SAT Solving and Applications

Computation of Backbones

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

University of Tübingen

30 December 2017

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Contents: Backbone Computation

Analysis of Configuration Constraints

Definition: Backbone

The backbone B_φ of a propositional formula φ is the set of literals which must be *true* in all models of φ.

Example (Configuration)

Let φ contain the set of constraints for the configuration of a system. Each variable represents a configuration option, the valid configurations are exactly the solutions (models) of φ. Option v is *necessary* in each configuration, if $v = 1$ in each model, i.e. if $v \in B_\varphi$.

Definition: Necessary, inadmissible and optional variables

- An option v in φ is *necessary (mandatory)*, if $v = 1$ in each model of φ.
- An option v in φ is *inadmissible (impossible)*, if $v = 0$ in each model of φ.
- An option v in φ is *possible (optional)*, if v is neither necessary nor impossible.

Analysis of Configuration Constraints

Proposition 1: Options and Models

- Option v is *necessary* in φ, if it *must be true* in (all models of) φ.
- v is *not necessary* in φ, if it *can be false* in (some model of) φ.
- v is *inadmissible* in φ, if it *must be false* in (all models of) φ.
- v is *not inadmissible* in φ, if it *can be true* in (some model of) φ.
- v is *optional* in φ, if it *can be true or false* in (a model of) φ.

Example (Configuration analysis)

If we set $v = 1$ in φ, which other variables/options become necessary? Inadmissible?

Proposition 2: Backbones and Options

Let B be the backbone of φ.

- Option v is *necessary* iff $v \in B$.
- Option v is *inadmissible* iff $\neg v \in B$.
- Option v is *optional* iff $v \notin B$ and $\neg v \notin B$.

Models and Implicants

Definition: Implicants and Prime Implicants

A set of literals is *consistent* if it does not contain any complementary literals. An *implicant* ν of a formula φ is a consistent set of literals such that every assignment μ which satisfies ν also satisfies φ. An implicant ν of φ is a *prime implicant* if there is no proper subset ν' of ν which is also an implicant of φ.

An implicant ν can be identified with the partial assignment π where $\pi(v) = 1$ iff $v \in \nu$, and $\pi(v) = 0$ iff $\neg v \in \nu$. Let $\mathcal{B}^* = \{0, 1, *\}$. Then ν can be identified with the (total) assignment π^* on $\mathit{vars}(\varphi)$, where $\pi^*(v) = \pi(v)$ iff $v \in \nu$ or $\neg v \in \nu$, and $\pi^*(v) = *$ otherwise. Here, $*$ denotes an "unassigned" or "don't care" value.

Remark

Current SAT solvers always return complete satisfying assignments (models) for efficiency reasons, but it is easily possible to return only partial satisfying assignments. Both correspond to implicants.

Implicants and Terms, Implicant Covers and DNF

Implicants as Terms

A set $\nu = \{\ell_1, \ldots, \ell_n\}$ of literals can be identified with the term $t_\nu = \{\ell_1 \wedge \cdots \wedge \ell_n\}$. If ν is an implicant of φ, then $t_\nu \models \varphi$. If μ is also an implicant of φ, then $t_\mu \vee t_\nu \models \varphi$. If $\varphi \equiv t_1 \vee \cdots \vee t_n$, then, clearly, all t_i are implicants of φ. From now on, we will identify implicants with their corresponding satisfying assignments as well as with their corresponding terms.

Definition (Implicant Cover)

Let $\mathcal{S} = \{\nu_1, \ldots, \nu_n\}$ be a set of implicants of φ. Then $\mathcal{S}$ *covers* (the models of) φ if for every model μ of φ there is an implicant $\nu \in \mathcal{S}$ which is satisfied by μ. I.e., every model μ has a subset which is a $\nu \in \mathcal{S}$, and every $\nu \in \mathcal{S}$ covers those models which have ν as a subset.

Lemma (Implicant Cover and DNF)

Let $\mathcal{S} = \{t_1, \ldots, t_n\}$ be a set of implicants covering φ. Then $\varphi \equiv t_1 \vee \cdots \vee t_n$.

