1991

H.P. Korver

The current state of bisimulation tools

Computer Science/Department of Software Technology Report CS-R9108 February

The Current State of Bisimulation Tools *

Henri Korver
Department of software technology, CWI
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands
e-mail: henri@cwi.nl

Abstract

It is a well-known fact that descriptions of concurrent systems often appear to be incorrect, in the sense that the behavior of the system does not correspond to the behavior the specifier had in mind. To overcome this problem a lot of research has been going on in the field of formal verification methods of software descriptions.

At the moment *bisimulation* is one of the most popular criteria for guaranteeing correctness of concurrent system descriptions. In this approach, concurrent systems are modeled as transition graphs, and verification amounts to establishing that the graph representing the implementation of the system is bisimilar to the graph representing the specification of the system. The main advantage of this approach is that bisimulation can be decided efficiently and fully automatically on finite transition graphs.

This paper contains a comparative evaluation of bisimulation techniques, as found in the following tools.

- ACP Bisimulation Tool
- Branching Bisimulation Tool
- Aldébaran
- The Concurrency Workbench (CWB)
- Tool for Automatic Verification (TAV)
- Winston
- Auto, Autograph and Ecrins

Further, the usability of these tools for PSF [MV89,MV90] is investigated. PSF is here proposed as a suitable representative of a wide range of formalisms that have come up for specifying concurrent systems.

Key Words & Phrases: process formalism, transition graph, bisimulation, tools.
1985 Mathematics Subject Classification: 68Q60.
1987 CR Categories: D.2.4, F.1.1.

*The research of the author was supported by RACE project no. 1046, Specification and Programming Environment for Communication Software (SPECS).

Report CS-R9108
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Contents

1 Preface **4**

2 Introduction **4**

3 Automatic Verification Methods **5**
- 3.1 Algebraic Approach 5
- 3.2 Logical Approach 5
- 3.3 Other Approaches 5

4 Bisimulation Equivalence as a Platform **6**
- 4.1 Transition Graphs and Bisimulation 6
- 4.2 Computing Bisimulation Equivalence 7
  - 4.2.1 The Minimal Approach 7
  - 4.2.2 The Maximal Approach 7
- 4.3 Minimizing Transition Graphs 9

5 Behavioral Equivalences as a Verification Method **9**
- 5.1 Trace Equivalence 9
- 5.2 Observational Equivalence 9
- 5.3 Branching Bisimulation Equivalence 10
- 5.4 Remaining Equivalences 11

6 Explaining Non-bisimilarity **11**
- 6.1 Distinguishing Formulas 11
- 6.2 The Game 12

7 Generating Transition Graphs **13**
- 7.1 Infinite Graphs 13
- 7.2 State Explosion 14

8 Tool Study **15**
- 8.1 Tool Investigation 15
  - 8.1.1 ACP-tool 15
  - 8.1.2 Branching-tool 15
  - 8.1.3 Aldébaran 15
  - 8.1.4 CWB 16
  - 8.1.5 TAV 16
  - 8.1.6 Winston 16
  - 8.1.7 INRIA-tools 17
- 8.2 Performance Comparison 18
- 8.3 Evaluation 19
- 8.4 Future Developments 20

9 Integration of Tools **22**

10 PSF and Bisimulation Tools **22**

11 Conclusion **23**

References **24**

1 Preface

Verification of the behavior of concurrent systems (satellites, computer networks etc.) is considered to be an important basis to guarantee reliable software to control these systems. The behavior of a (concurrent) system is often called a process.

One of the most popular verification methods is checking the external behavior of processes by means of a behavioral equivalence, usually a bisimulation equivalence. In this approach, processes are represented as finite state systems and these equivalences can be computed fully automatically.

In this setting we study how this kind of verification can be applied to processes that are described in PSF, a Process Specification Formalism [MV89,MV90]. PSF is a formalism which allows the specification of the control and the data aspects of processes. In fact, PSF is a combination of ACP [BW] which is a theory to specify the control aspects of processes and ASF [BHK] which is a formalism to specify the datatypes that are used by the processes. PSF is strongly related to other process specification formalisms like LOTOS [ISO] and CRL [SPECS].

To support verification of processes specified in the PSF-formalism, a bisimulation tool working on PSF (or more likely a derived representation of PSF) has been planned to be incorporated.

The idea is to use existing bisimulation tools to do the verification on PSF. The first step in this direction is a study of the most interesting bisimulation tools that are already built.

This document is in the first place an investigation of the current state of bisimulation tools. The usefulness of these tools for the PSF-environment is investigated. In particular, the question how to interface these tools to PSF is discussed.

The major conclusion of this document is that at the moment there exists no single bisimulation tool which directly satisfies all the needs for the PSF-environment, because most bisimulation tools are highly complementary. Under pressure of the CONCUR-project [Ba90b], tool developers are working on a common format to which all the tools may be interfaced. Therefore it is suggested to define a translator which transforms PSF to such a common format.

2 Introduction

Verification methods for concurrent systems can be classified in at least three families: theorem proving, model-checking, and bisimulation methods. The first family has theoretically the most power; it may be applied to many undecidable problems and in some sense it can deal with infinite objects. However, theorem proving methods have usually a high complexity and there is little hope to make them purely automatic. The model-checking and bisimulation methods are based on finite state systems, and are fully automatic.

This document focuses on bisimulation methods, however when other verification methods could or should interact with bisimulation methods, then these methods will be taken into consideration also, for example model checking and abstraction criteria.

Sections 4, 5, 6 and 7 treat the underlying theory of bisimulation techniques. Section 8 is actually the study of all important bisimulation tools that are available at the moment. Section 9 deals with the question how to interface bisimulation tools to each other. Section

10 discusses the usefulness of bisimulation tools for the PSF-environment. The next section is a global overview of the most important finite state based verification methods, and places the notion of bisimulation techniques in a (proper) context.

3 Automatic Verification Methods

In general, all fully automatic verification techniques on concurrent systems, are based on finite systems. This means that processes must be represented by a finite system. The most used finite representations of processes are *transition graphs* [Mil], *Kripke structures* [BCG] and *acceptance trees* [Hen]. *Petri nets* [Rei] are not directly suitable for bisimulation techniques and out of scope here. Below the most important finite state based verification approaches are sketched.

3.1 Algebraic Approach

In this approach, one tries to decide a specification-implementation equivalence between processes. To this end the specification-process should describe the external behavior of the implementation-process. And this can be checked automatically, in case these processes are finite state. There exists a whole spectrum of relevant specification-implementation equivalences. Roughly, all these equivalences are stronger than trace equivalence and weaker than (strong) bisimulation equivalence. Bisimulation equivalence plays here a central role, because usually the problem of deciding these equivalences between two finite transition systems can be reduced to it. As we will see later on.

3.2 Logical Approach

In the logical approach, properties of processes expressed in temporal logics are checked. In this approach, processes are mostly represented as Kripke structures, because traditionally temporal logics are defined on Kripke structures. The difference between a transition graph and a Kripke structure is that a transition graph has labeled transitions and a Kripke structure has labeled states. Recently some temporal logics have been adapted and defined on transition systems which are a more common model of concurrent systems [JKP,NV].

Generally, temporal logics are very expressive to define properties of processes, however the price is that logical formulas are sometimes difficult to understand.

3.3 Other Approaches

Abstraction Actions Abstract actions are sets of sequences of concrete actions. These abstract actions can be used to perform reductions on transition graphs [RSa,RSb]. The reduction technique permits observation of partial properties of a given transition graph and gives several observation angles on local and global properties of the transition graph. These reductions can be applied in combination with equivalence testing techniques [Ver].

