数理解析研究所講究録
第547巻 1985年 128-155

On Program Transformation with Tupling Technique

by

Akihiko Koga （古賀 明彦）

Systems Development Laboratory

Hitachi, Ltd.

Abstract: In this paper, we construct a theory for an automatic program transformation with tupling technique. We introduce a purely applicative programming language and formalize a problem to find a tuple for transformation of a recursive program written in the language. We discuss the transformation scheme with the tupling given as the solution to the problem and show that the execution efficiency will be improved by the new scheme. Under a certain constraint, we describe the method to find a tuple for a given program.

1. Introduction

For the productivity and reliability of software and easiness of the maintenance, we should write a clear and easily understandable program. Such programming manner, however often implies the inefficiency of the program. To overcome the contradictory situation, several program transformation approaches [1,2,3,5,6,7,8] have been proposed as a high-level optimization method. In this paper, we discuss the program transformation with the tupling technique [1,2,3,4, 6,8].

With the tupling technique, in order to eliminate the duplicated computation done by a given recursive program, we introduce an auxiliary function that computes the values of a function for several arguments simultaneously. The following example will make clear the fundamental idea of the tupling technique.

P: Fib(n)← **if** n≤1 **then** 1 **else** Fib(n-1)+Fib(n-2)

The recursive program above computes the Fibonacci number for a given integer n. The Figure below expresses how Fib is called during the execution of Fib(5) in ordinary computation.

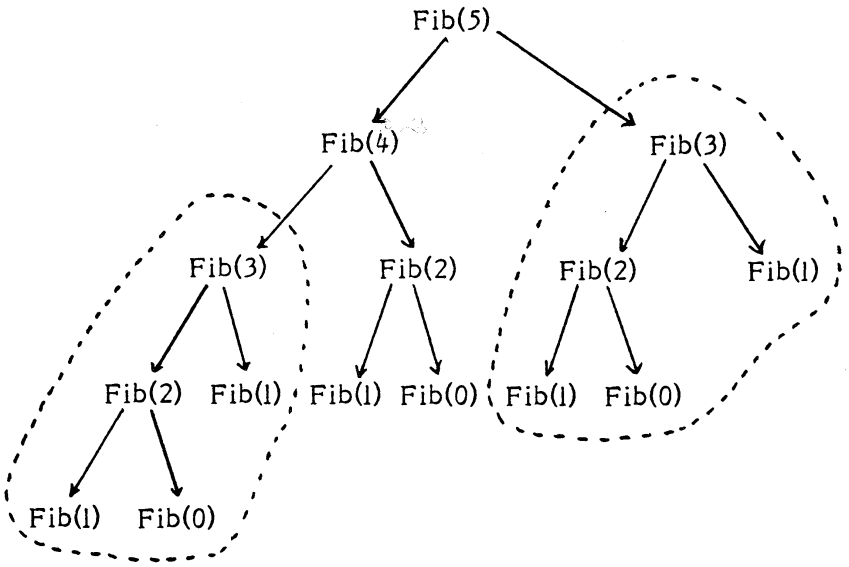


Figure 1.

In the figure above, arrows represent function calls. For example, arrows are drawn from Fib(5), one to Fib(4) and the other to Fib(3). This corresponds to the fact that in the computation of Fib(5), the function Fib is called for 4 and 3. Observing the figure, we can find that, say, Fib(3) is computed in the right branch

of Fib(5) while it is also computed in the left branch of Fib (5). Because of this phenomenon, the computation of Fib(n) requires exponential time in n. Now we introduce an auxiliary function Aux(n) that computes Fib(n) and Fib(n-1) simultaneously to eliminate the redundancy.

(1) $Aux(n) \Leftarrow \langle Fib(n), Fib(n-1) \rangle$

Here, $\langle a,b \rangle$ represents the pair (tuple) of a and b. By the simple symbolic manipulation, Aux(n) is transformed as follows:

$Aux(n) \leftarrow \langle Fib(n), Fib(n-1) \rangle$ (definition)

$Aux(n) = \langle$ **if** $n \leq 1$ **then** 1 **else** $Fib(n-1)+Fib(n-2), Fib(n-1) \rangle$

(Fib(n) in the first element of Aux(n) is expanded. This operation is called unfolding.)

= **if** $n \leq 1$ **then** $\langle 1, Fib(n-1) \rangle$ **else** $\langle Fib(n-1)+Fib(n-2), Fib(n-1) \rangle$

(the property of **if**-construct: φ(**if** p **then** a **else** b) = **if** p **then** φ(a) **else** φ(b))

(2) = **if** $n \leq 1$ **then** $\langle 1,1 \rangle$ **else** $\langle u+v, u \rangle$ **where** $\langle u,v \rangle = \langle Fib(n-1), Fib(n-2) \rangle$

Here, the expression "e_1 where $\langle u_1,\ldots,u_n \rangle = e_2$" means that to evaluate e_1 we use the values of u_i's which are results of the evaluation of e_2. Since we obtain the equation $Aux(n-1) = \langle Fib(n-1), Fib(n-2) \rangle$ from (1), the equation (2) is further transformed.

$Aux(n)$ = **if** $n \leq 1$ **then** $\langle 1,1 \rangle$ **else** $\langle u+v, u \rangle$ **where** $\langle u,v \rangle = Aux(n-1)$

If we regard this equation as the definition of Aux, we obtain the following program.

P2: { Fib(n)← u **where** <u,v> = Aux(n)
Aux(n)← **if** n≤1 **then**<1,1>else<u+v,u>**where** <u,v>=Aux(n-1)

With the program P2, the computation of Fib(n) is done in linear time in n. The computation of Aux(5) is pictures in Fig 2. Observing the figure, we can find the duplication of the computation has been removed.

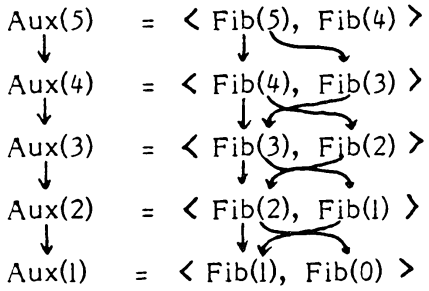
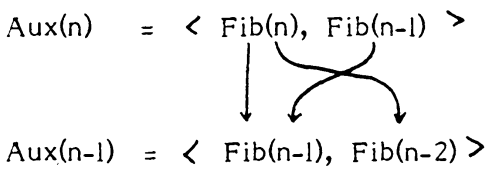


Figure 2.

