Static Analysis and Verification

SAV 2023/2024

Tomáš Vojnar, Jan Fiedor, Filip Konečný

vojnar@fit.vutbr.cz

Brno University of Technology
Faculty of Information Technology
Božetěchova 2, 612 66 Brno

SAT and SMT Solving

Introduction

❖ The SAT problem is a decision problem which asks whether a given propositional formula is satisfiable.

❖ To solve instances of SAT, the so called SAT-solvers are used, which implement a decision procedure for SAT.

❖ The SMT problem extends the SAT problem to satisfiability of first-order formulae with equality and atoms from various first-order theories:

- linear integer arithmetics, real arithmetics, uninterpreted functions, arrays, bit-vectors, ..., and their mixtures allowing variables shared among theories,
- e.g., $(p) \wedge (\neg q) \wedge (a = f(b - c)) \wedge (a - c \geq 7)$,

Typically, decidable theories are supported. Sometimes, quantifiers are allowed.

❖ Since satisfiability and validity are dual notions, we may be able to check validity of formulae by using a decision procedure for satisfiability (using negation of given formulae).

❖ SAT and SMT solving has found many applications in verification (e.g., within predicate abstraction or invariant checking), test generation, hardware synthesis, error trace minimisation, artificial intelligence, ...

SAT Solving

Propositional Logic

❖ The SAT problem, which asks whether a given propositional formula is satisfiable, is the first problem which has been proven to be NP-complete.

❖ Normally, we consider a propositional formula to be given in the conjunctive normal form (CNF), i.e., as a conjunction of clauses where a clause is a disjunction of literals, and a literal is a (possibly negated) propositional symbol.

❖ Stated formally, let P be a finite set of propositional symbols.

- If $p \in P$, then p is an atom, and p and $\neg p$ are literals of P.
- A clause is a disjunction of literals $l_1 \vee \ldots \vee l_n$.
- A CNF formula is a conjunction of one or more clauses $C_1 \wedge \ldots \wedge C_n$.

❖ An example: Let $P = \{p, q, r\}$. Then,

- p, q, r are possible atoms, $\neg p$ and r are examples of literals,
- $p \vee \neg q \vee \neg r$ is an example of a clause, and
- $(p \vee \neg q \vee \neg r) \wedge (\neg p \vee r)$ is an example of a CNF formula.

Propositional Logic

- Assignments:
  - An assignment M is a set of literals such that $\{p, \neg p\} \subseteq M$ for no p.
  - A literal l can be either true, false, or undefined in M. Assuming that $\neg\neg l = l$:
    - l is $true$ in M iff $l \in M$,
    - l is $false$ in M iff $\neg l \in M$,
    - l is $undefined$ in M otherwise.
  - If l is either $true$ or $false$ in M, we say that l is defined.
  - If for any $p \in P$, either $p \in M$ or $\neg p \in M$, we say that M is total (in other words, no literal in P is $undefined$ in M). Otherwise, M is partial.

- An example: Consider the assignment $M = \{\neg p, q\}$. In M, p, q are defined (p is $false$ and q is $true$), whereas r is $undefined$.

Propositional Logic

❖ Semantics of propositional logic. Let C be a clause, F a CNF formula, and M an assignment.

- In a partial assignment M, C can be either true, false, or undefined:
  - C is true in M iff $l \in M$ for at least one literal l in C,
  - C is false in M iff $\neg l \in M$ for each literal l in C,
  - otherwise, C is undefined in M.
- F is true (satisfied) in M (written $M \models F$) iff all clauses in F are true in M. M is said to be a model of F. If F has no models, then it is unsatisfiable.
- F is false (falsified) in M (written $M \not\models F$) iff some clause in F is false in M.

❖ A formula F' is a logical consequence of a formula F ($F \models F'$) iff F' is true in every model of F. F and F' are logically equivalent iff $F \models F'$ and $F' \models F$.

❖ An example: Consider an assignment $M = \{\neg p, q\}$, clauses $C_1 = (\neg p \vee \neg q)$, $C_2 = (q)$, $C_3 = (p \vee \neg q)$, $C_4 = (p \vee \neg q \vee r)$, and CNF formulae $F = C_1 \wedge C_2$, $F' = C_3 \wedge C_4$.

- C_1 and C_2 are true in M, C_3 is false in M, and C_4 is undefined in M.
- F is satisfied in M (M is a model of F), whereas F' is falsified in M (M is not a model of F').

Duality of Satisfiability and Validity

❖ For propositional logic, satisfiability and validity are dual notions which means that one can be expressed in terms of the other.

❖ Let F be a propositional formula.

- F is satisfiable iff there exists an assignment M such that $M \models F$.
- F is valid iff for all assignments M, $M \models F$.

❖ Clearly, the connection between satisfiability and validity is the following:

- F is valid iff $\neg F$ is unsatisfiable.

❖ An example: Checking validity of the formula $F = p \vee \neg p \vee q$ is equivalent to checking unsatisfiability of the formula $\neg F = \neg p \wedge p \wedge \neg q$.

❖ *Note*: For some theories (other than propositional logic), the negation of a formula may be outside of the theory – then, the above duality is not applicable within the theory, but between the theory and its dual theory.

Abstract DPLL Systems

- Most SAT-solvers build on variants of the classical Davis-Putnam-Longemann-Loveland (DPLL) procedure, which we will describe in terms of an abstract DPLL system.

- An abstract DPLL system is a pair $(S, \Rightarrow)$ where S is a set of states of the system and $\Rightarrow \subseteq (S \times S)$ is its set of transitions.
  - A state is a pair $M \parallel F$ where M is an assignment and F is a CNF formula.
  - A special fail state is also introduced.

- A derivation over $(S, \Rightarrow)$ is any sequence of the form $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ for $S_0, S_1, S_2, \ldots \in S$.

- To capture (to some extent) the evolution of an assignment, assignments are turned into sequences of literals (satisfying the requirements stated previously). The intuition is that the literals more to the right have been added to the assignment more recently.

- An example: $p, \neg q \parallel (p \vee \neg q), (\neg q \vee r)$ is a state.
  - Note: We often use "," instead of "$\wedge$".

Abstract DPLL Systems

- The transition relation of an abstract DPLL system is defined by means of transition rules.
  - Intuitively, a transition rule consists of a condition and an effect on system states.
  - The condition describes situations when the transition rule may be applied and the effect describes the result of such an application.

Abstract DPLL Systems

❖ The transition relation of an abstract DPLL system is defined by means of transition rules.

- Intuitively, a transition rule consists of a condition and an effect on system states.
- The condition describes situations when the transition rule may be applied and the effect describes the result of such an application.

❖ Given a CNF formula F whose satisfiability is to be checked, a derivation in an abstract DPLL system starts in the state $\emptyset \parallel F$.

Abstract DPLL Systems

❖ The transition relation of an abstract DPLL system is defined by means of transition rules.

- Intuitively, a transition rule consists of a condition and an effect on system states.
- The condition describes situations when the transition rule may be applied and the effect describes the result of such an application.

❖ Given a CNF formula F whose satisfiability is to be checked, a derivation in an abstract DPLL system starts in the state $\emptyset \parallel F$.

❖ The derivation progresses by applying the transition rules, possibly resulting

- in an extension of the assignment M by adding a literal for a so far undefined propositional symbol, or
- when M falsifies F and therefore another assignment has to be tried, in reverting M to some previous value and in extending it in a different way than before.
- Some other effects are also possible (learning, restart, ...).

Abstract DPLL Systems

❖ The transition relation of an abstract DPLL system is defined by means of transition rules.

- Intuitively, a transition rule consists of a condition and an effect on system states.
- The condition describes situations when the transition rule may be applied and the effect describes the result of such an application.

❖ Given a CNF formula F whose satisfiability is to be checked, a derivation in an abstract DPLL system starts in the state $\emptyset \parallel F$.

❖ The derivation progresses by applying the transition rules, possibly resulting

- in an extension of the assignment M by adding a literal for a so far undefined propositional symbol, or
- when M falsifies F and therefore another assignment has to be tried, in reverting M to some previous value and in extending it in a different way than before.
- Some other effects are also possible (learning, restart, ...).

❖ Some transition rules add a literal which should clearly be added, whereas others have a speculative nature—they come with a choice/decision of which literal over a given variable to add to the assignment (a literal added in such a way is annotated, e.g., $\neg p^d$).

