# NuSMV 使用教程

**摘要：**本教程简要介绍**模型检测**（Model Checking）技术，并提供对模型检测工具 NuSMV 的相关教程，包括安装方式、其接受的输入语言格式及使用实例。

## 一、 简要介绍

**模型检测**（Model Checking）是一种验证技术，它以蛮力搜索的方式遍历系统所有可能的状态。通过这种方式，可以证明给定的系统模型确实满足某个特性或者违反某个特性。目前模型检测最大的挑战是状态空间爆炸，最新的模型检测工具可以通过显式的状态空间枚举处理大约 $10^8$ 到 $10^9$ 个状态的状态空间，如果使用巧妙构造的算法和特定的数据结构，可以针对特定问题处理更大的状态空间（$10^{20}$ 个甚至更多状态）。模型检测最大的优势是能够毫无遗漏的发现系统所有的错误，比如模拟、仿真和测试未发现的细微错误。

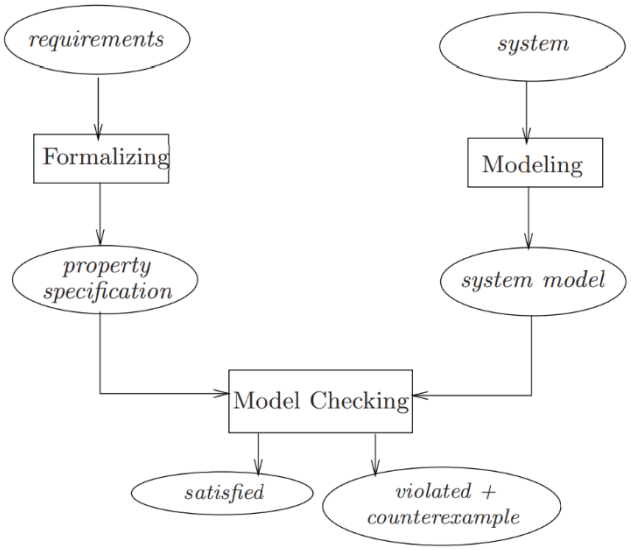



图 1 模型检测示意图

2001 年，基于 SMV(Symbolic Model Verifier)，Carnegie Mellon University(CMU)和 Istituto per la Ricerca Scientifica e Techolgica(IRST)联合开发出模型验证器 NuSMV，它主要是针对 SMV 2.4.4 版本的重新实现和扩展，重新定义了软件架构并加入了一些新特性。NuSMV 目前已发展到 2.6.0 版本。具体来说，NuSMV 从三个方面扩展了 SMV：

- 功能上，除了可以验证用 CTL 描述的规范外，还可以验证用 LTL 描述的规范；不仅实现了经典的基于 BDD 的符号模型检测技术外，还整合了基于 SAT 的有界模型验证技术(BMC)；提供了一个类似于 Unix 的 shell 的接口，方便用户使用。
- 相对于 SMV，NuSMV 定义了一个良好的软件系统架构，实现也更加模块化和开放，容易删除、替换或添加模块。例如，可以使用商用的 zchaff 包提供更加高效的有界模型验证技术。
- NuSMV 源码的注释、文档化更加完整，比 SMV 更加容易读和便于修改。这归因于 NuSMV 的一个目标是提供一个模型检测的通用平台，所以在编码上考虑到未来的扩展和修改。

**可用资源：**

1. NuSMV 工具网址：https://nusmv.fbk.eu/index.html
2. NuSMV 用户手册：https://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf
3. NuSMV 官方教程：https://nusmv.fbk.eu/NuSMV/tutorial/v26/tutorial.pdf

