Опубликовано в материалах 2-й межвузовской научной конференции по проблемам информатики СПИСОК-2011, с. 380-384.

А. В. Купцов
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Вывод nullness-контрактов из исходного кода с помощью графа потока управления

Развитие разработки программного обеспечения всегда было направлено на уменьшение ее стоимости. Эффективным способом удешевления разработки является создание средств для разработчика, помогающих устранять программные ошибки на раннем этапе разработки, а, как известно, стоимость устранения ошибки тем меньше, чем раньше она будет обнаружена [1].

Одним из распространенных видов ошибок, которые тяжело обнаружить на стадии разработки, являются ошибки, связанные с обращением по нулевой ссылке [2]. Существуют специальные анализаторы, позволяющие выявлять такие ошибки [3]. В данной работе рассматривается анализатор для языка C#, основанный графе потока управления, позволяющий выявлять некоторые такие ошибки. Его алгоритм требует для лучшей своей работы контракты вида «может ли метод вернуть или принять в качестве аргумента значение null». Такие контракты называются nullness-контрактами. В данной работе предлагается метод вывода nullness-контрактов, предназначенных для рассмотренного анализатора.

Граф потока управления

Если представить операции, совершаемые компьютером при выполнении программы, как узлы, а возможные переходы между операциями — как ребра, их соединяющие, то

получится граф потока управления [4]. Из-за различных языковых конструкций, используемых в разных языках, а также целей применения графа, его элементы могут иметь различные свойства. Дуги графа, рассматриваемого в данной работе, имеют описанные ниже свойства.

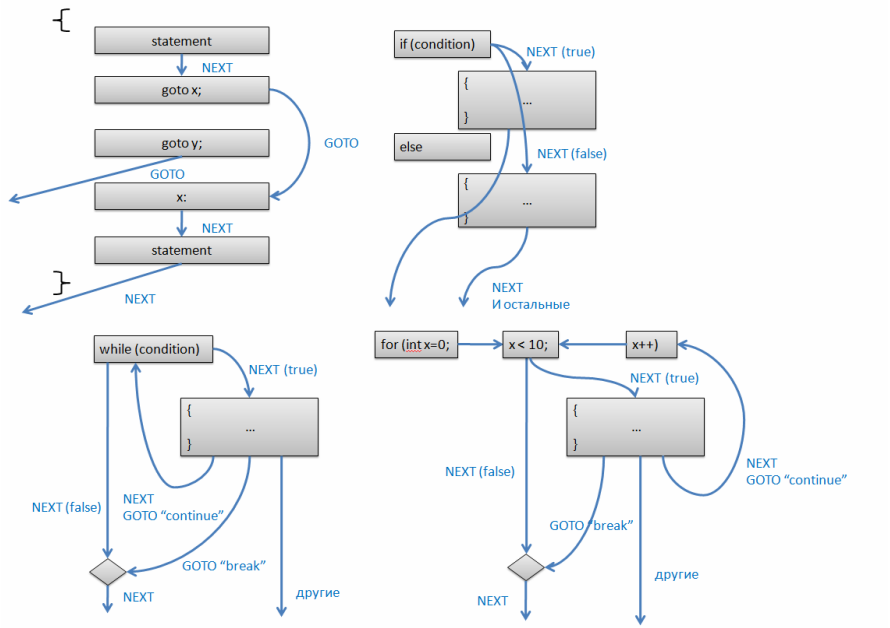

Рис. 1. Построение графа для последовательности операторов, операторов if, while, for.

Каждая дуга графа помечена типом перехода (рис. 1). В языке C# переходы между операциями можно поделить на четыре типа:

- NEXT — обычный переход (на следующее утверждение, на следующий вызов метода или оператора, в текущем утверждении или выражении и т. д.);
- GOTO — переход по операторам goto, break, continue;
- RETURN — штатный выход из метода с передачей управления в вызывающую подпрограмму;
- THROW — «аварийный» выход из метода (с броском исключения).

