ПОДХОДИ ЗА МОДЕЛИРАНЕ И ВЕРИФИКАЦИЯ НА ПРОГРАМНО ОСИГУРЯВАНЕ

Даниела Орозова
Бургаски свободен университет

APPROACHES FOR SOFTWARE MODELING AND VERIFICATION

Daniela Orozova
Burgas Free University

Abstract: *Високата степен на интеграция на информационните технологии в почти всички приложения води до необходимост от реализиране на надеждни програмни и апаратни средства. Важна цел от обучението на софтуерните инженери в Бургаския свободен университет е развитието на навици и умения у студентите, свързани с процесите на разработване, поддържане и оптимизиране на софтуер, както и оценка за надеждността и правилността на софтуера. В статията се представят основни методи и средства за моделиране и верификация на софтуер.*

Key words: *modeling tools, formal code review, programme verification, pragramme synthesis, education in programming.*

През последните години, доказателството на правилността на програмите заема важно място в науката информатика. Грешки в софтуера, могат да доведат до сериозно нарушаване на работата на инфраструктурната мрежа и дори до човешки жертви. Това мотивира големите разходи на софтуерните компании за проверка на коректността (верификацията) на разработвания софтуер.

Процесът на програмиране се състои в писане на програми, анотиране на програмите с предусловия и постусловия, удовлетворяващи дадена входно/изходна спецификация и доказване на правилността им чрез различни методи за формална верификация. Всички популярни методи за формална верификация на програми обаче са трудоемки. Това води до необходимостта програмите да бъдат синтезирани, т.е. да бъдат построявани паралелно с доказателството, че са правилни.

Стандартът за софтуерна верификация и валидация IEEE 1012-2004 [1] определя верификацията на програмното осигуряване като техника, която проверява дали документите (техническо задание, модел на предметната област, описание на архитектурата, програмен код, потребителска документация и др.), създадени в хода на разработката на програмното осигуряване, съответстват на други документи, определени като начални (входни) данни, а също дали тези документи съответстват на правилата и стандартите. На базата на анализ на литературните източници се дефинират следните подходи за верификация на програмно осигуряване: софтуерна експертиза, статичен анализ, формални методи, динамични методи и синтетични методи.

1. Основни подходи за верификация на софтуер

1.1. *Софтуерна експертиза* (review, inspection). Стандартът за софтуерна експертиза IEEE 1028-1997 [2] дефинира процеса, в който софтуера се проверява от клиенти, представители на потребителите или други заинтересовани страни за обсъждане или одобрение. В практиката се използват методите: експертиза на кода (code review), програмиране на две лица (pair programming), инспекция (inspection), проиграване (walkthrough), техническа експертиза (technical review), формална експертиза на кода (formal code review) [3] и др. Недостатък на този вид верификация е, че не може да бъде автоматизиран и изисква активно участие на разработчиците. Ефективността им зависи от опита и мотивацията на реализиращите верификацията.

1.2. *Статичен анализ* (static code analysis) [4] е анализ на софтуера, който се провежда без реално изпълнение на изследваните програми. Методите от този вид се използват за проверка на правилността на формализациите на проверяваните свойства и за търсене на често срещани грешки чрез използване на шаблони. Известни са голям брой средства за статичен анализ. Възможностите на анализа, извършван от тези средства варира от анализ само на отделни оператори и декларации, до анализ, който включва пълния изходен код на програмата. Недостатък на средствата е, че са приложими за откриване на ограничен вид грешки. Предимство е, че могат напълно да се автоматизират.

1.3. *Формални методи* (formal methods) се прилагат за верифициране на свойства, които могат да се изразят формално в рамките на някои математически модели и за които могат да се построят съответни формални модели. Недостатъци на този вид методи са, че построяването на формалните модели и осъществяването на анализа на моделите може да се реализира от висококвалифицирани специалисти. Построяването на формалните модели не може да се автоматизира, то винаги се извършва от човек. Предимства на тези методи са, че с тях се откриват сложни грешки от логическо естество, практически неоткриваеми от методите на другите подходи. Формалните методи за верификация на програми се явяват най-надеждното средство, осигуряващо правилното функциониране на програмното осигуряване. В зависимост от вида на използваните модели, методите и средствата за формална верификация се разделят на:

- дедуктивен анализ;
- проверка на моделите за изпълнимост;
- проверка за съгласуваност.

