论 Folding/Unfolding 程序转换的能力

朱鸿

(南京大学计算机软件研究所)

ON THE ABILITY OF FOLDING/UNFOLDING TRANSFORMATION

Zhu Hong

(*Institute of Computer Software, Nanjing University*)

ABSTRACT

This paper discusses the transformation ability of Burstall and Darlington's folding/unfolding system, i.e. what kind of programs can be derived from a given one. It is a generalization of the problems of correctness and completeness. In the paper, we proved a necessary condition of transformability, and got a bound of efficiency improvable by transformation. The partial correctness and incompleteness of the system are corollaries of the condition.

摘 要

本文讨论Burstall 与Darlington 提出的folding/unfolding 系统[1] 的程序转换能力，即讨论从一个给定的程序可以推导出什么样的程序。因此，这是正确性与完备性问题的推广。本文证明了可推导性的一个必要条件，并由此得到了该系统提高程序效率的一个界限。该系统的部分正确性和不完备性也均是该条件的推论。

§1. 引 言

1.1 背景

程序转换是一种对程序的形式处理。其基本思想是：程序开发中的许多困难的根源是要同时满足两个经常相互冲突的目标，即正确性和高效性。前者要求程序结构良好、清晰易懂，而后者往往为了高效而使程序具有错综复杂的执行策略，失去了清晰性。因此，转换式程序设计途径把程序开发分为两个阶段，每个阶段只注重解决一个问题，即在前一阶段书写一个正确、清晰的程序或功能规格说明，然后通过对程序的逐步处理和

1989 年9 月25 日收到，1990 年1 月23 日定稿，国家自然科学基金资助项目。

转换来得到一个高效的程序[2,3]。因此，用以进行程序转换的规则是否保持程序的正确性、是否能够有效地提高程序的效率具有重要意义。

Burstall 和Darlington 提出的folding / unfolding 转换方法[1] 具有简洁、清晰等特点，以此为基础的大量工作说明该方法具有一定的实用性和较广的应用范围[9~11]。但是，该系统的规则是不完备的，且仅保持部分正确性。而该方法究竟能够推导出什么样的程序，究竟能够提高多少效率仍然是未知的。

本文将讨论该系统的程序转换能力，即讨论从给定的程序能够推导出什么样的程序。这是正确性问题与完备性问题的推广。第二节将证明可推导性的一个必要条件，部分正确性和不完备性均是其推论。第三节讨论'Eureka' 在转换中的作用，证明了第二节得到的必要条件在具有eureka 的转换中仍然成立。第四节讨论该系统提高程序效率的能力，引进了递归程序的内在复杂性的概念，证明了内在复杂性的阶在folding / unfolding 转换中的不变性，由此得到了一个效率提高的界限。第五节结语，简要分析介绍了有关工作。

1.2 术语

为了讨论的方便，我们首先简要介绍一下folding/unfolding 系统的转换规则，以及本文中所使用的一些术语与符号。

Folding/unfolding 系统的转换对象与结果为一系列递归等式所构成的函数式程序:

$$\begin{cases} E_1 \Leftarrow F_1 \\ E_2 \Leftarrow F_2 \\ \quad \cdots \\ E_n \Leftarrow F_n \end{cases}$$

其中等式左边的表达式呈$f(e_1,\cdots,e_n)$, $(n>0)$ 形，f 为被定义的函数符，称为递归函数符，$e_1,\cdots,e_n$ 均为由形式参数通过构造函数符构成的表达式，右表达式为由基本函数符、形式参数、递归函数符构成的通常的表达式，其中允许出现呈下形的where 子句:

$$F \text{ where } (u_1,\cdots,u_n) = F'$$

或

$$F \text{ where } u = F'$$

其中F, F' 为表达式，$u, u_i, (i=1,\cdots,n)$ 为局部变量，F' 为这些局部变量的定义。

让$\mathcal{E}$ 为一系列定义一个递归函数符号f 的等式的集合:

$$\mathcal{E}: \begin{cases} f(e_{11},\cdots,e_{1n}) \Leftarrow F_1 \\ \quad \cdots \\ f(e_{m1},\cdots,e_{mn}) \Leftarrow F_n \end{cases}$$

