Българска академия на науките. Bulgarian Academy of Sciences
Аерокосмически изследвания в България. 14. Aerospace Research in Bulgaria
София. 1998. Sofia

Верификация на програмното осигуряване на компютърни системи за управление и системи за полунатурно моделиране чрез модели на Хоар [1]

Пламен Христов, Петър Гецов,
Пламен Ангелов

Институт за космически изследвания, БАН

Статията е продължение на "Подход за проектиране и изследване на компютърни системи за управление на летателни апарати на базата на модели на Хоар", публикувана в кн. 12/1996 г. на поредицата "Аерокосмически изследвания в България".

Верификация на програми – това е приложение на формални методи (ФМ) за доказване на коректността на програмната система преди началото на работата ѝ(а може би и в процеса на работа). Под коректност на програмата се разбира нейната адекватна реакция на различни входни данни, получаване на изходни данни за определено време (системи за реално време), липса на безкрайни цикли, безкрайно заемане на ресурси, което е особено сложно за доказване и достигане при паралелни системи.

Преди няколко години приложението на формални методи в програмното осигуряване (ПО) и в частност – в ПО на управляващи системи, се смяташе за недостатъчно обосновано по ред причини – приложението на сложните математични методи затрудняваше програмистите, смяташе се за невъзможно да се направи достатъчно надеждна спецификация и верификация на системата, приложението беше затруднено (а и сега е) при използване на класическата технология на програмирането, липсваха програмни пакети, автоматизиращи приложението на ФМ. Поради това първоначално намериха приложение методи, занимаващи се само със специфициране на ПО (Z - нотация [1-5] и др.). Отделно за управляващите системи и особено за сложните космически и авиационни системи се смяташе за недопустимо приложението на ФМ поради тяхната уникалност и сложните алгоритми.

[1] Изследванията се финансират от НФ"Научни изследвания" при МОНТ - дог. И-305/93.

Същевременно, по данни на NASA Langley Formal Methods Group [6], ред катастрофи с летателни апарати са станали поради ненадеждност на ПО, идваща от невъзможността за проверката му с класически методи.

- F14: попада в свредел поради грешки в тактическото ПО (Software Engineering Notes, Volume 9, Number 5).
- F18: катастрофира поради грешки в ПО (Software Engineering Notes, Volume 6, Number 2).
- AFTI-F16: лошо взаимодействие на процесите в програмната система, дрейф и шум в датчиците водят до отказ на системата. Приземява се с управление по един канал.
- X29: открита е грешка в ПО чрез симулации след 162 полета. Анализът показва, че грешката би могла да доведе до нестабилност и загуба на самолета.
- HiMAT катастрофира поради грешка по време в ПО, неоткрита при обширните тестове.

Защо са необходими ФМ и какво представляват?

Компютърната система може да не работи поради повреда във физическите компоненти или поради грешки в проекта/реализацията на ПО. За системи, които трябва да притежават свръхвисока надеждност и безопасност, трябва да се имат предвид и двата случая. Има установени технологии за откриване на повреди в компонентите, базирани на модели на Марков. Грешките в проекта на ПО представляват много по-голяма заплаха. За съжаление, няма научно обоснована защита против тази заплаха, която да е напълно приложима в практиката. Има три основни стратегии, които се прилагат в този случай:

1. Тестове (натурно моделиране);
2. Робастно ПО (Design Diversity), работоспособно при някои грешки (N - version програмиране, блокове за възстановяване и др.);
3. Избягване на грешките – ФМ (формална спецификация/верификация, автоматичен синтез на програми и др.).

Проблемът в натурното моделиране е необходимото време за доказване на свръхвисоката надеждност на системата. Например, за да се докаже 10^{-9} вероятност за отказ за 1 час мисия, системата трябва да се тества повече от 114 000 години.

Основната идея на робастното програмиране е да се използват отделни екипи за проектиране/реализация, за да се получат различни версии от една спецификация. След това се използват неточни прагови избиратели, за да се маскират ефектите от грешките в проектирането в една от версиите. Надеждата е, че недостатъците в проекта ще проявят грешките си независимо. За съжаление, този метод е отхвърлен при няколко експеримента с ПО с ниска надеждност. Освен това, този метод не може да бъде проверен за ПО с висока надеждност, тъй като се изисква прекалено много време.

Формалните методи, като че ли предлагат единствения защитен метод от грешки в проекта на програмата. За системите, за които е необходима свръхвисока надеждност, изглежда няма друг избор, освен да се проектират по най-строгия начин, известен досега, а именно – използване на ФМ [6].

