Годишник на секция "Информатика" **Annual of "Informatics" Section**
Съюз на учените в България **Union of Scientists in Bulgaria**
Том 3, 2010, 26-36 **Volume 3, 2010, 26-36**

Формални методи и средства за верификация на програмно осигуряване

Магдалина Тодорова
СУ "Св. Кл. Охридски", Факултет по математика и информатика
magda@fmi.uni-sofia.bg

Резюме: В статията е направен кратък преглед на методите за верификация на програмно осигуряване. Предложен е обзор на формалните методи и средства за верификация. Направена е класификация на известните методи за формална верификация в зависимост от вида на използваните модели на спецификацията и на реализацията на програмното осигуряване. Накратко са разгледани формалните методи и средства за дедуктивен анализ, за проверка на моделите и за проверка за съгласуваност.

1. Въведение

Увеличаването на сложността на програмното осигуряване (ПО) води до нарастване на количеството на грешките в него. Последното е свързано с увеличаване на загубите от тези грешки. Например, загубите на икономиката на САЩ от некачествено програмно осигуряване са от порядъка на 60 милиарда долара за година [1]. Могат да се приведат множество примери на грешки в програмното осигуряване, които са довели до нарушаване на работата на инфраструктурната мрежа, до разрушаване на космически апарати, до човешки жертви и др. Някои примери са коментирани в [2]. Това мотивира големите разходи на софтуерните компании за верификация на разработвания софтуер. В една типична търговска организация за разработка на софтуер разходите за верификация са от 50% до 70% от общите разходи за цялостната разработка на ПО [1].
Стандартът IEEE 1012 [3] определя *верификацията на ПО* като техника, която проверява дали артефактите (техническо задание, модел на предметната област, описание на архитектурата, програмен код, потребителска документация и др.), създадени в хода на разработката на ПО, съответстват на други артефакти, определени като начални (входни) данни, а също дали тези артефакти съответстват на правилата и стандартите.
В литературата са известни следните подходи за верификация на ПО:

- софтуерна експертиза (review, inspection);
- статичен анализ (static code analysis);
- формални методи;
- динамични методи;
- синтетични методи.

Софтуерна експертиза

В [4] софтуерната експертиза е определена като „процес или заседание, по време на което софтуерният продукт се проверява от персонала на проекта, от мениджъри, потребители, клиенти, представители на потребителите или други заинтересовани страни за обсъждане или одобрение". В практиката се използват методите: *експертиза на кода* (code review или peer review), *програмиране на две лица* (pair programming), *инспекция* (inspection), *проиграване* (walkthrough), *техническа експертиза* (technical review). Методите на софтуерната експертиза могат да се приложат към произволно свойство на ПО, към произволен артефакт на жизнения цикъл на ПО на произволен етап от разработката му. Недостатък на този вид верификация е, че не може да бъде автоматизиран и изисква активно участие на разработчиците на проекта, на мениджъри, потребители, клиенти, представители на потребителите и др. Основно предимство на този вид методи за верификация е високата им ефективност – по-голяма от тази на другите подходи за верификация. Тези методи намират от 50% до 90% от всички съществуващи грешки [5]. Ефективността им зависи от опита и мотивацията на реализиращите верификацията.

Статичен анализ

Статичният анализ е анализ на ПО, 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 срещащи се грешки чрез използване на шаблони. Сред за статичен анализ са: Parasoft [6], FindBugs [7], SourceAnalyzer [8], Svace Detector (Security Vulnerabilities and Critical Errors Detector) [9], T-SQL Analyzer [10]. Недостатъци на тези методи са, че са приложими само към кода или към определен формат за представяне на артефактите на ПО, както и че чрез тях се откриват ограничен вид грешки. Предимство на тези методи е, че напълно могат да се автоматизират, макар че понякога се налага ръчно да се определят някои компоненти.

Формални методи за верификация

Тези методи се използват за верификация на свойства, които могат да се изразят формално в рамките на някои математически модели, а също на тези артефакти, за които могат да се построят съответни формални модели. Някои методи и инструменти от този вид са представени в т. 3. Недостатъци на този вид методи са, че построяването на формалните модели изисква значителни усилия. Построяването на формалните модели и осъществяването на анализа на моделите може да се реализира от висококвалифицирани специалисти по формални модели. Наличието на такива специалисти не е голямо и цената на този вид верификация е висока. Построяването на формалните модели не може да се автоматизира, то винаги се извършва от човек. Съществуват инструменти, чрез които се автоматизира в значителна степен процесът на анализ на формални модели със сложност на промишлено равнище. Предимства на тези методи са, че чрез тях се откриват сложни грешки, практически неоткриваеми от методите на другите подходи, а също и че могат да се прилагат за верификация на апаратно осигуряване.

