Автоматическая генерация тестовых данных для оптимизаторов графических моделей

С.В. Зеленов, Д.В. Силаков

1. Введение

В настоящее время графическое моделирование активно применяется в различных отраслях промышленности: автомобилестроении (Ford, General Motors [6], Daimler Chrysler [7]), авиастроении (Boeing [8]), аэрокосмической промышленности (American Institute of Aeronautics and Astronautics [5]) и других.

Графическое моделирование позволяет создавать модели разрабатываемых систем из различных элементарных деталей на экране монитора, как в конструкторе. При этом от разработчиков не требуется изучения каких-либо формальных языков описания моделей.

Одной из наиболее распространенных сфер применения графического моделирования в настоящее время является создание исполняемого кода для микропроцессоров встроенных систем. Вместо написания кода вручную инженеры создают модель, описывающую работу устройства, а на основе этой модели специальная программа – генератор кода – создает код на языке программирования.

Среди существующих инструментов для создания моделей можно выделить инструменты Simulink и Stateflow, входящие в пакет Matlab от MathWorks [1], ASCET от ETAS [2], который может быть интегрирован в Matlab, а также Statemate от I-Logix [3], позволяющий создавать модели с помощью UML-диаграмм.

Для генерации кода на основе моделей Matlab используются Real-Time Workshop, входящий в Matlab, а также инструмент TargetLink от компании dSpace [4]. Существует инструмент для генерации кода на основе моделей ASCET.

В автомобилестроении в настоящее время стандартом де-факто являются пакеты Matlab и TargetLink, используемые компаниями Daimler Chrysler, General Motors, Ford, Toyota [6] и др.

Можно выделить следующие основные достоинства графического моделирования:

- простота моделирования для разработчиков (не требуется изучения формальных языков для описания моделей);
- автоматическая генерация спецификации и документации (фактически, сама модель является наглядной и в то же время формальной спецификацией системы);
- автоматическая генерация исполняемого кода, моделирующего работу системы, а также кода для исполнения в микропроцессорах встроенных систем;
- кроме того, современные средства моделирования обеспечивают простоту модульного проектирования.

Генераторы кода широко используются в таких областях, где к используемым программам предъявляются очень высокие требования (встроенные системы автомобилей, самолетов и т.п.). Соответственно, высокие требования предъявляются и к генераторам кода.

Можно выделить два основных требования:

- получаемый код должен быть сравним с кодом, написанным вручную, как по требованиям к ресурсам, так и по времени выполнения;
- структура кода должна соответствовать модели.

Инструменты, удовлетворяющие первому требованию, появились на рынке относительно недавно, в конце девяностых годов. Для достижения приемлемого качества кода выполняются различные оптимизирующие преобразования моделей. При этом остро встает вопрос о соответствии кода, полученного на основе оптимизированной модели, исходной конструкции. Проверка выполнения второго требования после проведения оптимизирующих преобразований является одной из основных задач тестирования генераторов кода.

Число всевозможных преобразований, как правило, велико, и для проверки корректности генератора требуется большое количество тестов, поэтому встает вопрос об автоматизации разработки тестовых данных.

Для формального описания преобразований моделей, осуществляемых трансляторами, используются так называемые правила преобразования графов [7]. Каждое правило состоит из двух частей – левой и правой. Левая часть описывает шаблонную конструкцию, которая будет подвергнута трансформации, и ограничения на параметры этой конструкции, при выполнении которых трансформация будет произведена. Правая часть описывает результирующую конструкцию, которая будет получена после преобразования.

Генераторы кода начали применяться в промышленности недавно, и работ, посвященных их тестированию, немного. Из существующих подходов можно выделить формальное доказательство корректности работы генератора

(например, [9]) и тестирование на основе правил преобразований графов – Classification Tree Method (CTM) [10, 11].

При использовании формального доказательства (например, [9]) возникают традиционные для этого метода проблемы – длительные сроки и сложность проверки, приводящие к высокой стоимости тестирования. Кроме того, в настоящее время для основных существующих генераторов кода отсутствуют общедоступные формальные описания. В промышленных масштабах формальные доказательства на данный момент не используется.

Метод CTM [10, 11] состоит из двух этапов:

– разбиение шаблона для тестирования на независимые области;
– разбиение полученных областей на классы эквивалентности.

