基于受控马尔科夫链的软件缺陷优化测试策略

包晓安[1] 姚 澜[1] 张晓文[1] 曹建文[2]

(浙江理工大学信息电子学院 杭州 310018)[1] (中国科学院软件研究所并行软件实验室 北京 100000)[2]

摘 要 目前许多文献都讨论的受控马尔科夫链软件测试模型,是通过对部分假设条件进行特殊化处理后得到的,这将导致模型的适用范围较小且偏离实际应用。依据软件控制论思想,通过一系列新的制约条件的转换,提出一种改善的、测试资源约束下的受控马尔科夫链模型来消除已有模型的缺陷。同时,该模型能够在高效性、复杂性和适用性3方面达到一个平衡点。为了证明其有效,根据该模型设计了一种新的软件缺陷优化测试策略,并对该策略进行了仿真实验,将其与传统的随机测试策略进行了比较。实验结果表明,该模型具有较高的实用性和有效性。

关键词 软件控制,软件缺陷测试,优化测试,受控马尔科夫链

中图法分类号 TP301 **文献标识码** A

Optimal Testing for Software Defects Based on Controlled Markov Chain

BAO Xiao-an[1] YAO Lan[1] ZHANG Xiao-wen[1] CAO Jian-wen[2]

(Department of Information & Electronics, Zhejiang Sci-Tech University, Hangzhou 310018, China)[1]

(Research and Development Center for Parallel Software, Institute of Software, Chinese Academy of Sciences, Beijing 100000, China)[2]

Abstract The Controlled Markov Chain(CMC) model for software testing discussed in most works at present is obtained from a series of assumptions, and partial of the assumptions have been specialized, which makes the scope of application of these models were comparatively small. And thus these models deviate from practical application. According to the software cybermetics, this article provided an improved CMC model with cost constraints by introducing a series of new transformation of limit condition. This model eliminates some of the defects of existing models. Meanwhile, the model can reach a balance in efficiency, complexity and applicability. In order to verify the effectiveness of the model, a new optimal testing strategy for software defects was designed according to the newly provided model. Through a simulation experiment, the strategy was compared with the traditional random testing strategy. The results show that our improved CMC model has high applicability and effectiveness.

Keywords Software cybernetics, Software defects test, Optimal testing strategy, Controlled Markov chain

1 引言

传统的软件测试方法[1,2],都是预先制定好测试策略,没有明确的优化目标,因此其效率和可靠性比较低。一系列制约条件下约束的受控马尔科夫链方法(CMC)可设计一个软件缺陷优化测试策略,以预先获得一个明确的优化目标。该方法是依据软件控制论思想[3]得到的,它将软件测试看作一个控制问题,被测软件作为受控对象,测试策略作为相应的控制器,二者形成一个闭环反馈系统,如图1所示。其原理是形式化软件测试中的反馈机制,并根据已知的测试目标设计并优化测试策略。本文不改变CMC软件测试方法的基本结构(见图1)[4]。

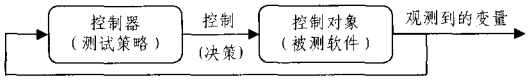


图1 软件测试看作控制问题

已有的关于CMC软件测试的研究所考察的问题大多是研究如何以最小的期望成本检测并移除所有的缺陷。但在实际的测试中,缺陷数是未知的,且发布版本的软件总是会存在缺陷[4],因此工程师或管理者可能对这样的问题感兴趣:给定一个成本约束,如何检测并移除尽可能多的缺陷。正是基于以上原因,本文通过一系列新的制约条件转换,提出了一种改善的、测试资源约束下的受控马尔科夫链模型。本文的目标是找到一个测试策略,使得移除缺陷所产生的总折扣最大。这样,该策略能够在成本约束内检测并移除尽可能多的缺陷,从而提高其有效性。

2 受控马尔科夫链软件测试模型

已有的受控马尔科夫链软件测试模型[4-6],由于考虑到计算复杂度等问题,对部分条件进行了特殊化处理(如被测软件包含的缺陷数一定、不同的软件缺陷被检测到的概率相等、检

到稿日期:2011-06-20 返修日期:2011-11-04 本文受浙江省自然科学基金项目(Y1100289,Y1100726,Y12F020198,Y12F020194)和浙江省科技厅公益技术研究项目(2011C3005)资助。

