Decision procedures and the DPLL procedure for satisfiability

Luciano Serafini – Maurizio Lenzerini

FBK-IRST, Trento, Italy – Sapienza Università di Roma

A.Y. 2021/22

Decision procedures - automated reasoning

Automated reasoning is the discipline that aims at solving basic decision problems in logic, the main of which are:

Four types of decision problems

- **Model Checking($\mathcal{I}, \phi$)**: $\mathcal{I} \overset{?}{\models} \phi$ Does the intepretation $\mathcal{I}$ satisfy ϕ, or equivalently, is $\mathcal{I}$ a model of ϕ?
- **Satisfiability(ϕ)**: $\overset{?}{\exists} \mathcal{I} \,.\, \mathcal{I} \models \phi$ Is there a model of ϕ?
- **Validity(ϕ)**: $\overset{?}{\models} \phi$ Is every interpretation for ϕ a model of ϕ?
- **Logical implication(Γ, ϕ)**: $\Gamma \overset{?}{\models} \phi$ Is every model of the set of formulas Γ a model of ϕ as well?
- **Logical inference(Γ, ϕ)**: $\Gamma \overset{?}{\vdash}_{\Sigma} \phi$ Is there a proof of ϕ in Σ from Γ ?

Model Checking

Model checking decision procedure

A model checking decision procedure, MCDP is an algorithm that checks if a formula ϕ is satisfied by an interpretation $\mathcal{I}$. Namely

$$\begin{aligned}\mathrm{MCDP}(\phi,\mathcal{I}) = \textit{true} &\quad \text{if and only if} \quad \mathcal{I} \models \phi \\ \mathrm{MCDP}(\phi,\mathcal{I}) = \textit{false} &\quad \text{if and only if} \quad \mathcal{I} \not\models \phi\end{aligned}$$

Observation

The procedure of model checking returns for all inputs either true or false since for every intepretation $\mathcal{I}$ and for every formula ϕ, we have that either $\mathcal{I} \models \phi$ or $\mathcal{I} \not\models \phi$.

Observation

We have already seen a naive algorithm for model checking in propositional logic.

Satisfiability

Satisfiability decision procedure

A satisfiability decision procedure SDP is an algorithm that takes in input a formula ϕ and checks if ϕ is satisfiable. Namely

$$
\begin{aligned}
\mathrm{SDP}(\phi) = \textit{Satisfiable} &\quad \text{if and only if} \quad \mathcal{I} \models \phi \text{ for some } \mathcal{I} \\
\mathrm{SDP}(\phi) = \textit{Unsatisfiable} &\quad \text{if and only if} \quad \mathcal{I} \not\models \phi \text{ for all } \mathcal{I}
\end{aligned}
$$

When $\mathrm{SDP}(\phi) =$ *satisfiable*, SDP can in addition return a model $\mathcal{I}$ (or, one of the models) of the formula ϕ.

Validity

Validity decision procedure

A decision procedure for Validity VDC, is an algorithm that checks whether a formula is valid. VDP can be based on a satisfiability decision procedure by exploiting the equivalence

$$\phi \text{ is valid if and only if } \neg\phi \text{ is not satisfiable}$$

$$\begin{array}{rcl} \mathit{VDP}(\phi) = \mathit{true} & \text{if and only if} & \mathrm{SDP}(\neg\phi) = \mathit{Unsatisfiable} \\ \mathit{VDP}(\phi) = \mathit{false} & \text{if and only if} & \mathrm{SDP}(\neg\phi) = \mathit{Satisfiable} \end{array}$$

When $\mathrm{SDP}(\neg\phi)$ returns *Satisfiable*, it may in addition returns an interpretation $\mathcal{I}$, and $\mathcal{I}$ is called a counter-model for ϕ, in the sense that is an intepretation that falsifies ϕ.

Logical implication

Logical implication decision procedure

A decision procedure for logical consequence (or implication) LCDP is an algorithm that checks whether a formula ϕ is a logical consequence of a finite set of formulas $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. LCDP can be implemented on the basis of satisfiability decision procedure by exploiting the property

$$\Gamma \models \phi \text{ if and only if } \Gamma \cup \{\neg\phi\} \text{ is unsatisfiable}$$

$$\begin{array}{rcl} \mathit{LCDP}(\Gamma, \phi) = \mathit{true} & \text{if and only if} & \mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\phi) = \mathit{Unatisfiable} \\ \mathit{LCDP}(\Gamma, \phi) = \mathit{false} & \text{if and only if} & \mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\phi) = \mathit{Satisfiable} \end{array}$$

When $\mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\phi)$ returns *Satisfiable*, it may in addition returns an interpretation $\mathcal{I}$, and $\mathcal{I}$ is a model for Γ and a counter-model for ϕ, i.e., a model of Γ that falsifies ϕ.

