ISSN1000-9825 Journal of Software 软件学报 2000,11(8):1035~1040

基于事件约束的分布式程序正确性测试*

顾庆 陈道蓄 于勤 谢立 孙钟秀

(南京大学计算机软件新技术国家重点实验室 南京 210093)

E-mail: guq@dislab.nju.edu.cn

摘要 由于并发的存在和不确定性,在以规约为基础来测试分布式程序的正确性时,必须考虑程序执行时的内部状态.这些内部状态通过端口显示为事件序列,程序规约需要对序列中各事件间的依赖关系作约定,即定义事件约束集.该文提出了E-CSPE(extended-constraints on succeeding and preceding events),以形式化描述这类事件约束,它由3个基本描述规则组成,分别对应于3种不同类型的事件约束.通过判断程序执行时所产生的事件序列集同这些事件约束集的一致性以及对约束集覆盖程度可以检测被测程序的正确性.

关键词 分布式程序测试,自动机模型,端口,CSPE(constraints on succeeding and preceding events),事件约束.

中图法分类号 TP311

测试一个程序可以从两个角度入手,即程序实现角度和程序规约角度.从程序实现角度出发可以确定程序结构,并基于结构覆盖选择测试用例.从程序规约角度出发需要根据规约判断程序执行的正确性.程序规约包括5个基本要素[1]:数据(data)、端口(ports)、动作(actions)、事件(events)以及线索(threads).根据这5个要素可以构成程序的行为模型(behavioral model),本文采用适合于分布式程序测试的自动机模型.

1 基于事件约束的分布式程序测试

测试分布式程序应当以程序规约为基础[1],需要对程序行为作出约定(behavior specification).行为约定以线索为基础,规定了程序在特定输入下的动作序列.对于分布式程序,为了避免状态-测试爆炸(state-transition and test case explosion)问题,可以通过用户定义的方式规定一些关键动作 $KEYACTION(KA)$,即动作序列 $s_A=\langle ACTION_{in},KA_1,KA_2,\ldots,KA_m,ACTION_{out}\rangle$.测试时为感知这些关键动作,可以通过端口将它们转化为关键事件 $KEYEVENT(KE)$,这时,动作序列成为事件序列 $s=\langle ACTION_{in},KE_1,KE_2,\ldots,KE_m,ACTION_{out}\rangle$.

对于事件序列 s,在关键事件的基础上,行为约定必须规定:(1) 任两个关键事件 KE_i 和 KE_j 间的依赖关系;(2) 这种依赖关系间的或然性.

针对分布式程序选择测试用例时需要避免指数级的测试规模,一种可行的方案是先针对被测程序确定它的端口,然后就端口模块自身而不是就整个程序来选择覆盖策略.该方案符合"从使用视角出发(operational profile[1,2])"这一基本理念.端口的数量和规模相对于整个程序所占比例一般很小,可以减少测试用例的数量,避免测试爆炸问题.

采用这种方法测试分布式程序需要考虑两个方面的问题.一方面是在执行测试用例所产生的事件序列集中,每一个事件序列都必须符合规约的约定;另一方面是在规约中每一个对事件间依赖关系的约定都必须在事

* 本文研究得到国家"九五"重点科技攻关项目基金(No.98-780-01-07-03)资助.作者顾庆,1972年生,博士,讲师,主要研究领域为分布式处理,软件测试.陈道蓄,1947年生,教授,博士生导师,主要研究领域为分布式计算,并行处理.于勤,1972年生,博士,主要研究领域为分布式处理,并行编译.谢立,1942年生,教授,博士生导师,主要研究领域为分布式处理,并行处理.孙钟秀,1936年生,教授,博士生导师,中国科学院院士,主要研究领域为分布式计算,并行处理.

本文通讯联系人:顾庆,南京 210093,南京大学计算机软件新技术国家重点实验室

本文 2000-01-17 收到原稿,2000-04-14 收到修改稿

件集中有所反映.前者是测试程序正确性的前提,而后者则是测试充分性的保证.

2 基于 E-CSPE 约束的分布式程序测试

测试一个分布式程序的正确性需要根据对程序行为的约定.该约定可以基于用户定义的关键事件,约定的具体形式随着行为模型的不同而不同.对于以自动机模型表示的分布式程序,一种可行的描述方案是 CSPE (constraints on succeeding and preceding events)约束[3].

CSPE 约束包括 3 种基本描述规则,分别是:

R1. $a[E_1;\rightarrow E_2]_U$:(always),在分布式程序 U 的执行中,事件 E_2 发生于事件 E_1 之后总是有效,不需要前提条件;但 E_1 之后 E_2 也可以不发生.

