Boolean SAT Solvers
A Foundational Perspective

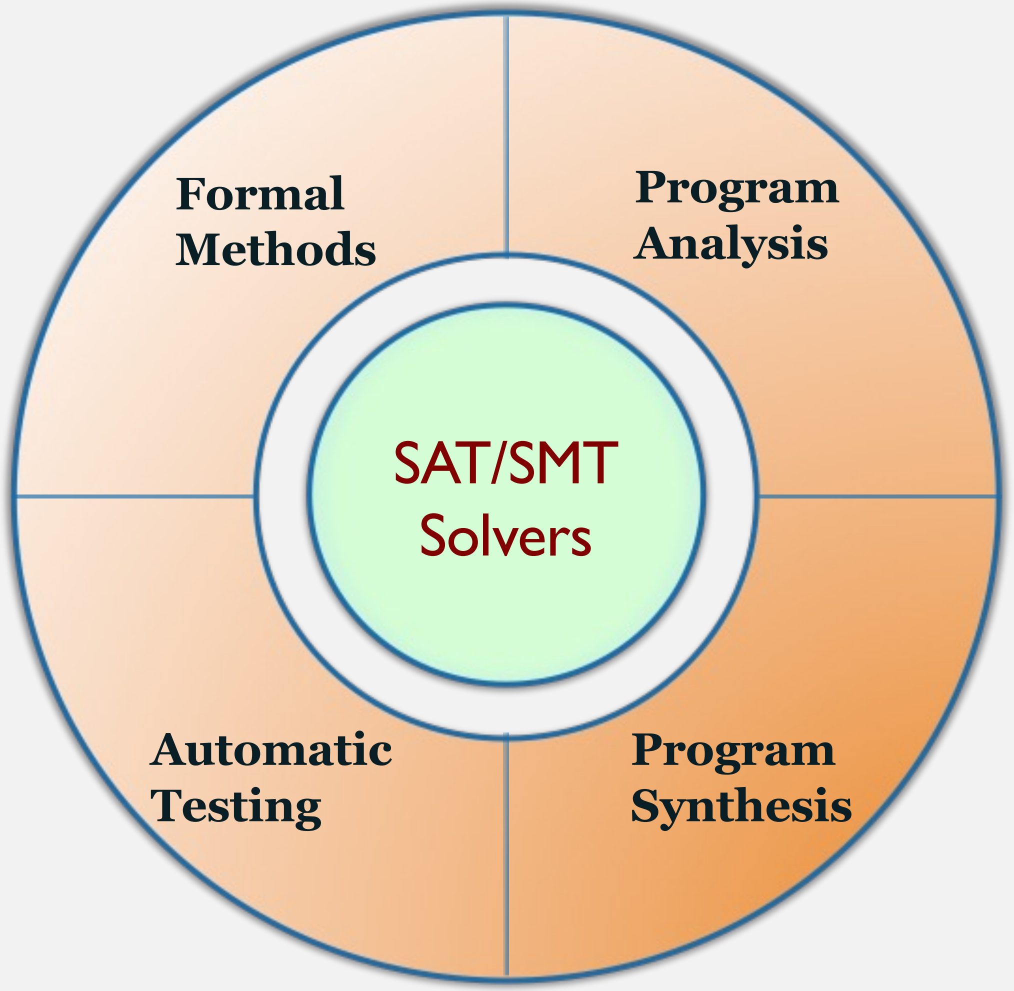
Vijay Ganesh
University of Waterloo, Canada

PART I

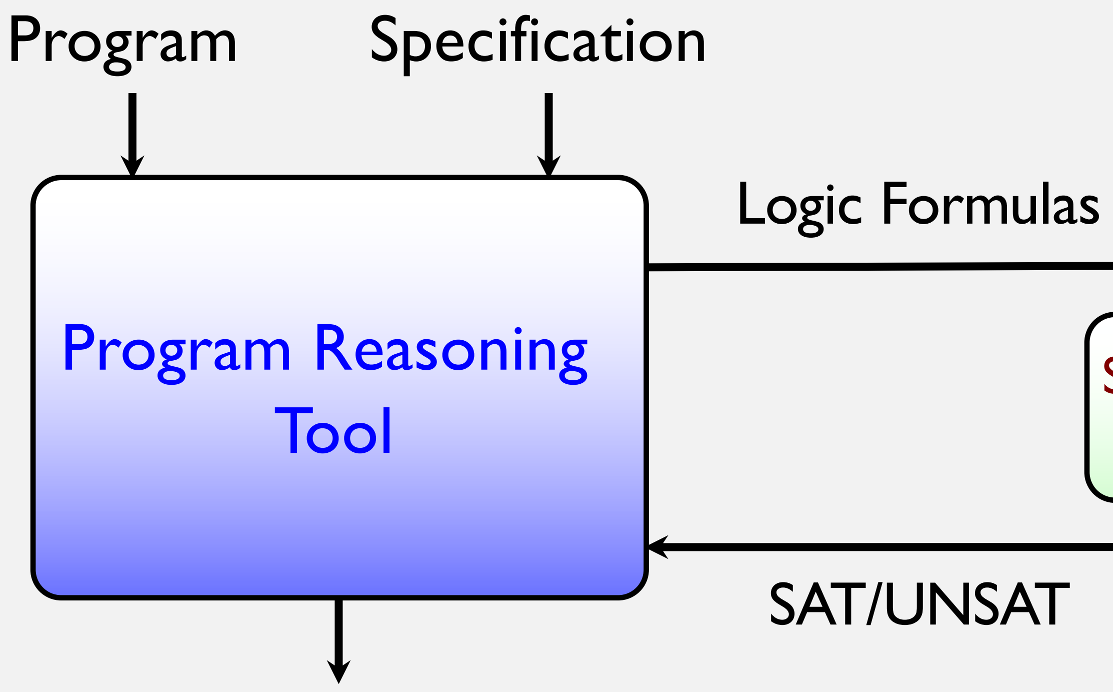
MOTIVATION

WHY SHOULD YOU CARE ABOUT SAT SOLVERS?

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

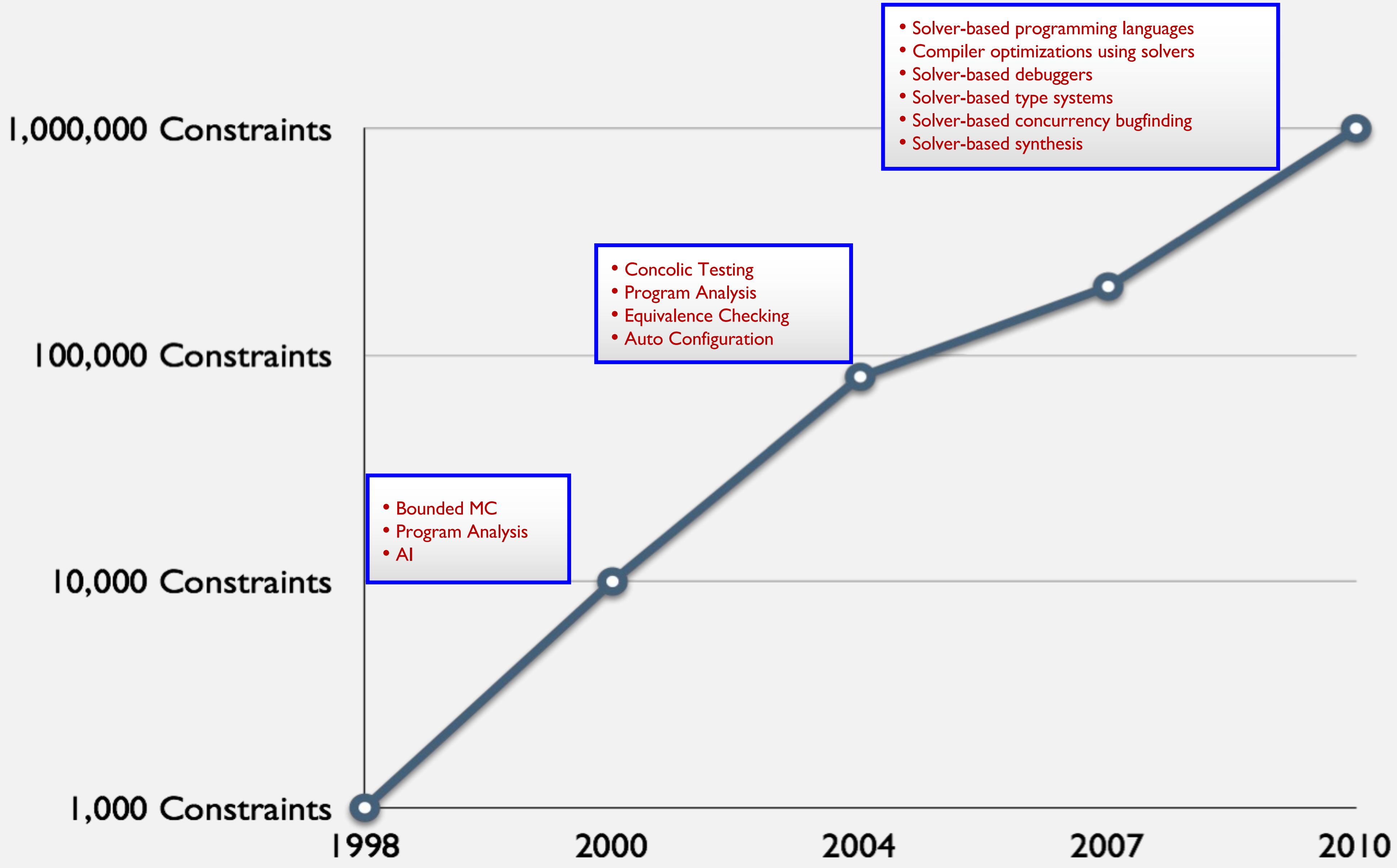


SOFTWARE ENGINEERING USING SOLVERS
ENGINEERING, USABILITY, NOVELTY



Program is correct?
or Generate Counterexamples (test cases)

SAT/SMT SOLVER RESEARCH STORY
A 1000X+ IMPROVEMENT

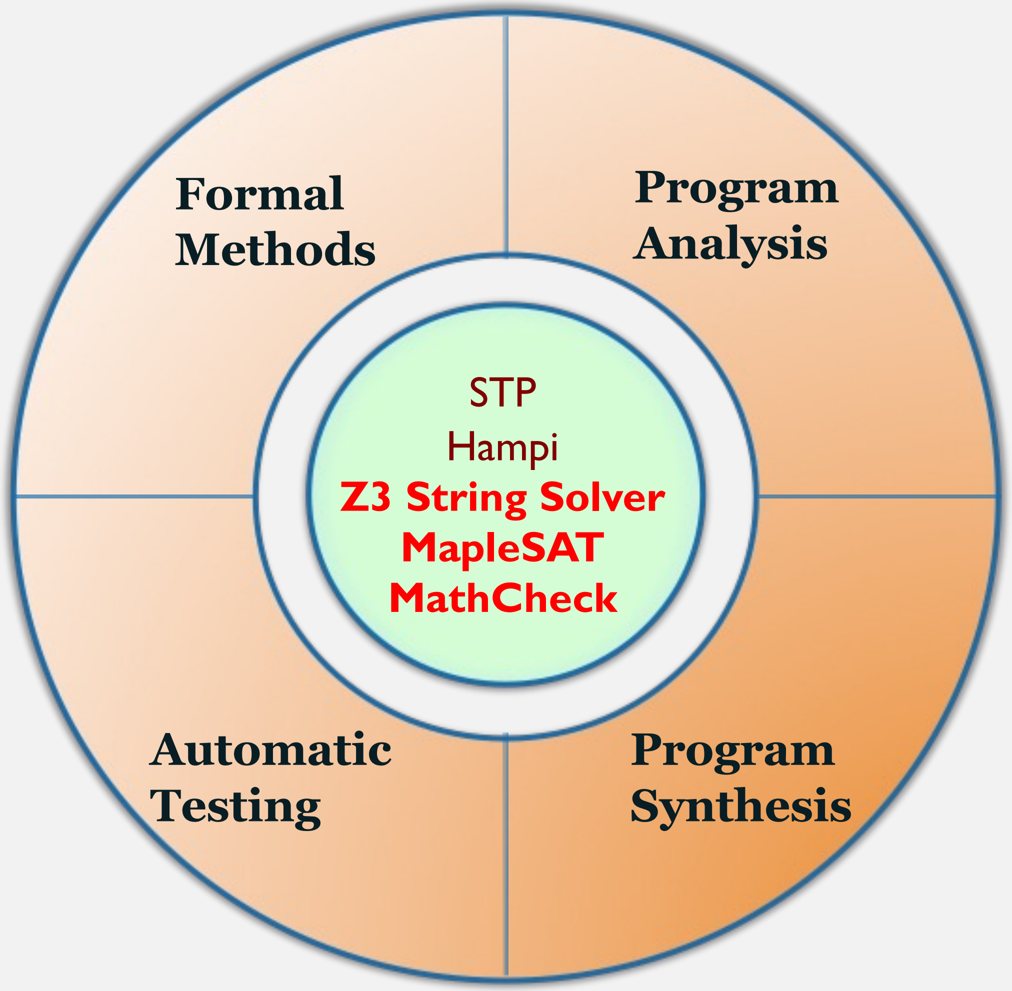


- Bounded MC
- Program Analysis
- AI

- Concolic Testing
- Program Analysis
- Equivalence Checking
- Auto Configuration

- Solver-based programming languages
- Compiler optimizations using solvers
- Solver-based debuggers
- Solver-based type systems
- Solver-based concurrency bugfinding
- Solver-based synthesis

IMPORTANT CONTRIBUTIONS
AN INDISPENSABLE TACTIC FOR ANY STRATEGY



TOPICS COVERED AND TAKEAWAY
LECTURES 1 AND 2

- CDCL (conflict-driven clause-learning) SAT solvers
- Programmatic SAT solvers, a step towards SMT solvers
- Machine learning inside sequential and parallel SAT solvers
- Deeper complexity-theoretic understanding of CDCL SAT solvers
- Open problems and future directions
- Helping you become a SAT power-user

TOPICS COVERED AND TAKEAWAY
LECTURES 3 AND 4

- High-level architecture of CDCL(T)/SMT solvers
- Combination of first-order theories
- First-order theories – EUF, strings,…
- Proof-complexity of SMT solvers
- Open problems and future directions
- Helping you become an SMT power-user

PART II

SAT SOLVER BASICS

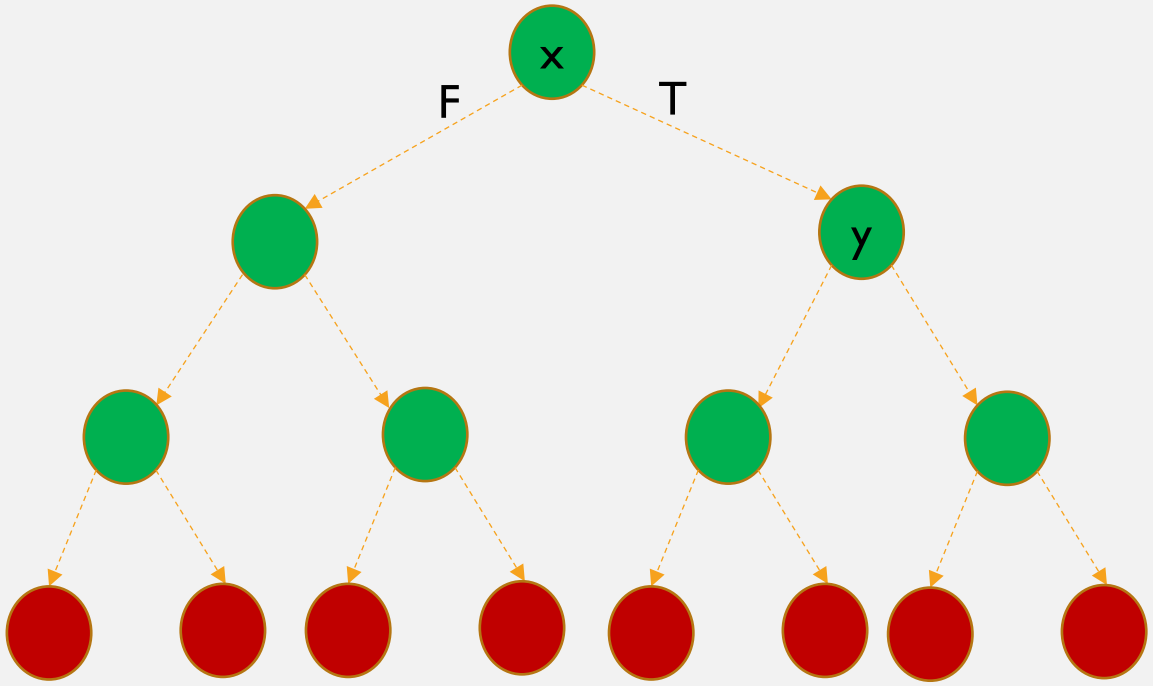
THE POWER OF CONFLICT-DRIVEN CLAUSE-LEARNING

THE BOOLEAN SATISFIABILITY PROBLEM
SOME STANDARD DEFINITIONS

- A **literal** p is a Boolean variable x or its negation $\neg x$. A **clause** C is a disjunction of literals. E.g., $(x_2 \vee \neg x_{41} \vee x_{15})$. 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.

