"SHIFT-LEFT" ALGORITHMS TRANSFORMING SEQUENTIAL PROCESSES INTO CONCURRENT ONES

Hoang Chi Thanh

Hanoi University of Science, VNUH

Abstract. Finding concurrent processes of a system is an objective of system controls, because it shows an optimal way to perform processes. In this paper we build two iterative algorithms for transforming sequential processes of a reliance alphabet and of a Place/Transition net into concurrent ones. The complexity of these algorithms is also considered.

Keywords: Reliance alphabet, trace, Petri net, concurrent step.

1. INTRODUCTION

Controls on concurrent systems always are an important and complicated problem. So far, optimal performance of processes occurred on a concurrent system is among the controls. For this purpose, recognition of concurrent processes of the system is essential. A lot of models, such as Petri nets [6,9], traces [1,3,4,8], CSP [7], CCS [5], process algebras [2] ..., have been being good tools for representing statistical as well as dynamical structure of systems. From these models, sequential processes of a system are not difficult to recognize but concurrent ones still are. Therefore, how to transform sequential processes of a system onto concurrent ones is a great problem on theory and application indeed.

A simple algorithm for finding the normal form of traces is a good solution for systems represented by reliance alphabet. Place/Transition net is one of suitable models to represent concurrent systems. But the language generated by a P/T net is sequential. It shows us only sequential performances of the net's processes. So when several transitions can be performed concurrently. We will concentrate on building an iterative algorithm for transforming sequential processes of a P/T net into concurrent ones.

This paper is organized as follows. Section 2 constructs a "shift-left" algorithm for finding the normal form of a trace. In Section 3 we propose the notation of concurrency in P/T nets. Section 4 builds up an iterative algorithm for finding sequences of maximal concurrent steps on a Place/Transition net from the net's sequence of single steps. Finally, some conclusions are given in Section 5.

2. Normal form of trace and its finding

The theory of traces was originated by A. Marzurkiewicz in [4] as an attempt to provide a good mathematical description of the behaviour of concurrent systems. The normal form of a trace gives an optimal way to perform a process represented

by the trace. In this section we will construct a simple iterative algorithm for finding the normal form of a trace.

2.1. Independence relation and traces

Definition 2.1: Let Σ be an alphabet.

1. *An independence relation* over Σ is a symmetric and irreflexive binary relation over Σ.

2. *A reliance alphabet* is a couple $C = (\Sigma, I)$, where Σ is an alphabet and I is an independence relation over Σ.

Considering adjacent independent symbols in a string to be commuting, one can relate different strings as follow.

Definition 2.2: Let $C = (\Sigma, I)$ be a reliance alphabet.

1. The relation $=_C \subseteq \Sigma^* \times \Sigma^*$ is defined as follow: for $x, y \in \Sigma^*$, $x =_C y$ if and only if there exist $x_1, x_2 \in \Sigma^*$ and $(a,b) \in I$ such that $x = x_1abx_2$ and $y = x_1bax_2$.

2. The *C-equivalence relation* $\equiv_C \subseteq \Sigma^* \times \Sigma^*$ is defined as the least equivalence relation over Σ^* containing $=_C$.

Thus, $\equiv_C$ identifies 'commutatively similar' strings – each such group of strings is called a trace.

Definition 2.3: Let $C = (\Sigma, I)$ be a reliance alphabet.

1. For an $x \in \Sigma^*$, *the trace of* x, denoted by $[x]_C$, is the equivalence class of $\equiv_C$ containing x.

2. *A trace* over C is a set t of strings over Σ such that $t = [x]_C$ for some $x \in \Sigma^*$ and x is called *a representative of* t.

3. A set of traces over C is called *a trace language* over C.

Example 2.4: Let $C = (\Sigma, I)$ be the reliance alphabet given by the undirected graph in Fig. 1.

