Автоматическое построение верификационных моделей для приложений на языке C

Ю. В. Юсупов
yury.yusupov@gmail.com

В. П. Котляров
r36959@motorola.com

Санкт-Петербургский государственный политехнический университет

Проверка на моделях (model checking) — это обширный раздел теории верификации программных систем, имеющий множество практических приложений. Данная работа посвящена важной задаче в рамках этого раздела — автоматическому построению моделей из текстов программ. В работе представлены принципы автоматического построения базовых протоколов Летичевского из программ на языке C, а также реализация, основанная на промышленном анализаторе кода KlocWork. Представлены результаты апробации подхода в рамках верификационной технологии VRS, рассмотрены перспективы и дальнейшие пути развития подхода.

Введение

Программное обеспечение (ПО) часто используется в критичных к ошибкам областях человеческой деятельности, поэтому гарантии качества производимого продукта являются неотъемлемой частью современного процесса производства ПО. Для достижения заданного уровня качества наряду с тестированием применяется верификация, которая означает проверку соответствия между требованиями и свойствами реализованной системы [1]. Дополнительной целью верификации являются поиск и обнаружение дефектов

и ошибок, внесённых во время разработки или модификации программы. Также формальная верификация обеспечивает гарантию того, что в системе будут проанализированы все ключевые свойства во всех режимах работы.

Подходы к верификации можно разбить на следующие две группы: дедуктивный, основанный на автоматическом доказательстве теорем, использовании графов и разнообразных специализированных алгебр [2], и проверка на моделях (model-cheking), цель которой заключается в построении модели системы и проверке требований для каждого возможного состояния этой модели [3]. В последнем случае модель программной системы строится на некотором дополнительном формальном языке.

Существует большое количество технологий и методов тестирования и верификации, основанных на модельном подходе, — UniTesk[1], Zing[2], Spec Explorer[3] и др. (обзоры можно найти в [4, 5]).

Однако для этих подходов остро стоит вопрос о трудоёмкости построения формальных спецификаций. Например, при использовании подходов UniTesk и Zing объём создаваемых формальных спецификаций сопоставим с объёмом исходных кодов системы, для которых они составляются. Очевидно, что создавать такие спецификации — очень трудоёмкая задача, и это во многом снижает эффективность практического использования проверки на моделях.

В данной работе описаны принципы автоматического построения верификационных моделей на языке базовых протоколов Летичевского [6] из программ на языке C, а также реализация этого подхода на основе промышленного анализатора кода KlocWork [7]. В работе представлены результаты апробации подхода с использованием верификационной технологии VRS [6], а также обсуждаются перспективы и дальнейшие пути развития подхода.

1. Контекст работы

1.1. Базовые протоколы

Данный формализм предложен в 1996 году известным математиком А. А. Летичевским для создания поведенческих моделей систем с целью верификации [8, 9].

[1] http://www.unitesk.ru.
[2] http://research.microsoft.com/zing.
[3] http://research.microsoft.com/specexplorer.

Базовый протокол является формальной записью утверждения о некоторых событиях, которые должны произойти в программе, алгоритме, протоколе при выполнении определенных условий. Например, пусть у нас есть следующее требование к системе: «Если система находится в состоянии busy, то после возврата трубки на базу она должна перейти в состояние idle». В данном утверждении можно выделить три составляющие: предусловие — «Если система находится в состоянии busy», процессная составляющая — «возврат трубки на базу» и постусловие — «система перейдёт в состояние idle».

В общем случае базовый протокол является тройкой Хоара [10] и имеет следующий вид:

$$\alpha \xrightarrow{\mu} \beta \ ,$$

где α и β — предусловие и постусловие соответственно, а μ — процессная составляющая базового протокола. Условия α и β задаются с помощью логических выражений некоторого базового языка и определяют условия на множестве состояний модели.

Разные базовые протоколы могут склеиваться по пред- и постусловиям, если состояние, описанное в постусловии одного базового протокола, совпадает с состоянием, описанным в предусловии другого. Таким образом, изолированно заданные поведенческие свойства образуют граф поведения модели, который и обходит верификатор.