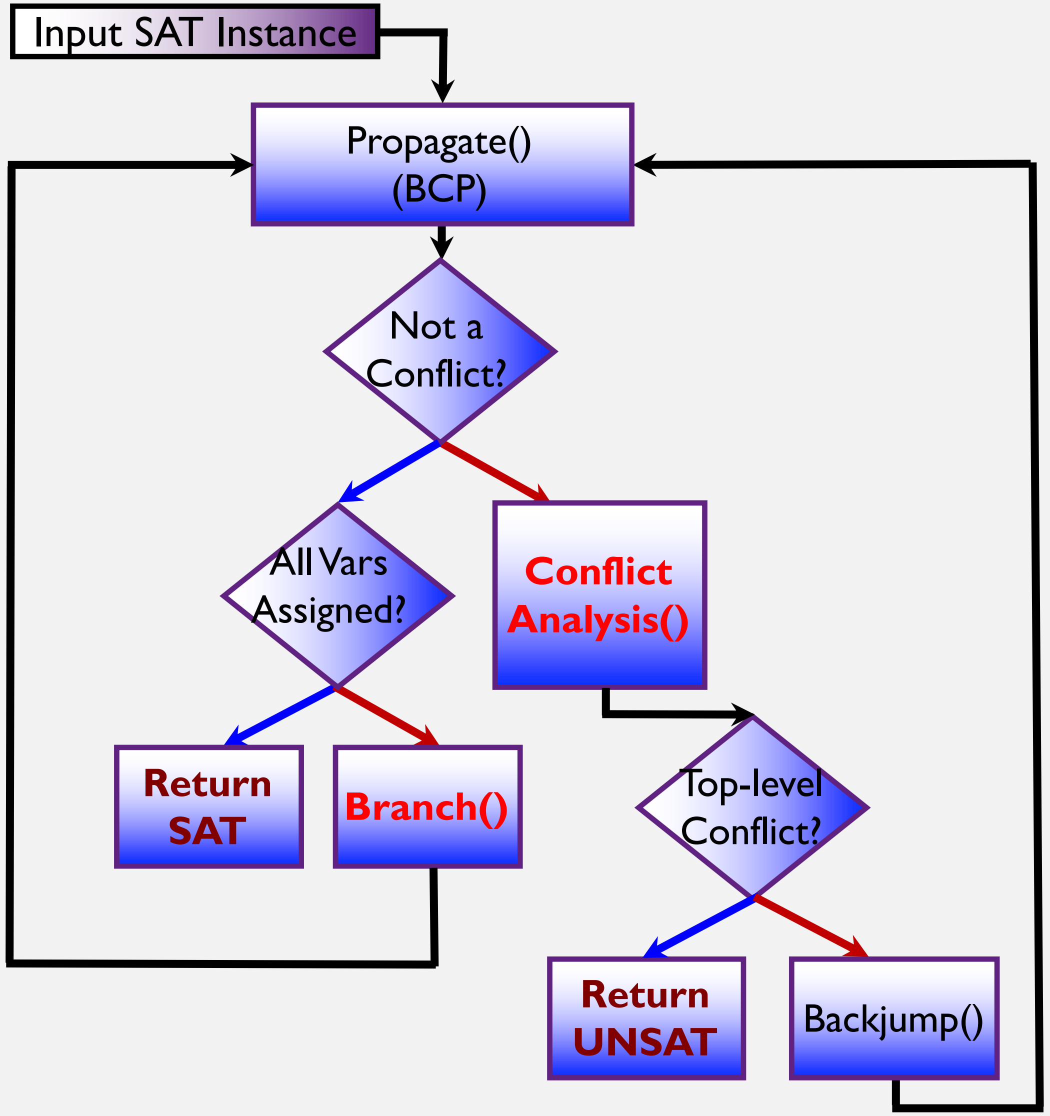
DPLL SAT SOLVER ARCHITECTURE (1958)
THE BASIC BACKTRACKING SAT SOLVER

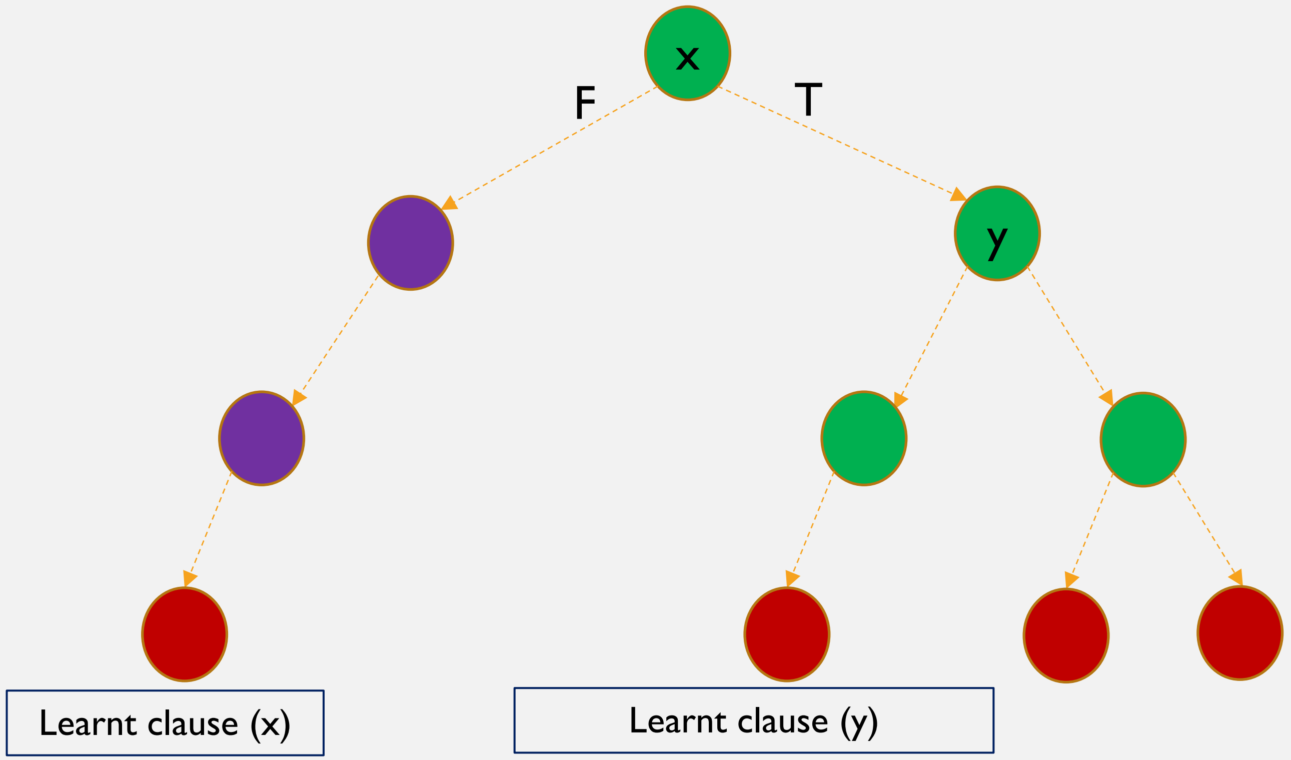
```
DPLL($\Theta_{cnf}$, assign) {
    Propagate unit clauses;
    if "conflict": return FALSE;
    if "complete assign": return TRUE;
    "pick decision variable x";
    return
        DPLL($\Theta_{cnf} |_{x=0}$, assign[x=0]) ||
        DPLL($\Theta_{cnf} |_{x=1}$, assign[x=1]);
}
```



DPLL stands for Davis, Putnam, Logemann, and Loveland

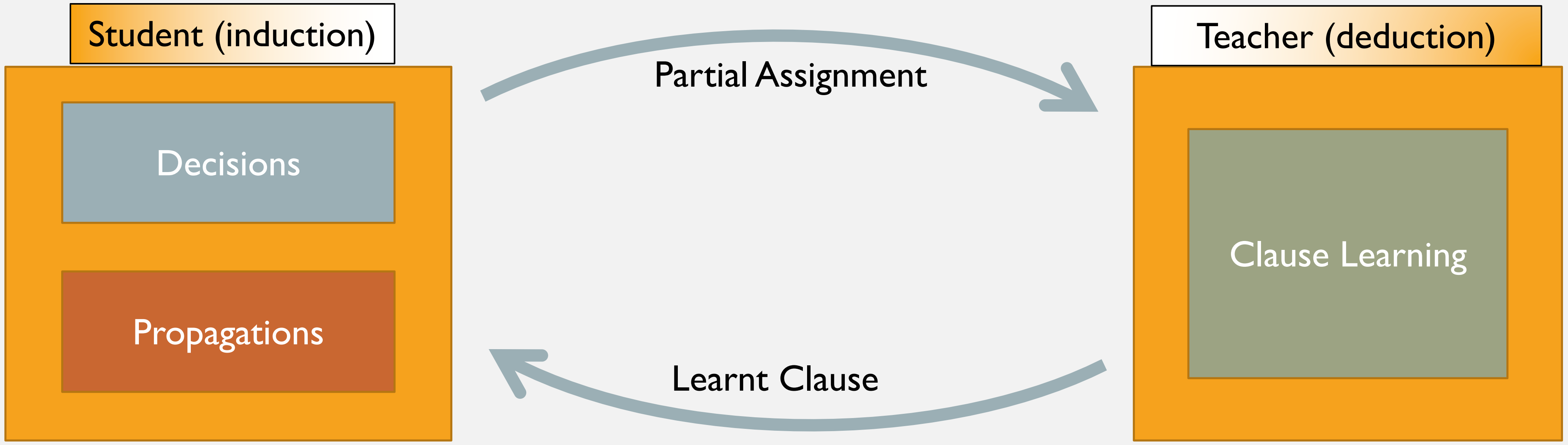
MODERN CDCL SAT SOLVER ARCHITECTURE OVERVIEW





AN ABSTRACTION OF A CDCL SAT SOLVER

TOWARDS MACHINE LEARNING IN SAT

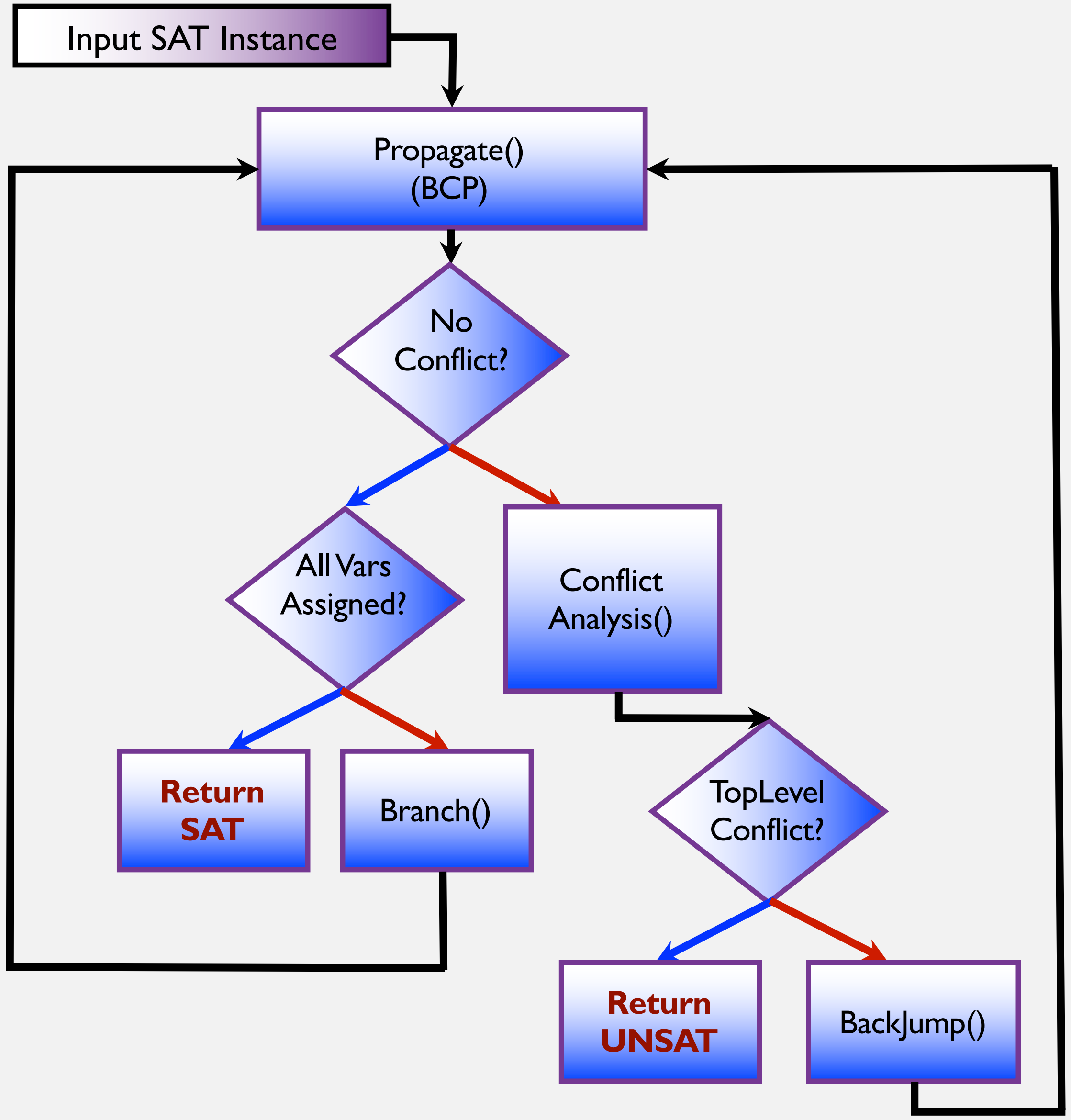


1. This abstraction captures the most essential aspects of CDCL. Combines synthesis (induction) with verification (deduction)
2. There is similar class of algorithms in reinforcement learning
3. Enabled us to design a new class of heuristics

PART III

Boolean Constraint Propagation (BCP)

MODERN CDCL SAT SOLVER ARCHITECTURE
KEY STEPS AND DATA-STRUCTURES



Key steps

- Branch() or Decide()
- BCP Propagate()
- Conflict analysis and learning()
- Backjump()
- Forget() or clause deletion()
- Restart()

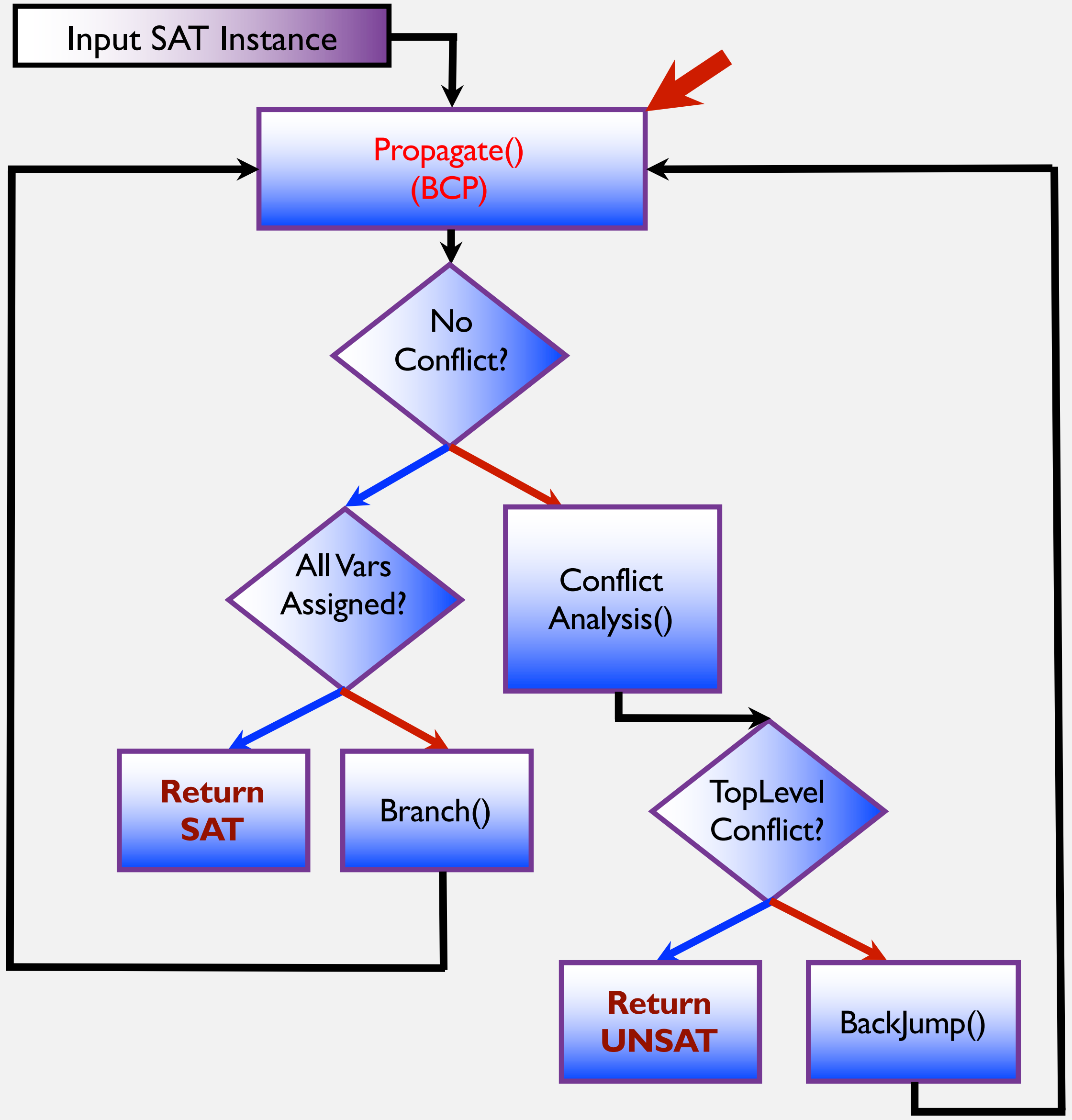
CDCL: conflict-driven clause-learning

- Conflict analysis is a key step
- Results in learning a conflict clause
- Learning clauses prunes the search space

Key data-structures (state of SAT solver):

- Activity of variables
- Input clause database
- Conflict clause database
- Conflict or implication graph
- Decision level (DL) of a variable
- Stack of partial assignments (AT)

MODERN CDCL SAT SOLVER ARCHITECTURE
BOOLEAN CONSTRAINT PROPAGATION



Goal of Boolean constraint propagation (BCP)

- Propagate inferences due to unit clauses
- Most time in solving goes into this step
- Corresponds to the unit resolution rule
- Two-watched literal scheme

