Model Checking: An Introduction

Meeting 2
Fundamentals of Programming Languages
CSCI 5535, Spring 2009

Announcements

• Office hours
  - M 1:30pm-2:30pm
  - W 5:30pm-6:30pm (after class)
  - and by appointment
  - ECOT 621
• Moodle problems?

A Double Header

• Two lectures
  - Model checking primer
  - Software model checking
  - SLAM and BLAST (tools)
• Some key players:
  - Model checking
    Ed Clarke, Ken McMillan, Amir Pnueli
  - SLAM
    Tom Ball, Sriram Rajamani
  - BLAST
    Ranjit Jhala, Rupak Majumdar, Tom Henzinger

Who are we again?

• We’re going to find critical bugs in important bits of software
  - using PL techniques!
• You’ll be enthusiastic about this
  - and thus want to learn the gritty details

Take-Home Message

• Model checking is the exhaustive exploration of the state space of a system, typically to see if an error state is reachable. It produces concrete counterexamples.
• The state explosion problem refers to the large number of states in the model.
• Temporal logic allows you to specify properties with concepts like “eventually” and “always”.

Overarching Plan

Model Checking (today)
  - Transition systems (i.e., models)
  - Temporal properties
  - Temporal logics: LTL and CTL
  - Explicit-state model checking
  - Symbolic model checking

Counterexample Guided Abstraction Refinement
  - Safety properties
    - Predicate abstraction “c2bp”
    - Software model checking “bebop”
    - Counterexample feasibility “newton”
  - Abstraction refinement weakest pre, thrm prv
### Model Checking

There are complete courses in model checking (see ECEN 5139, Prof. Somenzi).

*Model Checking* by Edmund M. Clarke, Orna Grumberg, and Doron A. Peled.

*Symbolic Model Checking* by Ken McMillan.

We will skim.

### Keywords

Model checking is an automated technique

Model checking verifies transition systems

Model checking verifies temporal properties

Model checking falsifies by generating counterexamples

A model checker is a program that checks if a (transition) system satisfies a (temporal) property

### Verification vs. Falsification

- What is verification?
  - *prove* that a property of a system holds

- What is falsification?
  - *disprove* that a property holds

### Temporal Properties

- **Temporal Property**
  - A property with time-related operators such as "invariant" or "eventually"

**Invariant** is true in a state if property $p$ is true in every state on all execution paths starting at that state $G, A6, \Box$ ("globally" or "box" or "forall")

**Eventually** is true in a state if property $p$ is true at some state on every execution path starting from that state $F, AF, \Diamond$ ("future" or "diamond" or "exists")
An Example Concurrent Program

- A simple concurrent mutual exclusion program
- Two processes execute asynchronously
- There is a shared variable turn
- Two processes use the shared variable to ensure that they are not in the critical section at the same time
- Can be viewed as a "fundamental" program: any bigger concurrent one would include this one

```
10: while (true) {
11:   wait(turn == 0);
12:   // critical section
13:   work(); turn = 1;
14: }

15: while (true) {
16:   wait(turn == 1);
17:   // critical section
18:   work(); turn = 0;
19: }
```

Reachable States of the Example Program

Example Properties of the Program

- "In all the reachable states (configurations) of the system, the two processes are never in the critical section at the same time"
- "pc1=12", "pc2=22" are atomic properties for being in the critical section

```
Invariant(¬(pc1=12 ∧ pc2=22))
```

- "Eventually the first process enters the critical section"

```
Eventually(pc1=12)
```

Analyzed System is a Transition System

- Labeled transition system
- Also called a Kripke Structure

```
T = (S, I, R, L)

- S = Set of states // standard FSM
- I ⊆ S = Set of initial states // standard FSM
- R ⊆ S × S = Transition relation // standard FSM
- L: S → 2^AP = Labeling function // this is new!

AP: Set of atomic propositions (e.g., "x=5" ∈ AP)

- Atomic propositions capture basic properties
- For software, atomic props depend on variable values
- The labeling function labels each state with the set of propositions true in that state
```

Example Properties of the Program

- "In all the reachable states (configurations) of the system, the two processes are never in the critical section at the same time"
- "pc1=12", "pc2=22" are atomic properties for being in the critical section

```
Invariant(¬(pc1=12 ∧ pc2=22))
```

- "Eventually the first process enters the critical section"

```
Eventually(pc1=12)
```

Temporal Logics

For what?

