Machine Learning for SAT Solvers

Vijay Ganesh
University of Waterloo, Canada

Monday Aug 26, 2019
Waterloo ML + Verification + Security Workshop, Canada

THE BOOLEAN SATISFIABILITY PROBLEM

- A **literal** p is a Boolean variable x or its negation $\neg x$. A **clause** C is a disjunction of literals. E.g., $(x_2 \lor \neg x_{41} \lor x_{15})$. A k-**CNF** formula is a conjunction of m clauses over n variables, with k literals per clause. An **assignment** is a mapping from variables to True/False**.** A **unit clause** C has exactly one unbound literal, under a partial assignment
- **Boolean SATisfiability problem**: given Boolean formulas in k-CNF, decide whether they are satisfiable. The challenge is coming up with an efficient procedure.
- A **SAT Solver** is a computer program that solves the SAT problem.
- The challenge for SAT solver developer is:
    - Develop a solver that works efficiently for a very large class of practical applications. Solvers must produce solutions for satisfiable instances, and proofs for unsatisfiable ones. Solvers must be extensible. Perhaps, the most important problem is to understand and explain why solvers work well even though the problem is NP-complete.

TALK OUTLINE

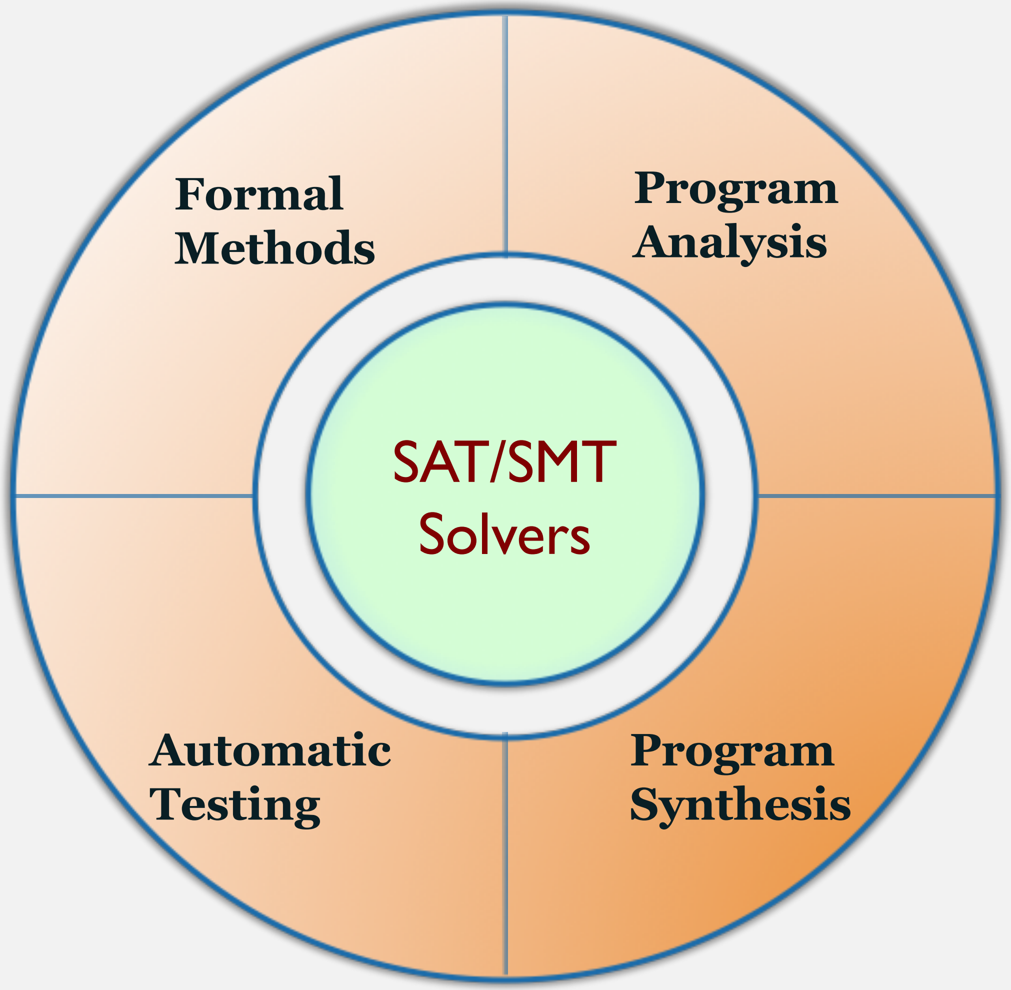
- Part I
    - Context and motivation for the Boolean SAT problem
- Part II
    - DPLL and CDCL SAT solvers
- Part III
    - Key research questions and insights
- Part IV
    - Heuristics are optimization engines, and machine learning (ML) for SAT. MapleSAT series of SAT solvers [LG+15, LG+16, LG+17, LG+18]
- Part V
    - Conclusions and takeaways
- Part VI
    - Logic-guided ML

PART I

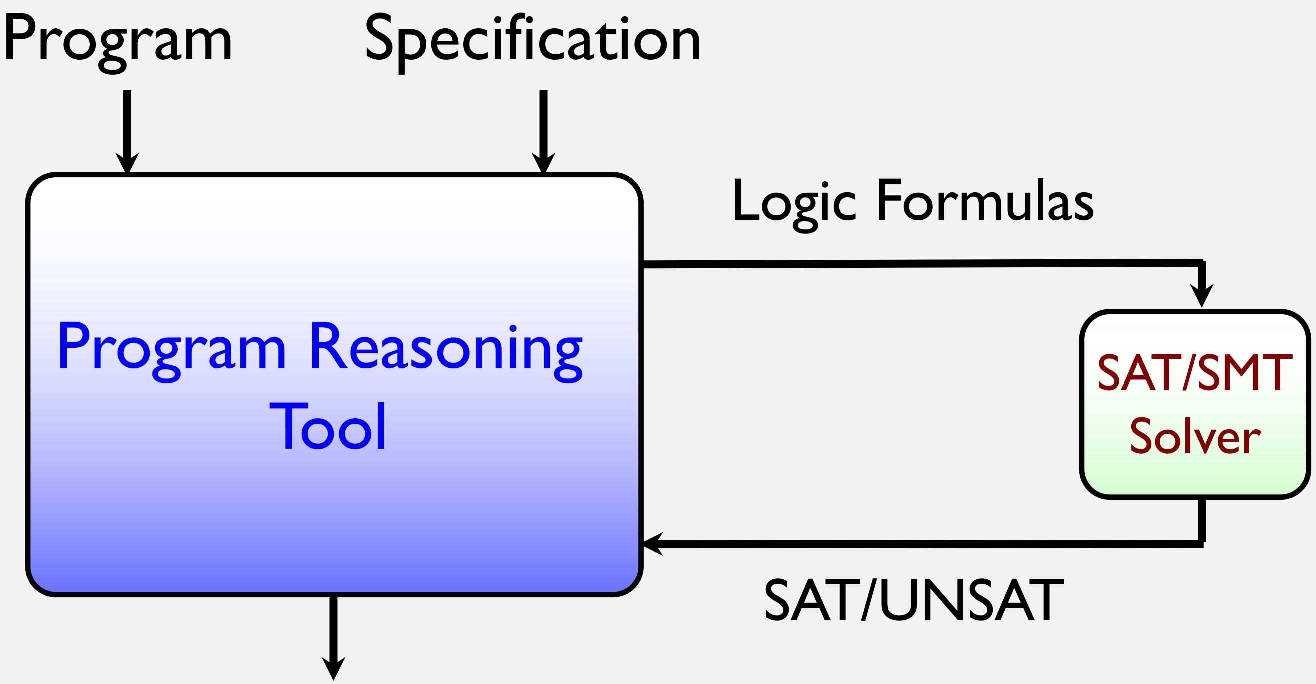
CONTEXT AND MOTIVATION

WHY SHOULD YOU CARE ABOUT SAT SOLVERS?

