Query Checking for Finite Linear-Time Temporal Logic

Rance Cleaveland

Department of Computer Science
University of Maryland
USA

Research supported by NSF, Office of Naval Research

Joint work with Samuel Huang

This Talk

- How to solve queries in Finite Linear Temporal Logic (LTL)!
  - What is a query?
  - Why do you want to solve them?
  - What's (Finite) LTL?
- Agenda
  - A bit about query checking
  - A bit about Finite LTL and Finite LTL queries
  - A bit about automata and Finite LTL
  - A bit about using automata to solve Finite LTL queries
  - A bit about a pilot study
  - Conclusion!

Query Checking (1/2)

- Traditional system specification via temporal logic
  - Write some formulas φ
  - Develop a system M
  - Check if M satisfies each φ ($M \vDash \varphi$)
- In practice: You often have M, but not φ
- Query checking: a way to extract φ from M

Query Checking (2/2)

- Queries = formulas with “holes” (missing subformulas)
- The query checking problem, classically
  - Given M
  - Given query $\varphi[var]$ (var is the hole)
  - Compute all (propositional) formulas γ such that M satisfies $\varphi[\gamma]$ ($\varphi[\gamma]$ is $\varphi[var]$ with all occurrences of var replaced by γ)
- Uses
  - Specification mining
  - System comprehension / understanding

Examples (LTL)

- $\mathrm{G}\ var$
  - G: "always"
  - Solutions to query: system invariants!
- $\mathrm{G}\ (var \rightarrow \mathrm{F}\ err)$
  - F: "eventually"
  - err: true in error states
  - Solutions to query: states that invariably lead to a future error!
- Note: multiple solutions usually! Are generally interested in extremal ones (maximal, minimal)

More Examples

- $\mathrm{G}(var \rightarrow \mathrm{F}\, m)$
  - m: "Microsoft share price closes higher"
  - Solutions to query: states from which Microsoft share price is guaranteed to increase (eventually)!
- $(var \wedge \neg gr) \rightarrow (\mathrm{F}\, gr)$
  - gr: "Goal reached"
  - Solutions to query: states from which the goal configuration can be reached

History of Query Checking

- Chan introduced notion in 2000 CAV paper for CTL
  - Showed that some queries have unique strongest solutions
  - Gave algorithm for computing strongest solutions in this case
- Subsequent developments for larger classes of CTL queries, some infinite-state systems by Bruns, Chechik, Godefroid, Gurfinkel, me(!) through early, mid 2000s
- Applications (e.g. “latent behavior detection” in UML, temporal-logic planner explanations)
- Extensions to LTL by Chokler, Gurfinkel, Strichman (2011), Huang and me(!) 2017

This Work

- For classical query checkers to work, you need model M
- What if you don't have M?
  - Could be proprietary
  - Could also not exist (e.g. stock market)
- This work: query checking (aka "query solving") from observed system executions
  - Given: finite set of finite state sequences ("executions"), query
  - Compute: solutions to query that make it true for each execution

Remainder of Talk

- Finite LTL (slightly different from LTL_f)
- Automata for Finite LTL formulas via tableaux
- Query checking using automata
- Proof of concept study
- Conclusion

Query Checking Generalizes Invariant Mining

- Given: collection of system executions
- Compute: invariants as association rules = propositional implications
  - Mining association rules = given states, look for associations rules that are (always / usually) true
  - Algorithm: Apriori (cf. Agrawal and Srikant 1994)
- Our earlier work: using Apriori to mine invariants from MATLAB / Simulink, e.g. $\mathrm{G}\ (brake \rightarrow cc\_inactive)$ (automotive cruise control)
  - RV 2010
  - ACM TECS 2017
  - SEFM 2018
- Query solving: generalizes invariant mining in that user specifies temporal form of queries to be solved

Finite LTL

- To have queries for finite sequences, need a logic for finite sequences
- LTL: infinite sequences!
- Finite LTL: LTL interpreted with respect to finite sequences
- Syntax

$$\varphi \ ::= \ a \mid \neg\, \varphi \mid \varphi \vee \varphi \mid \mathrm{X}\, \varphi \mid \varphi\, \mathrm{U}\, \varphi$$

a atomic proposition

$\neg$ negation

$\vee$ disjunction

X next-state

U until

Finite LTL Semantics (1/2)

- States: convey truth / falsity of atomic propositions
  - If A is (finite) set of all atomic propositions
  - State $\sigma \subseteq A$ assigns true to all $a \in \sigma$, false to all $a \notin \sigma$
- Regular LTL
  - Formulas interpreted with respect to infinite sequences of states
  - Infinite sequences always have
    - At least one state
    - A successor to every state in the sequence
- Finite sequences
  - May be empty (ε)
  - States do not always have successors in a sequence!