- For expressing properties

There are four basic temporal operators:

- X p
  - Next p, p holds in the next state

- G p
  - Globally p, p holds in every state, p is an invariant

- F p
  - Future p, p will hold in a future state, p holds eventually

- p U q
  - p Until q, assertion p will hold until q holds

- Precise meaning of these temporal operators are defined on execution paths
### Execution Paths

- A **path** in a transition system is an infinite sequence of states $(s_0, s_1, s_2, ...)$, such that $\forall i \geq 0. (s_i, s_{i+1}) \in R$
- A path $(s_0, s_1, s_2, ...)$ is an **execution path** if $s_0 \in I$
- Given a path $x = (s_0, s_1, s_2, ...)$
  - $x_i$ denotes the $i$th state: $s_i$
  - $x^i$ denotes the $i$th suffix: $(s_i, s_{i+1}, s_{i+2}, ...)$
- In some temporal logics one can quantify paths starting from a state using **path quantifiers**
  - $A$ : for all paths
  - $E$ : there exists a path

### Paths and Predicates

- We write $x \models p$
  - "the path $x$ makes the predicate $p$ true"
  - $x$ is a path in a transition system
  - $p$ is a temporal logic predicate
- Example:
  - $A x. \quad x \models G (\neg \neg \neg \neg (pc1=12 \land pc2=22))$

### Linear Time Logic (LTL)

- LTL properties are constructed from atomic propositions in $AP$: logical operators $\land, \lor, \neg$; and temporal operators $X, G, F, U$.
- The semantics of LTL is defined on paths:
  - Given a path $x$:
    - $x \models p$ iff $L(x_0, p)$
    - $x \models X p$ iff $x_1 \models p$
    - $x \models F p$ iff $\exists i \geq 0. x_i \models p$
    - $x \models G p$ iff $\forall i \geq 0. x_i \models p$
    - $x \models p U q$ iff $\exists i \geq 0. x_i \models q$ and $\forall j < i. x_j \models p$

### Satisfying Linear Time Logic

- Given a transition system $T = (S, I, R, L)$ and an LTL property $p$, $T$ **satisfies** $p$ if all paths starting from all initial states $I$ satisfy $p$

### Linear Time Logic (LTL)

- LTL properties are constructed from atomic propositions in $AP$: logical operators $\land, \lor, \neg$; and temporal operators $X, G, F, U$.
- The semantics of LTL is defined on paths:
  - Given a path $x$:
    - $x \models p$ iff $L(x_0, p)$
    - $x \models X p$ iff $x_1 \models p$
    - $x \models F p$ iff $\exists i \geq 0. x_i \models p$
    - $x \models G p$ iff $\forall i \geq 0. x_i \models p$
    - $x \models p U q$ iff $\exists i \geq 0. x_i \models q$ and $\forall j < i. x_j \models p$

### Computation Tree Logic (CTL)

- In CTL temporal properties use **path quantifiers**: $A$ : for all paths, $E$ : there exists a path
- The semantics of CTL is defined on states:
  - Given a state $s$
    - $s \models p$ iff $L(s, p)$
    - $s_0 \models EX p$ iff $\exists a path (s_0, s_1, s_2, ...). s_1 \models p$
    - $s_0 \models AX p$ iff $\forall paths (s_0, s_1, s_2, ...). s_1 \models p$
    - $s_0 \models EG p$ iff $\exists a path (s_0, s_1, s_2, ...). \forall i \geq 0. s_i \models p$
    - $s_0 \models AG p$ iff $\forall paths (s_0, s_1, s_2, ...). \forall i \geq 0. s_i \models p$
Linear vs. Branching Time

