IANOV SCHEMAS AUGMENTED BY A PUSHDOWN MEMORY

Nobuki TOKURA, Tadao KASAMI and Shukichi FURUTA

Department of Information and Computer Science
Faculty of Engineering Science, Osaka University
1-1 Machikaneyama, Toyonaka 560 Japan.

Summary

An IP-schema (Ianov schema with pushdown memory) has one register just as for Ianov schemas, but the finite-state control is augmented by a pushdown memory that can only be used for storing the contents of the register or fetching the contents of the top to the register. An IP(k)-schema is an IP-schema which uses only the first k memory locations of the pushdown memory. Let $C_{IP(k)}$ (or C_{IP}) be the class of IP(k)-schemas (or IP-schemas, respectively). It is shown that $C_{IP(0)} \subsetneq C_{IP(1)} \subsetneq C_{IP(2)} \subsetneq \cdots$, and for any i, $C_{IP(i)} \subsetneq C_{IP}$ and that the strong equivalence problem for the class C_{IP} is decidable. This answers the conjecture by Indermark positively in a stronger form. It is shown that the divergence problem for deBakker-Scott schemas augmented with a memory location M for which only the following two operations are permitted: $M \leftarrow X$ and $X \leftarrow M$ is undecidable. Finally, it is shown that the strong equivalence problem for IP-schemas with some functions specified to be invertible is decidable. Thus, Chandra's result on Ianov schemas with one invertible function is strengthened.

1 Introduction

Ianov schemas (I-schemas) are very simple schemas which have only one variable (or one register)[4]. I-schemas have been generalized in several directions. For example, 1) deBakker-Scott schemas (R-schemas) are I-schemas with recursion [10, 1], 2) Luckham-Park-Paterson schemas (LPP-schemas) are schemas with a finite number of registers [8] and 3) IM-schemas are I-schemas with a second variable (or a memory location) that can be used as a memory [5].

The words "memory" and "register" are discriminated here. A register X can store a value and operations such as $X \leftarrow f(X)$ (assignment) and $q(X)$ (test) are allowed. On the other hand, memory can be used only for storing a value, and operations allowed for a memory M are $M \leftarrow X$ (store) and $X \leftarrow M$ (fetch or load).

An IP-schema (Ianov schema with pushdown memory) has one register (called the main register X in the sequel) and a pushdown memory which can be used for storing the contents of X or fetching the contents of the top to X in last-in-first-out order. An IP(k)-schema is an IP-schema which uses only the first K memory locations of the pushdown memory. By definition, an IP(0)-schema (or an IP(1)-schema) is nothing else than an I-schema (or an IM-schema, respectively). Let $C_{IP(k)}$ be the class of IP(k)-schemas and let C_{IP} be the class of IP-schemas. It is shown that $C_{IP(0)} \subsetneq C_{IP(1)} \subsetneq C_{IP(2)} \subsetneq \cdots$, and for any i, $C_{IP(i)} \subsetneq C_{IP}$ (Theorem 1. a).

Indermark conjectured that the strong equivalence problem for IP(1)-schemas is decidable [5]. The main purpose of this paper is to answer his conjecture positively in a much stronger form by showing that the strong equivalence problem for IP-schemas is decidable (Theorem 2).

Both R-schemas and IP-schemas are generalized I-schemas augmented by a pushdown memory. Indermark showed that the class $C_{IP(1)}$ of IM-schemas is not translatable into the class C_R of R-schemas and vice versa [5]. The result for the class C_{IP} of IP-schemas is that the class C_R is not translatable into the class C_{IP} (Theorem 1. b). This is because R-schemas and IP-schemas use a pushdown memory in quite different ways. Indeed, R-schemas use a pushdown memory to remember the sequence of operations to be applied, but IP-schemas store the intermediate values of the main register X with control informations in a pushdown store.

The strong-equivalence problem for <u>free</u> R-schemas is known to be decidable [1], whereas the same problem for R-schemas remains open. Here, we consider another class of (monadic) recursive schemas called Boolean recursive schemas (BR-schemas). A BR-schema is a "right-linear recursive schema" in which a simple predicate symbol may be replaced by a Call statement for a Boolean I-subschema and recursive calls for Boolean subschemas are allowed, where a Boolean subschema is an I-schema with two exits. The class C_{BR} of BR-schemas is a generalization of the class of IB-schemas which is shown by Indermark to be translatable into the class $C_{IP(1)}$ and vice versa [5]. The class C_{BR} is effectively translatable into the class C_{IP} (Theorem 3). Thus, the strong-equivalence problem for BR-schemas are decidable, although a BR schema is not necessarily free.

The ability of schemas with both features of R-schemas and IP-schemas may be the next problem. We have a negative result that the divergence problem for

R-schemas augmented with a memory unit M for which only the following two operations are permitted: $M \leftarrow X$ and $X \leftarrow M$ is undecidable (Theorem 4).

As another application of IP-schemas, a class of IP-schemas with invertible functions is considered. Chandra investigated the class C_I of IP(0)-schemas with one function specified to be invertible, and showed that the strong equivalence problem and others for those schemas are decidable [2]. His result can be strengthened to the class C_I^* of IP-schemas for which a subset F_I of function symbols is specified such that for each $f \in F_I$, there exists an f^{-1} in F_I with $\forall x$ $(f(f^{-1}(x)) = f^{-1}(f(x)) = x)$. Indeed, it can be shown that the class C_I^* can be translatable into the class C_{IP} (Theorem 5).

2 Definitions and results

The definitions and notations used in the following are Indermark's ones [5] with some necessary modifications.

Let F, Q, N, and {STA, END, PUSH, POP} be disjoint sets, where F is a set of function symbols, Q is a set of predicate (test) symbols and $N = \{1,2,3,\ldots\}$.

The set of instructions over these sets is the disjoint union of following sets:

i) $\{STA\} \times N$: START instructions
ii) $N \times \{END\}$: STOP instructions
iii) $N \times F \times N$: Assignment instructions
iv) $N \times Q \times N \times N$: Test instructions
v) $N \times \{PUSH\} \times N_E \times N$: Pushdown instructions
vi) $N \times \{POP\} \times N^{E+1}$: Popup instructions,

where $N_E = \{1,2,\ldots,E\}$ $(E \in N)$ and $N^{E+1} = N \times \ldots \times N$ (E+1 times). The nonnegative integers occuring in the first component of these instructions are called labels.

A nonempty finite set S of instructions of types described above is called an IP-schema (Ianov schema augmented with a pushdown memory) if and only if S has the following properties:

1) S has exactly one start instruction and one or more stop instructions. It is assumed without loss of generality that if $(STA, n) \in S$, then $n = 1$.
2) Different instructions have different labels in their first components.
3) For each label occuring in an instruction of S, there exists an instruction with the label as the first component.
4) All the pushdown instructions and the popup instructions are defined for a fixed E which is called the E-number of the schema S.

An interpretation of F and Q is a pair $I = (D, \Gamma)$ where D is a (nonempty) set and Γ is a mapping that gives for each $f \in F$ a total function $f^I: D \to D$ and for each $q \in Q$ a total function $q^I: D \to \{0, 1\}$. A schema S whose function and test symbols are interpreted by I is called a program P, formally $P = (S, I)$.

The semantics of a program will be given by its effect on the state vector. Let

$$Z = N \times D \times \cup_{j \geq 0} (j, (D \times N_E)^j)$$

be the set of states. The first component of the state is a label of the next instruction to be executed. The second component is the contents of the program variable (the main register) X. The third component is the contents of the stack pointer J and if the value of J is j, then the $(D \times N_E) \times \ldots \times (D \times N_E)$ (j times) denotes the contents of the pushdown memory $M(1)M(2)\ldots M(j)$. Each memory element M(i) of the pushdown memory can store an element of D in its D-field and an integer e $(1 \leq e \leq E)$ in its E-field. As explained in the sequel, the D-field is used to save the contents of the main register X and the E-field is used to store finitary control information.

