Декомпиляция объектных файлов *.dcuil

А.А. Михайлов <mikhailov@icc.ru>
А.Е. Хмельнов <alex@icc.ru>
Институт динамики систем и теории управления
имени В.М. Матросова СО РАН,
664033, Россия, Иркутск, ул. Лермонтова, 134

Аннотация. Работа посвящена решению задачи декомпиляции одного из разновидностей формата DCU – файлов .dcuil, создаваемых компиляторами тех версий Delphi, которые работали для платформы .NET. Разработан метод решения этой задачи, состоящий из ряда этапов: синтаксический анализ кода CIL; формирование графа потока управления; генерация промежуточного представления; структурирование графа потоков управления; анализ потоков данных с учётом семантики команд CIL; улучшение промежуточного представления с учётом особенностей работы компилятора Delphi; генерация кода.

Ключевые слова: обратная инженерия; объектный код; Delphi.

DOI: 10.15514/ISPRAS-2017-29(6)-5

Для цитирования: Михайлов А.А., Хмельнов А.Е. Декомпиляция объектных файлов DCUIL. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 105-116. DOI: 10.15514/ISPRAS-2017-29(6)-5

1. Введение

Задача декомпиляции программного кода до сих пор не решена в полном объёме. Хотя существуют примеры декомпиляторов, корректно восстанавливающих исходный код для некоторых типов файлов (в основном это декомпиляторы кода виртуальных машин), для исполняемых файлов, содержащих машинный код, возможности всех существующих реализаций декомпиляторов очень ограничены, что затрудняет использование результатов их работы на практике. Сложность разработки таких декомпиляторов объясняется тем, что для полноценной декомпиляции исполняемых файлов требуется решить такие задачи, как: разделение кода и данных, учёт семантики машинных команд, вывод типов, распознавание системных библиотек. При этом, например, задача разделения кода и данных в общем случае является алгоритмически неразрешимой [1].

Объектные файлы содержат информацию о программном коде и данных, необходимую для сборки исполняемого файла редактором связей. Эти файлы более структурированы, чем исполняемые: код и данные в значительно

большей степени разделены, сохранена информация об именах подпрограмм и глобальных переменных (как определяемых, так и используемых), при этом объектные файлы содержат такой же машинный код, что и соответствующие исполняемые файлы. Таким образом, задача декомпиляции для объектных файлов должна решаться проще. Однако задача декомпиляции объектных файлов обычно не рассматривается, поскольку такие файлы в основном воспринимаются программистами, как некоторый кэш компилятора – вспомогательные данные, формируемые компилятором в ходе работы и не представляющие самостоятельной ценности.

Исключение составляет формат DCU [2] объектных файлов Delphi. Помимо образов памяти подпрограмм и глобальных данных, в файлах .dcu кодируется вся информация из интерфейсной части модуля, то есть они совмещают функции .obj и .h файлов, поэтому очень часто использующие Delphi разработчики распространяют свои модули и целые библиотеки модулей в формате DCU, без предоставления исходных текстов. При этом формат DCU частично изменяется с каждой новой версией продукта, поэтому программисты, зависящие от чужих модулей, должны полагаться на то, что разработчик такого модуля не прекратит свою работу и скомпилирует его для следующих версий компилятора, когда это потребуется. Время показывает, что эта надежда очень часто не оправдывается. Таким образом, декомпиляция объектных файлов DCU является актуальной для тех разработчиков, которые используют объектные файлы DCU без исходных кодов.

Несмотря на то, что в последних версиях Delphi платформа .NET не поддерживается, разработка метода декомпиляции для одной из разновидностей формата DCU открывает дорогу к разработке аналогичных методов для других разновидностей этого формата. Кроме того, декомпиляция файлов DCUIL может быть непосредственно востребована при решении задач, связанных с поддержкой унаследованного программного кода, скомпилированного для этой платформы.

