# SAT Solving

Armin Biere

Hausdorff School for Advanced Studies in Mathematics

Computational Combinatorial Optimization

Universität Bonn

September 12 - 16, 2022

# Dress Code of a Speaker at the Hausdorff School

- propositional logic:

|  |  |  |
|---|---|---|
| variables | **tie** **shirt** |  |
| negation | $\neg$ | (not) |
| disjunction | $\vee$ | (or) |
| conjunction | $\wedge$ | (and) |

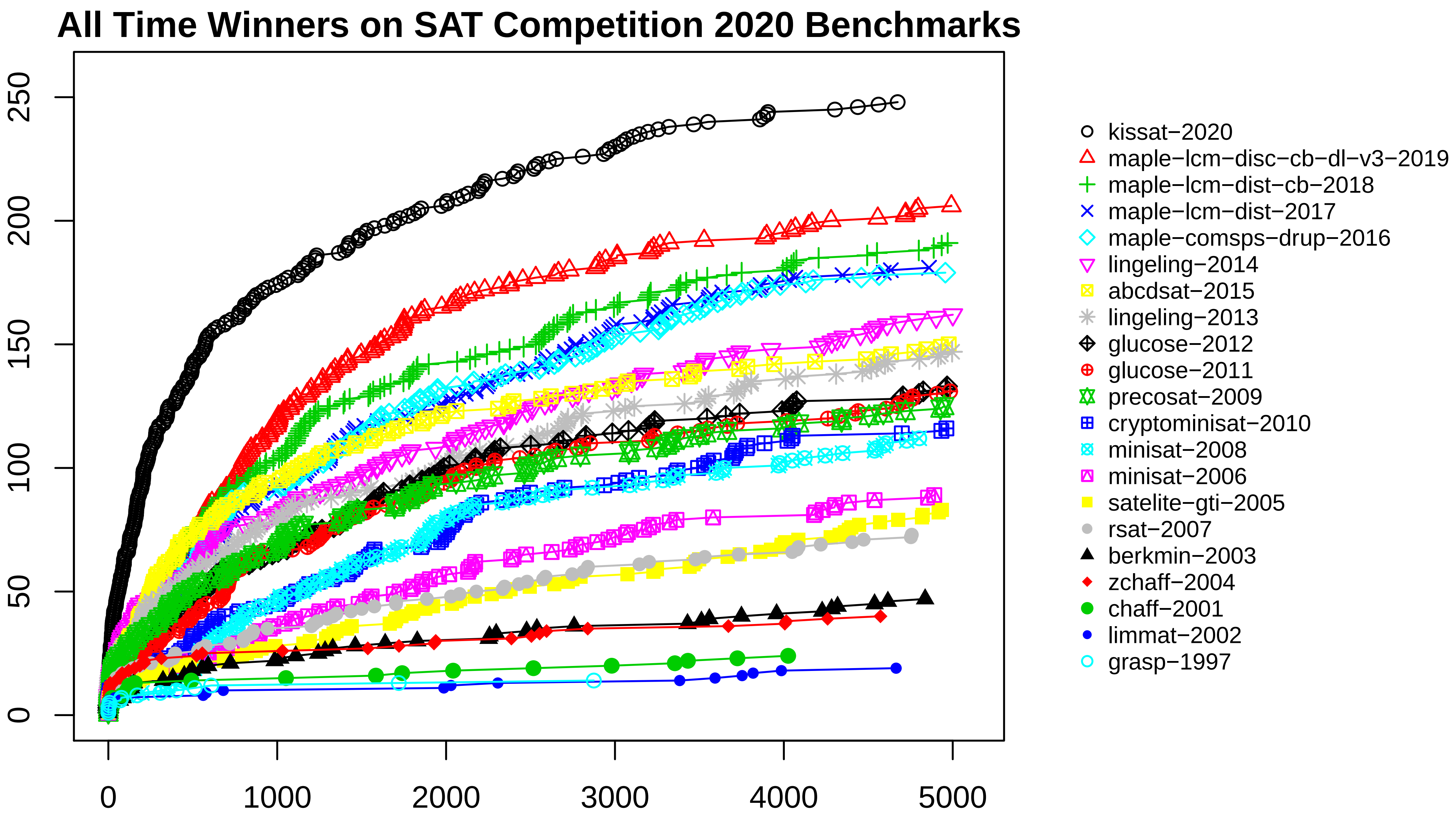
- clauses (conditions / constraints)

1. clearly one should not wear a **tie** without a **shirt** $\neg\textbf{tie} \vee \textbf{shirt}$
2. not wearing a **tie** nor a **shirt** is impolite $\textbf{tie} \vee \textbf{shirt}$
3. wearing a **tie** and a **shirt** is overkill $\neg(\textbf{tie} \wedge \textbf{shirt}) \equiv \neg\textbf{tie} \vee \neg\textbf{shirt}$

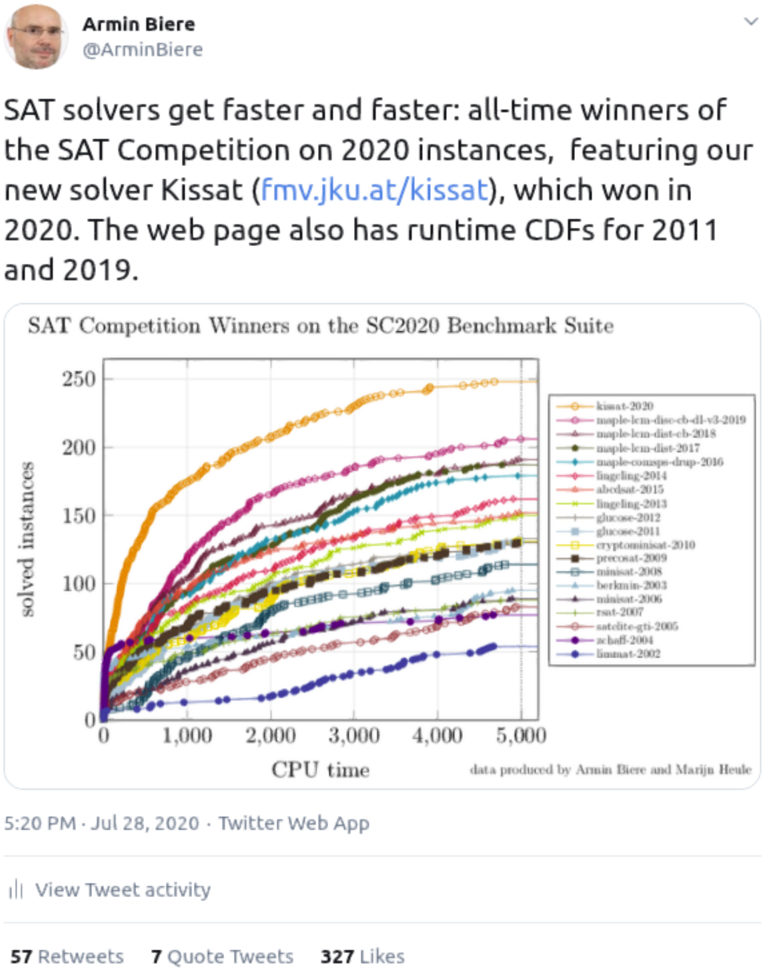
- Is this formula in conjunctive normal form (CNF) **satisfiable?**

$$(\neg\textbf{tie} \vee \textbf{shirt}) \wedge (\textbf{tie} \vee \textbf{shirt}) \wedge (\neg\textbf{tie} \vee \neg\textbf{shirt})$$

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
SAT
Competition 2007
Gold medal
Awarded to Picosat 535 solver
written by Armin Biere
for best performance by a solver in the industrial category, satisfiable benchmarks specialty.
The SAT2007 competition judges and organiz
Twelfth International Conference
On Theory and Applications of Satisfiability Testing
SAT
tition 2009
ld medal
solver
satisfiable and
Eighteenth International Conf
on Theory and Applications of Sa
Testing
SAT-Race 201
Parallel Track
3rd Prize
Awarded to:
Treengeling sr15bag
Written by:
Armin Biere
Eighteenth International Conference
on Theory and Applications of Satisfiability
Testing
SAT-Race 2015
Main Track
Special Prize:
Most Innovative Solver
Awarded to:
Lingeling sr15bal
Written by:
Armin Biere
SAT-Race 2015
2nd Prize
Plingeling sr15
Configurable SAT Solver Challenge 2014
SAT-Race 2010
Best Parallel Solver 2010
SAT Competition 2009
SAT COMPETITION 2013
2011
Twelfth International Conference
On Theory and Applications of Satisfiability
SAT
Competition 2009
Silver medal
Twelfth International Conference
Theory and Applications of Satisfiability
SAT
Competition 2009
Silver medal
Awarded to
written by
for best performance
benchmarks
precosat
International Symposium
Applications of Satisfiability Testing
Delft
University of Cincinnati
SAT Competition 2002
Winner
industrial benchmarks


All Time Winners on SAT Competition 2020 Benchmarks
0
50
100
150
200
250
0
1000
2000
3000
4000
5000
kissat-2020
maple-lcm-disc-cb-dl-v3-2019
maple-lcm-dist-cb-2018
maple-lcm-dist-2017
maple-comsps-drup-2016
lingeling-2014
abcdsat-2015
lingeling-2013
glucose-2012
glucose-2011
precosat-2009
cryptominisat-2010
minisat-2008
minisat-2006
satelite-gti-2005
rsat-2007
berkmin-2003
zchaff-2004
chaff-2001
limmat-2002
grasp-1997

# some recent Tweets

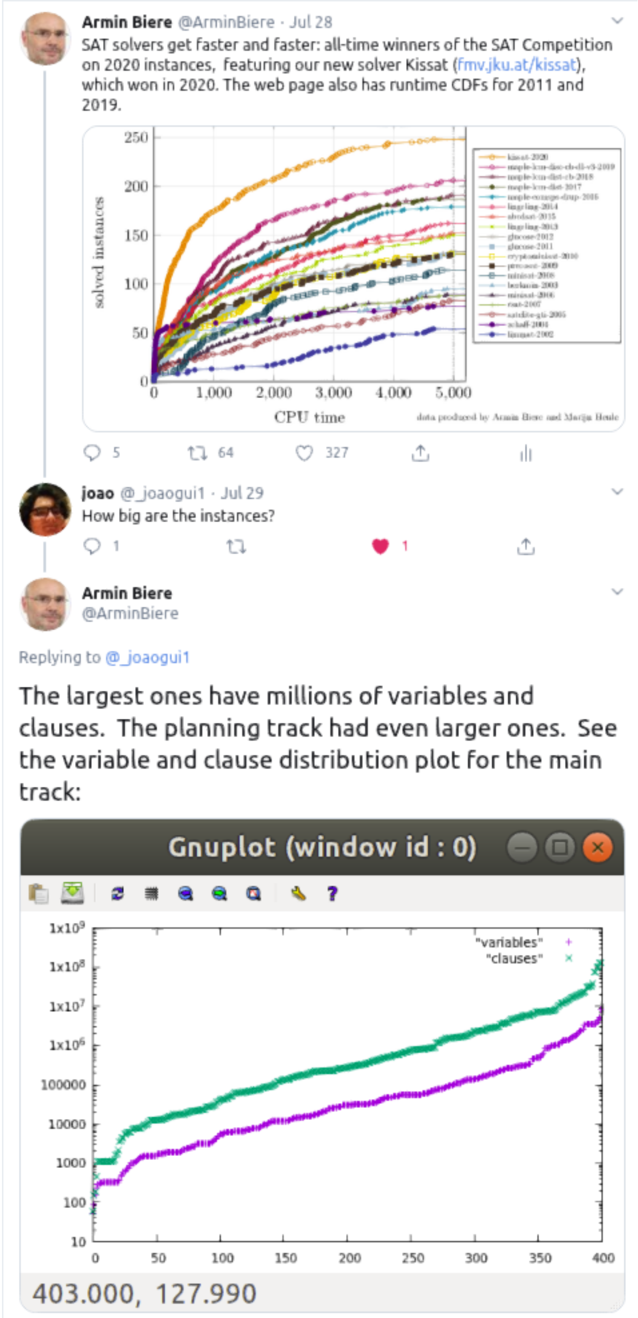


**Armin Biere**
@ArminBiere

SAT solvers get faster and faster: all-time winners of the SAT Competition on 2020 instances, featuring our new solver Kissat (fmv.jku.at/kissat), which won in 2020. The web page also has runtime CDFs for 2011 and 2019.

5:20 PM · Jul 28, 2020 · Twitter Web App

**57** Retweets **7** Quote Tweets **327** Likes



**Armin Biere** @ArminBiere · Jul 28

SAT solvers get faster and faster: all-time winners of the SAT Competition on 2020 instances, featuring our new solver Kissat (fmv.jku.at/kissat), which won in 2020. The web page also has runtime CDFs for 2011 and 2019.

**joao** @_joaogui1 · Jul 29

How big are the instances?

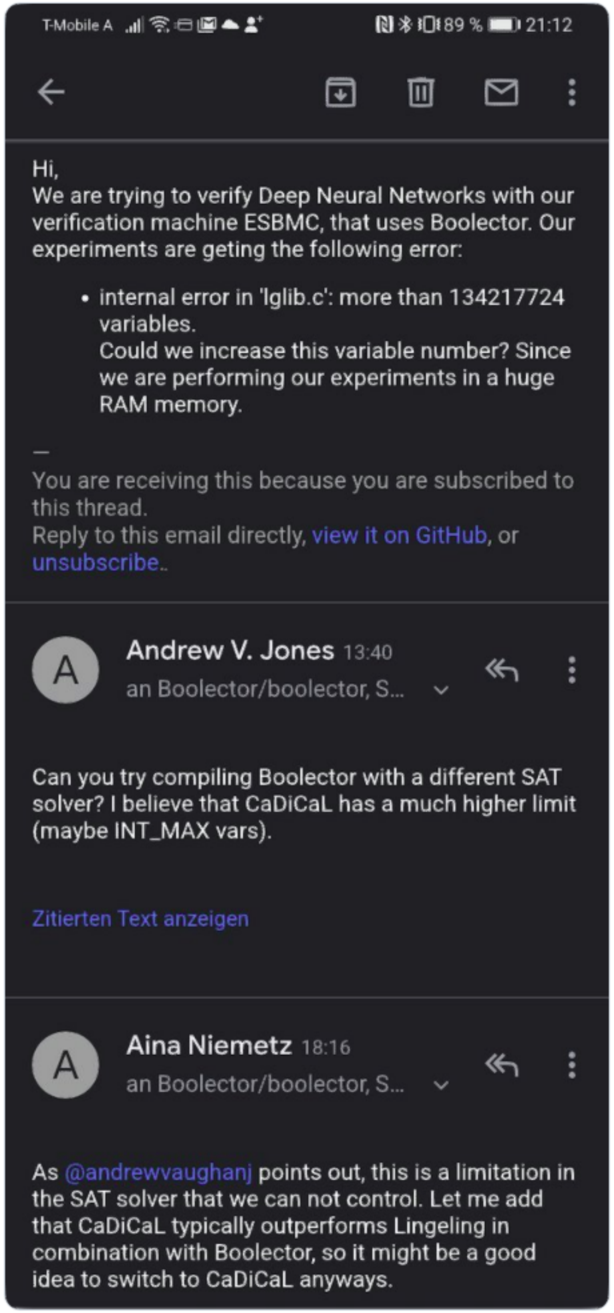
**Armin Biere**
@ArminBiere

Replying to @_joaogui1

The largest ones have millions of variables and clauses. The planning track had even larger ones. See the variable and clause distribution plot for the main track:

**Armin Biere**
@ArminBiere

Eventually I will need to support 64-bit variable indices (Lingeling has 2^27, CaDiCaL indeed 2^31 and Kissat 2^28 as compromise though it could easily do half a billion)



Hi,
We are trying to verify Deep Neural Networks with our verification machine ESBMC, that uses Boolector. Our experiments are geting the following error:

- internal error in 'lglib.c': more than 134217724 variables.
  Could we increase this variable number? Since we are performing our experiments in a huge RAM memory.

—
You are receiving this because you are subscribed to this thread.
Reply to this email directly, view it on GitHub, or unsubscribe.

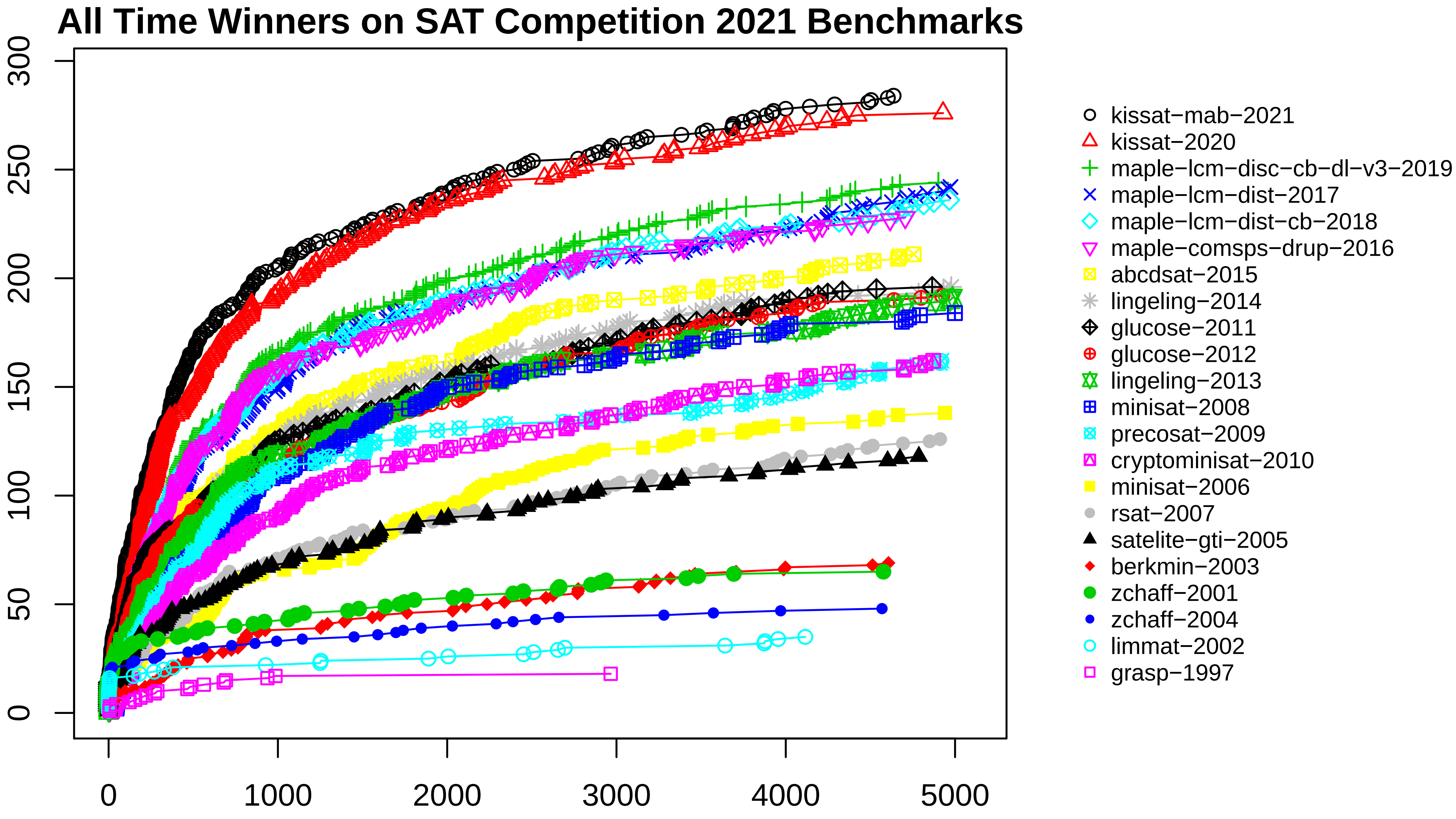
**Andrew V. Jones** 13:40
an Boolector/boolector, S...

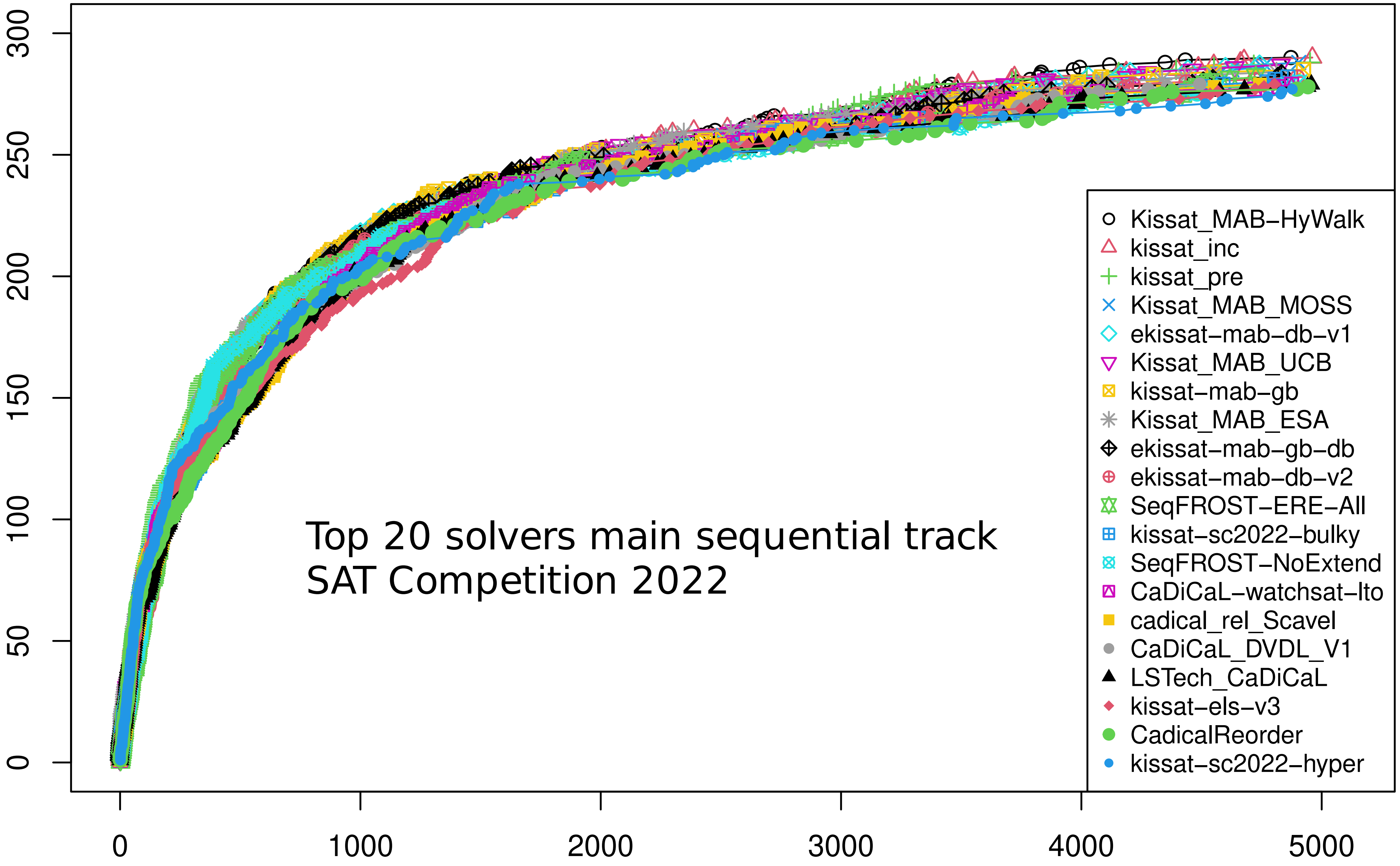
Can you try compiling Boolector with a different SAT solver? I believe that CaDiCaL has a much higher limit (maybe INT_MAX vars).

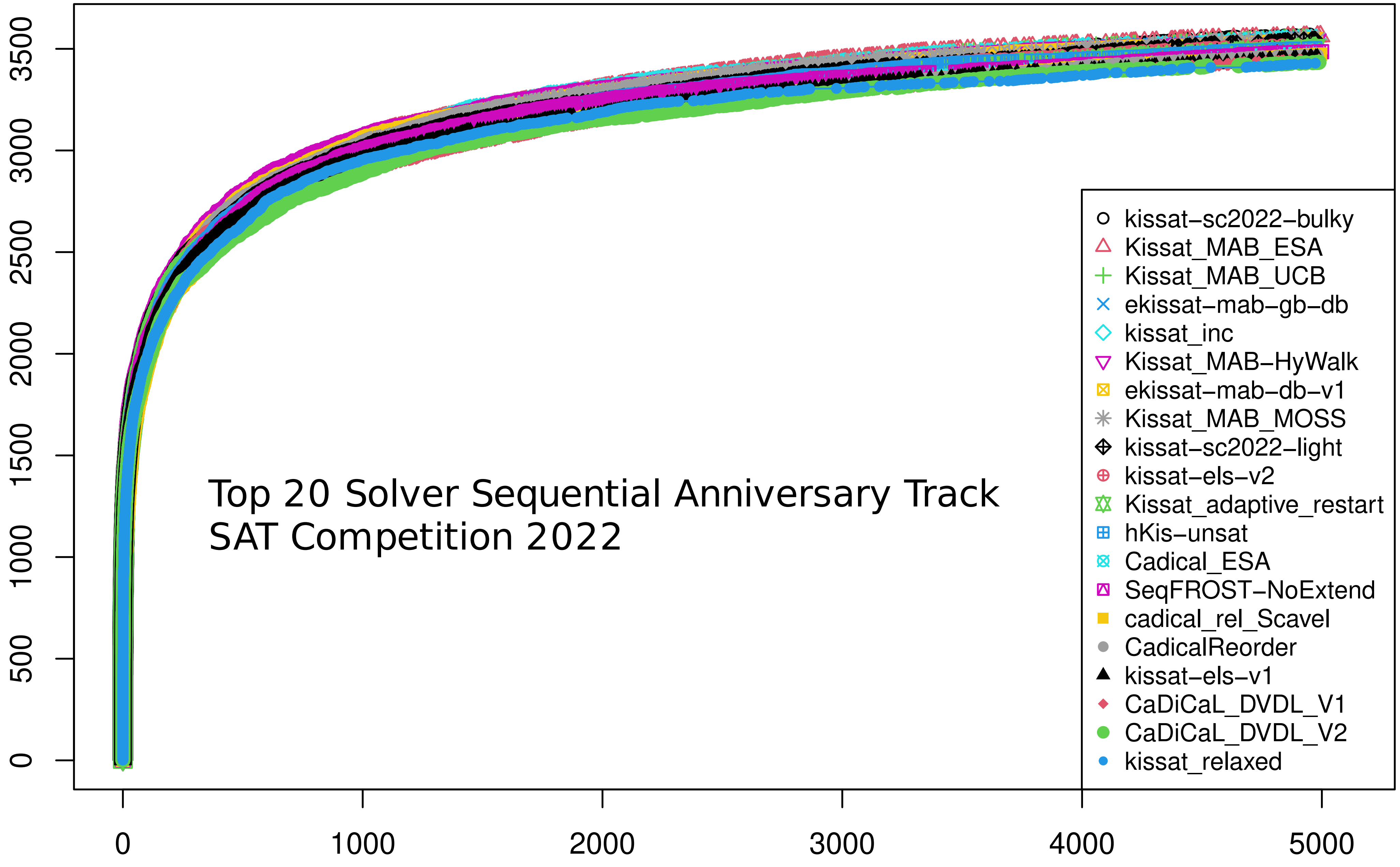
Zitierten Text anzeigen

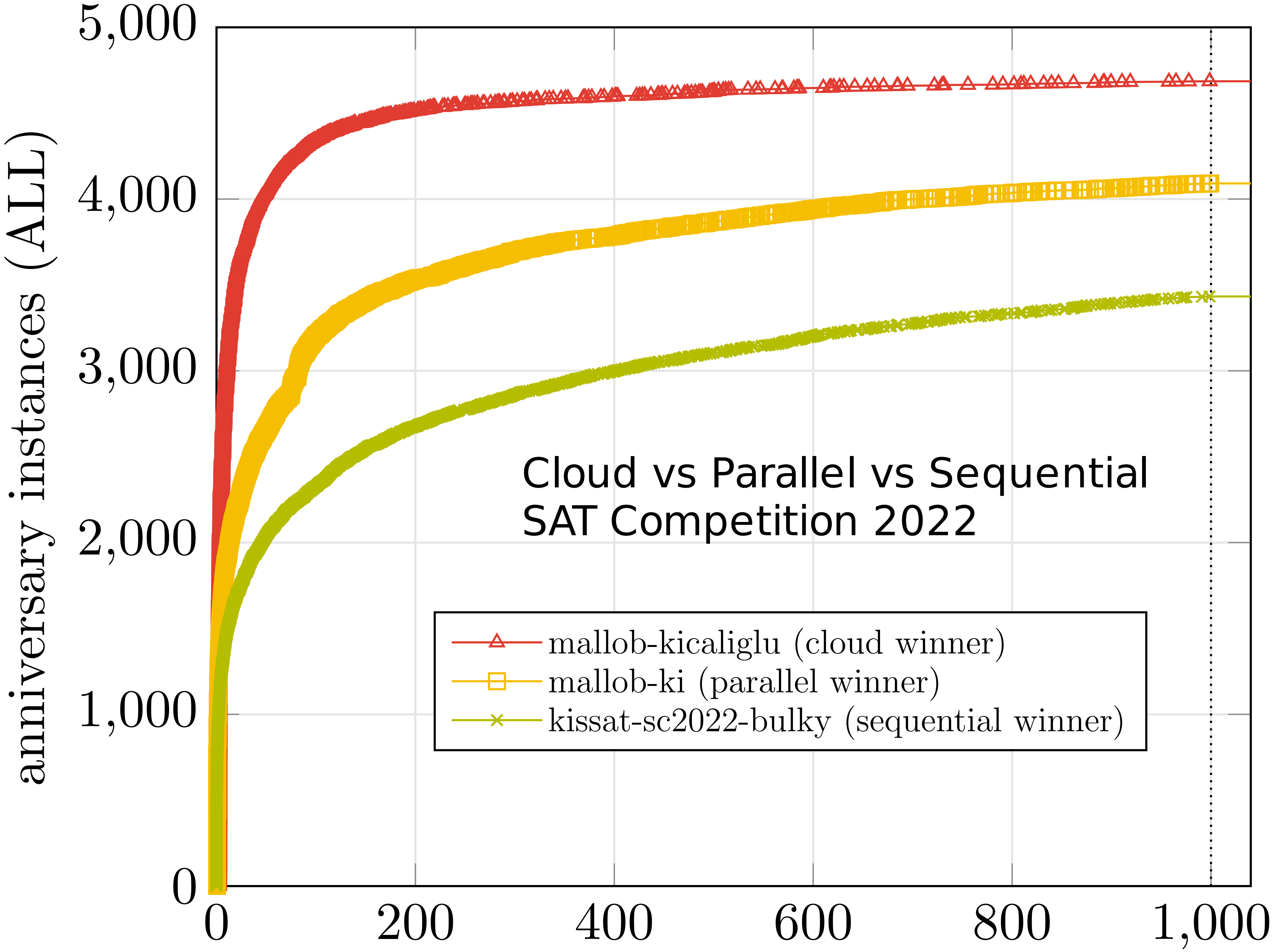
**Aina Niemetz** 18:16
an Boolector/boolector, S...

