Декларативный язык FlexT – инструмент анализа и документирования бинарных форматов данных

А.Е. Хмельнов <hmelnov@icc.ru>
И.В. Бычков <bychkov@icc.ru>
А.А. Михайлов <mikhailov@icc.ru>
Институт динамики систем и теории управления имени Матросова СО РАН, 664033, Россия, г. Иркутск, ул. Лермонтова, д. 134

Аннотация. Язык FlexT разработан для спецификации бинарных форматов данных. Язык является декларативным, рассчитанным на хорошее восприятие человеком, его основными конструкциями являются определения типов данных, которые напоминают определения типов в императивных языках программирования, но являются более гибкими. В работе сделан обзор возможностей современных проектов, направленных на спецификацию бинарных форматов файлов. Далее рассматриваются особенности языка FlexT, отдельно описываются возможности языка, позволяющие работать с форматами кодирования машинных команд. Кратко описаны реализованные программные системы, использующие интерпретатор FlexT и некоторые новые возможности поиска информации в бинарных файлах, которые даёт использование спецификаций.

Ключевые слова: спецификация бинарных форматов данных, спецификация кодирования машинных команд, декларативный язык, дизассемблер

DOI: 10.15514/ISPRAS-2016-28(5)-15

Для цитирования: 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., стр. 239-268. DOI: 10.15514/ISPRAS-2016-28(5)-15

1. Введение

При выборе существующих или разработке собственных форматов файлов для сохранения информации на диске, а также при необходимости организовать файловый обмен данными между различными приложениями, разработчики программного обеспечения встают перед выбором между бинарными и текстовыми форматами представления информации. Бинарные форматы в большинстве случаев являются более компактными, программный код для работы с бинарными данными также может быть значительно более простым

по сравнению с кодом для обработки текста (например, одной командой можно прочитать из бинарного файла запись, содержащую сразу несколько десятков полей). Однако, текстовые файлы являются более прозрачными: для их просмотра можно воспользоваться любым текстовым редактором. Поэтому, в настоящее время разработчики часто жертвуют эффективностью ради упрощения контроля правильности структуры файла, и выбирают текстовые форматы. Часто такие форматы основываются на синтаксисе XML, поскольку для этого синтаксиса существуют готовые библиотеки и инструментальные средства, облегчающие разработку алгоритмов чтения/записи информации, и автоматизирующие контроль правильности структуры файла.

Для поддержки использования нового формата данных в разрабатываемой программе можно подключить библиотеку для работы с этим форматом или написать собственный код с использованием спецификации формата. Любая библиотека навязывает разработчику свои структуры данных, которые, если программа пишется не с нуля, скорее всего, будут отличаться от уже используемых. В результате потребуется написать дополнительный программный код для конвертации данных, что скажется на скорости работы программы. Кроме того, подходящие библиотеки могут и отсутствовать. При отсутствии готовых библиотек, а также при желании получить более эффективную реализацию, придётся написать код для работы с форматом данных по его спецификации. Используемая спецификация может быть написана на естественном языке, кроме того, для изучения формата данных могут служить исходные тексты программ работающих с этим форматом и даже примеры относящихся к формату файлов. Рассмотрим особенности каждого из этих источников информации о формате более подробно:

- Спецификации на естественном языке практически всегда содержат неточности и неоднозначности толкования. То, что было очевидно автору спецификации, может быть непонятно читателю, или, может быть понято им превратно. Содержащаяся в такой спецификации информация не была верифицирована путём её использования для реальной работы с данными, поэтому очень часто она является ненадёжной.
- С другой стороны, исходные тексты программ для работы с рассматриваемыми файлами, прошли проверку на реальных данных и, скорее всего, не содержат подобных ошибок. Однако, в этих исходных текстах информация о структурах данных файла разбросана по коду, предназначенному для конкретного способа работы с этими данными, поэтому изучение такого кода в качестве документации по формату является очень непростой задачей.
- Ещё более сложной является задача восстановления недокументированного формата данных по примерам файлов этого формата. Она требует выполнения многочисленных попыток разбора файлов при некоторых предположениях об их структуре с выводом

всей извлекаемой информации для анализа. Программа, выполняющая такую работу, будет существенно сложнее программы, просто считывающей информацию во внутреннее представление.

Кроме того, при разработке программы, записывающей данные в бинарный формат, часто очень остро встаёт проблема её отладки: сложно понять, что требуется исправить, когда используемое для проверки правильности полученного файла приложение выдаёт неинформативное сообщение, рассчитанное на конечного пользователя, например, просто пишет: «Ошибка в формате файла» (без указания места и других подробностей).

Таким образом, является целесообразной разработка специальных инструментов для задачи анализа содержимого бинарных файлов с использованием спецификаций форматов данных. В статье рассматривается разработанный авторами язык FlexT, предназначенный для описания бинарных форматов данных, и реализованные с использованием интерпретатора этого языка инструменты, предназначенные для анализа бинарных файлов. Преимуществами использования формальных спецификаций являются:

- компактность, отсутствие не относящихся к способам хранения данных сведений, что облегчает их восприятие человеком;
- верифицируемость посредством их применения для анализа относящихся к формату корректных данных;
- возможность использования спецификаций для локализации ошибок в сгенерированных файлах.

Далее в главе 2 будет сделан обзор существующих средств спецификации бинарных форматов данных, в главе 3 будут рассмотрены особенности и основные конструкции языка FlexT. В главе 4 будут описаны типы данных языка FlexT, ориентированные на спецификацию кодирования машинных команд. В главе 5 будут рассмотрены разработанные программы, использующие спецификации для исследования бинарных файлов и ряд возможностей инструментов, использующих интерпретатор FlexT.

2. Обзор средств спецификации бинарных форматов данных

К моменту создания языка FlexT (конец 1990-х) задачей спецификации бинарных форматов данных почти никто не занимался. В качестве обзора литературы тогда удалось найти только одну хоть немного действующую ссылку на проект BFF (Binary File Format Definition), сейчас страница проекта существует и находится по адресу [1]. Первоочередной задачей разработки языка BFF являлось обратное проектирование формата AutoCAD DWG. Автор использовал термин «грамматика», т.е. отталкивался от концепции формальных грамматик при проектировании языка спецификаций. Другие бинарные форматы, помимо DWG, описаны не были, уровень описания формата DWG также не удаётся оценить.

За прошедшее время уровень понимания необходимости использования спецификаций для работы с бинарными данными существенно вырос. Рассмотрим появившиеся проекты, в основном в порядке их упоминания в выдаче поисковой системы.

Стандарт DFDL (Data Format Description Language) [2][3] разрабатывается для описания текстовых и бинарных данных, используемых в GRID-системах, судя по приводимым примерам, в основном табличных (на уровне возможностей формата CSV). Стандарт DFDL был опубликован, как рекомендованные предложения форума Open Grid в 2011 г. Целью разработки стандарта является унификация обработки (чтения и записи) из текстовых (XML, CSV) и бинарных данных содержащегося в них «набора информации» (information set). Описания форматов кодируются на XML. Имеется коммерческая реализация парсера DFDL в составе продукта IBM Integration Bus [4], а также открытая реализация [5]. Сами описания бинарных данных являются достаточно примитивными (на уровне записей с полями постоянного размера в терминах FlexT), в силу ограничений стандарта, а также потому, что существующий парсер поддерживает не все предусмотренные этим стандартом возможности [6]. Тем не менее, эта реализация позволяет, например, использовать для обмена данными простой бинарный формат, сопроводив его спецификацией на DFDL. Никакие описания настоящих форматов данных, таких как DBF, на DFDL найти не удалось. Популярность DFDL объясняется большим объёмом подготовленной документации по формату и участием в его разработке гиганта IBM.

Аналогичную задачу решает язык MFL (Message Format Language) в составе продукта WebLogic Integration фирмы Oracle [7]. В данном случае обмен информацией с использованием специальных бинарных форматов выполняется между приложениями, использующими эту платформу. Язык также основан на XML и позволяет описывать форматы с последовательной формой хранения информации (без указателей в терминах FlexT). Допускается использование повторяющихся элементов, а также необязательных полей. Для построения спецификации формата используется специальный интерактивный редактор Format Builder. Так же, как и DFDL, MFL не предназначен для спецификации произвольных бинарных форматов, а лишь тех, которые укладываются в рамки его возможностей, с учётом которых и должен проектироваться формат.

