Mathematical Logics

3. Decision procedure

Luciano Serafini

Fondazione Bruno Kessler, Trento, Italy

March 13, 2013

Decision procedures

Four tipes of questions

- **Model Checking($\mathcal{I}, \phi$)**: $\mathcal{I} \overset{?}{\models} \phi$. What is the truth value of ϕ in $\mathcal{I}$, or equivalently, does $\mathcal{I}$ satisfy ϕ or does it not satisfy ϕ.
- **Satisfiability(ϕ)**: $\overset{?}{\exists}\mathcal{I} . \mathcal{I} \models \phi$ Is there a model $\mathcal{I}$ that satisfies ϕ?
- **Validity(ϕ)**: $\overset{?}{\models} \phi$. Is ϕ satisfied by all the models $\mathcal{I}$?
- **Logical consequence(Γ, ϕ)**: $\Gamma \overset{?}{\models} \phi$ Is ϕ satisfied by all the models $\mathcal{I}$, that satisfies all the formulas in 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

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

Observations

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

A simple recursive MCDP

MCDP($\mathcal{I}, \phi$) applyes one of the following cases:

```
MCDP($\mathcal{I}, p$)
    **if** $\mathcal{I}(p) = true$
        **then** return YES
    **else** return NO

MCDP($\mathcal{I}, \phi \wedge \psi$)
    **if** MCDP($\mathcal{I}, A$)
        **then** return MCDP($\mathcal{I}, \psi$)
    **else** return NO

MCDP($\mathcal{I}, \phi \vee \psi$)
    **if** MCDP($\mathcal{I}, \phi$)
        **then** return YES
    **else** return MCDP($I, \psi$)

MCDP($\mathcal{I}, \phi \supset \psi$)
    **if** MCDP($\mathcal{I}, \phi$)
        **then** return MCDP($\mathcal{I}, \psi$)
    **else** return YES

MCDP($\mathcal{I}, \phi \equiv \psi$)
    **if** MCDP($\mathcal{I}, \phi$)
        **then** return MCDP($\mathcal{I}, \psi$)
    **else** return not(MCDP($\mathcal{I}, \psi$)
```

Satisfiability

Satisfiability decision procedure

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

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

When $\mathrm{SDP}(\phi) = \textit{satisfiable}$, SDP can return a (model) $\mathcal{I}$, that satisfies ϕ. Notice that this might not be the only one.

Satisfiability

Satisfiability decision procedure

A satisfiability decision procedure SDP is an algorithm that takes in input a formula ϕ and checks if ϕ is (un)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 return a (model) $\mathcal{I}$, that satisfies ϕ. Notice that this might not be the only one.

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 an interpretation $\mathcal{I}$, this interpretation is a counter-model for 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

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

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

When $\mathrm{SDP}(\neg\phi)$ returns an interpretation $\mathcal{I}$, this interpretation is a counter-model for ϕ.

Logical consequence

Logical consequence decision procedure

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

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

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

When $\mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\phi)$ returns an interpretation $\mathcal{I}$, this interpretation is a model for Γ and a counter-model for ϕ.

Logical consequence

Logical consequence decision procedure

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

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

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

When $\mathrm{SDP}(\gamma_1 \wedge \dots \wedge \gamma_n \wedge \neg\phi)$ returns an interpretation $\mathcal{I}$, this interpretation is a model for Γ and a counter-model for ϕ.

Proof of the previous property

Theorem

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

Proof.

$\Rightarrow$ Suppose that $\Gamma \models \phi$, this means that every interpretation $\mathcal{I}$ that satisfies Γ, it does satisfy ϕ, and therefore $\mathcal{I} \not\models \neg\phi$. This implies that there is no interpretations that satisfies together Γ and $\neg\phi$.

$\Leftarrow$ Suppose that $\mathcal{I} \models \Gamma$, let us prove that $\mathcal{I} \models \phi$, Since $\Gamma \cup \{\neg phi\}$ is not satisfiable, then $\mathcal{I} \not\models \neg\phi$ and therefore $\mathcal{I} \models \phi$.

$\square$

Davis-Putnam (DP) Algorithm

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

  Davis, Putnam. A Computing Procedure for Quantification Theory. Journal of the ACM, 7(3):201ˇ2013215, 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):394ˇ2013397, 1962.

- The DP algorithm is often confused with the more popular DLL algorithm. In the literature you often find the acronym DPLL.
- Basic framework for most current SAT solvers.
- We consider the DP algorithm . . .

