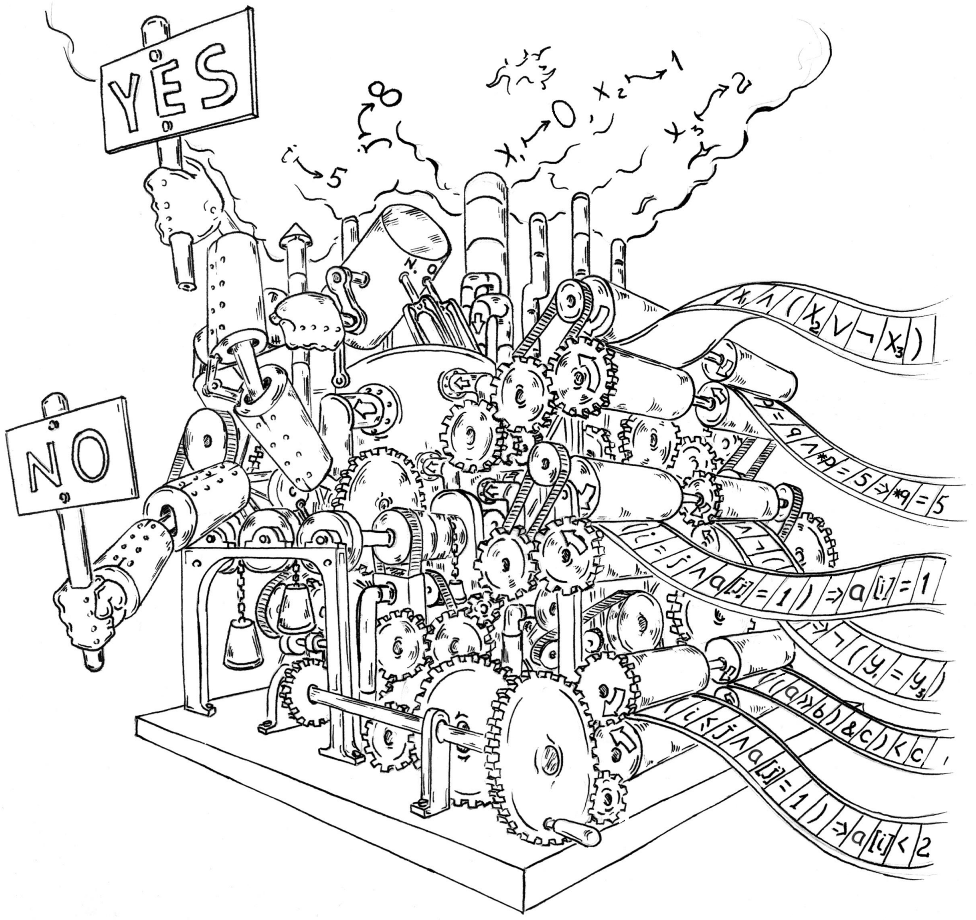
CSC 2547H: AUTOMATED REASONING WITH MACHINE LEARNING

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%)

Week	#Sign-ups
Week4: ml4sat	3
Week5: ml4smt	3
Week6: fm4ml	6
Week7: ml4code	2
Week8: dl4code	4
Week9: dl+logic	3
Week10: nv-sym	1

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)

Algorithm $\texttt{DPLL}(F)$:

  $\boxed{G \leftarrow \texttt{BCP}(F)}$ ← Better data structures

  if $G = \top$ **then return** true

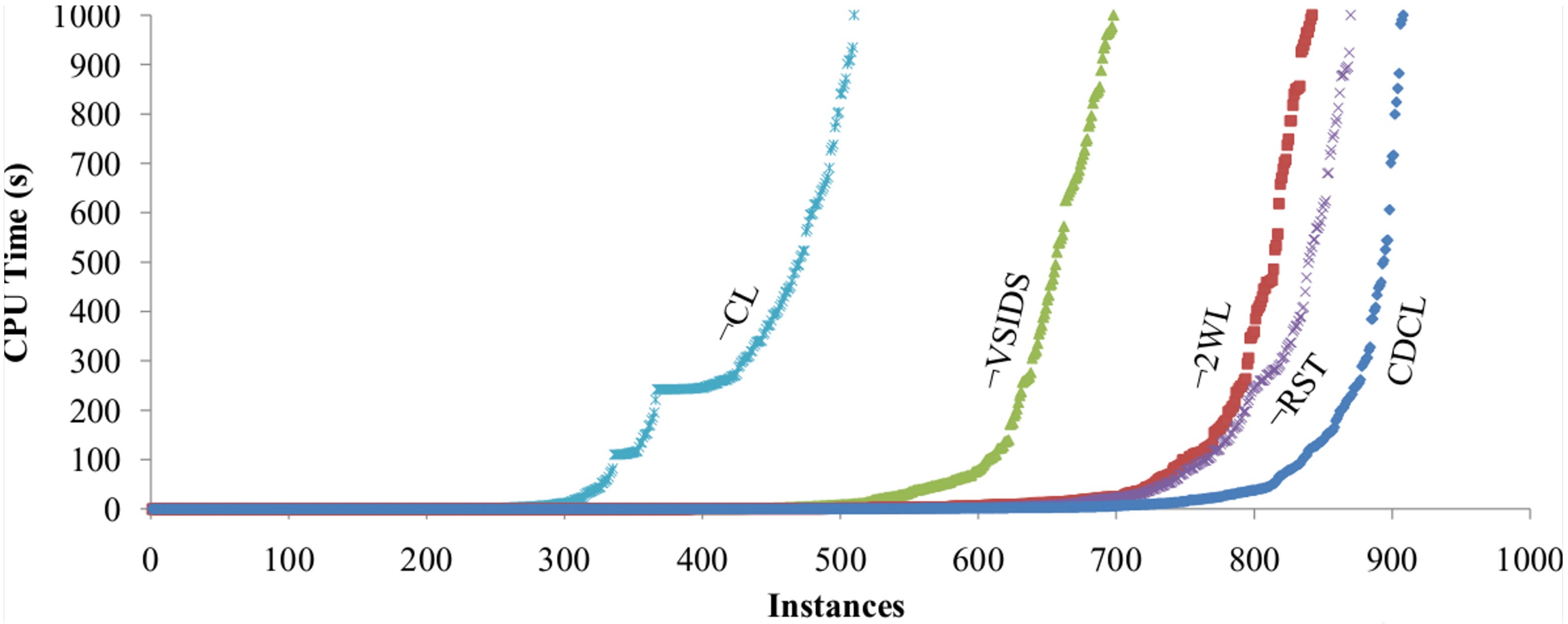
  if $G = \bot$ **then return** false

  $\boxed{p \leftarrow \texttt{Choose}(G)}$ ← Better branching heuristics

  return $\underbrace{\texttt{DPLL}(G\{p \mapsto \top\}) \parallel \texttt{DPLL}(G\{p \mapsto \bot\})}_{\text{Better backtracking}}$

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

$(\neg x_1 \vee x_2)$
$(\neg x_3 \vee x_2)$
$(x_1 \vee x_2)$
$(\neg x_3)$
$(\neg x_2)$

Cardinality Constraint Encodings

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

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

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**

$$\underbrace{(x_1 \vee x_2)}_{\textbf{clause}} \wedge (\underbrace{\neg x_1}_{\textbf{literal}} \vee \neg x_2)$$

