Algorithms for Computational Logic

Introduction

Emmanuel Hebrard (adapted from João Marques Silva, Inês Lynce and Vasco Manquinho)

Outline

1. Applications of SAT
Applications of SAT

1. Encoding a Problem into SAT
2. CSP Encoding
3. Analyzing Encodings
4. Encoding Global Constraints

CNF Encodings

- CNF-SAT is NP-complete, and therefore as powerful as general SAT as a language
- Most research has focused on algorithms for CNF-SAT
- However, if polynomial encodings necessarily exist, they are not always easy to find
- Not all encodings are equal
  - What is a good encoding?
  - How to design a good encoding
From SAT to CNF-SAT

From SAT to CNF-SAT via the rules of Boolean algebra:

\[(a \implies (c \land d)) \lor (b \implies (c \land e))\]

- Decompose the implications

\[(a \implies c) \land (a \implies d) \lor ((b \implies c) \land (b \implies e))\]

- Rearrange disjunctions and conjunctions (conjunctions and disjunctions are distributive)

\[((a \implies c) \lor (b \implies c)) \land ((a \implies d) \lor (b \implies c)) \land ((a \implies d) \lor (b \implies e))\]

- Rewrite implications as disjunctions

\[(\overline{a} \lor c \lor \overline{b}) \land (\overline{a} \lor c \lor \overline{b} \lor e) \land (\overline{a} \lor d \lor \overline{b} \lor c) \land (\overline{a} \lor d \lor \overline{b} \lor e)\]

- Remove subsumed clauses

\[(\overline{a} \lor c \lor \overline{b}) \land (\overline{a} \lor d \lor \overline{b} \lor e)\]

Distributing is not efficient:

\[(x_1^1 \land x_2^1 \land \ldots \land x_k^1) \lor (x_1^2 \land x_2^2 \land \ldots \land x_k^2) \lor \ldots \lor (x_1^n \land x_2^n \land \ldots \land x_k^n)\]

- Up to \(k^n\) clauses of size up to \(n\)

- Tseitin’s encoding is polynomial in every case. Idea?

- Add extra variables
• Rewrite implications as disjunctions
• For every nested conjunction \((a \land \overline{b} \land c)\), introduce a fresh variable \(f\) and the clauses \((a \land \overline{b} \land c) \iff f\):

\[
(a \land \overline{b} \land c) \implies f : (\overline{a} \lor b \lor \overline{c} \lor f)
\]

\[
f \implies (a \land \overline{b} \land c) : \begin{cases}
\bar{f} \lor a \\
\bar{f} \lor \overline{b} \\
\bar{f} \lor c
\end{cases}
\]

• For instance for \((a \implies (c \land d)) \lor (b \implies (c \land e)) = (\overline{a} \lor (c \land d)) \lor (\overline{b} \lor (c \land e)):

\[
(\overline{c} \lor \overline{d} \lor f_1) \land (\overline{f}_1 \lor c) \land (\overline{f}_1 \lor d) \land (\overline{e} \lor \overline{f}_1) \land (\overline{f}_2 \lor c) \land (\overline{f}_2 \lor e) \land (\overline{a} \lor f_1) \land (\overline{b} \lor f_2)
\]

---

**Playing Sudoku**

Fill empty cells such that each row, each column and each 3x3 grid contains all of the digits 1 to 9.
Constraint Model for Sudoku

Modeling the problem with integer variables:
- Rows: i = 1, . . . , 9
- Columns: j = 1, . . . , 9
- Variables: v_{i,j} \in \{1,2,\ldots,9\}, i,j \in \{1,\ldots,9\}

Constraints:
- Each value used exactly once in each row:
  - For i \in \{1,\ldots,9\}, for j < k \in \{1,\ldots,9\}: v_{i,j} \neq v_{i,k}
- Each value used exactly once in each column:
  - For j \in \{1,\ldots,9\}, for i < k \in \{1,\ldots,9\}: v_{i,j} \neq v_{k,j}
- Each value used exactly once in each 3 \times 3 sub-grid:
  - For i,j,k,l \in \{1,9\}, if (k \neq i OR l \neq j) AND \lceil \frac{i}{3} \rceil = \lceil \frac{k}{3} \rceil AND \lceil \frac{j}{3} \rceil = \lceil \frac{l}{3} \rceil: v_{i,j} \neq v_{k,l}
- Each clue corresponds to a variable assignment:
  \begin{align*}
  v_{1,4} &= 1, v_{1,6} = 5, v_{1,8} = 6, v_{2,7} = 7, v_{2,9} = 1, \\
  v_{3,1} &= 9, v_{3,3} = 1, v_{3,8} = 3, v_{4,5} = 2, v_{6,6} = 6, \ldots
  \end{align*}

