Автоматическое определение выполнимости наборов формул для операций сравнения

С. В. Зеленов, С. А. Зеленова
{zelenov, sophia}@ispras.ru

Аннотация. В статье излагается алгоритм автоматического определения реализуемости данного набора формул-сравнений. Алгоритм применяется в инструментах, реализующих технологию UniTESK. Его использование позволяет снизить трудозатраты на разработку спецификаций и повысить точность подсчета достигнутого тестового покрытия.

1. Введение

Масштабы современного программного обеспечения (ПО) уже не позволяют обеспечить его качественную разработку без привлечения эффективных средств автоматизации создания тестов. Развиваемая в ИСП РАН технология UniTESK [1,3,5] тестирования функциональности ПО основана на использовании формальных моделей. Для автоматизации подготовки тестов и анализа результатов в UniTESK требования к ПО представляются в виде формальных спецификаций в форме пред- и постусловий (см. на Рис. 1 пример спецификации операции извлечения квадратного корня). На основе спецификаций автоматически генерируются тестовые оракулы, которые используются для проверки корректности реакции целевого ПО в ответ на единичное тестовое воздействие (вызов одной операции с некоторыми аргументами). Тесты строятся путем перебора операций и итерации наборов аргументов для каждой операции.

Для оценки качества тестирования в UniTESK используются критерии покрытия спецификаций, автоматически извлекаемые из структуры последних. Базовым критерием является покрытие ветвей функциональности, каждая из которых помечается в спецификации оператором `branch` и соответствует тем ситуациям, в которых операция ведет себя «одинаково» (в примере на Рис. 1 выделены две ветви функциональности: когда аргумент нулевой и когда он положительный).

Наиболее детальным является критерий покрытия комбинаций элементарных условий. Он определяется всеми возможными комбинациями значений

элементарных логических формул, использованных в условиях ветвлений спецификации данной операции. Этот критерий является аналогом критерия покрытия комбинаций условий (multiple condition coverage, см. [4], [6]) для покрытия кода. В примере на Рис. 1 имеются две элементарные формулы:

```
f1 ≡ a >= 0,
f2 ≡ a == 0.
```

Однако при тестировании, нацеленном на достижение высокого уровня покрытия комбинаций элементарных условий, возможны проблемы, связанные с недостижимостью некоторых комбинаций в силу наличия семантических связей между элементарными формулами. А именно, если генератор отчетов не учитывает информацию о наличии недостижимых комбинаций условий, то достигнутое в результате тестирования покрытие будет им подсчитано некорректно, и в итоге в отчет о тестировании попадет заниженное значение реально достигнутого покрытия. Низкое значение достигнутого покрытия, фигурирующее в отчете, обычно служит стимулом для тестировщика к исследованию причин низкого покрытия и к попыткам его повысить, однако в случае некорректного подсчета достигнутого покрытия эта работа оказывается напрасной тратой времени.

```
specification int sqrt( double a )
{
    pre { return a >= 0; }
    post
    {
        if( a == 0 )
        {
            branch "zero";
            return sqrt == 0;
        } else {
            branch "positive";
            return a > 0 && abs( (sqrt*sqrt – x)/x ) < eps;
        }
    }
}
```

Рис. 1. Спецификация операции извлечения квадратного корня.

Так, в примере на Рис. 1 недостижимой является следующая комбинация значений элементарных формул:

```
{ f1 = false, f2 = true }.
```

В UniTESK такие проблемы решаются при помощи описания имеющихся семантических связей в виде логических выражений-тавтологий, построенных из элементарных формул и являющихся тождественно истинными в силу смысла этих формул. Так, в примере на Рис. 1 тавтология может быть описана следующей конструкцией:

```
tautology !( !(a >= 0) && a == 0 );
```

Кроме подобных естественных семантических связей между элементарными формулами, которые следуют из свойств операции сравнения, бывают ситуации, когда семантические связи между элементарными формулами имеют менее явный характер. Так, в примере на Рис. 2 в первом условии проверяется, что месяц, указанный в аргументе-дате, является февралем, а во втором условии проверяется, что число месяца не меньше 30-го. Поскольку в феврале всегда не более 29 дней, здесь необходимо описать следующую тавтологию:

```
tautology !( d.month() == FEB && d.day() >= 30 );
```

Как показывает практика, в реальных спецификациях достаточно большую долю тавтологий составляют именно естественные тавтологии, описывающие семантику операций сравнения. Ручное перечисление таких тавтологий является нетворческой, неинтересной, рутинной работой и может приводить к ошибкам в их описании в спецификации.

