# Программная реализация

**Аннотация**

Небольшое отступление о множествах, функциях полезности и прагматике. Уравнения с множествами (и вообще использование множеств) взялись из попытки компактного описания последовательностей кэш-промахов и кэш-попаданий (интересно, как с этой проблемой борется Genesys-Pro). Компактность удалось достичь. Была надежда, что эта компактная форма записи даст и неплохой алгоритм разрешения (с глубоким использованием особенностей операций над множествами). Однако на реальных данных формула, выписанная на множествах, получалась очень большой длины (пропорционально размеру самой кэш-памяти). К счастью, появилась иная идея описания кэш-попаданий и кэш-промахов в ограничениях – использование функций полезности (т.е. ограничения вида «тегсет еще не вытеснен/тегсет уже вытеснен»). В общем случае использование функций полезности не дает выигрыша, но при совместном рассмотрении кэш-памяти и TLB, т.е. при возможности использовать пересечение хранящихся в них данных, удается выписать ограничения компактного размера с использованием функций полезности.

Таким образом, в работе представлены два новых способа записи тестовых ситуаций в кэш-памяти с использованием ограничений – уравнения на множества тегов и уравнения с функциями полезности.

## Содержание

# 1 Структура генератора тестовых программ

В главе описывается система генерации тестовых программ на основе тестовых шаблонов. Входными данными системы являются:

- тестовый шаблон;
- описания тестовых ситуаций;
- начальное состояние микропроцессора;
- дополнительные параметры конфигурации.

Выходом системы является тестовая программа, удовлетворяющая тестовому шаблону с учетом данных описаний тестовых ситуаций и начального значения микропроцессора. Ядром системы является *генератор ограничений* (см. рис. 1). Ограничения разрешаются другим компонентом системы – *решателем ограничений*. Модель, построенную решателем ограничений, анализирует *генератор инструкций тестовой программы*, с целью построить готовую тестовую программу.

На рис. 2 показано сравнение предлагаемого генератора тестовых программ с известным генератором Genesys-Pro [1]. Оба генератора получают на вход тестовый шаблон, а на выходе у них тестовые программы. Однако Genesys-Pro на вход требует *architectural model* и *testing knowledge* – первая дает по сути эталонный симулятор микропроцессора, а второй описывает эвристики выбора аргументов для инструкции. Genesys-Pro не предполагает систематического описания семантики инструкций, выделения ветвей их функциональности, описания программных контрактов инструкций. Симулятор нужен для построения модельного состояния микропроцессора после исполнения очередной сгенерированной инструкции, а эвристики выбора аргументов составляют основу тех ограничений, которые описывают значения аргументов очередной инструкции. Идея заключается в разделении описания функции, которую реализует инструкция, и ограничения на аргументы инструкции. Другой особенностью Genesys-Pro является то, что поддерживаемые им тестовые шаблоны зачастую не фиксируют последовательность инструкций (это позволяет строить более простые ограничения, потому как генерируемая последовательность инструкций может по ходу генерации подстраиваться под уже сгенерированные инструкции со сгенерированными значениями аргументов, под состояние микропроцессора, в которое привели сгенерированные инструкции).

В отличие от Genesys-Pro в предлагаемом инструменте описание семантики инструкций задается в едином виде – в виде описаний тестовых ситуаций. Каждая

