Theoretical Computer Science 19 (1982) 105–115
North-Holland Publishing Company

NOTE

ON UNIVERSALITY OF CONCURRENT EXPRESSIONS WITH SYNCHRONIZATION PRIMITIVES

T. ITO and Y. NISHITANI

Department of Electrical Communications, School of Engineering, Tohoku University, Sendai, Japan

Communicated by M. Nivat
Received September 1981

Abstract. Concurrent expressions are a class of extended regular expressions with a shuffle operator (||) and its closure ($^{\Phi}$). The class of concurrent expressions with synchronization primitives, called synchronized concurrent expressions, is introduced as an extended model of Shaw's flow expressions. This paper discusses some formal properties of synchronized concurrent expressions from a formal language theoretic point of view. It is shown that synchronized concurrent expressions with three signal/wait operations are universal in the sense that they can simulate any semaphore controlled concurrent expressions and they can describe the class of recursively enumerable sets. Some results on semaphore controlled *regular* expressions are also included to give a taste of more positive results.

1. Introduction

A number of formal approaches to investigate properties of concurrent process are proposed in the literature. Shaw [7] proposed a model called "flow expression". Flow expressions are a class of extended regular expressions with a shuffle operation, its closure, an infinite cycle operation and synchronization primitives. Shaw [7] discusses the correspondence between flow expressions and concurrent processes and gives some elementary properties of flow expressions. Also, Shaw [8] discusses some relations between flow expressions and other models of concurrent processes. Some formal properties of flow expressions have been studied by Shaw, Araki–Tokura and others from a formal language theoretic point of view, as is mentioned in Shaw [8].

In this paper we introduce a slightly generalized formalism of flow expressions, called "synchronized concurrent expressions", which is more suitable for our theoretical study on concurrent processes. Concurrent expressions are a class of extended regular expressions with a shuffle operator (||) and its closure ($^{\Phi}$). The class of concurrent expressions with synchronization primitives (like semaphore primitives) is called *synchronized concurrent expressions* in this paper. We introduce

0304-3975/82/0000–0000/$02.75

a reasonable class of synchronization primitives defined as concurrent expressions, capable of describing various semaphore systems. Using this formalism we show that synchronized concurrent expressions with three signal/wait semaphore operations are universal in the sense that they can simulate any semaphore controlled concurrent expressions and they can describe the class of recursively enumerable sets. Some latest results on semaphore controlled *regular* expressions are also included to explain more positive theoretical aspects of these studies.

2. Synchronized concurrent expressions

In this section we give the definitions of
(1) concurrent expression,
(2) synchronized concurrent expression,
(3) semaphore systems and synchronization primitives in concurrent expressions and we explain some previous results.

2.1. Definition of concurrent expressions

Let Δ be a finite set of symbols. Then the set of concurrent expressions over Δ can be defined recursively as follows:
(1) ϕ, λ, $a(\varepsilon\Delta)$ are concurrent expressions,
(2) if E and F are concurrent expressions, then (E), $E+F$, EF, E^*, $E\|F$ and E^{Φ} are concurrent expressions.

The semantics of concurrent expressions can be defined as follows:
(1) $|\emptyset| = \emptyset$ ($\emptyset$: the empty set),
(2) $|\lambda| = \{\lambda\}$ (λ: the empty word),
(3) $|a| = \{a\}$ $(a \in \Delta)$,
(4) $|(E)| = |E|$,
(5) $|E+F| = |E| \cup |F|$,
(6) $|EF| = \{xy \mid x \in |E|, y \in |F|\}$,
(7) $|E^*| = \bigcup_{i=0}^{\infty} |E^i|$, where $E^0 = \lambda$, $E^{i+1} = E^i E$ for $i \geq 0$,
(8) $|E\|F| = \{x_1 y_1 x_2 y_2 \ldots x_n y_m \mid x_i \in \Delta^*, y_i \in \Delta^*, x_1 x_2 \ldots x_m \in |E|, y_1 y_2 \ldots y_m \in |F|\}$,
(9) $|E^{\Phi}| = \bigcup_{i=0}^{\infty} |E^{\|i}|$, where $E^{\|0} = \lambda$, $E^{\|i+1} = E^{\|i} \| E$ for $i \geq 0$.

