Announcements

- Homework 0 (“Preliminaries”) out, due Friday Saturday

- This Week
  - Dive into research motivating CSCI 5535

- Next Week
  - Begin foundations
Course Summary

Course At-A-Glance

• Part I: Language Specification
  - Semantics = Describing programs
  - Evaluation strategies, imperative languages

• Part II: Language Design
  - Types = Classifying programs
  - Typed \(\lambda\)-calculus, functional languages

• Part III: Applications
Core Topics

- Semantics
  - Operational semantics
    - rules for execution on an abstract machine
    - useful for implementing a compiler or interpreter
  - Axiomatic semantics
    - logical rules for reasoning about the behavior of a program
    - useful for proving program correctness
  - Abstract interpretation
    - application: program analysis

- Types
  - $\lambda$-calculus
    - tiny language to study core issues in isolation

But first ...
First Topic: Model Checking

- Verify properties or find bugs in software
- Take an important program (e.g., a device driver)
- Merge it with a property (e.g., no deadlocks)
- Transform the result into a boolean program
- Use a model checker to exhaustively explore the resulting state space
  - Result 1: program provably satisfies property
  - Result 2: program violates property “right here on line 92,376”!

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
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
- Software model checking
- Counterexample feasibility
- Abstraction refinement weakest pre, thrm p unauthorized

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”.

**Spoiler**

- This stuff really works!

- **Symbolic model checking** is a massive success in the model-checking field
- SLAM took the PL world by storm
  - Spawned multiple copycat projects
  - Launched Microsoft's Static Driver Verifier (released in the Windows DDK)

---

**Model Checking**

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

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

*Symbolic Model Checking* by Ken McMillan.

*We will skim.*
What is Model Checking?  Keywords?

David: Specification = what you want a program to do

Sound → Verifier

Ok, your program is “good” = satisfies satisfaction
(a.e., no deadlocks)

“I think there is a bug”

How we approximate

Abstraction

- exhaustive exploration of a state space
- error paths = counterexamples

What is Model Checking?

“state space explosion problem” = “everything is exponential”

Large number of states in the model

“temporal logic” concepts:

- eventually
- always
- until

“transition system” transitions
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 of a system holds
Verification vs. Falsification

- An automated verification tool
  - can report that the system is verified (with a proof);
  - or that the system was not verified.

- When the system was not verified, it would be helpful to explain why.
  - Model checkers can output an error counterexample: a concrete execution scenario that demonstrates the error.

- Can view a model checker as a falsification tool
  - The main goal is to find bugs

- So what can we verify or falsify?

Temporal Properties

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

- **Invariant**($p$)
  is true in a state if property $p$ is true in every state on all execution paths starting at that state

  \[ G, AG, \Box \text{ ("globally" or "box" or "forall")} \quad AG(\neg p \equiv 0) \]

- **Eventually**($p$)
  is true in a state if property $p$ is true at some state on every execution path starting from that state

  \[ F, AF, \Diamond \text{ ("future" or "diamond" or "exists")} \quad F(p \equiv 45) \]
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);
       // critical section
   12:   work(); turn = 1;
   13: }

11: // concurrently with

20: while (true) {
   21:   wait(turn == 1);
       // critical section
   22:   work(); turn = 0;
   23: }

Next: formalize this intuition ...

Reachable States of the Example Program

Each state is a valuation of all the variables: turn and the two program counters for two processes
Analyzed System is a Transition System

- Labeled transition system $T = (S, I, R, L)$
  - $S$ = Set of states // standard FSM
  - $I \subseteq S$ = Set of initial states // standard FSM
  - $R \subseteq S \times S$ = Transition relation // standard FSM
  - $L: S \to 2^{\text{AP}}$ = Labeling function // this is new!

