Encoding into SAT

Armin Biere

JOHANNES KEPLER
UNIVERSITÄT LINZ

SAT/SMT/AR Summer School 2018

Manchester, UK

July 3, 2018

Dress Code Summer School Speaker as SAT Problem

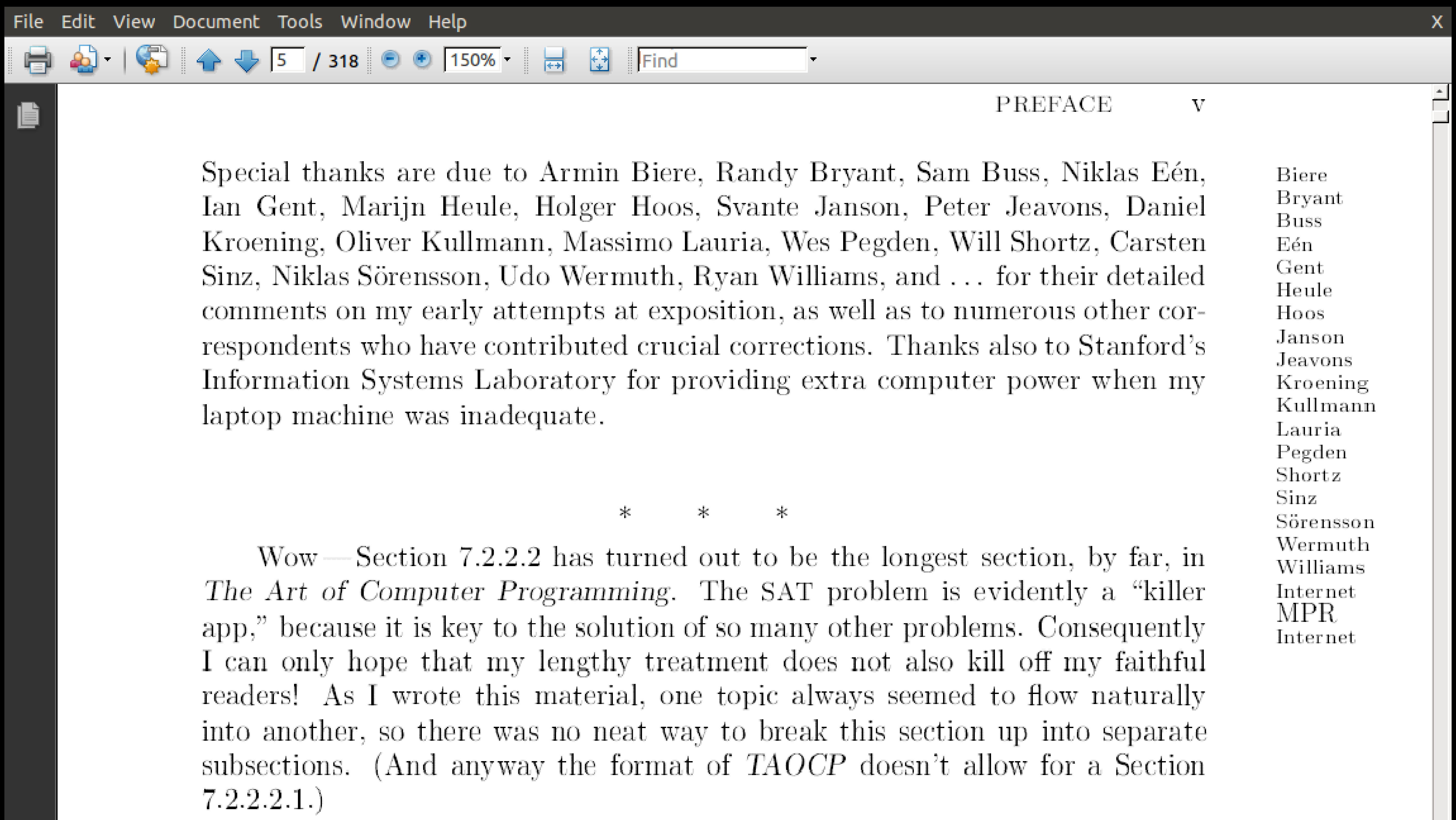
- propositional logic:
  - variables **tie** **shirt**
  - negation $\neg$ (not)
  - disjunction $\lor$ (or)
  - conjunction $\land$ (and)
- clauses (conditions / constraints)

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

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

$$(\neg\textbf{tie} \lor \textbf{shirt}) \land (\textbf{tie} \lor \textbf{shirt}) \land (\neg\textbf{tie} \lor \neg\textbf{shirt})$$


SAT-Race 2010 Award
"Plingeling"
by Armin Biere
is awarded the title of
Best Parallel Solver
SAT-Race 2010
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 organize
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 sr15baq
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
Parallel
2nd Prize
Awarded
Plingeling sr1
Written b
Armin B
Configurable SAT Solver Challenge 2014
SAT-Race 2010
Best Parallel Solver
2010
SAT Competition 2009
SAT+UNSAT
SAT COMPETITION 2013
Twelfth International Conference
On Theory and Applications of Satisfiability Testing
SAT
Competition 200
Silver meda
Twelfth International Conference
Theory and Applications of Satisfiabi
International Symposium
Applications of Satisfiability Testing
Delft
2002



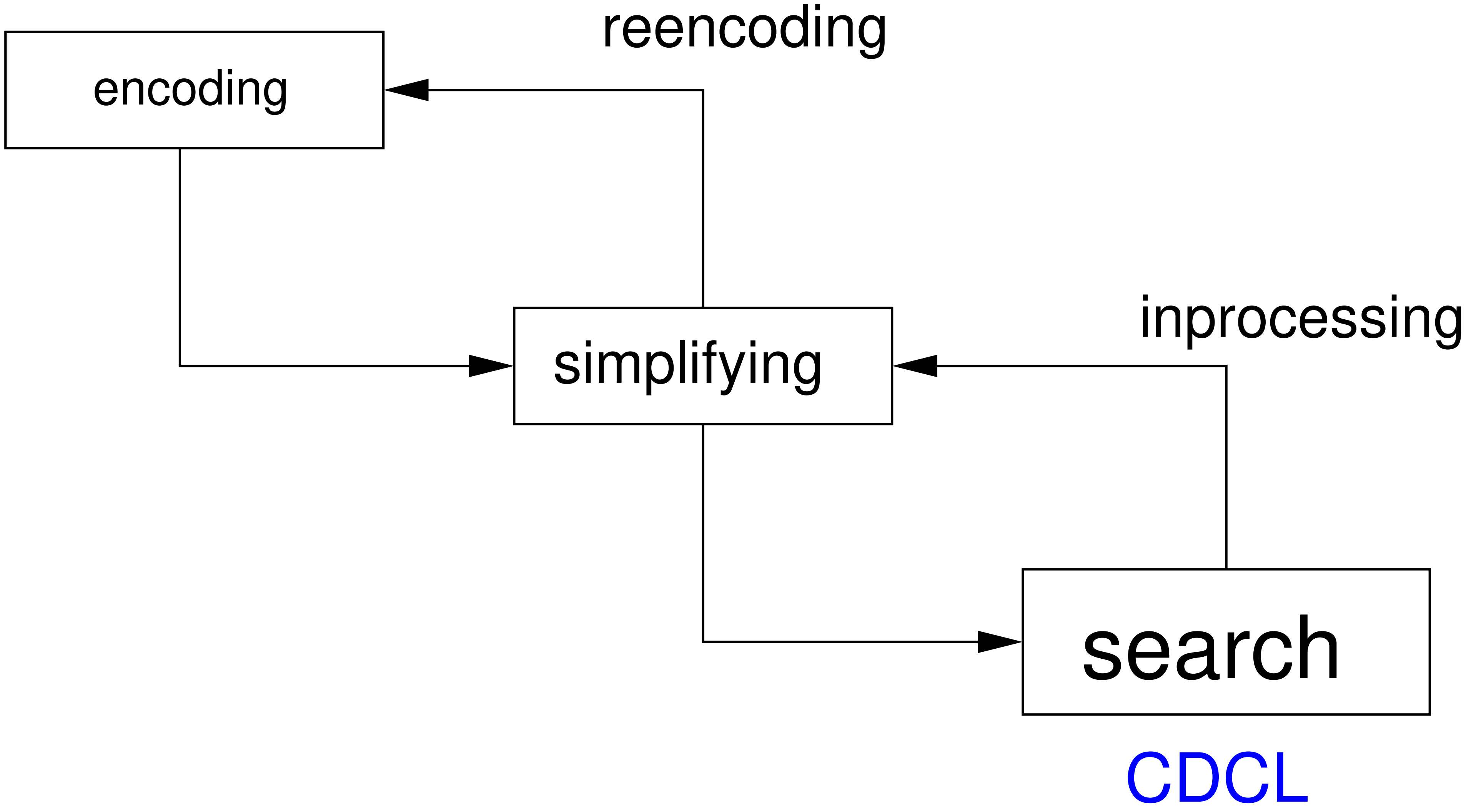
Special thanks are due to Armin Biere, Randy Bryant, Sam Buss, Niklas Eén, Ian Gent, Marijn Heule, Holger Hoos, Svante Janson, Peter Jeavons, Daniel Kroening, Oliver Kullmann, Massimo Lauria, Wes Pegden, Will Shortz, Carsten Sinz, Niklas Sörensson, Udo Wermuth, Ryan Williams, and . . . for their detailed 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.