Remark 1. In this paper concurrent expressions are defined as extended regular expressions with the shuffle operator ($\|$) and its closure ($^{\Phi}$). Shaw's definition of flow expressions contains an infinite cycle operation (ω), but the infinity operator (ω) was excluded int this paper since its descriptive power is beyond the standard theory of formal languages. The analysis of flow expressions in this paper is essentially based on the theory of formal languages. In this paper
- $\mathbf{R}(\Delta)$ denotes the set of regular expressions over Δ;
- $\mathbf{C}(\Delta)$ denotes the set of concurrent expressions over Δ.

Remark 2. Let $E = ab$ and $F = b$ as an example. Then we have

$$|E\|F| = \{abc, acb, cab\}, \qquad |E^{\Phi}| = \{\lambda, ab, abab, aabb, \ldots\}$$

The shuffle operation "$E\|F$" corresponds to the set of computation sequences by a concurrent process:

cobegin $E\|F$ **coend**

2.2. Synchronized concurrent expressions

Let Σ be a finite set of symbols and Γ be a finite set of synchronization primitives, where $\Sigma \cap \Gamma = \emptyset$. Then a concurrent expression over Σ and Γ is called a *synchronized concurrent expression*, and a language over Γ is called a *synchronization mechanism*. Given a synchronization mechanism K and a concurrent expression E, the language described by E with K-control is given as

$$L_K(E) = \{h(x) \mid x \in |E|, \bar{h}(x) \in K\}$$

where h and $\bar{h}$ are the following homomorphisms:

$$h(a) = \begin{cases} a & (a \in \Sigma), \\ \lambda & (a \in \Gamma); \end{cases} \qquad \bar{h}(a) = \begin{cases} \lambda & (a \in \Sigma), \\ a & (a \in \Gamma). \end{cases}$$

Let K be a synchronization mechanism and $\mathbf{K}$ be a family of languages in terms of synchronization mechanisms as follows:

$$L_K = \{L_K(E) \mid E \in \mathbf{C}(\Sigma \cup \Gamma), \Sigma \cap \Gamma = \emptyset\}, \qquad L_{\mathbf{K}} = \bigcup_{K \in \mathbf{K}} L_K$$

where $\mathbf{C}(\Sigma \cup \Gamma)$ is the set of concurrent expressions over $\Sigma \cup \Gamma$.

Remark 3. In the above definition Σ and Γ correspond to the set of basic processes and synchronization primitives, respectively. It may be interpreted that an element of x of $|E|$ is a computational sequence of a concurrent process denoted by E. The homomorphisms $h(x)$ and $\bar{h}(x)$ correspond to the sequences of basic processes and synchronization primitives in x, respectively. This formalism of characterizing process descriptions and synchronization mechanisms of synchronized concurrent expressions by homomorphisms is a slightly generalized and formalized version of flow expressions introduced by Shaw [7], in which only a restricted class of synchronization mechanisms are allowed. Our present formalism enables us to deal with synchronization mechanisms independently of process descriptions. In this paper we study the descriptive power of L_K and $L_{\mathbf{K}}$ with respect to K and $\mathbf{K}$ using this formalism.

