基于 UML 活动图的测试研究进展*)

高红梅 许 东 刘宗田
(上海大学计算机学院 上海 200072)

摘 要 UML 活动图不再是状态图的特例,它作为一种独立的模型广泛用于软件的行为建模。基于 UML 活动图的测试受到业界的普遍欢迎。然而从 UML 活动图自动生成完整的测试场景\用例成为一个难点。本文对基于 UML 活动图的测试进行了比较分析,总结了几种从 UML 活动图生成测试场景\用例的方法及其使用的算法,即反蚂蚁 Agent 方法、灰盒方法、自适应细菌 Agent 方法和系统的形式化方法。对这些方法进行了分析与比较,指出一些不足之处。最后对 UML 活动图测试的发展趋势做了一些展望。

关键词 UML 活动图,细-线程,测试场景,反蚁群 Agent,自适应细菌 Agent

Progress of Research on UML Activity Diagrams-based Testing

GAO Hong-Mei XU Dong LIU Zong-Tian
(School of Computer Engineering and Science, ShangHai University, Shanghai 200072)

Abstract UML activity diagram is no longer a special case of UML state diagram and it has been widely used in modelling software behaviours in the form of independent model. The testing based on UML activity diagram is universally welcomed by industry. However, it is difficult to automatically generate test scenarios\cases form UML activity diagrams. Some researches on the testing based on UML activity diagram are analysed in the paper, i. e., anti-ant-like agent, grey-box, adaptive bacteria agent and systematic formal methods. A few approaches to generate test scenarios\cases from UML activity diagrams are summarised and the resulting comments are depicted. Finally, the further research trend to UML activity diagram based testing is addressed.

Keywords UML activity diagram, Thin-thread, Test scenarios, Anti-ant-like agents, Adaptive bacteria agents

1 简介

UML 作为一种半形式化的建模语言,广泛用于描写需求分析和设计规格说明书,事实上它已经成为了面向对象的软件系统建模的工业标准。目前基于 UML 模型的场景测试受到越来越多的关注。场景代表了一个软件系统中的操作执行序列。基于场景的测试技术需要解决两个重要的问题,即测试场景的生成和测试场景的优化。UML 活动图可以描述系统为完成指定功能或任务所必须执行的活动序列,主要用于软件系统建模。从活动图产生测试场景,可以实现黑盒测试或系统功能测试,也可以用来验证实现与需求是否一致,较早发现缺陷。然而,由于 UML 活动图缺乏精确的语义描述,从中自动产生完整的测试场景\用例成为一个难题。

本文介绍四种从活动图中产生测试场景的方法。第 2 节对 UML 活动图做了一个概述;第 3 节介绍这四种产生测试场景的方法的实现思想;第 4 节对这四种方法进行比较分析;最后对基于 UML 活动图进行测试的发展趋势做了一些展望。

2 UML 活动图的综述

UML 活动图由结点和边组成。结点分为动作状态、活动状态、判断、泳道、分叉、汇合、对象、信号的发送和接受等类型。边代表数据流程或控制流程,表示活动、包含活动的对象的发生序列,分为控制流、信息流和信号流等类型。一个活动图中有一个或多个初始状态,可能有多个终止状态或流结束状态。判断(decision)结点根据监视条件产生分支结构。分叉结点表示有多少个处理是同步执行的,然后在一个汇合结点结束同步执行。参数的输入和输出也可以显示在活动图中,用附于活动图的矩形边框上的小矩形表示。还允许用户在活动图上建立横向、纵向和横纵向多种分区,用来描述活动图上与动作相关的主体,用泳道(swimlanes)表示。从一个开始结点到一个结束结点代表了整个活动图的完成。

3 从 UML 活动图中生成测试场景

这部分介绍四种从 UML 活动图中生成测试场景的方法及其使用的算法。

3.1 反蚂蚁 Agent(anti-ant-like agents)方法

近来许多人工智能技术应用于软件工程学,特别是应用在软件测试的研究中。其中主要括遗传算法和蚁群最优化算法的应用[1]等。

3.1.1 蚁群算法的基本思想

蚁群最优化算法的思想来源于自然界中蚁群寻找食物的过程,是对蚂蚁寻找食物行为的模拟。蚂蚁在觅食过程中,在它所经过的路径上留下浓度与食物源质量成比例的信息素(pheromone),并能够感知信息素的存在及其浓度,以此指导自己的运动方向,倾向于朝着信息素浓度高的方向移动。于

*)国家发展与改革委员会基金项目(编号 SNMCFIP-2006S001)。**高红梅** 硕士生,研究方向:软件测试;许 东 博士生,讲师,研究方向:软件工程;**刘宗田** 教授,研究方向:软件工程。