Equation Solving Equation solving is a kind of refinement strategy: if a specification B and a part of an implementation A of a process are known, solving such an equation amounts to a construction of the missing parts [Par].

4 Bisimulation Equivalence as a Platform

A popular technique for verifying finite-state systems involves the use of a *behavioral equivalence.* In this approach, specifications and implementations are formalized as finite-state machines, and verification amounts to establishing that an implementation is equivalent to (in the sense of *behaving the same as*) its specification. A number of equivalences have been proposed in the literature [BIM, BHR, NH, Hen, GW, LK, Mil] and several automated tools include facilities for computing them [BSV, CPSa, CPSb, GV, Zui, MSGS], as we shall see later on.

Bisimulation equivalence here plays a central role [Mil] because a variety of other equivalences may be described in terms of it and secondly because the problem of deciding these equivalences can be reduced to the problem of deciding bisimulation equivalence. In the following subsections the definition of bisimulation and the algorithm to compute it are described.

4.1 Transition Graphs and Bisimulation

Strong bisimulation or, shortly, bisimulation is defined on transition graphs. Vertices in these graphs correspond to the states a system may reach during execution, with one vertex being distinguished as the start state. The edges, which are directed, are labeled with the actions and represent the state transitions a system may undergo. The formal definition is the following.

Definition 4.1 *A* transition graph *is a quadruple* $\langle Q, q, Act, \rightarrow \rangle$, *where:*

- *Q is a set of states (vertices);*
- *$q \in Q$ is the start state;*
- *Act is a set of actions; and*
- *$\rightarrow \subseteq Q \times Act \times Q$ is the derivation relation (set of labeled edges).*

We shall often write $q_1 \xrightarrow{a} q_2$ to indicate that there is an edge labeled a from state q_1 to state q_2; in this case, we shall sometimes say that q_2 is an *a-derivative* of q_1. When a graph does not have a start state indicated, we shall refer to the corresponding triple as a *transition system.* A state in a transition system gives rise to a transition graph in the obvious way: let the given state be the start state, with the other three components of the transition graph coming from the transition system.

Reactive systems [Pnu] compute by interacting with their environment. For such systems, the traditional language equivalence of transition graphs theory is insufficiently discriminating, since the resolution of nondeterministic choices may leave a system in states that react differently to stimuli offered by the environment. Bisimulation equivalence remedies this shortcoming by requiring that equivalent systems have state sets that "match up" appropriately: the start states must be matched, and if two states are matched then they must have matching a-derivatives for any action a either one is capable of. These intuitions may be formalized in terms of *bisimulations* on a *single* transition system.

Definition 4.2 *Let $\langle Q, Act, \rightarrow \rangle$ be a transition system. Then a relation $R \subseteq Q \times Q$ is a* bisimulation *if R is symmetric and whenever $q_1 R q_2$, the following hold.*

- *If $q_1 \xrightarrow{a} q_1'$ then there is a q_2' such that $q_2 \xrightarrow{a} q_2'$ and $q_1' R q_2'$.*

Two states in a transition system are *bisimulation equivalent* if there is a bisimulation relating them. When q_1 and q_2 are bisimulation equivalent we shall write $q_1 \sim q_2$.

Let $G_1 = \langle Q_1, q_1, Act, \rightarrow_1 \rangle$ and $G_2 = \langle Q_2, q_2, Act, \rightarrow_2 \rangle$ be two transition graphs satisfying $Q_1 \cap Q_2 = \emptyset$. Then G_1 and G_2 are *bisimulation equivalent* exactly when the two start states, q_1 and q_2, are bisimulation equivalent in the transition system $\langle Q_1 \cup Q_2, Act, \rightarrow_1 \cup \rightarrow_2 \rangle$. This definition may be generalized to arbitrary transition systems (i.e., ones whoses state sets are not disjoint), at the cost of a slightly more complicated definition for the transition system in which bisimulation equivalence is to be computed.

4.2 Computing Bisimulation Equivalence

There exist two approaches to decide whether or not two transition graphs are bisimilar.

4.2.1 The Minimal Approach

Consider two transition graphs identified by the two start states s_1 and s_2. In [Lar] an algorithm is described which closely follows the definition of bisimulation. This algorithm acts like a theorem prover and tries to construct a bisimulation relation containing the pair (s_1,s_2). If this pair is included then the graphs identified by the start states s_1 and s_2 are bisimilar. Otherwise the algorithm terminates and fails. Although the idea of the algorithm is intuitively appealing, the complexity is in fact exponential.

4.2.2 The Maximal Approach

The most popular technique to check for bisimulation equivalence is based on partition-refinement algorithms. To decide whether two graphs T_1 and T_2 are bisimilar, one first takes the union of these graphs (call this union T). Then a partition refinement algorithm is applied on this graph T. When the two start states of T_1 and T_2 appear in the same equivalence class of the final partition of T, then one can conclude that the graphs T_1 and T_2 are bisimilar. Below we treat first the Kanellakis-Smolka algorithm and then the more advanced approach of Paige and Tarjan.

Kanellakis-Smolka

The Kanellakis-Smolka algorithm [KS] exploits the fact that an equivalence relation on a set of states may be viewed as a *partition*, or set of pairwise-disjoint subsets (called *blocks*) of the state set, whose union is the state set. In this representation, blocks correspond to

```
function $split(B, a, B') =$
  $\{\{ s \in B \mid \exists s' \in B'.\, s \xrightarrow{a} s' \}, \{ s \in B \mid \neg\exists s' \in B'.\, s \xrightarrow{a} s' \}\} - \{\emptyset\}$;

algorithm $bisim(Q, Act, \rightarrow)$;
  begin
    $P_1 := \{Q\}$;
    $P_2 := \emptyset$;
    while $find(P, a, B')$ do begin
      $P_2 := P_1$;
      $P_1 := \emptyset$;
      foreach $B \in P_2$ do $P_1 := P_1 \cup split(B, a, B')$;
    end
  end
```

Figure 1: The partition refinement algorithm for bisimulation equivalence.

the equivalence classes—so two states are equivalent exactly when they belong to the same block. Beginning with the partition containing one block (representing the trivial equivalence relation consisting of one equivalence class), the algorithm repeatedly *refines* this partition (by splitting blocks) until the associated equivalence relation becomes a bisimulation. In order to determine whether the partition needs further refining, the algorithm looks at each block in turn. If a state in a block B has an a-derivative in a block B' and another state in B does not, then the algorithm splits B into two blocks, one containing the states having an a-derivative in B' and the other containing the states that do not. We call block B' together with action a a splitter-pair. When no more splitting is possible, the resulting equivalence relation corresponds exactly to bisimulation equivalence on the given transition system. The algorithm is given in Figure 1. The procedure *find* searches for a splitter-pair in the current partition P. If the pair is found the function assigns the values of the pair to a and B', and returns true; otherwise it returns false. The function `split` is used to split one block with respect to another; notice that `split`$(B, a, B') = \{B\}$ (i.e. B is not split with respect to a and B') if either all the states in B, or none of them, have an a-derivative in B'. The algorithm terminates when no more splitting is possible. The worst-case complexity of *bisim* is $O(|\rightarrow| * |Q|)$.

