计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
49(10): 2054－2065, 2012

实时复杂事件处理的最坏响应时间估算

李 想[1] 范玉顺[1] 王宏安[2] 乔 颖[2]
[1](清华大学自动化系 北京 100084)
[2](中国科学院软件研究所 北京 100190)
(cqlixiang@gmail.com)

Estimation on Worst-Case Execution Time of Real-Time Complex Event Processing

Li Xiang[1], Fan Yushun[1], Wang Hongan[2], and Qiao Ying[2]
[1](*Department of Automation, Tsinghua University, Beijing* 100084)
[2](*Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract Real-time complex event processing (CEP) system is used to detect complex events from primitive event stream and must guarantee that the tasks of processing events can be completed in deadline. In order to guarantee that, a key problem is how to estimate the worst-case execution time (WCET) of the CEP program in a CEP system. In current WCET estimation methods for general programs, the range of the execution number of each sub-program needs to be annotated by developers. In a CEP program, however, ranges of execution numbers of sub-programs for detection of sub-event patterns are hard to directly obtain because of the complexity of CEP program. Although execution numbers of different sub-programs have relations and ranges can be solved from these relations, these ranges are still not strict enough, which will reduce the estimation accuracy. Thus current methods cannot accurately estimate the WCET of CEP programs. This paper presents a novel WCET estimation method for CEP program. In face of annotation difficulties, constraints among execution numbers of sub-programs are annotated, instead of ranges of these execution numbers. The constraints are generated from detection structures used by the CEP program. Results of simulations indicate that the method is effective and has higher accuracy.

Key words complex event processing; real-time; worst-case execution time; annotation; event stream

摘 要 实时复杂事件处理系统(CEP系统)用于从原子事件流中检测出复杂事件,需要确保事件处理任务在截止期内完成.确保实时性的关键问题是如何估算系统中复杂事件处理程序(CEP程序)的最坏响应时间.现有针对一般程序的估算方法需要标注对象程序中子程序执行次数的取值范围.然而,CEP程序较为复杂,难以直接获知子程序执行次数的取值范围.虽然执行次数间存在关联关系,可以间接求解出取值范围,但这样得到取值范围不够严格,使估算精度较低,因此现有估算方法难以直接使用.提出一种CEP程序的最坏响应时间估算方法.采用新标注方式,通过对CEP程序的检测结构进行分析,归纳出子程序执行次数间的关联约束,并使用关联约束进行标注,替代了标注其取值范围,避免了标注困难.实验表明方法具有较高估算精度.

关键词 复杂事件处理;实时;最坏响应时间;标注;事件流

中图法分类号 TP302.7

收稿日期:2012-06-05;**修回日期**:2012-07-26

基金项目:国家自然科学基金项目(61033005,61174169);国家"八六三"高技术研究发展计划基金项目(2012AA040915)

复杂事件处理系统(complex event processing,简称 CEP 系统)被用于从大量低层次事件(原子事件)中按照一定模式匹配得出"有意义的"高层次事件(复杂事件)[1],例如,使用 CEP 系统监控火灾,通过处理温度超标、烟雾浓度超标等事件可以判断火灾是否发生. CEP 的概念自从提出以来逐渐成为数据库等领域的研究热点.

现实中存在一类应用场景,任务需要在一定截止期内完成. 这类应用场景称为时间关键的应用场景,能够确保任务满足截止期要求的系统是实时系统[2]. 例如,国家标准规定,火灾数据处理时间不得超过 10 s[3],因此,用于时间关键应用场景中的实时 CEP 系统需要确保事件的处理能够在截止期内完成.

Sybase Aleri[4],Esper[5]等商业 CEP 系统将实时性作为重要特性,文献[6-7]等研究也指出实时性对 CEP 系统的重要性,但它们都更强调"快速"或"高性能". 然而,文献[2]指出,"快速"或"高性能"并不等于"实时". 例如,假设 S_1,S_2 两个 CEP 系统平均处理能力分别为 1 000 事件/ms 及 20 事件/ms,S_1 处理能力下限未知,S_2 下限为 10 事件/ms. 多数情况下处理同一个任务 S_1 性能远优于 S_2. 但由于处理能力下限未知,面对任意截止期,S_1 均有可能错过截止期;而 S_2 在截止期不小于 N/10 ms 时(N 为需处理事件数量)能确保任务的完成. 因此,S_2 是实时系统,S_1 不是.

确保实时性的关键问题是,如何估算 CEP 系统中负责从原子事件流中检测复杂事件的程序(简称 CEP 程序)的最坏响应时间,并将其与应用要求的截止期进行比较. 程序一次执行的响应时间定义为在无抢占情况下开始执行与结束执行的时间间隔. 最坏响应时间为在所有可能情况下响应时间的最大值.

在实时 CEP 研究领域,文献[8-9]等文献从复杂事件实时描述语言、实时 CEP 框架等方面进行了研究,其中文献[8]对最坏响应时间估算问题有所提及. 但据我们的了解,现有研究均未提出最坏响应时间估算方法.

在实时理论方面,研究者提出了许多针对一般程序的估算方法,涉及到源代码分析[10]、机器码分析[11]、硬件模型分析[12]等多方面. 这些方法在实际使用中,需要开发人员标注子程序的执行次数取值范围[13],标注的精确性将直接影响估算精度. 然而,CEP 程序非常复杂,上述取值范围难以直接获知. 虽然可以通过子程序执行次数间的关联关系解出其取值范围,但这样解出的取值范围不够严格,将降低估算精度. 因此,现有的估算方法难以直接使用.

对 CEP 程序的最坏响应时间估算问题,本文首次提出了一种估算方法. 面对标注困难改进了标注方式,通过分析程序的复杂事件检测结构,归纳出子程序执行次数间的关联约束进行标注,替代标注其取值范围. 实验表明本方法具有较高估算精度.

1　复杂事件处理

事件是应用场景中用户感兴趣的状态变化. 下面用 E 和 e 分别表示事件类型和事件实例. 事件分为原子事件和复杂事件. 原子事件指从应用场景中直接获得的事件;复杂事件指通过操作符结合原子事件、复杂事件,形成其他"有意义"的事件[14]. 作为例子,下面介绍 3 种常用操作符[6,14-15].

1) And:$E=And(E_1,E_2,\cdots,E_n)$,当所有成员事件 $E_1,E_2,\cdots,E_n$ 发生时 E 才发生;

2) Or:$E=Or(E_1,E_2,\cdots,E_n)$,只要有一个成员事件发生时 E 就发生;

3) Seq:$E=Seq(E_1,E_2,\cdots,E_n)$,成员事件需要按照顺序发生 E 才发生.

现有的 CEP 检测算法分为 3 类:①基于有限状态自动机的方法[16];②基于 Petri 网的方法[17];③基于有向图的方法[15]. 下面简介基于有向图的方法,以文献[15]提出的方法为例.

首先,将复杂事件模式集映射为有向图,方法如下:复杂事件模式中原子事件或复杂事件映射为节点,复杂事件节点与其成员事件节点通过有向边连接. 如果集合包含多个复杂事件模式,则合并多个有向图中相同节点. 如图 1 所示[15],$Seq(Seq(And(E_1,E_2),E_3),And(E_2,E_4))$映射为有向图.