Существует ряд проектов, направленных на описание сетевых протоколов, например, NetPDL [8] и BinPAC [9]. Язык NetPDL предназначен для документирования форматов пакетов сетевых протоколов и основан на синтаксисе XML. Язык BinPAC направлен на генерацию кода для чтения пакетов с использованием генератора синтаксических анализаторов Bison. Поскольку сама природа сетевых протоколов предполагает последовательное чтение данных, такие языки также позволяют описать лишь форматы с последовательной формой хранения информации (без указателей в терминах FlexT). Поддерживаются все основные конструкции, предназначенные для

описания потоков данных (массивы, записи, варианты в терминах Flext), параметризация типов. В языке NetPDL имеются простые битовые типы, в языке BinPAC битовых типов нет.

В ходе выполнения ранее упоминавшегося обзора в конце 1990-х годов остался незамеченным язык EAST (Enhanced Ada SubseT) [10], часть информации о котором была опубликована ещё в 1997. Язык разработан консультативным комитетом систем космических данных (CCSDS), в котором участвует и Российское космическое агентство, для облегчения обмена информацией, передаваемой космическими системами. В основу языка положена подсистема определения типов данных языка Ada. В языке EAST поддерживаются поля записей переменного размера, массивы с задаваемой в некотором поле длиной, а также со стоп-маркером; вариантные части, содержимое которых определяется некоторым внешним полем (выбор по внутреннему содержимому не предусмотрен). Более сложные, чем значение некоторого поля, выражения для параметров типов не поддерживаются. Большое внимание уделяется битовым типам данным и учёту порядка байтов. Отсутствуют указатели, как нехарактерные для данных, передаваемых последовательно. Используемый способ передачи параметров типов данных не слишком гибок и не очень удобен.

Язык HUDDL [11] разработан для описания форматов представления гидрографических данных. Язык также основан на синтаксисе XML и ориентирован на описание потоков бинарных данных (уровень записей и массивов в терминах FlexT). Спецификации используются для генерации кода для работы с данными на различных языках программирования. Авторам известен проект DFDL, но они предпочитают разработать свой язык, ориентированный на особенности гидрографических данных.

В проекте «Advanced Language Processing Technology Applied to Digital Records» [12] сотрудники военного НИИ из США рассматривают задачу интеграции разнородной цифровой информации, используемой для принятия военных решений. Предлагается использовать аппарат атрибутивных грамматик для описания бинарных форматов данных [13]. Далее используется генератор синтаксических анализаторов (парсеров) ANTLR [14] для автоматического построения по спецификации-грамматике кода чтения бинарного файла. Все приведённые примеры имеют дело с форматами с последовательной записью данных (в терминах FlexT: без использования указателей). Такой подход позволяет частично ускорить разработку модулей чтения информации для ряда форматов.

В проекте DataScript [15] спецификации форматов данных используются для генерации библиотек для чтения данных на языке Java. Вместе с исходными текстами [16] опубликованы два описания форматов: файла класса Java (без описания машинных команд) и DVI (оба формата являются последовательными, без указателей в терминах FlexT). Кроме того, в статье [15] приводятся фрагменты спецификации формата ELF, для описания которого

требуются указатели. Вместо указателей в DataScript используются метки, содержащие выражения с адресом фрагмента файла. С 2003 г. исходные тексты проекта не обновлялись. Спецификация рассматривается, как набор определений типов данных, этот подход наиболее близок используемому в языке FlexT. При этом для работы с битовыми типами определены отдельные (отличные от байтовых) типы данных (битовые поля), в полном объёме битовые типы не поддерживаются. Для описания вариантных блоков используется конструкция union, в которой происходит выбор того варианта, заданные для которого ограничения выполняются, сами ограничения могут задаваться в виде ожидаемых значений отдельных полей. Этот подход не позволяет описать выбор вариантов по задаваемому извне значению, например, когда тип блока задаётся в той записи, где находится указатель на блок.

Целью проекта Miraplacid Binary DOM (Binary Document Object Model) [17] является реализация универсальной библиотеки для доступа к бинарным и текстовым данным в стиле DOM, используемой для HTML-документов. В спецификациях бинарных данных (файлы с расширением `bdd`) используются массивы записи и указатели (в терминах FlexT), но не удалось найти никакой поддержки вариантных блоков. Также заметно отсутствие битовых типов данных. Спецификация представляет собой набор типов данных. На сайте [18] описан 21 формат, некоторые из них – текстовые (INI, RTF, XML). Реализована программа для просмотра содержимого бинарного файла по спецификации (отображается дерево обнаруженных структур данных и шестнадцатеричный дамп, в котором выделяется блок памяти, соответствующий выбранному узлу дерева)

Проект Kaitai Struct [19] ориентирован на создание парсеров бинарных данных по спецификациям. Спецификация в файле с расширением `ksi` по своей структуре напоминает формат JSON, в котором отступы используются вместо скобок. Основной структурой данных является последовательность (запись в терминах FlexT), которая может иметь повторяющиеся элементы (массивы в терминах FlexT). Поддерживаются указатели, а вместо вариантных типов данных могут использоваться условия вхождения полей в записи, что может иногда оказаться не слишком удобным, в особенности при задании условия для поля, соответствующего ветви варианта ELSE (там необходимо записать конкатенацию отрицаний условий всех остальных ветвей). Также не поддерживаются битовые типы данных.

Программа Synalyze It! [20] является средством просмотра/шестнадцатеричным редактором бинарных файлов. Эта программа предназначена для работы под macOS (распространяется платно), существует её бесплатная версия для других платформ под названием Hexinator [21]. Программа может использовать спецификации бинарных форматов для формирования дерева найденных в файле структур данных и выделения цветом соответствующих им фрагментов файла. Спецификации форматов данных называются грамматиками и хранятся в основанном на XML представлении. Для разработки спецификаций

реализован интерактивный редактор. В списке имеющихся описаний [22] в основном представлены специфичные для macOS форматы, но есть и более распространённые, такие как DBF, Shape или PE EXE. Используемый язык спецификаций нельзя считать полностью декларативным, поскольку описания форматов содержат достаточно большие вставки на языке Python.

Таким образом, к настоящему времени во многих предметных областях осознана необходимость использования спецификаций форматов данных. При этом иногда ставится задача не описания произвольных форматов, а лишь облегчения работы с некоторым их подмножеством. В некоторых случаях задача описания произвольных форматов ставится, но реализуются не все конструкции, необходимые для её решения. Часто для спецификаций выбирается представление XML, что затрудняет восприятие текста спецификации человеком и требует использования специальных средств просмотра и интерактивных редакторов, которые, хотя и облегчают выполнение ряда операций редактирования, но затрудняют выполнение некоторых других действий, например, не позволяют одновременно увидеть и изменить характеристики сразу нескольких узлов. В некоторых проектах о спецификации бинарного формата думают, как о грамматике, а в других спецификация рассматривается, как набор типов данных. Многие языки спецификаций были созданы для облегчения написания кода для работы с данными, поэтому эту задачу можно считать востребованной. Также очень заметно, что в статьях по тематике очень редко встречаются ссылки на аналоги, упоминались как аналоги лишь нескольких проектов: EAST, DFDL и DataScript.

3. Особенности языка FlexT

Бóльшая часть информации о формате данных на языке FlexT задаётся при помощи набора определений типов данных. При этом, по сравнению с типами данных в императивных языках программирования, типы данных FlexT могут содержать составляющие, размер которых определяется конкретными данными, представленными в этом формате. Т.е. можно сказать, что типы гибко подстраиваются под данные, этим объясняется название языка **FlexT** (от англ. **Flex**ible **T**ypes). После определения типов данных необходимо задать размещение в памяти некоторых элементов данных, относящихся к каким-то из этих типов. Такие элементы данных по аналогии с императивными языками программирования будем называть переменными (хотя они и описывают неизменяемые данные).

Синтаксис языка выбран так, чтобы он хорошо воспринимался человеком. При проектировании некоторых из рассмотренных в обзоре языков, например, языка EAST, такая задача явно формулировалась в требованиях к разработке. Рассмотрим основные особенности и принципы создания языка FlexT, которые делают его применимым для описания широкого круга форматов.

