CSC 2547H: AUTOMATED REASONING WITH MACHINE LEARNING

Xujie Si
Assistant Professor
Department of Computer Science
University of Toronto
Paper presentation

- **Grading rubrics**
  - **Preparation (15%)**
    - Sign up on Ed 5%
    - Get feedback from TA 5%
    - Practice Recording 5%
  - **Presentation (70% + 15% bonus)**
    - Provide the necessary background 10%
    - Explain the problem and main challenges 10%
    - Illustrate the main ideas clearly 15%
    - Show the main results 15% + demo (15% bonus)
    - Limitations / related / future work discussion 10%
    - Finish under time 10% (15 ~ 20 minutes depending on the sign-ups)
  - **Question Answering (15%)**
    - In-class QA (10%)
    - Ed QA (5%)

<table>
<thead>
<tr>
<th>Week</th>
<th>#Sign-ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week4: ml4sat</td>
<td>3</td>
</tr>
<tr>
<td>Week5: ml4smt</td>
<td>3</td>
</tr>
<tr>
<td>Week6: fm4ml</td>
<td>6</td>
</tr>
<tr>
<td>Week7: ml4code</td>
<td>2</td>
</tr>
<tr>
<td>Week8: dl4code</td>
<td>4</td>
</tr>
<tr>
<td>Week9: dl+logic</td>
<td>3</td>
</tr>
<tr>
<td>Week10: nv-sym</td>
<td>1</td>
</tr>
</tbody>
</table>
Lecture Overview

• Recap SAT solving
• MaxSAT and Incremental Solving
• Satisfiability modulo Theories (SMT)
• DPLL(T)
• Oppen-Nelson Combination
Recap: DPLL Algorithm

Davis–Putnam–Logemann–Loveland (1962)

1. **Algorithm** DPLL(F):
2. $G \leftarrow \text{BCP}(F)$
3. **if** $G = \top$ **then** return true
4. **if** $G = \bot$ **then** return false
5. $p \leftarrow \text{Choose}(G)$
6. return DPLL($G[p \mapsto \top]$) || DPLL($G[p \mapsto \bot]$)

Better data structures
Better branching heuristics
Better backtracking
Recap: Ablation study of Modern SAT Solver

Importance of major features: Clause Learning > VSIDS > 2WL > Restart

[Source: Katebi, Skallah & Marques-Silva 2011]
MaxSAT and Incremental solving

• Maximum satisfiability
  - Local search (sub-optimal)
  - Iterative SAT solving with cardinality constraints

• Incremental solving
  - Assumption variables trick

\[
\begin{align*}
\neg x_1 \lor x_2 \\
\neg x_3 \lor x_2 \\
x_1 \lor x_2 \\
\neg x_3 \\
\neg x_2
\end{align*}
\]
Cardinality Constraint Encodings

Table 1. Comparison of different encodings for $\leq k (x_1, \ldots, x_n)$.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>#clauses</th>
<th>#aux. vars</th>
<th>decided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naïve</td>
<td>$\binom{n}{k+1}$</td>
<td>0</td>
<td>immediately</td>
</tr>
<tr>
<td>Sequential unary counter (LT$_{SEQ}^{n,k}$)</td>
<td>$\mathcal{O}(n \cdot k)$</td>
<td>$\mathcal{O}(n \cdot k)$</td>
<td>by unit prop.</td>
</tr>
<tr>
<td>Parallel binary counter (LT$_{PAR}^{n,k}$)</td>
<td>$7n - 3 \lfloor \log n \rfloor - 6$</td>
<td>$2n - 2$</td>
<td>by search</td>
</tr>
<tr>
<td>Bailleux &amp; Boufkhad [3]</td>
<td>$\mathcal{O}(n^2)$</td>
<td>$\mathcal{O}(n \cdot \log n)$</td>
<td>by unit prop.</td>
</tr>
<tr>
<td>Warners [4]</td>
<td>$8n$</td>
<td>$2n$</td>
<td>by search</td>
</tr>
</tbody>
</table>

Introducing auxiliary variables helps to reduce the number of clauses

Carsten Sinz, Towards an optimal CNF encoding of Boolean cardinality constraints, Constraint Programming, 2005
SAT vs SMT

• SAT problem

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

Solution: \(\{x_1 = T, x_2 = \bot\}\)

• SMT problem

\[(x + y < 3 \lor c[i] = x) \land ((i \gg 2) = j \lor f(x) = y)\]