则$\mathcal{E}$ 可看作是定义了一个泛函$\mathcal{E}(f)$，这组等式所定义的函数即为$\mathcal{E}(f)$ 的最小不动点f^*，即

$$f^* = \mathcal{E}(f^*)$$

且对任意函数g:

$$g = \mathcal{E}(g) \Longrightarrow f^* \leq g$$

这里，$h \leq g \Longleftrightarrow h:x \leq g:x$，对任意对象$x$。$\leq$ 为对象域上的完全偏序关系。我们假设泛函$\mathcal{E}(f)$ 均是单调、连续的，则由Kleene 定理[4]，$\mathcal{E}(f)$ 的最小不动点为:

$$\lim_{i\to\infty} \mathcal{E}^i(\perp)$$

其中$\perp$ 为常底函数，$\lim_{i\to\infty}\mathcal{E}^i(\perp)$ 为集合$\{\mathcal{E}(\perp),\mathcal{E}(\mathcal{E}(\perp)),\cdots,\mathcal{E}^n(\perp),\cdots\}$ 在$\leq$ 关系下的最小上界。关于泛函及其单调性、连续性等概念的定义请详见[5]。

Folding/unfolding 系统由六条规则组成:

让$\mathcal{E}$ 为一系列等式的集合，$E \Leftarrow E', F \Leftarrow F' \in \mathcal{E}$,

(1) 实例化: 将等式$E \Leftarrow E'$ 中的一个或多个形式参数$u_1\cdots,u_k$ 代入以实例$a_1,a_2,\cdots,a_k$，得到一个新的等式$\tilde{E} \Leftarrow \tilde{E}'$，加入$\mathcal{E}$，得到$\mathcal{E}'$。我们记$\mathcal{E}'$ 为$I_\sigma^{E\Leftarrow E'}(\mathcal{E})$，$\sigma$ 为置换$(u_1/a_1, u_2/a_2,\cdots,u_k/a_k)$。

(2) unfolding: 将F' 中出现的一个E 的实例替换成相应的E' 的实例，得到F'', 从而得到一个新的等式$F \Leftarrow F''$, 加入$\mathcal{E}$, 得$\mathcal{E}'$。记之为$u_{E\Leftarrow E',\rho}^{F\Leftarrow F'}(\mathcal{E})$, 其中$\rho$ 为E 的实例在F' 中的出现的位置。

(3) folding: 将F' 中的E' 的一个出现替换成E, 得到F'', 将新等式$F \Leftarrow F''$ 加入$\mathcal{E}$, 得$\mathcal{E}'$。记之为$F_{E\Leftarrow E',\rho}^{F\Leftarrow F'}(\mathcal{E})$, ρ 为E' 在F' 中的出现。

(4) 抽象: 在等式$E \Leftarrow E'$ 的右表达式E' 中引入一个where 子句，得到表达式E'':

$$E'[F_1/u_1,\cdots,F_n/u_n] \text{ where } (u_1,\cdots,u_n)=(F_1,\cdots,F_n)$$

将等式$E \Leftarrow E''$ 加入$\mathcal{E}$, 得$\mathcal{E}'$, 记为$A_{\vec{u},\vec{F}}^{E\Leftarrow E'}(\mathcal{E})$, $\vec{u}=(u_1,\cdots,u_n)$, $\vec{F}=(F_1,F_2,\cdots,F_n)$, 其中

$$E'[F_1/u_1,\cdots,F_u/u_n]$$

为将E' 中的子表达式F_i 分别替换成u_i 后所得的表达式。

(5) 使用定律: 让$G=G'$ 为关于基本函数的代数定律，如'$x+y=y+x$'。对等式$E \Leftarrow E'$ 的右式E' 中的某个子表达式H 使用代数定律等价地替换成H', 得到E'', 将等式$E \Leftarrow E''$ 加入$\mathcal{E}$, 记之为$L_{l,\rho}^{E\Leftarrow E'}(\mathcal{E})$, 其中$l$ 为定律，ρ 为H 在E' 中的出现。

上述规则可看作是对泛函(或等式组) 的运算。程序转换的过程即为将上述算子不断地作用于初始等式组$\mathcal{E}$ 的过程，最后再从中选取一个子集作为结果程序的定义。为了讨论的方便，在不致混淆的前提下，我们将省略算子符号的上、下角。

§2. 基本定理

不失一般性，我们先讨论仅定义一个递归函数f 程序的转换过程。设等式组为$\mathcal{E}(f)$, 则有:

引理1　让$\mathcal{E}'(f)=I_\sigma^{E\Leftarrow F}(\mathcal{E}(f))$

(a) 对任意程序p,

$$\mathcal{E}(p)=\mathcal{E}'(p)$$

(b) 对任意$i=1,2,\cdots$,

$$\mathcal{E}^i(\perp)=\mathcal{E}'^i(\perp)$$

证明:

(a) 对$\mathcal{E}(p):x$ 的计算过程归纳易证对任意x，$\mathcal{E}(p):x=I_\sigma^{E\Leftarrow F}(\mathcal{E}(p)):x$。

(b) 对i 归纳，反复使用(a) 易得。　　证毕

引理2　让$\mathcal{E}'(f)=u^{F}_{E\Leftarrow E',\rho}(\mathcal{E}(f))$。则对任意$i=1,2,\cdots$,

$$\mathcal{E}'^{i}(\perp)\leq\mathcal{E}^{2i}(\perp)$$

证明：对i归纳。

奠基：当$i=1$时，由$\mathcal{E}(f)$的单调性及$\perp\leq\mathcal{E}(\perp)$,

$$\begin{aligned}\mathcal{E}'(\perp)&=\mathcal{E}[\rho\downarrow\mathcal{E}(f)](\perp)\quad(u\text{的定义})\\&\leq\mathcal{E}[f/\mathcal{E}(f)](\perp)\quad(\mathcal{E}\text{的单调性})\\&=\mathcal{E}(\mathcal{E}(f))(\perp)\\&=\mathcal{E}^{2}(\perp)\end{aligned}$$

其中$\mathcal{E}[\rho\downarrow E]$为将$\mathcal{E}$中出现$\rho$处替换为$E$所得的表达式。

归纳：设当$i=k$时成立，则当$i=k+1$时：

$$\begin{aligned}\mathcal{E}'^{k+1}(\perp)&=\mathcal{E}'(\mathcal{E}'^{k}(\perp))\\&=\mathcal{E}[\rho\downarrow\mathcal{E}(f)](\mathcal{E}'^{k}(\perp))\quad(u\text{ 的定义})\\&\leq\mathcal{E}[f/\mathcal{E}(f)](\mathcal{E}'^{k}(\perp))\quad(\mathcal{E}\text{ 的单调性})\\&=\mathcal{E}^{2}(\mathcal{E}'^{k}(\perp))\\&\leq\mathcal{E}^{2}(\mathcal{E}^{2k}(\perp))\quad(\text{归纳假设})\\&=\mathcal{E}^{2(k+1)}(\perp)\end{aligned}$$

证毕

引理3　让$\mathcal{E}'(f)=F^{F}_{E\Leftarrow E',\rho}(\mathcal{E}(f))$。则对任意$i=1,2,\cdots$, 有

$$\mathcal{E}'^{i}(\perp)\leq\mathcal{E}^{i}(\perp)$$

证明：对i归纳。

奠基：当$i=1$时，

$$\begin{aligned}\mathcal{E}'(\perp)&=\mathcal{E}[\rho\downarrow f](\perp)\quad(u\text{的定义})\\&\leq\mathcal{E}(\perp)\quad(\mathcal{E}\text{的单调性})\end{aligned}$$

归纳：设当$i=k$时成立，则当$i=k+1$时

$$\begin{aligned}\mathcal{E}'^{k+1}(\perp)&=\mathcal{E}(\rho\downarrow f)(\mathcal{E}'^{k}(\perp))\quad(u\text{的定义})\\&\leq\mathcal{E}(\rho\downarrow f)(\mathcal{E}^{k}(\perp))\quad(\text{归纳假设})\\&\leq\mathcal{E}(\mathcal{E}^{k}(\perp))\quad(\mathcal{E}\text{的单调性})\\&=\mathcal{E}^{k}(\perp)\end{aligned}$$

证毕

引理4　让$\mathcal{E}'(f)=A(\mathcal{E}(f))$, 或$\mathcal{E}'(f)=L(\mathcal{E}(f))$。

(a) 对任意程序g, $\mathcal{E}'(g)=\mathcal{E}(g)$;