通过有向图结构检测复杂事件,其过程如下:当原子事件发生时,其实例被相应的原子事件节点送往父节点. 复杂事件节点接收到实例,调用对应子程序(称为节点子程序)处理该实例,包括丢弃、储存等. 当复杂事件节点检测到子节点实例与其对应的模式相匹配时,判断复杂事件发生,产生一个新实例送往父节点. 如果出口节点产生实例,说明对应的复杂事件被检测到. 如图 1 所示,当事件 E_4 发生后,实例 e_4^1 被节点 E_4 送往节点 C. 因为储存有 E_2 的实例 e_2^2,节点 C 判断 $And(E_2,E_4)$事件发生,产生新实例 $And(e_2^2,e_4^1)$送往节点 A. 因为储存有实例 $Seq(And(e_1^2,$

$e_2^2)$, $e_3^1)$, 节点 A 判断复杂事件发生, 输出实例 $Seq(Seq(And(e_1^2, e_2^2), e_3^1), And(e_2^2, e_4^1))$[15].

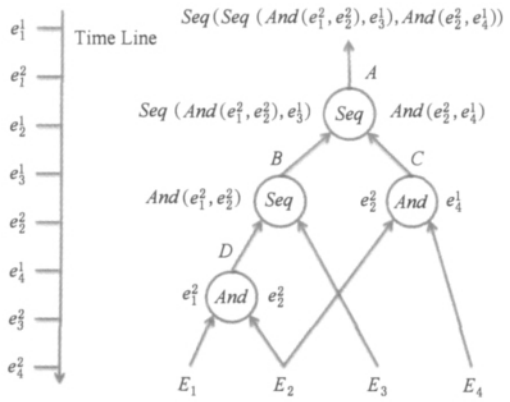


Fig. 1 Directed graph for detecting $Seq(Seq(And(E_1, E_2), E_3), And(E_2, E_4))$[15].

图 1 检测 $Seq(Seq(And(E_1, E_2), E_3), And(E_2, E_4))$的有向图[15]

2 CEP 程序最坏响应时间估算问题

CEP 的概念模型如图 2 所示. 在工业生产、商业交易等应用中, RFID、无线传感器网络等设备监测到的数据流, 经过清洗筛选生成原子事件实例流. CEP 系统执行 CEP 程序, 根据预定义的复杂事件模式, 在原子事件实例流中检测出复杂事件.

Fig. 2 Conceptual model of CEP.

图 2 复杂事件处理概念模型

考虑上述模型, CEP 程序的最坏响应时间定义如下: 给定硬件环境, 并给定复杂事件模式集, 当 CEP 程序 p 第 i 次执行时, 待处理原子事件实例序列 ES_i 包含 N_i 个原子事件实例, 即 $|ES_i| = N_i$. 假设执行过程中无抢占, 处理 ES_i 的开始时间和结束时间分别为 r_p^i 和 f_p^i. p 第 i 次执行的响应时间为 $R_p^i = f_p^i - r_p^i$. 考虑所有可能产生的原子事件实例序列, 响应时间最大值即为最坏响应时间, 即 $W_p = \max_i(R_p^i)$.

为了使估算结果可用于判断截止期是否被满足, 估算方法需要确保安全性. 设一种估算方法 M 的估算值为 W_p^M, 当满足 $W_p^M \geqslant W_p$ 时, M 具有安全性.

问题 1. CEP 程序的最坏响应时间估算问题为如何安全地精确地估算 CEP 程序的最坏响应时间.

应用所需的复杂事件模式集合及应用中可能产生的原子事件实例序列这两个因素影响 CEP 程序的最坏响应时间. 这两个因素取决于 CEP 系统所处的应用场景. 为了估算 CEP 程序的最坏响应时间, 本文假设: 1) 复杂事件模式集事先已预定义; 2) 原子事件实例序列长度(即包含的实例个数)有明确上限. 满足上述两项假设的应用场景广泛存在. 首先, 在许多场景中, 需要检测的事件模式相对固定, 在 CEP 系统运行前能预定义, 假设 1) 满足. 其次, 许多场景中, 判断复杂事件只需要考察一定时间内发生的原子事件实例. 因此, 许多实用 CEP 系统引入滑动窗口[6]. 考虑传感设备最小采样频率等因素, 滑动窗口内发生的原子事件实例数量上限可估算, 因此假设 2) 满足. 火灾监控、网络攻击监控等均属于此类应用场景. 本文在上述应用场景假设下展开研究, 同时, 也将讨论在其他应用场景假设下本文方法的适用性.

3 最坏响应时间估算相关工作

针对一般程序的最坏响应时间估算方法分为两类: 动态测量方法[18]与静态分析方法. 动态测量方法不具有安全性[13], 因此不能用于解决本文问题.

根据静态分析观点, 程序的最坏响应时间是在特定硬件环境下, 程序按照所有可能顺序执行语句所消耗时间的最大值[13]. 因此, 静态分析方法包含通过源代码分析寻找最坏语句执行路径[10]、通过硬件模型分析估算语句执行时间[12]以及通过机器码分析结合上述两个方面[11]等.

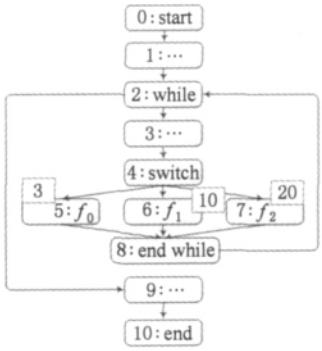
对于 CEP 程序, 硬件模型及机器码分析无特殊性, 现有分析方法可以适用, 不在本文讨论之列. 本文着重讨论源代码分析方法.

假设通过硬件分析已知每条语句执行时间, 源代码分析目标是寻找程序中耗时最多的语句执行序列.

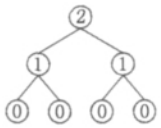
在现有方法中, 源代码首先需要映射为控制流图. 例如, 文献[10]介绍了一种映射方法: 源代码中无分支循环的相邻语句映射为"块"; 语句执行时间为块权值; 语句间控制流关系映射为"连接". 例如, 图 3(a)是一个树遍历程序, 它映射为控制流图, 如图 3(b)所示. 其中, 方形表示块, 块 5～7 的权值分别为子程序 f_0, f_1, f_2 的执行时间 3, 10, 20. 常见的控制流关系有 3 种: 顺序(如块 0～1)、分支(如块 4 与块 5～7)、循环(如块 2 与块 8 间所有块).

```
void traversal(Object ob) {
  Vector v = new Vector();
  Vector n = tree.getRoot();
  v.add(0, n);
  while(v.size != 0) {
    n = v.get(0);
    switch(n.type) {
    case 0: f0(ob); //texec = 3
    case 1: f1(ob); //texec = 10
    case 2: f2(ob); //texec = 20
    }
    for(i = n.childSize - 1; i >= 0; i--)
     v.addAall(0, n.getChileVertex(i))
  }
}
```

(a) A program for tree traversal



(b) The control-flow graph of the program



(c) An example of a tree

Fig. 3　An example for source codes analysis.

图 3　源代码分析示例

在控制流图中,耗时最多的语句执行序列对应于从入口块到出口块权值之和最大的路径.分析方法分为 3 类:1)结构合并方法[19];2)路径搜索方法[20];3)隐含路径计数方法,也称整数线性规划方法[10].