Linear Integer Arithmetic

Arrays

Bit Vectors

Uninterpreted Function

Theory Solver(s)

SAT Solver
A bit history

Nelson-Oppen 1979 → LPSAT 1999


Zapato 2003 → Zap 2005

Shostak 1984 → SVC 1996

CVC 2002/2004 → CVC Lite 2004

ICS 2001 → ICS+Chaff 2002

Simplics 2005 → Yices 2006

DPLL(T) 2002 → Ario 2005

Sammy 2005 → BarcelogicTools 2005

GETFOL 1993 → KSAT 1996

MATH-SAT 2001 → MATH-SAT 2002

MATH-SAT 2005

2007

CVC3

2011

CVC5

2021

z3

2008
State-of-the-art Applications

Propositional Logic

• A set of primitive symbols
  - \( p, q, r, \ldots \)

• A set of operator symbols (aka. logical connectives)
  - \( \neg, \lor, \land, \rightarrow, \leftrightarrow \)

• Formula
  - A primitive symbol is a formula
  - “A logical connective + formulas” is also a formula

• Negation Normal Form (NNF)
  - Negation is only applied to variables/symbols
  - Only AND, OR can be used

• Conjunctive Normal Form (CNF)
  - Conjunction of disjunctions
  - Tseytin transformation
First-order Logic

- **Terms**
  - Variables
  - Functions $f(t_1, ..., t_n)$
    - Constants are functions with arity 0

- **Predicate**
  - $P(t_1, ..., t_n)$
  - A bit like “primitive symbols” in proposal logic

- **Formula**
  - Logical connectives
  - Quantifiers

- **Sentence**
  - Free variable (i.e., not bound by quantifiers)
  - FOL formula without free variables
First-order logic

• **Signature $\Sigma$**
  - A set of functions and predicates (aka, non-logical symbols)

• **$\Sigma$-formula**
  - All functions and predicates are in $\Sigma$

• **$\Sigma$-theory**
  - A set of $\Sigma$-formula
  - A theory only restricts functions and predicates

• **Model (aka structure)**
  - A mapping from variables, constants, nonlogical symbols to domain elements
1 Algorithm DPLL(T)(F):
2 \( F_p \leftarrow \text{encode}(F) \)
3 \textbf{while} true \textbf{do} \n4 \quad S_p, \text{res} \leftarrow \text{SAT}(F_p) \n5 \quad \textbf{if} \ \text{res} = \bot \ \textbf{then} \ \textbf{return} \ \text{false} \n6 \quad G \leftarrow \text{decode}(S_p) \n7 \quad T\text{-res} \leftarrow T\text{-Solve}(G) \n8 \quad \textbf{if} \ T\text{-res} = \top \ \textbf{then} \ \textbf{return} \ \text{true} \n9 \quad F_p \leftarrow F_p \land \neg S_p \n10 \textbf{end}
DPLL(T) Basic + optimization

• \((x = 1) \land (x = 2 \lor x = 3)\)

1 Algorithm DPLL(T) (F):
2 \(\text{FP} \leftarrow \text{encode} (F)\)
3 while true do
4 \(\langle S_p, \text{res} \rangle \leftarrow \text{SAT} (\text{FP})\)
5 if \(\text{res} = \bot\) then return false
6 \(G \leftarrow \text{decode} (S_p)\)
7 \(\langle G', \text{res} \rangle \leftarrow \text{T-Solve} (G)\)
8 if \(\text{res} = \top\) then return true
9 \(\text{FP} \leftarrow \text{FP} \land \neg \text{encode} (G')\)
10 end

\(\neg S_p\)

Disable the UNSAT core instead of the entire assignment
Algorithm CDCL():

while true do
    \(\alpha \leftarrow \alpha \cup \{\text{Choose}\}\) \\
    while \(\text{BCP()} = \text{conflict}\) do \\
    \(\text{backtrack-level} \leftarrow \text{AnalyzeConflict()}\) \\
    if \(\text{backtrack-level} < 0\) then \\
    \(\text{return } \text{false}\) \\
    else \\
    \(\text{BackTrack()}\) \\
    end \\
end \\
if \(\alpha\) is full assignment then \\
\(\text{return } \text{true}\) \\
end 