(b) 对任意$i=1,2,\cdots$, $\mathcal{E}'^{i}(\perp)=\mathcal{E}^{i}(\perp)$。

证明：显然。　　证毕

引理5　让$\mathcal{E}'$为$\mathcal{E}$的子集，则对任意$i=1,2,\cdots$

$$\mathcal{E}'^{i}(\perp)\leq\mathcal{E}^{i}(\perp)$$

证明：显然。　　证毕

由上述引理，我们可得如下定理。

定理1　若使用folding/unfolding 方法可将程序$\mathcal{E}$转换成$\mathcal{E}'$，则存在常数$K>0$，使得对任意$i=1,2,\cdots$有

$$\mathcal{E}'^{i}(\perp)\leq\mathcal{E}^{Ki}(\perp)$$

证明：

对转换过程的长度归纳，使用引理1～5易得。　　证毕

定理1给出了可推导性的一个必要条件，由此，我们可以证明程序的不可推导性。

例1. 从程序

$$\mathcal{E}_1:\begin{cases}f(0)\Leftarrow 0\\ f(n+1)\Leftarrow f(n)\end{cases}$$

不可推导出程序

$$\mathcal{E}_2:f(n)\Leftarrow 0$$

证明：　对任意$i=1,2,\cdots$, $\mathcal{E}_2^i(\perp)=f(n)\Leftarrow 0$。且

$$\mathcal{E}_1^i(\perp)=\begin{cases}f(0)\Leftarrow 0\\ f(1)\Leftarrow 0 \qquad i=1,2,\cdots\\ \cdots\\ f(i)\Leftarrow 0\end{cases}$$

因此，对任意$s,t=1,2,\cdots$, 有

$$\mathcal{E}_2^s(\perp)\geq\mathcal{E}_1^s(\perp)\ \text{且}$$
$$\mathcal{E}_2^s(\perp)\neq\mathcal{E}_1^s(\perp)$$

即不存在常数K, 使$\mathcal{E}_2^i(\perp)\leq\mathcal{E}_1^{Ki}(\perp)$。由定理1, $\mathcal{E}_2$不能从$\mathcal{E}_1$推出。　　证毕

由于例1中的$\mathcal{E}_1$与$\mathcal{E}_2$定义的是同一个函数，所以有如下不完备性定理。

定理2　(不完备性定理)

存在程序$f,g,f=g$, 但用folding/unfolding 方法不能从f推出g。

证明：　易证例1中的$\mathcal{E}_1,\mathcal{E}_2$定义同一个函数。　　证毕

从定理1还可推出folding/unfolding 系统的部分正确性。

定理3　(部分正确性定理)

若从程序f可用folding/unfolding 系统推出g, 则$g\leq f$。

证明：　设f由等式组$\mathcal{E}(f)$定义，$\mathcal{E}'(f)$从$\mathcal{E}(f)$推出，它定义了g。则由定理1, 存在常数$K>0$使对任意$i=1,2,\cdots$,

$$\mathcal{E}'^{i}(\perp)\leq\mathcal{E}^{Ki}(\perp)$$

因此,

$$\begin{aligned} g &= \lim_{i\to\infty} \mathcal{E}'^{i}(\perp) \\ &\le \lim_{i\to\infty} \mathcal{E}^{Ki}(\perp) \\ &= \lim_{i\to\infty} \mathcal{E}^{i}(\perp) \\ &= f \end{aligned}$$

证毕

§3. 联立递归函数与'eureka'的能力

'Eureka' 在folding/unfolding 转换方法中具有十分重要的意义, 许多转换需要eureka 的帮助才能进行。从本质上来说, eureka 是一个辅助函数。因此, 整个方程组实际上是一组联立递归等式。第二节的结论容易推广到联立递归函数的情形。

让等式组$\mathcal{E}$ 定义了递归函数$f_1, f_2, \cdots, f_n$, $n \ge 1$,

$$\mathcal{E}_1 \begin{cases} f_1(e^1_{11}, \cdots, e^1_{1n_1}) = F_{11}(f_1, \cdots, f_n) \\ f_1(e^1_{21}, \cdots, e^1_{2n_1}) = F_{12}(f_1, \cdots, f_n) \\ \quad \cdots \\ f_1(e^1_{m_1 1}, \cdots, e^1_{m_1 n_1}) = F_{1m_1}(f_1, \cdots, f_n) \end{cases}$$

$$\cdots\cdots$$

$$\mathcal{E}_n \begin{cases} f_n(e^n_{11}, \cdots, e^n_{1n_n}) = F_{n1}(f_1, \cdots, f_n) \\ \cdots \\ f_n(e^n_{m_n 1}, \cdots, e^n_{m_n n_n}) = F_{nm_n}(f_1, \cdots, fn) \end{cases}$$

$\mathcal{E}$ 可看作是一个n 元泛函$\mathcal{E}(f_1, \cdots, f_n)$:

$$(f_1, \cdots, f_n) = (\mathcal{E}_1(f_1, \cdots, f_n), \cdots, \mathcal{E}_n(f_1, \cdots, f_n))$$