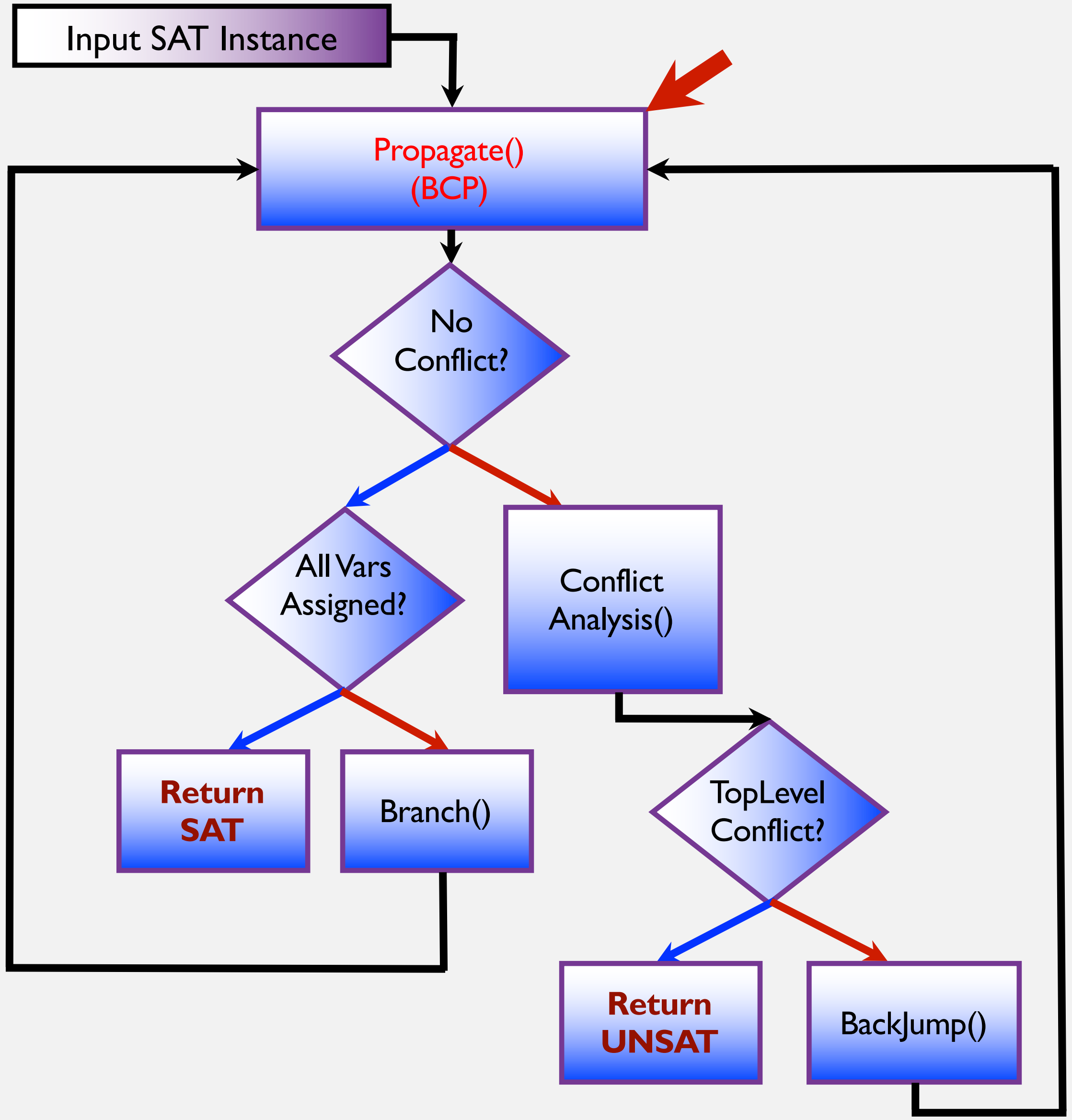
Unit resolution rule

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

$$\frac{(x_1 \vee \cdots \vee x_m \vee x_n) \quad (\neg x_n \vee y_1 \vee \cdots \vee y_m)}{(y_1 \vee \cdots \vee y_m)}$$

where x_i, y_j are Boolean literals, and $x_1 \vee \cdots \vee x_m$ are false under current partial assignment

MODERN CDCL SAT SOLVER ARCHITECTURE
BOOLEAN CONSTRAINT PROPAGATION



{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

Decide

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

Unit clause (BCP)

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

…

Unit clause (more BCP)

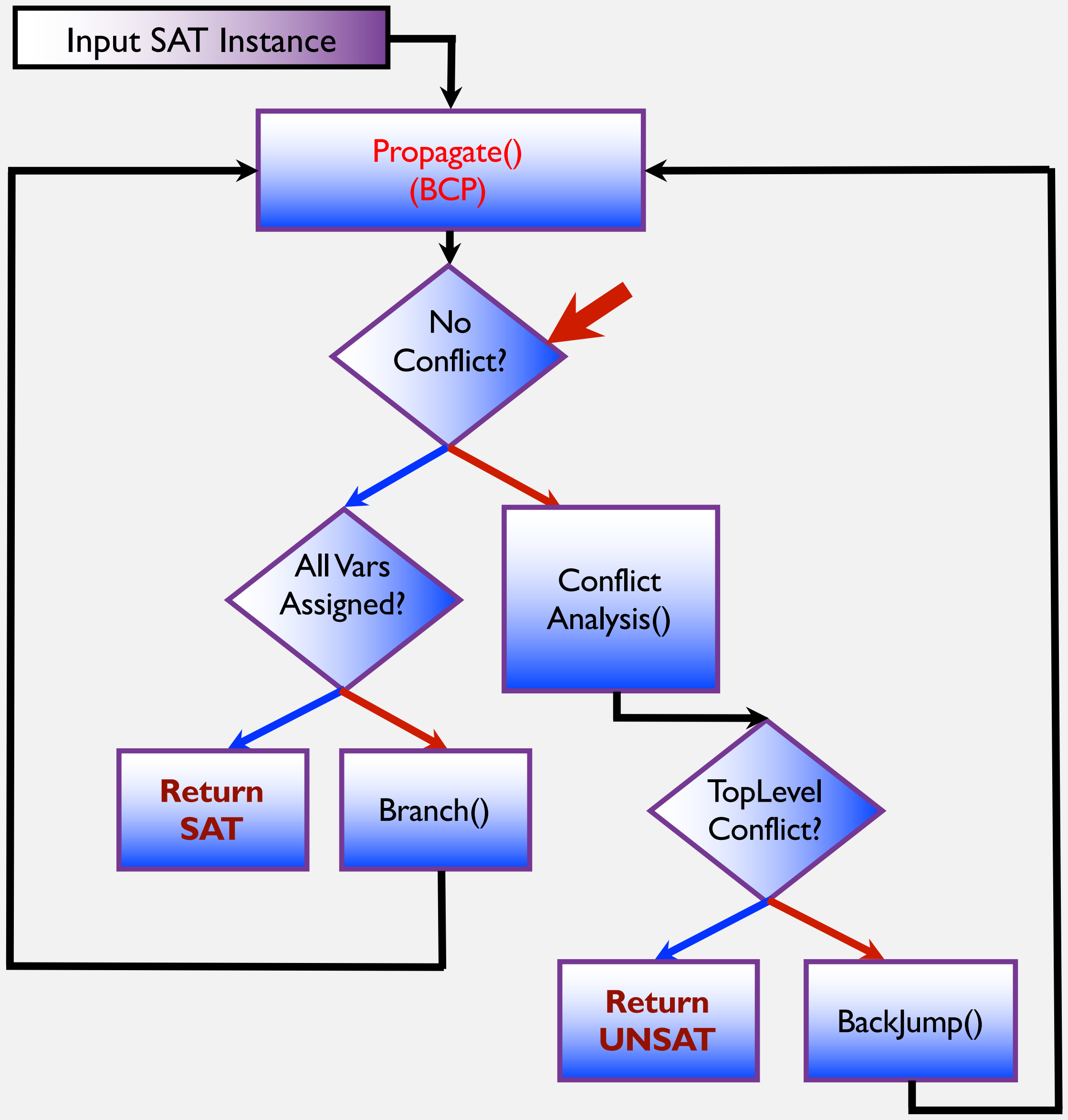
{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

CONFLICT! (Triggers conflict analysis and backjump)

{3, 6, -7, 8}
{1, 4, 7}
{-8, 4}
{-1, -3, 8}
{-3, -4, -8}
{-1, -2, 3, 4, -6}

MODERN CDCL SAT SOLVER ARCHITECTURE

WHAT IS A CONFLICT?



What is a conflict?

A partial assignment under which the input formula is not satisfiable.

What does the solver do upon reaching a conflict state?

Performs conflict analysis and learns a conflict clause.

What does the solver do upon reaching a non-conflict or inconclusive state?

Branches on a new unassigned variable.

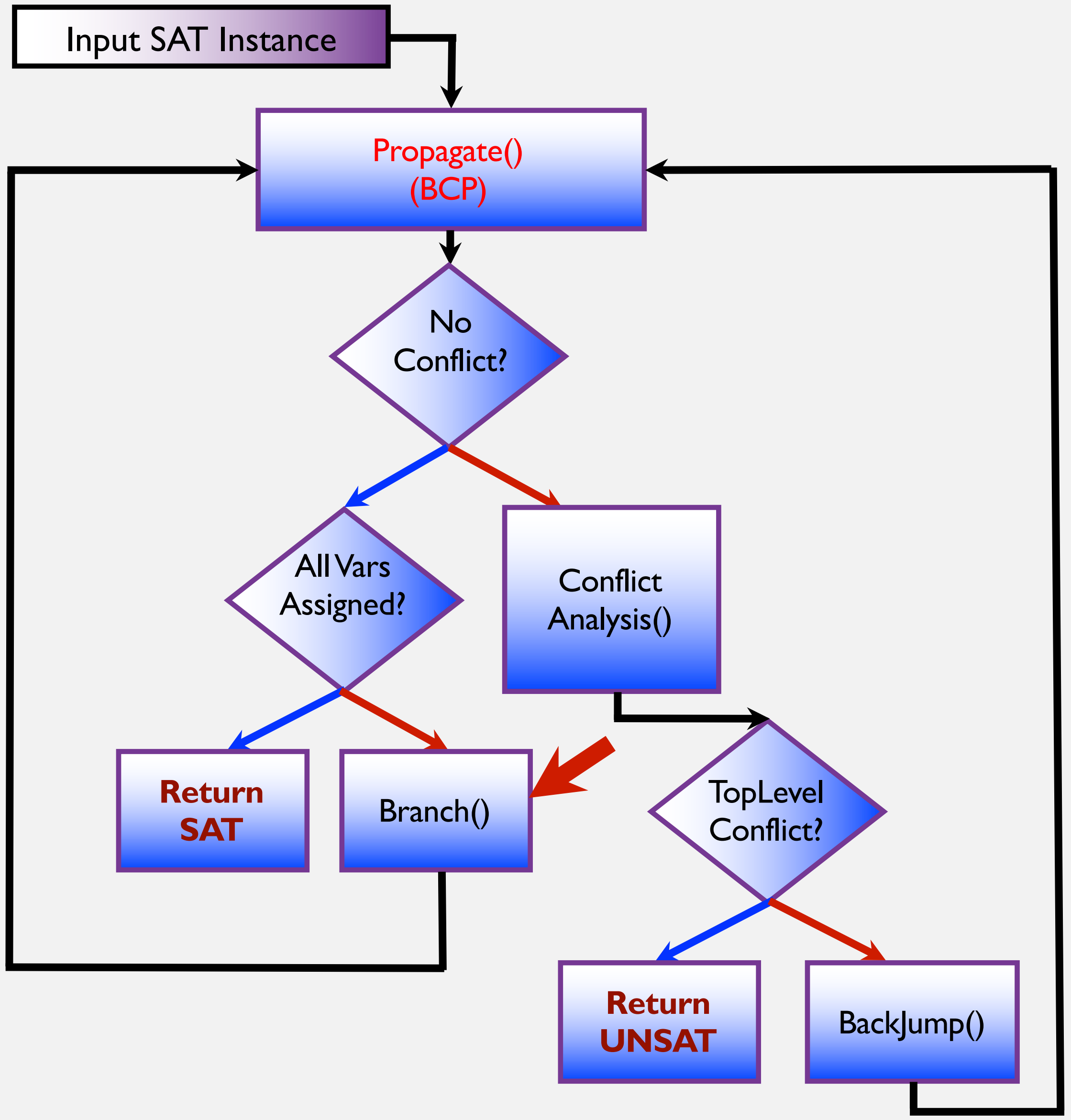
What is a top-level conflict?

A conflict that occurs when the solver has not made any decisions (or has not branched on any variable). This means input is UNSAT.

PART IV

Machine Learning based Branching in SAT

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 to 0, at the start of solver run

```
VSIDS() {
    Upon conflict
        * Bump activity of variables appearing on the conflict
          side of the conflict graph
        * Decay activity of all variables by a +ve constant < 1
    Return unassigned variable with highest activity
} //End of  VSIDS
```

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

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

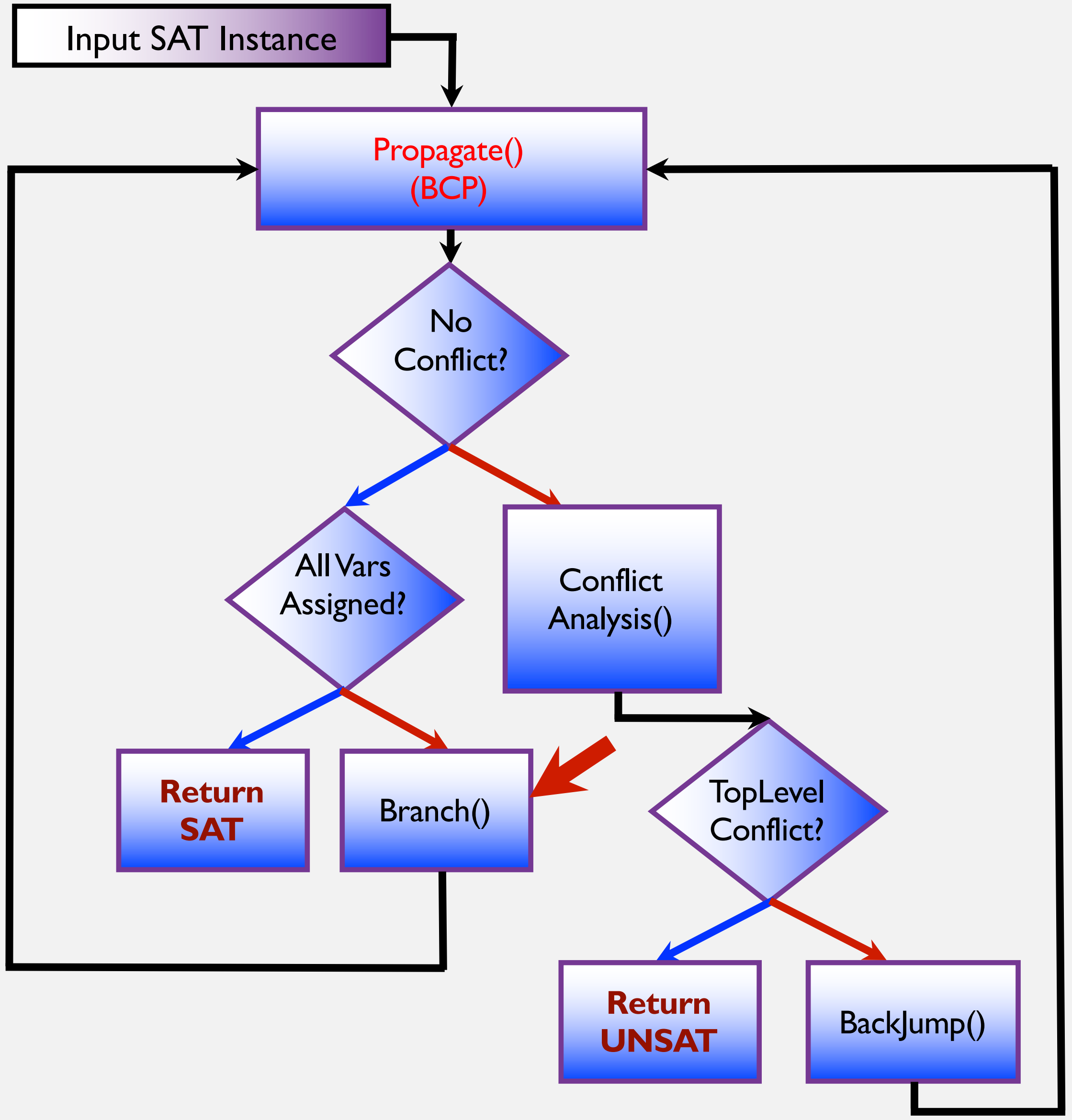
- A "dynamic" ranking function that ranks variables in a formula in descending order
- Re-ranks the variables at regular intervals throughout the run of a SAT solver
- We were unsatisfied with this understanding of VSIDS branching heuristic

Our experiments and results:

- We studied 7 of the most well-known branching heuristics in detail
- Modeled branching heuristics as techniques solving global learning rate optimization problem
- Which in turn led us to devise 3 new branching heuristics that for the first time beat VSIDS

MODERN CDCL SAT SOLVER ARCHITECTURE

DECIDE(): VSIDS BRANCHING HEURISTIC



VSIDS (Variable State Independent Decaying Sum) Branching

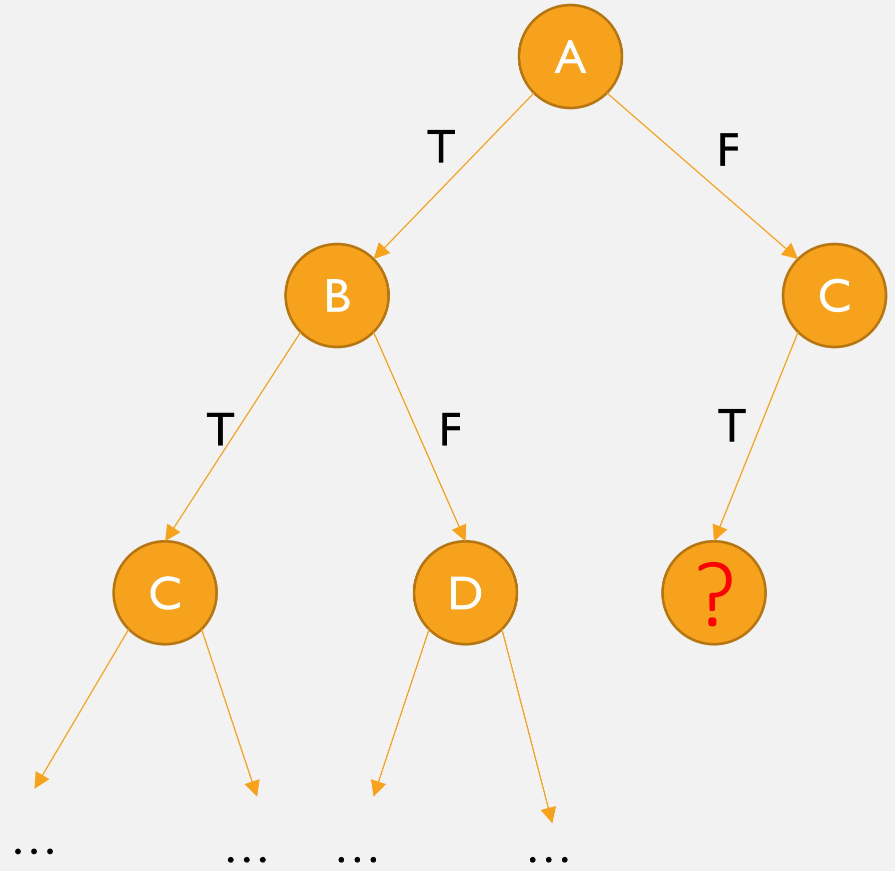
- Imposes dynamic variable order
- Each variable is assigned a floating-point value called activity
- Measures how "active" variable is in recent conflict clauses

VSIDS pseudo-code

- Initialize activity of all variables to 0, at the start of solver run

```
VSIDS() {
    Upon conflict
        * Bump activity of variables appearing on the conflict
          side of the conflict graph
        * Decay activity of all variables by a +ve constant < 1
    Return unassigned variable with highest activity
} //End of  VSIDS
```

WHAT DOES VSIDS DO?



