Проверка семантических ограничений языков C и C++ с помощью статического анализа

Игнатьев Валерий Николаевич

м.н.с.

Институт системного программирования РАН, Москва, Россия

E-mail: valery.ignatyev@ispras.ru

В работе предлагается формальная модель для представления ограничений языков программирования C и C++, которая позволяет задавать стилистические, синтаксические и семантические ограничения. Для этого используется аппарат логики предикатов, позволяющий определять ограничения на модели программы. Программа представляется в виде *аннотированного абстрактного семантического графа* (ААСГ) – структуры данных, основанной на абстрактном синтаксическом дереве, дополненном аннотациями в каждом узле и ребрами, например, из вершины, соответствующей использованию переменной к ее определению. Для каждого узла ААСГ задан его семантический тип, определяющий множество возможных атрибутов. Множество всех таких типов ограничено.

Для моделирования работы с памятью вводится понятие *область памяти* – декартово произведение класса памяти и её битового размера. Всего определено 10 классов памяти, объединенных в 4 группы: (1) стековая (автоматические переменные; параметры функций; память, выделенная `alloca`), (2) код (указатели на функции и метки), (3) данные (глобальные системные, глобальные статические и глобальные переменные), (4) динамическая память (выделенная `malloc` и `new`). Размер в битах обеспечивает поддержку битовых операций и структур с битовыми полями. Для массивов и структур использована *подобласть памяти* – декартово произведение базовой области памяти и битового размера. Состояние памяти программы задается в виде декартова произведения идентификатора оператора программы, контекста функции, который игнорируется для ускорения анализа в реализации, состояния переменных, т.е. соответствие переменной и её адреса и отображения адреса памяти и его значения. Для каждого выражения C и C++ определены «адрес» (l-value) и значение (r-value), которые по разработанным правилам вычисляются в соответствующие константы или области памяти.

Предложенная формализация позволяет построить однозначную классификацию ограничений в зависимости от требуемых для их за-

дания узлов и атрибутов ААСГ. В результате предложено 5 классов правил: (1) лексические; (2) синтаксические; (3) контекстные (ситуационные, т.е. требуется информация о состоянии нескольких узлов ААСГ); (4) контекстные, требующие дополнительного анализа (области памяти, анализ псевдонимов, циклов); (5) межмодульные. К последнему классу относятся правила, проверка которых производится на этапе линковки (связывания), требующие данных из нескольких модулей компиляции, например, анализ исключений.

Предложенные формализации использованы при реализации анализатора на основе открытых компиляторных инфраструктур LLVM/Clang[1]. Кроме того, были разработаны алгоритмы для эффективного поиска побочных эффектов, взаимных побочных эффектов, построения модели памяти и межмодульного анализа исключений. Работа анализатора осуществляется поэтапно, каждая следующая стадия обрабатывает более сложные и ресурсоемкие правила. Порядок проверки правил вычисляется автоматически на основе их определения. Также разработана возможность встраивания в популярные системы сборки проектов для обработки только тех исходных файлов, которые составляют результирующую программу. Лексическая, синтаксическая и семантическая стадии не генерируют ложных сообщений. На последних двух стадиях возможны ложные предупреждения из-за недостатков в алгоритмах анализа. Анализатор позволяет проверять более 50 различных правил. Измерение производительности показывает среднее замедление 22% по сравнению со сборкой проекта с отключенным анализом, что является приемлемым для использования при каждой сборке. Разработанный анализатор является частью статического анализатора Svace[2-3], который разрабатывается в ИСП РАН для промышленных коммерческих компаний.

Литература

1. Clang: a C/C++ frontend for LLVM. `http://clang.llvm.org/`

2. Иванников В. П., Белеванцев А. А., Бородин А. Е., Игнатьев В. Н., Журихин Д. М., Аветисян А. И. Статический анализатор Svace для поиска дефектов в исходном коде программ. // Труды ИСП РАН, т. 26, 2014, вып. 1, С. 231–249

3. Игнатьев В. Н. Использование легковесного статического анализа для проверки настраиваемых семантических ограничений языка программирования. // Труды ИСП РАН, т. 22, 2012, С. 169–187