Practical SAT Solving
Lecture 2
Carsten Sinz, Tomáš Balyo | April 25, 2016
Lecture Outline

- **Encodings**
  - Arithmetic progressions
  - Graph coloring
  - Finite-domain variables
  - At-most-one constraints / cardinality constraints
  - Integer multiplication / factorization
  - Tseitin encoding for circuits

- **Later in the lecture**
  - Hardware / software verification
  - Product configuration
  - Planning
A wide variety of problems can be encoded as SAT!

- (Finite) arithmetic
- Mathematical / practical combinatorial problems
- Hardware / software verification problems
- Planning problems

Chosen encoding highly influences runtime of SAT solver

- A lot of research on good encodings
- ...but still more an art than a science
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011?
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2, x_5, x_8$ are equally spaced.
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2$, $x_5$, $x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2$, $x_5$, $x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2, x_5, x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
- Encode what’s forbidden: $x_2 x_5 x_8 \neq 111$ is the same as $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$. 
Arithmetic Progressions

Find a binary sequence $x_1, \ldots, x_8$ that has no three equally spaced 0s and no three equally spaced 1s.

- What about 01001011? No, the 1s at $x_2, x_5, x_8$ are equally spaced.
- 6 Solutions:
  00110011, 01011010, 01100110, 10011001, 10100101, 11001100.
- Extending the problem to 9 digits, no solutions remain. How can we show this with a SAT solver?
- Encode what’s forbidden: $x_2 x_5 x_8 \neq 111$ is the same as $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$.
- Writing, e.g., $\overline{258}$ for the clause $(\overline{x_2} \lor \overline{x_5} \lor \overline{x_8})$, we arrive at 32 clauses for the 9 digit sequence:
  123, 234, \ldots, 789, 135, 246, \ldots, 579, 147, 258, 369, 159,
  $\overline{123}, \overline{234}, \ldots, \overline{789}, \overline{135}, \overline{246}, \ldots, \overline{579}, \overline{147}, \overline{258}, \overline{369}, \overline{159}$. 

Introduction Encodings
Carsten Sinz, Tomáš Balyo – SAT Solving
April 25, 2016
4/22
Theorem (van der Waerden)

If \( n \) is sufficiently large, every sequence \( x_1, \ldots, x_n \) of numbers \( 0 \leq x_i < r \) contains a number that occurs at least \( k \) times equally spaced.

- The smallest such number is the van der Waerden number \( W(r, k) \).
- We have seen that \( W(2, 3) = 9 \).
- For larger \( r, k \) the numbers are only partially known.
- E.g., \( W(2, 6) = 1132 \) was shown in 2008 by Kouril and Paul [1] (with the help of a SAT solver!), but \( W(2, 7) \) is yet unknown.
- \( 2^{2r^{2k+9}} \) is an upper bound for \( W(r, k) \) (shown in 2001 by Gowers [2]).
Graph Coloring

- McGregor graph (of order 10): planar, 110 nodes.
- Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner’s column “Mathematical Games”)
Graph Coloring

- McGregor graph (of order 10): planar, 110 nodes.
- Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner’s column “Mathematical Games”)
Graph Coloring

- McGregor graph (of order 10): planar, 110 nodes.
- Claim: Cannot be colored with less than 5 colors. (Scientific American, 1975, Martin Gardner’s column “Mathematical Games”)
Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.
Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- SAT encoding: use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$. 

Definition: Graph coloring

Given an undirected graph \( G = (V, E) \), a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most \( k \) colors is called a \( k \)-coloring. The Graph Coloring Problem asks whether a \( k \)-coloring for \( G \) exists.

- SAT encoding: use \( k \cdot |V| \) Boolean variables \( v_j \) for \( v \in V, 1 \leq j \leq k \), where \( v_j \) is true, if node \( v \) gets color \( j \).
- Clauses?
Graph Coloring: Encoding in SAT

Definition: Graph coloring

Given an undirected graph \( G = (V, E) \), a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most \( k \) colors is called a \( k \)-coloring. The Graph Coloring Problem asks whether a \( k \)-coloring for \( G \) exists.

- SAT encoding: use \( k \cdot |V| \) Boolean variables \( v_j \) for \( v \in V, 1 \leq j \leq k \), where \( v_j \) is true, if node \( v \) gets color \( j \).
- Clauses?

\[
( v_1 \lor \cdots \lor v_k ) \quad \text{for } v \in V \quad \text{("every node gets a color")}
\]
Graph Coloring: Encoding in SAT

Definition: Graph coloring

Given an undirected graph \( G = (V, E) \), a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most \( k \) colors is called a \( k \)-coloring. The Graph Coloring Problem asks whether a \( k \)-coloring for \( G \) exists.

