Encoding Applications into SAT

Marijn J.H. Heule
marijn@cs.utexas.edu
Capitalize on the performance of SAT solvers

formal verification  graph theory  bioinformatics  train safety

timetabling  number theory  cryptography  rewriting termination

encode SAT solver decode
Overview

Encoding common constraints

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
AtLeastOne

Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode $\text{AtLeastOne} (x_1, \ldots, x_n)$ into SAT?

**Hint:** This is easy...
AtLeastOne

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

\[ \text{AtLeastOne} (x_1, \ldots, x_n) \]

into SAT?

**Hint:** This is easy...

\[ (x_1 \lor x_2 \lor \cdots \lor x_n) \]
AtMostOne (1)

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

\[
\text{AtMostOne} \ (x_1, \ldots, x_n)
\]

into SAT?
AtMostOne (1)

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

\[ \text{AtMostOne} (x_1, \ldots, x_n) \]

into SAT?

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

\[ \bigwedge_{1 \leq i < j \leq n} (\lnot x_i \lor \lnot x_j) \]
AtMostOne (1)

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

AtMostOne ($x_1, \ldots, x_n$)

into SAT?

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

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

Is it possible to use fewer clauses?
Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode \texttt{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

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

resulting in $3n - 6$ clauses and $(n - 3)/2$ new variables
Given a set of Boolean variables $x_1, \ldots, x_n$, how to encode

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

into SAT?
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 $n$:

$$\bigwedge_{\text{even} \neq \neg} ((\neg)x_1 \lor (\neg)x_2 \lor \cdots \lor (\neg)x_n)$$
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 $n$:

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

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

⇓

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

⇒
if(a) f();
else
{
if(!b) h();
else g();
}
Equivalence checking example

**original C code**

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

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

Are these two code fragments equivalent?
Equivalence checking example

**original C code**

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

⇓

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

⇒

```c
if(a) f();
else {
    if(!b) h();
    else g();
}
```
Equivalence checking example

original C code

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

⇓

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

⇒

optimized C code

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

⇑

```c
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 \( \neg a \wedge \neg b \) then \( h \)
else if \( \neg a \) then \( g \)
else \( f \)

optimized C code :=

if \( a \) then \( f \)
else if \( b \) then \( g \)
else \( h \)
1. represent procedures as Boolean variables

**original C code** :=

\[
\text{if } \neg a \land \neg b \text{ then } h \\
\text{else if } \neg a \text{ then } g \\
\text{else } f
\]

**optimized C code** :=

\[
\text{if } a \text{ then } f \\
\text{else if } b \text{ then } g \\
\text{else } h
\]

2. compile code into Conjunctive Normal Form

\[
\text{compile}(\text{if } x \text{ then } y \text{ else } z) \equiv (\neg x \lor y) \land (x \lor z)
\]
Equivalence checking encoding (1)

1. represent procedures as Boolean variables

original C code :=
if \( \neg a \land \neg b \) 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 code into Conjunctive Normal Form

\[
\text{compile}(\text{if } x \text{ then } y \text{ else } z) \equiv (\neg x \lor y) \land (x \lor z)
\]

3. check equivalence of Boolean formulae

\[
\text{compile(} \text{original C code} \text{)} \Leftrightarrow \text{compile(} \text{optimized C code} \text{)}
\]
**Equivalence checking encoding (2)**

*compile*(original C code):

