

где $\gamma_z = \{|z_1| = \dots = |z_n| = \varepsilon\}$ и $\Gamma_x = \{|x_1| = \dots = |x_m| = \delta\}$ — циклы интегрирования; $0 < \delta \ll \varepsilon \ll 1$; $dz = dz_1 \wedge \dots \wedge dz_n$; $dx = dx_1 \wedge \dots \wedge dx_m$; $x^{\alpha+I} = x_1^{\alpha_1+1} \cdot \dots \cdot x_m^{\alpha_m+1}$; $\alpha! = \alpha_1! \cdot \dots \cdot \alpha_m!$; $(z - ci(Q^*(z, x, t))) = (z_1 - ci(Q_1^*(z, x, t))) \cdot \dots \cdot (z_n - ci(Q_n^*(z, x, t)))$; $\frac{\partial^{|\alpha|}}{\partial x^\alpha} = \frac{\partial^{\alpha_1+\dots+\alpha_m}}{\partial x_1^{\alpha_1} \cdot \dots \cdot \partial x_m^{\alpha_m}}$; δ_{ij} — дельта Кронекера.

Формулы (4) и (5) позволяют эффективно осуществлять синтаксический анализ монома, находя его синтаксический полином. Так, кратный интеграл (4) можно вычислить как повторный, используя формулу Коши и разложение в степенной ряд подынтегральной функции.

ЛИТЕРАТУРА

1. Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л. Алгебра. Языки. Программирование. Киев: Наукова думка, 1973.
2. Salomaa A. and Soittola M. Automata-Theoretic Aspects of Formal Power Series. N.Y.: Springer Verlag, 1978.
3. Сафонов К. В., Егорушкин О. И. О синтаксическом анализе и проблеме В. М. Глушкова распознавания контекстно-свободных языков Хомского // Вестник Томского государственного университета. 2006. Приложение № 17. С. 63–67.
4. Семёнов А. Л. Алгоритмические проблемы для степенных рядов и контекстно-свободных грамматик // Доклады АН СССР. 1973. № 212. С. 50–52.
5. Safonov K. V. On power series of algebraic and rational functions in C^n // J. Math. Analysis Appl. 2000. V. 243. P. 261–277.
6. Сафонов К. В. Об условиях алгебраичности и рациональности суммы степенного ряда // Матем. заметки. 1987. Т. 41. Вып. 3. С. 325–332.
7. Егорушкин О. И., Колбасина И. В., Сафонов К. В. О совместности систем символьных полиномиальных уравнений и их приложении // Прикладная дискретная математика. Приложение. 2016. № 9. С. 119–121.
8. Egorushkin O. I., Kolbasina I. V., and Safonov K. V. On solvability of systems of symbolic polynomial equations // Журн. СФУ. Сер. Матем. и физ. 2016. Т. 9. Вып. 2. С. 166–172.

УДК 004.4

DOI 10.17223/2226308X/11/40

РАБОТА СО СТЕКОМ В ЛЯПАСЕ¹

М. С. Недяк, В. О. Сафонов

Описана работа со стеком в модульном трансляторе с ЛЯПАСа, которая включает в себя механизмы вызова функции, работу с локальными переменными и параметрами функции. Рассматриваются механизмы обработки композиции функций.

Ключевые слова: ЛЯПАС, язык программирования, транслятор, соглашение о вызове.

Введение

Рассмотрим модуль для работы со стеком модульного транслятора [1, 2] с ЛЯПАСа [3]. Далее этот модуль и его язык называются стекоязыком.

В силу ограниченного количества переменных в языке, в текущей версии транслятора соглашение о вызовах выглядит следующим образом: перед вызовом функции

¹Работа поддержана грантом РФФИ, проект № 17-01-00354.

на стеке резервируется память под все переменные независимо от того, используются они в подпрограмме или нет. Такой метод использования памяти стека неэффективен.

В работе описаны новые механизмы вызова функции, работы с её локальными переменными и параметрами, предложен механизм обработки композиции функций.

1. Операции со стеком

Модуль транслирует операции входной программы предыдущего промежуточного языка (комплексояза) [4] в набор операций стекоязыка. Стекоязык состоит из операций комплексоязыка и содержит новые вспомогательные операции:

- 1) `stack_alloc N` — резервирует в стеке $8N$ байт;
- 2) `stack_free N` — освобождает в стеке $8N$ байт;
- 3) `call <function>` — помещает в вершину стека адрес возврата и передаёт управление функции *function*;
- 4) `ret` — передаёт управление по адресу возврата, который извлекается из вершины стека;
- 5) `push <variable>` — помещает в вершину стека значение *variable*;
- 6) `pop <variable>` — извлекает значение из вершины стека и помещает его в *variable*.

2. Правила передачи аргументов в функции

Разработаны следующие правила передачи аргументов в функции:

- 1) входные аргументы помещаются в стек;
- 2) резервируется память в стеке под выходные аргументы;
- 3) вызывается функция;
- 4) возвращаемые значения записываются в соответствующие локальные переменные;
- 5) освобождается память, занятая входными аргументами.