- SAT encoding: use \( k \cdot |V| \) Boolean variables \( v_j \) for \( v \in V, 1 \leq j \leq k \), where \( v_j \) is true, if node \( v \) gets color \( j \).

- Clauses?

\[
(v_1 \lor \cdots \lor v_k) \text{ for } v \in V \quad \text{ (“every node gets a color”)}
\]

\[
(u_j \lor v_j) \text{ for } u \rightarrow v, 1 \leq j \leq k \quad \text{ (“adjacent nodes have diff. colors”)}
\]
**Definition: Graph coloring**

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- **SAT encoding:** use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.

- **Clauses?**
  
  $(v_1 \lor \cdots \lor v_k)$ for $v \in V$ \hspace{1cm} (“every node gets a color”)
  
  $(u_j \lor \bar{v_j})$ for $u \dashv v$, $1 \leq j \leq k$ \hspace{1cm} (“adjacent nodes have diff. colors”)

- **What about multiple colors for a node?**
Graph Coloring: Encoding in SAT

Definition: Graph coloring

Given an undirected graph $G = (V, E)$, a graph coloring (proper vertex coloring) assigns a color to each node, such that all adjacent nodes have a different color. A graph coloring using at most $k$ colors is called a $k$-coloring. The Graph Coloring Problem asks whether a $k$-coloring for $G$ exists.

- **SAT encoding:** use $k \cdot |V|$ Boolean variables $v_j$ for $v \in V$, $1 \leq j \leq k$, where $v_j$ is true, if node $v$ gets color $j$.
- **Clauses?**
  
  $(v_1 \lor \cdots \lor v_k)$ for $v \in V$ \hspace{1cm} (“every node gets a color”)
  
  $(\overline{u_j} \lor v_j)$ for $u \rightleftarrows v$, $1 \leq j \leq k$ \hspace{1cm} (“adjacent nodes have diff. colors”)

- **What about multiple colors for a node?** → At-most-one constraints
Graph Coloring: Example

- \( V = \{ u, v, w, x, y \} \)
- Colors: red (=1), green (=2), blue (=3)
- Clauses:
  - “every node gets a color”
    \[
    (u_1 \lor u_2 \lor u_3) \\
    \vdots \\
    (y_1 \lor y_2 \lor y_3)
    \]
  - “adjacent nodes have different colors”
    \[
    (\overline{u}_1 \lor \overline{v}_1) \land \cdots \land (\overline{u}_3 \lor \overline{v}_3) \\
    \vdots \\
    (\overline{x}_1 \lor \overline{y}_1) \land \cdots \land (\overline{x}_3 \lor \overline{y}_3)
    \]
Graph Coloring: Example

- $V = \{u, v, w, x, y\}$
- Colors: red (=1), green (=2), blue (=3)
- Clauses:
  - "every node gets a color"
    \[(u_1 \lor u_2 \lor u_3) \land \ldots \land (u_3 \lor u_3)\]
  - "adjacent nodes have different colors"
    \[(\overline{u}_1 \lor \overline{v}_1) \land \ldots \land (\overline{u}_3 \lor \overline{v}_3)\]
    \[(\overline{x}_1 \lor \overline{y}_1) \land \ldots \land (\overline{x}_3 \lor \overline{y}_3)\]
Finite-Domain Variables

- Common in combinatorial problems: finite domain variables, e.g.:
  \[ x \in \{ v_1, \ldots, v_n \} \]
- Relationships between them expressed as equality-formulas, e.g.:
  \[ x = v_3 \Rightarrow y \neq v_2. \]
Finite-Domain Variables

- Common in combinatorial problems: finite domain variables, e.g.:
  \[ x \in \{ v_1, \ldots, v_n \} \]
- Relationships between them expressed as equality-formulas, e.g.:
  \[ x = v_3 \Rightarrow y \neq v_2. \]
- Direct encoding / “one-hot-encoding”:
  - Boolean variables \( x_v \): “x takes value v”
  - Must encode that each variable takes exactly one value from its domain (using at-least-one/at-most-one constraints)
  - Encoding of variables’ constraints simple
Finite-Domain Variables

