Nenofex: Expanding NNF for QBF Solving

Florian Lonsing and Armin Biere

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

Nenofex

- "**Ne**gation **No**rmal **F**orm **Ex**pansion"
- Solver for Quantified Boolean Formulae (QBF)
  - propositional formula + quantified variables
  - generalizes SAT
- Features
  - tree-based NNF representation
  - NNF expansion: less size increase for $\exists$-expansion than on CNF
  - tight, estimated expansion costs for greedy scheduling
  - NNF redundancy removal: techniques from circuit optimization
- Results on QBFEVAL'07 benchmark set
  - less frequently out-of-memory than resolution-based *Quantor* [Biere-SAT04]
  - important, but expensive redundancy removal on NNF
  - strong performance on instances from adder familiy (QBFLIB, Ayari)

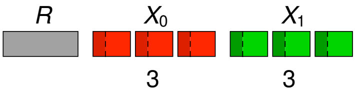
- QBF
  - PSPACE-complete decision problem
  - exponentially more succint than SAT
  - CNF + quantifier prefix (prenex normal form):

$$\underbrace{S_1 S_2 \dots S_{n-1} S_n}_{\text{quantifier prefix}} \underbrace{\phi}_{\text{CNF}}$$

  - S_i: linearly ordered *scopes*
    - two notions: sets of quantified variables and quantifier scopes (as usual)
    - quantifier scope of $x \in S_i$ in prefix ranges over whole formula ϕ
- Solving QBF by variable elimination:
  - from S_n to S_1
  - expansion, Q-resolution or skolemization
- Our focus: solve by expansion
  - Quantor: CNF-based, $\forall$-expansion for S_{n-1}, Q-resolution for S_n
  - Nenofex similar to Quantor but NNF-based, expansion only

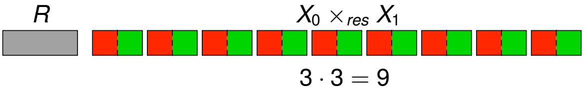
Motivation (1/2): NNF-expansion vs. CNF-resolution

Given: CNF $\phi \equiv R \wedge X_0 \wedge X_1$ with only $\exists$-variables



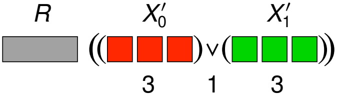
- sets X_0, X_1, R: clauses with negative, positive or no literal of variable x

Resolve x**:** $\phi_{res} \equiv R \wedge \bigwedge_{c \in (X_0 \times_{res} X_1)} c$

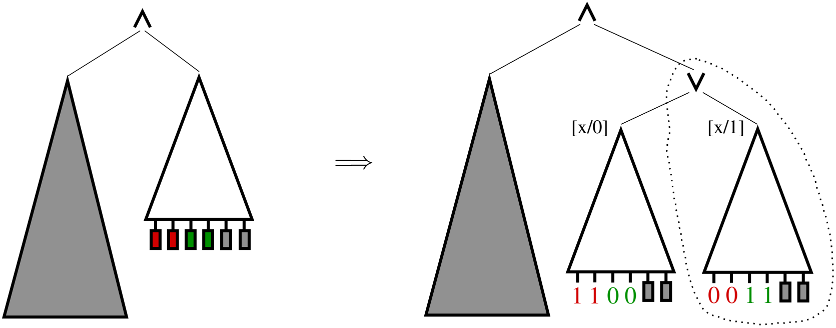


- generally: add $|X_0| \cdot |X_1|$ resolvents
- worst case: *quadratic* size increase

Expand x**:** $\phi_{exp} \equiv R \wedge ((X_0 \wedge X_1)[x/0] \vee (X_0 \wedge X_1)[x/1])$



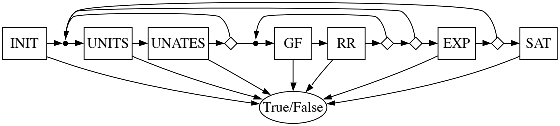
- add copy of ϕ by $\vee$, factor out R and assign x
- worst case: *linear* size increase



General ∃-expansion on NNF

- ϕ_{exp} grows linearly in the size of the subformula of x
- NNF allows compact representation for expanding ∃-variables
- size increase in ∀-expansions: NNF and CNF equivalent

Core Algorithm

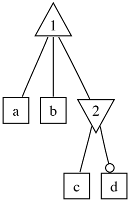


- Elimination of unit and pure literals (unates)
- Redundancy Removal
  - on small subformula only, cutoff criterion
- Expansion: $S_1 \ldots S_{n-1} S_n \ \phi$
  - from S_n to S_1, expand cheapest variable in S_{n-1} or S_n by scores
  - score(x): tight upper bound on size increase of NNF when expanding x
    - partial score recomputation
- SAT solving
  - only $\forall$($\exists$)-variables left $\rightarrow$ generate CNF by Tseitin transformation
  - PicoSAT backend

