

Extended Resolution Proofs for Conjoining BDDs

Carsten Sinz and Armin Biere

Johannes Kepler University Linz, Austria

Propositional Logic

□ SAT-Solvers

- DPLL algorithm with extensions, search/resolution-based, generate refutations for formulae in CNF
- Tremendous progress over the last years:
 - Clause learning, watched literals, adaptive decision heuristics, preprocessing

□ BDDs

- Formalism to uniquely represent Boolean functions
- Still common in hardware verification
- Not used as proof system so far

Why Propositional Logic Proofs?

- SAT-solvers and BDDs commercially employed

- Hardware verification
(Bounded Model Checking)
- Product configuration



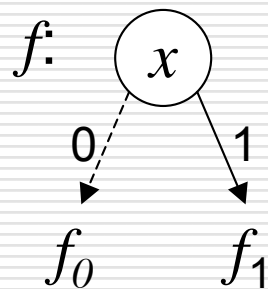
- Yes/No answer of solvers not sufficient
 - Counterexample or proof needed
 - Used for abstraction refinement, interpolant computation, proof checking, diagnosis, ...

State-of-the-Art & Challenge

- **Known:** BDD and search based techniques complement each other
[Uribe&Stickel, 1994; Groote&Zantema, 2003]
- Proof generation for search-based SAT-solvers current research topic
 - Proof systems for clause-learning SAT-Solvers
[Beame et al., 2004]
- Proofs for BDD-based solvers not yet covered by the literature
- **Challenge:** How to generate proofs when using BDDs for SAT-solving?

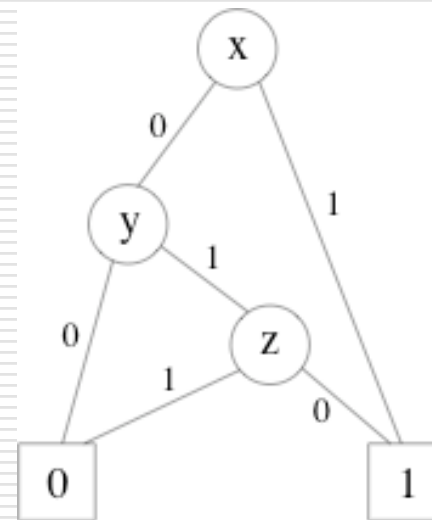
Binary Decision Diagrams (BDDs)

- BDDs uniquely represent Boolean functions



Node interpretation:

$$\begin{aligned} f &= \text{if } x \text{ then } f_1 \text{ else } f_0 \\ &= (x ? f_1 : f_0) \\ &= (x \rightarrow f_1) \wedge (\neg x \rightarrow f_0) \end{aligned}$$



BDD representing $x \vee (y \wedge \neg z)$
using ordering $x > y > z$

Using BDDs for SAT-Solving

- **Given:** $F = C_1 \wedge \dots \wedge C_n$ formula in CNF
- **Method:** Build F incrementally from BDDs for C_i by BDD-and operation:

Algorithm 1: BDD-and(a, b)

- 1: if $a = 0$ or $b = 0$ then return 0
- 2: if $a = 1$ then return b else if $b = 1$ then return a
- 3: $(x, a_0, a_1) = \text{decompose}(a)$; $(y, b_0, b_1) = \text{decompose}(b)$
- 4: if $x < y$ then return new-node($y, \text{BDD-and}(a, b_0), \text{BDD-and}(a, b_1)$)
- 5: if $x = y$ then return new-node($x, \text{BDD-and}(a_0, b_0), \text{BDD-and}(a_1, b_1)$)
- 6: if $x > y$ then return new-node($x, \text{BDD-and}(a_0, b), \text{BDD-and}(a_1, b)$)

- **Fact:** $\text{BDD}(F)=0$ iff. F unsatisfiable
- **Question:** How to build refutation proof for F if $\text{BDD}(F)=0$?
- **Solution:** Use Extended Resolution as proof system

Extended Resolution (ER)

