Truth Assignments as Conditional Autarkies

Armin Biere

joint work with

Benjamin Kiesl (CISPA, Saarbrücken)
Marijn Heule (CMU, Pittsburgh)

17th International Symposium on
Automated Technology for Verification and Analysis

ATVA'2019

Academia Sinica, Taipei, Taiwan
October 30, 2019


SAT-Race 2010 Award
"Plingeling"
by Armin Biere
is awarded the title of
Best Parallel Solver
Chair of SAT-Race Organizing Committee
SAT-Race 2010 Award
"Lingeling"
by Armin Biere
is awarded the title of
Second Prize Winn
Edinburgh, Scotland, July 14, 2010
Tenth International Conference
On Theory and Applications of Satisfiability Testing
Competition 2007
Gold medal
Twelfth International Conference
On Theory and Applications of Satisfiability Testing
tition 2009
d medal
Eighteenth International Conference
on Theory and Applications of Satisfiability Testing
SAT-Race 2015
Main Track
Special Prize:
Most Innovative Solver
Written by:
Armin Biere
3rd Prize
2nd Prize
SAT-Race 2010
Best Parallel Solver
2010
SAT COMPE TITION 2013
Twelfth International Conference
On Theory and Applications of Satisfiability Testing
Competition 200
Silver meda

Satisfiability (SAT) related topics have attracted researchers from various disciplines. Logic, applied areas such as planning, scheduling, operations research and combinatorial optimization, but also theoretical issues on the theme of complexity, and much more, they all are connected through SAT.

My personal interest in SAT stems from actual solving: The increase in power of modern SAT solvers over the past 15 years has been phenomenal. It has become the key enabling technology in automated verification of both computer hardware and software. Bounded Model Checking (BMC) of computer hardware is now probably the most widely used model checking technique. The counterexamples that it finds are just satisfying instances of a Boolean formula obtained by unwinding to some fixed depth a sequential circuit and its specification in linear temporal logic. Extending model checking to software verification is a much more difficult problem on the frontier of current research. One promising approach for languages like C with finite word-length integers is to use the same idea as in BMC but with a decision procedure for the theory of bit-vectors instead of SAT. All decision procedures for bit-vectors that I am familiar with ultimately make use of a fast SAT solver to handle complex formulas.

Decision procedures for more complicated theories, like linear real and integer arithmetic, are also used in program verification. Most of them use powerful SAT solvers in an essential way.

Clearly, efficient SAT solving is a key technology for 21st century computer science. I expect this collection of papers on all theoretical and practical aspects of SAT solving will be extremely useful to both students and researchers and will lead to many further advances in the field.

Edmund Clarke

Edmund M. Clarke, FORE Systems University Professor of Computer Science and Professor of Electrical and Computer Engineering at Carnegie Mellon University, is one of the initiators and main contributors to the field of Model Checking, for which he also received the 2007 ACM Turing Award.

In the late 90s Professor Clarke was one of the first researchers to realize that SAT solving has the potential to become one of the most important technologies in model checking.

ISBN 978-1-58603-929-5
ISSN 0922-6389

185

HANDBOOK
of satisfiability

Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh

IOS Press

Frontiers in Artificial Intelligence and Applications
HANDBOOK
of satisfiability
Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh
IOS Press

Part I. Theory and Algorithms

John Franco, John Martin:
A History of Satisfiability. 3-74

Steven David Prestwich:
CNF Encodings. 75-97

Adnan Darwiche, Knot Pipatsrisawat:
Complete Algorithms. 99-130

João P. Marques Silva, Inês Lynce, Sharad Malik:
Conflict-Driven Clause Learning SAT Solvers. 131-153

Marijn Heule, Hans van Maaren:
Look-Ahead Based SAT Solvers. 155-184

Henry A. Kautz, Ashish Sabharwal, Bart Selman:
Incomplete Algorithms. 185-203

Oliver Kullmann:
Fundaments of Branching Heuristics. 205-244

Dimitris Achlioptas:
Random Satisfiability. 245-270

Carla P. Gomes, Ashish Sabharwal:
Exploiting Runtime Variation in Complete Solvers. 271-288

Karem A. Sakallah:
Symmetry and Satisfiability. 289-338

Hans Kleine Büning, Oliver Kullmann:
Minimal Unsatisfiability and Autarkies. 339-401

Evgeny Dantsin, Edward A. Hirsch:
Worst-Case Upper Bounds. 403-424

Marko Samer, Stefan Szeider:
Fixed-Parameter Tractability. 425-454

Part II. Applications and Extensions

Armin Biere:
Bounded Model Checking. 457-481

Jussi Rintanen:
Planning and SAT. 483-504

Daniel Kroening:
Software Verification. 505-532

Hantao Zhang:
Combinatorial Designs by SAT Solvers. 533-568

Fabrizio Altarelli, Rémi Monasson, Guilhem Semerjian, Francesco Zamponi:
Connections to Statistical Physics. 569-611

Chu Min Li, Felip Manyà:
MaxSAT, Hard and Soft Constraints. 613-631

Carla P. Gomes, Ashish Sabharwal, Bart Selman:
Model Counting. 633-654

Rolf Drechsler, Tommi A. Junttila, Ilkka Niemelä:
Non-Clausal SAT and ATPG. 655-693

Olivier Roussel, Vasco M. Manquinho:
Pseudo-Boolean and Cardinality Constraints. 695-733

Hans Kleine Büning, Uwe Bubeck:
Theory of Quantified Boolean Formulas. 735-760

Enrico Giunchiglia, Paolo Marin, Massimo Narizzano:
Reasoning with Quantified Boolean Formulas. 761-780

Roberto Sebastiani, Armando Tacchella:
SAT Techniques for Modal and Description Logics. 781-824

Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, Cesare Tinelli:
Satisfiability Modulo Theories. 825-885

