Applications for Automated Reasoning

Marijn J.H. Heule

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f19/

Automated Reasoning and Satisfiability, September 5, 2019

Automated Reasoning Has Many Applications

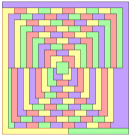
formal verification

security

bioinformatics

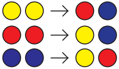
planning and scheduling

train safety



automated theorem proving

exploit generation



term rewriting termination

encode → **SAT/SMT solver** → **decode**

Automated Reasoning Has Many Applications

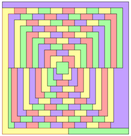
formal verification

security

bioinformatics

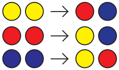
planning and scheduling

train safety



automated theorem proving

exploit generation



term rewriting termination

encode

SAT/SMT solver

decode

Overview

Applications:

- Equivalence checking
  - Hardware and software optimization
- Bounded model checking
  - Hardware and software verification
- Graph problems and symmetry breaking
  - Ramsey numbers, unavoidable subgraphs
- Arithmetic operations
  - Factorization, term rewriting

Equivalence Checking

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();
```

$\Downarrow$

```
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();
```

optimized C code

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

$\Downarrow$

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

$\Rightarrow$

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

$\Uparrow$

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(); }
```

$\Uparrow$

optimized C code

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

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

optimized C code :=

`if` a `then` f
`else if` b `then` g
`else` h

2. compile code into Conjunctive Normal Form

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

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

2. compile code into Conjunctive Normal Form

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

3. check equivalence of Boolean formulae

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

Equivalence checking encoding (2)

compile(**original C code**):

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

Equivalence checking encoding (2)

compile(**original C code**):

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

compile(**optimized C code**):

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

Equivalence checking encoding (2)

compile(**original C code**):

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

compile(**optimized C code**):

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

$$
(a \vee b \vee h) \vee ((\overline{a} \wedge \overline{b}) \vee ((a \vee g) \wedge (\overline{a} \vee f))
$$

$$
\Updownarrow
$$

$$
(\overline{a} \vee f) \wedge (a \vee ((\overline{b} \vee g) \wedge (b \vee h))
$$

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?

or equivalently:

Is the Boolean formula

$$\mathit{compile}(\textbf{original C code}) \not\Leftrightarrow \mathit{compile}(\textbf{optimized C code})$$

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?

or equivalently:

Is the Boolean formula

$$\mathit{compile}(\textbf{original C code}) \not\Leftrightarrow \mathit{compile}(\textbf{optimized C code})$$

satisfiable?

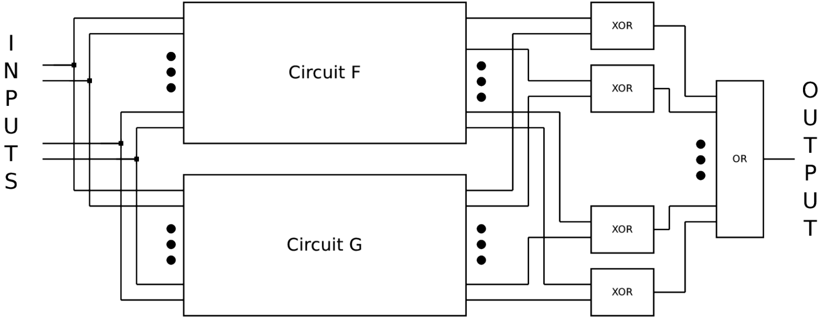
Such an assignment would provide a counterexample

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.



Bounded Model Checking

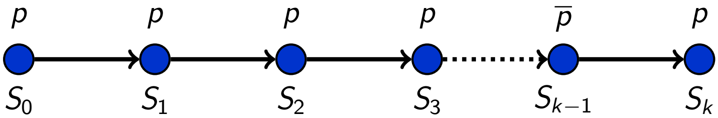
Bounded Model Checking (BMC)

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

Bounded Model Checking (BMC)

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

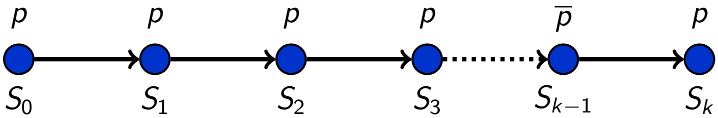
Is there a state reachable in k steps, which satisfies $\overline{p}$?



