SAT Solving and Applications
Computation of Prime Implicants

Prof. Dr. Wolfgang Küchlin
Dipl.-Inform. Martin Rathgeber
Dr. Eray Gençay
Rouven Walter, M.Sc.

University of Tübingen

30 November 2017

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Contents: Prime Implicant Computation

Short Motivation

Minimizing DNF

- A formula D is in DNF if it is given as a disjunction of *terms*, i.e. $D = t_1 \vee \ldots \vee t_n$, where $t_i = \ell_{i_1} \wedge \ldots \wedge \ell_{i_k}$ and all ℓ_{i_j} are *literals*.
- Each term t_i is called an *implicant* of D, because it implies D, i.e. $t_i \models D$.
- DNF is a normal form, but it is not unique; in general, many different DNF representations exist for a given F; by $D = \mathtt{DNF}(F)$ we denote any DNF of F.
- A term $t \not\equiv \perp$ is a *prime implicant* of F if it is an implicant of F where every literal ℓ in t is *necessary*, i.e. if ℓ is missing from t then $t \not\models F$. In this sense prime implicants are minimal implicants.
- A *prime DNF* $D = \mathtt{PDNF}(F)$ contains only prime implicants of F. All terms in a $\mathtt{PDNF}(F)$ are minimal, but not all may be necessary to represent F.
- Term t_i is *necessary* in $D = \mathtt{DNF}(F)$, if $(D \setminus \{t_i\}) \not\equiv F$.
- $D = \mathtt{DNF}(F)$ is *minimal* if it is a prime DNF where every term is necessary.
- D is a *minimum* DNF representation of F, if no other DNF representation D' exists which has fewer literals. This minimum need not be unique, but obviously any minimum DNF must be a minimal prime DNF.

Examples

1. Let $F = (x \to y)$. There are 3 models: $m_1 = \{\bar{x}, y\}$, $m_2 = \{\bar{x}, \bar{y}\}$, $m_3 = \{x, y\}$. There are 2 prime implicants: $p_1 = \neg x$, $p_2 = y$.
2. Let $D = (x \wedge y) \vee (x \wedge z) \vee y \vee (y \wedge z)$. There are 2 prime implicants $p_1 = y$ and $p_2 = x \wedge z$ because $D \equiv (x \vee y) \wedge (y \vee z)$.
3. Let $F = \{(a \vee b), (a \vee c), (\neg d \vee \neg e \vee \neg f)\}$.
   - $\{a, b, c, d, e, \neg f\}$ is a model of F.
   - $a \wedge b \wedge c \wedge d \wedge e \wedge \neg f$ and $a \wedge c \wedge \neg d$ are implicants of F
   - $p_1 = a \wedge \neg d$, $p_2 = a \wedge \neg f$ and $p_3 = b \wedge c \wedge \neg f$ are some prime implicants of F.

Models and Implicants

Terms as Sets

A term $t = \ell_1 \wedge \ldots \wedge \ell_n$ can be identified with the set $\{\ell_1, \ldots, \ell_n\}$ of its literals. We are only interested in non-trivial implicants $t \not\equiv \perp$, i.e. in terms whose sets are *consistent* (have a model) because they do not contain complementary literals.

Terms and Assignments

A term t can be identified with the total assignment τ on vars(t) which satisfies t, where $\tau(v) = 1$ iff $v \in t$, and $\pi(v) = 0$ iff $\neg v \in t$.

We will often identify an implicant t with the assignment τ induced by t.

Formal Definition: Implicants and Prime Implicants

An *implicant* t of a formula φ is a consistent set of literals such that $t \models \varphi$, i.e. any assignment ν which satisfies t also satisfies φ. An implicant t of φ is a *prime implicant* if there is no proper subset t' of t which is also an implicant of φ.

The above assignment ν to vars(φ) with $\nu(t) = 1$ and $\nu(\varphi) = 1$ must be an extension of τ. Since $t \models \varphi$, *any* extension of τ to the remaining variables in φ satisfies φ. If τ is prime, then it assigns a minimal subset of vars(φ) s.th. $\nu(\varphi) = 1$.

Covers, Prime Implicants and BD-resolution

Note that, if $t_1 \models \varphi$ and $t_2 \models \varphi$, then $t_1 \vee t_2 \models \varphi$.

Definition (Cover)

A set of terms $S = \{t_1, \ldots, t_n\}$ *covers* a formula φ if $\varphi \equiv t_1 \vee \cdots \vee t_n$.