Conjunctive Normal form

Definition (Conjunctive Normal form)

A formula ϕ is in conjunctive normal form, if it is of the form:

$$(l_{11} \vee \dots l_{1n_1}) \wedge \dots \wedge (l_{m1} \vee \dots \vee l_{mn_m})\times$$

where l_{ij} is a literal, i.e., a formula of the form p or $\neg p$, with p atomic proposition. Each $(l_{k1} \vee \dots \vee l_{1n_k})$ is called clause

Example

$$(p \vee \neg q) \wedge (r \vee p \vee \neg r) \wedge (p \vee p) \tag{1}$$

$$p \vee q \tag{2}$$

$$p \wedge q \tag{3}$$

$$p \wedge \neg q \wedge (r \vee s) \tag{4}$$

Proposition

existence	*Every formula can be reduced into CNF*
equivalence	$\models CNF(\phi) \equiv \phi$
normality	$\models \phi \equiv \psi$ *implies that* $CNF(\phi) = CNF(\psi)$ *(up to reordering of clauses and of literals inside each clauses)*

Reduction in CNF

Definition (the CNF function)

The function *CNF*, which transforms a propositional formula in its CNF is recursively defined as follows:

$$\begin{array}{rcl}
CNF(p) & = & p \quad \text{if } p \in \mathcal{P} \\
CNF(\neg p) & = & \neg p \quad \text{if } p \in \mathcal{P} \\
CNF(\phi \supset \psi) & = & CNF(\neg\phi) \otimes CNF(\psi) \\
CNF(\neg\neg\phi) & = & CNF(\phi) \\
CNF(\neg(\phi \wedge \psi)) & = & CNF(\neg\phi) \otimes CNF(\neg\psi) \\
CNF(\neg(\phi \vee \psi)) & = & CNF(\neg\phi) \wedge CNF(\neg\psi) \\
CNF(\phi \wedge \psi) & = & CNF(\phi) \wedge CNF(\psi) \\
CNF(\phi \vee \psi) & = & CNF(\phi) \otimes CNF(\psi)
\end{array}$$

where $(C_1 \wedge \cdots \wedge C_n) \otimes (D_1 \wedge \cdots \wedge D_m)$ is defined as

$$(C_1 \vee D_1) \wedge \cdots \wedge (C_1 \vee D_m) \wedge \cdots \wedge (C_n \vee D_1) \wedge \cdots \wedge (C_n \vee D_m)$$

Termination of CNF

Proposition

CNF terminates for every input ϕ.

Proof.

- We define the *complexity of the formula* ϕ as the maximal number of nested logical operators it contains.
- Termination of this *CNF* algorithm is guaranteed since the the complexity of the formula given in input to all the recursive applications of *CNF* is always decreasing.
- Since the complexity of every formula is finite, then after a finite number of recursive calls of *CNF*, the base case is reached.

□

CNF preserves the meaning of a formula

Proposition

$\models \phi \equiv CNF(\phi)$

Proof.

By induction on the definition of CNF.

base case; ϕ is a literal $CNF(\phi) = \phi$ and, form the fact that $\models \phi \equiv \phi$ we conclude that $\models CNF(\phi) \equiv \phi$

step case: ϕ is of the form $\psi \supset \theta$. By the induction hypothesis we have that $\models CNF(\neg\psi) \equiv \neg\psi$ and $\models CNF(\theta) \equiv \theta$. Furthermore,for every α and β, $\models CNF(\alpha) \otimes CNF(\beta) \equiv CNF(\alpha) \vee CNF(\beta)$. (Prove by exercize the simple example with $\alpha = p \wedge q$ and $\beta = r \wedge s$). furthermore, $\models (\psi \supset \theta) \equiv (\neg\psi \vee \theta)$. This implies that $\models CNF(\psi \supset \theta) \equiv \psi \supset \theta$.

other step cases By exercise.

□

Proposition

$\models \phi \equiv \psi$ *implies that* $CNF(\phi) = CNF(\psi)$ *(up to reordering of clauses and of literals inside each clauses)*

Compact representation of CNF

The CNF formula $(l_{11} \vee \ldots l_{1n_1}) \wedge \ldots \wedge (l_{m1} \vee \cdots \vee l_{mn_m})$ can be represented by a set of sets

$$\{\{l_{11}, \ldots, l_{1n_1}\}, \ldots, \{l_{m1}, \ldots, l_{mn_m}\}\}$$

The $\emptyset$ represents any unsatisfiable formula. (e.g. $p \wedge \neg p$).

