基于序列的子问题相容性技术

陈德泉 张永刚 辛 颖 刘文壮
(吉林大学计算机科学与技术学院 长春 130012)
(吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

摘 要 研究约束求解中的相容性技术,针对目前已有相容性的传播级别,提出一种新的相容性概念——基于序列的子问题相容性(SSAC),并给出相应的实现算法。然后分析其时空、空间复杂性及正确性,证明SSAC化简不改变原约束满足问题的解集,同时证明SSAC的约束传播能力介于SAC和AC之间。通过对随机问题和composed问题的测试表明,所提算法的效率是已有算法SAC-SDS和SAC-3的2～3倍。

关键词 约束满足,相容性技术,SSAC
中图法分类号 TP181 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.7.007

Singleton Sub-problem Arc Consistency Based on Order

CHEN De-quan ZHANG Yong-gang XIN Ying LIU Wen-zhuang
(College of Computer Science and Technology, Jilin University, Changchun 130012, China)
(Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China)

Abstract In investigation of the consistency techniques, a new consistency concept SSAC was proposed based on the recently propagation levels. The algorithm and properties of SSAC were given successively in the paper. Thereafter we concluded that simplification of SSAC dosen't change the solution set of the original constraint satisfaction problem. Meanwhile the propagation ability of SSAC is between SAC and AC. Experiments results show that the performance of SSAC is 2 or 3 times than the existed algorithms.

Keywords Constraint satisfaction, Arc consistency technique, SSAC

1 引言

约束满足问题(Constraint Satisfaction Problem, CSP)是人工智能研究领域的一个重要问题。在求解过程中,相容性技术(consistency technique)所起的作用是十分巨大的,该技术通过移走不相容的值和尽可能早地发现将来会产生冲突的值,来降低搜索空间,加速求解过程[1]。目前相容性技术不仅用在经典约束问题的求解中,在量化约束满足问题[2]、分布式约束满足问题[3]的求解中都有广泛的应用。目前的相容性技术中最为高效的是弧相容技术(Arc Consistency, AC)[4],如算法AC3[5]、AC2001[6]等。因此,在BT算法中进行弧相容维护(Maintaining Arc Consistency, MAC)已经成为当今CSP求解技术中的一个主流、高效的求解策略。此外,在问题求解前进行一定的预处理操作,可以减小问题的搜索空间,从而提高求解效率。目前,许多高效的预处理技术已被提出,singleton弧相容算法(Singleton Arc Consistency, SAC)是时间和空间都比较理想的一种相容性技术。Debruyne等人于1997年提出了SAC-1算法[7],并证明SAC的传播能力强于AC。2004年－2005年,R. Barták和C. Bessiere相继提出基于AC4和AC2001的SAC算法[8,9],这一系列算法都是从执行效率的角度对原有算法进行改进。2005年C. Lecoutre提出一种基于深度优先的贪婪SAC-3算法[10]。2008年,C. Bessiere提出双向singleton弧相容技术(Bidirectional Singleton Arc Consistency, BiSAC)[11],并证明BiSAC的传播能力强于SAC。针对上述算法的不足,我们提出了基于最先失败原则的约束传播算法FFP-AC[12]和基于预处理技术的约束求解算法MPAC与MPAC*[13]。此外通过研究现有的相容性技术,我们又提出完全独立相容性概念(Entirety Singleton Consistency, ESC)以及改进的SAC算法SAC-ESC[14],并针对现有相容性技术的单值传播特点,提出多值传播理论以及多值的弧相容定理,而后给出多值传播的相容性算法SAC-MP[15],改进了BiSAC算法,提出更为高效的算法BiSAC-2[16]。

约束满足问题P是由一个变量的集合(通常表示成$var(P)$)、每个变量的论域$dom(X)$以及关于这些变量的约束集合$C(P)$组成。若指明具体问题P的论域,其可用dom_P表示。特殊地,如果对于约束集合中的任何一个约束c,其所包

到稿日期:2014-06-12 返修日期:2014-09-30 本文受国家自然科学基金面上项目:基于自适应约束传播的约束求解方法研究(61170314),国家自然科学基金面上项目:结合自主搜索机制的约束求解方法研究(61373052)资助。
陈德泉(1986－),男,硕士生,主要研究方向为约束程序,E-mail:565649439@qq.com;张永刚(1975－),男,博士,副教授,主要研究方向为约束程序,E-mail:ygzhang@163.com(通信作者);辛 颖(1990－),女,硕士生,主要研究方向为约束程序,E-mail:292793238@qq.com;刘文壮(1988－),男,硕士生,主要研究方向为约束程序研究,E-mail:707122893@qq.com。

