Finite-State Verification or Model Checking

Finite State Verification (FSV) or Model Checking

- Holds the promise of providing a cost effective way of verifying important properties about a system
  - Not all faults are created equal
  - Invest effort into most important properties
- Several promising prototypes
  - Reachability Based
    - SPIN
    - Symbolic Model Checking (SMV)
    - LTSA
  - Flow Equations
    - Integer Necessary Conditions (INCA)
  - Data Flow Analysis
    - FLAVERS
High-Level Architecture of FSV Systems

Conservative Analysis

- If property verified, property holds for all possible executions of the system
- If property not verified:
  - An error found (in the system or in the property)
  OR
  - A spurious result
    - System model abstracts information to be tractable
    - Conservative abstractions usually over-approximate behavior
    - If inconsistency relies upon over-approximations, then a spurious result
      - e.g. every counterexample corresponds to an infeasible path
**System Model**

- Depends on property being verified
- Eliminate information that does not impact the proof
  - To keep the model as small as possible
- Abstraction techniques allow “states” in the model to be reduced/collapsed
  - Only keep track of the variables that are important to the property
    - Use slicing
    - Abstract values whenever possible
      - $x<0$, $x=0$, $x>0$

**Alphabet Refinement: Model reduction based on relevant events**

Can remove a node from the graph if it
- does not have an event associated with it, AND
- does not affect the flow of events through the graph
Some Properties of Properties

- State-based versus event-based
  - Once temperature is greater than 100 degrees, lock is true
  - Elevator door closes before elevator moves
- Single locations versus (sub)paths
  - Deadlock, race conditions, some mutual exclusion
  - Sequences of states or events
- Safety versus Liveness
  - Finitely refutable
  - Not finitely refutable
    - Look for cycles that violate the property or the absence of cycles that satisfy the property

A quick look at three approaches to FSV

- Reachability-based Model Checking
- Flow Equations
- Data Flow Analysis
Reachability-based Model Checking: some history

- Originally proposed for hardware
- Early 80’s: E. Clarke and Emerson; Quielle and Sifakis
- Late 80’s: Improved algorithms and property notations (E. Clarke, Emerson, Sistla)
- 90’s: Symbolic Model Checking (SMV) and other optimizations (Burch, E. Clarke, Dill, Long, and McMillan)
- Current: Hybrid approaches that combine model checking with
  - Theorem proving techniques
  - Symbolic execution
  - Optimization techniques (e.g., points to analysis)

Model Checking: Overview

- Properties usually expressed in a temporal logic
- System represented as a (possibly “abstracted”) reachability graph
  - State based=>show the values of all the “relevant” variables
- Reasoning engine propagates valid subformulas through the graph
High-Level Architecture of Model Checking

Representing Properties in CTL

- Temporal operators
  - $G$ - globally
  - $F$ - future
  - $X$ - next
  - $U$ - until
  - $R$ - release

- Path quantifiers
  - $A$ - for all paths
  - $E$ - for some path

- Examples:
  - $AG\ p$ means that for all paths from this state, $p$ is true and will remain true
  - $EF\ p$ means that for some path from this state, $p$ will eventually be true
Assigning propositions to nodes

- Mark nodes with propositions that are true at that node
- Each type of expression has a rule for how to propagate that expression through the graph
  - E.g., Mark a node with AXp (EXp) if for all (some) of its successors p is true

Propagating Propositions: AXp and EXp

X p - in the next state, p is true

Consider AX p

Consider EX p

Only need to look at the successors of the node of interest
**Propagating Propositions: AFp**

F p - at some time in the future, p is true

To propagate AF p:
- Mark nodes where p is true with AF p
- If all of a node’s successors are marked with AF p, mark that node
- Repeat Step 2 until a fixed point is reached

**Propagating Propositions: EFp**

F p - at some time in the future, p is true

To propagate EF p:
- Mark nodes where p is true with EF p
- If at least one of a node’s successors are marked with EF p, mark that node
- Repeat Step 2 until a fixed point is reached
Propagating Propositions
G p - globally in the future, p is true

• “Globally in the future” talks about paths of infinite length
• Need to identify strongly connected components (SCCs) in the graph
  • A subset of the graph in which every node is reachable from every other node in the subset
  • Can be computed in time linear to the size of the graph [Tarjan 1972]

To propagate AG p:
Identify SCCs
If p is true on every node in a SCC, mark every node in that SCC with AG p
If p is true on a node and all of its successors are marked with AG p, mark that node
Repeat Step 3 until a fixed point is reached
**Propagating Propositions**

