Случайные тесты с перебором классов инструкций

А.С. Куцаев[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, koutsaev@niisi.msk.ru, +7 916 174-61-16

Аннотация. Преимущество генераторов случайных тестов в простоте применения и возможности выдачи больших объемов кода. Стохастическое тестирование выявляет как обычные, так и специфические ошибки (редкие сочетания и недочеты спецификации). При этом увеличение покрытия теста не зависит прямо от управления генератором. Перебор размещений из *n* классов инструкций по *k* позволяет получить покрытие по соответствующим комбинациям инструкций при приемлемых размерах кода.

Ключевые слова: случайные тесты, покрытие, верификация, перебор классов инструкций

1. Введение

Стохастическое тестирование является существенной составляющей средств функциональной верификации [1], [2]. Оно позволяет выявить ошибки проекта, которые трудно обнаружить другими способами. Преимущество стохастического тестирования в возможности получать тестовый код большого объема. При достаточном разнообразии кода это позволяет вырабатывать тестовые ситуации, которые трудно предусмотреть. Однако большой объем теста не гарантирует эффективности: например, может оказаться, что некоторые сочетания инструкций не возникают вообще. Необходимо, чтобы с увеличением объема росло и покрытие теста, определенное каким-либо способом. Эффективность теста зависит от большого числа параметров, выбор которых рассматривается в данной работе.

В генераторах семейства tergen, разработанных во ФГУ ФНЦ НИИСИ РАН [3], [4], используется два основных подхода к выработке тестовых ситуаций:

- Массовая выдача случайных инструкций со случайными аргументами. Для нее задаются только режимы генерации и вероятности выбора.
- Создание тестовых ситуаций на основе описания, полученного при анализе ранее возникших проблем. Возможно частичное и полное задание фрагментов кода наряду со случайным выбором.

Исходные данные для генерации теста – это шаблон, который включает задание управляющих параметров и программу построения теста (заготовку тестового кода) на интерпретируемом языке. В предыдущей работе [5] подробно описаны возможности некоторых средств генератора тестов для создания специальных тестовых ситуаций. Цель настоящей работы – описание методики создания тестов большого объема. Эта задача определяет требования как к программе построения теста, так и к управляющим параметрам.

В главе 2 обсуждаются вопросы получения покрытия при различных вариантах перебора. В главе 3 приводится пример классификации инструкций и возможные нарушения порядка классов при переборе. Глава 4 посвящена возможностям настройки генератора для инструкций определенных классов. В главе 5 приведены выводы.

Далее T32 и T64 – это обозначения версий генератора тестов для 32- и 64-битового процессора, соответственно.

2. О переборе и покрытии

Покрытие теста можно определять разными способами. При случайном тестировании сложно указать зависимость величины покрытия от управляющих параметров генератора тестов. Для выбора этих параметров можно дать лишь самые примитивные рекомендации. Например, инструкции могут выбираться случайно, независимо друг от друга, согласно заданным вероятностям выбора. Чтобы тестировать инструкции, связанные с некоторой частью проекта, можно увеличить вероятности выбора этих инструкций, однако говорить о связи с каким-либо покрытием при этом нельзя.

В случае перебора по комбинациям инструкций покрытие по ним обеспечивается автоматически. Например, при тестировании используются переборы всех пар инструкций "каждая за каждой", всех троек и т.п. Такой подход оправдан, поскольку некоторые ошибки выявляются при определенном сочетании двух, трех и более инструкций. Однако, когда число инструкций процессора составляет несколько сотен, даже од-

нократный перебор всех допустимых троек инструкций дает значительный объем кода. При этом однократного перебора недостаточно минимум по двум причинам: неизбежные нарушения порядка следования инструкций, а также произвольность выбора их аргументов. Это делает перебор, скажем, всех четверок инструкций мало реальным.

Способ уменьшить объем кода при переборе за счет введения дополнительной случайности состоит в следующем. Все инструкции объединяются в классы по сходству тех или иных признаков. Например, класс может включать инструкции, использующие устройство целого умножения-деления. Затем вместо перебора самих инструкций делается перебор их классов, в ходе которого из очередного класса случайно выбирается одна инструкция. Такой способ делает реальным неоднократный перебор всех троек и четверок классов. Предполагается, что индивидуальные особенности отдельных инструкций в классе мало влияют на взаимовлияние их выполнения. Главная идея перебора - создание тестовых ситуаций за счет сочетания инструкций разного вида – сохраняется.