```c
if ¬a ∧ ¬b then h else if ¬a then g else f
≡ (¬(¬a ∧ ¬b) ∨ h) ∨ ((¬a ∧ ¬b) ∨ (if ¬a then g else f))
≡ (a ∨ b ∨ h) ∨ ((¬a ∧ ¬b) ∨ ((a ∨ g) ∧ (¬a ∨ f)))
```
Equivalence checking encoding (2)

**compile***(original C code):*

\[
\text{if } \neg a \land \neg b \text{ then } h \text{ else if } \neg a \text{ then } g \text{ else } f \\
\equiv \neg((\neg a \land \neg b) \lor h) \lor ((\neg a \land \neg b) \lor (\text{if } \neg a \text{ then } g \text{ else } f)) \\
\equiv (a \lor b \lor h) \lor ((\neg a \land \neg b) \lor ((a \lor g) \land (\neg a \lor f))
\]

**compile**(optimized C code):

\[
\text{if } a \text{ then } f \text{ else if } b \text{ then } g \text{ else } h \\
\equiv (\neg a \lor f) \land (a \lor (\text{if } b \text{ then } g \text{ else } h)) \\
\equiv (\neg a \lor f) \land (a \lor ((\neg b \lor g) \land (b \lor h)))
\]
Equivalence checking encoding (2)

**compile**(original C code):

\[
\begin{align*}
\text{if } & \neg a \land \neg b \text{ then } h \text{ else if } \neg a \text{ then } g \text{ else } f \\
& \equiv \neg(\neg a \land \neg b) \lor h \lor ((\neg a \land \neg b) \lor (\text{if } \neg a \text{ then } g \text{ else } f)) \\
& \equiv (a \lor b \lor h) \lor ((\neg a \land \neg b) \lor ((a \lor g) \land (\neg a \lor f))
\end{align*}
\]

**compile**(optimized C code):

\[
\begin{align*}
\text{if } & a \text{ then } f \text{ else if } b \text{ then } g \text{ else } h \\
& \equiv (\neg a \lor f) \land (a \lor (\text{if } b \text{ then } g \text{ else } h)) \\
& \equiv (\neg a \lor f) \land (a \lor ((\neg b \lor g) \land (b \lor h))
\end{align*}
\]

\[
\begin{align*}
(a \lor b \lor h) \lor ((\neg a \land \neg b) \lor ((a \lor g) \land (\neg a \lor f)) \\
\uparrow \\
(\neg a \lor f) \land (a \lor ((\neg b \lor g) \land (b \lor h))
\end{align*}
\]
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 $\text{compile} (\text{original C code}) \equiv \text{compile} (\text{optimized C code})$ satisfiable?

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

\[
\text{compile(\text{original C code})} \iff \text{compile(\text{optimized C code})}
\]

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*

\[
\text{compile(original C code)} \Leftrightarrow \text{compile(optimized C code)}
\]

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

Given two circuits, a miter is a 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. $\text{signal}_a = \text{signal}_b$)
Bounded Model Checking (BMC)

Given a property $p$: (e.g. $\text{signal}_a = \text{signal}_b$)

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

$p$

$\rightarrow$

$p$

$\rightarrow$

$p$

$\rightarrow$

$p$

$\rightarrow$

$\neg p$

$\rightarrow$

$p$

$S_0$

$S_1$

$S_2$

$S_3$

$S_{k-1}$

$S_k$
Bounded Model Checking (BMC)

Given a property $p$: (e.g. $\text{signal}_a = \text{signal}_b$)

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

Turing award 2007 for Model Checking
Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis
The reachable states in $k$ steps are captured by:

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

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

$$\neg P(S_0) \lor \neg P(S_1) \lor \cdots \lor \neg P(S_k)$$
The safety property $p$ is valid up to step $k$ if and only if $\mathcal{F}(k)$ is unsatisfiable:

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

$\mathcal{F}(2) = (\neg l_0 \land \neg r_0) \land \left( l_1 = l_0 \oplus r_0 \land r_1 = \neg r_0 \land l_2 = l_1 \oplus r_1 \land r_2 = \neg r_1 \right) \land \left( (\neg l_0 \lor \neg r_0) \land (\neg l_1 \lor \neg r_1) \land (\neg l_2 \lor \neg r_2) \right)$
Bounded Model Checking Example: Two-bit counter

Initial state $I$: $l_0 = 0, r_0 = 0$
Transition $T$: $l_{i+1} = l_i \oplus r_i, \quad r_{i+1} = \neg r_i$