Некоторые NEXT-переходы помечаются еще одним свойством: при выходе из булевого выражения добавляется значение true или false. Оно назначается в зависимости от направления, в котором произойдет соответствующий переход, если это выражение будет вычислено в то или иное значение.

С помощью графа потока управления можно делать простые статические анализы кода. Например, сообщения компиляторов и интегрированных средств разработки о недостижимом коде строятся на основании недостижимых узлов графа.

Nullness-анализ

Для динамического анализа исходного кода с помощью графа потока управления с каждой его дугой ассоциируется некоторое состояние анализатора. При обходе графа в каждом узле графа объединяются состояния всех входящих дуг с непустым состоянием, и формируются состояния исходящих дуг. Этот процесс останавливается, когда состояния перестанут меняться.

Nullness-анализ — это эвристический динамический анализ кода, призванный выявить возможные места в программе, где происходит обращение по нулевой ссылке (в терминах C# — NullReferenceException). Также это анализ позволяет выявлять другие возможные ошибки в коде. Так как этот анализ эвристический, он по своей природе не может быть точным на 100%. Поэтому различают два подхода: оптимистический — выявлять только те места в программе, где ошибка вероятность ошибки максимальна, и свести к минимуму ложные срабатывания, а также пессимистический — выявлять все места, где ошибка возможна. В данной работе рассматривается только оптимистический вид анализа.

В данном анализе подконтрольная переменная может находиться в следующих состояниях: UNKNOWN (состояние, введенное специально для оптимистического анализатора), TRUE, FALSE (оба — соответствующие состояния булевых

переменных), NULL, NOT-NULL (оба — состояния ссылочных переменных, имеющих соответственно значение null или отличное от него). Вектор состояний — это набор состояний, в котором каждой подконтрольной переменной сопоставлено ровно одно состояние. Множество векторов состояний — неупорядоченный набор различных векторов состояний. Такое множество определяет возможные комбинации состояний переменных на ребре графа перехода. В зависимости от языка программирования каждому типу узла сопоставляется правило создания множества векторов состояний для выходного ребра. Такого рода правила заключаются в том, что множества векторов состояний, полученных на входе в узел, объединяются, а затем изменяются по определенному критерию. Например, при присваивании в переменную значения, полученного в результате выполнения оператора создания экземпляра класса, состояние этой переменной во всех векторах изменяется на NOT-NULL. А при сравнении значений двух ссылочных переменных в NEXT(true)-ребро попадут вектора состояний, имеющие одинаковые состояния сравниваемых переменных, а в NEXT(false)-ребро — остальные. Для предопределенных операторов не сложно составить набор правил изменения состояний на ребрах, но уже при использовании методов нам не обойтись без контрактов вида «этот метод может вернуть null», «этот метод никогда не возвращает null», «в данный параметр нельзя подставить null». Контракты такого рода называются nullness-контрактами (или nullness-аннотациями).

Обойдя граф потока управления с помощью этого алгоритма можно найти места в коде, где высока вероятность ошибки использования ссылки со значением null. Например, если при обращении к объекту по ссылке окажется, что один из векторов состояний переменных на ребре, входящим в узел, соответствующий этому обращению, содержит состояние NULL для данной ссылки, то существует такой путь в программе, при выполнении по которому значение в переменной будет null. А значит, разработчика можно известить о его возможной ошибке предупреждением вида

«Возможно выбрасывание исключения NullReferenceException»

Вывод Nullness-контрактов

В данной работе вывод nullness-контрактов для краткости будет называться аннотированием. Анализатор, занимающийся аннотированием, называется аннотатором. Метод (или другая сущность программы, например, свойство, индексатор, аксессор, поле класса и т. д.) называется аннотированным, если он помечен nullness-контрактами.