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)

The reachable states in k steps are captured by:

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

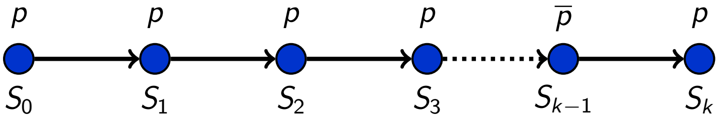
The property p fails in one of the k steps by:

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

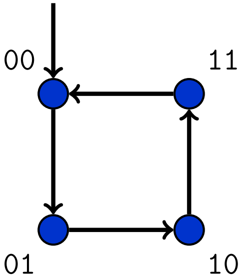
BMC Encoding (2)

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

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



Bounded Model Checking Example: Two-bit counter

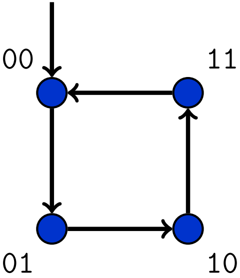


Initial state I: $l_0 = 0, r_0 = 0$

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

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

Bounded Model Checking Example: Two-bit counter



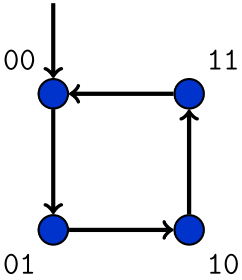
Initial state I: $l_0 = 0, r_0 = 0$

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

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

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

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

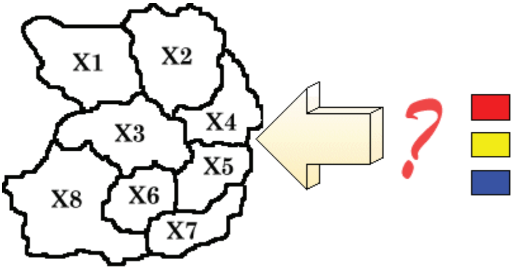
$$\mathcal{F}(2) = (\bar{l}_0 \wedge \bar{r}_0) \wedge \left(\begin{array}{c} l_1 = l_0 \oplus r_0 \wedge r_1 = \bar{r}_0 \wedge \\ l_2 = l_1 \oplus r_1 \wedge r_2 = \bar{r}_1 \end{array} \right) \wedge \left(\begin{array}{c} (l_0 \wedge r_0) \vee \\ (l_1 \wedge r_1) \vee \\ (l_2 \wedge r_2) \end{array}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

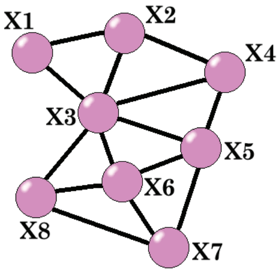
For $k = 2$, $\mathcal{F}(k)$ is unsatisfiable; for $k = 3$ it is satisfiable

Graphs and Symmetries

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
$x_{v,i}$	$i \in \{1, \dots, c\}$ $v \in \{1, \dots, \lvert V \rvert\}$	node v has color i

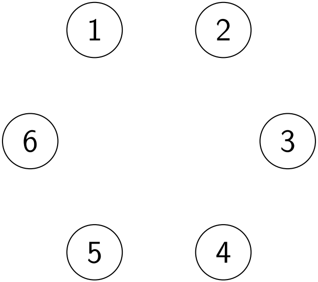
Clauses	Range	Meaning
$(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,c})$	$v \in \{1, \dots, \lvert V \rvert\}$	v is colored
$(\overline{x}_{v,s} \vee \overline{x}_{v,t})$	$s \in \{1, \dots, c-1\}$ $t \in \{s+1, \dots, c\}$	v has at most one color
$(\overline{x}_{v,i} \vee \overline{x}_{w,i})$	$(v, w) \in E$	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?