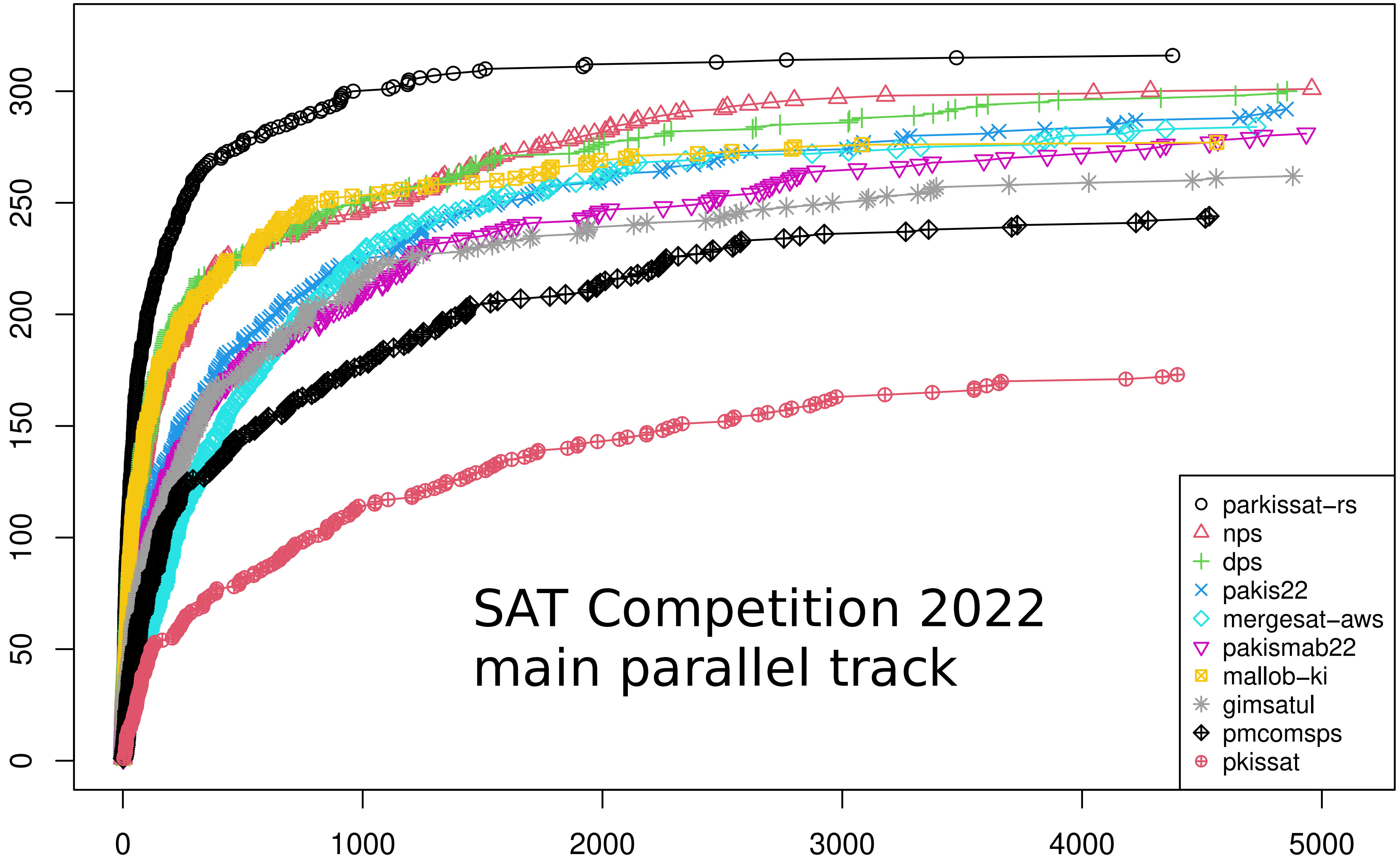
As @andrewvaughanj points out, this is a limitation in the SAT solver that we can not control. Let me add that CaDiCaL typically outperforms Lingeling in combination with Boolector, so it might be a good idea to switch to CaDiCaL anyways.


All Time Winners on SAT Competition 2021 Benchmarks
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
kissat−mab−2021
kissat−2020
maple−lcm−disc−cb−dl−v3−2019
maple−lcm−dist−2017
maple−lcm−dist−cb−2018
maple−comsps−drup−2016
abcdsat−2015
lingeling−2014
glucose−2011
glucose−2012
lingeling−2013
minisat−2008
precosat−2009
cryptominisat−2010
minisat−2006
rsat−2007
satelite−gti−2005
berkmin−2003
zchaff−2001
zchaff−2004
limmat−2002
grasp−1997


Top 20 solvers main sequential track
SAT Competition 2022
0
50
100
150
200
250
300
0
1000
2000
3000
4000
5000
○ Kissat_MAB-HyWalk
△ kissat_inc
+ kissat_pre
× Kissat_MAB_MOSS
◇ ekissat-mab-db-v1
▽ Kissat_MAB_UCB
☒ kissat-mab-gb
✳ Kissat_MAB_ESA
◈ ekissat-mab-gb-db
⊕ ekissat-mab-db-v2
✡ SeqFROST-ERE-All
⊞ kissat-sc2022-bulky
⊗ SeqFROST-NoExtend
⧅ CaDiCaL-watchsat-lto
■ cadical_rel_Scavel
● CaDiCaL_DVDL_V1
▲ LSTech_CaDiCaL
◆ kissat-els-v3
● CadicalReorder
● kissat-sc2022-hyper


Top 20 Solver Sequential Anniversary Track
SAT Competition 2022
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000
○ kissat−sc2022−bulky
△ Kissat_MAB_ESA
+ Kissat_MAB_UCB
× ekissat−mab−gb−db
◇ kissat_inc
▽ Kissat_MAB−HyWalk
⊠ ekissat−mab−db−v1
✳ Kissat_MAB_MOSS
◈ kissat−sc2022−light
⊕ kissat−els−v2
✡ Kissat_adaptive_restart
⊞ hKis−unsat
⊗ Cadical_ESA
⊠ SeqFROST−NoExtend
■ cadical_rel_Scavel
● CadicalReorder
▲ kissat−els−v1
◆ CaDiCaL_DVDL_V1
● CaDiCaL_DVDL_V2
• kissat_relaxed


5,000
4,000
3,000
2,000
1,000
0
0
200
400
600
800
1,000
anniversary instances (ALL)
Cloud vs Parallel vs Sequential
SAT Competition 2022
mallob-kicaliglu (cloud winner)
mallob-ki (parallel winner)
kissat-sc2022-bulky (sequential winner)


300
250
200
150
100
50
0
0
1000
2000
3000
4000
5000
○ parkissat−rs
△ nps
+ dps
× pakis22
◇ mergesat−aws
▽ pakismab22
☒ mallob−ki
✳ gimsatul
◇ pmcomsps
⊕ pkissat
SAT Competition 2022
main parallel track

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

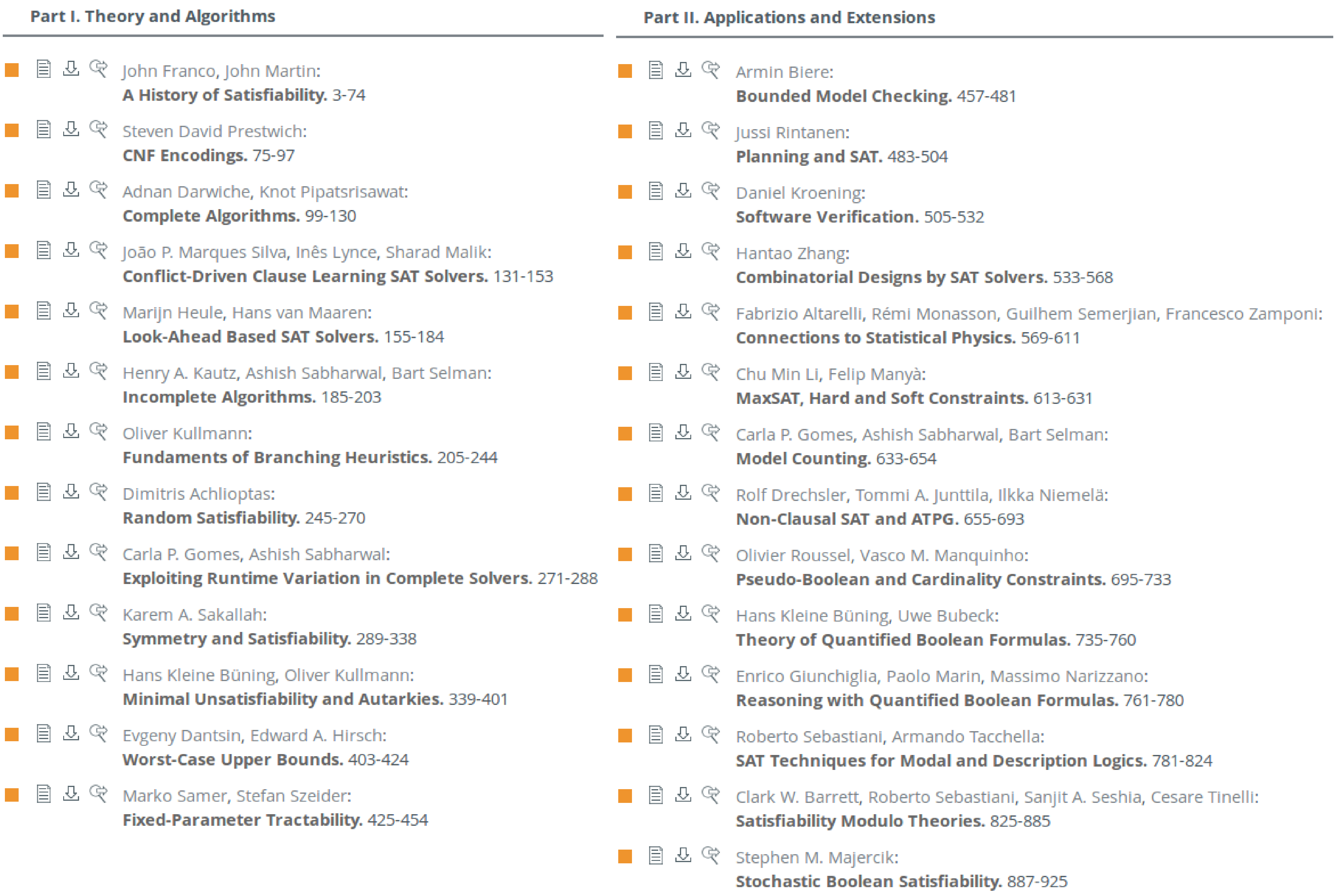
IOS Press

Frontiers in Artificial Intelligence and Applications

# HANDBOOK of satisfiability

Editors:
Armin Biere
Marijn Heule
Hans van Maaren
Toby Walsh

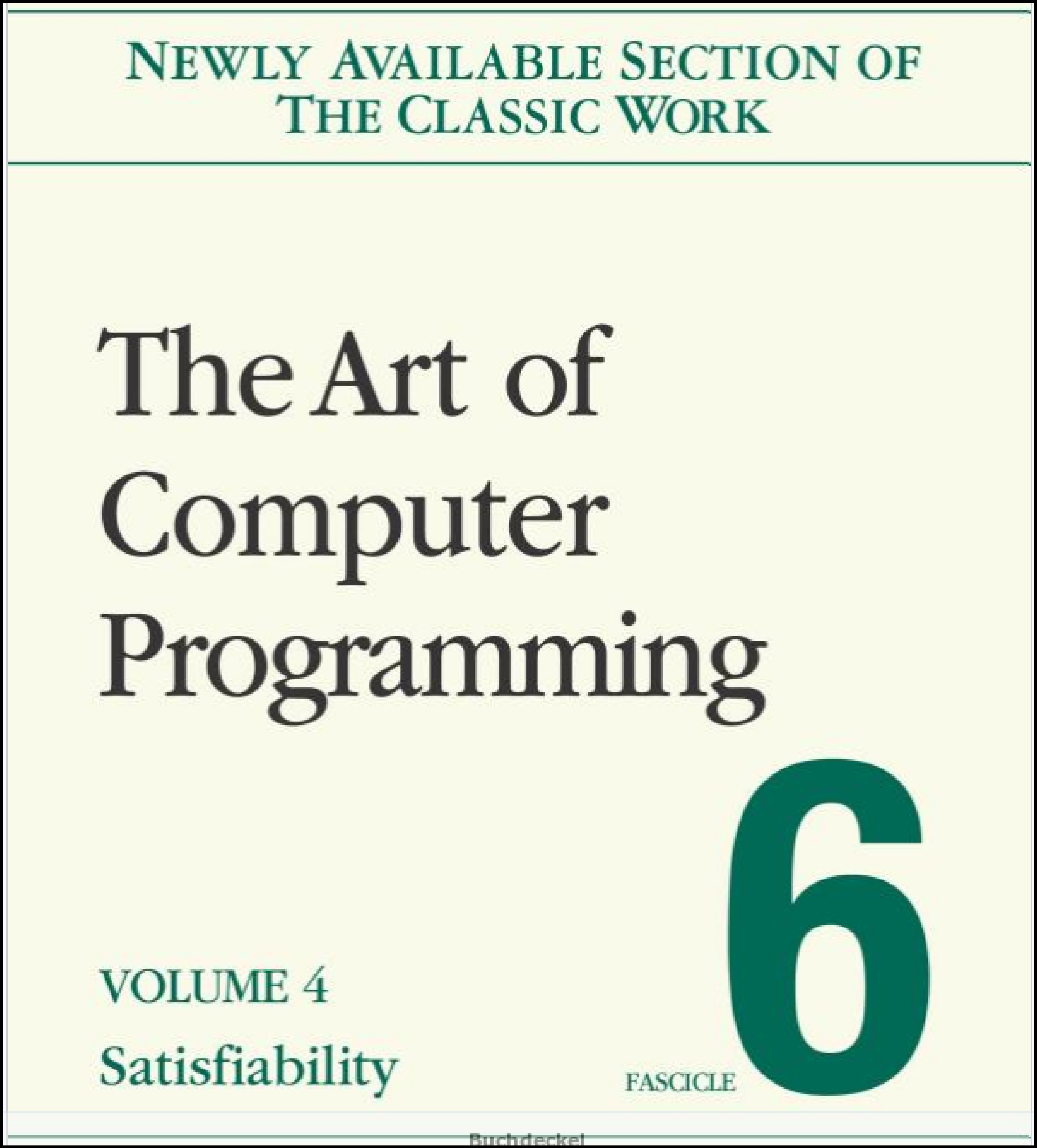
IOS Press



## Part I. Theory and Algorithms

## Part II. Applications and Extensions

P
9-8-06



NEWLY AVAILABLE SECTION OF THE CLASSIC WORK

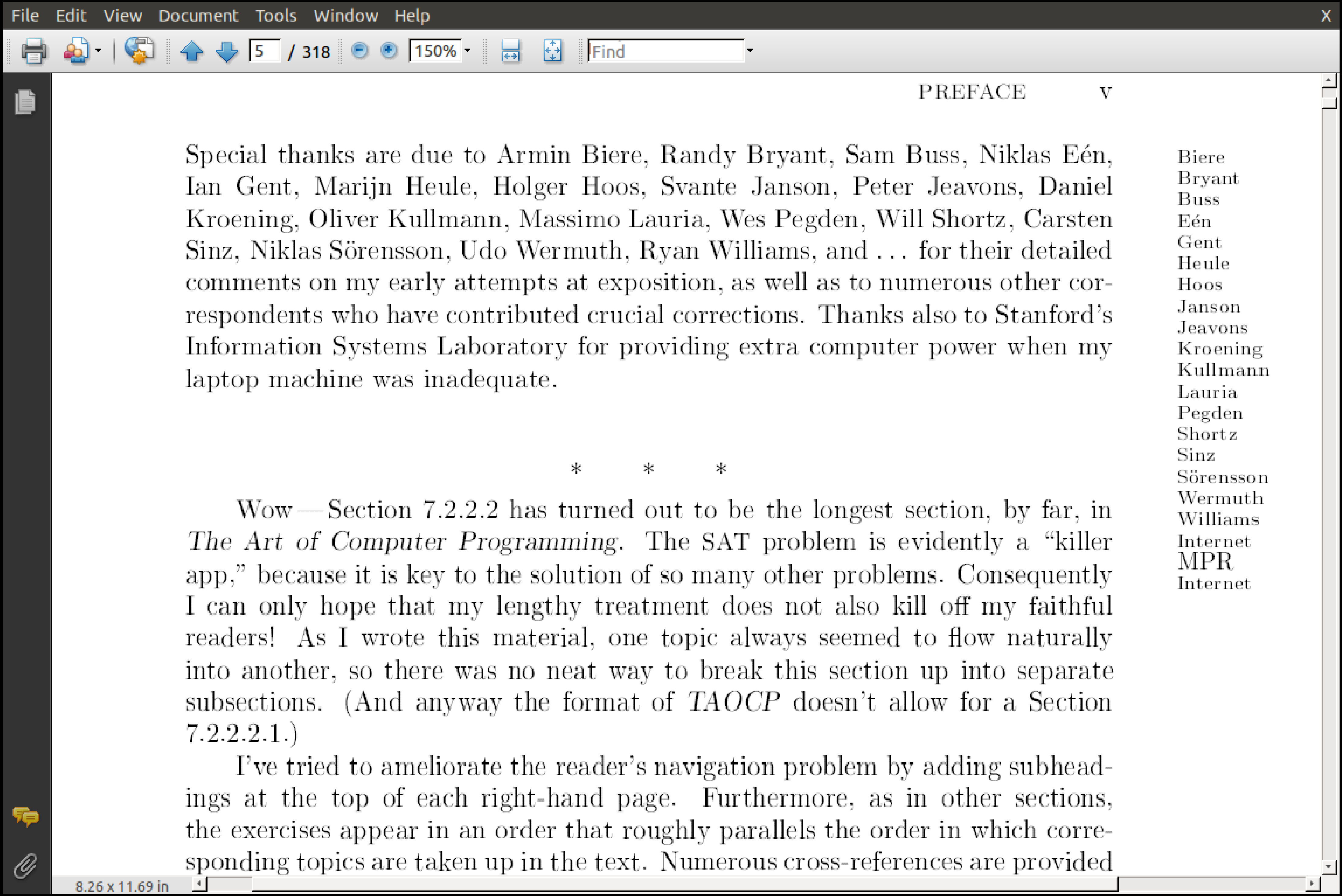
# The Art of Computer Programming

VOLUME 4

Satisfiability

FASCICLE 6

Buchdeckel



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.)

I've tried to ameliorate the reader's navigation problem by adding subheadings at the top of each right-hand page. Furthermore, as in other sections, the exercises appear in an order that roughly parallels the order in which corresponding topics are taken up in the text. Numerous cross-references are provided

# SAT Handbook 2nd Edition (2021)

*editors* Armin Biere, Marijn Heule, Hans van Maaren, Toby Walsh

*with many updated chapters and the **following 7 new chapters**:*

Proof Complexity Jakob Nordström and Sam Buss

Preprocessing Armin Biere, Matti Järvisalo and Benjamin Kiesl

Tuning and Configuration

Holger Hoos, Frank Hutter and Kevin Leyton-Brown

Proofs of Unsatisfiability Marijn Heule

Core-Based MaxSAT

Fahiem Bacchus, Matti Järvisalo and Ruben Martins

Proof Systems for Quantified Boolean Formulas

Olaf Beyersdorff, Mikoláš Janota, Florian Lonsing and Martina Seidl

Approximate Model Counting Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi

SAT solving is a key technology for 21st entury computer science.

*Edmund Clarke*

*2007 ACM Turing Award Recipient*

The SAT problem is evidently a killer app, because it is key to the solution of so many other problems.

*Donald Knuth*

*1974 ACM Turing Award Recipient*

The SAT problem is at the core of arguably the most fundamental question in computer science: What makes a problem hard?

*Stephen Cook*

*1982 ACM Turing Award Recipient*

Frontiers in Artificial Intelligence and Applications

# HANDBOOK of satisfiability

IOS Press

IOS Press Book

www.iospress.com

**February 2021**
**Approx. 1516 pp.**

**Print book**
Hardcover, in 2 parts
ISBN: 978-1-64368-160-3 (print)
€200 / US$250 / £180 excl. VAT

**Ebook**
ISBN: 978-1-64368-161-0 (online)
€200 / US$250 / £180 excl. VAT

**Discount Code**
Order your print book before April 15, 2021 and get 35% off!
Code: **SAT2021**

–35%

For more information and ordering check
**tiny.cc/SAT2021**

## Handbook of Satisfiability

Second Edition

**Editors: A. Biere, M. Heule, H. van Maaren, T. Walsh**
**Volume 336 of Frontiers in Artificial Intelligence and Applications**

Propositional logic has been recognized throughout the centuries as one of the cornerstones of reasoning in philosophy and mathematics. Over time, its formalization into Boolean algebra was accompanied by the recognition that a wide range of combinatorial problems can be expressed as propositional satisfiability (SAT) problems. Because of this dual role, SAT developed into a mature, multi-faceted scientific discipline, and from the earliest days of computing a search was underway to discover how to solve SAT problems in an automated fashion.

This book, the *Handbook of Satisfiability*, is the second, updated and revised edition of the book first published in 2009 under the same name. The handbook aims to capture the full breadth and depth of SAT and to bring together significant progress and advances in automated solving. Topics covered span practical and theoretical research on SAT and its applications and include search algorithms, heuristics, analysis of algorithms, hard instances, randomized formulae, problem encodings, industrial applications, solvers, simplifiers, tools, case studies and empirical results. SAT is interpreted in a broad sense, so as well as propositional satisfiability, there are chapters covering the domain of quantified Boolean formulae (QBF), constraints programming techniques (CSP) for word-level problems and their propositional encoding, and satisfiability modulo theories (SMT). An extensive bibliography completes each chapter.

This second edition of the handbook will be of interest to researchers, graduate students, final-year undergraduates, and practitioners using or contributing to SAT, and will provide both an inspiration and a rich resource for their work.

**Edmund Clarke**, 2007 ACM Turing Award Recipient: "SAT solving is a key technology for 21st century computer science."

**Donald Knuth**, 1974 ACM Turing Award Recipient: "SAT is evidently a killer app, because it is key to the solution of so many other problems."

**Stephen Cook**, 1982 ACM Turing Award Recipient: "The SAT problem is at the core of arguably the most fundamental question in computer science: What makes a problem hard?"

**IOS Press**
Nieuwe Hemweg 6B
1013 BG Amsterdam
The Netherlands
Tel.: +31 20 688 3355
Email: order@iospress.nl

Visit our website www.iospress.com for online ordering
For ebooks, go to www.ebooks.iospress.com
Follow us on Twitter: @IOSPress_STM
Follow us on Facebook: publisherspress

**IOS Press, Inc.**
6751 Tepper Drive
Clifton, VA 20124
USA
Tel.: +1 703 830 6300
Email: sales@iospress.com

## Contents

**Part I. Theory and Algorithms**

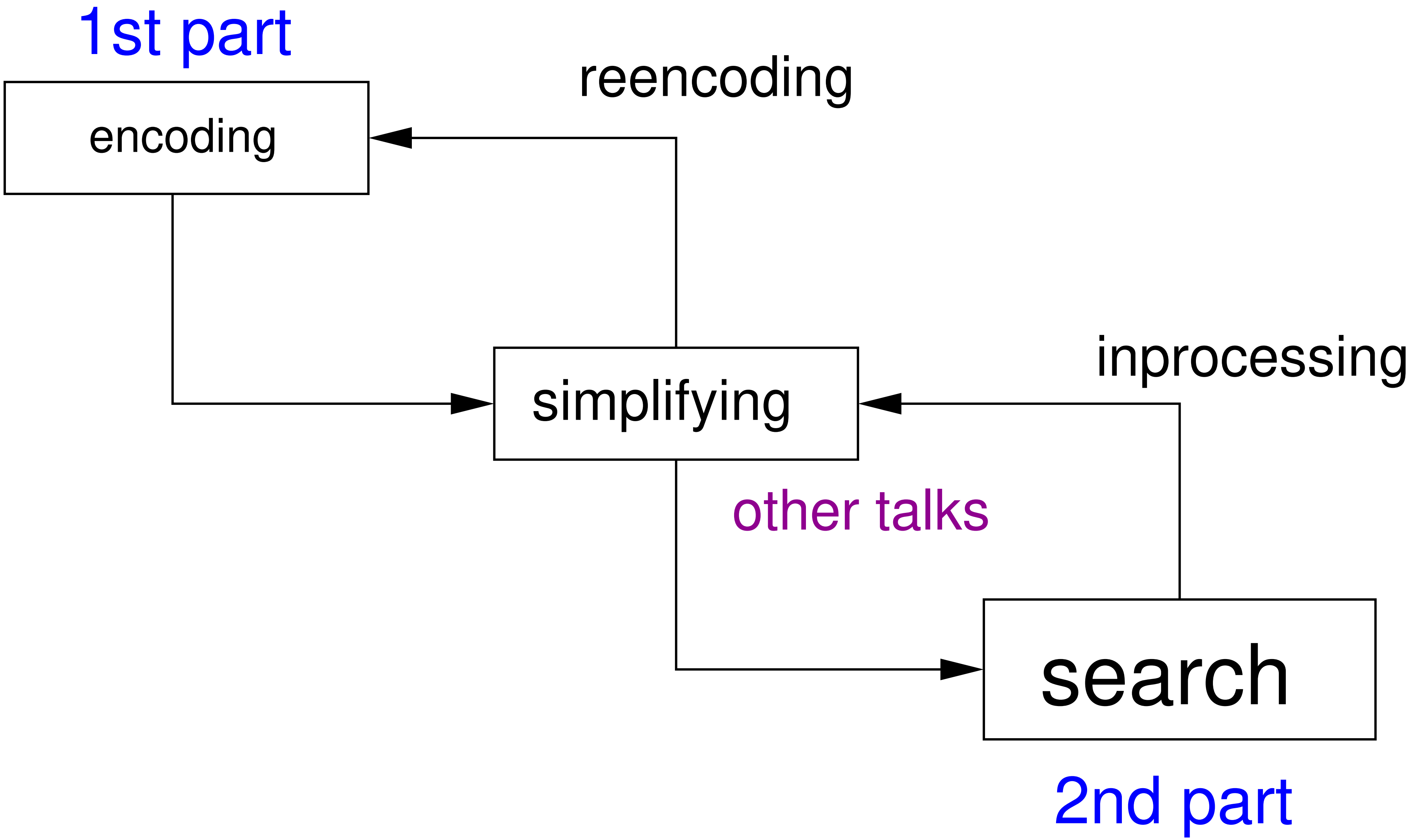
**Part II. Applications and Extensions**

**IOS Press**
Nieuwe Hemweg 6B
1013 BG Amsterdam
The Netherlands
Tel.: +31 20 688 3355
Email: order@iospress.nl

Visit our website www.iospress.com for online ordering
For ebooks, go to www.ebooks.iospress.com
Follow us on Twitter: @IOSPress_STM
Follow us on Facebook: publisherspress

**IOS Press, Inc.**
6751 Tepper Drive
Clifton, VA 20124
USA
Tel.: +1 703 830 6300
Email: sales@iospress.com

# What is Practical SAT Solving?



# Equivalence Checking If-Then-Else Chains

**original C code**

```
if(!a && !b) h();
else if(!a) g();
else f();
```

⇓

```
if(!a) {
  if(!b) h();
  else g();
} else f();
```

⇒

**optimized C code**

```
if(a) f();
else if(b) g();
else h();
```

⇑

```
if(a) f();
else {
  if(!b) h();
  else g(); }
```

How to check that these two versions are equivalent?

