PROPERTY-PRESERVING GENERATION OF TAILORED BENCHMARK PETRI NETS

BERNHARD STEFFEN, MARC JASPER (TU DORTMUND)
JACO VAN DE POL, JEROEN MEIJER (U TWENTE)
PROPERTY PRESERVING GENERATION

- Benchmarks, Challenges, and Competitions
  - Motivations and Examples
  - RERS-challenge design

- Generation of Sequential Programs

- Generation of Parallel Programs / Petri-Nets

- Solution Techniques

- Conclusion and Future Directions
IN PRAISE OF CHALLENGES AND COMPETITIONS
COMPARISON: OPEN PROBLEMS IN THEORY

- **Challenges:**
  - Hard (scalable) problems for the community to solve
  - Trigger measurable progress in the field
  - Award: achievements

- **Competition:**
  - A match (high-profile event), to appeal to the human nature
  - Impact: fair comparisons & standardisation of benchmarks
  - Award: prizes

- Which form is better to advance the scientific field?
  - Progress and collaboration happen in both cases
  - Motivation to participate (beginners, experts)
  - Variation is good!
EXAMPLES OF VERIFICATION COMPETITIONS
MODEL CHECKING ENTERED LATE – ALL COMPETITIONS ARE STILL ALIVE

  - We all recognize the enormous lump in SAT/SMT solver capabilities

- Termination Competition (2004-)
  - Enormous growth in solutions for an undecidable problem
  - Multiple categories (from term rewrite systems to programs)
  - Quality: certificates in the form of machine-checkable proofs

- SV-COMP (2012-, TACAS).................real code, scientific measurement
- VerifyThis (2012-) .................program modeling & verification, jury based
- MCC (2011-).................................Petri-Nets, relative tool confidence
- RERS (2012-)........................................................topic of today
RERS CHALLENGE (2012-2017) - REQUIREMENTS
MAIN ORGANISER: BERNHARD STEFFEN

- RERS = Rigorous Examination of Reactive Systems
  - CAS – condition-action-systems
  - PLC – programmable-logic-controllers

- Goals:
  - Off-line, solution based, free-style, favours hybrid approaches
  - Several benchmark formats to serve multiple communities
  - Scalable: provides multiple dimensions and complexities
  - Award structure mixes “challenge” and “competition” elements

- Technical novelty:
  - Solutions are known due to intricate benchmark synthesis
RERS EVOLUTION

- **2012**: Three dimensions
  - Reachability & LTL properties
  - Enumerated & Arithmetic data
  - Small – Medium – Large
- **2013**: Whitebox – Greybox – Blackbox models + array data structures
- **2014**: Distinguish 3 rewards: Achievements, Numeric score, Qualitative
- **2015**: + Runtime Verification problems (based on trace files)
- **2016**: + Parallel Problems (first pilot)
- **2017**: + Parallel Problems (Petri Nets, Promela)

- **Communities involved**: model checking, static analysis, runtime verification, active and passive automata learning, model-based testing
PROPERTY PRESERVING GENERATION

OUTLINE: CASE FOR SEQUENTIAL PROGRAMS

- Benchmarks, Challenges and Competitions

- Generation of Sequential Programs
  - LTL Synthesis
  - Property-Oriented Expansion
  - Model Checking
  - SAT Solving
  - Code Motion

- Generation of Parallel Programs / Petri-Nets

- Solution Techniques

- Conclusion and Future Directions

UNIVERSITY OF TWENTE.
**GENERATION = PROPERTY-PRESERVING TRANSFORMATION**

THE SCENARIO

![Diagram showing the scenario with the following components: Organizer, Benchmark Profile, Benchmark Generator, Solution, Evaluator, Participants, C Code, Promela Code, Java Code, Questionnaire. The design goal is highlighted as: the organiser can participate!]

**Design goal:**
the organiser can participate!
Benchmark Generator
Benchmark Generator (LTL patterns → Büchi → Mealy)

G(¬→F *)
G(¬U *)

LTL patterns

Manual or Random

UNIVERSITY OF TWENTE.

