CSC 2547H: AUTOMATED REASONING WITH MACHINE LEARNING

Xujie Si
Assistant Professor
Department of Computer Science
University of Toronto
Course Overview

• Goal
  - Introduction to the cutting-edge research on combining reasoning and machine learning
  - Understand relevant techniques and learn to use the state-of-the-art tools
  - With a special focus on symbolic constraints solving and programming applications

• Prerequisites
  - CSC373H1 Algorithm Design, Analysis & Complexity
  - CSC324H1 Principles of Programming Languages
  - CSC311H1 Introduction to Machine Learning

• Evaluation
  - Class participation (10%): attendance, in-class/online discussions
  - One Assignment (15%): “solver-aided” programming
  - Paper presentation + QAs (15%): 15-minute presentation by every student
  - Project (proposal 15%, presentation 20%, report 25%): up to 4 students/group
Course Structure and Schedule

- **Week 1-3: Intro to reasoning challenges**
  - SAT (week 1), SMT (week 2), Program Analysis & Synthesis (week 3)
- **Week 4-10: Paper presentations (~8 per week)**
  - Week 4: Machine learning for SAT
  - Week 5: Machine learning for SMT
  - Week 6: Formal methods for machine learning
  - Week 7: Classic machine learning for code
  - Week 8: Deep learning for code
  - Week 9: Deep learning and logic programming
  - Week 10: Neuro-symbolic systems
- **Week 11-12: Project presentations**
Presentation guidelines

• Preparation
  - Start as early as possible (at least two weeks in advance)
    - Check the course schedule: https://www.cs.toronto.edu/~six/csc-2547hs-w23.html
    - Make a post on Ed and indicate which paper you would like to present
  - Meet TA and ask for feedback (at least one week in advance)
  - Prepare a video recording (before the class)

• Content
  - Background
  - Problem & challenges
  - Main idea
  - Main results + demo (bonus)
  - Related/future work

• Tips
  - Clarity is most important (an obscure and confusing presentation is meaningless)
  - Your main goal is to teach others something cool from the selected paper
  - A big plus is to inspire others and yourself through your presentation
Logistics

• Office Hours
  - Instructor, Tuesday, 3 PM – 4 PM, BA 7072
  - TAs, 2 hours/week
  - Or by appointment

• Online Discussions
  - Ed
  - Ideally, all questions should go to Ed. Your post can be private if needed.

• Late submission policy
  - 15% off / day

• No plagiarism (absolutely)
  - Plagiarism detection software will be used

• No eating, drinking, etc. in class
Supplemental Textbooks

• No need to buy any
  - Recommended by not required
  - You can find (free) online copies
Instructor and TAs

• Instructor: Xujie Si
  - Assistant Professor
    ✗ UofT: 2023 – now
    ✗ McGill/Mila: 2021–22
  - PhD from UPenn (2020)
  - Homepage: https://www.cs.toronto.edu/~six/index.html

• Office: BA 7202

• Research interests:
  - PL: Program analysis, synthesis & verification
  - AI: Symbolic constraint solving, deep learning, neuro-symbolic systems
Intro - Jonathan Lorraine

- 4th year PhD candidate in machine learning group
- Advised by David Duvenaud
- Did MScAC at U of T beforehand

- My research focuses on nested optimization in Machine Learning
  - Hyperparameter optimization, learning in games, amortized optimization, …

- Homepage for more: https://www.cs.toronto.edu/~lorraine/
TA Intro: Zhaoyu Li

• 2\textsuperscript{nd} year PhD student
  - Spent 1\textsuperscript{st} year at McGill / Mila
  - Bachelor degree from Shanghai Jiao Tong University

• Research Interests
  - Neuro-symbolic learning and reasoning
  - Automated theorem proving

• Homepage
  - https://www.zhaoyu-li.com/
Some caveats (before we have fun) ...

• **Lots of paper reading**
  - 60+ research papers (see the tentative schedule)
  - Most are quite technical (you are required to read at least one carefully and deeply)

• **Lots of “hacking” (in different languages)**
  - Hand-on experiences of using/building/analyzing solvers
  - Kind of data scientist + system programming experts

• **Some necessary skills/traits**
  - Strong desire to learn new things (no one can force you to learn if you don’t want to)
  - Don’t be afraid of exotic notations (they are just notations, the ideas behind are essential)
  - Understand things with unknowns
  - Learn to read quickly (by ignoring some details)
  - Learn to debug complicated systems (by abstracting away certain details)
What is intelligence?

