Verification of Communication Protocols Using Data Flow Analysis *

Gleb N. Naumovich, Lori A. Clarke, and Leon J. Osterweil

April 29, 1996

email: **{naumovic|clarke|ljo}@cs.umass.edu**
Laboratory for Advanced Software Engineering Research
Computer Science Department
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

In this paper we demonstrate that data flow analysis is an effective approach for verifying requirements of communication protocols. Communication protocols are responsible for establishing the communication patterns between different processes within a distributed computer system. Data flow analysis is a static analysis method for increasing confidence in the correctness of software systems by automatically verifying that a given software artifact (e.g., design or code) must behave consistently with a specified requirement. In this case study, we apply the FLAVERS data flow analysis tool to pseudocode designs of the three way handshake connection establishment protocol and of the alternating bit protocol and prove that the behavior of the pseudocode is consistent with protocol behavioral requirement specifications. In addition, we show how assumptions about the environment in which a software system is executed can be incorporated into the analysis, using message losses as an example. We present experimental results and derive some guidelines about the classes of protocol requirement specifications that may be amenable to verification using FLAVERS.

*This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and the Advanced Research Projects Agency under Contract F30602-94-C-0137.

1 Introduction

In this paper we demonstrate the applicability of an incremental accuracy improving flow analysis approach [DC94] to the verification of properties of communication protocols. We undertake a small case study where we examine two protocols and verify several properties about them. We illustrate the effectiveness of the approach and demonstrate how it can be used to support modeling of the imperfect environment in which communication protocols must operate.

Communication protocols are software systems that are responsible for establishing and maintaining well-defined communication patterns between processes in a distributed system. Various techniques have been employed to assure these patterns of behaviors are correctly captured in design and code artifacts. In this work, we demonstrate the use of static dataflow analysis and argue that this is a particularly effective technique because it is computationally inexpensive, requires minimal human interaction, and is a general and flexible approach that can ensure the consistency of software artifacts, specifically design and code artifacts, to the communication behaviors dictated by communication protocols. Moreover, we demonstrate how it can be extended to incorporate assumptions about the execution environment.

Some features of communication protocols seem to make them particularly appealing subjects for analysis. Foremost they have clearly articulated rules about their behavior that can serve as the basis for verification. Protocol designs and implementations seem to concentrate upon effectively modularizing their communication mechanisms, thereby reducing the size of the representations that verification tools must analyze and helping to concentrate the attention of the verifier. In addition, the variables used to control communications in protocols tend to have finite domains, which also facilitates analyses.

On the other hand, protocols have some features that make them difficult to analyze. The need to reliably exchange messages through unreliable communication media that may create network faults, such as loss, duplication, insertion, reordering or corrupting of messages, requires protocols to use error-recovery algorithms, whose complexity complicates analyses. We demonstrate, using message losses as an example, how these network faults can be modeled and then incorporated into the data flow analysis.

In this work we apply the data flow approach to communication protocol verification, using the FLAVERS (Flow Analysis for Verifying Specifications) tool [DC94]. We evaluate the ability of FLAVERS to specify protocol requirements specifications as event sequences. We then verify that proposed pseudocode designs must adhere to these requirements even in the presence of the possibility of network faults. We evaluate the effectiveness of FLAVERS in the verification of two specific communication protocols: the alternating bit data transfer protocol and the three way handshake connection establishment protocol. A restricted class of message losses is modeled for each protocol. Specifically, we set an upper limit on the number of consecutive losses that can happen in each communication medium before a message is passed without loss. We describe how we can conclusively verify that a specific pseudocode design for each protocol satisfies a number of non-trivial requirements, both under the assumption of fault-free communications and of communications with the possibility of message losses.

The results that have been obtained are encouraging, showing the power of both the basic data flow technique and of incremental techniques directed at improving the accuracy of the analysis. Not surprisingly, verifications of protocols with the possibility of network faults prove to be tractable but considerably more expensive in terms of resource requirements.

The next section of this paper describes approaches that have been applied to the verification of distributed programs, in general, and networks, in particular. Section 3 gives a high-level overview of the FLAVERS data flow analysis approach and the way it can be used to improve analysis accuracy. Section 4 describes our approach for modeling network faults. Section 5 contains descriptions of the protocol design pseudocode used in this case study, each requirement checked, and results from the analysis. In addition, this section summarizes the overall results. Finally, section 6 offers some conclusions based on the results and also indicates directions for future work.

2 Related Work

Olender and Osterweil [OO90] suggest that there are two fundamental approaches to increasing confidence in software: detection (and removal) of errors and demonstration of the absence of errors. Both entail determining if a given software artifact (e.g., design or code) is consistent with specified properties. Testing concentrates on detecting errors, whereas formal methods and static analysis both detect errors and demonstrate their absence. In this section we review applications of testing and formal methods to communication protocols only briefly and concentrate on reviewing static analysis approaches, as that work is closest to our own.

Testing [ABC82] executes a program to determine whether its actual behaviors conform to its expected behaviors. Since the number of possible program executions is usually excessively large, it is impractical to use testing to demonstrate the absence of errors. Testing has been extensively applied to communication protocols [SL89, BP94], but we believe this work should be complemented by work aimed at demonstrating the absence of errors.

Formal verification systems [Hoa69, Pnu77] employ mathematical systems with rules of inference to prove the absence of errors by showing analytically, without actual execution of the program, that a program satisfies or contradicts a given property. Formal verification can formulate and prove intricate properties but are largely manual or semi-automated, and hence expensive, human intensive, and error prone. As such they may be useful for verifying small, critical systems, such as the kernel of life-critical distributed software, but the application of formal verification to larger, less critical systems is usually prohibitively expensive. Examples of the application of formal verification to the analysis of communication protocols are numerous. Kurose and Yemini [KY82] use temporal logic [Pnu77] to prove properties of a reliable connection establishment protocol, Sabnani and Schwartz [SS82] build the proof of properties of a multi-destination message transfer protocol using temporal logic, and Yodaiken and Ramamritham [YR92] prove some properties of a fault-tolerant broadcast protocol using functional specifications and relations semantics.

Static analysis, like formal verification, can demonstrate the absence of certain classes of errors from distributed systems without actually executing them. Unlike formal verification, they often entail little or no human intervention, making them less costly and less error-prone. Static analysis of distributed systems spans such approaches as reachability analysis, constrained expressions, and data flow analysis.

Reachability analysis enumerates possible program execution paths as possible global states, which form a state space that can then be searched for sequences of states that satisfy or violate a specified property. The number of reachable states often causes a so-called *state explosion*, where reachability space size exceeds the processing power or the memory capacity of the machine. Taylor [Tay83] shows that generating all reachable states for a program with a fixed number of communicating tasks is exponential in the number of tasks. Reif and Smolka [RS88] demonstrate the undecidability of generating all reachable states for an arbitrary distributed program. Reachability analysis has been a popular technique for statically analyzing small protocols (e.g. [VHC86, BNY86]), but we are concerned that the inherent exponential nature of this approach will prevent it from scaling up to the analysis of larger, more complex protocols.

