Liveness Checking as Safety Checking
FMICS, July 12 – 13, Malaga, Spain

Armin Biere, Cyrille Artho, Viktor Schuppan

http://www.inf.ethz.ch/~schuppan/
Safety vs. Liveness

Safety
Something bad will not happen

Liveness
Something good will eventually happen
<table>
<thead>
<tr>
<th></th>
<th>Safety</th>
<th>Liveness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Characterization</td>
<td>partial correctness</td>
<td>termination</td>
</tr>
<tr>
<td>Operations</td>
<td>( post, \cup, \subseteq, \cap , bad )</td>
<td>( pre, \cap, \subseteq, \setminus , good )</td>
</tr>
<tr>
<td>Tool support</td>
<td>almost all</td>
<td>less common</td>
</tr>
</tbody>
</table>
If the number of states is finite

1. a system with a liveness property can be transformed to a system with an equivalent safety property

2. the transformed system can be model-checked efficiently
Outline

Introduction

Counter-Based Translation

State-Recording Translation

Experimental Results
Lasso-Shaped Counterexamples

Lasso-shaped counterexample for $\text{AF}_p$

Always exists in a finite state system
Is $AF_k$ true?
Given Finite state system with $n$ states, liveness property $AF k$

Find Initialized path where $k$ is false for the first $n + 1$ states
Counter-Based Translation

Requires \( n \) forward iterations
\( \Rightarrow \) impractical for realistic systems
Find Initialized path where $k$ is false until a state is visited for the second time

But... State space search is memory-less

Guess start of loop, save guess in copy of state variables
State-Recording Translation

- **Saved state** (found)
- **State**
- **Save state**
- **Oracle**

The diagram illustrates the state recording process, with a focus on the saved state, state, save state, and oracle. The process includes saving states, identifying loop detection, and checking if the state is live.

- **Save state**
- **State**
- **Live state**
- **Detected loop**
- **Is live true?**

The diagram shows a sequence of states with arrows indicating the flow of the process. The saved state is highlighted, and the process of identifying a loop and checking if the state is live is depicted.
VAR

    state: -k..k;

DEFINE

    found := state = k;

ASSIGN

    init(state) := 0;
    next(state) :=
        case
            state = 0: {-1,1};
            state < 0 &
                state > -k: state-1;
            state > 0 &
                state < k: state+1;
            state = -k |
                state = k: 0;
        esac;

SPEC

    AF found
State-Recording Translation

VAR
  state: -k..k;
  _state: -k..k;
  save, saved: boolean;

DEFINE
  found := state = k;

ASSIGN
  init(state) := 0;
  next(state) :=
    case
      state = 0: {-1,1};
      state < 0 &
        state > -k: state-1;
      state > 0 &
        state < k: state+1;
      state = -k |
        state = k: 0;
    esac;

  init(_state) := case
    !saved & save: state;
    1: _state;
  esac;

  next(_state) := case
    !saved & save: state;
    1: _state;
  esac;

  -- save is an oracle
  init(saved) := 0;
  next(saved) := saved | save;

SPEC
  AF found
VAR
  state: -k..k;
  _state: -k..k;
  live, save, saved: boolean;

DEFINE
  found := state = k;

ASSIGN
  init(state) := 0;
  next(state) :=
    case
      state = 0: {-1,1};
      state < 0 &
        state > -k: state-1;
      state > 0 &
        state < k: state+1;
      state = -k |
        state = k: 0;
    esac;
  next(_state) := case
    !saved & save: state;
    1: _state;
  esac;
  init(live) := 0;
  next(live) := live | found;
  -- save is an oracle
  init(saved) := 0;
  next(saved) := saved | save;

SPEC
  AF found
VAR
  state: -k..k;
  _state: -k..k;
  live, save, saved: boolean;

DEFINE
  found := state = k;
  loop := saved & state = _state;

