Mathematical Logics

3. Decision procedure

Luciano Serafini

Fondazione Bruno Kessler, Trento, Italy

January 17, 2014

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 naive algorithm for model checking

A simple way to check if $\mathcal{I} \models \phi$

(1) Replace each occurrence of a propositional variables in ϕ with the truth value assigned by $\mathcal{I}$. I.e. replace each p with $\mathcal{I}(p)$ (2) Recursively apply the following reduction rules for connectives:

$$\begin{aligned} \mathit{true} \wedge \mathit{true} &= \mathit{true} \\ \mathit{true} \wedge \mathit{false} &= \mathit{false} \\ \mathit{false} \wedge \mathit{true} &= \mathit{false} \\ \mathit{false} \wedge \mathit{false} &= \mathit{false} \\ \hline \mathit{true} \vee \mathit{true} &= \mathit{true} \\ \mathit{true} \vee \mathit{false} &= \mathit{true} \\ \mathit{false} \vee \mathit{true} &= \mathit{true} \\ \mathit{false} \vee \mathit{false} &= \mathit{false} \\ \hline \neg \mathit{true} &= \mathit{false} \\ \neg \mathit{false} &= \mathit{true} \end{aligned}$$	$$\begin{aligned} \mathit{true} \rightarrow \mathit{true} &= \mathit{true} \\ \mathit{true} \rightarrow \mathit{false} &= \mathit{false} \\ \mathit{false} \rightarrow \mathit{true} &= \mathit{true} \\ \mathit{false} \rightarrow \mathit{false} &= \mathit{true} \\ \hline \mathit{true} \equiv \mathit{true} &= \mathit{true} \\ \mathit{true} \equiv \mathit{false} &= \mathit{false} \\ \mathit{false} \equiv \mathit{true} &= \mathit{false} \\ \mathit{false} \equiv \mathit{false} &= \mathit{true} \end{aligned}$$

A naive algorithm for model checking (example)

Example

- $\phi = p \vee (q \rightarrow r)$
- $\mathcal{I} = \mathcal{I}(p) = \textit{false}, \mathcal{I}(q) = \textit{false}, \mathcal{I}(r) = \textit{true}$

To check if $\mathcal{I} \models p \vee (q \rightarrow r)$ we:

(1) replace, p, q, and r in ϕ with $\mathcal{I}(p)$, $\mathcal{I}(q)$ and $\mathcal{I}(r)$, obtaining

$$\textit{false} \vee (\textit{false} \rightarrow \textit{true})$$

(1) recursively apply the reduction rules

$$\begin{array}{l} \textit{false} \vee (\textit{false} \rightarrow \textit{true}) \\ \textit{false} \vee \textit{true} \\ \textit{true} \end{array}$$

A simple optimization of MCDP

MCDP($\mathcal{I}, \phi$) with lazy evaluation

Idea: When you evaluate a conjunction, if the first conjunct is evaluated to false, then you can jump to the conclusion that the whole conjunction is false, without evaluating the second conjunct. Similar idea can be applied to the other connectives ($\vee$, $\rightarrow$ and $\equiv$)

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

MCDP($\mathcal{I}, \phi \wedge \psi$)
  if MCDP($\mathcal{I}, \phi$)
    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, ψ)

MCDP($\mathcal{I}, \phi \rightarrow \psi$)
  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 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 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

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

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

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

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 $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$. LCDP can be implemented on the basis of satisfiability decision procedure by exploiting the property

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

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

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

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

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

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

- 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

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

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

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 the two clauses are equivalent.

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

Multiple literals can be merged

If a clause contains more than one occurrence of the same literal then it is equivalent to the close obtained by deleting all but one of these occurrences:

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

Clauses as set of literals

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

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

Properties of formulas in CNF

Order of claused does not matter

If a clause C is obtained by reordering the literals of a clause C' then the two clauses are equivalent.

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

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 of the duplicated occurrences:

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

a CNF formula as a set of sets of literals

From the props. of clauses and of CNF formulas, we can represent a CNF formula as a set of sets of literals.