Proof: $\Rightarrow$: Let μ be a model, $\mu \models \varphi$. Then there is a t_i such that $\mu \models t_i$.
$\Leftarrow$: Trivial, because each t_i is an implicant $\square$

Computing Implicant Covers

As we have seen, any DNF of φ yields an implicant cover. But we can also use a SAT Solver to enumerate models of φ until φ is covered.

Implicant Cover by SAT

Algorithm 1: COVER(φ)

Input: Formula φ as set of clauses
Output: Implicant cover $\mathcal{S}$
$\mathcal{S}, \varphi' \leftarrow \emptyset, \varphi$
$\nu \leftarrow \texttt{SAT}(\varphi')$
while $\nu \neq \emptyset$ **do**
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\nu\}$
  $\varphi' \leftarrow \varphi' \cup \neg\nu$
  $\nu \leftarrow \texttt{SAT}(\varphi')$
return $(\mathcal{S})$

In each iteration, $\neg\nu = \{\neg\ell : \ell \in \nu\}$ is added to φ' as an additional clause until φ' has no more implicants. So far, COVER is not efficient, because it produces a cover set consisting of models, which corresponds to a DNF of Minterms.

Implicant Reduction by Rotatable Literals

Algorithm COVER becomes interesting and useful when we cover φ by implicants which are proper subsets of models, where the smallest subsets are prime implicants. Then each implicant covers all models in which it occurs as a subset. Correspondingly, each blocking clause then blocks the entire *set* of models which is covered, instead of only a *single* model. (Shorter blocking clauses yield more powerful constraints.) Thus we may obtain much smaller cover sets for φ.

Definition: Rotatable Literal

Literal ℓ in implicant ν is *rotatable* in φ if $\nu \setminus \{\ell\} \cup \{\neg\ell\}$ is also an implicant of φ.

Lemma (Implicant Reduction by Rotatable Literals)

If literal ℓ in implicant ν is rotatable in φ then $\nu \setminus \{\ell\}$ is also an implicant of φ.

Proof: If $\nu' = \nu \setminus \{\ell\}$ is *not* an implicant of φ, then there must be an assignment τ which satisfies ν' but does not satisfy φ. But τ must either satisfy ℓ or $\neg\ell$. In both cases τ satisfies an implicant of φ and hence satisfies φ $\square$

Implicant Reduction by Unit Literals

For $\ell \in \nu$, evaluation of φ by $\nu' = \nu \setminus \{\ell\} \cup \{\neg\ell\}$ detects whether ℓ is rotatable.

But we can also detect *all* rotatable literals of ν in φ in a single pass over the clauses of φ by looking for *unit literals*:

Definition: Unit Literal

Literal ℓ is *unit* in φ w.r.t. ν if there is a clause $\omega \in \varphi$ such that $\omega \cap \nu = \{\ell\}$.

Lemma (Janota, Lynce, Marques-Silva)

A literal ℓ in implicant ν is rotatable in φ iff neither ℓ nor $\neg\ell$ is unit in φ

Proof: If neither ℓ nor $\neg\ell$ are unit in φ, every clause remains satisfied even if ℓ is rotated. Conversely, if φ remains satisfied by rotating ℓ in ν, then ℓ and $\neg\ell$ cannot be unit in any clause of φ $\square$

Clearly, if ℓ occurs in an implicant ν, then $\neg\ell$ cannot be unit w.r.t. ν. So it is sufficient to check whether ℓ is unit w.r.t. ν. If not, then ℓ must be rotatable. But if it is unit, then it is *necessary* (or *required*) in ν in the sense that there is a clause $\omega \in \varphi$ which is no longer satisfied by ν if we remove ℓ from ν.

Implicant Reduction

Definition: Required Literal

Literal ℓ is *required* (or *necessary*) in assignment ν for φ iff ν is an implicant and ℓ is unit in φ w.r.t. ν.

