Российская академия наук
Сибирское отделение
Институт систем информатики
им. А. П. Ершова

С. В. Каличкин

ОБЗОР СРЕДСТВ СТАТИЧЕСКОЙ ОТЛАДКИ ПРОГРАММ

Препринт
112

Новосибирск 2004

Статический отладчик — это инструмент, который анализирует и визуализирует информацию, накопленную в результате статического анализа программы, для локализации ошибок в программе. Целью настоящей работы является обзор статических отладчиков по следующим аспектам: виды используемых анализов программы, особенности отображения информации о программе, ее визуализации и средств навигации по ней. Поскольку удалось обнаружить лишь два статических отладчика, в обзоре также рассматриваются другие виды инструментов: динамические отладчики, статические анализаторы и средства визуализации данных.

Siberian Division of the Russian Academy of Sciences
A. P. Ershov Institute of Informatics Systems

Stepan V. Kalichkin

A REVIEW OF TOOLS FOR STATIC DEBUGGING OF PROGRAMS

Preprint
112

Novosibirsk 2004

Static debugger is a tool which analyzes and visualizes information collected as a result of the static analysis of program in order to localize errors in it. The purpose of this work is to review static debuggers on the following aspects: kinds of program analyses, specific features of displaying the program information, visualization and navigation. As it was possible to find out only two static debuggers other tools are also considered in this review: dynamic debuggers, static analyzers and data visualization tools.

Статический анализатор ошибок Wasp [1] базируется на мощном и сложном потоковом анализе исходной программы. Генерируемые им сообщения об ошибках часто оказываются сложными для пользователя. В связи с этим назрела необходимость в разработке инструмента нового вида — статического отладчика, который в диалоговом взаимодействии с пользователем обеспечивает адекватную визуализацию разнообразной информации, полученной в результате потокового анализа программы. Целью настоящей работы является обзор статических отладчиков в следующих аспектах: виды используемых анализов программы, а также особенности отображения информации о программе, ее визуализации и средств навигации по ней. При изучении литературы обнаружилось, что термин статический отладчик не является общепринятым и встречается лишь однажды. В связи с этим в данной работе представленные аспекты рассматриваются в других инструментах: динамических отладчиках, статических анализаторах и средствах визуализации данных.

ВВЕДЕНИЕ

Разработка, отладка, модификация и сопровождение больших программных систем является весьма сложной задачей. Для поддержки разработки больших программ требуются качественные многофункциональные отладчики. Существует огромное количество разнообразных отладчиков, приспособленных под различные применения. Основная цель всех отладчиков — облегчить локализацию всевозможных ошибок в программе.

Состояние программы определяется текущим исполняемым оператором и данными программы в момент исполнения текущего оператора. Традиционные системы отладки обеспечивают три основные функции:

- выбор состояния программы,
- вывод состояния программы,
- модификация состояния программы.

Выбор состояния программы состоит в поиске и идентификации пользователем интересующих его данных и программных событий и, в частности, включает в себя такие действия, как установка узлов и трассировка программы. Вывод состояния программы предполагает выдачу по запросам пользователей содержимого различных переменных и массивов или состояний программного процесса (т.е. потоков управления и данных). Модификация состояния программы заключается в изменении состояния программного процесса.

Для локализации ошибок в программах применяют статические и динамические отладчики. Принципы этих отладчиков существенно различаются. В настоящее время в основном применяются динамические отладчики. Динамический отладчик обеспечивает анализ программы в процессе ее исполнения. Например, существуют такие развитые динамические отладчики, как SoftIce NuMega [2], gdb [3], jdb [4] и многие другие. Статический отладчик — это инструмент, который использует информацию, накопленную в результате статического анализа программы (реализуемого без ее исполнения), и на основе этой информации позволяет локализовать ошибки в программе, а также недобротности [5], возникающие при нарушении любого из критериев разумно организованной программы. Одним из видов статического анализа является потоковый анализ, применяемый в статическом анализаторе Wasp для программ на языках Oberon-2, Modula-2 и Java [1]. В настоящей работе рассматривается два статических отладчика MrSpidey [6] и Syntox [7, 8], а также система статического анализа ошибок PolySpace [9, 10] с развитыми средствами визуализации и навигации, представляющими интерес при разработке аналогичных средств в статических отладчиках.

Развитые средства визуализации программ и данных значительно повышают возможности отладки. Визуальные отладчики могут обеспечить комплексное представление структуры программ и их компонентов, в том числе текста программы, связей между модулями и обменов информацией между ними, структур данных и содержимого отдельных элементов этих структур.