Property-Perserving Generation, Bernhard Steffen, Jaco van de Pol

Zaragoza
29/06/2017
Benchmark Generator (LTL patterns → Büchi → Mealy)
Benchmark Generator (LTL patterns $\rightarrow$ Büchi $\rightarrow$ Mealy)

- **G(*$\rightarrow$F*)**
- **G(!* U *)**

**Pattern-Based LTL generation**

- **G(a$\rightarrow$F b)**
- **G(!b U a)**

**Büchi Synthesis**

**Classical Synthesis**

(LTL2BA, SPOT)

https://spot.lrde.epita.fr/
Benchmark Generator (LTL patterns $\rightarrow$ Büchi $\rightarrow$ Mealy)

- Benchmark Profile
- LTL patterns: $G(\neg \rightarrow F \neg)$ $G(\neg U \neg)$
- Pattern-Based LTL generation
- LTL formulas: $G(a \rightarrow F b)$ $G(\neg b U a)$
- Büchi Synthesis
- Büchi automaton
- Automatic Büchi to Mealy
- Mealy machine

Solution

Code

Questionnaire
Benchmark Generator (Properties and Solutions)

Pattern-Based LTL generation
G(¬→F *)
G(! U *)

LTL patterns

G(¬→F b)
G(!b U a)

LTL formulas

Büchi Synthesis

Büchi automaton

Büchi to Mealy

Mealy machine

Property Profile Completion

Solution

Code

Questionnaire
Benchmark Generator (Properties and Solutions)

- **Benchmark Profile**
- **LTL patterns**: $G(\neg \rightarrow F \neg)$, $G(\neg ! U \neg)$
- **Pattern-Based LTL generation**
- **LTL formulas**: $G(a \rightarrow F b)$, $G(\neg ! b U a)$
- **Büchi Synthesis**
- **Büchi automaton**
- **Model Checking**
- **Büchi to Mealy**
- **Mealy machine**
- **Property Profile Completion**
- **Solution**
- **Code**
- **Questionnaire**
Benchmark Generator (Properties and Solutions)

Property Specification

G(¬→F*)  
G(! U *)

LTL patterns

G(a→F b)  
G(!b U a)

LTL formulas

Pattern-Based LTL generation

Büchi Synthesis

Büchi automaton

Büchi to Mealy

Mealy machine

Property Profile Completion

Solution

Benchmark Profile

Code

(false R (! (iC & ! oY) | (! oY WU (oX & ! oY))))
output X occurs between input C and output Y

(false R (! iE | (true U oY)))
output Y responds to input E

(! iB WU (oX & ! iB))
output X occurs before input B

(false R (! ((oZ & ! iA) & (true U iA)) | (! oW U iA)))
output W does never occur between output Z and input A

Property-Perserving Generation, Bernhard Steffen, Jaco van de Pol
Zaragoza 29/06/2017
Benchmark Generator (Properties and Solutions)

**Benchmark Profile**

- **G(*→F*)**
- **G(!* U *)**

**LTL patterns**

**G(a→F b)**

**G(!b U a)**

**LTL formulas**

**Büchi Synthesis**

**Büchi automaton**

**Büchi to Mealy**

**Mealy machine**

**Solution**

**Specification**

1. 
   
   \[(\text{false } R \ (\neg (\neg \text{iC} \land \neg \text{oY}) \lor (\neg \text{oY} U (\text{oX} \land \neg \text{oY})))\]
   
   Output X occurs between input C and output Y
   
   Used for synthesis: no
   
   Satisfied: no

2. 
   
   \[(\text{false } R \ (\neg \text{iE} \lor (\text{true } U \text{oY})))\]
   
   Output Y responds to input E
   
   Used for synthesis: yes (positive)
   
   Satisfied: yes

3. 
   
   \[(\neg \text{iB} U (\text{oX} \land \neg \text{iB}))\]
   
   Output X occurs before input B
   
   Used for synthesis: no
   
   Satisfied: no

4. 
   
   \[(\text{false } R \ (\neg ((\text{oZ} \land \neg \text{iA}) \land (\text{true } U \text{iA}))) \lor (\neg \text{oW} U \text{iA})))\]
   
   Output W does never occur between output Z and input A
   
   Used for synthesis: no
   
   Satisfied: yes
Benchmark Generator (Property Oriented Expansion)

PoE: Manual or Random

- Pattern-Based LTL generation
- Büchi Synthesis
- Büchi to Mealy
- Property Profile Completion
- Mealy Machine Expansion