Paige and Tarjan

Paige and Tarjan [PT] developed a more sophisticated partition-refinement algorithm whose complexity is $O(|\rightarrow| * \log |Q|)$ for a transition system $\langle Q, Act, \rightarrow\rangle$. The essence of their approach is a more elaborate splitting procedure that, in certain instances, enables blocks to be split with respect to two blocks by examining only one of the two blocks. Their algorithm also uses the inverse of the $\rightarrow$ relation. It is beyond the scope of this document to consider this approach in full. The Paige and Tarjan approach becomes practical in the case of large graphs; for graphs of normal size, this algorithm has too much overhead.

4.3 Minimizing Transition Graphs

Partition algorithms (as Kanellakis-Smolka) are often used as a preprocessing step to compute minimal transition graphs. A minimal transition graph is bisimilar to the original graph and minimal in states and edges.

Once the bisimulation equivalence is computed (by the Kanellakis-Smolka algorithm) on the states of the transition graph T, the following simple procedure reduces T to a minimal graph T':

1. Each block of the final partition becomes a state in T'.
2. Let s_{B_1} be a state in block B_1; if $s_{B_1} \xrightarrow{a} s_{B_2}$ is a transition in T and s_{B_2} is an element of block B_2, then $B_1 \xrightarrow{a} B_2$ becomes a transition in T'.

Reducing the size of transition graphs by such a minimizing procedure may be beneficial in many areas: model checking [CLM], graph generation [Fer,MV] etc.

5 Behavioral Equivalences as a Verification Method

A number of behavioral equivalences may be characterized in terms of bisimulation equivalence on suitably transformed transition systems [CH, CPSa, CPSb]. These equivalences range from (strong) bisimulation to trace equivalence. In the following subsections we treat trace equivalence, and the two most popular equivalences used in bisimulation tools namely observational equivalence and branching bisimulation equivalence. The last subsection groups the remaining equivalences.

5.1 Trace Equivalence

Two states in a transition graph are *trace equivalent* when they are capable of engaging in the same sequences of actions. The problem of deciding trace equivalence can be reduced to the problem of bisimulation equivalence. Trace equivalence on states in a transition graph corresponds to bisimulation equivalence on a *determinized* version of the transition graph. A determinized transition graph is a transition graph where every state has at most one a-derivative for a given a. The space complexity of determinizing a graph can be at worst exponential (PSPACE-complete) in the states of the graph, however in practice the situations in which space is exponential are claimed to be very rare [CH].

5.2 Observational Equivalence

Observational equivalence is one of the most used behavioral equivalences for verifying concurrent systems. This has two reasons: first, observational equivalence was defined in order to verify the external (observable) behavior of concurrent systems. This is in contrast with strong bisimulation which is in fact only used as a platform for other more practical equivalences such as observational equivalence. Secondly, observational equivalence can be decided efficiently, compared to other equivalences like trace and failure equivalences etc. Below, we give the definition of observational equivalence (also called weak bisimulation) on a transition graph where a special unobservable action called τ is present.

Definition 5.1
Let $< Q, Act, \rightarrow >$ *be a transition system.*

- $\Longrightarrow$ *is the transitive and reflexive closure of* $\xrightarrow{\tau}$.
  $\overset{a}{\Longrightarrow}$ *is the composition of* $\Longrightarrow$ *and* $\xrightarrow{a}$ *and* $\Longrightarrow$.

- *A relation* $R \subseteq Q \times Q$ *is a* weak bisimulation *if* R *is symmetric and, whenever* $q_1 \; R \; q_2$, *the following hold.*

  - *If* $q_1 \overset{a}{\Longrightarrow} q_1'$ *then there is a* q_2' *such that* $q_2 \overset{a}{\Longrightarrow} q_2'$ *and* $q_1' \; R \; q_2'$.

The double arrow relation says that an observable a-step may be preceded and followed by unobservable τ-steps.

To compute observational equivalence the original transition graphs are transformed to *observation graphs*. These observation graphs are computed usually in three steps [BS]:

1. Add τ-loops to all the states (reflexive closure).
2. Add a τ-edge connecting the first and the last state of a sequence of τ-edges in the graph (transitive closure).
3. Add a τ-edge connecting the first and the last state of a sequence of τ-edges in which one a-edge occurs (transitive closure).

This τ-saturation procedure takes an amount of time that is polynomial in the size of the graph. The complexity of the transformation is determined by the complexity of computing the transitive closures above. Naive transitive closure algorithms work in $O(|Q|^3)$ [BS] and in the case of transition graphs with a large number of states sub-cubic algorithms tend to become beneficial and work in $O(|Q|^{2.376})$ [CW] time. Surprisingly, in most cases the subroutine for transitive closure accounts for more than 80 per cent of the execution time when determining observational equivalence [EF].

To compute observational equivalence, a partition algorithm for strong bisimulation (i.e. Kanellakis-Smolka) is applied to these observation graphs.

5.3 Branching Bisimulation Equivalence

Branching bisimulation [GW] is a behavioral equivalence on processes discovered recently, and resembles observational equivalence. However branching bisimulation equivalence is finer than observational equivalence, but in practice this refinement does not influence generality; in other words: all known practical protocols that have been verified by observational equivalence can also be verified by branching bisimulation equivalence. Branching bisimulation preserves the branching structure better than observational equivalence because the intermediate states are related as well: see the definition below. In [GW,GV] it is advocated that this more natural definition of branching structure has some interesting advantages.

Definition 5.2 *(Branching bisimulation)*

- *Let* $<Q, Act, \rightarrow>$ *be an transition graph. A relation* $R \subseteq Q \times Q$ *is called a branching bisimulation if it is symmetric and satisfies the following transfer property:*

  If rRs *and* $r \xrightarrow{a} r'$, *then either* $a{=}\tau$ *and* $r'Rs$; *or* $\exists s_0, .., s_n, s'$:
  $s = s_0, [\forall_{0<i\leq n} : s_{i-1} \xrightarrow{\tau} s_i]$ *and* $s_n \xrightarrow{a} s'$ *such that* $\forall_{1\leq i\leq n}\ rRs_i$ *and* $r'Rs'$.

- *Two states* r *and* s *are branching bisimilar, abbreviated* $r \approx_B s$ *or* $s \approx_B r$, *if there exists a branching bisimulation relating* r *and* s.

Branching bisimulation ($\approx_B$) is an equivalence relation.

In [GV] an efficient algorithm computing branching bisimulation is described. This algorithm uses a special refinement strategy which avoids the τ-saturation procedure which is costly both in time and space. Performance tests show increased efficiency in practice. The algorithm works in $O(|Q| * | \rightarrow |)$.

5.4 Remaining Equivalences

The remaining equivalences that reside between bisimulation and trace equivalence are grouped here. The reason for this is roughly speaking that they are less popular in practice. We mention acceptance, may, test, failure equivalences. These equivalences are usually defined on acceptance trees [Hen]: transition trees where nodes may be labeled with extra information, and are in the setting of tools not so popular compared with observational and branching equivalence.

A generalization of behavioral equivalence is the behavioral preorder; in this setting, an implementation satisfies a specification if the implementation is "greater or equal than" (intuitively: "behaves at least as well as") the specification. One interesting preorder is the *prebisimulation preorder.* However, the method to decide this preorder is significantly less efficient than observational equivalence and branching bisimulation.

6 Explaining Non-bisimilarity

When a verification fails it would be pleasant to get diagnostic information back about why it failed. For example a user would like to know why two states are not equivalent. The following subsections give the current state of the art.