* * *

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.)

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

What is Practical SAT Solving?



reencoding

encoding

simplifying

inprocessing

search

CDCL

Schur(2) = 4 5

see also: Marijn Heule, “Schur Number Five”, AAAI’18.

Schur(k) = n iff

n is the largest number such that
$N = \{1, \dots, n\}$ can be colored with k colors
and if $i + j = k$ for $i, j, k \in N$ then they do not have the same color

such equations are not monochromatic

$$1 + 1 = 2 \qquad 2 + 2 = 4 \qquad 1 + 3 = 4$$

$$1 + 4 = 5 \qquad 2 + 3 = 5$$

encoded in SAT (x = 1 iff x is colored with first color)

$$(\bar{x}_1 \vee \bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_1 \vee x_2) \;\wedge\; (\bar{x}_2 \vee \bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_2 \vee x_4) \;\wedge\; (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (x_1 \vee x_3 \vee x_4)$$

$$(\bar{x}_1 \vee \bar{x}_4 \vee \bar{x}_5) \wedge (x_1 \vee x_4 \vee x_5) \;\wedge\; (\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_5) \wedge (x_2 \vee x_3 \vee x_5)$$

DIMACS Encoder for Schur(2)

$$1 + 3 = 4$$

```
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char ** argv) {
  int n = argc > 1 ? atoi (argv[1]) : 5;
  printf ("p cnf %d 0\n", n);  // FIXME
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++) {
      int k = i + j;
      if (k <= n)
        printf ("%d %d %d 0\n", i, j, k),
        printf ("%d %d %d 0\n", -i, -j, -k);
    }
  return 0;
}
```

see also: Heule, Kullmann, Marek, “Solving and Verifying the boolean Pythagorean Triples problem via Cube-and-Conquer”, SAT’16.

DIMACS Encoder for Pythagorean Triples Problem

$$3^2 + 4^2 = 5^2$$

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char ** argv) {
  int n = argc > 1 ? atoi (argv[1]) : 7825;
  printf ("p cnf %d 0\n", n);  // FIXME
  for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++) {
      int k = sqrt (i*i + j*j);
      if (k <= n && i*i + j*j == k*k)
        printf ("%d %d %d 0\n", i, j, k),
        printf ("%d %d %d 0\n", -i, -j, -k);
    }
  return 0;
}
```

see also: Marijn Heule, “Schur Number Five”, AAAI’18.

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

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

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

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

satisfying assignment gives counter-example to equivalence

Negation Normal Form

Assumption: we only have conjunction, disjunction and negation as operators.

Formula is in Negation Normal Form (NNF),
if negations only occur in front of variables

$\Rightarrow$ all internal nodes in the formula tree are either ANDs or ORs

linear algorithms for generating NNF from an arbitrary formula

NNF generations includes elimination of non-monotonic operators (XOR, XNOR)

$$\text{NNF of} \quad f \leftrightarrow g \quad \text{is NNF of} \quad f \wedge g \vee \overline{f} \wedge \overline{g}$$

in this case the result can be exponentially larger without sharing
(parity, counting, adders, ...).

NNF Algorithm

```
Formula formula2nnf (Formula f, Boole sign)
{
  if (is_variable (f))
    return sign ? new_not_node (f) : f;

  if (op (f) == AND || op (f) == OR)
    {
      l = formula2nnf (left_child (f), sign);
      r = formula2nnf (right_child (f), sign);
      flipped_op = (op (f) == AND) ? OR : AND;
      return new_node (sign ? flipped_op : op (f), l, r);
    }
  else
    {
      assert (op (f) == NOT);
      return formula2nnf (child (f), !sign);
    }
}
```

Simple Translation of Formula into CNF

```
Formula formula2cnf (Formula f)
{
  if (is_cnf (f)) return f;

  if (op (f) == AND)
    {
      l = formula2cnf (left_child (f));
      r = formula2cnf (right_child (f));
      return new_node (AND, l, r);
    }
  else
    {
      assert (op (f) == OR);
      l = formula2cnf (left_child (f));
      r = formula2cnf (right_child (f));
      return merge_cnf (l, r);
    }
}
```

Merging two CNFs

```
Formula merge_cnf (Formula f, Formula g)
{
  res = new_constant_node (TRUE);
  for (c = first_clause (f); c; c = next_clause (f, c))
    for (d = first_clause (g); d; d = next_clause (g, d))
      res = new_node (AND, res, new_node (OR, c, d));
  return res;
}
```

$$\mathtt{f} \quad \vee \quad \mathtt{g} \qquad \equiv \qquad \bigwedge_{i=1}^{m} C_i \quad \vee \quad \bigwedge_{j=1}^{n} D_i \qquad \equiv \qquad \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{n} (C_i \vee D_j)$$

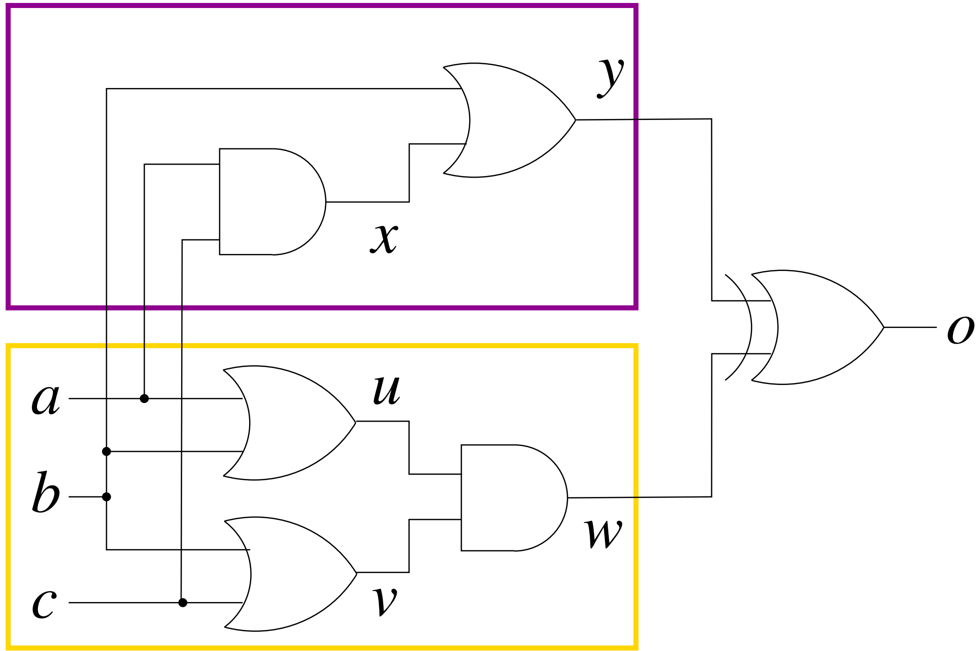
Formula to NNF to CNF

```
Formula encode (Formula f)
{
  Formula nnf = formula2nnf (f, 0);   // cheap
  Formula cnf = formula2cnf (nnf);    // expensive
  return cnf;
}
```

- NNF translation is linear (even for circuits)
- one merge operation is already quadratic $\Rightarrow$ NNF to CNF exponential
- whole "Formula to NFF to CNF" flow exponential
  - exponential in the number of alternations between OR and AND
- sometimes compact: $(a \wedge c) \vee b \quad \equiv \quad (a \vee b) \wedge (c \vee b)$

preprocessing!

Tseitin Transformation: Circuit to CNF



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

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

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

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:

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

Improving on Tseitin Encoding

- “cone of influence reduction” removes unrelated sub-formulas
- flatten associative operators (AND, OR) into multi-arity operators
  - without destroying sharing thus only applied to trees ⇒ variable elimination
- structural hashing of common sub-expression ⇒ AIGs
- switch to NNF based encoding close to leafs of the formula / circuit
  - as proposed by [Boy de la Tour’90] ⇒ variable elimination
  - but see also NiceDAGs [ChambersManoliosVroon’09]
- polarity based encoding
  - first described by [PlaistedGreenbaum’86]
  - can save half of the clauses ⇒ blocked clause elimination
- technolog based encoding [EénMishchenkoSörensson’07]
  - use heavy-weight circuit optimization for circuit chunks

Most of these Optimizations can be achieved by Preprocessing!

Example for Checking Aliasing

<u>original</u>

