DOI:10.7544/issn1000-1239.2015.20131904
52(6):1443-1451,2015

Hybrid-Fixing:上下文一致性错误的正确修复

陈小康　许　畅　江　磊
(计算机软件新技术国家重点实验室(南京大学)　南京　210023)
(南京大学计算机科学与技术系　南京　210023)
(chenxiaokang@outlook.com)

Hybrid-Fixing: Toward Sound Fixing of Context Inconsistency

Chen Xiaokang, Xu Chang, and Jiang Lei
(*State Key Laboratory for Novel Software Technology* (*Nanjing University*), *Nanjing* 210023)
(*Department of Computer Science and Technology*, *Nanjing University*, *Nanjing* 210023)

Abstract　In pervasive computing, environmental contexts are subject to frequent and rapid changes, and context-aware applications adapt their behavior accordingly. However, context inconsistency occurs due to various reasons including unpredictable and uncontrollable environmental noises and dynamics, which results in application anomaly or even failure. To address this problem, context inconsistency should be timely detected and then fixed in an automated and sound way. Based on our previous work we propose two techniques, named complete-fixing and CoSound-fixing, to fix context inconsistency automatically for context-aware applications. However, the two techniques are subject to some limitations in that complete-fixing has a time complexity issue and does work so efficiently, and CoSound-fixing does not have a satisfactory fixing success rate. In this paper, we propose a new fixing technique, named hybrid-fixing, which combines the static analysis of consistency constraints and the dynamic generation of repair actions to ensure the soundness of its generated repair cases, even when there exist complex dependencies inside consistency constraints. Experimental results show that our hybrid-fixing significantly increases the fixing success rate for detected context inconsistencies, as compared with CoSound-fixing when facing complex dependencies inside consistency constraints, while still incurring minor time cost only and achieving the fixing of context inconsistency in a fully automated way.

Key words　pervasive computing; context; inconsistency; automatic fixing; sound fixing

摘　要　在普适计算中,上下文持续快速变化,上下文感知应用根据上下文变化自动调整自身的行为以作出适应.然而,由于不可预测和控制的环境噪声以及环境动态变化等诸多因素的影响,环境上下文会发生一致性错误,从而导致应用表现异常甚至失效.为了解决这些问题,上下文一致性错误需要被自动并正确地修复,现基于已有工作提出了一项新的修复技术 hybrid-fixing,它结合了对一致性约束的静态分析和修复动作的动态产生,即使一致性约束内部存在复杂依赖关系,也能确保所生成的修复用例必然正确.实验结果表明,这项修复技术大幅提高了一致性约束内部存在复杂依赖关系下一致性错误修复的成功率,并只花费了很小的时间开销.

收稿日期:2013-12-17;**修回日期**:2014-05-23
基金项目:国家"八六三"高技术研究发展计划基金项目(2013AA01A213);国家自然科学面上基金项目(61472174);国家自然科学基金集成项目(91318301);国家自然科学基金创新群体项目(61321491)
通信作者:许　畅(changxu@nju.edu.cn)

关键词 普适计算;上下文;一致性错误;自动修复;正确修复

中图法分类号 TP311; TP309.2; TP393.08

在普适计算中,我们用上下文(context)来描述环境信息[1],如位置、温度、音量等.上下文感知应用(context-aware application)通过感知上下文变化,自动地调整自身的状态或行为,以更好地适应用户需求和环境的变化[2].由于存在环境噪声等因素,常常会引起上下文一致性错误,导致上下文感知应用的异常甚至失效[3].又因为在这种情况下人工的处理方式常常是昂贵且容易出错的[4],所以需要我们自动修复上下文一致性错误.

如图1所示,生产车间有2条生产线 L_1,L_2,叉车把产品从 A 端运上 L_1,经中转部分 B 到达 L_2,完成加工后从 C 端运走.每件产品上都贴1个拥有唯一 id 的RFID标签.RFID阅读器 $R_A(R_B,R_C)$ 分别读取经过 $A(B,C)$ 部分产品的 id.产品从 A 端到 B 端、B 端到 C 端各需10 s,当前时刻为 t.为检测上下文一致性错误,我们分析最近20 s内的上下文,c_A (c_B,c_C) 分别表示 $R_A(R_B,R_C)$ 在时间段 $t-40$ s到 $t-20$ s($t-30$ s到 $t-10$ s,$t-20$ s到 t)读到的产品 id.我们定义公式 $f_{ex1}:=f_{sub1}$ and f_{sub2} 来检测 R_B 和 R_C 漏读[4]情况,其中 $f_{sub1}:=\forall v_1\in c_A{}^2(\exists v_2\in c_B{}^4(\mathrm{eq}(v_1.id,v_2.id)^1))$,$f_{sub2}:=\forall v_2\in c_B{}^2(\exists v_3\in c_A{}^4(\mathrm{eq}(v_2.id,v_3.id)^1))$,谓词eq是用来判断是否相等.公式 f_{ex1} 的意思是:c_A 中每个 id,c_B 中都存在1个相同的 id;且 c_B 中每个 id,c_C 中也都存在1个相同的 id.公式中数字上标表示用户对修复的偏好,将在后文中说明.假设 c_A 中有 id 为01和 id 为02两个元素;c_B 中有 id 为01和 id 为02两个元素;R_C 漏读了 id 为01的元素,故 c_C 中只有 id 为02的元素.显然,当前的上下文并不满足公式 f_{ex1}.我们称上下文不满足公式的情况为上下文一致性错误.

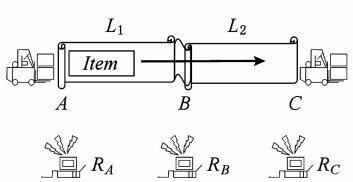

Fig. 1 The sample demonstration.

图1 样例演示

面对上下文一致性错误,传统的解决方案一般是研究如何删除上下文[3,5-6],如随机删除上下文、删除最可疑的上下文等策略.例如删除 c_B 中 id 为01的元素,这样修复的 f_{sub2} 同时却破坏了 f_{sub1},带来了新的一致性错误.我们的原有工作[7]提出一种修复上下文一致性错误的技术,能自动生成完整修复(complete-fixing)或正确修复(sound-fixing).若采用完整修复方案,我们可以生成3类修复策略:1)向 c_C 中添加 id 为01的元素;2)删除 c_B 中 id 为01的元素;3)把 c_B 中 id 为01和 c_C 中 id 为02的2个元素的 id 改为相等.但完整修复需要生成所有修复策略,效率相对较低,而且可以看出只有第1类策略是正确的,第2,3类策略修复了 f_{sub2} 的同时破坏了 f_{sub1} 使得 f_{ex1} 不满足,带来了新一致性错误,所以我们需要保证修复不带来新一致性错误,即正确性.正确修复是指所有修复策略都执行成功而不带来新一致性错误.原有工作[7]中能对简单约束(同1个上下文在某个公式中最多只出现1次)生成保守的正确修复技术(conservative-sound-fixing, CoSound-fixing),取得了很好的效果;但不能解决一致性约束内部存在复杂依赖关系(同1个上下文在某个公式中多次出现,如 f_{ex1} 中的 c_B)的情况.

