# 基于二型模糊逻辑的多线程数据竞争检测方法研究

杨 璐 余守文 严建峰
(苏州大学计算机科学与技术学院 苏州 215006)

**摘 要** 多线程机制以其诸多优势在程序开发中被广泛使用,然而随着多线程软件规模的增长,程序中潜存着许多并发缺陷,最常见的并发缺陷是数据竞争和死锁。目前,针对这些并发缺陷的检测手段都无法处理线程时序的不确定性,无法处理运行时环境对线程时序的影响,同时也不能计算这些并发缺陷发生的概率并根据概率生成其处理优先级。针对以上问题,提出了一种基于二型模糊逻辑的多线程数据竞争检测方法。该方法将传统的多线程时序分析和缺陷检测方法作为预处理,考虑程序运行时环境因素对线程时序的影响,利用二型模糊逻辑和隐马尔科夫模型对待检测程序建模,计算待检测程序在某一系统负载下的时序概率,并根据时序概率生成时序缺陷处理优先级列表供软件开发人员参考。

**关键词** 二型模糊逻辑,隐马尔科夫模型,数据竞争检测
**中图法分类号** TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.027

## Type-2 Fuzzy Logic Based Multi-threaded Data Race Detection

YANG Lu YU Shou-wen YAN Jian-feng
(School of Computer Science and Technology, Soochow University, Suzhou 215006, China)

**Abstract** Multi-threaded mechanism has been widely used in software development because of its advantages. However, with the growth of program scales, there are plenty of potential parallel defects in multi-threaded programs. The most common parallel defects are data race and deadlock. However, none of the traditional defect detection methods take into account the uncertainty of time sequence analysis and run-time environment. And it is hard to calculate the probability of parallel defects to generate a priority order list based on the probability. To solve these problems, we proposed a data race detection method based on type-2 fuzzy logic. This method considers the influence of run-time environment factors, and uses the traditional parallel defects detection methods as pre-processing step. Then it builds a time sequence analysis model for the target program based on type-2 fuzzy logic and hidden Markov model. It can calculate the probability of all the potential defects, then generates a priority order list for software developers to deal with defects and allocate resources.

**Keywords** Type-2 fuzzy logic, Hidden Markov model, Data race detection

## 1 引言

多线程机制以其诸多优势在软件开发过程中被广泛应用,也是当前大数据处理领域的基础技术之一。但是引入多线程机制也会带来一些问题:由于线程运行的时序不可预测,线程对共享资源的访问顺序也难以控制,因此在输入相同的情况下,很大可能会得到不同的输出,即每一次的运行结果都可能不同,这是由程序运行过程中并发事件发生顺序的不可预测所引起的。

由此导致的常见多线程软件并发缺陷包括数据竞争(Data Race)和死锁(Deadlock)。当软件缺陷发生的条件满足时,就会发生软件错误,这些错误可能会使整个软件或系统平台崩溃,相当于是埋在软件系统中的一颗定时炸弹。如果错误发生在对软件可信性要求极高的行业,如航空航天、股票金融、医疗卫生等,将会造成难以估量的巨大损失。

国内外已对该类并发缺陷检测的方法研究了多年,并提出了多种思路。目前主流的缺陷检测手段有:属于验证(verification)的方法,包括模型检验(model checking)[1-4]和定理证明(theorem proving)[5];属于分析(analysis)的方法,包括静态分析(static analysis)[6-13]、动态分析(dynamic analysis)[14-21]以及上述两者的混合技术[22-23];此外,还有测试方法[24-27]以及监控方法[28-30]。总的来说,上述传统的缺陷检测方法在检测数据竞争和死锁问题时的特点各异,在实际应用中也各有其优点和局限性。

到稿日期:2016-12-21 返修日期:2017-02-07 本文受国家自然科学基金项目(61202029,61272449,61572339),江苏省科技支撑计划重点项目(BE2014005-4)资助。

杨 璐(1982-),女,博士,副教授,CCF会员,主要研究方向为可信软件、机器学习;余守文(1993-),男,主要研究方向为软件工程;严建峰(1978-),男,博士,副教授,CCF会员,主要研究方向为并行计算、机器学习,E-mail:yanjf@suda.edu.cn。

并发缺陷主要是由多线程并发机制的不确定性(uncertainty)引起的,因此不确定性是缺陷检测中多线程时序分析的难点。然而,传统缺陷检测方法在处理多线程时序分析中普遍存在的不确定性时存在不足:这些方法给出的结果都是描述缺陷是否存在确定的结果(例如,1 代表缺陷被检测到,0 代表缺陷未被检测到),并未描述其不确定性。因此,这些方法没有对缺陷发生的概率进行评估,检测结果仍然存在误报和漏报的问题。在实际软件工程中,资源是有限的,无法对所有并发缺陷一视同仁地进行处理。因此,有必要计算并发缺陷发生的概率,并据此确定处理缺陷的优先级顺序,这样软件工程师才能将更多资源集中在高优先级(发生概率高)的缺陷上。然而,由于目前这部分工作的实际成功案例较少,因此缺陷发生的概率往往依赖于软件工程师的经验来评估。

为了解决多线程时序分析中的不确定性问题,将模糊系统理论引入时序分析领域来描述多线程程序中的不确定性。本文将环境因素(例如,系统的运行负载等)对多线程时序关系的影响看作是一个随机过程,用其描述多线程系统中的不确定性。通过对反映环境因素的观察值进行大量实验,得到训练样本集;在此基础上进行训练,采用基于区间型二型模糊逻辑的隐马尔科夫模型(IT2 FHMM)[31-32]建立多线程时序分析模型,以此构造环境因素和多线程时序关系之间的不确定性联系;在该时序分析模型的基础上,进一步进行时序分析,根据实际观察值序列直接计算获得在相应观察值序列下发生缺陷的概率,并进一步评估处理缺陷的优先级顺序。

采用模糊系统理论进行多线程时序分析研究的工作较少,仅有文献[33]等少量研究工作见诸报道,其工作基于一型模糊逻辑,且没有展示用于数据竞争和死锁检测领域的实验效果。为了强化模糊程度,采用二型模糊逻辑,相较于一型模糊逻辑,IT2 FHMM 能处理两种不确定性(随机性和模糊性),从而增强了传统隐马尔科夫模型的表达能力。为了验证本文方法的有效性,针对同一个目标程序在二型模糊逻辑和一型模糊逻辑上进行了实验,实验结果表明,本文方法在时序分析中的精确度更高,因此在缺陷检测领域的效果也更好。

本文第 2 节介绍了本文所需的理论基础,包括隐马尔科夫模型、二型模糊系统理论和基于区间二型模糊逻辑的隐马尔科夫模型;第 3 节介绍了基于 IT2 FHMM 进行数据竞争检测的方法,并使用具体示例程序进行实验,来验证方法的有效性;第 4 节总结了本文研究方法,并介绍了进一步的工作。

本文方法思路在死锁领域的初步探索见文献[34]。

## 2 隐马尔科夫模型和二型模糊系统

本节介绍隐马尔科夫模型和二型模糊系统的定义和相关算法,然后引入基于区间型二型模糊逻辑的隐马尔科夫模型,并将其用于多线程程序中数据竞争的检测。

### 2.1 隐马尔科夫模型

隐马尔科夫模型(Hidden Markov Model,HMM)[35]是一种双重嵌入式随机过程。在普通马尔科夫模型中,马尔科夫链的状态对观察者是可见的。而隐马尔科夫模型引入了状态转移的概念,其状态是不直接可见的,但在每个状态上产生的观测向量是可见的。每一个状态对于各自可以产生的观察值都有一个概率分布,因此观测向量序列在一定程序上可以反映出隐含状态序列的信息。

