基于带参系统 Murphi 模型的 SMV 自动建模①

段凯强[1,2], 李勇坚[1]

[1](中国科学院软件研究所 计算机科学国家重点实验室，北京 100190)
[2](中国科学院大学，北京 100190)

摘 要： 提出了一种基于带参系统的 Murphi 模型来完成对应的 SMV 自动化建模的方法。因为 Murphi 工具拥有带参特性，因此使用其对带参系统进行建模比较容易，而且得到的模型代码量比较少，易于阅读、理解和修改；而 SMV 模型则能实现更丰富的控制，如进行快速不变式检查和限界模型检测等，但是建模过程复杂，模型不易维护。我们通过对两者进行分析，首先提出了能够很好描述带参系统的一个语义模型，然后读入相应的 Murphi 模型并进行分析以获取其语义模型表示，最后再通过一系列的策略自动得到限定参数时的 SMV 模型。

关键词： 带参系统；模型检测；自动建模；形式验证

Automatic Modeling in SMV based on Murphi Model of Parameterized Systems

DUAN Kai-Qiang[1,2], LI Yong-Jian

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract： We present a method for automatic modeling in SMV of a parameterized system based on its corresponding Murphi Model. Modeling in Murphi for parameterized systems is easy because its parameterization feature, and a Murphi model is not very complex to read, understand and modify. In the other hand, an SMV model could give more powerful operations, such as quick invariant checking and bounded model checking, but it is very hard to model in SMV for a parameterized system and to maintain an SMV model. We present a semantic model which is able to describe a parameterized system, then analyze a Murphi model to create its semantic model, finally we get its corresponding SMV model automatically by a series of conversion strategies.

Keywords： parameterized systems; model checking; automatic modeling; formal verification

为了使用不同的验证策略从不同的角度对带参系统进行形式化验证，我们常常需要用不同的建模语言分别进行建模，以分别利用其特性。例如，Murphi[1]作为一种 on-the-fly 模型检测工具，每次运行都会遍历状态空间对不变式集进行检查，可以使用其带参特性方便地对带参系统进行建模，且语法类似于高级编程语言，易于阅读、理解和修改；SMV 语言如 NuSMV[2]则是一种符号模型检测工具，能够实现更丰富的控制，如计算出可达集后快速进行多次不变式的可满足行检查，以及进行限界模型检测等；另外，还有 Spin[3]和 UPPAAL[4]等其他模型检测工具，也各有其特点，适合某种场景下的应用。

很多工作需要用到多种建模工具对带参系统进行建模[5]，但潜在的工作量就成为一个严峻的挑战，主要体现在以下几个方面：

① 需要使用不同的语言分别建立模型；

② 某些建模语言对带参系统的建模过程存在困难，如 SMV 语言；

③ 使用无带参特性的语言对带参系统进行建模时，一般需要将参数具体化，但往往会因状态爆炸导致模型过于庞大，人力无法胜任；

④ 使用无带参特性的语言所建模型往往比较复杂，不易阅读和维护；

⑤ 因存在多个不同语言版本的模型，当需要对模型做出调整时，同步维护非常困难。

① 收稿时间:0000-00-00;收到修改稿时间:0000-00-00

为了解决这些困难，我们提出了一个内部语义模型来描述带参系统。使用表达能力较强的建模语言如 Murphi 对带参系统进行初步建模和调试，再将其自动转换为内部语义模型，最后采用一系列的策略自动得到限定参数时的 NuSMV 模型。通过改变策略，也可以将得到的内部语义模型方便地转换为其他具有相同语义的语言版本的模型。

1 Murphi和SMV的建模能力

1.1 Murphi 的建模能力

1.1.1 基本表达能力

因为 Murphi 支持带参，且其描述语言接近高级语言，易于调试，因此特别适合对带参系统进行最初的分析，以准确描述带参系统的特性。Murphi 的建模能力主要体现在以下方面：

