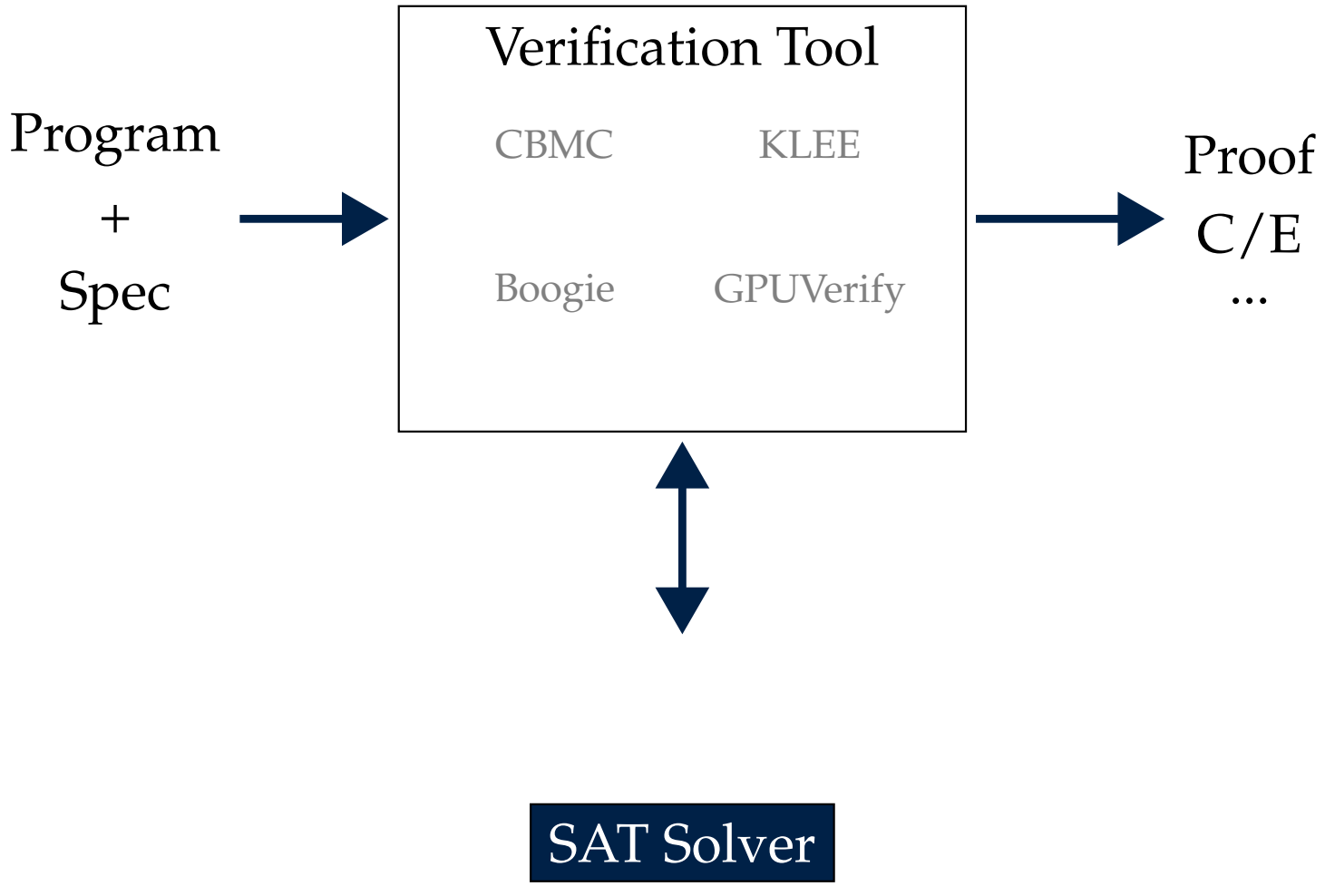
Optimally Propagating SAT Encodings

Martin Brain, **Liana Hadarean**,
Ruben Martins and Daniel Kroening

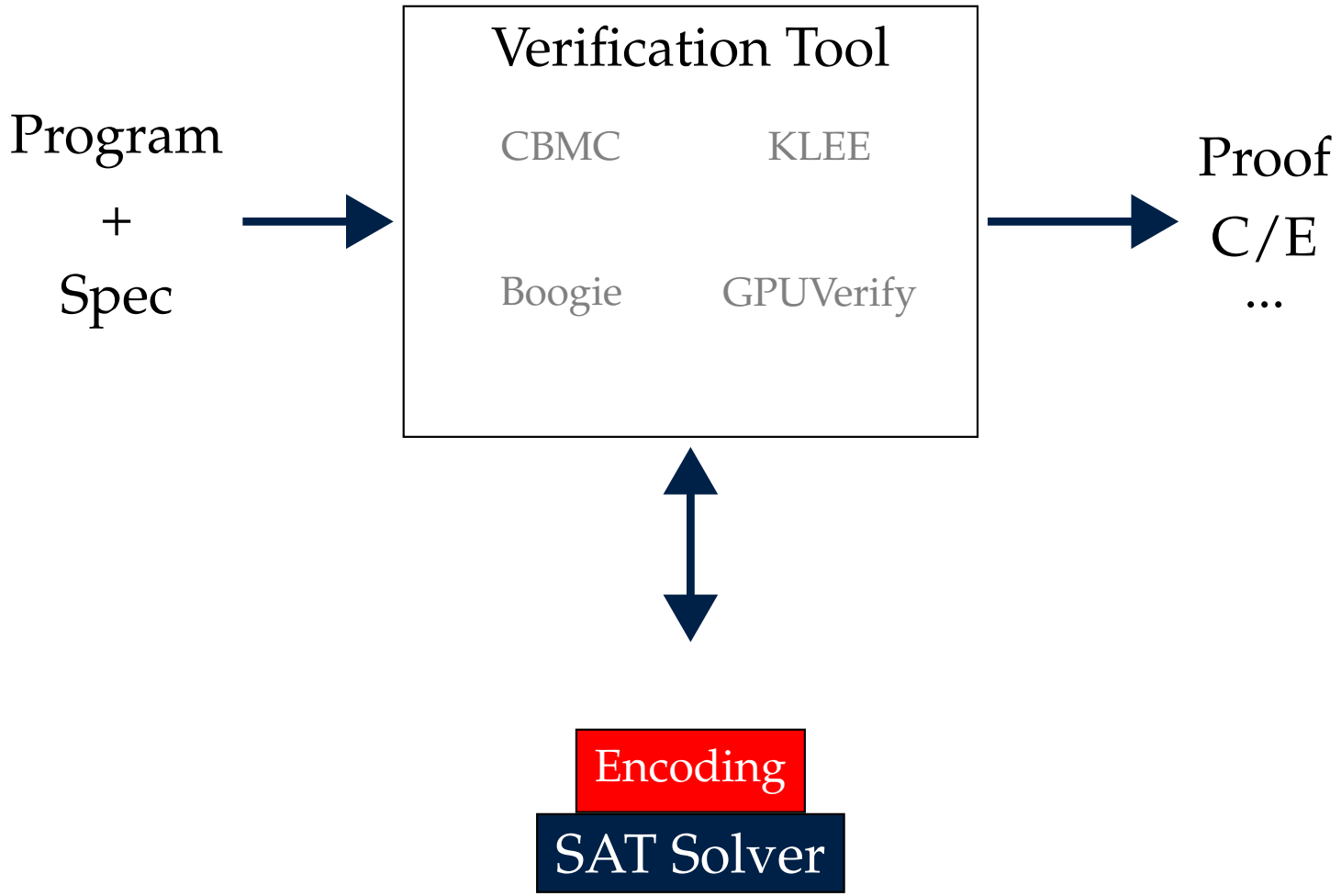
Oxford University

July 18, 2015

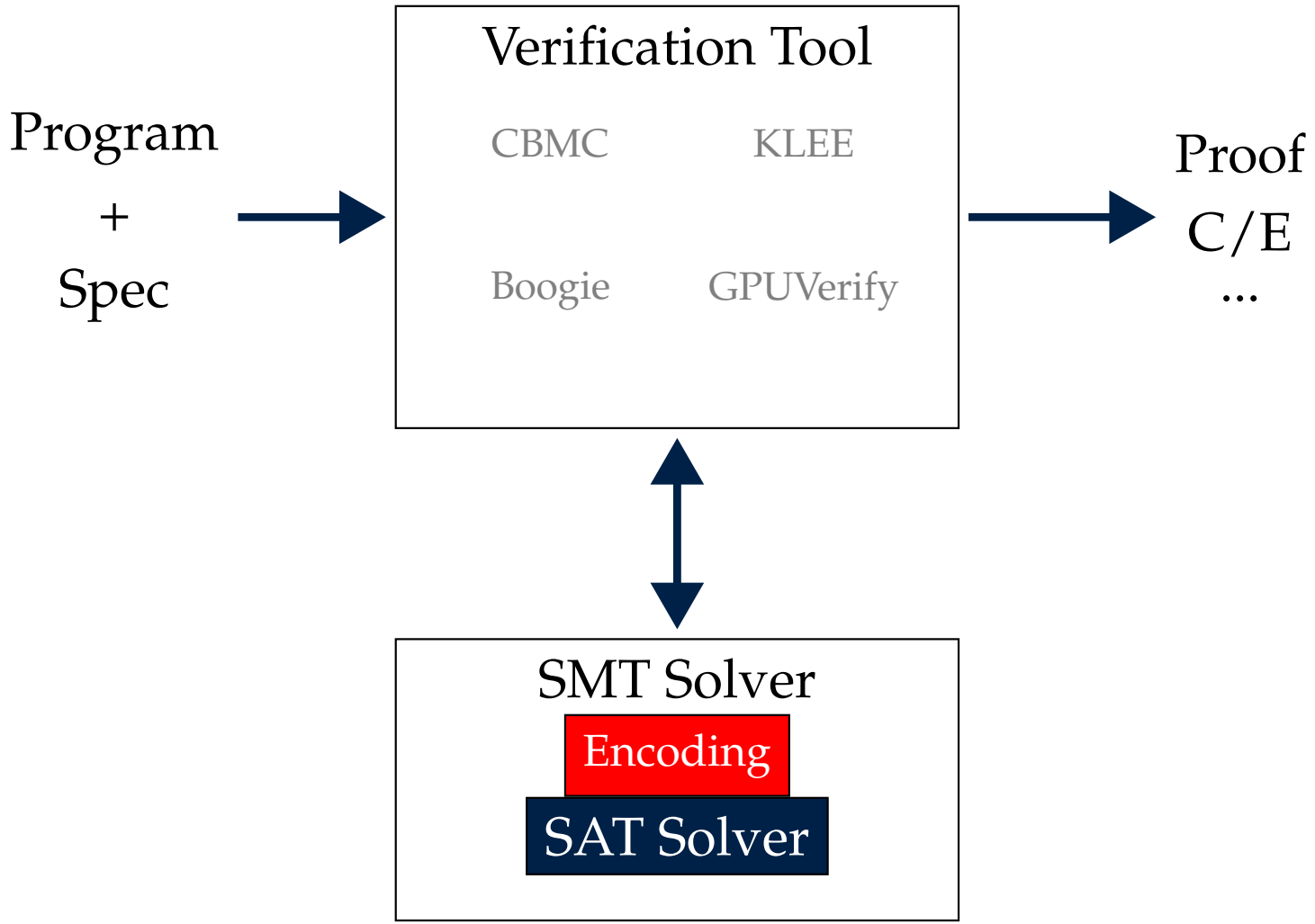
A COMMON PICTURE



A COMMON PICTURE



A COMMON PICTURE



SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

- reverse engineer bad encodings within the SAT solver! [**biere2014**]

SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

- reverse engineer bad encodings within the SAT solver! [**biere2014**]

How to design good encodings?

SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

- reverse engineer bad encodings within the SAT solver! [**biere2014**]

How to design good encodings?

- a bit of a “dark” art

SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

- reverse engineer bad encodings within the SAT solver! [**biere2014**]

How to design good encodings?

- a bit of a “dark” art
- smaller is better

SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

- reverse engineer bad encodings within the SAT solver! [**biere2014**]

How to design good encodings?

- a bit of a “dark” art
- smaller is better
- but not always

SAT Encodings

Modular - optimize encoding offline and reuse template.

Essential for SAT solver performance

- reverse engineer bad encodings within the SAT solver! [**biere2014**]

How to design good encodings?

- a bit of a "dark" art
- smaller is better
- but not always

SOTA: try, test and pick the best!

How to design SAT solver friendly encodings?

[1] Known as *propagation completeness* in AI knowledge compilation.

How to design SAT solver friendly encodings?

- property of interest: *propagation power*

[1] Known as *propagation completeness* in AI knowledge compilation.

How to design SAT solver friendly encodings?

- property of interest: *propagation power*

Can all logically entailed literals be inferred by unit propagation?

[1] Known as *propagation completeness* in AI knowledge compilation.

How to design SAT solver friendly encodings?

- property of interest: *propagation power*

Can all logically entailed literals be inferred by unit propagation?

- automatically generate *optimally propagating* encodings[1]

[1] Known as *propagation completeness* in AI knowledge compilation.

How to design SAT solver friendly encodings?

- property of interest: *propagation power*

Can all logically entailed literals be inferred by unit propagation?

- automatically generate *optimally propagating* encodings[1]
- abstract satisfaction framework for encodings

[1] Known as *propagation completeness* in AI knowledge compilation.

Preliminaries

Σ variables	v
literal	$l \in \{v, \neg v\}$
C_Σ clauses	$c = l_1 \vee \ldots \vee l_n$

Preliminaries

Σ variables	v
literal	$l \in \{v, \neg v\}$
C_Σ clauses	$c = l_1 \vee \ldots \vee l_n$
A_Σ assignments	$\mathsf{a} : \Sigma \to \{\bot, \top\}$
P_Σ partial assignments	$\mathsf{p} : \Sigma \to \{\bot, ?, \top\}$

Preliminaries

Σ variables	v
literal	$l \in \{v, \neg v\}$
C_Σ clauses	$c = l_1 \vee \ldots \vee l_n$
A_Σ assignments	$\mathsf{a} : \Sigma \to \{\bot, \top\}$
P_Σ partial assignments	$\mathsf{p} : \Sigma \to \{\bot, ?, \top\}$
unit propagation	$UP : (2^{C_\Sigma} \times P_\Sigma) \to P_\Sigma$

PRELIMINARIES

$$
\begin{array}{rl}
\Sigma \text{ variables} & v \\
\text{literal} & l \in \{v, \neg v\} \\
C_\Sigma \text{ clauses} & c = l_1 \vee \ldots \vee l_n
\end{array}
$$

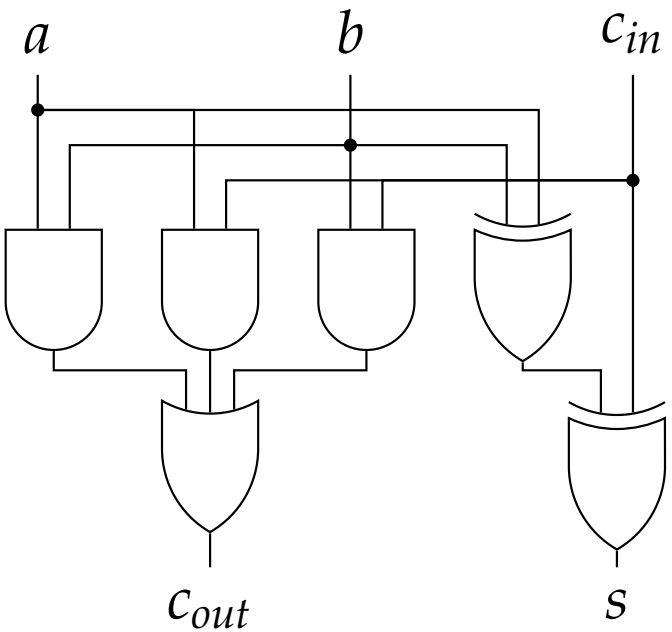
$$
\begin{array}{rl}
A_\Sigma \text{ assignments} & \mathsf{a} : \Sigma \to \{\bot, \top\} \\
P_\Sigma \text{ partial assignments} & \mathsf{p} : \Sigma \to \{\bot, ?, \top\}
\end{array}
$$

$$
\text{unit propagation} \quad UP : (2^{C_\Sigma} \times P_\Sigma) \to P_\Sigma
$$

$$
\left.
\begin{array}{l}
l \vee l_1 \vee \ldots \vee l_k \in C \\
\mathsf{p}(l_i) = \bot \text{ and } \mathsf{p}(l) = ?
\end{array}
\right\} \mathsf{p}[l \leftarrow \top]
$$

Full-adder

Example

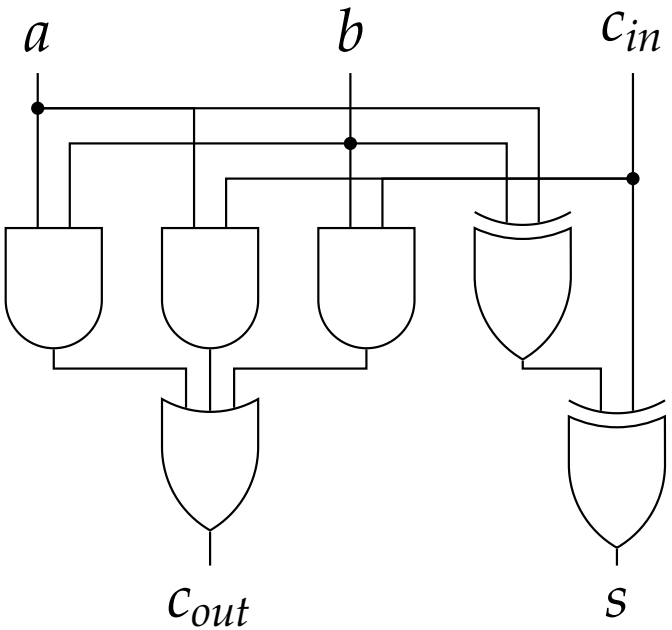


Truth table:

a	b	c_{in}	c_{out}	s
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

Full-adder

Example

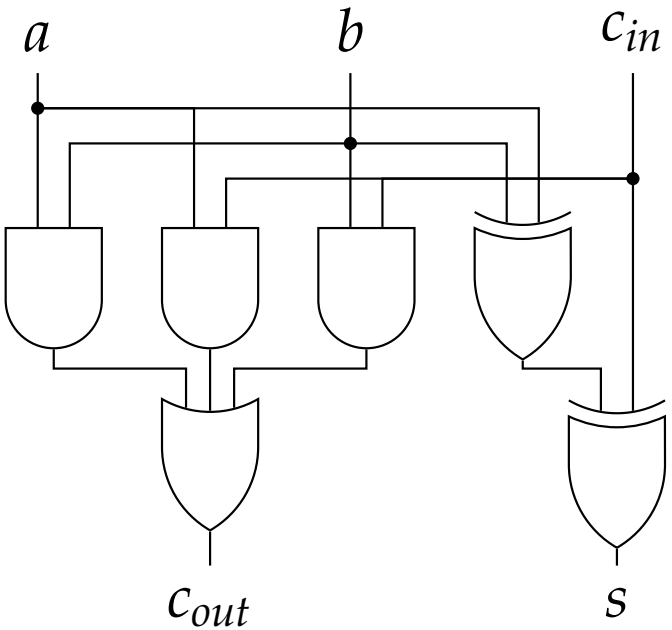


Encoding:

$$\{\neg a, \neg b, c_{in}, \neg s\} \quad \{\neg a, b, \neg c_{in}, \neg s\}$$
$$\{a, \neg b, \neg c_{in}, \neg s\} \quad \{a, b, c_{in}, \neg s\}$$
$$\{\neg a, \neg b, \neg c_{in}, s\} \quad \{\neg a, b, c_{in}, s\}$$
$$\{a, b, \neg c_{in}, s\} \quad \{a, \neg b, c_{in}, s\}$$
$$\{\neg a, \neg b, c_{out}\} \quad \{\neg a, \neg c_{in}, c_{out}\}$$
$$\{\neg b, \neg c_{in}, c_{out}\} \quad \{a, b, \neg c_{out}\}$$
$$\{a, c_{in}, \neg c_{out}\} \quad \{b, c_{in}, \neg c_{out}\}$$

Full-adder

Example



Encoding:

$$\{\neg a, \neg b, c_{in}, \neg s\} \quad \{\neg a, b, \neg c_{in}, \neg s\}$$
$$\{a, \neg b, \neg c_{in}, \neg s\} \quad \{a, b, c_{in}, \neg s\}$$
$$\{\neg a, \neg b, \neg c_{in}, s\} \quad \{\neg a, b, c_{in}, s\}$$
$$\{a, b, \neg c_{in}, s\} \quad \{a, \neg b, c_{in}, s\}$$
$$\{\neg a, \neg b, c_{out}\} \quad \{\neg a, \neg c_{in}, c_{out}\}$$
$$\{\neg b, \neg c_{in}, c_{out}\} \quad \{a, b, \neg c_{out}\}$$
$$\{a, c_{in}, \neg c_{out}\} \quad \{b, c_{in}, \neg c_{out}\}$$

Optimally propagating encodings

We want to encode a model $M \subset A_\Sigma$ (the truth table) using a set of clauses $E_M \subset C_\Sigma$ such that:

$$E_M = \{C \subset C_\Sigma | \mathsf{AofC}(C) = M\}$$

where:

$$\mathsf{AofC}(C) = \{\mathsf{a} \in A_\Sigma | \forall c \in C \,.\, \mathsf{a} \models c\}$$

Optimally propagating encodings

We want to encode a model $M \subset A_\Sigma$ (the truth table) using a set of clauses $E_M \subset C_\Sigma$ such that:

$$E_M = \{C \subset C_\Sigma | \mathsf{AofC}(C) = M\}$$

where:

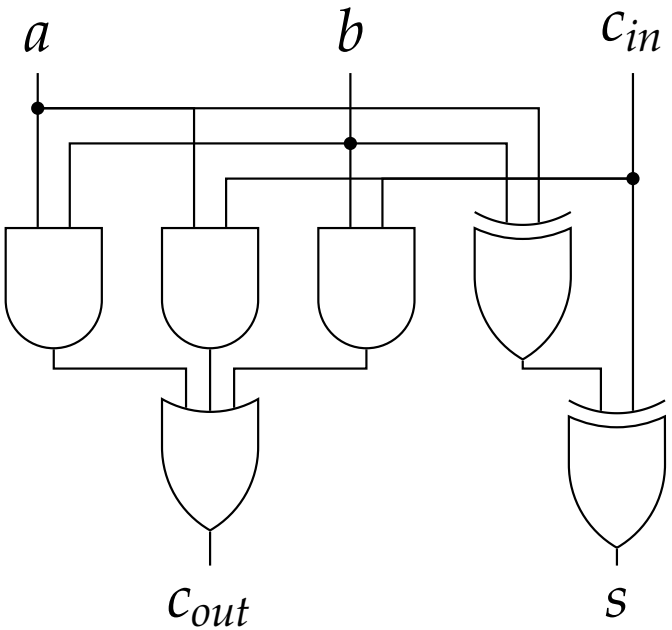
$$\mathsf{AofC}(C) = \{\mathsf{a} \in A_\Sigma | \forall c \in C \,.\, \mathsf{a} \models c\}$$

Optimally propagating encoding

E is optimally propagating if $E \wedge \mathsf{p} \models l$ then $UP(E, \mathsf{p})(l) = \top$.

Full-adder

Example



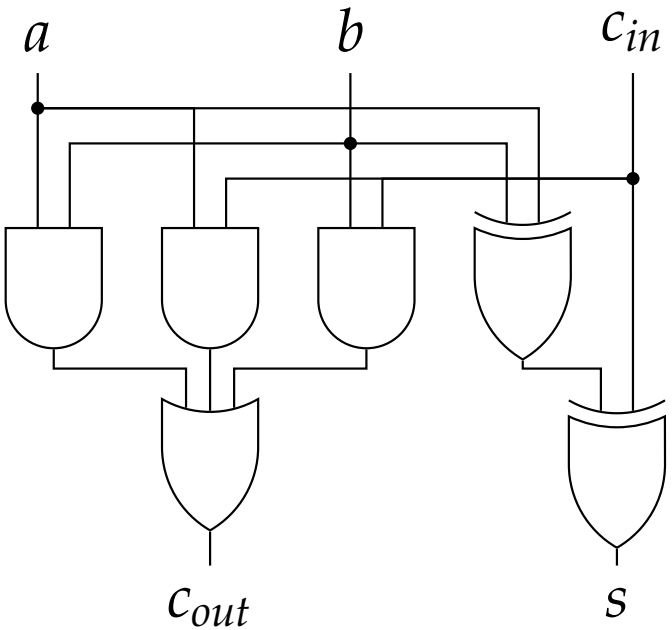
Encoding:

$$\{\neg a, \neg b, c_{in}, \neg s\} \quad \{\neg a, b, \neg c_{in}, \neg s\}$$
$$\{a, \neg b, \neg c_{in}, \neg s\} \quad \{a, b, c_{in}, \neg s\}$$
$$\{\neg a, \neg b, \neg c_{in}, s\} \quad \{\neg a, b, c_{in}, s\}$$
$$\{a, b, \neg c_{in}, s\} \quad \{a, \neg b, c_{in}, s\}$$
$$\{\neg a, \neg b, c_{out}\} \quad \{\neg a, \neg c_{in}, c_{out}\}$$
$$\{\neg b, \neg c_{in}, c_{out}\} \quad \{a, b, \neg c_{out}\}$$
$$\{a, c_{in}, \neg c_{out}\} \quad \{b, c_{in}, \neg c_{out}\}$$

Is this optimally propagating?

Full-adder

Example



Encoding:

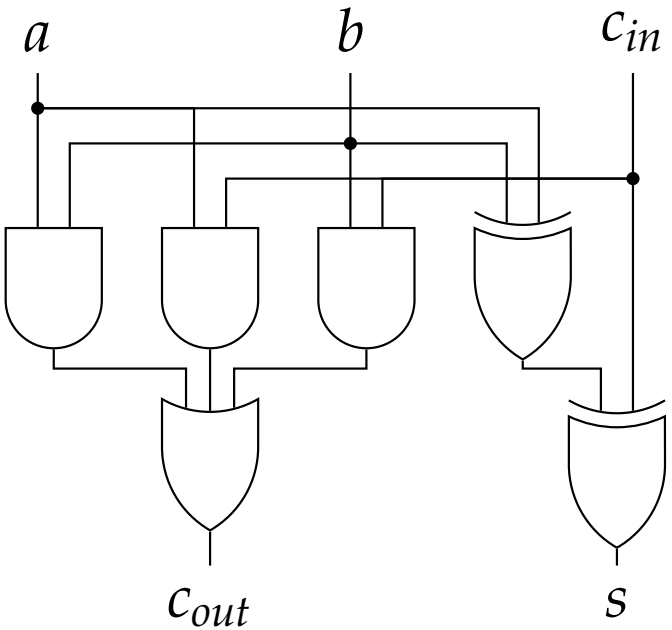
$\{\neg a, \neg b, c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$
$\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, b, c_{in}, \neg s\}$
$\{\neg a, b, c_{in}, s\}$
$\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg c_{in}, c_{out}\}$
$\{a, b, \neg c_{out}\}$
$\{b, c_{in}, \neg c_{out}\}$

$\mathsf{p} = [s, c_{out}]$

$UP(E, \mathsf{p}) = \mathsf{p}$

Full-adder

Example



Encoding:

$\{\neg a, \neg b, c_{in}, \neg s\}$ $\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$ $\{a, b, c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$ $\{\neg a, b, c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$ $\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$ $\{\neg a, \neg c_{in}, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$ $\{a, b, \neg c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$ $\{b, c_{in}, \neg c_{out}\}$

$$\mathsf{p} = [s, c_{out}, \neg a]$$

AUTOMATICALLY STRENGTHEN ENCODINGS

Given:

- initial encoding E_0
- reference encoding E_{ref}

AUTOMATICALLY STRENGTHEN ENCODINGS

Given:

- initial encoding E_0
- reference encoding E_{ref}

Return an encoding E such that:

AUTOMATICALLY STRENGTHEN ENCODINGS

Given:

- initial encoding E_0
- reference encoding E_{ref}

Return an encoding E such that:

- E is optimally propagating

AUTOMATICALLY STRENGTHEN ENCODINGS

Given:

- initial encoding E_0
- reference encoding E_{ref}

Return an encoding E such that:

- E is optimally propagating
- E is set-wise minimal

AUTOMATICALLY STRENGTHEN ENCODINGS

Given:

- initial encoding E_0
- reference encoding E_{ref}

Return an encoding E such that:

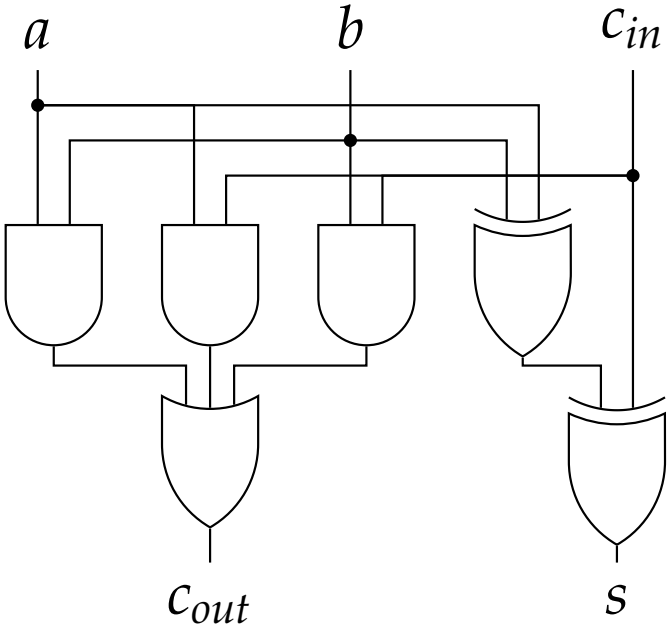
- E is optimally propagating
- E is set-wise minimal
- $\mathsf{AofC}(E) = \mathsf{AofC}(E_{ref})$

Encoding Algorithm

```
1  E ← E_0
2  PQ.push(λv.?)
3  while not PQ.empty() do
4      p ← PQ.pop()
5      foreach v such that UP(E, p)(v) =? do
6          foreach l ∈ {v, ¬v} do
7              p′ ← p [l ← ⊤]
8              if solve(E_ref, p′) = sat then
9                  PQ.push(p′)
10             else
11                 E ← E ∪{MUS(p′)}
12                 PQ.compact()
```

Full-adder

Encoding

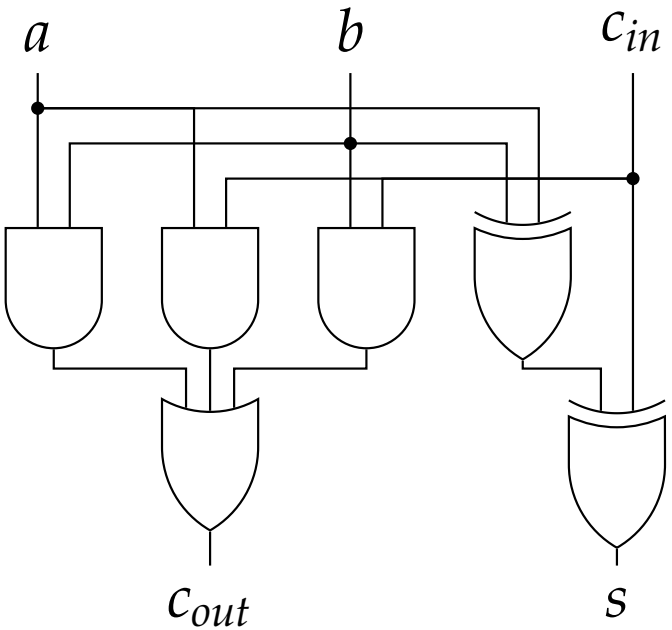


Encoding:

$$\{\neg a, \neg b, c_{in}, \neg s\} \quad \{\neg a, b, \neg c_{in}, \neg s\}$$
$$\{a, \neg b, \neg c_{in}, \neg s\} \quad \{a, b, c_{in}, \neg s\}$$
$$\{\neg a, \neg b, \neg c_{in}, s\} \quad \{\neg a, b, c_{in}, s\}$$
$$\{a, b, \neg c_{in}, s\} \quad \{a, \neg b, c_{in}, s\}$$
$$\{\neg a, \neg b, c_{out}\} \quad \{\neg a, \neg c_{in}, c_{out}\}$$
$$\{\neg b, \neg c_{in}, c_{out}\} \quad \{a, b, \neg c_{out}\}$$
$$\{a, c_{in}, \neg c_{out}\} \quad \{b, c_{in}, \neg c_{out}\}$$