Динамични методи за верификация

Тези методи за верификация анализират и оценяват свойства на ПО по резултатите от реалната работа на ПО или от работата на неговите модели и прототипи. Пример за такъв вид верификация е тестването [11]. Използват се два подхода за тестване – тестване от тип „черна кутия" (equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing и specification-based testing) и тестването от тип „бяла кутия" (API testing, code coverage, fault injection methods, mutation testing, static testing). Основните недостатъци на тези методи са: позволяват да се намерят само грешки, които възникват по време на работата на ПО; прилагането им изисква допълнителна подготовка – създаване на тестове, разработка на системи за тестване или мониторинг, което понякога е доста трудоемка задача. Основни предимства на динамичните методи са, че чрез тях се откриват сериозни грешки и че са най-малко ресурсоемки. Считат се за най-евтините техники за верификация и на практика са най-често използвани.

Синтетични методи за верификация

Тези методи интегрират инструменти от подходите, описани по-горе. Най-често обекти на интегриране са методите за статичен анализ, формален анализ и тестване. Целта е да се засилят предимствата и да се намалят недостатъците на тези методи за верификация. Най-разпространени методи от този вид са *верификация по време на изпълнение* (runtime verification) [12,13,14,15] и *тестване, основано на модели* (model based testing) [16,17,18]. Верификацията по време на изпълнение комбинира формална верификация с изпълнение на програмата. Най-често проверяваните свойства се описват чрез формален модел и се вграждат в системата за тестване. Методите, реализиращи тестване, базиращо се на модели, използват формални модели за получаване на тестовете.

Формалните методи се явяват най-надеждното средство, осигуряващо правилното функциониране на програмното осигуряване. Въпреки трудностите при прилагането им, тези методи са обект на активни изследвания през последните години. В следващите части на статията е направен сравнително подробен преглед на формалните методи и средства за верификация.
Формалните методи за верификация на ПО се основават на изграждане на формални модели за анализ на свойства на програмното осигуряване.

2. Формални модели за анализ на ПО

Формалните модели за анализ на програмно осигуряване могат да се разделят на: *модели, основани на свойства* (property-based models), *изпълними модели* (executable models) и *модели, интегриращи модели, основани на свойства и изпълними модели*.

Модели, основани на свойства

Тези модели описват свойства на програмното осигуряване. Такива са *логическите* и *алгебричните* модели.

Примери за средства за изграждане на логически модели:

- *Съждително смятане* (propositional calculus) [19, 20];
- *Предикатно смятане* (predicate calculus) [19, 20];
- *Предикатно смятане от по-висок ред* (higher-order predicate calculus) [19];
- *Ламбда смятане* (lambda calculus) [21];
- *Ламбда смятане от по-висок ред* (higher-order lambda calculus) [21] ;
- *Модални логики* (modal logics) [22];
- *Темпорални логики* (temporal logics) [23];
- *Линейни темпорални логики* (linear temporal logic, LTL) [23, 24];
- *μ-смятане* (или смятане с неподвижни точки, μ-calculus) [25];
- *Логики с явно време* (timed temporal logics) [23].

Примери за средства за изграждане на алгебрични модели:

- *Релационни алгебри* [26], лежащи в основата на релационни системи за управление на бази от данни;
- *Алгебрични модели на абстрактните типове данни* [27];
- *Алгебри на процесите* (process algebras, process calculus) [28].

Изпълними модели

Всички видове изпълними модели могат да се смятат за разширение и обобщение на крайни автомати.

Примери за средства за изграждане на изпълними модели:

- *Крайни автомати* (finite state machine) [29];
- *Системи за преход* (labeled transition systems) [30];
- *Взаимодействащи автомати* (communicating finite state machines) [29];
- *Йерархични автомати* (hierarchical state machines) [31];
- *Времеви автомати* (timed automata) [32];
- *Хибридни автомати* (hybrid automata) [33];
- *Мрежи на Петри* (Petri nets) [34];
- *Цветни мрежи на Петри* (coloured Petri nets) [35];
- *Предикатно преходни мрежи* (predicate/transition nets) [36];
- *ω-автомати* [37];
- *Абстрактни автомати* (abstract state machines) [38];
- *Обобщени мрежи* (generalized nets) [39, 40];
- *Цветни обобщени мрежи* (coloured generalized nets) [39, 40].

