Scalable Exploration of Functional Dependency by Interpolation and Incremental SAT Solving

Chih-Chun Lee[§], J.-H. Roland Jiang[§],
C.-Y. Ric Huang[§], and Alan Mishchenko[¶]

[§]National Taiwan University
[¶]University of California, Berkeley

Outline

- Introduction
- Prior work
- Our approach
- Experimental results
- Conclusions

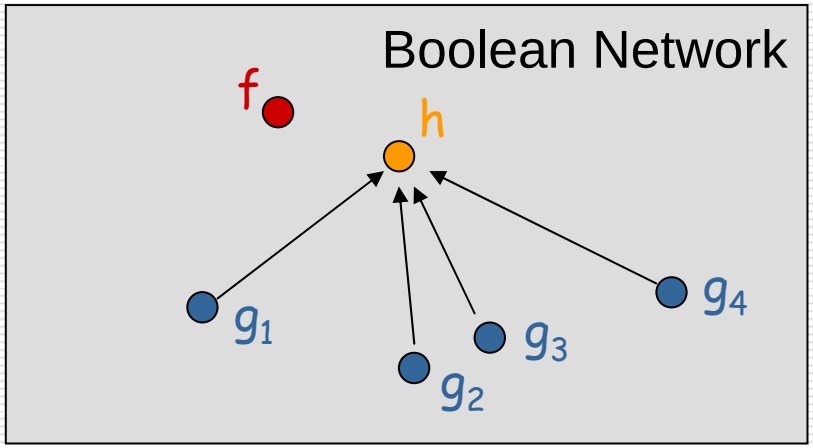
Introduction

- Functional dependency
  - $f(x) = h(g_1(x), g_2(x), \ldots, g_m(x)) = h(G(x))$
  - Under what condition can a function f be expressed as some function h over a set of functions G?
  - h exists $\Leftrightarrow \nexists a,b$ such that $f(a) \neq f(b)$ and $G(a)=G(b)$

i.e., G is more distinguishing than f

Applications

- Resynthesis/rewiring [KK04]
- Redundant register removal [LN91,STB96]
- BDD minimization [HD93]
- Verification reduction [JB04]
- …