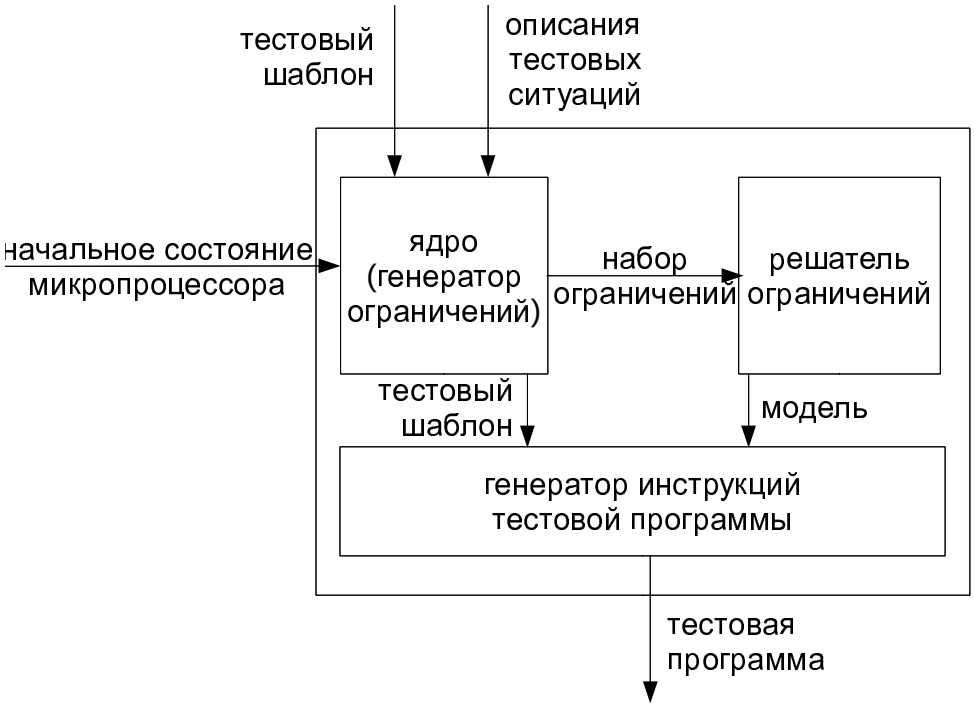


Рис. 1: Структура системы генерации тестовых программ

тестовая ситуация описывает не только ограничение на свои аргументы, но и результат исполнения инструкции *при данном ограничении на аргументы* инструкции. В функции, которую реализует инструкция, выделяются отдельные *ветви функциональности*, ситуации различного поведения инструкций, каждая ветвь функциональности становится отдельной тестовой ситуацией. Например, инструкция целочисленного сложения ADD может быть исполнена либо точно, либо с возникновением переполнения. Поэтому у этой инструкции можно выделить 2 ветви функциональности (точное исполнение и исполнение с переполнением), каждая ветвь дает свою тестовую ситуацию. Кроме того, предлагаемый генератор дает возможность указать начальное состояние микропроцессора, которое будет эффективно использовано при построении тестовой программы. Последовательность инструкций фиксирована и задается в тестовом шаблоне.

Описания тестовых ситуаций можно составлять по следующей схеме:

1. выделить тестируемые инструкции;

2. найти описание семантики выбранных инструкций (обычно оно входит в доку-

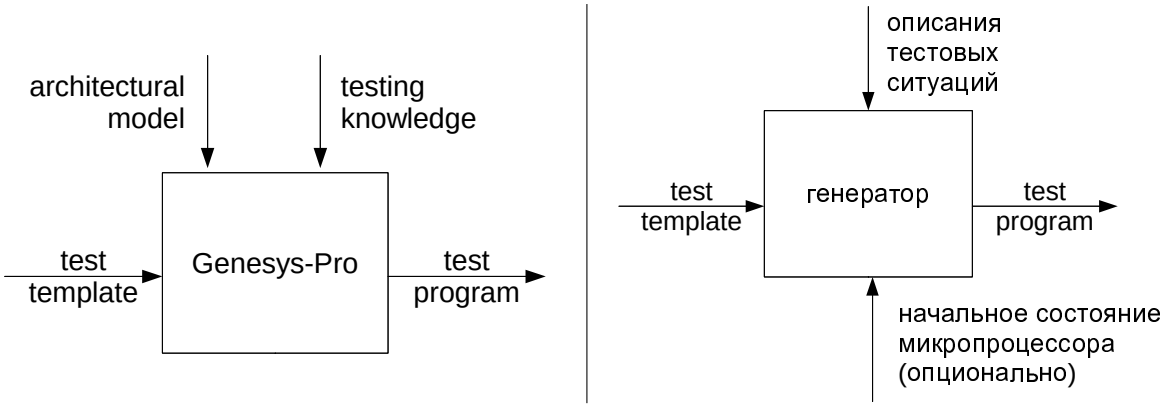


Рис. 2: Сравнение с Genesys-Pro

ментацию по тестируемой архитектуре);

3. для каждой инструкции выделить:
   - аргументы: имена и битовые длины;
   - предусловие (ограничение на значения аргументов инструкции, при которых она может быть результативно исполнена);
   - ветви функциональности инструкции (ситуации различного поведения инструкции);
4. для каждой ветви функциональности составить описание тестовой ситуации, поместив туда объявления аргументов, предусловие и операторы, описывающие поведение инструкции на данной ветви функциональности, т.е. вычисление выходного значения инструкции или создание условий возникновения исключительной ситуации.

