SRILAB

FastSMT

Learning to Solve SMT Formulas

Mislav Balunović, Pavol Bielik, Martin Vechev

fastsmt.ethz.ch

ETH zürich

SMT Solvers and Formulas

SMT Theories

Integers
Reals
Arrays
BitVectors
Strings
...

Find an assignment to all free variables in Φ such that Φ evaluates to true

Φ First-order logic formula → **SMT Solver** → SAT / UNSAT / UNKNOWN

Applications

SW & HW Verification
Neural Network Verification
Program Synthesis
Symbolic Execution
Program Analysis
...

SMT Formula Example

$(\sin(x))^3 = \cos(\log(y) \cdot x) \vee b \vee -x^2 \geq 2.3y) \wedge (\neg b \vee y < -34.4 \vee \exp(y) > \frac{y}{x})$ where $b \in \mathbb{B}, x, y \in \mathbb{R}$

Our Work

Replace the handcrafted strategies by learning to solve the SMT formulas

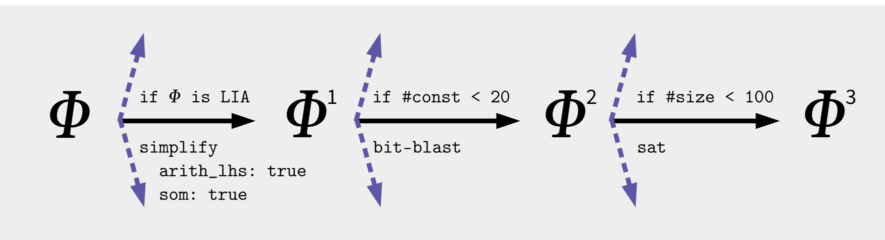
Φ First-order logic formula → **Learned Strategies** → SAT / UNSAT / UNKNOWN

such that:

1) Learning does not assume any prior knowledge (initially no formula is solvable)
2) The learned strategies can be integrated into existing state-of-the-art SMT solvers

How are Formulas Solved?

Formulas are solved by applying sequence of transformations until a base form (true/false) is reached