**Can compute fast?**

- Calculator

**Look like human?**

- Human-like robot

**Can differentiate cats from dogs?**

- Images of a cat and a dog

**Compute faster?**

- Supercomputer

**Can drive?**

- Self-driving car
What is Intelligence? (Cont.)
ARTIFICIAL GENERAL INTELLIGENCE

Human Intelligence, Animal Intelligence, Machine Intelligence, Alien Intelligence, you-name-it, ...

Which one is the strongest (so far)?

Which one will be the strongest?
Human Brain vs Turing Machine

After billion years of evolution, the universe produces intelligent brains

No computers can be more powerful than a Universal Turing Machine
Penrose’s three worlds philosophy

• Brain is fully determined by physics laws
• Physics laws are fully described by Math
• Math is created(?) by brain
Hardness Measure: NP-Complete, Undecidability

• **NP-complete (proudly originated from UofT)**
  - Nondeterministic Turing machine
  - Polynomial-time complete
  - A solution can be checked in polynomial time (on a deterministic Turing machine)
  - 3-SAT is the first well-known NP-complete problem (Cook, 1971)

• **Undecidability**
  - Impossible to construct an algorithm that always leads to yes-or-no answer
  - Regardless how long the algorithm may return
  - Halting problem is the first well-known undecidable problem
Main Progress: Heuristics + Engineering

Figure Credit: Armin Biere
WHY DOES IT MATTER?
A problem has been detected and Windows has been shut down to prevent damage to your computer.

UNMOUNTABLE_BOOT_VOLUME

If this is the first time you've seen this error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:

*** STOP: 0x000000ED (0x80F128D0, 0xc000009c, 0x00000000, 0x00000000)
A Killer Application: Software Verification

A Satisfiability Modulo Theories solver

x86  .NET CLR
SAGE, Pex, Yogi, Vigilante

Boogie

VCC  Spec #

Windows Drivers

SLAM Static Driver Verifier

Backend: SMT solver: Z3

SAT solver

Microsoft Windows

Windows 95

Windows Me

Windows 2000

Windows XP

Windows Vista

Windows 7

Windows 10
Shortly after the first successful moon landing, Dijkstra spoke to the head of development for the module software:

"How did you produce so many lines of perfect code?"

"Huh? We had a bug a few days before launch, it accidentally calculated the moon as repelling rather than attracting."

"Wow! Those guys were lucky to make it alive, then!"
Cannot be always lucky ...

Got a speeding ticket??  
It is kilometer NOT mile!!

Verification Tools Secure Online Shopping, Banking

Originally published in 2018

**Originating Technology/NASA Contribution**

Much is made of the engineering that enables the complex operations of a rover examining the surface of Mars—and rightly so. But even the most advanced robotics are useless if, when the rover rolls out onto the Martian soil, a software glitch causes a communications breakdown and leaves the robot frozen. Whether it is a Mars rover, a deep space probe, or a space shuttle, space operations require robust, practically fail-proof programming to ensure the safe and effective execution of mission-critical control systems.

Just as rovers are rigorously tested in simulated Martian conditions on Earth before actual mission launch, the software components must also be
Hard to be perfect, but statistics helps.
Learn/synthesize small code

Excel sees patterns and shows a preview

GitHub Copilot

1 # gcc -O3
2
3 .L0:
4      movq rsi, r9
5      movl ecx, ecx
6      shrq 32, rsi
7      andl 0xffffffff, r9d
8      movq rcx, rax
9      movl edx, edx
10     imulq r9, rax
11     imulq rdq, r9
12     imulq rsi, rdx
13     imulq rsi, rcx
14     addq rdx, rax
15     jae .L2
16     movabsq 0x100000000, rdx
17     addq rdx, rcx
18     .L2:
19      movq rax, rsi
20      movq rax, rdx
21      shrq 32, rsi
22      salq 32, rdx
23      addq rsi, rcx
24      addq r9, rdx
25      adcq 0, rcx
26      addq r8, rdx
27      adcq 0, rcx
28      addq rdi, rdx
29      adcq 0, rcx
30      movq rcx, r8
31      movq rdx, rdi

1 # STOKE
2
3 .L0:
4      shlq 32, rcx

Science
Hand-Drawn Images → Code → Images

for (i < 3)
    rectangle(3*i,-2*i+4,
              3*i+2,6)
    for (j < i + 1)
        circle(3*i+1,-2*j+5)

reflect(y=8)
for(i<3)
if(i>0)
    rectangle(3*i-1,2,3*i,3)
    circle(3*i+1,3*i+1)
Neuro-Symbolic Systems

Query
$2 + 3 \times 4 = ?$

DeepProbLog Program

%Neural predicate
nn(net, [X], Y, [0..9]) :: digit(X,Y).
%Background knowledge
addition(X,Y,Z) :- digit(X,N1), digit(Y,N2), Z = N1+N2.