LTL patterns
- G(¬→F *)
- G(!* U *)

LTL formulas
- G(a→F b)
- G(!b U a)

Büchi automaton

Solution

Code

Questionnaire
Property-Oriented-Expansion with Random 1/0 Properties

- Add random property to edges
- Distinguish states based on this extra hidden state
Property-Oriented-Expansion with Random 1/0 Properties

- Add random property to edges
- Distinguish states based on this extra hidden state
Property-Oriented-Expansion with Random 1/0 Properties

- Add random property to edges
- Distinguish states based on this extra hidden state
Property-Oriented-Expansion with Random 1/0 Properties

- So far: “equivalent state space”
- Now remove arbitrary edges
Benchmark Generator (Program Construction)

Pattern-Based LTL generation

G(\* -> F \*)
G(\!* U \*)

LTL patterns

G(a -> F b)
G(\!b U a)

LTL formulas

Büchi Synthesis

Büchi automaton

Büchi to Mealy

Property Profile Completion

Mealy machine

Mealy Machine Expansion

(initial) code model

discrimination tree

Program Model Construction

Solution

Questionnaire

Code

Benchmark Profile
Program Construction

Input:
1) The expanded Mealy Machine
2) A randomly generated Decision Tree
Program Construction

Random Construction: Leave/Node Association
Random Construction: Transition Selection
Program Construction

SAT-Solving for Transition Realisation

Hoare Triples guarantee correct transitions: Find $S$ such that

$$\{p_1 \land p_2 \land \neg p_4 \land \neg p_9\} S \{ (p_1 \land p_2 \land p_4 \land \neg p_8) \mid (p_1 \land \neg p_2 \land \neg p_5 \land p_{11}) \}$$
Benchmark Generator (final code generation)
Benchmark Generator (final code generation)
Benchmark Generator (final code generation)
Code Motion: avoid that conditions can be analysed locally

Busy Code Motion: Maximal Semantics-Preserving Movement

Lazy Code Motion: Minimal Semantics-Preserving Movement, with Computational Optimality
Benchmark Generator (final code generation)

- Benchmark Profile
- G(*→F*)
- G(!* U *)
- LTL patterns
- G(a→F b)
- G(!b U a)
- LTL formulas
- Büchi Synthesis
- Büchi automaton
- Büchi to Mealy
- Program Model Elaboration
- (initial) code model
- Program Model Construction
- Discrimination tree
- Mealy machine
- Mealy Machine Expansion
- Property Profile Completion
- Solution
void calculate_output(int input) {
    cf = (1);
    errorCheck();
    if((( 527 < a1183 && ((a1617==34) || !(a1490==33))) && ( a1198 <= -81 || ((a1362!=1) && ((a1299==1) || ((cf==1) && (a98==33))))))){
        ...
        if(((!(a1963==a1686[2]) || ((a1447==32) || (!a1404==33) && ((a80==33) && (cf==1))))))
            ...
        }
        ...
        if((!(a1963==a1686[2]) || ((a1447==32) || (!a1404==33) && ((a80==33) && (cf==1))))))
            ...
        }
        ...
    }
    ...
}

int main(int argc, char* argv[]) {
    int input;
    while (true) {
        scanf("%d", &input);
        calculate_output(input);
    }
}
PROPERTY PRESERVING GENERATION
OUTLINE: CASE FOR PARALLEL PROGRAMS

- Benchmarks, Challenges and Competitions
- Generation of Sequential Programs
- Generation of Parallel Programs / Petri-Nets
  - Modal Transition System Specifications
  - Parallel Decomposition of Benchmarks
  - Hardness Guarantees
  - Alphabet Extension
  - Petrinet Generation

- Solution Techniques
- Future Directions

Zaragoza
29/06/2017
SCENARIO PARALLEL BENCHMARK GENERATION
FIRST APPROACH FOR LTL
Round Robin Scheduling for 2 tasks

Parallel Composition with synchronisation on common action labels
Round Robin Scheduling for 3 tasks
Round Robin Scheduling for 3 tasks
Parallel Decomposition of Benchmark Scenarios

