Интегрированная инструментальная среда Poirot для изучения методов маскировки программ

А. В. Чернов

Аннотация. В данной работе описывается интегрированная инструментальная программная среда Poirot, разработанная для изучения методов маскировки программ.

1. Введение

В настоящее время является актуальной проблема защиты компьютерных программ от обратной инженерии, проводимой с целью модификации и/или включения фрагментов защищаемой программы во вновь разрабатываемый программный код. Защита программ в данном случае состоит в том, чтобы затруднить понимание деталей реализации компонент большой программной системы, сделав его настолько дорогим, чтобы дешевле было разработать и реализовать оригинальное программное обеспечение. Одним из способов такой защиты является *маскировка программ*, заключающаяся в применении к исходному тексту программы цепочки *маскирующих преобразований*, то есть преобразований, сохраняющих реализуемую программой функцию (являющихся функционально эквивалентными), но делающих понимание этой функции трудным.

Среди современных систем программирования уже имеются системы, поддерживающие различные методы маскировки программ для языков C и C++ [1], Java [2] и других. Они, как правило, выполняют лишь простейшие маскирующие преобразования текста программы и (или) графа потока управления программы и её структур данных. Методы, используемые в них, как правило, подобраны эмпирически и слабо обоснованы теоретически. С другой стороны, разработаны и успешно применяются в оптимизирующих компиляторах и средствах обратной инженерии весьма мощные методы анализа программ. Назрела необходимость анализа действия маскирующих преобразований и оценки устойчивости замаскированных программ против существующих методов анализа программ. Необходимым условием проведения такого анализа является разработка подходящей инструментальной системы.

В Институте системного программирования РАН разрабатывается инструментальная среда Poirot, обеспечивающая возможность маскировки и демаскировки программ, а также поддерживающая анализ маскирующих преобразований программ и разработку новых маскирующих преобразований. Интегрированная среда Poirot, являясь расширяемой, позволяет добавлять новые методы анализа и трансформации программ. Система Poirot помещена в открытый доступ в Internet [3] и может использоваться для анализа существующих методов маскировки программ и разработки новых методов маскировки программ, непосредственно для маскировки программ и анализа замаскированных программ, а также для решения некоторых проблем обратной инженерии. С помощью интегрированной среды Poirot исследованы методы маскировки программ, используемые одним из коммерческих маскировщиков, и выявлены слабые стороны этого метода. Интегрированная среда используется в учебном процессе на факультетах ВМиК МГУ и ФПМЭ МФТИ.

2. Общее описание

Интегрированная среда Poirot построена как набор связанных друг с другом инструментов, работающих над общим промежуточным представлением программ MIF. Для управления инструментами ИС предоставляется графический интерфейс пользователя. Общая архитектура интегрированной среды Poirot показана на Рис. 1.

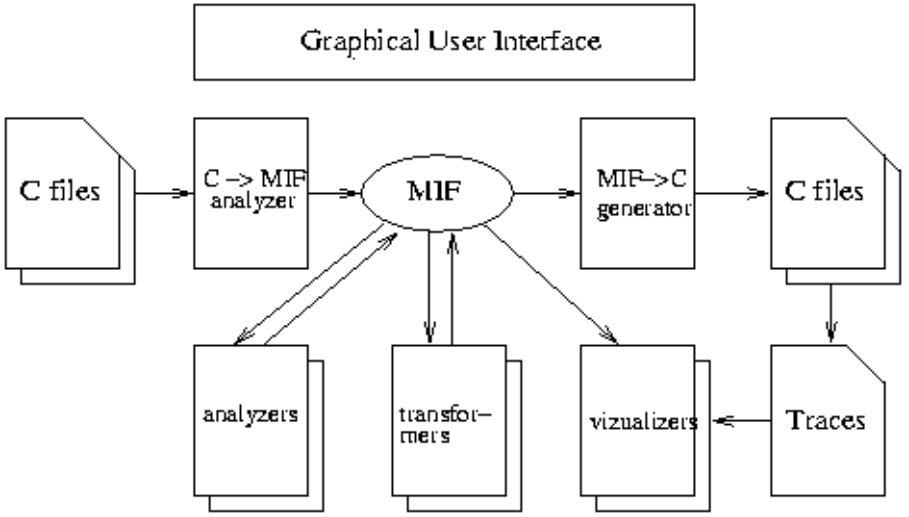

Рис. 1. *Архитектура интегрированной среды Poirot*

В настоящее время в качестве исходного и целевого языка программирования используется язык Си, но внутреннее представление разработано таким образом, чтобы поддерживать широкий класс процедурных и объектно-ориентированных языков программирования. Все компоненты ИС реализованы на языке Java, за исключением транслятора из Си в MIF, который реализован на языке Си.

