Applications for Automated Reasoning

Marijn J.H. Heule

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f22/

Automated Reasoning and Satisfiability

August 31, 2022

Automated Reasoning Has Many Applications

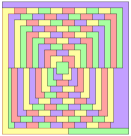
formal verification

security

bioinformatics

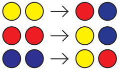
planning and scheduling

train safety



automated theorem proving

exploit generation



term rewriting termination

encode

automated reasoning

decode

Automated Reasoning Has Many Applications

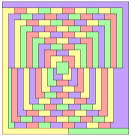
formal verification

security

bioinformatics

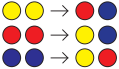
planning and scheduling

train safety



automated theorem proving

exploit generation

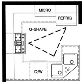


term rewriting termination

encode → **automated reasoning** → decode

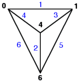
Microsoft IBM intel aws

Encoding problems into SAT

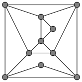


Architectural 3D Layout
[VSMM '07]
Henriette Bier

Edge-matching Puzzles
[LaSh '08]

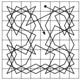


Graceful Graphs
[AAAI '10]
Toby Walsh

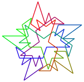


Clique-Width
[SAT '13, TOCL '15]
Stefan Szeider

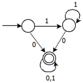
Firewall Verification
[SSS '16]
Mohamed Gouda



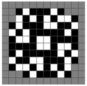
Open Knight Tours
Moshe Vardi



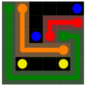
Van der Waerden numbers
[EJoC '07]



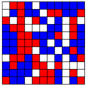
Software Model Synthesis
[ICGI '10, ESE '13]
Sicco Verwer



Conway's Game of Life
[EJoC '13]
Willem van der Poel



Connect the Pairs
Donald Knuth



Pythagorean Triples
[SAT '16, CACM '17]
Victor Marek

Collatz conjecture [Open]
Scott Aaronson

Equivalence Checking

Bounded Model Checking

Graphs and Symmetry Breaking

Arithmetic Operations

Equivalence Checking

Bounded Model Checking

Graphs and Symmetry Breaking

Arithmetic Operations

Equivalence checking introduction

Given two formulae, are they equivalent?

Applications:

- Hardware and software optimization
- Software to FPGA conversion

Equivalence checking example

original C code

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

Equivalence checking example

original C code

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

⇓

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

Equivalence checking example

original C code

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

$\Downarrow$

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

$\Rightarrow$

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

Equivalence checking example

original C code

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

$\Downarrow$

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

$\Rightarrow$

optimized C code

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

$\Uparrow$

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

Equivalence checking example

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(); }
```

⇒

Are these two code fragments equivalent?

Equivalence checking encoding (1)

1. represent procedures as Boolean variables

original C code :=

if $\overline{a} \wedge \overline{b}$ then h
else if $\overline{a}$ then g
else f

optimized C code :=

if a then f
else if b then g
else h

Equivalence checking encoding (1)

1. represent procedures as Boolean variables

original C code :=
if $\bar{a} \wedge \bar{b}$ then h
else if $\bar{a}$ then g
else f

optimized C code :=
if a then f
else if b then g
else h

2. compile code into Conjunctive Normal Form

   compile (if x then y else z) $\equiv (\bar{x} \vee y) \wedge (x \vee z)$

Equivalence checking encoding (1)

1. represent procedures as Boolean variables

original C code :=

$\texttt{if } \bar{a} \land \bar{b} \texttt{ then } h$
$\texttt{else if } \bar{a} \texttt{ then } g$
$\texttt{else } f$

optimized C code :=

$\texttt{if } a \texttt{ then } f$
$\texttt{else if } b \texttt{ then } g$
$\texttt{else } h$

2. compile code into Conjunctive Normal Form

   $\textit{compile}\,(\texttt{if } x \texttt{ then } y \texttt{ else } z) \equiv (\bar{x} \lor y) \land (x \lor z)$

3. check equivalence of Boolean formulae

   compile (**original C code**) $\Leftrightarrow$ *compile* (**optimized C code**)

Equivalence checking encoding (2)

compile (**original C code**):

$$
\begin{aligned}
&\texttt{if } \overline{a} \land \overline{b} \texttt{ then } h \texttt{ else if } \overline{a} \texttt{ then } g \texttt{ else } f && \equiv \\
&(\overline{(\overline{a} \land \overline{b})} \lor h) \land ((\overline{a} \land \overline{b}) \lor (\texttt{if } \overline{a} \texttt{ then } g \texttt{ else } f)) && \equiv \\
&(a \lor b \lor h) \land ((\overline{a} \land \overline{b}) \lor ((a \lor g) \land (\overline{a} \lor f)))
\end{aligned}
$$

Equivalence checking encoding (2)

compile (**original C code**):

$$
\begin{aligned}
&\texttt{if } \bar{a} \wedge \bar{b} \texttt{ then } h \texttt{ else if } \bar{a} \texttt{ then } g \texttt{ else } f && \equiv \\
&(\overline{(\bar{a} \wedge \bar{b})} \vee h) \wedge ((\bar{a} \wedge \bar{b}) \vee (\texttt{if } \bar{a} \texttt{ then } g \texttt{ else } f)) && \equiv \\
&(a \vee b \vee h) \wedge ((\bar{a} \wedge \bar{b}) \vee ((a \vee g) \wedge (\bar{a} \vee f)))
\end{aligned}
$$

compile (**optimized C code**):

$$
\begin{aligned}
&\texttt{if } a \texttt{ then } f \texttt{ else if } b \texttt{ then } g \texttt{ else } h && \equiv \\
&(\bar{a} \vee f) \wedge (a \vee (\texttt{if } b \texttt{ then } g \texttt{ else } h)) && \equiv \\
&(\bar{a} \vee f) \wedge (a \vee ((\bar{b} \vee g) \wedge (b \vee h)))
\end{aligned}
$$

Equivalence checking encoding (2)

compile (**original C code**):

$$
\begin{aligned}
