Variants of LTL Query Checking

Hana Chockler — IBM Research

Arie Gurfinkel — SEI

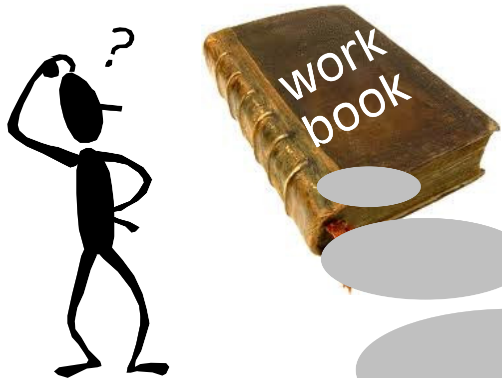
Ofer Strichman — Technion

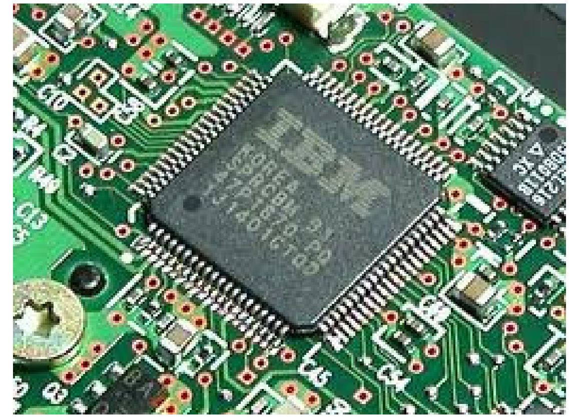
Problem Formulation – General Query Checking

We have the design:

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

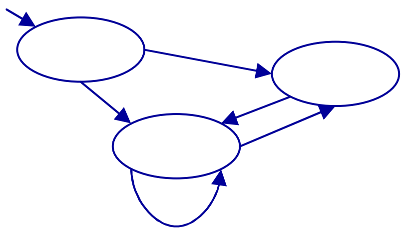
Wouldn't it be nice if an automated process could find the right property for us?

Query Checking

was defined by W. Chan in 2000

Model Checking:

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



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

Does M satisfy φ?

no

yes

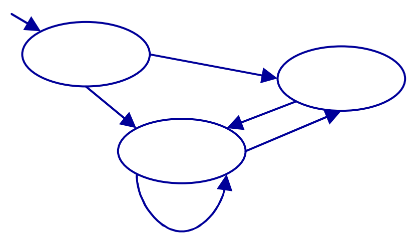
the system is correct!

counter example

Query Checking
was defined by W. Chan in 2000

Query Checking:

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



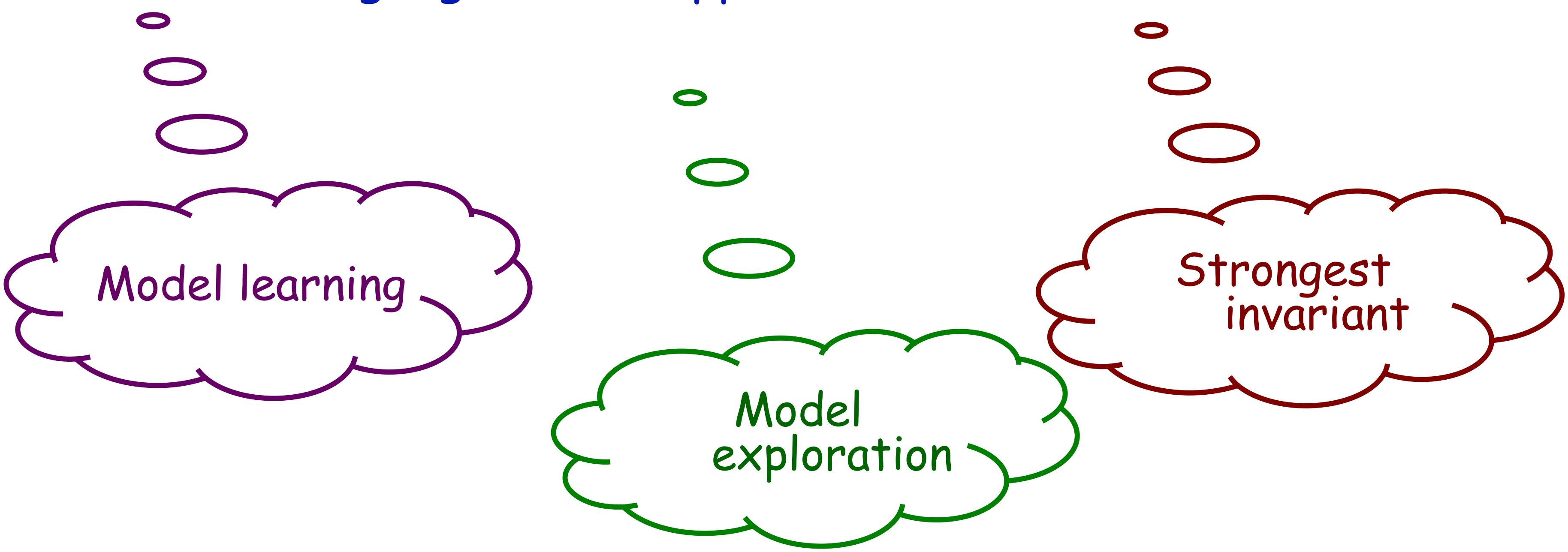
A skeleton of a formal specification – basically a formula with placeholders

What is the right φ?

A strongest φ with a given skeleton that holds in M

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



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:

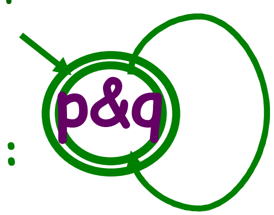
Linear Temporal Logic (LTL)

- 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

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.

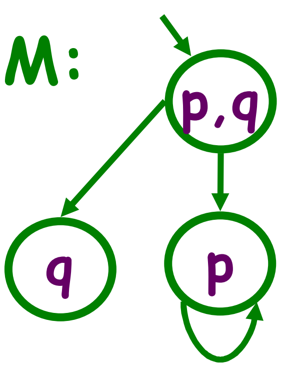
Preliminaries:

Model Checking:

- Construct an automaton B for the negation of the property φ
- Build the product MxB
- Check whether it is empty:
    - If it is empty, then M ⊨ φ
    - if not, accepting paths are counterexamples

B is an automaton for the negation of "eventually(¬p OR ¬q)"

M:



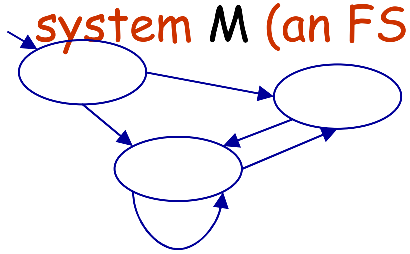
MxB is empty

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 $M \vDash \varphi[? \leftarrow f]$

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 φ?

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

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

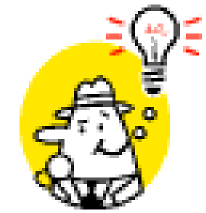
Option 1: f is stronger than g if models(f) $\subseteq$ models(g) (that is, f→g)
Option 2: f is stronger than g if |models(f)| < |models(g)|
Option 3: f is stronger than g if φ[f] → φ[g]

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 $|\text{models}(f)| < |\text{models}(g)|$

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 φ[f] is empty



Solution strategy:

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

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 MxB is empty

for a safety query

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

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

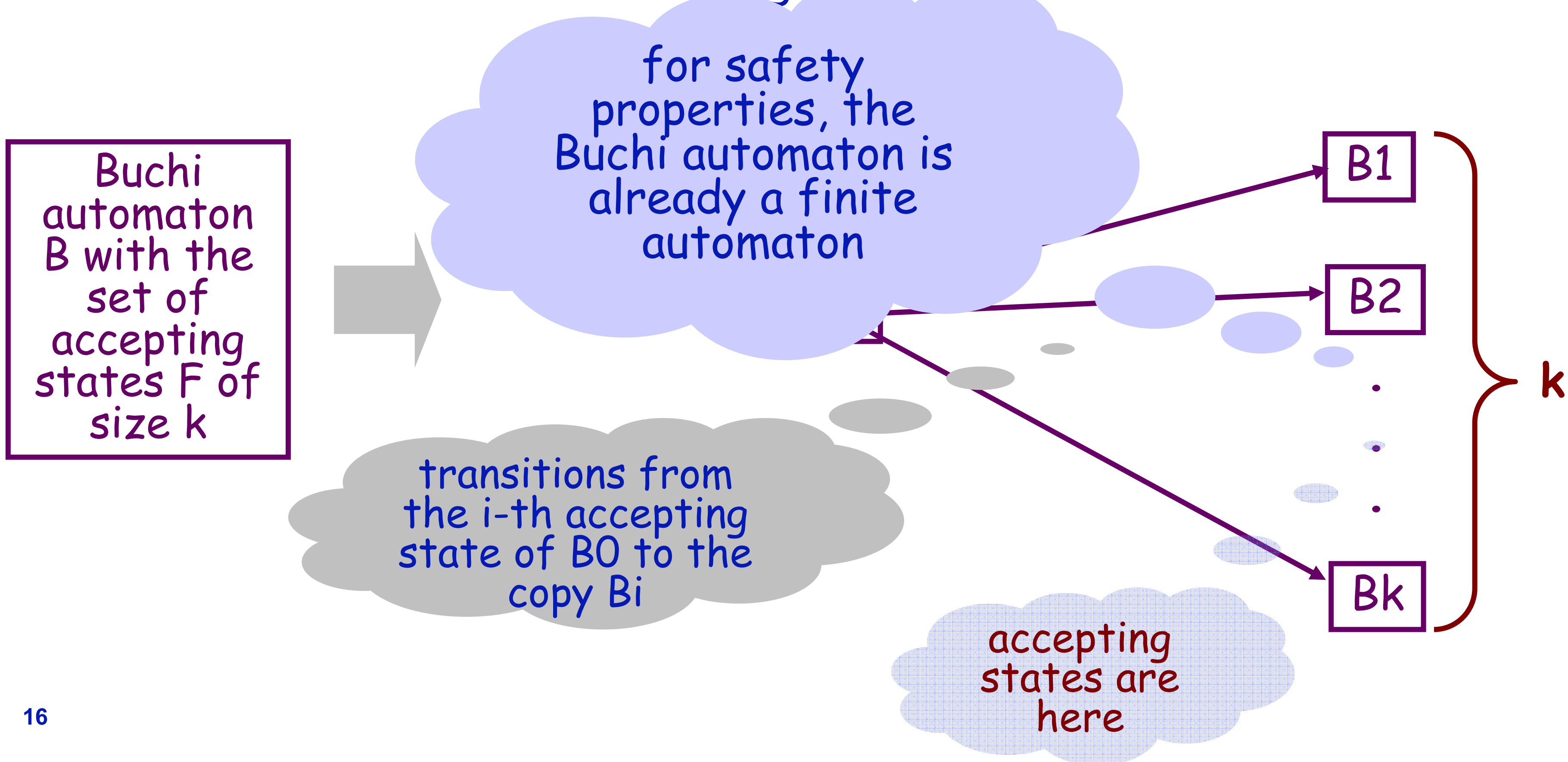
To cut all accepting paths

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



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 $s_0 \in S_{in}$. e_{s_0}
  - Accepting states are unreachable: for each $f \in F$. $\neg e_f$
  - For each transition <s,l,v>, we have: $e_s \wedge e_l \rightarrow e_v$
- Objective: to minimize $\Sigma\, e_l$

Solution: $\mathbf{f} = \vee\ \mathbf{l}$ 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.