The key step of the program transformation with the tupling technique is the introduction of the auxiliary function Aux(n). The reason that Aux(n) can be rewritten as the recursive function which refers only to Aux(n-1) is due to the fact that the elements of Aux(n-1), i.e., Fib(n) and Fib(n-1) can be represented by the elements of Aux(n-1),i.e.,Fib(n-1) and Fib(n-2).



In order to automate the program transformation with tupling technique, we should

discuss what tuple should be introduced as an auxiliary function and how efficient the transformed program would become. In Section 2, we introduce a programming language and postulate the problem to find a tuple for a program written in the language. In Section 3, we describe the transformation scheme and in Section 4, we discuss the relation between a given program and its tuples, and describe the method to find a tuple for the program.

2. Programming Language and Formalization of the Problem

In this section, we introduce a programming language and formalize the problem to find a tuple for a program written in the language. First, we suppose that the following sets and symbols are given.

F: a set of function symbols to be defined.

H: a set of known function symbols. Each element of H is associated with a non-negative integer called arity.

V: a set of variable symbols

X: an element of V

D: the domain of X

We define H* to be the simi-group generated from the functions of the form $D \to D$ in H. i.e.,

(i) $id \in H^*$, where $id : D \to D$ and $id(x)=x$ for any $x \in D$.

(ii) if $a \in H$ and $a : D \to D$ then $a \in H^*$

(iii) if $\rho, \sigma \in H^*$, then $\rho \cdot \sigma = \lambda x.\rho(\sigma(x)) \in H^*$

(iv) if $\sigma \in H^*$ and σ has its inverse σ^{-1}, then $\sigma^{-1} \in H^*$

In the sequel, we use Greek lower letters for the elements of H* and Roman lower letters for the elements of H.

A recursion system E on the above sets is the system of the equations of the

following form:

$$(2.1)\quad \begin{cases} f_1(X) \leftarrow t_1 \\ \quad \cdot \\ \quad \cdot \\ \quad \cdot \\ f_n(X) \leftarrow t_n \\ \text{with top function } f \in \{f_1,\ldots,f_n\} \subseteq F \end{cases}$$

The set of the function symbols $\{f_1,\ldots,f_n\}$ is denoted by F(E). t_i's are terms generated as follows:

(i) each element of V is a term

(ii) if $a \in H$ is a known function symbol with the arity m and $t'_1,\ldots,t'_m$ are terms, then $a(t'_1,\ldots,t'_m)$ is also a term.

(iii) if $\sigma \in H^*$ and $f_j \in F(E)$ then $f_j(\sigma(X))$ is a term.

(vi) if t'_1, t'_2 and t'_3 are terms then

if t'_1 **then** t'_2 **else** t'_3

is a term.

(v) if t'_1 and t'_2 are terms and $u_1,\ldots,u_m$ are variables, then

t'_1 **where** $\langle u_1,\ldots,u_m \rangle = t'_2$

is a term.

Note: (1) Although the function defined by the above system is always unary, we can manipulate a function with multiple arguments by regarding X as a vector.
(2) Symbols in F cannot nest in the right hand-side of the equations because of the generation rule (iii). Therefore we cannot manipulate, say, McCarthy's 91 function in our system.

Example 1.

The following recursion system defines the function which returns a list of Fibonacci numbers.

$$(2.2)\quad \begin{cases} \text{Flist}(n) \leftarrow \textbf{if } n \leq 0 \textbf{ then NIL else } \text{cons}(\text{Fib}(n), \text{Flist}(n-1)) \\ \text{Fib}(n) \leftarrow \textbf{if } n \leq 1 \textbf{ then } 1 \textbf{ else } \text{Fib}(n-1)+\text{Fib}(n-2) \\ \text{with top function Flist} \end{cases}$$

□

The transformation starts with the introduction of an auxiliary function $\text{Aux}(X) \leftarrow \langle g_1(\sigma_1(X)),\ldots g_m(\sigma_m(X))\rangle$ into the original system (2.1). Here, $g_i \in \{f_1,\ldots,f_n\}$ and $\sigma_i \in H^*$ $(1 \leq i \leq m)$. To transform the system (2.1) with this auxiliary function, we require that Aux(X) satisfies the following two conditions:

(A) f(X) can be represented by the element of Aux(X), i.e., by $g_1(\sigma_1(X)),\ldots,$ $g_m(\sigma_m(X))$.

(B) there are some $\rho_1, \ldots, \rho_k \in H^*$ such that each element of Aux(X) can be represented by $\text{Aux}(\rho_1(X)), \ldots, \text{Aux}(\rho_k(X))$.

In the rest of this section, we prepare some concepts to express these two conditions rigorously.

For a given recursion system E, we associate a function $D: F(E) \rightarrow P(F(E) \times H^*)$ as follows and call it the dependency of the system E (For a set A, P(A) is the power set of A).

for $f_i, f_j \in F(E)$,

$\langle f_j, \sigma\rangle \in D(E)$ iff the subterm $f_j(\sigma(X))$ appears in the term t_i, where t_i is the right-hand side of $f_i(X)$ in the system E.

We sometimes denote the dependency as $\langle D, f\rangle$ in order to emphasize the top function of the system.

Example 2.

For the system (2.2) given in Example 1, we associate D as follows:

$F(E) = \{ \text{Flist, Fib} \}$

$D(\text{Flist}) = \{ \langle \text{Flist}, \lambda n.n-1\rangle, \quad \langle \text{Fib}, \text{id}\rangle \}$

$D(Fib)=\{\langle Fib,\lambda n.n-1\rangle,\langle Fib,\ n.n-2\rangle\}$

□

The relation $D(g)=\{\langle g_1,\sigma_1\rangle,\ \ldots,\langle g_m,\sigma_m\rangle\}$ means that the value of $g(X)$ can be computed from the value of $g_1(\sigma_1(X))$, ..., $g_m(\sigma_m(X))$.

Definition

Let $A,B \subseteq H^*$, $\alpha \in H^*$ and $R \subseteq F \times H^*$.