程序运行时耗时最多的语句执行序列与循环变量及子程序的执行次数的取值范围有关.而这些取值范围通过分析控制流图无法获得.因此,需要开发人员根据对象程序以及输入情况对其进行标注[21].例如,如图 3(b)所示的情况,需要标注"while"循环次数上限以及 f_0, f_1, f_2 的执行次数上限,这与"*tree*"变量所储存的树结构有关.如果"*tree*"储存如图 3(c)所示的树,则"while"循环次数上限为 7 次,f_0, f_1, f_2 的执行次数分别为 4,2,1 次.

下面讨论现有方法用于分析 CEP 程序存在的问题.以基于有向图的 CEP 程序为例,标注节点子程序执行次数的取值范围,需要分析有向图检测结构.然而,一方面,有向图结构的复杂性使直接获知上述取值范围存在困难;另一方面,按照后面的分析,节点子程序执行次数间存在关联约束关系,通过关联约束可解出其取值范围.然而,相较于关联约束本身,这样解得的取值范围将使变量的定义域扩大,无效值增加.按照现有估算方法要求标注取值范围,后果是估算精度降低.后面模拟实验将通过案例详细分析此问题.由于上述问题的存在,使得现有估算方法难以直接用于估算 CEP 程序的最坏响应时间.

4　最坏响应时间估算方法

针对 CEP 程序最坏响应时间估算问题,本节提出一种新估算方法,通过标注节点子程序执行次数关联关系取代标注其取值范围,避免了标注困难.

对于 CEP 程序,硬件模型及机器码分析无特殊性,现有分析方法可以适用,不在本文讨论之列.假设通过硬件模型及机器码分析,源代码中语句执行时间已知.下面着重讨论如何改进源代码分析方法.

首先,本节以基于有向图的 CEP 程序为例,通过分析有向图,归纳节点子程序执行次数间的关联关系;其次,讨论如何利用关联关系进行标注,改进现有源代码分析方法;最后,讨论本估算方法的适用性及时间复杂度.

4.1　CEP 程序有向图检测结构分析

定义 1. 在 CEP 程序的有向图检测结构中,用 v_i 表示第 i 个节点,px_i 表示 v_i 的节点子程序的执行次数,也称 v_i 的访问次数,x_i 表示 v_i 产生的新实例数量,V_p 表示原子事件节点集合,V_C 表示复杂事件节点集合,V_i 表示 v_i 的所有成员节点集合.

下面分析 x_i 与 px_i 的关联关系.

1) 根据第 2 节的应用场景假设,原子事件实例序列的长度上限已知(设为 N).由于每个原子事件实例由相应的原子事件节点转发,因此所有原子事件节点的访问次数之和不大于 N,称为输入约束,即

$$\sum_{v_i \in V_P} px_i \leqslant N.$$

如图 1 所示,有 $px_{E_1}+px_{E_2}+px_{E_3}+px_{E_4}\leqslant N$. 输入约束与应用场景的假设有关. 上述输入约束形式来源于假设原子事件实例序列长度上限已知. 对于其他应用场景假设,可能存在不同形式的输入约束. 例如,假设存在一种应用场景,可以预估输入的每种原子事件的个数上限. 此时输入约束具有不同形式. 比如,对图 1 所示的有向图,假设输入序列最多包含 5,3,7,1 个 E_1,E_2,E_3,E_4,可得输入约束:$px_{E_1}\leqslant 5, px_{E_2}\leqslant 3, px_{E_3}\leqslant 7, px_{E_4}\leqslant 1$.

2) 原子事件节点仅进行实例的转发,因此,产生的新实例数与接收到的实例数相同,称为原子事件节点约束,即

$$x_i=px_i, v_i\in V_P,$$

例如图 1 中,节点 E_1 有约束 $x_{E_1}=px_{E_1}$.

3) 每当复杂事件节点接到子节点送来的实例时,它会调用节点子程序处理该实例一次. 因此复杂事件节点的访问次数等于其所有子节点产生实例数量之和,称为复杂事件节点输入约束,即

$$px_i=\sum_{v_j\in V_i}x_j, v_i\in V_C,$$

例如,在图 1 中,节点 D 有约束 $px_D=x_{E_1}+x_{E_2}$.

4) 一次执行复杂事件节点子程序,最多只能检测到复杂事件的一次发生,即产生一个新实例. 同时,并非每次执行都产生新实例. 例如,在图 1 中,节点 D 处理实例 e_2^2 产生新实例 $And(e_1^2,e_2^2)$,而处理实例 e_1^2 未产生新实例. 因此节点产生的新实例数不大于节点访问次数,称为复杂事件节点输出约束:

$$x_i\leqslant px_i, v_i\in V_C,$$

例如,在图 1 中,节点 D 有约束 $x_D\leqslant px_D=x_{E_1}+x_{E_2}$.

然而,通过进一步分析发现,上述约束条件不够严格. 例如,图 1 中,节点 D 由 $And(E_1,E_2)$ 映射,只有当 E_1 和 E_2 都发生时该事件才发生. 如果采取"累积"消耗策略[15],消耗不少于一个 E_1 的实例以及不少于一个 E_2 的实例才能产生一个 $And(E_1,E_2)$ 的实例. 因此,存在更严格的关联约束:

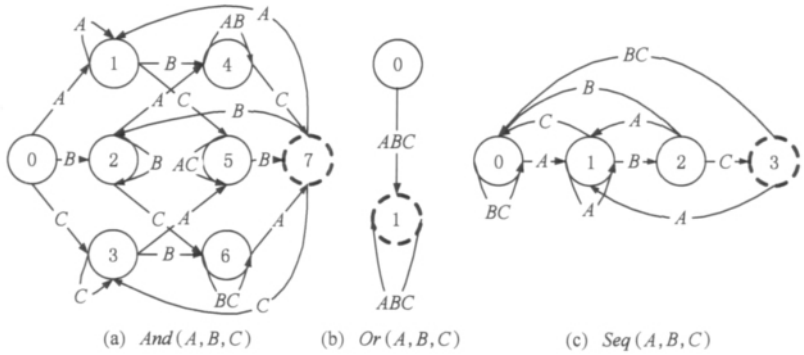
$$x_D\leqslant x_{E_1}\text{ 且 }x_D\leqslant x_{E_2}.$$

子问题 1. 为使复杂事件节点 v 产生 k 个实例,任一子节点 v_p 至少需向其输送 k_p 个实例,求 k_p.

若存在 k_p,则 v_p 向 v 输送的实例数 x_p 不小于 k_p,称为复杂事件节点内部约束,即

$$x_p\geqslant k_p.$$

为了求解 k_p 需要分析每个复杂事件子模式的语义. 一个复杂事件模式的语义可形式化为有限状态自动机[16]. 自动机包含一个初始状态、一个可接受状态和若干中间状态. 当一个成员事件实例发生时,自动机状态按照迁移规则进行一次迁移. 到达可接受状态表示复杂事件发生. 例如,图 4 展示了 $And(A,B,C)$,$Or(A,B,C)$ 和 $Seq(A,B,C)$ 语义形式化的自动机. 假设 $And(A,B,C)$ 自动机处于初始状态 0,3 个事件 A,B,C 依次发生,意味着自动机从状态 0 依次经过状态 1,4 迁移到状态 7. 由于状态 7 是可接受状态,说明复杂事件 $And(A,B,C)$ 发生.



(a) $And(A,B,C)$ (b) $Or(A,B,C)$ (c) $Seq(A,B,C)$

Fig. 4 Finite state machine formalized from complex event patterns.