Satisfiability of a set of clauses

- Let $N = C_1, \ldots, C_n = CNF(\phi)$
  - $\mathcal{I} \models \phi$ 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 a model $\mathcal{I}$ satisfies N we do not need to know the truth values that $\mathcal{I}$ assigns to all the literals appearing in N.
- For instance, if $\mathcal{I}(p) = true$ and $\mathcal{I}(q) = false$, 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 evaluation

A partial evaluation 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 Valuation

- Partial evaluations allow us to construct models for a set of clauses $N = \{C_1, \ldots, C_n\}$ incrementally
- DPLL starts with an empty valuation (i.e., the truth values of all propositional letters are not defined) and tries to extend it step by step to all variables occurring in $N = \{C_1, \ldots, C_n\}$.
- Under a partial valuation $\mathcal{I}$ literals and clauses can be true, false or undefined;
  - A clause is true under $\mathcal{I}$ if one of its literals is true;
  - A clause is false (or conflicting) if all its literals are false
  - otherwise C it is undefined (or unresolved).

DPLL

Simplification of a formula by an evaluated literal

For any CNF formula ϕ and atom p, $\phi|_p$ stands for the formula obtained from ϕ by replacing all occurrences of p by $\top$ and simplifying the result by removing

- all clauses containing the disjunctive term $\top$, and
- the literals $\neg\top$ in all remaining clauses

Similarly, $\phi|_{\neg p}$ is the result of replacing p in ϕ by $\bot$ and simplifying the result.

Example

For instance,

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

DPLL

Simplification of a formula by an evaluated literal

For any CNF formula ϕ and atom p, $\phi|_p$ stands for the formula obtained from ϕ by replacing all occurrences of p by $\top$ and simplifying the result by removing

- all clauses containing the disjunctive term $\top$, and
- the literals $\neg\top$ in all remaining clauses

Similarly, $\phi|_{\neg p}$ is the result of replacing p in ϕ by $\bot$ and simplifying the result.

Example

For instance,

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

DPLL (cont'd)

Unit clause

If a CNF formula ϕ contains a clause $C = \{l\}$ that consists of a single literal, it is a unit clause

Unit propoagation

If ϕ contains unit clause $\{l\}$ then, to satisfy ϕ we have to satisfy $\{l\}$ and therefore the literal l must be evaluated to True. As a consequence ϕ can be simplified using the procedure called UnitPropagation

```
UnitPropagation($\phi, \mathcal{I}$)
    **while** $\phi$ contains a unit clause $\{l\}$
        $\phi := \phi|_l$
        if $l = p$, then $\mathcal{I}(p) := true$
        if $l = \neg p$, then $\mathcal{I}(p) := false$
    **end**
```

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Example

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

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

Exercize

Use unit propagation to decide whether the formula

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

is satisfiable.

DPLL (cont'd)

Remark

Unit propagation is enough to decide the satisfiability problem when it terminates with the following two results:

- $\{\}$ as in the example above, then the initial formula is satisfiable, and a satisfying interpretation can be easily extracted from $\mathcal{I}$.
- $\{\ldots\{\}\ldots\}$, then 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 and the CNF is not empty and doesn't contain empty clauses. e.g.,

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

In this case we have to do a guess ....

DPLL definition

The Davis-Putnam-Logemann-Loveland procedure

. . . is an extension of the unit propagation method that can solve the satisfiability

```
DPLL($\phi, \mathcal{I}$)
    UnitPropagation($\phi, \mathcal{I}$)
    if $\phi$ contains the empty clause
        then return
    if $\phi = \{\}$
        then exit with $\mathcal{I}$
    select a literal $l \in C \in \phi$
    DPLL($\phi|_l, \mathcal{I} \cup \mathcal{I}(l) = true$)
    DPLL($\phi|_{\bar{l}}, \mathcal{I} \cup \mathcal{I}(l) = false$)
```

where: if $l = p$, $\bar{l} = \neg p$ and if $l = \neg p$ then $\bar{l} = p$

Other examples

Exercize

Check the following facts via DPLL

1. $\models (p \supset q) \wedge \neg q \supset \neg p$
2. $\models (p \supset q) \supset (p \supset \neg q)$
3. $\models (p \vee q \supset r) \vee p \vee q$
4. $\models (p \vee q) \wedge (p \supset r \wedge q) \wedge (q \supset \neg r \wedge p)$
5. $\models (p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$
6. $\models (p \vee q) \wedge (\neg q \wedge \neg p)$
7. $\models (\neg p \supset q) \vee ((p \wedge \neg r) \equiv q)$
8. $\models (p \supset q) \wedge (p \supset \neg q)$
9. $\models (p \supset (q \vee r)) \vee (r \supset \neg p)$

