CS 512, Spring 2017, Handout 12

SAT Solvers

Assaf Kfoury

12 February 2017

Origins and background

- As it name indicates, a **SAT solver** deals with *satisfiability* of propositional WFF's.
- Like the tableau method and the resolution method, SAT solvers are **refutation-based**, *i.e.*, they try to find reasons why a WFF φ is a logical contradiction.
- A SAT solver always terminates; if it terminates in what is called "failed state", then φ is a logical contradiction, otherwise φ is a satisfiable WFF.
- Like the tableaux method and the resolution method, a SAT solver as a decision procedure is only **refutation complete**, not **complete**, though this does not prevent us from using a SAT solver to decide semantic entailment in general.
- Refutation completeness of a SAT solver means that, if ψ is an unsatisfiable WFF (the last column in the truth-table of ψ are all **F**'s, expressed by the semantic entailment $\psi \models \bot$ or $\models \psi \to \bot$), then a SAT solver will confirm it by a terminating process of symbolic manipulation (pattern-matching, backtracking, term-rewriting) ending with "failed state" (which we can express by writing $\psi \vdash_{\text{SAT solver}} \bot$).
- There are many restrictions and extensions of the **satisfiability problem** (not covered in this course), some efficiently solvable and some just as hard (or harder) to solve as the unrestricted **satisfiability problem**. Click here for a survey.

Preprocessing required by SAT solvers

- SAT solvers, like the resolution method but unlike the tableaux method, require that an input WFF φ be in CNF.
- Typically the input WFF φ in CNF is written as a finite set of clauses, *i.e.*, $\varphi = \{C_1, \ldots, C_n\}$ where every C_i is a finite disjunction of literals (propositional variables and negated propositional variables).
- A SAT solver can start from a WFF φ in CNF, rather than in some other special form (*e.g.*, DNF), because φ can be translated into an equisatisfiable WFF φ' efficiently, specifically, in linear time – see pp 3-6 in Handout 11.

  (We already know that converting a propositional WFF ψ into an equivalent DNF ψ' is a NP-hard problem. See Handout 10, the slides with heading "Why DNF?", and also click here. But how about instead converting a propositional WFF ψ into an equisatisfiable, not necessarily equivalent, DNF ψ'? **No**, it is a bad idea.)

  Exercise: Show it is unlikely we will ever find an algorithm to transform an arbitrary propositional WFF ψ into an equisatisfiable DNF ψ' efficiently – unless P = NP.

  Hint: Satisfiability of a DNF can be carried out in low-degree polynomial time.
- Depending on the underlying algorithm and data structures it uses, a SAT solver requires further *preprocessing*, and also *inprocessing*, of the input WFF φ for the purpose of speeding up its execution.

Two main approaches to SAT solvers

1. SAT solvers based on **stochastic search**: The solver first guesses a **full assignment** (also called **full valuation**), *i.e.*, an assignment of truth-values to all propositional atoms. If the WFF evaluates to **F** under this assignment, it starts to flip truth-values of the atoms according to some heuristics. Typically, it counts the number of unsatisfied clauses and chooses the flip that minimizes this number.

Two main approaches to SAT solvers

1. SAT solvers based on **stochastic search**: The solver first guesses a **full assignment** (also called **full valuation**), *i.e.*, an assignment of truth-values to all propositional atoms. If the WFF evaluates to **F** under this assignment, it starts to flip truth-values of the atoms according to some heuristics. Typically, it counts the number of unsatisfied clauses and chooses the flip that minimizes this number.

2. SAT solvers based on **exhaustive search**: The solver traverses a binary tree, in which internal nodes are **partial valuations** and leaves are **full valuations**, and repeatedly **backtracks** in search of a satisfying full valuation.

   SAT solvers based on exhaustive search use what is known as the **DPLL** procedure, or a refined and more efficient version of the original **DPLL** procedure.

   The acronym "**DPLL**" stands for Martin **D**avis (1928-), Hilary **P**utnam (1926-2016), George **L**ogemann (1938-2012), and Donald **L**oveland (1934-) – the four mathematical logicians and computer scientists behind the early development of the procedure in the 1960's and the 1970's.

   Davis and **P**utnam here are the same who introduced the **resolution method** and, naturally enough, the DPLL procedure can be viewed an an extension and refinement of the resolution method.

Focus on DPLL

- The rest of this handout is on SAT solvers based on **exhaustive search**, *i.e.*, based on the **DPLL procedure** or one of its many variants.
- Good reason for this: Most modern SAT solvers are based on **exhaustive search**.

