基于覆盖率信息的软件可靠性增长测试实践

李秋英[1], 李海峰[1], 徐 刚[2]
(1. 北京航空航天大学 可靠性与系统工程学院, 北京 100191; 2. 中国兵器工业第五九研究所, 重庆 400039)

摘 要: 软件可靠性增长测试是公认的实现软件可靠性增长以满足软件可靠性定量要求的重要手段,但数量庞大的测试用例往往成为其在实践中进行大规模推广和应用的瓶颈。通过在传统软件可靠性增长测试中引入覆盖率信息的方法来降低可靠性增长测试中需要执行的测试用例数量,从而加快可靠性测试进程,并解决其在实际中不能应用的弊端。通过实践的尝试,希望架起软件可靠性理论研究与工程实践之间的桥梁,以推动软件可靠性工程活动在工程实际中的开展和应用。

关键词: 软件可靠性; 软件测试; 软件可靠性增长测试; 覆盖率; 工程实践
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2010)07-2594-04
doi:10.3969/j.issn.1001-3695.2010.07.054

Software reliability growth testing practice based on coverage

LI Qiu-ying[1], LI Hai-feng[1], XU Gang[2]
(1. *School of Reliability & Systems Engineering, Beihang University, Beijing* 100191, *China*; 2. *China Ordnance Industries No.* 59 *Research Institute, Chongqing* 400039, *China*)

Abstract: Software reliability growth testing is an important measure to increase the software reliability to meet its reliability requirement. But the huge quantity hinders its promotion and application in practice. This paper introduced coverage information into the process of software reliability growth testing to reduce the quantity of test case executed, which accelerated the reliability testing and overcome its shortcoming in practice. Also presented a practical work on a real software product, which aimed to promote the development and practice in the engineering field.

Key words: software reliability; software testing; software reliability growth testing; coverage; engineering practice

相对于硬件可靠性工程的大量应用,软件可靠性工程理论上取得了较大的突破,但工程实践上开展的却不多,一方面,既有方法本身不成熟、理论假设尚与工程实际存在差距的原因,也有缺乏实际应用的经历、不能通过实践检验理论的正确性并推动理论的进一步完善的原因;另一方面,统计学的理论基础使得软件可靠性测试在应用上往往意味着大数据量。例如,可靠性测试数据的数量往往是非常庞大的,人们对于实践的尝试往往因数量的庞大而却步。本文对能否在保持其本质特征的前提下,进行软件可靠性测试进程的加速,使得人们对软件可靠性加速测试展开了研究。

文献[1]从概念出发,给出了软件的加速强度测试的定义,并结合硬件说明了软件加速强度测试中的基本可靠性规律。文献[2,3]分别基于严酷度和重要度的思想,提出加大关键/重要操作的测试方法,加快暴露影响系统安全的缺陷。基于覆盖率信息也是软件可靠性加速测试思想之一,文献[4,5]提出了利用判定覆盖和数据流覆盖等信息预测软件失效的方法,利用参数化因子剔除在测试过程中既不增加覆盖率也不会引起软件失效的执行时间,从而解决测试方法的"饱和效应"问题。文献[6]是笔者在分析了软件可靠性测试用例的特征之后,提出了将覆盖率引入可靠性测试进程的灰盒测试方法,既保证可靠性测试统计特征,又充分利用覆盖信息,即统计特征和充分性信息一起辅助决策测试过程,这样的思想在后续发表的文献[7]中也得到了很好的体现。虽然理论上提出了上述成果,但大多数成果都未经过实践的检验。作为软件可靠性工程的一项重要内容,软件可靠性测试及加速测试在未来的应用都是必然的,需要提供细节为工程实践提供参考。为此,本课题组根据国内外已有的研究成果、应用经验,开发出一些较为具体的、面向实际应用的方法、工具为实际开展这项工作提供帮助,并结合某一软件从步骤、数据、结果等具体开展这方面的工作。本文旨在通过实践,不仅探讨覆盖率信息指导下的软件可靠性加速增长测试的理论和方法的可行性,而且希望通过这一实例,为软件可靠性测试方向理论的深入研究和工程的实际推广提供一个值得借鉴的范例和参考。