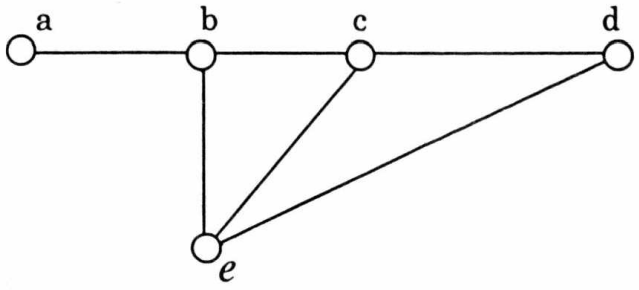


Fig.1

The trace t of $abcecd = [abcecd]_C = \{abcecd, bacecd, acbecd, abeccd, abcced, abcedc, baeccd, bacced, bacedc, acebcd, acbced, acbedc, aebccd, abecde, abccde\}$.

2.2. Normal form of traces

One can define a composition operation on trace language.

Definition 2.5: Let C = (Σ, I) be a reliance alphabet.

For traces t_1, t_2 over C, *the trace composition* of t_1 and t_2, denoted by $t_1.t_2$, is defined by $t_1 \cdot t_2 = [x_1.x_2]_C$, where x_1, $x_2 \in \Sigma^*$ are representatives of t_1 and t_2 respectively.

In general, a trace can be obtained as a trace composition of other traces, but trace decompositions of the given trace do not have to be unique. Hence it is desirable to have 'normal form' decomposition of traces.

Definition 2.6: Let C = (Σ, I) be a reliance alphabet and t be a trace over C. The following decomposition $t = t_1.t_2. \square t_m$, such that:

1. for all $1 \le i \le m$, $t_i \neq \varnothing$;

2. for all $1 \le i \le m$, t_i can be written as $[u_i]$, where $u_i \in \Sigma^*$, $\#_a(u_i)=1$ for each $a \in$ alph(u_i), and $(a,b) \in$ I for all $a, b \in$ alph(u_i) such that $a \neq b$; and

3. for all $1 \le i \le m$-1, if $t_i = [u_i]$ and $t_{i+1} = [u_{i+1}]$ then, for each $a \in$alph(u_{i+1}), there exists $b \in$ alph(u_i), such that $(a,b) \notin$ I; is called the normal form of the trace t.

In [1] J. I. Aalbersberg and G. Rozenberg pointed out that every trace can be uniquely decomposed into a normal form, i.e. a minimal number of 'maximal independent' parts. They have built two algorithms for finding the normal form of traces. The first algorithm is based on integer pointers and the second one is based on dependence graphs.

In the next subsection we build a simple iterative algorithm for finding the normal form of a trace from its representative.

2.3. Algorithm for finding the normal form of traces

We give an intuitive description of the algorithm. First we consider the input string as a sequence of single parts, every part consists of one symbol. The algorithm repeatedly goes back on the sequence of parts and shifts a symbol from a part onto the previous part if the symbol is independent with every symbol belonging to the previous part. When no symbol can be shifted, the algorithm terminates.

Algorithm 2.7 ("Shift-left" algorithm 1):

Input: A reliance alphabet C = (Σ, I) and a string $w \in \Sigma^*$.

Declaration: Let $k = |w|$, let v be an array of length k over 2^Σ and let u be an array of length k over Σ^*.

Computation:

1. for $i := 1$ to k do $v(i) := \{w[i]\}$;
2. $j := 2$;
3. repeat
4. for every $a \in v(j)$ do
5. begin

6. $i := j - 1$; OK := *false*;

7. while $\forall b \in v(i), (a,b) \in I$ do begin $i := i - 1$; OK := *true* end;

8. if OK then begin $v(j) := v(j) \setminus \{a\}$; $v(i) := v(i) \cup \{a\}$ end;

9. end;

10. if $v(j) = \varnothing$ then ignore $v(j)$ and decrease $k := k - 1$;

11. $j := j + 1$;

12. until $j > k$;

13. for $i := 1$ to k do $u(i) := \text{lin}(v(i))$;

Output: The strings u(1), u(2), ..., u(k).

