CS 5371 - Software Safety & Risk Analysis

Lecture 5: Software Model Checking

SOFTWARE MODEL CHECKING
Motivation

- Software plays a major role in our daily life.
- Errors in software can be fatal.
- Conventional verification techniques, e.g., testing, are not always adequate.
  - Software errors cost U.S. economy $59.5 billion annually.
  - $22.2 billion can be saved to U.S. economy if verification is done at earlier stages (NIST, June 2002).
- Formal verification techniques are effective at detecting errors
  - Theorem Provers
  - Model Checkers
  - Run-Time Monitors

Formal Verification Techniques

- Provide extra layer of assurance over testing and simulation
- Discover errors missed by traditional techniques
  - Theorem Proving
  - Runtime Monitoring
  - Model Checking
Theorem Proving

- Specify system and properties as logic formulas
- Derive properties from system

- System:
  - A -> B
  - B -> C
  - A

- Property:
  - C

Runtime Monitoring

- Specify system properties in temporal logic
- Verify system correctness at runtime
- Report violations

- System:
  - while result < 0 {
    - if x > 5 then
      - x ← x + 1
    - result ← result + 1
  }

- Property
  - G(x < 100)
Temporal Logic Model Checking

- Model checking is an automatic verification technique for finite state concurrent systems.
- Developed independently by Clarke and Emerson and by Queille and Sifakis in early 1980's.
- Specifications are written in propositional temporal logic. (Pnueli 77)
- Verification procedure is an intelligent exhaustive search of the state space of the design.

Advantages of Model Checking

- No proofs!!! (Algorithmic rather than Deductive)
- Fast (compared to other rigorous methods such as theorem proving)
- Diagnostic counterexamples
- No problem with partial specifications
- Logics can easily express many concurrency properties
Temporal Logic Model Checking

Main Disadvantage

State Explosion Problem:

- 2-bit counter

- n-bit counter has $2^n$ states
Main Disadvantage - Cont

n states, m processes

n^m states

Temporal Logic Model Checking

G is always followed by R or Y

Temporal Property

System Model

Model Checker

YES

NO, Counter Example
Temporal Logic Model Checking

Model (System Requirements)  \[ \text{Model Checker} \quad M \models \phi \]  Specification (System Property)  

Answer:
Yes, if model satisfies specification
Counterexample, otherwise

For increasing our confidence in the correctness of the model:
- Verification: The model satisfies important system properties
- Debugging: Study counter-examples, pinpoint the source of the error, correct the model, and try again

Mutual Exclusion Example

- Two process mutual exclusion with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available (S\(_0\)) or taken (S\(_1\))
- Initially both processes are in the Non-critical state and the semaphore is available --- \(N_1 N_2 S_0\)

\[
\begin{align*}
N_1 & \rightarrow T_1 \\
T_1 \land S_0 & \rightarrow C_1 \land S_1 \\
C_1 & \rightarrow N_1 \land S_0 \\
N_2 & \rightarrow T_2 \\
T_2 \land S_0 & \rightarrow C_2 \land S_1 \\
C_2 & \rightarrow N_2 \land S_0
\end{align*}
\]

**Mutual Exclusion Example**

*Initially both processes are in the Non-critical state and the semaphore is available --- N<sub>1</sub> N<sub>2</sub> S<sub>0</sub>*

\[
\begin{align*}
N_1 &\rightarrow T_1 \\
T_1 \land S_0 &\rightarrow C_1 \land S_1 \\
C_1 &\rightarrow N_1 \land S_0 \\
N_2 &\rightarrow T_2 \\
T_2 \land S_0 &\rightarrow C_2 \land S_1 \\
C_2 &\rightarrow N_2 \land S_0
\end{align*}
\]

**Specification – Desirable Property**

*No matter where you are there is always a way to get to the initial state*

\[
K \models GF (N_1 \land N_2 \land S_0)
\]
Mutual Exclusion Example

Model (System Requirements)

\[ M \models \varphi \]

Specification (System Property)

\[ K \models \text{GF (N}_1 \land N_2 \land S_0) \]

Answer: Yes

A Proof: For All possible behaviors
Mutual Exclusion Example
Mutual Exclusion Example
Specification – Desirable Property

No matter where you are there is no way to get to the initial state

\[ K \models GF (N_1 \land N_2 \land S_0) \]
Kripke Structure
\[ K = < S, P, R > (M = < S, P, R, L, (s_0)> ) \hspace{1cm} (s_0 \in S \text{ - initial state}) \]

- **S**: the set of possible global states
- **P**: a non-empty set of atomic propositions \( \{p_1, \ldots, p_k\} \) which express atomic properties of the global states, e.g., being an initial state, being an accepting state, or that a particular variable has a special value.
- **R**: \( S \times S \) a transition relation s.t. \( R(s,s') \) if \( s \) to \( s' \) is a possible atomic transition
- **L**: \( S \rightarrow 2^P \): a labeling function which defines which propositions hold in which states.
- **State explosion problem**: The size of \( S \) is often exponential in requirements/design.

