Incremental and Parallel Model Checking of Synchronous Systems

Temesghen Kahsai and Cesare Tinelli
The University of Iowa
Project Overview

- AFOSR Supported Research (#AF9550-09-1-0517)

- Collaborations
  - NYU (project partner)
  - ONERA, France (research collaborator)
  - Rockwell-Collins (power user)

- Overall Objective
  - Improve performance and scalability of techniques for verifying mission-critical embedded software
Project Overview

- **Scientific Approach**
  - Focus on verifying *functional properties of synchronous systems*
  - Develop *automated reasoning* techniques and tools based on a more powerful logic than propositional logic

- **Opportunities**
  - Use *Satisfiability Modulo Theories (SMT)* instead of SAT
  - Exploit recent *dramatic advances* in SMT technology
  - Reason about *finite*–as well as *infinite-state* systems
Our General Approach

- **SMT-based** model checking:
  - Automatically translate a system $S$ and property $P$ into a **first-order** logic with **built-in theories**
  - Try to prove or disprove $P$ **automatically** with an **inductive** model checker
  - Use an **SMT solver** as the main reasoning engine
  - Verify **control** and **data** properties
Main Focus

- Consider reactive systems specifiable with synchronous dataflow languages

- Use SMT-based $k$-induction to verify invariant functional properties of transition systems

- For experimental evaluations, work with systems written in Lustre
Previous Work (S5 2010)

- An SMT-based model checker for Lustre programs
- Enhancements to induction method (path compression, abstraction, invariant generation)
- Initial results on invariant generation
Current Developments

A novel incremental and parallel SMT-based model checker

- parallel temporal induction architecture based on message passing

- simultaneous, incremental verification of multiple properties

- incremental automatic invariant generation
Outline

- Temporal induction, Lustre
- KIND: an SMT-based model checker
- Automatic invariant generation
- Incremental and parallel model checking
PRELIMINARIES
Properties over System States

- **Invariant**: satisfied by all reachable states of a system $S$

- **$k$-inductive**: preserved by $k$-transitions sequences
  - $I(s_0), T(s_0, s_1), ..., T(s_{k-1}, s_k) \Rightarrow P(s_0), ..., P(s_k)$
  - $T(s_n, s_{n+1}), ..., T(s_{n+k}, s_{n+k+1}), P(s_n), ..., P(s_{n+k}) \Rightarrow P(s_{n+k+1})$

- **Inductive**: preserved by single transitions
  - $I(s_0) \Rightarrow P(s_0)$
  - $P(s_n), T(s_n, s_{n+1}) \Rightarrow P(s_{n+1})$
Lustre

- **Declarative** and **deterministic** specification/programming language for model-based development
- Used to model control systems
- Lustre programs = systems of **equational constraints between** input and output **streams**
Lustre

- Declarative and deterministic specification/programming language for model-based development
- Used to model control systems
- Lustre programs = systems of equational constraints between input and output streams

From Lustre to SMT

- Stream constraints can be reduced to Boolean and arithmetic constraints over system states
- SMT solvers can process this sort of constraints natively and efficiently
Our Model Checker: KIND (2010)

- Translation of
  - a Lustre program \( L \) and
  - a putative invariant property \( P \)
  into set \( F \) of SMT formulas

- SMT-based \( k \)-induction on \( F \) to prove or disprove \( P \) for \( L \)

- Different options:
  - quick mode (bug finder)
  - full mode
  - path compression ...

