软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(6):2115−2126 [doi: 10.13328/j.cnki.jos.006573]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

机械化验证一个高效的迭代数据流求解算法*

江 南[1], 汪吕蒙[2], 张晓瞳[2], 何炎祥[2]

[1](湖北工业大学 计算机学院, 湖北 武汉 430068)

[2](武汉大学 计算机学院, 湖北 武汉 430072)

通信作者: 何炎祥, E-mail: yxhe@whu.edu.cn

摘 要: 迭代计算数据流等式的解, 是数据流分析的常用方法. 计算支配节点, 从而识别自然循环, 是许多现代编译器优化分析的重要组成部分. 机械化验证高效的求解支配节点的算法通常是获得一个实际的“验证编译器”不可或缺的一部分. 为了形式化证明一个高效的迭代求解严格支配节点的算法(CHK), 首先建立了值域是逆序列表集合的半格结构, 逆序列表中的元素是控制流图中节点的逆后序遍历次序, 并证明了它是一个半格, 其偏序满足上升链条件. 然后使用半格结构, 实现了一个基于工作表的Kildall迭代算法, 计算严格支配节点. 接下来, 首先给出了控制流图中支配节点的定义性规范和相关性质定理, 然后构造并证明了迭代求解算法所满足的重要性质. 利用这些性质定理, 相对于定义性规范, 证明了该迭代求解算法的正确性和完备性. 最后进行总结, 并讨论未来工作. 整个形式化开发使用的是定理证明助手 Isabelle/HOL.

关键词: 机械化验证; 高效迭代算法; 支配节点

中图法分类号: TP311

中文引用格式: 江南, 汪吕蒙, 张晓瞳, 何炎祥. 机械化验证一个高效的迭代数据流求解算法. 软件学报, 2022, 33(6): 2115−2126. http://www.jos.org.cn/1000-9825/6573.htm

英文引用格式: Jiang N, Wang LM, Zhang XT, He YX. Mechanized Verification of Efficient Iterative Data-flow Algorithm. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2115−2126 (in Chinese). http://www.jos.org.cn/1000-9825/6573.htm

Mechanized Verification of Efficient Iterative Data-flow Algorithm

JIANG Nan[1], WANG Lü-Meng[2], ZHANG Xiao-Tong[2], HE Yan-Xiang[2]

[1](School of Computer Science, Hubei University of Technology, Wuhan 430068, China)

[2](School of Computer Science, Wuhan University, Wuhan 430072, China)

Abstract: The common way of performing data-flow analysis of programs is by solving the data-flow equations using an iterative algorithm. Finding dominators, thus identifying natural loops, is central to numerous modern compiler optimization. Basically, mechanized verification of an efficient algorithm of computing dominators is intergral part of a verified compiler. In order to prove an efficient algorithm of computing strict dominators formally, first, a semilattice structure whose domain is a set of all descending lists in which nodes in a control flow graph are represented by its reverse post order (rPO) number is constructed and proved to be a semilattice whose ordering satisfies the ascending chain condition. Then, using the semilattice structure, a worklist-based Kildall's algorithm that computes strict dominators is implemented. Next, a specification of dominators on a control flow graph is defined; key properties of the specification and the iterative algorithm are established, and the correctness and completeness of the algorithm are proved with respect to the definitional specification. Finally, the work is summarized and future research is presented. The whole development is carried out in Isabelle/HOL.

Key words: mechanized verification; efficient iterative algorithm; dominators

* 基金项目: 国家自然科学基金(61972293); 国家留学基金委地方合作项目(201808420414)
本文由“定理证明理论与应用”专题特约编辑曹钦翔副教授、詹博华副研究员、赵永望教授推荐.
收稿时间: 2021-08-16; 修改时间: 2021-10-14; 采用时间: 2022-01-04; jos 在线出版时间: 2022-01-28

1 高效的迭代数据流求解算法

迭代计算数据流等式的解, 是数据流分析的常用方法. 计算支配节点(dominator), 从而识别自然循环, 是许多现代编译器优化分析的重要组成部分. 支配节点定义为: 如果从入口节点到节点 j 的所有路径都包括节点 i, 则称 i 支配(dominate)j. 按照该定义, 控制流图中的任意节点都支配其自身. 如果 i 支配 j, 且 i 不等于 j, 则称 i 严格支配(strict dominate)j. 除入口节点以外, 在节点 j 的所有严格支配节点中, 存在一个节点 i', j 的所有其他严格支配节点都支配 i', 则称 i'是 j 的直接支配节点(immediate dominator).

最早使用迭代方法计算支配节点的是 Allen 和 Cocke[1]. 按照该方法, 除入口节点外, 计算严格支配节点即求解等式:

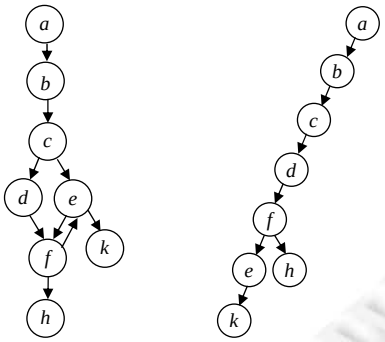
$$sdom(b_j)=\bigcap_i(b_i\cup sdom(b_i)) \tag{1}$$

其中, 节点 b_i 是节点 b_j 的直接前驱, $sdom(b_i)$和 $sdom(b_j)$分别代表 b_i 和 b_j 的严格支配节点集合. 初始, 入口节点的严格支配节点集合为空, 其他所有节点的严格支配节点集合包括所有节点. 如果将每个节点抽象为控制流图对应的深度优先搜索树(depth first search tree, DFST)的逆后序遍历(reverse post order, rPO)次序, 那么按照该次序处理每个节点, 通常可以加快收敛速度, 提高迭代算法的效率.

一个高效的迭代计算严格支配节点的算法由 Cooper、Harvey 和 Kennedy 等人提出(以下简称 CHK 算法)[2]. 算法的基本思想是: 使用列表(list)存储严格支配节点, 若按 rPO 逆序排序, 在每次迭代过程中, 取出按 rPO 升序排序的工作表首元素 h, 数据流传递函数(transfer function)取 h 的支配节点值(始终是一个逆序列表), 将 h 添加到该值的首部, 其结果仍将是一个逆序列表. 因此, 在传播中, 集合的交运算可以实现为两个逆序列表的“交”运算, 其结果仍是一个逆序列表, 它只包括参于运算的两个列表都具有的元素, 该计算可仅由一遍遍历两个列表而完成, 该计算对应于第 1 节定义的 *inter_sorted_rev* 函数.

