Knowledge representation and learning

3. Decision procedure

Luciano Serafini

Fondazione Bruno Kessler, Trento, Italy

March 13, 2023

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

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

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

b

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:

$$
\begin{array}{rcl}
\mathit{true} \wedge \mathit{true} & = & \mathit{true} \\
\mathit{true} \wedge \mathit{false} & = & \mathit{false} \\
\mathit{false} \wedge \mathit{true} & = & \mathit{false} \\
\mathit{false} \wedge \mathit{false} & = & \mathit{false} \\
\hline
\mathit{true} \vee \mathit{true} & = & \mathit{true} \\
\mathit{true} \vee \mathit{false} & = & \mathit{true} \\
\mathit{false} \vee \mathit{true} & = & \mathit{true} \\
\mathit{false} \vee \mathit{false} & = & \mathit{false} \\
\hline
\neg \mathit{true} & = & \mathit{false} \\
\neg \mathit{false} & = & \mathit{true}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{true} \rightarrow \mathit{true} & = & \mathit{true} \\
\mathit{true} \rightarrow \mathit{false} & = & \mathit{false} \\
\mathit{false} \rightarrow \mathit{true} & = & \mathit{true} \\
\mathit{false} \rightarrow \mathit{false} & = & \mathit{true} \\
\hline
\mathit{true} \equiv \mathit{true} & = & \mathit{true} \\
\mathit{true} \equiv \mathit{false} & = & \mathit{false} \\
\mathit{false} \equiv \mathit{true} & = & \mathit{false} \\
\mathit{false} \equiv \mathit{false} & = & \mathit{true} \\
\hline
\end{array}
$$

A naive algorithm for model checking (example)

Example

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

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

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

(2) recursively apply the reduction rules

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

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 connectives $\vee$ and $\rightarrow$. This leads to the following optimized rewriting rules:

$$\top \wedge \top \Rightarrow \top \qquad \top \vee * \Rightarrow \top \qquad * \rightarrow \top \Rightarrow \top$$
$$* \wedge \bot \Rightarrow \bot \qquad * \vee \top \Rightarrow \top \qquad \top \rightarrow \bot \Rightarrow \bot$$
$$\bot \wedge * \Rightarrow \bot \qquad \bot \vee \bot \Rightarrow \bot \qquad \bot \rightarrow * \Rightarrow \top$$

where "$*$" denotes either $\top$ or $\bot$ and therefore the corresponding expression does not need to be evaluated.

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 it return one model, amond all the possible models of ϕ.

Validity

Validity decision procedure

A decision procedure for Validity VDC, is an algorithm that checks whether a formula ϕ is valid.
A VDP can be obtained by using an SDP and exploiting the equivalence:

$$\phi \text{ is valid} \iff \neg\phi \text{ is not satisfiable}$$

```
def VDP(φ):
    if SDP(¬φ) == Unsatisfiable:
        Return valid
    else:
        Return not valid
```

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

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

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

When $\mathrm{SDP}(\gamma_1 \wedge \cdots \wedge \gamma_n \wedge \neg\phi)$ returns 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 $\mathcal{I} \models \phi$.

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):201ˇ013215, 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):394ˇ013397, 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 ...

Negated Normal form (NNF)

- A literal is either an atomic formula or the negation of an atomic formula.
- Examples of literals are p, q, $\neg p$, $\neg q$.
- A positive literal is antom (without negation in front)
- A negative literal is a literal with negation in front.

Definition (Negation Normal Form)

A formula is in negation normal form (NNF) if negation connective ($\neg$) occurs only in literals; or equivalently if the negation connective occurs only in front of atomic formulas.

Reduction to NNF

Any formula can be transformed into an equivalent formula in NNF, by applying the following set of rule that pushes the negation operator inwards

$$\begin{aligned} \neg\neg\phi &\Rightarrow \phi \\ \neg(\phi \wedge \psi) &\Rightarrow \neg\phi \vee \neg\psi \\ \neg(\phi \vee \psi) &\Rightarrow \neg\phi \wedge \neg\psi \\ \neg(\phi \rightarrow \psi) &\Rightarrow \phi \wedge \neg\psi \\ \neg(\phi \equiv \psi) &\Rightarrow (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi) \end{aligned} \tag{1}$$

Proposition

For every rule $\phi \Rightarrow \phi'$ in (1), ϕ and ϕ' are equivalent.

