Automatic Fault Localization for Property Checking

Stefan Staber, Roderick Bloem

Graz University of Technology

Görschwin Fey, Rolf Drechsler

University of Bremen

Motivation

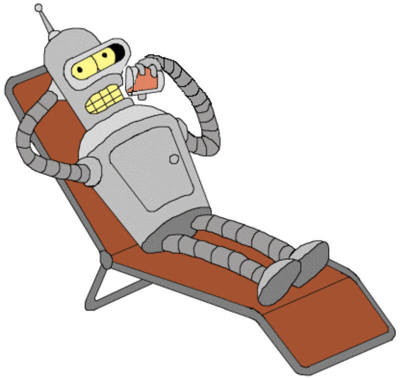
Verification

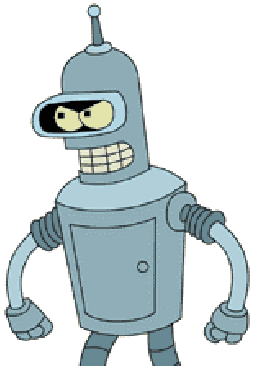
Specification holds

Specification fails

Fault localization

Fault correction





Related Work

Understandability of counterexamples
- Clarke et al 95: seminal work
- Ravi, Somenzi, Jin: decision points, “width” of counterexample

Comparing good and bad traces to find suspicious statements
- Groce, Zeller, Ball and Rajamani

Localization and Correction for
- Combinational circuits [various] or
- Sequential circuits [Wahba&Borrione, Ali et al.]
- Correction for sequential circuits with general fault models (computationally hard) [Jobstmann, Griesmayer, Staber, Bloem]

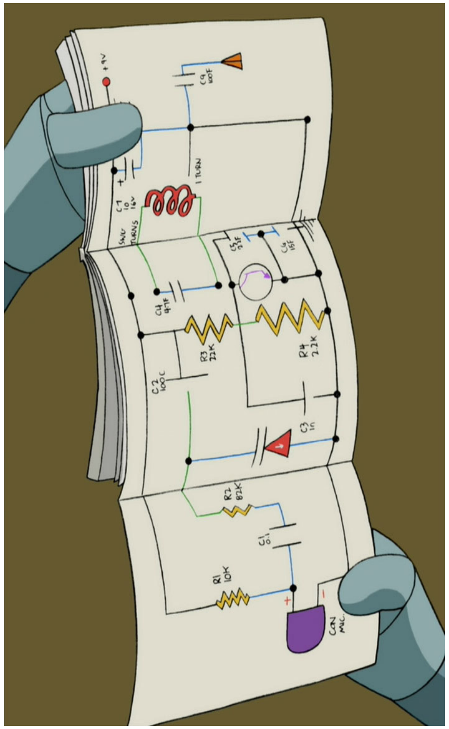
Contents

- Basic idea of localization
- Approach
- Problems with precision with a solution
- Efficiency & specificity
- Fault location on HDL level
- Experimental results

Localization

A counterexample shows a contradiction between actual behavior and specification

The question: *Can we find a* ***component*** *responsible for the contradiction?*



But: what is a component?

- Ideas presented here work for any component model!
- Gates or expressions are typical choices

Localization

Identify components responsible for a failure

Input

- Faulty design
- Set of *finite* failure traces
    - Liveness aspects are ignored
- Correct specification given in Linear Time Logic (LTL)

Output

- Set of fault candidates for the given traces

Simplifying assumptions for this presentation

- One faulty component
- One failure trace

Localization with Model Checking

Given a failure trace

1. Modify circuit
   - In first step it decides non-deterministically which component is faulty.
   - From then on, the faulty component has nondeterministic output
2. Fix inputs to failure trace
3. Model check: is there a trace that *fulfills* the formula?
   - I.e., is there a component and a behavior for the component so that the contradiction is resolved?

Localization with BMC

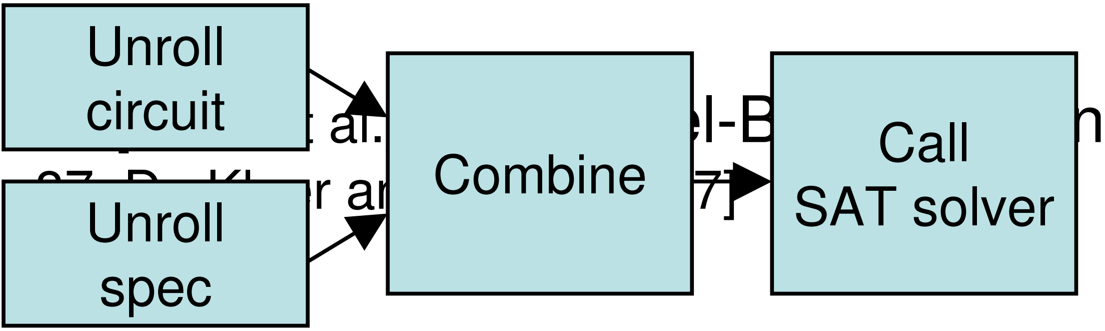
Given a failure trace

Approach

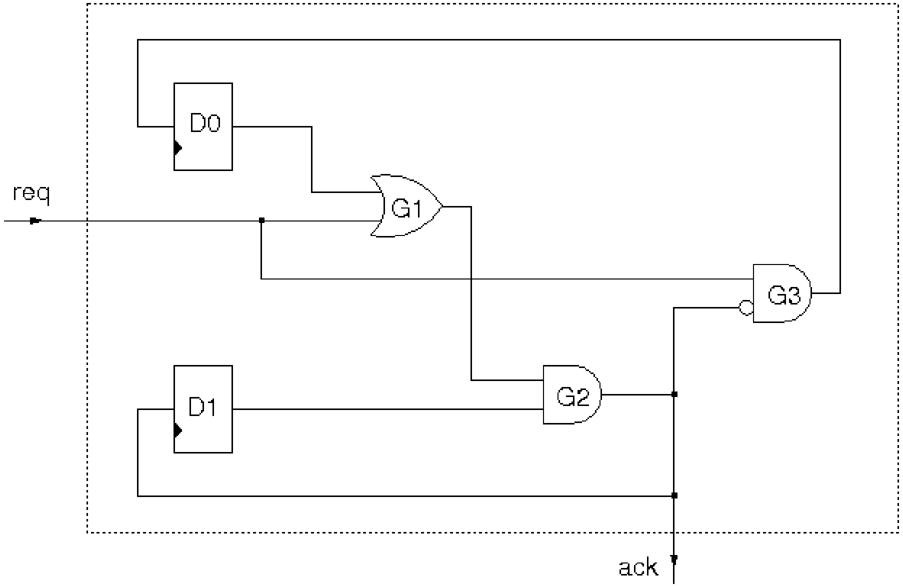
1. Unroll the circuit; introduce “abnormal predicates,” fix inputs to failure trace
2. Unroll LTL property using expansion rules
   (e.g. G a = a ∧ XG a)
3. Combine circuit & property
4. Call SAT-Solver and find valid assignment for the variables (notably abnormal predicates)

Mix of B t al. el-B hosis

[Reite r ar 7]