- **AP**: Set of atomic propositions (e.g., “x=5” $\in \text{AP}$)
  - 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

- “Eventually the first process enters the critical section”
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

\[ \text{Invariant}(\neg(pc1=12 \land pc2=22)) \]

- “Eventually the first process enters the critical section”

\[ \text{Eventually}(pc1=12) \]

Temporal Logics

There are four basic temporal operators:

- \( X \ p \)
  - \( \text{Next} \ p \), \( p \) holds in the next state
- \( G \ p \)
  - \( \text{Globally} \ p \), \( p \) holds in every state, \( p \) is an invariant
- \( F \ p \)
  - \( \text{Future} \ p \), \( p \) will hold in a future state, \( p \) holds eventually
- \( p \ U \ q \)
  - \( \text{Until} \ q \), 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
  \((s_0, s_1, s_2, \ldots)\), such that \(\forall i \geq 0. (s_i, s_{i+1}) \in R\)
- A path \((s_0, s_1, s_2, \ldots)\) is an **execution path** if \(s_0 \in I\)
- Given a path \(h = (s_0, s_1, s_2, \ldots)\)
  - \(h_i\) denotes the \(i^{th}\) state: \(s_i\)
  - \(h^i\) denotes the \(i^{th}\) suffix: \((s_i, s_{i+1}, s_{i+2}, \ldots)\)

- In some temporal logics one can quantify paths starting from a state using **path quantifiers**
  - \(A\) : for all paths (e.g., \(A\ h. \ldots\))
  - \(E\) : there exists a path (e.g., \(E\ h. \ldots\))

Paths and Predicates

- We write
  \[ h \models p \]
  “the path \(h\) makes the predicate \(p\) true”
  - \(h\) is a path in a transition system
  - \(p\) is a temporal logic predicate

- Example:
  \[ A \ h. \quad h \models G (\neg(pc1=12 \land pc2=22)) \]
  Mean thread 1 and thread 2 in the critical section at the same time
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 $h$:

$$h \models p$$

### LTL Semantics

- $h \models \text{ap}$ iff $L(h_0, \text{ap})$ where $L : S \rightarrow 2^{AP}$ is the labelling function.
- $h \models \text{X } p$ iff $h^1 \models p$ where $h^1$ denotes the next suffix.
- $h \models \text{F } p$ iff $h^i \models p$ for some $i \geq 0$.
- $h \models \text{G } p$ iff $h^i \models p$ for all $i \geq 0$.
- $h \models p \text{ U } q$ iff $h^i \models q$ for some $i \geq 0$ and $h^j \models p$ for all $j < i$.
Linear Time Logic (LTL)

- LTL properties are constructed from atomic propositions in AP; logical operators $\wedge$, $\lor$, $\neg$; and temporal operators $X$, $G$, $F$, $U$.
- The semantics of LTL is defined on paths:

  Given a path $h$:

  \[
  h \models p \iff L(h_0, \text{ap}) \quad \text{atomic prop}
  \]

  \[
  h \models X p \iff h^1 \models p \quad \text{next}
  \]

  \[
  h \models F p \iff \exists i \geq 0. \ h^i \models p \quad \text{future}
  \]

  \[
  h \models G p \iff \forall i \geq 0. \ h^i \models p \quad \text{globally}
  \]

  \[
  h \models p \cup q \iff \exists i \geq 0. \ h^i \models q \text{ and } \forall j<i. \ h^j \not\models p \quad \text{until}
  \]

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 $p$.
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 \( s \)
  \[ s \models ap \iff L(s, ap) \]
  \[ s_0 \models EX p \iff \exists \text{ a path } (s_0, s_1, s_2, ...). s_1 \models p \]
  \[ s_0 \models AX p \iff \forall \text{ paths } (s_0, s_1, s_2, ...). s_1 \models p \]
  \[ s_0 \models EG p \iff \exists \text{ a path } (s_0, s_1, s_2, ...). \forall i \geq 0. s_i \models p \]
  \[ s_0 \models AG p \iff \forall \text{ paths } (s_0, s_1, s_2, ...). \forall i \geq 0. s_i \models p \]

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 expressive powers of CTL and LTL are incomparable (\( LTL \subseteq CTL^*, CTL \subseteq CTL^* \))
- 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