$R =_{def} \{\langle g,\sigma\cdot\alpha\rangle \mid \langle g,\sigma\rangle \in R\}$

$RA =_{def} \bigcup_{\alpha\in A} R$

$AB =_{def} \{\rho\cdot\sigma \mid \rho\in A,\sigma\in B\}$

$A^k =_{def} \underbrace{A(A(\ldots A)\ldots)}_{k\ times}$ for $k>0$

□

Let $D_1,D_2:F(E) \to P(F(E) \times H^*)$ be two dependencies. Then the composition $D_1\cdot D_2:F(E) \to P(F(E) \times H^*)$ is defined by

$D_1\cdot D_2(g) =_{def} \bigcup_{\langle h,\sigma\rangle\in D_2(g)} D_1(h)$ for $g\in F(E)$

$D^k =_{def} \underbrace{D\ (D\ (\ldots\ D)\ldots)}_{k\ times}$ for $k > 0$

□

D^k is the dependency of the system which is obtained by k-time expansion of all the equations in the original recursion system of **D**. In the sequel, we suppose that a dependency **D** has the following property.

for any $k > 0$ and for any g $F(E)$, $\langle g,id\rangle \notin D^k(g)$

We denote the elements of H^* used in **D** by $Op(D)$. That is,

$Op(D) =_{def} \{\sigma\in H^* \mid \exists g,h\in F(E), \langle h,\sigma\rangle\in D(g)\}$

Finally, we suppose the following condition on **D**.

(C) there is a subset $G \subseteq H^*$ such that G is a group with respect to the composition and $Op(D) \subseteq G$.

This assumption is necessary to make the following definition of closure adequate.

The assumption, of course, put a restriction on our system. One of the solution to assure (C) is to modify the generation rule (iii) of term into (iii)' below, though we do not suppose this modification in the sequel.

(iii)' for $\sigma \in H^*$ such that its inverse exits,and $f_j \in F(E)$, $f_j(\sigma(X))$ is a term.

Definition

Let $\mathbf{D},f$ be a dependency and G be a group such that $Op(\mathbf{D}) \subseteq G$. The pair $\langle R,A\rangle$, $R \subseteq F(E) \times G$, $A \subseteq G$ is a closure of $\langle \mathbf{D},f\rangle$ iff R and A satisfy the following three conditions:

(i) $|R|<\omega$, $|A|<\omega$, $id \notin A^k$ for any $k>0$

(ii) $\langle f,id\rangle \in R$ or $D(f) \subseteq R$

(iii) for any $\langle g,\sigma\rangle \in R$, $\langle g,\sigma\rangle \in RA$ or $D(g)\sigma \subseteq RA$

In particular, if A is a singleton set $\{\alpha\}$, $\langle R,A\rangle$ is called linear closure and is denoted as R, .

□

Let $\langle R,A\rangle$ be a closure of $\langle \mathbf{D},f\rangle$.

$R = \{\langle g_1,\sigma_1\rangle, \ldots, \langle g_m,\sigma_m\rangle\}$,

$A = \{\alpha_1, \ldots, \alpha_k\}$

Then, R represents the auxiliary function Aux(X) to be introduced. That is Aux is defined by the following equation:

$$Aux(X) \leftarrow \langle g_1(\sigma_1(X)), \ldots, g_m(\sigma_m(X))\rangle$$

The conditions (A) and (B) correspond to the conditions (ii) and (iii) of the definition of the closure respectively.

3. Demonstration of Transformation with a Linear Closure

In this section, we describe the transformation procedure with a linear closure through the following recursion system. It is easy to extend the procedure to the

general case including the case of a non-linear closure.

$$\text{E1:}\left\{\begin{array}{l} f_1(X) \leftarrow \textbf{if } p_1(X) \textbf{ then } a_1(X) \textbf{ else } t_1 \\ \quad\vdots \\ f_n(X) \leftarrow \textbf{if } p_n(X) \textbf{ then } a_n(X) \textbf{ else } t_n \\ \text{with top function } f_1 \end{array}\right.$$

where $a_i, p_i \in H$, $F(E1)=\{f_1, \ldots, f_n\}$, and t_i is a term which does not contain **if**-construct nor **where**-construct. We suppose that a linear closure $\langle R,\alpha\rangle$ is found for the dependency $\langle D,f_1\rangle$ of the system E1.

$$R = \{\langle g_1,\rho_1\rangle, \ldots, \langle g_m,\rho_m\rangle\}$$

For simplicity, we assume that R contains $\langle f_1, id\rangle$ and $\langle g_1,\rho_1\rangle=\langle f_1,id\rangle$. We introduce the auxiliary function Aux(X):

$$Aux(X) = \langle g_1(\rho_1(X)), \ldots, g_m(\rho_m(X))\rangle$$

Since $\langle g_1,\rho_1\rangle=\langle f_1,id\rangle$, $f_1(X)$ can be represented as:

$$f_1(X) \leftarrow u_1 \ \textbf{where} \langle u_1, \ldots, u_m\rangle = Aux(X)$$

Next, we transform the definition of Aux(X) as follows:

$$Aux(X) \leftarrow \langle c_1, \ldots, c_m\rangle \ \textbf{where} \langle u_1, \ldots, u_m\rangle = Aux(\alpha(X)),$$

where c_i is the term defined as follows:

(1) Case $\langle g_i,\rho_i\rangle \in R\alpha$:

Let j be the integer such that $\langle g_i,\rho_i\rangle=\langle g_j,\rho_j\cdot\alpha\rangle$.

Then $c_i = u_j$.

(2) Case $\langle g_i,\rho_i\rangle \notin R\alpha$:

We assume that $g_i = f_{j_i}$. Then c_i is

$\textbf{if } p_{j_i}(\rho_i(X)) \textbf{ then } a_{j_i}(\rho_i(X)) \textbf{ else } t'_{j_i}$,

where t'_{i_j} is the term which is made from t_{j_i} as follows:

if a subterm $f_k(\sigma(X))$ appears in the term t_{j_i}, then by the condition (iii) of closure, $\exists\langle g_s,\rho_s\rangle \in R, \langle f_k,\sigma\cdot\rho\rangle=\langle g_s,\rho_s\cdot\alpha\rangle$ Then, the subterm $f_k(\sigma(X))$ in t_{j_i} is replaced by u_s.