Localization: Arbiter

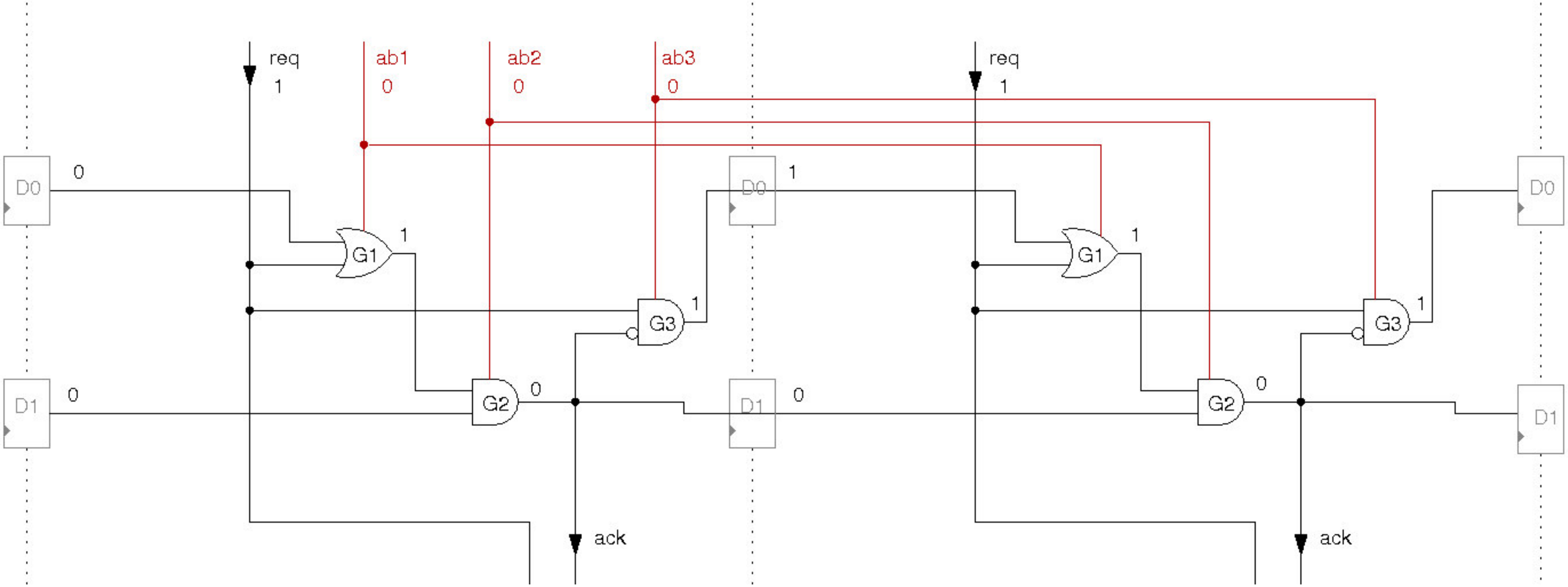


initial state
D0=0, D1=0

Property
$G(req \rightarrow (ack \vee X\,ack) \wedge (ack \rightarrow \neg X\,ack))$

fails for two consecutive requests (failure trace: req = 1; req = 1)
(We get no acks; G2 should be G1 $\wedge$ $\neg$D1)

1: Unroll, Introduce Predicates



Components:

G1: not **AB1** → (outG1**t0** = in1G1**t0** + in2G1**t0**),

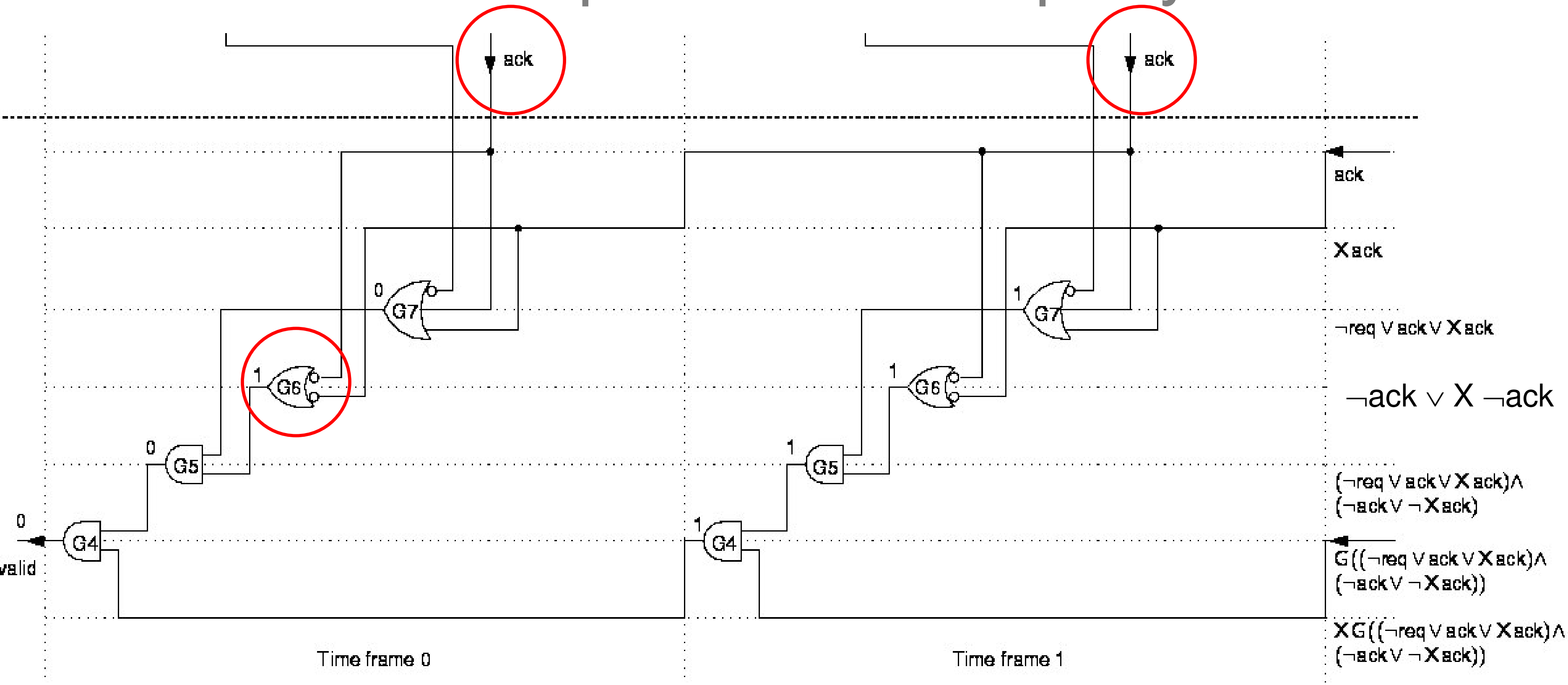
not **AB1** → (outG1**t1** = in1G1**t1** + in2G1**t1**)

G2: not **AB2** → (outG2**t0** = in1G2**t0** * in2G2**t0**),

not **AB2** → (outG2**t1** = in1G2**t1** * in2G2**t1**)

in1G1**t0** = 1 (failure trace t_0: req = 1), etc.