Implicant Reduction

Algorithm 2: ImplicantReduction(φ, ν)

Input: Satisfiable formula φ in CNF, implicant ν of φ
Output: Reduced implicant ν'

$\nu' \leftarrow \nu$
foreach $\ell \in \nu'$ **do**
  if *(ℓ is not required in ν' for φ)* **then**
    $\nu' \leftarrow \nu' \setminus \{\ell\}$

Note that whether ℓ is required or not depends on ν' (and on φ). In each iteration, any one of the literals which are *not* required can be removed, but then a literal may become unit which was not required before.

Abstract Computation of Prime Implicants

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

Algorithm 3: PRIME$(\mathcal{C}, \nu_0, \pi_0)$

Input: Satisfiable formula φ as set $\mathcal{C}$ of clauses, implicant ν_0 of $\mathcal{C}$, subset π_0 of prime implicant π

Output: Prime implicant π

$\nu, \pi \leftarrow \nu_0, \pi_0$
while $\ell \in \nu \setminus \pi$ **do**
  if *Required*$(\nu, \ell, \mathcal{C})$ **then**
    $\pi \leftarrow \pi \cup \{\ell\}$
  else
    $\nu \leftarrow \nu \setminus \{\ell\}$
return(π)

- Some initial π_0 is easy to find, e.g. propagated literals in ν_0.
- For "*Required*$(\nu, \ell, \mathcal{C})$" we may use "ℓ is unit in $\mathcal{C}$ w.r.t. ν" or we may use "ℓ is not rotatable in ν w.r.t. $\mathcal{C}$".
- Problem is efficiency: Naive Implicant Reduction is in $O(|\nu| * |\mathcal{C}|)$.

Computing Prime Implicant Covers

Instead of covering φ with models, we can now cover φ with *sets* of models, each represented by a prime implicant. In general, the cover will be (much) smaller.

Prime Implicant Cover by SAT

Algorithm 4: PrimeCOVER(φ)

Input: Formula φ as set of clauses
Output: Prime Implicant cover $\mathcal{S}$

$\mathcal{S}, \varphi' \leftarrow \emptyset, \varphi$
$\nu \leftarrow \texttt{SAT}(\varphi')$
while $\nu \neq \emptyset$ **do**
  $\pi \leftarrow \texttt{PRIME}(\varphi, \nu, \emptyset)$
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{\pi\}$
  $\varphi' \leftarrow \varphi' \cup \neg\pi$
  $\nu \leftarrow \texttt{SAT}(\varphi')$
return $(\mathcal{S})$

Implicants and Backbones

Proposition 3: Backbone literals and implicants

Every implicant of φ must contain all backbone literals of φ.

Proof: Let ν be an implicant of φ without literal ℓ. Hence $\nu \cup \neg\ell$ is also an implicant (because *every* assignment satisfying ν also satisfies φ). But the corresponding assignment falsifies ℓ. Hence ℓ is not a backbone literal. $\square$

Proposition 4: Backbone and implicants

Every implicant of φ is a superset of the backbone of φ, and the backbone is the intersection of all implicants.

Proposition 5: Backbone and implicant cover

Let $\mathcal{I}$ be a set of implicants that *cover* all models of φ, i.e. every model μ satisfies some $\nu \in \mathcal{I}$. Then ℓ is a backbone literal iff ℓ occurs in all $\nu \in \mathcal{I}$.

Proof: $\Rightarrow$ If ℓ is in the backbone, then it occurs in all implicants.
$\Leftarrow$ If ℓ occurs in all $\nu \in \mathcal{I}$ and $\mathcal{I}$ covers all models, then ℓ is satisfied by all models.

Global Ideas for Backbone Computation Methods

We need to compute the intersection of all implicants of φ. Since every implicant is contained in a model, we could intersect all models, but there are too many.