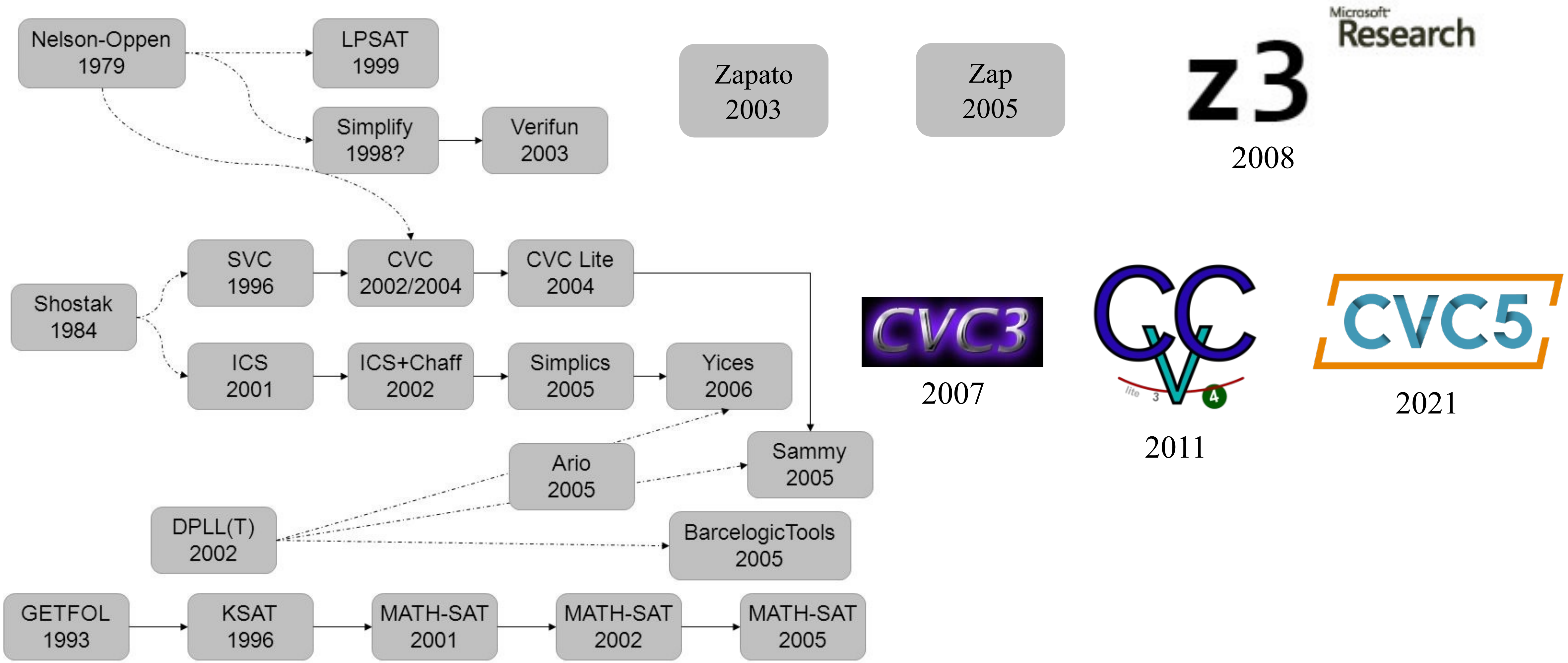
Solution: $\{x_1 = \top, x_2 = \bot\}$

- **SMT problem**

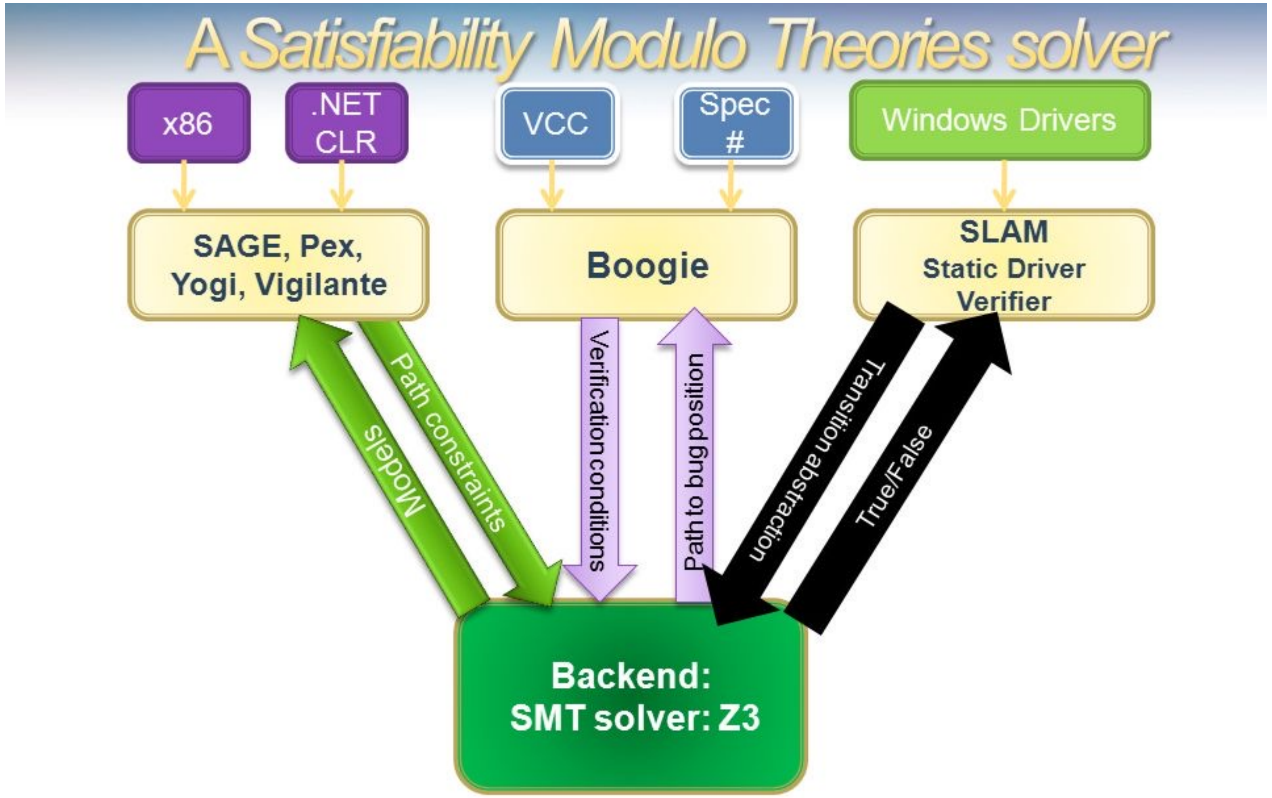
$$(\underbrace{x + y < 3}_{\text{Linear Integer Arithmetic}} \vee \underbrace{c[i] = x}_{\text{Arrays}}) \wedge (\underbrace{(i \gg 2) = j}_{\text{Bit Vectors}} \vee \underbrace{f(x) = y}_{\text{Uninterpreted Function}})$$