Note that any DNF of F covers F.

Recap: Prime Implicant Computation by BD-Resolution

- Any consistent term in any DNF of F is an implicant of F.
- Prime implicants of F can be computed by iterated BD-resolution between terms, starting from $D = \texttt{DNF}(F)$. A prime implicant p *absorbs* any implicant $t = p \wedge r$ by the absorption law $p \vee (p \wedge r) \equiv p$. Thus any term t containing a prime implicant can be deleted from D. In the end, all prime implicants are computed in this way and only prime implicants of F remain in D. Therefore the set of all prime implicants of F covers F.

BD-resolution often “explodes” on larger F, because there are too many implicants as intermediate results and even too many prime implicants in the end. In many applications such as formula minimization we only need a minimal prime DNF, i.e. some minimal set $D = \{p_1, \ldots, p_n\}$ of prime implicants which covers F.

SAT Solving and Implicants

Problem 1: Computing Prime Implicants by SAT Solving

- A prime implicant t of φ represents an assignment τ to a minimal subset of vars(φ) which satisfies φ.
- Any assignment ν which contains τ as a subset will satisfy φ. All variable assignments in ν which are not in τ are not essential (or "don't care") for the satisfaction of φ. For any such variable v, we "don't care" whether $v \mapsto 1$ or $v \mapsto 0$.
- Efficient CDCL SAT solvers always return *complete* assignments.
- **Task:** We must discover which variable assignments computed by the solver are necessary for the solution of F and which are unnecessary, or "don't care."

Problem 2: Covering a Formula by Prime Implicants using SAT Solving

- **Task**: Repeatedly compute prime implicants for F until F is covered.

Excursion: Quantified Boolean Formulas

Quantifiers

- $\exists x(P)$ There exists (an assignment to) x, such that P holds.
- $\forall x(P)$ For all (assignments to) x, formula P holds.

SAT

Is a given formula of propositional logic **satisfiable** or **not satisfiable**?

QBF

Is a given **fully quantified** Boolean formula **true** or **false**?

PQBF (partial QBF)

Given a **partially quantified** Boolean formula Q, compute an equivalent **quantifier-free** Boolean formula F.

Remark: SAT can be seen as an existential QBF problem: Is a fully existentially quantified formula true or false?

Excursion: Quantified Boolean Formulas

- In general, the result of a PQBF problem Q is a formula F in the remaining (or even fewer) variables.
- In the special case of QBF (and SAT), the result must be either $\top$ or $\bot$.

Example (QBF Problems)

$$\exists x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) \equiv \top$$

$$\forall x \exists y ((x \vee y) \wedge (\neg x \vee \neg y)) \equiv \top$$

$$\forall x \forall y ((x \vee y) \wedge (\neg x \vee \neg y)) \equiv \bot$$

Example (PQBF Problems)

$\varphi = \exists x \exists y \ (x \vee \neg w) \wedge (\neg x \vee z) \wedge (\neg z \vee y) \wedge (w \vee z)$

$$\varphi \equiv (\neg w \wedge z) \vee (w \wedge z) \equiv z$$

$\varphi = \exists x \forall y ((x \vee y \vee \neg u) \wedge (\neg x \vee \neg y \vee w))$

$$\varphi \equiv (\neg u \wedge \neg w) \vee (\neg u \wedge w) \vee (u \wedge w) \equiv \neg u \vee w$$

Excursion: Quantified Boolean Formulas

Quantifier Elimination (QE)

Given a PQBF formula Q, a *Quantifier Elimination (QE)* method computes an equivalent quantifier free formula F by eliminating from Q all quantified variables.

QE by Substitution

Let Q be a PQBF formula where x is *free* (not quantified).

- $\exists x : Q \equiv Q|_{x=\top} \vee Q|_{x=\bot}$
- $\forall x : Q \equiv Q|_{x=\top} \wedge Q|_{x=\bot}$

QE Method: Substitute and Simplify

Clearly, QE by Substitution "explodes" the formula. However, the formula can be simplified after each substitution, because a variable is replaced by a constant. Sometimes this is enough to keep the explosion in check.