- Should be **correct by construction**
- Should be **hard by construction**

Inverse **assume-guarantee** style for compositional reasoning

- Formalised as **Modal Transition Systems with weak refinement**
MTS: an LTS with \textit{may} (\(\text{---}\rightarrow\)) and \textit{must} (\(\rightarrow\)) transitions

- Every \textit{must} is also a \textit{may} (\(\rightarrow \subseteq \text{---}\rightarrow\))

\(M_1 \preceq M_2:\) \(M_1\) refines \(M_2\)

- \(M_1\) allows fewer \textit{may}-transitions than \(M_2\)
- \(M_1\) requires more \textit{must}-transitions than \(M_2\)

Distinguish \textit{parallel composition} and \textit{conjunction} of specifications:

- Both synchronize on common action labels

\[
\begin{align*}
\text{If } M_1 \preceq M_2 \text{ then } M_1 \parallel M \preceq M_2 \parallel M \quad (\text{---}\rightarrow \parallel \text{---}\rightarrow = \text{---}\rightarrow) \\
\text{If } M \preceq M_1 \text{ and } M \preceq M_2 \text{ then } M \preceq M_1 \land M_2 \quad (\text{---}\rightarrow \land \text{---}\rightarrow = \text{---}\rightarrow)
\end{align*}
\]
REPEAT THE FOLLOWING 4 STEP PROCEDURE

- Step 1: define parallel composition of Modal Transition Systems
- Step 2: select a Modal Contract $I$
- Step 3: compute the next system $M_s$
- Step 4: compute the next context $M_c$
STEP 1: PARALLEL COMPOSITION OF MTS
SYNCHRONISE ON LABELS, AGREE ON MUST
STEP 2: SELECT A MODAL CONTRACT
BY ADDING GREEN AND RED TRANSITIONS

Choose some assumptions on the context randomly
Under these assumptions, we can relax the system

- **Green**: select some must transitions (provided by context)
- **Red**: add some new transitions (forbidden for context)
STEP 3: COMPUTE THE NEXT SYSTEM $M_s$
RISKING THE RED TRANSITIONS

Redirect red transitions to a new sink state $r$
STEP 4: COMPUTE THE NEXT CONTEXT

\[ M_c = M_c^g \land M_c^r \]

Green only context: \( M_c^g \)

Red only context: \( M_c^r \)

(complement of forbidden steps)
GUARANTEES ON CORRECTNESS AND HARDNESS

- Correctness: $M_s \parallel (M_c^g \land M_c^r) \preceq M \models \varphi$ (so the properties still hold)

- Hardness 1: $M_s \not\models \varphi$ and $(M_c^g \land M_c^r) \not\models \varphi$ (all components needed)

- Hardness 2: can achieve $n$ parallel components with $\geq 2^n$ states
ALPHABET EXTENSION AND LABEL HIDING
THE CASE FOR WEAK REFINEMENT

- Blow up and obfuscate further by alphabet extension and label hiding

- This only preserves weak refinement ($\equiv$), allowing $\tau$-steps
  - Preserves branching, stutter-free properties
  - Only for convergent MTSs (no $\tau$-loops)

- Steps:
  1. Choose a new, extended alphabet $E$
  2. Choose an extra MTS $M_E$, whose must-steps are deadlock-free
  3. Replace a transition from each cycle in $M_E$ by a sequence of $M$-steps

- Guarantee: $\text{hide}_E(Ms \parallel (M_c^g \land M_c^r) \parallel M_E) \equiv M \vDash \varphi$
GENERATING A CHAIN OF CYCLERS
EXAMPLE: ROBIN MILNER’S ROUND-ROBIN SCHEDULER

CTL: AG AF a
LTL: GF a
ADDING MORE COMPLEXITY
COMPUTING THE RED & GREEN CONTEXTS

New modal contract

Green context:

Red context:
PARALLEL BENCHMARK IN LTS FORMAT
PARALLEL BENCHMARK IN PETRI-NET FORMAT

- Each location in the LTS becomes a place in the Petri-Net
  - Each “initial” place gets a token
- Each combination of transitions induces a Petri-Net transition
  - encoding all subsets of possible synchronisations (blowup!)
PROPERTY PRESERVING GENERATION

OUTLINE: CASE FOR PARALLEL PROGRAMS

- Benchmarks, Challenges and Competitions
- Generation of Sequential Programs
- Generation of Parallel Programs / Petri-Nets

