1998 International Computer Symposium
Workshop on Algorithms
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

A FASTER PARALLEL IMPLEMENTATION OF THE KANELLAKIS-SMOLKA ALGORITHM FOR BISIMILARITY CHECKING

Cheoljoo Jeong *Youngchan Kim* *Youngbae Oh*
Heungnam Kim

Laboratories for Computer and Software Technology
Electronics and Telecommunications Research Institute
Taejon, 305-350, Republic of Korea
Email:{cjeong,yckim,yboh,hnkim}@etri.re.kr

ABSTRACT

Bisimulation equivalence is one of the most important equivalence relations that capture the notion of 'behavioral equivalence' of concurrent systems. In this paper we present a randomized parallel implementation of the Kanellakis-Smolka algorithm that tests the bisimilarity of two finite CCS specifications in $O(n)$ time with $O(n^2)$ CRCW processors, when the semantic models are given as (S_1, A, T_1, s_1) and (S_2, A, T_2, s_2), respectively ($|S_1| + |S_2| = n, |T_1| + |T_2| = m$), and $|A| = 1$. Our implementation is faster than that of Lee and Rajasekaran [12] by a factor of $O(\lg n)$[1].

1. INTRODUCTION

Over the past decades much attention have been paid to the study of theories of concurrency such as CCS (Calculus of Communicating Systems) [9], CSP (Communicating Sequential Processes) [5], and π-calculus. And a fair portion of these research efforts were devoted to the study of the notion of 'equivalence' of systems. Since bisimulation equivalence relation was proposed as a notion of *behavioral equivalence* in [8], this equivalence relation has gained popularity not only in the software engineering community but in several seemingly unrelated areas: set theory, functional programming, game theory, computational complexity, and so on [1, 2, 10, 15].

Usually we use the bisimulation equivalence relation to test if two concurrent system specifications (e.g., in CCS) have the same behavior. But as the semantic models of process specifications encountered in industry are very huge in size, faster algorithms for determining the bisimulation equivalence of the models are more desirable.

When a finite CCS specification is given, its semantic model is usually taken to be a *labeled transition system* (LTS), where an LTS is defined to be a 4-tuple (S, A, T, s): S is the set of *states*, A is the set of *actions*, and $T \subseteq S \times A \times S$ is the *transition relation*, and $s \in S$ is the *start state*. When $(p, a, p') \in T$, we denote it by $p \xrightarrow{a} p'$.

The bisimilarity checking problem Given an LTS (S, A, T, s), a relation $R \subseteq S \times S$ is a strong bisimulation if, for any $a \in A$, $(p, q) \in R$ implies that

(1) if $p \xrightarrow{a} p'$, there exists $q' \in S$ such that $q \xrightarrow{a} q'$ and $(p', q') \in R$, and

(2) if $q \xrightarrow{a} q'$, there exists $p' \in S$ such that $p \xrightarrow{a} p'$ and $(p', q') \in R$.

When $(p, q) \in R$, we say that p and q are *bisimilar* and denote it by $p \sim q$. Given an LTS $(S, A, \rightarrow, s)$ and two states $p, q \in S$, the bisimilarity checking problem is to determine whether $p \sim q$ or not.

Kanellakis and Smolka discovered that the bisimilarity checking problem can be reduced to the *relational coarsest partition* problem[2] when the input LTS is finite [6] and the cardinality of the action set is 1.

The relational coarsest partition problem Given a finite set S, a binary relation T on S, and a partition $\Pi_0 = \{B_1, \cdots, B_p\}$ on S, the relational coarsest partition problem finding the partition $\Pi = \{C_1, \cdots, C_q\}$ such that

(1) for every $C_j \in \Pi$, there exits $B_i \in \Pi_0$ such that $C_j \subseteq B_i$,

(2) for every $p, q \in C_i$ and $C_j (\neq C_i)$, $T(p) \cap C_j = \emptyset$ if and only if $T(q) \cap C_j \neq \emptyset$, and[3]

(3) Π is the smallest partition that satisfies the above two conditions.

