Craig Interpolation in Logic Synthesis

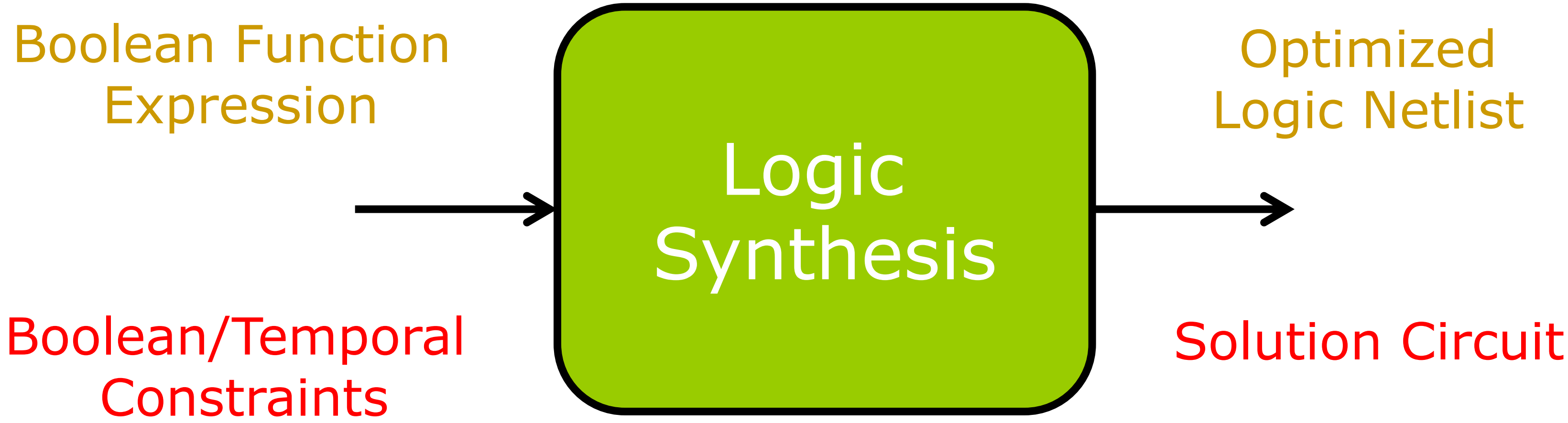
Jie-Hong Roland Jiang

Department of Electrical Engineering
Graduate Institute of Electronics Engineering
National Taiwan University

Outline

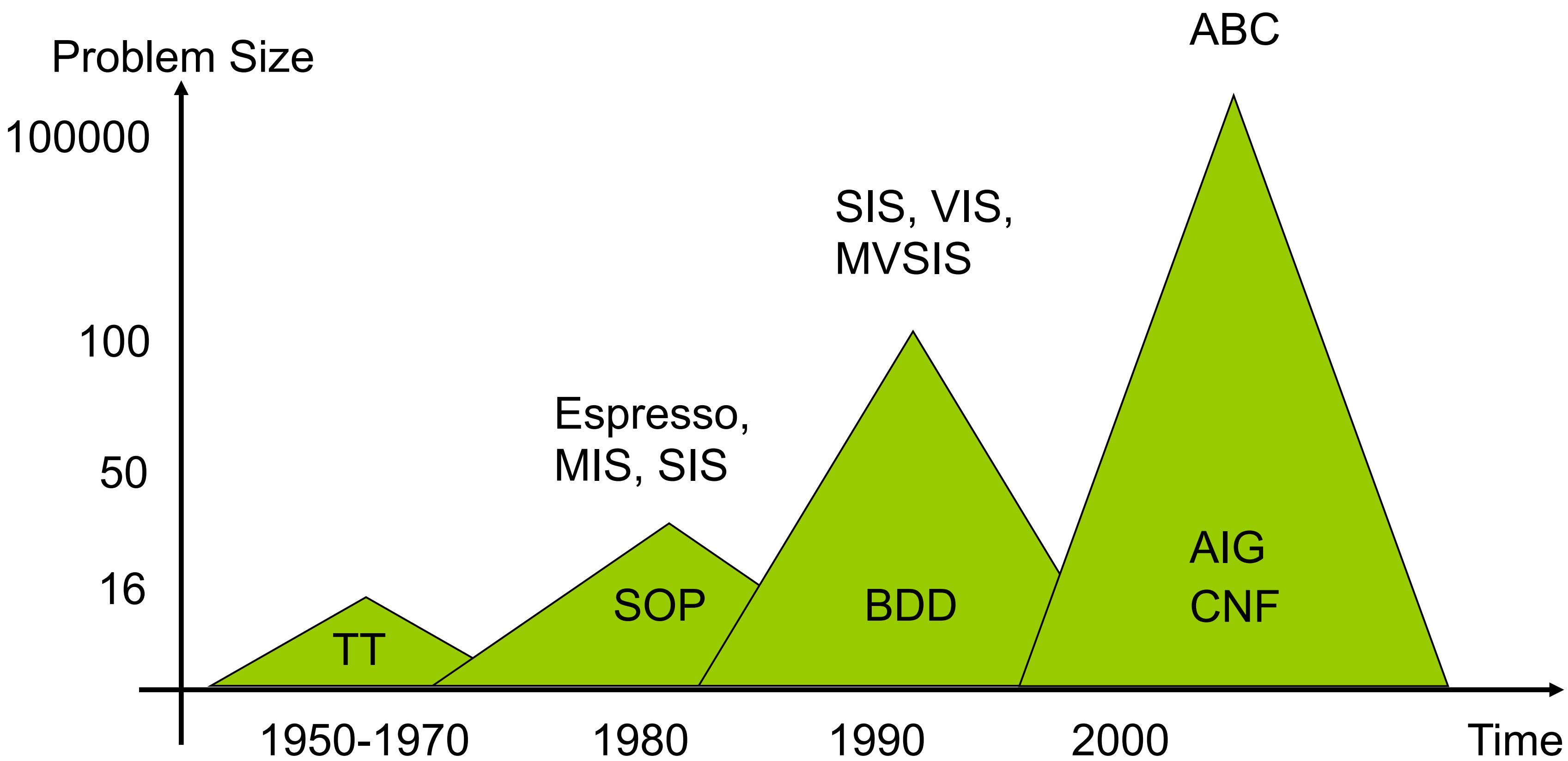
- Introduction
- Craig interpolation and Boolean (un)satisfiability
- Craig interpolation in logic synthesis
- Extensions and challenges