CHK 算法的前提条件是, 每个节点的支配节点列表是有序的. 在本文的实现中, 支配节点列表保持逆序. 图 1 给出了 CHK 算法作用在一个控制流图上的输入和输出, 其中,

- 图 1(a)是控制流图;
- 图 1(b)是该控制流图的一棵 DFST, 其后序遍历是 *k-e-h-f-d-c-b-a*; 将所有节点抽象为对应于该 DFST 的 rPO 次序, 因此, 节点 a、b、c、d、e、f、h、k 分别对应于 0、1、2、3、6、4、5、7;
- 图 1(c)给出了各节点抽象为 rPO 次序后对应的直接后继节点(节点 k 无直接后继节点, 以短横线表示)、初始严格支配节点、初始工作表(包括所有非“稳定”节点, 第 3 节将给出它的准确定义), 以及迭代完成后, 各节点的严格支配节点列表. 可以看出: 在迭代结果中, 每个节点的严格支配节点列表仍然是一个逆序列表, 并且列表的首元素就是该节点的直接支配节点.



(a)

(b)

节点	直接后继节点	初始严格支配节点	结果严格支配节点
0	1	[·]	[·]
1	2	[7,6,5,4,3,2,1,0]	[0]
2	3, 6	[7,6,5,4,3,2,1,0]	[1,0]
3	4	[7,6,5,4,3,2,1,0]	[2,1,0]
4	5, 6	[7,6,5,4,3,2,1,0]	[2,1,0]
5	0	[7,6,5,4,3,2,1,0]	[4,2,1,0]
6	4, 7	[7,6,5,4,3,2,1,0]	[2,1,0]
7		[7,6,5,4,3,2,1,0]	[6,2,1,0]

(c) 初始工作表: [0,1,2,3,4,5,6]

图 1 CHK 算法输入和输出示例

图 1 给出的是一个非可归约(nonreducible)[3]的控制流图, 目的是表明迭代算法对非可归约控制流图也是

适用的.

表 1 给出了作用在图 1 控制流图上的迭代传播计算过程. 第 1 列代表每次迭代取出的工作表首元素, 每次迭代过程中, 代表数据流传递结果值的第 2 列将始终保持逆序, 该值传播到第 1 列所代表节点的所有后继节点, 如第 3 列所示, 并与后继节点的原值进行“交”运算, 如果“交”运算的结果改变了原值, 则更新, 更新后的值以粗体显示. 若更新后值的后继节点不存在于当前工作表中, 则添加到工作表中, 并使工作表始终保持升序, 如表 1 中第 4 列所示.

表 1 CHK 迭代传播过程示例

当前工作表首元素	数据流函数传递的值	每次传播后数据流分析的值								更新后的工作表
		0	1	2	3	4	5	6	7	
0	[0]		**[0]**							[1,2,3,4,5,6]
1	[1,0]			**[1,0]**						[2,3,4,5,6]
2	[2,1,0]				**[2,1,0]**			**[2,1,0]**		[3,4,5,6]
3	[3,2,1,0]					**[3,2,1,0]**				[4,5,6]
4	[4,3,2,1,0]						**[4,3,2,1,0]**	[2,1,0]		[5,6]
5	[5,4,3,2,1,0]	[·]								[6]
6	[6,2,1,0]					**[2,1,0]**			**[6,2,1,0]**	[4,7]
4	[4,2,1,0]						**[4,2,1,0]**	[2,1,0]		[5,7]
5	[5,4,2,1,0]	[·]								[7]
7	[7,6,2,1,0]									[]

Cooper 等人指出: 采用这种方式实现的迭代算法对实际程序的控制流图进行求解, 其效率甚至快于接近线性时间复杂度、但却非常复杂的 Lengauer-Tarjan 算法(以下简称 LT 算法)[4,5].

机械化验证高效的求解支配节点的算法, 通常是获得一个实际的“验证编译器(verified compiler)”[6−11]不可或缺的一部分. 因此, 本文讨论如何在定理证明助手 Isabelle/HOL[12]的支持下, 证明 CHK 算法的正确性, 并进一步证明其完备性, 实现代码见 Isabelle 形式化证明开源站点[13].

本文剩余部分所述的节点都是指节点对应的 rPO 次序. 控制流图 G 中的节点类型是 nat. 入口节点是 0, 每个节点的支配节点列表是一个逆序列表, 其类型是 nat list. 由于每个节点已经抽象为其 rPO 次序, 除了入口节点外, 其他所有节点都存在一个直接前驱, 其 rPO 次序小于自身, 即满足性质:

$$\forall v\in set(g\_V\ G)-\{0\}.\exists prev.(prev,v)\in g\_E\ G\wedge prev<v \tag{2}$$

其中, $g\_V\ G$ 表示 G 中的所有节点; $g\_E\ G$ 表示 G 中的所有边, 其类型是$(nat\times nat)\ set$.

本文第 1 节讨论逆序列表半格结构(A,r,f)的建立和性质证明. 第 2 节讨论基于工作表(worklist)的 CHK 算法的实现, 迭代计算严格支配节点. 第 3 节首先给出支配节点的定义性规范和相关性质定理, 然后构造并证明迭代求解算法所满足的重要性质, 最后利用这些性质定理, 相对于支配节点的定义性规范, 证明 CHK 迭代求解算法的正确性和完备性. 最后总结全文.

2 严格支配节点列表的半格

为了迭代求解严格支配节点, 首先建立逆序列表的半格(semilattice)结构. 每个节点的严格支配节点都是控制流图中所有节点的子集, 因此, 代表严格支配节点的所有逆序列表的集合是

$$(rev\circ sorted\_list\_of\_set)\ `\ (Pow(set(g\_V\ G))) \tag{3}$$

其中, $\circ$是函数组合运算符, *sorted_list_of_set* 按照集合中元素的大小关系, 将集合转换为升序列表, *rev* 函数将其变为一个逆序; *Pow* 是幂集运算; `是映射运算符; *set* 函数将列表转换为集合. 公式(3)是数据流分析中解的值域.

两个逆序列表 *xs* 和 *ys* 的大小比较运算 *nodes_le* 定义为

$$nodes\_le\ xs\ ys=sorted(rev\ xs)\wedge sorted(rev\ ys)\wedge(set\ ys)\subseteq(set\ xs)\vee xs=ys \tag{4}$$

该定义要求两个列表 *xs* 和 *ys* 相同, 或者: 它们都是逆序的, 由 *sorted rev* 定义, 且 *set ys* 是 *set xs* 的子集.