ФМ се използват за специфициране и моделиране на поведението на системата и за математическа проверка на това, дали проектът и реализацията удовлетворяват функционалните и защитните изисквания на системата. Спецификацията, моделирането и верификацията могат да се направят с използване на различни техники и с различна степен на строгост. Една примерна разпространена последователност е показана по-долу:

Ниво 1: Формална спецификация на цялата или на част от системата.

Ниво 2: Формална спецификация на две или повече нива на абстракност и ръчно доказване на това, че детайлизираната спецификация съдържа по-абстрактната спецификация.

Ниво 3: Формални доказателства чрез програми, доказващи теореми.

В заключение на обзора може да се цитира [7] "Техниките, основани на формални методи, такива като езика CSP (Communicating Sequential Processes) доказано най-пълно удовлетворяват изискванията за преодоляване на сложността в специфицирането на конкурентни, вложени, разпределени системи и системи за реално време".

Днес съществуват ред готови програмни продукти, позволяващи специфициране и верификация на ПО с различна степен на абстрактност на спецификацията. Ще разгледаме накратко някои от тях.

• **B-Method** набор от математически методи за специфициране, проектиране и реализация на програмни компоненти. Системата се моделира като набор от взаимозависими абстрактни машини. Абстрактната машина се описва чрез AMN (Abstract Machine Notation), който е базиран на състояния език за формални спецификации. С AMN състоянието се моделира с използване на множества, отношения, функции, последователности и др. Методът определя как да се проверява спецификацията за устойчивост и как да се проверява проектът и реализацията за коректност (на данните и алгоритъма).

• **EVES/Verdi** – включва език за спецификации (Verdi), генератор на правила за доказване, автоматична система за заключение (NEVER), интерпретатор, компилатор. Verdi е формална нотация, базирана на нетипизирана теория на множествата, която може да се използва за представяне на строги математични концепции. Verdi се използва за доказване на теореми от теория на множествата, за доказване на функционална коректност на апаратни структури, програмни структури и др. EVES е система за верификация на програми, написани на Verdi [ORA – Canada].

• **FDR** (Failures-Divergence Refinement) – програмен пакет, който позволява автоматична проверка на много възможности на системи с краен набор състояния или диалогово изследване на процеси, които провалят тези проверки. Базиран е на стандартната (untimed) математична теория на взаимодействащите последователни процеси (CSP), която се интересува от реда, но не и от точното време на събитията. Разработен е специално за индустриални приложения. Прилаган е за проектиране на VLSI, настройка на мрежови протоколи, управление, обработка на сигнали и др.

• **PVS** – състои се от език за спецификации, предварително зададени теории, програма за доказване на теореми, утилити. Езикът за спецификации е базиран на класическа типизирана логика от високо ниво. PVC спецификациите са организирани в параметризирани теории, които съдържат предположения, дефиниции, аксиоми и теореми. Участници в проекта са NASA и други аерокосмически компании.

• **VDM – SL** (M Toolbox) – един от най-развитите формални методи, предназначен за функционална проверка на програмни системи. Централен елемент на VDM е езикът за спецификации – VDM – SL, който се използва на фазите на спецификация и реализация на ПО.

• **CABERNET** – поддържа формално изграждане на системи за реално време. Предоставя среда за специфициране, изпълнение и валидация на спецификациите на системи за реално време, базирани на временно и функционално разширение на мрежите на Петри.

• **VDM++** (Venus) – поддържа разработка на обектно-ориентирани конкурентни системи, на базата на комбинация от приложението на графично моделиране и формална спецификация. В края на процеса се достига до генериране на прототип.

Общи характеристики на разгледаните методи:
(по-скоро на процедурите за разработване на ПО с тях)

• В повечето методики се проверява спецификацията и реализацията, тъй като те са разработени за паралелни езици.

• Повечето методики проверяват за определени видове некоректност.

• Процесът на верификация обикновено е интерактивен и изисква достатъчно дълго време.

• Невъзможна е верификацията в реално време на автоматично създадени спецификации.

• Повечето продукти изискват сравнително мощни компютри (Sun SPARC, Silicon Graphics и др.).

• Верификацията е еднократен процес, протичащ (може би на няколко итерации) преди началото на работата на системата.

