FudgeFactor: Syntax-Guided Synthesis for Accurate RTL Error Localization and Correction

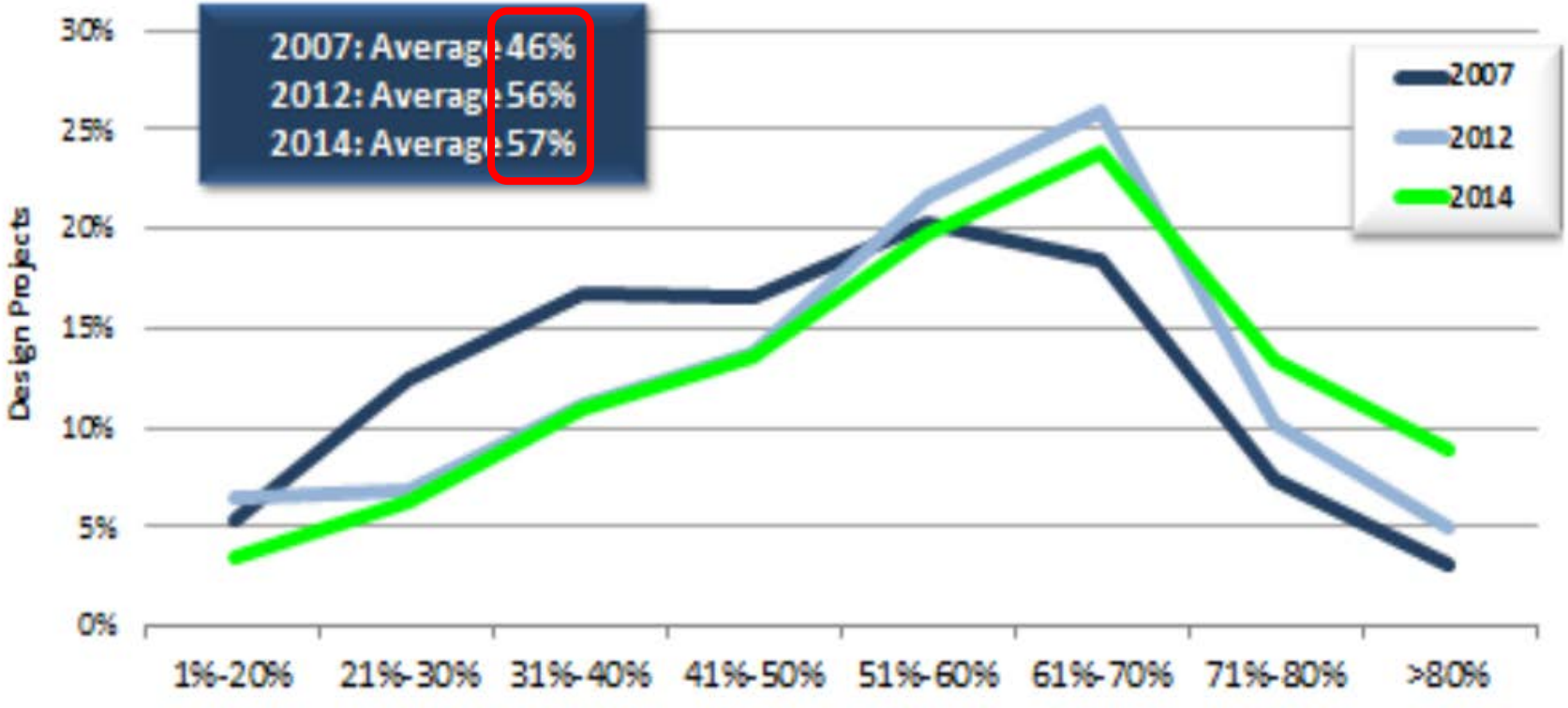
Paolo Ienne[1]

Joint work with Andrew Becker[1], Djordje Maksimovic[2],
David Novo[1], Mohsen Ewaida[1],
Andreas Veneris[2], and Barbara Jobstmann[1]

1: EPFL, Lausanne, CH

2: University of Toronto, Toronto, CA

Debug Time is Out of Control



Foster, H.: ***Trends in Functional Verification: a 2014 Industry Study***. **In** *Proceedings of the 52nd Annual Design Automation Conference (DAC '15).*

Approx. 37% of verification time is debug time.
→ Debug time is approx. 20% of the avg. total project time!

Key Insight I

- Engineers can spend hours debugging, only to find trivial root causes.
- Not an efficient use of engineer time.

If we could automatically fix simple errors,
we could save significant debugging time.

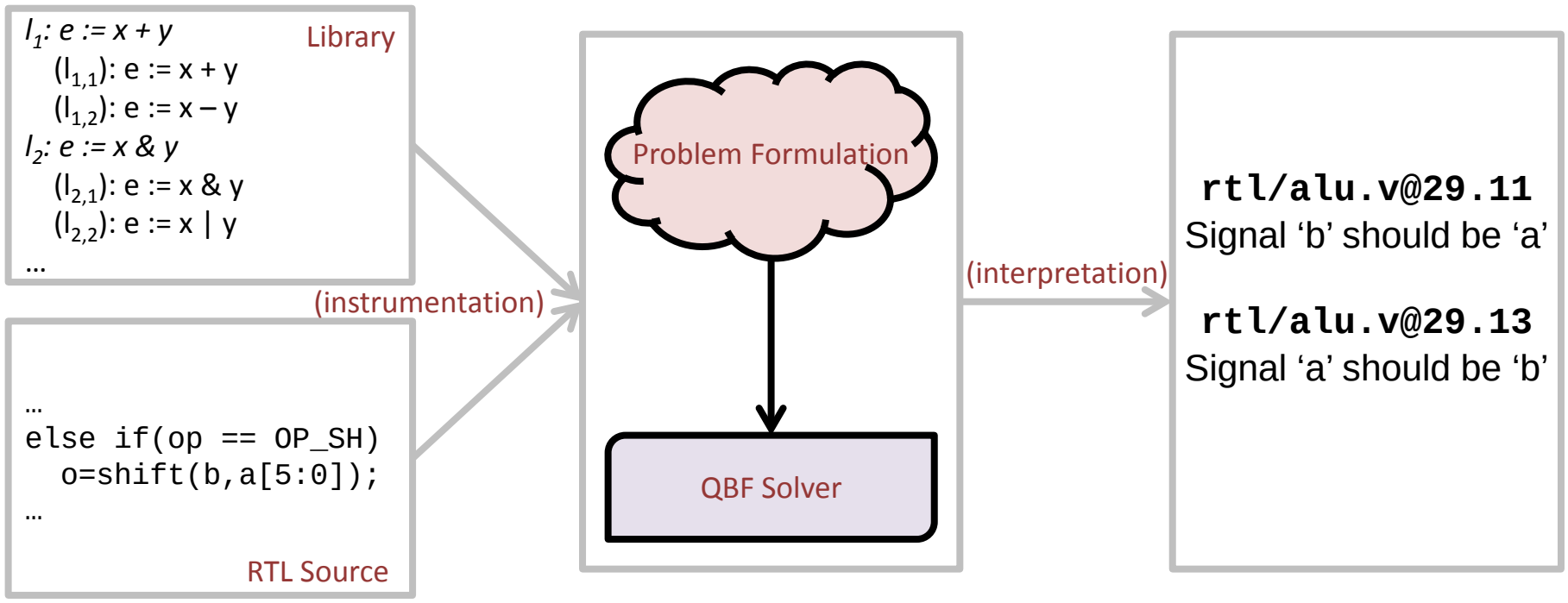
Key Insight II

- Some design errors are not modeled well by previous approaches (e.g. “wrong gate”).
  - Imagine an erroneous ‘+’ instead of ‘–’: many incorrect/missing gates!
- Many are *syntactically-close* to correct RTL, even if the resulting circuit is *semantically-far.*

Use the *almost-correct* RTL and a model of common errors to synthesize the correct design.