На рис. 1 проиллюстрирован пример применения метода к оптимизации, выполняющей вычисление константных аргументов блока Sum. В качестве областей здесь выделены блоки Sum и Const, классы эквивалентности строятся согласно количеству блоков Const в модели.

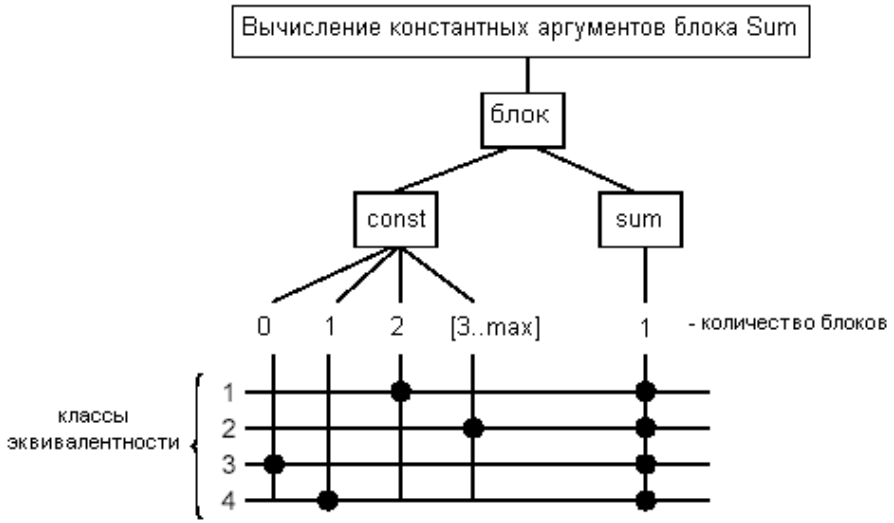

Рис. 1. Пример применения метода CTM

Преимуществом метода CTM является простота определения покрытия. Также имеется большой опыт использования этого метода в смежной области – для полуавтоматического создания тестов для готовых систем на основе их моделей; последним фактом обусловлен большой интерес промышленности к методу CTM [11].

Основным недостатком использования CTM для тестирования генераторов кода является отсутствие автоматизации. Разбиение шаблона на области и

далее на классы эквивалентности осуществляется вручную; при этом для реальных систем число получаемых классов эквивалентности, как правило, очень велико. Для полноты тестирования для каждого из этих классов необходимо получить, по крайней мере, один тест, а инструментов для автоматической генерации тестов в настоящее время не существует.

В настоящей статье предложен метод GraphOTK автоматической генерации тестовых данных для тестирования оптимизирующих трансляторов графических моделей. Предложенный метод позволяет решить проблему автоматической генерации тестовых данных, а также за счет параметризации генератора позволяет варьировать количественные и качественные характеристики получаемых тестовых данных.

2. Метод автоматизации тестирования оптимизаторов графических моделей

Метод GraphOTK автоматической генерации тестовых данных для тестирования оптимизирующих трансляторов графических моделей является развитием метода генерации тестовых данных для оптимизаторов в компиляторах языков программирования [12].

Метод заключается в том, чтобы построить для тестируемого оптимизатора представительное множество входных графических моделей следующим образом:

- построить абстрактную тестовую модель входных данных оптимизатора;
- в терминах абстрактной тестовой модели сформулировать критерий покрытия этих входных данных;
- перебрать соответствующие тестовые данные.

2.1. Построение абстрактной тестовой модели

Тестовая модель строится на основе абстрактного описания правил оптимизирующих трансформаций.

Алгоритм оптимизации формулируется с использованием формализма правил преобразования графов. Оптимизатор для осуществления своих трансформаций ищет шаблонную конструкцию – такое сочетание элементов графической модели, которое присутствует в левой части какого-нибудь правила преобразования графов. Поэтому для построения тестовой модели будем рассматривать только те элементы графических моделей, которые задействованы хотя бы в одном шаблоне.

Итак, на основании информации о шаблонных конструкциях из всех правил преобразования графов для данной оптимизации составляется список элементов графической модели, задействованных в этих шаблонах. После этого описывается множество тестовых модельных строительных блоков со следующими свойствами:

- каждому элементу графической модели из полученного списка соответствует свой вид тестового модельного строительного блока;
- строительные блоки могут связываться между собой, чтобы иметь возможность образовывать структуры, соответствующие шаблонам.

