Satisfiability Solvers are Static Analysers[⋆]

Vijay D'Silva[⋆⋆], Leopold Haller, and Daniel Kroening

Department of Computer Science, Oxford University
firstname.surname@cs.ox.ac.uk

Abstract. This paper shows that several propositional satisfiability algorithms compute approximations of fixed points using lattice-based abstractions. The Boolean Constraint Propagation algorithm (BCP) is a greatest fixed point computation over a lattice of partial assignments. The original algorithm of Davis, Logemann and Loveland refines BCP by computing a set of greatest fixed points. The Conflict Driven Clause Learning algorithm alternates between overapproximate deduction with BCP, and underapproximate abduction, with conflict analysis. Thus, in a precise sense, satisfiability solvers are abstract interpreters. Our work is the first step towards a uniform framework for the design and implementation of satisfiability algorithms, static analysers and their combination.

1 How I Learned to Stop SAT Solving and Love Abstract Interpretation

The abstract interpretation approach to program analysis is to compute properties of programs using lattices, transformers and fixed points [5]. The satisfiability approach is to encode programs as formulae that can be analysed with theorem provers [17]. The satisfiability approach has gained popularity in recent years due to dramatic improvements in the performance of propositional satisfiability solvers. The goal of much current research is to combine techniques based on abstract interpretation and based on satisfiability.

This paper shows that propositional satisfiability algorithms compute approximations of fixed points using lattices. Thus, analyses traditionally formulated over lattices and those formulated in terms of satisfiability can both be understood in terms of abstract interpretation. To appreciate the significance of such understanding, consider the program below, where φ is a formula with Boolean variables initialised to arbitrary values.

```
if ( φ ) { assert( false ) }
```

If φ is unsatisfiable, a program verifier that uses a SAT solver will conclude that the assertion is not violated. In contrast, a static analysis like constant propagation (or its conditional variant [26]) cannot always prove the absence of

[⋆] Supported by the Toyota Motor Corporation, EPSRC project EP/H017585/1 and ERC project 280053.

[⋆⋆] Supported by a Microsoft Research European PhD Scholarship.

assertion violations if a formula is unsatisfiable. This result is surprising because we show that all SAT solvers derived from the DPLL procedure use the same lattice as constant propagation. The insight of SAT algorithms is that we can use imprecise abstract domains to gain efficiency, and techniques like decisions and clause-learning to improve precision.

Contribution This paper demonstrates that a broad range of propositional satisfiability algorithms have natural abstract interpretation descriptions. Our contributions include the following characterisations.

1. Propositional satisfiability as a property of fixed points of transformers over the lattice of truth assignments.
2. Boolean Constraint Propagation (BCP) as a greatest fixed point computation over the same lattice as constant propagation.
3. The Davis Putnam Logemann and Loveland algorithm (DPLL) as a refinement of BCP that uses value-based trace partitioning.
4. The conflict driven clause learning algorithm (CDCL) as a combination of overapproximate deduction with underapproximate abduction.

In separate work, we used the formalisation presented here to embed the interval abstract domain inside CDCL and verify programs that are beyond the scope of existing techniques [12]. This paper is organised as follows: We give fixed point semantics to propositional formulae in § 2. To illustrate our approach on simple examples, we formalise truth tables and resolution in § 3. The DPLL algorithm and CDCL are covered in § 4 and § 5.

2 Propositional Satisfiability via Transformers

This section contains a new characterisation of propositional satisfiability using fixed points. We first recall background on propositional logic and lattices.

Propositional Logic. Fix a set *Prop* of propositional variables. A *literal* is a variable or its negation. A *clause* is a disjunction of literals and a *cube* is a conjunction of literals. A formula in *conjunctive normal form* (CNF) is a conjunction of clauses, and a formula in *disjunctive normal form* (DNF) is a disjunction of cubes. Note that the negation of a cube is a clause and vice versa.

The set of truth values is $\mathbb{B} \mathrel{\hat{=}} \{\mathsf{t}, \mathsf{f}\}$. An *assignment* $\sigma : \mathit{Prop} \to \mathbb{B}$ maps variables to truth values. An assignment σ is a *model* of φ, denoted $\sigma \models \varphi$, if σ satisfies φ and is a *countermodel* of φ otherwise. A formula is *satisfiable* if it has a model and is *unsatisfiable* otherwise.

Lattices A *lattice* $(L, \sqsubseteq, \sqcup, \sqcap)$ is a partially ordered set with a meet and a join. Two functions $f, g : Q \to L$ from a set Q to L can be *ordered pointwise*, denoted $f \sqsubseteq g$, if $f(x) \sqsubseteq g(x)$ holds for all x in Q. All functions over L can similarly be lifted pointwise to $Q \to L$. The least and greatest fixed points of a monotone function F on a complete lattice will be denoted $\mathsf{lfp}(F)$ and $\mathsf{gfp}(F)$, respectively.

Let id_S be the identity function. A *Galois connection* between posets $(C, \sqsubseteq)$ and $(A, \preccurlyeq)$, written $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$, is a pair of monotone functions $\alpha : C \to A$ and $\gamma : A \to C$ that satisfy the pointwise constraints $\alpha \circ \gamma \preccurlyeq id_A$ and $id_C \sqsubseteq \gamma \circ \alpha$.

We identify a few lattices of particular interest. The *lattice of truth values* $(\mathbb{B}, \Rightarrow, \vee, \wedge)$ consists of truth values with the implication order $\mathsf{f} \Rightarrow \mathsf{t}$. Disjunction is the join and conjunction is the meet of truth values. The *powerset lattice* over a set X, written $(\mathscr{P}(S), \subseteq, \cup, \cap)$, consists of all subsets of S order by inclusion. Let $(S, \sqsubseteq)$ be a poset. A set $Q \subseteq S$ is *downwards closed* if for every x in Q and y in S, $y \sqsubseteq x$ implies that y is in Q. A downwards closed set is called a *downset*. The *downset lattice* over $(S, \sqsubseteq)$, written $(\mathscr{D}(S), \subseteq, \cap, \cup)$, is the set of downsets of S ordered by inclusion. Downsets strictly generalise powersets because the powerset lattice of S is the downset lattice of S with the identity relation.

2.1 Concrete Semantics of Propositional Formulae

We present new, fixed point characterisations of the models and countermodels of a formula. Satisfiability and validity are properties of such fixed points.

