模型检验在航天测控软件上的应用研究

李运筹 尹 平
(北京跟踪与通信技术研究所 北京 100094)

摘 要 模型检验是确保程序质量的有效手段,能弥补软件测试的不足。但航天测控软件规模大、输入数据复杂、验证性质不明确等因素极大地阻碍了模型检验的应用。针对航天测控软件,分析了其特点以及对其执行模型检验的困难,提出了基于有界模型检验器 CBMC 的模型检验应用框架,包括航天测控数据的构造方法及验证性质的提取方法。随后,将该框架应用于外测数据处理软件,取得了良好的效果。

关键词 模型检验,CBMC,航天测控软件,蜕变测试
中图法分类号 TP311 **文献标识码** A

Research of Model Checking Application on Aerospace TT&C Software

LI Yun-chou YIN Ping
(Beijing Institution of Tracking and Telecommunication Technology, Beijing 100094, China)

Abstract Model checking is an efficient method to ensure software quality. However, complex input data and indistinct verification properties in large-scale aerospace TT&C software greatly hinders the application of model checking. After analyzing the characteristics of aerospace TT&C software and the difficulty of applying model checking, an application framework to aerospace TT&C software based on CBMC was proposed, including the construction method of aerospace measurement data and the extraction method of verification properties. The framework was then applied to the trajectory measurement data processing software and the result was satisfied.

Keywords Model checking, CBMC, Aerospace TT&C software, Metamorphic testing

1 引言

航天测控软件承担着航天器发射和飞行时的数据交换、遥外测数据处理、安控、显示等任务,是航天测控系统不可或缺的重要组成部分,对安全性、可靠性的要求较高,一旦发生系统失效,将造成严重损失甚至威胁相关人员的生命。软件测试是目前确保软件质量的主要方式,但是它有 3 点不足:1)测试效果取决于测试用例的设计;2)只能证伪而不能证明软件的正确性;3)实时、并发软件带来的执行不确定性造成测试发现的错误难以复现且问题定位困难。

模型检验是一种验证有限状态系统是否满足规范的形式化方法[1]。针对给定的验证性质,它能搜索系统的全部状态空间并给出正确性证明,或者在系统违反验证性质时给出反例执行路径,有效地弥补了软件测试的不足。自 1981 年 Clarke 等首次提出模型检验以来[2-3],模型检验已经有了成熟的理论基础并被广泛应用:现有成熟模型检验工具包括 CBMC[4], SLAM[5], SPIN[6], BLAST[7], Verisoft[8], Java PathFinder[9], NuSMV[10] 等;应用领域包括 MILS 系统[11]、高性能计算软件[12]、欧洲火车控制系统[13]、空客飞行控制系统[14]及众多嵌入式软件[15]。但航天测控领域程序规模过大、输入数据复杂及验证性质不明确等因素,造成模型检验在该领域的应用依旧十分有限。因此,研究如何将模型检验应用于航天测控软件显得至关重要。

本文在分析航天测控软件特点的基础上,基于有界模型检验器 CBMC,提出了航天测控软件的模型检验应用框架,包括针对测控数据的构造方法及针对数值计算型程序的验证性质提取方法。随后,将该框架应用于外测数据处理软件,成功找出了 CheckData 模块中隐藏的问题,为模型检验在航天测控软件上的应用提供了有效途径。

本文第 2 节介绍 CBMC 的相关知识;第 3 节分析航天测控软件应用模型检验面临的困难;第 4 节提出检验其应用模型的框架;第 5 节进行实验验证;最后总结全文。

2 CBMC 有界模型检验器

CBMC 是目前主流的针对 ANSI-C/C++程序的有界模型检验器,可运行于 Linux、Windows 及 MacOS X 平台。CBMC 直接对源码进行验证,无须对代码进行抽象建模。

2.1 CBMC 验证过程

CBMC 采用有界模型检验(Bounded Model Checking),其目标是检查验证性质在被验证程序上是否成立,其思想是将程序的状态集合和迁移关系转变为布尔公式,从而将程序输入空间的搜索转变为对符号状态空间的搜索,将程序验证问题转变为布尔公式的可满足性问题。