1 方法的基本原理

软件可靠性增长测试的概念和内容可参考文献[8]。基于覆盖率信息的软件可靠性增长测试的主要思想是对自动生成的大量可靠性测试用例利用覆盖率信息过滤无用的测试用例,即剔除不增加任何测试覆盖率的测试用例,而只应用能够增加某种测试覆盖率的有效测试用例进行可靠性测试,对收集到的失效数据进行相应处理并进行评估,从而实现加速软件可靠性增长测试的目的。

收稿日期: 2009-12-31; **修回日期**: 2010-02-26

作者简介:李秋英(1973-),女,黑龙江大庆人,博士,主要研究方向为软件可靠性工程、软件可靠性测试及软件可靠性测试充分性理论(li_qiuying@buaa.edu.cn);李海峰(1981-),男,博士研究生,主要研究方向为软件可靠性度量与建模;徐刚(1983-),男,助理工程师,主要研究方向为建设项目实施管理.

所谓无用的测试用例是借鉴文献[9]中提出关于有用测试量的概念。有用测试量是指当且仅当扩大了某种覆盖的测试量。例如,k 个测试数据达到了 15% 的语句覆盖,若 $k+1$ 个测试数据使得覆盖达到 20%,则说第 $k+1$ 个测试数据是有用的。值得注意的是,无用的测试用例不是真正意义上的无用,实际上文献[10]曾指出测试覆盖率与发现缺陷之间不存在必然联系。但由于这种关系的尚未可知,假定其在揭示软件失效方面的作用可以忽略的结论目前仍可以接受,即在发现软件缺陷方面,认为无用的测试用例是没有贡献的。但在可靠性评估方面,因为无用的测试用例也是按照操作剖面被抽取出来的,没有理由将它们视为不存在,即不能忽略其所占用的测试时间,在评估时需要通过补偿一定的测试时间来考虑这部分用例所起到的作用。

图 1 给出了基于覆盖率的软件可靠性增长测试流程,其中虚框中的部分是该方法区别于传统可靠性增长测试的主要步骤。

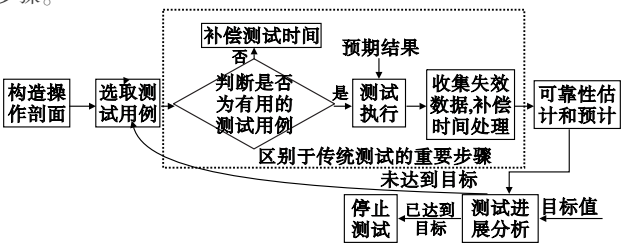

图1　基于覆盖率的软件可靠性增长测试流程

具体解释如下:

a)构造操作剖面,选取测试用例。

b)逐个分析测试用例集中每个用例的覆盖率情况(通常相对于执行用例来说,分析所占用的时间要少得多)。设执行第 i 个用例后测试覆盖率为 c_i,若该测试用例能增加测试覆盖率,即 $c_i > c_{i-1}$,则转向 c);否则转向 d)。

c)对第 i 个用例执行测试,并记录真实的测试时间 t_i(与上一个测试用例之间的时间间隔)。若发生了失效,设该失效的累积发生时间为 t,显然,$t = \sum_{m=1}^{i} t_m$,转向 e)。

d)对第 i 个用例不执行测试,给其一个补偿测试时间 $t_i = t_{\text{buchang}}$。补偿时间的确定通常有两种方法:一种是由开发人员对用例的复杂程度进行判断,直接给出可能的测试时间;另一种是先进行少量用例的执行,计算平均的测试时间,将平均测试时间作为补偿测试时间,转向 e)。

e)整理失效数据信息并进行软件可靠性评估。如果可靠性估计值达到目标值要求,则结束测试;否则转向 b)。

由上面的测试步骤可以明显看出,由于基于覆盖率信息的可靠性增长测试方法对不增加覆盖率的测试用例不进行测试,则较传统的增长测试方法可节省大量的测试时间和工作量。下面就通过实例来考察该方法可靠性评估的有效性。

2　实验与结果分析

为了与传统的软件可靠性测试进行对比,实验的形式是采用两个小组针对同一软件背靠背分别进行实验。设小组 A 的任务为进行传统的软件可靠性增长测试和评估,小组 B 的任务为在小组 A 生成的可靠性测试用例基础上,基于覆盖率信息的方法进行可靠性增长测试与评估。