图 4 以自动机表示复杂事件模式的语义

需要注意,虽然本节介绍的自动机构造与基于自动机的 CEP 程序中处理复杂事件所用的自动机类似,但这里仅用自动机来形式化复杂事件模式的语义,并非用于检测复杂事件. 分析的对象程序仍然是基于有向图的 CEP 程序.

在自动机中,成员事件的一个实例引起一次状态迁移,到达可接受状态表明一个复杂事件发生、一个新的复杂事件实例产生. 那么,子问题 1 等价于子问题 2.

子问题 2. 设 E 的成员事件为 $E_1,E_2,\cdots,E_f$,设自动机的输入字符集为 $CH=\{\zeta_1,\zeta_2,\cdots,\zeta_f\}$,每个字符对应一个成员事件. 用 s_{init} 及 s_{acc} 表示自动机的初始状态及可接受状态. 如果一个字符串能使自动机从 s_{init} 开始迁移并一共重复到达 s_{acc} 共 k 次,这

个字符串称为“k-字符串”.一个字符 $\zeta_p \in CH$ 在每个 k-字符串中最少出现 k_p 次,求 k_p.

定理 1. 子问题 2 与子问题 1 等价.

证明. 自动机 k 次到达 s_{acc} 意味着 E 发生 k 次,即 E 映射的节点产生了 k 个新实例.k-字符串至少包含 k_p 个字符 ζ_p,意味着成员事件 E_p 至少发生了 k_p 次,即节点 v_p 至少需要产生 k_p 个实例. 证毕.

为了求解 k_p,根据以下步骤构造一个字符串:

步骤 1. 构造一个子字符串 $\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}$ 包含最少 ζ_p,且可以使自动机从 s_{init} 迁移到 s_{acc},中间不经过 s_{acc},其中 $\zeta_{m1},\zeta_{m2},\cdots,\zeta_{mg}\in CH$.

步骤 2. 构造一个子字符串 $\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr}$ 包含最少 ζ_p,且可以使自动机从 s_{acc} 迁移回 s_{acc},中间不经过 s_{acc},其中 $\zeta_{l1},\zeta_{l2},\cdots,\zeta_{lr}\in CH$,且 $r\geqslant 1$.

步骤 3. 用 $\Psi(k)$ 表示构造的字符串,若 $k=1$,则 $\Psi(1)=\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}$;若 $k>1$,则

$$\Psi(k)=\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}\underbrace{\zeta_{l1}^{1}\zeta_{l2}^{1}\cdots\zeta_{lr}^{1}\cdots\zeta_{l1}^{k-1}\zeta_{l2}^{k-1}\cdots\zeta_{lr}^{k-1}}_{k-1\text{次}},$$

其中,前 g 个字符为 $\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}$,接着 $\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr}$ 循环出现 $k-1$ 次.

可以证明,按照此步骤构造的 $\Psi(k)$ 是一个包含最少字符 ζ_p 的 k-字符串.

定理 2. $\Psi(k)$ 是一个 k-字符串.

证明. $k=1$ 时,根据构造法步骤 1,$\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}$ 是一个 1-字符串.$k>1$ 时,$\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}$ 输入将使自动机从 s_{init} 第 1 次迁移到 s_{acc}.其后,根据构造法步骤 2,每个 $\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr}$ 输入均会使自动机到达 s_{acc} 一次.$\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr}$ 输入 $k-1$ 次后,自动机到达 s_{acc} 一共 k 次. 证毕.

定理 3. 在所有 k-字符串中 $\Psi(k)$ 含最少 ζ_p.

证明. 可用归纳法证明:1)$k=1$ 时,根据构造法步骤 1,$\Psi(1)=\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}$ 包含最少数量 ζ_p;2)归纳假设 $k=n-1$ 时,$\Psi(n-1)$ 包含最少 ζ_p.

$$\Psi(n-1)=\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}\underbrace{\zeta_{l1}^{1}\cdots\zeta_{lr}^{1}\cdots\zeta_{l1}^{n-2}\cdots\zeta_{lr}^{n-2}}_{n-2\text{次}},$$

需证明,$k=n$ 时,$\Psi(n)$ 含有最少 ζ_p:

$$\Psi(n)=\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg}\underbrace{\zeta_{l1}^{1}\cdots\zeta_{lr}^{1}\cdots\zeta_{l1}^{n-1}\cdots\zeta_{lr}^{n-1}}_{n-1\text{次}}.$$

归纳证明,设 $\zeta_{t1}\zeta_{t2}\cdots\zeta_{tq}$ 是任一 k-字符串,可切分为两个子字符串 $\zeta_{t1}\zeta_{t2}\cdots\zeta_{tj}$ 及 $\zeta_{tj+1}\cdots\zeta_{tq}$.输入前者使自动机从 s_{init} 开始迁移并重复达到 s_{acc} 共 $n-1$ 次.之后,输入后者使自动机第 n 次到达 s_{acc}.

根据归纳假设:

$$Ct_{\zeta_p}(\zeta_{t1}\zeta_{t2}\cdots\zeta_{tj})\geqslant Ct_{\zeta_p}(\Psi(n-1)),$$

$Ct_x(Y)$ 表示字符串 Y 中字符 x 出现的次数.

根据构造法步骤 1:

$$Ct_{\zeta_p}(\zeta_{tj+1}\cdots\zeta_{tq})\geqslant Ct_{\zeta_p}(\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr}),$$

因此:

$$Ct_{\zeta_p}(\zeta_{t1}\zeta_{t2}\cdots\zeta_{tq})=Ct_{\zeta_p}(\zeta_{t1}\zeta_{t2}\cdots\zeta_{tj})+Ct_{\zeta_p}(\zeta_{tj+1}\cdots\zeta_{tq})\geqslant Ct_{\zeta_p}(\Psi(n-1))+Ct_{\zeta_p}(\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr})=Ct_{\zeta_p}(\Psi(n)),$$

即 $k=n$ 时,在所有 k-字符串中,$\Psi(n)$ 含有最少 ζ_p. 证毕.

从 $\Psi(k)$,可解得 k_p:

$$k_p=Ct_{\zeta_p}(\Psi(k))=Ct_{\zeta_p}(\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg})+(k-1)Ct_{\zeta_p}(\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr}).$$

通过 Dijkstra 算法[22]可计算 $Ct_{\zeta_p}(\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg})$ 及 $Ct_{\zeta_p}(\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr})$.

算法 1. 计算使自动机从一个状态 s_a 迁移到另一个状态 s_b 的字符串包含的字符 ζ_p 的数量下限的 Dijkstra 算法.

输入:自动机结构及 s_a,s_b,ζ_p;

输出:s_b 的权值.

① 为自动机每个状态 s_i 赋初始权值 $a_{s_i}\leftarrow\infty$.创建集合 S 与 T,表示已求权值状态集与未求权值状态集.

② 字符 $\zeta_{s_i\to s_j}$ 使自动机从 s_i 迁移到 s_j,$w_{s_i\to s_j}$ 表示该迁移的权值.对所有迁移赋权值:

$$w_{s_i\to s_j}=\begin{cases}1, & \zeta_{s_i\to s_j}=\zeta_p,\\ 0, & \zeta_{s_i\to s_j}\neq\zeta_p.\end{cases}$$

③ $S=\{s_a\}$,$T=\{$其他状态$\}$.赋值 $a_{s_a}\leftarrow 0$.更新中状态权值:如果存在迁移 $s_a\to s_i$,$s_i\in T$,则赋值 $a_{s_i}\leftarrow a_{s_a}+w_{s_a\to s_i}$.