CBMC 验证程序的基本流程如图 1 所示,可分为 3 个步骤:1)语法解析并构建控制流图(CFG);2)循环展开并根据验证性质与执行路径构建布尔公式;3)使用 SAT/SMT 求解器求解。

李运筹(1994-),男,硕士,主要研究方向为程序验证、软件工程,E-mail:liyunchou94@163.com(通信作者);尹 平(1969-),女,博士,研究员,主要研究方向为软件工程。

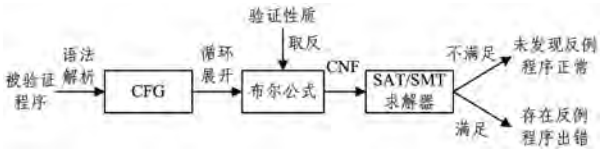

图 1 CBMC 验证流程

2.2 CBMC 验证性质

验证性质是 CBMC 判断程序是否正确的标准，是 CBMC 执行模型检验的核心。CBMC 使用断言(Assertions)来描述程序应该满足的性质。断言可由 CBMC 自动生成，也可由用户自行定义插入到源代码中。

自动生成的断言主要用于检查程序中常见的错误，减少验证时的重复工作。表 1 列出了 CBMC 现有的内嵌断言及其对应的验证性质[16]。

表 1 CBMC 内置断言列表

命令行参数	说明
--bounds-check	数组边界检测
--div-by-zero-check	除零检测
--pointer-check	指针检测
--memory-leak-check	内存泄露检测
--signed-overflow-check --unsigned-overflow-check	运算溢出检测
--float-overflow-check	浮点溢出检测
--nan-check	NaN 检测

CBMC 还支持用户在程序中插入自定义断言以验证程序的更多特殊性质，如并发程序执行顺序、特定逻辑运算结果等。自定义断言能够更准确地描述程序应该满足的条件，极大地拓宽了 CBMC 的验证范围，但如何提取断言，需要用户的进一步工作。

3 航天测控软件分析

本文提出使用 CBMC 对航天测控软件进行模型检验，主要原因有：1)我国航天测控软件主要由 C++语言开发，而 CBMC 是当前主流的支持 C++的模型检验器；2)CBMC 为开源工具，方便了解其原理并针对应用过程中的问题进行改造，可增强工具的适应性；3)CBMC 直接对源码进行验证，避免了对程序建模的过程，可大大降低验证难度及模型与实际软件不相符的风险。但应用 CBMC 对其进行模型检验时，面临的困难有以下 4 个方面。

(1)软件规模过大。以外测数据处理软件为例，其代码总量超过 25KLOC，这超出了模型检验工具的验证能力。

(2)软件中包含 CBMC 不支持的语法特性。比如 QT 库及 STL 库中大量出现的模板元编程。

(3)软件验证性质难以凝练。如何使用程序语言定义程序的正确性，是所有模型检验需要解决的核心难题。

(4)软件输入数据难以构造。例如，外测设备以 20Hz 的频率不断采集测量数据，若按模型检验的惯例，将每一时刻的测量数据都当作独立的输入变量将会带来以下两个问题：