Stephen M. Majercik:
Stochastic Boolean Satisfiability. 887-925

NEWLY AVAILABLE SECTION OF
THE CLASSIC WORK

The Art of Computer Programming

VOLUME 4

Satisfiability

FASCICLE 6

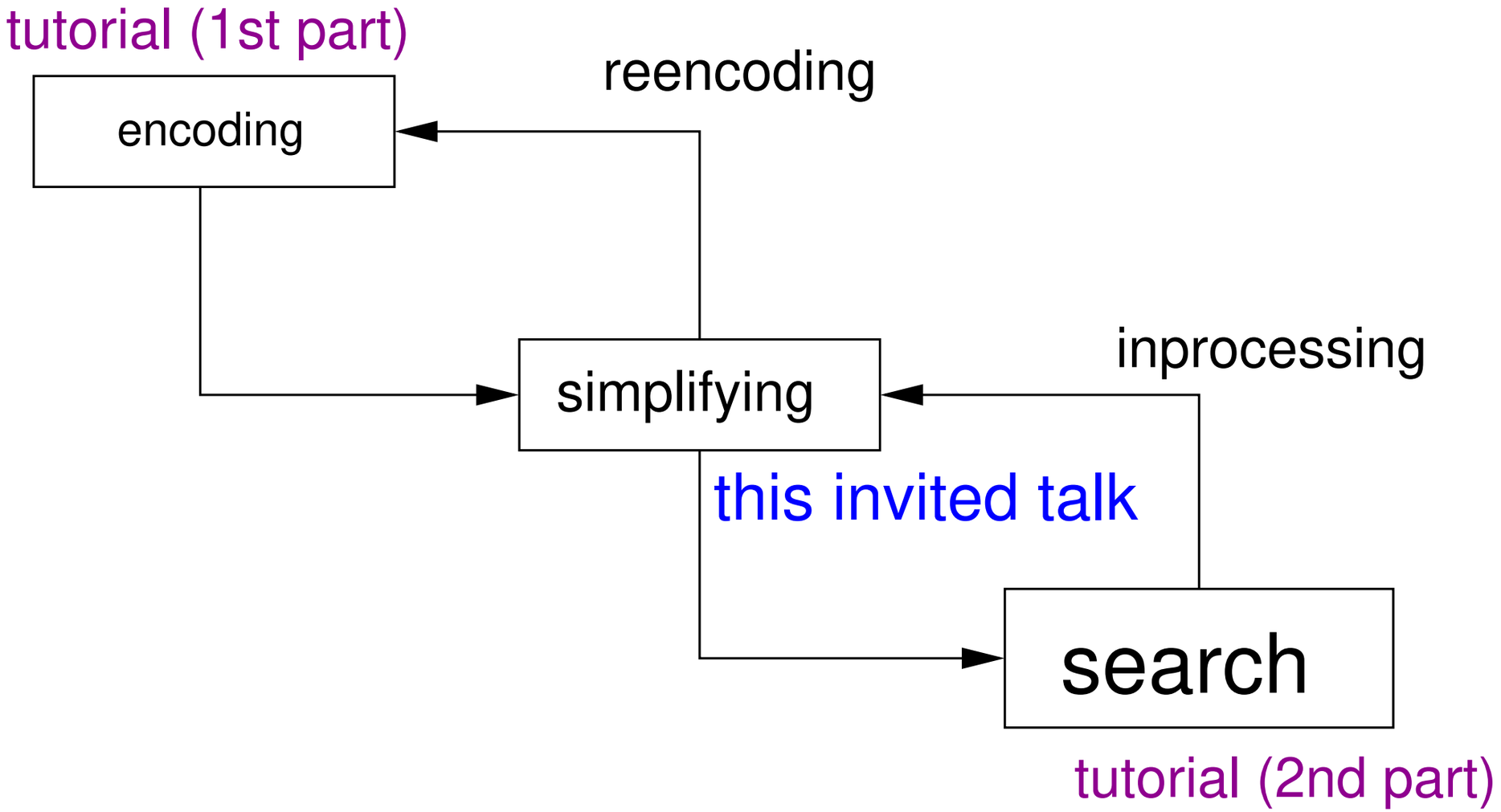
Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and . . . for their detailed comments on my early attempts at exposition, as well as to numerous other correspondents who have contributed crucial corrections. Thanks also to Stanford's Information Systems Laboratory for providing extra computer power when my laptop machine was inadequate.

Biere
Bryant
Buss
Eén
Gent
Heule
Hoos
Janson
Jeavons
Kroening
Kullmann
Lauria
Pegden
Shortz
Sinz
Sörensson
Wermuth
Williams
Internet
MPR
Internet

* * *

Wow—Section 7.2.2.2 has turned out to be the longest section, by far, in *The Art of Computer Programming*. The SAT problem is evidently a "killer app," because it is key to the solution of so many other problems. Consequently I can only hope that my lengthy treatment does not also kill off my faithful readers! As I wrote this material, one topic always seemed to flow naturally into another, so there was no neat way to break this section up into separate subsections. (And anyway the format of *TAOCP* doesn't allow for a Section 7.2.2.2.1.)

What is Practical SAT Solving?



