# Анализ на основе аннотаций для практического поиска дефектов в программах и библиотеках, написанных на языке Си

***Бородин Алексей Евгеньевич***

*Соискатель*

*Институт системного программирования Российской академии наук, Москва, Россия*

*E-mail:* alexey.borodin@ispras.ru

Рассматривается проблема разработки масштабируемого алгоритма статического анализа для поиска ошибок в исходном коде. Уровень ложных срабатываний должен быть приемлемым, и алгоритм должен подходить для анализа библиотечного кода. Не ставится задачи нахождения всех возможных ошибок.

Анализ библиотечного кода создаёт дополнительные сложности по сравнению с анализом кода приложений, т.к. неизвестен контекст, в котором будут вызываться библиотечные функции. При написании функции, программист обычно накладывает ограничения на контекст, в котором функция может быть вызвана, при этом некоторые пути выполнения будут несуществующими.

Например, функция foo на рис. 1 имеет три аргумента, о взаимосвязи которых ничего не известно. Инструкция разыменования p с меткой (3) приведёт к ошибке только для некоторых путей выполнения. Если не предполагается вызов функции в контексте, где и x и z имеют ненулевое значение, и p имеет ненулевое значение в начале функции, то ошибка никогда не произойдёт. Таким образом одна и та же инструкция может приводить или не приводить к ошибке в зависимости от предполагаемого контракта функции.

```
void foo(int x, int y, int z) {
    if(x) {
        p = 0;       //(1)
        if(y) q = 0; //(2)
    }
    if(z) *p = 1;    //(3)
    *q = 0;          //(4)
}
```

Рис. 1: Проблема определения возможных путей выполнения

Предлагается критерий для выдачи предупреждений в функции, который не зависит от предполагаемого контракта функции. Целью является найти ошибку в функции самой по себе, которая может быть вызвана в любом предполагаемом контексте, включая те, которые не встречаются в анализируемой программе. Предполагается, что для каждого выполнимого пути, проходящего через функции, известно как определить, что путь содержит ошибку. Рассмотрим инструкцию в некоторой функции программы, такую, что все пути, проходящие через эту инструкцию содержат ошибку. При этом ошибка может произойти в любом месте, как до, так и после рассматриваемой инструкции. Если ни один из этих путей не возможен в соответствии с контрактом, то инструкция не может быть выполнена ни в каком корректном контексте. Поэтому инструкция будет бесполезной, и предупреждение об этом факте не будет лишним. Если же контракт включает некоторые из этих путей, то ошибка может случиться во время выполнения программы. Таким образом в обоих случаях будет полезным выдать сообщение. Если возможно, что инструкция может быть выполнена без ошибки, мы не выдаём предупреждений, чтобы избежать ложных срабатываний.

Поясним использование критерия для выдачи предупреждений для функции foo на рис. 1. Функция содержит две потенциальные ошибки разыменования нулевых указателей: для p и q. Применим критерий для выдачи предупреждения для инструкции с меткой (2). Все пути выполнения, проходящие через эту инструкцию, также проходят через инструкция разыменования (4), и на всех этих путях происходит разыменование нулевого указателя в точке (4). Поэтому предупреждение будет выдано. Ошибка не обязательно произойдёт во время выполнения, но в этом случае разыменование не будет выполнено в любом контектсе, и желательно исправить это.

Применяя критерий для инструкции присваивания с меткой (1), получим, что существует путь (где z равно нулю и q не равно), который не содержит ошибку. Поэтому предупреждение не выдаётся.

Описанный критерий применяется для всех инструкций во всех функциях.

Был разработан алгоритм анализа, основанный на аннотациях, использующий вышеописанный критерий. Использование анализа, основанного на аннотациях, становится широко распространённой практикой [1]. Этот подход предполагает выполнение межпроцедурного анализа масштабируемым способом таким образом, что общая стоимость вычислений не намного выше, чем необходимо для пооче-

рёдного анализа отдельных функций. Аннотации функций используются в анализе инструкций вызова, чтобы моделировать поведение функции. Для анализа алиасов также как и в [2] используется предположение, что аргументы функции и глобальные переменные указывают на взаимно непересекающиеся области памяти, что позволяет существенно снизить сложность анализа алиасов.

Реализация алгоритма была включена в статического анализатор Svace [3–4]. Было реализована несколько детекторов, включая поиск ошибок разыменования нулевых указателей, переполнения буфера, использования неинициализированных переменных. Разные детекторы соответствуют разным видам ошибок, которые могут произойти при выполнении конкретного пути.

Инструмент использовался для просмотра ошибок в 23 проектах с открытым исходным кодом общим размером более 4 миллионов строк кода, включая busybox, ffmpeg, libxml2, openssl, sqlite3. Анализ занял 66 минут. Всего было выдано 918 предупреждений. Мы вручную проверили выборку из 102 предупреждений, 77% из которых оказались истинными срабатываниями. Некоторые из предупреждений, отмеченные как истинные, соответствуют ошибкам в исходном коде, которые не обязательно произойдут во время выполнения программы. Эти предупреждения соответствуют инструкциям, которые не могут быть выполнены ни в каком контексте, что также желательно исправить.

**Литература**

1. Aho A. V.,Lam M.,Sethi R.,Ullman J. D. Compilers: Principles, Techniques, & Tools with Gradiance. 2007.

2. Livshits V. B.,Lam M. Tracking pointers with path and context sensitivity for bug detection in C programs. // In ACM SIGSOFT Software Engineering Notes. Vol. 28. No. 5. ACM. 2003.

3. Аветисян А. И., Белеванцев А. А., Бородин А. Е., Несов В. С. Использование статического анализа для поиска уязвимостей и критических ошибок в исходном коде программ. // Труды Института системного программирования РАН 21. 2011.

4. Иванников В. П., Белеванцев А. А., Бородин А. Е., Игнатьев В. Н., Журихин Д. М., Аветисян А. И., Леонов М. И. Статический анализатор Svace для поиска дефектов в исходном коде программ. // Труды Института системного программирования РАН 26. 2014.