ASRSP——一种新的并发程序测试准则*)

卢炎生　卢　超
(华中科技大学计算机科学与技术学院　武汉 430074)

摘　要　可达性测试是目前较为成熟的一种并发程序测试方法，该方法解决了如何生成最小完备偏序测试序列集的问题。但研究表明，对于一般规模的并发程序，这一测试序列集仍然太大，以至穷尽测试无法完成。因此，目前亟需能投入实际应用的并发程序测试准则和相应的测试序列生成算法。本文提出了一种实用性较高的并发程序测试准则：全发送接收语句对(ASRSP)，并针对该准则提出了一种新的并发程序测试方法：全发送接收语句对可达性测试(ASRSP-RT)。该方法利用可达性测试生成测试序列集的完备性来保证覆盖所有的发送接收语句对，并在每次生成新序列之后及时去掉对覆盖剩下发送接收语句对无作用的序列，从而达到约简测试序列集的目的。

关键词　软件测试，测试准则，可达性测试，并发，全发送接收语句对

ASRSP: A New Testing Criterion for Concurrent Programs

LU Yan-Sheng　LU Chao
(Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　Reachability testing is a general approach to testing concurrent programs via generating the minimal complete partial ordered set of test sequences. Recent work shows, however, that it generally suffers from its scalability due to the large set size. As a result, it is infeasible even for normal scale concurrent programs in practice. To address this problem, the necessity is to establish appropriate testing criterions as well as introduce corresponding algorithms to generate test sequences. In this paper, we present ASRSP, a testing criterion for concurrent programs. Based on ASRSP, we further present ASRSP-RT, a novel algorithm to generate test sequences. Our evaluation shows that compared to traditional methods, the proposed ASRSP-RT can significantly reduce the set the test sequences as a result of satisfying the ASRSP criterion.

Keywords　Software testing, Testing criterion, Reachability testing, Concurrency, ASRSTP

1　引言

并发程序通常包含两个或多个并发执行的线程，这些线程通过相互合作来完成某些任务。一方面，并发程序允许使用并行机制提高计算效率，在当前有着广泛的应用；另一方面，并发程序的测试非常困难，因为并发程序在执行时的行为具有不确定性，使用同一测试输入执行被测程序两次，可能会得到两个不同的输出结果。因此，在现阶段对并发程序的测试方法进行深入研究有非常重要的意义。

并发程序的测试方法可以按测试策略分为不确定性测试[1](non-deterministic testing)和确定性测试[2](deterministic testing)两种。不确定性测试在给定程序输入的前提下随机地执行被测程序，这种测试方法实现简单，但效率不高[3,4]。确定性测试同时给定程序输入和同步序列(SYN-sequence)，迫使程序执行给定的同步序列，这种测试方法虽然可以重演并发程序的执行过程，但同步序列的生成方法和选择策略仍然是一个未解决的难题[5,6]。可达性测试[7](reachability testing)结合了不确定性测试和确定性测试的优点，动态生成被测程序的同步序列，而且在给定输入的前提下能够穷尽被测程序的可行(feasible)同步序列，具有很高的实用性，已成为近年来并发程序测试领域的一个研究热点。

2　研究现状

可达性测试的概念最初由 Hwang 等人提出，即通过穷尽共享变量读写操作的所有可能交替序列来完成并发程序的测试。虽然该方法只能处理简单的基于共享变量的并发程序，但其思想为并发程序的测试方法开辟了一个新的研究领域[7]。随后，Tai 针对异步消息通信程序的特征重新定义了可达性测试方法中的一些概念，并改进了相关算法，有效提高了可达性测试方法的实用性[8]。近年来，Lei 和 Carver 等人对可达性测试方法进行了持续而深入的研究，获得了一系列突破性进展。首先，Lei 和 Tai 提出了一种高效的竞争变体(race variant)计算方法，该算法能够一次计算出一个同步序列的所有竞争变体[9]。接着，Lei 和 Carver 将可达性测试方法扩展到了基于信号量的并发程序上[10]。随后，Carver 和 Lei 提出了一个通用并发程序模型，该模型屏蔽了采用不同的同步结构实现的并发程序之间的差异，从而可以将同一个可达性测试算法应用于不同类型的并发程序[11]。不久，Lei 和 Carver 再次修改了可达性测试算法，新的算法能够保证不执行重复的同步序列，且无需存储执行过的同步序列，从而有效提高了测试的效率[12]。最近，Lei 和 Carver 总结了所有这些研究成果，详细描述了并发程序可达性测试框架及相关算

*)国家"十一五"部委预研基金(No. 513150601)；湖北省自然科学基金(No. 2005ABA255)。**卢炎生**　教授，博士生导师，主要研究领域为软件工程、数据库系统；**卢　超**　博士生，主要研究领域为软件测试。

法，对算法的正确性进行了证明，并给出了算法性能评价的相关测试数据[13]。在国内，Li 和 Chen 等人也针对 Java 多线程程序提出了自己的可达性测试框架，并给出了同步序列的生成和选择策略[14]。