A variation of reachability analysis, called model checking, applies to programs where distributed processes can be modeled as communicating finite state automata [CES86]. With model checking, properties are formulated as temporal logic formulas and compared to the logical representation of the program, represented by a reachability graph with temporal logic propositions assigned to the nodes. Properties can be formulated more flexibly than with other reachability approaches, but model checking still suffers from the state space explosion problem. XESAR [RRSV87] is an example of a protocol-specific tool that employs model checking. Several approaches have been suggested for decreasing reachability analysis space requirements [BCM+90, Hol87, GW91, Val91, HGP92]. These approaches significantly improve the feasibility of reachability methods, but in general are still prohibitively expensive to use with most distributed systems.

The constrained expression approach is not based on reachability analysis [ABC+91]. Given a specified pattern of program events in the form of Finite State Automata (FSA), this technique attempts to see if the regular language of an FSA corresponds to some execution of the system. The conditions (e.g.,

maintaining the same number of send and receive events for all communication channels) that characterize all possible program executions are expressed as a system of linear inequalities. The given pattern of program events is also represented in the form of linear inequalities. Integer linear programming is used to solve the resulting system of inequalities. The implementation of this technique has been successfully applied to some distributed systems [ABC+91, Cor92]. However, it also has exponential worst-case bounds. To the best of our knowledge, this approach has not been applied to the analysis of protocols.

Data flow analysis [Hec77] captures static information about a system by computing fixed point data and control dependency information over an annotated flow graph. Data flow analyses have been formulated with low-order polynomial execution time and storage bounds [MR90]. Data flow based approaches for distributed systems have been used to verify both program-independent properties, such as deadlock freedom [MR91], detecting unreachable program statements and determining the values of program expressions [RS90], and concurrent def-use faults (e.g., referencing an undefined variable or defining an unused variable) [TO80], as well as program specific properties [DC94, HS96].

As noted earlier, protocol entities must inevitably communicate through unreliable communication media, which can cause messages to be lost, duplicated, inserted, or corrupted. Protocols describe only the behavior of the logical entities, and provide no information about the medium itself. We propose to verify that protocols will necessarily have desired properties when they communicate through unreliable media. We do this by modeling assumptions about the communication media in the form of finite state automata that constrain the data flow analysis. Cécé, Finkel and Iyer [CFI94] have addressed this with reachability techniques incorporating models of assumptions about network faults into a single FSA built for a protocol. A disadvantage of this approach is that assumptions about the protocol environment and protocol execution behaviors protocol are combined into a single model that must be rebuilt every time the assumptions about either change. Pachl [Pac87] uses state automata to model communication channels during reachability analysis with separate FSAs representing distributed processes and the assumptions about communication channels. We adopt this approach in our own data flow analysis.

In this paper, demonstrate that data flow analysis can be applied efficiently to verification of communication protocol properties. Unlike formal verification, which requires intensive human expertise, the data flow approach is primarily automated. Also, unlike other analysis approaches, which are computationally intractable, data flow analysis is relatively efficient. We show that the efficiency of data flow analysis, combined with techniques for improving its accuracy and for capturing assumptions about the medium, is an effective verification approach for communication protocols.

3 Overview of the Approach

In this section we describe FLAVERS, our use of Ada as a pseudocode protocol design language, and quantified regular expressions (QREs) as a requirements specification language.

3.1 FLAVERS

The FLAVERS static analysis tool performs data flow analysis to verify explicitly stated event sequence properties of distributed systems. Human analysts define sets of program events of interest and formulate properties to be checked as sequences of those events. FLAVERS translates property specifications into FSAs, whose transitions are responses to the events. Given a design or implementation of a distributed system and a property to which that system must adhere, FLAVERS determines whether the property holds on all system executions. FLAVERS models possible executions with a *Trace Flow Graph (TFG)*, whose nodes are annotated with the events of interest.

The TFG representation is the basis for inferring a set of **observable** system execution behaviors, while the FSA representation of a property is the basis for inferring a set of **expected** execution behaviors. The state propagation algorithm compares the sets of observable and expected behaviors and displays the results of this comparison to the user. Figure 1 gives a high-level view of the major components of FLAVERS.

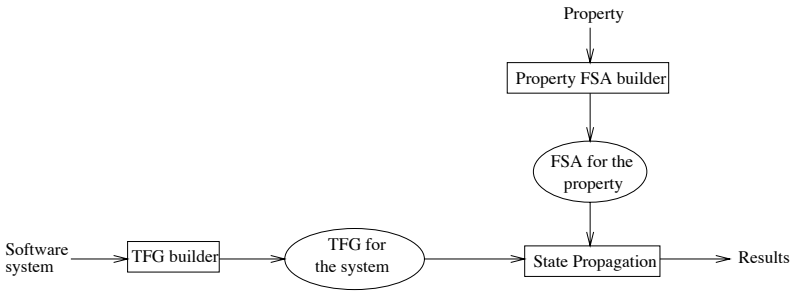


Figure 1: **FLAVERS components**. Boxes represent tools and ellipses represent generated internal representations. Arrows show data flow relations among the tools.

The analysis may report three possible kinds of results. It may demonstrate conclusively that the set of observable behaviors of the system is a subset of expected behaviors, which means that the property holds on **all** executions of the system. The analysis may demonstrate conclusively that observable and expected behaviors are disjoint, which means that the property holds on **no** executions of the system. Finally, the analysis may fail to demonstrate that the set of observable behaviors is either a subset of, or disjoint from, the expected behaviors, which means that the property **may** hold on **some** executions. In this latter case we say that the analysis result is *inconclusive*. This can occur either when the property really holds on some, but not all, executions of the system, or if the analysis is imprecise. Imprecision generally results when it is difficult to draw completely correct inferences about executability from the TFG model, which in order to be conservative over-estimates the executable behaviors of the system. Thus an inconclusive result may be spurious in that the property holds only on unexecutable TFG paths.

Imprecision is inherent in static analysis techniques as information is elided to yield representations of reasonable size. FLAVERS supports the refinement of analysis accuracy by allowing human analysts to incrementally add execution details in the form of *feasibility constraints*, represented by additional FSAs.to the TFG. When well chosen, these constraints eventually succeed in producing conclusive results or in

