+

FLAVERS (**FLow Analysis for VERification of Systems)**

http://laser.cs.umass.edu/tools/flavers

Based on slides by Lori Clarke
Presented by Heather Conboy

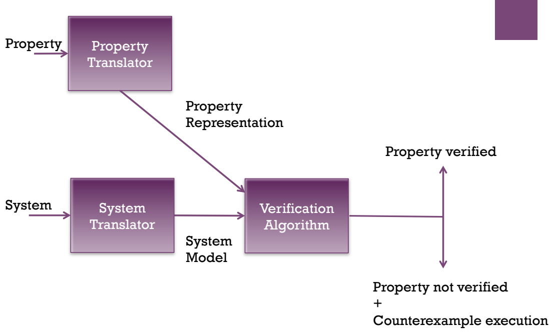
+ 2

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

+ 3

Architecture of Finite State Verifier



+ 4

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

+ 5

Examples of Finite-State Verification (FSV) Tools

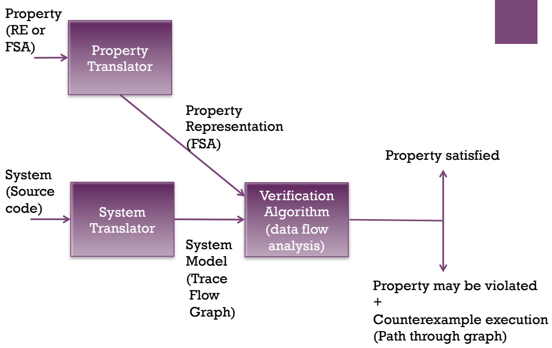
- Based on techniques such as:
  - Reachability, e.g, Spin, SMV, LTSA
  - Linear programming, e.g., INCA
  - **Data flow analysis, e.g., FLAVERS**
    - Ada, Java, or Little-JIL

+ 6

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 (Masticola 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

- A building with multiple floors has an elevator controller in charge of a single car
  - The passengers must be able to safely use the elevator car
- The car is initially stopped
  - The car may move between floors
- The car doors are initially closed
  - The car doors may be opened or closed

FLAVERS is Event-based

- **Recognizable events such as**
  - **Method calls**
    - **e.g., Close must be called before move**
  - **Thread interactions**
    - **e.g., Start must be called before Join**
  - **Arbitrary operations**
    - **e.g., S=true, IOException thrown**
- **Need to be able to treat events as indivisible actions**
  - **e.g., can treat close and move as atomic as long as they do not contain any events of concern**

Properties

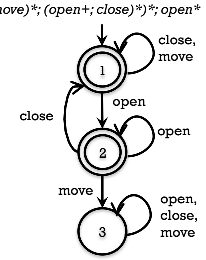
- Many interesting and important properties can be specified as a regular language and then represented as an FSA
  - Directly specified as FSA
  - Specified as quantified regular expression (QRE) and then converted from a QRE to an FSA
- Quantified means either:
  - An **all** property is a behavior that must always happen on all possible executions
  - A **none** property is a behavior that must never happen on any possible execution

Example: All Property

NATURAL LANGUAGE DESCRIPTION:
The elevator always closes its doors before moving

QUANTIFIED REGULAR EXPRESSION:
for events { open, close, move}
show all executions satisfy
((close | move); (open+; close)*)*; open**

FSA:



Representing System Models

- **Trace Flow Graph (TFG)**
  - **collection of annotated control flow graphs**
  - **intertask communication and interleavings are represented with additional nodes & edges**
  - **does not enumerate all reachable system states**
- **Conservative but over-approximates actual executable behaviors**
  - **All actual executions correspond to at least one potential execution**
  - **Some potential executions do not correspond to any actual execution**