包晓安(1973-),男,硕士,副教授,主要研究方向为软件测试、计算机信息处理、模式识别等,E-mail:zlyaolan@yahoo.cn;姚 澜(1988-),女,硕士生,主要研究方向为软件测试;张晓文(1985-),男,硕士生,主要研究方向为计算机信息处理。

测到一个缺陷就立即移除等)，而影响模型的适用范围及效率。本文针对软件测试过程，对制约条件进行了一系列转换，使其适用性更加广泛。

对有关测试问题，令

$$z_t=\begin{cases}1, & t\text{时刻决策检测到一个缺陷}\\ 0, & t\text{时刻决策没有检测到缺陷}\end{cases} \tag{1}$$

则本文的软件测试模型基于以下假设：

1)测试开始时，被测软件包含 N(N 可以任意大)个缺陷，各个缺陷的可检测性不相等。在本次测试活动中检测到的缺陷数的期望值为 $n(n\leqslant N)$。

2)测试过程中总的可用测试资源为 x_0。

3)N 个缺陷都只能处于以下任一状态之一：已移除、未检测到、已检测到但未移除，用符号表示如下：

$$Y_t^{(k)}=\begin{cases}0, & t\text{时刻第}k\text{个缺陷已移除}\\ 1, & t\text{时刻第}k\text{个缺陷未被检测到}\\ 2, & t\text{时刻第}k\text{个缺陷已检测到但未移除}\end{cases} \tag{2}$$

式中，$k=1,2,\cdots,N$ 且 $Y_0^{(1)}=Y_0^{(2)}=\cdots=Y_0^{(n)}=\cdots=Y_0^{(N)}=1$。

4)软件状态用 $\xi_t=(Y_t^{(1)},Y_t^{(2)},\cdots Y_t^{(n)},Y_t^{(n+1)},\cdots Y_t^{(N)},X_t)$ 表示，其中 X_t 表示 t 时刻剩下的测试资源(包括 t 时刻选择和执行测试用例所需的资源)。

5)目标状态是吸收态 $\xi_{\text{fin}}=(0,0,\cdots,0)$。

6)总共有 $m+2$ 个可能的不同决策，即决策集 $A=\{1,2,\cdots m,m+1,m+2\}$，其中 $m+1$ 用以移除一定数量的缺陷，决策 $m+2$ 用于当所有测试资源用完时将被测软件状态置于吸收态 $\xi_{\text{fin}}=(0,0,\cdots,0)$。

7)任何时刻都有 m 个可选决策，每次决策最多检测出一个缺陷，不管该缺陷是否会引发故障，其代价均为 $W_{\xi_t}(A_t)$，即消耗单位 $W_{\xi_t}(A_t)$ 的资源；决策 $m+1$ 的代价为 $W(m+1)$[7]，决策 $m+2$ 的代价为 $W(m+2)$，则：

$$W_{\xi_t}(A_i=i)=\begin{cases}W_{\xi_t}(i)>0, & \text{如果 } x_t>0\\ \infty, & \text{其它}\end{cases} \tag{3}$$

$$W_{\xi_t}(m+1)=\begin{cases}C, & \text{若 } x_t>C\text{ 且 }d\text{ 个新缺陷被检测到}\\ & \text{或 } x_t>C\text{ 且 }M>x_t,e\neq 0\\ \infty, & \text{其它}\end{cases} \tag{4}$$

式中，$M=\min\{W_{\xi_t}(1),W_{\xi_t}(2),\cdots,W_{\xi_t}(m)\}$。

$$W_{\xi_t}(m+2)=\begin{cases}\infty, & \text{如果 } M\leqslant x_t\text{ 或 }M>x_t\text{ 且 }x_t\geqslant C,e\neq 0\\ x_t, & \text{其它}\end{cases} \tag{5}$$