1)输入变量数量过大。模型检验会搜索所有输入空间确定验证性质是否成立，其搜索空间的规模随变量数量成指数式增长。

2)忽略现实条件的限制。独立考虑每一时刻的测量数据会割裂输入数据间原有的时空联系，生成的反例可能不具备现实意义。

4 验证框架

4.1 概述

针对航天测控软件特点和应用模型检验的实际困难，本文提出应用 CBMC 验证航天测控软件的框架，如图 2 所示。

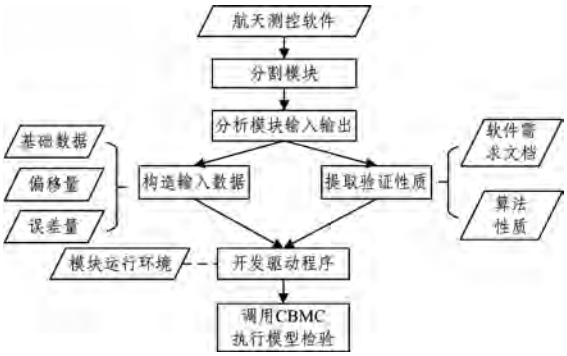

图 2 模型检验应用框架

首先借鉴组合验证思想，根据软件的功能需求、实现结构以及 CBMC 的验证能力，将程序分割为功能相对独立、耦合度低的模块，在分割过程中改写模块中不被支持的语法特性；然后分析每一模块的输入输出，并为模型检验构造输入数据、提取验证性质；最后开发驱动程序，整合模块的输入数据、运行环境以及验证性质，执行模型检验。

输入数据的构造与验证性质的提取是最核心也是最困难的环节。针对以测量数据作为输入的程序，若将每一时刻的测量量当作单独变量，输入变量的数目近乎无穷，远超 CBMC 验证能力。此外，如何提取验证性质、精准给定软件正确性的定义，是所有模型检验无法避免的核心问题。本文针对航天测控软件的特点，提出测量数据的构造方法以及验证性质的提取方法。

4.2 输入数据的构造

针对航天测控软件输入数据的现实特点，将测量数据分为基础数据、偏移量、误差量 3 部分，如图 3 所示。

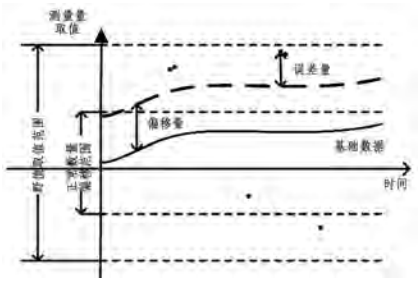

图 3 输入数据的构造

基础数据为选自实测或仿真测量数据的固定值，其中不包含偶然误差量，用于保留数据间的时空联系，使生成的数据具有现实意义。

偏移量会叠加在每一时刻的测量数据上，使基础数据能够自由平移，从而覆盖该测量量的所有合理取值范围，避免因选取固定的基础数据，造成搜索空间退化到单一的一组数据。

误差量用于定义测量过程中可能出现的偶然误差(主要是噪声与野值)，使数据符合现实情况。它会极大提升所构造的输入数据的不确定性，增强模型检验结果的可信度。

在执行模型检验时，偏移量与误差量的所有取值空间即为该模块的输入空间，输入量由五元组定义：

$$input \triangleq \langle bias, noise, wildNum, wildDef \rangle \quad (1)$$

其中，$bias$ 表示偏移量；$noise$ 表示噪声，通常仅考虑服从平均分布 $noise\sim N(a,b)$ 或正态分布 $noise\sim N(\mu,\sigma)$，故包含 2 个参数；$wildNum$ 表示野值总个数；$wildDef$ 包含对每个野值的定义 $\langle wildTime_i, wildValue_i\rangle (i=1,\cdots,wildNum)$，$wildTime_i$ 定义野值出现的时刻点，$wildValue_i$ 定义野值的取值，此时参数个数退化为 $1+2+1+2*wildNum$，在减少参数数量、剔除无意义输入空间的同时，保证了输入数据在合理取值范围内的正常变化且不受限制。

4.3　验证性质的提取

模型检验是针对验证性质的检验，提取验证性质是将用自然语言描述的程序正确性要求翻译为程序语言，并使之具体、精确且可量化的过程。提取验证性质时主要考虑两大信息来源：1）软件需求规格说明，其详尽地描述了软件适用的环境以及需要实现的功能，是提取验证性质的核心信息来源；2）软件实现的算法细节，针对数值计算型程序，可以根据软件所实现算法的特性，从侧面来验证算法实现的正确性。假设程序 P 实现了函数 f，$\vec{X}_1,\vec{X}_2,\cdots,\vec{X}_n$ 是 n 组不同的输入，$f(\vec{X})$ 和 $P(\vec{X})$ 分别为函数执行结果与程序执行结果。根据以上信息来源，可从以下两个角度来提取验证性质：