为了解决一致性约束内部存在复杂依赖关系的难题,我们设计了1种复合式修复技术(hybrid-fixing).这种技术,通过静态分析公式特征来指导修复的动态生成过程,确保修复的正确性.

本文的贡献主要有2个方面:1)能力方面.即使在一致性约束内部存在复杂依赖关系情况下也能为一致性错误生成正确修复;又因为分析粒度更精细,能挑选出更多满足正确性的修复策略.从这2方面提高修复成功率.2)效率方面.采用全新复合式方法,首先静态分析不满足正确性的修复行为特征;然后指导修复的动态生成过程.在生成修复过程中及时抛弃不满足正确性的修复策略,大大减少工作量从而提高效率.如果修复时间过长,上下文过期,即使修复成功也没有意义,故效率很重要.

1 一致性错误的检测

1.1 上下文

定义1. 上下文 *context*.

$context::=\{element_1,element_2,\cdots\}$;

$element::=\{(att_1,val_1),(att_2,val_2),\cdots\}$.

如定义1所示,我们以元组空间[7-9]的形式来定

义上下文.上下文(*context*)定义为元素(*element*)的集合,元素定义为属性(*attribute*)到值(*value*)的映射的集合,属性和值都用字符串表示.引言中上下文可以表示为 $c_A:=\{\{(id,01)\},\{(id,02)\}\}$, $c_B:=\{\{(id,01)\},\{(id,02)\}\}$, $c_C:=\{\{(id,02)\}\}$.

1.2 公式定义

定义 2. 公式 f.

$$f::=\forall v\in c^w(f)\mid\exists v\in c^w(f)\mid(f)\text{ and }(f)\text{ and}\cdots\text{and}(f)\mid(f)\text{ or }(f)\text{ or}\cdots\text{or}(f)\mid(f)\text{ implies}(f)\mid\text{not}(f)\mid\text{eq}(v_i.att_i,v_j.att_j)^w.$$

如定义 2 所示,我们使用一阶逻辑公式[4,7,10-11]来表示约束.在我们的约束语言中有 7 种不同的公式(formula):全称量词(∀)、存在量词(∃)、并(and)、或(or)、蕴涵(implies)、非(not)、相等(eq).与原工作相比[7],我们拓展了谓词并(and)和谓词或(or),使得它们支持 2 个及以上子公式,从而使得约束的表达更加简洁紧凑.公式递归定义,数字上标 w 表示用户在公式的不同层次对上下文一致性错误进行修复的偏好权重,用正整数表示,w 越大则偏好越大.例如公式 f_{sub1} 中的权重表示用户在第 1,2,3 层对上下文进行修复的偏好权重分别为 2,4,1.

1.3 真值计算

定义 3. 真值计算函数 $\mathcal{T}$.

1) $\mathcal{T}[\![\forall v\in c^w(f)]\!]_\alpha:=\top\wedge\mathcal{T}[\![f]\!]_{bind((v,x_1),\alpha)}\wedge\cdots\wedge\mathcal{T}[\![f]\!]_{bind((v,x_n),\alpha)}\mid x_i\in c$;

2) $\mathcal{T}[\![\exists v\in c^w(f)]\!]_\alpha:=\bot\vee\mathcal{T}[\![f]\!]_{bind((v,x_1),\alpha)}\vee\cdots\vee\mathcal{T}[\![f]\!]_{bind((v,x_n),\alpha)}\mid x_i\in c$;

3) $\mathcal{T}[\![(f_1)\text{ and }(f_2)\text{ and}\cdots\text{and}(f_n)]\!]_\alpha:=\mathcal{T}[\![f_1]\!]_\alpha\wedge\mathcal{T}[\![f_2]\!]_\alpha\wedge\cdots\wedge\mathcal{T}[\![f_n]\!]_\alpha$;

4) $\mathcal{T}[\![(f_1)\text{ or }(f_2)\text{ or}\cdots\text{or}(f_n)]\!]_\alpha:=\mathcal{T}[\![f_1]\!]_\alpha\vee\mathcal{T}[\![f_2]\!]_\alpha\vee\cdots\vee\mathcal{T}[\![f_n]\!]_\alpha$;

5) $\mathcal{T}[\![(f_1)\text{ implies}(f_2)]\!]_\alpha:=\mathcal{T}[\![f_1]\!]_\alpha\rightarrow\mathcal{T}[\![f_2]\!]_\alpha$;

6) $\mathcal{T}[\![\text{not}(f)]\!]_\alpha:=\neg\mathcal{T}[\![f]\!]_\alpha$;

7) $\mathcal{T}[\![\text{eq}(v_i.att_i,v_j.att_j)^w]\!]_\alpha:=eqStr(getE(v_i,\alpha).att_i,getE(v_j,\alpha).att_j)$.

与文献[4,7]类似,为了计算公式的真值,我们用 V 表示公式中变量的集合,E 表示元素的集合.我们定义变量赋值(variable assignment)为变量到元素映射的集合,即 $A:=\wp(V\times E)$.变量赋值在初始情况下为空.为了操纵变量赋值,我们引入了函数 $bind:(V\times E)\times A\rightarrow A$,向已有变量赋值中加入 1 个新的映射从而构造出 1 个新的变量赋值.变量赋值中的每个变量只能绑定到 1 个元素,例如:映射 $m:=(v_2,\{(id,02)\})$,变量赋值 $\alpha:=\{(v_1,\{(id,01)\})\}$, $bind(m,\alpha):=\{(v_2,\{(id,02)\}),(v_1,\{(id,01)\})\}$.我们定义函数 $getE$ 从变量赋值中取出变量映射的元素.如定义 3 所示,我们用 F 表示公式的集合,真值计算函数 $\mathcal{T}:F\times A\rightarrow\{\top,\bot\}$,用来计算 1 个公式在给定变量赋值下的真值[4,7];符号 $\top$ 和 $\bot$ 表示真和假,代表公式在当前变量绑定的情况下是满足的还是违反的;函数 $eqStr$ 用来判定 2 个字符串是否相等.