是，蚁群的集体行为便表现出信息正反馈现象：某一路径上走过的蚂蚁越多，则后来者选择该路径的概率就越大，因此质量好、距离近的食物源会吸引越来越多的蚂蚁，信息素浓度的增长速度会更快。蚂蚁个体之间就是通过这种信息的交流达到寻找食物和蚁穴之间最短路径的目的。

使用蚂蚁 Agent 遍历 UML 活动图，为自动生成测试场景提供一个可能的解决方案。然而原始的蚁群最优化算法却不能直接应用于活动图生成测试场景。受蚁群算法的启发，提出一种反蚁群 Agent 方法从 UML 活动图生成测试场景的方法。

3.1.2 *反蚁群算法*

反蚁群算法是通过一组蚂蚁在 UML 活动图中协作搜索，生成测试场景。对活动图的两种特殊结点做了一些规定：

· 活动图的结束结点被认为是有毒的食物，人工蚂蚁找到它就会死去。

· 活动图的分叉-汇合结点被认为是河的两岸，在它们之间的每条路径被看作浮桥。因此，人工蚂蚁要想过“河”，必须先在“河”上建立一座浮桥。

建立分叉-汇合浮桥是为了确保在分叉-汇合之间结点的所有可能组合的每条路径都能被蚂蚁搜索过。但是，它却不能保证在分叉-汇合之间的每条活动边至少被一只蚂蚁搜索过。因此，将这些活动边的信息素的递增值设为 $1/n!$（n 为分叉-汇合之间的活动边的数目）。如果还有未被访问的活动边，其它的蚂蚁就会进一步搜索，直到所有的活动边都被访问完为止。

与蚁群最优化算法相似，图中的蚂蚁能够感知从当前结点出发的边的信息素，根据边的信息素来决定下次移动的方向，而且在结点间移动时留下信息素。但是，与蚁群最优化算法和真实的蚂蚁不同，该方法中的人工蚂蚁展现了相反的行为，喜欢在没有被探索或很少被探索的边上留下信息素。

为了避免人工蚂蚁进入活动图的循环部分永无休止地搜索，或者是蚂蚁在活动图的某个中间位置上停止下来，对人工蚂蚁作出这样的规定：

· 蚂蚁是拥有有限的能量的，当且仅当蚂蚁耗尽能量，它才会停止搜索。

· 蚂蚁的数目 m 是可以增加的，以可以进行穷举的搜索。

· 蚂蚁在搜索过程中选择下一结点的策略是：选择与当前结点之间连接的边所含信息素最少的结点。

· 如果当前活动边不是建立在分叉-汇合之间的浮桥，则其上面的信息素的递增值为 1。如果当前活动边是浮桥，则其上面的信息素的递增值为 $1/n!$。

蚂蚁的行为见图 1 所示。

与 UML 中的用例类似，细-线程可以描述系统场景。细-线程比用例含有更多的信息。细-线程可以组成树型结构，可用来进行系统关联分析、风险分析、可跟踪性分析、覆盖分析、完整和一致性检查以及测试场景和测试用例的生成。代表某些共同特征的细-线程可以分成一组，称为细-线程组。与每个细-线程或细-线程组相关的条件可以看成它们的激活条件。如果附于细-线程上的条件满足，细-线程才被激活了。这些条件也可以组成一棵条件树。此外，UML 活动图还包含数据对象，可以作为活动的输入/输出，可以被更新。类似地，数据对象也可以组成一棵数据对象树。

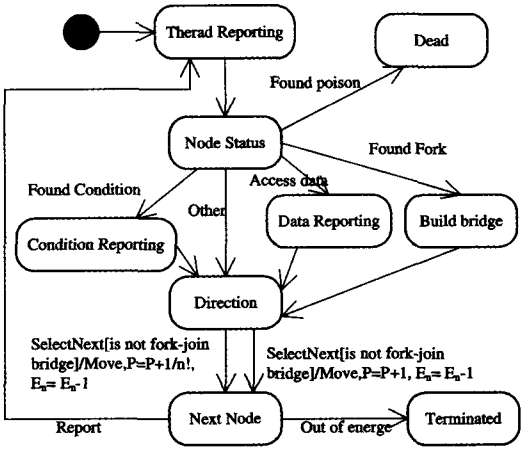


图 1 蚂蚁的行为状态机图

分别用具有层次结构的三种树及其之间的关系表示测试场景：(1)细-线程树；(2)监视条件树；(3)数据对象树。该方法利用蚂蚁 Agent 搜索活动图建立以上三棵树。在搜索的过程中，蚂蚁不断向细-线程树、监视条件树、数据对象树报告，用三种树记录它的行踪以及遇到的条件和数据对象。整个过程如图 2 所示。

