ISSN 0105-8517

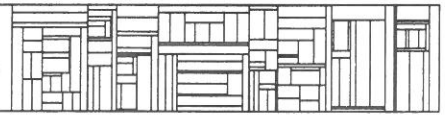
ANALYSIS OF CONCURRENT ALGORITHMS

by

Jørgen Staunstrup

DAIMI PB-129
January 1981

Computer Science Department
AARHUS UNIVERSITY
Ny Munkegade – DK 8000 Aarhus C – DENMARK
Telephone: 06 – 12 83 55



ANALYSIS OF CONCURRENT ALGORITHMS

Jørgen Staunstrup

Computer Science Department
Aarhus University
Ny Munkegade
DK-8000 Aarhus C

Abstract

Analyzing the running time of a concurrent algorithm can be as important as verifying its partial correctness or termination. A simple technique for analyzing the running time of a concurrent algorithm is presented. To analyze an algorithm with concurrent processes, the interaction between the processes must be considered. This is done by using the communication sequences of the processes as the basis of the analysis. The technique is used for analyzing and comparing three concurrent algorithms for finding the root of a real function.

1. INTRODUCTION

A concurrent algorithm specifies a number of processes P_1, P_2 ... , P_n which can be executed in parallel. This paper presents an example of how the running time of such a concurrent algorithm can be estimated. Techniques for estimating the running time of sequential algorithms (only one process) are very well developed [Knuth 1968] and [Aho, Hopcroft, and Ullman 1974]. Analyzing a concurrent algorithm with several processes presents additional problems because the interaction between the processes must be taken into account. Such interaction is for example necessary when the processes exchange intermediate results. The interaction between concurrent processes can be very complex to analyze, which is also why it is difficult to construct and verify concurrent algorithms. The challenge is of course to avoid the complexity in reasoning about the algorithms and still obtain realistic results.

2. ROOT SEARCHING

In this section a concurrent algorithm for finding the root of a continuous function, H, is presented. Assume that H is a real continuous function defined on the closed interval [a, b]. Assume furthermore that $H(a) \cdot H(b) \leq 0$ and that H has only one root in [a, b].

There are many well known sequential algorithms for finding the root, for example binary search. Let T_H denote the average time it takes to evaluate H. If T_H dominates other quantities in the running time, then it is well known that the running time, B_T, for binary search is:

$$B_T \approx T_H \cdot \log \frac{l_0}{eps}$$

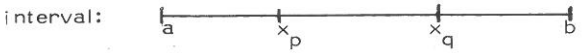
where eps is the accuracy with which the root is obtained and $l_0 = b-a$. (For binary search the worst, best, and average case running times are the same).

The above running time can be improved by letting several processes evaluate H at different interval points concurrently.

2.1 A Two Process Algorithm

The following algorithm [Kung 1976] with only two concurrent processes is simple, but manageable.

Two processes, p and q, evaluate the function H at two different interval points: x_p and x_q.

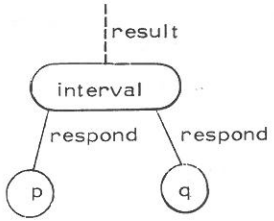


Like the binary search, the algorithm works by narrowing the interval. Assume that p finishes its evaluation of H first and $H(a) \cdot H(x_p) \leq 0$, i.e. the root is in $[a, x_p]$. The interval is now changed to $[a, x_p]$, therefore the work of q is wasted and q must be directed to work in the new interval $[a, x_p]$ as soon as possible. If on the other hand $H(a) \cdot H(x_p) > 0$, the root is in the interval $[x_p, b]$. In this case the work currently being done by q is utilized.

As we shall see later, the placement of x_p and x_q is crucial for the efficiency of the algorithm. Let D be a function for calculating x_p and x_q from a and b:

$$x_p = D(a, b, p) \quad \text{and} \quad x_q = D(a, b, q)$$

Administration of the interval is the central part of the algorithm. The interval is an abstract date type with two operations, respond and result:



The operation respond is used by p and q every time they have completed one evaluation of H. The operation result gives the root.
The notation from Staunstrup [1978] is used to specify such an abstract data type.

```
type interval (eps: real);
     state a, b: real;

     operation respond (Hx: real; id: (p, q); var x: real);
     when
1)        x ∈ [a, b], b-a>eps → b-a<b0-a0, x = D(a, b, id), H(a) · H(b) ≤ 0 []
2)        x ∉ [a, b], b-a>eps → x = D(a, b, id)
     end;

     operation result (var r: real);
     when
          b-a ≤ eps → r = a0
end  interval.
```

