты для автоматического управления памятью.

Работа библиотеки изучалась с помощью инструментов GNU gprof [11] и Valgrind [12]. Результаты анализа показали, что библиотека COT в среднем не уступает в производительности библиотеке ODB. При реализации судейской системы для проведения игр по защите информации CTF [13] COT превысила по быстродействию библиотеку ODB не менее чем на 9 %. Использование инструмента Valgrind позволило избавиться от утечек памяти.

ЛИТЕРАТУРА

1. *Ambler S. W.* Mapping Objects to Relational Databases: O/R Mapping In Detail / Ambysoft Inc. 2010. `http://www.agiledata.org/essays/mappingObjects.html`
2. *Страуструп Б.* Дизайн и эволюция языка C++. Объектно-ориентированный язык программирования. М.: ДМК Пресс, 2000. 448 с.
3. *Musteata B. and Lesser R.* Standard SQL Relational Database Language Guide and Reference. TLM, Inc., 1988. 275 p.
4. *Sobel J. M. and Friedman D. P* An Introduction to Reflection-Oriented Programming / Computer Science Department, Indiana University, USA. 1996. 20 p. `http://www.cs.indiana.edu/hyplan/jsobel/rop.ps.gz`
5. ODB: C++ Object-Relational Mapping (ORM) / Code Synthesis Tools CC. 2011. `http://www.codesynthesis.com/products/odb/`
6. *Deforche K.* Wt::Dbo Tutorial. 2010. `http://www.webtoolkit.eu/wt/doc/tutorial/dbo/tutorial.html`
7. SQL Injection / OWASP — The Open Web Application Security Project. 2011. `https://www.owasp.org/index.php/SQL_Injection`
8. *Dawes B., Abrahams D., and Rivera R.* Boost C++ Libraries. 2011. `http://www.boost.org/`
9. Django / Django Software Foundation. 2011. `http://www.djangoproject.com/`
10. The Python Tutorial / Python Software Foundation. 2011. `http://docs.python.org/tutorial/`
11. GNU gprof / Free Software Foundation Inc. 2009. `http://sourceware.org/binutils/docs/gprof/index.html`
12. Valgrind / Valgrind^{TM} Developers. 2011. `http://valgrind.org/`
13. *Ткаченко Н. О., Чернов Д. В.* Разработка и реализация сервера игры CTF // Прикладная дискретная математика. Приложение. 2010. № 3. С. 62–64.