Computational Logic and Satisfiability

IN4077

Applications

Marijn J.H. Heule

September 9, 2008

Overview

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

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();
```

```
 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 ¬a ∧ ¬b then h
else if ¬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 :=	**optimized C code** :=
`if` $\neg a \wedge \neg b$ `then` h	`if` a `then` f
`else if` $\neg a$ `then` g	`else if` b `then` g
`else` f	`else` h

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

   compile(`if` x `then` y `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 :=	**optimized C code** :=
`if` $\neg a \wedge \neg b$ `then` h	`if` a `then` f
`else if` $\neg a$ `then` g	`else if` b `then` g
`else` f	`else` h

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

   compile(`if` x `then` y `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**):

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

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**):

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

compile(**optimized C code**):

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

$$(a \vee b \vee h) \vee ((\neg a \wedge \neg b) \vee ((a \vee g) \wedge (\neg a \vee f)) \Leftrightarrow (\neg a \vee f) \wedge (a \vee ((\neg b \vee g) \wedge (b \vee h))$$

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

compile(**original C code**) $\Leftrightarrow\!\!\!\!\!/\;\;$ *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

compile(**original C code**) $\Leftrightarrow\!\!\!\!\!/\;\;$ *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)

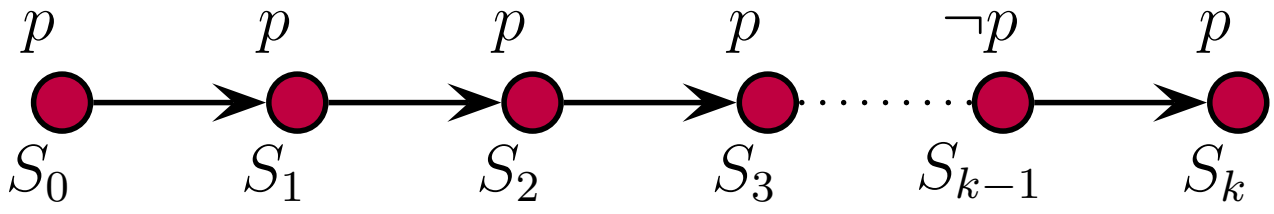
Bounded Model Checking (BMC)

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

Bounded Model Checking (BMC)

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

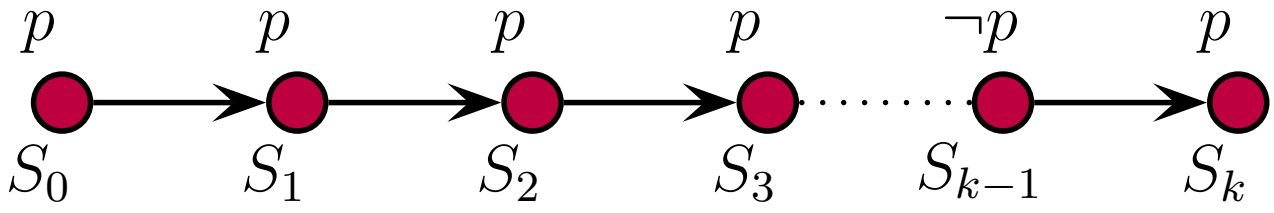
Is there a state reachable in k cycles, which satisfies $\neg p$?



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 Edmund M. Clarke

BMC Encoding (1)

The reachable states in k steps are captured by:

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

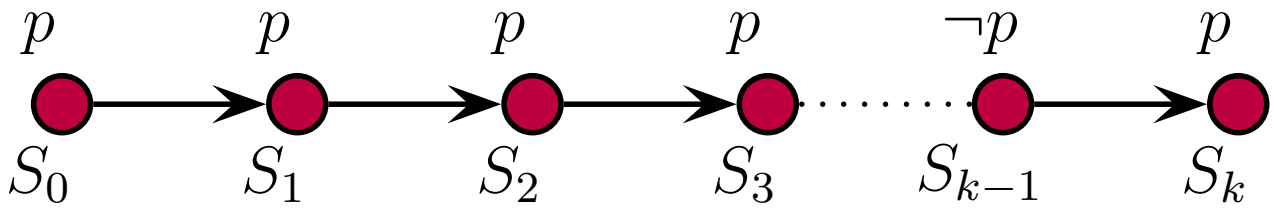
The property p fails in one of the k steps by:

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

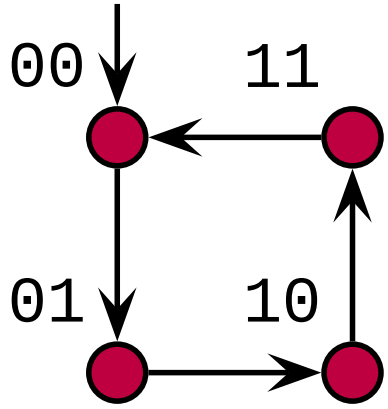
BMC Encoding (2)

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: $l_0 = 0, r_0 = 0$

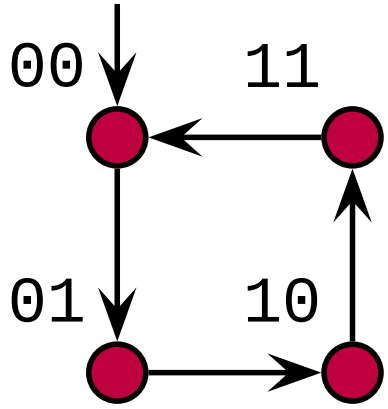
Transition T: $l_{i+1} = l_i \oplus r_i,$

$r_{i+1} = \neg r_i$

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

Bounded model checking Example

Two bit counter



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

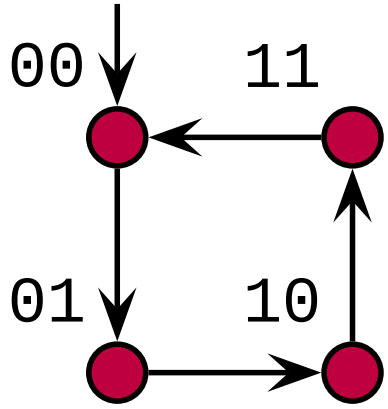
Transition T: $l_{i+1} = l_i \oplus r_i,$ $r_{i+1} = \neg r_i$