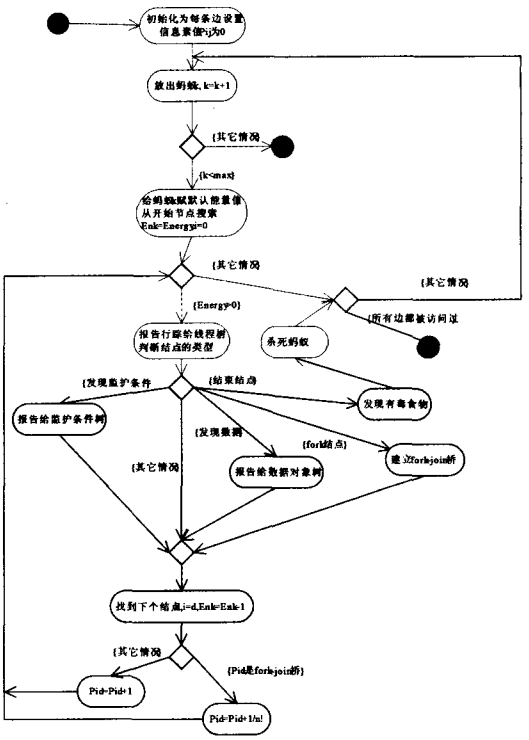


图 2 反蚁群算法流程图

反蚂蚁 Agent 方法直接使用了活动图模型，因此不需要额外地对图进行一些转化，简化了生成测试场景的过程。整个测试场景的生成过程也是自动完成的。

3.2 自适应 Agent(adaptive agent)方法

文[3]提出一种基于细菌行为的自适应 Agent 来搜索 UML 活动图，生成测试场景。

3.2.1 *自适应细菌 Agent 的基本思想*

自适应 Agent 模拟了一些细菌的行为，例如一种称为 Maxococcus xanthus 的细菌种类。Maxococcus xanthus 为了在泥沙中生存，在泥沙里四处移动，以寻找营养。随着移动，

此细菌能够留下一条粘液轨迹，使得泥沙润滑，更易于前行。因此，如果 Maxococcus xanthus 能够找到一条留有粘液的轨迹，它就能够沿着这条轨迹更容易地移动。与 Maxococcus xanthus 细菌类似，自适应 Agent 为了寻找营养就会在活动图中移动，随着它们的移动，它们就会留下粘液轨迹来标记它们的路径。但是与该细菌不同，自适应 Agent 实际上展现了与前述细菌习性相反的行为：

• 粘液浓度太高了，它就会变为不能移动的种胚，从而不能再移到更远的地方去寻找营养。因此，它被从寻找过程中移走。

• 与许多细菌一样，自适应 Agent 能够适应环境，当某个位置营养物非常充足时，到达那里的 Agent 的工作负荷就比较大，此 Agent 可以克隆自己来减轻它的工作负荷。由它克隆出来的那些 Agent 随后被分散到不同的方向来吸收营养物。

3.2.2 *自适应细菌 Agent 在活动图中的行为特性*

自适应细菌 Agent 在活动图中搜索形成测试场景，具有以下行为特性：

• Agent 只能以固定的速度沿边的方向移动。

• Agent 能够在它到达的活动边上释放固定数量的粘液。

• 当 Agent 到达含有高浓度粘液的活动边时就会变为种胚。

• 细菌 Agent 如果遇到抗生素就会被杀死。

• 在一般情况下，Agent 搜索一条活动边代表一个标准的工作负荷。一个 Agent 一次只能完成一个工作负荷。

• 在需要执行更多的工作负荷时，Agent 可以克隆自己，克隆出来的那些 Agent 继承了父 Agent 的所有信息，与父 Agent 有着相同的行为。

• 允许 Agent 循环查找，但是循环只限制被执行两次。

用这种 Agent 搜索活动图时，对活动图的两种特殊结点，做了以下假定：

• 活动图中的结束结点被认为是含有抗生素的地方，Agent 到达结束结点就会被杀死。

• 活动图中的分支结点被认为是含有丰富营养物的地方。如果一个分支结点有 m 条输出边，那么到达此分支结点的 Agent 要求的工作负荷就是标准工作负荷的 m 倍。因此，Agent 必须在分支结点克隆 $m-1$ 个 Agent(s)，用于搜索不同的分支边。