SOFTWARE ENGINEERING AND SAT/SMT SOLVERS
AN INDISPENSABLE TACTIC FOR ANY STRATEGY

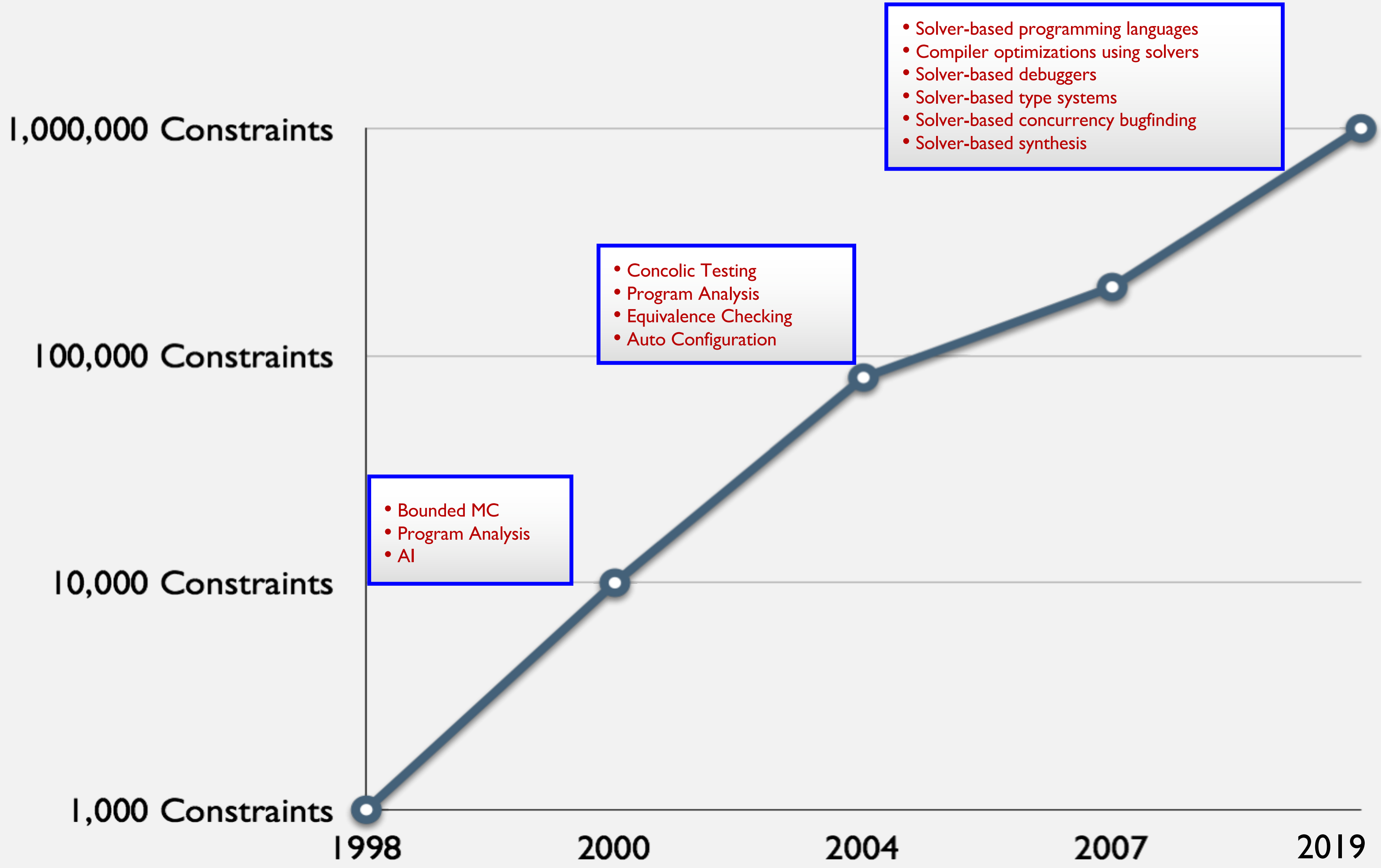


SOFTWARE ENGINEERING USING SOLVERS
ENGINEERING, USABILITY, NOVELTY

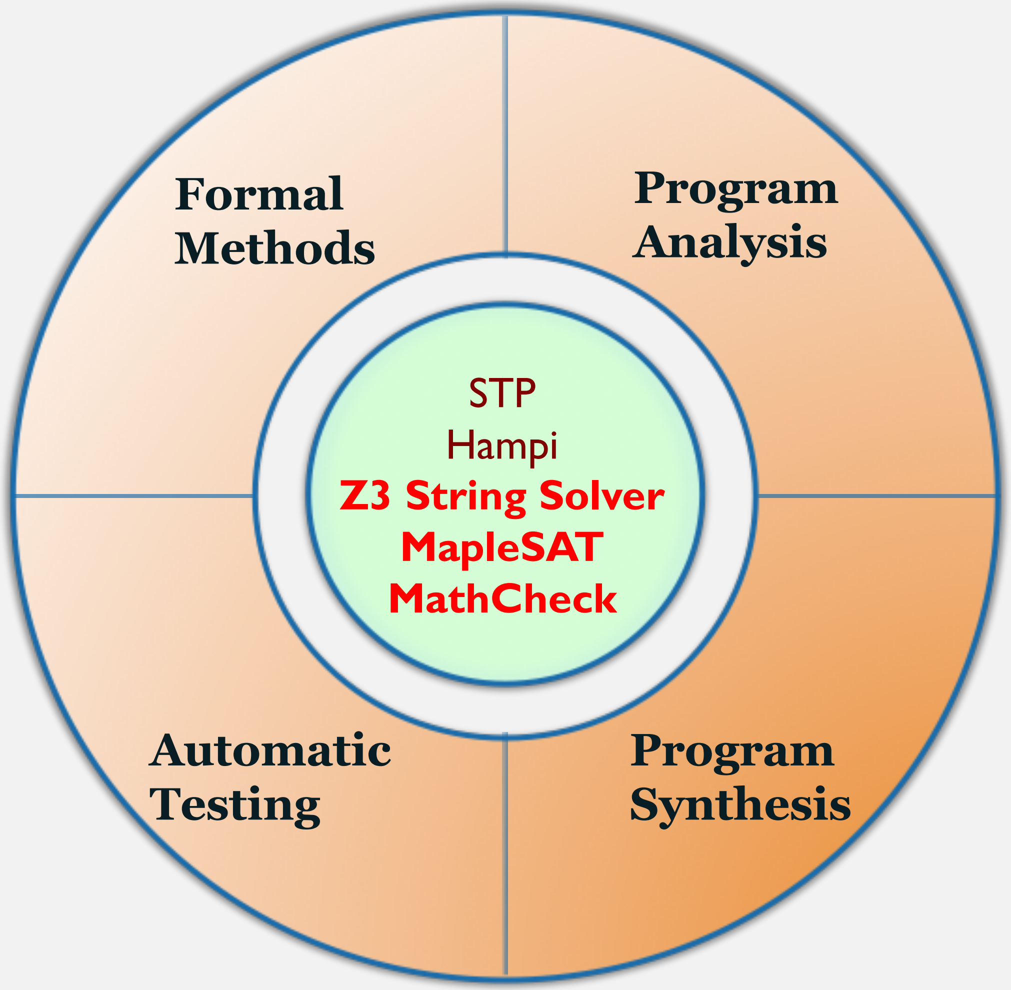


Program is correct?
or Generate Counterexamples (test cases)

SAT/SMT SOLVER RESEARCH STORY
A 1000X+ IMPROVEMENT



IMPORTANT CONTRIBUTIONS
AN INDISPENSABLE TACTIC FOR ANY STRATEGY



PART II

DPLL AND CDCL SOLVER ALGORITHMS

DPLL SAT SOLVER ARCHITECTURE (1958)
THE BASIC BACKTRACKING SAT SOLVER

DPLL(Θ_{cnf}, assign) {

Propagate unit clauses;

if *"conflict"*: **return** FALSE;

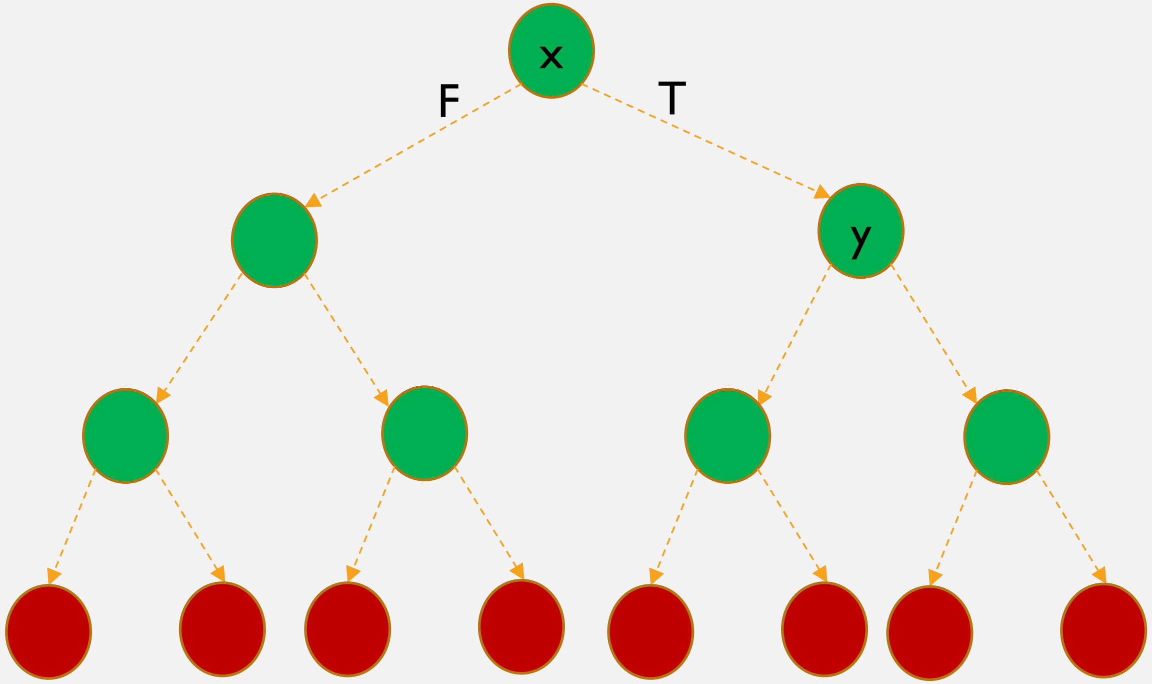
if *"complete assign"*: **return** TRUE;