Excursion: Quantified Boolean Formulas

Example (Substitute and Simplify)

$$
\begin{aligned}
\varphi &= \exists x \exists y\ (x \vee \neg w) \wedge (\neg x \vee z) \wedge (\neg z \vee y) \wedge (w \vee z) \\
QE(\varphi, x) = \varphi_x &= \varphi|_{x=\top} \vee \varphi|_{x=\bot} \\
&\equiv (\exists y\ (\top \vee \neg w) \wedge (\neg \top \vee z) \wedge (\neg z \vee y) \wedge (w \vee z))\ \vee \\
&\ (\exists y\ (\bot \vee \neg w) \wedge (\neg \bot \vee z) \wedge (\neg z \vee y) \wedge (w \vee z)) \\
&\equiv (\exists y\ (z) \wedge (\neg z \vee y) \wedge (w \vee z))\ \vee \\
&\ (\exists y\ (\neg w) \wedge (\neg z \vee y) \wedge (w \vee z)) \\
&\equiv (\exists y\ (z \wedge y) \wedge (w \vee z)) \vee (\exists y\ (\neg w \wedge z) \wedge (\neg z \vee y)). \\
QE(\varphi_x, y) = \varphi_{xy} &\equiv [((z \wedge \top) \wedge (w \vee z)) \vee ((\neg w \wedge z) \wedge (\neg z \vee \top))] \vee \\
&\ [((z \wedge \bot) \wedge (w \vee z)) \vee ((\neg w \wedge z) \wedge (\neg z \vee \bot))] \\
&\equiv [((z) \wedge (w \vee z)) \vee ((\neg w \wedge z))] \vee [((\neg w \wedge z) \wedge (\neg z))] \\
&\equiv [((z \wedge w)) \vee ((\neg w \wedge z))] \vee [((\neg w \wedge z) \wedge (\neg z))] \\
&\equiv (\neg w \wedge z) \vee (w \wedge z) \vee \bot \\
&\equiv z
\end{aligned}
$$

Implicants and Partial QBF

The following lemma lets us move from a partial assignment to (the variables of) φ associated with an implicant τ to a total assignment which satisfies a suitable restriction of φ.

Lemma

Let φ be a formula with $\mathcal{V} = \text{vars}(\varphi)$, and t a term with $\mathcal{T} = \text{vars}(t) \subseteq \text{vars}(\varphi)$. Let $\mathcal{R} = \mathcal{V} \setminus \mathcal{T}$, the rest of the variables in φ after $\mathcal{T}$ has been assigned. Then t is an implicant of φ iff the assignment τ associated with t satisfies the PQBF formula $\psi = \forall x \in \mathcal{R} : \varphi$. We have $\text{vars}(\psi) = \mathcal{T}$ and therefore τ is total on $\mathcal{T}$.

Beweis.

$\Rightarrow$: If t is an implicant of φ then every assignment on $\mathcal{V}$ containing t satisfies φ, no matter how variables in $\mathcal{R}$ are assigned. Hence, by definition, τ satisfies $\psi = \forall x \in \mathcal{W} : \varphi$.
$\Leftarrow$: If τ satisfies $\forall x \in \mathcal{R} : \varphi$, then by definition τ is a partial assignment such that *all* assignments containing τ satisfy φ.

□

Practical Algorithms: Implicant Reduction

The above lemma suggests a method for reducing (trimming) a given implicant t by a literal, yielding a shorter implicant. If no reduction is possible, the implicant must be prime. Note that we may start with the satisfying assignment produced by a SAT solver, which corresponds to an implicant.

Definition (Janota, Lynce, Marques-Silva: Rotatable Literal)

A literal ℓ in implicant t of φ is *rotatable* in φ if $(t \setminus \{\ell\}) \cup \{\neg\ell\}$ is also an implicant of φ.

Lemma (Implicant Reduction)

If literal ℓ in implicant t of φ is rotatable, then $t \setminus \{\ell\}$ is also an implicant of φ.
Proof: *$t \setminus \{\ell\}$ satisfies $\forall x : \varphi$, where x is the variable in ℓ.*

Simple evaluation of φ by the assignment associated with $(t \setminus \{\ell\}) \cup \{\neg\ell\}$ reveals whether ℓ is rotatable.

Alternatively, all clauses where ℓ occurs can be checked if they are only satisfied by t because of ℓ.

Unit Literals

A literal ℓ is *essential*, i.e. *not rotatable*, in an implicant t of a clause set $\mathcal{C}$ if there exists at least one clause which is only satisfied by t because of ℓ.