\( G \ p \) - globally in the future, \( p \) is true

To propagate \( EG \ p \):
- Mark every node in a cycle where \( p \) is true on every node in that cycle with \( EG \ p \)
- If \( p \) is true on a node and at least one of its successors are marked with \( EG \ p \), mark that node
- Repeat Step 2 until a fixed point is reached

**Propagation rules**

- Different rule for each formula type
- A property is true for a graph if it holds in the initial node(s)
- Need a reachability graph that shows the states (i.e., the values) of the relevant variables
- Example:
  - process 1 can be null, trying to obtain the lock, or in its critical region \((n1, t1, c1)\)
  - process 2 can be null, trying to obtain the lock, or in its critical region \((n2, t2, c2)\)
  - turn is a variable that indicates which process can obtain the lock \((0, 1, 2)\)
**Example: mutual exclusion protocol [McMillan]**

(process1 = n1, t1, c1  
process2 = n2, t2, c2  
turn = 0, 1, 2)

Example Property

- $\text{AG}(t1 \rightarrow \text{AF} c1)$
  - If process1 gets the lock ($t1$) then eventually it gets into its critical region ($c1$)

- Subformulas
  - $\text{AF} c1$
  - $t1 \rightarrow \text{AF} c1$
  - $\text{AG}(t1 \rightarrow \text{AF} c1)$

- Note, would like to prove this for all processes but FSV approaches usually must instantiate a fixed configuration of the system (and property)
Example: propagation

$AG(t_1 \rightarrow AF \ c_1)$

Need to continue propagating

- $AG(t_1 \rightarrow AF \ c_1)$
- $(t_1 \rightarrow AF \ c_1)$
  - equivalent to ($\neg t_1 \lor AF \ c_1$)
Example: propagation
\[ AG(t_1 \rightarrow AF \, c_1) = AG(\neg t_1 \lor AF \, c_1) \]

\begin{center}
\includegraphics[width=\textwidth]{example_diagram1}
\end{center}

\textit{(process1, process2, turn)}

Example: propagation
\[ AG(t_1 \rightarrow AF \, c_1) \]

\begin{center}
\includegraphics[width=\textwidth]{example_diagram2}
\end{center}

\textit{(process1, process2, turn)}
Example: propagation

Property holds in the initial node, therefore it holds for the model

\[
\begin{align*}
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{n}_1, \text{n}_2, 0 \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{n}_1, \text{t}_2, 2 \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{n}_1, \text{c}_2, 2 \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{t}_1, \text{n}_2, 1 \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{t}_1, \text{t}_2, 2 \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{t}_1, \text{c}_2, 2 \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{c}_1, \text{n}_2, 1 \\
\end{align*}
\]

Formula Propagation

- Propagate until no change
  - propagate from smaller to larger subformulas
  - Actually keep all the formulas associated with a node

\[
\begin{align*}
\text{AF} c_1, \\
\text{t}_1 \to \text{AF} c_1, \\
\text{AG}(t_1 \to \text{AF} c_1) & \quad \text{n}_1, \text{t}_2, 2 \\
\end{align*}
\]

- “smart” algorithm: linear in the size of model and size of the formula
  - But, model is exponential
  - Many optimization techniques
    - Symbolic model checking
Some observations: Model Checking

- Worst case bound **linear** in size of the model
  - Model exponential
  - Symbolic model checking encodes Boolean expressions and usually reduces the size of the model
- Experimentally often very effective!
  - Used to verify hardware and designs
  - Trying to develop appropriate abstractions to make it applicable to software systems

A quick look at three approaches to FSV

- Model Checking
- Flow Equations
- Data Flow Analysis
Flow Equations: some history

- Originally proposed for designs
- Early 80's: Initial development (Avrunin, Dillon, and Wileden)
- 90's: Optimized and extended to real-time (Avrunin, Buy, Corbett, Dillon, and Wileden)
- Current: INCA prototype (Avrunin, Corbett, and Siegel)

Flow Equations

- Model system as a set of finite state automata
- Use extended network flow inequalities to capture legal flow through a concurrent system
- Represent complement of the property and see if it is consistent with the set of system inequalities
Flow Equations

- Determine if combined system of inequalities is consistent
  - Use integer linear programming
- If consistent, there is at least one set of flows through automata that violate the property
  - Provides trace through the model (but may not be executable)

High-Level Architecture of INCA

- Integer Necessary Condition Analyzer
- System Translator
- Linear Inequalities
- Property Specification
- Integer Linear Programming Solver
- Linear Inequalities
- Property Verified
- Assignment to Variables
Example: Task Flow Equations

\[
x_1 = x_0 + x_3 \\
x_2 = x_1 \\
x_2 = x_3 + x_4 \\
x_0 = 1 \\
x_4 = 1
\]

\[
x_6 = x_5 + x_7 \\
x_6 = x_7 + x_8 \\
x_5 = 1 \\
x_8 = 1
\]

*Flow in to a node = Flow out of a node
*Flow in and out of a task is 1

Example: Inter-task Flow Equations

\[
x_2 = x_6 \\
x_{11} = x_7 + x_8
\]

Rendezvous are always matched:
# calls = # accepts
Example: Require Non-Negative Flow

∀j: 0 ≤ xj

Flow over edges is non-negative

Example: Property

Are there more occurrences of event a then event b?