In [6], Kanellakis and Smolka showed that, when a relational coarsest partition $\Pi = \{C_i\}$ is found, each block C_i is a bisimulation equivalence class of the input LTS (of the original bisimilarity checking problem), and presented an algorithm that solves this problem in $O(mn)$ time where $m = |T|$ and $n = |S|$[4]. Later, Fernandez [4] showed that Kanellakis-Smolka algorithm can be easily extended to

[1] $\lg n$ means the logarithm of n to the base 2.

[2] In essence, the two problems are equivalent problems in different settings when the size of the action set is 1.

[3] $T(p) = \{q : (p, q) \in T\}$.

[4] From now on we will denote the cardinality of the transition relation and the cardinality of the state set by m and n, respectively.

solve the the bisimilarity checking problem with multiple actions with no extra cost. In 1987, Paige and Tarjan [11] devised a faster algorithm for the problem which requires $O(m \lg n)$ time.

There are not so many research results on parallel algorithms for bisimilarity checking algorithms. In a work of Zhang and Smolka [16], an attempt has been made to parallelize the Kanellakis-Smolka algorithm but the algorithm was rather experimental and did not entail complexity analysis of the algorithm. Rajasekaran and Lee [12] has devised two CREW PRAM algorithms: one based on the Kanellakis-Smolka algorithm and the other based on the Paige-Tarjan algorithm. The former requires $O(n^{1+\epsilon})$ time and $O(m/n^{\epsilon})$ processors for any $\epsilon > 1$ and the latter requires $O(n \lg n)$ time and $O((m \lg n)/n)$ processors. Both algorithms assume that $|A| = 1$ but they can be extended to handle the case when $|A| > 1$ without changing the runtime complexities. In [3], the bisimilarity checking problem was proved to be P-complete, which implies that the problem does not seem to have any poly-logarithmic parallel algorithm.

Our algorithm solves the relational coarsest partition problem in $O(n)$ time using $O(n^2)$ processors on the CRCW PRAM, when $|A| = 1$. Our algorithm can be extended to handle the case when $|A| > 1$ with the same time and processor complexity using the techniques in [4, 12]. But our algorithm is a Monte Carlo algorithm with one-sided error: there is a non-zero probability that it errs when it outputs 'no.' Currently, we have not yet found a precise bound on the probability of the error.

The organization of this paper is as follows. In section 2, we present the model of computation and some primitive operations used throughout the algorithm. In section 3, we present the basic idea of this paper and a parallel algorithm for the relational coarsest partition problem and in section 4 we present the complexity analysis of our algorithm. Finally, in section 5 we give the conclusion and some further research directions.

2. PRELIMINARIES

2.1. Models of computation

Our algorithm assumes the PRAM (Parallel Random Access Machine) as its model of computation. The PRAM is a model for parallel computation with multiple synchronous processors and a single shared memory [7]. The PRAM models allow three different methods for resolution of memory access contention: CRCW (Concurrent Read, Concurrent Write), CREW (Concurrent Read, Exclusive Write), and EREW (Exclusive Read, Exclusive Write) PRAM.

Performance of parallel algorithms is measured by two functions of input size n: the time complexity $T(n)$ and the processor complexity $P(n)$. A parallel algorithm is said to be *efficient* if its time complexity is poly-logarithmic, i.e., $T(n) = O(\lg^k n)$ for some constant $k > 0$. Let $T^*(n)$ be the complexity of the optimal sequential algorithm[5] for some problem P and let the *work*, $W(n)$ of a parallel algorithm be defined to be the product of time and processors needed, i.e., $W(n) = T(n)P(n)$. Then a parallel algorithm is *optimal* when $W(n) = O(T^*(n))$.

2.2. The Kanellakis-Smolka algorithm for bisimilarity checking

Kanellakis and Smolka's algorithm [6] is based on the following observation:

> Let B be a set of states that are believed to be in the same bisimulation equivalence class. Given a set of states, *Splitter*, we can further partition B into two blocks B_1 and B_2 where B_1 is the set of states that have transitions into some states in *Splitter* and B_2 is the set of states that does not have any (see Figure 1).