6.1 Distinguishing Formulas

Bisimulation equivalence also has a *logical* characterization in terms of Hennessy-Milner Logic (HML) [HM]: two states are equivalent exactly when they satisfy the same HML-formulas. The syntax of HML is defined as follows, where $a \in Act$.

$$\Phi ::= tt \mid \neg\Phi \mid \Phi \wedge \Phi \mid \langle a \rangle \Phi$$

Given a transition system $T = \langle Q, Act, \rightarrow \rangle$, the interpretation of the logic maps each formula to the set of states for which the formula is "true"; Figure 2 gives the formal definition. In

$$\begin{array}{rcl} [\![tt]\!]_T &=& Q \\ [\![\neg\Phi]\!]_T &=& Q - [\![\Phi]\!]_T \\ [\![\Phi_1 \wedge \Phi_2]\!]_T &=& [\![\Phi_1]\!]_T \cap [\![\Phi_2]\!]_T \\ [\![\langle a\rangle\Phi]\!]_T &=& \{\, q \in Q \mid \exists q'.\, q \xrightarrow{a} q' \wedge q' \in [\![\Phi]\!]_T \,\} \end{array}$$

Figure 2: The semantics of formulas in Hennessy-Milner Logic.

the remainder of the paper we shall omit explicit reference to the transition system used to interpret formulas when it is clear from the context. Intuitively, the formula *tt* holds of any state, and $\neg\Phi$ holds of a state if Φ does not. $\Phi_1 \wedge \Phi_2$ holds of a state if both Φ_1 and Φ_2 do, while the modal proposition $\langle a\rangle\Phi$ holds if the state has an a-derivative for which Φ holds. We shall say that a state q in transition system T *satisfies* formula Φ if $q \in [\![\Phi]\!]_T$. The fact that bisimulation equivalence is induced by HML suggests a useful diagnostic methodology for tools that compute bisimulation equivalence: when two systems are found not to be equivalent, one may explain why by giving a formula satisfied by one and not the other. There are two methods to compute such a distinguishing formula.

Hillerström Hillerström [Hil] describes a technique for computing a Hennessy-Milner formula, which follows very closely the definition of bisimulation. His method relies on the use of a backtracking algorithm that is in fact exponential.

Cleaveland Cleaveland [Cle] describes a technique for computing a Hennessy-Milner formula that works in conjunction with a partition-refinement algorithm for computing bisimulation equivalence. This algorithm runs in polynomial time and is therefore far more advanced than the method of Hillerström above.

The formulas that are computed by the methods of Hillerström and Cleaveland often contain redundant information. Computing distinguishing formulas that do not contain redundant information, appears to be complex [Cle].

As a number of other behavioral equivalences may be characterized by bisimulation equivalence, the methods of Hillerström and Cleaveland can be used for the equivalences also.

6.2 The Game

To explain to a user why two states are not equivalent in a setting of bisimulation equivalences, it is also possible to play the following game against the system (computer program).

Consider two non-bisimilar graphs T_1 and T_2 : The system starts by selecting one of the two start states of T_1 and T_2, first. It chooses a transition to a state s_1, namely the transition which it knows that the other start state cannot match. Then the user may try to find a transition matching the transition of the system, say the transition leading to state s_2. In case the user cannot perform the given transition, obviously he must acknowledge that the processes are not bisimilar. Otherwise consider the states s_1 and s_2 as the new start states and repeat this game recursively. Finally, after sufficiently many repetitions of the game the user will have to admit that the graphs were not bisimilar. Note that the system may switch from one to the other start state in the repetition of the game.

In [Ogu] it is proved formally that when the system has a winning strategy, the graphs are not bisimulation equivalent. Conversely: when both graphs are bisimulation equivalent the system will not have a winning strategy.

The notion of a transition can be defined in different ways to apply the game to different notions of behavioral equivalences.

7 Generating Transition Graphs

Concurrent systems are usually specified in formalisms based on the well-accepted generic notion of process algebras: CCS [Mil], ACP [BW], CSP [BHR], MEIJE [Bou] etc. If one wants to apply bisimulation techniques to verify systems described in such a formalism, one has to convert such a specification to a finite transition graph. Transition graphs are generated by means of *structural conditional rules* that give such formalisms an operational semantics in the style of Plotkin.

Note however that these graphs may not be finite state, and infinite graphs are useless for bisimulation methods. Another problem is that during the generation of the graph the state space may become unacceptably large. These issues are the ingredients for the following subsections.

7.1 Infinite Graphs

Process algebra formalisms are usually based on process-operators, an action-structure and a recursive definition mechanism. Processes are regular when they have a finite state representation. Processes constructed by prefix sequential composition, nondeterministic choice and a standard recursion definition are always regular. This fact is well-known from automata theory and proved for processes as constructed above in [JP]. However nearly any extension on this concept (operators as parallel composition, hiding, restriction), makes the regularity problem undecidable. In [BBK], the regularity of some classes of processes is investigated. Process terms parametrized with data are mostly not regular.

Finiteness Conditions

As stated before, conditional rules define in a structural manner the semantics function that computes a transition system from a (closed) term of a process algebra. This definition is constructive: you can compute each transition of the transition system by building proof trees in which nodes are instances of the rules. When two nodes in this tree are syntactically equal then these nodes are contracted and the tree becomes a transition graph.

A term of a process algebra can be represented by a finite state system iff the transition system computed from the term using the operational rules is strongly bisimilar to some finite transition system (i.e. with a finite number of states and a finite number of transitions). In many process algebras this property is undecidable. Sufficient syntactic conditions to ensure that the transition system is finite have to be given to the conditional rules defining the operational semantics of the process formalism in question. In [MV] some interesting syntactic conditions are given, we mention here *guardedness*, *well-formed recursion* and *non-growing operators*. These conditions prevent in many cases from infinite graph generation.

Processes and Data

When the actions of process formalisms are parametrized by data-values ranging over infinite domains, there is little hope to obtain an operational semantics by means of finite state systems. However, there is a small class of parametrized processes which are data-independent [JP]; these processes may only write and read data without computing functions on the data. Such processes may be regular, if of course the control structure of the process is regular. Data-independent processes can be represented by symbolic data-values as actions, covering the infinite domains. The problem of determining these symbolic data-values is NP-complete; it seems to be very hard to find out whether or not a symbolic data-value already has been used in a process.

7.2 State Explosion

Process formalisms that describe concurrent systems always contain of a parallel operator to compose concurrent components of the system. To apply the verification techniques described in this document one has to convert the system description into a finite state graph. The parallel composition of two components causes a transition graph that has as state space the product of the state space of the components. This means that the number of states in the transition graph may be exponential in the number of parallel components. This situation where the state space grows exponentially occurs frequently in practical applications and is called *state explosion*. One method to cope with the state explosion problem is compositional minimization of finite state systems, which is motivated by the following observation of Graf and Steffen [GS].

> The state space of the complete system usually contains a large number of semantically equivalent states, because implementations often have numerous internal computations that are irrelevant for its external behavior and therefore are "hidden". Then it is usually possible to reduce the state space of a system drastically by collapsing equivalent states to a single state without effecting its observable properties. For example, a complex communication system can often be reduced to a simple buffer.

Unfortunately, the straightforward idea to just successively combine and minimize the components of the system is not satisfactory, because "local" minimization does not consider context constraints such as restriction and hiding of actions, and therefore may even lead to subsystems with a larger state space than the overall (global) system.