Property: For all paths, event a occurs more than event b

Property Equation: x2 > x11

Property Complement: x2 ≤ x11
Solving for a property

\[ \begin{align*}
x_1 &= 1 + x_3 \\
x_2 &= x_1 \\
x_2 &= x_3 + 1 \\
x_6 &= 1 + x_7 \\
x_6 &= x_7 + 1 \\
x_{10} &= 1 + x_{12} \\
x_{11} &= x_{10} \\
x_{11} &= x_{12} + 1 \\
x_2 &= x_6 \\
x_{11} &= x_7 + 1 \\
\forall j: 0 \leq x_j \\
x_2 &\leq x_{11}
\end{align*} \]

\}

Task Flow Equations

Inter-Task Flow Equations

Non-Negative Flow

Property Complement

Does this set of inequalities have a solution?

Solving for a property

\[ \begin{align*}
x_1 &= 1 + x_3 \\
x_2 &= x_1 \\
x_2 &= x_3 + 1 \\
x_6 &= 1 + x_7 \\
x_6 &= x_7 + 1 \\
x_{10} &= 1 + x_{12} \\
x_{11} &= x_{10} \\
x_{11} &= x_{12} + 1 \\
x_2 &= x_6 \\
x_{11} &= x_7 + x_8 \\
\forall j: 0 \leq x_j \\
x_2 &\leq x_{11}
\end{align*} \]

Solution exists

e.g., \( x_3, x_7, x_{12} = 0 \), all other \( x_i = 1 \)

\Rightarrow \text{property does not hold}
Example: Property

Are there more occurrences of event a then event b?
Property: For all paths, event a occurs more than event b
Counter Example: $x_3, x_7, x_{12}=0$, all other $x_i=1$

Some limitations

• Integer Linear Programming has an exponential worst case bound
• Interprocess order information is not preserved
  • only checks whether event counts are consistent
  • Like most static techniques, may produce spurious results
Benefits of the approach

• Does not enumerate the state space!
• ILP is often very efficient
  • Empirical evidence: linear inequality systems usually grow linearly and take sub-exponential times to solve
• In practice, INCA is often an effective technique

A quick look at three approaches to FSV

• Reachability-based Model Checking
• Flow Equations
• Data Flow Analysis
  • FLAVERS
High-Level Architecture of FSV Systems

Data Flow Based Verification: some history

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

- **FLow Analysis for VERification of Systems**
- Represents property as a finite-state automaton
- Reasoning engine based on data-flow analysis
- Relatively efficient because of the system model
  - collection of annotated control flow graphs
  - intertask communication and interleavings are represented with additional nodes & edges
  - does not enumerate all reachable states
- **over-approximates relevant executable behaviors**
  - Uses constraints to selectively improve precision of the model

TFG Construction

```plaintext
task body t1 is begin
    u;
    t2.synch;
    v;
    w;
end t1;
task t2 body is begin
    x;
    accept synch;
    y;
    z;
end t2;
```
Two Event-Based Models

TFG model

- **Conservative**
  - Represents all the sequences of events that occur in the program

- **Imprecise**
  - May include some sequences of events that can not occur in the program

- Resulting analysis is conservative, but may report spurious problems (false positives)
**FLAVERS**

- **Forward Flow, Any Path Problem**
- **IN and OUT are sets of FSA states**
- \( \text{IN}(n) = \bigcup_{m \in \text{pred}(n)} \text{OUT}(m) \)
- \( \text{OUT}(n) = \bigcup_{s \in \text{IN}(n)} \delta(s, n) \)
  - \( \delta \) is the FSA transition function
- **Result:** Let \( f \) be the final node of the TFG
  - All property: Want \( \text{OUT}(f) \subseteq \text{Accept}(P) \)
  - None property: Want \( \text{OUT}(f) \cap \text{Accept}(P) = \emptyset \)
  - \( \text{Accept}(P) \) is the set of accepting states of a property, \( P \)
- **Similar to Cesar, but state propagation being applied to a Trace Flow Graph**

---

**High-Level Architecture of FLAVERS**
State Propagation

Worklist: 2, 3

1: if
   {1}
   {1}

2: open
   {1}
   {2}

3: if
   {1,2}
   {1,2}

4: close
   {1,2}
   {1}

5: move
   {1,2}
   {1,3}

State Propagation

1: if
   {1}

2: open
   {2}

3: if
   {1,2}

4: close
   {1,2}

5: move
   {1,3}
**State Propagation**

1: if

2: open

3: if

4: close

5: move

...  
1: if (stopped) then  
2: open;  
end if;  
end if;  
...