Logic Synthesis



Backgrounds

- Historic evolution of data structures and tools in logic synthesis and verification



Courtesy of Alan Mishchenko

Boolean Function Representation

- Logic synthesis translates Boolean functions into circuits

- We need representations of Boolean functions for two reasons:
  - to represent and manipulate the actual circuit that we are implementing
  - to facilitate *Boolean reasoning*

Boolean Function

- A Boolean function f over input variables: $x_1, x_2, \ldots, x_m$, is a mapping f: $\mathbf{B}^m \rightarrow Y$, where $\mathbf{B} = \{0,1\}$ and $Y = \{0,1,d\}$
  - E.g.
  - The output value of $f(x_1, x_2, x_3)$, say, partitions $\mathbf{B}^m$ into three sets:
    - on-set (f =1)
      - E.g. {010, 011, 110, 111} (characteristic function $f^1 = x_2$)
    - off-set (f = 0)
      - E.g. {100, 101} (characteristic function $f^0 = x_1 \neg x_2$)
    - don't-care set (f = d)
      - E.g. {000, 001} (characteristic function $f^d = \neg x_1 \neg x_2$)

- f is an incompletely specified function if the don't-care set is nonempty. Otherwise, f is a completely specified function
  - Unless otherwise said, a Boolean function is meant to be completely specified

Boolean Function Representation & Conversion

- Truth table
  - Canonical
  - Useful in representing small functions
- SOP / DNF
  - Useful in two-level logic optimization, and in representing local node functions in a Boolean network
- POS / CNF
  - Useful in SAT solving and Boolean reasoning
- ROBDD
  - Canonical
  - Useful in Boolean reasoning
- Boolean network
  - Useful in multi-level logic optimization
- AIG
  - Useful in multi-level logic optimization and Boolean reasoning

Craig Interpolation & Boolean (Un)Satisfiability

Satisfiability

- The **satisfiability** (SAT) problem asks whether a given CNF formula can be true under some assignment to the variables

- In theory, SAT is intractable
  - The first shown NP-complete problem [Cook, 1971]

- In practice, modern SAT solvers work 'mysteriously' well on application CNFs with ~100,000 variables and ~1,000,000 clauses
  - It enables various applications, and inspires QBF and SMT (Satisfiability Modulo Theories) solver development

SAT Solving

- Ingredients of modern SAT solvers:
  - DPLL-style search
    - [Davis, Putnam, Logemann, Loveland, 1962]
  - Conflict-driven clause learning (CDCL)
    - [Marques-Silva, Sakallah, 1996 (GRASP)]
  - Boolean constraint propagation (BCP) with two-literal watch
    - [Moskewicz, Modigan, Zhao, Zhang, Malik, 2001 (Chaff)]
  - Decision heuristics using variable activity
    - [Moskewicz, Modigan, Zhao, Zhang, Malik, 2001 (Chaff)]
  - Restart
  - Preprocessing
  - Support for incremental solving
    - [Een, Sorensson, 2003 (MiniSat)]

Modern SAT Procedure

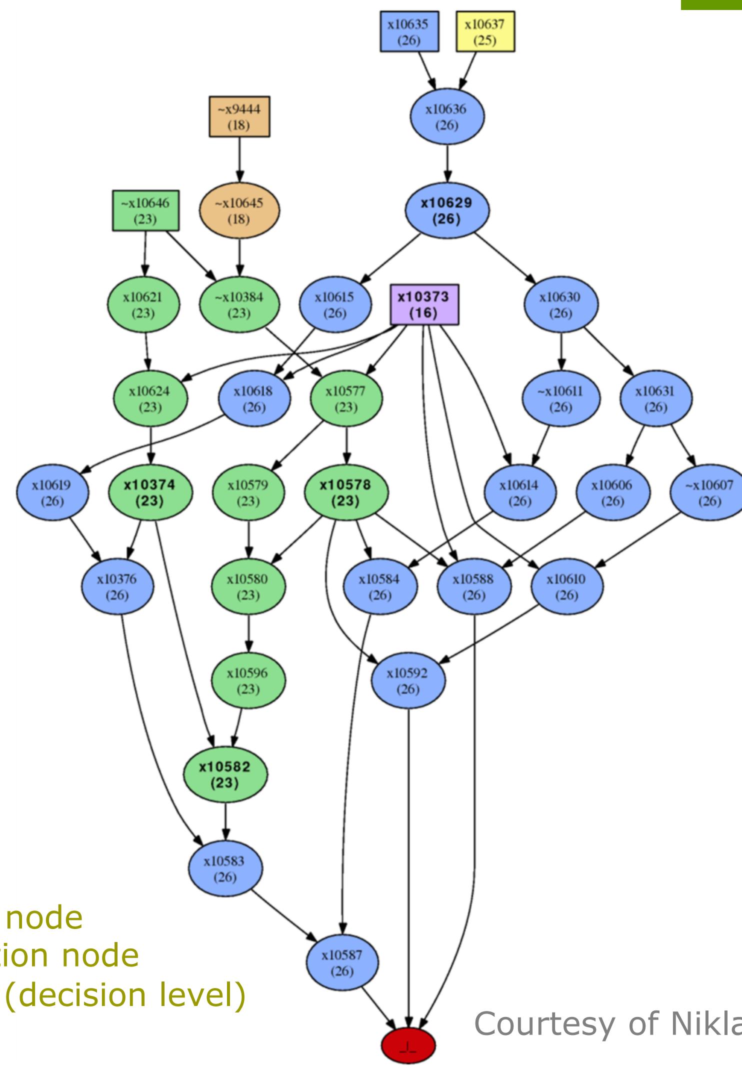
Algorithm `CDCL(Φ)`

```
{
  while(1)
     while there is a unit clause {l} in Φ
         Φ = BCP(Φ, l);
     while there is a pure literal l in Φ
         Φ = assign(Φ, l);
     if Φ contains no conflicting clause
        if all clauses of Φ are satisfied   return true;
        l := choose_literal(Φ);
        assign(Φ,l);
     else
        if conflict at top decision level   return false;
        analyze_conflict();
        undo assignments;
        Φ := add_conflict_clause(Φ);
}
```

