CIS 1921

Lecture 4: Modern Techniques in SAT Solving

Recap: DPLL (Pseudocode)

```
dpll(φ):
    if φ = ∅: return TRUE
    if ϵ ∈ φ: return FALSE
    if φ contains unit clause {ℓ}:
        return dpll(φ|ℓ)
    let x = pick_variable(φ)
    return dpll(φ|x) OR dpll(φ|x̄)
```

Recap: Iterative DPLL

```
dpll(φ):
    if unit_propagate() = CONFLICT: return UNSAT
    while not all variables have been set:
        let x = pick_variable()
        create new decision level
        set x = T
        while unit_propagate() = CONFLICT:
            if decision_level = 0: return UNSAT
            backtrack()
            set x = F
    return SAT
```

Chronological Backtracking

- DPLL uses **chronological backtracking:** when we find a conflict, backtrack to the *previous* decision level
- **Issue:** might reach conflicts (contradictions) caused by the same underlying reason over and over again

Chronological Backtracking

$$(1 \vee \overline{2})$$

$$(\overline{1} \vee 3 \vee 4)$$

$$(\overline{1} \vee \overline{3} \vee 4)$$

$$(\overline{1} \vee 3 \vee \overline{4})$$

$$(\overline{1} \vee \overline{3} \vee \overline{4})$$

Chronological Backtracking

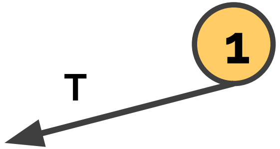
$$\left(1 \vee \overline{2}\right)$$

$$\left(\overline{1} \vee 3 \vee 4\right)$$

$$\left(\overline{1} \vee \overline{3} \vee 4\right)$$

$$\left(\overline{1} \vee 3 \vee \overline{4}\right)$$

$$\left(\overline{1} \vee \overline{3} \vee \overline{4}\right)$$



Chronological Backtracking

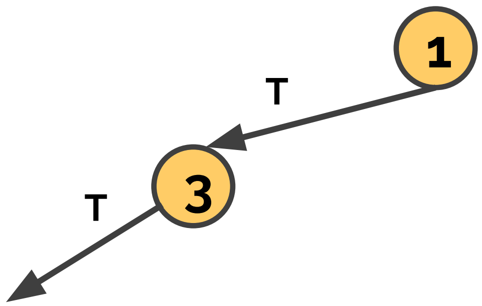
$$\left(1 \vee \overline{2}\right)$$

$$\left(\overline{1} \vee 3 \vee 4\right)$$

$$\left(\overline{1} \vee \overline{3} \vee 4\right)$$

$$\left(\overline{1} \vee 3 \vee \overline{4}\right)$$

$$\left(\overline{1} \vee \overline{3} \vee \overline{4}\right)$$



Chronological Backtracking

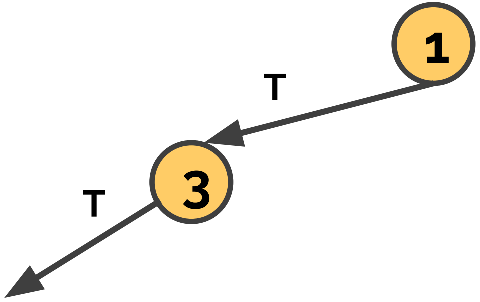
$(1 \vee \overline{2})$

$(\overline{1} \vee 3 \vee 4)$

$(\overline{1} \vee \overline{3} \vee 4)$ UNIT

$(\overline{1} \vee 3 \vee \overline{4})$

$(\overline{1} \vee \overline{3} \vee \overline{4})$



Chronological Backtracking

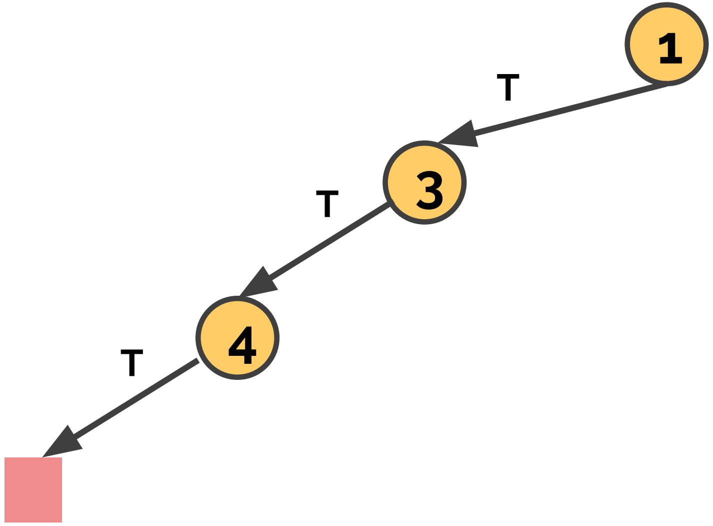
$$\left(1 \vee \overline{2}\right)$$

$$\left(\overline{1} \vee 3 \vee 4\right)$$

$$\left(\overline{1} \vee \overline{3} \vee 4\right)$$

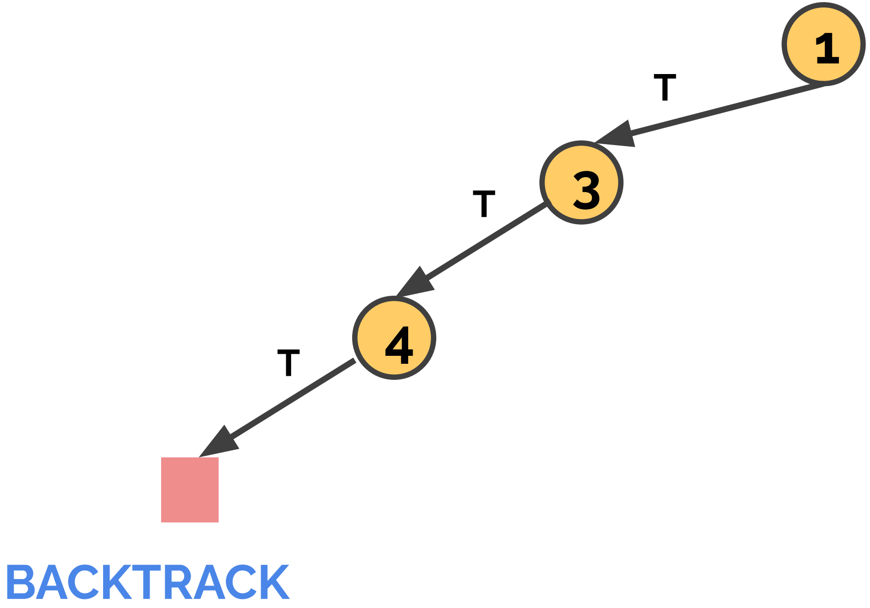
$$\left(\overline{1} \vee 3 \vee \overline{4}\right)$$

$$\left(\overline{1} \vee \overline{3} \vee \overline{4}\right)$$



Chronological Backtracking

$$(1 \vee \overline{2})$$
$$(\overline{1} \vee 3 \vee 4)$$
$$(\overline{1} \vee \overline{3} \vee 4)$$
$$(\overline{1} \vee 3 \vee \overline{4})$$
$$(\overline{1} \vee \overline{3} \vee \overline{4})$$



