基于不完全 Kripke 结构三值逻辑的模型检验*)

郭 建[1] 韩俊刚[2]

(西安电子科技大学 西安 710071)[1] (西安邮电学院 西安 710061)[2]

摘 要 模型检验技术是形式化验证中比较成熟的技术,但随着设计系统规模的增加,状态爆炸已成为其发展的一个主要问题。为解决此问题,本文提出对系统进行抽象,建立不完全的状态模型,在此状态模型上来验证表示其属性的逻辑公式。这样一个逻辑公式的真值除了真、假外,还出现了第三种情况:未知,即在这个状态模型下无法确定其真值,需要更多的状态信息才能确定。本文还讨论了二值逻辑的模型检验技术,在此基础上给出了基于不完全状态空间的三值逻辑的模型检验算法,此算法与二值逻辑模型检验算法相比,没有带来时间复杂度的增加,最后给出了三值逻辑模型检验算法的应用。

关键词 三值逻辑,模型检验,不完全 Kripke 结构

Model Checking Using Partial Kripke Structure with 3-Valued Temporal Logic

GUO Jian[1] HAN Jun-Gang[2]

(Xidian University, Xi'an 710071)[1] (Xi'an Institute of Post and Telecommunications, Xi'an 710071)[2]

Abstract Model checking is one of the attracting methods in formal verification, but the main disadvantage of model checking is the state explosion that might occur if the system being verified becomes larger. This paper presents a method of abstracting a system and setting up an incomplete state model, Partial Kripke Structure, upon which properties of the system is verified. Under such a structure, a 3-valued interpretation to CTL logic formulae is given, with a third truth value unknown($\perp$), which means " true or false unknown". A 3-valued CTL model checking algorithm based on partial Kripke structure is also provided. Compared with 2-valued model checking technique, the algorithm does not increase the time complexity. At last some applications of 3-valued logic model checking are addressed.

Keywords 3-valued logic, Model checking, Partial kripke structure

1 引言

模型检验是对有限状态系统进行形式化验证的一种有效的方法。但随着软硬件的发展,需要验证的系统规模越来越大,状态爆炸问题就成为其发展的瓶颈。对于这个问题的解决可以从两个方面考虑:一是对模型检验技术的改进,例如已经由状态的显式表示变成利用 BDD 对状态的隐式表示;二是抽象,根据并非状态模型中的所有状态都与所验证的属性有关这个特点,可以抽取有关的状态建立抽象的状态模型。经过抽象后,虽然状态数目大为减少,但会使系统中的某些属性变得不确定了,在一个状态下无法确定某个属性的真假,出现了第三种情况:未知。这样,抽象后的状态模型就无法用现有的模型检验工具进行验证。本文针对此情况,研究了三值逻辑(真、假、未知)的模型检验技术,并在原有技术的基础之上给出了三值逻辑的模型检验算法,此算法与原有的二值逻辑模型检验技术相比,没有增加其算法的复杂度,最后给出了三值逻辑模型检验技术的应用。

2 二值逻辑的模型检验

模型检验的第一步就是给出所要验证的系统应具有的属性,这里的属性是系统的关键属性,一旦了解了验证的属性,第二步的工作就是对系统进行形式化建模。为了能够验证系统,这个模型必须能够捕获认为是正确的属性;另外,该模型应舍弃对系统的细节描述,并且这样并不影响所要验证的属性,否则就会使系统变得非常复杂,不便于验证。例如当建模一个数字系统时,根据门和布尔值建模就可以了,而不需实际的物理电路。

对系统的建模,在模型检验中用 Kripke 结构来实现的。

2.1 Kripke 结构

一个 Kripke 结构是由状态集、状态之间的转移关系、每个状态上使一组原子命题为真的集合组成。它的定义如下:

定义 1[1] 设 AP 是一组原子命题,在 AP 上的一个 Kripke 结构 M 定义为四元组 $M=(S,S_0,R,L)$,其中,

1)S 是一个有限状态集合;

2)$S_0\subseteq S$ 是初始状态集合;

3)$R\subseteq S\times S$ 是转移关系,要求是完全的,即对 $\forall s\in S$ 都存在一个状态 $s'\in S$,使得 $(s,s')\in R$ 成立;

4)$L:S\to 2^{AP}$ 是一个函数,标记每个状态使某些原子命题为真的函数。

有时在不关心初始状态集的情况下,Kripke 结构的定义中就可以忽略初始状态集合的定义。

在 Kripke 结构中,从一个状态 s 开始的一条路径是一个无限状态序列 $\pi=s_0s_1\cdots$,其中 $s_0=s$,且 $R(s_i,s_{i+1})$ 对所有 $i\geq 0$ 成立。