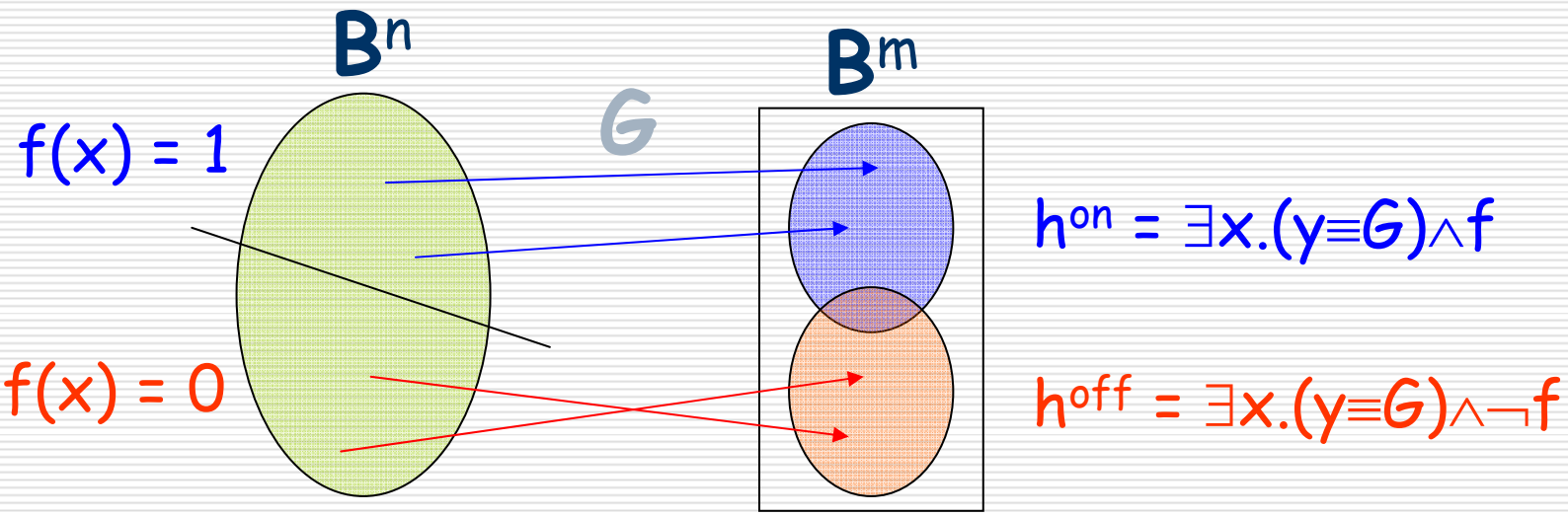
target function
base functions

Prior Work

- BDD-based computation of h

$h^{on} = \{y \in \mathbf{B}^m : y = G(x) \text{ and } f(x) = 1,\ x \in \mathbf{B}^n\}$

$h^{off} = \{y \in \mathbf{B}^m : y = G(x) \text{ and } f(x) = 0,\ x \in \mathbf{B}^n\}$



BDD-based Computation

- Not robust
  - 2 image computations for every choice of G
  - Inefficient when $|G|$ is large or when there are many choices of G
- Move on to SAT?

SAT-based Computation

- h exists $\Leftrightarrow$

  $\nexists a,b$ such that $f(a) \neq f(b)$ and $G(a)=G(b)$,

  i.e., $(f(x) \not\equiv f(x^*)) \wedge (G(x) \equiv G(x^*))$ is UNSAT

- But how to derive h?

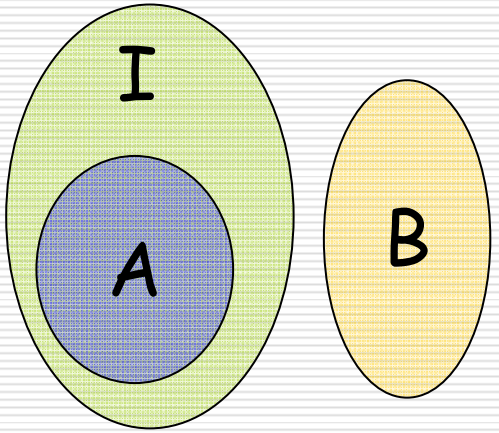
  Moreover how to select G?

Craig Interpolation

- $A \wedge B$ is UNSAT for clause sets A and B.

  Then there exists an interpolant I of A such that

  1. $A \Rightarrow I$
  2. $I \wedge B$ is UNSAT
  3. I refers only to the common variables of A and B