即

$$\begin{aligned} \mathcal{E}(f_1, \cdots, f_n) &= (\mathcal{E}_1(f_1, \cdots, f_n), \cdots, \mathcal{E}_n(f_1, \cdots, f_n)) \\ &= (\mathcal{E}_1, \cdots, \mathcal{E}_n)(f_1, \cdots, f_n) \end{aligned}$$

它所定义的函数为$\mathcal{E}(f_1, \cdots, f_n)$ 的最小不动点$(f_1^*, f_2^*, \cdots, f_n^*)$:

$$\begin{aligned} &(f_1^*, \cdots, f_n^*) = \mathcal{E}(f_1^*, \cdots, f_n^*) \text{ 且} \\ &(h_1, \cdots, h_n) = \mathcal{E}(h_1, \cdots, h_n) \Longrightarrow f_i^* \le h_i, i = 1, 2, \cdots, n \end{aligned}$$

由Kleene 定理, $(f_1^*, \cdots, f_n^*) = \lim_{i\to\infty} \mathcal{E}^i(\perp, \perp, \cdots, \perp)$

为了简便起见, 我们记$(f_1, \cdots, f_n)$ 为$\vec{f}$, $(f_1^*, \cdots, f_n^*)$ 记为$\vec{f}^*$, $\mathcal{E}(f_1, \cdots, f_n)$ 记为$\mathcal{E}(\vec{f})$, $\mathcal{E}(\perp, \cdots, \perp)$ 记为$\mathcal{E}(\vec{\perp})$。易知引理1~5 对联立递归方程仍然成立, 定理1 可推广为:

定理1′　若从$\mathcal{E}_1(\vec{f})$ 可推出$\mathcal{E}_2(\vec{f})$, 则存在常数$K>0$ 使对任意$i=1,2,\cdots$

$$\mathcal{E}_2^i(\vec{\perp}) \le \mathcal{E}_1^{Ki}(\vec{\perp}) \qquad (*)$$

设有等式组$\mathcal{E}_1(\vec{f}), \mathcal{E}_2(\vec{f})$。若它们不满足条件(*), 则由定理1′, 我们不能从$\mathcal{E}_1(\vec{f})$ 推出$\mathcal{E}_2(\vec{f})$。问题是，是否可以通过增加一些'eureka', 即增加一些方程$\mathcal{E}'$, 使$\mathcal{E}_1 \cup \mathcal{E}'$ 推出$\mathcal{E}_2$? 下面的定理说明，如果$\mathcal{E}'$ 仅仅引入新的函数，也就是说$\mathcal{E}'$ 的等式的左边的递归函数符号不出现$\vec{f}$ 中的递归函数符号，那么从$\mathcal{E}_1 \cup \mathcal{E}'$ 仍然不能推出$\mathcal{E}_2$。换言之，当(*) 不成立时，eureka 也是无能为力的。

定理4　让$\mathcal{E}_1(\vec{f}), \mathcal{E}_2(\vec{f})$ 为两个等式的集合。若不存在常数$K>0$, 使对任意$i=1,2,\cdots$

$$\mathcal{E}_2^i(\vec{\perp}) \le \mathcal{E}_1^{Ki}(\vec{\perp})$$

则不存在等式组$\mathcal{E}'(\vec{f},\vec{g})$, $\vec{f}$ 不出现在等式的左边，使$\mathcal{E}_1 \cup \mathcal{E}'$ 能够推导出$\mathcal{E}_2$。

证明: (反证法)

设等式组$\mathcal{E}'(\vec{f},\vec{g})$ 使$\mathcal{E}_1 \cup \mathcal{E}'$ 能够推导出$\mathcal{E}_2(\vec{f})$, 且$\vec{f}$ 不出现$\mathcal{E}'$ 中等式的左边。

让$\mathcal{E}^0(\vec{f},\vec{g}) = \mathcal{E}_1(\vec{f}) \cup \mathcal{E}'(\vec{f},\vec{g})$。则由定理1′ 存在$K>0$ 使对任意$i=1,2,\cdots$, 有

$$\mathcal{E}_2^i(\vec{\perp}) \le \mathcal{E}^{0Ki}(\vec{\perp})$$

由于$\vec{f}$ 不出现在$\mathcal{E}'$ 中等式的左边，$\vec{g}$ 不出现在$\mathcal{E}_1$ 中，所以

$$\mathcal{E}^0(\vec{f},\vec{g}) = (\mathcal{E}_1(\vec{f}), \mathcal{E}'(\vec{f},\vec{g}))$$