<u>Rank the variables</u>

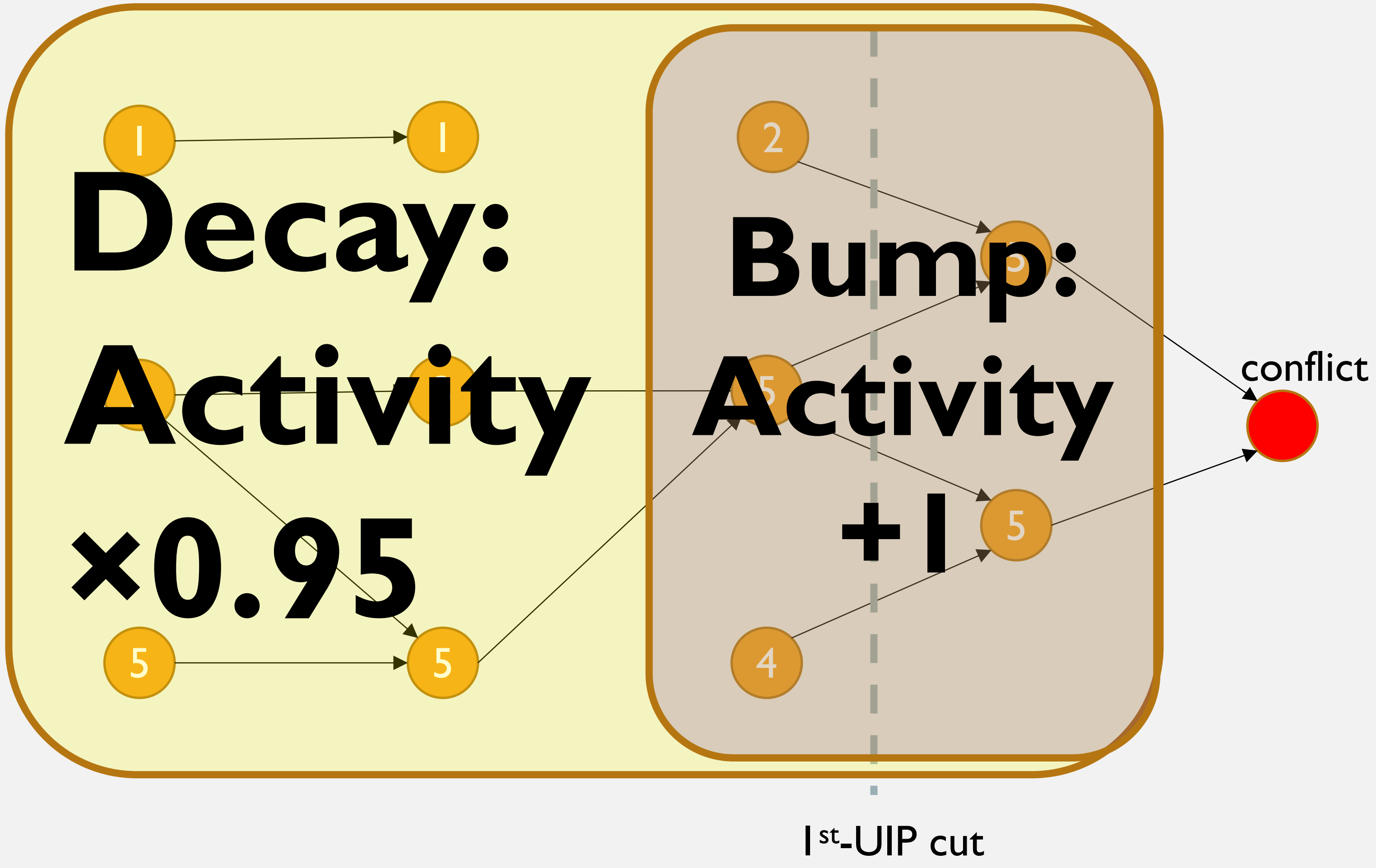
Activity[E] = 0.9

Branch on E next

Activity[F] = 0.8

Activity[G] = 0.7

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.

VSIDS: WHY BUMP AND DECAY?

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

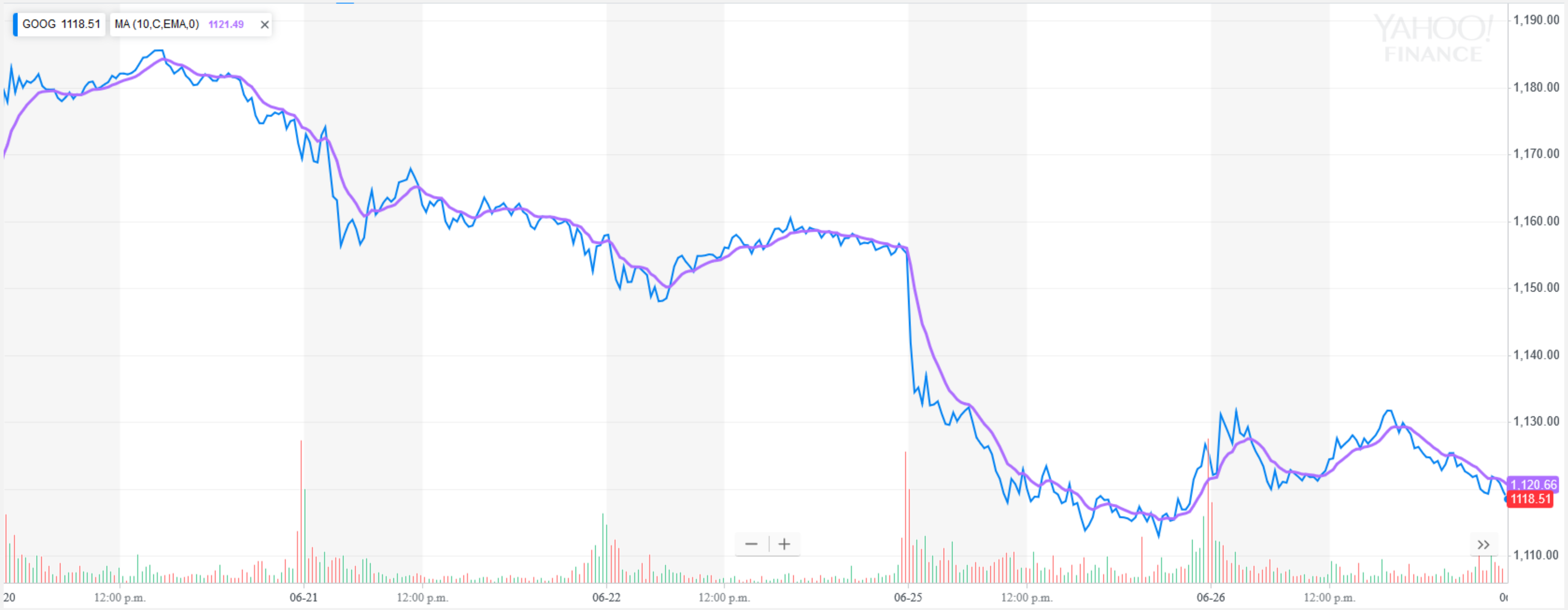
```
for all variables v:
        activity[v] = 0

  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
```

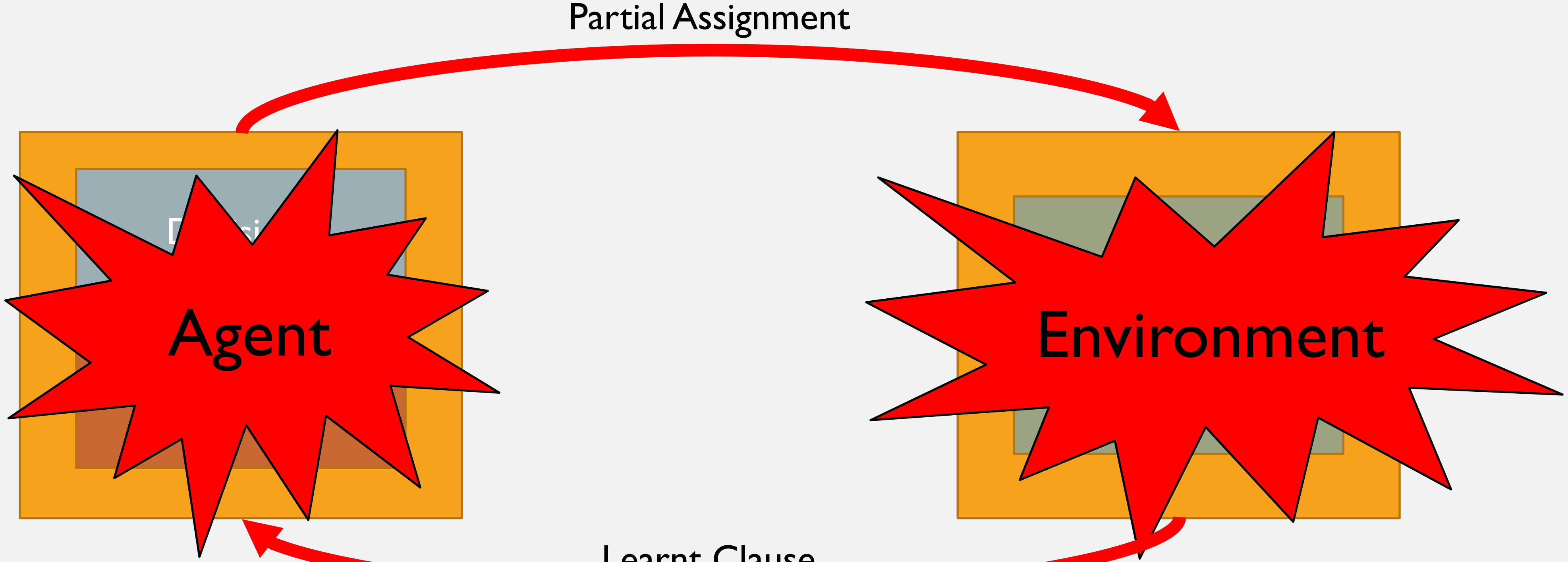
Decay observation:
$bump_{t-1} 0.95^1$
$+ bump_{t-2} 0.95^2$
$+ bump_{t-3} 0.95^3$
$+ \cdots$

More weight to recent bumps via exponential moving average

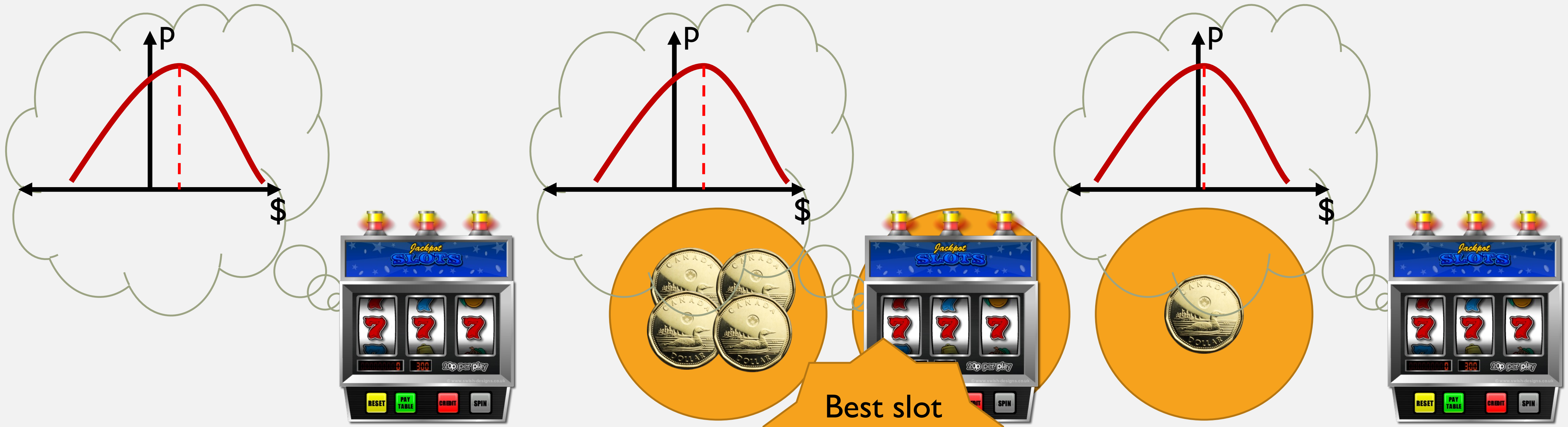
EXPONENTIAL MOVING AVERAGE



CDCL FEEDBACK LOOP



MULTI-ARMED BANDIT PROBLEM



Best slot machine to play (for now)

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

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

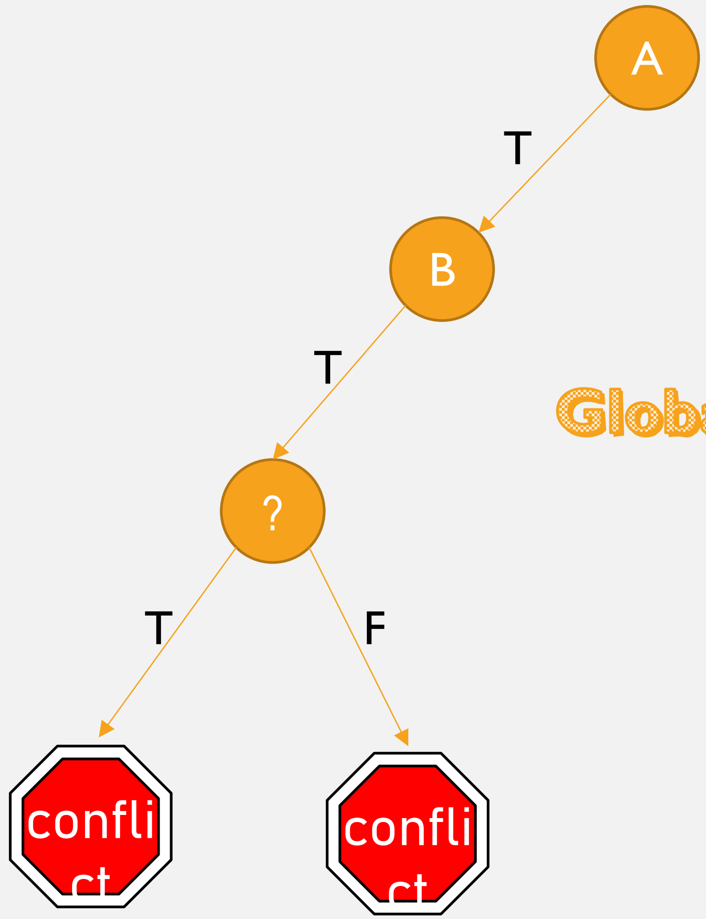
Less weight

More weight

REINFORCEMENT LEARNING AND CDCL

Reinforcement Learning	CDCL
Agent	Branching Heuristic + BCP
Environment	Clause learning
Policy	Variable Ranking
Action	Decision
Estimated Reward (Q)	Activity
Reward	Bump
Exponential Moving Average	Decay

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 methods that solve global learning rate (GLR) optimization problem

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

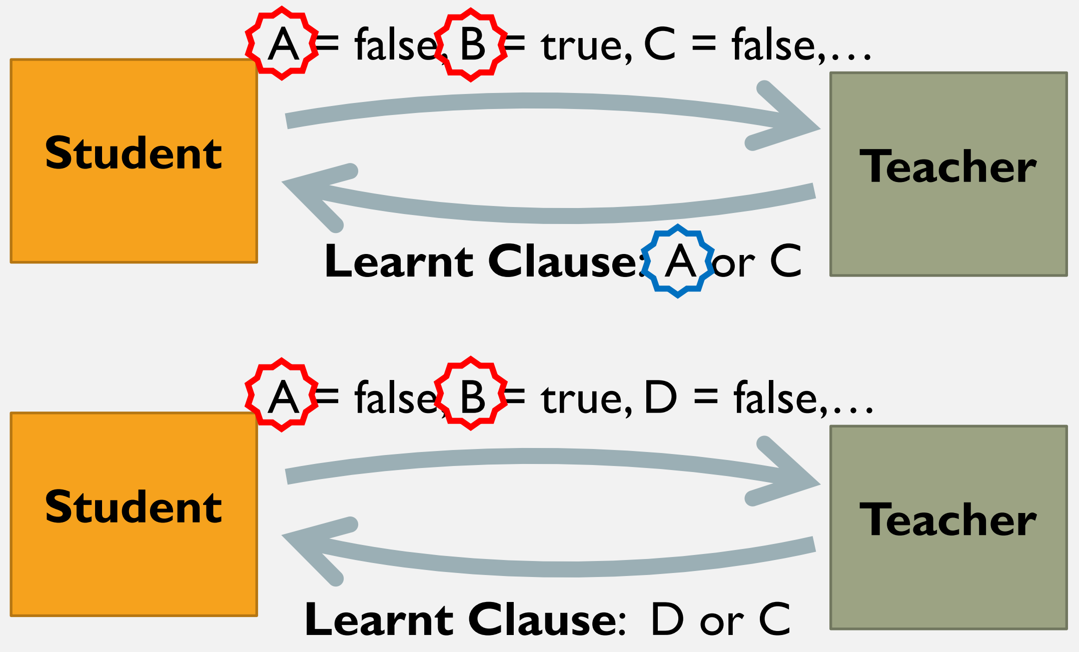
Finding 2: Branch on Conflict Analysis Variables maximizes GLR

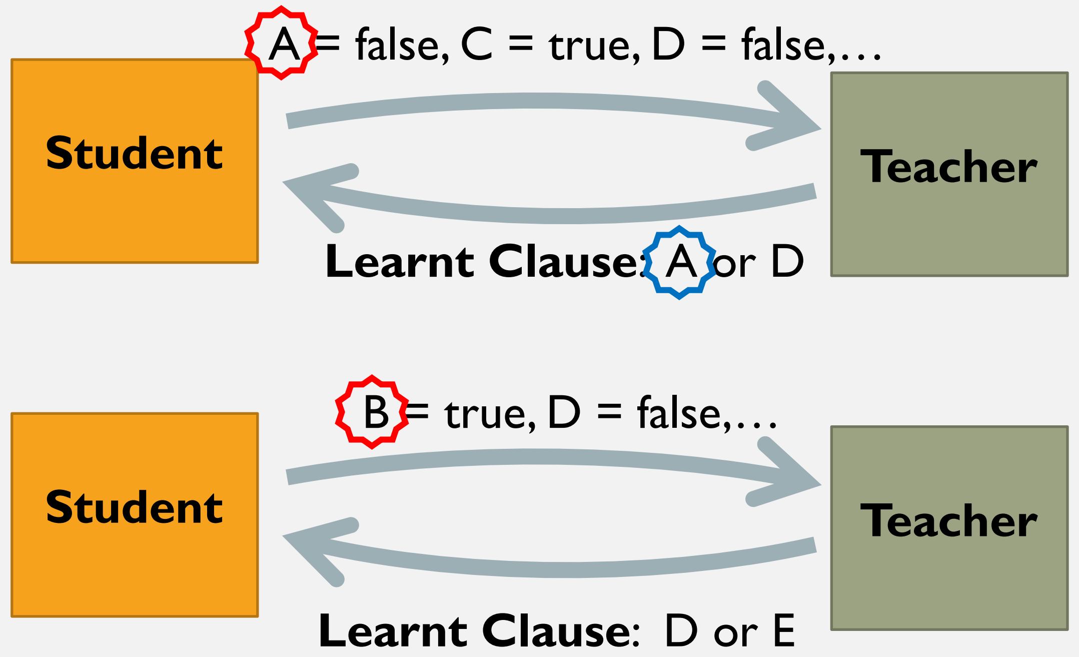
Successful branching heuristics focus on variables involved in “recent” conflicts to maximize GLR. Reward variables that gave you a conflict