3.1 Уровни языков спецификации форматов данных

По возможностям работы с описываемыми данными можно различать два уровня языков спецификации данных: *спецификация интерпретации* и *спецификация редактирования*. Язык спецификации интерпретации должен позволять *легко* описывать интерпретацию произвольного (в рамках определённых ограничений) *класса данных*. Под интерпретацией здесь понимается не преобразование всех данных в некоторый конкретный формат, а наличие в спецификации информации о способах извлечения из описанных данных значений свойств рассматриваемого класса данных. Например, спецификация формата файла растровой графики должна позволять извлечь из такого файла информацию о размерах изображения (ширина, высота) и цвете каждого пикселя, расположенного в границах изображения.

Язык спецификации редактирования должен кроме определения функций-наблюдателей для считывания информации давать определения конструкторов, позволяющих создавать по заданным свойствам новые экземпляры данных. При этом приходится учитывать дополнительные детали, например, распределение памяти, порядок порождения элементов данных, соглашения по их выравниванию, способ заполнения пропусков, алгоритм генерации идентификаторов новых объектов, и т.д. В текущей версии язык FlexT не поддерживает описание таких спецификаций.

Слова "легко", "простота" и т.п. часто употребляется при определении понятия "спецификация". Подразумевается, что запись некоторой информации на языке спецификации должна быть проще, чем на обычном языке программирования. Более существенным здесь является не субъективное понятие простоты, а отсутствие в спецификации избыточной информации. Язык спецификации должен быть как можно более декларативным: он должен описывать то, как размещаются данные, а не то, как их надо читать.

Одни и те же данные могут интерпретироваться разными способами. Например, файл одноканального растрового формата может отображаться, как изображение в тонах серого, или как матрица высот, или как двумерная таблица. В качестве базовой интерпретации, пригодной для описания любого формата данных, можно предложить такую, которая приписывает тип каждому элементу данных — выполняет *идентификацию типов данных*. Такая интерпретация является базовой и минимальной, поскольку при использовании некоторого элемента данных в интерпретации более высокого уровня всё равно придётся определить, к какому типу этот элемент относится. В качестве класса данных, в который они отображаются при идентификации типов, используется набор взаимосвязанных *элементов данных*. Каждый элемент данных характеризуется своим размещением (*адресом* и *размером*) и *типом*, и может содержать ссылки на другие элементы данных, а также необходимую для полного описания других элементов данных информацию. Размер элемента данных определяется его типом. Элементы данных могут быть составными — в этом случае внутри них можно выделить элементы меньшего размера.

Элемент данных можно сравнить с термом, а идентификацию типов данных — с эрбрановой интерпретацией в логике первого порядка. Интерпретация типов данных может быть *неполной*. В этом случае она содержит элементы данных, которым не сопоставлен тип или, точнее, в этом случае можно считать, что им сопоставлен примитивный тип — *сырые данные*.

Рассматриваемый в данной работе язык спецификации форматов данных FlexT позволяет описывать интерпретацию статических данных, в первую очередь — в виде идентификации типов. Данный язык применяется для описания форматов данных в программах просмотра/дизассемблере бинарных файлов, а также в программе ExeXpl, предназначенной для исследования кода и структур данных в исполняемых файлах Windows.

3.2 Динамические и статические типы данных

Под статическими типами данных здесь мы будем понимать аналоги большинства традиционных типов данных, реализованных в процедурных языках программирования. Их отличительной особенностью является то, что размер элемента данных такого типа и внутреннее размещение составляющих его элементов определены в момент компиляции и не зависят от конкретных данных. В процедурных языках программирования, по крайней мере, в тех, которые реально используются в настоящее время, составные типы данных могут содержать только статические составляющие.

Размер элемента данных динамического типа и внутреннее размещение его составляющих могут зависеть от конкретных данных. Далее слово "динамический" будет использоваться именно в этом смысле. Примером динамических типов в традиционных процедурных языках являются строковые константы. Например, для ASCIIZ строк, чтобы определить занимаемый ими размер, нужно просмотреть всю строку в поисках нулевого символа.

Динамические типы данных непригодны в качестве типов переменных, по крайней мере, если этот тип изменяемый (в терминах [23]), поскольку присваивание новых значений элементам таких данных может означать изменение размера, а такие операции ни один компилятор не сможет эффективно поддержать. Для полей типов данных, содержащих те же строки или записи с вариантами, память сразу выделяется по максимуму. Примером поддержки компилятором динамического перераспределения памяти, занимаемой значением переменной, являются huge-строки Delphi, память под которые автоматически запрашивается в куче, при этом сами переменные такого типа фактически являются указателями и всегда занимают 4 байта.

В то же время, если рассматривать статические данные, которые программа может только читать, то здесь иногда используются весьма изощрённые способы их кодировки. В качестве примера статических данных в коде программы можно привести RTTI Delphi — способ представления метаинформации о типах данных:


```
PTypeInfo = ^TTypeInfo;
TTypeInfo = record
  Kind: TTypeKind;
  Name: ShortString;
 {TypeData: TTypeData}
end;

function AfterString(Str: PShortString): Pointer; inline;
begin
  Result := PByte(Str) + Length(Str^) + 1;
end;

function GetTypeData(TypeInfo: PTypeInfo): PTypeData;
begin
  Result := AfterString(@TypeInfo^.Name);
end;
```

Листинг 1. Фрагменты файла TypeInfo.pas

Listing 1. Excerpts from the file TypeInfo.pas

Информация о типе данных закодирована в записи типа `TTypeInfo`, в которой за кодом вида типа данных и его именем следует запись `TypeData` с остальной информацией о типе. Определение поля `TypeData` заключено в комментарий, т.к. на самом деле эти данные находятся непосредственно после последнего символа строкового поля Name, размер которого зависит от длины имени конкретного типа. Поскольку смещение поля TypeData зависит от конкретных данных — имени типа, такую структуру нельзя непосредственно представить на Паскале. Приходится писать специальный код для доступа к этому полю – функцию `GetTypeData`.

Подобные трудности испытывают все авторы книг по форматам файлов данных при попытке описать эти данные, например, на языке C. Причина всех этих проблем — *в использовании для спецификации форматов данных языка, предназначенного для описания типов переменных*. В то же время, если отвлечься от необходимости придерживаться ограничений на типы переменных, то можно естественным образом поддержать в языке описание достаточно сложных зависимостей между элементами данных. Именно этот подход использован при проектировании языка FlexT.

3.3 Использование механизма определения типов данных

Таким образом, в качестве основного элемента языка спецификации форматов данных мы будем рассматривать механизм определения типов. В процедурных языках программирования механизм определения типов можно считать отдельным вложенным языком, т.к. типы данных влияют, например, на

семантику и синтаксис процедур для работы с данными этих типов, но не наоборот. Если в процедурных языках программирования основная информация программы содержится в коде процедур, то при описании способов хранения информации основная нагрузка ляжет именно на механизм определения типов. Рассмотрим, какими возможностями должен обладать такой механизм.

При идентификации типов данных любые физические данные будем рассматривать как набор элементов данных. Каждый такой элемент характеризуется своим адресом, размером и интерпретацией (типом). Составные элементы данных разбиваются на более мелкие элементы. Всю информацию, которую должна предоставить спецификация идентификации типов данных о каждом элементе данных, можно разбить на набор утверждений о том:

1. На какие составляющие, каких типов этот элемент разбивается;
2. Где находятся эти составляющие (каковы их адреса);
3. Сколько места они занимают.

При этом из того, что в рассматриваемых данных содержится некоторый элемент составного типа, выводится информация о типах и размещении его составляющих (из которой, в свою очередь, может выводиться информация о размере этого составного элемента).

Составляющие элемента данных могут размещаться либо на некотором фиксированном смещении от его начала, либо на некотором смещении от конца другой составляющей. Составной элемент некоторого типа может содержать либо фиксированное число составляющих, либо это число может определяться динамически, т.е. отличаться у разных экземпляров одного типа. Если число составляющих фиксировано, то их можно поименовать, в этом случае элемент данных можно описать как запись с полями — составляющими. При динамическом определении набора составляющих можно рассмотреть два основных случая: 1) когда содержание части полей определяет, какие ещё поля входят в состав записи — этот случай описывается при помощи вариантных типов данных; 2) когда число составляющих определяется динамически — при этом, т.к. размер спецификации конечен, она должна содержать итеративные элементы. Эти итеративные элементы можно выделить в массив. Более сложные случаи могут быть описаны при помощи комбинации массивов и вариантов. Т.е. можно рассчитывать, что для описания произвольных структурированных составных элементов данных достаточно использовать конструкторы записи, массива и варианта (в широком смысле).