Definition (Janota, Lynce, Marques-Silva)

A literal ℓ is *unit* in $\mathcal{C}$ w.r.t. an implicant t if there is a clause $\omega \in \mathcal{C}$ such that $\omega \cap t = \{\ell\}$.

Note that implicant t may only give a partial assignment τ, and that a literal ℓ may be unit w.r.t. one implicant and not unit w.r.t. another implicant.

Lemma (Unit stability)

Let $\mathcal{C}$ be a clause set and let s and t be two implicants of $\mathcal{C}$ with $s \subset t$.

1. *If literal $\ell \in s$ is unit in $\mathcal{C}$ w.r.t. t, then it is also unit in $\mathcal{C}$ w.r.t. s.*
2. *A literal ℓ which is not unit in $\mathcal{C}$ w.r.t. t may become unit in $\mathcal{C}$ w.r.t. s.*

Proof

1. If $\omega \cap t = \{\ell\}$ and $\ell \in s \subset t$ then also $\omega \cap s = \{\ell\}$.
2. Example: x is not unit in $\{(x \vee y)\}$ w.r.t. $t = \{x, y\}$, but is unit w.r.t. $\{x\}$.

Implicant Reduction by Removal of Rotatable Literals

Proposition (Janota, Lynce, Marques-Silva)

A literal ℓ in implicant τ is *rotatable* in φ if neither ℓ nor $\neg\ell$ is unit in φ w.r.t. τ.

Proof: Since ℓ occurs in an implicant, $\neg\ell$ cannot be unit in any clause ω, because then τ would not satisfy ω. If ℓ is unit, there is a clause ω which is satisfied by τ only because of ℓ; therefore ℓ cannot be rotated. If ℓ is not unit, every clause remains satisfied if ℓ is rotated. If ℓ is rotatable, neither ℓ nor $\neg\ell$ can be unit.

Implicant Reduction

Algorithm 1: ImplicantReduction(φ, τ)

```
Input: Satisfiable formula $\varphi$ in CNF, implicant $\tau$ of $\varphi$
Output: Reduced implicant $\tau'$
$\tau' \leftarrow \tau$
foreach $\ell \in \tau'$ do
    if ($\ell$ is not unit in $\varphi$ w.r.t. $\tau'$) then
        $\tau' \leftarrow \tau' \setminus \{\ell\}$
```

Example and Discussion

In general there will be more than one prime implicant $\pi \subset \tau$. Implicant Reduction reveals one of them, depending on the selection sequence of the literals.

Example: Selection sequences in Implicant Reduction

Let $\varphi = \{(x \vee y), (y \vee z)\}$ and $\tau = \{x, y, z\}$.

1. Remove $\ell_1 = x$; now $\ell_2 = y$ is unit; remove $\ell_3 = z$. Result: $\pi = \{y\}$.
2. Remove $\ell_1 = y$; now $\ell_2 = x$ is unit; also $\ell_3 = z$ is unit. Result: $\pi = \{x, z\}$.

Literals in τ which correspond to unit-propagated variables in the solver are always unit literals w.r.t. τ. Due to the Stability Lemma, they remain unit for all subsets to which τ is reduced by Algorithm 1, and hence occur in every prime implicant which can be produced from τ. However, also variables which were selected by the solver as decision variables may in the end turn out to be unit w.r.t. the final τ.

Example: Decision variables and unit literals

Let $\varphi = \{(\neg x \vee y), (\neg y \vee z), (\neg z \vee \neg x)\}$.
Solve φ: Decisions: $x = 0$; $y = 0$; $z = 0$. Result: $\tau = \{\neg x, \neg y, \neg z\}$.
Both $\ell_1 = \neg x$ and $\ell_2 = \neg y$ are unit w.r.t. τ.

Abstract Computation of Prime Implicants

Abstract Algorithm PRIME (Déharbe, Fontaine, Le Berre, Mazure)

Algorithm 2: PRIME($\mathcal{C}, \tau_0, \pi_0$)

Input: Satisfiable formula φ as set $\mathcal{C}$ of clauses, implicant τ_0 of $\mathcal{C}$, subset π_0 of prime implicant π

Output: Prime implicant π

```
$\tau, \pi \leftarrow \tau_0, \pi_0$
while $\ell \in \tau \setminus \pi$ do
    if Required($\tau, \ell, \mathcal{C}$) then
        $\pi \leftarrow \pi \cup \{\ell\}$
    else
        $\tau \leftarrow \tau \setminus \{\ell\}$
return($\pi$)
```

- Some initial π_0 is easy to find, e.g. all unit-propagated literals in τ_0.
- In "Implicant Reduction," *Required*$(\tau, \ell, \mathcal{C})$ = "ℓ is unit in $\mathcal{C}$ w.r.t. τ".
- Problem is efficiency: Naive Implicant Reduction is in O($|\tau| * |\mathcal{C}|$).