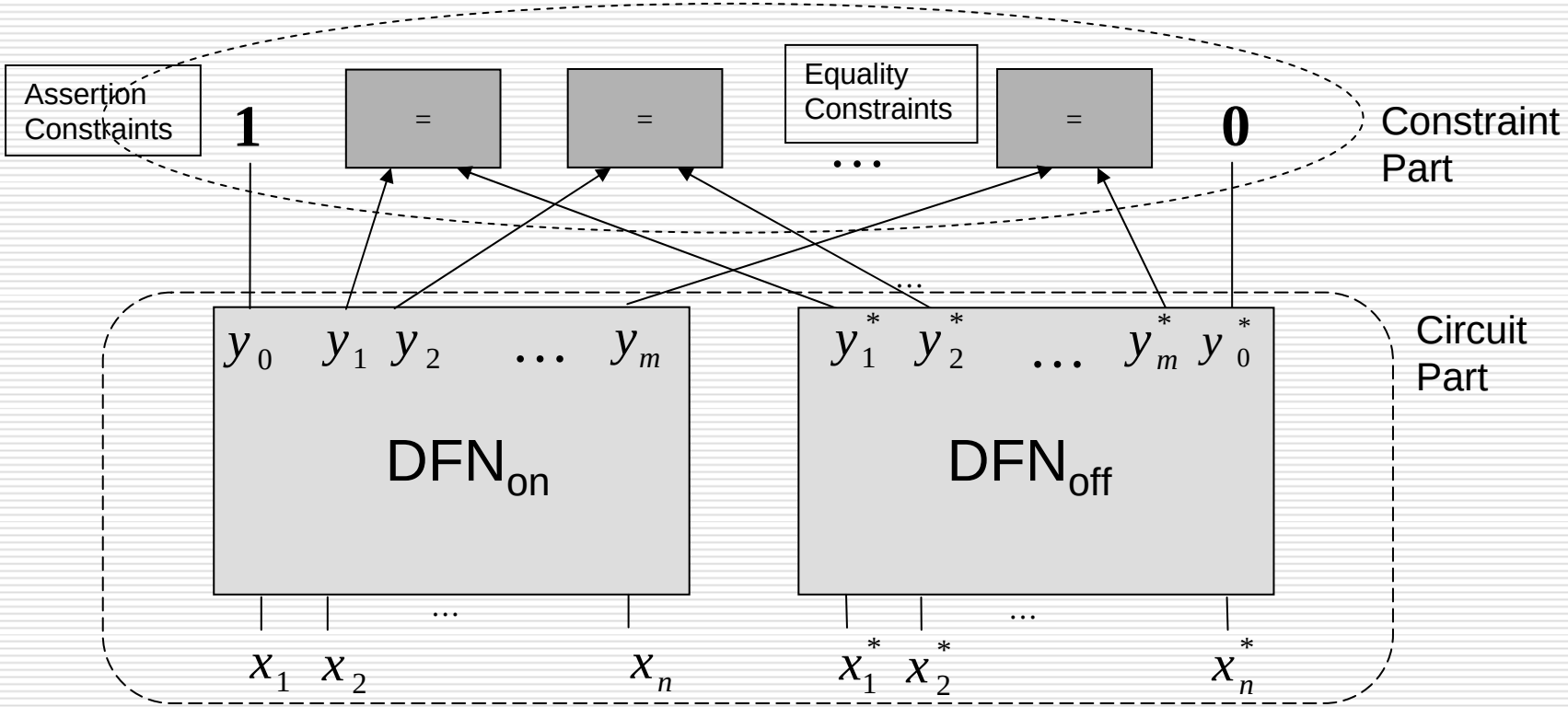
I is an abstraction of A

- Now what ?