根据定义 3,我们在真值计算时可以把公式 f_{ex1} 表示成 1 棵一致性计算树[4],如图 2 所示,计算出公式真值为 $\bot$.图 2 中元素 $e_{A,i}(e_{B,i},e_{C,i})$ 分别表示上下文 $c_A(c_B,c_C)$ 中第 i 个元素;数字 1~13 表示节点编号.

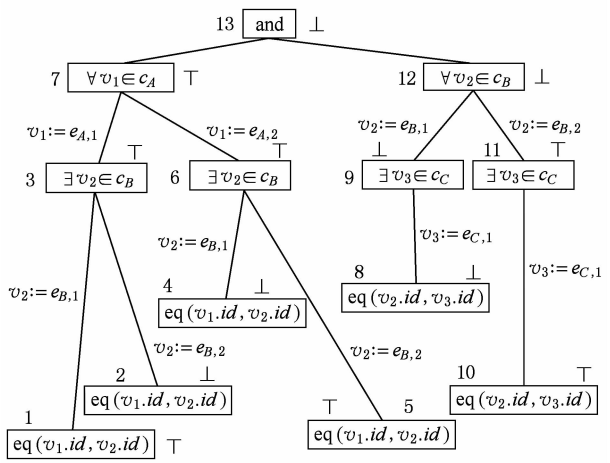

Fig. 2 Consistency computation tree.

图 2 一致性计算树

2 一致性错误的修复

2.1 修复行为

定义 4. 修复行为 ra.

$$ra::=add(e,c)\mid del(e,c)\mid chgEq(e,att,c,val)\mid chgDif(e,att,c).$$

如定义 4 所示,我们定义 4 种原子修复行为(repair action, ra):1)$add(e,c)$表示向上下文 c 中加入 1 个新元素 e.此时 e 的所有属性的值为 null,有待进一步分析确定其具体值.2)$del(e,c)$表示从上下文 c 中删除元素 e.3)$chgEq(e,att,c,val)$表示把上下文 c 中元素 e 的属性 att 的值修改为 val,val 不能为 null.4)$chgDif(e,att,c)$表示把使得上下文 c 中元素 e 的属性 att 的值修改为其他值.与原工作[7]相比,我们改进了的修复行为更加具体,执行更加简单.

2.2 修复用例和修复套件

定义 5. 修复用例 rc.

$rc::=\langle\{ra_1, ra_2, \cdots\}, weights\rangle$;

$rs::=\{rc_1, rc_2, \cdots\}$.

如定义 5 所示,我们定义在给定公式 f 和变量赋值 α 下生成的修复用例(repair case, rc)和修复套件(repair suite, rs). rc 包括 2 部分,即 ra 的集合和累加权重 $weights$(表示用户的偏好累加值). 修复行为 ra 的集合不为空. 1 个 rc 表示 1 种修复一致性错误的策略. rs 为 rc 的集合,代表各种可能策略,其中每个 rc 都是 1 种独立的选择.

2.3 辅助运算符

给定 2 个公式 f_1 和 f_2, rs_1 和 rs_2 分别是用来修复它们的 rs,其中 $rc_1 \in rs_1$ 并且 $rc_2 \in rs_2$. 定义$\oplus$运算:1) $rs_1 \oplus rs_2 ::= rs_1 \cup rs_2$,得到 1 个新的 rs,其中的 rc 只能修复 f_1 或 f_2 之一. 2) $rc_1 \oplus rc_2 ::= \langle actSet(rc_1) \cup actSet(rc_2), rc_1.weights + rc_2.weights\rangle$,得到 1 个新的 rc,能同时修复 f_1 且 f_2. 函数 $actSet$ 返回 rc 中的 ra 集合. 定义$\otimes$运算: $rs_1 \otimes rs_2 ::= \{rc_1 \oplus rc_2 \mid rc_1 \in rs_1 \wedge rc_2 \in rs_2\}$, if $rs_1 \neq \varnothing \wedge rs_2 \neq \varnothing$;否则,$\varnothing$. 运算得到 1 个新 rs,其中任意 1 个 rc 都能同时修复 f_1 且 f_2. rs 是$\{\varnothing\}$表示不能修复当前公式. 符号Σ和Π分别表示连续多个$\oplus$和$\otimes$运算.

2.4 修复生成函数

定义 6. 修复生成函数 $\mathcal{R}_{\mathcal{F}2\mathcal{T}}$, $\mathcal{R}_{\mathcal{T}2\mathcal{F}}$.

1) $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![\forall v \in c^w(f)]\!]_\alpha :=$
$\prod(\{\langle\{del(x_i, c)\}, w\rangle\} \oplus \mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![f]\!]_{bind((v,x_i),\alpha)} \mid x_i \in c \wedge \mathcal{T}[\![f]\!]_{bind((v,x_i),\alpha)} == \perp)$;

2) $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![(f_1)\ \text{and}\ (f_2)\ \text{and}\cdots\text{and}\ (f_n)]\!]_\alpha :=$
$\prod(\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![f_i]\!]_\alpha \mid \mathcal{T}[\![f_i]\!]_\alpha == \perp)$;

3) $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![\text{not}(f)]\!]_\alpha := \mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![f]\!]_\alpha$;

4) $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![\text{eq}(v_i.att_i, v_j.att_j)^w]\!]_\alpha :=$
$\{\langle\{chgEq(getE(v_m), att_m, getC(v_m), getE(v_n).att_n)\}, w\rangle \mid (m,n) \in \{(i,j),(j,i)\} \wedge getE(v_m).att_m \neq \text{null}\}$;

5) $\mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![\forall v \in c^w(f)]\!]_\alpha :=$
$(\{\langle\{add(z,c)\}, w\rangle\} \otimes \mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![f]\!]_{bind((v,z),\alpha)}) \oplus \sum(\mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![f]\!]_{bind((v,x_i),\alpha)} \mid x_i \in c)$;

6) $\mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![(f_1)\ \text{and}\ (f_2)\ \text{and}\cdots\text{and}\ (f_n)]\!]_\alpha :=$
$\sum(\mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![f_i]\!]_\alpha)$;

7) $\mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![\text{not}(f)]\!]_\alpha := \mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![f]\!]_\alpha$;

8) $\mathcal{R}_{\mathcal{T}2\mathcal{F}}[\![\text{eq}(v_i.att_i, v_j.att_j)^w]\!]_\alpha :=$
$\{\langle\{chgDif(getE(v_m), att_m, getC(v_m))\}, w\rangle \mid m \in \{i,j\}\}$.