2. Общая схема декомпиляции объектных файлов DCUIL

Рисунок 1. Схема декомпиляции объектного кода Delphi
Figure 1. Decompilation scheme of the Delphi object code

Декомпиляция кода в файлах DCU может выполняться отдельно для каждой подпрограммы, что существенно облегчает решение этой задачи. Для реализации декомпилятора объектных файлов DCUIL необходимо решить следующие основные задачи, (рис. 1): восстановление высокоуровневых операторов, генерация промежуточного представления, генерация кода, и выполнение оптимизаций, нацеленных на улучшение результата декомпиляции.

2.1 Загрузчик файла

Загрузчик осуществляет разбор входного файла в формате DCUIL, DCU. В качестве загрузчика использована программа DCU32INT, которая выполняет разбор файла в соответствии со спецификацией формата DCU на языке FlexT [3]. Загрузчик считывает последовательность тегов в исходном файле и ставит им в соответствие структуры данных, описывающие прочитанные утверждения. В ходе чтения теговых структур данных для каждого файла DCU формируется две таблицы: таблица адресов и таблица типов. Большинство структур данных файла DCU ссылаются на другие структуры по индексам в этих таблицах.

Описания подпрограмм содержат информацию о смещении образа памяти с кодом подпрограммы в блоке памяти модуля и размере этого образа.

За блоком памяти следует запись с таблицей перемещаемых адресов (FixUp), которая содержит информацию о том, в какие места блока памяти должны быть подставлены адреса различных процедур, переменных, описаний типов данных и других определений после их назначения редактором связей. Эта информация используется дизассемблером при разборе байт-кода для контроля правильности работы и отображения информации. Так, перемещаемые адреса могут встречаться только в операндах инструкций и не могут пересекаться с кодами команд. При наличии перемещаемого адреса в операнде информация об этом адресе используется при выводе операнда.

2.2 Процедура дизассемблирования

В программе DCU32INT реализован примитивный статический дизассемблер, который умеет:

- определять размер, занимаемый одной машинной командой;
- определять, что команда безвозвратно передаёт управление;
- обнаруживать ссылки из машинной команды (переходы на другие команды);
- отображать машинные команды.

Таких возможностей недостаточно для реализации декомпилятора, которому необходимо иметь всю информацию о семантике команды, доступную виртуальной машине исполняющей её.

Для получения семантики инструкций был использован проект Mono, реализованный на языке C#. Для этого был модифицирован декомпилятор ILSpy таким образом, чтобы на выходе он производил код близкий языку Delphi.

В результате декомпиляции части библиотеки Mono были получены две таблицы опкодов CIL:

- OneByteOpCodes – опкоды длиной один байт;
- TwoByteOpCodes – опкоды длиной два байта.

Каждое значение в таблице представляет собой объект, который содержит в себе всю необходимую информацию о семантике опкода:

- имя опкода;
- информацию о типе передачи управления;
- тип самого опкода;
- типы операндов;
- информацию о состоянии стека до выполнения команды и после;

В декомпиляторе стадия дизассемблирования сводится к сопоставлению последовательности байтов, кодирующих машинную команду некоторого выражения, описывающего её семантику. Команда CIL представляет собой закодированное по определённым правилам [4] указание для виртуальной машины на выполнение некоторой операции. Команда всегда начинается с кода команды. Код команды может занимать от одного до двух байтов, у двухбайтовых кодов первый байт всегда буден равен `0xFE` (т.е. ряд команд закодирован в дополнительной таблице, т.к. они не поместились в основной).

2.3 Генерация промежуточного представления

Промежуточное представление CILIR [5] (CIL Intermediate Representation), разработанное авторами в декомпиляторе DCUIL2PAS [6] реализовано в виде иерархии классов (рис. 2), с использованием техники подсчёта ссылок. Счётчики ссылок используются для экономии памяти и в дальнейшем для вычисления результатов выражений.

Промежуточный код для базовых блоков строится путём символьной интерпретации каждой команды CIL и сопоставления ей выражения (экземпляра класса), реализующего семантику. Начальное состояние каждого линейного участка кода характеризуется значениями параметров подпрограммы и локальных переменных, а также состоянием стека. Конечное состояние определяется в результате символьной интерпретации.