Negation Normal Form: only $\vee$ and $\wedge$, $\neg$ applied to literals only

NNF-trees

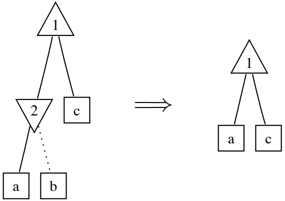
- internal nodes: operators $\vee$ and $\wedge$
- leaves: literal occurrence nodes (no sharing)
- $level(node) :=$ distance to root



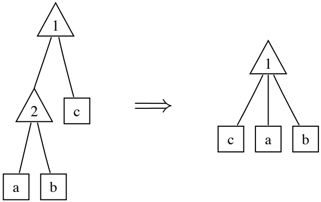
$a \wedge b \wedge (c \vee \neg d)$

Structural Restrictions: flat and compact NNF-trees (particularly for CNFs)

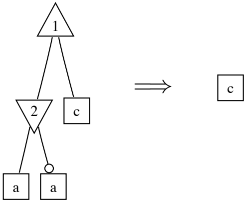
- number of children $n \geq 2$: operators denote n-ary boolean functions
  - $n = 1$ after deletion: merge nodes



- alternating types: $\mathit{type}(\mathit{parent}) \neq \mathit{type}(\mathit{child})$
  - apply associativity of $\vee$ and $\wedge$
  - prerequisite: n-ary operators



- one-level simplification: for var. $x, \otimes \in \{\vee, \wedge\}$, simplify $x \otimes x, x \otimes \neg x$
  - remove trivial redundancy
  - bottom-up recursive effects



Expansion (1/2)

Local Expansion for NNF: copy only relevant parts

- Def.: *ers(x)* $:=$ expansion-relevant subformula of variable x
  - smallest subformula which contains all occurrences of x
- finding *ers*(x) by scope reduction [AyariBasin02] in prenex formulae:

$$Qx(\phi \otimes \psi) \equiv Qx(\phi) \otimes \psi \qquad x \notin Vars(\psi), Q \in \{\forall, \exists\}, \otimes \in \{\vee, \wedge\}$$

In NNF-trees: for *ers(x)*, find *expansion-relevant subtree* to be copied

- correspondence: smallest subformulae and subtrees

Expansion-relevant LCAs of Variables: scope reduction in NNF-tree

- LCA: least common ancestor of set of nodes
- bottom-up approach for computing *ers(x)* starting from literals of x
- expansion-relevant LCA of x denotes expansion-relevant subtree

Expansion-relevant LCAs of Variables

- Def.: *expansion-relevant LCA* of $x :=$ node $lca(x)$ and set *LCA-children*
- set *LCA-children*: (proper) subset of children of node $lca(x)$
  - LCA-child c: subtree of c contains at least one occurrence of variable x

Expansion: $S_1 \ldots S_{n-1} S_n\ \phi$, $x \in S_n$, $type(S_n) \in \{\forall, \exists\}$

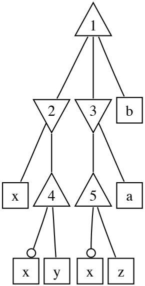
- replace *ers(x)* by $(ers(x)[x/0] \otimes ers(x)[x/1])$, $\otimes \in \{\vee, \wedge\}$

Expansion: $x \in S_{n-1}$, $type(S_{n-1}) = \forall$

- duplicate depending $\exists$-variables D_x from S_n

$$\begin{array}{rcl} D_x^{(0)} & := & \{y \in S_n \mid y \text{ has literals in } ers(x)\} \\ D_x^{(k+1)} & := & \{z \in S_n \mid z \text{ has literals in } ers(y') \text{ for some } y' \in D_x^k\}, k \geq 0 \\ D_x & := & \bigcup_k D_x^k \end{array}$$

- D_x: extended from CNF [BubeckKBüning-SAT07] to NNF
- universal expansion-relevant subformula $urs(x)$
  - contains all literals of x and of depending variables in D_x



NNF Redundancy Removal

Global Flow (GF): global analysis of logical flow of values

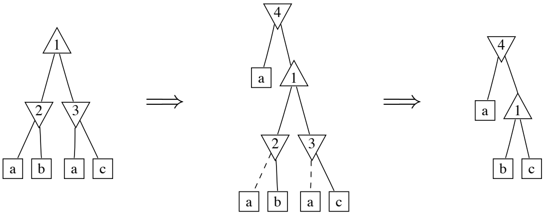
- implications: transform circuit to reduce size

$$x = 0 \rightarrow y = 0 : \; y \equiv x \wedge y \qquad x = 1 \rightarrow y = 1 : \; y \equiv x \vee y$$

Redundancy Removal (RR) by Automatic Test Pattern Generation (ATPG)