Proof.

To prove the property we have to shoe that for every transoformation $\phi \Rightarrow \phi'$ in (1) we have that ϕ and ϕ' are equivalent. □

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.

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

we also admit the clause with no literals, called the empty clause denoted by $\perp$

Satisfiability of a clause

- $\mathcal{I} \models C$ if there is a literal l_i s.t., $\mathcal{I} \models l_i$
- $\mathcal{I}$ never satisfies the empty clause $\mathcal{I} \not\models \bot$
- a unit clauses is a clause with one single literal
- Empty clauses and unit clauses play an important role in the satisfiability checking procedure,

Conjunctive normal form

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

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

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

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

$(p \vee q \vee r \vee \neg r)$ 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 $\models CNF(\phi) \equiv \phi$

Existence of CNF(ϕ)

A procedure for reducing a formula in CNF can be obtained by applying the following steps

1. transform the implications $\rightarrow$ and the equivalence $\leftrightarrow$ with equivalent formulas using $\neg$, $\wedge$ and $\vee$

$$A \rightarrow B \Longrightarrow \neg A \vee B$$
$$A \leftrightarrow B \Longrightarrow (\neg A \vee B) \wedge (\neg B \vee A)$$

2. push the negation operator $\neg$ in front of the atomic propositions (NNF)

$$\neg(A \vee B) \Longrightarrow \neg A \wedge \neg B$$
$$\neg(A \wedge B) \Longrightarrow \neg A \vee \neg B$$

3. Distribute disjunction $\vee$ over conjunction $\wedge$

$$A \vee (B \wedge C) \Longrightarrow (A \vee B) \wedge (A \vee C)$$
$$(A \wedge B) \vee C) \Longrightarrow (A \vee C) \wedge (B \vee C)$$

Termination of CNF

Proposition

CNF terminates for every input ϕ.

Proposition

ϕ is equivalent to CNF(ϕ).

Termination of CNF

Exercise 1:

Thransform the following formula in CNF

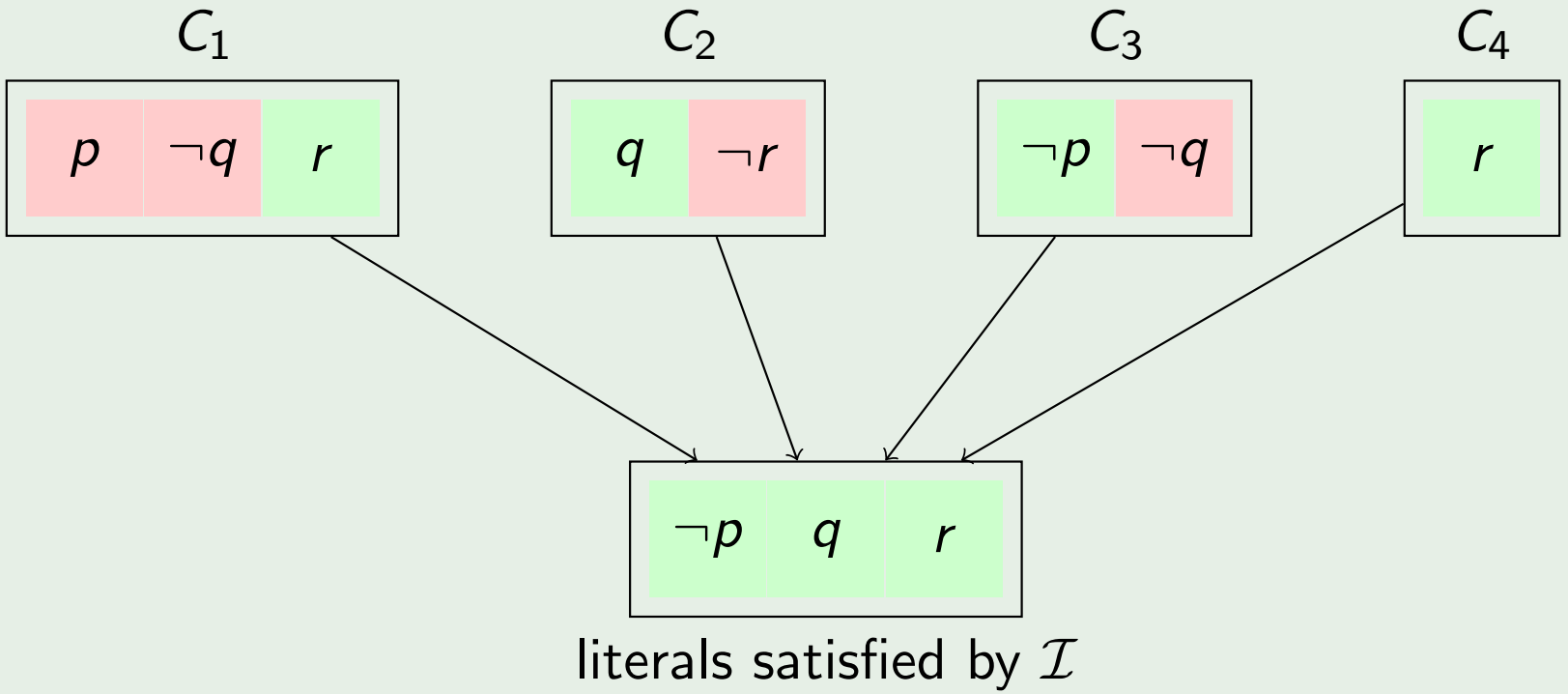
1. $(p \to q) \wedge \neg q \to \neg p$
2. $(p \to q) \to (p \to \neg q)$
3. $(p \vee q \to r) \vee p \vee q$
4. $(p \vee q) \wedge (p \to r \wedge q) \wedge (q \to \neg r \wedge p)$
5. $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$
6. $(p \vee q) \wedge (\neg q \wedge \neg p)$
7. $(\neg p \to q)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