- **LTL** is a **linear time logic**
  - When determining if a path satisfies an LTL formula we are only concerned with a single path
- **CTL** is a **branching time logic**
  - When determining if a state satisfies a CTL formula we are concerned with multiple paths
  - In CTL, the computation is instead viewed as a computation tree which contains all the paths
  - The computation tree is obtained by unrolling the transition relation

The expressive powers of CTL and LTL are incomparable ($\text{LTL} \subseteq \text{CTL}^*$, $\text{CTL} \subseteq \text{CTL}^*$)
- Basic temporal properties can be expressed in both logics
- Not in this lecture, sorry! (Take a class on Modal Logics)

Recall the Example

This is a labeled transition system

One path starting at state (turn=0, pc1=10, pc2=20)

A computation tree starting at state (turn=0, pc1=10, pc2=20)

LTL Satisfiability Examples

- $\Diamond \neg p$ holds, $\Diamond p$ does not hold
- $\Diamond p$ holds, $\Diamond \neg p$ does not hold
- $\Diamond \neg p$ does not hold, $\Diamond p$ does not hold
- $\Diamond p$ does not hold, $\Diamond \neg p$ holds, $\Diamond (\Diamond p)$ does not hold
- $\Diamond \neg p$ holds, $\Diamond p$ holds, $\Diamond \neg p$ does not hold
- $\Diamond p$ holds, $\Diamond \neg p$ holds, $\Diamond (\Diamond p)$ does not hold

On this path:

- $F p$ holds, $G p$ does not hold, $p$ does not hold
- $X p$ does not hold, $X (X p)$ holds, $X (X (X p))$ does not hold
- $F p$ holds, $G p$ holds, $p$ holds
- $X p$ holds, $X (X p)$ holds, $X (X (X p))$ holds
CTL Satisfiability Examples

At state s:
- Holds: EF p, EX (EX p), AX (EX p)
- Does Not Hold: EF p, EX (EX p)

\[ \text{AF, AG, EF, EG, EX, AX} \]

Model Checking Complexity

- Given a transition system \( T = (S, I, R, L) \) and a CTL formula \( f \):
  - One can check if a state of the transition system satisfies the formula \( f \) in \( O(|f| \times (|S| + |R|)) \) time.
  - Multiple depth first searches (one for each temporal operator) = explicit-state model checking

- Given a transition system \( T = (S, I, R, L) \) and an LTL formula \( f \):
  - One can check if the transition system satisfies the formula \( f \) in \( O(2^{|f|} \times (|S| + |R|)) \) time.

State Space Explosion

- The complexity of model checking increases linearly with respect to the size of the transition system \( (|S| + |R|) \)
- However, the size of the transition system \( (|S| + |R|) \) is exponential in the number of variables and number of concurrent processes.
- This exponential increase in the state space is called the state space explosion.
- Dealing with it is one of the major challenges in model checking research.
Algorithm: Temporal Properties = Fixpoints

- States that satisfy \( \text{AG}(p) \) are all the states which are not in \( \text{EF}(\neg p) \) (= the states that can reach \( \neg p \))
- Compute \( \text{EF}(\neg p) \) as the fixed point of \( \text{Func}: 2^S \rightarrow 2^S \)
- Given \( Z \subseteq S \)
  - \( \text{Func}(Z) = \neg p \cup \text{reach-in-one-step}(Z) \)
  - or \( \text{Func}(Z) = \neg p \cup \text{EX}(Z) \)
- Actually, \( \text{EF}(\neg p) \) is the least-fixed point of \( \text{Func} \)
  - smallest set \( Z \) such that \( Z = \text{Func}(Z) \)
  - to compute the least fixed point, start the iteration from \( Z = \emptyset \), and apply the \( \text{Func} \) until you reach a fixed point
  - This can be computed (unlike most other fixed points)

Pictorial Backward Fixed Point

inverse image of \( \neg p = \text{EX}(\neg p) \)