Conflict Analysis & Clause Learning

- There can be many learnt clauses from a conflict
- Clause learning admits non-chorological backtrack

- E.g.,
  {¬x10587, ¬x10588, ¬x10592}
  …
  {¬x10374, ¬x10582, ¬x10578, ¬x10373, ¬x10629}
  …
  {x10646, x9444, ¬x10373, ¬x10635, ¬x10637}



Box: decision node
Oval: implication node
Inside: literal (decision level)

Courtesy of Niklas Een

Clause Learning as Resolution

- **Resolution** of two clauses $C_1 \vee x$ and $C_2 \vee \neg x$:

$$\frac{C_1 \vee x \qquad\qquad C_2 \vee \neg x}{C_1 \vee C_2}$$

  where x is the **pivot variable** and $C_1 \vee C_2$ is the **resolvant**, i.e., $C_1 \vee C_2 = \exists x.(C_1 \vee x)(C_2 \vee \neg x)$

- A learnt clause can be obtained from a sequence of resolution steps
  - E.g.,
    Can find a resolution sequence leading to the learnt clause
    $\{\neg x10374, \neg x10582, \neg x10578, \neg x10373, \neg x10629\}$
    in the previous slides

Resolution

- Resolution is complete for SAT solving
  - A CNF formula is unsatisfiable if and only if there exists a resolution sequence leading to the empty clause
  - Example

$(a \vee b \vee c)(\neg a \vee c)(\neg b \vee \neg d)(\neg c)(c \vee d)$

$(b \vee c)$ (d)

$(c \vee \neg d)$

$(\neg d)$

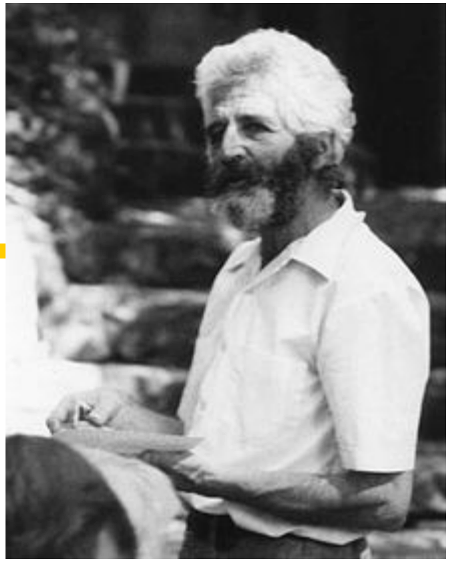
$()$

SAT Certification

- True CNF
  - Satisfying assignment (model)
    - Verifiable in linear time

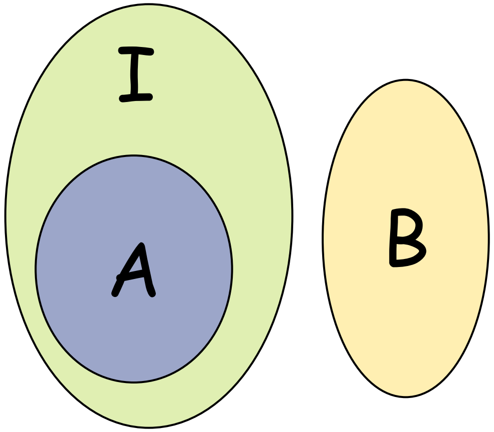
- False CNF
  - Resolution refutation
    - Potential exponential size

Craig's Interpolation



- [Interpolation in propositional logic]
  If $A \wedge B$ is UNSAT for formulae A and B, 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

W. Craig (1957a), Linear reasoning. A new form of the Herbrand-Gentzen theorem
W. Craig (1957b), Three uses of the HerbrandGentzen theorem in relating model theory and proof theory

Interpolant Computation

- Pudlak's method [Pudlak 1997]
  - Resolution to mux network
- McMillan's method [McMillan 2003]
  - Resolution to AND/OR circuit
- Iterative blocking [Chockler et al. 2012]
  1. Find a satisfying assignment α of A
  2. Generalize α to remove literals while maintaining $(\alpha \wedge B)$ UNSAT
  3. Add α to $I := (I \vee \alpha)$
  4. Update $A := (A \wedge \neg\alpha)$
  5. Repeat until A is UNSAT

Interpolant and Resolution Proof

- SAT solver may produce the resolution proof of an UNSAT CNF φ
- For $\varphi = \varphi_A \wedge \varphi_B$ specified, the corresponding interpolant can be obtained in time linear in the resolution proof

$$\overbrace{(a\vee b\vee c)(\neg a\vee c)}^{\varphi_A}\overbrace{(\neg b\vee\neg d)(\neg c)(c\vee d)}^{\varphi_B}$$

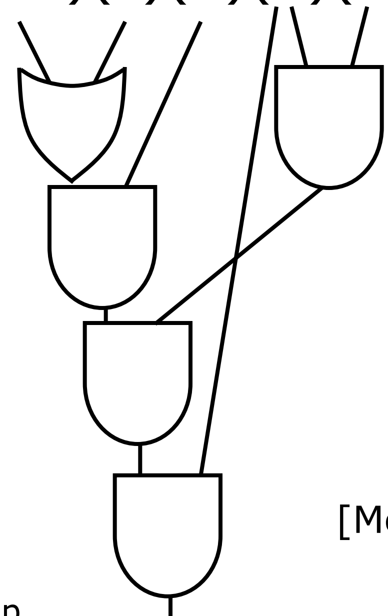
$(b\vee c)$ $\quad$ (d)