  Connecting I with h

The Magic

- $(f(x) \neq f(x^*)) \wedge (G(x) \equiv G(x^*))$ is UNSAT

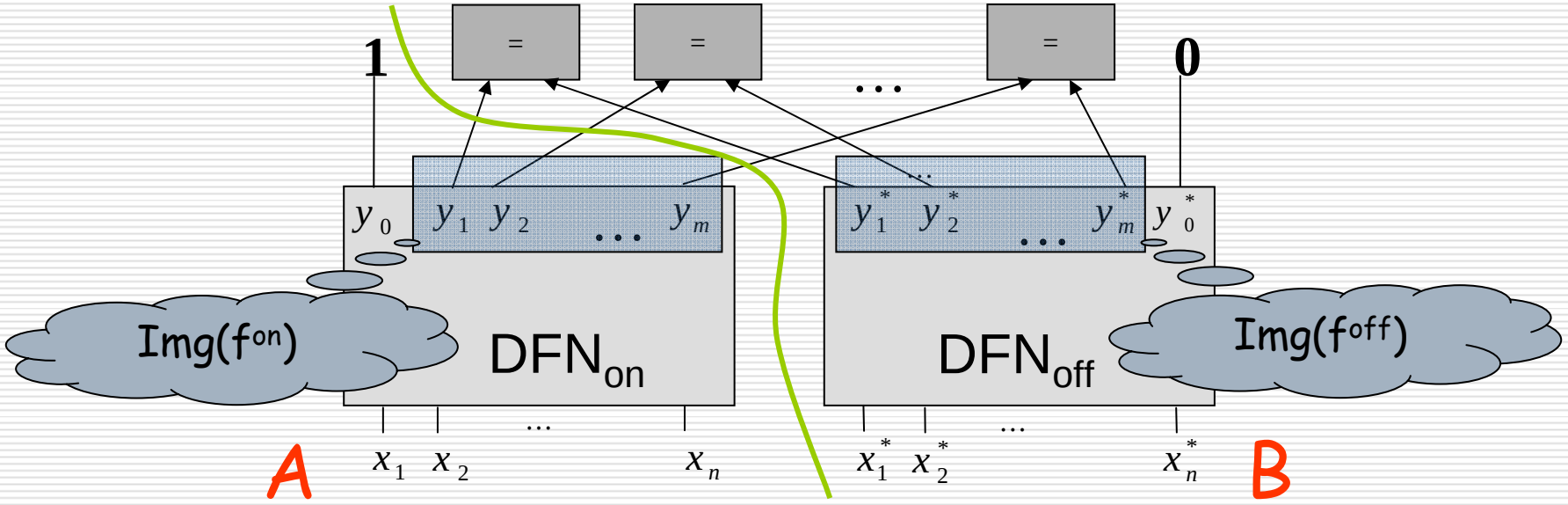


Premises

- Interpolant is of size linear to the refutation size
  - Resolution refutations of UNSAT CNF formulas can be generated from DPLL-style SAT solvers, and so can interpolants
- Circuit-to-CNF translation is doable in linear time
  - Intermediate variables are introduced

It Works!

- Clause set A: C_{DFNon}, y_0
- Clause set B: C_{DFNoff}, $\neg y_0^*$, $(y_i \equiv y_i^*)$ for $i = 1,\ldots,m$
- I is an overapproximation of $Img(f^{on})$ and is disjoint from $Img(f^{off})$
- I only refers to $y_1,\ldots, y_m$
- Therefore, I corresponds to a feasible implementation of h



Incremental SAT Solving

- Controlled equality constraints

  $(y_i \equiv y_i^*) \rightarrow (\neg y_i \vee y_i^* \vee \alpha_i)(y_i \vee \neg y_i^* \vee \alpha_i)$

  with auxiliary variables α_i

  α_i = true $\Rightarrow$ i^{th} equality constraint is disabled

  - Fast switch between target and base functions by unit assumptions over control variables
  - Fast enumeration of different base functions
  - Share learned clauses

SAT vs. BDD

- SAT
  - Pros
    - Detect multiple choices of G automatically
    - Scalable to large $|G|$
    - Fast enumeration of different target functions f
    - Fast enumeration of different base functions G
  - Cons
    - Single feasible implementation of h
- BDD
  - Cons
    - Detect one choice of G at a time
    - Limited to small $|G|$
    - Slow enumeration of different target functions f
    - Slow enumeration of different base functions G
  - Pros
    - All possible implementations of h