3.2.3 *自适应细菌 Agent 的搜索算法*

自适应细菌 Agent 的搜索算法的基本思想如下：

首先对于分叉-汇合对做了简单的分类：①原子分叉-汇合(AFJ)，如图 3 所示；②简单分叉-汇合(SFJ)，如图 4 所示；③简单嵌套分叉-汇合(SNFJ)，如图 5 所示；④分支嵌套分叉-汇合(BNFJ)，如图 6 所示。然后设计了一个从 SFJ 产生测试场景的算法，用于从 SFJ 中自动产生测试场景。给出了从 SFJ 中产生完全的测试场景数目计算公式：

$$NTS=\prod_{i=1}^{n-1}\sum_{j=0}^{m_{i+1}-1}\left(C_{\sum_{k=1}^{i}m_k+1}^{j+1}\times C_{m_{i+1}-1}^{j}\right)$$

封装一个从 SFJ 中生成测试场景的 Agent。

Agent 遇到分叉-汇合对时，如果是 SFJ，则调用 SFJ 算法；如果是 SNFJ 或 BNFJ，则将 SNFJ 或 BNFJ 分解为 SFJ，为每个分解的 SFJ 调用 SFJ 算法，然后再合并生成的测试场景。如图 5(a)所示的 SNFJ 分解成图 5(b)和图 5(c)所示的 SFJ。图 6(a)所示的 BNFJ 分解成图 6(b)和图 6(c)所示的 SFJ。

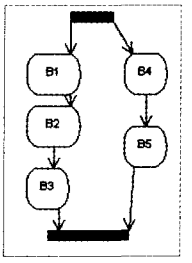


图 3 AFJ

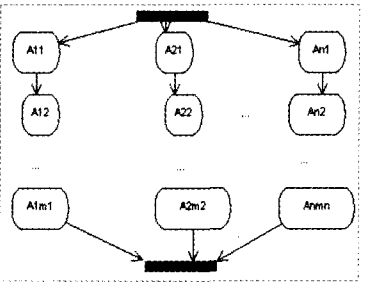
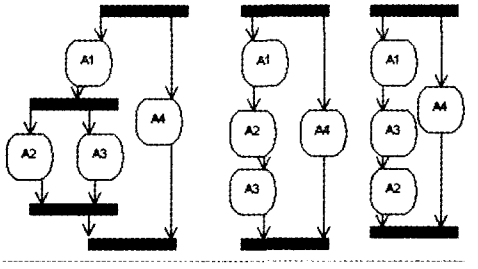


图 4 SFJ

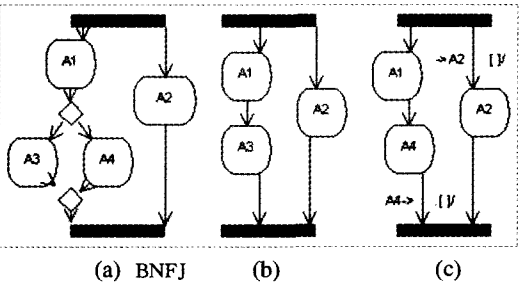
使用树作为从活动图中生成测试场景过程的中间存储结构。一个测试场景就是场景树从根到叶子的动作和迁移/监视条件组成的序列。算法流程图如图 7 所示。

自适应 Agent 方法对于从活动图中生成测试场景的过程是自动完成的，而且对于它所进行分类的分叉—汇合对，可以从中生成所有的测试场景。



(a) BNFJ (b) (c)

图 5



(a) BNFJ (b) (c)