The algorithm KS-BISIMILARITY in Figure 2 takes one LTS (S, A, T, s_0) as its input and finds the bisimulation equivalence classes with respect to the transition relation T. We can easily modify the algorithm to take two LTSs, (S_0, A, T_1, s_0) and (S_1, A, T_2, s_1), as its input and decide whether the LTSs are bisimilar. The following algorithm KS-EXTENDED-BISIMILARITY is a brief sketch of the modified algorithm.

```
KS-MODIFIED-BISIMILARITY(S_0, S_1, T_1, T_2, s_0, s_1)
1  S ← (S_0 ∪ S_1);
2  T ← (T_0 ∪ T_1);
3  KS-BISIMILARITY(S, T);
4  if s_0 and s_1 are members of the same block
5     then return True;
6     else return False;
```

The algorithm KS-BISIMILARITY starts with a single bisimulation equivalence relation, which is the state set S itself. It repeats the splitting procedure until no more splitting is possible.

3. A PARALLEL ALGORITHM BASED ON MULTIWAY SPLITTING

In the algorithm KS-BISIMILARITY (Figure 2), the **while** loop (from line 3 to line 16) is iterated at most $O(n)$ times, since at each iteration the number of blocks increases by at least one.

In this paper we show how we can reduce several iterations into one using *multiway splitting* technique, which enables us to split a block, $B \in \Pi$ with respect to multiple splitters.

3.1. Basic Idea: Multiway splitting

Suppose that there are a block B and two splitters, Sp_1 and Sp_2, as in Figure 3. Elements of B can be classified into $2^2 = 4$ categories: those with arcs to both Sp_1 and Sp_2

[5] or best known sequential algorithm

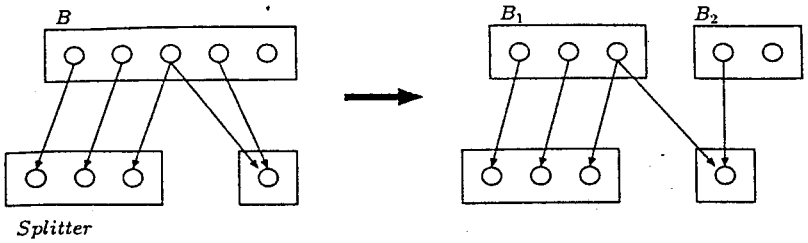

Figure 1: Splitting a block with respect to a splitter

```
KS-BISIMILARITY(S, T)
 1  Π ← {S};
 2  SplitterSet ← Π;
 3  while SplitterSet ≠ ∅
 4  do assign an element of SplitterSet to CurrentSplitter;
 5     SplitterSet ← SplitterSet − {CurrentSplitter};
 6     compute T^{-1}(CurrentSplitter);
 7     for each block B ∈ Π
 8     do if T^{-1}(CurrentSplitter) ∩ B ≠ ∅ and
 9           T^{-1}(CurrentSplitter) ∩ B ≠ B
10           then B' ← T^{-1}(CurrentSplitter) ∩ B;
11                B'' ← B − B';
12                Π ← Π − {B} ∪ {B', B''};
13                if B ∈ SplitterSet
14                  then SplitterSet
15                       ← SplitterSet − {B} ∪ {B', B''};
16                  else SplitterSet
17                       ← SplitterSet ∪ {B', B''}
18           else do nothing;
```

Figure 2: The Kanellakis–Smolka algorithm for bisimilarity checking

(B_2), those with arcs only to Sp_1 (B_1), those with arcs only to Sp_2 (B_3), and those with arcs to neither Sp_1 nor Sp_2 (B_4). When we split a block with respect to k splitters, the block is split into at most $\min\{2^k, n\}$ new blocks.