DP Procedure

[DavisPutnam'60]

forever

if $F = \top$ **return** *satisfiable*

if $\bot \in F$ **return** *unsatisfiable*

pick remaining variable x

add all resolvents on x

remove all clauses with x and $\neg x$

$\Rightarrow$ Bounded Variable Elimination [EénBiere-SAT'05]

Bounded Variable Elimination

[EénBiere-SAT'05]

Replace $(\boxed{\bar{x}} \vee a)_1$ $(\boxed{\bar{x}} \vee b)_2$ $(\boxed{\bar{x}} \vee c)_3$ $(\boxed{x} \vee \bar{a} \vee \bar{b})_4$ $(\boxed{x} \vee d)_5$ by $\cancel{(a \vee \bar{a} \vee \bar{b})_{14}}$ $(a \vee d)_{15}$ $\cancel{(b \vee \bar{a} \vee \bar{b})_{24}}$ $(b \vee d)_{25}$ $(c \vee \bar{a} \vee \bar{b})_{34}$ $(c \vee d)_{45}$

- number of clauses not increasing
- strengthen and remove subsumbed clauses too
- most important and most effective preproessing we have

Bounded Variable Addition

[MantheyHeuleBiere-HVC'12]

Replace $(a \vee d)$ $(b \vee d)$ $(c \vee d)$ $(a \vee e)$ $(b \vee e)$ $(c \vee e)$ by $(\bar{x} \vee a)$ $(\bar{x} \vee b)$ $(\bar{x} \vee c)$ $(x \vee d)$ $(x \vee e)$

- number of clauses has to decrease strictly
- reencodes for instance naive at-most-one constraint encodings

Proofs / RES / RUP / DRUP

- resolution proofs (RES) are simple to check but large and hard(er) to produce directly
- original idea for clausal proofs and checking them:
  - proof traces are sequences of “learned clauses” C
  - first check clause through unit propagation $\boxed{F \vdash_1 C}$ then add C to F
  - reverse unit implied clauses (RUP) [GoldbergNovikov’03] [VanGelder’12]
- deletion information:
  - “deletion” proof lines tell checker to forget clause and decreases checking time substantially
  - trace of added and deleted clauses (**D**RUP) [HeuleHuntWetzler-FMCAD’13 / STVR’14]
- RUP/RES tracks SAT Competion 2007, 2009, 2011, now DRUP/DRAT mandatory since 2013 to certify UNSAT
- big certified proofs:
  - Pythagorean Triples [HeuleKullmannMarek-SAT’16] (200TB), Schur Number Five [Heule-AAAI’18] (2PB)
  - Certification: Coq [CruzFilipeMarquesSilvaSchneiderKamp-TACAS’17 / JAR’19], ACL2, Isabelle

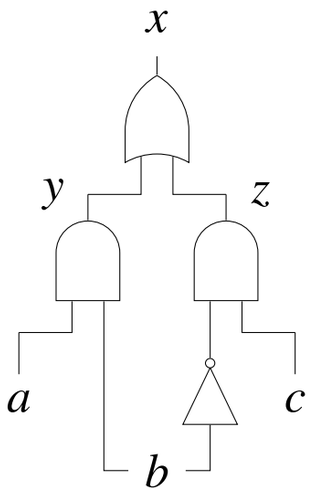
CNF	trace	extended trace	resolution trace	RUP	DRUP
p cnf 3 8					
-1 -2 -3 0	1 -2 -3 -1 0 0	1 -2 -3 -1 0 0	1 -1 -3 -2 0 0		
-1 -2 3 0	2 -2 3 -1 0 0	2 -2 3 -1 0 0	2 -1 3 -2 0 0		
-1 2 -3 0	3 2 -3 -1 0 0	3 2 -3 -1 0 0	3 2 -1 -3 0 0		
-1 2 3 0	4 2 3 -1 0 0	4 2 3 -1 0 0	4 2 -1 3 0 0		
1 -2 -3 0	5 1 -3 -2 0 0	5 1 -3 -2 0 0	5 -2 -3 1 0 0		
1 -2 3 0	6 1 3 -2 0 0	6 1 3 -2 0 0	6 -2 3 1 0 0		
1 2 -3 0	7 1 -3 2 0 0	7 1 -3 2 0 0	7 1 -3 2 0 0		
1 2 3 0	8 1 3 2 0 0	8 1 3 2 0 0	8 1 3 2 0 0		
	9 * 7 8 0	9 1 2 0 7 8 0	9 1 2 0 7 8 0	-2 -3 0	-2 -3 0
	10 * 9 5 6 0	10 1 0 9 5 6 0	10 -2 1 0 5 6 0	-3 0	d 1 -2 -3 0
	11 * 1 10 2 0	11 -2 0 1 10 2 0	11 1 0 10 9 0	2 0	d -1 -2 -3 0
	12 * 10 11 4 0	12 3 0 10 11 4 0	12 -1 -2 0 1 2 0	-1 0	-2 3 0
	13 * 10 11 3 12 0	13 0 10 11 3 12 0	13 -2 0 12 11 0	0	d 1 -2 3 0
			14 2 3 0 11 4 0		d -1 -2 3 0
			15 3 0 14 13 0		2 -3 0
			16 2 -3 0 11 3 0		d 1 2 -3 0
			17 -3 0 16 13 0		d -1 2 -3 0
			18 0 17 15 0		2 3 0
					d 1 2 3 0
					d -1 2 3 0
					-2 0
					0
	picosat -t	picosat -T	tracecheck -B	cadical	cadical -P1

Blocked Clause Elimination and Plaisted-Greenbaum Encoding and Monotone Input Removal

[Kullman-DAM'99] [JärvisaloHeuleBiere-TACAS'10] [JärvisaloHeuleBiere-JAR'12] [PlaistedGreenbaum-JSC'86]

Definition. Clause C *blocked* on literal $\boxed{\ell} \in C$ w.r.t CNF F if
for all resolution candidates $D \in F$ with $\bar{\ell} \in D$ the resolvent $(C \backslash \ell) \vee (D \backslash \bar{\ell})$ is tautological.