Будем называть тестовой модельной структурой граф, вершины которого – строительные блоки, а ребра – связи между строительными блоками.

2.2. Формулировка критерия покрытия

Проекция графических моделей в тестовые модельные структуры для данной абстрактной тестовой модели индуцирует разбиение множества графических моделей на классы эквивалентности. Один класс эквивалентности состоит из графических моделей, которые имеют одинаковое тестовое модельное представление, т.е. которые неразличимы для алгоритма оптимизации. Это свойство позволяет нам выдвинуть гипотезу, согласно которой на эквивалентных графических моделях оптимизатор работает одинаково. Следовательно, в желаемом тестовом наборе достаточно иметь не более одного представителя из каждого класса эквивалентности.

Поскольку множество тестовых модельных структур, т.е. множество классов эквивалентности, в общем случае бесконечно, то для создания тестового набора мы должны выбрать некоторое его конечное подмножество. Основанием для этого выбора должны служить те шаблоны, которые были выделены при анализе алгоритма оптимизации. Таким образом, критерий тестового покрытия формулируется в терминах абстрактной тестовой модели.

2.3. Создание генератора тестов

Для получения множества тестовых данных для целевого оптимизатора графических моделей в соответствии с методом GraphOTK необходимо разработать соответствующий генератор. Этот генератор должен создавать представительное (т.е. удовлетворяющее сформулированному критерию покрытия) множество тестовых данных (т.е. графических моделей).

Создание генератора представительного множества тестовых воздействий начинается с анализа алгоритма тестируемого оптимизатора, построения абстрактной модели и формулировки критерия тестового покрытия, как это было описано выше. После этого происходит разработка собственно генератора.

Генератор тестов состоит из двух компонентов. Первый компонент, называемый итератором, отвечает за последовательную генерацию тестовых модельных структур. Второй компонент, называемый меппером, отвечает за отображение каждой тестовой модельной структуры в графическую модель.

Итератор должен создавать множество модельных структур в соответствии с выбранным критерием тестового покрытия.

Для данной модельной структуры S меппер должен строить соответствующие тестовые данные, обладающие следующим свойством: графическая модель, построенная по тестовой модельной структуре S, имеет модельное представление, совпадающее с S.

По окончании разработки итератора и меппера они собираются в генератор. После этого можно проводить генерацию множества тестовых данных.

3. Генераторы тестовых данных для оптимизаторов графических моделей

3.1. Генератор тестовых данных для оптимизатора Switch-блока

Набор тестов предназначен для генератора кода, осуществляющего трансляцию и оптимизацию блока Switch.

У блока Switch (Рис. 2) имеются три входных сигнала (In1, control и In2). Каждый из входов может быть константой (блоком, подающим на выход всегда одно и то же число), внешним источником сигнала (блок InPort) или шаблонной контрукцией (например, арифметическим выражением). Switch-блок описывает структуру if-then-else – в зависимости от величины сигнала на входе control он передает на выход либо сигнал со входа In1, либо сигнал со входа In2. Если сигнал на входе control больше либо равен пороговому значению threshold, являющемуся параметром блока, на выход подается сигнал со входа In1, в противном случае на выход подается сигнал со входа In2.

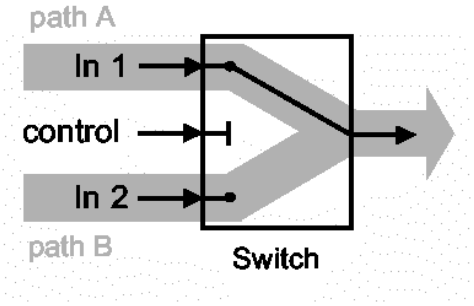

Рис. 2. Блок Switch

При генерации кода оптимизатор анализирует возможные значения величины сигнала на входе control и определяет, будет ли значение булевского выражения «control >= threshold» постоянным в процессе выполнения. Если это так, то одна из веток конструкции if-then-else, описываемой Switch-блоком, никогда не будет выполняться (т.е. на выход блока всегда будет подаваться сигнал с одного и того же входа) и может быть удалена из модели.

В соответствии с методом GraphOTK, на основе описания алгоритма оптимизации строится абстрактная модель тестов.

