Encoding Applications into SAT

Marijn J.H. Heule
Warren A. Hunt Jr.

The University of Texas at Austin

Dress Code as Satisability Problem

Propositional logic:

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

Three conditions / clauses:

- clearly one should not wear a **tie** without a **shirt** $\neg \text{tie} \vee \text{shirt}$
- not wearing a **tie** nor a **shirt** is impolite $\text{tie} \vee \text{shirt}$
- wearing a **tie** and a **shirt** is overkill $\neg(\text{tie} \wedge \text{shirt}) \equiv \neg \text{tie} \vee \neg \text{shirt}$

Is the formula $(\neg \text{tie} \vee \text{shirt}) \wedge (\text{tie} \vee \text{shirt}) \wedge (\neg \text{tie} \vee \neg \text{shirt})$ satisable?

Overview

Encoding common constraints

Applications:

- Equivalence checking
  - Hardware and software optimization
- Bounded model checking
  - Hardware and software verification
- Arithmetic operations
  - Factorization, term rewriting
- Graph coloring
  - Sudoku, timetabling

Common Constraints

AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtLeastOne}\,(x_1, \ldots, x_n)$$

into SAT?

Hint: This is easy...

AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtLeastOne}\,(x_1, \ldots, x_n)$$

into SAT?

Hint: This is easy...

$$(x_1 \vee x_2 \vee \cdots \vee x_n)$$

AtMostOne (1)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT?

AtMostOne (1)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires $n(n-1)/2$ binary clauses:

$$\bigwedge_{1 \le i < j \le n} (\neg x_i \vee \neg x_j)$$

AtMostOne (1)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires $n(n-1)/2$ binary clauses:

$$\bigwedge_{1 \leq i < j \leq n} (\neg x_i \vee \neg x_j)$$

Is it possible to use fewer clauses?

AtMostOne (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\textsc{AtMostOne}\ (x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

AtMostOne (2)

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\text{AtMostOne}\,(x_1, \ldots, x_n)$$

into SAT using a linear number of binary clauses?

By splitting the constraint using additional variables. Apply the direct encoding if $n \leq 4$ otherwise replace $\text{AtMostOne}\,(x_1, \ldots, x_n)$ by

$$\text{AtMostOne}\,(x_1, x_2, x_3, y) \wedge \text{AtMostOne}\,(\neg y, x_4, \ldots, x_n)$$

resulting in $3n - 6$ clauses and $(n - 3)/2$ new variables

Exclusive OR

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}\,(x_1, \ldots, x_n)$$

into SAT?

Exclusive OR

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\mathrm{XOR}\,(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even } \#\neg} ((\neg)x_1 \vee (\neg)x_2 \vee \cdots \vee (\neg)x_n)$$

Exclusive OR

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

$$\text{XOR}\,(x_1, \ldots, x_n)$$

into SAT?

The direct encoding requires 2^{n-1} clauses of length n:

$$\bigwedge_{\text{even}\ \#\neg} ((\neg)x_1 \vee (\neg)x_2 \vee \cdots \vee (\neg)x_n)$$

Make it compact: $\text{XOR}\,(x_1, x_2, y) \wedge \text{XOR}\,(\bar{y}, x_3, \ldots, x_n)$

Applications

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();
```

⇓

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

⇒

```
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(); }
```

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(); }
```

How to check that these two versions are equivalent?

Equivalence checking encoding (1)

1. represent procedures as independent Boolean variables

original C code :=

`if` $\neg a \wedge \neg b$ `then` h
`else if` $\neg a$ `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 independent Boolean variables

original C code :=

`if` $\neg a \wedge \neg b$ `then` h
`else if` $\neg a$ `then` g
`else` f

optimized C code :=

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

2. compile if-then-else into Conjunctive Normal Form

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

Equivalence checking encoding (1)

1. represent procedures as independent Boolean variables

original C code $:=$
`if` $\neg a \wedge \neg b$ `then` h
`else if` $\neg a$ `then` g
`else` f

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

2. compile if-then-else into Conjunctive Normal Form

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

3. check equivalence of Boolean formulae
$compile($**original C code**$) \Leftrightarrow compile($**optimized C code**$)$

Equivalence checking encoding (2)

compile(**original C code**):