④ 在 T 中选择权值最小的状态(表示为 $s_{\min}$),$S=S\cup\{s_{\min}\}$,$T=T-\{s_{\min}\}$.

⑤ 更新 T 中的状态权值:如果存在迁移 $s_{\min}\to s_i$,$s_i\in T$ 且 $a_{s_{\min}}+w_{s_{\min}\to s_i}<a_{s_i}$,则赋值 $a_{s_i}\leftarrow a_{s_{\min}}+w_{s_{\min}\to s_i}$.

⑥ 重复步骤④⑤直到 $T=\varnothing$.

可以通过上述算法,赋 $s_a=s_{\text{init}}$,$s_b=s_{\text{acc}}$ 时,求得 $Ct_{\zeta_p}(\zeta_{m1}\zeta_{m2}\cdots\zeta_{mg})=a_{s_b}$;赋 $s_a=s_{\text{acc}}$,$s_b=s_{\text{acc}}$ 时,求得 $Ct_{\zeta_p}(\zeta_{l1}\zeta_{l2}\cdots\zeta_{lr})=a_{s_b}$.

4.2 根据分析结果进行批注

本小节将使用上面得到的关联关系对现有隐含路径计数方法进行批注,使其可以用于估算 CEP 程序最坏响应时间.还将讨论如何使用上述约束条件对结构合并方法以及路径搜索方法进行批注.

隐含路径计数方法通过求解整数线性规划问题得出最坏响应时间. 以文献[10]的方法为例.

定义 2. 设 m_j 表示控制流图中第 j 块, y_j 表示 m_j 执行次数, c_j 表示 m_j 执行时间. 可以假设, 通过硬件模型分析, c_j 已知.

整数线性规划的目标函数为

$$\max\left(\sum_{m_j} c_j y_j\right).$$

通过分析控制流图顺序、分支、循环结构可得 y_i 间的约束关系. 例如, 通过分析图 3(b)所示的控制流图, 可得约束 $y_0 = y_1$, $y_4 = y_5 + y_6 + y_7$ 等.

对于基于有向图的 CEP 程序, 由于节点子程序的执行次数上限未知, 整数线性规划问题不可解, 因此, 上述隐含路径计数方法不能直接使用.

下面通过将 px 与 x 的约束条件转化为 y 的约束条件使整数线性规划问题可解. 首先, 有向图节点 v_i 接收到一个实例时, 其节点子程序处理此实例. 在控制流图中, v_i 的节点子程序映射为一个局部控制流图. 它含若干入口块(表示为 $m^i_{im\_j}$)及若干出口块(表示为 $m^i_{ex\_j}$). 一个节点子程序的执行次数, 等于其控制流图所有入口块执行次数之和, 也等于所有出口块执行次数之和, 即

$$px_i = \sum_{m^i_{im\_j}} y^i_{im\_j} = \sum_{m^i_{ex\_j}} y^i_{ex\_j},$$

其次, v_i 的节点子程序含有若干语句负责产生新实例. 在节点子程序控制流图中, 设块 $m^i_{rs\_j}$ 包含第 j 个输出语句. v_i 输出实例数等于所有 $m^i_{rs\_j}$ 执行次数之和, 即

$$x_i = \sum_{m^i_{rs\_j}} y^i_{rs\_j}.$$

通过上述等式, 可以将 px 与 x 的约束条件转化为 y 的约束, 使整数线性规划问题可解. 因此, 改进批注方式后的隐含路径计数方法可用于估算 CEP 程序的最坏响应时间.

下面讨论如何使用关联关系对结构合并以及路径搜索的估算方法进行批注, 这两种方法只支持批注子程序取值范围, 不支持直接批注约束条件. 为了将其用于估算 CEP 程序的最坏响应时间, 可以根据关联约束求得节点子程序的执行次数上限.

然而, 通过变量的关联约束解得的变量取值上限, 可能扩大变量的定义域. 将其用于批注, 估算精度较低. 例如, 对如图 1 所示的有向图, 设 $N=4$, 可解得 $x_{E_1} \leqslant 4$, $x_{E_2} \leqslant 4$. 然而, 相较于输入约束 $x_{E_1} + x_{E_2} + x_{E_3} + x_{E_4} \leqslant 4$, 它们增加了 x_{E_1} 和 x_{E_2} 的无效取值. 第 5 节模拟实验将进一步分析此示例. 因此, 对于隐含路径计数方法, 直接批注子程序执行次数关联约束优于批注通过关联约束解出的取值范围. 对于其他两类估算方法, 由于只支持批注取值范围, 因此可能估算精度较低.

4.3 讨 论

首先, 讨论本文方法对不同应用场景假设的适用性. 在 px 与 x 关联约束中, 除输入约束外, 其他约束与应用场景假设无关. 因此只要能根据应用场景假设, 归纳出输入约束, 限制每个原子事件节点的访问次数, 本方法就可以适用. 例如, 在 4.1 小节提到的每个原子事件实例数量上限已知的应用场景.

其次, 考察本方法对于基于有限状态自动机及基于 Petri 网的 CEP 程序的适用性.

本方法通过分析有向图检测结构, 归纳了节点访问次数 px 和产生新实例数 x 间的关联关系. 在有向图中, 一个节点对应于用于检测一个复杂事件子模式的子程序. 那么, 只要 CEP 程序能够划分出用于检测每个子模式的子程序, 且子程序间的调用关系或数据流传递关系能映射为有向无环图, 本方法就适用.

例如, 文献[17]提出使用 Petri 网检测复杂事件: 每个复杂事件子模式用一个 Petri 网检测. 多层次嵌套的复杂事件, 通过将成员事件 Petri 网的输出库所与上层复杂事件 Petri 网相应输入库所通过变迁连接, 实现多个子 Petri 网合并. 通过将子 Petri 网映射为节点, 子 Petri 网间的变迁映射为连接, 整个 Petri 网可以映射为有向无环图. 因此, 本方法适用于估算该类 CEP 程序的最坏响应时间.

对于基于有限状态自动机的 CEP 程序, 难以将其映射为有向无环图, 因而不能使用本方法.

下面分析时间复杂度. 本方法在现有的隐含路径计数方法基础上, 对有向图中每个节点仅增加 2～4 个约束条件, 增加的约束条件数量占约束条件总量的较小部分. 因此, 本方法与现有隐含路径计数方法的时间复杂度相同, 实际求解时间略有增长. 隐含路径计数方法需要求解整数线性规划问题, 如果略微放宽估算精度要求, 可以求解相应线性规划问题作为近似, 线性规划问题可在多项式时间内求解.

5 模拟实验

对 CEP 程序的最坏响应时间估算问题无现有估算方法, 因此, 本方法不能与已有方法进行比较.

本实验有 3 个目的: 1)讨论标注子程序执行次

数关联约束条件替代标注取值范围的必要性;2)讨论标注的准确性对估算精度的影响;3)讨论复杂事件模式集的复杂程度对估算精度的影响.

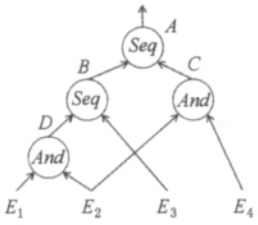
下面分别进行两个实验:1)指定复杂事件模式集;2)随机产生复杂事件模式集.