Property $P$: $\neg l_i \lor \neg r_i$

\[
F(2) = (\neg l_0 \land \neg r_0) \land \left( l_1 = l_0 \oplus r_0 \land r_1 = \neg r_0 \land l_2 = l_1 \oplus r_1 \land r_2 = \neg r_1 \right) \land \left( \neg l_0 \lor \neg r_0 \right) \land \left( \neg l_1 \lor \neg r_1 \right) \land \left( \neg l_2 \lor \neg r_2 \right)
\]

For $k = 2$, $F(k)$ is unsatisfiable; for $k = 3$ it is satisfiable
Graphs and Symmetries
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.
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.
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:

\[ F := (\neg x \lor y) \land (x \lor z) \land (y \lor z) \land (\neg x \lor \neg y) \land (\neg x \lor \neg z) \land (\neg y \lor \neg z) \]
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:

$$F := (x \lor y) \land (x \lor z) \land (y \lor z) \land (\neg x \lor \neg y) \land (\neg x \lor \neg z) \land (\neg y \lor \neg z)$$

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: $(x, y, z)(\neg y, \neg z, \neg x)$ is an edge-preserving bijection
Three Symmetries of the Example Formula

identity symmetry

$C_1$ $C_2$ $C_3$

$x$ $\neg x$ $y$ $\neg y$ $z$ $\neg z$

$C_4$ $C_5$ $C_6$

$(x, y, z, C_1, C_2, C_3, C_4, C_5, C_6)$

$(\neg x, \neg y, \neg z, C_4, C_5, C_6, C_1, C_2, C_3)$

$C_1$ $C_3$ $C_2$

$y$ $\neg y$ $x$ $\neg x$ $z$ $\neg z$

$C_4$ $C_6$ $C_5$

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

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

$C_2$ $C_1$ $C_3$

$x$ $\neg x$ $z$ $\neg z$ $y$ $\neg y$

$C_5$ $C_4$ $C_6$

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

$(z, y, C_2, C_5, C_1, C_4)$
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 \( \tau \) be a satisfying truth assignment for \( F \) and \( \sigma \) 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: \( x_1, \ldots, x_n \leq p_1, \ldots, p_n \).

\[
(\bar{x}_1 \lor p_1) \land (\bar{x}_1 \lor \bar{x}_2 \lor p_2) \land (p_1 \lor \bar{x}_2 \lor p_2) \land \\
(\bar{x}_1 \lor \bar{x}_2 \lor \bar{x}_3 \lor p_3) \land (\bar{x}_1 \lor p_2 \lor \bar{x}_3 \lor p_3) \land \\
(p_1 \lor \bar{x}_2 \lor \bar{x}_3 \lor p_3) \land (p_1 \lor p_2 \lor \bar{x}_3 \lor p_3) \land \ldots
\]
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.
Encoding 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 \( m_{i,j} = x_i \times y_j = \text{AND} (x_i, y_j) \)
   
   \( (m_{i,j} \lor \neg x_i \lor \neg y_j) \land (\neg m_{i,j} \lor x_i) \land (\neg m_{i,j} \lor y_j) \)
Multiplier encoding

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

2. Carry out \( c_{out} = 1 \) if and only if \( p_{in} + m_{i,j} + c_{in} > 1 \)
\((c_{out} \lor \neg p_{in} \lor \neg m_{i,j}) \land (c_{out} \lor \neg p_{in} \lor \neg c_{in}) \land (c_{out} \lor \neg m_{i,j} \lor \neg c_{in}) \land \)
\((\neg c_{out} \lor p_{in} \lor m_{i,j}) \land (\neg c_{out} \lor p_{in} \lor c_{in}) \land (\neg c_{out} \lor m_{i,j} \lor c_{in})\)
Multiplier encoding

1. Multiplication \( m_{i,j} = x_i \times y_j = \text{AND}(x_i, y_j) \)
\[
(m_{i,j} \lor \neg x_i \lor \neg y_j) \land (\neg m_{i,j} \lor x_i) \land (\neg m_{i,j} \lor y_j)
\]

2. Carry out \( c_{out} = 1 \) if and only if \( p_{in} + m_{i,j} + c_{in} > 1 \)
\[
(c_{out} \lor \neg p_{in} \lor \neg m_{i,j}) \land (c_{out} \lor \neg p_{in} \lor \neg c_{in}) \land (c_{out} \lor \neg m_{i,j} \lor \neg c_{in}) \land

(\neg c_{out} \lor p_{in} \lor m_{i,j}) \land (\neg c_{out} \lor p_{in} \lor c_{in}) \land (\neg 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 \neg p_{in} \lor \neg m_{i,j} \lor \neg c_{in}) \land (p_{out} \lor p_{in} \lor m_{i,j} \lor \neg c_{in}) \land

(\neg p_{out} \lor p_{in} \lor \neg m_{i,j} \lor \neg c_{in}) \land (p_{out} \lor p_{in} \lor \neg m_{i,j} \lor c_{in}) \land