Let $Asg \mathrel{\hat{=}} Prop \to \mathbb{B}$ be the set of assignments. The *concrete domain of assignments* is $(\mathscr{P}(Asg), \subseteq, \cup, \cap)$. A formula φ defines four *assignment transformers*. The name assignment transformers is used by analogy to state transformers and predicate transformers. Let X be a set of assignments. The *model transformer* mod_φ removes all countermodels of φ from X, the *countermodel transformer* $cmod_\varphi$ removes all models of φ from X, the *universal model transformer* $umod_\varphi$ adds all models of φ to X, and the *universal countermodel transformer* $ucmod_\varphi$ adds all countermodels of φ to X.

$$\begin{aligned} mod_\varphi(X) &\mathrel{\hat{=}} \{\sigma \in X \mid \sigma \models \varphi\} & umod_\varphi(X) &\mathrel{\hat{=}} \{\sigma \in Asg \mid \sigma \models \varphi \text{ or } \sigma \in X\} \\ cmod_\varphi(X) &\mathrel{\hat{=}} \{\sigma \in X \mid \sigma \not\models \varphi\} & ucmod_\varphi(X) &\mathrel{\hat{=}} \{\sigma \in Asg \mid \sigma \not\models \varphi \text{ or } \sigma \in X\} \end{aligned}$$

Properties of a formula can be expressed with transformers. The set of models of φ is $mod_\varphi(Asg)$, or equivalently, $umod_\varphi(\emptyset)$. The set of countermodels of φ is $cmod_\varphi(Asg)$, or equivalently, $ucmod_\varphi(\emptyset)$. Algebraic properties of assignment transformers aid in deriving fixed point characterisations of satisfiability. The *De Morgan dual* of a function f on $\mathscr{P}(Asg)$ is the function $\neg \circ f \circ \neg$.

Theorem 1. *The assignment transformers have the following properties.*

1. *The pairs $(mod_\varphi, ucmod_\varphi)$ and $(cmod_\varphi, umod_\varphi)$ are De Morgan duals.*
2. *There are two Galois connections as below.*

$$\mathscr{P}(Asg) \underset{mod_\varphi}{\overset{ucmod_\varphi}{\leftrightarrows}} \mathscr{P}(Asg) \qquad \mathscr{P}(Asg) \underset{cmod_\varphi}{\overset{umod_\varphi}{\leftrightarrows}} \mathscr{P}(Asg)$$

Consider the statement $\mathsf{assume}(\varphi)$. The strongest postcondition is equivalent to mod_φ and the weakest liberal precondition is equivalent to $ucmod_\varphi$. Sound approximations of these transformers are available in abstract domain libraries. Since our characterisation use these transformers, the overhead of lifting satisfiability algorithms to new domains is low. Theorem 2 provides several fixed point characterisations of satisfiability.

Theorem 2. *The following statements are equivalent.*

1. *A formula φ is unsatisfiable.*
2. *The set of assignments $mod_\varphi(Asg)$ is empty.*
3. *The set of assignments $umod_\varphi(\emptyset)$ is empty.*
4. *The set $cmod_\varphi(Asg)$ contains all assignments.*
5. *The set $ucmod_\varphi(\emptyset)$ contains all assignments.*
6. *The greatest fixed point $\mathsf{gfp}(mod_\varphi)$ contains no assignments.*
7. *The least fixed point $\mathsf{lfp}(umod_\varphi)$ contains no assignments.*
8. *The greatest fixed point $\mathsf{gfp}(cmod_\varphi)$ contains all assignments.*
9. *The least fixed point $\mathsf{lfp}(ucmod_\varphi)$ contains all assignments.*

Proof. Due to space restrictions, we do not prove all cases.

(*1 iff 2*) The formula φ is unsatisfiable exactly if it has no models. An assignment σ is in $mod_\varphi(Asg)$ exactly if σ is a model of φ. The set $mod_\varphi(Asg)$ is empty exactly if φ is unsatisfiable.

(*2 iff 5*) Recall that $ucmod_\varphi(X)$ is the De Morgan dual of mod_φ. If $mod_\varphi(Asg)$ is the emptyset, $ucmod_\varphi(\emptyset)$ equals $\neg mod_\varphi(\neg\emptyset)$, which equals Asg.

(*2 iff 4*) The function mod_φ is idempotent, meaning that $mod_\varphi(X)$ is equal to $mod_\varphi(mod_\varphi(X))$ for all X. Since mod_φ is monotone, $mod_\varphi(Asg)$ equals $mod_\varphi(mod_\varphi(Asg))$, so the greatest fixed point of mod_φ is $mod_\varphi(Asg)$. Thus $mod_\varphi(Asg)$ is empty exactly if $\mathsf{gfp}(mod_\varphi)$ is empty.

The argument for the remaining equivalences is similar.

Since all the transformers are idempotent, the fixed points in Theorem 2 may seem superfluous. A sound abstraction of an idempotent function is not necessarily idempotent, so iterating an abstract transformer can provide strictly better results than applying it once. This intuition is formalised by the method of *locally decreasing iterations* [13].

2.2 Abstract Satisfaction

We use the term *abstract satisfaction* for the application of abstract interpretation to design satisfiability algorithms. Abstract interpretation is typically used to overapproximate a least fixed point (such as reachable states), or to underapproximate a greatest fixed point (such as the set of dead variables at a program location). In contrast, we will overapproximate the greatest fixed point $\mathsf{gfp}(mod_\varphi)$ or underapproximate the least fixed point $\mathsf{lfp}(ucmod_\varphi)$. If an overapproximation of $\mathsf{gfp}(mod_\varphi)$ is the emptyset, φ is unsatisfiable. If an underapproximation of $\mathsf{lfp}(ucmod_\varphi)$ contains all assignments, φ is unsatisfiable. Combining information from different approximations yields better results than using either in isolation.

Abstract Interpretation. Assume a Galois connection $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$. The lattice C is called the *concrete domain* and A is called the *abstract domain*. If C is a powerset lattice, an abstract domain with respect to the subset order satisfies $x \subseteq \gamma(\alpha(x))$

and is called an overapproximation. An abstract domain with respect to the superset order satisfies $x \supseteq \gamma(\alpha(x))$ and is called an underapproximation.

Functions on a concrete domain are called *concrete transformers* and those on abstract domains are *abstract transformers.* The abstract transformer G soundly approximates F if $F \circ \gamma \sqsubseteq \gamma \circ G$ holds. The *best abstract transformer* $\alpha \circ F \circ \gamma$ represents the maximum precision that can be derived from an abstraction.

Abstract Interpretation of Satisfiability This section presents new, fixed point approximations of satisfiability.

Let $(O, \sqsubseteq, \sqcup, \sqcap)$ be an overapproximation of the domain of assignments and $(U, \preccurlyeq, \curlyvee, \curlywedge)$ be underapproximation. The approximation is formalised by the Galois connections below. The orders $\sqsubseteq$ and $\preccurlyeq$ both refine the subset order on assignments. That is, $a \sqsubseteq b$ implies $\gamma(a) \subseteq \gamma(b)$, and $x \preccurlyeq y$ implies $\gamma(x) \subseteq \gamma(y)$.

$$(\mathscr{P}(Asg), \subseteq) \underset{\alpha_O}{\overset{\gamma_O}{\leftrightarrows}} (O, \sqsubseteq) \qquad\qquad (\mathscr{P}(Asg), \supseteq) \underset{\alpha_U}{\overset{\gamma_U}{\leftrightarrows}} (U, \succcurlyeq)$$

Abstract transformers can be defined for over- or underapproximating abstractions. We use an overapproximation of the model transformer and underapproximations of the countermodel and universal countermodel transformers. An *abstract model transformer* $amod_\varphi^O : O \to O$, an *abstract countermodel transformer* $acmod_\varphi^U : U \to U$, and an *abstract universal countermodel transformer* $aucmod_\varphi^U : U \to U$ are monotone functions satisfying the constraints below.

$$mod_\varphi \circ \gamma_O \subseteq \gamma_O \circ amod_\varphi^O \qquad\qquad ucmod_\varphi \circ \gamma_U \supseteq \gamma_U \circ aucmod_\varphi^U$$
$$cmod_\varphi \circ \gamma_U \supseteq \gamma_U \circ acmod_\varphi^U$$

Theorem 3 provides sound and possibly incomplete characterisations of unsatisfiability. In contrast to concrete fixed points, the characterisations below are not equivalent because the domains and transformers may have different precision.