As the result we obtain the system E2 below.

$$\text{E2:}\begin{cases} f_1(X) \leftarrow u_1 \ \textbf{where} \langle u_1, \ldots, u_m \rangle = Aux(X) \\ Aux(X) \leftarrow \langle c_1, \ldots, c_m \rangle \ \textbf{where} \langle u_1, \ldots, u_m \rangle = Aux(\alpha(X)), \\ \text{with top function } f_1 \end{cases}$$

Example 3.
$$\text{(3.1)}\begin{cases} Fic(n) \leftarrow \textbf{if } (n \text{ is prime}) \textbf{ then } 1 \textbf{ else } Fic(n-1)+Fic(n-2) \\ \text{with top function Fic} \end{cases}$$

$\langle R, \alpha \rangle$ is a linear closure of this system where R and α are given by:

$$R = \{ \langle Fic, id \rangle, \langle Fic, \lambda n.n-1 \rangle \}$$

$$\alpha = \lambda n.n-1$$

Now we define the auxiliary function:

$$Aux(n) \leftarrow \langle Fic(n), Fic(n-1) \rangle$$

and transform the system (3.1) into

$$\text{(3.2)}\begin{cases} Fic(n) \leftarrow u_1 \ \textbf{where} \langle u_1, u_2 \rangle = Aux(n) \\ Aux(n) \leftarrow \langle \textbf{if } (n \text{ is prime}) \textbf{ then } 1 \textbf{ else } u_1+u_2, \ u_1 \rangle \ \textbf{where} \ u_1, u_2 = Aux(n-1) \\ \text{with top function Fic} \end{cases}$$

□

Now we describe the evaluation rule of **where**-clause in detail.

$$e_1 \ \textbf{where} \langle u_1, \ldots, u_m \rangle = e_2$$

is evaluated as follows:

First, e_1 is evaluated. If some of u_i, ~~i=1~~m are required for the evaluation of e_1, then the evaluation of e_2 starts. During the evaluation of e_2, the elements of e_2 which are not required for the evaluation of e_1 are not evaluated. This rule applies recursively.

Example 4. With the system (3.2), the evaluation of Fic(6) proceeds as follows:

Fic(6)
↓
Aux(6) = **Fic(6),** Fic(5)
↓
Aux(5) = **Fic(5), Fic(4)**
↓
Aux(4) = **Fic(4),** Fic(3)
↓
Aux(3) = **Fic(3), Fic(2)**
↓
Aux(2) = **Fic(2),** Fic(1)

The elements of Aux to be evaluated are written in boldface.

That is,

1. To evaluate Fic(6), the evaluation of u_1 is required. So, Aux(6) is called and its first element is evaluated.

2. To evaluate the first element of Aux(6), since 6 is not prime, both of u_1 and u_2 are required. So, Aux(5) is called and its first and second element are evaluated.

3. To evaluate the first element of Aux(5), since 5 is prime, none of u_is is required. On the other hand, to evaluate the second element of Aux(5), the u_1 is required. As the result, Aux(4) is called and its first element is evaluated.

4. So on.

□

Under the evaluation rule, the following proposition holds.

For any $d \in D$, if $f_1(d)$ terminates with the system E1, then $f_1(d)$ of the system E2 also terminates with the same value.

The proof is included in the appendix A for the system of Fic in Example 4. The efficiency of the transformed system is expressed as follows:

We first define a function **depth**: $F(E1) \times D \rightarrow \mathbf{N}$.

$$\mathbf{depth}(f_i,d) =_{def} \begin{cases} 1 & \text{if } p_i(d) \text{ holds} \\ 1+\max\{\ \mathbf{depth}(f_j,\sigma(d)) \mid f_j(\sigma(d)) \text{ appears in } t_i\} & \text{otherwise} \end{cases}$$

depth(f_1,d) represent the depth of the computation tree when $f_1(d)$ is evaluated with the system E1. Then,

The number of recursive calls to Aux when $f_1(d)$ is evaluated with the system E2 is less or equal to $m * \mathbf{depth}(f_1,d)$.

The proof is also given in the appendix A.

Note: The situation such as written below does not occur when we transform the system E1 to E2 because of the constraint (C) written in the last section.

$$(3.3)\quad \begin{cases} c_1=u_2 \\ c_2=u_1 \end{cases}$$

If such a situation occurs, the system E2 defines a function which never terminates for any $d \in D$. There may be several methods to prohibit the situation like (3.3). In this paper, we adopt the condition (C) for the simplicity.

3. Some Properties of Closures

In this section, we discuss the relation between a dependency and its linear closures, and describe some method to find a closure for a given dependency. The next theorem states a necessary condition for the existence of a linear closure.

Theorem 1.

Let $\langle D,f\rangle$ be a dependency. If

$$\frac{\left| \bigcup_{i=0}^{k} D^i(f) \right|}{k} \to \infty \quad (\text{as } k \uparrow \infty)$$

then no linear closure exists for $\langle D,f\rangle$.

[Proof]

Let $\langle R,\alpha\rangle$ be a linear closure of $\langle D,f\rangle$ and $M=|R|$. From Lemma 1 below, we obtain

$$\bigcup_{i=0}^{k} D^i(f) \subseteq \bigcup_{i=0}^{Mk} R\alpha^i \qquad \text{for any } k>0$$

Therefore,

$$\left|\bigcup_{i=0}^{k} D^i(f)\right| \le \left|\bigcup_{i=0}^{Mk} R\alpha^i\right| \le \sum_{i=0}^{Mk} |R\alpha^i| \le M(Mk+1).$$

This means that if a linear closure of $\langle D,f\rangle$ exists, $\left| \bigcup_{i=0}^{k} D^i(f) \right|$ increases at most in linear order of k.

Lemma 1.

Let $\langle R,\alpha\rangle$ be a linear closure of $\langle \mathbf{D},f\rangle$ and $M=|R|$. Then,

for any $k>0$ and any $\langle g,\sigma\rangle \in \mathbf{D}^k(f)$, $\exists m \leq Mk$ such that $\langle g,\sigma\rangle \in R\alpha^m$

[Proof]

It can be easily proven by the induction on k.

□

Example 5.

No linear closure exists for the following system (The system computes the number of the combination of n and m).

Comb(n,m) ← **if** (0<m<n) **then** Comb(n-1,m)+Comb(n-1,m-1) **else** 1

with top function Comb

$F(E) = \{\text{Comb}\}$

$\mathbf{D}(\text{Comb}) = \{\langle \text{Comb}, \lambda\langle n,m\rangle.\langle n-1,m\rangle\rangle, \langle \text{Comb}, \lambda\langle n,m\rangle.\langle n-1,m-1\rangle\rangle\}$

Indeed we can easily show

$\mathbf{D}^i(\text{Comb}) = \{\langle \text{Comb}, \lambda\langle n,m\rangle.\langle n-i,m-j\rangle\rangle \mid 0\leq j\leq i \text{ for any } i>0\}$

and $|\bigcup_{i=0}^{k} \mathbf{D}^i(\text{comb})|$ increases in the order of k^2. Therefore, by Theorem 1, there is no linear closure for $\langle \mathbf{D},\text{Comb}\rangle$.