Assume output true, thus single unit clause constraint (x)



$$
\begin{array}{ccccc}
(x) & & (x) & & (x) \\
(\boxed{x} \vee \bar{y})_1\ (\boxed{x} \vee \bar{z})_2\ (\bar{x} \vee y \vee z) & & (\bar{x} \vee y \vee z) & & (\bar{x} \vee y \vee z) \\
(\bar{y} \vee a)\ (\bar{y} \vee b)\ (\boxed{y} \vee \bar{a} \vee \bar{b})_3 & \Rightarrow & (\bar{y} \vee \boxed{a})_5\ (\bar{y} \vee b) & \Rightarrow & (\bar{y} \vee b) \\
(\bar{z} \vee \bar{b})\ (\bar{z} \vee c)\ (\boxed{z} \vee b \vee \bar{c})_4 & & (\bar{z} \vee \bar{b})\ (\bar{z} \vee \boxed{c})_6 & & (\bar{z} \vee \bar{b})
\end{array}
$$

Plaisted-Greenbaum encoding *drops* **upward** propagating clauses of only **positively** occurring gates.
Plaisted-Greenbaum encoding *drops* **downward** propagating clauses of only **negatively** occurring gates.

Unconstrained or monotone inputs can be removed too.

Resolution Asymmetric Tautologies (RAT)

"Inprocessing Rules" [JärvisaloHeuleBiere-IJCAR'12]

- justify complex preprocessing algorithms in Lingeling [Biere-TR'10]
  - examples are adding blocked clauses or variable elimination
  - interleaved with research (forgetting learned clauses = reduce)
- need more general notion of redundancy criteria
  - extension of blocked clauses
  - replace "resolvents on $\boxed{l}$ are tautological" by "resolvents on $\boxed{l}$ are RUP"

example: $(a \vee \boxed{l})$ RAT on l w.r.t. $(a \vee b) \wedge (l \vee c) \wedge \underbrace{(\bar{l} \vee b)}_{D}$

  - deletion information is again essential (DRAT) [HeuleHuntWetzler-FMCAD'13 / STVR'14]
  - now mandatory in the main track of the SAT competitions since 2013
  - pretty powerful: can for instance also cover symmetry breaking

"Clause Elimination for SAT and QSAT"

by Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl and Armin Biere

has been selected as the winner of the

2019 IJCAI-JAIR Best Paper Prize

with the following citation:

This paper describes fundamental and practical results on a range of clause elimination procedures as preprocessing and simplification techniques for SAT and QBF solvers. Since its publication, the techniques described therein have been demonstrated to have profound impact on the efficiency of state-of-the-art SAT and QBF solvers.
The work is elegant and extends beautifully some well-established theoretical concepts. In addition, the paper gives new emphasis and impulse to pre- and in-processing techniques - an emphasis that resonates beyond the two key problems, SAT and QBF, covered by the authors.

The IJCAI-JAIR Best Paper Prize is awarded to an outstanding paper published in the Journal of Artificial Intelligence Research in the preceding five calendar years.

Macao, 13 August 2019

Shaul Markovitch
Editor-in-Chief, JAIR

Holger H. Hoos
Chair, 2019 IJCAI-JAIR Best Paper Prize Selection Committee
Associate-Editor-in-Chief, JAIR

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Structural Reasoning Methods for Satisfiability Solving and Beyond

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl.-Ing. Benjamin Kiesl, BSc
Registration Number 1127227

to the Faculty of Informatics

at the TU Wien

Advisors: Assoc.-Univ.Prof. Dr. Martina Seidl
a.o. Univ.-Prof. Dr. Hans Tompits

The dissertation has been reviewed by:

Olaf Beyersdorff

Christoph Weidenbach

Vienna, 20th February, 2019

Benjamin Kiesl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Set Blocked Clauses (SBC)

[KieslSeidlTompitsBiere-IJCAR'16] [KieslSeidlTompitsBiere-LMCS'18]

C is *set blocked* on $L \subseteq C$ iff $(C \setminus L) \cup \bar{L} \cup D$ is a tautology for all $D \in F$ with a literal in $\bar{L}$

- easy to check if the "witness" L is given
  - NP hard to check otherwise ("exponential" in $|L|$)
- local redundancy property
  - only considering the resolution environment of a clause
  - in constrast to (R)AT / RUP
- strictly more powerful than blocked clauses ($|L| = 1$)
- most general local redundancy property *super blocked clauses*
  - strictly more powerful than blocked clauses
  - Π_2^P complete to chec

Example:

$C = \boxed{\text{a}} \vee \boxed{\text{b}}$ set blocked

in $F = (\bar{a} \vee b) \wedge (a \vee \bar{b})$

by $L = \{a, b\}$

Redundancy

"Short Proofs Without New Variables" [HeuleKieslBiere-CADE'17] best paper

Definition. A partial assignment α *blocks* a clause C if α assigns the literals in C to false (and no other literal).

Definition. A clause C is *redundant* w.r.t. a formula F if F and $F \cup \{C\}$ are satisfiability equivalent.

Definition. A formula F simplified by a partial assignment α is written as $F|\alpha$.

Theorem.

Let F be a formula, C a clause, and α the assignment blocked by C.

Then C is redundant w.r.t. F iff exists an assignment ω such that

(i) ω satisfies C and (ii) $F|\alpha \models F|\omega$.