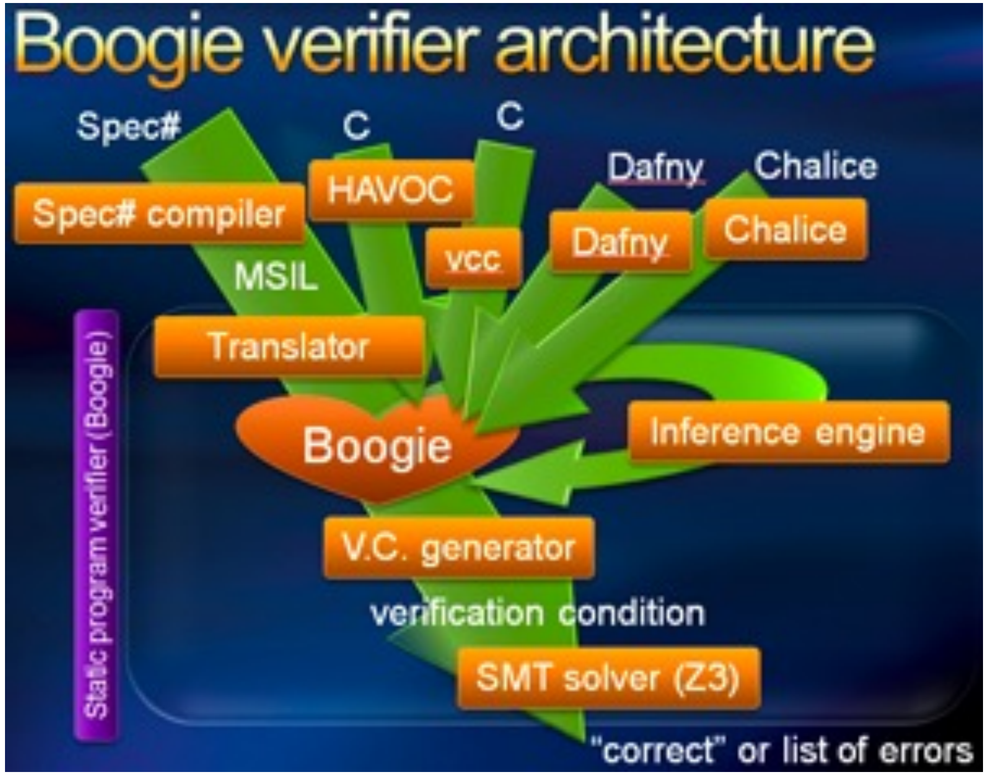
Theory Solver(s) ⇄ **SAT Solver**

A bit history



State-of-the-art Applications





https://github.com/dafny-lang/dafny SMT/Z3 tutorial: https://leodemoura.github.io/slides.html

Propositional Logic

- A set of primitive symbols
  - $p, q, r, \ldots$
- A set of operator symbols (aka. logical connectives)
  - $\neg,\ \vee,\ \wedge,\ \rightarrow,\ \leftrightarrow$
- 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 $f(t_1, \dots, tn)$
    - Constants are functions with arity 0
- **Predicate**
  - $P(t_1, \dots, t_n)$
  - 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 Σ**
  - A set of functions and predicates (aka, non-logical symbols)
- **Σ-formula**
  - All functions and predicates are in Σ
- **Σ-theory**
  - A set of Σ-formula
  - A theory only restricts functions and predicates
- **Model (aka structure)**
  - A mapping from variables, constants, nonlogical symbols to domain elements

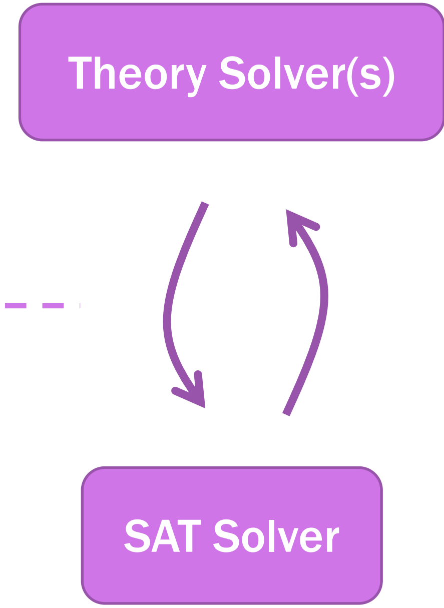
DPLL(T) Basic

```
Algorithm DPLL(T)(F):
    F_p ← encode (F)
    while true do
        S_p, res ← SAT(F_p)
        if res = ⊥ then return false
        G ← decode(S_p)
        T-res ← T-Solve(G)
        if T-res = ⊤ then return true
        F_p ← F_p ∧ ¬S_p
    end
```

$$x = y \wedge$$
$$((y = z \wedge \neg(x = z)) \vee x = z)$$

A: x=y; B: y=z; C: x=z

$$A \wedge$$
$$((B \wedge \neg C) \vee C)$$



DPLL(T) Basic + optimization

- $(x = 1) \land (x = 2 \lor x = 3)$

```
Algorithm DPLL(T)(F):
    F_p ← encode (F)
    while true do
        ⟨S_p, res⟩ ← SAT(F_p)
        if res = ⊥ then return false
        G ← decode(S_p)
        ⟨G′, res⟩ ← T-Solve(G)
        if res = ⊤ then return true
        F_p ← F_p ∧ ¬ encode(G′)
    end
```