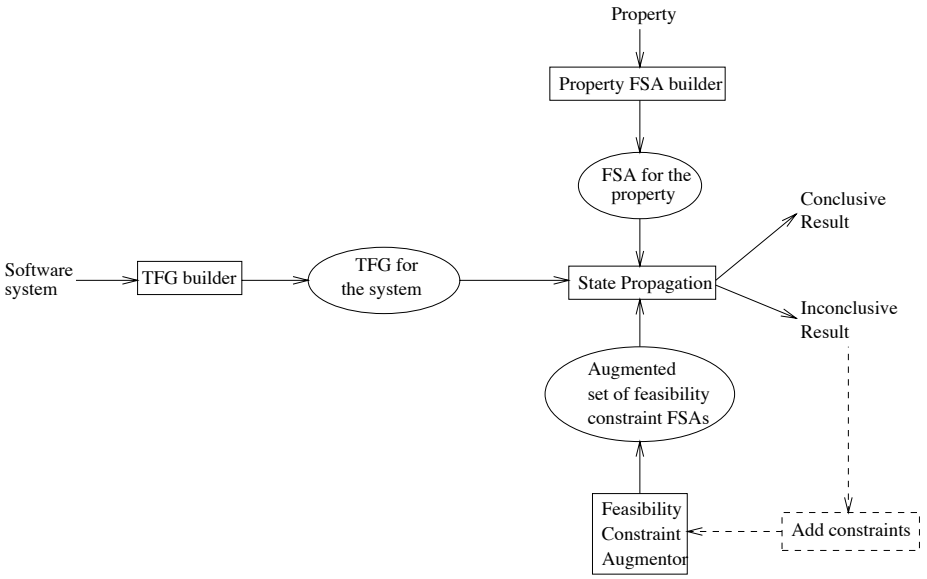


Figure 2: Incrementally improving accuracy of the FLAVERS analysis

convincing the analyst that the property does not hold on all system executions. Two common types of feasibility constraints are *variable automata*, which model the execution behavior of selected variables in the system, and *task automata*, which model all possible orders of events allowed by the control flow in a single process. In general, however, any system execution behaviors that can be represented by an FSA can be used as a feasibility constraint. Figure 2 demonstrates the principle of incremental accuracy improvement.

3.2 Protocol Design Language

A number of protocol specification and design languages have been proposed, the most prominent being CCS [Mil80], Z [Spi92], LOTOS [BB87], SDL [BH89], Estelle [BD87], and Promela [Hol91]. Our choice of Ada as a protocol design language was suggested by some positive experience with it [YGH82, CDG85] and by the fact that the current implementation of FLAVERS can process only Ada. The protocol design pseudocode used in our case studies defines each distributed process and each communication medium as a task, thereby simulating asynchronous message passing between protocols. To validate these choices we also represented the design of the alternating bit protocol in Promela and manually constructed its TFG representation. The result seemed essentially equivalent to the TFG representation of the Ada design. We believe that such equivalence would hold between Ada and Promela representations for other protocol designs as well.

3.3 Property Specification Language

FLAVERS uses *Quantified Regular Expressions (QRE)*, first used in the property specification language CECIL [OO90], to specify the event sequences to which communication protocol designs must adhere. A QRE represents a property as a regular language over the set of events used to define the protocol design. QREs have three components: an event alphabet, a quantifier, and the regular expression. The **all** quantifier indicates that all protocol executions must generate event sequences that belong to the specified regular language. The **none** quantifier indicates that no generated event sequence should belong to the language. Most protocol event sequence properties are easy to represent in regular form, which makes QREs a convenient property specification language. Property QREs are then translated into FSAs.

4 Modeling the Message Passing Environment

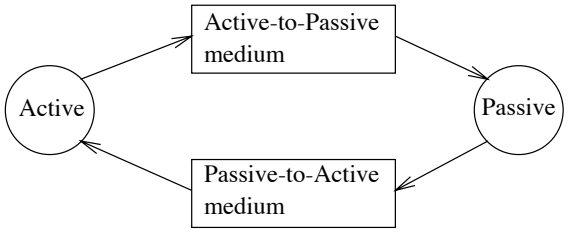


Figure 3: Components of the Three Way Handshake and Alternating Bit Protocols

Figure 3 shows the configuration of communicating processes, **Active** and **Passive**, and the media used for our case study. Each medium transfers messages in one direction only, and each may be subject to such faults as message losses, corruptions, delays, insertions, and duplications. Communication protocols should be able to recover from such faults. We represent user assumptions about the communication media by encoding a general skeleton design of each medium in a separate task, enhanced by one or more feasibility constraints that restrict the medium's behavior. The medium tasks encode only the identities points where messages can be either passed to, or received by, the medium. The feasibility constraints model the communications

over the medium as event sequences over the access points. This approach allows us to use the same task to model both media in figure 3, as well as us to explore different assumptions about their behaviors.

There are two categories of behaviors encapsulated by the media feasibility constraints: fault-free behaviors and various faulty behaviors. Fault-free behaviors represent the basic service a medium provides (e.g. a FIFO queue with buffer size five). Faulty behaviors represent network faults. This differentiation encourages a hierarchical approach to constructing media feasibility constraints: once fault-free behaviors are encoded as feasibility constraints, they can be augmented with faulty behaviors. This approach seems valuable when analyzing protocol code under varying assumptions about medium behavior.

We now describe the fault-free behaviors of the point-to-point communication media used in our case study and show how to augment these fault-free behaviors with assumptions about message losses in the media[1].

4.1 Modeling Fault-free Media

```
task Medium is
  entry In_Message;
  entry Out_Message;
end Medium;
```

```
task body Medium is
begin
  loop
    select
      accept In_Message;
      Transfer_Message;
    or
      accept Out_Message;
    end select;
  end loop;
end Medium;
```

Figure 4: Modeling Media by Ada Tasks

Figure 4 shows a typical task for a medium with two connection points, defined by accept statements **In_Message** and **Out_Message**. A message is received by a medium when an **In_Message** entry call occurs and is passed when an **Out_Message** entry call occurs. This representation alone is not sufficient for a meaningful implementation of connection patterns between distributed processes. Some examples of additional information that could be represented as feasibility constraints to define a more accurate specification of a point-to-point connection are

- specific rules that the medium must follow when passing messages, for example, specifying that it is necessary for a message to be received by the medium before the medium can pass it on.
- the size of the medium buffer;
- the kinds of messages (DATA, ACK, etc.) that the medium can handle and information associated with them (e.g. sequence numbers).

Figure 5 shows two feasibility constraints for a medium that can accept two different kinds of messages. Figure 5(a) shows a medium that can hold at most one message at a time, and figure 5(b) shows a FIFO medium that can hold at most two messages at a time.

[1] Although our approach to modeling communications between distributed processes is general enough to model various kinds of media, in this case study we restrict ourselves to point-to-point message passing