Propagation Redundant (PR)

[HeuleKieslBiere-CADE'17] [HeuleKieslBiere-JAR'19]

- more general than RAT: short proofs for pigeon hole formulas *without new variables*

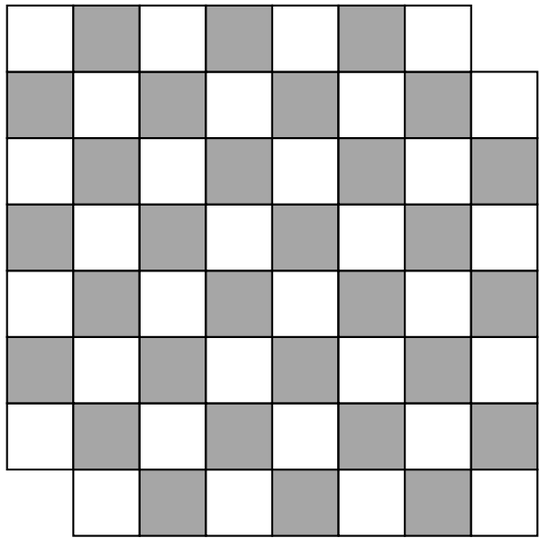
  C propagation redundant (PR) if exists assignment ω satisfying C with $F|_\alpha \vdash_1 F|_\omega$

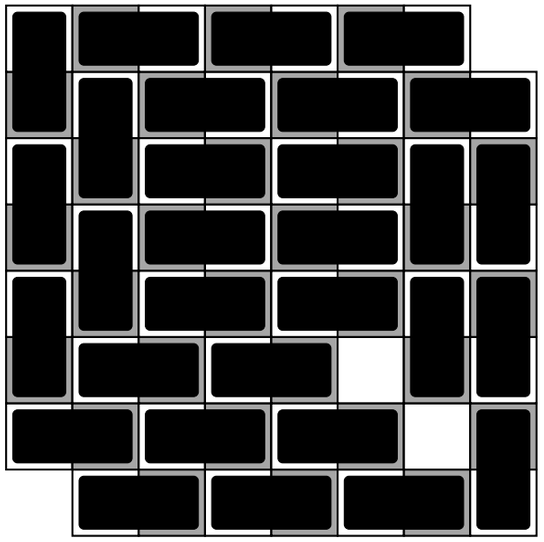
  so in essence replacing "$\models$" by "$\vdash_1$" (implied by unit propagation) where again α is the clause that blocks C

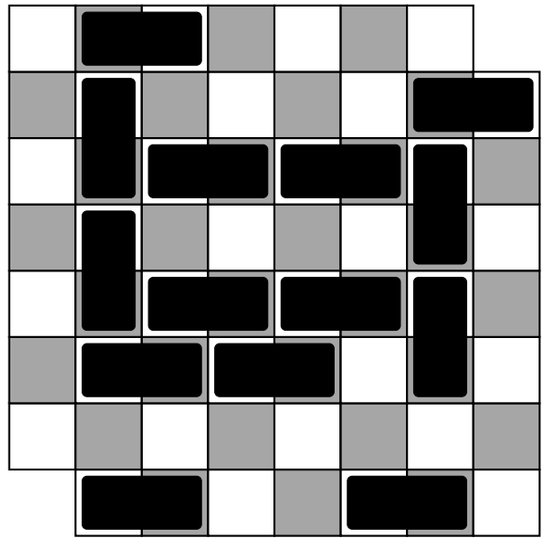
- Satisfaction Driven Clause Learning (SDCL) [HeuleKieslSeidlBiere-HVC'17] best paper
  - first automatically generated PR proofs
  - prune assignments for which we have other at least as satisfiable assignments
  - (filtered) positive reduct in SaDiCaL [HeuleKieslBiere-TACAS'19] nominated best paper
- translate PR to DRAT [HeuleBiere-TACAS'18]
  - only one additional variable needed
  - shortest proofs for pigeon hole formulas
- translate DRAT to extended resolution [KieslRebolaPardoHeule-IJCAR'18] best paper
- recent seperation results in [BussThapen-SAT'19]

Mutilated Chessboard

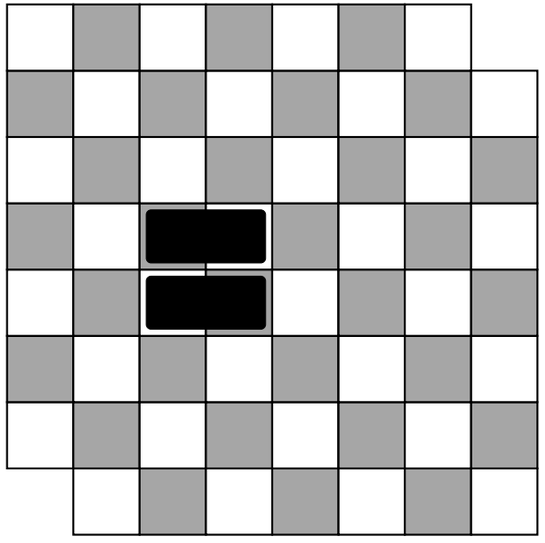
[HeuleKieslBiere-NFM'19]

