Introduction to Models and Properties

Computer Science and Artificial Intelligence Laboratory
MIT
Armando Solar-Lezama

Nov 22, 2011
## Recap

<table>
<thead>
<tr>
<th>Properties</th>
<th>Type Inference</th>
<th>Symbolic Execution</th>
<th>Abstract Interpretation</th>
<th>Model Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties of variables</td>
<td>Properties at program points</td>
<td>Properties at program points</td>
<td>Properties of execution traces</td>
<td></td>
</tr>
<tr>
<td>Flexible</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Push-button</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Model Checking Today

Hardware Model Checking
- part of the standard toolkit for hardware design
  - Intel has used it for production chips since Pentium 4
  - For the Intel Core i7, most pre-silicon validation was done through formal methods (i.e. Model Checking + Theorem Proving)
- many commercial products
  - IBM RuleBase, Synopsys Magellan, ...

Software Model Checking
- Static driver verifier now a commercial Microsoft product
- Java PathFinder used to verify code for mars rover

This doesn’t mean Model Checking is a solved problem
- Far from it
Model Checking Genesis

The paper that started it all

- Clarke and Emerson, Design and Synthesis of Synchronization Skeletons using branching time temporal logic

“Proof Construction is Unnecessary in the case of finite state concurrent systems and can be replaced by a model-theoretic approach which will mechanically determine if the system meets a specification expressed in propositional temporal logic”
Intellectual Roots

Two important developments preceded this paper

- Verification through exhaustive exploration of finite state models
  • G. V. Bochmann and J. Gecsei, *A unified method for the specification and verification of protocols*, Proc. IFIP Congress 1977

- Development of Linear Temporal Logic and its application to specifying system properties
  • A. Pnueli, *The temporal semantics of concurrent programs*. 1977
Model Checking

The model checking approach
(as characterized by Emerson)

- Start with a program that defines a finite state graph \( M \)
- Search \( M \) for patterns that tell you whether a specification \( f \) holds
- Pattern specification is flexible
- The method is efficient in the sizes of \( M \) and hopefully also \( f \)
- The method is algorithmic
So what exactly is a model?

Remember our friend \( \vdash \)?

- What does this mean? \( \vdash x \land y \Rightarrow x \)
  - The statement above can be established through logical deduction
  - Axiomatic semantics and type theory are deductive
    - The program, together with the desired properties make a theorem
    - We use deduction to prove the theorem

- What about this; is it true? \( \vdash x + y == 5 \)
  - We can not really establish this through deduction
  - We can say whether it’s true or false under a given model
    \([x=3, y=2] \models x + y == 5\)

You have seen this symbol too \( \models \)

- In operational semantics, the variable assignments were the model
- The program behavior was the theorem we were trying to prove under a given model
Consider the following sentence:
- \( S := \text{The class today was awesome} \)

Is this sentence true or false?
- that depends
  - What class is “the class”? What day is “today”?

We can give this sentence an Interpretation
- \( I := \text{The class is 6.820, Today is Tuesday Nov 22} \)

When an interpretation \( I \) makes \( S \) true we say that
- \( I \) satisfies \( S \)
- \( I \) is a model of \( S \)
- \( I \models S \)
The model checking problem

We are interested in deciding whether \( I \equiv S \) for the special case where
- \( I \) is a Kripke structure
- \( S \) is a temporal logic formula

Today you get to learn what each of these things are

But the high level idea is:
- Unlike axiomatic semantics, where the program was part of the theorem,
- The program will now be the \textit{model}
  - Well, not the program directly, but rather a kripke structure representing the program
Kripke Structures as Models

Kripke structure is a FSM with labels

Kripke structure = (S, S0, R, L)

- S = finite set of states
- S0 ⊆ S = set of initial states
- R ⊆ S x S = transition relation
- L : S → 2^AP = labels each state with a set of atomic propositions
Microwave Example

- $S = \{s_1, s_2, s_3, s_4\}$
- $S_0 = \{s_1\}$
- $R = \{ (s_1, s_2), (s_2, s_1), (s_1, s_4), (s_4, s_2), (s_2, s_3), (s_3, s_2), (s_3, s_3) \}$
- $L(s_1) = \{-\text{close, -start, -cooking}\}$
- $L(s_2) = \{\text{close, -start, -cooking}\}$
- $L(s_3) = \{\text{close, start, cooking}\}$
- $L(s_4) = \{-\text{close, start, -cooking}\}$

Can the microwave cook with the door open?
Kripke structures describe computations

A Kripke structure can describe an infinite process
- We can interpret it as an infinite tree

- We need a language to describe properties of paths down the computation tree
Linear Temporal Logic

Let $\pi$ be a sequence of states in a path down the tree

- $\pi := s_0, s_1, s_2, \ldots$
- Let $\pi_i$ be a subsequence starting at $i$

We are going to define a logic to describe properties over paths
Properties over states

State Formulas
- Can be established as true or false on a given state
- If \( p \in \{AP\} \) then \( p \) is a state formula
- if \( f \) and \( g \) are state formulas, so are \( (f \text{ and } g) \), \( (\text{not } f) \), \( (f \text{ or } g) \)
- Ex. \( (\text{not closed } \text{ and } \text{cooking}) \)
For paths

Path formulas
- a state formula p is also a path formula
  • p(π_i) := p(s_i)
- boolean operations on path formulas are path formulas
  • f and g(π_i) := f(π_i) and g(π_i)
- path quantifiers
  • G f (π_i) := globally f (π_i) = forall k>= i f (π_k) (may abbreviate as □ )
  • F f (π_i) := eventually f (π_i) = exists k>= i f (π_k) (may abbreviate as ◊ )
  • X f (π_i) := next f (π_i) = f (π_{i+1}) (may abbreviate as ◦ )
  • f U g (π_i) := f until g = exists k >= i s.t. g(π_k) and f(π_j) for i<=j<k

Given a formula f and a path π,
- if f(π) is true, we say that π ⊨ f
Examples

If you submit your homework (submit) you eventually get a grade back (grade)
- $G \ (\text{submit} \Rightarrow \text{F grade})$

You should get your grade before you submit the next homework
- $G \ (\text{submit} \Rightarrow X \ (\neg \text{submit U grade}))$
  - What’s wrong with $G \ (\text{submit} \Rightarrow (\neg \text{submit U grade})))$?

If assignment $i$ was submitted before drop date, you should get your grade before drop date
- $(G \ (\text{submit}_i \Rightarrow \text{F dropDate})) \Rightarrow ((G \ (\text{grade}_i \Rightarrow \text{F dropDate})))$
- and $G \ (\text{submit} \Rightarrow \text{F grade})$
Relationship to Kripke structure

A Kripke structure represents a set of paths
- We want to establish the validity of a formula \( f \) under a Kripke structure \( M \) and a start state \( s \)

problem:
- formula is defined for a path, Kripke structure has many paths
CTL* Logic

Add two extra path quantifiers
- \( A f := \) for all paths, \( f \)
- \( E f := \) for some path, \( f \)

Two important subsets:
- \( \text{LTL} : \) all formulas of the form \( A f \)
  - Ex: \( A(FG \ p) \)
- \( \text{CTL} : \) there must be a path quantifier before every linear operator
  - Ex: \( AG \ (EF \ p) \)
- The two are different!
Example:

What does the following formula mean
- \( A( F G p) \)

How about
- \( A( F A G p) \)

How about
- \( A(F E G p) \)