2.2 CTL 公式

前面给出了对系统的描述,可利用 Kripke 结构来建立模

*)基金项目:国家自然科学基金(90207015)。郭 建 讲师,博士研究生,主要研究领域为模型检验、硬件的形式化验证。韩俊刚 教授,博士生导师,主要研究领域为 ASIC 设计和形式化验证。

型，这里将给出对属性的描述。在 Kripke 结构中已经给出一组原子命题，以及每个状态上有哪些原子命题为真，我们利用布尔联结词“与、或、非”可以构造更为复杂的表达式来描述属性。另外，在交互式系统中，我们还关心如何来描述状态间的转移，而时态逻辑是一种能够描述状态间转移序列的逻辑。利用时态逻辑、逻辑联结词以及路径量词构成了 CTL 公式，可很好地描述系统的一些关键属性。CTL 的定义如下：

定义 2[1] 一个 CTL 公式递归定义为：

1)每个原子命题是一个 CTL；

2)若 f,g 是 CTL 公式，则在 $\neg f$、$f\wedge g$、$f\vee g$ 是 CTL 公式；

3)若 f,g 是 CTL 公式，则在 Xf、Ff、Gf、$f\cup g$ 前加上路径量词后也是 CTL 公式；

4)当且仅当有限次应用 1)2)3)所得到的符号串是 CTL 公式。

例如 $EF(f\wedge\neg g)$，$AG(\neg f\vee AFg)$都是 CTL 公式，第一个的含义是存在一条路径，在这条路径上有一个状态使 f 满足，而 g 不满足；第二个公式的含义是在所有路径上的所有状态，一旦 f 满足，则 g 最终会满足。

2.3 模型检验

模型检验是对一个给定的有限 Kripke 结构 M 和一个 CTL 公式 p，检查该结构 M 是否为 p 的一个模型，即检查是否存在一个子集 $S'\subseteq S$，对于 $\forall s'\in S'$都满足 p，并使得所有的初始状态都在这个状态集 S'中。如果是，则说明 M 是 p 的一个模型，否则该结构 M 不是 p 的一个模型，这时可以给出一个反例来说明。

3 不完全 Kripke 结构

3.1 不完全 Kripke 结构(Partial Kripke structure PKS)[2]

定义 3 不完全 Kripke 结构 M 是一个五元组(S, S_0, P, R, L)，其中：S：状态集合；S_0：初始状态集合；P：具有三值(真、假、未知)的原子命题集合；R：转移关系，$R\subseteq S\times S$；L：标签函数，$S\times P\to\{\text{true, false},\perp\}$，在 S 中的每个状态，每个原子命题所具有的真值(true, false, $\perp$)。$\perp$表示未知。

从这个定义可以看出，一个标准的 Kripke 结构是不完全 Kripke 结构的特殊情况。当原子命题只能取真和假时，不完全 Kripke 结构就变成了标准的 Kripke 结构。在标准的 Kripke 结构中，原子命题不能取“$\perp$”即“未知”，所以有时又把标准的 Kripke 结构称为完全的 Kripke 结构。

3.2 操作符的解释

在三值逻辑中，原子命题被定义成真、假和未知，那么逻辑运算符“$\wedge$、$\vee$、$\neg$”的解释就有所不同。在此，应用 Kleene[3]的最强正则三值命题逻辑。在这种逻辑中，“$\perp$”被解释成“未知是真还是假”。对“$\wedge$、$\neg$”的解释如下表所示。而对“$\vee$”的解释通过德·摩根定律$(p1\vee p2)=\neg(\neg p1\wedge\neg p2))$来得出，其真值表如下：

表 1 Kleene 的最强正则三值命题逻辑“$\wedge$、$\vee$、$\neg$”的真值表

p	F	T	$\perp$	$\wedge$	F	T	$\perp$	$\vee$	F	T	$\perp$
$\neg$p	T	F	$\perp$	F	F	F	F	F	F	T	$\perp$
				T	F	T	$\perp$	T	T	T	T
				$\perp$	F	$\perp$	$\perp$	$\perp$	$\perp$	T	$\perp$

3.3 三值逻辑 CTL 公式

时态逻辑可以用来描述软/硬件的属性。从前面已经看到，CTL 公式是在命题逻辑的基础上加一些时态算子和路径量词来构成的，在此，为了简化，首先考虑在三值逻辑下部分 CTL 公式，其它公式都可以用类似的算法或更为简单的算法来处理。