Далее применяется итерационный алгоритм анализа потоков данных для достигающих определений. Состояние локальных переменных и аргументов подпрограммы являются общим контекстом для всех базовых блоков, и используется только для хранения результатов, а не вычисления выражений.

Поэтому в качестве входного и выходного множества для передаточной функции рассматривается только состояние стека.

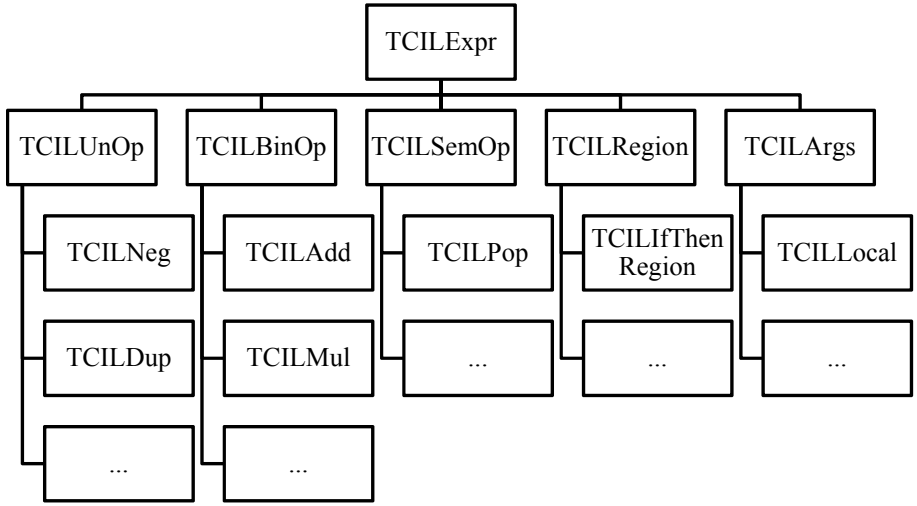

Рисунок 2. Иерархия классов промежуточного представления
Figure 2. Intermediate Representation Class Hierarchy

2.4 Упрощение сокращенных логических выражений

Для логических выражений, включающих логические связки `and` и `or`, компилятор может генерировать код для операторов условного перехода двумя разными способами:

- Полное вычисление выражений. Условие преобразуется в последовательность инструкций вычисления выражения, помещающих вычисленный результат на стек для последующего извлечения в качестве аргумента для опкода условного перехода.
- Сокращённое вычисление логических выражений (short-circuit boolean evaluation) с помощью условных выражений, основанных на следующих правилах:
    1. $A\ and\ B \Rightarrow if\ A\ then\ B\ else\ False$,
    2. $A\ or\ B \Rightarrow if\ A\ then\ True\ else\ B$.

Этот режим используется компилятором по умолчанию и позволяет не выполнять часть вычислений, если становится известно, что они уже не могут повлиять на результирующее значение выражения.

Случай полного вычисления логических выражений является наиболее простым для декомпилятора, поскольку результатом вычисления выражения,

извлеченного со стека в качестве операнда команды условного перехода, будет исходное логическое условие.

При декомпиляции логического выражения, вычисляемого сокращённым способом, на этапе улучшения промежуточного кода выполняется объединение логических условий по заранее определённому набору правил:

1. $if\ A\ then\ B\ else\ False \Rightarrow A\ and\ B$,
2. $if\ A\ then\ True\ else\ B \Rightarrow A\ or\ B$.

2.5 Генерация управляющего графа

Одной из наиболее важных задач декомпиляции является восстановление высокоуровневых операторов, таких как `if-then-else`, `if-then`, `while`, `for`, `case` и т.д.

Алгоритм для построения базовых блоков, представленный в [8], получает на вход последовательность трёхадресных команд. Далее в последовательности команд выделяются лидеры, которые разбивают её на линейные подпрограммы, которые начинаются с команды-лидера.

Для разбиения дизассемблируемой программы на базовые блоки используется информация о типе передачи управления, которая содержится в описывающей инструкцию структуре данных.

Генерация управляющего графа совершается в один проход и объединена с процессом дизассемблирования. В качестве параметра в функцию разбора передаётся указатель на процедуру, которая производит разбиение блока памяти на последовательности инструкций, которые соответствуют базовым блокам управляющего графа, и находит переходы между ними.