1.4. *Динамични методи* (dynamic methods). Тези методи за верификация анализират и оценяват свойства на програмното осигуряване по резултатите от реалната работа или от работата на неговите модели и прототипи. Пример за такъв вид верификация е тестването [5, 6]. В зависимост от областта на теста, тестването се категоризира като:

- тестване на единична функция или клас (unit test);
- тестване на група програмни модули и/или класове (module test, system test);
- крайно тестване;
- тестване за приемане на софтуера (functional test; performance, stress test).

Основните недостатъци на тези методи са: позволяват да се намерят само грешки, които възникват по време на работата на софтуера; прилагането им изисква допълнителна подготовка по създаване на тестове, разработка на системи за тестване или мониторинг. Основни предимства на динамичните методи са, че чрез тях могат да се открият сериозни грешки и са предпочитани, защото изискват най-малко ресурси. Считат се за най-евтините техники за верификация и на практика са най-често използвани.

1.5. *Синтетични методи* (synthetic methods). Тези методи за верификация интегрират инструменти от подходите, описани по-горе. Целта е да се засилят предимствата и да се намалят недостатъците на методите за верификация. Най-разпространени методи от този вид са: верификация по време на изпълнение (runtime verification) [7] и тестване, основано на модели (model based testing) [8]. Проверяваните свойства се описват чрез формален модел и се вграждат в системата за тестване.

Съществуват и методи, които интегрират формална верификация със статичен анализ. Формалната верификация, като използва формални методи, доказва коректността или некоректността на програмното осигуряване. В сравнение със софтуерната експертиза, статичния анализ и динамичните методи за верификация на програмно осигуряване тя е най-ефективният и най-надеждният метод за верификация.

2. Модели за анализ на предметната област

Известните в литературата формални модели за анализ на програмно осигуряване могат да се разделят на [9]: модели, основани на свойства, изпълними модели и модели интегриращи двата подхода.

2.1. Модели, основани на свойства (property-based models). Тези модели от своя страна се разделят на логически и алгебрични модели.

2.1.1. Средства за изграждане на логически модели са:

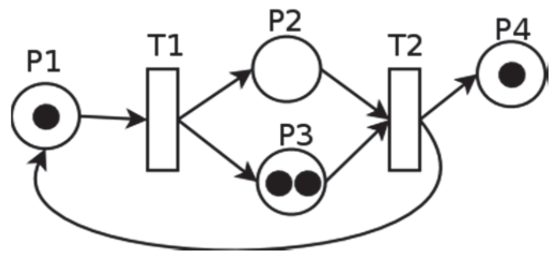
- Съждително смятане (propositional calculus) [10];
- Предикатно смятане (predicate calculus) [10];
- Предикатно смятане от по-висок ред (higher-order predicate calculus) [10];
- Ламбда смятане (lambda calculus) [11];
- Ламбда смятане от по-висок ред (higher-order lambda calculus) [11] ;
- Модални логики (modal logics) [12];
- Темпорални логики (temporal logics) [13];
- Линейни темпорални логики (linear temporal logic, LTL) [14];
- μ-смятане (или смятане с неподвижни точки, μ-calculus) [15];
- Логики с явно време (timed temporal logics) [13].

2.1.2. Средства за изграждане на алгебрични модели са:

- Релационни алгебри [16], лежащи в основата на релационните системи за управление на бази от данни;
- Алгебрични модели на абстрактните типове данни [17];
- Алгебри на процесите (process algebras, process calculus) [18].