到目前为止，可达性测试技术已经基本成熟，在给定被测程序和测试输入的前提下，Lei 等人的算法已经可以保证穷尽所有可行的序列，而且该方法直接处理事件之间的偏序关系，无需任何偏序约简技术[15~17]，从而直接生成最小完备测试序列集。但是，可达性测试技术的实用性目前仍不理想，这主要是由并发程序的复杂性所致。根据文[13]的测试数据，采用 SUMonitor 实现的哲学家就餐程序，当哲学家的数目增至 5 个时，生成的测试序列就多达 19330 个。对于这种小规模的被测程序，这一测试序列集显然太大。该文进一步指出，穷尽测试通常无法做到，目前急需制定一些测试准则，以及相关的测试序列选择算法，以约简测试序列集，提高可达性测试的实用性。虽然 Taylor 曾针对并发程序提出了一组基于并发状态图的测试准则[18]，但由于并发状态爆炸问题[19]至今没有解决，对于一般规模的并发程序根本无法构造出其并发状态图，使得这一组测试准则无法真正应用于实践。

针对上述问题，本文提出了一种实用性较高的并发程序测试准则：全发送接收语句对(ASRSP)，并针对该准则提出了一种新的并发程序测试方法：全发送接收语句对可达性测试(ASRSP-RT)。该方法利用可达性测试生成的测试序列集的完备性来保证覆盖所有的发送接收语句对，并在每次生成新的序列之后都及时约简掉对覆盖剩下发送接收语句对无作用的序列，使得算法能生成较小的测试序列集。

3 ASRSP 准则

根据 Lei 的可达性测试通用执行模型[11, 13]，所有的并发事件都可被统一为发送事件和接收事件，并将发送事件和接收事件之间的同步称为同步对(SYN-Pair)。事件是由执行语句产生的。本文将执行后产生发送事件的语句定义为发送语句，将执行后产生接收事件的语句定义为接收语句。可以认为，所有并发程序中的同步语句最终都可以转换为发送或者接收语句，因此本文提出的测试准则和相应的测试方法可以适用于所有类型的并发程序。

定义 1 如果并发程序 CP 中的接收语句 s_r 能够接收发送语句 s_s 发出的消息，则称$\langle s_s, s_r\rangle$为一个发送接收语句对，简称 *SRSP*，*CP* 的所有 *SRSP* 构成集合 $SRSPs(CP)$。

一般情况下，发送语句与接收语句通过端口进行匹配，如果发送语句 s_s 与接收语句 s_r 使用同一端口，则认为 s_r 可以接收 s_s 发出的消息。因此，对被测程序进行语法分析，可以计算出该程序的所有 *SRSP*。

定义 2 设使用输入 X 执行并发程序 *CP* 得到同步序列 Q，如果 Q 中存在同步对$\langle e_s, e_r\rangle$，e_s 由语句 s_s 产生，e_r 由语句 s_r 产生，则称对于输入 X，序列 Q 能覆盖 $SRSP\langle s_s, s_r\rangle$。

定义 3 使用输入 X，如果穷尽了并发程序 *CP* 的所有可行序列都无法覆盖某 *SRSP*，则称该 *SRSP* 不可覆盖。

定理 1 对于并发程序 *CP* 和测试输入 X，如果可达性测试过程没有覆盖 *CP* 的某 *SRSP*，则该 *SRSP* 不可覆盖。

Lei 已证明在给定测试输入的前提下，可达性测试可以穷尽被测程序的所有同步序列[13]，因此根据定义 3 可知定理 1 成立。

定义 4 定义测试准则为谓词 C，对于并发程序 *CP*、测试输入 X 和测试序列集 $SEQs$，当且仅当 $C(CP, X, SEQs)$ 为真时，$(CP, X, SEQs)$满足测试准则 C。

定义 5 对于并发程序 *CP*、测试输入 X 和测试序列集 $SEQs$，$(CP, X, SEQs)$满足全发送接收语句对(ASRSP)准则，当且仅当任给 $srsp \in SRSPs(CP)$，要么存在 $seq \in SEQs$，seq 能覆盖 $srsp$，要么 $srsp$ 不可覆盖。

定理 2 设使用输入 X 对并发程序 *CP* 进行可达性测试得到测试序列集 $SEQs$，则$(CP, X, SEQs)$满足准则 ASRSP。

根据定理 1，任给 $srsp \in SRSPs(CP)$，可达性测试过程要么能够覆盖 $srsp$，要么 $srsp$ 不可覆盖。因此，根据定义 5，$(CP, X, SEQs)$满足 ASRSP 准则，定理 2 成立。

虽然可达性测试可以达到 ASRSP，但一般情况下，达到 ASRSP 所需的测试序列集的规模远小于可达性测试所生成的测试序列集，因此本文针对 ASRSP 准则提出了 ASRSP-RT 测试方法。