3.2. Data structures

3.2.1. *Representation of an LTS*

Given an LTS (S, A, T, s_0) where $|A| = 1$, we can regard it as a directed graph $G = (V, E)$, where $V = S$ and $\langle P, Q\rangle \in E$ if and only if $(P, a, Q) \in T$ for $a \in A$. We can assume that $V = \{1, 2, \cdots, n = |S|\}$ without loss of generality.

G is represented by an indexed set of adjacency lists $\Gamma = \{\Gamma_i : 1 \le i \le n\}$. Each adjacency list Γ_v of $v \in V$ is implemented as a linear array of length n. In addition, for each vertex $v \in V$, we maintain an inverse adjacency list Γ_v^{-1}, which contains the vertices from which v is adjacent. Γ and Γ^{-1} satisfies the following property.

$$\Gamma_u(v) = \Gamma_v^{-1}(u) = \begin{cases} 1 & \text{if } \langle u, v\rangle \in E, \\ 0 & \text{if } \langle u, v\rangle \notin E. \end{cases}$$

Note that we can test if $\langle u, v\rangle \in E$ or not in $O(1)$ time using this representation and Γ is not changed throughout the execution of our algorithm.

3.2.2. *Representation of a partition* Π

A partition $\Pi = \{B_i\}$ of a state set $S = \{1, 2, \cdots, n\}$ is represented by a linear list of ordered tuples $\{\pi_i = \langle s_i, b_i, t_i\rangle : 1 \le i \le n\}$. $\pi_i = \langle s_i, b_i, t_i\rangle$ means that for the ith entry of Π contains the information for state s_i and s_i has block number b_i. The usage of t_i will be explained later. Two states s_i and s_j are in the same block if and only if $b_i = b_j$.

For example, let a state set $S = \{1, 2, 3, 4, 5, 6\}$ and a partition $\Pi = \{\{1, 2, 6\}, \{3, 4, 5\}\}$ is given. Then Π can be represented by $[\langle 1, 1, t_1\rangle, \langle 2, 1, t_2\rangle, \langle 3, 2, t_3\rangle, \langle 4, 2, t_4\rangle, \langle 5, 2, t_5\rangle, \langle 6, 1, t_6\rangle]$.

3.2.3. *Representation of a splitter set*

A splitter set, $SplitterSet$, is a set of splitters, where each splitter is a subset of S. As we can see in Figure 2, it is always the case the $SplitterSet \subseteq \Pi$. So, if we have a way to mark the blocks of Π which happen to be elements of $SplitterSet$, we don't have to maintain additional heavy data structures for the splitter set.

For this purpose we add one more bit to each entry of Π, that is, $\pi_i \in \Pi$ is now represented by $\langle s_i, b_i, t_i, p_i\rangle$ where p_i tells whether b_i is a splitter or not (e.g. $p_i = 1$ if and only if b_i is a splitter). Though there may be some redundancy this representation scheme serves us well.

3.3. Implementation of multiway splitting

Now let's investigate the details of our algorithm, PARALLELBISIMILARITY, given in Figure 4.

```
PARALLELBISIMILARITY(S, T)
1  Π ← {S};
2  // In general, Π = {S_1, S_2, ···, S_|Π|}.
3  while SplitterSet ≠ ∅
4  do assign k elements of SplitterSet to CurrentSplitters;
5     // Let CurrentSplitters = {Sp_1, Sp_2, ···, Sp_k}
6     // and Sp_i = {s_i1, s_i2, ···, s_i|Sp_i|}.
7     SplitterSet ← SplitterSet − CurrentSplitters;
8     MULTIWAYSPLIT(Π, CurrentSplitters);
9     CLEANUPPARTITION(Π);
```