1. For each variable, we can test whether there is any model in which it occurs positively, or negatively, respectively. This needs $2 \times n$ calls to SAT.
2. We can also start with a model m and test for each literal $\ell \in m$ whether it can also occur negatively in some other model. This needs only $1 + n$ calls to SAT. In addition, we can compute the intersection $\mathcal{I}$ of these models as we go, and only test the literals remaining in $\mathcal{I}$ (instead of all literals in m).
3. Instead of intersecting models we can intersect prime implicants. For each prime implicant π, *every* extension by any combination of the remaining literals, whether positive or negative, is an implicant. Hence no remaining literal can be in the backbone (i.e., the backbone is also the intersection of all *prime* implicants). We may stop as soon as the prime implicants cover φ.
4. A prime implicant contained in a model m can be computed by removing rotatable literals from m. By a very similar algorithm we can at the same cost compute the *intersection* $\mathcal{I}$ of all prime implicants contained in m. Again we test for each literal $\ell \in \mathcal{I}$ whether there is some other model m' with $\neg\ell \in m'$, and if so, we intersect $\mathcal{I}$ with the intersection of all prime implicants in m'. We stop the iteration after all literals remaining in $\mathcal{I}$ have been tested.

Practical Algorithms

There are two basic approaches for the computation of a backbone β of φ:

1. **Incremental** (from below): Start with an underestimation $\emptyset = \beta^* \subseteq \beta$ and check for each $x \in \mathit{vars}(\varphi)$ whether $\{x\}$ or $\{\neg x\}$ must be included in β^*.
2. **Refinement** (from above): Start with a model $\nu = \beta^*$ as overestimation $\beta^* \supseteq \beta$ and check for each $\ell \in \beta^*$ whether it must be removed from β^*.

Our incremental algorithm uses a generalization of the notion of "rotatable literal in φ w.r.t. ν": A literal ℓ is (globally) rotatable in φ, if there are implicants μ and ν of φ such that $\ell \in \mu$ and $\neg\ell \in \nu$. Then a literal is in the backbone *iff* it is not rotatable in φ. Since μ and ν may be different, a SAT Solver is needed in this case.

The first refinement algorithm computes the backbone as the intersection of the prime implicants in a cover of φ. An improvement lets us advance from computing a sequence of prime implicants to computing a sequence of intersections of the prime implicants contained in a model. Another variant directs the solver to compute prime implicants which also "trim" at least one literal from the backbone estimate until no more literals can be removed (or φ is covered).

Finally, an additional "core guided" refinement approach attempts to trim entire subsets of literals from a backbone estimate. The information contained in minimal unsatisfiable cores is used for guidance towards a removable subset.

Options, Implicants, and Backbones

Proposition 6: Options and implicants

Let φ be a satisfiable formula with backbone B. Let v be a variable of φ, μ any implicant of φ, and μ^* its associated complete assignment.

1. if $\mu^*(v) = 1$, then $\neg v \notin B$ (v is true in some model of φ, hence is not inadmissible and is possibly necessary).
2. if $\mu^*(v) = 0$, then $v \notin B$ (v is false in some model of φ, hence is not necessary and is possibly inadmissible).
3. if $\mu^*(v) = *$, then $v \notin B$ and $\neg v \notin B$ (v is false in some model and true in some model of φ, hence v is optional).

For configuration analysis, it is therefore convenient to define the following sets:

- Let $P_\varphi = \{v \in \mathit{vars}(\varphi) : v \in B\}$ (positive in all models: necessary)
- Let $P^*_\varphi = \{v \in \mathit{vars}(\varphi) : \neg v \notin B\}$ (positive in some model)
- Let $N_\varphi = \{v \in \mathit{vars}(\varphi) : \neg v \in B\}$ (negative in all models: inadmissible)
- Let $N^*_\varphi = \{v \in \mathit{vars}(\varphi) : v \notin B\}$ (negative in some model)
- Let $O_\varphi = \{v \in \mathit{vars}(\varphi) : v \notin P_\varphi \cup N_\varphi\}$ (not in B: optional)
- Let $B^*_\varphi = \{v \in \mathit{vars}(\varphi) : v \in P_\varphi \cup N_\varphi\}$ (in some polarity in B)