Для построения базовых протоколов модель системы должна быть представлена в виде взаимодействующих агентов [11]. Агент имеет имя и параметры. Набор параметров агента определяется его типом. При «исполнении» модели верификатором в каждый момент времени агент находится в некотором состоянии, которое характеризуется именем состояния агента, а также значениями его параметров. Для взаимодействия агентов между собой предусмотрен механизм обмена сообщениями.

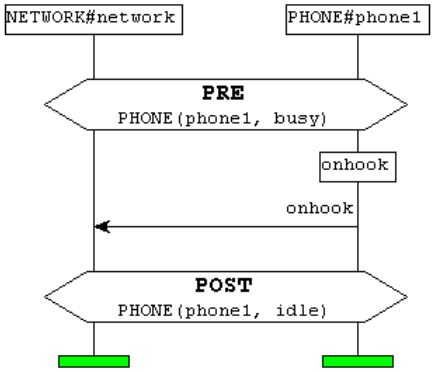
Запись утверждения о том, что агент A типа T находится в состоянии s, записывается в виде *T(A,s)*. А обращение к параметру p агента A типа T имеет следующий вид: $T\ A.p$.

Базовый протокол имеет синтаксис MSC-диаграммы [6], поэтому может представляться в двух формах: текстовой и графической (MSC/PR и MSC/GR) [12]. Для создания базового протокола задействуются следующие элементы MSC-диаграмм:

- instance (сущность) — для обозначения агентов;

- condition (условие) — для записи пред- и постусловий базового протокола;
- action (действие) — для описания локальных действий, происходящих в базовом протоколе;
- message (сигнал) — сигнал взаимодействия между агентами.

На рисунке приведён пример базового протокола в графической нотации, описывающий представленное выше требование.



Базовый протокол в графической нотации.

В верхней части диаграммы находится предусловие (**PRE**) базового протокола, в котором описано состояние модели. В данном примере состояние модели определяется состоянием агента *phone1*, имеющим значение *busy*. Агент *phone1* имеет тип PHONE.

Процессная часть базового протокола описывает набор событий, происходящих в системе после того, как предусловие становится истинным. В данном базовом протоколе присутствует локальное действие *onhook* (возврат трубки на базу), а также происходит отправка сообщения агенту *network* с типом NETWORK, информирующего о произошедшем действии.

В результате событий, описанных в процессной части, система переходит в состояние постусловия (**POST**), которое характеризуется сменой значения состояния агента *phone1* на *idle*.

1.2. VRS-технология

VRS-технология была создана совместно Институтом Кибернетики им. В.М. Глушкова НАН Украины и компанией Motorola для верификации функциональных поведенческих спецификаций, описанных с помощью базовых протоколов [6]. Она проверяет модели, извлекаемые из базовых протоколов, на недетерминированное поведение, достижимость определённых состояний, а также на полноту и непротиворечивость.

Автоматизированная верификация готовых программ с использованием инструментария VRS подразумевает наличие функциональных требований, по которым была реализована система, а также модели системы в виде множества базовых протоколов, построенной на основе программного кода, спецификаций проектирования и проч. В результате проверяется, удовлетворяют ли модели ПО требованиям к программной системе, то есть удовлетворяет ли требованиям само ПО.

Для проведения сеанса верификации по требованиям составляется список событий (например, сигналов, вызовов функций и т. п.) или базовых протоколов, содержащих описание этих событий, а также порядок их следования. Далее VRS проверяет, удовлетворяет ли модель системы этим условиям — содержит ли пути, куда входят события или протоколы в указанном порядке. Нахождение таких путей (трасс) является доказательством корректности поведения модели системы, заложенного в требованиях. Поиск осуществляется по сигналам взаимодействия между агентами или по названиям базовых протоколов в трассах.

Таким образом, трасса — это один из сценариев поведения модели, а поскольку модель строилась по программной системе, то можно сказать, что трасса — это сценарий поведения самой системы. Она получается как результат соединения нескольких базовых протоколов в одну последовательную цепочку. VRS выдает трассы в MSC/PR нотации.

После сеанса верификации составляется верификационный отчёт с указанием ошибок, неточностей и несоответствий, найденных в процессе верификации. К отчёту прилагаются полученные трассы, описывающие некорректные поведения системы.

