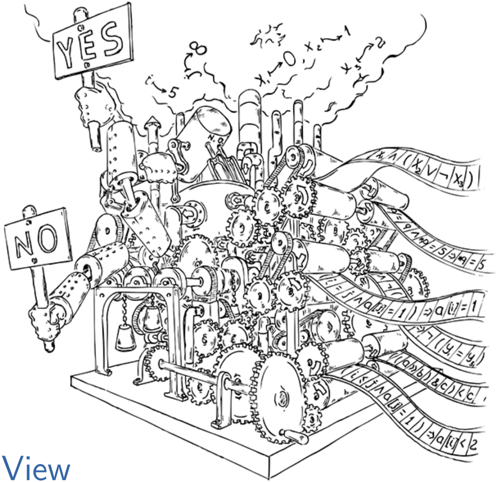
Propositional Encodings

Chapter 11



Decision Procedures

An Algorithmic Point of View

D.Kroening O.Strichman

Revision 1.0

Outline

1. Overview
2. Notation
3. A Basic Encoding Algorithm
4. Integration into DPLL
5. Theory Propagation and the DPLL(T) Framework
6. Theory Propagation and the DPLL(T) Framework
7. Optimizations and Implementation Issues

- Let T be a first-order Σ-theory such that:
  - T is quantifier-free.
  - There exists a decision procedure, denoted DP_T, for the conjunctive fragment of T.

- Example 1:
  - T is equality logic.
  - DP_T is the congruence closure algorithm.

- Example 2:
  - T is disjunctive linear arithmetic.
  - DP_T is the Simplex algorithm.

We will now study a framework that combines

- DP_T, and
- a SAT solver,

in various ways, in order to construct a decision procedure for T.

Propositional Encodings

We will now study a framework that combines

- DP_T, and
- a SAT solver,

in various ways, in order to construct a decision procedure for T.

This method is

- modular,
- efficient,
- competitive (all state-of-the-art SMT solvers work this way).

The two main engines in this framework work in tight collaboration:

- The SAT solver chooses those literals that need to be satisfied in order to satisfy the Boolean structure of the formula, and
- The theory solver DP_T checks whether this choice is consistent in T.

Let l be a Σ-literal.

- Denote by $e(l)$ the Boolean encoder of this literal.

Let t be a Σ-formula,

- Denote by $e(t)$ the Boolean formula resulting from substituting each Σ-literal in t with its Boolean encoder.

Notation

Let l be a Σ-literal.

- Denote by $e(l)$ the Boolean encoder of this literal.

Let t be a Σ-formula,

- Denote by $e(t)$ the Boolean formula resulting from substituting each Σ-literal in t with its Boolean encoder.

For a Σ-formula t, the resulting Boolean formula $e(t)$ is called the propositional skeleton of t.

- Example I: Let $l := x = y$ be a Σ-literal. Then $e(x = y)$, a Boolean variable, is its encoder.

Boolean encoders – examples

- Example I: Let $l := x = y$ be a Σ-literal. Then $e(x = y)$, a Boolean variable, is its encoder.

- Example II: Let

$$t := \; x = y \vee x = z$$

be a Σ-formula. Then

$$e(t) := \; e(x = y) \vee e(x = z)$$

is its Boolean encoder.

Let T be equality logic. Given an NNF formula

$$\varphi := \ x = y \wedge ((y = z \wedge x \neq z) \vee x = z)\ , \tag{1}$$

we begin by computing its propositional skeleton:

Let T be equality logic. Given an NNF formula

$$\varphi := \; x = y \wedge ((y = z \wedge x \neq z) \vee x = z) \, , \tag{1}$$

we begin by computing its propositional skeleton:

$$e(\varphi) := \; e(x = y) \wedge ((e(y = z) \wedge e(x \neq z)) \vee e(x = z)) \, . \tag{2}$$

Let T be equality logic. Given an NNF formula

$$\varphi := \; x = y \wedge ((y = z \wedge x \neq z) \vee x = z) \,, \tag{1}$$

we begin by computing its propositional skeleton:

$$e(\varphi) := \; e(x = y) \wedge ((e(y = z) \wedge e(x \neq z)) \vee e(x = z)) \,. \tag{2}$$

Note that since we are encoding *literals* and not *atoms*, $e(\varphi)$ has no negations and hence is trivially satisfiable.