Iterative Testing of Candidate Literals

In the following we assume for generality that a call of $\nu \leftarrow \mathrm{SAT}(\varphi)$ returns an implicant of φ, or else an empty set if φ is unsatisfiable.
Clearly, for the backbone B_φ we have $B_\varphi = P_\varphi \cup \{\neg v : v \in N_\varphi\}$

Basic iterative testing [Kaiser & Küchlin 2001]

Algorithm 5: Basic(φ)

Input: Satisfiable formula φ in CNF
Output: Sets P_φ and N_φ
$P_\varphi \leftarrow \emptyset$
$N_\varphi \leftarrow \emptyset$
foreach $v \in \mathrm{vars}(\varphi)$ **do**
  if $\mathrm{SAT}(\varphi \cup v) = \emptyset$ **then**
    $N_\varphi \leftarrow N_\varphi \cup v$
  if $\mathrm{SAT}(\varphi \cup \neg v) = \emptyset$ **then**
    $P_\varphi \leftarrow P_\varphi \cup v$
return P_φ, N_φ

Iterative Testing and Filtering

Algorithm Basic(φ) takes exactly $2 * |\mathrm{vars}(\varphi)|$ calls to SAT. It can be improved in practice by several observations:

- Each backbone literal ℓ can be added to φ as a unit clause $\{\ell\}$ to enable formula simplification by unit propagation.
- Each unit clause derived after adding to φ a unit clause $\{\ell\}$ with backbone literal ℓ again contains a backbone literal.
- Computed models (implicants) may be analyzed to make use of Propositions 6 and 7.

Proposition 7: Filtering

Let $v \in \mathrm{vars}(\varphi)$ and ν any implicant of φ.

- if $v \in \nu$ (hence $v \in P^*_\varphi$) and $v \in N^*_\varphi$ then $v \in O_\varphi$ (v is positive in some model and negative in some model).
- if $\neg v \in \nu$ (hence $v \in N^*_\varphi$) and $v \in P^*_\varphi$ then $v \in O_\varphi$ (v is negative in some model and positive in some model).
- if $v \in O_\varphi$ then $v \notin B$ and $\neg v \notin B$, i.e. $v \notin B^*$

Iterative testing with filtering [Kaiser & Küchlin 2001]

Algorithm 6: Filter(φ)

Input: Satisfiable formula φ in CNF
Output: Sets P and N

$P \leftarrow \emptyset$, $P^* \leftarrow \emptyset$, $N \leftarrow \emptyset$, $N^* \leftarrow \emptyset$
foreach $v \in \mathrm{vars}(\varphi)$ **do**
  if $v \notin N \cup P \cup P^*$ **then** // $\neg v$ is candidate for backbone
    $\nu \leftarrow \mathrm{SAT}(\varphi \cup v)$
    if $\nu = \emptyset$ **then** // $v \in N$
      $\nu' \leftarrow \mathrm{ImpliedUnits}(\neg v, \varphi)$, $(P,N) \leftarrow \mathrm{Update}(P,N,\nu')$, $\varphi \leftarrow \varphi \cup \nu'$
    else // filter P^*, N^* by model ν
      $(P^*, N^*) \leftarrow \mathrm{Filter}(\nu, P^*, N^*)$
  if $v \notin P \cup N \cup N^*$ **then** // v is candidate for backbone
    $\nu \leftarrow \mathrm{SAT}(\varphi \cup \neg v)$
    if $\nu = \emptyset$ **then** // $v \in P$
      $\nu' \leftarrow \mathrm{ImpliedUnits}(v, \varphi)$, $(P,N) \leftarrow \mathrm{Update}(P,N,\nu')$, $\varphi \leftarrow \varphi \cup \nu'$
    else // filter P^*, N^* by model ν
      $(P^*, N^*) \leftarrow \mathrm{Filter}(\nu, P^*, N^*)$

Subalgorithms of Filter()

$\nu' \leftarrow \mathrm{ImpliedUnits}(\ell, \varphi)$

If ℓ is a backbone literal, then all unit clauses implied by ℓ also contain backbone literals. For efficiency, ImpliedUnits() only computes the subset ν' derivable by Unit Propagation, including ℓ itself.