下面我们归纳证明对任意$i=1,2,\cdots$,

$$\mathcal{E}^{0i}(\vec{\perp}) = (\mathcal{E}_1^i(\vec{\perp}), \mathcal{E}'(x,y))$$

其中x,y 为从$\vec{\perp}$ 由$\mathcal{E}_1, \mathcal{E}'$ 构成的表达式。

奠基: 当$i=1$ 时

$$\mathcal{E}^0(\vec{\perp}) = (\mathcal{E}_1(\vec{\perp}), \mathcal{E}_1'(\vec{\perp},\vec{\perp}))$$

归纳: 设当$i=k$ 时成立，则当$i=k+1$ 时有:

$$\begin{aligned}
\mathcal{E}^{0k+1}(\vec{\perp}) &= (\mathcal{E}_1(\vec{f}), \mathcal{E}'(\vec{f},\vec{g}))(\mathcal{E}^{0k}(\vec{\perp})) \\
&= (\mathcal{E}_1(\vec{f}), \mathcal{E}'(\vec{f},\vec{g}))(\mathcal{E}_1^k(\vec{\perp}), \mathcal{E}'(x,y)) \quad \text{(归纳假设)} \\
&= (\mathcal{E}_1(\mathcal{E}_1^k(\vec{\perp})), \mathcal{E}'(\mathcal{E}_1^k(\vec{\perp}), \mathcal{E}'(x,y))) \\
&= (\mathcal{E}_1^{K+1}(\vec{\perp}), \mathcal{E}'(x',y'))
\end{aligned}$$

得证。从而

$$\mathcal{E}_2^i(\vec{\perp}) \le (\mathcal{E}_1^{Ki}(\vec{\perp}), \mathcal{E}'(x,y)), i=1,2,\cdots$$

由于$\mathcal{E}_2$ 中不含$\vec{g}$, 所以对任意$i=1,2,\cdots$

$$\mathcal{E}_2^i(\vec{\perp}) \le \mathcal{E}_1^{Ki}(\vec{\perp})$$

与前提矛盾。　　**证毕**

这里需要指出的是，定理1′ 只是可推导性的必要条件，而不是充要条件，所以，在满足该条件时，也可能推不出所需程序，但这时，eureka 可能是有所帮助的。定理4 说明，当不满足该条件时，无论是否增加eureka 均不能推导出所需程序。

§4. 递归函数的内在复杂性与效率提高的界限

这一节，我们讨论folding/unfolding 转换提高程序效率的能力。

大量的文献报道了用folding/unfolding 方法提高程序效率，综合高效算法的工作[6~8]。例如，如下定义的递归程序f,

$$\begin{cases} f(0)=1 \\ f(1)=1 \\ f(n+2)=f(n+1)+f(n) \end{cases}$$

在eureka 函数g 的帮助下，$g(n)=(f(n+1),f(n))$
可转换成如下函数:

$$\begin{cases} f(0)=1 \\ f(1)=1 \\ f(n+2)=u+v \text{ where } (u,v)=g(n) \\ g(0)=<1,1> \\ g(n+1)=<v,u+v> \text{ where } <u,v>=g(n) \end{cases}$$

文[1] 指出前者的时空复杂性是指数的，而后者是线性的。但是，这是在顺序计算模型下的复杂性。在并行计算的模型下，两者的时间复杂性都是线性的，而在图归约、懒惰计算模型下，两者的时、空复杂性都是线性的。如图1 所示。因此，在不同的计算模型下，folding/unfolding 所能提高的执行效率的结论是不同的。下面，我们先定义一个与计算模型无关的递归程序的计算复杂性的概念— 内在复杂性，讨论这样的复杂性在folding/unfolding 转换中的特性，然后讨论这样的复杂性与具体计算模型下的复杂性之间的关系，从而得到通过转换提高执行效率的界限。