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

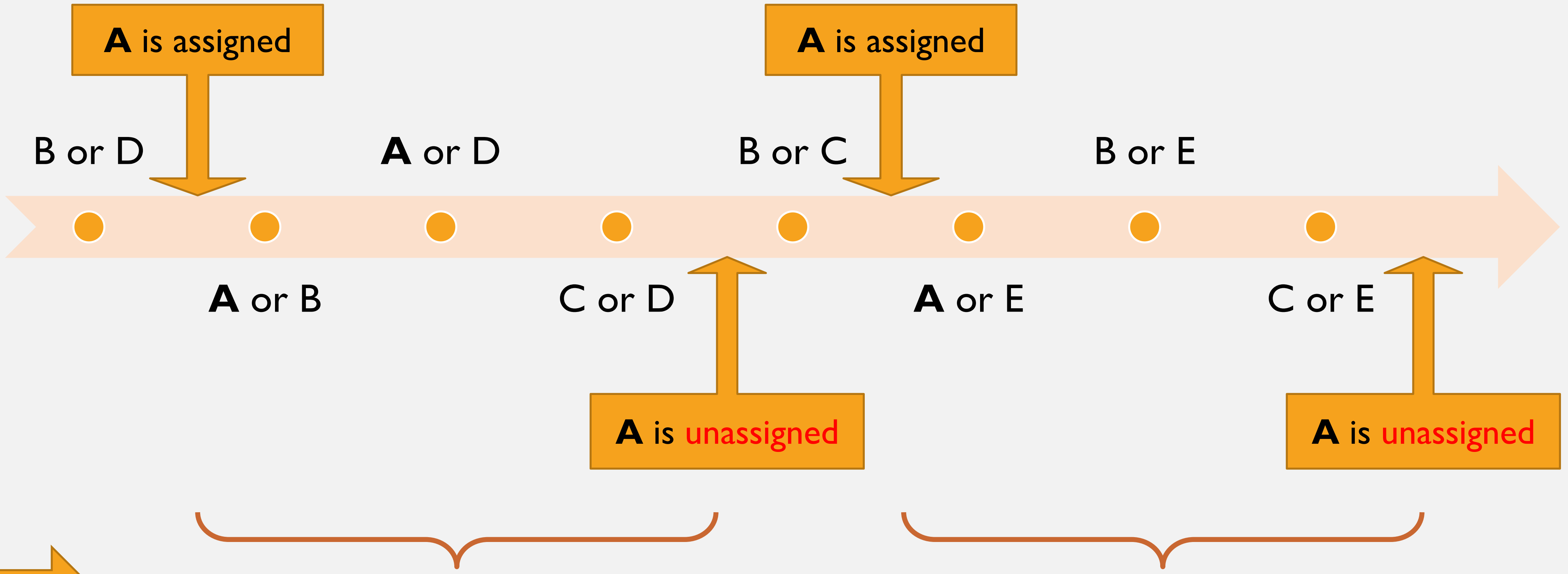




sampled_learning_rate(**A**) = 2/3

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

LEARNING-RATE BRANCHING (LRB) EXAMPLE



"Rewards" sampled_learning_rate(**A**) = 2/3 sampled_learning_rate(**A**) = 1/3

Activity(A) exponential moving average = $(1 - \alpha)^1 \times 2/3 + (1 - \alpha)^0 \times 1/3$

VSIDS

The reward is a constant

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

EMA performed for all variables at the same time

After each conflict, the activities of all variables are decayed

LRB

The reward is not constant

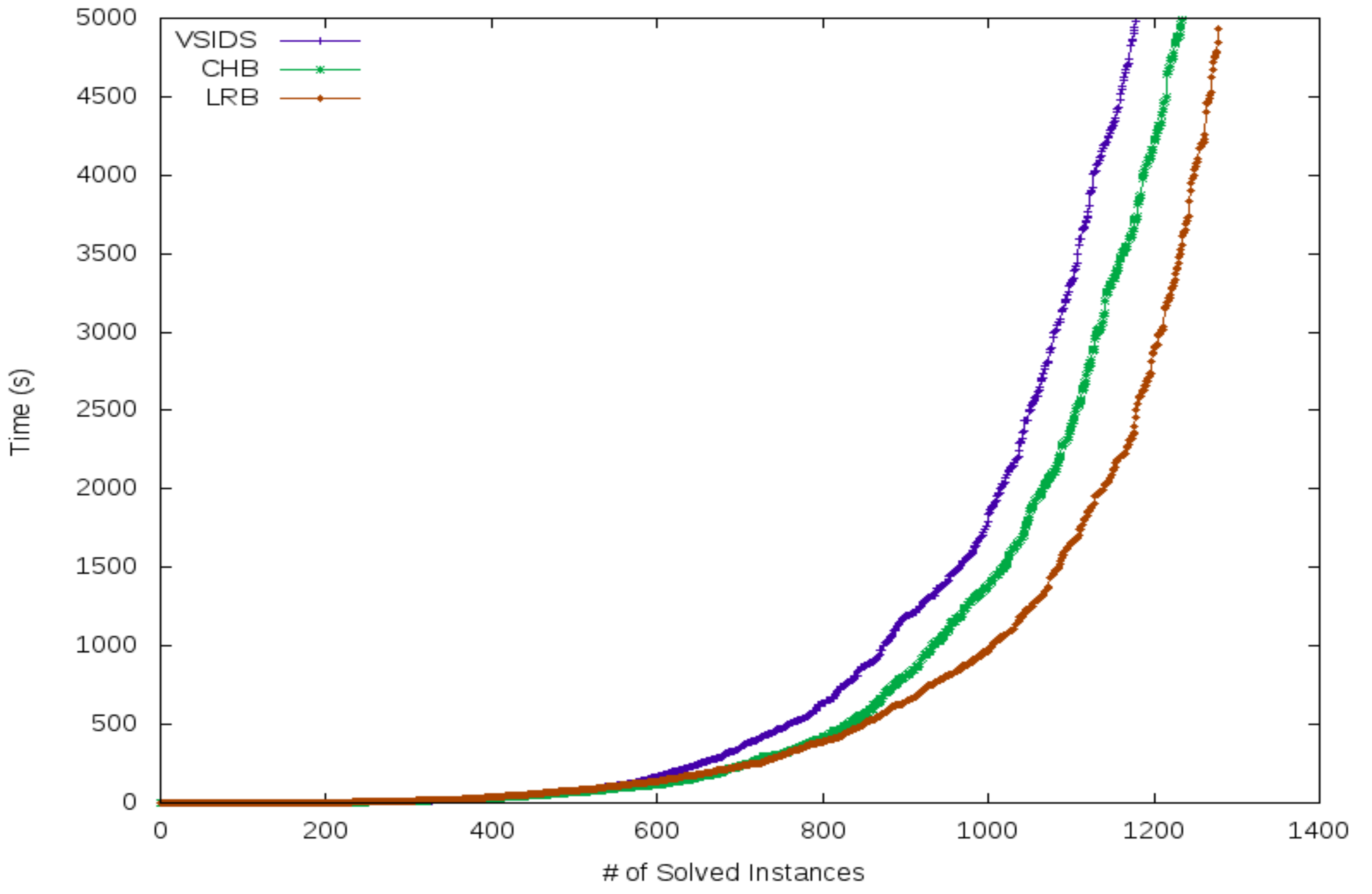
Every time a variable appears in a conflict analysis, the numerator of its learning rate reward is incremented. After each conflict, the denominator of each assigned variable's learning rate reward is incremented

EMA performed only when variable goes from assigned to unassigned

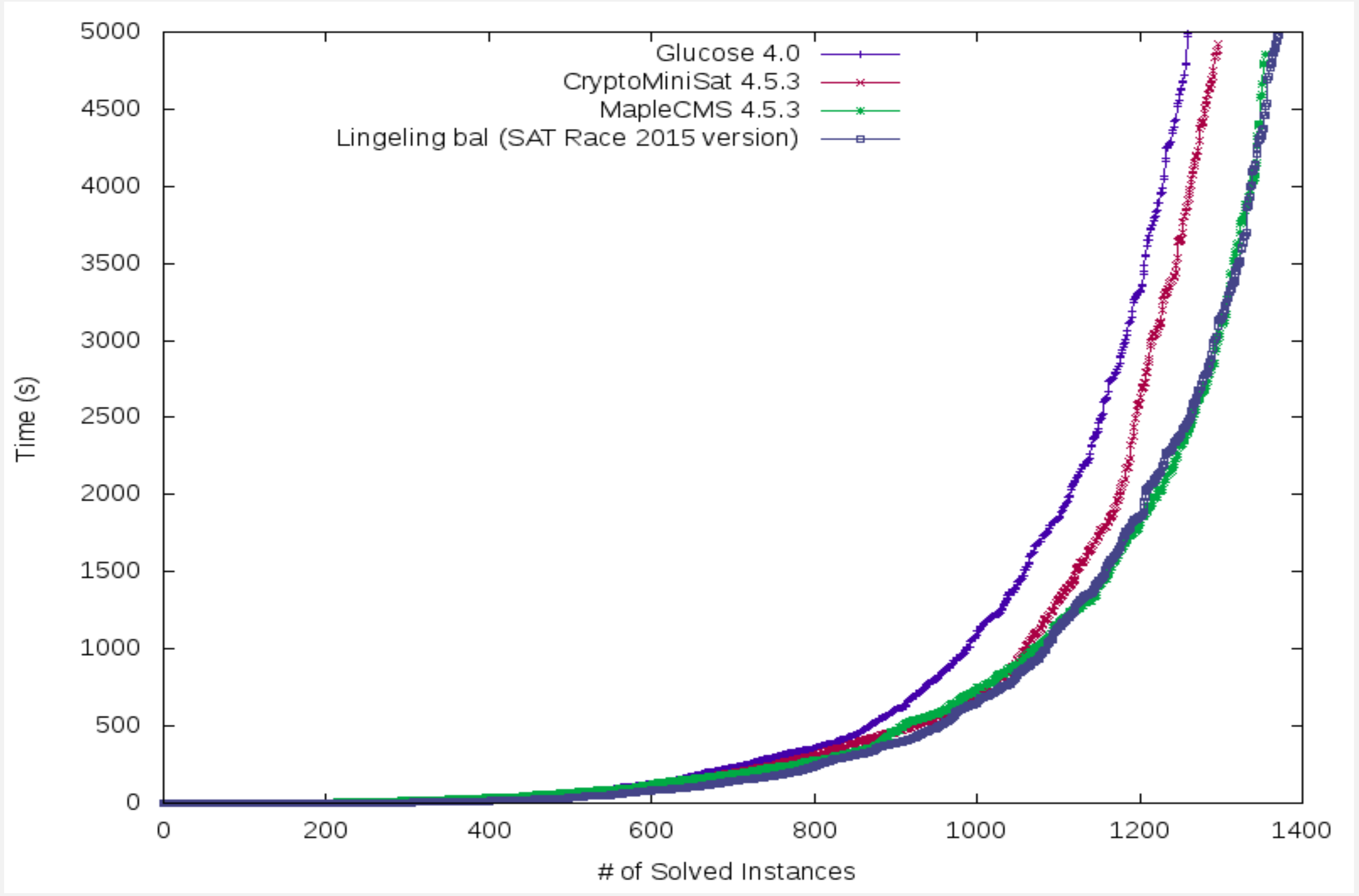
When a variable is unassigned, the variable receives the learning rate reward, and the estimate Q is updated.

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

CONCLUSIONS ON BRANCHING HEURISTICS

- View branching heuristics as a technique to solve an optimization problem whose objective function is defined in terms of global learning rate

- Leverage reinforcement learning to solve such optimization problems, making use of the enormous amount of online data generated by a CDCL SAT solver

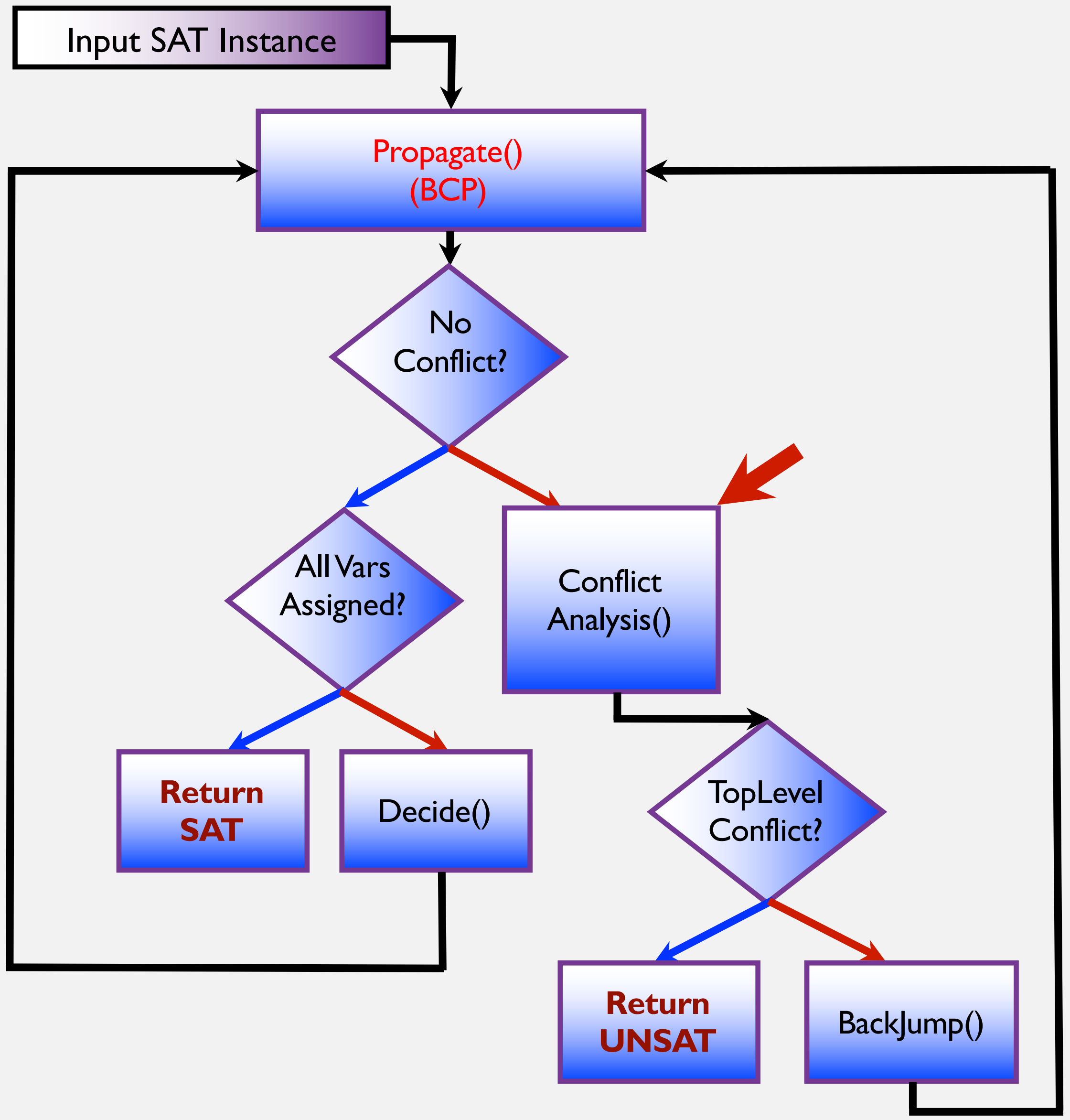
- LRB (and CHB) shown to be a significant improvement over state-of-the-art VSIDS branching heuristic, for the first time in 15 years

PART V

Conflict Analysis

MODERN CDCL SAT SOLVER ARCHITECTURE

CONFLICT ANALYSIS AND CLAUSE LEARNING: DEFINITIONS



Decision Level (DL)

- Map from variables in input formula to natural numbers
- All unit clauses and resultant propagations get DL = 0
- Every decision var gets a DL in increasing order >= 1
- All propagations due to decision at DL=x, get the DL=x

Conflict Graph (CG) or Implication Graph

- Directed Graph to record decisions and propagations
- Vertices: literals, Edges: unit clauses

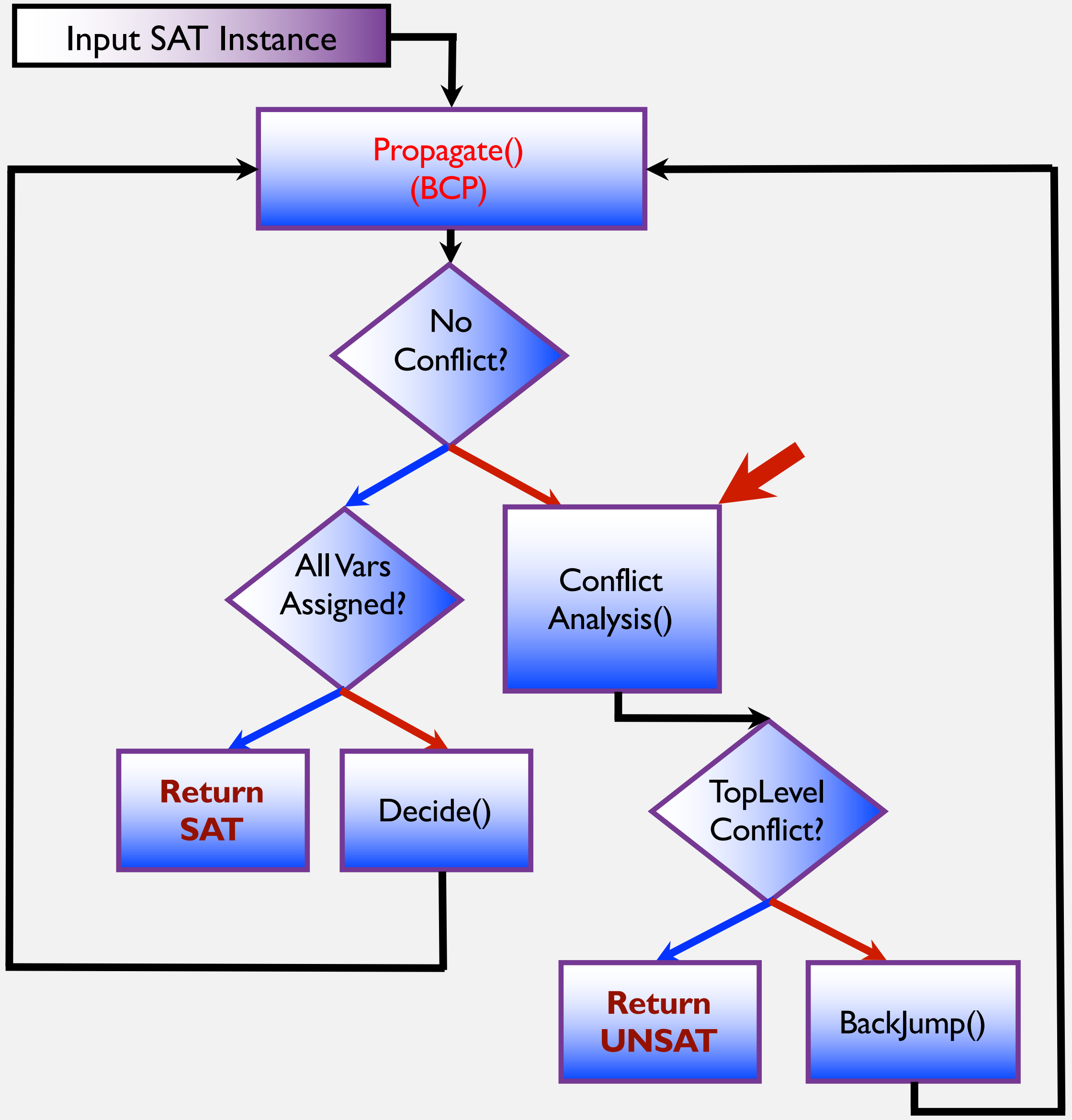
Conflict Clause (CC)