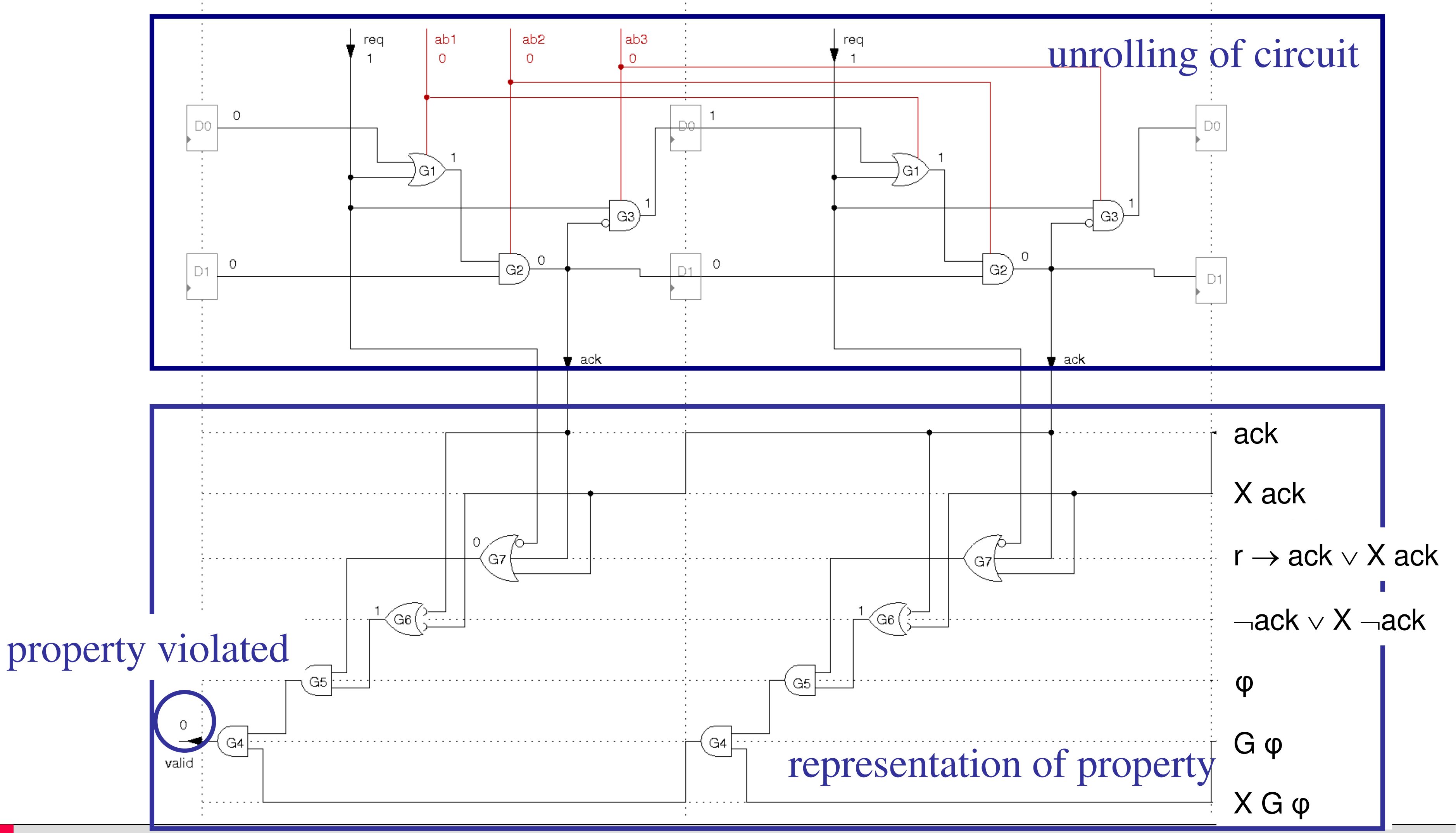
Step 2: Unroll Property



$$G((req \rightarrow (ack \vee X\ ack)) \wedge (ack \rightarrow \neg X\ ack))$$

Note: Free inputs on the right are left free: represent liveness part

Step 3: Combine



unrolling of circuit

property violated

representation of property

ack

X ack

r → ack ∨ X ack

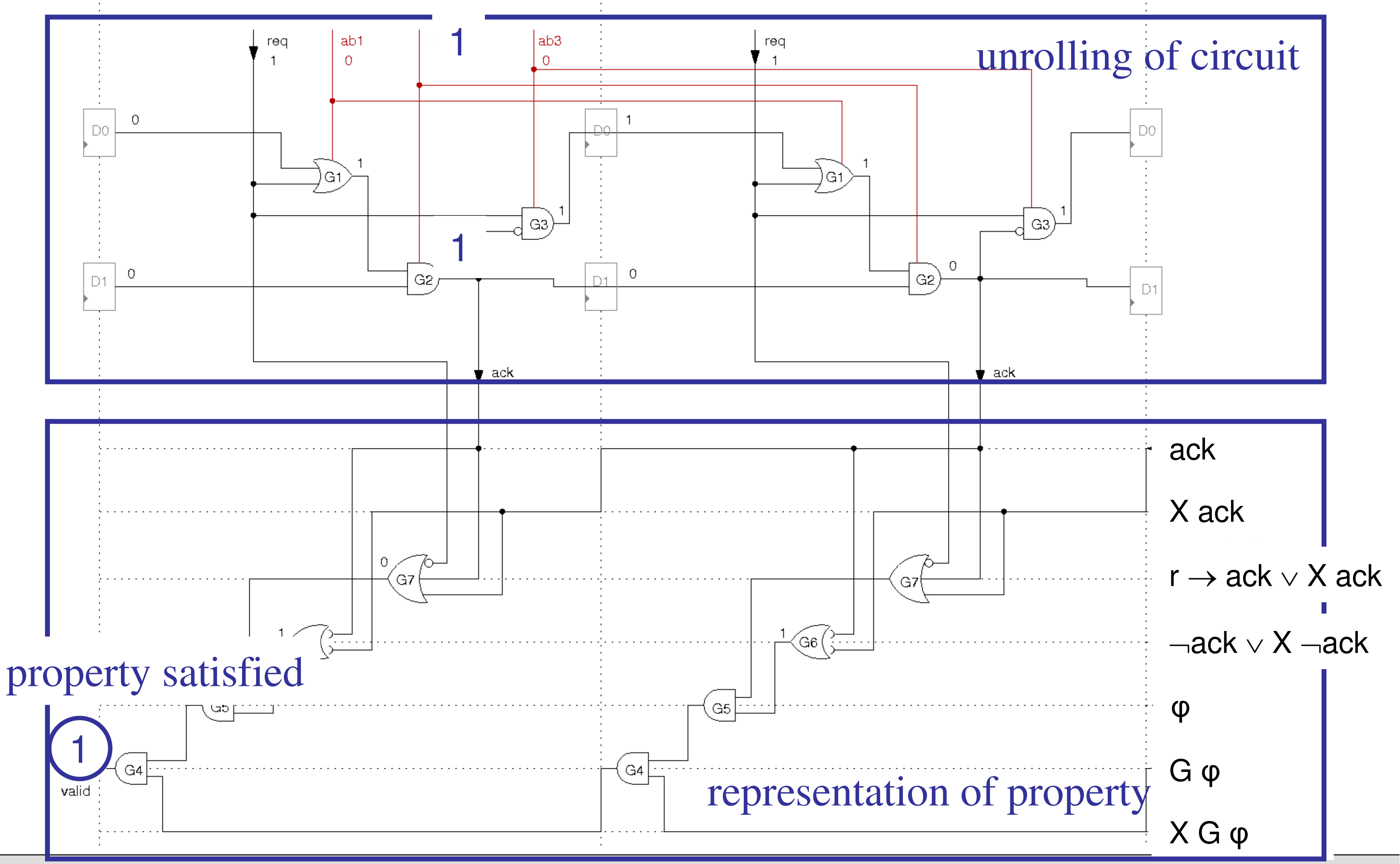
¬ack ∨ X ¬ack

φ

G φ

X G φ

Step 4: SAT Solver



unrolling of circuit

property satisfied

representation of property

ack

X ack

$r \rightarrow ack \vee X\ ack$

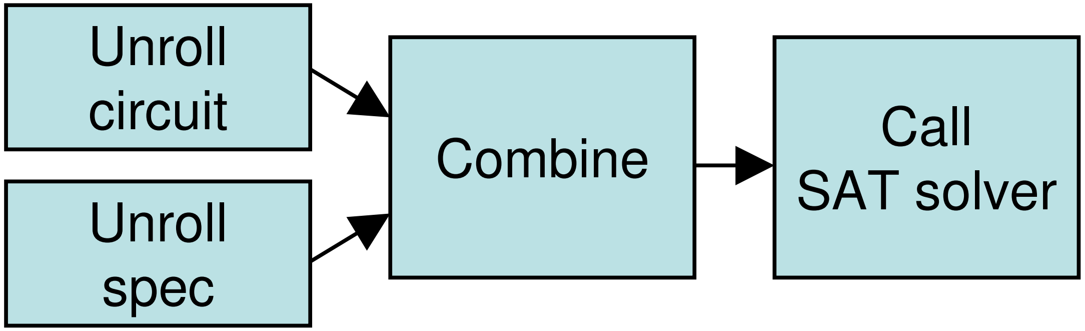
$\neg ack \vee X\ \neg ack$

φ

$G\ \varphi$

$X\ G\ \varphi$

Localization



Correctness & Completeness

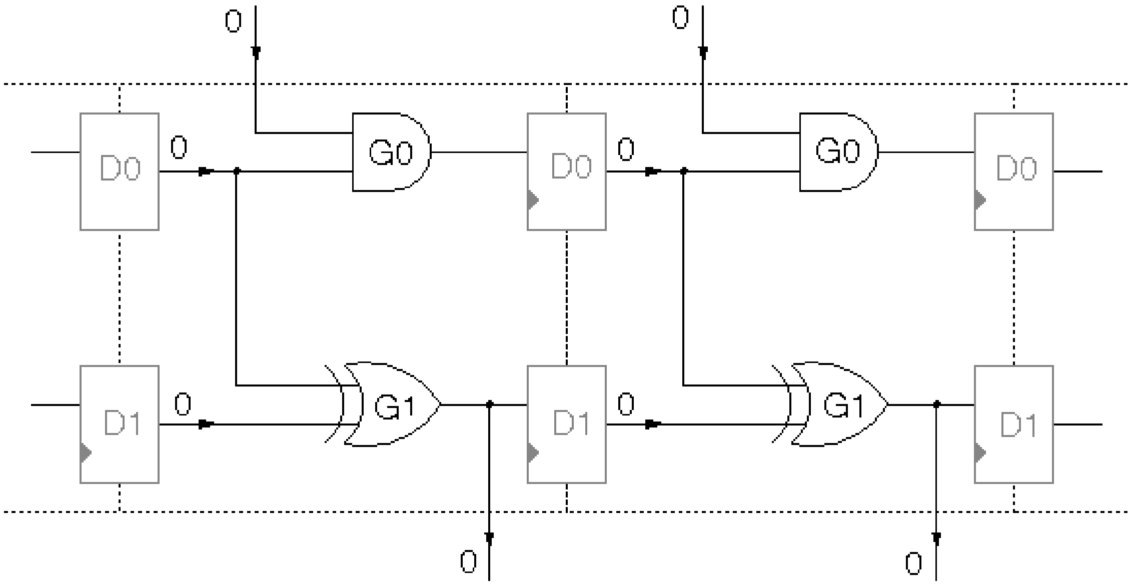
