Abstraction of parameterised systems: some remarks

with Ahmed BOUAJJANI (Liafa), Agathe MERCERON (de Vinci),
Tomas VOJNAR (TU Brno)

Peter Habermehl (LIAFA)

15 march 2004
Introduction

- Parameterised systems:
  - Systems with a parameterised number $n$ of components
  - All components are identical
    * plus perhaps a finite number of additional different components
  - Typical examples
    * Mutual exclusion algorithms
    * TTP protocol

- Verification: System is correct for all $n$.
  - Properties: for example safety, global liveness, individual liveness
  - Undecidable in general (even if individual processes are finite-state)
  - Several approaches: Induction, Network Invariants, invisible invariants, abstraction, etc.
Abstraction

- preserves in general safety properties
- Abstraction to a finite-state system
- Abstraction to a more powerful model which can be treated by symbolic methods
- allows to use results for infinite-state model checking
- Abstraction step by step
- Tool supported
Example: Bakery Algorithm

Original definition by Lamport 1974:

integer array choosing[1..n], ticket[1..n]
BEGIN integer j;
    I: choosing[i] := 1;
        ticket[j] := 1 + maximum (ticket[1],...,ticket[n]);
        choosing[i] := 0;
    W: FOR j = 1 step 1 UNTIL n DO BEGIN
        L2: IF choosing[j] != 0 THEN GOTO L2;
        L3: IF ticket[j] != 0 and (ticket[j],j) < (ticket[i],i)
            THEN GOTO L3;
    END;
    C: critical section;
        ticket[i] := 0; noncritical section; goto I;
END

Property: Mutual exclusion (At most one process in critical section)
Bakery Algorithm: Abstraction

Idea: Order processes according to ticket numbers and forget their identity.

- Configuration: String over $\Sigma = \{I, W, C\}$, for example $ICWW$

- Init: $I^+ = \{I, II, III, IIII, \ldots\}$.

- Transitions: Rewrite rules or transducers
  - $xIy \rightarrow xyW$ with $x, y \in \Sigma^*$
  - $xWy \rightarrow xCy$ provided that $x \in I^*$
  - $xCy \rightarrow Ixy$ with $x, y \in \Sigma^*$
  - Example: $III \Rightarrow IIIW \Rightarrow IIWW \Rightarrow IICW \Rightarrow ICWW \Rightarrow \ldots$

- Then use regular model checking for verifying $Post^*(Init) \cap \Sigma^*C\Sigma^*C\Sigma^* = \emptyset$.

Problem: Justify the abstraction
How to formalise the abstraction?

- We need a formal model to define the concrete and the abstract systems.

- We need a logic to describe
  - the transition relation of the concrete system
    - we need some arithmetic (at least comparison)
  - the transition relation of the abstract system
  - the abstraction function (relation)
Formal model

- Global variables

- One process is modelled as an extended automata
  - state type containing local variables
    - infinite domain variables (integer, parameterised integer, etc.)
    - finite domain variables (booleans, control state)
    - parameterised arrays

- To model the collection of processes, we can use an array \( \text{process}[1..n] \) of state considered as a global variable

- Transitions \( \text{trans}(\text{process}, \text{process}') \) are modelled by quantifying over indices
  - asynchronous: existential quantification
    \[
    \exists i. \text{trans}_i(\text{process}[i], \text{process}'[i]) \land \forall j \neq i. \text{process}[j] = \text{process}'[j]
    \]
  - synchronous: universal quantification
Special case

All processes are finite-state, global variables are $1..n$

- An array $\text{process}[1..n]:\text{finite type}$ can be coded in the decidable logic WS1S (or as a string over $\text{finite type}$)
  - Transitions (with limited arithmetic) are also coded in WS1S (or as a transducer)
  - Dedicated tools can be used
    * Regular Model checking tools, abstraction (PAX), etc.
    * even verification of liveness properties possible
Different types of abstraction

- Classical predicate abstraction for each process
  - Can not take into account dependencies between different processes

- Abstract \( n \) local variables into a finite number of globals

- Counter abstraction:
  - works for \( \text{process}[1..n] : \text{finite type} \)
  - introduces a counter for each finite value
  - counts how many processes have each value
  - forgets identity of processes
  - Correctness by construction

- for Bakery: replacing \( \text{ticket}[1..n] \) by another array (forget process identities)
Construction of an abstract system from a concrete one

- concrete transition relation: $\text{trans}(c, c')$

- abstraction relation: $\alpha(c, a)$

- the abstract transition relation $\text{trans}(a, a')$ is given as:

  $$\exists c, c'. \alpha(c, a) \land \alpha(c', a') \land \text{trans}(c, c')$$

- To get $\text{trans}(a, a')$ in a usable way one has to eliminate the quantifiers over arrays
Proving that an abstraction is correct

- concrete transition relation: \( \text{trans}(c, c') \)
- abstraction relation: \( \alpha(c, a) \)
- abstract transition relation: \( \text{trans}(a, a') \)

Proving that abstraction is correct:

\[
\forall c, c', a, a' : \text{trans}(c, c') \land \alpha(c, a) \land \alpha(c', a') \implies \text{trans}(a, a')
\]

- one can show that the negation is not satisfiable
- A little “easier” to use
- Difficulty depends on formulae for \( \alpha \) and \( \text{trans} \) which can contain quantifiers themselves
Logics with arrays

• Special case: array $a[1..n]$ of finite type

• Undecidable for arrays $a[1..n]$ of $[1..n]$
  - One can code computations of a 2-counter machine
    $\exists n \exists a[1..n] \forall i. \text{init}(a[0]) \land \text{step}(a[i], a[i + 1]) \land a[n].\text{state} = \text{halt}$

• Nevertheless, there are some decidable fragments based on cutoff techniques (Pnueli)

• Not immediately applicable for the formula showing correctness of the abstraction for the Bakery algorithm

• Weaken formula: replace $\exists a[1..m], k. \phi(a[k], k)$ by $\exists ak, k. \phi(ak, k)$
Conclusion

- Automation of abstraction for parameterised systems is difficult
- Even showing that abstraction is correct is difficult
- Still some hope to isolate decidable fragments of array logic or use other automatic techniques
- One can use theorem provers
- have some collection of different abstractions