R2. $\sim[E_1;\rightarrow E_2]_U$:(never),在分布式程序 U 的执行中,事件 E_2 发生于 E_1 之后总是无效(错误),不需要前提条件.

R3. $p[E_1;\rightarrow E_2]_U$:(possible),在分布式程序 U 的执行中,事件 E_2 可以发生于 E_1 之后,但这需要某个条件 P 的约束.

针对特定分布式程序,CSPE 约束可以根据对该程序的规约(抽象程序)表示自动导出,也可以用手工编写.

2.1 E-CSPE 约束

在实践中采用 CSPE 约束描述存在以下缺陷:

• 规则 R1 中的"a(always)"不够完善,例如,它不利于描述这样一种必然情形:即事件 E_1 发生后,事件 E_2 必然发生,否则无效(错误).

• 规则 R3 中的"p(possible)"不够清晰,因为它代表了多种可能性.

• CSPE 要求前后两个事件紧密相连,而在实际测试中可能需要某种抽象,例如不考虑中间一些属于过渡的非关键事件.

为解决该缺陷,现定义 E-CSPE(extended CSPE)约束规则如下:设 U 为一个分布式程序,E_1 和 E_2 为任意两个存在依赖关系的事件,α 和 β 为任意两个不完整的事件序列(子序列,可能为空),x 为程序的任一输入,C 为描述程序状态的谓词.

R1′. $a[[E_1;\rightarrow E_2]]_UC$:(always under condition),在程序 U 执行输入 x 时,若产生的事件前缀 $\alpha.E_1$ 满足条件 C,则总会存在子序列 β,使得 $\alpha E_1.\beta$ 满足条件 C,且 $\alpha E_1.\beta.E_2$ 为有效的事件前缀.

R2′. $m[[E_1;\rightarrow E_2]]_UC$:(must under condition),在程序 U 执行输入 x 时,若产生的事件前缀 $\alpha.E_1$ 满足条件 C,则必有子序列 β,使得 $\alpha E_1.\beta$ 满足条件 C,且 $\alpha E_1.\beta.E_2$ 为有效的事件前缀.

R3′. $\sim[[E_1;\rightarrow E_2]]_UC$:(never under condition),在程序 U 执行输入 x 时,若产生的事件前缀 $\alpha.E_1$ 满足条件 C,则不存在子序列 β,使得 $\alpha E_1.\beta$ 满足条件 C,且 $\alpha E_1.\beta.E_2$ 为有效的事件前缀.

2.2 E-CSPE 约束描述同 CSPE 间的关系

首先,规则 R1′和规则 R2′有如下关系:

定理 1. 规则 R2′蕴含规则 R1′.即:若 $m[[E_1;\rightarrow E_2]]_UC$,则必有 $a[[E_1;\rightarrow E_2]]_UC$.

证明:由定义易得.

R1′代表了依赖关系的一种可能性,它表示可以有多种选择.而 R2′代表一种必然性,它是确定的,没有其他选择. □

定理 2. E-CSPE 约束描述可以由 CSPE 定义.

证明:考虑连续的两个事件 E_1 和 E_2,有:

对规则 R1′,可以有两种定义:

• 由规则 R1 导出,即 $a[E_1;\rightarrow E_2]_U\Rightarrow a[[E_1;\rightarrow E_2]]_U$(true);

• 由规则 R3 导出,即 $p[E_1;\rightarrow E_2]_U\Rightarrow a[[E_1;\rightarrow E_2]]_U(P)$.

对规则 R2′,可以按如下方式定义:

设有限事件集合$\{E_1,E_2,\ldots,E_n\}$为程序U执行时的所有可能事件,若有$a[E_1;\rightarrow E_2]_U$,且对$\forall i:i\neq 2.\sim[E_1;\rightarrow E_i]_U$,则可以导出:$m[[E_1;\rightarrow E_2]]_U(\text{true})$.

对规则R3′,可以有如下定义:

- 由规则R2导出,即$\sim[E_1;\rightarrow E_2]_U\Rightarrow\sim[[E_1;\rightarrow E_2]]_U(\text{true})$;
- 由规则R3导出,即$p[E_1;\rightarrow E_2]_U\Rightarrow\sim[[E_1;\rightarrow E_2]]_U(\sim P)$.

2.3 E-CSPE约束间的关系

根据定义,E-CSPE的3个描述规则之间有如下的关系(设状态谓词C不完全依赖于某一特定事件,如E_1,E_2或E_3).