&\texttt{if } \overline{a} \wedge \overline{b} \texttt{ then } h \texttt{ else if } \overline{a} \texttt{ then } g \texttt{ else } f && \equiv \\
&(\overline{(\overline{a} \wedge \overline{b})} \vee h) \wedge ((\overline{a} \wedge \overline{b}) \vee (\texttt{if } \overline{a} \texttt{ then } g \texttt{ else } f)) && \equiv \\
&(a \vee b \vee h) \wedge ((\overline{a} \wedge \overline{b}) \vee ((a \vee g) \wedge (\overline{a} \vee f)))
\end{aligned}
$$

compile (**optimized C code**):

$$
\begin{aligned}
&\texttt{if } a \texttt{ then } f \texttt{ else if } b \texttt{ then } g \texttt{ else } h && \equiv \\
&(\overline{a} \vee f) \wedge (a \vee (\texttt{if } b \texttt{ then } g \texttt{ else } h)) && \equiv \\
&(\overline{a} \vee f) \wedge (a \vee ((\overline{b} \vee g) \wedge (b \vee h)))
\end{aligned}
$$

$$
\begin{gathered}
(a \vee b \vee h) \wedge ((\overline{a} \wedge \overline{b}) \vee ((a \vee g) \wedge (\overline{a} \vee f))) \\
\Updownarrow \\
(\overline{a} \vee f) \wedge (a \vee ((\overline{b} \vee g) \wedge (b \vee h)))
\end{gathered}
$$

Checking (in)equivalence

Reformulate it as a satisfiability (SAT) problem:
Is there an assignment to a, b, f, g, *and* h, *which results in different evaluations of the compiled codes?*

Checking (in)equivalence

Reformulate it as a satisfiability (SAT) problem:
Is there an assignment to a, b, f, g, *and* h, *which results in different evaluations of the compiled codes?*

Is the Boolean formula

$$x \leftrightarrow ((a \lor b \lor h) \land ((\overline{a} \land \overline{b}) \lor ((a \lor g) \land (\overline{a} \lor f)))) \land$$
$$y \leftrightarrow ((\overline{a} \lor f) \land (a \lor ((\overline{b} \lor g) \land (b \lor h)))) \land$$
$$(x \lor y) \land (\overline{x} \lor \overline{y})$$

satisfiable?

Such an assignment would provide a counterexample

Checking (in)equivalence

Reformulate it as a satisfiability (SAT) problem:
Is there an assignment to a, b, f, g, *and* h, *which results in different evaluations of the compiled codes?*

Is the Boolean formula

$$x \leftrightarrow ((a \lor b \lor h) \land ((\overline{a} \land \overline{b}) \lor ((a \lor g) \land (\overline{a} \lor f))) \land$$
$$y \leftrightarrow ((\overline{a} \lor f) \land (a \lor ((\overline{b} \lor g) \land (b \lor h))) \land$$
$$(x \lor y) \land (\overline{x} \lor \overline{y})$$

satisfiable?

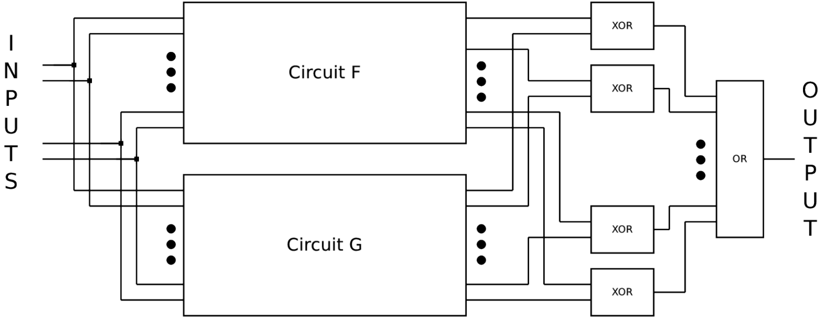
Such an assignment would provide a counterexample

Note: by concentrating on counterexamples we moved from co-NP to NP (not really important for applications)

Equivalence Checking via Miters

Equivalence checking is mostly used to validate whether two hardware designs (circuits) are functionally equivalent.

Given two circuits, a miter is circuit that tests whether there exists an input for both circuits such that the output differs.



Equivalence Checking

Bounded Model Checking

Graphs and Symmetry Breaking

Arithmetic Operations

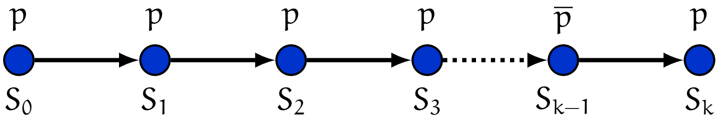
Bounded Model Checking (BMC)

Given a property p: (e.g. `signal_a = signal_b`)

Bounded Model Checking (BMC)

Given a property p: (e.g. `signal_a = signal_b`)

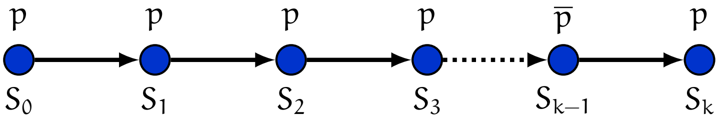
Is there a state reachable in k steps, which satisfies $\overline{p}$?



Bounded Model Checking (BMC)

Given a property p: (e.g. `signal_a = signal_b`)

Is there a state reachable in k steps, which satisfies $\overline{p}$?



Turing award 2007 for Model Checking
Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis

BMC Encoding (1)

Three components:

- I The description of the initial state
- T The transition of a state into the next state
- P The (safety) property

The reachable states in k steps are captured by:

$$I(S_0) \wedge T(S_0, S_1) \wedge \cdots \wedge T(S_{k-1}, S_k)$$