□

In the rest of this section, we describe some methods to find a closure for a given dependency. First, we state a method to eliminate an inessential factor from a dependency.

Theorem 2

Let G be a group such that $Op(\mathbf{D}) \subseteq G$ and suppose G is represented by a direct product $G_F \times G'$ for a finite subgroup $G_F \subseteq G$ and $G' \subseteq G$. Then, we define the projection $p:G \to G'$ as follows:

if $\sigma \in G$ is represented as $\sigma=\sigma_1\cdot\sigma_2$ for $\sigma_1 \in G_F$ and $\sigma_2 \in G'$

then $p(\sigma)=\sigma_2$

Further we define $D_p: F(E) \to P(F(E) \times G')$ as:

$$D_p(g) = \{\langle h, p(\sigma)\rangle \mid \langle h, \sigma\rangle \in D(g)\}$$

If $\langle R', A\rangle$ is a closure of $\langle D_p, f\rangle$, then $\langle R'G_F, A\rangle$ is a closure of $\langle D, f\rangle$.

[Proof]

The proof of the conditions (i) and (ii) of closure is omitted.

The condition (iii):

Suppose that $\langle g, \sigma\rangle \in R'G_F$. Then,

$\exists \langle g, \sigma_1\rangle \in R'$, $\exists \sigma_2 \in G_F$, $\sigma = \sigma_1 \cdot \sigma_2$

Case $\langle g, \sigma_1\rangle \in R'A$

$\langle g, \sigma_1 \cdot \sigma_2\rangle \in R'A\sigma_2 \subseteq R'(AG_F) = (R'G_F)A$

Case $D_p(g)\sigma_1 \subseteq R'A$

$$D(g)\sigma = D(g)\sigma_1 \cdot \sigma_2 \subseteq (D_p(g)G_F)\sigma_1 \cdot \sigma_2$$
$$= (D_p(g)\sigma_1)G_F\,\sigma_2$$
$$\subseteq (R'A)G_F$$
$$= (R'G_F)A$$

□

By this theorem, to find a closure of **D**, we first find a closure for D_p which is made from **D** by eliminating some finite group factor and then make the product of the closure and the finite group.

Example 6 The Towers of Hanoi

$$(4.1)\quad \begin{cases} \text{hanoi}(n,a,b,c) \leftarrow \textbf{if } n \le 0 \textbf{ then } \text{""} \\ \qquad\qquad\qquad \textbf{else } \text{hanoi}(n-1,a,c,b)\|\text{move}(a,c)\|\text{hanoi}(n-1,b,a,c) \\ \text{with top function hanoi} \end{cases}$$

where "" is a null string, $s_1 \| s_2$ is the concatenation of s_1 and s_2, and move(a,c) returns the string "move a disk from " || a || " to " || c.

$D(\text{hanoi})=\{ \langle \text{hanoi}, \lambda\langle n,a,b,c\rangle.\langle n-1,a,c,b\rangle\rangle, \langle \text{hanoi}, \lambda\langle n,a,b,c\rangle.\langle n-1,b,a,c\rangle\rangle\}$

If we let

$G=\{\lambda\langle n,a,b,c\rangle.\langle n+i,x,y,z\rangle \mid \langle x,y,z\rangle$ is a permutation of $\langle a,b,c\rangle$, $i\in Z\}$

$G_F=\{\lambda\langle n,a,b,c\rangle.\langle n,x,y,z\rangle \mid x,y,z$ is a permutation of $\langle a,b,c\rangle\}$

$G'=\{\lambda\langle n,a,b,c\rangle.\langle n+i,a,b,c\rangle \mid i\in Z\}$

then $Op(D)\subseteq G$ and $G=G_F \times G'$. In this case, we obtain

$D_p(\text{hanoi})=\{\langle \text{hanoi}, \lambda\langle n,a,b,c\rangle.\langle n-1,a,b,c\rangle\rangle\}$

and we can easily find a closure $\langle R',\alpha\rangle$ of $\langle D_p, \text{hanoi}\rangle$.

$R'=\{\langle \text{hanoi}, id\rangle\}$

$\alpha=\lambda\langle n,a,b,c\rangle.\langle n-1,a,b,c\rangle$

Therefore we obtain a linear closure $\langle R'G_F,\alpha\rangle$ for $\langle D, \text{hanoi}\rangle$.

□

The closure obtained by the method of theorem 2 is not always optimal. Indeed, for the system of hanoi, we will show in Appendix C that a better linear closure (the cardinality of R is less than that of R found in Example 6) exits.

Next we describe a method to construct a closure from a given dependency under certain conditions. In the following theorem, we make an assumption that the elements of Op(D) commute one another. The author has not found an effective method to construct a closure when Op(D) does not commute.

Theorem 3.

Let $\langle D,f\rangle$ be a dependency and G be a group such that $Op(D)\subseteq G$. If a finite subset A of G satisfies the following four conditions, a closure of the form $\langle R,A\rangle$ exits for $\langle D,f\rangle$.

(i) Each element of Op(D) is generated by the elements of A.

(ii) $id\notin A^i$ for any $i>0$.

(iii) the elements of A commute one another.

(iv) for each triple $\langle g,h,\rho\rangle$ such that $g,h \in F(E)$, and $\langle h,\rho\rangle \in D(g)$, we can associate an element $\beta(g,h,\rho)$ of A so that the following condition may hold.

for any $g_0,\ldots,g_k \in F(E)$ and any $\rho_1, \ldots, \rho_k \in G$ such that $g_0=g_k$ and $\langle g_i, \rho_i\rangle \in D(g_{i-1})$ for i $(1 \leq i \leq k)$

$(\rho_k \cdot \ldots \cdot \rho_1)\ \beta(g_0,g_1,\rho_1)^{-1} \cdot \ldots \cdot \beta(g_{k-1},g_k,\rho_k)^{-1} \in U_{i=0}^{\infty} A^i$

□

Since there are only finite number of triple $\langle g,h,\rho\rangle$ such that $\langle h,\rho\rangle \in D(g)$ and A is a finite set, we can check whether β exits or not by trying all the combinations.

