Recursive Program Schemata
and Formal Languages

Takayasu Ito
(Mitsubishi Electric Corp.)

Introduction

Theory of program schemata was initiated by Yanov.
Yanov solved the equivalence problem of Yanov's program schemata,
which are equivalent to Graph schemata.
Igarashi and Rutledge reduced Yanov's result into the equivalence
problem of finite automata.
Luckham-Park—Paterson, Karp-Miller, Ershov-Itkin and others ex-
tended Yanov's result in various ways in connection with automata
theory.

In this talk we discuss an application of formal language theory to
program schemata.
Applications of formal language theory to program schemata were
taken during 1966-1968 while I was in Stanford University.

A part of the results was published in my IEEE symposium on Switching
and Automata Theory. In this talk I am going to introduce some of
my unpublished results while I was in Stanford University.
Recursive Program Schemata

Example: \text{fact}[n] : function defining n!

\[
\text{fact}[n] = \begin{cases} 
1 & \text{if } n = 0 \\
\text{else } n \times \text{fact}[n-1] 
\end{cases}
\]

\[
\phi_i(x_1, \ldots, x_n) = \mathcal{E}_i(\phi_1, \ldots, \phi_m, x_i, \ldots, x_n)
\]
where \(\mathcal{E}_i\) is "conditional expression" containing \(\phi_1, \ldots, \phi_m, x_i, \ldots, x_n\).

A class of functions defined recursively in this way is called "recursively-defined functions".

If an interpretation is not given for elementary functions in recursively-defined functions, they are called "recursive program schemata".

Example:

\[
\phi[x] = \begin{cases} 
\pi[x] & \text{if } a[x] \text{ else } h[x, \phi[S[x]]] 
\end{cases}
\]

\[
\text{fact}[x] = \begin{cases} 
1 & \text{if } x = 0 \\
\text{else } \text{multiply}[x, \text{fact}[x-1]] 
\end{cases}
\]

Definition of Equivalence of Recursive Program Schemata

\[
\phi_i^{(1)}(x_1, \ldots, x_n) = \mathcal{E}_i^{(1)}(\phi_1^{(1)}, \ldots, \phi_m^{(1)}, x_i, \ldots, x_n)
\]

\[
\phi_i^{(2)}(x_1, \ldots, x_n) = \mathcal{E}_i^{(2)}(\phi_1^{(2)}, \ldots, \phi_m^{(2)}, x_i, \ldots, x_n)
\]

\[
\phi_i^{(1)}(x_1, \ldots, x_n) \equiv \phi_i^{(2)}(x_1, \ldots, x_n) \iff \forall x_1 \forall x_2 [\phi_i^{(1)}(x_i, \ldots, x_n) = \phi_i^{(2)}(x_i, \ldots, x_n)]
\]

Theorem 1 (Luckham-Park-Paterson)

"The equivalence problem of recursive program schemata is recursively undecidable"

Proof of Theorem 1 can be made by showing that any LPP schema (Luckham-Park-Paterson schema) can be simulated by a recursive
program schema.

Simple Recursive Program Schemata

If a recursive program schema contains only the functions of one argument and one variable, then it is called a simple recursive program schema.

Example:

\[ \varphi[x] = \text{if } p[x] \text{ then } f[x] \text{ else } \varphi[f[x]] \]

Notation:

(i) \[ \text{if } p \text{ then } \alpha \text{ else } \beta \triangleq \begin{cases} \alpha \lor \beta & \text{if } p \\ \bot & \text{otherwise} \end{cases} \]

(ii) \[ \varphi[\varphi[f[x]]] \triangleq [f \varphi][x] \]

Example:

\[ \varphi[x] = \begin{cases} f \lor f \varphi \bot & \text{if } p \\ \alpha \lor \beta & \text{otherwise} \end{cases}[x] \]

\[ \varphi \triangleq \begin{cases} f \lor f \varphi \bot & \text{if } p \\ \alpha \lor \beta & \text{otherwise} \end{cases} \]

This sort of expressions will be called "simple recursive program schema".

Assume that we have a simple recursive program schema defined by

\[ \varphi_i \triangleq \alpha_i (\varphi_1, \ldots, \varphi_n) \]

and \( \phi \) is the always undefined function. Then the solution of (*) can be given by \( \lim_{k \to \infty} \varphi_i^{(k)} \), the limit of \( \varphi_i^{(k)} \triangleq \alpha_i (\varphi_1, \ldots, \varphi_k) \), \ldots,

**Definition 1**

\( \varphi_i^{(k+1)} \triangleq \alpha_i (\varphi_i^{(k)}, \ldots, \varphi_n^{(k)}) \)

(i) simple recursive program schema defined by

\[ \varphi_i \triangleq \begin{cases} a_{i_0} \lor a_i, x_i \lor \cdots \lor a_{i_{n-1}}, x_{n-1} \lor a_n x_n \bot \cdots \bot \\ p_i, \ldots, p_i \end{cases} \]

(3)
is called "right linear".