Раздел 2 содержит описание предлагаемого языка описания тестовых шаблонов и тестовых ситуаций, пригодный для описания инструкций арифметической, логической подсистем, подсистемы обращения к памяти, инструкции переходов.

# 2 Описание тестовых шаблонов

Описание тестового шаблона состоит из следующих секций:

1. *заголовок шаблона*: объявление регистров и констант;

2. *тело шаблона*: инструкции и ограничения тестового шаблона.

Заголовок шаблона должен содержать объявления всех задействованных в тестовом шаблоне регистров (для каждого регистра указывается его имя и битовая длина) и констант (или по-другому, «непосредственных значений» – для каждой константы так же указывается ее имя и битовая длина). Регистр может использоваться в качестве аргумента-результата инструкции, константа не может использоваться в качестве аргумента-результата инструкции. Регистры и константы сохраняют свою битовую длину на протяжении всего тестового шаблона. Генератор ограничений трактует регистр как переменную со значением и генерирует начальное значение такой переменной, при которой выполнены все заявленные в тестовом шаблоне тестовые ситуации. Обычно для инициализации регистра достаточно одной-двух инструкций (это зависит от количества бит, которое может изменить одна инструкция). Константа трактуется как значение, которое не меняется. Для нее тоже генерируется значение и при составлении тестовой программы оно вставляется непосредственно на место аргумента инструкций.

Пример объявления регистра и константы:

```
<register id="z" length="64" />
<constant id="c" length="16" />
```

Тело тестового шаблона состоит из описаний инструкций и ограничений. Для инструкции необходимо указать:

- имя инструкции;
- аргументы инструкции (объявленные ранее регистры или константы);
- внешние переменные инструкции;
- тестовую ситуацию.

Тестовые ситуации на косвенные обращения не рассматриваются, поэтому в описания тестовых шаблонов не включены механизмы описания косвенных обращений.

*Механизм внешних переменных* инструкции позволяет формулировать ограничения на локальные переменные, определенные в разных тестовых ситуациях. При объявлении новой локальной переменной кроме имени можно указать идентификатор (имя надо указывать обязательно, а идентификатор – необязательно). Все идентификаторы внутри тестовой ситуации должны быть уникальными. Это может быть виртуальный адрес, физический адрес, некое внутреннее выражение. Если тестовая

ситуация является составной, то все тестовые ситуации должны определять выносимые на уровень тестового шаблона идентификаторы. На уровне тестового шаблона идентификатору ставится в соответствие некоторое новое уникальное внутри шаблона имя, которое можно использовать наравне с другими переменными (регистрами или константами).

Тестовая ситуация описывает некоторое поведение инструкции. Обычно можно выделить два типа поведений инструкции: существенное исполнение (вычисление значения, осуществление взаимодействия) и генерация исключения. При существенном исполнении тестовая ситуация описывает предусловие инструкции и набор условий и вычислений, определяющих данное поведение инструкции. При исполнении с генерацией исключения описывается предусловие инструкции и набор условий и вычислений, приводящих к возникновению исключения. Само возникновение исключения, как оператор, не описывается.

Тестовая ситуация состоит из набора *ветвей* – простейших поведений инструкции. Ветви могут *комбинироваться* в дизъюнкции и конъюнкции. Дизъюнкция ветвей (или их комбинаций) означает, что в данный момент инструкция может себя вести согласно хотя бы одной из ветвей. Конъюнкция ветвей (или их комбинаций) означает, что в данный момент инструкция может себя вести согласно всем ветвям одновременно. Объединенные в конъюнкцию ветви не могут иметь существенное исполнение.

Пример тестовой ситуации ветви:

```
<situation>
    <branch name="overflow" />
</situation>
```

Для ветви указывается имя. Оно используется при поиске соответствующего файла с описанием этой тестовой ситуации. Поиск производится на основе имени тестовой ситуации и имени инструкции, для которой она указана.