В настоящей статье мы излагаем алгоритм для автоматического определения того, имеются ли при данных значениях элементарных формул нарушения естественных тавтологий, описывающих семантику операций сравнения. Алгоритм основан на анализе структуры элементарных формул и не требует явного перечисления всех таких тавтологий.

Спецификация	Элементарные формулы

`specification int m( Date d )`
`{`
`  post`
`  {`
`  if( d.month() == FEB )`
`  {`
`  ...`
`  } else if( d.day() >= 30 )`
`{`
`  ...`
`  } else {`
`  ...`
`  }`
`  }`
`}`
	`f1 ≡ d.month() == FEB`
	
	`f2 ≡ d.day() >= 30`

Рис. 2. Спецификация с наличием неявной семантической связи между формулами.

2. Постановка задачи

Пусть в спецификации тестируемой операции имеются в том числе элементарные формулы-сравнения, т.е. формулы следующих видов:


```
a  < b,
a  > b,
a <= b,
a >= b,
a == b,
a != b,
```

где `a` и `b` — аргументы элементарных формул. Будем считать два аргумента одинаковыми только если совпадают их текстуальные представления. Кроме того, будем считать все аргументы переменными (т.е. будем игнорировать информацию об известных значениях константных аргументов).

При перечислении комбинаций значений формул для спецификации тестируемой операции всем элементарным формулам последовательно присваиваются все возможные наборы логических значений. Каждый фиксированный набор значений элементарных формул индуцирует некоторые конкретные отношения между аргументами формул-сравнений.

Для проверки истинности естественных тавтологий, описывающих семантику операций сравнения, на фиксированном наборе отношений между аргументами формул-сравнений требуется определить, является ли реализуемой такая система формул, т.е. существуют ли такие значения аргументов, при котором выполнялись бы все соответствующие отношения между ними.

Нереализуемость данной системы формул означает нарушение некоторой естественной тавтологии, описывающей семантику операций сравнения, и как следствие — недостижимость соответствующей комбинации значений формул.

3. Нереализуемые ситуации

Пусть дан некоторый набор формул-сравнений. Без ограничения общности можно считать, что каждая формула имеет один из следующих видов:

```
a  > b,
a >= b,
a == b,
a != b.
```

Рассмотрим, какие могут встречаться нереализуемые ситуации для системы таких формул.

1. Пусть в наборе формул имеется неравенство:

   ```
   a != b,
   ```

   и одновременно имеется поднабор формул (см. ниже), из которого следует равенство:

$$a == b.$$

Будем называть такую ситуацию *нарушением первого рода*.

Возможные виды формул, влекущих равенство переменных a и b суть следующие:

a. цепочка равенств, в которой участвуют a и b:

$$a == x_1 == \ldots == x_n == b;$$

b. две цепочки отношений, в которых участвуют a и b:

a [*rel*] y_1 [*rel*] ... [*rel*] y_k [*rel*] b,

b [*rel*] z_1 [*rel*] ... [*rel*] z_m [*rel*] a,

где каждый [*rel*] — это отношение из набора { >=, == }.

2. Пусть в наборе формул имеется строгое неравенство:

$$a > b,$$

и пусть имеется поднабор формул, индуцирующий неравенство:

$$b >= a,$$

а именно, цепочка сравнений, в которых участвуют a и b:

b [*rel*] x_1 [*rel*] ... [*rel*] x_n [*rel*] a,

где каждый [*rel*] — это отношение из набора { >, >=, == }.

Будем называть такую ситуацию *нарушением второго рода*.

Верна следующая

Теорема о реализуемости. Если данная система формул не содержит нарушений первого и второго рода, то возможна интерпретация этой системы в целых числах.

Доказательство этой теоремы будет приведено ниже. Из теоремы следует, что при игнорировании информации об известных значениях константных аргументов все возможные нереализуемые ситуации исчерпываются нарушениями первого и второго рода.