Abstract DPLL Systems

❖ The transition relation of an abstract DPLL system is defined by means of transition rules.

- Intuitively, a transition rule consists of a condition and an effect on system states.
- The condition describes situations when the transition rule may be applied and the effect describes the result of such an application.

❖ Given a CNF formula F whose satisfiability is to be checked, a derivation in an abstract DPLL system starts in the state $\emptyset \parallel F$.

❖ The derivation progresses by applying the transition rules, possibly resulting

- in an extension of the assignment M by adding a literal for a so far undefined propositional symbol, or
- when M falsifies F and therefore another assignment has to be tried, in reverting M to some previous value and in extending it in a different way than before.
- Some other effects are also possible (learning, restart, ...).

❖ Some transition rules add a literal which should clearly be added, whereas others have a speculative nature—they come with a choice/decision of which literal over a given variable to add to the assignment (a literal added in such a way is annotated, e.g., $\neg p^d$).

❖ If there is no transition from a state s, then s is final.

Abstract DPLL Systems

❖ In what follows, we first describe transition rules of the most basic, classical DPLL procedure, yielding the transition relation denoted $\Rightarrow_{Cl}$.

❖ Later, we will extend it by further rules which make the search in the state space progress faster and are therefore used in modern DPLL procedures, also called Conflict-Driven Clause Learning (CDCL) systems, yielding the transition relations denoted $\Rightarrow_M$, $\Rightarrow_L$, and $\Rightarrow_R$.

Classical DPLL Rules: $PureLiteral$

- $PureLiteral$:

$$M \parallel F \Rightarrow M\ l \parallel F \quad \text{if} \quad \begin{cases} l \text{ occurs in some clause of } F \\ \neg l \text{ occurs in no clause of } F \\ l \text{ is undefined in } M \end{cases}$$

- An example:

$$\begin{array}{rcll} \emptyset & \parallel & 1 \vee 2, 2 \vee \overline{4}, 1 \vee \overline{3} \vee \overline{4}, \overline{1} \vee 3 & \Rightarrow_{Cl} \quad [PureLiteral] \\ 2 & \parallel & 1 \vee 2, 2 \vee \overline{4}, 1 \vee \overline{3} \vee \overline{4}, \overline{1} \vee 3 & \Rightarrow_{Cl} \quad [PureLiteral] \\ 2\ \overline{4} & \parallel & 1 \vee 2, 2 \vee \overline{4}, 1 \vee \overline{3} \vee \overline{4}, \overline{1} \vee 3 & \end{array}$$

Classical DPLL Rules: $Decide$

- $Decide$:

$$M \parallel F \Rightarrow M\ l^d \parallel F \quad \text{if} \quad \begin{cases} l \text{ or } \neg l \text{ occurs in a clause of } F \\ l \text{ is undefined in } M \end{cases}$$

- An example:

$$\begin{array}{lcll} \emptyset & \parallel & 1 \vee \overline{2}, 2 \vee \overline{4}, 1 \vee \overline{3} \vee 4, \overline{1} \vee 3 & \Rightarrow_{Cl} \quad [Decide] \\ 1^d & \parallel & 1 \vee \overline{2}, 2 \vee \overline{4}, 1 \vee \overline{3} \vee 4, \overline{1} \vee 3 & \end{array}$$

- Let $M \parallel F$ be a state of a system and let M be written as a sequence of the form $M_0 l_1^d M_1 l_2^d M_2 \ldots l_k^d M_k$ where l_i^d are all the decision literals in M. We say that the state $M \parallel F$ is at decision level k and that all literals of each $l_i^d M_i$ belong to the decision level i.

Classical DPLL Procedures: $UnitPropagate$

- $UnitPropagate$:

$$M \parallel F,\ C \vee l \Rightarrow M\ l \parallel F,\ C \vee l \quad \text{if} \quad \begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$$

- An example:

$$\begin{array}{rcll} \emptyset & \parallel & 1 \vee \overline{2}, 2 \vee \overline{4}, 1 \vee \overline{3} \vee 4, \overline{1} \vee 3 & \Rightarrow_{Cl} \quad [Decide] \\ 1^d & \parallel & 1 \vee \overline{2}, 2 \vee \overline{4}, 1 \vee \overline{3} \vee 4, \overline{1} \vee 3 & \Rightarrow_{Cl} \quad [UnitPropagate] \\ 1^d\ 3 & \parallel & 1 \vee \overline{2}, 2 \vee \overline{4}, 1 \vee \overline{3} \vee 4, \overline{1} \vee 3 & \end{array}$$

Classical DPLL Procedures: $BackTrack$

- $BackTrack$:

$$M\ l^d\ N \parallel F,\ C \Rightarrow M \neg l \parallel F,\ C \quad \text{if} \quad \begin{cases} M\ l^d\ N \models \neg C \\ N \text{ contains no decision literals} \end{cases}$$

- We call the clause C conflicting.

- An example:

$$\begin{array}{rcllr} \emptyset & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [Decide] \\ 1^d & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [UnitPropagate] \\ 1^d\ 2 & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [BackTrack] \\ \overline{1} & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & & \end{array}$$

Classical DPLL Procedures: $Fail$

- $Fail$:

$$M \parallel F,\ C \Rightarrow FailState \quad \text{if} \quad \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

- An example:

$$\begin{array}{rcllcl}
\emptyset & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [Decide] \\
1^d & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2 & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [BackTrack] \\
\overline{1} & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [UnitPropagate] \\
\overline{1}\ \overline{2} & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [Fail] \\
 & & FailState & &
\end{array}$$

Classical DPLL Procedures: $Fail$

❖ $Fail$:

$$M \parallel F,\ C \Rightarrow FailState \quad \text{if} \quad \begin{cases} M \models \neg C \\ M \text{ contains no decision literals} \end{cases}$$

❖ An example:

$$\begin{array}{rcllr}
\emptyset & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [Decide] \\
1^d & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2 & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [BackTrack] \\
\overline{1} & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [UnitPropagate] \\
\overline{1}\ \overline{2} & \parallel & 1 \vee \overline{2}, \overline{1} \vee 2, 1 \vee 2, \overline{1} \vee \overline{2} & \Rightarrow_{Cl} & [Fail] \\
 & & FailState & &
\end{array}$$

❖ On the other hand, if no further rule is applicable and the derivation has not got to $FailState$, the given formula is satisfiable.

- In particular, the assignment in the final state is an example of a satisfying assignment for the formula.

Classical DPLL Procedures: Applying the Rules

- The rules are not applied in a completely random order.
- The priorities for applying the rules are as follows:
  1. If $Fail$ or $BackTrack$ are applicable, they are applied.
  2. Otherwise, $UnitPropagate$ and $PureLiteral$ are applied if possible.
  3. Only if no other rule can be applied, $Decide$ is used.
- The motivation is clear: reducing the amount of guessing if possible.
- The use of $Decide$ is then subject to various heuristics:
  - Choose the unassigned variable and value that satisfies the maximum number of unsatisfied clauses (expensive: requires going through all clauses for each decision).
  - Choose the unassigned variable and value that appears in the biggest number of clauses.
  - MOM, VSIDS, ... – beyond the scope of this lecture.

Modern DPLL Systems

- Modern DPLL systems do not use $PureLiteral$ in the derivation process.
  - It is used as a preprocessing step for efficiency reasons.

- Further, a more sophisticated $backtracking$ mechanism is used.
  - $BackTrack$ tries to solve conflicts by reverting the value of the last decision literal regardless of whether it is relevant for the current conflict or not.
  - This can lead to a series of useless back- and forth-computations.
  - Instead, one would like to move faster to the relevant decision levels and resolve the current conflict earlier. This is achieved by using the $BackJump$ rule.