2.3. Semaphore systems and synchronization primitives in concurrent expressions

Let $\Gamma_n = \{\sigma_i, \omega_i \mid i = 0, 1, \ldots, n-1\}$, where σ_i and ω_i correspond to the wait and signal operations on the ith semaphore variable, respectively. Using concurrent

expressions we define four semaphore systems over Γ_n in the following way:

(1) Counter semaphore: $C(n)$

$$C(n) = |(\sigma_0\omega_0 + \sigma_0)^{\Phi} \| (\sigma_1\omega_1 + \sigma_1)^{\Phi} \| \cdots \| (\sigma_{n-1}\omega_{n-1} + \sigma_{n-1})^{\Phi}|,$$

$$\mathbf{C} = \{C(n) \mid n \geq 0\}, \text{ where } C(0) = \{\lambda\};$$

(2) [0]-counter semaphore: $C_0(n)$

$$C_0(n) = |(\sigma_0\omega_0)^{\Phi} \| (\sigma_1\omega_1)^{\Phi} \| \cdots \| (\sigma_{n-1}\omega_{n-1})^{\Phi}|,$$

$$\mathbf{C}_0 = \{C_0(n) \mid n \geq 0\}, \text{ where } C_0(0) = \{\lambda\};$$

(3) Binary semaphore: $B(n)$

$$B(n) = |(\sigma_0\omega_0 + \sigma_0)^* \| (\sigma_1\omega_1 + \sigma_1)^* \| \cdots \| (\sigma_{n-1}\omega_{n-1} + \sigma_{n-1})^*|,$$

$$\mathbf{B} = \{B(n) \mid n \geq 0\}, \text{ where } B(0) = \{\lambda\};$$

(4) [0]-binary semaphore: $B_0(n)$

$$B_0(n) = |(\sigma_0\omega_0)^* \| (\sigma_1\omega_1)^* \| \cdots \| (\sigma_{n-1}\omega_{n-1})^*|,$$

$$\mathbf{B}_0(n) = \{B_0(n) \mid n \geq 0\}, \text{ where } B_0(0) = \{\lambda\}.$$

Remark 4. From the standpoint of operating systems practice we have the following correspondence.

(1) ω_i and σ_i of $C(n)$ and $C_0(n)$ correspond to the following wait(s_i) and signal(s_i), respectively:

wait(s_i): **when** $s > 0$ **do** $s \leftarrow s - 1$,

signal(s_i): **when** true **do** $s \leftarrow s + 1$;

(2) ω_i and σ_i of $B(n)$ correspond to "**when** $s_i = 1$ **do** $s_i \leftarrow 0$" and "**when** true **do** $s_i \leftarrow 1$", respectively;

(3) ω_i and σ_i of $B_0(n)$ correspond to "**when** $s_i = 1$ **do** $s_i \leftarrow 0$" and "**when** $s_i = 0$ **do** $s_i \leftarrow 1$", respectively.

In the above definition of the semaphore systems we can observe the interesting fact that the synchronization mechanisms based on counter semaphores and binary semaphores can be defined as a kind of well-structured concurrent expressions in the sense that they are expressed without any complicated nesting structure and they are readable from left to right. These well-structured descriptions of synchronization mechanisms may be interesting and worthwhile for future studies.

Extending the above definitions of semaphore systems we can define the following general synchronization mechanism:

(5) Let $\mathbf{CE}(n)$ be a family of synchronization mechanisms defined by concurrent expression with n synchronization primitives. Then we denote $\mathbf{CE} = \bigcup_{n=0}^{\infty} \mathbf{CE}(n)$.

Notice that in this definition of **CE** n synchronization primitives are not restricted to the semaphores. $C(n)$, $C_0(n)$, $B(n)$ and $B_0(n)$ may be considered to be special cases of $\mathbf{CE}(2n)$.

(6) A concurrent expression controlled by a semaphore system will be called as a *semaphore controlled concurrent expression.* In the discussions below we are interested in L_K and $L_{\mathbf{K}}$ when K and $\mathbf{K}$ belong to the semaphore systems of **CE**.

2.4. Some previous results

Under our present formalism we can state the results obtained in the study of flow expressions as follows:

T1: For a concurrent expression E, $|E|$ is not necessarily context-free but context-sensitive; if E does not contain the shuffle closure operator ($^{\Phi}$), then $|E|$ is regular.

T2: If a concurrent expression E does not contain any shuffle closure ($\|$), then $L_{B(n)}(E)$ is regular [7].

