Motivation

- Software systems increasingly relied upon in many critical domains
  - e.g., aeronautics, banking
- If system contains an error, then it could lead to serious harm to peoples’ lives and livelihoods
  - e.g., people could be injured or killed, money could be lost
- Thus systems need to be validated to gain assurance that the systems satisfy their properties
  - One common validation approach is finite-state verification techniques that algorithmically check whether or not all potential executions of a system satisfy a given property

Architecture of Finite State Verifier

- System
- Property
- Property Translator
- Property Representation
- Verification Algorithm
- System Model
- Property verified
- Property not verified
- Counterexample execution

Examples of Properties

- No deadlock
- Mutual exclusion
- Always define variable v before use variable v
- For an elevator controller, always close doors before move
- For a file, never throw an IO exception

Examples of Finite-State Verification (FSV) Tools

- Based on techniques such as:
  - Reachability, e.g., Spin, SMV, LTSA
  - Linear programming, e.g., INCA
  - Data flow analysis, e.g., FLAVERS
  - Ada, Java, or Little-JIL

History of Data Flow Analysis for Verification

- Mid-70’s: Originally proposed for def-ref anomalies in FORTRAN systems (Osterweil and Fosdick)
- Early 80’s: Extended to general properties (Olender and Osterweil) & concurrency (Taylor and Osterweil)
- 90’s primarily for properties of Ada systems
  - Deadlock detection (Masciolla and Ryder)
  - Efficient representation of concurrency & incremental precision improvement (Dwyer and Clarke)
- Recent: Optimizations, Java systems (Avrunin, Clarke, Cobleigh, Naumovich, and Osterweil)
Architecture of FLAVERS

Example: Elevator Controller

Properties

Example: All Property

Representing System Models
Abstracting System Models

- TFG abstracts information to be tractable, e.g.,
- Only model variables relevant to property
- Abstract values of variable, e.g.,
- Concrete \( x \) is Integer
- Abstract \( x \) is \((x<0, x==0, x>0)\)
- Conservative abstractions usually overapproximate behavior

Example: System Model as TFG

```java
public class Elevator {
    boolean stopped;
    ...
    public static void main() {
        ...
        1: if (stopped) {
            2: openDoors();
        }
        3: if (stopped) {
            4: closeDoors();
        }
        5: moveToNextFloor();
        } // end of main
    } // end of Elevator
}
```

Overview of Verification Algorithm: State Propagation Algorithm

- Given a system modeled as a TFG and a given property represented as an FSA
- Each node of the TFG is associated with the states of the property that the system could be in at that point in the system
- Data flow analysis propagates states through the nodes
  - Since there are a finite number of states and nodes, a fixed point will be reached and the verification results can then be determined

Details about the State propagation Algorithm

- Initially, the start state of the property is associated with the initial node of the TFG
- On each iteration, update the set of states associated with the current node
  - Apply the event annotating the current node to all sets of states associated with all previous nodes of the TFG
- When fixed point is reached, the verification results are determined by considering the set of states associated with the final node of the TFG
  - An all property is verified if only accepting states
  - A none property is verified if only non-accepting states

Example: State-Propagation for Initial Node 1

Example: State-Propagation for Node 2
Example: State-Propagation for Node 4
Worklist: 1, 2, 3, 4, 5

Example: State-Propagation
Fixed Point
Worklist: 1, 2, 3, 4, 5

Example: State-Propagation
Determining Verification Results
Worklist: 1, 2, 3, 4, 5

Interpreting Verification Results
- If property verified, property satisfied for all possible traces of the system
- If property not verified:
  - A real counterexample that illustrates property violation
  - Modeling error found (in the system or in the property)
  - OR
    - A spurious result when inconsistency relies upon overapproximations of system model
    - e.g. every counterexample corresponds to an infeasible path

Incrementally Adding Precision to System Models
- Constraints describe conditions necessary for feasible execution represented as FSAs
- Special violation state is entered when an infeasible path is detected
- Violation is a trap state; once it is entered, never leave that state

Example: Variable Constraint for stopped
+ Example: More Precise TFG with Stopped events

public class Elevator {
  boolean stopped;
  ...
  public static void main() {
    if (stopped) {
      openDoors();
    }
    if (stopped) {
      closeDoors();
    }
    moveToNextFloor();
  }
}

+ Architecture of FLAVERS Incorporating Constraints

+ Other Examples of Constraints

- Automatically generated
- Variable constraint tracks value of given variable
- Supported types of variables are boolean, enumerated, integer range
- Task constraint tracks program counter of given task
- User-defined, e.g., assumptions about environment

+ State Propagation Algorithm Incorporating Constraints

- Each node associated with a set of tuples
- Each tuple has a position for each property FSA and each constraint FSA
- If the current node is associated with a current tuple where a constraint FSA reached its violation state, then that tuple is not propagated to any next nodes
- Result looks at paths that are feasible with respect to the constraints
  - The property state is the same as before
  - Every constraint must be in an accepting state
Example: State-Propagation

Example: State-Propagation

Results

Discussion about FLAVERS

- Overall complexity is $O(N^2 \cdot S)$
- $N$ is the # nodes in the model
- $S$ is the number of states: property x constraints
- More precisely $O(N^2 \cdot SP \cdot SC1 \cdot \ldots \cdot SCn)$
- In our experience, many important properties can be proven with a small number of constraints
- Experimentally: performance sub-cubic

Evaluation of FLAVERS

- Applied to collection of concurrent and sequential systems such as
  - elevator, dining philosophers, reader writers, producers consumers, Chiron user interface
- Measured
  - Size of system model
  - Number of the TFG nodes and edges
  - Number of constraints needed
  - Performance in terms of space and time

Benefits of FLAVERS

- Data Flow Analysis determines if the property is valid or not
- Efficient
- Always terminates
- Conservative
- Only validates the property if it is true for all/no possible executions
- When it can not validate the property, it provides a counter example trace
- Relatively easy to use
- Relatively easy to write properties compared to predicate calculus or temporal logic
- Do not have to understand how the system works

Drawbacks of FLAVERS

- Cannot express some properties of interest
  - DeadLock
  - Compound data types, e.g., for all $I, A[I] > A[I+1]$
  - Some counting, e.g. # Inserts > # Deletes
- Infeasible paths
- Usually requires several iterations to determine needed constraints
Comparative Evaluation of FSV Tools

p07 Comparison (Original)

Comparative Evaluation of FSV Tools

p07 Comparison (Decomposed)

Some Research Directions for FSV

- Support for specifying properties, e.g.,
  - Property patterns
- Support for modeling systems, e.g.,
  - eliminating infeasible paths by employing such techniques as symbolic execution or theorem proving
  - abstracting variable values
- Support for optimizing verification algorithms, e.g.,
  - Alphabet refinement, partial order reduction, symbolic representations
- Support for visualizing counterexample traces

References


Demonstration of FLAVERS for Java...