ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.8, August 2009, pp.2074−2086
doi: 10.3724/SP.J.1001.2009.03378

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

基于时序逻辑证明编译优化程序变换的保义性*

陶秋铭[1,2+], 赵 琛[1,3], 郭 亮[3]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[2](中国科学院 研究生院,北京 100049)

[3](中国科学院 软件研究所 基础软件国家工程研究中心,北京 100190)

Proving Soundness of Program Transformations in Optimizing Compilation Based on Temporal Logic

TAO Qiu-Ming[1,2+], ZHAO Chen[1,3], GUO Liang[3]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[3](National Engineering Research Center for Fundamental Software, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: E-mail: taoqiuming@itechs.iscas.ac.cn

Tao QM, Zhao C, Guo L. Proving soundness of program transformations in optimizing compilation based on temporal logic. *Journal of Software*, 2009,20(8):2074−2086. http://www.jos.org.cn/1000-9825/3378.htm

Abstract: Two kinds of program transformations widely-used in optimizing compilation, statement exchange and variable substitution, are investigated and their soundness conditions are formally defined with CTL-FV, an extension of the temporal logic CTL (computation tree logic). Sound statement exchange T_{exch} and sound variable substitution T_{sub} are defined with conditioned rewriting rules and their soundness is proved under an inductive proof frame. In addition, based on T_{exch}, the soundess of another transformation, dependence-preserving statement reordering inside basic blocks of programs, is also proved with a constructive method.

Key words: temporal logic; formal specification; optimizing compilation; program transformation; statement exchange; variable substitution; statement reordering

摘 要: 基于时序逻辑 CTL(computation tree logic)的一种扩展 CTL-FV 对优化编译中的语句交换和变量替换这两种常见变换的保义性条件给出了形式刻画,采用含条件重写规则定义了保义语句交换 T_{exch} 和保义变量替换 T_{sub},并基于一种归纳证明框架对它们的保义性进行了证明.此外,基于变换 T_{exch} 对程序基本块内保依赖语句重排的保义性也给出了一种构造性的证明.

关键词: 时序逻辑;形式规约;优化编译;程序变换;语句交换;变量替换;语句重排

中图法分类号: TP314 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60573164 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA010201 (国家高技术研究发展计划(863))

Received 2007-12-28; Revised 2008-03-11; Accepted 2008-04-30

编译优化通过对程序进行等价变换来改善程序的性能,是现代编译器不可缺少的重要功能.在过去 40 年中,人们开发出各种编译优化技术以减少程序的冗余计算,节省存储空间和能耗,或充分利用目标体系结构的计算潜力[1−3].随着编译优化在编译器中所占的比重逐渐增大,其正确性问题近年来也日益受到关注[4].

编译优化的正确性包含两层含义:一是保义性,即优化变换不改变程序的原有功能;二是有效性,即优化变换确实能够改善程序的性能.其中,保义性对于计算的安全性而言显得更为重要.近年来与此相关的研究有 3 个方面:(1) 在形式语义基础上分析和证明各种编译优化的保义性[5−11];(2) 对编译优化具体实现的保义性进行形式验证[12,13];(3) 对编译优化具体实现的保义性进行测试[14].其中,探求编译优化保义性的形式证明,不仅可以为编译优化的形式验证提供依据,而且可以为新的编译优化技术的保义性提供理论分析方法.

2002 年,Lacey 等人[5,6]采用一种归纳证明方法对过程式语言程序上多种经典优化[1]的保义性进行了形式证明.在他们的方法中,编译优化被表示为含条件重写规则:$\mathcal{I}\Rightarrow\mathcal{I}'$ If ϕ,其含义为:如果条件ϕ满足,则可实施变换$\mathcal{I}\Rightarrow\mathcal{I}'$.其中,$\phi$采用一种扩展的时序逻辑 CTL-FV 表达.除了文献[5,6]之外,过程式语言上优化转换保义性证明的相关研究还包括文献[10,11].这些研究对经典优化的保义性考察得较多,然而对于各种高级优化(指令调度、向量化、Cache 优化等[2,3])的保义性还未能给出严格证明.

在优化编译器中存在着这样一类程序变换,它们本身不是优化,然而它们在编译器的多个阶段以及各种高级优化中被大量使用.显然,对这类程序变换的保义性进行考察和证明也是必要的,并可为进一步探讨各种高级优化的保义性提供基础.在本文中,我们考察了这类变换中的两种变换——语句交换和变量替换,基于文献[5,6]中的方法框架定义了保义语句交换 T_{exch} 和保义变量替换 T_{sub},并证明了它们的保义性.此外,基于变换 T_{exch} 对程序基本块内保依赖语句重排 T_{dep} 的保义性也给出了一种构造性的证明.

本文第 1 节介绍文献[5,6]提出的程序变换保义性证明框架.第 2 节依次对 3 种保义变换 T_{exch},T_{dep} 和 T_{sub} 进行形式定义和保义性证明.第 3 节介绍相关研究,对若干问题进行讨论.第 4 节给出全文总结.

1 基础原理

1.1 一种简单的过程式语言

本文延续使用文献[5,6]提出的一种简单的过程式语言(本文中称作 SimLan).其程序具有如下形式:

0: **read** *x*; 1: *stmt*; 2: *stmt*; …; *n*−1: *stmt*; *n*: **write** *y*;

程序中的所有语句都有一个顺序标号,并且所有程序均以语句“**read** *x*”开始,以语句“**write** *y*”结束.其他语句可由如下文法来定义:

stmt::=*x*:=*exp*|**if** *x* **then** *label* **else** *label*|**skip**

exp::=*const*|*x*|$op_1(exp_1,\ldots,exp_{k_1})$|$op_2(exp_1,\ldots,exp_{k_2})$|$op_m(exp_1,\ldots,exp_{k_m})$

x::=*a*|*b*|*c*|…

label::=1|2|…|*n*

…

在上述文法中,*a*,*b*,*c*,…是程序可使用的所有变量,它们构成集合 $Variable$.$op_i(i=1,...,m)$是各种操作符,例如+,−,*,它们构成集合 Op.一个程序π中出现的所有变量的集合记作 $vars(\pi)$,出现的所有表达式的集合记作 $expr(\pi)$,一个表达式 *exp* 中出现的所有变量的集合记作 $vars(exp)$或 $FV(exp)$.

SimLan 语言所有程序的集合记为 *pgm*.下面通过一系列的定义给出程序的操作语义.首先假设已经给定一个值域 *Value*,以及 *Op* 中操作符的解释函数$[\![\cdot]\!]_{op}$:$Value^*\to Value$,并且假定 *Value* 域中包含一个特定值 True.

定义 1(程序存储). 一个程序存储是一个从变量到值的映射.程序存储的集合记作 $Store=Variable\rightharpoonup Value$.假设一个程序存储为$\sigma$:$X\to Value$,它的域 $dom(\sigma)=X\subseteq Variable$,有如下定义:

(i) 映射$\sigma[x\mapsto v]$:$X\cup\{x\}\to Value$ 称作σ到 $X\cup\{x\}$的扩展,有$\sigma[x\mapsto v](x)=v$ 并且$\forall y\in X\setminus\{x\}$:$\sigma[x\mapsto v](y)=\sigma(y)$;

(ii) 设 $X'\subseteq X$,映射$\sigma|_{X'}$:$X'\to Value$ 称作σ到 X'的约束,有$\forall x\in X'$:$\sigma|_{X'}(x)=\sigma(x)$.$\sigma|_{X\setminus\{x\}}$可简记为$\sigma\setminus\{x\}$.

定义 2(表达式估值). 一个表达式估值是一个函数$[\![\cdot]\!]\cdot:exp\to Store\to Value$,其递归定义如下:

(i) $[\![x]\!]\sigma=\sigma(x)$;

(ii) $[\![op(e_1,\ldots,e_n)]\!]\sigma=[\![op]\!]_{op}([\![e_1]\!]\sigma,\ldots,[\![e_n]\!]\sigma)$.

定义 3(程序状态). 对于给定的程序$\pi\in pgm$,一个程序状态是一个二元组(p,σ),其中,p 是π中的标号,σ是一个程序存储,有 $vars(\pi)\subseteq dom(\sigma)$.所有程序状态的集合记作 $PgmState$.对于输入值 $v\in Value$,初始程序状态为$In_\pi(v)=(1,\sigma)$,有$\sigma(x)=v$ 并且$\forall y\in var(\pi)\backslash\{x\}:\sigma(y)=$True.注意,这里我们假定在一个程序的初始状态时,初始指令 **read** 已经执行,所以变量 x 含值 v.

定义 4(语义迁移关系). 记标号 p 处的语句为$\mathcal{I}p$,程序$\pi\in pgm$ 最后的标号记为 $exit(\pi)$.程序的语义迁移关系 $\to\subseteq PgmState\times PgmState$ 定义如下:

(i) 如果$\mathcal{I}p=$(**skip**),那么$(p,\sigma)\to(p+1,\sigma)$成立;

(ii) 如果$\mathcal{I}p=(x:=exp)$,那么$(p,\sigma)\to(p+1,\sigma[x\mapsto[\![exp]\!]\sigma])$成立;

(iii) 如果$\mathcal{I}p=$(**if** x **then** p_1 **else** p_2),并且$\sigma(x)=$True,那么$(p,\sigma)\to(p_1,\sigma)$成立;

(iv) 如果$\mathcal{I}p=$(**if** x **then** p_1 **else** p_2),并且$\sigma(x)\neq$True,那么$(p,\sigma)\to(p_2,\sigma)$成立;

(v) $(exit(\pi),\sigma)\to(exit(\pi),\sigma)$成立.

其中,(v)表示程序若到达最后一条语句则不断循环迁移下去.作此定义可使程序等价性证明变得简便.

定义 5(计算前缀). 对于程序$\pi\in pgm$ 和输入值 $v\in Value$,计算前缀 C 是一个有限或无限的序列:

$$C=\pi,v\vdash(p_0,\sigma_0)\to(p_1,\sigma_1)\to(p_2,\sigma_2)\to\ldots$$

其中,$(p_i,\sigma_i)\in PgmState$,$(p_0,\sigma_0)=In_\pi(v)$,并且有$(p_i,\sigma_i)\to(p_{i+1},\sigma_{i+1})(i\geq 0)$.对于程序$\pi$和输入值 v,计算前缀的集合记作 $\mathcal{T}_{pfx}(\pi,v)$.注意,定义 4 中的(v)使得我们可以得到任意长度的计算前缀,即便程序执行已到达最后一句.