Let $\mathcal{B}$ be a Boolean formula, initially set to $e(\varphi)$, i.e.,

$$\mathcal{B} := e(\varphi) .$$

Let $\mathcal{B}$ be a Boolean formula, initially set to $e(\varphi)$, i.e.,

$$\mathcal{B} := \; e(\varphi)\;.$$

As a second step, we pass $\mathcal{B}$ to a SAT solver.

Let $\mathcal{B}$ be a Boolean formula, initially set to $e(\varphi)$, i.e.,

$$\mathcal{B} := e(\varphi) .$$

As a second step, we pass $\mathcal{B}$ to a SAT solver.

Assume that the SAT solver returns the satisfying assignment

$$\alpha := \{e(x = y) \mapsto \text{TRUE},\ e(y = z) \mapsto \text{TRUE},\ e(x \neq z) \mapsto \text{TRUE},\\ e(x = z) \mapsto \text{FALSE}\} .$$

- Denote by $\hat{Th}(\alpha)$ the conjunction of the literals corresponding to this assignment.

$$\hat{Th}(\alpha) := \; x = y \wedge y = z \wedge x \neq z \wedge \neg(x = z)\ .$$

- Denote by $\hat{Th}(\alpha)$ the conjunction of the literals corresponding to this assignment.

$$\hat{Th}(\alpha) := \; x = y \land y = z \land x \neq z \land \neg(x = z) \; .$$

- The decision procedure DP_T now has to decide whether $\hat{Th}(\alpha)$ is satisfiable.

- Denote by $\hat{Th}(\alpha)$ the conjunction of the literals corresponding to this assignment.

$$\hat{Th}(\alpha) := \ x = y \land y = z \land x \neq z \land \neg(x = z)\ .$$

- The decision procedure DP_T now has to decide whether $\hat{Th}(\alpha)$ is satisfiable.

$\hat{Th}(\alpha)$ is not satisfiable, which means that the negation of this formula is a tautology.

Thus $\mathcal{B}$ is conjoined with $e(\neg \hat{Th}(\alpha))$, the Boolean encoding of this tautology:

$$e(\neg \hat{Th}(\alpha)) := \ (\neg e(x = y) \vee \neg e(y = z) \vee \neg e(x \neq z) \vee e(x = z)) \ .$$

Thus $\mathcal{B}$ is conjoined with $e(\neg \hat{T}h(\alpha))$, the Boolean encoding of this tautology:

$$e(\neg \hat{T}h(\alpha)) := \; (\neg e(x = y) \vee \neg e(y = z) \vee \neg e(x \neq z) \vee e(x = z)) \; .$$

- This clause contradicts the current assignment, and hence blocks it from being repeated.
- Such clauses are called **blocking clauses**.

Thus $\mathcal{B}$ is conjoined with $e(\neg \hat{Th}(\alpha))$, the Boolean encoding of this tautology:

$$e(\neg \hat{Th}(\alpha)) := \ (\neg e(x = y) \vee \neg e(y = z) \vee \neg e(x \neq z) \vee e(x = z)) \ .$$

- This clause contradicts the current assignment, and hence blocks it from being repeated.
- Such clauses are called **blocking clauses**.
- We denote by t the formula – also called the **lemma** – returned by DP_T (in this example $t := \neg \hat{Th}(\alpha)$).

After the blocking clause has been added, the SAT solver is invoked again and suggests another assignment, for example

$$\begin{aligned}\alpha' := \ & \{e(x = y) \mapsto \text{TRUE},\ e(y = z) \mapsto \text{TRUE},\ e(x = z) \mapsto \text{TRUE}, \\ & e(x \neq z) \mapsto \text{FALSE}\}\ .\end{aligned}$$

After the blocking clause has been added, the SAT solver is invoked again and suggests another assignment, for example

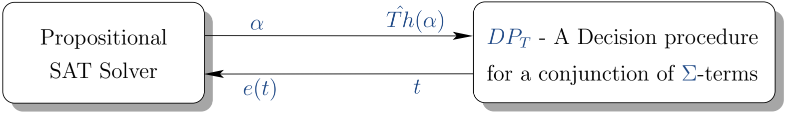
$$\begin{aligned}\alpha' := \ & \{e(x = y) \mapsto \text{TRUE},\ e(y = z) \mapsto \text{TRUE},\ e(x = z) \mapsto \text{TRUE},\\ & e(x \neq z) \mapsto \text{FALSE}\}\ .\end{aligned}$$

The corresponding Σ-formula

$$\hat{Th}(\alpha') := \ x = y \wedge y = z \wedge x = z \wedge \neg(x \neq z) \tag{3}$$

is satisfiable, which proves that φ, the original formula, is satisfiable.