两个逆序列表的上确界运算定义为 *nodes_sup*, 以下称其为合并运算, 使用 *inter_sorted_rev* 定义为

$$nodes\_sup=(\lambda x\ y.inter\_sorted\_rev\ x\ y) \tag{5}$$

其中, *inter_sorted_rev* 是一个递归函数:

$$\left.\begin{array}{l} inter\_sorted\_rev\,[\cdot]\,l2=[\cdot] \\ inter\_sorted\_rev\,l1\,[\cdot]=[\cdot] \\ inter\_sorted\_rev\,(x1\#l1)\,(x2\#l2)= \\ \quad (\textbf{if}\ (x1>x2)\ \textbf{then}\ (inter\_sorted\_rev\,l1\,(x2\#l2)) \\ \quad\ \textbf{else}\ (\textbf{if}\ (x1=x2)\ \textbf{then}\ x1\#(inter\_sorted\_rev\,l1\,l2) \\ \quad\quad \textbf{else}\ inter\_sorted\_rev\,(x1\#l1)\,l2)) \end{array}\right\} \tag{6}$$

其中, *c* 代表 *xs* 在合并之后是否发生了改变. 可以证明: 两个逆序列表合并运算的结果仍然是逆序的, 其元素是两个逆序列表对应集合的交集, 即定理 *inter_sorted_correct*.

lemma inter_sorted_correct:

assumes "*sorted*(*rev xs*)" and "*sorted*(*rev ys*)"

shows "*set*(*inter_sorted_rev xs ys*)=*set xs*∩*set ys*∧*sorted*(*rev*(*inter_sorted_rev xs ys*))"

利用上述定义, 可将半格结构定义为一个三元组 *nodes_semi*:

$$nodes\_semi=((rev\circ sorted\_list\_of\_set)`(Pow(set(g\_V\,G))),nodes\_le,nodes\_sup) \tag{7}$$

将半格结构中的值域、大小比较运算以及合并运算分别表示为 *A*, *r* 和 *f*, *r*. *f* 运算分别记为$\sqsubseteq_r$和$\sqcup_f$, 可以证明三元组(*A*,*r*,*f*)是一个半格, 即满足以下条件:

$$\left.\begin{array}{l} order\ r\ A\wedge closed\ A\ f\ \wedge \\ (\forall x\in A.\,\forall y\in A.\,x\sqsubseteq_r x\sqcup_f y)\wedge(\forall x\in A.\,\forall y\in A.\,y\sqsubseteq_r x\sqcup_f y) \\ (\forall x\in A.\,\forall y\in A.\,\forall z\in A.\,x\sqsubseteq_r z\wedge y\sqsubseteq_r z\rightarrow x\sqcup_f y\sqsubseteq_r z \end{array}\right\} \tag{8}$$

其中, *order r A* 和 *closed A f* 的定义分别是

$$\left.\begin{array}{l} order\ r\ A\leftrightarrow(\forall x\in A.\,x\sqsubseteq_r x)\wedge \\ \qquad\qquad (\forall x\in A.\,\forall y\in A.\,x\sqsubseteq_r y\wedge y\sqsubseteq_r x\rightarrow x=y)\wedge \\ \qquad\qquad (\forall x\in A.\,\forall y\in A.\,\forall z\in A.\,x\sqsubseteq_r y\wedge y\sqsubseteq_r z\rightarrow x\sqsubseteq_r z) \\ closed\ A\ f\leftrightarrow(\forall x\in A.\,\forall y\in A.\,x\sqcup_f y\in A) \end{array}\right\} \tag{9}$$

因此, *f* 运算在集合 *A* 上是闭的, *r* 是一个偏序, 它满足自反、反对称和传递性质.

偏序 *r* 满足"上升链条件(ascending chain condition, acc)", 即它不会无限增长, 由谓词 *acc* 定义. 为了证明 *acc r* 成立, 可以利用 Isabelle/HOL 支持的 Well-foundedness 概念. Well-foundedness 由谓词 *wf* 定义, 它等价于 *acc*:

$$acc\ r\leftrightarrow wf\{(y,x).x\sqsubset_r y\} \tag{10}$$

其中, $x\sqsubset_r y\leftrightarrow x\sqsubseteq_r y\wedge x\neq y$.

由于可以证明: 对于两个逆序列表 *xs* 和 *ys*, 如果其元素对应的集合满足真子集关系, 即 *set xs*⊂*set ys*, 那么 *xs* 列表的长度小于 *ys* 列表的长度, 即定理 *sorted_rev_subset_len_lt*:

lemma sorted_rev_subset_len_lt:

assumes "*sorted*(*rev xs*)" and "*sorted*(*rev ys*)" and "(*set xs*)⊂(*set ys*)"

shows "*length xs*<*length ys*"

该定理表示, 可以为 *nodes_le* 这个偏序关系指定由列表到自然数的度量函数 *length*. 对于两个有序列表 *xs* 和 *ys*, 如果 *xs* 对应的集合是 *ys* 对应集合的子集, 并且 *xs*≠*ys*, 那么 *xs* 对应的集合也是 *ys* 对应集合的真子集. 联合定理 *sorted_rev_subset_len_lt*, 可以证明 *nodes_le* 的逆(converse)满足谓词 *wf*, 即定理 *wf_nodes_le*:

lemma wf_nodes_le: *Wellfounded.wf*{(*y*,*x*).*nodes_le x y*∧*x*≠*y*}.

因此可以证明定理 *acc_nodes_le*:

lemma acc_nodes_le: *acc_nodes_le*.

将所有节点的支配节点的类型定义为 *nat list list*, 它们之间的大小比较关系可以定义为 *Listn.le nodes_le*, 记 *Listn.le nodes_le ss ss′*为 $ss[\sqsubseteq_r]ss'$, 其中, *Listn.le r*=*list_all2*($\lambda x\ y.x\sqsubseteq_r y$).

可以证明: 如果 *acc r* 成立, 那么 *acc* (*Listn.le r*)成立. 因此, 利用定理 *acc_nodes_le* 可以推得 *acc*(*Listn.le nodes_le*), 利用 *acc* 和 *wf* 的等价, 可以推得定理 *wf_listn_le*:

lemma wf_listn_le: *Wellfounded.wf*{(*y*,*x*).$x[\sqsubset_r]y$}.

3 迭代计算严格支配节点

为了求解每个节点的严格支配节点列表, 构造“稳定(stable)”函数:

$$stable\ r\ step\ ss\ p \leftrightarrow (\forall (q,s) \in set(step\ p(ss!p)).s \sqsubseteq_r ss!q) \tag{11}$$

其中, *ss* 是所有节点的严格支配节点的列表. *step* 是流函数, 它的类型是 *nat*⇒*nat list*⇒(*nat*×*nat list*) *list*. 该流函数不仅刻画了流函数的执行结果, 而且还记录了流函数继续向后执行的位置, 它是节点 *p* 的所有后继节点. *step* 的定义是

$$step\ n\ input = map(\lambda pc.(pc,(transf\ n\ input)))\ (rev(sorted\_list\_of\_set(succs\ n))) \tag{12}$$

其中, *transf n input*=*n*#*input*. 运算符“#”在列表首位置添加元素 *n*, *succs n* 取得节点 *n* 的所有后继节点集合. 由于函数 *sorted_list_of_set* 返值是升序列表, 利用函数 *rev* 转换为逆序列表.