3: if (stopped) then  
4: close;  
end if;  
end if;  
...

5: move;  
...

**Incrementally Improving Precision**

System Trace Flow Graph Property Translator Property FSA Constraints Property Specification Property Verified Counter Example Trace through TFG
Boolean Variable Constraint

== is a predicate
= is assignment

S==t
S=t
S==f
S=f

S==t
S=t
S==f
S=f

S==t
S=t
S==f
S=f

== is a predicate
= is assignment
Constraints

- Are represented as FSAs
- Describe conditions necessary for feasible execution
- Have a special *violation state* that is entered when an infeasible path is detected
  - Violation is a trap state; once it is entered, never leave that state

How do constraints affect the data flow equations

- IN and OUT are now sets of tuples of FSA states
- Merge is still union
- Transfer function now has to look at each FSA state in the in-tuple when computing the out-tuple
- 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
Elevator Revisited

1, 2, 4: if (stopped) then
3: open;
end if;

5, 6, 8: if (stopped) then
7: close;
end if;

9: move;

State Propagation

Worklist: t, u, t, u, v, x, y, 8

1: if
<1,u>
<1,t>

2: S==t
<1,t>
<2,t>

3: open
<1,t>
<2,t>

4: S==f
<1,u>
<1,f>

5: if
<2,t>,<1,f>
<2,t>,<1,f>

6: S==t
<2,t>,<1,f>

7: close
<2,t>,<1,f>
<2,t>,<1,v>

8: S==f
<2,t>,<1,f>

9: move
<2,t>,<1,v>
State Propagation

Worklist: 2, 3, 4, 5, 6, 8

1: if <1,u> <1,u>
   2: S==t <1,t>
   3: open <2,t>
   4: S==f <1,f>
   5: if <2,t>,<1,f> <2,t>,<1,f>
   6: S==t <2,t>,<1,v>
   7: close <1,u>
   8: S==f <1,t>
   9: move <2,t>,<1,f>

State Propagation

Worklist: 2, 3, 4, 5, 6, 8, 9

1: if <1,u> <1,u>
   2: S==t <1,t>
   3: open <2,t>
   4: S==f <2,f>
   5: if <2,t>,<1,f> <2,t>,<1,f>
   6: S==t <2,t>,<1,v>
   7: close <1,u>
   8: S==f <1,t>
   9: move <2,t>,<1,f>
State Propagation

Worklist: \( \mathbb{Z}, \mathbb{X}, \mathbb{Z}, \mathbb{X}, \mathbb{X}, \mathbb{X}, \mathbb{X}, \mathbb{X} \)

1: if

2: \( S = t \)

3: open

4: \( S = f \)

5: if

6: \( S = t \)

7: close

8: \( S = f \)

9: move

1: close

2: open

3: move

4: open

5: close

6: move

7: open

8: close

9: move

\(<1,u>, <2,t>, <1,v>\)
**Some Observations: Data Flow Analysis**

- Overall complexity is $O(N^2S)$
  - $N$ is the number of nodes in the model
  - $S$ is the number of states: property $\times$ constraints
- More precisely
  $$O(N^2 \cdot S_p \cdot S_{C_1} \cdot S_{C_2} \cdots S_{C_n})$$
  - In our experience, many important properties can be proven with a small number of constraints
  - Experimentally: performance sub-cubic
- Usually requires several iterations to determine needed constraints
- Constraints
  - Many automatically generated on request

**Improving Precision**

- Use constraints to improve precision
- Given a CFG $G$, a property $P$, and constraints $C_1, \ldots, C_n$
  - Alphabet refine $G$ wrt $(\Sigma_P \cup \Sigma_{C_1} \cup \ldots \cup \Sigma_{C_n})$
  - Want $(L(G) \cap L(C_1) \cap \ldots \cap L(C_n)) \subseteq L(P)$
FSV Summary

- A number of techniques for optimizing the model
  - Abstraction of values
  - Bounded analysis
  - Partial order
- Will always have some limitations on size
- The growth of distributed systems and the difficulty of testing such systems has increased attention on FSV
  - Microsoft, NASA, Motorola, Intel, Ford, ...
- Verify some of the properties of a system