INF5140: Specification and Verification of Parallel Systems

Lecture 11-3 – The Great Debates

Gerardo Schneider

Department of Informatics
University of Oslo

INF5140, Spring 2006

Outline

CTL vs LTL

- Two main types of temporal logic used in model checking
  - CTL (Computational Tree Logic): mainly used in hardware verification
  - LTL (Linear Temporal Logic): almost exclusively used for software verification
- Main debates concern
  - Relative expressiveness
  - Relative complexity
- Their expressiveness are incomparable
  - CTL can express *reset* properties (LTL cannot)
    - From every state there exists *at least one* execution that can make the system return to the initial state
  - LTL can express *fairness* (CTL cannot, although it can be embedded into the verification algorithms)
    - Every cyclic execution either must (or may not) traverse specific types of states infinitely often

CTL vs LTL
Complexity

- Worst case complexity
  - CTL is linear on the size of the formula
  - LTL is exponential on the size of the formula
- In practice there is no big difference in performance!
  - A nice example is given in Holzmann's pp. 564

CTL vs LTL
Complexity

- The worst-case behavior of LTL converters is rare
- Practical LTL formula usually have two or three temporal operators
- What affects complexity is not the number of *potentially* reachable sates but the *effectively* reachable states
- LTL verification algorithms can more easily be implemented with on-the-fly strategies (no need to build the whole state space)

Symbolic Verification

- Symbolic verification algorithms (BDD-based methods) are very effective in hardware verification
  - Boolean data (bit-vectors) are common in hardware verification which are well represented using BDDs
- In general they perform poorly in software verification problems
- The performance depends critically on the variable ordering chosen for the BDDs
  - Choosing an optimal variable ordering is NP-complete
- The memory used by a BDD-based method is determined by the number of nodes in the BDD structure

Explicit Verification

- In software verification partial order reduction strategies perform very well
  - Complex and highly correlated data structures, common in software verification problems, are not easily exploited with BDDs
  - Partial order reduction techniques apply well to asynchronous process execution
  - Synchronous, clocked, operations not well suitable for partial order reduction
- Computing the optimal reduction is NP-complete
- The memory used by an explicit state method is determined by the number of states stored
- In terms of memory consumption (number of bytes used), there is **no** big difference between symbolic and explicit verification methods

Breadth-First Search vs Depth-First Search

- In explicit verification, is it better to use breadth-first (BFS) or depth-first (DFS) default search algorithm?
- Spin's default algorithm is DFS
  - BFS is a user-defined option
- Main advantage of BFS: for *safety* properties, it finds the shortest counterexample
  - Usually DFS finds a longer path
- Main advantage of DFS: to get counterexamples it suffices to print out the content of the stack
  - With a BFS, more information needs to be stored
- For *liveness* properties or properties of infinite sequences, both algorithm may be used
  - Tarjan's classic DFS algorithm and Spin nested DFS method are efficient variants of DFS
- In verification of hybrid and real-time systems BFS algorithms are more common
  - In specific applications (e.g. polygonal hybrid systems) DFS algorithms seem more natural

Tarjan Search vs Nested Search

- The classical way to detect the presence of infinite accepting runs in a finite reachability graph is to use Tarjan's DFS algorithm for constructing all the SCC of the graph
- In the worst-case, Tarjan's algorithm visit every reachable state twice
- Advantage: it detects **all** accepting runs
- Spin nested DFS algorithm does not detect all accepting runs
- The search is set up s.t. an accepting run corresponds to a counterexample of a correctness claim
- The worst-case time complexity is the same as Tarjan's algorithm but the memory overhead is lower
- Tarjan's algorithm makes the implementation of strong fairness constraints easy
  - Spin does no support strong fairness

Event vs States

- Existing model checking techniques usually represent finite-state machines as annotated graphs using formalisms which are:
  - State-based, or
  - Event-based
- Both frameworks are interchangeable
- It is difficult, however, to express actions (events) which are data-dependent
  - Difficult to annotate the program and to specify correctness claims
- Spin is an *explicit state* model checker
  - The verifier builds a global state reachability graph
- Correctness properties are also formalized as simple boolean properties of system states

Event vs States

- Some properties are difficult to establish in a state-based setting and others are easier than in an event-based
- For example, for the correctness property:

  "Always within a finite amount of time after the transmission of a message, the message will be received at its destination"

  - If the message is sent to a buffered channel, the state changes as a result of the send and receive events
  - If send and receive are rendez-vous handshakes, recording the execution in a way observable to Spin is a bit subtle
- The ideal is to have a combined approach without adding any verification penalty
  - See "State/Event-based Software Model Checking" by Chaki et al (based on *labeled Kripke structures*)