2.1　被测软件介绍

被测软件是本课题组开发的一个商场数据库管理系统,借助它可以充分地了解商场的销售情况、库存情况和供应情况。该软件包含八个人机交互界面,其中一个为主操作界面,可通过按钮调用其他界面,其他界面之间不能相互调用。该系统在数据库方面拥有商品、连锁店、仓库、销售、库存、供应共六个互相联系的表。

实验前该软件已经过了充分的调试和测试,排除了基本缺陷。之所以选择这样的软件是为了测试过程的可控。在测试前对被测软件植入 50 个缺陷,通过对前期使用数据的积累,能够获得较为精确的操作剖面。这样,当发现缺陷时可以对缺陷立即排除。

2.2　构造操作剖面

采用 Musa 的经典方法自顶向下逐层细化来构造操作剖面[10]。最终的操作剖面如表 1 所示。

表 1　软件操作剖面

操作名称	发生概率	操作名称	发生概率	操作名称	发生概率
仓库信息保存	0.0015	库存信息全部	0.030817	统计信息到货情况	0.019856
仓库信息查找	0.01232	库存信息删除	0.021686	统计信息发货情况	0.020951
仓库信息取消	0.0015	库存信息添加	0.0033	统计信息供应情况	0.020951
仓库信息全部	0.013141	库存信息添加保存	0.03	统计信息极限总销售量	0.000985
仓库信息删除	0.009035	库存信息添加取消	0.027897	统计信息商品信息	0.020404
仓库信息添加	0.0015	连锁店信息保存	0.0022	统计信息月最低销售量	0.000985
仓库信息添加保存	0.009	连锁店信息查找	0.01232	统计信息月最高销售量	0.000985
仓库信息添加取消	0.005201	连锁店信息取消	0.0022	退出	0.052655
供应信息保存	0.0022	连锁店信息全部	0.012731	销售信息保存	0.0033
供应信息查找仓库号	0.013852	连锁店信息删除	0.009856	销售信息查找连锁店号	0.023303
供应信息查找商品号	0.01847	连锁店信息添加	0.0022	销售信息查找商品号	0.021071
供应信息查找月份	0.023852	连锁店信息添加—保存	0.012	销售信息查找月份	0.023303
供应信息取消	0.0022	连锁店信息添加—取消	0.010101	销售信息取消	0.0033
供应信息全部	0.014382	商品信息保存	0.0022	销售信息全部	0.026355
供应信息删除	0.016653	商品信息查找	0.032789	销售信息删除	0.041904
供应信息添加	0.0022	商品信息取消	0.0022	销售信息添加	0.0033
供应信息添加保存	0.026	商品信息全部	0.0351	销售信息添加保存	0.03
供应信息添加取消	0.024392	商品信息删除	0.03513	销售信息添加取消	0.022882
库存信息保存	0.0033	商品信息添加	0.0022	连锁店信息删除	0.009856
库存信息查找仓库号	0.024812	商品信息添加保存	0.025	连锁店信息添加	0.0033
库存信息查找商品号	0.024812	商品信息添加取消	0.022067	连锁店信息添加—保存	0.03
库存信息取消	0.0033	统计信息储量统计	0.009856	连锁店信息添加—取消	0.022882

2.3　传统软件可靠性增长测试与评估

依据传统的软件可靠性增长测试工作流程,对商场信息管理系统软件进行软件可靠性增长测试,主要步骤包括:

a)构造操作剖面并生成测试用例。构造如 2.2 节所述的被测软件的操作剖面,并利用自行开发的可靠性测试用例生成工具(TCS)[12]自动生成了 800 个测试用例。测试用例的形式如表 2 所示。

表 2　部分可靠性增长测试用例

软件名称	系统	内容
库存信息添加—取消	商场数据库管理系统	版本　version1.0　用例编号　SRT156 第 1 步　仓库号 = 56 第 2 步　商品号 = 21 第 3 步　库存量 = 8337 第 4 步　文字描述 = 取消添加内容,点击按钮 预期结果　取消添加库存信息,包括仓库号、商品号、存入日期、现存量、显示原状
商品信息查找	超市信息管理系统	版本　version 1.0　用例编号　SRT157 第 1 步　商品号 = 30 预期结果　查找所输入商品号对应的商品名和单价
库存信息查找商品号	超市信息管理系统	版本　version 1.0　用例编号　SRT158 第 1 步　商品号 = 78 预期结果　显示存有商品号对应商品的仓库号、保存日期、现存量

b)执行测试用例并记录失效数据。在测试过程中记录缺陷的发生时间,对发现的缺陷立即排除,并假设不引入新的缺陷。本次软件可靠性增长测试共发现失效 50 个。失效数据信息如表 3 所示。