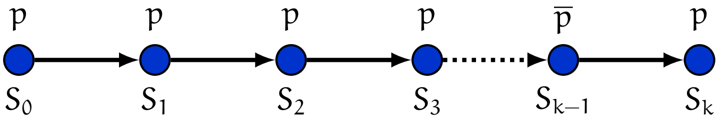
The property p fails in one of the k steps by:

$$\overline{P}(S_0) \vee \overline{P}(S_1) \vee \cdots \vee \overline{P}(S_k)$$

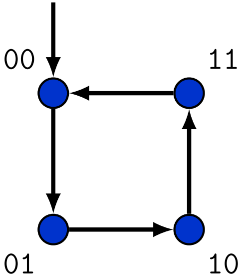
BMC Encoding (2)

The safety property p is valid up to step k
if and only if $F(k)$ is unsatisfiable:

$$F(k) = I(S_0) \wedge \bigwedge_{i=0}^{k-1} T(S_i, S_{i+1})) \wedge \bigvee_{i=0}^{k} \overline{P}(S_i)$$

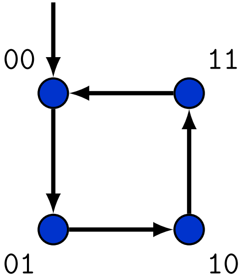


Bounded Model Checking Example: Two-bit counter



Initial state I:	$l_0 = 0, r_0 = 0$
Transition T:	$l_{i+1} = l_i \oplus r_i,$ $r_{i+1} = \bar{r}_i$
Property P:	$\bar{l}_i \vee \bar{r}_i$

Bounded Model Checking Example: Two-bit counter



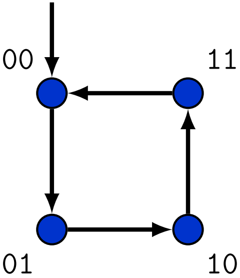
Initial state I: $l_0 = 0, r_0 = 0$

Transition T: $l_{i+1} = l_i \oplus r_i,$ $r_{i+1} = \bar{r}_i$

Property P: $\bar{l}_i \vee \bar{r}_i$

$$F(2) = (\bar{l}_0 \wedge \bar{r}_0) \wedge \left(\begin{array}{c} l_1 = l_0 \oplus r_0 \wedge r_1 = \bar{r}_0 \wedge \\ l_2 = l_1 \oplus r_1 \wedge r_2 = \bar{r}_1 \end{array} \right) \wedge \left(\begin{array}{c} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{array} \right)$$

Bounded Model Checking Example: Two-bit counter



Initial state I: $l_0 = 0, r_0 = 0$

Transition T: $l_{i+1} = l_i \oplus r_i,$
$r_{i+1} = \bar{r}_i$

Property P: $\bar{l}_i \vee \bar{r}_i$

$$F(2) = (\bar{l}_0 \wedge \bar{r}_0) \wedge \begin{pmatrix} l_1 = l_0 \oplus r_0 \wedge r_1 = \bar{r}_0 \wedge \\ l_2 = l_1 \oplus r_1 \wedge r_2 = \bar{r}_1 \end{pmatrix} \wedge \begin{pmatrix} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{pmatrix}$$

For $k = 2$, $F(k)$ is unsatisfiable; for $k = 3$ it is satisfiable

Equivalence Checking

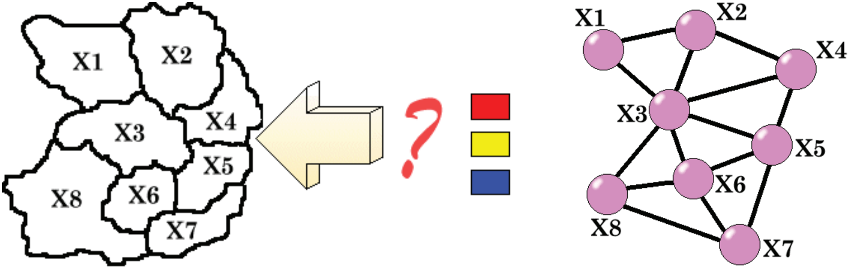
Bounded Model Checking

Graphs and Symmetry Breaking

Arithmetic Operations

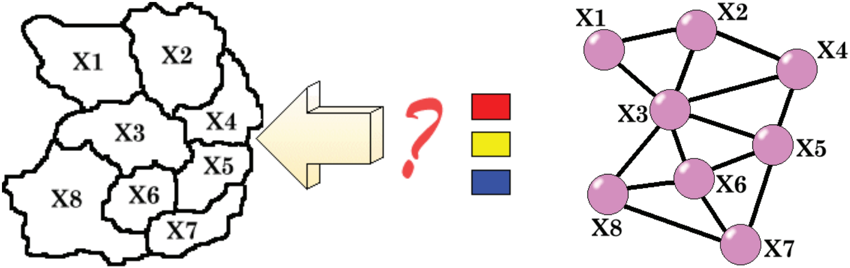
Graph coloring

Given a graph $G(V, E)$, can the vertices be colored with k colors such that for each edge $(v, w) \in E$, the vertices v and w are colored differently.



Graph coloring

Given a graph $G(V, E)$, can the vertices be colored with k colors such that for each edge $(v, w) \in E$, the vertices v and w are colored differently.



Problem: Many symmetries!!!

Graph coloring encoding

Variables	Range	Meaning
$x_{v,i}$	$i \in \{1, \ldots, c\}$ $v \in \{1, \ldots, \|V\|\}$	node v has color i

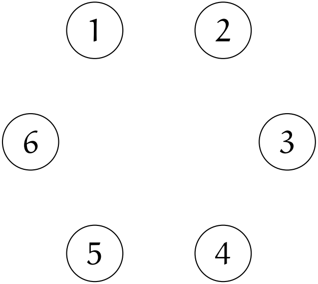
Clauses	Range	Meaning
$(x_{v,1} \lor x_{v,2} \lor \cdots \lor x_{v,c})$	$v \in \{1, \ldots, \|V\|\}$	v is colored
$(\overline{x}_{v,s} \lor \overline{x}_{v,t})$	$s \in \{1, \ldots, c-1\}$ $t \in \{s+1, \ldots, c\}$	v has at most one color
$(\overline{x}_{v,i} \lor \overline{x}_{w,i})$	$(v, w) \in E$	v and w have a different color
???	???	breaking symmetry

Unavoidable Subgraphs and Ramsey Numbers

A connected undirected graph G is an unavoidable subgraph of clique K of order n if any red/blue edge-coloring of the edges of K contains G either in red or in blue.

Ramsey Number $R(k)$: What is the smallest n such that any graph with n vertices has either a clique or a co-clique of size k?

$$\begin{aligned} R(3) &= 6 \\ R(4) &= 18 \\ 43 \leq R(5) &\leq 49 \end{aligned}$$

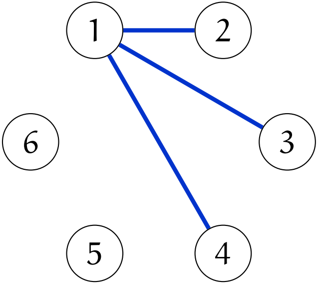


Unavoidable Subgraphs and Ramsey Numbers

A connected undirected graph G is an unavoidable subgraph of clique K of order n if any red/blue edge-coloring of the edges of K contains G either in red or in blue.

Ramsey Number $R(k)$: What is the smallest n such that any graph with n vertices has either a clique or a co-clique of size k?

$$\begin{aligned} R(3) &= 6 \\ R(4) &= 18 \\ 43 \leq\ R(5) &\leq 49 \end{aligned}$$

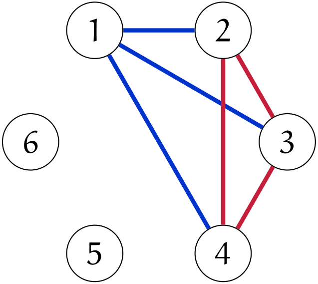