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

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

Bounded model checking Example

Two bit counter



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

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

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

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

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

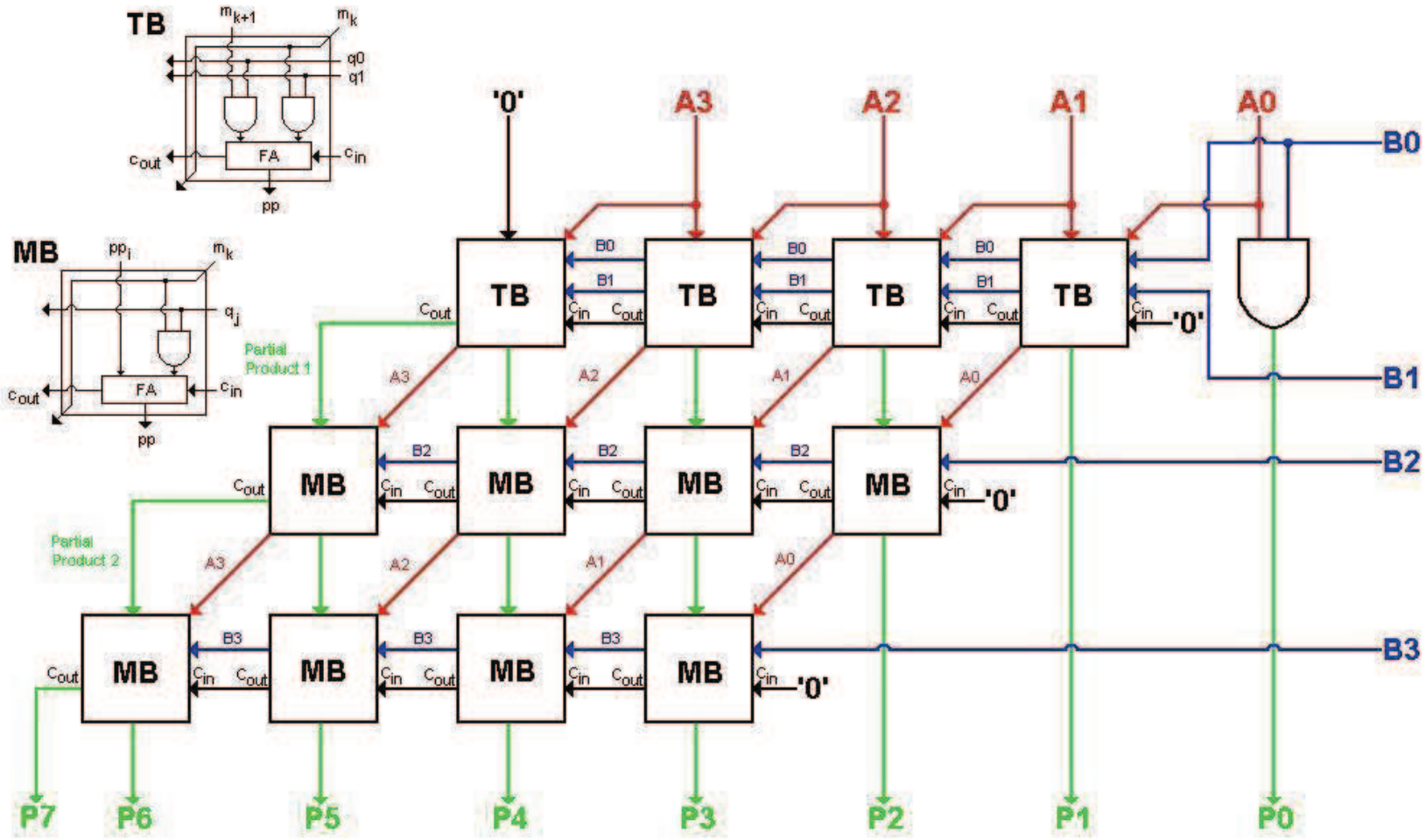
How to encode arithmetic operations into SAT?

Efficient encoding using electronic circuits

Applications:

- factorization (not competitive)
- term rewriting

4x4 Multiplier circuit



Multiplier encoding

1. Multiplication $m_{i,j} = x_i \times y_j$

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

Multiplier encoding

1. Multiplication $m_{i,j} = x_i \times y_j$

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

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

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

Multiplier encoding

1. Multiplication $m_{i,j} = x_i \times y_j$

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

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

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

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

$$\begin{aligned}
&(p_{out} \vee \neg p_{in} \vee \neg m_{i,j} \vee \neg c_{in}) \wedge \quad (p_{out} \vee p_{in} \vee m_{i,j} \vee \neg c_{in}) \wedge \\
&(\neg p_{out} \vee p_{in} \vee \neg m_{i,j} \vee \neg c_{in}) \wedge \quad (p_{out} \vee p_{in} \vee \neg m_{i,j} \vee c_{in}) \wedge \\
&(\neg p_{out} \vee \neg p_{in} \vee m_{i,j} \vee \neg c_{in}) \wedge \quad (p_{out} \vee \neg p_{in} \vee m_{i,j} \vee c_{in}) \wedge \\
&(\neg p_{out} \vee \neg p_{in} \vee \neg m_{i,j} \vee c_{in}) \wedge \quad (\neg p_{out} \vee p_{in} \vee m_{i,j} \vee c_{in})
\end{aligned}$$

Arithmetic operations: Is 27 prime?

$$
\begin{array}{cccccccc}
 & & & x_3 & x_2 & x_1 & x_0 & \\
 & & x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 & & y_0 \\
 & x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 & & & y_1 \\
x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 & & & & y_2 \\
x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 1 &
\end{array}
$$