表 3　传统可靠性测试收集的失效数据信息

失效序号	累积失效时间/s	失效序号	累积失效时间/s	失效序号	累积失效时间/s	失效序号	累积失效时间/s
1	14	14	467	27	558	40	4201
2	27	15	1888	28	1960	41	5646
3	49	16	645	29	2021	42	5864
4	111	17	716	30	2046	43	7162
5	129	18	909	31	2189	44	7410
6	168	19	989	32	2413	45	8889
7	190	20	1020	33	2522	46	9844
8	215	21	1225	34	2731	47	12086
9	256	22	1406	35	3030	48	16002
10	274	23	1457	36	3318	49	17464
11	380	24	1564	37	3389	50	18989
12	418	25	1602	38	3446		
13	445	26	1758	39	4045		

c)可靠性评估。得到失效数据之后,利用软件可靠性模型对失效数据进行分析,以得出软件的可靠性估计值。在自行开发的软件可靠性分析工具(SRAT)[13]中提供了若干种经典的可靠性模型,可以根据不同失效数据集的特点,选取合适的软件可靠性模型对被测软件的可靠性水平值进行估计。将本次可靠性测试中收集到的失效数据集输入 SRAT 软件中,得到如下的被测软件可靠性评估结果:

对商场数据库管理系统软件的可靠性失效数据进行分析,从第 1 点开始分析数据,为可靠性增长,所推荐选用的模型是 GO 模型,评估结果为当前失效强度 ROCOF = 5.39E - 5(/s)。

2.4　基于覆盖率信息的软件可靠性增长测试与评估

基于覆盖率信息的软件可靠性增长测试与传统的软件可靠性增长测试的流程大体相同,分为以下几个部分:

a)确定操作剖面并生成测试用例。操作剖面与 2.2 节构造的方法相同,此处就利用 2.2 节的结果,并利用 2.3 节中生成的 800 个测试用例。

b)测试运行及失效记录。从第 1 章可以看出,基于覆盖信息的软件可靠性测试与传统软件可靠性测试的不同主要体现在测试运行步骤上,即基于覆盖信息的软件可靠性测试需要逐步分析用例,且只对有用的测试用例进行测试;而传统的软件可靠性测试则对所有的用例都要测试。例如,表 4 给出的是实际执行的测试用例,即能够增加覆盖率的有用的测试用例。

条件所限,本文采用的覆盖率信息为用例执行时的语句覆盖率,并利用 True Coverage 软件自动记录测试过程中的语句覆盖率情况。借鉴部分可靠性测试用例执行信息及开发人员的经验,单个用例平均的测试时间取 24.98 s,因此为每一个无用的测试用例补偿的平均测试时间为 25 s,这种做法被证实在工程实际中是可接受的,避免因为担心影响最终评估结果的精度而必须执行所有的测试用例。例如,表 4 中用例编号为 13 的测试用例的累积失效时间 262 的由来包括两部分,用例 11 发现第 8 个缺陷的累积时间是 215 s,用例 12 因为未增加覆盖率,所以不执行,补偿测试时间为 25 s,用例 13 发现第 9 个缺陷的失效间隔时间为 22 s,因此,用例 13 发现第 9 个缺陷时的累积失效时间为 215 + 25 + 22 = 262 s。篇幅所限,表 4 列出的累积失效时间是经过补偿时间处理后的失效数据信息。

