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

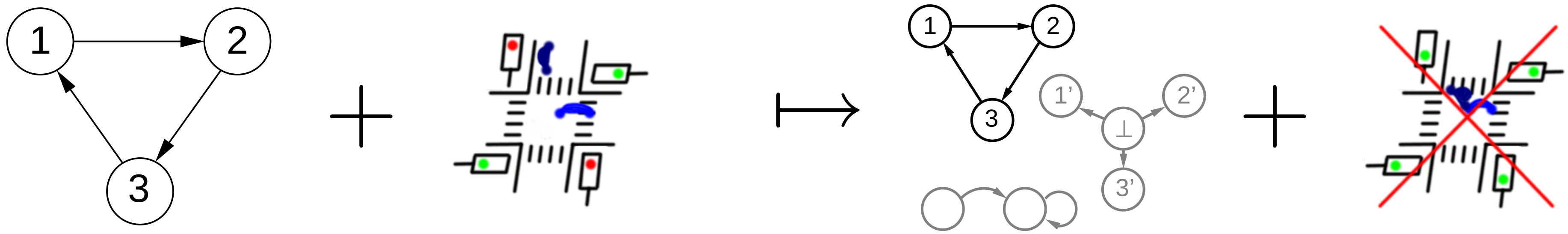
Liveness

Something good will eventually happen

Safety vs. Liveness

	Safety	Liveness
Characterization	partial correctness	termination
Operations	post, $\cup$, $\subseteq$, $\cap$ bad	pre, $\cap$, $\subseteq$, $\setminus$ good
Tool support	almost all	less common