Chronological Backtracking

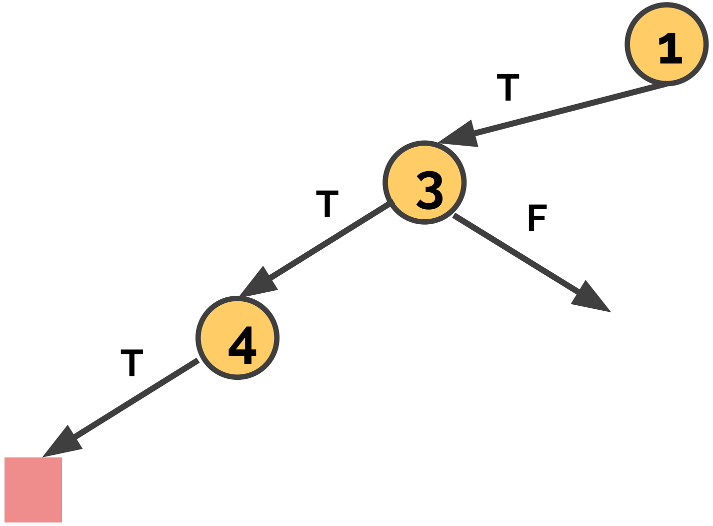
$$\left(1 \vee \overline{2}\right)$$

$$\left(\overline{1} \vee 3 \vee 4\right)$$

$$\left(\overline{1} \vee \overline{3} \vee 4\right)$$

$$\left(\overline{1} \vee 3 \vee \overline{4}\right)$$

$$\left(\overline{1} \vee \overline{3} \vee \overline{4}\right)$$



Chronological Backtracking

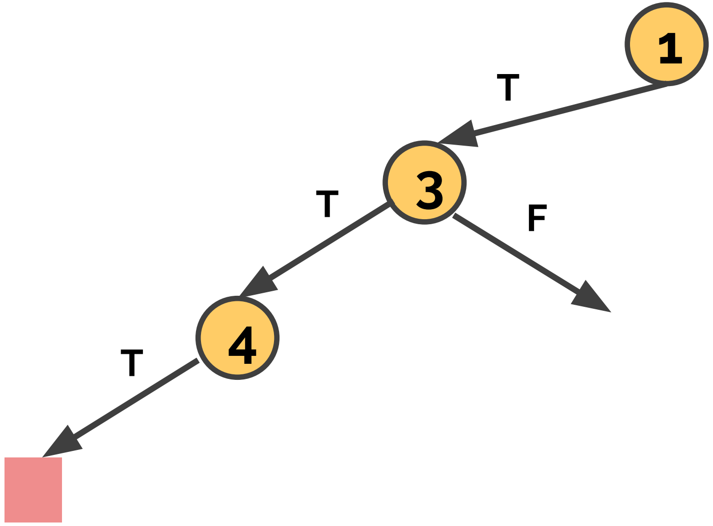
$(\boxed{1} \vee \bar{2})$

$(\boxed{\bar{1}} \vee \boxed{3} \vee 4)$ UNIT

$(\boxed{\bar{1}} \vee \boxed{\bar{3}} \vee 4)$

$(\boxed{\bar{1}} \vee \boxed{3} \vee \bar{4})$

$(\boxed{\bar{1}} \vee \boxed{\bar{3}} \vee \bar{4})$



Chronological Backtracking

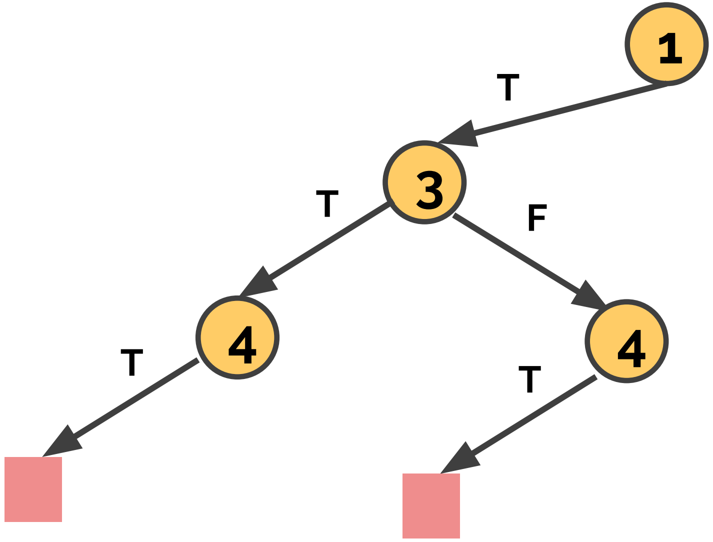
$(1 \vee \overline{2})$

$(\overline{1} \vee 3 \vee 4)$

$(\overline{1} \vee \overline{3} \vee 4)$

$(\overline{1} \vee 3 \vee \overline{4})$

$(\overline{1} \vee \overline{3} \vee \overline{4})$



Chronological Backtracking

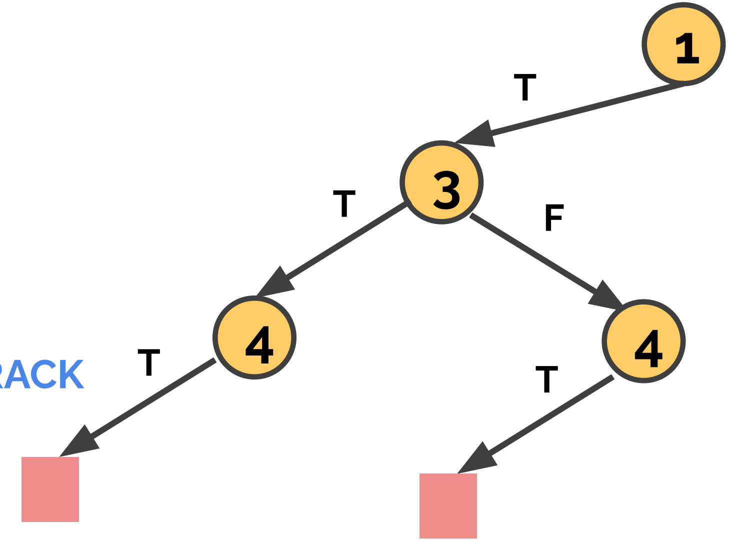
$$\left(1 \vee \overline{2}\right)$$

$$\left(\overline{1} \vee 3 \vee 4\right)$$

$$\left(\overline{1} \vee \overline{3} \vee 4\right)$$

$$\left(\overline{1} \vee 3 \vee \overline{4}\right)$$ BACKTRACK

$$\left(\overline{1} \vee \overline{3} \vee \overline{4}\right)$$



Chronological Backtracking

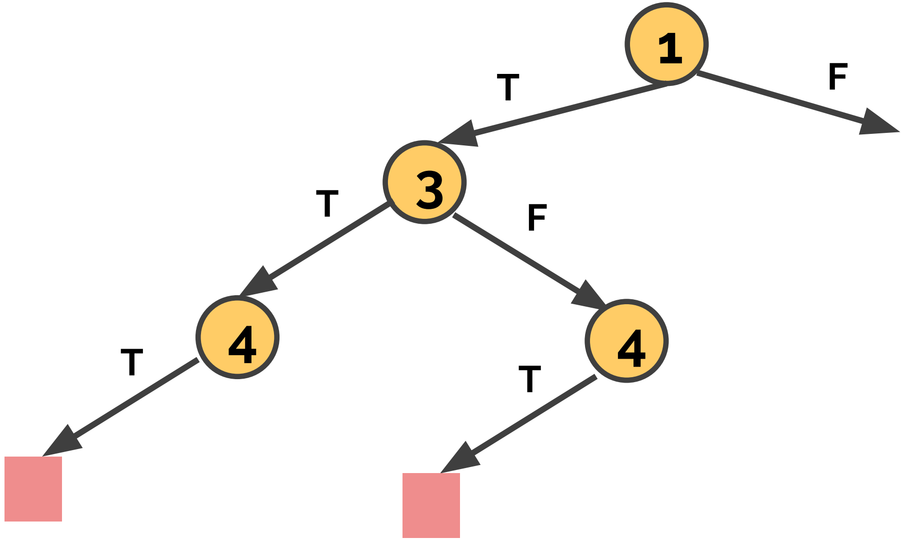
$$\left(1 \lor \overline{2}\right)$$

$$\left(\overline{1} \lor 3 \lor 4\right)$$

$$\left(\overline{1} \lor \overline{3} \lor 4\right)$$

$$\left(\overline{1} \lor 3 \lor \overline{4}\right)$$

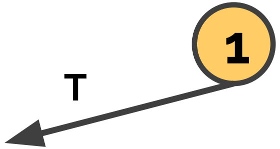
$$\left(\overline{1} \lor \overline{3} \lor \overline{4}\right)$$



Chronological Backtracking

$$(1 \vee \overline{2})$$
$$(\overline{1} \vee 3 \vee 4)$$
$$(\overline{1} \vee \overline{3} \vee 4)$$
$$(\overline{1} \vee 3 \vee \overline{4})$$
$$(\overline{1} \vee \overline{3} \vee \overline{4})$$

UNSAT subformula



Chronological Backtracking

$$(1 \vee \overline{2})$$

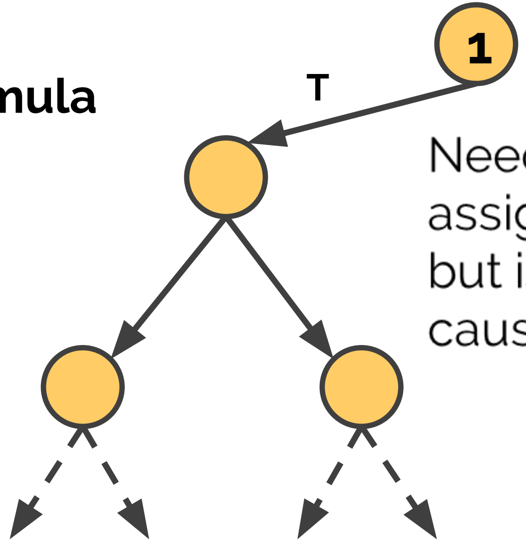
$$(\overline{1} \vee 3 \vee 4)$$

$$(\overline{1} \vee \overline{3} \vee 4)$$

$$(\overline{1} \vee 3 \vee \overline{4})$$

$$(\overline{1} \vee \overline{3} \vee \overline{4})$$

UNSAT subformula



Need to rule out all assignments of 2, 3, 4, but issue was really caused by $1 = T$!