Satisfying a CNF

$\mathcal{I} \models \{C_1, \ldots, C_n\}$, if $\mathcal{I}$ satisfies at least on literal for every C_i

Example

$$\{\{p, \neg q, r\}, \{q, \neg r\}, \{\neg p, \neg q\}, \{r\}\} \quad (2)$$



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:

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

If we continue, the formula will grow exponentially.

CND: Exponential explosion

$$\text{CNF}(p \equiv (q \equiv (r \equiv s)))$$

result in 8 clauses

$$(\neg r, \neg s, \neg q, p) \quad (r, s, \neg q, p) \quad (\neg s, r, q, p) \quad (\neg r, s, q, p)$$
$$(\neg r, \neg s, q, \neg p) \quad (r, s, q, \neg p) \quad (\neg s, r, \neg q, \neg p) \quad (\neg r, s, \neg q, \neg p)$$

CNF: Exponential explosion

$$p \equiv (q \equiv (r \equiv (s \equiv t)))$$

result in $8 \cdot 2 = 16$ clauses

$$
\begin{array}{cccc}
(\neg s, \neg t, \neg r, \neg q, p) & (s, t, \neg r, \neg q, p) & (\neg t, s, r, \neg q, p) & (\neg s, t, r, \neg q, p) \\
(\neg s, \neg t, r, q, p) & (s, t, r, q, p) & (\neg t, s, \neg r, q, p) & (\neg s, t, \neg r, q, p) \\
(\neg s, \neg t, \neg r, q, \neg p) & (s, t, \neg r, q, \neg p) & (\neg t, s, r, q, \neg p) & (\neg s, t, r, q, \neg p) \\
(\neg s, \neg t, r, \neg q, \neg p) & (s, t, r, \neg q, \neg p) & (\neg t, s, \neg r, \neg q, \neg p) & (\neg s, t, \neg r, \neg q, \neg p)
\end{array}
$$

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

$$p1 \equiv (p2 \equiv n3) \qquad n3 \equiv (p3 \equiv n4)$$
$$n4 \equiv (p4 \equiv n5) \qquad n5 \equiv (p5 \equiv p6)$$

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 pair of atomic formulas p and q are equi-satisfiable, but not equivalent (i.e., $p \equiv q$ is not valid)
- Another example: Introducing names for subformulas The formula $a \wedge b \rightarrow c$ is equi-satisfiable to the formula $(n \equiv a \wedge b) \wedge (n \rightarrow c)$, but the two formulas

$$(a \wedge b) \rightarrow c \qquad (a \wedge b \equiv n) \wedge (n \rightarrow c)$$

  are not equivalent.
- Equisatisfiability is a much weaker notion than equivalence. But useful if all we want to do is determine satisfiability.

