SAT, CSP, and proofs

Ofer Strichman
Technion, Haifa

Tutorial HVC’13
The grand plan for today

- Intro: the role of SAT, CSP and proofs in verification
- SAT – how it works, and how it produces proofs
- CSP - how it works, and how it produces proofs
- Making proofs smaller
Why SAT?

Example: is \((x_1 \land (x_2 \lor \neg x_1))\) satisfiable?

\[ x_1, x_2 \in \mathcal{B} \]

Applications in verification:

- Formal verification:
  - (Bounded) model checking for hardware [1999 -- ]
    - Over a dozen commercial tools
  - (Bounded) model checking for software [2001 -- ]
    - CBMC, SAT-ABS, CLLVM, …
  - Satisfiability Modulo Theories (SMT) [2003 -- ]
    - e.g. MS – Z3 used in dozens of software analysis tools (SymDiff, VCC, Havoc, Spec#, …)
Why SAT?

Example: is \((x_1 \land (x_2 \lor \neg x_1))\) satisfiable?

\(x_1, x_2 \in B\)

- Applications in verification:
  - Simulation:
    - Test generation for hardware
    - Test generation for software via SMT
      - MS-SAGE, KLEE, …
Why CSP (Constraints Satisfaction Problem)?

Example:

is \((\text{AllDiff}(x_1, x_2, x_3) \lor x_1 < x_2 + 3 \land x_2 > x_3 - 1))\) satisfiable?

\(x_1, x_2, x_3 \in [0..10] \cap \mathbb{Z}\)

- Applications in verification:
  - Formal verification: ??
  - Simulation: test generation for hardware
Why CSP (Constraints Satisfaction Problem)?

Example:

is (AllDiff(x₁, x₂, x₃) ∨ x₁ < x₂ + 3 ∧ x₂ > x₃ - 1)) satisfiable?

x₁, x₂, x₃ ∈ [0..10] ∩ ℤ

- A Higher-level modeling language
  - Can lead to an order of magnitude smaller model size.
    - Does not matter much in practice

- Certain constraints can be solved faster than in SAT
  - Some (e.g. “all-different”) can be solved directly in P
SAT and CSP

- SAT is crawling towards CSP
  - Various SAT solvers now support high-level constraints over Boolean variables:
    - Cripto-minisat supports XOR constraints
    - MiniSat+ supports cardinality constraints $\sum w_i x_i \leq c$

- CSP is crawling towards SAT:
  - Some solvers support reduction to SAT
  - Solution strategy now mimics SAT
Why proofs?

- Traditionally the focus was on finding models
  - No information was given in case of UNSAT

- As of Chaff (2003 - ) solvers produce proofs
  - Originally just to validate result
Why proofs?

Several killer-applications (SAT):
- Validate UNSAT results
- From the proof we can extract an unsat core
  - Used in formal verification [AM03, KKB09, BKOSSB07…]
- Uses of the proof itself:
  - Interpolation-based model checking [M03].

Can we foresee usage for proofs in CSP?
The grand plan for today

- Intro: the role of SAT, CSP and proofs in verification
- SAT – how it works, and how it produces proofs
- CSP - how it works, and how it produces proofs
- Making proofs smaller
Conjunctive Normal Form: Conjunction of disjunction of literals. Example:

\[(\neg x_1 \lor \neg x_2) \land (x_2 \lor x_4 \lor \neg x_1) \land \ldots\]

Polynomial transformation to CNF due to Tseitin (1970)
- Requires adding auxiliary variables.
Main steps — SAT

**SAT**

- “Decide”
  - Variable, value

- “Boolean Constraints Propagation (BCP)”
  - infer implied assignments

- “Analyze conflict”
  - applies learning
  - computes backtracking level
About that “constraints propagation”

- **given** \((-x_1 \lor x_2) \land (x_2 \lor x_4 \lor -x_1) \land \ldots\)

  BCP: \(x_1 = 1 \Rightarrow x_2 = 0 \Rightarrow x_4 = 1 \Rightarrow \ldots\)
SAT essentials
Implication graphs and learning

Current truth assignment: \{x_9=0\atop1, x_{10}=0\atop3, x_{11}=0\atop3, x_{12}=1\atop2, x_{13}=1\atop2\}