СРЕДСТВА ДИНАМИЧЕСКОГО АНАЛИЗА И ОТЛАДКИ

Работа динамических отладчиков в основном базируется на установке в программе контрольных точек останова различных видов. Реализация этой возможности может поддерживаться на уровне центрального процессора, микроконтроллера, операционной системы. Также могут использоваться отладчики-симуляторы и внутрисхемные эмуляторы. В любом случае отладка происходит во время исполнения программы, поэтому такие отладчики также называются отладчиками периода исполнения. Динамический отладчик позволяет выполнить основные четыре функции:

- запуск программы;
- установка условных и безусловных точек останова;
- отображение данных, использующихся в отлаживаемой программе;
- внесение изменений в данные программы.

Отладчики типа debug и afdPro [11] под управлением операционной системы MS-DOS умели работать только на уровне языка машинных инструкций, т.е. ассемблера. Такие отладчики отображали лишь содержимое памяти по заданным адресам в шестнадцатеричном, двоичном или восьмеричном виде, содержимое регистров центрального процессора и ассемблерный код исходной программы. Естественно, что такие отладчики сложно применять для больших программ, написанных на языках высокого уровня.

Затем появились отладчики, работающие в терминах языка высокого уровня. Для этого они используют дополнительную информацию о программе, которую предоставляет компилятор. Эту информацию называют символьной (отладочной) информацией. Практически все компиляторы и ассемблеры в том или ином виде генерируют символьную информацию. Отслеживание процесса выполнения программы производится отладчиком по ее исходному тексту. В общем случае одна строка исходного текста преобразуется компилятором в несколько машинных команд. Отладчик отображает на экране исходный текст программы и, используя таблицу соответствия номеров строк исходного текста абсолютным адресам кода программы, может выполнять программу «по строкам», исполняя за один шаг все машинные команды, сгенерированные компилятором для текущей строки. Таблица номеров строк позволяет также производить контекстные действия с текстом программы, например, выполнять программу «до курсора», т.е. до указанного пользователем места в исходном тексте ставить точки останова на указанные строки и т.п. Контекстные действия удобны тем, что разработчики могут оперировать с исходным текстом и данными программы, абстрагируясь от машинных команд и регистров процессора. Одним из таких отладчиков является отладчик системы разработки программного обеспечения XDS [20] для языков Модула-2 и Оберон-2. В отладчики встраивают мощные вычислители выражений, с помощью которых предоставляется возможность при отладке программы работать с выражениями языка программирования в привычном для него виде. Так же можно работать с логическими объектами программы. Отладчики осуществляют удобную навигацию по программе. Например, если при отладке исследуется поведение функции, код которой содержится в другом модуле, отладчик автоматически находит нужный модуль и отображает исходный текст. Современные динамические отладчики также знают адреса подпрограмм, функций и меток кода и умеют находить соответствующие модули и исходный текст функции по ее имени. При обращении к библиотеке динамического связывания (DLL) отладчик XDS, например, динамически создает всю необходимую информацию о ее структуре, что позволяет работать с

программами, часть кода которых находится в библиотеках DLL. Отладчик SoftICE [2] позволяет устанавливать точки останова по имени подпрограммы не только для отлаживаемой программы, но и для системных вызовов ОС.

Для полноценной отладки разработчику необходимо иметь возможность в любой момент просмотреть данные, которыми манипулирует программа. Отладчики умеют отображать любые используемые программой данные в наиболее подходящем виде. Объекты могут быть разной сложности — от простых типов данных до сложных конструкций языков высокого уровня типа структур, массивов и т.п. Обладая указанной выше информацией, отладчик, получив от пользователя имя объекта, определяет его тип, в соответствии с которым производится визуализация объекта, при этом имеется возможность редактировать его содержимое. Многие отладчики для визуализации объектов используют графическое представление, но некоторые делают это в текстовом (gdb [3]). Текстовый вариант, в частности, незаменим при удаленной отладке через низкоскоростные соединения.

Вышеперечисленные функции реализуют большинство современных отладчиков — gdb, jdb и все отладчики интегрированных сред разработки (IDE), такие как интегрированный отладчик Delphi [12], отладчик C++ Builder и другие. Существует также отдельный вид динамических отладчиков — интерфейсные. Наиболее яркий представитель таких отладчиков — это DDD [13]. Такие отладчики предоставляют общий интерфейс для подчиненного отладчика. Обычно подчиненные отладчики управляются из командной строки.