Constraint Satisfaction Problems

Constraint Satisfaction Problem (CSP)

Data: a triplet \( \mathcal{X}, D, C \) where:
- \( \mathcal{X} \) is a ordered set of variables
- \( D \) is a domain
- \( C \) is a set of constraints, where for \( c \in C \):
  - its scope \( S(c) \) is a list of variables
  - its relation \( R(c) \) is a subset of \( D^{\mid S(c)\mid} \)

Question: does there exist a solution \( \sigma \in D^{\mid \mathcal{X}\mid} \) such that for every \( c \in C \), \( \sigma(S(c)) \in R(c) \)?

Projection

The projection \( \sigma(\mathcal{X}) \) of a tuple \( \sigma \) on a set of variables \( \mathcal{X} = (x_{i_1}, \ldots, x_{i_k}) \subseteq \mathcal{X} \) as the tuple \( (\sigma(x_{i_1}), \ldots, \sigma(x_{i_k})) \)

Example: the constraint \( x + y = z \) (on the Boolean ring)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

S(\( x + y = z \))

R(\( x + y = z \))
Sudoku Example

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>( S(x \neq y) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

Sudoku

- \( \mathcal{X} = (v_1, \ldots, v_{9,9}) \)
- \( \mathcal{D} = \{1, \ldots, 9\} \)
- \( \mathcal{C} \): inequalities on rows, columns and subsquares; clues

Encoding Integer Domains

- Variable \( x \) with domain \( \{0, \ldots, n - 1\} \):
  - Log encoding: Boolean variables \( x_0, \ldots, x_{\lfloor \log n \rfloor} \) stands for \( x = \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \)
  - Direct encoding: Boolean variable \( x_j \) stands for variable \( x \) takes value \( j \)

- Direct encoding requires *consistency clauses* because it is not a bijection:
  - \( \sum_{j=1}^{n} x_j \geq 1 \): encode with \( (x_1 \lor x_2 \lor \ldots \lor x_n) \)
  - \( \sum_{j=1}^{n} x_j \leq 1 \) encode with: *Pairwise encoding* or *Sequential counters*
Pairwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with pairwise incompatibilities:
  - $x = i$ implies $x \neq j$
  - $\bigwedge_{1 \leq i < j \leq n} (\bar{x}_i \lor \bar{x}_j)$
  - $O(n^2)$ binary clauses
  - Encoding relations is easy and efficient:

  $\begin{array}{c|cccccccc}
  j & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
  x_j(x = j) & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
  \end{array}$

- One clause to forbid every non-tuple in the relation $R(c)$, e.g. for $x \neq y$:

  $\begin{array}{c|c}
  x & y \\
  \hline
  \bar{x}_1 \lor \bar{y}_1 \\
  \bar{x}_2 \lor \bar{y}_2 \\
  \bar{x}_3 \lor \bar{y}_3 \\
  \bar{x}_4 \lor \bar{y}_4 \\
  \bar{x}_5 \lor \bar{y}_5 \\
  \bar{x}_6 \lor \bar{y}_6 \\
  \vdots & \vdots
  \end{array}$

   Applications of SAT

Sequential Counter Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with sequential counter
- Introduce Boolean variables $s_1, \ldots, s_{n-1}$ with $s_i$ standing for $x \leq i$
  - $x = i$ implies $x \leq i$
  - $x = i$ implies $x > i - 1$
  - $x \leq i - 1$ implies $x \leq i$

  $\bigwedge_{1 \leq i < n} ((\neg x_i \lor s_i) \land (\neg x_i \lor \neg s_{i-1}) \land (\neg s_{i-1} \lor s_i))$
  $\land (\neg x_1 \lor s_1) \land (\neg x_n \lor \neg s_{n-1})$

  - $O(n)$ binary clauses ; $O(n)$ auxiliary variables

  $\begin{array}{c|cccccccc}
  j & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
  x_j(x = j) & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0
  \end{array}$

  $\begin{array}{c|cccccccc}
  j & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
  s_j(x \leq j) & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
  \end{array}$