含的变量个数都是2，则称该问题为二元约束满足问题。其解 ρ 是为变量集中的每个变量从有限论域中寻找一个值，使得所有的约束都被满足，用 $Val_X(\rho)$ 表示变量 X 在解 ρ 中的取值，一般用 $sol(P)$ 表示问题 P 解的集合。通常说问题 P 与 Q 等价，当且仅当 $sol(P)=sol(Q)$。为了提高效率，通常在求解前和求解过程中积极地使用约束来删除变量中一些不参与解的值，这个过程称为约束传播、域过滤或者相容性技术。而这一技术最为核心的是弧相容技术，其它众多相容性技术大多是基于此展开的。对于二元 CSP 的约束图中的某一个弧 (X,Y)，称它是弧相容的，当且仅当对于 X 论域中能满足 X 上一元约束的每一个值 a，都在 Y 的论域中存在一个值 b，使得 b 满足 Y 上一元约束，并且 (a,b) 满足 X 和 Y 上的二元约束 $C(X,Y)$。一个 CSP 是弧相容的，当且仅当它的约束图中的每一条弧都是弧相容的。问题 P 在进行弧相容检查时，如得到某一变量的论域为空，则此问题 P 无解，此时记为：$AC(P)=\perp$。文献[17]给出了其详细概念和相关术语。

在弧相容技术的基础上，文献[7-11]引入 singleton 弧相容的概念。问题 P 是 singleton 弧相容的，当且仅当对于 $\forall X \in var(P)$，$\forall a \in dom(X)$，$P|_{X=a}$ 是弧相容的。其中 $P|_{X=a}$ 是将原问题 P 中 X 的论域 $dom(X)$ 用单独的 $\{a\}$ 进行替换。此后满足 singleton 弧相容的一系列算法也相应地被提出。

SAC-SDS[9]是一个基于 AC-2001 的算法，它和 AC-2001 都是以牺牲最优时间复杂度来节省空间，但也维持一些必要的结构来避免重复的检查操作，这样能在算法的时间和空间上都达到一个比较理想的情况。其思想是为每一个 (X,a) 存储一个局部约束传播等待队列 Q 和一个弧相容的论域。利用这一结构，能知道当某个值被移走时，哪些论域需要做约束传播处理，同时它也能记录最后一次传播过程的结果。

SAC-3[10]是一种基于深度优先、贪婪搜索模式下的算法。它具有低额的空间代价和相对较优的时间复杂度，通过利用一种贪婪的搜索和在搜索过程中递增的弧相容维护来避免重复的约束传播过程。如果遇到一个死结点，则重新搜索其它值对的组合。并用搜索过程中的冲突信息指导搜索，在预处理阶段中迫使 SAC-3 算法可能找到幸运解。

本文在原有工作的基础上进行研究与总结，提出一种与以往传播级别不同的相容性技术——基于序列的子问题相容性技术(Singleton Subproblem Arc Consistency，SSAC)，在给出相应算法的同时，分析其时间、空间复杂性并证明其正确性。而后对 SSAC 的若干性质加以证明，并对其约束传播能力进行分析，得出重要结论：SAC＞SSAC＞AC(＞代表传播能力强于)。对随机问题和 composed 问题进行了测试，实验数据表明：新提出的算法的效率是已有算法 SAC-SDS 和 SAC-3 的 2～3 倍。

2 基于序列的子问题相容性

目前相容性技术在约束求解中应用十分广泛，不仅在求解经典约束满足问题中有重要作用，在量化约束满足问题、分布式约束求解中都有相应的相容性概念和算法用于加速求解效率。针对目前流行的 SAC 和 AC 技术，研究它们传播级别的不同，提出一种新级别的相容性技术，该技术与 SAC 相比具有更为高效的移除冗余值的能力。本节首先给出基于序列的子问题概念和算法，在下一节将对其时空复杂性进行分析，并证明若干性质。