На первый взгляд порядок перебора неважен, лишь бы он был полным. Однако это не совсем так. Равномерность распределения инструкций каждого класса по последовательности также имеет значение. Например, если большинство регистровых переходов собрано в одной части кода, инструкции загрузки целевых адресов будут сильнее искажать результат перебора.

Рассмотрим этот вопрос подробнее. Пусть есть множество из n объектов (классов инструкций). Для наглядности будем представлять классы их номерами от 0 до n-1. Рассмотрим перебор, т.е. последовательность всех размещений с повторением из n по k, где k - целое, от 2 и выше (случай k=1 допустим, но интереса не представляет). Число размещений в этой последовательности $N = n^k$. Для размещения $\{i_0, i_1, \dots i_{k-1}\}$ введем номер i, задаваемый выражением

$$i = i_0 n^{k-1} + i_1 n^{k-2} + \cdots + i_{k-1} \qquad (1)$$

Последовательность всех размещений из n классов по k, упорядоченную по возрастанию номеров, объявим основной. Раскрывая скобки, перейдем к последовательности классов, длина которой $k \times N$. Если ее зациклить, можно заметить, что каждая подпоследовательность длины k встречается в ней ровно k раз (далее для краткости будем называть это свойством k вхождений). Есть и более сложные способы перебора, например, последовательность де Брёйна, в которой каждая подпоследовательность длины k встречается ровно один раз. Однако при случайном тестировании, где требуется именно неоднократное повторение комбинаций, такое усложнение лишено смысла.

В основной последовательности классы распределены неравномерно. Рассмотрим, например, размещения из 4 классов по 2, т.е. все пары классов с номерами 0, 1, 2, 3.

0,0, 0,1, 0,2, 0,3, 1,0, 1,1, 1,2, 1,3,
2,0, 2,1, 2,2, 2,3, 3,0, 3,1, 3,2, 3,3,...

В каждой четверти этой последовательности один из классов присутствует 5 раз, а остальные - по одному. Желательно иметь более равномерное распределение классов, но с сохранением "свойства k вхождений".

Зададим перестановку основной последовательности в виде $j = j\,(i)$,

$$j(i) = (i \times M)_{mod\ N}. \qquad (2)$$

где i и j - порядковые номера размещений в последовательности до и после перестановки, а M - любое число, взаимно простое с N (или с n, что то же). Легко показать, что обратная перестановка имеет тот же вид:

$$i(j) = (j \times L)_{mod\ N}. \qquad (3)$$

где L находится из соотношения $(L \times M)_{mod\ N} = 1$. Непосредственная проверка для значений n и k, представляющих интерес, показывает, что такие перестановки сохраняют "свойство k вхождений".

Оценка равномерности распределения номеров для разных M делается стандартно. Выберем номер класса i, разобьем последовательность на равные части, найдем число номеров i в каждой части и оценим среднеквадратичное отклонение (СКО). Оказывается, что у основной последовательности ($M = 1$) оценка СКО максимальна для любого номера, т.е. это самая неравномерная последовательность.

Если число частей разбиения равно числу классов n, то при $k > 1$ и $M = n^2 + 1$ оценка СКО нулевая: в каждой части число вхождений выбранного номера одно и то же. Однако если разбивать последовательность на другое число частей, оценки СКО сильно разнятся и не позволяют выбрать наилучшее M. Одновременно можно заметить, что при указанном выборе M результат содержит довольно сильную регулярность. Например, начало последовательности размещений из 5 по 3 после перестановки с $M = 5^2 + 1 = 26$ имеет вид:

{0,0,0),(1,0,1},(2,0,2},{3,0,3},{4,0,4},...

Далее первый и третий номера в размещениях начинают различаться, но закономерность изменения остается очевидной.

При анализе СКО с различными числами от-

резков для $k \le 4$ оказывается, что не худшие результаты для M дают полиномы от n степени k - 1, в которых коэффициенты преимущественно разные, а свободный член равен 1 либо n - 1. Последовательность номеров видимой регулярности не проявляет.

Количество инструкций, получаемое за один перебор, не слишком велико. Например, для размещений из 5 классов по 4 получим последовательность из 625 размещений, что дает 2500 инструкций. В случайном тесте перебор можно повторить несколько раз, при этом число вхождений каждой подпоследовательности длины 4 будет расти равномерно.