Пример тестовой ситуации, включающей комбинацию ветвей:

```
<situation>
    <or>
        <and>
            <branch name="overflow" />
            <branch name="normal" />
        </and>
        <branch name="zero" />
    </or>
```

```
    </situation>
```

Эту комбинацию ветвей можно прочесть следующим образом: *данная инструкция должна себя вести как overflow с normal или как zero*.

В описаниях тестовых ситуациях могут быть фиксированы обращения к различным подсистемам микропроцессора. Указание тестовой ситуации в шаблоне может фиксировать и тестовую ситуацию на эти обращения (а для полных тестовых шаблонов оно *должно* фиксировать тестовую ситуацию на эти обращения). Среди подсистем можно выделить различные уровни кэш-памяти и TLB. Содержимое секции тестовых ситуаций обращений к подсистемам определяется архитектурой тестируемого микропроцессора. Например, оно может быть следующим:

```
<situation>
        ...
        <access>
            <cache level="1" type="DATA" id="miss" />
            <cache level="2" type="DATA" id="miss" />
            <cache level="3" type="DATA" id="hit" />
            <tlb  id="invalid">
                <microtlb type="DATA" id="miss" />
            </tlb>
        </access>
</situation>
```

Это описание говорит о том, что при исполнении данной инструкции в кэш-памяти данных первого и второго уровней должен быть кэш-промах, а в кэш-памяти данных третьего уровня – кэш-попадание; в TLB должна произойти тестовая ситуация invalid, а в MicroTLB данных – промах. Интерпретация терминов `cache, tlb, microtlb, miss, hit, invalid` заложена в части генератора ограничений, ответственного за тестируемую архитектуру.

Кроме инструкций тестовый шаблон может содержать ограничения (assert) на текущее состояние микропроцессора и введенные внешние переменные. Можно выделить следующие основные применения этих ограничений:

1. задание зависимостей на адреса разных инструкций (например, у двух инструкций одинаковые физические адреса при разных виртуальных адресах – одна инструкция определяет свои виртуальный и физический адрес, другая инструкция делает то же, а ограничение фиксирует связь значений этих четырех переменных);

2. задание ветви в графе потока управления: при тестировании инструкций перехода с некоторым сравнением вместо их непосредственного описания предлагается описывать конкретные результаты сравнений (истинно это сравнение в данный момент или ложно); тестовый шаблон не позволяет описывать разветвленные потоки управления, разрешается описывать лишь последовательности инструкций, поэтому такие дополнительные ограничения позволяют описать в тестовом шаблоне один из путей в графе потока управления и сгенерировать для этого пути свою тестовую программу.

Описание ограничения, как и описание тестовой ситуации, может состоять из указания ветви или их комбинаций. Пример:

```
<assert>
    <or>
        <and>
            <branch name="ff1" />
            <branch name="ff2" />
            <branch name="ff3" />
        </and>
        <branch name="ff4" />
    </or>
</assert>
```

Пример описания тестового шаблона целиком:

```
<template>
  <register id="x" length="64" />
  <register id="y" length="64" />
  <register id="z" length="64" />
  <constant id="c" length="16" />

  <instruction name="ADD">
      <argument name="x" />
      <argument name="y" />
      <argument name="z" />
      <external name="v1" id="virtual" />
      <external name="p1" id="phys" />
      <situation>
          <or>
              <and>
```

```
                <branch name="overflow" />
                <branch name="normal" />
            </and>
            <branch name="zero" />
        </or>

        <access>
            <cache level="1" type="DATA" id="miss" />
            <cache level="2" type="DATA" id="miss" />
            <cache level="3" type="DATA" id="hit" />
            <tlb  id="invalid">
                <microtlb type="DATA" id="miss"></microtlb>
            </tlb>
        </access>

    </situation>
</instruction>

<assert>
    <or>
        <and>
            <branch name="ff1" />
            <branch name="ff2" />
            <branch name="ff3" />
        </and>
        <branch name="ff4" />
    </or>
</assert>
</template>
```

# 3 Описание тестовых ситуаций

Описание тестовой ситуации состоит из следующих секций:

1. *заголовок тестовой ситуации*: объявление аргументов инструкции;
2. *тело тестовой ситуации*: последовательность операторов.