"pick decision variable x";

return

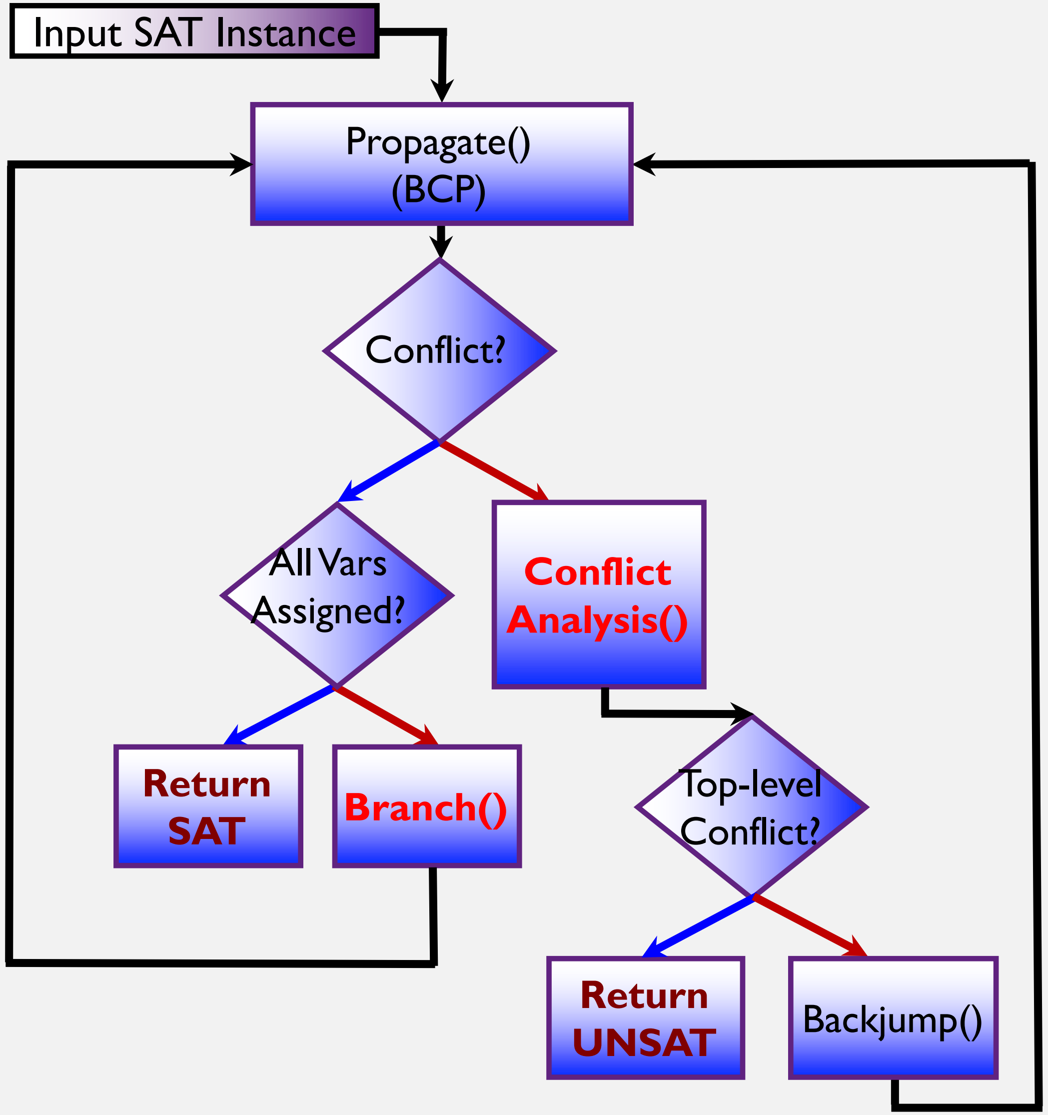
DPLL($\Theta_{cnf}\big|_{x=0}$, assign[x=0]) ||
DPLL($\Theta_{cnf}\big|_{x=1}$, assign[x=1]);

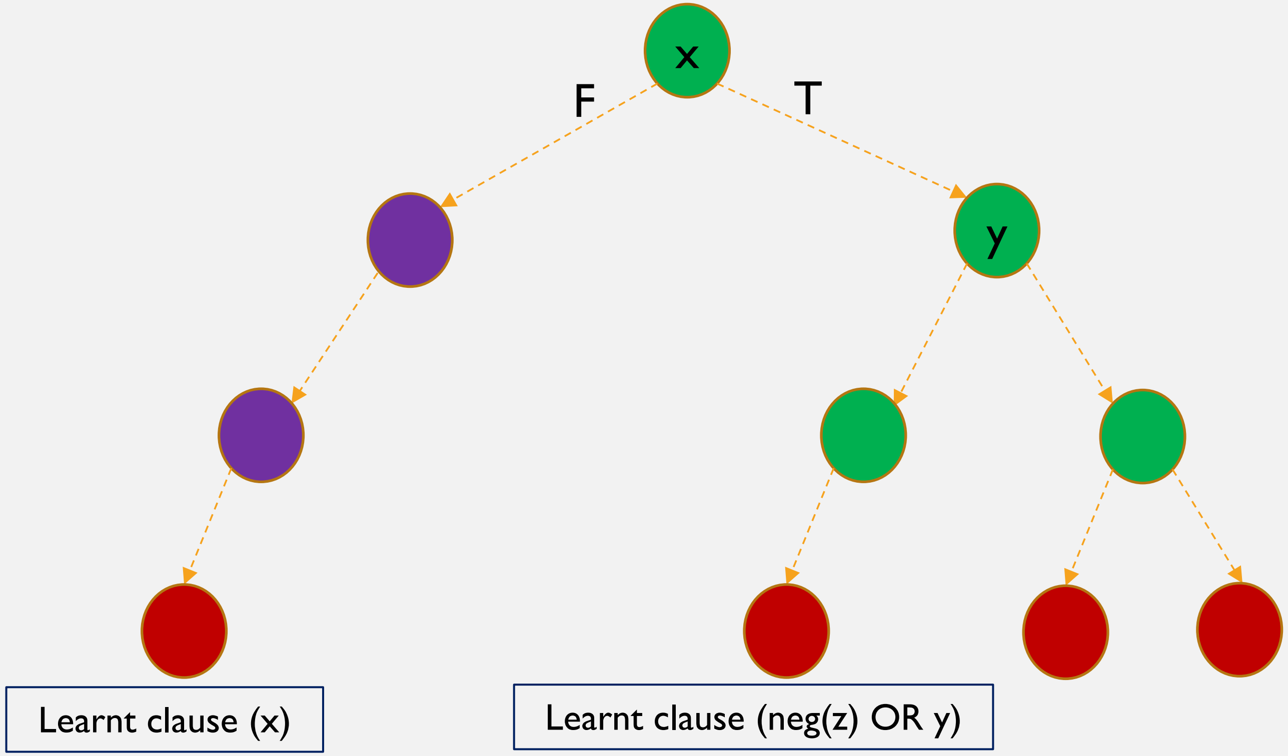
}



DPLL stands for Davis, Putnam, Logemann, and Loveland

MODERN CDCL SAT SOLVER ARCHITECTURE OVERVIEW





PART III

RESEARCH QUESTIONS

WHY ARE SAT SOLVERS EFFICIENT AT ALL?

RESEARCH QUESTIONS AND RESULTS
WHY ARE SAT SOLVERS EFFICIENT AT ALL?

Understanding the efficacy of solvers (practical proof systems)

Proof complexity

- CDCL SAT solvers are polynomially-equivalent to merge resolution
- Proof complexity of SMT solvers [RKG18]

Parameterized complexity

- Introduced the merge parameter as a basis for upper bound analysis [ZG+18]
- Merge as a feature for machine learning based clause deletion

Machine learning based solver design

- Introduced the idea of 'solver as a collection of machine learning based optimization engines' [LG+16,LG+17,LG+18]
- Successfully used it develop new ML-based branching and restart policies in MapleSAT

THE CONTEXT
PARAMETERIZED PROOF-COMPLEXITY FOR FORMAL METHODS

General resolution — The rule is form of modus ponens. Proof is a directed acyclic graph (DAG).

$$\frac{(x_1 \vee \cdots \vee x_n) \quad (\neg x_n \vee y_1 \ldots \vee y_m)}{(x_1 \vee \cdots \vee x_{n-1} \vee y_1 \ldots \vee y_m)}$$

Merge resolution — Derived clauses have to share literals to apply rule. Proof is a DAG.

$$\frac{(x_1 \vee \cdots \vee x_n) \quad (\neg x_n \vee \cdots x_{n-1})}{(x_1 \vee \cdots \vee x_{n-1})}$$