4 ASRSP-RT 测试

ASRSP-RT 测试整体上可分为静态分析和动态执行两部分，其框架如图 1 所示。对于被测程序 CP，静态分析负责其确定 $SRSPs$ 等集合，动态执行负责生成和执行测试序列。

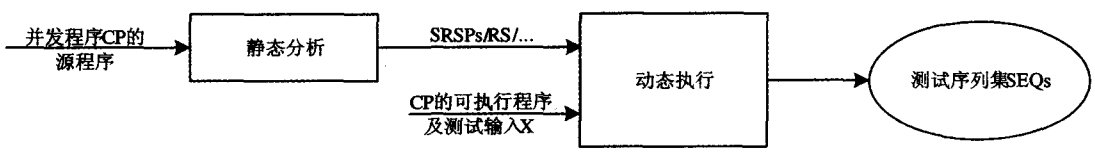

图 1 ASRSP-RT 测试框架

4.1 静态分析

设并发程序 *CP* 由 n 个线程组成，$Thread[i](1 \leqslant i \leqslant n)$表示 *CP* 的第 i 个线程，静态分析部分主要计算出如下集合：

$SRSPs$：*CP* 的所有 *SRSP* 构成的集合；

S：*CP* 所有 *SRSP* 的语句构成的集合；

RS：S 中所有接收语句构成的集合；

$RSB[s](s \in S)$：与 s 在同一线程内，且在 s 之前的接收语句的集合。

上述集合的计算可以通过基本语法分析完成。如无特别说明，本文中所有语句的集合都是 S 的子集，不考虑 S 以外的同步语句是因为这种语句会造成无法同步的程序错误。

4.2 动态执行

动态执行主要由算法 modifiedReachabilityTesting(简称 mRT)，modifiedConstructRaceTable(简称 mCRT)和 updateSetsAndRaceTable(简称 uSART)构成。

算法 mRT 完成动态执行部分的主要框架，如图 2 所示。

原可达性测试算法[13]从随机执行被测程序收集初始序列开始，然后根据收集的序列计算变体，再对变体执行基于前缀的测试得到新的序列，不断重复这个过程直到没有新的变体生成，算法结束。算法 mRT 对原算法进行了四处修改：

```
1. function modifiedReachabilityTesting(CP: a concurrent program, X: an input of CP){
2.     collect a SYN-sequence Q0 by executing CP with input X non-deterministically;
3.     let rt0 be an empty RaceTable;
4.     updateSetsAndRaceTable(Q0, rt0);
5.     if (SRSPs = Φ) return;
6.     Qs = {Q0};
7.     while (Qs != Φ){
8.         withdraw a Q from Qs;
9.         rt = modifiedConstructRaceTable(Q, RS)
10.        while(rt! = Φ){
11.            withdraw a row from rt;
12.            collect a SYN-sequence Q' by conduct a modified prefix-based testing with (Q, row);
13.            updateSetsAndRaceTable(Q', rt);
14.            if (SRSPs = Φ) return;
15.            Qs = Qs ∪ {Q'};
16.        }
17.    }
18.}
```

图 2　算法 modifiedReachabilityTesting

第一，在整体结构控制上，新算法逐个处理记录的序列，对于每个序列计算并处理其竞争表，在该序列的竞争表没有处理完之前，不处理新的序列，这一修改体现于第 7 行和第 10 行构成的嵌套循环结构；

第二，新算法采用了新的竞争表构造算法（mCRT），算法 mCRT 只处理与 RS 集合中的接收语句对应的接收事件，对于其他的接收事件不予考虑，因为如何处理这些事件的竞争条件对于覆盖剩下的 SRSP 不起作用，这一修改体现于第 9 行；

第三，新算法每收集到一个新的序列之后，都会使用该序列来更新集合 *SRSPs*、*RS*，以及当前正在处理的竞争表，这是因为新的序列可能已经完成了对被测程序中某些 *SRSP* 的覆盖，此时应更新 *SRSPs* 以测试 *SRSPs* 是否为空，更新 *RS* 使得下次计算竞争表时 *RS* 更小，并更新当前竞争表以去掉对覆盖剩下的 *SRSP* 不起作用的行，这一修改体现于第 4 行和第 13 行；

第四，新算法在每次收集新序列并更新 *SRSPs* 之后，都会判断 *SRSPs* 是否为空集，*SRSPs* 为空集表示已达到 ASRSP 准则，算法结束，如果算法在执行完 Q_S 中的所有序列后结束，则 *SRSPs* 中剩下的 *SRSP* 是不可达 *SRSP*，这种情况仍视为达到 ASRSP 准则，这一修改体现于第 5 行和第 14 行。

算法 mCRT 针对 RS 中竞争集非空的接收事件计算竞争表，而原竞争表构造算法是对所有竞争集非空的接收事件构造竞争表，这一修改较为简单直接，限于篇幅，此处略去。