Модели, интегриращи черти на модели, основани на свойства и изпълними модели

- *Логика на Хоор* (Hoare logic) [41];
- *Обобщения на логиката на Hoare*, *динамични* или *програмни логики* (dynamic logics, program logics) [42];
- *Програмиране с договори* (design by contracts) [43].

3. Методи и средства за формална верификация на ПО

За да се верифицира формално някакво свойство е необходимо да се провери формално определено съответствие между моделите на *спецификацията* (проверяваното свойство) и *реализацията* (проверявания артефакт). Възможни са:

	модел на спецификацията	*модел на реализацията*	*метод за формална верификация*
1	модел, основан на свойства	модел, основан на свойства	дедуктивен анализ (доказване на теореми)
2	модел, основан на свойства	изпълним модел	проверка на моделите
3	изпълним модел	модел, основан на свойства	не съществуват
4	изпълним модел	изпълним модел	проверка за съгласуваност

В първия случай верификацията на спецификацията S в реализацията P се свежда до доказване на изводимостта P ⊢ S. Последната се осъществява чрез методи, наречени дедуктивен анализ (theorem proving).
Във втория случай верификацията на спецификацията S в реализацията P се свежда до проверка на изпълнимостта P ⊨ S. За целта се използват методи за проверка на модела (model checking).
Третият случай практически не се реализира. Причината е, че така реализацията става по-абстрактна от спецификацията.
В четвъртия случай верификацията на спецификацията S в реализацията P се осъществява чрез редукция или симулация.

3.1. Методи и средства за дедуктивен анализ

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 [44] и Хоор [41] в края на 60-те години на миналия век. Използват съждителното смятане като формален модел, основан на свойства. В основата им лежат логиката на Hoare и предложената от Флойд техника за доказване завършването на изпълнението на операторите за цикъл. Последната се основава на инварианти на цикъл и монотонността на ограничаваща функция, свързана с цикъла. С цел опростяване на техниката, предложена от Флойд, Дейкстра [45] – 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 [46]. 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

Най-известните средства в тази категория са: ***theorem prover Vampire*** [47] – среда за автоматично доказване на теореми за класическата логика от първи ред; ***KeY*** [48] – формално средство, което интегрира дизайн, реализация, формална спецификация и формална верификация на обектно-ориентиран софтуер, притежава средство за доказване на теореми, изразени чрез динамичната логика от първи ред; ***ACL2*** [49] – език за програмиране, чрез който могат да се моделират компютърни системи и инструмент за доказване на свойства на модели; ***E theorem prover*** [50] – модерно средство за доказване на теореми за логики от първи ред с равенства; ***Minlog*** [51, 52] – интерактивен доказвач на теореми, базиращ се на естественото дедукционно смятане от първи ред; ***Waldmeister*** [53] – високоефективно средство за доказване на теореми, изразени чрез равенствената логика; ***Darwin*** [54] – ефективно средство за доказване на теореми, изразени чрез моделно-еволюционното смятане (Model Evolution Calculus).

Средства, основани на логики от по-висок ред

Най-широко известни и използвани такива средства са: ***PVS*** (*Prototype Verification System*) [55] – система за верификация, базираща се на спецификационен език, интегриран с допълнителни инструменти и средства за доказване на теореми; ***HOL*** (Higher-Order Logic) [56] – среда за интерактивно доказване на теореми в логика от по-висок ред; ***Isabelle*** [57] – интерактивна среда за доказване на теореми, наследник на HOL; ***Coq*** [58] – среда за доказване на теореми, изразени чрез смятането на индуктивни конструкции (Calculus of Inductive Constructions). При извършване на доказателства с тези системи често се налага намесата на висококвалифициран специалист, който трябва да формулира помощни леми или да промени схемата на доказателството.