定义 6(语义函数). 对于程序$\pi\in pgm$,语义函数$[\![\pi]\!]:Value\rightharpoonup Value$ 是一个部分函数,其定义为:如果存在计算前缀"$\pi,v\vdash(p_0,\sigma_0)\to(p_1,\sigma_1)\to\ldots\to(p_k,\sigma_k)$"使得 $p_k=exit(\pi)$,那么$[\![\pi]\!](v)=\sigma_k(y)$,否则无定义.

1.2 CTL(computation tree logic)的一种扩展:CTL-FV

CTL-FV 是时序逻辑 CTL 的一种扩展,这里我们直接给出它的形式定义,详细解释参见文献[5,6].

定义 7(CTL-FV 模型). 给定原子命题公式集 AP,一个 CTL-FV 模型是一个三元组$\mathcal{M}=(S,\to,V)$,其中,S 是一个状态集合,$\to\subseteq S\times S$ 是 S 集合元素上的关系.函数 $V:S\to 2^{AP}$ 称作估值函数,它将任一状态映射到在该状态时为真的原子命题的集合.需要注意的是,为了表达程序中语句的特征,AP 中的元素可以具有谓词的形式,其变量为程序中的变量或表达式.以下定义本文中具体使用的 CTL-FV 模型.

定义 8(程序控制流模型). 给定程序$\pi\in pgm$,其控制流模型定义为$\mathcal{M}_{cf}(\pi)=(S,\to_{cf},V)$,其中:

(i) 状态集 S 定义为 $S=\{1,2,\ldots,exit(\pi)\}$.

(ii) 状态迁移关系$\to_{cf}:S\times S$ 定义如下:$p\to_{cf}p'$,如果有

$$(\mathcal{I}p=(\textbf{if } x \textbf{ then } p_1 \textbf{ else } p_2)\wedge p'\in\{p_1,p_2\})\vee((\mathcal{I}p=(\textbf{skip})\vee\mathcal{I}p=(x:=e))\wedge p'=p+1)\vee(\mathcal{I}p=(\textbf{write } y)\wedge p'=p).$$

(iii) 估值函数 $V:S\to 2^{AP}$ 定义为:

$node(q)\in V(p)$,如果 $p=q$;

$stmt(\mathcal{I})\in V(p)$,如果$\mathcal{I}p=\mathcal{I}$;

$def(x)\in V(p)$,如果$\exists e\in expr(\pi):\mathcal{I}p=(x:=e)$;

$use(z)\in V(p)$,如果$\exists x\in vars(\pi),\exists e\in expr(\pi),\exists p_1,p_2\in\{1,\ldots,exit(\pi)\}$

$$(\mathcal{I}p=(x:=e)\wedge z\in vars(e))\vee\mathcal{I}p=(\textbf{if } z \textbf{ then } p_1 \textbf{ else } p_2)\vee(\mathcal{I}p=(\textbf{write } y)\wedge y=z);$$

$trans(e)\in V(p)$,如果 $e\in expr(\pi)\wedge\forall x\in vars(e):def(x)\notin V(p)$.

定义 9(路径). CTL-FV 模型$\mathcal{M}=(S,\to,V)$上的一条路径$(n_i)_{i\geq 0}$是 S^*中的一个状态序列,它满足 $n_i\to n_{i+1}(i\geq 0)$(此时,该路径称为向前路径)或 $n_{i+1}\to n_i(i\geq 0)$(此时,该路径称为向后路径).一条路径称为最大路径,其含义为:该路径为无穷长或具有形式:$(n_i)_{0\leq i\leq k}$.其中,n_k 为结束点(如果$(n_i)_{0\leq i\leq k}$ 是一条向前路径,那么$\neg\exists n\in S:n_k\to n$;如果$(n_i)_{0\leq i\leq k}$ 是

一条向后路径,那么$\neg\exists n\in S:n\rightarrow n_k$).$S$上所有最大路径的集合记为$S^{MAX}$.注意,根据程序控制流模型的定义,该模型上的最大向前路径都是无穷长的.

假设对于程序$\pi\in pgm$ 和输入值 $v\in Value$,存在有穷计算前缀 $C\in\mathcal{T}_{pfx}(\pi,v):C=\pi,v\vdash(p_0,\sigma_0)\rightarrow(p_1,\sigma_1)\rightarrow\ldots\rightarrow(p_t,\sigma_t)$,可知,对于所有 $k(0\le k\le t)$有:

(i) $(n_i)_{0\le i\le t-k}=p_k\rightarrow_{cf}p_{k+1}\rightarrow_{cf}\ldots\rightarrow_{cf}p_t$(其中,$n_i=p_{i+k},0\le i\le t-k$)是$\mathcal{M}_{cf}(\pi)$中的一条向前路径;

(ii) $(n_i)_{0\le i\le k}=p_0\rightarrow_{cf}p_1\rightarrow_{cf}\ldots\rightarrow_{cf}p_k$(其中,$n_i=p_i,0\le i\le k$)是$\mathcal{M}_{cf}(\pi)$中的一条最大向后路径.

定义 10(CTL-FV 公式语法). CTL-FV 公式的语法定义如下:

ϕ::=true|false|$ap(x_1,\ldots,x_n)|\neg\phi|\phi_1\wedge\phi_2|\phi_1\vee\phi_2|\mathsf{A}\psi|\mathsf{E}\psi|\bar{\mathsf{A}}\ \psi|\bar{\mathsf{E}}\ \psi$,

ψ::=$\mathsf{X}\phi|\phi_1\mathsf{U}\phi_2|\phi_1\mathsf{W}\phi_2$.

定义 11(CTL-FV 公式满足性). CTL-FV 的子句表示为$\mathcal{M},n\vDash_\theta\phi$,其中,$\mathcal{M}$是 CTL-FV 模型(根据上下文可以省略),$n$ 是模型中的一个状态,ϕ是一个公式,θ 是一个解释.CTL-FV 公式的满足性定义如下:

$\mathcal{M},n\vDash_\theta$true,当且仅当 true;

$\mathcal{M},n\vDash_\theta ap(x_1,\ldots,x_n)$,当且仅当 $ap(\theta x_1,\ldots,\theta x_n)\in V(n)$;

$\mathcal{M},n\vDash_\theta$false,当且仅当 false;

$\mathcal{M},n\vDash_\theta\neg\phi$,当且仅当$\mathcal{M},n\vDash_\theta\phi$不满足;

$\mathcal{M},n\vDash_\theta\phi_1\wedge\phi_2$,当且仅当($\mathcal{M},n\vDash_\theta\phi_1$)并且($\mathcal{M},n\vDash_\theta\phi_2$);

$\mathcal{M},n\vDash_\theta\phi_1\vee\phi_2$,当且仅当($\mathcal{M},n\vDash_\theta\phi_1$)或者($\mathcal{M},n\vDash_\theta\phi_2$);

$\mathcal{M},n\vDash_\theta\mathsf{A}\psi$,当且仅当$(\forall(n=n_0\rightarrow n_1\rightarrow\ldots)\in S^{MAX}:\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\psi)$;

$\mathcal{M},n\vDash_\theta\bar{\mathsf{A}}\ \psi$,当且仅当$(\forall(\ldots\rightarrow n_1\rightarrow n_0=n)\in S^{MAX}:\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\psi)$;

$\mathcal{M},n\vDash_\theta\mathsf{E}\psi$,当且仅当$(\exists(n=n_0\rightarrow n_1\rightarrow\ldots)\in S^{MAX}:\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\psi)$;

$\mathcal{M},n\vDash_\theta\bar{\mathsf{E}}\ \psi$,当且仅当$(\exists(\ldots\rightarrow n_1\rightarrow n_0=n)\in S^{MAX}:\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\psi)$;

$\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\mathsf{X}\phi$,当且仅当($n_1$ 存在并且 $n_1\vDash_\theta\phi$);

$\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\phi_1\mathsf{U}\phi_2$,当且仅当$(\exists i\ge0:\mathcal{M},n_i\vDash_\theta\phi_2\wedge\forall0\le j<i:\mathcal{M},n_j\vDash_\theta\phi_1)$;

$\mathcal{M},(n_i)_{i\ge0}\vDash_\theta\phi_1\mathsf{W}\phi_2$,当且仅当$(\exists i\ge0:\mathcal{M},n_i\vDash_\theta\phi_2\wedge\forall0\le j<i:\mathcal{M},n_j\vDash_\theta\phi_1)$或者$(\forall k\ge0:n_k\vDash_\theta\phi_1$ 并且 n_{k+1} 存在).

1.3 程序变换及其保义性

定义 12(程序变换). 一个程序变换可以表示为具有如下形式的含条件重写规则:

$$p_1:\mathcal{I}p_1\Rightarrow\mathcal{I}'p_1;\ p_2:\mathcal{I}p_2\Rightarrow\mathcal{I}'p_2;\ \ldots;\ p_n:\mathcal{I}p_n\Rightarrow\mathcal{I}'p_n;\ \textbf{if}\ p_1\vDash\phi_1;\ p_2\vDash\phi_2;\ \ldots;\ p_n\vDash\phi_n,$$

其中,$p_i(i=1,\ldots,n)$是标号自由变量,$p_i:\mathcal{I}p_i\Rightarrow\mathcal{I}'p_i$ 表示将标号 p_i 位置处的语句$\mathcal{I}p_i$ 转换为语句$\mathcal{I}'p_i$,$p_i\vDash\phi_i(i=1,\ldots,n)$是实施变换的一组前提条件($\phi_i$ 为 CTL-FV 公式).当一个程序变换 T 实施于程序$\pi\in pgm$,可知变换所得的程序π'和原程序π具有相同的语句数,并且除了标号 $p_i(i=1,\ldots,n)$之外,所有其他标号处的语句都是相同的.