Кроме механизма описания структуры отдельного элемента данных нужны ещё механизмы для описания зависимостей между различными элементами. Наиболее важным случаем такой зависимости является указатель — элемент данных, в котором закодирована информация об адресе и типе другого

элемента данных. Также следует учесть, что для правильной интерпретации одних элементов данных может потребоваться информация, содержащаяся в других элементах. Например, запись может содержать поля `Счётчик` и `Таблица`, где поле `Таблица` является массивом, число элементов которого записано в поле `Счётчик`. Для описания подобных случаев используется механизм параметризации типов данных.

3.4 Краткое описание языка FlexT

В текущей реализации интерпретатора FlexT программа компилируется после загрузки разбираемых данных. Это создаёт некоторые сложности при попытке использования спецификаций для других целей, например, для генерации кода чтения. С другой стороны, при таком подходе упрощается написание спецификаций за счёт возможности использования динамических выражений, например, в директивах условной компиляции, а именно максимальная простота написания спецификаций и является целью создания языка.

Дадим несколько пояснений, необходимых для понимания следующих далее фрагментов кода. Язык FlexT является нечувствительным к регистру. Перевод строки может быть разделителем, если он происходит в месте возможного окончания конструкции (величина отступа при этом значения не имеет). Вложенные определения типов можно брать в круглые скобки, тем самым решается вопрос о принадлежности блоков дополнительной информации о типе.

3.4.1 Выражения, параметры и свойства типов

Типы данных могут иметь ряд свойств, набор которых зависит от конструктора этого типа, например, размер и число элементов у массива, или номер случая у варианта. Каждый тип имеет свойство Size (Размер). Свойства могут задаваться конкретным значением при определении типа, либо некоторым выражением, которое вычисляет значение данного свойства через значения и/или свойства вложенных элементов данных сложного типа, а также, м.б. через значения параметров типа. Некоторые правила для вычисления свойства Размер могут автоматически добавляться компилятором, если они не указаны явно, это касается, например, случая, когда известен размер всей записи и всех её полей, кроме последнего. Параметры в объявлении типа представляют ту информацию, которую необходимо указать дополнительно при использовании (вызове) данного типа.

Выражения для значений параметров и свойств типов оцениваются интерпретатором в ленивом режиме [24], т.е. они вычисляются только при необходимости использования такого значения. В некоторых случаях это позволяет избежать переполнения стека.

3.4.2 Ссылки на свойства в выражениях

Выражения вычисляются в контексте элемента данных некоторого типа, ссылка на этот элемент обозначается символом `'@'`, при этом `'@' ':' <Имя>` означает ссылку на параметр или свойство <Имя> данного элемента. Также в выражениях могут использоваться специальные конструкции: `<Адресное выражение>'@'` — ссылка на родительский элемент, т.е. на элемент данных того типа, в контексте которого данный тип определён. Пример:

```
T1 struc
  int Cnt1
  int Cnt2
  array[@.Cnt1] of (array[@@.Cnt2] of word) Tbl
ends
```

Выражения для параметра конструктора типа вычисляются в контексте вызова этого конструктора, поэтому значение свойства `Count` конструктора главного массива `Tbl` вычисляется в контексте типа `T1`, а для вложенного массива — в контексте типа массива `Tbl`, поэтому требуется написать `@@.Cnt2`, чтобы сослаться на поле `Cnt2` в записи – родителе главного массива.

`<Переменная>':' '@'` — ссылка на элемент даных-хозяин, т.е. на элемента данных, в который вложен рассматриваемый элемент. Т.к. некоторый тип может использоваться в разных составных типах, чтобы воспользоваться этой конструкцией, необходим оператор приведения к типу с проверкой: `<Переменная> 'AS' <Тип>`.

`'@' ':' '#'` - номер вложенного элемента данных в содержащем его элементе, например, индекс массива.

3.4.3 Блоки с дополнительной информацией о типе

Для определения различных типов данных в языке существуют конструкторы (массива, записи, варианта, и т.д.), синтаксис записи которых существенно различается. В то же время есть ряд задач, которые требуется решать для любых типов, независимо от того, при помощи каких конструкторов они определяются. Для этих целей используются блоки с дополнительной информацией о типе, которые могут быть записаны через `':'` после любого определения типа. К этим блокам относятся:

Табл. 1. Блоки с дополнительной информацией о типе

Table 1. The blocks of data type additional information

Блок	Пример	Описание
утверждений	`:[@:Size=@.Len, @.offset:Cnt=@.count]`	Позволяет задать значения свойств типа данных или параметров его составляющих с использованием параметров типа или информации о его составляющих
условий корректности	`:assert[@.Op>=0x80]`	Задаёт логическое условие, которое должно выполняться для данных, относящихся к этому типу
отображения	`:displ=('#',HEX(2*@))`	Позволяет переопределить способ отображения значений типа
именования	`:autoname=('sec_',@.tag)`	Аналогичен displ, но служит для задания имён переменных
дополнительного свойства типа	`:let Val=(@.0)exc(@.1)`	Позволяет задать способ вычисления свойств типа, которые далее могут использоваться в выражениях

Блок утверждений можно использовать, чтобы связать параметры типа со значениями его свойств в том случае, когда конструктор типа не позволяет это сделать. Например, конструктор массива позволяет задать количество элементов, но не суммарный размер массива. Также этот блок используется, чтобы задать свойства определяемого типа по значениям его составляющих. Т.к. компилятор FlexT является однопроходным, в выражениях для параметров типа нельзя сослаться на ещё не прочитанные составляющие, поэтому, чтобы, например, задать значение параметра поля записи по значению следующего за ним другого поля следует использовать блок утверждений записи.

Блок именования может использоваться для переменных, положение которых в памяти было найдено при помощи указателей. По умолчанию в этом случае переменная обозначается квалификатором доступа к ней через цепочку указателей, который может оказаться очень длинным и не слишком информативным. При наличии блока именования вместо этого квалификатора будет использоваться сформированная этим блоком строка. Эту возможность удобно использовать, например, когда в состав данных типа входит информация о некотором имени объекта, который эти данные описывают.

Блок дополнительного свойства типа позволяет описать способ вычисления некоторого значения, закодированного этим типом данных. Например, во многих бинарных форматах используется приём, когда целое число может быть представлено различным количеством байтов в зависимости от его значения. Выражение, описывающее декодирование такого числа, может быть задано, как дополнительное свойства этого типа данных. При работе с данными значение этого свойства так же, как и остальных свойств, вычисляется в ленивом режиме: при первом обращении с последующим запоминанием.