- Clause returned by the Conflict Analysis() function
- Added to conflict database (conflict DB)
- Constructed as a cut in the CG
- Implied by input formula
- Prunes the search space

Assignment Trail (AT)

- A stack of partial assignment, annotated with DL info

MODERN CDCL SAT SOLVER ARCHITECTURE
CONFLICT ANALYSIS AND CLAUSE LEARNING



Goal of conflict analysis and clause learning

- To identify the root cause of a conflict
- Learnt clause or conflict clause prunes the search space
- Adds clauses to the clause DB such that BCP can become "more complete"
- Learn a clause such that the decisions that led to the root cause are not repeated

CDCL can be shown to be equivalent to general resolution (Pipatsrisawat, Darwiche 2009. Atserias, Fichte, Thurley 2011):

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

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

MODERN CDCL SAT SOLVER ARCHITECTURE
CONFLICT ANALYSIS DETAILS: IMPLICATION GRAPH

Partial Clause DB

$W_1 = (\neg X_1 + X_2)$

$W_2 = (\neg X_1 + X_3 + X_9)$

$W_3 = (\neg X_2 + \neg X_3 + X_4)$

$W_4 = (\neg X_4 + X_5 + X_{10})$

$W_5 = (\neg X_4 + X_6 + X_{11})$

$W_6 = (\neg X_5 + \neg X_6)$

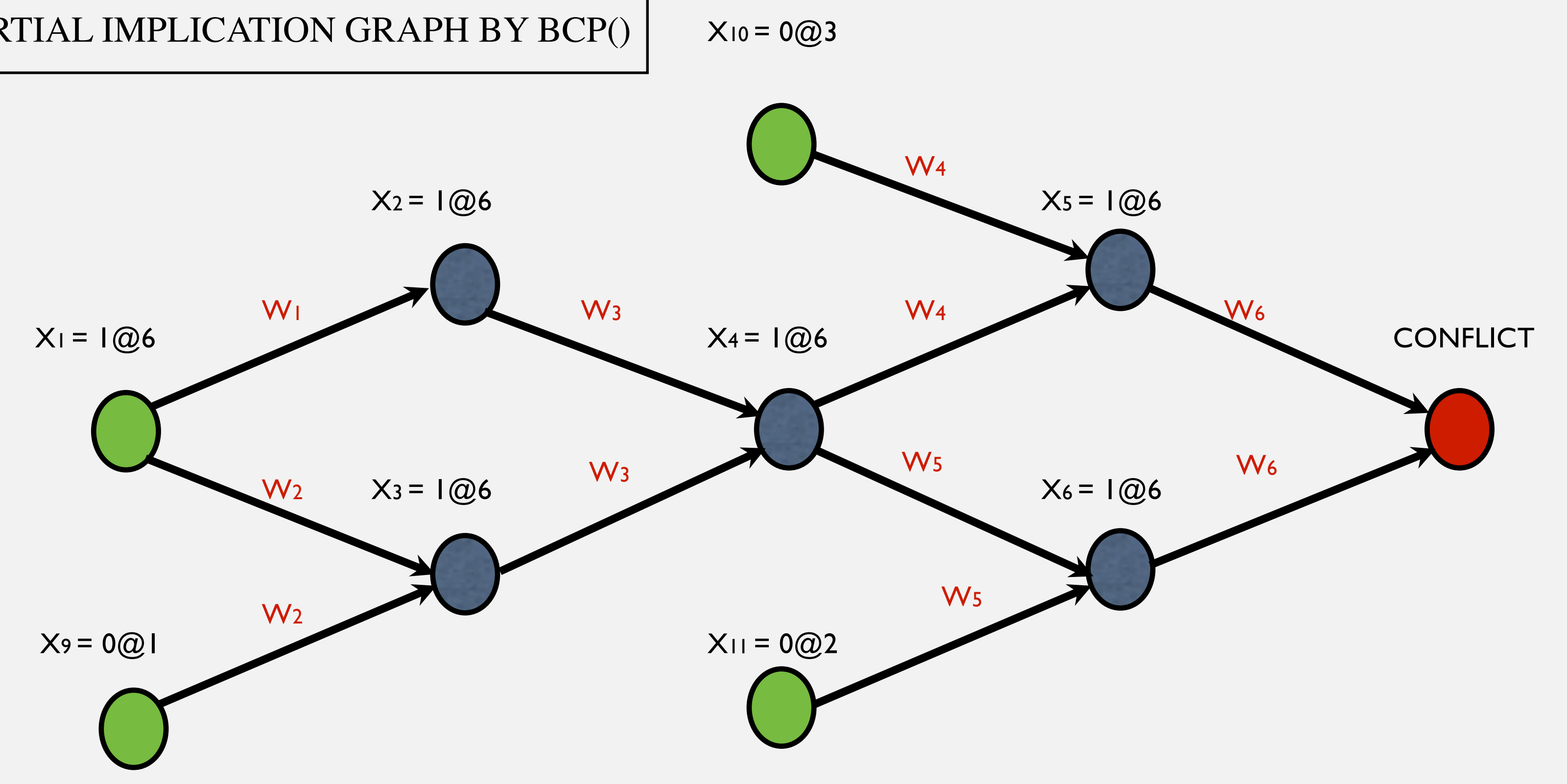
$W_7 = (X_1 + X_7 + \neg X_{12})$

$W_8 = (X_1 + X_8)$

$W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})$

Assignment trail: $\{X_9 = 0@1, X_{11} = 0@2, X_{10} = 0@3, X_{12} = 1@4, X_{13} = 1@5, ...\}$
Current decision: $\{X_1 = 1@6\}$

PARTIAL IMPLICATION GRAPH BY BCP()



MODERN CDCL SAT SOLVER ARCHITECTURE

CONFLICT ANALYSIS: DECISION LEARNING SCHEME

Partial Clause DB

$W_1 = (\neg X_1 + X_2)$

$W_2 = (\neg X_1 + X_3 + X_9)$

$W_3 = (\neg X_2 + \neg X_3 + X_4)$

$W_4 = (\neg X_4 + X_5 + X_{10})$

$W_5 = (\neg X_4 + X_6 + X_{11})$

$W_6 = (\neg X_5 + \neg X_6)$

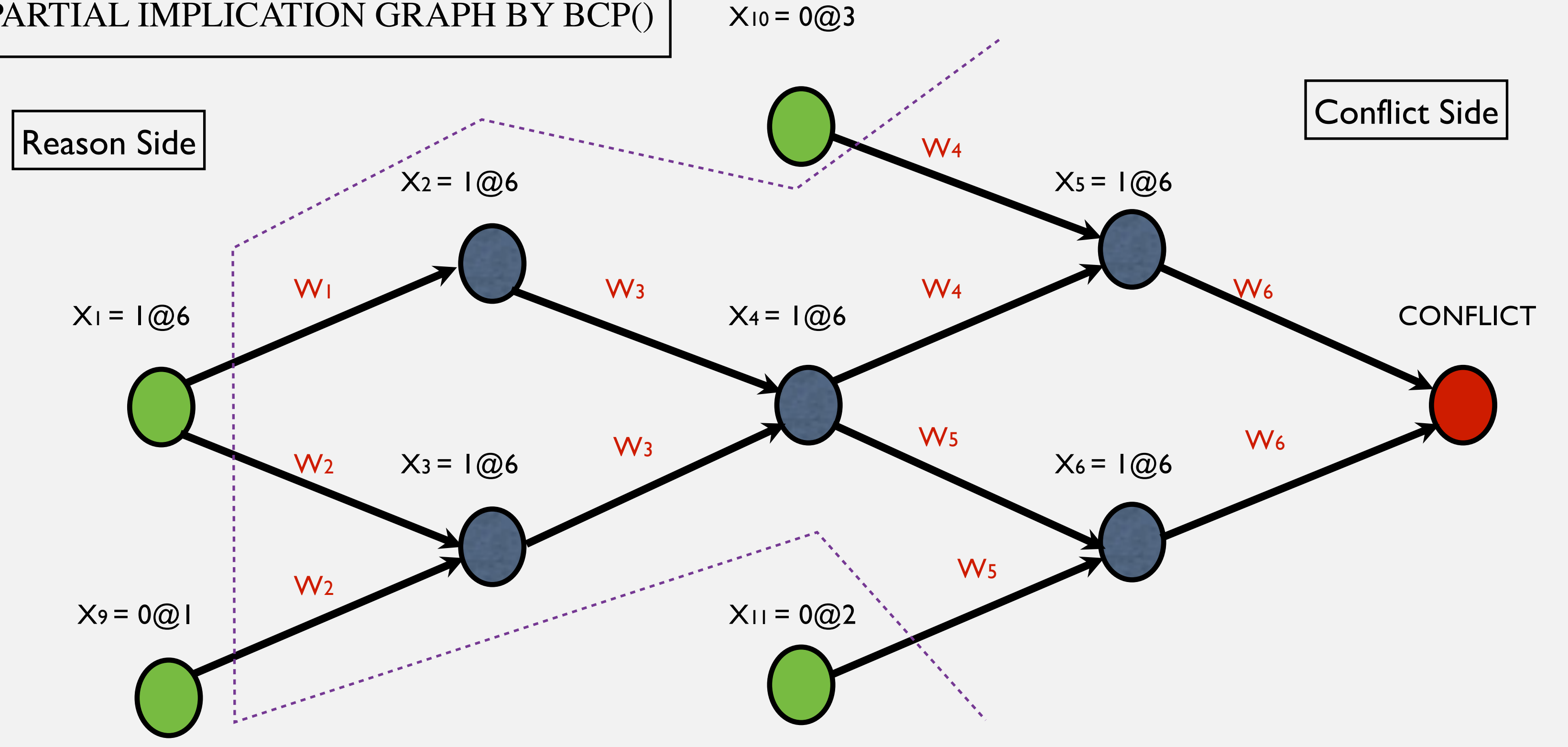
$W_7 = (X_1 + X_7 + \neg X_{12})$

$W_8 = (X_1 + X_8)$

$W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})$

Assignment trail: $\{X_9 = 0@1, X_{11} = 0@2,\ X_{10} = 0@3,\ X_{12} = 1@4,\ X_{13} = 1@5, \ldots\}$
Current decision: $\{X_1 = 1@6\}$

PARTIAL IMPLICATION GRAPH BY BCP()



Decision learning scheme identifies decision variables on the current assignment trail responsible for conflict. Learns the following conflict clause: $(\neg X_1 + X_9 + X_{10} + X_{11})$

MODERN CDCL SAT SOLVER ARCHITECTURE
CONFLICT ANALYSIS DETAILS: BACKTRACK

Partial Clause DB

$W_1 = (\neg X_1 + X_2)$

$W_2 = (\neg X_1 + X_3 + X_9)$

$W_3 = (\neg X_2 + \neg X_3 + X_4)$

$W_4 = (\neg X_4 + X_5 + X_{10})$

$W_5 = (\neg X_4 + X_6 + X_{11})$

$W_6 = (\neg X_5 + \neg X_6)$

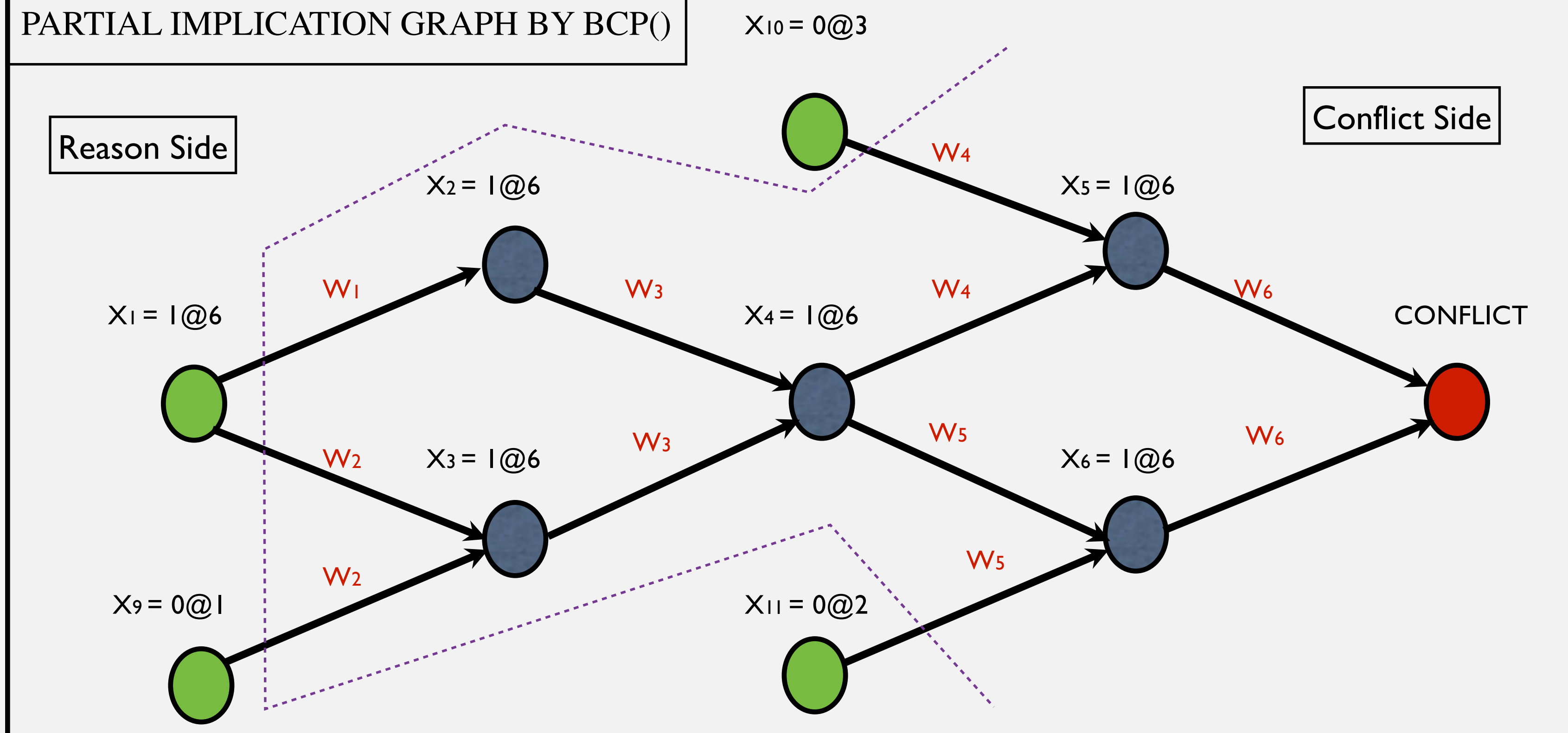
$W_7 = (X_1 + X_7 + \neg X_{12})$

$W_8 = (X_1 + X_8)$

$W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})$

Assignment trail: $\{X_9 = 0@1, X_{11} = 0@2, X_{10} = 0@3, X_{12} = 1@4, X_{13} = 1@5, \ldots\}$
Current decision: $\{X_1 = 1@6\}$

PARTIAL IMPLICATION GRAPH BY BCP()