Indeed, any assignment that satisfies $\hat{Th}(\alpha')$ also satisfies φ.



The information flow between the two components of the decision procedure.

There are many improvements to this basic procedure.

There are many improvements to this basic procedure.

One such improvement is:
"Invoke the decision procedure DP_T after partial assignments, rather than waiting for a full assignment."

There are many improvements to this basic procedure.

One such improvement is:
"Invoke the decision procedure DP_T after partial assignments, rather than waiting for a full assignment."

- A contradicting partial assignment leads to a more powerful lemma t, as it blocks all assignments that extend it.

There are many improvements to this basic procedure.

One such improvement is:
"Invoke the decision procedure DP_T after partial assignments, rather than waiting for a full assignment."

- A contradicting partial assignment leads to a more powerful lemma t, as it blocks all assignments that extend it.
- Theory propagation: When the partial assignment is not contradictory, it can be used to derive implications that are propagated back to the SAT solver.

Continuing the example above, consider the partial assignment

$$\alpha := \{e(x = y) \mapsto \text{TRUE},\ e(y = z) \mapsto \text{TRUE}\}\ , \tag{4}$$

Continuing the example above, consider the partial assignment

$$\alpha := \{e(x = y) \mapsto \text{TRUE},\ e(y = z) \mapsto \text{TRUE}\}\ , \tag{4}$$

and the corresponding formula that is transferred to DP_T,

$$\hat{Th}(\alpha) := \ x = y \wedge y = z\ . \tag{5}$$

Continuing the example above, consider the partial assignment

$$\alpha := \{e(x = y) \mapsto \text{TRUE},\ e(y = z) \mapsto \text{TRUE}\} , \tag{4}$$

and the corresponding formula that is transferred to DP_T,

$$\hat{Th}(\alpha) := x = y \wedge y = z . \tag{5}$$

DP_T concludes that $x = z$ is implied, and hence inform the SAT solver that $e(x = z) \mapsto \text{TRUE}$ and $e(x \neq z) \mapsto \text{FALSE}$ are implied by the current partial assignment α.

Plan

We will now formalize three versions of the algorithm:

1. Simple
2. Incremental
3. DPLL(T)

Notation

- $lit(\varphi)$ – the set of literals in a given NNF formula φ.

- $lit_i(\varphi)$ – the i-th distinct literal in φ
  (assuming some predefined order on the literals).

Notation

- $lit(\varphi)$ – the set of literals in a given NNF formula φ.

- $lit_i(\varphi)$ – the i-th distinct literal in φ
(assuming some predefined order on the literals).

- α – For a given encoding $e(\varphi)$, denotes an assignment (either full or partial), to the encoders in $e(\varphi)$.

Notation

- $Th(lit_i, \alpha)$ – For an encoder $e(lit_i)$ that is assigned a truth value by α, denotes the corresponding literal:

$$Th(lit_i, \alpha) \doteq \begin{cases} lit_i & \alpha(lit_i) = \text{TRUE} \\ \neg lit_i & \alpha(lit_i) = \text{FALSE}\,. \end{cases} \tag{6}$$

Notation

- $Th(lit_i, \alpha)$ – For an encoder $e(lit_i)$ that is assigned a truth value by α, denotes the corresponding literal:

$$Th(lit_i, \alpha) \doteq \begin{cases} lit_i & \alpha(lit_i) = \text{TRUE} \\ \neg lit_i & \alpha(lit_i) = \text{FALSE}\ . \end{cases} \tag{6}$$

- $Th(\alpha) \doteq \{Th(lit_i, \alpha) \mid e(lit_i) \text{ is assigned by } \alpha\}$

Notation

- $Th(lit_i, \alpha)$ – For an encoder $e(lit_i)$ that is assigned a truth value by α, denotes the corresponding literal:

$$Th(lit_i, \alpha) \doteq \begin{cases} lit_i & \alpha(lit_i) = \text{TRUE} \\ \neg lit_i & \alpha(lit_i) = \text{FALSE} \,. \end{cases} \tag{6}$$

- $Th(\alpha) \doteq \{Th(lit_i, \alpha) \mid e(lit_i) \text{ is assigned by } \alpha\}$
- $\hat{T}h(\alpha)$ – a conjunction over the elements in $Th(\alpha)$.