一个 $N$ 状态的隐马尔科夫模型可以用三元组 $\lambda=\{\pi_i, a_{ij}, b_i(o_t), 1\leqslant i,j\leqslant N\}$ 表示。$\pi_i$ 是初始的状态分布;模型中包含 $N$ 个隐含状态 $S_j(1\leqslant j\leqslant N)$,每一个状态对于相应的观察值都有一个概率密度函数 $b_j(o_t)$,该概率密度函数反映了在 $t$ 时刻观察值为 $o_t$ 的概率;$a_{ij}$ 表示从状态 $S_i$ 转移到状态 $S_j$ 的概率,$\sum_{j=1}^{N} a_{ij}=1$。在隐马尔科夫模型中的观察值序列 $O_1,O_2,\cdots,O_T$ 可由状态序列 $X_1,X_2,\cdots,X_T$ 产生。

构成隐马尔科夫模型需要满足以下 3 个条件:

(1)状态序列需要满足马尔科夫性,可以构成马尔科夫链,$P(X_i|X_{i-1},\cdots,X_1)=P(X_i|X_{i-1})$。

(2)状态与具体的时间无关,$P(X_{i+1}|X_i)=P(X_j|X_{j-1})$。

(3)观察值的输出仅与当前的状态相关,$P(O_1,\cdots,O_T|X_1,\cdots,X_T)=\prod P(O_t|X_t)$。

用隐马尔科夫模型可以解决以下 3 种问题:

(1)给定观察序列 $O=O_1,O_2,\cdots,O_T$ 及模型 $\lambda$,计算 $P(O|\lambda)$。即已知模型参数,计算某一特定输出序列的概率。通常使用 forward 算法解决。

(2)给定观察序列 $O=O_1,O_2,\cdots,O_T$ 及模型 $\lambda$,选择一个对应的状态序列 $S=q_1,q_2,\cdots,q_T$,使得 $S$ 能够最为合理地解释观察序列 $O$。即已知模型参数,寻找最优(最大似然)的产生某一特定输出序列的隐含状态的序列。通常使用 Viterbi 算法解决。

(3)给定观察值序列 $O$ 及维数 $N$ 和 $M$,调整模型参数 $\lambda$,使得 $P(O|\lambda)$ 最大。即已知输出序列,寻找最可能的状态转移和输出概率,可用于训练模型。通常使用 Baum-Welch 算法解决。

在上述问题中,本文主要针对问题(2)进行求解。

### 2.2 二型模糊系统理论

模糊系统理论以 1965 年提出的模糊集合理论为基础,用隶属度和模糊集合描述事物中存在的不确定现象。它将人类专家系统对某些事物和过程的操作归纳为一系列模糊规则,以模糊推理为核心处理事物中广泛存在的模糊性。

传统的模糊系统理论用隶属度(membership)这一概念对集合中的元素加以模糊化,得到的模糊集合称为一型模糊集(Type-1 Fuzzy Sets,T1 FS),基于一型模糊集建立的系统称为一型模糊系统(Type-1 Fuzzy Logic System)[36]。在一型模糊逻辑中,模糊集里的元素不是确定地属于或不属于该集合,而是用隶属度来表示一个元素属于该模糊集的程度,隶属度是一个确定值。

为了增强描述的集合模糊度,将一型模糊集进行扩展,进一步对集合中元素的隶属度进行模糊化,这种模糊集就是二型模糊集(Type-2 Fuzzy Sets,T2 FS),以二型模糊集为基础建立的系统则称为二型模糊系统(Type-2 Fuzzy Logic System)[37-40]。在二型模糊逻辑中,不仅某个元素属于某个集合的隶属度具有模糊性,该隶属度自身也具有模糊性,隶属度的隶属函数(membership function,MF)是三维的。随机数据的可能性可以用主隶属函数(principal membership function)来

表示。由于主隶属度的不确定性，可以用误差棒来表示主隶属度，例如，$[c_1-\mu, c_1+\mu]$和$[c_2-\mu, c_2+\mu]$(见图 1(a))。而可能性的模糊性，即在主隶属度误差棒内的不确定性，可以进一步使用次隶属函数(secondary membership function)来描述：在误差棒内的次隶属度可以服从一个一型的高斯型隶属函数(见图 1(b))或是一型的区间型隶属函数(见图 1(c))，对应的模糊集称为高斯二型模糊集或区间二型模糊集。二型模糊集用两个基本概念来描述不确定性，分别是次隶属函数和 footprint of uncertainty(FOU)。

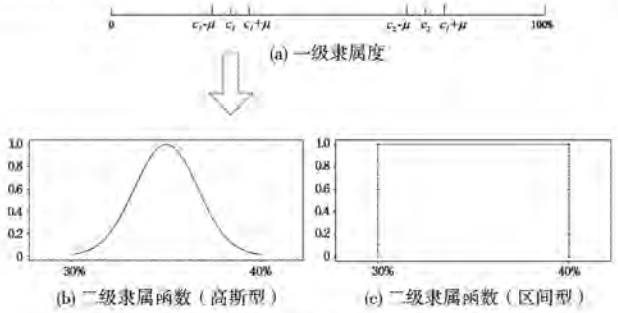


图 1　二型模糊逻辑的模糊化

基于二型模糊逻辑的模型输出的结果不是一个精确值的集合，而是一个二型模糊集，需要把该二型模糊集转换成确定的输出。在精确化之前必须先经过降型处理，常用的二型模糊集降型方法有：Center-of-Sums Type-Reduction，Centroid Type-Reduction，Modified Height Type-Reduction，Height Type-Reduction，Center-of-Sets Type-Reduction 等。

一个典型的二型模糊系统包括以下模块：规则库(Rules)、输入模糊器(Fuzzifier)、推理引擎(Interference)、精确器(Defuzzifier)，这些模块与一型模糊系统相同；不同的是，由于二型模糊系统操作的对象是二型模糊集，因此其比一型模糊系统多了降型器(Type-reducer)，如图 2 所示。

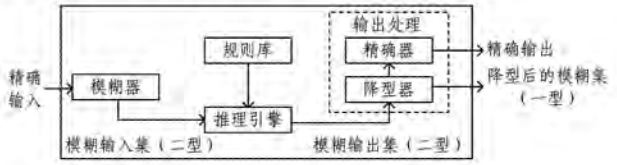


图 2　二型模糊系统原型

### 2.3　基于二型模糊逻辑的隐马尔科夫模型

在实际情况中使用隐马尔科夫模型时，往往得不到足够的模型训练数据，并且训练数据中可能存在噪声，这些都会影响隐马尔科夫模型评估未知参数的能力。针对这种情况，可以使用基于二型模糊逻辑的隐马尔科夫模型(Type-2 Fuzzy HMM，T2 FHMM)[31-32]，以有效处理样本数据中存在的噪声。该模型的优点在于，它能处理两种不确定性(随机性和模糊性)，从而增强传统隐马尔科夫模型的表达能力。这两种不确定性在 T2 FHMM 中体现在将不确定输入映射到二型模糊集，而 T2 FHMM 的输出是一个不确定的一型模糊集。在处理模糊信息时，二型模糊逻辑往往比一型模糊逻辑更精确。