Given an interpretation I, each instruction i except a start or stop instruction defines a state transition function $\Delta_i: Z \to Z$ as follows:

0) A state $(n, d, j, \langle d_1, e_1\rangle, \ldots, \langle d_j, e_j\rangle)$ is denoted shortly as z in 1) - 4).
1) For an assignment instruction $i = (n, f, n')$, Δ_i is given by
$z \to (n', f^I(d), j, \langle d_1, e_1\rangle, \ldots, \langle d_j, e_j\rangle)$.
2) For a test instruction $i = (n, p, n_0, n_1)$, Δ_i is given by
$z \to (n_{p^I(d)}, d, j, \langle d_1, e_1\rangle, \ldots, \langle d_j, e_j\rangle)$.
3) For a pushdown instruction $i = (n, PUSH, e, n')$, Δ_i is given by
$z \to (n', d, j+1, \langle d_1, e_1\rangle, \ldots, \langle d_j, e_j\rangle, \langle d, e\rangle)$, that is, the second component (the content of X) is pushed to the pushdown memory with the control information "e".
4) For a popup instruction $i = (n, POP, n_1, \ldots, n_E, n')$, Δ_i is given by
$z \to (n_{e_j}, d_j, j-1, \langle d_1, e_1\rangle, \ldots, \langle d_{j-1}, e_{j-1}\rangle)$ if $j > 0$;
$z = (n, d, 0) \to (n', d, 0)$ if $j = 0$.
5) In all other cases, Δ_i is undefined.

Let $P = (S, I)$ be a program. P defines a transition function $\Delta_p : Z \to Z$ defined by

$$\Delta_p = \cup_{i \in S} \Delta_i.$$

For $z, z' \in Z$, "$z \vdash z'$" means that $\Delta_p(z) = z'$. If $z = z'$ or $z = z_1 \vdash z_2 \vdash \ldots \vdash z_n = z'$ for some n, then we write $z \vdash^* z'$.

Without loss of generality, the initial state of P is assumed to be $z_0 = (1, d_0, 0)$ unless otherwise stated, where d_0 represents the initial value of X (or an input value) and $d_0 \in D$ as defined. In the sequel, $\overset{*}{\vdash} z$ will mean that $z_0 \overset{*}{\vdash} z$. The program P will generate a state sequence (or execution sequence) in Z by iteration of Δ_p on $z_0 = (1, d_0, 0)$. Let Comp(S, I) denote the state sequence, that is, $z_0, \Delta_p(z_0), \Delta_p^2(z_0), \ldots$. Thus, P = (S, I) computes a partial function $V_p: D \to D$ as follows:

$V_p(d_0) = d'$ if the state sequence is finite ending with some state $(n, d', j, \langle d_1, e_1\rangle, \ldots, \langle d_j, e_j\rangle)$ such that $(n, \text{END}) \in S$.

$V_p(d_0)$ = undefined, otherwise,

The definition of V_p is stated for an IP-schema, but $V_p : D \to D$ is definable in the same way for schemas of other classes. The following definition is not restricted to IP-schemas.

Definition Let S_1 and S_2 be schemas and C_1 and C_2 classes of schemas.

1) A schema S_1 is included in a schema S_2 (notation $S_1 \subset S_2$) if for every interpretation I, either both $V_{(S_1,I)}(d_0)$ and $V_{(S_2,I)}(d_0)$ are defined with the same value or $V_{(S_2,I)}$ is undefined.
2) A schema S_1 is (strongly) equivalent to S_2 (notation $S_1 \equiv S_2$) if and only if $S_1 \supset S_2$ and $S_2 \supset S_1$.
3) A schema S halts if for every interpretation I, $V_{(S,I)}(d_0)$ is defined.
4) A schema S diverges if for every interpretation I, $V_{(S,I)}(d_0)$ is not defined.
5) C_1 is translatable into C_2 if and only if for each $S \in C_1$, there is $S' \in C_2$ such that $S \equiv S'$.

Remark In the above definition, the final values of the pushdown memory are of no concern. In this sense, the pushdown memory is used only as a working storage.

If for every interpretation I, the value of the stack pointer J of a schema S does not exceed a constant k, then the schema S is called an IP(k)-schema. Let C_{IP} and $C_{IP(k)}$ be the classes of IP-schemas and IP(k)-schemas, respectively. By definition, the class $C_{IP(0)}$ is identical with the class of I-schemas and the class $C_{IP(1)}$ is identical with the class of IM-schemas studied by Indermark [5].

A deBakker-Scott schema (or R-schema) is a system of statements of the form [10]:

1) $F_i \leftarrow F_j \circ F_k$
2) $F_i \leftarrow$ if P_r then F_j else F_k
3) $F_i \leftarrow f_j$
4) $F_i \leftarrow \lambda$ (identify function),

where each F_i appearing in the system exactly once on the left hand side. F_i's are called function variables. One function variable F is distinguished as the initial symbol, and the schema is regarded as computing the function defined by the F. Let C_R be the class of R-schemas.

Indermark stated the next result [5].

Theorem(Indermark) The class of IP(1) schemas is not translatable into the class of R-schemas and vice versa.

Theorem 1 a) $C_{IP(0)} \subsetneq C_{IP(1)} \subsetneq C_{IP(2)} \subsetneq \cdots$, and for any i, $C_{IP(i)} \subsetneq C_{IP}$.

b) The class of R-schemas (C_R) is not translatable into the class of IP-schemas (C_{IP}). (The converse is also true from Indermark's theorem.)

This result can be summarized as Fig.1. In this figure, a point denotes an equivalence class which consists of all the schemas equivalent to a schema.

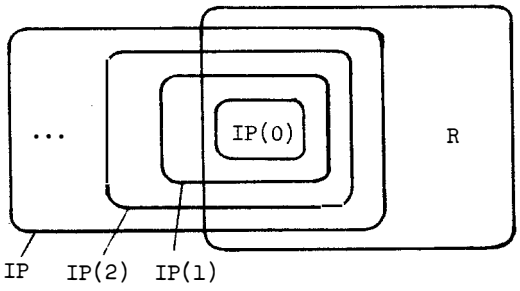

Fig.1 SUMMARY OF RESULTS

Considering the results of Glushkov, Ito and himself [3, 6], Indermark conjectured that the strong-equivalence problem of IP(1)-schemas is decidable [5]. The next result answers his conjecture affirmatively.

Theorem 2 The halting problem and the inclusion problem (therefore, the divergence problem and the strong equivalence problem) for IP-schemas are all decidable.

Definition A (monadic) Boolean Recursive schema (BR-schema) is a finite list of definitional equations:

$F_1 \leftarrow$ if P_1 then α_1 else β_1,
$\vdots$
$F_\ell \leftarrow$ if P_ℓ then α_ℓ else β_ℓ,
$G_1 \leftarrow$ if $P_{\ell+1}$ then γ_1 else δ_1,
$\vdots$
$G_k \leftarrow$ if $P_{\ell+k}$ then γ_k else δ_k,

where $F_1, \ldots, F_\ell, G_1, \ldots, G_k$ are function variables. $P_1, \ldots, P_{\ell+k}$ are in $\{q_1, \ldots, q_n\} \cup \{G_1, \ldots, G_k\}$, $\alpha_1, \beta_1, \ldots, \alpha_\ell, \beta_\ell$ are strings in $\{\lambda, F_1, \ldots, F_\ell\}^* \cdot \{f_1, \ldots, f_m\}$ and $\gamma_1, \delta_1, \ldots, \gamma_k, \delta_k$ are strings in

$\{G_1,\ldots, G_k, q_1,\ldots, q_m\}\cdot\{f_1,\ldots, f_m\}^* \cup \{\text{TRUE, FALSE}\}$. The identify function is denoted by λ. ($\{f_1,\ldots, f_m\}^*$ is the set of all finite length strings over $f_1,\ldots, f_m$ including λ.)

Let C_{BR} be the class of BR-schemas.

Theorem 3 The class C_{BR} of BR-schemas is effectively translatable into the class C_{IR} of IP-schemas.

We conjecture that the converse is also true. It may be worth to note that a BR-schema is not necessarily free. However, the strong equivalence problem of BR-schemas is decidable by Theorems 2 and 3.

The ability of schemas with both features of R-schemas and IP-schemas is stated in the next theorem.

Theorem 4 The divergence problem for R-schemas augmented with a memory unit M for which only the following two operations are permitted: M ← X and X ← M is undecidable.

This theorem follows from Theorem 4' that the divergence problem for R-schemas with a reset instruction RESET which stores a constant d into the main register is undecidable.

Let C_I^* be the class of IP-schemas with some functions specified to be invertible, where f is invertible if $\forall x\, f^{-1}(f(x)) = f(f(x)) = x$. This class contains the class C_I investigated by Chandra [2].

Theorem 5 The class C_I^* of IP-schemas with invertible functions is effectively translatable into the class C_{IP} of IP-schemas.

Chandra also investigated the class C_c of I-schemas with some functions specified to be commutative and showed that the strong-equivalence problem and others are decidable. The decision problems for IP-schemas with some commutative functions are open.