- Resolution calculus: one inference rule

$$\frac{C \cup \{l\} \quad \{\bar{l}\} \cup D}{C \cup D}$$

C, D : clauses
 l : literal occurring positively in C
and negatively in D

- Extended Resolution: adds extension rule

- Introduces new variable and clauses
- „Definitions“

$$\frac{}{\text{CNF}(x \leftrightarrow F)}$$

x : new variable (neither occurring
in F nor in current clause set)
 F : arbitrary formula

- Goal: derive empty clause

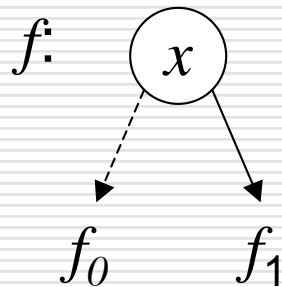
[Tseitin, 1970]

ER Proof Generation Outline

(for unsatisfiable $F = C_1 \wedge \dots \wedge C_n$)

1. Compute BDDs B_i for all clauses C_i .
2. Add definitions $f \leftrightarrow (x ? f_1 : f_0)$ for all BDD nodes occurring in any B_i , with c_i being the variable of the top node of B_i .
3. Produce ER proofs $F \vdash c_i$ for all clauses.
4. Incrementally compute BDDs H_i for conjunctions (top nodes h_i) and add ER definitions for all nodes:
$$H_2 = \text{BDD-and}(B_1, B_2) \quad H_i = \text{BDD-and}(B_i, H_{i-1})$$
5. Produce proofs for $h_2 \leftarrow c_1 \wedge c_2$ and $h_i \leftarrow c_i \wedge h_{i-1}$.

ER Proofs from BDDs (I): Definitions for BDD nodes



$$\begin{aligned}
 f &= \text{if } x \text{ then } f_1 \text{ else } f_0 \\
 &= (x \rightarrow f_1) \wedge (\neg x \rightarrow f_0)
 \end{aligned}$$

□ Use Extension Rule to define node:

CNF($f \leftrightarrow \text{if } x \text{ then } f_1 \text{ else } f_0$)

(introduces one new variable f
for each BDD node)

=

($\neg f \ \neg x \ f_1$)

($\neg f \ x \ f_0$)

($f \ \neg x \ \neg f_1$)

($f \ x \ \neg f_0$)

ER Proofs from BDDs (II): Proofs for Clauses C_i

- **Given:** BDD B_i for clause $C_i = l_1 \vee \dots \vee l_k$ with top node c_i
- **Task:** Generate proof for $F \vdash c_i$
- **Method:**
 - Introduce clauses for each BDD node d_j of B_i
 - c_i can then be derived by a linear, regular resolution proof starting with $(l_1 \dots l_j)$
 - Details: see paper
 - **Complexity:** k applications of the extension rule, $2k$ resolution steps

ER Proofs from BDDs (III): Conjunctions (BDD-and)

- Build proof of $f \wedge g \rightarrow h$ recursively
 - from $f_0 \wedge g_0 \rightarrow h_0$ and $f_1 \wedge g_1 \rightarrow h_1$
 - where $f = \text{if } x \text{ then } f_1 \text{ else } f_0$, $g = \text{if } x \text{ then } g_1 \text{ else } g_0$
and $h = \text{if } x \text{ then } h_1 \text{ else } h_0$

$$\begin{array}{c}
 \begin{array}{c}
 \vdots \\
 \frac{(\bar{f}x f_0) \quad (\bar{f}_0 \bar{g}_0 h_0)}{(\bar{f}x \bar{g}_0 h_0)} \\
 \frac{(\bar{g}x g_0) \quad (\bar{f}x \bar{g}_0 h_0)}{(\bar{f} \bar{g} x h_0)} \\
 \frac{(hx \bar{h}_0) \quad (\bar{f} \bar{g} x h_0)}{(\bar{f} \bar{g} h x)}
 \end{array}
 \quad
 \begin{array}{c}
 \vdots \\
 \frac{(\bar{f}_1 \bar{g}_1 h_1) \quad (\bar{f} \bar{x} f_1)}{(\bar{f} \bar{x} \bar{g}_1 h_1)} \\
 \frac{(\bar{f} \bar{x} \bar{g}_1 h_1) \quad (\bar{g} \bar{x} g_1)}{(\bar{f} \bar{g} \bar{x} h_1)} \\
 \frac{(\bar{f} \bar{g} \bar{x} h_1) \quad (h \bar{x} \bar{h}_1)}{(\bar{f} \bar{g} h \bar{x})}
 \end{array}
 \\
 \hline
 (\bar{f} \bar{g} h)
 \end{array}$$

- **Complexity:** Requires 7 resolutions for each recursive step.