Theorem 3. *A propositional formula φ is unsatisfiable if at least one of the conditions below hold.*

1. *The set $\gamma_O(\mathsf{gfp}(amod_\varphi^O))$ is empty.*
2. *The set $\gamma_U(\mathsf{lfp}(aucmod_\varphi^U))$ contains all assignments.*
3. *The set $\gamma_O(x) \cap \neg\gamma_U(y)$ is empty in $(x, y) = \gamma_{OU}(\mathsf{gfp}(amc_\varphi^{OU}))$.*

Theorem 3 follows from the soundness of abstract interpretation. The rest of the paper shows that satisfiability algorithms compute these abstract fixed points.

3 Sound and Complete Abstractions

In this section, we formalise the construction of truth tables and resolution proofs in the abstract satisfaction framework. Truth table construction is abstract transformer application and the resolution rule is a sound abstract transformer. Repeated application of the resolution rule is abstract transformer iteration.

Truth Tables A truth table is an enumeration that represents whether each truth assignment satisfies a formula. In abstract satisfaction, truth tables are a representation of the domain of assignments and truth table construction is application of the best abstract transformer for a formula. Binary Decision Diagrams are semantically equivalent but have a more efficient representation.

Example 1. This example illustrates the order on truth tables. Consider the formula $\varphi = p \wedge \neg q$. The set of assignments $\{p, q\} \to \mathbb{B}$ is shown in gray below. The truth tables for the formulae p and $\neg q$ are shown below.

p	q	p		$\neg q$		$p \wedge \neg q$
f	f	f		t		f
f	t	f	$\sqcap$	f	$=$	f
t	f	t		t		t
t	t	t		f		f

If the implication order on $\mathbb{B}$ is lifted to truth tables, the truth table for $p \wedge \neg q$ is the pointwise meet of the truth tables for p and $\neg q$. $\lhd$

A *truth table* is a function in $\mathit{Table} \mathrel{\hat{=}} \mathit{Asg} \to \mathbb{B}$. The domain of truth tables $(\mathit{Table}, \sqsubseteq, \sqcup, \sqcap)$ is ordered by pointwise lifting of the implication order on truth values. Specifically, $T_1 \sqsubseteq T_2$ if $T_1(\sigma) \Rightarrow T_2(\sigma)$ for every assignment σ. A set of assignments X abstracts to the truth table T that maps assignments in X to t and all other assignments to f. The functions α and γ below form a Galois connection, are bijections and satisfy that $\gamma \circ \alpha$ and $\alpha \circ \gamma$ are identity functions. That is, the Galois connection is a Galois isomorphism, meaning that truth tables do not abstract information.

$$\alpha(X) \mathrel{\hat{=}} \{\sigma \mapsto \mathsf{t} \mid \sigma \in X\} \cup \{\sigma \mapsto \mathsf{f} \mid \sigma \notin X\} \qquad \gamma(T) \mathrel{\hat{=}} \{\sigma \mid T(\sigma) = \mathsf{t}\}$$

Consider the best abstract transformer for mod_φ, denoted amod_φ. Observe that $\mathit{amod}_\varphi(\top)$ represents the truth table for φ. Thus, truth table construction can be viewed as transformer application. The completeness of truth-table construction is expressed as $\mathit{mod}_\varphi \circ \gamma = \gamma \circ \mathit{amod}_\varphi$.

Resolution The *resolution principle* states that an assignment satisfying the clauses $C \vee p$ and $\neg p \vee D$ also satisfies $C \vee D$ [21]. The variable p is the *pivot* and $C \vee D$ is the *resolvent*. Resolution is sound but is not complete for deriving arbitrary implications. For example, the formula $p \wedge q$ implies $p \vee \neg q$, but this implication cannot be derived by resolution. Resolution is refutation complete: a formula is unsatisfiable exactly if the empty clause can be derived by resolution.

In abstract satisfaction, CNF formulae, with the superset order, are an abstract domain, and resolution defines an abstract transformer. The abstract transformer is a sound but incomplete abstraction.

Let Lit be the set of literals over the propositional variables Prop, and $\mathit{Clause} \mathrel{\hat{=}} \mathscr{P}(\mathit{Lit})$ be the set of clauses. The CNF *domain* $\mathit{CNF} \mathrel{\hat{=}} \mathscr{P}(\mathit{Clause})$ contains sets of clauses with the superset order $(\mathit{CNF}, \supseteq, \cap, \cup)$. The superset order underapproximates implication because $\varphi \supseteq \psi$ entails $\varphi \Rightarrow \psi$ but the converse is not true. The functions below are related by the Galois connection $(\mathscr{P}(Asg), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (CNF, \supseteq)$.

$$\alpha(X) \mathrel{\hat{=}} \{C \mid X \subseteq mod_C(Asg)\} \qquad \gamma(\varphi) \mathrel{\hat{=}} mod_\varphi(Asg)$$

We formalise resolution with a transformer. The resolvents derived from φ with pivot x are denoted $res(x, \varphi)$. The *resolution transformer* $Res_\varphi : CNF \to CNF$ adds all possible resolvents to a set of clauses.

$$res(x, \varphi) \mathrel{\hat{=}} \{C \vee D \mid x \vee C \text{ and } \neg x \vee D \text{ are in } \varphi\}$$
$$Res_\varphi(\psi) \mathrel{\hat{=}} \varphi \cup \psi \cup \bigcup_{x \in Prop} res(x, \varphi)$$

We express properties of resolution next. Logical soundness stating that every clause derived by resolution is implied by φ becomes the condition $\alpha \circ mod_\varphi \supseteq Res_\varphi \circ \alpha$. Res_φ is not idempotent, so multiple applications of resolution yield more resolvents than a single application. The set of clauses derived by resolution is the fixed point $\mathsf{gfp}(Res_\varphi)$. Resolution is not complete for arbitrary implications, so in general, $\alpha(\mathsf{gfp}(mod_\varphi))$ is a strict superset of $\mathsf{gfp}(Res_\varphi)$. The refutation completeness of resolution becomes the condition that $\gamma(\mathsf{gfp}(Res_\varphi))$ is the empty set exactly if $\mathsf{gfp}(Res_\varphi)$ contains the empty clause.

4 Fixed Point Refinement

In this section, we formalise the classic DPLL procedure. We first characterise Boolean Constraint Propagation as abstract fixed point iteration.