Tseitin's Transformation

Tseitin's 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 a literal.
- 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 a formula ϕ, let p_ϕ be its representative and let $subf(\phi)$ be the set of all subformulas of ϕ (including ϕ itself).
- Consider the formula

$$p_\phi \wedge \bigwedge_{\psi_1 \circ \psi_2 \in subf(\phi)} \mathrm{CNF}(p_{\psi_1 \circ \psi_2} \equiv p_{\psi_1} \circ p_{\psi_2}) \qquad (3)$$

- (3) is in CNF
- Claim: it is equisatisfiable to ϕ.
- The proof is by standard induction; left as homework exercise.

Tseitin's Transformation and Size

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

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

- $|subf(\phi)|$ is the bound by the number of connectives in ϕ.
- Each formula $\text{CNF}(p_\psi \equiv p_{\psi_1} \circ p_{\psi_2})$ 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 te formula ϕ equal to $p \vee q \rightarrow p \wedge \neg r$ 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:

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

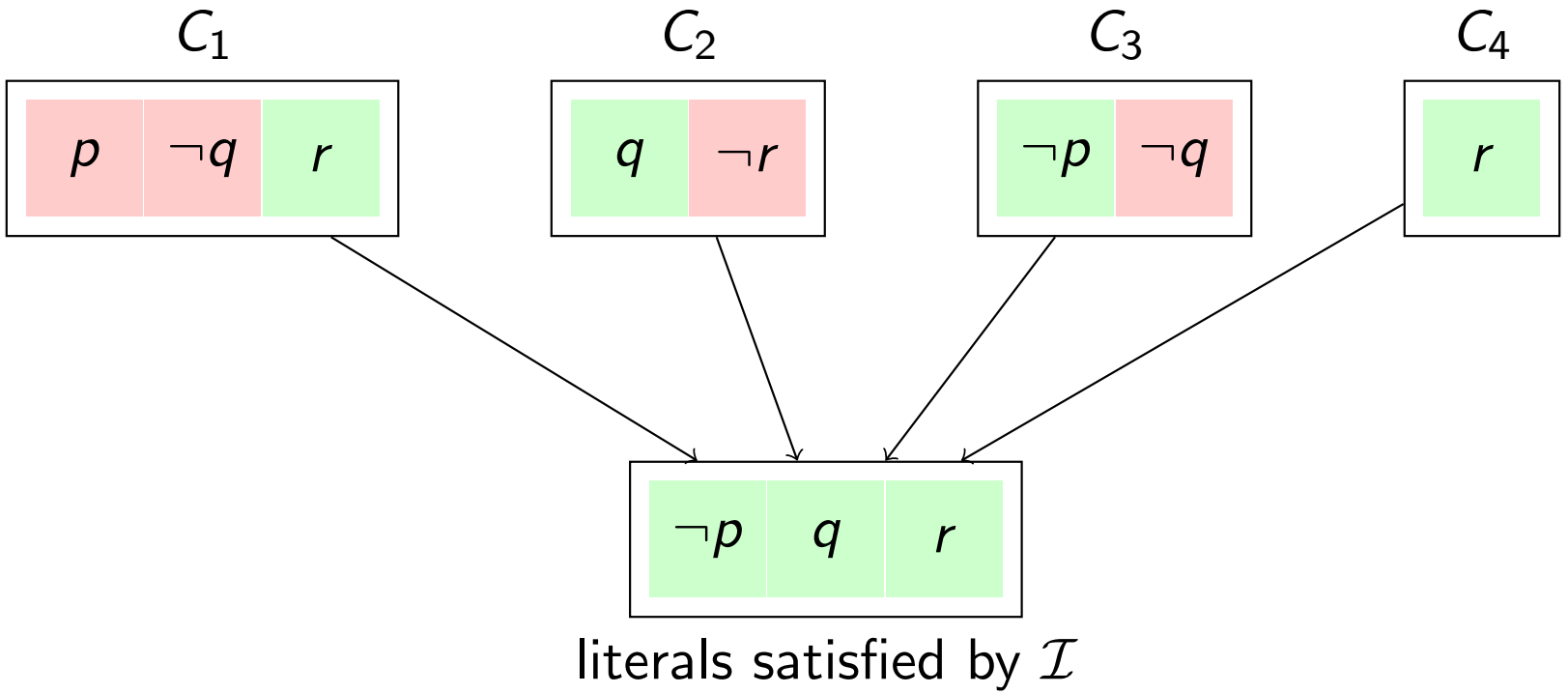
3. The formula is equisatisfiable to ϕ and is in CNF.