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$1^d\ 2\ 3^d\ 4 \quad \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [Decide]$$

Classical DPLL Systems: Useless Backtracking

- Note: We assume that the "..." in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$1^d\ 2\ 3^d\ 4 \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [Decide]$$

$$1^d\ 2\ 3^d\ 4\ 5^d \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [UnitPropagate]$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$
\begin{array}{lll}
1^d\ 2\ 3^d\ 4 & \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [Decide] \\
1^d\ 2\ 3^d\ 4\ 5^d & \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| \quad \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [BackTrack]
\end{array}
$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$
\begin{array}{lcll}
1^d\ 2\ 3^d\ 4 & \| & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [Decide] \\
1^d\ 2\ 3^d\ 4\ 5^d & \| & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [BackTrack] \\
1^d\ 2\ 3^d\ 4\ \overline{5} & \| & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_{Cl} \quad [UnitPropagate]
\end{array}
$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$
\begin{array}{lcll}
1^d\ 2\ 3^d\ 4 & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots & \Rightarrow_{Cl} \quad [Decide] \\
1^d\ 2\ 3^d\ 4\ 5^d & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots & \Rightarrow_{Cl} \quad [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots & \Rightarrow_{Cl} \quad [BackTrack] \\
1^d\ 2\ 3^d\ 4\ \overline{5} & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots & \Rightarrow_{Cl} \quad [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ \overline{5}\ 6 & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots & \Rightarrow_{Cl} \quad [BackTrack]
\end{array}
$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$1^d\ 2\ 3^d\ 4 \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [Decide]$$

$$1^d\ 2\ 3^d\ 4\ 5^d \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [UnitPropagate]$$

$$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [BackTrack]$$

$$1^d\ 2\ 3^d\ 4\ \overline{5} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [UnitPropagate]$$

$$1^d\ 2\ 3^d\ 4\ \overline{5}\ 6 \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [BackTrack]$$

$$1^d\ 2\ \overline{3} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad \Rightarrow_{Cl} \quad [Decide]$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$
\begin{array}{lcllr}
1^d\ 2\ 3^d\ 4 & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide] \\
1^d\ 2\ 3^d\ 4\ 5^d & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ 3^d\ 4\ \overline{5} & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ \overline{5}\ 6 & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ \overline{3} & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide] \\
1^d\ 2\ \overline{3}\ 5^d & \| & \overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate]
\end{array}
$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$1^d\ 2\ 3^d\ 4 \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [Decide]$$

$$1^d\ 2\ 3^d\ 4\ 5^d \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [UnitPropagate]$$

$$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [BackTrack]$$

$$1^d\ 2\ 3^d\ 4\ \overline{5} \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [UnitPropagate]$$

$$1^d\ 2\ 3^d\ 4\ \overline{5}\ 6 \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [BackTrack]$$

$$1^d\ 2\ \overline{3} \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [Decide]$$

$$1^d\ 2\ \overline{3}\ 5^d \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [UnitPropagate]$$

$$1^d\ 2\ \overline{3}\ 5^d\ \overline{6} \quad \| \quad \overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6}\lor\overline{1},\ 5\lor 6\lor\overline{1},\ \overline{5}\lor 6\lor\overline{1},\ 5\lor\overline{6}\lor\overline{1},\ \ldots \quad \Rightarrow_{Cl} \quad [BackTrack]$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "…" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$1^d\ 2\ 3^d\ 4$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[Decide]$
$1^d\ 2\ 3^d\ 4\ 5^d$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6}$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ 3^d\ 4\ \overline{5}$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ 3^d\ 4\ \overline{5}\ 6$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ \overline{3}$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[Decide]$
$1^d\ 2\ \overline{3}\ 5^d$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ \overline{3}\ 5^d\ \overline{6}$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ \overline{3}\ \overline{5}$	$\Vert$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6}\vee\overline{1},\ 5\vee 6\vee\overline{1},\ \overline{5}\vee 6\vee\overline{1},\ 5\vee\overline{6}\vee\overline{1},\ \dots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$
\begin{array}{llllll}
1^d\ 2\ 3^d\ 4 & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide] \\
1^d\ 2\ 3^d\ 4\ 5^d & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ 3^d\ 4\ \overline{5} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ \overline{5}\ 6 & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ \overline{3} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide] \\
1^d\ 2\ \overline{3}\ 5^d & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ \overline{3}\ 5^d\ \overline{6} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ \overline{3}\ \overline{5} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ \overline{3}\ \overline{5}\ 6 & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack]
\end{array}
$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$$
\begin{array}{lllll}
1^d\ 2\ 3^d\ 4 & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide] \\
1^d\ 2\ 3^d\ 4\ 5^d & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ 3^d\ 4\ \overline{5} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ 3^d\ 4\ \overline{5}\ 6 & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ \overline{3} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide] \\
1^d\ 2\ \overline{3}\ 5^d & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ \overline{3}\ 5^d\ \overline{6} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
1^d\ 2\ \overline{3}\ \overline{5} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [UnitPropagate] \\
1^d\ 2\ \overline{3}\ \overline{5}\ 6 & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [BackTrack] \\
\overline{1} & \| & \overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots & \Rightarrow_{Cl} & [Decide]
\end{array}
$$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$1^d\ 2\ 3^d\ 4$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[Decide]$
$1^d\ 2\ 3^d\ 4\ 5^d$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6}$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ 3^d\ 4\ \overline{5}$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ 3^d\ 4\ \overline{5}\ 6$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ \overline{3}$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[Decide]$
$1^d\ 2\ \overline{3}\ 5^d$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ \overline{3}\ 5^d\ \overline{6}$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ \overline{3}\ \overline{5}$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ \overline{3}\ \overline{5}\ 6$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$\overline{1}$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[Decide]$
$\overline{1}\ 3^d$	$\Vert$	$\overline{1}\lor 2, \overline{3}\lor 4, \overline{5}\lor\overline{6}\lor\overline{1}, 5\lor 6\lor\overline{1}, \overline{5}\lor 6\lor\overline{1}, 5\lor\overline{6}\lor\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$

Classical DPLL Systems: Useless Backtracking

❖ Note: We assume that the "$\ldots$" in the formula below hide a clause containing 1 and a clause containing 3; hence, $PureLiteral$ is not applicable here.

$1^d\ 2\ 3^d\ 4$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[Decide]$
$1^d\ 2\ 3^d\ 4\ 5^d$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6}$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ 3^d\ 4\ \overline{5}$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ 3^d\ 4\ \overline{5}\ 6$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ \overline{3}$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[Decide]$
$1^d\ 2\ \overline{3}\ 5^d$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ \overline{3}\ 5^d\ \overline{6}$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$1^d\ 2\ \overline{3}\ \overline{5}$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$1^d\ 2\ \overline{3}\ \overline{5}\ 6$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[BackTrack]$
$\overline{1}$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[Decide]$
$\overline{1}\ 3^d$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$[UnitPropagate]$
$\overline{1}\ 3^d\ 4$	$\|$	$\overline{1}\vee 2, \overline{3}\vee 4, \overline{5}\vee\overline{6}\vee\overline{1}, 5\vee 6\vee\overline{1}, \overline{5}\vee 6\vee\overline{1}, 5\vee\overline{6}\vee\overline{1}, \ldots$	$\Rightarrow_{Cl}$	$\ldots$

Modern DPLL Systems: $BackJump$

- $BackJump$:

$$M\ l^d\ N \parallel F, C \Rightarrow M\ l' \parallel F, C \quad \text{if} \quad \begin{cases} M\ l^d\ N \models \neg C, \text{ and there is} \\ \text{some clause } C' \vee l' \text{ such that:} \\ \quad F,\ C \models C' \vee l' \text{ and } M \models \neg C', \\ \quad l' \text{ is undefined in } M, \text{ and} \\ \quad l' \text{ or } \neg l' \text{ occurs in } F, C \end{cases}$$

- In the definition of the $BackJump$ rule, the clause C is called conflicting. The clause $C' \vee l'$ is called a backjump clause. The literal l^d marks the target of the backjump.
  - The backjump clause must be entailed by F, C so that the procedure does not start solving a stronger formula than the given one (strengthened by assuming $C' \vee l'$ must hold).
  - The fact that $M \models \neg C'$ implies that l' must hold.
  - l' must be undefined in M so that it can be made defined.
  - The last requirement requires l' to be related to the given SAT instance.

- It can be shown that if there is a conflicting clause, one can always either apply $Fail$ or $BackJump$.