Arithmetic operations: Is 27 prime?

$$
\begin{array}{cccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 \\
 & & & x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 & y_0 \\
 & & x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 & & y_1 \\
 & x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 & & & y_2 \\
x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 1 &
\end{array}
$$

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

Arithmetic operations: Is 27 prime?

$$
\begin{array}{cccccccc}
 & & & x_3 & x_2 & x_1 & x_0 & \\
 & & x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 & & y_0 \\
 & x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 & & & y_1 \\
x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 & & & & y_2 \\
x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 0 & 1 & 1 &
\end{array}
$$

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

Arithmetic operations: Is 29 prime?

$$
\begin{array}{cccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 \\
 & & & x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 & y_0 \\
 & & x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 & & y_1 \\
 & x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 & & & y_2 \\
x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 & 1 &
\end{array}
$$

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

Arithmetic operations: Is 29 prime?

$$
\begin{array}{cccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 \\
 & & & x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 & y_0 \\
 & & x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 & & y_1 \\
 & x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 & & & y_2 \\
x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 & & & & y_3 \\
\hline
0 & 0 & 1 & 1 & 1 & 0 & 1 &
\end{array}
$$

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:

$aa \rightarrow bc$

$bb \rightarrow ca$

$cc \rightarrow ab$

Arithmetic operations: Term rewriting

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

Example set of rules:

$aa \rightarrow bc$

$bb \rightarrow ca$

$cc \rightarrow ab$

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

Example solved by Hofbauer, Waldmann (2006)

Combinatorial problems

Encoding is critical when dealing with hard combinatorial problems

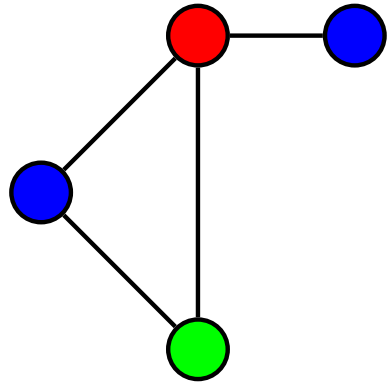
Problems are (relatively) small but very hard

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 of each $(v, w) \in E$, v and w are colored differently.



Problem: Many symmetries!!!

Graph coloring encoding

Variables	Range	Meaning
$x_{i,j}$	$i \in \{1, \ldots, c\}$ $j \in \{1, \ldots, \lvert V \rvert\}$	node j has color i

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

$$\forall i, n : x_{i,1,n} \vee x_{i,2,n} \vee x_{i,3,n} \vee x_{i,4,n}$$

4x4 Sudoku clauses

$$\forall i, n : x_{i,1,n} \vee x_{i,2,n} \vee x_{i,3,n} \vee x_{i,4,n}$$

	2		
1			2
	1	4	

$$\left.\begin{aligned}
& \neg x_{1,1,n} \vee \neg x_{1,2,n} \\
& \neg x_{1,1,n} \vee \neg x_{1,3,n} \\
& \neg x_{1,1,n} \vee \neg x_{1,4,n} \\
\forall n : & \neg x_{1,1,n} \vee \neg x_{2,1,n} \\
& \neg x_{1,1,n} \vee \neg x_{3,1,n} \\
& \neg x_{1,1,n} \vee \neg x_{4,1,n} \\
& \neg x_{1,1,n} \vee \neg x_{2,2,n}
\end{aligned}\right\}$$