3,000m Overview

1) Build library of common RTL errors: assume simple, common errors.
2) Add *possibility* of incorporating suitable fixes for all matched suspected errors.
3) Solver finds if some combination actually fixes the error.

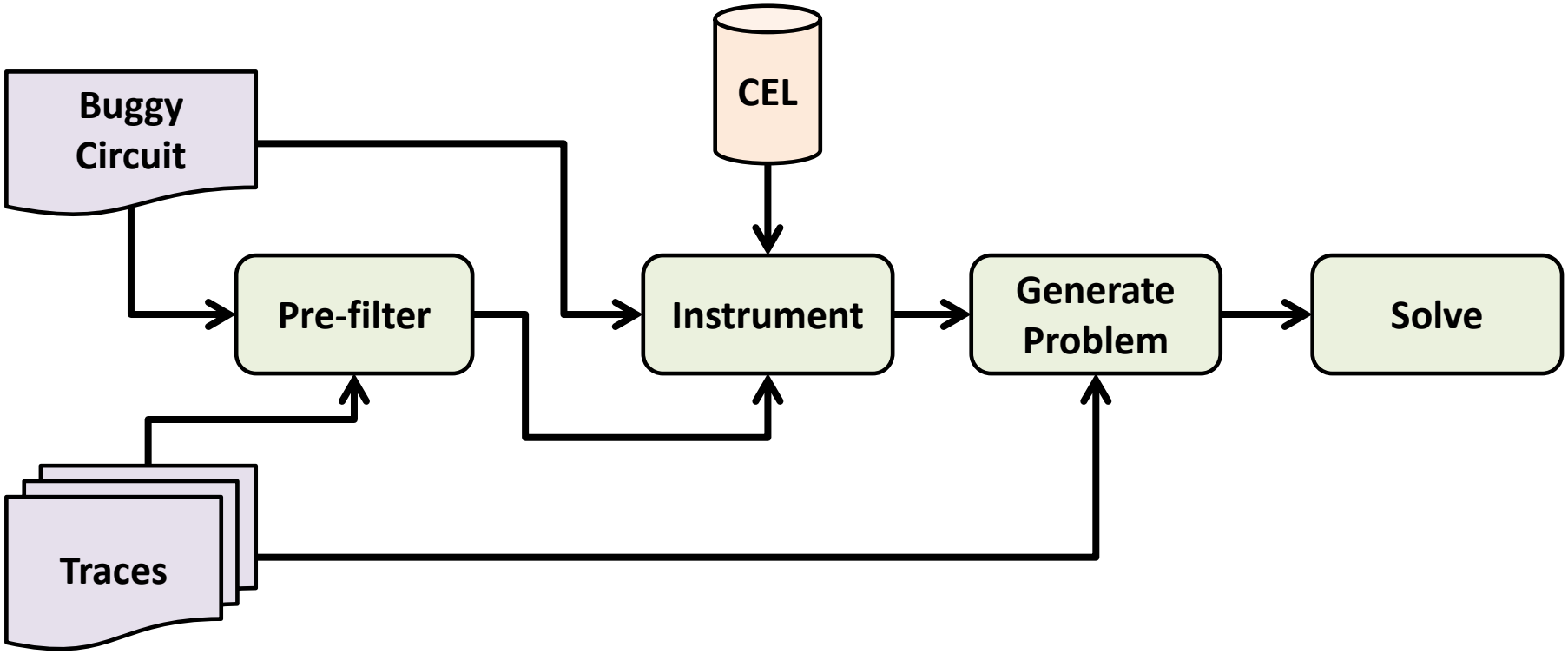


Buggy circuit design and library of common RTL errors provided to software suite

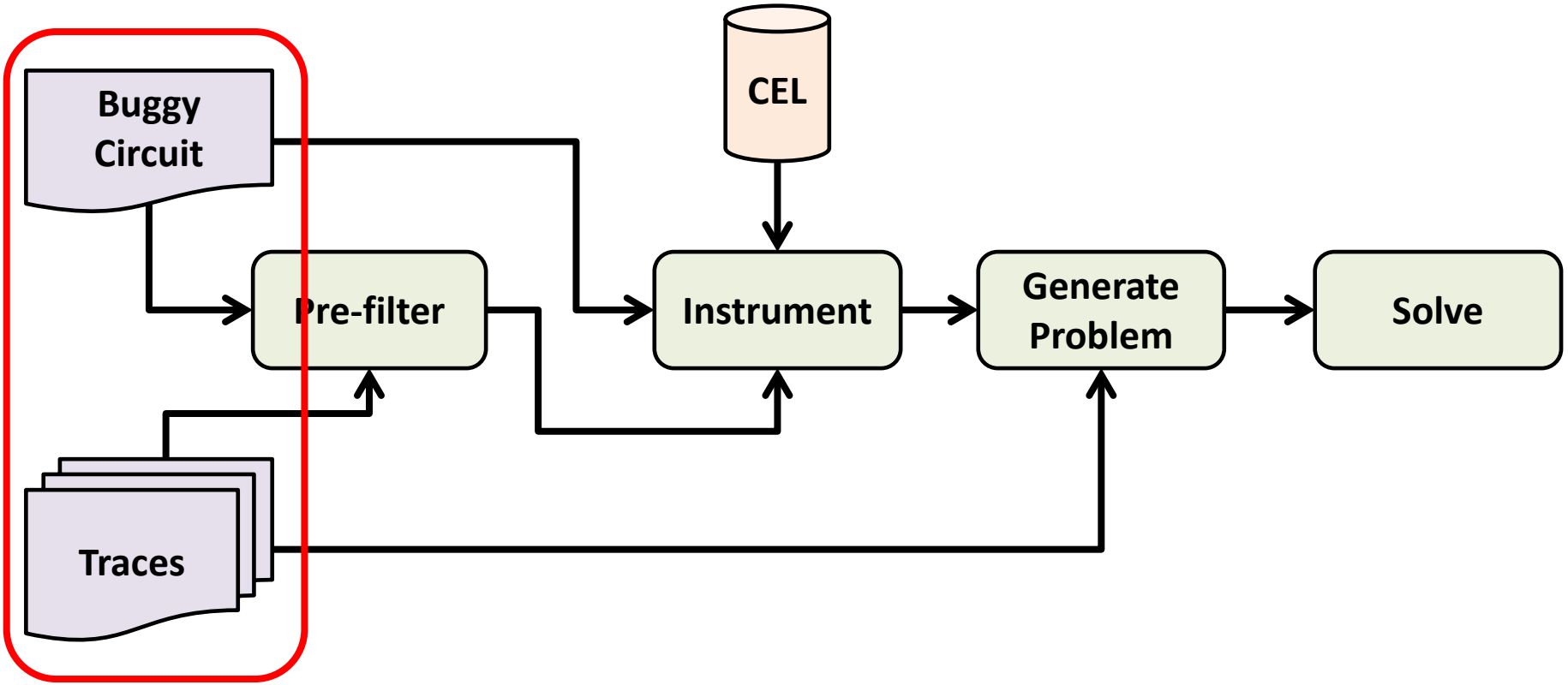
Software tools determine suspicious RTL, apply matching error rules, and find fixing combination(s)