Figure 4: A parallel algorithm for bisimilarity checking

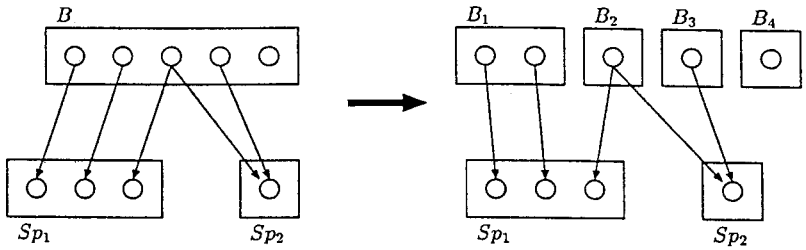

Figure 3: Splitting a block with respect to multiple splitters

3.3.1. *Marking for splitting*

Suppose that we want to use multiway splitting technique with degree k. Then we assume that each state s is associated with a linear array $m^s = [m_1^s, \cdots, m_k^s]$ of size k. m_i^t is used to denote the fact that state t has a transition to ith splitter. The algorithm for splitting a block with respect to k splitters, $Sp = \{Sp_1, Sp_2, \cdots, Sp_k\}$ is as follows:

MULTIWAYSPLIT(S, Sp)
1 // Let $Sp = \{Sp_1, \cdots, Sp_k\}$
2 // and $Sp_i = \{s_{i1}, \cdots, s_{i|Sp_i|}\}$.
3 **for** each state $s \in S$ **in parallel**
4 **do for** each $1 \le i \le k$ **in parallel**
5 **do** $m_i^s \leftarrow 0$;
6 **for** each splitter $Sp_i \in Sp$ **in parallel**
7 **do for** each element $s_{ij} \in Sp_i$ **in parallel**
8 **do for** each element $t \in T^{-1}(s_{ij})$ **in parallel**
9 **do** $m_i^t \leftarrow 1$;

MULTIWAYSPLIT procedure can be executed in $O(1)$ time using $O(n^2)$ CRCW processors. This procedure

3.3.2. *Cleaning up the partition* Π

At the end of each iteration for refining the partition Π, we need to clean up the representation for Π so as to guarantee the integrity of Π. That is, at the beginning of next iteration, it should be that two states s_i and s_j are in the same block of Π if and only if $b_i = b_j$.

As we can see in step 9 of MULTIWAYSPLIT procedure, we modify $m^s = [m_1^s, \cdots, m_k^s]$ for each state $s \in S$ at each iteration. We should assign new block numbers to states in accordance with the new $\{m^s : s \in S\}$.

CLEANUPPARTITION(Π)
1 // Let $\Pi = [\pi_1, \cdots, \pi_j = \langle s_i, b_i, t_i, p_i \rangle, \cdots, \pi_n]$.
2 // Let $M = \{m^{s_i} = [m_1^{s_i}, \cdots, m_k^{s_i}] : s_i \in S\}$.
3 **for** each $m^{s_i} \in M$ **in parallel**
4 **do for** each $1 \le j \le k$ **in parallel**
5 **do** $m_j^{s_i} \leftarrow m_j^{s_i} \times 2^j$;
6 $t_i \leftarrow \sum_{j=1}^{k} m_j^{s_i}$;
7 sort Π lexicographically on b_i and t_i;
8 // b_i': the new block number of s_i.
9 compute $\{b_i'\}$ such that $b_i' = b_j'$ iff $b_i = b_j$ and $t_i = t_j$;
10 **for** each $1 \le j \le n$ **in parallel**
11 **do** $b_j \leftarrow b_j'$;

After this cleanup procedure, each state s_i has its block number b_i. Step 7 of procedure CLEANUPPARTITION can be executed using any stable, optimal parallel sort algorithm and step 9 can be implemented using list ranking operations and parallel prefix sums operations [13] using $O(\lg n)$ time and $O(n)$ processors (in Figure 5 we can see a result of step 7 computation.). After all, CLEANUPPARTITION procedure requires $O(\lg n)$ time and $O(nk)$ processors.

3.3.3. *Computing the new splitter set*

Now let's consider *SplitterSet*. If a splitter $Sp \in$ *SplitterSet* has been split into Sp_1 and Sp_2 in procedure MULTIWAYSPLIT, then we need to add Sp_1 and Sp_2 to *SplitterSet* (step 13 through step 17 of Figure 2). As we noted in section 3.2.3., this job amounts to adjusting the values of $\{p_i\}$.

