службы BOINC. Основу расчётной части составляют SAT-решатели minisat-1.14.1 и minisat-2.0, модифицированные с учётом особенностей КНФ, кодирующих задачи обращения дискретных функций [4]. Исполняемые файлы расчётной части взаимодействуют с BOINC-клиентом при помощи функций библиотеки DC-API, что позволяет, в частности, реализовать периодическое сохранение промежуточных данных вычислений в виде контрольных точек. На момент написания данной работы в проекте SAT@home принимает участие более тысячи активных добровольцев и задействовано более двух тысяч активно работающих ПК. Средняя производительность проекта за весь период работы составляет 1,7 терафлопс.

В декабре 2011 г. в SAT@home был запущен эксперимент по решению задачи криптоанализа генератора ключевого потока A5/1. На сегодняшний день наиболее эффективным методом криптонализа данного генератора является «rainbow-метод» [5]. Однако доступные rainbow-таблицы [5] покрывают ключевое пространство примерно на 88 % (при «реалистичных» ограничениях на условия криптоанализа). Нами были сгенерированы 10 тестов, не поддающихся криптоанализу с использованием этих rainbow-таблиц. На момент написания данной работы в проекте SAT@home успешно решены 9 из них (в среднем на решение каждого теста уходило 2 недели работы проекта).

ЛИТЕРАТУРА

1. Платформа BOINC для организации добровольных вычислений. `http://boinc.berkeley.edu/`
2. Проект добровольных вычислений SAT@home. `http://sat.isa.ru/pdsat/`
3. *Balaton Z., Gombas G., Kacsuk P., et al.* Sztaki desktop grid: a modular and scalable way of building large computing grids // 21th Intern. Parallel and Distributed Processing Symposium. Long Beach, California, USA, 2007. P. 1–8.
4. *Semenov A., Zaikin O., Bespalov D., and Posypkin M.* Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system // LNCS. 2011. V. 6873. P. 473–483.
5. A5/1 Cracking project. `http://reflextor.com/trac/a51/wiki`

УДК 004.431:004.432:004.436

АНАЛИЗ ОБЪЕКТНЫХ ФАЙЛОВ DELPHI С ИСПОЛЬЗОВАНИЕМ СПЕЦИФИКАЦИИ СЕМАНТИКИ МАШИННЫХ КОМАНД

А. А. Михайлов

Процесс декомпиляции является важной, а при решении некоторых задач (таких, как поддержка программного обеспечения без возможности использования исходного кода или восстановление исходного кода) и неотъемлемой частью разработки программного обеспечения. В общем случае (для произвольных исполняемых файлов) эта задача является очень сложной, например требуется разделить память программы на код и данные. В объектных файлах Delphi программа оказывается более структурированной, например выделены блоки памяти, соответствующие коду каждой процедуры; имеется информация о типах данных; может присутствовать отладочная информация. Таким образом, при работе с файлами `dcu` [1] задача декомпиляции становится более достижимой.

В общем виде формат файла `dcu` выглядит следующим образом: сначала идёт заголовок, в котором содержится общая информация о файле, такая, как размер, время

компиляции и т. д. После заголовка следует поток теговой информации. Основную массу тегов можно разделить на следующие группы:

1) описания включаемых модулей и объектных файлов;
2) описания импортируемых из этих модулей определений (типов данных, процедур и т. д.);
3) описания определений (типов данных, процедур, функций и т. д.) из данного модуля;
4) блок памяти, составленный из блоков кода для процедур и функций, образов констант и т. д.;
5) информация для редактора связей (в какие места блока памяти необходимо занести адреса, получаемые из других модулей);
6) отладочная информация.

В программе DCU32INT [2] используется статический дизассемблер, который, в частности, не может определить имя виртуального метода по коду его вызова: для этого требуется проследить последовательность присвоений регистров и ячеек памяти и по смещению в таблице виртуальных методов извлечь имя вызываемого метода. Для решения подобных задач требуется использовать информацию о семантике машинных команд. Основным результатом данной работы является создание структур данных, предназначенных для описания семантики машинных команд процессоров семейства Intel x86 [3] и механизмов их использования в сочетании с дизассемблером DCU32INT. Предложенные структуры данных позволяют точно описать семантику наиболее важных для рассматриваемой задачи инструкций. При разборе инструкции используется результат её анализа существующим статическим дизассемблером, позволяющий определить выполняемую операцию и её аргументы. Семантика выполняемых операций задается при помощи специализированных операторов и выражений, предназначенных для решения этой задачи. Для грубого определения семантики остальных (не описанных явно) команд используются таблицы «опкодов» [4], в которых содержится информация об операндах, мнемоника инструкции, изменяемых и тестируемых флагах и т. д. Хотя грубые описания семантики не позволяют определить точные выражения для вычисляемых операцией значений, они могут дать информацию о времени жизни в ячейках памяти. В результате при анализе кода подпрограмм в `dcu` появляется возможность анализа потоков данных, порождаемых наиболее важными конструкциями языка Delphi, например вызовом виртуального метода.