Safety vs. Liveness: Finite State Systems



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

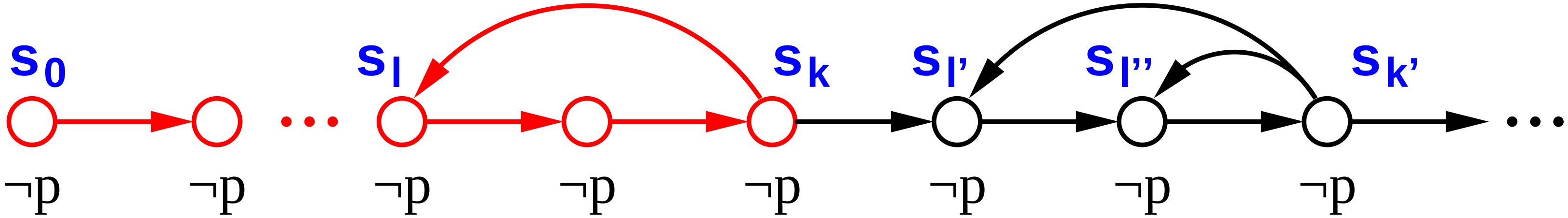
Introduction

Counter-Based Translation

State-Recording Translation

Experimental Results

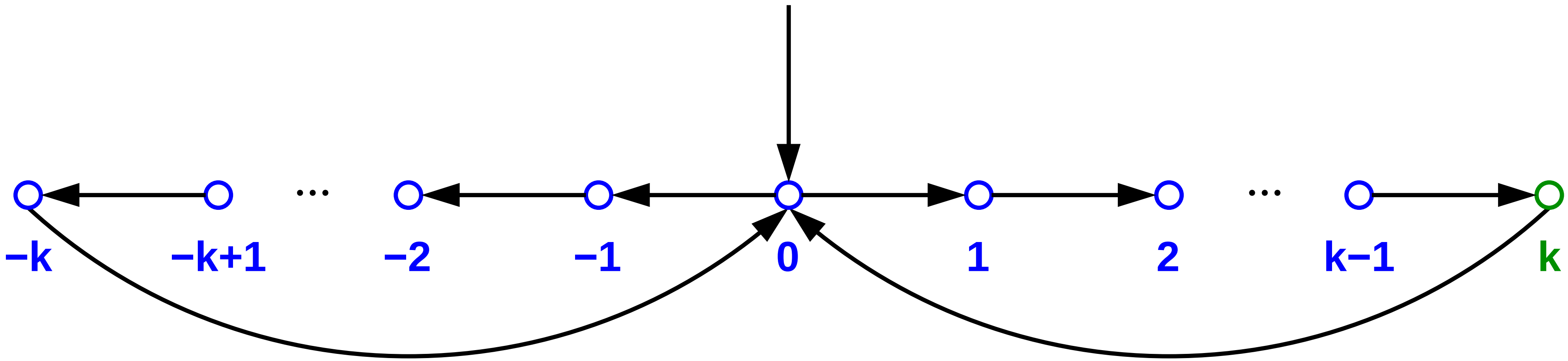
Lasso-Shaped Counterexamples



Lasso-shaped counterexample for $\mathbf{AF}\ p$

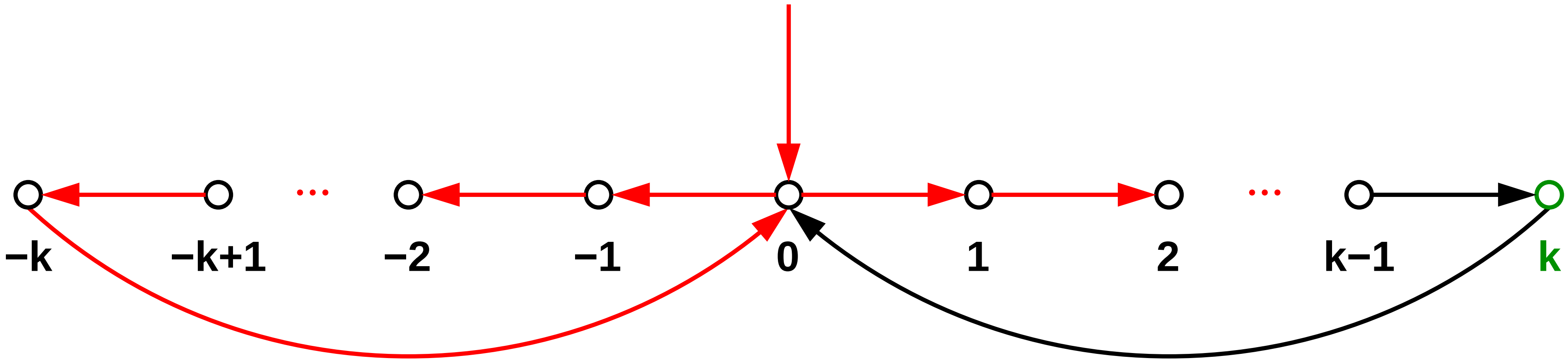
Always exists in a finite state system

Example



Is $\mathbf{AF}\ k$ true?