  $\begin{array}{c|cccccccc}
  j & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
  x_j(x = j) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
  \end{array}$

   Applications of SAT
### Encoding Relations in Bitwise Encoding

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>conflict clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_2)</td>
<td>(x_1)</td>
<td>(x_0)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- Assume \(x = 1\), that is: \(x_2 = 0\), \(x_1 = 0\) and \(x_0 = 1\)
- The clause \((x_2 \lor x_1 \lor \overline{x}_0 \lor y_2 \lor y_1 \lor y_0)\) does not unit propagate!
- Unit propagation is **weaker** on the log encoding
- Notion of **Arc Consistency**

### Arc Consistency

- Let \(\mathcal{X}\) be a set of variables and \(\mathcal{D}\) be a domain:
  - \(\mathcal{D}(x)\) is the set of possible values for variable \(x \in \mathcal{X}\)

#### Arc Consistency

A constraint \(c\) is **Arc Consistent** on domain \(\mathcal{D}\) if and only if for every \(x \in S(c)\) and for every \(j \in \mathcal{D}(x)\), there exists a tuple \(\sigma \in R(c)\) such that \(\sigma(x) = j\).

- **Achieving Arc Consistency** on domain \(\mathcal{D}\) with respect to constraint \(c\) corresponds to reducing \(\mathcal{D}\) to the maximum \(\mathcal{D}' \subseteq \mathcal{D}\) such that \(\mathcal{D}'\) is arc consistent
  - If \(\mathcal{D}'\) is empty there is no solution satisfying relation \(c\) on domain \(\mathcal{D}\)

#### Applications of SAT

| \(R(c)(\cdot)\) | \(\{1, 2, 3\}\) | \(\{2, 3, 4, 5, 6\}\) | \(\{3, 4, 5, 6, 7\}\) |
| \(\mathcal{D}(\cdot)\) | \(\{1, 2, 3, 4, 5\}\) | \(\{1, 2, 5, 6, 7\}\) | \(\{1, 2, 3, 4, 5, 6\}\) |
Comparison of Encodings

- The size of the encoding is an important feature
  - Sequential counters are more concise than pairwise incompatibilities
- We have seen that unit propagation might not be the same on two logically equivalent encodings
  - Log vs. direct encoding of the constraint $x \neq y$
- We can ask whether a Boolean encoding of a constraint $c$ achieves arc consistency on domain $D$
  - Defined in the same way using the natural isomorphism between domain encodings

Encodings of the “Less than” Constraint

<table>
<thead>
<tr>
<th>Negative encoding</th>
<th>Support encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>one clause for every non-tuple in $R(c)$</td>
<td>one clause for every value, encoding its support values in $R(c)$</td>
</tr>
<tr>
<td>$x \neq y$</td>
<td>$(y_1)$</td>
</tr>
<tr>
<td>1  2</td>
<td>$(x_1 \lor y_1)$</td>
</tr>
<tr>
<td>1  3</td>
<td>$(x_2 \lor y_1)$</td>
</tr>
<tr>
<td>1  4</td>
<td>$(x_3 \lor y_1)$</td>
</tr>
<tr>
<td>2  3</td>
<td>$(x_4 \lor y_1)$</td>
</tr>
<tr>
<td>2  4</td>
<td>$(x_2 \lor y_2)$</td>
</tr>
<tr>
<td>3  4</td>
<td>$(x_3 \lor y_2)$</td>
</tr>
</tbody>
</table>

- The support encoding unit propagates $\bar{y}_1$ and $\bar{x}_4$, whereas the negative encoding does not
- Suppose that we know $x \neq 1$ ($\bar{x}_1$ is a new true literal)
- Unit propagation on the support encoding achieves arc consistency
Tseitin Encoding of Table Constraints

- Support encoding is only defined for binary relations.
- For dense relations, negative encoding is efficient.
  - For instance, the constraint $x \neq y$ contains $\frac{n-1}{n}$ tuples, and the negative encoding achieves arc consistency.
- Alternative to negative encoding for sparse constraints?