От друга страна, верификацията на една реконфигурираща се система трябва да бъде непрекъснат процес – всяка нова конфигурация се изразява и в нова спецификация (възможно – съставена без участие на човек), коректността на която от своя страна трябва да бъде доказана. Това изисква метод, позволяващ непрекъсната верификация. В блоково-модулните системи, при наличието на диалогов и/или автоматичен програмен генератор, ситуацията се опростява от това, че не е необходимо да се проверява крайната реализация на ПО, тъй като тя се генерира автоматично по спецификацията (проверява се само коректността на спецификацията).

Основни параметри на метода за верификация:

• **опростен формален подход** – приема се, че ако спецификацията отговаря на закона на Хоар за съответната структура на процесите, тя е коректна;

• "Основната отговорност" за коректността на програмата се прехвърля върху **спецификацията**. Тъй като се работи със стандартни структури и обекти (модели на Хоар) и формалната спецификация се съставя автоматично по технологичната, то се предполага, че програмата се съставя правилно.

Описание на метода. Спецификацията на системата е описанието на предполагаемото ѝ поведение. Когато системата е изградена от взаимодействащи процеси, най-естествено при наблюдение на поведението е разглеждането на нейния протокол и протоколите на съставящите я процеси. Спецификацията на един процес се състои от описание на неговите протоколи и техните свойства. Специфицира се последователността от действия, които извършва процесът, като особено внимание при взаимодействащите процеси се обръща на спецификацията на взаимодействията по каналите на процеса.

Ако P е обект, отговарящ на спецификацията S, то се казва, че P удовлетворява S, съкратено **P sat S**. Това означава, че S описва всички възможни резултати от наблюдението на поведението на P, или, с други думи, S = TRUE всеки път, когато нейните променливи приемат значения, получени в резултат от наблюдението на обекта P, т. е. –

$$\forall np.np \in \mathrm{trace}(\mathrm{P}) \Rightarrow \mathrm{S},$$

където *np* е променлива, която означава произволен протокол на процеса P.

Доказва се коректността не на програмни структури (от рода на *x:=y*, IF THEN ELSE, FORK, JOIN и др.), а на структури от обекти (процеси, агрегати и др.) – например:

$System = P_1 \| P_2$; CSP - описание на системата.

CSP - описание на процесите

$P_1 = Adder_{11} \rightarrow Prop_{11} \rightarrow Stat_{11} \rightarrow SepPoint_{11} \rightarrow Prop_{12}$;

$P_2 = Prop_{21} \rightarrow Prop_{22} \rightarrow SepPoint_{21} \rightarrow Stat_{21}$;

Протоколна системата:

$trace\,(System) = trace(P_1) \bigcup trace(p_2)$;

Протоколна P_1:

$trace(P_1) = (Adder_{11}, Prop_{11}, Stat_{11}, SepPoint_{11}, Prop_{12})$;

Протоколна P_2:

$trace(P_2) = (Prop_{21}, Prop_{22}, SepPoint_{21}, Stat_{21})$;

За коректността на такава структура е достатъчно протоколите (сумата от протоколите на съставящите я обекти) да отговарят на изведените в CSP закони.

Основен проблем. Извеждане на протоколите – *пр*, *пр'*, *пр''* и т.н. (извеждат се на базата на *азбуките* на процесите и агрегатите). Азбука на **процес**: {Prop, Stat1, Stat2, Matrix, SeparatePoint, Mul, Adder, Vector...}. Азбука на **агрегат**: {input, Transfer Function, output}. Transfer Function е множество от предавателни функции: пропорционална, апериодична, колебателна, сумиране и др.

- **Протоколи на процеса** – последователност от звена, които участват в процеса.
- **Протоколи на агрегата** – стойности на входа и изхода на агрегата.
- **Изчисляване на протоколите.**

В теорията на Хоар са изведени закони, позволяващи да се докаже, че процесът P съответства на своята спецификация S. Ето някои от тях:

а. STOP sat ($пр$ = <>) – това е процес, който не прави нищо и има празен протокол;

б. Ако P sat S, то (c $\rightarrow$ P) sat ($пр$ = <> V $пр_0$ = c & S($пр'$)), където $пр'$ е произволен протокол на (c $\rightarrow$ P). Всички закони са приведени в [8].

Законите на Хоар определят коректността на структури от процеси (c -> P; c->P | b->T; c->P || b->T и др.). Коректността на самите процеси - P, T и др., се определя от коректността на агрегатите, от които са съставени, тъй като връзките между агрегатите са линейни.

