Метод автоматического восстановления переменных из трассы исполнения программы[1]

М.А. Климушенкова, В.А. Макаров,
maria.klimushenkova@ispras.ru, Vladimir.Makarov@orimtech.com

Аннотация. В работе описывается метод восстановления локальных переменных из трассы исполнения программы. Метод использует одну из схем анализа потоков данных – достигающие определения. В ней также рассматриваются существующие подходы к решению задачи восстановления переменных.

1. Введение

В целях обеспечения безопасности использования программных средств, разработанных сторонними исполнителями, часто возникает необходимость анализировать их исполняемые файлы. Это очень сложный и трудоемкий процесс. Для облегчения работы аналитика разработано много различных инструментальных средств. Применение описываемого алгоритма повышает уровень абстракции данных, что позволяет аналитику сконцентрироваться на логике обработки данных, а не на особенностях их представления в коде.

2. Обзор работ в области восстановления переменных

В настоящее время существует не так много работ в области восстановления переменных и их типов. В работах [1], [2] и [3] используется схожий подход при выделении локальных переменных. Он основан на анализе потоков данных, вычислении достигающих определений и построении def-use chains. При этом обработка в них происходит не отдельными инструкциями, а базовыми блоками.

Все рассмотренные решения задачи восстановления локальных переменных в качестве входных данных используют ассемблерный код программы. Описываемый метод предполагает анализ трассы исполнения программы вместо анализа ее статического кода. В процессе выполнения программы могут быть пройдены не все ветки, и некоторые инструкции могу остаться

[1] Работа поддержана грантом РФФИ (11-07-00353).

необработанными. Несмотря на это, использование трассы имеет ряд следующих преимуществ. Исходный код программы не всегда может быть доступен, а анализ ассемблерного кода, полученного при дисассемблировании исполняемого файла, осложняется применяемыми методами обфускации. При динамическом анализе может быть использована информация не только о типах команд и их аргументах, но и о значениях, принимаемых элементами памяти.

3. Описание предлагаемого метода

Любая программа оперирует ограниченным количеством элементов памяти. Один и то же регистр может последовательно представлять разные переменные. Предлагаемый метод позволяет вычислить время жизни переменных и их размещение.

Анализируются только регистры общего назначения и ячейки памяти, принадлежащие стеку (предполагается, что каждая ячейка в статической памяти является отдельной глобальной переменной). Выделение параметров, передаваемых в функцию, является отдельной задачей и в данной работе не рассматривается.

Алгоритм последовательно анализирует все принадлежащие функции инструкции. Они могут *определять* элементы памяти, то есть присваивать им новые значения, или просто использовать их. Для каждой инструкции проходим все элементы, которые она использует или изменяет. Если инструкция определяет элемент, то создаем новое определение, а также новую переменную, и делаем изменения в карте достигающих определений.[4] Если инструкция использует элемент, то из карты достигающих определений узнаем, какой переменной он соответствует. Далее вносим информацию в таблицу соответствия элемент-переменная. Из этой таблицы мы можем узнать, какую переменную представлял тот или иной элемент на каждом шаге.

Будем считать, что определение достигает произвольной точки, если на пути от определения до этой точки элемент не переопределяется. Следует так же раскрыть понятия «определение элемента» и «использование элемента».

Каждая инструкция может создавать четыре типа зависимостей [5]:

- Kill – возникает при изменении значения элемента на новое, не зависящее от других элементов.
- Check – возникает при проверке значение элемента.
- Update – возникает при изменении значений элементов.
- Get/Set – возникает при чтении из памяти/записи в память, во вход зависимости включается элементы от которых зависит адрес.

Выходные элементы зависимости Kill получают новые значения, таким образом, для них возникают новые определения. Входные элементы

зависимости Check не изменяют своего значение, то есть используются. Для зависимости Update возможны три варианта: элемент присутствует только на входе, то есть его значение не изменяется – происходит использование; элемент присутствует и на входе и на выходе зависимости, то есть его значение изменяется, также происходит использование; элемент присутствует только на выходе, то есть получает новое значение – возникает новое определение. Ситуации для зависимостей Get/Set схожи. Элементы на выходе получают новые значения – возникают новые определения, а элементы на входе только используются.

Использование трассы в качестве входных данных для алгоритма позволяет обработать несколько проходов исполнения функции. При анализе каждого прохода происходит уточнение информации, полученной на предыдущих итерациях. Если выполнение происходило по разным веткам, то возможно возникновение ситуации, изображенной на рис. 1.

При первом проходе для регистра eax будет создана переменная var1, которая далее используется в команде сложения. При втором проходе для регистра eax будет создана другая переменная var2, на следующем шаге нужно регистру eax поставить в соответствие переменную var2, но он уже соответствует переменной var1. Необходимо объединить переменные var1 и var2 в одно множество, представляющее собой одну переменную.

Последовательно обработав все проходы функции, мы получаем информацию о разбиении элементов памяти, используемых ее командами, на отдельные переменные.

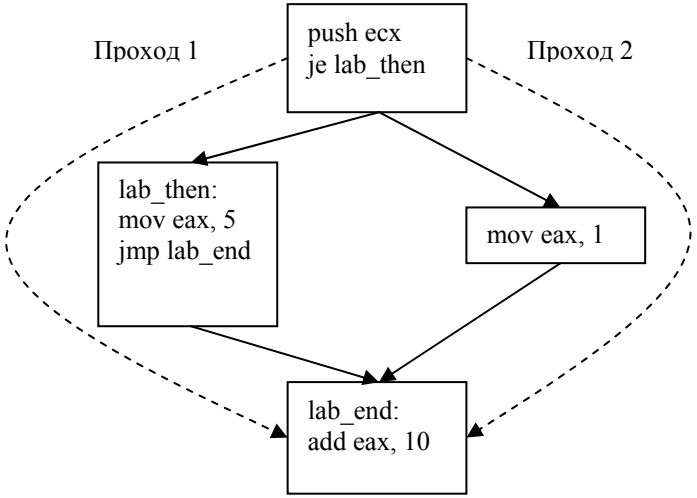

Рис 1. Варианты исполнения программы.

4. Заключение

В данной работе был предложен метод решения двух задач: задачи разделения элементов памяти на локальные переменные и задачи слияния информации, полученной при анализе нескольких проходов функции. Он реализован в виде плагина к среде TrEx.

В ходе дальнейших исследований планируется разработать алгоритм, который на основе информации, полученной при динамическом анализе, будет восстанавливать базовые и производные типы данных.

Список литературы

[1] Cifuentes, C. Reverse Compilation Techniques, PhD thesis, Queensland University of Technology, 1994.

[2] Долгова К. Н., Чернов А. В. Автоматическое восстановление типов в задаче декомпиляции. Программирование, номер 2, журнал Российской академии наук, Москва - 2009.

[3] A. Mycroft. Type-based decompilation. European Symp. on Programming, 1576:208 – 223, 1999.

[4] Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы. Принципы, технологии и инструментарий. Второе издание. «Вильямс» Москва - 2011

[5] В.А. Падарян, А.И. Гетьман, М.А. Соловьев. Программная среда для динамического анализа бинарного кода. Труды Института Системного Программирования. Том 16. 2009. стр. 51-72.