3. Классы инструкций MIPS32

Для инструкций MIPS32, исключая сопроцессор плавающей точки (FPU), можно рассмотреть следующие классы:

- Загрузка-сохранение.
- Вычисления: арифметика, сдвиги, логика, битовые перестановки.
- Целочисленное умножение-деление (длительные операции).
- Переходы: условные и безусловные.
- Пересылки.
- Некоторые специальные инструкции.

Здесь можно заметить, что не все сочетания допустимы. Два перехода подряд запрещены, а два умножения-деления приведут к большой задержке, не говоря уже о возможной потере результатов. При задании классификации можно пометить отдельные классы как не допускающие повторов. При переборе из двух вхождений такого класса останется только первое.

Для задания классов в программе шаблона используются переменные типа tree. Такие же переменные используются и для случайного выбора инструкций согласно заданным весам, но здесь каждое поддерево верхнего уровня задает отдельный класс. По умолчанию веса поддеревьев нулевые. Ненулевой вес поддерева верхнего уровня означает, что данный класс не допускает повторов при переборе. Более того, если задать один и тот же ненулевой вес для двух классов, то при удалении повторов эти два класса будут считаться одним. Ниже показан пример задания классов инструкций в шаблоне (для экономии места часть инструкций не включена).

```
tree tt_alu = {
 Lost {
  Align=20 {lb=5, lw=5, sb=5, sw=5},
  NAlig=20 {lwl=5, lwr=5, swl=5, swr=5}},
 Calcul {
  AImm=20 {addi=5, andi=5, ori=5},
  ABin=20 {add=5, sub=5, and=5, or=5},
  AExt=20 { slt=5, sltu=5, clo=5, clz=5 } },
 MulDiv=1  {
  MD1=20 {div=5, mult=5, mul=5 },
  MAS=20 {madd=5, msub=5 } },
 BrJmp=2  {
  Jumps=20 { jr=5, jalr=5, j=5, jal=5},
  Bran=20 { beq=5, bne=5, bgez=5, bltz=5},
  Likely=20 {beql=5, bgezl=5, bltzl=5 } },
 Move=3 {movf=5, movt=5, movn=5, movz=5 }
};
```

Это дерево имеет 5 поддеревьев верхнего уровня. Два из них имеют веса по умолчанию (0), т.е. при переборе нет ограничений на соседство инструкций, выбранных из этих поддеревьев. Три поддерева имеют ненулевые веса. Это значит, что соседство инструкций из одного и того же поддерева не допускается. Для поддеревьев BrJmp и MulDiv смысл запрета обсуждался выше, а для поддерева Move это просто исключение повторений несложных инструкций.

Последовательность инструкций (классов) может также нарушаться из-за следующих действий и приемов, используемых при генерации случайных тестов.

- Предзагрузка. Это прием, который отделяет инструкции загрузки адреса в регистр от использования этого регистра. Предзагрузка позволяет исключить предопределенные комбинации с участием инструкций, использующих загружаемые регистры. Место для инструкций предзагрузки выбирается случайно либо задается, а сами инструкции (T32) или данные для них (T64) формируются позже, после вычисления содержимого регистра.
- Перезагрузка. Если регистровые переходы происходят слишком часто, предзагрузка не успевает их подготовить и регистр перезагружается непосредственно перед переходом. То же делается при наработке регистров баз адресов данных, поэтому стоит выполнить ее заранее, до перебора.
- Исключения ALU и/или FPU. Инструкции, их вызывающие, могут добавляться случайно для усложнения условий выполнения. Вместе с тем они искажают порядок комбинаций классов.
- Разделители. Между некоторыми инструкциями требуется вставка 1-2 разделяющих инструкций в силу особенностей реализации процессора. Генератор вставляет разделители автоматически. Так, чтение регистра hi или lo и запись в них разделяются двумя инструкциями (T32).

- Загрузка условий перехода. При генерации условия перехода должны быть известны. Это может приводить к случайному выбору условия и перезагрузке регистра непосредственно перед переходом.
- Самопроверка. При массовой выдаче инструкций вызов процедуры самопроверки вставляется в код, как только наберется достаточное число измененных регистров.

С учетом нарушений последовательности понятно, что перебор классов инструкций в случайном тесте нужно повторить несколько раз, чтобы дать больше возможностей для реализации всех комбинаций классов.

Инструкции FPU при классификации можно разделить на инструкции одинарной и двойной точности. Это вызвано требованием архитектуры к совместимости по формату выходных и входных регистров. Данное требование является естественным при программировании, однако в случайных тестах его требуется обеспечивать отдельно. В действительности, когда содержимое регистра записано в регистровый файл, оно может интерпретироваться в любом формате. То же относится к результату пересылки или загрузки из памяти, когда формат результата не определен. В остальных случаях, таких как передача данных через bypass, лучше избегать смешивания форматов FPU. Классы инструкций одинарной либо двойной точности можно использовать совместно с классами ALU (см. выше).