因此, 节点 *p* 是“稳定”的, 是指: 流函数在节点 *p* 处执行后的值小于 *p* 的所有后继节点的值. 所有节点都是稳定的, 由 *stables* 定义:

$$stables\ r\ step\ ss \leftrightarrow (\forall p < size\ ss.stable\ r\ step\ ss\ p) \tag{13}$$

如果所有节点都是稳定的, 数据流分析到达一个固定点, 此时, 数据流分析的值是数据流等式(1)的解.

采用 Kildall 工作表的迭代算法, 工作表的类型是 *nat list*. 迭代函数 *iter* 利用 Isabelle/HOL 支持的 while 组合子, 可以定义为

$$\left.\begin{array}{l} iter\ f\ step\ ss\ w = \mathbf{while}\ (\lambda(ss,w).\ w \neq [\cdot]) \\ \qquad (\lambda(ss,w).\ let\ p = hd\ w\ in\ propa\ f\ (step\ p\ (ss!p))\ ss\ (tl\ w)) \\ \qquad (ss,w) \end{array}\right\} \tag{14}$$

其中, 传播函数 *propa* 是递归定义:

$$\left.\begin{array}{l} propa\ f\ [\cdot] \quad ss\ wl = (ss,wl) \\ propa\ f\ (q'\#qs)\ ss\ wl = (let\ (q,s) = q';\ u = (s \sqcup_f ss!q); \\ \qquad wl' = (\mathbf{if}\ u = ss!q\ \mathbf{then}\ wl\ else\ sorted\_list\_of\_set(insert\ q\ (set\ wl))) \\ \qquad \text{in}\ propa\ f\ qs\ (ss[q := u])\ wl') \end{array}\right\} \tag{15}$$

因此, 传播函数将 *transf*(*hd w*) (*ss*!*hd w*)的结果和(*hd w*)的每个后继节点的值进行合并运算, 如果后继节点的原值与合并运算的结果值不相等, 则更新为合并运算的结果值, 并将该后继节点添加到工作列表中. 工作列表通过 *sorted_list_of_set* 始终保持升序, 使得每次迭代取的工作列表首元素是具有最小 rPO 的节点.

初始时, 设置入口节点的严格支配节点列表为空, 其他节点的严格支配节点包括所有节点, 它是一个逆序列表, 即 *rev*[0..<*length*(*g_V G*)], 初始状态由 *start* 定义:

$$start = [\cdot]\#(replicate\ (length(g\_V\ G)-1)\ (rev[0..<length(g\_V\ G)])) \tag{16}$$

初始工作表包括所有非稳定节点, 即“*unstables r step start*”, 其中, *unstables* 的定义是:

$$unstables\ r\ step\ ss = sorted\_list\_of\_set\ \{p.p < size\ ss \wedge \neg stable\ r\ step\ ss\ p\} \tag{17}$$

按照 *stable* 和 *stables* 的定义, 可以证明初始工作表包括所有具有后继节点的节点:

lemma unstable_start: *unstable r step start*=*sorted_list_of_set*({*p*.*succs p*≠{·}∧*p*<*length start*}).

每次取工作表首元素, 设为 *n*, 在数据流分析处理过程中, *transf n input* 的结果将总是逆序列表. 传播函数将该值传播到 *n* 的所有后继. 在迭代过程中, 工作列表因取出工作列表首元素而变小, 或者那些与合并运算结果值不同的后继节点(如果不存在于工作列表中)添加到工作表中. 由于合并运算的结果总大于等于参于合并

运算的两个列表, 而偏序关系 *nodes_le* 不会无限增长, 给定 *start* 初值, 该迭代算法总会到达一个稳定点.

令 *kildall r f step ss*=*fst*(*iter f step ss* (*unstable r step ss*)), 于是, 利用公式(8)定义的半格, 迭代计算严格支配节点的主函数 *dom_kildall* 是:

$$dom\_kildall = kildall\ (fst(snd\ nodes\_semi))\ (snd(snd\ nodes\_semi))\ step \tag{18}$$

结合初值 *start*, 可以定义节点 *i* 是否严格支配节点 *j* 为 *strict_dom*:

$$strict\_dom\ i\ j = (let\ res = (dom\_kildall\ start)!j\ \text{in}\ i \in set\ res) \tag{19}$$

以及节点 *i* 是否支配节点 *j* 为 *dom*:

$$dom\ i\ j = (let\ res = (dom\_kildall\ start)!j\ \text{in}\ i \in set\ res \vee i = j) \tag{20}$$

利用 *strict_dom* 和 *dom* 的定义, 可以直接推得 *dom_sdom* 和 *zero_dom_zero* 这两个定理.

- *lemma dom_sdom*: $dom\ i\ j \Rightarrow i \neq j \Rightarrow strict\_dom\ i\ j$;
- *lemma zero_dom_sdom*: $dom\ i\ 0 \leftrightarrow i = 0$.

为了证明迭代算法的正确性, 第 4 节将讨论更多更为复杂的性质定理.

4 正确性和完备性

本节证明第 3 节迭代计算支配节点的正确性和完备性. 迭代算法的正确性是指: 通过迭代计算得到节点 *i* 和节点 *j* 之间的支配关系满足某个预先定义的支配关系规范, 将由第 4.2 节末的定理 *dom_sound* 进行陈述, 其中, *dominate* 是支配关系的定义性规范. 由定理 *dom_sound* 将进一步推导出严格支配关系的正确性: 通过迭代计算得到的节点 *i* 和节点 *j* 之间的严格支配关系满足某个预先定义的严格支配关系规范, 将由第 4.2 节末的定理 *sdom_sound* 加以陈述, 其中, *strict_dominate* 是严格支配关系的定义性规范. 迭代算法的完备性是指: 所有满足支配关系规范定义的严格支配关系, 都能经由迭代计算而得到, 它是严格支配关系正确性的逆, 将由第 4.3 节末的定理 *sdom_complete* 给出陈述.

4.1 支配节点的定义性规范

为了证明第 3 节迭代计算支配节点的正确性, 首先给出支配节点的定义性规范. 定义由入口节点 0 到达任意节点的路径函数为 *path_entry*:

$$\left.\begin{array}{l} path\_entry\ 0 : path\_entry\ E\ [\cdot]\ 0 \\ path\_entry\_prepend : [\![(u,v) \in E;\ path\_entry\ E\ l\ u]\!] \Rightarrow path\_entry\ E\ (u\#l)\ v \end{array}\right\} \tag{21}$$

按照该定义, 入口节点到达某个节点 *n* 的路径列表的首元素不是 *n*. 给定控制流图 *G*, 定义支配和严格支配节点的正确性规范分别为 *dominate* 和 *strict_dominate*:

$$\left.\begin{array}{l} dominate\ i\ j \equiv \forall pa.path\_entry\ (g\_E\ G)\ pa\ j \rightarrow (i \in set\ pa \vee i = j) \\ strict\_dominate\ i\ j \equiv \forall pa.path\_entry\ (g\_E\ G)\ pa\ j \rightarrow (i \in set\ pa \wedge i \neq j) \end{array}\right\} \tag{22}$$

这两个定义性规范满足的性质包括:

- *lemma sdominate_trans*: $dominate\ n1\ n2 \Rightarrow n1 \neq n2 \Rightarrow strict\_dominate\ n1\ n2$;
- *lemma sdominate_dominate_succs*: $strict\_dominate\ i\ j \Rightarrow j \in succs\ k \Rightarrow dominate\ i\ k$.