Программа на языке Си транслируется в промежуточное представление с помощью компонента **C to MIF analyzer**. В настоящее время поддерживается стандарт ISO C90 и некоторые расширения GNU. Ведётся работа над реализацией поддержки стандарта ISO C99. Чтобы обеспечить независимость интегрированной среды от деталей реализации стандартной библиотеки Си для каждой конкретной платформы и обеспечить возможность корректной генерации программы на Си по её внутреннему представлению, анализатор использует собственный набор стандартных заголовочных файлов (stdio.h и т. д.). На уровне стандартной библиотеки полностью поддерживается стандарт ISO C90 и некоторые заголовочные файлы POSIX. Внутреннее представление программы находится в памяти интегрированной среды, но возможно сохранение внутреннего представления в файле.

Компонент **MIF to C Generator** позволяет по программе во внутреннем представлении получить программу на языке Си. При генерации программы корректно генерируются директивы #include для всех использованных в исходной программе системных заголовочных файлов. Текущая версия генератора не сохраняет управляющую структуру высокого уровня, то есть, например, циклы while, for, do while в программе, сгенерированной из внутреннего представления, заменяются операторами if и goto. Для проведения полустатического анализа программ генератор поддерживает несколько типов инструментирования программы. Инструментирование программы заключается во внесении в текст программы специальных операторов, собирающих информацию о ходе выполнения программы. В настоящее время генератор поддерживат инструментализацию программы для сбора полных трасс выполнения программы, профилирование базовых блоков и дуг [4], профилирование значений [5]. Собранные в результате выполнения инструментированной программы профили выполнения могут впоследствии использоваться для анализа и преобразования программ.

Компоненты, обозначенные на Рис. 1 как **Analyzers**, реализуют различные методы статического и полустатического анализа программ. При этом сама программа не трансформируется, а во внутреннее представление программы добавляется полученная в результате выполнения анализа информация. В интегрированной среде эти компоненты доступны посредством пункта меню “Analyze”. Например, алгоритм разбиения программы на базовые блоки, доступный через пункт меню “Mark basic blocks”, строит граф потока управления программы, создаёт соотвествующие структуры данных в памяти

системы и добавляет ссылки на построенный граф в структуры данных внутреннего представления программы.

Компоненты, обозначенные на Рис. 1 как **Transformers**, реализуют различные преобразования программ. При этом результатом работы компонента трансформации является новоя программа во внутреннем представлении, для которой в интегрированной среде создаётся новое окно. Исходная программа сохраняется неизменной. Трансформационные компоненты доступны в интегрированной среде посредством пунктов меню “Optimize”, “Transform” и “Obfuscate” в зависимости от класса преобразования.

Компоненты, обозначенные на Рис. 1 как **Vizualizers**, реализуют различные алгоритмы визуализации информации о программе. Эти компоненты доступны посредством пункта меню “Vizualize” интегрированной среды.

3. Промежуточное представление

Промежуточное представление MIF используется всеми инструментами интегрированной среды. Оно является представлением среднего уровня [6] и спроектировано таким образом, чтобы представлять программы, написанные на широком спектре процедурных и объектно-ориентированных языков программирования.

```
int fib(int n)                  L2:   DISPLAY
{                               L3:   VAR     $int,n,
  int a, b, c;                        END     L2
  a = 1;                        L4:   DISPLAY
  b = 1;                        L5:   VAR     $int,a,
  if (n <= 1) return 1;         L6:   VAR     $int,b,
  for (; n > 1; n--) {          L7:   VAR     $int,c,
    c = a + b;                        END     L4
    a = b;                      L8:   FUNC    $int,@1,L2,L2,L4,fib
    b = c;                            LOAD    L5,$int,1
  }                                   LOAD    L6,$int,1
  return c;                           JGT     L3,1,L22,
}                                     LOAD    @1,$int,1
                                      JMP     L9,
                                L22:  JLE     L3,1,L23,
                                      ADD     L7,L5,L6
                                      MOVE    L5,L6
                                      MOVE    L6,L7
                                      DECR    L3
                                      JMP     L22,
                                L23:  MOVE    @1,L7
                                L9:   END     L8
                                      EXPORT  L8,fib
```