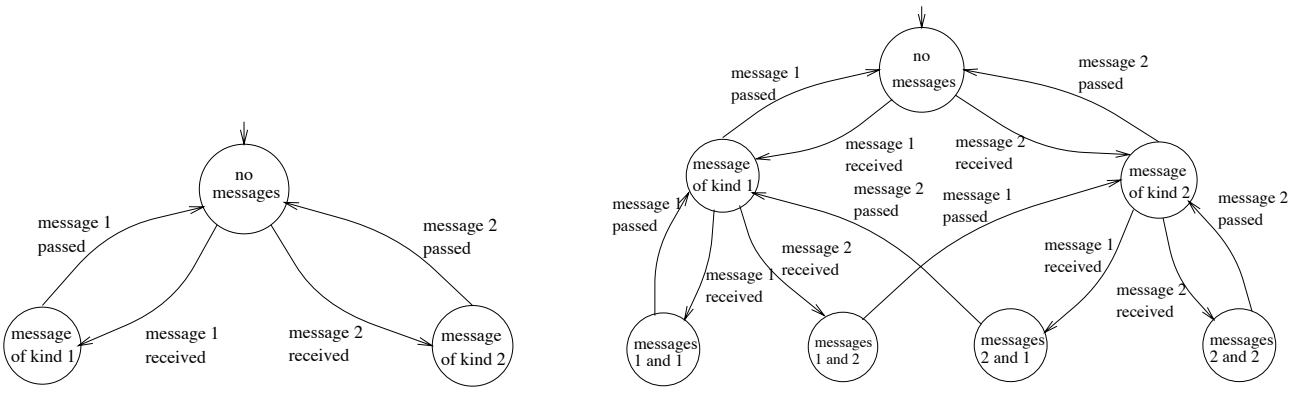


Figure 5: Modeling Media

4.2 Modeling Message Losses

We picked message losses as an example of faulty behavior exhibited in real networks. Other faulty behaviors, such as message corruptions, duplications, and insertions can be modeled analogously. Figure 6 shows how the possibility of message losses can be modeled. The medium task in 6(a) was obtained from the medium task in figure 4 by inserting an **if** statement on one branch of which the event **Lose_Message** occurs. If the value of boolean **NotSuccessful** cannot be predetermined to be always false, then data flow analysis must assume that every time a message is obtained (**In_Message** accepted), a loss is possible. A medium FSA with buffering capacity of one that can pass messages of two kinds is shown in figure 6(b). In that medium FSA, the occurrence of a **Lose_Message** event triggers a transition to the state **No Messages**, meaning that no messages can be obtained from the medium at that point. When the branch containing **Lose_Message** is not taken, the message is assumed not to be lost and can be obtained from the medium.

The resulting model of media behaviors is gratifyingly simple, but also models some unfortunate behaviors, such as indefinite repetitions, where all messages in the system are always lost. This behavior causes a livelock, with one protocol entity sending messages that are always lost by the medium, causing the entity to resend them indefinitely. To exclude such worst-case scenarios from consideration, *fairness assumptions* about the medium are usually made. For example, Kurose and Yemini [KY82] state that a medium is *fair with respect to a message* **m** if it can lose, corrupt or duplicate **m** at most a finite number of times before passing it without a fault. A medium is *fair* if it is fair with respect to any message it receives. We approximate this notion of a fair medium by bounding the number of times a medium can lose, duplicate, or corrupt a given message before passing it through correctly. Figure 7 demonstrates the FSA for a medium with buffer size one that can pass two different kinds of messages with at most one message loss. In our case study, each property was checked assuming up to the maximal number of consecutive message losses in the media.

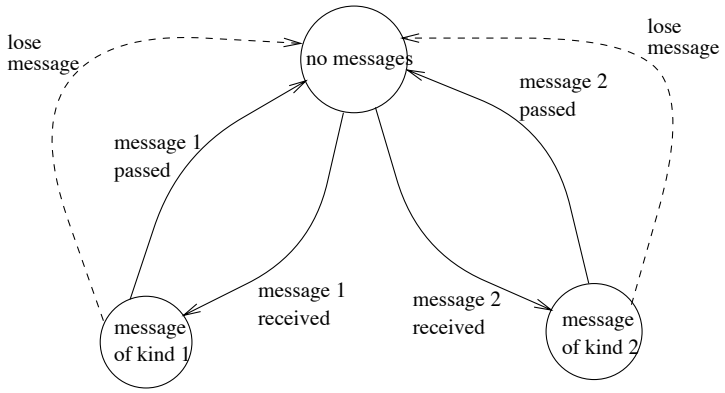
5 Experiments

This section presents the experimental results of running FLAVERS on the three-way handshake connection establishment protocol (3WHS) [KY82] and the alternating bit data transfer protocol (AB) [BSW69]. We chose these case studies for a variety of reasons. Both protocols are popular and well known, and each has been the subject of numerous verification techniques. The protocols come from different classes, the AB

```
task Medium is
  entry In_Message;
  entry Out_Message;
end Medium;

task body Medium is
begin
  loop
    select
      accept In_Message;
      if NotSuccessful then
        Lose_Message;
      else
        Transfer_Message;
      end if;
    or
      accept Out_Message;
    end select;
  end loop;
end Medium;
```

(a)



(b)

Figure 6: Modeling Losses

is a data transfer protocol, and the 3WHS is a connection establishment protocol. In addition, both are straightforward to implement, yet non-trivial. Also, both protocols provide error-checking mechanisms to allow recovery from network faults.

For each protocol we present a description of the protocol, the assumptions made about the communication media, and each requirement specification that we verified with FLAVERS, as well as a short discussion of subjective issues involved in checking the requirement. We were able to verify conclusively all specifications that we analyzed, although the number of constraints required depended on the protocol and requirement. For each protocol and requirement specification, the analysis results are given in a tabular form that lists the execution time taken by FLAVERS, the number of nodes and edges in the protocol TFG, and the data flow lattice size, which is exponential in the product of the number of states of all feasibility constraint and requirement FSAs. The time taken by the property checker is a sum of the application time (the processor clock time) and the system time (operating system overhead) as measured by the UNIX **time** command on a DEC Alpha Station 200 4/233 with 128 megabytes of physical memory. We also include the number of TFG nodes visited during analysis. The lattice size and number of visited nodes serve as metrics of the complexity of the analysis. At the end of the section we summarize our results and venture some guidelines on using FLAVERS for the analysis of communication protocols.

5.1 The Three Way Handshake Connection Establishment Protocol

5.1.1 Description

This protocol defines a sequence of messages exchanged by two communicating protocol entities to establish a connection. One of the communicating entities initiates the connection and is called **Active.** The other accepts the connection request and is called **Passive.** The entities are connected by half-duplex communication media (channels) as shown in figure 3. **Active** initiates a session by sending a SYN message and entering a **synsent** state. **Passive** is initially in the **listening** state, which changes to the **synrcvd** state upon receiving the SYN message. **Passive** then responds with a SYNACK message that acknowledges the receipt of the SYN message. Having received the SYNACK message, **Active** acknowledges the fact by sending an ACK message and enters an **established** state. When **Passive** receives the ACK message it

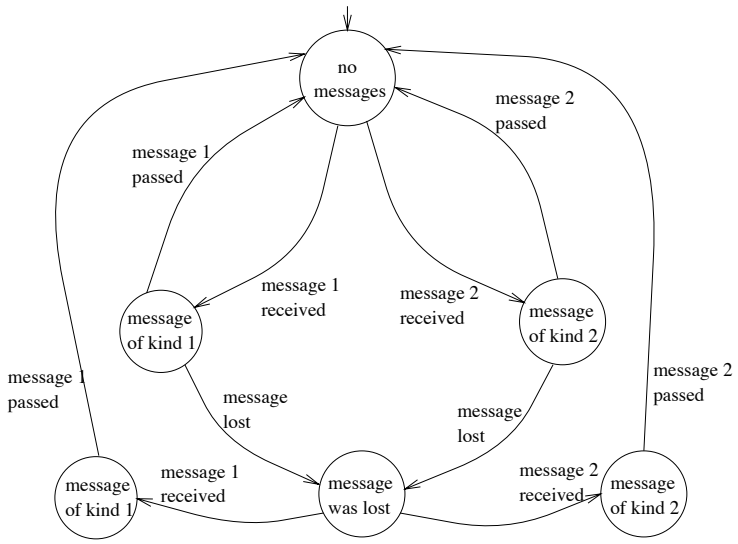


Figure 7: Modeling Fair Losses

also enters its **established** state. The connection is considered established when both entities are in their **established** states at the same time. This completes the basic connection establishment protocol.