```
assert (i != k);
a[i] = a[k];
a[j] = a[k];
```

<u>optimized</u>

```
int t = a[k];
a[i] = t;
a[j] = t;
```

$i \neq k$	$t = \text{read}(a, k)$	
$b_1 = \text{write}(a, i, t)$	$c_1 = \text{write}(a, i, t)$	static-single assignment (SSA)
$b_2 = \text{write}(b_1, j, s)$	$c_2 = \text{write}(c_1, j, t)$	
$s = \text{read}(b_1, k)$		

<u>original</u> $\neq$ <u>optimized</u> iff $b_2 \neq c_2$

$b_2 \neq c_2$ iff $\exists l$ with $\text{read}(b_2, l) \neq \text{read}(c_2, l)$

Aliasing Example Continued 1

thus $\underline{\text{original}} \neq \underline{\text{optimized}}$ iff

$$\begin{aligned}
&i \neq k \\
&t = \text{read}(a,k) \\
&b_1 = \text{write}(a,i,t) \\
&b_2 = \text{write}(b_1,j,s) \\
&c_1 = \text{write}(a,i,t) \\
&c_2 = \text{write}(c_1,j,t) \\
&s = \text{read}(b_1,k) \\
&\text{read}(b_2,l) \neq \text{read}(c_2,l)
\end{aligned}$$

satisfiable

Aliasing Example Continued 2

thus $\underline{\text{original}} \neq \underline{\text{optimized}}$ iff

$$
\begin{aligned}
&i \neq k \\
&t = \text{read}(a, k) \\
&b_1 = \text{write}(a, i, t) \\
&b_2 = \text{write}(b_1, j, s) \\
&c_1 = \text{write}(a, i, t) \\
&c_2 = \text{write}(c_1, j, t) \\
&s = \text{read}(b_1, k) \\
&u = \text{read}(b_2, l) \\
&v = \text{read}(c_2, l) \\
&u \neq v
\end{aligned}
$$

satisfiable

Aliasing Example Continued 3

after eliminating c_2

$i \neq k$
$t = \text{read}(a, k)$
$b_1 = \text{write}(a, i, t)$
$b_2 = \text{write}(b_1, j, s)$
$c_1 = \text{write}(a, i, t)$
$c_2 = \text{write}(c_1, j, t)$
$s = \text{read}(b_1, k)$
$u = \text{read}(b_2, l)$
$v = (i = j \;\; ? \;\; t : \text{read}(c_1, l))$
$u \neq v$

Aliasing Example Continued 4

after eliminating c_2, c_1

$i \neq k$
$t = \text{read}(a,k)$
$b_1 = \text{write}(a,i,t)$
$b_2 = \text{write}(b_1, j, s)$
$c_1 = \text{write}(a,i,t)$
$c_2 = \text{write}(c_1, j, t)$
$s = \text{read}(b_1,k)$
$u = \text{read}(b_2,l)$
$v = (l = j \;\; ? \;\; t : (l = i \;\; ? \;\; t : \text{read}(a,l)))$
$u \neq v$

Aliasing Example Continued 5

after eliminating c_2, c_1, b_2

$i \neq k$

$t = \text{read}(a, k)$

$b_1 = \text{write}(a, i, t)$

$b_2 = \text{write}(b_1, j, s)$

$c_1 = \text{write}(a, i, t)$

$c_2 = \text{write}(c_1, j, t)$

$s = \text{read}(b_1, k)$

$u = (l = j \ \ ? \ \ s : \text{read}(b_1, l))$

$v = (l = j \ \ ? \ \ t : (l = i \ \ ? \ \ t : \text{read}(a, l)))$

$u \neq v$

Aliasing Example Continued 6

after eliminating c_2, c_1, b_2, b_1

$i \neq k$
$t = \text{read}(a, k)$
$b_1 = \text{write}(a, i, t)$
$b_2 = \text{write}(b_1, j, s)$
$c_1 = \text{write}(a, i, t)$
$c_2 = \text{write}(c_1, j, t)$
$s = (k = i \ ? \ t : \text{read}(a, k))$
$u = (l = j \ ? \ s : (l = i \ ? \ t : \text{read}(a, l)))$
$v = (l = j \ ? \ t : (l = i \ ? \ t : \text{read}(a, l)))$
$u \neq v$

Aliasing Example Continued 7

result after “write” elimination

$$
\begin{aligned}
&i \neq k \\
&t = \mathsf{read}(a, k) \\
&s = (k = i \;\;?\;\; t : \mathsf{read}(a, k)) \\
&u = (l = j \;\;?\;\; s : (l = i \;\;?\;\; t : \mathsf{read}(a, l))) \\
&v = (l = j \;\;?\;\; t : (l = i \;\;?\;\; t : \mathsf{read}(a, l))) \\
&u \neq v
\end{aligned}
$$

Aliasing Example Continued 8

after eliminating conditionals (if-then-else)

$$
\begin{aligned}
&i \neq k\\
&t = \mathsf{read}(a,k)\\
&k = i \;\rightarrow\; s = t\\
&k \neq i \;\rightarrow\; s = \mathsf{read}(a,k)\\
&l = j \;\rightarrow\; u = s\\
&l \neq j \wedge l = i \;\rightarrow\; u = t\\
&l \neq j \wedge l \neq i \;\rightarrow\; u = \mathsf{read}(a,l)\\
&l = j \;\rightarrow\; v = t\\
&l \neq j \wedge l = i \;\rightarrow\; v = t\\