3 Example and proofs

Let Σ be an alphabet. Let Σ^* be the set of finite length strings over Σ including the empty string λ. For two strings w_1 and $w_2 \in \Sigma^*$, let $w_1 > w_2$ denote that w_2 is a proper suffix of w_1, that is, there exists non-empty string w' such that $w_1 = w'w_2$. We write $w_1 \geq w_2$ if $w_1 = w_2$ or $w_1 > w_2$. For a set A, $|A|$ will denote the number of elements in the set A. For a string w, $|w|$ will denote the length of w.

Let $L_A^S = \{n \mid (n, f_i, n') \in S, f_i \in F\}$. Similarly, let L_{PUSH}^S and L_{POP}^S denote the sets of labels of pushdown instructions and popup instructions in S, respectively. Now define $\#^S = |S| - 1$ (excluding the start instruction), $\#_A^S = |L_A^S|$ and $\#_{PUSH}^S = |L_{PUSH}^S|$. If S is understood, S can be omitted, e.g. L_A and $\#_A$.

Let T_Σ^m be the set of m-ary Σ-trees, which are finite rooted trees whose nodes are labeled with elements of Σ and each node has at most m successors. Each node in a tree of T_Σ^m can be refered by using a string of $\{1,\ldots, m\}^*$. That is, the root is represented by λ. If a string $\alpha \in \{1,\ldots, m\}^*$ represents a node α, then a string $i\cdot\alpha$ represents the i-th successor of the node α. Let $\ell(\alpha)$ denote the label of node α. For a node α in a tree t, let "the subtree below α" be the subtree whose nodes are in $\{\beta \text{ in } t \mid \beta \geq \alpha\}$. If a node α in t has no successors, then the node α is called a leaf.

For a set F of function symbols, its (monadic) Herbrand universe H_F is defined to be the set of all strings of the following form:

1) $d_0 \in H_F$

2) if $f_i \in F$ and $t \in H_F$, then $f_i t \in H_F$.

Consider here an interpretation $I = (D, \Gamma)$, where $D = H_F$ and Γ gives for each $f \in F$ a total function f^I: $H_F \to H_F$ such that $f^I(t) = ft$, $t \in H_F$ and Γ gives for each predicate symbol q a total function q^I: $H_F \to \{0, 1\}$. There is no restriction on the assignments to the predicate symbols. The interpretation I described above is called a Herbrand interpretation. To our present purpose, it suffices to consider the class of Herbrand interpretations only [8, 9]. Thus, we consider Herbrand interpretations unless otherwise stated.

Let $\tilde{T}^m_{\{0,1\}^n}$ be the set of infinite m-ary trees with each node labeled with an element of $\{0, 1\}^n = \{0,1\} \times\ldots\times \{0, 1\}$ (n times). Each tree $t \in \tilde{T}^m_{\{0, 1\}^n}$ represents an interpretation I for a function symbol set $\{f_1,\ldots, f_m\}$ and a predicate symbol set $\{q_1,\ldots, q_n\}$. Let $h: \{1,\ldots, m\}^* \to \{f_1,\ldots, f_m\}^*$ be a homomorphism such that $h(i) = f_i$ $(i = 1,\ldots,m)$. In the sequel, the symbol h will be used in this sense. The label $\ell(\alpha)$ of node α is

$$(q_1^I(h(\alpha)d_0),\ldots,q_n^I(h(\alpha)d_0)).$$

For a given interpretation I, let t_I be the tree which represents I, and for a given tree $t \in T^m_{\{0,1\}^n}$, let I_t be the interpretation represented by t. For a finite tree t, let $I|t$ denote the finite interpretation I restricted to $\{h(\alpha)d_0 \mid \alpha \in t\}$, where "$\alpha \in t$" denotes that α is a node of t.

In the sequel, let $h_0(\alpha)$ denote $h(\alpha)d_0$ and let $h_0^{-1}(d)$ be a string $\alpha \in \{1,\ldots, m\}^*$ such taht $d = h_0(\alpha)$.

Example 1

Consider a program schema $S \in C_{IP(1)}$.

S = {(STA, 1), (1, f, 2), (2, p, 4, 3), (3, END),
(4, PUSH, 1, 5), (5, g, 6), (6, p, 5, 7),
(7, POP, 1, 3)}.

Fig. 2. a represents an interpretation I. The

computation of S with I proceeds as follows:

$(1,d_0,0) \vdash (2,fd_0,0) \vdash (4,fd_0,0) \vdash (5,fd_0,1,fd_0)$
$\vdash (6,gfd_0,1,fd_0) \vdash (7,gfd_0,1,fd_0) \vdash (1,fd_0,0)$
$\vdash (2,ffd_0,0) \vdash (4,ffd_0,0) \vdash (5,ffd_0,1,ffd_0)$
$\vdash (6,gffd_0,1,ffd_0) \vdash (5,gffd_0,1,ffd_0)$
$\vdash (6,ggffd_0,1,ffd_0) \vdash (7,ggffd_0,1,ffd_0)$
$\vdash (1,ffd_0,0) \vdash (2,ffd_0,0) \vdash (3,fffd_0,0),$

where the E-field is omitted for simplicity. Thus,

$$V_{(s,I)}(d_0) = fffd_0.$$

The computation of S with the interpretation I can be intuitively visualized as a move on a tree (Fig.2.b)).

The schema S will not halt for an interpretation I' such that $P^{I'}(g^i fd_0) = 0$ for all $i \geq 0$.

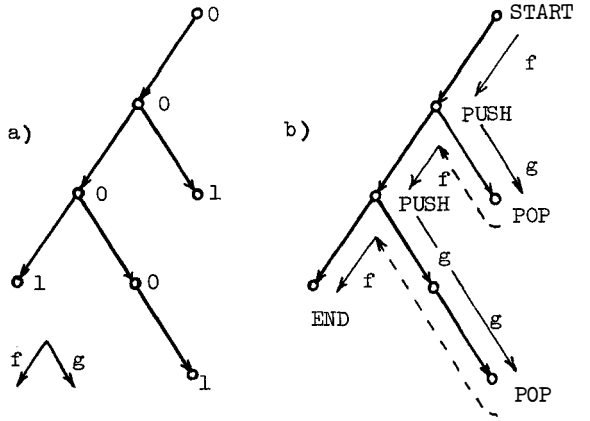

Fig.2 a) A tree representing an interpretation I,
b) A computation of S with I.

For an IP schema, the following lemmas hold.

Lemma 1. For each state $(n,d,j,<d_1,e_1>,\ldots,<d_j,e_j>)$ which occurs in a state sequence Comp(S, I),

$$d_0 \leq d_1 \leq d_2 \leq \cdots \leq d_j \leq d. \qquad (1)$$

Proof. For the initial state $(1,d_0,0)$, (1) holds. If (1) holds for the k-th state $(n,d,j,<d_1,e_1>,\ldots,<d_j,e_j>)$ in Comp(S,I), then it can be shown that (1) holds for the (k+1)-st state. For example, if $(n,PUSH,e,n') \in S$, then the next state will be $(n',d,j+1,<d_1,e_1>,\ldots,<d_j,e_j>,<d,e>)$ and (1) holds. Similar arguments hold for other instructions.

By the similar proof method, the next lemma can be shown.

Lemma 2. If $V_{(S,I)}(d_0) = t$, then

$$\begin{aligned} d_1 &\leq t \quad \text{if } j > 0, \\ d &\leq t \quad \text{if } j = 0 \end{aligned} \qquad (2)$$

for each state $(n,d,j,<d_1,e_1>,\ldots,<d_j,e_j>)$ in Comp (S,I).

Let $Sup(S,I) = \{d \mid (n,d,j,<d_1,e_1>,\ldots,<d_j,e_j>) \in Comp(S,I)\}$, that is, Sup(S,I) is the set of all the values of the main register X evaluated in the computation of S with an interpretation I.

Lemma 3 If an interpretation I' gives for every predicate symbol q a predicate $q^{I'}$ such that $q^{I'}(t) = q^I(t)$ for $t \in Sup(S, I)$, then

$$Comp(S, I) = Comp(S, I').$$

This follows from the fact that an IP-schema can test only the value of the main register X.

Remark. Lemma 3 holds for R-schemas too if Sup(S, I) is considered to be the set of all the values evaluated in the computation.

Proof of Theorem 1. a:

In this paper, it is shown only that $C_{IR(1)} \subsetneq C_{IP(2)}$ by using the following schema S. The rest of the proof of Theorem 1. a can be proven similarly.