Counter Based Prime Implicant Computation

Algorithm 3: PRIMEbyCounter($\mathcal{C}, \tau_0, \pi_0$)

Input: Satisfiable formula φ as set $\mathcal{C}$ of clauses, implicant τ_0 of $\mathcal{C}$, subset π_0 of prime implicant π. The literals in $\tau_0 \setminus \pi_0$ are candidates for inclusion in π.

Output: Prime implicant π

```
1 Initialize:
    • $\tau, \pi \leftarrow \tau_0, \pi_0$
    • For all $\ell \in \tau$ set up watch list $W(\ell)$ with all clauses in which $\ell$ occurs.
    • For all clauses $c$ set up counter $N[c]$ for counting the number of positive
      literals from $\tau$ in $c$; for all $c$ initialize $N[c] \leftarrow 0$.
2 Count positive literals in each $c$:
  For all $\ell \in \tau$ increase $N[c] = N[c] + 1$ for all $c$ in $W(\ell)$.
3 Process remaining candidates: for all $\ell \in (\tau \setminus \pi)$ do
    • if $\ell$ is unit in some $c$ then include $\ell$ in $\pi$:
      if $\exists c \in W(\ell) : N[c] = 1$ then $\pi \leftarrow \pi \cup \{\ell\}$, else
    • if $\ell$ is not unit remove $\ell$ from candidates $\tau$:
      $\tau \leftarrow \tau \setminus \{\ell\}$; for all $c \in W(\ell)$ do $N[c] = N[c] - 1$
4 Return $\pi$
```

Counter Based Prime Implicant Computation

- PRIME-by-Counter is from Déharbe, Fontaine, Le Berre, Mazure (2013).
- In Step 3, we can pick *any* $\ell \in (\tau \setminus \pi)$, because there are no more units: every remaining candidate is rotatable. Depending on our choice we may arrive at different prime implicants contained in τ_0.
- Complexity is in $\mathcal{O}(\sum_{c\in\mathcal{C}} |c|)$, i.e. linear in size of $\mathcal{C}$ provided the size of all watch lists is bounded by a constant independent of the size of $\mathcal{C}$.

New idea: Prime implicants using watched literals

$N[c]$ is only interesting if $N[c] = 1$: then some $\ell \in c$ has become unit.

- In each clause, watch two literals *satisfied* by τ. (We can reuse/adapt the 2-watched literals scheme of CDCL, starting from the final solver state.)
- Since τ is an implicant, at least one literal is satisfied by τ.
- If a second satisfied literal does not exist, the first one is unit and hence in π.
- If two (or more) satisfied literals exist, any one of them is rotatable.
- After a rotatable literal ℓ is removed from τ, we must remove it from the watch lists in all clauses c: either we find a replacement to watch in c, or the other literal watched in c is now unit and must go to π.

Prime Implicant Computation using Watched Literals

```
Algorithm 4: PRIMEbyWatches($\mathcal{C}, \tau_0, \pi_0, W$)
Input: Satisfiable formula $\varphi$ as set $\mathcal{C}$ of clauses as final solver state (including
       2-watched literals data structure), implicant $\tau_0$ of $\mathcal{C}$, subset $\pi_0$ of prime
       implicant $\pi$, set $W$ of all watch lists. The literals in $\tau_0 \setminus \pi_0$ are candidates
       for inclusion in $\pi$.
Output: Prime implicant $\pi$
  1 Initialize: replace unsatisfied by satisfied literals in watches.
      • $\tau, \pi \leftarrow \tau_0, \pi_0$
      • For all $\ell \in (\tau \setminus \pi)$ : Remove $\neg\ell$ from watched literals in all clauses by calling
        UnwatchAndPropagate($\neg\ell, \mathcal{C}, \tau, \pi, W$)
  2 Now every $\ell \in (\tau \setminus \pi)$ is rotatable:
    Pick any one, remove it from $\tau \setminus \pi$, and propagate the consequences:
    For all $\ell \in (\tau \setminus \pi)$ do: $\{\tau \leftarrow \tau \setminus \{\ell\}$; UnwatchAndPropagate($\ell, \mathcal{C}, \tau, \pi, W$)$\}$
  3 Return $\pi$
```

- Complexity of PRIMEbyWatches is in $\mathcal{O}(\sum_{c \in \mathcal{C}} |c|)$ (see below).