&l \neq j \wedge l \neq i \;\rightarrow\; v = \mathsf{read}(a,l)\\
&u \neq v
\end{aligned}
$$

now treat “read” as uninterpreted function (say f)

Aliasing Example Continued 9

after "Ackermanization" using $x = \text{read}(a,k), \quad y = \text{read}(a,l)$

$$
\begin{array}{l}
i \neq k \\
t = x \\
k = i \;\rightarrow\; s = t \\
k \neq i \;\rightarrow\; s = x \\
l = j \;\rightarrow\; u = s \\
l \neq j \wedge l = i \;\rightarrow\; u = t \\
l \neq j \wedge l \neq i \;\rightarrow\; u = y \\
l = j \;\rightarrow\; v = t \\
l \neq j \wedge l = i \;\rightarrow\; v = t \\
l \neq j \wedge l \neq i \;\rightarrow\; v = y \\
u \neq v \\
k = l \;\rightarrow\; x = y
\end{array}
$$

10 variables remain

use 4-bit bitvectors to encode $0..15$

Bit-Blasting of Bit-Vector Equality

equality $x = y$ of 4-bit bitvectors x, y as new literal $\ell_{x=y}$

$$[x_3, x_2, x_1, x_0]_4 = [y_3, y_2, y_1, y_0]_4$$

$$\ell_{x=y} \quad \leftrightarrow \quad \bigwedge_{i=0}^{3} x_i \leftrightarrow y_i$$

and then use Tseitin encoding

Bit-Blasting of Bit-Vector Addition

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

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

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

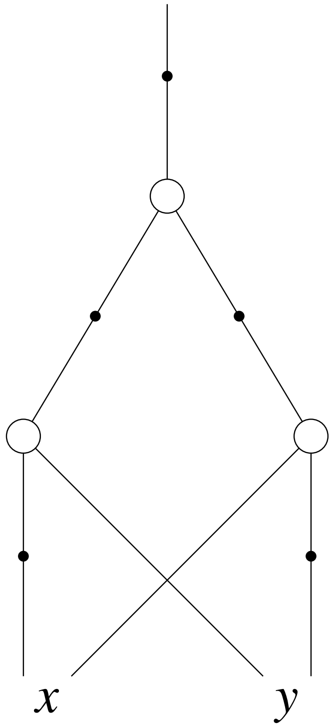
where

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

Intermediate Representations

- encoding directly into CNF is hard, so we use intermediate levels:
  1. application level (SSA, encoding execution semantics)
  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

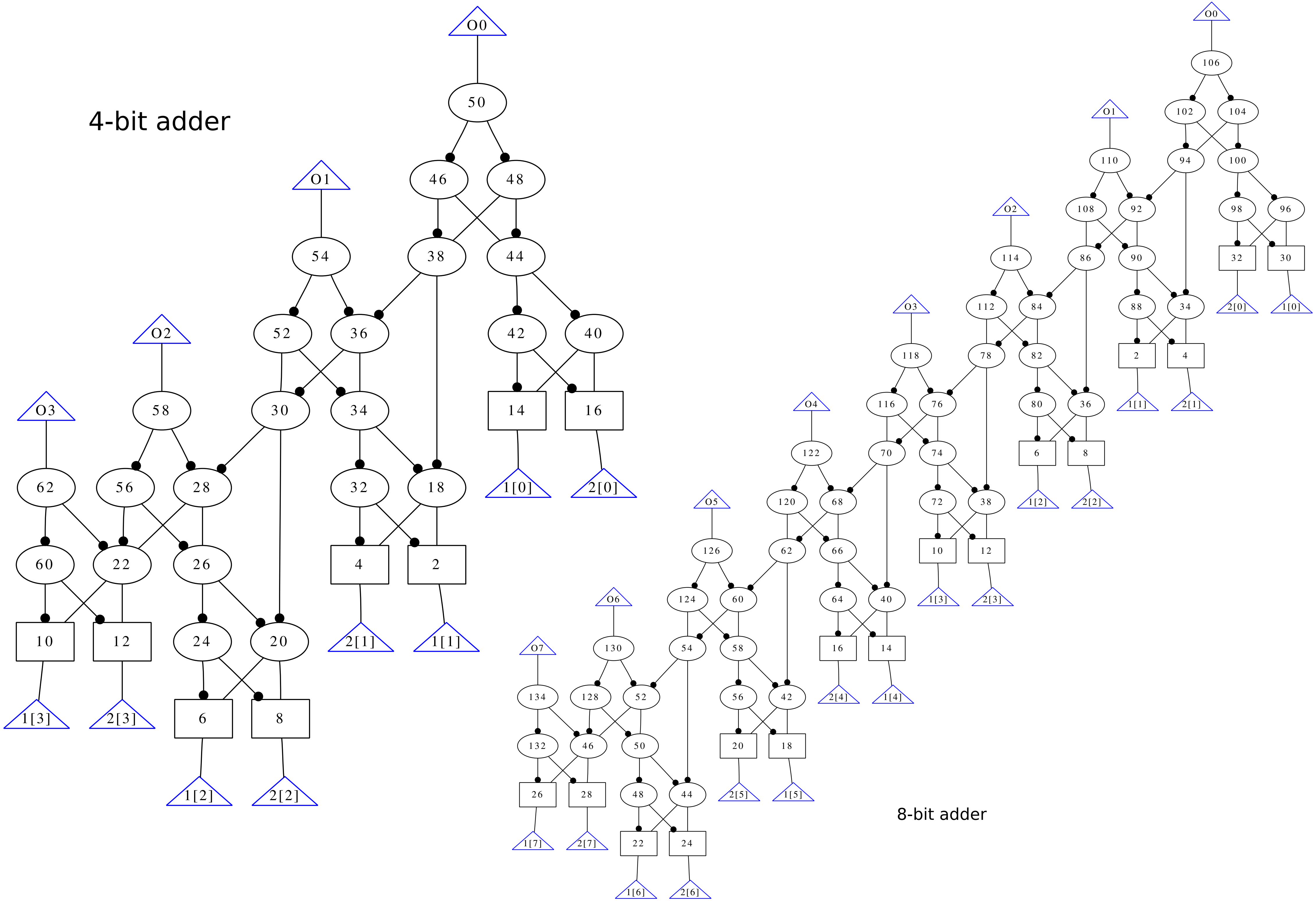
XOR as AIG



negation/sign are edge attributes

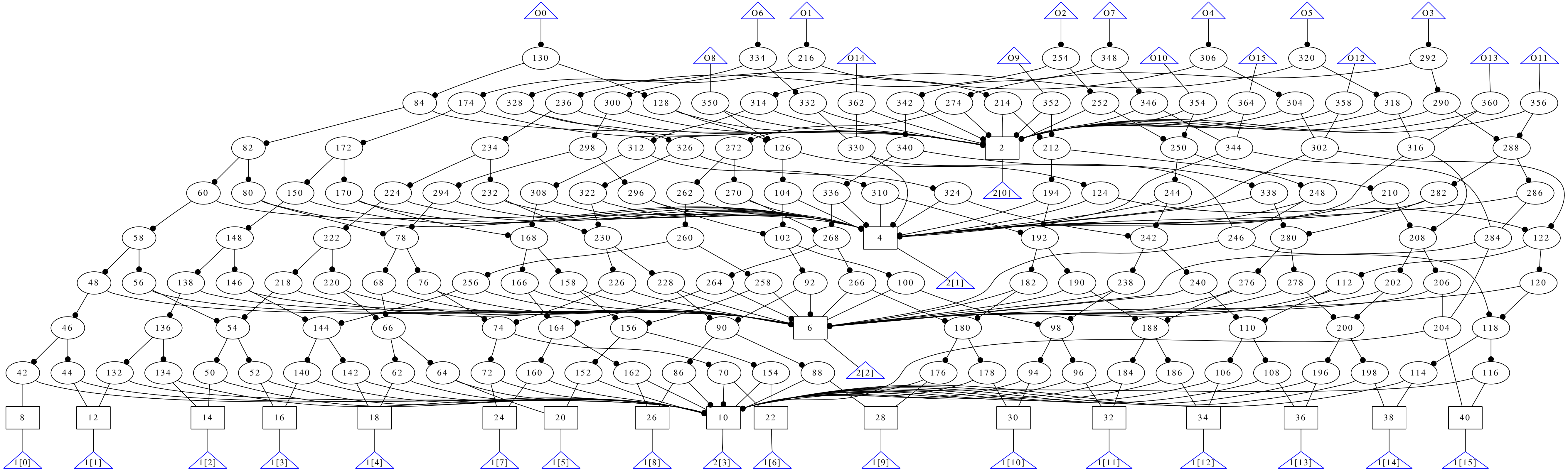
not part of node

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

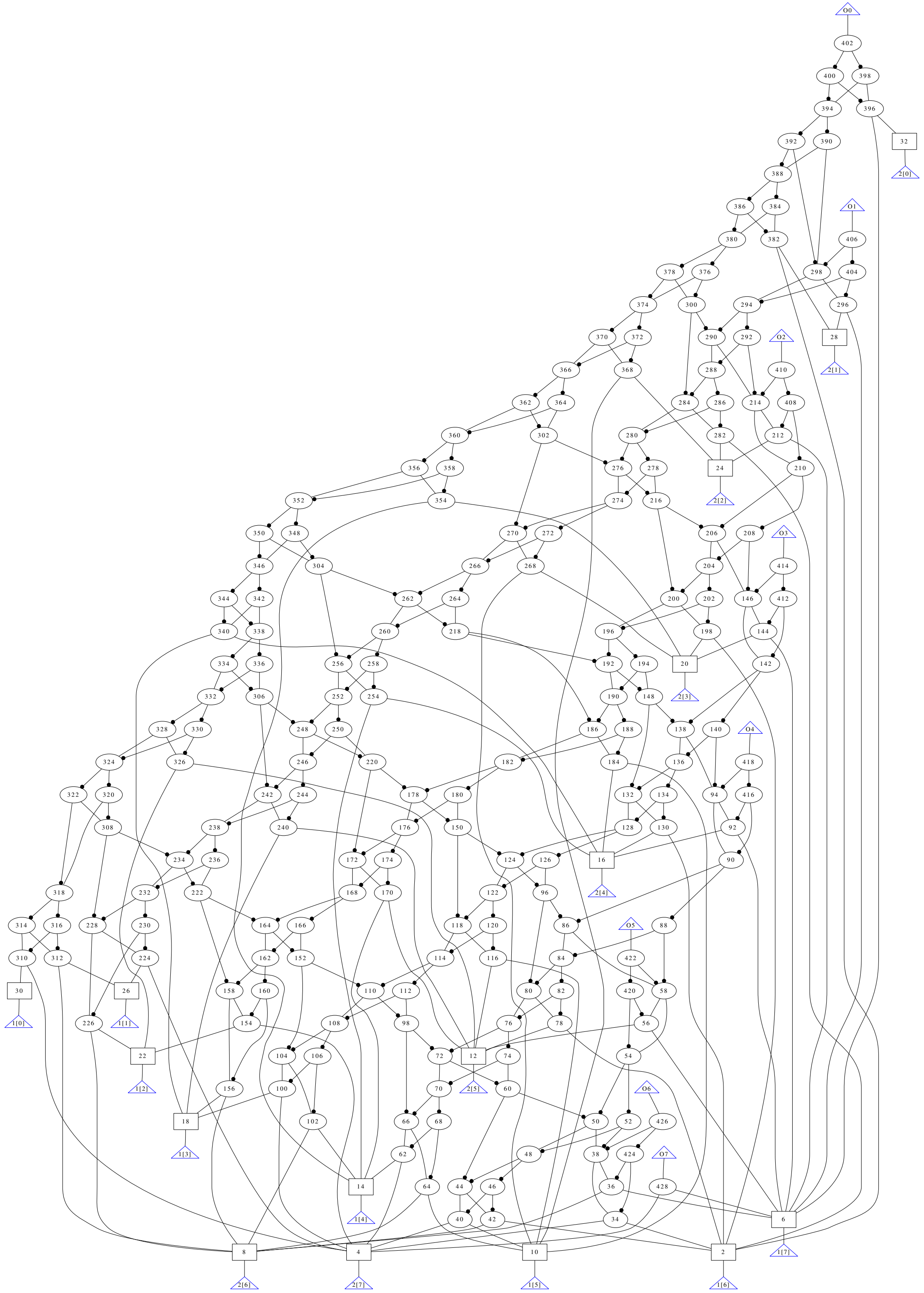


4-bit adder

8-bit adder



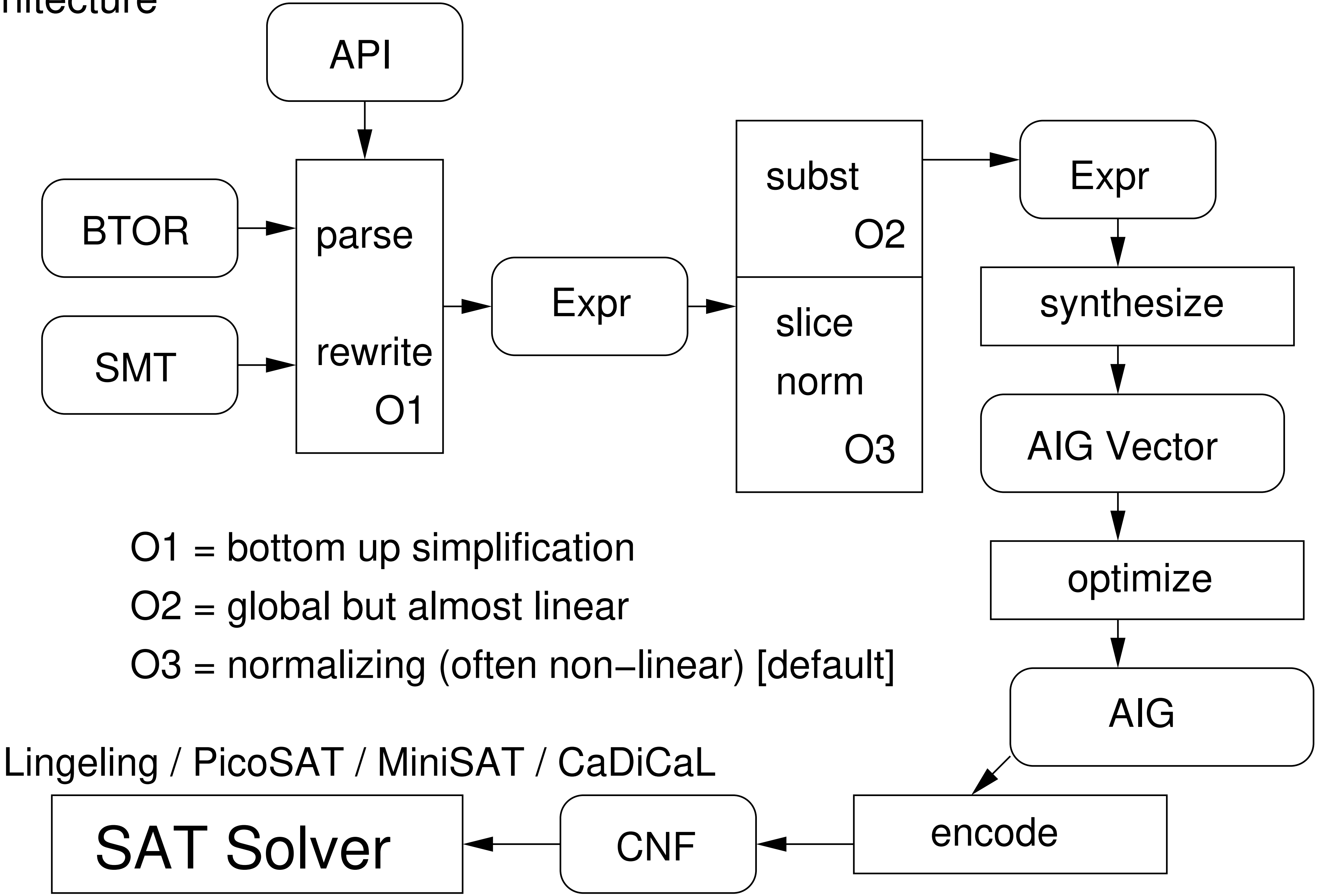
bit-vector of length 16 shifted by bit-vector of length 4


O0
O1
O2
O3
O4
O5
O6
O7
32
28
24
20
16
12
30
26
22
18
14
8
4
10
2
6
2[0]
2[1]
2[2]
2[3]
2[4]
2[5]
2[6]
2[7]
1[0]
1[1]
1[2]
1[3]
1[4]
1[5]
1[6]
1[7]

Complexity of Bit-Blasting

- can handle arbitrary strange computations
  - in essence anything your computer can compute
  - like non-linear constraints, division, modulo, bit-wise operators, shift, . . .
  - floating points (well . . . see Martin Brain's talk tomorrow)
  - can make full use of power of SAT solvers (including preprocessing)
- sometimes generates big and hard SAT formulas
  - deciding bit-vector logic NEXPTIME complete [KovásznaiFröhlichBiere'12]
  - since bit-width is encoded logarithmically
  - even one 32-bit multiplication needs 2000+ AIG nodes
- software engineering for bit-blasting is tricky
  - SAT solvers not good at structural hashing [HeuelJärvisaloBiere'13] [HeuleBiere'13]
  - so need to maintain an intermediate format like AIGs
  - not easy in an incremental setting (inprocessing?)

Boolector Architecture



O1 = bottom up simplification
O2 = global but almost linear
O3 = normalizing (often non–linear) [default]

Encoding Logical Constraints

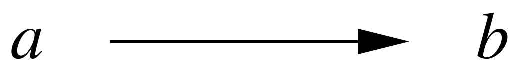
- Tseitin construction suitable for most kinds of “model constraints”
  - assuming simple operational semantics: encode an interpreter
  - small domains: one-hot encoding large domains: binary encoding

    check out “order encoding” too

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

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

Encoding Reachability in Prolog for Graph with 2 Nodes