$(a \vee b) \wedge (c \vee \neg b) \wedge (\neg b)$ is represented by the set of sets $\{\{a, b\}, \{c, \neg b\}, \{\neg b\}\}$

Proposition

existence *Every formula can be reduced into CNF*

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

Reduction in CNF

Definition (the CNF function)

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

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

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

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

CNF transformation example

$$
\begin{aligned}
& CNF((a \wedge b) \vee (c \wedge d)) && = \\
& CNF(a \wedge b) \otimes CNF(c \wedge d) && = \\
& (CNF(a) \wedge CNF(b)) \otimes (CNF(c) \wedge CNF(d)) && = \\
& (a \wedge b) \otimes (c \wedge d) && = \\
& (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \wedge d)
\end{aligned}
$$

CNF transformation example

$$
\begin{aligned}
&CNF((\neg((p \to q) \land (p \lor q \to r)) \to (p \to r))) &=\\
&CNF(\neg\neg((p \to q) \land (p \lor q \to r))) \otimes CNF(p \to r) &=\\
&CNF((p \to q) \land (p \lor q \to r)) \otimes (CNF(\neg p) \otimes CNF(r)) &=\\
&(CNF(p \to q) \land CNF(p \lor q \to r)) \otimes (\neg p \lor r) &=\\
&((CNF(\neg p) \otimes CNF(q)) \land (CNF(\neg(p \lor q)) \otimes CNF(r))) \otimes (\neg p \lor r) &=\\
&((\neg p \otimes q) \land ((CNF(\neg p) \land CNF(\neg q)) \otimes CNF(r))) \otimes (\neg p \lor r) &=\\
&((\neg p \otimes q) \land ((\neg p \land \neg q) \otimes r)) \otimes (\neg p \lor r) &=\\
&((\neg p \lor q) \land (\neg p \lor r) \land (\neg q \lor r)) \otimes (\neg p \lor r) &=\\
&((\neg p \lor q \lor \neg p \lor r) \land (\neg p \lor r \lor \neg p \lor r) \land (\neg q \lor r \lor \neg p \lor r) &=\\
&((\neg p \lor q \lor r) \land (\neg p \lor r) \land (\neg q \lor r \lor \neg p)
\end{aligned}
$$

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.

$\square$

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 $\psi \rightarrow \theta$. 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, $\models (\psi \rightarrow \theta) \equiv (\neg\psi \vee \theta)$. This implies that $\models CNF(\psi \rightarrow \theta) \equiv \psi \rightarrow \theta$.

other step cases By exercise.

□

CNF transformation

Cost of CNF

CNF is a normal form, it is simpler since it uses only 3 connective (e.g., $\wedge$, $\vee$ and $\neg$) in a very specific form. Checking satisfiability/validity of a formula in CNF is easier. But there is a price: ...

Example (Exponential explosion)

Compute the CNF of

$$p1 \equiv (p2 \equiv (p3 \equiv (p4 \equiv (p5 \equiv p6)))).$$

The first step yields:

$$\begin{aligned} & CNF(p1 \rightarrow (p2 \equiv (p3 \equiv (p4 \equiv (p5 \equiv p6)))))\wedge \\ & CNF((p2 \equiv (p3 \equiv (p4 \equiv (p5 \equiv p6)))) \rightarrow p1) \end{aligned}$$

If we continue, the formula will grow exponentially.

Contrasting exponential explosion

Replace subformulas

$$p1 \equiv (p2 \equiv (p3 \equiv (p4 \equiv (p5 \equiv p6))))$$

by names:

$$n5 \equiv (p5 \equiv p6)$$

$$p1 \equiv (p2 \equiv (p3 \equiv (p4 \equiv n1)))$$

After several steps

$$\begin{array}{ll} p1 \equiv (p2 \equiv n3) & n3 \equiv (p3 \equiv n4) \\ n4 \equiv (p4 \equiv n5) & n5 \equiv (p5 \equiv p6) \end{array}$$

The resulting formula is different from (and not equivalent to) the initial one. But they are equi-satisfiable,

Equi-Satisfiability

Two formulas ϕ and ϕ' are equisatisfiable iff:

ϕ is satisfiable if and only if ϕ' is satisfiable

- If two formulas are equi-satisfiable, are they equivalent? No!
- Example: Any satisfiable formula (e.g., p) is equisat as $\top$ But clearly, $p \equiv \top$ is not valid!
- Another example: Introducing names leads to equisatisfiable formulas. E.g. the formula $a \wedge b$ is equisatisfiable of the formula $(n \equiv a \wedge b) \wedge n$, but it is not true that

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

- Equisatisfiability is a much weaker notion than equivalence. But useful if all we want to do is determine satisfiability.

Tseitin's Transformation

Tseitins transformation

converts formula ϕ to equisatisfiable formula ϕ' in CNF with only a linear increase in size.

Tseitin's transformation procedure I

- Step 1: Introduce a new variable p_ψ for every subformula ψ of ϕ (unless ψ is already an atom).
- For instance, if $\phi = \psi_1 \wedge \psi_2$, introduce two variables p_{ψ_1} and p_{ψ_2} representing ψ_1 and ψ_2 respectively.
- p_{ψ_1} is said to be representative of ψ_1 and p_{ψ_2} is is representative of ψ_2.