Backjumping

- Not every decision actually contributes to a conflict
- **Idea:** upon conflict, instead of backtracking one level to the last decision, **backjump** to an *important* decision
    - i.e., a decision that contributed to the conflict
- But how do we know what is an important decision?

Decision Levels

What are the different ways in which variables get assigned in DPLL?

Decision Levels

- A **decision** refers to any time our algorithm *arbitrarily* assigns a variable (without being forced to do so)
    - Selecting a literal and assigning it True is a decision
    - Unit propagation & reassigning selected literal after backtracking are not decisions
- All assignments implied by the i^{th} decision are said to be on the i^{th} **decision level**
    - Can *assignments* ever be on the zeroth decision level?

Decision Levels

When we backtrack, all assignments made at the current decision level get unassigned.

Example: DPLL

$(\overline{1} \vee \overline{2})$

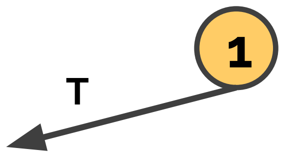
$(\overline{1} \vee 2)$

$(1 \vee \overline{2} \vee 3)$

$(1 \vee 2 \vee \overline{4})$

Unit!

Steps



1	2	3	4
T			

Example: DPLL

$(\overline{1} \lor \overline{2})$

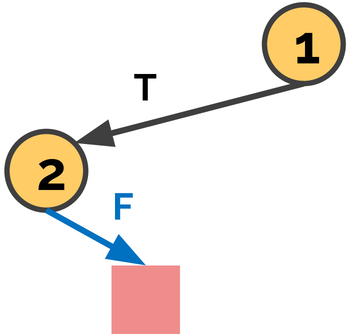
$(\overline{1} \lor 2)$

$(1 \lor \overline{2} \lor 3)$

$(1 \lor 2 \lor \overline{4})$

Conflict!

Steps



1	2	3	4
T	F		

Example: DPLL

$(\overline{1} \vee \overline{2})$

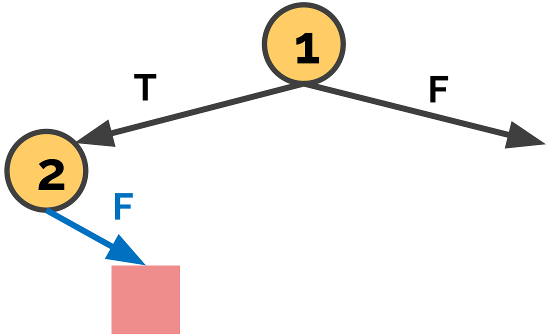
$(\overline{1} \vee 2)$

$(1 \vee \overline{2} \vee 3)$

$(1 \vee 2 \vee \overline{4})$

1	2	3	4
F			

Steps



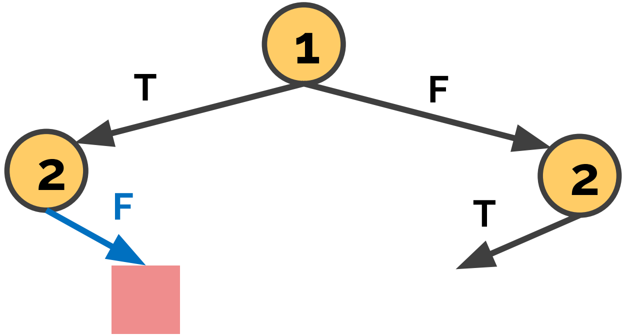
Example: DPLL

$$(\overline{1} \vee \overline{2})$$
$$(\overline{1} \vee 2)$$
$$(1 \vee \overline{2} \vee 3)$$
$$(1 \vee 2 \vee \overline{4})$$

Unit!

1	2	3	4
F	T		

Steps



Example: DPLL

$(\bar{1} \vee \bar{2})$

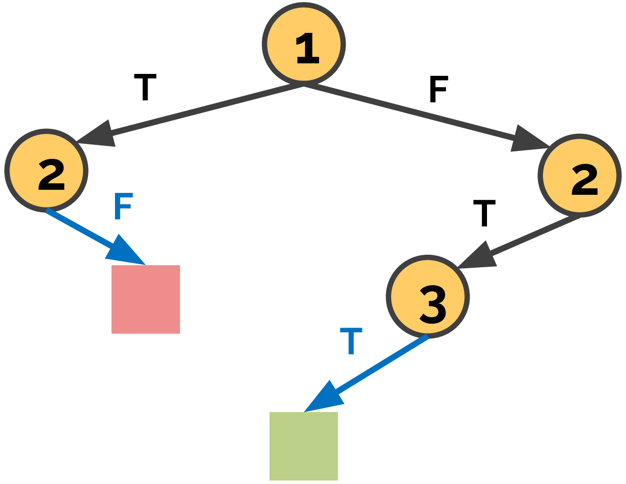
$(\bar{1} \vee 2)$

$(1 \vee \bar{2} \vee 3)$

$(1 \vee 2 \vee \bar{4})$

1	2	3	4
F	T	T	

Steps



Implication Graphs

- An **implication graph** G is a DAG whose vertices are literal assignments at a particular decision level, as well as a time stamp of when the vertex was created
  - Ex: $\overline{x}@3$, t = 4: represents setting x to False at decision level 3, and the vertex was created at time = 4
  - Assignments can be decisions or due to unit propagation/backtracking
  - Start at t = 1, and every time a new vertex is added, increment t by 1.
- Can also contain special vertex $\perp$ representing a conflict
- There is an edge $x@i \rightarrow y@j$ if the assignment $x@i$ directly implied the assignment $y@j$
  - i.e., $y@j$ was set by unit propagation from a clause containing $\overline{x}$

Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

Implication Graphs

c_1: $\overline{2} \lor \overline{3} \lor \overline{4}$

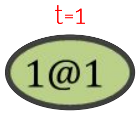
c_2: $\overline{3} \lor \overline{5} \lor \overline{6}$

c_3: $4 \lor 6 \lor 7$

c_4: $\overline{7} \lor \overline{8}$

c_5: $\overline{1} \lor \overline{7} \lor \overline{9}$

c_6: $\overline{1} \lor 8 \lor 9$



Implication Graphs

c_1: $\overline{2} \lor \overline{3} \lor \overline{4}$

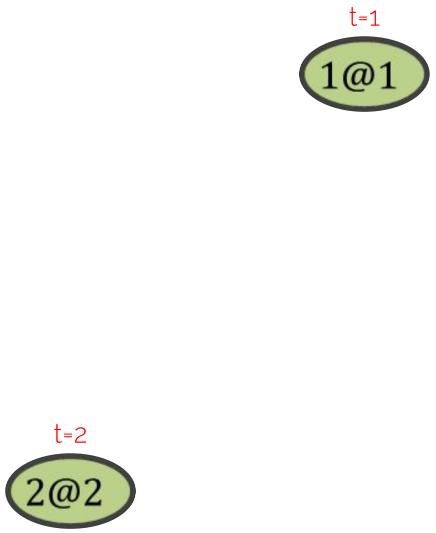
c_2: $\overline{3} \lor \overline{5} \lor \overline{6}$

c_3: $4 \lor 6 \lor 7$

c_4: $\overline{7} \lor \overline{8}$

c_5: $\overline{1} \lor \overline{7} \lor \overline{9}$

c_6: $\overline{1} \lor 8 \lor 9$



Implication Graphs

$c_1: \quad \overline{2} \vee \overline{3} \vee \overline{4}$

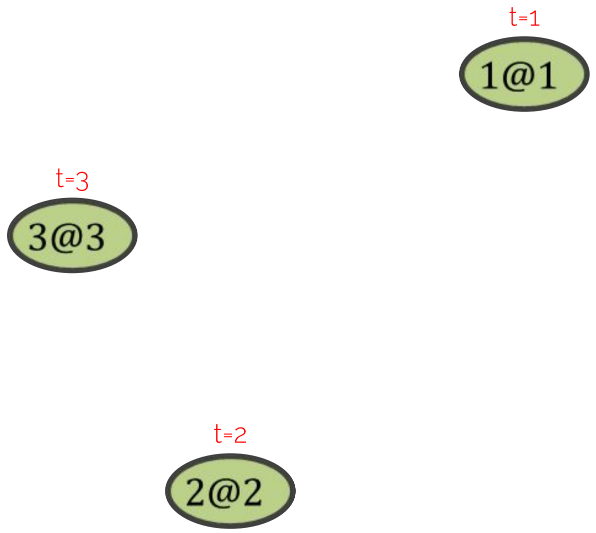
$c_2: \quad \overline{3} \vee \overline{5} \vee \overline{6}$

$c_3: \quad 4 \vee 6 \vee 7$

$c_4: \quad \overline{7} \vee \overline{8}$

$c_5: \quad \overline{1} \vee \overline{7} \vee \overline{9}$

$c_6: \quad \overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\boxed{\overline{2}} \vee \boxed{\overline{3}} \vee \boxed{\overline{4}}$