3.5 Типы данных языка FlexT

Основные конструкторы типов данных языка FlexT приведены в следующей таблице:

Табл. 2. Типы данных языка FlexT

Table 2. The FlexT language data types

Тип	Пример	Описание
Целые числа	`num-(6)`	Характеризуются размером и наличием знака, для знаковых чисел используется дополнительный код.
Пустой	`Void`	Тип с размером 0, позволяет пометить место в памяти.
Символьные	`char, wchar, wcharr`	В выбранной кодировке или Unicode с разными порядками байтов
Перечислимый	`enum byte (A=1,B,C)`	Задаёт наименования константам базового типа данных

Перечисление термов	`enum TBit8 fields ( R0: TReg @0.3,...) of( rts(R0) = 000020_,...)`	Облегчает описание кодирования машинных команд, т.к. в основном для них используется этот приём: в составе целого числа выделяются битовые поля, использование которых определяется остальными битами числа
Множество	`set 8 of ( OLD ^ 0x02, ...)`	Даёт наименование битам, биты могут задаваться своим номером (символ `'='` после имени) или маской (символ `'^'`)
Запись	`struc Byte Len array[@.Len]of Char S ends`	Последовательное размещение именованных и, возможно, разнотипных составляющих в памяти
Вариант	`case @.Kind of vkByte: Byte else ulong endc`	Выбор типа содержимого по заданной в параметрах (внешней) информации
Проверка	`try FN: TFntNum Op: TDVIOp Endt`	Выбор типа содержимого по внутренней информации: выбирается первый тип, для которого выполняется условие корректности
Массив	`array[@.Len]of str array of str ?@[0]= 0!byte;`	Последовательное размещение однотипных составляющих в памяти (размеры которых могут различаться). Может задаваться количество элементов, общий размер массива, или стоп-условие.
Сырые данные	`raw[@.S]`	Неинтерпретируемые данные, отображаются как hex-дамп

Выравнивание	`align 16 at &@;`	Пропуск неиспользуемых данных для выхода на кратное заданному значению смещение относительно базового адреса
Указатель	`^TTable near=DWORD, ref=@:Base+@;`	Использует значение базового типа, для задания адреса (для файлов – смещения от начала) данных указанного типа в памяти.
Предварительное объявление	`Forward`	Используется при циклических зависимостях между типами данных
Машинные команды	`codes of TOpPDP ?(@.Op >=TWOpCode.br)and...;`	Используется для дизассемблирования машинных команд

Помимо рассмотренных конструкторов типов данных вызов типа с подстановкой фактических параметров на место формальных рассматривается как специальный конструктор типа и может сопровождаться своими блоками дополнительной информации, например, блоком отображения. Синтаксически вызов типа отличается от ссылки на тип по наличию круглых скобок после его имени (даже когда у типа нет параметров). При вызове типа параметры могут указываться либо позиционно, либо по имени (как при вызове методов интерфейсов COM). Таким образом, часть параметров вызова типа может задаваться при его определении, а другая часть – в блоке утверждений.

Все типы данных могут иметь как байтовое, так и битовое размещение, в зависимости от настроек блока типов. Кроме того, при определении нового типа учитывается установленный порядок байтов. Порядок байтов влияет на вновь определяемые типы, но не на их вызовы. В большинстве форматов используется конкретный порядок байтов, поэтому его достаточно установить один раз в начале спецификации, если это – MSB, а LSB не требует и этих действий, т.к. он установлен по умолчанию. Существуют такие форматы, как TIFF, где порядок байтов выбирается для всего файла в зависимости от его сигнатуры. Однако пример формата Shape показывает, что существуют и форматы со смешенным использованием разных порядков байтов.

3.6 Блоки определений

Программа на FlexT состоит из нескольких видов блоков определений, в которых определения разделяются переводом строки. Основные блоки определений:

1) Блок констант
   ```
   'const' nl
   {<Имя константы> '=' <Int Expr> ';', nl}*
   ```
   В выражениях для вычисления значений констант могут использоваться переменные из блока данных, поэтому значения некоторые из констант могут быть динамическими, т.е. зависящими от обрабатываемых данных.
2) Блок типов
   ```
   'type' ['bit'] nl
   {<Имя типа> <Определение типа>, nl}*
   ```
   содержит определения типов. Признак `bit` указывает на то, что все определяемые в этом блоке типы будут иметь битовое, а не байтовое размещение.
3) Блок данных
   ```
   'data' [<Блок>] nl
   {<Int Expr> [';'] <Определение типа> <Имя>, nl}*
   ```
   содержит определения переменных описываемого формата. `<Int Expr>` задаёт адрес переменной в гдавном адресном пространстве, если имя блока не указано, или смещение в блоке памяти `<Блок>`. Так же, как и при определении констант, выражение для адреса может содержать обращения к другим переменным, так что адрес переменной может определяться динамически. `<Определение типа>` должно задавать тип без свободных параметров, т.е. должны быть определены все свойства базового конструктора этого типа и типов его составляющих, если таковые имеются. После выражения для адреса можно ставить признак его окончания ';'. Это необходимо делать лишь в том случае, если начало <Определения типа> может быть воспринято, как продолжение выражения.
4) Блок кода
   ```
   'code' ['(' <Имя типа кода> ')'] [<Блок>] nl
   {<Int Expr> [';'] <Имя>, nl}*
   ```
   содержит начальные адреса и названия частей кода в блоке памяти `<Блок>` или в главном адресном пространстве, если имя блока не указано. При указании имени типа кода память, начиная с указанных точек входа, разбирается в соответствии с этим типом данных. Тип кода должен быть задан конструктором `CODES` и быть полностью

определённым, при этом он описывает кодирование машинных инструкций и признаки команд, завершающих код (таких как RET или JMP). Если тип кода не задан, то используется код процессоров Intel 80x86.

Специальное имя `'-'` означает, что указанный адрес задаёт не начало, а конец части кода. Эта возможность оказывается полезной в тех случаях, когда дизассемблер не может самостоятельно правильно определить окончание последовательности команд. Это может произойти, например, в следующем случае:

```
CALL Terminate
<Данные>
```

Здесь процедура Terminate всегда вызывает функцию API для завершения работы программы.

3.7 Условная компиляция

Для описания форматов данных, которые могут иметь вариации, зависящие от версии формата или других его особенностей удобно использовать условную компиляцию. В условиях для выбора вариантов кода могут использоваться выражения, зависящие от обрабатываемых данных, поэтому версии структур данных автоматически определяются по уже прочитанным переменным. Этот механизм является очень эффективным при использовании спецификаций на FlexT для описания сложных форматов с долгой историей. Он не увеличивает сложность структур данных интерпретатора в отличие от использования вариантных типов данных. Однако, если в спецификации используется условная компиляция, то её очень неудобно обрабатывать в отсутствии примера данных. Таким образом, роль условной компиляции для FlexT можно сравнить с ролью препроцессора для языка C: позволяет получить эффективный конечный результат, но усложняет анализ кода. Для описания большинства форматов данных условная компиляция не требуется, но в некоторых случаях её нечем заменить. Возможно, что для получения универсального кода чтения данных по спецификации, содержащей условную компиляцию, должны быть разработаны более сложные алгоритмы трансляции, добавляющие вариантные элементы в структуры данных и модифицирующие выражения со ссылками на такие вариантные составляющие.


```
data
  Ident:Size TEHdr Hdr

type
%$IF Hdr.e_machine=TEMachine.EM_386;
  include elf_386.rfi
%$ELSIF Hdr.e_machine=TEMachine.EM_SPARC;
  include elfSPARC.rfi
%$ELSIF Hdr.e_machine=TEMachine.EM_M32;
  include elf_M32.rfi
%$ELSIF Hdr.e_machine=TEMachine.EM_PPC;
  include elf_ppc.rfi
%$ELSE
type
  TE_R_TYPE byte
  TE_Machine_Flags EWord
%$END Machine
```

Рис. 1. Пример использования условной компиляции

Fig. 1. An example of usage of conditional compilation

На врезке приведён пример использования условной компиляции (выдержка из описания формата ELF – загрузочных и объектных модулей Unix). Здесь в зависимости от типа процессора подключается соответствующая спецификация перечислимого типа `TE_R_TYPE` и множества `TE_Machine_Flags`, используемых в описании таблицы перемещений.

3.8 Алгоритм работы интерпретатора

Процесс разбора содержимого бинарного файла по спецификации интерпретатором FlexT состоит из следующих шагов:

1) Отображение (file mapping) бинарного файла в память.
2) Чтение спецификации, в ходе которого создаются объявленные в блоках `data` переменные и запоминаются адреса кода из блоков `code` (создаются части кода с нулевой длиной). Ранее объявленные переменные могут использоваться в зависящих от данных выражениях спецификации. Если в этих выражениях используется разыменование указателей, то на процесс чтения могут влиять и отличные от объявленных переменных элементы данных, доступные через эти указатели.
3) Обход элементов данных в поисках указателей. Выполняется только для элементов данных тех типов, в состав которых указатели входят. При обнаружении указателя создаётся переменная того типа, на который ссылается этот указатель, если такая переменная не была

создана раньше. После этого выполняется обход составляющих новой переменной. Кроме того, если структуры данных содержат указатели на код, то обнаруженные адреса добавляются в список частей кода.