Совокупность сгенерированных трасс может использоваться для автоматизированного создания тестов к программной системе. В качестве инструмента автоматизации тестирования совместно с

VRS используется инструмент TAT (Test Automation Toolset) [13]. Интегрированный комплекс VRS/TAT обеспечивает генерацию тестов по VRS-трассам, а также их автоматический «прогон» [13]. В работах [14, 15] описано, как комплекс VRS/TAT был адаптирован для тестирования прикладного и системного ПО мобильных Java-телефонов.

1.3. Анализатор кода KlocWork

В данной работе было решено воспользоваться готовым анализатором кода KlocWork, созданным и развиваемым одноимённой компанией[4]. Он является промышленным анализатором C, C++ и Java-приложений, имеет открытый интерфейс к промежуточным представлениям (АСД — абстрактное синтаксическое дерево) разобранного кода и обладает возможностью по расширению базовой функциональности.

Основными особенностями инструмента являются:

- большие объемы обрабатываемого кода систем (4–5 MLOC);
- высокая скорость обработки исходного кода за счёт оптимизированных алгоритмов анализа;
- использование унифицированных интерфейсов, позволяющих получать доступ к промежуточным данным статического и динамического анализа кода систем.

На основе АСД KlocWork проводит синтаксический анализ кода и его проверку на удовлетворение разным свойствам. Это делается с помощью специальных обработчиков (checkers), реализованных как динамически подключаемые библиотеки. Обработчик содержит правила обхода АСД с целью проверки заданных свойств кода, его реализация возможна на языках C/C++. Правила создания собственных обработчиков подробно описаны в документации продукта [16].

[4] http://www.klocwork.com.

2. Принципы отображения C-кода в базовые протоколы

2.1. Основные идеи

C-приложение представляется в виде двух агентов, принадлежащих разным типам, — агент-приложение и агент-окружение. Агент-приложение моделирует все приложение целиком и имеет набор параметров — все переменные исходного приложения. Базовые протоколы строятся для каждого оператора программы — присваивания, условного оператора, цикла и т.д. Для сложных операторов создаётся несколько базовых протоколов — на конструкцию в целом и на все ее подконструкции.

Агент-окружение моделирует внешнее для программы окружение и служит для приёма сигналов от агента-приложения, информирующих о происходящих событиях в модели. Такие сигналы являются удобным средством для ручного анализа трасс (для исследования поведения программной модели), а также используются VRS в процессе верификации для поиска трасс.

Предусловие базового протокола описывает состояние приложения до выполнения текущего оператора, а постусловие — состояние после выполнения. Состояние агента-программы изменяется в зависимости от того, в каком месте модели находится сейчас верификатор, а значения его параметров — в зависимости от выполнения алгебраических операций над ними, соответствующих алгебраическим операциям над переменными формализуемого выражения.

Изменения состояния агента отражаются в постусловии базового протокола, которое описывает состояние приложения после выполнения текущего оператора.

Процессная часть протокола содержит локальное действие, в теле которого содержится информация об обрабатываемом выражении, и сигналы взаимодействия между агентами. Сигналы делятся на сигналы обращения к переменным и функциям.

2.2. Детали

Сохранение потока управления. Для сохранения в модели потока управления программы и с целью упрощения генерации трасс в формулы пред- и постусловий каждого базового протокола, помимо описания состояния агента-приложения, вводится специальный атрибут. Он задаёт порядок «склеивания» базо-

вых протоколов в соответствии с потоком управления исходного С-приложения. Данный атрибут меняет своё значение в постусловии каждого протокола, обеспечивая тем самым уникальность состояния модели.

Сигналы модели. Для индикации вызова функции используются два сигнала с именем функции в качестве параметра (parameter): Function_Call_Start(parameter) и Function_Call_End(parameter). Первый находится в протоколе, передающем управление множеству протоколов, описывающих вызываемую функцию, а второе — в протоколе, возвращающем управление в основной поток из вызываемой функции. Кроме того, в локальном действии первого протокола указывается имя вызываемой функции со всеми параметрами.

Еще одна пара сигналов — Variable_Used(parameter) и Variable_Modified(parameter) — используется для индикации факта обращения к переменным (использование и изменение соответственно). В качестве параметра используется имя переменной.