结合传统隐马尔科夫模型的表示方法，T2 FHMM 可表示为$\tilde{\lambda}=\{\tilde{S}, \tilde{q}_t, \tilde{a}_{ij}, \tilde{b}_j(o_t), c_{jm}, \mu_{jm}, \Sigma_{jm}\}$。$\tilde{\lambda}$表示该模型参数的集合，具体的参数含义如下：

1)$\tilde{S}$ 表示隐含模糊状态的状态空间，$\tilde{S}=\{\tilde{S}_1, \tilde{S}_2, \cdots, \tilde{S}_N\}$，其中$\tilde{S}_j$ 是一个隐含模糊状态。

2)$\tilde{q}_t$ 表示在时刻 $t$ 访问的隐含模糊状态。

3)$\tilde{a}_{ij}$ 表示从状态 $\tilde{S}_i$ 转移到状态 $\tilde{S}_j$ 的模糊状态转移概率矩阵。

4)$\tilde{b}_j(o_t)$表示观察值 $o_t$ 在时刻 $t$ 对应于模糊状态 $\tilde{S}_j$ 的二型隶属函数。在 T2 FHMM 中，主隶属函数描述观察值 $o_t$ 属于模糊状态 $\tilde{q}_t$ 的程度，次隶属函数$\tilde{b}_j(o_t)$描述主隶属函数的模糊性，决定了 $o_t$ 对应于 $\tilde{S}_j$ 的隶属度。$\tilde{b}_j(o_t)$是对隐含模糊状态 $\tilde{q}_t$ 的概率分布。

5)$c_{jm}$ 表示 $j$ 时刻模糊状态 $\tilde{S}_j$ 第 $m$ 个混合模型的系数。

6)$\mu_{jm}$ 表示 $j$ 时刻模糊状态 $\tilde{S}_j$ 第 $m$ 个混合模型的均值。

7)$\Sigma_{jm}$ 表示 $j$ 时刻模糊状态 $\tilde{S}_j$ 第 $m$ 个混合模型的协方差矩阵。

设 $h_{\tilde{\lambda}t}(o_t)$表示模糊观察值向量 $o_t$ 的二型隶属度函数。给定一个 T2 FHMM $\tilde{\lambda}$和一个观察值向量序列 $O=o_1, o_2, \cdots, o_T$，可用 $h_{\tilde{\lambda}}(O)$来表示序列 $O$ 属于 T2 FHMM $\tilde{\lambda}$的隶属度的程度。$h_{\tilde{\lambda}}(O)$是一个二型模糊集，其不确定性体现在$\tilde{b}_j(o_t)$和$\tilde{a}_{ij}$中。

$\tilde{b}_j(o_t)$有两种实现：具有不确定均值的高斯型主隶属函数和具有不确定标准差的高斯型主隶属函数。以用具有确定均值 $\mu$ 和不确定标准差$\tilde{\sigma}\in[\underline{\sigma}, \bar{\sigma}]$的高斯型主隶属函数来实现$\tilde{b}_j(o_t)$为例，其具有如下函数形式，如图 3 所示。

$$f(x)=\exp\left[-\frac{1}{2}\left(\frac{x-u}{\sigma}\right)^2\right], \tilde{\sigma}\in[\underline{\sigma}, \bar{\sigma}] \tag{1}$$

其中，上界 MF $\bar{f}(x)$为：

$$\bar{f}(x)=\mathrm{N}(\mu, \bar{\sigma}; x) \tag{2}$$

下界 MF $\underline{f}(x)$为：

$$\underline{f}(x)=\mathrm{N}(\mu, \underline{\sigma}; x) \tag{3}$$

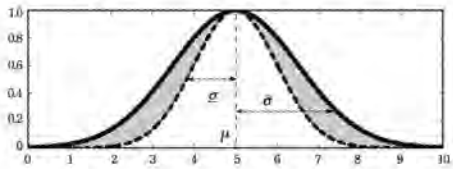


图 3　具有确定均值与不确定标准差的高斯型一级隶属函数

$\tilde{\sigma}\in[\underline{\sigma}, \bar{\sigma}]$区间的大小对结果会产生影响。其区间大小的选择依赖于对该模型不确定性估计的先验知识，可以使用以下形式来描述：

$$\underline{\sigma}=k_1*\sigma, \bar{\sigma}=k_2*\sigma, k_1\in(0,1], k_2\in[1,+\infty) \tag{4}$$

其中，$k_1$ 和 $k_2$ 为不确定因子，用于描述模型的不确定性，其值在重估模型参数之前就需要被确定。

### 2.4　基于区间型二型模糊逻辑的隐马尔科夫模型

由于 T2 FHMM 的运算，特别是其中的降型运算，较为复杂。因此可以设次隶属度为 1 来简化运算，这样得到的二型模糊集就是区间型二型模糊集。本文将区间型二型模糊逻辑(Interval Type-2 Fuzzy Logic)应用于隐马尔科夫模型，称为基于区间型二型模糊逻辑的隐马尔科夫模型(Inverval

Type-2 Fuzzy Hidden Markov Model，IT2 FHMM)，使之更适用于实际。

在 IT2 FHMM 中，可将所有变量用区间型隶属度函数来表示，例如$\tilde{a}_{ij}=[\underline{a}_{ij},\overline{a}_{ij}]$，$\tilde{b}_j(o_t)=[\underline{b}_j(o_t),\overline{b}_j(o_t)]$，$h_{\tilde{\Omega}_t}(o_t)=[\underline{h}_{\tilde{\Omega}_t}(o_t),\overline{h}_{\tilde{\Omega}_t}(o_t)]$等。在 IT2 FHMM 的算法中，相应的变量也可以用区间表示和参与运算。

与传统 HMM 类似，IT2 FHMM 同样也有用于解决 3 个问题的相关算法。文献[31-32]中对 IT2 FHMM 模型的定义进行了详细说明，包括基于 IT2 FHMM 的 forward-backward 算法、Viterbi 算法、Baum-Welch 算法，以及 IT2 FHMM 的训练与降型方法，此处不再赘述。

## 3 基于 IT2 FHMM 的数据竞争检测

本节介绍将 IT2 FHMM 用于多线程数据竞争检测的方法，并设计实验来验证该方法的有效性。

该方法的整体框架如图 4 所示，具体可分为如下 5 个步骤。

(1)基于静态分析的预处理。扫描待检测程序，根据待检测程序的时序关系和数据竞争发生的条件，采用静态分析方法分析包括以“线程中的点对”描述的潜在缺陷的位置，以及以“点对中程序点的执行时序”描述的潜在缺陷发生的条件。

(2)设计实验产生训练数据。由于数据竞争发生的条件要求多个线程访问共享变量没有确定的顺序，因此通过在潜在的数据竞争位置对待检测程序插桩，可以监测潜在数据竞争点对在当前环境中的时序关系。从环境因素中选择合适的观察值，循环运行该插桩程序多次，将大量实验中得到的点对的时序关系的概率分布作为模型的训练数据集。