Designer gets back *meaningful* error diagnosis exactly describing the problem and necessary fix

1,500m Overview

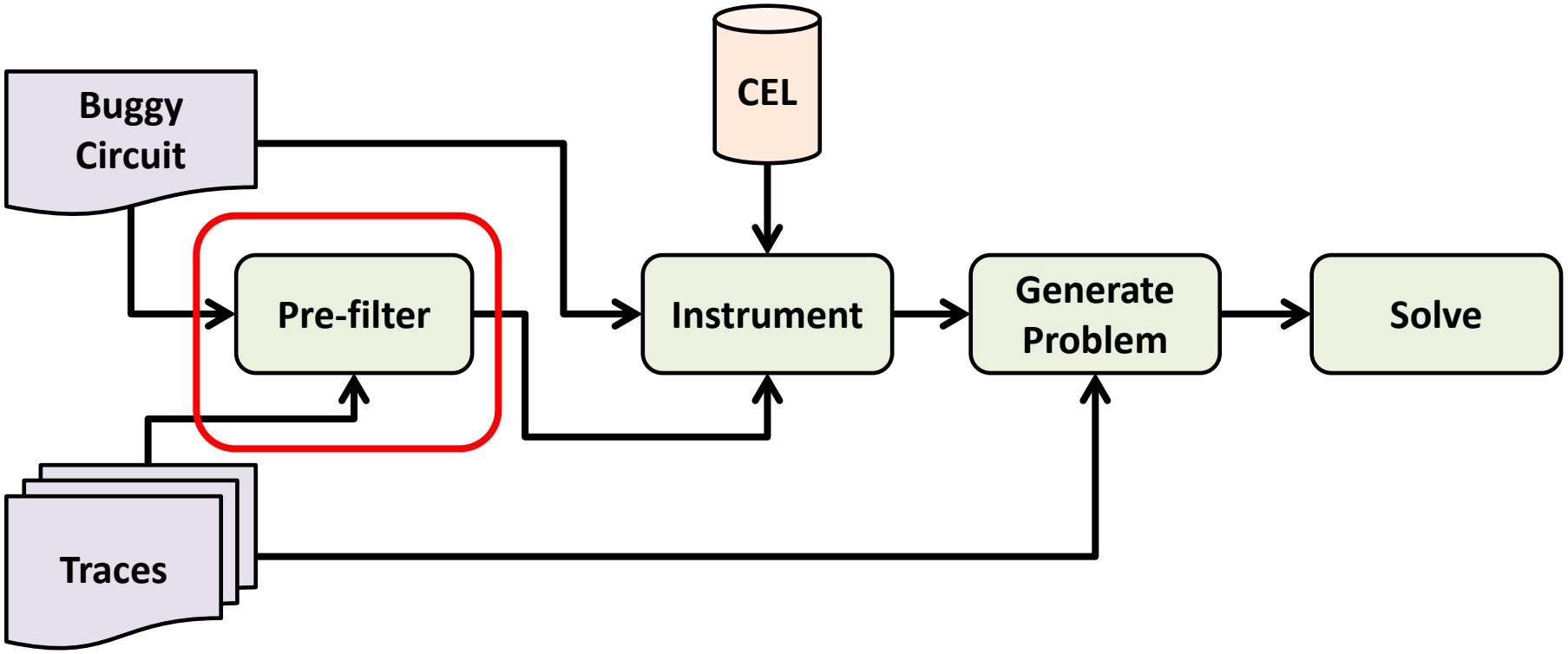


1,500m Overview



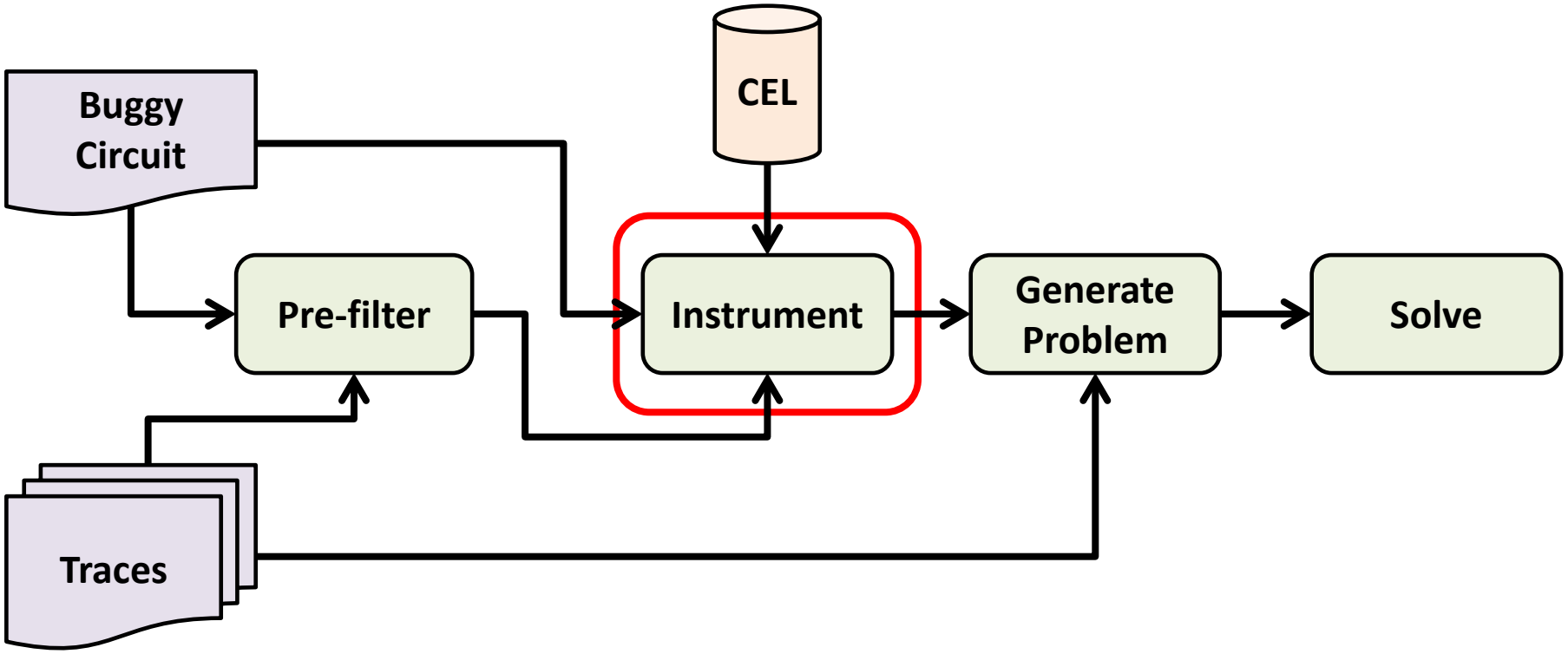
Start with a buggy circuit and erroneous test vectors