定义 4 设 P 是非空有限的原子命题公式集合，在不完全 Kripke 结构下，一个 CTL 公式可以用下列的抽象语法表示，其中 $p\in P$：

$\phi::=p|\neg\phi|\phi_1\wedge\phi_2|AX\phi|EX\phi|E(\phi_1\cup\phi_2)|A(\phi_1\cup\phi_2)$。

任何一个 CTL 公式在不完全 Kripke 结构中的语义定义如下：

定义 5 设一个命题模态逻辑公式 ϕ 在不完全 Kripke 结构 $M=(S, P, R, L)$中的一个状态 s 下的真值，记做 $[(M,s)\vdash\phi]$，归纳定义为：

(1)$[(M,s)\vdash p]=L(s,p)$；

(2)$[(M,s)\vdash\neg\phi]=comp([(M,s)\vdash\phi])$；

(3)$[(M,s)\vdash\phi_1\wedge\phi_2]=[(M,s)\vdash\phi_1]\wedge[(M,s)\vdash\phi_2]$；

(4)$[(M,s)\vdash AX\phi]=\bigwedge_{s\to s'}[(M,s')\vdash\phi]$。

(5)$[(M,s)\vdash EX\phi]=\bigvee_{s\to s'}[(M,s')\vdash\phi]$。

(6)$[(M,s)\vdash E(\phi_1\cup\phi_2)]=$T 当存在一条路径 $s_0s_1\cdots$和某个 i，使得$[(M,s_i)\vdash\phi_2]=$T，且任意 $j<i$，都有$[(M,s_j)\vdash\phi_1]=$T。

$[(M,s)\vdash E(\phi_1\cup\phi_2)]=$ F 当对任意一条路径 $s_0s_1\cdots$和某个 i，或使得$[(M,s_i)\vdash\phi_2]=$F，或使得$[(M,s_i)\vdash\phi_2]=$T，但在此路径上必存在一个 $j<i$，使$[(M,s_j)\vdash\phi_1]=$F

$[(M,s)\vdash E(\phi_1\cup\phi_2)]=\perp$ 其它。

(7)$[(M,s)\vdash A(\phi_1\cup\phi_2)]=$ T 当任意一条路径 $s_0s_1\cdots$和某个 i，使得$[(M,s_i)\vdash\phi_2]=$T，且任意 $j<i$，都有$[(M,s_j)\vdash\phi_1]=$T。

$[(M,s)\vdash A(\phi_1\cup\phi_2)]=$ F 当存在一条路径 $s_0s_1\cdots$和某个 i，或使得$[(M,s_i)\vdash\phi_2]=$F，或使得$[(M,s_i)\vdash\phi_2]=$T，但在此路径上必存在一个 $j<i$，使$[(M,s_j)\vdash\phi_1]=$F

$[(M,s)\vdash A(\phi_1\cup\phi_2)]=\perp$ 其它。

4 三值命题模态逻辑公式的模型检验

4.1 三值逻辑公式的模型检验算法

根据 AX 和 EX 的语义，可以证明下面的公式：

$AX\phi\equiv\neg EX\neg\phi$

那么上面的逻辑公式就可递归定义为：

$\phi::=p|\neg\phi|\phi_1\wedge\phi_2|EX\phi|E(\phi_1\cup\phi_2)|A(\phi_1\cup\phi_2)$。

对于任何一个 CTL 逻辑公式 ϕ 只能是下列六种形式之一：$p(p\in P)$，$\neg\phi$，$\phi_1\wedge\phi_2$，$EX\phi$，$E(\phi_1\cup\phi_2)$和 $A(\phi_1\cup\phi_2)$。在进行模型检验时，只要对这六种形式给出相应的算法即可。此算法是在 Clark 的模型检验算法[10]的基础之上进行改进而形成的。对于每一个状态，都设置两个集合：一个是在此状态下使公式为真的公式集合 $label()$，一个是此状态下使公式为未知的公式集合 $label_\perp()$。

1)$p(p\in P)$。把在状态 s 下原子命题为真的原子命题归入 $label(s)$，以及此状态 s 下原子命题为未知的原子命题归入 $label_\perp(s)$；

2)$\neg\phi$。根据 Kleene 的最强正则三值命题逻辑可知：把 $\neg\phi$ 归入状态$\{s|\phi\notin label(s) and\phi\notin label_\perp(\phi)\}$的 $label()$，并且把$\neg\phi$ 归入状态$\{s|\phi\in label_\perp()\}$的 $label_\perp()$；