```
edge(a,b).

reach(X,Y) :- edge(X,Y).
reach(X,Y) :- edge(X,Z), reach(Z,Y).

?- reach(b,a).
```

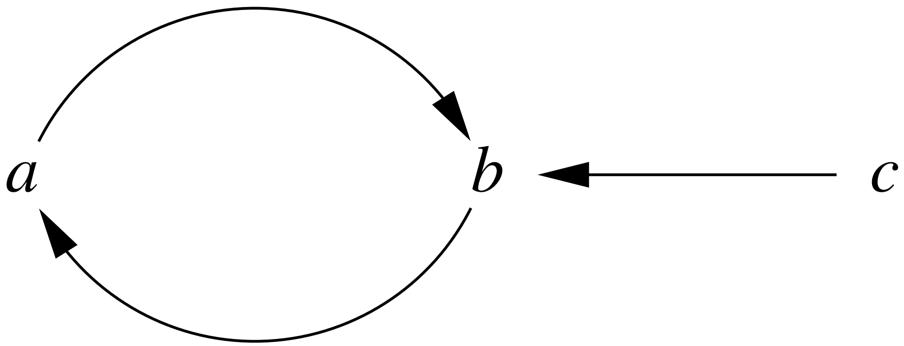
Wrong SAT Encoding for Graph with 2 Nodes