В процессе построения графа потоков управления каждый узел снабжается меткой со счетчиком ссылок и все инструкции условного и безусловного перехода приводятся к единому виду:

- *If x op y Then goto label* – переход по условию;
- *goto label* – безусловный переход.

Особого внимания заслуживает обработка операторов обработки исключительных ситуаций. В Delphi для обработки исключительных ситуаций используется два оператора:

- *try/finally* – применяется, когда необходимо, чтобы код в секции finally выполнился в любом случае.
- *try/except* – при возникновении исключительной ситуации исполнение основного фрагмент кода прекращается, и выполнение передается в секцию *except*.

Для каждого блока кода подпрограммы, если в нём используются операторы обработки исключительных ситуаций, в объектных и исполняемых файлах хранится специальная таблица. В ней содержится вся необходимая информация

для их обработки: смещение до `try`; размер защищаемого блока кода; адрес и размер кода обработчика исключительных ситуаций; тип оператора (`finally`, `except`). В зависимости от размера кода подпрограммы могут применяться более компактные версии кодирования записей таблицы Small или менее компактные Fat.

При генерации управляющего графа оператор `try/finally` обрабатывается достаточно просто: необходимо разбивать последовательность операторов на части, соответствующие блокам `try` и `finally`, в соответствии с информацией об адресах и размерах защищаемого блока и обработчика исключительных ситуаций. При обработке `try/except` необходимо сформировать новый базовый блок, поскольку поток управления в случае ошибки не достигает кода, следующего за оператором `except`.

2.6 Восстановление высокоуровневых операторов

На практике в большинстве работ, посвященных декомпиляции, используются два подхода к анализу потока управления отдельных процедур. Первый подход использует дерево доминаторов для поиска естественных циклов, и в дальнейшем использует их для оптимизации. Второй подход, называемый интервальным анализом, включает методы, которые позволяют анализировать структуру процедуры в целом и разбивать её на вложенные участки, называемые интервалами. Теория интервалов была предложена Алленом [9] в начале 1970-х годов и использовалась для проведения оптимизаций при более тщательном анализе потоков данных. Наиболее глубокий вариант интервального анализа, называемый структурным анализом, был предложен Цифуентес [10]. Данный метод классифицирует абсолютно все структуры потока управления в процедуре. На первом этапе метод производит выделение и структурирование циклов. Далее, в порядке, обратном обходу в глубину, на граф накладываются шаблоны, соответствующие высокоуровневым операторам, и с помощью семантически эквивалентных преобразований граф сводится к одной абстрактной вершине, которая содержит в себе всю иерархию вложенных интервалов. В теории компиляции методы анализа потока данных на основе анализа интервалов называются методами устранения.

На основе анализа дерева доминирующих вершин разработан алгоритм структурирования управляющего графа, в основе которого лежит предложенный Джонсоном с коллегами [11] в 1994 году метод структурирования управляющего графа путем представления его в виде иерархии SESE-регионов (Single Entry Single Exit).

В работе [11] отмечено, что два любых SESE-региона в управляющем графе должны быть, либо вложенными друг в друга, либо непересекающимися. Структурный анализ на основе SESE-регионов и PST (Program Structure Tree) обычно используется для эффективного построения промежуточного представления в SSA (Static Single Assignment) форме, а также для более

эффективного анализа потоков данных в процессе компиляции, и в нём не предусматривается классификация выделяемых регионов. Для решения этих задач существенным является требование, что SESE-регион образует именно пара дуг, т. е. узел схождения имеет только одну входящую дугу, а узел расхождения – только одну исходящую дугу.

В данной работе выделяются двухтерминальные регионы (TT-регион), которые соответствуют схождению потока управления в уграфе и его последующему расхождению. Требования к TT-региону являются более слабыми, чем требования к SESE-региону. При этом, каждый SESE-регион является и TT-регионом, но не наоборот.