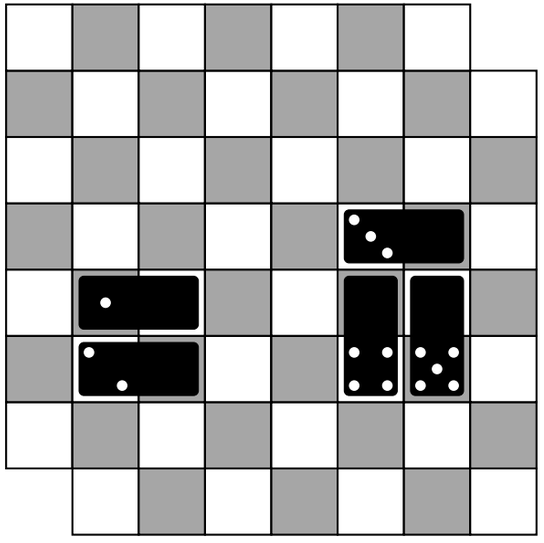


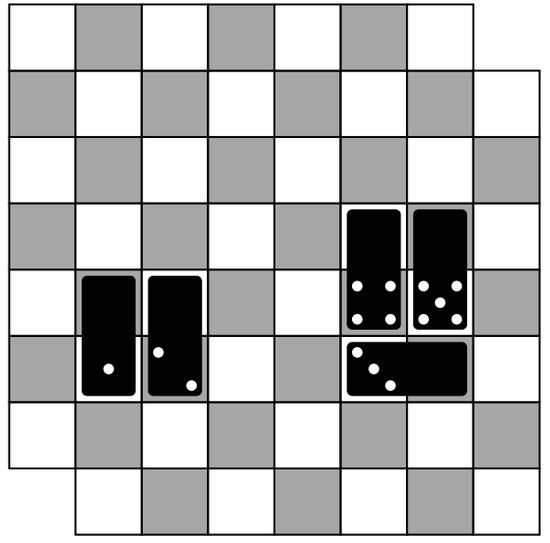




CDCL



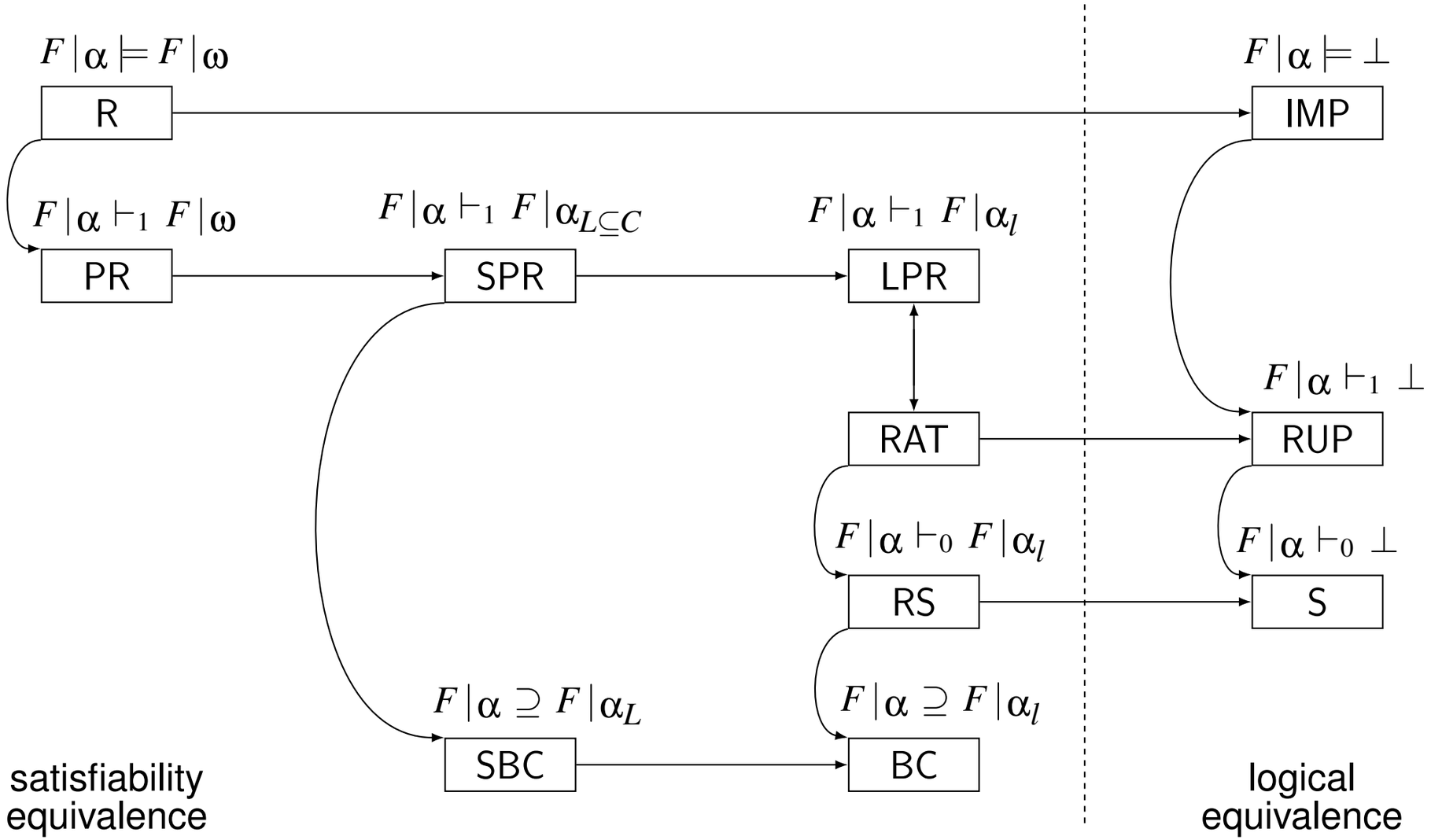




SDCL

Landscape of Clausal Redundancy

[HeuleKieslBiere-JAR'19]



CDCL(formula F)

```
1  α := ∅
2  forever do
3     α := UnitPropagate(F, α)
4     if α falsifies a clause in F then
5        C := AnalyzeConflict()
6        F := F ∧ C
7        if C is the empty clause ⊥ then return UNSAT
8        α := BackJump(C, α)
13    else
14       if all variables are assigned then return SAT
15       l := Decide()
16       α := α ∪ {l}
```

SDCL(formula F)

```
1  α := ∅
2  forever do
3     α := UnitPropagate(F, α)
4     if α falsifies a clause in F then
5        C := AnalyzeConflict()
6        F := F ∧ C
7        if C is the empty clause ⊥ then return UNSAT
8        α := BackJump(C, α)
9     else if the pruning predicate P_α(F) is satisfiable then
10       C := AnalyzeWitness()
11       F := F ∧ C
12       α := BackJump(C, α)
13    else
14       if all variables are assigned then return SAT
15       l := Decide()
16       α := α ∪ {l}
```

Positive and Filtered Positive Reduct

[HeuleKieslSeidlBiere-HVC'17] [HeuleKieslBiere-TACAS'19]

In the *positive reduct* consider all clauses satisfied by α but remove unassigned literals and add C:

Definition. Let F be a formula and α an assignment. Then, the *positive reduct* of F and α is the formula $G \wedge C$ where C is the clause that blocks α and $G = \{\mathsf{touched}_\alpha(D) \mid D \in F \text{ and } D|_\alpha = \top\}$.