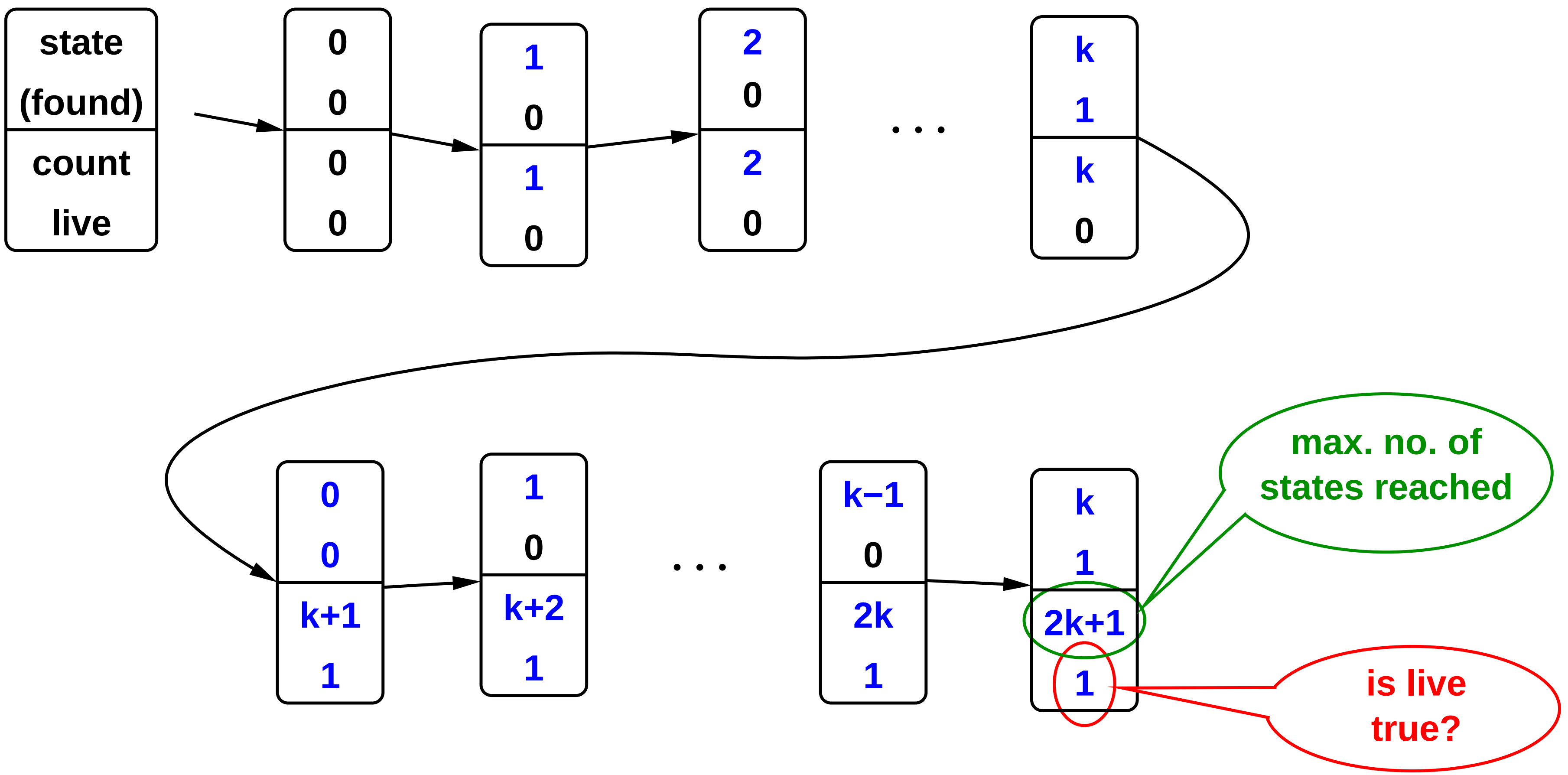
Counter-Based Translation



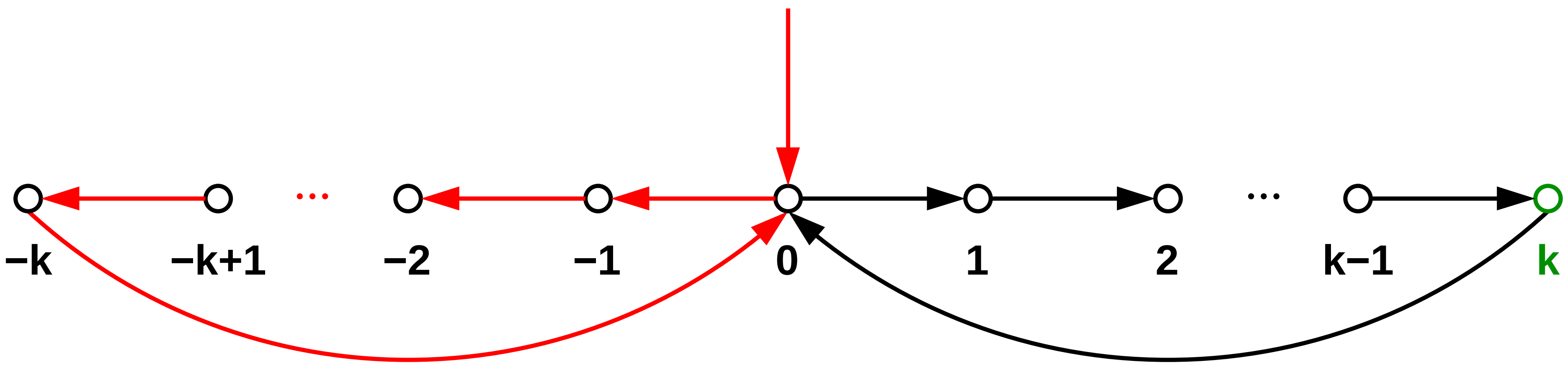
Given Finite state system with n states, liveness property $\mathbf{AF}\ k$