Example 2.8: Let C be the reliance alphabet given in Example 2.4 and let $w = aecbbed$. Computing by the above algorithm, we have:

v : {a}, {e}, {c}, {b}, {b}, {e}, {d} ; $k = 7$

v : {a}, {e, c}, {b}, {b}, {e}, {d} ; $k = 6$

v : {a, b}, {e, c}, {b}, {e}, {d} ; $k = 5$

v : {a, b}, {e, c, b}, {e}, {d} ; $k = 4$

v : {a, b}, {e, c, b}, {e, d} ; $k = 3$

Hence, the output of the algorithm is $u(1) = ab$, $u(2) = bce$, $u(3) = de$.

Formalizing the above we get the following result.

Theorem 2.9: Let C = (Σ, I) be a reliance alphabet and let $w \in \Sigma^*$. Let strings $u(1), u(2),..., u(k)$ be the output of Algorithm 2.7 for the input (C,w). Then $[u(1)].[u(2)] \ldots [u(k)]$ is the normal form of $[w]$.

The algorithm is very simple and easy to implement on computer. It transfers a sequential process represented by a string into a concurrent process. The complexity of this algorithm is $O(k^2)$.

3. Concurrency in P/T nets

First of all, we recall some notations concerning Petri nets.

3.1. Place/Transition nets

A Petri net is a triple N = (P, T, F), where P, T are disjoint sets and $F \subseteq (P \times T) \cup (T \times P)$ is a relation, so-called *the flow relation* of the net N.

A net is *simple* if and only if its two different elements have no common pre-set and post-set. A simple net is used to represent statistical structure of a system. From a simple net one can construct different net models by adding some components for representing dynamical structure of the system. The Place/Transition net is such a net and is defined in [6] as follows:

Definition 3.1: The 6-tuple $\Sigma = (P, T, F, K, M^0, W)$ is called *a Place/Transition net* iff:

1. $\mathcal{N} = (P, T, F)$ is a simple net, whereas an element of P is called *a place* and an element of T is called *a transition*.

2. $K : P \to N \cup \{\infty\}$ is a function showing *a capacity* on each place.

3. $W : F \to N \setminus \{\infty\}$ is a function assigning *a weight* on each arc of the flow relation F.

4. $M^0 : P \to N \cup \{\infty\}$ is *an initial marking*, which is not greater than capacity on each place, i.e.: $\forall p \in P,\ M^0(p) \le K(p)$.

The initial marking represents given tokens on each place of a net. The tokens are not greater than the capacity of the corresponding place. If tokens on each place belonging to the pre-set of some transition are greater than or equal to weight of the arc connecting this place to the transition, i.e. it is enough for "paying", then the initial marking can activate the corresponding transition. After performing the transition, tokens on each place belonging to the pre-set of this transition are decreased by weight of the arc connecting the corresponding place to this transition and tokens on each place belonging to the post-set of this transition are increased by weight of the arc connecting this transition to the corresponding place. It must be ensured that new tokens on each place are not greater than the capacity of that place.

When the initial marking activates some transition, the transition is performed and then we get a new marking, the new marking can activate another transition and the process repeatedly continues in such a way. Therefore, the activities happened on a P/T net will be mathematically formalized as follows:

The marking $M : P \to N \cup \{\infty\}$ can activate a transition t iff:

1. $\forall p \in {}^\bullet t,\ M(p) \ge W(p, t)$ and

2. $\forall p \in t^\bullet,\ M(p) \le K(p) - W(t, p)$, where ${}^\bullet t, t^\bullet$ are the pre-set and the post-set of t.

In such a case, the marking M is so-called *t-activating*. After performance of the transition t, we get the following new marking:

$$M'(p) = \begin{cases} M(p) - W(p,t) & , \text{if } p \in {}^\bullet t \setminus t^\bullet \\ M(p) + W(t,p) & , \text{if } p \in t^\bullet \setminus {}^\bullet t \\ M(p) - W(p,t) + W(t,p) & , \text{if } p \in {}^\bullet t \cap t^\bullet \\ M(p) & , \text{otherwise} \end{cases}$$

and we often write that: $M[t > M'$.