Abstracting System Models

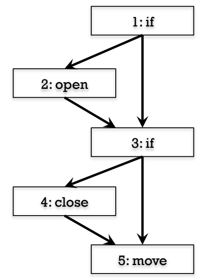
13

- 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

14

```
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

15

- 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

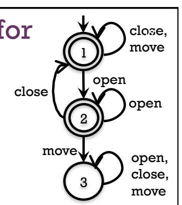
Details about the State propagation Algorithm

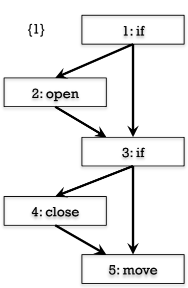
16

- 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

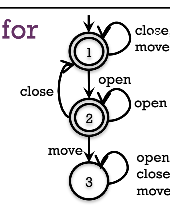
Worklist: ~~1~~

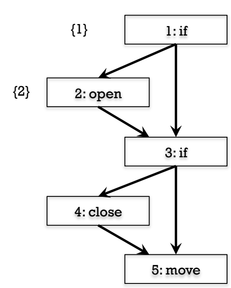




Example: State-Propagation for Node 2

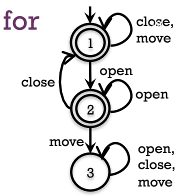
Worklist: ~~1~~, ~~2~~, 3

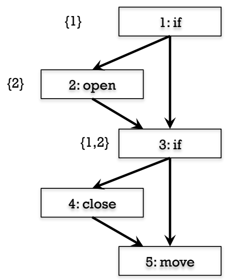




Example: State-Propagation for Node 4

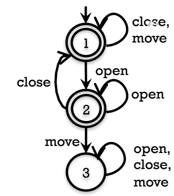
Worklist: ~~1~~, ~~2~~, ~~3~~, 4, 5

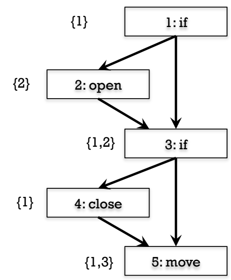




Example: State-Propagation Fixed Point

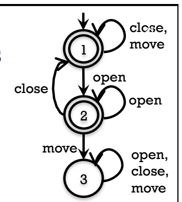
Worklist: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~

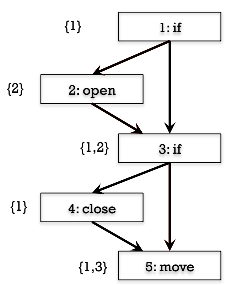




Example: State-Propagation Determining Verification Results

Worklist: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~





Property not verified because final node 5 is associated with property non-accepting state 3

Counterexample execution is 1, 2, 3, 5

+ Interpreting Verification Results

22

- If property verified, property satisfied for all possible traces of the system
- If property not verified:
  - A real counterexample that illustrates property violation
    - system error found
    - 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

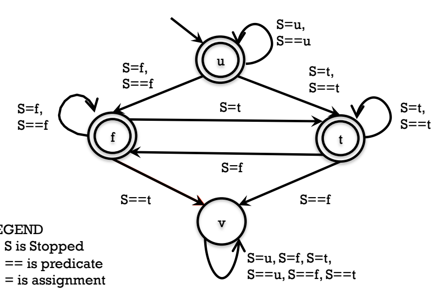
+ Incrementally Adding Precision to System Models

23

- 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

24

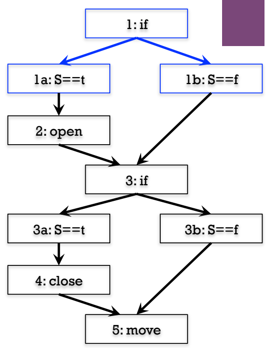


LEGEND
- S is Stopped
- == is predicate
- = is assignment

Example: More Precise TFG with Stopped events

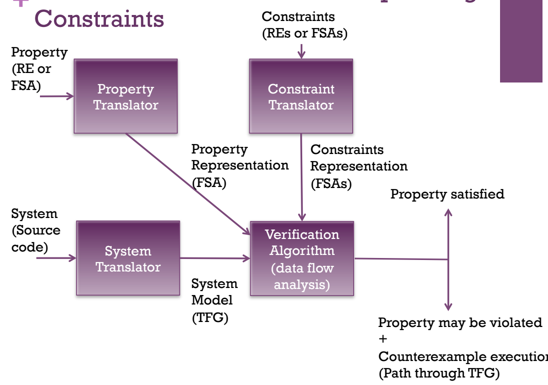
25