S = {(STA, 1), (1, g, 2), (2, p, 4, 3), (3, END),
(4, PUSH, 5), (5, f, 6), (6, p, 8, 7),
(7, POP, 1, 3), (8, PUSH, 9), (9, g, 10),
(10, p, 9, 11), (11, POP, 5, 3)}.

This schema is in $C_{IP(2)}$. Assume that there exists an IP(1)-schema S' such taht $S \equiv S'$. Let I be the following interpretation:

1) there exists an integer k such taht $p^I(g^k d_0) = 1$ and if $t < g^k d_0$ then $p^I(t) = 0$,
2) for each $i < k$, there exists an integer ℓ_i such that $p^I(f^{\ell_i} g^i d_0) = 1$ and if $t < f^{\ell_i} g^i d_0$, then $p^I(t) = 0$,
3) for $i < k$ and $j < \ell_i$, there exists an integer $h_{j,i}$ such that $p^I(g^{h_{j,i}} f^j g^i d_0) = 1$,
4) for at least one $\bar{i} < k$, $\ell_{\bar{i}} > \#_A^{S'} \times$ (the E-number of S').

Clearly, $V_{(S,I)}(d_0) = g^k d_0$ and $V_{(S',I)} = g^k d_0$. If S' does not evaluate $g^{h_{j'i'}} f^{j'} g^{i'} d_0$ for some i', j' and h such that $0 < i' < k$, $0 < j' < \ell_i$ and $h > 0$, there exists an interpretation I' for which

$$V_{(S,I')}(d_0) = \text{undefined but } V_{(S',I')} = g^k d_0. \qquad (3)$$

Indeed, I' can be chosen to be identical to I except that $p^{I'}(g^{h_{j'i'}} f^{j'} g^{i'} d_0) = 0$ for all $h \geq 0$. Then, by Lemma 3, (3) follows. Therefore, S' should evaluate all the elements of Sup(S,I). Since $V_{(S',I)}(d_0) = g^k d_0$, the contents of the memory location M(1) must be always of the form $g^i d_0$ $(i = 0,\ldots, k)$ by Lemma 2. Consider the evaluation of $g^{h_{\bar{i}j}} f^j g^{\bar{i}} d_0$ $(h > 0$, $\ell_{\bar{i}} > j > 0)$. After the evaluation of $g^{h_{\bar{i}j}} f^j g^{\bar{i}} d_0$ $(h > 0)$ for some j, the contents of the register of S' must return to the value of the form $g^i d_0$ $(i \leq \bar{i})$ by a POP instruction to evaluate $g^{h_{\bar{i}j'}} f^{j'} g^{\bar{i}} d_0$ $(h > 0)$ for $j' \neq j$. Then, S' should evaluate $fg^{\bar{i}} d_0$ by the same assignment instruction with the same control information in M(1) because of the selection of $\ell_{\bar{i}}$. Thus, the computation of S' repeats itself cyclically thereafter and $V_{(S',I)}(d_0)$ should be undefined. Hence, a contradiction.

Notation. Let $z_1 = (n_1, d^1, j_1, \langle d_1, e_1\rangle, \ldots, \langle d_{j_1}, e_{j_1}\rangle)$ and $z_2 = (n_2, d^2, j_2, \langle d'_1, e'_1\rangle, \ldots, \langle d'_{j_2}, e'_{j_2}\rangle) \in \mathrm{Comp}(S, I)$ and $z_1 \vdash z_2$.

1) $z_1 \vdash_d z_2$ means that $d^1 \geq d$ and $d^2 \geq d$,
2) $z_1 \vdash_{\nabla d} z_2$ means that $d = d^2 = i \cdot d^1$ for some $i \in \{1, \ldots, m\}$.
3) $z_1 \vdash_{d\Delta} z_2$ means that $d^1 \geq d > d^2$.
4) $z_1 \vdash_d^* z_2$ means that $z_1 = z_2$ or there exist $z^1, z^2, \ldots, z^\ell$ such taht $z_1 = z^1 \vdash_d z^2 \vdash_d \cdots \vdash_d z^\ell = z_2$.
5) $\vdash_{\nabla d}^* z_2$ means that there exists a state z_1 such that $(1, d_0, 0) \vdash^* z_1 \vdash_{\nabla d} z_2$.

Directly from the definition, we have the following lemma.

Lemma 4 Let

$z_1 = (n, d, j, \langle d_1, e_1\rangle, \ldots, \langle d_j, e_j\rangle) \in \mathrm{Comp}(S, I)$,
$z_2 = (n', d', j', \langle d'_1, e'_1\rangle, \ldots, \langle d'_{j'}, e'_{j'}\rangle) \in \mathrm{Comp}(S, I)$.

1) If $\vdash_{\nabla d} z_1 \vdash_d^* z_2$, then the state z_1 is reached by executing an assignment instruction $(n'', f_i, n) \in S$, and $d_j < d \leq d'$, $j' \geq j$ and $d'_i = d_i$ $(1 \leq i \leq j)$.
2) If $z_1 \vdash_{e\Delta} z_2$ for some e $(d \geq e > d')$, then $(n, \mathrm{POP}, n', n'') \in S$, $j > 0$ and $j' = j-1$, $d' = d_j$.

Sup(S, I) may or may not be infinite even if Comp(S, I) is infinite.

Lemma 5 Assume that Sup(S, I) is finite. Then, Comp(S, I) is infinite if and only if a) there exists a state which appears twice in Comp(S, I) or b) there exists a state $\varepsilon = (n, d, j, d_1, \ldots, d_j)$ in Comp(S, I) such that

$$d_{i-1} < d_i = \ldots = d_j = d \text{ and } j = i + \#_{\mathrm{PUSH}} + 1. \quad (4)$$

Proof. We will show only that if there is a state with (4) in Comp(S, I), then Comp(S, I) will be infinite. For the state ε, we can find the following states α, β, γ and δ in Comp(S, I):

$\alpha = (n', d, i, \langle d_1, e_1\rangle, \ldots, \langle d_i, e_i\rangle)$
$\beta = (n'', d, k, \langle d_1, e_1\rangle, \ldots, \langle d_k, e_k\rangle)$
$\gamma = (n'', d, k', \langle d_1, e_1\rangle, \ldots, \langle d_{k'}, e_{k'}\rangle)$
$\delta = (n''', d, j-1, \langle d_1, e_1\rangle, \ldots, \langle d_{j-1}, e_{j-1}\rangle)$

Such that

1) $\alpha \vdash_d^* \beta \vdash_d^* \gamma \vdash_d^* \delta \vdash_d^* \varepsilon$
2) $n', n'', n''' \in L_{\mathrm{PUSH}}$,
3) $d_1 \leq d_2 \leq \ldots \leq d_{i-1} < d_i = \ldots = d_j = d$,
4) $i \leq k < k' \leq j-1 = i + \#_{\mathrm{PUSH}}$,
5) for each state in the transition $\beta \vdash_d^* \gamma$, its third component is larger than k.

In this case,

$\alpha \vdash_d^* \beta = (n'', d, k, \langle d_1, e_1\rangle, \ldots, \langle d_k, e_k\rangle)$
$\vdash_d^* \gamma = (n'', d, k', \langle d_1, e_1\rangle, \ldots, \langle d_{k'}, e_{k'}\rangle)$
$\vdash_d^* (n'', d, k+2\ell, \langle d_1, e_1\rangle, \ldots, \langle d_{k+2\ell}, e_{k+2\ell}\rangle)$
$\vdash_d^* \ldots,$

where $\ell = k'-k$. Thus, the computation never halts.

It remains to show that α, δ, β and γ exist.

α: There is at least one state such that a) its first component is an element of L_{PUSH}, b) the second component is d, c) the third component is i for which $d_{i-1} < d_i = d$ and d) $\alpha \vdash_d^* \varepsilon$. Choose arbitrary one as α.

δ: Choose the state (the first state if there are two or more candidates) such that a) and b) are the same as a) and b) of α, respectively, c) the third component is j-1 and d) $\alpha \vdash_d^* \delta \vdash_d^* \varepsilon$.

ζ_ℓ $(i \leq \ell < j-1)$: Choose the last state such that a) and b) are the same as a) and b) of α, respectively, c) the third component is ℓ and d) $\alpha \vdash_d^* \zeta_\ell \vdash_d^* \delta$,

Then,

$$\alpha \vdash_d^* \zeta_i \vdash_d^* \zeta_{i+1} \vdash_d^* \cdots \vdash_d^* \zeta_{j-2} \vdash_d^* \delta,$$

in which there are at least two states with the same label in the first component. It can be verified that for each state in the transition $\zeta_\ell \vdash_d^* \zeta_{\ell'}$, $(i \leq \ell < \ell' \leq j-1)$, the third component is larger than ℓ except the state ζ_ℓ itself. Thus, β and γ as described above can be chosen from $\{\alpha, \zeta_i, \zeta_{i+1}, \ldots, \zeta_{j-2}, \delta\}$.