5.1　实验1:指定复杂事件模式集

1) 参数设置

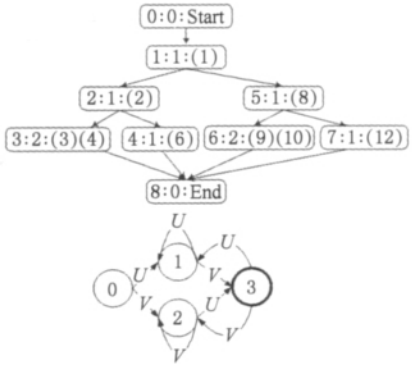
在本实验中,预定义复杂事件模式集仅包含 $Seq(Seq(And(E_1,E_2),E_3),And(E_2,E_4))$. 本实验将展示本文方法的估算过程,分析直接标注子程序执行次数关联约束条件的必要性,同时分析标注的准确性对估算精度的影响.

分析对象如图5所示,包括用于检测复杂事件的有向图结构,检测 And 和 Seq 类复杂事件的节点子程序的伪代码、其控制流图及语义自动机. 复杂事件模式、有向图及伪代码来自于文献[15]. 控制流块上标注"块 id:时间权值:(语句序号1)(语句序号2)…". 为了减少其他影响因素,假设伪代码每条语句的执行时间为1,因此块权值等于块包含的语句数.



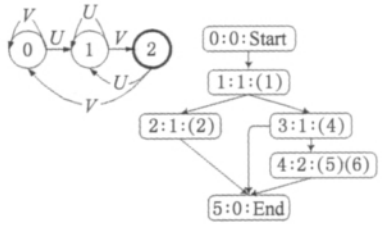
(a) Directed graph for detecting $Seq(Seq(And(E_1,E_2)\ E_3),And(E_2,E_4))$

```
And(U,V)
1  if left event u is signalled
2    if v's list is not empty
3      Pass <all v's,u> to the parent;
4      FLUSH V's buffer;
5    else
6      Append u to U's list;
7  else /*right event v is signaled*/
8    if U's list is not empty
9      Pass <all u's,v> to the parent;
10     FLUSH U's buffer;
11   else
12     Append v to V's list.
```



(b) Pseudo codes for detecting $And(U,V)$, their control flow graphs and their finite state machines

```
Seq(U,V)
1  if left event u is signalled
2    Append u to U's list;
3  else /*right event v is signaled*/
4    if U list is not empty
5      Pass <all u's,v> to parent;
6      FLUSH U's buffer.
```



(c) Pseudo codes for detecting $Seq(U,V)$, their control flow graphs and their finite state machines

Fig. 5　The case for experiment 1.

图5　实验1案例

2) 估算过程

根据文献[10]提出的隐含路径计数方法,目标函数为

$$\max\Big(\sum_{i\in\{A,B\},j=1\sim5}c_j^i y_j^i+\sum_{i\in\{C,D\},j=1\sim7}c_j^i y_j^i\Big).$$

通过分析控制流图可得约束集Ⅰ:

$i\in\{A,B\}$, $y_0^i=y_1^i=y_5^i=y_2^i+y_3^i$, $y_4^i\leqslant y_3^i$;

$i\in\{C,D\}$, $y_0^i=y_1^i=y_2^i+y_5^i$, $y_2^i=y_3^i+y_4^i$;

$y_5^i=y_6^i+y_7^i$, $y_8^i=y_3^i+y_4^i+y_6^i+y_7^i$.

由于未对 y_0^A,y_0^B,y_0^C,y_0^D 进行约束,因此上述整数线性规划问题无解,需要增加约束条件.

通过分析有向图检测结构可得约束集Ⅱ,包括输入约束、原子事件节点约束、复杂事件节点输入约束、复杂事件节点输出约束:

$\sum_{i=1\sim4} x_{E_i} \leqslant N$, $px_D = x_{E_1} + x_{E_2}$, $x_D \leqslant px_D$, $px_C = x_{E_2} + x_{E_4}$, $x_C \leqslant px_C$, $px_B = x_D + x_{E_3}$, $x_B \leqslant px_B$, $px_A = x_B + x_C$, $x_A \leqslant px_A$.

通过分析形式化自动机可得约束集Ⅲ,包括复杂事件节点内部约束:

$$x_D \leqslant x_{E_1}, x_D \leqslant x_{E_2}, x_C \leqslant x_{E_2}, x_C \leqslant x_{E_4},$$

$$x_B \leqslant x_D, x_B \leqslant x_{E_3}, x_A \leqslant x_B, x_A \leqslant x_C.$$

将 x 及 px 与 y 进行连接可得约束集Ⅳ:

$$i \in \{A,B\}, px_i = y_0^i = y_5^i, x_i = y_4^i;$$

$$i \in \{C,D\}, px_i = y_0^i = y_8^i, x_i = y_3^i + y_6^i.$$

本实验中取 $N=4$,通过约束集Ⅱ和约束集Ⅲ,可解出 x 与 px 的取值范围:

$$i = 1\sim4, x_{E_i} \leqslant 4; px_D, px_C, px_B \leqslant 4;$$

$$x_D, x_C, px_A \leqslant 2; x_B, x_A \leqslant 1.$$

为了考察标注准确性对估算精度的影响,根据标注的约束集不同,分 3 种方法估算最坏 CEP 程序最坏响应时间:1) M_1:标注约束集Ⅰ,Ⅱ,Ⅲ,Ⅳ,包含 47 个约束条件;2) M_2:标注约束集Ⅰ,Ⅱ,Ⅳ,包含 39 个约束条件;3) M_3:标注子程序执行次数的取值范围,因此只标注约束集Ⅰ,Ⅳ,Ⅴ,包含 42 个约束条件. 其中,方法 M_1 是本文提出的方法.

为了探讨约束条件的准确性的影响,本实验设置了方法 M_2. 这里选择去掉约束集Ⅲ的原因是,去掉约束集Ⅰ,Ⅱ,Ⅳ均将造成整数线性规划问题无解. 而约束集Ⅲ包含的复杂事件节点内部约束是对复杂事件节点输出约束的改进,去掉约束集Ⅲ不会造成问题无解,同时可以使得标注的准确性降低.

为了探讨标注约束条件而非取值范围的必要性,本实验设置了方法 M_3. 方法 M_3 按照现有估算方法要求,只标注了节点子程序执行次数的取值范围.

3) 实验结果

$N=4$ 时,实验结果如表 1 所示. 穷举发现,当输入序列 $ES=\{e_1,e_2,e_3,e_4\}$ 时,响应时间取最大值 $W_p = 26$. M_1, M_2, M_3 估算值分别为 $W_p^{M_1} = 26$, $W_p^{M_2} = 80$, $W_p^{M_3} = 44$. M_1, M_2, M_3 方法的计算时间分别为 9.6 ms, 8.5 ms, 7.3 ms.

Table 1 Estimation Values of the WCET ($N=4$)

表 1 最坏响应时间估算结果($N=4$)

Methods of Estimation	W_p	M_1	M_2	M_3
Worst-case Execution Time	26	26	80	44
Calculation Time/ms		9.6	8.5	7.3

首先,本文方法(M_1)具有 100%的估算精度,表明本文的方法对于简单的复杂事件模式非常有效.