Logical Reasoning

addition([2,3], 8) :- digit([2], 0), digit([3], 8), 8 is 0+8.
... addition([2,3], 8) :- digit([2], 5), digit([3], 3), Z is 5+3.
...

Neural network evaluation

nn(net, [2], Y, [0..9]) :: digit([2], Y).

2 + 3 \times 4 = ?

Compose

Execute

SceneGraph

01
attr
tall

02
attr
tree

03
attr
giraffe

attr
rhino

<part geometry="z_2">
<pose ref="left" trans="left" />
<pose ref="right" trans="right" />
</part>

...
Let’s zoom into the very bottom
SAT Preliminaries

• Variables
  - w, x, y, z, a, b, c, x₁, x₂, ...

• Literals
  - Variables or their negations, e.g., x, ¬y (or ¬y)

• Clauses
  - Disjunction of literals, e.g., a ∨ x₁ ∨ y

• Formula
  - Conjunction of clauses, e.g., (x₁ ∨ ¬y) ∧ (x₂ ∨ ¬x₁)

• Model
  - A partial/total mapping from variables to True (T) or False (⊥),
    - e.g., {x₁ → T, x₂ → T, y → ⊥}

• Formula can be satisfiable (SAT) or unsatisfiable (UNSAT)
Notation simplification

• Literal
  - Use $i$ to denote $x_i$
  - Use $-i$ (or $\bar{i}$) to denote $\neg x_i$ (or $\overline{x_i}$)

• Clause
  - Use a set $\{x_1, \neg x_2, x_3\}$ to represent a disjunction $x_1 \lor \neg x_2 \lor x_3$
  - Which can be further simplified as $\{1, -2, 3\}$

• Formula
  - Use a set $\{c_1, c_2, c_3\}$ to represent a conjunction $c_1 \land c_2 \land c_3$
  - E.g., $(x_1 \lor \neg x_3) \land (x_2 \lor \neg x_1)$ can be simplified as $\{\{1, -3\}, \{2, -1\}\}$

• Model
  - Use a set to represent a mapping
  - $\{x_1 \rightarrow T, x_2 \rightarrow \bot, x_3 \rightarrow T\}$ can simplified as $\{1, -2, 3\}$

DIMACS format:
```
c This is a comment
c DIMACS format
p cnf 3 2
  1 2 -3 0
-2 3 0
c here is a solution
s 1 -2 3
```

Formula: $(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_2 \lor x_3)$

Solution: $\{x_1 \rightarrow T, x_2 \rightarrow \bot, x_3 \rightarrow T\}$
Basic preprocessing

• Pure literal
  - A variable $x$ occurs only positively (or only negatively)
  - Remove all clauses containing $x$

• Tautology clause
  - A clause is always true (thus can be removed)
  - E.g., $x \lor x_2 \lor \neg x_1 \lor \cdots$

• Subsumption
  - $c_1$ subsumes $c_2$ if and only if $c_1 \Rightarrow c_2$ (or $c_1 \subseteq c_2$)
  - E.g., $(x_1 \lor \neg x_3) \Rightarrow (x_1 \lor x_2 \lor \neg x_3)$ (or $\{1,-3\} \subseteq \{1,2,-3\}$
  - $c_2$ can be removed safely

• Unit propagation
  - A clause contains a single literal $l$
  - $l$ has to be true if there exists a solution or model
Scalability of (practical) SAT Solving

- **1952**: Quine, ≈10 var
- **1960**: DP, ≈10 var
- **1962**: DLL, ≈10 var
- **1986**: BDDs
- **1988**: SOCRATES, ≈3k var
- **1992**: GSAT
- **1994**: HannibalGRASP, ≈3k var
- **1996**: Stålmarck, ≈1k var
- **2001**: Chaff, ≈300 var
- **2002**: Berkmin, ≈10k var