3)$\phi_1 \wedge \phi_2$。对任意一个状态 s,若 $\phi_1 \in label(s)$且 $\phi_2 \in label(s)$,则把公式 $\phi_1 \wedge \phi_2$ 归入到 $label(s)$中;否则若 $\phi_1 \in label(s)$或 $\phi_1 \in label_\perp(s)$且 $\phi_2 \in label(s)$或 $\phi_2 \in label_\perp(s)$,则把公式 $\phi_1 \wedge \phi_2$ 归入到 $label_\perp(s)$中。

4)$EX\phi$。对任意一个状态 s,若存在 s',$(s,s') \in R$ 且 $\phi \in label(s')$,则把公式 $EX\phi$ 归入 $label(s)$;否则若存在 s',$(s,s') \in R$ 且 $\phi \in label_\perp(s')$,则把公式 $EX\phi$ 归入 $label_\perp(s)$。

5)$E(\phi_1 \cup \phi_2)$。若公式 $E(\phi_1 \cup \phi_2)$在不完全 *Kripke* 结构的某个状态 s 下为真,则存在一条从 s 开始的路径,使得在这条路径上的某个状态下 ϕ_2 为真,且在这个状态前的所有状态都 ϕ_1 为真。

此算法是求出满足 $E(\phi_1 \cup \phi_2)$的所有的状态,第一步求出使 ϕ_2 为真的所有状态集合 T 和使 ϕ_2 为未知的状态集合 $T_\perp$,然后分别把公式 $E(\phi_1 \cup \phi_2)$加入到它们的集合 $label()$和 $label_\perp()$中;第二步对 T 中的所有状态进行搜索,检查 T 中的每一个直接前驱状态 s 是否使公式 ϕ_1 为真,若为真,则把公式 $E(\phi_1 \cup \phi_2)$加入到 $label(s)$中,并把 s 归入到集合 T 中;若 ϕ_1 在状态 ϕ_1 下为未知,则把公式 $E(\phi_1 \cup \phi_2)$加入到 $label_\perp(s)$中,并把 s 归入到集合 $T_\perp$ 中。当 T 中的所有状态都检查完成后,就可找到满足 $E(\phi_1 \cup \phi_2)$的所有的状态。第三步找出使公式为未知的那些状态,方法类似于第二步,只是在此是对 $T_\perp$ 中的所有状态进行搜索。

具体的算法如下:

```
void checkeu(φ1,φ2)
{T={s|φ2∈label(s)}; T⊥={s|φ2∈label⊥(s)};
  while(all s∈T) label(s)=label(s)∪{ E[φ1Uφ2]};
  while(T≠∅)
    {choose s∈T;T=T\{s};
    while(all t and R(t,s))
    if(E[φ1Uφ2]∉label(t)and φ1∈label(t))
      { label(t)=label(t)∪{ E[φ1Uφ2]};
        T=T∪{t};}
      else if((φ1 ∈ label⊥(t) and E[φ1Uφ2]∉label(t) and E
[φ1Uφ2]∉label⊥(t))
          {label⊥(t)=label⊥(t)∪{ E[φ1Uφ2]};
            T⊥=T⊥∪{t};}
  }
while(T⊥≠∅)
    { choose s∈T⊥;T⊥=T⊥\{s};
    while(all t and R(t,s))
    if((φ1∈label⊥(t)or φ1∈label(t))and E[φ1Uφ2]∉label(t)and
E[φ1Uφ2]∉label⊥(t))
      { label⊥(t)=label⊥(t)∪{ E[φ1Uφ2]};
        T⊥=T⊥∪{t};}
    }
}
```

6)$A(\phi_1 \cup \phi_2)$。若公式 $A(\phi_1 \cup \phi_2)$在不完全 Kripke 结构的状态 s 下为真,则对所有从 s 开始的路径,都存在一个状态使 ϕ_2 为真,并且在这个状态前的所有状态都使 ϕ_1 为真;若在 s 状态下公式 $A(\phi_1 \cup \phi_2)$的真值为未知,则在从 s 开始的所有路径上都存在一个状态使 ϕ_2 为真或未知,并且在这个状态前的所有状态都 ϕ_1 使为真或未知;对于其他情况,则公式 $A(\phi_1 \cup \phi_2)$在状态 s 下为假。

同前面的算法类似,在此为每个状态设置两个集合 $label()$和 $label_\perp()$分别表示在此状态下公式为真和未知。并在下面的算法中设置了一个全局标志变量 $flag$ 标志在某个状态下使公式 $A(\phi_1 \cup \phi_2)$为未知,对公式 $f=A(\phi_1 \cup \phi_2)$在状态 s 下的检查分多种情况:

a. 若 $\phi_2 \in label(s)$,则把 f 加入到 $label(s)$中,返回;

b. 若 $\phi_2 \in label_\perp(s)$,则把 f 加入到 $label\perp(s)$中,返回;

c. 若 $\phi_1 \notin label(s)$,则 $b=false$,返回;

d. 若 $\phi_1 \notin label_\perp(s)$,则 $b=\perp$,到 g.

