Decision Procedures

An Algorithmic Point of View

Decision Procedures for Propositional Logic

D. Kroening O. Strichman

ETH/Technion

Version 1.0, 2007

Part I

Decision Procedures for Propositional Logic

Outline

1. Modeling with Propositional Logic
   - SAT Example: Equivalence Checking if-then-else Chains
   - SAT Example: Circuit Equivalence Checking
2. Formal Definition SAT
3. Conjunctive Normal Form
   - Definition
   - Tseitin Transformation
   - DIMACS CNF

Optimization of if-then-else chains

original C code

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

$\Downarrow$

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

$\Rightarrow$

optimized C code

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

$\Uparrow$

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

1. Represent procedures as *independent* Boolean variables

$$\begin{array}{ll} \mathit{original} := & \qquad \mathit{optimized} := \\ \quad \textbf{if } \neg a \wedge \neg b \textbf{ then } h & \qquad \quad \textbf{if } a \textbf{ then } f \\ \quad \textbf{else if } \neg a \textbf{ then } g & \qquad \quad \textbf{else if } b \textbf{ then } g \\ \quad \textbf{else } f & \qquad \quad \textbf{else } h \end{array}$$

2. Compile if-then-else chains into Boolean formulae

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

3. Check equivalence of Boolean formulae

$$\boxed{\text{compile}(\mathit{original}) \quad \Leftrightarrow \quad \text{compile}(\mathit{optimized})}$$

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

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

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

Reformulate it as a satisfiability (SAT) problem:

> Is there an assignment to a, b, f, g, h,
> which results in different evaluations of *original*
> and *optimized*?

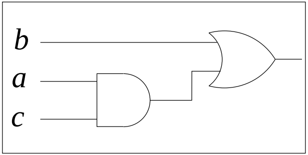
Reformulate it as a satisfiability (SAT) problem:

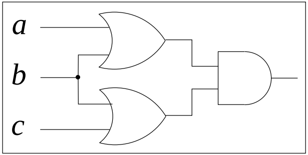
> Is there an assignment to a, b, f, g, h,
> which results in different evaluations of *original* and *optimized*?

or equivalently:

> Is the boolean formula
> $\text{compile}(\mathit{original}) \not\leftrightarrow \text{compile}(\mathit{optimized})$
> satisfiable?

Such an assignment provides an easy to understand counterexample





$b \lor a \land c$ $\quad\quad$ $(a \lor b) \land (b \lor c)$

equivalent?

$$b \lor a \land c \quad \Leftrightarrow \quad (a \lor b) \land (b \lor c)$$

SAT (Satisfiability) the classical NP-complete problem:
Given a propositional formula f over n propositional variables $V = \{x, y, \ldots\}$.

Is there are an assignment $\sigma : V \to \{0, 1\}$ with $\sigma(f) = 1$?

- **SAT belongs to NP**
  There is a *non-deterministic* Touring-machine deciding SAT in polynomial time:
  guess the assignment σ (linear in n), calculate $\sigma(f)$ (linear in $|f|$)
  Note: on a *real* (deterministic) computer this still requires 2^n time
- **SAT is complete for NP** (see complexity / theory class)

Implications for us: general SAT algorithms are probably exponential in time (unless NP $=$ P)

Definition (Conjunctive Normal Form)

A formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

each clause C is a disjunction of literals

$$C = L_1 \vee \ldots \vee L_m$$

and each literal is either a plain variable x or a negated variable $\overline{x}$.

Definition (Conjunctive Normal Form)

A formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

each clause C is a disjunction of literals

$$C = L_1 \vee \ldots \vee L_m$$

and each literal is either a plain variable x or a negated variable $\overline{x}$.

Example $(a \vee b \vee c) \wedge (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee \overline{c})$

CNF for Parity Function is Exponential

b

0	1	0	1
1	0	1	0
0	1	0	1
1	0	1	0

a c

d

$$a \oplus b \oplus c \oplus d$$

- no merging in the Karnaugh map
- all clauses contain all variables
- CNF for parity with n variables has 2^{n-1} clauses

CNF for Parity Function is Exponential

b

	0	1	0	1
a	1	0	1	0
a	0	1	0	1
	1	0	1	0

c

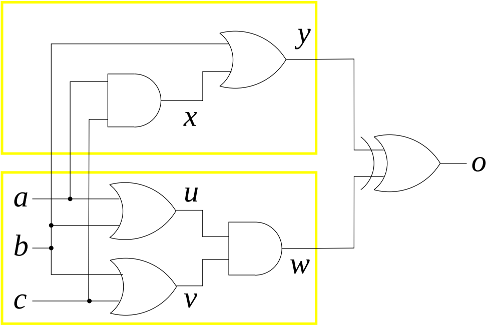
d

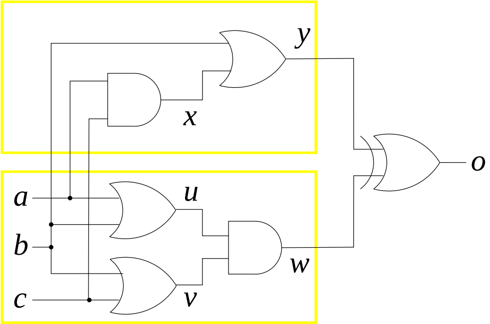
$a \oplus b \oplus c \oplus d$

- no merging in the Karnaugh map
- all clauses contain all variables
- CNF for parity with n variables has 2^{n-1} clauses

Better ideas?