Локальные действия. Локальное действие базового протокола служит для лучшей «читаемости» трасс, а его тело содержит имя формализуемой конструкции. Например, для выражения с вызовом функции локальное действие будет содержать имя вызываемой функции со всеми параметрами.

3. Реализация

3.1. Основные моменты

Процесс формализации С-кода и получение модели программы в виде базовых протоколов можно разбить на два основных этапа. Вначале за счет инструмента KlocWork происходит анализ исходного кода программы и построение его промежуточного АСД-представления. На втором этапе совершается обход АСД с помощью специально созданного обработчика, а также формирование и генерация базовых протоколов. Данная компонента реализована на языке С. Эффективность работы обработчика достигается за счёт применения специальных методов обхода АСД, описанных в документации к KlocWork [16].

Для построения базовых протоколов по С-программе использовались следующие данные:

- model, env — агенты, характеризующие программную систему и внешнее окружение;
- MODEL, ENV — типы агентов model и env;
- MODEL#model, ENV#env — сущности, отображающие агент-приложение и агент-окружение на MSC диаграмме;
- MODEL(model, FUNC) — отображение состояния агента model. FUNC имя исполняемой в данный момент функции;
- FUNC_VAR — отображение переменной VAR, принадлежащей функции FUNC;
- control_flow — атрибут, отвечающий за поток управления;
- function_call_point — атрибут для сохранения точки вызова функции.

Для получения текстового представления всей структуры базового протокола использовался шаблон MSC/PR диаграммы, имеющий следующий вид:

mscdocument *<имя файла>*;
msc *<имя базового протокола>*;
ENV#env: instance;
MODEL#model: instance;
all: condition PRE /*MODEL(model, *<состояние агента>*);
*<атрибуты и параметры агента>**/;
MODEL#model: action '*<локальное действие>*';
MODEL#model: out *<сигнал>*(*<параметры сигнала>*) **to ENV#env;**
ENV#env: in *<сигнал>*(*<параметры сигнала>*) **from MODEL#model;**
all: condition POST /*MODEL(model, *<состояние агента>*);
*<атрибуты и параметры агента>**/;
ENV#env: endinstance;
MODEL#model: endinstance;
endmsc;

Полученная диаграмма поступает на вход генератора базовых протоколов.

4. Текущий статус, апробация, возможные пути применения и развития

Тестирование разработанных средств проводилось на небольших проектах (порядка 1KLOC). Проверялись процедуры обработки вершин и алгоритмы сохранения потока управления (порядок следования базовых протоколов в соответствии с программной логикой). В ходе экспериментов было установлено, что построение модели для проекта, содержащего 50 функций, занимает около 3

секунд. При этом было получено порядка 1000 базовых протоколов.

Статус программной реализации данной разработки таков. На данный момент реализованы процедуры обработки и построения базовых протоколов для всех вершин АСД С-программ. Дальнейшая работа будет направлена на глубину анализа С-конструкций, так как сейчас полная обработка осуществляется только для выражений с унарными операциями над одним операндом и бинарными операциями над двумя операндами (присвоение). Кроме того, будут исследоваться возможности формализации ссылок и вызовов системных функций (например, функции работы над строками) для повышения адекватности модели.

Возможные варианты практического использования результатов данной работы следующие. Поскольку множество базовых протоколов содержит формальное описание реализуемого программой алгоритма, то полученная модель может использоваться для верификации поведенческих свойств модели. Кроме того, в процессе выполнения модели осуществляется проверка разных свойств безопасности программы (переполнение массивов, использование неинициализированных переменных и т.п.). С помощью VRS-технологии возможно построение графа вызовов С-приложения с целью изучения и анализа модели, а также для облегчения процесса навигации по множеству базовых протоколов.

Говорить о полной автоматизации построения формальных моделей С-программ с помощью предложенного подхода пока рано, так как генерируемые формализмы нуждаются в анализе и дополнении. В большинстве случаев необходимо дополнить пред- и постусловия протоколов формулами базового языка для внесения в них недостающей программной логики, которая не была разобрана по причине неглубокого анализа конструкций.