图 6

3.3 使用灰盒方法从 UML 活动图中生成测试用例的方法

灰盒方法吸取了白盒方法和黑盒方法的特点，从活动图生成测试用例[2,6]。在生成测试用例之前，首先要遍历活动

图产生测试场景。

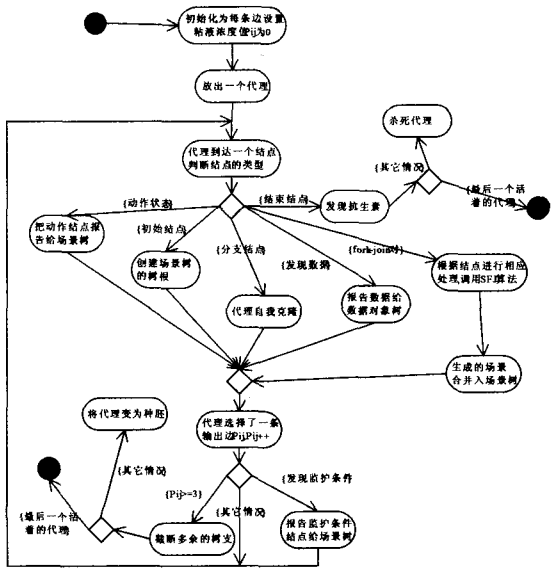


图 7　自适应 Agent 方法流程图

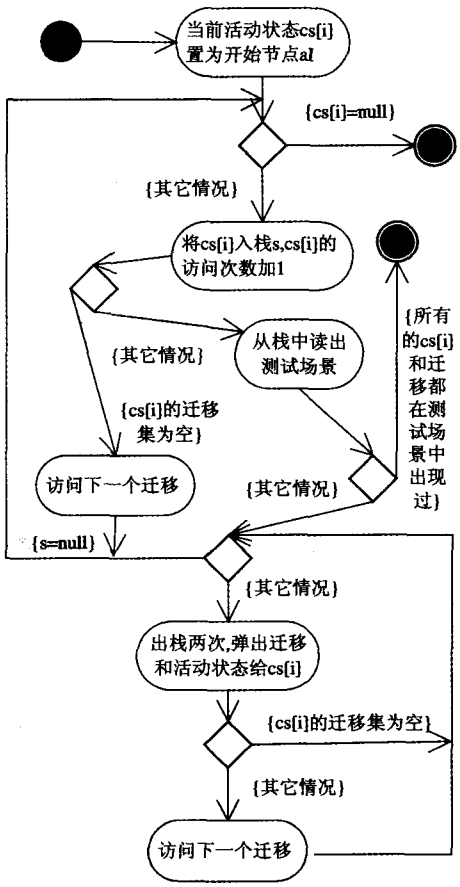


图 8　灰盒测试生成测试场景流程图

1. 灰盒测试的基本思想

文[2,6]在产生测试场景之前，对 UML 的可测性做了一些假定。假设活动图中的的分支结点和合并结点是成对出现，而且分叉和汇合结点也是成对出现的，分叉结点只能有两条输出边，提出了一个基本路径覆盖准则作为测试完成准则，即活动图上的所有路径至少要执行过一次。对于循环只执行一次，这样生成的路径称作基本路径。从初始活动开始，按照深度优先 DFS(Depth First Search method)开始遍历活动图，环路最多执行一次，所有动作状态和迁移都被覆盖，得到所有基本路径，即测试场景。

2. 灰盒测试的算法

这种方法的实现过程如下：

1)使用栈作为测试场景的中间存储结构。如果当前栈中状态为空时，则首先访问这个活动，然后从栈底到栈顶读出栈中数据作为一个测试场景，符合约束条件的返回测试场景，不符合的置空。如果测试场景没有覆盖所有的活动和迁移，回溯到上一个活动，继续遍历可以触发的活动。

2)为了得到所有测试场景，从初始状态开始按照 DFS 算法遍历活动图。即当前活动不为空，活动进栈，活动访问计数器加 1。从当前状态触发的迁移集中选择此刻只能被从当前状态触发的迁移，将此迁移和约束条件入栈。新的状态就是从当前状态中去除此迁移的前态，加上此迁移的后态，然后继续遍历。

3)循环只能被执行一次。如果遇到循环，活动访问次数达到 2 次，递归进入下个可以触发的迁移。

算法流程如图 8 所示。使用灰盒方法从 UML 活动图生成测试场景，再根据测试场景产生测试用例。

3.4　一种自动生成测试场景的形式化方法

UML 活动图可以说是一个广义的有向图。活动图中的数据流和控制流都有循环和回溯流向的可能，如果用深度优先或广度优先来遍历活动图生成测试场景\用例，其适用范围就很小了。前面所述的自适应 Agent 方法，不能处理更为复杂的分叉-汇合结构，如后面介绍的混合分叉-汇合结构。所以，从一般的 UML 活动图中自动生成测试场景存在一定的困难。

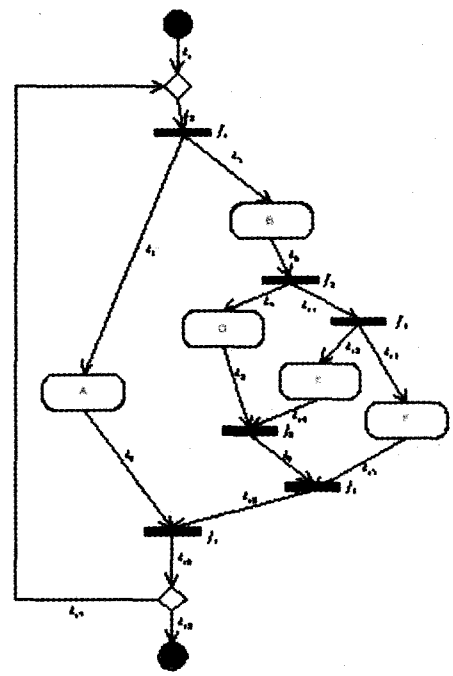


图 9　含有混合分叉-汇合的活动图

文[7]提出一种系统的方法，通过形式化的定义和推理，给出另一种从 UML 活动图自动生成测试场景的途径。为了更好地处理含有分叉-汇合的活动图，摒弃了文[3]中的原子分叉-汇合(AFJ)类型，因为它是简单的分叉-汇合种类 (SFJ) 的一种特例。重新将分叉-汇合结点分成以下四类：简单的分

