DISTRIBUTED LTL MODEL-CHECKING

Luboš Brim

brim@fi.muni.cz

Parallel and Distributed Systems Laboratory

Department of Computer Science
Faculty of Informatics

Masaryk University Brno

Overview of the Talk

- About CRCIM and **ParaDiSe**
- Distributed LTL Model-Checking
  - Dependency Structure
  - Negative Cycles
  - LTL Properties
- Verification Tool
- Other Work

ParaDiSe

- CRCIM – Czech Research Consortium for Informatics and Mathematics

  Members: Charles University Prague, Masaryk University Brno
  Institute of Informatics, Prague
  Institute of Information Theory and Automation, Prague

- Parallel and Distributed Systems Laboratory – ParaDiSe

  `www.fi.muni.cz/paradise`

- Research in ParaDiSe organized under themes:

  Algorithms and Tools
  for Practical Verification of Concurrent Systems

ParaDiSe

- Staff
  - 4 permanent members
    Luboš Brim, Ivana Černá, Mojmír Křetínský, Antonín Kučera
  - 9 PhD students
  - 12–15 undergraduate students

- Funding
  Faculty of Informatics, Government grant, Grant Agency grants
  no industrial support

Explicit-State LTL Model-Checking

- Emptiness problem for Büchi automata
- Searching for accepting cycles in the graph
- Nested DFS – linear algorithm
- Cycles are recognized using DFS postorder
- Postorder problem is P-complete
- LTL Model-Checking is not in NC $\Rightarrow$ difficult to parallelize in theory
- Is it possible to solve the problem on real-life cases ?

It seems that YES !!

Distributed LTL Model-Checking

- Cluster of Workstations (no shared memory)
- On-the-Fly
- Explicit-state (enumerative)

How to Detect Cycles in Parallel

- Easy for cycles placed on one workstation
- More difficult for cycles splitted among workstations

Distributed LTL Model-Checking

Three approaches to detecting cycles:

- Ensure the postorder
- Do not use DFS
- Employ particular knowledge about the problem

Maintaining the DFS Postorder

- Second DFS must be started from the accepting states in the postorder defined by the primary DFS
- The order of accepting states is important
- Special data structure (dependency structure) is used to maintain the proper order of accepting states

Maintaining the DFS Postorder

- Dependency structure:
  - Each workstation maintains its own local dependency structure
  - Dynamic – vertices are added and removed
  - Border states and accepting states
  - Edges represent reachability among these states

- Additional memory required:
  ($O(n.r)$ on average, where r is the maximal out-degree and n is the number of states)

- Nested procedures are not performed in parallel

Negative Cycles

- Reduce BA emptiness problem to another one which can be distributed more easily
- Detecting of negative cycles in the SSSP problem
- Given a triple (G, s, l), where $G = (V, E)$ is a directed graph with n vertices and m edges, $l : E \longrightarrow R$ is a length function, and $s \in V$ is the source vertex.
- If there is a negative cycle reachable from s, the graph is not feasible

Negative cycle problem is to decide whether G is feasible.

Negative Cycles

- Negative cycle problem and Büchi automaton emptiness problem:

  A Büchi automaton corresponds to a directed graph G_A.
  Let $G^A = (G_A, s, l)$, where $l : E_A \longrightarrow \{0, -1\}$ is the length function such that $l(u, v) = -1$ iff u is an accepting state.

- Various strategies: walk to root cycle detection strategy

- $\mathcal{O}(\frac{m.n}{p})$, where p is the number of processors

- from $\mathcal{O}(m + n)$ to $\mathcal{O}(mn)$

Property Driven Distribution

- uses the verified property to partition the state space – eliminate division of accepting cycles.
- Büchi automaton which is obtained as a synchronous product of two automata.
- each state has two parts: the one given by the modeled system and the other one given by the negative claim automaton (representing negation of the verified formula).
- use the decomposition of the negative claim automaton into maximal SCCs as a heuristic to partition the state space.

Property Driven Distribution

- Three types of SCCs in the negative claim automaton:
  - F – any cycle within the component contains at least one accepting state
  - P – there is at least one accepting cycle and one non-accepting cycle within the component
  - type N – there is no accepting cycle within the component

- $\boldsymbol{N}$ – reachability

- $\boldsymbol{F}$ – can be detected sequentially without using the nested search and we place each component on a separate workstation

- $\boldsymbol{P}$ – distributed detection

Other Work on Distribution

- Distribution of Branching Logics (CTL, CTL*, AFMC)
- **Distributed Verification Environment – DiVinE**
  - environment for easy implementation of our own distributed verification algorithms on clusters of workstations
  - experimental evaluation and comparison
  - Main characteristics:
    - support for the distributed generation of the state space
    - dynamic load balancing, re-partitioning
    - distributed generation of counter-examples
    - algorithms integration and cooperation

Other Work in ParaDiSe

- YAHODA - The Database of Verification Tools
  - 42 tools
  - http://yahoda.fi.muni.cz
- Verification of IPv6 protocol
- Randomization
- Theoretical Background
  - Exact classification of the decidability/complexity boundaries for existing verification techniques
  - Equivalence-checking and model-checking with various classes of models