Для отладки исполняемых двоичных файлов можно использовать DDD с любым из подчиненных отладчиков GDB, DBX (используется в большинстве коммерческих Unix систем), Ladebug (на платформах Tru64 и Linux/Alpha), XDB (на старых HP-UX системах). Для отладки байт-кодовых Java-программ можно использовать DDD с JDB, отладчиком Java, который поставляется с JDK 1.1 или более поздними версиями. Для отладки программ на языке Python [14] можно использовать DDD с PYDB — отладчиком для Python. Для отладки программ на Perl [15] можно использовать DDD с отладчиком Perl из Perl 5.003 и более поздних версий.

Основная функция интерфейсных отладчиков — это обеспечить унифицированное управление подчиненными отладчиками, а также отображение информации, навигацию по программе и визуализацию объектов в удобном виде, независимо от подключенного подчиненного отладчика.

Несколько специфичный тип отладки — удаленная отладка. Удаленная отладка позволяет использовать ресурсы некой удаленной системы при

изучении поведения некоторого процесса в целевой системе, т.е. в системе, которая непосредственно исполняет отлаживаемую программу. В общем случае удаленный отладчик состоит из двух основных модулей: менеджера на удаленной платформе и агента отладки на целевой стороне. Менеджер служит для обеспечения пользовательского интерфейса, т.е. для приема команд, их обработки и посылки на целевую сторону, а также для приема, обработки и выдачи информации от агента, который осуществляет непосредственную работу с отлаживаемой системой. Такую отладку может осуществлять gdb [3], например, при использовании целевой системы VxWorks. Удаленная отладка применяется в случае нехватки ресурсов для одновременной работы отладчика и отлаживаемой программы на одной машине.

Современные динамические отладчики позволяют диагностировать и локализовать довольно обширный круг ошибок. Но существуют проблемы, для решения которых нет общего подхода и которые решаются плохо либо не решаются вообще существующими отладчиками. Отладчики не позволяют выявить ошибки планирования, синхронизации и связи в многопоточных и многопроцессных задачах. Неправильная синхронизация потоков порождает трудноуловимые "плавающие" ошибки, спонтанно проявляющиеся с некоторой (возможно пренебрежительно малой) вероятностью. Для локализации таких ошибок применяется лишь мониторинг — сбор данных о ходе выполнения программы с минимальным вмешательством в работу целевой системы и без остановки отлаживаемой задачи и модификации ее данных. В результате сложно локализовать такие ошибки, которые приводят к состоянию “гонок” — несогласованного обмена данными между потоками и взаимоблокировок (deadlock) в программе. Ошибки, связанные с тем, что данные задачи были изменены другой задачей, также сложно локализовать с помощью отладчиков. Есть попытки построения отладчиков для программ реального времени в многозадачных системах и для распределенных систем. Но проблемы асинхронного характера приложений, связи между процессорами, взаимодействия задач, выполняющихся на разных процессорах, далеки от окончательного разрешения.

СРЕДСТВА СТАТИЧЕСКОЙ ОТЛАДКИ

С помощью традиционных средств тестирования и динамической отладки принципиально невозможно обнаружить все ошибки в программе. Альтернативный подход для нахождения ошибок — это статический анализ

программ. Статический анализ не требует входных данных для программы и производится без ее исполнения. Статический анализ может находить лишь ошибки динамической семантики языка программирования. Ошибки статической семантики обнаруживает компилятор.

Одним из видов такого анализа является потоковый анализ. Ошибки, обнаруживаемые на базе потокового анализа, определяются как нарушения потоковых связей программы. Одним из первых статических анализаторов ошибок, базирующихся на простом потоковом анализе, является lint [16] для программ на языке С. Это довольно простой анализатор, который определяет конструкции, содержащие ошибки статической семантики и синтаксиса языка программирования. lint более строго, чем С-компилятор, контролирует правила типизации. Среди обнаруживаемых семантических ошибок — недостижимые операторы; циклы, в которые входят не с начала; описанные, но не используемые автоматические переменные; логические выражения с константными значениями. Кроме того, проверяется использование функций, и обнаруживаются функции, возвращающие значения в одних местах, но не возвращающие в других; функции, вызываемые с различным числом аргументов или с аргументами разных типов; функции, значения которых не используются, и функции, значения которых не возвращаются, но используются. Кроме lint сейчас существует множество статических анализаторов, которые реализуют более полный анализ по сравнению с компилятором и в результате работы выдают расширенные сообщения об ошибках статической семантики и семантики периода исполнения языка программирования.

Мощные анализаторы ошибок базируются на сложном потоковом анализе и оперируют большим количеством информации, получаемой из анализируемой программы, со сложными взаимосвязями. Генерируемые сообщения об ошибках могут оказаться достаточно объемными и сложными для пользователя. Для визуального представления и детального исследования этой информации в диалоговом взаимодействии с пользователем был бы полезен инструмент нового вида — статический отладчик.