（1）基于软件单次执行结果 $P(\vec{X})$

通过判断软件单次执行的输出结果是否符合预期来直接验证程序的正确性。定义预期结果有两种途径：

1）直接定义软件的标准输出 $f(\vec{X})$ 为预期结果，比较实际输出 $P(\vec{X})$ 是否与之相等，如文献[17]中的方法。但对多数程序，尤其是数值计算型程序，标准输出 $f(\vec{X})$ 常常难以获取。

2）根据软件功能需求，定义软件标准输出 $f(\vec{X})$ 应该满足的条件，检查实际输出 $P(\vec{X})$ 是否满足。如：数据检择模块能检测并剔除输入测量数据 $\vec{X}=\langle x_1,x_2,\cdots,x_m\rangle$ 中的野值并进行平滑处理，其中 $x_j(j=1,\cdots,m)$ 为各时刻的测量值，其输出结果 $P(\vec{X})=\langle y_1,y_2,\cdots,y_m\rangle$ 应具有相对稳定的变化趋势。据此可提取性质：检择结果的任意两个相邻测量数据之差不超过一定阈值：

$$|y_j-y_{j+1}|<\varepsilon \tag{2}$$

（2）基于软件多次执行结果 $P(\vec{X}_i)(i=1,\cdots,n)$ 间的关系

部分情况下，程序单次执行的预期输出难以获取或描述，这是测试时存在的 oracle 问题[18]。因此，本文借鉴蜕变测试（Metamorphic Testing）思想[19]，基于程序所实现函数的特性（如周期性、单调性或其他更贴切的函数性质）考虑程序的多个执行结果之间的关系提取验证性质，从侧面验证程序的正确性。若 $\vec{X}_i(i=1,\cdots,n)$ 之间满足关系 r，则 $f(\vec{X})$ 之间满足 r_f：

$$r(\vec{X}_1,\vec{X}_2,\cdots,\vec{X}_n)\Rightarrow r_f(f(\vec{X}_1),f(\vec{X}_2),\cdots,f(\vec{X}_n)) \tag{3}$$

根据此蜕变关系，构造相应输入，可得到程序输出之间应该满足的关系，将其作为待验证性质：

$$r(\vec{X}_1,\vec{X}_2,\cdots,\vec{X}_n)\Rightarrow r_f(P(\vec{X}_1),P(\vec{X}_2),\cdots,P(\vec{X}_n)) \tag{4}$$

通过这种方式得到的性质，仅仅是程序正确的充分不必要条件。提取的函数性质越贴近函数本身，越具象化，验证的准确度越高，检错能力越强。再以数据检择模块为例，程序剔除野值的能力不应受野值具体取值的影响，据此提取性质：给定两组加入不同野值的测量输入 $\vec{X}_1$ 和 $\vec{X}_2$，二者的检择输出结果理应相同或仅有微小差别：

$$P(\vec{X}_1)\cong P(\vec{X}_2) \tag{5}$$

5　实验验证

5.1　被验证软件简介

本文将上述框架应用于航天测控软件中的外测数据处理软件，以验证模型检验框架的可行性与有效性。外测数据处理软件使用航天器发射、飞行时光学和无线电等外测设备的测量数据，解算出航天器的位置、速度信息。综合考虑软件的功能需求与实现结构，本文从外测数据处理软件源代码中分割出 CheckData 模块作为被检验程序。该模块执行过象限处理和数据检择操作，功能独立，是所有输入数据必经的处理环节，至关重要。此外，该模块与程序其他模块的耦合度较低，在执行过程中与其他模块没有复杂调用关系。

CheckData 模块首先对输入的方位角 A 测量数据进行过象限处理，再根据数据合理性判断算法剔除数据中的野值，并在适当的时刻基于历史数据采用最小二乘法平滑数据。若第 i 个测得的方位角数据为 A_i，历史检择结果保存为 A_i'，过象限处理的具体步骤为：