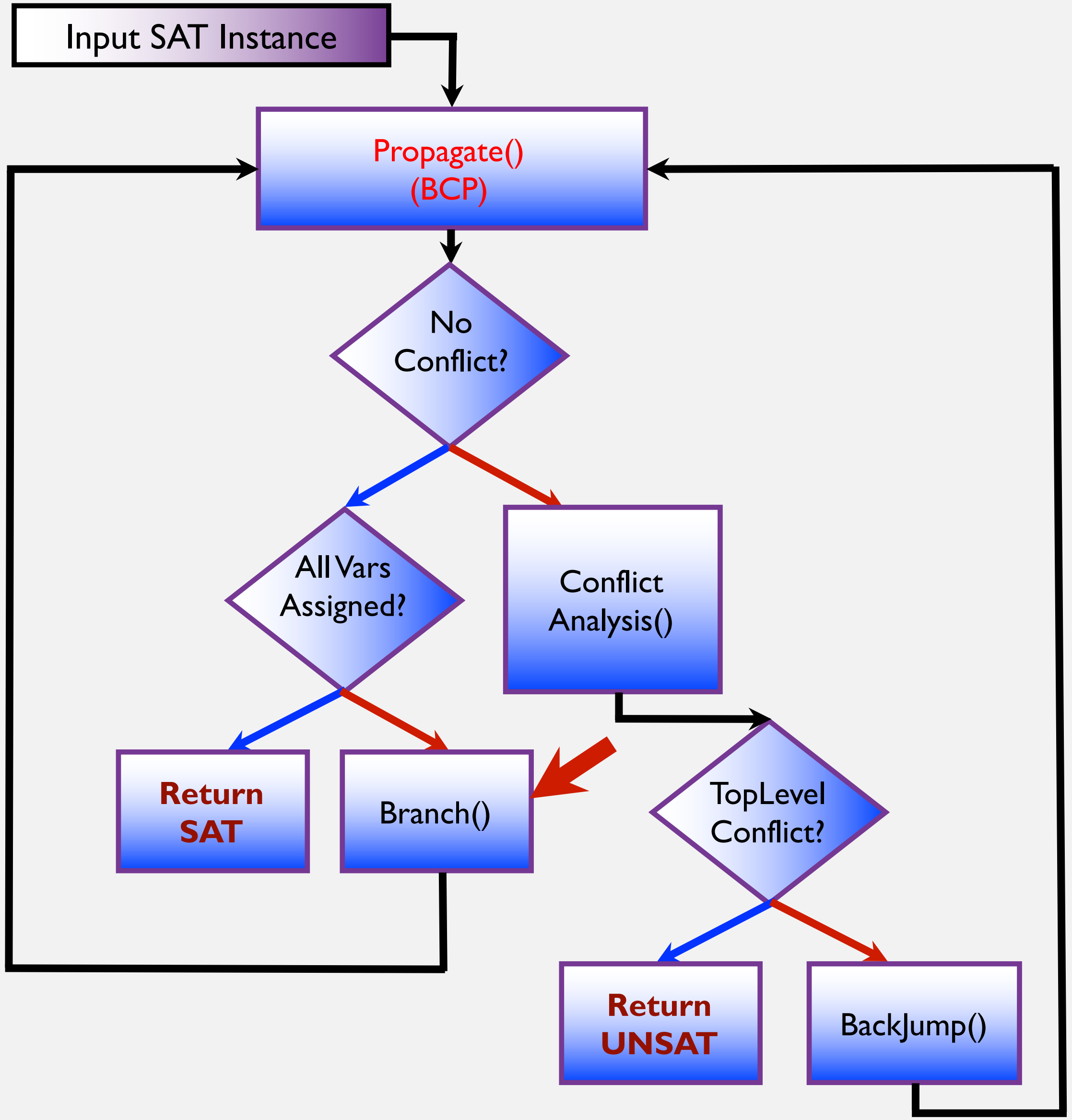
Unit resolution — One clause must be unit. Proof is a DAG.

$$\frac{(x_n) \quad (\neg x_n \vee y_1 \ldots y_m)}{(y_1 \vee \cdots \vee y_m)}$$

Tree resolution — Same rules as general resolution. Proof is a tree. Not allowed to reuse lemmas unlike DAG proofs.

HEURISTICS AS OPTIMIZATIONS PROCEDURES

MACHINE LEARNING FOR SOLVERS



- SAT solvers as a proof system that attempts to produce proofs for input unsatisfiable formulas in the shortest time possible
- In other words, certain sub-routines of a SAT solver implement proof rules (e.g., BCP implements the unit resolution rule),
- Other sub-routines aim to optimally select, schedule, or initialize proof rule application
- These optimization procedures operate in a data-rich environment, need to be adaptive and online
- Machine learning to the rescue!! Transforming solver design from "an art to a science"

PART IV

MACHINE LEARNING BASED BRANCHING HEURISTICS

PROBLEM STATEMENT: WHAT IS A BRANCHING HEURISTIC?
A METHOD TO MAXIMIZE LEARNING RATE

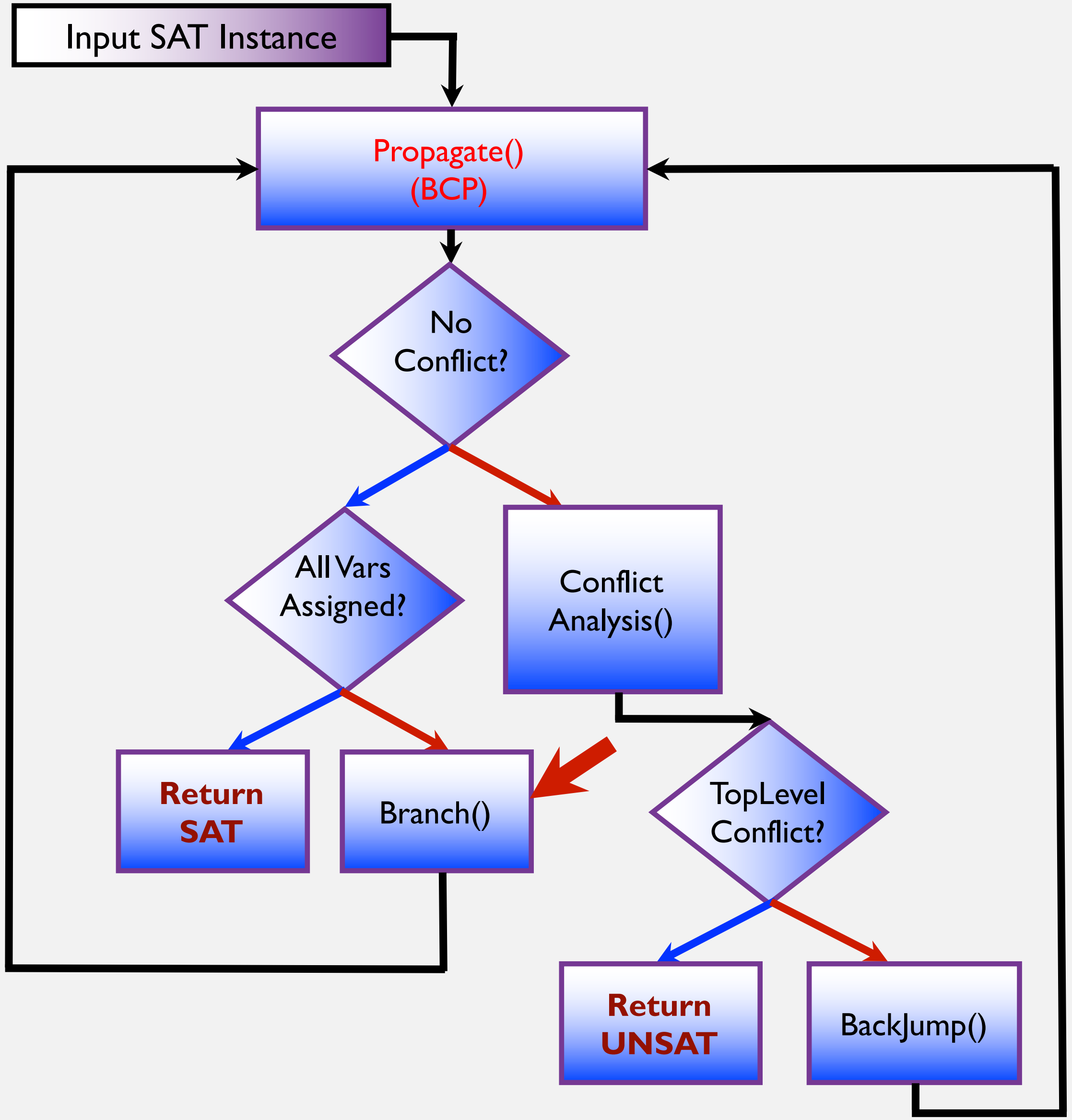
Question: What is a variable selection (branching) heuristic?