- Common in combinatorial problems: finite domain variables, e.g.:
  \[ x \in \{ v_1, \ldots, v_n \} \]

- Relationships between them expressed as equality-formulas, e.g.:
  \[ x = v_3 \Rightarrow y \neq v_2. \]

- Direct encoding / “one-hot-encoding”:
  - Boolean variables \( x_v \): “\( x \) takes value \( v \)”
  - Must encode that each variable takes exactly one value from its domain (using at-least-one/at-most-one constraints)
  - Encoding of variables’ constraints simple

- Log-encoding / binary encoding:
  - Boolean variables \( b^x_i \) for \( 0 \leq i < \lceil \log_2 n \rceil \)
  - Each value gets assigned a binary number, e.g.
    \[ v_1 \rightarrow 00, \; v_2 \rightarrow 01, \; v_3 \rightarrow 10 \]
  - Inadmissible values must be excluded, e.g.:
    \[ x \in \{ v_1, v_2, v_3 \} \] requires \( (b^x_0 \lor b^x_1) \)
  - Encoding of constraints can become complicated
Comparing Encodings

- Size: number of variables, number of clauses
- Propagation properties
At-Most-One Constraints

**Definition**

\( \text{AtMostOne}(x_1, \ldots, x_n) \) is the constraint that no more than 1 variable / literal out of \( x_1, \ldots, x_n \) is set to True.
At-Most-One Constraints

Definition

AtMostOne\(x_1, \ldots, x_n\) is the constraint that no more than 1 variable / literal out of \(x_1, \ldots, x_n\) is set to True.

- Alternative notations: \(\leq 1 (x_1, \ldots, x_n)\), \(x_1 + \cdots + x_n \leq 1\)
At-Most-One Constraints

Definition

AtMostOne($x_1, \ldots, x_n$) is the constraint that no more than 1 variable / literal out of $x_1, \ldots, x_n$ is set to True.

- Alternative notations: $\leq 1 (x_1, \ldots, x_n)$, $x_1 + \cdots + x_n \leq 1$
- Naive (pairwise) encoding: add clauses $(\overline{x_i} \lor \overline{x_j})$ for $1 \leq i < j \leq n$
  - Results in $\binom{n}{2} = \frac{n \cdot (n-1)}{2}$ clauses
At-Most-One Constraints

Definition

\text{AtMostOne}(x_1, \ldots, x_n) \text{ is the constraint that no more than 1 variable / literal out of } x_1, \ldots, x_n \text{ is set to True.}

- Alternative notations: \( \leq 1 (x_1, \ldots, x_n), \quad x_1 + \cdots + x_n \leq 1 \)
- Naive (pairwise) encoding: add clauses \((\overline{x_i} \lor \overline{x_j})\) for \(1 \leq i < j \leq n\)
  - Results in \(\binom{n}{2} = \frac{n \cdot (n-1)}{2}\) clauses
- Can we do better?
At-Most-One Constraints

Definition

AtMostOne\((x_1, \ldots, x_n)\) is the constraint that no more than 1 variable / literal out of \(x_1, \ldots, x_n\) is set to True.

- Alternative notations: \(\leq 1(x_1, \ldots, x_n)\), \(x_1 + \cdots + x_n \leq 1\)
- Naive (pairwise) encoding: add clauses \((\overline{x_i} \lor \overline{x_j})\) for \(1 \leq i < j \leq n\)
  - Results in \(\binom{n}{2} = \frac{n \cdot (n-1)}{2}\) clauses
- Can we do better? Yes!
At-Most-One Constraints

Definition

AtMostOne\((x_1, \ldots, x_n)\) is the constraint that no more than 1 variable / literal out of \(x_1, \ldots, x_n\) is set to True.

- Alternative notations: \(\leq 1 \ (x_1, \ldots, x_n), \quad x_1 + \cdots + x_n \leq 1\)
- Naive (pairwise) encoding: add clauses \((\overline{x_i} \lor \overline{x_j})\) for \(1 \leq i < j \leq n\)
  - Results in \(\binom{n}{2} = \frac{n \cdot (n-1)}{2}\) clauses
- Can we do better? Yes!
- Encodings and their complexity (in number of clauses) [3]:
  - Pairwise encoding: \(O(n^2)\)
  - Sequential counter: \(O(n)\)
  - Bitwise encoding: \(O(n \log n)\)
Cardinality Constraints: Motivation

Is there a 4-coloring of the McGregor graph that uses one color at most 7 times?
Cardinality Constraints: Motivation

Is there a 4-coloring of the McGregor graph that uses one color at most 7 times?
Definition

\[ \leq k(x_1, \ldots, x_n) \] is the constraint that no more than \( k \) variables / literals out of \( x_1, \ldots, x_n \) are set to True.
Cardinality Constraints

**Definition**

\[ \leq k \left( x_1, \ldots, x_n \right) \] is the constraint that no more than \( k \) variables / literals out of \( x_1, \ldots, x_n \) are set to True.