Заголовок тестовой ситуации должен содержать объявления всех аргументов инструкции. Для каждого аргумента указывается его имя (локальное внутри данной

тестовой ситуации), статус «только для чтения/результат» и битовая длина. Последовательность аргументов тестовой ситуации должна совпадать с последовательностью аргументов инструкции с точностью до переименования. Иными словами, первый аргумент тестовой ситуации должен обозначать первый аргумент инструкции (их битовые длины должны совпадать), второй аргумент тестовой ситуации – второй аргумент инструкции и так далее. Аргументы, помеченные статусом «только для чтения» не могут менять свое значение во время всей тестовой ситуации. Аргументы, помеченные статусом «результат» обязаны получить значение в данной инструкции. Тестовая ситуация может иметь произвольное количество аргументов обоих статусов в произвольном порядке. Статус «только для чтения» позволяет передать в качестве аргументов инструкции одинаковые переменные (одинаковые регистры или константы). Однако все аргументы инструкции, соответствующие аргументам тестовой ситуации со статусом «результат», должны иметь разные имена (они могут быть среди аргументов со статусом «только для чтения»).

Пример:

```
<argument name="rt" state="result" length="64" />
<argument name="base" state="readonly" length="64" />
<argument name="offset" state="readonly" length="16" />
```

Тело тестовой ситуации состоит из последовательности операторов трех видов:

- оператор `let` – объявление новой локальной переменной вместе с ее инициализацией;
- оператор `assert` – фиксация некоторого ограничения на значения переменных;
- оператор `procedure` – вызов процедуры (ее семантика не задается в описании тестовой ситуации).

Тестовая ситуация по своей сути является ветвью функциональности инструкции. Поэтому ее описание содержит лишь последовательность операторов, условные операторы и операторы цикла отсутствуют.

Оператор `let` объявляет новую переменную и инициализирует ее результатом вычисления некоторого выражения. Оператор может содержать указание имени переменной (оно должно быть новым) *или* указание идентификатора новой переменной (все идентификаторы внутри тестовой ситуации должны быть разными; идентификатор может совпадать с именем этой или любой другой переменной). Безымянные операторы `let` позволяют задать идентификатор существующей переменной.

Тело оператора содержит выражение, результат вычисления которого станет значением объявляемой переменной. Выражение может содержать следующие операции:

- переменные (`var`) и константы (`constant`); допустимы только неотрицательные константы, у каждой константы должна быть указана битовая длина;
- битовые операции: выделение бита с заданными номером (`bit`), выделение непрерывной последовательности бит с заданными границами (`bits`), битовая конкатенация (`concat`), битовая степень (`power`);
- арифметические операции: сложение (`sum`), вычитание (`sub`); операции проводятся по модулю экспоненты битовой длины аргументов.

Производится строгая «проверка типов», т.е. битовых длин аргументов операций. Например, сложению подвергаются только выражения с одинаковыми битовыми длинами. В частности это позволяет автоматически вычислить битовую длину объявляемой переменной.

Пример:

```
<let name="vAddr" id="virtual">
    <sum>
        <sign_extend size="64"><var>offset</var></sign_extend>
        <var>base</var>
    </sum>
</let>
```

Оператор `assert` позволяет указать ограничение, справедливое в некоторый момент на значениях аргументов тестовой ситуации и локальных переменных. Выражения объединяются с помощью отношений сравнения. Пример:

```
    <assert>
        <eq>
            <bits end="1" start="0"><var>vAddr</var></bits>
            <constant length="2">0</constant>
        </eq>
    </assert>
```

Оператор `procedure` позволяет указать более сложное действие, в котором могут участвовать различные подсистемы микропроцессора. Семантика процедур не фиксируется при описании тестовой ситуации. Аргументы, наоборот, фиксируются.

Каждый аргумент может иметь идентификатор (это позволяет располагать аргументы в произвольном порядке, указывая семантику каждого аргумента с помощью идентификатора). Тело аргумента может быть следующих типов:

- выражение на определенные к моменту вызова процедуры переменные; выражение задается с использованием того же синтаксиса, что и в операторе `let`;
- новая переменная (`new`);
- символьная константа (`symbol`).