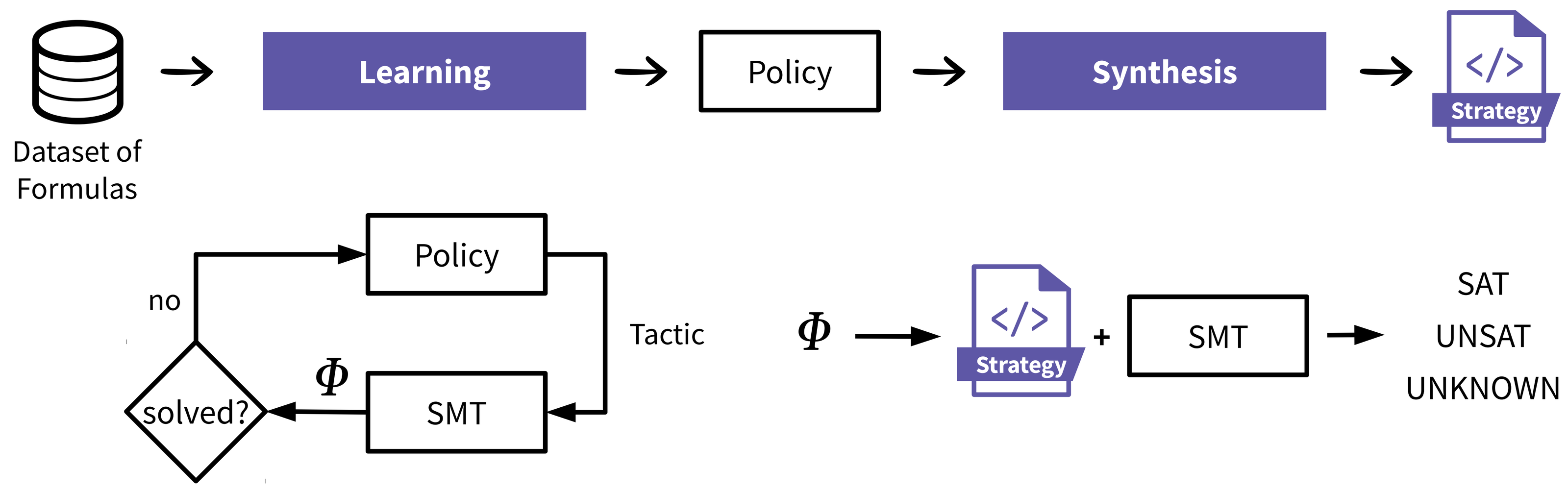
Constant Folding `x + 0 → x`

Gaussian Elimination `x = 1 ∧ y > x → y > 1`

Bit Blasting $x = 5 \rightarrow x_3 = 0 \wedge x_2 = 1 \wedge x_1 = 0 \wedge x_0 = 1$

To be **fast**, **SMT solvers rely on a set of handcrafted strategies** which define how to transform given formula

Our Approach



Learn a policy to select next transformation(tactic)

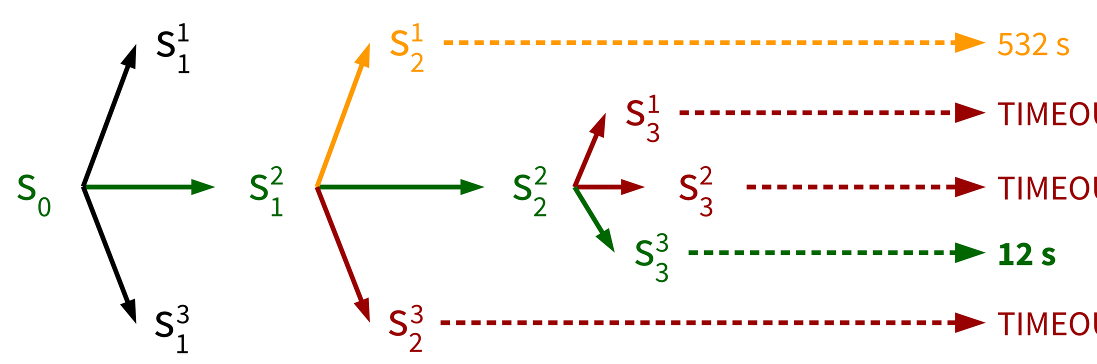
Synthesize a program from the learned policy and use it to guide the SMT solver

Learning

Given a formula Φ find sequential strategies that are fast at solving Φ

Phrase the learning problem as a tree search over sequences of tactics

Different formulas often require different sequences of tactics in order to be solved fast