- Naive encoding: add clauses \( \left( x_{i_1} \lor \cdots \lor x_{i_{k+1}} \right) \) for \( 1 \leq i_1 < \cdots < i_{k+1} \leq n \)

- Results in \( \binom{n}{k} \) clauses
Cardinality Constraints

Definition

\[ \leq k(x_1, \ldots, x_n) \] is the constraint that no more than \( k \) variables / literals out of \( x_1, \ldots, x_n \) are set to True.

- Naive encoding: add clauses \((\overline{x_{i_1}} \lor \cdots \lor \overline{x_{i_k+1}})\) for \( 1 \leq i_1 < \cdots < i_{k+1} \leq n \)
  - Results in \( \binom{n}{k} \) clauses
- Better encodings:
  - Sequential counter: \( \mathcal{O}(n \cdot k) \)
  - Parallel counter: \( \mathcal{O}(n) \)
Cardinality Constr: Sequential Counter

\[
\begin{align*}
\neg x_1 & \lor s_{1,1} \\
\neg s_{1,j} & \quad \text{for } 1 < j \leq k \\
\neg x_i & \lor s_{i,1} \\
\neg s_{i-1,1} & \lor s_{i,1} \\
\neg x_i & \lor \neg s_{i-1,j-1} \lor s_{i,j} \\
\neg s_{i-1,j} & \lor s_{i,j} \\
\neg x_i & \lor \neg s_{i-1,k} \\
\neg x_n & \lor \neg s_{n-1,k} \\
\end{align*}
\]

\[
\begin{align*}
\text{for } 1 < j \leq k & \quad \text{for } 1 < i < n
\end{align*}
\]
Problem

Given a formula $F$ in propositional logic with operations $\land$, $\lor$ and $\neg$, how can it be encoded in CNF?

Example

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

First approach: convert to NNF (Negation Normal Form), then apply distributive law.

Example (cont’d)

$F_{NNF} = (x \land \neg y) \lor z \lor (x \land \neg w)$

$F_{CNF} = (x \lor z) \land (x \lor z \lor \neg w) \land (\neg y \lor z \lor x) \land (\neg y \lor z \lor \neg w)$

Problem: Applying the distributive law may result in an exponential blow-up.
Encoding Circuits

**Problem**
Given a formula $F$ in propositional logic with operations $\land$, $\lor$ and $\neg$, how can it be encoded in CNF?

**Example**

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

First approach: convert to NNF (Negation Normal Form), then apply distributive law.

**Example (cont'd)**

- $F_{\text{NNF}} = (x \land \neg y) \lor z \lor (x \land \neg w)$
- $F_{\text{CNF}} = (x \lor z) \land (x \lor z \lor \neg w) \land (\neg y \lor z \lor \neg w)$

Problem: Applying the distributive law may result in an exponential blow-up.
Encoding Circuits

Problem

Given a formula $F$ in propositional logic with operations $\wedge$, $\lor$ and $\neg$, how can it be encoded in CNF?

Example

$F = \neg((\neg x \lor y) \wedge (\neg z \wedge \neg(x \wedge \neg w)))$

First approach: convert to NNF (Negation Normal Form), then apply distributive law.
Encoding Circuits

Problem
Given a formula $F$ in propositional logic with operations $\land$, $\lor$ and $\neg$, how can it be encoded in CNF?

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

First approach: convert to NNF (Negation Normal Form), then apply distributive law.

Example (cont’d)

$F_{\text{NNF}} = (x \land \neg y) \lor z \lor (x \land \neg w)$
$F_{\text{CNF}} = (x \lor z) \land (x \lor z \lor \neg w) \land (\neg y \lor z \lor x) \land (\neg y \lor z \lor \neg w)$
Encoding Circuits

Problem
Given a formula $F$ in propositional logic with operations $\land$, $\lor$ and $\neg$, how can it be encoded in CNF?