Experimental Setup

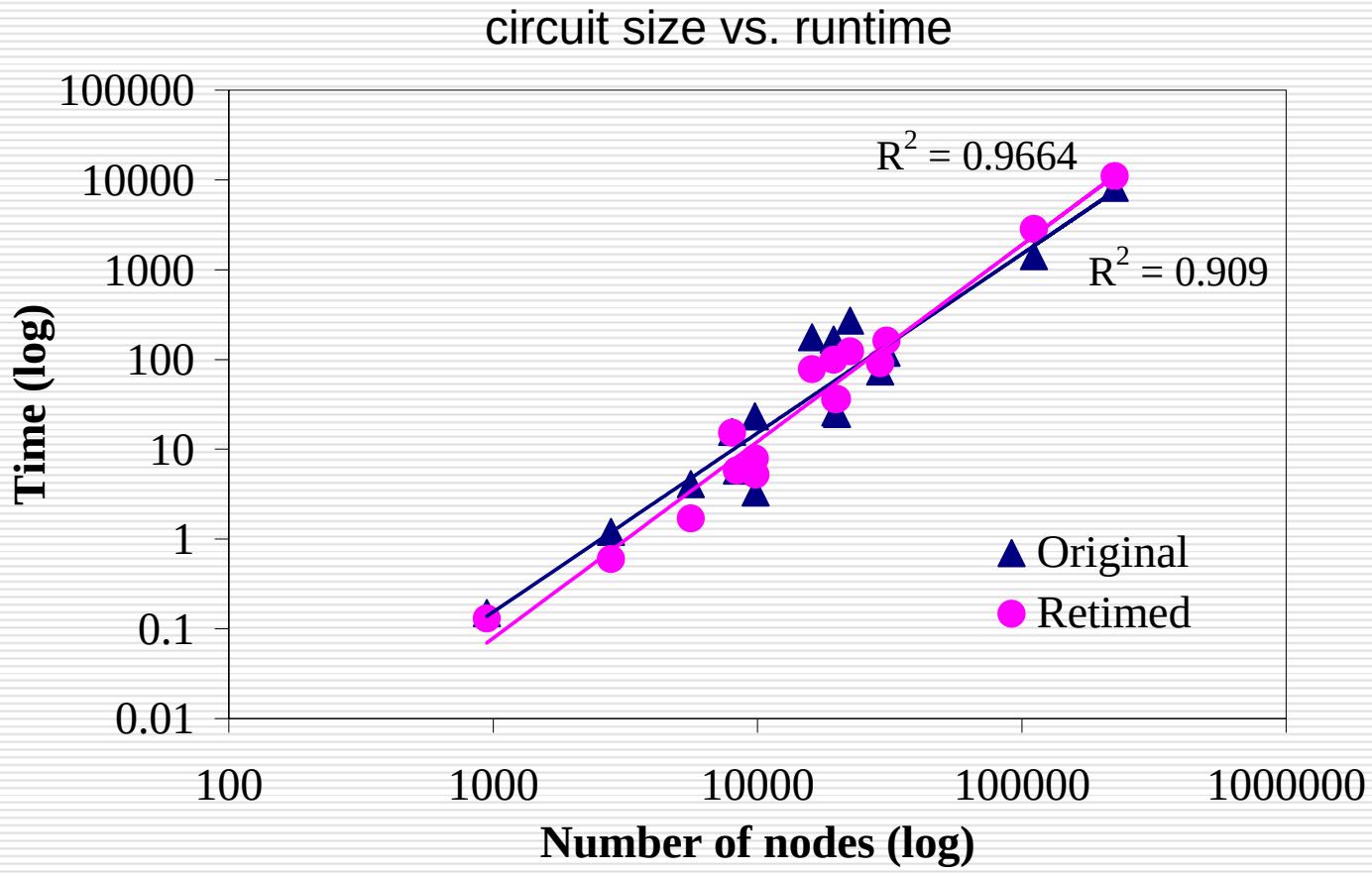
- Implemented in ABC using MiniSAT
- Detect functional dependency among transition functions of sequential circuits