2.2. Изпълними модели (executable models). Известни са следните средства за изграждане на изпълними модели:

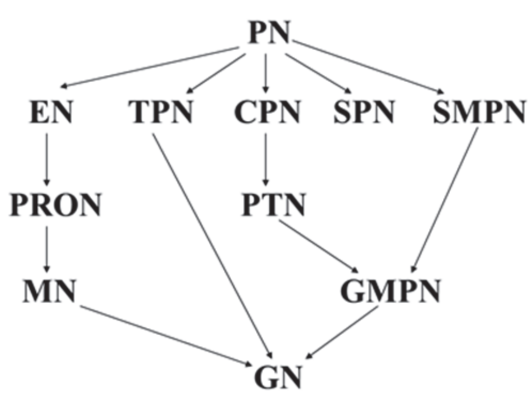
- Крайни автомати (finite state machine);
- Системи за преход (labeled transition systems);
- Взаимодействащи автомати (communicating finite state machines);
- Йерархични автомати (hierarchical state machines);
- Времеви автомати (timed automata);
- Хибридни автомати (hybrid automata);
- Абстрактни автомати (abstract state machines);
- Мрежи на Петри (PN – Petri nets) – просто и ясно средство за моделиране. Те имат много разширения, добавящи редица нови свойства и възможности за моделиране. Независимо от техните спецификации, всички тези инструменти имат позиции (обозначени с Pi) и преходи (обозначени с Ti) с дъги и ядра, които се движат в мрежата.



Фигура 1. *Пример на мрежа на Петри*

Основни разширения на мрежите на Петри са:

- o EN – Evaluation nets добавят продължителност на движението на токена;
- o TPN – Temporal PN добавят момент за активиране на прехода;
- o CPN – Color PN. При цветните мрежи на Петри всяко ядро има цвят и може да се движи само по дъги от същия цвят;
- o SPN – Stochastic PN. Изборът на дъга при стохастичните мрежи на Петри се основава на произволно генерирано число;
- o SMPN – Self Modifying PN. При самомодифициращите се мрежи на Петри, изборът на дъга се основава на генерирано двоично число 0 или 1, като ако се генерира 0, то преходът на ядрото не се извършва;
- o PRON – Pro-Net въвежда тип на прехода;
- o PTN – Predicate/Transition Net. Предикатно-преходните мрежи дефинират условие на прехода;
- o MN – M Net, при които ядрата изпълняват и допълнителни процедури;
- o GMPN – Generalized Modifying PN – ядрото се поглъща, когато предикатът генерира 0, а в противен случай при генериране на 1 ядрото извършва прехода.
- o GN Generalized Net – обединяват различните разширения в единен формализъм за описание на паралелно протичащи процеси. Различните разширения на мрежите на Петри и връзката между тях се представя на фигура 2.



Фигура 2. *Разширения на мрежите на Петри*

2.3. Модели, интегриращи моделите, основани на свойства и изпълними модели:

- Логика на Хоар (Hoare logic) [19];

- Обобщения на логиката на Хоар, динамични или програмни логики (dynamic logics, program logics) [19];
- Програмиране с договори (design by contracts) [20].

3. Средства за формална верификация на програмно осигуряване

За да се верифицира формално свойство на софтуера е необходимо формално да се провери определено съответствие между *моделите на спецификацията* и *реализацията*. В литературата се разглеждат няколко основни подхода.

3.1. Методи и средства за ***дедуктивен анализ***. В този случай верификацията на спецификацията S в реализацията P се свежда до доказване на изводимост, която най-често се записва чрез P $\vdash$ S. Последната се осъществява чрез методи, наречени дедуктивен анализ или доказателство на теореми (theorem proving).

При тези методи програмата се разглежда като формално твърдение, за което трябва да се докажат изразени чрез предикати свойства. Исторически първите методи за дедуктивен анализ на програми са предложени от Флойд и Хоар [19] в края на 60-те години на миналия век. В основата им лежат логиката на Хоар и предложената от Флойд и усъвършенствана от Манна и Пнюели техника за доказване завършването на изпълнението на операторите за цикъл. Последната се основава на инварианти на цикъл и монотонността на функцията за завършване на оператора за цикъл (ограничаваща функция, свързана с цикъла). С цел опростяване на техниката, предложена от Флойд, Дейкстра предлага техниката на преобразуващите предикати. Следват аналогични методи за верификация на програми, съдържащи масиви, указатели, обръщения към процедури и функции, а също и за верификация на паралелни програми. Прилагането на методите на дедуктивния анализ за верификация на практически значими програмни системи води до изграждането на специализирани средства за автоматично построяване на доказателства (provers, proof assistants). Тези средства могат да се разделят в две категории:

- средства, основани на разширени съждителни логики или логики от първи ред;
- средства, основани на логики от по-висок ред като: HOL (Higher-Order Logic), Isabelle, Coq или PVS (Prototype Verification System).