После того, как выделены все TT-регионы, полученный граф с помощью семантически эквивалентных преобразований сводится в одну абстрактную вершину, содержащую в себе иерархию подпрограммы. Для этого применяется итеративный алгоритм наложения шаблонов, на каждой итерации которого рассматривается TT-регион, имеющий наибольший уровень вложенности.

В качестве выделяемых шаблонов в основном используются подграфы, которые соответствуют высокоуровневым конструкциям языка Delphi: `block`, `while`, `repeat`, `if-then`, `if-then-else`, `case`, `unresolved` .

Для вычисления дерева доминаторов и постдоминаторов использовался алгоритм [12]. Хотя он имеет не самую лучшую теоретическую оценку сложности из алгоритмов, представленных в работах [13][14][15], но на практике использование его оказывается предпочтительней на графах с менее 30 000 вершин из-за маленькой скрытой константы и простоты реализации.

2.7 Результаты тестирования

Разработанный декомпилятор DCUIL2PAS был протестирован на специально подготовленном наборе процедур, взятых из модификации алгоритма LZW [16], написанного на языке Delphi. Поскольку прямых аналогов декомпилятору объектных файлов dcuil нет, было принято решение провести сравнительное тестирование с инструментов ILSpy, поскольку он наиболее близок по своим характеристикам к DCUIL2PAS и распространяется по свободной лицензии, для оценки качества – мера качества декомпиляции [17]:

$$C_{decom} = \sum_{prog \in TS} \frac{\max(0, K' - K)}{KLOC(prog)},$$

где TS – тестовый набор программ; prog – исходная программа; KLOC(prog) – количество тысяч значимых строк кода программы prog; K – сумма штрафов исходной программы; K' – сумма штрафов восстановленной исходной программы.

Штрафы за артефакты трансляции и неполноту восстановления (табл. 1) были изменены в соответствии с требованиями декомпиляции объектного кода

Delphi. Подсчет меры качества производился для каждой процедуры отдельно, при этом не учитывалось качество восстановления интерфейсной части модуля.

Табл. 1. Штрафы за артефакты трансляции и неполноту восстановления
Table 1. Penalties for translation artifacts and incompleteness of restoration

Конструкции программы	Назначаемые штрафы
Невосстановление имени переменной	1
Оператор перехода goto	3
Выход из середины цикла break	1
Оператор прерывания цикла continue	1
Невосстановленный оператор for	1

Полученные оценки качества декомпиляции представлены в табл. 2. На всех примерах мера качества разработанного декомпилятора оказалась выше, чем у ILSpy. Это связано в первую очередь с тем, что в процессе обработки объектных модулей линкером теряется часть информации об исходной программе, а также с тем, что декомпилятор ILSpy изначально разрабатывался, исходя из соображений, что исходная программа была написана не на языке Delphi.

Табл. 2. Мера качества декомпиляции
Table 2. Measure of decompilation quality

Название	DCUIL2PAS	ILSpy
BitWise	62.5	133.3
Compression	18.6	146
LZRW1KHCompressor	75	140
GetMatch	0	166.6

Помимо сравнительного анализа декомпилятором в пакетном режиме была разобрана стандартная библиотека VCL Delphi 8. Результаты, которые в большей степени демонстрируют производительность работы декомпилятора, приведены в табл. 3.

Табл. 3. Результат пакетной обработки (производительность)
Table 3. Batch processing result (performance)

Название	Кол-во файлов	Размер (мб)	Время обработки (с)
Delphi 8 VCL	325	39	396

Для оценки качества декомпиляции стандартной библиотеки VCL Delphi 8 в автоматическом режиме было просчитано количество процедур и функций, восстановленных в структурном виде (без использования оператора `goto`). Тестирование показало (табл. 4), что в 98,7% случаях удается восстановить программу без операторов `goto`.

Таблица 4. Результат пакетной обработки (качество)
Table 4. Batch processing result (quality)

Название	Кол-во процедур	Без goto	C goto	%
Delphi 8 VCL	9003	8879	124	1.3

Разработанный декомпилятор объектных файлов DCUIL позволяет восстанавливать исходный код на языке Delphi, который в большинстве случаев пригоден для дальнейшей его компиляции и полностью семантически эквивалентен исходному представлению программы. Данное программное средство позволяет существенно сократить время на решение задач, связанных с поддержкой и переработкой унаследованного и стороннего программного обеспечения, исходные тексты которого не предоставлялись или были утрачены; позволяет находить закладки в готовых модулях и компонентах Delphi под .NET; искать и исправлять ошибки.