Symbolic Model Checking

- Symbolic model checking represent state sets and the transition relation as Boolean logic formulas
- Fixed point computations manipulate sets of states rather than individual states
- Recall: we needed to compute \( \text{EX}(Z) \), but \( Z \subseteq S \)
- Fixed points can be computed by iteratively manipulating these formulas
- Use an efficient data structure for manipulation of Boolean logic formulas
- Binary Decision Diagrams (BDDs)
- SMV (Symbolic Model Verifier) was the first CTL model checker to use BDDs

Binary Decision Diagrams (BDDs)

- Efficient representation for boolean functions (a set can be viewed as a function)
- Disjunction, conjunction complexity: at most quadratic
- Negation complexity: constant
- Equivalence checking complexity: constant or linear
- Image computation complexity: can be exponential

Key Terms

- Counterexample guided abstraction refinement (CEGAR)
  - A successful software model-checking approach. Sometimes called "Iterative Abstraction Refinement".
- SLAM = The first CEGAR project/tool.
  - Developed at MSR
- Lazy Abstraction = CEGAR optimization
  - Used in the BLAST tool from Berkeley

Building Up To Software Model Checking via Counterexample Guided Abstraction Refinement

There are easily dozens of papers.

We will skim.
What is Counterexample Guided Abstraction Refinement (CEGAR)?

Verification by ...

Model Checking?

Theorem Proving?

Dataflow Analysis or Program Analysis?

Verification by **Model Checking**

Example ( ) {
1: do {
    lock();
    old = new;
2:    q = q->next;
3:    if (q != NULL) {
4:        q->data = new;
5:        unlock();
6:        new ++;
7:    } 
8:    while (new != old);
9:    unlock();
10:   return;
11: } }

1. (Finite State) Program
2. State Transition Graph
3. Reachability
   - Program = finite state model
   - State explosion
   - State exploration
   - Counterexamples

Precise [SPIN,SMV,Banana,ITF]

Verification by **Theorem Proving**

Example ( ) {
1: do {
    lock();
    old = new;
2:    q = q->next;
3:    if (q != NULL) {
4:        q->data = new;
5:        unlock();
6:        new ++;
7:    } 
8:    while (new != old);
9:    unlock();
10:   return;
11: } }

1. Loop Invariants
2. Logical Formulas
3. Check Validity
   - Invariant:
     lock ∧ new = old
     ¬lock ∧ new ≠ old

Verification by **Program Analysis**

Example ( ) {
1: do {
    lock();
    old = new;
2:    q = q->next;
3:    if (q != NULL) {
4:        q->data = new;
5:        unlock();
6:        new ++;
7:    } 
8:    while (new != old);
9:    unlock();
10:   return;
11: } }

1. Dataflow Facts
2. Constraint System
3. Solve Constraints
   - Imprecision: fixed facts
   - Abstraction
   - Type/Flow analyses

Precise [ESC,POC]

Verification by **Program Analysis**

Example ( ) {
1: do {
    lock();
    old = new;
2:    q = q->next;
3:    if (q != NULL) {
4:        q->data = new;
5:        unlock();
6:        new ++;
7:    } 
8:    while (new != old);
9:    unlock();
10:   return;
11: } }

1. Dataflow Facts
2. Constraint System
3. Solve Constraints
   - Imprecision: fixed facts
   - Abstraction
   - Type/Flow analyses

Scalable [Cqual,ESP]
Combining Strengths

Theorem Proving
- Need loop invariants (will find automatically)
- Behaviors encoded in logic (used to refine abstraction)
- Theorem provers (used to compute successors, refine abstraction)

Program Analysis
- Imprecise (will be precise)
- Abstraction (will shrink the state space we must explore)

Model Checking
- Finite-state model, state explosion (will find small good model)
- State space exploration (used to get a path sensitive analysis)
- Counterexamples (used to find relevant facts, refine abstraction)

For Next Time

- Read "Lazy Abstraction" - for the main ideas, ok to skim Sec. 7