Рис. 2. *Функция fib и её внутреннее представление*

Программа в представлении MIF представляет собой последовательность четвёрок, которые используются для представления как декларативной, так и императивной информации о программе. Текстуальное представление MIF используется как интерфейс между анализатором языка и интегрированной средой и для хранения анализируемых программ. На Рис. 2 дан текст внутреннего представления для функции fib, вычисляющей числа Фибоначчи.

Транслятор программ из Си в MIF генерирует промежуточный файл, содержащий программу во внутреннем представлении в текстовом формате, как показано на Рис. 2. Далее происходит импорт программы в текстовом представлении MIF в интегрированную среду. При этом происходит её синтаксический и семантический анализ корректности внутреннего представления отображение внутреннего представления в систему объектов в памяти. Классы, используемые для хранения программы в представлении MIF в памяти, предоставляют интерфейс для классов, реализующих различные методы анализа. Кроме того, классы хранят результаты анализа программы, которые не сохраняются в текстовом представлении, например, графы потока управления функций.

В текстовом представлении MIF для идентификации переменных, структур, точек перехода программы используются метки (на Рис. 2 это `L2`, `L3` и т. д.), которые в памяти заменяются на объектные ссылки. Для хранения промежуточных результатов вычислений используются абстрактные регистры (*псевдорегистры*), записываемые в текстовом представлении как @<номер>, например, `@1`. Псевдорегистры отделяются от обычных переменных, поскольку, во-первых, псевдорегистры не могут иметь алиасов и, во-вторых, для псевдорегистров (за исключением возвращаемого значения функции) выполняется свойство статически однократного присваивания. Псевдорегистры являются первоочередными кандидатами для размещения на регистры процессора при генерации низкоуровневого (ассемблерного) кода. Локальные переменные и функции группируются в дисплеи (см. инструкцию `DISPLAY` на Рис. 2), допускается динамическое создание и уничтожение дисплеев (в стековом порядке). В любой точке внутреннего представления функции статически известен стек дисплеев, существующий в момент выполнения этой точки. Для переходов через границы используются дисплейные списки, то есть списки создаваемых и уничтожаемых дисплеев при переходе.

Семантика большинства инструкций промежуточного представления очевидна из её названия. Так, инструкция `LOAD` сохраняет в переменной или псевдорегистре, задаваемом первым параметром инструкции, указанное значение указанного типа. Инструкция `ADD` складывает значения, хранящиеся в переменных или псевдорегистрах, задаваемых вторым и третьим аргументами инструкции, и помещает результат в переменную или псевдорегистр, задаваемый первым параметром инструкции. Требуется, чтобы все три аргумента инструкции имели строго один и тот же тип. Необходимые инструкции преобразования типов `CAST` добавляются при трансляции

программы из Си во внутреннее представление. Полное описание внутреннего представления MIF доступно по сети Интернет [3].

Для хранения программы во внутреннем представлении в памяти интегрированной среды используется система классов. Базовым классом является класс `MIFElem`, главным образом определяющий набор констант, идентифицирующих элементы представления. Объекты этого класса используются для представления терминальных узлов, таких как имена базовых встроенных типов представления MIF (например, `$bool`, `$int` и т.д.). Класс `MIFValue` используется для хранения литеральных констант и содержит ссылку на объект, хранящий значение литерала. Класс `MIFReg` используется для хранения псевдорегистров. Класс `MIFInstr` используется для хранения инструкций и наследуется в нескольких специальных случаях (заголовок функции, определение переменной, φ-функция).