3. Заключение

Результат декомпиляции файлов .dcuil оказывается более качественным, более понятным для исследователя кода по сравнению с результатами декомпиляции исполняемых файлов .NET, поскольку в нём отображается дополнительная информация, не попадающая в исполняемые файлы, например, имена переменных. Кроме того, учёт таких особенностей компилятора, как сокращённое оценивание логических выражений, на стадии улучшения промежуточного представления позволяет получить более понятный код.

Многие этапы разработанного метода и реализованные для их работы подпрограммы не зависят от особенностей кода для платформы .NET. Основную сложность для распространения метода на другие платформы представляет описание семантики машинных инструкций реальных процессоров, система команд которых сложнее байт-кода виртуальной машины, как по количеству инструкций, так и по их эффектам.

Список литературы

[1]. Turing A. M. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London mathematical society, vol. 2, no. 1, 1937, pp. 230–265.

[2]. Спецификация формата DCU на языке FlexT. 2017. URL: http://geos.icc.ru:8080/scripts/WWWBinV.dll/ShowR?DCU32.rfi (дата обращения: 2017-10-09).

[3]. А. Е. Хмельнов, И. В. Бычков, А. А. Михайлов. Декларативный язык FlexT — инструмент анализа и документирования бинарных форматов данных. Труды ИСП РАН, том 28, вып. 5. 2016 г., pp. 239–268. DOI: 10.15514/ISPRAS-2016-28(5)-15

[4]. Necula G. C., McPeak S., Rahul S. P. et al. CIL: Intermediate language and tools for analysis and transformation of C programs. International Conference on Compiler Construction. Springer, 2002, pp. 213–228.

[5]. Михайлов А. А. Промежуточное представление подпрограмм в задаче декомпиляции объектных файлов dcuil. Вестник Бурятского государственного университета, №. 9-3, 2014 г., стр. 32-38.
[6]. Михайлов А. А. Анализ графа потоков управления в задаче декомпиляции подпрограмм объектных файлов dcuil. Вестник Новосибирского государственного университета. Серия: Информационные технологии, том 12, №. 2, 2014 г., стр. 74-79.
[7]. Allen F. E., Cocke J. A program data flow analysis procedure. Communications of the ACM, vol. 19, №. 3, 1976, p. 137.
[8]. Aho A. V. Compilers: Principles, Techniques and Tools (for Anna University), 2/e. Pearson Education India, 2003.
[9]. Allen F. E. Control flow analysis. ACM Sigplan Notices, ACM, vol. 5, № 7, 1970, pp. 1-19.
[10]. Cifuentes C. Structuring decompiled graphs. Compiler Construction. Springer Berlin/Heidelberg, 1996, pp. 91-105.
[11]. Johnson R., Pearson D., Pingali K. The program structure tree: Computing control regions in linear time. ACM SigPlan Notices, vol. 29, №. 6, 1994, pp. 171-185.
[12]. Cooper K. D., Harvey T. J., Kennedy K. A simple, fast dominance algorithm. Software Practice & Experience, vol. 4, №. 1-10, 2001, pp. 1–8.
[13]. Ахо А, Дж Ульман. Теория синтаксического анализа, перевода и компиляции. Том 1. Компиляция. М., Мир, 1978.
[14]. Lengauer T., Tarjan R. E. A fast algorithm for finding dominators in a flowgraph. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 1, № 1, 1979, pp. 121–141.
[15]. Кнут Д. Э. Искусство программирования. Т. 3. Сортировка и поиск. Издательский дом Вильямс, 2000..
[16]. Williams R. N. An extremely fast Ziv-Lempel data compression algorithm. Data Compression Conference, 1991. DCC'91, IEEE, 1991, pp. 362–371.
[17]. Трошина Е. Н. Исследование и разработка методов декомпиляции программ: Кандидатская диссертация. Московский государственный университет имени М. В. Ломоносова. 2009.