On the construction of R, β has the following meaning:

If $\langle g,\rho\rangle \in R-RA$, then for any $\langle h,\sigma\rangle \in D(g)$, $\langle h,\sigma\cdot\rho\rangle \in RA$ must hold. We construct R so that $\langle h,\sigma\cdot\rho\rangle \in R\ \beta(g,h,\sigma) \subseteq RA$.

The Proof of Theorem 3

We show the procedure to construct R of Theorem 3. The procedure is composed of four steps.

STEP 1: We construct the tree T(D,f) from D and f according the following algorithm. Each node of T(D,f) is labeled by the element of F(E).

ALGORITHM EXPAND

Input D, f;

Output T(D,f);

BEGIN

Create a root node s and label it f;

V:= {s} ;

WHILE not empty(V) DO

v:=get(V); /* choose an element of V and assign it to the variable v */

V:=V- {v};

FOR all $g \in F(E)$ such that $\exists \tau . \langle g, \tau \rangle \in D(\text{label}(v))$ DO

Create a new node v', label it g and create an edge (v → v');

IF v' has no ancestor labeled by g THEN V:=V ∪ { v' } END IF;

END FOR

END WHILE

END

Note that the tree T(**D**,f) is uniquely determined.

Example 7

Let

$F(E) = \{ \mathbf{F}, \mathbf{G}, \mathbf{H} \}$

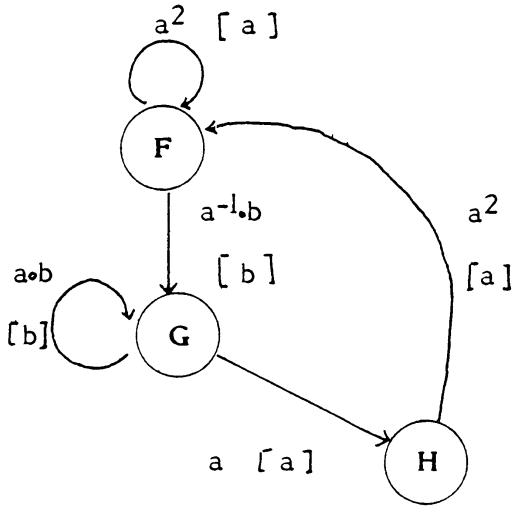
$A = \{a,b\}$ We suppose that a and b commute each other.

$\mathbf{D}(\mathbf{F}) = \{ \langle \mathbf{F}, a^2 \rangle, \langle \mathbf{G}, a^{-1} \cdot b \rangle \}$

$\mathbf{D}(\mathbf{G}) = \{ \langle \mathbf{H}, a \rangle, \langle \mathbf{G}, a \cdot b \rangle \}$

$\mathbf{D}(\mathbf{H}) = \{ \langle \mathbf{F}, a^2 \rangle \}$

D can be represented by the graph below.



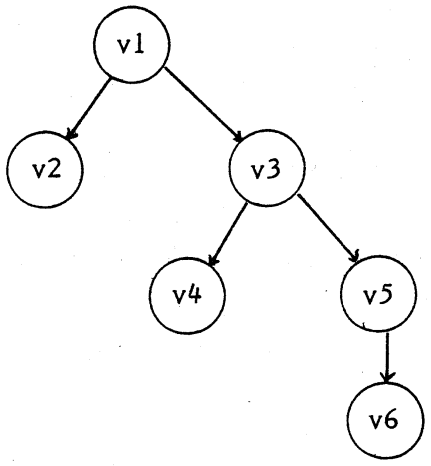
In the graph, from, say g, two arrows are drawn, one labeled a to h and the other

labeled a•b to itself. This means $D(G)=\{\langle H,a\rangle, \langle G,a\cdot b\rangle\}$. The values of β which satisfies the condition (iv) of Theorem 3 are written in [ ].

$\beta(F,F,a)=a$ $\beta(F,G,a^{-1}b)=b$ $\beta(G,H,a)=a$

$\beta(G,G,a\ b)=a$ $\beta(H,F,a^2)=a$

Applying the algorithm EXPAND to D and f, we obtain the following tree T(D,f).



label(v1)=F

label(v2)=F

label(v3)=G

label(v4)=G

label(v5)=H

label(v6)=F

□

STEP 2: We define a function

$C : Node(T(D,f)) \rightarrow P(G)$

according to the following algorithm ASSOCIATE.

ALGORITHM ASSOCIATE

Input D, T(D,f), ; /* is given by the condition (iv) of Theorem 3 */

Output C;

BEGIN

FOR all v Node(T(D,f)) DO C(v):=empty_set END FOR;

V:={s}; /* s is the root node of T(D,f) */

C(v):= { id };

WHILE not empty(V) DO

```
    v:=get(V);
    V:=V-{v};
    FOR all v' such that (v → v')∈Edge(T(D,f)) DO
      FOR all ρ∈G such that <label(v'),ρ>∈D(label(v)) DO
        FOR all σ∈C(v) DO
          C(v'):=C(v') ∪ {ρ·σ·β(label(v),label(v'),ρ)^-1
        END FOR
      END FOR
    END FOR
  END WHILE
END
```

We can easily prove the following proposition about C.

Proposition 1.

Let $(v \to v') \in Edge(T(D,f))$, label(v)=g, label(v')=h. Then,

for any $\sigma \in C(v)$, any $<h,\rho> \in D(g)$, $\rho \cdot \sigma \cdot \beta(g,h,\rho)^{-1} \in C(v')$

Example 8

We obtainn the following C, applying the algorithm ASSOCIATE to the tree T(D,f) in Example 7.

$C(v1)=\{id\}$ $C(v2)=\{a\}$

$C(v3)=\{a^{-1}\}$ $C(v4)=\{a^{-1}.b\}$

$C(v5)=\{a^{-1}\}$ $C(v6)=\{id\}$

□

STEP 3: We define a function

$B : Node(T(D,f)) \to P(G)$

by

$$B(v) = C(v)\,(\bigcup_{i=0}^{m(v)} A^i) \qquad \text{for } v \in \text{Node}(T(D,f)).$$

where m(v) is defined as follows:

Let h be the label (function symbol) of v and let $v_1, \ldots, v_n$ be the descendants of v labeled h (if no such descendant, n is 0). By the condition (iv) of Theorem 3 and the way of construction of C, there is an integer $m \geq 0$ such that $C(v_j) \subseteq C(v)(\bigcup_{i=0}^{m} A^i)$ for any j, $1 \leq j \leq n$. we define m(v) to be the smallest one among such numbers.