Example of Tseitin Transformation: Circuit to CNF





$$
\begin{array}{rcll}
 & & o & \wedge \\
(x & \leftrightarrow & a \wedge c) & \wedge \\
(y & \leftrightarrow & b \vee x) & \wedge \\
(u & \leftrightarrow & a \vee b) & \wedge \\
(v & \leftrightarrow & b \vee c) & \wedge \\
(w & \leftrightarrow & u \wedge v) & \wedge \\
(o & \leftrightarrow & y \oplus w) &
\end{array}
$$

$$o \wedge (x \rightarrow a) \wedge (x \rightarrow c) \wedge (x \leftarrow a \wedge c) \wedge \ \ldots$$

$$o \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee c) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge \ \ldots$$

Algorithmic Description of Tseitin Transformation

Tseitin Transformation

1. For each non input circuit signal s generate a new variable x_s
2. For each gate produce complete input / output constraints as clauses
3. Collect all constraints in a big conjunction

Algorithmic Description of Tseitin Transformation

Tseitin Transformation

1. For each non input circuit signal s generate a new variable x_s
2. For each gate produce complete input / output constraints as clauses
3. Collect all constraints in a big conjunction

- The transformation is satisfiability equivalent:
  the result is satisfiable iff and only the original formula is satisfiable
- Not equivalent in the classical sense to original formula:
  it has new variables

Tseitin Transformation

1. For each non input circuit signal s generate a new variable x_s
2. For each gate produce complete input / output constraints as clauses
3. Collect all constraints in a big conjunction

- The transformation is satisfiability equivalent: the result is satisfiable iff and only the original formula is satisfiable
- Not equivalent in the classical sense to original formula: it has new variables
- You an get a satisfying assignment for original formula by projecting the satisfying assignment onto the original variables

Tseitin Transformation: Input / Output Constraints

Negation:
$$x \leftrightarrow \overline{y} \Leftrightarrow (x \rightarrow \overline{y}) \wedge (\overline{y} \rightarrow x)$$
$$\Leftrightarrow (\overline{x} \vee \overline{y}) \wedge (y \vee x)$$

Disjunction:
$$x \leftrightarrow (y \vee z) \Leftrightarrow (y \rightarrow x) \wedge (z \rightarrow x) \wedge (x \rightarrow (y \vee z))$$
$$\Leftrightarrow (\overline{y} \vee x) \wedge (\overline{z} \vee x) \wedge (\overline{x} \vee y \vee z)$$

Conjunction:
$$x \leftrightarrow (y \wedge z) \Leftrightarrow (x \rightarrow y) \wedge (x \rightarrow z) \wedge ((y \wedge z) \rightarrow x)$$
$$\Leftrightarrow (\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{(y \wedge z)} \vee x)$$
$$\Leftrightarrow (\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z} \vee x)$$

Equivalence:
$$x \leftrightarrow (y \leftrightarrow z) \Leftrightarrow (x \rightarrow (y \leftrightarrow z)) \wedge ((y \leftrightarrow z) \rightarrow x)$$
$$\Leftrightarrow (x \rightarrow ((y \rightarrow z) \wedge (z \rightarrow y)) \wedge ((y \leftrightarrow z) \rightarrow x)$$
$$\Leftrightarrow (x \rightarrow (y \rightarrow z)) \wedge (x \rightarrow (z \rightarrow y)) \wedge ((y \leftrightarrow z) \rightarrow x)$$
$$\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge ((y \leftrightarrow z) \rightarrow x)$$
$$\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge (((y \wedge z) \vee (\overline{y} \wedge \overline{z})) \rightarrow x)$$
$$\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge ((y \wedge z) \rightarrow x) \wedge ((\overline{y} \wedge \overline{z}) \rightarrow x)$$
$$\Leftrightarrow (\overline{x} \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{z} \vee y) \wedge (\overline{y} \vee \overline{z} \vee x) \wedge (y \vee z \vee x)$$

- Goal is smaller CNF (less variables, less clauses)
- Extract multi argument operands
  (removes variables for intermediate nodes)
- NNF: half of AND, OR node constraints may be removed due to monotonicity
- use *sharing*

DIMACS CNF

- DIMACS CNF format = standard format for CNF
- Used by most SAT solvers
- Plain text file with following structure:

  `p cnf` <# variables> <# clauses>
  <clause> 0
  <clause> 0
  ...
- One or more lines per clause

- Every clause is a list of numbers, separated by spaces
- A clause ends with 0
- Every number $1, 2, \ldots$ corresponds to a variable

→ variable names (e.g., a, b, $\ldots$) have to be mapped to numbers

- A negative number corresponds to negation

→ Let a have number 5. Then $\overline{a}$ is -5.