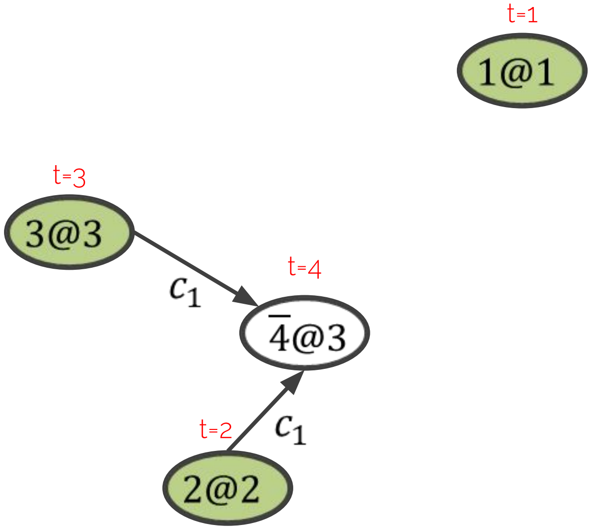
c_2: $\boxed{\overline{3}} \vee \overline{5} \vee \overline{6}$

c_3: $\boxed{4} \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\boxed{\overline{1}} \vee \overline{7} \vee \overline{9}$

c_6: $\boxed{\overline{1}} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

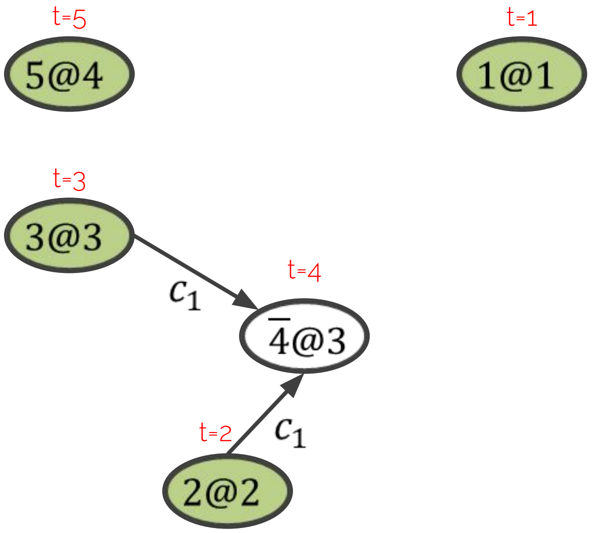
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

$$c_1: \quad \overline{2} \vee \overline{3} \vee \overline{4}$$

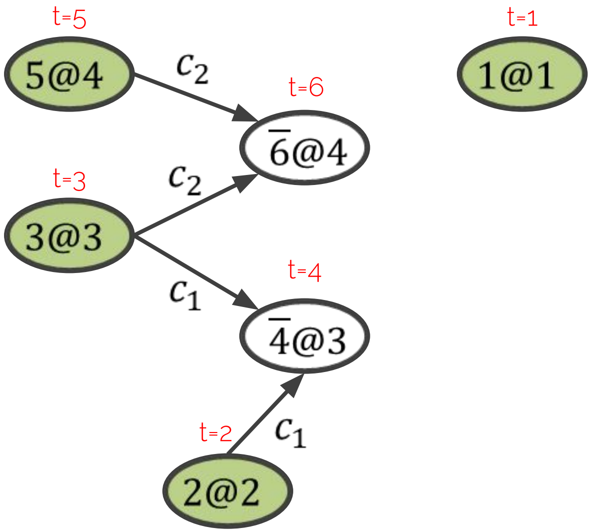
$$c_2: \quad \overline{3} \vee \overline{5} \vee \overline{6}$$

$$c_3: \quad 4 \vee 6 \vee 7$$

$$c_4: \quad \overline{7} \vee 8$$

$$c_5: \quad \overline{1} \vee \overline{7} \vee \overline{9}$$

$$c_6: \quad \overline{1} \vee 8 \vee 9$$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

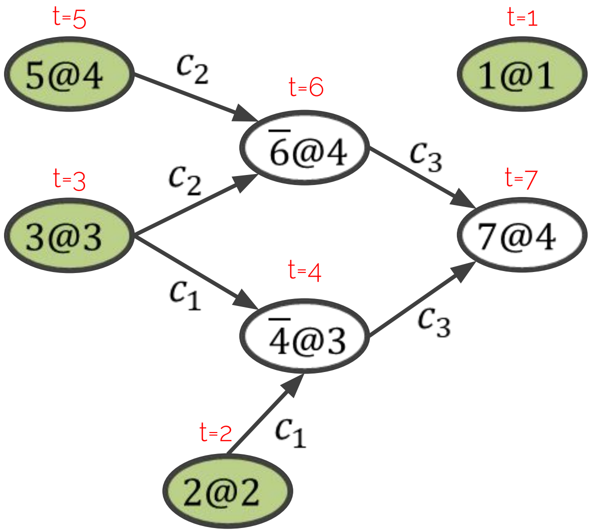
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

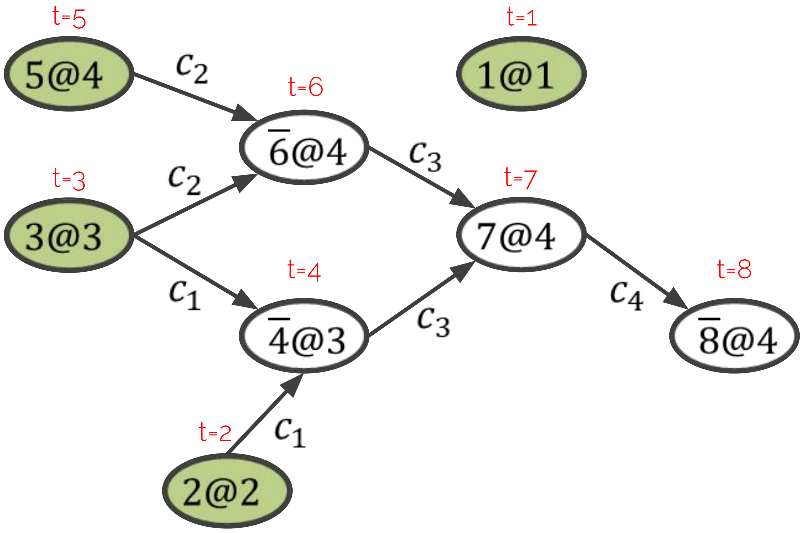
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$



Implication Graphs

c_1: $\overline{2} \lor \overline{3} \lor \overline{4}$

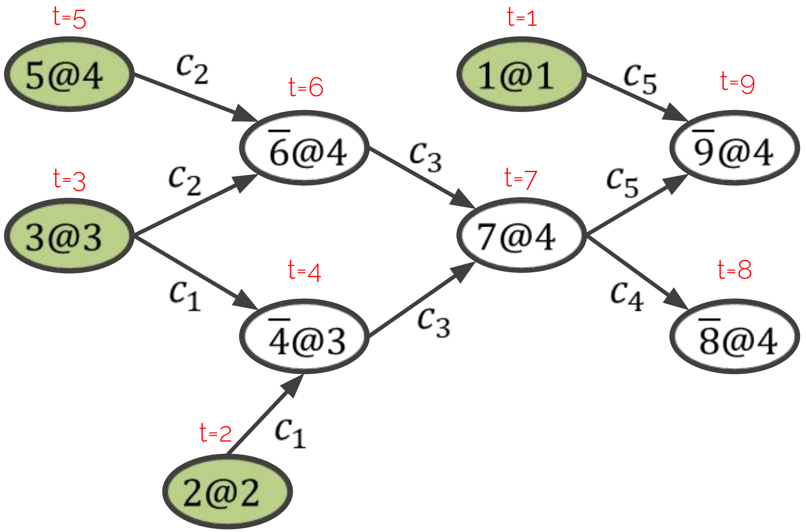
c_2: $\overline{3} \lor \overline{5} \lor \overline{6}$

