УДК 004.4

С. С. Захарченко
Петербургский государственный университет путей сообщения
Императора Александра I

ПОДХОД К КОЛИЧЕСТВЕННОМУ ОЦЕНИВАНИЮ РЕЗУЛЬТАТОВ ПОИСКА УЯЗВИМОСТЕЙ В ПАРАЛЛЕЛЬНЫХ ПРОГРАММНЫХ СИСТЕМАХ С ИСПОЛЬЗОВАНИЕМ МЕТОДА ПРОВЕРКИ НА МОДЕЛИ*

Метод проверки на модели позволяет определить, выполняются ли предъявляемые формальные требования на модели проверяемой системы. Для оценки защищенности системы требуется разработка количественных показателей результатов проверки на модели. В качестве модели параллельной программной системы используется система переходов, на основе которой даются определения маршрута и критического маршрута. Приводятся показатели для оценки полученных результатов, основанные на маршрутах достижения состояний реализации уязвимости. Полученные результаты демонстрируются на примере модели программной системы, реализующей принцип взаимного исключения процессов.

проверка на модели, критический маршрут, параллельные программные системы, поиск уязвимостей, показатели результатов верификации.

Введение

Метод проверки на модели зарекомендовал себя в качестве эффективного средства анализа поведения параллельных и распределенных программных систем. Существующие методы и алгоритмы позволяют проверять работу систем, пространство состояний которых имеет очень большие размеры.

В качестве результатов верификации предоставляется информация о том, выполняются ли заданные требования на представленной модели исследуемой программной системы. Инструмент (верификатор), с помощью которого осуществляется проверка, в случае выявления нарушения требования дает возможность получения последовательности переходов между состояниями, приводящими к этому нарушению. Эти данные позволяют рассчитывать значения показателей, которые в дальнейшем могут применяться для решения различных задач, в том числе для анализа эффектов, вызываемых внесением изменений в модель.

Количественный анализ широко применяется в методе вероятностной проверки на модели (Probabilistic Model Checking), основанном на использовании цепей Маркова и марковского процесса принятия решений [1]. Метод анализа с использованием марковского процесса принятия решений хорошо подходит для моделирования параллельных программных систем, однако для этого требуется построение вероятностной модели.

Предлагаемый подход основан на работе [2], в которой рассматривается построение графа атак для анализа сетевой защищенности и разработки необходимых мер противодействия нарушителю. Кроме того, предлагаемый подход к количественной оценке результатов верификации не требует построения вероятностной модели, т. к. использует результаты обычной проверки на модели. Каждое выявленное нарушение требования рассматривается как наличие некоторой уязвимости в программном обеспечении, а последовательность переходов для ее достижения – как способ реализации найденной уязвимости.

1 Модель параллельной программной системы

Параллельная программная система состоит из множества взаимодействующих

*Работа выполнена в рамках гранта 13-07-13105-офи-м-РЖД.

друг с другом процессов $\Pi = \{\Pi_1, \ldots, \Pi_N\}$. В качестве модели процесса $\Pi_i \in \Pi$ предлагается использовать систему переходов, представляющую собой направленный граф, в котором вершинами являются состояния процесса, а ребрами – переходы между состояниями. На основе синхронной композиции процессов можно получить бесконечную последовательность из конечного числа состояний, именуемую также вычислением. Такая последовательность будет описывать поведение системы в целом.

Результатом верификации, в случае наличия нарушений требований, является некоторая последовательность переходов между состояниями, приводящая к нарушению. Такая последовательность называется маршрутом. В отличие от вычисления системы маршрут является конечным, т. е. имеет как начальное, так и конечное состояния. Исходя из определения вычисления системы, маршрутом ρ между состояниями s_m и s_k будет являться конечная последовательность состояний:

$$\rho = s_m, \ldots, s_k, \qquad (1)$$

где $s_m, s_k \in S$ – состояния программной системы.