## 二、 NuSMV 的安装

**推荐第 2 种，方便快捷!!**

1、从源码安装（仅展示 GNU/Linux 系统），系统要求：GNU/Linux，比如 Ubuntu。（以下例程基于 Ubuntu 20.04 LTS amd64）

```
# 安装依赖
sudo apt install gcc g++ flex bison cmake tar gzip libxml2 libreadline6-dev doxygen texlive texmaker

# 在 Ubuntu 上进行编译
wget http://nusmv.fbk.eu/distrib/NuSMV-2.6.0.tar.gz
tar zxvf NuSMV-2.6.0.tar.gz
cd NuSMV-2.6.0/NuSMV
mkdir build
cd build
cmake ..
gedit code/nusmv/shell/cmd/cmdHelp.c
# 修改 58 行为: "int command_number;"
gedit doc/prog-man/cmake_install.cmake
# 删除 49 行内容: "/html"
gedit ../../cudd-2.4.1.1/util/pipefork.c
# 修改 43 行为: "#if (defined __linux__) || (defined __hpux) || (defined __osf__) || (defined _IBMR2) || (defined __SVR4) || (defined __CYGWIN32__) || (defined __MINGW32__)"
gedit ../../MiniSat/MiniSat_v37dc6c6_nusmv.patch
# 修改 679 行为: "+extern "C" void MiniSat_Delete(MiniSat_ptr ms)"
gedit ../../NuSMV/cmake/combine_grammar.py
# 修改 41 行为: "for key in sorted(d, reverse=True):"
make
sudo make install
```

使用方式：

```
NuSMV /usr/local/share/nusmv/examples/smv-dist/counter.smv
```

注：上述安装过程将 NuSMV 安装至目录/usr/local 中，其中示例文件目录为/usr/local/share/nusmv/examples。

2、下载二进制文件、解压运行即可。

1）GNU/Linux 系统，比如 Ubuntu：

```
# GNU/Linux libc6 (686) 32-bit
wget https://nusmv.fbk.eu/distrib/NuSMV-2.6.0-linux32.tar.gz
tar zxvf NuSMV-2.6.0-linux32.tar.gz
sudo cp -R NuSMV-2.6.0-Linux/* /usr/local/

# GNU/Linux libc6 (x86) 64-bit
wget https://nusmv.fbk.eu/distrib/NuSMV-2.6.0-linux64.tar.gz
tar zxvf NuSMV-2.6.0-linux64.tar.gz
sudo cp -R NuSMV-2.6.0-Linux/* /usr/local/
```

使用方式：

```
NuSMV /usr/local/share/nusmv/examples/smv-dist/counter.smv
```

注：上述安装过程将 NuSMV 安装至目录/usr/local 中，其中示例文件目录为/usr/local/share/nusmv/examples。

2）Windows 系统：

```
# Windows archive 32-bit (586)
# 下载地址 https://nusmv.fbk.eu/distrib/NuSMV-2.6.0-win32.tar.gz
# 解压缩包，比如至 C:\Program Files (x86)\NuSMV-2.6.0-win32
# 然后配置 PATH，编辑 path，添加 C:\Program Files (x86)\NuSMV-2.6.0-win32\bin
# 使用方式，打开 cmd 键入： NuSMV "C:\Program Files (x86)\NuSMV-2.6.0-win32\share\nusmv\examples\smv-dist\counter.smv"
```

注：上述安装过程将 NuSMV 安装至目录 C:\Program Files (x86)\NuSMV-2.6.0-win32 中，其中示例文件目录为 C:\Program Files

(x86)\NuSMV-2.6.0-win32\share\nusmv\examples。

```
# Windows archive 64-bit (x86)
# 下载地址 https://nusmv.fbk.eu/distrib/NuSMV-2.6.0-win64.tar.gz
# 解压缩包，比如至 C:\Program Files (x86)\NuSMV-2.6.0-win64
# 然后配置 PATH，编辑 path，添加 C:\Program Files (x86)\NuSMV-2.6.0-win64\bin
# 使用方式，打开 cmd 键入： NuSMV "C:\Program Files (x86)\NuSMV-2.6.0-win64\share\nusmv\examples\smv-dist\counter.smv"
```