4. Используемая модель

Для изложения алгоритма проверки того, что данная система формул-сравнений является реализуемой, мы используем модель в виде раскрашенного ориентированного графа. Вершины графа соответствуют

аргументам формул. Дуги и ребра[1] графа, покрашенные в разные цвета, соответствуют собственно отношениям между аргументами:

- отношение строгого порядка (>) представляется красной дугой от меньшего аргумента к большему;
- отношение нестрогого порядка (>=) представляется синей дугой от меньшего аргумента к большему;
- отношение неравенства (! =) представляется черным ребром;
- отношение равенства (==) представляется зеленым ребром.

При этом приведенные выше нереализуемые ситуации представляются следующими подграфами:

- ситуация 1.a соответствует циклу, который содержит только несколько зеленых ребер и ровно одно черное ребро;
- ситуация 1.b соответствует ориентированному циклу, который содержит только синие дуги и зеленые ребра, и некоторые две вершины которого кроме того связаны черным ребром;
- ситуация 2 соответствует ориентированному циклу, который содержит несколько синих дуг и зеленых ребер, а также не менее одной красной дуги.

Заметим, что ориентированный цикл из ситуации 1.b означает равенство всех аргументов, соответствующих его вершинам.

5. Алгоритм проверки реализуемости

Пусть дан граф, представляющий некоторую систему формул-сравнений. Алгоритм проверки реализуемости этой системы состоит из следующих этапов:

1. Отождествление (склеивание) всех пар вершин, связанных зелеными ребрами (поскольку зеленые ребра означают равенство аргументов).

   В результате:
   - нереализуемая ситуация 1.a станет соответствовать черной петле;
   - ориентированный цикл из нереализуемой ситуации 1.b станет содержать только синие дуги.

2. Поиск всех ориентированных циклов, составленных только их синих дуг, и отождествление для каждого такого цикла всех входящих в него вершин ввиду равенства соответствующих им аргументов (см. замечание в конце предыдущего раздела).

[1] Ребра считаются неориентированными, а дуги – ориентированными.

В результате нереализуемая ситуация 1.b станет соответствовать черной петле. Таким образом, исходная система формул имеет нарушение первого рода тогда и только тогда, когда в образовавшемся на втором этапе графе имеется черная петля.

3. Поиск черной петли.
   Если черных петель в графе нет, то исходная система формул может иметь только нарушения второго рода. Ввиду отсутствия в графе ориентированных циклов, составленных только из синих дуг (они исчезли на втором этапе), любой ориентированный цикл, составленный только из дуг, автоматически будет содержать хотя бы одну красную дугу. Таким образом, исходная система формул имеет нарушение второго рода тогда и только тогда, когда в образовавшемся на третьем этапе графе имеется ориентированный цикл, составленный только из дуг.

4. Поиск ориентированного цикла, составленного только из дуг.
   Если таких циклов в графе нет, то исходная система формул не имеет нарушений ни первого, ни второго рода.

Заметим, если система формул не имеет нарушений, то соответствующий ей граф, образовавшийся после работы приведенного здесь алгоритма, обладает следующими свойствами:

- граф составлен только из дуг и черных ребер;
- ни одно из черных ребер не образуют петли;
- подграф, образованный выбрасыванием всех черных ребер, ацикличен.

6. Доказательство теоремы о реализуемости

Пусть дана система формул-сравнений, которая не содержит нарушений первого и второго рода. Рассмотрим соответствующий этой системе граф, образовавшийся после работы алгоритма, приведенного в предыдущем разделе. Будем строить интерпретацию исходной системы в целых числах путем присвоения вершинам графа некоторых целых значений так, чтобы удовлетворялись все имеющиеся в графе отношения. Поскольку все аргументы исходных формул являются переменными, мы можем назначать вершинам графа произвольные значения.

Заметим, что поскольку в графе нет черных петель, т.е. все черные ребра соединяют разные вершины, то для того, чтобы удовлетворить всем имеющимся в системе отношениям неравенства (! =), которые представляются черными ребрами, достаточно присваивать всем вершинам различные значения.