定义 1 一个约束满足问题 $P=(V,D,C)$，对于给定的序列 $d=(X_1,X_2,\cdots,X_n)$，P 是子问题 singleton 弧相容的，当且仅当 P 是弧相容的，并且对于任意 (X_i,a)，$P_i|_{X_i=a}$ 是弧相容的。其中 $P_i=(V_i,D_i,C_i)$，$V_i=\{X_j|X_j\leqslant X_i\}$，$D_i=\{D_{X_j}|X_j\in V_i\}$，$C_i=\{c|c\in C\wedge\forall X_j\in scope(c),X_j\in V_i\}$。

给出一个例子说明定义 1，如图 1、图 2 所示，给定序列：$d=(X_1,X_2,X_3,X_4)$。图 1 表示的约束满足问题不是 SSAC 相容的，因为对于变量 X_3 中 2 而言，$AC(P_3|_{X_3=2})=\perp$(P_3 见定义 1)，即 P_3 实例化以后不是弧相容的。而图 2 的问题则是弧相容的，并且所有子问题 P_i 实例化以后都是弧相容的，故此问题是 SSAC 相容的。

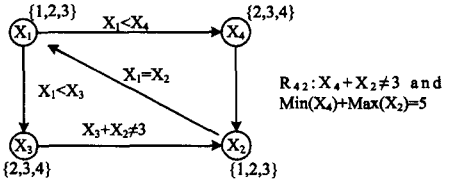

图 1 不满足 SSAC 的约束满足问题

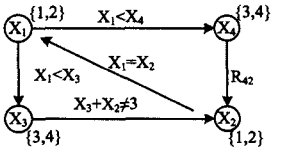

图 2 满足 SSAC 的约束满足问题

下面给出一种实现基于序列的子问题相容性的算法 SSAC。代码见 Algorithm SSAC。首先算法 SSAC 用弧相容算法对问题 P 进行化简(见算法第 1 行)，然后按照定义 1 构建子问题 P_i，进行弧相容检查，并将移走的值对加入到队列 Q 中。最后对原问题 P 进行弧相容检查，同样将移出的值对加入到 Q 中(算法 2－11 行)。通过函数 updateSubproblems 对 Q 中的元素进行更新传播等待队列 *Pendinglist*(此队列中的元素表示相应的子问题的论域已经发生变化，可能导致此问题已经不是弧相容的，为此需要进行相应的判断，故将其加入到等待队列 *Pendinglist*)。从队列中取出元素进行弧相容检查，对那些 $AC(P_i|_{X_i=a})=\perp$ 的元素，将其从原问题中删除，并加入到队列 Q 中，以便下一次更新等待队列 *Pendinglist*。算法第 12－24 行实现上述传播过程。对于实现定义 1 的算法，若仅用算法 SSAC 的第 1－8 行是不够的。因为，如果序列子问题 P_i 判断完毕以后，可能存在某些值已经从原始问题中移出，这样就导致该问题不是弧相容的。因此需要下面的约束进行传播处理。

```
Algorithm SSAC
1. P=AC(P);
2. for each Xi in V do
3.    initialize Pi by the definition 1;
4.    for each a in D(Xi) do
5.       if AC(Pi|Xi=a)=⊥ then
6.          remove the value a from the variable Xi;
7.          if D(Xi)=NULL then return false;
8.          Q=Q+{(Xi,a)};
9. P'=AC(P,{Xi});
10. Q=Q+P\P';
11. P=P'
12. updateSubproblems(Q);
13. while Pendinglist ≠ NULL do
```

```
14.   pop (Xi,a) from Pendinglist;
15.   if a in D(Xi) then
16.     if AC(Pi|Xi=a)=⊥ then
17.       remove the value a from the variable Xi;
18.       if D(Xi)=NULL then return false;
19.       P'=AC(P,{Xi});
20.       Q=P \ P';
21.       P=P';
22.       if Q=⊥ then return false;
23.           else Q=Q+{(Xi,a)}
24.       updateSubproblems(Q);
25. return true;

updateSubproblems(Q)
1. for each (Xi,a) in Q do
2.   foreach Pj such that (Xi,a) in Pj do
3.     Pj=Pj\{(Xi,a)};
4.     for each (Xj,b) in Pj do
5.       Pendinglist=Pendinglist+{(Xj,b)};
```