Не все маршруты в исследуемой системе являются безопасными: как известно, в программных продуктах постоянно обнаруживаются различные уязвимости. В данном случае маршрут представляет собой последовательность переходов между состояниями, которые ведут к состоянию, в котором нарушается заданное требование, выражающее признаки наличия уязвимости. Обозначим множество таких состояний как $S^{crit} \subseteq S$. Маршрут называется критическим, если в него входит хотя бы одно состояние $s \in S^{crit}$. В формальном виде множество критических маршрутов P^{crit} в исследуемой программной системе можно определить следующим образом:

$$\rho_j \in P^{crit},$$

$$\text{если} \quad \exists i : s_i \in S^{crit} \wedge s_i \in \rho_j. \qquad (2)$$

Формула (2) определяет требования к состояниям, входящим в маршрут. Однако возможна ситуация, когда маршрут нарушает некоторое правило обработки информационных объектов программной системы, что также может приводить к нарушениям требований безопасности. В таком случае маршрут также будет являться критическим. Пусть *rule* – некоторое правило переходов между состояниями. Оно задает некоторую последовательность переходов. Функция *CheckRoute* проверяет, выполняется ли правило *rule* для маршрута ρ. Тогда:

$$\rho_j \in P^{crit},$$

$$\text{если} \quad CheckRoute(rule, \rho_j) = false. \qquad (3)$$

Формула (3) выражает требования к конкретной последовательности ρ переходов между состояниями, определяемыми правилом *rule*.

Для спецификации требований используется темпоральная логика, позволяющая выражать требования к системе во времени. Ранее уже был проведен анализ сложности проверки на модели параллельной программной системы с использованием темпоральной логики линейного времени [3]. Учитывая полученные результаты, а также такие параметры, как сложность модульной верификации, сложность проверки правильности формул и наличие алгоритмов проверки «на лету», предлагается использовать логику LTL (linear temporal logic – темпоральная логика линейного времени).

Особенностью темпоральной логики линейного времени является описание событий для единственного пути вычисления. В интерпретации темпоральной логики основное отношение доступности – это развитие в течение времени [4], т. е. возможность перейти из одного состояния в другое с течением времени. Состояние s' доступно из состояния s, если состояние s на временной оси находится раньше состояния s' (рис. 1).

Рис. 1. Временная достижимость состояний

Такое отношение доступности обозначается как R (s, s'). Временная ось в данном случае является линейной и дискретной. Таким образом, модель темпоральной логики состоит из ω-последовательностей, т. е. бесконечных последовательностей состояний в виде $\sigma = s_0, s_1, \ldots$ В таких последовательностях состояние s_j доступно из s_i, если $i \le j$.

Формулы темпоральной логики линейного времени могут состоять из:

1. множества предикатов AP;
2. множества операторов булевой алгебры $\{\wedge, \vee, \neg\}$;
3. множества модальных операторов $\{\Box, \Diamond, \bigcirc, U\}$.

Таким образом, формулы, составленные из элементов указанных множеств, являются формулами темпоральной логики линейного времени.

2 Показатели оценки результатов верификации

Как было отмечено выше, результатом проверки на модели является условие выполнения проверяемого требования на модели программной системы. В случае отрицательного результата верификаторы позволяют получить последовательность состояний, которая приводит к нарушению. Такая последовательность является критическим маршрутом.

Однако обнаружение маршрута нарушения требования не дает какой-либо информации кроме того, что нарушение имеется. Для того чтобы оценить результаты проверки на модели, предлагаются показатели, основанные на работах [5] и [6]. Однако в данном случае показатели применяются для оценки параллельного программного обеспечения.

Количество маршрутов (NR – number of routes). Данный показатель определяет количество различных переходов, выполняемых исследуемой системой, которые приводят к нарушениям, определяемым формулами (2) и (3). Другими словами, он определяет количество способов выполнения некоторого запрещенного действия. Чем больше способов достижения состояния нарушения, тем выше вероятность использования уязвимости:

$$NR = | P^{crit} |. \qquad (4)$$

Кратчайший маршрут (SR – shortest route). Чаще всего в исследуемой системе обнаруживается несколько маршрутов достижения некоторого состояния нарушения. Данный показатель выражает минимальное количество переходов, которые должна выполнить исследуемая система для того, чтобы достичь такого состояния нарушения. Очевидно, что чем меньше необходимо выполнить переходов, тем выше вероятность эксплуатации уязвимости. В качестве точки отсчета принимается начальное состояние параллельной программной системы $s_0 \in S_0$.

$$SR = \min(l(\rho_1), l(\rho_2), \ldots, l(\rho_N))$$

для $$N = | P^{crit} |, \qquad (5)$$

где $l(\rho_i)$ – длина маршрута.