Nullness-анализ может быть использован для аннотирования возможности возвращения значения null методом или иной сущностью программы. Если считать, что метод, в котором выводятся контракты, использует уже аннотированные сущности, то в результате анализа его графа потока управления мы можем сделать вывод о том, может ли он вернут null. Для этого рассматриваются все его RETURN-ребра. Если существует вектор на RETURN-ребре с состоянием возвращаемого значения NULL, то метод помечается контрактом [CanBeNull] (здесь и далее в квадратные скобки обозначают контракты, обозначение введено, чтобы не было путаницы с обозначением состояний на ребрах графа потока управления), если все RETURN-ребра содержат только состояния NOT-NULL возвращаемого значения, то метод помечается контрактом [NotNull]. Аналогично выводятся контракты для out-параметров и постусловий ref-параметров.

Чтобы аннотировать предусловие для параметра (например, для обычного параметра или ref-параметра) используется подход, при котором метод анализируется дважды: первый анализ проводился с начальными состояниями параметра UNKNOWN, второй — с состоянием NULL. Затем графы, полученные в результате этих двух анализов, сравниваются. Если во втором графе существуют выходные ребра THROW, отсутствующие в первом графе, то значение null, подставленное в качестве аргумента этого

параметра приведет к исключению (в сравнении с ненулевым значением). А значит, необходима проверка значения перед передачей его в метод. Такие параметры помечаются контрактом [NotNull].

Если не учитывать полиморфизм, то аннотировать сущности в библиотеке можно итеративно следующим образом: на каждой итерации аннотируем все сущности программы, остановка алгоритма происходит, когда на очередной итерации не удается вывести новые контракты. Данный алгоритм сходится, потому что на все предусловия и постусловия могут только усиливаться при очередном проходе аннотатора, а количество возможных предусловий и постусловий ограничено.

При полиморфизме возникает вопрос, что делать с базовыми сущностями, если конкретные аннотируются по-разному. Чтобы контракты сущностей полиморфной иерархии были согласованы, необходимо соблюдать следующее:

- Предусловия конкретных параметров всегда не сильнее предусловий базовых.
- Постусловия конкретных сущностей всегда не слабее постусловий базовых.

Сила условий определяется в следующем порядке: отсутствие контракта (Unknown), [CanBeNull], [NotNull]. Ввиду того, что постусловия сущностей не могут быть сильнее постусловий конкретных, а так же того, что аннотатор не может ослаблять условия, важен порядок аннотирования сущностей. Если использовать порядок «сначала конкретные сущности, затем базовые», появляются похожие проблемы в выводе несогласованных по иерархии предусловий. Поэтому алгоритм был усовершенствован следующим образом. Каждая итерация аннотатора делится на две части: в первой аннотатор обходит сущности в порядке «сначала конкретные, затем базовые», аннотируя только постусловия, а во второй — аннотируя только предусловия в порядке «сначала базовые, затем конкретные сущности».

Использование согласования сущностей в полиморфных иерархиях позволяет проаннотировать на 37% больше

сущностей, чем аннотатор, который не учитывает несогласованные сущности, а просто не выводит их в результат. Сравнение проводилось на стандартных библиотеках .NET Framework.

Список используемых источников

1. *Билл Грэхем (Bill Graham), Пол Леру (Paul N. Leroux), Тодд Лендри (Todd Landry)* Использование статического и динамического анализа для повышения качества продукции и эффективности разработки. QNX Software Systems, Klocwork. http://www.swd.ru/index.php3?pid=828
2. *Knute Axelson,Mary Bellino,Dave Harper,Dave Iffland* Coding: The Handbook for Information Technology. Blue Line Press Inc., 2005.
3. *Arnout F.M. Engelen* Nullness Analysis of Java Source Code. Al-Imam Muhammad ibn Saud Islamic University, Master's thesis. 2006.
4. *Thomas William Reps, Mooly Sagiv, Jorg Bauer* Program analysis and compilation, theory and practice. Springer-Verlag, 2007.