Experimental Results

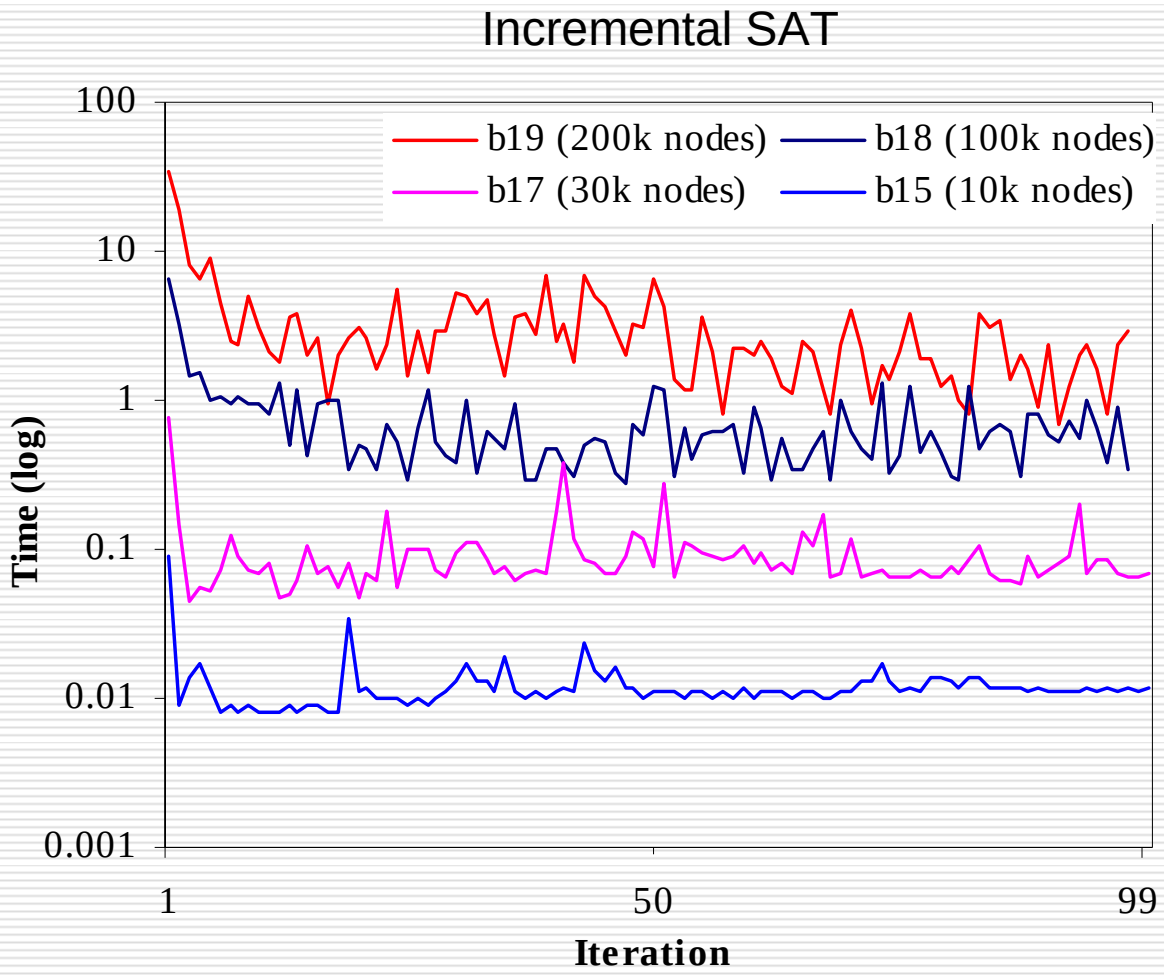
SAT vs. BDD

		Original			Retimed			SAT (original)		BDD (original)		SAT (retimed)		BDD (retimed)	
Circuit	#Nodes	#FF.	#Dep-S	#Dep-B	#FF.	#Dep-S	#Dep-B	Time	Mem	Time	Mem	Time	Mem	Time	Mem
s5378	2794	179	52	25	398	283	173	1.2	18	1.6	20	0.6	18	7	51
s9234.1	5597	211	46	x	459	301	201	4.1	19	x	x	1.7	19	194.6	149
s13207.1	8022	638	190	136	1930	802	x	15.6	22	31.4	78	15.3	22	x	x
s15850.1	9785	534	18	9	907	402	x	23.3	22	82.6	94	7.9	22	x	x
s35932	16065	1728	0	--	2026	1170	--	176.7	27	1117	164	78.1	27	--	--
s38417	22397	1636	95	--	5016	243	--	270.3	30	--	--	123.1	32	--	--
s38584	19407	1452	24	--	4350	2569	--	166.5	21	--	--	99.4	30	1117	164
b12	946	121	4	2	170	66	33	0.15	17	12.8	38	0.13	17	2.5	42
b14	9847	245	2	--	245	2	--	3.3	22	--	--	5.2	22	--	--
b15	8367	449	0	--	1134	793	--	5.8	22	--	--	5.8	22	--	--
b17	30777	1415	0	--	3967	2350	--	119.1	28	--	--	161.7	42	--	--
b18	111241	3320	5	--	9254	5723	--	1414	100	--	--	2842.6	100	--	--
b19	224624	6642	0	--	7164	337	--	8184.8	217	--	--	11040.6	234	--	--
b20	19682	490	4	--	1604	1167	--	25.7	28	--	--	36	30	--	--
b21	20027	490	4	--	1950	1434	--	24.6	29	--	--	36.3	31	--	--
b22	29162	735	6	--	3013	2217	--	73.4	36	--	--	90.6	37	--	--