注：上述安装过程将 NuSMV 安装至目录 C:\Program Files (x86)\NuSMV-2.6.0-win64 中，其中示例文件目录为 C:\Program Files (x86)\NuSMV-2.6.0-win64\share\nusmv\examples。

3）MacOS 系统：

```
# MacOSX Darwin (x86) 64-bit
# 下载地址 https://nusmv.fbk.eu/distrib/NuSMV-2.6.0-macosx64.tar.gz
# 其余步骤请参考 1）
```

## 三、 实例介绍

NuSMV 用 smv 输入语言（规定的一种文本格式）来描述 Kripke 结构和待验证的规范。在 NuSMV 中 Kripke 结构常称为 Finite State Machine（FSM）。其输入语言中，表达式和语句类似于 C 语言。NuSMV 有两个重要的表达式：init 表达式和 next 表达式。

- init 表达式用于描述初始状态；
- next 表达式用于描述转移关系。

其程序（比如 counter.smv）常被称为 smv 程序，由模块（MODULE）构成。模块由模块名和模块定义组成，模块定义又由形参（parameter）

和主体（body）部分组成。模块主体部分分为三类：Variables 部分、Constraint 部分和 Specification 部分。

- Variables 部分用于描述 Kripke 模型的状态集；
- Constraint 部分用于描述 Kripke 模型的转移关系和对模型的一些限制；
- Specification 部分用来描述待验证规范。

另 smv 程序至少要有一个称为 main 的模块，且 main 模块不能有形参。可以使用多个模块描述 FSM，然后组合成一个整体的 FSM。

一个典型的 smv 程序的结构如下：

```
MODULE main //至少要有一个 main 模块，为系统建模
    VAR
        ... //状态变量声明
    ASSIGN
        ... //初始状态和转移关系的声明
    SPEC(或 LTLSPEC、CTLSPEC) //规范定义，可选
        ... //使用 CTL 或 LTL 描述特验证的系统规范
MODULE submodule //各个子模块的定义，可选
    ... //同 main 模快
```

1）Variables 部分有两大类：

- State Varibles（状态的赋值表示具体的某个状态）；
- Input Varibles（通过标记关系来表示状态）。

分别以关键字 VAR 或 IVAR 表示。Variables 的类型仅为 boolean、integer、enum、word、array 以及 set 类型。

2）Constraints 的种类有 assign、trans、init、invar、fairness 等，分别以关键字 ASSIGN、TRANS、INIT、INVAR、FAIRNESS 表示。

为了更加方便地描述 FSM，NuSMV 还引入了 DEFINE。DEFINE 定义的符号的可看成是一个宏。

3）Specification 部分可以使用 CTL 公式，也可以使用 LTL 公式。

以下为 counter.smv 源程序的示例：

```
MODULE main
    VAR
        bit0: counter_cell(TRUE);
        bit1: counter_cell(bit0.carry_out);
        bit2: counter_cell(bit1.carry_out);
    SPEC
      AG AF bit2.carry_out
MODULE counter_cell(carry_in)
    VAR
      value : boolean;
    ASSIGN
        init(value) := FALSE;
        next(value) := value xor carry_in;
    DEFINE
      carry_out := value & carry_in;
```