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

First approach: convert to NNF (Negation Normal Form), then apply distributive law.

Example (cont’d)
$F_{\text{NNF}} = (x \land \neg y) \lor z \lor (x \land \neg w)$
$F_{\text{CNF}} = (x \lor z) \land (x \lor z \lor \neg w) \land (\neg y \lor z \lor x) \land (\neg y \lor z \lor \neg w)$

Problem: Applying the distributive law may result in an exponential blow-up.
Tseitin Encoding

Idea
Introduce new variables for subformulas.

Example:

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

"Triplets" using subformulas:

- Example:
  
  \[ f = a \lor b, \quad a = x \land y, \ldots \]

Encode each triplet (as equivalence) in CNF:

\[
(f \lor a \lor b) \land (f \lor a) \land (f \lor b) \land \ldots
\]

One additional clause \( f \) to assert that \( F \) must be true

Note: Sometimes implication instead of equivalence is sufficient
Tseitin Encoding

Idea

Introduce new variables for subformulas.

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

- "Triplets" using subformulas:
  \[ f = a \lor b, \quad a = x \land y, \ldots \]

- Encode each triplet (as equivalence) in CNF:
  \[ (f \lor a \lor b) \land (f \lor a) \land (f \lor b) \land \ldots \]

  One additional clause \( f \) to assert that \( F \) must be true

Note: Sometimes implication instead of equivalence is sufficient
Tseitin Encoding

Idea

Introduce new variables for subformulas.

- Example:
  \[ F = (x \land \neg y) \lor (z \lor (x \land \neg w)) \]
- “Triplets” using subformulas:
  \[ f = a \lor b, \quad a = x \land \bar{y}, \ldots \]

\[ \begin{align*}
  & a: \land \\
  & b: \lor \\
  & x \\
  & \neg y \\
  & z \\
  & \land : c \\
  & \neg w \\
\end{align*} \]
Tseitin Encoding

Idea

Introduce new variables for subformulas.

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

- "Triplets" using subformulas:
  \[ f = a \lor b, \quad a = x \land \bar{y}, \ldots \]

- Encode each triplet (as equivalence) in CNF:
  \[ (f \lor a \lor b) \land (f \lor \bar{a}) \land (f \lor \bar{b}) \land \ldots \]
Tseitin Encoding

Idea

Introduce new variables for subformulas.

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

- “Triplets” using subformulas:
  \[ f = a \lor b, \quad a = x \land \neg y, \ldots \]

- Encode each triplet (as equivalence) in CNF:
  \[ (f \lor a \lor b) \land (f \lor \neg a) \land (f \lor \neg b) \land \ldots \]

- One additional clause \((f)\) to assert that \(F\) must be true
Tseitin Encoding

Idea

Introduce new variables for subformulas.

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

- “Triplets” using subformulas:
  \[ f = a \lor b, \quad a = x \land \neg y, \ldots \]

- Encode each triplet (as equivalence) in CNF:
  \[ (f \lor a \lor b) \land (f \lor \neg a) \land (f \lor \neg b) \land \ldots \]

- One additional clause (\( f \)) to assert that \( F \) must be true

- Note: Sometimes implication instead of equivalence is sufficient
Tseitin Encoding: Definition

Given a formula $F$ in propositional logic using connectives $\land$, $\lor$ and $\neg$. The Tseitin-Encoding $\mathcal{T}(F)$ of $F$ is a formula in CNF defined by:

$$\mathcal{T}(F) = d_F \land \mathcal{T}^*(F)$$

$$\mathcal{T}^*(F) = \begin{cases} 
\mathcal{T}_{\text{def}}(F) \land \mathcal{T}^*(G) \land \mathcal{T}^*(H), & \text{if } F = G \circ H \text{ and } \circ \in \{\land, \lor\} \\
\mathcal{T}_{\text{def}}(F) \land \mathcal{T}^*(G), & \text{if } F = \neg G \\
\text{True,} & \text{if } F \in \mathcal{V} 
\end{cases}$$

$$\mathcal{T}_{\text{def}}(F) = \begin{cases} 
(d_F \lor d_G) \land (d_F \lor d_H) \land (d_F \lor \overline{d_G} \lor \overline{d_H}), & \text{if } F = G \land H \\
(d_F \lor d_G \lor d_H) \lor (d_F \lor \overline{d_G}) \land (d_F \lor \overline{d_H}), & \text{if } F = G \lor H \\
(d_F \lor \overline{d_G}) \land (d_F \lor d_G), & \text{if } F = \neg G 
\end{cases}$$