Unavoidable Subgraphs and Ramsey Numbers

A connected undirected graph G is an unavoidable subgraph of clique K of order n if any red/blue edge-coloring of the edges of K contains G either in red or in blue.

Ramsey Number $R(k)$: What is the smallest n such that any graph with n vertices has either a clique or a co-clique of size k?

$$\begin{aligned} R(3) &= 6 \\ R(4) &= 18 \\ 43 \leq \quad R(5) &\leq 49 \end{aligned}$$

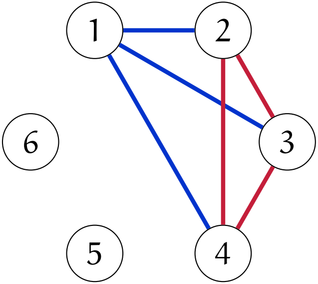


Unavoidable Subgraphs and Ramsey Numbers

A connected undirected graph G is an unavoidable subgraph of clique K of order n if any red/blue edge-coloring of the edges of K contains G either in red or in blue.

Ramsey Number $R(k)$: What is the smallest n such that any graph with n vertices has either a clique or a co-clique of size k?

$$R(3) = 6$$
$$R(4) = 18$$
$$43 \leq R(5) \leq 49$$



SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

Example formula: an unavoidable path of two edges

Consider the formula below — which expresses the statement whether path of two edges unavoidable in a clique of order 3:

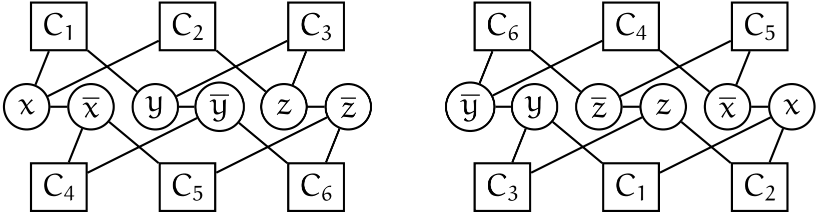
$$F := \overbrace{(x \vee y)}^{C_1} \wedge \overbrace{(x \vee z)}^{C_2} \wedge \overbrace{(y \vee z)}^{C_3} \wedge \overbrace{(\overline{x} \vee \overline{y})}^{C_4} \wedge \overbrace{(\overline{x} \vee \overline{z})}^{C_5} \wedge \overbrace{(\overline{y} \vee \overline{z})}^{C_6}$$

Example formula: an unavoidable path of two edges

Consider the formula below — which expresses the statement whether path of two edges unavoidable in a clique of order 3:

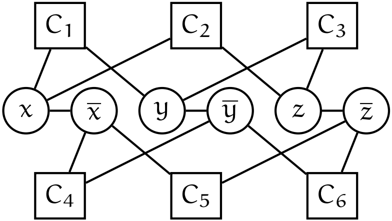
$$F := \overbrace{(x \vee y)}^{C_1} \wedge \overbrace{(x \vee z)}^{C_2} \wedge \overbrace{(y \vee z)}^{C_3} \wedge \overbrace{(\overline{x} \vee \overline{y})}^{C_4} \wedge \overbrace{(\overline{x} \vee \overline{z})}^{C_5} \wedge \overbrace{(\overline{y} \vee \overline{z})}^{C_6}$$

A clause-literal graph has a vertex for each clause and literal, and edges for each literal occurrence connecting the literal and clause vertex. Also, two complementary literals are connected.

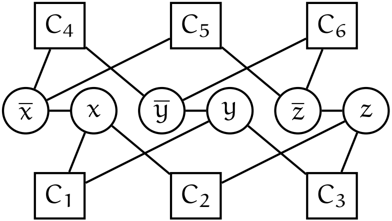


Symmetry: $(x,y,z)(\overline{y},\overline{z},\overline{x})$ is an edge-preserving bijection

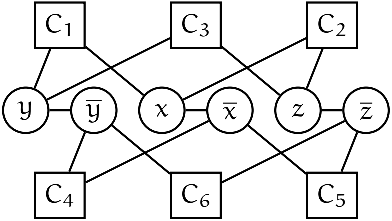
Three Symmetries of the Example Formula



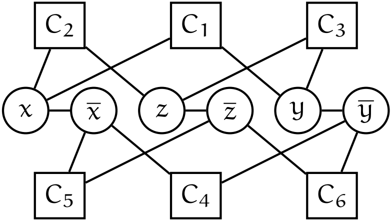
identity symmetry



$(x, y, z, C_1, C_2, C_3, C_4, C_5, C_6)$
$(\bar{x}, \bar{y}, \bar{z}, C_4, C_5, C_6, C_1, C_2, C_3)$



$(x, y, C_2, C_5, C_3, C_6)$
$(y, x, C_3, C_6, C_2, C_5)$



$(y, z, C_1, C_4, C_2, C_5)$
$(z, y, C_2, C_5, C_1, C_4)$

Convert Symmetries into Symmetry-Breaking Predicates

A symmetry $\sigma = (x_1, \ldots, x_n)(p_1, \ldots, p_n)$ of a CNF formula F is an edge-preserving bijection of the clause-literal graph of F, that maps literals x_i onto p_i and $\overline{x}_i$ onto $\overline{p}_i$ with $i \in \{1, \ldots, n\}$

Given a CNF formula F. Let α be a satisfying truth assignment for F and σ a symmetry for F, then $\sigma(\alpha)$ is also a satisfying truth assignment for F.

Symmetry $\sigma = (x_1, \ldots, x_n)(p_1, \ldots, p_n)$ for F can be broken by adding a symmetry-breaking predicate:
$x_1, \ldots, x_n \leq p_1, \ldots, p_n$.

$$(\overline{x}_1 \vee p_1) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee p_2) \wedge (p_1 \vee \overline{x}_2 \vee p_2) \wedge$$
$$(\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee p_3) \wedge (\overline{x}_1 \vee p_2 \vee \overline{x}_3 \vee p_3) \wedge$$
$$(p_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee p_3) \wedge (p_1 \vee p_2 \vee \overline{x}_3 \vee p_3) \wedge \ldots$$

Symmetry Breaking in Practice

In practice, symmetry breaking is mostly used as a preprocessing technique.