Prime Implicant Computation using Watched Literals

Algorithm 5: UnwatchAndPropagate($\ell, \mathcal{C}, \tau,$ **ref** π, **ref** W)

Input: Literal ℓ is removed from all watch lists in W and, if possible, replaced in the watch list for each c by a satisfied literal in c. If this is not possible, the other watched literal in c is added to π.

Output: W, and possibly π, updated as side effect

```
1 Replace $\ell$ in all watches:
  For all $c \in W(\ell)$ :
      • Is there a replacement for $\ell$ in $c$?
        If $\exists \ell' \in c \cap \tau : \ell' \notin W^{-1}(c)$ then remove $c$ from $W(\ell)$ and add $c$ to $W(\ell')$
      • If there is no replacement, the other watched literal in $c$ ist now unit in $c$:
        Else $\pi \leftarrow \pi \cup (W^{-1}(c) \setminus \{\ell\}$
```

- $W^{-1}(c)$ is the set of (satisfied) literals watched in c. In a usual implementation, these are the first two literals in c, so no extra data structure is required.
- In the course of PRIMEbyWatches, each clause is only scanned once, if we remember where we found the last *true* literal. To the left of this place all literals are *false* except for the first two (watched) literals.

Prime Implicant Enumeration

Prime Implicant Enumeration

This is the problem to compute the set $\Pi(\varphi)$ of *all* prime implicants of φ. Two approaches:

- Compute $\Pi(\varphi)$ by BD-resolution starting from any DNF of φ. Problems:
  - Very many resolvents are computed, $\Pi(\varphi)$ is only a small subset.
  - For large φ, already computing an equivalent DNF(φ) explodes (is impossible). Computing a satisfiability equivalent disjunct is not enough.
- Extend abstract algorithm PRIME. Problems:
  - Algorithm PRIME depends on the model τ. It can only produce prime implicants contained in τ (as subsets). So all models of φ must be enumerated.
  - For each model τ, the prime implicant produced by PRIME in general depends on the sequence of literal selections from τ. All selection sequences must be tried to produce all prime implicants which are subsets of τ.

Prime Implicant Cover Computation

Definition (Prime Implicant Cover)

For a formula φ, a *Prime Implicant Cover* of φ is a set $\Psi(\varphi) = \{\pi_1, \ldots, \pi_n\}$ of prime implicants which *covers* φ in the sense that $\pi_1 \vee \cdots \vee \pi_n \equiv \varphi$. (The cover is empty if φ is unsatisfiable.)

Prime Implicant Cover Computation

A Prime Implicant Cover of φ can be computed similar to model enumeration:

1. $\Psi \leftarrow \emptyset$.
2. Compute a model $\tau \leftarrow \mathsf{SAT}(\varphi)$. If $\tau = \emptyset$ then return Ψ.
3. Compute a prime implicant $\pi \leftarrow \mathsf{PRIME}(\varphi, \tau)$ and let $\Psi \leftarrow \Psi \cup \{\pi\}$.
4. Block π in φ by $\varphi \leftarrow \varphi \wedge \neg\pi$.
5. Go to Step 2.

The cover computed above is arbitrary. In general there are many covers. The problem to find the minimum of all covers is a famous NP-complete problem.

Applications: MaxSAT

To compute MaxSAT(φ), add a blocking variable $b_i \in B$ to each clause in φ and use a SAT solver to find models ν of $\varphi' \wedge \mathrm{CNF}(\sum b_i < k)$ with decreasing $k = |\nu \cap B|$ until the formula becomes inconsistent.

Model ν may contain unnecessary blocking variables, so work with prime implicants to achieve a tighter bound k for a faster convergence to the minimum.