$(c\vee\neg d)$

$(\neg d)$

$()$

$(b\vee c)(c)(1)(1)(1)$



$= (b\vee c)$

[McMillan, 2003]

Craig Interpolation in Logic Synthesis

Craig Interpolation in Logic Synthesis

- **Functional dependency [Lee et al. 2007]**
- Bi-decomposition [Lee et al. 2008]
- Ashenhurst decomposition [Lin et al. 2008]
- Relation determinization [J et al. 2009]
- Quantifier elimination [J 2009]
- Engineering change order [Wu et al. 2010, Ling et al. 2011]
- Decoder synthesis [Liu et al. 2011, Tu, J 2013]
- Don’t care based optimization [Mishchenko et al. 2011]
- Cyclic dependency [Backes and Riedel 2012]
- Iterative layering [Petkovska et al. 2013]
- …

Disclaimer: The above list is not meant to be complete.

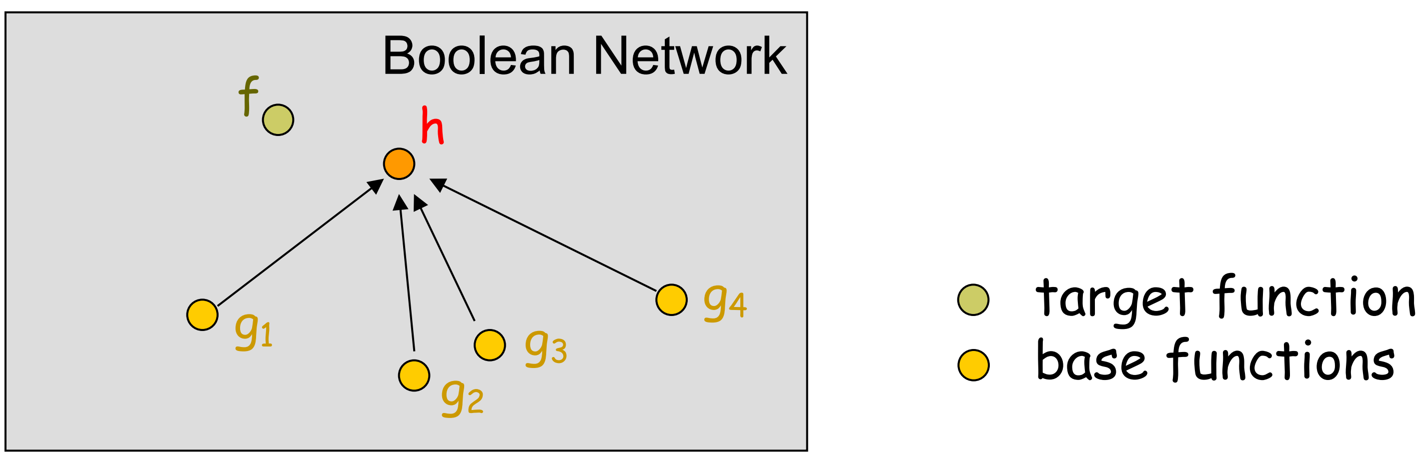
Functional Dependency

- $f(x)$ **functionally depends** on $g_1(x)$, $g_2(x)$, …, $g_m(x)$ if $f(x) = h(g_1(x), g_2(x), \ldots, g_m(x))$, denoted $h(G(x))$
  - Under what condition can function f be expressed as some function h over a set $G=\{g_1,\ldots,g_m\}$ of functions ?
  - 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

Motivation

- Applications of functional dependency
  - Resynthesis/rewiring
  - Redundant register removal
  - BDD minimization
  - Verification reduction
  - …

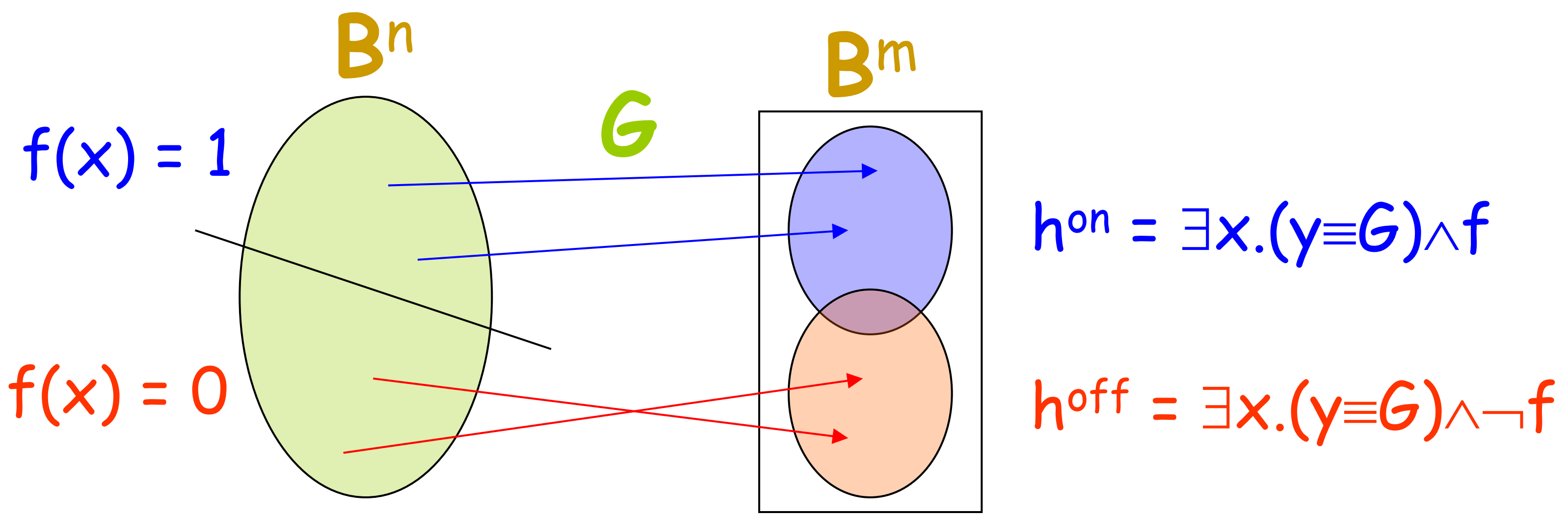


BDD-Based Computation

- 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

- Pros
  - Exact computation of h^{on} and h^{off}
  - Better support for don't care minimization