Full-adder

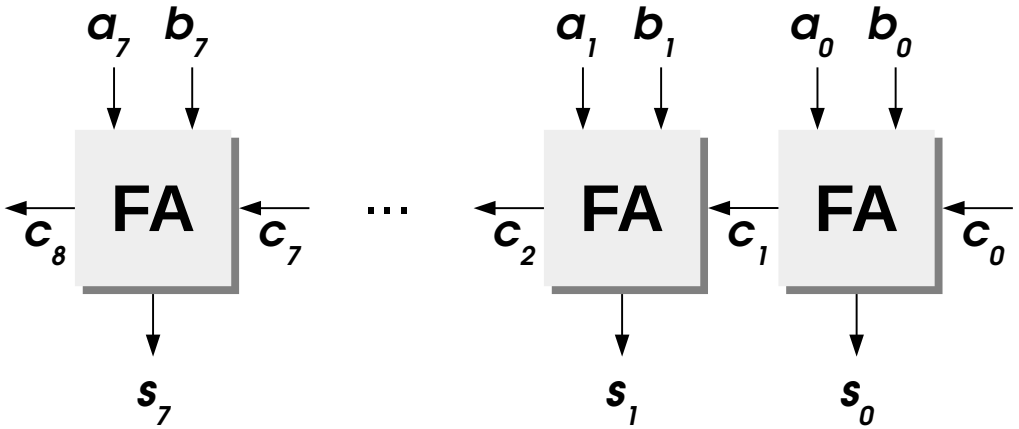
Encoding



Optimal Encoding:

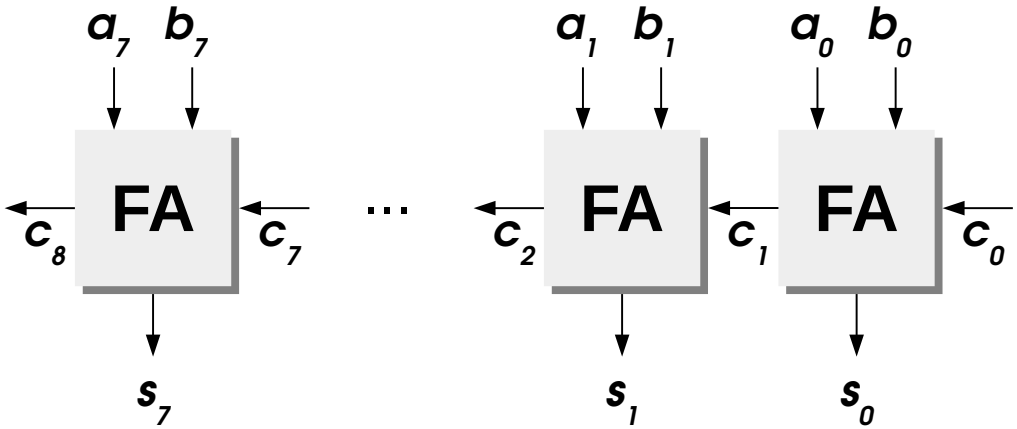
$$\{c_{in}, \neg s, \neg c_{out}\} \quad \{b, \neg s, \neg c_{out}\}$$
$$\{\neg a, \neg b, \neg c_{in}, s\} \quad \{\neg a, s, c_{out}\}$$
$$\{\neg a, \neg b, c_{out}\} \quad \{\neg b, \neg c_{in}, c_{out}\}$$
$$\{a, b, \neg c_{out}\} \quad \{b, c_{in}, \neg c_{out}\}$$
$$\{a, \neg s, \neg c_{out}\} \quad \{a, b, c_{in}, \neg s\}$$
$$\{\neg c_{in}, s, c_{out}\} \quad \{\neg b, s, c_{out}\}$$
$$\{\neg a, \neg c_{in}, c_{out}\} \quad \{a, c_{in}, \neg c_{out}\}$$

N-BIT ADDER



4-bit adder:

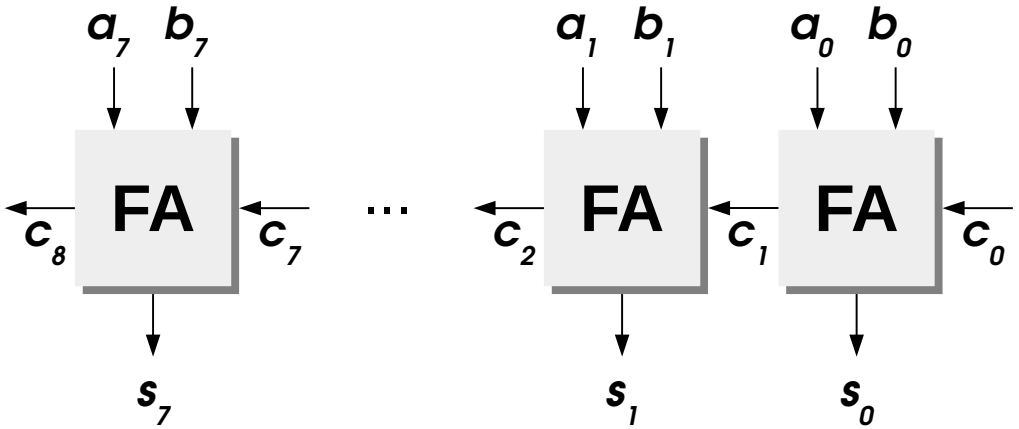
N-BIT ADDER



4-bit adder:

- reference encoding: 26 variables, 58 clauses

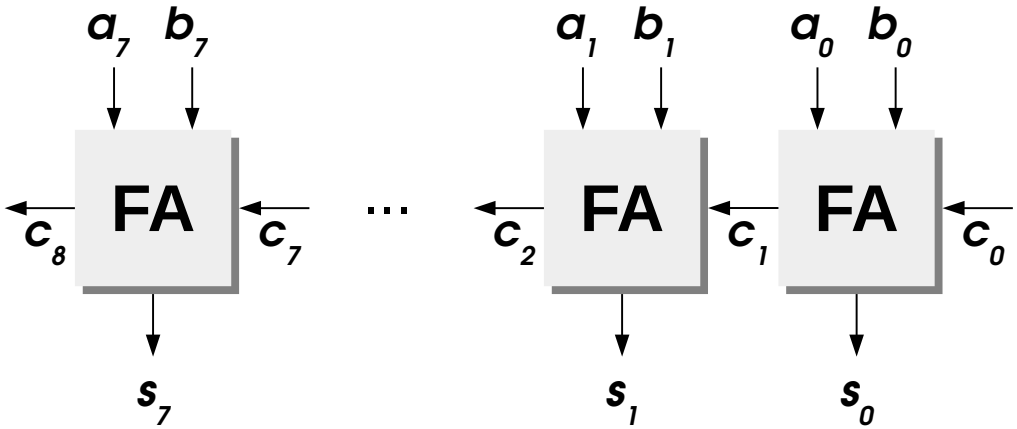
N-BIT ADDER



4-bit adder:

- reference encoding: 26 variables, 58 clauses
- optimal encoding: 12 variables, 336 clauses

N-BIT ADDER



4-bit adder:

- reference encoding: 26 variables, 58 clauses
- optimal encoding: 12 variables, 336 clauses

Problem

No auxiliary variables!

Auxiliary variables

Greedily extend vocabulary with *heuristically chosen* auxiliary variables Aux and their definitions Def.

```
1  E ← OPE(E_0, E_ref, Σ)
2  while Aux ≠ ∅ do
3      foreach aux ∈ Aux do
4          E′ ← OPE(E_0, E_ref ∧ Def(aux), Σ ∪ {aux})
5          if |E′| < |E| then
6              E ← E′
7              Σ ← Σ ∪ {aux}
8              E_ref ← E_ref ∪Def(aux)
9          Aux ← Aux \{aux}
10 return E
```

AUXILIARY VARIABLES

EXAMPLE

$$\mathsf{Aux} = \{v_1, v_2, v_3 \ldots\}$$

with definitions:

$$\begin{aligned}
\mathsf{Def}(v_1): \quad & v_1 \quad \Leftrightarrow \quad a_1 \wedge b_1 \\
\mathsf{Def}(v_2): \quad & v_2 \quad \Leftrightarrow \quad a_1 \oplus b_1 \\
\mathsf{Def}(v_2): \quad & v_3 \quad \Leftrightarrow \quad c_1 \wedge (a_1 \oplus b_1) \\
& \ldots
\end{aligned}$$

AUXILIARY VARIABLES

EXAMPLE

$$\mathsf{Aux} = \{v_1, v_2, v_3 \ldots\}$$

with definitions:

$$\begin{aligned}
\mathsf{Def}(v_1): \quad & v_1 \quad \Leftrightarrow \quad a_1 \wedge b_1 \\
\mathsf{Def}(v_2): \quad & v_2 \quad \Leftrightarrow \quad a_1 \oplus b_1 \\
\mathsf{Def}(v_2): \quad & v_3 \quad \Leftrightarrow \quad c_1 \wedge (a_1 \oplus b_1) \\
& \ldots
\end{aligned}$$

4-bit adder with auxiliary variables: 15 variables, 43 clauses

AUXILIARY VARIABLES

EXAMPLE

$$\mathsf{Aux} = \{v_1, v_2, v_3 \ldots\}$$

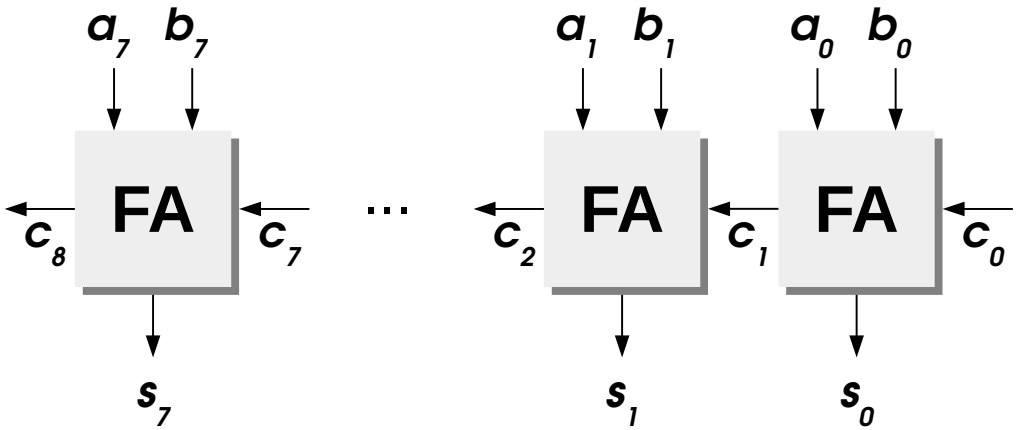
with definitions:

$$\begin{array}{lrcl} \mathsf{Def}(v_1): & v_1 & \Leftrightarrow & a_1 \wedge b_1 \\ \mathsf{Def}(v_2): & v_2 & \Leftrightarrow & a_1 \oplus b_1 \\ \mathsf{Def}(v_2): & v_3 & \Leftrightarrow & c_1 \wedge (a_1 \oplus b_1) \\ & \ldots & & \end{array}$$

4-bit adder with auxiliary variables: 15 variables, 43 clauses

- our algorithm learns the carry bit!