Finite LTL Semantics

- Let $\pi \in (2^A)^* = \sigma_1 \cdots \sigma_n$ be a finite sequence of states
- We define $\pi \vDash \varphi$ "π satisfies φ"
  - $\pi \vDash a$ iff $a \in \sigma_1$
    - So σ_1 must exist!
    - Implication: $\varepsilon \nvDash a$
    - Another implication: $\varepsilon \vDash \neg a$
  - $\pi \vDash \mathrm{X}\,\varphi$ iff $\sigma_2 \cdots \sigma_n \vDash \varphi$
    - Again, σ_1 must exist!
    - Implication: $\varepsilon \nvDash \mathrm{X}\,\varphi$ for any φ
    - Another implication: $\varepsilon \vDash \neg\mathrm{X}\neg\varphi$ for any φ
  - $\pi \vDash \varphi_1 \,\mathrm{U}\, \varphi_2$ iff
    - Suppose $\varepsilon \vDash \varphi$
    - What satisfies true $\mathrm{U}\,\varphi$?

$$
\begin{array}{cccccccc}
\sigma_1 & \sigma_2 & \cdots & \sigma_{i+1} & \sigma_i & \cdots & \sigma_n \\
[& [& \cdots & [& [& & \\
\varphi_1 & \varphi_1 & & \varphi_1 & \varphi_2 & &
\end{array}
$$

Derived Operators in Finite LTL

- $\text{true} \overset{\text{def}}{=} a \vee \neg a$
- $\text{false} \overset{\text{def}}{=} \neg\text{true}$
- $\varphi_1 \wedge \varphi_2 \overset{\text{def}}{=} \neg(\neg\varphi_1 \vee \neg\varphi_2)$
- $\overline{\mathrm{X}}\, \varphi \overset{\text{def}}{=} \neg \mathrm{X} \neg \varphi$ "weak next"
- $\varphi_1 \,\mathrm{R}\, \varphi_2 \overset{\text{def}}{=} \neg(\neg\varphi_1 \,U\, \neg\varphi_2)$ "release"
- $\mathrm{F}\, \varphi \overset{\text{def}}{=} \text{true}\; \mathrm{U}\; \varphi$ "eventually"
- $\mathrm{G}\, \varphi \overset{\text{def}}{=} \neg \mathrm{F} \neg \varphi$ "always"

Workarounds!

- $\neg a$ 
  - $\varepsilon \vDash \neg\, a$
  - $\sigma_1 \cdots \sigma_n \vDash (\neg a) \wedge \mathrm{X}\,\mathrm{true}$ iff $n \geq 1$ and $a \notin \sigma_1$ 
- $\mathrm{F}\ \neg a$ 
  - $\pi \vDash \mathrm{F}\neg a$ all π
  - $\sigma_1 \cdots \sigma_n \vDash \mathrm{F}\,(\neg a \wedge \mathrm{X}\,\mathrm{true})$ iff $a \notin \sigma_i$ some i 
- $\mathrm{G}\ a$ 
  - $\pi \nvDash \mathrm{G}\, a$ all π
  - $\sigma_1 \cdots \sigma_n \vDash \mathrm{G}\,(a \vee \overline{\mathrm{X}}\,\mathrm{false})$ iff $a \in \sigma_i$ all i 

Facts about Finite LTL

- $\emptyset \subseteq A$ is state making all atomic propositions false
  - Let ϕ be a propositional formula (no X, U, etc.)
  - Then $\varepsilon \vDash \phi$ iff $\emptyset \vDash \phi$
- This fact implies:
  - Usual propositional identities hold (deMorgan, distributivity, etc.).
  - A principled $O(n)$ strategy for encoding LTL_f exists!

