BDDs and Extended Resolution

Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

Alpine Verification Meeting 2006

Centro Stefano Franscini
Monte Verità, Ascona, Switzerland
6. May 2006

Joint Work with Carsten Sinz and Toni Jussila

- tremendous progress in recent years
  - learning: RelSAT, GRASP
  - watched literals: SATO, CHAFF, MiniSAT
  - decision heuristics: DLIS, VSIDS, BerkMin
  - preprocessing: Hyper Binary Resolution, SATeLite
- also pushed by applications
  - Propositional Properties, Consistency, Bounded Model Checking
  - core engine in many applications of verification
- extended and partially replaced BDD based techniques in the last 6 years

- preprocessing
  - faster algorithms (linear in practice)
    - equivalence reasoning, hyper binary resolution, variable instantiation, recursive learning / Stålmarck's Method
  - integration with conflict driven assignment loop
    - all of those above plus clause distribution and BDDs
  - proofs!
- combination with Theories resp. Constraints
  - CSP, SMT, bit vector verification problems as in Cogent, C32SAT

Why Proofs?

A. Biere, FMV, JKU Linz

- **certification**, **testing**, e.g. correctness of SAT solver implementations

- **diagnosis**, e.g. *reason* for unsatisfiability of product configuration

- abstraction and **refinement**

  – if abstract counter example can not be mapped to concrete domain

  – concept of **core** variables or core clauses

  – refinement only needs to take core into account

- interpolation, **interpolands** can be generated from resolution proof

Proofs Background

A. Biere, FMV, JKU Linz

- clause learning enables simple proof generation
  - *antecedents* of learned clause as input clause
  - each variable is resolved only once (*regular*)
  - *linear* resolution chain, each resolution consumes one *input* clause
- clause learning as proof system
  - arbitrary cuts (not just 1st UIP) allow to simulate arbitrary regular input clause resolution proofs
  - with arbitrary restarts even this restriction is lifted
  - even in practice there seem to be even better cuts than 1st UIP cuts: JeruSAT, new versions of MiniSAT (1.14) and ZCHAFF

Proof Example

A. Biere, FMV, JKU Linz

The following general resolution proof:

```
     1 2     -1 2   /      1 -2     -1 -2
     ------------  /       --------------
          2       /               -2
          -------/----------------         2nd chain
                /    0
1st chain      /
```

can be split into two regular input resolution chains

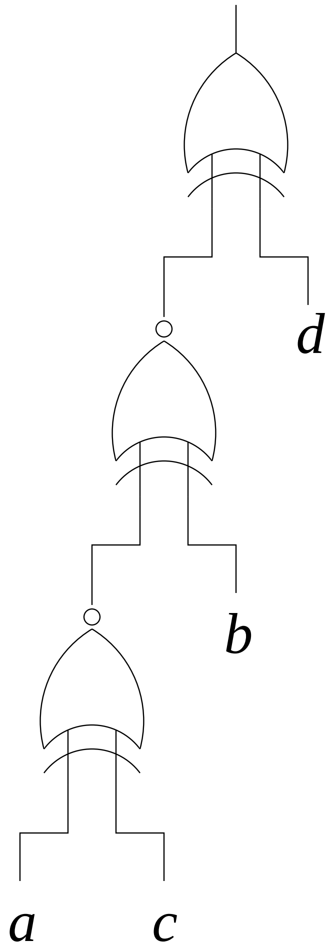
```
1 1 2   0       0   c original clause 1 (no antecedents)
2 -1 2  0       0   c original clause 2 (no antecedents)
3 1 -2  0       0   c original clause 3 (no antecedents)
4 -1 -2 0       0   c original clause 4 (no antecedents)
5 2     0 1 2   0   c learned clause 5 (antecedents 1 2)
6       0 3 4 5 0   c learned clause 6 (antecedents 3 4 5)
```