# Compilation

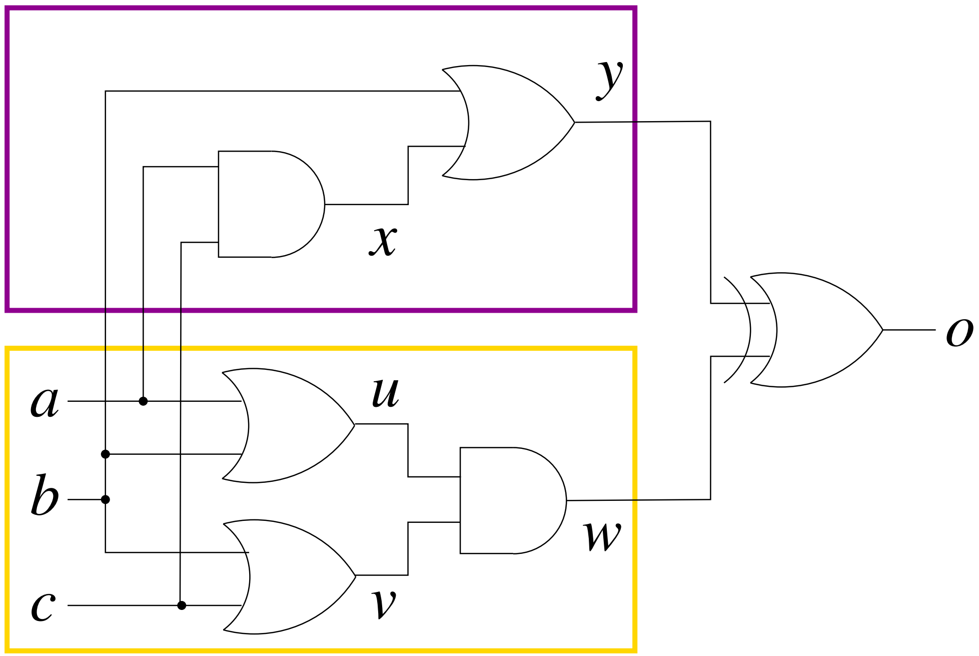
$$
\begin{aligned}
\mathit{original} &\equiv \textbf{if } \neg a \wedge \neg b \textbf{ then } h \textbf{ else } \textbf{ if } \neg a \textbf{ then } g \textbf{ else } f \\
&\equiv (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge \textbf{ if } \neg a \textbf{ then } g \textbf{ else } f \\
&\equiv (\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \;\vee\; a \wedge f)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{optimized} &\equiv \textbf{if } a \textbf{ then } f \textbf{ else } \textbf{ if } b \textbf{ then } g \textbf{ else } h \\
&\equiv a \wedge f \;\vee\; \neg a \wedge \textbf{ if } b \textbf{ then } g \textbf{ else } h \\
&\equiv a \wedge f \;\vee\; \neg a \wedge (b \wedge g \;\vee\; \neg b \wedge h)
\end{aligned}
$$

$$
(\neg a \wedge \neg b) \wedge h \;\vee\; \neg(\neg a \wedge \neg b) \wedge (\neg a \wedge g \;\vee\; a \wedge f) \quad \not\Leftrightarrow \quad a \wedge f \;\vee\; \neg a \wedge (b \wedge g \;\vee\; \neg b \wedge h)
$$

satisfying assignment gives counter-example to equivalence

# Tseitin Transformation: Circuit to CNF



$$
\begin{array}{r}
o \;\wedge \\
(x \leftrightarrow a \wedge c) \;\wedge \\
(y \leftrightarrow b \vee x) \;\wedge \\
(u \leftrightarrow a \vee b) \;\wedge \\
(v \leftrightarrow b \vee c) \;\wedge \\
(w \leftrightarrow u \wedge v) \;\wedge \\
(o \leftrightarrow y \oplus w)\;\;\;
\end{array}
$$

$$o \wedge (x \rightarrow a) \wedge (x \rightarrow c) \wedge (x \leftarrow a \wedge c) \wedge \ \ldots$$

$$\boxed{o \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee c) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge \ \ldots}$$

# Tseitin Transformation: Gate Constraints

Negation:

$$\begin{aligned} x \leftrightarrow \overline{y} &\Leftrightarrow (x \rightarrow \overline{y}) \wedge (\overline{y} \rightarrow x) \\ &\Leftrightarrow (\overline{x} \vee \overline{y}) \wedge (y \vee x) \end{aligned}$$

Disjunction:

$$\begin{aligned} x \leftrightarrow (y \vee z) &\Leftrightarrow (y \rightarrow x) \wedge (z \rightarrow x) \wedge (x \rightarrow (y \vee z)) \\ &\Leftrightarrow (\overline{y} \vee x) \wedge (\overline{z} \vee x) \wedge (\overline{x} \vee y \vee z) \end{aligned}$$

Conjunction:

$$\begin{aligned} x \leftrightarrow (y \wedge z) &\Leftrightarrow (x \rightarrow y) \wedge (x \rightarrow z) \wedge ((y \wedge z) \rightarrow x) \\ &\Leftrightarrow (\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{(y \wedge z)} \vee x) \\ &\Leftrightarrow (\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z} \vee x) \end{aligned}$$

Equivalence:

$$\begin{aligned} x \leftrightarrow (y \leftrightarrow z) &\Leftrightarrow (x \rightarrow (y \leftrightarrow z)) \wedge ((y \leftrightarrow z) \rightarrow x) \\ &\Leftrightarrow (x \rightarrow ((y \rightarrow z) \wedge (z \rightarrow y)) \wedge ((y \leftrightarrow z) \rightarrow x) \\ &\Leftrightarrow (x \rightarrow (y \rightarrow z)) \wedge (x \rightarrow (z \rightarrow y)) \wedge ((y \leftrightarrow z) \rightarrow x) \\ &\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge ((y \leftrightarrow z) \rightarrow x) \\ &\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge (((y \wedge z) \vee (\overline{y} \wedge \overline{z})) \rightarrow x) \\ &\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge ((y \wedge z) \rightarrow x) \wedge ((\overline{y} \wedge \overline{z}) \rightarrow x) \\ &\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge (\overline{y} \vee \overline{z} \vee x) \wedge (y \vee z \vee x) \end{aligned}$$

# Bit-Blasting of Bit-Vector Addition

addition of 4-bit numbers $x, y$ with result $s$ also 4-bit: $s = x + y$

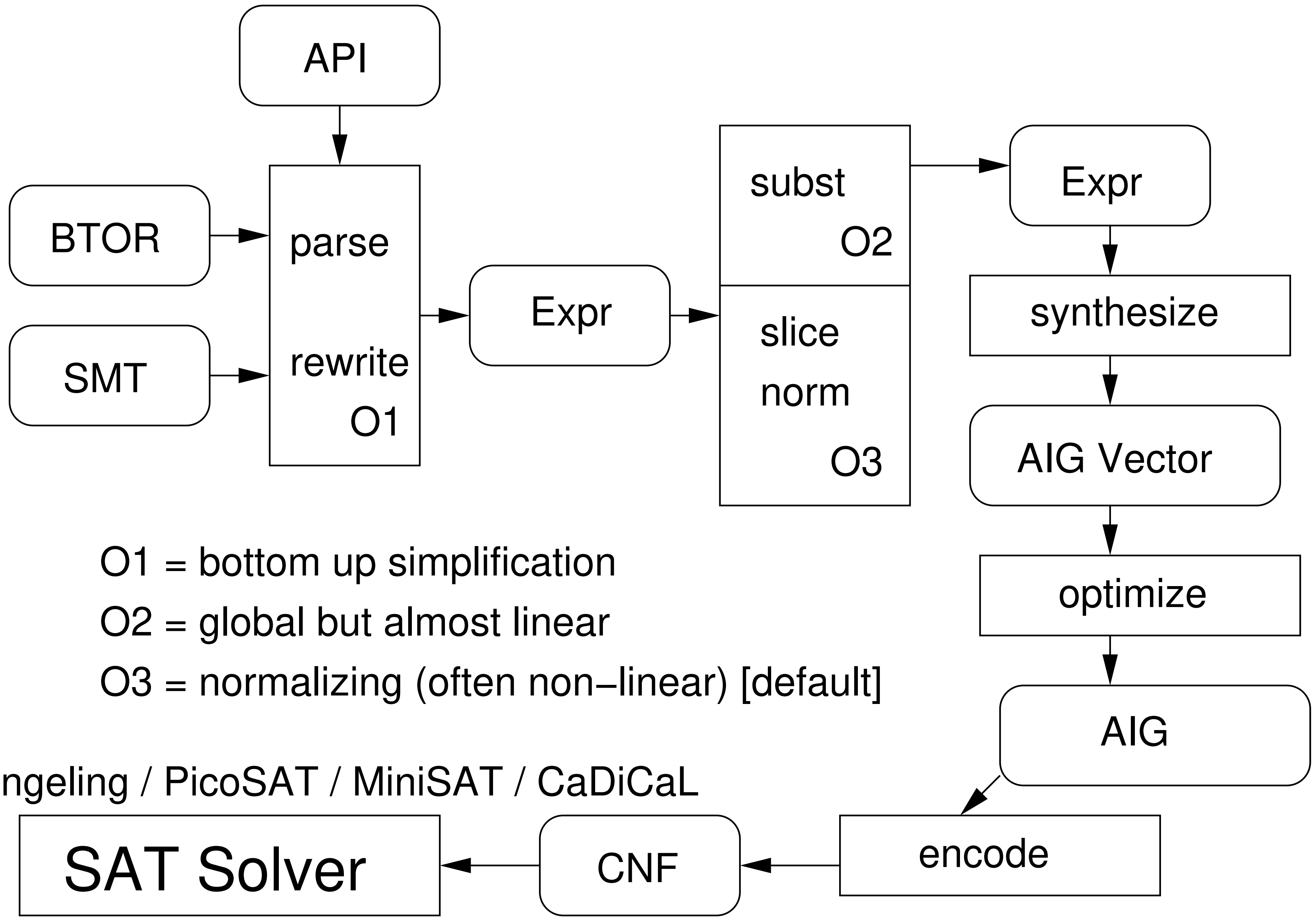
$$[s_3, s_2, s_1, s_0]_4 \ = \ [x_3, x_2, x_1, x_0]_4 + [y_3, y_2, y_1, y_0]_4$$

$$\begin{array}{rcl}
[s_3, \cdot\,]_2 & = & \mathsf{FullAdder}(x_3, y_3, c_2) \\
[s_2, c_2]_2 & = & \mathsf{FullAdder}(x_2, y_2, c_1) \\
[s_1, c_1]_2 & = & \mathsf{FullAdder}(x_1, y_1, c_0) \\
[s_0, c_0]_2 & = & \mathsf{FullAdder}(x_0, y_0, \mathit{false})
\end{array}$$

where

$$\begin{array}{rcl}
[\, s \,, o \,]_2 & = & \mathsf{FullAdder}(x, y, i) \quad \text{with} \\
s & = & x \text{ xor } y \text{ xor } i \\
o & = & (x \wedge y) \vee (x \wedge i) \vee (y \wedge i) \ = \ ((x + y + i) \geq 2)
\end{array}$$

# Boolector Architecture



API

BTOR

SMT

parse

rewrite

O1

Expr

subst

O2

slice

norm

O3

Expr

synthesize

AIG Vector

optimize

AIG

encode

CNF

Lingeling / PicoSAT / MiniSAT / CaDiCaL

SAT Solver

O1 = bottom up simplification

O2 = global but almost linear

O3 = normalizing (often non−linear) [default]

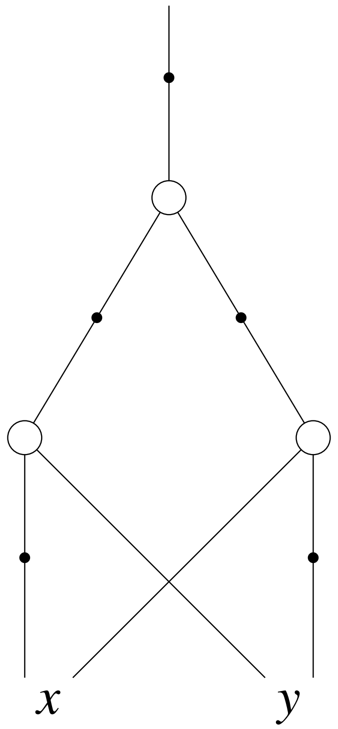
# Intermediate Representations

- encoding directly into CNF is hard, so we use intermediate levels:

  1. application level
  2. bit-precise semantics world-level operations (bit-vectors)
  3. bit-level representations such as And-Inverter Graphs (AIGs)
  4. conjunctive normal form (CNF)

- encoding “logical” constraints is another story

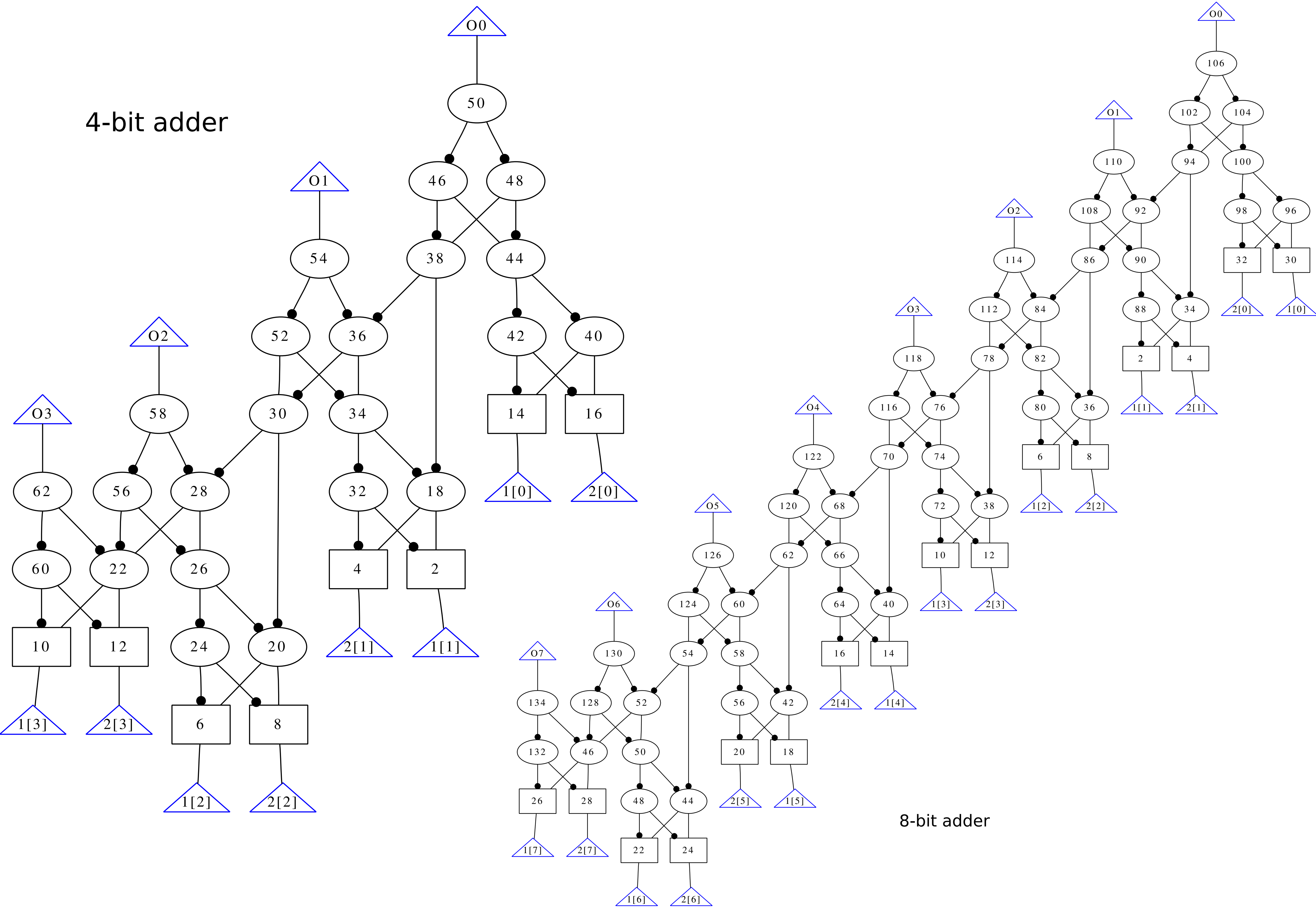
# XOR as AIG



negation/sign are edge attributes

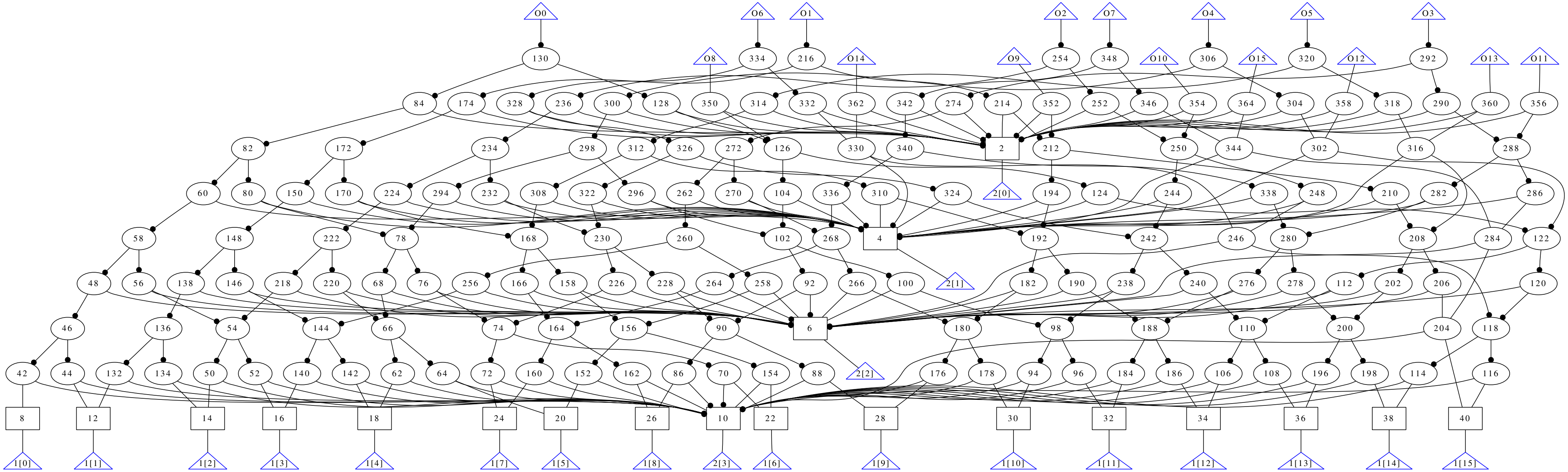
not part of node

$$x \text{ xor } y \;\equiv\; (\overline{x} \wedge y) \vee (x \wedge \overline{y}) \;\equiv\; \overline{\overline{(\overline{x} \wedge y)} \wedge \overline{(x \wedge \overline{y})}}$$

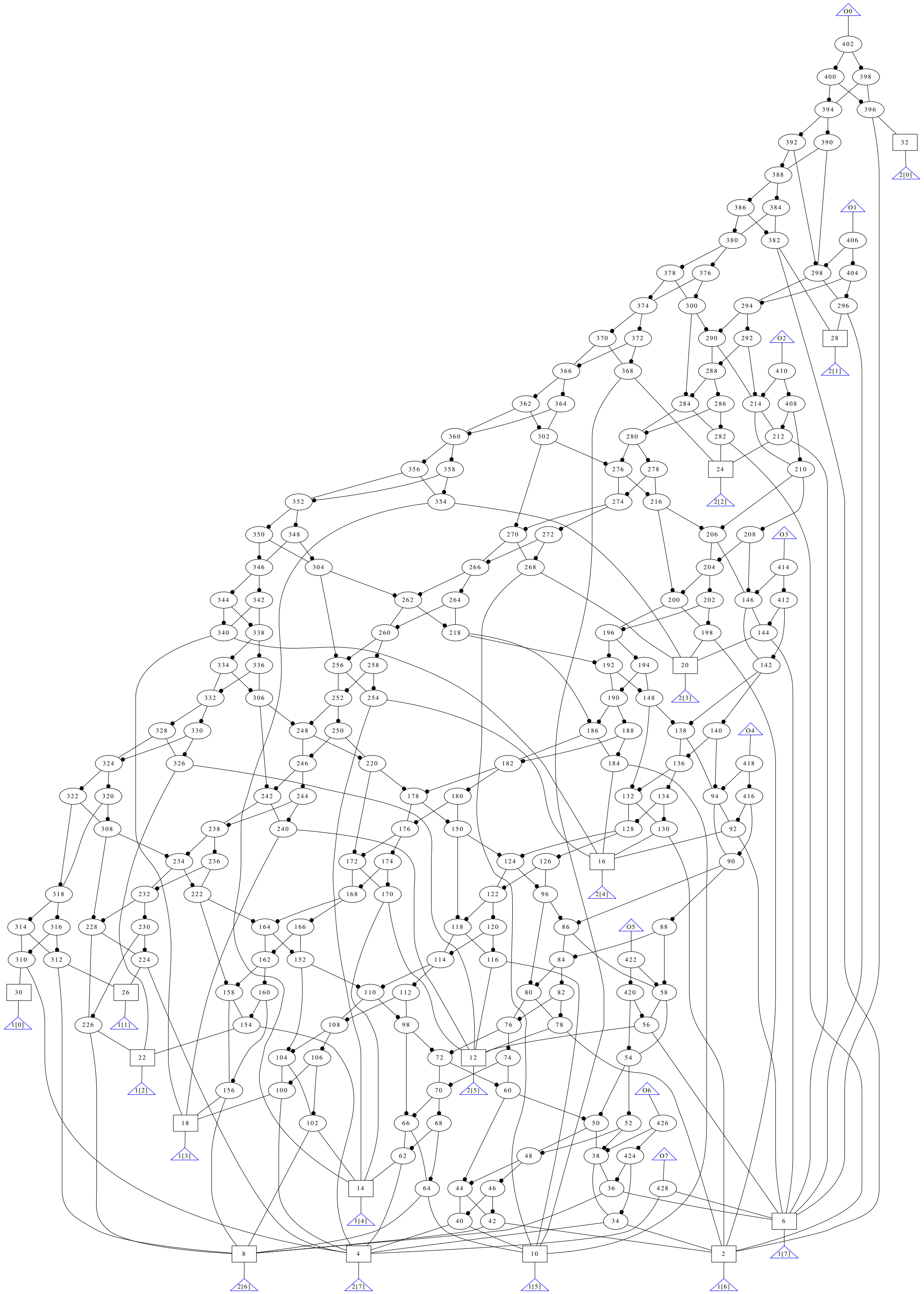


4-bit adder

8-bit adder



bit-vector of length 16 shifted by bit-vector of length 4


O0
402
400
398
394
396
392
390
32
388
2[0]
386
384
O1
380
382
406
378
376
298
404
374
300
294
296
370
372
290
292
O2
28
366
368
288
410
2[1]
362
364
284
286
214
408
360
302
280
282
212
356
358
276
278
24
210
352
354
274
216
2[2]
350
348
270
272
206
208
O3
346
304
266
268
204
414
344
342
262
264
200
202
146
412
340
338
260
218
196
198
144
334
336
256
258
192
194
20
142
332
306
252
254
190
148
2[3]
328
330
248
250
186
188
138
140
O4
324
326
246
220
182
184
136
418
322
320
242
244
178
180
132
134
94
416
308
238
240
176
150
128
130
92
234
236
172
174
124
126
16
90
318
232
222
168
170
122
96
2[4]
314
316
228
230
164
166
118
120
86
O5
88
310
312
224
162
152
114
116
84
422
30
26
158
160
110
112
80
82
420
58
1[0]
1[1]
226
154
108
98
76
78
56
22
104
106
72
12
74
54
1[2]
156
100
70
2[5]
60
O6
18
102
66
68
50
52
426
1[3]
62
48
38
424
O7
14
44
46
36
428
64
1[4]
40
42
34
6
8
4
10
2
1[7]
2[6]
2[7]
1[5]
1[6]

# Encoding Logical Constraints

- Tseitin construction suitable for most kinds of “model constraints”
  - assuming simple operational semantics: encode an interpreter
  - small domains: one-hot encoding large domains: binary encoding

- harder to encode properties or additional constraints
  - temporal logic / fix-points
  - environment constraints

- example for fix-points / recursive equations: $x = (a \vee y), \quad y = (b \vee x)$
  - has unique least fix-point $x = y = (a \vee b)$
  - and unique largest fix-point $x = y = \mathit{true}$ but unfortunately …
  - … only largest fix-point can be (directly) encoded in SAT
    otherwise need stable models / logical programming / ASP