T3: $L_{\mathbf{B}} = \mathbf{RE}$, where **RE** is a family of recursively enumarable sets. [1]

T4: $L_{\mathbf{CE}} = L_{C(6)} = L_{C_0(6)} = L_{B(6)} = L_{B_0(6)}$. [6]

According to T1 we can say that any synchronization mechanism which belongs to **CE** is a context-sensitive control structure; $B(n)$ and $B_0(n)$ are regular control structures for $n \geq 1$; $C(n)$ and $C_0(n)$ are not regular control structures for $n \geq 1$. In this paper we improve and extend T4 using a different simulation technique, which will be explained in the following sections.

2.5. Examples

Before moving onto the detailed discussions we give several examples to give an intuitive appeal of synchronized concurrent expressions.

(1) Correspondence between concurrent expressions and programs

[*program*]	[*concurrent expression*]
if B **then** E **else** F	$E+F$
begin E; F **end**	EF
while B **do** E	E^*
cobegin $E\|F$ **coend**	$E\|F$
for $i = 1$ **to** n **do** createprocess $[E, i]$	E^{Φ}
wait(s)	ω_s
signal(s)	σ_s

(2) $L_{B_0(1)}[(\omega ab\sigma)\|(\omega cd\sigma)] = \{abcd, cdab\}$,

$L_B[(\sigma S\omega)^{\Phi}] = |S^*|$ assuming that σ and ω does not appear in S,

$L_B[a\omega b\sigma] = \emptyset$ (in this case the process is deadlocked).

3. Universality of synchronized concurrent expressions

Intuitively a semaphore system may be universal in the sense that any synchronization mechanism in operating systems practice can be realized by a semaphore system. Using the above formalism we give a formal proof that four kinds of semaphore systems given in Section 2

can simulate each other,

can simulate any synchronization mechanism within *CE* and

can describe the family of recursively enumberable sets;

eventually we reach the universality of synchronized concurrent expressions.

That is, we can prove the following theorem:

Theorem 1. $L_{CE} = L_{C} = L_{C_0} = L_{B} = L_{B_0} = \mathbf{RE}$.

In order to prove Theorem 1 we need some preliminary definitions and results

Definition. c_i $(0 < i \leq n-1)$ is the following mapping from $(\Sigma \cup \Gamma_n)^*$ into the set of integers: for $x \in (\Sigma \cup \Gamma_n)^*$ and a $(\Sigma \cup \Gamma_n)$,

$$c_i(\lambda) = 0, \qquad c_i(xa) = \begin{cases} c_i(x)+1 & \text{for } a = \sigma_i, \\ c_i(x)-1 & \text{for } a = \omega_i, \\ c_i(x) & \text{for } a \notin \{\sigma_i, \omega_i\}. \end{cases}$$

Proposition. *For* $x \in (\Sigma \cup \Gamma_n)$ $\bar{h}(x) \in C(n)$ *if and only if for any* $x' \in \mathrm{prefix}(x)$ *and each* i $(0 \leq i \leq n-1)$, $c_i(x') \geq 0$.

Definition. Let Δ and Δ' be the finite sets of symbols and f be a mapping from Δ into $\mathbf{C}(\Delta')$. For a concurrent expression E on Δ $f(E)$ denotes a concurrent expression on Δ' which is obtained from E by replacing every occurrence of a $(\in\Delta)$ by $f(a)$. When $|f(x)|$ denotes a single word for $x(\in\Delta^*)$, $f(x)$ may be treated as a word on Δ', where we assume $f(\lambda) = \lambda$.

Proposition. *Let* E *be a concurrent expression on* Δ *and let* $f: \Delta \to \mathbf{C}(\Delta' \cup \{[\,,\,]\})$ *be the following mapping:*

$$f(a) = [E_a]$$

where [*and*] *are the symbols which do not belong to* Δ' *and* $E_a \in \mathbf{C}(\Delta')$.