Wednesday, 15 June 2011
Translation of
- a Lustre program \( L \) and
- a putative invariant property \( P \)
into set \( F \) of SMT formulas

SMT-based \( k \)-induction on \( F \) to prove or disprove \( P \) for \( L \)

End users:
- Rockwell-Collins
- Verimag
- ....
AUTOMATIC INVARIANT GENERATION
Improving Accuracy: Invariant Discovery

Problem

\( k \)-induction may fail to show that an invariant property \( P \) is indeed invariant

Observation

This happens only when \( T(s, s') \) holds for some unreachable state \( s \)

Possible Solution

1. Find some other invariant \( \text{Inv}(s) \) for \( S \)
2. Strengthen \( T(s, s') \) into \( T'(s, s') = T(s, s') \land \text{Inv}(s') \)
A Novel Invariant Discovery Method


- “Smart brute force” approach
- Discover invariants by sifting through a large set $C$ of automatically generated formulas
- Relies on efficiency of SMT engine
- A compact representation and efficient sifting of $C$ in selected cases
A Novel Invariant Discovery Method


- Automatic invariant generator for Lustre programs, KIND-INV
KIND-INV Architecture

Lustre program

```luster
node therm_control (...)
var ...
let ....
tel
```

Options:

- `bool`
- `int`
- `no_trivial_invariants`
- `no_redundant_edges`

SMT Solver

```luster
node therm_control (...)
var ...
let ....
assert Inv1 and
Inv2 and ...
Invn
tel
```

or

```luster
node therm_control (...)
var ...
let ....
assert true
tel
```
KIND + KIND-INV: Precision Results

503 Lustre benchmarks with valid property [HT08]

- Without invariants: 309 (309) 194
- With Bool invariants: 386 (386) 117
- With Int invariants: 411 (411) 92
- With Bool+Int invariants: 431 (431) 72
INCREMENTAL PARALLEL KIND
Motivation

The user of sequential KIND has to:

- make an informed guess on whether to use KIND in quick or full mode
  - quick mode: good for invalid properties
  - full mode: needed for valid properties

- may need to first generate invariants offline with KIND-INV and then pass them to KIND
Achievements

In **sequential** KIND :-

- A single inductive invariant at the end of the process

In **parallel** KIND :-

- No need to guess which KIND mode to use
- Invariant generation is done in parallel
- A new incremental invariant generation
Parallel KIND Architecture

- **Base Step Process**
  - Input: Lustre program
  - Output: SMT solver

- **Inductive Step Process**
  - Input: SMT solver
  - Output: SMT solver

- **Invariant generator**
  - Inputs: Int Invariants, Bool Invariants
  - Output: Candidate Generator
  - Output: Invariant generator

- **M1, M2, M3, M4**
  - Arrows indicating data flow and control flow
Experimental Results

- **Sequential KIND (full mode) vs Parallel KIND**
- **Invalid benchmarks**

![Graph showing comparison between Sequential KIND (Full mode) and Parallel KIND]
Experimental Results

- **Sequential KIND (quick mode)** vs **Parallel KIND**
- Invalid benchmarks
Experimental Results

- Incremental vs Non-Incremental invariant generation
- Valid benchmarks provable only with additional invariants
Experimental Results

- Incremental vs Non-Incremental invariant generation
- Valid benchmarks provable without additional invariants
Multi-property Verification

- Typical industrial model checking applications:
  - verification of multiple properties for the same program

- In sequential and parallel KIND:
  - verification of only single property
  - multiple property are treated as a “single” property

\[ P = P_1 \land P_2 \land \cdots \land P_n \]
Multi-property Verification

Essence of multiple properties:

- Several properties may share the “cone of influence”
  - dis(proving) several properties simultaneously yield a significant speedup.

- A conjunct of properties is stronger than individual ones
Incremental Parallel KIND

- Checks the validity of multiple properties simultaneously

- But is able to detect which property are individually valid or invalid
  - depending on their difficulty
  - reporting its findings progressively, as it goes

- Proven properties are immediately used, as additional invariants, to prove remaining ones
SUMMARY & FUTURE WORK
Summary

- A parallel and incremental SMT-based model checking architecture
- Simultaneous verification of multiple properties
- Efficient use of automatic invariant generation
- KIND: a competitive model checker for Lustre programs
Future Work

- Incorporation of new in-house SMT solver, CVC4, into KIND
- Additional invariant discovery techniques in parallel
- Modular verification
- Front-end for more languages besides Lustre
- Certified model checking
Thank you

http://clc.cs.uiowa.edu/