To overcome the problem of context constraints, it is proposed in [GS] to use *interface processes* supplied by the user to guide the reduction strategy. In fact this approach has a lot in common with the more natural notion of algebraic manipulation on process-terms. Complex process-terms can be manipulated by algebraic laws (axioms) to a configuration where subterms have become manageable for bisimulation tools and can be reduced compositionally [Ba90a,Vaa].

In this algebraic approach it is sometimes possible to rewrite process-expressions to a configuration which is optimal for successive reduction, for example configurations where restriction and hiding operators are driven as deep as possible into the process-term [SV].

8 Tool Study

This section gives an overview of the tools that have been experimented with. The tools are only considered on aspects that are related to the issue of verification by bisimulation methods. So simulation, model-checking and user interfaces are here not important; if there is no connection to bisimulation methods. The following table shows the tools that we consider.

Tool	Site	Implementation	References
ACP-tool	PTT-RNL	Pascal	[Zui]
Branching-tool	CWI	Pascal	[GV]
Aldébaran	IMAG	C	[Fer]
CWB	Univ. Edinburgh	ML	[CPSa,CPSb,CPSc,EF]
TAV	Univ. Aalborg	C-Prolog	[GLZ]
Winston	SUNY Stony Brook	C	[MSGS]
Auto, Autograph, Ecrins	INRIA	LeLisp	[RSa,RSb,SV,MSV]

8.1 Tool Investigation

8.1.1 ACP-tool

The ACP-tool has the concrete syntax of ACP as input. A front-end generates transition graphs by operational rules in the style of Plotkin that characterize τ-bisimulation (observational equivalence) semantics. This means that the semantics of these graphs are pinned to the semantics of τ-bisimulation and cannot be used for other equivalences.

The most important functionality of the ACP-tool is deciding τ-bisimulation between ACP-processes. The ACP-tool is the only bisimulation tool that works on ACP syntax. Unfortunately the ACP-tool does not support a minimization function to minimize the processes. However a minimization function may be added by a 'simple' extension of the ACP-tool.

8.1.2 Branching-tool

The Branching-tool has plain transition graphs as input. This tool computes branching bisimulation by the partition algorithm described in [GV]. In a postprocessing step, transition graphs can be minimized with respect to branching bisimulation equivalence. Practical tests show that the Branching-tool is the most efficient tool available at the moment [EFJ].

8.1.3 Aldébaran

The input of the tool of Aldébaran consists of networks of transition systems. Networks are constructed by the parallel composition of transition graphs. This parallel composition is implemented by a binary parallel operator, a restriction operator and a hiding operator. The semantics of this parallel operator is parametrized. The user is (in a restricted sense) allowed to define his own semantics for parallel composition and communication (mostly asynchronous vs synchronous).

Aldébaran computes 4 equivalences by the Paige-Tarjan algorithm: strong bisimulation equivalence, observational equivalence, acceptance model equivalence and safety equivalence. For each equivalence a minimize function is available. The user can decide whether or not transition graphs have to be reduced before being composed. There is an option to successively compose and reduce the graphs during graph generation. As pointed out in subsection 7.2, successive reduction is not always the right approach and therefore Aldébaran supports a reduction strategy that reduces first the expressions that have the lowest number of visible actions after parallel composition. The trick of this strategy is that the number of visible actions after parallel composition is computed without really constructing the graph. This is done by computing inherited and synthesized attributes to collect information about visibility of actions throughout the whole process expression.

8.1.4 CWB

The Concurrency Workbench (CWB) is under development at University of Edinburgh, University of Sussex and the Swedish Institute of Computer Science (SICS). The CWB uses the syntax of CCS for the input of processes. This tool checks whether or not two processes are semantically equivalent by standard techniques [KS,PT]. The most used equivalences are bisimulations but also a whole variety of other equivalences is supported like may-, must-, testing- and 2/3-equivalences.

A unique feature of the CWB is the support of preorders. For each equivalence supported by the CWB there is also a preorder version available. Preorders are in fact generalizations of equivalence relations. When two processes are completely specified, a preorder will be an instance of its associated equivalence. When a process is not specified completely, and contains holes that have to be filled in later, a preorder is not an equivalence anymore and is then considered as a specification-implementation relation.

In the approach of preorders relations on incomplete process specifications the CWB also supports equation solving which is in fact a semi-automatical technique to fill in these open holes to obtain an implementation of a specified process. The method works by successively transforming equations into simpler equations, in parallel with the generation of a solution [Par]. These transformations can be performed automatically by the system according to certain heuristics, the user can apply these heuristics interactively.

8.1.5 TAV

The TAV-system (Tool for Automatic Verification) is also based on the framework of CCS. It supports deciding strong and weak bisimilarity (observational equivalence)between CCS-processes in a different way than other tools. The algorithms to decide the equivalences are a direct implementation of the minimal approach (as described in section 4.2.1) in Prolog [Hil]. A drawback is that a minimization function is not derivable in the minimal approach.

What mostly distinguishes TAV from the other systems is that TAV offers explanations for the answers, i.e. it generates a distinguishing formula when two states are not equivalent. These formulas are generated by the method of Hillerström. Experiments showed that distinguishing formulas are usable for debugging purposes.

8.1.6 Winston

The input of Winston consists of graphical networks. In the section about Autograph graphical networks will be discussed in some detail. Winston computes strong bisimulation and observational equivalence by the Kanellakis-Smolka algorithm. Recently, some optimizations were implemented to speed up the τ-saturation phase when computing observational equivalence. These optimizations are based on the removal of τ-edges in the graph without changing the semantics of observational equivalence. Reducing the number of τ-edges in the graph will speed up τ-saturation. The results are very promising. It has been shown in practice that τ-removal techniques work well on graphs that have a specific structure (for instance deterministic transitions). However, it seems that finding patterns in the graph to which τ-removal can be applied is in general very complex. This has to be tested further.

8.1.7 INRIA-tools

Ecrins

Ecrins [MSV,MV] is in the first place a generic transition graph generator, as a preprocessor for Auto. Ecrins supports a format in which the concrete syntax and abstract syntax definitions of a process formalism can be defined, together with structural conditional rules that gives them an operational semantics. The Ecrins *calculus compiler* uses the first part to produce a scanner, a parser and abstract syntax structures for expressions of the calculus. The format is general enough to handle most usual process formalisms in the literature. From the semantics part, the compiler will produce the functions for building and combining transition graphs from process terms. The conditional rules which define the semantics are checked against different syntactic conditions in order to prevent infinite graph generation. The functions for building and combining transition graphs from terms are parametrized by equivalences that may reduce the graphs before composition. These equivalences are extracted from the Auto-system to reduce graphs before further composition.

Secondly, Ecrins has the ability to check for the validity of bisimulation laws in process algebras, by specialized bisimulation algorithms in combination with theorem proving methods. To be more explicit about this, Ecrins supports the notion of FH-bisimulation, which is a generalization of strong bisimulation in the sense that processes may also contain free variables. This allows Ecins to check bisimulation laws fully automatically. For instance, the soundness of the bisimulation law $X + X = X$ can be checked by deciding FH-bisimulation: $X + X \sim_{FH} X$.

Auto

The Auto-system [SV, RSb] is transition graph based and uses Ecrins to produce these transition graphs. This implies that Auto can be used for any process formalism that can be specified in the format of Ecrins. Auto supports checking of equivalences (bisimulation, observational and trace equivalence) in the standard way.