В случае оптимизации Switch-блока в описании алгоритма используется следующие термины: блок Switch и различные блоки библиотеки Simulink, из которых могут быть составлены модельные конструкции для входов блока Switch. Для данного алгоритма оптимизации являются важными значения параметров блоков модели, влияющие на величину выходного сигнала блока (например, амплитуда сигнала у блока Sine Wave). Таким параметрам в абстрактной модели соответствуют свойства терминов.

Шаблоном для оптимизатора является блок Switch с различными комбинациями значений порогового параметра и области возможных значений входа control.

В процессе генерации строятся различные модели Simulink, содержащие блок Switch. В качестве входов блока Switch строятся различные модельные конструкции с различными параметрами.

Целью генерации является получение набора тестов, удовлетворяющего следующим условиям:

- набор должен содержать модели с блоком Switch; в качестве каждого входа блока Switch должны быть перебраны все наследники узла InputSystemBlock абстрактной модели тестов;
- для блоков, имеющих более одного входа, в качестве каждого входа должны быть перебраны все наследники узла InputSystemBlock; если на вход должен подаваться непрерывный сигнал, то в качестве такого входа должны быть перебраны все наследники блока Signal;
- для блоков с переменным количеством входов с минимально допустимым числом входов N тесты должны содержать модели, где каждый такой блок будет иметь N входов, и модели, где он будет иметь N+1 вход;
- тесты должны содержать модели, где величина сигнала на входе control у блока Switch всегда не меньше порогового значения, тесты, где эта величина всегда меньше порогового значения, а также тесты, где значение булевского выражения «control >= threshold» изменяется в процессе выполнения модели;
- для блоков с несколькими входами, над каждым из которых может быть произведена одна из операций некоторого фиксированного набора (например, у блока Sum для каждого входа можно указать знак – ‘+’ или ‘–‘), набор тестов должен содержать блоки со всеми возможными операциями (например, для упомянутого выше блока Sum тесты должны содержать блоки, где по крайне мере один аргумент имеет знак ‘+’, и блоки, где по крайне мере один аргумент имеет знак ‘–’).

Путем настройки параметров генератора можно получать тесты, не содержащие блоков-генераторов сигналов и блоков-преобразователей сигналов.

По умолчанию в генерируемых тестах присутствуют как корректные, так и некорректные модели, выполнение которых в среде Matlab приводит к возникновению исключительных ситуаций. Возможна настройка генератора для генерации только корректных моделей. В этом случае, в частности, гарантируется соблюдение динамической семантики при выполнении математических функций, для чего в процессе генерации осуществляются следующие действия:

- для математических функций, область определения которых является отрезком, интервалом или полуинтервалом (asin, acos, acosh, atanh, log, log10, sqrt) на пути входного сигнала помещается блок Saturation, ограничивающий величину сигнала;
- для фунции reciprocal (1/x) и для блока Product в случае деления входной сигнал сравнивается с нулем (с помощью блока Relational Operator) и результат сравнения (1 – если сигнал равен нулю, 0 в противном случае) прибавляется к величине сигнала;
- для блоков, осуществляющих побитовые операции (BitwiseLogicalOperator) берется абсолютное значение входного сигнала (с помощью блока Abs) и входной сигнал приводится к типу uint32 (с помощью блока DataTypeConversion);
- для функции возведения в степень (блок Math Function, функция pow) в случае, если первый аргумент (основание степени) равен нулю, а второй (показатель степени) отрицателен, вместо нуля блоку передается значение ‘1’. (Сравнение значений осуществляется при помощи блоков Relational и Logic, результат сравнения прибавляется к первому аргументу блока, осуществляющего возведение в степень).

В случае, когда требуется получать как корректные, так и некорректные модели, перечисленные выше действия не осуществляются.

Таблица 1 содержит объем и время генерации сгенерированного множества тестовых данных тестов.

В приложении А приведен аннотированный пример сгенерированных тестовых данных для оптимизации SwitchBlock.

Количество тестов	**Размер тестов, MB**	**Время генерации**
3 112	64	5 m. 07 s.

Таблица 1. Характеристики сгенерированных тестов для оптимизации SwitchBlock.

3.2. Генератор тестовых данных для оптимизатора Flowchart

Набор тестов предназначен для генератора кода, осуществляющего трансляцию и оптимизацию графа Flowchart блока Stateflow Chart, описывающего конструкцию if-then-else.