Real-time vs Timeless Verification

- Real-time verification techniques need the explicit representation of time (discrete or continuous)
  - Typical properties are response-time and time deadlines
- The computational complexity of real-time analysis is high (undecidable in many cases)
- Spin focus only on *functional* and *logical* correctness issues
  - No assumption about relative speed of execution of asynchronous processes
  - Time is abstracted away
- Promela only has a rudimentary notion of timeout (which is not timed)
- Spin is not meant to be used as a performance analysis tool

Probabilities vs Possibilities

- Standard model checking algorithm may be modified in order to include probability
- In most cases the inclusion of probabilities increase the verification complexity
  - In some cases (e.g. there are examples in real-time verification), adding probability simplifies the verification analysis
- Promela/Spin does not have probabilities and only deal with possible system behaviors

Asynchronous vs Synchronous Systems

- Most hardware model checkers have a synchronous view
  - All process actions are clock-driven
  - Every process take a step at every clock tick
- Asynchronous behavior can be modeled in the synchronous setting
- Distributed systems are essentially asynchronous
- Spin is one of the few asynchronous model checkers
- Advantages of asynchronous model checkers:
  - Greater verification efficiency
    - Only w.r.t. to explicit state verification methods (not compared with symbolic algorithms)
- Main disadvantage of asynchronous model checkers:
  - Difficult to model synchronous behavior
    - That's why Spin is not good for hardware verification

Interleaving Semantics vs True Concurrency

- True concurrency semantics allows the simultaneous execution of actions, in addition to the interleaving of actions
- In distributed systems, two asynchronous processes may execute actions at the same time
- In terms of verification, true concurrency implies the addition of more transitions to the verification model
- Is true concurrency really needed?
  1. If the process actions access either distinct data objects or none at all
     - Interleaving gives a correct interpretation: the simultaneous execution of actions is indistinguishable from any sequential interleaving
  2. If two processes access shared data
     - By representing the data objects at some level of granularity it is always possible to accurately describe the possible behavior with interleaving semantics
- Hence, interleaving semantics offer the simplest sufficient model

Open vs Closed Systems

- Traditionally model checking is based on two requirements on the model: finiteness and being closed
- To be closed, a system must include all the possible inputs and possible interactions with the environment
- Open systems are more difficult to analyze since in many cases the environment behavior is not known
- In practice open systems are verified by making worst-case assumptions about the environment
  - This is compatible with the assume-guarantee style of reasoning

Backward vs Forward Reachability

- Many verification problems can be reduced to reachability analysis
- Reachability may be done using a forward or a backward analysis
- Forward: Verification start in the initial state and computes the set of successors in the reachability graph
  - It terminates when the intersection of reachable states with the intended (bad) state is non-empty
- Backward: Verification start in a final state (or a state marked as *bad*) and computes the set of predecessors in the reachability graph
  - It terminates when the intersection of reachable states with the *initial* state is non-empty
- In both cases heuristics may direct the search space
- Which one to use depends on the particular application and the kind of properties
  - For instance, in real-time verification backward reachability is widely used for safety properties (the timed automata containing an error state)

Compositional vs Non-compositional Verification

- In fact there is no big debate about this topic since everybody wants to have a compositional verification method
- Compositionality is good for
  - Increasing local reasoning
  - Minimizing the state-space search (using less memory)
  - Parallelizing (distributing) the verification algorithm without repeating computation
- If it so good, why not to write only compositional algorithms?
- Obtaining compositional algorithms is not easy in general
  - Many approaches are non-compositional or they are restricted to certain kind of subsystems and properties
  - In many cases the underlying system is not compositional by nature
  - Sometimes it is only possible to obtain partial compositional algorithms, still needing a global computation but in a reduced state-space

Deductive vs Algorithmic Verification

- Verification can be done following a deductive approach (like Manna & Pnueli's formalism) or algorithmically (like Spin)
- Deductive verification
  - Usually uses theorem provers/assistants
  - Difficult to automatize
  - Application domain: any kind of system (finite and infinite)
  - Needs a lot of expertise for using it
- Algorithmic verification
  - Completely automatic in most cases; semi-automatic in some applications
  - Usually restricted to finite-state systems
    - Automatic procedures for infinite-state systems has been developed using special techniques and/or suitable abstractions
  - A push-button procedure: almost everybody can use it
- Some attempts have been done to combine both approaches

Further Reading and Final Remarks

- Except for the last three topics the rest was based on Appendix B of Holzmann's book "The Spin Model Checker"