SYNCHRONOUS PRODUCT GENERATION FOR CONTROLLER OPTIMIZATION

Vince Molnár, András Vörös
Budapest University Of Technology And Economics
Fault Tolerant Systems Research Group
• Authors come from *model checking*
  – Qualitative verification of large models
  – Special techniques to handle complex systems

• Similar problems in controller optimization
  – Similarities between controlled systems and LTL model checking
  – Quantitative aspects appear

Can our techniques be used here?
OUTLINE

• Motivation
• Symbolic techniques
  – Decision diagrams
  – Saturation
• Synchronous product generation
  – Symbolic encoding
  – Using saturation

• Possible applications
  – Reachability
  – Cheapest path
• Other useful model checking techniques
  – Temporal logics
  – Cheapest (counter)example
• **Motivation**

• **Symbolic techniques**
  – Decision diagrams
  – Saturation

• **Synchronous product generation**
  – Symbolic encoding
  – Using saturation

• **Possible applications**
  – Reachability
  – Cheapest path

• **Other useful model checking techniques**
  – Temporal logics
  – Cheapest (counter)example
MOTIVATION

Controller specification → Controller synthesis → Controller

Process → Product generation

Often gets very complicated

Optimization → Synchronous product
MOTIVATION

Controller specification → Controller synthesis → Controller

Process → Save space by on-the-fly generation → Product generation

Optimization → Synchronous product
• Motivation
• **Symbolic techniques**
  – Decision diagrams
  – Saturation
• **Synchronous product generation**
  – Symbolic encoding
  – Using saturation

• Possible applications
  – Reachability
  – Cheapest path
• **Other useful model checking techniques**
  – Temporal logics
  – Cheapest (counter)example
MULTI-VALUED DECISION DIAGRAM

- Derived from decision trees
  - variables are ordered into levels

Decision tree

terminal constants

(0,1,1)→1
MULTI-VALUED DECISION DIAGRAM

- Derived from decision trees
  - variables are ordered into levels
- Special reduction rules
  - in a bottom-up fashion, applying reduction from level-to-levels
- Compact representation of multi valued functions
**SYMBOLOC ALGORITHM**

- **Symbolic encoding instead of explicit state representation**
  - Decomposition is needed
- **Saturation uses componentwise encoding**

System Model

- Component 1
- Component 2
- Component 3
SPECIAL ITERATION

- Uses the primarily defined order of the decision diagram variable encoding
- Local exploration in a greedy manner
  - Efficiently exploits locality of systems
- Exploring global synchronization events if needed

Saturation explores and generates the state space representation by a node-to-node transitive closure algorithm.
• **Motivation**

• **Symbolic techniques**
  – Decision diagrams
  – Saturation

• **Synchronous product generation**
  – Symbolic encoding
  – Using saturation

• **Possible applications**
  – Reachability
  – Cheapest path

• **Other useful model checking techniques**
  – Temporal logics
  – Cheapest (counter)example
• **Controlled system is given as high-level model**
  – Components, events
  – Labeled states

• **Controller can be specified as:**
  – Temporal logic formula
  – Büchi automaton
    • Reading labels of states
• **Controlled system is given as high-level model**
  - Components, events
  - Labeled states

• **Controller can be specified as:**
  - Temporal logic formula
  - Büchi automaton
    • Reading labels of states

Accepted language: Any word with infinitely many 0’s
SYMBOLIC ENCODING

How to map components to decision diagram variables?
How to encode transitions?
How to do these efficiently?
How to map components to decision diagram variables?

- Each component gets its own variable
- The controller automaton gets an additional one
- Variable ordering?
  - Controller goes to the bottom level
  - Locality can be preserved
How to encode transitions?

• **Synchronization is required:**
  – Controller must act based on step of controlled system
  – Locality cannot be preserved
    • Labels of every local state necessary
    • Local transitions does not know unaffected states
    • Local $\rightarrow$ global
    • Saturation is degraded

• **Solution using constrained saturation**
  – Omit synchronization by system state
    • Transitions can be freely combined
  – *Constrain* target states
    • To legal states
How to do these efficiently?

- **Constraint for target states**
  - Prevent transitions that lead to illegal states

- **Legal states?**

![Tableau automaton](image)
How to do these efficiently?

- **Constraint for target states**
  - Prevent transitions that lead to illegal states

- **Legal states:**
  - Input condition of automaton state is satisfied by system state
  - Can be precomputed based on the automaton
  - Does not affect locality

- **Constrained saturation:**
  - Explores the state space of the product directly
  - Constrains explored states to legal states
  - Legal states can be characterized without enumerating them
    - By evaluating them on the fly
OUTLINE

- Motivation
- Symbolic techniques
  - Decision diagrams
  - Saturation
- Synchronous product generation
  - Symbolic encoding
  - Using saturation

- Possible applications
  - Reachability
  - Cheapest path
- Other useful model checking techniques
  - Temporal logics
  - Cheapest (counter)example
POSSIBLE APPLICATIONS

- **Reachability**
  - Does the controller allow the system to **reach a desired** state?
  - Does it force the system to **avoid bad states**?

- **Shortest path**
  - What is the cheapest way to reach a state?
  - What is the cheapest way to fire an event in the high-level model?
  - Cheap in terms of:
    - Number of used transitions
    - Controller states passed
    - Controller transitions

- **Feasability of controller sequences**
  - Can the controller perform a sequence of actions in the system?
  - What is the cheapest way to do this?
• Motivation
• Symbolic techniques
  – Decision diagrams
  – Saturation
• Synchronous product generation
  – Symbolic encoding
  – Using saturation
• Possible applications
  – Reachability
  – Cheapest path
• Other useful model checking techniques
  – Temporal logics
  – Cheapest (counter)example
• **Temporal properties**
  – Describe temporal behavior with logic
  – E.g.: Signal alarm whenever a fault is detected

• **Model checking**
  – Exhaustive analysis of possible behaviors regarding the temporal property
  – **Counterexample** if property does not hold, or **example** where property holds
    • Shortest
    • Cheapest

• **Possible applications:**
  – Does the controller let the system perform a desired behavior?
  – Does the controller force the system to avoid a bad behavior?
  – What is the cheapest way to perform a certain behavior?
SUMMARY

• Complex systems can have very large state spaces
  – Symbolic techniques can efficiently handle this

• On-the-fly generation of synchronous product
  – Controller and controlled system together

• Efficient algorithm using saturation
  – Constrained saturation allows legal states only

• Possible applications in optimization
  – Qualitative and quantitative analysis
TÁMOP-4.2.2.C-11/1/KONV-2012-0004
National Research Center for Development and Market Introduction of Advanced Information and Communication Technologies

THANK YOU FOR YOUR ATTENTION!