```
edge_a_b &                  (reach_a_a <- edge_a_a & reach_a_a) &
                            (reach_a_a <- edge_a_b & reach_b_a) &
(reach_a_a <- edge_a_a) &   (reach_a_b <- edge_a_a & reach_a_b) &
(reach_a_b <- edge_a_b) &   (reach_a_b <- edge_a_b & reach_b_b) &
(reach_b_a <- edge_b_a) &   (reach_b_a <- edge_b_a & reach_a_a) &
(reach_b_b <- edge_b_b) &   (reach_b_a <- edge_b_b & reach_b_a) &
                            (reach_b_b <- edge_b_a & reach_a_b) &
                            (reach_b_b <- edge_b_b & reach_b_b) &


reach_b_a
```

Right SAT Encoding for Graph with 2 Nodes

```
!edge_a_a &     (reach_a_a_1 <-> (edge_a_a | (edge_a_a & reach_a_a_0))) &
edge_a_b &      (reach_a_a_1 <-> (edge_a_a | (edge_a_b & reach_b_a_0))) &
!edge_b_a &     (reach_a_b_1 <-> (edge_a_b | (edge_a_a & reach_a_b_0))) &
!edge_b_b &     (reach_a_b_1 <-> (edge_a_b | (edge_a_b & reach_b_b_0))) &
                (reach_b_a_1 <-> (edge_b_a | (edge_b_a & reach_a_a_0))) &
!reach_a_a_0 &  (reach_b_a_1 <-> (edge_b_a | (edge_b_b & reach_b_a_0))) &
!reach_a_b_0 &  (reach_b_b_1 <-> (edge_b_b | (edge_b_a & reach_a_b_0))) &
!reach_b_a_0 &  (reach_b_b_1 <-> (edge_b_b | (edge_b_b & reach_b_b_0))) &
!reach_b_b_0 &  (reach_a_a_2 <-> (edge_a_a | (edge_a_a & reach_a_a_1))) &
                (reach_a_a_2 <-> (edge_a_a | (edge_a_b & reach_b_a_1))) &
                (reach_a_b_2 <-> (edge_a_b | (edge_a_a & reach_a_b_1))) &
                (reach_a_b_2 <-> (edge_a_b | (edge_a_b & reach_b_b_1))) &
                (reach_b_a_2 <-> (edge_b_a | (edge_b_a & reach_a_a_1))) &
                (reach_b_a_2 <-> (edge_b_a | (edge_b_b & reach_b_a_1))) &
                (reach_b_b_2 <-> (edge_b_b | (edge_b_a & reach_a_b_1))) &
                (reach_b_b_2 <-> (edge_b_b | (edge_b_b & reach_b_b_1))) &

(reach_b_a_0 | reach_b_a_1 | reach_b_a_2)
```

Encoding Reachability in Prolog for Graph with 3 Nodes



```
edge(a,b).
edge(b,c).
edge(c,b).

reach(X,Y) :- edge(X,Y).
reach(X,Y) :- edge(X,Z), reach (Z,Y).

?- reach(a,c).
```

Wrong SAT Encoding for Graph with 3 Nodes

```
edge_a_b &                (reach_a_a <- edge_a_a & reach_a_a) &
edge_b_a &                (reach_a_a <- edge_a_b & reach_b_a) &
edge_c_b &                (reach_a_a <- edge_a_c & reach_c_a) &
                          (reach_a_b <- edge_a_a & reach_a_b) &
(reach_a_a <- edge_a_a) & (reach_a_b <- edge_a_b & reach_b_b) & (reach_c_a <- edge_c_a & reach_a_a) &
(reach_a_b <- edge_a_b) & (reach_a_b <- edge_a_c & reach_c_b) & (reach_c_a <- edge_c_b & reach_b_a) &
(reach_a_c <- edge_a_c) & (reach_a_c <- edge_a_a & reach_a_c) & (reach_c_a <- edge_c_c & reach_c_a) &
(reach_b_a <- edge_b_a) & (reach_a_c <- edge_a_b & reach_b_c) & (reach_c_b <- edge_c_a & reach_a_b) &
(reach_b_b <- edge_b_b) & (reach_a_c <- edge_a_c & reach_c_c) & (reach_c_b <- edge_c_b & reach_b_b) &
(reach_b_c <- edge_b_c) & (reach_b_a <- edge_b_a & reach_a_a) & (reach_c_b <- edge_c_c & reach_c_b) &
(reach_c_a <- edge_c_a) & (reach_b_a <- edge_b_b & reach_b_a) & (reach_c_c <- edge_c_a & reach_a_c) &
(reach_c_b <- edge_c_b) & (reach_b_a <- edge_b_c & reach_c_a) & (reach_c_c <- edge_c_b & reach_b_c) &
(reach_c_c <- edge_c_c) & (reach_b_b <- edge_b_a & reach_a_b) & (reach_c_c <- edge_c_c & reach_c_c) &
                          (reach_b_b <- edge_b_b & reach_b_b) &
                          (reach_b_b <- edge_b_c & reach_c_b) &
                          (reach_b_c <- edge_b_a & reach_a_c) &
                          (reach_b_c <- edge_b_b & reach_b_c) &
                          (reach_b_c <- edge_b_c & reach_c_c) &
reach_a_c
```

Right SAT Encoding for Graph with 3 Nodes

```
!edge_a_a  &    !reach_a_a_0 &
 edge_a_b  &    !reach_a_b_0 &
!edge_a_c  &    !reach_a_c_0 &
 edge_b_a  &    !reach_b_a_0 &
!edge_b_b  &    !reach_b_b_0 &
!edge_b_c  &    !reach_b_c_0 &
 edge_c_a  &    !reach_c_a_0 &
!edge_c_b  &    !reach_c_b_0 &
!edge_c_c  &    !reach_c_c_0 &

(reach_a_a_1 <-> (edge_a_a | (edge_a_a & reach_a_a_0)))  &  (reach_a_a_2 <-> (edge_a_a | (edge_a_a & reach_a_a_1))) &
(reach_a_a_1 <-> (edge_a_a | (edge_a_b & reach_b_a_0)))  &  (reach_a_a_2 <-> (edge_a_a | (edge_a_b & reach_b_a_1))) &
(reach_a_a_1 <-> (edge_a_a | (edge_a_c & reach_c_a_0)))  &  (reach_a_a_2 <-> (edge_a_a | (edge_a_c & reach_c_a_1))) &
(reach_a_b_1 <-> (edge_a_b | (edge_a_a & reach_a_b_0)))  &  (reach_a_b_2 <-> (edge_a_b | (edge_a_a & reach_a_b_1))) &
(reach_a_b_1 <-> (edge_a_b | (edge_a_b & reach_b_b_0)))  &  (reach_a_b_2 <-> (edge_a_b | (edge_a_b & reach_b_b_1))) &
(reach_a_b_1 <-> (edge_a_b | (edge_a_c & reach_c_b_0)))  &  (reach_a_b_2 <-> (edge_a_b | (edge_a_c & reach_c_b_1))) &
(reach_a_c_1 <-> (edge_a_c | (edge_a_a & reach_a_c_0)))  &  (reach_a_c_2 <-> (edge_a_c | (edge_a_a & reach_a_c_1))) &
(reach_a_c_1 <-> (edge_a_c | (edge_a_b & reach_b_c_0)))  &  (reach_a_c_2 <-> (edge_a_c | (edge_a_b & reach_b_c_1))) &
(reach_a_c_1 <-> (edge_a_c | (edge_a_c & reach_c_c_0)))  &  (reach_a_c_2 <-> (edge_a_c | (edge_a_c & reach_c_c_1))) &
(reach_b_a_1 <-> (edge_b_a | (edge_b_a & reach_a_a_0)))  &  (reach_b_a_2 <-> (edge_b_a | (edge_b_a & reach_a_a_1))) &
(reach_b_a_1 <-> (edge_b_a | (edge_b_b & reach_b_a_0)))  &  (reach_b_a_2 <-> (edge_b_a | (edge_b_b & reach_b_a_1))) &
(reach_b_a_1 <-> (edge_b_a | (edge_b_c & reach_c_a_0)))  &  (reach_b_a_2 <-> (edge_b_a | (edge_b_c & reach_c_a_1))) &
(reach_b_b_1 <-> (edge_b_b | (edge_b_a & reach_a_b_0)))  &  (reach_b_b_2 <-> (edge_b_b | (edge_b_a & reach_a_b_1))) &
(reach_b_b_1 <-> (edge_b_b | (edge_b_b & reach_b_b_0)))  &  (reach_b_b_2 <-> (edge_b_b | (edge_b_b & reach_b_b_1))) &
(reach_b_b_1 <-> (edge_b_b | (edge_b_c & reach_c_b_0)))  &  (reach_b_b_2 <-> (edge_b_b | (edge_b_c & reach_c_b_1))) &
(reach_b_c_1 <-> (edge_b_c | (edge_b_a & reach_a_c_0)))  &  (reach_b_c_2 <-> (edge_b_c | (edge_b_a & reach_a_c_1))) &
(reach_b_c_1 <-> (edge_b_c | (edge_b_b & reach_b_c_0)))  &  (reach_b_c_2 <-> (edge_b_c | (edge_b_b & reach_b_c_1))) &
(reach_b_c_1 <-> (edge_b_c | (edge_b_c & reach_c_c_0)))  &  (reach_b_c_2 <-> (edge_b_c | (edge_b_c & reach_c_c_1))) &
(reach_c_a_1 <-> (edge_c_a | (edge_c_a & reach_a_a_0)))  &  (reach_c_a_2 <-> (edge_c_a | (edge_c_a & reach_a_a_1))) &
(reach_c_a_1 <-> (edge_c_a | (edge_c_b & reach_b_a_0)))  &  (reach_c_a_2 <-> (edge_c_a | (edge_c_b & reach_b_a_1))) &
(reach_c_a_1 <-> (edge_c_a | (edge_c_c & reach_c_a_0)))  &  (reach_c_a_2 <-> (edge_c_a | (edge_c_c & reach_c_a_1))) &
(reach_c_b_1 <-> (edge_c_b | (edge_c_a & reach_a_b_0)))  &  (reach_c_b_2 <-> (edge_c_b | (edge_c_a & reach_a_b_1))) &
(reach_c_b_1 <-> (edge_c_b | (edge_c_b & reach_b_b_0)))  &  (reach_c_b_2 <-> (edge_c_b | (edge_c_b & reach_b_b_1))) &
(reach_c_b_1 <-> (edge_c_b | (edge_c_c & reach_c_b_0)))  &  (reach_c_b_2 <-> (edge_c_b | (edge_c_c & reach_c_b_1))) &
(reach_c_c_1 <-> (edge_c_c | (edge_c_a & reach_a_c_0)))  &  (reach_c_c_2 <-> (edge_c_c | (edge_c_a & reach_a_c_1))) &
(reach_c_c_1 <-> (edge_c_c | (edge_c_b & reach_b_c_0)))  &  (reach_c_c_2 <-> (edge_c_c | (edge_c_b & reach_b_c_1))) &
(reach_c_c_1 <-> (edge_c_c | (edge_c_c & reach_c_c_0)))  &  (reach_c_c_2 <-> (edge_c_c | (edge_c_c & reach_c_c_1))) &