4) Статическое дизассемблирование кода. Для каждой ещё не обработанной части кода выполняется обход её команд. Обход заканчивается по достижении, либо команды с выполненным стоп-условием (например, `ret` или `jmp`), либо следующей части кода. При обходе команды проверяются на наличие переходов и ссылок на данные. Для обнаруженных переходов (ссылок на код) также создаются новые части кода, если они не попадают внутрь существующих, иначе существующая часть кода разбивается на две. В результате формируется список частей кода, каждая из которых содержит непрерывную последовательность команд без внутренних точек входа.

5) Вывод результатов. Выполняется обход с распечаткой содержимого обнаруженных переменных и частей кода в порядке их адресов. Оставшиеся неразобранными блоки памяти между последовательными переменными и частями кода распечатываются в виде шестнадцатеричного дампа (при желании отображение этих фрагментов памяти можно отключить).

4. Использование спецификаций для анализа машинного кода

В предшествующих главах уже упоминались типы данных и шаги алгоритма разбора, предназначенные для работы с машинными командами. Рассмотрим эти возможности языка FlexT в одном месте и более подробно. Машинные команды используются не только для файлов с исполняемым кодом для реальных процессоров (таких, как Intel 80x86, ARM), но и для виртуальных машин (Java VM, .NET MSIL). Кроме того, некоторые форматы, например, шрифты TTF или файлы инсталляторов, могут содержать байт-код для бинарного представления используемых там сценариев. Таким образом, исследование файлов многих форматов будет неполным без отображения содержащихся там машинных инструкций.

В большинстве случаев представление машинной команды состоит из постоянной части, задающей вид команды и переменной части, задающей её аргументы, если таковые имеются. В качестве описания постоянной части для байт-кодов виртуальных машин, как правило, достаточно использовать перечислимый тип данных, поскольку там просто содержится целочисленный код команды (формат TTF является исключением из этого правила), при этом переменную часть удобно описать при помощи вариантного типа по коду команды.

Для машинных команд реальных процессоров используется более плотное кодирование, поэтому в постоянной части для некоторых команд выделяются битовые поля, содержащие часть аргументов, например, номера регистров. Будем называть сочетания таких полей с кодом инструкции термами. Описать кодирование термов можно было бы при помощи типа данных «проверка» и ряда вспомогательных битовых типов, но такое описание будет очень громоздким и неэффективным (при попытке сделать это на практике терпения хватило лишь на описание десятка команд). Поэтому, под влиянием проекта [25][26] в язык FlexT включён специальный тип данных «перечисление термов». В определении такого типа сначала задаётся список всех битовых полей, применяемых для кодирования команд (каждое поле характеризуется битовыми смещением, размером и типом), а затем описываются сами термы с указанием того, какие из этих полей они используют. Вместе с термом указывается его база – число, задающее значения не входящих в поля битов. В записи базы терма допускается использовать подчерки для обозначения переменной части — битов, входящих в состав полей (такая маска проверяется на соответствие упоминаемым в терме полям). В реализации типа данных «перечисление термов» используется дерево решений, позволяющее быстро найти терм по его коду. В выражениях можно обращаться к любым полям перечисления термов, как к полям записи, однако, если у выбранного терма это поле не используется, то такое обращение вызывает ошибку вычисления этой части выражения.

Если системы команд RISC-процессоров состоят только из постоянной части, то для описания команд CISC-процессоров требуется задавать кодирование переменной части в зависимости от терма, выбранного в перечислении термов. Для этих целей реализована разновидность типа данных «вариант» с выбором содержимого по значению перечисления термов. В определении такого типа база терма в списке выбора обозначает все соответствующие этому терму значения постоянной части. Для выбора варианта в этом случае также строится дерево решений, но лишь по тем термам, которые упоминаются в определении типа.

После описания кодирования отдельной команды необходимо определить кодовый тип. В интерпретаторе FlexT кодовый тип реализуется на базе массива со стоп-условием и записывается аналогичным образом, за исключением использования идентификатора `codes` вместо `array`. При этом стоп-условие используется для выделения команд, завершающих последовательность, например, команд безусловного перехода или выхода из процедуры. В описаниях команд, вызывающих переход, должны использоваться указатели на определяемый кодовый тип, которые будут отслеживаться дизассемблером для обнаружения других частей кода. Кроме того, у типа команд может быть определено, как в приведённом примере, вычисляемое свойство `isCall`, которое в этом случае используется дизассемблером для различения команд перехода и вызовов процедур (влияет на префиксы генерируемых меток).

```
type bit
TBit num+(1)
TBit2 num+(2)
…
TsBit24 num-(24)
TRN enum TBit4 (R0,R1,… R12,SP,LR,PC)
…
TShiftOp enum TBit2 (SHL,SHR,ASR,ROR)
TBrOfs24 ^TOpARMSeq NIL- =TsBit24 REF=
  (&(@:@)+4+@*4);
type
TARMOpCode enum TBit32 fields (
  Cond: TCond @28.4, //Op.execution condition
  Rn: TRN @16.4, //Source register
  Rd: TRN @12.4, //Destination register
  Shift: TShiftOp @5.2,
…
  SM: TBit @20.1
) of (
 //Data processing rotate right with extend
  _DPRRX(cond,ArOp,S,Rn,Rd,Rm) =
    0b____000_I________I____0000I0110____,
 //Move status register to register
  MSR(cond,R,Rd) =
    0b____0001I0_001111I____0000I00000000,
  …
 //Branch and Branch with link
  Branch(cond,BL,BrOfs) =
    0b____101_I________I________I________,
 //Software interrupts
  SWI(cond,SWIOfs) =
    0b____1111I________I________I________
):let IsStop=(((@.BL=0)and(@.cond>=
  TCond.AL)) exc 0
);:let isCall=((@.BL=1) exc 0);
TOpARMSeq codes of TARMOpCode ?@:isStop;:
  displ=('(', ShowArray(@,(NL,HEX(&@,4),': ',
  @)),NL,')')
```

Рис. 2. Фрагменты спецификации кодирования команд процессора ARM

Fig. 2. Excerpts from the instruction encoding specification of the ARM processor

На врезке приведены фрагменты спецификации кодирования команд процессора ARM. Многоточием обозначены пропущенные фрагменты, аналогичные оставшимся соседним. Сначала определяются битовые типы данных, используемых в командах полей, включая перечислимые типы для декодирования значений тех из этих полей, которые задают регистры, конкретные операции из семейства однотипных, условия перехода и т.д. Также на базе типа 24-битных знаковых чисел определяется тип указателя `TBrOfs24`, используемый в командах перехода. В выражении для вычислении адреса, на который ссылается указатель, в качестве базы используется адрес владельца

(`&(@:@)`), т.е. перечисления термов, в состав которого входит указатель. Далее определяются перечисление термов `TARMOpCode`, описывающее кодирование отдельной команды и кодовый тип данных `TOpARMSeq` на базе этих команд.

Описания кодирования машинных команд позволяют использовать разработанные инструменты анализа бинарных файлов для новых процессоров. Использование описаний кодирования машинных команд, унифицированных с описаниями остальных типов данных, позволяет применять для исследования команд все средства, разработанные для анализа данных. Например, можно воспользоваться механизмами поиска структур данных для обнаружения фрагментов кода заданного вида и получения отчёта по найденным адресам. Имеется опыт применения этого подхода для получения описаний типов данных интерпретатора посредством поиска всех вызовов процедуры регистрации типа в коде исполняемого файла интерпретатора. В некоторых случаях непосредственные аргументы команд могут интерпретироваться, как указатели, что позволяет выделять в памяти соответствующие переменные. С использованием блока отображения для типа данных команды можно при необходимости довести способ отображения команды до принятого в дизассемблерах, а можно добавить вывод дополнительных сведений о её свойствах или составляющих.

С другой стороны, во многих исполняемых файлах присутствуют структуры данных, например RTTI или таблицы виртуальных методов, содержащие адреса кода. Описание таких элементов данных на FlexT как указателей на код позволяет обнаружить этот код и пометить его как относящийся к определённым данным, что существенно облегчает дальнейшее понимание программы.

5. Применение основанных на FlexT инструментов для анализа бинарных файлов

Для работы с бинарными данными реализован ряд инструментов, основанных на использовании написанных на FlexT спецификаций: BinView, BinExpl, ExeXpl, а также Web-приложение, доступное по ссылке [27].