1,500m Overview



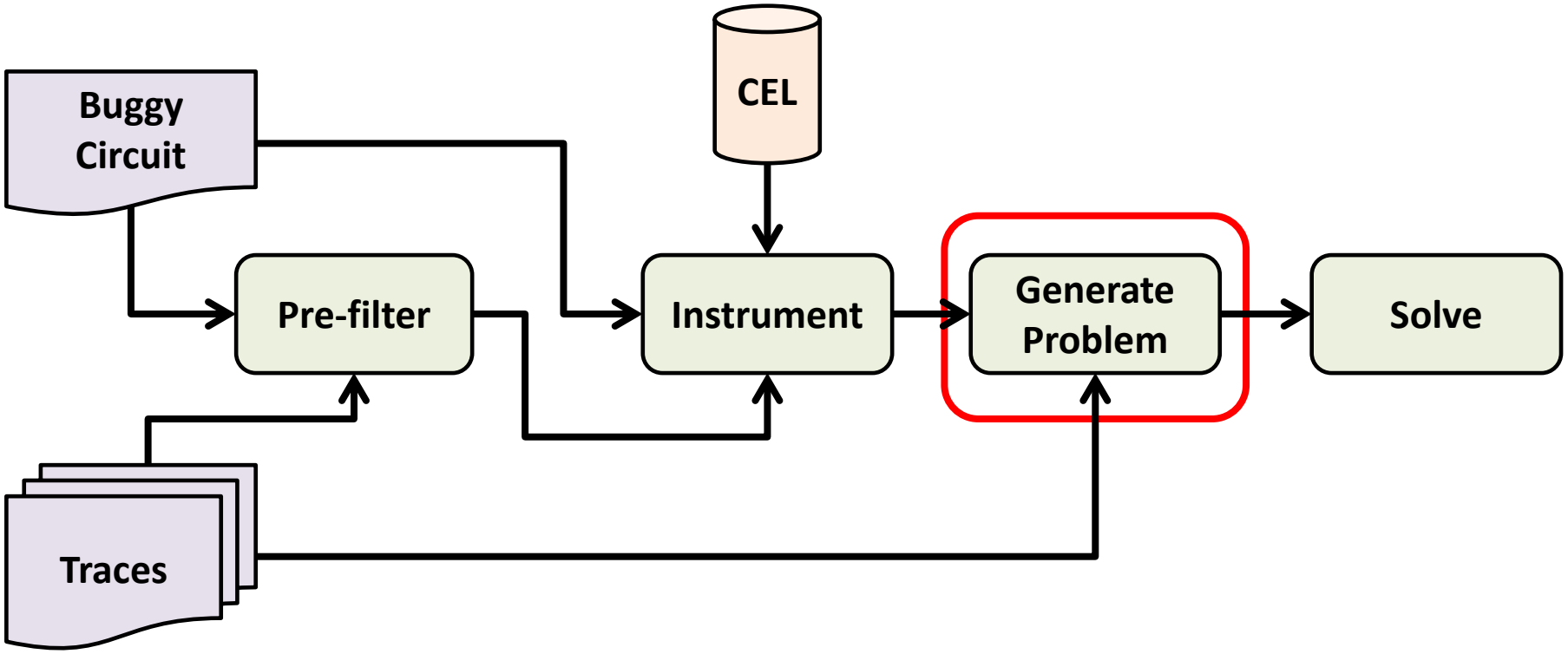
Find suspect locations and pre-filter

1,500m Overview



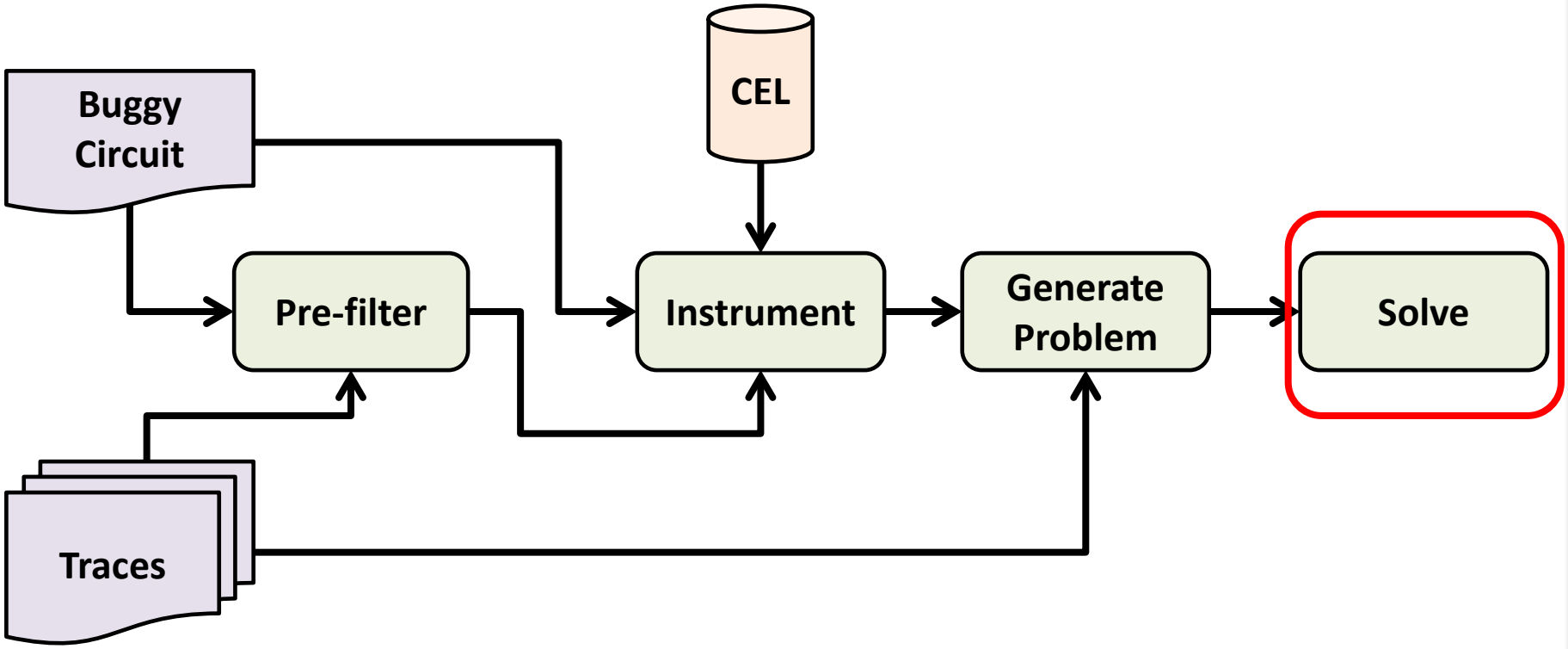
Use “common error library” to add possible fixes