叉-汇合种类（SFJ）、简单的嵌套分叉-汇合种类（SNFJ）、分叉嵌套分叉-汇合种类（BNFJ）、混合分叉-汇合种类（MFJ，即分叉和汇合没有配成对，故将它们称为混合的分叉-汇合）。如图 9 所示。文[7]针对这几种不同类型的含有分叉-汇合的活动图给出不同的算法。其实现方法概述如下：

1）为分叉-汇合中的每个结点建立一个计算“候选集”；

2）为候选迁移集中的每个迁移分配操作的优先权，即如果迁移的目标是汇合（join）结点，则它的优先级低于迁移的目标是动作的结点；

3）依据嵌套分叉-汇合和简单的分叉-汇合的不同，作出这样的规定：如果迁移的目标是分叉（fork）结点，则它的优先级高于迁移的目标是动作的结点。如果某个结点候选迁移集中的 k（k 小于候选迁移集中的迁移数）个迁移的目标是相同的汇合结点，则创造一个新结点作为这个结点的孩子。此新结点的候选迁移集是它父结点的候选迁移集除去 k 个迁移并上它的输出集。这样就通过合并一些迁移。根据这个思想得出从含有嵌套分叉-汇合的活动图中生成测试场景的算法。

4）依据嵌套分叉-汇合和分支嵌套分叉-汇合结构的不同，作出这样的规定：如果迁移的目标是分支（branch）结点，则它的优先级高于迁移的目标是动作（action）的结点。得到从含有分支嵌套分叉-汇合的活动图推出测试场景的算法。

5）利用表来记录场景树的结点。如果某个结点候选迁移集中的所有迁移的目标是相同的汇合结点，则创造一个新结点作为这个结点的孩子。否则，对于每个迁移的目标结点作为这个结点的孩子，此新结点的候选迁移集是它父结点的候选迁移集除去该迁移并上它的输出集。

6）不断循环直到表为空。

文[7]还对含有循环的 UML 活动图进行较为深入的分析。在活动图中，循环是可以嵌套的、混联的，还存在无限的循环。在生成测试场景时，制定循环覆盖准则是必要的。例如让循环至多执行一次等。一般而言，在有嵌套循环的活动图中，设定每个循环可以被执行 n 次，而 n 的默认值是 1，并且 n 可以被测试人员赋予一个新的大于 1 的有效值。由此可以推出；

如果有 n 层嵌套循环，执行的次数分别为 m_1，m_2，…m_n，那么第 i 层循环（$1\leqslant i\leqslant n$）执行的次数是 $m_1\times m_2\times\cdots\times m_i$。

在上面生成的场景树中，从树根到树叶的每一条路径就构成一个测试场景。整个算法的流程图见图 10。

4 从 UML 活动图中生成测试场景方法的比较

以上介绍从 UML 活动图中生成测试场景的几种方法，文[5]介绍了从 UML 活动图产生测试用例的概念模型。反蚂蚁 Agent 方法和自适应 Agent 方法主要受到了人工智能技术启迪，反蚂蚁 Agent 方法使用从活动图中自动生成测试线程，再进一步生成测试场景。下面是简单的述评。

4.1 关于反蚂蚁 Agent 方法

反蚂蚁 Agent 方法拓展了生成测试线程的方法，使得从活动图中自动生成测试线程。但是这种算法主要存在以下缺点：

- 一群反蚂蚁 Agent 从活动图的开始结点进行搜索，为了达到活动图的某些部分而不得不对某些边重复多次访问，以增加它的信息素浓度。因此导致对活动图进行冗余的搜索，降低生成过程的效率。
- 对于一个活动图，开始派发蚂蚁的数量 m，很难精确确定。m 值过大，增大开销；m 值过小，无法保证活动图的每个结点和边都至少被访问一次。
- 反蚂蚁 Agent 方法没有限制分叉的输出边的数目，但处理一个分叉-汇合对生成的测试场景只包含了在分叉和汇合之间的通过浮桥连接的执行路径，因此对分叉-汇合对可能遗漏一些测试场景。