# Example of Logical Constraints: Cardinality Constraints

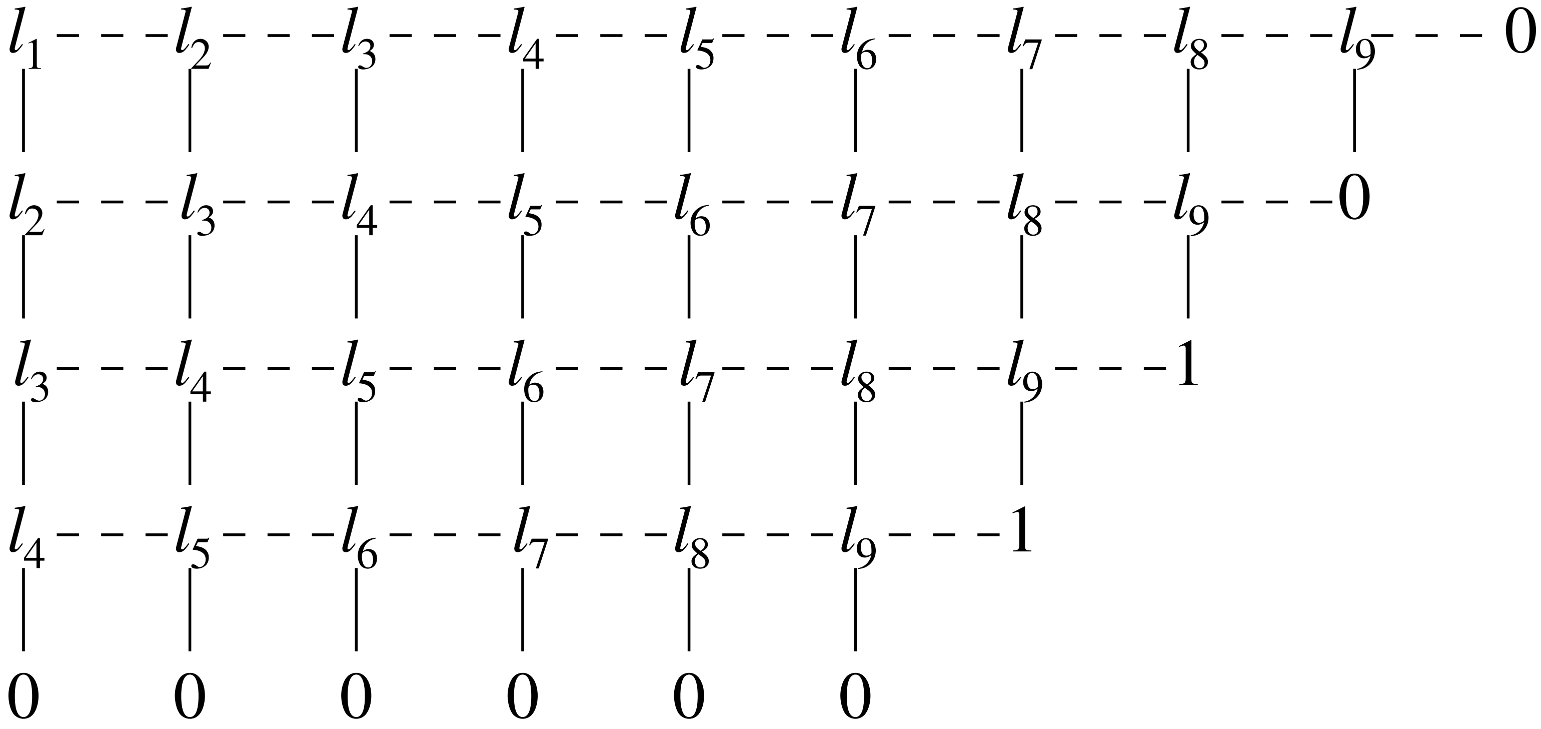
- given a set of literals $\{l_1, \dots l_n\}$
  - constraint the number of literals assigned to *true*
  - $l_1 + \dots + l_n \geq k$ or $l_1 + \dots + l_n \leq k$ or $l_1 + \dots + l_n = k$
  - combined make up exactly all fully symmetric boolean functions
- multiple encodings of cardinality constraints
  - naïve encoding exponential: at-most-one quadratic, at-most-two cubic, etc.
  - quadratic $O(k \cdot n)$ encoding goes back to Shannon
  - linear $O(n)$ parallel counter encoding [Sinz'05]
- many variants even for at-most-one constraints
  - for an $O(n \cdot \log n)$ encoding see Prestwich's chapter in Handbook of SAT
- Pseudo-Boolean constraints (PB) or 0/1 ILP constraints have many encodings too

$$2 \cdot \overline{a} + \overline{b} + c + \overline{d} + 2 \cdot e \geq 3$$

actually used to handle MaxSAT in SAT4J for configuration in Eclipse

# BDD-Based Encoding of Cardinality Constraints

$$2 \leq l_1 + \cdots l_9 \leq 3$$



$l_1$ $l_2$ $l_3$ $l_4$ $l_5$ $l_6$ $l_7$ $l_8$ $l_9$ 0

$l_2$ $l_3$ $l_4$ $l_5$ $l_6$ $l_7$ $l_8$ $l_9$ 0

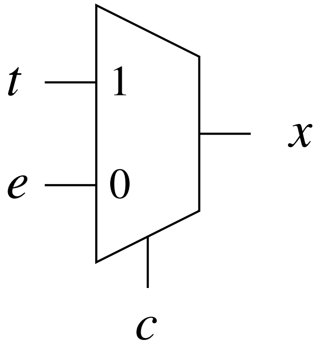
$l_3$ $l_4$ $l_5$ $l_6$ $l_7$ $l_8$ $l_9$ 1

$l_4$ $l_5$ $l_6$ $l_7$ $l_8$ $l_9$ 1

0 0 0 0 0 0

If-Then-Else gates (MUX) with “then” edge downward, dashed “else” edge to the right

# Tseitin Encoding of If-Then-Else Gate



$$
\begin{aligned}
x \leftrightarrow (c \ ?\ t : e) \quad &\Leftrightarrow \quad (x \rightarrow (c \rightarrow t)) \wedge (x \rightarrow (\bar{c} \rightarrow e)) \wedge (\bar{x} \rightarrow (c \rightarrow \bar{t})) \wedge (\bar{x} \rightarrow (\bar{c} \rightarrow \bar{e})) \\
&\Leftrightarrow \quad (\bar{x} \vee \bar{c} \vee t) \wedge (\bar{x} \vee c \vee e) \wedge (x \vee \bar{c} \vee \bar{t}) \wedge (x \vee c \vee \bar{e})
\end{aligned}
$$

minimal but not arc consistent:

- if $t$ and $e$ have the same value then $x$ needs to have that too
- possible additional clauses

$$(\bar{t} \wedge \bar{e} \rightarrow \bar{x}) \equiv (t \vee e \vee \bar{x}) \qquad\qquad (t \wedge e \rightarrow x) \equiv (\bar{t} \vee \bar{e} \vee x)$$

- but can be learned or derived through preprocessing (ternary resolution)
  keeping those clauses redundant is better in practice

# DIMACS Format

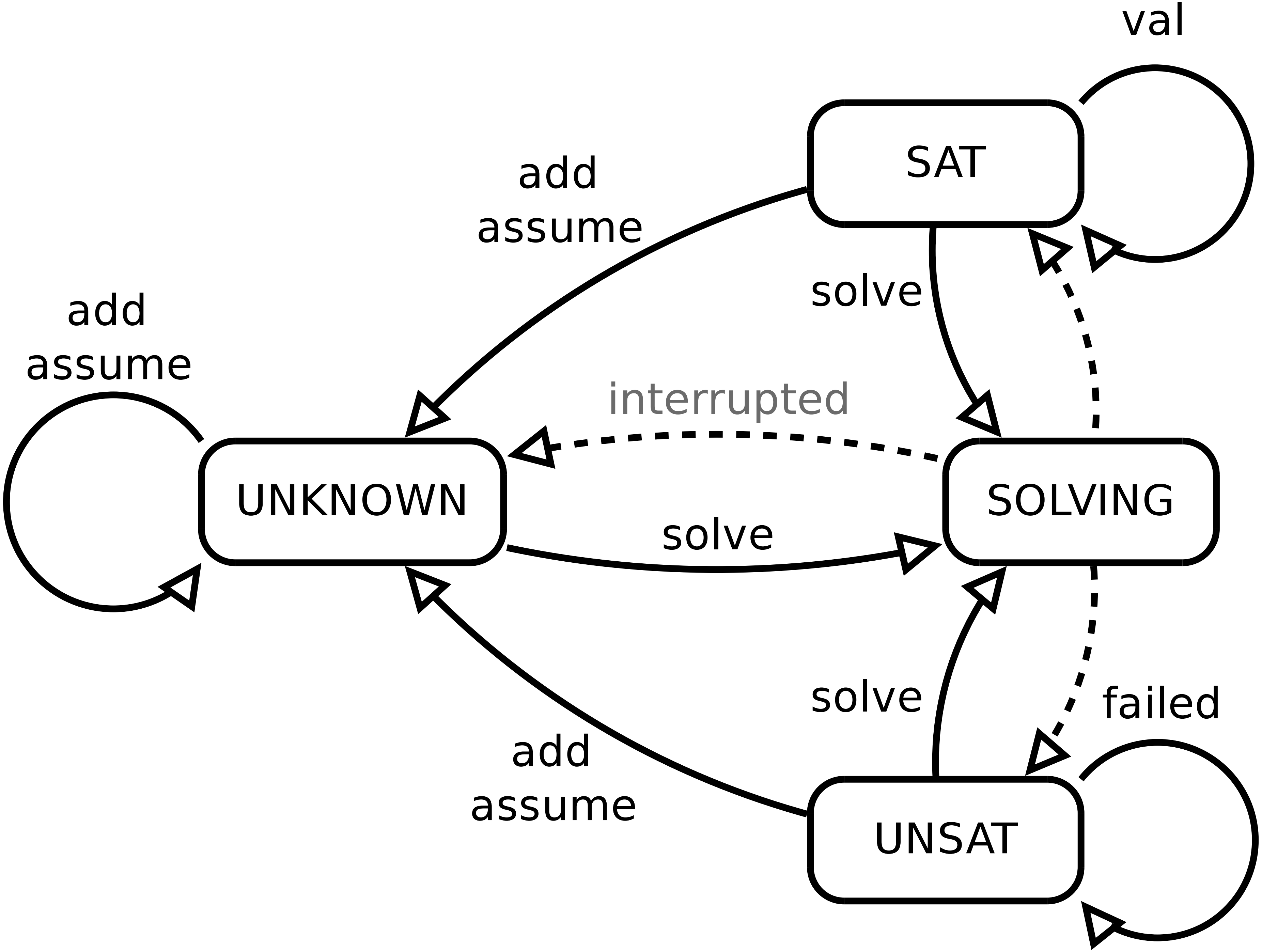
```
$ cat example.cnf
c comments start with ’c’ and extend until the end of the line
c
c variables are encoded as integers:
c
c   ’tie’   becomes ’1’
c   ’shirt’ becomes ’2’
c
c header ’p cnf <variables> <clauses>’
c
p cnf 2 3
-1  2 0           c  !tie  or  shirt
 1  2 0           c   tie  or  shirt
-1 -2 0           c  !tie  or !shirt
```

```
$ picosat example.cnf
s SATISFIABLE
v -1 2 0
```

# SAT Application Programmatic Interface (API)

- incremental usage of SAT solvers
  - add facts such as clauses incrementally
  - call SAT solver and get satisfying assignments
  - optionally retract facts
- retracting facts
  - remove clauses explicitly: complex to implement
  - push / pop: stack like activation, no sharing of learned facts
  - MiniSAT assumptions [EénSörensson'03]
- assumptions
  - unit assumptions: assumed for the next SAT call
  - easy to implement: force SAT solver to decide on assumptions first
  - shares learned clauses across SAT calls
- IPASIR: Reentrant Incremental SAT API
  - used in the SAT competition / race since 2015 [BalyoBiereIserSinz'16]

# IPASIR Model



```
#include "ipasir.h"
#include <assert.h>
#include <stdio.h>
#define ADD(LIT) ipasir_add (solver, LIT)
#define PRINT(LIT) \
  printf (ipasir_val (solver, LIT) < 0 ?  " -" #LIT : " " #LIT)
int main () {
  void * solver = ipasir_init ();
  enum { tie = 1, shirt = 2 };
  ADD (-tie); ADD ( shirt); ADD (0);
  ADD ( tie); ADD ( shirt); ADD (0);
  ADD (-tie); ADD (-shirt); ADD (0);
  int res = ipasir_solve (solver);
  assert (res == 10);
  printf ("satisfiable:"); PRINT (shirt); PRINT (tie); printf ("\n");
  printf ("assuming now: tie shirt\n");
  ipasir_assume (solver, tie); ipasir_assume (solver, shirt);
  res = ipasir_solve (solver);
  assert (res == 20);
  printf ("unsatisfiable, failed:");
  if (ipasir_failed (solver, tie)) printf (" tie");
  if (ipasir_failed (solver, shirt)) printf (" shirt");
  printf ("\n");
  ipasir_release (solver);
  return res;
}
```

```
$ ./example
satisfiable: shirt -tie
assuming now: tie shirt
unsatisfiable, failed: tie
```

# IPASIR Functions

```
const char * ipasir_signature ();

void * ipasir_init ();

void ipasir_release (void * solver);

void ipasir_add (void * solver, int lit_or_zero);

void ipasir_assume (void * solver, int lit);

int ipasir_solve (void * solver);

int ipasir_val (void * solver, int lit);

int ipasir_failed (void * solver, int lit);

void ipasir_set_terminate (void * solver, void * state,
                           int (*terminate)(void * state));
```

```
#include "cadical.hpp"
#include <cassert>
#include <iostream>
using namespace std;
#define ADD(LIT) solver.add (LIT)
#define PRINT(LIT) \
  (solver.val (LIT) < 0 ? " -" #LIT : " " #LIT)
int main () {
  CaDiCaL::Solver solver; solver.set ("quiet", 1);
  enum { tie = 1, shirt = 2 };
  ADD (-tie), ADD ( shirt), ADD (0);
  ADD ( tie), ADD ( shirt), ADD (0);
  ADD (-tie), ADD (-shirt), ADD (0);
  int res = solver.solve ();
  assert (res == 10);
  cout << "satisfiable:" << PRINT (shirt) << PRINT (tie) << endl;
  cout << "assuming now: tie shirt" << endl;
  solver.assume (tie), solver.assume (shirt);
  res = solver.solve ();
  assert (res == 20);
  cout << "unsatisfiable, failed:";
  if (solver.failed (tie)) cout << " tie";
  if (solver.failed (shirt)) cout << " shirt";
  cout << endl;
  return res;
}
```

```
$ ./example
satisfiable: shirt -tie
assuming now: tie shirt
unsatisfiable, failed: tie
```

# DP / DPLL

- dates back to the 50'ies:

  1st version DP is resolution based ⇒ preprocessing

  2nd version D(P)LL splits space for time ⇒ $\boxed{\text{CDCL}}$

- **ideas:**
  - 1st version: eliminate the two cases of assigning a variable in space or
  - 2nd version: case analysis in time, e.g. try $x = 0, 1$ in turn and recurse

- most successful SAT solvers are based on variant (CDCL) of the second version

  works for very large instances

- recent ($\leq$ 25 years) optimizations:

  backjumping, learning, UIPs, dynamic splitting heuristics, fast data structures

# DP Procedure

```
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 $\neg x$
```

⇒ Bounded Variable Elimination

# D(P)LL Procedure

```
$DPLL(F)$
    $F := BCP(F)$                                        boolean constraint propagation
    if $F = \top$ return satisfiable
    if $\bot \in F$ return unsatisfiable
    pick remaining variable $x$ and literal $l \in \{x, \neg x\}$
    if $DPLL(F \wedge \{l\})$ returns satisfiable return satisfiable
    return $DPLL(F \wedge \{\neg l\})$
```

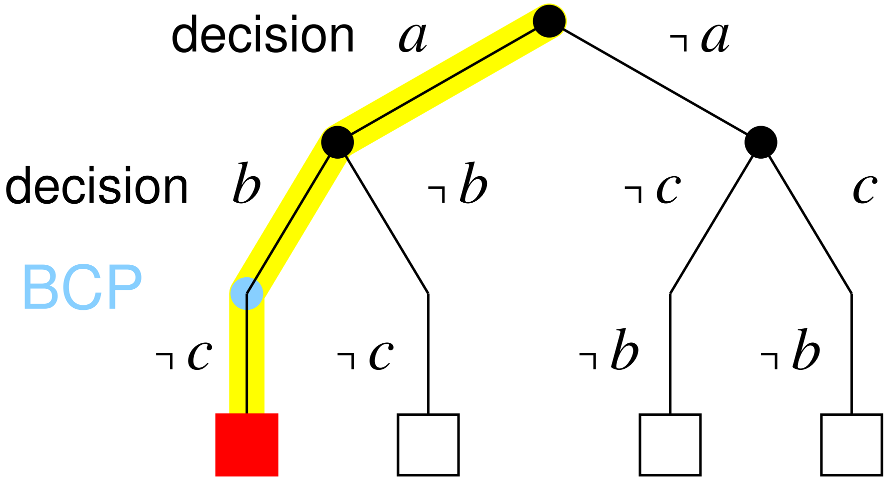
⇒ CDCL

# DPLL Example

$a = 1$

$b = 1$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

# Conflict Driven Clause Learning (CDCL)

[MarqueSilvaSakallah'96]

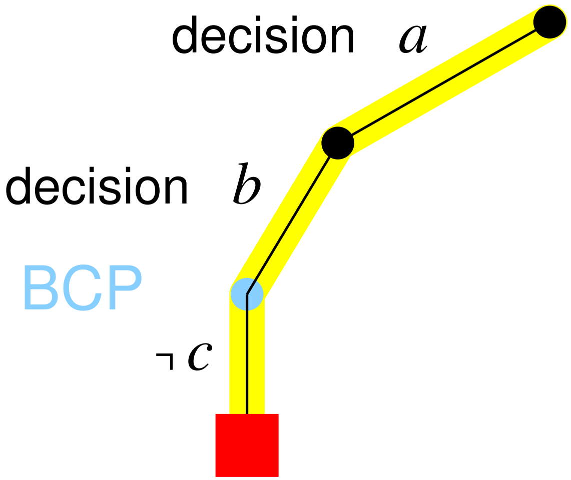
- first implemented in the context of GRASP SAT solver
  - name given later to distinguish it from DPLL
  - not recursive anymore
- essential for SMT
- learning clauses as no-goods
- notion of implication graph
- (first) unique implication points

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 1$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

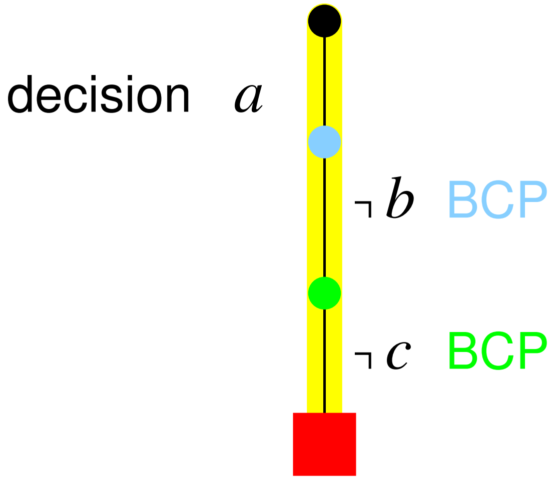
learn $\neg a \vee \neg b$

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$



decision $a$

$\neg b$ BCP

$\neg c$ BCP

clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

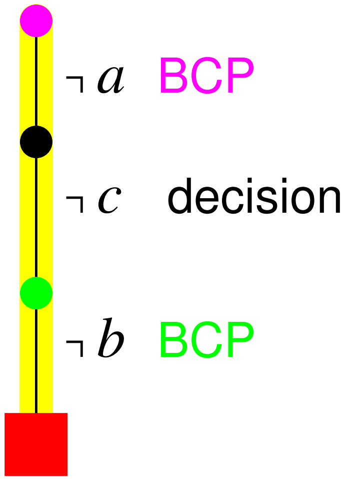
learn $\neg a$

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$



$\neg a$ BCP

$\neg c$ decision

$\neg b$ BCP

clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

$\neg a$

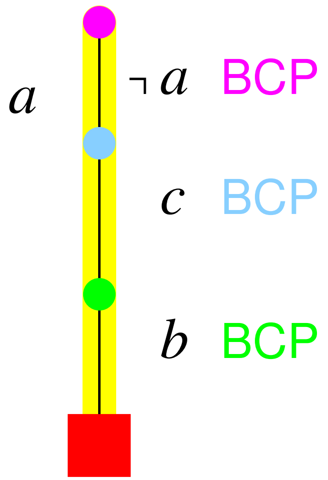
learn $c$

# Conflict Driven Clause Learning (CDCL)

$a = 1$

$b = 0$

$c = 0$



clauses

$\neg a \vee \neg b \vee \neg c$

$\neg a \vee \neg b \vee c$

$\neg a \vee b \vee \neg c$

$\neg a \vee b \vee c$

$a \vee \neg b \vee \neg c$

$a \vee \neg b \vee c$

$a \vee b \vee \neg c$

$a \vee b \vee c$

$\neg a \vee \neg b$

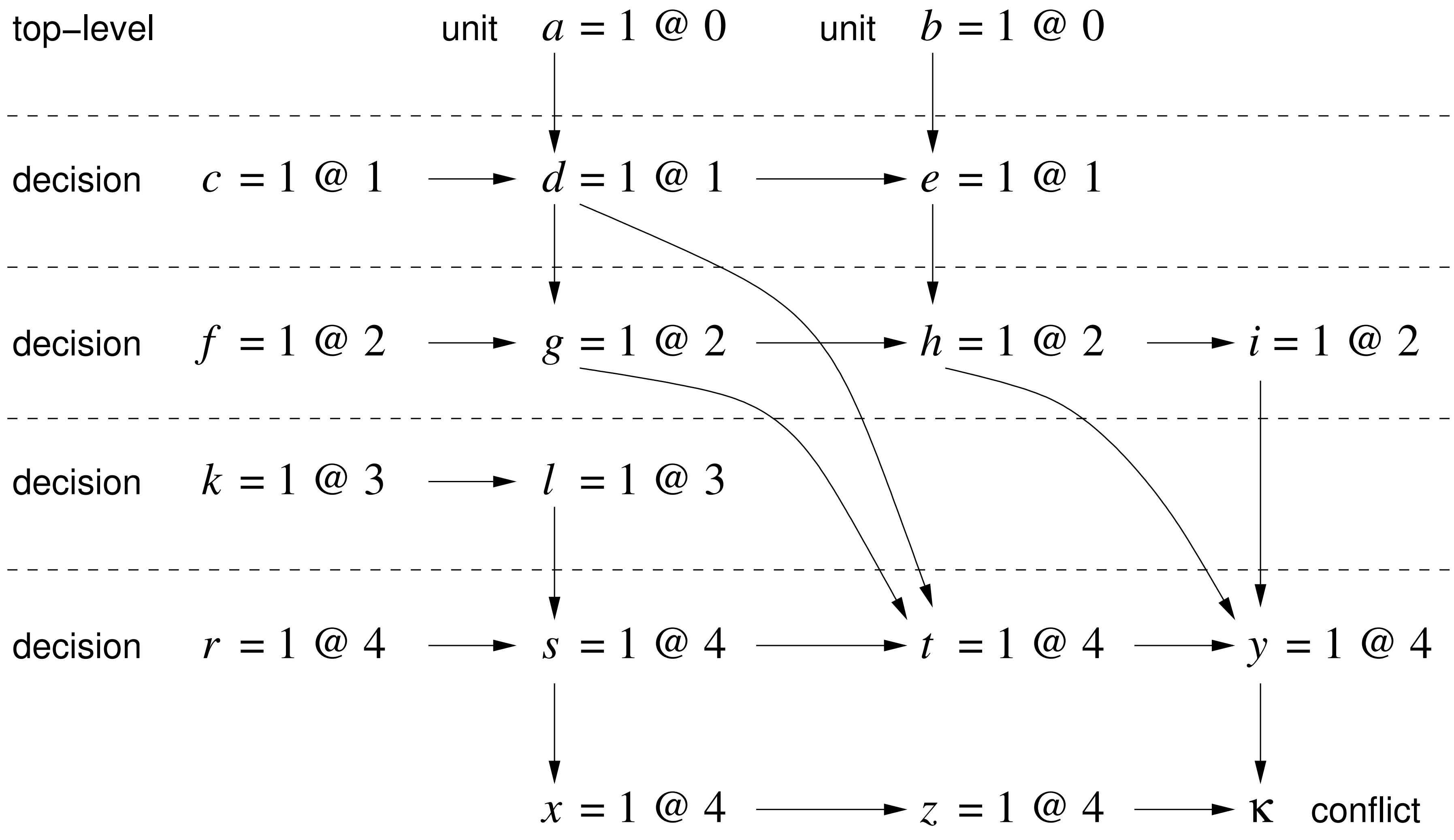
$\neg a$

$c$

learn $\bot$

empty clause

# Implication Graph



top−level unit $a = 1 @ 0$ unit $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

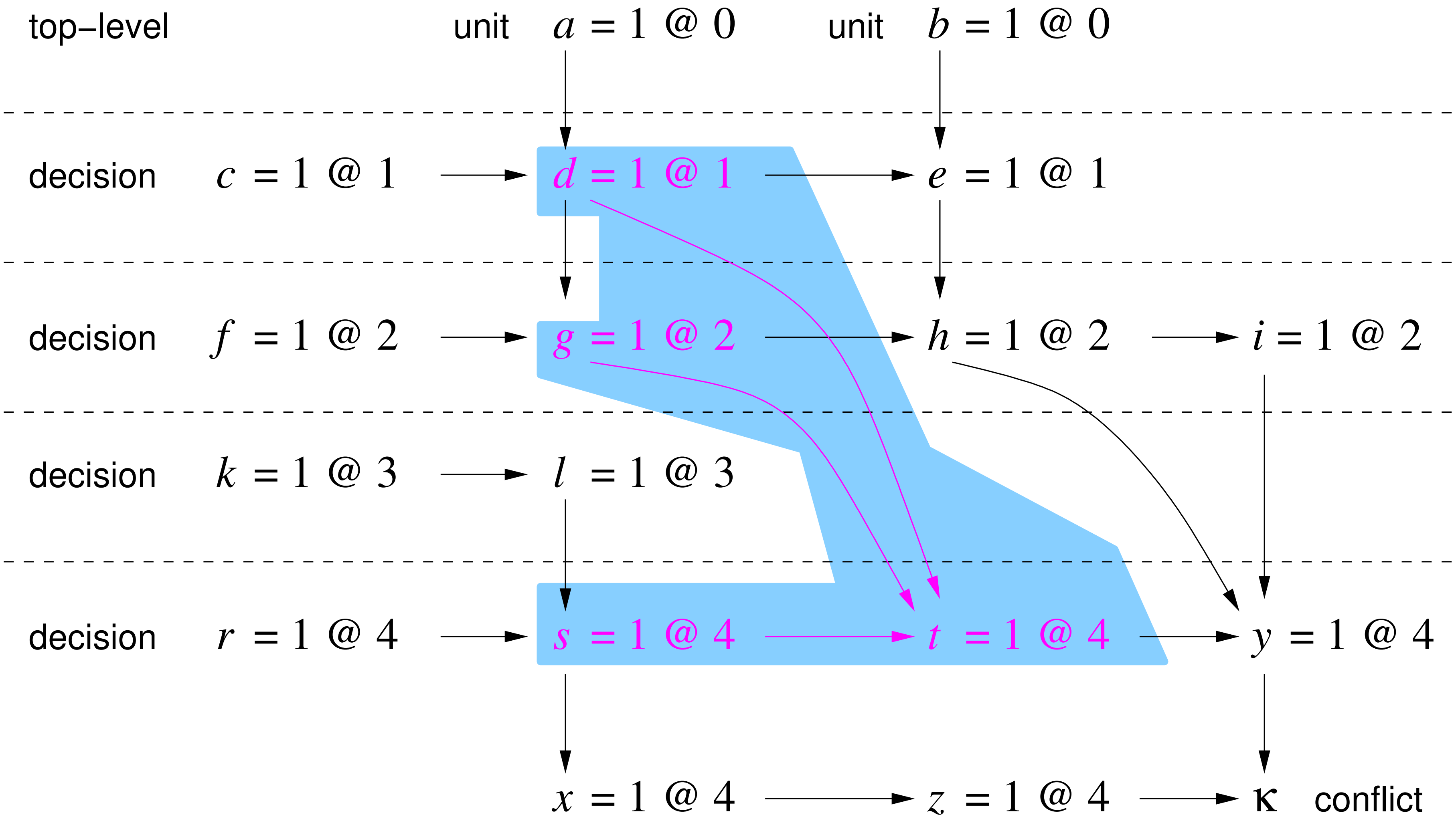
decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

decision $k = 1 @ 3$ $l = 1 @ 3$

decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

# Antecedents / Reasons



top−level unit $a = 1 @ 0$ unit $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