4.2 正确性

为了证明该迭代算法的正确性, 首先定义:

$$\left.\begin{array}{l} wf\_dom\ ss\ w \equiv (\forall s \in set\ ss.\ s \in A) \wedge \\ \quad (\forall p < n.\ sorted\ (rev(ss!p)) \wedge \\ \quad\quad (ss!p \neq rev[0..<n] \rightarrow (\forall x \in set(ss!p).\ x < p)) \wedge \\ \quad\quad (ss!p = rev[0..<n] \rightarrow (\exists x \in set\ w.\ (x,p) \in g\_E\ G \wedge x < p)) \wedge \\ \quad\quad (p \notin set\ w \rightarrow stable\ r\ step\ ss\ p)) \wedge \\ \quad sorted\ w \wedge length\ ss = n \wedge (\forall x \in set\ w.\ x < n) \end{array}\right\} \tag{23}$$

其中, n 是节点个数, A、r 和 f 是第 2 节定义的半格, *stable* 函数在第 3 节已给出定义. *wf_dom* 刻画了在迭代计算严格支配节点过程中保持的重要性质: 每个节点的严格支配节点都是值域 A 中的值. 每个节点的严格支配节点列表都保持逆序; 如果某个节点 p 的严格支配节点不是 $rev[0..<n]$, 即非初始值, 那么 p 的所有严格支配节点都小于 p; 反之, 如果节点 p 的严格支配节点是 $rev[0..n]$, 那么在工作列表中总存在一个节点 x, 它是节点 p 的直接前驱, 并且它小于 p; 所有不存在于工作列表中的节点都是稳定的. 工作列表保持升序. 代表所有节点的严格支配节点列表的长度是 n. 工作列表中所有节点都小于 n.

可以证明, 初始状态 *start* 和初始工作节点列表"*unstables r step start*"满足这些不变式, 即定理 *wf_start*:

$$lemma\ wf\_start: wf\_dom\ start(unstables\ r\ step\ start).$$

接下来证明 *wf_dom* 定义的不变式在迭代传播过程中的保持性定理 *propa_dom_invariant*:

$$\begin{aligned}&lemma\ propa\_dom\_invariant:\\&\ wf\_dom\ ss\ w\Rightarrow\\&\ w\neq[\cdot]\Rightarrow\\&\ propa\ f\ (step(hd\ w)\ (ss!hd\ w))\ ss(tl\ w)=(ss',w')\Rightarrow\\&\ wf\_dom\ ss'\ w'.\end{aligned}$$

为了完成这个证明目标, 首先构造并证明流函数的有界性、保持性和单调性. 有界性定义为

$$bounded\ step\ n\leftrightarrow(\forall p<n.\forall s.\forall(q,s')\in set(step\ p\ s).q<n) \tag{24}$$

容易证明 *bounded_step*:

$$lemma\ bounded\_step: bounded\ step\ n.$$

流函数的保持性定理构造为 *pres_step*:

$$\begin{aligned}&lemma\ pres\_step:\\&assumes\ "(q,\tau)\in set(step\ p\ (ss!p))"\\&and\ "\forall n\in set(ss!p).p>n"\\&and\ "ss!p\in A"\\&and\ "p<n"\\&shows\ "\tau\in A?".\end{aligned}$$

由于 A 中所有列表都是逆序的, 在满足"$\forall n\in set((ss!p)).p>n$"的条件下, $p\#ss!p$ 仍然是逆序, p 和 $ss!p$ 中的元素都是合法节点, 因此, 该定理可以得证.

在满足 $sorted(rev(transf\ p\ \tau))$的前提下, 可以构造并证明流函数的单调性 *mono_step*:

$$\begin{aligned}&lemma\ mono\_step:\forall\tau\ p\ \tau'.\tau\in A\wedge p<n\wedge\tau\sqsubseteq_r\tau'\rightarrow\\&\qquad sorted(rev(transf\ \ p\ \tau))\rightarrow\\&\qquad set(step\ \ p\ \tau)\ \{\sqsubseteq_r\}\ set(step\ \ p\ \tau'),\end{aligned}$$

其中, $A\{\sqsubseteq_r\}B\equiv\forall(p,\tau)\in A.\exists\tau'.(p,\tau')\in set\ B\wedge\tau\sqsubseteq_r\tau'$.

在每次迭代过程中, 流函数计算的是 $transf\ (hd\ w)\ (ss!hd\ w)$. 在满足 *wf_dom ss w* 的前提下, 由 *wf_dom* 的定义可得:

(1) $\forall p<n.ss!p\neq rev[0..<n]\rightarrow\forall x\in set(ss!p).x<p$;

(2) $\forall p<n.ss!p=rev[0..<n]\rightarrow(\exists x\in set\ w.(x,p)\in g\_E\ G\wedge x<p)$;

以及$\forall p<n.\ sorted(rev(ss!p))$, $\forall x\in set\ w.x<n$ 和 *sorted w*. 于是, $hd\ w<n$ 和 $sorted(rev(ss!hd\ w))$成立. 可推得 $ss!hd\ w$ 不可能是 $rev[0..<n]$, 这是因为: 如果 $ss!hd\ w=rev[0..<n]$, 由结论(2)可推得$\exists x\in set\ w.(x,hd\ w)\in g_E\ G\wedge x<hd\ w)$. 显然, 这与 *sorted w* 所要求的"*w* 中的所有元素都大于等于首元素 *hd w*"相矛盾. 联合结论(1), 可以推得:

(3) $\forall x\in set(ss!hd\ w).x<hd\ w$.

联合 $sorted(rev(ss!hd\ w))$, 可推得 $transf\ (hd\ w)\ (ss!hd\ w)$的结果仍然是逆序列表. 由结论(3)、*wf_dom* 的定义以及保持性定理, 可以证明$(q,\tau)\in set(step\ (hd\ w)\ (ss!hd\ w)).\tau\in A$. 于是, 可以建立传播之后的值$(ss',w')$和传播

之前的值(*ss*,*w*)之间的联系.

- $(q,\tau)\in set(step\ (hd\ w)\ (ss!hd\ w)).ss'!q=(hd\ w\#(ss!hd\ w))\sqcup_f ss!q$;
- $(q,\tau)\notin set(step\ (hd\ w)\ (ss!hd\ w))\Rightarrow q<length\ ss\Rightarrow ss'!q=ss!q$.