Example Use of the Notation

Let

$$lit_1 = (x = y),\ lit_2 = (y = z),\ lit_3 = (z = w)\ , \tag{7}$$

Example Use of the Notation

Let

$$lit_1 = (x = y),\ lit_2 = (y = z),\ lit_3 = (z = w)\ , \tag{7}$$

and let α be a partial assignment such that

$$\alpha :=\ \{e(lit_1) \mapsto \text{FALSE},\ e(lit_2) \mapsto \text{TRUE}\}\ .$$

Example Use of the Notation

Let

$$lit_1 = (x = y),\ lit_2 = (y = z),\ lit_3 = (z = w)\ , \tag{7}$$

and let α be a partial assignment such that

$$\alpha := \ \{e(lit_1) \mapsto \text{FALSE},\ e(lit_2) \mapsto \text{TRUE}\}\ .$$

Then

$$Th(lit_1, \alpha) := \ \neg(x = y),\ Th(lit_2, \alpha) := \ (y = z)\ ,$$

Let

$$lit_1 = (x = y),\ lit_2 = (y = z),\ lit_3 = (z = w)\ , \tag{7}$$

and let α be a partial assignment such that

$$\alpha := \ \{e(lit_1) \mapsto \text{FALSE},\ e(lit_2) \mapsto \text{TRUE}\}\ .$$

Then

$$Th(lit_1, \alpha) := \ \neg(x = y),\ Th(lit_2, \alpha) := \ (y = z)\ ,$$

and

$$Th(\alpha) := \ \{\neg(x = y), (y = z)\}\ .$$

Example Use of the Notation

Let

$$lit_1 = (x = y),\ lit_2 = (y = z),\ lit_3 = (z = w)\ , \tag{7}$$

and let α be a partial assignment such that

$$\alpha := \ \{e(lit_1) \mapsto \text{FALSE},\ e(lit_2) \mapsto \text{TRUE}\}\ .$$

Then

$$Th(lit_1, \alpha) := \ \neg(x = y),\ Th(lit_2, \alpha) := \ (y = z)\ ,$$

and

$$Th(\alpha) := \ \{\neg(x = y), (y = z)\}\ .$$

Conjoining these terms gives us

$$\hat{Th}(\alpha) := \ \neg(x = y) \wedge (y = z)\ .$$

Notation

- T – a Σ-theory.

Notation

- T – a Σ-theory.
- DP_T a decision procedure for the conjunctive fragment of T.

Notation

- T – a Σ-theory.
- DP_T a decision procedure for the conjunctive fragment of T.
- Let DEDUCTION be a procedure based on DP_T, which receives a conjunction of Σ-literals as input, and
  - decides whether it is satisfiable, and,
  - if the answer is negative, returns constraints over these literals.

function LAZY-BASIC(φ)
    $\mathcal{B} := e(\varphi)$;
    while (TRUE) **do**
        $\langle \alpha, res \rangle :=$ SAT-SOLVER($\mathcal{B}$);
        if $res =$ "Unsatisfiable" **then return** "Unsatisfiable";
        else
            $\langle t, res \rangle :=$ DEDUCTION($\hat{Th}(\alpha)$);
            if $res =$ "Satisfiable" **then return** "Satisfiable";
            $\mathcal{B} := \mathcal{B} \wedge e(t)$;

Consider the following three requirements from the clause t that is returned by DEDUCTION:

1. The formula t is T-valid, i.e., t is a tautology in T. For example, if T is the theory of equality, then $x = y \wedge y = z \longrightarrow x = z$ is T-valid.

Consider the following three requirements from the clause t that is returned by DEDUCTION:

1. The formula t is T-valid, i.e., t is a tautology in T. For example, if T is the theory of equality, then $x = y \wedge y = z \longrightarrow x = z$ is T-valid.
2. The atoms in t are restricted to those appearing in φ.

Consider the following three requirements from the clause t that is returned by Deduction:

1. The formula t is T-valid, i.e., t is a tautology in T. For example, if T is the theory of equality, then $x = y \wedge y = z \longrightarrow x = z$ is T-valid.
2. The atoms in t are restricted to those appearing in φ.
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause.

Consider the following three requirements from the clause t that is returned by DEDUCTION:

1. The formula t is T-valid, i.e., t is a tautology in T. For example, if T is the theory of equality, then $x = y \wedge y = z \longrightarrow x = z$ is T-valid.
2. The atoms in t are restricted to those appearing in φ.
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause.