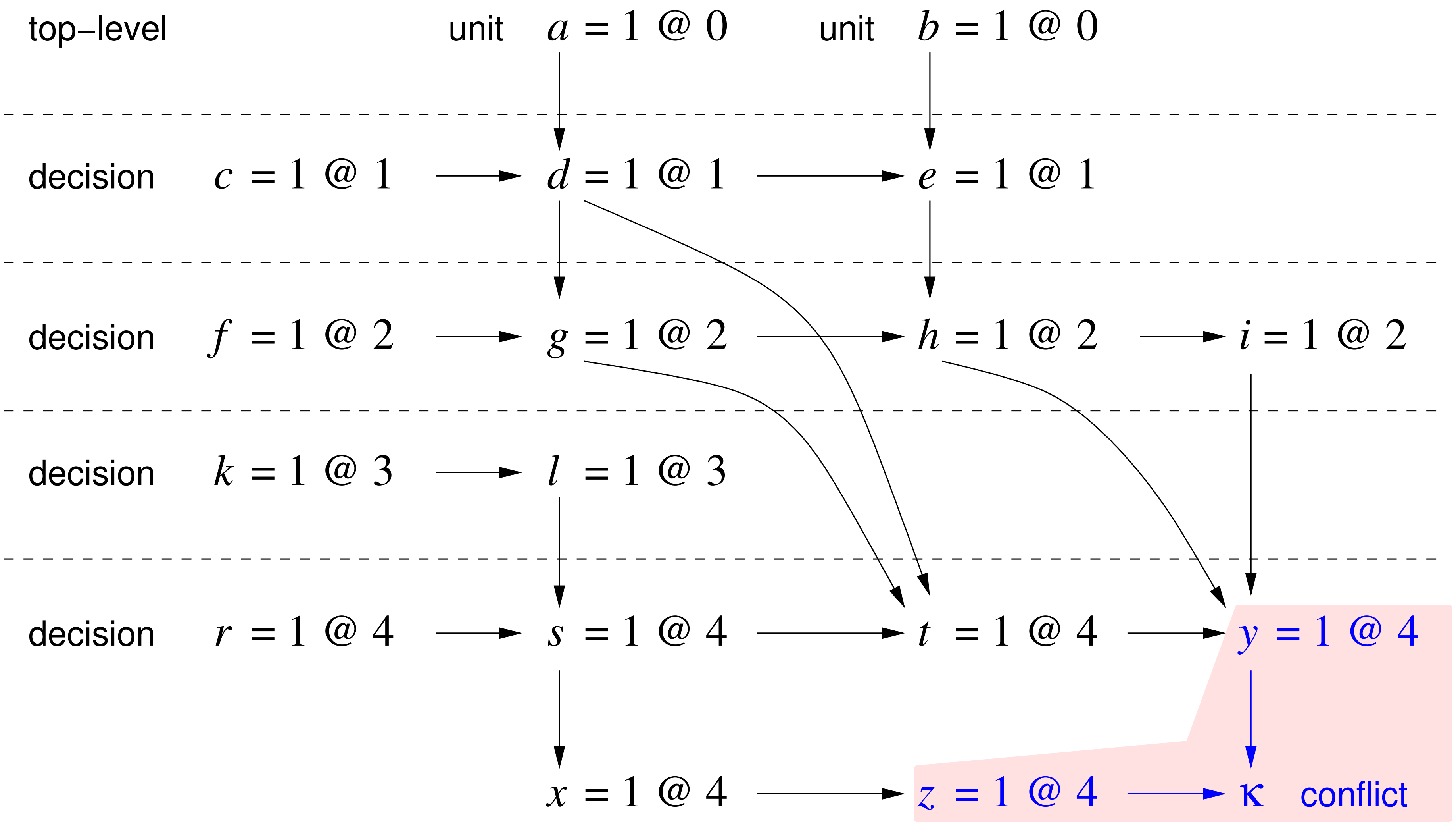
decision $k = 1 @ 3$ $l = 1 @ 3$

decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

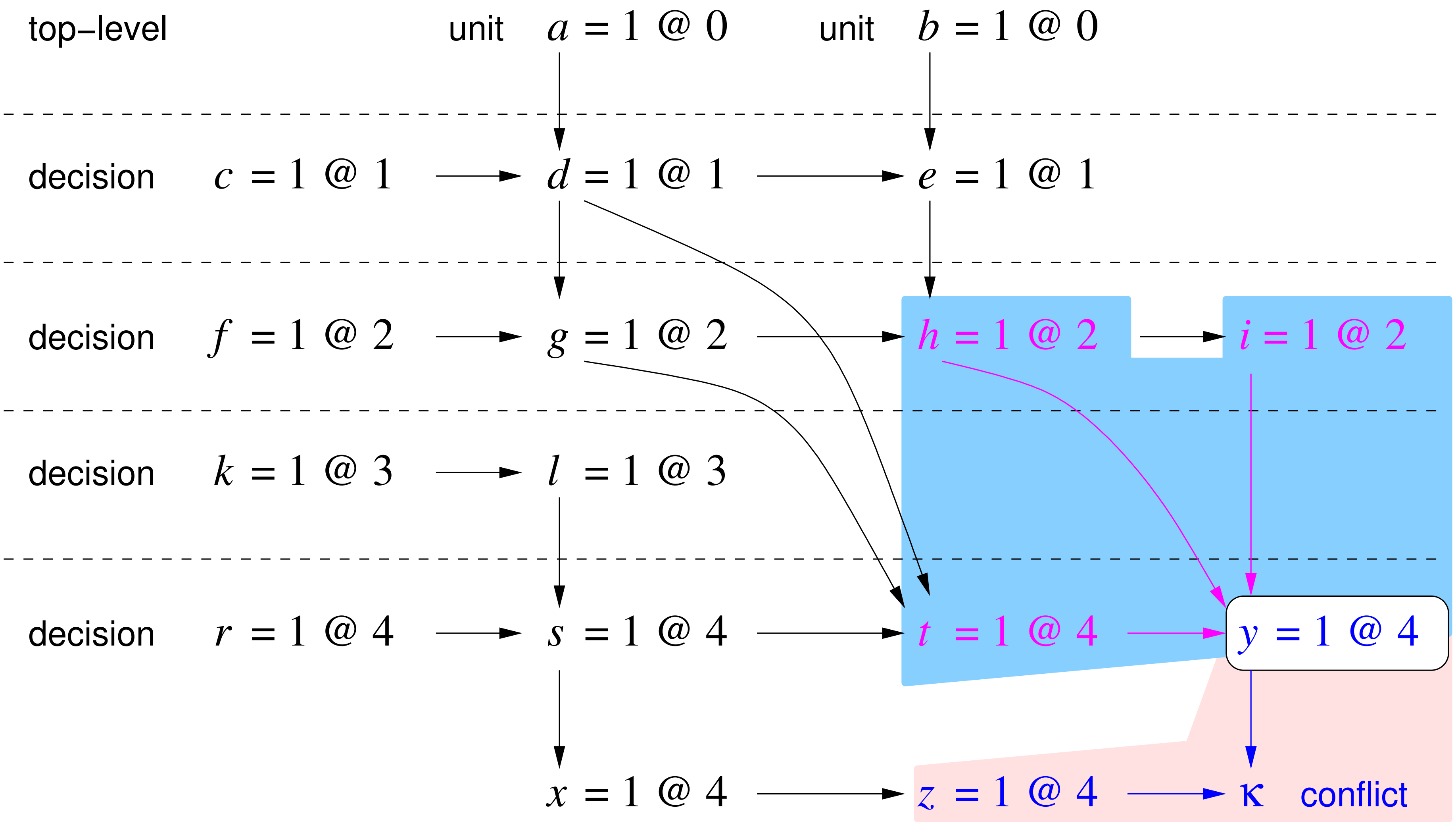
$$d \wedge g \wedge s \rightarrow t \qquad \equiv \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee t)$$

# Conflicting Clauses



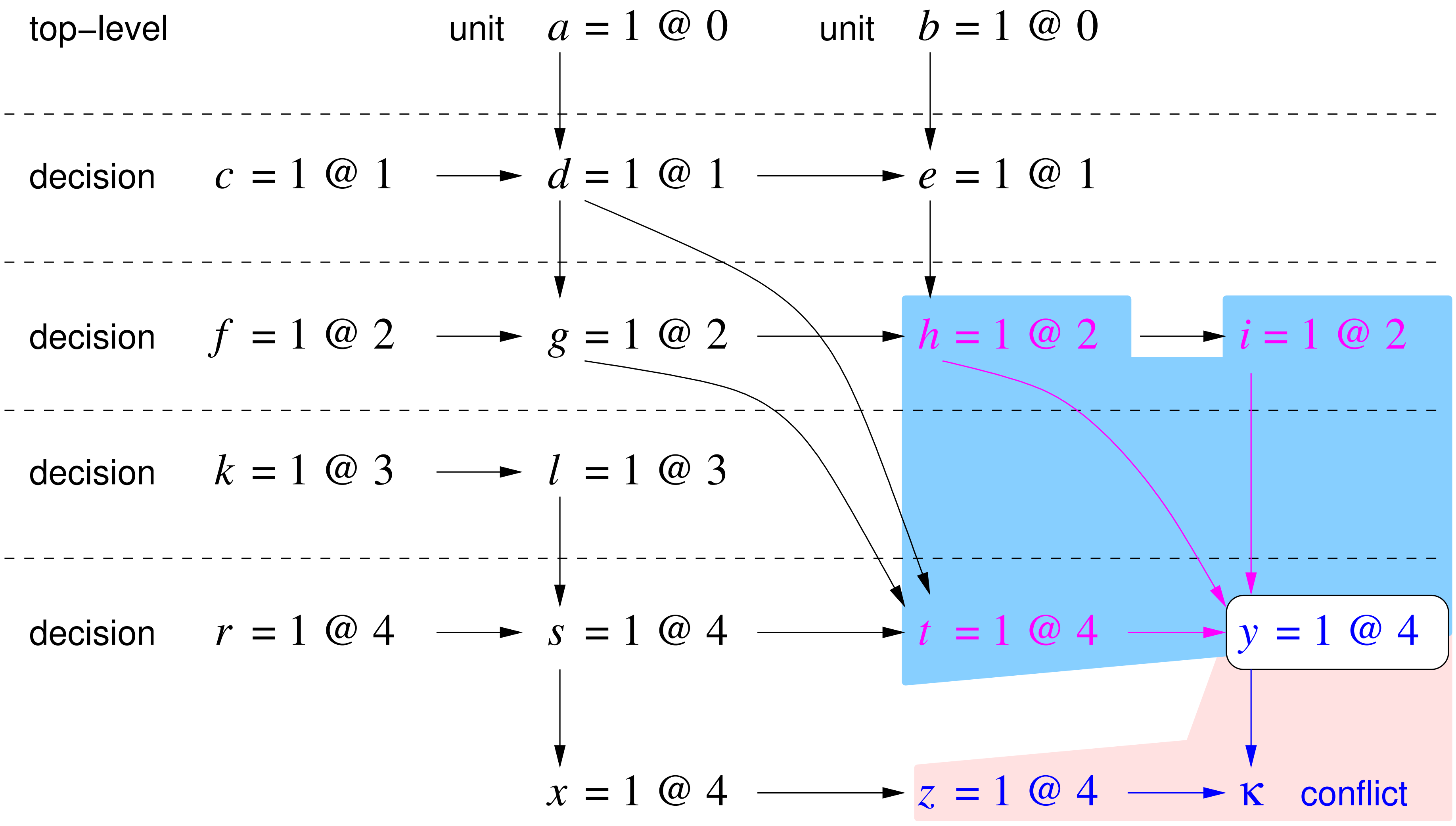
$$\neg(y \wedge z) \qquad \equiv \qquad (\bar{y} \vee \bar{z})$$

# Resolving Antecedents 1st Time



$(\bar{h} \vee \bar{i} \vee \bar{t} \vee y)$ $(\bar{y} \vee \bar{z})$

# Resolving Antecedents 1st Time



top–level unit $a = 1 @ 0$ unit $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

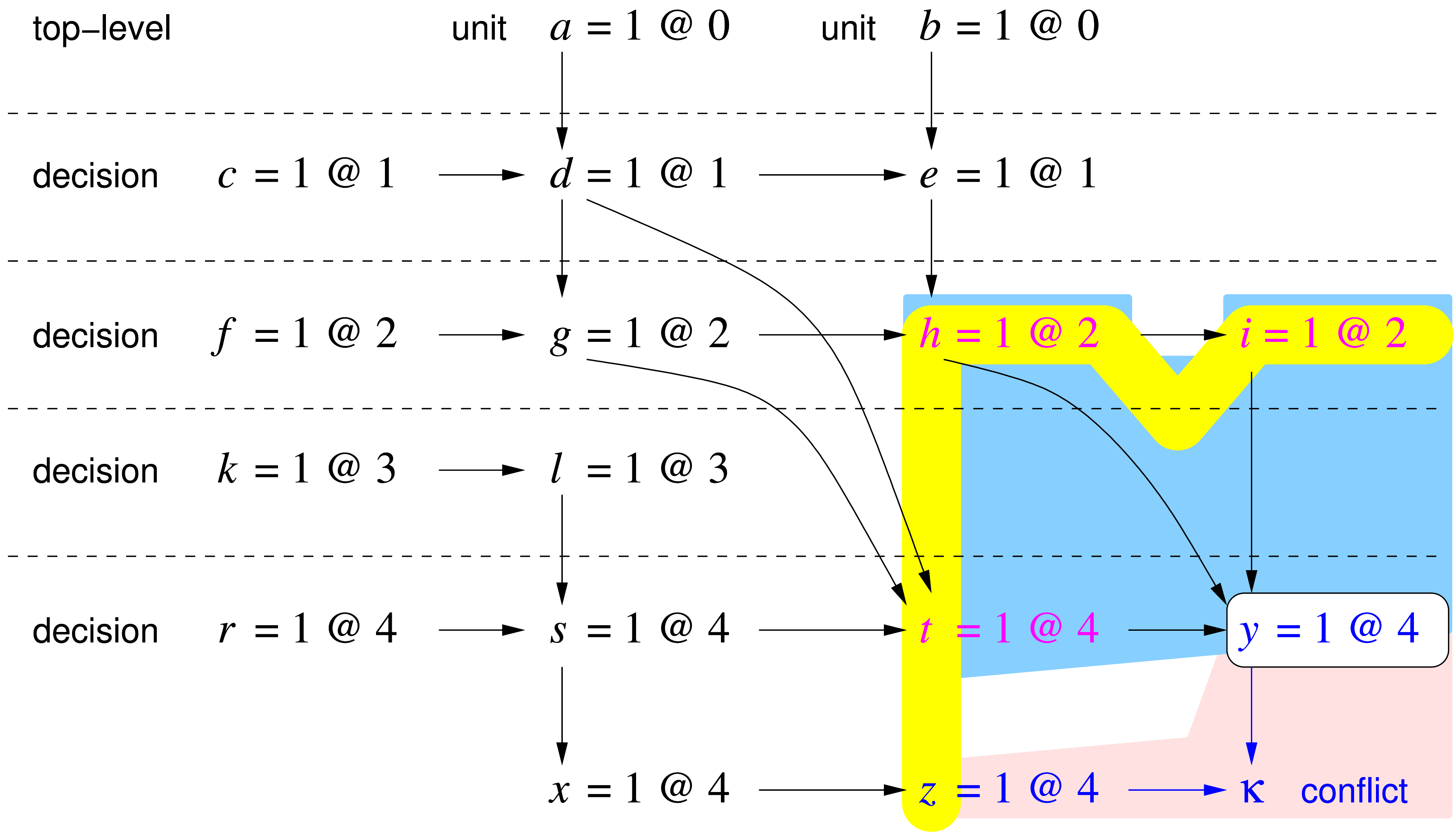
decision $k = 1 @ 3$ $l = 1 @ 3$

decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

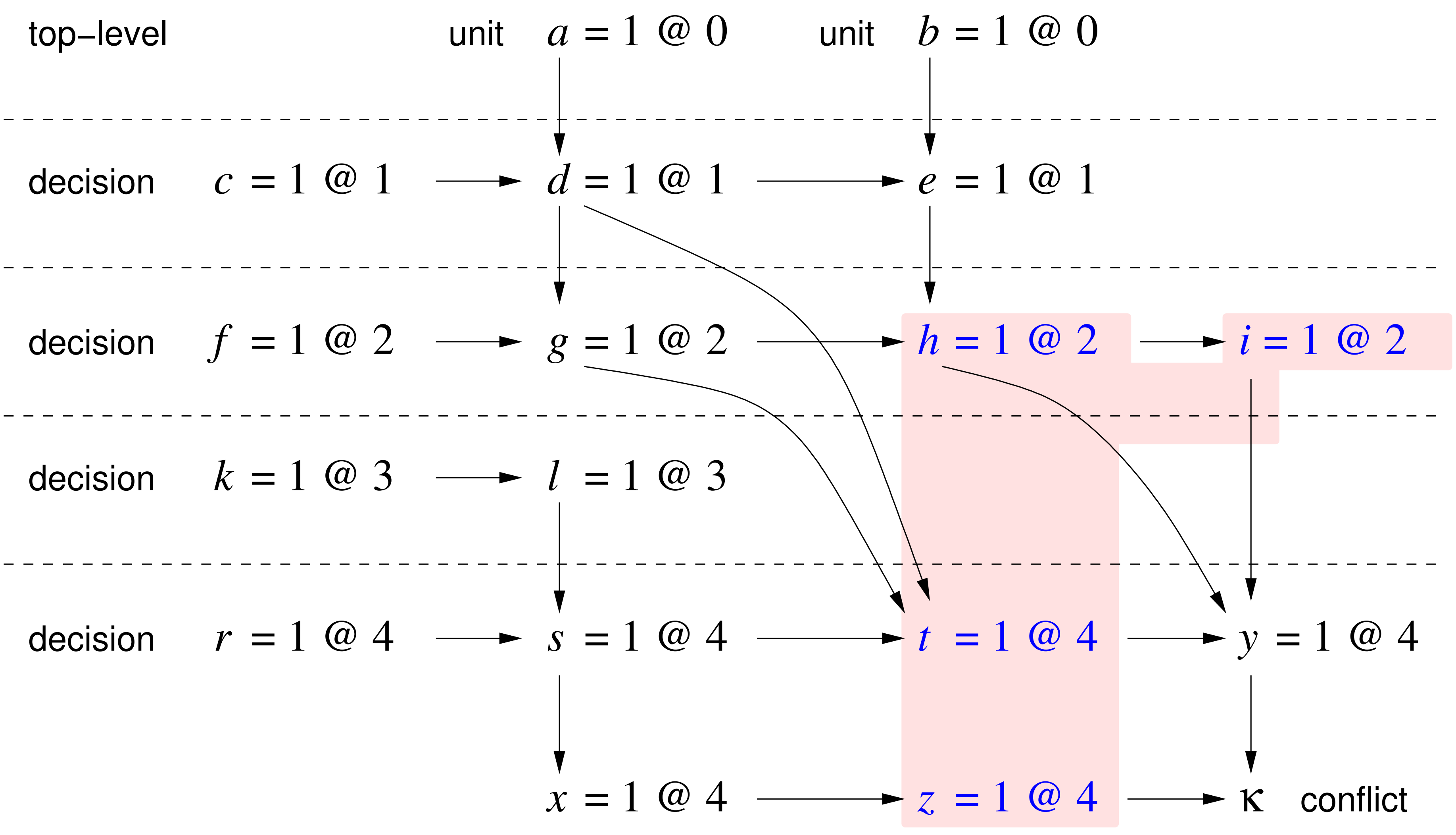
$$\frac{(\bar{h} \lor \bar{i} \lor \bar{t} \lor y) \qquad (\bar{y} \lor \bar{z})}{\boxed{(\bar{h} \lor \bar{i} \lor \bar{t} \lor \bar{z})}}$$

# Resolvents = Cuts = Potential Learned Clauses



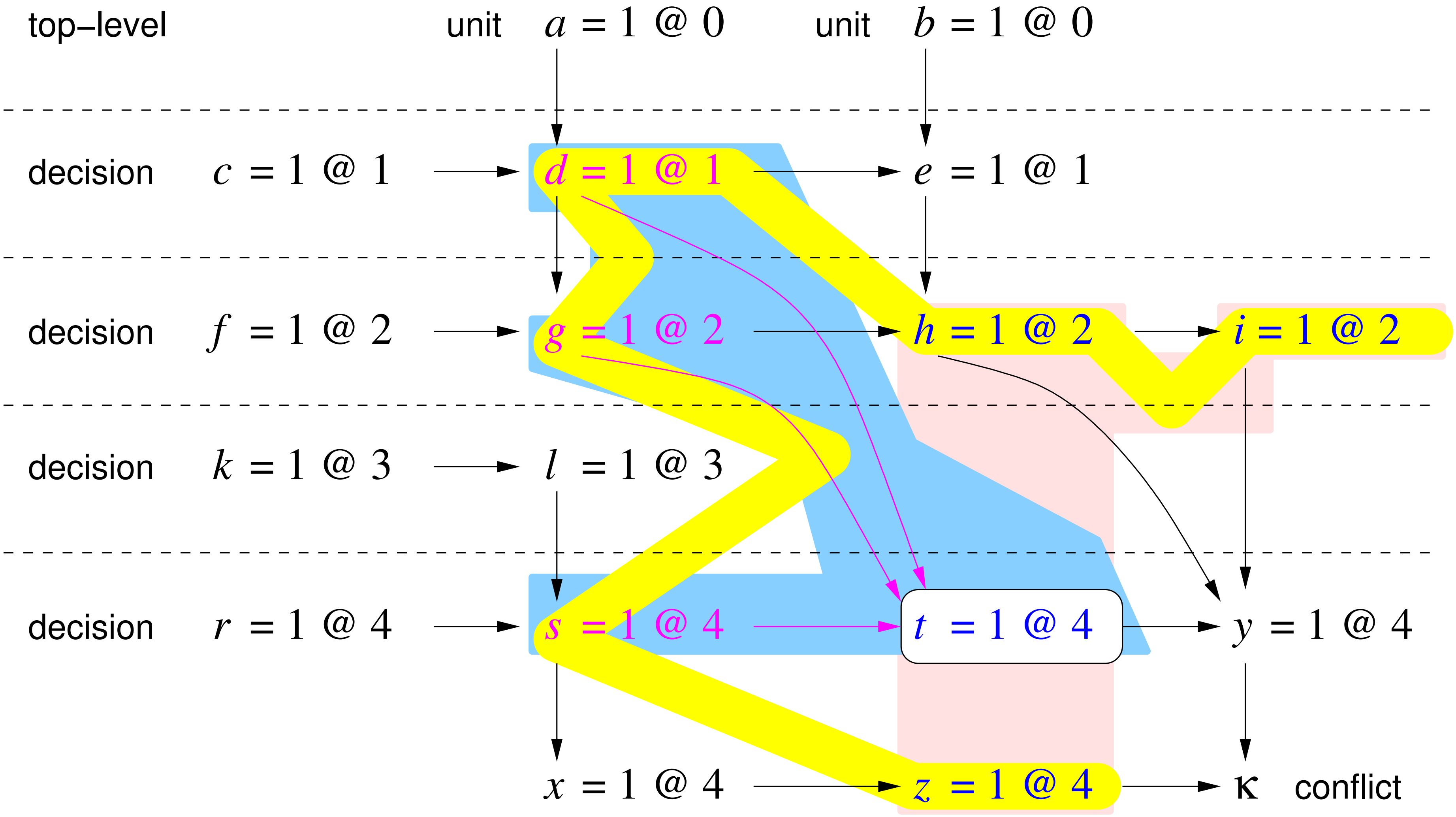
$$\frac{(\bar{h} \vee \bar{i} \vee \bar{t} \vee y) \qquad (\bar{y} \vee \bar{z})}{(\bar{h} \vee \bar{i} \vee \bar{t} \vee \bar{z})}$$

# Potential Learned Clause After 1 Resolution



$(\bar{h} \vee \bar{i} \vee \bar{t} \vee \bar{z})$

# Resolving Antecedents 2<sup>nd</sup> Time



top−level unit $a = 1 @ 0$ unit $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

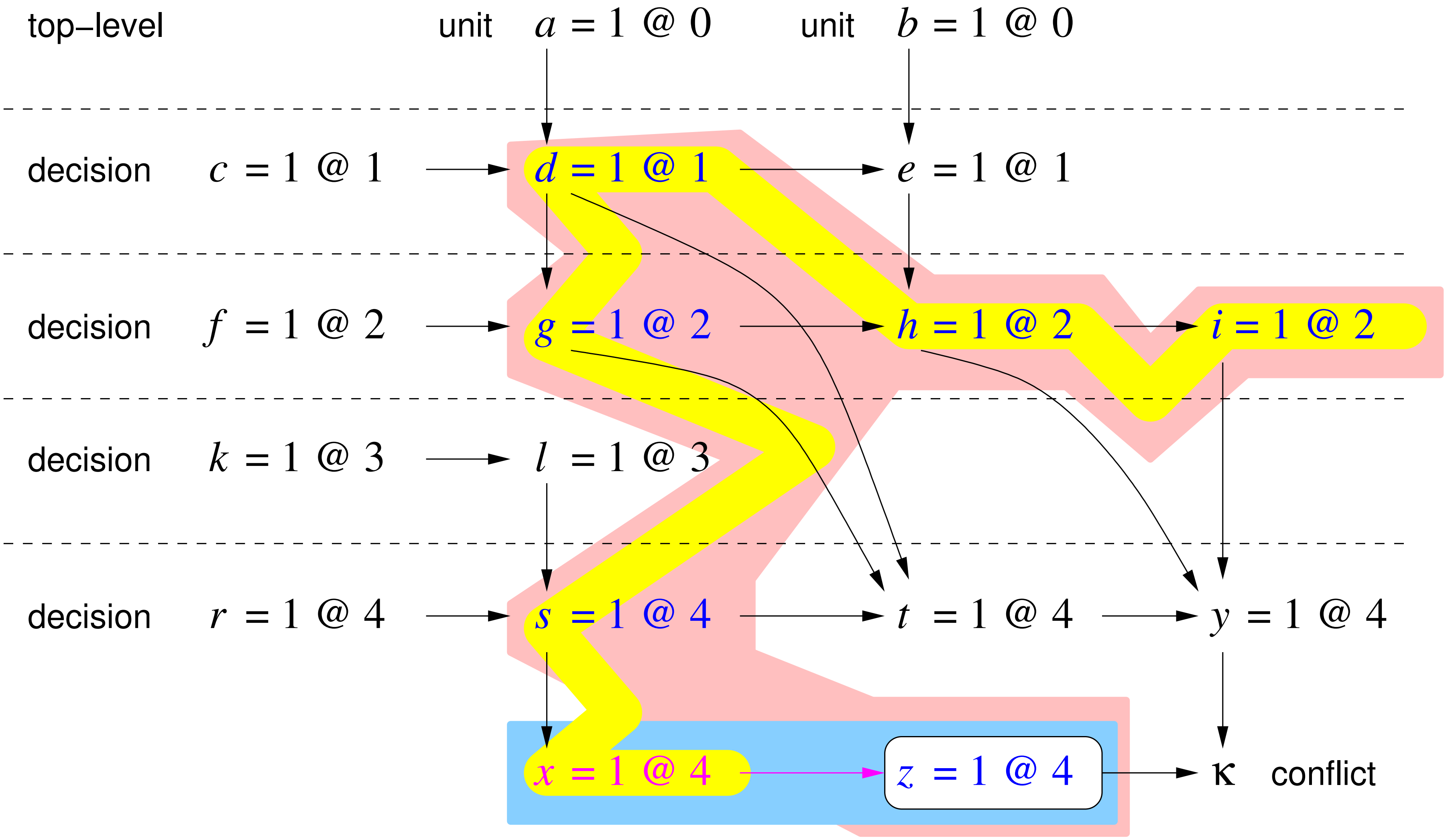
decision $k = 1 @ 3$ $l = 1 @ 3$

decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

$$\frac{(\bar{d}\vee\bar{g}\vee\bar{s}\vee t)\qquad(\bar{h}\vee\bar{i}\vee\bar{t}\vee\bar{z})}{(\bar{d}\vee\bar{g}\vee\bar{s}\vee\bar{h}\vee\bar{i}\vee\bar{z})}$$

# Resolving Antecedents 3rd Time



top–level unit $a = 1 @ 0$ unit $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