4.1 Boolean Constraint Propagation

The workhorse of all solvers based on DPLL is the Boolean Constraint Propagation (BCP) routine. BCP repeatedly applies a transformation called the *unit rule* to a data structure called a *partial assignment.* In abstract satisfaction, partial assignments are an abstract domain, the unit rule is the best abstract transformer for a clause, and BCP computes a greatest fixed point.

Example 2. We illustrate BCP with the formula below.

$$\varphi \mathrel{\hat{=}} p \wedge (\neg p \vee \neg q) \wedge (q \vee r \vee \neg s) \wedge (q \vee r \vee s)$$

Initially, nothing is known about the formula, encoded by the empty set. Then, BCP concludes that p must be true in every satisfying assignment. Since p must be true, BCP concludes that q must be false to satisfy the clause $\neg p \vee \neg q$.

$$\pi_0 \mathrel{\hat{=}} \top \qquad \pi_1 \mathrel{\hat{=}} \langle p{:}\mathsf{t} \rangle \qquad \pi_2 \mathrel{\hat{=}} \langle p{:}\mathsf{t}, q{:}\mathsf{f} \rangle$$

All the remaining clauses have more than one literal unassigned, so BCP terminates. BCP is a sound but incomplete deduction procedure. BCP need not begin

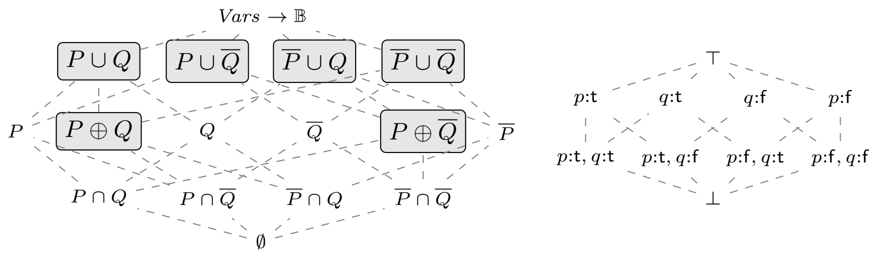


Fig. 1. Domains for assignments over p and q. The concrete domain $\mathscr{P}(Asg)$ is on the left. The set P contains assignments that map p to true. Partial assignments are on the right. The shaded elements of $\mathscr{P}(Asg)$ cannot be represented as partial assignments.

with π_0 as above. We can begin by assuming p is true, q is false, and r is false, written $\pi \mathrel{\hat{=}} \langle p{:}\mathsf{t}, q{:}\mathsf{f}, r{:}\mathsf{f}\rangle$. Given π, BCP concludes, from $(q \vee r \vee \neg s)$, that s must be false and from $(q \vee r \vee s)$ that s must be true. This situation, denoted $\bot$, is a *conflict*. No assignment extending π satisfies φ. $\lhd$

We show that partial assignments are an abstract domain. A *partial assignment* is a partial function in $Prop \to \mathbb{B}$. Consider the set $\{\mathsf{t}, \mathsf{f}, \top\}$ with the *information order* $\mathsf{t} \sqsubseteq \top$ and $\mathsf{f} \sqsubseteq \top$. We model a partial assignment as a total function $\pi : Prop \to \{\mathsf{t}, \mathsf{f}, \top\}$, where for each variable p, $\pi(x)$ is $\top$ if π is undefined on p. The *domain of partial assignments* $(PAsg, \sqsubseteq)$ contains a set $PAsg \mathrel{\hat{=}} (Prop \to \{\mathsf{t}, \mathsf{f}, \top\}) \cup \{\bot\}$, of partial assignments extended with a least element $\bot$, called a *conflict*. The order between non-$\bot$ elements is the pointwise lifting of the information order. A partial assignment in which p is t and other variables map to $\top$ is written $\langle p{:}\mathsf{t}\rangle$. Figure 1 depicts partial assignments over two variables.

A variant of the partial assignments domain is used for constant propagation [16] and is equivalent to the Cartesian abstraction [4]. In abstract interpretation parlance, partial assignments as presented here are a reduction of the Cartesian abstraction domain in which the empty set has a unique representation. The abstraction and concretisation functions $\alpha_{PAsg} : \mathscr{P}(Asg) \to PAsg$ and $\gamma_{PAsg} : PAsg \to \mathscr{P}(Asg)$ below are standard and are known to form a Galois connection.

$$\alpha_{PAsg}(\emptyset) \mathrel{\hat{=}} \bot \quad \alpha_{PAsg}(S) \mathrel{\hat{=}} \left\{ x \mapsto \bigsqcup \{\sigma(x) \mid \sigma \in S\} \mid x \in Prop \right\}, \text{ for } S \neq \emptyset$$
$$\gamma_{PAsg}(\bot) \mathrel{\hat{=}} \emptyset \quad \gamma_{PAsg}(\pi) \mathrel{\hat{=}} \{\sigma \in Asg \mid \text{ for all } x \text{ in } Prop, \sigma(x) \sqsubseteq \pi(x)\}$$

We formalise the unit rule. The *unit rule* states that if all but one literals in a clause are false under a partial assignment, the remaining literal must be true. It is defined by a function $\mathsf{unit} : Clause \times PAsg \to PAsg$. The image of a clause θ

under a partial assignment π is false if π and makes all literals in θ false.

$$\mathsf{unit}(\theta, \pi) \mathrel{\hat{=}} \begin{cases} \bot & \text{if } \pi(\theta) \text{ is } \mathsf{f} \\ \pi \cup \{p \mapsto \mathsf{t}\} & \text{if } \theta \text{ is } \psi \vee p \text{ and } \pi(\psi) = \mathsf{f} \\ \pi \cup \{p \mapsto \mathsf{f}\} & \text{if } \theta \text{ is } \psi \vee \neg p \text{ and } \pi(\psi) = \mathsf{f} \\ \pi & \text{otherwise} \end{cases}$$

Example 3. We illustrate the unit rule with $\varphi \mathrel{\hat{=}} \neg p \wedge (p \vee \neg q)$. Assume we have best abstract transformers for literals. The abstract transformer for φ is derived by replacing conjunction and disjunction by pointwise meet and join.

$$amod_{\varphi} \mathrel{\hat{=}} amod_{\neg p} \sqcap (amod_{p} \sqcup amod_{\neg q})$$

We compute a greatest fixed point in the partial assignments domain.

$$\pi_0 \mathrel{\hat{=}} \langle p{:}\top, q{:}\top\rangle \qquad \pi_1 \mathrel{\hat{=}} \langle p{:}\mathsf{f}, q{:}\top\rangle \qquad \pi_2 \mathrel{\hat{=}} \langle p{:}\mathsf{f}, q{:}\mathsf{f}\rangle \qquad \pi_3 \mathrel{\hat{=}} \langle p{:}\mathsf{f}, q{:}\mathsf{f}\rangle$$

Applying the unit rule generates the same sequence. $\lhd$

Lemma 1. *For a fixed clause θ, the unit rule is equivalent to the best abstract transformer:* $\mathsf{unit}(\theta, \pi) = \alpha_{PAsg} \circ mod_\theta \circ \gamma_{PAsg}(\pi)$.