Focus on DPLL

- The rest of this handout is on SAT solvers based on **exhaustive search**, *i.e.*, based on the **DPLL procedure** or one of its many variants.
- Good reason for this: Most modern SAT solvers are based on **exhaustive search**.

- I choose a presentation in two parts:
  1. What I call the "Classical DPLL" procedure, which explains the basic DPLL approach.

     More than one way of doing this. My presentation is based on:
     R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories", *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.

  2. What I call "Modern Extensions of Classical DPLL" (some of them, not all of them), which explains ways that make the classical procedure perform more efficiently.

Classical DPLL Procedure: What Is a *Partial Valuation*?

- We formulate the **Classical DPLL** procedure as a ***transition system*** consisting of 5 ***transition rules***, which are used to operate over a domain of ***states***.
- The notion of a state requires a preliminary definition of ***partial valuation***.

Classical DPLL Procedure: What Is a *Partial Valuation*?

- We formulate the **Classical DPLL** procedure as a ***transition system*** consisting of 5 ***transition rules***, which are used to operate over a domain of ***states***.
- The notion of a state requires a preliminary definition of ***partial valuation***.

Definition

- If φ is a propositional WFF, then a *partial valuation* (or *model*) $\mathcal{M}$ for φ is a an assignment of truth values *to some* (and possibly – but not necessarily – *to all*) the propositional atoms in φ.

Classical DPLL Procedure: What Is a *Partial Valuation*?

- We formulate the **Classical DPLL** procedure as a ***transition system*** consisting of 5 ***transition rules***, which are used to operate over a domain of ***states***.
- The notion of a state requires a preliminary definition of ***partial valuation***.

Definition

- If φ is a propositional WFF, then a *partial valuation* (or *model*) $\mathcal{M}$ for φ is a an assignment of truth values *to some* (and possibly – but not necessarily – *to all*) the propositional atoms in φ.
- If $\mathcal{M}$ is a partial valuation for φ, we write $\mathcal{M}$ as a sequence of atoms or negated atoms occurring in φ.

Classical DPLL Procedure: What Is a *Partial Valuation*?

- We formulate the **Classical DPLL** procedure as a ***transition system*** consisting of 5 ***transition rules***, which are used to operate over a domain of ***states***.
- The notion of a state requires a preliminary definition of ***partial valuation***.

Definition

- If φ is a propositional WFF, then a *partial valuation* (or *model*) $\mathcal{M}$ for φ is a an assignment of truth values *to some* (and possibly – but not necessarily – *to all*) the propositional atoms in φ.
- If $\mathcal{M}$ is a partial valuation for φ, we write $\mathcal{M}$ as a sequence of atoms or negated atoms occurring in φ.
- For example, if $\varphi := \neg\big((q_1 \vee \neg q_2) \wedge q_3\big)$, then a partial valuation $\mathcal{M}$ for φ may be the sequence $\neg q_1\ q_3$ meaning that $\mathcal{M}$ assigns **F** to q_1 and **T** to q_3.
  Fact: In this example, $\mathcal{M}$ can be extended to a total valuation that satisfies φ by assigning **T** to q_2.

Classical DPLL Procedure: What Is a *Partial Valuation*?

- We formulate the **Classical DPLL** procedure as a ***transition system*** consisting of 5 ***transition rules***, which are used to operate over a domain of ***states***.
- The notion of a state requires a preliminary definition of ***partial valuation***.

Definition

- If φ is a propositional WFF, then a *partial valuation* (or *model*) $\mathcal{M}$ for φ is a an assignment of truth values *to some* (and possibly – but not necessarily – *to all*) the propositional atoms in φ.
- If $\mathcal{M}$ is a partial valuation for φ, we write $\mathcal{M}$ as a sequence of atoms or negated atoms occurring in φ.
- For example, if $\varphi := \neg\big((q_1 \vee \neg q_2) \wedge q_3\big)$, then a partial valuation $\mathcal{M}$ for φ may be the sequence $\neg q_1\ q_3$ meaning that $\mathcal{M}$ assigns **F** to q_1 and **T** to q_3.
  Fact: In this example, $\mathcal{M}$ can be extended to a total valuation that satisfies φ by assigning **T** to q_2.
- Another partial valutation $\mathcal{M}'$ for $\varphi := \neg\big((q_1 \vee \neg q_2) \wedge q_3\big)$ may be the sequence $q_1\ q_3$ meaning that $\mathcal{M}$ assigns **T** to both q_1 and q_3.
  Fact: $\mathcal{M}'$ can**not** be extended to a total valuation that satisfies φ.