(\neg p_{out} \lor \neg p_{in} \lor m_{i,j} \lor \neg c_{in}) \land (p_{out} \lor \neg p_{in} \lor m_{i,j} \lor c_{in}) \land

(\neg p_{out} \lor \neg p_{in} \lor \neg m_{i,j} \lor c_{in}) \land (\neg p_{out} \lor p_{in} \lor m_{i,j} \lor c_{in})
\]
Arithmetic operations: Is 27 prime?

\[
\begin{array}{cccccc}
\ x_3 & x_2 & x_1 & x_0 \\
\ x_3y_0 & x_2y_0 & x_1y_0 & x_0y_0 & y_0 \\
\ x_3y_1 & x_2y_1 & x_1y_1 & x_0y_1 & y_1 \\
\ x_3y_2 & x_2y_2 & x_1y_2 & x_0y_2 & y_2 \\
\ x_3y_3 & x_2y_3 & x_1y_3 & x_0y_3 & y_3 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
\ x_3 & x_2 & x_1 & x_0 & y_0 & y_1 & y_2 & y_3 \\
0 & 0 & \ 1 & 1 & 0 & 1 & 1
\end{array}
\]
Arithmetic operations: Is 27 prime?

\[
\begin{array}{cccc}
  x_3 & x_2 & x_1 & x_0 \\
  x_3 y_0 & x_2 y_0 & x_1 y_0 & x_0 y_0 & y_0 \\
  x_3 y_1 & x_2 y_1 & x_1 y_1 & x_0 y_1 & y_1 \\
  x_3 y_2 & x_2 y_2 & x_1 y_2 & x_0 y_2 & y_2 \\
  x_3 y_3 & x_2 y_3 & x_1 y_3 & x_0 y_3 & y_3 \\
  0 & 0 & 1 & 1 & 0 & 1 & 1 & 1
\end{array}
\]

Prime: \((x_1 \lor x_2 \lor x_3) \land (y_1 \lor y_2 \lor y_3)\)
Arithmetic operations: Is 27 prime?

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_3$</td>
<td>$x_2$</td>
<td>$x_1$</td>
<td>$x_0$</td>
<td></td>
</tr>
<tr>
<td>$x_3y_0$</td>
<td>$x_2y_0$</td>
<td>$x_1y_0$</td>
<td>$x_0y_0$</td>
<td></td>
</tr>
<tr>
<td>$y_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3y_1$</td>
<td>$x_2y_1$</td>
<td>$x_1y_1$</td>
<td>$x_0y_1$</td>
<td></td>
</tr>
<tr>
<td>$y_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3y_2$</td>
<td>$x_2y_2$</td>
<td>$x_1y_2$</td>
<td>$x_0y_2$</td>
<td></td>
</tr>
<tr>
<td>$y_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3y_3$</td>
<td>$x_2y_3$</td>
<td>$x_1y_3$</td>
<td>$x_0y_3$</td>
<td></td>
</tr>
<tr>
<td>$y_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Prime: $(x_1 \lor x_2 \lor x_3) \land (y_1 \lor y_2 \lor y_3)$
Arithmetic operations: Is 29 prime?

<table>
<thead>
<tr>
<th>$x_3$</th>
<th>$x_2$</th>
<th>$x_1$</th>
<th>$x_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_3y_0$</td>
<td>$x_2y_0$</td>
<td>$x_1y_0$</td>
<td>$x_0y_0$</td>
</tr>
<tr>
<td>$y_0$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3y_1$</td>
<td>$x_2y_1$</td>
<td>$x_1y_1$</td>
<td>$x_0y_1$</td>
</tr>
<tr>
<td>$y_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3y_2$</td>
<td>$x_2y_2$</td>
<td>$x_1y_2$</td>
<td>$x_0y_2$</td>
</tr>
<tr>
<td>$y_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x_3y_3$</td>
<td>$x_2y_3$</td>
<td>$x_1y_3$</td>
<td>$x_0y_3$</td>
</tr>
<tr>
<td>$y_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Prime: $(x_1 \lor x_2 \lor x_3) \land (y_1 \lor y_2 \lor y_3)$
Arithmetic operations: Is 29 prime?

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:

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

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

Example solved by Hofbauer, Waldmann (2006)
Arithmetic operations: Term rewriting

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

Example set of rules:

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

\[
\begin{align*}
  bbaa & \rightarrow_R bbbc & bacc & \rightarrow_R baab & \rightarrow_R bbcb & \rightarrow_R \\
  accb & \rightarrow_R aabb & aacc & \rightarrow_R abcc & \rightarrow_R abab
\end{align*}
\]
Arithmetic operations: Term rewriting

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

Example set of rules:

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

$bbaa \rightarrow_R bbbc \rightarrow_R bacc \rightarrow_R baab \rightarrow_R bbcb \rightarrow_R$

$accb \rightarrow_R aabb \rightarrow_R aaac \rightarrow_R abcc \rightarrow_R abab$

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
Encoding Applications into SAT

Marijn J.H. Heule
marijn@cs.utexas.edu