c_3: $4 \lor 6 \lor 7$

c_4: $\overline{7} \lor \overline{8}$

c_5: $\overline{1} \lor \overline{7} \lor \overline{9}$

c_6: $\overline{1} \lor 8 \lor 9$



Implication Graphs

c_1: $\overline{2} \lor \overline{3} \lor \overline{4}$

c_2: $\overline{3} \lor \overline{5} \lor \overline{6}$

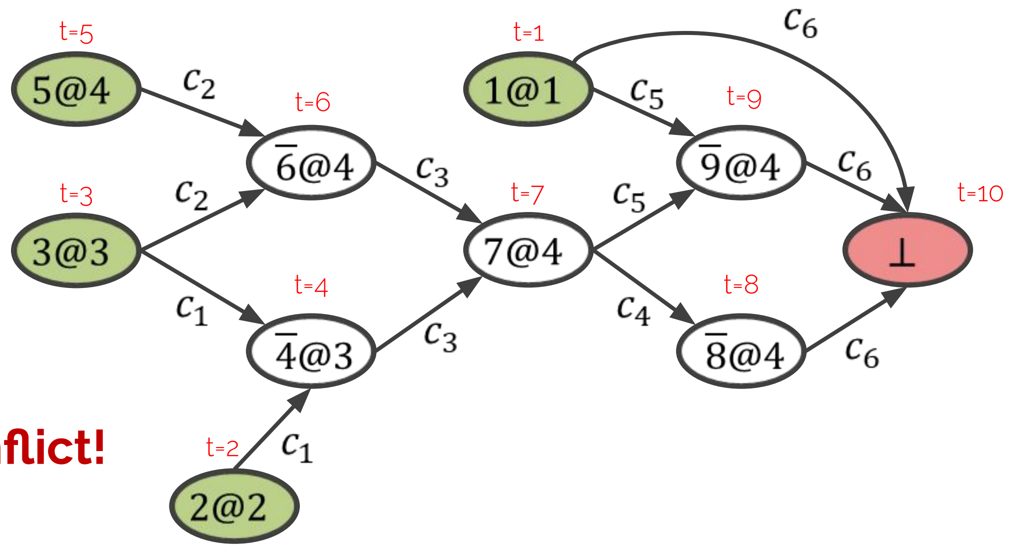
c_3: $4 \lor 6 \lor 7$

c_4: $\overline{7} \lor \overline{8}$

c_5: $\overline{1} \lor \overline{7} \lor \overline{9}$

c_6: $\overline{1} \lor 8 \lor 9$

Conflict!

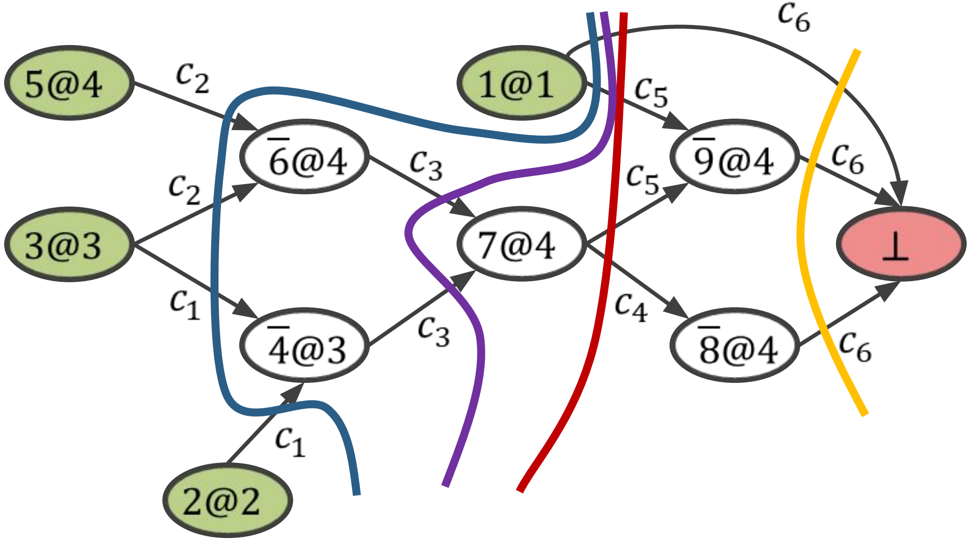


Conflicts

- A **conflict set** of assignments (collectively) imply a conflict
- A **conflict cut** in an implication graph is a bipartition of the vertices $V = R \cup C$ such that:
  - *Reason side* R contains all decisions (source nodes)
  - *Conflict side* C contains the conflict node (a sink)
  - No edges cross $C \rightarrow R$, only $R \rightarrow C$
- The set of vertices with an outgoing edge crossing a given conflict cut forms a conflict set

Ex: Conflict Cuts

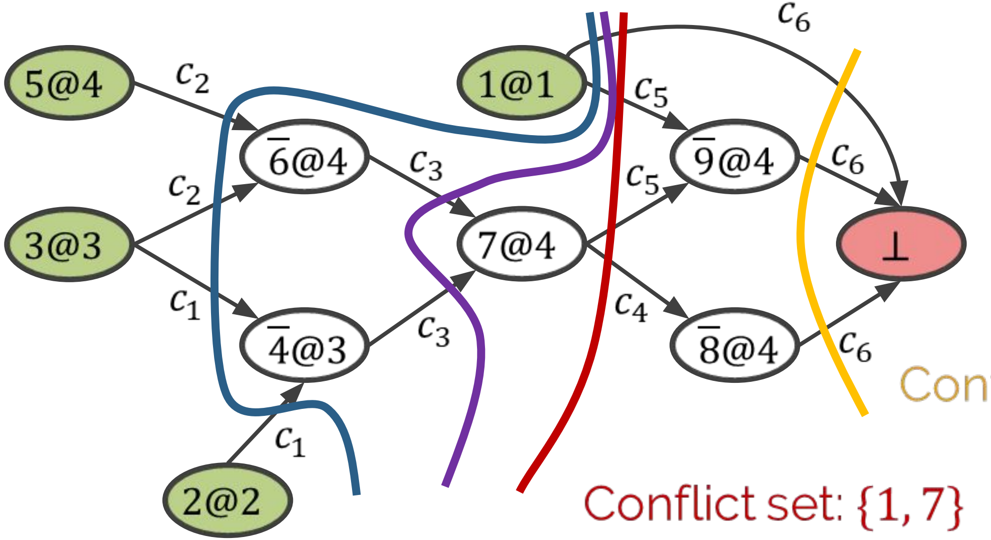
REASON SIDE



CONFLICT SIDE

Ex: Conflict Cuts

REASON SIDE



CONFLICT SIDE

Conflict set: $\{1, \overline{8}, \overline{9}\}$

Conflict set: $\{1, 7\}$

Conflict set: $\{1, 2, 3, 5\}$

Conflict set: $\{1, \overline{4}, \overline{6}\}$

And more...

Note that a conflict set is a subset of our current assignments.

Note that a conflict set is a subset of our current assignments.

"Once we have found a conflict set (aka a subset of bad assignments), we never want to revisit this set of bad assignments in the future."

Clause Learning

- **Observation:** Given a conflict set $\{x_1, \overline{x_2}, x_3, \ldots, x_k\}$, we know that in a plausible satisfying assignment, at least one "mismatch" must exit.
- Can derive the **conflict clause** $(\overline{x_1} \lor x_2 \lor \overline{x_3} \lor \ldots \lor \overline{x_k})$
- **Conflict-driven clause learning (CDCL):** add conflict clauses to the original CNF we're solving
    - Introduced by GRASP (1996); revolutionized SAT solving
    - Many solvers have **aggressive deletion policy** for long, "inactive," "unhelpful" learned clauses – avoid explosion in CNF size

$c_1: \overline{2} \vee \overline{3} \vee \overline{4}$

$c_2: \overline{3} \vee \overline{5} \vee \overline{6}$

$c_3: 4 \vee 6 \vee 7$

$c_4: \overline{7} \vee \overline{8}$

$c_5: \overline{1} \vee \overline{7} \vee \overline{9}$

$c_6: \overline{1} \vee 8 \vee 9$

Conflict set: $\{1, \overline{4}, \overline{6}\}$

$\rightarrow$

$c_1: \overline{2} \vee \overline{3} \vee \overline{4}$

$c_2: \overline{3} \vee \overline{5} \vee \overline{6}$

$c_3: 4 \vee 6 \vee 7$

$c_4: \overline{7} \vee \overline{8}$

$c_5: \overline{1} \vee \overline{7} \vee \overline{9}$

$c_6: \overline{1} \vee 8 \vee 9$

$c_7: (\overline{1} \vee 4 \vee 6)$

Asserting Clauses

- Many conflict cuts – how do we decide which to choose to build a conflict clause?
- **Goal:** after backjumping, be able to apply new knowledge from learned clause right away
    - Want learned clause to become a unit clause right after backjumping

Asserting Clauses

- A learned clause is **asserting** if it contains only one variable set on the same decision level as conflict
- **Observation:** iff a clause is asserting, it will become a unit clause after backtracking
- How far can we backjump and still have asserting clauses become unit clauses?
  - Backjump to *second-largest* (i.e., deepest) decision level in asserting clause (or zeroth level if asserting clause has size 1)
    - i.e., return to that decision level (don't undo the decision)
  - Called the **asserting level**

CDCL (Pseudocode)

```
cdcl(φ):
    if unit_propagate() = CONFLICT: return UNSAT
    while not all variables have been set:
        let x = pick_variable()
        create new decision level; set x = T
        while unit_propagate() = CONFLICT:
            if level = 0: return UNSAT
            let (conflict_cls, assrt_lvl) = analyze_conflict()
            let φ = φ ∪ { conflict_cls }
            # discard all assignments after asserting level
            backjump(assrt_lvl)
    return SAT
```

Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

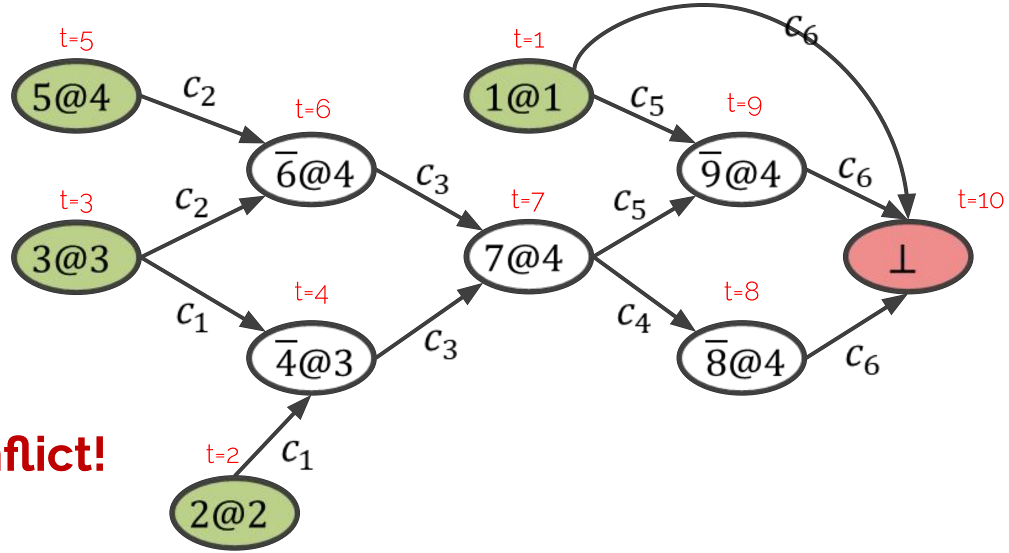
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$ **Conflict!**



Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

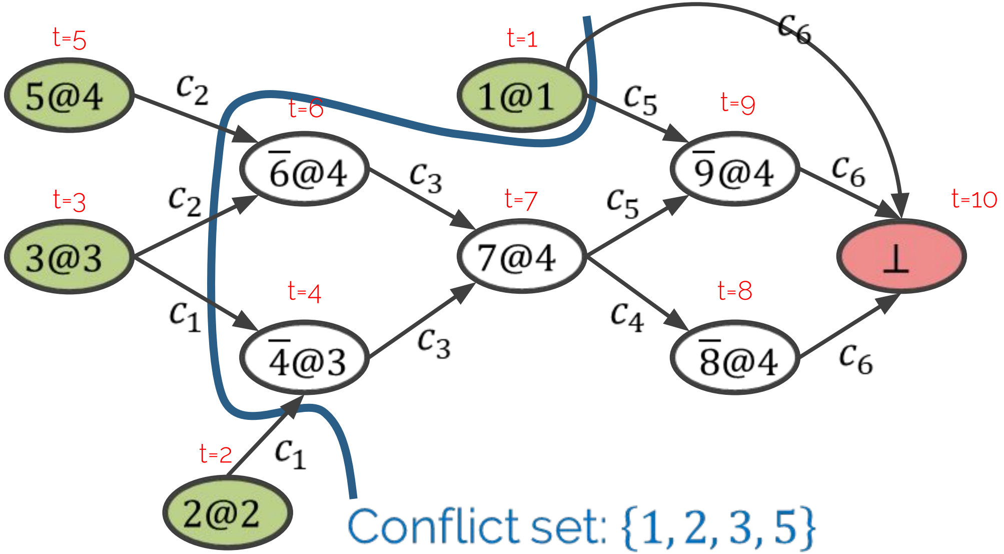
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee 8$

c_5: $\overline{1} \vee \overline{7} \vee 9$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$



Conflict set: $\{1, 2, 3, 5\}$

Asserting level = 3 (from 3@3)

Decision Levels

When we **backjump** from level j to level i, all variables that got assigned *after* level i until and including level j get unassigned

Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

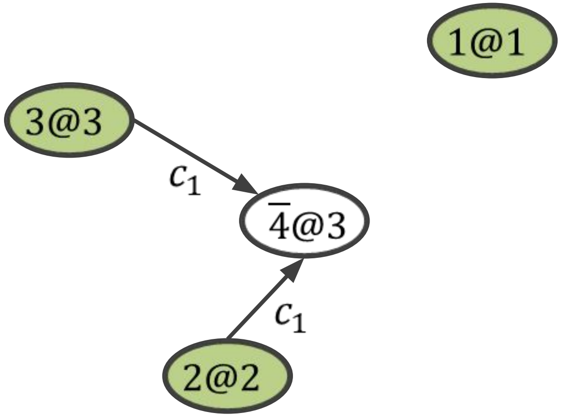
c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$

Backjump!



Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

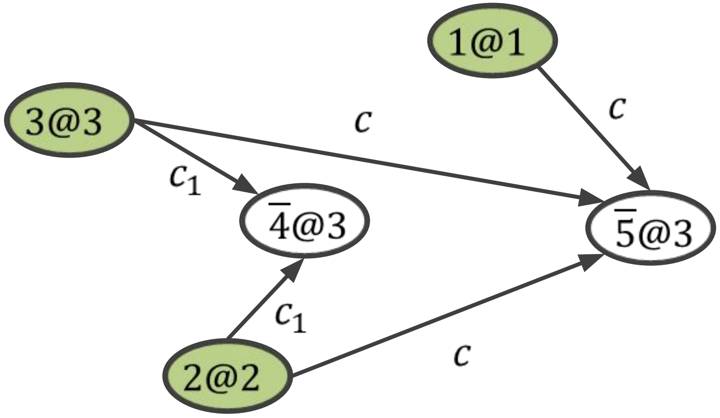
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$ **Unit!**



Asserting Level Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

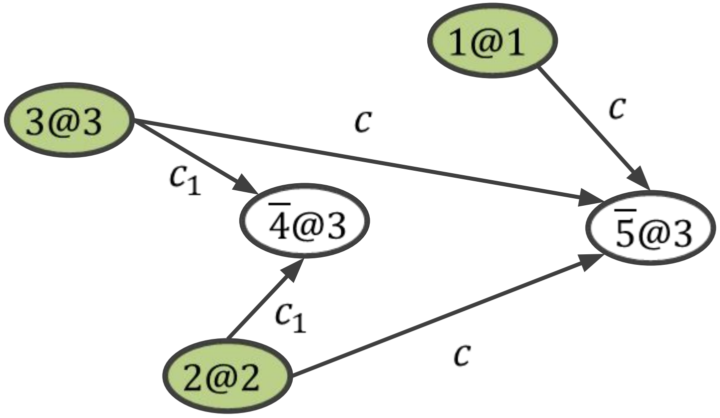
c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{2} \vee \overline{3} \vee \overline{5}$ **Unit!**

Wait a second... same outcome as backtracking with DPLL.



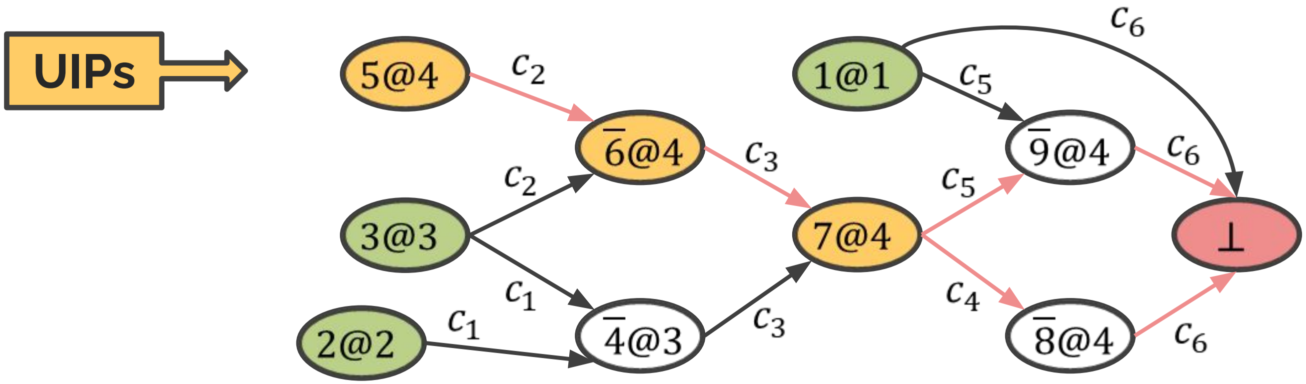
Q1: What *type* of clause do we want to add to the formula?

A1: Asserting Clauses, because they will allow us to Unit Propagate immediately

Q2: How do we find the *best* asserting clause?