The abstract data type has a state space with two components a and b. How these are given an initial value is not considered here. State changes are specified by transitions of the form R → U, where R and U are predicates. A transition R → U can only take place if R is satisfied, performing the transition results in a state satisfying U. Each transition is indivisible. The endpoints of the interval should change when one of the processes finishes evaluation of H on an interval point. The new endpoints of the interval can by any pair of points a, b such that i) b-a is smaller than b_0-a_0 (a_0, b_0 are the values of a and b before the transition), and ii) $H(a) \cdot H(b) \leq 0$ (the root is within [a, b]). These requirements are specified in line 1).

When the interval is changed, one of the processes might work outside the current interval, i.e. $x \notin [a, b]$. In this case the process is directed to work on the correct interval point next time it calls respond. This is specified in line 2).

It is quite easy to show that the above specified algorithm is partially correct and that it converges, the proof is omitted here.

2.2 Communication Sequences

The communication sequences of an abstract data type are all sequences of completed operation calls which arise when the abstract data type is used. The communication sequences for the interval are of the form:

$$\sigma = \text{respond}(H_1, i_1, x_1).\ \text{respond}(H_2, i_2, x_2) \ldots \text{respond}(H_k, i_k, x_k).\ \text{result}(r)$$

whereas the sequence:

$$\sigma = \text{respond}(H_1, i_1, x_1). \ \ldots\ .\ \text{result}(r).\ \text{respond}(H_j, i_j, x_j). \ \ldots$$

is not a communication sequence, because no call of respond can be completed after a call of result is completed.

The algorithm is analyzed by showing various properties of the communication sequences. Assume that σ is a communication sequence for the interval, then the following notation is convenient:

$|\sigma|$: the length of σ i.e. the number of operations in σ
$\|\sigma\|_p$: the number of occurrences of respond(-, p, -) in σ
$\|\sigma\|_q$: the number of occurrences of respond(-, q, -) in σ

Finally, let Π denote the set of all communication sequences for the interval.

3. COMPLEXITY MEASURES

In this section a number of alternative ways of analyzing the running time of the above algorithm are considered. It was assumed that evaluating H dominates the running time. Since there is one evaluation of H for each call of respond, there is a direct relationship between the length of a communication sequence and the running time of the corresponding execution. Consider a communication sequence σ, then $\|\sigma\|_p$ is the number of function evaluations performed by P and $\|\sigma\|_q$ is the number of evaluations performed by Q. If we assume that both processes on the average take time T_H to evaluate H, the running time of the execution corresponding to σ is:

$$T(\sigma) = \max(\|\sigma\|_p, \|\sigma\|_q)\ T_H.$$

Different complexity measures are useful for different purposes, but usually the worst, best, and average case is considered. These can, however, not be defined straightforwardly as:

$$\max_{\sigma\in\Pi}(T(\sigma)),$$

$$\min_{\sigma\in\Pi}(T(\sigma)), \quad \text{and}$$

$$\sum_{\sigma\in\Pi} p(\sigma) \cdot T(\sigma) \qquad \text{(where } p(\sigma) \text{ is the probability of } \sigma).$$

The maximum of $T(\sigma)$ is obtained when

$$\sigma = \text{respond}(-,p,-).\ \ldots\ \text{respond}(-,p,-)$$

i.e. only one process responds. Although this is a maximum, it is much too conservative. If two processes are executed with approximately the same speed, it could never be observed in an execution of the above specified algorithm. Instead of considering the set of all communication sequences, the notion of an observable communication sequence is defined. The observable communication sequences is a subset of all communication sequences which is selected as a model of the behaviour of the algorithm.

The observable sequences can for example be defined by a regular expression or a finite state machine. In section 4 several examples of observable sequences are given. The set of all observable communication sequences is denoted τ, $\tau\subseteq\Pi$. The following complexity measures can now be defined:

Worst observable case:

$$W(\tau) = \max_{\sigma\in\tau} T(\sigma)$$

Best observable case:

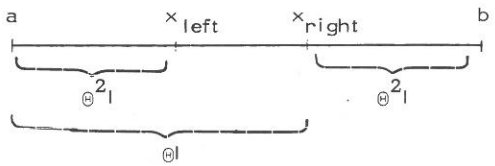
$$B(\tau) = \min_{\sigma\in\tau} T(\sigma)$$

4. IMPLEMENTATIONS OF THE CONCURRENT SEARCHING ALGORITHM

Two different implementations of the specification from section 2 are considered and it is shown how their running time is analyzed. The only difference between the two implementations is in the choice of D, i.e. the subdivision of the interval.