如定义 6 所示,我们定义修复生成函数 $\mathcal{R}_{\mathcal{F}2\mathcal{T}}$和 $\mathcal{R}_{\mathcal{T}2\mathcal{F}}$. $\mathcal{R}_{\mathcal{F}2\mathcal{T}}$表示在当前赋值为 α 且 f 真值为$\perp$的情况下生成 rs,通过执行 rs 使 f 的真值为$\top$. 要使公式$\forall v \in c(f)$由$\perp$变$\top$,需要删除上下文 c 中所有使公式 f 为$\perp$的元素,或在公式下一层修复. 要使公式$(f_1)\ \text{and}\ (f_2)\ \text{and}\cdots\text{and}\ (f_n)$由$\perp$变$\top$,需要把那些为$\perp$的子公式全部修复. 要使公式 not($f$)由$\perp$变$\top$,只要使 f 的真值由$\top$变$\perp$. 要使公式 eq($v_i.att_i, v_j.att_j$)由$\perp$变$\top$,可以把第 1 个元素属性的值改为第 2 个元素属性的值或相反,当然目标值不为 null. $\mathcal{R}_{\mathcal{T}2\mathcal{F}}$表示在当前赋值为 α 且 f 真值为$\top$的情况下生成 rs,通过执行 rs 使 f 的真值为$\perp$. 要使公式$\forall v \in c(f)$由$\top$变$\perp$,可以向上下文 c 中添加 1 个新元素,其属性值暂时为 null,并继续分析确定属性的具体值或在公式下一层修复. 要使公式$(f_1)\ \text{and}\ (f_2)\ \text{and}\cdots\text{and}\ (f_n)$由$\top$变$\perp$,只需要使其中任意子公式真值变为$\perp$. 要使公式 not($f$)由$\top$变$\perp$,只要使 f 的真值由$\perp$变$\top$. 要使公式 eq($v_i.att_i, v_j.att_j$)由$\top$变$\perp$,改变其中任意 1 个属性的值即可. 函数 $getC$ 能获取公式中变量关联的上下文,例如公式 f_{ex1} 中 $getC(v_1)$将返回上下文 c_A.

我们选择公式$\forall$, and, not, eq 作为核心集;$\exists$, or, implies 作为非核心集,非核心集可用核心集表示[4]. 定义 6 是修复生成函数在核心集上的定义,在非核心集上的定义可由定义 6 推导出来,见原工作[7]. 如图 2, $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![f_{ex1}]\!]_\varnothing := \{\langle\{add(z, c_C), chgEq(z, id, c_C, 01)\}, 5\rangle, \langle\{del(e_{B,1}, c_B)\}, 2\rangle, \langle\{chgEq(e_{B,1}, id, c_B, 02)\}, 1\rangle, \langle\{chgEq(e_{C,1}, id, c_C, 01)\}, 1\rangle\}$. z 为元素$\{(id, \text{null})\}$,可见这种修复是完整的.

2.5 修复行为的执行

算法 1. 修复执行算法.

输入:公式 f、修复套件 rs;

输出:$success$.

① $success := \text{FALSE}$;

② $sort(rs)$;

③ $i := 1$;

④ WHILE $success = \text{FALSE}$ && $i \leqslant size(rs)$

⑤ DO 执行 rs 中第 i 个修复用例中的所有修复行为

⑥ IF 公式 f 的真值发生变化

⑦ THEN $success := \text{TRUE}$;

⑧ ELSE 辙销执行 rs 中第 i 个修复用例中的所有修复行为;

⑨ i++;

⑩ RETURN *success*.

算法 1 为修复执行算法,我们执行 rs 来修复公式 f. 若修复成功,则 *success* 为 TRUE,否则 *success* 为 FALSE. 行①用标签 *success* 来标记是否修复成功. 行②函数 *sort* 用于对 rs 中的 $rc_i \in rs$ 按照平均权重降序排列. 平均权重为累加权重与修复行为数量的比值,如〈{$add(z,c_C)$,$chgEq(z,id,c_C,01)$}, 5〉的平均权重为 2.5. 行④~⑨按照平均权重高到低顺序执行 rc_i,修复成功就退出循环,否则上下文恢复到修复前的状态执行下一个 rc,函数 *size* 获得 rs 中 rc 的数量. 行⑩返回是否修复成功的标志.

3 复合式修复 hybrid-fixing

我们用一致性计算树来评估当前的上下文是否满足公式. 但是由于元素共享导致公式内部的依赖,当修复公式的一部分时,却可能影响到公式其他部分的真值,最终导致整个修复失败. 如图 2 所示,如果执行 $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![f_{\text{ex1}}]\!]_{\varnothing}$ 中第 2 个 rc,删除元素 $e_{B,1}$,修复了节点 12;但元素 $e_{B,1}$ 是共享的,节点 3,7,13 的真值也跟着改变,使得修复失败. 因此,在原工作[7]中,我们提出对共享的元素加锁,在修复时禁止对有锁元素的操作,从而预防了负面效果产生. 在公式内部依赖比较简单时,这种方法能够取得比较好的效果,但这种机制有 2 个弱点:1)加锁的机制过于严格,可能把所有的 rc 都过滤掉;2)不能处理一致性约束内部存在复杂依赖关系这类情况.

1 个 rc 由 3 种基本操作(add,$delete$,$change$)组成. 通过细致分析我们发现 1 个 *repair case* 如果不满足正确性,要满足以下 3 个必要条件:1)这个 rc 中包含了对共享元素的操作;2)这些对共享元素的操作会改变公式其他部分的真值;3)对共享元素的操作会改变整个公式的真值. 因此,我们对公式进行分析,找出满足这 3 个条件的 rc,把它们过滤掉,使得剩下的 rc 是正确的.