① 状态变量，用于描述带参系统特定时刻的一个状态，支持复杂的自定义数据结构；

② 初始状态，用于描述带参系统的初始状态；

③ 迁移规则，描述带参系统向下一个状态迁移所需要满足的条件以及迁移向何种状态；

④ 不变式，刻画需要验证的带参系统的性质，一般是一个或多个谓词公式；

⑤ 过程和函数，这是一种高级语言特性，用来简化建模过程，抽象出具有类似功能的代码片段，从而能够实现代码复用；

⑥ 带参能力，在 Murphi 中，状态变量、初始状态、迁移规则和不变式都支持参数化，从而为描述带参系统提供了极其强大的支持。

1.1.2 Murphi 的执行模型

使用 Murphi 进行建模的带参系统是以这样一种方式执行的：系统有一个或者多个初始状态，在初始状态满足一定条件的时候就会向后续状态发生迁移，这由一系列的规则进行描述；确切地说，系统处于某个状态的时候，有可能满足多条规则的迁移条件，此时系统会随机选择一个规则进行迁移。因为需要保证系统在任何可达状态下都满足其用不变式描述的性质，因此实际上需要遍历所有可达状态来完成验证。

既然这是一个遍历过程，因此 Murphi 的执行模型可以使用两种策略，即深度优先或者宽度优先，Murphi 默认使用宽度优先策略，也就是说，执行模型为一个循环，在循环内部，先从等待队列获取当前状态，然后筛选出迁移条件被满足的所有规则，然后将这些规则的后续状态放入等待队列中。

另外，需要指出的是，Murphi 规则之间的执行过程是原子的，也就是说不会有多条规则同时执行；而在一个规则的内部，其语句序列的执行是顺序的。

1.2 SMV 的建模能力

1.2.1 基本表达能力

SMV 所能够描述的系统与 Murphi 类似，但是不能支持参数化。具体来说体现在：

① 状态变量，不支持复杂的自定义数据结构；

② 赋值，通过 init 关键字刻画初始状态，通过 next 关键字刻画状态迁移；

③ 不变式，通常用谓词公式描述；

④ 模块，用于对建模过程进行一定程度上的抽象，从而实现代码复用；

⑤ process 类型的状态变量，能够模拟出类似于 Murphi 中的规则的功能。

1.2.2 SMV 的执行模型

SMV 的执行模型与 Murphi 的也非常相似，也是当满足一定条件是系统进入下一个状态，但是有一点与 Murphi 是完全不同的。在 SMV 中的一个赋值语句块中，所有用 next 关键字标记的赋值语句的执行是并行的，而不是顺序的。

2 带参系统的语义模型

2.1 标号迁移系统

通过上边的分析，不难发现，能够使用 Murphi 或者 SMV 描述的带参系统，都是能够使用标号迁移系统模型[6]描述的。

一个标号迁移系统是一个四元组$<\sum, S, \Delta, I>$，其中：

① $\sum$是标号集合，描述系统发生迁移的条件；

② S 是状态集合，描述系统的可达状态；

③ Δ 是 $S\times\sum\times S$ 的子集，是一个有标号的迁移关系，描述带参系统的迁移；

④ I 是初始状态集。

因此，要准确对一个能够用 Murphi 或者 SMV 描述的带参系统进行建模，就必须对这些基本概念有足够强的表达能力。

除此之外，因为我们考虑的是带参系统，因此还要集中精力支持其带参特性。

2.2 语义模型

为了实现从其 Murphi 模型自动转换得到其他语言版本模型的目标，我们设计了一个语义模型来从语义的角度全面描述带参系统。这个内部语义模型的定义如图 1 所示。

```
type const = Intc of int | Strc of string | Boolc of bool
type typedef = Enum of string * const list
type paramdef = Paramdef of string * string
type paramref = Paramref of string
   | Paramfix of string * string * const
type vardef = Arrdef of (string * paramdef list) list * string
type var = Arr of (string * paramref list) list
type exp = Const of const | Var of var | Param of paramref
   | Ite of formula * exp * exp
and formula = Chaos | Miracle | Eqn of exp * exp
   | Neg of formula | AndList of formula list
   | OrList of formula list| Imply of formula * formula
   | ForallFormula of paramdef list * formula
   | ExistFormula of paramdef list * formula
type statement = Assign of var * exp
   | Parallel of statement list
   | IfStatement of formula * statement
   | IfelseStatement of formula * statement * statement
   | ForStatement of statement * paramdef list
type prop =
   Prop of string * paramdef list * formula
type rule =
   Rule of string * paramdef list * formula * statement
type protocol = {
   name: string; types: typedef list; vardefs: vardef list;
   init: statement; rules: rule list; properties: prop list;
}
```