Theorem 1. b The class of R-schemas (C_R) is not translatable into the class of IP-schemas (C_{IP}).

Proof. Consider the following R-schemas S_b:

S_b: F ← if p then G·F·f else λ ,
G ← if p then G else f.

Now define

$$I(S_b) = \{p^I(d_0)p^I(fd_0)\ldots p^I(f^\ell d_0) \mid (S_b, I) \text{ halts and } \ell \geq |\mathrm{Sup}(S_b, I)|\}$$

It can be easily verified that

$$I(S_b) = \{1^n 0^n \mid n \geq 0\} \cdot \{0, 1\}^*.$$

On the other hand, an IP-schema with one function symbol f can be simulated by a finite automaton with marks (mfa). Given an IP-schema S, we construct the corresponding mfa M as follows:

IP-schema S	mfa M
(STA, 1)	q_1 is the initial state.
(n, END)	q_n is an accepting state.
(n, f, n')	move the head right and go to $q_{n'}$
(n, p, n_0, n_1)	if the head reads 0, then go to q_{n_0} else go to q_{n_1}
(n, PUSH, e, n')	put a mark e on the square the head points to and go to $q_{n'}$ (if the number of marks on the square exceeds $\#_{\mathrm{PUSH}}$ go to a rejecting state q_∞
(n, POP, n_1,..., n_E, n')	if there is no mark on the tape then go to q_n, else move the head to the nearest square with a mark and delete a mark in Last-in-first-out order and if the deleted mark is e, then go to q_{n_e}.

The head of M corresponds to the main register of S and the input tape for M corresponds to the interpretation for S. If S is in state $(n, f^{i}d_0, j, \langle f^{i_1}d_0, e_1\rangle, \ldots, \langle f^{i_j}d_0, e_j\rangle)$, then M is in state q_n with the head pointing to the (i+1)-st square from the left end, while marks $e_\ell, \ldots, e_{\ell'}$ are placed on the $(i_\ell+1)$-st square if $i_\ell = i_{\ell+1} = \ldots = i_{\ell'}$. If a square has $\#_{PUSH}$ marks or more, then S will not halt by Lemma 5. Thus, the number of marks on each square is bounded. A tape which M accepts represents an interpretation for which the IP-schema halts. Here, it is known [7, 12] that for each mfa, there exists an equivalent finite automaton. Thus, $I(S_b)$ can not be recognizable by any mfa.

Theorem 2 The halting problem and the inclusion problem (therefore, the divergence problem and the strong equivalence problem) for IP-schemas are all decidable.

Proof. We will show that for given schemas S_1 and S_2, there exist C_1, C_2, R, D_1 and D_2 which are sets of trees explained below such that

1) S_1 halts if and only if $D_1 = \phi$,
2) $S_1 \not\subseteq S_2$ if and only if $(C_1 \cap C_2 \cap R) \cup (C_1 \cap D_2) = \phi$.

Next, we will show that there exist tree automata M_{C_1}, M_{C_2}, M_R, M_{D_1} and M_{D_2} which recongnize C_1, C_2, R, D_1 and D_2, respectively. It is known [11] that the sets recognized by tree automata are closed under the Boolean operations and that the emptiness problem is decidable. Thus, the problems are decidable.

Intuitively, each tree in $C_1 \cap C_2 \cap R$ (if any) represents an interpretation I and the information on the behavior of S_1 and S_2 for which $V_{(S_1,I)}(d_0)$ and $V_{(S_2,I)}(d_0)$ are both defined but $V_{(S_1,I)}(d_0) \neq V_{(S_2,I)}(d_0)$. Each tree in $C_1 \cap D_2$ (if any) represents an interpretation I and the information on the behavior of S_1 and S_2 for which (S_1, I) will halt but (S_2, I) will never halt.

Let $F = \{f_1, \ldots, f_m\}$ and $Q = \{q_1, \ldots, q_n\}$ denote the set of function symbols and predicate symbols used in S_1 and S_2, respectively. Let U_ℓ (ℓ = 1, 2) be the set of functions

$u^\ell: L^\ell_* \times \{0, 1\} \cup \{(1,0)\} \to L^\ell_{POP} \cup \{*, **, \infty, -\}$

with $L^\ell_* = \{n \mid (n', f_i, n) \in S_\ell\}$. The symbols *, **, ∞ and - will be explained in the sequel. $|U_1|$ and $|U_2|$ are finite by definition. We consider here a set of trees T^m_U in place of $\tilde{T}^m_{\{0,1\}^n}$, where $U = \{0, 1\}^n \times (U_1 \cup \{\lambda\}) \times (U_2 \cup \{\lambda\})$. The label $\ell(\alpha)$ of node α of a tree in T^m_U consists of three parts: $v(\alpha) \in \{0,1\}^n$, $u^1_\alpha \in U_1 \cup \{\lambda\}$ and $u^2_\alpha \in U_2 \cup \{\lambda\}$. $v(\alpha)$ represents an interpretation as in $\tilde{T}^m_{\{0,1\}^n}$. The auxiliary information u^1_α and u^2_α are used to detect "divergence" in a finite tree.

C_ℓ (ℓ = 1, 2) is a subset of T_U which is defined by considering the behavior of S_ℓ.

Definition For an interpretation I, let $T_C(S_\ell, I)$ be the set of trees t which satisfy the following conditions 1) - 5). And let $C_\ell = \bigcup_I T_C(S_\ell, I)$.

1) (S_ℓ, I) eventually halts.
2) For each node α of t,
$$v(\alpha) = (q^I_1(h_0(\alpha)), \ldots, q^I_n(h_0(\alpha))).$$
3) Let $\tilde{t}$ be the set of nodes of t which are not leaves. Then,
$$\tilde{t} \supset \{h_0^{-1}(d) \mid d \in \mathrm{Sup}(S_\ell, I)\}.$$
4) For each node α($\neq \lambda$), $u^\ell_\alpha(n, z)$ must satisfy the following conditions a0) -a2), where $n \in L^\ell_*$ and $z \in \{0, 1\}$. Let $\bar{j} = [j = 0 \to 0; T \to 1]$.

a0) $u^\ell_\alpha(n, \bar{j}) = -$ if and only if there are no states $(n, h_0(\alpha), j, \langle d_1,e_1\rangle, \ldots, \langle d_j,e_j\rangle)$ such that $(1, d_0, 0) \vdash^* (n, h_0(\alpha), j, \langle d_1,e_1\rangle, \ldots, \langle d_j,e_j\rangle)$ in the computation of (S_ℓ, I).

a1) $u^\ell_\alpha(n, 1) = n' \in L^\ell_{POP}$ if and only if in the computation of (S_ℓ, I), $j > 0$ and
$$\vdash^*_{\forall h_0(\alpha)} (n, h_0(\alpha), j, \langle d_1,e_1\rangle, \ldots, \langle d_j,e_j\rangle)$$
$$\vdash^*_{h_0(\alpha)} (n', d', j, \langle d_1,e_1\rangle, \ldots, \langle d_j,e_j\rangle)$$
$$\vdash_{h_0(\alpha)\Delta} (n_{e_j}, d_j, j-1, \langle d_1,e_1\rangle, \ldots, \langle d_{j-1},e_{j-1}\rangle)$$
By lemma 4, $d_j < h_0(\alpha)$ and $(n', POP, n_1, \ldots, n_E, n'') \in S_\ell$.

a2) $u^\ell_\alpha(n, \bar{j}) = **$ (or *) if and only if in the computation of (S_ℓ, I),
$$\vdash^*_{\forall h_0(\alpha)} (n, h_0(\alpha), j, \langle d_1,e_1\rangle, \ldots, \langle d_j,e_j\rangle)$$
$$\vdash^*_{h_0(\alpha)} (n', d', j', \langle d_1,e_1\rangle, \ldots, \langle d_{j'},e_{j'}\rangle),$$
where $(n', END) \in S_\ell$ and $d' = h_0(\alpha)$ (or $d' > h_0(\alpha)$, respectively.)