3.2. Методи и средства за проверка на модели. В този случай верификацията на спецификацията S в реализацията P се свежда до ***проверка на изпълнимост*** и се записва чрез P $\models$ S. За целта се използват методи за проверка на модела (model checking).

Същността на тези методи се описва чрез следните три стъпки. На първата стъпка се конструира модел на програмата, най-често основан на система от преходи. На втората стъпка моделът на програмата се допълва със спецификациите на свойствата, които трябва да притежава програмата. На третата стъпка се осъществява верификацията на свойствата на програмата. В процеса на работа на алгоритъма за проверка на модела, се построява множество от състояния на модела, в които се изпълняват спецификациите. В случай, че не е намерено състояние на модела, което не удовлетворява спецификацията, алгоритъмът връща *„истина“*. В противен случай, алгоритъмът връща *„лъжа“* и дава информация защо не е в сила спецификацията.

Първите методи за проверка на моделите са предложени в началото на 80-те години на миналия век. Те реализират *пълно изследване на модела на Крипке* с помощта на автоматични средства за проверка на формули от логиката CTL (Computation Tree Logic). След тези методи са разработени *символните методи*. При тях проверката се осъществява чрез обработка на Ordered Binary Decision Diagrams (OBDD) автомат, съответстващ на модела. Недостатъци на тези методи са, че не всяка система може да се представи в достатъчно компактен вид чрез OBDD, както и че за някои видове времеви логики алгоритмите за проверка на моделите са неефективни.

Някои от най-известните в практиката инструменти за проверка на модели са:

- BLAST (Berkeley Lazy Abstraction Software Verification Tool) – средство за проверка на модели за C програми;
- CADP (Construction and Analysis of Distributed Processes) – средство за проектиране на протоколи и разпределени системи. Инструменти за проверка на модели за различни темпорални логики и μ-смятане в системата CADP са модулите му EVALUATOR и XTL;
- GNTicker е инструмент за симулация на обобщеномрежови модели;
- UPPAAL е средство за моделиране, валидация и верификация на вградени системи и системи в реално време;
- NuSMV реализира символна проверка на модели;
- HyTech е автоматизирано средство за верификация на хибридни модели на вградени системи;
- Design/CPN е пакет от средства, поддържащ използването на цветни мрежи на Петри (CP-nets или CPN).

3.3. Методи и средства за ***проверка на съгласуваност***. В този случай верификацията на спецификацията S в реализацията P се осъществява чрез редукция или симулация. В литературата тези методи са известни като методи за проверка за съгласуваност. Методите за проверка на съгласуваност анализират съответствието между двата изпълними модела – на проверяваните свойства и на проверявания документ. Повечето използват тестване и затова понякога се отнасят и към методите за тестване на базата на модели. Съществуват и методи, които използват аналитични методи за проверка на съгласуваност. Сред най-известните среди за проверка на съгласуваност са Verity-Check и модулите BISIMULATOR и REDUCTOR на средата CADP за проектиране на комуникационни протоколи и разпределени системи.

Заключение

Анализирайки съществуващите методи за верификация на програми, софтуерните инженери на конкретното програмно осигуряване избират един или друг метод за верификация. За целите на обучението по програмиране, за проверка коректността на програмния код най-често се прилага тестването – динамичен метод за верификация на програми. Прилагат се, макар и по-рядко, и методи за формална верификация. Формалната верификация, за разлика от другите методи за верификация, е основана на математическо доказателство за коректност на програмата. В сравнение с другите методи за верификация на програми е най-ефективният и най-надеждният метод за верификация, но прилагането му изисква значителни усилия.

Това изследване е подкрепено от фонд научни изследвания на Бургаския свободен университет като част от проект Д-9/2020 „Data Science в образователното пространство за синя кариера“.