4. Настройки и программа

Ниже даны некоторые рекомендации по выбору параметров настройки и пример программы шаблона. В составных именах параметров настройки имена подразделов разделены двоеточием.

4.1. Регистровые аргументы

Общий выбор регистров используется там, где не нужен (или не дал результата) учет специфики инструкций, таких как обращения к памяти и переходы. Традиционным приемом для создания тестовых ситуаций здесь является выбор недавно использованных регистров. Это может создавать зависимости по данным в конвейере. Вероятности (веса) заданы в подразделе Register раздела :Settings.

При случайном выборе регистра может последовательно применяться несколько попыток. Сначала случайно выбирается вариант возможного использования регистра, записанного предыдущей инструкцией: на чтение, на запись либо никак. Если условия позволяют, выбор принимается. Если выбор принят, перед текущей инструкцией может быть вставлена еще одна, заведомо вызывающая исключительную ситуацию ALU или FPU по заданным вероятностям LastRegExcALU или LastRegExcFPU, соответственно.

Вторая попытка основана на т.н. возрасте использования регистра. Это число тактов, проходящих от предыдущего использования на чтение или на запись до возможного использования в текущей инструкции. Если возраст использования попадает в заданный интервал, регистр считается использованным, иначе - новым. Все регистры, допустимые для данного аргумента, делятся на новые или использованные на чтение и/или запись - всего 4 подмножества, одно из которых выбирается случайно, а в нем выбирается произвольный регистр. Если выбранное подмножество пусто, выбирается произвольный регистр из всех допустимых. Такие случаи нередки, так как инструкции имеют разное время выполнения, и настроить границы возраста использования на все сразу нельзя.

4.2. Инструкции перехода

Для условных переходов генератор позволяет случайно выбирать по весам условие перехода "истина" или "ложь". Однако если содержимое регистра условия не соответствует выбору, генератор добавляет в код инструкции перезагрузки регистра, лишние с точки зрения задачи тестирования. Случайное содержимое регистров условия примерно отвечает вероятностям выбора 50:50. Если это допустимо, то можно отменить обязательную перезагрузку регистров. Она останется только для случая, когда содержимое регистра условия неизвестно.

Для выбора целевого адреса обычно достаточно задать выравнивание адреса от 8 до 64 байт. Если в шаблоне описана одна область кода, больше ничего не нужно. Иначе предусмотрен случайный выбор перехода либо в ту же самую область (близкий переход), либо в любую другую. Для близких переходов задаются также допустимые границы смещения адреса.

Как уже отмечалось, для регистровых переходов предзагрузка регистра позволяет высвободить место перед переходом для любых допустимых инструкций. Для больших тестов удобно использовать случайную предзагрузку, которая вставляется автоматически. Случайная предзагрузка настраивается в подразделе Preload_Insertion. Очередная предзагрузка вставляется через N инструкций после места использования предыдущей, где N выбирается каждый раз случайно. Чем выше частота выдачи регистровых переходов, тем меньше должно быть N. Если предзагрузка не успевает готовить реги-

стры, нужно снизить частоту выдачи (вес) регистровых переходов по сравнению с переходами с другой адресацией.

4.3. Загрузка и сохранение данных

Эти инструкции используют адресацию база+смещение (T32), а также база+индекс (T64). Повторное использование регистров адреса снижает число дополнительных инструкций перезагрузки. Чтобы содержимое этих регистров не затиралось случайно, используется режим сохранения баз (установка Tergen:KeepBaseRegs ≥ 1). В этом режиме для операции загрузки-сохранения сначала случайно выбирается адрес данных, а затем подбираются регистры, содержимое которых позволяет получить этот адрес. Если таких регистров не меньше KeepBaseRegs, регистр адреса выбирается из них случайно. Иначе выбирается еще один регистр, в который загружается подходящая база. Этот регистр становится защищенным от случайной записи. Такая техника выбора регистров часто позволяет обойтись вообще без загрузки баз, за исключением начального участка теста. Условием для этого служит подходящее распределение памяти: если областей памяти много, и для каждой требуется отдельная база, регистров может не хватить даже при минимальных требованиях (KeepBaseRegs=1).