(ii) if the defining equation is

\[ x_i \equiv \bigvee_{P_i} a_i \bigvee_{P_i} a_i, x_i, b_i, \bigvee_{P_i \ll a_i} x_{i+1}, b_{i+1}, \bigvee_{P_i \ll a_i} x_n b_i \ll \ldots \]

then it is called "linear".

**Definition 2**

\[ E[ P ] : \text{ program event of } \]

Program event of \( P \) is the set of operator sequences of \( P \) under schematic interpretation.

In case of schematic interpretation, semantic functions are defined on operator sequences, assigning truth values to predicates and having the interpretation left open to operators.

**Example:**

\[ P \equiv L_e \bigvee_{P} f P f \bigvee_{P} g P g \]

\[ E( P ) : \text{ solution of } P = E U f P f U g P g \]

**Theorem 2**

(i) If a simple recursive program schema is right-linear, there exists a Yanov schema equivalent to it.

(ii) Equivalence problem of right linear simple recursive program schemata is decidable.

(iii) For any simple recursive program schemata

\[ P_1 \text{ and } P_2, \text{ if } P_1 \models P_2, \text{ then } E( P_1 ) = E( P_2 ) \]
Defininition

For a class of simple recursive program schema \( \mathcal{F}_d \), we define the class of program events by \( \mathcal{E}(\mathcal{F}_d) = \{ \mathcal{E}(P) : P \in \mathcal{F}_d \} \).

Theorem 3

For the class of simple recursive program schemata \( \mathcal{F}_d \)

(i) \( \mathcal{F}_d \) is right linear \( \Rightarrow \) \( \mathcal{E}(\mathcal{F}_d) \) is regular.

(ii) \( \mathcal{F}_d \) is linear \( \Rightarrow \) \( \mathcal{E}(\mathcal{F}_d) \) is linear CFL.

(iii) \( \mathcal{F}_d \) is the class of simple recursive program schemata \( \mathcal{E}(\mathcal{F}_d) \) is CFL.

\( \text{Val}_\Pi\{P\} \): value of \( P \) under a schematic interpretation \( \Pi \)

\( \text{Eval}_\Pi\{P\} = \Xi_1 \hat{\omega}_1 , \cdots , \Xi_n \hat{\omega}_n \)

for \( \text{Val}_\Pi\{P\} = \Xi_1 , \cdots , \Xi_n \)

where

\( \hat{\omega}_k : \) a sequence consisting of \( 0^i \) and \( 1^j \)

\( \hat{\omega}_k = \Pi(\hat{\omega}_k) (\Xi_1 , \cdots , \Xi_1) \)

\( \hat{\omega}_k = \hat{\omega}_{k1} , \cdots , \hat{\omega}_{kn} \)

Defininition 3

Execution event: \( \hat{\mathcal{E}}(P) \) = set of \( \text{Eval}_\Pi\{P\} \)

Theorem 4

(i) For any simple recursive program schemata \( P_1 \) and \( P_2 \)

\( P_1 \preceq P_2 \iff \hat{\mathcal{E}}(P_1) = \hat{\mathcal{E}}(P_2) \)

(ii) For any simple recursive program schema \( P \),

\( \hat{\mathcal{E}}(P) \) defines a deterministic CFL.

(iii) Termination problem for simple recursive program
schemata is decidable.

Outline of Proof

(i) @ \( P_1 \cong P_2 \) and \( \Pi \) is a schematic interpretation

\[ \Rightarrow \text{Val}_I ( P_1 ) = \text{Val}_I ( P_2 ) \]

and \( \text{Execval}_I ( P_1 ) = \text{Execval}_I ( P_2 ) \)

\[ \Rightarrow \text{\~E} ( P_1 ) = \text{\~E} ( P_2 ) \]

\( \text{\~E}(P)=\text{\~E}(P) \) and \( \Pi \) is a schematic interpretation:

\[ \Rightarrow \text{Execval}_I [ P_1 ] \in \text{\~E} ( P_2 ) \]

\[ \Rightarrow \text{Execval}_I [ P_1 ] = \text{Execval}_I [ P_2 ] \]

(ii) Show that \( \text{\~E} ( P_1 ) \) is accepted by deterministic pushdown automata.

(iii) Follow from (ii)

The results in this talk and my IEEE paper tell us that some decision problems on program schemata are reducible to decision problems on formal languages.