1,500m Overview



Use traces to generate a problem instance

1,500m Overview



Solve*: find which potential fixes actually correct errors

*Using Solar-Lezama's CEGIS solver; now we also support Yices

Fault Localization Pre-filter

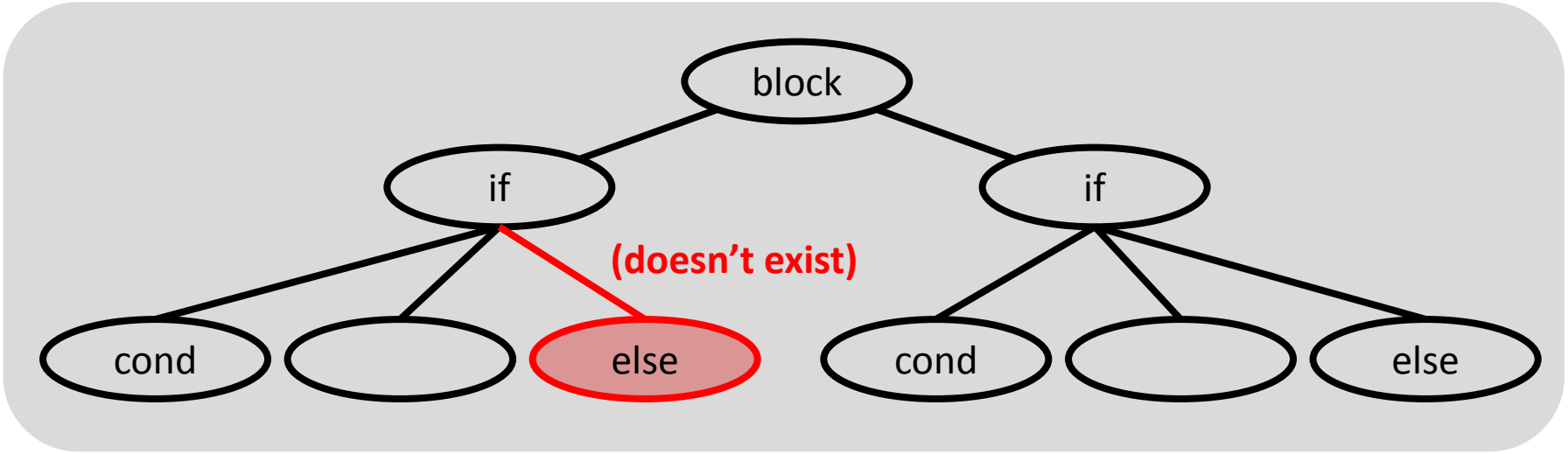
- We use a commercial tool based on existing localization approach [1] to pre-select areas of the circuit on which to focus.
  - Tool output has too many false-positives.
  - We increase specificity and avoid designers chasing false leads.
- Only apply rule matching and instrumentation on these suspect areas.

[1] A. Smith, A. Veneris, M. F. Ali, A. Viglas.
Fault Diagnosis and Logic Debugging Using Boolean Satisfiability. *IEEE TCAD*, October 2005.

Common Error Library

- Extensible library of ‘rules’ heuristically modelling and correcting typical errors.
- Explicitly modeled by humans (by the tool designers—not circuit designers).
- Mostly based on matching fragments of the *Abstract Syntax Tree* (AST).
  - Special kind of specification similar to subgraph isomorphism; extra conditions sometimes req’d.
- Unroll sequential circuits to depth necessary.

Example: Error Rule C



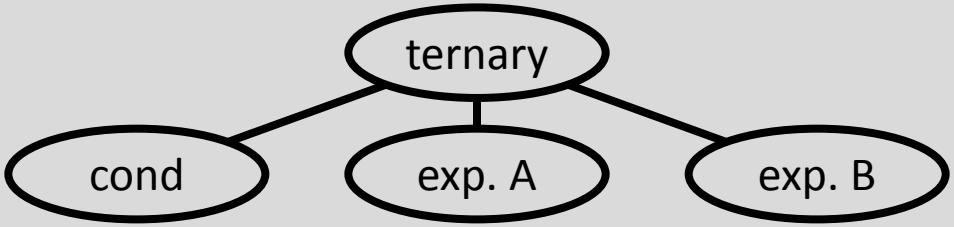
Matches:

```
if(…)
  …
if(…)
  …
else
  …
```

Allows Option Of:

```
if(…)
  …
else if(…)
  …
else
  …
```

Example: Error Rule G



Matches:

cond? A : B

⇒

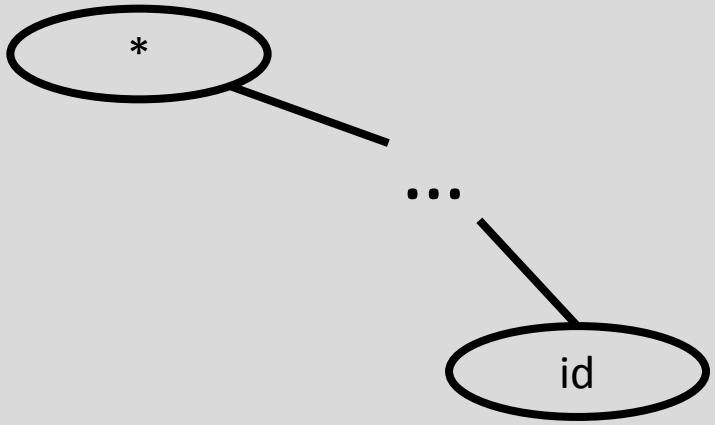
Allows Option Of:

cond? B : A

Example: Error Rule D

*—one of:

- Assign
- Statement
- Port connection



Matches:

any identifier in a 'right hand side' usage

z = x + y

e.g.