3.2. Методи и средства за проверка на модели

Същината на тези методи се описва в три стъпки. На първата стъпка се конструира модел на програмата (ПО), най-често основан на система от преходи. На втората стъпка моделът на програмата се допълва със спецификациите на свойствата, които трябва да притежава програмата (ПО). Най-често спецификациите се задават чрез езиците на динамични или времени логики или чрез техни варианти с неподвижни точки. На третата стъпка се осъществява верификацията на свойствата на програмата (ПО). 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 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. Реализират пълно изследване на модела на Крипке с помощта на автоматични средства за проверка на формули от логиката CTL (Computation Tree Logic) [59]. След тези методи са разработени символните методи [60]. При тях проверката се осъществява чрез обработка на OBDD (Ordered Binary Decision Diagrams) автомат, съответстващ на модела. Последното позволява да се проверяват

модели с количество на състоянията до 10 000 – 10 500. Неудобство е, че не всяка система може да се представи в достатъчно компактен вид чрез OBDD. Оказва се, че за някои видове времеви логики алгоритмите за проверка са неефективни. За LTL и CTL те са линейни в зависимост от размера автомата, но са PSPACE-пълни в зависимост от дължината на проверяваната формула. За μ-смятането е намерен алгоритъм за проверка на модели [61], сложността на който може да се изрази чрез формулата $O((S\Phi)^{d/2}T\Phi)$, където S е броят на състоянията на модела, Φ е дължината на проверяваната формула, T е размерът на модела, а d – броят на вложените редувания на операторите на най-голямата и най-малката неподвижна точка във Φ. Голяма част от изследванията са свързани с търсене на различни фрагменти на μ-смятането, в които може да се приложат по-ефективни алгоритми.

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

BLAST (Berkeley Lazy Abstraction Software Verification Tool) е средство за проверка на модели за C програми [62];

CADP (Construction and Analysis of Distributed Processes) е средство за проектиране на протоколи и разпределени системи [63]. Инструменти за проверка на модели за различни темпорални логики и μ-смятане в CADP са модулите му EVALUATOR и XTL;

GNTicker е инструмент за симулация на обобщени мрежови модели [64, 65];
UPPAAL е средство за моделиране, валидация и верификация на вградени системи и системи в реално време [66];

NuSMV реализира символна проверка на модели [67];

HyTech е автоматизирано средство за верификация на хибридни модели на вградени системи [68];

Design/CPN е пакет от средства, поддържащ използването на цветни мрежи на Петри (CP-nets или CPN) [69].

Още методи и средства за проверка на модели могат да се намерят в [70].

3.3. Методи и средства за проверка на съгласуваност

Методите за проверка на съгласуваност анализират съответствието между двата изпълними модела – на проверяваните свойства и на проверявания артефакт. Повечето използват тестване и затова се отнасят към методите 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*** [71] и модулите BISIMULATOR [72] и REDUCTOR [73] на средата ***CADP*** за проектиране на комуникационни протоколи и разпределени системи.

4. Заключение

Формалната верификация, като използва формални методи, доказва коректността или некоректността на програмното осигуряване. В сравнение със софтуерната експер-

тиза, статичния анализ и динамичните методи за верификация на ПО тя е най-ефективният и най-надеждният метод за верификация, който има недостатъка, че прилагането му изисква значителни усилия и висококвалифицирани специалисти, които да го реализират. Пътят за преодоляването на тези недостатъци е интегрирането му с други методи за верификация и използване на инструменти за автоматично доказване на теореми и за автоматична проверка на модели.

Литература

[1] The Economic Impacts of Inadequate Infrastructure for Software Testing. NIST Report, May 2002, http://www.nist.gov/director/planning/upload/report02-3.pdf (последно посетена на 21.01.2011).

[2] Roosen-Runge, P. H., Software verification tools, part I, P. H. ROOSEN-RUNGE, 1999, 2003, 2007.

[3] IEEE 1012-2004 Standard for Software Verification and Validation. IEEE, 2005.

[4] IEEE Std. 1028-1997, "IEEE Standard for Software Reviews", IEEE Computer Society, 1997.

[5] Wong,Y. K., Modern Software Review: Techniques and Technologies. IRM Press, 2006, http://lib.mexmat.ru/books/24127 (последно посетена на 29.01.2011).

[6] http://en.wikipedia.org/wiki/Parasoft (последно посетена на 21.01.2011).

[7] Hovemeyer, D., W. Pugh, FindBugs™ Manual, University of Maryland, 2008 http://findbugs.sourceforge.net/manual/index.html.

[8] http://ru.wikipedia.org/wiki/SourceAnalyzer#Ссылки (последно посетена на 21.01.2011).

[9] Маликов, О.Р., В.С. Несов, Автоматический поиск уязвимостей в больших программах, Известия ТРТУ, Тематический выпуск "Информационная безопасность", №7 (62), 2006, с. 114-120, http://www.contrterror.tsure.ru/www/magazine7/07-28-Nesov-Malikov.htm (последно посетена на 21.01.2011).

[10] http://www.dbbest.com/p/TSQAnalyzer.aspx.