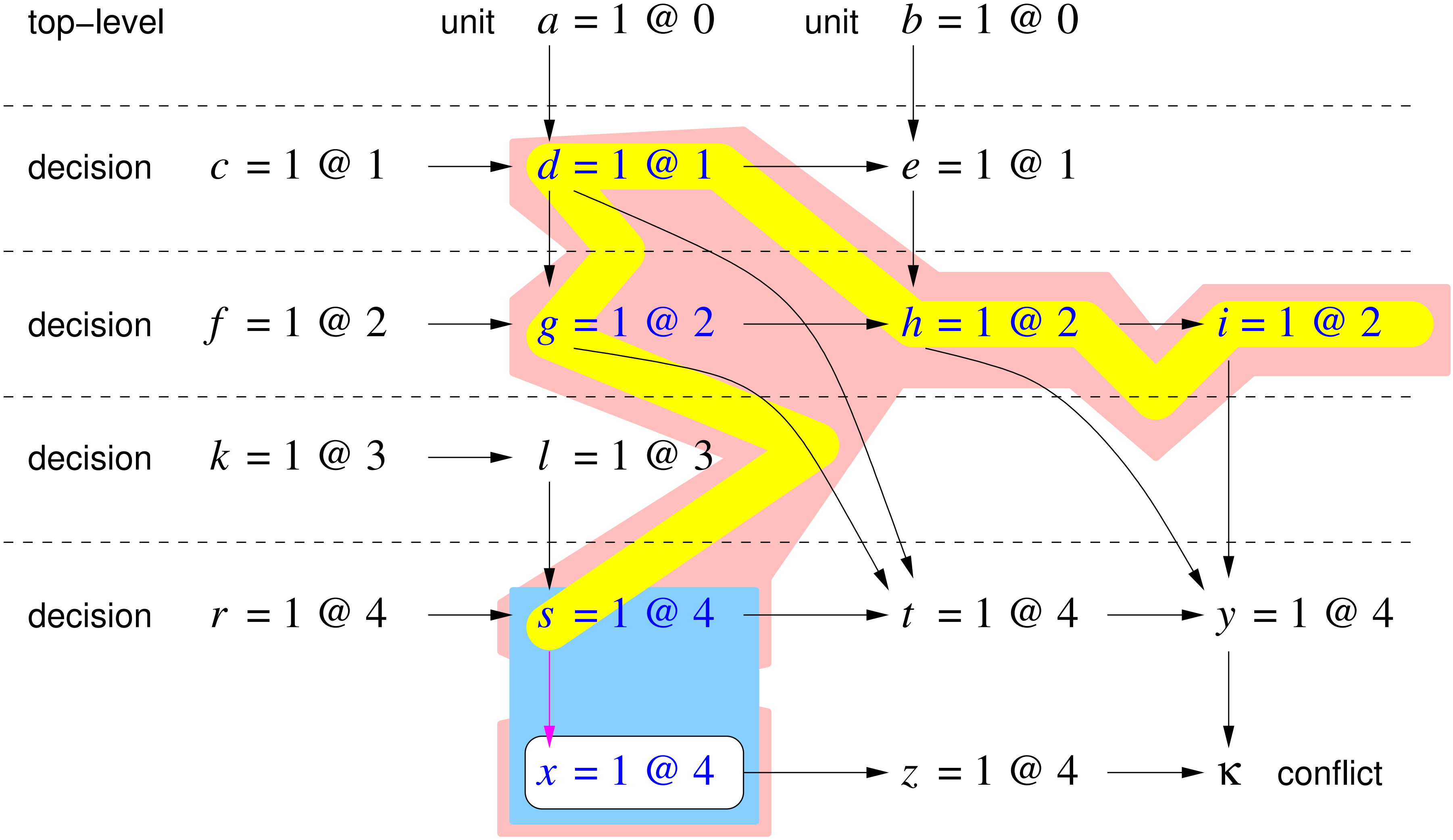
decision $k = 1 @ 3$ $l = 1 @ 3$

decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

$$\frac{(\overline{x} \vee z) \qquad (\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i} \vee \overline{z})}{(\overline{x} \vee \overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i})}$$

# Resolving Antecedents 4th Time



top–level unit $a = 1 @ 0$ unit $b = 1 @ 0$

decision $c = 1 @ 1$ $d = 1 @ 1$ $e = 1 @ 1$

decision $f = 1 @ 2$ $g = 1 @ 2$ $h = 1 @ 2$ $i = 1 @ 2$

decision $k = 1 @ 3$ $l = 1 @ 3$

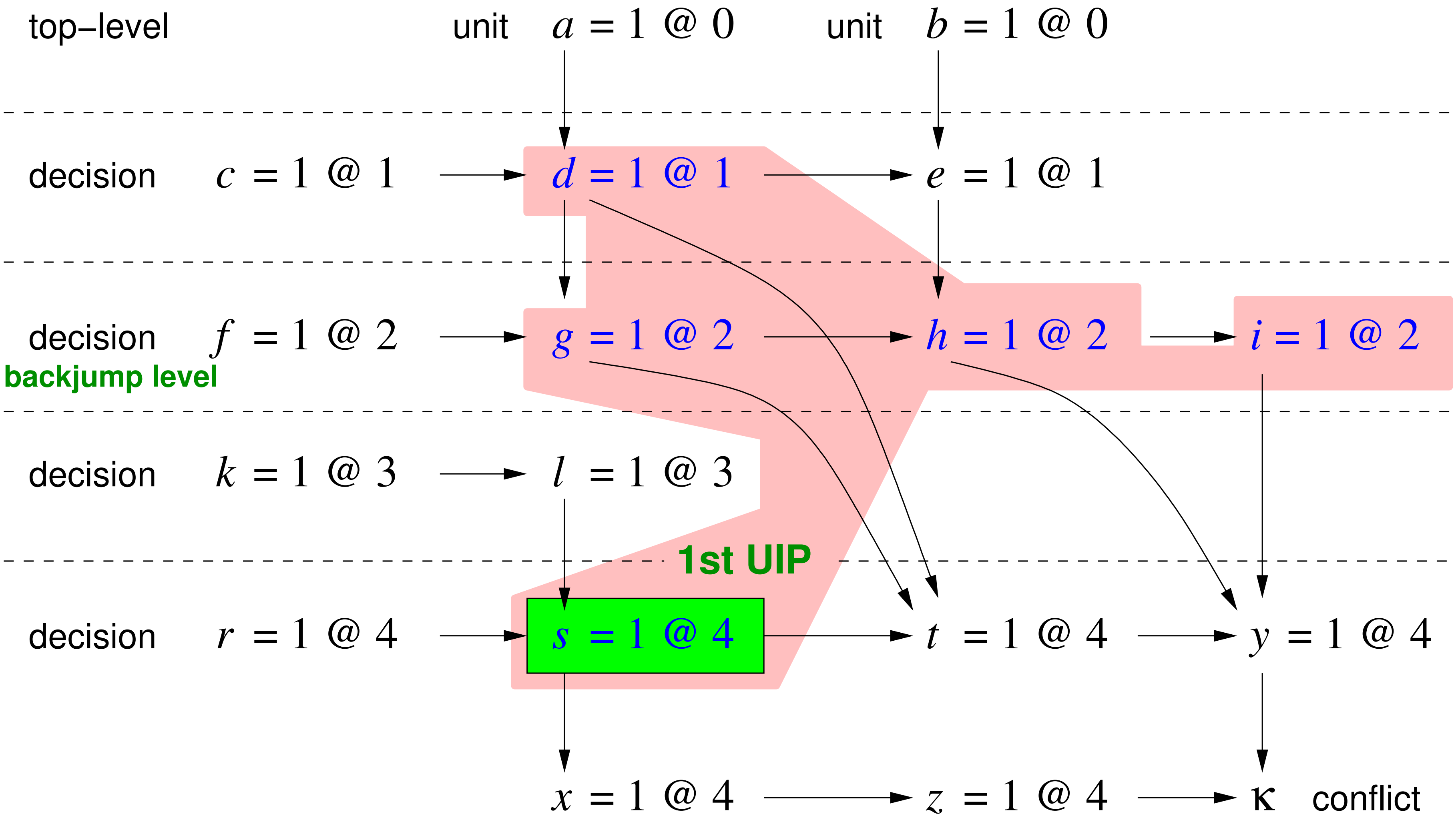
decision $r = 1 @ 4$ $s = 1 @ 4$ $t = 1 @ 4$ $y = 1 @ 4$

$x = 1 @ 4$ $z = 1 @ 4$ $\kappa$ conflict

$$\frac{(\bar{s} \vee x) \qquad (\bar{x} \vee \bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}{(\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}$$

self subsuming resolution

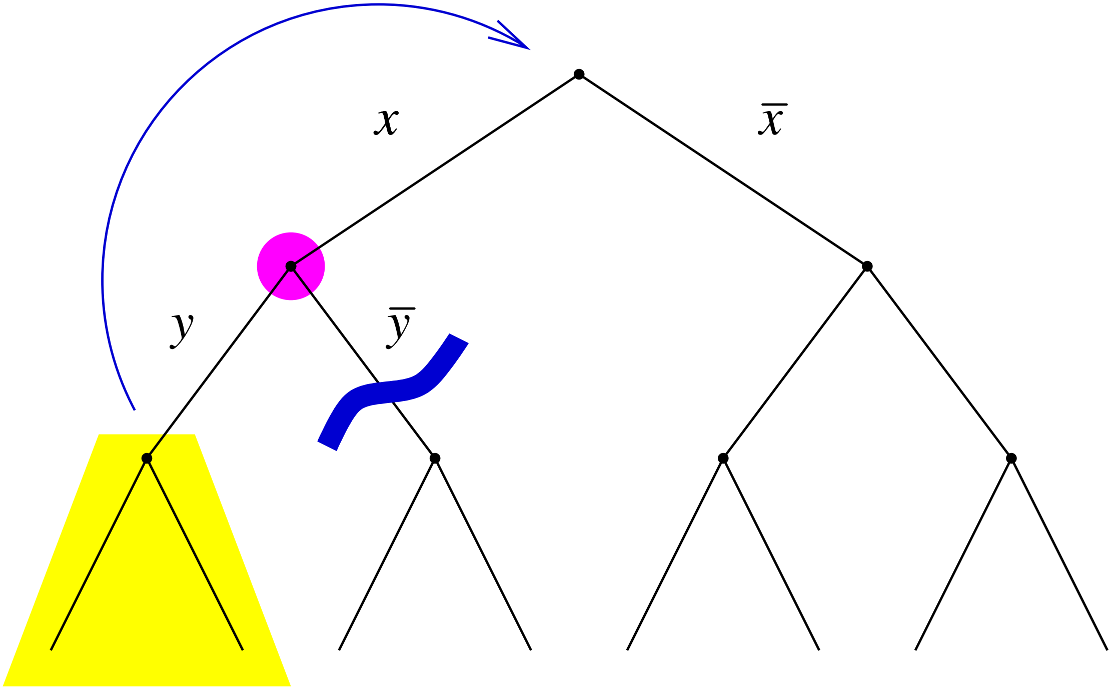
# 1st UIP Clause after 4 Resolutions



$$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i})$$

UIP = unique implication point     dominates conflict on the last level

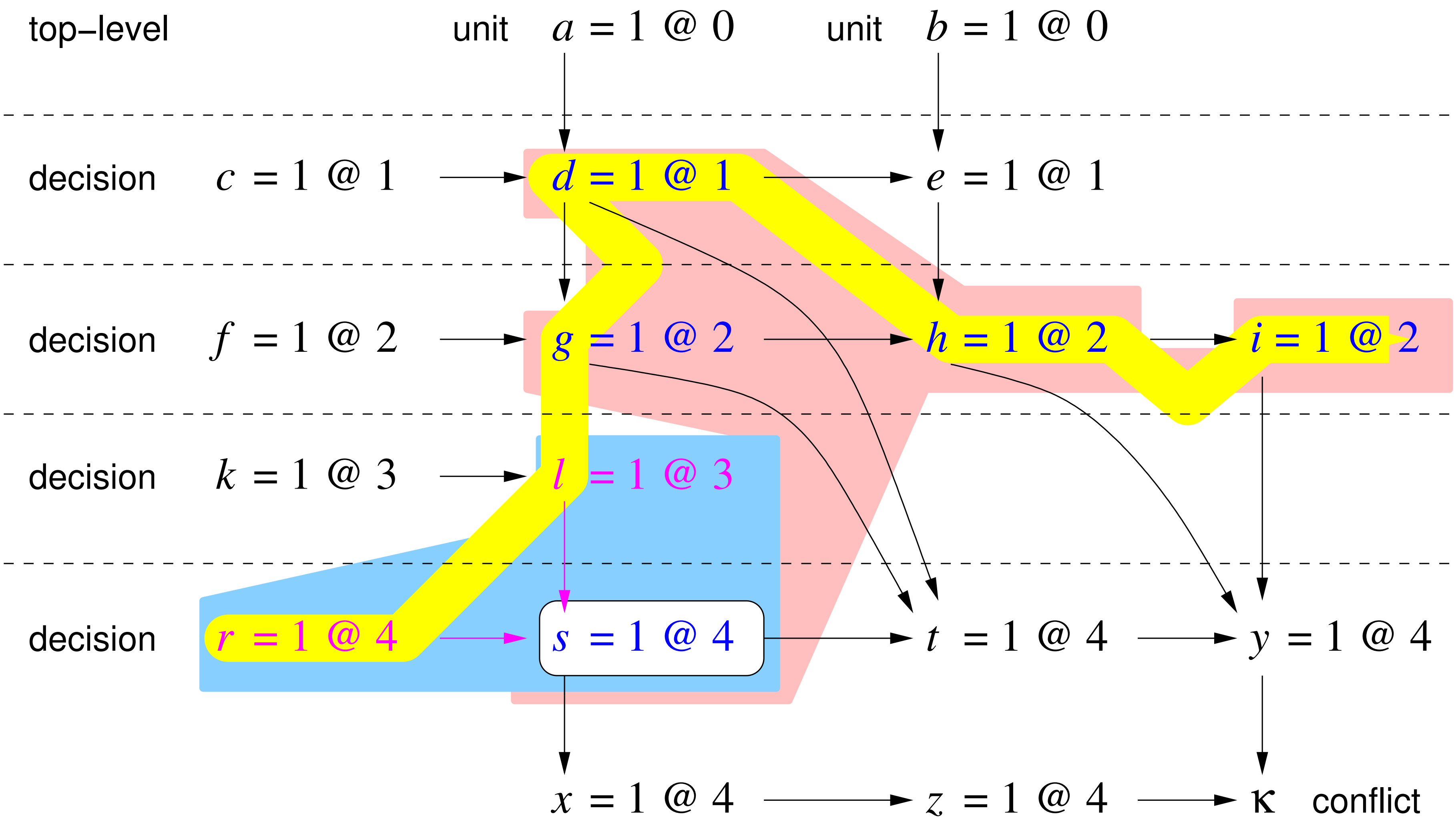
# Backjumping



If $y$ has never been used to derive a conflict, then skip $\overline{y}$ case.

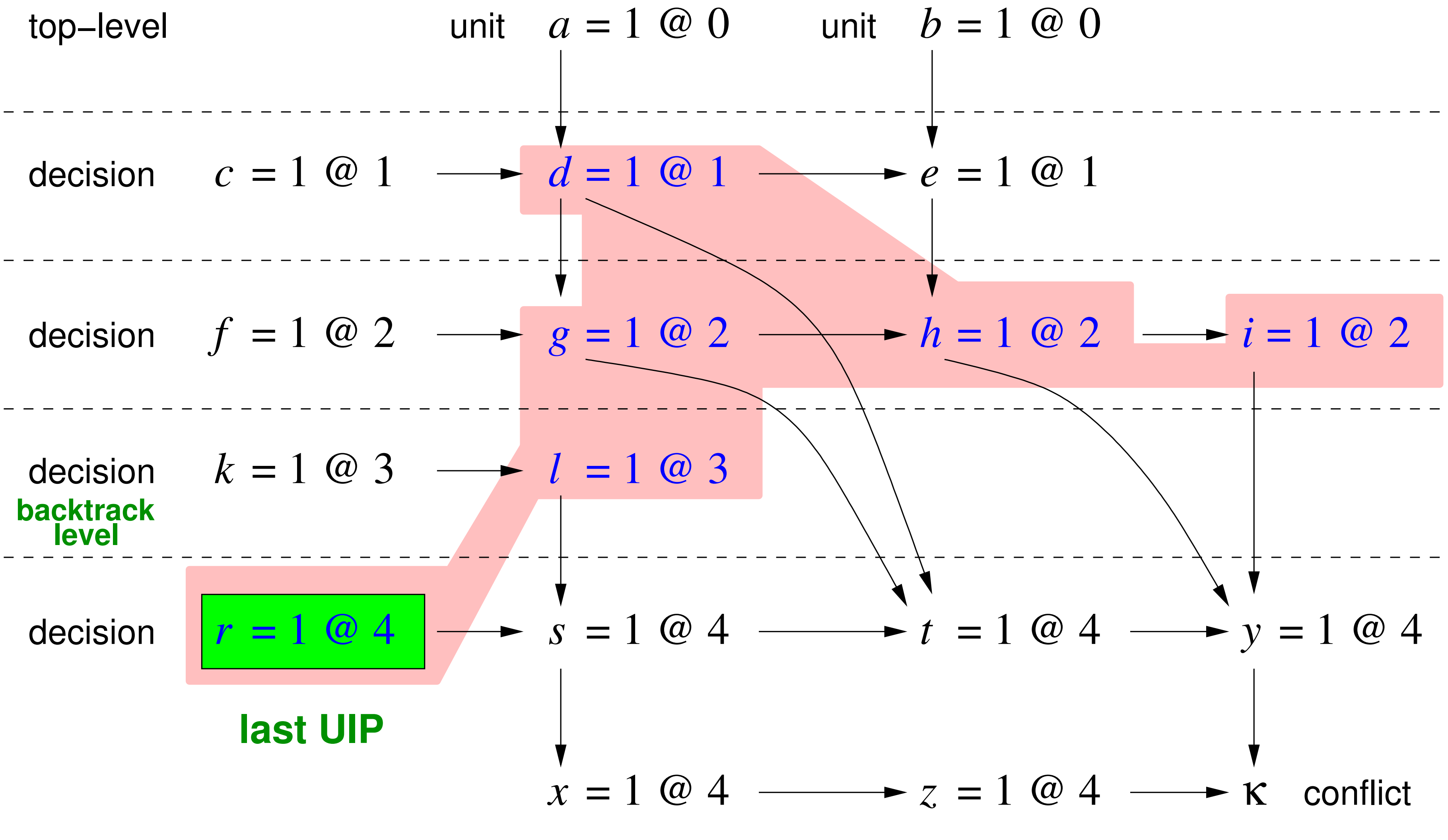
Immediately jump back to the $\overline{x}$ case – assuming $x$ was used.

# Resolving Antecedents 5th Time



$$\frac{(\bar{l} \lor \bar{r} \lor s) \qquad (\bar{d} \lor \bar{g} \lor \bar{s} \lor \bar{h} \lor \bar{i})}{(\bar{l} \lor \bar{r} \lor \bar{d} \lor \bar{g} \lor \bar{h} \lor \bar{i})}$$

# Decision Learned Clause



top–level

unit $a = 1 \ @\ 0$

unit $b = 1 \ @\ 0$

decision $c = 1 \ @\ 1$

$d = 1 \ @\ 1$

$e = 1 \ @\ 1$

decision $f = 1 \ @\ 2$

$g = 1 \ @\ 2$

$h = 1 \ @\ 2$

$i = 1 \ @\ 2$

decision $k = 1 \ @\ 3$

$l = 1 \ @\ 3$

backtrack level

decision $r = 1 \ @\ 4$

$s = 1 \ @\ 4$

$t = 1 \ @\ 4$

$y = 1 \ @\ 4$

last UIP

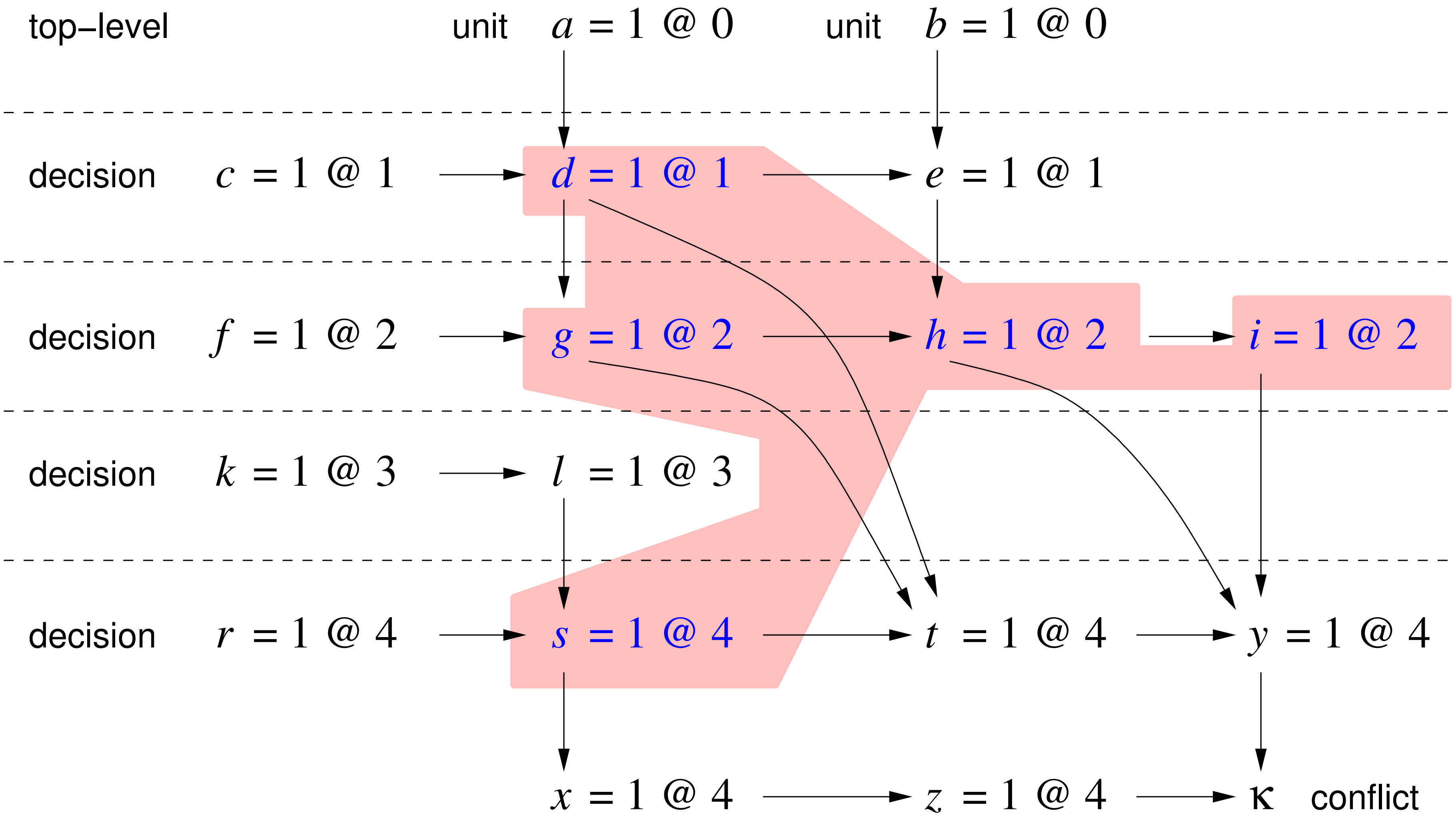
$x = 1 \ @\ 4$

$z = 1 \ @\ 4$

$\kappa$ conflict

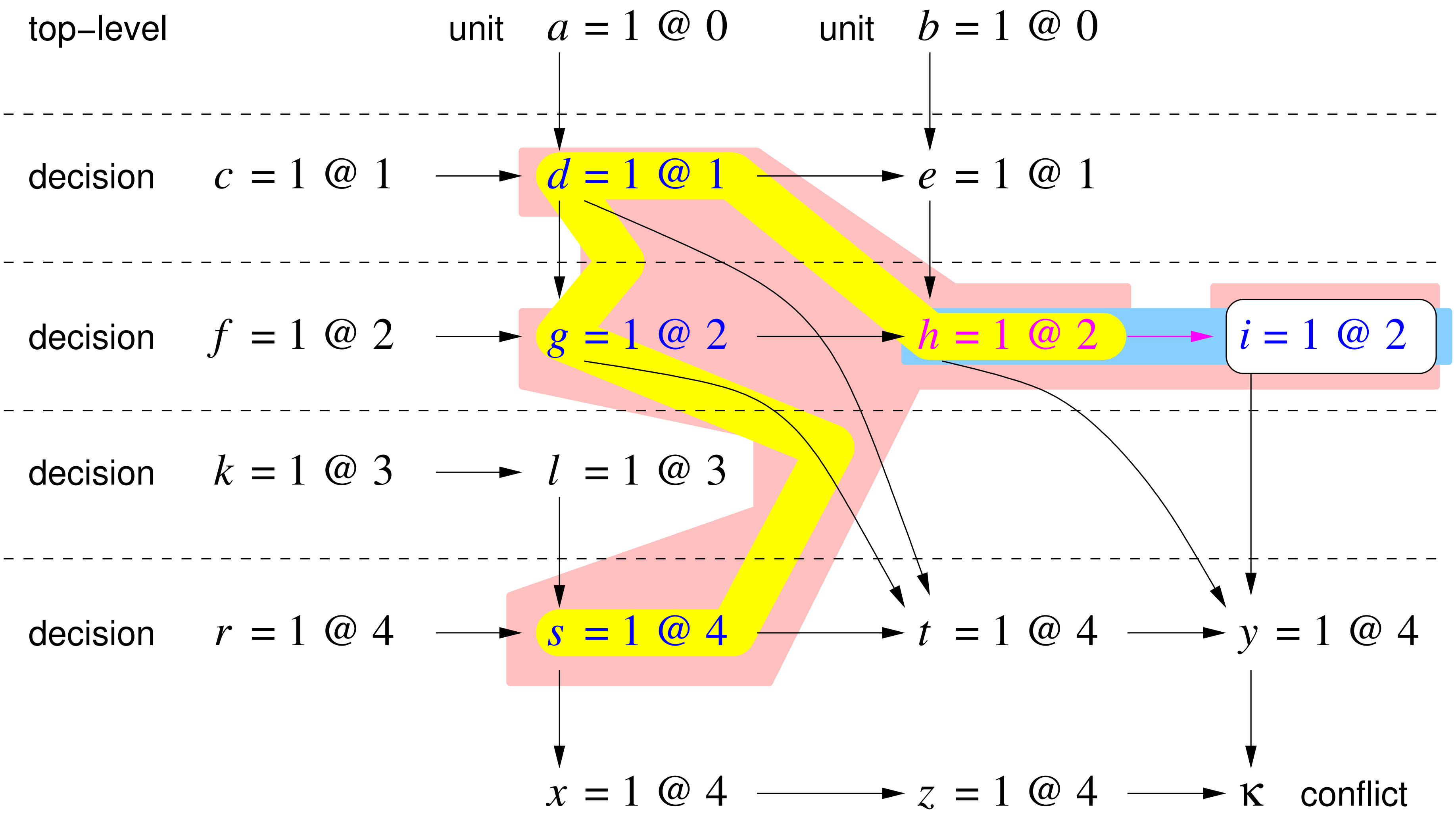
$$(\bar{d} \vee \bar{g} \vee \bar{l} \vee \bar{r} \vee \bar{h} \vee \bar{i})$$

