软件模型检测中的抽象*)

袁志斌　徐正权　王能超
(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　软件模型检测对保证软件的正确性和可靠性具有十分重要的意义，而抽象是减轻模型检测中状态爆炸问题最重要的技术之一。本文综述当前广泛应用于软件模型检测中的抽象技术，介绍了该领域的进展及研究方向。

关键词　软件模型检测，抽象技术，状态爆炸

Abstraction in Software Model Checking

YUAN Zhi-Bin　XU Zheng-Quan　WANG Neng-Chao
(Department of Computer Science and Technology, Huazhong University of Science Technology, Wuhan 430074)

Abstract　Software model checking is of significant importance to guarantee the correctness and reliability of software systems. Abstraction is one of the most important technique for reducing the state explosion problem in model checking. In this paper, we provide a survey of these abstraction techniques that are used widely in software model checking. Some achievements and research directions for future work are also discussed.

Keywords　Software model checking, Abstraction technique, State explosion

1　引言

随着社会对信息技术的依赖性日益增长，如何提高处于信息技术核心的计算机软件的可靠性和正确性就成为了一个紧迫的问题。自20世纪60年代起，很多计算机科学家对程序验证进行了研究，提出了各种理论和方法。其中形式化验证方法植根于严格的数学与逻辑，其发展前景为大家所看好。

形式化验证技术主要包括定理证明和模型检测两种方法。定理证明[1,2]是将要验证的系统及性质用合适的逻辑系统表示为逻辑公式，然后用定理证明器证明性质在系统中是否被满足。定理证明的优点是它可以适用于各类系统，包括无穷状态系统；其缺点是自动化程度不高，在证明的过程中需要大量的人工干预。如果公式被证伪，定理证明不能提供相关的诊断信息。

模型检测[3,4]是一种关于系统性质验证的算法方法，它通过状态空间搜索的方法来检测一个给定的计算模型是否满足某个用时序逻辑公式表示的特定的性质。对于有穷状态系统，这个问题是可判定的，即用计算机程序可以在有限时间内自动确定。模型检测技术的优点是自动化程度高，不需要使用者掌握大量的逻辑知识。当所设计的系统不满足某个性质时，模型检测工具可以返回一个反例，通过对反例的解读可以得到性质不成立的原因，为系统的修正提供重要线索。

模型检测是基于对状态空间的穷举搜索，对于并发系统，其状态的数目往往随并发分量的增加而呈指数增长。因此，当系统的并发分量较多时，直接对其状态空间进行搜索，实际上是不可能的，这就是所谓的状态爆炸问题。由于软件涉及无穷数据域上的运算，所以状态爆炸问题十分突出，已成为模型检测应用于软件系统的一个具有挑战性的难题。

对于这一难题人们提出了很多减少和压缩状态空间的方法，如符号模型检测[5]、On-the-fly模型检测[6]、对称模型检测[7,8]、程序切片[9]、抽象[10~12]等。在众多的状态减少和压缩技术中，抽象技术是解决状态爆炸问题的最有潜力的方法之一，同时抽象也是有效集成模型检测和定理证明的方法[13]，通过抽象技术可以实现两种形式化方法的互补。

所谓抽象指的是从具体对象构建其抽象模型的理论、方法、技术和工具[14]。抽象的基本思想是通过去掉一些与验证性质无关的信息，从而使得检测可以在一个相对简单的系统上进行。一个有效的抽象应该满足以下特征：第一，所产生的抽象模型要足够地小，这样才能使模型检测得以完成；第二，所产生的模型不能太粗糙，否则不能保证在模型中保留所需要的检测信息。

2　抽象的理论与方法

根据性质的保存可以将抽象分为弱性质保存抽象和强性质保存抽象[15]。设函数 α 将状态迁移系统 M 映射到抽象状态迁移系统 M'，对于性质 f，如果 M' 满足性质 f，则 M 也满足性质 f，这时称 α 对性质 f 是弱性质保存抽象。如果反过来也成立，则称 α 对性质 f 是强性质保存抽象。强性质保存抽象可以保证具体系统与抽象系统性质的一致性，但是系统的压缩程度有限，所以实际应用中以弱性质保存抽象为主。