If the sequence of [*and*] *in* $y \in |f(E)|$ *is of the form of* $([\,])^*$, *then there exists* $x \in |E|$ *such that* $y \in |f(x)|$, *and* x *and* y *are given as*

$$x = a_1 a_2 \ldots a_m \qquad (a_j \in \Delta),$$

$$y = [y_1][y_2]\ldots[y_m] \qquad ([y_j] \in |f(a_j)|).$$

Lemma 1.

$$L_{\mathbf{CE}} \subseteq L_{C(n+3)}, \qquad L_{\mathbf{CE}(n)} \subseteq L_{C_0(n+3)},$$

$$L_{\mathbf{CE}(n)} \subseteq L_{B(n+3)}, \qquad L_{\mathbf{CE}(n)} \subseteq L_{B_0(n+3)}.$$

This lemma constitutes the major part of Theorem 1. In this paper we give the outline of the proof of this lemma, since the complete proof is rather lengthy.

Outline of the proof of Lemma 1. Let $\Gamma = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\}$ be a set of synchronization primitives. In order to establish Lemma 1 we construct a concurrent expression E' on $\Sigma \cup \Gamma_{n+3}$ for a concurrent expression E_K on Γ and a concurrent expression E on $\Sigma \cup \Gamma$. For this concurrent expression E' we show

$$L_{|E_K|}(E) = L_{C(n+3)}(E') = L_{C_0(n+3)}(E') = L_{B(n+3)}(E') = L_{B_0(n+3)}(E').$$

Let], [,] and [be σ_{n+1}, ω_{n+1}, σ_{n+2}, ω_{n+2}, respectively, for $\Gamma_{n+3} = \{\sigma_0, \omega_0, \sigma_1, \omega_1, \ldots, \sigma_{n+1}, \omega_{n+1}, \sigma_{n+2}, \omega_{n+2}\}$. We define the following mappings:

$$f_1 : \Sigma \cup \Gamma \to \mathbf{C}(\Sigma \cup \Gamma_{n+3}),$$

$$f_1(a) = [a] \qquad (a \in \Gamma),$$

$$f_1(\gamma_i) = [\omega_i \sigma_n] \qquad (\gamma_i \in \Gamma),$$

$$f_2 : \Gamma \to \mathbf{C}(\Gamma_{n+3}),$$

$$f_2(\gamma_i) =]\sigma_i \omega_n[\qquad (\gamma_i \in \Gamma).$$

Then the required concurrent expression E' can be given as follows:

$$E' =]](f_1(E) \| f_2(E_K))[[.$$

Lemma 1 can be proved by showing that

$$L_{|E_K|}(E) \subseteq L_{B_0(n+3)} \quad \text{and} \quad L_{C(n+3)}(E') \subseteq L_{|E_K|}(E)$$

since we have

$$L_{B_0(n)}(E'') \subseteq L_{B(n)}(E'') \subseteq L_{C(n)}(E'') \quad \text{and} \quad L_{B_0(n)}(E'') \subseteq L_{C_0(n)}(E'') \subseteq L_{C(n)}(E'')$$

for any concurrent expression E''.

Proof of Theorem 1. Using Lemma 1 we can easily prove Theorem 1. From Lemma 1 we have $L_{\mathbf{CE}} \subseteq L_{\mathbf{C}}$. From the definition of $C(n)$, $\mathbf{CE}(n)$ and $\mathbf{CE}$, $C(n) \in \mathbf{CE}(2n) \subseteq \mathbf{CE}$; hence we have $L_{\mathbf{C}} \subseteq L_{\mathbf{CE}}$. Thus we get $L_{\mathbf{CE}} = L_{\mathbf{C}}$; we can establish the similar results for $\mathbf{C}_0$, $\mathbf{B}$ and $\mathbf{B}_0$.