(3)构建 IT2 FHMM 模型。将步骤(2)中得到的训练数据集作为输入，结合 IT2 FHMM 建立多线程时序分析模型，使用 Viterbi 算法初始化模型，然后使用 forward-backward 算法和 Baum-Welch 算法重估模型参数。调整经验参数的值，重新训练，优化建模效果。

(4)基于 IT2 FHMM 的时序分析。使用步骤(3)训练后的 IT2 FHMM，使用 Viterbi 算法计算潜在数据竞争位置点对的时序概率。

(5)评估数据竞争的后处理。根据步骤(4)中计算得到的点对的时序概率，评估步骤(1)中检测到的潜在数据竞争位置点对，计算数据竞争发生的概率，并根据该概率计算出数据竞争发生的优先级，生成优先级列表以供程序开发人员参考。

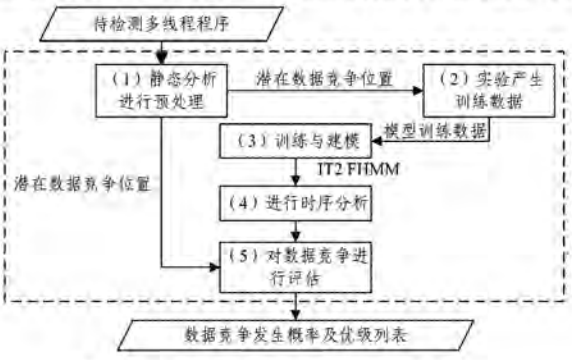


图 4 基于 IT2 FHMM 的数据竞争检测方法

多线程程序中的数据竞争问题[71]是指程序处于这样的状态：由于访问数据的顺序不同，在输入相同的情况下也可能得到不同的输出。在多线程程序中，两个不同线程的事件需要同时满足以下两个条件才可能会产生数据竞争的情况：

(1)这两个事件同时访问同一个共享变量，并且其中至少有一个事件执行的是“写”操作。

(2)这两个事件并未由一个同步对象保护，即这两个事件是非原子的；或者这两个事件之间没有先行发生顺序。

例如，考虑两个线程的情况，数据竞争的场景示例如下：线程 1 和线程 2 同时访问一个共享变量 $y$，并且写共享变量 $y$ 的数据，线程 1 和线程 2 之间没有强制的执行顺序。由于访问共享变量的顺序不同，在输入相同的情况下可能得到不同的输出，这样两个线程就发生了数据竞争，如图 5 所示。图中竖实线表示线程的执行，虚线表示线程之间的交互，双向箭头虚线表示数据竞争对。

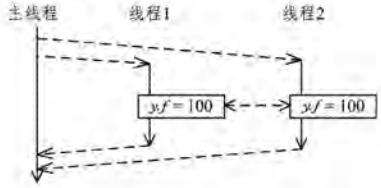


图 5 数据竞争场景示例

两个线程发生数据竞争的最基本的条件是这两个线程访问相同的内存地址；当两个线程访问两个不同的变量时，由于两个变量可能指向同一个内存地址，因此也有可能发生数据竞争。因此，别名分析(Alias Analysis)是数据竞争检测的基础，它的检测精度将显著地影响数据竞争的检测精度。

### 3.1 基于 IT2 FHMM 的数据竞争检测示例程序

本文将用如下示例程序来验证本文方法的有效性，该程序的编码语言为 C++。

```
1. unsigned WINAPI ThreadFun1(void* pM){
2.     int* i=(int*)pM;
3.     (*i)++;//写共享数据存储
4.     return 0;
5. }
6. unsigned WINAPI ThreadFun2(void* pM){
7.     int* j=(int*)pM;
8.     (*j)++;//写共享数据存储
9.     return 0;
10. }
11. unsigned WINAPI ThreadFun3(void* pM){
12.     int* j=(int*)pM;
13.     Sleep(5000);//延迟 5s
14.     (*j)++;//写共享数据存储
15.     return 0;
16. }
17. int main(){
18.     HANDLE hThread[4];
19.     int i=0;
20.     int j=0;
21.     //创建 4 个线程
22.     //第一组两个线程访问变量 i 存在数据竞争，线程函数都是
```

```
    ThreadFun1
23.  hThread[0] = (HANDLE)_beginthreadex(NULL, 0, &
     ThreadFun1,(void*)&i,0,NULL);
24.  hThread[1] = (HANDLE)_beginthreadex(NULL, 0, &
     ThreadFun1,(void*)&i,0,NULL);
25.  //第二组两个线程访问变量 j,线程函数分别是 TheardFun2
     和 ThreadFun3
26.  hThread[2] = (HANDLE)_beginthreadex(NULL, 0, &
     ThreadFun2,(void*)&j,0,NULL);
27.  hThread[3] = (HANDLE)_beginthreadex(NULL, 0, &
     ThreadFun2,(void*)&j,0,NULL);
28.  WaitForMultipleObjects(4,hThread,true,INFINITE);//等
     待所有线程结束
29.  CloseHandle(hThread[0]);
30.  CloseHandle(hThread[1]);
31.  return 0;
32. }
```

在上述示例程序中,主线程 main 中调用_beginthreadex 来创建子线程。子线程 1 与子线程 2 为第一组,这两个子线程执行的操作为 ThreadFunc1;子线程 3 与子线程 4 为第二组,其中子线程 3 执行的操作为 ThreadFunc2,子线程 4 执行的操作为 ThreadFunc3,ThreadFunc2 与 ThreadFunc3 的区别在于 ThreadFunc3 多了一句延迟语句。

### 3.2　基于静态分析的预处理

为了进行数据竞争检测,本文定义别名集合 S,对该别名集合访问的事件有 E1 和 E2,如果 E1 和 E2 同时满足以下 3 个条件,那么它们构成数据竞争:

1)E1.thread != E2.thread

2)E1.action == WRITE || E2.action == WRITE

3)(E1→E2)∧(E2→E1)!= Ø

本文根据数据竞争发生的条件,选择采用基于静态分析的方法。该部分程序共分为 3 个模块:预处理模块(DRPre-Analysis),主分析模块(DRAnalysis),竞争分析模块(getData-Race)。其流程图如图 6 所示,以下将对 3 个模块的功能作详细说明。

(1)预处理模块:通过用户指定的目录寻找程序代码文件,然后静态扫描待检测程序,获取待检测程序中程序员自定义的函数、变量,记录自定义函数的参数列表,并将普通代码语句简单格式化预处理,以方便后面的模块统一分析。这里的简单格式化预处理包括:去除语句前后的空格,和语句末尾的空格,获得赋值语句中等号前、后的变量或函数名称。

(2)主分析模块:通过分析格式化的语句信息,建立变量间的别名关系,存在别句关系的变量被存到一个别名列表中。从主函数开始分析程序,如果一条语句访问了别名列表中的变量,就对该变量建立一条访问记录,并记录是在哪个线程中操作,以及操作的性质是读还是写。