根据抽象技术中是如何控制信息流失的方式可以将抽象技术分为欠近似[16]和过近似[10]。欠近似是指去掉了系统中与所验证的性质无关的部分性质，使得抽象系统中的行为比原系统少，因此在抽象系统中不成立的性质在原系统中也不成立。这种抽象方式主要用在程序设计的早期阶段，一般是手工完成，而且很难保证与验证相关的行为不被去掉。过近似是将具体系统中的状态的集合映射成抽象系统中的一个状态，这样抽象系统中的状态得以大幅减少，而系统中的迁移却

*)国家自然科学基金(70271069)支持。袁志斌　博士生，主要研究方向：程序正确性、形式化方法；徐正权　教授，博士，主要研究方向：软件工程、形式化方法；王能超　教授，博士生导师，主要研究方向：并行计算。

得以增加，使得抽象系统拥有的行为会比原系统多，因此一个性质在抽象系统中成立就意味着该性质在原系统中成立。

模拟[17,18]和抽象解释[19]是用来构造抽象及证明抽象正确性常用的两个理论框架。模拟用来研究抽象迁移系统和具体迁移系统之间的结构关系。简而言之，每一个具体迁移都存在一个抽象迁移对它进行模拟，所以抽象迁移系统上的不可达性也会在具体系统中成立。

抽象解释是通过抽象函数 α 和具体函数 γ 来反映具体状态和抽象状态之间的对应关系。抽象函数 α 将具体状态集合映射到与之相对应的抽象状态格中的最小元素。具体函数 γ 将抽象状态映射到与之相对应的最大具体状态集合。$\langle\alpha,\gamma\rangle$ 形成一个 Galois 连接。

两种框架最重要的区别是抽象解释框架给出抽象性质的计算方法，而模拟主要讨论抽象系统的性质保存，并不涉及一个模拟是如何计算的。尽管抽象解释和模拟是两个不同的框架，它们之间是可以相互转化的。给定具体集合和抽象集合之间的模拟关系 ρ，则由在关系 ρ 下的后像集 post[ρ]和最弱前置条件 wp[ρ]所构成的二元关系〈post[ρ]，wp[ρ]〉是与 ρ 相对应的 Galois 连接；如果对具体状态集合的划分构造一个抽象格，则每一个 Galois 连接都可以表达为二元关系〈post[ρ]，wp[ρ]〉的形式。在功能方面，两种框架都可以处理过抽象和欠抽象。

在抽象的理论研究中，其可靠性和完备性是十分重要的概念。在抽象系统中成立的性质则一定在所对应的具体系统中成立，则称该抽象是可靠的[20]。如果抽象算法所得到的抽象系统与原系统是双模拟关系，则称该算法是精确的。对于一个具体系统，如果存在一个与之对应的有穷抽象系统，则可以有一个算法构建该抽象系统，称该算法是完备的[20]。

以下介绍目前常用的一些抽象方法，这些方法并不是完全相互独立的，在实际的验证系统中往往需要集成多种抽象方法，以取得较好的效果。

2.1 数据抽象

数据抽象[10]是一种通用性很强的抽象技术，其基本思想是通过去掉系统的部分数据信息来构建状态数较小的系统模型。当系统变量的定义域为无穷域时，将导致系统拥有无穷状态，而通常所需验证的性质与系统变量的具体值无关，因此可以为每一个变量定义一个有穷的抽象数据类型，来替代原数据类型。设程序中有 n 个变量，其中第 i 个变量的域为 D_i，则程序的状态空间为 $D_1\times D_2\times\cdots\times D_n$。设函数 h_i 将域 D_i 映射到抽象域 D_i'，则 $h=(h_1,h_2,\cdots,h_n)$ 将程序的状态空间映射到了抽象状态空间。接下来，将具体系统上的操作定义到抽象数据上。这样，每一个具体变量都有一个定义在抽象域上的抽象变量与之对应，且每一个具体操作都定义了相应的抽象操作。用这种方法就可以构造出一个状态空间较小的抽象系统。数据抽象可以直接从程序代码中构造抽象模型，其构造过程一般是手工完成的。用数据抽象构造的抽象系统的状态空间会得以压缩，而系统行为却比原系统多，因此这种抽象是过近似的。