Other examples

Exercize

Check the following facts

1. $(p \supset q) \models \neg p \supset \neg q$
2. $(p \supset q) \wedge \neg q \models \neg p$
3. $p \supset q \wedge r \models (p \supset q) \supset r$
4. $p \vee (\neg q \wedge r) \models q \vee \neg r \supset p$
5. $\neg(p \wedge q) \equiv \neg p \vee \neg q$
6. $(p \vee q) \wedge (\neg p \supset \neg q) \equiv q$
7. $(p \wedge q) \vee r \equiv (p \supset \neg q) \supset r$
8. $(p \vee q) \wedge (\neg p \supset \neg q) \equiv p$
9. $((p \supset q) \supset q) \supset q \equiv p \supset q$

Reducing Graph Coloring to SAT

graph k-coloring problem

A k-coloring of a graph is a labelling of its vertices with at most k colors such that no two vertices sharing the same edge have the same color.

Reduction to SAT

The problem of generating a k-coloring of a graph $G = (V, E)$ can be reduced to SAT as follows.

- For every $v \in V$ and every $i \in \{1, \ldots, k\}$, introduce an atom p_{vi} to represent the fact that the node v is labelled with the i-th color.

Reducing Graph Coloring to SAT

Reduction to SAT (cont'd)

- The propositional formulas:

$$\bigwedge_{v\in V}\left(\bigvee_{1\leq i\leq k} p_{vi}\right)$$

  represents the fact that all the vertexes need to be colored with at least one color.
- the formula

$$\bigwedge_{v\in V}\left(\bigwedge_{1\leq i<j\leq k} \neg(p_{vi}\wedge p_{vj})\right)$$

  represents the fact that a node can be colored with at most one color
- the formula

$$\bigwedge_{(v,w)\in E}\left(\bigwedge_{1\leq i\leq k} \neg(p_{vi}\wedge p_{wi})\right)$$

  represents the fact that every two adjacent nodes (v, w) cannot be labelled with the same color i.

MiniSat `http://minisat.org`

About

MiniSat is a minimalistic, open-source SAT solver, developed to help researchers and developers alike to get started on SAT. It is released under the MIT licence, and is currently used in a number of projects (see "Links"). On this page you will find binaries, sources, documentation and projects related to MiniSat, including the Pseudo-boolean solver MiniSat+ and the CNF minimizer/preprocessor SatELite.

How to use MiniSat

Input format

MiniSat, like most SAT solvers, accepts its input in a simplified "DIMACS CNF" format, which is a simple text format. Every line beginning "c" is a comment. The first non-comment line must be of the form:

```
p cnf NUMBER_OF_VARIABLES NUMBER_OF_CLAUSES
```

Each of the non-comment lines afterwards defines a clause. Each of these lines is a space-separated list of variables; a positive value means that corresponding variable (so 4 means x4), and a negative value means the negation of that variable (so -5 means -x5). Each line must end in a space and the number 0.

```
c Here is a comment
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

is the representation of the CNF

$$\{\{x_1, \neg x_5, x_4\}, \{\neg x_1, x_5, x_3, x_4\}, \{\neg x_3, \neg x_4\}\}$$

Invoking MiniSat

MiniSAT's usage is:

```
minisat [options] [INPUT-FILE [RESULT-OUTPUT-FILE]]
```

MiniSat output format

- When run, miniSAT sends to standard error a number of different statistics about its execution. It will output to standard output either "`SATISFIABLE`" or "`UNSATISFIABLE`" (without the quote marks), depending on whether or not the expression is satisfiable or not.
- If you give it a `RESULT-OUTPUT-FILE`, MiniSat will write text to the file. The first line will be "`SAT`" (if it is satisfiable) or "`UNSAT`" (if it is not). If it is `SAT`, the second line will be set of assignments to the boolean variables that satisfies the expression. (There may be many others; it simply has to produce one assignment).
- for example the output file of the previous example is

  ```
  SAT
  1 2 -3 4 5 0
  ```

  This means that it is satisfiable, with the model $\mathcal{I}$ with $\mathcal{I}(x_1) = true, \mathcal{I}(x_2) = true, \mathcal{I}(x_3) = false, \mathcal{I}(x_4) = true$ and $\mathcal{I}(x_5) = true$.