(3)竞争分析模块:依赖于主分析模块的结果,结合数据竞争条件,对每一个别名集合的访问记录进行分析,从而得到检测结果。

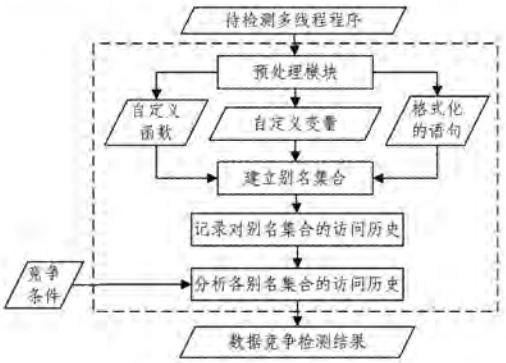


图 6　数据竞争静态分析程序流程图

通过对 3.1 节中示例程序进行静态分析,可以得到其程序执行的时序图,如图 7 所示,其中各重要时序点对应的事件如表 1 所列。

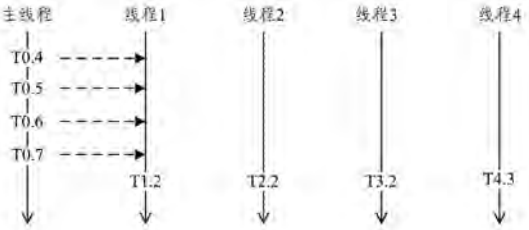


图 7　示例程序时序图

本文用 T$n.m$ 的形式描述语句的位置,表示线程 $n$ 中编号为 $m$ 的语句。

表 1　示例程序重要时序点对应事件

| 时序点 | 事件 | 时序点 | 事件 |
|---|---|---|---|
| T0.4 | 创建线程 1 | T1.2 | 写变量 $i$ |
| T0.5 | 创建线程 2 | T2.2 | 写变量 $i$ |
| T0.6 | 创建线程 3 | T3.2 | 写变量 $j$ |
| T0.7 | 创建线程 4 | T4.3 | 写变量 $j$ |

静态分析的结果表明,在示例程序中存在两组潜在数据竞争点对,它们分别为(T1.2,T2.2)和(T3.2,T4.3)。

本方法将静态分析的结果(潜在的数据竞争位置点对)应用于基于 IT2 FHMM 的方法对数据竞争进行分析的过程中。

### 3.3　设计实验产生训练数据

在本步骤中,首先需要明确 IT2 FHMM 模型训练需要的是什么数据,进而设计实验。

在多线程平台上,线程的运行情况是不可预见的,影响多线程程序的环境因素包括内部影响因素和外部影响因素。内部影响因素是指与多线程程序本身相关的因素,如线程中的时序控制语句等这类线程内部控制代码语句;外部影响因素是指与线程运行时环境相关的因素,比如操作系统负载状况。环境因素综合影响多线程程序的时序,本文以主要外部影响因素即系统的运行负载作为影响线程时序的环境因子。由于系统的负载无法直接描述,本文使用 CPU 使用率 $C(0\leqslant C\leqslant 100\%)$来描述系统的运行负载。

在 2.1 节中提到了可以利用隐马尔科夫模型解决的 3 种典型问题,对于第(2)种典型问题,由于在数据竞争问题中点对之间的时序关系比同一线程中不同时序点的次序重要,因此在已知模型参数之后,通常不会用 HMM 来计算某一观察值输出序列对应的隐含状态序列,而是计算某一隐含状态发

生的概率。或者说，HMM 用于数据竞争检测的有意义用法是：某一特定观察值输出序列和隐含状态序列的长度为 1。对于 IT2 FHMM，其训练数据应该是一段观察值序列，在数据竞争的应用中就是对示例程序插桩之后，插桩代码所反映的(T1.2，T2.2)和(T3.2，T4.3)的时序信息。然而，由于数据竞争应用的特殊性，可以使用一个统计量作为训练数据，即(T1.2，T2.2)和(T3.2，T4.3)中某一时序序列所占的比例，该比例近似等于某一特定运行时环境下点对的时序的概率。使用该统计量作为训练数据的优势是可以解决训练数据过多时模型训练的收敛时间会变得很长的问题，并且可以解决训练数据中的噪声问题。实验表明，这种做法可以有效缩短模型的训练时间，优化模型训练的效果。

明确了模型训练所需的数据之后，采用如下步骤得到训练数据。示例程序实验的系统环境为 64 位 Windows 7 操作系统，Intel Core i5-2430M(双核，2.4GHz)。

(1)在 3.1 节的示例程序中插桩代码，将每次运行之后变量 $i$ 和变量 $j$ 的值输出到文件。

(2)重复运行上述插桩程序 1000 次，得到当前系统负载情况下点对(T1.2，T2.2)和点对(T3.2，T4.3)各自的时序概率。

(3)后台运行 CPU 使用率控制工具，模拟不同的系统负载情况，在不同的 CPU 使用率下重复步骤(1)和步骤(2)。

(4)整理实验结果数据，将其组织为模型训练要求的数据格式。

### 3.4 构建 IT2 FHMM 模型

IT2 FHMM 可表示为 $\tilde{\lambda}=\{\tilde{S},\tilde{q}_t,\tilde{a}_{ij},\tilde{b}_j(o_t),c_{jm},\mu_{jm},\Sigma_{jm}\}$，本步骤中需要求解模型参数。在示例程序中，由于点对(T1.2，T2.2)与点对(T3.2，T4.3)所对应的 IT2 FHMM 是相似的，模型的建立过程完全相同，因此本节以(T1.2，T2.2)点对为例来演示模型的参数求解过程。

在训练模型之前，需要确定以下参数：

1)在示例程序中，点对(T1.2，T2.2)对应的 IT2 FHMM 的状态空间 $\tilde{S}=\{\tilde{q}_1,\tilde{q}_2\}$，状态数为 2，$\tilde{q}_1$ 代表时序关系 T1.2→T2.2，$\tilde{q}_2$ 代表时序关系 T2.2→T1.2。

2)$\tilde{a}_{ij}$ 表示状态转移概率矩阵，$\tilde{a}_{ij}=[\underline{a}_{ij},\bar{a}_{ij}]$。根据先验知识，取 $\underline{a}_{ij}=a_{ij}*0.9$，$\bar{a}_{ij}=a_{ij}*1.1$。$\tilde{b}_j(o_t)$ 反映隐含状态为 $\tilde{q}_t$ 时观察值的概率分布。$\tilde{b}_j(o_t)$ 用具有不确定均值 $\sigma=[\underline{\sigma},\bar{\sigma}]$ 的高斯主隶属函数来表示，其中 $\underline{\sigma}=k_1*\sigma$，$\bar{\sigma}=k_2*\sigma$，$k_1\in(0,1]$，$k_2\in[1,\infty)$。根据先验知识，取 $k_1=0.90$，$k_2=1.05$。$Ms$ 表示每个状态所对应的高斯混合模型中高斯分量的个数，根据先验知识，取 $Ms$ 的值为 2，这是模型复杂度和模型效果互相妥协的结果。

3)每个状态下，高斯混合模型的建立还需要求解相应的参数 $c_{jm}$，$\mu_{jm}$，$\Sigma_{jm}$，求解这些参数可以使用 IT2 FHMM 相应的 Baum-Welch 算法，根据 3.3 节中的实验结果进行重估。

确定以上参数后，可以采用 IT2 FHMM 相关的算法进行模型初始化和训练的工作，步骤设计如下：首先，使用基于 IT2 FHMM 的 Viterbi 算法对 IT2 FHMM 的未确定参数进行初始化；其次，使用基于 IT2 FHMM 的 forward-backward 算法计算得到前向因子和后向因子；最后，使用 IT2 FHMM 相应的 Baum-Welch 算法来进行模型参数训练，重估模型参数，得到本例对应的 IT2 FHMM。