Linear vs. Branching Time

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

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

On this path:

Holds

Does Not Hold
LTL Satisfiability Examples

○ p does not hold  ● p holds

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

CTL Satisfiability Examples

○ p does not hold  ● p holds

At state s:

Holds  Does Not Hold
CTL Satisfiability Examples

At state $s$:
- EF $p$, EX (EX $p$), AF ($p$), $p$ holds
- AF $p$, AG $p$, AG ($p$), EX $p$, EG $p$, p does not hold

At state $s$:
- EF $p$, AF $p$, EX (EX $p$), EX $p$, EG $p$, $p$ holds
- AG $p$, AG ($p$), AF ($p$) does not hold

At state $s$:
- EF $p$, AF $p$, AG ($p$), EX $p$, EG $p$, p holds
- EG ($p$), EF ($p$) does not hold
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
**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 \(AG(p)\) are all the states which are not in \(EF(\neg p)\) (= the states that can reach \(\neg p\)).
- Compute \(EF(\neg p)\) as the **fixed point** of \(Func: 2^S \rightarrow 2^S\).
- Given \(Z \subseteq S\),
  - \(Func(Z) = \neg p \cup \text{reach-in-one-step}(Z)\) \(\text{Called the inverse image of } Z\)
- Actually, \(EF(\neg p)\) is the **least-fixed point** of \(Func\).
  - smallest set \(Z\) such that \(Z = Func(Z)\).
  - to compute the least fixed point, start the iteration from \(Z=\emptyset\), and apply the \(Func\) until you reach a fixed point.
  - This can be **computed** (unlike most other fixed points).
Pictorial Backward Fixed Point

\[ \text{(Inverse Image of } \neg p) = \text{EX}(\neg p) \]

\[ \text{(initial states that violate } AG(p)) \]
\[ = (\text{initial states that satisfy } EF(\neg p)) \]
\[ = (\text{states that can reach } \neg p = EF(\neg p)) \]
\[ = (\text{states that violate } AG(p)) \]

This fixed point computation can be used for:
- verification of EF(\neg p)
- or falsification of AG(p)

... and similar fixed points handle the other cases

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 reach-in-one-step(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

Building Up To

Software Model Checking via Counterexample Guided Abstraction Refinement

There are easily dozens of papers.

We will skim.
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.

What is Counterexample Guided Abstraction Refinement (CEGAR)?

Verification by ...

Model Checking?

Theorem Proving?

Dataflow Analysis or Program Analysis?
Verification

Example ( ) {
    1: do{
            lock();
            old = new;
            q = q->next;
        }
        if (q != NULL){
            2: q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    5: unlock();
    return;
}

Is this program correct?

What does correct mean?

How do we determine if a program is correct?

Verification by Model Checking

Example ( ) {
    1: do{
            lock();
            old = new;
            q = q->next;
        }
        if (q != NULL){
            2: q->data = new;
            unlock();
            new ++;
        }
    } while(new != old);
    5: unlock();
    return;
}

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

Precise [SPIN,SMV,Bandera,JPF]

51

52
Verification by Theorem Proving

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

1. Loop Invariants
2. Logical Formulas
3. Check Validity

Invariant:

$\neg \text{lock} \land \text{new} = \text{old}$
$\lor$
$\neg \text{lock} \land \text{new} \neq \text{old}$

Verification by Theorem Proving

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

1. Loop Invariants
2. Logical Formulas
3. Check Validity

- Loop invariants
- Multithreaded programs
+ Behaviors encoded in logic
+ Decision procedures

Precise [ESC,PCC]
Verification by **Program Analysis**

**Example**

```c
Example ( ) {
1:  do{
    lock(); /*
        old = new;
>next;  q = q-
2:   if (q != NULL){
  3:     q->data = new;
        unlock();
  new ++;
    }
4:  } while(new != old);
5:  unlock();
   return;
}
```

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

• Post about today’s class and reading
• Read “Lazy Abstraction”
  - for the main ideas, ok to skim Sec. 7