Definition: Gate g is **repairable** for a set of counterexamples if you can correct the faulty circuit by replacing Gate g by some combinational logic in terms of inputs and state variables

- No new flip flops
- Is this a wise choice?
- Alternatives
  - keep same inputs to gate
  - find any realizable function

Theorem: Only repairable gates are valid fault candidates

Fault candidates may not be repairable. Let’s fix that!

Example: Incorrect Fault Candidate

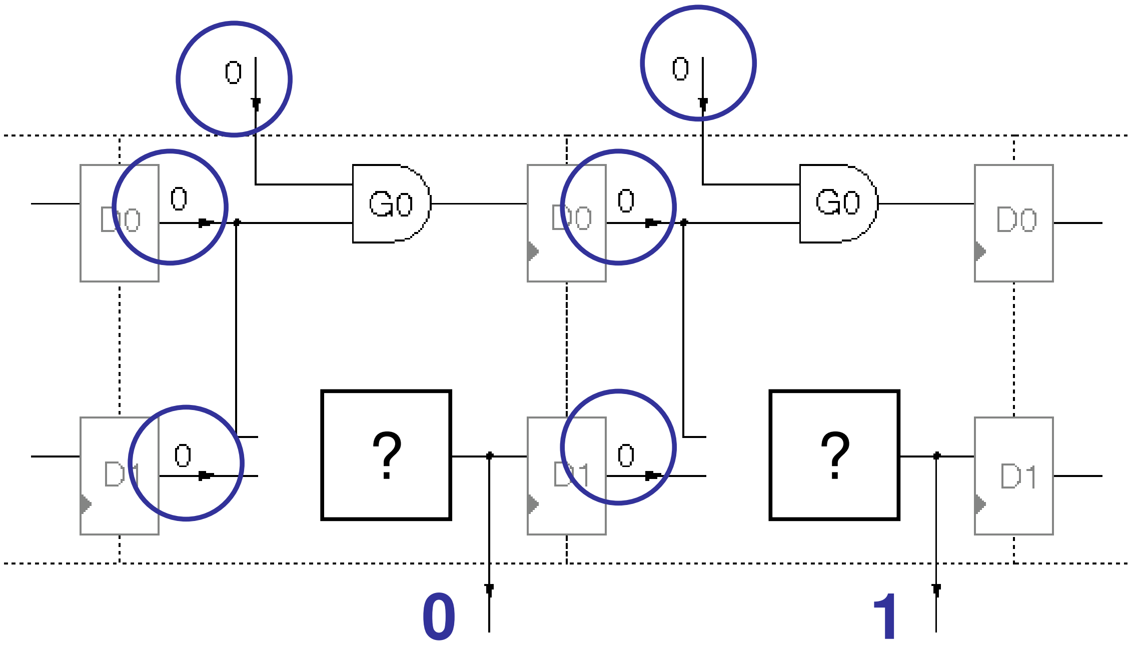


Spec: (out=0) ∧ X(out=1)

Fault candidates: G0, G1

Repairable: G0

Example: Incorrect Fault Candidate



Spec: (out=0) ∧ X(out=1)

Fault candidates: G0, G1

Repairable: G0

There is no combinational repair for G1!