---

Properties

- **Safety properties**
  - Invariants, deadlocks, reachability, etc.
  - Can be checked on finite traces
  - “something bad never happens”

- **Liveness Properties**
  - Fairness, response, etc.
  - Infinite traces
  - “something good will eventually happen”
Specification Language: Linear Temporal Logic (LTL)

- Widely used property specification language.
- Expressibility allows modeling of software properties such as liveness and safety.
- Applicable to numerous verification tools
  - Model checkers Spin, NuSMV, and Java Pathfinder
  - Runtime verification of Java programs

LTL Syntax:

- Atoms: atomic propositions as in propositional logic
- Formulas (Q is an LTL formula):
  \[ Q ::= \text{true} \mid \text{false} \mid (\neg Q) \mid (Q \rightarrow Q) \mid (Q \land Q) \mid (Q \lor Q) \mid (X Q) \mid (F Q) \mid (G Q) \mid (Q U Q) \mid (Q W Q) \]
  - \( p \in \text{ATOMS} \)
  - \( X \) (Next)
  - \( F \) (Future, <>)
  - \( G \) (Global [])
  - \( U \) (Until)
  - \( W \) (Weak until)
LTL Formulas (Manna et. al. 89)

- "True" is a well-formed LTL formula.
- "False" is a well-formed LTL formula.
- If $a$ and $b$ are well-formed LTL formulas then so is:
  - $\neg a$
  - $a \lor b$
  - $a \land b$
  - $a \mathbf{U} b$ (a until b)
  - $X a$ (next a)
  - $\Diamond a$ (eventually a)
  - $\Box a$ (always a)

Examples

- $(((Fp) \land (Gq)) \rightarrow (p W r))$
- $(F (p\rightarrow(Gr)) \lor(( \neg q) \mathbf{U} p))$
- $(p W (q W r))$
- $((G(Fp)) \rightarrow (F (q \lor s)))$
Examples:

- The printer is always ready:
  - $G \ p$

- The printer will eventually be ready:
  - $F \ p$

- The printer will be ready until a job arrives:
  - $p \ U \ j$

- After a job arrives, sooner or later the printer will be ready
  - $G(j \rightarrow F \ p)$

Clause 10: Until

- $P \ U \ Q$
  - $P \ P \ P \ Q \ _ \ _$  TRUE
  - $P \ Q \ _ \ _$  FALSE
  - $Q \ _ \ _ \ _ \ _ \ _$  TRUE
  - $P \ P \ P \ P \ P \ P$  FALSE
  - $Q \ Q \ Q \ P \ _ \ _$  TRUE
  - $Q \ Q \ Q \ Q \ Q$  TRUE
  - $Q \ Q \ Q \ (PQ) \ Q$  TRUE

- The first thing holds continuously until the second thing holds.
- They don’t have to hold in the same state
- The second thing has to occur
Examples

- It is impossible to get to a state where we are started but not ready.
  - $G \neg(\text{started} \land \text{ready})$
- If a request is made, it will be serviced
  - $G(\text{requested} \rightarrow F \text{ serviced})$
- P is enabled infinitely often on every path
  - $G F \text{ enabled}$
- Whatever happens, P will eventually become permanently deadlocked
  - $F G \text{ deadlock}$
- If the process is enabled infinitely often, then it runs infinitely often
  - $G F \text{ enabled} \rightarrow G F \text{ running}$
- An elevator moving up at the second floor does not go down if it has passengers traveling to the 5th floor
  - $G (\text{floor2} \land \text{directionup} \land \text{Button5}) \rightarrow (\text{directionup U Floor5})$

Tomorrow

- No class 😊
- Assignments 😒
  - Suggest a paper for your brief (7-10 mins) presentation (no slides).
    - Place in dropbox folder (students_research_papers)
    - Due by 11:59 am On Thursday
  - Team final project
    - Prepare a proposal project
    - On Thursday, you will be asked to clearly describe the system and the properties you aim to check
  - Install NuSMV model checker on your laptop and bring to class on Thursday
    - [http://nusmv.fbk.eu/NuSMV/download/getting-v2.html](http://nusmv.fbk.eu/NuSMV/download/getting-v2.html)