e. 对 $\phi_1 \in label(s)$和 $\phi_1 \in label\perp(s)$的,做 f.

f. 把 s 压入堆栈,对 s 的所有直接后继做上述步骤。

在此递归算法中,一旦 b 为假,就不再对 s 的其它直接后继做检查,而直接返回,表明在 s 状态下公式 $A(\phi_1 \cup \phi_2)$的真值为假;若 $\phi_1 \in label_\perp(s)$,则设置 $flag=$true,继续检查,直到对 s 的所有直接后继都检查完,这时可根据 $flag$ 判断 f 在此状态下的真值,若 $flag=$true,则表明在检查的过程中有使公式 f 为未知的状态,那么在状态 s 下公式为未知,否则在状态 s 下公式为真。具体算法如下:

```
flag=false;
/*设置一个全局变量,来判断在检查过程中是否存在有些状态使公
式为未知。*/
checkau(f,s,b)
/* f=A(φ1∪φ2), s 为一状态,b 为一标志,b=false,则 f∉label
(s);b=⊥,表明目前还无法判断其真值;b=true, 则 f∈label(s)或 f
∈label⊥(s). */
{ if arg2(f)∈label(s)
    {label(s)=label(s)∪{f}; b=true; return;}
  else if arg2(f)∈label⊥(s)
      { label⊥(s)=label⊥(s)∪{f}; b=true; return;}
    else if arg1(f)∉label(s)
        {b=false; return;}
        else if arg1(f)∉label⊥(s)
            {b=false; return;}
          else
            flag=true;
  /* arg2(f)表示去 f 的第二个参数,即 φ2,arg1(f)表示去 f 的第
二个参数,即 φ1。*/
push(s,ST);
  /*设置一个堆栈来存放还为确定公式 f 在此状态的真值。*/
for all s1∈sucessor(s) do /* successor(s)为所有 s 的直接后继*/
  { au(f,s1,b1);
    if b1==false
      {pop(ST);b=false; return;}
  }
pop(ST);
if falg==true
  {label⊥(s)=label⊥(s)∪{f}; b=true; return;}
else
  {label(s)=label(s)∪{f}; b=true; return;}
}
```

在未知的情况下,可以通过一个数组记录对此公式检查过程中有哪些状态使公式为未知,以便了解模型应该在哪些方面扩充。一般来说,最先进入数组的那些状态,是使公式为未知的最底层的状态,把这些状态扩充后,就可判断公式的真假。