Remark 5. The recursive enumerability of Theorem 1 follows from T3, but the proof of T3 by Araki–Tokura is complicated since they use the original Shaw flow expression and the binary semaphore system for simulation. Using the formalism

of this paper we can give a neat proof of $L_{CE} = \mathbf{RE}$ in a straightforward manner, simulating the behaviour of 2-counter automaton by a synchronized concurrent expression. Intuitively speaking this can be done by simulating a counter behaviour as a synchronization mechanism

$$|((\gamma\bar{\gamma})^{\Phi}\gamma_0)^*|$$

where γ, $\bar{\gamma}$ and γ_0 correspond to the increment (+1), the decrement (−1) and the zero test of the counter, respectively. According to this proof we can show the following stronger result:

Corollary. $L_{CE}^{reg} = \mathbf{RE}$ *where* $L_{CE}^{reg} = \{L_{CE}(E) \mid E \in \mathbf{R}(\Sigma \cup \Gamma), \Sigma \cap \Gamma = \emptyset\}$ *and* $\mathbf{R}(\Sigma \cup \Gamma)$ *is the set of regular expressions over* $\Sigma \cup \Gamma$.

That is, the regular expressions controlled by **CE** can describe recursively enumerable sets; hence we can say that synchronized regular expressions are universal.

4. Universality of synchronized concurrent expressions with three synchronization primitives

The results on universality of synchronized concurrent expressions can be strengthened as follows:

Theorem 2. $L_{CE} = L_{C(3)} = L_{C_0(3)} = L_{B(3)} = L_{B_0(3)} = \mathbf{RE}$.

The proof of this theorem can be shown by showing the following lemmas, and Lemma 3 constitute the major part of Theorem 2.

Lemma 2. $L_{C(n)} \subseteq L_{C_0(n)}$, $L_{C(n)} \subseteq L_{B(n)}$, $L_{C(n)} \subseteq L_{B_0(n)}$ *for* $n = 0, 1, 2, \ldots$.

Lemma 3. $L_{B_0(n)} \subseteq L_{C(3)}$ *for* $n = 0, 1, 2, \ldots$.

Outline of the proof of Lemma 2. The proof of Lemma 2 can be given by showing that for any concurrent expression E on $\Sigma \cup \Gamma_n$ we can construct a concurrent expression E' such that

$$L_{C(n)}(E) = L_{C_0(n)}(E') = L_{B(n)}(E') = L_{B_0(n)}(E').$$

Let f_1 be the following mapping:

$$f_1 : \Gamma_n \to \Gamma_n; \quad f_1(\sigma_i) = \omega_i \text{ for } \sigma_i \in \Gamma_n; \quad f_1(\omega_i) = \sigma_i \text{ for } \omega_i \in \Gamma_n.$$

Then the required concurrent expression E' can be given as follows:

$$E' = E \| f_1(E_C)$$

where E_C is the concurrent expression corresponding to $C(n)$, that is,

$$E_C = (\sigma_0\omega_0+\sigma_0)^{\Phi}\| \cdots \|(\sigma_{n-1}\omega_{n-1}+\sigma_{n-1})^{\Phi}.$$

For this E' we can show

$$L_{C(n)}(E)\subseteq L_{B_0(n)}(E') \quad \text{and} \quad L_{C(n)}(E')\subseteq L_{C(n)}(E).$$

From these results we can prove Lemma 2, since for any expression E'' we have

$$L_{B_0(n)}(E'')\subseteq L_{B(n)}(E'')\subseteq L_{C(n)}(E''),$$
$$L_{B_0(n)}(E'')\subseteq L_{C_0(n)}(E'')\subseteq L_{C(n)}(E'').$$

Outline of the proof of Lemma 3. The proof of Lemma 3 can be given by showing that for any concurrent expression E on $\Sigma\cup\Gamma_n$ we can construct a concurrent expression E' on $\Sigma\cup\Gamma_3$ such that

$$L_{B_0(n)}(E)=L_{C(3)}(E').$$

The required concurrent expression E' can be given as

$$E'=S(0)]f(E)[W(0)$$

where

(1)] and [correspond to σ_0 and ω_0, respectively;