现在来证明 *propa_dom_invariant*. 首先利用 *cases* 证明命令, 区分属于 *hd w* 后继的节点和不属于 *hd w* 后继的节点. 证明进一步涉及不同情况下的 *cases* 分析, 譬如节点的严格支配节点是否包括所有节点、工作列表是否为空列表等等, 并涉及与控制流图相关的许多细节问题. 譬如, 在该证明中要求控制流图中的节点没有自循环, 即不存在节点到节点本身的边. 由于控制流图中的节点已经抽象为其 rPO 次序, 因此对于除入口节点以外的任意节点 *v*, 都存在一个直接前驱节点 *prev*, 满足 *prev*<*v*, 如公式(2)所示.

证明了 *wf_dom* 定义的不变式在迭代过程中的保持性之后, 继续证明算法终止时所满足的后置条件, 除了满足 *wf_dom* 定义的不变式外, 它们还包括:

(1) *stables r step res*;

(2) *start* $[\sqsubseteq_r]$ *res*

(3) $\forall ts\in list\ n\ A.start\ [\sqsubseteq_r]\ ts\wedge stables\ r\ step\ ts\rightarrow res\ [\sqsubseteq_r]\ ts$.

其中, *res*=*dom_kildall start*, 是迭代计算的结果. 由于已经证明了每次迭代的结果满足不变式: $\forall p<n.\ p\notin set\ w\rightarrow stables\ r\ step\ ss\ p$, 因此, 如果 *w* 为空, 可以直接证明条件(1). 利用类似证明上述 *wf_dom* 不变式保持性的证明模式, 可以证明每次迭代的结果 *ss*′满足 *start* $[\sqsubseteq_r]$ *ss*′以及$\forall ts\in list\ n\ A.\ start\ [\sqsubseteq_r]\ ts\wedge stables\ r\ step\ ts\rightarrow ss'\ [\sqsubseteq_r]\ ts$, 于是, 条件(2)和条件(3)能够得证.

接下来, 证明中止关系满足 Well-foundedness 性质. 工作表是一个升序列表, 容易证明:

$$x\subset y\Rightarrow finite\ y\Rightarrow length\ (sorted\_list\_of\_set\ x)<length\ (sorted\_list\_of\_set\ y).$$

于是可以证明:

$$Wellfounded.wf\{\lambda(x,y).(sorted\_list\_of\_set\ x,sorted\_list\_of\_set\ y)\ `\ finite\_psubset\},$$

其中, $finite\_psubset=\{(A,B).A\subset B\wedge finite\ B\}$.

联合第 2 节证明的定理 *wf_listn_le*, 由 Isabelle/HOL 支持的定理 *wf_lex_prod* 可推得中止关系满足 Well-foundedness 性质, 即定理 *wf_termination_rel*:

$$\begin{aligned}&lemma\ wf\_termination\_rel:\\&Wellfounded.wf\ \{(ss',ss).\ ss\ [\sqsubset_r]\ ss'\}\langle *lex*\rangle\\&\qquad\{\lambda(x,y).(sorted\_list\_of\_set\ x,sorted\_list\_of\_set\ y))\ `\ finite\_psubset\}.\end{aligned}$$

即, 迭代计算严格支配节点的中止关系满足 Well-foundedness 性质. 该性质表明了中止性: 由于执行了合并运算的传播后的值总大于传播前的值, 按照 *Listn.le r* 关系满足上升链条件, 不会无穷增长; 传播后的工作列表总是一个有限真子集, 因此工作列表中的元素总会逐渐减少.

最后可以证明 *propa_termination*:

$$\begin{aligned}&lemma\ propa\_termination:\\&(propa\ f\ (step\ (hd\ w)\ (s!hd\ w))\ a\ (tl\ w),(ss,w)\in\\&\{(ss',ss).\ ss\ [\sqsubset_r]\ ss'\}\langle *lex*\rangle\\&\{\lambda(x,y).(sorted\_list\_of\_set\ x,sorted\_list\_of\_set\ y))\ `\ finite\_psubset\}.\end{aligned}$$

即迭代总会沿着该中止关系逐步减小, 算法总会终止.

综合以上证明, 可以推得定理 *iter_dom_properties*:

$$\begin{aligned}&lemma\ iter\_dom\_properties:\\&(iter\ f\ step\ start\ (unstables\ r\ step\ start)\ =\ (ss',w')\rightarrow\\&wf\_dom\ ss'\ w'\wedge\ stables\ r\ step\ ss'\wedge(start\sqsubseteq_r\ ss'\wedge\\&\forall ts\in list\ n\ A.\ start\ [\sqsubseteq_r]\ ts\wedge stables\ r\ step\ ts\rightarrow ss'\ [\sqsubseteq_r]\ ts,\end{aligned}$$

并证明"*kildall r f step start*"是迭代等式的解, 它满足以下条件:

$$\begin{aligned}&kildall\ r\ step\ start \in list\ n\ A \wedge\\ &stables\ r\ step\ (kildall\ r\ step\ start) \wedge (start \sqsubseteq_r ss' \wedge\\ &\forall ts \in list\ n\ A.\ start\ [\sqsubseteq_r]\ ts \wedge stables\ r\ step\ ts \rightarrow (kildall\ r\ step\ start)\ [\sqsubseteq_r]\ ts.\end{aligned}$$

于是, 数据流等式(1)的解 *kildall r f step start* 在每个节点处都是稳定的. 在稳定状态下, 如果当前迭代状态中的工作列表不为空, 再次调用传播函数时, 不会再添加新的后继节点到工作列表中, 迭代状态将始终保持不变, 因此, 工作列表将随每次迭代而减少, 最后为空.

利用 *iter_dom_properties* 可以证明 *dom* 和 *strict_dom* 满足一个重要性质 *sdom_dom_succs*:

$$lemma\ sdom\_dom\_succs: strict\_dom\ i\ j \Rightarrow j \in succs\ k \Rightarrow dom\ i\ k.$$

即: 如果节点 *i* 严格支配节点 *j*, *j* 是节点 *k* 的一个直接后继, 那么 *i* 支配 *k*. 结合第 3 节中证明的定理 *dom_sdom*, 可以直接证明 *dom_succs*:

$$lemma\ dom\_succs: dom\ i\ j \Rightarrow i \neq j \Rightarrow j \in succs\ k \Rightarrow dom\ i\ k.$$

利用定理 *dom_succs*, 可以建立迭代计算严格支配节点与控制流图路径之间的关系, 即 *dom* 和 *path_entry* 之间所满足的一个重要性质 *path_entry_dom*:

$$\begin{aligned}&lemma\ path\_entry\_dom:\\ &\wedge pa\ n\ d.\ path\_entry\ (g\_E\ G)\ pa\ n \Rightarrow dom\ d\ n \Rightarrow d \in set\ pa \vee d = n.\end{aligned}$$

该定理表示: 如果从入口节点到节点 *n* 具有一条路径 *pa*, 并且按照迭代计算的结果, 节点 *d* 支配 *n*, 那么 *d* 肯定存在于 *pa* 中, 或者 *d*=*n*. 证明过程是: 在归纳规则 *path_entry.induct* 进行归纳. 对于基本步, 如果 *dom d* 0, 由第 3 节证明的定理 *zero_dom_zero* 可推得 *d*=0; 对于归纳步, 利用 *dom_succs* 可以得证.