The basic protocol does not reliably assure connection establishment when the protocol entities communicate via an unreliable medium. For example, the basic protocol will not perform as designed when a SYNACK message is corrupted, causing **Active** to receive an ACK message. The protocol can be made more robust by having both protocol entities choose an initial sequence number for the current session's messages (to distinguish them from previous or future session messages). The protocol assures that both entities learn the initial sequence number of their correspondent remote entity, and that each entity maintains counts for the messages it sends and for the messages it receives. If at any time during the connection establishment procedure the sequence number of a received message is not the sequence number expected, the entity assumes that the message was not sent during the current session, and either ignores it or resets the connection as described below. **Active** generates its initial sequence number before sending the SYN message. Having received the SYN message, **Passive** learns about the initial sequence number of **Active** and thus knows the correct sequence number of the next incoming message. At this point **Passive** generates its own initial sequence number and passes it to **Active** with the SYNACK message. If at some point during the connection establishment procedure an entity is not in its **established** state and an incoming message has an unexpected sequence number or type, the entity sends a RESET message. If the entity is in its **established** state, and receives a message with an incorrect sequence number, only an ACK message is sent. In this case the remote entity decides whether it can fix the problem by resending some of the messages sent previously or if the RESET message must be sent. When a RESET message is received by **Active**, it starts a new session by generating a new initial sequence number and sending a SYN message. When **Passive** receives a RESET message, it assumes that the session will be restarted by **Active** and waits for a SYN message.

All of these behavioral specifications should be verified for any proposed design or implementation of this protocol. FLAVERS could verify these by expressing them as QREs. We illustrate this by first verifying what we call the **correct sequence numbers** requirement, which states that the 3WHS protocol guarantees that, on all executions where both entities eventually enter their **established** states, each entity expects a message with a correct sequence number. In addition we formulate and verify two more behavioral specifications, **passive states** and **seniority**.

5.1.2 Assumptions about the Media

We analyzed our 3WHS protocol design under two different assumptions about the media: a fault-free medium and a medium that can lose no more than one message of each kind in a row. This means, for example, that having lost an ACK message once, the medium will necessarily pass the next ACK. This is an interpretation of the fairness assumption discussed in section 4. In both cases we assume that message delays are not significant enough to cause timeouts in either entity, and that the only cause of a timeout is a message loss.

5.1.3 Requirements Verified

Correct Sequence Numbers

This requirement specifies that once both **Active** and **Passive** entities have reached their **established** states, the sequence number of the next message sent by one entity is always equal to the sequence number of the next message to be received by the other. Kurose and Yemini [KY82] verified this requirement for 3WHS design pseudocode by manually constructing a formal proof. This requirement must hold on all possible executions of the protocol. Its FSA representation, given in figure 8, is written in terms of four

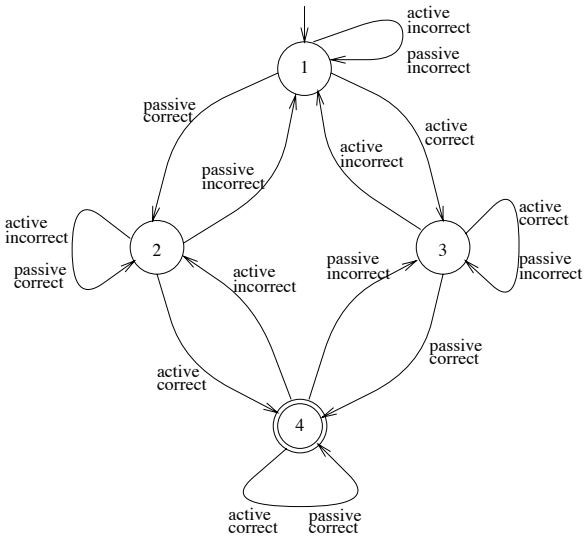


Figure 8: FSA for **correct sequence numbers**

events with which the protocol design pseudocode must be annotated: **passive correct** (**active correct**), which represents an event of the expected sequence number of **Passive** (**Active**) becoming correct with respect to the current **Active** (**Passive**) sequence number; and **passive incorrect** (**active incorrect**), which represents an event of the expected sequence number of **Passive** (**Active**) becoming corrupted as a consequence of a message loss.

To accurately annotate our pseudocode design, we had to model only whether or not the expected sequence number is correct with respect to the sequence number of the next message to be sent by the remote entity. In addition, to achieve desired accuracy, we incrementally added FSAs modeling the current state of **Active** (**initial**, **synsent** or **established**); the current state of **Passive** (**listening**, **synrcvd** or **established**); a global variable **termination** that is set to true when both **Active** and **Passive** are in their respective **established** states to indicate that the connection is established; and task automata for both **Active** and **Passive**. To support modeling the possibility of message losses, a variable that stores the last message sent by an entity was modeled for both entities.

The following table presents the empirical data for the FLAVERS data flow analysis of the **correct sequence numbers** requirement of the 3WHS protocol.

	fault-free	one loss at a time
# TFG nodes	128	160
# TFG edges	4108	5364
$\log_2$(lattice size)	111,488,400	31,948,860,000
application + system analysis time (sec.)	328.80	2121.47
# visited nodes	1403	4538

It was not hard for the human verifier to set up FLAVERS to analyze this requirement. Task automata represent a standard accuracy improving technique used by FLAVERS, and their creation is fully automated. It was quite clear that the variables representing states of **Active** and **Passive** and the variable **termination** affected the requirement and had to be modeled to improve the analysis accuracy. Identifying the places in the pseudocode where the events associated with a sequence number becoming correct or incorrect was only slightly more time-consuming. This required knowledge of the protocol design. Given that many communication protocols use message sequence numbers to provide reliable message passing, special techniques for automatically detecting these kinds of events and inserting the annotations would prove worthwhile and appear quite doable.