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

Equivalence checking encoding (2)

compile(**original C code**):

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

compile(**optimized C code**):

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

Equivalence checking encoding (2)

compile(**original C code**):

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

compile(**optimized C code**):

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

$$(a \lor b \lor h) \lor ((\neg a \land \neg b) \lor ((a \lor g) \land (\neg a \lor f)) \Leftrightarrow (\neg a \lor f) \land (a \lor ((\neg b \lor g) \land (b \lor h))$$

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

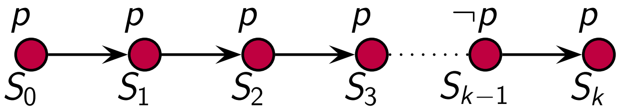
Bounded Model Checking (BMC)

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

Bounded Model Checking (BMC)

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

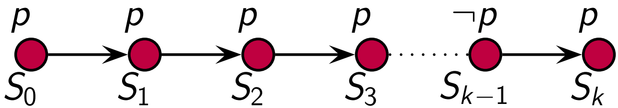
Is there a state reachable in k cycles, which satisfies $\neg p$?



Bounded Model Checking (BMC)

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

Is there a state reachable in k cycles, which satisfies $\neg p$?



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:

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

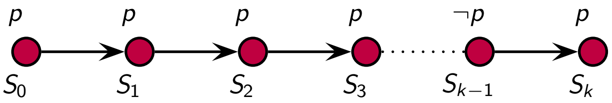
The property p fails in one of the k steps by:

$$\neg P(S_0) \vee \neg P(S_1) \vee \cdots \vee \neg P(S_k)$$

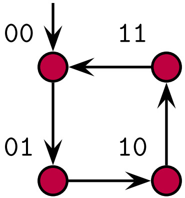
BMC Encoding (2)

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

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



Bounded model checking Example



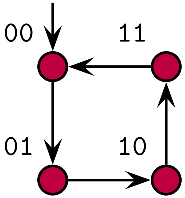
Two bit counter

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

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

Property P: $\neg l_i \vee \neg r_i$

Bounded model checking Example



Two bit counter

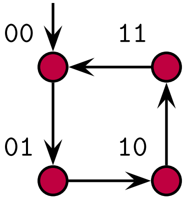
Initial state I: $l_0 = 0, r_0 = 0$

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

Property P: $\neg l_i \vee \neg r_i$

$$\mathcal{F}(2) = (\neg l_0 \wedge \neg r_0) \wedge \begin{pmatrix} l_1 = l_0 \oplus r_0 \wedge r_1 = \neg r_0 \ \wedge \\ l_2 = l_1 \oplus r_1 \wedge r_2 = \neg r_1 \end{pmatrix} \wedge \begin{pmatrix} (\neg l_0 \vee \neg r_0) \ \wedge \\ (\neg l_1 \vee \neg r_1) \ \wedge \\ (\neg l_2 \vee \neg r_2) \end{pmatrix}$$

Bounded model checking Example



Two bit counter

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

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

Property P: $\neg l_i \vee \neg r_i$

$$\mathcal{F}(2) = (\neg l_0 \wedge \neg r_0) \wedge \begin{pmatrix} l_1 = l_0 \oplus r_0 \wedge r_1 = \neg r_0 \ \wedge \\ l_2 = l_1 \oplus r_1 \wedge r_2 = \neg r_1 \end{pmatrix} \wedge \begin{pmatrix} (\neg l_0 \vee \neg r_0) \ \wedge \\ (\neg l_1 \vee \neg r_1) \ \wedge \\ (\neg l_2 \vee \neg r_2) \end{pmatrix}$$