Proof. Consider a partial assignment π and the best abstract transformer $amod_\theta \mathrel{\hat{=}} \alpha_{PAsg} \circ mod_\theta \circ \gamma_{PAsg}$. We distinguish the cases in the definition of unit.

($\pi(\theta)$ *is* f) If π makes every literal in θ false, $\mathsf{unit}(\theta, \pi) = \bot$. No assignment in $\gamma_{PAsg}(\pi)$ will θ, so $mod_\theta(\gamma_{PAsg}(\pi))$ is the empty set and by definition of α_{PAsg}, from $amod_\theta(\pi) = \bot$.

($\theta = \psi \vee p$ *and* $\pi(\psi) = \mathsf{f}$) Here, $\mathsf{unit}(\theta, \pi) = \pi \cup \{p \mapsto \mathsf{t}\}$. Since p is unassigned, $\pi(p) = \top$, and $\gamma(\pi)$ contains assignments in which every p is true and false and all in φ are false. The set $mod_\theta(\gamma_\pi(\pi))$ only includes assignments that satisfy p because no other literal is satisfied. All other variables are unaffected. Thus, $\alpha_{PAsg}(mod_\theta(\gamma_\pi(\pi)))$ equals $\pi \cup \{p \mapsto \mathsf{t}\}$.

(π *undefined for multiple variables in* θ) The unit rule leaves π unchanged. At least two literals in θ are undefined in π, so $mod_\theta(\gamma_{PAsg}(\pi))$ contains an assignment that makes one true and the other false and vice-versa. Consequently, the variables for both literals map to $\top$ in $\alpha_{PAsg}(mod_\theta(\gamma_{PAsg}(\pi)))$ and π is unchanged, as required.

BCP maps a formula φ and a partial assignment π representing an assumption to the result of applying the unit rule repeatedly with all clauses till no changes are observed. Formally, BCP is a function $\mathsf{bcp} : CNF \times PAsg \to PAsg$.

Let φ be a formula, θ represent a clause, and $amod_\theta$ be the best abstract transformer for mod_θ. We model the effect of *concrete deduction* from a partial assignment Δ with the concrete transformer $mod_{\varphi,\Delta}$.

$$mod_\varphi : PAsg \times \mathscr{P}(Asg) \to \mathscr{P}(Asg) \qquad mod_{\varphi,\Delta}(x) \mathrel{\hat{=}} mod_\varphi(x \cap \gamma(\Delta))$$

The abstract *deduction transformer* below overapproximates $mod_{\varphi,\Delta}$.

$$ded_\varphi : PAsg \times PAsg \to PAsg \qquad ded_{\varphi,\Delta}(\pi) \mathrel{\hat{=}} \bigsqcap\{amod_\theta(\pi \sqcap \Delta) \mid \theta \text{ is in } \varphi\}$$

The soundness constraint $mod_{\varphi,\Delta} \circ \gamma_{PAsg} \subseteq \gamma_{PAsg} \circ ded_{\varphi,\Delta}$ implies that all conclusions derived by $ded_{\varphi,\Delta}$ are satisfied by all models of φ in Δ. Example 4 shows that the deduction transformer is not complete.

Example 4. The formula $\varphi \mathrel{\hat{=}} (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q) \wedge (p \vee q)$ is unsatisfiable. The best abstract transformer satisfies $\alpha_{PAsg}(mod_{\varphi,\top}(\gamma_{PAsg}(\top))) = \bot$ whereas the deduction transformer satisfies $\alpha_{PAsg}(mod_{\varphi,\top}(\gamma_{PAsg}(\top))) = \top$. Thus, the abstract deduction transformer is incomplete. $\triangleleft$

Theorem 4. *The result of Boolean Constraint Propagation* $\mathsf{bcp}(\varphi, \Delta)$ *is equivalent to the greatest fixed point* $\mathsf{gfp}(ded_{\varphi,\Delta})$.

In abstract interpretation terms, BCP is bottom-up abstract interpretation of Boolean expressions with locally decreasing iterations [13, 4].

4.2 The Classic DPLL Algorithm

We say classic DPLL, or DPLL, for the algorithm of Davis, Logemann, and Loveland [10]. The DPLL algorithm simplifies the algorithm of Davis and Putnam [11] by eliminating the resolution and pure literal rules. If BCP is viewed as a static analysis, DPLL can be understood as running BCP on the sequence of programs below. In abstract satisfaction terms, DPLL dynamically restricts the range of values a variable can take to improve precision. It is a procedure to dynamically discover value-based trace partitions [20].

$$\mathtt{P}_0 \mathrel{\hat{=}} \begin{array}{l}\mathtt{if}(\varphi)\\ \quad\mathtt{assert}(\mathsf{f})\end{array} \qquad \mathtt{P}_1 \mathrel{\hat{=}} \begin{array}{l}\mathtt{if}(p)\ \mathtt{P}_0\\ \mathtt{else}\ \mathtt{P}_0\end{array} \qquad \mathtt{P}_2 \mathrel{\hat{=}} \begin{array}{l}\mathtt{if}(q)\ \mathtt{P}_1\\ \mathtt{else}\ \mathtt{P}_1\end{array}$$

Example 5. Revisit the formula $\varphi \mathrel{\hat{=}} (\neg p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee \neg q) \wedge (p \vee q)$ which could not be refuted by BCP. Since $\mathsf{gfp}(ded_{\varphi,\top})$ is $\top$, DPLL concludes that precision was lost and computes two fixed points $\mathsf{gfp}(ded_{\varphi,\langle p:\mathsf{t}\rangle})$ and $\mathsf{gfp}(ded_{\varphi,\langle p:\mathsf{f}\rangle})$. Both fixed points are $\bot$, so DPLL concludes that φ is unsatisfiable. $\triangleleft$

DPLL operates in two phases, using two abstract domains. One phase consists of deduction under assumptions and uses BCP. The other phase refines assumptions and is formalised next. DPLL only considers assumptions that can be represented by partial assignments, but such a restriction is not necessary.

Example 6. Figure 2 illustrates partitions of two variable assignments. An element $\cdots/\mathsf{f},\mathsf{f}$ represents a partition in which one block contains the assignment $\{p \mapsto \mathsf{f}, q \mapsto \mathsf{f}\}$ and the other block contains all other assignments. DPLL can be run using the assignments in each partition as assumptions. The partition lattice is large, with the size given by the *Bell number*.

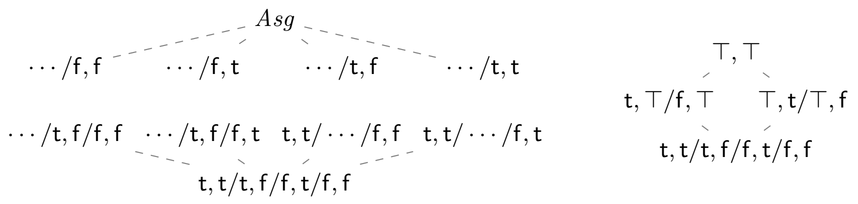


Fig. 2. The concrete domain for case-based reasoning is the lattice of partitions over assignments. The abstract domain only contains partitions that can be expressed as partial assignments.