定义 13(变换实施关系). 设有程序$\pi,\pi'\in pgm$,$l_j(j=1,\ldots,n)$是程序π中的标号,程序π的形式为

$$0:\ \textbf{read}\ x;\ \ldots;\ l_1:\mathcal{I}l_1;\ \ldots l_2:\mathcal{I}l_2;\ \ldots;\ l_n:\mathcal{I}l_n;\ \ldots;\ m:\textbf{write}\ y.$$

设 T 是一个程序变换,其形式如定义 12 所描述.变换实施关系 $Apply(\pi,\pi',l_1,l_2,\ldots,l_n,T)$成立的条件为:存在$(l_1,l_2,\ldots,l_n)$的一个置换序列$(s_1,s_2,\ldots,s_n)$(设 $l_i=s_{k_i},i=1,\ldots,n$),使得$(p_1,p_2,\ldots,p_n)$被映射为$(s_1,s_2,\ldots,s_n)$时,存在某个解释θ,使得如下条件成立:

(i) CTL-FV 子句$\mathcal{M}_{cf}(\pi),s_i\vDash_\theta stmt(\mathcal{I}s_i)\wedge\phi_i(i=1,\ldots,n)$全部满足;

(ii) π'具有如下形式

$$0:\ \textbf{read}\ x;\ \ldots;\ l_1:\theta(\mathcal{I}'\,s_{k_1});\ \ldots;\ l_2:\theta(\mathcal{I}'\,s_{k_2});\ \ldots;\ l_n:\theta(\mathcal{I}'\,s_{k_n});\ \ldots;\ m:\textbf{write}\ y.$$

易知,如果 $Apply(\pi,\pi',l_1,\ldots,l_n,T)$成立,那么对程序$\pi$在标号 $l_1,\ldots,l_n$ 处实施程序变换 T 将得到程序π'.

定义 14(程序变换保义性). 两个程序$\pi_1,\pi_2\in pgm$ 等价的条件为$[\![\pi_1]\!]=[\![\pi_2]\!]$,即对于任意值 $v\in Value$,$[\![\pi_1]\!](v)$和$[\![\pi_2]\!](v)$均为无定义或者$[\![\pi_1]\!](v)=[\![\pi_2]\!](v)$.一个程序变换 T 是保义的,其含义为:对于任意一个程序$\pi\in pgm$,如果 T 能

够在π上实施并且得到程序π',那么一定有$[\![\pi]\!]=[\![\pi']\!]$.

以下是证明程序变换保义性的几个重要引理,其中,引理 1 给出了一种归纳证明框架.引理 1、引理 2、引理 4 的详细解释和证明可参见文献[5];引理 3、引理 5 的证明分别与引理 2、引理 4 类似,这里不再赘述.

引理 1. 设有程序$\pi\in pgm$和程序变换 T,并且 T 能够在π上实施得到程序π'.如果π的计算前缀和π'的计算前缀之间存在关系$\mathcal{R}$,使得对于任意输入值 $v\in Value$ 以及计算前缀 $C\in\mathcal{T}_{pfx}(\pi,v)$和 $C'\in\mathcal{T}_{pfx}(\pi',v)$,

$$C=\pi,v\vdash(p_0,\sigma_0)\to\ldots\to(p_t,\sigma_t);\ C'=\pi',v\vdash(p'_0,\sigma'_0)\to\ldots\to(p'_t,\sigma'_t).$$

若以下条件均成立,则π和π'等价(即$[\![\pi]\!]=[\![\pi']\!]$).

(i) 基始.$((\pi,v\vdash(p_0,\sigma_0)),(\pi',v\vdash(p'_0,\sigma'_0)))\in\mathcal{R}$.

(ii) 归纳.如果$(p_t,\sigma_t)\to(p_{t+1},\sigma_{t+1})$,并且$(p'_t,\sigma'_t)\to(p'_{t+1},\sigma'_{t+1})$,那么 $C\mathcal{R}C'\Rightarrow C_2\mathcal{R}C'_2$,其中:

$$C_2=\pi,v\vdash(p_0,\sigma_0)\to\ldots\to(p_t,\sigma_t)\to(p_{t+1},\sigma_{t+1});C'_2=\pi',v\vdash(p'_0,\sigma'_0)\to\ldots\to(p'_t,\sigma'_t)\to(p'_{t+1},\sigma'_{t+1}).$$

(iii) 等价性.如果 $C\mathcal{R}C'$,那么$(p_t=exit(\pi)\Leftrightarrow p'_t=exit(\pi'))$,并且$(p_t=exit(\pi)\wedge p'_t=exit(\pi')\Rightarrow\sigma_t(y)=\sigma'_t(y))$.

引理 2. 设$\pi,v\vdash(p_0,\sigma_0)\to\ldots\to(p_t,\sigma_t)$是一个计算前缀,并且如下条件成立,那么 $p_t\vDash\phi_1$.

(i) $p\vDash\mathsf{A}(\phi_1\mathsf{W}\phi_2)$或者 $p\vDash\mathsf{A}(\phi_1\mathsf{U}\phi_2)$;(ii) $\exists i{:}i<t\wedge p_i=p\wedge(\forall j{:}i\le j<t\Rightarrow p_j\vDash\neg\phi_2)$;(iii) $p_t\vDash\neg\phi_2$.

引理 3. 设$\pi,v\vdash(p_0,\sigma_0)\to\ldots\to(p_t,\sigma_t)$是一个计算前缀,并且如下条件成立,那么 $p_t\vDash\phi_1$.

(i) $p\vDash\mathsf{AXA}(\phi_1\mathsf{W}\phi_2)$或者 $p\vDash\mathsf{AXA}(\phi_1\mathsf{U}\phi_2)$;(ii) $\exists i{:}i<t\wedge p_i=p\wedge(\forall j{:}i<j<t\Rightarrow p_j\vDash\neg\phi_2)$;(iii) $p_t\vDash\neg\phi_2$.

引理 4. 设$\pi,v\vdash(p_0,\sigma_0)\to\ldots\to(p_t,\sigma_t)$是一个计算前缀,并且如下条件成立,那么$\mathcal{A}$成立.

(i) $p\vDash\bar{\mathsf{A}}(\phi_1\mathsf{W}\phi_2)$或者 $p\vDash\bar{\mathsf{A}}(\phi_1\mathsf{U}\phi_2)$;(ii) $(\exists i{:}i<t\wedge(p_i\vDash\phi_2)\wedge(\forall j{:}i\le j<t\Rightarrow p_j\vDash\phi_1))\Rightarrow\mathcal{A}$;(iii) $\exists i<t{:}p_i\vDash\neg\phi_1$; (iv) $p_t=p$.

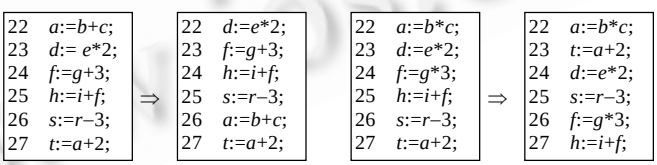
引理 5. 设$\pi,v\vdash(p_0,\sigma_0)\to\ldots\to(p_t,\sigma_t)$是一个计算前缀,并且如下条件成立,那么$\mathcal{A}$成立.

(i) $p\vDash\bar{\mathsf{A}}\mathsf{X}\bar{\mathsf{A}}(\phi_1\mathsf{W}\phi_2)$或者 $p\vDash\bar{\mathsf{A}}\mathsf{X}\bar{\mathsf{A}}(\phi_1\mathsf{U}\phi_2)$;(ii) $(\exists i{:}i<t\wedge(p_i\vDash\phi_2)\wedge(\forall j{:}i<j<t\Rightarrow p_j\vDash\phi_1))\Rightarrow\mathcal{A}$;(iii) $\exists i<t{:}p_i\vDash\neg\phi_1$; (iv) $p_t=p$.

2 3 种保义变换

2.1 保义语句交换T_{exch}

优化编译中,对语句位置进行重排是一种十分常见的程序变换.语句重排变换仅仅改变程序中语句执行次序而不增加或减少语句的任何执行.图 1 给出了两个典型的应用示例.示例 1 中,语句"a:=b+c"被移近至引用变量 a 的语句"t:=a+2"处,可以减轻对这段程序进行寄存器分配的压力[2,3].示例 2 中,含乘法运算的语句被散布到含加法运算的语句群中,这种变换在编译后端的软件流水、VLIW 指令编码等功能[2,3]中较为常见.



Example 1 Example 2

Fig.1 Two usage examples of statement reordering

图 1 语句重排的两个应用示例

语句重排的基本形式是将两条语句的位置互换,我们称其为语句交换.两条语句要能够互换并且保持原程序的语义,需要有一定的条件.下面我们定义保义语句交换 T_{exch} 并证明其保义性.本文第 2.2 节将基于 T_{exch} 的保义性证明程序基本块中保持依赖的语句重排变换的保义性.

定义 15(保义语句交换 T_{exch}). 保义语句交换 T_{exch} 用含条件重写规则表示如下:

p: u:=$e_1\Rightarrow v$:=e_2;

q: v:=$e_2\Rightarrow u$:=e_1;

if

$p\models\neg use(v)\wedge\neg def(v)\wedge AXA(\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)\cup node(q))$;

$q\models\neg use(u)\wedge\neg def(u)\wedge \bar{A}\ X\ \bar{A}\ (\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)\cup node(p))$.

该定义的变换条件部分包含 3 层直观含义:(1) p 语句中不出现变量 v 的引用和定值,q 语句中不出现变量 u 的引用和定值;(2) p 语句之后的所有向前路径一定会到达 q 语句,q 语句之前的所有向后路径也一定会到达 p 语句;(3) p 语句和 q 语句之间的所有路径区段上不出现对变量 u,v 的定值或引用,也不出现对表达式 e_1 和 e_2 中引用变量的定值.

注意,T_{exch} 的条件定义中,$p\models\neg def(v)$和 $q\models\neg def(u)$表明 $u\neq v$.实际上,当 $u=v$ 时,也存在 p 和 q 处语句可交换的情形,但需再追加条件:(a) $e_1=e_2$;或者(b) q 语句之后的所有路径上在出现对 u 重新定值的语句之前都没有对 u 的引用. (a)条件下,p 和 q 处语句完全相同,显然可交换;(b)条件下,p 语句和 q 语句实际上都是无用代码[1],此时可以借助“无用代码删除”变换[1]来证明 p,q 可交换.文献[5]已经考察和刻画了无用代码的特征,并证明了“无用代码删除”变换的保义性.这里为了重点关注语句交换本身的特征,故在 T_{exch} 的条件定义中排除了 $u=v$ 的情形.