Для хранения информации о графе потока управления и деревьях доминирования используются классы `BasicBlock`, `FlowGraph`. Каждая инструкция MIF содержит ссылку на первую инструкцию своего базового блока, а первая инструкция содержит ссылку на соответствующий объект класса `BasicBlock`. Кроме того, заголовок функции содержит ссылку на объект класса `FlowGraph` этой функции.

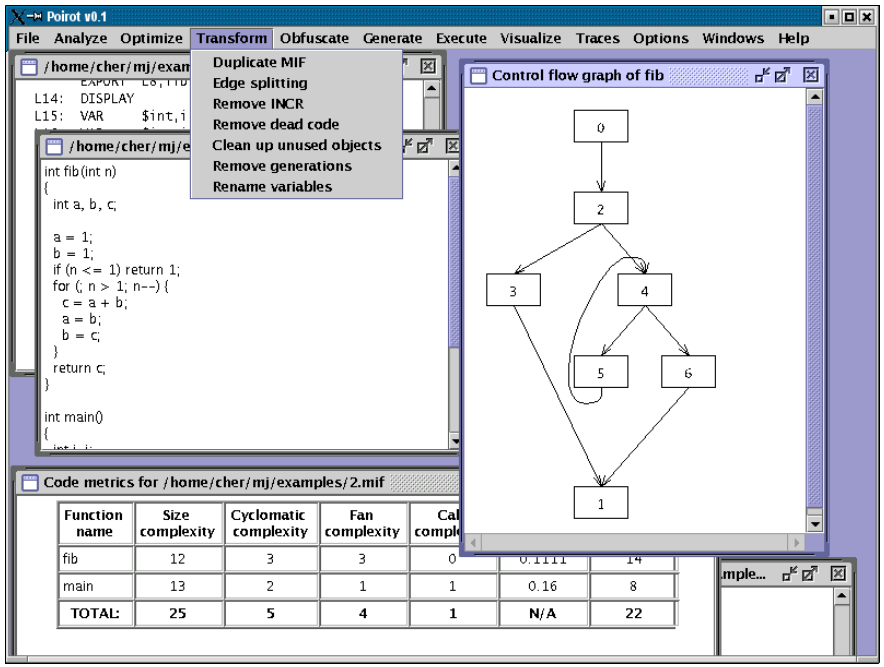

Рис. 3. *Вид экрана при работе интегрированной среды*

4. Интерфейс пользователя

Пользователь управляет интегрированной средой с помощью графического интерфейса пользователя. Главное меню предоставляет доступ к следующим меню групп инструментов: File, Analyze, Optimize, Transform, Obfuscate, Generate, Execute, Visualize, Traces, Options, Windows, Help. Типичный вид экрана при работе интегрированной среды показан на Рис. 3.

В настоящее время реализована лишь прототипная версия среды Poirot, в силу чего некоторые пункты меню не работают. Реализация этих пунктов связана с дополнительными исследованиями, в результате которых возможно появление новых меню и коррекция уже имеющихся.

В меню File собраны стандартные инструменты работы с файлами для диалоговой интегрированной среды, которые позволяют открывать существующий файл на языке Си или в промежуточном представлении, сохранять файл на языке Си, завершать работу в интегрированной среде. Меню Windows позволяет быстро переключаться в окна специального назначения (окно компиляции) или выбрать окно из списка всех окон, открытых в среде. Меню Help предназначено для доступа к встроенной справочной информации по интегрированной среде. В настоящее время оно содержит лишь пункт About, выводящий информацию о версии ИС, авторах и лицензионном соглашении. Все остальные пункты меню открывают доступ к специализированным инструментам и описаны в дальнейших разделах.

4.1. Меню Analyze

Пункт меню Analyze открывает доступ к инструментам статического и полустатического анализа программ общего назначения, то есть не предназначенным специально для анализа замаскированных программ. Этот пункт меню содержит следующие подпункты:

- Make MIF – оттранслировать файл на языке Си во внутреннее представление. Чтобы воспользоваться этим пунктом меню необходимо сделать активным окно с файлом на Си, которую необходимо транслировать. В результате трансляции создаётся новое окно, в котором отображается внутреннее представление в текстовой форме. Перевод программы во внутреннее представление – необходимый шаг для её анализа или трансформации, так как все остальные инструменты ИС манипулируют программой во внутреннем представлении.