表 4　有用的测试用例收集的失效数据信息

用例编号	失效编号	累积失效时间/s	覆盖率/%	用例编号	失效编号	累积失效时间/s	覆盖率/%
1	1	14	0.5272	64	22	1530	56.7663
2	2	27	2.8120	66	23	1583	58.8752
3	3	49	7.5571	71	24	1706	61.5114
4	—	93	10.3691	72	—	1720	62.5659
5	4	111	12.4780	73	25	1744	62.8697
6	5	129	16.5202	79	26	1909	63.1124
7	—	155	19.5079	86	27	2076	63.2046
8	6	168	21.7926	89	28	2170	63.4446
9	—	177	22.8471	92	29	2243	63.6012
10	7	190	25.6591	93	30	2268	63.7961
11	8	215	28.9982	97	—	2377	64.3234
13	9	262	31.2830	99	31	2422	65.7293
14	10	280	34.4464	108	32	2640	65.9986
15	—	302	35.3251	111	33	2723	66.2566
17	—	351	36.5554	119	34	2953	67.8435
18	—	360	36.9069	127	35	3151	69.2443
19	11	377	37.1358	137	36	3410	70.1230
21	12	412	37.7856	140	37	3502	70.2988
22	13	439	39.8946	143	38	3571	71.1775
23	14	461	40.4218	145	—	3608	71.3533
25	—	500	40.5975	161	—	4004	72.0562
26	—	520	41.6520	165	39	4137	73.4622
28	15	574	42.1793	171	40	4323	73.9823
29	—	589	43.2337	225	41	5673	74.5167
33	16	683	43.7610	234	42	5900	74.8476
34	—	704	45.6942	264	—	6674	75.3954
36	17	755	46.5729	281	43	7086	76.8014
37	—	801	47.2759	290	44	7318	78.9104
44	18	974	49.9121	338	45	8531	79.6134
48	—	1065	50.9666	378	46	9527	81.3356
49	19	1089	53.9543	469	47	11803	82.0213
50	20	1120	54.4815	633	48	15902	82.5415
51	—	1164	55.0088	688	49	17286	83.7622
57	21	1329	55.6742	759	50	19060	84.7785

注:表中的“—”表示该用例执行时未发现失效。

c)可靠性估计。利用 SRAT 软件对表 4 中的失效数据信息进行分析,得出如下的可靠性估计结果:

对商场数据库管理系统软件的可靠性失效数据进行分析,从第 1 点开始分析数据,为可靠性增长,选用模型是 GO 模型,评估结果为当前失效强度 ROCOF = 5.65E - 5(/s)。

2.5　结果分析

将基于覆盖率信息的软件可靠性测试的数据信息和评估结果与传统的软件可靠性测试相比较,可以发现:

a)传统软件可靠性测试的测试工作量为 800 个测试用例,而基于覆盖率信息的软件可靠性测试的测试工作量仅为 68 个测试用例,缩减工作量 91.5%,大大加快了可靠性测试进程。实际上,在本实验中进行一次传统可靠性增长测试的时间远超过 5 h,而加速后的测试时间仅为 30 min,再加上分析的时间,不到 1 h。

b)发现缺陷的效率并未降低,图 2 将两种实验下的失效数据信息绘制在一张图中,可以看出加速后的测试过程基本覆盖了原始测试过程的失效信息,在缩短测试时间的情况下发现了与传统实验同样多的失效。

c)分析两次实验的可靠性评估结果,相对误差为 4.8%,

即加快测试进程的同时并未带来评估精度的降低。

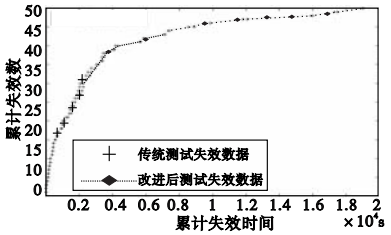

图2　累积失效数据对比图

3　结束语

本文通过一个基于覆盖率的软件可靠性增长测试实践,分析了该方法的可行性和有效性。与传统的软件可靠性测试相比,基于覆盖率信息的软件可靠性增长测试不但可以更加快速地发现被测软件中的缺陷、大大降低测试的工作量,而且可以较为准确地对软件可靠性水平进行评估。此外,本文通过对整个测试过程中需要进行的重点工作、收集哪些数据、如何进行结果分析等方面的整理和总结,为软件可靠性测试及加速测试等方面的进一步研究奠定了一定的应用基础和提供一定的参考。由于受时间和条件所限,目前的研究还存在不足,还可以从以下方向进行努力:

a)增加所选用软件的代表性,缩短与工程实际应用软件的差距。

b)增加覆盖率的种类,特别是已有工具支撑的、且在覆盖率准则中包含关系处于上层的覆盖率[10]的应用。

c)提高测试过程的自动化水平。

参考文献:

[1] CHAN H A. Accelerated stress testing for both hardware and software [C]//Proc of Annual Reliability and Maintainability Symposium. 2004:346-351.