- Cons
  - 2 image computations for every choice of G
  - Inefficient when $|G|$ is large or when there are many choices of G

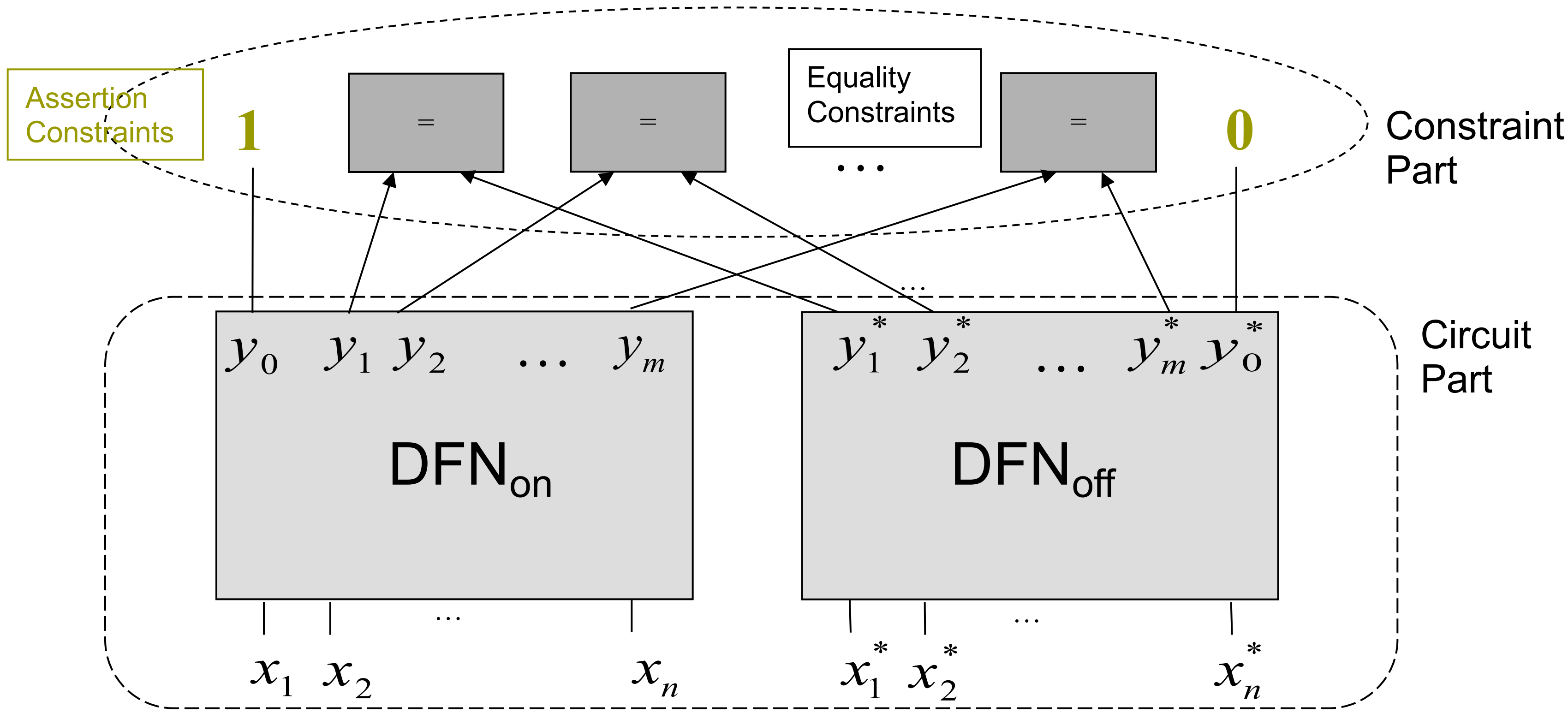
SAT-Based Computation

- h exists $\Leftrightarrow$
  $\nexists a,b$ such that $f(a) \neq f(b)$ and $G(a)=G(b)$,
  i.e., $(f(x) \neq f(x^*)) \wedge (G(x) \equiv G(x^*))$ is UNSAT

- How to derive h? How to select G?