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

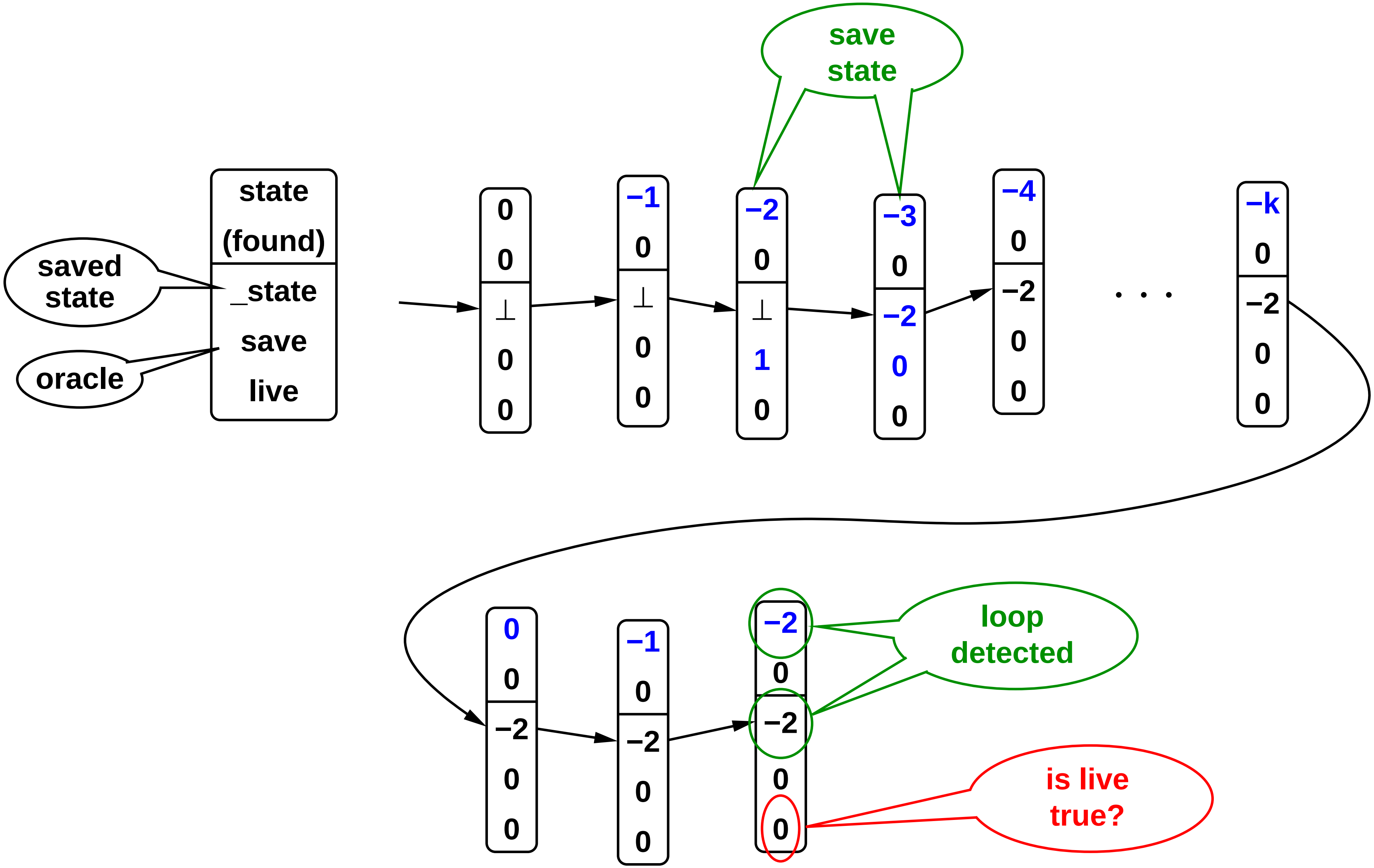


Find Initialized path where k is false until a state is visited for the second time

But. . . State space search is memory-less

Guess start of loop, save guess in copy of state variables

State-Recording Translation



State-Recording Translation


```
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;
```

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

-- save is an oracle

init(saved) := 0;
next(saved) := saved | save;
```

```
SPEC
  AF found
```

State-Recording Translation

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

```
DEFINE
  found := state = k;
```

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