Статические анализаторы могут обнаруживать следующие типы недобротностей [5] и ошибок:

- выход индекса за границы массива,
- деление на ноль, извлечение корня из отрицательного числа,
- разыменование нулевого указателя,
- выход за границы отведенной памяти,
- отсутствие инициализации переменной,

- неиспользуемое значение,
- неправильное приведение динамического типа,
- использование «мертвой» переменной, которая перестала существовать,
- недостижимая ветвь,
- переполнение при арифметических операциях и при приведении типа,
- нулевое число повторений цикла, бесконечные циклы и рекурсии,
- выделение памяти размером меньшим либо равным нулю и др.

Статический отладчик **MrSpidey** [6], написанный для языка Scheme, базируется на компонентном статическом анализе, определяющем отношения на множествах значений объектов анализируемой программы. Это частный случай анализа программ на языке ML, реализованного Хейнтзом (Heintze) [17]. Анализ возможно применять для объектно-ориентированных и процедурных языков. Анализ состоит из двух фаз: фазы спецификации и фазы решения. Во время фазы спецификации определяются ограничения на множестве значений, которые могут принимать программные выражения. Во время фазы решения процедура анализа огрубляет ограничения, т.е. делает конечными потенциально бесконечные множества величин, которые должны удовлетворять определенным ранее ограничениям. В результате получается аппроксимированное множество значений для каждого используемого в программе выражения.

Данный вид анализа был взят за основу отладчика MrSpidey по трем основным причинам:

- анализ Хейнтза дает точные программные инварианты для языков, подобных Scheme, даже для сложных программных конструкций и данных;
- анализ Хейнтза интуитивно понятен для программиста. Анализ интерпретирует программные операторы в простые операторы, которые могут во время исполнения принимать определенное множество значений, и распространяет эти множества через весь процесс исполнения программы;
- с помощью алгоритма анализа Хейнтза возможно предоставить дополнительные объяснения по каждому из полученных инвариантов.

Данный вид анализа имеет ряд недостатков, основной из них — это ограничение на размер программы. Время анализа растет в кубической зависимости от размера программы. Программы размером более 2000 строк

сложно анализировать. Проблема решается с помощью применения компонентного анализа. Компонентный анализ — это усовершенствованный анализ Хейнтза. В процессе компонентного анализа генерируются и огрубляются отношения независимо для каждого программного модуля. Затем эти упрощенные отношения объединяются и решаются, что в результате дает инвариантную характеристику поведения всей программы в целом. Такой вид анализа позволяет производить обработку программы значительно больших объемов по сравнению с анализом Хейнтза. Результаты компонентного анализа сохраняются в файле для каждого модуля, что позволяет существенно экономить время при повторном анализе.

MrSpidey — это статический отладчик с графическим отображением программных инвариантов и детальной информацией об их происхождении. MrSpidey использует информацию об инвариантах, полученную в результате статического анализа. На основе этой информации он определяет потенциально ошибочные ситуации в программе, часть из которых является достоверными ошибками. Программист может детально исследовать каждую потенциально ошибочную ситуацию.

Для выделения опасных операторов используется интерактивное графическое представление информации, благодаря чему можно легко и быстро определить, что же действительно происходит в конкретной точке при исполнении программы. Для этого MrSidey:

- выделяет опасные программные операторы,
- устанавливает тип каждого программного выражения,
- предоставляет графическое объяснение по каждому инварианту.

MrSpidey предоставляет эту информацию программисту с помощью выделения шрифтом и цветом результатов анализа. Дополнительная информация предоставляется через контекстное выпадающее меню, которое ассоциировано с каждым из таких помеченных элементов. Таким образом, в отладчике осуществляется визуализация информации, накопленной в результате анализа.

Любые примитивные операторы, которые могут вызвать ошибку в программе во время исполнения, выделяются красным цветом. Безопасные примитивные операторы выделяются зеленым цветом. Любая функция, которая может быть вызвана с неверным количеством аргументов, отображается с ключевым словом *lambda* и подсвечивается красным. MrSpidey также предоставляет общую информацию по каждой из найденной опасной операции с гиперссылкой на соответствующий оператор.

Через контекстное выпадающее меню также предоставляется значительная часть информации по каждому выражению в анализируемой программе. Эта информация не отображается немедленно, т.к. это приводило бы к значительному загромождению текста программы.