Allows Option Of:

any electrically-compatible identifier

z = x + a

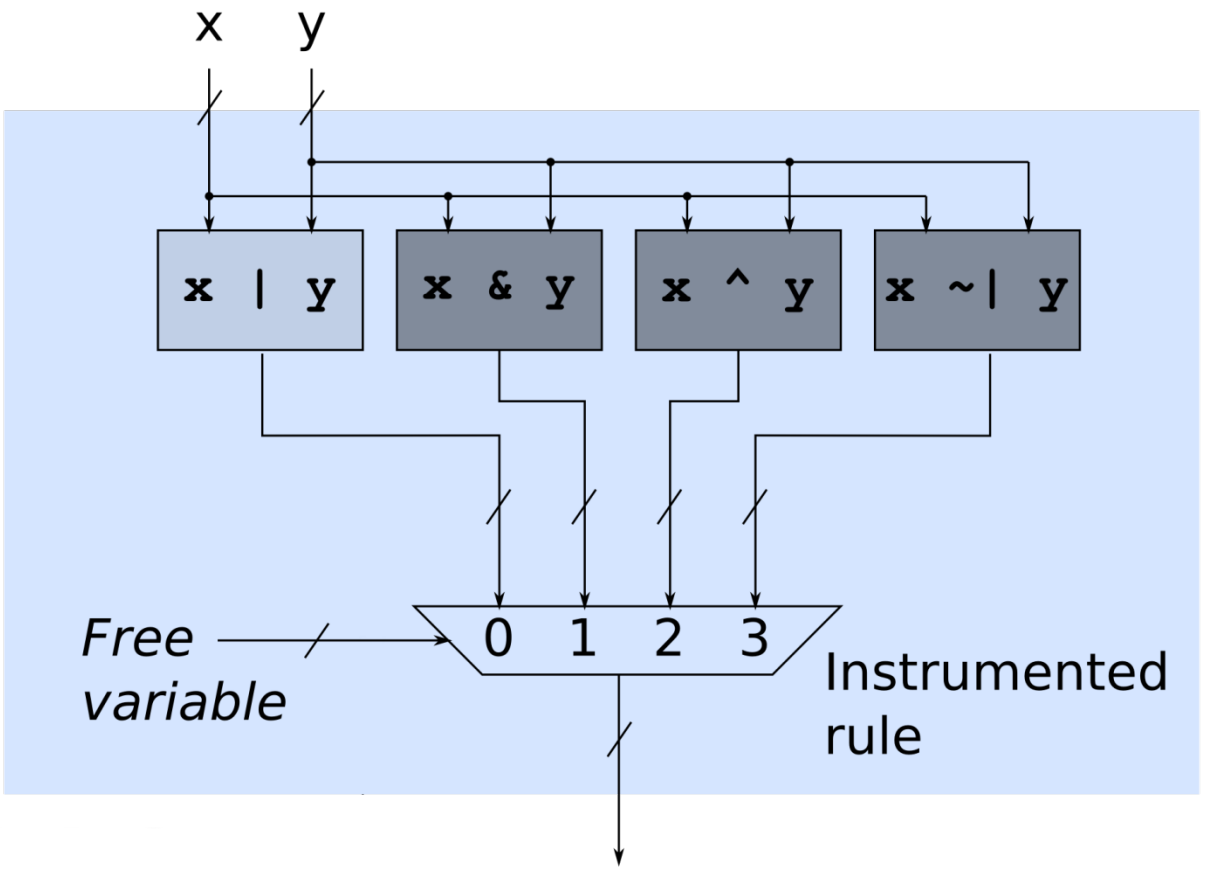
Rules List

Rule	Checker (if the subgraph looks like...)	Transformer (insert these options...)
A	Signal indexing operation	Indices and ranges may be shifted to the left or right by one.
B	Incomplete case without default	Signals assigned in case get a default assignment of any compatible signal, or a pure free variable.
C	If ... If ... Else assigning the same signal	Allow use of a parallel If ... Else If ... Else with the same conditions.
D	Signal in any statement explicitly mentioned in candidate set	Allow referring instead to any compatible signal.
E	A bitwise comparison operator	Allow comparing with any other bitwise comparison operator instead.
F	A constant value on right-hand side; not an index/range	Allow using instead any constant value (a pure free variable).
G	A ternary expression	Allow using instead the same ternary expression, but with the condition inverted.

A total of 7 general rules are implemented now, but *nearly* any syntactic change could be modeled.

Rule Application Example

- Original
  "x | y"
  might be:
  "x | y" or
  "x & y" or
  "x ^ y" or
  "x ~| y"



- Free variables select which behavior is actually exposed.

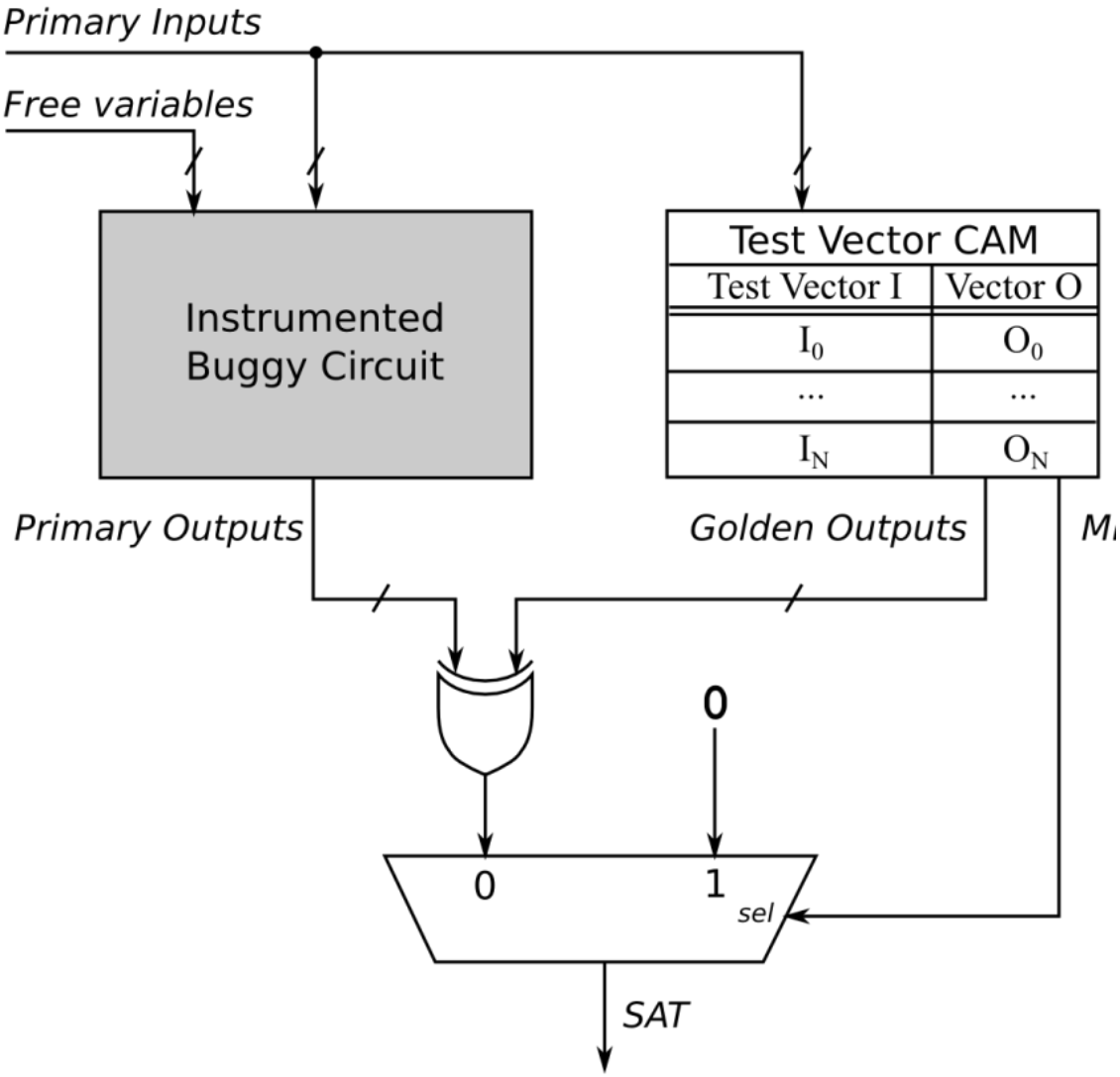
Limits of Rule Applicability

- *Almost* any syntax changes can be modeled.
- Cannot model changes to areas which must be statically determined at synthesis time.
  - "initial" blocks (if anyone cares)
  - "for" generate loop bounds
  - "synopsys translate_off"-style directives