```
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
```



Architecture of FLAVERS Incorporating Constraints

26



Other Examples of Constraints

27

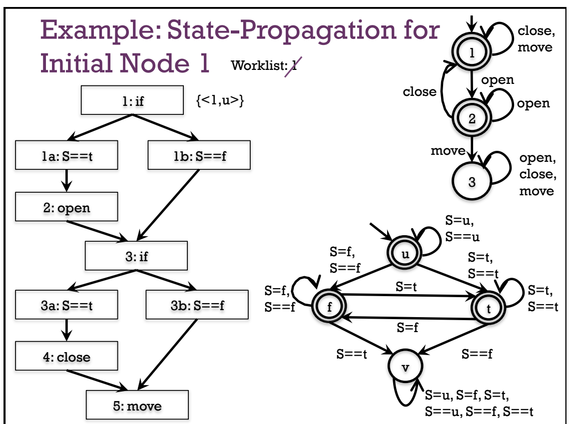
- Automatically generated
  - Variable constraint tracks value of given variable
    - Supported types of variables are boolean, enumerated, integer range
  - Task constraint tracks program counter of given task
- User-defined, e.g.,
  - Assumptions about environment

State Propagation Algorithm Incorporating Constraints

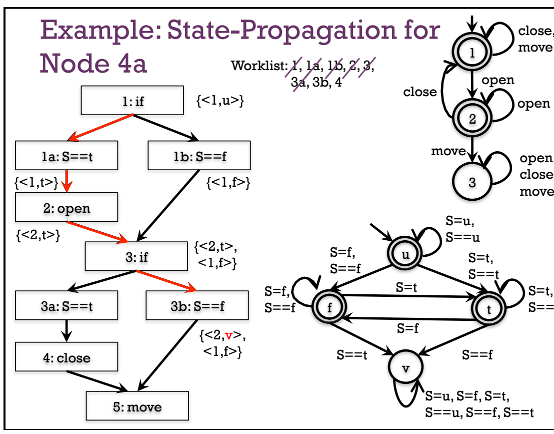
28

- 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**

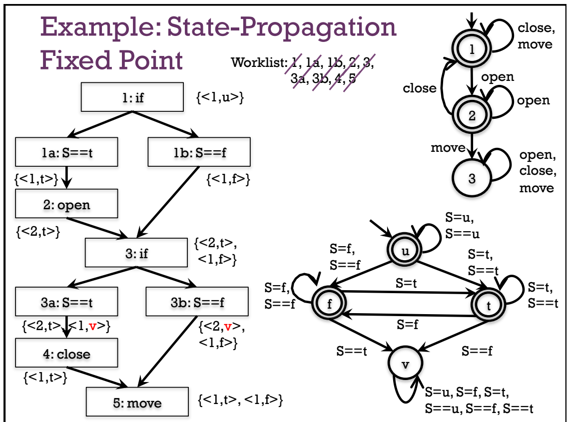
Example: State-Propagation for Initial Node 1



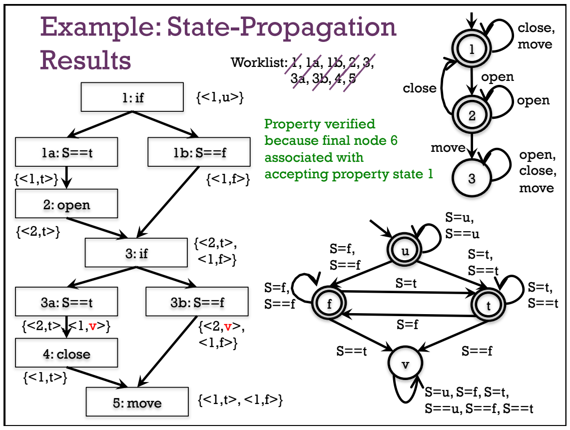
Example: State-Propagation for Node 4a



Example: State-Propagation Fixed Point



Example: State-Propagation Results



+ Discussion about FLAVERS

33

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

+ Evaluation of FLAVERS

34

- 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

35

- 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

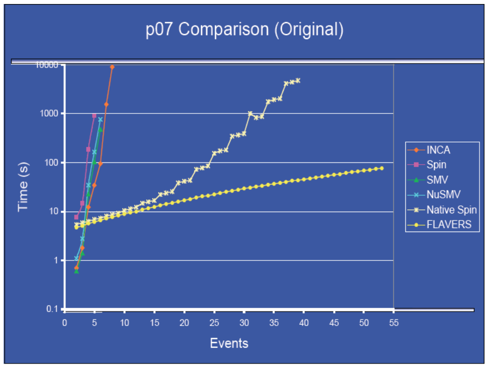
+ Drawbacks of FLAVERS

36

- 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

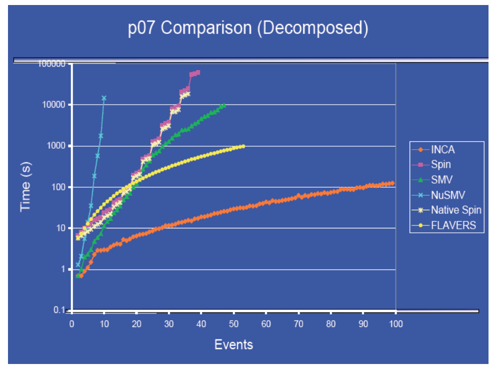
+ Comparative Evaluation of FSV Tools

37



+ Comparative Evaluation of FSV Tools

38



+ Some Research Directions for FSV

39

- 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

40

- Matthew B. Dwyer, Lori A. Clarke, Jamieson M. Cobleigh, and Gleb Naumovich. 2004. Flow analysis for verifying properties of concurrent software systems. *ACM Trans. Softw. Eng. Methodol.* 13, 4 (October 2004), 359-430. DOI=10.1145/1040291.1040292 http://doi.acm.org/10.1145/1040291.1040292

+ Demonstration of FLAVERS for Java…

41