Passive States

This behavioral requirement maintains that on no protocol executions does the **Passive** entity go to the **synrcvd** state directly from the **established** state, without passing through the **listening** state first. The FSA representation of the logical complement for this requirement is given in figure 9. This requirement is

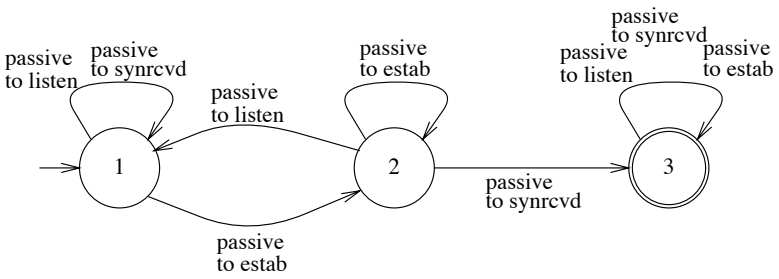


Figure 9: FSA for **passive states**

written in terms of three events: **passive_to_estab**, **passive_to_listen**, and **passive_to_synrcvd**.

For this requirement, only the variable representing the states of the **Passive** entity had to be modeled to constrain the analysis for both the fault-free and possible media losses cases. The empirical data for this requirement are presented in the following table:

	fault-free	one loss at a time
# TFG nodes	49	55
# TFG edges	205	227
$\log_2$(lattice size)	16	16
application + system analysis time (sec.)	3.33	3.80
# visited nodes	377	425

Note that since the same requirement and constraint were used for both kinds of media, the lattice size does not change, and thus the slight increase in running time for the faulty media is due solely to the increased number of visited nodes.

Of all requirements specifications verified, this one was the easiest for a human verifier to check with FLAVERS, mainly because only a small number of events needed to be specified and because only one variable had to be modeled to ensure conclusive results. The pseudocode was annotated only at places

where this variable was modified, again suggesting the possibility of automatic annotation.

Seniority

This requirement states that on all executions of the protocol, the **Passive** entity never goes to the **established** state if **Active** is not in its **established** state. Figure 10 shows the FSA representation of this requirement.

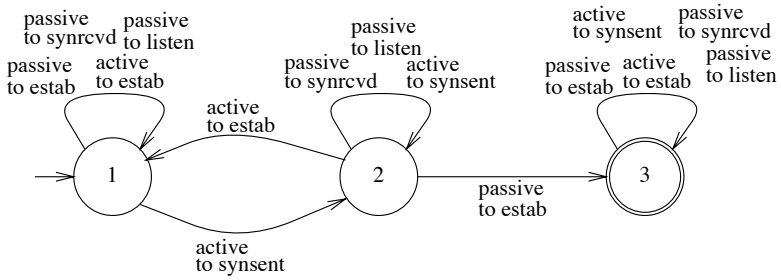


Figure 10: FSA for **seniority**

To achieve the desired analysis accuracy, we had to add task automata for both **Active** and **Passive** as well as FSAs for the current state of **Active**, the current state of **Passive**, and a global variable **termination**. In addition, as was the case with our verification of the **correct sequence numbers** requirement, it was necessary to model variables that store the last message sent by an entity in order to model the possibility of message losses.

The empirical data for this requirement are

	fault-free	one loss at a time
# TFG nodes	102	134
# TFG edges	2118	3042
$\log_2$(lattice size)	4,536,000	1,517,322,000
application + system analysis time (sec.)	398.29	1844.69
# visited nodes	1941	3718

Setting up the FLAVERS analysis for this requirement was very straightforward. Similar to the **passive states** requirement, annotations of the design pseudocode with events were obvious. Deciding which variables to model to ensure the desired analysis accuracy required only a brief examination of the design pseudocode.

5.2 Alternating Bit Protocol

5.2.1 Description

Here we describe our experiments with the alternating bit data transfer protocol. To simplify the design pseudocode, we considered half-duplex, rather than full-duplex transmissions, assuming that connection establishment preceded the data transfer phase and unambiguously identified one of the entities as **Active** and the other as **Passive**. With this protocol, **Active** sends only DATA messages and **Passive** sends only ACK messages acknowledging DATA messages received. Figure 3 shows a high-level design of the network described by the protocol. Note that the same network configuration can use both 3WHS and AB protocols — 3WHS can be used first to establish a connection session between the two entities, and then AB can be used to exchange DATA messages.

The AB protocol specifies that each DATA message sent by **Active** is assigned the alternation bit used for simple error checking. Having sent a DATA message with the alternating bit set, **Active** waits until either an ACK message has been received from the remote entity or a timeout has occurred. In the case of a timeout, the last data message is resent. When an ACK message is received by **Active**, its alternating bit is checked. If it is the same as the alternating bit of the last DATA message sent, it is assumed that the

message has been received correctly by **Passive**, and the next DATA message can be sent. Otherwise, if the alternating bits of the DATA and ACK messages differ, it is assumed that an error has occurred during a corrupted transmission, and the last DATA message is resent. **Passive** does no error checking and only replies to each DATA message it receives with an ACK message carrying the same alternating bit as the DATA message.

5.2.2 Assumptions about the Media

We analyzed a pseudocode design of the AB protocol under three different assumptions about the media: fault-free media, media that can lose at most one message in a row, and media that can lose at most two messages in a row. In all cases the model assumed that message delays do not cause timeouts in either entity, which means that the only cause of a timeout can be a message loss.

5.2.3 Requirements Verified

We verified that our design conformed to three requirements specifications: **wait for acknowledgment**, **message reception**, and **number of repetitions**. To save space, we do not give FSA representations for these requirements.

Wait for Acknowledgment

This requirement states that for each DATA message sent by **Active** with an alternating bit set to 0, an ACK message with the alternating bit set to 0 must eventually be received by **Active** before it sends any DATA messages with the alternating bit set to 1.

Message Reception

This requirement verifies the rules of message reception for the Passive entity. It states that if a DATA message d is sent by **Active**, then no other DATA messages are sent until d is either lost or acknowledged by **Passive**.

Number of Repetitions

This requirement has different formulations depending on the number of losses modeled by each medium. If no message losses are possible, **Active** (not surprisingly) never sends the same DATA message twice in a row. In the presence of no more than one message loss in each medium, we can show that **Active** may have to repeat the same DATA message with an alternating bit 0 up to four, but never **five** times in a row. When up to two messages in a row can be lost by each medium, **Active** may have to repeat the same DATA message with an alternating bit 0 no more than 9, but never **ten** times in a row.

To conclusively verify these three requirements, we had to incrementally add task automata for both **Active** and **Passive** as well as model a global variable **termination** that is set to true when both **Active** and **Passive** are in their respective **established** states at the same time, a variable that stores the alternating bit of the last DATA message sent by **Active**, and a variable that stores the alternating bit of the last ACK message sent by **Passive**.