ASSIGN
  init(state) := 0;
  next(state) :=
    case
      state = 0: {-1,1};
      state < 0 &
        state > -k: state-1;
      state > 0 &
        state < k: state+1;
      state = -k |
        state = k: 0;
    esac;
  next(_state) := case
    !saved & save: state;
    1: _state;
  esac;
  init(live) := 0;
  next(live) := live | found;
  -- save is an oracle
  init(saved) := 0;
  next(saved) := saved | save;

SPEC
  AF found
  AG (loop -> live)
## Complexity

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameter</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>no. of states</td>
<td>$</td>
</tr>
<tr>
<td>On-the-fly</td>
<td>no. of reachable states</td>
<td>$</td>
</tr>
<tr>
<td>Symbolic</td>
<td>BDD size</td>
<td>linear in the product of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– size of original BDDs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– no. of state bits</td>
</tr>
<tr>
<td></td>
<td></td>
<td>– size of BDD for states in which $p$ holds</td>
</tr>
<tr>
<td></td>
<td>diameter</td>
<td>$d_p \leq 4d + 3$</td>
</tr>
<tr>
<td></td>
<td>radius</td>
<td>$r_p \leq r + 3d + 3$</td>
</tr>
</tbody>
</table>
...is straightforward:

– add one state bit per fairness constraint $f_i$

– remember if $f_i$ was true on the loop, define $fair := \bigwedge_i f_i$

– replace $\mathsf{AG} \; loop \Rightarrow live$ with $\mathsf{AG} \; (loop \land fair) \Rightarrow live$
Arbitrary LTL formulae $f$ can be verified

- using a tableau construction for $f$ and

- checking $\neg (s \neg f \land \text{EG True})$ under fairness constraints

Special translation rules can be derived

- e.g. $A p_1 U p_2 \equiv A (p_1 W p_2 \land F p_2)$
Experimental Results - Skipping Counter

Is $\text{AF}_k$ true?
State-recording translation requires fewer iterations
IEEE 1394 (FireWire)

– serial high speed bus

– $n$ nodes with $p$ ports each form a tree

Tree Identify Protocol

– elect node as unique leader during initialization

– liveness property: $\textbf{AF} \ (node[0].root \ | \ ... \ | \ node[n-1].root)$

– contention may arise $\Rightarrow$ resolve with two \textit{fair} coin throws

– modeled and verified with Cadence SMV
## Experimental Results - IEEE 1394 FireWire

### Verification of safe model is possible

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p$</th>
<th>$t_{live}$</th>
<th>$t_{safe}$</th>
<th>$\frac{t_{safe}}{t_{live}}$</th>
<th>$\frac{mem_{safe}}{mem_{live}}$</th>
<th>$mem_{safe}$</th>
<th>$mem_{live}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>0.9</td>
<td>4.2</td>
<td>4.20/0.9 = 4.67</td>
<td>1.90/0.9 = 2.11</td>
<td>0.07</td>
<td>0.40</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1.9</td>
<td>11.1</td>
<td>11.10/1.9 = 5.84</td>
<td>2.70/1.9 = 1.42</td>
<td>0.20</td>
<td>0.78</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4.7</td>
<td>28.2</td>
<td>28.20/4.7 = 6.00</td>
<td>5.50/4.7 = 1.16</td>
<td>0.44</td>
<td>1.30</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>11.3</td>
<td>39.5</td>
<td>39.50/11.3 = 3.49</td>
<td>7.60/11.3 = 0.67</td>
<td>0.70</td>
<td>1.95</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>76.1</td>
<td>283.1</td>
<td>283.10/76.1 = 3.71</td>
<td>53.60/76.1 = 0.71</td>
<td>3.78</td>
<td>9.58</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>450.7</td>
<td>1567.7</td>
<td>1567.70/450.7 = 3.48</td>
<td>259.50/1567.7 = 0.16</td>
<td>29.22</td>
<td>31.76</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>357.3</td>
<td>1376.2</td>
<td>1376.20/357.3 = 3.85</td>
<td>204.80/357.3 = 0.57</td>
<td>14.00</td>
<td>35.55</td>
</tr>
</tbody>
</table>