算法 uSART 负责更新 *SRSPs*、*RS* 及竞争表，如图 3 所示。

该算法逐个处理同步序列 *Q* 中的同步对，对于同步对 *sp*，第 7 行从 *SRSPs* 中删除与之相应的 *SRSP* 〈*s*, *r*〉，因为该 *SRSP* 已被 *sp* 覆盖；设 *Q* 中由语句 *s* 执行产生的事件构成集合 Events(*Q*, *s*)，第 8～16 行计算语句集合 *RSR*，对 *RSR* 中的任意语句 *st*，*Events*(*Q*, *st*) 中的事件会影响 *Events*(*Q*, *s*) ∪ *Events*(*Q*, *r*) 中的事件，但不会影响任何未覆盖的 SRSP 中的语句在 *Q* 中的事件，第 9～12 行先确定 *RSR* 的范围，第 13～16 行从 *RSR* 去掉对集合 *SNC* 中的语句有影响的语句，*SNC* 是由剩下未覆盖的 *SRSP* 中的语句构成的；第 17 行更新 *RS*，新的 *RS* 将用于下一次竞争表的计算；第 18～20 行根据 *RSR* 约简当前竞争表，从竞争表中去掉 *RSR* 中的语句对应的事件，以及这些事件控制的事件，然后去掉竞争表中由于列删除而产生的重复行。

```
1. updateSetsAndRaceTable(SYN-sequence Q, RaceTable rt){
2.     let SP be the set of all the SYN-Pairs of Q;
3.     while(SP!= Φ && SRSPs!= Φ){
4.         withdraw a sp from SP;
5.         let (s, r) be the corresponding SRSP of sp;
6.         if ((s, r) ∈ SRSPs){
7.             SRSPs = SRSPs – {(s,r)}; //update SRSPs
8.             RSR = Φ; //RSR means Receiving Statements that can be Removed from RS;
9.             if(s is not in any SRSP of SRSPs)
10.                RSR = RSR ∪ RSB[s];
11.            if(r is not in any SRSP of SRSPs)
12.                RSR = RSR ∪ RSB[r] ∪ {r};
13.            let SNC be the set of Statements in the SRSPs;
14.            foreach(r' ∈ RSR)
15.                if (∃s'∈SNC, r' ∈ RSB[s'])   //SNC means Statements Not Covered;
16.                    RSR = RSR – {r'};
17.            RS = RS – RSR;      //update RS;
18.            let es be the corresponding events of statements in RSR;
19.            remove the columns of es and events controled by es from rt;
20.            remove the duplicated rows from rt;
21.        }
22.    }
23.}
```

图 3　算法 updateSetsAndRaceTable

5 算法分析

算法 mRT 的时间复杂度为 $O(m\times|Emax|^2\times|Vmax|)$，其中 m 为该算法产生的序列数，$|Emax|$ 为最大序列事件数，$|Vmax|$ 为最大序列变体数。文[13]已给出构造竞争表的时间复杂度为 $O(|R|^2\times|V|)$，其中 R 是竞争表表头接收事件的个数，$|V|$ 是序列 Q 产生的竞争变体的个数。由于算法 mCRT 只针对 RS 中的接收事件计算竞争表，因此其时间复杂度为 $O(|RnRS|^2\times|V|)$，算法 mRT 消耗于计算竞争表的时间复杂度为 $O(m\times|Emax|^2\times|Vmax|)$。对于算法 *uSART*，主要考虑其更新 $SRSPs$、RS 和当前竞争表的时间复杂度。对于整个测试过程，总共会进行 $|SRSPs|$ 次约简，每次约简计算 RSR 的时间复杂度为 $O(|RSR|\times|SNC|)$。根据 RSR 和 SNC 的定义，$|SNC|=2|SRSPs|$，$|RSR|<|RS|=|SRSPs|$，因此，算法 mRT 消耗于约简序列和计算覆盖的时间复杂度为 $O(|SRSPs|^3)$。由于并发程序的可行同步序列数随程序规模呈指数增长，即随着程序规模的增长，$m\times|Emax|^2\times|Vmax|$ 将远大于 $|SRSPs|^3$。综上，算法 mRT 的时间复杂度为 $O(m\times|Emax|^2\times|Vmax|)$。

算法 mRT 是可结束的。原可达性测试算法已被证明是可结束的[13]，而新算法只是对原算法的序列生成过程进行剪枝处理，以生成更少的序列，即新算法生成的序列集是原算法在相同情况下生成的序列集的子集，因此新算法在遍历完有效序列之后，一定会结束。

