Международная научно-техническая мультиконференция “Проблемы информационно-компьютерных технологий и мехатроники”. Материалы международной научно-технической конференции “Многопроцессорные вычислительные и управляющие системы ” (МВУС-2007). Т. 1, с. 198–203.

В.С. Гуров, А.А. Шалыто, Б.Р. Яминов

Технология верификации автоматных моделей программ без их трансляции во входной язык верификатора

Санкт-Петербургский государственный университет информационных технологий, механики и оптики, г. Санкт-Петербург,

jaminov@rain.ifmo.ru, shalyto@mail.ifmo.ru, vadim.gurov@jetbrains.com

Автоматные модели программ [1] широко используются для создания реактивных управляющих систем [2] поскольку автоматы позволяют наглядно и надежно представлять поведение системы. Реактивные системы часто используются в областях, в которых требуется безотказная работа и цена ошибки чрезвычайно велика. По этой причине возникла необходимость в автоматической проверке соблюдения заданных свойств автоматной программой.

Одним из методов для автоматической проверки программ является *Model checking* [3, 4]. Это автоматизированный подход, позволяющий для заданной модели поведения системы с конечным (возможно, очень большим) числом состояний и логического свойства (требования) проверить, выполняется ли это свойство в рассматриваемых состояниях модели. Проверяемые свойства формулируются в *темпоральной* логике, позволяющей задавать не только мгновенное состояние системы, но и *историю* развития системы во времени.

Идею метода *Model checking* реализуют специальные программы – *верификаторы*. Такая программа получает на вход модель, описанную на входном языке верификатора, и свойство, которое проверяется (возможно, на другом языке). На выходе программа формирует либо сообщение о том, что указанное свойство на заданной модели выполняется, либо набор действий (сценарий), приводящий к его нарушению.

Существует большое количество верификаторов, в том числе, и с открытыми кодами. Поэтому имеет смысл воспользоваться одним из уже существующих. Для того чтобы проверять свойства автоматных моделей с помощью уже существующего верификатора требуется осуществлять две операции:

- транслировать автоматную модель во входной язык верификатора;
- при получении сценария ошибки, транслировать его обратно в автоматную модель.

Существует несколько работ [4, 5], в которых упомянутые операции выполнялись вручную. Однако до сих пор неизвестны автоматические верификаторы автоматных моделей программ.

Цель настоящей работы – создать автоматический верификатор программ рассматриваемого класса.

Трансляция автоматной модели во входной язык верификатора и обратно порождает ряд проблем. Во-первых, возможно возникновение наведенных ошибок – ошибок, которых не было в исходной модели, но они появились в верифицируемой модели из-за недостаточно корректной трансляции. Во-вторых, порождаются «лишние» состояния, которые не влияют на результат верификации, но увеличивают объем работы верификатора.

В данной работе предлагается новый подход к решению задачи верификации автоматных моделей программ. Его идея заключается в том, чтобы миновать этап трансляции и позволить верификатору осуществлять переходы прямо в автоматной модели. Такой подход позволяет решить перечисленные выше проблемы.

Для реализации идеи требуется верификатор с *расширяемым* входным языком, для которого кроме стандартных типов объектов (булев тип, целочисленный, строковый и т.д.) можно создавать более сложные и абстрагированные структуры и специфицировать их свойства. Это позволит ограничить дробление модели на мелкие элементарные состояния.

Кроме того, требуется интерпретатор автоматной модели – инструментальное средство, которое по событиям осуществляет переходы между состояниями в автомате, вычисляет значения условий на переходах и запускает вложенные автоматы. Верификатор будет использовать это средство для перемещения между состояниями в автоматной модели.

В качестве интерпретатора было выбрано инструментальное средство *Unimod* [6, 7], которое предназначено для создания и интерпретации программ, основанных на автоматном подходе. Это средство позволяет создавать реактивные системы, в которых выделяются:

1. Источники событий – сущности, которые поставляют автомату события для обработки. Это «сенсоры» реактивной системы.
2. Система иерархически связанных автоматов, описывающая поведение реактивной системы. Переход в автомате в общем случае

помечается событием и условием, при выполнении которого переход активируется. Автоматы могут иметь гиперсостояния и быть вложены в состояния других автоматов. Действия могут помечать состояния и/или переходы.

3. Объекты управления – сущности, у которых можно вызывать действия и запрашивать значения предикатов.

В качестве верификатора с расширяемым входным языком выбрано средство *Bogor* [8, 9]. Инструментальные средства *Bogor* и *Unimod* реализованы на одном языке программирования (*Java*). Это позволяет их весьма легко интегрировать.

В данной работе было создано расширение входного языка *Bogor* в виде нового класса – `AutomataModel`. Объект этого класса представляет собой автоматную модель. Для него задано лишь одно действие: `step` (шаг), которое совершает обработку очередного события в модели. Также у объекта этого класса можно получать различную информацию о состояниях автоматов, вызванных в ходе шага выходных воздействиях и других свойствах, которые могут понадобиться для формулировки верифицируемого требования к модели.