Theorem. Let F be a formula, α an assignment, and C the clause that blocks α.
Then, C is $\boxed{\text{SBC}}$ by an $L \subseteq C$ with respect to F if and only if the assignment α_L satisfies the positive reduct.

We obtain the *filtered* positive reduct by not taking *all* satisfied clauses of F but only those for which the untouched part is not implied by $F|_\alpha$ via unit propagation:

Definition. Let F be a formula and α an assignment. Then, the *filtered positive reduct* of F and α is the formula $G \wedge C$ where C is the clause that blocks α and $G = \{\mathsf{touched}_\alpha(D) \mid D \in F \text{ and } F|_\alpha \not\vdash_1 \mathsf{untouched}_\alpha(D)\}$.

Theorem. Let F be a formula, α an assignment, and C the clause that blocks α.
Then, C is $\boxed{\text{SPR}}$ by an $L \subseteq C$ with respect to F if and only if the assignment α_L satisfies the filtered positive reduct.

where SPR extends SBC in the same way by propagation as RAT extends BC

Experiments

[HeuleKieslBiere-TACAS'19]

formula	MapleChrono	[HVC'17]	plain CDCL	positive	filtered	ACL2
Urquhart-s3-b1	2.95	5.86	16.31	> 3600	**0.02**	0.09
Urquhart-s3-b2	1.36	2.4	2.82	> 3600	**0.03**	0.13
Urquhart-s3-b3	2.28	19.94	2.08	> 3600	**0.03**	0.16
Urquhart-s3-b4	10.74	32.42	7.65	> 3600	**0.03**	0.17
Urquhart-s4-b1	86.11	583.96	> 3600	> 3600	**0.32**	2.37
Urquhart-s4-b2	154.35	1824.95	183.77	> 3600	**0.11**	0.78
Urquhart-s4-b3	258.46	> 3600	129.27	> 3600	**0.16**	1.12
Urquhart-s4-b4	> 3600	> 3600	> 3600	> 3600	**0.14**	1.17
Urquhart-s5-b1	> 3600	> 3600	> 3600	> 3600	**1.27**	9.86
Urquhart-s5-b2	> 3600	> 3600	> 3600	> 3600	**0.58**	4.38
Urquhart-s5-b3	> 3600	> 3600	> 3600	> 3600	**1.67**	17.99
Urquhart-s5-b4	> 3600	> 3600	> 3600	> 3600	**2.91**	24.24
hole20	> 3600	1.13	> 3600	**0.22**	0.55	6.78
hole30	> 3600	8.81	> 3600	**1.71**	4.30	87.58
hole40	> 3600	43.10	> 3600	**7.94**	20.38	611.24
hole50	> 3600	149.67	> 3600	**25.60**	68.46	2792.39
mchess_15	51.53	1473.11	2480.67	> 3600	**13.14**	29.12
mchess_16	380.45	> 3600	2115.75	> 3600	**15.52**	36.86
mchess_17	2418.35	> 3600	> 3600	> 3600	**25.54**	57.83
mchess_18	> 3600	> 3600	> 3600	> 3600	**43.88**	100.71

Simulating Headlines

FAN Algorithm from ATPG [Fujiwara-ISCAS'85]

CNF

$$F'(I,S,T,x,y,z) = H'(J,x,y,z,T) \wedge G'(K,S,x) \quad \text{with}$$

$$H'(J,x,y,z,T) = \underbrace{(\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (x \vee \overline{y} \vee \overline{z})}_{\text{Tseitin encoding of top AND gate in } H} \wedge H''(J,y,z,T)$$

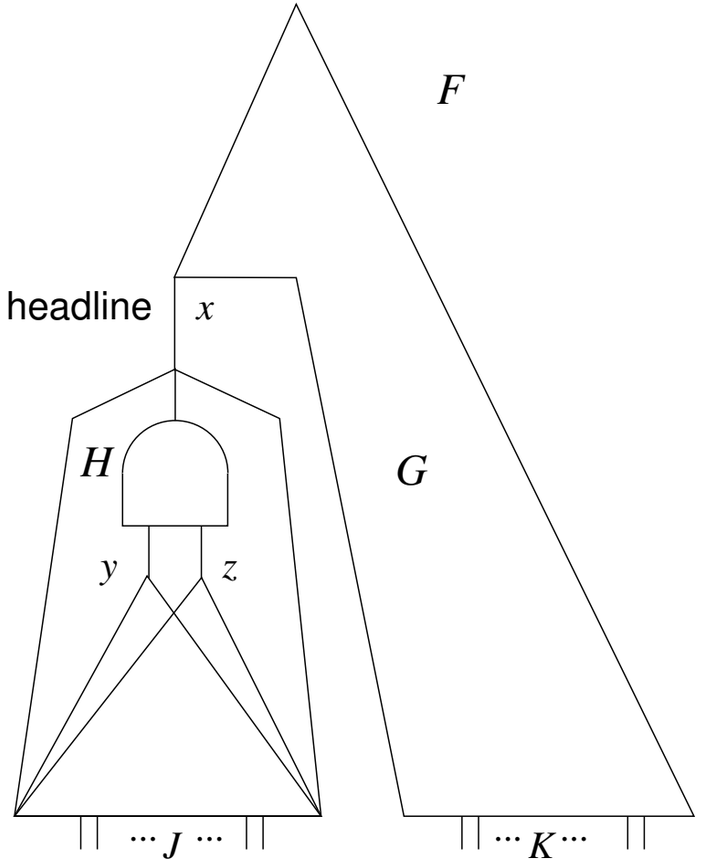
Formula

$$F(I) = G(H(J),K) = \exists x.\,(x = H(J)) \wedge G(x,K)$$

assume $\sigma_0(H(J)) = \sigma_0(x) = 0$

assume $\sigma_1(H(J)) = \sigma_1(x) = 1$

Drop H?