The marking M' can activate some other transition and then we get another marking M"... The set of all markings reachable from the marking M is denoted by $\mathcal{R}[M]$. This set is so-called *a state space* of the net Σ. It is the environment for activities to be happened.

Let $\Sigma = (P, T, F, K, M^0, W)$ be a P/T net. Let $M^1, M^2, \ldots, M^n$ be a sequence of markings and $t^1, t^2, \ldots, t^n$ be a sequence of transitions of the net Σ, such that:

$$M^{i-1}[\, t^i > M^i \,,\ i = 1, 2, \ldots, n.$$

The sequence $M^0[\, t^1 > M^1[\, t^2 > M^2 \ldots M^{n-1}[\, t^n > M^n$ illustrates a sequential process of the net and the word $\alpha = t^1 t^2 \ldots t^n \in T^*$ is called *activities sequence* on the net.

The set of all activities sequences on the net Σ is called *the language generated by the net* Σ and denoted by $L(\Sigma)$:

$$L(\Sigma) = \{\, t^1 t^2 \ldots t^n \mid \exists\, M^1, M^2, \ldots, M^{n-1}, M^n \in \mathcal{R}[M^0] :$$

$$M^0[\, t^1 > M^1[\, t^2 > M^2 \ldots M^{n-1}[\, t^n > M^n \}$$

But the language generated by a net is sequential. It shows us only sequential performances of the P/T net's processes.

So when several transitions of the net can be performed concurrently.

3.2. Sequence of maximal concurrent steps

Let $U \subseteq T$ be a subset of transitions of the net Σ and $U \neq \varnothing$.

Definition 3.2: The subset U is called *a step* on the net Σ iff there is a marking $M \in \mathcal{R}[M^0]$ satisfying the following inequalities:

1. $\forall\, p \in {}^\bullet U,\ M(p) \geq \sum_{t \in U} W(p, t)$ and

2. $\forall\, p \in U^\bullet,\ M(p) \leq K(p) - \sum_{t \in U} W(t, p).$

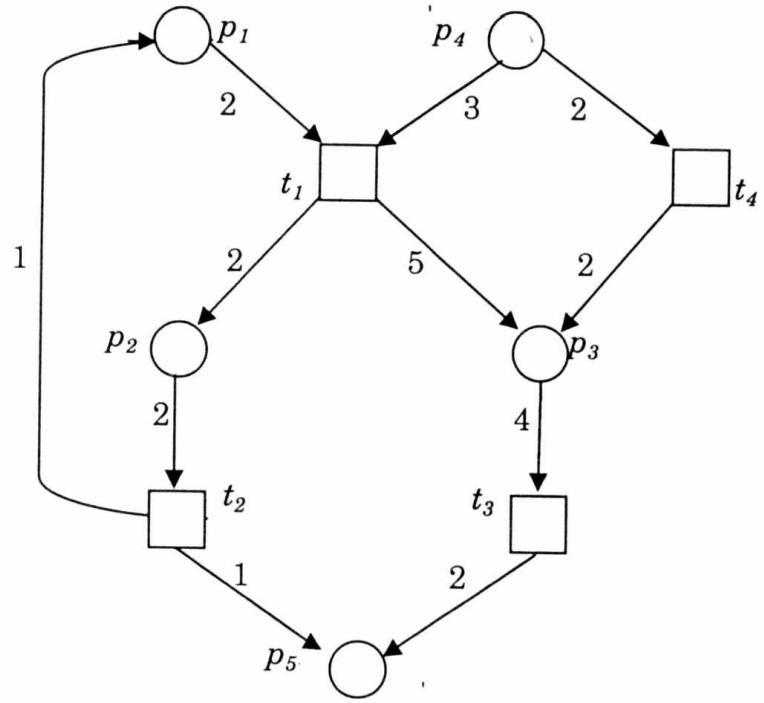
In such a case, the transitions in the step U can be performed concurrently and after their performance we get the following marking:

$$M'(p) = \begin{cases} M(p) - \sum_{t \in U} W(p,t) & , \text{if } p \in {}^\bullet U \setminus U^\bullet \\ M(p) + \sum_{t \in U} W(t,p) & , \text{if } p \in U^\bullet \setminus {}^\bullet U \\ M(p) - \sum_{t \in U} W(p,t) + \sum_{t \in U} W(t,p) & , \text{if } p \in {}^\bullet U \cap U^\bullet \\ M(p) & , \text{otherwise} \end{cases}$$

We also denote that: $M[\, U > M'$ and the marking M is called U-activating.