An abstract lattice of partitions reduces the cases that must be considered. Figure 2 depicts partitions that can be expressed as partial assignments. The partition consisting of the two sets represented by $p \iff q$ and $p \iff \neg q$ cannot be expressed with partial assignments but the partition consisting of $p \iff \mathsf{f}$ and $p \iff \mathsf{t}$ can. ◁

An *abstract partition* is a set $\chi \subseteq A$ of elements from an abstract domain satisfying that $\{\gamma(a) \mid a \in \chi\}$ is a partition. Given two abstract partitions, χ_1 *refines* χ_2, denoted $\chi_1 \preccurlyeq \chi_2$, if for every a_2 in χ_2, there is an a_1 in χ_1 such that $a_1 \sqsubseteq a_2$. An abstract partition represents cases used in deduction. Let $(\mathit{Cases}(\mathit{PAsg}), \preccurlyeq)$ be the set of abstract partitions over partial assignments ordered by refinement.

Let χ be an abstract partition. The *case deduction* transformer models the effect of using each block of a partition as an assumption.

$$\mathit{acase}_\varphi : \chi \to \mathit{PAsg} \qquad \mathit{acase}_\varphi \mathrel{\hat{=}} \{\Delta \mapsto \mathsf{gfp}(\mathit{ded}_{\varphi,\Delta}) \mid \Delta \in \chi\}$$

In the refinement step, a variable that is currently undefined is used to refine a block of the partition. We model selection of an unassigned variable with a function $\mathit{pick} : \mathit{PAsg} \to \mathit{Prop}$ that maps a partial assignment π to a variable p for which $\pi(p) = \top$. The *case split* function $\mathit{split} : \mathit{PAsg} \to \mathscr{P}(\mathit{PAsg})$ formalises refinement of a partition based on deduction.

$$\mathit{split}(\pi) \mathrel{\hat{=}} \{\pi \sqcap \langle p{:}\mathsf{t}\rangle, \pi \sqcap \langle p{:}\mathsf{f}\rangle \mid p = \mathit{pick}(\mathsf{gfp}(\mathit{ded}_{\varphi,\pi}))\}$$

DPLL runs until the formula is shown to be unsatisfiable or a satisfying assignment is found. Satisfying assignments are formalised using covering. An element a in a lattice *covers* $\bot$ if there is no distinct a' satisfying $\bot \sqsubseteq a' \sqsubseteq a$. Elements of PAsg covering $\bot$ are assignments. If deduction under every block of a partition yields $\bot$, the formula is unsatisfiable.

Algorithm 1 presents an abstract interpretation perspective of DPLL. Since every function acase_φ represents a trace partition [20], DPLL can be understood as a procedure to dynamically discover a trace partition.

```
Abstract-DPLL($\varphi, \chi$)
    Compute $acase_\varphi$
    if $acase_\varphi(\Delta) = \bot$ for all $\Delta$ in $\chi$ then return UNSAT
    if $acase_\varphi(\Delta)$ covers $\bot$ for some $\Delta$ in $\chi$ then return SAT
    else
        $\chi \leftarrow (\chi \setminus \{\Delta\}) \cup split(\Delta)$
    Abstract-DPLL($\varphi, \chi$)
```

Algorithm 1: DPLL as fixed point computation with dynamic refinement

5 Conflict Driven Clause Learning

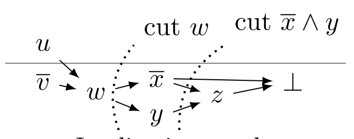
This section formalises the Conflict Driven Clause Learning (CDCL) algorithm. Though CDCL historically derives from DPLL, DPLL can naturally be viewed as a recursive search procedure, while the search pattern of CDCL is more intricate. DPLL uses case based reasoning to refine an analysis. CDCL uses clause learning to refine the transformers used to compute a fixed point. In terms of programs, every iteration of CDCL generates and analyses a program of the form below.

$$\mathtt{P}_0 \mathrel{\hat{=}} \mathtt{if}(\varphi)\ \mathtt{assert}(\mathtt{f}) \qquad \mathtt{P}_1 \mathrel{\hat{=}} \mathtt{if}(\theta_1)\ \mathtt{P}_0 \qquad \mathtt{P}_2 \mathrel{\hat{=}} \mathtt{if}(\theta_2)\ \mathtt{P}_1$$

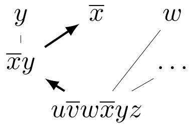
Example 7. This example illustrates a run of CDCL on a formula φ.

$$\varphi \mathrel{\hat{=}} \{\ \{\neg u, v, w\}, \{\neg w, \neg x\}, \{\neg w, y\}, \{x, \neg y, z\}, \{x, \neg z\}, \{x, y\}, \{\neg y, \neg x\}\ \}$$

CDCL initially proceeds like DPLL and alternates BCP and decisions. The steps in BCP are recorded by an implication graph shown below. A directed edge from u to w and from $\neg v$ to w indicates that BCP deduced that w is true if u is true and v is false. A cut in the graph represents a conjunction of literals. A cut that separates u and $\neg v$ from $\bot$ represents a sufficient condition for a conflict. The disjunction of formulae represented by a set of cuts is also sufficient for a conflict.



Implication graph



Choices

The first step of conflict analysis is to heuristically choose a cut. A single cut is used rather than a set to save space. Suppose the solver chooses the cut $\neg x \wedge y$.

The second step is to generalise the cut. Observe that if $\neg x$ holds, the unit rule and the clause $\{x, y\}$ imply y. Similarly, the solver can use y and $\{\neg x, \neg y\}$ to deduce $\neg x$. The conflict can be generalised to either or $\neg x$ or y. If $\neg x$ is sufficient for a conflict, its negation x must be satisfied by all models of φ. The solver *learns* the clause $\{x\}$ and continues with model search. $\lhd$

We view CDCL as operating in two phases. In the *model search* phase, CDCL uses BCP to draw conclusions about all models of φ. Since $\varphi \Rightarrow \psi$ if all models

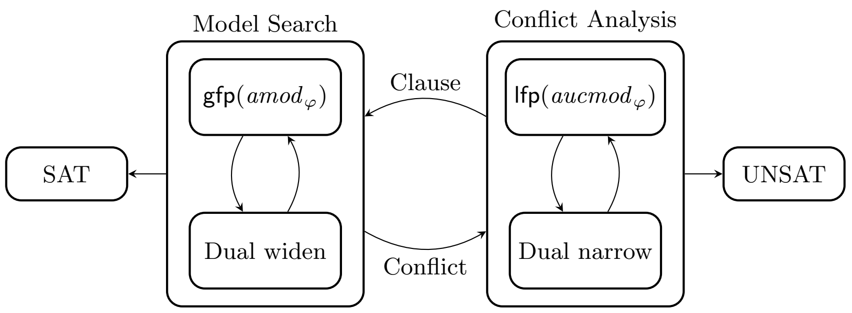


Fig. 3. Abstract Interpretation view of CDCL

of φ satisfy ψ, we say that BCP *overapproximates deduction*. The incompleteness of BCP translates into imprecision in an abstract transformer. CDCL uses decisions to gain precision. That is, CDCL makes assumptions (that we write as a formula $\varDelta$) until it finds a satisfying assignment, or until $\varphi \wedge \varDelta \Rightarrow \mathsf{f}$. Unlike in DPLL, only one assumption is made, so the use of assumptions is unsound.