Ackermann Constraints

Let same(i,j) be true if state and inputs are the same in time i and time j.

for all gates g, for all time steps i, j:

$\text{same}(i,j) \rightarrow g(ti)=g(tj)$

#Ackermann constraints ~ (#gates · k^2 · #counterexamples^2)

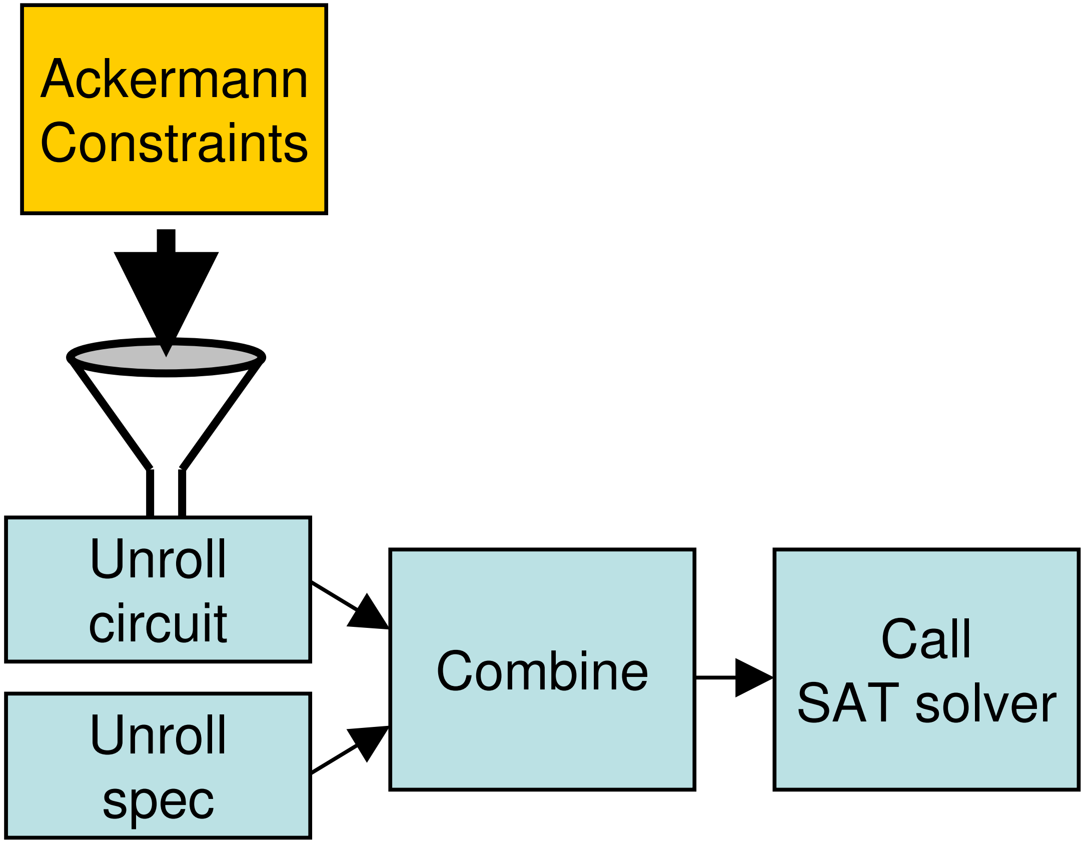
Does not not add decision variables

Theorem:

For every fault candidate there is a repair that works for all counterexamples in the set

One can construct a **repair suggestion** from the satisfying assignments

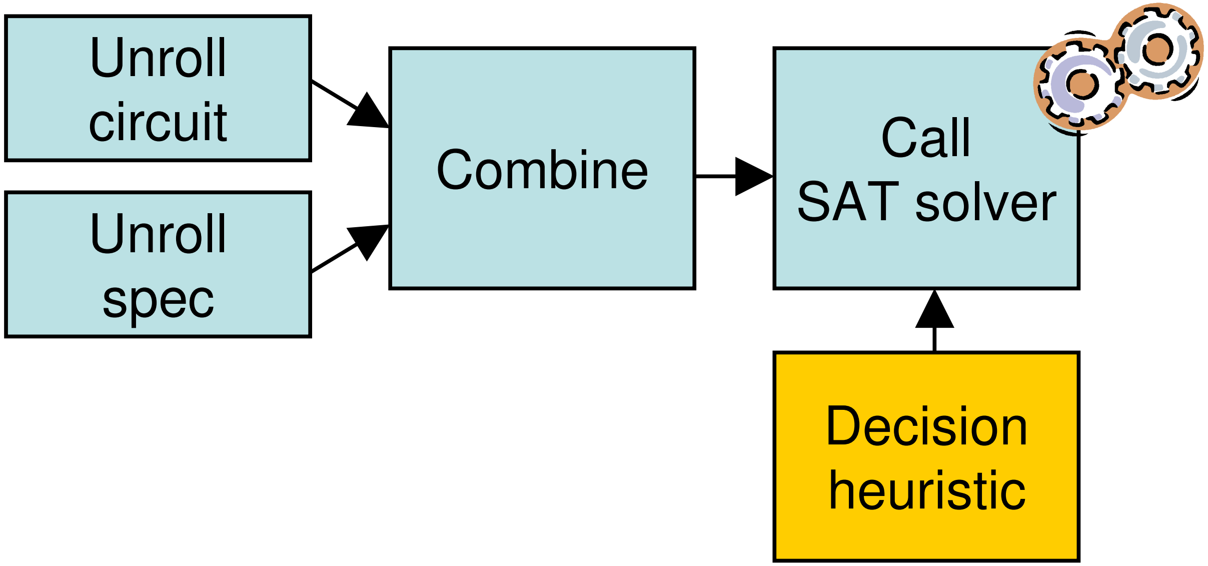
Ackermann Constraints



Efficiency: the SAT solver

Decide, in this order

1. Which component is incorrect
2. The value of this component in time step 1, 2, …
3. Rest is Boolean Constraint Propagation