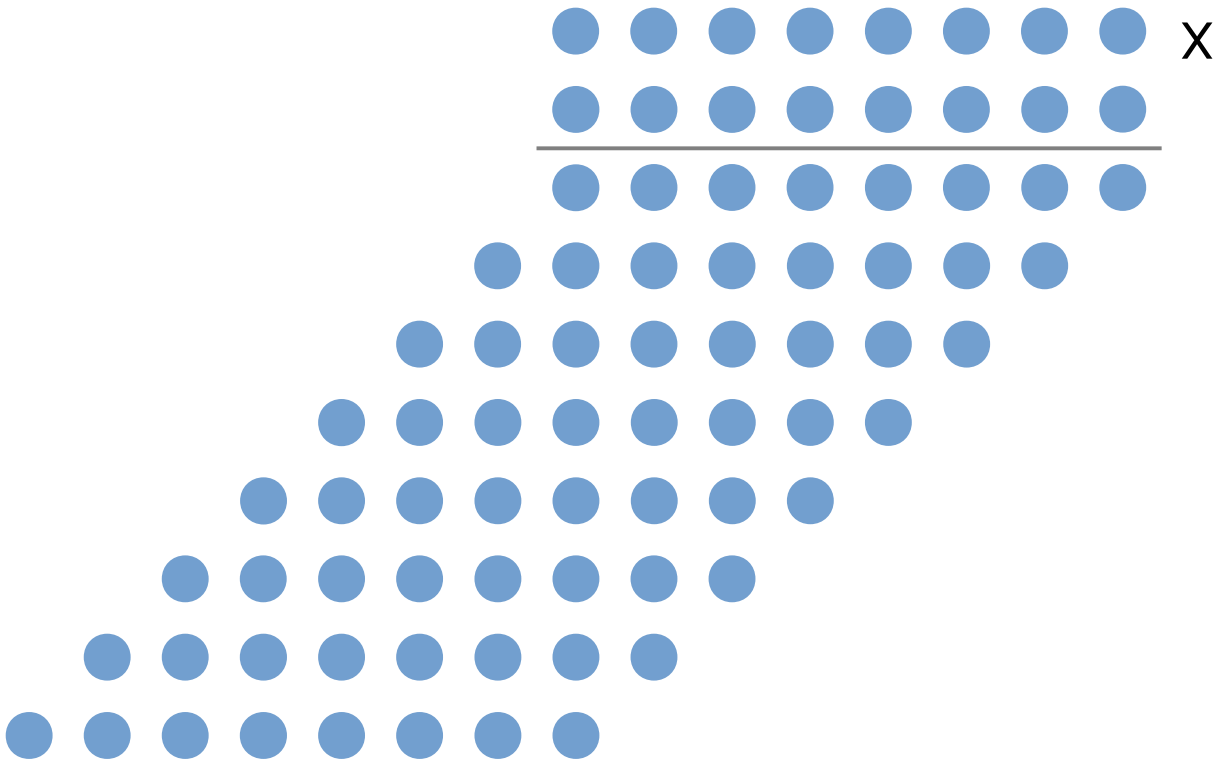
Encoding gadgets



- algorithm not meant to scale to n-bit encodings
- use small optimal gadgets and compose
- sometimes gadget composition is optimal