Proof of the previous property

Theorem

$\Gamma \models \phi$ if and only if $\Gamma \cup \{\neg\phi\}$ is unsatisfiable.

Proof.

$\Rightarrow$ If $\Gamma \models \phi$, then every model of Γ satisfies ϕ, and therefore we cannot find a model of Γ where $\neg\phi$ is true. It follows that $\Gamma \cup \{\neg\phi\}$ is unsatisfiable.

$\Leftarrow$ If $\Gamma \cup \{\neg\phi\}$ is unsatisfiable, then either (a) Γ itself is unsatisfiable, or (b) Γ has at least one model, but $\neg\phi$ is false in every model of Γ. If (a) is the case, then Γ logically implies everything, and therefore $\Gamma \models \phi$. If (b) is the case, the set of models of Γ is nonempty, and ϕ is true in all such model. So, we have shown that $\Gamma \models \phi$ holds in all cases.

□

Davis-Putnam procedure for satisfiability

- In 1960, Davis and Putnam published a SAT algorithm.

  Davis, Putnam. A Computing Procedure for Quantification Theory. Journal of the ACM, 7(3):201-215, 1960.

- In 1962, Davis, Logemann, and Loveland improved the DP algorithm.

  Davis, Logemann, Loveland. A Machine Program for Theorem-Proving. Communications of the ACM, 5(7): 394–397, 1962.

- Basic framework for most current SAT solvers.
- It assumes the formula to be in conjunctive normal form.

Conjunctive Normal form

Definition

- A literal is either a propositional variable or the negation of a propositional variable.

$$p, \quad \neg q$$

- A clause is a disjunction of literals.

$$(a \vee \neg b \vee c)$$

- A formula is in conjunctive normal form, if it is a conjunction of clauses.

$$(p \vee \neg q \vee r) \wedge (q \vee r) \wedge (\neg p \vee \neg q) \wedge r$$

Conjunctive Normal form

Conjunctive Normal form

A formula in conjunctive normal form has the following shape:

$$(l_{11} \vee \cdots \vee l_{1n_1}) \wedge \ldots \wedge (l_{m1} \vee \cdots \vee l_{mn_m})$$

equivalently written as

$$\bigwedge_{i=1}^{m} \left(\bigvee_{j=1}^{n_j} l_{ij} \right)$$

where l_{ij} is the j-th literal of the i-th clause composing ϕ

Example

$(p \vee \neg q) \wedge (r \vee p \vee \neg r) \wedge (p \vee p)$, $p \vee q$,

$p \wedge q$, $p \wedge \neg q \wedge (r \vee s)$

Equi-satisfiable formulas

Two formulas ϕ and ϕ' are equi-satisfiable iff:

ϕ is satisfiable if and only if ϕ' is satisfiable

- If two formulas are equi-satisfiable, are they equivalent? Not necessarily!
    - Example: any satisfiable formula (e.g., p) is equi-satisfiable with $\top$, but clearly, $p \equiv \top$ is not valid!
    - Example: Introducing names leads to equi-satisfiable formulas. E.g. the formula $a \wedge b$ is equi-satisfiable of the formula $(n \equiv a \wedge b) \wedge n$, but it is not true that

$$(a \wedge b) \equiv (n \wedge (a \wedge b \equiv n))$$

- Equi-satisfiability is a weaker notion than equivalence. It is useful if all we want to do is determine satisfiability.

Tseitin's transformation

Tseitin's transformation ...

... converts any propositional formula ϕ into an equi-satisfiable formula ϕ' in CNF with only a linear increase in size.

Key ideas:

- give a name to every subformula (except for literals), and use this name as a fresh propositional letter "representing" the subformula
- for every formula ψ of the form $p_1 \equiv (p_2 \circ p_3)$ (where $\circ$ is a connective), there is a CNF formula *CNF*(ψ) that is equivalent to ψ
- transform the original formula ψ into a conjunction of many *CNF*(ψ_i) obtained from ψ, one for each subformula of ψ

Tseitin's transformation

If ψ is any formula, then for every subformula ϕ of ψ, we define its name n_ϕ as follows:

- n_ϕ is simply ϕ if ϕ is a literal;
- n_ϕ is a new propositional letter if ϕ is not a literal.