4.2 例子

下面给出手工验证一个 CTL 逻辑公式的例子。设一个不完全 Kripke 结构 $M=(S, P, R, L)$

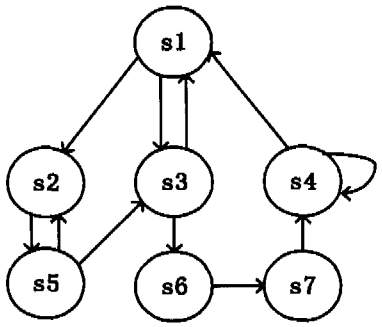


图 1 不完全 Kripke 结构 M 的状态图

如图所示,其初始状态集合为$\{s_1\}$,其中,

$L(s_1)=\{p, q, r, s\}$

$L(s_2)=\{p, q, \perp, \neg s\}$

$L(s_3)=\{\perp, \neg q, \neg r, s\}$

$L(s_4)=\{\perp, \perp, r, \neg s\}$

$L(s_5)=\{\neg p, q, r, \perp\}$

$L(s_6)=\{p, q, r, \neg s\}$

$L(s_7)=\{p,\perp,\perp,\neg s\}$

验证公式：$\neg(p\wedge q)\wedge EX(r\wedge s)$。这里用 $S(f)$ 表示满足公式 f 的状态集合。

$label(p)=\{s_1, s_2, s_6, s_7\}$

$label_\perp(p)=\{s_3, s_4\}$

$label(q)=\{s_1, s_2, s_5, s_6\}$

$label_\perp(q)=\{s_4, s_7\}$

$label(r)=\{s_1, s_4, s_5, s_6\}$

$label_\perp(r)=\{s_2, s_7\}$

$label(s)=\{s_1, s_3\}$

$label_\perp(s)=\{s_5\}$

$label(p\wedge q)=\{s_1, s_2, s_6\}$

$label_\perp(p\wedge q)=\{s_4, s_7\}$

$label(\neg(p\wedge q))=\{s_3, s_5\}$

$label_\perp(\neg(p\wedge q))=\{s_4, s_7\}$

$label(r\wedge s)=\{s_1\}$

$label_\perp(r\wedge s)=\{s_5\}$

$label(EX(r\wedge s))=\{s_3, s_4\}$

$label_\perp(EX(r\wedge s))=\{s_2\}$

$label(\neg(p\wedge q)\wedge EX(r\wedge s))=\{s_3\}$

$label_\perp(\neg(p\wedge q)\wedge EX(r\wedge s))=\{s_4\}$

由于$\{s_1\}$不包含在 $label(\neg(p\wedge q)\wedge EX(r\wedge s))\cup label_\perp(\neg(p\wedge q)\wedge EX(r\wedge s))$，所以此公式在 M 结构中的真值为假。

4.3 模型检验的复杂度分析

三值模型检验是在 Clarke 的二值模型检验算法的基础上修改而得到的。在 Clarke 的算法中是找出那些满足 CTL 公式 f 的状态，为此设置一个 *label* 函数，对 CTL 公式从里到外搜索，找到满足 f 的子公式的状态，直到搜索到 f，这样，就找到了满足 f 的状态。在我们的算法中，也是利用此方法，只是此时设置两个函数 *label* 和 $label_\perp$，*label* 用于搜索满足子公式的状态，而 $label_\perp$ 用作搜索使子公式为未知(⊥)的状态，此时，函数 *label* 和 $label_\perp$ 同时进行，在计算子公式为真的过程中，计算子公式为“⊥”的情况，由于 Clarke 的二值模型检验算法的时间复杂度是 $O((|S|+|R|)\cdot|f|)$，因此：

定理 1 对 CTL 逻辑公式进行三值模型检验的时间复杂度与 Clarke 的二值模型检验算法的时间复杂度一样，是 $O((|S|+|R|)\cdot|f|)$。

证明略。

5 应用

5.1 在硬件验证中的应用

模型检验技术对硬件设计进行验证是比较成熟的一种形式化方法，但是对硬件进行模型验证时，都限定了变量的取值是 0 或 1 两种情况。在运用硬件描述语言设计时，无论是 VHDL 还是 Verilog，对于变量的取值还有第三种情况，即不关心(can't care)，也就是该变量的取值不影响整个设计，它既可以取 0，也可以取 1。对于这种情况的设计进行模型检验时，常常把它变成两种情况：为 1 时，为 0 时。这样就增加了状态数目，而状态数目的增加是影响模型检验能否进行的一个重要因素。

三值模型检验技术就可以直接处理不关心(can't care)，把这种情况设为“⊥”，然后进行模型检验，若对规格描述进行模型检验时，若得出的结论是假，表明该系统不满足此规格描述；若得出的结论是真，表明该系统满足此规格描述；若得出的结论是“⊥”，表明该系统无法确定此规格描述，说明此规格描述与某些 can't care 变量的“⊥”值有关，那么进一步说明在此设计中，这些变量的取值影响整个设计，它们的某些取值使此规格描述为真，而某些取值使它为假，此时，我们就可以应用可满足性技术找出一组使它为假的取值。这样就可以对具有不关心(can't care)情况的设计进行模型检验。

5.2 在 SOC 验证中的应用

目前片上系统(System On a Chip，SOC)设计已成为集成电路设计中的热门领域。随着系统规模的扩大和复杂性的增加，验证问题成为 SOC 设计中最大的挑战。而对于 SOC 的验证可分为对 SOC 总线系统的验证和对 IP 核自身的验证以及整体的验证三个部分。

基于 IP 核的系统设计可以看成各种 IP 核通过总线把它们相互连接起来。由于 IP 核来自不同的厂商，那么这就需要有一个标准总线连接它们。另外，我们还需要一些接口逻辑，称为 Glue，以便把 IP 核与标准总线连接在一起。在一些情况下，若 IP 核符合这个标准总线的要求，就可以直接把它连接到总线上，而不需要 Glue。在一个分层模型中，桥(Bridge)是用来扩展前面的系统，它是总线之间的连接。