The empirical data for the cases of fault-free media, one loss media, and two losses media are as follows:

Wait for acknowledgment

	fault-free	one loss at a time	two losses at a time
# TFG nodes	53	67	67
# TFG edges	181	332	332
$\log_2$(lattice size)	295,488	1,080,000	2,116,800
application + system analysis time (sec.)	52.90	547.11	1238.96
# visited nodes	1002	2103	2658

Message Reception

	fault-free	one loss at a time	two losses at a time
# TFG nodes	53	67	67
# TFG edges	181	332	332
$\log_2$(lattice size)	492,840	1,800,000	3,528,000
application + system analysis time (sec.)	53.76	544.58	1279.05
# visited nodes	1010	2103	2658

Number of Repetitions

	fault-free	one loss at a time	two losses at a time
# TFG nodes	53	67	67
# TFG edges	181	332	332
$\log_2$(lattice size)	393,984	2,520,000	8,467,200
application + system analysis time (sec.)	55.61	741.28	3463.33
# visited nodes	1002	2295	3729

We noted similar levels of human effort to perform FLAVERS analysis for all three requirements. For all three the same event sets and feasibility constraints were used, and annotations were needed only at the clearly identifiable sites of message sends, receives and losses, for which FLAVERS has an automatic annotator. These similarities make it possible to run all three FLAVERS analyses simultaneously. We opted for separate verifications to assure that our measured empirical data were consistent with other 3WHS data.

5.3 Some Observations

For nearly all requirements verified on both protocols, it was surprisingly easy to identify the variables needed to constrain the control flow in the problem. The only exception was the need for modeling **termination** variables because of a current limitation of FLAVERS when checking liveness properties. For the majority of requirements it was very easy to annotate the source code with the events used by the requirements. Only in the case of the **correct sequence numbers** requirement, checked on the 3WHS protocol, was a more thorough subjective scrutiny of the pseudocode needed to determine precisely where to put expected sequence numbers annotations. Task automata were used as a standard technique to improve the accuracy of the analysis. They were required in all cases except when checking the **passive states** requirement of the 3WHS protocol. Creating task automata is fully automated, thus requiring no modeling effort from the human verifier.

We determined lattice sizes to estimate the complexity of each data flow problem solved. This size depends on the number of, and the number of states in, feasibility constraints needed to achieve the desired analysis accuracy. Execution time was shown to correlate with lattice size, indicating that it is desirable to limit the size and number of feasibility constraints where possible. It appears that a requirement should be specified first, and then protocol specification variable dependencies analyzed, so that only variables affecting events in the requirement are modeled. Another observed dependency is a positive correlation between the number of TFG nodes and edges and the number of pseudocode annotations. Since the size of a TFG depends on the number of annotations, reducing the number and size of feasibility constraints can lead to reductions in the size of the TFG representation. A smaller TFG representation for a protocol design reduces the number of TFG nodes and edges that must be evaluated by FLAVERS.

Although the number of requirements and protocols examined is very modest, comparing timings for different requirements suggests some interesting tendencies. For example, verifying the **passive states** requirement on the 3WHS protocol confirms the hypothesis that FLAVERS analysis cost is influenced by the degree to which a requirement is centralized [Dwy95]. We call a requirement centralized when it is specified in terms of events from only a small group of distributed processes in a protocol specification. Centralized requirements are usually easier to check than non-centralized ones because fewer variables have to be modeled. For example, the **passive states** requirement refers only to events local to the **Passive** entity, making it easier to check than the **seniority** requirement, which refers to events from both **Passive**

and **Active** entities. This notion of centrality suggests a compositional approach to the analysis of protocols. It is quite likely that even very complex requirements formulated for a protocol as a whole can be decomposed into a number of more trivial requirements, each referring to only a subset of the distributed processes defined by the protocol.

As expected, we found it much harder to verify protocols in the presence of network faults. Although our modeling of losses is very restricted, it is nevertheless clear that for most of the examples considered, the analysis of a system in the presence of losses is at least an order of magnitude more expensive than the analysis of the system with no losses possible. From the human verifier's perspective, however, adding the possibility of faulty behaviors to the analysis was very easy. In most cases it required only the addition of the media feasibility constraints. Checking **correct sequence numbers** and **seniority** requirements of the 3WHS protocol also required modeling an additional variable that stores the most recent message sent by **Active**, but this was immediately evident and straightforward. This shows that, although the analysis of protocols in the presence of possibility of losses requires much more computational resources than when network faults are not taken into account, very little additional effort is required from the human verifier to incorporate these faulty behaviors into the FLAVERS analysis.

6 Conclusions

Data flow analysis is a powerful static analysis technique. We have shown that the data flow approach can be used to verify that non-trivial communication protocols requirements specifications are satisfied by pseudocode designs. A number of requirements specifications for the three way handshake connection establishment and the alternating bit data transfer protocols have been formulated as event sequences and conclusively verified using the FLAVERS data flow analysis tool. The time spent by the human verifier in preparing the analyses and incrementally adding feasibility constraints was reasonable, ranging from a few hours for the most difficult requirement, **correct sequence numbers**, to a few minutes for **passive states**. This, in conjunction with the fact that all requirements were verified conclusively, leads us to believe that incremental accuracy improving flow analysis is a powerful and practical approach to verifying communication protocol requirements specifications. The actual analysis times seem to us to be more interesting as relative, not absolute, measures. They give a useful notion of the increase in the analysis complexity when faulty behaviors of the media are modeled. We expect to refine FLAVERS to achieve significant reductions in the analysis time in future work with this system.

A number of directions for improving the current implementation of FLAVERS have been uncovered by the experiments. The ability to handle liveness conditions would allow verification of a broader class of protocol requirements specifications. One of the ways to address this problem is to introduce *anchored* quantified regular expressions [OO90], which allow the analysis to be restricted to a particular part of the TFG. Another drawback of the FLAVERS implementation is its limitation in supporting variable modeling. Only boolean variables can now be modeled automatically. Automated support for creating variable automata should significantly reduce the effort required from the human verifier to prepare analyses. The rapid increase in the analysis time for communication models that allow for the possibility of message losses might be an indicator of the need for a more effective state propagation algorithm when the lattice for a problem becomes large. Efficiency of the analysis is closely related to the size and connectedness of the TFG for a given protocol specification. Optimization techniques aimed at reducing the number of TFG edges should increase the efficiency of the analysis significantly. Also, by automating the decomposition of protocols and requirements, we can employ a hierarchical approach to the analysis of complicated requirements; by representing a requirement as a conjunction of a number of simpler requirements, the analysis time can be reduced if the time that it takes to prove each of the simpler requirements is significantly less than the time to prove the complex requirement.

Overall, the results presented in this paper are encouraging in showing the applicability of the incremental accuracy improving flow analysis technique to the verification of communication protocols. We consider this an extremely promising approach for the protocol domain as well as other domains where event sequences

capture some of the behavior requirements.

7 Acknowledgments

The authors gratefully acknowledge the assistance of Jim Kurose in providing background and insights into communications protocols, and the support of Matt Dwyer, Tim Chamillard, George Avrunin in understanding, applying and interpreting the results of FLAVERS.

References

[ABC82] W. Richards Adrion, M.A. Branstad, and J.C. Cherniavski. Validation, Verification, and Testing of Computer Software. *ACM Computing Surveys*, 14(2):159–192, June 1982.

[ABC+91] George S. Avrunin, Ugo A. Buy, James C. Corbett, Laura K. Dillon, and Jack C. Wileden. Automated Analysis of Concurrent Systems with the Constrained Expression Toolset. *IEEE Transactions on Software Engineering*, 17(11):1204–1222, November 1991.

