Inferring Temporal System Properties

Samuel Huang, joint work with Rance Cleaveland

University of Maryland, College Park

April 28th, 2011
Model Checking Problem

Ask: \( M \models \phi \)?

- \( M \) is a model
- \( \phi \) is a property/requirement
- \( \models \) is a satisfaction relation
Related Problems

Synthesis Problem

Find a suitable $M$: $\square \models \phi$

Property extraction problem

Find all suitable $\phi$: $M \models \{\square\}$
Related Problems

Synthesis Problem

Find a suitable \( M: \square \models \phi \)

Property extraction problem

Find all suitable \( \phi: \quad M \models \{\square\} \)
Motivation for Requirements Extraction

- System comprehension
- System reconstruction
  - Incomplete/missing/out-dated documentation
  - “Implicit” (and sometimes unintended) requirements (during construction of system)
- Requirements extraction can serve as a way to estimate high level behavior of a system in terms of the properties that it exhibits.
Automatic Requirement Extraction from Test Cases [ACH+10] (joint work with Fraunhofer and Robert Bosch)
By varying the method by which test cases are generated, we extracted different degrees of requirements

- Randomized - yielded sparse and lower total number of requirements
- Structurally guided (MCDC coverage) - more complete overall requirement set

Iterating requirement extraction process helped lead to refinement of final results
By varying the method by which test cases are generated, we extracted different degrees of requirements

- Randomized - yielded sparse and lower total number of requirements
- Structurally guided (MCDC coverage) - more complete overall requirement set

Iterating requirement extraction process helped lead to refinement of final results
Observations

- By varying the method by which test cases are generated, we extracted different degrees of requirements
  - Randomized - yielded sparse and lower total number of requirements
  - Structurally guided (MCDC coverage) - more complete overall requirement set

- Iterating requirement extraction process helped lead to refinement of final results
Observations

- By varying the method by which test cases are generated, we extracted different degrees of requirements
  - Randomized - yielded sparse and lower total number of requirements
  - Structurally guided (MCDC coverage) - more complete overall requirement set
- Iterating requirement extraction process helped lead to refinement of final results
In this work we assumed the model was known to us, and a test suite was generated to satisfy some coverage criterion on the model. What can be done without knowledge of the model?

Given a set of a system’s executions $E$, what properties can be discovered of the system that hold “true?”

Here a “true” property means one with some measure of accuracy over the execution set $E$, such as satisfying some support. [AIS93]

The properties discovered should be in some understandable and usable format, such as a temporal logic.
Treat set $E$ as a sequence database, and incorporate sequential pattern mining. [AS95, YHA03, Moe07]

Can mine patterns of the form

$$A \rightarrow B \rightarrow C \rightarrow \ldots$$

Which can be rewritten as

$$F (A \rightarrow XF (B \rightarrow XF (\ldots))))$$
Sequential pattern mining algorithms do not only return patterns that are correct 100% of the time. Typically they require a support parameter, which specifies how often rules must be correct to be considered significant.

The previous rule is more properly written as

\[ P_{\geq s} [F (A \rightarrow XF (B \rightarrow XF (\ldots)))] \]

Here, the rule has been written in probabilistic temporal logic expressing uncertainty in its occurrence.
Recent work [LKL07, LKL08] discovers rules of a software code base (JBoss Application Server) in an effort to uncover underlying program design and identify bugs. Characterization of temporal logic fragments that are covered is unclear.

Next Steps

- Expand supported fragment of temporal logic as much as possible. How far can we go?
- Different fragments are useful for different application domains
  - Software engineering/program analysis:
    \[ \text{event} \rightarrow F\neg (\text{power stays on}) \]
  - Metabolic pathways:
    \[ \text{protA} \rightarrow \text{protB} \rightarrow \neg \text{protC} \rightarrow \text{protD} \]
    \[ \text{protA} \rightarrow \text{protB} \downarrow \text{protC} \rightarrow \text{protD} \]
Thanks!

Samuel Huang, joint work with Rance Cleaveland

Inferring Temporal System Properties
Christopher Ackermann, Rance Cleaveland, Samuel Huang, Arnab Ray, Charles P. Shelton, and Elizabeth Latronico.

Automatic requirement extraction from test cases.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami.

Mining association rules between sets of items in large databases.

Rakesh Agrawal and Ramakrishnan Srikant.

Mining sequential patterns.
Laurence Calzone, François Fages, and Sylvain Soliman.
Biocham: an environment for modeling biological systems and formalizing experimental knowledge.


David Lo, S.-C Khoo, and C Liu.
Mining temporal rules from program execution traces.

In *Proc. of Int. Work. on Program Comprehension through Dynamic Analysis, 2007.*

David Lo, Siau-Cheng Khoo, and Chao Liu.
Mining past-time temporal rules from execution traces.

In *Proceedings of the 2008 international workshop on dynamic analysis: held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2008), WODA ’08, pages 50–56, New York, NY, USA, 2008. ACM.*
Fabian Moerchen.
Unsupervised pattern mining from symbolic temporal data.

Xifeng Yan, Jiawei Han, and Ramin Afshar.
CloSpan: Mining Closed Sequential Patterns in Large Datasets.