The following proposition can be easily proven.

Proposition 2.

(1) Let s be the root node of T(D,f). Then $id \in C(s) \subseteq B(s)$.

(2) For any node v, $B(v)\text{-}B(v)A \subseteq C(v)$.

Example 9.

B is constructed as follows for the tree T(D,f) given in Example 7.

$B(v1) = \{ id, a, b \}$

$B(v2) = \{ a \}$

$B(v3) = \{ a^{-1}, id, a^{-1}.b \}$

$B(v4) = \{ a^{-1}.b \}$

$B(v5) = \{ a^{-1} \}$

$B(v6) = \{ id \}$

For instance, the node v1 has the descendants v2 nad v6 of the same label F. The smallest integer $m \geq 0$ which satisfies $C(v2) \cup C(v6) \subseteq \cup C(v1)\,(\bigcup_{i=0}^{m} A^i)$ is 1. Indeed,

$C(v2) \cup C(v6) = \{ a \} \cup \{ id \} = \{ id, a \}$

$C(v1)\,(\bigcup_{i=0}^{1} A^i) = \{ id \}\{ id, a, b \} = \{ id, a, b \}$

B(vj) can be computed for other node vj's similarly.

□

STEP4: Finally we construct R from B.

$$R = \bigcup_{v \in Node(T(D,f))} \{ \langle label(v), \sigma \rangle \mid \sigma \in B(v) \}$$

Then, $\langle R,A \rangle$ is a closure for $\langle D,f \rangle$. Indeed the condition (i) of closure is obvious. The condition (ii) is derived from (1) of the proposition 2.

The condition (iii): Let $\langle h,\tau \rangle \in R\text{-}RA$. By the definition of R, there is a node v of T(D,f) such that

(4.2) $label(v)=h$ and $\tau \in B(v)$

Since $\langle h,\tau \rangle \in R\text{-}RA$, we obtain

(4.3) $\tau \in B(v)\text{-}B(v)A$.

Now, we prove that for any $\langle g,\sigma \rangle \in D(h)$, $\langle g,\sigma\cdot\diamondsuit \rangle \in RA$.

By the way to construct B, if v is a leaf node, then we can choose another one which satisfies (4.2). So, we assume from the first, v is not a leaf node. By the way to construct T(D,f), v has a child v' of the label g. By (4.3) and (2) of the proposition 2, we obtain

$\tau \in C(v)$.

By the proposition 1,

$\sigma\cdot\tau\cdot\beta(h,g,\sigma)^{-1} \in C(v')$

Since $\beta(h,g,\sigma) \in A$, $\sigma\cdot\tau \in C(v')A$. Therefore,

$\sigma\cdot\tau \in C(v')A \subseteq B(v')A$

Since $label(v')=g$, we obtain $\langle g,\sigma\cdot\tau \rangle \in RA$.

Thus, theorem 5 is proven.

Example 10

From the B found in Example 9, we obtain

$R=\{\langle F,id \rangle, \langle F,a \rangle, \langle F,b \rangle, \langle G,a^{-1} \rangle, \langle G,id \rangle, \langle G,a^{-1}\cdot b \rangle, \langle H, a^{-1} \rangle\}$

□

5. Conclusion and Problem

For a given recursion system, we formalized the problem of finding a tuple as the problem of finding a closure for the dependency of the system. We have shown the transformation procedure and the improvement of the efficiency when a linear closure is found for a given system. Under a certain condition, we described a method to find a closure for a given dependency.

The method described in this paper does not successfully transform the following system into a linear system.

$$(5.1)\quad \begin{cases} Fn1(X) \leftarrow \textbf{if } p(X) \textbf{ then } a(X) \textbf{ else } c(Fn1(a(b(X))), Fn2(b(X))) \\ Fn2(X) \leftarrow Fn1(a(X)) \\ \text{with top function } Fn1 \end{cases}$$

$$\mathbf{D}(Fn1)=\{\langle Fn1, a\cdot b\rangle,\ \langle Fn2, b\rangle\}$$

$$\mathbf{D}(Fn2)=\{\langle Fn1, a\rangle\}$$

For any $j>0$, no linear closure exists for the dependency $\langle \mathbf{D}^j, Fn1\rangle$. However, if we expand Fn2 in the first equation of (5.1) using the second one, we obtain the following system (5.2) which can be easily rewritten into linear one.

$$(5.2)\quad \begin{cases} Fn1(X) \leftarrow \textbf{if } p(X) \textbf{ then } a(X) \textbf{ else } c(Fn1(a(b(X))), Fn1(a(b(X)))) \\ \text{with top function } Fn1 \end{cases}$$

To compute $\mathbf{D}\cdot\mathbf{D}$ for a dependency $\mathbf{D}$ corresponds to the expansion of all the equations in the original system. The operation does not reflect the expansion of only a part of equations.

Acknowledgment

The author would like to express his deep appreciate to Professor Taiichi Yuasa

and Professor Reiji Nakajima of Research Institute for Mathematical Sciences Kyoto University for their useful advice. Through the discussion with Mr. Takashi Sakuragawa, a student there, the author could obtain a good insight to this problem. Finally, the author would like to thank Mr. Tatsuya Hagino, a student of Edinburgh University, for his kindness to prepare necessary papers from Edinburgh University.

References

1. Burstall,R.M., Darlington,J.: A Transformation System for Developing Recursive Programs. JACM 24, 1, pp44-67 (1977)

2. Cohen,N.H.: Eliminating Redundant Recursive Calls. ACM Trans. Prog. Lang. Syst. 4. 1, pp256-299 (1983)

3. Feather,M.S.: A System for Assisting Program Transformation. ACM Trans. Lang. Syst. 4. 1, pp1-20 (1982)

4. Friedman,D.P., Wise,D,S,: Functional Combination. Computer Languages. Vol. 3. pp31-35 (1978)

5. Pettorossi,A.: Improving Memory Utilization in Transforming Programs. Lecture Notes in Computer Science No. 64, pp416-425. Berlin-Heidelberg New York: Springer (1978)

6. Pettorossi,A.: Transformation of Programs and Use of "Tupling Strategy". Proc. of Informatica '77 Conference, Bled, Yugoslavia, 3-103, pp1-6 (1977)

7. Pettorossi,A., Burstall,R.M.: Deriving Very Efficient Algorithm for Evaluation Linear Recurrence Relations Using the Program Transformation Technique. Acta Informatica 18, pp181-206 (1982)

8. Wand,M.: Continuation-based Program Transformation Strategies. JACM 27, 1, pp164-180

Appendix A

We prove the equivalence of the system (3.1) and (3.2) in Section 3, and compare the efficiencies. To distinct two Fic's in the system (3.1) and (3.2), we rename the latter Fic2.