Рассмотрим правила на примере. Дана операция вызова функции на языке комплексоязыка:

```
1 call func1 f, g, /, a, b, c
```

Результат трансляции:

```
1 push f
2 push g
3 stack_alloc 3
4 call func1
5 pop c
6 pop b
7 pop a
8 stack_free 2
```

3. Правила обработки композиции функций

Разработаны следующие правила обработки композиции функций:

- 1) для внутренней функции выполняются правила 1–3 передачи аргументов в функцию;
- 2) для каждой следующей функции:
 - а) резервируется память на стеке под выходные параметры;
 - б) вызывается функция;

- 3) возвращаемые значения записываются в соответствующие локальные переменные;
- 4) освобождается память стека от входных аргументов всех функций.

Вызовы функций композиции «накладываются» друг на друга. Рассмотрим правила на примере. Пусть дана операция композиции на языке комплексояз:

```
1 call_composition f3, f2, f1, /, a, /, d, c
```

Результат трансляции:

```
1 push a
2 stack_alloc 2
3 call f1
4 stack_alloc 1
5 call f2
6 stack_alloc 2
7 call f3
8 pop d
9 pop c
10 stack_free 4
```

4. Правила работы с локальными переменными и параметрами функции

Разработаны следующие правила работы с локальными переменными и параметрами:

- 1) резервируется место в стеке под локальные переменные;
- 2) каждой переменной ставится в соответствие новое название:
 - а) локальным переменным l_0, l_1, \dots, l_n ;
 - б) параметрам p_{n+2}, p_{n+3}, \dots ;
 номер нового названия равен сдвигу относительно указателя на вершину стека;
- 3) названия переменных заменяются на новые названия;
- 4) перед операцией `ret` зарезервированное место на стеке освобождается.

Рассмотрим данные правила на примере. Дана функция на комплексоязе:

```
1 definition function(k/i)
2 move acc, k
3 move i, acc
4 inc i
```

Результат трансляции:

```
1 function:
2 stack_alloc 1
3 move l0, p3
4 move p2, l0
5 inc p2
6 stack_free 1
7 ret
```

Заключение

Разработаны и реализованы в модульном трансляторе ЛЯПАС [1]:

- 1) правила обработки вызова функций;

- 2) правила обработки композиции функций;
- 3) правила работы с локальными переменными и параметрами функции.

ЛИТЕРАТУРА

1. Стефанцов Д. А., Сафонов В. О., Першин В. В. и др. Модульный транслятор с языка ЛЯ-ПАС // Прикладная дискретная математика. Приложение. 2016. № 8. С. 122–126.
2. <https://github.com/tsu-iscd/lyapas-1cc> — LYaPAS Compiler Chain. 2018.
3. Агibalов Г. П., Липский В. Б., Панкратова И. А. О криптографическом расширении и его реализации для русского языка программирования // Прикладная дискретная математика. 2013. № 3. С. 93–104.
4. <https://github.com/tsu-iscd/lyapas-1cc/blob/73b21bcd5f674bc6762a379bc32f71f61ee51164/doc/cyaz.md> — LYaPAS Cyaz Documentation. 2018.

УДК 510.52

DOI 10.17223/2226308X/11/41

О ГЕНЕРИЧЕСКОЙ СЛОЖНОСТИ ПРОБЛЕМЫ ДИСКРЕТНОГО ЛОГАРИФМА В ГРУППАХ ТОЧЕК ЭЛЛИПТИЧЕСКИХ КРИВЫХ НАД КОНЕЧНЫМИ ПОЛЯМИ¹

А. Н. Рыбалов

Изучается генерическая сложность проблемы дискретного логарифма в группах точек эллиптических кривых над $\text{GF}(p)$, где p — простое. Доказывается, что её естественная подпроблема генерически трудноразрешима (то есть трудна для почти всех входов) при условии, что проблема дискретного логарифма для эллиптических кривых трудноразрешима в классическом смысле.

Ключевые слова: генерическая сложность, дискретный логарифм, эллиптическая кривая.

Введение

Эллиптическая криптография занимается разработкой криптосистем с открытым ключом, основанных на эллиптических кривых над конечными полями. В качестве базиса для этих криптосистем используется проблема дискретного логарифма в группах точек эллиптических кривых над конечными полями. Основное преимущество эллиптической криптографии заключается в том, что на сегодняшний день не известно даже субэкспоненциальных алгоритмов решения проблемы дискретного логарифма на эллиптических кривых, в отличие от проблемы дискретного логарифма в конечных полях. Рассмотрим проблему дискретного логарифма в группах точек эллиптических кривых над конечными полями $\text{GF}(p)$, где p — простое. Эллиптические кривые над такими полями используются в протоколах электронной цифровой подписи ECDSA и ГОСТ Р 34.10-2012.

В [1] развита теория генерической сложности вычислений. В рамках этого подхода алгоритмическая проблема рассматривается не на всём множестве входов, а на некотором подмножестве «почти всех» входов. Такие входы образуют так называемое генерическое множество. Понятие «почти все» формализуется введением естественной меры на множестве входных данных. С точки зрения современной криптографии интересны такие алгоритмические проблемы, которые, являясь (гипотетически) трудными в классическом смысле, остаются трудными и в генерическом смысле, т. е. для почти

¹Работа поддержана грантом РФФИ, проект № 18-41-550001.