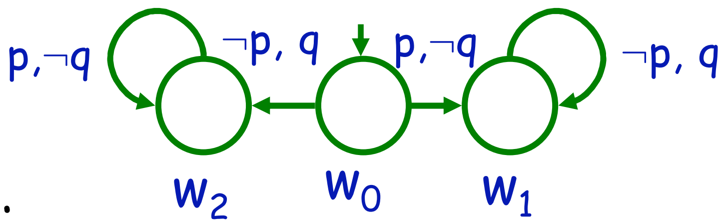
- For $\pi \in \Pi$, let
  - $\mathbf{g}(\pi^-) = \{\tau \mid \langle\tau, \neg?\rangle \in \pi\}$
  - $\mathbf{g}(\pi^+) = \{\tau \mid \langle\tau, ?\rangle \in \pi^+\}$ // sets of assignments

- f should contradict at least one edge in each path $\pi \in \Pi$
  - For $\tau \in \mathbf{g}(\pi^-)$, it is sufficient that $f \models \tau$.
  - For $\tau \in \mathbf{g}(\pi^+)$, it is sufficient that $f \not\models \tau$.

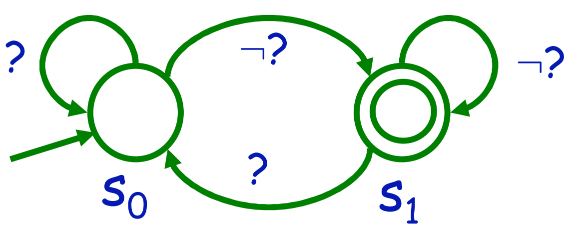
Example

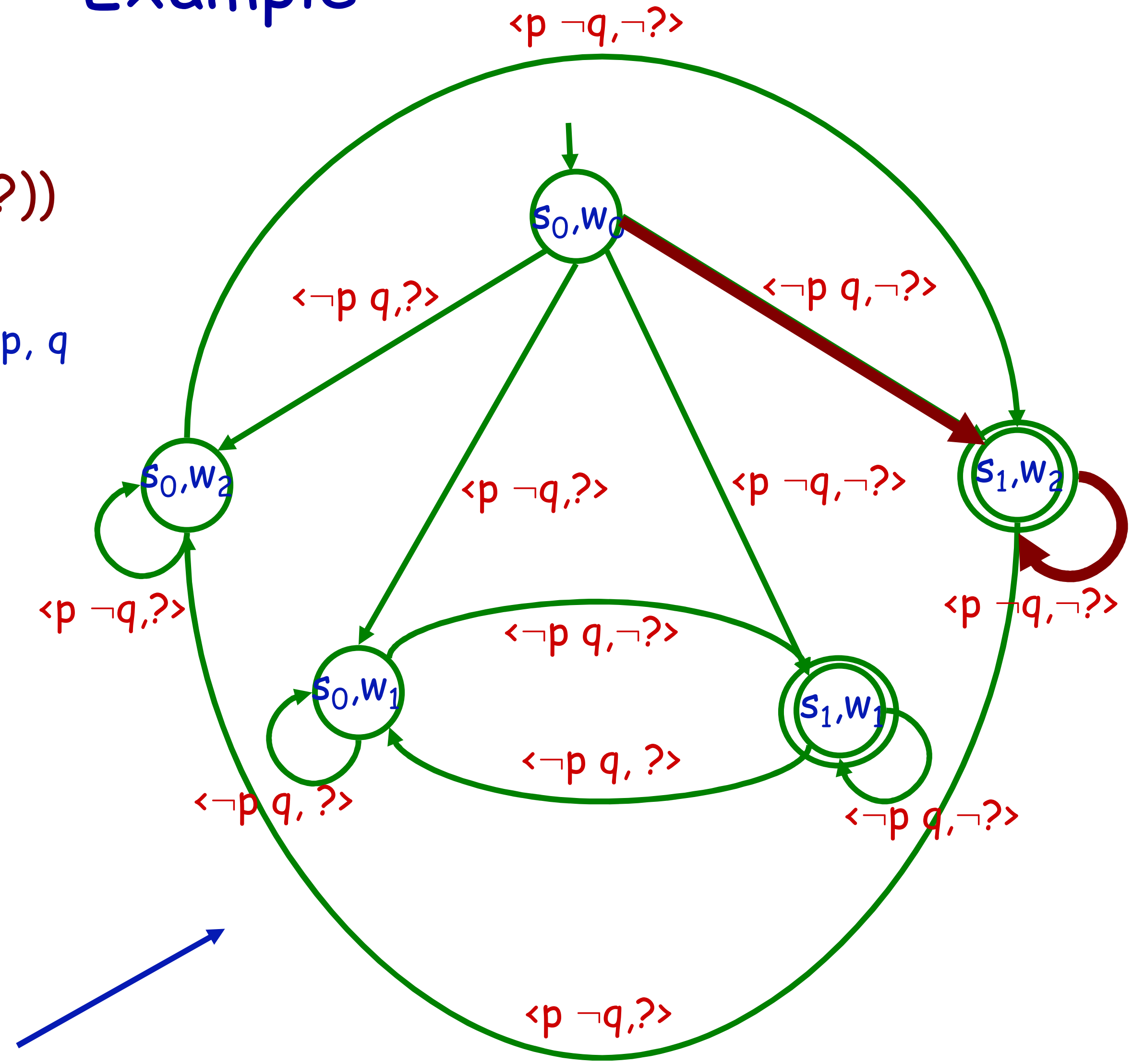
$\varphi[?]$ = eventually(always(?))
$\neg\varphi[?]$ = always(eventually(¬?))