4.1 The Golden Section Algorithm

Kung [1976] has suggested choosing the subdivision points x_o and x_q as the golden section ($\Theta = (\sqrt{5}-1)/2 \approx 0.618$) of the interval $[a,b]$.



$$l = b - a$$
$$D(a,b,\text{left}) = a + \Theta^2 l$$
$$D(a,b,\text{right}) = b - \Theta^2 l \quad (= a + \Theta l)$$

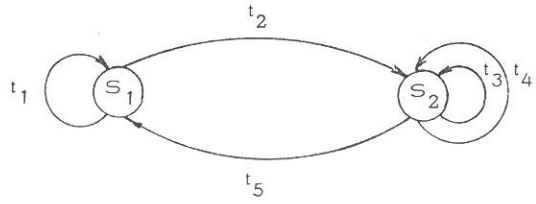
where left is the name of the process (p or q) working on the leftmost subdivision point, and right is the other process. Note, that $x_{right} = x_{left} + \Theta^2(\Theta l)$, so when the interval is reduced to $[x_{left}, b]$, x_{right} automatically becomes the left division point of the new interval $[x_{left}, b]$.

We want to find an expression for σ, from which it is easy to determine the reduction of the interval length performed by each transition. The algorithm can be in two different states, characterized by the following predicates:

S_1: $\quad x_p \in [a,b]$ and $x_q \in [a,b]$

S_2: $\quad x_p \notin [a,b]$ xor $x_q \notin [a,b]$

The following transitions can be made between these states:



The two transitions t_3 and t_4 differ in that t_3 reduces the interval length by Θ^2 and t_4 reduces it by Θ. The communication sequences are determined by the following regular expressions:

$$\sigma : \lfloor t_1, \; t_2(t_3, t_4)^* \; t_5 \rceil^*$$

Each transition reduces the interval length by some fraction α $(0 < \alpha \leq 1)$. If the transition is substituted by its reduction α in the above expression we get:

$$eps \approx \lfloor \Theta, \; \Theta^2(\Theta^2, \Theta)^* \cdot 1 \rceil^* \cdot l_0,$$

where l_0 is the initial length of the interval. The maximum of $|\sigma|$ (length of σ) can immediately be derived from this expression, namely the n such that:

$$eps \approx \Theta^n \cdot l_0 \text{ i.e. } n = \log_{\phi} \frac{l_0}{eps} \quad \text{where } \phi = \frac{1}{\Theta}$$

Similarly for the minimum: $\log_{\phi} 2 \frac{l_0}{eps}$

The set of observable communication sequences for the golden section algorithm is defined by the following regular expression:

$$\sigma : \lfloor t_1, \; t_2t_3t_5, \; t_2t_4t_5, \; t_2t_5 \rceil^*$$

Thus, in the observable sequences, there are approximately the same number of responds from p and q, without requiring strict alternation. The length of σ, $|\sigma|$, is found in the same way as above:

$$eps \approx \lfloor \Theta, \; \Theta^2\Theta^2 \cdot 1, \; \Theta^2 \cdot \Theta \cdot 1, \; \Theta^2 \cdot 1 \rceil^* \cdot l_0$$

From this, it is easily seen that:

$$\frac{3}{2} \log_{\phi} 2 \frac{l_0}{eps} \leq |\sigma| \leq \log_{\phi} \frac{l_0}{eps} .$$

In the observable sequences each of the two processes contribute with approximately half of the responds, so:

$$W(\tau): \quad \frac{1}{2} \log_{\phi} \frac{l_0}{eps} \approx \underline{0.72B}_T$$

$$B(\tau): \quad \frac{3}{2} \cdot \frac{1}{4} \log_{\phi} \frac{l_0}{eps} \approx \underline{0.54B}_T$$

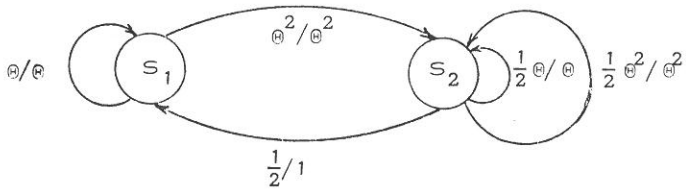
4.1.1 Analysis of The Average Running Time

The finite state model is also used to find the average running time. Consider first the set of all communication sequences, Π, and assume that:

i) the probability of finding the root in a given interval is proportional to the length of the interval,

ii) the system is memoryless, i.e. there is always the same probability, 1/2, of process p responding next, regardless which process responded last. Similarly there is always probability 1/2 of process q responding next.