$\cancel{\neg S_p}$

Disable the UNSAT core
instead of the entire assignment

DPLL(T) Basic + optimization

```
Algorithm CDCL():
    while true do
        α ← α ∪ {Choose()}
        while BCP() = conflict do
            backtrack-level ← AnalyzeConflict()
            if backtrack-level < 0 then
                return false
            else
                BackTrack()
            end
        end
        if α is full assignment then
            return true
        end
    end
```

```
Algorithm CDCL(T)():
    while true do
        α ← α ∪ {Choose()}
        while BCP() = conflict do
            backtrack-level ← AnalyzeConflict()
            if backtrack-level < 0 then
                return false
            else
                BackTrack()
            end
        end
        if α is full assignment then
            if T-Solver(α) then return true
            AddClauses()
        end
    end
```

Theory of EUF

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

ϕ is T-satisfiable if there exists a structure satisfies both ϕ and T

$$\forall x.\ x = x \qquad \text{(Reflexivity)}$$

$$\forall x.\ \forall y.\ x = y \implies y = x \qquad \text{(Symmetry)}$$

$$\forall x.\ \forall y.\ \forall z.\ x = y \land y = z \implies x = z \qquad \text{(Transitivity)}$$

$$\forall \bar{x}, \bar{y}.\ \bigwedge_{i=1}^{n} x_i = y_i \implies f(\bar{x}) = f(\bar{y}) \qquad \text{(Functional Congruence)}$$

Animations of deciding EUF

Exercises of EUF

$$f(a, b) = a \wedge f(f(a, b), b) \neq a$$

UNSAT

$$a = b \wedge b = c \wedge g(f(a), b) = g(f(c), a) \wedge f(a) \neq b$$

SAT

A simple application of EUF

```
int fun1(int y) {
  int x, z;
  z = y;
  y = x;
  x = z;

  return x * x;
}
```

$$
\begin{gathered}
z = y \wedge \\
y1 = x \wedge \\
x1 = z \wedge \\
ret1 = x1 * x1
\end{gathered}
$$

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

$$ret2 = y * y$$

$$
\begin{gathered}
z = y \wedge \\
y1 = x \wedge \\
x1 = z \wedge \\
ret1 = x1 * x1 \wedge \\
ret2 = y * y \wedge \\
not\ (ret1 = ret2)
\end{gathered}
$$

$$
\begin{gathered}
z = y \wedge \\
y1 = x \wedge \\
x1 = z \wedge \\
ret1 = f(x1, x1) \wedge \\
ret2 = f(y, y) \wedge \\
not\ (ret1 = ret2)
\end{gathered}
$$

A simple application of EUF

```
int fun1(int y) {
  int x, z;
  z = y;
  y = x;
  x = z;

  return x * (x+1);
}
```

$$z = y \wedge$$
$$y1 = x \wedge$$
$$x1 = z \wedge$$
$$t1 = x1 + 1 \wedge$$
$$ret1 = x1 * t1$$

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

$$t2 = y + 1 \wedge$$
$$ret2 = t2 * y$$

"partially interpreted functions"

$$\forall x, y \;\; f(x, y) = f(y, x)$$

$$z = y \wedge$$
$$y1 = x \wedge$$
$$x1 = z \wedge$$
$$t1 = g(x1,1) \wedge$$
$$ret1 = f(x1, t1) \wedge$$
$$t2 = g(y, 1) \wedge$$
$$ret2 = f(t2, y) \wedge$$
$$not\ (ret1 = ret2)$$