定理 1. T_{exch} 是保义的,即对于程序 $\pi\in pgm$,如果 T_{exch} 能够在 π 上实施并得到程序 π',那么一定有 $[\![\pi]\!]=[\![\pi']\!]$.

证明:设对于程序 $\pi\in pgm, T_{exch}$ 能够在 π 上实施并且得到程序 π'.再设对于任意输入值 $v\in Value$,有 $C\in\mathcal{T}_{pfx}(\pi,v), C'\in\mathcal{T}_{pfx}(\pi',v)$:

$$C=\pi,v\vdash(p_0,\sigma_0)\rightarrow\ldots\rightarrow(p_t,\sigma_t);\ C'=\pi',v\vdash(p'_0,\sigma'_0)\rightarrow\ldots\rightarrow(p'_t,\sigma'_t).$$

下面定义 $\mathcal{R}$ 关系.设 $\phi_1\triangleq\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)$,定义 $\mathcal{R}$ 关系如下:如果 $C\mathcal{R}_1C'$,并且 $C\mathcal{R}_2C'$,那么 $C\mathcal{R}C'$成立,其中,$\mathcal{R}_1$ 和 $\mathcal{R}_2$ 分别定义如下:

(i) $C\mathcal{R}_1C'$,如果 $p_t=p'_t$,并且以下任意一个条件成立:

1. $\sigma_t=\sigma'_t$;
2. $\exists i{:}i<t\wedge p_i=p\wedge(\forall j{:}i<j<t\Rightarrow p_j\models\neg node(q))\wedge\sigma_t\backslash\{u,v\}=\sigma'_t\backslash\{u,v\}$.

(ii) $C\mathcal{R}_2C'$,如果有$(\exists i{:}i<t\wedge(p_i\models node(p))\wedge(\forall j{:}i<j<t\Rightarrow p_j\models\phi_1))\Rightarrow(\sigma_t(u)=[\![e_1]\!]_{exp}\sigma'_t\wedge\sigma'_t(v)=[\![e_2]\!]_{exp}\sigma_t)$.

下面基于引理 1 证明 $[\![\pi]\!]=[\![\pi']\!]$.

(I) 基始情形:若 $C=\pi,v\vdash(p_0,\sigma_0), C'=\pi',v\vdash(p'_0,\sigma'_0)$,易知 $C\mathcal{R}_1C'$和 $C\mathcal{R}_2C'$都成立,故 $C\mathcal{R}C'$成立.

(II) 归纳情形:假设 $C_1\mathcal{R}C'_1$,其中:

$$C_1=\pi,v\vdash(p_0,\sigma_0)\rightarrow\ldots\rightarrow(p_t,\sigma_t);\ \ C'_1=\pi',v\vdash(p'_0,\sigma'_0)\rightarrow\ldots\rightarrow(p'_t,\sigma'_t).$$

根据程序语义迁移关系的定义可知,一定存在唯一的状态(p_{t+1},σ_{t+1})和 (p'_{t+1},σ'_{t+1}),使得$(p_t,\sigma_t)\rightarrow(p_{t+1},\sigma_{t+1})$, $(p'_t,\sigma'_t)\rightarrow(p'_{t+1},\sigma'_{t+1})$,如此分别得到:

$$C_2=\pi,v\vdash(p_0,\sigma_0)\rightarrow\ldots\rightarrow(p_t,\sigma_t)\rightarrow(p_{t+1},\sigma_{t+1});\ \ C'_2=\pi',v\vdash(p'_0,\sigma'_0)\rightarrow\ldots\rightarrow(p'_t,\sigma'_t)\rightarrow(p'_{t+1},\sigma'_{t+1}).$$

下面分别证明 $C_2\mathcal{R}_1C'_2$ 和 $C_2\mathcal{R}_2C'_2$.

(1) $C_2\mathcal{R}_1C'_2$. 共分 4 种情形加以证明.首先根据 p_t 可能的位置分为 3 类情形:① $p_t=p$;② $p_t=q$;③ $p_t\neq p$ 且 $p_t\neq q$.对于情形③又分两种情形,即 $C_1\mathcal{R}_1C'_1$ 成立的两种条件情形.

情形 1:$p_t=p$.此时有:

$C_1\mathcal{R}C'_1\wedge\mathcal{I}p_t=(u:=e_1)\wedge\mathcal{I}'\ p'_t=(v:=e_2)\Rightarrow C_1\mathcal{R}_1C'_1\wedge\mathcal{I}p_t=(u:=e_1)\wedge\mathcal{I}'\ p'_t=(v:=e_2)$

$\Rightarrow${在 $\mathcal{R}_1$ 定义的两种情形中,$\sigma_t\backslash\{u,v\}=\sigma'_t\backslash\{u,v\}$都成立}

$$\sigma_t\backslash\{u,v\}=\sigma'_t\backslash\{u,v\}\wedge\mathcal{I}p_t=(u:=e_1)\wedge\mathcal{I}'\ p'_t=(v:=e_2)\wedge p_t=p'_t$$

$\Rightarrow${根据程序语义迁移关系以及程序存储的性质}

$$\sigma_t\backslash\{u,v\}=\sigma'_t\backslash\{u,v\}\wedge\sigma_{t+1}\backslash\{u\}=\sigma_t\backslash\{u\}\wedge\sigma'_{t+1}\backslash\{v\}=\sigma'_t\backslash\{v\}\wedge p_{t+1}=p'_{t+1}=p_t+1$$

$\Rightarrow\sigma_{t+1}\backslash\{u,v\}=\sigma'_{t+1}\backslash\{u,v\}\wedge p_{t+1}=p'_{t+1}$

$\Rightarrow$ {$p_t=p\neq q$} $(\exists i{:}i<t+1\wedge p_i=p\wedge(\forall j{:}i<j<t+1\Rightarrow p_j\models\neg node(q)))\wedge\sigma_{t+1}\backslash\{u,v\}=\sigma'_{t+1}\backslash\{u,v\}\wedge p_{t+1}=p'_{t+1}$

$\Rightarrow C_2\mathcal{R}_1C'_2$ {$\mathcal{R}_1$ 定义的第 2 种情形}

情形 2:$p_t=q$.此时有:

$C_1\mathcal{R}\,C_1' \wedge \mathcal{I}p_t=(v:=e_2)\wedge\mathcal{I}'\ p_t' =(u:=e_1)\Rightarrow C_1\mathcal{R}_1\,C_1' \wedge C_1\mathcal{R}_2\,C_1' \wedge \mathcal{I}p_t=(v:=e_2)\wedge\mathcal{I}'\ p_t' =(u:=e_1)$

$\Rightarrow${在$\mathcal{R}_1$定义的两种情形中,$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}$都成立}

$$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}\wedge C_1\mathcal{R}_2\,C_1' \wedge \mathcal{I}p_t=(v:=e_2)\wedge\mathcal{I}'\ p_t' =(u:=e_1)\wedge p_t= p_t'$$

$\Rightarrow${因为:(i) $q\models \bar{A}\ X\ \bar{A}\ (\phi_1 U node(p))$;(ii) $(\exists i:i<t\wedge(p_i\models node(p))\wedge(\forall j:i<j<t\Rightarrow p_j\models\phi_1))\Rightarrow\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t(\mathcal{R}_2$定义);(iii) $\exists i<t:p_i\models\neg\phi_1$(根据 $q\models \bar{A}\ X\ \bar{A}\ (\phi_1 U node(p))$可知,在 C_1 中,每个 q 的前面一定会出现 p,又易知 $p\models\neg\phi_1$ 成立);(iv) $p_t=q$.根据引理 5 可知,$\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t$}

$$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}\wedge\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t\wedge\mathcal{I}p_t=(v:=e_2)\wedge\mathcal{I}'\ p_t' =(u:=e_1)\wedge p_t= p_t'$$

$\Rightarrow${根据程序语义迁移关系定义}

$$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}\wedge\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t\wedge\sigma_{t+1}=\sigma_t[v\mapsto[\![e_2]\!]_{exp}\sigma_t]\wedge$$
$$\sigma_{t+1}'=\sigma_t'[u\mapsto[\![e_1]\!]_{exp}\,\sigma_t']\wedge p_{t+1}= p_{t+1}'=p_t+1$$

$\Rightarrow${由$\sigma_{t+1}=\sigma_t[v\mapsto[\![e_2]\!]_{exp}\sigma_t]$可得$\sigma_{t+1}\backslash\{v\}=\sigma_t\backslash\{v\}$和$\sigma_{t+1}\backslash\{u\}=\sigma_t[v\mapsto[\![e_2]\!]_{exp}\sigma_t]\backslash\{u\}$;由$\sigma_{t+1}'=\sigma_t'[u\mapsto[\![e_1]\!]_{exp}\,\sigma_t']$可得$\sigma_{t+1}'\backslash\{u\}=\sigma_t'\backslash\{u\}$和$\sigma_{t+1}'\backslash\{v\}=\sigma_t'[u\mapsto[\![e_1]\!]_{exp}\,\sigma_t']\backslash\{v\}$;由$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}$和$\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t$可得$\sigma_t'\backslash\{u\}=\sigma_t[v\mapsto[\![e_2]\!]_{exp}\sigma_t]\backslash\{u\}$;由$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}$和$\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'$可得$\sigma_t\backslash\{v\}=\sigma_t'[u\mapsto[\![e_1]\!]_{exp}\,\sigma_t']\backslash\{v\}$}

$$\sigma_{t+1}'\backslash\{u\}=\sigma_t'\backslash\{u\}\wedge\sigma_t'\backslash\{u\}=\sigma_t[v\mapsto[\![e_2]\!]_{exp}\sigma_t]\backslash\{u\}\wedge\sigma_{t+1}\backslash\{u\}=\sigma_t[v\mapsto[\![e_2]\!]_{exp}\sigma_t]\backslash\{u\}\wedge\sigma_{t+1}\backslash\{v\}=\sigma_t\backslash\{v\}\wedge$$
$$\sigma_t\backslash\{v\}=\sigma_t'[u\mapsto[\![e_1]\!]_{exp}\,\sigma_t']\backslash\{v\}\wedge\sigma_{t+1}'\backslash\{v\}=\sigma_t'[u\mapsto[\![e_1]\!]_{exp}\,\sigma_t']\backslash\{v\}\wedge p_{t+1}= p_{t+1}'$$