Консольная программа BinView позволяет получить результат разбора бинарного файла в одном из текстовых форматов: просто текст, HTML, RTF, TeX. Формат обрабатываемого файла определяется автоматически по его расширению и содержимому. Для автоматического поиска спецификации формата используется следующая логика: проверяется файл с именем <Расширение>`.rfh`, если таковой имеется в текущем каталоге или в каталоге библиотеки спецификаций. Далее проверяются все описания форматов, сопоставленные расширению в файле `ref.cfg` из библиотечного каталога. Для проверки соответствия обрабатываемого файла некоторой спецификации используются блоки утверждений `assert` из файла спецификации: заданные в этих блоках логические выражения оцениваются сразу после чтения и, если они

оказываются ложными или ошибочными, то попытка применить спецификацию прекращается. Таким образом, если, например, блок утверждения находится сразу после объявления переменной для сигнатуры файла и значение этой переменной не соответствует ожидаемому, то на этом чтение спецификации заканчивается. В результате выбирается первая спецификация, для которой все утверждения выполняются. После чтения спецификации формата выполняется поиск дополнительной спецификации структур данных обрабатываемого файла, которая может находиться в файле <Имя файла>`.ref`. Таким образом, при необходимости можно описать структуры данных, обнаруженные в конкретном файле, но не отражённые в общей спецификации формата.

Сгенерированный программой BinView листинг может оказаться очень большим. В результате, например, браузер будет очень долго открывать полученный HTML файл. Программа BinExpl служит для интерактивного отображения результатов разбора бинарных файлов, по тем же спецификациям, что и BinView. При этом динамически генерируются фрагменты листинга, что позволяет просматривать содержимое очень больших файлов (основное ограничение на размер: файл должен целиком помещаться в 32-разрядное пространство адресов при использовании FileMapping).

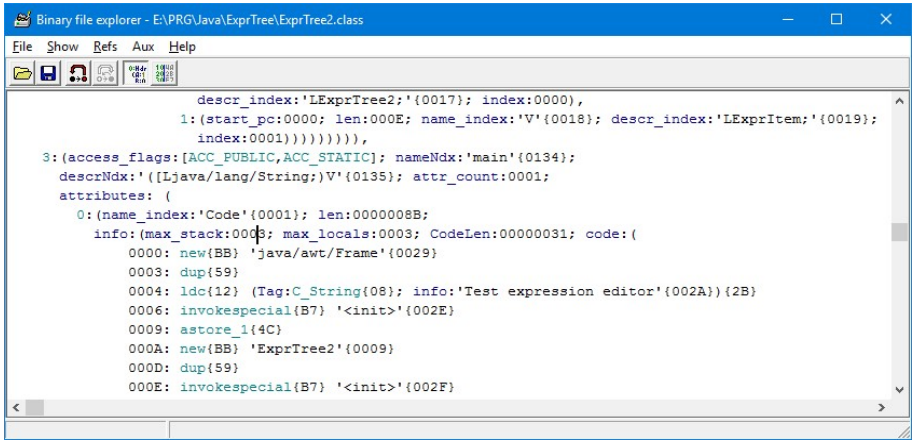

Рис. 3. Окно программы BinExpl при разборе файла класса Java

Fig. 3. The main form of the program BinExpl when parsing a Java class file

Программа ExeXpl служит для интерактивного исследования исполняемых файлов Windows. При этом спецификации на FlexT используются для описания структур данных исследуемых программ и, в том числе, структур данных, характерных для конкретного компилятора (RTTI, таблиц виртуальных методов, обработчиков ошибок и т.д.). В ходе изучения программы пользователь может добавлять в спецификацию наименования для

исследованных частей кода, что облегчает понимание тех фрагментов кода, которые используют уже описанные. Т.к. код выделяется по результатам статического анализа, не могут быть автоматически обнаружены, например, фрагменты, на которые ссылаются таблицы перехода, сгенерированные для операторов выбора (switch/case). Но такие таблицы могут быть описаны, как содержащие указатели на код структуры данных, что позволяет обнаружить оставшиеся неразобранными фрагменты кода.

С использованием спецификаций на FlexT реализован ряд механизмов поиска, которые невозможно выполнить другим способом: поиск структур данных, генерация кода по сценариям поиска данных, поиск элементов данных в файлах.

Поиск структур данных позволяет обнаружить фрагменты памяти, соответствующие заданному условию на выбранный пользователем тип данных. В ходе такого поиска алгоритм проверяет каждый адрес на возможность разместить в этом месте указанную структуру данных так, чтобы она отвечала заданному критерию, например, условию корректности `assert` искомого типа данных. Таким образом можно, например, найти все таблицы виртуальных методов классов. Этот процесс поиска можно оформить в виде сценария генерации отчётов об обнаруженных структурах данных. Для применения таких сценариев необходимо подключить к спецификации модули с определениями используемых при поиске типов данных, после чего в процессе поиска будет сформирован текстовый отчёт обо всех обнаруженных адресах в заданной в сценарии поиска форме (для этого применяется синтаксис блоков отображения типов). Этот механизм может использоваться для генерации фрагмента спецификации с описаниями найденных данных, например, на базе информации о генерируемых конкретным компилятором структурах данных. Также приходилось использовать этот механизм для получения информации о реализованных в интерпретаторе языка сценариев классах с информацией об их членах.

Механизм поиска элементов данных в файлах позволяет найти среди файлов с указанным расширением те, которые содержат данные указанного типа, отвечающие заданному условию. Здесь поиск идёт не по всей памяти, а среди составляющих переменных, найденных в файле при помощи спецификации. Этот механизма позволяет, например, найти метафайлы, в которых содержатся команды рисования окружности определённого размера.

6. Заключение

В данной работе рассмотрен язык спецификации интерпретации данных FlexT, который позволяет описывать достаточно широкий набор форматов данных при помощи простых синтаксических конструкций, являющихся расширением характерного для традиционных процедурных языков программирования набора конструкторов типов данных. Возможно также его использование для

спецификации кодирования машинных команд. Реализованный интерпретатор использует спецификации для идентификации типов данных.

На языке FlexT с различной степенью завершённости было описано около сотни форматов данных, в том числе, исполняемых и объектных файлов, содержащих программный код. Приведём расширения некоторых из этих форматов: EXE (MZ,NE,LE), ELF, CLA, TPU, OBJ, Mach-o, TTF, HLP, SHP, DBF. Также конструкции языка использовались при декодировании различных программ и описании характерных для конкретных компиляторов форматов данных, например, RTTI Delphi и Visual Studio. Многие форматы удавалось описать практически с одного прохода, т.е. по мере чтения документации получалось сразу переводить содержащиеся там сведения в конструкции языка FlexT. При работе с описаниями форматов в редком из них не были обнаружены ошибки. Т.е. переход от описания для человека к описанию для машины, которое можно сразу же проверить на настоящих данных, приводит к существенному повышению достоверности информации, и в этом может состоять один из наиболее важных результатов применения рассматриваемого языка. Возможность интерпретации результатов разбора в качестве теста на соответствие спецификации и реальных данных, может применяться, как для проверки корректности данных по уже отлаженной спецификации, так и для проверки спецификации на соответствие примерам данных и, в том числе, для обратного проектирования недокументированных форматов.

Наиболее существенным ограничением для текущей версии языка является невозможность полного описания таких форматов, при интерпретации которых необходимо строить сложные вспомогательные структуры данных, непосредственно не представленные в файле. Например, такая необходимость часто возникает при описании сжатых данных, при декомпрессии которых используется динамическое построение словарей (как LZW), дерева Хаффмана, моделей контекстов (PPM) и других подобных структур. Более подробную информацию о языке FlexT можно найти по адресу [27].