```
SPEC
  AF found
```

State-Recording Translation


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

```
SPEC
  AF found                          AG (loop -> live)
```

Complexity

Algorithm	Parameter	Size
Explicit	no. of states	$\lvert S_p\rvert = 2\lvert S\rvert(\lvert S\rvert+1) = O(\lvert S\rvert^2)$
On-the-fly	no. of reachable states	$\lvert R_p\rvert \leq 2\lvert R\rvert(\lvert R\rvert+1) = O(\lvert R\rvert^2)$
Symbolic	BDD size	linear in the product of – size of original BDDs – no. of state bits – size of BDD for states in which p holds
	diameter	$d_p \leq 4d+3$
	radius	$r_p \leq r+3d+3$

Adding Fairness

. . . 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 $\mathbf{AG}\, loop \Rightarrow live$ with $\mathbf{AG}\,(loop \wedge fair) \Rightarrow live$

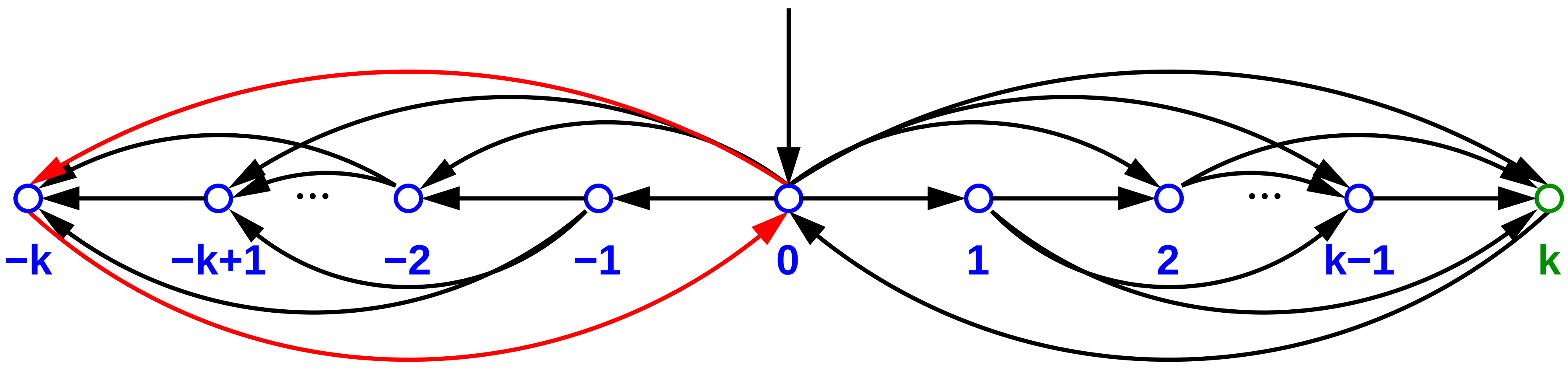
Arbitrary LTL formulae f can be verified

- using a tableau construction for f and
- checking $\neg\,(s_{\neg f}\ \wedge\ \mathbf{EG}\,True)$ under fairness constraints

Special translation rules can be derived

- e. g. $\mathbf{A}\,p_1\,\mathbf{U}\,p_2 \ \equiv\ \mathbf{A}\,(p_1\,\mathbf{W}\,p_2\ \wedge\ \mathbf{F}\,p_2)$

Experimental Results - Skipping Counter



Is $\mathbf{AF}\,k$ true?

Experimental Results - Skipping Counter