Modern DPLL Systems: $BackJump$

- $BackJump$:

$$M\ l^d\ N \parallel F, C \Rightarrow M\ l' \parallel F, C \quad \text{if} \quad \begin{cases} M\ l^d\ N \models \neg C, \text{ and there is} \\ \text{some clause } C' \lor l' \text{ such that:} \\ \quad F,\ C \models C' \lor l' \text{ and } M \models \neg C', \\ \quad l' \text{ is undefined in } M, \text{ and} \\ \quad l' \text{ or } \neg l' \text{ occurs in } F \text{ or in } M\ l^d\ N \end{cases}$$

- An example:

$$\begin{array}{rcll} 1^d\ 2\ 3^d\ 4 & \parallel & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_M \quad [Decide] \\ 1^d\ 2\ 3^d\ 4\ 5^d & \parallel & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_M \quad [UnitPropagate] \\ 1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \parallel & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \Rightarrow_M \quad [BackJump] \\ 1^d\ 2\ \overline{5} & \parallel & \overline{1} \lor 2, \overline{3} \lor 4, \overline{5} \lor \overline{6} \lor \overline{1}, 5 \lor 6 \lor \overline{1}, \overline{5} \lor 6 \lor \overline{1}, 5 \lor \overline{6} \lor \overline{1}, \ldots & \quad [\overline{1} \lor \overline{5}] \end{array}$$

Here:

- $\overline{1} \lor \overline{5}$ is the backjump clause $C' \lor l'$ (to see that $F, C \models C' \lor l'$, use resolution on $\overline{5} \lor \overline{6} \lor \overline{1}$ and $\overline{5} \lor 6 \lor \overline{1}$).
- $\overline{5}$ serves as l'.
- 3^d as l^d.

$BackJump$ **vs.** $BackTrack$

- To summarise the differences of $BackJump$ and $BackTrack$:
  - $BackTrack$ undoes the last decision l^d, going back to the previous decision level and adding $\neg l$ to it.
  - $BackJump$ may jump over decision levels that are irrelevant to the conflict.

$$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \Rightarrow_{Cl} \quad [BackJump]$$
$$1^d\ 2\ \overline{5} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad [\overline{1} \vee \overline{5}]$$

  - $BackJump$ jumps over the decision 3^d and its consequence 4, which are totally unrelated with the reasons for the falsity of the conflicting clause $\overline{5} \vee 6 \vee \overline{1}$.

$BackJump$ vs. $BackTrack$

❖ To summarise the differences of $BackJump$ and $BackTrack$:

- $BackTrack$ undoes the last decision l^d, going back to the previous decision level and adding $\neg l$ to it.
- $BackJump$ may jump over decision levels that are irrelevant to the conflict.

$$1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \Rightarrow_{Cl} \quad [BackJump]$$
$$1^d\ 2\ \overline{5} \quad \| \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \quad [\overline{1} \vee \overline{5}]$$

  – $BackJump$ jumps over the decision 3^d and its consequence 4, which are totally unrelated with the reasons for the falsity of the conflicting clause $\overline{5} \vee 6 \vee \overline{1}$.

- $BackJump$ may analyse the reasons that produced the conflicting clause (see further).

$BackJump$ vs. $BackTrack$

- To summarise the differences of $BackJump$ and $BackTrack$:
  - $BackTrack$ undoes the last decision l^d, going back to the previous decision level and adding $\neg l$ to it.
  - $BackJump$ may jump over decision levels that are irrelevant to the conflict.

$$\begin{array}{rcll} 1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots \Rightarrow_{Cl} & [BackJump] \\ 1^d\ 2\ \overline{5} & \| & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}, \ldots & [\overline{1} \vee \overline{5}] \end{array}$$

    - $BackJump$ jumps over the decision 3^d and its consequence 4, which are totally unrelated with the reasons for the falsity of the conflicting clause $\overline{5} \vee 6 \vee \overline{1}$.
  - $BackJump$ may analyse the reasons that produced the conflicting clause (see further).
- Conflict-driven learning:
  - To prevent future similar conflicts, backjump clauses (also called lemmas) may be added to the formula being handled.

Finding $BackJump$ Clauses

❖ A typical DPLL implementation will save the sequence of propagated literals and remember for each one of them the clause that caused its propagation. This information can be used for constructing the so-called conflict graph:

Finding $BackJump$ Clauses

❖ A typical DPLL implementation will save the sequence of propagated literals and remember for each one of them the clause that caused its propagation. This information can be used for constructing the so-called conflict graph:

- Start with nodes corresponding to negations of the literals in the conflicting clause.
- For each generated node n corresponding to some literal l'' from Ml^dN, add as predecessors nodes corresponding to literals in Ml^dN which allowed the propagation rule to be applied and obtain l'' in Ml^dN.

Finding $BackJump$ Clauses

❖ A typical DPLL implementation will save the sequence of propagated literals and remember for each one of them the clause that caused its propagation. This information can be used for constructing the so-called conflict graph:

- Start with nodes corresponding to negations of the literals in the conflicting clause.
- For each generated node n corresponding to some literal l'' from Ml^dN, add as predecessors nodes corresponding to literals in Ml^dN which allowed the propagation rule to be applied and obtain l'' in Ml^dN.
- To find a backjump clause, it suffices to cut the graph into two parts:
  - The first part must contain (at least) all the literals with no incoming arrows.
  - The second part must contain (at least) all the literals with no outgoing arrows.
  - Exactly one of the literals with cut outgoing edges belongs to the current (i.e., the last) decision level. The negation of this literal will act as the literal l' in the $BackJump$ rule.
- The backjump clause $C' \vee l'$ is then the disjunction of the negation of the literals with cut outgoing edges.

Finding $BackJump$ *Clauses*

- Some intuition behind the use of conflict graphs:
  - The fact that $F, C \models C' \vee l'$ is clear since F, C and the negation of $C' \vee l'$ cannot be satisfied (since $\neg(C' \vee l') \models \neg C$).
  - l' is chosen as a negation of a literal in $Ml^d N$, hence the last requirement of $BackJump$ is satisfied.
  - At the same time note that it is always possible to find $\neg l'$ which is the only one at the current decision level with cut outgoing edges.
    - One can (though not necessarily) choose the current decision point and let all other nodes in the first part of the conflict graph be from lower decision levels.

Finding $BackJump$ *Clauses*

- Some intuition behind the use of conflict graphs:
  - The fact that $F, C \models C' \vee l'$ is clear since F, C and the negation of $C' \vee l'$ cannot be satisfied (since $\neg(C' \vee l') \models \neg C$).
  - l' is chosen as a negation of a literal in $Ml^d N$, hence the last requirement of $BackJump$ is satisfied.
  - At the same time note that it is always possible to find $\neg l'$ which is the only one at the current decision level with cut outgoing edges.
    - One can (though not necessarily) choose the current decision point and let all other nodes in the first part of the conflict graph be from lower decision levels.

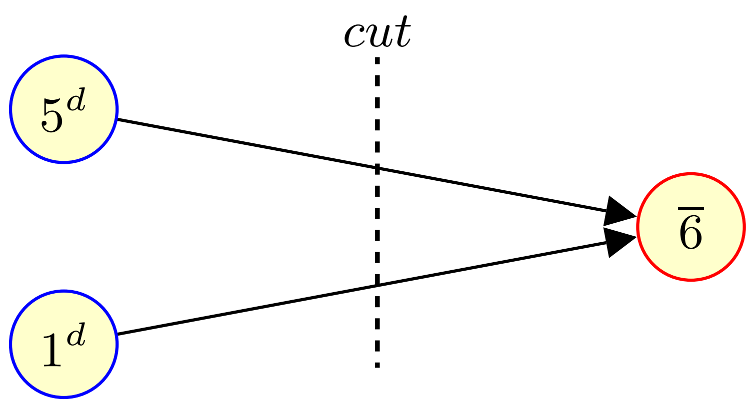
- It remains to choose the $BackJump$ target l^d:
  - Due to a single node from the current decision level has cut outgoing edges, $M \models \neg C'$ for M preceding the current decision.
  - One may proceed from the current decision point as far to the left in the assignment as possible till negations of the literals in C' appear in M, and hence $M \models \neg C'$. In this way, decisions irrelevant to the conflict are jumped over.
  - As $\neg l'$ is at the current decision level, it is clearly the case that l' is undefined in M.