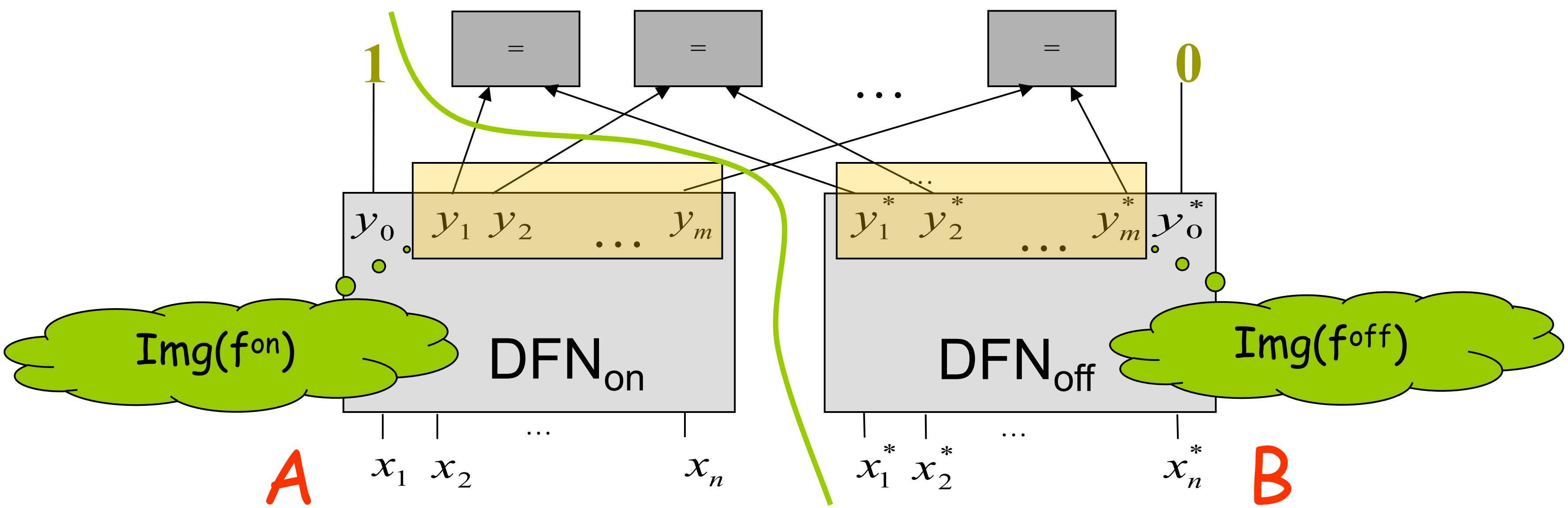
SAT-Based Computation

- $(f(x) \not\equiv f(x^*)) \wedge (G(x) \equiv G(x^*))$ is UNSAT



Deriving *h* with Craig Interpolation

- Clause set A: C_{DFNon}, y_0
- Clause set B: C_{DFNoff}, $\neg y_0^*$, $(y_i \equiv y_i^*)$ for i =1,…,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

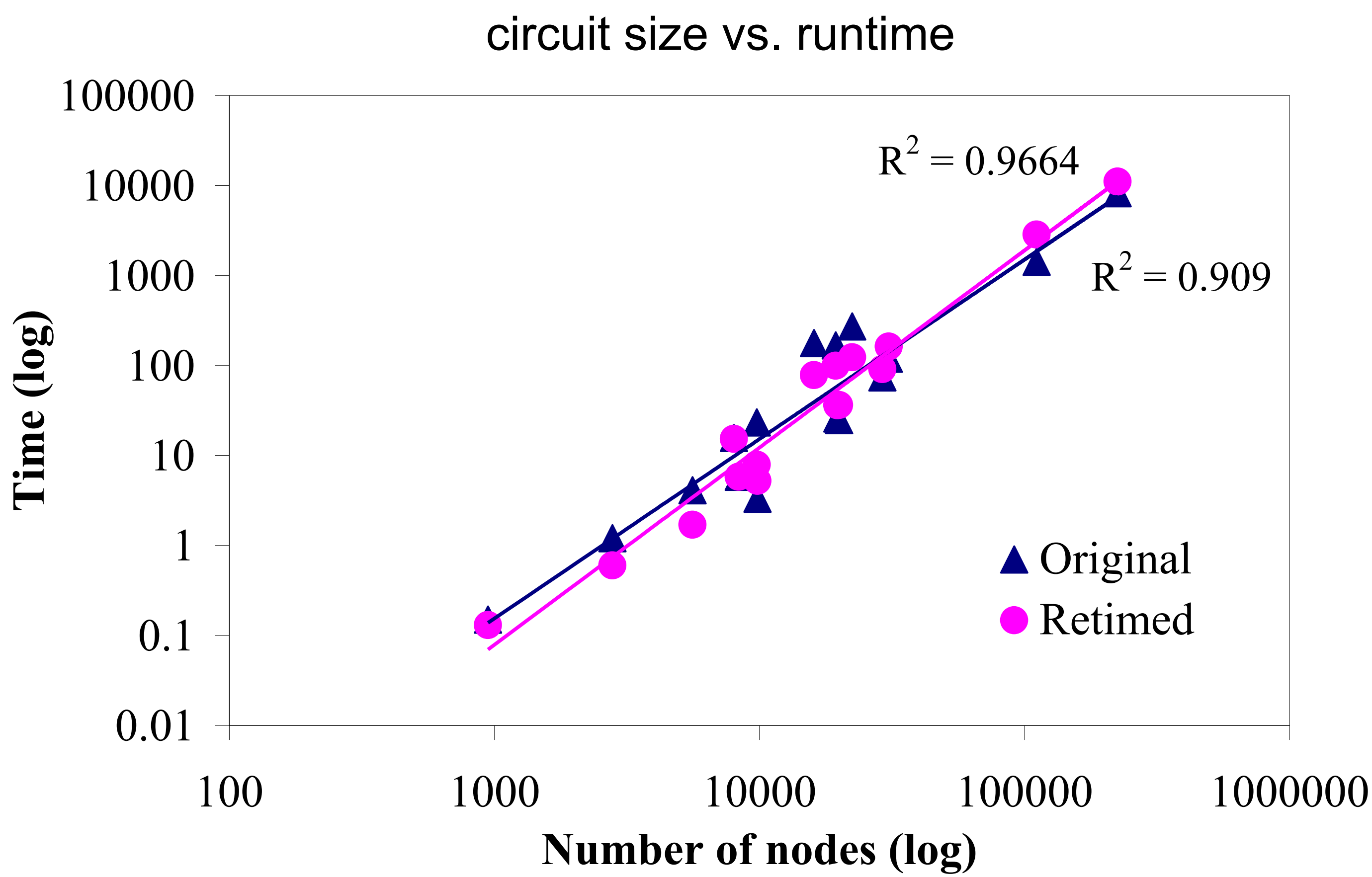
- SAT
  - Pros
    - Detect multiple choices of G automatically
    - Scalable to large $|G|$
    - Fast enumeration of different target functions f
    - Fast enumeration of different base functions G
  - Cons
    - Single feasible implementation of h
- BDD
  - Cons
    - Detect one choice of G at a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