Tseitin's transformation procedure II

- Step 2: Consider each subformula $\psi \equiv \psi_1 \circ \psi_2$ ($\circ$ is an arbitrary boolean connective)
- Stipulate representative of ψ is equivalent to representative of $\psi_1 \circ \psi_2$

$$p_\psi \equiv p_{\psi_1} \circ p_{\psi_2}$$

- Step 3: Convert $p_\psi \equiv p_{\psi_1} \circ p_{\psi_2}$ to equivalent CNF
- Observe: Since $p_\psi \equiv p_{\psi_1} \circ p_{\psi_2}$ contains at most three propositional variables and exactly two connectives, size of this formula in CNF is bound by a constant.

Tseitin's transformation procedure III

- Given original formula ϕ, let p_ϕ be its representative and let $subf(\phi)$ be the set of all subformulas of ϕ (including ϕ itself).
- Then, introduce the formula

$$p_\phi \wedge \bigwedge_{\psi_1 \circ \psi_2 \in subf(\phi)} CNF(p_{\psi_1 \circ \psi_2} \equiv p_{\psi_1} \circ p_{\psi_2})$$

- Claim: This formula is equisatisfiable to ϕ.
- The proof is by standard induction; left as homework exercise.
- Formula is also in CNF because conjunction of CNF formulas is in CNF.

Tseitin's Transformation and Size

- Using this transformation, we converted ϕ to an equisatisfiable CNF formula ϕ'.
- What about the size of ϕ?

$$p_\phi \wedge \bigwedge_{\psi_1 \circ \psi_2 \in subf(\phi)} CNF(p_{\psi_1 \circ \psi_2} \equiv p_{\psi_1} \circ p_{\psi_2})$$

- $|subf(\phi)|$ is the bound by the number of connectives in ϕ.
- Each formula $CNF(p_\psi \equiv p_{\psi_1} \circ p_{\psi_2})$ has constant size.
- Thus, trasformation causes only linear increase in formula size.
- More precisely, the size of resulting formula is bound by $3n + 2$ where n is size of original formula

Tseitin's Transformation - Example

Convert $\phi : p \vee q \rightarrow p \wedge \neg r$ to equisatisfiable CNF formula.

1. For each subformula, introduce new variables:
   x_1 for ϕ, x_2 for $p \vee q$, x_3 for $p \wedge \neg r$, and x_4 for $\neg r$.
2. Stipulate equivalences and convert them to CNF:

$$
\begin{array}{rcl}
x_1 \equiv (x_2 \rightarrow x_3) & \Rightarrow & \phi_1 : (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_1) \wedge (\neg x_3 \vee x_1) \\
x_2 \equiv (p \vee q) & \Rightarrow & \phi_2 : (\neg x_2 \vee p \vee q) \wedge (\neg p \vee x_2) \wedge (\neg q \vee x_2) \\
x_3 \equiv (p \wedge x_4) & \Rightarrow & \phi_3 : (\neg x_3 \vee p) \wedge (\neg x_3 \vee x_4) \wedge (\neg p \vee \neg x_4 \vee x_3) \\
x_4 \equiv \neg r & \Rightarrow & \phi_4 : (\neg x_4 \vee \neg r) \wedge (x_4 \vee r)
\end{array}
$$

3. The formula is equisatisfiable to ϕ and is in CNF.

$$x_1 \wedge \phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4$$

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 $\mathcal{I}(p) = \mathit{true}$ 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 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).

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 $\perp$ 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 $\textsc{UnitPropagation}$

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

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

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

$$\{\{p\},\{\neg p,\neg q\},\{\neg q,r\}\}$$
$$\{\{p\},\{\neg p,\neg q\},\{\neg q,r\}\}|_p \qquad \mathcal{I}(p) = true$$
$$\{\{\top\},\{\neg\top,\neg q\},\{\neg q,r\}\}$$
$$\{\{\neg q\},\{\neg q,r\}\}$$
$$\{\{\neg q\},\{\neg q,r\}\}$$
$$\{\{\neg q\},\{\neg q,r\}\}|_{\neg q} \qquad \mathcal{I}(q) = false$$
$$\{\{\top\},\{\top,r\}\}$$
$$\{\}$$

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)

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 $\mathcal{I}$.
- $\{\dots\{\}\dots\}$, 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

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

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

Other examples

Exercize

Check the following facts

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

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 $\mathcal{I}(x_1) = \mathit{true}, \mathcal{I}(x_2) = \mathit{true}, \mathcal{I}(x_3) = \mathit{false}, \mathcal{I}(x_4) = \mathit{true}$ and $\mathcal{I}(x_5) = \mathit{true}$.