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



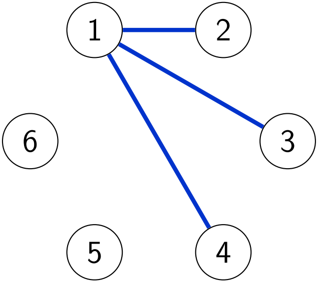
SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

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?

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



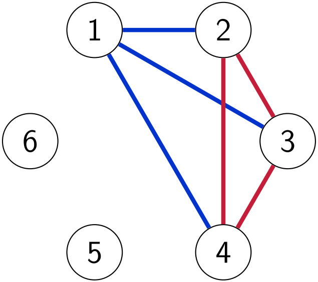
SAT solvers can determine that $R(4) = 18$ in 1 second using symmetry breaking; w/o symmetry breaking it requires weeks.

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?

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



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:

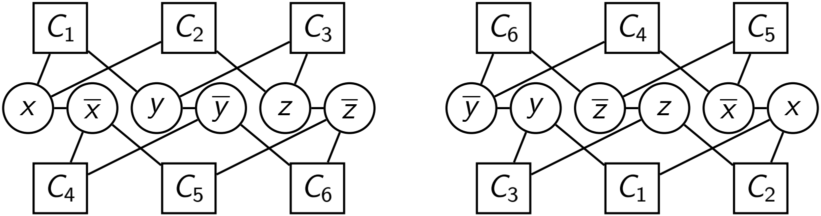
$$F := \overbrace{(x \vee y)}^{C_1} \wedge \overbrace{(x \vee z)}^{C_2} \wedge \overbrace{(y \vee z)}^{C_3} \wedge \overbrace{(\overline{x} \vee \overline{y})}^{C_4} \wedge \overbrace{(\overline{x} \vee \overline{z})}^{C_5} \wedge \overbrace{(\overline{y} \vee \overline{z})}^{C_6}$$

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:

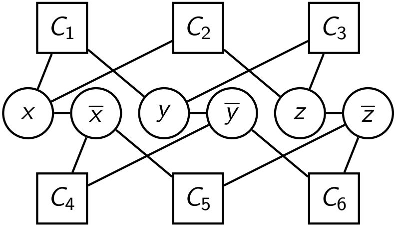
$$F := \overbrace{(x \vee y)}^{C_1} \wedge \overbrace{(x \vee z)}^{C_2} \wedge \overbrace{(y \vee z)}^{C_3} \wedge \overbrace{(\overline{x} \vee \overline{y})}^{C_4} \wedge \overbrace{(\overline{x} \vee \overline{z})}^{C_5} \wedge \overbrace{(\overline{y} \vee \overline{z})}^{C_6}$$

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.

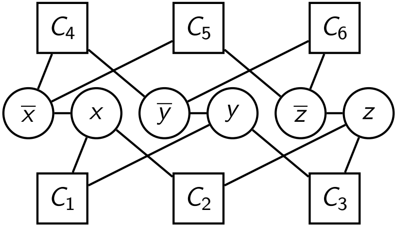


Symmetry: $(x,y,z)(\overline{y},\overline{z},\overline{x})$ is an edge-preserving bijection

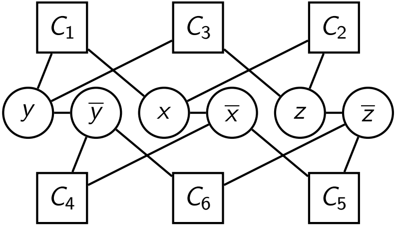
Three Symmetries of the Example Formula



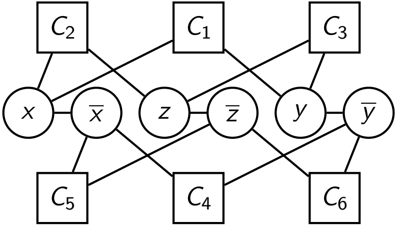
identity symmetry



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



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



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

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 $\bar{x}_i$ onto $\bar{p}_i$ with $i \in \{1, \ldots, n\}$.