$\Rightarrow\sigma_{t+1}'\backslash\{u\}=\sigma_{t+1}\backslash\{u\}\wedge\sigma_{t+1}'\backslash\{v\}=\sigma_{t+1}\backslash\{v\}\wedge p_{t+1}= p_{t+1}'$

$\Rightarrow${如前所述,$u\neq v$,因此根据 $\sigma_{t+1}'\backslash\{u\}=\sigma_{t+1}\backslash\{u\}\wedge\sigma_{t+1}'\backslash\{v\}=\sigma_{t+1}\backslash\{v\}$可得 $\sigma_{t+1}'=\sigma_{t+1}$} $\sigma_{t+1}'=\sigma_{t+1}\wedge p_{t+1}= p_{t+1}'$

$\Rightarrow C_2\mathcal{R}_1\,C_2'$ {$\mathcal{R}_1$定义的第 1 种情形}

情形 3:$p_t\neq p,p_t\neq q,C_1\mathcal{R}_1\,C_1'$成立的条件 1 满足(即$\sigma_t=\sigma_t'$).此时有:

$C_1\mathcal{R}\,C_1' \wedge p_t\neq p\wedge p_t\neq q\wedge p_t= p_t' \wedge\sigma_t=\sigma_t'$

$\Rightarrow p_t= p_t' \wedge\mathcal{I}p_t=\mathcal{I}'\ p_t' \wedge\sigma_t=\sigma_t'$ {根据 $p_t\neq p\wedge p_t\neq q\wedge p_t= p_t'$ 可知$\mathcal{I}p_t=\mathcal{I}'\ p_t'$ }

$\Rightarrow p_{t+1}= p_{t+1}'\wedge\sigma_{t+1}'=\sigma_{t+1}\Rightarrow C_2\mathcal{R}_1\,C_2'$ {$\mathcal{R}_1$定义的第 1 种情形}

情形 4:$p_t\neq p,p_t\neq q,C_1\mathcal{R}_1\,C_1'$成立的条件 2 满足.此时有:

$C_1\mathcal{R}\,C_1' \wedge p_t\neq p\wedge p_t\neq q\wedge p_t= p_t' \wedge(\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\models\neg node(q))\wedge\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\})$

$\Rightarrow${因为:(i) $p\models AXA(\phi_1 U node(q))$;(ii) $(\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\models\neg node(q))$;(iii) $p_t\models\neg node(q)$,由引理 3 可知 $p_t\models\phi_1$,即 $p_t\models\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)$;此外,根据 $p_t\neq p\wedge p_t\neq q\wedge p_t= p_t'$ 可得 $\mathcal{I}p_t=\mathcal{I}'\ p_t'$ }

$$(p_t\models\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1))\wedge\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}\wedge p_t= p_t'\wedge$$
$$\mathcal{I}p_t=\mathcal{I}'\ p_t'\wedge(\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\models\neg node(q)))\wedge p_t\neq q$$

$\Rightarrow\sigma_{t+1}\backslash\{u,v\}=\sigma_{t+1}'\backslash\{u,v\}\wedge p_{t+1}= p_{t+1}'\wedge(\exists i:i<t+1\wedge p_i=p\wedge(\forall j:i<j<t+1\Rightarrow p_j\models\neg node(q)))$

$\Rightarrow C_2\mathcal{R}_1\,C_2'$ {$\mathcal{R}_1$定义的第 2 种情形}

(2) $C_2\mathcal{R}_2\,C_2'$.分 5 种情形讨论.

情形 1:$p_t=p,C_1\mathcal{R}_1\,C_1'$成立的条件 1 满足(即$\sigma_t=\sigma_t'$).

由$\mathcal{I}q=(v:=e_2)\wedge q\models\neg use(u)$可知 $u\notin FV(e_2)$,又由$\mathcal{I}p_t=\mathcal{I}p=(u:=e_1)$可知$\sigma_{t+1}\backslash\{u\}=\sigma_t\backslash\{u\}$,因此有:

$$\sigma_{t+1}'(v)=\sigma_t'[v\mapsto[\![e_2]\!]_{exp}\,\sigma_t'](v)\{\text{因为 } p_t' =p_t=p,\mathcal{I}'\ p_t' =\mathcal{I}'p=(v:=e_2)\}$$
$$=[\![e_2]\!]_{exp}\,\sigma_t' =[\![e_2]\!]_{exp}\sigma_t\{\text{因为 }\sigma_t=\sigma_t'\}$$
$$=[\![e_2]\!]_{exp}\sigma_{t+1}\{\sigma_{t+1}\backslash\{u\}=\sigma_t\backslash\{u\};u\notin FV(e_2)\}$$

同样地,可知$\sigma_{t+1}(u)=[\![e_1]\!]_{exp}\,\sigma_{t+1}'$也成立.

易知$\sigma_{t+1}(u)=[\![e_1]\!]_{exp}\,\sigma_{t+1}'\wedge\sigma_{t+1}'(v)=[\![e_2]\!]_{exp}\sigma_{t+1}\Rightarrow C_2\mathcal{R}_2\,C_2'$ {根据$\mathcal{R}_2$定义}

情形 2:$p_t=p, C_1\mathcal{R}_1 C_1'$ 成立的条件 2 满足.此时有:

$p_t=p\wedge(\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\vDash\neg node(q)))\wedge\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}$

$\Rightarrow p_t=p\wedge(\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\vDash\neg node(q))\wedge p_t\vDash\neg node(q)$

$\Rightarrow$\{根据引理 3 可知 $p_t\vDash\phi_1$,推理过程类似 $C_2\mathcal{R}_1 C_2'$ 证明情形 4 中的推理过程\}$(p_t\vDash\neg\phi_1)\wedge(p_t\vDash\phi_1)$

$\Rightarrow$FALSE

这表明情形 2 不可能存在.

情形 3:$p_t=q$.

由$\mathcal{I}p_t=\mathcal{I}q=(v:=e_2)$可知 $p_t\vDash\neg\phi_1(\phi_1=\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1))$.

此时有如下推理:

$\exists i:i<t+1\wedge(p_i\vDash node(p))\wedge(\forall j:i<j<t+1\Rightarrow p_j\vDash\phi_1)\Rightarrow p_t\vDash\phi_1$\{由 $p_t\neq p$ 可知 $i<t$\}

$\Rightarrow$\{又因为 $p_t\vDash\neg\phi_1$\} FALSE $\Rightarrow\sigma_{t+1}(u)=[\![e_1]\!]_{exp}\,\sigma'_{t+1}\wedge\sigma'_{t+1}(v)=[\![e_2]\!]_{exp}\sigma_{t+1}$

由$\mathcal{R}_2$ 定义可知 $C_2\mathcal{R}_2 C_2'$ 成立.

情形 4:$p_t\neq p, p_t\neq q, p_t\vDash\neg\phi_1$.与情形 3 类似,可得结论:

$$(\exists i:i<t+1\wedge(p_i\vDash node(p))\wedge(\forall j:i<j<t+1\Rightarrow p_j\vDash\phi_1))\Rightarrow(\sigma_{t+1}(u)=[\![e_1]\!]_{exp}\,\sigma'_{t+1}\wedge\sigma'_{t+1}(v)=[\![e_2]\!]_{exp}\sigma_{t+1})$$

由$\mathcal{R}_2$ 定义可知 $C_2\mathcal{R}_2 C_2'$ 成立.

情形 5:$p_t\neq p, p_t\neq q, p_t\vDash\phi_1$.此时有如下推理:

$\exists i:i<t+1\wedge(p_i\vDash node(p))\wedge(\forall j:i<j<t+1\Rightarrow p_j\vDash\phi_1)$

$\Rightarrow$\{由 $p_t\neq p$ 可知 $i<t$\} $\exists i:i<t\wedge p_i\vDash node(p)\wedge(\forall j:i<j<t\Rightarrow p_j\vDash\phi_1)$

$\Rightarrow$\{$C_1\mathcal{R}_2 C_1'$\} $\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t$

$\Rightarrow$\{因为 $p_t\vDash\phi_1,\phi_1=\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)$,又有$\mathcal{I}p_t=\mathcal{I}'\,p_t'$(根据 $p_t\neq p, p_t\neq q$ 和 $p_t=p_t'$ (因为 $C_1\mathcal{R}_1 C_1'$)),所以也有 $p_t'\vDash\phi_1$.由 $p_t\vDash\neg def(u)$可得$\sigma_{t+1}(u)=\sigma_t(u)$;由 $p_t\vDash trans(e_2)$可得$[\![e_2]\!]_{exp}\sigma_{t+1}=[\![e_2]\!]_{exp}\sigma_t$;类似可得 $\sigma'_{t+1}(v)=\sigma_t'(v)$和$[\![e_1]\!]_{exp}\,\sigma'_{t+1}=[\![e_1]\!]_{exp}\,\sigma_t'$\}

$\sigma_t(u)=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma_t'(v)=[\![e_2]\!]_{exp}\sigma_t\wedge\sigma_{t+1}(u)=\sigma_t(u)\wedge[\![e_1]\!]_{exp}\,\sigma'_{t+1}=[\![e_1]\!]_{exp}\,\sigma_t'\wedge\sigma'_{t+1}(v)=\sigma_t'(v)\wedge[\![e_2]\!]_{exp}\sigma_{t+1}=[\![e_2]\!]_{exp}\sigma_t$

$\Rightarrow\sigma_{t+1}(u)=[\![e_1]\!]_{exp}\,\sigma'_{t+1}\wedge\sigma'_{t+1}(v)=[\![e_2]\!]_{exp}\sigma_{t+1}$

由$\mathcal{R}_2$ 定义可知 $C_2\mathcal{R}_2 C_2'$ 成立.