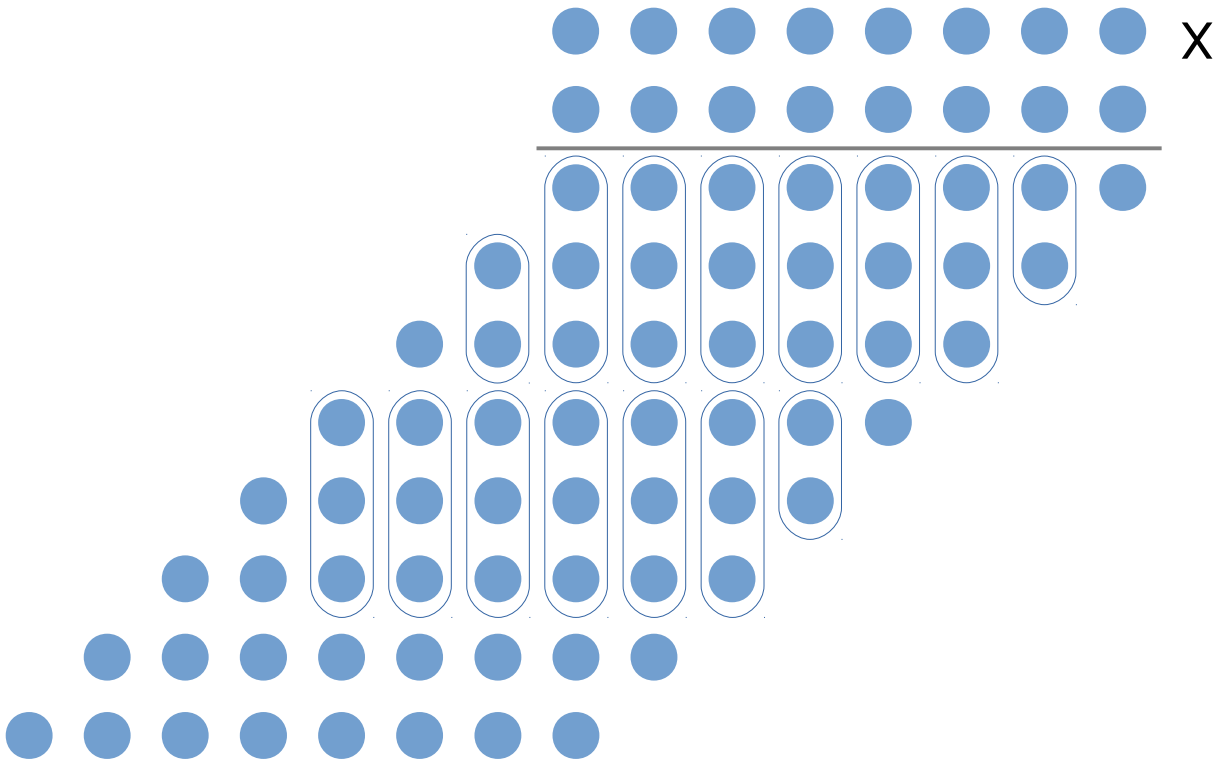
Examples of gadgets

Multipliers



Examples of gadgets

Multipliers

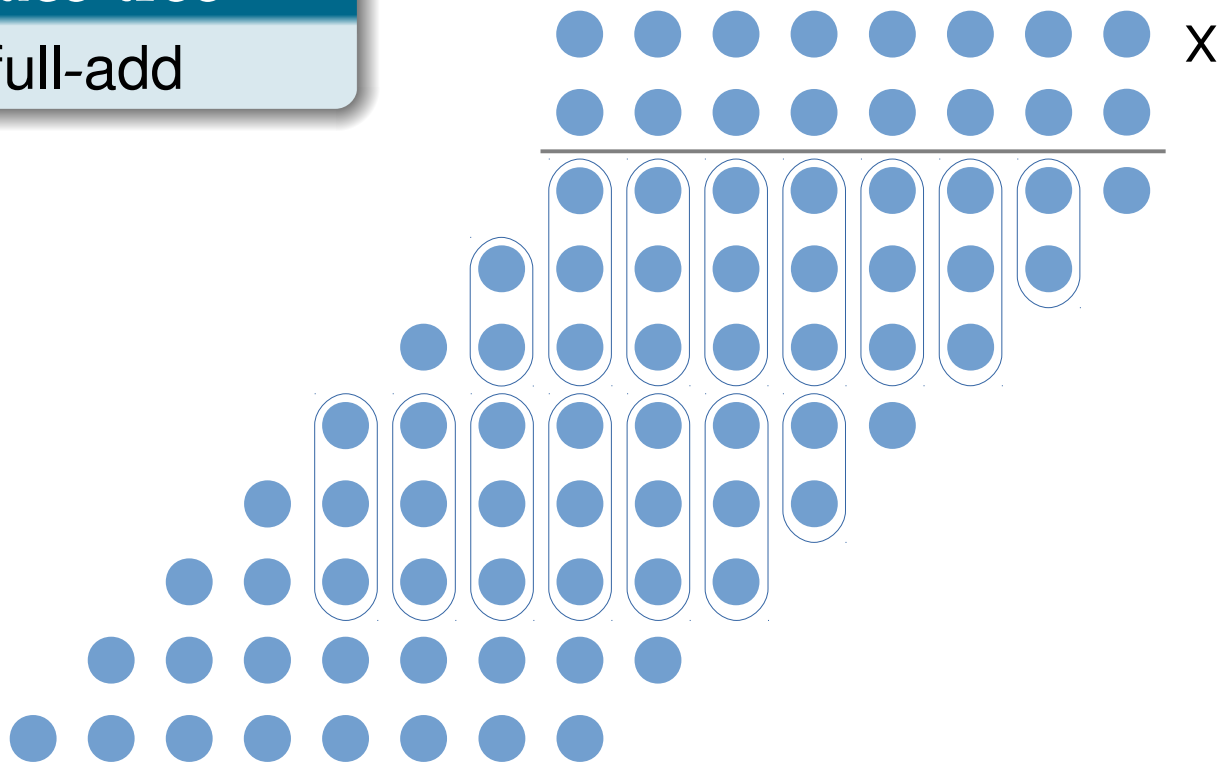


Examples of gadgets

Multipliers

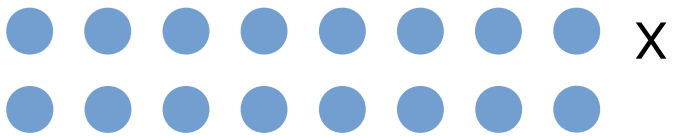
Wallace-tree

- full-add



Examples of gadgets

Multipliers

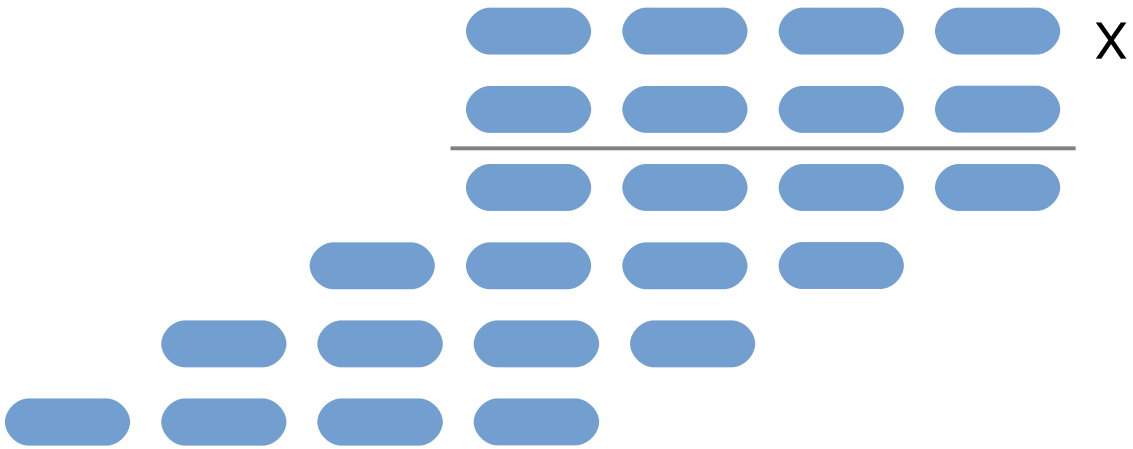


Examples of gadgets

Multipliers

Examples of gadgets

Multipliers

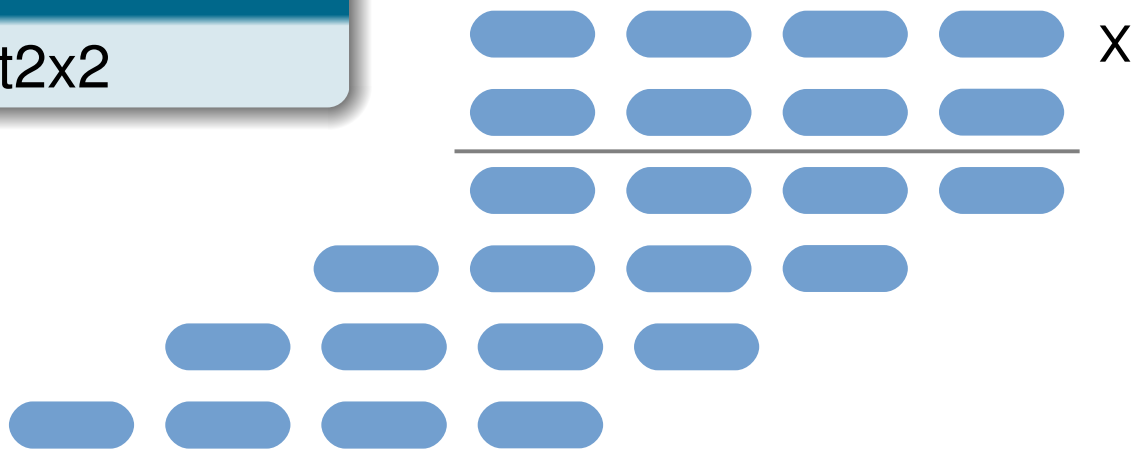


Examples of gadgets

Multipliers

Block2

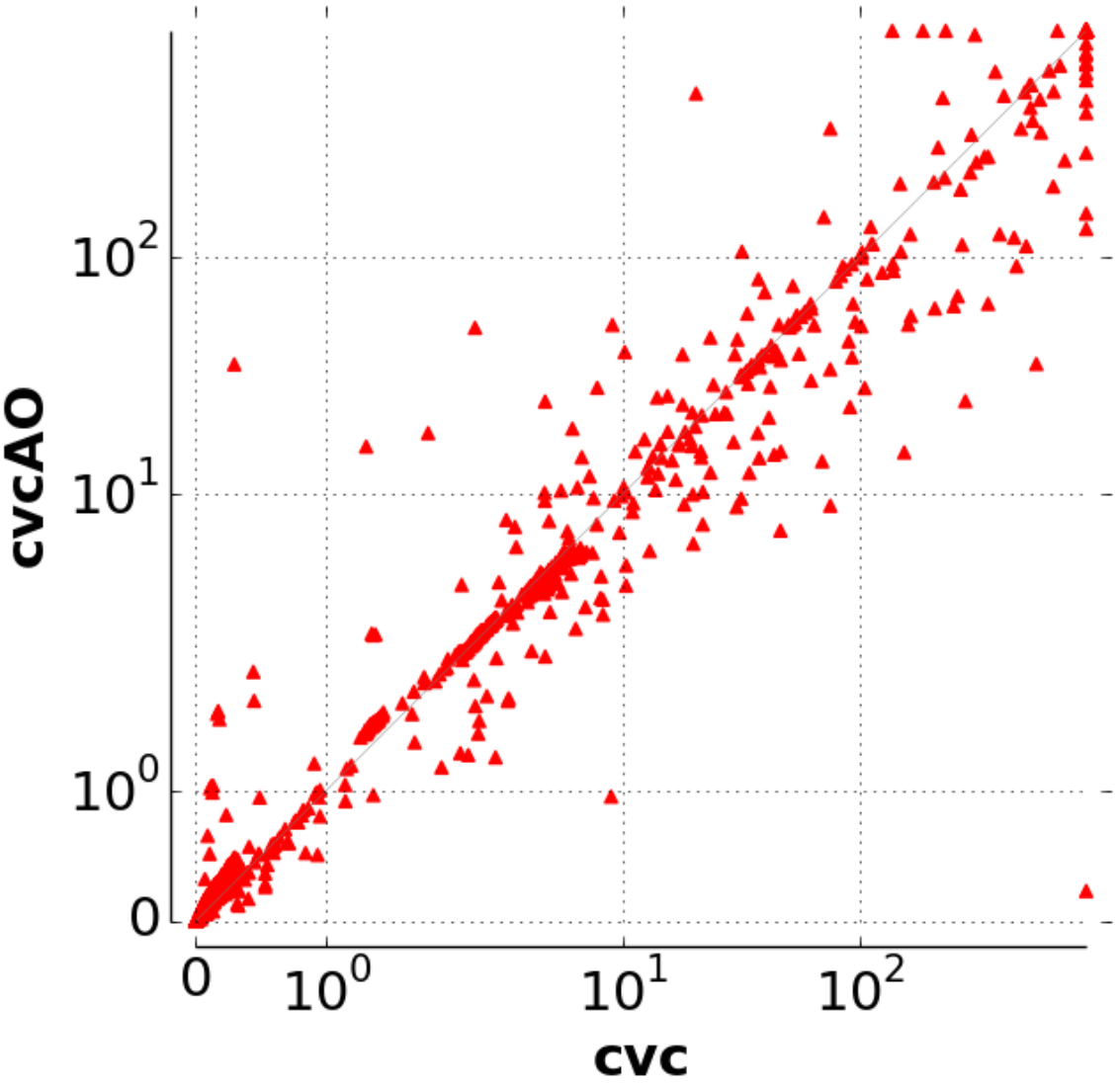
- mult2x2



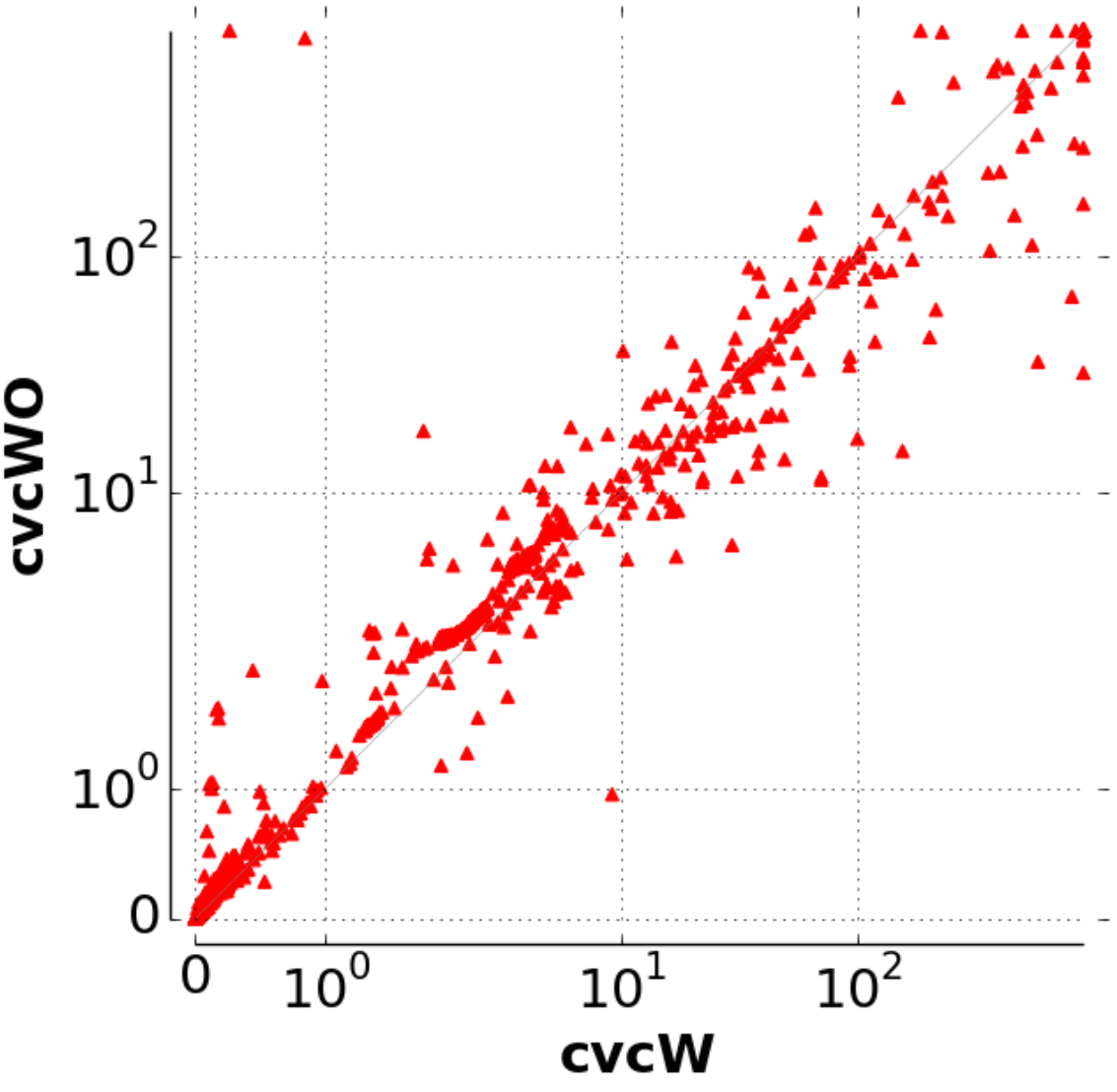
Generated encodings

		Original enc.		Optimal enc.			Optimal enc. w/ aux. vars		
Benchmark	Type	#Vars	#Cls	#Vars	#Cls	#time (s)	#Vars	#Cls	#time (s)
ite-gadget	prim	4	6	4	6	<0.01	4	6	<0.01
ult-gadget	prim	5	10	4	6	<0.01	4	6	<0.01
slt-gadget	prim	4	6	4	6	<0.01	4	6	<0.01
full-add	prim	8	17	5	14	<0.01	5	14	<0.01
full-add-base4	prim	33	74	10	120	0.31	12	86	140.40
bc3to2	prim	20	46	8	76	0.03	10	57	5.32
bc7to3	prim	27	68	10	254	0.49	14	136	769.50
mult2	prim	30	66	8	19	0.02	8	19	0.50
mult-const3	prim	16	33	6	20	<0.01	6	20	0.03
mult-const5	prim	25	50	9	24	0.01	9	24	0.21
mult-const7	prim	38	105	9	32	0.01	9	32	0.62
ult-6bit	comp	33	68	13	158	27.73	15	38	timeout
add-3bit	comp	18	39	9	96	0.09	11	29	10.05
add-4bit	comp	26	58	12	336	3.89	15	43	1,607.50
bc3to2-3bit	comp	38	78	12	1,536	11.72	16	69	timeout
mult-4bit	comp	36	97	12	670	5.26	12	670	298.95