图 1. 语义模型的定义

2.2.1 语义模型的基本组件

基本组件包括常量、自定义类型、参数定义、参数引用、变量定义和变量引用。

① 常量 const，描述三种基本数据类型，分别是整形常量 Intc，字符串常量 Strc 和布尔常量 Boolc；

② 自定义类型 typedef，类似于高级编程语言中的枚举，用来描述状态变量的取值空间；

③ 参数定义 paramdef，用来支持参数化特性，是一个二元组<参数名称, 参数类型名称>；

④ 参数引用 paramref，用来支持参数化特性，有非实例化和实例化两种形式，前者仅需要参数名称，后者是一个三元组<参数名称, 参数类型名称, 值>；

⑤ 变量定义 vardef，用来描述状态变量，支持结构化的定义，是一个二元组<结构化变量名称, 变量类型名称>，其中结构化变量名称是一个列表，其每个元素都是一个二元组<部分变量名称, 参数定义>，这样的设计可以支持自定义数据结构，实现对类似于 state.vars[i].value 这样的变量的定义；

⑥ 变量引用，与变量定义类似，列表二元组变成<部分变量名称, 参数引用>。

2.2.2 语义模型的高级组件

高级组件刻画更组粒度的表达形式，包括表达式、谓词公式、语句、性质和规则。

① 表达式 exp，有四种形式，分别为常量表达式 Const，变量表达式 Var，参数表达式 Param 和条件表达式 Ite 即三元组<条件, 满足条件时的值, 不满足条件时的值>；

② 谓词公式 formula，包括多种形式，分别为真值 Chaos，假值 Miracle，相等谓词 Eqn 即使用二元组<值 1, 值 2>来描述相等关系，否定谓词 Neg，合取谓词 AndList，析取谓词 OrList，蕴含谓词 Imply 即使用二元组<值 1, 值 2>来描述前者与后者之间的蕴含关系，以及具有参数化形式的量词谓词 ForallFormula 和 ExistFormula；

③ 性质 prop，是一个三元组<名称, 参数定义, 谓词公式>；

④ 规则 rule，是一个四元组<名称, 参数定义, 迁移条件, 赋值动作序列>。

需要强调的是，这里的表达式组件 exp 和谓词公式组件 formula 是互相递归定义的，因为 exp 中的条件表达式形式需要一个谓词公式作为条件，而 formula 显然需要 exp 作为其基本组成部分来表示一个值。

2.2.3 语义模型的顶级组件

语义模型的顶级组件用来描述整个带参系统，是一个六元组 $S = <N, T, V, I, R, P>$，其中：

① N 即 *name* 是一个字符串，代表系统名称；

② T 即 *types* 是自定义类型集合；

③ V 即 *vardefs* 是一个变量集合，用于描述带参系统所可能出现的每个状态，也决定状态空间；

④ I 即 *init* 是初始状态集合；

⑤ R 即 *rules* 是规则集合，用于描述带参系统各状态间的迁移关系；

⑥ P 即 *properties* 是安全性质集合，描述带参系统所必须满足的不变式。

3 转换策略和自动建模

自动化建模的一般过程如图 2 所示。即先用一个编译器将 Murphi 模型提升到内部语义模型，然后再通过相应的转换策略得到 NuSMV 模型。

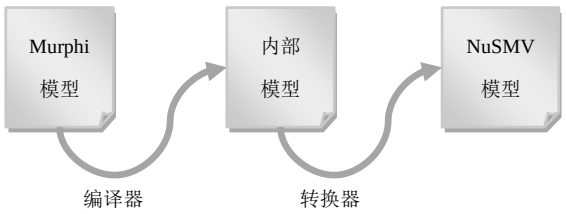

图 2. 自动化建模的一般过程

3.1 从 Murphi 模型到语义模型

因为 Murphi 模型和内部语义模型都为更好地描述带参系统而特别支持了带参特性，因此从 Murphi 模型转换到语义模型的过程是相当直观的，只需要使用常规的编译技术即可，主要需要注意对定义和引用的处理、对非参量的处理和对自定义数据结构的处理。