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

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 $l \in C$, $\mathcal{I} \models l$



Paertial evaluation

- 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 $\bot$ and simplifying the result.

Example

For instance,

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

DPLL Unit propagation

Unit clause

- If a CNF formula ϕ contains a clause $C = \{l\}$ that consists of a single literal, it is a unit clause
- 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$(\mathcal{I}, \phi)$

1: **while** ϕ contains a unit clause $\{l\}$ **do**
2: $\quad \mathcal{I}, \phi \leftarrow \mathcal{I} \cup \{l\}, \phi|_l$
3: $\quad$ **if** $\{\} \in \phi$ **then**
4: $\quad\quad$ **return** $\mathcal{I}, \phi$
5: $\quad$ **end if**
6: **end while**
7: **return** $\mathcal{I}, \phi$

DPLL (cont'd)

Example

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

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

Exercise 2:

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 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.,

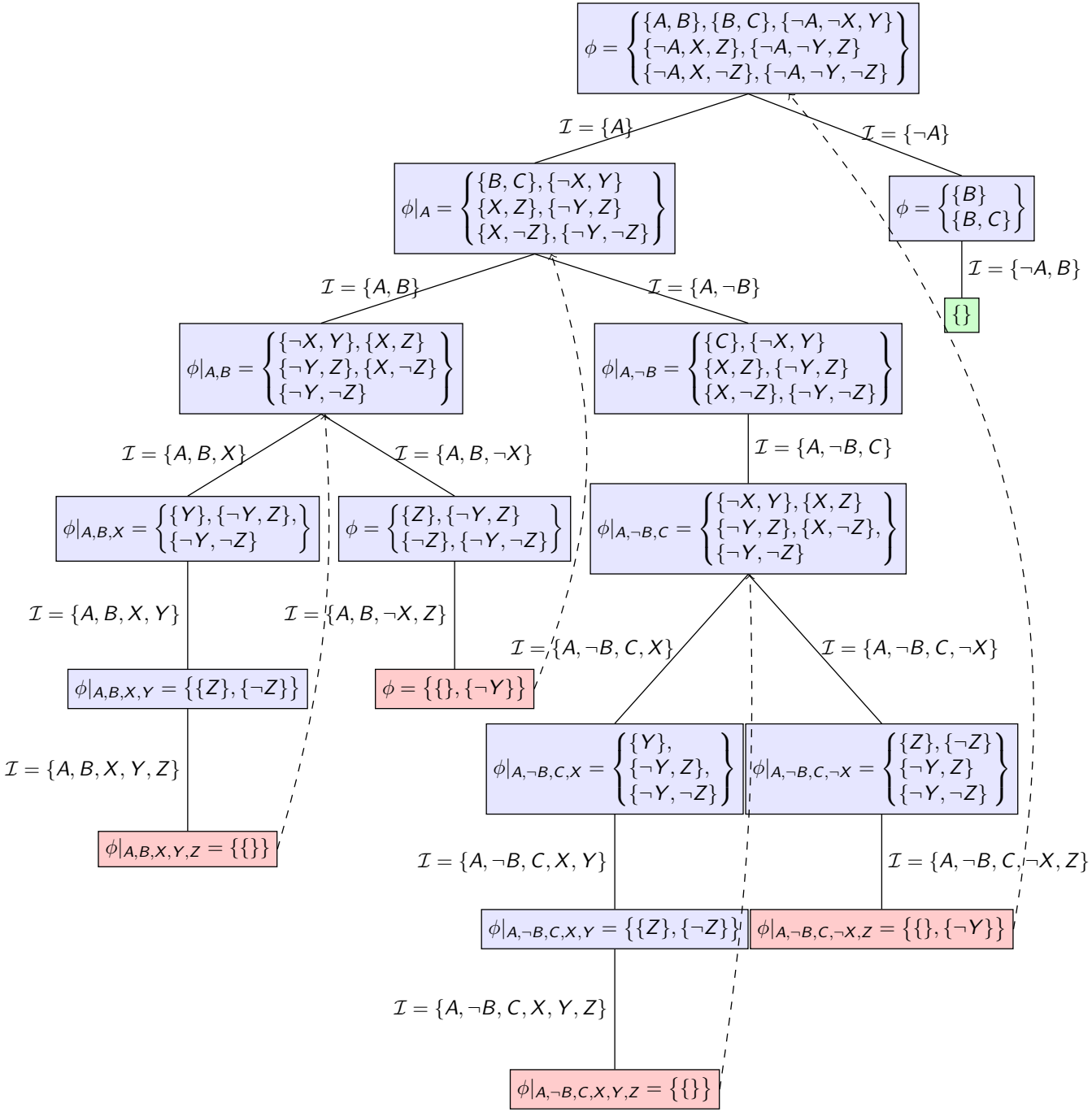
$$\{\{p, q\}, \{\neg q, r\}\}$$

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

DPLL definition

The Davis-Putnam-Logemann-Loveland procedure

1: $\mathcal{I}, \phi \leftarrow \text{UnitPropagation}(\mathcal{I}, \phi)$
2: **if** $\{\} \in \phi$ **then**
3: **return** Unsatisfiable
4: **end if**
5: **if** $\phi = \{\}$ **then**
6: **return** $\mathcal{I}$
7: **else**
8: select a l from some clause $C \in \phi$
9: $\mathcal{I} = \text{DPLL}(\phi|_l, \mathcal{I} \cup \{l\})$
10: **if** $\mathcal{I} \neq$ Unsatisfiable **then**
11: **return** $\mathcal{I}$
12: **else**
13: $\mathcal{I} \leftarrow \text{DPLL}(\phi|_{\bar{l}}, \mathcal{I} \cup \{\bar{l}\})$
14: **return** $\mathcal{I}$
15: **end if**