- Mark basic blocks – разметить базовые блоки во всех функциях программы. В каждой функции данного файла выделяются базовые блоки и строится граф потока управления. Шаг выделения базовых блоков выполняется неявно во многих инструментах анализа и трансформации программы. В результате выделения базовых блоков текущее окно будет содержать информацию о базовых блоках каждой функции.

- Convert to SSA – преобразовать программу во внутреннем представлении в программу со статически однократным присваиванием SSA [7]. В результате преобразования в представление SSA создаётся новое окно с именем “SSA form of oldtitle”, где oldtitle – имя окна с исходной программой. Представление SSA программы применяется для некоторых алгоритмов анализа программы, например, продвижения констант, а также в некоторых алгоритмах маскировки, например, внесения дублирующего кода.

- Convert from SSA – преобразовать программу из представления SSA в обычное представление, удалив из программы все поколения переменных и φ-функции. Программа во внутреннем представлении SSA-MIF не может быть непосредственно откомпилирована или переведена в Си. Для этого предварительно программа должна быть переведена в обычную форму. Результат преобразования помещается в отдельное окно с именем “Regular form of oldtitle”, где oldtitle – имя окна с исходной программой в представлении SSA.

- Mark dead code – пометить мёртвый код. По результатам анализа потоков данных программы выявляются “мёртвые” инструкции. В окне с внутренним представлением программы такие инструкции помечаются символом “D” в первой позиции строки.

- Liveness – установить интервалы жизни переменных. В окне с внутреннем представлением программы около каждой инструкции отмечаются переменные, которые живы в этой точке программы.

- Static slicing – построить статический слайс программы. В дополнительном диалоговом окне запрашиваются параметры слайса. Результат построения слайса отображается в отдельном окне.

- Redundancy detection – выявить в программе частично избыточные инструкции. Инструкции, которые могут быть удалены, отмечаются в программе символом “D”, а инструкции, которые должны быть перемещены, помечаются символом “M”.

- Induction variable detection – выявить в программе индуктивные переменные циклов.

- Alias detection – провести анализ алиасов. Для каждого косвенного доступа к памяти выявляется множество переменных, которые могут быть затронуты этой операцией. Все методы статического анализа, реализованные в интегрированной среде, реализуют консервативные алгоритмы, если только более точная информация об алиасах не была заранее вычислена с помощью данного инструмента.

- Graph coloring – выполнить минимизацию переменных с помощью раскраски графа.

- Show edge profiling results – показать результаты профилирования дуг. Создаётся новое окно с графом потока управления программы, на каждая дуга которого помечена числом её прохождений при профилировании.

- Show code metrics – вычислить и вывести в отдельное окно метрики сложности кода программы. Вычисляются метрика размера функции, цикломатической сложности, сложности графа вызовов, сложности потока данных.

- Show value profiling results – показать результаты профилирования значений.

4.2. Меню Optimize

В данном меню собраны инструменты выполняющие статическую оптимизации программ. Пункты меню кратко описаны ниже.

- Copy propagation – выполнить алгоритм продвижения копий. Результат оптимизации отображается в отдельном окне.

- Simple constant folding – выполнить свёртку константных выражений, заменяя инструкции вычисления выражения, значение которого известно при компиляции, на константу-результат. Доступный в данном пункте алгоритм выполняет свёртку константных выражений в границах базовых блоков. Результат свёртки констант отображается в отдельном окне.

- Constant propagation – выполнить продвижение констант в рамках всей функции.

- Pipehole optimizations – выполнить простейшие оптимизации графа потока управления программы, например, устранение безусловных переходов на инструкции безусловных переходов.

- Eliminate dead code – устранить из программы мёртвый код.

- Eliminate redundant assignments – провести анализ интервалов жизни переменных и устранить избыточные присваивания.

- Partial redundancy elimination – провести устранение частичной избыточности во всех функциях программы. Результат отображается в новом окне.

- CSE – выполнить устранение общих подвыражений. Результат оптимизации отображается в новом окне.

- Loop optimization – провести вынос инвариантов цикла за цикл.

- Tail-call optimization – заменить хвостовую рекурсию итерацией.