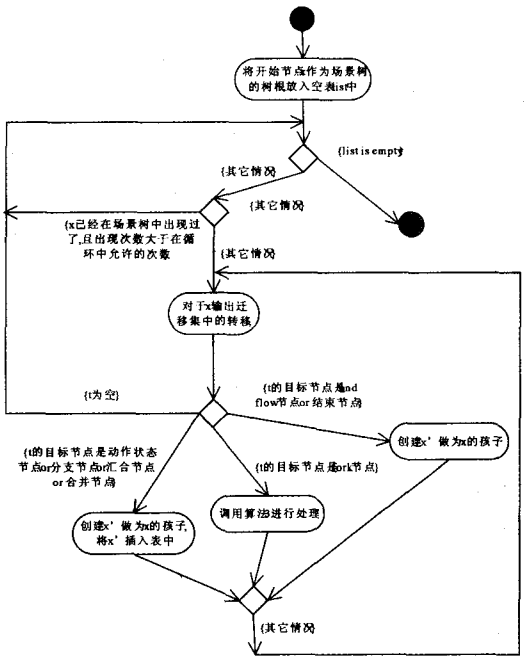


图 10 算法流程图

4.2 关于自适应细菌 Agent 方法

自适应 Agent 方法取消对于活动图的大部分限制，允许分叉结点有多于两条的输出边，这与反蚂蚁 Agent 方法相同。但是它利用一个 Agent 从开始结点进行探索，在需要时让 Agent 进行克隆，让更多的 Agent 进行探索。在特定情况下，这种 Agent 会变成种胚而从搜索过程中移走。与反蚂蚁 Agent 方法相比，避免了冗余的搜索。自适应 Agent 方法能够为所提出的几种分叉-汇合对结构生成所有的测试场景。但是这种算法也存在以下缺点：

- 在处理 SFJ 时需要建立一个堆栈，但要在 UML 活动图的规格说明中自动识别 SFJ 比较困难，需要编写新的识别算法。
- 需要分裂 NFJ 和 BNFJ 成单个的 SFJ，同样需要编写新的分裂算法。
- 不能够自动处理活动图中的混合分叉-汇合。

4.3 关于灰盒方法

使用灰盒方法生成测试场景，再生成测试用例。定义了基本路径、测试场景等基本概念。采用深度优先遍历活动图。对 UML 活动图做了一些可测性限制，这就限制了方法本身的适用范围。

- 文[1]假设活动图分叉结点只有两条输出边。事实上分叉结点可有多条输出边，这就限制了该方法的使用范围。
- 采用深度优先遍历活动图得到的测试场景是不完整的。由于活动图是含有循环和中断的有向图，从技术上讲，采用深度优先遍历或广度优先遍历活动图，很难达到一定的测

（下转第 281 页）

我们利用用户态的JVM对驱动程序进行隔离，这样充分保护了整个系统内核的安全。严格定义了Java虚拟机和外部的接口，并且利用用户态进程的虚拟机将驱动程序对外界的影响降到最小。出于性能和现有操作系统的中断处理机制，我们将与硬件直接相关的部分留在了内核中，并在其上增加了一个内核管理模块，用于和上层JVM进行通信。这样的结构可以监控每个进入内核的请求，严格控制了上层驱动对内核的操作行为，起到了隔离的作用。

将原Linux内核中的USB协议栈移植到了图1所示系统中。系统性能测试表明，我们的系统可以满足一般USB设备的中断响应要求，数据吞吐量可以满足摄像头设备的要求。但与原本地代码驱动相比，数据处理能力下降了约50%。相信这些性能的降低随着计算机硬件性能的不断提高可以得到改善，而新的系统架构将会保证操作系统在一个更为稳定的状态运行。

参考文献

1 毛德操，胡希明. LINUX内核源代码情景分析. 杭州：浙江大学出版社，2001
2 Chou Andy, Yang Junfeng, Chelf B, et al. An empirical study of operating systems errors. In: Proceedings of the Eighteenth ACM Symposium on Operating Systems principles, Oct. 2001
3 Härtig H, Hohmuth M, Liedtke J, et al. The performance of microkernel based systems. In: Proceedings of 16th ACM Symposium Operating System Principles, 1997. 66～77
4 Herder J N, Bos H, Gras B, et al. Construction of a highly dependable operating system. In: Dependable Computing Conference, 2006. EDCC '06. Sixth European, Oct, 2006. 3～12
5 Herder J N, Bos H, Gras B, et al. Tanenbaum. The architecture of a reliable operating system. In: 12th ASCI conference, Jun, 2006
6 Herder J N, Bos H, Gras B, et al. Tanenbaum. Minix 3: a highly reliable, self-repairing operating system. ACM SIGOPS Operating Systems Review, Jul. 2006(3)
7 Hunt G, Larus J R, Abadi M, et al. An overview of the singularity project: [Technical report]. Microsoft Research, Oct 2005
8 Ostrand T J, Weyuker E J. The distribution of faults in a large industrial software system. ACM SIGSOFT Software Engineering Notes, Jul. 2002(4)
9 Rubini A, Corbet J. Linux Device Drivers. 2nd edition. OReilly, Jun 2001
10 Swift M M, Bershad B N, Levy H M. Improving the reliability of commodity operating systems. ACM Transactions on Computer Systems (TOCS), Feb. 2005(1)
11 Swift M M, Martin S, Levy H M, et al. Nooks: an architecture for reliable device drivers. In: EW10: Proceedings of the 10th Workshop on ACM SIGOPS European Workshop: Beyond the PC, New York, NY, USA: ACM Press, 2002. 102～107
12 Tanenbaum A S, Woodhull A S. Operating Systems Design and Implementation. 3rd edition. Prentice Hall, Jan. 2006
13 Tanenbaum A S, Herder J N, Bos H. Can we make operating systems reliable and secure? IEEE Computer, May 2006, 39(5): 44～51