The values of $\{p_i\}$ can be decided executing step 9 of CLEANUPPARTITION. Suppose that after the execution of step 7 we got an instance of Π as in Figure 5(a). Let $p_i = p_2 = p_6 =$ **True**, which means that the first block of Π is a splitter. It's obvious that the splitter should be split into multiple splitters if and only if there exists an element of this block such that $b_i \neq t_i$. After all, the new splitter set can be computed in the CLEANUPPARTITION procedure with no additional cost.

Comments on the algorithm We should consider the case when the number of blocks in Π is less than k. At the first few iterations this really is the case. To overcome this problem, we arbitrarily partition Π so that the number of blocks will be greater than $k-1$. One possible partition may be obtained by isolating $k-1$ vertices from a block of size $\ge k$. That is, given a block $B = \{s_1, \cdots, s_k, s_{k+1}, \cdots, s_l\}$, split this into k blocks $\{\{s_1\}, \cdots, \{s_{k-1}\}, \{s_k, \cdots, s_l\}\}$. This job does not incur additional cost but this makes our algorithm a Monte Carlo algorithm with one-sided error.

Since we may arbitrarily split 'bisimilar states' into different bisimulation equivalences, the equivalence class found may be finer in granularity than the desired one. So our algorithm has a non-zero probability of error when it outputs 'no.' Currently, we have not yet found a precise bound on the probability of the error.

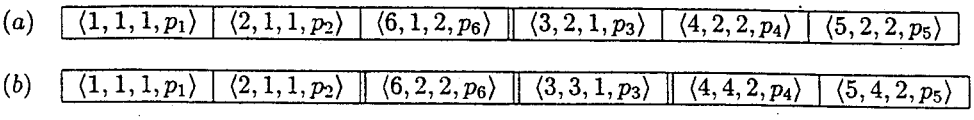

Figure 5: Computing new block number of the states

4. COMPLEXITY ANALYSIS OF THE ALGORITHM

In this section we claim that when we accelerate each iteration by splitting the blocks of Π with respect to k splitters we can reduce the number of iterations within a factor of k though we may not reduce the time complexity to be polylogarithmic.

Lemma 1 Suppose that we use multiway splitting of degree k. When there are at least k splitters in $SplitterSet$ at each iteration, the **while** loop of PARALLELBISIMILARITY is iterated at most $O(n/k)$ times.

Proof Initially, $SplitterSet$ consists of S alone and the exit condition of the **while** loop is "$SplitterSet \neq \emptyset$." Each splitter in $CurrentSplitters$ either take part in splitting the blocks of Π or does no contribution in splitting (let's call this splitters *idle*). In the latter case, the splitter (together with the elements of splitter) is thrown away and not used any more.

Let r_i be the number of splitters in $SplitterSet$ just after the ith iteration and $r_0 = 1$. Note that $0 \leq r_i \leq n$. Let $X_i = \bigcup_{P \in SplitterSet} P$ be the set of states that are elements of some splitter of $SplitterSet$ after the ith iteration. Then, if $i < j$, $X_i \supseteq X_j$.

Let's consider two extreme cases: when the splitters are always idle, and when the splitters are always non-idle. In the first case, $X_i = X_{i-1} - k$ and the number of iterations will be $O(n/k)$. In the second case, $r_i = r_{i-1}/2$ and in $O(\lg n)$ iterations, the $SplitterSet$ will contains n splitters. When there are n splitters in $SplitterSet$, in the next iteration all the splitters must be idle and thrown away.

So, the worst case is when the splitters are always idle (though this does not seem to occur in the real computation), the number of iterations of the **while** loop is $O(n/k)$. **Q.E.D.**

Lemma 2 When we use multiway splitting of degree k, each iteration of the **while** loop of PARALLELBISIMLARITY takes at most $O(\lg n)$ time.