S. Huang and R. Cleaveland, “A tableau construction for Finite Linear-Time Temporal Logic”, *Journal of Logic and Algebraic Methods in Programming* (2022).

From (Finite) LTL to Automata

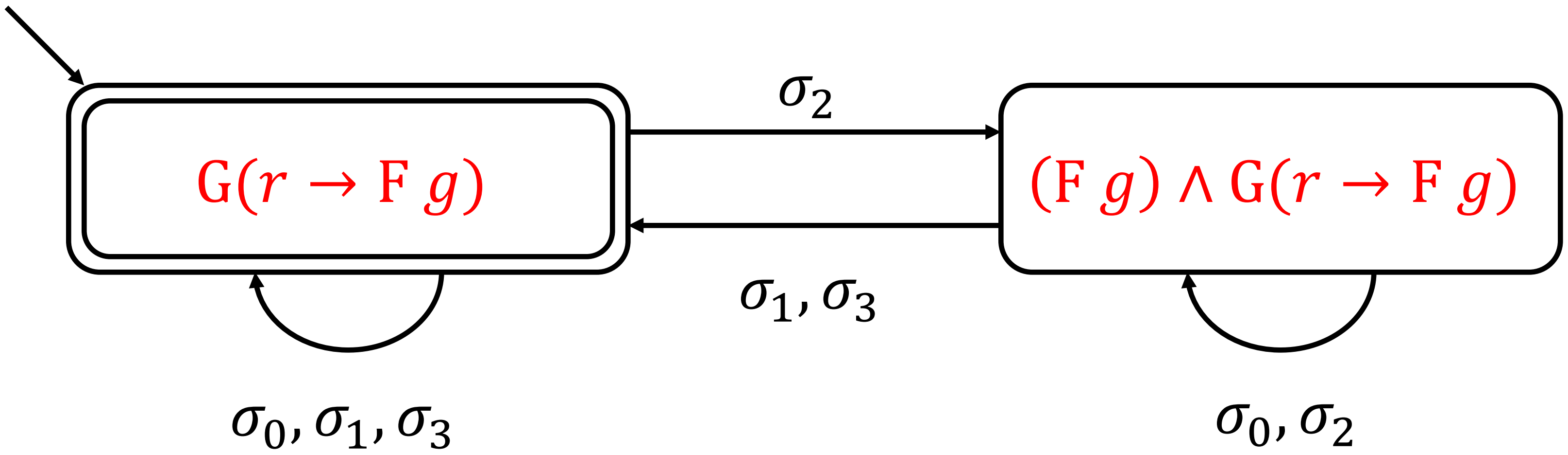
- Motivation for Finite LTL: query checking over finite sets of system behaviors!
- Important mathematical questions
  - Satisfiability: is a (Finite) LTL formula satisfiable?
  - Model checking: do a system's finite behaviors satisfy a given formula?
  - Synthesis: generate a sequence / system satisfying a given formula
- How to address these questions? Automata
  - Generate finite-state machines from formulas
  - Use resulting machines as basis for analysis procedures
- Notation: $L(\varphi) = \{\pi \in (2^A)^* \mid \pi \vDash \varphi\}$

Tableau Constructions

- Basis for generating automata from LTL: tableaux
  - Generate automata whose vertices are labeled by sets of formulas, transitions labeled by propositional states (= assignments of truth values to atomic propositions, recall)
  - Key property: sequences accepted by a given vertex are the sequences making the associated formula true
- For LTL, sequences are infinite, so automata are ω-automata (e.g. Büchi) accepting infinite sequences
- For Finite LTL, regular finite automata suffice
  - Automata accept / reject finite sequences
  - Construction still associates formula with each vertex

Example

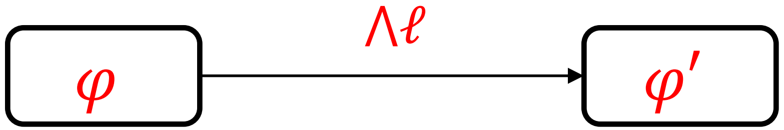
- $\mathrm{G}(r \rightarrow \mathrm{F}\ g)$
  - r, g are atomic propositions standing for "request" and "grant"
  - Property asserts that every request is eventually granted
  - What is automaton for this Finite LTL formula?