16: **end if**


$\phi = \begin{Bmatrix} \{A,B\},\{B,C\},\{\neg A,\neg X,Y\} \\ \{\neg A,X,Z\},\{\neg A,\neg Y,Z\} \\ \{\neg A,X,\neg Z\},\{\neg A,\neg Y,\neg Z\} \end{Bmatrix}$
$\mathcal{I} = \{A\}$
$\mathcal{I} = \{\neg A\}$
$\phi|_A = \begin{Bmatrix} \{B,C\},\{\neg X,Y\} \\ \{X,Z\},\{\neg Y,Z\} \\ \{X,\neg Z\},\{\neg Y,\neg Z\} \end{Bmatrix}$
$\phi = \begin{Bmatrix} \{B\} \\ \{B,C\} \end{Bmatrix}$
$\mathcal{I} = \{\neg A, B\}$
$\{\}$
$\mathcal{I} = \{A,B\}$
$\mathcal{I} = \{A,\neg B\}$
$\phi|_{A,B} = \begin{Bmatrix} \{\neg X,Y\},\{X,Z\} \\ \{\neg Y,Z\},\{X,\neg Z\} \\ \{\neg Y,\neg Z\} \end{Bmatrix}$
$\phi|_{A,\neg B} = \begin{Bmatrix} \{C\},\{\neg X,Y\} \\ \{X,Z\},\{\neg Y,Z\} \\ \{X,\neg Z\},\{\neg Y,\neg Z\} \end{Bmatrix}$
$\mathcal{I} = \{A,B,X\}$
$\mathcal{I} = \{A,B,\neg X\}$
$\mathcal{I} = \{A,\neg B,C\}$
$\phi|_{A,B,X} = \begin{Bmatrix} \{Y\},\{\neg Y,Z\}, \\ \{\neg Y,\neg Z\} \end{Bmatrix}$
$\phi = \begin{Bmatrix} \{Z\},\{\neg Y,Z\} \\ \{\neg Z\},\{\neg Y,\neg Z\} \end{Bmatrix}$
$\phi|_{A,\neg B,C} = \begin{Bmatrix} \{\neg X,Y\},\{X,Z\} \\ \{\neg Y,Z\},\{X,\neg Z\}, \\ \{\neg Y,\neg Z\} \end{Bmatrix}$
$\mathcal{I} = \{A,B,X,Y\}$
$\mathcal{I} = \{A,B,\neg X,Z\}$
$\mathcal{I} = \{A,\neg B,C,X\}$
$\mathcal{I} = \{A,\neg B,C,\neg X\}$
$\phi|_{A,B,X,Y} = \{\{Z\},\{\neg Z\}\}$
$\phi = \{\{\},\{\neg Y\}\}$
$\phi|_{A,\neg B,C,X} = \begin{Bmatrix} \{Y\}, \\ \{\neg Y,Z\}, \\ \{\neg Y,\neg Z\} \end{Bmatrix}$
$\phi|_{A,\neg B,C,\neg X} = \begin{Bmatrix} \{Z\},\{\neg Z\} \\ \{\neg Y,Z\} \\ \{\neg Y,\neg Z\} \end{Bmatrix}$
$\mathcal{I} = \{A,B,X,Y,Z\}$
$\phi|_{A,B,X,Y,Z} = \{\{\}\}$
$\mathcal{I} = \{A,\neg B,C,X,Y\}$
$\mathcal{I} = \{A,\neg B,C,\neg X,Z\}$
$\phi|_{A,\neg B,C,X,Y} = \{\{Z\},\{\neg Z\}\}$
$\phi|_{A,\neg B,C,\neg X,Z} = \{\{\},\{\neg Y\}\}$
$\mathcal{I} = \{A,\neg B,C,X,Y,Z\}$
$\phi|_{A,\neg B,C,X,Y,Z} = \{\{\}\}$