8)Z_t 仅仅依赖于软件状态 ξ_t，其中一个测试决策检测到缺陷的概率由矩阵 Θ 决定：

$$\Theta=\begin{bmatrix}\rho_1^{(1)} & \rho_1^{(2)} & \cdots & \rho_1^{(n)}\\ \rho_2^{(1)} & \rho_2^{(2)} & \cdots & \rho_2^{(n)}\\ \vdots & \vdots & \ddots & \vdots\\ \rho_m^{(1)} & \rho_m^{(2)} & \cdots & \rho_m^{(n)}\end{bmatrix}$$

$$\begin{aligned}&P_r\{Z_t=1|A_i=i\}=\sum_{i=1}^{n}\rho_i^{(k)}\\ &P_r\{Z_t=0|A_i=i\}=1-\sum_{i=1}^{n}\rho_i^{(k)}\end{aligned} \tag{6}$$

式中，$k=1,2,\cdots,n$。

9)如果状态 ξ_t 下采取的决策引发了一个未被检测到的缺陷，则其代价减少 $\delta_{\xi_t}(A_t)>0$，称之为决策 A_t 产生的代价折扣。决策 $m+1$ 和 $m+2$ 不会产生折扣。

10)当有 d 个新缺陷被检测到，且 $C<x_t$，则 d 个相应的缺陷将立即从被测软件移除，且不引发新缺陷。当检测到 $e(e<d)$ 个新缺陷时，测试资源不够了，则立即移除这 e 个缺陷，且执行决策 $m+1$ 时，其代价与移除 d 个缺陷是相同的。

对以上定义的一个受控马尔科夫链进行如下解释：

在实际的测试中，测试人员并不知道缺陷总数 N，软件发布版本总是会存在缺陷，不可能在一次测试活动中检测到所有缺陷，需要定期对软件进行检测，每次检测到的缺陷数是不定且有限的，此处设为 n。其余 $N-n$ 个缺陷在此次测试活动中检测出来的概率极小，视为小概率事件，则 t 时刻的状态可表示为 $\xi_t=(Y_t^{(1)},Y_t^{(2)},\cdots Y_t^{(n)},X_t)$。

如果剩下的总资源 x_t 小于任何一个决策所需的代价，即 $M>x_t$(x_t 不一定等于 0)，则被测软件转移到吸收态，测试过程也相应结束。如果剩下的总资源大于某个决策所需的代价，即使已经检测到 n 个缺陷，测试过过程仍将继续。本文将剩余的测试资源作为测试过程停止与否的唯一标准。

式(5)表示当 $M\leqslant x_t$ 时，不可执行 $m+2$，而应执行决策 $1,2,\cdots,m$ 或 $m+1$。但由于当 d 个新缺陷被检测到时，只要满足条件，就会立即被移除(即在 $t-1$ 时刻就会被移除)，因此此时只可能执行决策 $1,2,\cdots,m$；式(4)表示当 $M>x_t$，检测到 e 个缺陷时，执行决策 $m+1$，且代价为常数 C；式(3)表示当测试资源为 0 时，不可执行决策 $1,2,\cdots,m$，而应执行决策 $m+2$。事实上是测试资源决定了各个状态下决策的选择。

其中软件的状态转移与 t 时刻采取的决策有关，且有如下状态转移函数：

$$\begin{aligned}&q_{\xi_t\xi_{t+1}}(i)=P_r\{\xi_{t+1}|\xi_t,A_t=i\}\\ &=\begin{cases}\sum_{k=1}^{n}\rho_i^{(k)}, & Z_t=1,x_{t+1}\geqslant 0\\ 1-\sum_{k=1}^{n}\rho_i^{(k)}, & Z_t=0,x_{t+1}\geqslant 0\\ 0, & \text{其它}\end{cases}\end{aligned} \tag{7}$$

式中，$x_{t+1}=x_t-W_{\xi_t}(i)+l\delta_{\xi_t}(i)$，$l=\begin{cases}1, & Z_t=1\\ 0, & Z_t=0\end{cases}$，且 $i=1,2,\cdots,m$。

$$\begin{aligned}&q_{\xi_t\xi_{t+1}}(m+1)=P_r\{\xi_{t+1}|\xi_t,A_t=m+1\}\\ &=\begin{cases}1, & \text{如果 } c<x_t\text{ 且 }d\text{ 个缺陷被检测到}\\ & \text{或 } e\neq 0\text{ 且 }M>x_t\\ 0, & \text{其它}\end{cases}\end{aligned} \tag{8}$$

$$\begin{aligned}&q_{\xi_t\xi_{t+1}}(m+2)=P_r\{\xi_{t+1},m+2\}\\ &=\begin{cases}0, & \text{若 } M\leqslant x_t\text{ 或 }C\leqslant x_t\text{ 且 }M>x_t\\ 1, & \text{其它}\end{cases}\end{aligned} \tag{9}$$