Backtracking would undo only the last decision variable: $X_1 = 1@6$

MODERN CDCL SAT SOLVER ARCHITECTURE

CONFLICT ANALYSIS DETAILS: 1UIP LEARNING SCHEME

Partial Clause DB

$W_1 = (\neg X_1 + X_2)$

$W_2 = (\neg X_1 + X_3 + X_9)$

$W_3 = (\neg X_2 + \neg X_3 + X_4)$

$W_4 = (\neg X_4 + X_5 + X_{10})$

$W_5 = (\neg X_4 + X_6 + X_{11})$

$W_6 = (\neg X_5 + \neg X_6)$

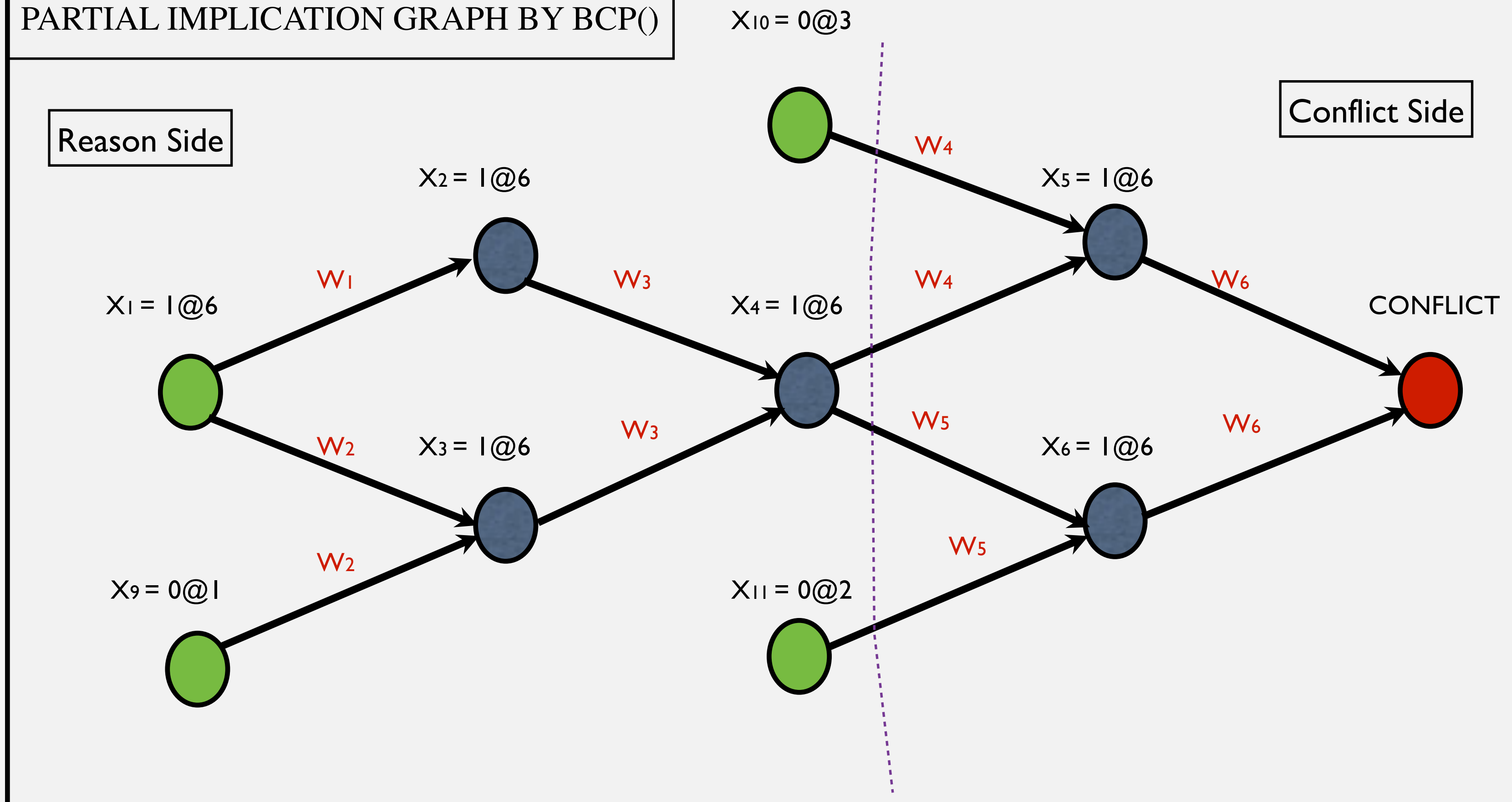
$W_7 = (X_1 + X_7 + \neg X_{12})$

$W_8 = (X_1 + X_8)$

$W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})$

Assignment trail: $\{X_9 = 0@1, X_{11} = 0@2,\ X_{10} = 0@3,\ X_{12} = 1@4,\ X_{13} = 1@5, ...\}$
Current decision: $\{X_1 = 1@6\}$

PARTIAL IMPLICATION GRAPH BY BCP()



1UIP learning scheme results in the following conflict or learnt clause:
$(\neg X_4 + X_{10} + X_{11})$

MODERN CDCL SAT SOLVER ARCHITECTURE

CONFLICT ANALYSIS DETAILS: 1UIP-DRIVEN BACKJUMP

Partial Clause DB

$W_1 = (\neg X_1 + X_2)$

$W_2 = (\neg X_1 + X_3 + X_9)$

$W_3 = (\neg X_2 + \neg X_3 + X_4)$

$W_4 = (\neg X_4 + X_5 + X_{10})$

$W_5 = (\neg X_4 + X_6 + X_{11})$

$W_6 = (\neg X_5 + \neg X_6)$

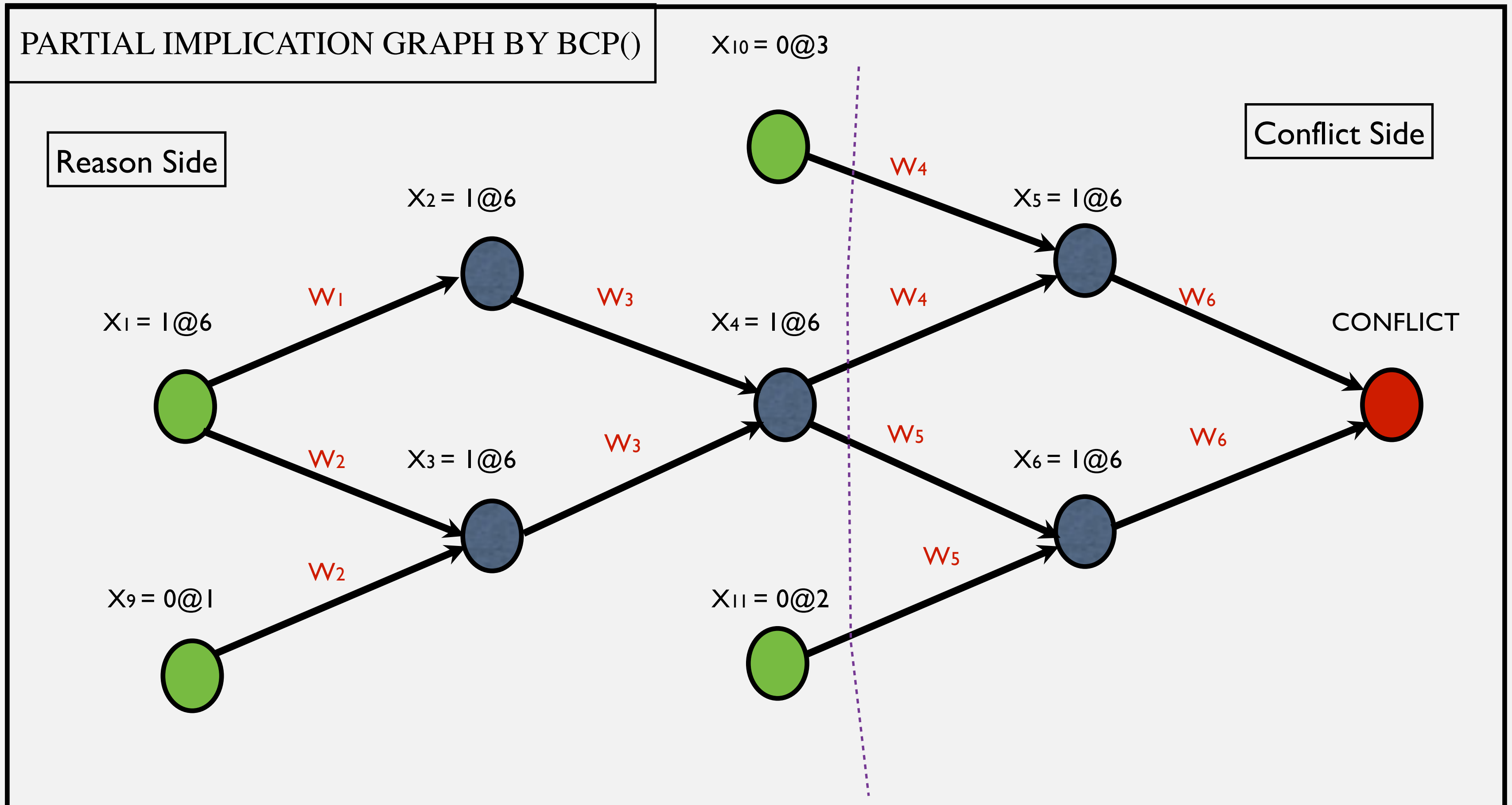
$W_7 = (X_1 + X_7 + \neg X_{12})$

$W_8 = (X_1 + X_8)$

$W_9 = (\neg X_7 + \neg X_8 + \neg X_{13})$

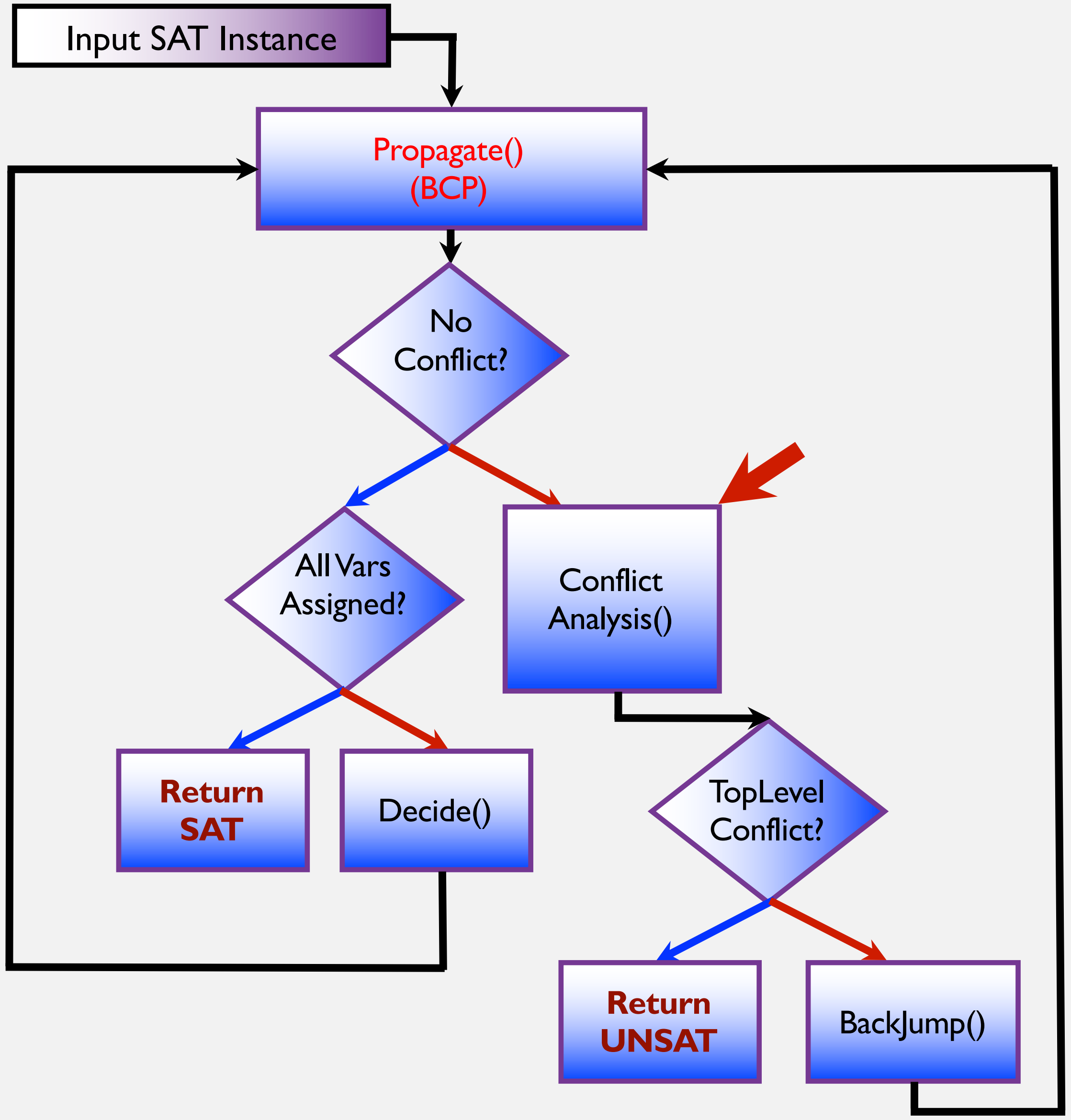
Assignment trail: $\{X_9 = 0@1, X_{11} = 0@2, X_{10} = 0@3, X_{12} = 1@4, X_{13} = 1@5, \ldots\}$
Current decision: $\{X_1 = 1@6\}$

PARTIAL IMPLICATION GRAPH BY BCP()



Backjump until the decision variable at the second-highest DL in conflict clause is unset: for this graph, the solver will backjump to DL=3.

MODERN CDCL SAT SOLVER ARCHITECTURE
RESTARTS AND CLAUSE DELETION



Restarts

- Delete the assignment trail, and start the search again
- Idea: change variable ordering and learn better clauses
- All learnt clauses and variable activity are preserved
- How to optimize restart frequency?

  Tradeoff:
  - Too frequent restarts can cause wasted effort in building search tree
  - However, frequent restarts do increase learnt clause quality

Clause deletion

- Delete clause at regular intervals
- Gets rid of useless NON-active learnt clauses
- Saves space, and propagation effort
- Very hard to accurately predict clause quality, i.e., whether or not a clause will be needed in the future
- Machine learning to predict clause quality?

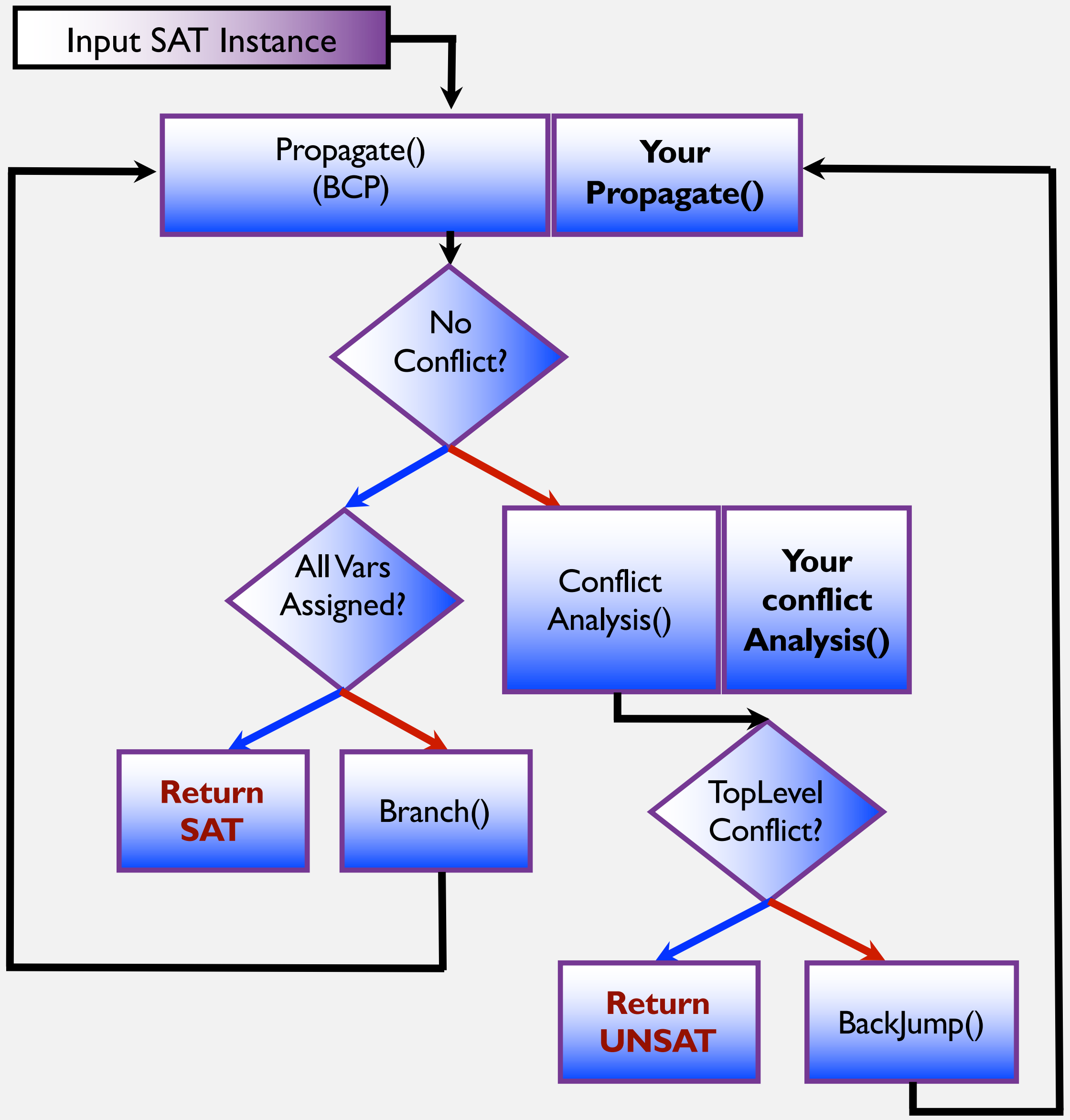
PART VI

PROGRAMMATIC SAT SOLVER: A STEP TOWARDS SMT

EXTENDING SAT WITH YOUR OWN CODE

PROGRAMMATIC SAT SOLVER ARCHITECTURE

EXTENDING SAT: A STEP TOWARDS CDCL(T)



Key steps

- Branch() or User-supplied branching routine()
- BCP Propagate() and User-supplied propagator()
- Conflict analysis and learning(), and User-supplied conflict analysis and learning()
- Backjump()
- Forget() or clause deletion()
- Restart()

All other aspects of CDCL SAT solver remain unchanged in this setting.

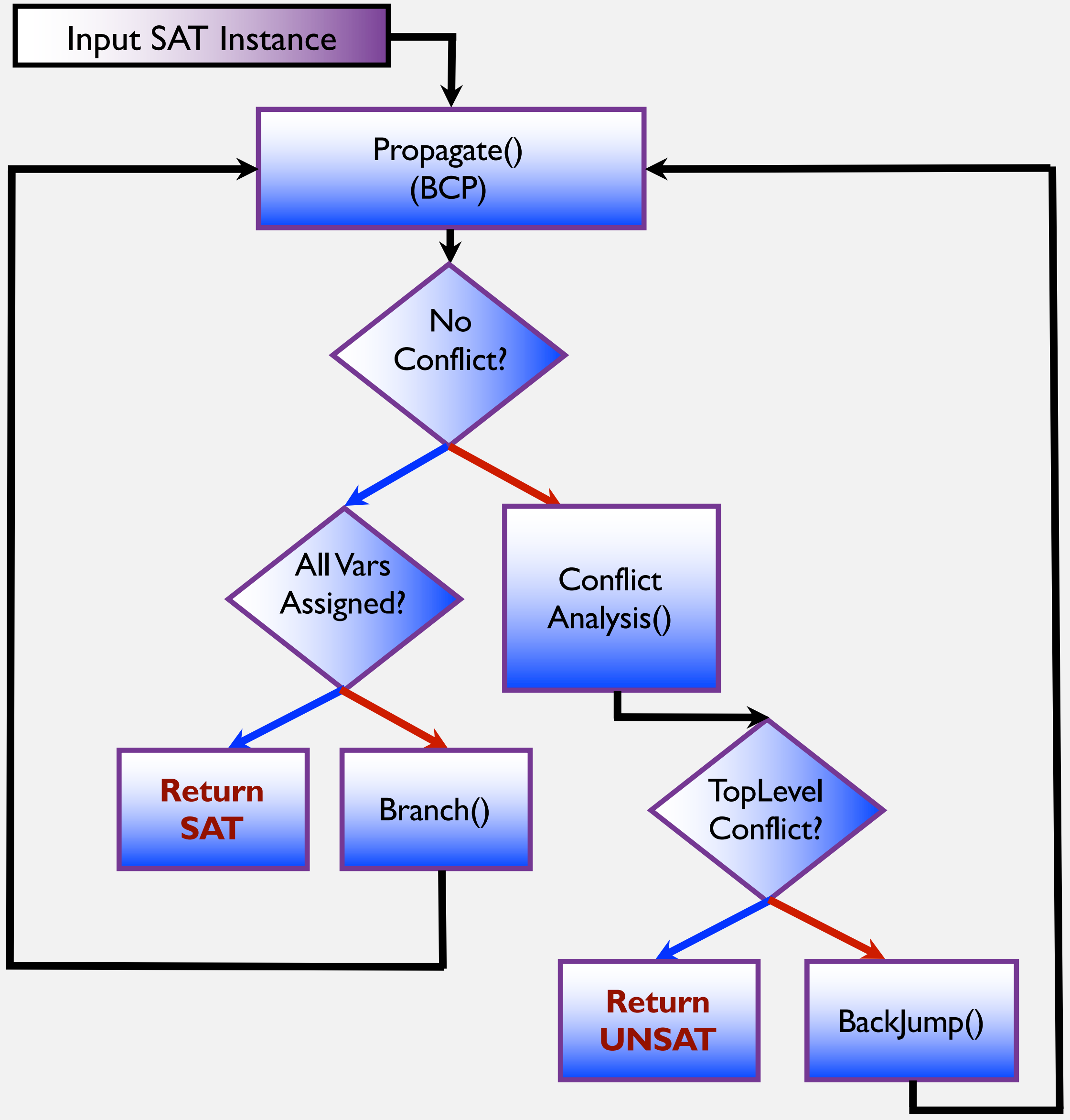
Managed to solve problems with this approach that are otherwise very difficult, e.g., inversion of SHA-1 and SHA-2 cryptographic hash functions via algebraic fault attacks using programmatic SAT solver. (Nejati, Horacek, Gebotys, G. @ CP 2018)

PART VII

CDCL SAT SOLVER

SOUNDNESS, COMPLETENESS, TERMINATION

MODERN CDCL SAT SOLVER ARCHITECTURE
SOUNDNESS, COMPLETENESS, TERMINATION



By the term ”solver” below, we are referring to a CDCL solver with perfect non-deterministic branching and restarts, and no clause deletion.