在模型参数重估时会遇到二型模糊集降型的问题，需要将结果从区间值转化为确定的值或概率。二型模糊集的降型问题主要有 3 种解决方法：取区间上界值，取区间下界值，取区间的平均值。本例采用取算术平均值的方法来降型，因为训练数据中可能存在噪音，而算术平均值对训练数据中的噪音影响具有很好的鲁棒性。

本方法所采用的建模过程并非单一地对待检测软件建模，而是对软件和其运行的环境这一整体建模，因此某次建模结果只适用于待检测程序以及其当时的运行环境，当环境改变或针对不同的待检测程序时，需要重新获取训练数据来建立模型。

### 3.5 基于 IT2 FHMM 进行时序分析

根据训练后的模型，采用基于 IT2 FHMM 的 Viterbi 算法进行推理，从而获得在不同系统负载下的时序概率。本文同时还引入基于一型模糊逻辑的隐马尔科夫模型(Type-1 Fuzzy Hidden Markov Model，T1 FHMM)与 IT2 FHMM 进行对比，以检验 IT2 FHMM 用于时序分析的效果。实验结果如表 2 和表 3 所列。

表 2 点对(T1.2，T2.2)在 T1 FHMM 和 IT2 FHMM 中的时序分析结果

| CPU 使用率/% | 实验结果 | T1 FHMM | IT2 FHMM |
|---|---|---|---|
| 3 | 0.6700 | 0.5478 | 0.6612 |
| 16 | 0.5669 | 0.5478 | 0.5739 |
| 24 | 0.4827 | 0.5477 | 0.5051 |
| 35 | 0.5475 | 0.5477 | 0.5261 |
| 42 | 0.5268 | 0.5840 | 0.4941 |
| 52 | 0.4974 | 0.8385 | 0.4800 |
| 64 | 0.4913 | 0.4571 | 0.5120 |
| 73 | 0.5666 | 0.4521 | 0.5682 |
| 85 | 0.5183 | 0.4521 | 0.6682 |

表 3 点对(T3.2，T4.3)在 T1 FHMM 和 IT2 FHMM 中的时序分析结果

| CPU 使用率/% | 实验结果 | T1 FHMM | IT2 FHMM |
|---|---|---|---|
| 3 | 0.9851 | 0.5879 | 0.9548 |
| 16 | 0.9457 | 0.5834 | 0.9546 |
| 24 | 0.8732 | 0.5788 | 0.9332 |
| 35 | 0.9089 | 0.5720 | 0.9189 |
| 42 | 0.7892 | 0.5597 | 0.8660 |
| 52 | 0.8282 | 0.5479 | 0.7915 |
| 64 | 0.8452 | 0.5473 | 0.7046 |
| 73 | 0.7320 | 0.3340 | 0.6273 |
| 85 | 0.7474 | 0.2240 | 0.5711 |

本例中，选取了 9 组 CPU 使用率下点对的时序进行考查。表 2 为点对(T1.2，T2.2)的时序分析结果，表 3 为点对(T3.2，T4.3)的时序分析结果。表中第一列为 CPU 使用率；第二列为实际多次运行插桩示例程序而得到的在实际情况下点对在不同系统负载下的隐含状态 $q_1$，即 T1.2→T2.2 发生的概率；第三列为 T1 FHMM 计算出的隐含状态 $q_1$ 发生的概率；第四列为 IT2 FHMM 计算出的隐含状态 $q_1$ 发生的概率。

将表 2 和表 3 中的数据在坐标轴上描绘出来，可以得到如图 8 和图 9 所示的结果。

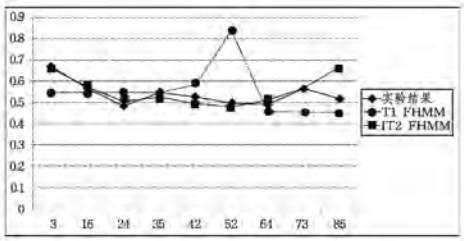


图 8　点对(T1.2,T2.2)的时序结果图

图 8 对应点对(T1.2,T2.2)的实验结果。从图中可以看出,对于点对(T1.2,T2.2),在不同的系统负载条件下,隐含状态 $q_1$ 发生的概率在 60%左右浮动;相比于 T1 FHMM 的结果,IT2 FHMM 计算出的时序概率更好地吻合了 $q_1$ 实际发生的概率,而 T1 FHMM 虽然可以大致与实际情况吻合,但由于模型的模糊程度不够,时序概率计算中在某一位置出现了较大偏差。

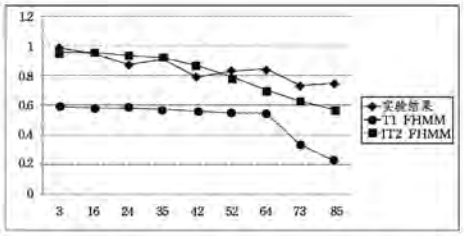


图 9　点对(T3.2,T4.3)的时序结果图

图 9 对应点对(T3.2,T4.3)的实验结果。由于时序位置点 T4.3 之前有一段延迟代码,导致在不同的系统负载下,隐含状态 $q_1$(即 T3.2→T4.3)的概率高于隐含状态 $q_1$(即 T4.3→T3.2)的概率。在该时序的计算中,IT2 FHMM 计算的时序结果比 T1 FHMM 有更好的表现。

图 8 和图 9 直观地表现出了 IT2 FHMM 用于多线程程序时序分析时具有令人满意的效果。我们也可采用量化的手段对时序分析效果进行评估。

记 $P$ 为实际情况中状态 $q_1$ 发生的概率,$P_1$ 为 T1 FHMM 计算得到的相应系统环境下状态 $q_1$ 发生的概率,$P_2$ 为 IT2 FHMM 计算得到的相应系统环境下状态 $q_1$ 发生的概率。本文实验结果是大量运行示例程序得到的状态 $q_1$ 发生的概率,可以近似当作实际情况中 $P$ 的值。记 $D_i=|P-P_i|$ 为模型计算结果与实际值之间的距离(偏差值),在比较不同建模方法的计算结果与实际值的差距时,可以采用 $\bar{D}_i=(\sum_{i=1}^{n}|P-P_i|)/n$ 进行计算。在上述两例中,两种建模方法的计算结果与实际值的距离如表 4 和表 5 所列。

表 4　点对(T1.2,T2.2)的建模结果与实际值的距离

| CPU 使用率/% | $D_1$ | $D_2$ |
|---|---|---|
| 3 | 0.1222 | 0.0088 |
| 16 | 0.0191 | 0.0070 |
| 24 | 0.0650 | 0.0224 |
| 35 | 0.0002 | 0.0214 |
| 42 | 0.0572 | 0.0327 |
| 52 | 0.3411 | 0.0174 |
| 64 | 0.0342 | 0.0207 |
| 73 | 0.1145 | 0.0016 |
| 85 | 0.0662 | 0.1499 |

表 5　点对(T1.2,T2.2)的建模结果与实际值的距离

| CPU 使用率/% | $D_1$ | $D_2$ |
|---|---|---|
| 3 | 0.3972 | 0.0303 |
| 16 | 0.3623 | 0.0089 |
| 24 | 0.2944 | 0.0600 |
| 35 | 0.3369 | 0.0100 |
| 42 | 0.2295 | 0.0768 |
| 52 | 0.2803 | 0.0367 |
| 64 | 0.2979 | 0.1406 |
| 73 | 0.3980 | 0.1047 |
| 85 | 0.5234 | 0.1763 |