(reach_b_a_0 | reach_b_a_1 | reach_b_a_2)
```

Encoding Least Fix-Points

- incremental encoding for least fix-points [GebserKaufmannNeumannSchaub'07]
  - use the "wrong encoding" and call SAT solver
  - if unsatisfiable no least fix-point exists
  - if satisfiable check solution for cyclic dependencies
  - if there is no cyclic dependency then the model is a least fix-point
  - otherwise add clause which removes cycle and continue
- other incremental encodings
  - simple path constraints in BMC / k-induction [EénSörensson'03]
  - lazy clause encoding in CP [OhrimenkoStuckeyCodish'07]
  - lemmas on demand for SMT [deMouraRueß'02] [BrummayerBiere'09]
- lazy encodings might result in adding exponential many clauses
- encoding temporal properties (LTL) for BMC [LatvalaBiereHeljankoJuntilla'04]
  - temporal operators with least fix-point semantics: $\mathsf{F}p$, $p\,\mathsf{U}\,q$
  - needs only two "iterations" due to monotonicity of the semantics

Example of Logical Constraints: Cardinality Constraints

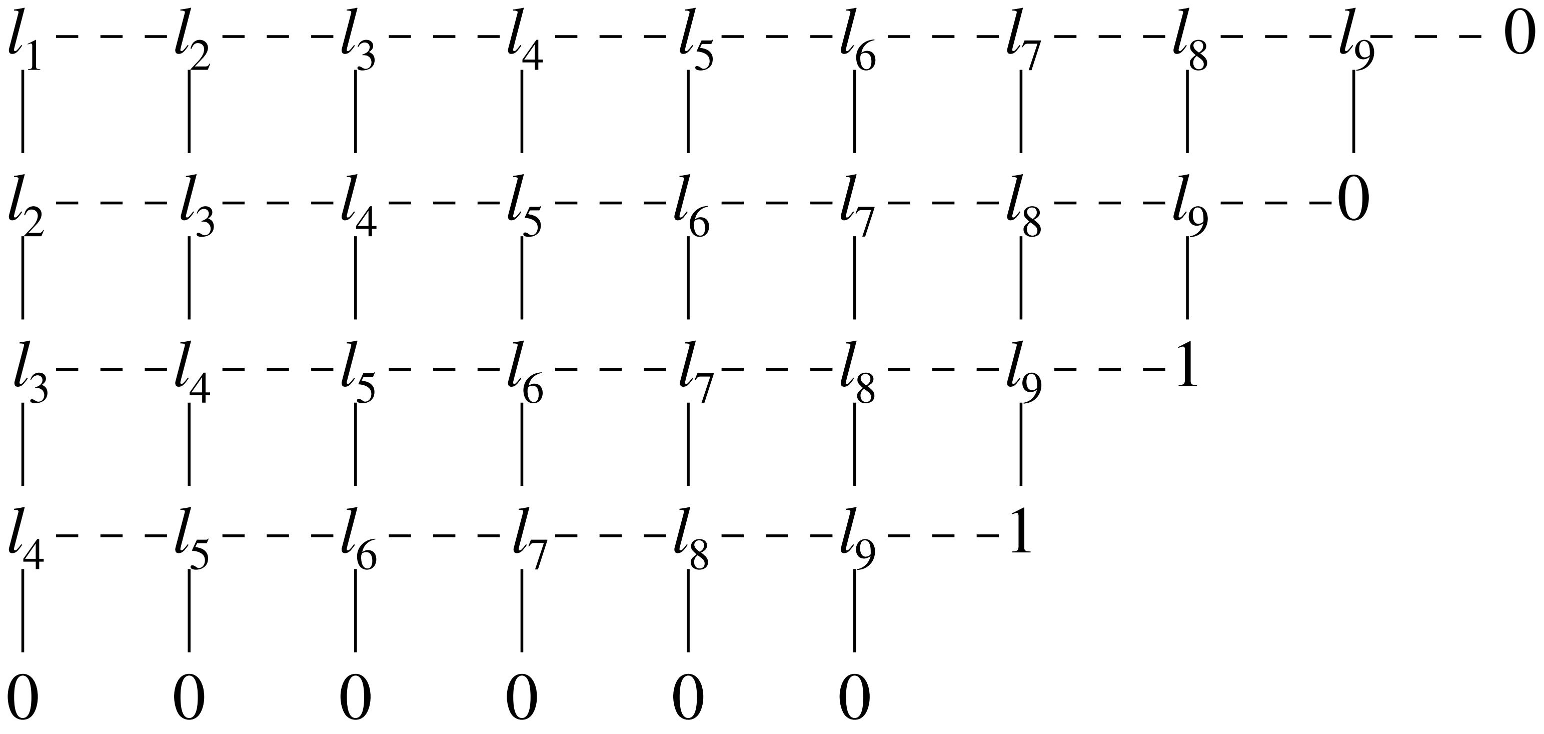
- given a set of literals $\{l_1, \ldots l_n\}$
    - constraint the number of literals assigned to *true*
    - $l_1 + \cdots + l_n \geq k$ or $l_1 + \cdots + l_n \leq k$ or $l_1 + \cdots + l_n = k$
    - combined make up exactly all fully symmetric boolean functions
- multiple encodings of cardinality constraints
    - naïve encoding exponential: at-most-one quadratic, at-most-two cubic, etc.
    - quadratic $O(k \cdot n)$ encoding has its roots in [Shannon'38]
    - linear $O(n)$ parallel counter encoding [Sinz'05]
- many variants even for at-most-one constraints
    - see [BiereLeBerreLoncaManthey'14] [MantheyHeuleBiere'13] for references
- 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

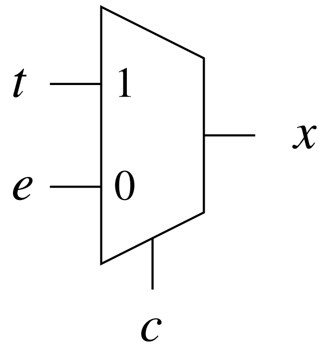
BDD-Based Encoding of Cardinality Constraints

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



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

Tseitin Encoding of If-Then-Else Gate and Arc Consistency



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

this is a minimal size CNF but the CNF is not **arc consistent**

- if t and e have the same value then x needs to have that too

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

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

DIMACS Format

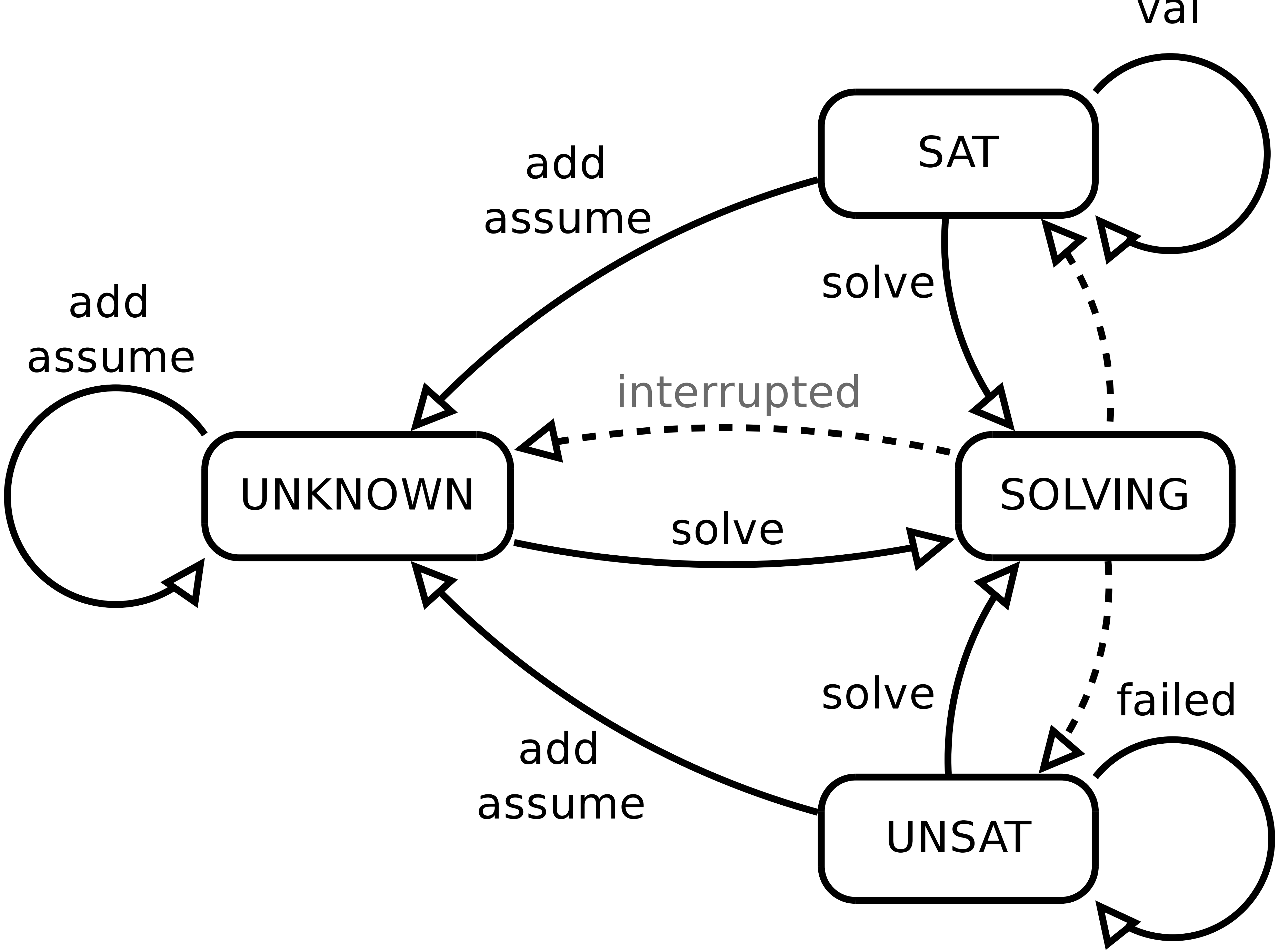
```
$ cat example.cnf
c comments start with ’c’ and extend until the end of the line
c
c variables are encoded as integers:
c
c   ’tie’   becomes ’1’
c   ’shirt’ becomes ’2’
c
c header ’p cnf <variables> <clauses>’
c
p cnf 2 3
-1  2 0           c  !tie  or  shirt
 1  2 0           c   tie  or  shirt
-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 UNSAT, UNSAT, . . . , UNSAT, SAT vs. SAT, SAT, . . . ,UNSAT
- retracting facts
  - remove clauses explicitly: complex to implement
  - push / pop: stack like activation, no sharing of learned facts (as in SMTLIB)
  - 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