Current decision assignment: \{x_1=1\atop6\}

\begin{align*}
\omega_1 &= (\neg x_1 \lor x_2) \\
\omega_2 &= (\neg x_1 \lor x_3 \lor x_9) \\
\omega_3 &= (\neg x_2 \lor \neg x_3 \lor x_4) \\
\omega_4 &= (\neg x_4 \lor x_5 \lor x_{10}) \\
\omega_5 &= (\neg x_4 \lor x_6 \lor x_{11}) \\
\omega_6 &= (\neg x_5 \lor \neg x_6) \\
\omega_7 &= (x_1 \lor x_7 \lor \neg x_{12}) \\
\omega_8 &= (x_1 \lor x_8) \\
\omega_9 &= (\neg x_7 \lor \neg x_8 \lor \neg x_{13})
\end{align*}

We learn the conflict clause \(\omega_{10} : (\neg x_1 \lor x_9 \lor x_{11} \lor x_{10})\)

and backtrack to the highest (deepest) dec. level in this clause (6).
Implication graph, flipped assignment

\[ \omega_1 = (\neg x_1 \lor x_2) \]
\[ \omega_2 = (\neg x_1 \lor x_3 \lor x_9) \]
\[ \omega_3 = (\neg x_2 \lor \neg x_3 \lor x_4) \]
\[ \omega_4 = (\neg x_4 \lor x_5 \lor x_{10}) \]
\[ \omega_5 = (\neg x_4 \lor x_6 \lor x_{11}) \]
\[ \omega_6 = (\neg x_5 \lor x_6) \]
\[ \omega_7 = (x_1 \lor x_7 \lor \neg x_{12}) \]
\[ \omega_8 = (x_1 \lor x_8) \]
\[ \omega_9 = (\neg x_7 \lor \neg x_8 \lor \neg x_{13}) \]
\[ \omega_{10} : (\neg x_1 \lor x_9 \lor x_{11} \lor x_{10}) \]
Non-chronological backtracking

Which assignments caused the conflicts?

\[
\begin{align*}
x_9 &= 0@1 \\
x_{10} &= 0@3 \\
x_{11} &= 0@3 \\
x_{12} &= 1@2 \\
x_{13} &= 1@2
\end{align*}
\]

These assignments are sufficient for causing a conflict.

Backtrack to decision level 3
Learning and resolution

- Learning of a clause = inference by resolution.
  - To be explained

- This is the key for producing a machine-checkable proof
Resolution

...By example:

\[
\frac{(x_1 \lor x_2) \quad (\neg x_1 \lor x_3 \lor x_4)}{(x_2 \lor x_3 \lor x_4)}
\]

Formally:

\[
\frac{(a_1 \lor \ldots \lor a_n \lor \beta) \quad (b_1 \lor \ldots \lor b_m \lor (\neg \beta))}{(a_1 \lor \ldots \lor a_n \lor b_1 \lor \ldots \lor b_m)}
\quad \text{(Binary Resolution)}
\]
Resolution proof

A proof: \((1 \ 3) \land (-1 \ 2 \ 5) \land (-1 \ 4) \land (-1 \ -4) \vdash (3 \ 5)\)
Resolution proof $\Rightarrow$ Hyper resolution proof

A proof: $(1\ 3) \land (-1\ 2\ 5) \land (-1\ 4) \land (-1\ -4) \vdash (3\ 5)$
Conflict clauses and resolution

Consider the following example:

- Conflict clause: $c_5: (x_2 \lor \neg x_4 \lor x_{10})$
- We show that $c_5$ is inferred by resolution from $c_1, \ldots, c_4$
Conflict clauses and resolution

- **Conflict clause:** $c_5: (x_2 \lor \neg x_4 \lor x_{10})$

- **BCP order:** $x_4, x_5, x_6, x_7$
  - $T_1 = \text{Res}(c_4, c_3, x_7) = (\neg x_5 \lor \neg x_6)$
  - $T_2 = \text{Res}(T_1, c_2, x_6) = (\neg x_4 \lor \neg x_5 \lor x_{10})$
  - $T_3 = \text{Res}(T_2, c_1, x_5) = (x_2 \lor \neg x_4 \lor x_{10})$
The Resolution-Graph

(Hyper) Resolution Graph

\( \omega_{10} \) can be inferred via resolution from \( \omega_1 \ldots \omega_6 \)
The resolution graph

What is it good for?
Example: for computing an **Unsatisfiable core**

[Picture Borrowed from Zhang, Malik SAT’03]
The grand plan for today

- Intro: the role of SAT, CSP and proofs in verification
- SAT – how it works, and how it produces proofs
- CSP - how it works, and how it produces proofs
- Making proofs smaller
# Main steps — SAT and CSP*