Specification

- Formal specifications not always available.
  - Test benches with millions of vectors are not feasible to use as ‘black box’ specifications.
- Compromise: use (very) abstract specification.
- Spec. is just one known-failing test vector and two others, to cover other parts of the design.
  - Intuition: syntax guidance → less need for exactness.
  - Totally arbitrary, but works well so far.
  - More (and more general) rules may require more precise specification.

Specification II



Potential Pollution

- With so many changes allowed, solution space can be filled with over-complicated solutions.

(original)

```
if(A == 1'b0)
  Z = X;
else
  Z = Y;
```

(proper fix)

```
if(A == 1'b0)
  Z = E;
else
  Z = F;
```

2 changes

Potential Pollution

- With so many changes allowed, solution space can be filled with over-complicated solutions.

(original)

```
if(A == 1'b0)
  Z = X;
else
  Z = Y;
```

(pollution)

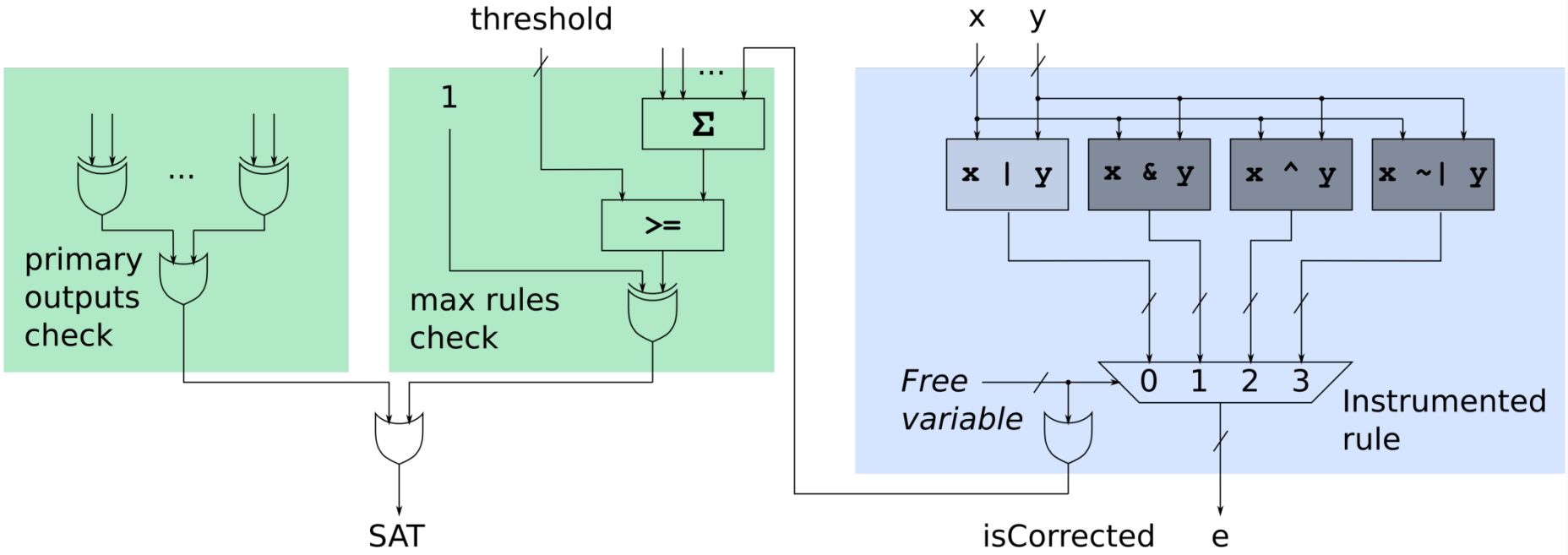
```
if(A != 1'b1)
  Z = E;
else
  Z = F;
```

4 changes

Avoiding Pollution

- Further constrain the free variables.
- No more than *t* free vars. may be non-zero.
  - **I.e., maximum *t* simultaneous corrections.**
  - Successively increase this threshold *t* until we find corrections, or exceed a maximum threshold.
    - Simple linear sweep; use binary search if many corrections are allowed.

Final Specification



Not only do the primary outputs (*e* here) have to match, but the number of applied corrections must be below some threshold. This threshold is then swept to find the minimal corrections.

Experimental Methodology

- First three designs are from OpenCores; CPU is from GitHub [2].
- We used the CPU as a rule demonstrator.
  - Only a sample of injected errors presented here.
- All other designs use only ‘real’ bugs from commit history or bugs injected by third party.
  - **Not used in any way to develop rules.**

[2] https://github.com/jmahler/mips-cpu
http://opencores.com/project,divider
http://opencores.com/project,aes_core
http://opencores.com/project,simple_spi

Experiments

- Multiple buggy versions of four designs:
  - SPI: SPI master controller
    - ~15k AND-Inverter gates after unrolling
  - AES: Pipelined 128-bit AES module
    - ~87k AND-Inverter gates after unrolling
  - Div: Pipelined signed-by-unsigned integer divider with 16-bit dividend and 8-bit divisor
    - ~97k AND-Inverter gates after unrolling
  - CPU: Basic 5-stage pipelined MIPS processor
    - ~35k AND-Inverter gates after unrolling

Example of Corrected Error

- A typical ‘copy & paste’ error in one version of the SPI design (spi_bug4).

(original)

```
assign wp_p1 = wp + 2'h2;
assign wp_p2 = wp + 2'h2;
```

(corrected)

```
assign wp_p1 = wp + 2'h1;
assign wp_p2 = wp + 2'h2;
```

Experimental Results I

Buggy Design	# RTL Changes	Solved?	Fixing Rule(s)	Matched Rules	Total AST Size	# Matched AST Nodes	SLOC
spi_bug1	1	✓	D	ABDEF	2968	20	271
spi_bug2	–	–	–	BD	2964	2	266
spi_bug3	–	–	–	DEF	2968	10	266
spi_bug4	1	✓	F	ABDF	2968	13	266
aes_bug1	–	–	–	ADFG	5080	19	467
aes_bug2	1	✓	D	ABDG	5251	33	467
div_bug1	–	–	–	ADF	2486	13	163
div_bug2	2	✓	DD	AD	2478	8	165
div_bug3	–	–	–	ADF	2486	13	165
div_bug4	1	✓	D	ADF	2502	10	165
div_bug5	–	–	–	ADF	2516	15	168
div_bug6	–	–	–	ADF	2528	20	165
div_bug7	2	✓	DD	ADF	2510	12	165
cpu_bug1	1	✓	B	BDG	3842	4	530
cpu_bug2	1	✓	C	CDEF	3846	5	531

8/15 corrected *properly*; signal replacement rule by far most common.