# 1st UIP Clause after 4 Resolutions



$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h} \vee \overline{i})$

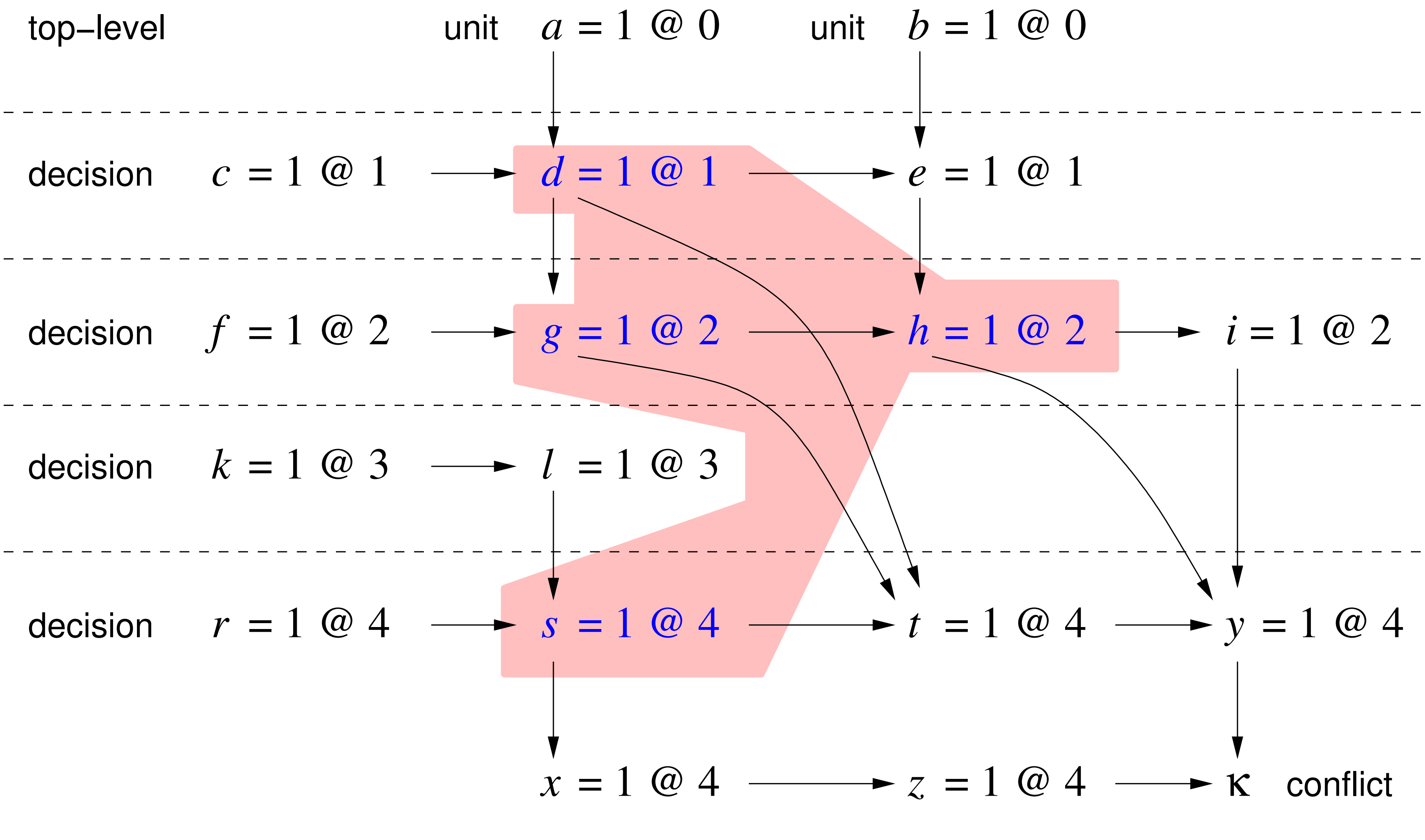
# Locally Minimizing 1st UIP Clause



$$\frac{(\bar{h} \vee i) \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h} \vee \bar{i})}{(\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h})}$$

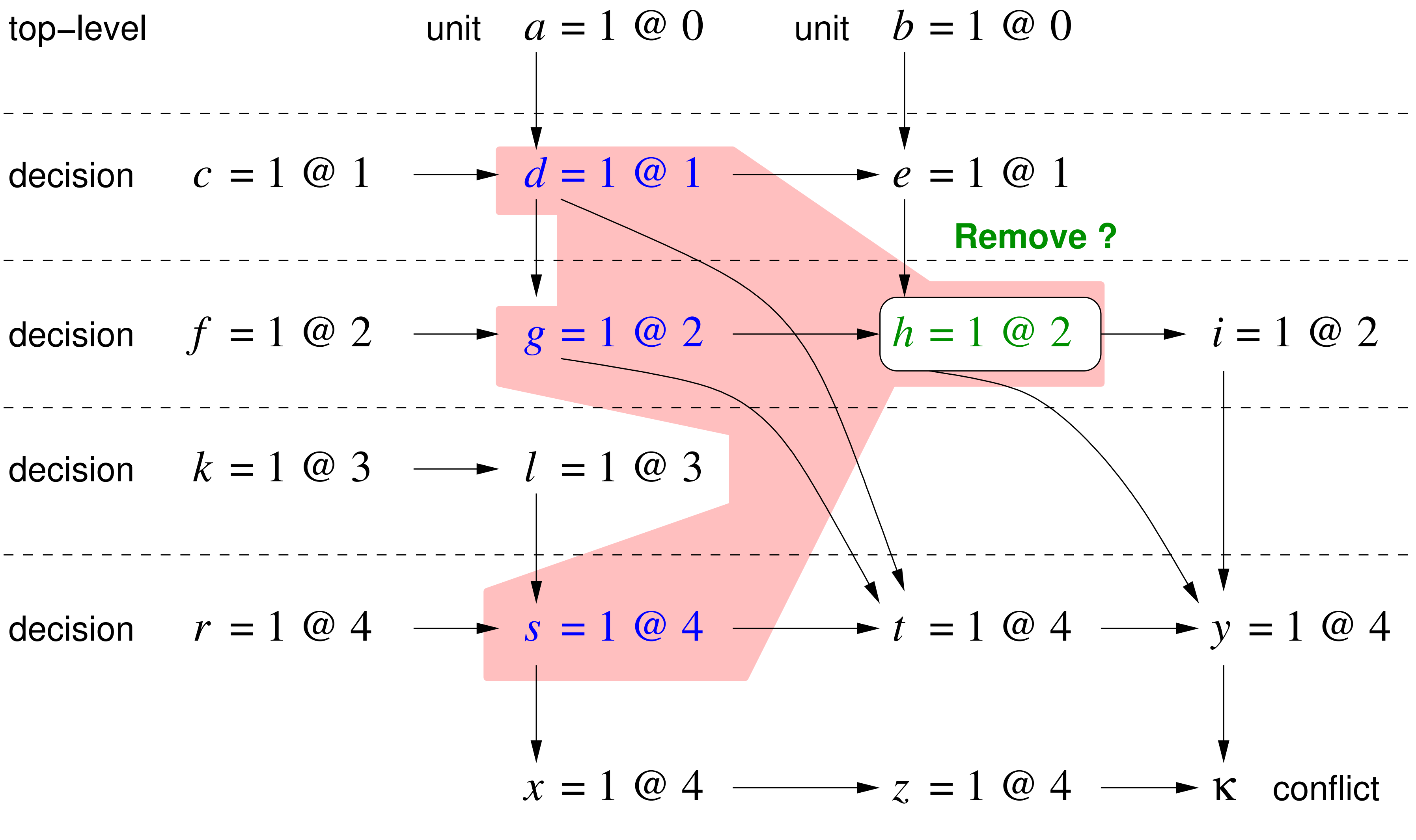
self subsuming resolution

# Locally Minimized Learned Clause



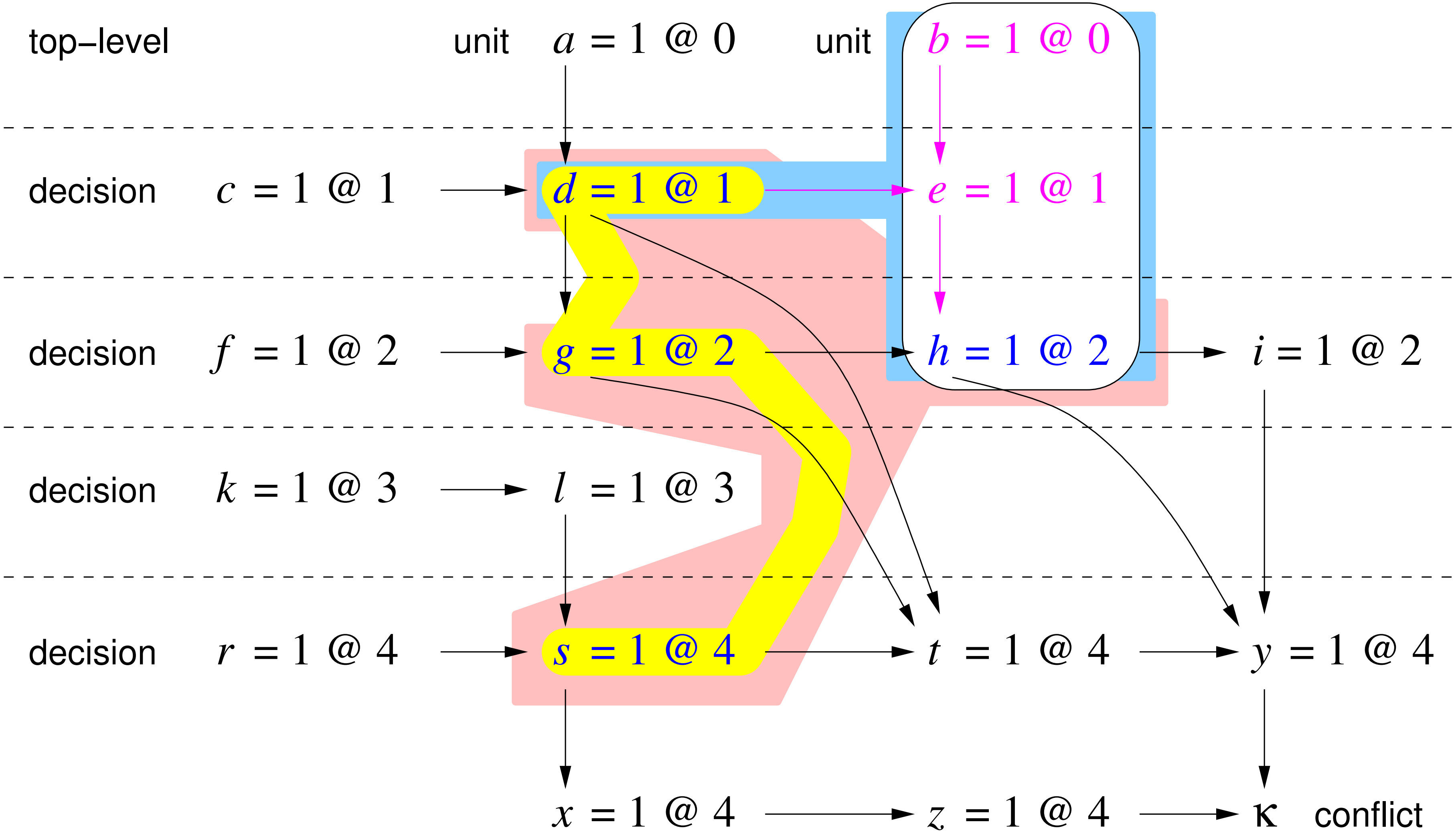
$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \overline{h})$

# Minimizing Locally Minimized Learned Clause Further?



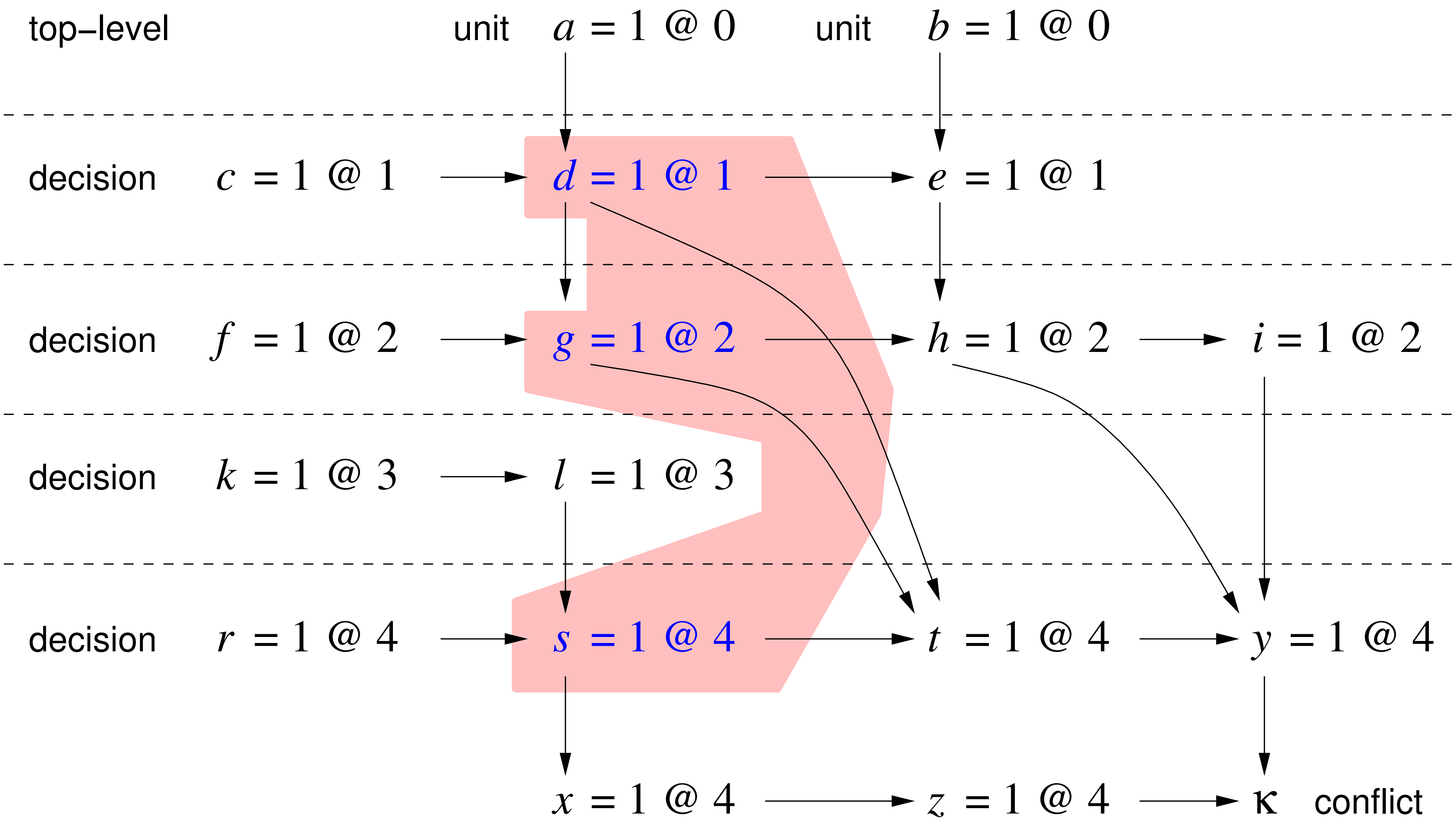
$$(\overline{d} \vee \overline{g} \vee \overline{s} \vee \cancel{\overline{h}})$$

# Recursively Minimizing Learned Clause



$$\frac{(b) \qquad \dfrac{(\bar{d} \vee \bar{b} \vee e) \qquad \dfrac{(\bar{e} \vee \bar{g} \vee h) \qquad (\bar{d} \vee \bar{g} \vee \bar{s} \vee \bar{h})}{(\bar{e} \vee \bar{d} \vee \bar{g} \vee \bar{s})}}{(\bar{b} \vee \bar{d} \vee \bar{g} \vee \bar{s})}}{(\bar{d} \vee \bar{g} \vee \bar{s})}$$

# Recursively Minimized Learned Clause



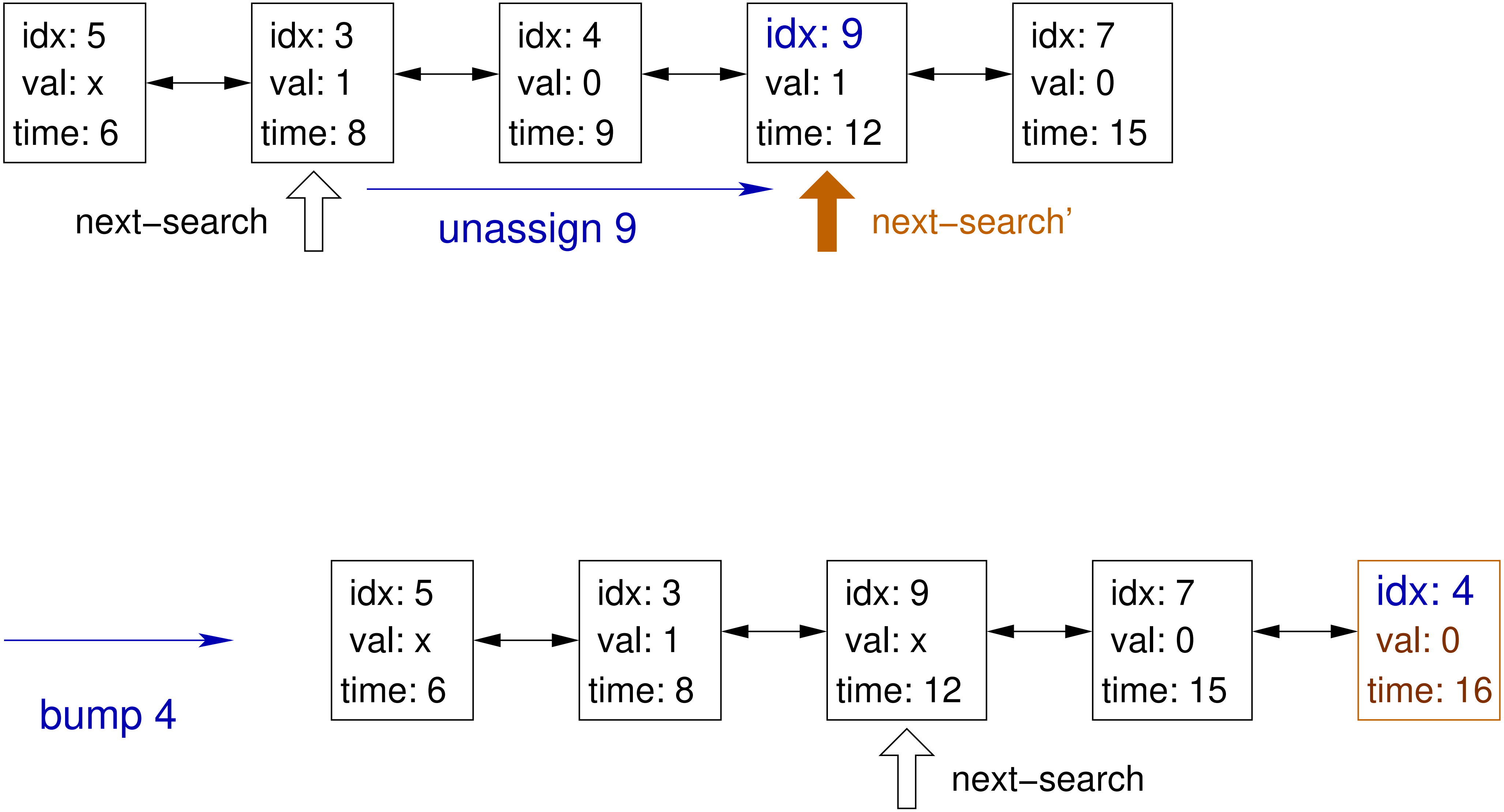
$(\overline{d} \vee \overline{g} \vee \overline{s})$

# Decision Heuristics

- number of variable occurrences in (remaining unsatisfied) clauses (LIS)
  - eagerly satisfy many clauses with many variations studied in the 90ies
  - actually expensive to compute
- dynamic heuristics
  - **focus on variables which were usefull recently in deriving learned clauses**
  - can be interpreted as reinforcement learning
  - started with the VSIDS heuristic [MoskewiczMadiganZhaoZhangMalik’01]
  - most solvers rely on the exponential variant in MiniSAT (EVSIDS)
  - recently showed VMTF as effective as VSIDS [BiereFröhlich-SAT’15] survey
- look-ahead
  - spent more time in selecting good variables (and simplification)
  - related to our Cube & Conquer paper [HeuleKullmanWieringaBiere-HVC’11]
  - “The Science of Brute Force” [Heule & Kullman CACM August 2017]
- EVSIDS during stabilization VMTF otherwise [Biere-SAT-Race-2019]

# Fast VMTF Implementation

- Siege SAT solver [Ryan Thesis 2004] used variable move to front (VMTF)
  - bumped variables moved to head of doubly linked list
  - search for unassigned variable starts at head
  - variable selection is an online sorting algorithm of scores
  - classic “move-to-front” strategy achieves good amortized complexity
- fast simple implementation for caching searches in VMTF [BiereFröhlich’SAT15]
  - doubly linked list does not have positions as an ordered array
  - bump $=$ move-to-front $=$ dequeue then insertion at the head
- time-stamp list entries with “insertion-time”
  - maintained invariant: all variables right of next-search are assigned
  - requires (constant time) update to next-search while unassigning variables
  - occassionally (32-bit) time-stamps will overflow: update all time stamps


idx: 5
val: x
time: 6
idx: 3
val: 1
time: 8
idx: 4
val: 0
time: 9
idx: 9
val: 1
time: 12
idx: 7
val: 0
time: 15
next–search
unassign 9
next–search'
bump 4
idx: 5
val: x
time: 6
idx: 3
val: 1
time: 8
idx: 9
val: x
time: 12
idx: 7
val: 0
time: 15
idx: 4
val: 0
time: 16
next–search

# Variable Scoring Schemes

[BiereFröhlich-SAT'15]

$s$ old score $s'$ new score

| | variable score $s'$ after $i$ conflicts | | |
|---|---|---|---|
| | bumped | not-bumped | |
| STATIC | $s$ | $s$ | static decision order |
| INC | $s+1$ | $s$ | increment scores |
| SUM | $s+i$ | $s$ | sum of conflict-indices |
| VSIDS | $h_i^{256} \cdot s + 1$ | $h_i^{256} \cdot s$ | original implementation in Chaff |
| NVSIDS | $f \cdot s + (1-f)$ | $f \cdot s$ | normalized variant of VSIDS |
| EVSIDS | $s + g^i$ | $s$ | exponential MiniSAT dual of NVSIDS |
| ACIDS | $(s+i)/2$ | $s$ | average conflict-index decision scheme |
| VMTF$_1$ | $i$ | $s$ | variable move-to-front |
| VMTF$_2$ | $b$ | $s$ | variable move-to-front variant |

$0 < f < 1$ $g = 1/f$ $h_i^m = 0.5$ if $m$ divides $i$ $h_i^m = 1$ otherwise

$i$ conflict index $b$ bumped counter

# Basic CDCL Loop

```
int basic_cdcl_loop () {
  int res = 0;

  while (!res)
         if (unsat) res = 20;
    else if (!propagate ()) analyze ();    // analyze propagated conflict
    else if (satisfied ()) res = 10;       // all variables satisfied
    else decide ();                        // otherwise pick next decision

  return res;
}
```

# Reducing Learned Clauses

- keeping all learned clauses slows down BCP — kind of quadratically
  - so SATO and RelSAT just kept only “short” clauses
- better periodically delete “useless” learned clauses
  - keep a certain number of learned clauses — “search cache”
  - if this number is reached MiniSAT reduces (deletes) half of the clauses
  - then maximum number kept learned clauses is increased $\boxed{\text{geometrically}}$
- LBD (glucose level / glue) prediction for usefulness — [AudemardSimon-IJCAI’09]
  - LBD = number of decision-levels in the learned clause
  - allows $\boxed{\text{arithmetic}}$ increase of number of kept learned clauses
  - keep clauses with small LBD forever ( $\leq 2 \ldots 5$)
  - three Tier system by — [Chanseok Oh]
- eagerly reduce hyper-binary resolvents derived in inprocessing

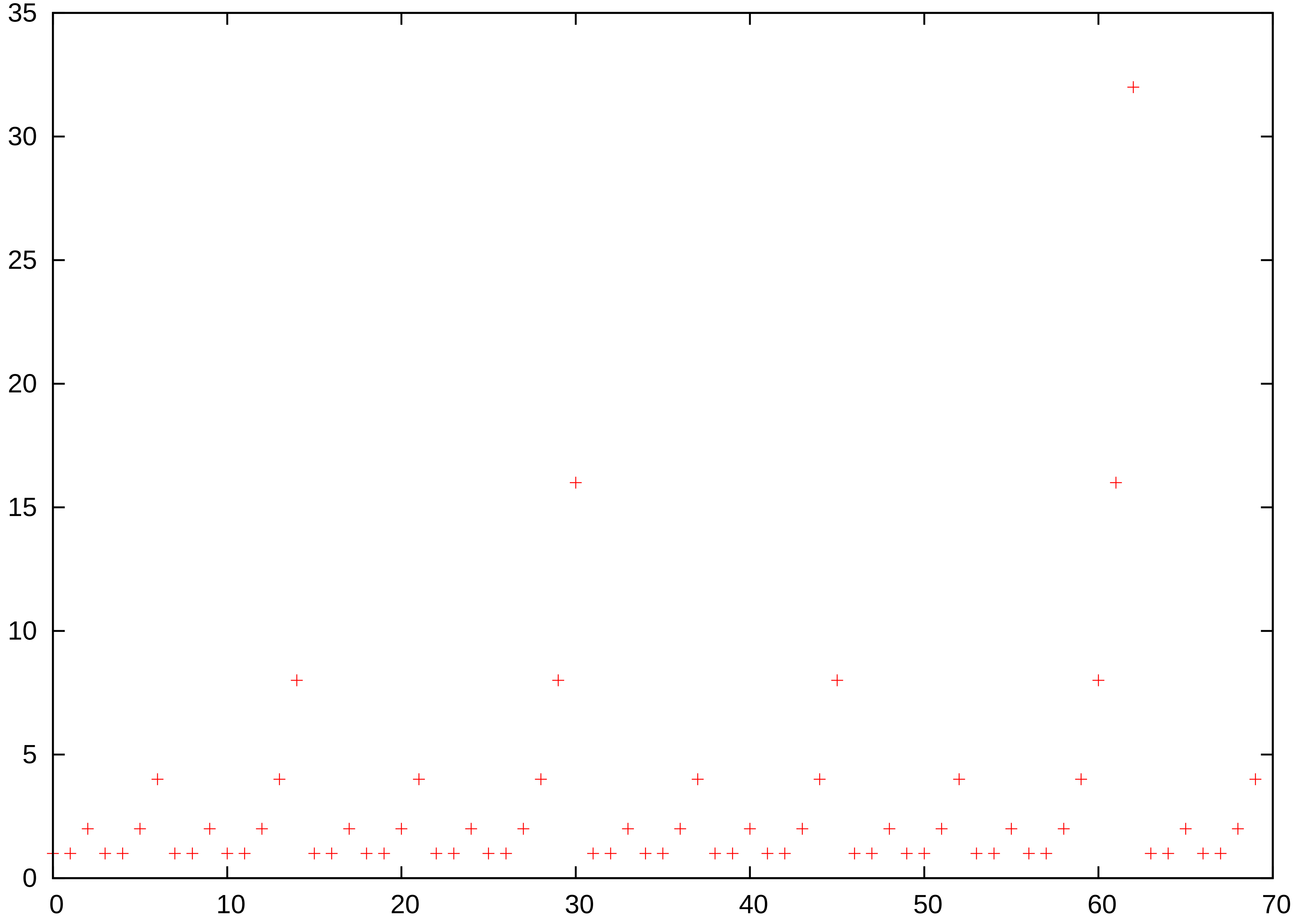
# Restarts

- often it is a good strategy to abandon what you do and restart
  - for satisfiable instances the solver may get stuck in the unsatisfiable part
  - for unsatisfiable instances focusing on one part might miss short proofs
  - restart after the number of conflicts reached a restart limit
- avoid to run into the same dead end
  - by randomization (either on the decision variable or its phase)
  - and/or just keep all the learned clauses during restart
- for completeness dynamically increase restart limit
  - arithmetically, geometrically, Luby, Inner/Outer
- Glucose restarts [AudemardSimon-CP'12]
  - short vs. large window exponential moving average (EMA) over LBD
  - if recent LBD values are larger than long time average then restart
- interleave "stabilizing" (no restarts) and "non-stabilizing" phases [Chanseok Oh]

  call it now "stabilizing mode" and "focused mode"

# Luby’s Restart Intervals

70 restarts in 104448 conflicts



# Luby Restart Scheduling

```
unsigned
luby (unsigned i)
{
  unsigned k;

  for (k = 1; k < 32; k++)
    if (i == (1 << k) - 1)
      return 1 << (k - 1);

  for (k = 1;; k++)
    if ((1 << (k - 1)) <= i && i < (1 << k) - 1)
      return luby (i - (1 << (k-1)) + 1);
}

limit = 512 * luby (++restarts);
...  // run SAT core loop for ’limit’ conflicts
```

# Reluctant Doubling Sequence

[Knuth'12]