(2) For integers t (≥ 0) and i (≥ 0), let $\langle t\rangle_i$ be the value of the ith position of the binary representation of t, and let $N=2^n-1$. For an integer t ($0\leq t\leq N$), $W(t)$ and $S(t)$ are defined as the following words on Γ_3:

$$W(t)=\omega_1^t\omega_2^{N-t}, \qquad S(t)=\sigma_1^t\sigma_2^{N-t};$$

(3) $f:\Sigma\cup\Gamma_n\to \mathbf{C}(\Sigma\cup\Gamma_3)$,

$$f(a)=[a] \quad (a\in\Sigma),$$
$$f(\sigma_i)=([W(\bar{k}_1)S(k_1)]+[W(\bar{k}_2)S(k_2)]$$
$$+\cdots+[W(\bar{k}_{2^n-1})S(k_{2^n-1})]) \mid (\sigma_i\in\Gamma_n),$$
$$f(\omega_i)=([W(k_1)S(\bar{k}_1)]+[W(k_2)S(\bar{k}_2)]$$
$$+\cdots+[W(k_{2^n-1})S(\bar{k}_{2^n-1})]) \quad (\omega_i\in\Gamma_n)$$

assuming that k_j ($1\leq j\leq 2^{n-1}$) is an integer such that $\langle k_j\rangle_i=1$ and $0\leq k_j\leq N$ ($=2^n-1$), and $\bar{k}_j=k_j-2^i$. In order to show that $L_{B_0(n)}(E)=L_{C(3)}(E')$ we need to use the following propositions.

Proposition. *For $x\in(\Gamma\cup\Sigma_n)^*$, $\bar{h}(x)\in B(n)$ if and only if $0\leq c_i(x')\leq 1$ for any $x'\in$* prefix(x) *and every i ($0\leq i\leq n-1$) and $c_i(x)=0$ for every i ($0\leq i\leq n-1$).*

Proposition. *Let*

$$y = S(t_0)]f(u_0)[W(t')S(t_1)]f(u_1)[W(t_2')S(t_2)]f(u_2)\ldots$$
$$[W(t_m')S(t_m)]f(u_m)[W(t_M')$$

where $j \in \Sigma^*$ $(0 \leq j \leq m)$ *and* $M = m+1$. *Then*

$$\bar{h}(y) \in C(3) \text{ if and only if } t_j = t_{j+1}' \text{ for every } j \ (0 \leq j \leq m).$$

Using the above results we can prove our main theorem as follows:

Proof of Theorem 2. By Theorem 1, $L_{\mathbf{CE}} = L_{\mathbf{B}_0}$; by Lemma 3, $L_{\mathbf{B}_0} \subseteq L_{C(3)}$; by Lemma 2, $L_{C(3)} \subseteq L_{C_0(3)}$. $L_{C_0(3)} \subseteq L_{\mathbf{CE}}$ follows from the definitions. Thus we have proved

$$L_{\mathbf{CE}} = L_{C(3)} = L_{C_0(3)}.$$

Similarly we can prove

$$L_{\mathbf{CE}} = L_{B(3)} = L_{B_0(3)}.$$

This completes the proof of Theorem 2.

5. Concluding remarks

We can summarize our results in this paper as follows:

(1) $L_{\mathbf{CE}} = L_{\mathbf{C}} = L_{\mathbf{C}_0} = L_{\mathbf{B}} = L_{\mathbf{B}_0} = \mathbf{RE}$;

(2) $L_{\mathbf{CE}} = L_{\mathbf{Rg}} = L_{C(3)} = L_{C_0(3)} = L_{B(3)} = L_{B_0(3)} = \mathbf{RE}$

where **Rg** is the class of regular expressions defined over Γ_n. (Notice that B and B_0 belong to **Rg**.)

It remains open if the class of synchronized concurrent expressions with only two synchronization primitives are universal.