本文仍用图 1、图 2 的例子说明算法 SSAC 的执行过程。利用算法 SSAC 对图 1 的问题进行化简，化简后的问题如图 2 所示。首先构建子问题 P_1，P_2，P_3，P_4，进行弧相容检查，得到 $AC(P_3|_{X_3=2})=\perp$，$AC(P_4|_{X_4=2})=\perp$，为此将 $(X_3,2)$ 和 $(X_4,2)$ 从问题 P 中移走，并加入到 Q 中，之后对 P 进行弧相容检查，移走 $(X_1,3)$ 和 $(X_2,3)$，并加入 Q 中，再调用 updateSubproblems 更新等待队列进行约束传播直至队列为空。

3 复杂性分析和性质证明

根据 SAC 算法的正确性和 SSAC 的定义，容易证明以下性质。在证明下述定理的过程中用到了 AC 算法的正确性和唯一性[4]。

定理 1 算法 SSAC 的时间复杂度是 $O(end^3)$，空间复杂度是 $O(n^2d^2)$。

证明：算法 SSAC 的时间复杂度可以分成两部分，第 1—11 行和第 12—25 行。首先第 1—11 行的时间复杂度为 $n\times d\times O(ed^2)$（其中 $O(ed^2)$ 是弧相容算法的时间复杂度）。每一个值对 (X_i,a) 只能从 P 中移走一次，因此 while 循环至多执行 $n\times d$ 次。此时时间复杂度是 $n\times d\times O(ed^2)$，即 SSAC 的时间复杂度是 $O(end^3)$。算法所用额外空间包括子问题、队列 Q 和 $PendlingList$。两个队列空间复杂度为 $O(nd)$，子问题所用空间是：

$$d\times d+2d\times d+\cdots+(n)\times d\times d=\frac{n(n+1)}{2}\times d\times d$$

即算法的空间复杂度为 $O(n^2d^2)$。

定理 2 约束满足问题 P，则有 $sol(P)=sol(SSAC(P))$。

证明：证明上述性质仅需证明 SSAC 算法移走的任何值都不参与解即可。设对于任意被移走的值 a_j（$a_j\in dom(X_j)$），必然存在 $AC(P_j|_{X_j=a})=\perp$（第 5 行和第 16 行）或者从 19 行中移走，因为 P_j 是 P 的子问题，由弧相容算法的一致性和正确性可以得出 a_j（$a_j\in dom(X_j)$）不参与 P 的解。故此性质得证。

定理 3 算法 SSAC 是正确的。

证明：设对于任意被算法 SSAC 移走的值 a_j（$a_j\in dom(X_j)$），必然存在 $AC(P_j|_{X_j=a})=\perp$（第 5 行和第 16 行）或者从 19 行中移走，由定义 1 可知：此值（a_j）不满足 SSAC 的定义，应被移除。设存在一个不满足定义 1 的值 a_j（$a_j\in dom(X_j)$）没有被移走。当算法 SSAC 终止时，从算法的执行过程可以看出一定存在 $AC(P_j|_{X_j=a})\neq\perp$ 且 P 是弧相容的，满足定义 1，矛盾。故不满足 SSAC 定义的值已经全部移走。综上，算法 SSAC 是正确的。

下面对 SSAC 技术的传播级别进行分析，可以得出以下结论。

定理 4 SAC 强于 SSAC，并且 SSAC 强于 AC。

证明：利用弧相容算法的正确性和一致性，可得：若 $P|_{X_i}=a$ 是弧相容的，则 $P_i|_{X_i=a}$ 是弧相容的（P_i 见定义 1）；若 $AC(P_i|_{X_i=a})=\perp$，则 $AC(P|_{X_i=a})=\perp$。由 SAC 的定义和 SSAC 的定义可以得出：SSAC 移走值的集合是 SAC 移走值的集合的子集，故 SAC 强于 SSAC，再由定义 1 容易得出 SSAC 强于 AC。

从而可以得出关系：SAC>SSAC>AC。

4 实验结果

本文使用随机的约束满足问题和 composed 问题（http://cpai.ucc.ie/05/Benchmarks.html）进行测试。为了比较算法效率，用移除一个冗余值所需要的时间（cpu/#rmvs）和所做的约束检查次数（#chks/#rmvs）来衡量算法性能，并与目前流行的 SAC-SDS 和 SAC-3 作对比。使用 C++语言在“明月”[12-16]平台上编写程序，测试环境为：硬件 DELL Intel PIV 3.0GHz CPU/512MB RAM；软件 Windows XP Professional SP2/Visual C++6.0，其中将每个用例测试 10 次取平均值作为最后结果。