$\mathcal{T}(F)$ introduces a new variable $d_S$ for each subformula $S$ of $F$. $\mathcal{T}(F)$ is satisfiable iff $F$ is satisfiable.
Tseitin Encoding: Example

\[ F = (x \land \neg y) \lor (z \lor (x \land \neg w)) \]
Tseitin Encoding: Example

\[ F = \left( a, S_a \right) \lor \left( c, S_c \right) \]

Use auxiliary variables for subformulas as indicated above.
Tseitin Encoding: Example

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

- Use auxiliary variables for subformulas as indicated above.
- To simplify exposition, we treat negative literals like variables in \( T(F) \).
Tseitin Encoding: Example

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

- Use auxiliary variables for subformulas as indicated above.
- To simplify exposition, we treat negative literals like variables in \( \mathcal{T}(F) \).

\[
\mathcal{T}(F) = f \land \mathcal{T}^*(F) \\
= f \land \mathcal{T}_{\text{def}}(F) \land \mathcal{T}^*(S_a) \land \mathcal{T}^*(S_b) \\
= f \land \mathcal{T}_{\text{def}}(F) \land \mathcal{T}_{\text{def}}(S_a) \land \mathcal{T}_{\text{def}}(S_b) \land \mathcal{T}_{\text{def}}(S_c) \\
= f \land (f \lor a \lor b) \land (f \lor \overline{a}) \land (f \lor \overline{b}) \land \cdots \\
\mathcal{T}_{\text{def}}(F)
\]
Tseitin Encoding: Example (cont’d)

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

\[ \mathcal{T}(F) = f \land T_{\text{def}}(F) \land T_{\text{def}}(S_a) \land T_{\text{def}}(S_b) \land T_{\text{def}}(S_c) \]

\[ = f \land (\bar{f} \lor a \lor b) \land (f \lor \bar{a}) \land (f \lor \bar{b}) \]
\[ \land (\bar{a} \lor x) \land (\bar{a} \lor \bar{y}) \land (a \lor \bar{x} \lor y) \]
\[ \land (\bar{b} \lor z \lor c) \land (b \lor \bar{z}) \land (b \lor \bar{c}) \]
\[ \land (\bar{c} \lor x) \land (\bar{c} \lor \bar{w}) \land (c \lor \bar{x} \lor w) \]
Tseitin Encoding: Example (cont’d)

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

\[
T(F) = f \land T_{\text{def}}(F) \land T_{\text{def}}(S_a) \land T_{\text{def}}(S_b) \land T_{\text{def}}(S_c)
\]
\[
= f \land (\overline{f} \lor a \lor b) \land (f \lor \overline{a}) \land (f \lor \overline{b}) \\
\land (\overline{a} \lor x) \land (\overline{a} \lor \overline{y}) \land (a \lor \overline{x} \lor y) \\
\land (\overline{b} \lor z \lor c) \land (b \lor \overline{z}) \land (b \lor \overline{c}) \\
\land (\overline{c} \lor x) \land (\overline{c} \lor \overline{w}) \land (c \lor \overline{x} \lor w)
\]
Tseitin Encoding: Example (cont’d)

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

\[ T(F) = f \land T_{\text{def}}(F) \land T_{\text{def}}(S_a) \land T_{\text{def}}(S_b) \land T_{\text{def}}(S_c) \]
\[ = f \land (\overline{f} \lor a \lor b) \land (f \lor \overline{a}) \land (f \lor \overline{b}) \land (\overline{a} \lor x) \land (\overline{a} \lor \overline{y}) \land (a \lor \overline{x} \lor y) \land (\overline{b} \lor z \lor c) \land (b \lor \overline{z}) \land (b \lor \overline{c}) \land (\overline{c} \lor x) \land (\overline{c} \lor \overline{w}) \land (c \lor \overline{x} \lor w) \]
Tseitin Encoding: Example (cont’d)

\[ F = \underbrace{a, S_a}_{\text{a, } S_a} \lor \underbrace{b, S_b}_{\text{b, } S_b} \lor \underbrace{c, S_c}_{\text{c, } S_c} \]

\[ \mathcal{T}(F) = f \land \mathcal{T}_{\text{def}}(F) \land \mathcal{T}_{\text{def}}(S_a) \land \mathcal{T}_{\text{def}}(S_b) \land \mathcal{T}_{\text{def}}(S_c) \]