Another way of handling UF

Get rid of uninterpreted functions (UFs) by rewriting

$$\varphi^{EUF} \longrightarrow \varphi^{E}$$

Ackermann Reduction

$(x_1 \neq x_2) \vee (F(x_1) = F(x_2)) \vee (F(x_1) \neq F(x_3))$

Flatten constraints: $(x_1 \neq x_2) \vee (f_1 = f_2) \vee (f_1 \neq f_3)$

Functional consistency constraints:

$$(x_1 = x_2 \Rightarrow f_1 = f_2) \wedge$$
$$(x_1 = x_3 \Rightarrow f_1 = f_3) \wedge$$
$$(x_2 = x_3 \Rightarrow f_2 = f_3)$$

Two possible encodings:

(**Satisfiability** checking) functional consistency constraints $\wedge$ flatten constraints

(**Validity** checking) functional consistency constraints $\Longrightarrow$ flatten constraints

Difference logic

- **Linear constraints**

$$x \geq y + c$$

$$x \geq c$$

$$(s_i, f_i)$$

Job Scheduling

- N jobs, $\mathrm{T_i}$ is execution time for job I
- Need to finish all jobs before T
- Some jobs cannot execute at the same time

$$f_i \geq s_i + T_i$$

$$s_i \geq f_j \vee s_j \geq f_i$$

$$T \geq f_i$$

Theory of Arrays

John McCarthy, 1962

- Model arrays as functions $read(a, i)$ $write(a, i, v)$
- Read-over-write axioms

$$\forall\, a, i, j, v{:}\ i = j \Rightarrow read(write(a, i, v), j) = v$$
$$\forall\, a, i, j, v{:}\ i \neq j \Rightarrow read(write(a, i, v), j) = read(a, j)$$

$$write(a, i, v)[i] = v$$
$$write(a, i, v)[j] = a[j] \text{ for } i \neq j$$

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

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

$$
\begin{aligned}
nat &:= succ(pred : nat) \mid zero; \\
list &:= cons(car : tree,\ cdr : list) \mid null; \\
tree &:= node(children : list) \mid leaf(data : nat);
\end{aligned}
$$

- **Constructor, Selector, Tester**
- **function symbol $\Leftrightarrow$ constructor, selector**
- **predicate symbol $\Leftrightarrow$ each tester**

Example: *list* **of** *int*

- Constructors: $cons : (int, list) \longrightarrow list$, $null : list$
- Selectors: $car : list \longrightarrow int$, $cdr : list \longrightarrow list$
- Testers: is_cons, is_null

$$
\begin{aligned}
&\forall x_1, \dots, x_n.\ is_C(C(x_1, \dots, x_n)) \approx \mathsf{true} \\
&\forall x_1, \dots, x_n.\ is_{C'}(C(x_1, \dots, x_n)) \approx \mathsf{false} \\
&\forall x_1, \dots, x_n.\ S_C^{(i)}(C(x_1, \dots, x_n)) \approx x_i \qquad \text{for all } i = 1, \dots, n \\
&\forall x_1, \dots, x_n.\ S_{C'}^{(i)}(C(x_1, \dots, x_n)) \approx t_{C'}^i \qquad \text{for all } i = 1, \dots, n'
\end{aligned}
$$

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

Barrett, et al. *An Abstract Decision Procedure for a Theory of Inductive Data Types*, JSAT 2007