Оптимизатор ищет в графе вершины, соединенные двумя и более дугами. Каждой дуге ставится в соответствие условие, в случае выполнения которого осуществляется переход по этой дуге, а также может быть определено действие, осуществляемое при переходе. Для каждой вершины графа одна из исходящих дуг не должна иметь условия – она соответствует ветви “else” конструкции “if-then-else”.

Все дуги, соединяющие одни и те же вершины графа, заменяются оптимизатором на одну дугу. Все проверки условий и выполнение соответствующих действий описываются как действие новй дуги. Пример преобразования графа, осуществляемого оптимизатором, показан на Рис.3.

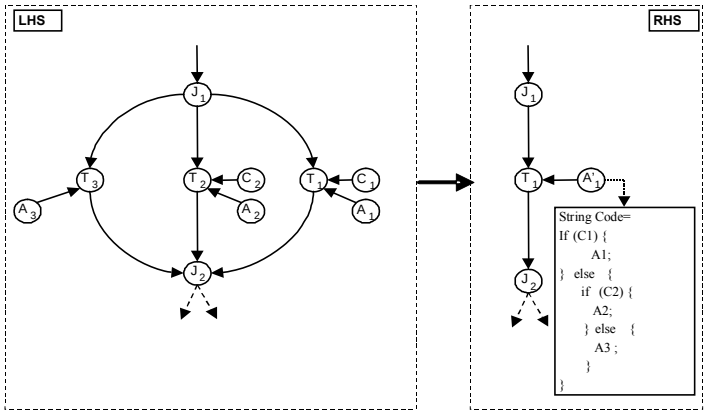

Рис.3. Пример преобразования, осуществляемого оптимизатором Flowchart

В соответствии с методом GraphOTK, на основе описания алгоритма оптимизации строится абстрактная модель тестов.

В случае оптимизации Flowchart в описании алгоритма используются следующие термины: StateflowMachine (часть блока Simulink Stateflow Chart, которая содержит описание flowchart-графа) и вершины графа.

Шаблоном для оптимизатора является блок Stateflow Chart, который содержит не менее двух вершин, соединенных дугами.

В процессе генерации для блока Stateflow Chart строятся различные циклические и ациклические графы, описывающие структуры if-then-else различной разветвленности и глубины вложенности. Гарантируется, что из

начальной вершины графа можно достичь любой другой его вершины. Число вершин в графах изменяется от 3 до 10. Число дуг, соединяющих произвольные две вершины, изменяется от 0 до 3.

Каждой (кроме одной) дуге, исходящей из каждой вершины, поставлено в соответствие условие, заключающееся в проверке принадлежности входного сигнала блока заданному интервалу. Гарантируется, что условия condition-узлов дуг, исходящих из одной вершины, являются взаимоисключающими.

Каждой дуге поставлено в соответствие действие; в результате выполнения действий формируется величина выходного сигнала блока в зависимости от величины входного сигнала. Для дуг, исходящих из начальной вершины графа, действие заключается в присваивании выходному сигналу величины входного сигнала, умноженной на некоторый коэффициент (различный для различных дуг). Для остальных дуг действие заключается в прибавлении к величине выходного сигнала величины входного сигнала, умноженной на некоторый коэффициент (различный для различных дуг). Каждый тест представляет собой модель, содержащую блок Statflow Chart.

Входной сигнал подается извне (с помощью блока InPort), выходной сигнал подается на выход модели (блок OutPort).

Целью генерации является получение набора тестов, содержащих блок Stateflow Chart и удовлетворяющих следующим требованиям:

- тесты должны содержать графы с количеством конечных вершин (т.е. вершин, у которых нет исходящих дуг) от 1 до 5 (значение 5 выбрано в целях получения приемлемого количества тестов);
- между начальной вершиной графа и каждой из конечных вершин должны быть пути (по дугам через другие вершины графа, без учета циклов) длины от 1 до 5 (длина равна числу дуг на пути); должны быть перебраны все возможные сочетания длин пути (т.е. должны быть графы, где все пути имеют длину 1, графы с путями длины 1 и 2 и т.д.);
- для каждой упомянутой выше комбинации длин пути набор тестов должен содержать как ациклические графы, так и графы с циклами;
- число дуг между вершинами графа на каждом из путей должно варьироваться от 1 до 3 (что соответствует безусловному переходу, конструкции “if-else” и конструкции “if-elseif-else”).