（1）$A_1'=A_1$。

（2）以 δ_0 为门限（一般取值范围为 350°～359°），判断：

$$|A_{i+1}-A_i'|>\delta_0 \tag{6}$$

$$|A_{i+2}-A'_{i+1}|\leqslant\delta_0 \tag{7}$$

若式（6）和式（7）同时成立，则此处数据过象限：

当 $|A_{i+1}-A_i'|<-\delta_0$，即连续测量得到的方位角数据逐渐增加直至越过 360°，重新从 0°开始增加时，令 $A'_{i+1}=A_{i+1}+360°$。

当 $|A_{i+1}-A_i'|>\delta_0$，即连续测量得到的方位角数据不断减小直至越过 0°，重新从 360°开始减小时，令 $A'_{i+1}=A_{i+1}-360°$。

若式（6）和式（7）不同时成立，则 $A'_{i+1}=A_{i+1}$。

（3）依次取 $i=2,3,\cdots,N-2$，重复步骤（2）。

实验在正确的程序中插入错误，注释了程序中的过象限处理部分。

5.2　验证过程与结果

实验在 Pentium（R）3.20 GHz Dual-Core CPU E6700，2 GB 内存，Ubuntu 16.04 环境下使用 CBMC v5.7 对 CheckData 模块执行模型检验。

使用外测数据仿真软件，得到一段时间内的测量数据作为构造输入的基础数据，部分数据如表 2 所列。

表 2　基础输入数据

编号	1	2	3	4
绝对时/s	37860.61	37860.66	37860.71	37860.76
方位角 A/°	347.76	347.89	348.02	348.15

随后，构造输入数据伪代码如下。

```
功能：构造输入数据
输入：仿真得到的原始测量数据集
输出：模型检验工具中的输入数据
begin
int bias=nondet_int();//生成偏移量
__CPROVER_assume(abs(bias) <=360);
//根据实际情况限定方位角偏移量范围
wildTime=GenerateWildTime();//生成野值发生位置；
while(读入基础数据 basicData !=null)
  basicData+=bias;
```

```
  if (是野值插入位置)
  double wildValue=nondet_double();
  //根据设备规格说明确定野值取值范围
  __CPROVER_assume(abs(wildValue) < 1000)
  basicData+=wildValue;
  end if
end while
end
```

除 CBMC 自动生成的断言外，如 4.3 节所述，为该模块构造的验证性质如下：

```
__VERIFIER_assert(fabs(Result[i]-Result[i-1]) <=fabs(Result[i-1]-Result[i-2]) * 10);
__VERIFIER_assert(fabs(Result1[i]-Result2[i]) <=Result1[i] * 0.05);
//根据功能需求，二者检择阈值分别设定为不超过上一次变化的 10 倍及不超过当前测量值的 5%
```

开发驱动程序将被验证程序模块、输入数据与验证性质整合为文件 checkData.cpp，原始 CheckData 模块含代码 218 行，最终文件含代码 511 行。使用如下命令执行模型检验：

```
cbmc checkData.cpp--function driver--unwind 8--trace--slice-formula--drop-unused-functions--error-label ERROR--all-functions
```

CBMC 将 CheckData.cpp 转换为含有 61891 个布尔变量以及 254199 个子句的 SAT 公式。用时 0.763s 找到违反性质的反例，输出片段如下所示，其中 *bias* 为偏移量。

```
** Results:
[__VERIFIER_assert.error_label.1] error label ERROR:FAILURE
…
Trace for __VERIFIER_assert.error_label.1:
…
State 20 file checkData.cpp line 263 function driver thread 0
---------------
  bias=12
…
```

CBMC 在有限的时间及内存下，准确地发现了以表 2 为基础构造的方位角输入上叠加 12°的偏移量后，程序过象限处理出现问题。

可以看到，该方法仅需指明软件输出需要满足的条件，便可自动查找软件缺陷，不需要依赖开发人员经验事先指明软件缺陷可能出现的位置。同时能以数据流形式，给出详尽的程序反例执行路径，极大方便了后续定位缺陷、修复缺陷的工作。