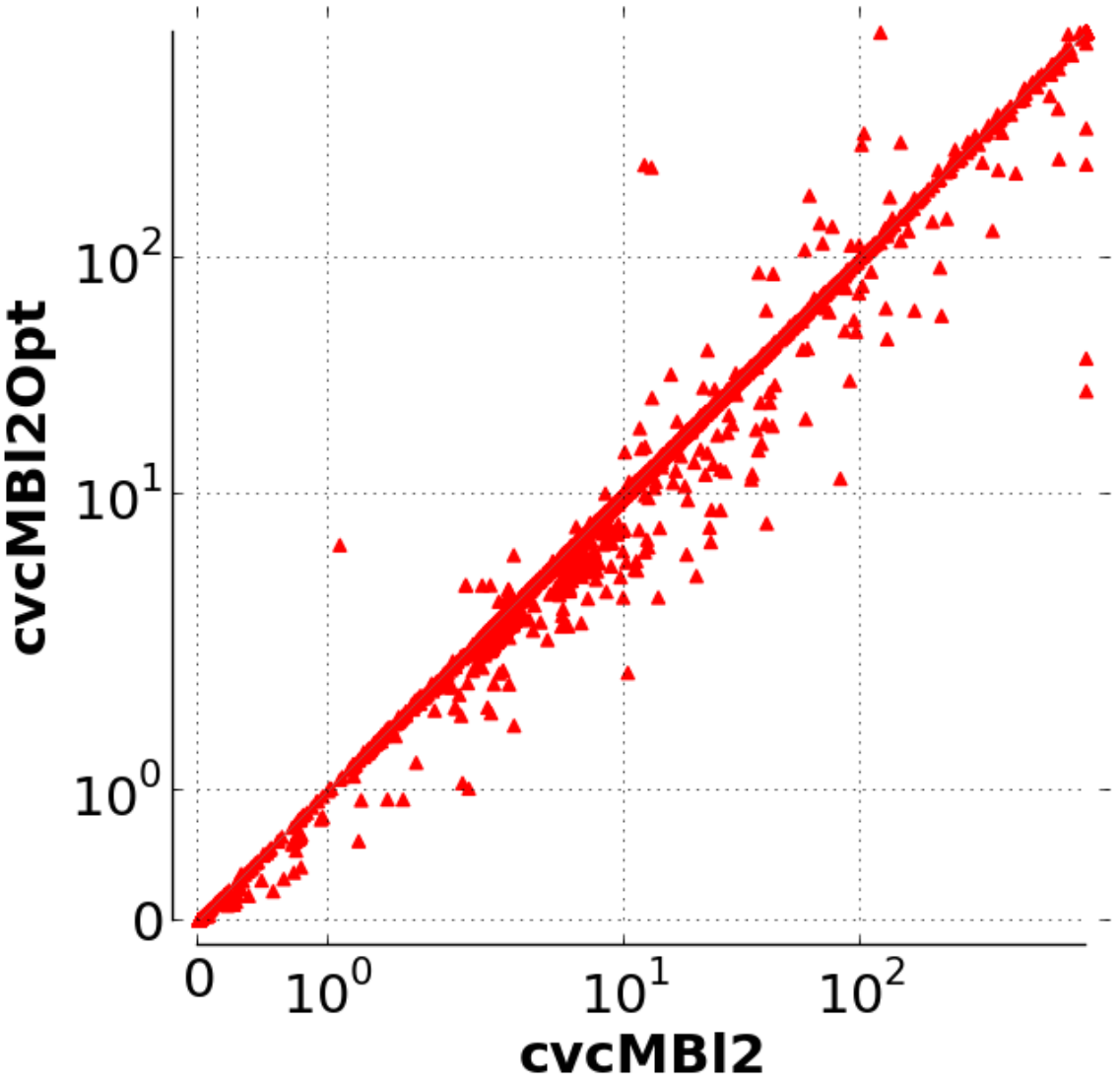
Performance impact



Performance impact*



Performance impact*



Performance impact*

	cvc		cvcAO		cvcALO		cvcW		cvcWO		cvcB2		cvcB2O	
set	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)
VS3 (11)	1	136.5	0	0.0	1	48.3	2	1328.9	0	0.0	**2**	**262.4**	1	359.7
bmc (135)	135	631.5	134	472.7	134	490.0	135	655.1	134	479.9	**135**	**613.1**	134	497.4
bmc14 (66)	64	1481.8	64	1480.3	64	1480.2	64	1427.7	64	1477.8	**64**	**1426.8**	64	1477.9
bb (52)	39	2606.2	39	2059.9	**39**	**1954.7**	39	2588.8	39	2079.8	39	2564.1	39	2090.9
bb3 (79)	40	3113.9	39	3591.0	**43**	**4981.5**	39	2370.0	39	4718.0	40	3464.8	40	4997.7
ca (23)	8	5.0	9	4.2	**11**	**470.7**	8	30.1	7	30.6	8	2.0	8	3.4
fft (23)	**8**	**857.1**	7	48.6	7	175.0	8	872.9	7	48.8	8	872.1	7	48.6
float (213)	159	12638.2	**172**	**15345.1**	165	9221.8	159	11305.1	170	13995.3	163	10799.0	170	13875.5
gul (6)	6	13.5	**6**	**6.8**	6	7.9	6	11.8	6	10.3	6	7.9	6	15.3
rubik (7)	6	605.3	6	608.6	**7**	**1379.1**	6	600.6	6	608.4	6	588.2	6	609.4
stp (1)	1	58.9	1	58.2	1	59.5	1	58.5	**1**	**57.7**	1	58.4	1	58.7
stps (426)	424	61.3	424	95.7	424	106.7	**424**	**61.3**	424	96.5	424	61.4	424	96.7
tacas (5)	5	1229.2	5	1185.2	5	1156.6	5	1231.5	5	1171.5	5	1224.7	**5**	**1140.6**
uclid (416)	416	1512.1	416	1516.5	416	1706.4	**416**	**1368.9**	416	1719.3	416	1403.1	416	1746.0
uum (8)	2	10.2	2	10.1	2	10.1	2	10.0	**2**	**10.0**	2	10.0	2	10.0
wien (18)	14	21.3	**14**	**15.1**	14	19.8	14	20.3	14	28.4	14	18.5	14	21.5
	1328	24982.0	1338	26498.0	**1339**	**23268.3**	1328	23941.6	1334	26532.2	1333	23376.5	1337	27049.4

Performance impact*

	cvc		cvcAO		cvcALO		cvcW		cvcWO		cvcB2		cvcB2O	
set	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)
VS3 (11)	1	136.5	0	0.0	1	48.3	2	1328.9	0	0.0	**2**	**262.4**	1	359.7
bmc (135)	135	631.5	134	472.7	134	490.0	135	655.1	134	479.9	**135**	**613.1**	134	497.4
bmc14 (66)	64	1481.8	64	1480.3	64	1480.2	64	1427.7	64	1477.8	**64**	**1426.8**	64	1477.9
bb (52)	39	2606.2	39	2059.9	**39**	**1954.7**	39	2588.8	39	2079.8	39	2564.1	39	2090.9
bb3 (79)	40	3113.9	39	3591.0	**43**	**4981.5**	39	2370.0	39	4718.0	40	3464.8	40	4997.7
ca (23)	8	5.0	9	4.2	**11**	**470.7**	8	30.1	7	30.6	8	2.0	8	3.4
fft (23)	**8**	**857.1**	7	48.6	7	175.0	8	872.9	7	48.8	8	872.1	7	48.6
float (213)	159	12638.2	**172**	**15345.1**	165	9221.8	159	11305.1	170	13995.3	163	10799.0	170	13875.5
gul (6)	6	13.5	**6**	**6.8**	6	7.9	6	11.8	6	10.3	6	7.9	6	15.3
rubik (7)	6	605.3	6	608.6	**7**	**1379.1**	6	600.6	6	608.4	6	588.2	6	609.4
stp (1)	1	58.9	1	58.2	1	59.5	1	58.5	**1**	**57.7**	1	58.4	1	58.7
stps (426)	424	61.3	424	95.7	424	106.7	**424**	**61.3**	424	96.5	424	61.4	424	96.7
tacas (5)	5	1229.2	5	1185.2	5	1156.6	5	1231.5	5	1171.5	5	1224.7	**5**	**1140.6**
uclid (416)	416	1512.1	416	1516.5	416	1706.4	**416**	**1368.9**	416	1719.3	416	1403.1	416	1746.0
uum (8)	2	10.2	2	10.1	2	10.1	2	10.0	**2**	**10.0**	2	10.0	2	10.0
wien (18)	14	21.3	**14**	**15.1**	14	19.8	14	20.3	14	28.4	14	18.5	14	21.5
	1328	24982.0	1338	26498.0	**1339**	**23268.3**	1328	23941.6	1334	26532.2	1333	23376.5	1337	27049.4

*except of brummayerbiere2!