- Solution Techniques (LTSmin)
  - Static Analysis
  - Black-Box Learning
  - Brute force model-checking

- Conclusion and Future Directions

Jaco van de Pol
Theo Ruijs
Steven ten Brinke
Jeroen Meijer

RERS 2012-2017
STTT 16(5), 2014
SOLUTION TECHNIQUES

- Brute-force Model Checking
  - LTSmin high-performance model checking, as a base-line
- Control-flow abstractions, CEGAR
  - Should scale to larger systems
- Static Analysis
  - Checking reachability of code, invariants
  - Reverse-engineer obfuscation: compression
- Blackbox: automata learning L* (LearnLib)
  - Learn models by synthesizing & testing intermediate hypotheses
  - Guide the testing by fuzzing (AFL)
  - Guide the learning by model-checking the hypotheses (Jeroen Meijer)
    - Based on: Peled, Vardi, Yannakakis: Blackbox Checking
BRUTE-FORCE HIGH-PERFORMANCE MODEL CHECKING
LTSMIN AS A BASE-LINE – FLEXIBLE ARCHITECTURE

C-code

Partial-Order Reduction
- Stubborn sets

Promela

Variable Reordering
- bandwidth reduction

PNML

LTL model checker
- multi-core NDFS
- multi-core SCCs

Mu-calculus checker
- symbolic algorithms
- multi-core BDDs

API

API

C-code

Partial-Order Reduction
- Stubborn sets

Promela

Variable Reordering
- bandwidth reduction

PNML

LTL model checker
- multi-core NDFS
- multi-core SCCs

Mu-calculus checker
- symbolic algorithms
- multi-core BDDs

API
## RESULTS

**LTSMIN – REACHABLE STATE SPACE**

2016, parallel problems

<table>
<thead>
<tr>
<th>P</th>
<th>Number of States</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>10, ~1.00e+01</td>
</tr>
<tr>
<td>104</td>
<td>6222, ~6.22e+03</td>
</tr>
<tr>
<td>107</td>
<td>13981200, ~1.40e+07</td>
</tr>
<tr>
<td>110</td>
<td>9298708627, ~9.30e+09</td>
</tr>
<tr>
<td>113</td>
<td>72476206296, ~7.25e+10</td>
</tr>
<tr>
<td>116</td>
<td>74706813725952, ~7.47e+13</td>
</tr>
<tr>
<td>119</td>
<td>4921222304004403200, ~4.92e+18</td>
</tr>
<tr>
<td>120</td>
<td>192832843687901134848, ~1.93e+20</td>
</tr>
</tbody>
</table>

2017: Milner’s round-robin scheduler

<table>
<thead>
<tr>
<th>structure</th>
<th>State space</th>
<th>CTL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>peak</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>(6.55 \cdot 10^4)</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>(4.29 \cdot 10^9)</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>(1.84 \cdot 10^{19})</td>
</tr>
<tr>
<td>64</td>
<td>64</td>
<td>(3.40 \cdot 10^{38})</td>
</tr>
<tr>
<td>128</td>
<td>128</td>
<td>(1.15 \cdot 10^{77})</td>
</tr>
</tbody>
</table>
PROPERTY PRESERVING GENERATION
OUTLINE: CASE FOR PARALLEL PROGRAMS

- Benchmarks, Challenges and Competitions
- Generation of Sequential Programs
- Generation of Parallel Programs / Petri-Nets

- Solution Techniques

- Conclusion and Future Directions
CONCLUSION

- Generation of **correct** and **controllably hard** problems by construction
  - The complexity of the problems can be tuned
  - The solutions to the properties are known beforehand
  - In principle, even the organiser can participate in the challenge

- **Sequential case:**
  - We export the models to C-code and Java-code

- **Parallel case:**
  - We export the models to Petri-Nets and Promela
FUTURE WORK

- Expand on the properties
  - CTL – weak refinement preserves branching properties
  - Add time, probabilities, …

- Extend this to real parallel code
  - Goal: generate parallel C-code based on OpenMP
  - Focus on the detection of race conditions

- Increase the scientific understanding of verification algorithms
  - Controlled variation along various dimensions of complexity
  - Learn how complexity axes relate to effective algorithmic techniques