- A "dynamic" ranking function that ranks variables in a formula in descending order
- Re-ranks the variables at regular intervals throughout the run of a SAT solver
- We were unsatisfied with this understanding of VSIDS branching heuristic

Our experiments and results: [LG+15, LGPC16, LGPC+16, LGPC17, LGPC18]

- We studied 7 of the most well-known branching heuristics in detail
- Viewed branching as prediction engines that attempt to maximize global learning rate
- In turn led us to devise new ML-based branching that for the first time matched VSIDS

MODERN CDCL SAT SOLVER ARCHITECTURE
DECIDE(): VSIDS BRANCHING HEURISTIC



VSIDS (Variable State Independent Decaying Sum) Branching

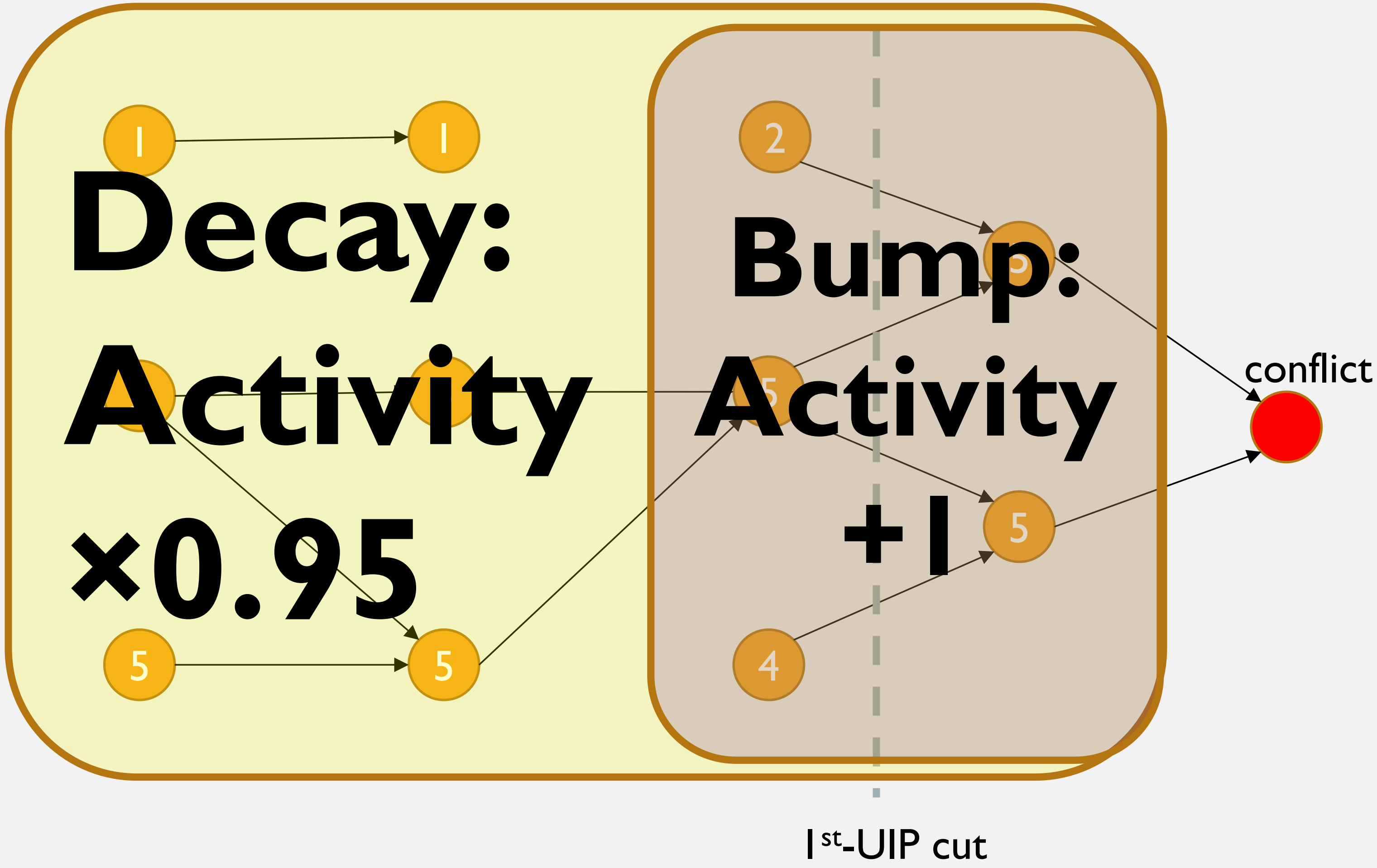
- Imposes dynamic variable order
- Each variable is assigned a floating-point value called activity
- Measures how "active" variable is in recent conflict clauses

VSIDS pseudo-code

- Initialize activity of all variables (vars) to 0

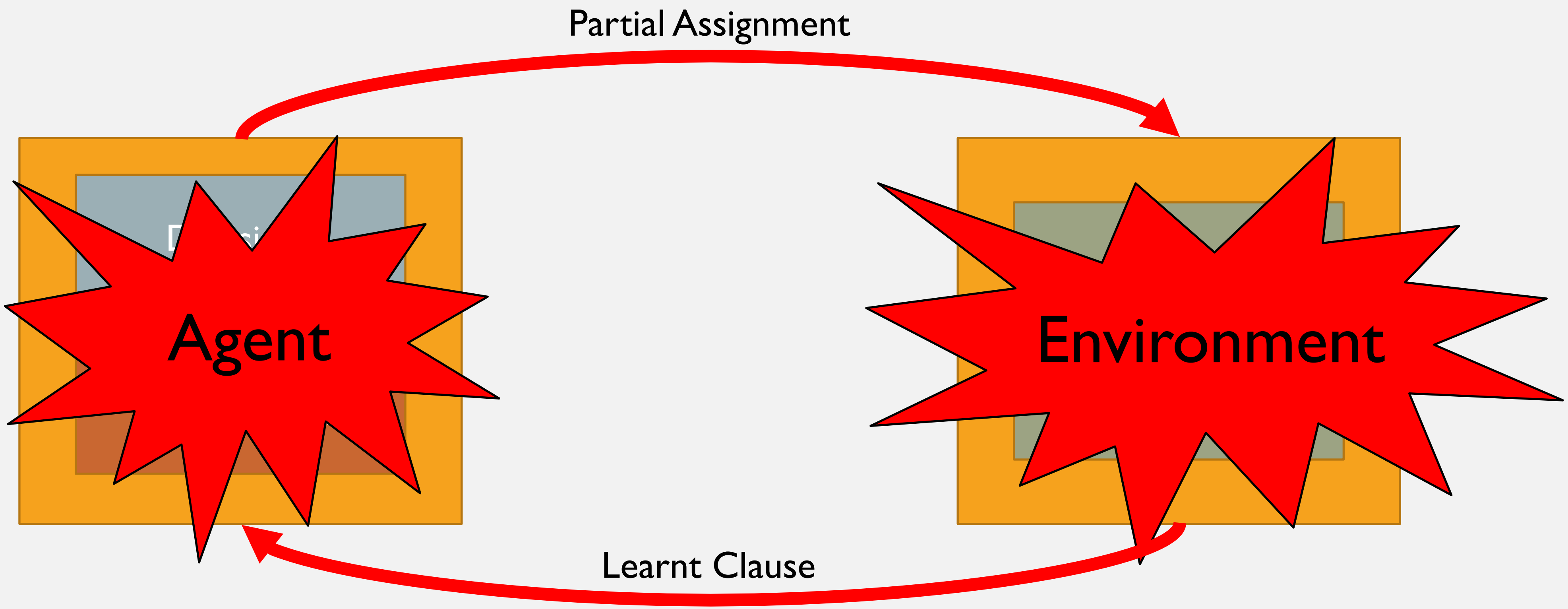
```
VSIDS() {
    Upon conflict
        * Bump activity of vars appearing on the conflict
          side of the implication graph
        * Decay activity of all vars by a constant c: 0 < c < 1
    Branch on unassigned var with highest activity
} //End of  VSIDS
```

VSIDS[1]: CONFLICT DRIVEN BRANCHING



1. *Chaff: Engineering an efficient SAT solver*. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. DAC 2001.

CDCL FEEDBACK LOOP



VSIDS: WHY BUMP AND DECAY?

```
for all variables v:
        activity[v] = 0

conflict:
        for all variables v between cut and conflict:
                activity[v] += 1
        for all variables v in learnt clause:
                activity[v] += 1
        for all variables v:
                activity[v] *= 0.95
```

Bump observation:
~12 times more likely to cause conflicts when branched on

Decay observation:
$bump_{t-1}0.95^1$
$+ bump_{t-2}0.95^2$
$+ bump_{t-3}0.95^3$
$+ \cdots$

More weight to recent bumps via exponential moving average

EXPONENTIAL MOVING AVERAGE

REINFORCEMENT LEARNING AND CDCL

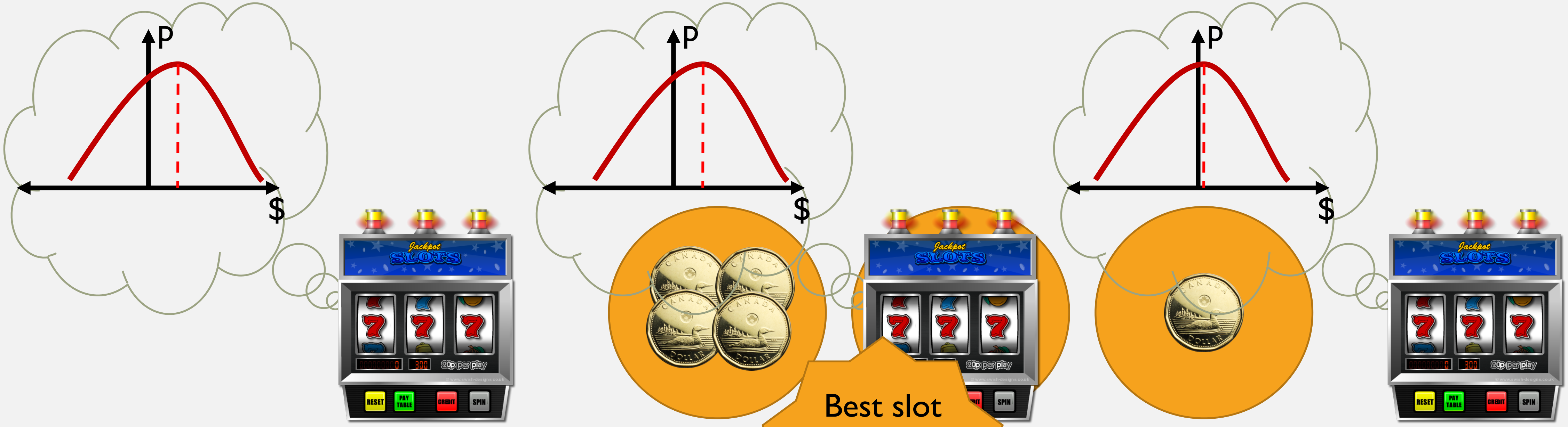
Reinforcement Learning

- Agent
- Environment
- Policy
- Action
- Estimated Reward (Q)
- Reward
- Exponential Moving Average