[BB87] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems, North-Holland*, 14:25–59, 1987.

[BCM+90] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, 1990.

[BD87] S. Budkowski and P. Dembinski. An introduction to Estelle: A specification language for distributed systems. *Computer Networks and ISDN Systems, North-Holland*, 14:3–23, 1987.

[BH89] F. Belina and D. Hogrefe. The CCITT-specification and description language SDL. *Computer Networks and ISDN Systems*, 16, 4:311–341, 1989.

[BNY86] Naser Barghouti, Nihal Nounou, and Yechiam Yemini. An integrated protocol development environment. *Proc. 6th int. symp. on protocol specification, testing and verification, June 10-13, 1986*, 1986.

[BP94] Gregor Bochmann and Alexandre Petrenko. Protocol testing: Review of methods and relevance for software testing. In *Proc. 1994 International Symposium on Software Testing and Analysis*, August 1994.

[BSW69] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex lines. *Comm. of the ACM*, 12(5):260–265, 1969.

[CDG85] R. Castanet, A. Dupeux, and P. Guitton. Ada, a well suited language for specification and implementation of protocols. *Proc. 5th int. symp. on protocol specification, testing and verification, June 10-13, 1985*, 1985.

[CES86] E. M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CFI94] Gérard Cécé, Alain Finkel, and S. Purishothaman Iyer. Duplication, insertion and lossiness errors in unreliable communication. In *Proc. of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering*, December 1994.

[Cor92] James C. Corbett. Verifying General Safety and Liveness Properties with Integer Programming. In *Proceedings of the Fourth Workshop on Computer Aided Verification*, pages 337–348, 1992.

[DC94] Matthew Dwyer and Lori Clarke. Data Flow Analysis for Verifying Properties of Concurrent Programs,. In *ACM SIGSOFT'94 Software Engineering Notes, Proceedings of the Second ACM Sigsoft Symposium on Foundations of Software Engineering, v. 19, n. 5*, pages 62–75, December 1994.

[Dwy95] Matthew Dwyer. *Data Flow Analysis for Verifying Correctness Properties of Concurrent Programs.* PhD thesis, University of Massachussetts, Amherst, 1995.

[GW91] Patrice Godefroid and Pierre Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. In *Proceedings of the Third Workshop on Computer Aided Verification*, pages 417–428, July 1991.

[Hec77] M.S. Hecht. *Flow Analysis of Computer Programs.* North-Holland, New York, 1977.

[HGP92] G. J. Holzmann, P. Godefroid, and D. Pirottin. Coverage preserving reduction strategies for reachability analysis. In *Proc. 12th Int. Conf on Protocol Specification, Testing, and Verification, INWG/IFIP*, Orlando, Fl., June 1992.

[Hoa69] C.A.R. Hoare. An Axiomatic Basis of Computer Programming. *Communications of the ACM*, 12(10):576–583, October 1969.

[Hol87] Gerard J. Holzmann. On limits and possibilities of automated protocol analysis. *Protocol Specification, Testing and Verification VII, IFIP 1987*, pages 339–344, 1987.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall Software Series, 1991.

[HS96] W. E. Howden and G. M. Shi. Linear and structural event sequence analysis. In *ISSTA*, 1996.

[KY82] James F. Kurose and Yechiam Yemini. The specification and verification of a connection establishment protocol using temporal logic. *Proc. 2nd int. workshop on protocol specification, testing and verification, May 17-20*, 1982.

[Mil80] Robin Milner. *A Calculus of Communicating Systems, Lecture Notes in Computer Science, Vol.92.* Springer-Verlag, Berlin, 1980. Appeared as vol. 92, *Lecture Notes in Computer Science.*

[MR90] T. J. Marlowe and B. G. Ryder. Properties of Data Flow Frameworks. *Acta Informatica*, (28):121–163, 1990.

[MR91] S.P. Masticola and B.G. Ryder. A Model of Ada Programs for Static Deadlock Detection in Polynomial Time. In *Proceedings of the Workshop on Parallel and Distributed Debugging*, pages 97–107. ACM, May 1991.

[OO90] Kurt M. Olender and Leon J. Osterweil. Cecil: A Sequencing Constraint Language for Automatic Static Analysis Generation. *IEEE Transactions on Software Engineering*, 16(3):268–280, March 1990.

[Pac87] Jan Pachl. Protocol description and analysis based on a state transition model with channel expressions. *Protocol Specification, Testing and Verification VII, IFIP 1987*, pages 207–219, 1987.

[Pnu77] Amir Pnueli. The Temporal Logic of Programs. In *Proceedings of the Eighteenth Symposium on Foundations of Computer Science, Providence, 1977*, pages 46–57, 1977.

[RRSV87] J. L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. Verification in Xesar of the sliding window protocol. *Protocol Specification, Testing and Verification VII, IFIP 1987*, pages 235–248, 1987.

[RS88] J. Reif and S. Smolka. The Complexity of Reachability in Distributed Communicating Processes. *Acta Informatica*, 25(3):333–354, 1988.

[RS90] John H. Reif and Scott A. Smolka. Data flow analysis of distributed communicating processes. *International Journal of Parallel Programming*, 19(1), 1990.

[SL89] Deepinder P. Sidhu and Ting-Kau Leung. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, 15(4):413–426, April 1989.

[Spi92] J. M. Spivey. *The Z Notation A Reference Manual.* Prentice Hall, International Series in Computer Science, 1992.

[SS82] Krishan Sabnani and Mischa Schwartz. Verification of a multidestination protocol using temporal logic. *Proc. 2nd int. workshop on protocol specification, testing and verification, May 17-20*, 1982.

[Tay83] Richard N. Taylor. Complexity of Analyzing the Synchronization Structure of Concurrent Programs. *Acta Informatica*, 19:57–84, 1983.

[TO80] Richard N. Taylor and L. J. Osterweil. Anomaly Detection in Concurrent Software by Static Data Flow Analysis. *IEEE Transactions on Software Engineering*, 6(3):265–278, May 1980.

[Val91] A. Valmari. A Stubborn Attack on State Explosion. In Edmund M. Clarke and R.P. Kurshan, editors, *Computer-Aided Verification 90*, pages 25–41. American Mathematical Society, Providence RI, 1991. Number 3 in DIMACS Series in Discrete Mathematics and Theoretical Computer Science.

[VHC86] Son T. Vuong, Daniel D. Hui, and Don D. Cowan. Valira — a tool for protocol validation via reachability analysis. *Proc. 6th int. symp. on protocol specification, testing and verification, June 10-13, 1986*, 1986.

[YGH82] Larry Yelowitz, Susan Gerhart, and G. Hilborn. Modeling a network protocol in affirm and ada. *Proc. 2nd int. workshop on protocol specification, testing and verification, May 17-20, 1982*, 1982.

[YR92] V. Yodaiken and Krithi Ramamritham. Verification of a Reliable Net Protocol. In *Formal Techniques in Real-Time and Fault-Tolerant Systems, Lecture Notes in Computer Science No. 571*, pages 193–215. Springer-Verlag,, 1992.