Таким образом, разработан механизм описания спецификаций семантики машинных команд. С его помощью решаются задачи извлечения имён вызываемых виртуальных методов, распространения констант, подстановки имён используемых переменных и т. д., которые позволяют существенно сократить временные ресурсы, требуемые на получение описания алгоритма исследуемой программы. Конечной целью исследования является разработка методов анализа и программы декомпилятора объектных файлов Delphi на основе программы DCU32INT в язык высокого уровня, максимально соответствующий исходному коду исследуемой программы.

ЛИТЕРАТУРА

1. *Хмельнов А. Е.* Язык FlexT для спецификации бинарных форматов данных: дис. ... канд. техн. наук. Иркутск, 2000. 118 с.
2. `http://hmelnov.icc.ru/DCU/index.ru.html` — Инструмент DCU2INT (для разбора юнитов Delphi).

3. http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html — Intel Architecture Software Developer's Manual.
4. http://ref.x86asm.net/geek32.html — X86 Opcode and Instruction Reference.

УДК 004.4'41

АНАЛИЗ НЕКОТОРЫХ СЕМАНТИЧЕСКИХ АСПЕКТОВ ИСХОДНЫХ ТЕКСТОВ ПРОГРАММ НА ОСНОВЕ ФОРМАЛЬНЫХ СПЕЦИФИКАЦИЙ СИНТАКСИСА И СЕМАНТИКИ

А.М. Саух

Целью работы является создание инструментального средства, позволяющего осуществлять семантический анализ исходных текстов программ, написанных на различных языках программирования, на основе формальных спецификаций лексики, синтаксиса и семантики целевого языка программирования. Рассматривается задача анализа некоторых семантических аспектов исходных текстов однопоточных программ на императивных языках программирования, таких, как объявление и использование идентификаторов, а также выявления областей их использования на основе анализа потоков исполнения программы. Целью такого анализа является извлечение семантически значимых конструкций и алгоритмов работы из текста программы и представление их в независимом от конкретного языка виде.

Анализ проводится в несколько этапов. Сначала производится лексический и синтаксический анализ, сопровождающийся построением абстрактного синтаксического дерева по исходному тексту программы. В зависимости от конкретного языка построение таблицы идентификаторов и дерева пространств имён происходит либо в один этап — сразу при построении синтаксического дерева, либо в два этапа, первый из которых осуществляется при построении дерева, а второй — при вторичном обходе построенного дерева.

В качестве отправной точки для решения задачи построения синтаксического дерева используется система генераторов компиляторов Lex и Yacc. Генераторы модифицированы так, что вместо исходного кода анализирующих конечных автоматов, который необходимо сначала скомпилировать, сразу создаются готовые к работе структуры анализатора. В качестве формата файлов спецификаций используются оригинальные форматы файлов спецификаций лексики и синтаксиса Lex и Yacc, модифицированные в части описания действий, исполняемых при свёртке по синтаксическим правилам. Лексические правила описываются в виде набора регулярных выражений, которым сопоставлены управляющие конструкции, регулирующие возврат обнаруженных лексем в синтаксический анализатор. Синтаксические правила представляют собой описания структур языка, которым сопоставлены управляющие конструкции, регулирующие построение абстрактного синтаксического дерева, таблиц идентификаторов, дерева пространств имён и других элементов (такой подход описан в [1]). Синтаксическим анализатором поддерживаются контекстно-свободные грамматики, разбираемые по алгоритму LALR.

Семантический анализ производится на основе нескольких обходов абстрактного синтаксического дерева с построением конструкций, описывающих объявление и использование идентификаторов (типов, переменных, функций, классов, методов и т. д.), организацию пространств имён и ограниченно описывающих потоки исполнения команд, что позволяет анализировать использование тех или иных участков кода.