CDCL

- Branching Heuristic + BCP
- Clause learning
- Variable Ranking
- Decision
- Activity
- Bump
- Decay

MULTI-ARMED BANDIT PROBLEM



sample average = $1/3 \times \$4$ … $\$3$ + $1/3 \times \$1$

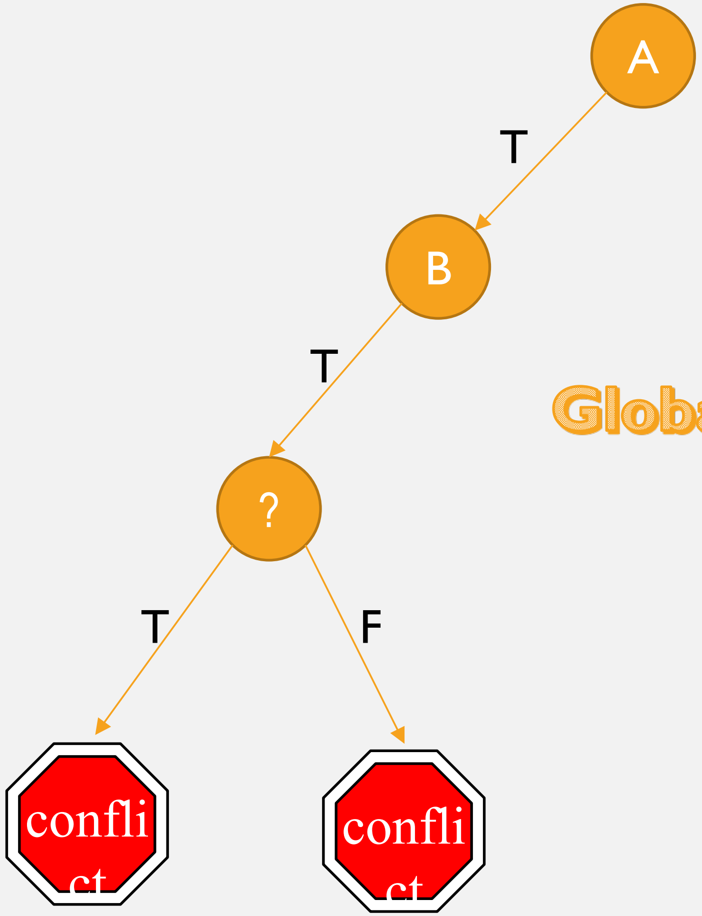
exponential moving average = $(1-\alpha)^2 \times \$4$ … $\times \$3$ + $(1-\alpha)^0 \times \$1$

Best slot machine to play (for now)

Less weight

More weight

WHAT IS A GOOD OBJECTIVE FOR BRANCHING?



$$\text{Global learning rate (GLR)} = \frac{\#\text{conflicts}}{\#\text{decisions}}$$

# of lemmas

of “cases”

PROBLEM STATEMENT: WHAT IS A BRANCHING HEURISTIC?
OUR FINDINGS

Finding 1: Global Learning Rate Maximization

Branching heuristics are prediction engines which predict variables to branch on that will maximize

Global Learning Rate (GLR) = (# of conflicts)/(# of decisions)

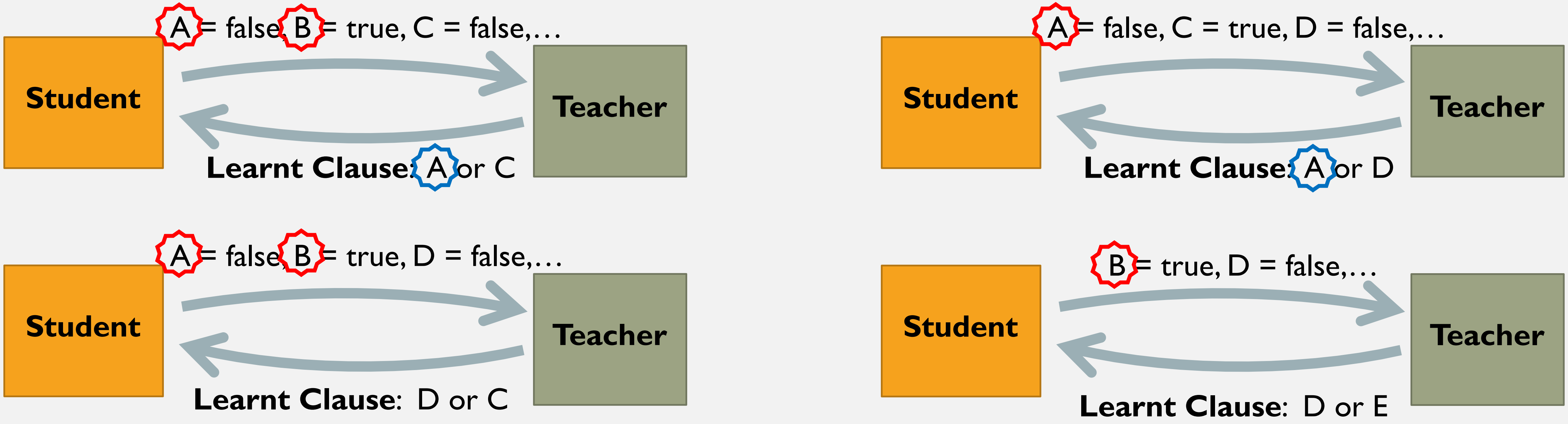
Finding 2: Branch on Conflict Analysis Variables ‘maximizes’ GLR

Successful branching heuristics focus on variables involved in ‘recent’ conflicts to maximize GLR. Reward variables that gave you a conflict

Finding 3: The Searchlight Analogy a la Exploitation vs. Exploration (multiplicative decay)

Focus on recent conflicts, maximize learning, then move on. One can use reinforcement learning for such a heuristic.

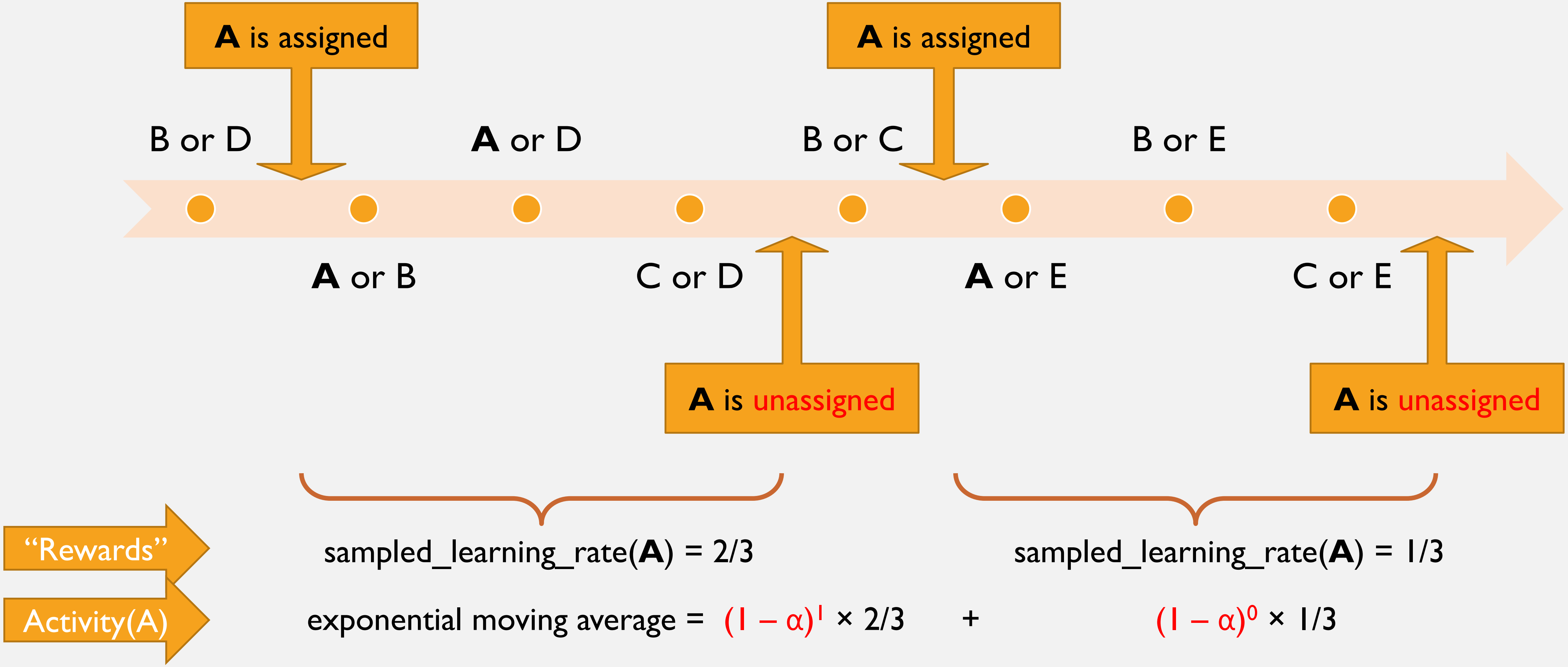
LEARNING RATE EXAMPLE



sampled_learning_rate(**A**) = 2/3

sampled_learning_rate(**B**) = 0/3

LEARNING-RATE BRANCHING (LRB) EXAMPLE



VSIDS

<u>The reward is a constant</u>

Every time a variable appears in a conflict analysis, its activity is additively bumped by a constant