The Tseitin's transformation $T(\psi)$ of ψ is the conjunction of

- n_ψ
- *CNF*$(q \equiv \neg n_\phi)$ for every non-literal subformula of the form $\neg\phi$ having name q
- *CNF*$(q \equiv (n_{\phi_1} \circ n_{\phi_2}))$ for every subformula of the form $\phi_1 \circ \phi_2$ having name q, where $\circ \in \{\vee, \wedge, \rightarrow, \equiv\}$.

Tseitin's transformation

We still need to understand what *CNF* does. We proceed to analyze all cases.

- $CNF(q \equiv \neg p) = (\neg q \vee \neg p) \wedge (p \vee q)$
- $CNF(q \equiv (p \wedge r)) = (q \vee \neg p \vee \neg r) \wedge (\neg q \vee p) \wedge (\neg q \vee r)$
- $CNF(q \equiv (p \vee r)) = (\neg q \vee p \vee r) \wedge (q \vee \neg p) \wedge (q \vee \neg r)$
- $CNF(q \equiv (p \rightarrow r)) = (\neg q \vee \neg p \vee r) \wedge (p \vee q) \wedge (\neg r \vee q)$
- $CNF(q \equiv (p \equiv r)) =$
  $(q \vee p \vee r) \wedge (q \vee \neg p \vee \neg r) \wedge (\neg q \vee p \neg r) \wedge (\neg q \vee \neg p \vee r)$

Tseitin's transformation

Theorem

For every formula ψ, $T(\psi)$ is satisfiable if and only if ψ is. Moreover, $T(\psi)$ is 3-CNF formula whose size is linear with respect to the size of ψ.

Tseitin's transformation - Example

Let ψ be the formula $(p \vee q) \rightarrow (p \wedge \neg r)$

1. For each non-literal subformula, introduce new letters: x_1 for ψ, x_2 for $p \vee q$, x_3 for $p \wedge \neg r$
2. Stipulate equivalences and convert them to CNF:

$$\begin{aligned}
CNF(x_1 \equiv (x_2 \rightarrow x_3)) \quad &\Rightarrow \quad \phi_1 : (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_1) \wedge \\
&\qquad (\neg x_3 \vee x_1) \\
CNF(x_2 \equiv (p \vee q)) \quad &\Rightarrow \quad \phi_2 : (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge \\
&\qquad (\neg q \vee x_2) \\
CNF(x_3 \equiv (p \wedge \neg r)) \quad &\Rightarrow \quad \phi_3 : (\neg x_3 \vee p) \wedge (\neg x_3 \vee \neg r) \wedge \\
&\qquad (\neg p \vee r \vee x_3)
\end{aligned}$$

3. The formula $T(\psi)$ is:

$$x_1 \wedge \phi_1 \wedge \phi_2 \wedge \phi_3$$

Properties of $\wedge$ and $\vee$

Commutativity of $\wedge$:	$\phi \wedge \psi$	$\equiv$	$\psi \wedge \phi$
Commutativity of $\vee$:	$\phi \vee \psi$	$\equiv$	$\psi \vee \phi$
Absorption of $\wedge$:	$\phi \wedge \phi$	$\equiv$	ϕ
Absorption of $\vee$:	$\phi \vee \phi$	$\equiv$	ϕ

Properties of clauses

Order of literals does not matter

If a clause C is obtained by reordering the literals of a clause C' then C and C' are equivalent. E.g., $(p \vee q \vee r \vee \neg r) \equiv (\neg r \vee q \vee p \vee r)$

Multiple literals can be merged

If a clause contains more than one occurrence of the same literal, then it is equivalent to the clause obtained by deleting all but one such occurrences. E.g., $(p \vee q \vee r \vee q \vee \neg r) \equiv (p \vee q \vee r \vee \neg r)$

Clauses as set of literals

It follows from these properties that we can represent a clause as a set of literals, by living disjunction implicit and by ignoring replication and order of literals

$(p \vee q \vee r \vee \neg r)$ is represented by the set $\{p, q, r, \neg r\}$

Properties of formulas in CNF

Order of clauses does not matter

If a CNF formula ϕ is obtained by reordering the clauses of a CNF formula ϕ' then ϕ and ϕ' are equivalent. E.g.,

$$(a \vee b) \wedge (c \vee \neg b) \wedge (\neg b) \equiv (c \vee \neg b) \wedge (\neg b) \wedge (a \vee b)$$

Multiple clauses can be merged

If a CNF formula contains more than one occurrence of the same clause, then it is equivalent to the formula obtained by deleting all but one such occurrences. E.g.,

$$(a \vee b) \wedge (c \vee \neg b) \wedge (a \vee b) \equiv (a \vee b) \wedge (c \vee \neg b)$$

A CNF formula can be seen as a set of clauses

It follows from the properties of clauses and CNF formulas that we can represent a CNF formula as a set of sets of literals. E.g.,