(上接第267页)

试覆盖准则。

·假设活动图中的每个循环都执行一次，没有讨论嵌套循环等更为复杂的结构。

4.4 关于系统的形式方法

在文[7]中提出的系统化方法，是对自适应细菌Agent的一种改进：1)在处理SFJ时不需要建立一个堆栈，而是通过为每个结点定义一个迁移"候选集"和"迁移输出集"，再对候选集的每个迁移分配操作优先级来实现。2)不需要分裂NFJ和BNFJ成SFJ，而是重新定义了算法自动完成测试场景的生成。3)能够自动处理活动图中的混合分叉-汇合。4)在工具的实现上，定义了一个描述UML活动图的XML Schema，从UML建模工具(或其它插件)生成的XMI文件中提取表示UML活动图的XML文档。工具读入XML文件，然后直接生成测试场景，整个生成过程能够自动完成。但是这种方法也不能适用从更一般的UML活动图产生测试场景\用例，也同样没有包括UML的全部建模元素，如中断、拴(Pin)等，尚需进行进一步的研究。几种方法的算法效率和适用范围见表1。

表1 算法效率和适用范围

方法	算法效率	适用范围
反蚂蚁Agent	慢	较小
自适应细菌Agent	较慢	较大
灰盒方法	较快	小
系统形式方法	快	大

进一步的工作 本文介绍了从UML活动图中生成测试场景的方法，总结了几种从UML活动图生成测试场景\用例的方法及其使用的算法，即反蚂蚁Agent方法、灰盒方法、自适应细菌Agent方法和系统的形式化方法。对这些方法进行了分析与比较，指出一些不足之处。

由于UML活动图语义的不精确性，各种活动状态之间的存在多种组合，且没有严格的规则，基于UML活动图的测试需要进一步的研究：

·要对UML活动图各种活动状态之间的多种组合进行进一步的总结分类，如上述方法中对分叉-汇合类型的划分也不是充分的，需要进一步研究。

·要考虑到UML活动图中的其他元素，特别是UML 2.0对活动图新增的元素，如流终结状态、发送和接受信号、信息栓(Pin)、中央缓存结点[8]等。

·要考虑到UML活动图的其他逻辑结构，如异常处理机制、可中断区域、扩展区域和多维分区[8]等。

·针对UML活动图的新的元素和新的结构，需要扩展测试覆盖准则，以提高测试的充分性。

·现有的UML活动图的测试覆盖准则尚需进行评估。

参考文献

1 Li H, Lam C P. Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams. In: Proc. TESTCOM 2005, LNCS 3502, Montreal, 2005
2 Wang L, Yuan J, Yu X, et al. Generating Test Cases from UML Activity Diagram Based on Gray-Box Method. In: Proc. APSEC'04, 2004
3 Xu Dong, Li H, Lam C P. Using Adaptive Agents to Automatically Generate Test Scenarios from the UML Activity Diagrams. In: Proc. of 12th APSEC 2005, IEEE Computer Society Press, Taipei, 2005
4 Bai X, Lam C P, Li H. An Approach to generate the Thin-threads from the UML Diagrams. In: Proc. COMPSAC 2004, Hong Kong, 2004
5 张楣，刘超，孙昌爱. 基于UML活动图模型的测试用例生成技术研究. 北京航空航天大学学报，2001，27(4)
6 袁洁松，王林章，李宣东，等. UMLTGF：一个基于灰盒方法从UML活动图生成测试用例的工具. 计算机研究与发展，2006(1)
7 Xu Dong, Lam C P, Li H Z. A Systematic Approach to Automatically Generate Test Scenarios from UML Activity Diagrams. In: The Proceeding of the Third IASTED International Conference on Advances in Computer Science and Technology, April 2-4, 2007, Phuket, Thailand, 2007
8 Object Management Group. Unified Modeling Language: Superstructure. version 2.0 July 2005. Available at www.omg.org