M:



B:





product automaton

0-1-ILP formulation for the example

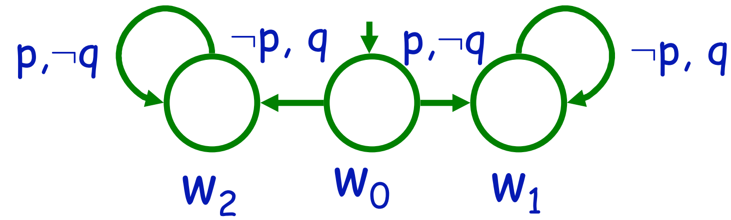
Min $e_{\neg pq} + e_{p\neg q}$

subject to

accepting path from the previous slide

1. $e_{\neg pq} \vee e_{p\neg q}$
2. $e_{\neg pq} \vee e_{p\neg q} \vee \neg e_{p\neg q}$
3. $e_{p\neg q} \vee e_{\neg pq}$
4. $e_{p\neg q} \vee e_{\neg pq} \vee \neg e_{\neg pq}$
5. $e_{\neg pq} \vee \neg e_{p\neg q}$
6. $e_{p\neg q} \vee \neg e_{\neg pq}$

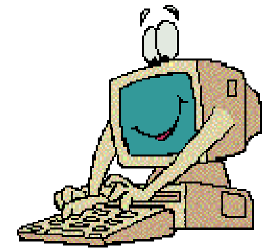
M:



Optimal solution = $e_{\neg pq} = e_{p\neg q} = 1$, hence **f = ¬pq ∨ p¬q = p ⊕ q**

φ[f] = eventually(always(p ⊕ q))

Complexity



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

$$O(|B|) \quad = \quad O(|M| \cdot 2^{|\varphi|})$$

size of the finite automaton

Solving 0-1-ILP is bound by exponent on the number of variables, which is double-exponential in ***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 models(f) $\subseteq$ models(g) (in other words, $f \rightarrow g$)
- Option 3: f is stronger than g if $\varphi[f] \rightarrow \varphi[g]$

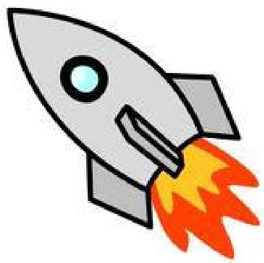
solved using lattices

- 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


Thank
You!


Questions?

Model Checking

Is the system correct?

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

A formal specification ψ

Does M satisfy ψ?

no

yes

counter example