Besides this, Auto performs reductions on transition graphs with respect to abstraction criteria. Abstraction criteria are abstract actions which symbolize a set of sequences of real actions that are abstracted from. In Auto these sequences of actions can be defined by means of regular expressions extended with some predicates on the action structure. To illustrate the notion of abstract actions: consider observational equivalence. In this setting an a-step is an abstract action because in fact it is a sequence of τ-steps followed by an a-step followed by a sequence of τ-steps. Some experiments show that the concrete syntax of these abstraction criteria is not powerful enough at the moment to express abstraction criteria that characterize for example observational equivalence semantics, in a straightforward way. On the other hand, abstraction criteria are well suited for debugging purposes [Ver].

Autograph

Autograph [RSa,RSb] is a mouse-driven multi-window environment to support a two-way interface for Auto. It can be used as an interactive input device for Auto, and as an output device.

- Graphical Input: Terms of process calculi can be represented graphically. Briefly: Boxes next to one another represent parallel subterms, containment of boxes represents the subterm relation, edges without outside ports generate action restrictions and edges between ports with different names generate relabelings. The graphical presentation has an even more simpler interpretation.

  In addition, the semantics may be altered, using a system of semantical annotations by functions from a special menu in the Autograph system. Justification for these annotations is drawn from the Ecrins system, where a syntactic formalism is defined creating new calculi through new operators. This part of Autograph is still under development and is to be described in full in the final version of [RSb].

- Graphical output: When Auto performs analysis on its input it produces informative output like transformed transition graphs, subtransition graphs, paths, states, actions etc. In Autograph, these objects can be passed back graphically, to regain the geometrical structure, this is convenient for debugging purposes.

8.2 Performance Comparison

The most costly operations in the context of bisimulation techniques are graph-generation and graph-transformation (for instance τ-saturation). In comparison with these operations the partitioning of the graph to compute the equivalent states, can be neglected. Not only the time performance of these operations is bad, but also the space performance. Bad space performance influences the time performance negatively in this case. It would be interesting to test whether reduction at generation time influences also (besides space performance) the time performance when the protocols become larger. This has not been done as of now. In [EF,EFJ,GV] there are some comparisons of time and space performance of the tools considered, here are the results:

1. Branching-tool: Best results, the reason for this is the absence of τ-saturation.

2. Aldébaran: Good results probably due to efficient C implementation and an efficient partitioning algorithm.

3. Auto: Acceptable but less performance than the previous tools probably due to the LISP-implementation (garbage collections ...).

4. ACP-tool: Bad performance when the protocols become larger, due to inefficient space handling.

5. Winston: Recently the algorithm for observational equivalence has been optimized in this tool. The results are very good when graphs have a high rate of deterministic transitions.

6. CWB: Poor performance compared with 1, 2, 3. The ML-implementation is probably the main reason.

TAV and Ecrins use theorem proving methods to decide their equivalences, and are therefore inefficient compared to the standard (partition-based) techniques.

Tools like the Branching-tool, Aldébaran and Auto have the best space performance (can handle the largest transition graphs). Auto and Aldébaran reduce their graphs at generation time, the Branching-tool does not. But the Branching-tool has no τ-saturation which saves a lot of space when the transition graph contains a lot of τ's.

Space performance seems to be more important than time performance: computing for a long time to verify a protocol is acceptable, because in most cases as a protocol has to be verified only once. But running out of space is unacceptable in any case.

8.3 Evaluation

The summary below shows the state of the tools at the time this document is being written. For future developments, see next section.

ACP-tool:

- inefficient in time and space
- lacks of a minimization function

Branching-tool:

- supports two functions: minimize function and a partitioning function for branching bisimulation
- most efficient bisimulation tool at the moment.

TAV:

- inefficient in time and space
- supports diagnostic answers, namely: distinguishing formulas.
- lacks of a minimization function

Ecrins:

- a generic graph-generator. The question how much efficiency must be paid for the generality of the graph generator is open.

- Soundness checking of bisimulation laws by FH-bisimulation: a prototype at the moment.

CWB:

- inefficient
- very versatile, supports a lot of equivalences, but most of these are not often used because of inefficiency.
- the only tool that supports preorders and equation solving.
- at the moment the CWB does not work directly on transition graphs which makes the interface to other tools difficult.

Winston:

- good performance when transition graphs contain a lot of deterministic transitions.

Auto:

- acceptable efficient
- supports abstraction criteria, but this is not worked out sufficiently.
- reduces transition graphs at generation time

Autograph:

- a two-way graphical interface, offers possibilities to make Auto and other systems user-friendly

Aldébaran:

- reduces transition graphs at generation time
- acceptable efficient

8.4 Future Developments

The state of bisimulation tools is very young at the moment and the success of these tools will depend on the future developments of these tools.

TAV There are two future developments:

- Support of more equivalences.
- A C-implementation of TAV, making TAV more efficient.

Branching-tool Description of an efficient algorithm to produce a distinguishing property when two states are not branching bisimilar [Kor].

ACP-tool The development has stopped.

CWB Work will continue on the development of the CWB, implementing more algorithms for analysis, building a graphics interface, and applying it to more examples.

- Examples of proof will be examined in which data theories and infinite state spaces play an important part. On the basis of this work the next stage in the CWB will be designed and implemented.
- Algorithms for the analysis of concurrent processes will be extended. A prime example is to extend the algorithm for finding bisimulations in finite state spaces to processes with a restricted form of value passing where the values range over infinite domains. A prototype has already been established [Lee].
- Implementation of the algorithm of Cleaveland [Cle] to produce distinguishing formulas to explain why a bisimulation failed.

INRIA-tools The development of INRIA tools will continue. This means an extension of new theoretical and pragmatic features, enforcing interfaces between the tools and also with the CWB, and thereby avoiding unnecessary duplication of software.

- The Ecrins system will be extended to deal with equivalences based on general abstraction mechanisms. This includes the extension of the abstract action reduction technique of Auto to the general case of possibly infinite transition systems defined by structural rules.
- Extension of the finite transition graph approach. Auto should take its input not only from Meije but from arbitrary process calculi, provided that appropriate syntactical restrictions are given to restrict terms to generate finite transition graphs. This should of course be done by the Ecrins system. Some progress in this direction has already been made, see previous sections.
- Enhancements of Autograph. A general graphical interface to deal uniformly with all Ecrins-definable calculi, including graphical mechanisms to draw user-defined operators with arguments. Also development of graphical display and debugging tools for Auto are planned.

Aldébaran There are some directions for future work.

- The definition of equivalence relations compatible with a set of properties (temporal logic formulae).
- Finding strategies to control state explosion occurring in the parallel composition.

Winston

- Optimizing the algorithm computing observational equivalence.
- Extension of the current graphical interface.

9 Integration of Tools

The ultimate goal is to achieve a common format to which all existing tools can be interfaced. This means that a concurrent system specified in a process formalism can be translated to this common tool language and all available tools can be used to verify this system. At the moment a concrete unified representation of transition graphs is proposed as a common format for tools based on finite representations [RSb]. Tools like Aldébaran, Branching-tool and Auto are already interfaced to each other.

However, having transition graphs as a common format may be a severe problem: transition graphs may become tremendously large, when they are generated from a process expression with parallel components (state explosion). At the moment, there is little hope to obtain a more advanced common format that represents processes more concisely, such as networks of graphs where parallel composition is not expanded.

10 PSF and Bisimulation Tools

In the PSF-environment it is planned to implement a translator that translates a subset of PSF to transition graphs. Because transition graphs are the common format for bisimulation tools, in principle all bisimulation tools respecting this common format can be used.