令 τ 为软件转移到吸收态的首达时间，则有以下等式成立：

$$x_0=\sum_{t=0}^{\tau}[W_{\xi_t}(A_t)-\rho_{A_t}^{*}\delta_{\xi_t}(A_t)]+x_\tau+(\frac{n}{d}+1)C \tag{10}$$

令

$$\rho_{A_t}^{*}(A_t=i)$$

$$=\begin{cases}\sum_{k=1}^{n}\rho_i^{(k)}\operatorname{sign}(Y_t^{(k)}-2)\operatorname{sign}(-Y_t^{(k)}), & i=1,2,\cdots,m\\ 0, & i=m+1,m+2\end{cases} \tag{11}$$

式中，$\operatorname{sign}(x)$为符号函数，$x>0$ 时，其值为 1；$x=0$ 时，其值为 0；否则，为-1。则有

$$J_\omega(x_0)=\sum_{t=0}^{\tau}\rho_{A_t}^{t}\delta_{\xi_t}(A_t) \tag{12}$$

式中，ω 表示测试策略，$J_\omega(x_0)$ 表示测试过程停止时 ω 产生的代价总折扣。由于式(10)最后一项中 n 和 C 都是常数，因此其值只与 d 有关。当 d 一定时，该项为常数，消耗的测试资源一定，即不影响测试决策选取。$x(\tau)$ 虽然不是一个定值，但它是测试策略(即是决策 $1,2,\cdots,m$ 的排列顺序)确定之后剩余的测试资源，因此可以认为不影响测试决策的形成。

3 软件缺陷优化测试策略

根据受控马尔科夫链理论[8]可知，存在一个确定性静态控制策略使 $J_\omega(x_0)$ 最大。“确定性”是指该策略每次都选取一个独一无二的最优决策。“静态”是指该最优策略是一个关于 ξ_t 的函数，且依赖于历史测试状态。为了决定该最优测试策略，本文使用如下逐次值近似值法：

$$\text{令 } v_{s+1}(\xi)=\min_{1\leqslant i\leqslant m}\{W_\xi^*(i)+\sum_{\eta\neq\xi_{fin}}q_{\xi\eta}(i)v_s(\eta)\} \tag{13}$$

式中，$W_\xi^*(i)=-\sum_{\eta}q_{\xi\eta}(i)\delta_s(i)$。

$$\text{令 } v(\xi)=\lim_{s\to\infty}v_s(\xi) \tag{14}$$

式中，$v(\xi)$ 是状态 ξ 的价值函数，最优决策正是在此基础上进行选择的。由此可得如下定理。

定理 1 根据条件(1)—(10)，有：

$$v(\xi_t)=\begin{cases}\min\limits_{1\leqslant i\leqslant m}\{W_{\xi_t}^*(i)+\rho_i^*v(\xi_{t+1}^{(k)})+(1-\rho_i^*)v(\xi_{t+1}^{(0)})\}, & \text{若 } M\leqslant x_t\\ 0, & \text{其它}\end{cases} \tag{15}$$

式中，$\xi_{t+1}^{(k)}$ 表示决策 A_t 检测到一个缺陷时状态 ξ_t 的下一状态，$\xi_{t+1}^{(0)}$ 表示决策 A_t 没有检测到缺陷时状态 ξ_t 的下一状态。上述定理明确定义了如何在成本约束下对软件进行测试。而决策 $m+1$ 和 $m+2$ 对决策的优化选择没有影响，因此优化策略只需对决策 $1,2,\cdots,m$ 的选取或排列进行优化。针对测试中能考虑被检测出的缺陷数的期望值 n，对 n 个缺陷的检测和移除过程进行优化，当已有 n 个缺陷检测并被移除且 $M\leqslant x_t$ 时，则可得到如下测试策略：

$$A_{\xi_t}^*=\begin{cases}\forall i 1\leqslant i\leqslant m,\xi_t=(0,\cdots,0,Y_t^{(n+1)},\cdots,Y_t^{(N)},x_t)\text{且 }x\neq 0\\ \arg\min\limits_{1\leqslant i\leqslant m}\{W_{\xi_t}^*(i)+\rho_i^*v(\xi_{t+1}^{(k)})+(1-\rho_i^*)v(\xi_{t+1}^{(0)})\},\text{其它}\end{cases} \tag{16}$$