State	2^A
σ_0	$\{\ \}$
σ_1	$\{g\}$
σ_2	$\{r\}$
σ_3	$\{g, r\}$

The Tableau Construction for Finite LTL

- Basic step
  - Convert Finite LTL formula φ into automaton normal form (ANF)
    - Literal ℓ: a or $\neg a$ for some $a \in A$
    - ANF clause: $(\bigwedge\ell) \wedge \mathrm{N}\varphi'$ where N is either X or $\overline{\mathrm{X}}$
    - ANF: $\bigvee C$ where each C is an ANF clause
    - Example: $\mathrm{F}\ a$ (a atomic) can be converted into ANF formula $a \vee \mathrm{X}\ (\mathrm{F}\ a)$
  - Turn each ANF clause $C = \bigwedge\ell \wedge \mathrm{N}\varphi'$ into transitions
  - Accepting states: those whose formulas are satisfied by ε (syntactically checkable)
  - Result: NFA M_φ with $L(M_\varphi) = L(\varphi)$



Discussion

- #states $\leq 2^{|\varphi|}$
- Experimental evaluation (184 LTL formulas from Spot benchmark): on average, #states $= 0.69 \cdot |\varphi|$
- Other methods for generating NFAs from Finite LTL go "through" e.g. Büchi automata (or alternating automata, …)
  - Finite LTL φ translated into LTL φ' over "infinite-ized" sequences
  - Procedure used to convert φ' to Büchi automaton
  - Büchi automaton then converted to NFA for φ'
- Pros / cons
  - Pro: Highly optimized procedures for LTL-to-Büchi!
  - Con: Loss of connection between automaton states and formulas

Finite LTL Queries

- … Finite LTL formulas with "holes" (denoted var)
  - We write $\varphi[var]$ to emphasize presence of var
  - If γ is a formula then formula $\varphi[\gamma]$ is the instantiation of $\varphi[var]$ by γ
- Example
  - Let m, a, g be atomic propositions reflecting "Microsoft / Amazon / Google share price rises" on a given day.
  - State sequences: daily stock information over period of days/weeks/etc.
  - Query: $\varphi[var] = \mathrm{G}\,(var \rightarrow \mathrm{F}\,m)$
    - var is hole
    - $\varphi[a \wedge \neg g] = \mathrm{G}\,((a \wedge \neg g) \rightarrow \mathrm{F}\,m)$ is instantiation of $\varphi[var]$ by $\gamma = a \wedge \neg g$
    - Instantiation says: "it is always the case that if Amazon goes up and Google does not on a given day, then Microsoft goes up eventually"

The Finite LTL Query-Checking Problem

- Given:
  - $\Pi = \{\pi_1, \ldots, \pi_n\} \subseteq (2^A)^*$
  - Finite LTL query $\varphi[var]$
- Compute:

  $QC(\varphi[var], \Pi) =$ all propositional γ such that $\pi_i \vDash \varphi[\gamma]$ all i
- E.g.

  If $\varphi[var] = \mathrm{G}\ (var \rightarrow \mathrm{F}\ m)$ then $QC(\varphi[var], \Pi)$ returns characterization of all states guaranteeing that Microsoft stock goes up eventually!

