Variants of LTL Query Checking

Hana Chockler
IBM Research

Arie Gurfinkel
SEI

Ofer Strichman
Technion
Problem Formulation - General Query Checking

We have the design:

What is the property?

This is not as silly as you might think!
Problem: Model Understanding

Is it
“always(request -> eventually(grant))”
or
“always(request -> next(grant))”
or
“always((request AND not_busy) -> next(grant))”
or
“always(request -> next(grant AND busy))”
Or maybe something else?
Current mode of working:
Try properties one after another until you find the right ones

Usually, we are looking for the strongest properties that hold in the design

Very time-consuming and frustrating

Wouldn't it be nice if an automated process could find the right property for us?
Query Checking
was defined by W. Chan in 2000

Model Checking:

A mathematical model of the system M (an FSM):

Does M satisfy \( \varphi \)?

- no
- yes
- counter example

I only have a vague idea about the right property to check ...

the system is correct!
Query Checking was defined by W. Chan in 2000

**Query Checking:**

A mathematical model of the system $M$ (an FSM):

A skeleton of a formal specification - basically a formula with placeholders

What is the right $\varphi$?

A strongest $\varphi$ with a given skeleton that holds in $M$

I found the property!
Why this particular setting?

- A skeleton gives an idea about the kind of property the verification engineer has in mind.
- Usually, the skeleton is accompanied by the set of signals, which can be used to turn the skeleton into a property - so no uninteresting signals can appear.

- Model learning
- Model exploration
- Strongest invariant
Related Work

- Definition of query checking for CTL; a subset of CTL for which there is a single solution [Chan]
- Solving query checking with alternating automata [Bruns and Godefroid]
- Solving query checking with lattices [Gurfinkel, Chechik, and Devereux]

All these algorithms use some form of repeated model checking
Preliminaries:

Linear Temporal Logic (LTL)

- In addition to Boolean operators, has temporal operators: always, eventually, next, and until:
  - always(p AND q) - p AND q are true in all states
  - p until q - on each path, p holds until q holds

Buchi Automata

- Automata on infinite computations: accept if a path visits an accepting state an infinite number of times

\[ p \land q \]

A accepts all paths on which p AND q are true in all states:

Systems as labeled state-transition graphs (FSM)

- Each state is labeled with atomic propositions (variables) that are true in this state; all states have outgoing transitions; computations are infinite paths on the graph.

- A system satisfies a property if all its computations satisfy this property.
Preliminaries:

Model Checking:

- Construct an automaton $B$ for the negation of the property $\varphi$
- Build the product $M \times B$
- Check whether it is empty:
  - If it is empty, then $M \models \varphi$
  - If not, accepting paths are counterexamples

$B$ is an automaton for the negation of “eventually($\neg p \lor \neg q$)”

$M \times B$ is empty

$M$ satisfies the property
Our contribution: LTL Query Checking

- Problem formulation: Given an FSM (Kripke structure) $M$ and an LTL query $\varphi[?]$, both over $\Sigma = 2^{AP}$, find a strongest propositional formula $f$ such that $M \models \varphi[?] \leftarrow f$

A mathematical model of the system $M$ (an FSM):

An LTL query $\varphi[?]$ - an LTL formula with placeholders

The set $AP'$ of atomic propositions to be used to construct $f$

What is the right $\varphi$?

A strongest propositional $f$ such that $M \models \varphi[?] \leftarrow f$

I found the property!
Why propositional $f$ and why over $AP'$?

Usually, the type of the property is defined by its temporal operators; then, query checking finds a propositional $f$ that fits - for example, “always(?)” can be used to find a strongest invariant

$AP'$ is a subset of signals over which $f$ is constructed

What is a “strongest“?

**Option 1:** $f$ is stronger than $g$ if $\text{models}(f) \subseteq \text{models}(g)$ (that is, $f \rightarrow g$)

**Option 2:** $f$ is stronger than $g$ if $|\text{models}(f)| < |\text{models}(g)|$

**Option 3:** $f$ is stronger than $g$ if $\varphi[f] \rightarrow \varphi[g]$
Our contribution: LTL Query Checking

♦ We present solutions for all definitions of strongest
♦ The most interesting one is to Option 2:
  ◊ f is stronger than g if |models(f)| < |models(g)|

The solution reduces the query checking problem to an optimization problem in Linear Integer Programming over binary variables (0-1-ILP), or, equivalently, a problem for a Pseudo-Boolean Solver (PBS)
**Intuition:** compute $f$ such that the product of $M$ with the automaton for the negation of $\varphi[f]$ is empty

**Solution strategy:**

- Let $\Sigma' = \Sigma \cup '?' \cup '¬?'$.
- **Construct** $P = M \times B_{\neg \varphi}$ over $\Sigma'$
  - In this product '?' and '¬?' are the same as 'true', i.e. they synchronize on everything.
- Let $\Pi$ be the set of lasso-shaped accepting paths in $P$
- We will find the strongest $f$ that eliminates all elements of $\Pi$. 