Termination: A solver is guaranteed to terminate on all inputs.

Proof sketch: The search tree for any Boolean formula is finite. The solver is designed never to repeat the same order of decisions, and hence in the worst case will terminate after an exhaustive search.

Soundness and completeness: A solver asserts that the input is UNSAT if and only if it is indeed UNSAT.

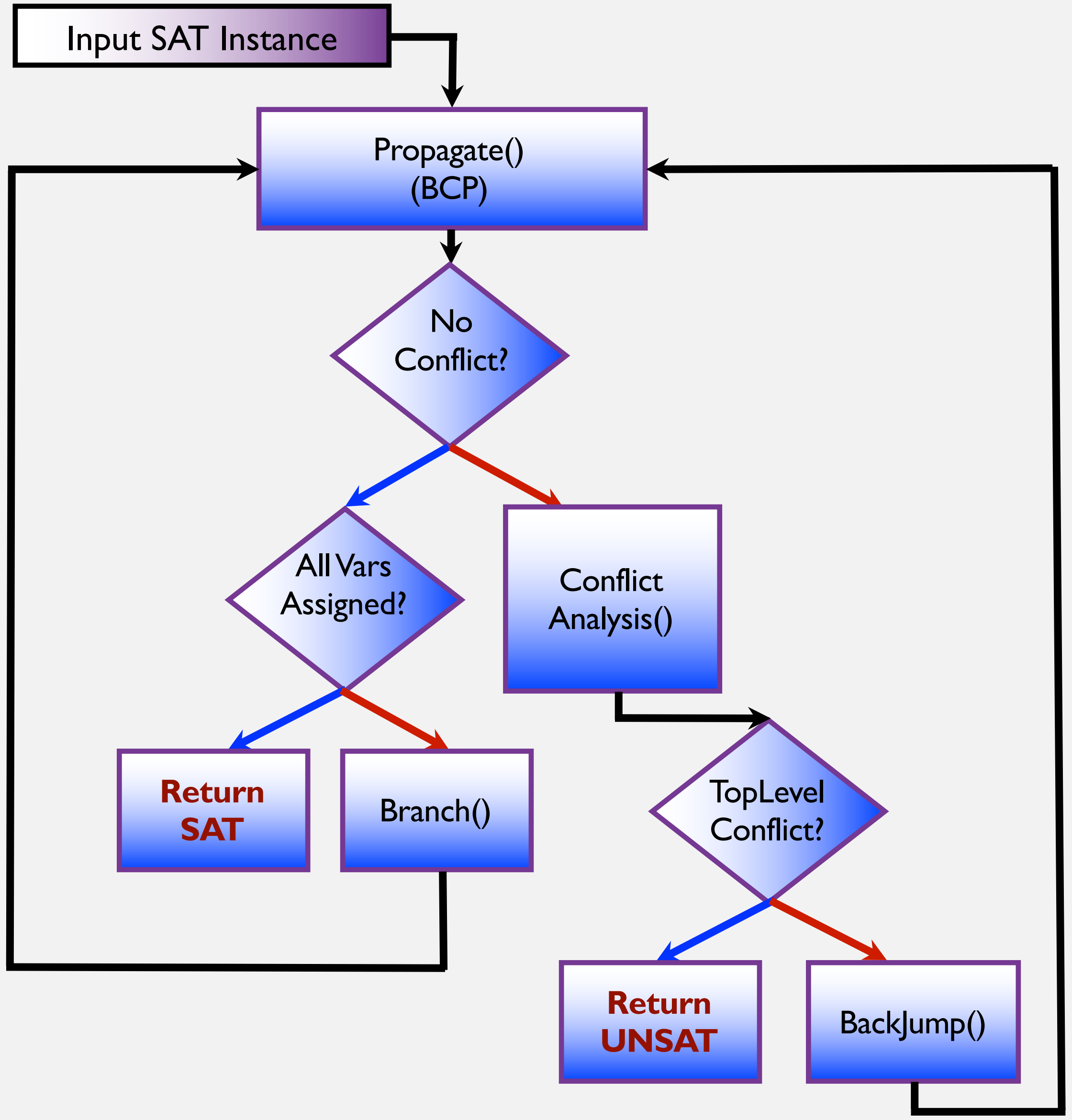
Proof sketch: It is possible to show that solvers (as proof systems) are polynomially equivalent to a sound and complete proof system for Boolean logic called “General Resolution”. (Pipatsrisawat and Darwiche 2009, Atserias, Fichte, and Thurley 2011)

PART VIII

CDCL SAT SOLVER

IMPORTANT MILESTONES

MODERN CDCL SAT SOLVER ARCHITECTURE
IMPORTANT MILESTONES



The DPLL algorithm by Davis, Putnam, Logemann, and Loveland in 1958, 1962.

Conflict-Driven Clause-Learning (CDCL) algorithm by Marques-Silva and Sakallah 1996. (AI researchers who built Truth Maintenance Systems also contributed to this idea.)

Decide/branch and efficient propagate (BCP) by Malik et al. 2001, Zabih and McAllester 1988.

Restarts by Selman & Gomes 2001.

MiniSAT by Een & Sorensson 2003.

Clause deletion by Simon and Audemard 2009.

CDCL SAT solver are equivalent to general resolution by Pipatsrisawat, Darwiche in 2009, and Atserias, Fichte, Thurley in 2011.

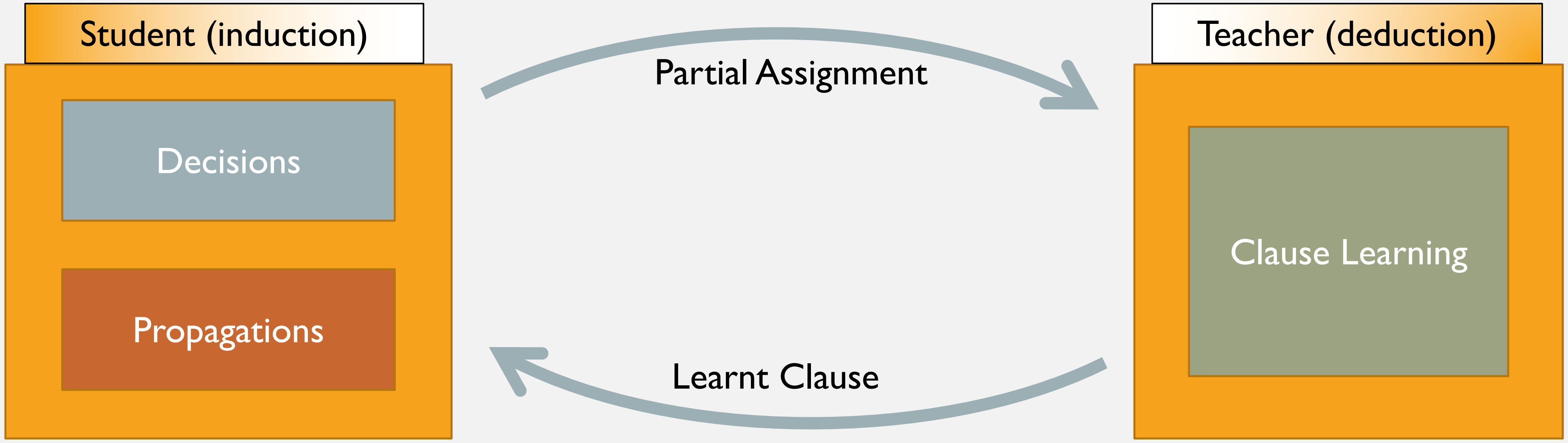
Machine learning inside SAT solvers by Liang, Poupart, G. in 2016.

PART IX

CONCLUSIONS AND TAKEAWAY

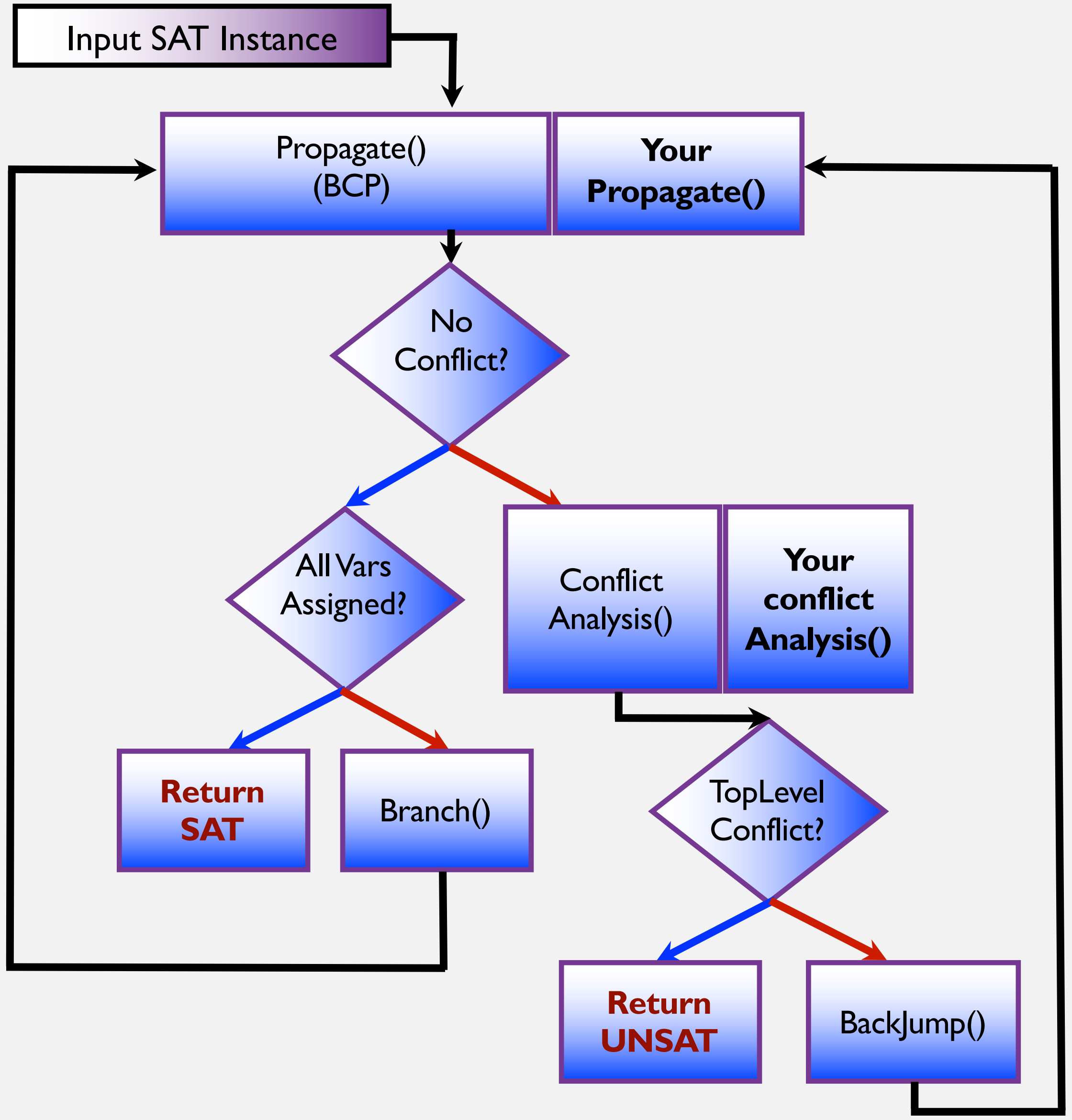
CDCL: THE STUDENT-TEACHER MODEL PROGRAMMATIC SAT

AN ABSTRACTION OF A CDCL SAT SOLVER
TOWARDS MACHINE LEARNING IN SAT



1. Learning through mistakes, and combining inductive and deductive reasoning
2. There is similar class of algorithms in reinforcement learning
3. Enabled us to design a new class of heuristics

PROGRAMMATIC SAT SOLVER ARCHITECTURE
EXTENDING SAT: A STEP TOWARDS CDCL(T)



Key steps

- Branch() or User-supplied branching routine()
- BCP Propagate() and User-supplied propagator()
- Conflict analysis and learning(), and User-supplied conflict analysis and learning()
- Backjump()
- Forget() or clause deletion()
- Restart()

All other aspects of CDCL SAT solver remain unchanged in this setting.

Managed to solve problems with this approach that are otherwise very difficult, e.g., inversion of SHA-1 and SHA-2 cryptographic hash functions via algebraic fault attacks using programmatic SAT solver. (Nejati, Horacek, Gebotys, G. @ CP 2018)

MODERN CDCL SAT SOLVER ARCHITECTURE
REFERENCES

1. Marques-Silva, J.P. and K.A. Sakallah. *GRASP: A Search Algorithm for Propositional Satisfiability*. IEEE Transactions on Computers 48(5), 1999. pp. 506-521.

2. Marques-Silva, J.P. and K.A. Sakallah. *GRASP: A Search Algorithm for Propositional Satisfiability*. ICCAD, 1996.

3. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. *CHAFF: Engineering an efficient SAT solver*. DAC, 2001, 530-535.

4. Armin Bierre, Marijn Heule, Hans van Maaren, and Toby Walsh (Editors). *Handbook of Satisfiability*. 2009. IOS Press. http://www.st.ewi.tudelft.nl/sat/handbook/

5. M. Davis, G. Logemann, and D. Loveland. *A machine program for theorem proving*. CACM 1962.

6. Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. *Learning Rate Based Branching Heuristic for SAT Solvers*. SAT 2016. https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/

7. Knot Pipatsrisawat and Adnan Darwiche. *On the Power of Clause-learning SAT Solvers as Resolution Engines*. Aritificial Intelligence journal, Volume 175(2), 2011. pp. 512-525. (Conference version in CP 2009)

8. Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. *Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution*. Journal of Aritificial Intelligence Research, volume 40, 2011. pp 353-373.

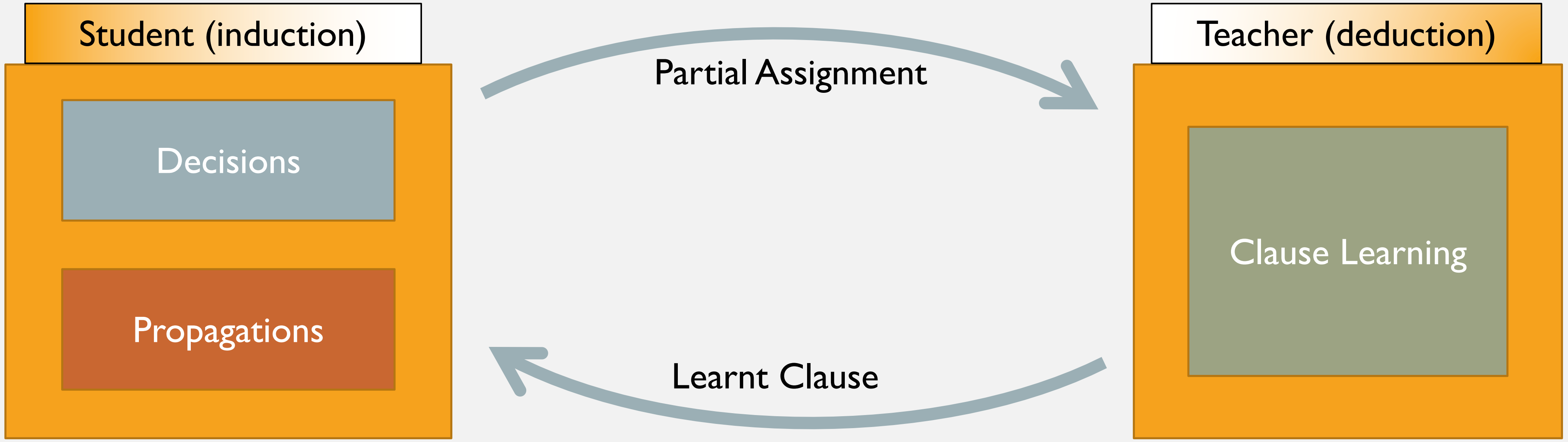
9. Saeed Nejati, Jan Horacek, Catherine Gebotys, and Vijay Ganesh. *Algebraic Fault Attack on SHA Hash Functions using Programmatic SAT Solvers*. CP 2018.

MODERN CDCL SAT SOLVER ARCHITECTURE

IMPORTANT SAT SOLVERS AND RESOURCES

1. MiniSAT: http://www.minisat.se/
2. Glucose: http://www.labri.fr/perso/lsimon/glucose/
3. MapleSAT: https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
4. Lingeling: http://fmv.jku.at/lingeling/
5. CryptoMiniSAT: https://github.com/msoos/cryptominisat/
6. SAT Competition: http://www.satcompetition.org/

QUESTIONS?



1. Learning through mistakes, and combining inductive and deductive reasoning

2. There is similar class of algorithms in reinforcement learning

3. Enabled us to design a new class of heuristics