[11] http://en.wikipedia.org/wiki/Software_testing#Testing_methods (последно посетена на 21.01.2011).

[12] Bauer, A., 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 (последно посетена на 01.02.2011).

[13] http://www.time-rover.com (последно посетена на 07.02.2011).

[14] Cheon, Y., G. T. Leavens, A runtime assertion checker for the Java Modeling Language (JML). Proc. of International Conference on Software Engineering Research and Practice (SERP'02), CSREA Press, June 2002, pp. 322-328.

[15] Delgado, N., A. Gates, S. Roach, A Taxonomy and Catalog of Runtime Software-Fault Monitoring Tools, IEEE Transactions on Software Engineering, 30 (12), December 2004, pp. 859-872.

[16] Broy, M., B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner (eds.). Model Based Testing of Reactive Systems, Springer, 2005 (646 pages).

[17] https://goldpractice.thedacs.com/practices/mbt/

[18] Utting, M., B. Legeard, Practical model-based testing: a tools approach, ELSEVIER, 2007.

[19] Манна, З., Математическа теория на информатиката, София, Наука и изкуство, 1983.

[20] Мендельсон, Э., Введение в математическую логику. Москва, Наука, 1971.

[21] Барендрегт, Х., Лямбда-исчисление. Его синтаксис и семантика. М.: Мир, 1985.

[22] Фейс, Р., Модальная логика. Москва, Наука, 1974.

[23] Venema, Y., Temporal Logic, http://staff.science.uva.nl/~yde/papers/TempLog.pdf (последно посетена на 30.01.2011).

[24] http://www.eecs.qmul.ac.uk/~pm/SaR/2004ltl.pdf (последно посетена на 01.02.2011).

[25] Bradfield, J., C. Stirling, Modal Mu-Calculi, 2005, http://homepages.inf.ed.ac.uk/jcb/Research/MLH-bradstir.pdf (последно посетена на 01.02.2011).

[26] Roman, St., Access Database Design & Programming, Third Edition, O'Reilly Media, Inc., 2002, 978-0-596-00273-2, 448 pages.

[27] Guttag, J., Algebraic Specification of Abstract Data Types, http://www-sst.informatik.tu-cottbus.de/~db/doc/People/Broy/Software-Pioneers/Guttag_new.pdf.

[28] Baeten, J.C.M., A Brief History of Process Algebra, http://www.win.tue.nl/fm/0402history.pdf (последно посетена на 30.01.2011).

[29] http://spinroot.com/spin/Doc/Book91_PDF/F8.pdf(последно посетена на 30.01.2011).

[30] Panangaden, P., Notes on Labelled Transition Systems and Bisimulation, 2009 http://www.cs.mcgill.ca/~prakash/Courses/comp330/Notes/lts09.pdf (последно посетена на 27.01.2011).

[31] Rajee, A., M. Yannakakis, Model Checking of Hierarchical State Machines, ACM-TRANSACTION, 2002. http://www.cparity.com/projects/AcmClassification/samples/503503.pdf.

[32] Rajee, A., Timed Automata, http://repository.upenn.edu/cgi/viewcontent.cgi?article=1105&context=cis_reports (последно посетена на 30.01.2011).

[33] Henzinger, T., The Theory of Hybrid Automta, Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96), pp. 278-292.

[34] Peterson, J., Petri Nets, Computing Surveys, vol. 9, No. 3, 1977. https://www.rose-hulman.edu/class/se/OldFiles/csse373/Spring2009/Resources/peterson77.pdf (последно посетена на 30.01.2011).

[35] Jensen, K. Coloured Petri nets, EATCS Monographs on Theoretical Computer Science, Vol. 1, Berlin, Springer, 1992.

[36] Genrich, H. Predicate/transition nets. Lecture Notes in Computer Science, vol. 254, 1986, pp. 207-247.

[37] Thomas, W., Automata on infinite objects, In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume B: Formal Models and Semantics, Elsevier Science Publishers, Amsterdam, 1990, pp. 133-192.

[38] http://www.eecs.umich.edu/gasm/intro.html (последно посетена на 30.01.2011).

[39] Atanassov, K., Generalized nets. World Scientific, Singapore, New Jersey, London, 1991.

[40] Atanassov, K., On generalized nets theory. Prof. Marin Drinov Academic Publishing House, ISBN 978-954-322-237-7, Sofia, 2007.

[41] Hoare, C. A. R., An axiomatic basis for computer programming. Communications of the ACM, 12(10), 1969, pp. 576–585.