1.59 < $\frac{t_{safe}}{t_{live}}$ < 6

0.73 < $\frac{mem_{safe}}{mem_{live}}$ < 6
Conclusion

Contribution

- Transform liveness properties to equivalent safety properties

Benefits

- Use commercial/proprietary tools for safety to verify liveness
- Lift some theoretical results for safety to liveness
- Find counterexample traces of minimal length

Future work

- Reduce number of state bits needed
- Apply method to ATPG or STE
Keep out!
Backup slides
Safety – Symbolic Computation

\[ \mu X. (X \cup \text{post}(X))(\text{init}) \]

\[ \text{init} \cup \text{post}(\text{init}) \cup \text{post}^2(\text{init}) \]

\[ \text{post}, \cup, \subseteq, \cap \text{ bad} \]
\[ \forall X. (X \cap \text{pre}(X)) \not\text{not good} \]

\[ \text{pre}^2(\not\text{good}) \cap \text{pre}(\not\text{good}) \cap \not\text{good} \]

\[ \text{pre}(\not\text{good}) \cap \not\text{good} \]

\[ \not\text{good} \]

\[ \text{pre}, \cap, \subseteq, \setminus \text{good} \]
VAR
  state: -k..k;

DEFINE
  found := state = k;

ASSIGN
  init(state) := 0;
  next(state) :=
    case
      state = 0: {-1,1};
      state < 0 &
        state > -k: state-1;
      state > 0 &
        state < k: state+1;
      state = -k |
        state = k: 0;
    esac;

SPEC
  AF found
VAR
  state: -k..k;
  count: 0..2*k+1;

DEFINE
  found := state = k;

ASSIGN
  init(state) := 0;
  next(state) :=
    case
      state = 0: {-1,1};
      state < 0 &
        state > -k: state-1;
      state > 0 &
        state < k: state+1;
      state = -k |
        state = k: 0;
    esac;

  init(count) := 0;
  next(count) := case
    count < 2*k+1: count+1;
    count = 2*k+1: count;
  esac;

SPEC
  AF found
VAR
  state: -k..k;
  count: 0..2*k+1;
  live: boolean;

DEFINE
  found := state = k;

ASSIGN
  init(state) := 0;
  next(state) :=
  case
    state = 0: {-1,1};
    state < 0 &
      state > -k: state-1;
    state > 0 &
      state < k: state+1;
    state = -k |
      state = k: 0;
  esac;

  init(count) := 0;
  next(count) := case
    count < 2*k+1: count+1;
    count = 2*k+1: count;
  esac;

  init(live) := 0;
  next(live) := live | found;

SPEC
  AF found
VAR
  state: -k..k;
  count: 0..2*k+1;
  live: boolean;

DEFINE
  found := state = k;
  stop := count = 2*k+1;

ASSIGN
  init(state) := 0;
  next(state) :=
    case
      state = 0: {-1,1};
      state < 0 &
        state > -k: state-1;
      state > 0 &
        state < k: state+1;
      state = -k |
        state = k: 0;
    esac;
  init(count) := 0;
  next(count) := case
    count < 2*k+1: count+1;
    count = 2*k+1: count;
  esac;
  init(live) := 0;
  next(live) := live | found;

SPEC
  AF found
  AG (stop -> live)
## Experimental Results - IEEE 1394 FireWire

| $n$ | $p$ | check true | | check false |
|-----|-----|------------|------------|
|     |     | live | safe | live | safe |
| 2   | 2   | 19   | 55   | 24   | 0    |
| 2   | 3   | 19   | 55   | 24   | 0    |
| 2   | 4   | 19   | 59   | 24   | 0    |
| 3   | 2   | 21   | 55   | 23   | 0    |
| 3   | 3   | 21   | 56   | 23   | 0    |
| 3   | 4   | 21   | 56   | 23   | 0    |
| 4   | 2   | 31   | 98   | 36   | 0    |