Средняя длина маршрута (ARL – average route length). Показатель дает возможность оценки всего множества маршрутов, позволяющих достичь заданного состояния нарушения. Чем ближе значение средней длины маршрута к значению длины кратчайшего маршрута SR, тем выше вероятность эксплуатации уязвимости:

$$ARL = \frac{\sum_{i=1}^{NR} l(\rho_i)}{NR}. \qquad (6)$$

Нормированная средняя длина маршрута (NARL – normalized average route length). Данный показатель позволяет увидеть более четкую картину по сравнению с показателем ARL, так как последний показатель не всегда дает полное представление об изменении защищенности в ту или иную сторону при устранении уязвимости, т. к. учитывает изменение количества критических маршрутов:

$$NARL = \frac{ARL}{NR}. \qquad (7)$$

Например, для системы, в которой существует четыре критических маршрута ($l(\rho_1) = 3$, $l(\rho_2) = 4$, $l(\rho_3) = 4$ и $l(\rho_4) = 5$) показатели ARL и NARL = 4. При устранении маршрута ρ_2 показатель ARL не изменит своего значения, а NARL станет равным 1,33. Следовательно, чем выше показатель NARL, тем меньше вероятность использования уязвимости.

Среднеквадратическое отклонение (SDRL – standard deviation of route length). Данный показатель дает возможность фильтрации некоторых маршрутов, длина которых больше среднего значения в пределах среднеквадратического отклонения:

$$SDRL = \frac{\sqrt{\sum_{i=1}^{NR}(l(\rho_i) - ARL)^2}}{NR}. \qquad (8)$$

Мода (Mo – mode). В контексте критических маршрутов параллельных программных систем мода показывает наиболее часто используемое количество переходов, необходимых для достижения состояния нарушения:

$$Mo = f(l(\rho_1), l(\rho_2), ..., l(\rho_N)), N = |P^{crit}|. \quad (9)$$

Медиана (Me – median). Позволяет получить длину маршрута, которая находится в середине всех значений длин маршрутов. Этот показатель удобно применять в том случае, когда показатель *ARL* не дает возможности адекватной оценки длины маршрутов для достижения заданного состояния нарушения. Кроме того, с помощью медианы можно определить часть программы, которой стоит уделить особое внимание с точки зрения защищенности. Для множества маршрутов, ранжированных по длинам, медиана вычисляется по следующей формуле:

$$Me = \begin{cases} l(\rho_i)_{\frac{N+1}{2}}, & \text{при нечетном количестве маршрутов} \\ \dfrac{l(\rho_i)_{\frac{N}{2}} + l(\rho_j)_{\frac{N}{2}+1}}{2} & \text{при четном количестве маршрутов} \end{cases} \qquad (10)$$

Формула (10) представляет собой стандартную формулу для расчета медианы в теории вероятностей, в качестве исходных данных для которой в настоящей работе используются результаты проверки на модели.

3 Пример использования показателей

Практическое применение полученных показателей осуществлялось для модели алгоритма Деккера [7]. Данный алгоритм реализует механизм взаимного исключения между параллельно работающими и взаимодействующими друг с другом процессами. Исходный код модели на языке Promela был взят из примеров, поставляемых вместе со средой JSpin. Графическое представление систем переходов процессов *p* и *q* исследуемой программной системы показано на рис. 2.