A given CNF formula is first transformed into a clause-literal graph. Symmetries are detected in the clause-literal graph. An efficient tool for this is `saucy`.

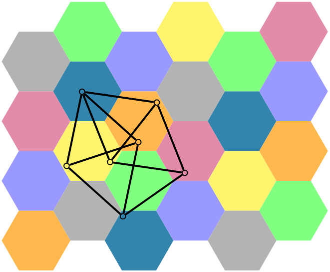
The symmetries can broken by adding symmetry-breaking predicates to the given CNF.

Many hard problems for resolution, such as pigeon hole formulas, can be solved instantly after symmetry-breaking predicates are added.

Chromatic Number of the Plane [Nelson '50]

How many colors are required to color the plane such that each pair of points that are exactly 1 apart are colored differently?

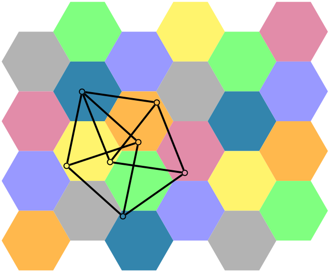
- The Moser Spindle graph shows the lower bound of 4
- A colored tiling of the plane shows the upper bound of 7
- Lower bound of 5 [DeGrey '18] based on a 1581-vertex graph



Chromatic Number of the Plane [Nelson '50]

How many colors are required to color the plane such that each pair of points that are exactly 1 apart are colored differently?

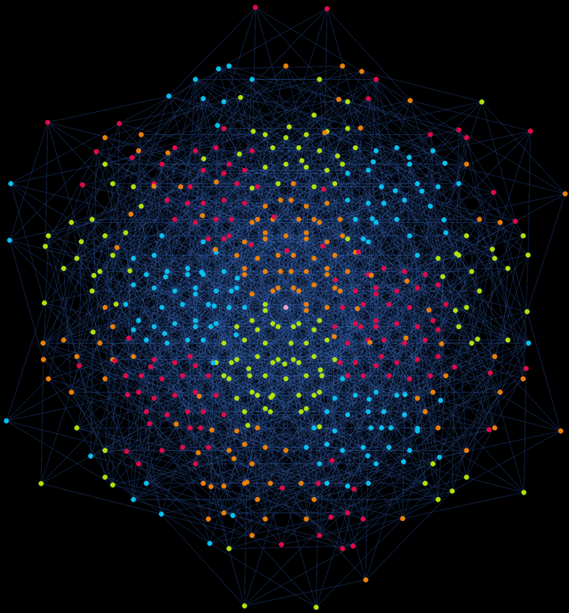
- The Moser Spindle graph shows the lower bound of 4
- A colored tiling of the plane shows the upper bound of 7
- Lower bound of 5 [DeGrey '18] based on a 1581-vertex graph



We found smaller graphs with SAT:

- 874 vertices on April 14, 2018
- 803 vertices on April 30, 2018
- 610 vertices on May 14, 2018

Record by Proof Minimization: 529 Vertices [Heule 2019]



Equivalence Checking

Bounded Model Checking

Graphs and Symmetry Breaking

Arithmetic Operations

Arithmetic operations: Introduction

How to encode arithmetic operations into SAT?

Arithmetic operations: Introduction

How to encode arithmetic operations into SAT?

Efficient encoding using electronic circuits

Arithmetic operations: Introduction

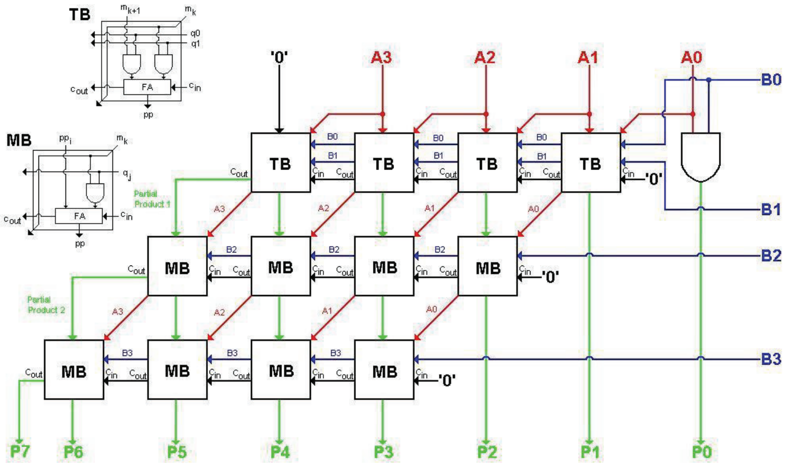
How to encode arithmetic operations into SAT?

Efficient encoding using electronic circuits

Applications:

- factorization (not competitive)
- term rewriting

Arithmetic operations: 4x4 Multiplier circuit



Arithmetic operations: Multiplier encoding

1. Multiplication $m_{i,j} = x_i \times y_j = \textsc{And}\ (x_i, y_j)$
$(m_{i,j} \lor \overline{x}_i \lor \overline{y}_j) \land (\overline{m}_{i,j} \lor x_i) \land (\overline{m}_{i,j} \lor y_j)$

Arithmetic operations: Multiplier encoding

1. Multiplication $m_{i,j} = x_i \times y_j = \textsc{And}\ (x_i, y_j)$
$(m_{i,j} \lor \overline{x}_i \lor \overline{y}_j) \land (\overline{m}_{i,j} \lor x_i) \land (\overline{m}_{i,j} \lor y_j)$

2. Carry out $c_{out} = 1$ if and only if $p_{in} + m_{i,j} + c_{in} > 1$
$(c_{out} \lor \overline{p}_{in} \lor \overline{m}_{i,j}) \land (c_{out} \lor \overline{p}_{in} \lor \overline{c}_{in}) \land (c_{out} \lor \overline{m}_{i,j} \lor \overline{c}_{in}) \land$
$(\overline{c}_{out} \lor p_{in} \lor m_{i,j}) \land (\overline{c}_{out} \lor p_{in} \lor c_{in}) \land (\overline{c}_{out} \lor m_{i,j} \lor c_{in})$

Arithmetic operations: Multiplier encoding

1. Multiplication $m_{i,j} = x_i \times y_j = \textsc{And}\ (x_i, y_j)$

$$(m_{i,j} \lor \overline{x}_i \lor \overline{y}_j) \land (\overline{m}_{i,j} \lor x_i) \land (\overline{m}_{i,j} \lor y_j)$$

2. Carry out $c_{out} = 1$ if and only if $p_{in} + m_{i,j} + c_{in} > 1$

$$(c_{out} \lor \overline{p}_{in} \lor \overline{m}_{i,j}) \land (c_{out} \lor \overline{p}_{in} \lor \overline{c}_{in}) \land (c_{out} \lor \overline{m}_{i,j} \lor \overline{c}_{in}) \land$$
$$(\overline{c}_{out} \lor p_{in} \lor m_{i,j}) \land (\overline{c}_{out} \lor p_{in} \lor c_{in}) \land (\overline{c}_{out} \lor m_{i,j} \lor c_{in})$$

3. Parity out p_{out} of variables p_{in}, $m_{i,j}$ and c_{in}

$$(p_{out} \lor \overline{p}_{in} \lor \overline{m}_{i,j} \lor \overline{c}_{in}) \land \quad (p_{out} \lor p_{in} \lor m_{i,j} \lor \overline{c}_{in}) \land$$