Finding $BackJump$ *Clauses: Example 1*

- An example:

  $1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} \quad || \quad \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6} \vee \overline{1}, 5 \vee 6 \vee \overline{1}, \overline{5} \vee 6 \vee \overline{1}, 5 \vee \overline{6} \vee \overline{1}$

- The corresponding conflict graph:

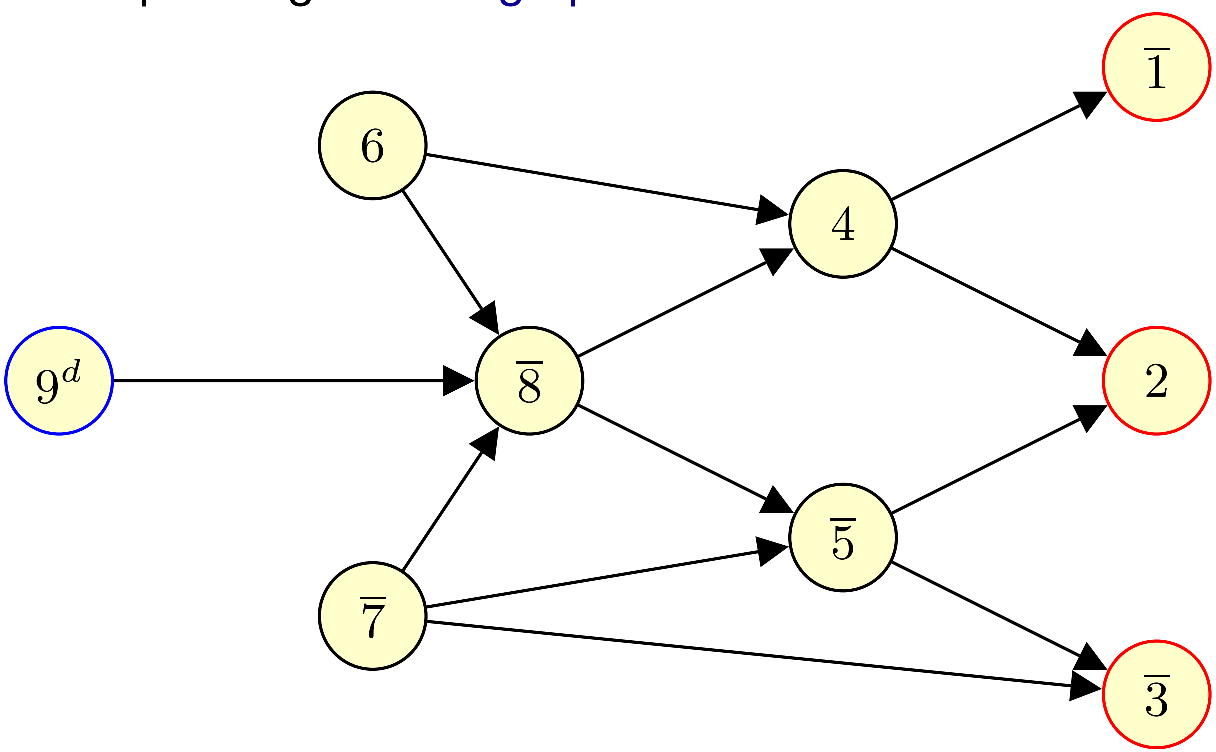


- Literals with cut-outgoing edges are 1 and 5. Literal 5 is the only one from the current decision level, hence all the conditions for the construction of a backjump clause are fulfilled and the backjump clause $\overline{1} \vee \overline{5}$ is obtained with $l' = \overline{5}$.

- The conflict graph analysis does not give the best backjump clause $\overline{1}$, but it is still better than $BackTrace$.

Finding $BackJump$ Clauses: Example 2

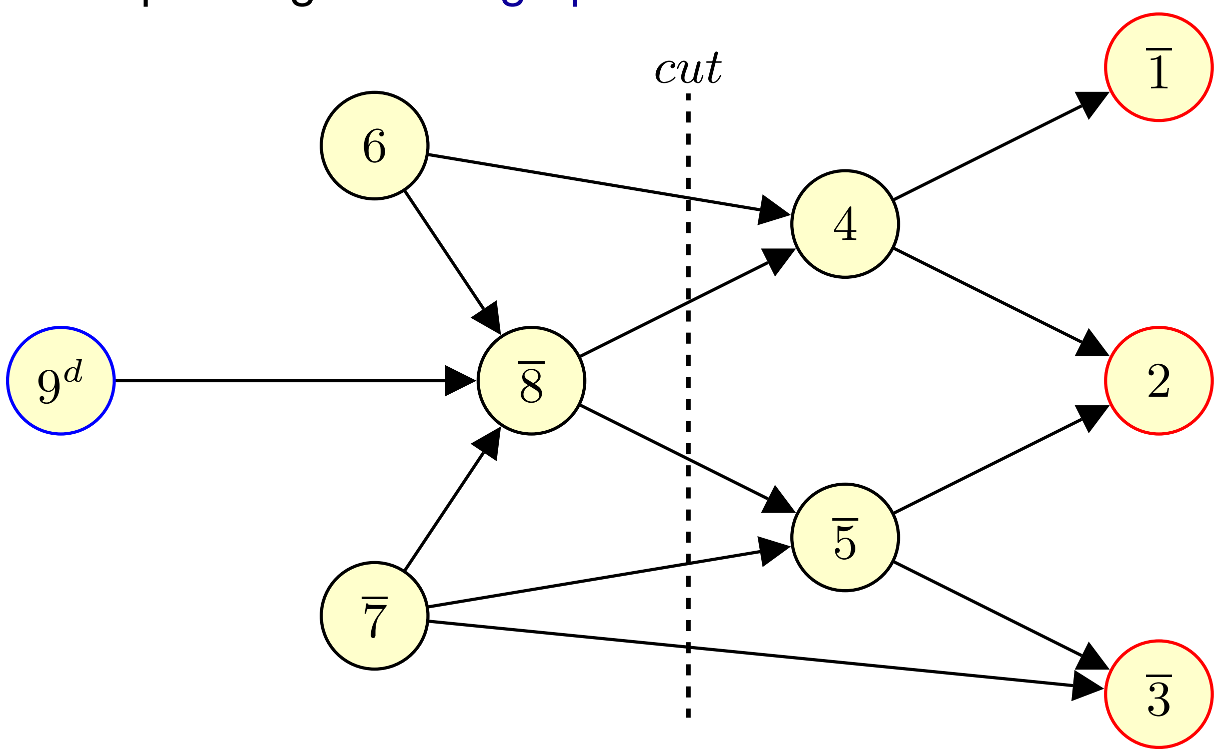
- An example:
  - Consider a state of the form $M \parallel F$ where
    - F, among other clauses, contains:
      $\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}, 8 \vee 7 \vee \overline{5}, \overline{6} \vee 8 \vee 4, \overline{4} \vee \overline{1}, \overline{4} \vee 5 \vee 2, 5 \vee 7 \vee \overline{3}, 1 \vee \overline{2} \vee 3$,
    - M is of the form: $\ldots 6 \ldots \overline{7} \ldots 9^d\ \overline{8}\ \overline{5}\ 4\ \overline{1}\ 2\ \overline{3}$.
  - The corresponding conflict graph:



Finding $BackJump$ Clauses: Example 2

❖ An example:

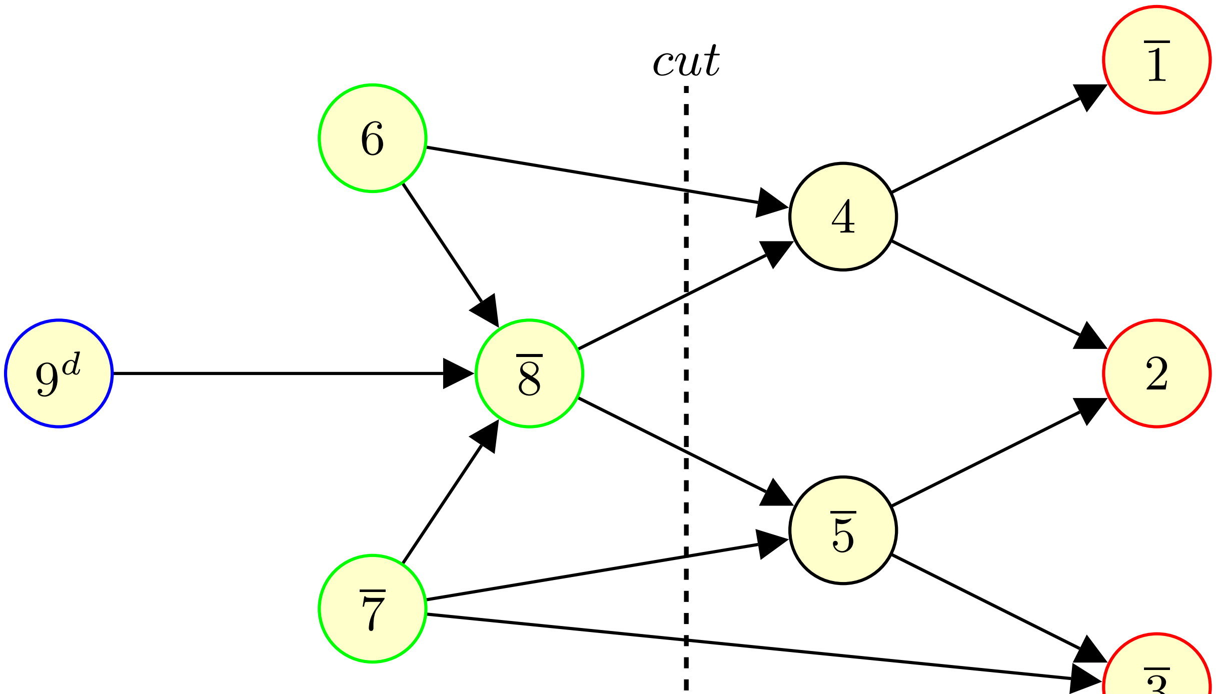
- Consider a state of the form $M \parallel F$ where
  - F, among other clauses, contains:
    $\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}, 8 \vee 7 \vee \overline{5}, \overline{6} \vee 8 \vee 4, \overline{4} \vee \overline{1}, \overline{4} \vee 5 \vee 2, 5 \vee 7 \vee \overline{3}, 1 \vee \overline{2} \vee 3$,
  - M is of the form: $\ldots 6 \ldots \overline{7} \ldots 9^d\ \overline{8}\ \overline{5}\ 4\ \overline{1}\ 2\ \overline{3}$.
- The corresponding conflict graph:



Finding BackJump Clauses: Example 2

❖ An example:

- Consider a state of the form $M \parallel F$ where
  - F, among other clauses, contains:
    $\overline{9} \vee \overline{6} \vee 7 \vee \overline{8}, 8 \vee 7 \vee \overline{5}, \overline{6} \vee 8 \vee 4, \overline{4} \vee \overline{1}, \overline{4} \vee 5 \vee 2, 5 \vee 7 \vee \overline{3}, 1 \vee \overline{2} \vee 3$,
  - M is of the form: $\ldots 6 \ldots \overline{7} \ldots 9^d \; \overline{8} \; \overline{5} \; 4 \; \overline{1} \; 2 \; \overline{3}$.
- The corresponding conflict graph:



The backjump clause is $8 \vee 7 \vee \overline{6}$ and $l' = 8$.

Modern DPLL Systems with Learning

- $Learn$:

$$M \parallel F \Rightarrow M \parallel F,\ C \quad \text{if} \quad \begin{cases} \text{all atoms of } C \text{ occur in } F \text{ or in } M \\ F \models C \end{cases}$$

- The clauses to learn may be, e.g., backjump clauses used in $BackJump$.

- An example:

$$\begin{array}{rcll} 1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} & \Rightarrow_L \quad [BackJump] \\ 1^d\ 2\ \overline{5} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} & \Rightarrow_L \quad [Learn] \\ 1^d\ 2\ \overline{5} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2},\ \overline{2} \vee \overline{5} & \end{array}$$

Modern DPLL Systems: Forgetting

❖ $Forget$:

$M \parallel F,\ C \Rightarrow M \parallel F$ if $\left\{ F \models C \right.$

❖ Forgetting is applied to reduce the number of clauses to deal with.

❖ Typically (though not only), some learnt lemmas may be removed when they become less useful.

- To detect such situations, notions of relevance or activity may be used (in the latter case, one can, e.g., monitor the number of unit propagations or conflicts in which a clause was recently involved).

❖ An example:

$$\begin{array}{llll} 1^d\ 2\ \overline{5} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2},\ \overline{2} \vee \overline{5} & \Rightarrow_L \quad [Forget] \\ 1^d\ 2\ \overline{5} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} & \end{array}$$

Modern DPLL System with Restart

- $Restart$:
  $M \parallel F \Rightarrow \emptyset \parallel F$

- Applied when the search is not making enough progress according to some measure.

- The additional knowledge of the search space compiled into the newly learned lemmas will lead the heuristics for $Decide$ to behave differently, possibly making the procedure to explore the search space in a more compact way.

- To ensure termination, the number of derivation steps between restarting is strictly increasing.

- An example:

$$
\begin{array}{rcll}
1^d\ 2\ 3^d\ 4\ 5^d\ \overline{6} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} & \Rightarrow_R \quad [BackJump] \\
1^d\ 2\ \overline{5} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2} & \Rightarrow_R \quad [Learn] \\
1^d\ 2\ \overline{5} & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2},\ \overline{2} \vee \overline{5} & \Rightarrow_R \quad [Restart] \\
\emptyset & \parallel & \overline{1} \vee 2, \overline{3} \vee 4, \overline{5} \vee \overline{6}, 6 \vee \overline{5} \vee \overline{2},\ \overline{2} \vee \overline{5} &
\end{array}
$$

SMT Solving

Motivation

❖ For many applications, a logic that is richer than propositional logic is needed.

❖ Therefore, using the apparatus of first-order logic (FOL) or even higher-order logics (HOL) becomes necessary, enabling us to use various interesting theories such as:

- difference logic, equality with uninterpreted functions, integer linear arithmetics, real arithmetics, Presburger arithmetics, WSkS, theories of lists, arrays, etc.

❖ To deal with at least some formulae of the above kind, SMT solving, i.e., checking satisfiability modulo theories, extends SAT solving by checking satisfiability of first-order formulae with equality and atoms from various first-order theories (usually decidable ones are only allowed), in some cases even with quantifiers (although their support is limited).

❖ We restrict ourselves to formulae without quantifiers. Then, the only syntactic difference from the purely propositional case is that we consider the set P to contain equality and atomic formulae from various first-order logical theories.

❖ There exist other approaches for checking satisfiability of first-order formulae (e.g., fully automated first-order theorem proving implemented, for instance, in Vampire), which may be, e.g., better in dealing with quantifiers.

❖ Decision procedures for higher-order logics (such as WSKS, MSO) do also exist (e.g., MONA based on tree automata).

First-Order Theories

- For the formulae whose satisfiability is to be checked, the same definitions and notations as in the case of propositional logic will be used with the only difference: the set P over which formulae are built is a countable set of first-order atomic formulae.

- A theory T is a set of closed first-order formulae (axioms). A formula F is T-satisfiable if $F \wedge T$ is satisfiable in the first-order sense. Otherwise, F is T-unsatisfiable.

- Similarly as before, $M \models F$ denotes that an assignment M is a propositional model of a formula F (atomic formulae are viewed as propositional symbols: the theory is ignored).

- If $M \models F \wedge T$, then M is said to be a T-model of F.

- Let F, G be two formulae. If $F \wedge \neg G$ is T-unsatisfiable, then F entails G in T (written $F \models_T G$). A theory lemma is a clause C such that $\emptyset \models_T C$.

- The SMT problem for a theory T is the problem of determining, given a formula F, whether F is T-satisfiable.