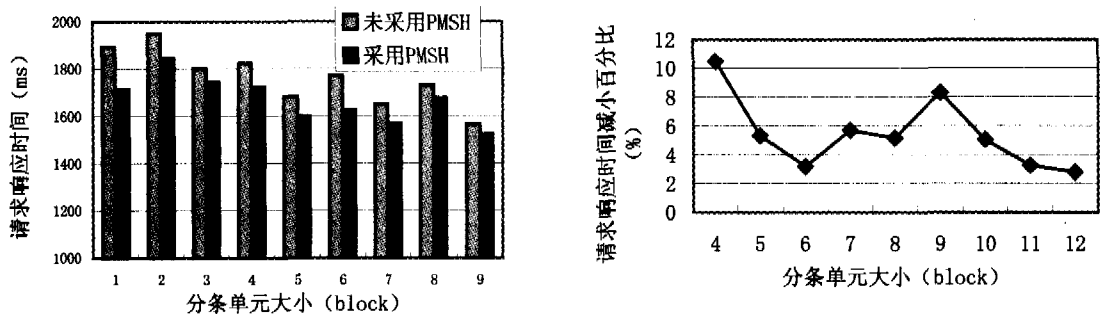
在验证了 IP 核与总线之间的连接电路 Glue 后，以及对总线与总线之间的 Brigde 验证之后，就需要验证相互连接的总线及总线与 IP 核之间的电路，即要对整个设计连接后进行验证。这时就不需要考虑细节问题，抽象出与整个设计有关的状态即可，在抽象的过程中会涉及到一些可能的转移，这时就出现了三值逻辑，那么此时可以用不完全 Kripke 结构来描述整个设计，而用三值逻辑公式说明其属性，然后用三值逻辑的模型检验技术验证其属性，若为真，表明其系统满足属性；若为未知，表明目前的结构无法确定此属性，需要更完全的 Kripke 结构来描述其结构。

6 与相关工作的比较

在 Glenn Bruns 的文[2]中给出了关于三值逻辑模型检验的算法，该算法也是在原有的二值逻辑的基础上改进形成的三值逻辑模型检验算法。在他的算法中，把一个三值逻辑公式分成两个二值逻辑公式，一个是对所有未知的都取“真”值，一个是对所有未知的都取“假”值。对取“真”值的这个逻辑公式来检查在所建的模型中是否为“假”，若为“假”，则表明原有的三值逻辑公式在此模型下为“假”；然后对取“假”值的这个二值逻辑公式来检查在所建的模型中是否为“真”，若为“真”，则表明原有的三值逻辑公式在此模型下为“真”。对于其他情况，则为“未知”。此算法主要是检查公式的“真”、“假”，而对“未知”是根据公式的真和假来推断出来的。本文提出的算法是在模型检验过程中对公式的真与未知情况做检查，再由此推断出公式为假的情况。此方法的一个好处就是在对公式的检查过程中，及早发现子公式为未知的情况，这样，在对状态模型进行扩充时，就了解哪些状态信息需要细化，以便能够检查出此属性公式的真假，为后面的工作提供了根据。

小结 模型检验的状态爆炸问题是影响其应用的一个关键问题，对系统抽象是解决的一种方法。在抽象的过程当中，可能出现一些无法判定的情况，即对一些属性在目前的结构中无法确定其真或假，这时在模型检验中就出现三值逻辑：

(下转第 278 页)



(a)请求响应时间　　(b)性能提高百分比

图3　RAID_IBM 的 I/O 性能，trace 为 ibm_18es

4.2　实验结果及分析

对于不同的分条单元大小：在 trace 为 hp_2247a 时 RAID_hp 的 I/O 性能，以及在 trace 为 ibm_18es 时 RAID_IBM 的 I/O 性能分别见图 2 和图 3。其中(a)图为阵列的请求响应时间，(b)图为采用了 PMSH 的阵列对应于原阵列的性能提高百分比。

图 2(a)和图 3(a)表明：在本文讨论的各种分条单元大小的情况下，通过采用 PMSH 策略，即分条单元根据访问率的高低在数据传输率不同的磁盘分区间迁移，2 个实验阵列的 I/O 性能得到显著提高。其中图 2(b)显示，在 trace 为 hp_2247a 时，RAID_hp 采用 PMSH 策略后，请求响应时间较原阵列降低 5.4%～18.5%；图 3(b)显示，在 trace 为 ibm_18es 时 RAID_IBM 采用 PMSH 策略后，请求响应时间降低 2.8%～10.5%。但从图中我们并未观察到性能提高百分比与阵列分条单元大小之间的关系。

由图 2(b)，图 3(b)可以看到：与 RAID_hp 相比，RAID_IBM 通过 PMSH 获得的性能提高相对较小。这可能是因为与磁盘 hp_c3323a 相比，磁盘 IBM_DNES-309170W 具有更多的磁盘分区，且各分区间数据传输率的差异较小，从而使得阵列 RAID_IBM 通过将热分条单元数据迁移到快分区以提高 I/O 性能的潜力也相对较小。