Объем и время генерации сгенерированного множества тестовых данных тестов приведены в Таблица 2.

В приложении А приведен аннотированный пример сгенерированных тестовых данных для оптимизации Flowchart.

Количество тестов	Размер тестов, MB	Время генерации
1 335	60	3 m. 45 s.

Таблица 2. Характеристики сгенерированных тестов для оптимизации Flowchart.

4. Заключение

В статье предложен метод автоматической генерации тестовых данных для тестирования оптимизирующих трансляторов графических моделей. Предложенный метод позволяет решить проблему автоматической генерации тестовых данных, а также за счет параметризации генератора позволяет варьировать количественные и качественные характеристики получаемых тестовых данных.

В соответствии с предложенным методом были разработаны генераторы тестовых данных для нескольких оптимизаторов графических моделей, которые используются в коммерческих проектах в автомобильной промышленности.

Литература

[1]. The MathWorks, www.mathworks.com
[2]. ETAS ASCET. http://en.etasgroup.com/products/ascet/
[3]. I-Logix. http://www.ilogix.com
[4]. dSPACE, www.dspace.com
[5]. Paul A. Barnard. Software Development Principles Applied to Graphical Model Development. // AIAA Modeling and Simulation Technologies Conference and Exhibit, San Francisco, California, Aug. 15-18, 2005. (https://tagteamdbserver.mathworks.com/ttserverroot/Download/28446_Barnard%20AIAA-2005-5888.pdf)
[6]. Ranville S., Black P. Automated Testing Requirements – Automotive Perspective. // The Second International Workshop on Automated Program Analysis, Testing and Verification. 2001. (http://hissa.nist.gov/~black/Papers/autoTestReqsWAPATV.rtf)
[7]. Conrad, M., Dörr, H., Schürr, A., Stürmer, I. Graph-Transformations for Model-based Testing. // GI-Lecture Notes in Informatics. 2002. N 12. P. 39-50.
[8]. MathWorks Tools Help Land Unpiloted Boeing Spacecraft. MathWorks User Stories. (https://tagteamdbserver.mathworks.com/ttserverroot/Download/452_9797v00_Boeing_SMV_ROI.pdf)
[9]. Glesner S., Geiss R., Boesler B. Verified Code Generation for Embedded Systems. // Electronic Notes in Theoretical Computer Science. 2002. 65. N 2.
[10]. Grochtmann M., Grimm K. Classification-Trees For Partition Testing. // Software Testing, Verification and Reliability. 1993. N 3 (2). P. 63-82.
[11]. Sturmer I. Integration of the Code Generation Approach in the Model-Based Development Process By Means Of Tool Certification. // Journal of Integrated Design and Process Science. 2004. Vol. 8 (2). P.1-11
[12]. С.В. Зеленов, С.А. Зеленова, А.С. Косачев, А.К. Петренко. Применение модельного подхода для автоматического тестирования оптимизирующих компиляторов // CIT Forum, 2003. http://www.citforum.ru/SE/testing/compilers/

Приложение А.
Примеры сгенерированных тестовых данных

Пример 1. Тестовые данные для оптимизации SwitchBlock.

Графическая модель тестовых данных для примера 1 изображена на Рис. 4.