其次,约束集Ⅴ较之约束集Ⅱ,Ⅲ扩大了部分变量的定义域. 例如,考虑约束集Ⅴ,有可行取值 $x_{E_1} = 4$, $x_{E_2} = 4$. 而约束集Ⅱ有 $x_{E_1} + x_{E_2} \leqslant 4$,上述取值不是可行取值. 因此,在方法 M_3 中,相较于关联约束(约束集ⅡⅠ,Ⅲ),只考虑取值范围(约束集Ⅴ),使节点子程序执行次数的定义域扩大,增加了无效取值,后果是降低了估算精度. 实验表明,使用关联约束取代取值范围进行标注是必要的,可以有效提高估算精度.

第三,在方法 M_2 中,缺乏严格的约束条件(缺乏约束集Ⅲ)使得估算值与真实值有较大差距. 因此,归纳更加严格的约束条件对提高估算精度是有意义的.

考察计算时间,M_1 的约束条件数量分别比 M_2 和 M_3 仅增加 20.5%及 11.9%,求解整数线性规划问题的时间代价增幅也相应较小.

5.2 实验 2:随机复杂事件模式集

在本实验中,随机生成的复杂事件模式集,分析模式集的复杂程度对估算精度的影响.

1) 参数设置

我们根据文献[15]提出的 CEP 算法开发了 CEP 程序. 实验中估算方法基于文献[10]所提出的隐含路径计数方法进行改进. 为了减少影响因素,同样假设设每条语句的执行时间为 1,块时间权值等于其包含的语句数.

为了更好地讨论准确标注的重要性,本实验同样计算两种方法 M_1 和 M_2 下的估算值及估算精度. 方法 M_1(本文方法)包含所有的约束,方法 M_2 不包含复杂事件节点内部约束.

在实验中,以 r_p^M 度量估算精度为

$$r_p^M = W_p^M / W_p^{\max},$$

W_p^M 是估算方法 M 对最坏响应时间的估算值;$W_p^{\max}$ 是 CEP 程序多次运行后统计得出的实际响应时间的最大值:给定 N,随机生成 m 个长度为 N 的原子事件实例序列,分别输入并运行 CEP 程序,测量每次的响应时间,m 个响应时间的最大值即为 $W_p^{\max}$. 使用 $W_p^{\max}$ 替代真实的最坏响应时间 W_p 的原因是,N 取值较大时难以穷举所有可能的输入序列,真实的最坏响应时间 W_p 难以获知.

由于安全性要求 $W_p^{\max} \leqslant W_p \leqslant W_p^M$,因此,$r_p^M \geqslant 100\%$,越靠近 100%说明 M 精度越高. 同时,M 实际估算精度 W_p^M / W_p 满足 $W_p^M / W_p \leqslant r_p^M$,说明 M 实际

精度不低于 r_p^M.

随机生成复杂事件模式集的参数如下:

① 事件数 n_E:复杂事件模式集中包含的所有复杂事件模式、子模式、原子事件总数.

② 原子事件数 n_P:所有复杂事件模式中包含的原子事件数量,$n_P \leqslant n_E$.

③ 最大成员数 n_c:复杂事件成员事件个数上限.

复杂事件模式集的复杂程度可由 n_E,n_P 及 n_c 反映.同等条件下,n_E 越大 n_P 越小或 n_c 越大,说明复杂事件模式集越复杂.

算法 2. 复杂事件模式集随机生成算法.

输入:n_E,n_P,n_c 及可供选择的事件操作符;

输出:复杂事件模式集 Γ,原子事件集 Γ_P.

① 构建事件集 $\Gamma=\varnothing$及原子事件集 $\Gamma_P=\varnothing$;

② 新建 n_P 个原子事件加入 Γ 及 Γ_P;

③ 新建 1 个复杂事件模式 E_i,在[2,n_c]间随机生成数 r,在 Γ 中随机选择 r 个事件作为 E_i 的成员事件.将 E_i 加入 Γ.

④ 重复步骤③,直到$|\Gamma\}=n_E$.

实验中,在 n_E,n_P,n_c 同一套取值下,随机生成复杂事件模式集 F 次,估算与测量结果取平均值.

表 2 列举了本实验中所涉及的参数.

2) 实验步骤

本实验按照下列步骤进行:

① 给定 n_E,n_P,n_c,生成复杂事件模式集;

Table 2　List of Parameters in Experiment 2

表 2　实验 2 涉及参数

Titles	Name	Details
Parameters	n_E	Number of events
	n_P	Number of primitive events
	n_c	Maximal number of component events in each complex event
	F	Number of rule sets under the same parameters
	m	Number of input queues
	N	Number of instances in an input queue
Results	W_p,$W_p^{M_1}$,$W_p^{M_2}$	Actual WCET and estimation values with M_1 and M_2
	$r_p^{M_1}$,$r_p^{M_2}$	Estimation accuracies of M_1 and M_2

② 给定 N,计算 $W_p^{M_1}$,$W_p^{M_2}$;

③ 根据 N 随机生成一个原子事件实例序列,执行 CEP 程序,测量其实际响应时间 R_p^i;

④ 重复步骤③m 次,计算 $W_p^{\max}=\max\limits_{i=1\sim m}R_p^i$;

⑤ 计算 $r_p^{M_1}$,$r_p^{M_2}$;

⑥ 重复步骤①～⑤ F 次,计算 $W_p^{M_1}$,$W_p^{M_2}$,$W_p^{\max}$,$r_p^{M_1}$,$r_p^{M_2}$ 的平均值$\overline{W_p^{M_1}}$,$\overline{W_p^{M_2}}$,$\overline{W_p^{\max}}$,$\overline{r_p^{M_1}}$,$\overline{r_p^{M_2}}$;

⑦ 改变 n_E,n_P 或 n_c,重复步骤①～⑥.