Difference Logic

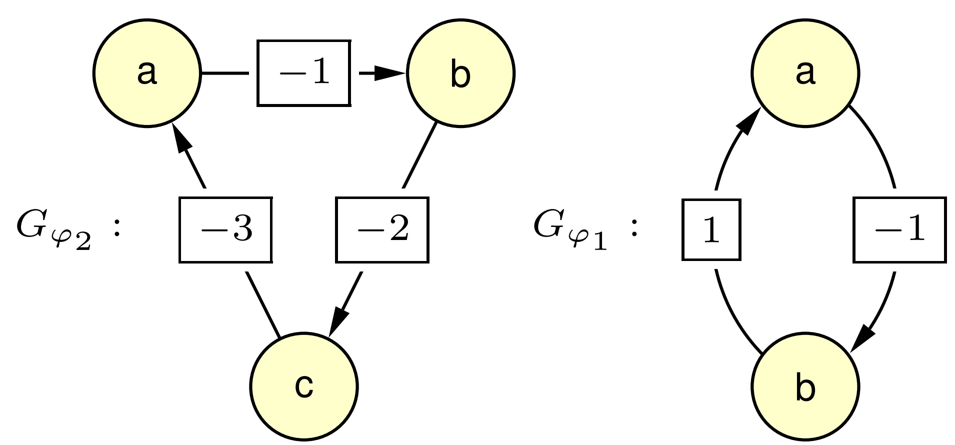
❖ In the theory of difference logic (DL), atomic formulae are syntactically restricted to the form $a - b \leq k$ where a, b are variables and k is a constant. The difference logic can be interpreted over integers, rationals, or reals.

❖ Examples:

- $\varphi_1 : a - b \leq -1 \;\wedge\; b - a \leq 1$
- $\varphi_2 : a - b \leq -1 \;\wedge\; b - c \leq -2 \;\wedge\; c - a \leq -3$

❖ A decision procedure – given a conjunctive DL formula:

- construct a weighted graph with an edge $a \xrightarrow{k} b$ for each atomic formula $a - b \leq k$,
- the formula is satisfiable iff there is no cycle with a negative accumulated weight in the graph.



❖ Constraints of the form $a \leq k$ can be added by transforming them to $a - zero \leq k$ and subsequently shifting a satisfying solution (if there is one) such that $zero$ gets evaluated to 0.

The EUF Theory

- The theory of equality with uninterpreted functions (EUF) consists of the following axioms:

A_r :	$\forall x . x = x$	(reflexivity)
A_s :	$\forall x, y . x = y \rightarrow y = x$	(symmetry)
A_t :	$\forall x, y, z . x = y \wedge y = z \rightarrow x = z$	(transitivity)
A_{c_1} :	for each positive integer n and n-ary function symbol f, $\forall x_1, \ldots, x_n, y_1, \ldots, y_n . x_1 = y_1 \wedge \cdots \wedge x_n = y_n \rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$	(function congruence)
A_{c_2} :	for each positive integer n and n-ary predicate symbol p, $\forall x_1, \ldots, x_n, y_1, \ldots, y_n . x_1 = y_1 \wedge \cdots \wedge x_n = y_n \rightarrow p(x_1, \ldots, x_n) = p(y_1, \ldots, y_n)$	(predicate congruence)

- Equality $=$ is an interpreted predicate symbol.
- Other symbols are uninterpreted.
- An example of a lemma: $a = b \wedge b = c \rightarrow g(f(a), b) = g(f(c), a)$.

The EUF Theory: An Example

❖ Show that $\varphi : (b = c) \wedge (f(b) = c) \wedge (g(f(c)) = a) \wedge (a \neq g(b))$ is unsatisfiable.

- Assume φ is satisfiable, i.e., there is an assignment σ under which φ evaluates to $true$.

1.	$\sigma \models \varphi$	(assumption)	8.	$\sigma \models f(c) = c$	(7,3, transitivity)
2.	$\sigma \models b = c$	(1,$\wedge$)	9.	$\sigma \models c = f(c)$	(8, symmetry)
3.	$\sigma \models f(b) = c$	(1,$\wedge$)	10.	$\sigma \models b = f(c)$	(2,9, transitivity)
4.	$\sigma \models g(f(c)) = a$	(1,$\wedge$)	11.	$\sigma \models g(b) = g(f(c))$	(10, congruence)
5.	$\sigma \models \neg(a = g(b))$	(1,$\wedge$)	12.	$\sigma \models g(b) = a$	(11,4, transitivity)
6.	$\sigma \models f(b) = f(c)$	(2, congruence)	13.	$\sigma \models a = g(b)$	(12, symmetry)
7.	$\sigma \models f(c) = f(b)$	(6, symmetry)	14.	$\sigma \not\models a = g(b)$	(5, $\neg$)

- Lines 13 and 14 are contradictory, therefore, φ is unsatisfiable.
- An example of entailment:
  - $(b = c) \wedge (f(b) = c) \wedge (g(f(c)) = a) \models_T (a = g(b))$.

❖ EUF can be decided by computing the congruence closure of all equivalences and checking that it does not contradict with any inequality.

The Eager SMT Approach

- The idea of eager techniques is to take advantage of the power of existing SAT-solvers. The main steps of the approach are:
  1. Translate the input formula F into a satisfiability-preserving propositional formula F'.
  2. Ask a SAT-solver whether F' is satisfiable.

- However, this approach is not often used in practice since:
  - the translations are specific for each theory,
  - the technique may run out of memory or time quickly on many practical problems,
  - an alternative lazy approach can perform several orders of magnitudes faster.

The Lazy SMT Approach

- Lazy techniques split tasks of the decision procedure into two co-operating components:
  - a propositional SAT-solver, which deals with the boolean skeleton of the given formula and views atomic formulae as simple propositional symbols, and
  - theory solvers (T-solvers), which implement decision procedures for the given theories T. It is sufficient to consider that a T-solver is only able to decide satisfiability of a conjunction of atomic formulae in T.

- The first step is to build a dictionary of atomic predicates that appear in the given formula, recognising their positive and negative appearances.

- An example. Is $F : (a > 3) \wedge (a \leq 3 \vee a < 1 \vee a > 2)$ satisfiable in DL over integers?
  -

$a > 3$	$a \leq 3$	$a < 1$	$a > 2$
p_1	$\neg p_1$	p_2	p_3

  - The SAT-solver views the formula as $p_1 \wedge (\neg p_1 \vee p_2 \vee p_3)$.

- To switch between the propositional and theory views, functions $\mathcal{T}$ and $\mathcal{P}$ will be used.
  - E.g., $\mathcal{T}(p_1 \vee p_2) = (a > 3 \vee a < 1)$ and $\mathcal{P}(a > 3 \vee a < 1) = (p_1 \vee p_2)$.

The Lazy SMT Algorithm

❖ Given an input formula F, a basic version of the lazy SMT approach uses the following interaction of a SAT-solver and a T-solver (assume that a single theory is involved):

1. A SAT-solver checks if $\mathcal{P}(F)$ is satisfiable. If $\mathcal{P}(F)$ is unsatisfiable, then F is T-unsatisfiable, and the procedure terminates. Otherwise, $\mathcal{P}(F)$ is satisfiable, and the procedure continues by the next step.
2. Let M be the satisfying assignment given by the SAT-solver ($M \models \mathcal{P}(F)$). The theory solver checks whether $\mathcal{T}(M)$ is T-satisfiable. If $\mathcal{T}(M)$ is T-satisfiable, F is T-satisfiable, and the procedure terminates. Otherwise, $\mathcal{T}(M)$ is T-unsatisfiable, and the procedure continues by the next step.
3. The T-solver provides a theory lemma (a clause $C = \neg l_1 \vee \ldots \vee \neg l_m$ such that $l_1, \ldots, l_m$ appear in $\mathcal{T}(M)$ and $\emptyset \models_T C$; in an extreme case, $C = \neg\mathcal{T}(M)$) which is added to the set of clauses. Then, the SAT-solver is restarted, and the whole process repeated from Step 1.

❖ Having more theories requires a more complicated algorithm combining them: typically, the Nelson-Oppen combination procedure (or some of its variants) is used.

- We will briefly get to Nelson-Oppen at the end of the lecture – more details are beyond the scope of the lecture.