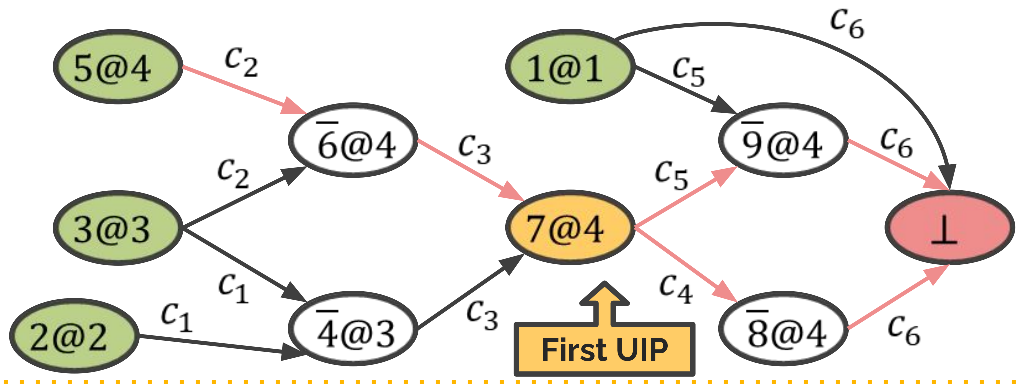
Unique Implication Points

- **Unique implication point (UIP):** a node in the implication graph that all paths from the most recent decision variable to the conflict must pass through
- **Intuition:** at the decision level of the conflict, the UIP is a literal that, by itself, implies a contradiction



The 1-UIP Scheme

- The **"first" UIP** is the closest UIP to the conflict node
  - i.e., *the UIP with the highest timestamp*
- When we reach a conflict, cut after the first UIP
  - One side has vertices with t <= t*, other with t > t*



1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

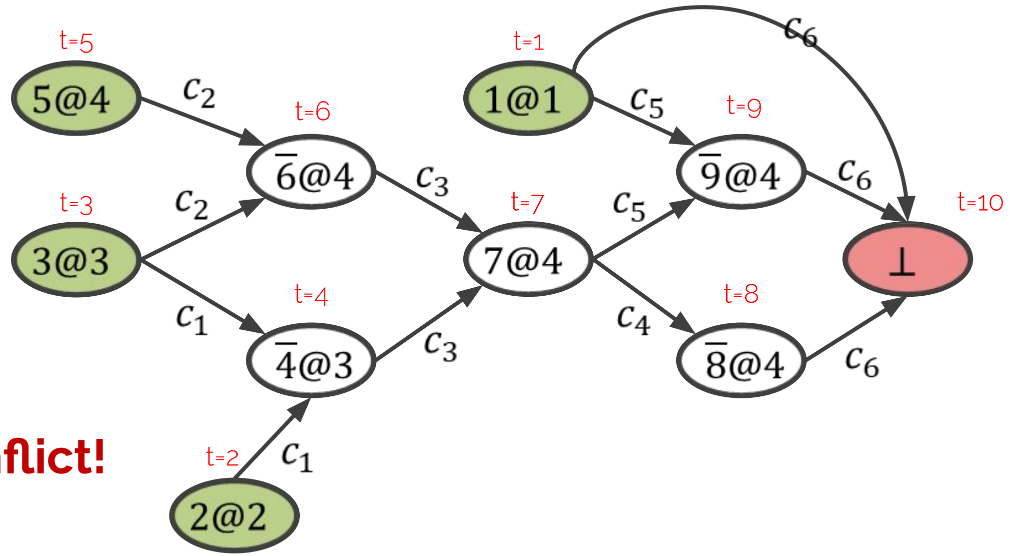
c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee 8$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$ **Conflict!**



1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

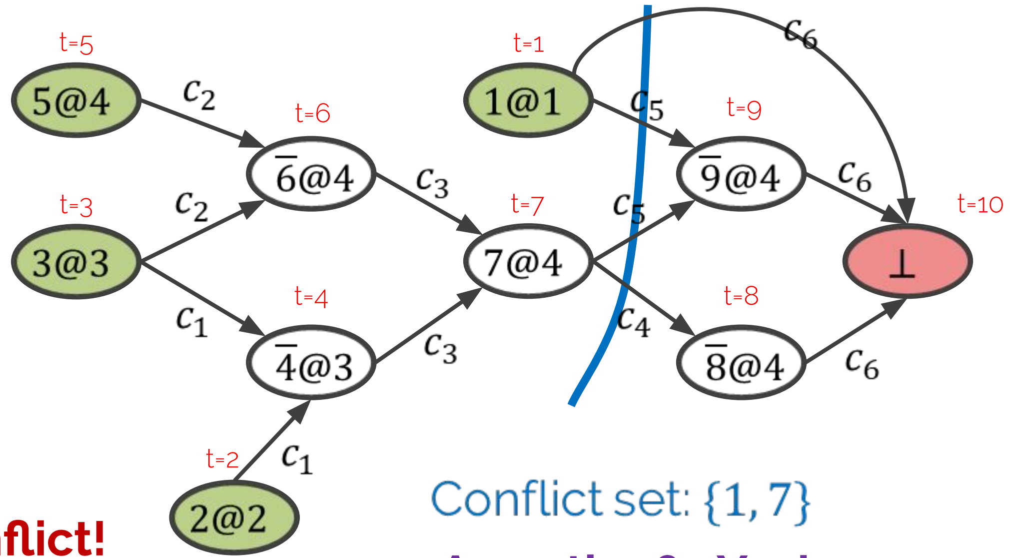
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$ **Conflict!**

c: $\overline{1} \vee \overline{7}$



Conflict set: $\{1, 7\}$

Asserting? **Yes!**

Level? **1**

1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{7}$

Backjump!

1-UIP Backjump

c_1: $\overline{2} \vee \overline{3} \vee \overline{4}$

c_2: $\overline{3} \vee \overline{5} \vee \overline{6}$

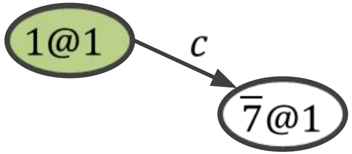
c_3: $4 \vee 6 \vee 7$

c_4: $\overline{7} \vee \overline{8}$

c_5: $\overline{1} \vee \overline{7} \vee \overline{9}$

c_6: $\overline{1} \vee 8 \vee 9$

c: $\overline{1} \vee \overline{7}$ **Unit!**



Restarts

- Problem: if we make bad early guesses, can get stuck in fruitless areas of search tree
- Solution: periodically **restart** the search – throw away the current partial assignment
  - Modern solvers favor aggressive restart policy
    - MiniSAT, PicoSAT: every ~100 conflicts
- **Key idea:** CDCL is deterministic, so why won't we end up back where we were?
  - Learned clauses remain in formula after restart

Incremental SAT Solving

- CDCL solvers give us a new method in our toolkit!

add_clause(C): add clause C to the formula

- New clauses can only rule out previously satisfying assignments
- Can re-solve CNF with new clauses added
- **Key:** keep learned clauses generated during last call to solve()
- Simple use case: generating all satisfying assignments

Introducing: PennSAT

- **HW2: PennSAT (due in a couple weeks)**
- Features:
  - DPLL-based
  - Iterative
  - Maintains propagation queue
  - No Two-Watched Literals
  - Static most-frequent decision heuristic
- This assignment is tricky – **start early!**
  - Requires solid understanding

Iterative DPLL

- A **decision** refers to any time our algorithm *arbitrarily* assigns a variable (without being forced to do so)
  - Selecting a literal and assigning it True is a decision
  - Unit propagation & reassigning selected literal after backtracking are not decisions
- All assignments implied by the i^{th} decision are said to be on the i^{th} **decision level**
  - Can assignments ever be on the zeroth decision level?

Iterative DPLL

- Maintain an **assignment stack** with the assignments from each decision level
  - Whenever we make a new decision, copy the current assignment onto the top of the stack
- To backtrack: pop the current assignment off the stack, restoring the previous one
- Keep a **propagation queue** of literals that are set to False
  - Take literals from the queue and check if their watching clauses are empty/unit

Assignment Stack

T	T	F	T	T
T	T	F		
T				
1	2	3	4	5

Set $2 = T$. Propagate $3 = F$.

Set $1 = T$

Assignment Stack

Pop!

T	T	F	T	T

Backtrack!

T	T	F		
T				
1	2	3	4	5

Set $2 = T$. Propagate $3 = F$.