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

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

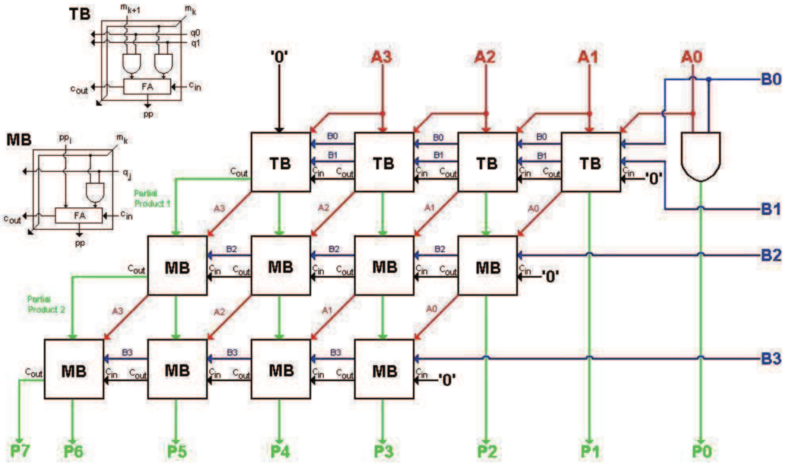
How to encode arithmetic operations into SAT?

Efficient encoding using electronic circuits

Applications:

- factorization (not competitive)
- term rewriting

4x4 Multiplier circuit



Multiplier encoding

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

$(m_{i,j} \lor \neg x_i \lor \neg y_j) \land (\neg m_{i,j} \lor x_i) \land (\neg m_{i,j} \lor y_j)$

Multiplier encoding

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

$(m_{i,j} \vee \neg x_i \vee \neg y_j) \wedge (\neg m_{i,j} \vee x_i) \wedge (\neg m_{i,j} \vee y_j)$

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

$(c_{out} \vee \neg p_{in} \vee \neg m_{i,j}) \wedge (c_{out} \vee \neg p_{in} \vee \neg c_{in}) \wedge (c_{out} \vee \neg m_{i,j} \vee \neg c_{in}) \wedge$
$(\neg c_{out} \vee p_{in} \vee m_{i,j}) \wedge (\neg c_{out} \vee p_{in} \vee c_{in}) \wedge (\neg c_{out} \vee m_{i,j} \vee c_{in})$

Multiplier encoding

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

$(m_{i,j} \vee \neg x_i \vee \neg y_j) \wedge (\neg m_{i,j} \vee x_i) \wedge (\neg m_{i,j} \vee y_j)$

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

$(c_{out} \vee \neg p_{in} \vee \neg m_{i,j}) \wedge (c_{out} \vee \neg p_{in} \vee \neg c_{in}) \wedge (c_{out} \vee \neg m_{i,j} \vee \neg c_{in}) \wedge$
$(\neg c_{out} \vee p_{in} \vee m_{i,j}) \wedge (\neg c_{out} \vee p_{in} \vee c_{in}) \wedge (\neg c_{out} \vee m_{i,j} \vee c_{in})$

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

$(p_{out} \vee \neg p_{in} \vee \neg m_{i,j} \vee \neg c_{in}) \wedge$
$(\neg p_{out} \vee p_{in} \vee \neg m_{i,j} \vee \neg c_{in}) \wedge$
$(\neg p_{out} \vee \neg p_{in} \vee m_{i,j} \vee \neg c_{in}) \wedge$
$(\neg p_{out} \vee \neg p_{in} \vee \neg m_{i,j} \vee c_{in}) \wedge$

$(p_{out} \vee p_{in} \vee m_{i,j} \vee \neg c_{in}) \wedge$
$(p_{out} \vee p_{in} \vee \neg m_{i,j} \vee c_{in}) \wedge$
$(p_{out} \vee \neg p_{in} \vee m_{i,j} \vee c_{in}) \wedge$
$(\neg p_{out} \vee p_{in} \vee m_{i,j} \vee c_{in})$

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?

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

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

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?

			x_3	x_2	x_1	x_0	
			$x_3 y_0$	$x_2 y_0$	$x_1 y_0$	$x_0 y_0$	y_0
		$x_3 y_1$	$x_2 y_1$	$x_1 y_1$	$x_0 y_1$		y_1
	$x_3 y_2$	$x_2 y_2$	$x_1 y_2$	$x_0 y_2$			y_2
$x_3 y_3$	$x_2 y_3$	$x_1 y_3$	$x_0 y_3$				y_3
0	0	1	1	1	0	1	

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 $\mathbf{N}^4$ 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 terminates

Arithmetic operations: Term rewriting linear functions

The linear functions:

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

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

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

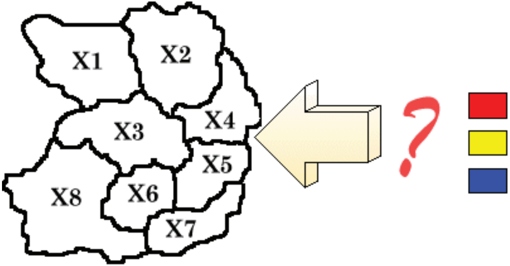
Checking decrease properties using linear algebra

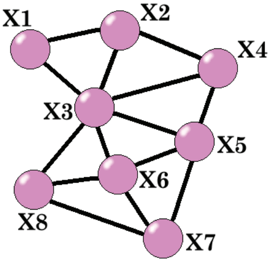
Combinatorial problems

- Encoding is critical when dealing with hard combinatorial problems
- Problems are (relatively) small but very hard
- The most compact representation is not necessarily the best performing
- Mostly based on graph coloring encoding

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

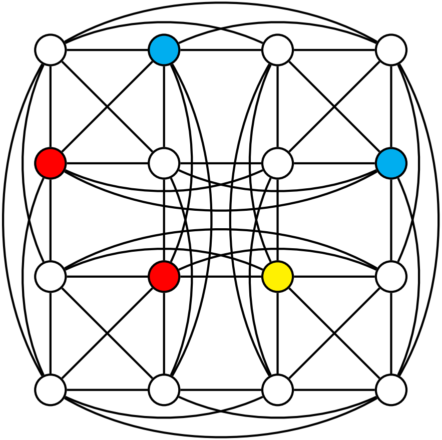
Clauses	Range	Meaning
$(x_{v,1} \vee x_{v,2} \vee \cdots \vee x_{v,c})$	$v \in \{1, \ldots, \lvert V \rvert\}$	v is colored
$(\neg x_{v,s} \vee \neg x_{v,t})$	$s \in \{1, \ldots, c-1\}$ $t \in \{s+1, \ldots, c\}$	v has at most one color
$(\neg x_{v,i} \vee \neg x_{w,i})$	$(v, w) \in E$	v and w have a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

Sudoku

1								6
		6		2		7		
7	8	9	4	5		1		3
			8		7			4
				3				
	9				4	2		1
3	1	2	9	7			4	
	4			1	2		7	8
9		8						

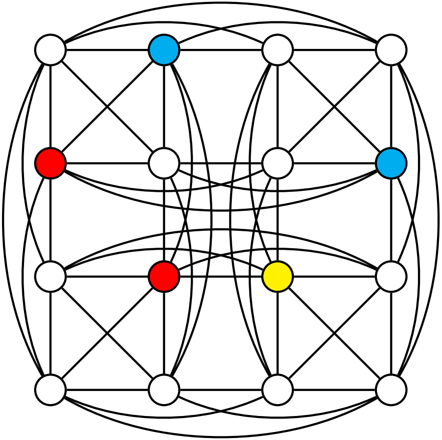
4x4 Sudoku clauses

	2		
1			2
	1	4	



4x4 Sudoku clauses

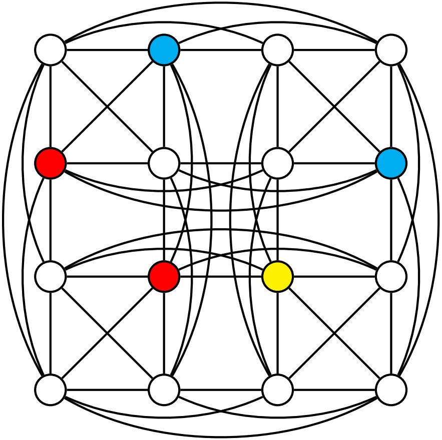
	2		
1			2
	1	4	



$$(x_{v,1} \lor x_{v,2} \lor x_{v,3} \lor x_{v,4})$$

4x4 Sudoku clauses

	2		
1			2
	1	4	



$$(x_{v,1} \vee x_{v,2} \vee x_{v,3} \vee x_{v,4})$$

$$(\neg x_{v,i} \vee \neg x_{w,i})$$

Timetables (1)

Suitable for SAT solving

Timetables (2)

Not suitable for SAT solving

Encoding Applications into SAT

Marijn J.H. Heule
Warren A. Hunt Jr.

The University of Texas at Austin