$(P, N) \leftarrow \mathrm{Update}(P, N, \nu')$

Positive variables in ν' are added to P, negative variables are added to N.

$(P^*, N^*) \leftarrow \mathrm{Filter}(\nu, P^*, N^*)$

Implicant ν is used to update our knowledge about variables occurring positive, respectively negative, in some model.

- All variables occurring positive in ν are added to P^*, all variables occurring negative are added to N^*. (This may include backbone variables already contained in P or N.)
- All variables in vars(φ) which do not occur in ν are optional and are added to both P^* and N^*.

Refinement by Trimming Backbone Estimates

The backbone β of φ is the intersection of all implicants of φ. Refinement algorithms start with an overapproximation β^* of β and refine it by removing literals. In the following algorithm, we direct the SAT Solver to compute a new implicant which contains the negation of at least one literal in the backbone estimate β^*. If this is no longer possible, we have reached the backbone $\beta = \beta^*$.

Backbone Refinement by Implicant Intersection

Algorithm 7: ImplicantIntersection(φ)

Input: Formula φ as set of clauses
Output: Backbone β of φ

$\nu \leftarrow \texttt{SAT}(\varphi)$
$\beta^* \leftarrow \nu$
while $\nu \neq \emptyset$ **do**
  $\beta^* \leftarrow \beta^* \cap \nu$
  $\varphi \leftarrow \varphi \cup \neg\beta^*$
  $\nu \leftarrow \texttt{SAT}(\varphi)$
return (β^*)

Refinement by Trimming Backbone Estimates

It turns out that Algorithm 7 is slow in practice because the SAT problems are hard to solve. It can be improved by removing rotatable literals from each implicant, because if a literal is rotatable, then there is also an implicant containing the rotated literal.

Hence, instead of computing models we can also compute prime implicants by calling Algorithm PRIME on the models. We can stop as soon as we have computed a prime implicant cover of φ, because every implicant has a prime implicant from the cover as a subset.

Morevoer, there is an easy enhancement of algorithm PRIME which efficiently computes the intersection of *all* prime implicants contained in each model, so we need fewer additional intersection computations.

Finally, instead of flipping at least one variable at a time we can try to flip entire chunks of variables, using *core guided* algorithms.

Backbones as the Intersection of a Prime Implicant Cover

Iterative Backbone Refinement by Prime Implicant Intersection

Algorithm 8: PrimeImplicantIntersection(φ)

Input: Formula φ as set of clauses
Output: Backbone β of φ

$\varphi' \leftarrow \varphi$
$\nu \leftarrow \texttt{SAT}(\varphi')$
$\beta \leftarrow \nu$
while $\nu \neq \emptyset$ **do**
  $\nu' = \texttt{PRIME}(\varphi, \nu, \emptyset)$
  $\beta \leftarrow \beta \cap \nu'$
  $\varphi' \leftarrow \varphi' \cup \neg\nu'$
  $\nu \leftarrow \texttt{SAT}(\varphi')$
return (β)

The refinement process stops as soon as the entire φ has been "covered" by the computed prime implicants. At that moment, the backbone consists of the intersection of *all* prime implicants.

Intersection of All Prime Implicants of a Model

In general, an implicant ν may contain any number of different prime implicants. Depending on the sequence in which rotatable literals are removed from ν, algorithm PRIME will produce any of the contained prime implicants.

Suppose the literals x and y are both rotatable (i.e. not unit) w.r.t. ν. Then there must be a prime implicant without x, and there must be a prime implicant without y (maybe the same one, maybe different ones). It follows that the intersection of the prime implicants contained in ν does not contain either x or y. (Note that the intersection of two different prime implicants cannot be a prime implicant itself.)

Let algorithm `PRIME_INTER`(φ, ν) compute the intersection of all prime implicants of ν by removing from ν *all* rotatable literals at once (retaining only those that are unit), instead of just one at a time. Note that the result will in general not be a prime implicant.

Then we get a new algorithm for backbone computation by using `PRIME_INTER` instead of `PRIME` in Algorithm 8.