More formally

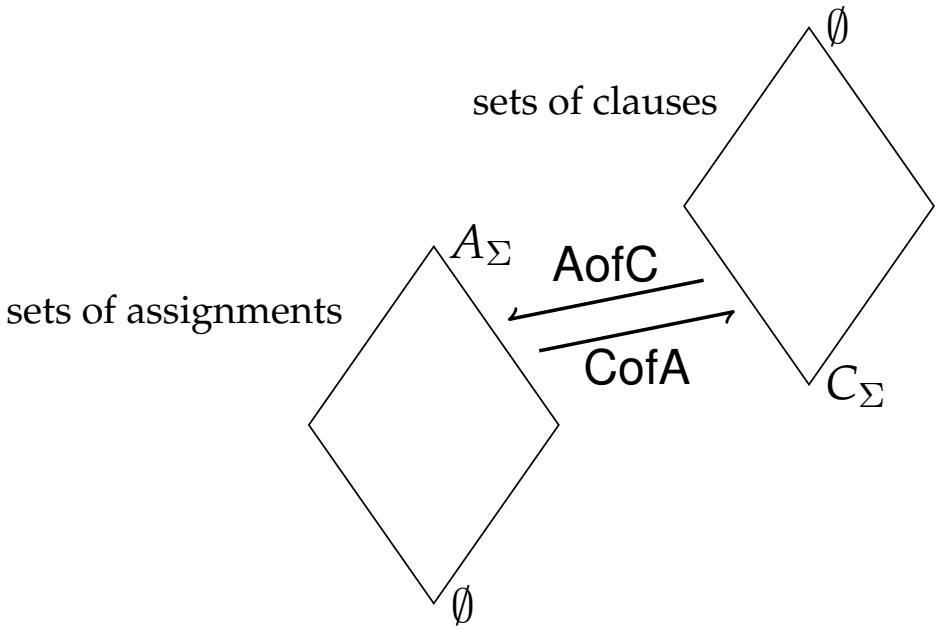
Assignments that satisfy clauses C:

$$\mathsf{AofC}(C) = \{\mathsf{a} \in A_\Sigma | \forall c \in C \,.\, \mathsf{a} \models c\}$$

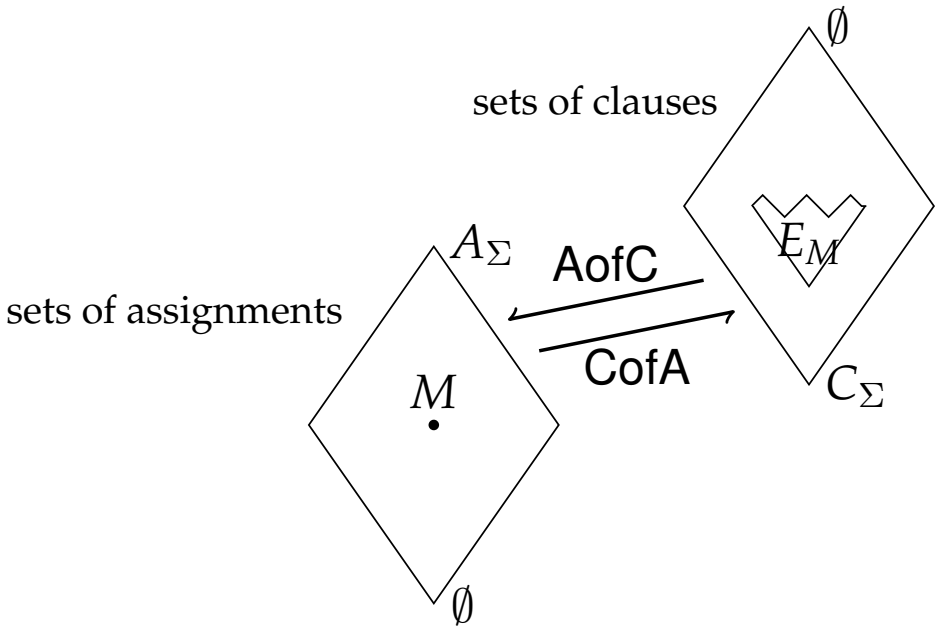
Clauses satisfied by assignments in A:

$$\mathsf{CofA}(A) = \{c \in C_\Sigma | \forall \mathsf{a} \in A \,.\, \mathsf{a} \models c\}$$

As a Picture


∅
sets of clauses
A_Σ
AofC
sets of assignments
CofA
C_Σ
∅

The Picture Continued



Partial assignment lattice

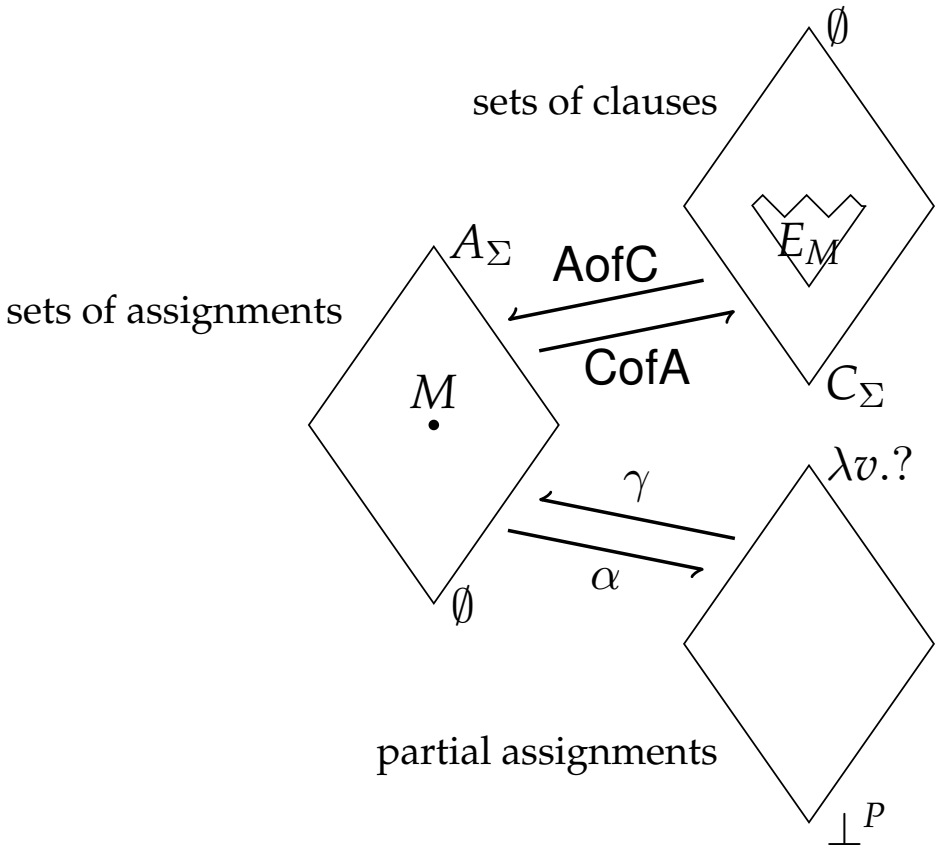
Order partial assignments as follows:

$$\mathsf{p}_1 \sqsubseteq \mathsf{p}_2 \Leftrightarrow \forall v \in \Sigma.\mathsf{p}_2(v) \neq ? \Rightarrow \mathsf{p}_1(v) = \mathsf{p}_2(v)$$

Galois connection between 2^{A_Σ} and P_Σ:

- $\alpha(A)$ most complete partial assignment consistent with all $\mathsf{a} \in A$
- $\gamma(\mathsf{p})$ all assignments that extend p

The Picture Continued



$\alpha(A)$ most complete partial assignment consistent with all $\mathsf{a} \in A$
$\gamma(\mathsf{p})$ all assignments that extend p

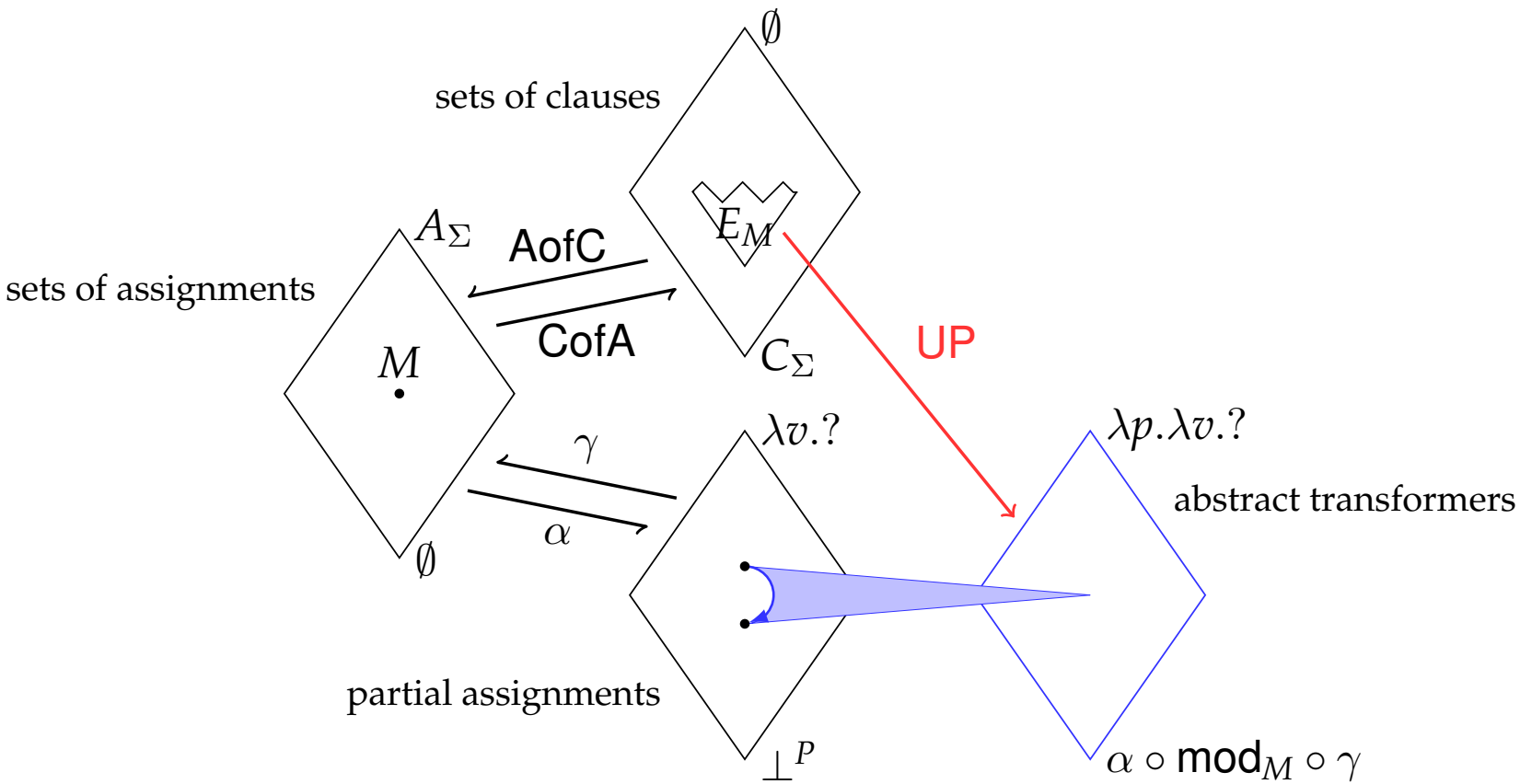
Unit propagation

$$\mathsf{UP} : 2^{C_\Sigma} \times P_\Sigma \to P_\Sigma$$

Unit propagation

$$\mathsf{UP} : 2^{C_\Sigma} \to (P_\Sigma \to P_\Sigma)$$

The Picture Complete



Conclusion

- propagation power is a useful criteria for encoding design
- but not the only one:
  - effect on learning, decisions etc.
- abstract satisfaction offers a good way to think about encodings
- other applications: comparison of encodings, compositionality etc.

References I

[1] Biere, Armin and Le Berre, Daniel and Lonca, Emmanuel and Manthey, Norbert. “Detecting cardinality constraints in CNF”. SAT 2014