Proof The most subtle part of the algorithm is step 3 to step 5 of CLEANUPPARTITION. When $k = n$ then we should calculate 2^n and this number is too big for us to accept that this computation takes constant time. But when $k = \lg n$, the largest value produced in step 5 is $2^{\lg n} = O(n)$ and we can assume that the computation takes constant time. After all, taking into the time and processor complexities of the procedures (of section 3.3) used in PARALLELBISIMILARITY, an execution of the **while** loop takes $O(\lg n)$ time with $O(n^2)$ CRCW processors. **Q.E.D.**

Now, the following theorem can be derived from Lemma 1 and Lemma 2.

Theorem 1 The algorithm PARALLELBISIMILARITY runs in $O(\lg n)$ time using $O(n^2)$ processors on the CRCW PRAM.

5. CONCLUSION

In this paper we presented a parallel implementation of the Kanellakis-Smolka algorithm for bisimilarity checking. Our algorithm works with $O(n^2)$ processors and $O(n)$ time in the CRCW PRAM, which is faster than Lee and Rajasekaran's algorithm by a factor of $O(\lg n)$. But our algorithm has a non-zero probability of error when it outputs 'no.' At present, we have not yet found a precise bound on the probability of the error and the research is going on to prove that the upper bound of the error is small enough.

There are several research directions on bisimilarity checking algorithms.

- Lee and Rajasekaran [12] posed the open problem of finding $O(n^\epsilon)$-time algorithm for the bisimilarity checking problem for any $\epsilon > 1$.
- Since the bisimilarity checking problem is $\mathcal{P}$-complete, it's not likely that there exists a polylogarithmic algorithm for the problem. So, it'll be better to look for a randomized parallel algorithm for a problem and to prove that it is in RNC.

6. REFERENCES

[1] S. Abramsky and L. Ong. Full abstraction in the lazy lambda calculus. *Information and Computation*, 105:159–267, 1993.

[2] P. Aczel. *Non-Well-Founded Sets*. Number 14 in CSLI Lecture Notes. Stanford: Center for the Study of Languages and Information, 1988.

[3] J. L. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is p-complete. *Formal Aspects of Computing*, 4(6A):638–648, 1992.

[4] Jean-Claude Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science*

of Computer Programming, 13(2–3):219–236, May 1990.

[5] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, Hertfordshire, 1985.

[6] P. C. Kanellakis and S. A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, pages 228–240, 1983.

[7] R. M. Karp and V. Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, pages 869–932. MIT Press, Cambridge, MA, 1990.

[8] R. Milner. *A Calculus of Communicating Systems*. Number 92 in Lecture Notes in Computer Science. Springer-Verlag, New York, NY, 1980.

[9] R. Milner. *Communication and Concurrency*. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[10] F. Moller and S. A. Smolka. On the complexity of bisimulation. *ACM Computing Surveys*, 25(2):287–289, June 1995.

[11] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, December 1987.

[12] S. Rajasekaran and I. Lee. Parallel algorithms for relational coarsest partition problem. Technical Report MS-CIS-93-71, Computer and Information Science Department, Univeristy of Pennsylvania, Philadelphia, PA, July 1993. appeared in CAV'94.

[13] J. H. Reif, editor. *Synthesis of Parallel Algorithms*. Morgan Kaufmann, San Mateo, CA, 1992.

[14] S. A. Smolka, O. Sokolsky, and S. Zhang. On the parallel complexity of bisimulation and model checking. In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra: A Bisimulation Perspective*, number 53 in CSLI Lecture Notes, chapter 13, pages 257–287. CSLI Publications, 1995.

[15] C. Stirling. Local model checking games. In *Proceedings of CONCUR'95*, number 962 in Lecture Notes in Computer Science, pages 1–11. Springer-Verlag, 1995.

[16] S. Zhang and S. A. Smolka. Towards efficient parallelization of equivalence checking algorithms. In *Proceedings of FORTE '92*, pages 133–146, October 1992.