Определянето на коректността на агрегатите става на базата на техните спецификации. Спецификациите се определят на базата на дискретните преходни характеристики на звеното и интеграла на Дюамел и физически описват реакцията на агрегата при произволни входни въздействия. Общата формула за определяне на реакцията на агрегата е:

$$y(t) = h(t)\,x(0) + \int_0^t h(t-\tau)\,x(\tau)\,d\tau,$$

където $h(t)$ е преходната функция на агрегата, $x(0)$ – значението на $x(t)$ при $t = 0$, $y(t)$ – изходен сигнал, $x(t)$ – входен сигнал, или

$$y_k = -\sum_{i=1}^{n} a_i\, y_{k-i} + \sum_{j=0}^{m} b_j\, u_{k-j},$$

където a_i и b_j са коефициенти на диференчното уравнение, y_k и u_k – дискретни входен и изходен сигнал на агрегата.

– за агрегата П – звено:

$$\text{PA} = \left(0 \le np_i \le np'_{i-1} \Rightarrow | np'_i - K * np_i | \le 1\right),$$

където np_i са специални променливи, обозначаващи произволни протоколи на агрегатите (например – за РА – np е входен сигнал, а np' – изходен); np'_{i-1} е изходният сигнал на предходния агрегат, K – коефициент на усилване на звеното.

– за агрегата Апериодично звено от първи ред (AZ1) –

$$y(t) = e^{-\frac{t}{T_1}} y_0 + k \int_0^t e^{-\frac{t-\tau}{T_1}} u(\tau) d\tau.$$

Като се приеме t_0=(k-1)T и t=kT, се получава

$$y_k = e^{-\frac{T}{T_1}} y_{k-1} + k \int_{(k-1)T}^{kT} e^{-\frac{kT-\tau}{T_1}} u(\tau) d(\tau).$$

Диференчното уравнение в случая има вида

$$y_k = -a_1\, y_{k-1} + b_1\, u_{k-1}.$$

Коефициентите на уравнението имат следния вид:

$$a_1 = -e^{-\frac{T}{T_1}} \text{ и } \quad b_1 = k_1 T_1 \left\{ 1 - e^{-\frac{T}{T_1}} \right\}.$$

От тук спецификацията на AZ1 е:

$$AZ1 = (0 \le np_i \le np'_{i-1}) \Rightarrow | np'_i - \left(-a_1\, y_{k-1} + b_1\, u_{k-1}\right) | \le 1.$$

Определянето на коректността става чрез предварително изчисляване на протоколите на Хоар за процеси и агрегати и изследване на съответствието на тези протоколи на законите на Хоар. За процесите протокола се определя от последователността от агрегати, които го формират, и от връзките му с други процеси. За агрегатите предварително може да се определи само диапазона на изменение на променливата np. В този смисъл методът за верификация може да продължи работата си в реално време като част от метода за контрол на работата на системата чрез протоколи на Хоар.

Методът може да се представи като няколко последователни стъпки:

1. Определяне на типа (структурата) на процеса или подсистемата;
2. Изчисляване на протоколите чрез функцията trace (P).

Скелетът на функцията trace (P) е показан по-долу. Тя работи по типовите конструкции на Хоар (алтернативни процеси, паралелни процеси, рекурсивно определени процеси, процеси с избор и др.), протоколите на които са изведени в [8].

trace (P)= (R)* &

if P = ($STOP_p$) then trace (P) = {<>};

elsif P = (c $\rightarrow$ P) then trace (P) = {<> U }<c> $\wedge$ t|t $\in$ trace(P)};

elsif P = (c $\rightarrow$ P | d $\rightarrow$ Q)then trace(P)= {t|t = <> V (t_0 = c & t'$\in$ trace(P)V (t_0 = d & t'$\in$ trace(Q))};

elsif P = (x:B $\rightarrow$ P(x)) = {t|t = <> V ($t_0 \in$ B & t'$\in$ trace(P(t_0)))}
elsif P = (mX : A.F(x)) = $\bigcup_{n \geq 0}$ trace(F^n(STOP$_A$));
elsif P = (P || Q)then trace(P)= trace(P)$\bigcap$trace(Q);
else t = trace(P||Q) then
trace(P||Q)={t|\(t |α R)$\in$ trace(P)& (t |\α Q)$\in$ trace(Q)& t$\in$ (αR U αQ)*}
..... други конструкции на Хоар.....
else
end.