Solving the Query-Checking Problem

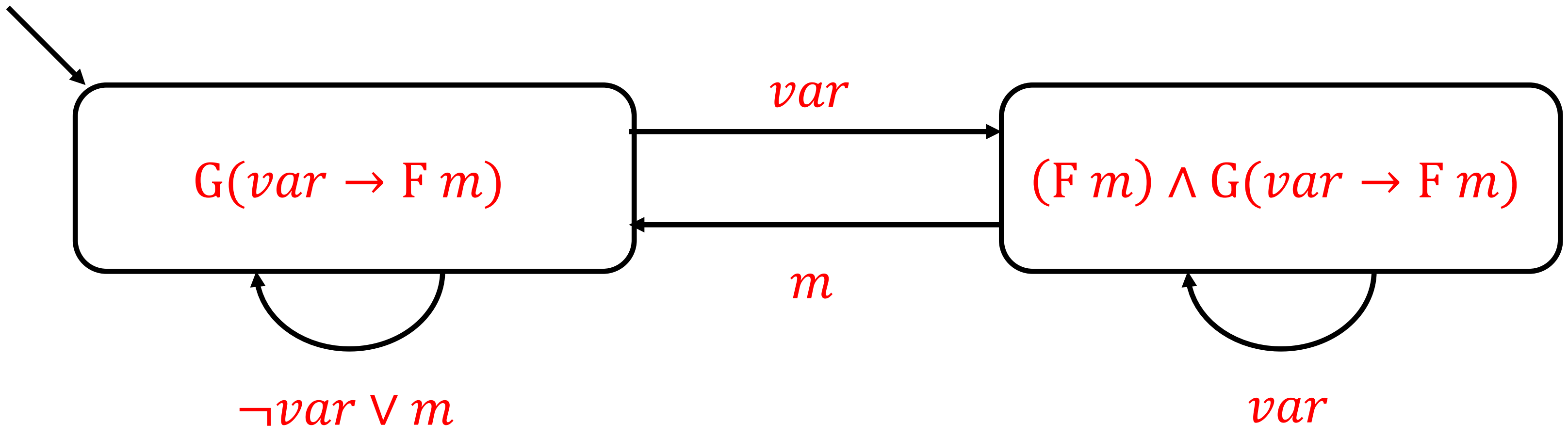
- Brute force: enumerate all (semantically distinct) possible solutions
  - If $|A| = n$ then there are 2^A possible states
  - If there are m states then there are 2^m semantically distinct propositions
    - Proposition = set of states (those making proposition true)
    - For each subset of states there is a distinct proposition!
  - So if $|A| = n$ then there are 2^{2^n} possible distinct propositions
    - If $|A| = 2$ then there are $2^{2^2} = 2^4 = 16$ distinct propositions
    - If $|A| = 8$ then there are $2^{2^8} = 2^{256} \approx 10^{78}$ = # of atoms in the observable universe possible propositions
- A better approach: use automata!

Automata and Query Checking

- Recall automaton-based approach to LTL model checking (= checking if every (infinite) execution $\pi \in L(M)$ satisfies LTL formula φ)
  - Build Büchi automaton $B_{\neg\varphi}$ for $\neg\varphi$
  - Check if $L(M) \cap L(B_{\neg\varphi}) = \emptyset$ by composing $M, B_{\neg\varphi}$
- We do something similar to compute $QC(\Pi, \varphi[var])$
  - Negate query $\varphi[var]$, obtaining $\neg\varphi[var]$
  - Make automaton M_Π such that $L(M_\Pi) = \Pi$
  - Construct query automaton $M_{\neg\varphi[var]}$ with property that for all propositional γ, $L(M_{\neg\varphi[\gamma]}) = L(\neg\varphi[\gamma])$
  - Compose $M_\Pi, M_{\neg\varphi[var]}$ to obtain query automaton $M_{\Pi,\neg\varphi[var]}$
  - Compute all propositional γ such that $L(M_{\Pi,\neg\varphi[\gamma]}) = \emptyset$