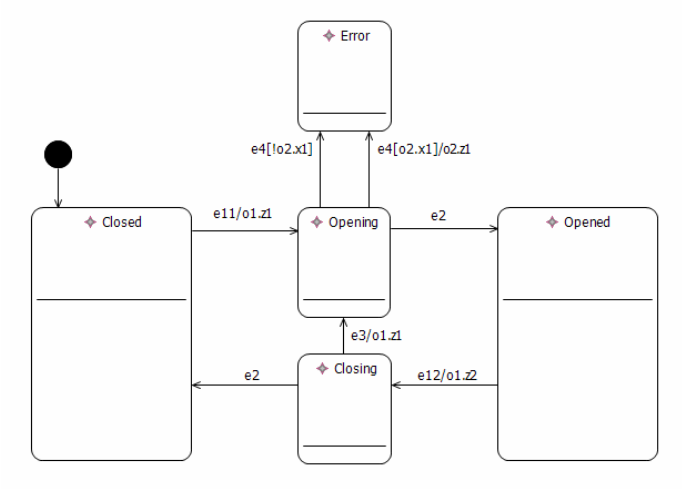
Глобальное состояние объекта нового класса кодируется как набор состояний каждого автомата. Это означает, что из одинакового набора состояний автоматов в модели должно следовать одинаковое ее поведение на любые воздействия. Данное утверждение может нарушаться, если часть логики реактивной системы хранится в объектах управления. К примеру, в одном и том же наборе состояний автоматов некоторый флаг внутри объекта управления может быть разным, что приведет к разному поведению. Методы решения задачи для такого случая будут описаны в конце настоящей работы.

За счет создания класса `AutomataModel` спецификация автоматной модели во входном файле верификатора *Bogor* сводится всего лишь к одному бесконечному циклу, в котором совершается шаг автоматной модели. Сама модель инициализируется из *Unimod*-файла. С точки зрения верификатора, обработка одного события в автоматной модели происходит атомарно.

Для решения задачи верификации автоматных моделей остается записать проверяемые требования. Это можно сделать во входном файле Bogor в виде *автомата Бюхи* [3], либо в виде формулы темпоральной логики *LTL*, используя в утверждениях методы класса `AutomataModel`.

Созданный верификатор был апробирован на нескольких примерах и успешно доказывал верные утверждения о моделях и

выдавал сценарии нарушений для неверных утверждений. Приведем один из таких примеров – модель дверей лифта (см. рисунок).



Автомат дверей лифта

При открытии дверей может произойти ошибка (e4). При закрытии двери могут встретить препятствие (e3), в случае чего они снова открываются. Верификатору подавалась на вход модель дверей лифта и перечисленные ниже утверждения.

- При отсутствии ошибки (e4) для любой «справедливой» истории [3] состояние *Opened* (*Открыта*) посещается бесконечное количество раз. Справедливая история означает, что любое ожидаемое событие когда-нибудь произойдет, и модель не остановится навсегда в состоянии, из которого есть выход по некоторому событию. Результат верификации данного утверждения – «Выполняется».
- При отсутствии ошибки (e4) для любой «справедливой» истории состояние *Closed* (*Закрыта*) посещается бесконечное количество раз. Результат верификации – «Нарушается». Верификатор выводит контрпример в виде пути, когда при каждой попытке закрыться дверь встречает препятствие (e3).

В данной работе при верификации реактивных систем предполагается, что в объектах управления нет логики. Источники событий и объекты управления отключаются от системы, и работа проводится с «изолированной» автоматной моделью. Однако в ряде случаев логика может быть также и в объектах управления. К примеру, может существовать зависимость между входными и выходными

воздействиями объекта, которая не будет учтена при верификации. Это приведет к несуществующим историям развития, а значит, возможно, к наведенным ошибкам.

Тем не менее, эта проблема разрешима.

1. Можно вручную формулировать такие зависимости (ограничения) в темпоральной логике и добавлять их в верифицируемую формулу для исключения невозможных сценариев ошибок.
2. В будущем предполагается создание формальной системы задания пред- и постусловий для действий объектов управления, для того чтобы автоматически учитывать указанные зависимости при верификации.

Данная работа показывает, как можно весьма просто и эффективно верифицировать автоматные модели. Работа выполнена как дополнение к инструментальному средству *Unimod*. Это позволяет получить единое средство для создания, интерпретации и верификации реактивных систем.

1. Шалыто А. А. Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/l/
2. Шалыто А. А., Туккель Н. И. SWITCH-технология – автоматный подход к созданию программного обеспечения «реактивных» систем //Программирование. 2001. № 5. http://is.ifmo.ru/works/switch/l/
3. Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. М.: МЦНМО. 2002. http://is.ifmo.ru/verification/_klark_gamberg_pered_verification.djvu
4. Вельдер С. Э., Шалыто А. А. Введение в верификацию автоматных программ на основе метода *Model checking*. http://is.ifmo.ru/download/modelchecking.pdf
5. Кузьмин Е. В., Соколов В. А. Моделирование, спецификация и верификация «автоматных» программ. Ярославский государственный университет им. П.Г. Демидова. http://is.ifmo.ru/verification/
6. UniMod project. http://unimod.sourceforge.net/
7. Гуров В., Мазин М., Нарвский А., Шалыто А. UML. SWITCH-технология. Eclipse //Информационно-управляющие системы, 2004, № 6. http://is.ifmo.ru/works/uml-switch-eclipse/
8. Bogor website. http://bogor.projects.cis.ksu.edu/
9. Robby A., Dwyer M., Hatcliff J. Bogor: A Flexible Framework for Creating Software Model Checkers. TAIC PART 2006. http://projects.cis.ksu.edu/docman/view.php/8/125/SAnToS-TR2006-2.pdf