# Contents

1 Introduction .......................... 4
   1.1 Preface .......................... 4
   1.2 Verification of Programs ......... 6
   1.3 The Interleaving Model .......... 8
   1.4 Linear Temporal Logic .......... 9
   1.5 Event Structures ................. 10
   1.6 The Traces Model ................. 12
   1.7 Translation between Traces and Event Structures 14
   1.8 Petri Nets ....................... 15
   1.9 Interleaving Set Temporal Logic . 16

2 Proving Partial Order Properties .... 19
   2.1 Motivation and Intuition for the Proof Rules . 19
   2.2 A Complete Proof System for Safety and Eventuality Properties 25
   2.3 Example .......................... 32
   2.4 Soundness and Completeness ........ 34

3 Verification of Distributed Programs using Representative Interleaving Sequences 42
   3.1 Introduction ..................... 42
   3.2 The Verification Method .......... 43
      3.2.1 Verification Using Representatives . 44
      3.2.2 Proof Rules .................. 46
      3.2.3 Proof Lattices ............... 49
   3.3 A Detailed Example ............... 50
   3.4 Applicability of the Method ...... 54
      3.4.1 The Generalization Property . 55
      3.4.2 Communication Closed Layers . 56
      3.4.3 Adding the Termination Assumption 59
      3.4.4 Completeness and Efficiency . 59

4 Defining Conditional Independence Using Collapses 60
   4.1 Introduction ..................... 60
   4.2 Conditional Traces ............... 61
   4.3 Examples: Two Programs ........... 64
   4.4 An Application: Verifying Total Correctness 66

© Technion - Israel Institute of Technology, Elyachar Central Library
### 5 Expressing and Proving Serializability in Temporal Logic

#### 5.1 Preliminaries
- 5.1.1 Variables
- 5.1.2 External and Internal Operations

#### 5.2 Expressing Serializability in Linear Temporal Logic

#### 5.3 Expressing Commutativity-Based Serializability
- 5.3.1 No failure assumption
- 5.3.2 Adding Aborts and Active Transactions
- 5.3.3 Proof Rules for Verifying Serializability
- 5.3.4 Example for Verification of an Algorithm
List of Figures

1.1  Net A ................................................................. 16
1.2  Runs of Net A ...................................................... 17
2.1  Nets B and C ......................................................... 20
2.2  Two runs of \{\alpha, \gamma\} ....................................... 22
2.3  Net D ................................................................. 25
2.4  Net E ................................................................. 27
2.5  Proof graph for Net E ............................................. 28
2.6  An example of a tree for rule back ......................... 29
2.7  Verification sketch ................................................ 33
2.8  Graphs for next and sback ..................................... 33
3.1  An algorithm with two phases ................................. 43
3.2  A simple decomposition .......................................... 45
3.3  Example of a confusion ......................................... 46
3.4  Program SORT ..................................................... 50
3.5  Outline of proof for SORT ...................................... 51
3.6  Single wave propagation ......................................... 52
3.7  Migrating a single value to the right ....................... 53
3.8  A diamond .......................................................... 55
3.9  Sketch for the proof of the diamond property ............. 56
5.1  A tree for proving serializability for 2PL .................. 83
Abstract

There exist two main approaches to defining semantics for concurrent programs. The first and earlier approach represents an execution of a concurrent program as a sequence of events (an event is the execution of a program operation) that are executed, called an interleaving sequence. The second, newer approach, describes an execution as a partially ordered set of events. The existence of an order between a pair of events corresponds to the fact that one of them is executed before the other, while its non-existence means the possibility that they are executed concurrently.

Partial order semantics has recently the subject of extensive research. There are many related versions of partial order semantic models, designed to permit specification of concurrent programs. The logic ISTL* was suggested to give specifications based on these models. This logic reconciles partial orders with interleavings generated by taking their linearizations (completions to total orders). Thus the logic enjoys the advantageous of both approaches.

The purpose of this research is to supply verification methods for properties of partial order semantics, and for properties expressible over interleaving semantics, whose proof within the framework of partial order semantics is simpler and more convenient.

A sound and complete proof system will be presented for properties that interest many researchers in this field. Such properties can express for example the concurrent execution of segments of the program, finding a distributed snapshot (global state) of the program, the execution of the program according with communication closed layers, and the immediate response to a request. These properties are based upon partial order semantics and cannot be expressed in logics based on interleaving semantics in a natural way.

Classes of properties are presented that can be expressed directly in interleaving semantics, but establishing their verification on partial order semantics results in a simpler and more intuitive proof. A separate proof system is given that is complete only for such properties and is easier to apply than the previous one. The convenience of the proof method is demonstrated for verifying total correctness (termination of the program with respect to given initial and final conditions).

Although the verification methods presented herein are independent of some aspects of the partial order model, and thus can be based on various partial order semantics, they will be presented using the traces model of Mazurkiewicz. This model has the advantage of a simple representation. A partial order is represented as the set of its total orders, denoted as strings.

The main disadvantage of this model is the requirement of a fixed dependency or independence between events obtained by the execution of the same pair of operations (dependency between operations implies that they must be interordered). Thus, the semantics of traces is extended to conditional traces, which allow dependency to vary from state to
state. In this way, not only is the applicability of the methods presented here extended, but a simple and strong semantic tool for representing partial orders is provided.

Finally, the approach is used to handle a property of concurrency control of databases called *serializability*. This property requires that transactions, which are collections of operations that are supposed to execute a single user's request, are executed as if they were atomic. That is, their execution is equivalent to a sequence of occurrences in which their is no overlapping among their events, although in reality concurrency and interaction between their operations are allowed. This property will be expressed in ISTL* and it will be shown how it can be verified for concurrency control algorithms.