Unsatisfiable Cores

Definition: Unsatisfiable Core

Let φ be a set of clauses.

- An *unsatisfiable core (unsat core, UC)* of φ is an unsatisfiable subset $\varphi' \subseteq \varphi$.
- An unsatisfiable core φ_c is a *minimal unsatisfiable core (MUC)* if every proper subset $\varphi'_c \subsetneq \varphi_c$ is satisfiable.

Example

$\varphi = \{\{y\}, \{x\}, \{y, \neg z\}, \{\neg y\}, \{\neg x, z\}\}$

- φ is an unsat core, but not a MUC (e.g. $\{\{y\}, \{\neg y\}\}$ is unsatisfiable)
- $\{\{y\}, \{\neg y\}\}$ is a (global) MUC.
- $\{\{x\}, \{y, \neg z\}, \{\neg y\}, \{\neg x, z\}\}$ is a (locally) minimal unsat core

- A minimal unsat core need not be the (globally) smallest MUC
- An unsat core may contain more than one MUC.
- Current SAT Solvers can produce an UC at about 10% overhead. UC information is used by *Core Guided Algorithms*, e.g. to provide proofs of unsatisfiability, for MaxSAT computation or for backbone computation.

Core-Guided Variable Flips

So far, we test for each literal ℓ in an implicant ν whether it can be flipped. In the following, we try to flip all (or at least some number of) literals simultaneously.

Proposition: Flipping chunks of literals

Let ν be an implicant of φ. Let $\overline{\nu} = \{\ell : \neg\ell \in \nu\}$. If $\overline{\nu}$ is also an implicant, then all variables in ν are optional. This can be tested by calling SAT($\varphi \cup \overline{\nu}$).

Usually, not all variables can be flipped at the same time. But the Unsat Core returned by the SAT Solver will often contain a proper subset of $\overline{\nu}$ which prevents the simultaneous flip. A new flip of the remaining literals can be tried after this subset is removed. After removing all unsat cores from the candidate set for flipping, two cases remain:

- either the flipping candidate becomes empty so that no subset of the backbone estimate can be simultaneously flipped and removed.
- or all literals remaining in the candidate can be flipped and removed from the backbone estimate. (The refinement process may now be iterated on the refined estimate, because the unsat cores of the past iteration may not have been minimal and smaller cores may now expose further flippable variables.)

Another algorithm must now reduce the refined estimate to the true backbone.

Core-Guided filtering of backbone estimate

Algorithm 9: Core-Guided Filter(β^*)

Input: Implicant β^* of φ as a superset of the backbone β
Output: Refined backbone estimate β^*

```
1 ν ← β*                                          /* Flipping candidate ν */
2 ν̄ = {ℓ : ¬ℓ ∈ ν}                                /* Try to flip all literals in ν */
3 (μ, core) ← SAT(φ ∪ ν̄)                          /* Compute new implicant μ or unsat core */
4 while ν ≠ ∅ and μ = ∅ do
5 |   ν ← ν \ {ℓ : ¬ℓ ∈ core}                      /* Remove non-flippable literals from ν */
6 |   ν̄ = {ℓ : ¬ℓ ∈ ν}
7 |   (μ, core) ← SAT(φ ∪ ν̄)                      /* Try to flip remaining literals in ν */
8 ν ← ν ∩ μ                                       /* μ contains flipped variables from ν */
9 return β* ∩ ν
```

Note that in the special case when the core contains a single literal $\ell \in \overline{\nu}$, then $\neg\ell$ is a backbone literal which may be returned separately.

Literature

A. Kaiser and W. Küchlin. Detecting inadmissible and necessary variables in large propositional formulae. In *Intl. Joint Conf. on Automated Reasoning (Short Papers)*. June 2001.

J. Marques-Silva, M. Janota, I. Lynce. On Computing Backbones of Propositional Theories. In *Proc. European Conf. on Art. Intell.*, pages 15–20, 2010.

M. Janota, I. Lynce, J. Marques-Silva. Algorithms for Computing Backbones of Propositional Formulae. *AI Communications*, 2015.