定理 3 设使用输入 X 对并发程序 CP 进行 ASRSP-RT 测试得到测试序列集 $SEQs$，$(CP, X, SEQs)$ 满足 ASRSP 准则。

证明：对于 CP 中的任意 $SRSP\langle s_s, s_r\rangle$ 和算法 mRT 生成的随机序列 Q_0，如果 $SRSP\langle s_s, s_r\rangle$ 未被 Q_0 中的同步对覆盖，那么算法 uSART 会将 $RSB[s_s]$ 和 $RSB[s_r]$ 中的语句以及 s_r 本身都保留在 RS 中，因此，在 $Events(Q_0, s_s)\cup Events(Q_0, s_r)$ 中任意事件之前发生的事件都不会从竞争表中约简掉，这保证了所有可能影响语句 s_s 和语句 s_r 同步的 Q_0 的竞争变体都被保留下来以生成新的序列，直到算法 mRT 产生某序列 Q 覆盖 $SRSP\langle s_s, s_r\rangle$，或者算法 mRT 结束，此时已穷尽可能影响语句 s_s 和语句 s_r 同步的序列，即 $SRSP\langle s_s, s_r\rangle$ 不可被覆盖。因此，$SRSPs(CP)$ 中的语句对要么能被 $SEQs$ 中的某序列覆盖，要么该语句对不可覆盖，根据定义 5，$(CP, X, SEQs)$ 满足 ASRSP 准则，证毕。

一般情况下，给定被测程序 CP 和测试输入 X，构造满足 ASRSP 准则的序列集所需要的测试列列数远小于完成可达性测试需要的测试序列数。不失一般性，考虑并发程序 CP 的如下程序片断，设线程 $T(r)$ 中有接收语句集 $R=\{r_1, r_2, \cdots, r_k\}$，另有 k 个线程中的 k 个发送语句构成集合 $S=\{s_1, s_2, \cdots, s_k\}$，任给 $r\in R, s\in S, (s, r)$ 构成一个 $SRSP$。对于上述程序片断，完成可达性测试需要 $k!$ 个序列，而构造满足 ASRSP 准则的最小测试序列集只需要 k 个序列。任何并发程序均存在上述程序片断，如果一个并发程序不存在上述程序片断，则执行该并发程序得到的同步序列将没有竞争集存在，即该并发程序的同步结构不会引入执行时的不确定性，对它进行可达性测试是没有意义的。虽然 ASRSP-RT 测试还不能生成满足 ASRSP 准则的最小测试序列集，但由于它在每次执行一个测试序列之后，都会进行剪枝以去掉对覆盖剩下的 SRSP 无意义的竞争变体，其生成的序列数必然远小于完成可达性测试所需的序列数。

6 案例

考虑图 4(a)所示典型并发程序 CP_1，采用 Lei 的可达性测试方法，其测试过程如图 4(b)所示。

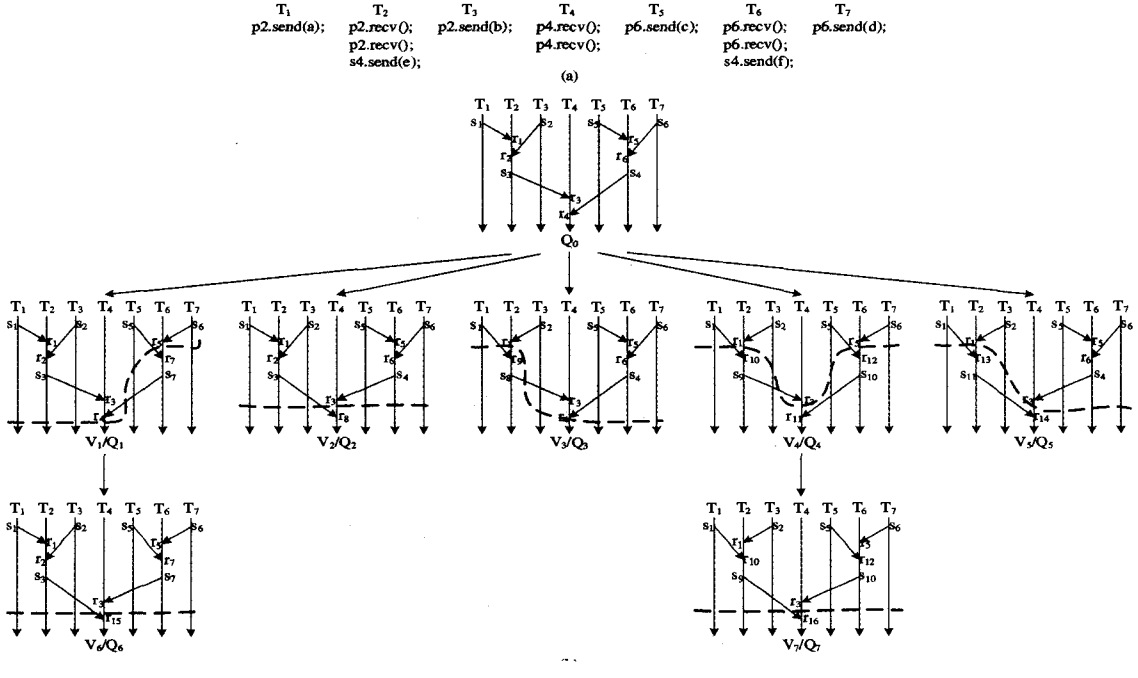

图 4 并发程序 CP_1 及其可达性测试过程

本文使用时空图(space-time diagram)来描述并发程序的执行过程(即同步序列)。时空图使用垂直向下的箭头表示线程，连接线程之间的箭头表示事件的同步。测试程序首先对 CP_1 进行不确定性测试得到同步序列 Q_0，然后计算 Q_0 的竞