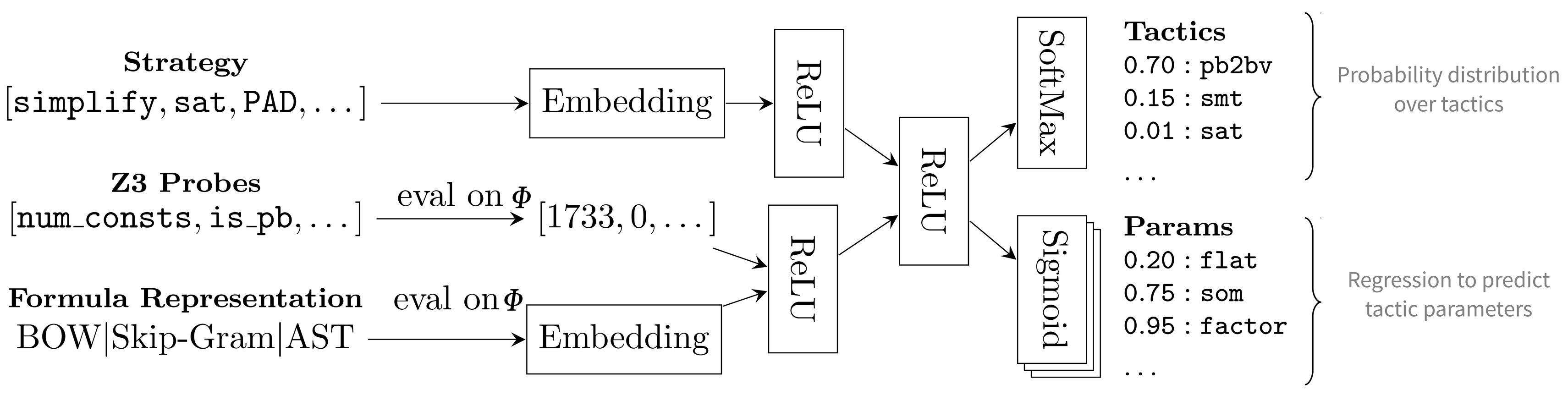


(State) s = Φ x Tactics* (Action) a = Tactics x Params

Examples of Strategies

```
simplify with {arith_lhs:true, som:true}; norm_bounds; lia2pb; pb2bv; bit_blast; sat
simplify with {local_ctx: true}; sat; bit_blast; sat
```

Learn a policy to select next tactic



Loss = weighted average of cross-entropy loss + mean-square-error
tactics params

Training algorithm based on DAgger [1]

Algorithm 1: Iterative algorithm used to train policy π

Data: Formulas , Number of iterations N, Number of formulas to sample K, Exploration rates β, Exploration policy $\pi_{explore}$ (e.g., random policy)

Result: Trained policy π, Explored strategies

1. $\mathcal{D} \leftarrow \emptyset$; $\leftarrow \emptyset$; $\pi \leftarrow$ policy initialization
2. **for** $i = 1$ **to** N **do**
3. $\hat{\pi} \leftarrow \beta_i \pi + (1 - \beta_i)\pi_{explore}$ ▷ policy $\hat{\pi}$ explores with probability $(1 - \beta_i)$
4. $\mathcal{Q} \leftarrow \mathcal{Q} \cup$ Top K most likely strategies for each formula in according to $\hat{\pi}$
5. $\mathcal{D} \leftarrow \mathcal{D} \cup$ Extract training dataset from strategies
6. $\pi \leftarrow$ Retrain model π on $\mathcal{D}$

[1] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. AISTATS'11

Synthesis

Synthesize an interpretable program with branches that selects a strategy that performs best on a dataset of formulas

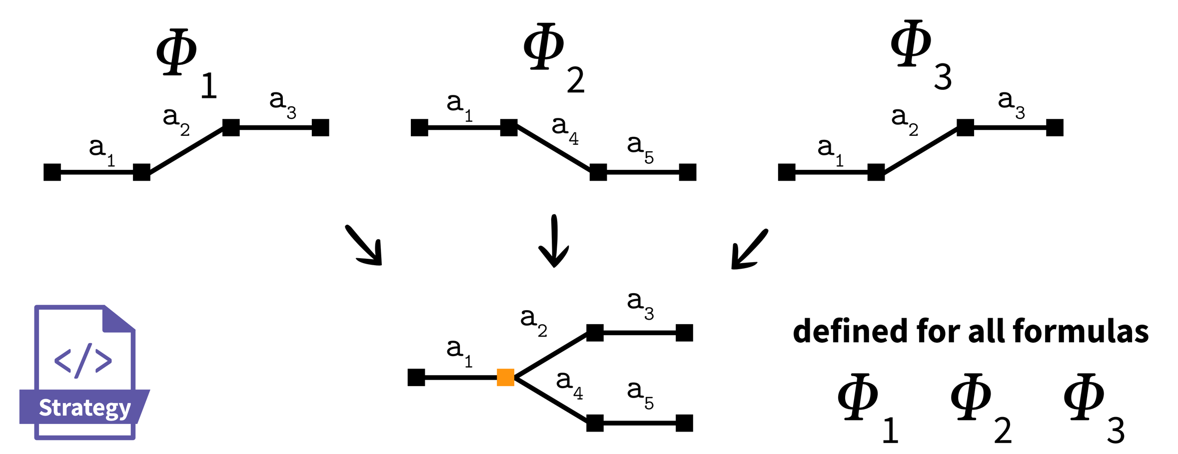
✔avoids overhead of running the policy ✔enables integration with state-of-the-art SMT solvers