After a conflict is found, CDCL enters the *conflict analysis* phase. The goal of conflict analysis is to derive a formula θ such that $\varphi \wedge \theta$ implies f. Given formulae φ and ψ, the task of deriving θ such that $\varphi \wedge \theta \Rightarrow \psi$ is called abduction. Conflict analysis only derives those θ that can be expressed as a cube, so this step underapproximates abduction. The abstract interpretation view of CDCL is illustrated in Figure 3 and formalised next.

Model Search and Extrapolation As before, BCP is a greatest fixed point computation with the abstract transformer $amod_\varphi$. Decisions are used to increase precision by iterating below the greatest fixed point $\mathsf{gfp}(ded_{\varphi,\top})$. Recall that widening operators are used to ascend up a lattice in a least fixed point computation. Decisions underapproximate the greatest fixed point computed by BCP and are dual widening operators [8]. Widening is typically used to enforce convergence. The goal of decisions is not convergence, so we use the term *extrapolation*, suggested in [8] for a weakening of widening without a convergence requirement.

A *downwards extrapolation* on a lattice is a function $f : L \to L$ satisfying $f(a) \sqsubseteq a$ for all a. Such a function is usually called *reductive* or *decreasing*, but we prefer extrapolation to emphasise the connection to widening. We model decisions with the downwards extrapolation function below.

$$\begin{aligned} ext\!\downharpoonleft &: PAsg \to PAsg \\ ext\!\downharpoonleft(\pi) &\mathrel{\hat{=}} \pi \sqcap \langle p{:}b\rangle \text{ where } p = pick(\pi) \text{ and } b \in \mathbb{B} \end{aligned}$$

The model search phase of CDCL computes $\pi = \mathsf{gfp}(ded_{\varphi,\top})$. If π is $\bot$, the formula is unsatisfiable. If π covers $\bot$, the formula is satisfiable. In other cases, extrapolation is used to derive a partial assignment $\varDelta = ext\!\downharpoonleft(\pi)$. This partial assignment represents the new assumptions that will be used. Model search continues by computing $\mathsf{gfp}(\mathit{ded}_{\varphi,\Delta})$. Extrapolation is typically used to accelerate convergence of a fixed point computation by losing precision while preserving soundness. The application of extrapolation to gain precision at the cost of soundness in CDCL is unusual.

Conflict Analysis and Interpolation If model search with extrapolation discovers an element Δ such that $\mathsf{gfp}(\mathit{ded}_{\varphi,\Delta})$ is $\bot$, CDCL enters the conflict analysis phase. The goal of conflict analysis is to generalise the reason for the conflict. In terms of concrete transformers, we have that $\mathit{mod}_{\varphi}(\gamma(\Delta))$ is empty and wish to compute the set of countermodels $\mathit{ucmod}_{\varphi}(\gamma(\Delta))$. This set is underapproximated using an underapproximate domain and transformer.

Example 8. This example illustrates the domain and transformers used for conflict analysis. Revisit the implication graph in Example 7. Every cut in the graph that separates the vertices u and $\neg v$ from $\bot$ is a reason for a conflict. Such cuts can be computed by traversing the graph starting from $\bot$.

$$C_0 = \{\{\bot\}\} \qquad C_1 = \{\{\bot\}, \{\neg x, z\}\} \qquad C_2 = \{\{\bot\}, \{\neg x, z\}, \{\neg x, y\}\}$$

Note that a graph cut is a set of vertices, so the set of graph cuts is a set of sets of vertices. Unlike breadth-first reachability, which only maintains a set of vertices, the iteration above maintains a set of sets of vertices. $\lhd$

We formalise the domain and transformer for conflict analysis. A cut in the implication graph represents a conjunction of literals, so every cut c can be represented by a partial assignment π_c. A set of cuts is a set of partial assignments. If c is a set of vertices representing a cut, every set of vertices d that contains c also represents a cut. If c is contained in d, the corresponding partial assignments satisfy $\pi_d \sqsubseteq \pi_c$. The domain for conflict analysis is downwards closed sets (downsets) of partial assignments.

Let $(\mathscr{D}(PAsg), \subseteq)$ be the family of downsets of partial assignments. We make the standard assumption that downsets are represented by their maximal elements. The lattice of downsets is an underapproximating abstract domain with the following abstraction and concretisation functions [7].

$$\alpha_{\mathscr{D}}(X) = \bigcup \{\pi{\downarrow} \mid \gamma_{PAsg}(\pi) \subseteq X\} \qquad \gamma_{\mathscr{D}}(Y) = \{\gamma_{PAsg}(\pi) \mid \pi \in Y\}$$

Since every set of assignments is also a set of partial assignments, this abstract domain can represent all sets of assignments. We also note that the downset lattice is called the *disjunctive completion* of an abstract domain.

We model *concrete abduction* with the transformer below.

$$\mathit{ucmod}_{\varphi} : PAsg \times \mathscr{P}(Asg) \to \mathscr{P}(Asg) \quad \mathit{ucmod}_{\varphi,\Delta}(x) \mathrel{\hat{=}} \mathit{ucmod}_{\varphi}(x \cup \gamma_{PAsg}(\Delta))$$

An *abstract abduction* transformer $\mathit{abd}_{\varphi} : PAsg \times \mathscr{D}(PAsg) \to \mathscr{D}(PAsg)$ underapproximates concrete abduction and maps a partial assignment Δ and set Q to a set of partial assignments derived from Q.

We describe an instance of abduction which formalises clause minimisation [23]. In general, other techniques such as cutting a conflict graph [22] may also be used.

$$\mathit{minimise}_{\varphi,\Delta}(P) \mathrel{\hat{=}} \{\pi \in \mathit{PAsg} \mid \exists\theta \in \mathit{Form}.\ \mathit{amod}_\theta(\pi) \sqsubseteq \pi', \pi' \in P \cup \{\Delta\}\}$$

The conflict minimisation transformer $\mathit{minimise}_{\varphi,\Delta}$ finds all partial assignments from which a known conflict can be deduced with the unit rule. Applying abduction may produce a set of partial assignments. Conflict analysis is expensive, so solvers heuristically choose a single partial assignment and return to model search.

In a dual manner to deduction, underapproximating the set of reasons for a conflict can be viewed as a least fixed point computation. Recall that narrowing operators are used to overapproximate the limit of a decreasing iteration sequence [8]. A dual narrowing operator can be used to underapproximate the limit of an increasing iteration sequence. Choosing a reason for a conflict can be viewed as dual narrowing. For similar reasons to our use of extrapolation, the term *interpolation* is more appropriate because convergence is not an issue. The use of the term interpolation should not be confused with Craig interpolants.