Литература:

1. IEEE 1012-2004 Standard for Software Verification and Validation. IEEE, 2005, http://pesona.mmu.edu.my/~wruslan/SE2/Readings/detail/Reading-7.pdf.
2. IEEE Std. 1028-1997, „IEEE Standard for Software Reviews“, IEEE Computer Society, 1997.
3. R. Radice, Software Inspections, http://www.methodsandtools.com/archive/archive.php? id=29

4. B. A. Wichmann, A. A. Canning, D. L. Clutterbuck, L. A. Winsbarrow, N. J. Ward, and D. W. R. Marsh, Industrial Perspective on Static Analysis, Software Engineering Journal, Mar. 1995, 69-75, http://www.ida.liu.se/~TDDC90/papers/industrial95.pdf
5. Standard glossary of terms used in Software Testing Version 2.2, 2012, http://www.astqb.org/documents/ISTQB_glossary_of_testing_terms_2.2.pdf
6. Software Testing-Testing Life Cycles, http://www.etestinghub.com/testing_lifecycles, php#2
7. A. Bauer, M. Leucker, Ch. Schallhart, Runtime Verification for LTL and TLTL, ACM Transactions on Software Engineering and Methodology (TOCEM), 2009. http://users.cecs.anu.edu.au/~baueran/publications/tosem-rv.pdf
8. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (eds.). Model Based Testing of Reactive Systems, Springer, 2005
9. http://is.ifmo.ru/books/_modelbased_testing_of_reactive _systems.pdf (646 pages).
10. Тодорова М., Подходи, програмни среди и езици за проверка на коректността на програми и прилагането им при подготовката на софтуерни специалисти, Хабилитационен труд, София, 2013.
11. Э. Мендельсон, Введение в математическую логику. Москва, Наука, 1971.
12. Х. Барендрегт, Лямбда-исчисление. Его синтаксис и семантика. М., Мир, 1985.
13. Р. Фейс, Модальная логика. Москва, Наука, 1974.
14. Y. Venema, Temporal Logic, http://staff.science.uva.nl/~yde/papers/TempLog.pdf
15. Linear-time logic, http://www.eecs.qmul.ac.uk/~pm/SaR/2004ltl.pdf
16. J. Bradfield, C. Stirling, Modal Mu-Calculi, 2005, http://homepages.inf.ed.ac.uk/jcb/Research/MLH-bradstir.pdf
17. Д. Орозова, Релационни и нерелационни бази от данни, Годишник на БСУ, 2017, 51-66, ISSN 1311-221-X.17.
18. J. Guttag, Algebraic Specification of Abstract Data Types, http://wwwsst.informatik.tu-cottbus.de/~db/doc/People/Broy/Software-Pioneers/Guttag_new.pdf
19. W. Fokkink, Introduction to Process Algebra, 2nd edition, 2007, http://www.few.vu.nl/~wanf/BOOKS/procalg.pdf
20. D. Harel, D. Kozen, J. Tiuryn, Review of Dynamic Logic, MIT Press, 2000, 459 pp, ISBN 0262082896, http://www.ccs.neu.edu/home/riccardo/papers/harel-dl.pdf
21. B. Meyer, Applying Design by Contract, IEEE Computer 25(10), Oct. 1992, pp. 40-51.
22. M. Todorova, D. Orozova, Training Difficulties in Deductive Methods of Verification and Synthesis of Program, International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 9, No. 7, 2018, pp. 18-22, ISSN (print):2158-107X, ISSN (online):2156-5570, doi:10.14569/IJACSA.2018.090703
23. M. Todorova, D. Orozova, Verification and Synthesis of Programs in Introductory Courses in Functional Programming, Proceedings of the 11th Annual International Technology, Education and Development Conference (INTED), Valencia,, 2017, стр.8195-8203, ISSN (print):2340-1079, ISBN:978-84-617-8491-2
24. M. Todorova, D. Orozova, Applying Deductive Verification to Bachelor Degree Courses in Programming, Proceedings of the the 10th Annual International Conference of Education, Research and Innovation, Seville, 16-18 November, 2017, стр.5055-5065, ISBN: 978-84-697-6957-7, ISSN (print):2340-1095
25. M. Todorova, D. Orozova, Software protection integrating registration-number and anti-debugging protections, Proceedings of the Ninth International Conference Information Systems & Grid Technologies, редактор/и: Vl. Dimitrov, V. Georgiev, издателство: St. Kliment Ohridski University Press, 2015, стр.138-151.