<table>
<thead>
<tr>
<th>SAT</th>
<th>CSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Decide”</td>
<td>Same</td>
</tr>
<tr>
<td>- Variable, value</td>
<td></td>
</tr>
<tr>
<td>“Boolean Constraints Propagation (BCP)”</td>
<td></td>
</tr>
<tr>
<td>- infer implied assignments</td>
<td></td>
</tr>
<tr>
<td>“Analyze conflict”</td>
<td>Same</td>
</tr>
<tr>
<td>- applies learning</td>
<td></td>
</tr>
<tr>
<td>- computes backtracking level</td>
<td></td>
</tr>
</tbody>
</table>

*As implemented in PCS / Michael Veksler
About that “constraints propagation”

- **Given** \( x_1, x_2, x_3 \in [1..3], \text{AllDifferent}(x_1, x_2, x_3) \)

  \[ \text{CP: } x_1 = 1 \Rightarrow x_2, x_3 \in [2..3] \]
What about CSP proofs?

- SAT solvers generate proofs:
  - From initial clauses to ( ).
  - Inference is via the binary-resolution rule.

- Unlike SAT solvers, CSPs:
  - have non-Boolean domains, and
  - non-clausal constraints.

- Can this gap be bridged?
  - The following is based on [SV10]
Signed CNF

... by examples:

- A positive signed literal: \( a \in \{1, 2, 3\} \).

- A negative signed literal: \( a \in \overline{\{1, 2, 4\}} \).

- A signed clause is a disjunction of signed literals. e.g.,

\[
(a \in \{1, 5\} \lor b \in \overline{\{4\}})
\]
Signed resolution

- A binary-resolution rule for signed-CNF:

\[
\frac{(\text{Labels}_1 \lor x \in A) \land (x \in B \lor \text{Labels}_2)}{(\text{Labels}_1 \lor x \in A \land B \lor \text{Labels}_2)} \quad (\text{sRes}(x))
\]

- Signed-clauses ✓

- What about other constraints?
  e.g. $\neq, \leq, \text{allDifferent}(v_1, \ldots, v_k)$

  should we just convert CSP to signed CNF?
Signed resolution

- **Q:** should we just convert CSP to signed CNF?
- **A:** No, because it is generally inefficient:
  - e.g., $x \neq y$ requires:
    
    $$(x \in \overline{\{1\}} \lor y \in \overline{\{1\}}) \land (x \in \overline{\{2\}} \lor y \in \overline{\{2\}}) \land \ldots$$
Towards a solution...

- **Solution:** introduce clauses *lazily*.

- **Consider a general constraint** $c$, such that:
  - In the context of $l_1 \land l_2 \land \ldots \land l_n$,
  - propagation of $c$ implies $l$:

  $$(l_1 \land l_2 \land \ldots \land l_n \land c) \rightarrow l$$
Towards a solution…

\[(l_1 \land l_2 \land \ldots \land l_n \land c) \rightarrow l\]

- Find an explanation clause \(e\) such that:
  - \(e\) is not too strong: \(c \rightarrow e\)
  - \(e\) is strong enough: \((l_1 \land l_2 \land \ldots \land l_n \land e) \rightarrow l\)
The structure of a CSP proof

\[ e_1, e_2, e_3 \] – explanation clauses.
For every constraint there is an explanation clause:

\[
\frac{\text{constraint}}{\text{explanation clause}} (\text{rule name})
\]
Explanation rules — example 1

- Constraint: \( x \neq y \)

\[
\begin{align*}
\frac{x \neq y}{x \in \{m\} \lor y \in \{m\}} (Ne(m))
\end{align*}
\]

\( m = \) the value that triggered the rule
Explanation rules — example 1

Propagation:
- context: $l_1 : (x = 1), \quad l_2 : (y \in [1..100])$.
- constraint: $c : x \neq y$.
- implies: $l : (y \in [2..100])$.

$$e : \quad (x \in \overline{1} \lor y \in \overline{1}) \quad // \quad Ne(1)$$

... indeed:
- $c^{Ne(1)} \quad e$
- $(l_1 \land l_2 \land e) \rightarrow l$
Explanation rules — example 2

Constraint: $x \leq y$

$$\frac{x \leq y}{(x \in (-\infty, m] \lor y \in [m + 1, \infty))} (LE(m))$$

Instantiate $m$ with $\text{max}(\text{domain}(y))$
Explanation rules — example 2

Propagation:
- context: \( l_1 : (x \in [1..3]), l_2 : (y \in [0..2]) \)
- constraint: \( c : x \leq y \).
- implies: \( l : x \in [1..2] \)