2.2 基于抽象解释的抽象

许多抽象技术都可以看作是抽象解释框架的应用[12]。基于偏序集合(特别是格)上单调函数的抽象解释是关于计算机语言近似语义的理论。其基本思想可以看作是通过程序的部分执行来获取关于程序的近似语义信息。因为抽象解释框架构造的是程序的过近似抽象模型，因此程序中的每一个状态迁移都会在抽象系统中有一个抽象迁移与之相对应。

对给定的抽象域，抽象解释提供了一个将原系统自动解释到抽象域的一般框架。一般来说，抽象解释应包括抽象数据域、Galois 连接$\langle\alpha,\gamma\rangle$以及抽象操作集合。其中，抽象函数 α 和具体函数 γ 都是单调函数。

以下简单介绍几种常见的抽象解释的形式。符号抽象：就是通过用整数的符号替代其具体数值，将整数域映射到抽象域{gt，lt，0}上，其中 gt，lt，0 分别表示变量大于 0、小于 0 和等于 0。然后，将程序上的具体操作在抽象域上重新定义。通过以上步骤可以获得原程序的一个近似系统。这种近似系统对于一些包含 $x=0$ 这样的命题是十分有用的。区间抽象：对于变量具体值的变化达到某一个关键点时才会对基于该变量的命题产生影响的情况，可以用关键点的值来替代所对应的区间值的集合，这就是所谓的区间抽象。数据抽象也可以看作是抽象解释的一种应用。

经典的抽象解释框架是用来证明安全性的，该框架并没有考虑时序逻辑和模型检测，而且抽象解释理论并不专用于某个特定的性质或特定的程序，因此它是具有普遍意义的理论框架。

2.3 谓词抽象

谓词抽象[21~24]是应用范围最广的软件系统抽象技术之一。谓词抽象技术首先由 *Gref* 和 *Saidi* 于 1997 年提出。该技术实现了自动将无穷状态系统映射到有穷状态系统的方法，而且基于谓词抽象的软件模型检测有效地结合了定理证明和模型检测技术。在谓词抽象中，抽象状态对应着在该状态上成立的谓词公式的集合。其构造过程是从抽象初始状态出发，通过定理证明器来判定后续状态中每一个谓词公式是否成立。

Das 和 Dill 应用谓词抽象技术对复杂系统的应用进行了研究[25]，他们用布尔变量替代谓词公式，通过这种方法就消除了原系统中的数据变量，使得状态空间大为压缩。

Clarke 提出了相似的技术[11,26]，其基本思想是用与谓词相对应的原子公式来构造抽象函数，并提出了消除抽象反例方法。如果一个反例是伪反例，则找出与具体模型中真实路径不相对应的抽象反例的最短前缀序列。通过将该序列中的最后一个抽象状态所对应的具体状态集合划分为几个子集来消除伪反例。Bensalem 提出了与前向反例分析相对应的后向算法，虽然该算法对伪反例的分析思想与前者相同，但是这种算法可以获得完全不同的抽象模型。

在应用谓词抽象时，选择合适的谓词公式集合是十分重要的。因为抽象系统的状态数与谓词公式的数量呈指数关系，因此谓词公式的个数直接决定了抽象模型系统的规模。其次，在抽象模型的建立过程中，调用定理证明器的次数与谓词公式的数量也呈指数关系，所以谓词公式的数量也决定了构造抽象模型的开销。谓词抽象应用的早期通常由用户确定谓词集合，然后利用定理证明器来计算抽象模型。这种用户驱动的谓词选择方法对于大型程序的效率较低。最近的研究方向是利用各种判定过程来自动寻求合适的谓词集合。

2.4 基于反例的抽象求精

一个有效的抽象应该满足以下特征：第一，所产生的抽象模型要足够地小，这样才能使模型检测得以完成；第二，所产生的模型不能太粗糙，否则不能保证在模型中保留所需要的检测信息。从理论上讲，为证明某性质而构造一个精确的抽象系统，其难度和直接在原系统上对该性质进行证明的难度

是相同的。因此在实际构造抽象系统的过程中，总是首先构造一个相对粗糙的初始抽象模型，然后采用逐步求精的方法来产生一个适用的抽象模型。通过这种方法既可以降低难度，又可以得到适当精度的抽象模型。在基于抽象的模型检测中，另一个追求的目标就是抽象系统的自动生成。通过基于反例的抽象求精方法可以自动生成抽象系统。图1给出了基于反例的谓词抽象求精方法[27]的框架。其过程简介如下：