$(\neg b) \wedge (b \vee a \vee b) \wedge (c \vee \neg b) \wedge (\neg b)$

is represented by $\{\{a, b\}, \{c, \neg b\}, \{\neg b\}\}$

Satisfiability of a set of clauses

- Let $\psi = \{C_1, \ldots, C_n\}$
  - $\mathcal{I} \models \psi$ if and only if $\mathcal{I} \models C_i$ for all $i = 1..n$;
  - $\mathcal{I} \models C_i$ if and only if for some $l \in C$, $\mathcal{I} \models l$
- To check if an intepretation $\mathcal{I}$ satisfies ψ we do not necessarily need to know the truth values that $\mathcal{I}$ assigns to all the literals appearing in ψ.
- For instance, if $\mathcal{I}(p) = \mathit{true}$ and $\mathcal{I}(q) = \mathit{false}$, we can say that $\mathcal{I} \models \{\{p, q, \neg r\}, \{\neg q, s, q\}\}$, without considering the evaluations of $\mathcal{I}(r)$ and $\mathcal{I}(s)$.

Partial interpretation

A partial intepretation is a partial function that associates to some propositional variables of the alphabet P a truth value (either true or false) and can be undefined for the others.

Partial interpretations

- Partial interpretations can be used to construct models for a set of clauses $\psi = \{C_1, \ldots, C_n\}$ incrementally
- Under a partial interpretation $\mathcal{I}$, literals and clauses can be true, false or undefined; a clause C
  - is true under $\mathcal{I}$ if at least one of its literals is true;
  - is false (or "conflicting") under $\mathcal{I}$ if all its literals are false
  - is undefined (or "unresolved") under $\mathcal{I}$, otherwise.
- The algorithms that exploit partial intepretations usually start with an empty intepretation (where the truth values of all propositional letters are undefined) and tries to extend it step by step to the various variables occurring in $\{C_1, \ldots, C_n\}$.

The procedure

The DPLL procedure uses

- a subroutine called UnitPropagation
- a subroutine called Simplification, used in turn by UnitPropagation

Simplification

- If λ is a literal of the form p, then $\neg\lambda$ denotes $\neg p$.
- If λ is a literal of the form $\neg p$, then $\neg\lambda$ denotes p.

Simplification of a formula by a literal

For any CNF formula ϕ and a literal λ, $\phi|_\lambda$ stands for the formula obtained from ϕ by

- removing all clauses containing the literal λ, and
- removing the literals $\neg\lambda$ in all remaining clauses

Example

For instance,
$\{\{p, q, \neg r\}, \{\neg p, \neg r\}\}|_{\neg p} = \{\{q, \neg r\}\}$

Unit propagation

Unit clause

A unit clause is a clause containing a single literal.

If the CNF formula ϕ contains a unit clause $\{\lambda\}$, then to satisfy ϕ the literal λ must be evaluated to True. As a consequence ϕ can be simplified using the following procedure $\textsc{UnitPropagation}$, that is invoked on ϕ and a partial intepretation $\mathcal{I}$, and in turn returns a CNF formula and a partial interpretation.

Unit propagation

```
$\textsc{UnitPropagation}(\phi,\mathcal{I})$
  while $\phi$ contains a unit clause $\{\lambda\}$
    $\phi := \phi|_\lambda$;
    if $\lambda = p$, then $\mathcal{I}(p) := true$;
    if $\lambda = \neg p$, then $\mathcal{I}(p) := false$
  end;
  return $(\phi,\mathcal{I})$
```

The DPLL procedure

Example

$$\textsc{UnitPropagation}(\{\{p\},\{\neg p,\neg q\},\{\neg q,r\}\},\emptyset)$$

$$\begin{array}{lr}
\{\{p\},\{\neg p,\neg q\},\{\neg q,r\}\} & \\
\{\{p\},\{\neg p,\neg q\},\{\neg q,r\}\}|_p=\{\{\neg q\},\{\neg q,r\}\} & \mathcal{I}(p)=\mathit{true} \\
\{\{\neg q\},\{\neg q,r\}\} & \\
\{\{\neg q\},\{\neg q,r\}\}|_{\neg q}=\{\} & \mathcal{I}(q)=\mathit{false}
\end{array}$$

the procedure returns $(\{\},\mathcal{I})$

The DPLL procedure

Remark

In some cases, unit propagation applied to ϕ is enough to decide satisfiability of ϕ, for example when it terminates with one of the following two results:

- $(\{\}, \mathcal{I})$ (as in the example above), in which case the initial formula is satisfiable, and a satisfying interpretation can be easily extracted from $\mathcal{I}$;
- $(\phi, \mathcal{I})$, where $\{\} \in \phi$, in which case the initial formula is unatisfiable.