该程序为 3 位二进制计数器电路的模型。以下简要分析：

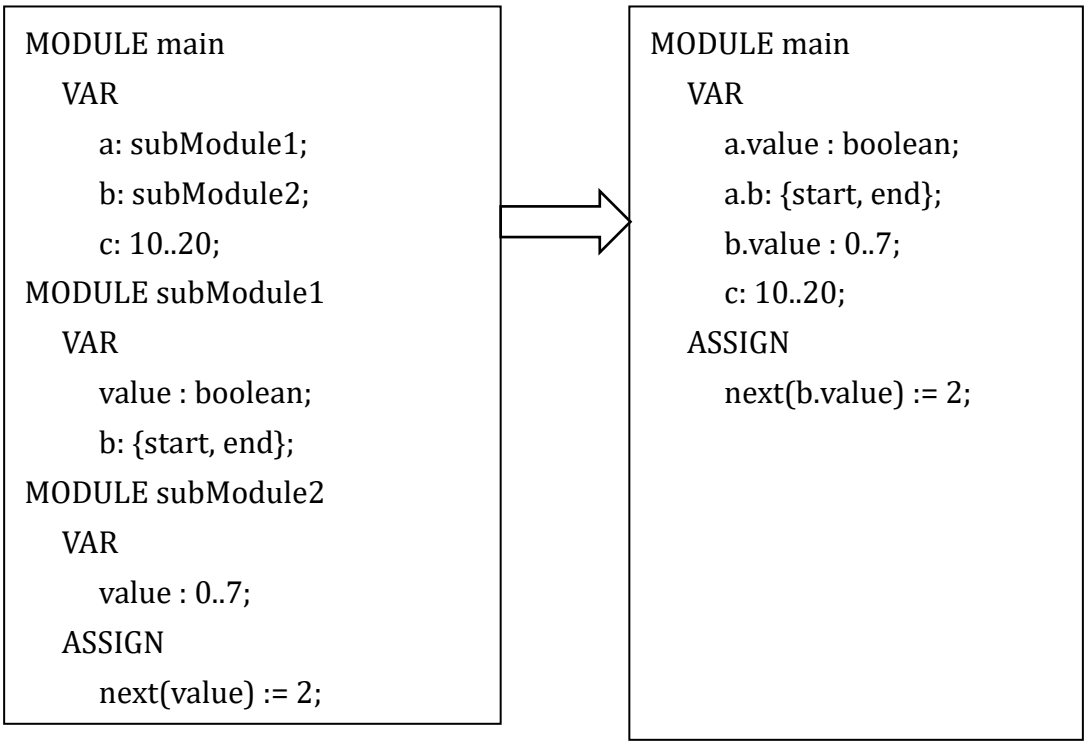
由 main 模块可知，调用了 3 次 counter_cell 模块，所以整体模块拥有 3 个 boolean 变量 bit0.value, bit1.value, bit2.value。ASSIGN 语句中的 init 指定初始状态为（bit2.value, bit1.value, bit0.value）=（0，0，0）。ASSIGN 语句中的 next 指定下一状态：

- bit0.value = bit0.value⊕TRUE = ¬bit0.value；
- bit1.value = bit1.value⊕bit0.carry_out = bit1.value⊕(bit0.value & TRUE) = bit1.value⊕bit0.value；
- bit2.value = bit2.value⊕bit1.carry_out = bit2.value⊕(bit1.value & bit0.carry_out) = bit2.value⊕(bit1.value & (bit0.value & TRUE)) = bit2.value⊕(bit1.value & bit0.value)。

由上述转移关系可知，（bit2.value, bit1.value, bit0.value）：（0，0，0）→（0，0，1）→（0，1，0）→……→（1，1，1）→（0，0，0）。因此该程序为 3 位二进制计数器电路的模型。

要验证的规范为 CTL 规范：AG AF bit2.carry_out，即 AG ( AF bit2.value & bit1.value & bit0.value )，该规范含义：对 CTL 计算树中所有路径，路径中所有节点，该节点的所有后续路径，路径中存在一个节点使得该节点满足 bit2.value = bit1.value = bit0.value = TRUE。从计数器的模型中，容易想象该规范是满足的（计数器从 000 一直增到 111，再变为 000，以此循环下去）。

事实上，含有多个模块程序可以转化为仅含一个 main 模块的程序。以下为示例：




注：上述程序中 b.value 表示 0-7 之间的整数，包含 0、7。更多示例查看 share/nusmv/examples 目录。