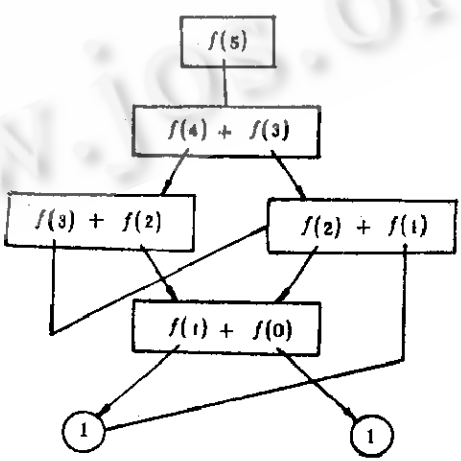

图1 计算$f(5)$的图归约示意图

设$|\cdot|$为输入的复杂度函数，即从输入域D到$N=\{1,2,\cdots\}$的映射。

定义1　如下定义的函数$ch(n)$称为递归函数$f=E(f)$的相对于$|\cdot|$的内在复杂度函数:

$$ch(n)=\max\{n(x)\,\big|\,|x|=n\},n=1,2,\cdots$$

其中$n(x)=\min\{t|x\in Dom(E^t(\perp))\}$。$Dom(f)=\{x|f:x\neq\perp\}$, $\max X=\infty$若X为无限集。

若存在常数$c_1,c_2>0$和自然数n_0，使对任意$n>n_0$有

$$c_1g(n)\leq ch(n)\leq c_2g(n)$$

则称$ch(n)$的阶为$g(n)$，记为$ch(n)=O(g(n))$。

直观地说，内在复杂度反映了递归函数的递归调用深度。这是因为，如果计算x时可通过深度为t的调用来完成，则$x\in E^t(\perp)$，反之，若$x\in E^t(\perp)$，则调用深度小于等于t的计算不能获得$f(x)$的值。例如:

例2. 让$|n|=n$。

a. 定义函数f为

$$\begin{cases}f(0)\Leftarrow 0\\ f(n+1)\Leftarrow f(n)\end{cases}$$

f相对于$|\cdot|$的内在复杂度为$ch(n)=n$。这是因为对任意$i=0,1,2,\cdots$, $\min\{t|i\in Dom(E^t(\perp))\}=i$。

b. 函数

$$\begin{cases}f(0)\Leftarrow 1\\ f(1)\Leftarrow 1\\ f(n+2)\Leftarrow f(n+1)+f(n)\end{cases}$$

的内在复杂度是线性的，即$ch(n)=O(n)$。而函数

$$\begin{cases}f(0)=1\\ f(1)=1\\ f(n+2)=u+v\ \text{where}\ (u,v)=g(n)\\ g(0)=<1,1>\\ g(n+1)=<v,u+v>\ \text{where}\ (u,v)=g(n)\end{cases}$$

的内在复杂度也是线性的，即$ch'(n)=O(n)$。

c. 函数$f(n)=\sum_{i=1}^{n}i$的如下定义

$$\begin{cases}f(1)=1\\ f(n+1)=f(n)+n+1\end{cases}$$

的内在复杂度是线性的，但如下定义是$O(\log_2 n)$ 阶的。

$$\begin{cases} f(1)=1 \\ f(n+1)=f\left(\left[\frac{n+1}{2}\right]\right)+f\left(n+1-\left[\frac{n+1}{2}\right]\right) \\ \qquad\qquad +\left[\frac{n+1}{2}\right]\cdot\left(n+1-\left[\frac{n+1}{2}\right]\right) \end{cases}$$

下面的定理说明folding/unfolding 转换不能降低递归程序的内在复杂性的阶。

定理5 设程序$f=\mathcal{E}(f)$ 通过folding/unfolding 转换成$f=\mathcal{E}'(f)$, 且它们的内在复杂度分别为$ch(n)$ 和$ch'(n)$, 则存在常数$K>0$, 使得

$$ch'(n)\geq K\cdot ch(n)$$

证明: 让

$$n(x)=\min\{t|x\in Dom(\mathcal{E}^t(\perp))\}$$
$$n'(x)=\min\{t|x\in Dom(\mathcal{E}'^t(\perp))\}$$

由定理1', $\mathcal{E}'^t(\perp)\leq\mathcal{E}^{kt}(\perp)$。从而，对任意$x$,

$$x\in Dom(\mathcal{E}'^t(\perp))\Longrightarrow x\in Dom(\mathcal{E}^{kt}(\perp))$$

即$Kn'(x)\geq n(x)$。所以，$n'(x)\geq\frac{1}{K}n(x)$, 由内在复杂性的定义,

$$\begin{aligned} ch'(n)&=\max\{n'(x)\,\big|\,|x|=n\} \\ &\geq\max\{\frac{1}{k}n(x)\,\big|\,|x|=n\} \\ &=\frac{1}{K}\max\{n(x)\,\big|\,|x|=n\} \\ &=\frac{1}{K}ch(n) \end{aligned}$$