<u>Exponential Moving Average (EMA) performed for all variables at the same time</u>

After each conflict, the activities of all variables are decayed

LRB

<u>The reward is not constant</u>

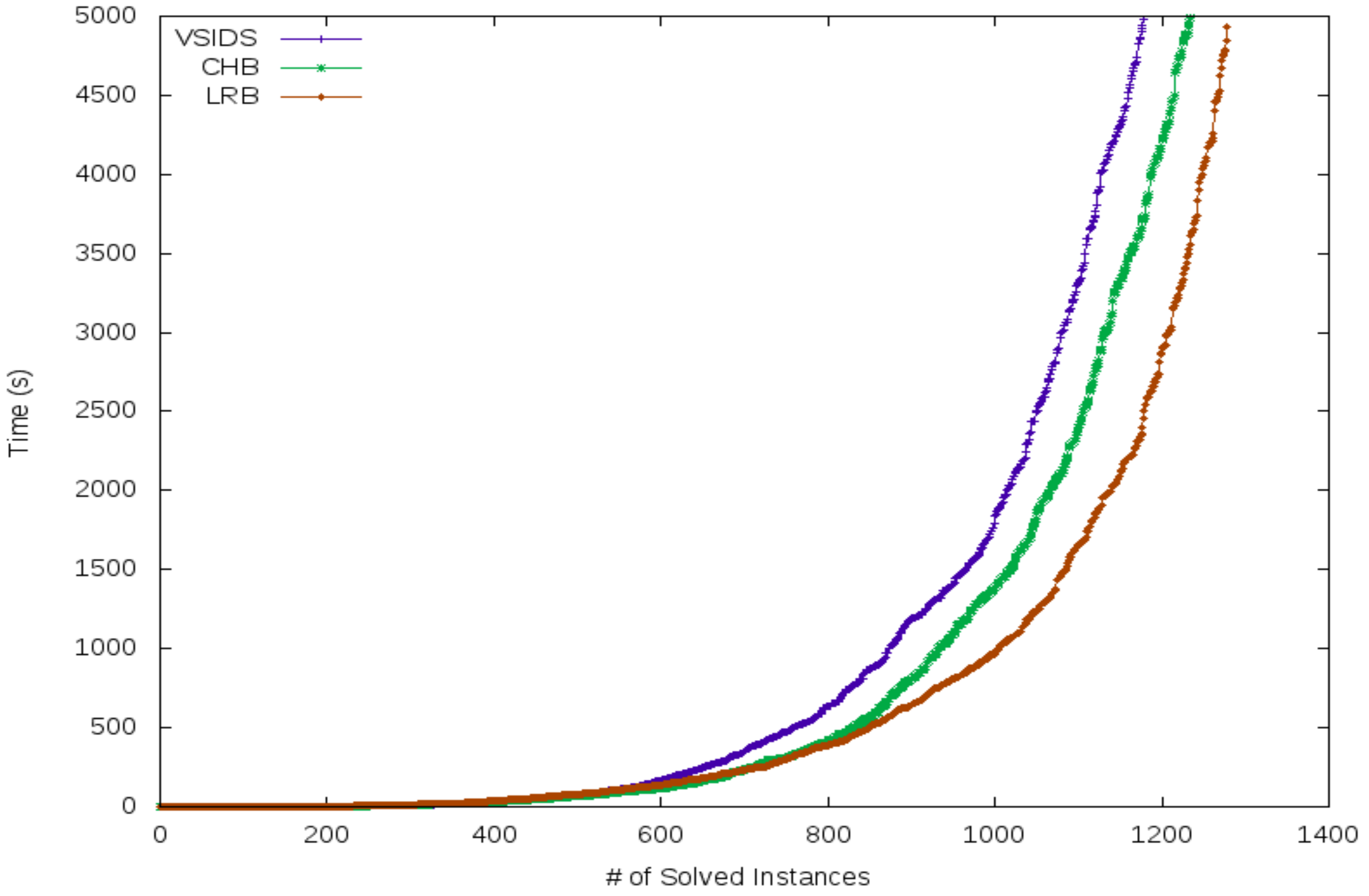
Every time a variable appears in a conflict analysis, the numerator of its learning rate reward is incremented. After each conflict, the denominator of each assigned variable's learning rate reward is incremented

<u>EMA performed only when variable goes from assigned to unassigned</u>

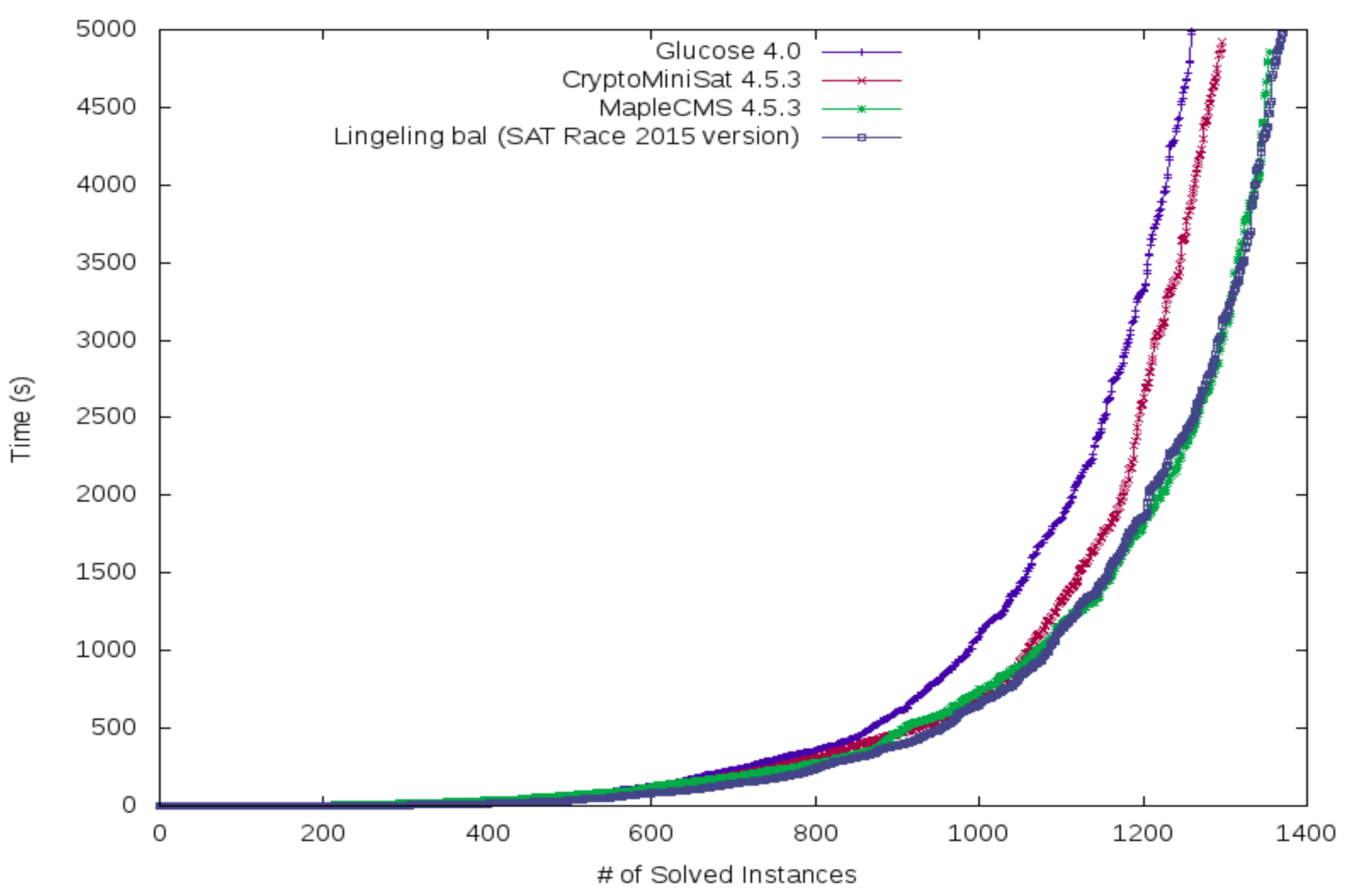
When a variable is unassigned, the variable receives the learning rate reward, and the estimate Q is updated.

Most importantly, we understand why bumping certain variables and why performing multiplicative decay helps.