Set $1 = T$

Iterative DPLL (Pseudocode)

```
dpll(φ):
    if unit_propagate() = CONFLICT: return UNSAT
    while not all variables have been set:
        let x = pick_variable()
        create new decision level
        set x = T
        while unit_propagate() = CONFLICT:
            if decision_level = 0: return UNSAT
            backtrack()
            set x = F
   return SAT
```

Testing a SAT Solver

- SAT solvers have tons of complicated logic... how to check for soundness bugs?
  - Hard and tedious to figure out all cases to unit test
- **Random testing:** generate random CNF formulas to test against reference solver
- If reference solver is not available, can at least check that satisfying assignments are valid

Debugging a SAT Solver

- Once we've found a bug, how do we find the mistake in the code?
- **Print debugging:** stick a bunch of print statements in relevant places and look at the console
- Easy, but not as effective for complex systems
    - Easy to forget to print something, or print in wrong place

Debugging a SAT Solver

Debugging in VS Code

- **Debugger:** allows us to stop program mid-execution, run code line-by-line, inspect values of local variables

```
    def __init__(self, n: int, cnf: CNF, activity_heu
        # The number of variables
        self.n = n
        # The CNF as a list of clauses
        self.cnf = preprocess(cnf)
        # A stack of partial truth assignments: lists
        self.assignment_stack = [[None] * (n+1)]
```

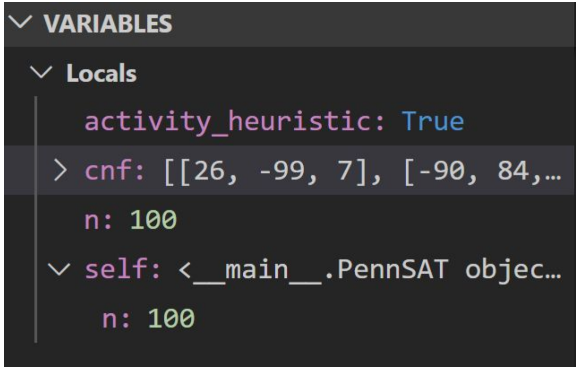
- **Breakpoint:** STOP at this line of code

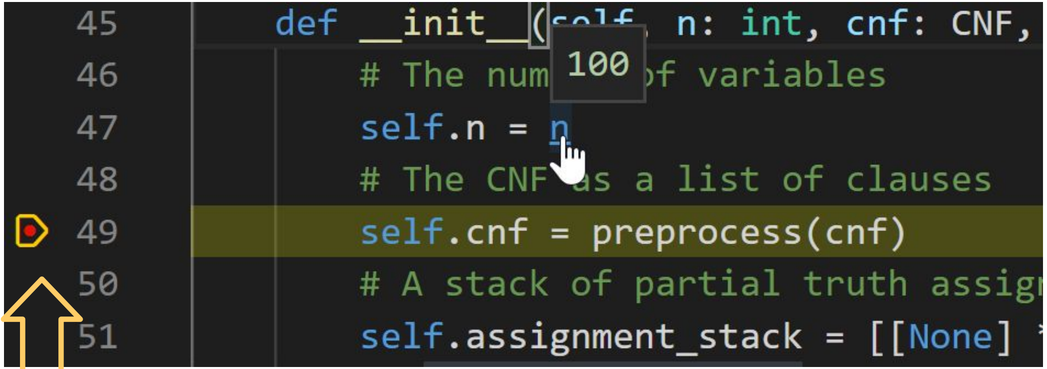
Debugging in VS Code

- After breakpoints set: Run > Start Debugging (F5)

View or *modify* current variables & values

Hover over variables to inspect values





Stopped right before line 49!

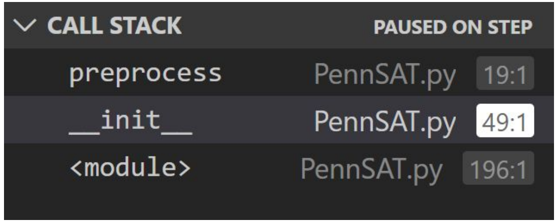
Debugging in VS Code

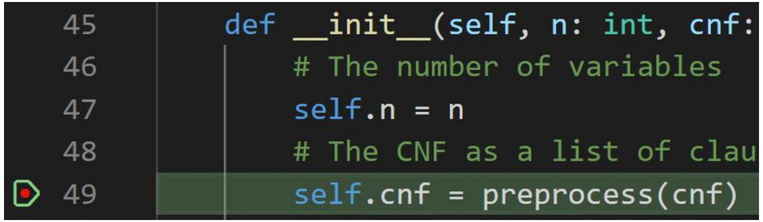
- Control flow: 

-  Continue (F5): run until next breakpoint hit
-  Step Over (F10): run just one more line of code
-  Restart (Ctrl+Shift+F5): start over from beginning
-  Stop (Shift+F5): quit the debugger

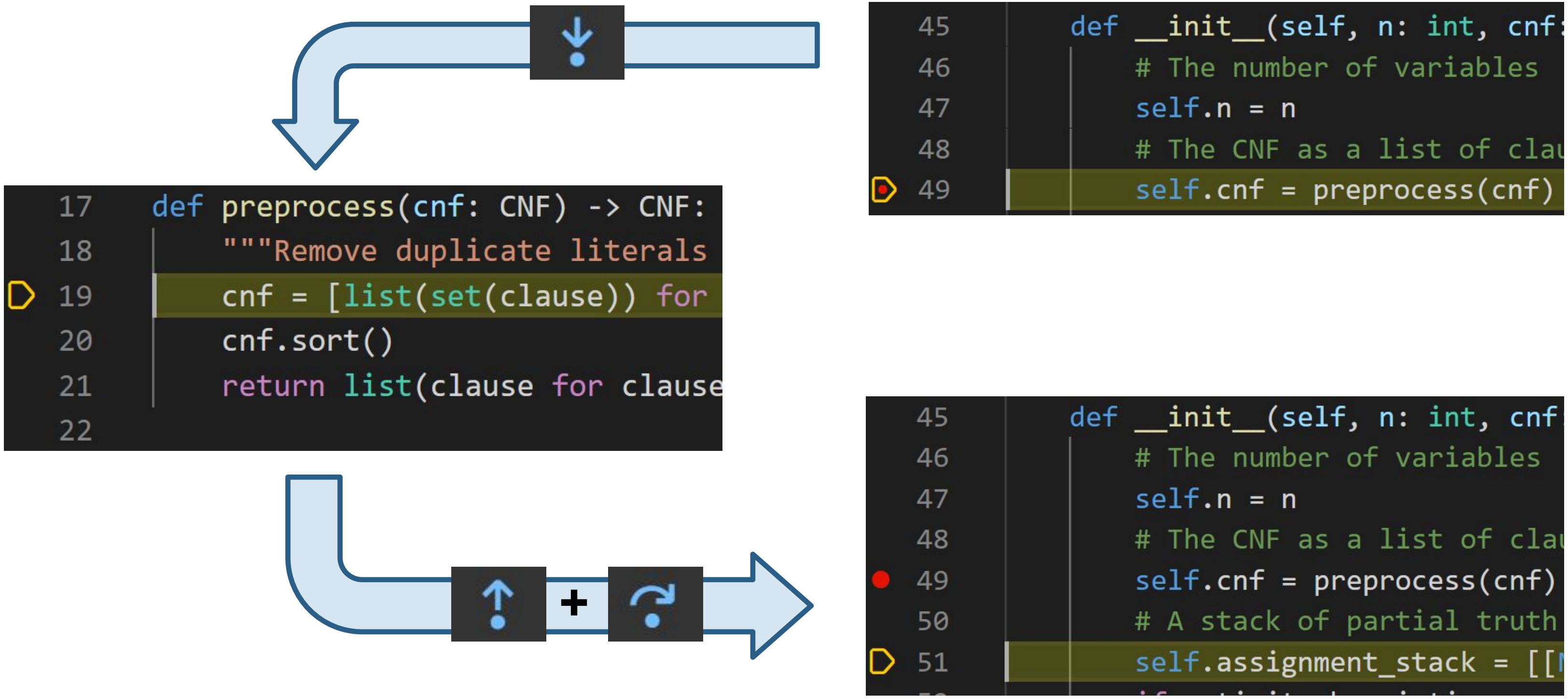
Debugging in VS Code

- Step Into (F11): enter code of first function called on the current line and resume debugging there
- Step Out (Shift+F11): run until the current function returns; resume debugging from parent function
- Can click to view different levels of the call stack
  - Useful for inspecting values of local vars in different scopes





Debugging in VS Code



Stay Diligent

"Things may come to those who wait, but only the things left by those who hustle" ~Abraham Lincoln

References

A. Biere, *Handbook of satisfiability*. Amsterdam: IOS Press, 2009.

E. Torlak, "A Modern SAT Solver," *CSE507: Computer-Aided Reasoning for Software Engineering*. [Online]. Available: https://courses.cs.washington.edu/courses/cse507/.

V. Ganesh, *On the Unreasonable Effectiveness of Boolean SAT Solvers*. Saarbrücken, Germany: Max Planck Institute, 2017. Available: https://docs.google.com/a/gsd.uwaterloo.ca/viewer?a=v&pid=sites&srcid=Z3NkLnV3YXRlcmxvby5jYXxtYXBsZXNhdHxneDoz YzQ3NDJjYjk4YWE4YTA0

J. Liang, V. Ganesh, E. Zulkoski, A. Zaman, and K. Czarnecki. "Understanding VSIDS Branching Heuristics in Conflict-Driven Clause-Learning SAT Solvers." *Hardware and Software: Verification and Testing Lecture Notes in Computer Science*, 2015, 225–41. https://doi.org/10.1007/978-3-319-26287-1_14.