争变体得到 V_1，V_2，V_3，V_4 和 V_5，分别对应中间五个时空图虚线以上的部分，再使用这五个竞争变体分别对 CP_1 执行基于前缀的测试(prefix-based testing)得到新的同步序列 Q_1，Q_2，Q_3，Q_4 和 Q_5。分析 Q_1 得到变体 V_6，分析 Q_2 和 Q_3 都无法再得到新的变体，分析 Q_4 得到变体 V_7，分析 Q_5 无法得到新的变体，于是对 V_6 和 V_7 执行基于前缀的测试得到 Q_6 和 Q_7。再分析 Q_6 和 Q_7，均无法得到新的变体，可达性测试过程结束，共测试 8 个序列。

对图 4(a)所示并发程序 CP_1 使用本文算法进行 ASRSP-RT 测试，其过程如图 5 所示。

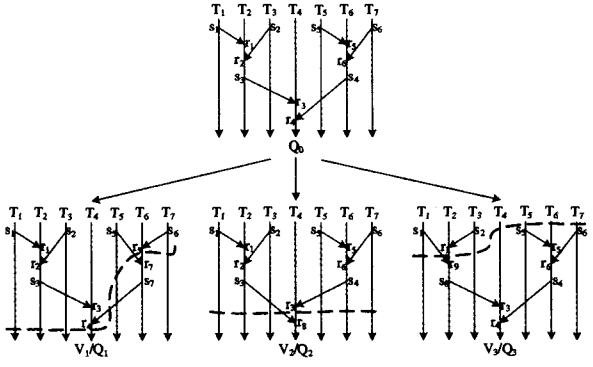

图 5　并发程序 CP_1 的 ASRSP-RT 测试过程

设 s_{ij} 为第 i 个线程的第 j 条同步语句，先对并发程序 CP_1 执行静态分析，得到如下集合：

$SRSPs=\{(s_{11}, s_{21}), (s_{11}, s_{22}), (s_{23}, s_{41}), (s_{23}, s_{42}), (s_{31}, s_{21}), (s_{31}, s_{22}), (s_{51}, s_{61}), (s_{51}, s_{62}), (s_{63}, s_{41}), (s_{63}, s_{42}), (s_{71}, s_{61}), (s_{71}, s_{62})\}$；

$S=\{s_{11}, s_{21}, s_{22}, s_{23}, s_{31}, s_{41}, s_{42}, s_{51}, s_{61}, s_{62}, s_{63}, s_{71}\}$；

$RS=\{s_{21}, s_{22}, s_{41}, s_{42}, s_{61}, s_{62}\}$；

根据本文 4.1 节对集合 RSB 的定义，其计算并不复杂，限于篇幅，这里不一一列出。

表 1　Q_0 的竞争表的初始状态

r_1	r_3	r_5
0	0	1
0	1	0
1	0	0
1	0	1
1	1	0

然后对 CP_1 执行算法 mRT：首先得到同步序列 Q_0，使用同步序列 Q_0 和空竞争表执行算法 uSART，使得 $SRSPs$ 被更新为$\{(s_{11}, s_{22}), (s_{23}, s_{42}), (s_{31}, s_{21}), (s_{51}, s_{62}), (s_{63}, s_{41}), (s_{71}, s_{61})\}$，$RS$ 不变。分析 Q_0 和 RS 得到竞争表如表 1 所示，取第一行 row_1 后当前竞争表如表 2 所示，Q_0 和 row_1 构成变体 V_1，执行 V_1 后得到序列 Q_1，然后使用 Q_1 和表 2 所示竞争表执行算法 uSART，使得 $SRSPs$ 被更新为$\{(s_{11}, s_{22}), (s_{23}, s_{42}), (s_{31}, s_{21}), (s_{63}, s_{41})\}$，$RS$ 不变，因此当前竞争表不变；取当前竞争表(表 2)的第一行 row_2 后竞争表如表 3 所示，Q_0 和 row_2 构成变体 V_2，执行后得到序列 Q_2，然后使用 Q_2 和表 3 所示竞争表执行算法 uSART，使得 $SRSPs$ 被更新为$\{(s_{11}, s_{22}), (s_{31}, s_{21})\}$，$RS$ 被更新为$\{s_{21}, s_{22}\}$，此时计算出 $RSR=\{r_3, r_4, r_5, r_6\}$，可约简当前竞争表(表 3)，首先删除列 r_3 和 r_5，然后合并剩下行，得到表 4。表 4 只剩下 1 行，取该行得到变体 V_3，执行后得到 Q_3，使用 Q_3 和空竞争表执行算法 uSART，使得 SRSPs 被更新为空集，于是 ASRSP-RT 测试完成，共测试 4 个序列。