本策略即为本文基于对受控马尔科夫链的一系列制约条件的转换而得到的优化测试策略，它使总折扣 $J_\omega(x_0)$ 最大，能够在成本约束(x_0 单位的测试资源)内检测并移除最多的缺陷。

4 仿真实验

为了证明本文所设计的优化软件测试策略的有效性，通过仿真实验将其与随机软件测试策略[9,10]进行比较。在随机测试策略中，运用均匀概率分布从决策集中随机地选择决策，即各个可选决策被选取的概率相等；在优化测试策略中，假设软件的相关参数的真值已知。

例 1 假设 $n=6,d=2,x_0=80,\delta_{\xi_t}(2)=3.5,\delta_{\xi_t}(1)=6,A=\{1,2,3,4\},W_{\xi_t}(1)=8,W_{\xi_t}(2)=5.5,\Theta=\begin{bmatrix}0.020&0.005&0.015&0.008&0.025&0.012\\0.003&0.024&0.002&0.010&0.006&0.015\end{bmatrix}$，分别对优化测试策略和随机测试策略进行了 10 次仿真实验，表 1 为相应的仿真结果。

表 1 对例 1 分别采用随机测试策略与优化测试策略的仿真结果

i	1	2	3	4	5	6	7	8	9	10	Avg
Rc(i)	7	10	13	0	4	9	5	9	8	6	7.1
Oc(i)	8	12	7	2	5	6	6	10	9	7	7.5
Rd(i)	2	4	3	1	2	5	3	2	4	3	3.1
Od(i)	4	6	5	3	3	5	4	4	5	5	4.4
Rr(i)	21.5	38	28.5	19	21.5	47.5	28.5	21.5	38	28.5	29.25
Or(i)	38	76	47.5	28.5	28.5	47.5	38	38	47.5	47.5	43.7

其中，$Rc(i)$、$Oc(i)$ 分别为第 i 次仿真随机测试和优化测试所用的用例数；$Rd(i)$、$Od(i)$ 分别为第 i 次仿真随机测试和优化测试检测并移除的缺陷数；$Rr(i)$、$Or(i)$ 分别为第 i 次仿真随机测试和优化测试产生的总折扣。

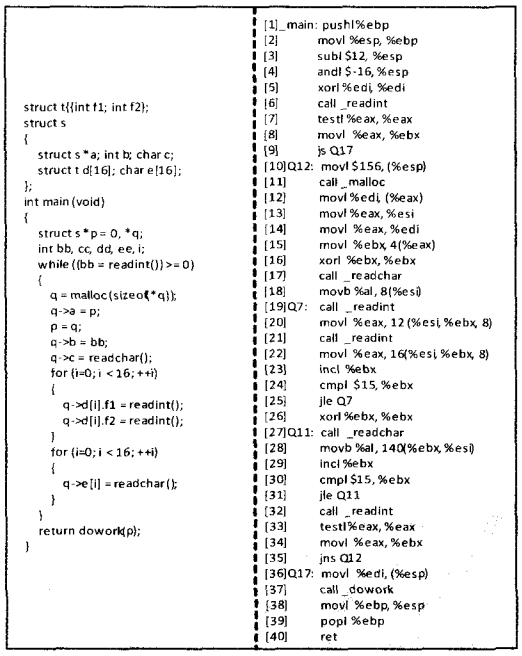
通过对表中数据进行分析和计算可知，当测试资源用完时，最优测试策略大约比随机测试策略多产生 49% 的折扣，同时大约多检测到 42% 的缺陷，而最优测试策略所用的测试用例数比随机测试策略大约只多 6%。