因此, 现在可以构造并证明迭代算法的正确性定理 *dom_sound*, 它表明: 第 3 节中通过迭代算法计算出的支配关系满足第 4.1 节中定义的支配关系规范:

$$lemma\ dom\_sound: dom\ i\ j \Rightarrow dominate\ i\ j.$$

主要利用上述证明的定理 *path_entry_dom*, 该正确性定理可以得证. 利用 *dom_sound*, 可直接证明迭代计算的严格支配关系也是正确的:

$$lemma\ sdom\_sound: strict\_dom\ i\ j \Rightarrow j \in set(g\_V\ G) \Rightarrow strict\_dominate\ i\ j.$$

4.3 完备性

完备性是正确性的逆, 刻画为

$$lemma\ sdom\_complete: strict\_dominate\ i\ j \Rightarrow j < length\ start \Rightarrow strict\_dom\ i\ j.$$

即: 所有满足支配节点规范定义的严格支配关系, 都能经由迭代算法计算而得到.

通过反向证明来证明完备性. 构造定理 *notsdom_notsdominate*: 如果经由迭代计算, 节点 *i* 不严格支配节点 *j*, 那么按照支配关系规范, *i* 也不严格支配 *j*, 即 *notsdom_notsdominate*:

$$\begin{aligned}&lemma\ notsdom\_notsdomiante:\\ &\neg strict\_dom\ i\ j \Rightarrow j < length\ start \Rightarrow non\_strict\_dominate\ i\ j,\end{aligned}$$

其中, *non_strict_dominate* 定义为

$$non\_strict\_dominate\ i\ j = \exists pa.path\_entry(g\_E\ G)\ pa\ j \wedge (i \notin set\ pa) \tag{25}$$

显然, 由该定义可推得非 *strict_dominate i j* 成立:

$$lemma\ nonstrict\_eq: non\_strict\_dominate\ i\ j \Rightarrow \neg strict\_dom\_dominate\ i\ j.$$

为了证明 *notsdom_notsdominate*, 需要证明在迭代过程中保持的另外一个不变式, 构造为

$$\begin{aligned}&(\forall i.\ i < n \wedge k \notin set(ss!i) \rightarrow non\_strict\_dominate\ k\ i) \Rightarrow\\ &propa\ f\ (step\ (hd\ w)\ (ss!hd\ w))\ ss\ (tl\ w) = (ss', w') \Rightarrow\\ &(\forall i.\ i < n \wedge k \notin set(ss'!i) \rightarrow non\_strict\_dominate\ k\ i) \Rightarrow\end{aligned}$$

该不变式保持性的证明与第 4.2 节中定理 *propa_dom_invariant* 的证明类似, 这里不再赘述.

于是能够证明 *complete_aux*:

$$\begin{aligned}&\textit{lemma complete\_aux}:\\&\quad \textit{iter } f\ \textit{step start}\ (\textit{unstables } r\ \textit{step start}) = (ss', w') \rightarrow\\&\quad i < n \wedge ss'!i = \textit{res} \wedge k \notin \textit{set res} \rightarrow\\&\quad \textit{non\_strict\_dominate } k\ i.\end{aligned}$$

结合第 4.2 节中已经证明的定理 *wf_termination_rel* 和 *propa_termination*, 可以证明 *notsdom_notsdominate*. 因此, 结合 *contrapos_nn*、*nonstrict_eq* 可以证明迭代算法的完备性 *dom_complete*:

$$\textit{lemma sdom\_complete}: \textit{strict\_dominate } i\ j \Rightarrow j < \textit{length start} \Rightarrow \textit{strict\_dom } i\ j.$$

5 总 结

本文讨论了如何使用定理证明助手 Isabelle/HOL 证明一个高效的迭代算法, 计算数据流图中节点的支配关系. 该迭代计算将节点抽象为逆后序遍历的次序, 按照该次序, 工作表保持升序, 迭代分析的值保持逆序. 用这种方式对实际程序的控制流图进行求解, 其效率甚至快于 LT 算法. 为了机械化证明该算法, 建立了逆序列表的半格结构, 并证明了它是一个半格, 且满足“上升链条件”. 然后使用半格结构, 实现了一个基于工作表的 Kildall 迭代算法. 在给出了控制流图中支配节点的定义性规范和相关性质定理后, 构造并证明了迭代求解算法所满足的重要性质. 利用这些性质定理, 相对于定义性规范, 证明了该迭代求解算法的正确性和完备性.

迭代计算是求解支配关系的一个主要方法, 另外两类主要方法是:

1) 基于复杂图变换和路径压缩的 LT 算法以及各种改进方法[14−17], 这类方法可以获得线性时间复杂度;
2) 直接的计算方法, 由非常早期的支配关系研究者们提出[18,19], 其基本思想是: 从节点 i 的一条路径中依次移除节点, 如果移除节点后 i 不可达, 则该节点是 i 的一个支配节点. 这类方法的时间复杂度与实际实现时所用的数据结构等非常相关.

鉴于 LT 算法的复杂性, Blazy 等人使用定理证明助手 Coq, 在 Compcert SSA (static single assignment)项目中使用后验的方式, 验证了 LT 算法的正确性[20]. 他们首先使用 Ocaml 实现了一个简化版的 LT 算法, 然后在 Coq 中编写了确认器(validator), 能够快速检查执行 LT 算法所得结果的正确性, 最后证明了确认器的正确性. Zhao 等人在 Vellvm 项目中也实现了一个验证的基于 SSA 的优化编译器[21], 他们使用了 Compcert 项目中的 Kildall 模块, 证明了 CHK 算法的正确性和完备性. Coq 的底层逻辑是非简单的依赖类型, Isabelle/HOL 是实现了 Church A——简单类型理论的通用证明助手, 两者在定义、定理的构造以及证明方式上非常不同. 因此, 本文的工作并不是 Zhao 等人研究工作的简单重复. 譬如: 他们在半格结构中使用了 option 类型, None 用来表示最小值, 其解释是这个设计导致了更好的执行性能; 在本文的实现过程中, 发现不需要提升(lift)到 option 类型, 最小值是包括所有节点的逆序列表, 并且, 未提升到 option 类型的证明更为流畅.