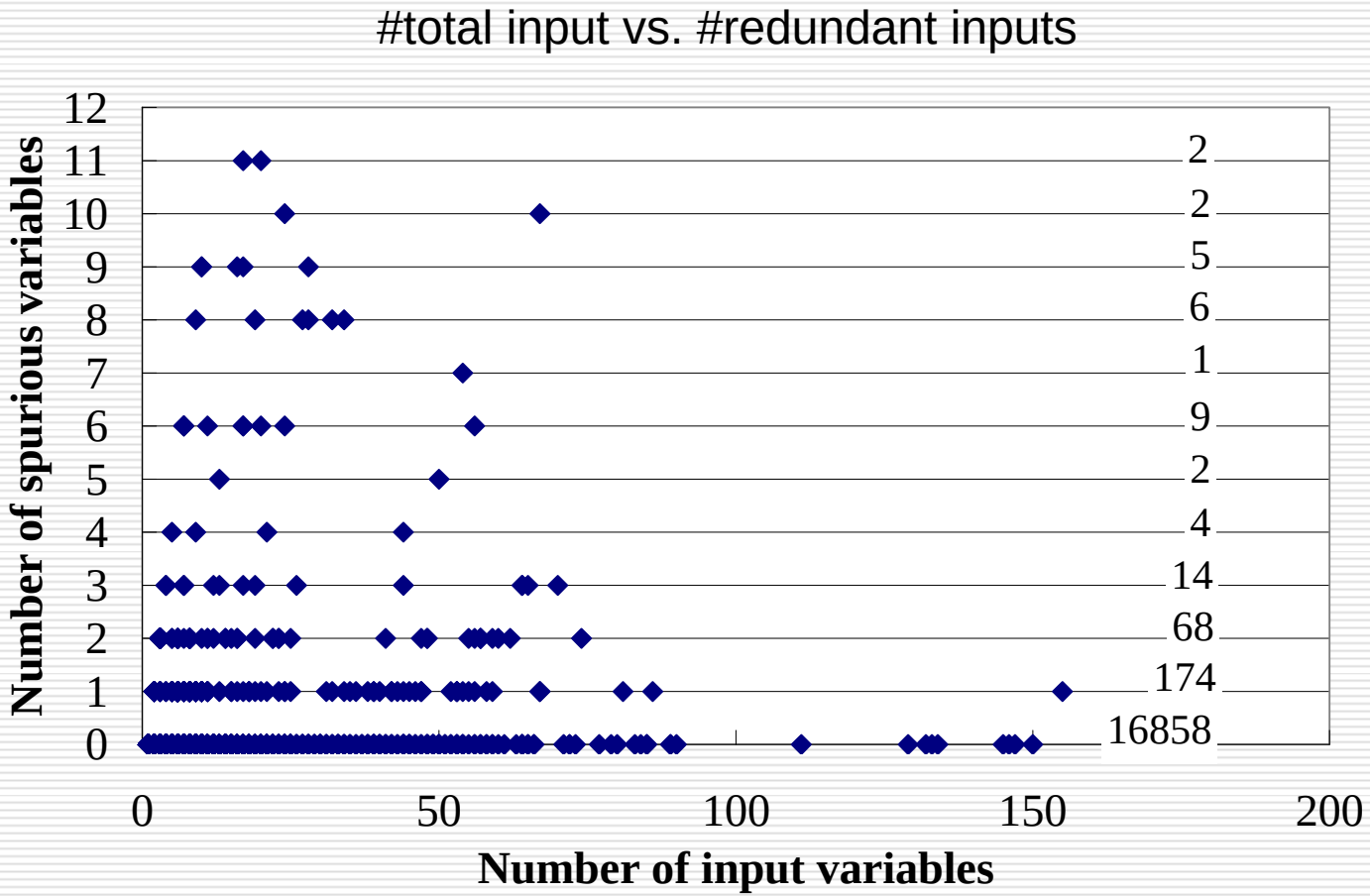
Experimental Results (cont'd)