5) At the root λ, $u^\ell_\lambda(1, 0) = **$ (or *) if and only if $\vdash^* (n, d, j, \langle d_1,e_1\rangle, \ldots, \langle d_j,e_j\rangle)$, where $(n, END) \in S_\ell$ and $d = d_0$ (or $d > d_0$, respectively.)

R is the set of trees t which no node α such that $u^1_\alpha(n, z) = u^2_\alpha(n', z') = **$ for some n, z, n' and z'. By the definition of C_1, C_2 and R, for an interpretation I,

$V_{(S_1, I)}(d_0) = d_1$, $V_{(S_2, I)}(d_0) = d_2$ and $d_1 \neq d_2$ if and only if there is a tree t in $C_1 \cap C_2 \cap R$ such that $v(\alpha) = (q^I_1(h_0(\alpha)), \ldots, q^I_n(h_0(\alpha)))$ for each node α in t, $u^1_{h_0^{-1}(d_1)}(n, z) = **$ for some n and z, and $u^2_{h_0^{-1}(d_2)}(n', z') = **$ for some n' and z'.

Next, D_ℓ is defined as a union of two sets D^f_ℓ and D^i_ℓ.

Definition For an interpretation I, let $T_{Df}(S_\ell, I)$ be the set of trees t which satisfy the following

conditions 1) - 5). And let $D^f_\ell = \bigcup_I T_{Df}(S_\ell, I)$.

1) (S_ℓ, I) never halts but $Sup(S_\ell, I)$ remains finite.
2) Same as 2) in the definition of c_ℓ.
3) Same as 3) in the definition of c_ℓ.
4) For each node α ($\neq \lambda$), $u^\ell_\alpha(n, z)$ must satisfy the following conditions b0) - b2), where $n \in L^\ell_*$ and $\bar{j} = [j = 0 \to 0; T \to 1]$.
b0) Same as a0) in the definition of C_ℓ.
b1) Same as a1) in the definition of C_ℓ.
b2) $u^\ell_\alpha(n, \bar{j}) = \infty$ if and only if in the computation of (S_ℓ, I), either

$\vdash^*_{\nabla h_0(\alpha)} (n, h_0(\alpha), j, \langle d_1,e_1\rangle,\ldots,\langle d_j,e_j\rangle)$
$\vdash^*_{h_0(\alpha)} (n', d', j', \langle d_1,e_1\rangle,\ldots,\langle d_j,e_j,\rangle)$
$\vdash^*_{h_0(\alpha)} (n', d', j', \langle d_1,e_1\rangle,\ldots,\langle d_j,e_j,\rangle)$

that is, the same state appears twice (Lemma 5.a) or

$\vdash^* (n, h_0(\alpha), j, \langle d_1,e_1\rangle,\ldots, \langle d_j,e_j\rangle)$
$\vdash^*_{h_0(\alpha)} (n', d', j', \langle d_1,e_1\rangle,\ldots, \langle d_j,,e_j,\rangle)$

and there exists a $k > j$ such taht

$$d' = d_k = d_{k+1} = \ldots = d_j,,\ j' > k + \#_{PUSH},$$

that is, the case of Lemma 5.b.
5) At the root, $u^\ell_\lambda(1, 0) = \infty$.

Next, a tree in D^i_ℓ will represent a computation with infinite $Sup(S_\ell, I)$ on the basis of the following Lemmas 6 and 7.

Lemma 6 Assume that Sup(S, I) is infinite. Consider a tree t such that $\alpha \in t$ if and only if $h_0(\alpha) \in Sup(S, I)$. The tree t has exactly one infinite path.

Proof. Without loss of generality, assume that there are two infinite paths P_1 and P_2 branching at a node α. If there is a transition

$\vdash^* \beta = (n, d, j, \langle d_1,e_1\rangle,\ldots,\langle d_j,e_j\rangle$
$\vdash^* \gamma = (n',d',j',\langle d'_1,e'_1\rangle,\ldots,\langle d'_j,,e'_j,\rangle)$,

where $h_0^{-1}(d)$ in P_1 , $h_0^{-1}(d) > \alpha$; $h_0^{-1}(d')$ in P_2 , $h_0^{-1}(d') > \alpha$, then there exists a state δ such that $\beta \vdash^* \delta \vdash^* \gamma$ and $\delta = (n",d",j",\langle d"_1,e"_1\rangle,\ldots,\langle d"_j",e"_j",\rangle)$ with $d" \leq h_0(\alpha)$. In this case, $j" \leq |\alpha|\cdot\#_{PUSH}$ by Lemma 1 and Lemma 5. Thus, there are only finite candidates for such a δ. Therefore, the main register X can change its value from d with $h_0^{-1}(d)$ in P_1 and $h_0^{-1}(d) > \alpha$ to d' with $h_0^{-1}(d')$ in P_2 and $h_0^{-1}(d') > \alpha$ and vice versa only finite times considering Lemma 5. a. Hence one of the paths P_1 and P_2 must be finite.

Definition For an interpretation I, let $T_{Di}(S_\ell, I)$ be the set of trees t which satisfy the following conditions 1) - 6). And let $D^i_\ell = \bigcup_I T_{Di}(S_\ell, I)$.

1) (S_ℓ, I) never halts and $Sup(S_\ell, I)$ is infinite.
2) Same as 2) in the definition of C_ℓ.
3) There exists a node α_0 in t which is called a c-node. This node α_0 is on the unique infinite path of t whose existence is assured by Lemma 6. If $d \in Sup(S_\ell, I)$ with $d \neq h_0(\alpha_0)$, then $\alpha = h_0^{-1}(t) \in t$ and α is **not** a leaf. The node α_0 may or may not be a leaf.
4) For each node α ($\neq \lambda$), $u^\ell_\alpha(n, z)$ must satisfy the following condition c0)- c2), where $n \in L^\ell_*$ and $\bar{j} = [j = 0 \to 0; T \to 1]$.
c0) Same as a0) in the definition of C_ℓ, except that for each node β with $\beta > \alpha_0$, u^ℓ_β is λ.
c1) Same as a1) in the definition of C_ℓ.
c2) $u^\ell_\alpha(n, \bar{j}) = \infty$ if and only if in the computation of (S_ℓ, I), there is no state $\delta = (n', d', j', \langle d'_1,e'_1\rangle,\ldots,\langle d'_j,,e'_j,\rangle)$ such that
$\vdash^* (n, d, j, \langle d_1,e_1\rangle,\ldots,\langle d_j,e_j\rangle) \vdash^* \delta$
and $d' < d$.
5) At the root, $u^\ell_\lambda(1, 0) = \infty$.
6) There is at least one node β_0 such that
$\beta_0 < \alpha_0$, $v(\alpha_0) = v(\beta_0)$ and $u^\ell_{\alpha_0} = u^\ell_{\beta_0}$.

Lemma 7 If $Sup(S_\ell, I)$ is infinite, then there exists a tree in D^i_ℓ. Conversely, there exists a computation with infinite $Sup(S_\ell, I)$ for a tree in D^i_ℓ.

Proof. The first part may be direct by definition. The condition 6) can be satisfied because $\{0, 1\}^n \times (U_1 \cup \{\lambda\}) \times (U_2 \cup \{\lambda\})$ is finite. The last part: Let t' be a subtree below the node β_0 whose existence is assured by condition 6). Define $\Gamma = \{\gamma \mid \gamma\beta_0 \in t'\}$. We write $\alpha_0 = \gamma_0\beta_0$ ($\gamma_0 \neq \lambda$). Now an interpretation I' is defined from the label $v(\alpha)$ of t as follows:

a) If $\alpha \in t$, then $(q_1^{I'}(h_0(\alpha)),\ldots, q_n^{I'}(h_0(\alpha))) = v(\alpha)$.
b) For $\gamma \in \Gamma$ and $k > 0$,
$(q_1^{I'}(h_0(\gamma\gamma_0^k\beta_0)),\ldots, q_n^{I'}(h_0(\gamma\gamma_0^k\beta_0))) = v(\gamma\beta_0)$.

Then, $Sup(S_\ell, I')$ will be infinite, because for an $n \in L^\ell_*$, $u^\ell_{\beta_0}(n, \bar{j}) = \infty$, the computation (S_ℓ, I') will proceed as follows:

$\vdash^* (n, h_0(\beta_0), j, \langle d_1,e_1\rangle,\ldots,\langle d_j,e_j\rangle)$
$\vdash^* (n, h_0(\gamma_0\beta_0), j', \langle d_1,e_1\rangle,\ldots,\langle d_j,,e_j,\rangle)$
$\vdash^* (n, h_0(\gamma_0^2\beta_0), j", \langle d_1,e_1\rangle,\ldots,\langle d_j",e_j"\rangle)$
$\vdash^* \cdots$

which never halts.