对于点对(T1.2,T2.2)的分析结果,$\bar{D}_1=0.0911$,$\bar{D}_2=0.0313$;对于点对(T3.2,T4.3)的分析结果,$\bar{D}_1=0.3467$,$\bar{D}_2=0.0716$。以上两组数据反映出 IT2 FHMM 的时序计算结果更接近于真实值,并且 IT2 FHMM 用于多线程程序的时序分析可以得到更为理想的结果。

### 3.6　后处理:评估数据竞争概率及生成优先级

在描述数据竞争发生的概率时,本文采用数据竞争确定性[41]这一概念作为数据竞争发生概率的评价标准。在示例中,记 $P_a$ 为 T1.2→T2.2 发生的概率,$P_b$ 为 T2.2→T1.2 发生的概率,$P_c$ 为 T3.2→T4.2 发生的概率,$P_d$ 为 T4.2→T3.2 发生的概率。点对(T1.2,T2.2)的数据竞争确定性 $C_1=1-|P_a-P_b|=1-|P_a-(1-P_a)|=1-|2P_a-1|$;同理,点对(T3.2,T4.3)的数据竞争确定性 $C_2=1-|P_c-P_d|=1-|2P_c-1|$。

本文以 IT2 FHMM 计算出的时序概率作为某一时序状态的概率,考虑到模型训练数据中存在噪音的影响,选取时序概率的中值作为最终时序发生的概率。因此,上文中点对(T1.2,T2.2)对应的 $P_a=0.5261$,$P_b=0.4739$,点对(T3.2,T4.3)对应的 $P_c=0.8660$,$P_d=0.1340$。可以求出两组潜在数据竞争位置点对发生时序竞争的确定性分别为 $C_1=0.9478$,$C_2=0.268$。由此得出结论,第一组的潜在数据竞争位置点对处在当前系统平台下发生数据竞争的确定性远高于第二组潜在数据竞争位置点对。

数据竞争确定性反映出当前系统环境下数据竞争发生的可能性,可以直接将其作为处理数据竞争缺陷的优先级提供给软件工程师作为处理数据竞争缺陷的参考依据。

### 3.7　工具实现

我们还实现了静态数据竞争检测预处理工具、CPU 使用率控制工具、IT2 FHMM 和 T1 FHMM 时序分析工具,用于支持本文方法。

静态数据竞争检测预处理工具基于以对象为中心的数据竞争检测方法,输出待检测程序中潜在的数据竞争位置点,其运行界面如图 10 所示。

图 10　静态数据竞争检测预处理工具界面

CPU 使用率控制工具调用 SetProcessAffinityMask 方法，在后台通过控制 CPU 闲时和忙时的比例实现对 CPU 使用率的调节。

IT2 FHMM 和 T1 FHMM 时序分析工具将模型训练数据作为输入，采用 Viterbi 算法对 IT2 FHMM 和 T1 FHMM 模型参数进行初始化，相比于随机初始化模型参数的方法，采用 Viterbi 算法进行初始化可以加快模型训练阶段的收敛速度。该工具还实现了基于 IT2 FHMM 和 T1 FHMM 的 Baum-Welch 算法和 Viterbi 算法，并采用取算术平均值的方法对二型模糊集降型。模型训练结束后，可以根据 Viterbi 算法计算出在某一 CPU 使用率下的隐含状态序列和隐含状态发生的概率，直观地展示 IT2 FHMM 和 T1 FHMM 计算结果的差异，验证这两个模型应用于时序分析的效果。其运行界面如图 11 所示。

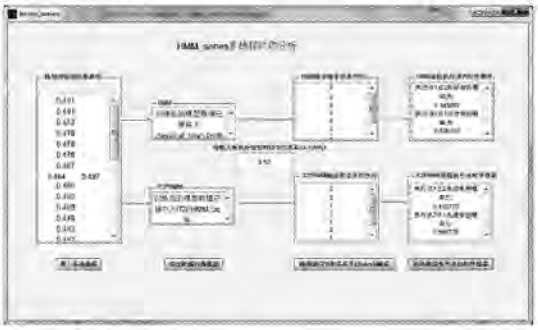

图 11 IT2 FHMM 和 T1 FHMM 时序分析工具

## 4 总结与进一步工作

### 4.1 总结

本文提出将基于二型模糊逻辑的隐马尔科夫模型应用于多线程程序时序分析中的方法，为多线程程序的时序分析提供了一种新的思路，同时将其与传统的数据竞争检测方法结合后作为传统检测方法的后端，可以有效提高传统检测方法的精确度。本方法最大的优点在于考虑了系统环境对线程时序的影响，分析了时序发生的概率，并基于时序概率计算了缺陷发生的优先级。本文以数据竞争检测为例，详细介绍了将其应用于实际应用领域的过程：首先采用传统静态分析工具扫描待检测程序，获取其中的潜在数据竞争位置；其次，对待检测程序插桩并多次运行，获取其在当前环境中的时序概率，并将取得的概率作为 IT2 FHMM 的训练数据；然后，采用基于 IT2 FHMM 的 Baum-Welch 算法进行模型训练；最后，利用基于 IT2 FHMM 的 Viterbi 算法进行时序分析，并根据计算得到的时序概率生成数据竞争的优先级列表。

### 4.2 进一步工作

(1)本方法依赖传统数据竞争静态分析方法对待检测程序作预处理，预处理结果的好坏直接影响本方法的结果。本文在设计过程中对预处理部分的检测方法还需要进一步完善。

(2)影响多线程程序时序的环境因素很多，包括内部因素和外部因素。本文以 CPU 使用率为例，在进一步的工作中还将考虑其他因素。

(3)在多核平台上运行的多线程程序，其时序实际有 3 种状态：a 先于 b，a 后于 b，a 和 b 真并行(real-parallel)。虽然第三种状态发生的可能性不大，但进一步的工作中也要将其纳入考虑的范围。

(4)在实际应用环境中，待检测多线程程序中线程的数量可能非常多，潜在并行缺陷的位置也会很多，模型训练和时序分析将成为系统效率的短板，未来将研究模型优化算法，使本方法适用于实际大规模多线程程序。

## 参 考 文 献


[1] KIM K, YAVUZ K T, SANDERS B. Precise Data Race Detection in a Relaxed Memory Model Using Heuristic-Based Model Checking[C]// Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE2009). IEEE Computer Society, 2009: 495-499.

[2] AGARWAL R, STOLLER S. Run-Time Detection of Potential Deadlocks for Programs with Locks, Semaphores, and Condition Variables[C]// Proceedings of the 4th Workshop on Parallel and Distributed Systems: Testing and Debugging (PADTAD2006). ACM, 2006: 51-60.

[3] JOSHI P, NAIK M, SEN K, et al. An Effective Dynamic Analysis for Detecting Generalized Deadlocks[C]// Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE2010). ACM, 2010: 327-336.

[4] HENZINGER T, JHALA R, MAJUMDAR R. Race Checking by Context Inference[C]// Proceedings of the 25th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2004). ACM, 2004: 1-13.

[5] FLANAGAN C, LEINO K, LILLBRIDGE M, et al. Extended Static Checking for Java[C]// Proceedings of the 23rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2002). ACM, 2002: 234-245.