(1)随机约束满足用例

随机问题产生器随机地生成二元约束满足问题（http://www.lirmm.fr/~bessiere/generator.html），问题的难度和规模可以根据参数的设置来控制。由于问题具有随机性，因此绝大部分通用约束求解算法和通用启发式策略都采用这种测试方法[18]。二元随机约束满足问题有 4 个描述参数（n，m，pc，pt），其中 n 代表变量的个数，m 代表变量论域的大小（这里假设所有变量的论域大小相同），pc 代表此约束满足问题的密度，即约束图中边的个数与 $n*(n-1)/2$（完全图中边的个数）的比值，pt 为每个二元约束的松紧度。从图 3—图 5 可以清晰地看出，当问题规模为 $n=100$，$m=10$，$pt=0.40$ 时，pc 从 0.060 变化到 0.123 时（间隔为 0.001），本文算法 SSAC 移走每个值所需的时间仅是 SAC-3 和 SAC-SDS 的 1/3～1/2。从图 6、图 7 可以得出，当问题规模不变，$pt=0.50$ 时，pc 从 0.040 变化到 0.079 时（间隔为 0.001）。本文算法移走每个值所用时间大约是 SAC-3 和 SAC-SDS 的 1/3，甚至更少。对于移走每个值所做的约束检查次数，从图中可看出提出的算法有显著优势。