APPLE-TO-APPLE RESULTS (MINISAT WITH VSIDS VS. CHB VS. LRB)



COMPARISON WITH STATE-OF-THE-ART: CRYPTOMINISAT, MAPLECMS, GLUCOSE, AND LINGELING



RESULT: GLOBAL LEARNING-RATE

- Global Learning Rate: # of conflicts/# of decisions
- Experimental setup: ran 1200+ application and hand-crafted instances on MapleSAT with VSIDS, CHB, LRB, Berkmin, DLIS, and JW with 5400 sec timeout per instance on StarExec

Branching Heuristic	Global Learning Rate
LRB	0.452
MVSIDS	0.410
CHB	0.404
CVSIDS	0.341
BERKMIN	0.339
DLIS	0.241
JW	0.107

PART V

CONCLUSIONS AND TAKEAWAY

CONCLUSIONS AND TAKEAWAY
RESULTS EXPLAINING THE POWER OF SAT SOLVERS

- CDCL SAT solvers are polynomially-equivalent to merge resolution
- Proof complexity of SMT solvers [RKG18]

Proof complexity

Understanding the efficacy of solvers (practical proof systems)

Parameterized complexity

- Introduced the merge parameter as a basis for upper bound analysis [ZG18]
- Merge as a feature for machine learning based clause deletion

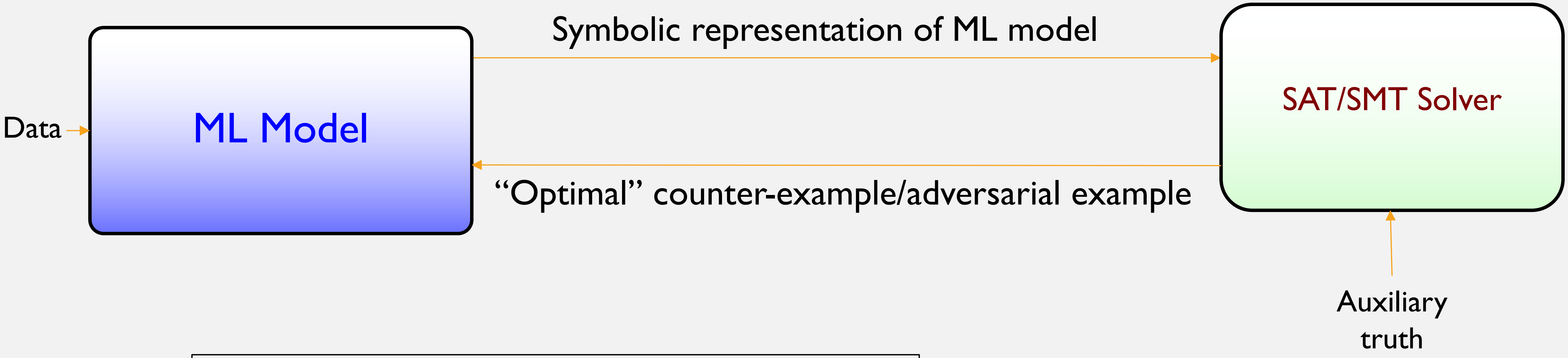
Machine learning based solver design

- Introduced the idea of ‘solver as a collection of machine learning based optimization engines’[LG+16,LG+17,LG+18]
- Successfully used this paradigm to develop new ML-based branching, restart, initialization, and splitting policies in MapleSAT

PART VI

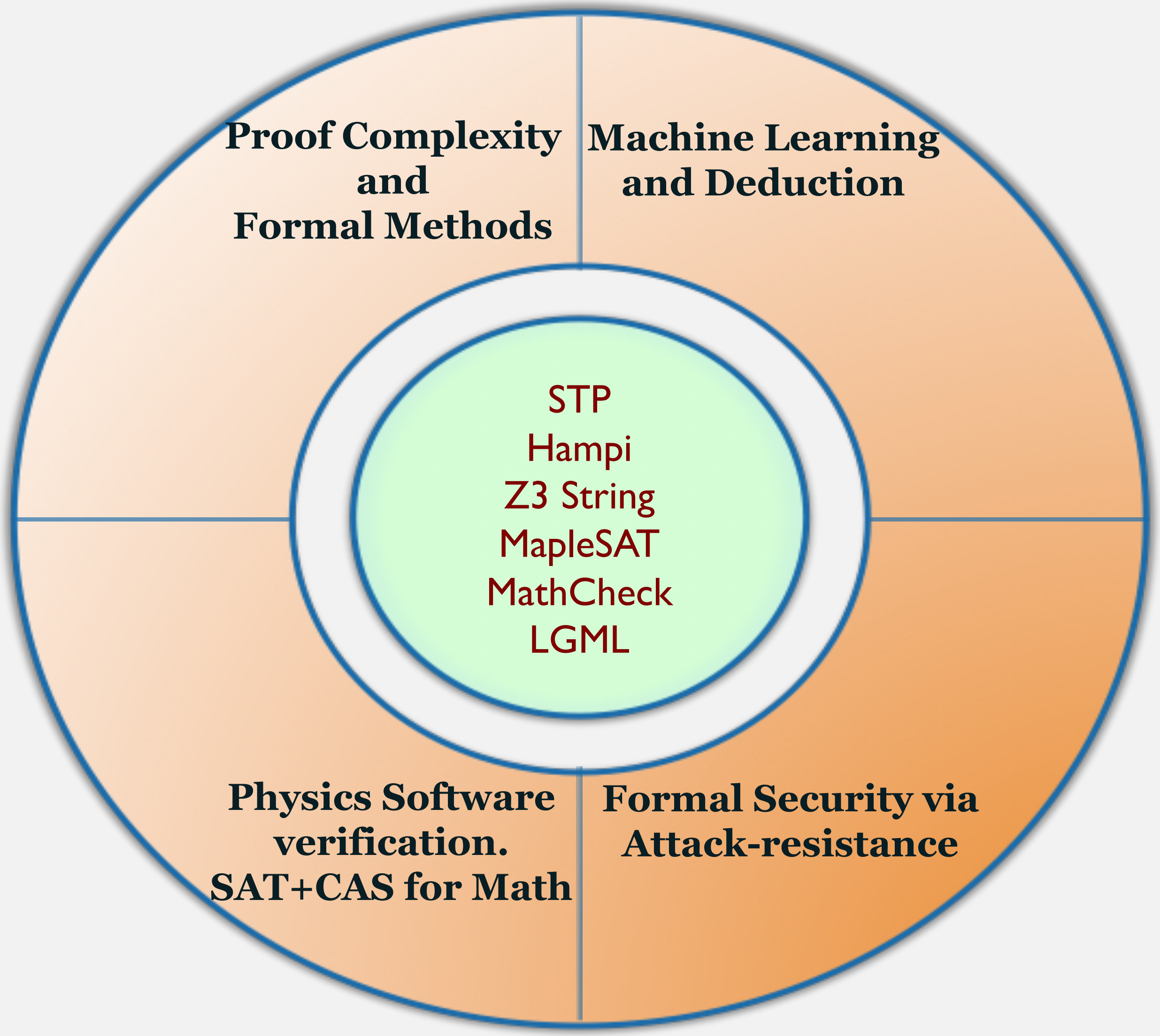
One more thing…

LOGIC GUIDED MACHINE LEARNING



Preliminary results: used this idea to learn the Pythagorean theorem and the Sine function from data

CURRENT RESEARCH PROGRAM



REFERENCES

- [MMZZM01] Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L. and Malik, S. Chaff: Engineering an efficient SAT solver. DAC 2001
- [BKS03] Beame, P., Kautz, H., Sabharwal, A. Understanding the Power of Clause Learning. IJCAI 2003
- [WGS03] Williams, R., Gomes, C.P. and Selman, B. Backdoors to typical case complexity. IJCAI 2003
- [B09] Biere, A. Adaptive restart strategies for conflict driven SAT solvers. SAT 2008
- [BSG09] Dilkina, B., Gomes, C.P. and Sabharwal, A. Backdoors in the context of learning. SAT 2009
- [KSM11] Katebi, H., Sakallah, K.A. and Marques-Silva, J.P. Empirical study of the anatomy of modern SAT solvers. SAT 2011
- [AL12] Ansótegui, C., Giráldez-Cru, J. and Levy, J. The community structure of SAT formulas. SAT 2012
- [NGFAS14] Newsham, Z., Ganesh, V., Fischmeister, S., Audemard, G. and Simon, L. Impact of community structure on SAT solver performance. SAT 2014
- [LGZC15] Liang, J.H., Ganesh, V., Zulkoski, E., Zaman, A. and Czarnecki, K. Understanding VSIDS branching heuristics in CDCL SAT solvers. HVC 2015.
- [LGRC15] Liang, J.H., Ganesh, V., Raman, V., and Czarnecki, K. SAT-based Analysis of Large Real-world Feature Models are Easy. SPLC 2015
- [LGPC16] Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Learning Rate Based Branching Heuristic for SAT Solvers. SAT 2016
- [LGPC+16] Liang, J.H., Ganesh, V., Poupart, P., and Czarnecki, K. Conflict-history Based Branching Heuristic for SAT Solvers. AAAI 2016
- [RKG18] Robere, R., Kolkolova, A., Ganesh, V. Proof Complexity of SMT Solvers. CAV 2018