Tseitin encoding

<table>
<thead>
<tr>
<th>$x \neq y$</th>
<th>1 2</th>
<th>1 3</th>
<th>1 4</th>
<th>2 3</th>
<th>2 4</th>
<th>3 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z_{1,3}$</td>
<td>$(x_1 \land y_3)$</td>
<td>$z_{1,4}$</td>
<td>$(x_1 \land y_4)$</td>
<td>$z_{2,3}$</td>
<td>$(x_2 \land y_3)$</td>
<td>$z_{2,4}$</td>
</tr>
<tr>
<td>$(z_{1,2} \lor z_{1,3} \lor z_{1,4})$</td>
<td>$(x_1 \lor z_{1,2}) \land (y_1 \lor z_{1,2})$</td>
<td>$(z_{2,3} \lor z_{2,4})$</td>
<td>$(y_2 \lor z_{1,2})$</td>
<td>$(x_3 \lor z_{3,4})$</td>
<td>$(y_3 \lor z_{1,3}) \land (z_2 \lor z_{1,3})$</td>
<td>$(x_4)$</td>
</tr>
</tbody>
</table>

Applications of SAT

Tseitin’s Encoding of Table Constraints

- Consider a constraint $c$ of arity $|S(c)| := a$
- Let $S = \prod_{x \in S(c)} D(x)$ be the set of valid tuples (allowed by the domain $D$) with $|S| := s$
- Let $R(c) \cap S$ be the set of consistent tuples (valid and allowed by the constraint) with $|R(c) \cap S| := t$
- The negative encoding requires $\Theta(a(s - t))$ space
- Tseitin’s encoding requires $\Theta(at)$ space

Tseitin encoding and Arc Consistency

Unit propagation on Tseitin’s encoding is an optimal algorithm to achieve Arc Consistency on a table constraint.

- Tseitin’s encoding takes linear space
- Unit propagation takes linear time
- There is no sublinear algorithm to achieve arc consistency
Upgrading a Linux Distribution

- Let $U = \{p_1, \ldots, p_n\}$ be all versions of all Linux packages.
- Let $C \subseteq U^2$ be a set of conflicts (packages pairwise incompatible).
- For every package $p_i \in U$, we have:
  - a set $D_i$ of dependencies with $d \in D_i$ a set of packages such at least one of them is required for package $p_i$.
- An installation profile $P \subseteq U$ is valid iff, for every $p_i \in P$:
  - $C_i \cap P = \emptyset$ (there is no incompatibilities).
  - For each $d \in D_i, d \cap P \neq \emptyset$ (the dependencies are satisfied).
- An installation profile $P \subseteq U$ is non-regressive with respect to profile $P^o$ iff for each $p_i \in P^o, V_i \cap P \neq \emptyset$.

**Upgradeability problem**

Given a current installation profile $P^o$ and a package $p$, does there exist two sets of packages $P^+$ and $P^-$ such that $P \cup P^+ \setminus P^-$ is a valid non-regressive installation profile and contains $p$.

**Encoding**

- A variable $p_i$ for every $p_i \in U$: whether package $p_i$ should be in the installation profile.

- Constraints

  - **Compatibilities**: $\bar{p}_i \lor \bar{p}_j \quad \forall p_i \in U, \forall (p_i, p_j) \in C$

  - **Dependencies**: $\bar{p}_i \lor \bigvee_{p_j \in d} p_j \quad \forall p_i \in d, \forall d \in D_i$

  - **Non-regression**: $p_i \lor \bigvee_{p_j \in V_i, p_j} \quad \forall p_i \in P^o$
Objective

- Minimize the number of changes
- Introduce new variables to encode the delta

\[ p_i^\Delta \iff \bar{p}_i \land p_i \lor p_i^\Delta \lor \bar{p}_i \quad \forall p_i \in P^o \]

\[ p_i^\Delta \iff p_i \land p_i^\Delta \lor \bar{p}_i \lor \bar{p}_i^\Delta \quad \forall p_i \notin P^o \]

- Optimization is usually done by successive constraints
  - Top-down: \( \sum_{i=1}^n p_i < ub_0; \sum_{i=1}^n p_i < ub_1; \ldots ; \sum_{i=1}^n p_i < ub_k \) (with \( ub_i \) a feasible number of packages)
  - Bottom-up: \( \sum_{i=1}^n p_i > lb_0; \sum_{i=1}^n p_i > lb_1; \ldots ; \sum_{i=1}^n p_i > lb_k \) (with \( ub_i \) a infeasible number of packages)
  - Binary search

- How to encode a cardinality constraint?