С помощью меню можно получить следующую информацию:

- множество возможных значений для каждого выражения и переменной,
- тип для каждого выражения в программе,
- графическое разъяснение (с помощью стрелок) для каждого инварианта и др.

Начало и конец каждой графической стрелки в отладчике ассоциированы с гиперссылками, которые позволяют осуществлять быструю контекстную навигацию. Нажатие кнопки мыши в начале стрелки помещает фокус к выражению возле конца стрелки, и наоборот. Это особенно удобно в больших программах. Если отображение стрелок (например, при выводе всех предков, участвующих в получении определенного инварианта) сильно загромождает текст программы, MrSpidey позволяет настроить фильтры, чтобы ограничить количество отображаемых стрелок по определенному признаку. Так, при использовании правильной комбинации отображения предков и фильтрации, программист может легко увидеть процесс изменения определенного объекта в ходе исполнения программы.

Если текст программы содержится в нескольких файлах (модулях), то может оказаться, что начало или конец стрелки ссылается на выражение в другом программном модуле. В этом случае MrSpidey рисует стрелку, которая начинается (или заканчивается) на левом поле программы. Нажатие кнопки мыши на поле дает возможность переместиться в нужный модуль к искомому выражению. Таким образом, с помощью всей этой информации программист может легко исследовать каждую потенциально ошибочную ситуацию и определить, действительно ли ошибка имеет место быть. Статический отладчик MrSpidey — это эффективный инструмент для исследования сложных программ.

Существует прототип статического отладчика **Syntox** [7, 8]. Отладчик создан с научно-исследовательскими целями и позволяет производить статическую отладку программ, написанных на языке типа Pascal. Он основан на потоко-нечуствительном анализе. Syntox можно использовать для локализации ошибок, связанных с диапазонами скалярных величин, таких, как индексация массивов, диапазон подтипов и др.

Для отладки программист может вставить в исходный текст программы специальные утверждения. Нарушение этих утверждений рассматривается как ошибки периода исполнения. Возможно использование утверждений двух типов. Инвариантное (безусловное) — утверждение, свойство которого должно всегда выполняться в данной контрольной точке. Экспериментальное — утверждение, которое должно выполниться хотя бы один раз за весь период исполнения программы. Это такое утверждение, что любое исполнение программы должно в конечном итоге приводить к заданной контрольной точке, причем с состоянием памяти, удовлетворяющим свойствам экспериментального утверждения. Экспериментальное утверждение можно использовать для определения множества данных программы (в частности, множества значений локальных переменных или входных параметров), при которых это утверждение будет выполняться. Например, инвариантное утверждение со свойством false означает, что конкретная точка никогда не должна быть достигнута, тогда как экспериментальное утверждение true означает, что контрольная точка должна быть достигнута хотя бы один раз. В частности, назначить контрольную точку на завершение программы можно, если поставить экспериментальное утверждение со свойством true в конце программы, при этом Syntox может определить множество входных параметров, при которых программа завершается. Инвариантные и экспериментальные утверждения можно легко смешивать, что дает программисту хорошую гибкость для построения правильных условий и исследования поведения программы.

Отладчик Syntox имеет интерфейс, реализованный для системы X Window. После анализа программы пользователь может нажать кнопку мыши на любой программной конструкции для отображения всплывающего окна, содержащего свойства данной конструкции. Например, можно посмотреть возможные диапазоны значений параметров, переданных при вызове функции. Когда процедура принимает ссылки в качестве параметра, отладчик предоставляет множество всех переменных, на которые может ссылаться эта ссылка. Экспериментальное и инвариантное утверждения могут быть помещены перед любой программной конструкцией.

Этот отладчик гораздо слабее в сравнении с MrSpidey как по мощности используемого анализа, так и по возможностям визуализации данных. Отладчик не позволяет работать с программами, у которых процедуры передаются в качестве параметров.

Система **PolySpace Verifier** [8, 9] разработана с целью автоматического обнаружения ошибок и контроля одновременного доступа к разделяемым переменным для программ, написанных на языках C и Ada. С помощью

этой системы можно локализовать все описанные ранее типы недобротностей [5] и ошибок. Кроме этого система обнаруживает конфликты доступа к разделяемым данным в программе, порождающей набор взаимодействующих процессов.

Графический пользовательский интерфейс реализует визуализацию обнаруженных ошибок и навигацию по программе посредством следующей системы окон.