3) 实验结果

实验结果如图 6 所示,从左至右分别是变化 n_E,n_P,n_c 的实验结果,图 6(a)～(c)展示了估算方法 M_1,

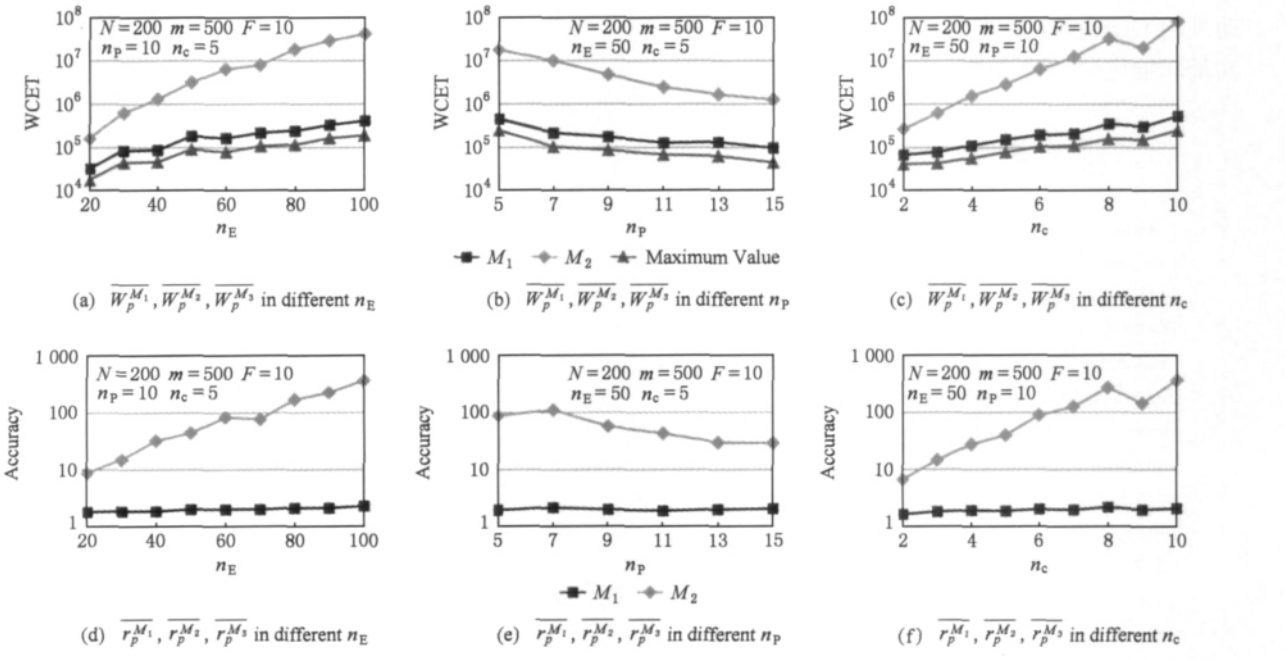


Fig. 6　Results of experiment 2.

图 6　实验 2 结果

M_2 估算出的及统计得到最坏响应时间,图 6(d)～(f)展示了 M_1,M_2 的估算精度,所有图的纵轴都是指数的.

从实验结果可以看出,本文方法(M_1)的估算精度较高,在事件模式集的各种复杂程度下估算精度保持在 200%左右,这说明本文方法对复杂事件模式集的复杂程度不敏感,在复杂程度较高的情况下仍然保持了较高的估算精度.

作为比较,去掉复杂事件节点内部约束后(M_2 方法),估算精度降低了 1～2 个数量级. 例如 $n_E=50$,$n_P=10$,$n_c=10$ 时估算精度低至 36 783%. 凸显了提高标注准确性对提高估算精度有重要意义.

6 结 论

本文描述了 CEP 程序的最坏响应时间估算问题,并提出了一种估算方法. 方法通过对 CEP 程序的检测结构进行分析归纳节点子程序执行次数之间的关联约束关系,用关联关系进行标注代替了标注执行次数的取值范围. 模拟实验表明,本方法能够有效地估算 CEP 程序的最坏响应时间,具有较高的估算精度. 同时,实验表明,用子程序执行次数的约束条件取代取值范围进行标注是有必要的,同时,改进标注的准确性具有重要的意义.

接下来的工作包括 3 方面:1)进一步归纳关联约束条件,提高估算精度;2)解析求解本方法的估算精度上下限;3)拓展本方法的适用范围,对于基于自动机的 CEP 程序等其他类型 CEP 程序,如何分析其最坏响应时间需要进一步研究.

参 考 文 献

[1] Luckham D C. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems [M]. Reading, MA: Addison-Wesley, 2001

[2] Buttazzo G C. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications [M]. 2nd ed. Berlin: Springer, 2005

[3] General Administration of Quality Supervision, Inspection and Quarantine of the People's Republic of China. GB4717-2005 Fire Alarm Control Units [S]. Beijing: China Zhijian Publishing House, 2005
(中华人民共和国国家质量监督检验检疫总局. GB4717-2005 火灾报警控制器[S]. 北京: 中国标准出版社, 2005)

[4] Sybase Inc. Sybase aleri: The award-winning complex event processing (CEP) platform [EB/OL]. [2012-06-09]. http://www.sybase.com/products/financialservicessolutions/complex-event-processing

[5] EsperTech Inc. Esper: Event processing for Java [EB/OL]. [2012-06-09]. http://www.espertech.com/products/esper.php

[6] Wu E, Diao Y, Rizvi S. High-performance complex event processing over streams [C] //Proc of ACM SIGMOD'06. New York: ACM, 2006: 407－418

[7] Agrawal J, Diao Y, Gyllstrom D, et al. Efficient pattern matching over event streams [C] //Proc of ACM SIGMOD'08. New York: ACM, 2008: 147－160

[8] Magid Y, Adi A, Barnea M, et al. Application generation framework for real-time complex event processing [C] //Proc of the 32nd Annual IEEE Int Computer Software and Applications Conf. Piscataway, NJ: IEEE, 2008: 1162－1167

[9] Anicica D, Rudolphb S, Fodorc P, et al. Real-time complex event recognition and reasoning—A logic programming approach [J]. An Int Journal of Applied Artificial Intelligence, 2012, 26(1/2): 6－57

[10] Li Y T S, Malik S, Wolfe A. Performance estimation of embedded software with instruction cache modeling [J]. ACM Trans on Design Automation of Electronic Systems, 1999, 4(3): 257－279

[11] Gustafsson J, Ermedahl A, Lisper B. Towards a flow analysis for embedded system C programs [C] //Proc of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. Piscataway, NJ: IEEE, 2005: 287－300

[12] Heckmann R, Langenbach M, Thesing S, et al. The influence of processor architecture on the design and the results of WCET tools [J]. Proc of the IEEE, 2003, 91(7): 1038－1054

[13] Wilhelm R, Engblom J, Ermedahl A, et al. The worst—Case execution-time problem—Overview of methods and survey of tools [J]. ACM Trans on Embedded Computing Systems, 2008, 7(3): 1－53

[14] Zang Chuanzhen. Research on complex event processing and its applications in enterprise information systems [D]. Beijing: Tsinghua University, 2007 (in Chinese)
(臧传真. 复杂事件处理机制研究及其在企业信息系统中的应用[D]. 北京: 清华大学, 2007)

[15] Krishnaprasad V. Event detection for supporting active capability in an OODBMS: Semantics, architecture, and implementation [D]. Gainesville, Florida: Database Systems R&D Center, CIS Department, University of Florida, 1994

[16] Gehani N H, Jagadish H V, Shmueli O. Event specification in an active object-oriented database [J]. ACM SIGMOD Record, 1992, 21(2): 81－90

[17] Gatziu S, Dittrich K R. Events in an active object-oriented database system [R]. Zurich: University of Zurich, 1993

[18] Wenzel I, Kirner R, Rieder B, et al. Measurement-based worst-case execution time analysis [C] //Proc of the 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems. Piscataway, NJ: IEEE, 2005: 7－10

[19] Colin A, Puaut I. Worst case execution time analysis for a processor with branch prediction [J]. Real-Time System, 2000, 18(2/3): 249－274

[20] Stappert F, Ermedahl A, Engblom J. Efficient longest executable path search for programs with complex flows and pipeline effects [C] //Proc of the 2001 Int Conf on Compilers, Architecture, and Synthesis for Embedded Systems. New York: ACM, 2001: 132－140

[21] Puschner P, Burns A. Guest editorial: A review of worst-case execution-timeAnalysis [J]. Real-Time System, 2000, 18(2/3): 115－128

[22] Dijkstra E W. A note on two problems in connexion with graphs [J]. Numerische Mathematik, 1959, 1(1): 269－271

Li Xiang, born in 1983. PhD, post-doctorate. His current research interests include real-time complex event processing.

Fan Yushun, born in 1962. PhD, professor and PhD supervisor. His main research interests include enterprise integration, workflow, service computing, and modern integrated manufacturing system.

Wang Hongan, born in 1963. PhD, professor and PhD supervisor. His main research interests include real-time database and human-computer interaction.

Qiao Ying, born in 1973. PhD, associate professor and master supervisor. Her main research interests include real-time intelligence.