Однако мы уже в ближайшее время планируем апробацию представленных в работе средств на реальном проекте в компании Motorola с целью более точно выявить недостатки и наметить приоритетные направления развития. Помимо этого, дальнейшая работа будет направлена на адаптацию подхода к приложениям, реализованным на языках C++ и Java.

Заключение

Кроме доводки и промышленных экспериментов текущей версии предложенных в работе средств предполагается развивать следующие исследовательские направления. Используя базовые протоколы, полученные по исходному коду программы, для генерации трасс, мы тем самым получаем поведенческие сценарии, моделирующие поведение программы. Эти сценарии являются детальным описанием поведения программы, так как описывают её на низком уровне — уровне исходного кода. Порой такая детализация бывает излишней, и требуется получить описание поведения программной модели на более высоком уровне. Для этого можно использовать средства VRS, позволяющие создавать многоуровневые поведенческие модели (например, граф вызовов, модель программы на компонентном уровне). Кроме того, преимущество многоуровневых моделей, используемых в VRS-технологии, заключается в возможности их исполнения, а следовательно, получения многоуровневых трасс. Поведенческие сценарии верхних уровней могут служить как для верификации, так и для решения задачи восстановления документации.

Список литературы

[1] IEEE Standard Glossary of Software Engineering Terminology, ANSI/IEEE Standard 610.12–1990, New York, 1990.

[2] *Sipma H. B., Uribe T. E., Manna Z.* Deductive model checking // Proceedings of International Conference on Computer-Aided Verication, 1996.

[3] *Clarke E. M., Schlingloff H.* Model checking // A. Voronkov (ed.). Handbook of Automated Deduction. Elsevier, 2000.

[4] *Visser W., Havelund K., Brat G. et al.* Model checking programs. Automated Software Engineering Journal. 10(2). April 2003.

[5] *Fernandez J.-C., Jard C., Jeron Th., Viho C.* Using on-the-fly verification techniques for the generation of test suites // Proc. 8th Conference on Computer Aided Verification, volume 1102 of Lecture Notes in Computer Science, New Brunswick, August 1996.

[6] *Letichevsky A., Kapitonova J., Letichevsky Jr.et al.* Basic protocols, message sequence charts, and the verification of requirements specifications, Computer Networks // The International Journal of Computer and Telecommunications Networking. Vol. 49. N 5. 2005. P. 661–675.

[7] *Fisher G.* The Next Generation of Source Code Analysis, Klocwork, Whitepaper, February, 2008.

[8] *Летичевский А. А., Капитонова Ю. В., Летичевский А. А. (мл.) и др.* Спецификация систем с помощью базовых протоколов // Кибернетика и системный анализ. 2005. № 4. С. 3–21.

[9] *Baranov S., Jervis C., Kotlyarov V. et al.* Leveraging UML to deliver correct telecom applications in UML for Real: Design of Embedded Real-Time Systems by L. Lavagno, G. Martin, B. Selic (ed.). P. 323–342, Kluwer Academic Publishers, 2003.

[10] *Hoare C. A. R.* Communicating sequential processes, Prentice Hall. London, 1985.

[11] *Летичевский А. А., Капитонова Ю. В., Летичевский А. А. (мл.) и др.* Инсерционное программирование // Кибернетика и системный анализ. 2003. № 1. С. 19–32.

[12] ITU Recommendation Z.120. Message Sequence Charts (MSC), 11/99.

[13] *Дробинцев П. Д.* Интегрированная технология обеспечения качества программных продуктов с помощью верификации и тестирования: Канд. дис., СПГПУ. 2006. 238 с.

[14] *Карпов А. Н.* Технология настраиваемой генерации тестов по формальным спецификациям для встроенных приложений и программных интерфейсов, реализованных на Java-подобных языках: Канд. дис., СПГПУ. 2007. 145 с.

[15] *Голубев А. А.* Методики создания и внедрения агентов в прикладное и системное программное обеспечение для автоматизации тестирования и мониторинга встроенных вычислительных систем: Канд. дис., СПГПУ. 2007. 150 с.

[16] Klocwork Extensibility Interface User's Guide, Document version 1.3, KlocWork, 2006.