Now let us explain the construction of tree automaton M_{C_1}. Other automata can be constructed similarly. In the sequel, it may not be necessary to distinguish between S_1 and S_2. Thus, C_ℓ, D^i_ℓ, D^f_ℓ, u^ℓ_α, S_ℓ, L^ℓ_* and L^ℓ_{POP} will be simply written as C, D^i, D^f, u_α, S, L_* and L_{POP}, respectively.

A point in the construction of M_C is that the check of condition 4 in the definition of C can be done by examining the consistency of the label u_α **at** node α and the labels $u_{1\alpha},\ldots, u_{m\alpha}$ **at** the successors

of node α. Thus, M_C starts the examinination from the leaves.

Description of M_C

A) M_C places its heads with status "busy" on all the leaves of the given tree t. Each head is in status "busy" or in status "ready".

B) When a head is on a leaf α, if $u_\alpha(n, z) = -$ for every n and z, then the head will be in status "ready", otherwise the tree t is rejected. (This examines the conditions 1 and 4. a0.)

C) For a node α ($\neq \lambda$) which has successors, if there is a head with status "ready" for every successor of node α, then these heads will be removed and a new head with status "busy" will be placed on the node α. M_C proceeds as follows:

C.1) For each (n, z) with $u_\alpha(n, z) \neq -$, compute $w_\alpha(n, z)$, where the function w_α is defined below by using an auxiliary function $\bar{w}_\alpha(n, z, B, \omega)$. If for every (n, z) with $u_\alpha(n, z) \neq -$, $u_\alpha(n, z) = w_\alpha(n, z)$ then go to step C.2, otherwise reject the tree t.

$\bar{w}_\alpha(n, z, B, \omega)$ is expressed by a nested conditional expression, where B and ω are used to detect the looping on a basis of Lemma 5, that is, B is used to detect the repeated occurences of a state and $\omega \in \{1,\ldots,E\}^*$ is used to detect the case of Lemma 5. b. The $\bar{w}_\alpha(n, z, B, \omega)$ described here can be equally used for M_{Df}. In the definition, $v_i(\alpha)$ denotes the i-th component of $v(\alpha)$, and B' denotes $B \cup \{(n, \omega)\}$. If $\omega = \omega_1 \ldots \omega_k$, $\omega_i \in \{1,\ldots,E\}$, then
$TOP(\omega) = \omega_k$, $POP(\omega) = \omega_1 \ldots \omega_{k-1}$ and $\omega_e = \omega_1 \ldots \omega_k e$.

$w_\alpha(n, z) = \bar{w}_\alpha(n, z, \phi, \lambda)$.

$\bar{w}_\alpha(n, z, B, \omega) =$

[$(n, \omega) \in B \rightarrow \infty$;
$|\omega| > \#_{PUSH} \rightarrow \infty$;
$(n, q_i, n_0, n_1) \in S \rightarrow \bar{w}_\alpha(n_{v_i(\alpha)}, z, B', \omega)$;
$(n, f_i, n') \in S \rightarrow$
[$\omega \neq \lambda$ or $z = 1 \rightarrow$
[Comment: In this bracket, s will denote $u_{i\alpha}(n', 1)$.
$s \in L_{POP}$ & $(s, POP, n_1,\ldots,n_E,n') \in S \rightarrow$
[$\omega \neq \lambda \rightarrow \bar{w}_\alpha(n_{TOP(\omega)}, z, B', POP(\omega))$;
$\omega = \lambda \rightarrow s$];
$s = *$ or $\infty \rightarrow s$;
$s = **$ $\rightarrow *$;
$s = -$ $\rightarrow$ undefined];
$\omega = \lambda$ & $z = 0 \rightarrow$
[Comment: In this bracket, s will denote $u_{i\alpha}(n', 0)$.
$s = *$ or $\infty \rightarrow s$;
$s = **$ $\rightarrow *$;
$s = -$ $\rightarrow$ undefined];
$(n, PUSH, e, n') \in S \rightarrow \bar{w}_\alpha(n', z, B', \omega \cdot e)$;
$(n, POP, n_1,\ldots,n_E,n') \in S \rightarrow$
[$\omega \neq \lambda \rightarrow \bar{w}_\alpha(n_{TOP(\omega)}, z, B', POP(\omega))$;
$\omega = \lambda$ & $z = 1 \rightarrow n$;
$\omega = \lambda$ & $z = 0 \rightarrow \bar{w}_\alpha(n', z, B', \lambda)$];
$(n, END) \in S \rightarrow **$]

This computation always halts by the use of B and ω. Here, $(n, \omega) \in B$ with $\omega = \omega_1 \ldots \omega_k$, $\omega_i \in \{1,\ldots, E\}$ means that for an integer j,
$\vdash^* (n,d,j+k,\langle d_1,e_1\rangle,\ldots,\langle d_j,e_j\rangle,\langle d,\omega_1\rangle,\ldots,\langle d,\omega_k\rangle)$
$\vdash^* (n,d,j+k,\langle d_1,e_1\rangle,\ldots,\langle d_j,e_j\rangle,\langle d,\omega_1\rangle,\ldots,\langle d,\omega_k\rangle)$,
where $h_0(\alpha) = d > d_j$, and that S enters a loop.
In the same way, $|\omega| > \#_{PUSH}$ means that for a state n and an integer j,
$\vdash^* (n, h_0(\alpha), j+|\omega|, \langle d_1,e_1\rangle,\ldots,\langle d_{j+|\omega|},e_{j+|\omega|}\rangle)$,
where $d_j < h_0(\alpha) = d_{j+1} = \ldots = d_{j+|\omega|}$ and that S enters a loop by Lemma 5. b.

C.2) In parallel with C.1, M_C examines whether there is an (n, z) such taht $u_{i\alpha}(n, z) \neq -$ but $u_{i\alpha}(n, z)$ is never referred in the computation of $w_\alpha(n', z')$ for any (n', z'). If there is such an (n, z), then the tree is rejected (Condition a0). Otherwise, the status of the head pointing to the node α will be changed to "ready", and Step C is repeated for other nodes to be examined if any. If all the node except the root are examined, then go to D.

D) At the root λ, if $u_\lambda(1, 0) = w_\lambda(1, 0) = *$ or $**$, then the tree t is accepted; otherwise t is rejected.

By the construction, it may be clear that M_C recognizes C.

Remarks The set D^f can be recognized in a similar way as C by replacing "$u_\lambda(1, 0) = *$ or $**$" by "$u_\lambda(1, 0) = \infty$" in the step D. The set D^i also can be recognized similarly with some modifications, where it is not necessary to examine the nodes β for which u_β is λ, but condition 6) must be examined. The set R can be easily recognized.

Theorem 3 The class of C_{BR} of BR-schemas is effectively translatable into the class C_{IP} of IP-schemas.

Sketch of proof. For a given BR-schema S, we can construct an equivalent IP-schema S'. To implement the recursive calls by using the pushdown memory of S', each occurrence of symbols $G_1,\ldots, G_k$ in the right hand side of the definitional equations of S' has a unique identification number between 1 and E. If a call with the identication number e to a Boolean subschema G_i is to be executed, S' will push the content of the main register X and the number e and go to the

subschema G_i. The subschema G_i has two exits: True and False. If a return from the G_i is to be executed through the exit True (or False), S' will execute a popup instruction (n, POP, n_1, n_2,..., n_E, n') whose n_e is the label of the instruction to be executed next when the answer of the Boolean subschema called by a call with the identication number e is True (or False). This is the essense of the construction.

Theorem 4' The divergence problem for R-schemas with a reset instruction RESET which stores a constant d into the main register is undecidable.

Proof. The problem is reduced to the Post correspondence problem. Let

$P = \{(\alpha_1, \beta_1),\ldots, (\alpha_n, \beta_n)\}$, $\alpha_i, \beta_i \in \{0, 1\}^* - \lambda$.

Given P, consider a schema S_p in Fig.3. F_β can be obtained from F_α by replacing "call F_α" with "call F_β" and by replacing "call G_{α_i}" with "call G_{β_i}". B is an (n + 1)-way switch. In Fig.3, G_{α_1} is shown as a representative and others are similar to G_{α_1}.

By construction, the values of the main register X at point A and at point C in the main schema are the same. Assume that the control reaches to the point B under an interpretation I. Then, the following relation (5) holds.