Вместе с приведенным примером в виде формулы LTL сформулировано утверждение о том, что оба процесса обязательно будут выполняться. Предикат *csp* показывает, что процесс *p* находится в критическом участке программы, а предикат *csq* – что процесс *q* находится в критическом участке программы. Таким образом, в модели программы проверяется отсутствие возможности одновременного нахождения процессов в критическом участке, выраженное формулой $\neg$ *(csp* $\wedge$ *csq)*. Под критическим участком программы понимается та часть, в которой процесс обращается к общим ресурсам.

Для получения большего количества исходных данных о результатах верификации в проверяемую формулу намеренно вносились случайные изменения. Измененные формулы LTL, а также количественные оценки результатов верификации представлены в таблице.

Заключение

Исходя из полученных значений, можно сделать вывод, что потенциальные уязвимости, описываемые правилами 1 и 4, будут использованы с меньшей вероятностью,

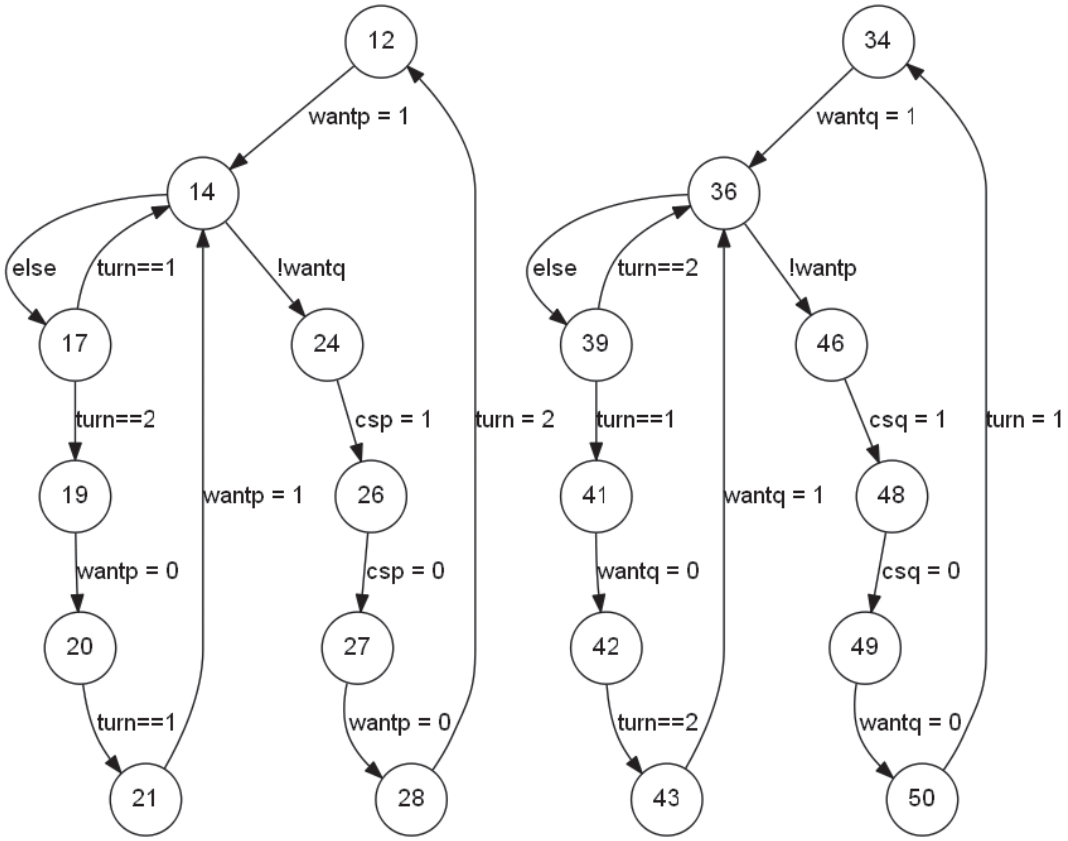

Рис. 2. Системы переходов процессов *p* и *q*

тогда как уязвимости 2 и 3 – с большей. В дальнейшем для получения подробной информации об исследуемом программном обеспечении необходимо проведение достаточного количества испытаний. Полученные экспериментальные данные могут быть использованы для расчета граничных значений для каждого показателя.