对定义和引用的处理会出现在两个方面，一个是变量，一个是参数，但是处理方法是类似的。因为定义和引用一般会在语法树的不同位置出现，因此只需要区别处理，也就是如果是在定义的语法位置出现则转换为定义，否则就是引用。

一般地，在 Murphi 中，刻画带参系统的参量会出现于规则、不变式、forall/exists 量词和 for 语句中，但是偶尔这里也会出现实际上没有刻画带参特性但是具有参数形式的量。这时只需要先将其按照正常的参量做处理，然后再将非参数类型展开，最后由于展开后依然保持参量形式，因此还需要将其转化为普通的常量。

处理自定义数据结构有两种方案：一种是在语法分析阶段就将复合数据结构展开为多个变量，在转换到内部模型中；另一种是先保留复合数据结构的形式，转换完毕后再在内部模型中将复合数据结构展开。两种方案的复杂度相近，但由于在实现上，我们使用函数式语言处理内部模型，进行复合变量展开有一定优势，因此采用第二种方案。

3.2 从语义模型到 NuSMV 模型

因为内部的语义模型是从语义角度描述带参系统，因此实际上它可以转换到任何具有相同语义的语言的模型，当然也包括 NuSMV 语言。

3.2.1 处理分支语句

与 Murphi 不同，NuSMV 语言并不支持分支语句，但是支持条件表达式。两者关系如图 3 所示。

Murphi	NuSMV

if condition1 then
  statements1
elif condition then
  statements2
else
  statements3
endif
	case
	condition1: value1;
	contidion2: value2;
	TRUE: value3;
	esac

图 3. Murphi 和 NuSMV 对分支结构的支持

显然 NuSMV 对分支结构的支持是表达式粒度的，而内部的语义模型与 Murphi 一致，因此需要先对分支语句进行预处理，使得一个分支语句内仅包含对一个变量的赋值，从而可以直接将其转换为 NuSMV 中的分支表达式。

3.2.2 处理规则

NuSMV 中不能直接支持规则，但是可以使用 process 变量结合模块来模拟规则，问题在于依然不支持规则的前置条件，同时也需要事先提供规则内会用到的所有变量。第一个问题比较简单，可以使用 NuSMV 的条件表达式来模拟前置条件。这里重点解决第二个问题，即提取规则内会用到的所有的变量。

从最细的粒度观察，可以发现变量都是存在于表达式中的，而表达式的存在形式中，常量和参数引用都不是变量，变量表达式可以直接提取出一个变量，条件表达式可以递归地从其条件和两个值中分别提取出变量并给出并集。更粗粒度的结构只需要将其拆分为细粒度的结构，直到拆分到表达式级别上进行提取，这一过程可以使用递归方便地实现。

3.2.3 其他

NuSMV 的变量可以包含部分特殊符号，包括半角的中括号和点号，因此可以方便地表示展开后的自定义数据结构。

另外，NuSMV 本身是不支持带参的，因此在转换之前需要实例化参数。因为 NuSMV 是自动生成的，因此只需要修改对应的 Murphi 模型，就可以方便地将维护工作同步到 NuSMV 模型中，从而也使 NuSMV 的建模间接参数化。

3.3 赋值语句顺序化

在 Murphi 中，赋值语句块是顺序执行的，而在 NuSMV 中，同一个赋值块中的语句则是并发执行的。两者的区别可以用图 4 来说明，假设 *x* 和 *y* 的初值分别为 1 和 2。

Murphi	NuSMV
x := 3; y := x;	next(x) := 3; next(y) := x;