Such as above, we can find sequences of steps on the net. As big are the steps as high concurrency is.

Example 3.3: Consider a P/T nets presented by the labelled directed bipartite graph in Figure 2.



	K	M^0
p_1	**3**	3
p_2	**2**	0
p_3	**8**	1
p_4	**5**	5
p_5	**4**	1

Fig 2. A P/T net

On this net we recognize the following sequential process:

	M^0	$[t_1>$	M^1	$[t_2>$	M^2	$[t_3>$	M^3	$[t_4>$	M^4
p_1	3		1		2		2		2
p_2	0		2		0		0		0
p_3	1		6		6		2		4
p_4	5		2		2		2		0
p_5	1		1		2		4		4

The initial marking M^0 can activate the subset $\{t_1, t_4\}$ and we have:

$$M^0 [\{t_1, t_4\} > M^{1'} = (1, 2, 6, 0, 1).$$

So the subset $\{t_1, t_4\}$ is a concurrent step on this P/T net.

Let $M^0[U_1 > M^1[U_2 > M^2 \ldots M^{k-1}[U_k > M^k$ be a sequence of steps on the net Σ. The sequence illustrates a concurrent process on the net. What steps are "biggest"? The following definition answers this question.

Definition 3.4: The sequence of steps $M^0[U_1 > M^1[U_2 > M^2 \ldots M^{k-1}[U_k > M^k$ is called *a sequence of maximal concurrent steps* iff for each $i = 1, 2, \ldots, k$-1:

$$\forall t \in U^{i+1},\ M^{i-1} \text{ is not } (U_i \cup \{t\}) \text{ -activating.}$$

If transitions of each step can be performed concurrently, then the total time for performance of the process decreases considerably. Therefore, we always expect to find sequences of maximal concurrent steps and at that time, the performance of processes becomes optimal. In order to do so, we propose second "shift-left" algorithm in the next section.

4. Algorithm finding sequences of maximal concurrent steps

Let $\Sigma = (P, T, F, K, M^0, W)$ be a P/T net. Each sequential process: $M^0[t^1 > M^1[t^2 > M^2 \ldots M^{k-1}[t^k > M^k$ of the net may be considered as a sequence of single steps: $M^0[\{t^1\} > M^1[\{t^2\} > M^2 \ldots M^{k-1}[\{t^k\} > M^k$ and as an input of our algorithm.

The algorithm repeatedly goes back on the sequence of steps and shifts a transition from a step into the previous step if when adding the transition to this previous step, the obtained step becomes activated by the corresponding marking. When no transition can be shifted, the algorithm terminates.

Algorithm 4.1 ("Shift-left" algorithm 2):

Input: A sequential process $M^0[t_1 > M^1[t_2 > M^2 \ldots M^{k-1}[t_k > M^k$ on a P/T net.

Output: A sequence of maximal concurrent steps on the net.

1. for $i := 1$ to k do $U_i := \{t_i\}$;
2. for $j := 1$ to k-1 do
3. begin
4. $i := j$;
5. while $i >= 1$ do
   begin
   for every transition $t \in U_{i+1}$,
   if M^{i-1} is $(U_i \cup \{t\})$-activating then
   begin we replace:

$$M^i(p) = \begin{cases} M^{i-1}(p) - \sum_{t \in U_i \hat{a}(t)} W(p,t) & , \text{if } p \in {}^\bullet(U_i \cup \{t\}) \setminus (U_i \cup \{t\})^\bullet \\ M^{i-1}(p) + \sum_{t \in U_i \hat{a}(t)} W(t,p) & , \text{if } p \in (U_i \cup \{t\})^\bullet \setminus {}^\bullet(U_i \cup \{t\}) \\ M^{i-1}(p) - \sum_{t \in U_i \hat{a}(t)} W(p,t) + \sum_{t \in U_i \hat{a}(t)} W(t,p) & , \text{if } p \in {}^\bullet(U_i \cup \{t\}) \cap (U_i \cup \{t\})^\bullet \\ M^{i-1}(p) & , \text{otherwise} \end{cases}$$