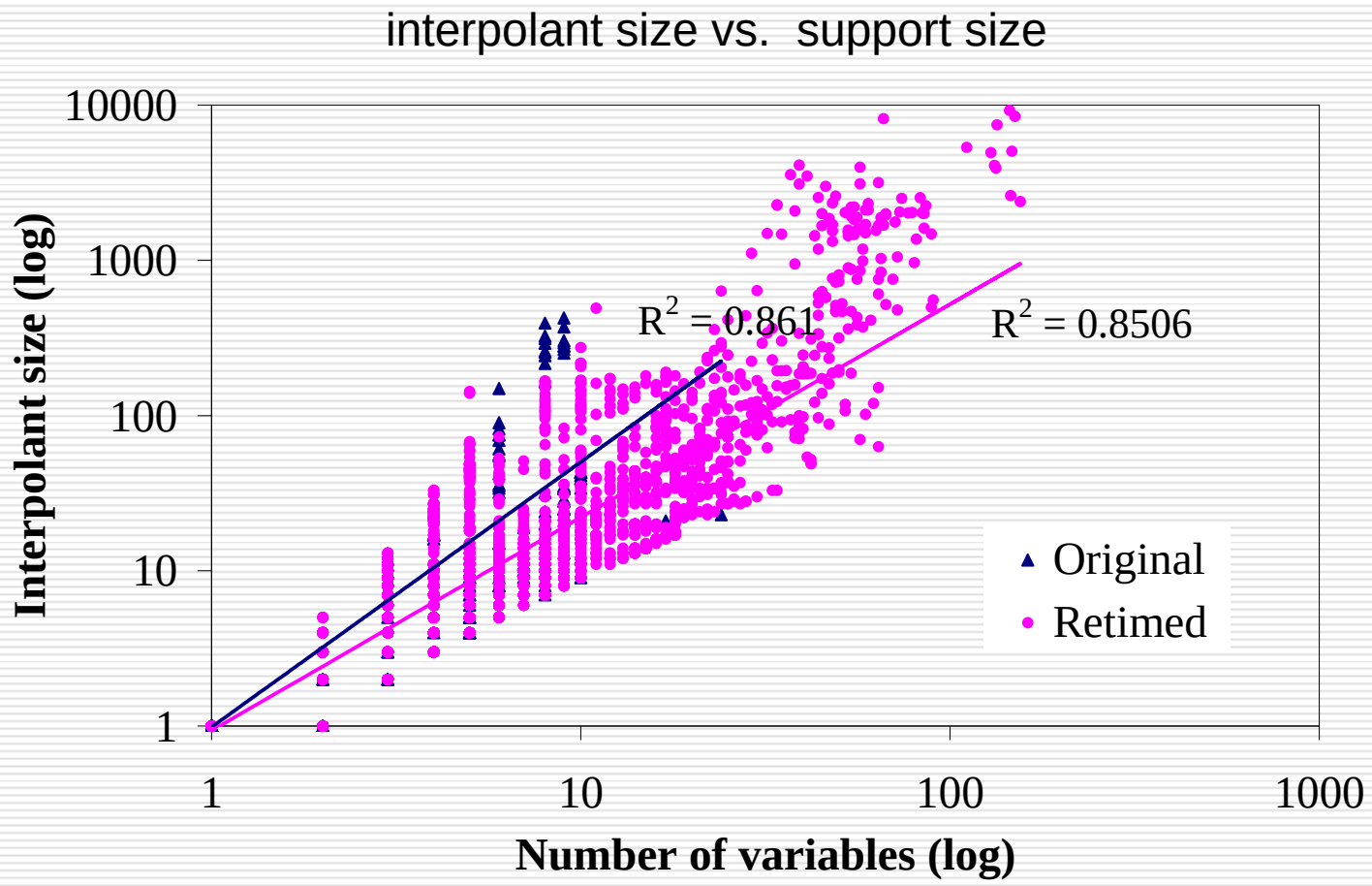
Experimental Results (cont'd)



Experimental Results (cont'd)



Experimental Results (cont'd)



Conclusions and Future Work

- Have shown
  - Functional dependency is computable with pure SAT-solving and is scalable to large designs
- To try
  - Don’t-care generation for h
  - Other applications of Craig interpolation
  - Hybrid approach combining BDD and SAT

Thank You