Classical DPLL Procedure: What Is a *State*?

Definition

- A *state* in the Classical DPLL is a pair of the form $\mathcal{M} \| \varphi$ where φ is a propositional WFF in CNF and $\mathcal{M}$ is a partial valuation for φ.

Classical DPLL Procedure: What Is a *State*?

Definition

- A *state* in the Classical DPLL is a pair of the form $\mathcal{M} \| \varphi$ where φ is a propositional WFF in CNF and $\mathcal{M}$ is a partial valuation for φ.
- For example, a state of Classical DPLL may look like:

$$p_1\ p_2\ \neg q_1\ \ \|\ \ \{\, p_2,\ \neg p_1 \vee \neg q_1,\ \neg p_1 \vee q_2,\ q_1 \vee \neg q_2 \vee p_1,\ \neg p_2 \vee p_1 \vee \neg q_3,\ \neg p_1 \vee p_2,\ q_3 \vee p_2 \,\}$$

  where

  - the partial valuation (left of "$\|$") assigns **T** to p_1, **T** to p_2, and **F** to q_1,
  - the CNF (right of "$\|$") is here written as a set of 7 clauses.

Classical DPLL Procedure: What Is a *State*?

Definition

- A *state* in the Classical DPLL is a pair of the form $\mathcal{M} \| \varphi$ where φ is a propositional WFF in CNF and $\mathcal{M}$ is a partial valuation for φ.
- For example, a state of Classical DPLL may look like:

$$p_1\ p_2\ \neg q_1\ \ \|\ \ \{\, p_2,\ \neg p_1 \vee \neg q_1,\ \neg p_1 \vee q_2,\ q_1 \vee \neg q_2 \vee p_1,\\ \neg p_2 \vee p_1 \vee \neg q_3,\ \neg p_1 \vee p_2,\ q_3 \vee p_2 \,\}$$

  where

  - the partial valuation (left of "$\|$") assigns **T** to p_1, **T** to p_2, and **F** to q_1,
  - the CNF (right of "$\|$") is here written as a set of 7 clauses.

- **Fact:** In the preceding example, the partial valuation (left of "$\|$") can be extended to a total valuation, namely, "$p_1\ p_2\ \neg q_1\ q_2$", that satisfies the CNF (right of "$\|$").

Classical DPLL Procedure: 5 Transition Rules

UnitPropagate

$\mathcal{M} \parallel \varphi \cup \{C \vee \ell\} \quad \Longrightarrow \quad \mathcal{M}\,\ell \parallel \varphi \cup \{C \vee \ell\}$ **if** $\mathcal{M} \models \neg C$ and
ℓ is undefined in $\mathcal{M}$.

PureLiteral

$\mathcal{M} \parallel \varphi \quad \Longrightarrow \quad \mathcal{M}\,\ell \parallel \varphi$ **if** ℓ occurs in a clause of φ,
$\neg\ell$ occurs in no clause of φ,
and ℓ is undefined in $\mathcal{M}$.

Decide

$\mathcal{M} \parallel \varphi \quad \Longrightarrow \quad \mathcal{M}\,\ell^{\mathsf{d}} \parallel \varphi$ **if** ℓ or $\neg\ell$ occurs in a clause of φ
and ℓ is undefined in $\mathcal{M}$.

Fail

$\mathcal{M} \parallel \varphi \cup \{C\} \quad \Longrightarrow \quad$ FailState **if** $\mathcal{M} \models \neg C$ and
$\mathcal{M}$ contains no decision literals.

Backtrack

$\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N} \parallel \varphi \cup \{C\} \quad \Longrightarrow \quad \mathcal{M}\,\neg\ell \parallel \varphi \cup \{C\}$ **if** $\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N} \models \neg C$ and
$\mathcal{N}$ contains no decision literals.

Classical DPLL Procedure: Example

Below is a derivation by Classical DPLL. For better readibility:

- We denote the atoms $q_1, q_2, q_3, \ldots$ by their indeces $1, 2, 3, \ldots$.
- We denote the negation $\neg q_k$ by $\bar{k}$.