$$(\overline{p}_{out} \lor p_{in} \lor \overline{m}_{i,j} \lor \overline{c}_{in}) \land \quad (p_{out} \lor p_{in} \lor \overline{m}_{i,j} \lor c_{in}) \land$$
$$(\overline{p}_{out} \lor \overline{p}_{in} \lor m_{i,j} \lor \overline{c}_{in}) \land \quad (p_{out} \lor \overline{p}_{in} \lor m_{i,j} \lor c_{in}) \land$$
$$(\overline{p}_{out} \lor \overline{p}_{in} \lor \overline{m}_{i,j} \lor c_{in}) \land \quad (\overline{p}_{out} \lor p_{in} \lor m_{i,j} \lor c_{in})$$

Arithmetic operations: Is 27 prime?

			x_3	x_2	x_1	x_0	
			x_3y_0	x_2y_0	x_1y_0	x_0y_0	y_0
		x_3y_1	x_2y_1	x_1y_1	x_0y_1		y_1
	x_3y_2	x_2y_2	x_1y_2	x_0y_2			y_2
x_3y_3	x_2y_3	x_1y_3	x_0y_3				y_3
0	0	1	1	0	1	1	

Arithmetic operations: Is 27 prime?

$$
\begin{array}{cccccccc}
 & & & x_3 & x_2 & x_1 & x_0 & \\
 & & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & y_0 \\
 & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & & y_1 \\
 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & & y_2 \\
x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 1 &
\end{array}
$$

Prime: $(x_1 \lor x_2 \lor x_3) \land (y_1 \lor y_2 \lor y_3)$

Arithmetic operations: Is 27 prime?

			x_3	x_2	x_1	x_0	
			x_3y_0	x_2y_0	x_1y_0	x_0y_0	y_0
		x_3y_1	x_2y_1	x_1y_1	x_0y_1		y_1
	x_3y_2	x_2y_2	x_1y_2	x_0y_2			y_2
x_3y_3	x_2y_3	x_1y_3	x_0y_3				y_3
0	0	1	1	0	1	1	

Prime: $(x_1 \vee x_2 \vee x_3) \wedge (y_1 \vee y_2 \vee y_3)$

Arithmetic operations: Is 29 prime?

			x_3	x_2	x_1	x_0	
			x_3y_0	x_2y_0	x_1y_0	x_0y_0	y_0
		x_3y_1	x_2y_1	x_1y_1	x_0y_1		y_1
	x_3y_2	x_2y_2	x_1y_2	x_0y_2			y_2
x_3y_3	x_2y_3	x_1y_3	x_0y_3				y_3
0	0	1	1	1	0	1	

Prime: $(x_1 \vee x_2 \vee x_3) \wedge (y_1 \vee y_2 \vee y_3)$

Arithmetic operations: Is 29 prime?

$$
\begin{array}{cccccccc}
 & & & x_3 & x_2 & x_1 & x_0 & \\
 & & & x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & y_0 \\
 & & x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & & y_1 \\
 & x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & & & y_2 \\
x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 & 1 &
\end{array}
$$

Prime: $(x_1 \lor x_2 \lor x_3) \land (y_1 \lor y_2 \lor y_3)$

Arithmetic operations: Term rewriting

Given a set of rewriting rules,
will rewriting always terminate?

Arithmetic operations: Term rewriting

Given a set of rewriting rules,
will rewriting always terminate?

Example set of rules:

- $aa \rightarrow_R bc$
- $bb \rightarrow_R ac$
- $cc \rightarrow_R ab$

Arithmetic operations: Term rewriting

Given a set of rewriting rules,
will rewriting always terminate?

Example set of rules:

- $aa \rightarrow_R bc$
- $bb \rightarrow_R ac$
- $cc \rightarrow_R ab$

$$bb\underline{aa} \rightarrow_R b\underline{bb}c \rightarrow_R ba\underline{cc} \rightarrow_R b\underline{aa}b \rightarrow_R \underline{bb}cb \rightarrow_R$$
$$a\underline{cc}b \rightarrow_R aa\underline{bb} \rightarrow_R a\underline{aa}c \rightarrow_R ab\underline{cc} \rightarrow_R abab$$

Arithmetic operations: Term rewriting

Given a set of rewriting rules,
will rewriting always terminate?

Example set of rules:

- $aa \rightarrow_R bc$
- $bb \rightarrow_R ac$
- $cc \rightarrow_R ab$

$$bb\underline{aa} \rightarrow_R b\underline{bb}c \rightarrow_R ba\underline{cc} \rightarrow_R b\underline{aa}b \rightarrow_R \underline{bb}cb \rightarrow_R$$
$$a\underline{cc}b \rightarrow_R aa\underline{bb} \rightarrow_R a\underline{aa}c \rightarrow_R ab\underline{cc} \rightarrow_R abab$$

Strongest rewriting solvers use SAT (e.g. AProVE)

Example solved by Hofbauer, Waldmann (2006)

Arithmetic operations: Term rewriting proof outline

Proof termination of:

- $aa \rightarrow_R bc$
- $bb \rightarrow_R ac$
- $cc \rightarrow_R ab$