Окно *Run Time Error View*. В этом окне модули программы, функции и найденные в них ошибки представлены в виде дерева. Имена функций и модулей подсвечиваются цветом, соответствующим наиболее серьёзной найденной ошибке. Ошибки классифицируются на четыре категории:

- безусловные ошибки (подсвечиваются красным цветом);
- потенциальные ошибки (подсвечиваются оранжевым);
- недостижимые фрагменты кода (подсвечиваются серым);
- безопасные операторы (подсвечиваются зелёным).

Предоставляется статистика по каждому модулю и функции программы. В частности, отображается количество обнаруженных ошибок каждой категории, количество строк текста программы и так далее. Имеется возможность разворачивать и сворачивать ветки отображаемого дерева для детализации информации об ошибках в конкретном модуле и функции программы. Нажатие кнопки мыши на ошибке позволяет переместиться к участку исходного текста программы, который привёл к её возникновению. Можно использовать фильтрацию ошибок по следующим критериям:

- по критичности ошибки (например, гамма-фильтр удаляет все, кроме безусловных ошибок и недостижимых фрагментов кода);
- по типу ошибки (например, ошибки переполнения в арифметических операциях);
- по цвету.

Окно *исходного текста*. Используется для навигации и исследования исходного текста программы.

Окно *дерева вызовов*. Используется в целях документации и навигации.

Словарь общих (глобальных) данных программы. Цветом отображается риск одновременного доступа для каждой глобальной переменной многопоточной программы:

- разделяемая переменная, защищённая от одновременного доступа, — зелёный;

- разделяемая переменная, не защищённая от одновременного доступа, — оранжевый;
- неразделяемая переменная — чёрный.

Нажатие кнопки мыши на переменной позволяет увидеть её объявление в окне исходного текста. Далее, в отдельном окне, можно визуализировать граф одновременного доступа к переменной. На нем представлены все возможные использования и присваивания этой переменной. Узлы с левой стороны графа представляют точки входа в потоки, в которых осуществляется доступ к переменной. Затем слева направо можно проследить всю цепочку вызовов функций, которая приводит к операции чтения или записи переменной. Нажатие кнопки мыши на узле графа позволяет отобразить участок исходного текста соответствующей функции в цепочке. В окне словаря общих данных программы можно отобразить полный список операций чтения и записи переменной с указанием имени задачи и функции, в которой это происходит, и отображением соответствующего участка исходного текста.

ВИЗУАЛИЗАЦИЯ ПРОГРАММ В СИСТЕМЕ HYPERCODE

Рассмотрим систему визуализации HyperCode [18, 19], которая используется для перепроектирования программ.

Визуализатор HyperCode состоит из нескольких визуальных компонентов. Каждый компонент предоставляет пользователю некоторое специфическое видение исследуемой программы, отображающее, как правило, специальным образом интерпретированную информацию, собранную различными видами анализа.

Для визуализации текста программы используется компонент HCSource. Следует отметить такие приемы, как использование курсора, меняющего форму в зависимости от типа конструкции, над которой он спозиционирован; возможность сворачивания синтаксических конструкций путем замены ее текста, включая все вложенные подконструкции, короткой надписью, что позволяет скрыть глубоко вложенные конструкции, тем самым облегчая навигацию по программе.

Компонент HCContext служит для отображения отношения синтаксической вложенности конструкций. Программа представляется в виде леса, корнями которого являются модули (файлы), составляющие программу. В этом компоненте реализована метафора «постепенного остывания», позволяющая наглядно отобразить последовательность выбора конструкций. Каждый раз, когда конструкция становится текущей, она «резко нагревает-

ся» — ее цвет меняется на ярко-красный. Затем, при выборе других конструкций, цвет постепенно темнеет, пока не возвращается к «холодному» черному цвету.

Компонент для визуализации данных является одним из основных, поскольку данные программы и их описания являются наиболее часто исследуемыми объектами. Акцент сделан на отображение не только структур данных, а скорее на их организацию. Можно привести примеры, когда описания данных не соответствуют их физическому размещению. Например, конструкции типа union в C, redefine в Коболе и т.п. накладывают на один и тот же участок памяти объекты различных типов. В тех случаях, когда это используется не для реализации логических вариантов, а для различного доступа к памяти, пользователю необходимо точно знать схему размещения. Поэтому имеет смысл наглядное отображение взаимного расположения элементов памяти: того, что один элемент памяти вкладывается в другой либо они в точности совпадают, либо пересекаются. Подобное представление объясняет происхождение данного элемента памяти и его вложенности в описанные переменные и позволяет более полно представить граф зависимостей данных.

В HyperCode также имеется компонент визуализации управления. Для этого используется представление управляющей структуры программы в графовом виде. При отображении графа используется ряд приемов.