	check true						check false					
k	live		count		safe		live		count		safe	
4	2	5	10	0	5	0	2	4	9	0	2	0
8	2	9	18	0	5	0	2	8	17	0	2	0
12	2	13	26	0	5	0	2	12	25	0	2	0
16	2	17	34	0	5	0	2	16	33	0	2	0

State-recording translation requires fewer iterations

Experimental Results - IEEE 1394 FireWire

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: $\mathbf{AF}\ (node[0].root \mid \ldots \mid node[n-1].root)$
- contention may arise $\Rightarrow$ resolve with two fair coin throws
- modeled and verified with Cadence SMV

Experimental Results - IEEE 1394 FireWire

		check true				check false			
		live		safe		live		safe	
n	p	sec	MNod	sec	MNod	sec	MNod	sec	MNod
2	2	0.9	0.07	4.2	0.40	1.1	0.10	2.6	0.28
2	3	1.9	0.20	11.1	0.78	2.7	0.22	6.8	0.60
2	4	4.7	0.44	28.2	1.30	5.5	0.40	16.0	0.94
3	2	11.3	0.70	39.5	1.95	7.6	0.72	12.1	0.77
3	3	76.1	3.78	283.1	9.58	53.6	3.68	86.8	4.22
3	4	450.7	29.22	1567.7	31.76	259.5	19.59	554.4	14.36
4	2	357.3	14.00	1376.2	35.55	204.8	12.50	644.2	24.86

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

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

Verification of safe model is possible

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

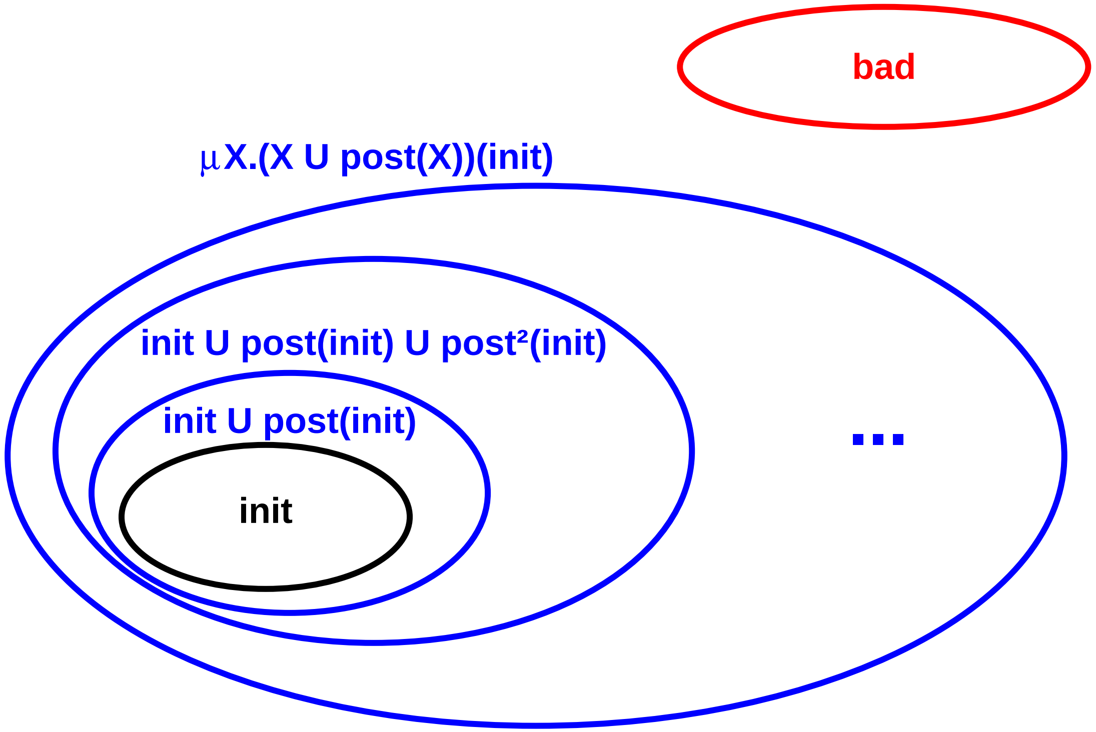