With these assumptions the model becomes:



(The notation α/β on a transition means that the transition is performed with probability α and makes a reduction of β in the interval length).

The average running time is found by first finding the average reduction of the interval length in each call of respond. This average is called R. The probabilities of being in the two states are:

$$p(S_1) = \frac{1}{3-2\Theta}, \quad p(S_2) = \frac{2\Theta^2}{3-2\Theta} \qquad \text{[Feller 1950]}$$

Let T be the set of all transitions. For any $t_i \in T$, r_i is the reduction in the interval

length made by t_i and p_i is the probability of performing t_i. The average reduction of the interval length performed by one transition is:

$$R = \prod_{t_i \in T} r_i^{p_i}$$

To see this view the p_i's as relative frequencies, then there will be on the average k_i, $k_i/n = p_i$ occurences of t_i in a sequence of length n.

The reductions of such a sequence can be expressed as:

$$r_{i_1} \cdot r_{i_2} \cdot \ldots \cdot r_{i_n} = r_1^{k_1} \cdot r_2^{k_2} \cdot r_j^{k_j}$$

$$= \prod_{t_i \in T} r_i^{k_i}$$

The average reduction of this sequence is the number R such that:

$$R^n = r_1^{k_1} \cdot r_2^{k_2} \cdot r_j^{k_j}$$

thus:

$$R = \sqrt[n]{r_1^{k_1} \cdot r_2^{k_2} \cdot r_j^{k_j}}$$

$$= r_1^{k_1/n} \cdot r_2^{k_2/n} \cdot r_j^{k_j/n}$$

$$= \prod_{t_i \in T} r_i^{k_i/n}$$

$$= \prod_{t_i \in T} r_i^{p_i}$$

The average reduction for the golden section algorithm becomes:

$$R = (\Theta^2)^{p(S_1)} \cdot \Theta^{2}_{(\Theta)}{}^{p(S_1)} \cdot \Theta_{(\Theta^2)}{}^{\frac{1}{2}\,p(S_2)} \cdot \Theta^{2}_{(\Theta)}{}^{\frac{1}{2}\,p(S_2)} \cdot \Theta_{.1}{}^{\frac{1}{2}\,p(S_2)}$$

$$\approx 0.594$$

The average length of the communication sequence is therefore:

$$A(|\sigma|): \log_{1/R} \frac{l_0}{eps} \approx 1.34\, B_T$$

Note, this average is an average over all communication sequences.

Since the worst and best case analyses were based on the observable sequences, it would be natural to use these for the average case analysis also. The method described above can be used for this, but there are more states in the model and the average reduction which is obtained is almost the same as the above, but more tedious to compute. The conclusion is that no matter which of the models is used, we have:

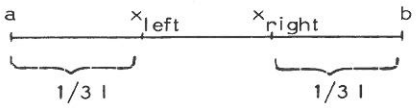
$$|\sigma| \approx 2 * \|\sigma\|_p \; (\approx 2 * \|\sigma\|_p).$$

From this it follows that the average running time of the golden section algorithm is:

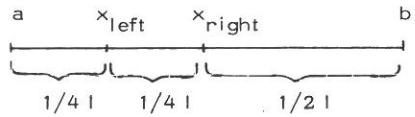
$$A(\Pi): \frac{1}{2} A(|\sigma|)$$
$$\approx \underline{0.67\, B_T}$$

4.2 The Equidistant Algorithm

The most obvious way to subdivide the interval is to cut the interval in three pieces of the same length.



If the left process responds that the root is in the interval $[x_{left}, b]$ then x_{right} is working on the center point of the new interval. The new x_{left} becomes:



These two patterns are sufficient. As it was the case with the golden section algorithm, different calls of respond may give different reductions of the interval length. The same technique is therefore used to determine the length of the communication sequences.