Таким образом, предложенные показатели позволяют рассчитывать количественные характеристики результатов выполнения проверки на модели. С их помощью предлагается осуществлять анализ защищенности параллельных программных систем путем расчета значений для исходной системы и для системы после попытки устранения выявленных нарушений требований.

Сравнение полученных результатов позволит сделать вывод о том, улучшилась или ухудшилась защищенность системы в измененной конфигурации. Кроме того, использование предложенных характеристик позволит получать оценку эффективности предлагаемых мер по устранению обнару-

ТАБЛИЦА. Анализ критических маршрутов

№	Требование	NR	SR	ARL	NARL	SDRL	Mo	Me
1.	$\square\neg\diamond csp \wedge \neg\square\diamond csq$	24	29	119,00	4,96	50,70	143	143
2.	$\square(csp \wedge \neg csq)$	148	1	80,09	0,54	36,73	81	81
3.	$\square(csp \wedge \square csq)$	284	1	129,71	0,46	68,78	199	124
4.	$\square\neg csp \wedge \neg\square\diamond csq$	12	27	106,17	8,85	46,19	–	128

женных уязвимостей в случае возможности устранения уязвимости несколькими различными способами.

Библиографический список

1. **Kwiatkowska, M.,** Norman, G., Parker, D. (2010). Advances and Challenges of Probabilistic Model Checking. Proceedings of 48th Annual Allerton Conference on Communication, Control and Computing, Urbana–Champaign, 1691–1698.

2. **Sheyner, O.,** Wing, J. M., Jha, S., Lippmann, R., Haines, J. (2002). Automated Generation and Analysis of Attack Graphs. Proceedings of the 2002 IEEE Symposium on Security and Privacy, Berkley, 273–284.

3. **Сложность** проверки модели параллельных программных систем / С. С. Захарченко // Программные продукты и системы. – 2012. – № 98. – Вып. 2. – С. 39–42.

4. **Manna, Z.,** Pnueli, A. (1981). Verification of Concurrent Programs. Part I: The Temporal Framework. Stanford, 62 p.

5. **Krautsevich, L.,** Martinelli, F., Yautsiukhin, A. (2010). Formal approach to security metrics. What does «more secure» mean for you? Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, Copenhagen, 162–169.

6. **Idika, N.,** Bhargava, B. (2012). Extending Attack Graph-Based Security Metrics and Aggregating Their Application. IEEE Transactions on dependable and secure computing, 9 (1), 75–85.

7. **Dijkstra, E. W.** (2002). Cooperating Sequential Processes. The Origin of Concurrent Programming, New York, 65–138.

УДК 004.75

Н. А. Игнатов
Московский государственный университет путей сообщения

ПОСТРОЕНИЕ ЗАДАЧИ ОПТИМИЗАЦИИ ПРЕДОСТАВЛЕНИЯ ВИРТУАЛЬНЫХ РЕСУРСОВ В ЦЕНТРАХ ОБРАБОТКИ ДАННЫХ, ОСНОВАННЫХ НА ОБЛАЧНЫХ ТЕХНОЛОГИЯХ

Рассматривается проблема расчета оптимального объема необходимых для обеспечения работы приложений виртуальных ресурсов, запускаемых в центре обработки данных, построенном с применением облачных технологий. Под оптимальным объемом виртуальных ресурсов понимается сочетание таких характеристик, как мощность процессора, объем оперативной памяти и устройства хранения данных, пропускная способность сети. Они предоставляются конечным пользователям посредством запуска определенного количества экземпляров виртуальных машин за минимальную стоимость в условиях соблюдения требований к качеству обслуживания системы. В ходе исследования построена и решена с использованием симплекс-метода задача оптимизации процесса предоставления виртуальных ресурсов, относящаяся к классу задач линейного программирования.

методы оптимизации, виртуализация, центр обработки данных, виртуальная машина, качество обслуживания.

Введение

Сегодня в транспортной отрасли активно применяются информационные системы, обеспечивающие деятельность различных направлений: административно-организационного, научно-производственного, финансово-экономического, информационно-