- ATPG: structural testing of circuits (NP-complete)
- assume fault f at single signal s in circuit C: stuck-at-{0,1} fault model
- find input $v = (pi_0, \ldots, pi_n)$ such that $C(v) \neq C_f(v)$ uniquely caused by f
- no such v: f is not testable, does not affect C, can remove HW at s

GF+RR implementation: incomplete, polynomial-time



- full benchmark set (1136 instances) from QBFEVAL'07
- Pentium IV 3.0 GHz, Ubuntu Linux, limits 900 seconds and 1.5 GB
- Quantor as reference: CNF-based, similar strategy
  - three versions of Nenofex: GF, RR enabled/disabled

		Nenofex		
	Quantor	*GF, RR*	*no GF, RR*	*no GF, no RR*
Solved	**421**	361	352	313
OOT	**32**	124	103	83
OOM	683	**651**	681	740
MEM-$\cup$	1.10e6	1.15e6	1.17e6	1.23e6
MEM-$\cap$	10473	18472	19693	28422

	Quantor only	Both	Nenofex only	Sum
Solved	79	342	19	440
OOT	18	14	110	142
OOM	80	603	48	731

Results

- less space-outs than CNF-based Quantor
  - node implementation in Nenofex not optimized for memory
- redundancy removal expensive but crucial for performance
  - GF, RR cause more time outs
- 19 uniquely solved instances

- equivalence checking of *n*-bit ripple-carry adders [AyariBasin02]
  - structured QBF-encodings of monadic second order formulae
  - hard instances in previous QBF evaluations

	optimizations enabled				*optimizations disabled*			
Name	*SAT-Vars*	*SAT-Clauses*	*Time (Exp.)*	*Mem*	*SAT-Vars*	*SAT-Clauses*	*Time (Exp.)*	*Mem*
adder-2-unsat	41	94	0.07 (0.07)	<1	46	106	<0.01	<1
adder-4-unsat	240	585	0.37 (0.36)	2.6	284	712	0.06 (0.04)	<1
adder-6-unsat	722	1748	1.32 (1.22)	4.2	892	2241	0.23 (0.10)	4.2
adder-8-unsat	1586	3776	2.89 (2.60)	6.6	2004	5038	0.66 (0.24)	6.6
adder-10-unsat	3098	7277	5.62 (4.67)	10.2	3892	9745	1.88 (0.50)	10.2
adder-12-unsat	5126	12007	9.58 (7.47)	15.1	6644	16552	5.04 (0.89)	15.1
adder-14-unsat	8064	18755	15.48 (11.10)	21.3	10448	25999	13.31 (1.54)	21.3
adder-16-unsat	11921	27565	24.90 (15.35)	29.2	15596	38638	31.13 (2.47)	29.2
adder-2-sat	60	133	0.04 (0.04)	<1	76	118	<0.01	<1
adder-4-sat	236	549	0.39 (0.38)	2.4	550	1386	0.05 (0.04)	<1
adder-6-sat	1358	3259	1.58 (1.42)	3.3	1855	4779	0.39 (0.13)	3.3
adder-8-sat	6016	14663	4.91 (3.23)	5.0	5073	13127	1.64 (0.39)	4.7
adder-10-sat	8563	20901	8.87 (5.86)	6.9	10421	26988	5.94 (1.24)	7.7
adder-12-sat	17099	41795	20.10 (10.24)	11.4	20518	52481	17.86 (3.34)	14.6
adder-14-sat	56947	141095	92.29 (23.45)	67.4	39935	103316	53.32 (9.21)	23.5
adder-16-sat	85836	213038	173.80 (42.94)	46.5	119018	309598	372.58 (41.50)	65.6

- Results
  - SAT-solving time dominates expansion time in large instances
  - no optimizations: less expansion time but larger CNFs
  - Quantor, sKizzo, squolem, ebddres:
    - comparable on adder-{2,4}-{sat,unsat}, sKizzo slower on adder-{2,...,10}-sat
    - abort on adder-{12,14,16}-{sat,unsat}, adder-{6,..,16}-unsat

Summary: Nenofex

- Expansion-based QBF solver for NNF
  - $\exists$-expansion: linear vs. quadratic size increase on NNF and CNF
  - NNF-trees: flat formula representation
- Local expansion: scope reduction by quantifier rules
  - expansion-relevant subformulae, subtrees and LCAs
- Variables scores for greedy scheduling
  - tight upper bound on actual size increase of NNF-tree
- Redundancy removal: treat NNF-tree as circuit
  - GF: deriving implications for circuit transformations
  - ATPG-based RR: untestable faults correspond to redundant HW
  - implementation: incomplete, on small subtree only
- Experiments
  - less space-outs than CNF-based solver Quantor
  - GF+RR crucial for performance, although NNF more compact than CNF
  - adder-benchmarks
- Future work
  - optimize for run time and memory
  - incremental maintainance of scores