Delphi object files decompiler

A.A. Mikhailov <mikhailov@icc.ru>
A.E. Hmelnov <petrov@icc.ru>
Matrosov Institute for System Dynamics and Control Theory of Siberian Branch of Russian Academy of Sciences,
Lermontov str., 134, Post Box 292 664033, Irkutsk, Russia

Abstract. The work is devoted to solving the problem of decompiling one of the types of DCU - .dcuil format files created by the compilers of those versions of Delphi that worked for the .NET plat-form. A method for solving this problem is developed, consisting of a number of steps: syntactic analysis of the CIL code; control flow graph generation; intermediate representation generation; structuring control flow graph; dataflow analysis; intermediate representation optimization; code generation.

Keywords: reverse engineering; object code; Delphi.

DOI: 10.15514/ISPRAS-2017-29(6)-5

For citation: Mikhailov A.A., Hmelnov A.E. Delphi object files decompiler. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 6, 2017, pp. 105-116 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-5

References

[1]. Turing A. M. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London mathematical society, vol. 2, no. 1, 1937, pp. 230–265.
[2]. DCU format specification in FlexT. 2017. URL: http://geos.icc.ru:8080/scripts/WWWBinV.dll/ShowR?DCU32.rfi. accessed: 10.09.2017
[3]. A.Y. Hmelnov, I.V. Bychkov, A.A. Mikhailov. A declarative language FlexT for analysis and documenting of binary data formats. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 5, 2016, pp. 239-268. (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-15
[4]. Necula G. C., McPeak S., Rahul S. P. et al. CIL: Intermediate language and tools for analysis and transformation of C programs. International Conference on Compiler Construction. Springer, 2002, pp. 213–228.
[5]. Mikhailov A. A. Intermediate representation for dcuil files decompilation. Vestnik Buryatskogo gosudarstvennogo universiteta [Bulletin of the Buryat State University], No. 9-3, 2014, pp 32-38 (in Russian).
[6]. Mikhailov A. A. Control flow analysis for dcuil files decompilation. Vestnik Novosibirskogo gosudarstvennogo universiteta. Seriya: Informacionnye tekhnologii [Novosibirsk State University Journal of Information Technologies], vol. 12, no 2. 2014, pp 74-79 (in Russian).
[7]. Allen F. E., Cocke J. A program data flow analysis procedure. Communications of the ACM, vol. 19, №. 3, 1976, p. 137.
[8]. Aho A. V. Compilers: Principles, Techniques and Tools (for Anna University), 2/e. Pearson Education India, 2003.
[9]. Allen F. E. Control flow analysis. ACM Sigplan Notices, ACM, vol. 5, № 7, 1970, pp. 1-19.
[10]. Cifuentes C. Structuring decompiled graphs. Compiler Construction. Springer Berlin/Heidelberg, 1996, pp. 91-105.
[11]. Johnson R., Pearson D., Pingali K. The program structure tree: Computing control regions in linear time. ACM SigPlan Notices, vol. 29, №. 6, 1994, pp. 171-185.
[12]. Cooper K. D., Harvey T. J., Kennedy K. A simple, fast dominance algorithm. Software Practice & Experience, vol. 4, №. 1-10, 2001, pp. 1–8.
[13]. Alfred V. Aho, Jeffrey D. Ullman. The theory of parsing, translation, and compiling. 1978.
[14]. Lengauer T., Tarjan R. E. A fast algorithm for finding dominators in a flowgraph. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 1, № 1, 1979, pp. 121–141.
[15]. Donald E. Knuth. The Art of Computer Programming: Sorting and Searching. Addison Wesley Series in Computer Science and Information Processing, 2000. Vol. 3.
[16]. Williams R. N. An extremely fast Ziv-Lempel data compression algorithm. Data Compression Conference, 1991. DCC'91, IEEE, 1991, pp. 362–371.
[17]. Troshina E. N. Decompilation methods. Phd dissertation. Lomonosov Moscow State University. 2009 (in Russian).