表 2　取出第一行之后的竞争表

r_1	r_3	r_5
0	1	0
1	0	0
1	0	1
1	1	0

表 3　取出第一行之后的竞争表

r_1	r_3	r_5
1	0	0
1	0	1
1	1	0

表 4　约简之后的竞争表

r_1
1

结束语　本文针对并发程序测试提出了一种实用性较高的测试准则，ASRSP 准则，并针对该准则提出了 ASRSP-RT 测试方法。该方法继承了原可达性测试产生偏序序列和不产生重复序列的优点，利用可达性测试生成测试序列集的完备性来保证覆盖被测程序的所有 SRSP，并在每次生成新的序列之后都及时约简掉对覆盖剩下 SRSP 无作用的序列，使得算法能生成较小的测试序列集。目前对并发程序测试准则的研究尚处于初期阶段，就 ASRSP 准则及 ASRSP-RT 测试方法而言，我们下一步的工作将主要集中于如何改进 ASRSP-RT 测试方法使之能生成满足 ASRSP 准则的最小测试序列集。

参 考 文 献

1　Tai K C. Testing of concurrent software. In: Proceedings of the 13th Annual International Computer Software and Applications Conference. Orlando, 1989. 62～64

2　Taylor R N. A General-purpose Algorithm for Analyzing Concurrent Programs. Communications of the ACM, 1983, 26(5): 362～376

3　Edelstein O, Farchi E, Nir Y, et al. Multithread Java Program Test Generation. IBM Systems J, 2002, 41(1): 111～125

4　Stoller S D. Testing Concurrent Java Programs Using Randomized Scheduling. Electr Notes Theor Comput Sci, 2002, 70(4): 143～158

5　Yang R D, Chung C G. A Path Analysis Approach to Concurrent Program Testing. Information and Software Technology, 1992, 34(1): 43～56

6　Yang C, Souter A L, Pollock L L. All-du-Path Coverage for Parallel Programs. ACM SIGSOFT Software Engineering Notes, 1998, 23(2): 153～162

7　Hwang G H, Tai K C, Huang T L. Reachability testing: An approach to testing concurrent software. International Journal of Software Engineering and Knowledge Engineering, 1995, 5(4): 493～510

8　Tai K C. Reachability testing of asynchronous message-passing programs. In: Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems. NW Washington, DC USA, 1997. 50～61

9　Lei Y, Tai K C. Efficient reachability testing of asynchronous message passing programs. In: Proceedings of the 8th IEE Int'l Conference on Engineering for Complex Computer Systems. NW Washington, DC USA, 2002. 35～44

10　Lei Y, Carver R. Reachability testing of semaphore-based programs. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04). NW Washington, DC USA, 2004. 312～317

11 Carver R, Lei Y. A general model for reachability testing of concurrent programs. In: Davies J, et al. ed. Proceedings of the 6th International Conference on Formal Engineering Methods, Formal Methods and Software Engineering. Seattle, WA, USA: Springer, 2004. 76～98

12 Lei Y, Carver R. A New Algorithm for Reachability Testing of Concurrent Programs. In: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05). NW Washington, DC USA, 2004. 346～355

13 Lei Y, Carver R H. Reachability testing of concurrent programs. IEEE Transations on Software Engineering, 2006, 32(6): 382～403

14 Li S Q, Chen H Y, Sun Y X. A framework of reachability testing for Java multithread programs. SMC(3) 2004. 2730～2734

15 Godefroid P. Software Model Checking: The VeriSoft Approach. Formal Methods in System Design, 2005, 26(2): 77～101

16 Havelund K, Pressburger T. Model Checking Java Programs Using Java PathFinder. Int'l J Software Tools for Technology Transfer (STTT), 2000, 2(4): 366～381

17 Flanagan C, Godefroid P. Dynamic Partial Order Reduction for Model Checking Software. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, NY USA, 2005. 110～121

18 Taylor R N, Levine D L, Kelly C D. Structural Testing of Concurrent Programs. IEEE Transations on Software Engineering, 1992, 18(3): 206～214

19 Ulrich A, Konig H. Specification-based Testing of Concurrent Systems. In: Proceedings of the IFIP Joint Int'l Conf Formal Description Techniques and Protocol Specification, Testing, and Verification (FORTE/PSTV '97). London, 1997. 7～22

(上接第 231 页)

算式 $f-f\Theta B$ 和图像的真实边缘较为吻合，f 的第一个低灰度值和其后的几个高灰度值都成功的被检测到，由于经过腐蚀运算后，目标区域中的灰度值变化较小，而边缘部分的灰度值降低较多，然后用原图像减去腐蚀运算的结果这样边缘部分的灰度值就会明显比区域内的灰度值高，亦即边缘部分的灰度值得到了提高，而非边缘部分的值变化幅度不太大，也就起到了边缘检测的效果。$[f\oplus B]-[f\Theta B]$算式称为形态学梯度计算，从曲线可以看出，该算式对于边缘的细节并不敏感，并不适合检测细小的边缘和边缘密度比较大的图像[6]。