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee \overline{x_1}) \wedge (x_4 \vee \overline{x_2} \vee \overline{x_1})$$

- 4 variables, 3 clauses
- CNF file:

```
p cnf 4 3
1 2 -3 0
2 -1 0
4 -2 -1 0
```

formula:

$$
\begin{array}{rcl}
 & & o \;\wedge \\
(x & \leftrightarrow & a \wedge c) \;\wedge \\
(y & \leftrightarrow & b \vee x) \;\wedge \\
(u & \leftrightarrow & a \vee b) \;\wedge \\
(v & \leftrightarrow & b \vee c) \;\wedge \\
(w & \leftrightarrow & u \wedge v) \;\wedge \\
(o & \leftrightarrow & y \oplus w)
\end{array}
$$

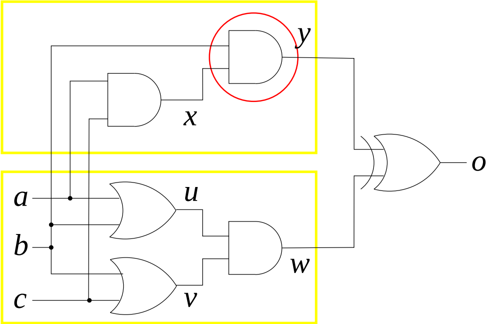
number assignment:

variable	number
o	1
a	2
c	3
x	4
b	5
y	6
u	7
v	8
w	9

Simply in order of occurrence.

formula	clauses	DIMACS
o	o	`1 0`
$x \leftrightarrow a \wedge c$	$a \vee \overline{x}$	`2 -4 0`
	$c \vee \overline{x}$	`3 -4 0`
	$\overline{a} \vee \overline{c} \vee x$	`-2 -3 4 0`
$y \leftrightarrow b \vee x$	$\overline{x} \vee y$	`-4 6 0`
	$\overline{b} \vee y$	`-5 6 0`
	$x \vee b \vee \overline{y}$	`4 5 -6 0`
$u \leftrightarrow a \vee b$	$\overline{a} \vee u$	`-2 7 0`
	$\overline{b} \vee u$	`-5 7 0`
	$a \vee b \vee \overline{u}$	`2 5 -7 0`
$v \leftrightarrow b \vee c$	$\overline{b} \vee v$	`-5 8 0`
	$\overline{c} \vee v$	`-3 8 0`
	$b \vee c \vee \overline{v}$	`5 3 -8 0`
$w \leftrightarrow u \wedge v$	$u \vee \overline{w}$	`7 -9 0`
	$v \vee \overline{w}$	`8 -9 0`
	$\overline{u} \vee \overline{v} \vee w$	`-7 -8 9 0`
$o \leftrightarrow y \oplus w$	$\overline{y} \vee \overline{w} \vee \overline{o}$	`-6 -9 -1 0`
	$y \vee w \vee \overline{o}$	`6 9 -1 0`
	$\overline{y} \vee w \vee o$	`-6 9 1 0`

Let's change the circuit!



$$
\begin{array}{rcl}
 & & o \;\wedge \\
(x & \leftrightarrow & a \wedge c) \;\wedge \\
(y & \leftrightarrow & b \wedge x) \;\wedge \\
(u & \leftrightarrow & a \vee b) \;\wedge \\
(v & \leftrightarrow & b \vee c) \;\wedge \\
(w & \leftrightarrow & u \wedge v) \;\wedge \\
(o & \leftrightarrow & y \oplus w)
\end{array}
$$

Is the CNF satisfiable?

- Output of the SAT solver:

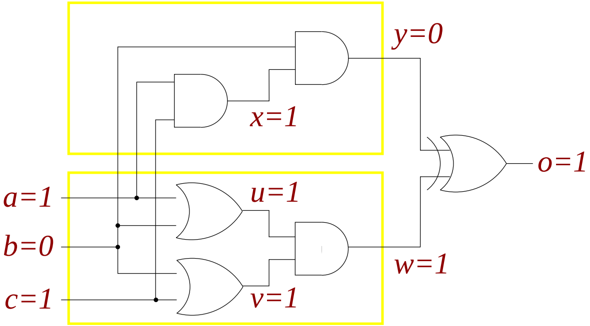
```
SATISFIABLE
1 2 3 4 -5 -6 7 8 9
```

- Values of the variables:

variable	number	value
o	1	1
a	2	1
c	3	1
x	4	1
b	5	0
y	6	0
u	7	1
v	8	1
w	9	1

- Caveat: there is more than one solution

Satisfying assignment mapped to the circuit:



variable	value
o	1
a	1
c	1
x	1
b	0
y	0
u	1
v	1
w	1