我们把元素共享分为 2 类:纵向共享和横向共享. 纵向共享是指在 1 条公式中同 1 个上下文同时出现在父公式和子公式中. 例如公式 $\forall v_i \in c(\exists v_j \in c(\_f))$,其中 c 出现了 2 次,c 中的元素都是共享的. $\_f$ 表示某个公式,因为这个公式的内容对我们的分析没有影响,只是一个占位符,故不具体说明. 后文中含义相同. 纵向共享又可以按照定义时上下文前面量词的类型分为以下 3 种类型:1)纵向共享上下文前面的量词都是全称量词或等价于全称量词,生成对上下文 c 中的修复行为可能包含 3 种:$del(\_e,c)$,$chgTo(\_e,\_att,c,\_val)$,$chgDif(\_e,\_att,c)$. $\_e$ 代表上下文中某个具体元素,$\_att$ 代表元素的某个具体属性,$\_val$ 代表某个值. 后文中含义相同. 其中 $del(_e,c)$不会带来新一致性错误,其他 2 种都可能带来新一致性错误. 2)纵向共享上下文前面的量词都是存在量词或等价于存在量词时,生成对上下文 c 中的修复行为可能包含 3 种:$add(\_e,c)$,$chgTo(\_e,\_att,c,\_val)$,$chgDif(\_e,\_att,c)$. 其中 $add(\_e,c)$ 不会带来新一致性错误,对新添加元素的属性修改也不会带来新一致性错误,另外 2 种都可能带来新一致性错误. 3)纵向共享上下文前面的量词同时以 2 种量词或其等价形式出现,$add(\_e,c)$,$del(\_e,c)$,$chgTo(\_e,\_att,c,\_val)$,$chgDif(_e,_att,c)$都有可能带来新一致性错误.

横向共享可以按照元素共享带来的负面效果分为 2 类:1)元素共享的负面效果不会改变整个公式的真值;2)元素共享的负面效果可能会改变整个公式的真值. 分为以下 2 种情况:1)同一个上下文出现在 and(或等价于 and)的子公式中,如 $\forall v_i \in c(\_f)$ and $\exists v_j \in c(\_f)$. 分为以下 3 种情况:①共享的上下文前面出现的量词都是全称量词或其等价形式,$chgTo(_e,_att,c,_val)$和 $chgDif(_e,_att,c)$都可能改变整个公式的真值;②共享的上下文前面出现的量词都是存在量词或其等价形式,$chgTo(_e,_att,c,_val)$和 $chgDif(_e,_att,c)$都可能改变整个公式的真值;③共享的上下文前面同时存在 2 种量词或其等价形式,$add(\_e,c)$,$del(\_e,c)$,$chgTo(_e,_att,c,_val)$和 $chgDif(_e,_att,c)$都可能改变整个公式的真值. 2)全称量词的子公式中的元素共享. 如公式 $\forall v_i \in c_i(\exists v_j \in c_j(\_f))$,此时对 c_j 中共享元素的操作会改变整个公式的真值,$chgTo(_e,_att,c_j,_val)$和 $chgDif(_e,_att,c_j)$都有可能改变整个公式的真值.

算法 2. Filtering 算法.

输入:公式 f;

输出:S_{rejAdd},S_{rejDel},S_{rejChg}.

① LET $S_{\text{rejAdd}}:=\varnothing$,$S_{\text{rejDel}}:=\varnothing$,$S_{\text{rejChg}}:=\varnothing$;

② $S_{ctx}:=vSharedCtxs(f)$;

③ FOR EACH ctx in S_{ctx}

④ DO IF $checkQuantifier(ctx,f)=1$

⑤ THEN $S_{rejAdd}:=S_{rejAdd}\cup\{ctx\}$;
⑥ $S_{rejDel}:=S_{rejDel}\cup\{ctx\}$;
⑦ $S_{rejChg}:=S_{rejChg}\cup\{ctx\}$;
⑧ ELSE $S_{rejChg}:=S_{rejChg}\cup\{ctx\}$;
⑨ 初始化一个空栈 s;
⑩ $push(s,f)$;
⑪ WHILE(!$isEmpty(s)$)
⑫ $curF:=pop(s)$;
⑬ IF ($type(curF,f)=\forall$)
⑭ THEN $S_{rejChg}:=S_{rejChg}\cup getCtxs(subForm(curF,0))$;
⑮ $push(s,subForm(curF,0))$;
⑯ ELSE IF($type(curF,f)=and$)
⑰ THEN $S_{rejChg}:=S_{rejChg}\cup lSharedCtxs(subForms(curF))$;
⑱ FOR EACH ctx in $lSharedCtxs(subForms(curF))$
⑲ DO IF $checkQuantifier(ctx,f)=1$
⑳ THEN $S_{rejDel}:=S_{rejDel}\cup\{ctx\}$;
㉑ $S_{rejAdd}:=S_{rejAdd}\cup\{ctx\}$;
㉒ FOR EACH $subF$ in $subForms(curF)$
㉓ DO $push(subF)$;
㉔ ELSE IF($type(curF,f)=\exists$)
㉕ THEN $push(s,subForm(curF,0))$;
㉖ ELSE IF($type(curF,f)=$not)
㉗ THEN $push(s,subForm(curF,0))$;
㉘ ELSE IF($type(curF,f)=$or)
㉙ THEN FOR EACH $subF$ in $subForms(curF)$;
㉚ DO $push(subF)$;
㉛ IF($type(curF,f)=$implies)
㉜ THEN $push(subForm(curF,0))$;
㉝ $push(subForm(curF,1))$;
㉞ RETURN.