Unroll circuit

Unroll spec

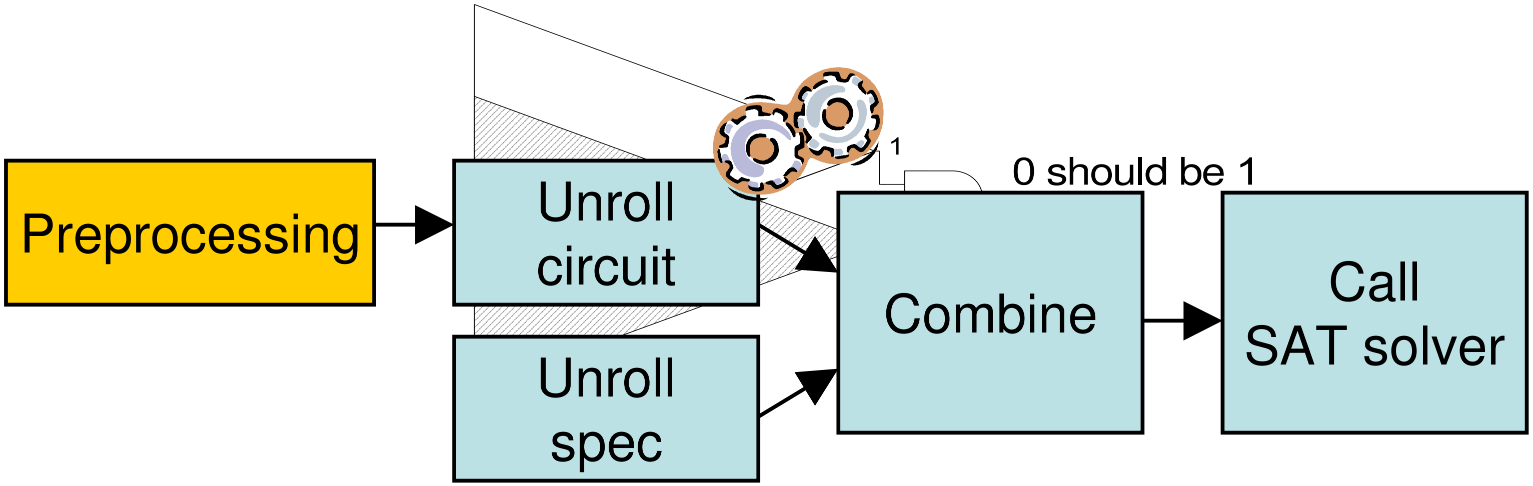
Combine

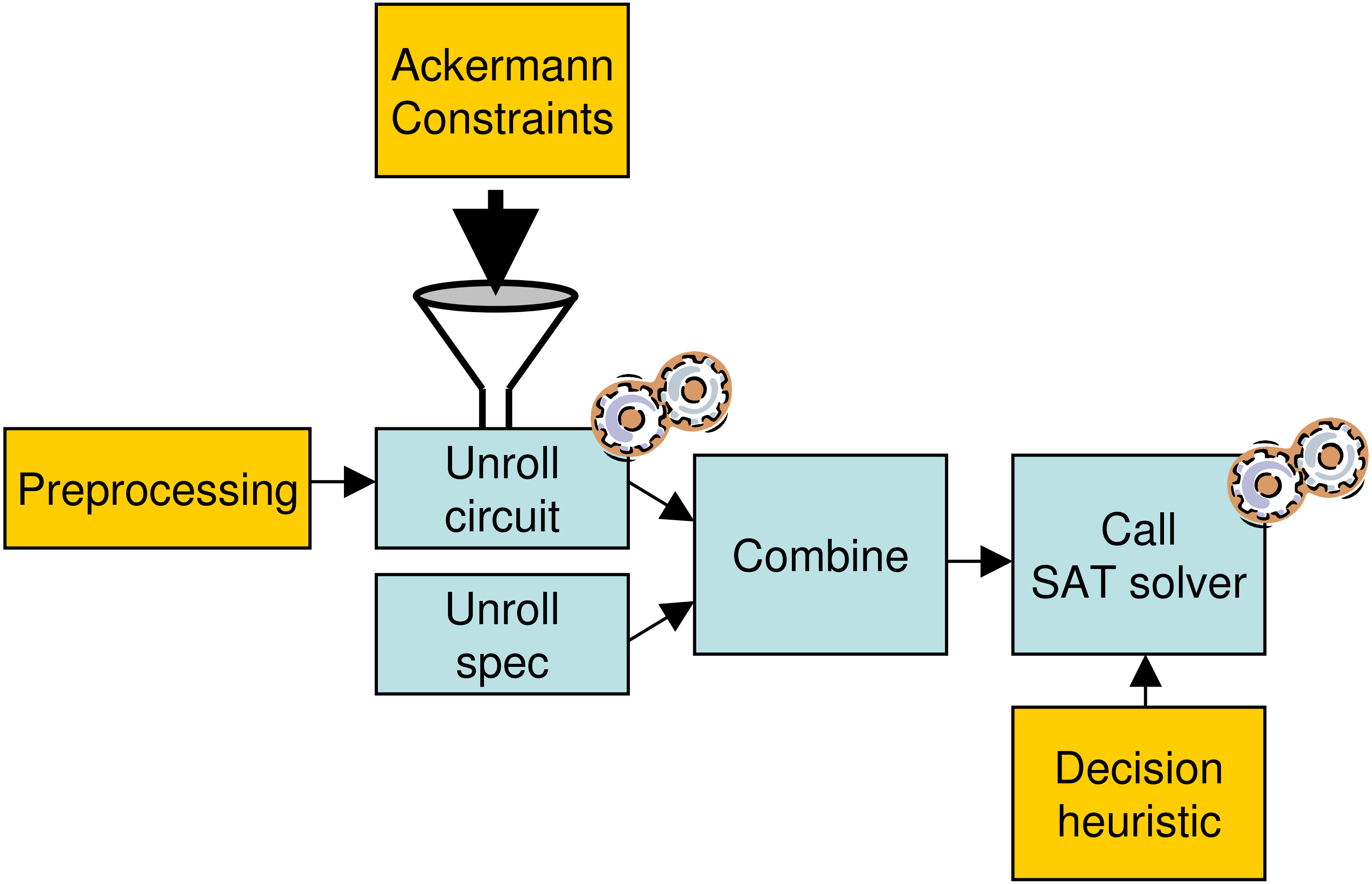
Call SAT solver

Decision heuristic

Efficiency: Simulation Based Preprocessing

Back-propagation constrains the area that contains the fault.





Ackermann Constraints

Preprocessing

Unroll circuit

Unroll spec

Combine

Call SAT solver

Decision heuristic

Source Level

Original Program

```
…
L5: a = b + 1;
```

Annotated Program

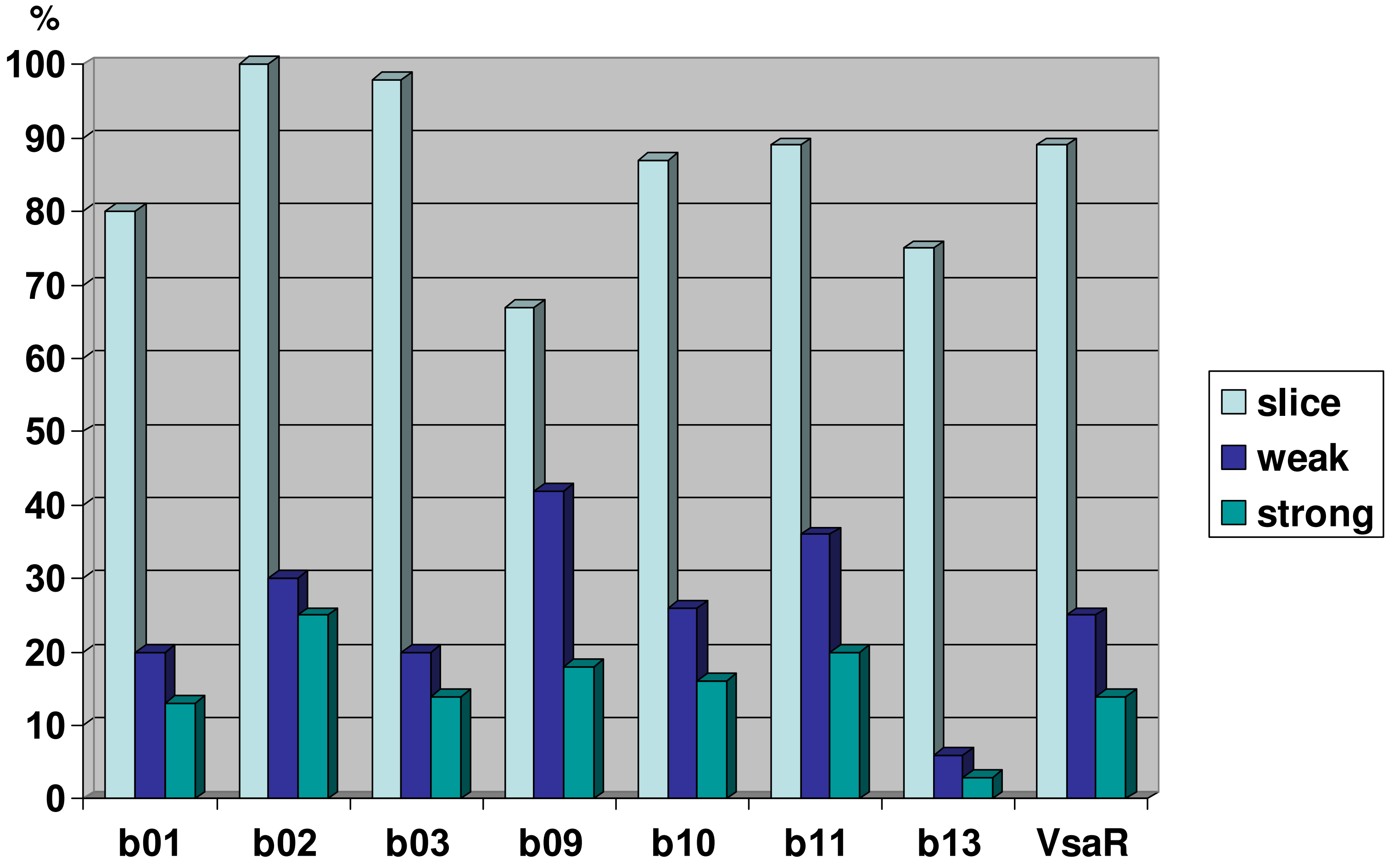
```
abnormal = nondet;
…
if(abnormal != L5)
   a = b + 1;
else
   a = nondet;
```