Algorithm CDCL(T()):

while true do \\
\(\alpha \leftarrow \alpha \cup \{\text{Choose}\}\) \\
while \(\text{BCP()} = \text{conflict}\) do \\
\(\text{backtrack-level} \leftarrow \text{AnalyzeConflict()}\) \\
if \(\text{backtrack-level} < 0\) then \\
\(\text{return } \text{false}\) \\
else \\
\(\text{BackTrack()}\) \\
end \\
end \\
if \(\alpha\) is full assignment then \\
\(\text{return } \text{true}\) \\
if \(\text{T-Solver(}\alpha\text{)}\) then \(\text{return } \text{true}\) \\
\(\text{AddClauses()}\) \\
end \\
end
Theory of EUF

A theory \( T \) is a set of formula (can be thought as axioms, i.e, “extra constraints”)

\( \phi \) is \( T \)-satisfiable if there exists a structure satisfies both \( \phi \) and \( T \)

\[
\forall x. \, x = x \quad \text{(Reflexivity)}
\]

\[
\forall x. \, \forall y. \, x = y \implies y = x \quad \text{(Symmetry)}
\]

\[
\forall x. \, \forall y. \, \forall z. \, x = y \land y = z \implies x = z \quad \text{(Transitivity)}
\]

\[
\forall \bar{x}, \bar{y}. \, \bigwedge_{i=1}^{n} x_i = y_i \implies f(\bar{x}) = f(\bar{y}) \quad \text{(Functional Congruence)}
\]
Animations of deciding EUF
Exercises of EUF

\[ f(a, b) = a \land f(f(a, b), b) \neq a \quad \text{UNSAT} \]

\[ a = b \land b = c \land g(f(a), b) = g(f(c), a) \land f(a) \neq b \quad \text{SAT} \]
A simple application of EUF

```c
int fun1(int y) {
    int x, z;
    z = y;  \[ z = y \land \]
    y = x;  \[ y_1 = x \land \]
    x = z;  \[ x_1 = z \land \]
    return x * x;
    \[ ret_1 = x_1 \times x_1 \]
}

int fun2(int y) {
    return y * y;
    \[ ret_2 = y \times y \]
}
```

\[
r_{ret2} = y \times y
\]

\[
z = y \land
\]
\[
y_1 = x \land
\]
\[
x_1 = z \land
\]
\[
ret_1 = x_1 \times x_1
\]

\[
r_{ret1} = x_1 \times x_1
\]

\[
not \ (ret_1 = ret_2)
\]

\[
r_{ret1} = f(x_1, x_1) \land
\]
\[
r_{ret2} = f(y, y) \land
\]
\[
not \ (ret_1 = ret_2)
\]
A simple application of EUF

```
int fun1(int y) {
    int x, z;
    z = y;
    y = x;
    x = z;
    return x * (x+1);
}
```

```
int fun2(int y) {
    return (y+1) * y;
}
```

“partially interpreted functions”

\[
\forall x, y \ f(x, y) = f(y, x)
\]

\[
\begin{align*}
z & = y \
y1 & = x \
x1 & = z \
t1 & = x1 + 1 \
ret1 & = x1 * t1
\end{align*}
\]

\[
\begin{align*}
z & = y \
y1 & = x \
x1 & = z \
t1 & = g(x1,1) \
ret1 & = f(x1, t1) \
t2 & = g(y, 1) \
ret2 & = f(t2, y) \
not (ret1 = ret2)
\end{align*}
\]
Another way of handling UF

Get rid of uninterpreted functions (UFs) by rewriting

\( \varphi^{EUF} \longrightarrow \varphi^E \)  

\((x_1 \neq x_2) \lor (F(x_1) = F(x_2)) \lor (F(x_1) \neq F(x_3))\)

**Flatten constraints:** \((x_1 \neq x_2) \lor (f_1 = f_2) \lor (f_1 \neq f_3)\)

**Functional consistency constraints:**

\[
(x_1 = x_2 \implies f_1 = f_2) \land \\
(x_1 = x_3 \implies f_1 = f_3) \land \\
(x_2 = x_3 \implies f_2 = f_3)
\]

Two possible encodings:

(Satisfiability checking) functional consistency constraints \(\land\) flatten constraints

(Validity checking) functional consistency constraints \(\Rightarrow\) flatten constraints
Difference logic

• Linear constraints

\[ x \geq y + c \]
\[ x \geq c \]