综上所述,在所有可能的情形下,$C_2\mathcal{R}_1 C_2'$ 和 $C_2\mathcal{R}_2 C_2'$ 均成立,因此 $C_2\mathcal{R} C_2'$ 成立.

(III) 等价性:如果 $C\mathcal{R}C'$,那么 $C\mathcal{R}_1C'$成立.根据$\mathcal{R}_1$ 关系的定义可知 $p_t=p_t'$,又因为 $exit(\pi)=exit(\pi')$,所以必然有 $p_t=exit(\pi)\Leftrightarrow p_t'=exit(\pi')$.此外,当 $p_t=exit(\pi), p_t'=exit(\pi')$时,根据$\mathcal{R}_1$ 关系的定义,要么$\sigma_t=\sigma_t'$ 成立,此时显然有$\sigma_t(y)=\sigma_t'(y)$;要么如下条件成立:(a) $\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\vDash\neg node(q))$;(b) $\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}$.在此情形下,由$\mathcal{I}p_t=($**write** $y)$得 $p_t\vDash\neg node(q)$.因为:(i) $p\vDash\mathrm{AXA}(\phi_1\mathrm{U}node(q))$;(ii) $(\exists i:i<t\wedge p_i=p\wedge(\forall j:i<j<t\Rightarrow p_j\vDash\neg node(q))$;(iii) $p_t\vDash\neg node(q)$,根据引理 3 可得 $p_t\vDash\phi_1$,即 $p_t\vDash\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)$.由此可知,$y\neq u$ 且 $y\neq v$.又因为$\sigma_t\backslash\{u,v\}=\sigma_t'\backslash\{u,v\}$,所以$\sigma_t(y)=\sigma_t'(y)$.

综上,根据引理 1 可知$[\![\pi]\!]=[\![\pi']\!]$. □

2.2 保依赖重排T_{dep}

如第 2.1 节所述,语句重排在寄存器分配优化、软件流水等多种优化中有着广泛的应用.任何语句重排执行之前都必须分析语句之间的数据依赖关系[2,3].关于语句重排有一个重要的定理,即任意一个语句重排,若能够保持原程序中语句之间的数据依赖关系,那么该语句重排变换是保义的[2,3].该定理是软件流水、向量化、Cache 优化等高级优化[2,3]的基础.本节主要证明程序基本块范围内保持数据依赖语句重排的保义性.在后续工作中,我们准备将数组变量扩充到 SimLan 语言中,以便进一步考察循环体中语句执行实例上的保依赖重排的保义性.

定义 16(数据依赖关系). 设程序π在标号 p 和 q 处分别有语句 S_1 和 S_2,分别为 $u:=e_1$ 和 $v:=e_2$.如果有

$(u\in vars(e_2)\wedge(p\models \mathsf{EXE}(\neg def(u)\mathsf{U}node(q))))\vee(v\in vars(e_1)\wedge(p\models \bar{\mathsf{E}}\,\mathsf{X}\,\bar{\mathsf{E}}\,(\neg def(v)\mathsf{U}node(q))))\vee$
$(u=v\wedge(p\models \mathsf{EXE}(\neg def(u)\mathsf{U}node(q)))),$

则称语句 S_1 和 S_2 之间存在数据依赖关系 δ,记为 $S_1\delta S_2$.

定义 17(基本块). 对于程序 $\pi\in pgm$,其中包含赋值语句 $S_1,S_2,\ldots,S_n$,设它们的标号分别为 $p_1,p_2,\ldots,p_n$,如果有 $p_i\models \mathsf{AX}node(p_{i+1})$($i$=1,2,…,$n$−1)并且 $p_i\models \bar{\mathsf{A}}\,\mathsf{X}node(p_{i-1})$($i$=2,3,…,$n$),那么 $S_1,S_2,\ldots,S_n$ 构成一个基本块,可以用矩阵记为 $[S_1,S_2,\ldots,S_n]^T$.

引理 6. 设程序 π 含基本块 $B=[S_1,S_2,\ldots,S_n]^T$,对 B 中前后相邻的两条赋值语句 S_t 和 S_{t+1}($t\in\{1,2,\ldots,n-1\}$),设它们分别为 $u:=e_1$ 和 $v:=e_2$,如果有 $\neg(S_t\delta S_{t+1})$,那么对 S_t 与 S_{t+1} 可实施保义语句交换 T_{exch},并且交换之后所得基本块 $B'(=[S_1,\ldots,S_{t+1},S_t,\ldots,S_n]^T)$ 对 B 上的依赖关系依然保持,即如果 δ 和 δ' 分别为 B 和 B' 上的依赖关系,那么有:

$$S_i\delta S_j\Leftrightarrow S_i\delta' S_j(i,j\in\{1,2,\ldots,n\}).$$

证明:S_t 和 S_{t+1} 前后相邻,设其标号分别为 p 和 q,有 $p\models \mathsf{AX}node(q)$,$q\models \bar{\mathsf{A}}\,\mathsf{X}node(p)$,根据 CTL-FV 公式的满足性,易知如下结论成立:

(i) $p\models \mathsf{AXA}(\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)\mathsf{U}node(q))$;

(ii) $q\models \bar{\mathsf{A}}\,\mathsf{X}\,\bar{\mathsf{A}}\,(\neg use(v)\wedge\neg def(v)\wedge trans(e_2)\wedge\neg use(u)\wedge\neg def(u)\wedge trans(e_1)\mathsf{U}node(p))$;

又因为 $\neg(S_t\delta S_{t+1})$,可知:

(iii) $p\models\neg use(v)\wedge\neg def(v)$,并且 $q\models\neg use(u)\wedge\neg def(u)$.

由语句交换变换 T_{exch} 的定义,综合结论(i)~(iii)可知,对 S_t 与 S_{t+1} 可实施保义语句交换 T_{exch},变换前后程序等价.

下面证明对 S_t 与 S_{t+1} 实施 T_{exch} 变换后,有 $S_i\delta S_j\Leftrightarrow S_i\delta' S_j(i,j\in\{1,2,\ldots,n\})$.其中,$\delta,\delta'$ 分别为 B,B' 上的依赖关系.

若在 B 上 $S_i\delta S_j$ 成立,易知有 $i<j$;又易知 $\neg(S_t\delta S_{t+1})\Rightarrow\neg(S_{t+1}\delta' S_t)$,进而可知,若在 B' 上 $S_i\delta' S_j$ 成立,也有 $i<j$.由此可知,若 $S_i\delta S_j$ 或 $S_i\delta' S_j$ 成立,均有 $i<j$.以下分情形讨论:

(1) $i<j<t$ 或 $t+1<i<j$.此时,基本块 B 中 S_i 和 S_j 之间的语句(包括 S_i 和 S_j)在实施 T_{exch} 变换前后没有发生任何次序的改变,易知此时 $S_i\delta S_j\Leftrightarrow S_i\delta' S_j$ 成立;

(2) $i<t,j=t$.此时 $S_j=S_t$(为 $u:=e_1$),设 S_i 为 $w:=e_3$,其标号为 r;

(a) 如果 $S_i\delta S_t$,那么有:① $(w\in vars(e_1)\wedge(r\models \mathsf{EXE}(\neg def(w)\mathsf{U}node(p))))$ 或② $(u\in vars(e_3)\wedge(p\models \bar{\mathsf{E}}\,\mathsf{X}\,\bar{\mathsf{E}}\,(\neg def(u)\mathsf{U}node(r))))$,或③ $(u=w\wedge(r\models \mathsf{EXE}(\neg def(u)\mathsf{U}node(p))))$.

若情形①成立,可知 $w\neq v$,否则 $v\in vars(e_1)$ 导致 $S_t\delta S_{t+1}$ 成立,矛盾.在 B' 中,S_t 和 S_{t+1} 的标号变为 q 和 p,S_i 和 S_t 之间的路径上增加了语句 S_{t+1}(为 $v:=e_2$),而 $w\neq v$,易知在 B' 中 $r\models \mathsf{EXE}(\neg def(w)\mathsf{U}node(q))$ 成立,又知 $w\in vars(e_1)$ 不变,所以有 $S_i\delta' S_t$ 成立.若情形②成立,S_i 和 S_t 之间的路径上增加了语句 S_{t+1}(为 $v:=e_2$),而 $u\neq v$(否则 $S_t\delta S_{t+1}$ 成立),则在 B' 中 $q\models \bar{\mathsf{E}}\,\mathsf{X}\,\bar{\mathsf{E}}\,(\neg def(u)\mathsf{U}node(r))$ 成立,又知 $u\in vars(e_3)$ 不变,所以有 $S_i\delta' S_t$ 成立.若情形③成立,其情形与情形②类似,亦有 $S_i\delta' S_t$ 成立.综上所述,$S_i\delta S_t\Rightarrow S_i\delta' S_t$ 成立.

(b) 如果在 B' 中有 $S_i\delta' S_t$,因为与在 B' 中相比,在 B 中 S_i 和 S_t 之间的路径上仅是少了一句 S_{t+1},其他不变,根据数据依赖关系定义易知,在 B 中必有 $S_i\delta S_t$,所以 $S_i\delta' S_t\Rightarrow S_i\delta S_t$.

综合情形(a)、情形(b)可知,此时 $S_i\delta S_j\Leftrightarrow S_i\delta' S_j$ 成立.

(3) $i<t,j>t+1$.设 S_i 和 S_j 分别为 $w:=e_3$ 和 $h:=e_4$,其标号分别为 r 和 g.

(a) 如果 $S_i\delta S_j$,那么有:① $(w\in vars(e_4)\wedge(r\models \mathsf{EXE}(\neg def(w)\mathsf{U}node(g))))$ 或② $(h\in vars(e_3)\wedge(g\models \bar{\mathsf{E}}\,\mathsf{X}\,\bar{\mathsf{E}}\,(\neg def(h)\mathsf{U}node(r))))$ 或③ $(w=h\wedge(r\models \mathsf{EXE}(\neg def(w)\mathsf{U}node(g))))$.

若情形①成立,那么 $r\models \mathsf{EXE}(\neg def(w)\mathsf{U}node(g))$,可知 $u\neq w,v\neq w$,由此易知在 B' 中 $r\models \mathsf{EXE}(\neg def(w)\mathsf{U}node(g))$ 仍成立.又因为 $w\in vars(e_4)$,可得 $S_i\delta' S_j$.情形②、情形③与情形①类似,可得 $S_i\delta' S_j$.因此 $S_i\delta S_j\Rightarrow S_i\delta' S_j$ 成立.