[2] TANG Dong, HECHT H. An approach to measuring and assessing dependability for critical software systems[C]//Proc of the 8th International Symposium on Software Reliability Engineering. 1997:192-202.

[3] HECHT M, HECHT H. Use of importance sampling and related techniques to measure very high reliability software[C]//Proc of Aerospace Conference Proceedings. 2000:533-546.

[4] CHEN M H, LYU M R, WONG W E. Effect of code coverage on software reliability measurement[J]. IEEE Trans on Reliability, 2001,50(2):165-170.

[5] BISHOP P G. Estimating residual faults from code coverage[C]// Proc of the 21st International Conference on Computer Safety, Reliability and Security. 2002:10-13.

[6] 李秋英,刘斌,阮镰. 灰盒测试方法在软件可靠性测试中的应用[J]. 航天学报,2002,23(5):455-458.

[7] KUBALL S,MAY J. Test-adequacy and statistical testing: combining different properties of a test-set[C]//Proc of the 15th International Symposium on Software Reliability Engineering. 2004:161-172.

[8] 陆民燕. 软件可靠性增长测试的研究[J]. 航空学报,1995,16(增刊):88-93.

[9] CHEN M H, HORGAN J R, MATHUR A P, *et al*. A time/structure based model for estimating software reliability, Technical Report SERC-TR-117-P[R]. West Lafayette, Indiana: Purdue University, 1992.

[10] 朱鸿, 金凌紫. 软件质量保障与测试[M]. 北京:科学出版社, 1997:75-85.

[11] LYU M R. 软件可靠性工程手册[M]. 刘喜成,译. 北京:电子工业出版社,1997:167-193.

[12] 艾骏, 陆民燕, 阮镰. 实时嵌入式软件可靠性测试数据自动生成方法[J]. 测控技术,2007,26(3):59-61.

[13] 丛珉, 陆民燕, 白云峰. 软件可靠性预计方法研究及实现[J]. 北京航空航天大学学报,2002,28(1):34-38.

(上接第 2593 页)

5　结束语

本文提出并实现了一种基于流演算的动态规划程序设计语言 DPPLFC。该语言通过定义动作表达式对 FLUX 进行扩充,能够描述顺序、并发、非确定选择等复杂动作,方便用户编写应用程序。通过重新定义谓词 Trans 和 Final,给出了 DPPLFC 程序的语义。DPPLFC 语言的动态规划算子高效地实现了动态环境下的再次离线规划机制。DPPLFC 语言为动态环境下的智能机器人的动态规划提供了一种解决方案。实例证明了 DPPLFC 语言的可行性和高效性。

参考文献:

[1] 吴向军,姜云飞,凌应标. STRIPS 规划领域中动作效果关系的研究[J]. 软件学报,2007,18(6):1328-1349.

[2] LEVESQUE H J, REITER R, LESPERANCE Y, *et al*. GOLOG: a logic programming language for dynamic domains[J]. Journal of Logic Programming,1997,31(1-3):59-83.

[3] GIACOMO G D, LESPERANCE Y, LEVESQUE H J. ConGolog: a concurrent programming language based on the situation calculus[J]. Artificial Intelligence,2000,121(1-2):109-169.

[4] GIACOMO G D, LEVESQUE H. An incremental interpreter for high-level programs with sensing[C]//Proc of Logical Foundation for Cognitive Agents: Contributions in Honor of Ray Reiter. Berlin: Springer, 1999:86-102.

[5] THIELSCHER M. Reasoning robots: the art and science of programming robotic agents [M]. Netherlands: Springer, 2005.

[6] THIELSCHER M. FLUX: a logic programming method for reasoning agent[C]// Proc of Theory and Practice of Logic Programming. New York: Cambridge University Press, 2005:533-565.

[7] LIU Yi-song, ZHONG Shan, ZHAN Yong-zhao. Reasoning about action for behavioral animation of intelligent virtual agents[J]. The International Journal of Virtual Reality, 2008, 7(3):37-42.

[8] 刘一松,文占朝. 基于流演算的智能虚拟人模型研究与实现[J]. 计算机应用研究,2009,26(8):2968-2970.

[9] LESPERANCE Y, NG H K. Integrating planning into reactive high-level robot programs[C]//Proc of the 2nd International Cognitive Robotics Workshop. 2000:49-54.