Example

- Let $\varphi = \{(\neg x \vee y), (\neg x \vee z), (x), (\neg y), (x \vee y), (\neg z \vee y)\}$.
- Solve
  $\varphi' = \{(b_1 \vee \neg x \vee y), (b_2 \vee \neg x \vee z), (b_3 \vee x), (b_4 \vee \neg y), (b_5 \vee x \vee y), (b_6 \vee \neg z \vee y)\}$
- In SAT(φ') $= \{x, y, z, b_1, b_2, b_3, b_4, \neg b_5, \neg b_6\}$, blocking variables b_1, b_2, b_3 are (unnecessarily) satisfied although their clauses are already satisfied.

Applications: Backbones

Backbone Computation

- The *backbone* B of a formula φ is the set of all literals which are *true* in all models of φ, i.e. $B(\varphi) = \cap_{\nu \in \mathcal{M}(\varphi)}$.
- Note that a prime implicant π is the set of all literals which are *true* in all models which are supersets of π.
- Hence the backbone of φ is the set of all literals which are *true* in all prime implicants Π of φ, i.e. the backbone B is the intersection $B(\varphi) = \cap_{\pi \in \Pi(\varphi)}$ of all prime implicants. Usually, $\Pi(\varphi)$ will be smaller than the set of all models $\mathcal{M}(\varphi)$.
- Clearly, it is also sufficient to take the intersection of any subset Ψ of $\Pi(\varphi)$ which *covers* φ (i.e. the disjunction of all $\pi \in \Psi(\varphi)$ is equivalent to φ). This set will be even smaller.
- This yields an algorithm for computing the backbone $B(\varphi)$: Enumerate prime implicants of φ until φ is covered and take their intersection.

Applications: Explanation of SAT, Formula Minimization

In practical applications, it is essential that the results of SAT-Solving can be explained, i.e. justified to the user independent of the solver's correctness. Such a result, which can be verified independently, is also called a *certificate*. A result of "UNSAT" can be explained by a resolution proof. A first certificate of "SAT" is the computed model. However, there may be very many literals in the model and many of those may be unnecessary, i.e. irrelevant to the result "SAT".

Explanation of Satisfiability

There are various possible certificates for the satisfiability of φ:

- The model $\tau \leftarrow \mathrm{SAT}(\varphi)$. This is a "single point of satisfaction."
- Some prime implicant π contained in τ. Then π represents a whole *set* of models of φ. This is a small "area of satisfaction."
- A prime implicant cover of φ. This is a short and comprehensive representation of *all* models of φ, the entire "area of satisfaction."
- A minimum prime implicant cover of φ. This is a "shortest comprehensive representation" of φ. The intersection of all cover terms is the backbone.
- From a prime implicant cover of φ (which is a DNF of φ) a parenthesized formula can be produced by extracting common subterms.

Example: Formula Minimization

The following is a condition, in Daimler syntax, for the use of a part in Daimler's A-class automobiles.

(P39+P84+510/P39+P84+512/P39+P84+514/P39+P84+523/P39+P84+526/
P39+P84+527/P39+510+950/P39+512+950/P39+514+950/P39+523+950/
P39+526+950/P39+527+950)+-516+-(P27+510)+-(P27+512)+-(P27+514)+
-(P27+523)+-(P27+526)+-(P27+527)+-(P59+510)+-(P59+512)+
-(P59+514)+-(P59+523)+-(P59+526)+-(P59+527)+-(P34+510)+
-(P34+512)+-(P34+514)+-(P34+523)+-(P34+526)+-(P34+527)+
-(P37+510)+-(P37+512)+-(P37+523)+-(P37+527)+-(140+806)+-M133

A DNF produced by a good algorithm has 784 terms, while there is a DNF with 24 terms using only prime implicants.

The latter minimal prime DNF can be automatically converted to the following equivalent parenthesized formula:
P39+-516+-M133+-P27+-P34+-P59+(P84/950)+
(-510+-512+-523+-527+
(-140/-806)+(526/514)/-P37+(510/512/514/523/526/527)+(-806/-140))

Literature

Vasco M. Manquinho, Paulo F. Flores, João P. Marques-Silva, Arlindo L. Oliveira. Prime Implicant Computation Using Satisfiability Algorithms. In: *ICTAI'97*, 1997.

Mikolás Janota, Inês Lynce, João P. Marques-Silva. Algorithms for Computing Backbones of Propositional Formulae. *AI Communications*, 2012.

David Déharbe, Pascal Fontaine, Daniel Le Berre, Bertrand Mazure. Computing prime implicants. In: *Formal Methods in Computer-Aided Design (FMCAD)*, 2013.