证毕

定义2 如果在一个计算模型中，每增加递归调用的深度一层，至少使用一个时间单位，占用一个空间单位的话，则称该模型是合理的。

显然，并行计算、顺序计算、串归约、图归约等模型都是合理的。

引理6 在合理模型中，任意一个递归程序$f=E(f)$ 的内在复杂性$ch(n)$, 时间复杂性$T(n)$ 和空间复杂性$S(n)$ 满足下列条件:

$$ch(n)\leq T(n)$$
$$ch(n)\leq S(n)$$

证明: 显然。 证毕

由此，我们可以得一个与具体计算模型无关的关于程序转换所能够提高的效率的界限。

定理6　设程序$f = E(f)$ 的内在复杂度阶为$O(g(n))$，则用folding/unfolding 系统进行转换后的程序的时间复杂度$T(n)$ 和空间复杂度$S(n)$ 的阶不可能小于$O(g(n))$。即

$$T(n) \geq O(g(n))$$
$$S(n) \geq O(g(n))$$

证明:

设从$f = E(f)$，通过folding/unfolding 转换成$f = E'(f)$，其内在复杂度为$ch'(n)$，时、空复杂度分别为$T(n)$ 和$S(n)$。由引理6，$T(n) \geq ch'(n)$ 且$S(n) \geq ch'(n)$。再由定理5，$ch'(n) = O(ch(n)) = O(g(n))$。　　证毕

例3. 线性搜索算法不能转换成折半搜索算法，因为前者的内在复杂性是$O(n)$ 的，而后者是$O(\log_2 n)$。

§5. 结语

Folding/unfolding 系统的部分正确性和不完备性早已为众所周知。[12] 和[13] 等文中都给出了该系统的部分正确性的证明。[1] 文给出了本文中的例1 作为完备性的反例，但没有给出一个令人满意的证明，更没有说明是否可以通过增加一个eureka 使转换成为可能。这里我们不仅给出了形式的证明，而且定理4 说明这样的eureka 也是不存在的。

本文我们得到了一个较强的关于可转换性的必要条件，部分正确性和不完备性均是其推论，并由此得到了一个通过转换提高效率的界限。关于这样的必要条件，eureka 在转换中的作用，以及效率提高的界限的研究均未见报道。

参考文献

[1] Burstall, R. M. & Darlington, J., "A Transformation System for Developing Recursive Programs", J.ACM., Vol. 24, No. 1, Jan. 1977, pp 44~67.

[2] Pepper, P. (ed), "Program Transformation and Programming Environments", Springer-Verlag, 1984.

[3] Meertens, L. G. L. T. (ed), "Program Specification and Transformation", IFIP, North-Holland, 1987.

[4] Kleene, S. C., "Introduction to Metamathematics", North-Holland, 1951.

[5] Manna, Z., "Mathematical Theory of Computation", McGraw-Hill, 1974.

[6] Darlington, J., "A Synthesis of Several Sorting Algorithms", Acta Informatica, Vol. 11, No.1, pp 1~31, 1978.

[7] Pettorossi, A. & Burstall, R. M., "Deriving Very Efficient Algorithms for Evaluating Linear Recurrence Relations Using the Program Transformation Technique", Acta Informatica, Vol. 18, pp 181~209, 1982.

[8] Darlington, J., "The Design of Efficient Data Representations", in <Automatic Program Construction>, Biermann, A. W., Guiho, G. & Kodratoff, Y. (eds), Macmillian, 1984.

[9] Darlington, J., "Program Transformation in the ALICE Project", in [2], 1984.

[10] Darlington, J., "The Structured Description of Algorithm Derivations", in <Algorithmic Languages>, de Bakker & van Vilet (eds), IFIP, North-Holland, 1981, pp221~250.

[11] Feather, M. S., "A System for Asisting Program Transformation", ACM TOPLAS, Vol. 4, No. 1, Jan. 1982, pp 1~20.

[12] Kott, L., "Unfold/Fold Program Transformation", in <Algebraic Methods in Semantics>, Nivat, M. & Reynolds, J. C. (eds), Cambridge University Press, 1985.

[13] Sun Yongqian, et al, "Termination Preserving Problem in the Transformation of Applicative Programs", J. of Comput. Sci. & Technol., Vol. 2, No.3, pp 191~201, 1987.