结论及展望　本文提出了一种在磁盘阵列中进行分条单元数据迁移的策略，即利用多分区磁盘具有远轴心区数据传输率较高的特点，根据分条单元的访问率，动态地将访问率高的分条单元迁移到其所在磁盘的数据传输率较高的分区。实验证明该策略可以使磁盘阵列的 I/O 性能得到较大的提高：请求响应时间减少 2.8%～18.5%。文章还表明：阵列磁盘的各分区间的数据传输率的差异越大，则通过该数据迁移策略能够提高的性能相对也就越大。

我们将在接下来的工作中把阵列中分条单元在磁盘分区迁移的策略运用到采用了日志技术的 RAID[11]，直觉上它比传统磁盘阵列更适合于利用数据迁移来提高 I/O 性能。因为数据的访问时间包括三部分：寻道时间，旋转延迟和数据传输时间，对于采用了日志技术的 RAID，大多数磁盘请求为大数据访问，数据传输时间占其访问时间的绝大部分；此外，采用日志技术的 RAID 有一个“垃圾清理”过程，因而数据迁移既可以在系统空闲时进行，也可以在阵列“垃圾清理”时进行。

参 考 文 献

1　Meter R V. Observing the effects of multi-zone disks. In: Annual Technical Conf. Anaheim, CA (USENIX, ed.), 1997. 19～30
2　Seagate. Seagate Web Page. http://www.seagate.com. 2000
3　Meter R V. Observing the effects of multi-zone disks. In: Annual Technical Conf. Anaheim, CA (USENIX, ed.), 1997. 19～30
4　Ghandeharizadeh S, Ierardi D, Kim D, Zimmermann R. Placement of Data in Multi-Zone Disk Drives. In: Second Intl. Baltic Workshop on DB and IS, 1996
5　Wang J, Hu Y. PROFS-Performance-Oriented Data Reorganization for Log-Structured File System on Multi-Zone. Modeling. Analysis and Simulation of Computer and Telecommunication Systems. In: Proc. Ninth International Symposium, Aug. 2001. 285～292
6　Chen S, Thapar M. Zone-bit-recording-enhanced video data layout strategies. In: Proc. of the 4th int'l Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'96), 1996
7　Nerjes G, Muth P, Weikum G. Stochastic service guarnantees for continuous data on multi-zone disks. In: Proc. of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposim on Principles of Database Systems, Tucson, Arizona, 1997. 154～160
8　Patterson D A, Chen P, Gibson G, Katz R H. Introduction to Redundant Array of Independent Disk. In: 34th IEEE Computer Society Intl. Conf. Intellectual Leverage, Digest of Papers, 1989. 112～117
9　Sivan-Zimet M. A comparison of access pattern(92-99). UCSC CS290S Project, Dec. 2000
10　Ganger G R, Patt Y N. Using system-level models to evaluate I/O subsystems designs. IEEE Transactions on Computers, 1998. 667～678
11　Gabber E, Korth H F. Data Logging: A Method for Efficient Data Updates in Constantly Active RAIDs. Data Engineering. In: Proc. 14th Intl. Conf. 1998. 144～153

(上接第 266 页)

真、假和未知。本文提出了在不完全 Kripke 结构中基于三值 CTL 逻辑公式的模型检验的算法。同时，对于三值模型检验算法的复杂度及其应用也做了分析，说明此算法在对硬件进行模型验证时，并没有增加其复杂度，是一种可行的方法，能够解决一些领域由于状态爆炸无法进行模型检验的问题。

参 考 文 献

1　Clarke E M, Grumberg O, Peled A D. Model Checking. MIT Press 1999. 171～184
2　Bruns G, Godefroid P. Model Checking Partial State Spaces with 3-valued Temporal Logics. In: Proc. of the 11th Conf. on Computer Aid Verification, of Lecture Notes in Computer Science, Springer Verlag, July 1999, 1633: 274～287
3　Kleene S C. Introduction to Metamathematics. North Holland, 1987
4　Huth M, Jagadeesan R, Schmidt D. Modal transition systems: a foundation for three-valued program analysis. In: Sands D, ed. Proc. of the European Symposium on Programming (ESOP' 2001), Springer Verlag, April 2001. 155～169
5　Gdoerfroid P, Jagadeesan R. On the expressiveness of 3-Valued Models. VMCAI2003, LNCS 2575, 2003. 206～222