But the transition graphs generated from PSF may be tremendously large. To cope with this it is needed to reduce intermediate graphs during the construction of the global graph using an equivalence. Note that this equivalence must be a congruence for the composition operation of the reduced components. To meet this claim, the translation from PSF to transition graphs should be parametrized by all favorite equivalences to reduce intermediate graphs at generation time. This parametrization is not an easy task because every equivalence induces its own optimal reduction strategy or optimal term configuration.

Below the usefulness of the studied bisimulation tools for the PSF-environment is investigated.

ACP-tool Although the ACP-tool works directly on ACP which is a part of the definition of PSF, this tool is not suitable for the PSF-environment. The reason for this is the fact that the tool is not efficient; the tool runs out of space even with simple protocols.

Branching-tool This tool supports just one equivalence, namely branching bisimulation. Branching bisimulation is claimed to be one of the most interesting behavioral equivalences on processes. At the moment the branching-tool is the most efficient tool available [EFJ,GV]. This tool is already interfaced to other tools like Aldébaran and Auto, moreover the tool is interfaced to the finite state graphs in the PSF-environment.

CWB The CWB is not efficient but very versatile. This tool is the only tool that checks preorders on processes. The concept of preorder relations on processes is in fact a generalization of the concept of bisimulation relations on processes in the sense of allowing open holes in process specifications. At the moment the preorder-checking algorithms are not efficient. And besides, the language definition of PSF is not directly suitable for preorder-semantics.

Aldébaran The most interesting feature of the Aldébaran tool is that it supports a strategy to reduce graphs before they are composed. The question whether or not this strategy is optimal and constitutes a serious answer to state explosion is open.

Winston In the Winston-tool some effort is put into optimization of the algorithm that computes observational equivalence. The results are very promising, and we hope that these optimizations are a structural approach towards more efficiency.

TAV TAV is inefficient in computing equivalences because the methods are based on theorem proving in a Prolog environment. An interesting feature of TAV is that it is the only bisimulation tool known that explains why a bisimulation fails, by means of a distinguishing formula. However the method that TAV uses to compute a distinguishing formula, is backtracking based and is in fact exponential in time and space. Currently a more advanced algorithm [Cle] working in polynomial time is being implemented in the CWB. In this case TAV is not useful anymore for the PSF-environment.

Ecrins It would be a nice exercise to see what subset of PSF fits into the format of Ecrins. One problem is that the communication mechanism of PSF does not fit into the format, but this can be remedied by a technical extension of the format. In case this would have been realised, Ecrins generates the graphs from PSF automatically, and in turn these graphs can be analyzed by the Auto-system. In this approach the graphs can be reduced by the equivalences supported by Auto before composing. The main question is to what extent the generality of the Ecrins system influences the efficiency of the graph generation procedure.

Auto The Auto-system computes all standard equivalences and has acceptable performance. Auto also performs verification by reducing transition graphs with respect to abstract actions. However the subject of abstract actions is not worked out enough to be relevant for the PSF-environment at the moment.

Autograph As an output device, the animation tool builders in the PSF-environment can learn how the output of bisimulation tools can be animated.

11 Conclusion

Bisimulation techniques are one of the few straightforward methods to check the correctness of concurrent systems fully automatically. However, bisimulation techniques can only be applied to concurrent systems that have a regular control structure; for example Telecom-systems do not always have such a structure and therefore bisimulation methods are not complete enough to do verification for all real life applications.

As bisimulation is based on a finite state representation of processes, it suffers from the state explosion problem. At the moment, there are no methods that are powerful enough to overcome the state explosion problem. For the time being, the best thing to do is reducing the size of generated graphs as early as possible. Therefore a translator is suggested to generate graphs (from PSF) that can be reduced compositionally at generation time by an equivalence. It is obvious that this reduction will be performed by bisimulation tools.

Because the state of the art of bisimulation tools is highly complementary at the moment, we hope that PSF can be translated to a common format of transition graphs to which all available tools will be interfaced in the near future. In this approach all current technology can be imported into the PSF-environment.

References

[Ba90a] J.C.M. Baeten, editor: *Applications of process algebra.* Cambridge University Press, 1990.

[Ba90b] J.C.M. Baeten: Description of CONCUR. In *Bulletin of the EATCS*, n. 40, p. 418, February 1990.

[Bou] G. Boudol: Notes on algebraic calculi of processes. In K. Apt, editor, *Logics and Models of Concurrent Systems*, pages 261–303. Springer-Verlag, 1985. NATO ASI Series F13.

[BBK] J.C.M Baeten, J.A Bergstra and J.W. Klop: Decidability of bisimulation equivalence for processes generating context-free languages. In *Proceedings PARLE conference*, Eindhoven, Vol. 2 (Parallel languages), LNCS 259, Springer-Verlag, pages 94-113, 1987.

[BCG] M.C. Browne, E.M. Clarke, and O. Grümberg: Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1,2):115–131, 1988.

[BHK] J.A. Bergstra, J. Heering and P. Klint: ASF – an algebraic specification formalism. *Report CS-R8705*, CWI, Amsterdam, 1987.

[BHR] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe: A Theory of Communicating Sequential Processes. In *Journal of the ACM*, v. 31, n. 3, pages 560-599, July 1984.

[BRSV] G. Boudol, V. Roy, R. De Simone and D. Vergamini: Process Calculi, From Theory To Practice: Verification Tools. *INRIA Report 1098*, Octobre 1989.

[BS] T. Bolognesi and S. A. Smolka: Fundamental Results for the Verification of Observational Equivalence: a Survey. In *Proceedings of the IFIP 7th International Symposium on Protocol Specification, Testing, and Verification*, North-Holland, 1987.

[BSV] G. Boudol, V. Roy, R. de Simone and D. Vergamini: Process Algebras and Systems of Communicating Processes. In *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, Lecture Notes in Computer Science 407, pages 1-10, Springer-Verlag, Berlin, 1990.

[BW] J.C.M. Baeten and W.P. Weijland: *Process algebra.* Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[Cle] R. Cleaveland: On Automatically Distinguishing Inequivalent Processes. In *Proceedings: Workshop on Computer-Aided Verification (R. Kurshan and E.M. Clarke, editors)*, DIMACS technical report 90-31, Vol. 2, New Yersey, 1990. To appear in Lecture Notes in Computer Science.

[CH] R. Cleaveland and M. Hennessy: Testing Equivalence as a Bisimulation Equivalence. In *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, Lecture Notes in Computer Science 407, pages 11-23, Springer-Verlag, Berlin, 1990.

[CLM] E.M. Clarke, D.E. Long and K.L. McMillian: Compositional model checking. In *Proceedings 4th Annual Symposium on Logic in Computer Science (LICS)*, Asilomar, California, IEEE Computer Society Press, Washington, pages 354-362, 1989.

[CPSa] R. Cleaveland, J. Parrow and B. Steffen: A Semantics-Based Tool for the Verification of Finite-State Systems. In *Proceedings of the Ninth IFIP Symposium on Protocol Specification, Testing and Verification*, pages 287-302. North-Holland, Amsterdam, 1990.

[CPSb] R. Cleaveland, J. Parrow and B. Steffen: The Concurrency Workbench. In *Proceedings of the Workshop on Automatic Verification Methods for Finite-State Systems*, Lecture Notes in Computer Science 407, pages 24-37, Springer-Verlag, Berlin, 1989.