Proof outline:

- Interpret a,b,c by linear functions $[a], [b], [c]$ from $\mathbb{N}^4$ to $\mathbb{N}^4$
- Interpret string concatenation by function composition
- Show that if $[uaav]\ (0,0,0,0) = (x_1, x_2, x_3, x_4)$ and $[ubcv]\ (0,0,0,0) = (y_1, y_2, y_3, y_4)$ then $x_1 > y_1$
- Similar for $bb \rightarrow ac$ and $cc \rightarrow ab$
- Hence every rewrite step gives a decrease of $x_1 \in \mathbb{N}$, so rewriting terminates

Arithmetic operations: Term rewriting linear functions

The linear functions:

$$[a](\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 3 \\ 0 & 0 & 2 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$[b](\vec{x}) = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$$

$$[c](\vec{x}) = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \\ 3 \\ 0 \end{pmatrix}$$

Checking decrease properties using linear algebra

Arithmetic operations: Solving Mathematical Challenges

Recent articles in Quanta Magazine:

- **Computer Search Settles 90-Year-Old Math Problem** August 19, 2020
- **Computer Scientists Attempt to Corner the Collatz Conjecture** August 26, 2020
- **How Close Are Computers to Automating Mathematical Reasoning?** August 27, 2020

Arithmetic operations: Collatz

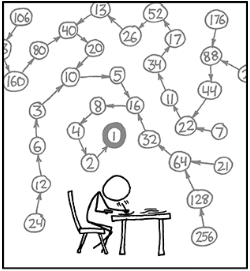
Resolving foundational algorithm questions

$$\mathrm{Col}(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ (3n+1)/2 & \text{if } n \text{ is odd} \end{cases}$$

$\texttt{while}(n > 1)\ n = \mathrm{Col}(n);$ terminates?

Find a non-negative function $\mathrm{fun}(n)$ s.t.

$$\forall n > 1 : \mathrm{fun}(n) > \mathrm{fun}(\mathrm{Col}(n))$$



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

source: xkcd.com/710

Arithmetic operations: Collatz

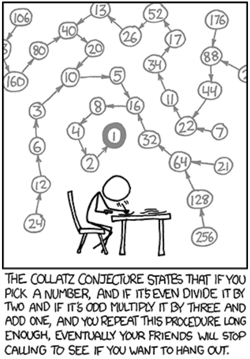
Resolving foundational algorithm questions

$$\mathtt{Col}(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ (3n+1)/2 & \text{if } n \text{ is odd} \end{cases}$$

$\mathtt{while}(n > 1)\ n = \mathtt{Col}(n);$ terminates?

Find a non-negative function $\mathtt{fun}(n)$ s.t.

$$\forall n > 1 : \mathtt{fun}(n) > \mathtt{fun}(\mathtt{Col}(n))$$



source: xkcd.com/710

$\mathtt{fun}(3)$	$\mathtt{fun}(5)$	$\mathtt{fun}(8)$	$\mathtt{fun}(4)$	$\mathtt{fun}(2)$	$\mathtt{fun}(1)$
$\mathbf{t}(\mathbf{t}(\vec{0}))$	$\mathbf{t}(\mathbf{f}(\mathbf{t}(\vec{0})))$	$\mathbf{t}(\mathbf{f}(\mathbf{f}(\mathbf{f}(\vec{0}))))$	$\mathbf{t}(\mathbf{f}(\mathbf{f}(\vec{0})))$	$\mathbf{t}(\mathbf{f}(\vec{0}))$	$\mathbf{t}(\vec{0})$

Arithmetic operations: Collatz

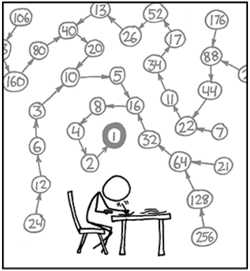
Resolving foundational algorithm questions

$$\mathrm{Col}(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ (3n+1)/2 & \text{if } n \text{ is odd} \end{cases}$$

$\texttt{while}(n > 1)\; n = \mathrm{Col}(n);$ terminates?

Find a non-negative function $\mathrm{fun}(n)$ s.t.

$$\forall n > 1 : \mathrm{fun}(n) > \mathrm{fun}(\mathrm{Col}(n))$$



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

source: xkcd.com/710

$\mathrm{fun}(3)$	$\mathrm{fun}(5)$	$\mathrm{fun}(8)$	$\mathrm{fun}(4)$	$\mathrm{fun}(2)$	$\mathrm{fun}(1)$
$\mathbf{t}(\mathbf{t}(\vec{0}))$	$\mathbf{t}(\mathbf{f}(\mathbf{t}(\vec{0})))$	$\mathbf{t}(\mathbf{f}(\mathbf{f}(\mathbf{f}(\vec{0}))))$	$\mathbf{t}(\mathbf{f}(\mathbf{f}(\vec{0})))$	$\mathbf{t}(\mathbf{f}(\vec{0}))$	$\mathbf{t}(\vec{0})$
$\binom{5}{1}$	$\binom{4}{1}$	$\binom{3}{1}$	$\binom{2}{1}$	$\binom{1}{1}$	$\binom{0}{1}$

using $\mathbf{t}(\vec{x}) = \begin{pmatrix} 1 & 5 \\ 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ and $\mathbf{f}(\vec{x}) = \begin{pmatrix} 1 & 3 \\ 0 & 0 \end{pmatrix} \vec{x} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

Arithmetic Operations: Collatz as Rewriting System