Список литературы

[1]. Faase F.J. BFF: A grammar for Binary File Formats [Электронный ресурс] URL: http://www.iwriteiam.nl/Ha_BFF.html

[2]. Data Format Description Language (DFDL) [Электронный ресурс] URL: https://www.ogf.org/ogf/doku.php/standards/dfdl/dfdl

[3]. IBM Knowledge Center [Электронный ресурс] Data Format Description Language (DFDL) URL: http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/df20060_.htm

[4]. IBM Integration Bus [Электронный ресурс] URL: http://www-03.ibm.com/software/products/en/ibm-integration-bus/

[5]. Daffodil: Open Source DFDL [Электронный ресурс] URL: https://opensource.ncsa.illinois.edu/confluence/display/DFDL

[6]. IBM Knowledge Center [Электронный ресурс] Unsupported features URL: http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/df00150_.htm

[7]. WebLogic Integration 7.0 [Электронный ресурс] Building Format Definitions. URL: https://docs.oracle.com/cd/E13214_01/wli/docs70/diuser/fmtdef.htm

[8]. NetPDL Language Specification. http://www.nbee.org/doku.php?id=netpdl:index

[9]. BinPAC. https://www.bro.org/sphinx/components/binpac/README.html

[10]. The data description language EAST specification (CCSD0010). [Электронный ресурс] URL: http://mtc-m16c.sid.inpe.br/col/sid.inpe.br/mtc-m18@80/2009/07.21.13.31/doc/CCSDS%20644.0-B-2.pdf

[11]. Calder B.R., Masetti G. Huddler: a multi-language compiler for automatically generated format-specific data drivers. U.S. Hydrographic Conference (US HYDRO) 2015 Доступно по ссылке: http://www.hypack.com/ushydro/2015/papers/pdf/Calder_Huddler_for_automatic_data_drivers.pdf

[12]. Georgia Tech Research Institute [Электронный ресурс] Digital Archives Research URL: http://perpos.gtri.gatech.edu/

[13]. Underwood W. Grammar-Based Specification and Parsing of Binary File Formats. The International Journal of Digital Curation Vol. 7, No. 1, 2012, pp. 95-106 Доступно по ссылке: http://www.ijdc.net/index.php/ijdc/article/viewFile/207/276

[14]. Parr T. ANTLR (ANother Tool for Language Recognition) [Электронный ресурс] URL: http://www.antlr.org/

[15]. Godmar Back. 2002. DataScript - A Specification and Scripting Language for Binary Data. In Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering (GPCE '02), Don S. Batory, Charles Consel, and Walid Taha (Eds.). Springer-Verlag, London, UK, UK, 66-77.

[16]. DataScript [Электронный ресурс] URL: http://datascript.sourceforge.net/

[17]. Binary data definition language [Электронный ресурс] URL: http://www.binarydom.com/sdk/doc/bddl.shtml

[18]. Binopedia [Электронный ресурс] URL: http://binopedia.org/

[19]. Kaitai Struct [Электронный ресурс] URL: http://kaitai.io/

[20]. Synalyze It! [Электронный ресурс] URL: https://www.synalysis.net/

[21]. Hexinator [Электронный ресурс] URL: https://hexinator.com/hexinator-windows/

[22]. Synalyze It! [Электронный ресурс] The Grammar Page. https://www.synalysis.net/formats.xml

[23]. Лисков Б., Гатэг Дж. Использование абстракций и спецификаций при разработке программ: Пер. с англ. - М.:Мир, 1989.

[24]. Филд А., Харрисон П. Функциональное программирование: Пер. с англ. - М.:Мир, 1993

[25]. Ramsey N., Fernandez M.F. 1995. The New Jersey machine-code toolkit. In Proceedings of the USENIX 1995 Technical Conference Proceedings (TCON'95). USENIX Association, Berkeley, CA, USA, 24-24.

[26]. Ramsey N., Fernandez M.F. The New Jersey Machine-Code Toolkit [Электронный ресурс] URL: http://www.cs.tufts.edu/~nr/toolkit/

[27]. Хмельнов А.Е. Главная страница по языку FlexT. http://hmelnov.icc.ru/FlexT/

A declarative language FlexT for analysis and documenting of binary data formats

A.Y. Hmelnov <hmelnov@icc.ru>
I.V. Bychkov <bychkov@icc.ru>
A.A. Mikhailov <mikhailov@icc.ru>
Matrosov Institute for System Dynamics and Control Theory of the Siberian Branch of the Russian Academy of Sciences,
134, Lermontova st., Irkutsk, 664033, Russia

Abstract. The language FlexT (Flexible Types) is intended for specification of binary data formats. The language is declarative and designed to be well understood for human readers. Its main elements are the data type declarations, which look very much like the usual type declarations of the imperative programming languages, but are more flexible. In the article we first give a review of the capabilities of the modern projects oriented to specification of binary file formats. Then we consider the main features of the FlexT language and, in particular, the features that help to describe the formats of encoding of machine instructions. Finally we briefly describe the software developed, which is based upon the FlexT interpreter and some new capabilities of information search, which makes possible the use of the specifications.

Keywords: specifications of binary data formats, specification of encoding of machine instructions, declarative language, disassembler

DOI: 10.15514/ISPRAS-2016-28(5)-15

For citation: 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

References

[1]. Faase F.J. BFF: A grammar for Binary File Formats. http://www.iwriteiam.nl/Ha_BFF.html
[2]. Data Format Description Language (DFDL). https://www.ogf.org/ogf/doku.php/standards/dfdl/dfdl
[3]. IBM Knowledge Center. Data Format Description Language (DFDL). http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/df20060_.htm
[4]. IBM Integration Bus. http://www-03.ibm.com/software/products/en/ibm-integration-bus/
[5]. Daffodil: Open Source DFDL. https://opensource.ncsa.illinois.edu/confluence/display/DFDL
[6]. IBM Knowledge Center. Unsupported features. http://www.ibm.com/support/knowledgecenter/SSMKHH_10.0.0/com.ibm.etools.mft.doc/df00150_.htm

[7]. WebLogic Integration 7.0. Building Format Definitions. https://docs.oracle.com/cd/E13214_01/wli/docs70/diuser/fmtdef.htm
[8]. NetPDL Language Specification. http://www.nbee.org/doku.php?id=netpdl:index
[9]. BinPAC. https://www.bro.org/sphinx/components/binpac/README.html
[10]. The data description language EAST specification (CCSD0010). http://mtc-m16c.sid.inpe.br/col/sid.inpe.br/mtc-m18@80/2009/07.21.13.31/doc/CCSDS%20644.0-B-2.pdf
[11]. Calder B.R., Masetti G. Huddler: a multi-language compiler for automatically generated format-specific data drivers. U.S. Hydrographic Conference (US HYDRO) 2015 Available at URL: http://www.hypack.com/ushydro/2015/papers/pdf/Calder_Huddler_for_automatic_data_drivers.pdf
[12]. Georgia Tech Research Institute. Digital Archives Research. http://perpos.gtri.gatech.edu/
[13]. Underwood W. Grammar-Based Specification and Parsing of Binary File Formats. The International Journal of Digital Curation Vol. 7, No. 1, 2012, pp. 95-106 Available at URL: http://www.ijdc.net/index.php/ijdc/article/viewFile/207/276
[14]. Parr T. ANTLR (ANother Tool for Language Recognition). http://www.antlr.org/
[15]. Godmar Back. 2002. DataScript - A Specification and Scripting Language for Binary Data. In Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering (GPCE '02), Don S. Batory, Charles Consel, and Walid Taha (Eds.). Springer-Verlag, London, UK, UK, 66-77.
[16]. DataScript. http://datascript.sourceforge.net/
[17]. Binary data definition language. http://www.binarydom.com/sdk/doc/bddl.shtml
[18]. Binopedia. http://binopedia.org/
[19]. Kaitai Struct. http://kaitai.io/
[20]. Synalyze It!. https://www.synalysis.net/
[21]. Hexinator. https://hexinator.com/hexinator-windows/
[22]. Synalyze It! The Grammar Page. https://www.synalysis.net/formats.xml
[23]. B. Liskov, J. Guttag, Abstraction and Specification in Program Development, The MIT Press, 1986.
[24]. Field A.J., Harrison P.G. Functional Programming, Addison-Wesley, Wokingham, UK, 1988
[25]. Ramsey N., Fernandez M.F. 1995. The New Jersey machine-code toolkit. In Proceedings of the USENIX 1995 Technical Conference Proceedings (TCON'95). USENIX Association, Berkeley, CA, USA, 24-24.
[26]. Ramsey N., Fernandez M.F. The New Jersey Machine-Code Toolkit. http://www.cs.tufts.edu/~nr/toolkit/
[27]. Hmelnov A.Y. The home page of FlexT. http://hmelnov.icc.ru/FlexT/