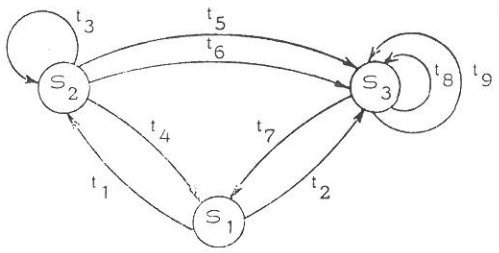
The algorithm can be in three different states, characterized by the following three predicates:

S_1: $x_p \in [a,b]$ and $x_q \in [a,b]$ and $x_p = a + \frac{1}{3}l$ and $x_q = b - \frac{1}{3}l$

S_2: $x_p \in [a,b]$ and $x_q \in [a,b]$ and $x_p = a + \frac{1}{4}l$ and $x_q = b - \frac{1}{2}l$

S_3: $(x_p \notin [a,b]$ xor $x_q \notin [a,b])$ and $(x_p = a + \frac{1}{3}l$ or $x_q = b - \frac{1}{3}l)$

The following transitions can be made between these states:



The communication sequences are determined by the following regular expressions:

$$\sigma: [t_1 t_3 * t_4, (t_1 t_3 * (t_5, t_6), t_2)(t_8, t_9) * t_7] *$$

The observable communication sequences are defined by the following regular expression:

$$\sigma: [t_1 t_3 * t_4, (t_1 t_3 * (t_5, t_6), t_2) t_8 t_7, (t_1 t_3 * (t_5, t_6), t_2) t_9 t_7, (t_1 t_3 * (t_5, t_6), t_2) t_7] *$$

By substituting each transition with its reduction of the interval length we get:

$$\text{eps} \approx [\tfrac{2}{3}(\tfrac{1}{2}) * \tfrac{3}{4}, (\tfrac{2}{3}(\tfrac{1}{2}) * (\tfrac{1}{2}, \tfrac{1}{4}), \tfrac{1}{3})\tfrac{1}{3}1, (\tfrac{2}{3}(\tfrac{1}{2}) * (\tfrac{1}{2}, \tfrac{1}{4}), \tfrac{1}{3})\tfrac{2}{3}1, (\tfrac{2}{3}(\tfrac{1}{2}) * (\tfrac{1}{2}, \tfrac{1}{4}), \tfrac{1}{3})1] * \cdot l_0$$

It is easy to see that the maximum length of the communication sequence is obtained by the path: $t_1 t_4$. The minimum length is obtained by the path: $t_2 t_8 t_7$.

$$3 \cdot \log_9 \frac{l_0}{\text{eps}} \le |\sigma| \le 2 \log_2 \frac{l_0}{\text{eps}}$$

In the observable sequences each of the two processes contribute with approximately half of the responds, so:

$$W(\tau): \tfrac{1}{2} \cdot 2 \log \frac{l_0}{\text{eps}} = \underline{B_T}$$

$$B(\tau): \tfrac{1}{2} \cdot 3 \log \frac{l_0}{\text{eps}} = \underline{0.47\, B_T}$$

In the worst case, there is no gain in using two processors, the running time is the same as the running time of binary search.

The average running time is found in the same way as for the golden section algorithm.

$$A(\Pi): \underline{0.69\, B_T}$$

On the average the equidistant algorithm is slightly slower than the golden section algorithm. The results are summarized in the following table:

	Best	Average	Worst
Golden section algorithm	0.54 B_T	0.67 B_T	0.72 B_T
Equidistant algorithm	0.47 B_T	0.69 B_T	1 B_T

5. GENERALIZING THE ALGORITHM

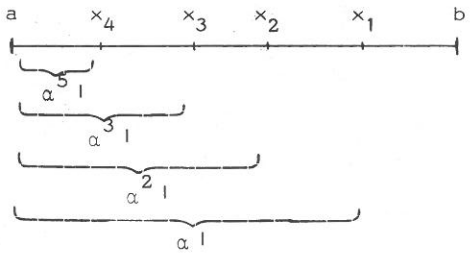
In this section we consider finding searching algorithms with more than two processes. The abstract data type interval can immediately be used by any number of processes, n. Since the golden section algorithm works so well for n = 2, it would be nice to find a generalization of this algorithm for n > 2. Let l denote the length of the interval. To generalize the algorithm an α ($0 < \alpha < 1$) must be found such that:

$$\alpha + \alpha^n = 1$$

$$\alpha^2 + \alpha^{n-1} = 1$$

.
.
.