5 相关工作

本文与基于软件控制论的自适应软件测试研究紧密相关：Cai 首次提出了软件控制论思想，文献[3]对该思想进行了详细的描述；文献[11]对基于软件控制论的软件测试的可行性和有效性进行了验证，并提出用受控马尔科夫链方法来进行软件测试；文献[4]提出了一种基于软件控制论的优化测试和自适应测试方法，减少了传统随机测试移除一定量缺陷所需的测试用例数；文献[5]将自适应测试方法扩展到软件构件的测试；文献[13]在受控马尔科夫链方法的框架内讨论了具有测试用例限制的构件软件的自适应测试。然而，以上研究都是基于简化的受控马尔科夫链软件测试模型，对部分假设条件进行特殊化处理，模型的适用范围较小。文献[14]介绍了一种新的自适应软件测试过程的稳定性模型，该模型能够根据现场实际测量的错误数据客观地反映软件测试过程稳定性的变化情况；文献[6]提出了一个测试资源约束下的模型，对已有的关于优化测试和自适应测试的研究进行推广；文献[12]通过引入不同缺陷检测到的概率不相等、批量调试等假设，提出了一种改进的受控马尔科夫链软件测试模型，旨在克服已有模型的缺陷。但这些模型假设被测软件包含的缺陷数是一定的，并限定了测试次数，当设定的缺陷数或测试次数用完时，测试过程就会停止，从而使得测试效率较低。

结束语 本文针对受控马尔科夫链软件测试模型，对制约条件进行了转换。其中包括，①由于软件缺陷数是个未知数，本文对每次测试过程中可检测到的缺陷数计算出一个期望值；②用测试资源作为测试过程是否停止的标准，以提高其效率；③修改了移除缺陷和将系统置于目标状态不会消耗资源的假设，并引入故障检测率矩阵 Θ，用以表示各个缺陷具有不同的可检测性，同时对检测到的缺陷进行批量调试。通过这一系列条件转换，本文得到了一种新的软件测试模型，使得模型在高效性、复杂性和适用性 3 方面达到了一个平衡点。根据本模型设计了一种新的优化软件缺陷测试策略，策略使得移除缺陷产生的总折扣最大，因此能够在成本约束内检测并移除最多的缺陷。仿真实验数据表明，测试资源用完时，最优测试策略所用的测试用例数只略微多于随机测试策略，优

(下转第 136 页)



```
struct t{int f1; int f2};
struct s
{
   struct s *a; int b; char c;
   struct t d[16]; char e[16];
};
int main(void)
{
   struct s *p = 0, *q;
   int bb, cc, dd, ee, i;
   while((bb = readint()) >= 0)
   {
      q = malloc(sizeof(*q));
      q->a = p;
      p = q;
      q->b = bb;
      q->c = readchar();
      for(i=0; i < 16; ++i)
      {
         q->d[i].f1 = readint();
         q->d[i].f2 = readint();
      }
      for(i=0; i < 16; ++i)
      {
         q->e[i] = readchar();
      }
   }
   return dowork(p);
}
```

```
[1]_main: pushl %ebp
[2]       movl %esp, %ebp
[3]       subl $12, %esp
[4]       andl $-16, %esp
[5]       xorl %edi, %edi
[6]       call _readint
[7]       testl %eax, %eax
[8]       movl %eax, %ebx
[9]       js Q17
[10]Q12:  movl $156, (%esp)
[11]      call _malloc
[12]      movl %edi, (%eax)
[13]      movl %eax, %esi
[14]      movl %eax, %edi
[15]      movl %ebx, 4(%eax)
[16]      xorl %ebx, %ebx
[17]      call _readchar
[18]      movb %al, 8(%esi)
[19]Q7:   call _readint
[20]      movl %eax, 12(%esi,%ebx, 8)
[21]      call _readint
[22]      movl %eax, 16(%esi,%ebx, 8)
[23]      incl %ebx
[24]      cmpl $15, %ebx
[25]      jle Q7
[26]      xorl %ebx, %ebx
[27]Q11:  call _readchar
[28]      movb %al, 140(%ebx,%esi)
[29]      incl %ebx
[30]      cmpl $15, %ebx
[31]      jle Q11
[32]      call _readint
[33]      testl %eax, %eax
[34]      movl %eax, %ebx
[35]      jns Q12
[36]Q17:  movl %edi, (%esp)
[37]      call _dowork
[38]      movl %ebp, %esp
[39]      popl %ebp
[40]      ret
```

(a) 源程序　　(b) 汇编程序

图 4

```
struct s2
{
   t1 f1;  /at offset 0 */
   t2 f2;  /at offset 4 */
   t3 f3;  /at offset 8 */
   struct s1  /* sizeof(struct s1) == 8 */
   {
      t4 f1;  /at offset 0 */
      t5 f2;  /at offset 4 */
   } f4[];  /at offset 12 */
   t6 f5[];  /at offset 140 */
};
```