1996 SATO, ≈1k var
SAT Competition Winners on the SC2020 Benchmark Suite

CPU time

solved instances

data produced by Armin Biere and Marijn Heule
Let’s brainstorm a bit …

• It is easy to check whether an assignment is satisfiable or not

• Algorithm-0
  - Randomly generate an assignment and check if it is a satisfiable solution

• Algorithm-1
  - Let’s enumerate all possible assignments, say in lexicographic order
  - E.g., starting with all variables are True, then only one variable is False… then two …

• Algorithm-2
  - Once a variable’s truth value has been decided, we can simplify the formula
  - We may enumerate in different orders
• A.k.a, Boolean Constraint Propagation (BCP)
  - If a clause consists of a single literal (called unit clause), that literal has to be True.
  - Simplify the formulation, and perform BCP recursively if there is new unit clause(s)
Empty Formula vs Empty Clause

- Empty formula is trivially satisfied.
- Empty clause cannot be satisfied.

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

\[
x_1 = T 
\]

Empty formula $\equiv$ no more constraints

Empty clause $\equiv$ no more literals that can be assigned to True
Other basic pre-processings

• Pure literal
  - A variable $x$ occurs only positively (or only negatively)
  - Remove all clauses containing $x$

• Tautology clause
  - A clause is always true (thus can be removed)
  - E.g., $x \lor x \lor \neg x \lor \cdots$

• Subsumption
  - $c_1$ subsumes $c_2$ if and only if $c_1 \Rightarrow c_2$ (or $c_1 \subseteq c_2$)
  - E.g., $(x_1 \lor \neg x_3) \Rightarrow (x_1 \lor x_2 \lor \neg x_3)$ (or $\{1, -3\} \subseteq \{1, 2, -3\}$
  - $c_2$ can be removed safely
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
Optimizing BCP

- BCP takes 80-90% of solver time
- Classic implementation
  - For each clause, have counters for satisfied, falsified, and unresolved literals
  - When a literal is set or unset, update counters for all relevant clauses
- “2 watched literals” trick
  - A clause does not affect the search if it has two or more non-falsified literals
  - Only need to pick two literals to watch for each clause
- When either is falsified
  - Check if there is another non-falsified literal, use that one as the new watched literal
  - Otherwise, the current clause becomes unit clause
Advantages of "2 watched literals"

- Fewer clauses are visited when a literal is set
- \textit{unset} is $O(1)$
  - No literals are falsified
  - Watched literals are unchanged
- When a literal is frequently "set-then-unset"
  - Fewer clauses will be affected
  - When a clause is affected for the very first time, another watched literal is chosen
Two popular branching heuristics

- Dynamic Literal Individual Sum (DLIS)

- Variable State Independent Decaying Sum (VSIDS)

```plaintext
1 Function ChooseDLIS(F):
   2   for clause cl ∈ F do
   3      for literal lit ∈ cl do
   4         count[lit] ← count[lit] + 1
   5      end
   6   end
   7 return literal w. the max. count

Dynamic Literal Individual Sum (DLIS)

1 Function ChooseVSIDS(F):
   2 // initialize scores
   3   score[v ∈ vars] ← 0
   4 // when adding a learnt clause
   5   for v ∈ learnt clause do
   6       score[v] ← score[v] + 1
   7   end
   8 // after every N steps
   9   for v ∈ vars do
   10      score[v] ← \frac{\text{score}[v]}{c}
   11   end
   12 return variable w. the highest score

Variable State Independent Decaying Sum (VSIDS)
```
Decision Levels

\[ \mathcal{F} = (r) \land (\bar{r} \lor s) \land \\

(\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b) \\
(\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d) \]

• Decisions / Variable Branchings:
  \( w = 1, x = 1, y = 1, z = 1 \)
  \( r@0 = 1 \)
  \( w@1 = 1 \)
  \( x@2 = 1 \)
  \( d@4 = 1 \)

\[ \neg b_1 \lor \neg b_2 \lor \cdots \lor \neg b_n \lor h \\
( b_1 \land b_2 \land \cdots \land b_n ) \Rightarrow h \]

Horn clause: a clause with at most one positive literal
Conflict-driven Clause Learning

Marques-Silva & Sakallah, 1996

\[ F = \{ c_1, c_2, c_3, c_4, c_5, c_6, \ldots, c_9 \} \]

Many steps later...

\[ x_1 = T, x_4 = T \]

\[ \neg x_1 \lor \neg x_4 \quad \text{NOT} \ (x_1 = T \land x_4 = T) \]

\[ \neg x_1 \lor \neg x_8 \lor x_7 \quad \text{NOT} \ (x_1 = T \land x_8 = T \land x_7 = \bot) \]
Non-chronological backtracking

Backtrack to level $d_{k-1}$ so that $C$ will become a unit clause immediately.