If there are n such that this set of equations have a solution, we immediately have an elegant n process algorithm. For n = 4 the above set of equations does not have a solution, but we can choose α ($0 < \alpha < 1$) as the root of the polynomium $\alpha^2 + \alpha^3 = 1$. In this case $\alpha + \alpha^5 = 1$. Thus the following subdivision of the interval can be used:



$\alpha = .755$, $\alpha^2 = .570$, $\alpha^3 = .430$, ($\alpha^4 = .325$), $\alpha^5 = .245$

Let p_1 denote the process currently working on x_1, and let p_2 denote the process currently working on x_2 etc. When one of the processes respond, the processes are permuted. For example, when p_1 responds that the root is in $[a, x_1]$, p_2 becomes p_1, p_3 becomes p_2, and p_1 becomes p_3. This is written $\gamma = (312-)$. The following is a description of the algorithm using this notation.

Process Responding	Location of Root	New I	New Permutation
1	$r \in [x_1, b]$ →	$I = \alpha^5 I$,	$\gamma = (2 - - -)$
	$r \in [a, x_1]$ →	$I = \alpha I$,	$\gamma = (3\ 1\ 2\ -)$
2	$r \in [x_2, b]$ →	$I = \alpha^3 I$,	$\gamma = (2\ 1\ -\ -)$
	$r \in [a, x_2]$ →	$I = \alpha^2 I$,	$\gamma = (-\ 3\ 1\ 2)$
3	$r \in [x_3, b]$ →	$I = \alpha^2 I$,	$\gamma = (3\ 1\ 2\ -)$
	$r \in [a, x_3]$ →	$I = \alpha^3 I$,	$\gamma = (-\ -\ 1\ 2)$
4	$r \in [x_4, b]$ →	$I = \alpha I$,	$\gamma = (-\ 3\ 1\ 2)$
	$r \in [a, x_4]$ →	$I = \alpha^5 I$,	$\gamma = (-\ -\ -\ 2)$

The bars indicate which processes are no longer working on useful subdivision points, and whose work is therefore useless. (Few of the above given permutations are immediately obvious, they are based on properties of α such as: $\alpha^6 = \alpha - \alpha^2$, which are simple to show. Further justification is therefore not given here).

The details of the running time analysis are not given here, but the average running time is computed by the technique presented above:

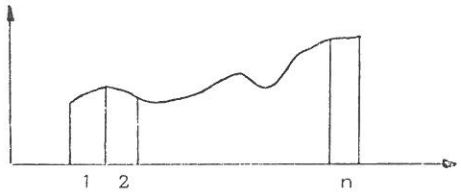
$$A(\Pi) \approx \frac{0.46\, B_T}{}$$

Kung [1976] suggests other generalizations of the algorithm to n processes.

CONCLUSION

In this paper three different concurrent algorithms have been described and analyzed. This analysis shows that using two concurrent processes gives a significant reduction of the running time, and that using four processes gives a further significant reduction.

The aim of this work is to find techniques for analyzing concurrent algorithms. The main problem is to handle the intricate interaction patterns that will often be caused by such algorithms. Some concurrent algorithms have little or no interaction. An example of this is doing numeric integration of some functions by n processes, where each process computes the integral over a small interval:



The analysis of such concurrent algorithms is of course trivial and consequently not very interesting. In algorithms requiring more synchronization the speed increases will, however, be smaller as the root searching algorithms show.

Acknowledgement

Torry Skak Gaarde made many valuable suggestions and corrections to the calculations presented in this paper, in particular he suggested the formula for calculating the average reduction of the interval length.

References

[Aho, Hopcroft, and Ullman 1974] The Design and Analysis of Computer Algorithms, A. V. Aho, J. E. Hopcroft and J. D. Ullman, Addison Wesley 1974.

[Feller 1950] An Introduction to Probability Theory and its Applications, W. Feller, John Wiley and Sons 1950.

[Knuth 1968] The Art of Computer Programming I - III, D. E. Knuth, Addison Wesley 1968.

[Kung 1976] Synchronized and Asynchronous Algorithms, H. T. Kung in Algorithms and Complexity, J. F. Traub (ed.) Academic Press 1976.

[Staunstrup 1978] Specification, Verification, and Implementation of Concurrent Programs, University of Southern California, Los Angeles 1978.