图 4. Murphi 和 NuSMV 中语句块的区别

这两个语句块有类似的形式，但是却有完全不同的表现。Murphi 的语句块执行后，*x* 和 *y* 的值都是 3，但 NuSMV 的语句块执行后，*x* 的值为 3，*y* 的值却为 1，原因正是 NuSMV 的语句块是并发执行的，也就是 *y* 的新值为 *x* 更新之前的值。解决这个问题的方法是在内部模型建立后，对赋值语句块进行替换，使其与 Murphi 的语义一致。如图 4 所示的例子中，*y* := *x* 将会被替换为 *y* := 3。

4 实验结果及分析

如表 1 所示，我们对一系列的带参系统进行了实验，通过其 Murphi 模型自动建立其对应的 NuSMV 模型，且得到的 NuSMV 模型能够满足一般科研需要。

表 1. 一些带参系统的实验结果

	Murphi 模型大小/KB	数据类型数目	规则数目	NuSMV 模型大小/KB
MutualEx	0.6	2	4	2.5
MESI	0.9	2	4	5.2
MOESI	1.0	2	5	4.9
Germanish	2.1	3	6	14.6
German	4.1	6	13	26.1
Flash	18.6	19	62	546.2
down	0.7	4	1	2.2

我们已经把表 1 中所有实例的 Murphi 模型和自动生成的 NuSMV 的模型放在网上[7]。最后一个实例是具有顺序花赋值语句的，可以验证转换过程可以解决赋值语句顺序化的问题；其他的实例都是带参缓存一致性协议。

进行分析不难发现， NuSMV 模型的规模与原始 Murphi 模型的规模、自定义数据类型的多少和规则数目都是正相关的，表明随着带参系统复杂度的提高，NuSMV 模型的规模也急剧增大；另外，NuSMV 模型规模也都显著大于 Murphi 模型，对这样的模型手工进行维护，几乎是不可能的，但是这里可以通过维护其对应的 Murphi 模型后进行转换，间接维护 NuSMV 模型，也使得 NuSMV 模型有了间接参数化的能力。

另外，我们还将 Murphi 语言描述的谓词公式转换为了 Z3 工具[8]中的 SMT2 语言描述的谓词公式，以进行可满足性检查。这些工作大大减少了在带参缓存一致性协议的形式化验证工作[9]中需要的手动工作量，也是本文的一个重要应用。

5 结论

对带参系统从多个角度使用不同的工具进行建模，具有重要的实践意义。但由于带参系统的复杂性，直接手工完成这些工作存在极大的困难。我们提出了一个能够全面描述带参系统特性的语义模型，实现了从 NuSMV 建模自动化。我们的工作实现了以下突破：

① 对复杂带参系统的多工具方便建模和维护；

② 使不支持参数化的 NuSMV 工具间接参数化；

③ 内部语义模型从语义角度描述带参系统，可以使用特定转换策略转换为用更多工具描述的模型。

参考文献

1 Dill DL. The Murphi verification system. Computer Aided Verification. Springer Berlin Heidelberg, 1996: 390–393.

2 Cimatti A., Clarke E, Giunchiglia F. NuSMV: a new symbolic model verifier. Computer Aided Verification, 1999: 495–499.

3 Holzmann GJ. The model checker SPIN. IEEE Transactions on software engineering. 1997 May 1;23(5):279.

4 Bengtsson J, Larsen K, Larsson F, Pettersson P, Yi W. UPPAAL—a tool suite for automatic verification of real-time systems. Springer Berlin Heidelberg; 1996.

5 曹燊, 李勇坚. 基于不变量查找的 German 协议验证. 计算机系统应用, 2015, 24(11): 173-8.

6 张文辉. 软件系统行为与程序正确性. http://lcs.ios.ac.cn/~zwh/pv/pv13.pdf

7 https://github.com/lyj238/murphismv

8 De Moura, Leonardo, and Nikolaj Bjørner. Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2008: 337-340.

9 Li Y, Pang J, Lv Y, Fan D, Cao S, Duan K. ParaVerifier: An Automatic Framework for Proving Parameterized Cache Coherence Protocols. In Automated Technology for Verification and Analysis. Springer International Publishing. 2015 Oct 12:207-213.