Parameterized Verification

Javier Esparza
Technical University of Munich
Many computer systems consist of a number of copies of a (often very simple) template.
A Ripple-Carry Adder
Dijkstra’s Mutual Exclusion Algorithm

The program for the \( i \)th computer \((1 \leq i \leq N)\) is:

```
"integer j;
Li0:  b[i] := false;
Li1:  if k \neq i then
Li2:  begin c[i] := true;
Li3:  if b[k] then k := i;
go to Li1
end
else
Li4:  begin c[i] := false;
    for j := 1 step 1 until N do
        if j \neq i and not c[j] then go to Li1
end;
critical section;
c[i] := true; b[i] := true;
remainder of the cycle in which stopping is allowed;
go to Li0"
```
An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle

DANNY DOLEV, MARIA KLAWE, AND MICHAEL RODEH*

behavior of an active process $v$.

A0. Send the message $\langle 1, \max(v) \rangle$.
A1. If a message $\langle 1, i \rangle$ arrives do as follows:
   1. If $i \neq \max(v)$ then send the message $\langle 2, i \rangle$,
      and assign $i$ to left($v$).
   2. Otherwise, halt—$\max(v)$ is the global maximum.
A2. If a message $\langle 2, j \rangle$ arrives do as follows:
   1. If left($v$) is greater than both $j$ and $\max(v)$
      then assign left ($v$) to $\max(v)$,
      and send the message $\langle 1, \max(v) \rangle$.
   2. Otherwise, become passive.
A Cache-Coherence Protocol
struct DEVICE_EXTENSION {
    int pendingIo;
    bool stoppingFlag;
    bool stoppingEvent;
};

bool stopped;

void main() {
    DEVICE_EXTENSION *e = malloc(sizeof(DEVICE_EXTENSION));
    e->pendingIo = 1;
    e->stoppingFlag = false;
    e->stoppingEvent = false;
    stopped = false;
    async BCSP_PnpStop(e);
    BCSP_PnpAdd(e);
}

int BCSP_IoIncrement(DEVICE_EXTENSION *e) {
    if (e->stoppingFlag)
        return -1;
    atomic {
        e->pendingIo = e->pendingIo + 1;
    }
    return 0;
}

void BCSP_IoDecrement(DEVICE_EXTENSION *e) {
    int pendingIo;
    atomic {
        e->pendingIo = e->pendingIo - 1;
        pendingIo = e->pendingIo;
    }
    if (pendingIo == 0)
        e->stoppingEvent = true;
}

void BCSP_PnpAdd(DEVICE_EXTENSION *e) {
    int status;
    status = BCSP_IoIncrement (e);
    if (status == 0) {
        // do work here
        assert !stopped;
    }
    BCSP_IoDecrement(e);
}

void BCSP_PnpStop(DEVICE_EXTENSION *e) {
    e->stoppingFlag = true;
    BCSP_IoDecrement(e);
    assume e->stoppingEvent;
    // release allocated resources
    stopped = true;
}
A Model of a Biochemical System
Robot swarms, flocks of birds ...
Given:

– a template (a piece of code, an automaton ...), and
– a dangerous location (program line, state ...)

Decide: is there a number $N$ such that $N$ copies of the template can reach a global configuration in which at least one copy has reached the location?

• Conventional model-checking tools can only check instances for particular values of $N$.

Can we prove safety for every $N$?
Identities

• The safety problem is undecidable if the code may use the process identity to organize the communication topology.
  Leader election algorithm by Dolev et al. uses identities
• But many systems do not use identities.
  Dijkstra's algorithm, MESI-protocol, Bluetooth driver, Biochemical systems
• And in others, processes must remain anonymous.
Anonymous Crowds

• Goal: investigate the decidability and complexity of the safety problem for crowds in which
  • every process executes exactly the same code, (anonymous crowds), and
  • the number of processes is unknown to the processes.
Safety Problem for Anonymous Crowds

• **Given:** a finite automaton $A$ and a „dangerous“ state $\ell$ of $A$.
• **Decide:** Is there a number $N$ such that the anonymous crowd $A \parallel A \parallel \cdots \parallel A$ can reach a global state in which at least one of the $N$ copies is at $\ell$?
Verifiers want low complexity
High or Low Complexity?

Verifiers hope for low complexity

„Crowd designers“ (swarm intelligence, population protocols, crowdsourcing) hope for high complexity
The Roman Senators
Global guards

- Process can make a move if the current state of all other processes satisfies some condition

- The safety problem is undecidable
  (Emerson and Kahlon, LICS‘03)
Reliable broadcast
– A process can send a message if its local state satisfies some condition
– All other processes receive the message instantaneously

- The safety problem is decidable
  (E., Finkel, and Mayr, LICS‘99)
- The complexity of the problem (provably) exceeds any primitive recursive function.
  (Schmitz and Schnoebelen, CONCUR’13)
Comm. Mechanisms: Rendez-vous

Rendez-vous
- Synchronous exchange of a message between two processes

- The safety problem is EXPSPACE-complete (if symmetry-breaking mechanism available)
  (Lipton ’76, Rackoff TCS´78)
- Recently: ``Reasonable`` algorithms matching this complexity
  (Bozzelli and Ganty RP´11, Majumdar and Zhang, CONCUR´13)
- Efficient heuristics (E., Ledesma-Garza, Majumdar, Meyer, Niksic CAV´14)
Shared memory with locking
- Processes compete for a lock
- Process owning the lock can perform reads and writes

- Equivalent to the rendez-vous case
- In both cases parametrization still makes the problem "exponentially more complex"
Comm. Mechanisms: Shared Memory, II

Shared memory, no locking
- Concurrent reads and writes allowed, no atomic sections

- The safety problem is NP-complete (with symmetry-breaking mechanism) (E., Ganty, Majumdar CAV‘14)
- Parametrization makes the problem simpler (PSPACE-complete for fixed number of processes)
Open Questions

• Beyond finite automata:
  – What if participants can count?
• Verification of liveness properties
• Heuristics for different application areas
  – Drivers
  – Ad-hoc networks
• Anonymous computing
  – What can be computed by an anonymous crowd?
  – Guerraoui et al, Wattenhofer et al.
Other topics

• Algorithms for probabilistic model-checking
  – Infinite-state Markov Chains
  – Automata-theoretic constructions
• Constraint-based verification
  – Applying SMT-technology to verification problems
• Generic solvers for fixed-point equations
• Verified model-checkers (with Tobias Nipkow)
• Automatic synthesis
• Distributed negotiations

Funded by PUMA and 3 DFG projects