Practical Evaluation

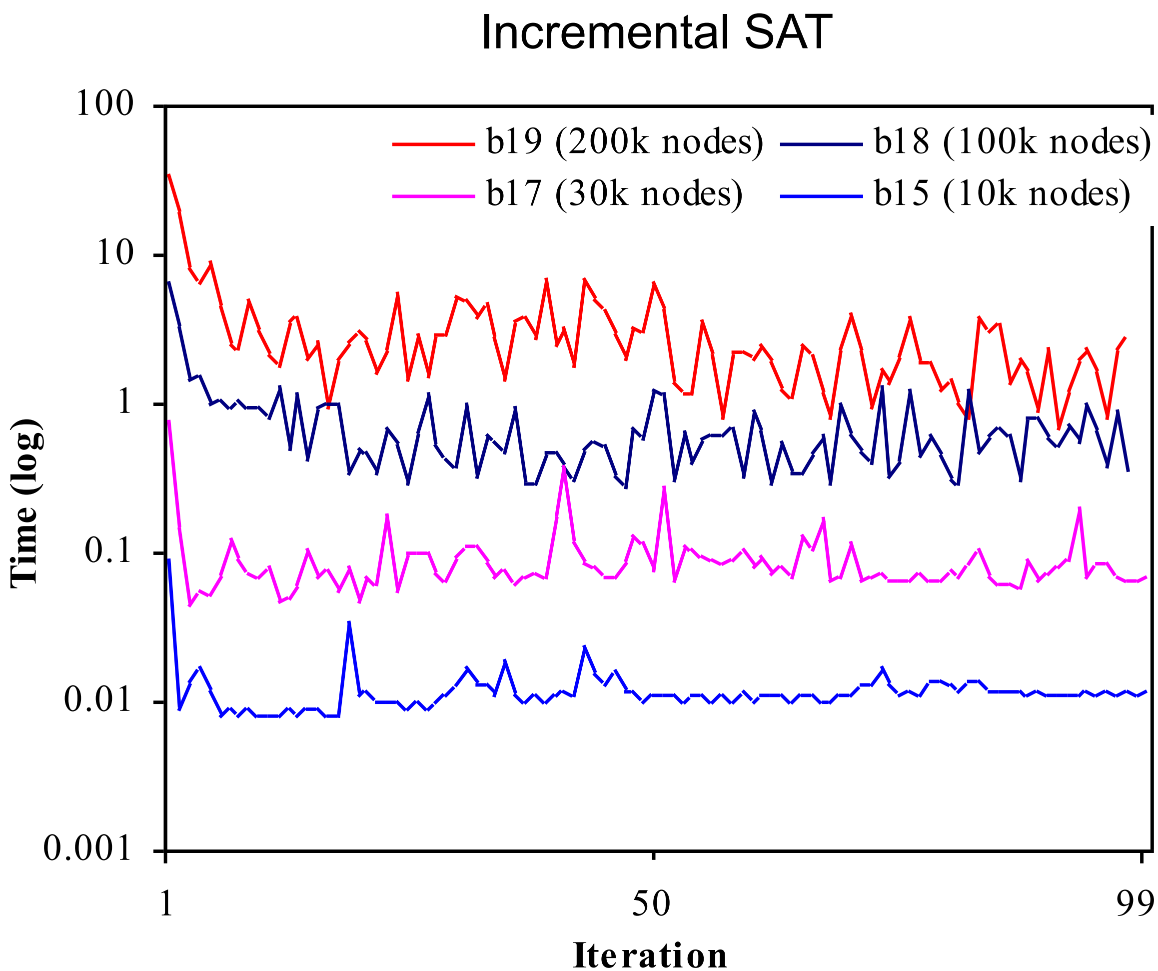
SAT vs. BDD

		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	--	--

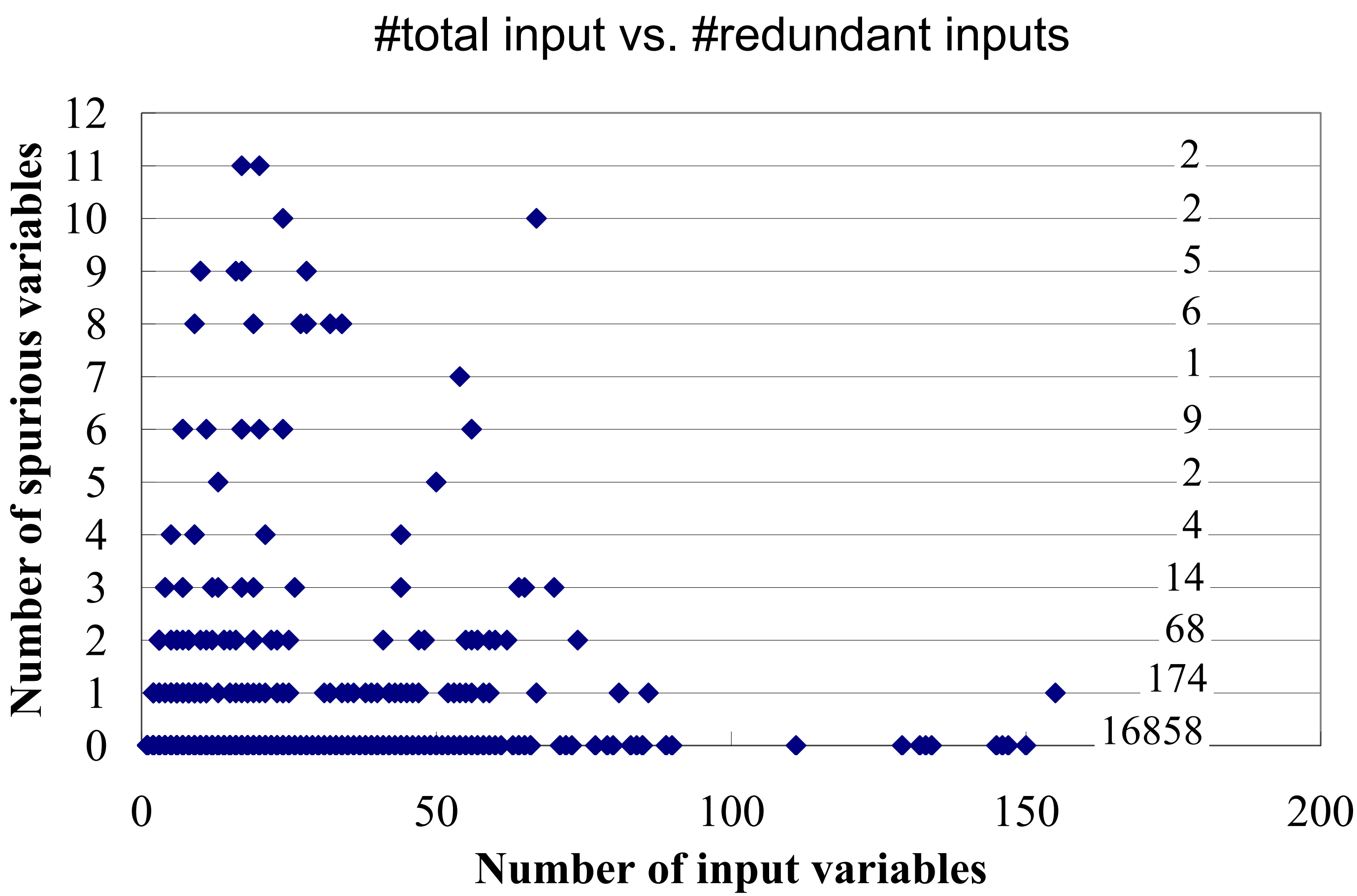
Practical Evaluation



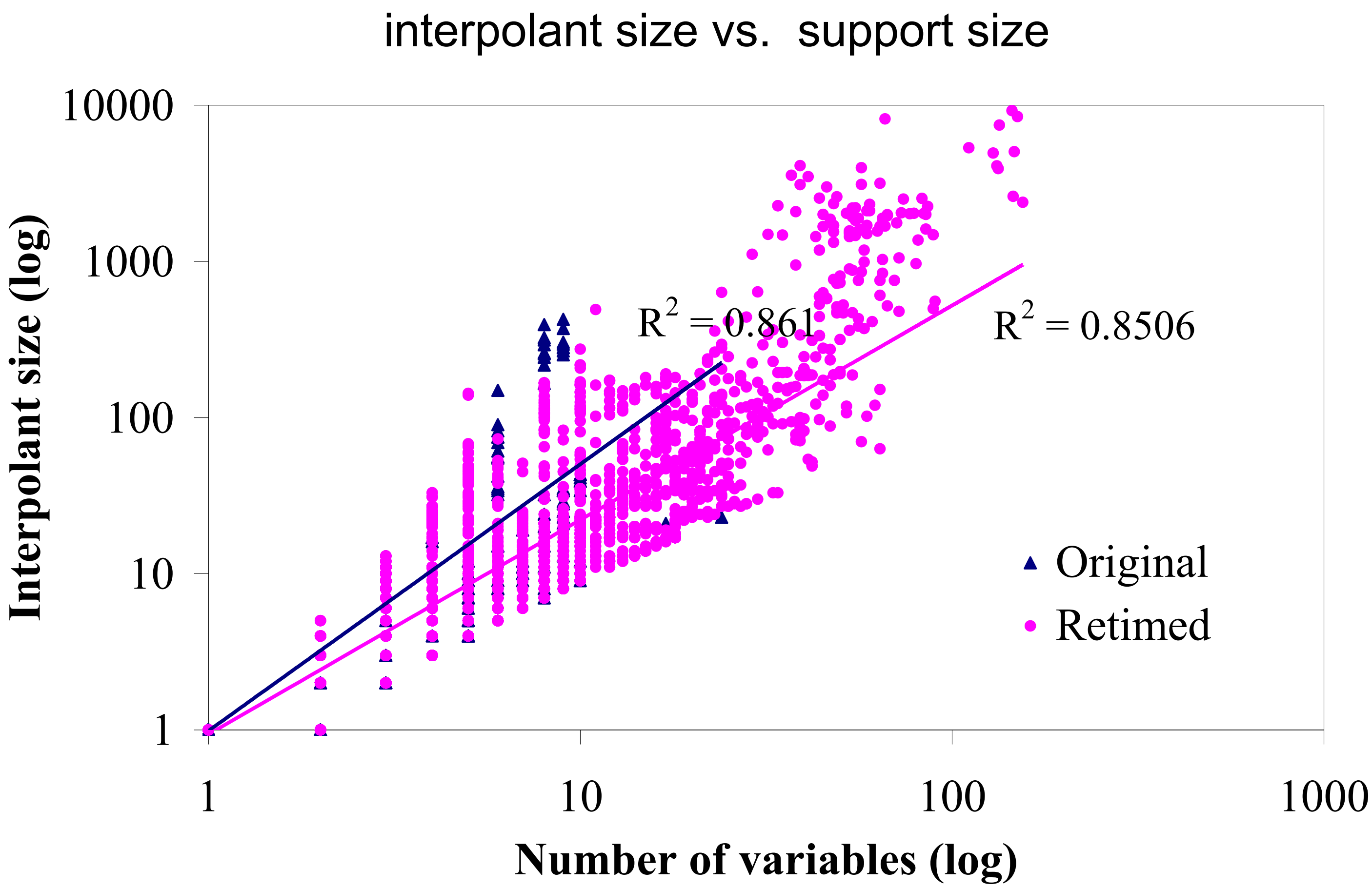
Practical Evaluation



Practical Evaluation



Practical Evaluation



Directions and Challenges

- More applications in logic synthesis
- Compact interpolants wanted!
  - Resolution simplification [Backes, Riedel 2010]
  - Interpolant circuit minimization [Cabodi et al. 2019]
  - Interpolant of XOR constraints [Han, J 2012]
- Interpolation in other domains and reflection in logic synthesis
  - Linear arithmetic
    - [Han, J 2012, Scholl et al. 2014]
  - Satisfiability Modulo Theories (SMT)
    - [McMillan 2005, Cimatti et al. 2010]

Thanks for Your Attention!