The universality of synchronized concurrent expressions implies a rather negative result for analysis of concurrent processes. This leads us to study a more restricted class of synchronized concurrent expressions. We have already mentioned such a class, called a class of synchronized regular expressions.

A regular expression over Σ and Γ is called a synchronized regular expression. Given a synchronization mechanism K we define a family of languages as follows:

$$L_K^{\mathrm{reg}} = \{L_K(E) \mid E \in \mathbf{R}(\Sigma \cup \Gamma), \Sigma \cap \Gamma = \emptyset\}, \qquad L_{\mathbf{K}}^{\mathrm{reg}} = \bigcup_{K \in \mathbf{K}} L_K^{\mathrm{reg}}.$$

For the class of synchronized regular expressions we have the following results:

(i) $L_{\mathbf{CE}}^{\mathrm{reg}} = \mathbf{RE}$;

(ii) $L_{\mathbf{B}}^{\mathrm{reg}} = L_{\mathbf{Rg}}^{\mathrm{reg}} = \mathbf{REG}$ (**REG**: the family of regular sets);

(iii) $\mathbf{REG} = L_{\mathbf{Rg}}^{\mathrm{reg}} = L_{\mathbf{B}_0}^{\mathrm{reg}} = L_{\mathbf{B}}^{\mathrm{reg}} \subsetneq L_{\mathbf{C}}^{\mathrm{reg}} \subsetneq L_{\mathbf{C}_0}^{\mathrm{reg}} = \mathbf{PN}$ where **PN** is the family of Petri net languages;

(iv) $L_{C(n)}^{\mathrm{reg}} \subsetneq L_{C(n+1)}^{\mathrm{reg}} \quad (n \geq 0)$;

(v) $L_{\mathbf{DYCK}}^{\mathrm{reg}} = L_{\mathrm{DYCK}(2)}^{\mathrm{reg}} = \mathbf{CFL}$ where DYCK(n) is the Dyck synchronization mechanism over Γ_n defined as follows:

Let

$$G_D = (V_T, V_N, P, S), \qquad V_T = \Gamma_n = \{\sigma_1, \omega_1, \sigma_2, \omega_2, \ldots, \sigma_n, \omega_n\},$$

$$V_N = S, \qquad P = \{S \to S\sigma_i S\omega_i S \mid 1 \leq i \leq n\} \cup \{S \to \lambda\}.$$

The Dyck synchronization mechanism over Γ_n – DYCK(n) – can be defined as the language generated by the above grammar G_D. We also define as

$$\mathbf{DYCK} = \{\mathrm{DYCK}(n) \mid n \geq 0\}.$$

This result establishes a characterization theorem of context-free languages by a class of synchronized regular expressions.

The details of these results on synchronized regular expressions will be given elsewhere.

References

[1] T. Araki and N. Tokura, Unsolvability of the equivalence problems of flow expressions, *Trans. IECE Japan* **J62-D** (5) (1979) (in Japanese).

[2] T. Araki and N. Tokura, On the descriptive power of flow expressions and event expressions, *Trans. IECE Japan* **J63-D** (8) (1980) (in Japanese).

[3] P.C. Fisher, Turing machines with restricted memory access, *Information and Control* **9** (4) (1966).

[4] A.N. Habermann *Introduction to Operating System Design* (Science Research Associates, 1976).

[5] J.E. Hopcroft and J.D. Ullman, *Formal Languages and their Relation to Automata Theory* (Addison-Wesley, Reading, MA, 1969).

[6] Y. Nishitani and T. Ito, On the descriptive power of semaphore controlled flow expressions, *Convention Record of IECE of Japan* (August 1980) (in Japanese).

[7] A.C. Shaw, Software descriptions with flow expressions, *IEEE Trans. Software Engrg.* **4** (3) (1978).

[8] A.C. Shaw, Software specification languages based on regular expression, in: *Software Development Tools* (Springer, Berlin, 1980).