Given a CNF formula F. Let τ be a satisfying truth assignment for F and σ a symmetry for F, then $\sigma(\tau)$ 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: $x_1, \ldots, x_n \leq p_1, \ldots, p_n$.

$$
\begin{aligned}
&(\bar{x}_1 \vee p_1) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee p_2) \wedge (p_1 \vee \bar{x}_2 \vee p_2) \wedge \\
&(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee p_3) \wedge (\bar{x}_1 \vee p_2 \vee \bar{x}_3 \vee p_3) \wedge \\
&(p_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee p_3) \wedge (p_1 \vee p_2 \vee \bar{x}_3 \vee p_3) \wedge \ldots
\end{aligned}
$$

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`.

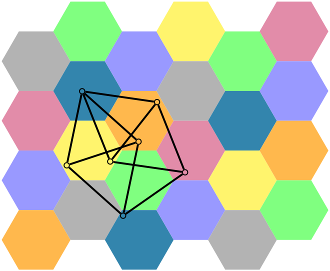
The symmetries can broken by adding symmetry-breaking predicates to the given CNF.

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?

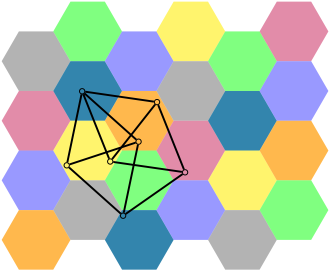
- The Moser Spindle graph shows the lower bound of 4
- A colored tiling of the plane shows the upper bound of 7
- Lower bound of 5 [DeGrey '18] based on a 1581-vertex graph



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?

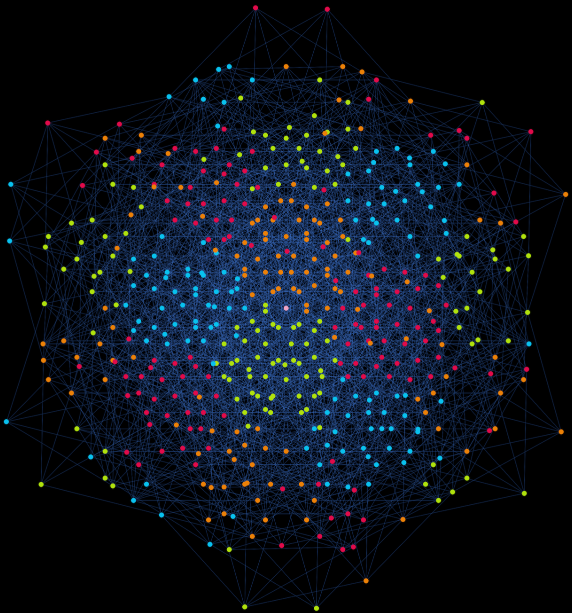
- The Moser Spindle graph shows the lower bound of 4
- A colored tiling of the plane shows the upper bound of 7
- Lower bound of 5 [DeGrey '18] based on a 1581-vertex graph



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]



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

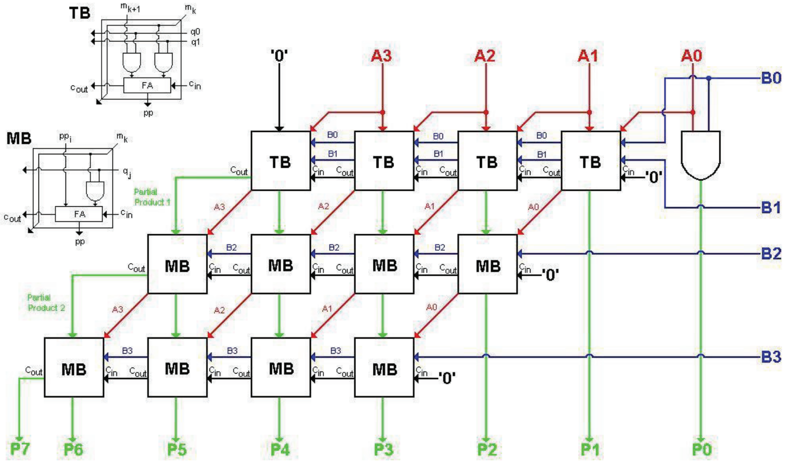
How to encode arithmetic operations into SAT?