The first requirement is sufficient for guaranteeing soundness.

Consider the following three requirements from the clause t that is returned by DEDUCTION:

1. The formula t is T-valid, i.e., t is a tautology in T. For example, if T is the theory of equality, then $x = y \wedge y = z \longrightarrow x = z$ is T-valid.
2. The atoms in t are restricted to those appearing in φ.
3. The encoding of t contradicts α, i.e., $e(t)$ is a blocking clause.

The first requirement is sufficient for guaranteeing soundness.

The second and third requirements are sufficient for guaranteeing termination.

Two of the requirements can be weakened:

Two of the requirements can be weakened:

- Requirement 1: the clause t can be any formula that is implied by φ, and not just a T-valid formula.

Two of the requirements can be weakened:

- Requirement 1: the clause t can be any formula that is implied by φ, and not just a T-valid formula.
- Requirement 2: the clause t may refer to atoms that do not appear in φ, as long as the number of such new atoms is finite.

Two of the requirements can be weakened:

- Requirement 1: the clause t can be any formula that is implied by φ, and not just a T-valid formula.
- Requirement 2: the clause t may refer to atoms that do not appear in φ, as long as the number of such new atoms is finite.
  - For example, in equality logic, we may allow t to refer to all atoms of the form $x_i = x_j$ where x_i, x_j are variables in $var(\varphi)$, even if only some of these equality predicates appear in φ.

- Let $\mathcal{B}^i$ be the formula $\mathcal{B}$ in the i-th iteration of the loop.

- Let $\mathcal{B}^i$ be the formula $\mathcal{B}$ in the i-th iteration of the loop.
- The constraint $\mathcal{B}^{i+1}$ is strictly stronger than $\mathcal{B}^i$ for all $i \geq 1$, because clauses are added but not removed between iterations.

2. We can do better...

- Let $\mathcal{B}^i$ be the formula $\mathcal{B}$ in the i-th iteration of the loop.

- The constraint $\mathcal{B}^{i+1}$ is strictly stronger than $\mathcal{B}^i$ for all $i \geq 1$, because clauses are added but not removed between iterations.

- As a result, any conflict clause that is learned while solving $\mathcal{B}^i$ can be reused when solving $\mathcal{B}^j$ for $i < j$.

2. We can do better...

- Let $\mathcal{B}^i$ be the formula $\mathcal{B}$ in the i-th iteration of the loop.
- The constraint $\mathcal{B}^{i+1}$ is strictly stronger than $\mathcal{B}^i$ for all $i \geq 1$, because clauses are added but not removed between iterations.
- As a result, any conflict clause that is learned while solving $\mathcal{B}^i$ can be reused when solving $\mathcal{B}^j$ for $i < j$.
- This is a special case of incremental satisfiability.

- Hence, invoking an incremental SAT solver in line 4 can increase the efficiency of the algorithm.

- Hence, invoking an incremental SAT solver in line 4 can increase the efficiency of the algorithm.

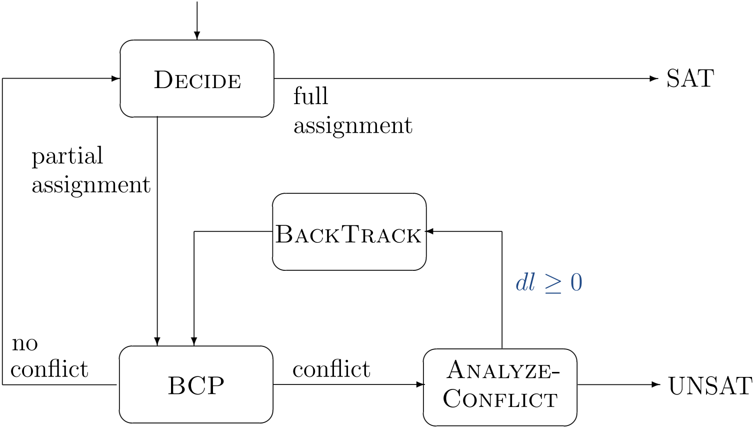
- A better option is to integrate DEDUCTION into the DPLL-SAT algorithm, as shown in the following algorithm.

- Hence, invoking an incremental SAT solver in line 4 can increase the efficiency of the algorithm.

- A better option is to integrate DEDUCTION into the DPLL-SAT algorithm, as shown in the following algorithm.

- This algorithm uses a procedure ADDCLAUSES, which adds new clauses to the current set of clauses at run time.