结束语 模型检验是提升软件质量的有力工具。针对航天测控软件对质量的要求及当前模型检验在该领域应用的匮乏，本文分析了航天测控软件的特点以及对其执行模型检验面临的困难。基于有界模型检验器 CBMC，提出航天测控软件的模型检验应用框架。其中，针对测量数据的复杂，提出基于基础数据、偏移量以及误差量的输入数据构造方法。针对数值计算型软件的特点，提出两类验证性质提取方法。随后将该框架成功应用于外测数据处理软件，取得了良好效果，证明该框架是可行和有效的。

下一步工作包括将该模型检验框架进一步应用于航天测控系统中的其他软件并加以完善。此外，探究、总结并构建验证性质库，以减少同类软件验证时的工作量。

参 考 文 献

[1] 王蓁蓁. 模型检验综述[J]. 计算机科学，2013，40(Z6)：1-14.

[2] CLARKE E M，EMERSON E A. Design and synthesis of synchronization skeletons using branching time temporal logic[C]// Workshop on Logic of Programs. Springer，Berlin，Heidelberg，1981：52-71.

[3] CLARKE E M，EMERSON E A，SISTLA A P. Automatic verification of finite-state concurrent systems using temporal logic specifications[J]. ACM Transactions on Programming Languages and Systems (TOPLAS)，1986，8(2)：244-263.

[4] CLARKE E，KROENING D，LERDA F. A tool for checking ANSI-C programs[C]// International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer，Berlin，Heidelberg，2004：168-176.

[5] BALL T，RAJAMANI S. Checking temporal properties of software with boolean programs[C]// Proceedings of the Workshop on Advances in Verification. 2000：2-9.

[6] HOLZMANN G J. The SPIN Model Checker：Primer and Reference Manual[M]. Addision-Wesley，2003.

[7] BEYER D，HENZINGER T A，JHALA R，et al. Checking memory safety with Blast[C]// International Conference on Fundamental Approaches to Software Engineering. Springer，Berlin，Heidelberg，2005：2-18.

[8] GODEFROID P. Software model checking：The VeriSoft approach[J]. Formal Methods in System Design，2005，26(2)：77-101.

[9] Jpf project[OL]. http://babelfish.arc.nasa.gov/trac/jpf.

[10] CIMATTI A，CLARKE EM，GIUNCHIGLIA F，et al. NuSMV：A new symbolic model checker[J]. International Journal on Software Tools for Technology Transfer (STTT)，2000，2(4)：410-425.

[11] DELANGE J，PAUTET L，KORDON F. Design，implementation and verification of MILS systems[J]. Software：Practice and Experience，2012，42(7)：799-816.

[12] PERVEZ S，GOPALAKRISHNAN G，KIRBY R M，et al. Formal methods applied to high-performance computing software design：a case study of mpi one-sided communication-based locking[J]. Software：Practice and Experience，2010，40(1)：23-43.

[13] CHIAPPINI A，CIMATTI A，MACCHI L，et al. Formalization and validation of a subset of the European Train Control System [C]// 2010 ACM/IEEE 32nd International Conference Software Engineering. IEEE，2010：109-118.

[14] BOCHOT T，VIRELIZIER P，WAESELYNCK H，et al. Model checking flight control systems：The airbus experience[C]// Proceedings of the 31st International Conference on Software Engineering (ICSE 2009). Companion Volume，IEEE，2009：18-27.

[15] HAVELUND K，LOWRY M R，PENIX J. Formal analysis of a spacecraft controller using SPIN[J]. IEEE Transactions on Software Engineering，2001，27(8)：749-765.

[16] KROENING D，CLARKE E. The CPROVER User Manual [EB/OL]. http://www.cprover.org/cbmc/doc/manual.pdf.

[17] METTA R. Verifying code and its optimizations：An experience report[C]// 2011 IEEE Fourth International Conference Software Testing，Verification and Validation Workshops (ICSTW). IEEE，2011：578-583.

[18] CHEN T Y，CHEUNG S C，YIU S M. Metamorphic testing：a new approach for generating next test cases：Technical Report HKUST-CS98-01[R]. Hong Kong：Department of Computer Science，Hong Kong University of Science and Technology，1998.

[19] 董国伟，徐宝文，陈林，等. 蜕变测试技术综述[J]. 计算机科学与探索，2009，32(2)：130-143.