Experimental Results

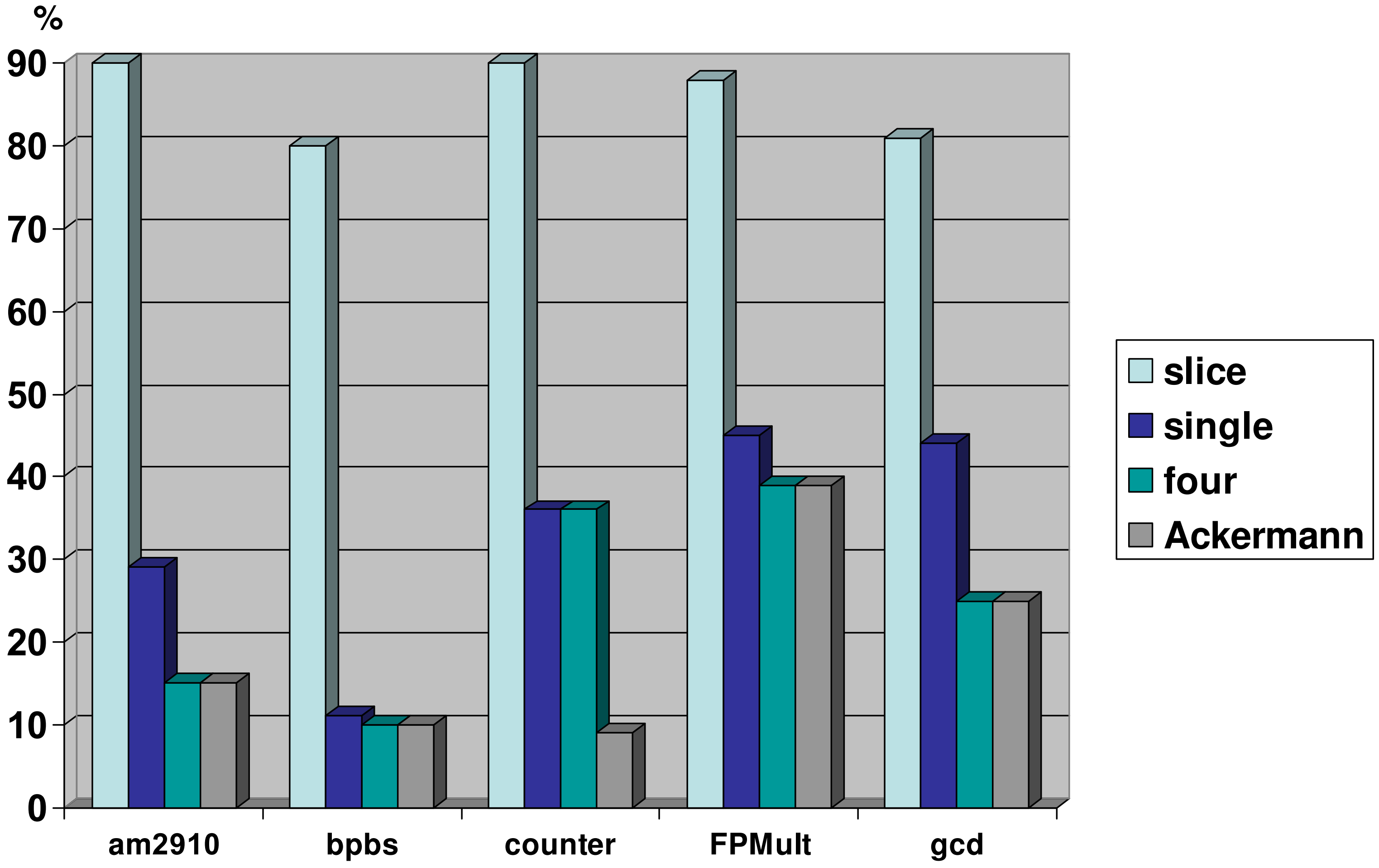
Speed

- Localization time comparable to BMC time
- SAT techniques cause up to 40x speedup
- Speed depends on counterexample length
- Preprocessing:
  - saves runtime in cases where number of components can be reduced
  - otherwise overhead is low

Specificity: Weak & Strong Spec



Specificity: One & Four Counterexamples



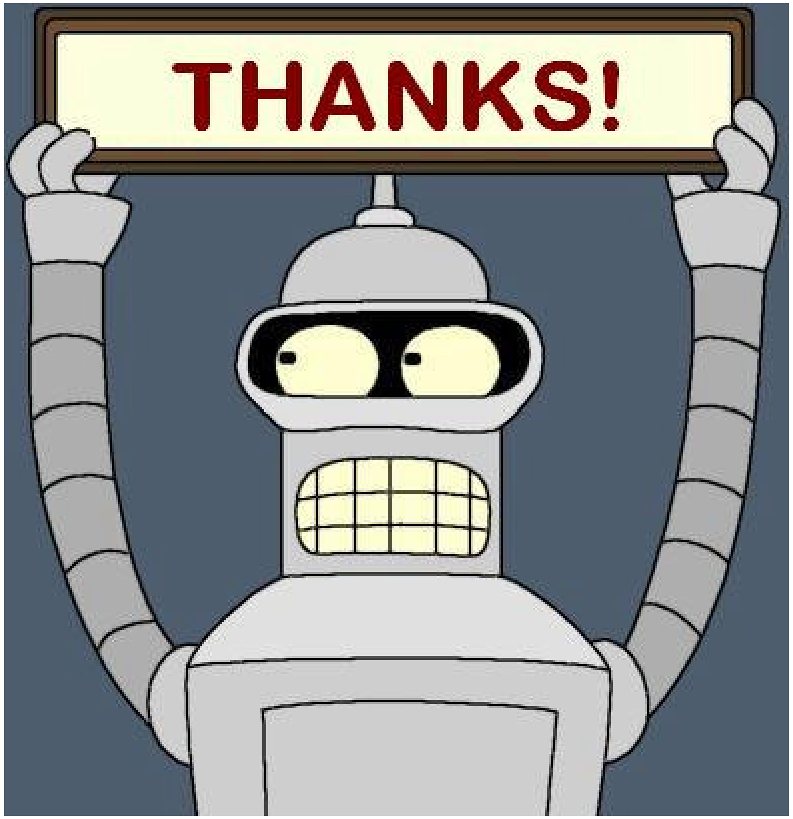
Experimental Results

Specificity (fault candidates/total components)

- Multiple counterexamples improve specificity from 31% to 25% (79% static slice)
- Ackermann constraints improve specificity from 25% to 23%
- Strong specification improves specificity from 26% to 15% (86% static slice)
- Specificity varies by example (3-25% for strong specs)

Conclusion

- Localization finds fault candidates
- Based on BMC (with one extra variable per component)
- Flexible when it comes to input language
- Simple to implement



THANKS!