如算法 2 所示,我们把上面的思想表述成 Filtering 算法.给定公式 f,通过算法 2 我们找到 3 个上下文集合 S_{rejAdd},S_{rejDel},S_{rejChg}.根据 f 和当前上下文可生成 rs,对于任意 $rc\in rs$,如果存在 $add(\_e,c)\in rc\wedge c\in S_{rejAdd}$,或 $del(\_e,c)\in rc\wedge c\in S_{rejDel}$,或 $chgTo(\_e,\_att,c,\_val)\in rc\wedge c\in S_{rejChg}$,或 $chgDif(\_e,\_att,c)\in rc\wedge c\in S_{rejChg}$,则我们认为 rc 不满足正确性,要过滤掉.行①初始化 3 个上下文集合 S_{rejAdd},S_{rejDel},S_{rejChg},表示不能向上下文 $c(c\in S_{rejAdd})$ 中添加元素,不能删除 $c(c\in S_{rejDel})$ 中的元素,不能修改 $c(c\in S_{rejChg})$ 中的元素.行②~⑧是纵向分析过程.行②函数 $vSharedCtxs(f)$ 表示返回公式 f 中所有纵向共享的上下文集合.例如 $f_{ex2}:=\forall v_1\in c^1(\text{not}(\forall v_2\in c^2(\text{eq}(v_1,id,v_2,id)^1)))$,$vSharedCtxs(f_{ex2})$ 返回 $\{c\}$.函数 $checkQuantifier(ctx,f)$ 用来检测公式 f 中上下文 ctx 前面量词的情况,如果同时以 2 种量词或其等价形式出现,返回 1;否则返回 0.例如 $check\text{-}Quantifier(c,f_{ex2})$ 会返回 1,因为 $\forall v_2\in c$ 前出现了 not,等价于存在量词.行⑨~㉝是横向分析过程.行⑨~⑩初始化 1 个栈,并把当前公式 f 压入栈中.行⑪~㉝逐层分析公式 f.函数 $isEmpty(s)$ 用来判断当前栈 s 是否为空.行⑬~⑮,表示当前公式的类型等价于全称量词时,子公式中定义的任何上下文中的元素都是共享的,不能改变.函数 $type(curF,rootF)$ 能够判断当前公式 $curF$ 在整个公式 $rootF$ 中的具体类型.例如 $type(\forall v_2\in c,f_{ex2})$ 得到当前公式类型等价于 $\forall$.函数 $subForm(f,index)$ 表示返回公式 f 的第 $index$ 个子公式.函数 $getCtxs(f)$ 表示返回公式 f 中定义的上下文.行⑮把子公式入栈,待分析.行⑯~㉓,表示当前公式的类型等价于 and 时,子公式中所有定义的横向共享的上下文都不可以改变.函数 $subForms(f)$ 表示返回公式 f 的所有子公式.函数 $lSharedCtxs(f_1,f_2,\cdots,f_n)$ 表示返回公式 $f_1,f_2,\cdots,f_n$ 中共享的上下文集合.行⑱~㉓,分析每个横向共享的上下文,如果共享的上下文以 2 种量词或其等价形式出现,则既不能向这个上下文里添加元素,也不能删除其中的元素.行㉔~㉝,基本是子公式压入栈,待处理.例如,对于公式 f_{ex1} 通过算法 2 进行分析会得到 $S_{rejAdd}:=\{c_B\}$,$S_{rejDel}:=\{c_B\}$,$S_{rejChg}:=\{c_B,c_C\}$,过滤掉 $\mathcal{R}_{\mathcal{F}2\mathcal{T}}[\![f_{ex1}]\!]_\varnothing$ 中不满足正确性的 rc,得到 $rc\langle\{add(z,c_C),chgEq(z,id,c_C,01)\},5\rangle$,将留下元素 z 在添加之前并不是属于上下文 c_C,故可对其进行修改.

4 实验评估

我们设计了下面的实验来验证 hybrid-fixing 的实际效果.通过这个实验,我们需要回答 2 方面的研究问题:1)相对于原技术 CoSound-fixing,hybrid-fixing 是否能提高修复成功率?2)hybrid-fixing 的时间效率怎么样?

如图 3 所示为某车间示意图,车间里有 3 条生产线 L_1,L_2,L_3.L_1 和 L_2 是 2 条老生产线,要合作才能完成加工;L_3 是新生产线,可独立完成加工.每

件产品贴有唯一 id 的 RFID 标签.产品从 L_1 的 A 端运上生产线,完成第 1 道工序后由中转部分运到 L_2 上,完成第 2 道工序后由叉车运走.同理,产品从 E 端运上 L_3,完成加工后从 F 端运走.现在为了监控产品在生产线上的加工情况,在 A 端(L_1 和 L_2 的中转部分、D 端、E 端、F 端),各设置 1 个 RFID 阅读器 R_1(R_2,R_3,R_4,R_5)来读取经过的产品信息.每个 RFID 阅读器都可能漏读[4]经过它们的产品,同时 R_4(R_5)可能错读[4]到 A(D)端的产品.当前时间为 t,且 c_1,c_2,c_3,c_4,c_5 分别是 R_1,R_2,R_3,R_4,R_5 在时间段 $t-40$ s 到 $t-20$ s,$t-30$ s 到 $t-10$ s,$t-20$ s 到 t,$t-40$ s 到 $t-20$ s,$t-20$ s 到 t 读到的产品 id.我们设计了 6 条公式来检测上下文一致性错误:

$$f_1:(\forall v_1 \in c_1^{\,2}(\exists v_2 \in c_2^{\,4}(\mathrm{eq}(v_1.id, v_2.id)^1)))$$
$$\text{and}(\forall v_2 \in c_2^{\,2}(\exists v_3 \in c_3^{\,4}(\mathrm{eq}(v_2.id, v_3.id)^1)));$$

$$f_2:(\forall v_1 \in c_3^{\,2}(\exists v_2 \in c_2^{\,4}(\mathrm{eq}(v_3.id, v_2.id)^1)))$$
$$\text{and}(\forall v_2 \in c_2^{\,2}(\exists v_1 \in c_1^{\,4}(\mathrm{eq}(v_2.id, v_1.id)^1)));$$

$$f_3:(v_4 \in c_4^{\,2}(\exists v_5 \in c_5^{\,4}(\mathrm{eq}(v_4.id, v_5.id)^1)));$$

$$f_4:(v_5 \in c_5^{\,2}(\exists v_4 \in c_4^{\,4}(\mathrm{eq}(v_5.id, v_4.id)^1)));$$

$$f_5:(v_1 \in c_1^{\,2}(\mathrm{not}(\exists v_4 \in c_4^{\,4}(\mathrm{eq}(v_1.id, v_4.id)^1))));$$

$$f_6:(v_3 \in c_3^{\,2}(\mathrm{not}(\exists v_5 \in c_5^{\,4}(\mathrm{eq}(v_3.id, v_5.id)^1))));$$

其中,f_1 和 f_2 检测 L_1 上漏读的情况;f_3 和 f_4 检测 L_2 上漏读的情况;f_5 和 f_6 检测 R_4,R_5 的错读的情况.根据 RFID 的特性,漏读和错读的概率存在负相关性关系.图 4 和图 5 中的参数设置满足这种特性,m 和 c 分别表示漏读和错读的概率.

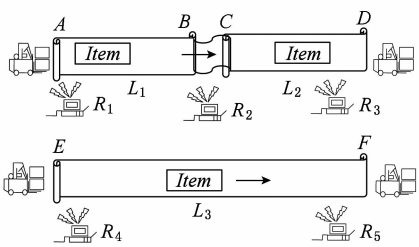

Fig. 3 Experimental scene.