Many other theories

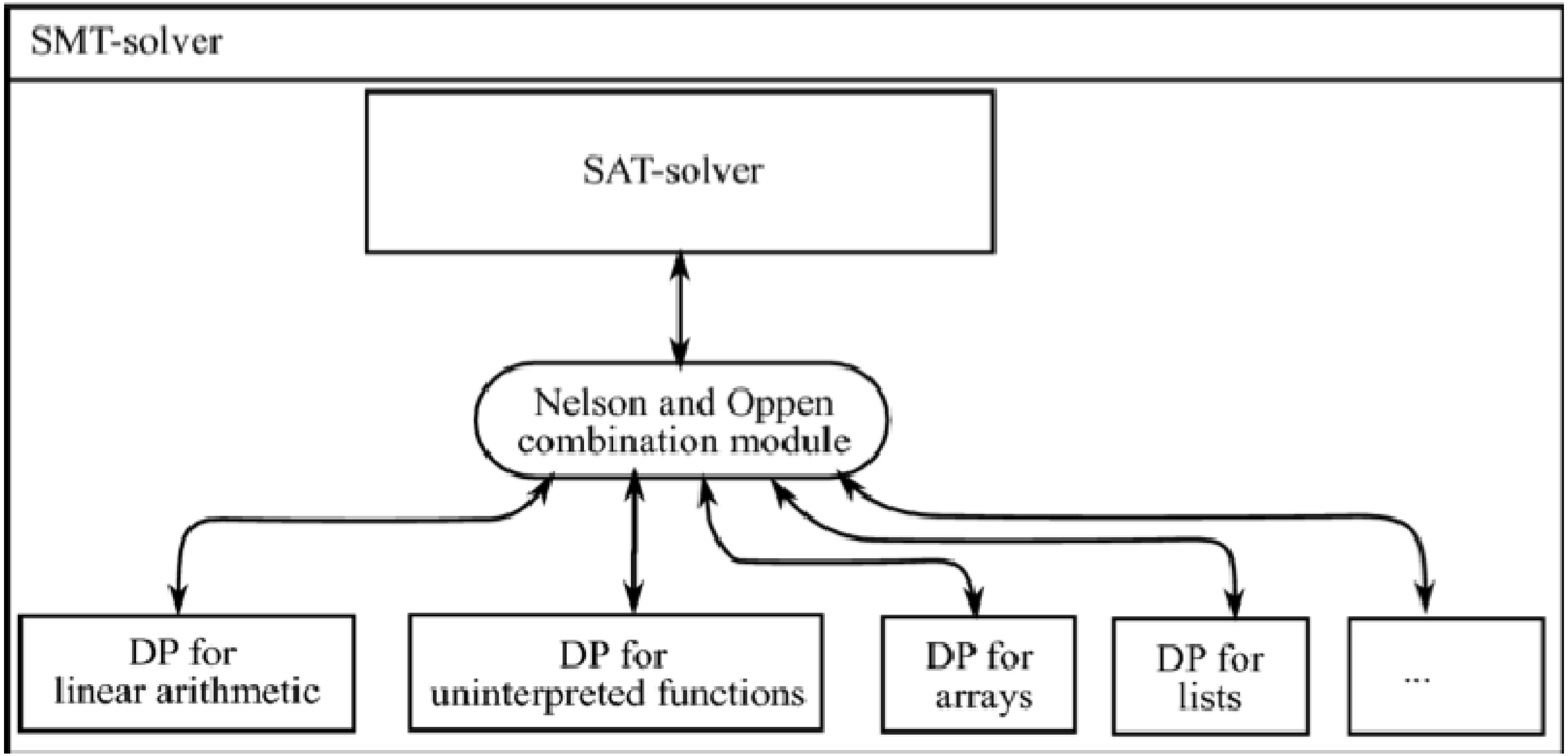
- Theory of string
- Theory of bit vector
- Theory of linear arithmetic
- Theory of non-linear arithmetic
- Theory of integer linear arithmetic
- …

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 Σ_1 and Σ_2**
- **The combined theory $T_1 \oplus T_2$ has**
  - signature $\Sigma_1 \cup \Sigma_2$ and
  - the union of their axioms.
- **Let ϕ 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 ϕ with a new auxiliary variable a**
2. **Constrain the formula with a = ϕ**

$$x_1 \leq f(x_1)$$

$$\underbrace{x_1 \leq a_1 \wedge a_1 = f(x_1)}_{\textit{Pure}\text{ expressions, shared variables}}$$

Uninterpreted Functions

Arithmetic

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
- ϕ is satisfiable $\Leftrightarrow F_1 \wedge \cdots \wedge F_n$ 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 $F_1, \ldots, F_n$ are SAT)