Classical DPLL Procedure: Example

Below is a derivation by Classical DPLL. For better readibility:

- We denote the atoms $q_1, q_2, q_3, \ldots$ by their indeces $1, 2, 3, \ldots$.
- We denote the negation $\neg q_k$ by $\bar{k}$.

$$
\begin{array}{rcll}
\varnothing & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(Decide)} \\
1^{\mathsf{d}} & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(UnitPropagate)} \\
1^{\mathsf{d}}\,\bar{2} & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(UnitPropagate)} \\
1^{\mathsf{d}}\,\bar{2}\,3 & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(UnitPropagate)} \\
1^{\mathsf{d}}\,\bar{2}\,3\,4 & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(Backtrack)} \\
\bar{1} & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(UnitPropagate)} \\
\bar{1}\,4 & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(Decide)} \\
\bar{1}\,4\,\bar{3}^{\mathsf{d}} & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 & \Longrightarrow \text{(UnitPropagate)} \\
\bar{1}\,4\,\bar{3}^{\mathsf{d}}\,2 & \| & \bar{1} \vee \bar{2},\ 2 \vee 3,\ \bar{1} \vee \bar{3} \vee 4,\ 2 \vee \bar{3} \vee \bar{4},\ 1 \vee 4 &
\end{array}
$$

Classical DPLL Procedure

Definition

A state S is a ***final state*** if one of two conditions holds:

1. S is the token "FailState",
2. S is of the form $\mathcal{M} \parallel \varphi$ where $\mathcal{M}$ is a total valuation for the CNF φ.

Classical DPLL Procedure

Definition

A state S is a ***final state*** if one of two conditions holds:

1. S is the token "FailState",
2. S is of the form $\mathcal{M} \parallel \varphi$ where $\mathcal{M}$ is a total valuation for the CNF φ.

Theorem

Let φ be a WFF in CNF. Then:

1. *Every derivation by Classical DPLL which starts with the state $\varnothing \parallel \varphi$ always terminates with a final state, i.e.:*

$$\varnothing \parallel \varphi \implies S_1 \implies \cdots \implies S_n$$

for some $n \geqslant 1$ and where S_n is a final state.

Classical DPLL Procedure

Definition

A state S is a ***final state*** if one of two conditions holds:

1. S is the token "FailState",
2. S is of the form $\mathcal{M} \parallel \varphi$ where $\mathcal{M}$ is a total valuation for the CNF φ.

Theorem

Let φ be a WFF in CNF. Then:

1. *Every derivation by Classical DPLL which starts with the state $\varnothing \parallel \varphi$ always terminates with a final state, i.e.:*

   $$\varnothing \parallel \varphi \implies S_1 \implies \cdots \implies S_n$$

   for some $n \geqslant 1$ and where S_n is a final state.
2. *If the final state S_n is of the form $\mathcal{M} \parallel \varphi$, then φ is satisfiable and the total valuation $\mathcal{M}$ is a model of φ.*

Classical DPLL Procedure

Definition

A state S is a ***final state*** if one of two conditions holds:

1. S is the token "FailState",
2. S is of the form $\mathcal{M} \parallel \varphi$ where $\mathcal{M}$ is a total valuation for the CNF φ.

Theorem

Let φ be a WFF in CNF. Then:

1. *Every derivation by Classical DPLL which starts with the state $\varnothing \parallel \varphi$ always terminates with a final state, i.e.:*

   $$\varnothing \parallel \varphi \implies S_1 \implies \cdots \implies S_n$$

   for some $n \geqslant 1$ and where S_n is a final state.
2. *If the final state S_n is of the form $\mathcal{M} \parallel \varphi$, then φ is satisfiable and the total valuation $\mathcal{M}$ is a model of φ.*
3. *If the final state S_n is the token "FailState", then φ is unsatisfiable.*

Proof.

Left to you. □

Modern Extensions of the DPLL Procedure

- Modern SAT solvers are based on the classical DPLL procedure, in which they introduce several modifications for efficiency.

Modern Extensions of the DPLL Procedure

- Modern SAT solvers are based on the classical DPLL procedure, in which they introduce several modifications for efficiency.
- Among the efficiencies introduced in modern SAT solvers:
  - Rule **PureLiteral** is used as a pre-processing step and excluded from the rules driving the solver, *i.e.*, it is applied repeatedly before all other rules until it cannot be applied, after which it is not used.

Modern Extensions of the DPLL Procedure

- Modern SAT solvers are based on the classical DPLL procedure, in which they introduce several modifications for efficiency.
- Among the efficiencies introduced in modern SAT solvers:
  - Rule **PureLiteral** is used as a pre-processing step and excluded from the rules driving the solver, *i.e.*, it is applied repeatedly before all other rules until it cannot be applied, after which it is not used.
  - Rule **Backtrack** is replaced by a more general and powerful backtracking mechanism, the so-called **Backjump** rule.
  - . . . and other efficiency modifications.