- Reduce number of state bits needed
- Apply method to ATPG or STE

Backup-Slides

Keep out!

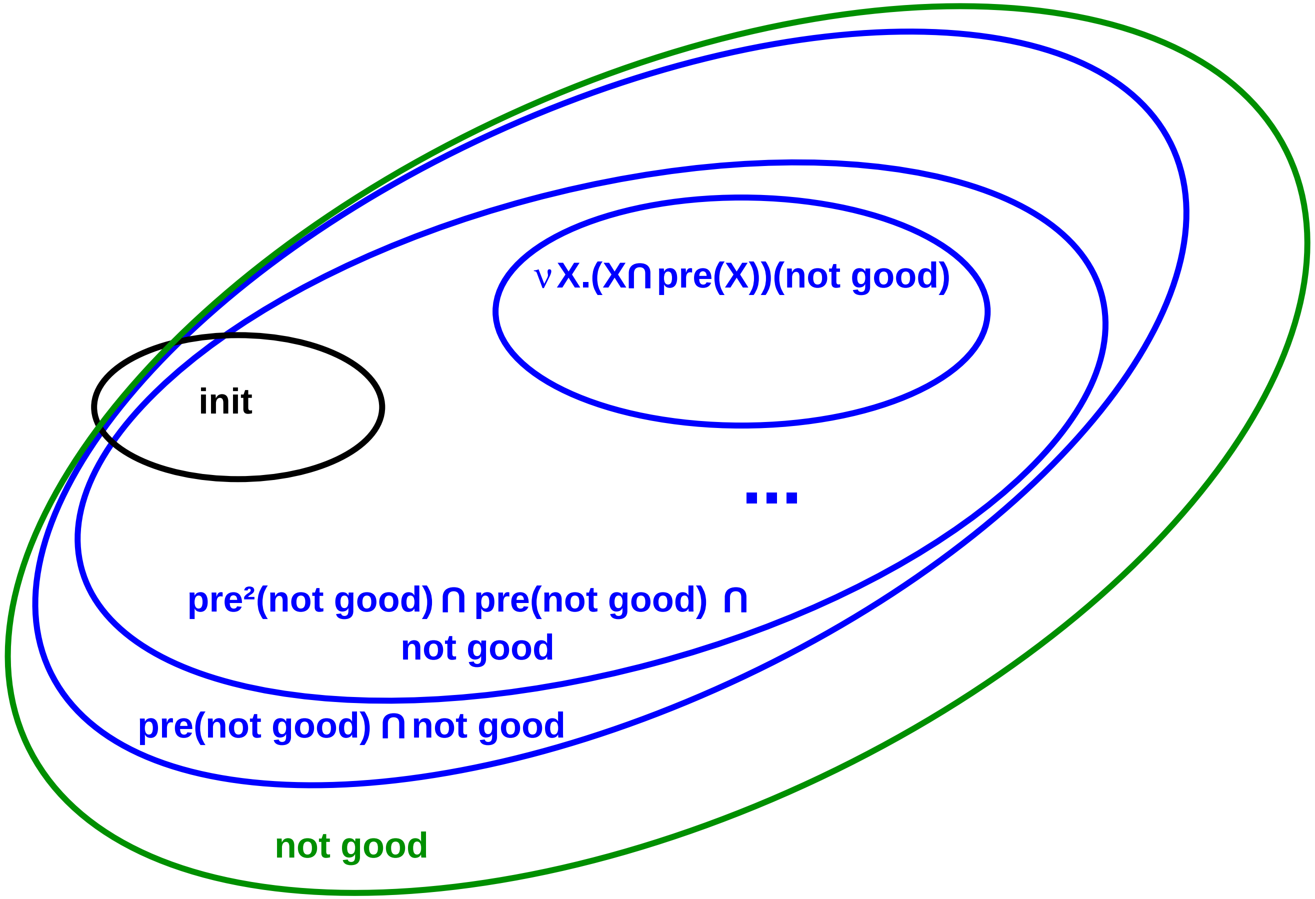
Backup slides

Safety – Symbolic Computation



$$post, \cup, \subseteq, \cap\ bad$$

Liveness – Symbolic Computation



$$pre, \cap, \subseteq, \setminus\ good$$

Counter-Based Translation

```
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
```

Counter-Based Translation


```
VAR
  state: -k..k;                    count: 0..2*k+1;
```

```
DEFINE
  found := state = k;
```

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

```
SPEC
  AF found
```

Counter-Based Translation


```
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
```

Counter-Based Translation


```
VAR
  state: -k..k;                    count: 0..2*k+1;
                                   live: boolean;
```

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

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

SPEC

~~AF found~~

`AG (stop -> live)`

Experimental Results - IEEE 1394 FireWire

		check true				check false			
n	p	live		safe		live		safe	
2	2	19	55	24	0	19	15	13	0
2	3	19	55	24	0	19	16	13	0
2	4	19	59	24	0	19	17	13	0
3	2	21	55	23	0	21	15	11	0
3	3	21	56	23	0	21	16	11	0
3	4	21	56	23	0	21	16	11	0
4	2	31	98	36	0	31	21	19	0