(b) 如果 $S_i\delta' S_j$,与情形(a)相似,可得 $S_i\delta' S_j \Rightarrow S_i\delta S_j$ 成立.

综合情形(a)、情形(b)可知,此时 $S_i\delta S_j \Leftrightarrow S_i\delta' S_j$ 成立.

(4) $i=t, j=t+1$.$S_i\delta S_j$ 不可能成立(因为与 $\neg(S_t\delta S_{t+1})$ 矛盾),又因为此时在 B' 中,S_i 在 S_j 之后,$S_i\delta' S_j$ 也不可能成立,故而 $S_i\delta S_j \Leftrightarrow S_i\delta' S_j$ 成立.

(5) $i<t, j=t+1$.

(6) $i=t, j>t+1$.

(7) $i=t+1, j>t+1$.

最后 3 种情形均与情形(2)类似,可得 $S_i\delta S_j \Leftrightarrow S_i\delta' S_j$.

综上可知,在 B 上对 S_t 与 S_{t+1} 实施 T_{exch} 变换之后,B' 对 B 上的依赖关系依然保持. □

定义 18(基本块上的重排). 设程序 π 含基本块 B,B 中的语句依次为 $S_1,S_2,\ldots,S_n$,B 上的重排可以表示为 $S_1 \Rightarrow S_{i_1}, S_2 \Rightarrow S_{i_2}, \ldots, S_n \Rightarrow S_{i_n}$ ($(i_1,i_2,\ldots,i_n)$是$(1,2,\ldots,n)$的一个置换序列),也可表示为 $[S_1,S_2,\ldots,S_n]^T \Rightarrow [S_{i_1},S_{i_2},\ldots,S_{i_n}]^T$ (注,此处的程序变换定义中对变换条件没有要求,并省略了语句的标号).

定义 19(保依赖重排 T_{dep}). 设程序 π 含基本块 $B_S=[S_1,S_2,\ldots,S_n]^T$,在 B_S 上的保依赖重排 T_{dep} 是指在 B_S 上保持数据依赖关系的重排,即,若转换后所得基本块为 $B_T=[S_{i_1},S_{i_2},\ldots,S_{i_n}]^T$,设 δ 和 δ' 分别为 B_S 和 B_T 上的数据依赖关系,那么有 $S_i\delta S_j \Leftrightarrow S_i\delta' S_j (i,j\in\{1,2,\ldots,n\})$.

定理 2. T_{dep} 是保义的,即对于程序 $\pi\in pgm$,如果 T_{dep} 能够在 π 上实施并得到程序 π',那么一定有 $[\![\pi]\!]=[\![\pi']\!]$.

证明:记 π 上原基本块为 $B_S=[S_1,S_2,\ldots,S_n]^T$,经 T_{dep} 变换之后,得到 π' 中对应的基本块为 $B_T=[S_{i_1},S_{i_2},\ldots,S_{i_n}]^T$.设 δ 和 δ' 分别为 B_S 和 B_T 上的数据依赖关系,对于 B_S 中的任意两个语句 S_p 和 S_q,如果有 $S_p\delta S_q$,则根据 T_{dep} 定义可知,在 B_T 上有 $S_p\delta' S_q$,再根据数据依赖以及基本块的定义易知,在 B_T 中 S_p 一定在 S_q 之前.因此有结论:如果有 $S_p\delta S_q$,那么在 B_T 中 S_p 一定在 S_q 之前.下面证明 $[\![\pi']\!]=[\![\pi]\!]$.

如果 $B_T=B_S$,那么自然有 $[\![\pi']\!]=[\![\pi]\!]$;

如果 $B_T\neq B_S$,设 B_T 和 B_S 中第 1 对不同的语句是 S_{i_j} 和 $S_j (1\le j\le n-1)$,重排变换不减少语句,故在 B_S 中必存在 $S_k=S_{i_j}$ $(k>j)$.根据我们前面所得的结论可知 $S_j,S_{j+1},\ldots,S_{k-1}$ 中的任意一个和 S_k 之间都不存在依赖关系,即 $\neg(S_m\delta S_k)$ $(m=j,j+1,\ldots,k-1)$(否则,根据前面的结论,在 B_T 中 S_m 应该在 S_k 之前,而实际上 S_m 都在 S_k 之后,矛盾).记 $B_0=B_S$,根据引理 6,可将 B_0 中的 S_k 和 S_{k-1} 进行保义语句交换 T_{exch} 得到 B_1(如图 2 所示),并且 B_1 对 B_0 上依赖关系依然保持.因此可知在 B_1 上,$S_j,S_{j+1},\ldots,S_{k-2}$ 中的任意一个和 S_k 之间都不存在依赖关系.据此又知,可将 B_1 中的 S_k 与 S_{k-2} 进行交换得到 B_2,并且 B_2 对 B_1 上依赖关系依然保持.由此又可将 B_2 中的 S_k 与 S_{k-3} 进行交换得到 B_3.依此操作下去,直到得到 B_{k-j}(如图 2 所示).

$$B_0=\begin{bmatrix} S_1\\ \vdots\\ S_j\\ \vdots\\ \vdots\\ S_{k-2}\\ S_{k-1}\\ S_k\\ \vdots\\ S_n \end{bmatrix} \Rightarrow B_1=\begin{bmatrix} S_1\\ \vdots\\ S_j\\ \vdots\\ \vdots\\ S_{k-2}\\ S_k\\ S_{k-1}\\ \vdots\\ S_n \end{bmatrix} \Rightarrow B_2=\begin{bmatrix} S_1\\ \vdots\\ S_j\\ \vdots\\ \vdots\\ S_k\\ S_{k-2}\\ S_{k-1}\\ \vdots\\ S_n \end{bmatrix} \Rightarrow \cdots \Rightarrow B_{k-j}=\begin{bmatrix} S_1\\ \vdots\\ S_k\\ S_j\\ \vdots\\ \vdots\\ S_{k-2}\\ S_{k-1}\\ \vdots\\ S_n \end{bmatrix}$$

Fig.2 Continually conducting the transformation T_{exch}

图 2 连续实施变换 T_{exch}

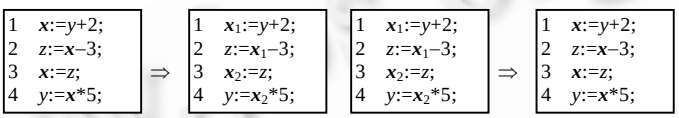
将 $B_0,B_1,B_2,\ldots,B_{k-j}$ 对应的程序分别记为 $\pi_0,\pi_1,\pi_2,\ldots,\pi_{k-j}$.综上所述,$[\![\pi_{k-j}]\!]=\ldots=[\![\pi_1]\!]=[\![\pi_0]\!]=[\![\pi]\!]$,并且 B_{k-j} 上的数据依赖关系与 B_0(即 B_S)上的相同,因此 B_T 也是 B_{k-j} 上的一个保依赖重排.而此时,B_T 和 B_{k-j} 前端相同语句序列比

B_T 和 B_0 前端相同语句序列在长度上增加了.记 $B_{M_1}=B_{k-j}$,相应的程序为 π_{M_1}.如果 $B_T=B_{M_1}$ 那么有$[\![\pi']\!]=[\![\pi_{M_1}]\!]=[\![\pi]\!]$;如果 $B_T\neq B_{M_1}$,设此时第 1 对不同的语句为 S_{i_m} 和 S_{t_m},有 $j+1\leq m\leq n-1$.仿照图 2 对 B_{M_1} 再做一系列交换,设得到的基本块为 B_{M_2},相应程序为 π_{M_2},那么同样地,可知$[\![\pi_{M_2}]\!]=[\![\pi_{M_1}]\!]$,$B_T$ 也是 B_{M_2} 上的一个保依赖重排,而且此时 B_T和 B_{M_2} 前端相同语句序列的长度再次增加.因为 B_T中语句序列长度是有限的,依此类推,可知必然存在一个有限的基本块序列: $B_{M_1},B_{M_2},...,B_{M_r}$,它们对应的程序为 $\pi_{M_1},\pi_{M_2},...,\pi_{M_r}$,有$[\![\pi']\!]=[\![\pi_{M_r}]\!]=\ldots=[\![\pi_{M_2}]\!]=[\![\pi_{M_1}]\!]=[\![\pi]\!]$.

综上所述,$[\![\pi']\!]=[\![\pi]\!]$成立,即 T_{dep} 是保义的. □

2.3 保义变量替换T_{sub}

在优化编译过程中,常常需要将一个语句中的变量替换为另一个变量,图 3 给出了这种变换的两种典型应用示例.在示例 1 中,一个变量被替换为多个变量,这在静态单赋值(static single assignment,简称 SSA)[2]的构建过程中较为常见.在示例 2 中,通过变量替换一个较大的变量集被映射到一个较小的变量集中,这在寄存器分配优化时较为常见,可以减缓寄存器分配的压力[2,3].



Example 1 Example 2

Fig.3 Two usage examples of variable substitution

图 3 变量替换的两个应用示例

若要保持程序语义,变量替换须有一定的条件,下面定义保义变量替换 T_{sub}.

定义 20(保义变量替换 T_{sub}). 保义变量替换 T_{sub} 可用含条件重写规则表示如下:

p_1: $u:=f_1\Rightarrow v:=f_1$;

...

p_r: $u:=f_r\Rightarrow v:=f_r$;

q_1: $w_1:=e_1\Rightarrow w_1:=e_1[u\rightarrowtail v]$;

…

q_n: $w_n:=e_n\Rightarrow w_n:=e_n[u\rightarrowtail v]$;

q_{n+1}: **if** u **then** LT_{n+1} **else** $LF_{n+1}\Rightarrow$**if** v **then** LT_{n+1} **else** LF_{n+1};

...

q_m: **if** u **then** LT_m **else** $LF_m\Rightarrow$**if** v **then** LT_m **else** LF_m;

if

$p_i\models \mathsf{AXA}(\neg use(v)\,\mathsf{W}\,def(v)\wedge\neg use(v))\wedge\mathsf{AXA}(\neg use(u)\vee node(q_1)\vee node(q_2)\vee\ldots\vee node(q_m)\,\mathsf{W}\,def(u)\wedge(\neg use(u)\vee node(q_1)\vee node(q_2)\vee\ldots\vee node(q_m)))$ $(i=1,2,\ldots,r)$;

$q_j\models use(u)\wedge\neg use(v)\wedge \overleftarrow{\mathsf{A}}\,\mathsf{X}\,\overleftarrow{\mathsf{A}}\,(\neg def(u)\wedge\neg use(v)\wedge\neg def(v)\,\mathsf{U}\,node(p_1)\vee node(p_2)\vee\ldots\vee node(p_r))$ $(j=1,2,\ldots,m)$.