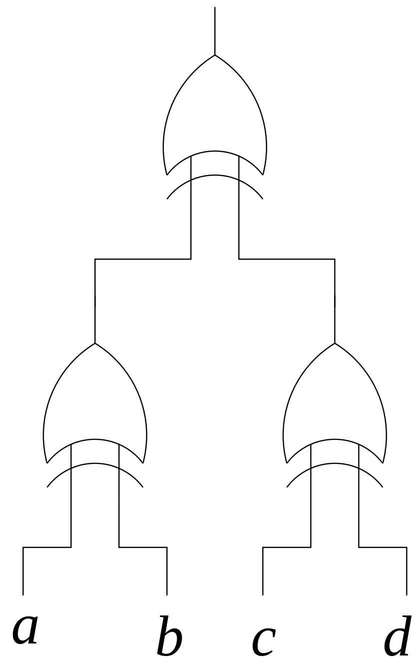
Pidegeon Hole Formulas and XORs

A. Biere, FMV, JKU Linz

- **pidgeon hole problems**: fit n pidgeons into $n-1$ holes
  - example application: FPGA routing
  - have exponential sized resolution proofs, BDDs work better
  - manually constructed polynomial extended resolution proofs exists
  - can be generalized to cardinality constraints
- problems with **XORs** can be bad for resolution, e.g. Urquhart problems
  - application: arithmetic circuits, such as multipliers
  - can solved by equivalence reasoning, e.g. Gauss over GF(2)
  - Gauss can be simulated polynomially by BDD operations

Equivalence Checking of XOR Chains





Contribution

- BDD based SAT solver EBDDRES
  - conjoins clauses by BDD-and operation
  - quantifies variables by bucket elimination
  - generates extended resolution proof if result *false* (new)
- generated proofs for examples hard for DPLL based solvers with learning
  - pigeon hole problem, XORs, . . .
  - still could not handle some problems, supposed to be easy for BDDs
- allows for instance BDD based preprocessing with proofs

- proposed by Tseitin'70 as an extension to resolution of Robinson'65
  - extend CNF by definitions of fresh variables

    $x = a \wedge b$ results in clauses $(\overline{x}\, a)(\overline{x}\, b)(x\, \overline{a}\, \overline{b})$

  - but otherwise do not change anything!
- one of the **most powerful** proof systems
  - definitions can represent lemmas, functions, ...
  - which variable to introduce?
  - so far only manually constructed extended resolution proofs existed
- EBDDRES generates extended resolution proofs automatically

- for each BDD f **define** a variable f in the CNF (using the same name)

$$f = \text{if } x \text{ then } f_1 \text{ else } f_0 \qquad g = \text{if } x \text{ then } g_1 \text{ else } g_0$$

$$\begin{array}{ll}(\overline{f}\;\overline{x}\;f_1) & (\overline{g}\;\overline{x}\;g_1)\\(\overline{f}\;x\;f_0) & (\overline{g}\;x\;g_0)\\(f\;\overline{x}\;\overline{f_1}) & (g\;\overline{x}\;\overline{g_1})\\(f\;x\;\overline{f_0}) & (g\;x\;\overline{g_0})\end{array}$$

- clauses are trivially converted into BDDs
  - long chain of BDD nodes, one child of each node *true*
  - for the top node f of the BDD one can derive the unit clause (f)
- proof that every “cache line” can be resolved (through ordinary resolution)

Handling Clauses: Example

A. Biere, FMV, JKU Linz

```
1 -1 2 3 0 0          c original clause

2 4 0 0               c constant true BDD node

11 -7 -3 4 0 0        c             7(3)   (node 7, variable 3)
12 -7 3 6 0 0         c              : \
13 7 -3 -4 0 0        c              :  \
14 7 3 -6 0 0         c              :   4 (true)
                      c              :
7 -6 -2 4 0 0         c             6(2)   (node 6, variable 2)
8 -6 2 5 0 0          c              : \
9 6 -2 -4 0 0         c              :  \
10 6 2 -5 0 0         c              :   4 (true)
                      c
3 -5 -1 -4 0 0        c             5(1)   (node 5, variable 2)
4 -5 1 4 0 0          c              : \
5 5 -1 4 0 0          c              :  \
6 5 1 -4 0 0          c       4 (true)   -4 (false)

15 7 0 1 13 14 9 10 6 2 0
```

Handling Cache Lines

A. Biere, FMV, JKU Linz

cache line: $(\overline{f}\ \overline{g}\ h)$

$$(\text{if } x \text{ then } f_1 \text{ else } f_0) \quad \wedge \quad (\text{if } x \text{ then } g_1 \text{ else } g_0) \quad \equiv \quad (\text{if } x \text{ then } h_1 \text{ else } h_0)$$

$$\begin{array}{ccc}
(\overline{f}\ \overline{x}\ f_1) & (\overline{g}\ \overline{x}\ g_1) & (\overline{h}\ \overline{x}\ h_1) \\
(\overline{f}\ x\ f_0) & (\overline{g}\ x\ g_0) & (\overline{h}\ x\ h_0) \\
(f\ \overline{x}\ \overline{f_1}) & (g\ \overline{x}\ \overline{g_1}) & (h\ \overline{x}\ \overline{h_1}) \\
(f\ x\ \overline{f_0}) & (g\ x\ \overline{g_0}) & (h\ x\ \overline{h_0})
\end{array}$$