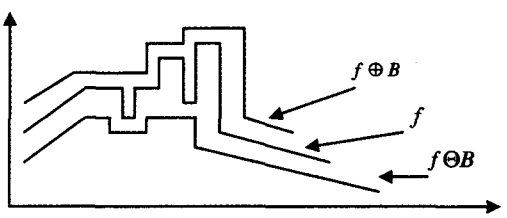

图 3　膨胀、腐蚀和原始曲线的对比

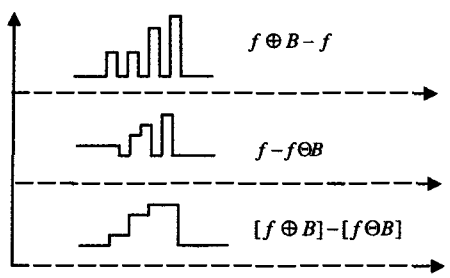

图 4　三种算式的对比

总的来说，分别采用$[f\oplus B]-[f\Theta B]$、$f\oplus B-f$ 和 $f-f\Theta B$ 三种形态学算子对线云和其周围的像素进行模板运算。三个算式中，$f\oplus B-f$ 和 $f-f\Theta B$ 算式适合于检测较细并且密集的边缘，而$[f\oplus B]-[f\Theta B]$提取的边缘较厚，适合于对不太密集且较厚的边缘进行提取。

4　试验分析

选取某城市的郊区机场遥感图像作为样本如图 5(a) 所示，将本文算法对其进行试验。图 5(b)是 Canny 算法的检测结果，图 5(c)是 Prewitt 算法的检测结果，图 5(d)～图(f)是本文算法的检测结果，其中图 5(d)是$[f\oplus B]-[f\Theta B]$算子对线云运算提取的结果，图 5(e)是 $f\oplus B-f$ 的提取结果，图 5(f)是 $f-f\Theta B$ 的提取结果。

根据对图 5 不同算法的检测结果对比分析可以看出，本文算法对于机场跑道、密集的公路和住宅区、海岸线检测的检测结果都较为理想。对线云进行模板运算时，[f⊕B]－[fΘB]更适合于提取稀疏的厚边缘，f⊕B－f 和 f－fΘB 则更适合于检测密集型边缘。

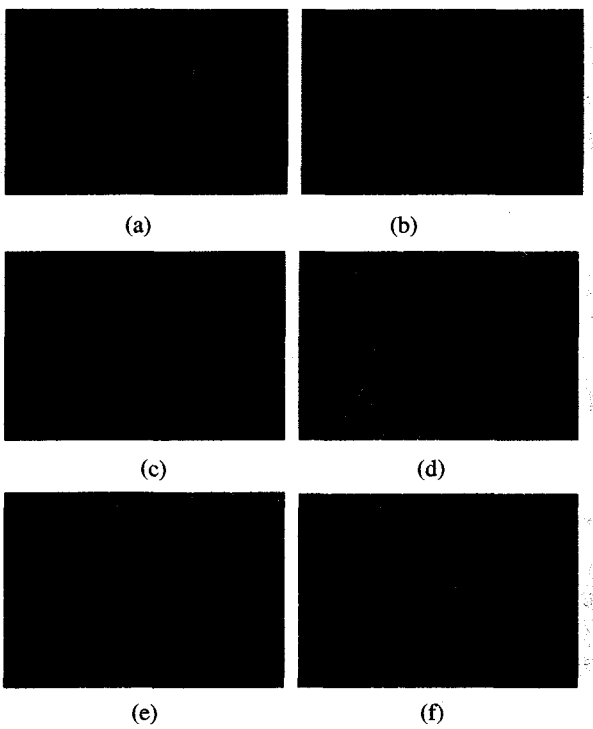

图 5　图像边缘检测对比图

结束语　利用云模型构建面云并计算面云之间相交得到的线云，可以对包含不确定信息的图像进行边缘检测。离云核心概念越近，其确定性越高，再结合数学形态学的检测算子，以隶属度作为因子进行模板运算，在云模型解决图像不确定性因素的同时，还能对兴趣目标的边缘进行分类检测。

参 考 文 献

1　雷丽珍. 数字图像边缘检测方法的探讨[J]. 测绘通报，2006，3：40～42

2　秦昆，李德毅，许凯. 基于云模型的图像分割研究[J]. 测量信息与工程，2006，31(5)：3～5

3　邸凯昌，李德毅，李德仁. 云理论及其在空间数据发掘和知识发现中的应用[J]. 中国图像图形学报，1999，11(4)：930～935

4　章琉晋. 图像分割[M]. 北京：科学出版社，2001

5　薛丽霞，王佐成，李永树，汪林林. 基于云模型的模糊边缘检测[J]. 西南交通大学学报，2006，41(1)：85～90

6　王宇，王乘，刘吉平. 一种基于数学形态学的遥感图像边缘检测算法[J]. 重庆邮电学院学报(自然科学版)，2003，15(2)：57～60