CMurphi工具调研报告和使用规范

段凯强

# 主要功能

CMurphi，即Caching Murphi，是一个模型检测工具，目前版本是[5.4.9](http://mclab.di.uniroma1.it/site/softwareopensource/cmurphi5.4.9.tgz)，前身为斯坦福大学的Murphi。CMurphi目前的维护者是意大利Università degli Studi di Roma La Sapienza的Enrico Tronci所带领的团队。CMurphi使用C++语言开发，源码一般会随同软件一起发布。

CMurphi由Murphi编译器和Murphi描述性语言构成，运行于Linux系统中。

* Murphi编译器根据Murphi描述生成一个特定目的的验证器。这个验证器可以用来检查系统的特性，比如错误断言，不变量和死锁。
* Murphi描述语言是一个高级描述语言，针对状态有限的异步并行系统。Murphi在某些特性上是高级的，这些特性可以在很多高级编程语言如Pascal或者C中找到，同时这些特性也是Murphi的一部分。例如，Murphi拥有用户定义的数据类型，过程，以及描述的参数化方法。

CMurphi的使用过程如图1-1所示。

使用Murphi语言对系统建模

建立.m文件

.cpp文件

可执行文件

验证结果

图1-1 CMurphi的使用

# 安装和配置

CMurphi需要自行编译可执行版本。编译环境要求：

* Linux系统
* g++
* LEX词法分析器生成器，一般使用flex
* YACC语法分析器生成器，一般使用byacc
* GNU make

如上准备就绪后，进入CMurphi的src目录，运行命令

* make或者make mu，生成可执行文件
* make clean，清理编译过程中产生的中间文件
* make cleanall，清理所有中间文件，包括LEX和YACC所生成的文件

编译成功后，该目录即可出现可执行文件mu，可根据需要将CMurphi的安装目录添加到环境变量中。

# 使用指南

## 验证系统的步骤

1. 编写Murphi语言程序，命名如pingpong.m，注意一定要以.m结尾
2. 运行命令

$(murphipath)/mu pingpong.m

会生成文件pingpoing.cpp

1. 使用g++编译器编译cpp文件，注意需要使用-I参数指定需要包含的文件所在目录，即CMurphi中的include目录

g++ pingpong.cpp -I $(murphipath)/include

生成a.out可执行文件

1. 运行生成的可执行文件即可开始模型检测过程

./a.out <args>

## 执行模型

Murphi状态，是对描述中的所有全局变量的赋值。描述的执行可以由下边的无限循环生成。

无限循环：

寻找当前状态下所有的规则，这些规则的条件值均为true。也就是说，条件表达式的值为true，可以为全局变量赋当前值。

任意选择一个规则并且执行其动作，进入另一个状态。

注意，Murphi描述是非确定性的，因为在上边的第2步中进行了任意的选择。用户无法控制这个选择的过程，所以不管那条规则被选择，“正确”的Murphi程序必须能够做正确的事情。然而，一旦一条规则被选择，动作就是确定的了（有确定的下一个状态）。

这个执行模型对描述通过共享变量交互的异步系统（以不同的速度运行不同的进程）是有好处的，这种系统中，进程A通过写一个进程B将会读取的变量来影响进程B。信息传递可以通过读写缓冲区变量或者数组来进行建模。

Murphi模拟器在所有的规则中任意选择一条来进入新的状态。另一方面，验证器决定所有可能的选择的结果。当前的验证器通过广度优先搜索或者深度优先搜索来做到这个，这两种方法都在一个巨大的哈希表中存储状态，这样就可以在遇到一个曾经遇到过的状态时中断搜索。

当验证器生成状态时，会检查各种条件。有时会出现运行时错误，大部分都是赋值或者数组索引出界。有显式"assert"和"error"语句可以在一个动作中调用。如果这些语句中的一个发生了，验证器就会中止并且打印出诊断信息，包含导致由初始状态运行到错误状态的重新排序的状态列表。验证器也会在不变量表达式（用户给出的描述中的一部分）对当前状态的值为false或者当前状态“死锁”（无法成功跳出当前状态）的时候这么做（打印诊断信息）。