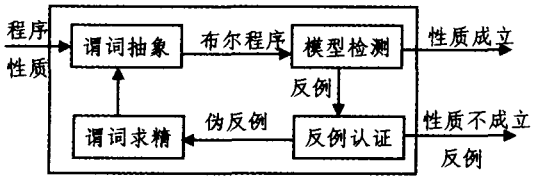

图1 基于反例的谓词抽象求精框图

(1)程序抽象：对给定的初始谓词集合，对程序构造有穷状态迁移系统。

(2)应用模型检测算法检测所需要验证的性质在抽象模型上是否成立。如果成立，则可以报告该性质在原系统中成立，验证过程结束；否则，检测工具会报告一个反例，计算过程进入下一步。

(3)反例确认。这一步的主要目标是判断第2步中产生的反例是否是伪反例。根据反例所提供的信息在原系统中寻找反例所对应的行为。若能找到，则报告所检测的性质为假，验证过程结束；否则，该反例为伪反例，意味着该抽象模型太粗糙，需要进行进一步求精，过程转入下一步。

(4)谓词求精。对谓词集合进行变换以消除已经发现的伪反例。基于新的谓词集合，验证过程进入下一个循环。

Henzinger提出的懒惰抽象技术是一个高效的抽象系统计算方法[28]。其基本思想是在基于反例的谓词抽象求精方法中使用了on-the-fly技术，也就是在构造抽象迁移系统及求精的过程中根据验证的需要来进行抽象状态的计算，避免对无关的状态进行计算。

从以上的讨论可以看出，抽象技术对于大状态空间，特别是无穷状态空间的复杂系统的验证是十分重要的。各种抽象技术的适用范围、技术特点不尽相同，表1对主要的抽象技术从自动化程度、通用性，以及抽象的种类进行比较。

表1 抽象技术比较

技术	自动化程度	通用性	抽象种类
数据抽象	低	高	过近似
基于抽象解释的抽象	低	高	过近似
谓词抽象	中	中	过近似
基于反例的谓词求精	高	中	过近似
懒惰抽象	高	中	过近似

3 基于抽象的软件模型检测工具

抽象技术在软件模型检测工具中得到了广泛的应用，下面就简单介绍几个基于抽象的软件模型检测工具。

SLAM[29,30]是微软公司所开发的C语言程序的模型检测工具包，该工具已经成功应用于验证Windows XP中的一些驱动程序的接口的正确性。SLAM是基于反例的迭代抽象求精的方法来构造原C语言程序的抽象模型——布尔程序，该程序包含了原C语言程序所有的控制流结构，但是只包含布尔变量。该工具包主要包括以下组成部分：C2bp是一个谓词抽象工具，负责将C语言程序抽象为布尔程序；Bebop是一个基于布尔程序的模型检测工具；Newton是一个发现新的谓词并对原布尔程序进行求精的工具。SLAM的特点是自动化程度高，而且可以有效降低检测中的伪反例的数量。

JavaPathFinder(JPF)[31,32]是美国国家航空航天局开发的针对java语言的验证工具。JPF集成了模型检测、程序分析和测试等技术。在状态压缩方面JPF应用了以下技术：偏序规约、谓词抽象、基于类型的抽象、对称化简等。JPF的模型检测过程是在一个自行开发的可以运行所有java字节码的java虚拟机上完成的，通过这种方法使得JPF支持java所有语言特征及java库的模型检测。JPF的另一个重要特征是它支持分布式存储的模型检测。迄今为此，JPF已经应用于多个实际系统，取得了很好的效果。

BLAST[28,33]是加州大学伯克莱分校开发的针对C程序的软件模型检测工具。该工具是基于反例自动抽象求精的技术来构造抽象模型的。BLAST的特点是在模型构造、模型检测、模型求精的循环中用懒惰抽象技术，使得效率得以提高。

MAGIC[34,35]是卡内基梅隆大学开发的针对C程序的软件模型检测工具。该工具也是基于反例自动抽象求精的技术来构造抽象模型。MAGIC的特点是不仅可以检测顺序程序，而且可以检测并发程序。MAGIC的另一个特点是实现了模型分解技术，该工具可以将大系统分解成多个小的、易于处理的小系统，分别进行验证，通过这种方法可以大大提高模型检测的规模。