\[ = f \land (\bar{f} \lor a \lor b) \land (f \lor \bar{a}) \land (f \lor \bar{b}) \land (\bar{a} \lor x) \land (\bar{a} \lor \bar{y}) \land (a \lor \bar{x} \lor y) \land (\bar{b} \lor z \lor c) \land (b \lor \bar{z}) \land (b \lor \bar{c}) \land (\bar{c} \lor x) \land (\bar{c} \lor \overline{w}) \land (c \lor \bar{x} \lor w) \]
Tseitin Encoding: Example (cont’d)

\[ F = \left( x \land \neg y \right) \lor \left( z \lor \left( x \land \neg w \right) \right) \]

\[ T(F) = f \land T_{\text{def}}(F) \land T_{\text{def}}(S_a) \land T_{\text{def}}(S_b) \land T_{\text{def}}(S_c) \]
\[ = f \land (\overline{f} \lor a \lor b) \land (f \lor \overline{a}) \land (f \lor \overline{b}) \]
\[ \land (\overline{a} \lor x) \land (\overline{a} \lor \overline{y}) \land (a \lor \overline{x} \lor y) \]
\[ \land (\overline{b} \lor z \lor c) \land (b \lor \overline{z}) \land (b \lor \overline{c}) \]
\[ \land (\overline{c} \lor x) \land (\overline{c} \lor \overline{w}) \land (c \lor \overline{x} \lor w) \]
Tseitin Encoding: Example (cont’d)

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

\[ T(F) = f \land T_{\text{def}}(F) \land T_{\text{def}}(S_a) \land T_{\text{def}}(S_b) \land T_{\text{def}}(S_c) \]

\[ = f \land (\overline{f} \lor a \lor b) \land (f \lor \overline{a}) \land (f \lor \overline{b}) \]

\[ \land (\overline{a} \lor x) \land (\overline{a} \lor \overline{y}) \land (a \lor \overline{x} \lor y) \]

\[ \land (\overline{b} \lor z \lor c) \land (b \lor \overline{z}) \land (b \lor \overline{c}) \]

\[ \land (\overline{c} \lor x) \land (\overline{c} \lor \overline{w}) \land (c \lor \overline{x} \lor w) \]

\[ \overset{\text{SAT}}{=} \]

\[ (a \lor b) \land (\overline{a} \lor x) \land (\overline{a} \lor \overline{y}) \land (\overline{b} \lor z \lor c) \land (\overline{c} \lor x) \land (\overline{c} \lor \overline{w}) \]
Plaisted-Greenbaum Optimization [4]

\[ T(F) = d_F \land T^1(F) \]

\[ T^p(F) = \begin{cases} 
T^p_{\text{def}}(F) \land T^p(G) \land T^p(H), & \text{if } F = G \circ H \text{ and } \circ \in \{\land, \lor\} \\
T^p_{\text{def}}(F) \land T^{p+1}(G), & \text{if } F = \neg G \\
\text{True}, & \text{if } F \in \mathcal{V}
\end{cases} \]

\[ T^1_{\text{def}}(F) = \begin{cases} 
(d_F \lor d_G) \land (d_F \lor d_H), & \text{if } F = G \land H \\
(d_F \lor d_G \lor d_H), & \text{if } F = G \lor H \\
(d_F \lor \overline{d_G}), & \text{if } F = \neg G
\end{cases} \]

\[ T^0_{\text{def}}(F) = \begin{cases} 
(d_F \lor \overline{d_G} \lor \overline{d_H}), & \text{if } F = G \land H \\
(d_F \lor \overline{d_G}) \land (d_F \lor \overline{d_H}), & \text{if } F = G \lor H \\
(d_F \lor d_G), & \text{if } F = \neg G
\end{cases} \]
Integer Multiplication / Factorization

\[ s_{\text{out}} = s_{\text{in}} \oplus c_{\text{in}} \oplus A_i B_j \]
\[ c_{\text{out}} = s_{\text{in}} c_{\text{in}} + s_{\text{in}} A_i B_j + c_{\text{in}} A_i B_j \]
Choosing a good encoding is very important!
- Encoding highly influences runtime of SAT solver
- Hard to come up with good encodings

Cardinality constraints are important in many practical applications

Tseitin encoding allows to carry over structure to CNF