$$\dfrac{\dfrac{(hx\overline{h_0}) \qquad \dfrac{(\overline{g}xg_0) \qquad \dfrac{(\overline{f}xf_0) \qquad \dfrac{\vdots}{(\overline{f_0}\,\overline{g_0}h_0)}}{(\overline{f}x\overline{g_0}h_0)}}{(\overline{f}\,\overline{g}xh_0)}}{(\overline{f}\,\overline{g}hx)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\vdots}{(\overline{f_1}\,\overline{g_1}h_1)} \qquad (\overline{f}\,\overline{x}f_1)}{(\overline{f}\,\overline{x}\,\overline{g_1}h_1)} \qquad (\overline{g}\,\overline{x}g_1)}{(\overline{f}\,\overline{g}\,\overline{x}h_1)} \qquad (h\overline{x}\,\overline{h_1})}{(\overline{f}\,\overline{g}h\overline{x})}}{(\overline{f}\,\overline{g}h)}$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	MINISAT			EBDDRES											
	solve resources		trace size	solve resources		trace gen.	trace size		bdd nodes	recursive bdd and-steps					trace chk
							ASCII	binary		all	triv.	lines	red.	core	
	sec	MB	MB	sec	MB	sec	MB	MB	$\times 10^3$	$\times 10^3$	$\times 10^3$	$\times 10^3$	$\times 10^3$	$\times 10^3$	sec
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	-	*	-	-	-	-	-	-	-	-	-	-

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	MINISAT			EBDDRES							EBDDRES, quantification						
	solve resources		trace size	solve resources		trace				bdd nodes	solve resources		trace				bdd nodes
						gen	ASCII	bin	chk				gen	ASCII	bin	chk	
	sec	MB	MB	sec	MB	sec	MB	MB	sec	$\times 10^3$	sec	MB	sec	MB	MB	sec	$\times 10^3$
ph7	0	0	0	0	0	0	1	0	0	3	0	5	0	12	4	1	60
ph8	0	4	1	0	0	0	3	1	0	15	1	14	1	49	15	4	236
ph9	6	4	11	0	0	0	3	1	0	8	6	52	4	186	59	14	864
ph10	44	4	63	1	17	1	30	10	2	136	20	214	16	683	*	*	2974
ph11	884	6	929	1	13	1	21	8	2	35	-	*	-	-	-	-	-
ph12	*	-	-	2	22	1	33	12	3	31	-	*	-	-	-	-	-
ph13	*	-	-	10	126	7	260	92	20	850	-	*	-	-	-	-	-
ph14	*	-	-	9	111	7	204	74	18	166	-	*	-	-	-	-	-
mutcb8	0	0	0	0	0	0	2	1	0	10	0	0	0	3	1	0	16
mutcb9	0	4	0	0	5	0	5	2	0	27	0	4	0	6	2	0	35
mutcb10	0	4	1	0	8	0	11	4	1	58	0	5	0	11	4	1	59
mutcb11	1	4	4	1	17	1	31	10	2	153	1	8	1	23	7	2	123
mutcb12	8	4	22	2	32	2	69	22	5	320	1	13	1	38	12	3	198
mutcb13	112	5	244	7	126	5	181	61	13	817	2	24	2	70	22	5	347
mutcb14	488	8	972	14	250	10	393	132	27	1694	4	37	3	127	40	8	621
mutcb15	*	-	-	36	498	26	1009	*	*	4191	6	52	5	211	67	14	1012
mutcb16	*	-	-	-	*	-	-	-	-	-	12	104	9	391	126	26	1821
urq35	95	4	218	2	22	1	37	13	3	24	0	0	0	1	0	0	5
urq45	*	-	-	-	*	-	-	-	-	-	0	0	0	1	0	0	10
urq55	*	-	-	-	*	-	-	-	-	-	0	0	0	2	1	0	15
urq65	*	-	-	-	*	-	-	-	-	-	0	4	0	6	2	0	34
urq75	*	-	-	-	*	-	-	-	-	-	0	4	0	7	2	0	39
urq85	*	-	-	-	*	-	-	-	-	-	0	5	0	10	3	1	59
fpga108	0	2		6	47	4	135	47	11	186	8	92	6	239	77	18	1088
fpga109	0	0		3	44	2	70	24	6	83	10	114	8	323	105	9	1434
fpga1211	0	0		53	874	37	1214	*	*	1312	-	*	-	-	-	-	-

Summary

- first **automatic generation** of extended resolution proofs
- extended resolutions proofs as **generic proof format**
- **proof checker** for resolution proofs “extends” easily
- **enabler** for further applications of extended resolution