Efficient encoding using electronic circuits

Applications:

- factorization (not competitive)
- term rewriting

4x4 Multiplier circuit



Multiplier encoding

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

Multiplier encoding

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

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

Multiplier encoding

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

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

3. Parity out p_{out} of variables p_{in}, $m_{i,j}$ and c_{in}
$$\begin{array}{ll}
(p_{out} \vee \overline{p}_{in} \vee \overline{m}_{i,j} \vee \overline{c}_{in}) \wedge & (p_{out} \vee p_{in} \vee m_{i,j} \vee \overline{c}_{in}) \wedge \\
(\overline{p}_{out} \vee p_{in} \vee \overline{m}_{i,j} \vee \overline{c}_{in}) \wedge & (p_{out} \vee p_{in} \vee \overline{m}_{i,j} \vee c_{in}) \wedge \\
(\overline{p}_{out} \vee \overline{p}_{in} \vee m_{i,j} \vee \overline{c}_{in}) \wedge & (p_{out} \vee \overline{p}_{in} \vee m_{i,j} \vee c_{in}) \wedge \\
(\overline{p}_{out} \vee \overline{p}_{in} \vee \overline{m}_{i,j} \vee c_{in}) \wedge & (\overline{p}_{out} \vee p_{in} \vee m_{i,j} \vee c_{in})
\end{array}$$

Arithmetic operations: Is 27 prime?

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

Arithmetic operations: Is 27 prime?

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

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

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

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

Arithmetic operations: Is 29 prime?

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

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

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:

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

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 $\mathbf{N}^4$ to $\mathbf{N}^4$
- 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 $x_1 \in \mathbf{N}$, 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

Collatz Conjecture

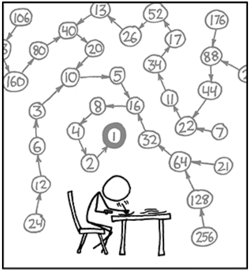
Resolving foundational algorithm questions

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

Does $\texttt{while}(n > 1)\ n = Col(n);$ terminate?

Find a non-negative function $fun(n)$ s.t.

$$\forall n > 1 : fun(n) > fun(Col(n))$$



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

Collatz Conjecture

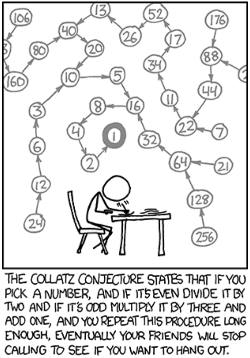
Resolving foundational algorithm questions

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

Does `while(n > 1) n = Col(n);` terminate?

Find a non-negative function $fun(n)$ s.t.

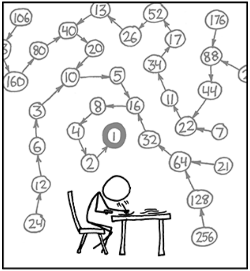
$$\forall n > 1 : fun(n) > fun(Col(n))$$



source: xkcd.com/710

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

Collatz Conjecture



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

Resolving foundational algorithm questions

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

Does $\texttt{while}(n > 1)\ n = Col(n);$ terminate?

Find a non-negative function $fun(n)$ s.t.

$$\forall n > 1 : fun(n) > fun(Col(n))$$

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

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

The Collatz Conjecture as Rewriting System

Consider the following functions:

- Binary system: $f(x) = 2x$, $t(x) = 2x + 1$
- Ternary system: $p(x) = 3x$, $q(x) = 3x + 1$, $r(x) = 3x + 2$
- Start and end symbols: $c(x) = 1$, $d(x) = x$

$$\begin{array}{llll}
D_1: \; fd \to_R d & F_1: \; fp \to_R pf & T_1: \; tp \to_R qt & C_1: \; cp \to_R ct \\
D_2: \; td \to_R rd & F_2: \; fq \to_R pt & T_2: \; tq \to_R rf & C_2: \; cq \to_R cff \\
 & F_3: \; fr \to_R qf & T_3: \; tr \to_R rt & C_3: \; cr \to_R cft
\end{array}$$