Modern Extensions of the DPLL Procedure

- A basic modern SAT solver omits the rule **PureLiteral** from the Classical DPLL procedure, but includes the 3 rules:

  UnitPropagate, **Decide**, and **Fail** (as before),

  together with (at least) the new rule **Backjump** instead of **Backtrack**.

[1] If you are interested, an examination is in R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories", *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.

Modern Extensions of the DPLL Procedure

- A basic modern SAT solver omits the rule **PureLiteral** from the Classical DPLL procedure, but includes the 3 rules:

  UnitPropagate, **Decide**, and **Fail** (as before),

  together with (at least) the new rule **Backjump** instead of **Backtrack**.

- **Backjump**

$$\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N} \parallel \varphi \cup \{C\} \quad \Longrightarrow \quad \mathcal{M}\,\ell' \parallel \varphi \cup \{C\}$$

if $\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N} \models \neg C$ and there is some clause $C' \vee \ell'$ such that:

1. $\varphi \cup \{C\} \models C' \vee \ell'$,
2. $\mathcal{M} \models \neg C'$,
3. ℓ' is undefined in $\mathcal{M}$, and
4. ℓ' or $\neg\ell'$ occurs in φ or in $\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N}$.

---

[1] If you are interested, an examination is in R. Nieuwenhuis, A. Oliveras, and C. Tinelli, “Solving SAT and SAT Modulo Theories”, *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.

Modern Extensions of the DPLL Procedure

- A basic modern SAT solver omits the rule **PureLiteral** from the Classical DPLL procedure, but includes the 3 rules:

  UnitPropagate, **Decide**, and **Fail** (as before),

  together with (at least) the new rule **Backjump** instead of **Backtrack**.

- **Backjump**

$$\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N} \parallel \varphi \cup \{C\} \quad \Longrightarrow \quad \mathcal{M}\,\ell' \parallel \varphi \cup \{C\}$$

if $\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N} \models \neg C$ and there is some clause $C' \vee \ell'$ such that:

1. $\varphi \cup \{C\} \models C' \vee \ell'$,
2. $\mathcal{M} \models \neg C'$,
3. ℓ' is undefined in $\mathcal{M}$, and
4. ℓ' or $\neg\ell'$ occurs in φ or in $\mathcal{M}\,\ell^{\mathsf{d}}\,\mathcal{N}$.

- Although **Backjump** is more efficient than **Backtrack**, it is a little more difficult to understand.[1]

---

[1] If you are interested, an examination is in R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories", *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.

Modern Extensions of the DPLL Procedure

- For efficiency, it turns out that **Backjump** works even better in the presence of two (non-essential, but more helpful for backtracking) rules:

  Forget and **Learn**

[2] Again, if you are interested, an examination is in R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories", *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.

Modern Extensions of the DPLL Procedure

- For efficiency, it turns out that **Backjump** works even better in the presence of two (non-essential, but more helpful for backtracking) rules:

  Forget and **Learn**

Forget

$\mathcal{M} \parallel \varphi \cup \{C\} \implies \mathcal{M} \parallel \varphi$ **if** $\varphi \models C$

Learn

$\mathcal{M} \parallel \varphi \implies \mathcal{M} \parallel \varphi \cup \{C\}$ **if** $\varphi \models C$ and each atom of C occurs in φ or in $\mathcal{M}$

---

[2] Again, if you are interested, an examination is in R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories", *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.

Modern Extensions of the DPLL Procedure

- For efficiency, it turns out that **Backjump** works even better in the presence of two (non-essential, but more helpful for backtracking) rules:

  Forget and **Learn**

Forget

$\mathcal{M} \parallel \varphi \cup \{C\} \implies \mathcal{M} \parallel \varphi$ **if** $\varphi \models C$

Learn

$\mathcal{M} \parallel \varphi \implies \mathcal{M} \parallel \varphi \cup \{C\}$ **if** $\varphi \models C$ and each atom of C occurs in φ or in $\mathcal{M}$

- Although the soundness of **Forget** and **Learn** is relatively easy to understand, *i.e.*, "their presence does not turn an unsatisfiable WFF into a satisfiable WFF," it is more difficult to understand why they improve efficiency (in conjunction with **Backjump**).[2]

[2] Again, if you are interested, an examination is in R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Solving SAT and SAT Modulo Theories", *Journal of the ACM*, Vol. 53, No. 6, November 2006, pp. 937-977.