基于抽象的软件模型检测工具很多，其余不一一列举。值得注意的是，每一个开发工具在对付状态爆炸问题上都采用了多种方法相结合，以期达到一个较好的效果。

结束语 由于软件系统的可靠性受到了越来越多的关注，形式化验证技术已经得到了相当程度的重视，基于抽象的软件验证也取得了丰硕的成果。但是，由于软件本身所固有的复杂性，使得这一领域尚有大量的课题期待进一步的研究。

对程序进行抽象的方法众多，而且很多方法是正交的。多种抽象技术的结合无疑会提高压缩的效率，但是迄今为止还缺乏一个有效的、集成多种抽象技术的框架，毫无疑问这是一个极具研究价值的方向。

现在的系统模型及检测技术都主要集中在对安全性的验证上，而软件系统的活性验证也是一个十分重要的内容。建立适合活性验证的抽象模型描述方法、抽象模型的构造算法，以及逐步求精的方法是今后的一个重点研究对象。

参考文献

1 Boulton R J. Efficiency in a Fully-Expansive Theorem Prover: [Phd thesis]. University of Cambridge Computer Laboratory, May 1994

2 Rajan S, Shankar N, Srivas M K. An integration of model checking with automated proof checking. In: Computer-Aided Verification, 1995. 84～97

3 Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching time temporal logic. In: Proc. Workshop on Logic of Programs, Vol 131 of LNCS, 1981. 52～71

4 Clarke E M, Grumberg O, Peled D. Model Checking. Massachusetts: MIT Press, 1999

5 McMillan K L. Symbolic Model Checking. Kluwer Academic Publishers, 1993

6 Holzmann G J. Design and Validation of Computer Protocals. NewJersey: Prentice Hall, 1991

7 Emerson E A, Sistla A P. Symmetry and model checking. Formal

Methods in System Design, 1996, 9(1/2): 105～130
8 Emerson E A, Trefler R J. From asymmetry to full symmetry: new techniques for symmetry reduction in model checking. In: Correct Hardware Design and Verification Methods. Vol 1703 of LNCS, 1999. 142～156
9 Weiser M. Program slicing. IEEE Transactions on Software Engineering, 1984, 10(4): 352～357
10 Clarke E M, Grumberg O, Long D E. Model checking and abstraction. ACM Transactions on Programming Languages and System (TOPLAS), 1994, 16(5): 1512～1542
11 Clarke E, Grumberg O, Jha S. Counterexample-Guided abstraction Refinement for symbolic Model checking, Journal of the ACM, 2003, 50(5): 752～794
12 Dams D, Gerth R, Grumberg O. Abstract interpretation of reactive systems. ACM Transactions on Programming Languages and System(TOPLAS), 1997, 19(2): 253～291
13 Rushby J. Integrated formal verification: using model checking with automated abstraction, invariant generation, and theorem proving. In: Theoretical and practical aspects of SPIN model checking. 5th and 6th international SPIN workshops, 1999. 1～11
14 Dams D. Abstraction in software model checking: Principles and practice(tutorial overview and bibliography). Model Checking Software. Vol 2318 of LNCS, Springer-Verlag, 2002
15 Dams D. Abstract Interpretation and Partition Refinement for Model Checking: [PhD thesis]. Eindhoven University of Technology, P. O. Box 513, 5600 MB Eindhoven, The Netherlands, July 1996
16 Lee W, Pardo A, Jang J, et al. Tearing based abstraction for CTL model checking. In: International Conference of Computer-Aided Design, 1996. 76～81
17 Park D. Concurrency and automata on infinite sequences. In: Proceedings of the 5th GI-Conference on Theoretical Computer Science, Springer-Verlag, 1981. 167～183
18 Milner R. An algebraic definition of simulation between programs. In: Proceedings of the 2nd International Joint Conference on Artificial Intelligence, 1971. 481～489
19 Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of programming languages, 1977. 238～252
20 Dams D, Namjoshi K S. The existence of finite abstractions for branching time model checking. In Nineteenth Annual IEEE Symposium on Logic in Computer Science(LICS'04), IEEE Computer Society Press, 2004. 335～344
21 Graf S, Saidi H. Construction of abstract state graphs with PVS. In: Proc. 9th International Conference on Computer Aided Verification(CAV'97). Vol 1254 of LNCS, Springer-Verlag, 1997. 72～83
22 Colon M, Uribe T. Generating finite-state abstractions of reactive systems using decision procedures. In: Computer Aided Verification, 1998. 293～304
23 Ball T, Podelski A, Rajamani S K. Boolean and Cartesian Abstraction for Model Checking C Programs. In: Proceedings of TACAS: Tools and Algorithms for the Construction and Analysis of Systems. Genova, Italy, April 2001
24 Ball T, Majumdar R, Millstein T, et al. Automatic Predicate Abstraction of C Programs. In: Proceedings of the Conference on Programming Language Design and Implementation(PLDI 2001), Snowbird, Utah, June, 2001
25 Das S, Dill D L, Park S. Experience with predicate abstraction. In Computer-Aided Verification, Vol 1633 of LNCS, Springer-Verlag, July 1999. 160～171
26 Clarke E, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement, In: Computer Aided Verification, 2000. 154～169
27 Clarke E, Kroening D, Sharygina N, et al. Predicate Abstraction of ANSI-C programs using SAT, Formal Method in System Design, 2004, 25: 105～127
28 Henzinger T A, Jhala R, Majumdar R, et al. Lazy Abstraction. In: Proceedings of the 29th Annual Symposium on Principles of Programming Languages(POPL). ACM Press, 2002. 58～70
29 Ball T, Rajamani S K. The SLAM project: debugging system software via static analysis. In: Proceedings of the 29th ACM SIGPLANSIGACT Symposium on Principles of Programming Languages. ACM Press, 2002. 1～3
30 SLAM. http://reseach. microsoft. com/slam
31 Java PathFinder. http://ase. arc. nasa. gov/visser/jpf
32 Visser W, Havelund K, Brat G, et al. Model Checking Programs. Proceedings of the 15th International Conference on Automated Software Engineering(ASE), Grenoble, France, September 2000
33 BLAST. http://www-cad. eecs. berkeley. edu/～rupak/blast
34 MAGIC. http://www. cs. cmu. edu/～chaki/magic
35 Chaki S J. A counterexample guided abstraction refinement framework for verifying concurrent c programs: [PhD thesis]. Carnegie Mellon University, 2005