Рассмотрим ациклический подграф, образованный выбрасыванием всех черных ребер. По определению дуг, использующихся в нашей модели, каждая синяя или красная дуга ведет от вершины, соответствующей меньшему аргументу, к вершине, соответствующей большему аргументу. Таким образом, для того, чтобы удовлетворить всем имеющимся в системе отношениям порядка, достаточно в качестве искомого целого значения присвоить каждой вершине ее номер в списке, полученном в результате топологической сортировки [2] данного подграфа.

Теорема доказана.

7. Учет целочисленных констант

Пусть в данном наборе элементарных формул для некоторых аргументов известны их константные числовые значения. Тогда если известны также типы аргументов (целочисленные или типы чисел с плавающей точкой), то даже при отсутствии нарушений первого и второго рода данная система формул может не быть реализуемой.

Рассмотрим следующий пример. Пусть для целого `a` среди элементарных формул имеются следующие:

```
a > 1,
a < 2.
```

Тогда комбинация значений `true` для обеих этих формул, недостижима.

Более общей ситуацией является цепочка отношений:

$$N\ [rel]\ x_1\ [rel]\ \ldots\ [rel]\ x_k\ [rel]\ M,$$

где все аргументы целые, `N` и `M` — константы, а каждый [*rel*] — это отношение из набора { >, >=, == }. В этом случае система формул не является реализуемой, если количество отношений строгого порядка (>) в цепочке больше, чем разность `(N − M)`, поскольку в этом случае между `N` и `M` не найдется достаточно различных целых чисел, чтобы присвоить их в качестве значений всем аргументам x_i из цепочки.

Для эффективного перечисления всех таких цепочек отношений удобно использовать ациклический подграф, получающийся выбрасыванием всех черных ребер из графа, образовавшегося после окончания работы алгоритма проверки наличия нарушений первого и второго рода.

Рассмотрение ситуаций, когда в цепочке какой-то из аргументов не является целым, существенно усложняется в связи с дискретностью представления на ЭВМ чисел с плавающей точкой и непредставимостью в этом типе некоторых (в том числе целых) чисел. В частности, существуют такие последовательные числа[2] с плавающей точкой `X` и `Y`, что между ними есть число, представимое

[2] Т.е. такие, между которыми не существует других чисел того же типа.

как целое, но не как число с плавающей точкой, например, $3 \cdot 10^7$ представимо в виде числа с плавающей точкой однократной точности, а следующим таким представимым числом является $3 \cdot 10^7 + 2$. Поэтому система формул $3 \cdot 10^7 < a$ и $a < 3 \cdot 10^7 + 2$ невыполнима, если a имеет тип числа с плавающей точкой однократной точности и выполнима, если a — 32-битное целое. Анализ таких ситуаций мы оставляем за рамками настоящей статьи.

8. Заключение

В работе предложен алгоритм поиска в системе элементарных формул нарушений естественных тавтологий, описывающих семантику операций сравнения. Алгоритм основан на анализе структуры элементарных формул и не требует явного перечисления всех таких тавтологий.

Алгоритм успешно используется в инструментах, реализующих технологию тестирования UniTESK [1,3,5]. Он позволяет повысить точность автоматического измерения достигнутого тестового покрытия, а также дает возможность тестировщику не писать многочисленные тавтологии, описывающие взаимосвязи между имеющимися в спецификации формулами сравнения, что позволяет существенно сократить ручные трудозатраты и количество ошибок при написании спецификаций целевого ПО.

Литература

[1] А. В. Баранцев, И. Б. Бурдонов, А. В. Демаков, С. В. Зеленов, А. С. Косачев, В. В. Кулямин, В. А. Омельченко, Н. В. Пакулин, А. К. Петренко, А. В. Хорошилов. Подход UniTesK к разработке тестов: достижения и перспективы. Труды ИСП РАН, 5:121–156, Москва, 2004.

[2] Т. Кормен, Ч. Лейзерсон, Р. Ривест. Алгоритмы: построение и анализ. М.: МЦНМО, 1999.

[3] В. В. Кулямин, А. К. Петренко, А. С. Косачев, И. Б. Бурдонов. Подход UniTesK к разработке тестов. Программирование, 29(6):25–43, 2003.

[4] A. P. Mathur. Foundations of Software Testing. Copymat Services, 2006.

[5] UniTESK. http://www.unitesk.ru/

[6] H. Zhu, P. A. V. Hall, J. H. R. May. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 29(4):366–427, Dec. 1997.