- Expression simplification – выполнить алгебраические упрощения арифметических и логических выражений. Например, выражение 1*x заменяется на x.

- Strength reduction – выполнить алгоритм “понижения прочности” выражений, в которые входит индуктивная переменная цикла.

- Basic-block reordering – выполнить полустатическую (основанную на профилировании дуг) оптимизацию перестановки базовых блоков.

4.3. Меню Transform

Меню Transform предоставляет интерфейс к различным алгоритмам трансформации программ, которые непосредственно не относятся ни к алгоритмам оптимизации программ, ни к алгоритмам маскировки программ. Пункты этого меню описаны ниже.

- Duplicate MIF – создать новое окно с копией программы из текущего окна.

- Edge splitting – выполнить “расщепление дуг”. Такому преобразованию подвергаются все дуги графа потока управления, исходящие из базового блока с более чем одним последующим базовым блоком и входящие в базовый блок с более чем одним предшествующим базовым блоком. Такие дуги разрываются новым пустым базовым блоком. Данное преобразование необходимо проводить перед некоторыми преобразованиями, которые добавляют или переносят инструкции, например, при переводе в форму SSA.

- Remove INCR – удалить из программы все инструкции инкремента и декремента, заменяя их на соответствующую арифметическую операцию с присваиванием.

- Remove dead code – удалить из программы инструкции, которые были помечены как мёртвые, но ещё не удалены. Такие инструкции отображаются в окне программы помеченными символом “D” в первой позиции строки.

- Clean up unused objects – удалить из промежуточного представления неиспользуемые объекты, например, типы данных, на которые не ссылок.

- Remove generations – переименовать переменные, чтобы устранить поколения переменных, возможно оставшиеся после преобразования из формы SSA.

- Rename variables – переименовать все переменные программы таким образом, чтобы они имели имена вида v<номер>, нумеруя их от 1.

- Specailize function – создать специализированный вариант функции на основании результатов профилирования значений.

4.4. Меню Obfuscate

Это меню позволяет применять к программе различные маскирующие преобразования. Пункты меню перечислены далее.

- Scramble identifiers – выполнить маскировку идентификаторов. В зависимости от режима маскировки, управляемого пользователем, все идентификаторы могут нумероваться подряд, либо нумероваться случайными числами, либо случайным образом меняться местами.

- Insert opaque – вставить непрозрачный предикат в заданное место функции. Вид и параметры непрозрачного предиката задаются в диалоговом окне.

- Insert dead code – добавить мёртвый код с заданными характеристиками в указанное место функции. Параметры маскирующего преобразования задаются в диалоговом окне.

- Insert identity – вставить в текст функции тождество из библиотеки тождеств.

- Inline function – выполнить прямую вставку одной функции в тело другой функции.

- Unroll loop – выполнить полную или частичную развёртку циклов.

- Shuffle instructions – переставить инструкции в базовом блоке произвольным образом, но не нарушая зависимостей по данным между инструкциями.

- Clone function – создать новую функцию, точную копию указанной функции.

- Clone basic block – создать новый базовый блок, являющийся точной копией указанного базового блока, и модифицировать граф потока

управления функции, чтобы обе копии выполнялись с примерно равной частотой.

- Cross edges – выполнить преобразования зацепления дуг и создания псевдоциклов.

- Globalize variables – выполняет перенос локальных переменных функций на статический уровень.

- Make dispatcher – выполнить перестройку потока управления функции и введение диспетчера.

- Move locals to arrays – поместить все локальные переменные в один или несколько локальных массивов (для каждого типа создаётся отдельный массив).

- Obfuscate new – выполнить все шаги метода маскировки [8].

4.5. Меню Generate

Меню Generate предоставляет доступ к инструментам, позволяющим перевести программу во внутреннем представлении обратно в программу на языке Си. Пункт меню Generate C для заданной программы во внутреннем представлении генерирует программу на Си и отображает её в отдельном окне. Пункт меню Generate Instrumented C аналогичен предыдущему пункту, но полученный код на Си будет содержать инструкции для сбора профиля путей выполнения программы. Пункт Generate edge profiled MIF позволяет включить в данную программу во внутреннем представлении вставляются инструкции для сбора профиля дуг. Полученную программу далее необходимо конвертировать с помощью пункта меню Generate C.