Lazy SMT: An Illustration

- An example. Is $F : (a > 3) \wedge (a \leq 3 \vee a < 1 \vee a > 2)$ satisfiable?
$a > 3$	$a \leq 3$	$a < 1$	$a > 2$
p_1	$\neg p_1$	p_2	p_3

    $$\frac{(a > 3) \wedge (a \leq 3 \vee a < 1 \vee a > 2)}{p_1 \wedge (\neg p_1 \vee p_2 \vee p_3)}$$
  - Iteration 1. $\mathcal{P}(F) : p_1 \wedge (\neg p_1 \vee p_2 \vee p_3)$
    1. The SAT-solver finds out that F is satisfiable.
    2. The SAT-solver comes with $M = \{p_1, p_2\}$. The T-solver finds out that $\mathcal{T}(M) = a > 3 \wedge a < 1$ is T-unsatisfiable.
    3. The T-solver returns a theory lemma $L = a \leq 3 \vee a \geq 1$, which is added as a new clause to the formula ($F := F \wedge L$).
  - Iteration 2. $\mathcal{P}(F) : p_1 \wedge (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_1 \vee \neg p_2)$
    1. The SAT-solver finds out that F is satisfiable.
    2. The SAT-solver comes with $M = \{p_1, \neg p_2, p_3\}$. The T-solver confirms that $\mathcal{T}(M) = a > 3 \wedge a \geq 1 \wedge a > 2$ is T-satisfiable and the procedure answers that F is satisfiable (for $a > 3$, after simplification).

Lazy SMT: Extensions

❖ To summarise, there are two causes for the current assignment M not to be satisfying during a run of an SMT decision procedure:

- standard propositional (detected by $BackJump$ and $Fail$ rules), or
- theory-related: a contradiction occurs in the current assignment. This is not (and cannot be) detected by a SAT-solver itself, and so a T-solver must be used to discover it.

❖ There exist various extensions of the basic lazy SMT, e.g.,:

- On-line SAT-solver. The presented version of a lazy procedure is off-line, meaning that the procedure is restarted when a new lemma is learned. Alternatively, in an on-line approach, the SAT-solver continues from the current assignment after learning a lemma (and since the learned lemma is always a conflicting clause, it can proceed either by applying the $BackJump$ or $Fail$ rule).
- Incremental SAT-solver. The SAT-solver can query a T-solver continuously during the construction of an assignment M about its T-satisfiability. In other words, it does not have to wait until a propositionally satisfying assignment is found. Depending on the cost of such queries, they can be made after each step of the SAT-solver or at regular intervals.

Theory Propagation

❖ Theory propagation is a principle which will later be incorporated into abstract DPLL(T) systems: an extension of abstract DPLL systems for SMT procedures. Similarly as in the propositional DPLL, definitions will be made in terms of rules.

❖ Up to now, the T-solver has been used to validate that an assignment M is T-satisfiable only. However, it can also be used to guide the SAT-solver. This is achieved by the following rule.

❖ $TheoryPropagate$:

$$M \parallel F \Rightarrow Ml \parallel F \quad \text{if} \quad \begin{cases} M \models_T l \\ l \text{ or } \neg l \text{ occurs in } F \\ l \text{ is undefined in } M \end{cases}$$

❖ In a state $M \parallel F$, the rule allows to add all (or at least some) literals $l_1, \ldots, l_m$ undefined in M where l_i or $\neg l_i$ occurs in F and $M \models_T l_i$, i.e., literals which are entailed in T by the current assignment. Then, the system can move to state $Ml_1, \ldots, l_m \parallel F$.

❖ Depending on how expensive the theory propagation is (for a specific solver), it can be performed exhaustively (performs all possible theory propagations) or non-exhaustively (propagates only literals which are cheap to compute).

Abstract DPLL(T)

❖ SMT solving can be formalised in terms of abstract DPLL(T) systems that extend abstract DPLL systems.

❖ The rules of abstract DPLL(T) systems can be split into three groups:

- DPLL rules: $Decide$, $Fail$, $UnitPropagate$, $Restart$, which are exactly the same as in SAT solving. (Note: no $PureLiteral$ due to having to consider the theory too.)
- DPLL/theory rules: modified $Learn$, $Forget$, and $BackJump$ rules where some of the original propositional entailments are replaced by theory entailment.
- Theory rules: the $TheoryPropagate$ rule.

❖ Note that each propositional entailment is a theory entailment, and so purely propositional reasoning may still be used in the modified $BackJump$, $Learn$, and $Forget$ – apart from that, theory reasoning is allowed too (used, e.g., to learn theory lemmas or theory-dependent backjump clauses).

❖ We denote derivations in the system with all the above rules $\Rightarrow_{FT}$ (Full DPLL Modulo Theories).

Abstract DPLL(T)

- $T - BackJump$:

$$M l^d N \parallel F, C \Rightarrow M l' \parallel F, C \quad \text{if} \quad \begin{cases} M l^d N \models \neg C, \text{ and there is} \\ \text{some clause } C' \vee l' \text{ such that:} \\ \quad F, C \models_T C' \vee l' \text{ and } M \models \neg C', \\ \quad l' \text{ is undefined in } M, \text{ and} \\ \quad l' \text{ or } \neg l' \text{ occurs in } F \text{ or in } M l^d N \end{cases}$$

- $T - Learn$:

$$M \parallel F \Rightarrow M \parallel F, C \quad \text{if} \quad \begin{cases} \text{all atoms of } C \text{ occur in } F \text{ or in } M \\ F \models_T C \end{cases}$$

  - Applied in association with $T - BackJump$ or when a satisfying M is T-inconsistent and a theory lemma is to be added.

- $T - Forget$:

$$M \parallel F, C \Rightarrow M \parallel F \quad \text{if} \quad \left\{ M \models_T C \right.$$

Theory Propagation: Example 1

- Is the following difference logic formula satisfiable?

$$\varphi : ((x > 2) \vee (x < -15)) \wedge (x > -10) \wedge ((x < 0) \vee (x > 0))$$

Let us abbreviate φ as $(\varphi_1 \vee \varphi_2) \wedge \varphi_3 \wedge (\varphi_4 \vee \varphi_5)$.

$$\begin{array}{rcccl}
\emptyset & \| & \varphi & \Rightarrow_{FT} & [UnitPropagate\ \varphi_3] \\
x > -10 & \| & \varphi & \Rightarrow_{FT} & [TheoryPropagate\ \varphi_2] \\
x > -10, \overline{x < -15} & \| & \varphi & \Rightarrow_{FT} & [UnitPropagate\ \varphi_1] \\
x > -10, \overline{x < -15}, x > 2 & \| & \varphi & \Rightarrow_{FT} & [TheoryPropagate\ \varphi_4] \\
x > -10, \overline{x < -15}, x > 2, \overline{x < 0} & \| & \varphi & \Rightarrow_{FT} & [UnitPropagate\ \varphi_5] \\
x > -10, \overline{x < -15}, x > 2, \overline{x < 0}, x > 0 & \| & \varphi & &
\end{array}$$

Theory Propagation: Example 2

- Consider the EUF logic and a clause set F containing:

$$\begin{array}{ll} & \ldots \\ (1) & (a = b) \vee (g(a) \neq g(b)) \\ (2) & (h(a) = h(c)) \vee p \\ (3) & (g(a) = g(b)) \vee \neg p \\ & \ldots \end{array}$$

- Let the DPLL(T) procedure be in a state: $M, c = b, f(a) \neq f(b) \parallel F$.

- Consider the derivation:

	Step	New Literal	Reason
(I)	$Decide$	$h(a) \neq h(c)$	
(II)	$TheoryPopagate$	$a \neq b$	since $h(a) \neq h(c) \wedge c = b \models_T a \neq b$
(III)	$UnitPropagate$	$g(a) \neq g(b)$	because of $a \neq b$ and (1)
(IV)	$UnitPropagate$	p	because of $h(a) \neq h(c)$ and (2)

- Now, Clause (3) is conflicting. A possible backjump clause is, e.g., $h(a) = h(c) \vee c \neq b$.

Combining Theories

The Nelson-Oppen Method

- Assume theories with disjoint sets of function and predicate symbols (up to equality).
- Variables used in atoms falling into different theories may be shared.
- The very basic idea of the Nelson-Oppen theory combination method is the following:
  - T-solvers must report all equalities among variables.
  - When the given formula is propositionally satisfiable and the derived assignment M is T-consistent for each involved theory T and the part of M which involves atoms of T, all equalities discovered in each theory are added within the theories in which they were not discovered, and the process is re-run till either no new equalities are found or a conflict is found.
- If a contradiction is ever found, then the formula is clearly unsatisfiable. For satisfiability, the following conditions suffice:
  - Theories must admit countably infinite models.
  - Theories must be convex.
    - A theory is convex iff whenever a satisfiable conjunction of literals entails a disjunction of equalities of variables, then it entails one of the equalities.
    - Biggest obstacle in practice, yet applicable for many common theories.