Autarkies and Conditional Autarkies

[MonienSpeckenmeyer-DAM'85] [HeuleKieslSeidlBiere-HVC'17]

Definition. Assignment α is an *autarky* for F if α satisfies all $C \in F$ with $var(\alpha) \cap var(C) \neq \emptyset$.

In other words, an autarky satisfies every clause it touches.

Example. Let $F = (a \vee b \vee \overline{c}) \wedge (\overline{b} \vee c \vee \overline{d}) \wedge (\overline{a} \vee d)$ and $\alpha = bc$.
Then, α touches only the first two clauses. Since it satisfies them, it is an *autarky* for F.

Definition. Assignment $\alpha = \gamma \cup \beta$ is a *conditional autarky* for F with conditional part γ and autarky part β if β satisfies all $C \in F|_\gamma$ with $var(\alpha) \cap var(C) \neq \emptyset$.

Thus a conditional autarky satisfies every clause its autarky part touches after applying the conditional part.

Example. Let $F = (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee d)$ and $\alpha = \gamma \cup \beta = abc$, $\gamma = a$, $\beta = bc$.
Then, β touches the first three clauses, α satisfies them, thus α is a *conditional autarky* for F with conditional part γ and autarky part β.

Globally Blocked Clauses

contribution of the paper in the proceedings

Definition. A clause C is *globally blocked* by a set $\boxed{L}$ of literals in a formula F if $L \cap C \neq \emptyset$ and for all $D \in F$ with a literal in $\overline{L}$ but no literal from L, the clause $(D \setminus \overline{L}) \vee C$ is a tautology.

Example. Let $F = (a \vee \overline{b} \vee \overline{c}) \wedge (\overline{a} \vee b \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{a} \vee d)$
then the clauses $(a \to b)$ and $(a \to c)$ are both globally blocked for $L = \{b, c\}$.

Theorem. Let F be a formula, let C be a clause, let L be a set of literals such that $L \cap C \neq \emptyset$,
Define the assignments $\gamma = \overline{C \setminus L}$ and $\beta = L$. Then, C is globally blocked by L in F iff $\gamma \cup \beta$ is a conditional autarky.

Thus globally blocked clauses can be found by “computing” conditional autarkies!

Algorithms

LeastConditionalPart(assignment α, formula F)

1 $\alpha_c := \emptyset$

2 **for** $C \in F$ **do**

3 **if** α touches C without satisfying C **then**

4 $\alpha_c := \alpha_c \cup (\alpha \cap \overline{C})$

5 **return** α_c

IsGloballyBlocked(clause C, formula F, assignment α)

6 $\alpha_c :=$ LeastConditionalPart(α, F), $\alpha_a := \alpha \setminus \alpha_c$

7 $\alpha' := \alpha_a \cup (\alpha_c \cap \overline{C})$

8 **if** $(\alpha' = \alpha)$ **then return** $\alpha_a \cap C \neq \emptyset$

9 **return** IsGloballyBlocked(C, F, α')

1. Split the assignment into a conditional part α_c and an autarky part α_a (one initial call to LeastConditionalPart).
   Mark the resulting literals of α_c and save them on a *conditional stack*, gather *candidate clauses* (those with a literal that is true but not yet in the conditional part) and watch a true literal in all clauses with a true literal.
2. For each candidate clause C:
3. If C contains no literal from α_a, continue with next clause (goto 2).
4. Watch one literal l_a of α_a in C and mark all literals in C to be part of C. Actually have a variable pointing to the literal l_a.
5. For each unprocessed literal l_c on the conditional stack:
6. If $\bar{l}_c \in C$ (cheap check since literals in C are marked) continue (goto 5).
7. Unassign $\bar{l}_c \in C$ and push it on an *unassigned* stack.
8. For each unassigned literal u on the unassigned stack not processed yet:
9. For each clause D watched by u (through watches initialized in step 1):
10. Search for a replacement literal $r \in D$ which satisfies D. If such r is found, stop watching D with u, watch it with r instead, and continue with next clause D watched by u (goto 9).
11. Otherwise no replacement is found.
12. If there is no literal $k \in \alpha_a$ with $\bar{k} \in D$, continue with next clause D watched by u (goto 9).
13. For each literal $k \in \alpha_a$ with $\bar{k} \in D$:
14. Put k into the conditional part α_c by using another mark bit and push it onto the conditional stack.
15. If k is different from the watched literal $l_a \in C$ (see step 4), continue with the next unassigned and unprocessed literal u on the unassigned stack.
16. Otherwise, search for a replacement of l_a in C.
17. If no replacement is found, C is not a globally-blocked clause; continue with next candidate clause
    (goto 2 – thus jump out of four loops) .
18. If there are no unprocessed literals, neither on the conditional nor on the unassigned stack, and we still watch a literal of α_a in the candidate clause C, then we now reached a fix-point and C is globally blocked.
19. Eliminate C and put the autarky part as witness (found by traversing the assignment trail) and C on the extension stack for witness reconstruction.
20. Pop literals from unassigned stack and reassign them to their original value.
21. Pop literals from conditional stack pushed after initialization in step 1 and unmark their conditional bit.
22. Now we are back to the initial assignment after step 1, with the initial literals of the conditional part α_c marked as such and the literals of α_a unmarked.
23. Unmark literals marked in step 4 and continue with next clause (goto 2).

https://github.com/arminbiere/cadical/blob/master/src/condition.cpp

Personal SAT Solver History