Sequential Strategies

Obtained by running the learned policy on a dataset of formulas

Strategy with Branches

Single strategy with synthesized branch for each state with multiple outgoing edges



Decision Tree Learning

Synthesize predicates for each node in the tree in a top-down fashion

Multi-label entropy of a dataset of formulas

$$H(\mathcal{F}) = -\sum_{q \in \mathcal{Q}} p(q)\log(p(q)) + (1 - p(q))\log(1 - p(q))$$

p(q) denotes ratio of formulas solved by strategy q

Cost associated with branch *b*

$$cost(b, \mathcal{F}_{\texttt{true}}, \mathcal{F}_{\texttt{false}}) = \frac{|\mathcal{F}_{\texttt{true}}|}{|\mathcal{F}|} H(\mathcal{F}_{\texttt{true}}) + \frac{|\mathcal{F}_{\texttt{false}}|}{|\mathcal{F}|} H(\mathcal{F}_{\texttt{false}})$$

$H(F_{false})$ is entropy of formulas for which branch predicate evaluates to false

Example

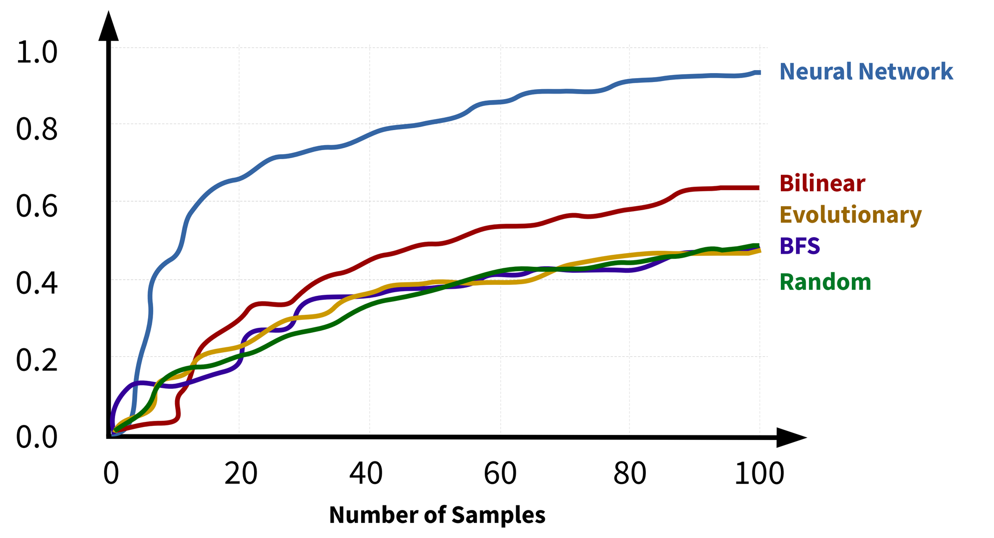
	Strategy Runtimes Φ_1	Φ_2	Φ_3	Formula Measures Φ_1	Φ_2	Φ_3	Candidate Branch	Cost
$a_1; a_2; a_3$	10	100	100	200	250	30	`if true then a2 else _`	0.276
$a_1; a_4; a_5$	30	TIMEOUT	20	num_expr			`if num_expr > 100 then a2 else a4`	0.2