定理3. 在相关条件C成立的情况下,规则R1′具有传递性.即:若有$a[[E_1;\rightarrow E_2]]_UC$,且$a[[E_2;\rightarrow E_3]]_UC$,必有$a[[E_1;\rightarrow E_3]]_UC$.

证明:按定义,程序U在执行输入x时,产生的事件前缀$\alpha..E_1$满足条件C,存在子序列β,使得$\alpha E_1.\beta$满足条件C,并且$\alpha E_1.\beta..E_2$为有效的事件前缀.此时,若C成立,则存在子序列γ,使得$\underline{\alpha E_1.\beta}..E_2.\gamma$满足条件$C$,并且$\alpha E_1.\beta..E_2.\gamma.E_3$为有效的事件前缀.即:$U$在执行输入$x$时产生的事件前缀$\alpha..E_1$满足条件$C$,存在子序列$\underline{\beta.E_2.\gamma}$,使得$\alpha E_1.\underline{\beta.E_2.\gamma}$满足条件$C$,并且$\alpha E_1.\underline{\beta.E_2.\gamma}..E_3$为有效的事件前缀,从而$a[[E_1;\rightarrow E_3]]_UC$.定理得证. □

将该定理推广,不难得出以下两个推论.

推论1. 若有$m[[E_1;\rightarrow E_2]]_UC$,且$m[[E_2;\rightarrow E_3]]_UC$;→必有$m[[E_1;\rightarrow E_3]]_UC$.证明略.

结合定理1,可以得出:

推论2. 若有$m[[E_1;\rightarrow E_2]]_UC$,且$a[[E_2;\rightarrow E_3]]_UC$,或有$a[[E_1;\rightarrow E_2]]_UC$,且$m[[E_2;\rightarrow E_3]]_UC$,则必有$a[[E_1;\rightarrow E_3]]_UC$.

考虑规则R3′,可以得出:

定理4. 若有$\sim[[E_1;\rightarrow E_3]]_UC$,且$a[[E_2;\rightarrow E_3]]_UC$,则有$\sim[[E_1;\rightarrow E_2]]_UC$.

证明:用反证法易得. □

结合定理1,可以进一步得出以下两个推论.

推论3. 若有$\sim[[E_1;\rightarrow E_3]]_UC$,且$m[[E_2;\rightarrow E_3]]_UC$,则有$\sim[[E_1;\rightarrow E_2]]_UC$.

推论4. 若有$m[[E_1;\rightarrow E_2]]_UC$,且$\sim[[E_2;\rightarrow E_3]]_UC$和$\sim[[E_3;\rightarrow E_2]]_UC$,则有$\sim[[E_1;\rightarrow E_3]]_UC$.证明略.

2.4 基于E-CSPE约束的程序验证

E-CSPE约束描述有3个基本描述规则,分别对应于3种类型的事件约束.就一个生成(可行)的事件序列s,它可以覆盖(符合)一个事件约束,或者违反一个事件约束,或者代表程序出错.具体来讲,有:

- $a[[E_1;\rightarrow E_2]]_UC$:若$s$中包含一个事件前缀$\alpha E_1.\beta..E_2$,其中$\alpha E_1$和$\alpha E_1.\beta$都满足条件$C$,则称序列$s$覆盖该事件约束.
- $m[[E_1;\rightarrow E_2]]_UC$:若$s$中包含一个事件前缀$\alpha E_1.\beta..E_2$,其中$\alpha E_1$和$\alpha E_1.\beta$都满足条件$C$,则称序列$s$覆盖该事件约束.若$s$中包含事件前缀$\alpha E_1$满足条件$C$,但其后不存在满足上述条件的$\beta..E_2$,则称序列$s$违反该事件约束.
- $\sim[[E_1;\rightarrow E_2]]_UC$:若$s$中包含事件前缀$\alpha E_1$,但不包含事件前缀$\alpha E_1.\beta..E_2$,或者$s$中包含该前缀,但$\alpha E_1$和$\alpha E_1.\beta$不同时满足条件$C$,则称序列$s$覆盖该事件约束.反之,若$s$中包含事件前缀$\alpha E_1.\beta.E_2$,且$\alpha E_1$和$\alpha E_1.\beta$同时满足条件$C$,则称序列$s$违反该事件约束.

3 实例研究

下面以生产者-消费者问题为例,研究基于E-CSPE约束的分布式程序测试.该问题由3个单元组成:生产者、消费者和仓库.其中"仓库"可认为是数据对象,"生产者"和"消费者"是端口对象.