- Сокрытие служебных процедур. При правильной организации программ в них выделяются процедуры нижнего уровня, обеспечивающие обмен, обработку ошибочных ситуаций и т.п. Эти процедуры не влияют на логику управления в программе, но, поскольку вызываются повсеместно, существенно усложняют граф управления. HCCallie дает пользователю возможность указать *фильтр* — перечень тех процедур (точнее, шаблоны их имен), которые не следует включать в граф управления.
- Пользователя чаще всего интересует только лишь некоторая *окрестность* графа вызовов для текущей процедуры, определяемая количеством поколений вызываемых и вызывающих процедур. Эта техника становится еще эффективнее в сочетании с так называемой метафорой «рыбьего глаза»: объекты (процедуры), находящиеся в центре внимания, отображаются крупнее и ярче, чем находящиеся на удалении.
- Развертка графа. Размер графа при этом обычно возрастает, но зато граф становится более простым для восприятия. В этом случае поль-

зователь видит начальные отрезки путей в графе вызовов, начинающиеся в текущей процедуре. Отождествлению различных вершин, изображающих одну и ту же процедуру, может помочь тщательная раскраска вершин графа. Используется следующий метод: все непосредственно вызывающие и вызываемые процедуры имеют одинаковую яркость и равномерно распределяются по спектру. Процедура следующего поколения находится в части спектра своего родителя и менее ярка, если только она уже не появлялась ранее, тогда ее цвет берется с уже существующей вершины.

Компонент HCTrace — интерактивный аниматор передачи управления в исследуемой программе. Аниматор осуществляет *псевдовыполнение* программы. Пользователь может начать псевдовыполнение с произвольной точки. Аниматор с определенной, управляемой пользователем частотой выбирает очередную конструкцию и делает ее текущей. В тот момент когда очередная конструкция зависит от реальных данных (например, при обработке условного оператора или при переходе по вычисляемой метке), аниматор предлагает сделать этот выбор пользователю. При этом аниматор хранит динамическую цепочку вызовов и корректно осуществляет возврат из процедур.

ЗАКЛЮЧЕНИЕ

Исходной целью настоящей работы был обзор статических отладчиков, используемых в них способов отображения информации о программе, ее визуализации и средств навигации по ней. Статический отладчик — это инструмент, который в диалоговом взаимодействии с пользователем обеспечивает адекватную визуализацию разнообразной информации, полученной в результате статического анализа программы. Инструмент такого рода был бы полезен для статического анализатора Wasp. Однако выяснилось, что термин *статический отладчик* не является общепринятым и встречается лишь однажды и в ином смысле: как система статического анализа со встроенными средствами визуализации данных. Другая аналогичная система названа абстрактным отладчиком. В связи с этим решено провести также обзор аналогичных инструментов среди динамических отладчиков, статических анализаторов и средств визуализации данных, проецируя их на статические отладчики.

В настоящее время существует большое количество качественных многофункциональных динамических отладчиков. Среди них особо стоит вы-

делить интерфейсные отладчики, основной функцией которых является унифицированное управление подчиненными отладчиками, отображение информации, навигация по программе и визуализация объектов в удобном виде, независимо от подключенного подчиненного отладчика. Применяемый подход реализации отладчика в виде интерфейсных и подчиненных компонентов можно было бы использовать для реализации статических отладчиков.

Syntox является экспериментальной системой. Она использует потоконечуствительные алгоритмы анализа и не позволяет работать с программами, у которых процедуры передаются в качестве параметров. Визуализация информации, полученной в результате статического анализа, производится с помощью всплывающего окна в зависимости от выбранной программной конструкции. Отладчик MrSpidey имеет развитые средства навигации и визуализации данных, интерактивное графическое представление информации, графическое отображение программных инвариантов с детальной информацией об их происхождении. Интересен подход, применяемый для отображения процесса изменения определенного объекта в ходе исполнения программы. Используются графические стрелки и фильтрация. К сожалению, этот отладчик может использоваться только для отладки программ, написанных на языке Scheme, и не предназначен для работы с современными языками программирования, например, такими, как Java и C++. Наиболее развитой системой, безусловно, является PolySpace Verifier. С ее помощью можно локализовать широкий круг недобротностей и ошибок. Система позволяет производить анализ программ, написанных на языках Ada, C и C++. Она имеет развитый интерфейс, средства визуализации и навигации по программе.

Система визуализации HyperCode предоставляет высокоуровневые средства визуализации разнообразных связей в программных системах: богатые возможности для навигации по программе, отображение синтаксической вложенности конструкций, метафора «рыбьего глаза», интерактивный аниматор и др.