- Hence, invoking an incremental SAT solver in line 4 can increase the efficiency of the algorithm.
- A better option is to integrate DEDUCTION into the DPLL-SAT algorithm, as shown in the following algorithm.
- This algorithm uses a procedure ADDCLAUSES, which adds new clauses to the current set of clauses at run time.
- Before seeing this algorithm let us first recall DPLL...


Decide
SAT
full assignment
partial assignment
BackTrack
$dl \geq 0$
no conflict
BCP
conflict
Analyze-Conflict
UNSAT

2. Pseudo-code for DPLL

```
function DPLL
    if BCP() = "conflict" then return "Unsatisfiable";
    while (TRUE) do
        if ¬DECIDE() then return "Satisfiable";
        else
            while (BCP() = "conflict") do
                backtrack-level := ANALYZE-CONFLICT();
                if backtrack-level < 0 then return "Unsatisfiable";
                else BackTrack(backtrack-level);
```

function LAZY-DPLL
    ADDCLAUSES($cnf(e(\varphi))$);
    if BCP() = “conflict” **then return** “Unsatisfiable”;
    while (TRUE) **do**
        if $\neg$DECIDE() **then**   ▷ Full assignment
            $\langle t, res\rangle$:=DEDUCTION($\hat{Th}(\alpha)$);
            if res=“Satisfiable” **then return** “Satisfiable”;
            ADDCLAUSES($e(t)$);
            while (BCP() = “conflict”) **do**
                $backtrack\text{-}level$:= ANALYZE-CONFLICT();
                if $backtrack\text{-}level < 0$ **then return** “Unsatisfiable”;
                else BackTrack($backtrack\text{-}level$);
        else
            while (BCP() = “conflict”) **do**
                $backtrack\text{-}level$:= ANALYZE-CONFLICT();
                if $backtrack\text{-}level < 0$ **then return** “Unsatisfiable”;
                else BackTrack($backtrack\text{-}level$);

- Consider a formula φ that contains an integer variable x_1 and, among others, the literals $x_1 \geq 10$ and $x_1 < 0$.

- Consider a formula φ that contains an integer variable x_1 and, among others, the literals $x_1 \geq 10$ and $x_1 < 0$.

- Assume that the DECIDE procedure assigns $e(x_1 \geq 10) \mapsto \text{TRUE}$ and $e(x_1 < 0) \mapsto \text{TRUE}$.

- Consider a formula φ that contains an integer variable x_1 and, among others, the literals $x_1 \geq 10$ and $x_1 < 0$.
- Assume that the DECIDE procedure assigns $e(x_1 \geq 10) \mapsto$ TRUE and $e(x_1 < 0) \mapsto$ TRUE.
- Inevitably, any call to DEDUCTION results in a contradiction between these two facts, independently of any other decisions that are made.

- However, the algorithms we saw so far do not call DEDUCTION until a full satisfying assignment is found.
  - Thus, the time taken to complete the assignment is wasted.

- However, the algorithms we saw so far do not call DEDUCTION until a full satisfying assignment is found.
  - Thus, the time taken to complete the assignment is wasted.

- Further, the refutation of this full assignment may be due to other reasons (i.e., a proof that a different subset of the assignment is contradictory).
  - Hence, additional assignments that include the same wrong assignment to $e(x_1 \geq 10)$ and $e(x_1 < 0)$ are not ruled out.

Early call to DEDUCTION can serve two purposes:

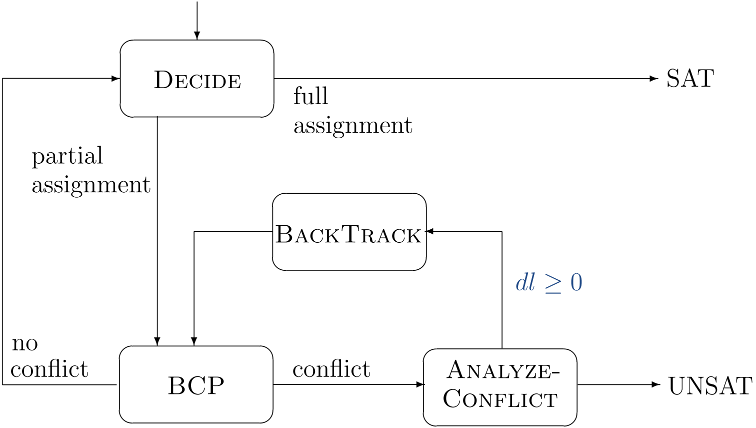
Early call to DEDUCTION can serve two purposes:

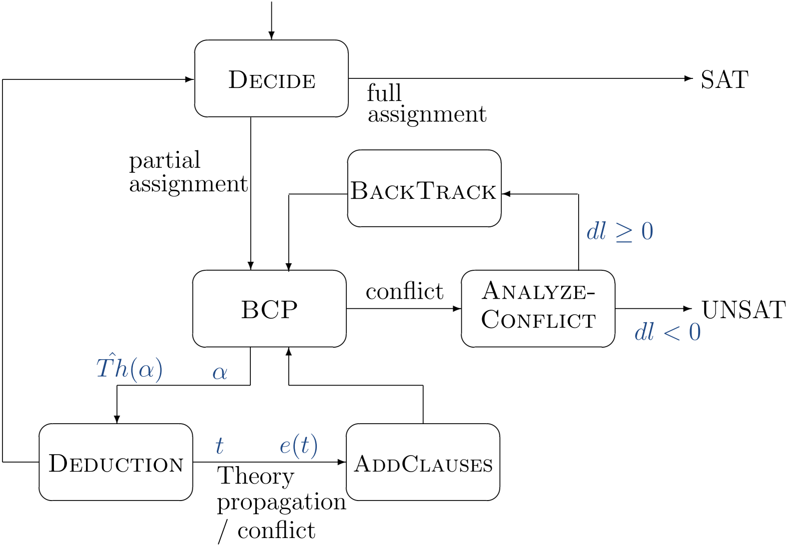
1. Contradictory partial assignments are ruled out early.

Early call to DEDUCTION can serve two purposes:

1. Contradictory partial assignments are ruled out early.
2. Allows theory propagation.
   - Continuing our example, once $e(x_1 \geq 10)$ has been assigned TRUE, we can infer that $e(x_1 < 0)$ must be FALSE and avoid the conflict altogether.

This brings us to the next version of the algorithm, called $\mathrm{DPLL}(T)$.


DECIDE
SAT
full
assignment
partial
assignment
BACKTRACK
$dl \geq 0$
no
conflict
BCP
conflict
ANALYZE-
CONFLICT
UNSAT



DECIDE
SAT
full assignment
partial assignment
BACKTRACK
$dl \geq 0$
BCP
conflict
ANALYZE-CONFLICT
UNSAT
$dl < 0$
$\hat{Th}(\alpha)$
α
DEDUCTION
t
$e(t)$
ADDCLAUSES
Theory propagation / conflict

function DPLL(T)
    ADDCLAUSES($cnf(e(\varphi))$);
    if BCP() = "conflict" **then return** "Unsatisfiable";
    while (TRUE) **do**
        if $\neg$DECIDE() **then return** "Satisfiable"; ▷ Full assignment
        repeat
            while (BCP() = "conflict") **do**
                $backtrack\text{-}level$:= ANALYZE-CONFLICT();
                if $backtrack\text{-}level < 0$ **then return** "Unsatisfiable";
                else BackTrack($backtrack\text{-}level$);
            $\langle t, res \rangle$:=DEDUCTION($\hat{Th}(\alpha)$);
            ADDCLAUSES($e(t)$);
        until $t \equiv$ TRUE

If $\hat{Th}(\alpha)$ is satisfiable, we require t to fulfill one of the following two conditions in order to guarantee termination:

If $\hat{Th}(\alpha)$ is satisfiable, we require t to fulfill one of the following two conditions in order to guarantee termination:

1. The clause $e(t)$ is an asserting clause under α. This implies that the addition of $e(t)$ to $\mathcal{B}$ and a call to BCP leads to an assignment to the encoder of some literal.

If $\hat{Th}(\alpha)$ is satisfiable, we require t to fulfill one of the following two conditions in order to guarantee termination:

1. The clause $e(t)$ is an asserting clause under α. This implies that the addition of $e(t)$ to $\mathcal{B}$ and a call to BCP leads to an assignment to the encoder of some literal.
2. When $\mathrm{D{\small EDUCTION}}$ cannot find an asserting clause t as defined above, t and $e(t)$ are equivalent to TRUE.

The second case occurs, for example, when all the Boolean variables are already assigned, and thus the formula is found to be satisfiable.

3. Theory Propagation

Various ways to perform theory propagation:

- After every decision / after every assignment

Various ways to perform theory propagation:

- After every decision / after every assignment
- Partial / Exhaustive theory propagation – propagate all that is implied by the current partial assignment.