[CPSc] R. Cleaveland, J.G. Parrow and B. Steffen: The Concurrency Workbench: Operating Instructions. *Technical Note 10*, University of Edinburgh, Laboratory for Foundation of Computer Science, September 1988.

[EF] P. Ernberg, L. Fredlund: Identifying some bottlenecks of the Concurrency Workbench, 10 April 1990. *Not published yet.*

[CW] D. Coppersmith and S. Winograd: Matrix multiplication via arithmetic progressions. In *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing,* New York City, pages 1–6, 1987.

[EFJ] P. Ernberg, L. Fredlund and B. Jonsson: Specification and Validation of a Simple Overtaking Protocol using LOTOS. *SICS technical report*, T90006, ISSN 1100-3154, 1990.

[Fer] J. Cl. Fernandez: Aldébaran, A tool for verification of communicating processes. *User Manual.*

[GLZ] J. Godskesen, K. Larsen and M. Zeeberg: TAV User Manual. *Technical report R 89-19*, 1989.

[GS] S. Graf and B. Steffen: Compositional Minimization of Finite State Processes. In *Proceedings: 1990 Workshop on Computer-Aided Verification (R. Kurshan and E.M. Clarke, editors)*, DIMACS technical report 90-31, Vol. 1, New Yersey, 1990. To appear in Lecture Notes in Computer Science.

[GV] J.F. Groote and F.W. Vaandrager: An efficient algorithm for branching bisimulation and stuttering equivalence. *In Proceedings* 17th *ICALP (M.S. Paterson, editor)*, Warwick, volume 443 of LNCS, pages 626-638, Springer-Verlag, 1990.

[GW] R.J. Van Glabbeek and W.P. Weijland: Branching Time and Abstraction in Bisimulation semantics. In *Proceedings 11th IFIP World Comp. Congress*, San Fransisco, 1989.

[Hen] M. Hennessy: *Algebraic Theory of Processes.* MIT Press, Boston, 1988.

[Hil] M. Hillerström: Verification of CCS-processes. *M.Sc. Thesis*, Computer Science Department, Aalborg University, 1987.

[HM] M. Hennessy and R. Milner: Algebraic Laws for Nondeterminism and Concurrency. In *Journal of the Association for Computing Machinery*, v. 32, n. 1, pages 137-161, January 1985.

[ISO] ISO: Information processing systems - open system interconnections - LOTOS - a formal description technique based on the temporal ordering of observational behavior. *ISO/TC 97/SC 21 N DIS8807*, 1987.

[JKP] B. Jonsson, A. H. Khan and J. Parrow: Implementing a Model Checking by Adapting Existing Automated Tools, *In Proceedings of Workshop On Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 1989.

[JP] B. Jonsson, J. Parrow: Deciding Bisimulation Equivalences for a Class of Non-Finite-State Programs. *In Proceedings of the Sixth Annual Symposium on Theoretical Aspects of Computer Science*, 1989.

[Kor] H.P. Korver: Computing Distinguishing Formulas for Branching Bisimulation. *Not published yet.*

[KS] P.C. Kanellakis and S.C. Smolka: CCS Expressions, Finite State Processes and Three Problems of Equivalence. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, 1983.

[Lar] K. G. Larsen: Context-dependent Bisimulation between Processes. *Ph. D. Thesis*, University of Edinburgh, 1986.

[Lee] C.-H. Lee: Implementering av CCS med värdeöverföring, *SICS Technical Report*, 1989 (in Swedish)

[LK] Larsen, K. and A. Skou: Bisimulation through Probabilistic Testing. In *Proceedings of the ACM Symposium on Principles of Programming Languages*, 1989.

[Mil] R. Milner: *Communication and Concurrency.* Prentice Hall, 1989.

[MSGS] J. Malhotra, S.A. Smolka, A. Giacalone and R. Shapiro: Winston, A Tool for Hierarchinal Design and Simulation of Concurrent Systems. In *Proceedings of the Workshop on Specification and Verification of Concurrent Systems*, University of Stirling, Scotland, 1988.

[MSV] E. Madeleine, R. De Simone and D. Vergamini: Ecrins A Proof Laboratory for Process Calculi, *User Manual*, INRIA, 20 October 1989.

[MV] E. Madeleine and D. Vergamini: Finiteness Conditions and Structural Construction of Automata for all Process Algebras. In *Proceedings Workshop on Computer-Aided Verification (R. Kurshan and E.M. Clarke, editors)*, DIMACS technical report 90-31, Vol. 1, New Yersey, 1990. To appear in Lecture Notes in Computer Science.

[MV89] S. Mauw and G.J. Veltink: An introduction to PSF_d. In J. Díaz and F. Orejas, editors, *Proceedings TAPSOFT 89, Vol. 2*, volume 352 of *Lecture Notes in Computer Science*, pages 272–285, Springer-Verlag, 1989.

[MV90] S. Mauw and G.J. Veltink: A process specification formalism. In *Fundamenta Informaticae*, XIII: pages 85–139, 1990.

[NH] R. DeNicola and M. Hennessy: Testing Equivalences for Processes. In *Theoretical Computer Science*, v. 34, pages 83-133, 1983.

[NV] R. De Nicola and F.W. Vaandrager: Three logics for branching bisimulation. *CWI Report CS-R9001*, 1990. Extended abstract in *Proceedings 5th Annual Symposium on Logic in Computer Science (LICS 90)*, Philadelphia, USA, IEEE Computer Society Press, Washington, 1990.

[Ogu] H.M. Oguztuzun: A game characterization of the observational equivalence of processes. In *Proceedings AMAST Conference*, Iowa City Iowa, pages 195-196, 1989.

[Par] J. Parrow: Submodule Construction as Equation Solving in CCS. In *Proceedings of the Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science 287, pages 103-123, Springer-Verlag, Berlin, 1989.

[Pnu] A. Pnueli: Linear and Branching Structures in the Semantics and Logics of Reactive Systems. In *Lecture Notes in Computer Science 194*, pages 14-32, Springer-Verlag, Berlin, 1985.

[PT] R. Paige and R.E. Tarjan: Three Partitioning Refinement Algorithms. In *SIAM Journal of Computing*, 16 (6), pages 973-989, December 1987.

[Rei] W. Reisig: *Petri nets – an introduction.* EATCS Monographs on Theoretical Computer Science, Volume 4. Springer-Verlag, 1985.

[RSa] V. Roy and R. De Simone: An Autograph Primer. *INRIA Technical Report 112*, 1989.

[RSb] V. Roy and R. De Simone: Auto - Autograph. Submitted to *Workshop on Computer-Aided Verification*, Princeton, 1990.

[Si] R. De Simone: Higher-Level Synchronizing Devices in Meije-SCCS. In *Theoretical Computer Science 37*, pages 245-267, North-Holland, 1985.

[SPECS] SPECS: Definition of MR and CRL Version 2.0. *SPECS-deliverable D.WP5.4*, SPECS, 1989.

[SV] R. De Simone and D. Vergamini: Aboard Auto. *INRIA Technical Report RT111*, 1989.

[Vaa] F.W. Vaandrager: Some Observations on redundancy in a context. In *Applications of Process Algebra (J.C.M. Baeten, ed.)*, Cambridge University Press, pages 237-269, 1990.

[Ver] D. Vergamini, Verification of Distributed Systems: an Experiment. In *Formal Properties of Finite Automata and Applications*, LNCS 386, 1990.

[Zui] H. Zuidweg: Verification by Abstraction and Bisimulation. In *Proceedings of Workshop On Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 1989.