Proposition A1

Let $n \in Z$. If Fic(n) of (3.1) terminates, then Fic2(n) of (3.2) also terminates and the number of the recursive calls to Aux is less than or equal to 2*depth(Fic,n)

[Proof]

For $n \in Z$, we define a set Q(n) of sequences of elements of $Z \times \{1, 2\}$ as follows:

(1) $\langle n,1\rangle \in Q(n)$

(2) Suppose that $[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$

(i) Case $r_k=2$

$[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle, \langle q_k-1,1\rangle] \in Q(n)$

(ii) Case $r_k=1$

if q_k is not prime,

$[\langle q_1,r_1\rangle, \ldots, \langle q_k-1, j\rangle] \in Q(n)$ for $j=1, 2$

The relation $[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$ means that in the evaluation of Fic2(n) of the system (3.2), the r_kth element of $Aux(q_k)$ is evaluated.

For $s= [\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$, we define

$length(s) =_{def} k$

We can easily verify the following properties of Q(n).

Properties of Q(n)

(1) If $[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$ then $q_1=n$ and $r_1=1$.

(2) The length of the element of Q(n) is bounded if and only if Fic2(n) terminates. Particularly, if $M=\max_{s \in Q(n)}(length(s))$, then during the evaluation of Fic2(n) the number of recursive calls to Aux is less than or equal to M.

(3) Let $[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$ and $q_{i_1}, \ldots, q_{i_m}$

be q_i's whose r-part are 1. Then $m \geq k/2$.

By (1) and (3) of the properties of Q(n), and by the following lemma A1, if Fic(n) of (3.1) terminates, the length of the elements of Q(n) is bounded. Therefore, by (2) of the properties of Q(n), Fic2(n) terminates.

Lemma A1

Let $[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$ and $q_{i_1}, \ldots, q_{i_m}$ be the sequence such that $r_{i_j}=1$ for j, $1 \leq j \leq m$. Then, to evaluate $Fic(q_{i_j})$, $Fic(q_{i_{j+1}})$ is called $(1 \leq j \leq m)$.

[Proof]

Omitted

□

Proposition A2

If Fic2(n) terminates, $Fic(n)=Fic2(n)$.

[Proof]

By the next lemma A2 and the fact $\langle n,1\rangle \in Q(n)$, the first element of Aux(n) is equal to $Fic(n-1+1)=Fic(n)$.

□

Lemma A2.

If $[\langle q_1,r_1\rangle, \ldots, \langle q_k,r_k\rangle] \in Q(n)$, then during the evaluation of Fic2(n) of the system (3.2), the r_kth element of $Aux(q_k)$ is evaluated to $Fic(q_k-r_k+1)$.

[Proof]

This lemma can be proven by the induction on k.

□

Therefore, the equivalence of (3.1) and (3.2) is proven. More generally, to prove the equivalence of system E1 and E2, we construct Q(d) for $d \in D$ as a set of the sequence of the elements of $D \times \{ 1, 2, \ldots, m \}$ and express the evaluation of the elements of Aux(X).

Appendix B

We construct a better linear closure for hanoi than the one found in Example 7.

Theorem B

Let $\langle D,f\rangle$ be a dependency and suppose that there is $\langle f,\rho\rangle\in D(f)$. If

$$\exists j \quad |D^j(f)|=|D^{j+1}(f)|$$

$\langle D^j(f)\rho^{-j},\rho\rangle$ is a linear closure for $\langle D,f\rangle$.

[Proof]

It is easily shown that $D^{i+1}(f)=D^i\cdot D(f)$ for any $i\geq 0$. Therefore, we obtain

$$D^{j+1}(f)=D^j\cdot D(f)=\bigcup_{\langle g,\sigma\rangle\in D(f)} D^j(g)\sigma \supseteq D^j(f)\rho.$$

On the other hand, since $|D^{j+1}(f)| = |D^j(f)\rho|$,

$$D^{j+1}(f) = D^j(f)\rho .$$

Now we prove that $\langle D^j(f)\rho^{-j},\rho\rangle$ is a linear closure for $\langle D, f\rangle$. The conditions (i) and (ii) of closure obviously hald. To prove the condition (iii), suppose that $\langle g,\sigma\rangle\in D^j(f)\rho^{-j}$. Then,

$$\langle g,\sigma\cdot\rho^j\rangle \in D^j(f)$$

Therefore, $D(g)\sigma\cdot\rho^j \subseteq D^{j+1}(f) = D^j(f)\rho$.

Therefore, $D(g)\sigma \subseteq (D^j(f)\rho^{-j})\rho$.

□

If we check the cardinalities of i'th hold of D for the system of hanoi (4.1) for i= 1,2,3, ..., we obtain

$$|D(\text{hanoi})| = 2,\ |D^2(\text{hanoi})| = 3,\ |D^3(\text{hanoi})| = 3, \ldots$$

By theorem B, $\langle R,\alpha\rangle$ is a linear closure for the system, where

$$\alpha = \lambda\langle n,a,b,c\rangle.\langle n-1,a,c,b\rangle,$$

$$R = D^2(\text{hanoi})\alpha^{-2}$$

$$= \{\langle \text{hanoi}, \text{id}\rangle,\ \langle \text{hanoi}, \lambda\langle n,a,b,c\rangle .\langle n,b,c,a\rangle\rangle,$$

$$\langle \text{hanoi}, \lambda\langle n,a,b,c\rangle .\langle n,c,a,b\rangle\rangle\}.$$

The cardinality of R found here is 3 and so R is simpler than the one found in

Example 6. Indeed, if we define Aux as

Aux(n,a,b,c) ⇐ < hanoi(n,a,b,c), hanoi(n,b,c,a), hanoi(n,c,a,b) >,

the system (4.1) is transformed into the following system.

hanoi(n,a,b,c) ⇐ u **where** <u,v,w> = Aux(n,a,b,c)

Aux(n,a,b,c) ⇐ **if** n≤0 **then** < "", "", "" >

else < u || move(a,c) || w,

w || move(b,a) || v,

v || move(c,b) || u >

where <u, v, w> = Aux(n-1,a,c,b)

with top function hanoi