Explanation:
- \( e : (x \in (-\infty, 2] \lor y \in [3, \infty)) \).  /// = LE(2)

...indeed:
- \( c \xrightarrow{\text{LE(2)}} e \)
- \( (l_1 \land l_2 \land e) \rightarrow l \)
Each constraint has its rule …

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Name</th>
<th>Inference rule</th>
</tr>
</thead>
</table>
| \( a \neq b \)    | \( \text{Ne}(m) \) | \[
\frac{a \neq b}{(a \neq m \lor b \neq m)}
\] |
| \( x \leq y \)    | \( \text{LE}(m) \) | \[
\frac{x \leq y}{(x \in (-\infty, m] \lor y \in [m+1, \infty))}
\] |
| \( a = b \)       | \( \text{Eq}(D) \) | \[
\frac{a = b}{(a \notin D \lor b \in D)}
\] |
| \( a \leq b + c \) | \( \text{LE}_{+}(m, n) \) | \[
\frac{a \leq b + c}{(a \in (-\infty, m+n] \lor b \in [m+1, \infty) \lor c \in [n+1, \infty))}
\] |
| \( a = b + c \)   | \( \text{EQ}_{+}(l_{b, u_{b, l_{c}}, u_{c}}) \) | \[
\frac{a = b + c}{(a \in [l_{b} + l_{c}, u_{b} + u_{c}] \lor b \notin [l_{b}, u_{b}] \lor c \notin [l_{c}, u_{c})}
\] |
| \( \text{AllDiff}(v_{1}, \ldots, v_{k}) \) | \( \text{AD}(D, V) \) | \[
\frac{\text{AllDiff}(v_{1}, \ldots, v_{k})}{(V_{v \in V} v \notin D)}
\] |

\[\vdots\]
So this is how the proof looks like…

$e_1, e_2, e_3$ – explanation clauses.
The grand plan for today

- Intro: the role of SAT, CSP and proofs in verification
- SAT – how it works, and how it produces proofs
- CSP - how it works, and how it produces proofs
- Making proofs smaller
Minimizing the core

- The proof is not unique.
  - Different proofs / different cores.

- Can we find a minimum / minimal / smaller cores/proofs?
Minimizing the core

- **Core compression**
  - Smaller core [ZM03, …]
  - Minimal core [DHN06, …]
  - Min-core-biased search [NRS’13]

- **Proof compression:**
  - Exponential-time transformations [GKS’06]
  - Linear time transformations
    - “Recycle pivots” [BFHSS’08], …
Core compression (smaller core)

- A basic approach: run until reaching a fixpoint [chaff]
Core compression (minimal core)

Initially $\varphi$'s clauses are unmarked

- Remove an unmarked clause $c \in \varphi$?
  - SAT($\varphi$) ?
    - yes: mark $c$
    - no: $\varphi := \text{core}$
  - All marked: Return $\varphi$
Proof-compression linear-time transformation / “Recycle-pivots”

- Based on the following fact:
  - Every resolution proof can be made ‘regular’
  - … which means that each pivot appears not more than once on every path.
Proof-compression linear-time transformation / “Recycle-pivots”
Proof-compression linear-time transformation / “Recycle-pivots”

Reconstruct proof
Collect “removable literals”
Proof-compression linear-time transformation / “Recycle-pivots”
Proof-compression linear-time transformation / “Recycle-pivots”

- Resolution graphs are DAGs
  - So, a node is on more than one path to the empty clause
Proof-compression linear-time transformation / “Recycle-pivots”

- Resolution graphs are DAGs
  - So, a node is on more than one path to the empty clause
Proof-compression linear-time transformation / “Recycle-pivots”
Proof-compression linear-time transformation / “Recycle-pivots”

Does A dominate B?

Dominance relation can be found in $O(|E| \log |V|)$

Problem: need to be updated each time.
Proof-compression linear-time transformation / “Recycle-pivots”

- Possible solution:
  - Stop propagating information across nodes with more than one child.
Proof-compression
linear-time transformations /recent advances

- Recycle pivots with intersection

- Local transformation Framework

- Lower units

- Structural hashing

All implemented in PeRIPLO by S. Rollini. Together they reduce the proof size by ~40%.
SAT and CSP are not only about finding models
  - They can provide proofs

Proofs are important for
  - validation
  - extracting cores
  - various formal-verification techniques

Minimizing proofs/cores is a subject for intense research.
Questions ?