Let m be the least positive integer such that

$B(f^{m+1}d) = n + 1$

$B(f^{j}d) = i(j),\ j \leq m,\ i(j) < n$

and let $b_\ell \in \{0, 1\}$ denote the value $P^I(f^\ell f^{m+1}d)$. Let L be the least integer for which $q^I(f^L f^{m+1}d) = 1$. Then,

$$b_{L-1}b_{L-2}\cdots b_2 b_1 = \alpha_{i(1)}\cdots\alpha_{i(m)}. \quad (5)$$

If the computation starts from the point C with the initial value d and reaches to the point D, then under the interpretation I,

$$b_{L-1}b_{L-2}\cdots b_2 b_1 = \beta_{i(1)}\cdots\beta_{i(m)}. \quad (6)$$

From (5) and (6), S_p halts under an interpretation I if and only if there exists an index sequence i(1),..., i(m) such that

$$\alpha_{i(1)}\cdots\alpha_{i(m)} = \beta_{i(1)}\cdots\beta_{i(m)}.$$

Thus, if the divergence problem is decidable, then the Post correspondence problem is decidable. This is a contradiction.

Remark. Theorem 4 directly follows from Theorem 4' because the same arguments will do by replacing the first RESET by M ← X and the second RESET by X ← M in the main schema. By modifying the proof slightly, it can be shown that the convergence problem is undecidable.

Theorem 5 The class of IP-schemas with invertible functions (C_I^*) is effectively translatable into the class of IP-schemas (C_{IP}).

Proof. For simplicity, we consider an IP_0-schema S with invertible functions. (The general case can be proven by allowing additional control information.)

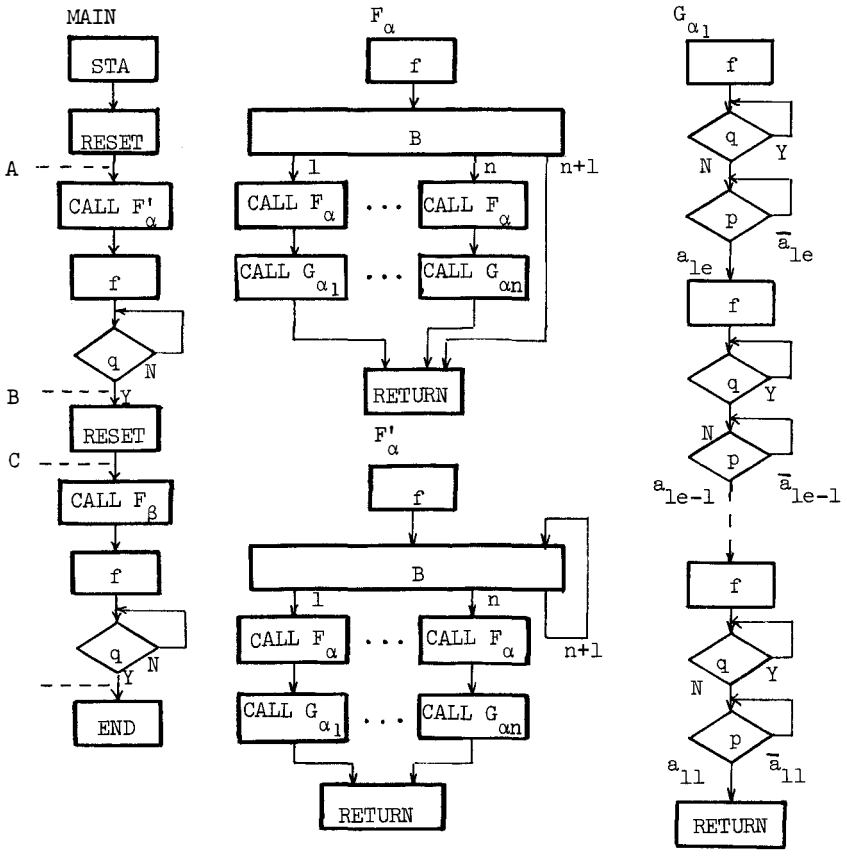

Fig.3 S_p,
where $\alpha_l = a_{l1}a_{l2}\cdots a_{le}$,
and $\bar{a} = 1-a$.

Let $\{f_1, \ldots, f_\ell, f_1^{-1}, \ldots, f_\ell^{-1}, f_{\ell+1}, \ldots, f_m\}$ be the set of function symbols of S, where f_i has no inverse for $\ell < i < m$. Let $F_I = \{f_1, \ldots, f_\ell, f_1^{-1}, \ldots, f_\ell^{-1}\}$. In the following, a construction of an IP-schema S' such that $S \equiv S'$ is shown. In the next definition of S', the label of instructions of S' are elements of $N \times F_I \cup \{\lambda, *\}$ and an element of $F_I \cup \{\lambda\}$ is stored in each E-field of the pushdown memory of S'. This is deviated from the original definition of IP-schemas, but this can be easily fitted by encoding with elements of N.

S	S'
(STA, 1)	(STA, <1, λ>)
(n, END)	(<n, e>, END) , $e \in F_I \cup \{\lambda\}$
(n, f_i, n'), $f_i \in F_I$	(<n, e>, PUSH, e, <n, *>), where $e \in (\{\lambda\} \cup F - \{f_i^{-1}\})$ (<n, *>, f_i, <n', f_i>) (<n, f_i^{-1}>, POP, <n', f_1>,..., <n', f_ℓ^{-1}>, <n', λ>, -), where "-" is a don't care.
(n, f_i, n'), $f_i \notin F_I$	(<n, e>, f_i, <n', λ>), where $e \in F_I \cup \{\lambda\}$.
(n, p_i, n_0, n_1)	(<n, e>, p_i, <n_0,e>, <n_1,e>), where $e \in F_I \cup \{\lambda\}$.

S' simulates S in the following way:

1) If S has a value w in X, then S' has the reduced form $\bar{w}$ in the main register X' of S' which is obtained from w by successive reductions

$$f_i f_i^{-1} = f_i^{-1} f_i = \text{identity}.$$

Indeed, the next example may clarify the intuitive notion used in the construction of S'.

Let $f_i \bar{t}$ be not reducible.

S	X	S'	X'	pushdown memory of S'
n	t	<n, e>	$\bar{t}$	ω
n'	$f_i t$	<n, *>	$\bar{t}$	$\omega \cdot <\bar{t}, e>$
		<n', f_i>	$f_i \bar{t}$	$\omega \cdot <\bar{t}, e>$
n"	$f_i^{-1} f_i t$	<n", e >	$\bar{t}$	ω

2) If S has a value ft with $f \notin F_I$, then the second component of the current state of S' is λ, and if S has a value ft with $f \in F_I$ and $f\bar{t}$ is not reducible, then the second component of the current state of S' is f. Each E-field of the pushdown memory of S' is used to save the second component of a state. In this way, S' simulates S.

References

1. E.Ashcroft, Z.Manna and A.Pnueli, "Decidable properties of monadic functional schemas," JACM, vol.20, pp.489-499, 1973.
2. A.K.Chandra, "On the decision problems of program schemas with commutative and invertible functions," Conf. Rec. of ACM Symp. on Principles of Prog. lang. pp.235-242, 1973.
3. V.Glushkov, "Automata theory and formal microprogram transformations," Kibernetika vol.1, pp.1-9, 1965.
4. I.Ianov, "The logical schemas of algorithms," English translation in Problems of Cybernetics, vol.1, pp.82-140, Pergamon Press, 1960.
5. K.Indermark, "On Ianov schemas with one memory location," in Lecture Notes in Computer Science 2, G.Goos and J.Hartmanis(eds.), pp.284-293, Springer-Verlag, 1973.
6. T.Ito, "A theory of formal microprograms," Summer School, St. Raphael 1971.
7. R.Kosaraju, "Finite state automata with markers," 4th Princeton Conf. on Inf. Sci. and Systems, p.380, 1970.
8. D.C.Luckham, D.M.R.Park and M.S.Paterson, "On formalized computer programs," JCSS, vol.4, pp.220-249, 1970.
9. Z.Manna, "Program Schemas," in Currents in the theory of computing, A.V.Aho(ed.), Prentice-Hall, 1973.
10. M.S.Paterson "Decision Problems in Computational models," SIGPLAN Notices, vol.7, pp.74-82, 1972.
11. J.W.Thatcher and J.B.Wright, "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic," Math. Syst. Theory vol.2, pp.57-81, 1968.
12. N.Tokura and T.Kasami, "Automata with labelled tree inputs," Trans. IECE, Japan, vol.57-D, pp.353-360, 1974.