Example of Not Corrected Error

- Some missing functionality in part of the key expansion in AES (aes_bug1).
- Note: not *fundamentally* uncorrectable.

(original)

```
always @(posedge clk)
w[0] <= #1 kld?
        key[127:96] :
        w[0] ^ rcon;
```

(NOT corrected)

```
always @(posedge clk)
w[0] <= #1 kld?
        key[127:96] :
        w[0] ^ subword ^ rcon;
```

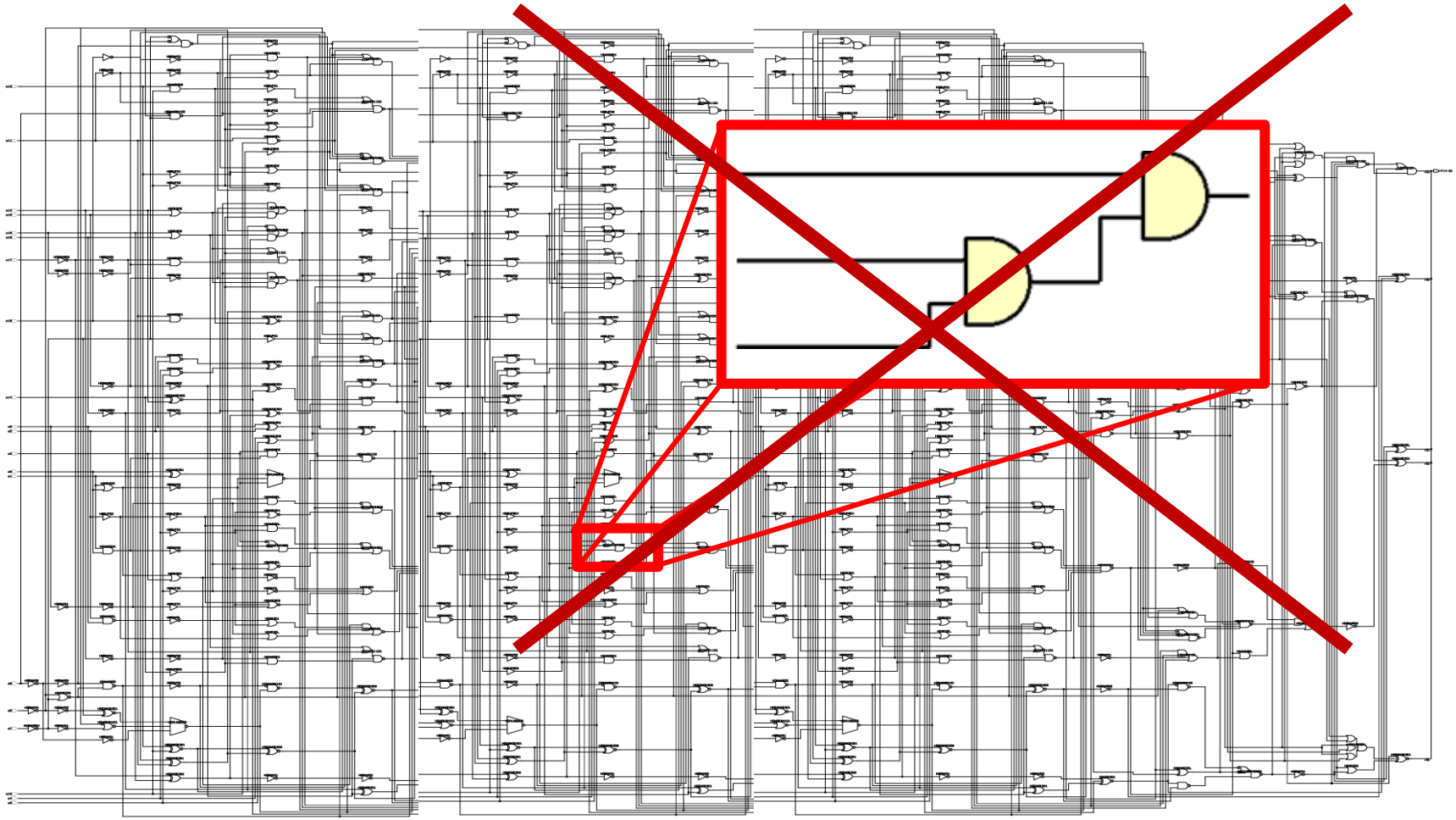
Experimental Results II

All *answered* in <10min.

This is why we pre-filter!

Buggy Design	# Free Var. Bits	Total Solver Time (s)	# Golden Gates	Unroll Frames	Blowup
spi_bug1	92	1.90	14468	20	2.94x
spi_bug2	8	1.69	14468	20	1.20x
spi_bug3	35	2.23	14468	20	1.90x
spi_bug4	65	1.66	14468	20	2.14x
aes_bug1	373	18.71	86878	6	1.07x
aes_bug2	62	517.40	86878	6	1.29x
div_bug1	33	32.28	96767	48	2.30x
div_bug2	20	71.47	96767	48	2.12x
div_bug3	30	21.82	96767	48	2.28x
div_bug4	26	78.90	96767	48	2.24x
div_bug5	37	49.05	96767	48	3.20x
div_bug6	32	17.75	96767	48	1.99x
div_bug7	30	101.46	96767	48	3.15x
cpu_bug1	12	87.53	34294	15	2.28x
cpu_bug2	46	60.05	34294	15	2.56x

A New Help for Debugging



A New Help for Debugging

```
[...]

always @(state)
  begin
     case (state)
       zero:
         out = 4'b0000;
       one:
         out = 4'b0001;
       two:
         out = 4'b0010;
       three:
         out = 4'b0100;
     endcase
  end

always @(posedge clk or posedge reset)
  begin
     if (reset)
       state = zero;
     else
       case (state)
         zero:
           state = three;
         one:
           if (in)
             state = zero;
           else
             state = two;
         two:
           state = one;
         three:
           state = zero;
       endcase
  end

[...]
```

```
default:
  out = 4'b0000;
```

```
one;
```

```
three;
```

Conclusions

- All solutions found were actual, proper fixes.
  - Not guaranteed to be true!
  - Parameters (e.g. no. of traces) can be tweaked.
  - Needs more thorough investigation.
- Healthy proportion of designs were corrected.
- Objectively reasonable run times.
  - Run this *first* upon error discovery; debug manually in parallel. No time wasted.

A. Becker, D. Maksimović, D. Novo, M. Owaida, A. Veneris, B. Jobstmann, and P. Ienne.
FudgeFactor: Syntax-guided synthesis for accurate RTL error localization and correction.
In *Proceedings of the 11th Haifa Verification Conference*, pages 259-275, Haifa, Israel, November 2015.