Various ways to perform theory propagation:

- After every decision / after every assignment
- Partial / Exhaustive theory propagation – propagate all that is implied by the current partial assignment.
- Refer only to existing predicates / add auxiliary ones.

Exhaustive theory propagation after each assignment: what does this mean ?

Various ways to perform theory propagation:

- After every decision / after every assignment
- Partial / Exhaustive theory propagation – propagate all that is implied by the current partial assignment.
- Refer only to existing predicates / add auxiliary ones.

Exhaustive theory propagation after each assignment: what does this mean ?

That's right, no possible conflicts on the theory side.

How to check whether a predicate p is implied by $\hat{Th}(\alpha)$?

- Plunging – is $\hat{Th}(\alpha) \wedge \neg p$ satisfiable ?

How to check whether a predicate p is implied by $\hat{Th}(\alpha)$?

- Plunging – is $\hat{Th}(\alpha) \wedge \neg p$ satisfiable ?

- Theory-specific propagation. For example, in equality logic build the equality graph corresponding to $Th(\alpha)$. Infer equalities/disequalities from the graph.

How to check whether a predicate p is implied by $\hat{Th}(\alpha)$?

- Plunging – is $\hat{Th}(\alpha) \wedge \neg p$ satisfiable ?
- Theory-specific propagation. For example, in equality logic build the equality graph corresponding to $Th(\alpha)$. Infer equalities/disequalities from the graph.
- Note that theory propagation matters for efficiency, not correctness.

How to check whether a predicate p is implied by $\hat{Th}(\alpha)$?

- Plunging – is $\hat{Th}(\alpha) \wedge \neg p$ satisfiable ?

- Theory-specific propagation. For example, in equality logic build the equality graph corresponding to $Th(\alpha)$. Infer equalities/disequalities from the graph.

- Note that theory propagation matters for efficiency, not correctness.

- How much propagation is cost-effective is a subject for research, and depends on T.

- Normally theory propagation is done by transferring clauses to the the DPLL solver.

- It turns out to be inefficient – few (less than 0.5%) are actually used.

- Instead – add implied literals directly to the implication stack.
  - This causes a problem in ANALYZE-CONFLICT() – can you see what problem ?

- The problem: $\textsc{Analyze-conflict}()$ requires an antecedent clause for each implication, in order to compute the conflict clause and backtrack level.

- The problem: ANALYZE-CONFLICT() requires an antecedent clause for each implication, in order to compute the conflict clause and backtrack level.

- Theory propagation without clauses breaks this mechanism – there are implications without antecedents.

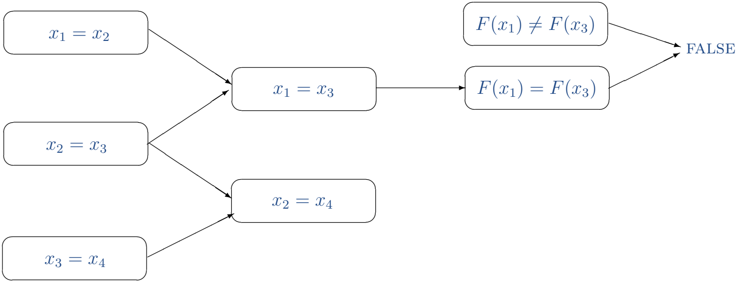
- The problem: ANALYZE-CONFLICT() requires an antecedent clause for each implication, in order to compute the conflict clause and backtrack level.
- Theory propagation without clauses breaks this mechanism – there are implications without antecedents.
- Solution – DP_T should be able to explain an implication post-mortem, in the form of a clause.

- When $\hat{Th}(\alpha)$ is unsatisfiable, the lemma (clause returned by DEDUCTION) rules out α.

- When $\hat{Th}(\alpha)$ is unsatisfiable, the lemma (clause returned by DEDUCTION) rules out α.

- Ideally, it should be generalized as much as possible.

- When $\hat{Th}(\alpha)$ is unsatisfiable, the lemma (clause returned by DEDUCTION) rules out α.
- Ideally, it should be generalized as much as possible.
- Solution: analyze the reason for unsatisfiability.
  Build lemma accordingly.


$x_1 = x_2$
$x_2 = x_3$
$x_3 = x_4$
$x_1 = x_3$
$x_2 = x_4$
$F(x_1) \neq F(x_3)$
$F(x_1) = F(x_3)$
FALSE