Temporal Logics for Reactive Systems

[Pnueli FOCS 77, TCS 81]

Transformational systems

```
get input;
compute something;
return result;
```

- Transformational view follows from the initial use of computers as advanced calculators: A component receives some input, does some calculation and then returns a result.

Reactive systems

```
while (true) {
    receive some input,
    send some output
}
```

- Nowadays, the reactive system view seems more natural: components which continuously interact with each other and their environment without terminating
Transformational vs. Reactive Systems

Transformational systems
get input;
{pre-condition}
compute something;
{post-condition}
return result;

Reactive systems
while (true) {
    receive some input,
    send some output
}

• Earlier work in verification uses
  the transformational view:
  – halting problem
  – Hoare logic
  – pre and post-conditions
  – partial vs. total correctness

• For reactive systems:
  – termination is not the main
    issue
  – pre and post-conditions are
    not enough

Temporal Logics

Temporal Logics
• Invariant $p$ ($G p$, $AG p$, $\square p$)
• Eventually $p$ ($F p$, $AF p$, $\diamond p$)
• Next $p$ : ($X p$, $AX p$, $\Diamond p$)
• $p$ Until $q$ : ( $p U q$, $A(p U q)$ )

Branching vs. Linear Time

Transition system:

LTL view
$G(p)$
$F(p)$

CTL view
$AF(p)$, $EG(p)$
Automated Verification of Finite State Systems
[Clarke and Emerson 81], [Queille and Sifakis 82]

Transition Systems

- $S$: Set of states (finite)
- $I \subseteq S$: Set of initial states
- $R \subseteq S \times S$: Transition relation

Verification vs. Falsification

Verification:
show: initial states $\subseteq$ truth set of $p$

Falsification:
find: a state $\in$ initial states $\cap$ truth set of $\neg p$
generate a counter-example starting from that state

Model checking problem: Given a temporal logic property, does the transition system satisfy the property?
- Complexity: linear in the size of the transition system

Temporal Properties $\equiv$ Fixpoints
[Emerson and Clarke 80]

$\text{EF}(\neg p) \equiv$ states that can reach $\neg p \equiv \neg p \cup \text{Pre}(\neg p) \cup \text{Pre(Pre}(\neg p)) \cup \ldots$

$\text{EG}(\neg p) \equiv$ states that can avoid reaching $p \equiv \neg p \cap \text{Pre}(\neg p) \cap \text{Pre(Pre}(\neg p)) \cap \ldots$

$\text{EF}(\neg p)$
initial states that satisfy $\text{EF}(\neg p)$
$\equiv$ initial states that violate $\text{AG}(p)$

$\text{EG}(\neg p)$
initial states that satisfy $\text{EG}(\neg p)$
$\equiv$ initial states that violate $\text{AF}(p)$
Symbolic Model Checking

[McMillan et al. LICS 90]

- Represent sets of states and the transition relation as Boolean logic formulas

- Fixpoint computation becomes formula manipulation
  - pre and post-condition computations: Existential variable elimination
  - conjunction (intersection), disjunction (union) and negation (set difference), and equivalence check

- Use an efficient data structure
  - Binary Decision Diagrams (BDDs)

SMV [McMillan 93]

- BDD-based symbolic model checker
- Finite state
- Temporal logic: CTL
- Focus: hardware verification
  - Later applied to software specifications, protocols, etc.
- SMV has its own input specification language
  - concurrency: synchronous, asynchronous
  - shared variables
  - boolean and enumerated variables
  - bounded integer variables (binary encoding)
    - SMV is not efficient for integers, can be fixed
LTL Properties $\equiv$ Büchi automata
[Vardi and Wolper LICS 86]

- Büchi automata: Finite state automata that accept infinite strings

- A Büchi automaton accepts a string when the corresponding run visits an accepting state infinitely often

- The size of the property automaton can be exponential in the size of the LTL formula

\[ G\ p \]
\[ F\ p \]
\[ G\ (F\ p) \]

SPIN [Holzmann 91, TSE 97]

- Explicit state, finite state
- Temporal logic: LTL
- Input language: PROMELA
  - Asynchronous processes
  - Shared variables
  - Message passing through (bounded) communication channels
  - Variables: boolean, char, integer (bounded), arrays (fixed size)

- Property automaton from the negated LTL property
- Product of the property automaton and the transition system (on-the-fly)
- Show that there is no accepting cycle in the product automaton
- Nested depth first search to look for accepting cycles
- If there is a cycle, it corresponds to a counterexample behavior that demonstrates the bug
Model Checking Research

• These key ideas and tools inspired a lot of research
  [Clarke, Grumberg and Peled, 99]
  – efficient symbolic representations
  – partial order reductions
  – abstraction
  – compositional/modular verification
  – model checking infinite state systems (pushdown automata)
  – model checking real time systems
  – model checking hybrid systems
  – model checking programs
  – ...

Model Checking Impact

• Model checking research had significant impact in other areas. Some examples:
  • Software Engineering:
    – Chaki et al. "Modular Verification of Software Components in C" ICSE 03, ACM SIGSOFT distinguished paper
    – Betin Can at al. "Application of Design for Verification with Concurrency Controllers to Air Traffic Control Software" ASE 05 best paper
  • Systems:
  • Also conferences in Security and Programming Languages have plenty of model checking papers nowadays!
Other issues

• Abstraction
• Bounded model checking
• Dealing with infinite-state transition system
• Automated synthesis

Abstract Interpretation [Cousot and Cousot POPL 77]

• Abstract interpretation provides a general framework for defining abstractions
• The size of the state space of an abstracted system is smaller than the original system, which makes static analysis of the abstract state space feasible
• Different abstract domains can be combined using the abstract interpretation framework
• Abstract interpretation framework also provides conservative approximation techniques such as widening for computing approximations of fixpoints
Predicate Abstraction \cite{Graf and Saidi CAV 97}

- An automated abstraction technique that reduces the state space of a program by removing some variables from the program and just keeping information about a set of predicates about them
- Given a program and a set of predicates, predicate abstraction abstracts the program so that only the information about the given predicates are preserved
- The abstracted program adds nondeterminism since in some cases it may not be possible to figure out what the next value of a predicate will be based on the predicates in the given set
- One needs an automated theorem prover to compute the abstraction

Counter-example Guided Abstraction Refinement \cite{Clarke et al. CAV 00, Ball and Rajamani SPIN 00}

The basic idea in counter-example guided abstraction refinement is the following:

- First look for an error in the abstract program (if there are no errors, we can terminate since we know that the original program is correct)
- If there is an error in the abstract program, generate a counter-example path on the abstract program
- Check if the generated counter-example path is feasible using a theorem prover.
- If the generated path is infeasible add the predicate from the branch condition where an infeasible choice is made to the predicate set and generate a new abstract program.
Bounded Model Checking [Biere et al. TACAS 99]

• Represent sets of states and the transition relation as Boolean logic formulas

• Instead of computing the fixpoints, unroll the transition relation up to certain fixed bound and search for violations of the property within that bound

• Transform this search to a Boolean satisfiability problem and solve it using a SAT solver