其中,$f_i(i=1,2,\ldots,r)$和 $e_i(i=1,2,\ldots,n)$为表达式,$e_i[u\rightarrowtail v](i=1,2,\ldots,n)$是将表达式 e_i 中的变量 u 的所有出现替换为 v 而得的表达式.注意,标号变量 $p_i(i=1,2,\ldots,r)$和 $q_j(j=1,2,\ldots,m)$的值并不要求是单调递增的,即 p_i 可以大于 p_{i+1},q_j 可以大于 q_{j+1},p_i 也可以大于 q_j.

T_{sub} 定义的变换动作的直观含义为:将对 u 变量进行定值的若干处语句$(p_1,p_2,\ldots,p_r)$和对 u 变量有引用的若干处语句$(q_1,q_2,\ldots,q_m)$中的变量 u 进行替换.注意,语句 $p_1,p_2,\ldots,p_r$ 中只替换对 u 的定值,语句 $q_1,q_2,\ldots,q_m$ 中只替换对 u 的引用.T_{sub} 定义的变换条件部分包含如下 4 层直观含义:(1) $p_i(i=1,2,\ldots,r)$语句之后的所有向前路径上,在出现对 v 重新定值之前不出现对 v 的引用;(2) $p_i(i=1,2,\ldots,r)$语句之后的所有向前路径上,在出现对 u 重新定值之前不出现对 u 的引用,除非出现的是语句 $q_j(j=1,2,\ldots,m)$中对 u 的引用;(3) $q_j(j=1,2,\ldots,m)$语句中有对 u 的引用,但不

出现对 v 的引用;(4) $q_j(j=1,2,\ldots,m)$语句之前的所有向后路径一定会到达 $p_i(i=1,2,\ldots,r)$中的某一个,并且在到达之前不出现对 u,v 的定值,也不出现对 v 的引用.

定理 3. T_{sub} 是保义的,即对于$\pi\in pgm$,如果 T_{dep} 能够在π上实施并得到程序π',那么一定有$[\![\pi]\!]=[\![\pi']\!]$.

定理 3 与定理 1 的证明方法类似,亦基于引理 1,其证明过程非常长,限于篇幅,此处略,详见文献[15].

3 相关研究与讨论

对于优化变换的保义性证明,早期研究考察的主要是函数式语言程序上的优化变换[7−9],不是本文关注重点,在此不作讨论.文献[5,6]是近年来在过程式程序优化变换保义性证明方面的显著研究成果.此前,文献[10]基于一种扩展的 Kleene 代数 KAT(Kleene algebra with tests)提出了一种验证编译优化保义性的方法,将程序表示为 KAT 代数项,并根据代数公理检验程序变换前后的等价性.与文献[5,6]相比,文献[10]的方法存在明显的局限性:对程序变换的保义性验证依赖于具体的源程序,在每次变换之后都要验证一下源程序和目标程序的等价性.文献[11]针对静态程序分析以及基于这些分析的优化给出了一种基于指称语义进行保义性证明的方法.在文献[11]和文献[5,6]这两类方法中,本文之所以选择后者作为基础,主要是因为后者将程序变换的动作跟条件显式地刻画出来,较为直观,也更容易被编译优化的开发者所理解.

从本文第 2.2 节对保依赖重排 T_{dep} 的定义可以看出,某些程序变换虽然对被处理的程序没有特殊要求,但对变换动作本身却有约束,这种约束一般通过变换前后程序之间的关系或变换后程序的特征加以表达.因此我们认为,程序变换的表示形式可以由"$\mathcal{I}\Rightarrow\mathcal{I}'$ If ϕ"扩充为"Pre-Condition;$\mathcal{I}\Rightarrow\mathcal{I}'$;Post-Feature",其含义为:如果原程序满足前置条件 Pre-Condition,那么执行变换$\mathcal{I}\Rightarrow\mathcal{I}'$,并且该变换满足性质 Post-Feature.

"保持依赖的语句重排能够保持原程序的功能",此结论被称为依赖基础定理(fundamental theorem of dependence)[3].本文对保依赖重排 T_{dep} 的保义性证明是在程序操作语义基础上的一种构造性的证明,有助于更好地理解依赖基础定理.本文对保依赖重排 T_{dep} 的保义性证明是基于 T_{exch} 的保义性的,由此可知,某些简单程序变换的保义性证明可以为复杂程序变换的保义性证明提供基础.

当多种优化变换交织在一起时,证明其整体保义性的可能途径有两种:一种是将多种优化变换综合地用一条重写规则表示并对其证明;另一种是识别出各个优化变换,按其组合方式将整体变换拆分成多个阶段,每个阶段都实施一种变换,然后先分别证明各个变换的保义性,再证明阶段与阶段之间的保义性.

本文对各程序变换所定义的前提条件都是充分条件,但不知是否是必要条件.我们将进一步考察这些条件是否必要,并寻找各程序变换的最弱前提条件.此外,我们准备对 SimLan 语言扩充数组变量的定义,以便对循环携带的数据依赖[3]以及与此相关的优化变换的保义性进行考察.

SimLan 是一种过程式语言,对于面向对象语言的程序,编译器通常会先把它翻译成过程式的中间语言程序,然后再在中间语言程序上进行优化变换,因此,SimLan 语言上定义的保义优化变换可以在中间语言程序优化过程中被借鉴或使用,而对于专门针对面向对象特征的某些优化变换,则需要另外寻找保义性证明的途径.

4 总 结

本文的工作主要有两项.一项工作是对优化编译中语句交换和变量替换这两种常用的程序变换进行了考察,探索了它们的保义性条件.应用文献[5,6]中的方法定义了保义语句交换 T_{exch} 和保义变量替换 T_{sub},并基于文献[5,6]中的归纳证明框架对它们的保义性进行了证明.另一项工作是定义了基本块上保依赖语句重排变换 T_{dep},并基于 T_{exch} 对 T_{dep} 的保义性给出了一种构造性的证明.我们今后拟考察过程式语言程序上其他更多程序变换的保义性,尤其是各种高级优化变换的保义性.此外,也将尝试寻找各种程序变换的最弱保义条件.

致谢 衷心感谢中国科学院软件研究所计算机科学重点实验室李广元老师和周翔同学对本文工作的指导和帮助.衷心感谢各位评审老师对本文工作提出的宝贵意见和建议.

References:

[1] Aho AV, Sethi R, Ullman JD. Compilers: Principles, Techniques, and Tools. Boston: Addison-Wesley, 1986.

[2] Muchnick SS. Advanced Compiler Design and Implementation. San Francisco: Morgan Kaufmann Publishers, 1997.

[3] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures: A Dependence-Based Approach. San Francisco: Morgan Kaufmann Publishers, 2002.

[4] Soffa ML. Developing a foundation for code optimization. In: Duesterwald E, ed. Proc. of the 13th Int'l Conf. of Compiler Construction (CC 2004). London: Springer-Verlag, 2004. 1−4.

[5] Lacey D, Jones ND, Wyk EV, Frederiksen CC. Compiler optimization correctness by temporal logic. Higher-Order and Symbolic Computation, 2004,17(3):173−206.

[6] Frederiksen CC. Correctness of classical compiler optimizations using CTL. Electronic Notes in Theoretical Computer Science, 2002,65(2):37−51.

[7] Wand M. Specifying the correctness of binding-time analysis. In: Deusen MV, Lang B, eds. Proc. of the 20th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'93). New York: ACM Press, 1993. 137−143.

[8] Wand M, Siveroni I. Constraint systems for useless variable elimination. In: Appel A, Aiken A, eds. Proc. of the 26th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'99). New York: ACM Press, 1999. 291−302.

[9] Kobayashi N. Type-Based useless variable elimination. ACM SIGPLAN Notices, 1999,34(11):84−93.

[10] Kozen D, Patron M. Certification of compiler optimizations using Kleene algebra with tests. In: Lloyd JW, Dahl V, Furbach U, Kerber M, Lau K, Palamidessi C, Pereira LM, Sagiv Y, Stuckey PJ, eds. Proc. of the 1st Int'l Conf. on Computational Logic (CL 2000). London: Springer-Verlag, 2000. 568−582.

[11] Benton N. Simple relational correctness proofs for static analyses and program transformations. ACM SIGPLAN Notices, 2004, 39(1):14−25.

[12] Lerner S, Millstein T, Chambers C. Automatically proving the correctness of compiler optimizations. ACM SIGPLAN Notices, 2003,38(5):220−231.

[13] Necula GC. Translation validation for an optimizing compiler. ACM SIGPLAN Notices, 2000,35(5):83−94.

[14] Kossatchev AS, Posypkin MA. Survey of compiler testing methods. Programming and Computing Software, 2005,31(1):10−19.

[15] Tao QM, Zhao C, Guo L. Soundness proof of the program transformation of variable substitution. Technical Report, No.NERCFS20080315, Beijing: National Engineering Research Center for Fundamental Software, Institute of Software, the Chinese Academy of Sciences, 2008 (in Chinese with English abstract).

附中文参考文献:

[15] 陶秋铭,赵琛,郭亮.变量替换程序变换的保义性证明.科技报告,No.NERCFS20080315,北京:中国科学院软件研究所基础软件国家工程研究中心,2008.

陶秋铭(1979－),男,江苏通州人,博士,主要研究领域为编译器质量保障,软件测试与验证.

郭亮(1976－),男,博士,高级工程师,主要研究领域为嵌入式系统,编译技术.

赵琛(1967－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为编译技术及应用,软件测试方法和工具.