Interpretation using the functions above:

$$D_1 \; : \; 2x \to x$$

$$D_2 \; : \; 2x + 1 \to 3x + 2 \quad (= (3(2x + 1) + 1)/2)$$

$$F_1 \; : \; 6x \to 6x$$

$$T_3 \; : \; 6x + 5 \to 6x + 5$$

Collatz Rewriting Example

$$
\begin{array}{lll}
D_1: & fd \to_R d \\
D_2: & td \to_R rd
\end{array}
\qquad
\begin{array}{ll}
F_1: & fp \to_R pf \\
F_2: & fq \to_R pt \\
F_3: & fr \to_R qf
\end{array}
\qquad
\begin{array}{ll}
T_1: & tp \to_R qt \\
T_2: & tq \to_R rf \\
T_3: & tr \to_R rt
\end{array}
\qquad
\begin{array}{ll}
C_1: & cp \to_R ct \\
C_2: & cq \to_R cff \\
C_3: & cr \to_R cft
\end{array}
$$

$$
\begin{array}{ccccccccccccccccc}
c\underline{td} & \to & \underline{cr}d & \to & cf\underline{td} & \to & c\underline{fr}d & \to & \underline{cq}fd & \to & cff\underline{fd} & \to & cf\underline{fd} & \to & c\underline{fd} & \to & cd \\
 & D_2 & & C_3 & & D_2 & & F_3 & & C_2 & & D_1 & & D_1 & & D_1 & \\
3 & \to & 5 & \to & 5 & \to & 8 & \to & 8 & \to & 8 & \to & 4 & \to & 2 & \to & 1
\end{array}
$$

Collatz Rewriting Example

$$
\begin{array}{llll}
 & F_1: \; fp \to_R pf & T_1: \; tp \to_R qt & C_1: \; cp \to_R ct \\
D_1: \; fd \to_R d & & & \\
 & F_2: \; fq \to_R pt & T_2: \; tq \to_R rf & C_2: \; cq \to_R cff \\
D_2: \; td \to_R rd & & & \\
 & F_3: \; fr \to_R qf & T_3: \; tr \to_R rt & C_3: \; cr \to_R cft
\end{array}
$$

$$
\begin{array}{ccccccccccccccccc}
c\underline{td} & \to & \underline{cr}d & \to & cf\underline{td} & \to & c\underline{fr}d & \to & \underline{cq}fd & \to & cff\underline{fd} & \to & cf\underline{fd} & \to & c\underline{fd} & \to & cd \\
 & D_2 & & C_3 & & D_2 & & F_3 & & C_2 & & D_1 & & D_1 & & D_1 & \\
3 & \to & 5 & \to & 5 & \to & 8 & \to & 8 & \to & 8 & \to & 4 & \to & 2 & \to & 1
\end{array}
$$

Can we prove termination of the Collatz rewriting system?

Collatz Rewriting Example

$$\begin{array}{lllll} & F_1: fp \to_R pf & T_1: tp \to_R qt & C_1: cp \to_R ct \\ D_1: fd \to_R d & & & \\ & F_2: fq \to_R pt & T_2: tq \to_R rf & C_2: cq \to_R cff \\ D_2: td \to_R rd & & & \\ & F_3: fr \to_R qf & T_3: tr \to_R rt & C_3: cr \to_R cft \end{array}$$

$$\begin{array}{ccccccccccccccccc} c\underline{td} & \to & \underline{cr}d & \to & cf\underline{td} & \to & c\underline{fr}d & \to & \underline{cq}fd & \to & cff\underline{fd} & \to & cf\underline{fd} & \to & c\underline{fd} & \to & cd \\ & D_2 & & C_3 & & D_2 & & F_3 & & C_2 & & D_1 & & D_1 & & D_1 & \\ 3 & \to & 5 & \to & 5 & \to & 8 & \to & 8 & \to & 8 & \to & 4 & \to & 2 & \to & 1 \end{array}$$

Can we prove termination of the Collatz rewriting system?

The full system is still too hard, but subsystems (removing one of the rules) are doable (although not with existing tools).

Applications for Automated Reasoning

Marijn J.H. Heule

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f19/

Automated Reasoning and Satisfiability, September 5, 2019