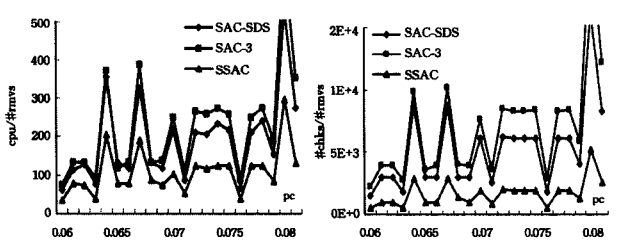

图 3 问题（100，10，pc，0.40），pc 在区间 [0.060，0.081] 变化时，cpu/#rmvs 和 #chks/#rmvs 的对比

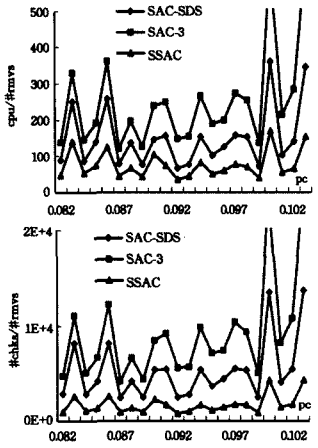

图 4　问题〈100,10,pc,0.40〉,pc 在区间 [0.082,0.103]变化时，cpu/＃rmvs 和＃chks/＃rmvs 的对比图

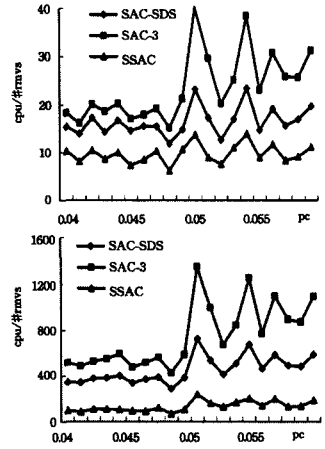

图 6　问题〈100,10,pc,0.50〉,pc 在区间 [0.040,0.059]变化时，cpu/＃rmvs 和＃chks/＃rmvs 的对比图

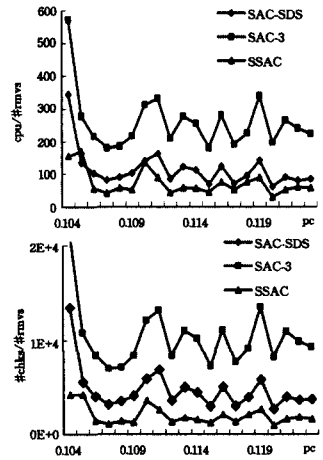

图 5　问题〈100,10,pc,0.40〉,pc 在区间 [0.104,0.123]变化时，cpu/＃rmvs 和＃chks/＃rmvs 的对比图

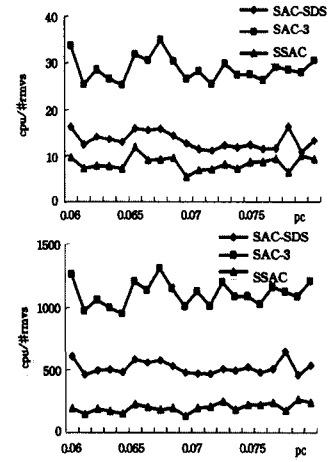

图 7　问题〈100,10,pc,0.50〉,pc 在区间 [0.060,0.079]变化时，cpu/＃rmvs 和＃chks/＃rmvs 的对比图

表 1　composed 问题的实验结果

instances	algorithm	cpu(ms)	＃chks(K)	＃rm	cpu/＃rm	＃chks/＃rm	Wipe
composed-25-1-2	SAC-SDS	31	1241	74	0.42	16778	yes
	SAC-3	16	423	70	0.23	6137	yes
	SSAC	16	212	57	0.28	3734	yes
composed-25-1-25	SAC-SDS	31	1774	82	0.38	21634	yes
	SAC-3	15	512	70	0.21	7432	yes
	SSAC	0	249	72	0	3464	yes
composed-25-1-40	SAC-SDS	63	2546	82	0.76	23578	yes
	SAC-3	15	558	60	0.25	9473	yes
	SSAC	16	261	80	0.2	3269	yes
composed-75-1-25	SAC-SDS	157	7673	84	1.86	91346	yes
	SAC-3	31	1660	61	0.51	27672	yes
	SSAC	32	699	72	0.44	9712	yes

(2)composed 问题实例

从表 1 中可以看出本文提出的算法 SSAC 的效率是 SAC-SDS 和 SAC-3 的 2～3 倍左右。由于 composed 问题不是 SAC 也不是 SSAC 的，且算法执行的过程不同，因此算法 SAC-SDS 和算法 SAC-3 终止时(发现空论域)，所移走冗余值的数量是不同的。

结束语　本文针对目前约束求解的热点——相容性技术进行研究与总结，提出一种全新的相容性技术——基于序列的子问题相容性算法，在给出该算法的时空复杂度之后，对其若干性质加以证明。并得出重要结论：SAC＞SSAC＞AC。实验结果表明提出的算法具有极为明显的性能优势。目前该技术仅用于经典约束满足问题，相信经过扩展可用于求解非经典约束满足问题，并发挥重要作用。

参 考 文 献

[1] Bartak R. Theory and Practice of Constraint Propagation[C]// Proceedings of the 3rd Workshop on Constraint Programming in Decision and Control. Gliwice, Poland, 2001:7-14

[2] Gent I P, Nightingale P, Rowley A, et al. Solving quantified constraint satisfaction problems[J]. Artificial Intelligence, 2008, 172(6/7):738-771

(下转第 37 页)

prove flow-insensitive pointer analysis[J]. ACM SIGPLAN Notices. ACM, 1998, 33(5): 97-105

[10] Yu H, Xue J, Huo W, et al. Level by level: making flow-and context-sensitive pointer analysis scalable for millions of lines of code[C]// Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization. ACM, 2010: 218-229

[11] Whaley J, Lam M S. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams[J]. ACM SIGPLAN Notices. ACM, 2004, 39(6): 131-144

[12] King J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394

[13] Hampapuram H, Yang Y, Das M. Symbolic path simulation in path-sensitive dataflow analysis[J]. ACM SIGSOFT Software Engineering Notes. ACM, 2005, 31(1): 52-58

[14] Xu Z, Kremenek T, Zhang J. A memory model for static analysis of C programs[M]// Leveraging Applications of Formal Methods, Verification, and Validation. Springer Berlin Heidelberg, 2010: 535-548

[15] Canet G, Cuoq P, Monate B. A Value Analysis for C Programs [C]// 2013 IEEE 13th International Working Conference on Source CodeAnalysis and Manipulation (SCAM). IEEE, 2009: 123-124

[16] Miné A. Weakly relational numerical abstract domains[D]. Paris: École Normale Supérreure, 2004

[17] Necula G C, McPeak S, Rahul S P, et al. CIL: Intermediate language and tools for analysis and transformation of C programs [C]// Compiler Construction. Springer Berlin Heidelberg, 2002: 213-228

[18] Chen L, Li R, Wu X, et al. Static analysis of list-manipulating programs via bit-vectors and numerical abstractions[C]// Proceedings of the 28th Annual ACM Symposium on Applied Computing. ACM, 2013: 1204-1210

[19] 李梦君，李舟军，陈火旺. 基于抽象解释理论的程序验证技术[J]. 软件学报，2008，19(1)：17-26
Li Meng-jun, Li Zhou-jun, Chen Huo-wang. Program verification techniques based on the abstract interpretation theory[J]. Journal of Software, 2008, 19(1): 17-26