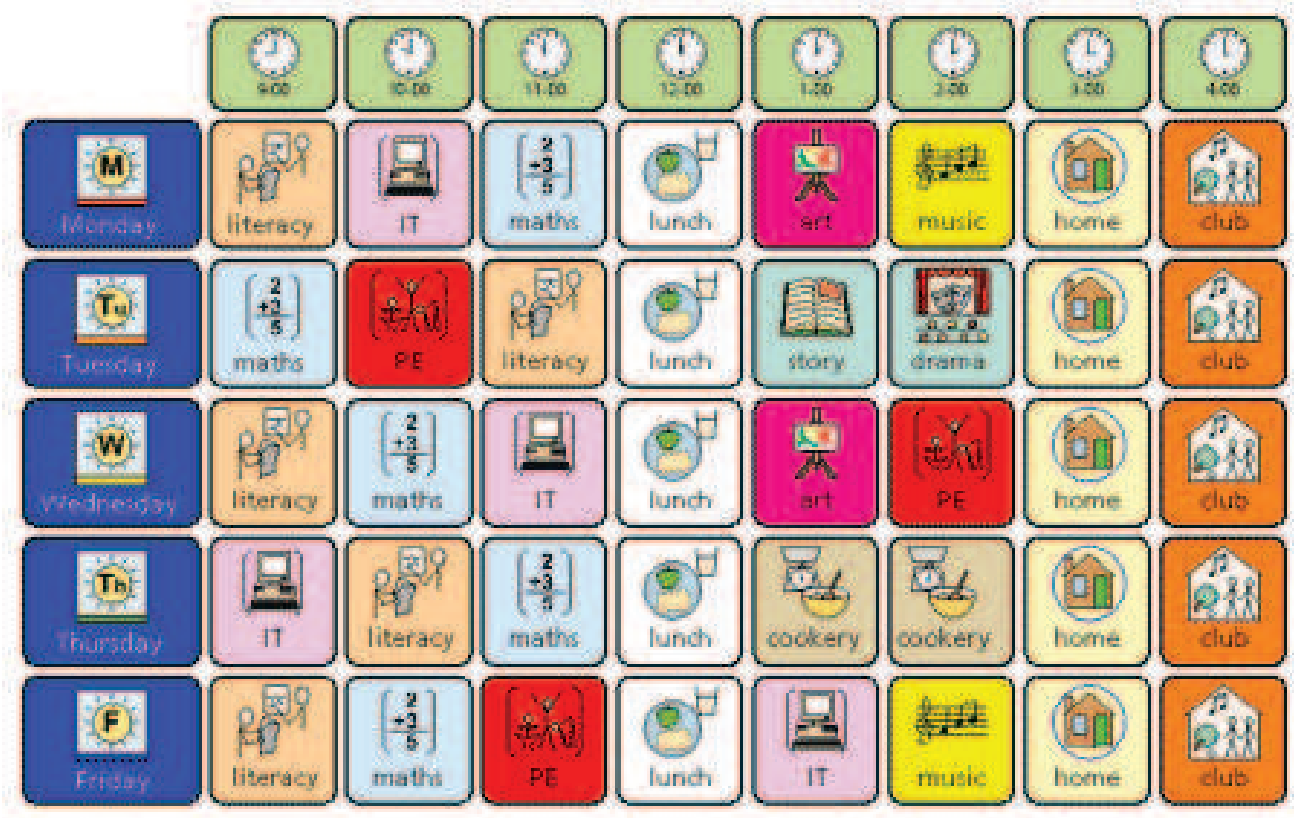
4x4 Sudoku clauses

$$\forall i, n : x_{i,1,n} \vee x_{i,2,n} \vee x_{i,3,n} \vee x_{i,4,n}$$

	2		
1			2
	1	4	

$$\forall n : \left.\begin{array}{l} \neg x_{1,1,n} \vee \neg x_{1,2,n} \\ \neg x_{1,1,n} \vee \neg x_{1,3,n} \\ \neg x_{1,1,n} \vee \neg x_{1,4,n} \\ \neg x_{1,1,n} \vee \neg x_{2,1,n} \\ \neg x_{1,1,n} \vee \neg x_{3,1,n} \\ \neg x_{1,1,n} \vee \neg x_{4,1,n} \\ \neg x_{1,1,n} \vee \neg x_{2,2,n} \end{array}\right\} \times 16$$

Timetables (1)



Suitable for SAT solving

Timetables (2)

Not suitable for SAT solving

Computational Logic and Satisfiability

IN4077

Applications

Marijn J.H. Heule

September 9, 2008