There are cases in which $\textsc{UnitPropagation}$ does terminate with none of the above case, i.e., when there is no unit clauses to consider, the CNF is nonempty, and it does not contain empty clauses. e.g.,

$$\{\{p, q\}, \{\neg q, r\}\}$$

The DPLL procedure

The Davis-Putnam-Logemann-Loveland procedure

To check the CNF formula ϕ for satisfiability, the procedure is invoked by $\mathrm{DPLL}(\phi, \emptyset)$.

```
$\mathrm{DPLL}(\phi, \mathcal{I}')$
    $(\psi, \mathcal{I}) := \textsc{UnitPropagation}(\phi, \mathcal{I}')$;
    if $\psi$ contains $\{\}$
    then return $(\{\{\}\}, \emptyset)$
    elseif $\psi = \{\}$ then return $(\{\}, \mathcal{I})$
    else select a literal $\lambda \in C \in \psi$;
        if $\mathrm{DPLL}(\psi \cup \{\{\lambda\}\}, \mathcal{I}) = (\{\}, \mathcal{I}'')$
        then return $(\{\}, \mathcal{I}'')$
        else return $\mathrm{DPLL}(\psi \cup \{\{\neg\lambda\}\}, \mathcal{I})$
```

- UnitPropagation realizes the “unit propagation rule”
- The last “if then else” realizes the “splitting rule”

Properties of the DPLL procedure

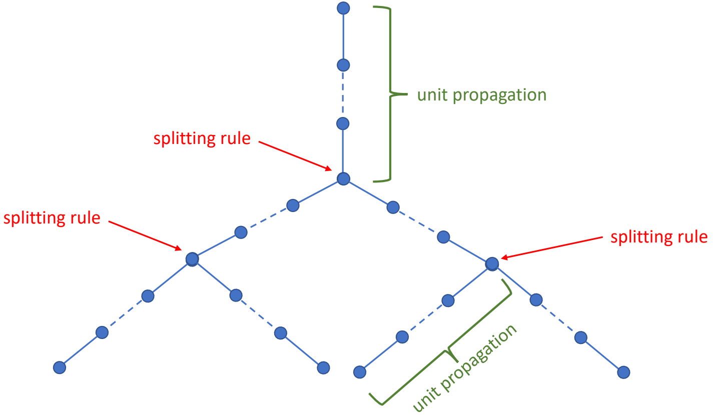
We first discuss the properties of UNITPROPAGATION. By observing that every execution of the instructions inside the WHILE loop eliminates one clause from the formula and possibly a set of literals from the other clauses of the input formula, we can easily conclude the following.

Theorem

For any $\phi, \mathcal{I}$, UNITPROPAGATION$(\phi, \mathcal{I})$ terminates, and runs in polynomial time with respect to the size of ϕ.

Derivation tree

The computation done by $\mathrm{DPLL}(\phi, \emptyset)$ can be described by a binary tree, in which every path from the root to a splitting node, or from a splitting node to another splitting node, or from a splitting node to a leaf is the sequence of operations of unit propagation.



Properties of the DPLL procedure

Since the length of every path from the root to a leaf is bound to n, where n is the number of variables in ϕ, we have that the size of the binary tree is at most 2^n. From this observation the following theorem on the worst-case time complexity follows.

Theorem

For any ϕ, $\mathrm{DPLL}(\phi, \emptyset)$ terminates, and its time complexity is $O(2^m)$, where m is the size of ϕ.

Properties of the DPLL procedure

The following theorem sanctions the correctness of DPLL.

Theorem

For any ϕ, $\mathrm{DPLL}(\phi, \emptyset)$ returns

- $(\{\}, \mathcal{I})$ if ϕ is satisfiable, and
- $(\{\{\}\}, \emptyset)$ if ϕ is unsatisfiable.

Note that when $\mathrm{DPLL}(\phi, \emptyset)$ returns $(\{\}, \mathcal{I})$, $\mathcal{I}$ is a model of ϕ.

Horn clauses

Definition

A clause is a Horn clause if it has at most one positive literal. A Horn formula is a formula in CNF all of whose clauses are Horn clauses.

Theorem

A Horn formula ϕ is satisfiable if and only if $\textsc{UnitPropagation}(\phi, \emptyset)$ returns $(\psi, \mathcal{I})$, with ψ different from $\{\{\}\}$.

Theorem

Satisfiability of Horn formulas can be solved in polynomial time.

We leave as an exercise the proof of the above theorems.