本文选择机械化证明 CHK 迭代算法的原因是:

1) 迭代算法基于格理论框架(lattice theoretic framework)[22−24], 使得能够在格理论的指导下, 系统性地机械化证明一个特定迭代算法的正确性和完备性;
2) 在采用 CHK 迭代算法对实际程序的控制流图进行求解时, 具有媲拟普遍认可为高效的 LT 算法的效率, 而机械化验证高效的求解支配节点的算法通常是获得一个实际的“验证编译器(verified compiler)”不可或缺的一部分;
3) 支配关系不仅用于程序优化, 还可用于诸如反编译、电路测试以及各类组件系统等, 而通过机械化验证建立这类应用的高可靠性也是非常有益的.

本文未来的研究工作包括利用支配节点识别和恢复循环, 证明一种不具有结构化控制指令的语言(譬如 JVML)翻译到具有结构化控制指令语言(譬如 WebAssembly)的正确性, 以及编译优化的正确性等.

References:

[1] Allen F, Cocke J. Graph-theoretic constructs for program flow analysis. Technical Report, RC 3923 (17789), IBM Thomas J. Watson Research Center, 1972.

[2] Cooper KD, Harvey TJ, Kennedy K. A simple, fast dominance algorithm. Technical Report, Rice University, 2006.

[3] Hecht MS, Ullman JD. Flow graph reducibility. SIAM Journal on Computing, 1972, 1: 188−202.

[4] Tarjan RE. Finding dominators in directed graphs. SIAM Journal on Computing, 1974, 3(1): 62−89.

[5] Lengauer T, Tarjan RE. A fast algorithm for finding dominators in a flow graph. ACM Trans. on Programming Languages and Systems, 1979, 1(1): 115−120.

[6] Boyle JM, Resler RD. Do you trust your compiler? Computer, 1999, 32(5): 65−73.

[7] Safonov VO. Trustworthy Compilers. John Wiley & Sons, 2010.

[8] He YX, Wu W. Theories and Technologies of Trusted Compiler Construction. Beijing: Science Press, 2013 (in Chinese).

[9] Liu Y, Gan YK, Wang SY, Dong Y, Yang F, Shi G, Yan X. Trustworthy translation for eliminating high-order operation of a synchronous dataflow language. Ruan Jian Xue Bao/Journal of Software, 2015, 26(2): 332−347 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4785.htm [doi: 10.13328/j.cnki.jos.004785]

[10] Shang S, Gan YK, Shi G, Wang SY, Dong Y. Key translations of the trustworthy compiler L2C and its design and implementation. Ruan Jian Xue Bao/Journal of Software, 2017, 28(5): 1233−1246 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5213.htm [doi: 10.13328/j.cnki.jos.005213]

[11] Jiang N, Li QA, Wang LM, Zhang XT, He YX. Overview on mechanized theorem proving. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 82−112 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5870.htm [doi: 10.13328/j.cnki.jos.005870]

[12] Nipkow T, Paulson LC, Wenzel M. Isabelle/HOL: A proof assistant for higher-order logic. Springer, 2002. http://isabelle.in.tum.de/doc/tutorial.pdf

[13] Jiang N. A data flow analysis algorithm for computing dominators. In: Proc. of the Archive of Formal Proofs. Formal Proof Development, 2021. https://isa-afp.org/entries/Dominance_CHK.html

[14] Georgiadis L, Tarjan RE. Dominator tree verification and vertex-disjoint paths. In: Proc. of the 16th Annual ACM-SIAM Symp. on Discrete Algorithms. Vancouver, 2005. 433−442.

[15] Georgiadis L, Werneck RF, Tarjan RE, August DI. Finding dominators in practice. In: Proc. of the 12th European Symp. on Algorithms. Bergen, 2004. 677−688.

[16] Georgiadis L, Laura L, Parotsidis N, Tarjan RE. Dominator certification and independent spanning trees: An experimental study. In: Proc. of the 12th Int'l Symp. on Experimental Algorithms. 2013. 284−295.

[17] Georgiadis L, Giannis K, Italiano GF, *et al*. Dynamic dominators and low-high orders in DAGs. In: Proc. of the 27th European Symp. on Algorithms. Munich, 2019. 50:1−50:18.

[18] Lowry ES, Medlock CW. Object code optimization. Communications of the ACM, 1969, 12(1): 13−22.

[19] Purdom JPW, Moore EF. Immediate predominators in a directed graph. Communications of the ACM, 1972, 15(8): 777−778.

[20] Blazy S, Demange D, Pichardie D. Validating dominator trees for a fast, verified dominance test. In: Proc. of the 6th Int'l Conf. on Interactive Theorem Proving. Nanjing, 2015. 84−95.

[21] Zhao J, Zdancewic S. Mechanized verification of computing dominators for formalizing compilers. In: Proc. of the 2nd Int'l Conf. on Certified Programs and Proofs. Kyoto, 2012. 27−42.

[22] Kildall GA. A unified approach to global program optimization. In: Proc. of the ACM Symp. on Principles of Programming Languages. New York, 1973. 194−206.

[23] Kam JB, Ullman JD. Global data flow analysis and iterative algorithms. Journal of the ACM, 1976, 23(1): 158−171.

[24] Klein G, Nipkow T. A machine-checked model for a Java-like language, virtual machine, and compiler. ACM Trans. on Programming Language and System, 2006, 28(4): 619−695.

附中文参考文献:

[8] 何炎祥, 吴伟. 可信编译构造理论与关键技术. 北京: 科学出版社, 2013.

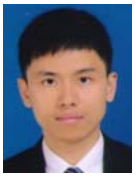
[9] 刘洋, 甘元科, 王生原, 董渊, 杨斐, 石刚, 闫鑫. 同步数据流语言高阶运算消去的可信翻译. 软件学报, 2015, 26(2): 332−347. http://www.jos.org.cn/1000-9825/4785.htm [doi: 10.13328/j.cnki.jos.004785]

[10] 尚书, 甘元科, 石刚, 王生原, 董渊. 可信编译器 L2C 的核心翻译步骤及其设计与实现. 软件学报, 2017, 28(5): 1233−1246. http://www.jos.org.cn/1000-9825/5213.htm [doi: 10.13328/j.cnki.jos.005213]

[11] 江南, 李清安, 汪吕蒙, 张晓瞳, 何炎祥. 机械化定理证明研究综述. 软件学报, 2020, 31(1): 82−112. http://www.jos.org.cn/1000-9825/5870.htm [doi: 10.13328/j.cnki.jos.005870]

江南(1976－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为可信软件, 机器证明, 编程语言.

张晓瞳(1989－), 男, 硕士, 主要研究领域为可信软件, 可信编译, 软件缺陷预测.



汪吕蒙(1988－), 男, 硕士, 主要研究领域为 CPU+GPU 异质系统架构, GPGPU 功耗优化.

何炎祥(1952－), 男, 博士, 教授, 博士生导师, CCF 会士, 主要研究领域为可信软件, 分布式并行处理, 高性能计算.