Cardinality Constraints

- How to handle cardinality constraints, \( \sum_{j=1}^n x_j \leq k \)?
  - General form: \( \sum_{j=1}^n x_j \gg k \), with \( \gg \in \{<, \leq, =, \geq, >\} \)
  - Special case when \( k = 1 \) \( \sum_{j=1}^n x_j \leq 1 \)
    - AtMost1 constraints was the subject of the previous class

- Solution #1:
  - Use native PB solver, e.g. BSOL, PBS, Galena, Pueblo, etc.
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - E.g. Minisat++, Open-WBO, QMaxSat, MSUnCore, WPM2, etc.

- Solution #2:
  - Encode cardinality constraints to CNF
  - Use SAT solver
**General Cardinality Constraints**

- **General form:** \( \sum_{j=1}^{n} x_j \leq k \) (or \( \sum_{j=1}^{n} x_j \geq k \))
  - **Sequential counters**
    - Clauses/Variables: \( O(nk) \) [S05]
  - **BDDs**
    - Clauses/Variables: \( O(nk) \) [ES06]
  - **Sorting networks**
    - Clauses/Variables: \( O(n \log^2 n) \) [ES06]
  - **Cardinality Networks:**
    - Clauses/Variables: \( O(n \log^2 k) \) [ANORC09, ANORC11a]
  - **Totalizer**
    - Clauses: \( O(nk) \), Variables: \( O(n \log k) \) [BB03]
  - **Pairwise Cardinality Networks**
    - ...

---

**Sequential Counter Encoding**

Assume the general form: \( \sum_{i=1}^{n} x_i \leq k \)

- For each variable \( x_i \), create \( k \) additional variables \( s_{i,j} \) that are used as counters.
  - \( s_{i,j} = 1 \) if at least \( j \) of variables \( \{x_1 \ldots x_i\} \) are assigned value 1
  - \( s_{i,j} = 0 \) if at most \( j - 1 \) of variables \( \{x_1 \ldots x_i\} \) are assigned value 1

**Encoding:**

\[
\begin{align*}
\neg s_{i,j} & \quad \forall j : 1 < j \leq k \\
\neg x_i \lor s_{i,1} & \quad \forall i : 1 \leq i < n \\
\neg s_{i-1,j} \lor s_{i,j} & \quad \forall i,j : 1 \leq i < n, 1 < j \leq k \\
\neg x_i \lor \neg s_{i-1,j-1} \lor s_{i,j} & \quad \forall i,j : 1 < i < n, 1 < j \leq k \\
\neg x_i \lor \neg s_{i-1,k} & \quad \forall i : 1 < i \leq n
\end{align*}
\]
Does the sequential counter encoding achieve arc consistency on the cardinality constraint?

When is the constraint \( \sum_{j=1}^{n} x_j \leq k \) not arc consistent?

1. When more than \( k \) variables are true
2. When exactly \( k \) variables are true and at least 1 variable can be true

The value ‘false’ is always arc consistent

In all other cases, unassigned variables are indistinguishable: so any one of them can be true (in particular if all other are false)

Let see if unit propagation forbids (1) and (2)
Totalizer Encoding

- CNF encoding for cardinality constraints \( \sum_{i=1}^{n} x_i \leq k \)
- Count in unary how many of the \( n \) variables \( (x_1 \ldots x_n) \) are assigned value 1
- \( O(n \log n) \) new variables
- \( O(n^2) \) new clauses
  - Can be improved to \( O(nk) \)

Visualize the encoding as a tree
- Each node is (name : variables : sum)
  - Literals are at the leaves
  - Each node counts in unary how many leaves are assigned to 1 in its subtree
    - Example: if \( b_2 = 1 \), then 2 of the leaves \( (x_3, x_4, x_5) \) are assigned to 1
- Root node has the output variables \( (o_1 \ldots o_5) \) that count how many variables are assigned to 1
- To encode \( x_1 + x_2 + x_3 + x_4 + x_5 \leq 3 \) just set \( o_4 = 0 \) and \( o_5 = 0 \)
Suppose that an intermediate node $P$ that counts up to $n_1$ has two child nodes $Q$ and $R$ that count up to $n_2$ and $n_3$, respectively.

Note that $n_1 = n_2 + n_3$.

**Encoding:**

\[
\bigwedge_{\alpha \leq \alpha_{n_2}} \neg q_{\alpha} \lor \neg r_\beta \lor p_\sigma \quad \text{where,} \quad p_0 = q_0 = r_0 = 1
\]