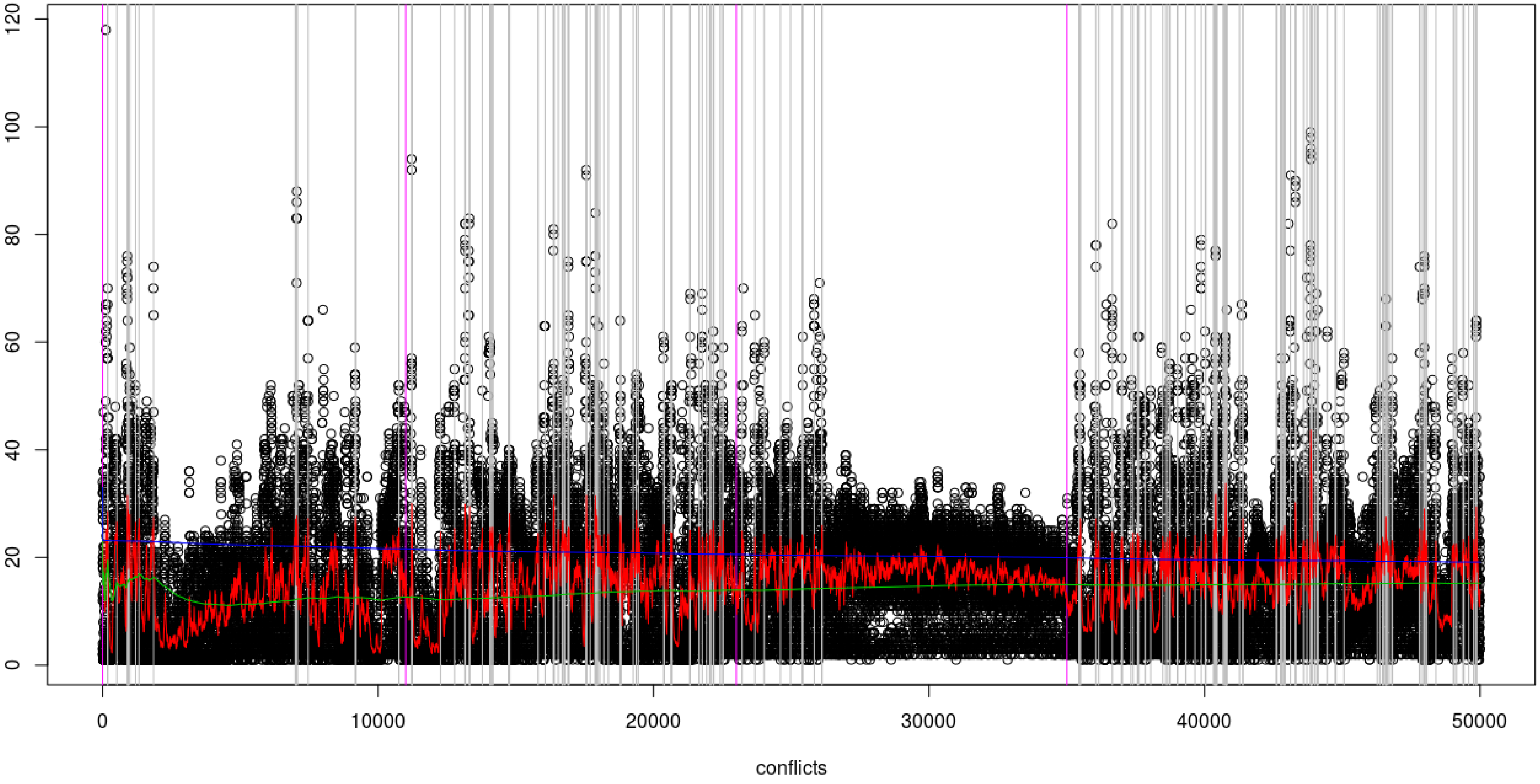
$$\begin{aligned} (u_1, v_1) &= (1, 1) \\ (u_{n+1}, v_{n+1}) &= ((u_n \,\&\, -u_n == v_n) \;?\; (u_n + 1, 1) : (u_n, 2v_n)) \end{aligned}$$

$$(1,1),\ (2,1),\ (2,2),\ (3,1),\ (4,1),\ (4,2),\ (4,4),\ (5,1),\ \ldots$$

# Restart Scheduling with Exponential Moving Averages

[BiereFröhlich-POS'15]

| | | | |
|---|---|---|---|
| ○ | LBD | — | fast *EMA* of LBD with $\alpha = 2^{-5}$ |
| \| | restart | — | slow *EMA* of LBD with $\alpha = 2^{-14}$ (ema-14) |
| \| | inprocessing | — | *CMA* of LBD (average) |



# Phase Saving and Rapid Restarts

- phase assignment:
  - assign decision variable to 0 or 1?
  - "Only thing that matters in satisfiable instances" [Hans van Maaren]
- "phase saving" as in RSat [PipatsrisawatDarwiche'07]
  - pick phase of last assignment (if not forced to, do not toggle assignment)
  - initially use statically computed phase (typically LIS)
  - so can be seen to maintain a **global full assignment**
- rapid restarts
  - varying restart interval with bursts of restarts
  - not only theoretically avoids local minima
  - works nicely together with phase saving
- reusing the trail can reduce the cost of restarts [RamosVanDerTakHeule-JSAT'11]
- target phases of largest conflict free trail / assignment

  [Biere-SAT-Race-2019] [BiereFleury-POS-2020] [CaiZhang-SAT21] [CaiZhangFleuryBiere-JAIR22]

# CDCL Loop with Reduce and Restart

```
int basic_cdcl_loop_with_reduce_and_restart () {

  int res = 0;

  while (!res)
         if (unsat) res = 20;
    else if (!propagate ()) analyze ();    // analyze propagated conflict
    else if (satisfied ()) res = 10;       // all variables satisfied
    else if (restarting ()) restart ();    // restart by backtracking
    else if (reducing ()) reduce ();       // collect useless learned clauses
    else decide ();                        // otherwise pick next decision

  return res;
}
```

# Code from our SAT Solver CaDiCaL

newest Version 1.3.1 from June 18

```
while (!res) {
       if (unsat) res = 20;
  else if (!propagate ()) analyze ();      // propagate and analyze
  else if (iterating) iterate ();          // report learned unit
  else if (satisfied ()) res = 10;         // found model
  else if (search_limits_hit ()) break;    // decision or conflict limit
  else if (terminated_asynchronously ())   // externally terminated
    break;
  else if (restarting ()) restart ();      // restart by backtracking
  else if (rephasing ()) rephase ();       // reset variable phases
  else if (reducing ()) reduce ();         // collect useless clauses
  else if (probing ()) probe ();           // failed literal probing
  else if (subsuming ()) subsume ();       // subsumption algorithm
  else if (eliminating ()) elim ();        // variable elimination
  else if (compacting ()) compact ();      // collect variables
  else if (conditioning ()) condition ();  // globally blocked clauses
  else res = decide ();                    // next decision
}
```

https://github.com/arminbiere/cadical

https://fmv.jku.at/cadical

# Two-Watched Literal Schemes

- original idea from SATO [ZhangStickel'00]
  - invariant: always watch two non-false literals
  - if a watched literal becomes false replace it
  - if no replacement can be found clause is either unit or empty
  - original version used head and tail pointers on Tries
- improved variant from Chaff [MoskewiczMadiganZhaoZhangMalik'01]
  - watch pointers can move arbitrarily SATO: head forward, tail backward
  - no update needed during backtracking
- one watch is enough to ensure correctness but looses arc consistency
- reduces visiting clauses by 10x
  - particularly useful for large and many learned clauses
- blocking literals [ChuHarwoodStuckey'09]
- special treatment of short clauses (binary [PilarskiHu'02] or ternary [Ryan'04])
- cache start of search for replacement [Gent-JAIR'13]

# Parallel SAT

- vector units, GPU, multi-core, cluster, cloud
- application level parallelism usually trivial
- classic work on guiding path principle
- portfolio (with sharing)
- (concurrent) cube & conquer
- control vs. data flow parallelism
- achieve low-level parallelism even though even already BCP is P-complete

⇒ Handbook of Parallel Constraint Reasoning

⇒ still many low-level programming issues left

# 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" lines tell checker to forget clause, 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],
    similar papers for ACL2, Isabelle, …

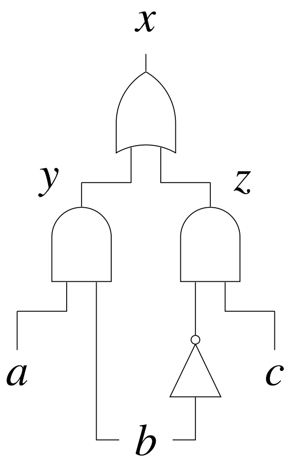
```
  CNF        trace               extended trace      resolution trace   RUP       DRUP

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, Plaisted-Greenbaum Encoding, Monotone Input Removal

[Kullman-DAM'99] [JärvisaloHeuleB-TACAS'10] [JärvisaloHeuleB-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 $(x)$



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

PG encoding drops **upward** propagating clauses of only **positively** occurring gates.
PG encoding drops **downward** propagating clauses of only **negatively** occurring gates.

Unconstrained or monotone inputs can be removed too.

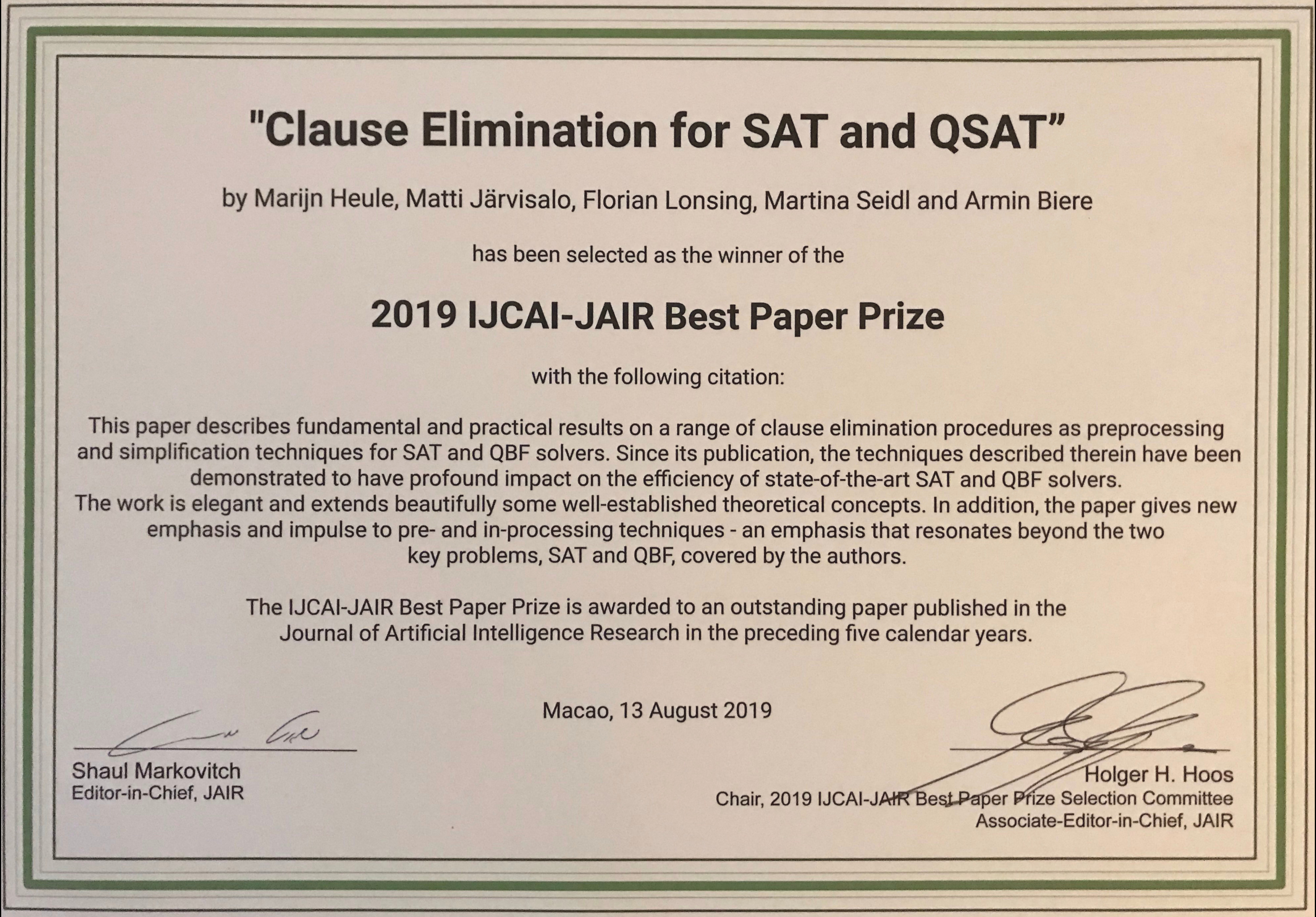
# Resolution Asymmetric Tautologies (RAT)

"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: $(a \lor \boxed{l})$ RAT on $l$ w.r.t. $(a \lor b) \land (l \lor c) \land \underbrace{(\bar{l} \lor b)}_{D}$

  - 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 $(C \backslash L) \cup \bar{L} \cup D$ 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
  - $\Pi_2^P$ complete to chec

Example:

$C = \boxed{a} \vee \boxed{b}$ 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 $\alpha$ blocks a clause $C$ if $\alpha$ 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 $\alpha$ is written as $F\,|\,\alpha$.

**Theorem.**

Let $F$ be a formula, $C$ a clause, and $\alpha$ the assignment blocked by $C$.

Then $C$ is redundant w.r.t. $F$ iff exists an assignment $\omega$ such that

$(i)$ $\omega$ 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

$$\boxed{C \text{ propagation redundant (PR) if exists } \omega \text{ satisfying } C \text{ with } \quad F|\alpha \;\vdash_1\; F|\omega}$$

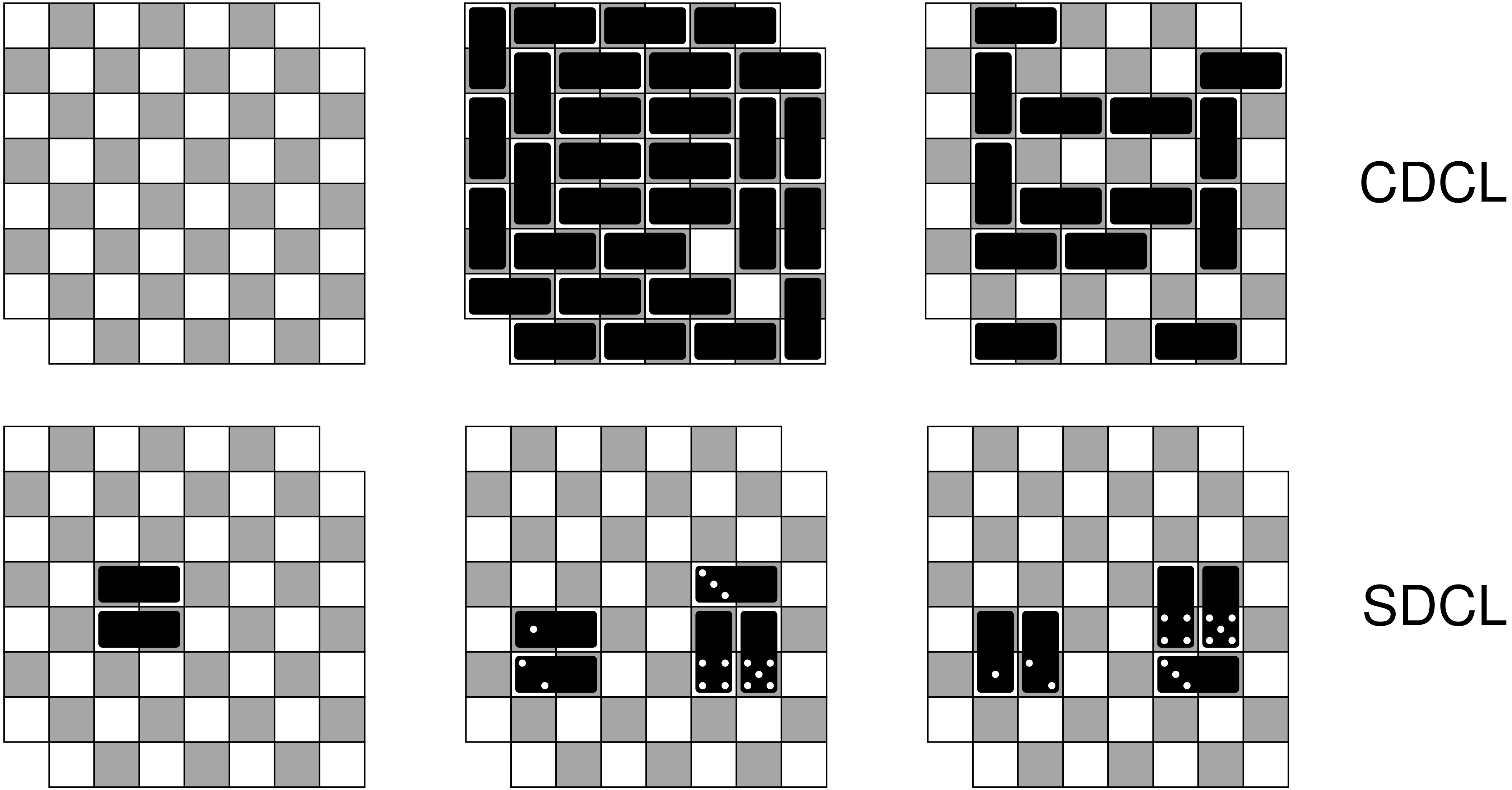
so in essence replacing "$\models$" by "$\vdash_1$" (implied by unit propagation)

where again $\alpha$ 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] nom. 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]
  but PR and can not simulate covered clauses [BarnettCernaBiere-IJCAR'20]

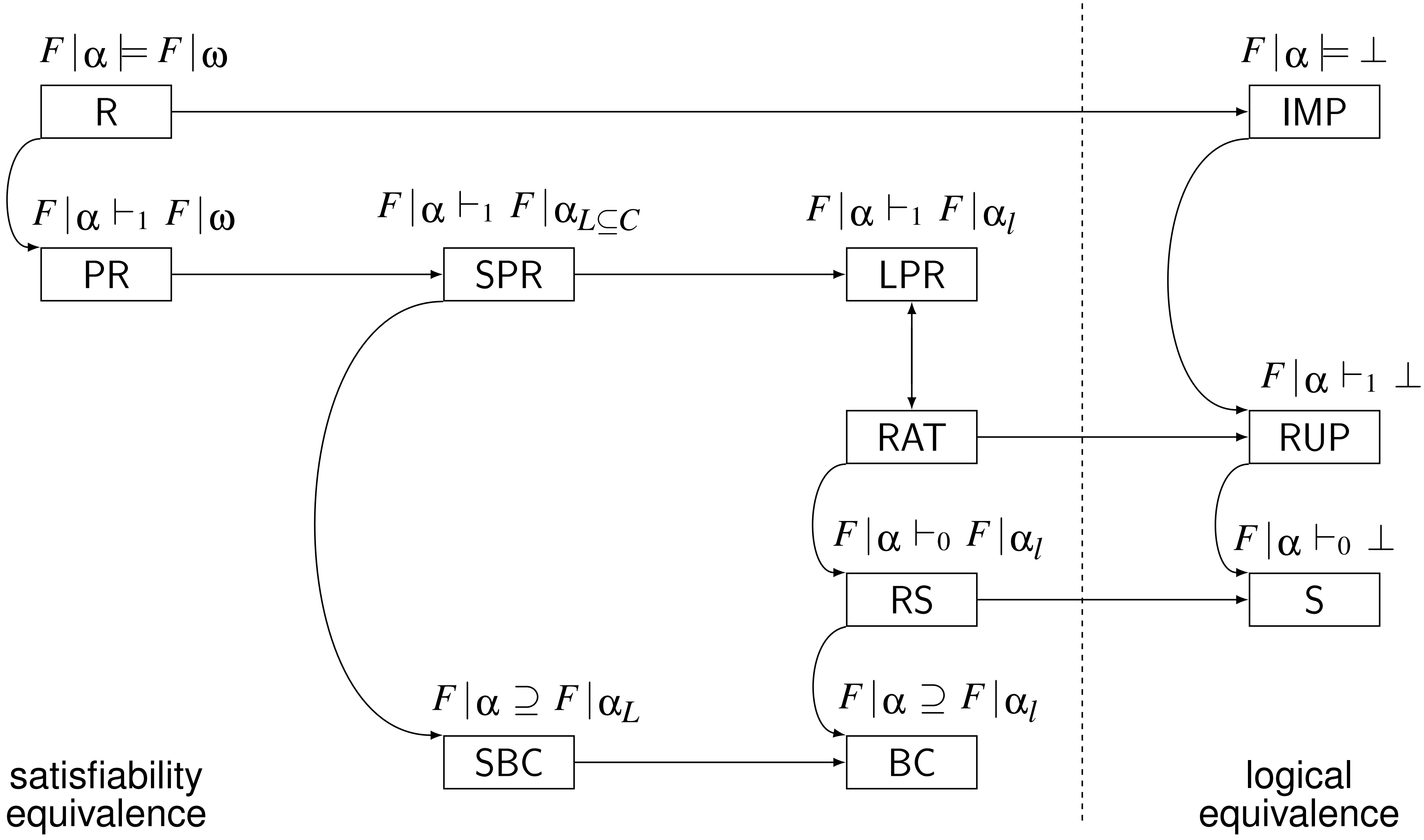
# Mutilated Chessboard

[HeuleKieslBiere-NFM'19]



# Landscape of Clausal Redundancy

[HeuleKieslBiere-JAR'19]



$F\,|\,\alpha \models F\,|\,\omega$

R

$F\,|\,\alpha \models \bot$

IMP

$F\,|\,\alpha \vdash_1 F\,|\,\omega$

PR

$F\,|\,\alpha \vdash_1 F\,|\,\alpha_{L\subseteq C}$

SPR

$F\,|\,\alpha \vdash_1 F\,|\,\alpha_l$

LPR

$F\,|\,\alpha \vdash_1 \bot$

RAT

RUP

$F\,|\,\alpha \vdash_0 F\,|\,\alpha_l$

RS

$F\,|\,\alpha \vdash_0 \bot$

S

$F\,|\,\alpha \supseteq F\,|\,\alpha_L$

SBC

$F\,|\,\alpha \supseteq F\,|\,\alpha_l$

BC

satisfiability equivalence

logical equivalence

```
CDCL(formula $F$)
$\alpha := \emptyset$
forever do
    $\alpha :=$ UnitPropagate$(F, \alpha)$
    if $\alpha$ falsifies a clause in $F$ then
        $C :=$ AnalyzeConflict$()$
        $F := F \wedge C$
        if $C$ is the empty clause $\perp$ then return UNSAT
        $\alpha :=$ BackJump$(C, \alpha)$




    else
        if all variables are assigned then return SAT
        $l :=$ Decide$()$
        $\alpha := \alpha \cup \{l\}$
```

```
SDCL(formula $F$)
$\alpha := \emptyset$
forever do
    $\alpha :=$ UnitPropagate$(F, \alpha)$
    if $\alpha$ falsifies a clause in $F$ then
        $C :=$ AnalyzeConflict$()$
        $F := F \wedge C$
        if $C$ is the empty clause $\perp$ then return UNSAT
        $\alpha :=$ BackJump$(C, \alpha)$
    else if the pruning predicate $P_\alpha(F)$ is satisfiable then
        $C :=$ AnalyzeWitness$()$
        $F := F \wedge C$
        $\alpha :=$ BackJump$(C, \alpha)$
    else
        if all variables are assigned then return SAT
        $l :=$ Decide$()$
        $\alpha := \alpha \cup \{l\}$
```

# Positive and Filtered Positive Reduct

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

In the positive reduct consider clauses satisfied by $\alpha$, unassigned literals and add $C$:

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

**Theorem.** Let $F$ be a formula, $\alpha$ an assignment, and $C$ the clause that blocks $\alpha$. Then, $C$ is $\boxed{\text{SBC}}$ by an $L \subseteq C$ with respect to $F$ if and only if the assignment $\alpha_L$ 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 $\alpha$ an assignment. Then, the filtered positive reduct of $F$ and $\alpha$ is the formula $G \wedge C$ where $G = \{\text{touched}_\alpha(D) \mid D \in F \text{ and } F|_\alpha \nvdash_1 \text{untouched}_\alpha(D)\}$.

**Theorem.** Let $F$ be a formula, $\alpha$ an assignment, and $C$ the clause that blocks $\alpha$. Then, $C$ is $\boxed{\text{SPR}}$ by an $L \subseteq C$ with respect to $F$ if and only if the assignment $\alpha_L$ 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] | 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 |

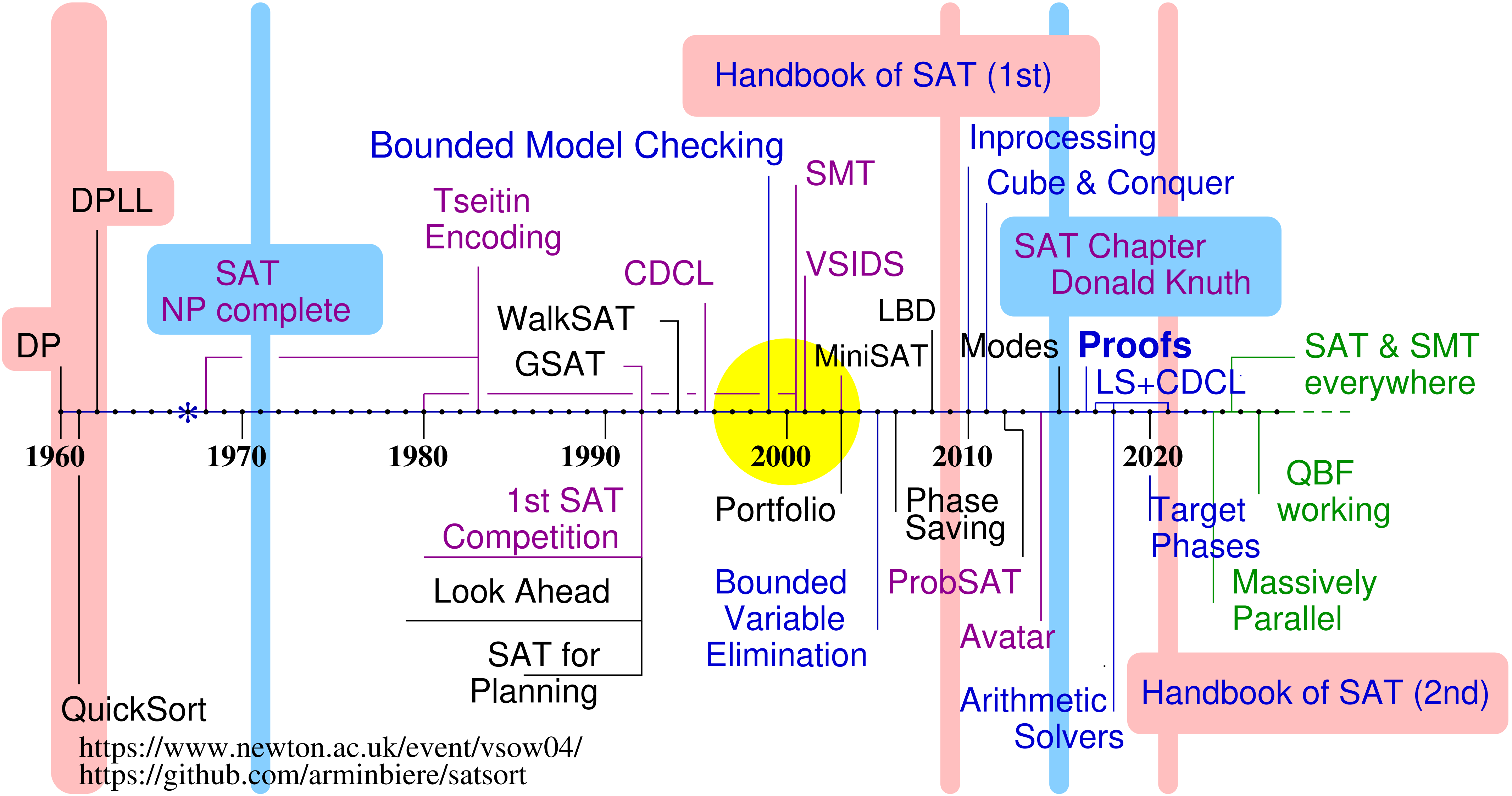
# Further things we could discuss …

- relation to proof complexity Banff, Fields, Dagstuhl seminars
- extensions formalisms: QBF, Pseudo-Boolean, #SAT, …
- local search this year's best solvers have all local search in it
- challenges: arithmetic reasoning (and proofs)

  best paper [KaufmannBiereKauers-FMCAD'17] [PhD thesis Daniela Kaufmann 2020]
- chronological backtracking [RyvchinNadel-SAT'18] [MöhleBiere-SAT'19]
- incremental SAT solving

  best student paper [FazekasBiereScholl-SAT'19] [PhD thesis of Katalin Fazekas in 2020]
- parallel and distributed SAT solving Handbook of Parallel Constraint Reasoning, …
- and probably many more …

# Personal SAT Solver History



https://www.newton.ac.uk/event/vsow04/
https://github.com/arminbiere/satsort