**essentially, we treat '?' as a wild card**
Bird's-eye view of the solution - series of reductions

Problem 1: find a strongest $f$ such that the product $M \times B$ is empty

Problem 2: find a minimum cutting set for a Buchi automaton

Problem 3: find a minimum cutting set for a finite automaton

To cut all accepting paths

0-1 ILP
From Problem 2 to Problem 3:

Problem 2: find a minimum cutting set for a Buchi automaton

Problem 3: find a minimum cutting set for a finite automaton

for safety properties, the Buchi automaton is already a finite automaton

- Buchi automaton $B$ with the set of accepting states $F$ of size $k$
- transitions from the $i$-th accepting state of $B_0$ to the copy $B_i$
- accepting states are here
Reducing the minimum cutting set of a finite automaton to a 0-1-ILP problem

- Each edge in the automaton has its labeling
- We only leave edges that exist in the automaton regardless of the value of '?' and edges that can exist depending on the value of '?'
- With each labeling with a positive occurrence of '?' we associate a positive propositional variable
- With each labeling with a negative occurrence of '?' we associate a negated propositional variable
- With each state we associate a propositional variable
- Constraints:
  - Initial states are reachable: for each \( s_0 \in S \), \( e_{s_0} \)
  - Accepting states are unreachable: for each \( f \in F \), \( \neg e_f \)
  - For each transition \( \langle s, l, v \rangle \), we have: \( e_s \land e_l \rightarrow e_v \)
- Objective: to minimize \( \Sigma e_l \)

Solution: \( f = \lor l \) for which \( e_l = 1 \)
Why is this correct?

- \( f \) can be represented as DNF where each term represents a full assignment
  - This corresponds to the truth table of \( f \).

- For \( \pi \in \Pi \), let
  - \( g(\pi^-) = \{\tau \mid \langle \tau, \neg ? \rangle \in \pi\} \)
  - \( g(\pi^+) = \{\tau \mid \langle \tau, ? \rangle \in \pi^+\} \) // sets of assignments

- \( f \) should contradict at least one edge in each path \( \pi \in \Pi \)
  - For \( \tau \in g(\pi^-) \), it is sufficient that \( f \models \tau \).
  - For \( \tau \in g(\pi^+) \), it is sufficient that \( f \nem \tau \).
Example

\( \varphi[?] = \text{eventually(always(?))} \)

\( \neg \varphi[?] = \text{always(eventually(\neg ?))} \)

**M:**

- **w_2:** \( p, \neg q \to \neg p, q \to p, \neg q \to \neg p, q \)

**B:**

- **s_0:** \( ? \to ? \)
- **s_1:** \( ? \to ? \)

Product automaton
0-1-ILP formulation for the example

Min $e_{-pq} + e_{p-q}$
subject to
1. $e_{-pq} \lor e_{p-q}$
2. $e_{-pq} \lor e_{p-q} \lor \neg e_{p-q}$
3. $e_{p-q} \lor e_{-pq}$
4. $e_{p-q} \lor e_{-pq} \lor \neg e_{-pq}$
5. $e_{-pq} \lor \neg e_{p-q}$
6. $e_{p-q} \lor \neg e_{-pq}$

Accepting path from the previous slide

Optimal solution $= e_{-pq} = e_{p-q} = 1$, hence $f = \neg pq \lor p \neg q = p \oplus q$

$\varphi[f] = \text{eventually(always}(p \oplus q))$
Complexity

Size of the product automaton (= of model checking):

\[ O(|B|) = O(|M| \cdot 2^{|\varphi|}) \]

Solving 0-1-ILP is bound by exponent on the number of variables, which is double-exponential in \( \mathsf{AP}' \) - a small subset of variables that are taken into consideration when computing \( f \).

likely to be more efficient in practice
In the paper but not in the presentation:

- Option 1: $f$ is stronger than $g$ if $\text{models}(f) \subseteq \text{models}(g)$ (in other words, $f \Rightarrow g$)
- Option 3: $f$ is stronger than $g$ if $\varphi[f] \Rightarrow \varphi[g]$

solved using lattices

- Multiple placeholders - solved similarly using a 0-1-ILP
Summary:

♦ Motivation
♦ Definition of query checking
♦ Introducing query checking for LTL
♦ Automata-based algorithm for computing a strongest solution
♦ Complexity

Future work:

♦ More efficient algorithms
♦ Query checking with temporal placeholders
♦ Characterization of queries for which exactly one strongest solution exists
Questions?
Model Checking

Is the system correct?

A mathematical model of the system $M$ (an FSM):

A formal specification $\psi$

Does $M$ satisfy $\psi$?

no

yes

the system is correct!

counter example