Job Scheduling
- N jobs, \( T_i \) is execution time for job \( i \)
- Need to finish all jobs before \( T \)
- Some jobs cannot execute at the same time

\[ (s_i, f_i) \]
\[ f_i \geq s_i + T_i \]
\[ s_i \geq f_j \lor s_j \geq f_i \]
\[ T \geq f_i \]
Theory of Arrays

John McCarthy, 1962

• Model arrays as functions
  \( read(a, i) \quad write(a, i, v) \)

• Read-over-write axioms

\[
\forall a, i, j, v: i = j \Rightarrow read(write(a, i, v), j) = v
\]
\[
\forall a, i, j, v: i \neq j \Rightarrow read(write(a, i, v), j) = read(a, j)
\]

\( ITE(i = j, v, read(a, j)) \)

Apply this trick exhaustively, only read operations remain,
Which can be further treated as uninterpreted functions.
If we further use Ackermann reduction, all will become equality logic constraints.
Theory of Inductive Data Types

\[
\begin{align*}
\text{nat} & := \text{succ}(\text{pred} : \text{nat}) \mid \text{zero}; \\
\text{list} & := \text{cons}(\text{car} : \text{tree}, \text{cdr} : \text{list}) \mid \text{null}; \\
\text{tree} & := \text{node}(\text{children} : \text{list}) \mid \text{leaf}(\text{data} : \text{nat});
\end{align*}
\]

- Constructor, Selector, Tester
- function symbol \(\iff\) constructor, selector
- predicate symbol \(\iff\) each tester

Example: list of int

- Constructors: \(\text{cons} : (\text{int}, \text{list}) \to \text{list}, \text{null} : \text{list}\)
- Selectors: \(\text{car} : \text{list} \to \text{int}, \text{cdr} : \text{list} \to \text{list}\)
- Testers: \(\text{is\_cons}, \text{is\_null}\)

Example: \(\forall x : \text{list}. (x = \text{null} \lor \exists y : \text{int}, z : \text{list}. x = \text{cons}(y, z))\)

Many other theories

- Theory of string
- Theory of bit vector
- Theory of linear arithmetic
- Theory of non-linear arithmetic
- Theory of integer linear arithmetic
- ...

26
Combining theories

• **Approach #1**
  - Reduce all theories to a common logic (e.g., propositional logic), if possible.

• **Approach #2**
  - Combine decision procedures of the individual theories.
  - The Nelson-Oppen method

Greg Nelson and Derek Oppen, *simplification by cooperating decision procedures*, 1979
Nelson-Oppen combination
The Theory-Combination problem

- Given theories $T_1$ and $T_2$ with signatures $\Sigma_1$ and $\Sigma_2$
- The combined theory $T_1 \oplus T_2$ has
  - signature $\Sigma_1 \cup \Sigma_2$ and
  - the union of their axioms.
- Let $\phi$ be a $\Sigma_1 \cup \Sigma_2$ formula.
- Does $T_1 \oplus T_2 \models \phi$?
The Theory-combination problem

- **Undecidable** (even when the individual theories are decidable).

- Under **certain restrictions**, it becomes decidable.

- We will assume the following restrictions:
  - \( T_1 \) and \( T_2 \) are decidable, quantifier-free, first-order theories with equality.
  - Disjoint signatures (other than equality): \( \Sigma_1 \cap \Sigma_2 = \emptyset \)
The Nelson-Oppen method (preprocessing)

Purification: validity-preserving transformation of the formula after which predicates from different theories are not mixed.

1. Replace an `alien’ sub-expression $\phi$ with a new auxiliary variable $a$
2. Constrain the formula with $a = \phi$

$$x_1 \leq f(x_1)$$

$$x_1 \leq a_1 \land a_1 = f(x_1)$$

Uninterpreted Functions

Arithmetic

Pure expressions, shared variables
The Nelson-Oppen method (easy case)

• Then we are left with several sets of pure expressions $F_1, \ldots, F_n$
• Each set belongs to some pure theory which we can decide
• $\phi$ is satisfiable $\iff F_1 \land \cdots \land F_n$ is satisfiable
• If any $F_i$ is unsatisfiable, then claim UNSAT (easy case!)
The Nelson-Oppen method (hard case)

- Q: How do different theories communicate?
  - Hint: they are only “connected” by equality constraints
- A: Broadcasting newly discovered equality constraints to other theories
- Either UNSAT is reached (some $F_i$ becomes UNSAT)
  Or there is no new equality constraints (all $F_1, ..., F_n$ are SAT)