4.6. Меню Execute

Это меню предоставляет доступ к инструментам, которые позволяют скомпилировать программу и запустить её на выполнение.

4.7. Меню Visualize

Данное меню предоставляет доступ к инструментам визуализации программы. Пункты меню описываются ниже.

- Show dominator tree – визуализировать дерево доминирования функции. Базовый блок B1 доминирует над базовым блоком B2, если любой путь из ENTRY в B2 проходит через B1. Доминирование называется непосредственным, если на любом пути из B1 в B2 не существует базового блока B3, который доминировал бы над B2. Если блок B1 непосредственно доминирует над B2, то в дереве доминирования они соединяются дугой.

При выборе этого пункта меню появляется дополнительное диалоговое окно, которое позволяет выбрать функцию текущего файла.

- Show rev. dominator tree – визуализировать дерево постдоминирования функции. Базовый блок B1 постдоминируется блоком B2, если любой путь из B2 в EXIT проходит через B1. При выборе этого пункта меню появляется дополнительное диалоговое окно, которое позволяет выбрать функцию текущего файла.
- Show flowgraph – визуализировать граф потока управления функции. Имя функции выбирается в дополнительном диалоговом окне.
- Show dependancy graph – визуализировать граф зависимостей функции. Граф зависимостей объединяет граф зависимостей по управлению и по данным.
- Show Xrefs – визуализировть граф перекрёстных ссылок программы.
- Show callgraph – визуализировать граф вызовов программы.

4.8. Меню Traces

В данное меню включены инструменты для сбора трасс программы и дальнейшего полустатического анализа.

- Show trace flowgraph – построить полустатический граф потока управления функции на основании собранного профиля путей. В построенном графе поддерживается фильтрация базовых блоков с последующей перестройкой программы.
- Show trace callgraph – построить полустатический граф вызовов функций.
- Show trace xrefs – построить полустатический граф перекрёстных ссылок.
- Make MIF – выполнить алгоритм полустатического устранения недостижимого кода. Из программы удаляются инструкции, которые на данных тестовых наборах ни разу не выполнились.
- Make plain MIF – построить линейную последовательность выполнившихся инструкций программы.
- Analyze coverage – выполнить анализ тестового покрытия. Выделяются дуги и пути, которые не были пройдены ни разу на тестовом наборе.

- Compare traces – сравнить несколько трасс, которые собраны при одинаковых входных данных и выявить отличающиеся участки.
- Slice along trace – построить динамический слайс.
- Compare basic block – выбрать и сравнить два базовых блока одной и той же функции или разных функций.

4.9. Меню Options

С помощью данного меню можно настроить интегрированную среду, изменяя шрифты, цвета, пути к необходимым внешним инструментам (компилятору языка Си и конвертеру из языка Си в MIF).

5. Заключение

В данной работе представлена интегрированная среда (ИС) Poirot, разрабатываемая в ИСП РАН. Интегрированная среда в настоящее время находится в активной разработке. Среди инструментов, разрабатываемых или проектируемых в настоящее время, следует особо отметить командный язык для управления инструментами ИС. Такой командный язык позволит ещё более повысить гибкость использования ИС Poirot.

Литература

1. Jammer Code Obfuscator. http://rzr.online.fr/docs/shop/jammer.htm
2. Google Web Directory. Obfuscators. http://directory.google.com/Top/Computers/Programming/Languages/Java/Development_Tools/Obfuscators/
3. The Poirot IRE download page. http://www.ispras.ru/groups/ctt/ire.html
4. T. Ball, J. R. Larus. Optimally Profiling and Tracing Programs. In *Proceedings of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'92)*, 1992.
5. B. Calder, P. Feller, A. Eustace. Value Profiling. In *Proceedings of the 30th International Symposium on Microarchitecture (MICRO-30)*, 1997.
6. S. Muchnick. Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers, 1997.
7. R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4): 450--90, October 1991.
8. А. В. Чернов. Об одном методе маскировки программ. “Труды Института системного программирования РАН”, Том 4, под ред. В. П. Иванникова. М.: ИСП РАН, 2003.