4.4. Самопроверка

При самопроверке вызов процедуры сравнения в шаблоне возможен как явно, при помощи функции check, так и по счетчику готовых регистров. Когда один оператор генерирует много инструкций, функция check не подходит, остается только вариант со счетчиком. Установка Tergen:AutoTest задает количество измененных регистров с известным содержимым, при котором добавляется вызов процедуры сравнения. Обычно задается значение от 8 до 10: при меньшем значении растет число вызовов и накладные расходы памяти, при большем растет риск пропустить ошибку из-за повторной записи в тот же регистр.

Самопроверка связана также с двумя областями памяти. В одну (область кадров) записываются данные о содержимом регистров на стадии генерации кода для последующего сравнения, в другую (системная область) - информация о найденной ошибке. Размер области кадров нужно задавать с запасом, но обычно она занимает примерно столько же, сколько код.

4.5. Программа

Для перебора n классов инструкций служит оператор xgroup. Его параметры - дерево инструкций, число k и количество инструкций, которые нужно выработать. Число k - это число элементов в размещениях из n классов по k, $\{i_0, i_1, \ldots i_{k-1}\}$. По умолчанию при переборе используется основная последовательность размещений (M = 1). Для перехода к оптимизированной последовательности нужно увеличить значение k на 20. Здесь значение M задается полиномом $M = 2\ n^2 + n - 1$. Пример оператора с перебором троек классов:

xgroup (tt_alu, 23, 4000)

Пример дерева инструкций (переменная tt_alu типа tree) приведен выше.

Цикл времени выполнения в тестовом коде позволяет выполнить одни и те же инструкции в разных условиях: на первой итерации цикла инструкции загружаются в кэш, на следующих читаются из кэша. Для организации цикла нужно использовать оператор loop, иначе генератор может не обеспечить корректность выполнения второй и следующих итераций цикла. В частности, в цикле loop учитывается возможность изменения условий переходов. Нужно заметить, что в случае выдачи очень большого числа инструкций цикл может оказаться длиннее кэша, и тогда его использование мало что даст.

Оператор loop также обеспечивает корректность предзагрузки в цикле. Обычно после использования предзагруженного регистра запись в него разрешена. В цикле это не так: если загрузка константы в регистр происходит вне цикла или на большем уровне вложенности, то есть опасность порчи содержимого перед началом следующей итерации. В операторе loop уровни вложенности для предзагрузки контролируются.

5. Заключение

Использование перебора классов инструкций при генерации случайных тестов позволяет оценивать покрытие по комбинациям сочетаниям классов инструкций. Объем тестового кода получается гораздо меньшим, чем при переборе комбинаций самих инструкций. Это дает возможность компенсировать нарушения последовательности классов за счет многократного повторения перебора. Генератор тестов позволяет избежать предопределенных комбинаций, таких как загрузка регистра непосредственно перед переходом по этому регистру.

Публикация выполнена в рамках государственного задания ФГУ ФНЦ НИИСИ РАН «Проведение фундаментальных научных исследований (47 ГП)» по теме № FNEF-2022-0004 «Разработка архитектуры, системных решений и методов для создания микропроцессорных ядер и коммуникационных средств семейства систем на кристалле двойного назначения. 0580-2022-0004», Рег № 122041100063-6.

Random Tests with Enumeration of Instruction Classes

A.S. Koutsaev

Abstract. The advantage of random test generators is ease of use and the ability to issue large amounts of code. Stochastic testing reveals both ordinary and specific errors (rare combinations and specification flaws). At the same time, the increase in test coverage does not directly depend on the generator control parameters. Enumeration of placements of n instruction classes by k allows to obtain coverage for the corresponding combinations of instructions with acceptable code size.

Keywords: random tests, coverage, verification, enumeration of instruction classes

Литература

1. Bergeron J. Writing Testbenches using SystemVerilog. Springer, 2006.

2. Хисамбеев И.Ш. Роль стохастического тестирования в функциональной верификации микропроцессоров. Программные продукты и системы 2012, №3, 107-112.

3. И.В. Грибков, А.В. Захаров и др. Стохастическое тестирование в системе INTEG. Программные продукты и системы. 2007, № 2, 22–26.

4. И.В. Грибков, А.В. Захаров и др. Развитие системы стохастического тестирования INTEG. Программные продукты и системы 2010, №2, 14-22.

5. А.С. Куцаев. Развитие генератора случайных тестов tergen. Труды НИИСИ РАН 2018, Том 8, № 1, 19-26.