事件是发生在端口的状态变迁,为定义关键事件和选择测试用例,可以画出$FSM_{生产者}$(finite state machine)

和 $FSM_{消费者}$，如图 1 所示.

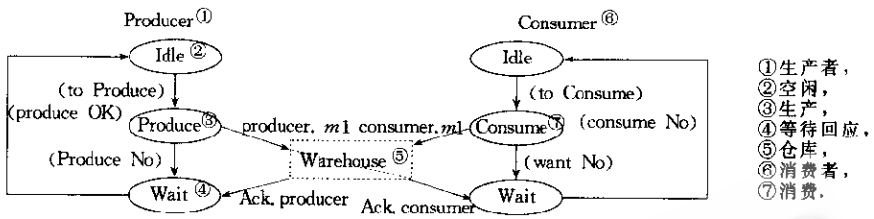

Fig. 1　The FSM presentation of producer-consumer problem
图1　生产者和消费者对象的自动机表示

对于生产者，图 1 中定义了 3 个事件："toProduce"，"produceNo"和"produceOK"；消费者也是 3 个事件："toConsume"，"wantNo"和"consumeNo". 考虑生产者和消费者间的同步关系图(图中用虚线表示)，可定义关键事件为"produceNo"和"consumeNo"，为确立事件的唯一性，可加上生产者和消费者以及产品的标识，即"produce-xNo-i"和"consume-yNo-j".

根据生产者-消费者问题的基本约定以及如图 1 所示的自动机表示，可按照 E-CSPE 约束描述列出以下事件约束(设仓库容量等于 N；每个生产者或消费者都有唯一的标识，用 x，y 表示，* 代表任一生产者或消费者；每个产品也有唯一的标识，用 i，j 表示，? 代表任一产品)：

B1. a[[#;→produce-*No-?]](true)

B2. ~[[#;→consume-*No-i]](事件 produce-*No-i 尚未发生)

B3. m[[produce-*No-i;→consume-*No-i]](true)

B4. a[[produce-*No-?;→produce-*No-?]](从 # 开始的 produce-*No-? 事件数——从 # 开始的 consume-*No-? 数<N)

B5. a[[consume-*No-?;→consume-*No-?]](从 # 开始 produce-*No-? 数>consume-*No-? 数)

B6. ~[[produce-*No-?;→produce-*No-?]](从 # 开始的 produce-*No-? 事件数——从 # 开始的 consume-*No-? 数=N)

B7. ~[[consume-*No-?;→consume-*No-?]](从 # 开始 produce-*No-? 数= consume-*No-? 数)

B8. a[[produce-*No-?;→consume-*No-?]](true)

B9. ~[[produce-*No-i;→produce-*No-i]](true)　(同一产品只能生产一次)

B10. ~[[consume-*No-i;→consume-*No-i]](true)　(同一产品只能消费一次)

B11. a[[consume-xNo-i;→consume-xNo-j]](子序列 produce-yNo-i.β..produce-yNo-j 已经发生)

根据规则 B10 和 B11，按照定理 3 可导出：

B12. ~[[consume-xNo-j;→consume-xNo-i]](子序列 produce-yNo-i.β..produce-yNo-j 已经发生)

为验证本文所提出方案的有效性，我们用 Java 语言实现了生产者-消费者问题. 其中，生产者和消费者为两个不同的线程类，"仓库"为一同步(synchronized)对象，生产者访问仓库使用"put()"方法调用，而消费者使用"get()"方法调用.

在实现中我们分别设置了 5 个程序错误：

(1) 完全没有同步，即仓库没有实现为同步对象；

(2) 生产者没有同步，即"put()"方法没有按照同步方式实现；

(3) 消费者没有同步，即"get()"方法没有按照同步方式实现；

(4) "put()"方法有实现错误；

(5) "get()"方法有实现错误.

其中错误(4)和错误(5)用作参考，因为它们可以通过传统的单元测试手段检测出来.

表 1 列出了采用不确定性测试时的测试结果(每一个测试用例分别执行 10 次再取平均结果，仓库容量 N=6).