Некоторые из перечисленных в данной работе концепций динамических отладчиков, а также возможностей отображения информации о программе, ее визуализации и средств навигации по ней после той или иной степени адаптации планируется использовать при разработке статического отладчика для анализатора Wasp. Первоначально планируется реализация статического отладчика, работающего в текстовом режиме с управлением из командной строки. Далее следует разработать интерфейсный отладчик на базе современных графических библиотек, с помощью которого будет осущест-

вляться графическое управление подчиненным отладчиком. В интерфейсном отладчике возможна реализация многих из приведенных в данной работе концепций визуализации данных.

Автор благодарен С. К. Черноножкину за конструктивную критику и множество полезных замечаний по работе.

СПИСОК ЛИТЕРАТУРЫ

1. **Шелехов В.И., Куксенко С.В.** Статический анализатор семантических ошибок периода исполнения // Программирование. — 1998. — N 6. — С. 23–43.
2. Using SoftICE. — Compuware Corporation, 1998. — http://www.midisa.omnet.ru/book/documents.zip
3. **Столмен Р., Пеш Р., Шебс С. и др.** Отладка с помощью GDB. Отладчик GNU уровня исходного кода. — 2000. — http://www.linux.org.ru/books/GNU/gdb/gdb_toc.html
4. **Документация** по JDK Tools. Отладчик Java-классов JDB. — JavaCamp-Group and Sun Microsystems Inc. — http://cad.ntu-kpi.kiev.ua/~netlib/java/JSB/jdktool/jdb.html
5. **Поттосин И. В.** “Хорошая программа”: попытка точного определения понятия // Программирование. — 1997. — № 2. — С. 3–17.
6. **Flanagan C.** Effective Static Debugging via Componential Set-Based Analysis: PhD thesis. — Houston, May 1997. — 173 p.
7. **Bourdoncle F.** Assertion-Based Debugging of Imperative Programs by Abstract Interpretation // Fourth European Software Engineering Conf. — Springer-Verlag, 1993. — P. 501–516.
8. **Bourdoncle F.** Abstract Debugging of Higher-Order Imperative Languages // Conf. on Programming Language Design and Implementation. — ACM Press, 1993. — P. 46–55.
9. **PolySpace** C Developer Edition. — PolySpace Technologies. — http://www.polyspace.com/datasheets/c_psde.htm
10. **Abstract** Interpretation. — PolySpace Technologies. — http://www.polyspace.com/docs/Abstract_Interpretation_paper.pdf
11. **Матушкин Г.Г.** Полноэкранный профессиональный отладчик AFDPROR. — Новосибирск, 1998 — (Препр./ НГТУ).
12. **Кэнту М.** Delphi 7 для профессионалов. — СПб.: Питер, 2004. — 1104 с.
13. **Зеллер А.** Отладка в DDD. Руководство пользователя и справочник. — 2000. — http://www.gnu.net.ru/ddd/dddm_toc.html
14. **van Rossum G.** Python Tutorial — PythonLabs, 2003. — http://www.python.org/doc/2.3.3/tut/tut.html
15. **Кристиансен Т., Торкингтон Н.** Perl. Библиотека программиста. — СПб.: Питер, 2000. — 736 с.

16. **Darwin Ian F.** Checking C Programs with lint. — O'Reilly, 1988. — 81 p.
17. **Heintze N.** Set-based analysis of ML programs // Proc. of the ACM Conf. on Lisp and Functional Programming, 1994. — P. 306–317.
18. **Бульонков М.А., Бабурин Д.Е.** HyperCode — открытая система визуализации программ // Автоматический реинжиниринг программ. — Изд-во СПбГУ, 2000. — С. 165–183
19. **Авербух В.Л.** Визуализация программного обеспечения: Автореф. дис... канд. физ.-мат. наук. — Екатеринбург, 1995. — 168 с.

Еремин А.В. Архитектура настраиваемой среды отладки // Проблемы программирования. — Киев, 2000. — № 3–4 — С. 89–102.

С. В. Каличкин

ОБЗОР СРЕДСТВ СТАТИЧЕСКОЙ ОТЛАДКИ ПРОГРАММ

Препринт
112

Рукопись поступила в редакцию 10.12.04
Редактор З. В. Скок
Рецензент С. К. Черноножкин

Подписано в печать 12.05.04
Формат бумаги 60 × 84 1/16 Объем 1.3 уч.-изд.л., 1.4 п.л.
Тираж 60 экз.

ЗАО РИЦ «Прайс-курьер»
630090, г. Новосибирск, пр. Акад. Лаврентьева, 6, тел. (383-2) 34-22-02