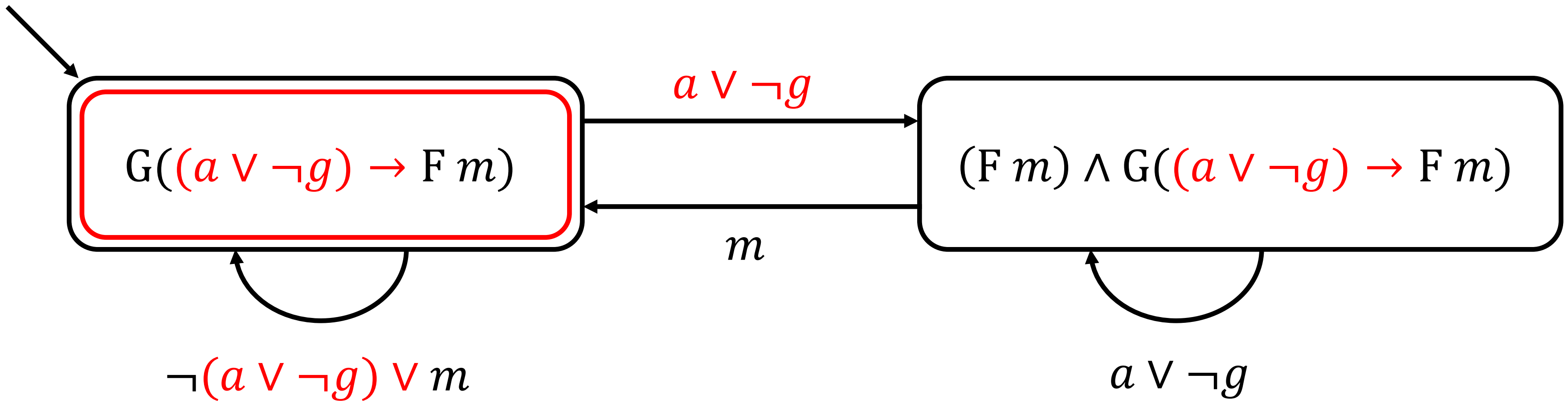
Query Automata

- Like (symbolic) NFAs constructed from Finite LTL except:
  - States labeled by queries rather than formulas
  - Transitions labeled by propositional queries rather than formulas
  - Acceptance depends on var
- Example: $M_{\varphi[var]}$ where $\varphi[var] = \mathrm{G}(var \to \mathrm{F}\ m)$



Instantiating a Query Automaton

- If $M[var]$ is a query automaton and γ is a propositional formula then $M[\gamma]$ is the finite automaton obtained by
  - Instantiating all the queries in $M[var]$ with γ; and
  - Labeling all states whose instantiated query accept ε as accepting
- E.g. $M[a \vee \neg g]$:



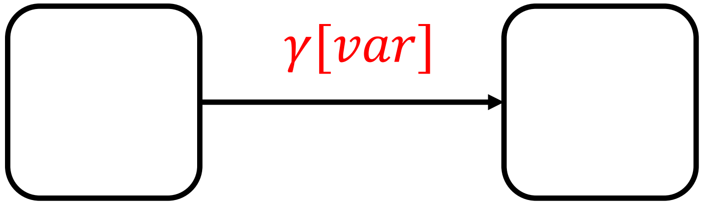
Constructing Query Automata

- Tableau method for Finite LTL can be used to construct query automata for Finite LTL queries!
  - Treat var as atomic proposition
  - Apply Finite LTL tableau method
  - Ignore accepting states in resulting automata
- Simplification: queries in states can be replaced by propositional queries; see paper

S. Huang and R. Cleaveland. "Temporal-Logic query checking over finite data streams," *International Journal on Software Tools for Technology Transfer* (2022).

Shattering Query Automata

- Key operation for $QC(\Pi, \varphi[var])$: find γ so that $L\left(M_{\Pi,\neg\varphi[\gamma]}\right) = \emptyset$
- The shattering problem for finite-query automata
  - Given: $M_{\varphi[var]}$
  - Compute: all γ such that $L(M_{\varphi[\gamma]}) = \emptyset$
- General approach: systematically search for γ that shatter edge queries (i.e. make $\varphi[\gamma] \equiv \text{false}$), make states non-accepting.
- E.g.
  - Suppose transition label is $\gamma[var] = var \wedge (a \vee b)$
  - $\gamma[\neg a \wedge \neg b] = (\neg a \wedge \neg b) \wedge (a \vee b) \equiv \text{false}$
  - Setting $var = \neg a \wedge \neg b$ shatters this edge!



Complexity

- $O(2^{2^{|\varphi[var]|}})$ in worst case
- In practice: $O(2^{|\varphi[var]|})$
- Optimizations in paper improve run-time

Pilot Study

Experimental evaluation on synthetic data from past data-mining competitions involving product sales, promotions

- Can deal with up to six atomic propositions, depending on sequence length
- Some correlations among promotions / products and other products detected

Conclusions

- Finite LTL queries: "templates" for formal specifications
- Query checking: given observations of system behavior, figure out how to instantiate templates so every sequence satisfies them
- Approach is based on connection between Finite LTL formulas, automata
- "Proof of concept" implementation and experimental study
- Future directions
  - More thorough evaluation
  - Applications!
  - Other logics besides Finite LTL (mu-calculus, Allen intervals, time, …)
  - Relaxed query checking ("near invariants")
  - Noisy LTL

THANKS!

Rance Cleaveland

rance@cs.umd.edu