Набор допустимых процедур и идентификаторов их аргументов задается генератором ограничений.

Пример:

```
<procedure name="AddressTranslation">
        <argument id="physical"><new name="pAddr"/></argument>
        <argument id="virtual"><var>vAddr</var></argument>
        <argument id="points_to"><symbol name="DATA"/></argument>
        <argument id="points_for"><symbol name="LOAD"/></argument>
</procedure>
```

Пример описания тестовой ситуации целиком:

```
<situation>
    <argument name="rt" state="result" length="64" />
    <argument name="base" state="readonly" length="64" />
    <argument name="offset" state="readonly" length="16" />

    <let name="vAddr" id="virtual">
        <sum>
            <sign_extend size="64"><var>offset</var></sign_extend>
            <var>base</var>
        </sum>
    </let>

    <assert>
        <eq>
            <bits end="1" start="0"><var>vAddr</var></bits>
            <constant length="2">0</constant>
        </eq>
```

```
    </assert>

    <procedure name="AddressTranslation">
            <argument id="physical"><new name="pAddr"/></argument>
            <argument id="virtual"><var>vAddr</var></argument>
            <argument id="points_to"><symbol name="DATA"/></argument>
            <argument id="points_for"><symbol name="LOAD"/></argument>
    </procedure>

    <let id="physical"><var>pAddr</var></let>

    <let name="dwByteOffset">
        <bits end="2" start="0"><var>vAddr</var></bits>
    </let>

    <!-- dwByteOffset can be changed according to BigEndian/LittleEndian -->

    <procedure name="LoadMemory">
        <argument id="data"><new name="memdoubleword"/></argument>
        <argument id="size"><symbol name="WORD"/></argument>
        <argument id="physical"><var>pAddr</var></argument>
        <argument id="virtual"><var>vAddr</var></argument>
        <argument id="points_to"><symbol name="DATA"/></argument>
    </procedure>

    <procedure name="BytesSelect">
        <argument id="type"><symbol name="WORD"/></argument>
        <argument id="from"><new name="data"/></argument>
        <argument id="content"><var>memdoubleword</var></arument>
        <argument id="index"><var>dwByteOffset</var></argument>
    </procedure>

    <assert>
        <eq>
            <var>rt</var>
            <sign_extend size="64"><var>data</var></sign_extend>
        </eq>
    </assert>
</situation>
```

# 4 Генератор ограничений (ядро)

Генератор ограничений имеет модульную структуру (см. рис. 3). В его составе есть модуль, ответственный за генерацию ограничений для общих механизмов описания архитектур микропроцессоров, такие как работа с регистрами, работа с последовательностью инструкций, работа с локальными переменными в тестовых ситуаций, и модуль, специфичный для тестируемой архитектуры (он содержит алгоритмы генерации ограничений для таких механизмов, как трансляция адресов, как кэш-память).

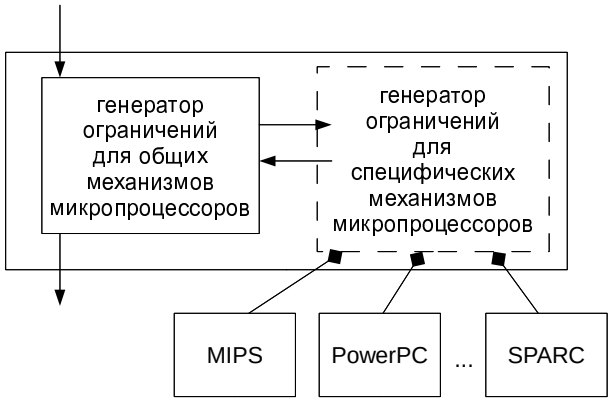


Рис. 3: Структура генератора ограничений

Генерация ограничений производится последовательно для каждой инструкции тестового шаблона. Однако разрешение ограничений может происходить быстрее при некотором порядке ограничений, это можно учесть при генерации ограничений.

# Список литературы

[1] Allon Adir, Eli Almog, Laurent Fournier, Eitan Marcus, Michal Rimon, Michael Vinov, and Avi Ziv. Industrial experience with test generation languages for processor verification. *IEEE Design and Test of Computers*, 21(2):84–93, Mar./Apr. 2004.