and $U_i := U_i \cup \{t\}$, $U_{i+1} := U_{i+1} \setminus \{t\}$, i.e. we "shift-left" the transition t from the step U_{i+1} into the previous step U_i. After shifting, if $U_{i+1} = \varnothing$ then we ignore both U_{i+1} and M^{i+1};

end ;

$i := i - 1$

end :

6. end ;
7. stop.

The algorithm goes back because we want not only making steps maximal concurrent but also reducing the number of steps.

Example 4.2: We apply this algorithm to the P/T net drawn in Fig. 2 with the input $M^0[t_1 > M^1[t_2 > M^2[t_3 > M^3[t_4 > M^4$.

- After 1st going back:

The marking M^0 is $\{t_1, t_2\}$- activating, so we shift left t_2 and get the following sequence:

	M^0	$[\{t_1,t_2\} >$	M^1	$[\{t_3\} >$	M^2	$[\{t_4\} >$	M^3
p_1	3		2		2		2
p_2	0		0		0		0
p_3	1		6		2		4
P_4	5		2		2		0
P_5	1		2		4		4

- After 2nd going back:

The marking M^1 now is $\{t_3,t_4\}$-activating, so we shift left t_4 and obtain the following sequence:

	M^0	$[\{t_1,t_2\} >$	M^1	$[\{t_3,t_4\} >$	M^2
P_1	3		2		2
P_2	0		0		0
P_3	1		6		4
P_4	5		2		0
P_5	1		2		4

The obtained sequence M^0 $[\{t_1,t_2\} > M^1[\{t_3,t_4\} > M^2$ is indeed a sequence of maximal concurrent steps of the net..

Theorem 4.3: When Algorithm 4.1 terminates, its output is the sequence of maximal concurrent steps of the P/T net.

The complexity of this algorithm is $O(|T|.|P|.k^2)$. So the complexity is square in the number of steps. This algorithm is simple and easy to implement on computers.

5. Conclusion

In the paper, we construct one more a very simple algorithm for finding the normal form of a trace; propose the notation of concurrent step on a P/T net and build up an efficient algorithm to transform sequential processes of a P/T net into concurrent ones. These algorithms are not only useful for concurrency control on systems but also suitable for calculating concurrent behaviours of a system with

dynamical structures. The algorithms may be applied for processing transactions on database, for finding the normal form of a semi-trace.

Acknowledgement: This paper was written during my stay at De Montfort University, Leicester, UK. I would like to thank Professor Hongji Yang, Dr. Dang Van Hung and the IIST/UNU for my valuable time at Leicester.

References

1. J.I. Aalbersberg and G. Rozenberg, *Theory of Traces,* Theoretical Computer Science **60**(1988), pp. 1-82.
2. J.A. Bergstra and J.W. Klop, *Algebra for communicating processes with abstraction*, Theoretical Computer Science 37, **1**(1985), pp. 77-121.
3. R. Janicki, *Trace semantics for communicating sequential processes,* Tech. Report R-85-12, Univ. of Aalborg, Denmark, 1985.
4. A. Mazurkiewicz, *Concurrent program schemes and their interpretations*, DAIMI Report PB-78, Aarhus Univ., Denmark, 1977.
5. R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.
6. W. Reisig, *Petri Nets: An Introduction*, Springer-Verlag, 1985.
7. A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice Hall, 1998.
8. H.C. Thanh, *An algorithm for finding the normal form of traces and synchronous traces*, Journal of Computer Science and Cybernetics, Vol. 17, No. 1(2001), pp. 72-77.
9. H.C. Thanh, *Control problem on Timed place/transition nets,* VNUH Journal of Science, Vol. XX, No. **4**(2004), pp. 48-55.