ER Proofs from BDDs (III): Conjunctions (BDD-and)

- Size of resulting proof linear in number of recursive BDD-and calls (*i.e. linear in the runtime of the BDD-algorithm*)
- Proof may contain trivial clauses
 - Can easily be eliminated, see paper
- Missing variables in intermediate BDDs can also be handled easily
 - Simplified recursive proof pattern

Implementation: EBDDRES

- Performs BDD computations
- Generates extended resolution proofs fully automatically
- Good performance on some SAT instances that are hard for DPLL/resolution-based provers (e.g. [pigeon hole](#))
- Proof-checker for resolution-based solvers can easily be adapted for ER proofs
 - Only non-cyclicity test for extension rule applications has to be added

EBDDRES: Results

	MINISAT			EBDDRES											trace chk sec
	solve resources sec	trace size MB	trace size MB	solve resources sec	trace gen. sec	trace size ASCII MB	trace size binary MB	bdd nodes $\times 10^3$	recursive all $\times 10^3$	bdd triv. $\times 10^3$	and- lines $\times 10^3$	steps red. $\times 10^3$	core $\times 10^3$		
ph7	0	0	0	0	0	0	1	0	3	20	10	10	0	10	0
ph8	0	4	1	0	3	0	3	1	15	67	34	33	0	33	0
ph9	6	4	11	0	3	0	3	1	8	90	45	45	0	45	0
ph10	44	4	63	1	17	1	30	10	136	538	270	269	1	268	2
ph11	887	6	929	1	13	1	21	8	35	670	335	334	1	333	2
ph12	*	-	-	2	28	1	33	12	31	1150	575	574	1	573	3
ph13	*	-	-	10	102	8	260	92	850	5230	2615	2614	2	2612	20
ph14	*	-	-	10	111	7	204	74	166	6554	3278	3276	2	3274	18
mutcb8	0	0	0	0	4	0	4	1	23	73	37	37	0	36	0
mutcb9	0	4	0	0	5	0	12	4	64	193	97	96	0	96	1
mutcb10	0	4	1	1	17	1	35	12	177	577	289	288	1	287	3
mutcb11	1	4	4	3	32	2	89	29	419	1380	691	690	3	686	6
mutcb12	8	4	22	6	62	5	188	64	906	2743	1372	1371	3	1368	13
mutcb13	113	5	244	15	146	12	452	155	2040	6398	3199	3198	8	3190	30
mutcb14	491	8	972	50	578	38	1465	*	6225	20520	10261	10260	20	10240	*
mutcb15	*	-	-	-	*	-	-	-	-	-	-	-	-	-	-
mutcb16	*	-	-	-	*	-	-	-	-	-	-	-	-	-	-
urq35	96	4	218	2	28	1	37	13	24	1216	608	608	0	608	3
urq45	*	-	-	-	*	-	-	-	-	-	-	-	-	-	-
fpga108	0	0		6	47	4	135	47	186	4087	2044	2043	3	2040	11
fpga109	0	0		3	44	2	70	24	83	2218	1109	1109	1	1108	6
fpga1211	0	0		54	874	38	1214	*	1312	33783	16892	16891	41	16850	*
add16	0	0	0	0	4	0	6	2	30	100	51	50	1	49	0
add32	0	0	0	1	9	1	24	8	122	445	223	222	4	217	2
add64	0	4	0	12	146	9	338	112	1393	5892	2948	2944	19	2925	23
add128	0	4	0	-	*	-	-	-	-	-	-	-	-	-	-

Summary

- First proof generator for BDD-based SAT-solving
 - Moreover first prover that automatically generates Extended Resolution proofs
- Extended resolution proofs as generic proof format
- Enabler for further applications of extended resolution