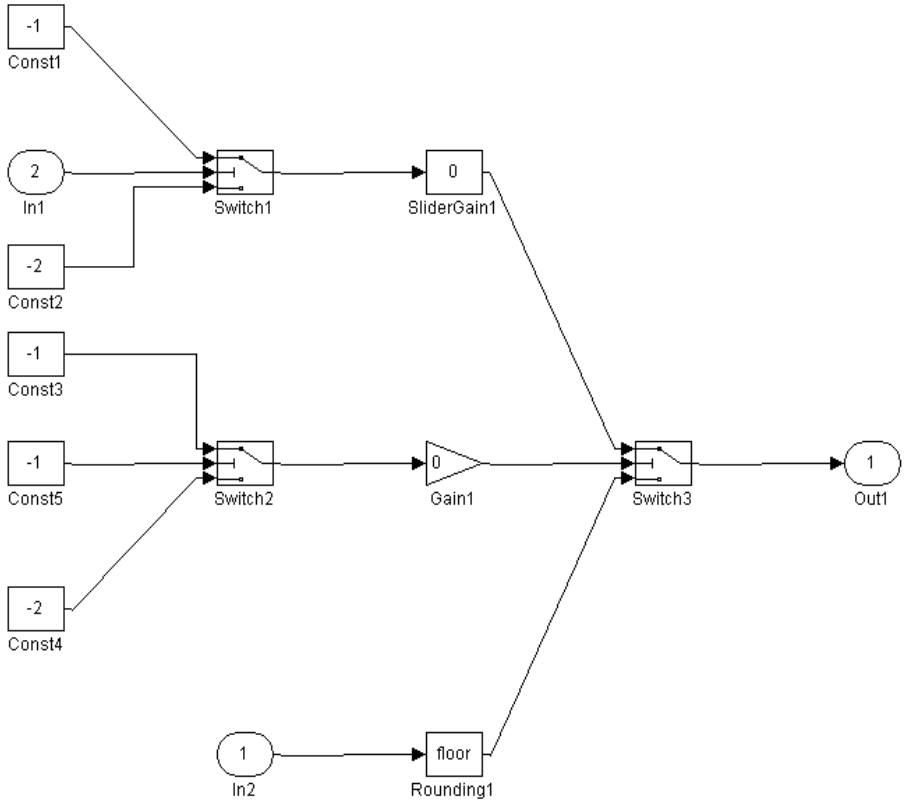

Рис. 4. Графическая модель сгенерированных тестовых данных для оптимизации SwitchBlock (пример 1).

В этой модели `Swith2.threshold = Switch3.threshold = -1`. Оба блока всегда будут передавать на выход сигнал с верхнего входа; в результате оптимизации блок Switch2 может быть удален из модели. Величина сигнала на входе *control* блока Switch1 не может быть оценена на основе анализа данной системы, поскольку этот сигнал подается извне.

Пример 2. Тестовые данные для оптимизации Flowchart.

Графическая модель тестовых данных для примера 2 изображена на Рис. 5.

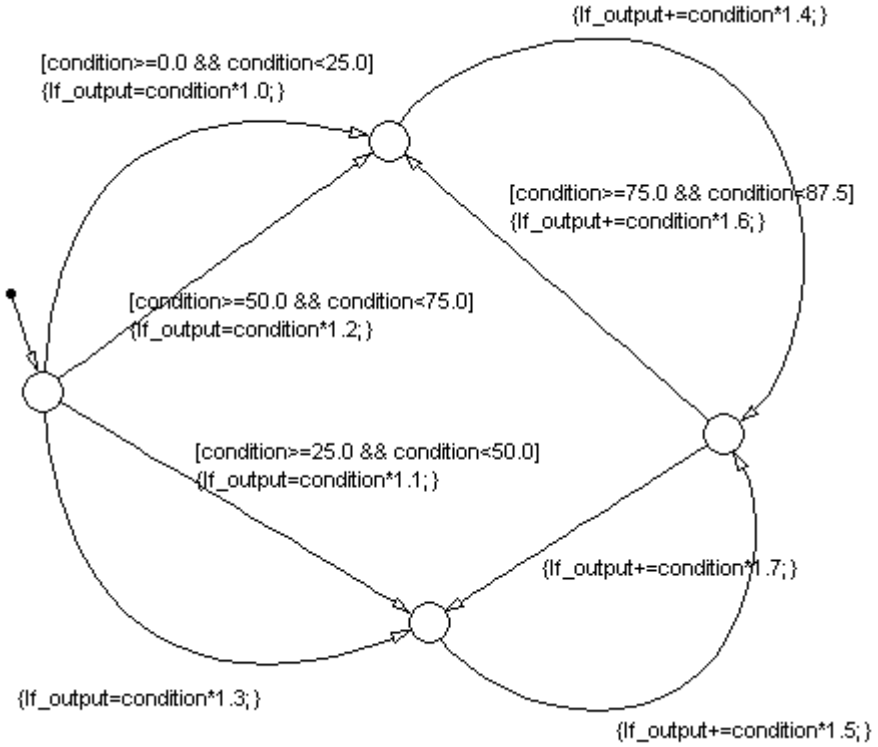

Рис. 5. Графическая модель сгенерированных тестовых данных для оптимизации Flowchart (пример 2).

Этот граф содержит два простых цикла. Условия переходов таковы, что в процессе исполнения ни при каких значениях входного сигнала на верхнем цикле зацикливания не произойдет, а на нижнем зацикливание возникнет при любых входных сигналах.