[20] SNUReal-Time Benchmark Suite[OL]. http://archi.snu.ac.kr/realtime/benchmark

[21] von Hanxleden R, Holsti N, Lisper B, et al. WCET tool challenge 2011: report[OL]. http://hdl.handle.net/22909/10354

（上接第 31 页）

[3] Yokoo M, Durfee E H, Ishida T, et al. Distributed constraint satisfaction for formalizing distributed problem solving[C]// Proceedings of the 12th IEEE International Conference on Distributed Computing System. Yokohama, Japan: IEEE Computer Society Press, 1992: 614-621

[4] Bessiere C, Regin J C, Yap R H C, et al. An optimal coarse-grained Arc Consistency algorithm[J]. Artificial Intelligence, 2005, 165(2): 165-185

[5] Mackworth A K. Consistency in networks of relations[J]. Artificial Intelligence, 1977, 8(1): 99-118

[6] Bessiere C, Regin J C. Refining the basic constraint propagation algorithm[C]// Proceedings of the 17th International Joint Conference on Artificial Intelligence. Seattle, USA: Morgan Kaufmann, 2001: 309-315

[7] Deburyne R, Bessiere C. Some practical filtering techniques for the constraint satisfaction problem[C]// Proceedings of the 15th International Joint Conference on Artificial Intelligence. Nagoya, Japan: Morgan Kaufmann, 1997: 412-417

[8] Bartak R, Erben R. A new algorithm for singleton arc consistency[C]// Proceedings of FLAIRS Conference. Florida, USA: AAAI Press, 2004: 257-262

[9] Bessiere C, Debruyne R. Optimal and suboptimal singleton arc consistency algorithms[C]// Proceedings of the 19th International Joint Conference on Artificial Intelligence. Edinburgh, UK: Professional Book Center, 2005: 54-59

[10] Lecoutre C, Cardon S. A greedy approach to establish singleton arc consistency[C]// Proceedings of the 19th International Joint Conference on Artificial Intelligence. Edinburgh, UK: Professional Book Center, 2005: 199-204

[11] Bessiere C, Bessiere R. Theoretical analysis of singleton arc consistency and its extensions[J]. Artificial Intelligence, 2008: 172 (1): 29-41

[12] 孙吉贵，朱兴军，张永刚，等. 最先失败原则的约束传播算法[J]. 小型微型计算机系统，2008，29(4)：678-681
Sun J G, Zhu X J, Zhang Y G. Constraint Propagation on Fail First Principle[J]. Mini-Micro Systems, 2008, 29(4): 678-681

[13] 孙吉贵，朱兴军，张永刚，等. 一种基于预处理技术的约束满足问题求解算法[J]. 计算机学报，2008，31(6)：919-926
Sun J G, Zhu X J, Zhang Y G. An Approach of Solving Constraint Satisfaction Problem Based on Preprocessing[J]. Chinese Journal of Computers, 2008, 31(6): 919-926

[14] 朱兴军，孙吉贵，张永刚，等. 一种新的基于完全独立相容性的预处理技术[J]. 自动化学报，2009，35(1)：71-76
Zhu X J, Sun J G, Zhang Y G. A new preprocessing technique based on entirety singleton consistency[J]. Acta Automatica Sinica, 2009, 35(1): 71-76

[15] 朱兴军，张永刚，李莹，等. 多值传播的相容性技术[J]. 自动化学报，2009，35(10)：1296-1301
Zhu X J, Zhang Y G, Li Y, et al. Consistency Technique of Multi-Value Propagation[J]. Acta Automatica Sinica, 2009, 35 (10): 1296-1301

[16] 刘春晖，朱兴军，孙吉贵. 一种改进的双向 singleton 弧相容算法[J]. 吉林大学学报(工学版)，2008，3(28)：666-670
Liu C H, Zhu X J, Sun J G. Improved bidirectional singleton arc consistency algorithm[J]. Journal of Jilin University Engineering and Technology Edition, 2008, 3(28): 666-670

[17] Tsang E P K. Foundations of Constraint Satisfaction[M]. Academic Press, London and San Diego, 1993

[18] Gent I P, Macintyre E, Prosser P, et al. Random constraint satisfaction: Flaws and structure[J]. Journal of Constraints, 2001, 6 (4): 345-372