[6] NAIK M, PARK C S, SEN K, et al. Effective Static Deadlock Detection[C]// Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE2009). IEEE Computer Society, 2009: 386-396.

[7] MATSAKIS N, GROSS T. A Time-Aware Type System for Data-Race Protection and Guaranteed Initialization[C]// Proceedings of the 25th ACM International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA2010), 2010. ACM, 2010: 634-651.

[8] KAHOLN V, YANG Y, SANKARANARAYANAN S, et al. Fast and Accurate Static Data-Race Detection for Concurrent Programs[C]// Proceedings of the 19th International Conference on Computer Aided Verification (CAV2007), 2007. Heidelberg: Springer Berlin, 2007: 226-239.

[9] VOUNG J, JHALA R, LERNER S. RELAY: Static Race Detection on Millions of Lines of Code[C]// Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2007. ACM, 2007: 205-214.

[10] PRATIKAKIS P, FOSTER J, HICKS M. Locksmith: Practical Static Race Detection for C[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 2011, 33(1): 1-55.

[11] TERAUCHI T. Checking Race Freedom via Linear Programming[C]// Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2008). ACM, 2008: 1-10.

[12] KAHLON V, SINHA N, KRUUS E, et al. Static Data Race Detection for Concurrent Programs with Asynchronous Calls[C]//

Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009. ACM, 2009: 13-22.

[13] CHUGH R, VOUNG J, JHALA R, et al. Dataflow Analysis for Concurrent Programs via Datarace Detection[C]// Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2008). ACM, 2008: 316-326.

[14] MARINO D, MUSUVATHI M, NARAYANSAMY S. LiteRace: Effective Sampling for Lightweight Data-Race Detection [C]// Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2009). ACM, 2009: 134-143.

[15] FLANAGAN C, FREUND S. Atomizer: A Dynamic Atomicity Checker For Multithreaded Programs[J]. Science of Computer Programming, 2008, 71(2): 89-109.

[16] POZNIANSKY E, SCHUSTER A. MultiRace: Efficient On-the-Fly Data Race Detection in Multithreaded C++ Programs[J]. Concurrency and Computation: Practice and Experience, 2007, 19(3): 327-340.

[17] ELMAS T, QADEER S, TASIRAN S. Goldilocks: A Race and Transaction-Aware Java Runtime[C]// Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2007). ACM, 2007: 245-255.

[18] FLANAGAN C, FREUND S. FastTrack: Efficient and Precise Dynamic Race Detection[C]// Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2009). ACM, 2009: 121-133.

[19] BOND M, COONS K, MCKINLEY K. PACER: Proportional Detection of Data Races[C]// Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2010). ACM, 2010: 255-268.

[20] TIAN C, NAGARAJAN V, GUPTA R, et al. Dynamic recognition of synchronization operations for improved data race detection[C]// Proceedings of the 2008 International Symposium on Software Testing and Analysis, 2008. ACM, 2008: 143-154.

[21] LI D, SRISA-AN W, DWYER M. SOS: Saving Time in Dynamic Race Detection with Stationary Analysis[C]// Proceedings of the 26th ACM International Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA 2011). ACM, 2011: 35-50.

[22] CHOI J D, LEE K, LOGINOV A, et al. Efficient and Precise Datarace Detection for Multithreaded Object-Oriented Programs [C]// Proceedings of the 23rd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2002). ACM, 2002: 258-269.

[23] NISHIYAMA H. Detecting Data Races using Dynamic Escape Analysis based on Read Barrier[C]// Proceedings of the 3rd Virtual Machine Research and Technology Symposium, 2004. USENIX Association, 2004: 10-10.

[24] JOSHI P, NAIK M, PARK C S, et al. CalFuzzer: An Extensible Active Testing Framework for Concurrent Programs[C]// Proceedings of the 21st International Conference on Computer Aided Verification (CAV2009). Heidelberg: Springer Berlin, 2009: 675-681.

[25] JOSHI P, PARK C S, SEN K, et al. A Randomized Dynamic Program Analysis Technique for Detecting Real Deadlocks[C]// Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2009). ACM, 2009: 110-120.

[26] SEN K. Race Directed Random Testing of Concurrent Programs [C]// Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2008). ACM, 2008: 11-21.

[27] MUSUVATHI M, QADEER S, BALL T, et al. Finding and Reproducing Heisenbugs in Concurrent Programs[C]// Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI 2008). USENIX Association, 2008: 267-280.

[28] CHEN F, SERBANUTA T, ROSU G. jPredictor: A Predictive Runtime Analysis Tool for Java[C]// Proceedings of the 30th International Conference on Software Engineering (ICSE2008). ACM, 2008: 221-230.

[29] SMARAGDAKIS Y, EVANS J, SADOWSKI C, et al. Sound Predictive Race Detection in Polynomial Time[C]// Proceedings of the 39th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL2012). ACM, 2012: 387-400.

[30] BODDEN E, HAVELUND K. Aspect-Oriented Race Detection in Java[J]. IEEE Transactions on Software Engineering, 2010, 36(4): 509-527.

[31] ZENG J, XIE L, LIU Z Q. Type-2 fuzzy Gaussian Mixture Models[J]. Pattern Recognition, 2008, 41(12): 3636-3643.

[32] ZENG J, LIU Z Q. Type-2 Fuzzy Markov Random Fields and Their Application to Handwritten Chinese Character Recognition[J]. IEEE Transactions on Fuzzy Systems, 2008, 16(3): 747-760.

[33] KONG D G, TAN X B, XI H S, et al. Hidden Markov Model for Multi-Thread Programs Time Sequence Analysis[J]. Journal of Software, 2010, 21(3): 461-472. (in Chinese)
孔德光,谭小彬,奚宏生,等. 多线程程序时序分析的隐 Markov 模型[J]. 软件学报, 2010, 21(3): 461-472.

[34] WENG D, YANG L, LIU Q, et al. Type-2 Fuzzy Logic Based Deadlock Detection[J]. International Journal of Digital Content Technology and its Applications, 2012, 6(1): 429-438.

[35] STAMP M. A revealing introduction to hidden markov models [J]. Leee Assp Magruine, 2004, 1(24): 258-261.

[36] KLIR G, CLAIR U, YUAN B. Fuzzy Set Theory: Foundations and Applications[M]. Prentice Hall, 1997.

[37] MENDEL J, JHON R, LIU F. Interval Type-2 Fuzzy Logic Systems Made Simple[J]. IEEE Transactions on Fuzzy Systems, 2006, 14(6): 808-821.

[38] MENDEL J. Advances in Type-2 Fuzzy Sets and Systems[J]. Information Sciences, 2007, 177(1): 84-110.

[39] WU D R, MENDEL J. Uncertainty Measures for Interval Type-2 Fuzzy Sets[J]. Information Sciences, 2007, 177(23): 5378-5393.

[40] MENDEL J. Type-2 Fuzzy Sets and Systems: an Overview[J]. IEEE Computational Intelligence Magazine, 2007, 2(1): 20-29.

[41] WENG D L. Research for Type-2 Fuzzy Logic Based Deadlock and Data Race Detection[D]. Shuzhou: Soochow University, 2012. (in Chinese)
翁东良. 基于二型模糊逻辑的死锁与数据竞争检测方法研究[D]. 苏州:苏州大学, 2012.