(上接第 248 页)

加而减小，但 PA 算法的性能始终优于其它算法。

由此，可以得出可倒置法的性能优于基本回收算法，尽早启动算法以及约束向量法的结论。

结论 在实时多处理器系统中，为了提高系统的性能，使得新到达的任务有较多的机会得到执行，常常采取资源回收算法回收未被任务使用的资源。在研究已有资源收回算法的基础上，提出可倒置法。可倒置法通过允许只存在资源访问冲突的任务之间出现执行顺序倒置，提高资源回收效率，从而能够更好地回收资源。模拟结果表明，可倒置法优于基本回收算法、尽早启动算法以及约束向量法。

参 考 文 献

1 Ramamritham k, stankovic J A, Scheduling algorithms and operating systems support for real-time systems. Proceeding of IEEE, 1994, 82(1): 55～67
2 Shin K G, Ramanathan R. Real-time computing a new discipline of computer science and engineering. Proceedings of IEEE, 1994, 82(1): 6～24
3 Ramamritham K, Stankovic J A. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE transactions on Parallel and distributed systems, 1989, 1(2): 184～194
5 Hou C J, Shin K G. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. IEEE transactions on Computers, 1997, 46(12): 1338～1356
6 Hong K S, Leung J Y T. On-line scheduling of real-time tasks. IEEE transactions on computer, 1992, 41(10): 1326～1331
7 Shen C, Ramamritham K, Stankovic J A. Resource reclaiming in multiprocessor real-time systems. IEEE Transactions on parallel and distributed systems, 1993, 4(4): 382～397
8 Manimaran G, Murthy C S R, Vigay M, et al. New algorithms for resource reclaiming from precedence constrained tasks in multiprocessor real-time systems. Journal of parallel and distributed computing, 1997, 44(2): 123～132
9 Gupta I, Manimaran G. A new strategy for improving the effectiveness of resource reclaiming algorithms in multiprocessor real-time systems. 1998