$C_{learnt} = l_{d_1} \lor l_{d_2} \lor \cdots \lor l_{d_k}$

where $d_1 < d_2 < \cdots < d_k$

(failed)

$\neg x_1 \lor \neg x_8 \lor x_7$

$x_{1@1} = T$

$x_{8@2} = T$

$x_{7@2} = T$

$x_{1@1} = T$

$x_{4@1} = \bot$
How to choose a proper conflict?

• **Unique Implication Point (UIP)**
  - A single node at level \( d \) such that all paths from the current decision literal \((lit@d)\) to the conflict \((k@d)\)
  - Obviously, the source node \((lit)\) and the sink node \((k)\) are UIPs.

• **First UIP strategy**
  - Pick the conflict that consists of the closest UIP to the conflict node
Restart

• Restart from scratch once in a while

• Why useful at all?
  - (Some) Learnt clauses will be kept
  - Even with same heuristics, the search will go to a different direction
  - With learnt clauses, extra pre-processing (or “in-processing”) can be performed
Ablation Study of Modern CDCL Solver

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

[Source: Katebi, Skallah & Marques-Silva 2011]
Well-known SAT solvers

- Chaff (2002)
  - https://www.princeton.edu/~chaff/zchaff.html
- MiniSat (2005)
  - http://minisat.se/
- Glucose (2009)
  - https://www.labri.fr/perso/lisimon/glucose/
- CaDiCaL (2017)
  - http://fmv.jku.at/cadical/
- Kissat (2020)
  - https://github.com/arminbiere/kissat
Proof

- A satisfiable solution is easy to check/verify

- What if a solver claims UNSAT?
  - A sequence of resolution ends up with an empty clause
  - CDCL is essentially constructing a resolution-based proof when an instance is UNSAT
  - The proof size could be quite large
  - For the pigeonhole problem, the resolution-based proof size is exponential

Armin Haken, *The intractability of resolution*, Theoretical Computer Science, 1985

Limitation of CDCL

Pigeonhole principle

If we put \( n + 1 \) pigeons into \( n \) holes, there exists at least one hole with more than one pigeons.

How to encode PHP as a SAT solving problem?

CDCL-based SAT solvers suffer from solving PHP instances
Satisfiability vs Validity

• Satisfiability
  - There exists some assignment so that the formula is true.
  - \( \exists x_1, x_2, \ldots, x_n. \phi(x_1, \ldots, x_n) \)

• Validity
  - For all assignments, the formula is true.
  - \( \forall x_1, x_2, \ldots, x_n. \phi(x_1, \ldots, x_n) \)

• The negation of one is equivalent to the other
  - NOT \( (\exists x_1, x_2, \ldots, x_n. \phi(x_1, \ldots, x_n)) \equiv \forall x_1, x_2, \ldots, x_n. \neg(\phi(x_1, \ldots, x_n)) \)
  - NOT \( (\forall x_1, x_2, \ldots, x_n. \phi(x_1, \ldots, x_n)) \equiv \exists x_1, x_2, \ldots, x_n. \neg(\phi(x_1, \ldots, x_n)) \)
Pigeonhole Principle

“For all possible arrangements, some hole contains one or more pigeons.”

“There exists an arrangement, no hole contains more than one pigeons.”

“There exist an arrangement, each hole contains one (or less) pigeon.”

Tautology
Negation
UNSAT
Rephrase
UNSAT
SAT Encoding of Pigeonhole problem

• N+1 pigeons, N holes
• Boolean variable $x_{i,j}$
  - the $i$-th pigeon is placed in the $j$-th hole
• Each pigeon should be in some hole
  - $x_{i,1} \lor x_{i,2} \lor \cdots \lor x_{i,n}$
  - N+1 clauses
• No hole contains more than one pigeon
  - Any pair of pigeons should not be in the same hole
  - $\neg x_{i,j} \lor \neg x_{k,j}$ where $i \neq k$
  - For each hole, $\frac{(n+1)\cdot n}{2}$ clauses
Demo: try Minisat on Pigeonhole

- Minisat solver
  - http://minisat.se/
- PHP constraints generation
  - https://user.it.uu.se/~tjawe125/software/pigeonhole/