Table 1 Test results of the producer-consumer problem

表 1 生产者-消费者问题的测试结果

Test case① / Fault type②	1 producer③ 1 consumer④	2 producers 2 consumers	10 producers 10 consumers	100 producers 100 consumers
1	Rules⑤: B2,B3,B7 Number of events⑥: 5	Rules: B2,B3,B7,B10 Number of events: 4	Rules: B2,B3,B6,B7,B10 Number of events: 5	Rules: B6 Number of events: 8
2	* Rules: B6 Number of events: 13 $N=2$	* Rules: B6 Number of events: 15 $N=2$	Rules: B6 Number of events: 128 $N=6$	Rules: B6 Number of events: 8 $N=6$
3	Rules: B3,B7,B10 Number of events: 22	Rules: B3,B7,B10,B12 Number of events: 49	Rules: B2,B3,B7,B10,B12 Number of events: 58	Rules: B7,B10 Number of events: 64
4	Rules: B3 Number of events: 6	Rules: B3 Number of events: 6	Rules: B3 Number of events: 8	Rules: B3 Number of events: 10
5	Rules: B3 Number of events: 5	Rules: B3,B10,B12 Number of events: 7	Rules: B3 Number of events: 6	Rules: B10 Number of events: 16

Here the "rules" denote the set of E-CSPE rules violated when the first error was found.

* Denotes that the size of the warehouse is not the default value. For example, $N=2$

其中的规则是指每次执行测试用例首次发现错误时所违反的规则.

* 表示仓库容量不是缺省值,如 $N=2$

①测试用例,②错误类型,③生产者,④消费者,⑤规则,⑥平均事件数.

为节省篇幅,表中没有列出对事件约束的覆盖情况.按照表中结果,除少数情况外(错误 2),即使在比较复杂的情况下,如有 100 个生产者和 100 个消费者,本测试方案也可以有效地发现程序出错(一般所需事件数量<70).对于错误 2,如果使用确定性测试,如生产者连续生产 7 次而消费者不动作,就可以发现错误.可见,基于 E-CSPE约束描述的测试方案是行之有效的.

4 结论和相关工作

基于事件约束的测试技术早期被用于 Ada 语言程序的跟踪调试.Carver 等人[3]应用CSPE 描述事件约束,并采用 CSPE-1 准则判断事件序列对 CPSE 约束的覆盖和一致性程度.本文所做的工作是在其基础上的进一步细化.

Chen 等人[4]提出了一个通过执行环境(测试用例)判断对象等价性(正确性)的测试技术.该技术使用针对被测对象等价的执行序列(输入事件),再根据输出序列(输出事件)判断结果对象的等价性.

分布式程序测试的主要难点是并发导致的不确定性和同步问题.对于单个的分布单元可以利用已有的串行程序测试技术对其进行充分的测试.但当这些单元相互协作时就必须考虑新的测试技术和方案.基于事件约束的测试技术是一个良好的选择.本文从程序规约角度出发,明确事件是通过端口感知的程序内部状态变化,再提出E-CSPE 约束描述来定义事件间的依赖关系,即事件约束.当基于程序规约的事件约束所描述的事件集不同于用户定义的关键事件集时,可以根据 E-CSPE 约束规则间的关系推导出新的事件约束.根据端口的状态变迁覆盖可以选择测试用例,执行测试用例一般有两种方式:确定性测试和不确定性测试.通过执行测试用例所产生的事件序列集对规约约束集的一致性和覆盖程度可以检测被测程序的正确性.

参考文献

1 Jorgensen P C. Software Testing-A Craftsman's Approach. Boca Raton, Florida: CRC Press, 1995

2 Frankl P G, Hamlet R G, Littlewood B *et al*. Evaluating testing methods by delivered reliability. IEEE Transactions on Software Engineering, 1998,24(8):586~601

3 Carver R H, Tai Kou-chung. Use of sequencing constraints for specification based testing of concurrent programs. IEEE Transactions on Software Engineering, 1998,24(6):471~490

4 Chen H Y, Tse T H, Chan F T *et al*. In black and white: an integrated approach to class level testing of object oriented programs. ACM Transactions on Software Engineering and Methodology, 1998,7(3):1～41

Validation Test of Distributed Program Based on Event Sequencing Constraints

GU Qing CHEN Dao-xu YU Meng XIE Li SUN Zhong-xiu

(*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)

Abstract Because of concurrency and non-determinism, the inner-states of a running distributed program should be considered a part from the start and the end states, when the program is to be tested and the validated. According to specification based testing, these inner-states will tranform into event sequences through the ports of the program, and the co-relations among these events, i.e. the set of event sequencing constraints should be provided by the program's specification. In this paper, the authors introduce the E-CSPE (extended-constraints on succeeding and preceding events), a method to formalize these event constraints. The E-CSPE includes three basic discription rules, which correspond to three different types of such event constraints. Based on the consistency with and coverage of these event constraints by the event sequences produced by the program execution, the validity of the distributed implementation can be tested.

Key words Distributed program testing, finite state machine, port, CSPE (constraints on succeeding and preceding events), event sequencing constraint.