Exercizes

Exercise 3:

Check the following facts via DPLL

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

Other examples

Exercise 4:

Check the following facts

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

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 $\mathcal{I}(x_5) = \mathit{true}$.

Translating CNF into MiniSat input format

Example

- $\{\{A, \neg B, \neg D\}, \{\neg A, \neg B, \neg C\}, \{\neg A, C, \neg D\}, \{\neg A, B, C\}\}$

```
c A -> 1, B -> 2, C -> 3, D -> 4
p cnf 4 4
1 -2 -4 0
-1 -2 -3 0
-1 3 -4 0
-1 2 3 0
```

Translating CNF into MiniSat input format

Example

- $\{\{A, B, C\}, \{\neg B, \neg C\}, \{\neg B, \neg A\}, \{\neg A, B, C\}, \{A, B\}$ $\{A, C\}, \{B, \neg A, \neg C\}\}$

```
c A -> 1, B -> 2, C -> 3
p CNF 3 7
1 2 3 0
-2 -3 0
-1 -2 0
-1 2 3 0
1 2 0
1 3 0
-1 2 -3 0
```

Translating CNF into MiniSat input format

exercise

Rewrite in MiniSat input format (DIMACS) the following set of clauses:

- $\{\{\neg A, B\}, \{\neg C, D\}, \{\neg E, \neg F\}, \{F, \neg E, \neg B\}\}$

Obtaining more than one assignment from MiniSat

- MiniSat searches for one assignment that satisfies a CNF, and if there is one, it is returned.
- Question: How can I obtain more than one assignment?
- Answer: Suppose that MiniSat $C_1, \ldots, C_n$ returns $l_1, \ldots, l_n$. To check if there is another assignment, different from $l_1, \ldots, l_n$ we can check if $C_1, \ldots, C_n \wedge \neg(l_1 \wedge \cdots \wedge l_n)$ is satisfiable
- Notice that $\neg(l_1 \wedge \cdots \wedge l_n)$ is the clauses $\neg l_1 \vee \cdots \vee \neg l_n$
- In practice we can rerun MiniSat on $C_1, \ldots, C_n, \{\neg l_1, \ldots, \neg l_n\}$

Satisfiability in Python

- install Python 3

```
$> brew install python
```

- Install `python-sat`

```
$> pip install python-sat
```

- Check if $\{\neg p, q\}, \{\neg q, r\}$ is consistent (encode p, q, and r in 1, 2, and 3).

```
> from pysat.solvers import Minisat22
> Gamma = Minisat22()
> Gamma.add_clause([-1,2])     # cnf for p -> q
> Gamma.add_clause([-2,3])     # cnf for $ -> r
> print(Gamma.solve())
True
> print(Gamma.get_model())
[-1, -2, -3]
> for m in Gamma.enum_models(): print(m)
[-1, -2, -3]
[-1, -2, 3]
[-1, 2, 3]
[1, 2, 3]
```