Syntax of Strategy Language used by Z3 solver

```
(Strategy)     q      ::=  t | q; q | if p then q else q | q else q |
                           repeat q,c | try q for c | using t with params
(Predicates)   p      ::=  p ∧ p | p ∨ p | expr ⋈ expr
(Expressions)  expr   ::=  c | probe | expr ⊕ expr
(Constants)    c      ∈    Consts = ℚ
(Probes)       probe  ::=  Probe → ℚ,   Probe = { num_consts, is_pb, ... }
(AOperators)   ⊕      ::=  + | − | ∗ | /
(BOperators)   ⋈      ::=  > | < | ≥ | ≤ | = | ≠
(Parameter)    param  ::=  (Param, ℚ),   Param = { hoist_mul, flat, som, ... }
(Parameters)   params ::=  ϵ | param; params
```

Evaluation: Learning

Runtime Ratio of Best Sampled Strategy vs Best Known Strategy



Find fastest strategy for each formula

2.6x
90th percentile

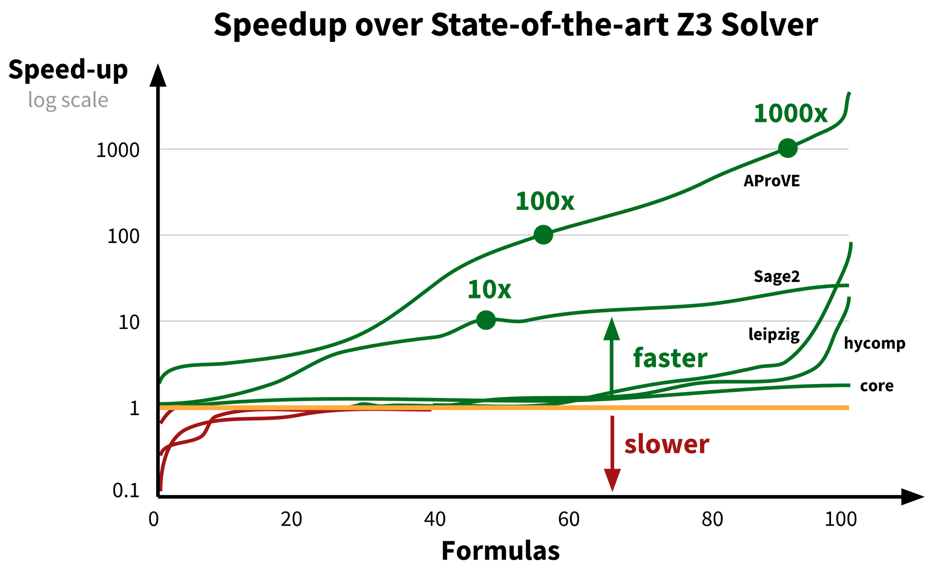
31x
50th percentile

310x
10th percentile

90th percentile denotes that 90% of the formulas have speedup at-least 2.6x

Evaluation: Synthesis

Speedup over State-of-the-art Z3 Solver



Synthesized Strategy vs Handcrafted Z3 Strategy

	Formulas Solved: Both	Only Z3	Only Learned	None	Speedup Percentile: 90^{th}	50^{th}	10^{th}
leipzig	55	5	1	7	1.4×	9.9×	21.7×
Sage2	2488	200	705	3051	0.8×	1.2×	3.1×
core	270	0	0	0	1.2×	1.3×	1.8×
hycomp	1547	93	112	230	0.4×	1.1×	2.3×
AProVE	1365	76	112	159	3.2×	65.1×	860.8×
Total	53.6%	3.5%	8.7%	34.1%	1.4×	15.7×	178.0×

Strategy Generalization

Training	Testing		Average Speedup
10 seconds	10 seconds	→	31x
10 seconds	10 minutes	→	32x

Thirty-second Conference on Neural Information Processing Systems
2nd - 8th December 2018, Montreal