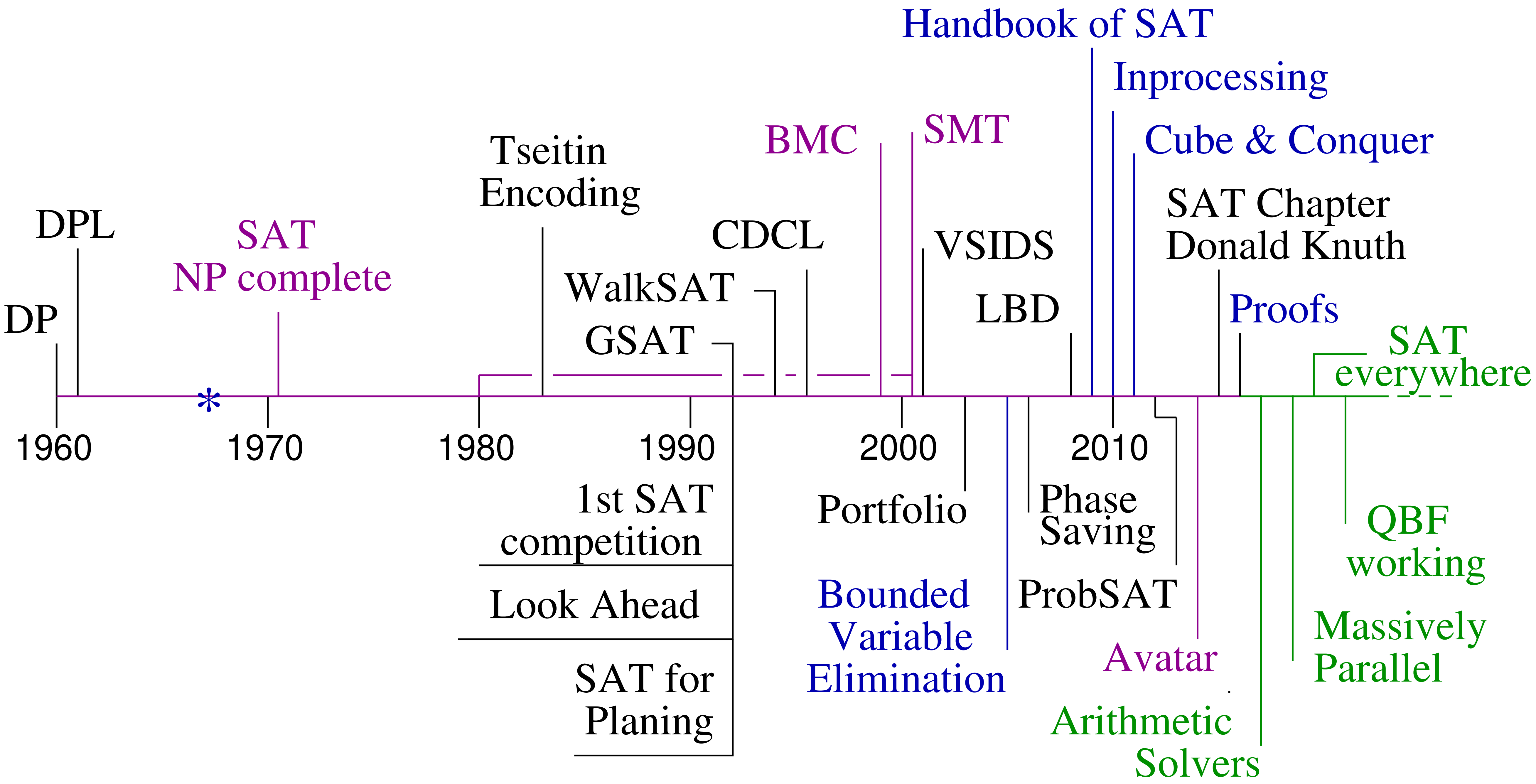
```c
#include "ipasir.h"
#include <assert.h>
#include <stdio.h>
#define ADD(LIT) ipasir_add (solver, LIT)
#define PRINT(LIT) \
  printf (ipasir_val (solver, LIT) < 0 ?  " -" #LIT : " " #LIT)
int main () {
  void * solver = ipasir_init ();
  enum { tie = 1, shirt = 2 };
  ADD (-tie); ADD ( shirt); ADD (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));
```

Personal SAT Solver History



DP

DPL

*

SAT
NP complete

Tseitin
Encoding

GSAT

WalkSAT

CDCL

BMC

SMT

VSIDS

LBD

Handbook of SAT

Inprocessing

Cube & Conquer

SAT Chapter
Donald Knuth

Proofs

SAT
everywhere

1960

1970

1980

1990

2000

2010

1st SAT
competition

Look Ahead

SAT for
Planing

Portfolio

Bounded
Variable
Elimination

Phase
Saving

ProbSAT

Avatar

Arithmetic
Solvers

QBF
working

Massively
Parallel

Links

- https://fmv.jku.at/limboole
- https://fmv.jku.at/aiger
- https://github.com/Boolector/boolector
- https://github.com/biotomas/ipasir
- https://github.com/arminbiere/cadical
- https://github.com/arminbiere/lingeling

Jobs

- new LIT AI Lab
  - Linz Institute of Technology (LIT)
  - Artificial Intelligence (AI)
- new LIT AI PhD School
- world-class experts
  - machine learning Hochreiter, Widmer
  - SAT / SMT / AR Biere, Seidl, Kauers
- deductive & inductive reasoning

We are Hiring!

PostDocs + PhDs