## 参数选项

如表3-1。

表3-1 CMurphi的参数选项

|  |  |  |
| --- | --- | --- |
| 类别 | 参数 | 含义 |
| 一般 | -h | 帮助 |
| -l | 许可信息 |
| 验证策略  默认-v | -s | 模拟器 |
| -v或-vbfs | 验证时使用广度优先搜索 |
| -vdfs | 验证时使用深度优先搜索 |
| -ndl | 不检查死锁 |
| 其他  默认  -m8  -p3  -loop1000 | -m<n> | 哈希表的存储量，单位是MB |
| -k<n> | 同上，单位是KB |
| -loop<n> | 允许循环最多执行n次 |
| -p | 使用冗余模式 |
| -p<n> | 每10n次事件报告一次进展，n范围为1~5 |
| -pn | 不打印进展报告 |
| -pr | 打印规则信息 |
| 错误跟踪  默认 -tn | -tv | 打印跟踪信息，默认与-td一起使用 |
| -td | 只写出与前一个状态的差异（在模拟模式下，只打印冗余模式下的差异） |
| -tf | 打印完整的跟踪信息 |
| -ta | 所有生成的状态至少打印一次 |
| -tn | 不打印跟踪信息 |
| 压缩技术  默认  -sym3  -permlimit10 | -nosym | 不使用对称约简 |
| -nomultiset | 不使用multiset约简 |
| -sym<n> | 使用对称约简 |
| -permlimit<n> | 在alg 3中检查的序列的最大数目，n含义  0 - 一般  1 - 彻底  2 - 快速  3 - 低存储  4 - 另一种快速 |
| Hash压缩  默认40位 | -b<n> | 用来存储hash值的位数 |
| -d dir | 将跟踪信息写入文件dir/sci.trace中 |

# 典型示例

以CMurphi自带的示例程序toy/pingpong.m为例。

1. 进入目录toy
2. 编译murphi代码

../../src/mu pingpong.m

输出结果为

|  |
| --- |
| ===========================================================================  Caching Murphi Release 5.4.9  Finite-state Concurrent System Compiler.  Caching Murphi Release 5.4.9 is based on Murphi release 3.1.  Caching Murphi Release 5.4.9 :  Copyright (C) 2009 - 2012 by Sapienza University of Rome.  Murphi Release 3.1 :  Copyright (C) 1992 - 1999 by the Board of Trustees of  Leland Stanford Junior University.  ===========================================================================  Call with the -l flag or read the license file for terms  and conditions of use.  Run this program with "-h" for the list of options.  Bugs, questions, and comments should be directed to  "melatti@di.uniroma1.it".  CMurphi compiler last compiled date: Mar 10 2015  ===========================================================================  Code generated in file pingpong.cpp |

1. 编译cpp文件

g++ pingpong.cpp -I ../../include

即可生成可执行文件a.out

1. 运行可执行文件

./a.out

结果中的Status为No error found表示验证成功。

|  |
| --- |
| ==========================================================================  Protocol: pingpong  Algorithm:  Verification by breadth first search.  with symmetry algorithm 3 -- Heuristic Small Memory Normalization  with permutation trial limit 10.  Memory usage:  \* The size of each state is 32 bits (rounded up to 8 bytes).  \* The memory allocated for the hash table and state queue is  8 Mbytes.  With two words of overhead per state, the maximum size of  the state space is 476219 states.  \* Use option "-k" or "-m" to increase this, if necessary.  \* Capacity in queue for breadth-first search: 47621 states.  \* Change the constant gPercentActiveStates in mu\_prolog.inc  to increase this, if necessary.  Warning: No trace will not be printed in the case of protocol errors!  Check the options if you want to have error traces.  ==========================================================================  Status:  No error found.  State Space Explored:  4 states, 6 rules fired in 0.10s. |