An *upwards interpolation* on a lattice is a function $f : L \times L \to L$ satisfying that $a \sqsubseteq b \Rightarrow a \sqsubseteq f(a,b) \sqsubseteq b$ for all a, b. We model heuristic choice among candidates as the upwards interpolation function below.

$$\mathit{int}\!\upharpoonleft\, : \wp(\mathit{PAsg}) \times \wp(\mathit{PAsg}) \to \wp(\mathit{PAsg})$$
$$\text{For } P \subseteq Q,\ \mathit{int}\!\upharpoonleft\!(P,Q) \mathrel{\hat{=}} \{\mathit{choose}(p,Q) \mid p \text{ is maximal in } P\}$$

The statement $\mathit{choose}(p,Q)$ above is defined when p is an element of Q and returns a maximal element q of Q with $p \sqsubseteq q$.

Example 9. In Example 8, the initial conflict is $p = \langle u:\mathsf{t}, v:\mathsf{f}, w:\mathsf{t}, x:\mathsf{f}, y:\mathsf{t}, z:\mathsf{f}\rangle$. The two graph cuts produce the set of candidates $Q = \{\langle w:\mathsf{t}\rangle, \langle x:\mathsf{f}, y:\mathsf{t}\rangle\}$. The second element of the set is chosen. This corresponds to the application of upwards interpolation $\mathit{int}\!\upharpoonleft\!(\{p\}, Q) = \{\langle x:\mathsf{f}, y:\mathsf{t}\rangle\}$. $\lhd$

6 Related Work and Discussion

Standard static analysis is, of necessity, incomplete and computes approximations. A surprising insight of our work is that satisfiability procedures operate over imprecise abstractions but obtain sound and complete results. The main reason is that SAT solvers use techniques to refine the precision of an analysis.

The verification literature contains numerous examples of domain refinement, originating in [6]. A very popular refinement technique at present is Counterexample Guided Abstraction Refinement (CEGAR) [3]. We believe the refinement in SAT solvers is very different from CEGAR. Each iteration of the CEGAR loop requires constructing a new abstraction and new transformers. In stark contrast, SAT solvers never change the domain. This immutability is crucial for efficiency as abstract domain implementations can be highly optimised. In fact, SAT algorithms can be understood as a portfolio of techniques for refinement without domain manipulation.

The refinement in BCP is to compute a fixed point instead of applying a transformer. BCP uses locally decreasing iterations [13] to refine conditional constant propagation [26], which in turn refines constant propagation [16]. The refinement in DPLL is to compute a set of fixed points instead of a single fixed point. A run of DPLL can be understood as a search for a sufficiently precise set of fixed points or as a search for a trace partition [15, 20]. CDCL uses two types of refinements. Decisions refine the starting element for fixed point iteration to eliminate precision loss. Conflict analysis refines the input constraints.

We are not aware of existing program analysis techniques that generalise CDCL in a strict mathematical sense but there are several tantalizing similarities that deserve closer study. Transformer refinement in predicate abstraction [1] achieves a similar effect to clause learning. Counterexample DAGs in [14] play a similar role to implication graphs, while the combination of testing with weakest preconditions in Yogi [2] and with interpolants in lazy annotation [18] resembles the interplay between decisions and conflict analysis.

The breadth and diversity of the satisfiability literature made it infeasible to cover all but a few *propositional satisfiability* procedures in this paper. Stålmarck's method is not covered in this paper but can naturally be understood as an extension of BCP that combines case-based refinement with joins. Thakur and Reps [24, 25] have recently applied abstract interpretation to generalise Stålmarck's method and shown that this generalisation has applications beyond SAT solving.

We conjecture that algorithms for solving satisfiability in a theory (SMT) have abstract interpretation characterisations and may independently exist in the static analysis literature. The analysis of a formula based on its propositional structure in DPLL(T) [19] is remarkably similar to the program analysis using control flow paths. The Nelson-Oppen combination procedure was recently shown to be an instance of the iterative reduced product [9]. We believe that these are but a few directions that must be explored en route to an exciting unification of the theory and practice of decision procedures and static analysers.

Acknowledgements We are deeply indebted to the French static analysis community, and Patrick and Radhia Cousot in particular, for their encouragement and support.

References

1. T. Ball, R. Majumdar, T. D. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *Proc. of Programming Language Design and Implementation*, pages 203–213. ACM Press, 2001.
2. N. E. Beckman, A. V. Nori, S. K. Rajamani, and R. J. Simmons. Proofs from tests. In *Proc. of Software Testing and Analysis*, pages 3–14. ACM Press, 2008.
3. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *JACM*, 50:752–794, 2003.

4. P. Cousot. Abstract interpretation. MIT course 16.399, Feb.–May 2005.
5. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252. ACM Press, 1977.
6. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282. ACM Press, 1979.
7. P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.
8. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, Aug. 1992.
9. P. Cousot, R. Cousot, and L. Mauborgne. The reduced product of abstract domains and the combination of decision procedures. In *FoSSaCS*, pages 456–472, 2011.
10. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *CACM*, 5:394–397, July 1962.
11. M. Davis and H. Putnam. A computing procedure for quantification theory. *JACM*, 7:201–215, July 1960.
12. V. D'Silva, L. Haller, D. Kroening, and M. Tautschnig. Numeric bounds analysis with conflict-driven learning. In *TACAS*, pages 48–63. Springer, 2012.
13. P. Granger. Improving the results of static analyses programs by local decreasing iteration. pages 68–79, 1992.
14. B. S. Gulavani, S. Chakraborty, A. V. Nori, and S. K. Rajamani. Automatically refining abstract interpretations. In *TACAS*, volume 4963 of *LNCS*, pages 443–458. Springer, 2008.
15. L. H. Holley and B. K. Rosen. Qualified data flow problems. In *POPL*, pages 68–82, New York, NY, USA, 1980. ACM Press.
16. G. A. Kildall. A unified approach to global program optimization. In *POPL*, pages 194–206, New York, NY, USA, 1973. ACM.
17. J. C. King. *A Program Verifier*. PhD thesis, 1969.
18. K. L. McMillan. Lazy annotation for program testing and verification. In *CAV*, pages 104–118, 2010.
19. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *JACM*, 53:937–977, 2006.
20. X. Rival and L. Mauborgne. The trace partitioning abstract domain. *TOPLAS*, 29(5):26, 2007.
21. J. A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1):23–41, Jan. 1965.
22. J. a. P. M. Silva and K. A. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.
23. N. Sörensson and A. Biere. Minimizing learned clauses. In *SAT*, pages 237–243, 2009.
24. A. Thakur and T. Reps. A Generalization of Stålmarck's Method. In *SAS*. Springer, 2012.
25. A. Thakur and T. Reps. A method for symbolic computation of abstract operations. In *CAV*. Springer, 2012.
26. M. N. Wegman and F. K. Zadeck. Constant propagation with conditional branches. *TOPLAS*, 13:181–210, April 1991.