3. Проверка на протоколите (избор от множество стандартни протоколи) чрез функцията SAT(P,S(*np*)), която в общ вид е приведена по-долу.

В този смисъл методът е част от метода за контрол на работата на системата чрез протоколи на Хоар, тъй като протоколите могат да се проверяват както предварително, така и по време на работа.

SAT(P,S(*np*)) =
if P = STOP then
if *np* = <> then return TRUE;
else return FALSE;
end;
elsif P sat S(*np*) & (c $\rightarrow$ P)then
if (*np* = <> V (*np*$_0$ = c & S(*np'*))) then return TRUE;
else return FALSE;
end;
elsif P sat S(*np*) & (c $\rightarrow$ d $\rightarrow$ P)then
if (*np* $\leq$ <c,d> V (*np* i <c,d> & S(*np''*))) then return TRUE;
else return FALSE;
elsif P sat S(*np*) & Q sat T(*np*)& (c $\rightarrow$ P|d $\rightarrow$ Q) then
if (*np* = <> V (*np*$_0$ = c & S(*np'*)) V (*np*$_0$ = d & T(*np'*))) (*S(*np'*) и T(*np'*)) са спецификации на избраната алтернатива*)then return TRUE;
else return FALSE;
elsif $\forall$x $\in$ B.(P(x) sat S(*np*,x)) & (x:B $\rightarrow$ P(x)) then
if (*np* = <> V (*np*$_0$ $\in$ B & S(*np'*, *np*$_0$))) then returnTRUE; (*процес с избор от множество*)
else return FALSE;
elsif P sat S(*np*) & Q sat T(*np*)& P || Q then
if S(*np* |\ α P)& T(*np* |\ α Q) then return TRUE; (* паралелни процеси *)
else return FALSE;
------ други стандартни структури ------
else return FALSE;
end;

Програмно осигуряване на метода за верификация

ПО работи на базата на масиви от данни тип RECORD, в които е записана спецификацията на процесите, агрегатите (тип и параметри) и връзките между тях (каналите). Реализирано е на Modula-2.

Изчислителни експерименти с метода за верификация

С метода бяха извършени ред експерименти, като му бяха задавани коректни спецификации и такива, съдържащи некоректности – например при взаимодействие на няколко процеса по един физически канал може да

се зададе един от каналите да извършва непременно синхронно взаимодействие, т. е. – да изчаква готовност от комуникиращия с него процес. Това ще блокира работата на другите процеси, взаимодействащи по същия физически канал.

Изводи

Предимствата на метода са следните:

✓ предлага се опростен универсален подход;

✓ основан е на строга математична теория (CSP), което позволява поголямо доверие към резултатите от работата му;

✓ покрива всички видове некоректност на програмното осигуряване – deadlock, разходимост, безкрайно заемане на ресурси и др;

✓ позволява проверка както преди началото на работата на системата, така и в реално време.

Методът е предназначен за работа в средата на блоково-модулни системи, базирани на модели на Хоар, с предварително определен набор от системни обекти.

Литература

1. Gerhart, S.L. Applications of Formal Methods : Developing Virtuoso Software. – IEEE Software, Sept.1990.
2. Hall, A. Seven Myths of Formal Methods. – IEEE Software, Sept.1990.
3. Spivey, J. M. Specifying a Real–Time Kernel. – IEEE Software, Sept.1990.
4. Delisle, N., D. Garlan. A Formal Specification of an Oscilloscope. – IEEE Software, Sept.1990.
5. Kemmerer, R. A. Integrating Formal Methods into the Development Process. – IEEE Software, Sept.1990.
6. NASA Langley Formal Methods Program. URL: http://atb-www.larc.nasa.gov/.
7. Hinchey, M., S.A. Jarvis. Concurrent Systems: Formal Development in CSP. McGraw –Hill International Series in Software Engineering published 12th January 1995.
8. Hoare, C.A.R. Communicating Sequential Processes. PRENTICE –HALL International Ltd., London, 1985.

Постъпила на 9. IV. 1997

A software verification of computer control systems and real time simulation systems using Hoare's models

Plamen Christov, Peter Getzov, Plamen Angelov

(Summary)

Some possibilities of a formal methods application are considered and especially the Hoare's models application to the moving objects computer control systems and real time simulation systems software development. Some common questions of formal methods application are briefly described. A Hoare's models based method is presented. It is characterised by simplicity and automatic real time working. Ways and means of system objects Hoare's protocols are given and the procedures implementing the method too.