图 5　复杂类型重构的中间结果

结束语　本文给出了汇编算法基础上数据类型自动重构的算法，该算法能够对简单及复杂数据类型进行准确的恢复。由于算法是基于简单类型的有穷格及有限内存访问标记集合上的操作，因此其能够在有限循环内结束。该算法是目前正在开发的类型重构工具的关键技术，下一步将尝试采用动态的分析方法对二进制代码进行分析，形成动、静结合的分析方法，争取在复杂数据类型重构的准确度上有进一步的提高。

参 考 文 献

[1] 陈凯明，刘宗田. 反编译研究现状及进展 [J]. 计算机科学，2001，28(5)：113-115

[2] 肖海，陈平. 基于运行时类型分析的整形漏洞二进制检测和定位系统 [J]. 计算机科学，2011，38(1)：140-143

[3] Lin Z, Zhang X, Xu D. Automatic reverse engineering of data structures from binary execution[C]// Proceedings of the Network and Distributed System Security Symposium. 2010

[4] Cifuentes C. Reverse Compilation Techniques, Queensland University of Technology[D]. Department of Computer Science, July 1994

[5] Mycroft A. Type-Based Decompilation[C]// Proceedings of the 8th European Symposium on Programming Languages and Systems. March 1999：208-223

[6] Emmerik M V. Static Single Assignment for Decompilation[D]. Queensland：The University of Queensland，2007

(上接第 119 页)

化测试策略产生的总折扣和检测到的缺陷数均明显多于随机测试策略，因此本文的优化测试策略优于随机测试策略。拟在以后的研究中进一步扩展制约条件，以提高其效率和适用性。

参 考 文 献

[1] 陈明. 软件测试技术[M]. 北京：清华大学出版社，2011：11-61

[2] Dalley J L. The art of software testing[C]// Dayton. Proceedings of the IEEE 1991 National Aerospace and Electronics Conference. Dayton，1991：757-760

[3] Cai K Y, Cangussu J W, DeCarlo R A, et al. An overview of software cybernetics[C]// Proc. the 11th International Workshop on Software Technology and Engineering Practice. Chicago：IEEE Computer Society Press，2004：77-86

[4] Cai K Y. Optimal software testing and adaptive software testing in the context of software cybernetics[J]. Information and Software Technology，2002，44(02)：841-855

[5] Chen T Y, Li Y C, Ning W Y, et al. Adaptive testing of software components[J]. ACM Symposium on Applied Computing，2005，17(5)：1463-1469

[6] Wong W E, Hu Hai. Optimal and adaptive testing with cost constraints[C]// Proceedings of the 2006 International Workshop on Automation of Software Test. Shanghai：ACM，2006：71-77

[7] Okamura H, Furumura H, Dohi T. On the effect of fault removal in software testing Bayesian reliability estimation approach[C]// Washington. Proceedings of the 17th International Symposium on Software Reliability Engineering. Washington：IEEE Computer Society，2006：247-255

[8] Derman C. Finite State Markovian Decision Processes[M]. New York：Academic Press，1970：234-305

[9] Loo P S, Tsai W K. Random testing revisited [J]. Information and Software Technology，1988，30(7)：402-417

[10] Chen T Y, Kuo F C, Liu Huai, et al. Does Adaptive Random Testing Deliver a Higher Confidence than Random Testing [C]// IEEE the 18th International Conference on Quality Software. Chicago：IEEE Computer Society，2008：145-154

[11] Cai K Y. A controlled Markov chains approach to software testing [J]. IEEE Transactions on Software Engineering，2002，21(6)：312-320

[12] Hu Hai, Jiang Chang-hai. Adaptive Software Testing in the Context of an Improved Controlled Markov[C]// Annual IEEE International Computer Software and Applications Conference. Chicago：IEEE Computer Society，2008：853-858

[13] 殷脂，曹渠江. 构件软件的自适应测试[J]. 计算机应用，2005，25：417-420

[14] 江韩斌，崔小乐，周豪. 自适应软件测试过程的稳定性模型[J]. 微电子学与计算机，2008，25，(8)：98-102