Automatic Fault Localization for Property Checking

Stefan Staber, Roderick Bloem

Graz University of Technology

Görschwin Fey, Rolf Drechsler

University of Bremen

Specificity: The Specification

A more complete spec yields a better diagnosis

- This is an important factor in specificity!
- more properties may mean less efficiency

Motivation

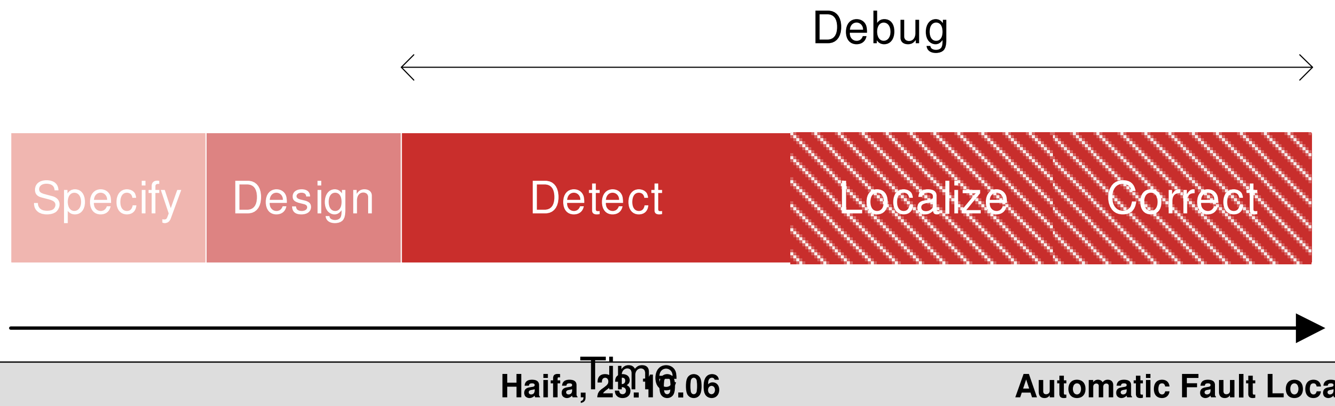
Debugging:

1. Detect failure
2. Localize fault
3. Correct fault

Manual Localization & Correction takes significant time

- Bugs fixes at very end of design cycle (high risk)

Important problem, but little research!



Extensions: QBF

Can we find a fix that works for all inputs?

Alternating quantifiers:

- ∃ diagnosis s.t.∀ inputs in t0 ∃ output in t0 s.t. ∀ inputs in t1 ∃ output in t1

Quantified Boolean Formula

Can we extract a repair from a QBF solver?

Much like [Jobstmann, CAV’05], where we use BDDs

- Faster than BDDs?

Bremen Implementation

Uses hierarchical structure from source to gate level

Modified synthesis tool

Advantage of hierarchical information

Diagnosis granularity

Ask Görschwin for more details!

Graz Verilog Implementation

Requires minimal modifications to VIS-BMC package, easy adaptable

Introduce abnormal predicates

- Simple annotation of source code by Perl script

We negate LTL formula when computing diagnosis, because BMC looks for counterexample

Fix counterexample

- Add to LTL formula

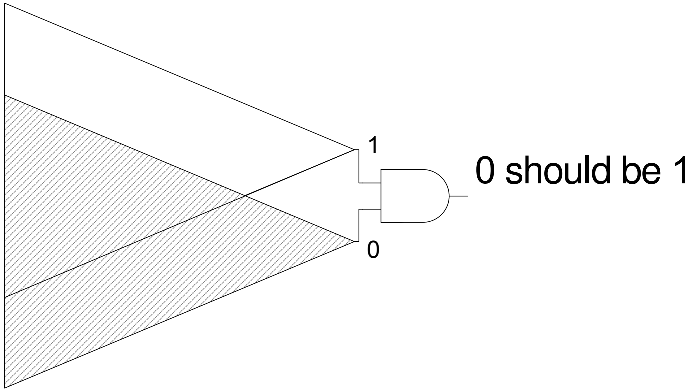
One abnormal predicate

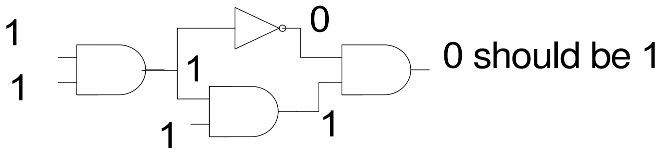
- Add to LTL formula

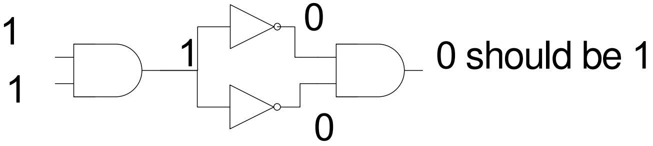
Simulation Based Preprocessing

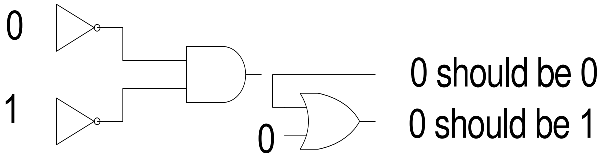
Back-propagation constrains the area that contains the fault.

Three examples in which back-propagation is not perfect.









The Formula

With a SAT solver:

Single fault:

SAT(cex(k) ∧ circuit(k) ∧ property(k) ∧ oneAbnormal ∧ valid=1)

Two faults:

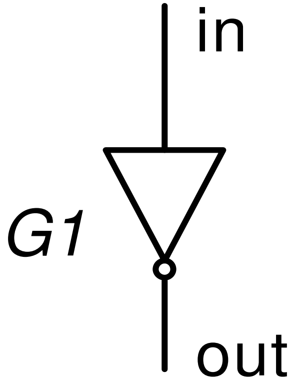
SAT(cex(k) ∧ circuit(k) ∧ property(k) ∧ twoAbnormal ∧ valid=1)

0/1 ILP (PBS): Minimize |abnormal| subject to

cex(k) ∧ SAT(circuit(k) ∧ property(k) ∧ valid=1

Multiple diagnoses: add blocking clauses containing only ab signals.

Example: Unrealizable



Spec: out ↔ X in

Diagnosis: G1

Repairable: -