图 3 实验场景

修复成功率:=(成功修复一致性错误总次数/一致性错误发生总次数)×100%.从图 4 可看出,hybrid-fixing 的修复成功率在不同的漏读和错读的

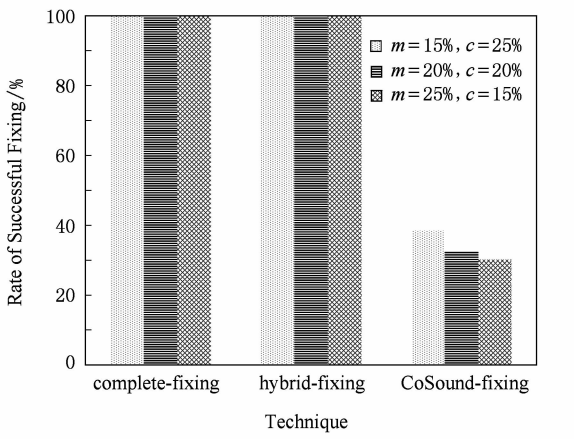

Fig. 4 The rate of successful fixing.

图 4 修复成功率

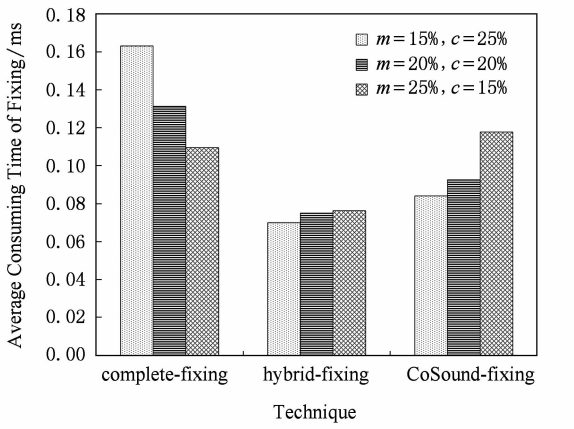

Fig. 5 The average consuming time of fixing.

图 5 平均修复时间消耗

情况下都比 CoSound-fixing[7]的修复成功率有很大提高,因为 CoSound-fixing 不能处理一致性约束内部存在复杂依赖关系这类情况,如 f_1 和 f_2.从图 5 可看出,hybrid-fixing 的平均修复时间是最低的,这很重要,因为在上下文一致性错误的修复过程中,时效性很关键.图 3 所示的场景中,3 种技术在不同的参数(漏读概率和错读概率)配置下,都独立运行了 24 h,产品进入生产线上的时间间隔符合泊松分布[12]($\lambda:=5$ s).我们硬件配置是一台普通 PC 机,配有 Intel CPU(core 2、主频 3 GHz),软件配置是 MS Windows 7 和 Oracle Java 1.7.0_09(虚拟机堆内存为 512 MB).

5 相关工作

软件实体的一致性管理是重要的研究问题.Nentwich 等人[10]针对分布式文件系统的一致性错误生成抽象修复,但这种抽象修复只能指出要修复

的位置,至于怎么修复要靠用户决定,正确性也只能由用户保证. Xiong 等人[13]基于谓词逻辑定义约束能够生成修复的区间,但需要用户来选择具体的值,同时也不支持一阶逻辑的表达形式. Demsky 等人[14]对数据结构进行修复,但不是完整的. Egyed[15]提出基于变量类型对 UML 模型进行修复的技术,既不是完整的,也不是自动的. Xiong 等人[16]提出 Beanbag 语言来支持对模型的一致性错误进行自动修复,它假设用户更新是正确的,但这种假设在上下文一致性错误中不成立.

Xu 等人[4,17-18]针对上下文一致性错误问题,系统地讨论了一致性错误的检测方法、提高效率的手段以及处理方案. 文献[3,5]提出用概率模型对上下文的可信度进行建模,并在合适的时间点删除可疑的上下文. 文献[6]讨论在不同的处理策略下,如何减小应用带来的负面影响.

另外,约束满足问题(CSP)也是一类传统的一致性管理问题,解决这种问题的方法一般是试图找到一组满足所有解,但这些方法往往是低效、不完整的且难以对大规模的空间进行搜索[16]. 相比之下,我们的方法生成的修复不需要大量的搜索.

6 总结和将来工作

在本文中,我们提出了能在一致性约束内部存在复杂依赖关系的情况下,修复上下文一致性错误的新技术——hybrid-fixing. 实验证明,这种技术在处理具有复杂依赖关系的上下文一致性错误方面,能够大大提高修复成功率,同时还能提高时间效率. 对于一致性约束之间存在复杂依赖关系这个问题,可根据具体需要对问题进行转化,把一致性约束合并起来进行处理.

同时,本文中也存在着不足和有待提高之处. 在分析约束内部依赖关系时,我们只考虑到了元素的粒度,而没有精确到元素的属性这个最细的粒度. 如果有具体需要,这方面还可进行优化.

参 考 文 献

[1] Henricksen K, Indulska J, Rakotonirainy A. Modeling Context Information in Pervasive Computing Systems [M]. Berlin: Springer, 2002: 79-117

[2] Xu C, Cheung S, Ma X, et al. Adam: Identifying defects in context-aware adaptation [J]. Journal of Systems and Software, 2012, 85(12): 2812-2828

[3] Chen C, Ye C, Jacobsen H. Hybrid context inconsistency resolution for context-aware services [C] //Proc of 2011 IEEE Int Conf on Pervasive Computing and Communications. Los Alamitos, CA: IEEE Computer Society, 2011: 10-19

[4] Xu C, Cheung S, Chan W, et al. Partial constraint checking for context consistency in pervasive computing [J]. ACM Trans on Software Engineering and Methodology, 2010, 19(3): 1-61

[5] Bu Y, Gu T, Tao X, et al. Managing quality of context in pervasive computing [C] //Proc of the 6th Int Conf on Quality Software. Washington: IEEE Computer Society, 2006: 193-200

[6] Xu C, Ma X, Cao C, et al. Minimizing the Side Effect of Context Inconsistency Resolution for Ubiquitous Computing [G] //LNICST 104: Mobile and Ubiquitous Systems: Computing, Networking, and Services. Berlin: Springer, 2012: 285-297

[7] Chen Xiaokang, Xu Chang, Jiang Lei. Automated fixing of inconsistent contexts [J]. Journal of Fronters of Computer Science & Technology, 2013, 7(4): 326-336 (in Chinese) (陈小康, 许畅, 江磊. 非一致上下文的自动修复技术[J]. 计算机科学与探索, 2013, 7(4): 326-336)