[42] Harel, D., D. Kozen, J. Tiuryn, Review of Dynamic Logic (Foundations of Computing), ACM DL, 2001.

[43] Meyer, B., Applying Design by Contract, IEEE Computer 25(10), Oct. 1992, pp. 40-51.

[44] Floyd, R. W., Assigning meaning to programs. Proc. of Symposium in Applied Mathematics. J. T. Schwartz, ed. Mathematical Aspects of Computer Science, 19, 1967, pp.19-32.

[45] Dijkstra, E. A., Discipline of programming, Prentice Hall, 1976.

[46] Keller, R., Formal Verification of Parallel Programs, Communications of the ACM, vol. 19, N. 7, July, 1976, pp. 371-384.

[47] http://www.vprover.org/ (последно посетена на 19.01.2011).

[48] Giese, M., Taclets and the KeY Prover, http://folk.uio.no/martingi/pub/uitp03.pdf (последно посетена на 30.01.2011).

[49] ACL2 Version 4.2, http://www.cs.utexas.edu/users/moore/acl2/ (последно посетена на 25.01.2011).

[50] http://en.wikipedia.org/wiki/E_theorem_prover (последно посетена на 30.01.2011).

[51] Schwichtenberg, H., H. Benl, U. Berger, M. Seisenberger, W. Zuber, Proof theory at work: Program development in the Minlog system, Automated Deduction, W. Bibel and P.H. Schmitt, eds., Vol. II, Kluwer, 1998.

[52] http://www.minlog-system.de/ (последно посетена на 09.02.2011).

[53] http://www.mpi-inf.mpg.de/~hillen/waldmeister/index.htm (последно посетена на 30.01.2011).

[54] http://combination.cs.uiowa.edu/Darwin/ (последно посетена на 28.01.2011).

[55] http://www.csl.sri.com/projects/pvs/ (последно посетена на 30.01.2011).

[56] http://hol.sourceforge.net/ (последно посетена на 24.01.2011).

[57] http://www.cl.cam.ac.uk/research/hvg/Isabelle/ (последно посетена на 30.01.2011).

[58] Huet, G., G. Kahn, Ch. Paulin-Mohring, The Coq Proof Assistant - A Tutorial, 8.2 2009, http://coq.inria.fr/V8.2pl1/files/Tutorial.pdf (последно посетена на 30.01.2011).

[59] http://en.wikipedia.org/wiki/Kripke_structure (последно посетена на 28.01.2011).

[60] McMillan, K., Symbolic Model Checking, Kluwer Academic Publishers Norwell, MA, USA, ISBN 0792393805, 1993.

[61] Browne, A., E.M. Clarke, S. Jha, D.E. Long, W. Marrero, An improved algorithm for the evaluation of fixpoint expressions, Theoretical Computer Science, vol. 178, Issue 1-2, May 30, 1997.

[62] http://mtc.epfl.ch/software-tools/blast/index-epfl.php (последно посетена 30.01.2011).

[63] http://en.wikipedia.org/wiki/Construction_and_Analysis_of_Distributed_Processes (последно посетена на 28.01.2011).

[64] Trifonov T., K. Georgiev, GNTicker – A software tool for efficient interpretation of generalized net models. Issues in Intuitionistic Fuzzy Sets and Generalized Nets, Vol. 3. Warsaw, 2005.

[65] Trifonov T., K. Georgiev, K. Atanassov, Software for modelling with Generalised Nets. Issues in intuitionistic fuzzy sets and generalized nets, Vol. 6, 2008, 36-42.

[66] http://www.uppaal.com/ (последно посетена на 03.02.2011).

[67] http://nusmv.fbk.eu/ (последно посетена на 02.02.2011).

[68] http://embedded.eecs.berkeley.edu/research/hytech (последно посетена на 30.01.2011).

[69] Lindstrøm, B., L. Wells, Design/CPN – Performance Tool Manual, 1999. http://www.daimi.au.dk/designCPN/man/Misc/Performance.pdf (последно посетена на 01.02.2011).

[70] http://en.wikipedia.org/wiki/List_of_model_checking_tools (последно посетена на 01.02.2011).

[71] http://www.veritable.com/Computer/Formal_Validation/Verity-Check/verity-check.html (последно посетена на 04.02.2011).

[72] http://www.inrialpes.fr/vasy/cadp/man/bisimulator.html (последно посетена на 30.01.2011).

[73] http://www.inrialpes.fr/vasy/cadp/man/reductor.html (последно посетена на 30.01.2011).