[8] Julien C, Roman G. EgoSpaces: Facilitating rapid development of context-aware mobile applications [J]. IEEE Trans on Software Engineering, 2006, 32(5): 281-298

[9] Murphy A, Picco G, Roman G. LIME: A coordination model and middleware supporting mobility of hosts and agents [J]. ACM Trans on Software Engineering and Methodology, 2006, 15(3): 279-328

[10] Nentwich C, Emmerich W, Finkelstein A. Consistency management with repair actions [C] //Proc of the 25th Int Conf on Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2003: 455-464

[11] Nentwich C, Capra L, Emmerich W, et al. Xlinkit: A consistency checking and smart link generation service [J]. ACM Trans on Internet Technology, 2002, 2(2): 151-185

[12] Ahrens J, Dieter U. Computer methods for sampling from gamma, beta, poisson and bionomial distributions [J]. Computing, 1974, 12(3): 223-246

[13] Xiong Y, Hubaux A, She S, et al. Generating range fixes for software configuration [C] //Proc of the 34th Int Conf on Software Engineering. Piscataway, NJ: IEEE, 2012: 58-68

[14] Demsky B, Rinard M. Goal-directed reasoning for specification-based data structure repair [J]. IEEE Trans on Software Engineering, 2006, 32(12): 931-951

[15] Egyed A. Fixing inconsistencies in UML design models [C] //Proc of the 29th Int Conf on Software Engineering. Los Alamitos, CA: IEEE Computer Society, 2007: 292-301

[16] Xiong Y, Hu Z, Zhao H, et al. Supporting automatic model inconsistency fixing [C] //Proc of the 7th Joint Meeting of the European Software Engineering Conf and the ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2009: 315-324

[17] Xu C, Cheung S. Inconsistency detection and resolution for context-aware middleware support [C] //Proc of the 10th Joint Meeting of European Software Engineering Conf and the 13th ACM SIGSOFT Int Symp on Foundations of Software Engineering. New York: ACM, 2005: 336-345

[18] Xu C, Cheung S, Chan W. Incremental consistency checking for pervasive context [C] //Proc of the 28th Int Conf on Software Engineering. New York: ACM, 2006: 292-301

Chen Xiaokang, born in 1989. MSc. His main research interests include context inconsistency resolution, software engineering and pervasive computing.

Xu Chang, born in 1977. PhD and associate professor. Member of China Computer Federation. His main research interests include software engineering, software testing and analysis, and pervasive computing.

Jiang Lei, born in 1988. MSc. His main research interests include context inconsistency resolution, software engineering and pervasive computing(ladd. cn@gmail. com).

2013 年《计算机研究与发展》高被引论文 TOP10

排名	论文信息
1	孟小峰，慈祥. 大数据管理:**概念、技术与挑战**[J]. 计算机研究与发展，2013，50(1)：146-169 Meng Xiaofeng and Ci Xiang . Big Data Management: Concepts, Techniques and Challenges [J]. Journal of Computer Research and Development, 2013, 50(1): 146-169
2	傅颖勋，罗圣美，舒继武. **安全云存储系统与关键技术综述**[J]. 计算机研究与发展，2013，50(1)：136-145 Fu Yingxun, Luo Shengmei, and Shu Jiwu. Survey of Secure Cloud Storage System and Key Technologies. Journal of Computer Research and Development, 2013, 50(1): 136-145
3	李建中，刘显敏. **大数据的一个重要方面:数据可用性**[J]. 计算机研究与发展，2013，50(6)：1147-1162 Li Jianzhong and Liu Xianmin. An Important Aspect of Big Data: Data Usability [J]. Journal of Computer Research and Development, 2013, 50(6): 1147-1162
4	刘大有,陈慧灵,齐红,杨博. **时空数据挖掘研究进展**[J]. , 2013, 50(2): 225-239 Liu Dayou, Chen Huiling, Qi Hong, and Yang Bo. Advances in Spatiotemporal Data Mining. Journal of Computer Research and Development, 2013, 50(2): 225-239
5	廖彬，于炯，孙华，年梅. **基于存储结构重配置的分布式存储系统节能算法**[J]. 计算机研究与发展，2013，50(1)：3-18 Liao Bin, Yu Jiong, Sun Hua, and Nian Mei. Energy-Efficient Algorithms for Distributed Storage System Based on Data Storage Structure Reconfiguration [J]. Journal of Computer Research and Development, 2013, 50(1): 3-18
6	贾冬艳，张付志. **基于双重邻居选取策略的协同过滤推荐算法**[J]. 计算机研究与发展，2013，50(5)：1076-1084 Jia Dongyan and Zhang Fuzhi. A Collaborative Filtering Recommendation Algorithm Based on Double Neighbor Choosing Strategy [J]. Journal of Computer Research and Development, 2013, 50(5): 1076-1084
7	武建佳，赵伟. **WInternet:从物网到物联网**[J]. 计算机研究与发展，2013，50(6)：1127-1134 Wu Jianjia and Zhao Wei. WInternet: From Net of Things to Internet of Things [J]. Journal of Computer Research and Development, 2013, 50(6): 1127-1134
8	余凯，贾磊，陈雨强，徐伟. **深度学习的昨天、今天和明天**[J]. 计算机研究与发展，2013，50(9)：1799-1804 Yu Kai, Jia Lei, Chen Yuqiang, and Xu Wei. Deep Learning: Yesterday, Today, and Tomorrow [J]. Journal of Computer Research and Development, 2013, 50(9): 1799-1804
9	熊金波，姚志强，马建峰，李凤华,李琦. **基于行为的结构化文档多级访问控制**[J]. 计算机研究与发展，2013，50(7)：1399-1408 Xiong Jinbo, Yao Zhiqiang, Ma Jianfeng, Li Fenghua, and Li Qi. Action-Based Multilevel Access Control for Structured Document [J]. Journal of Computer Research and Development, 2013, 50(7): 1399-1408
10	张振海，李士宁，李志刚，陈昊. **一类基于信息熵的多标签特征选择算法**[J]. 计算机研究与发展，2013，50(6)：1177-1184 Zhang Zhenhai, Li Shining, Li Zhigang, and Chen Hao. Multi-Label Feature Selection Algorithm Based on Information Entropy [J]. Journal of Computer Research and Development, 2013, 50(6): 1177-1184

数据来源:中国知网;统计日期:2014-12-18