A Generalization of Stålmarck's Method

Aditya Thakur[1] and Thomas Reps[1,2]

[1] University of Wisconsin; Madison, WI, USA
[2] GrammaTech, Inc.; Ithaca, NY, USA

Abstract. This paper gives an account of Stålmarck's method for validity checking of propositional-logic formulas, and explains each of the key components in terms of concepts from the field of abstract interpretation. We then use these insights to present a framework for propositional-logic validity-checking algorithms that is parametrized by an abstract domain and operations on that domain. Stålmarck's method is one instantiation of the framework; other instantiations lead to new decision procedures for propositional logic.

1 Introduction

A tool for validity checking of propositional-logic formulas (also known as a tautology checker) determines whether a given formula φ over the propositional variables $\{p_i\}$ is true for all assignments of truth values to $\{p_i\}$. Validity checking is dual to satisfiability checking; validity of φ can be determined using a SAT solver by checking whether $\neg\varphi$ is satisfiable and complementing the answer: $\text{VALIDITY}(\varphi) = \neg\text{SAT}(\neg\varphi)$.

With the advent of SAT-solvers based on conflict-directed clause learning (i.e., CDCL SAT solvers) [10] and their use in a wide range of applications, SAT methods have received increased attention during the last twelve years. Previous to CDCL, a fast validity checker (and hence a fast SAT solver) already existed, due to Stålmarck [12]. Stålmarck's method was protected by Swedish, U.S., and European patents, which may have discouraged experimentation by researchers to improve upon the method [13]. Indeed, one finds relatively few publications that concern Stålmarck's method—some of the exceptions being Harrison [9], Cook and Gonthier [3], and Björk [2].

In this paper, we give a new account of Stålmarck's method by explaining each of the key components in terms of concepts from the field of abstract interpretation [4]. In particular, we show that Stålmarck's method is based on a certain *abstract domain* and a few *operations* on that domain. For the program-analysis community, the abstract-interpretation account explains the principles behind Stålmarck's method in terms of familiar concepts. In the long run, our hope is that a better understanding of Stålmarck's algorithm will lead to better tools, either

- better program-analysis tools that import principles found in Stålmarck's method into program analyzers, or

– improvements to Stålmarck-based validity checkers by (i) incorporating domains other than the ones that have been used in Stålmarck's implemented version of the method, or (ii) improving the method in other ways by incorporating additional techniques from the field of abstract interpretation.

The contributions of the paper can be summarized as follows:

– We explain Stålmarck's method in terms of abstract interpretation [4]—in particular, we show that it is one instance of a more general algorithm.
– The vantage point of abstract interpretation provides new insights on the existing Stålmarck method. In particular, we use it to describe the elements of the Dilemma Rule—the inference rule that distinguishes Stålmarck's method from other approaches to propositional reasoning—as follows:
  Branch of a Proof: In Stålmarck's method, each branch of the proof tree is associated with a so-called *formula relation* [12, §2.8]. In abstract-interpretation terms, each branch is associated with an abstract-domain element. As discussed in §4, the set of formula relations associated with a formula φ under consideration is one particular abstract domain.
  Splitting: The step of splitting the current goal into sub-goals can be expressed in terms of meet ($\sqcap$).
  Application of simple deductive rules: Stålmarck's method applies a set of simple deductive rules after each split. In abstract-interpretation terms, the rules perform a *semantic reduction* [5] by means of a technique called *local decreasing iterations* [7].
  "Intersecting" results: The step of combining the results obtained from an earlier split are described as an "intersection" in Stålmarck's papers. In the abstract-interpretation-based framework, the combining step is the *join* ($\sqcup$) of two abstract-domain values.
  This more general view of Stålmarck's method furnishes insight on when an invocation of the Dilemma Rule fails to make progress in a proof. In particular, both branches of a Dilemma may each succeed (locally) in advancing the state of the proof, but the abstract domain used to represent proof states may not be precise enough to represent the common information when the join of the two branches is performed; consequently, the global state of the proof is not advanced.
– Adopting the abstract-interpretation viewpoint leads to a parametric framework for validity checking parameterized by an abstract domain that supports a small number of operations. The advantages of this approach are:
  • We prove correctness at the framework level, once and for all, instead of for each instantiation.
  • Instantiations of the framework with different abstract domains lead to different decision procedures for propositional logic. Stålmarck's method is the instantiation of our framework in which the abstract domain tracks equivalence relations between subformulas—or, equivalently, 2-variable Boolean affine relations (2-BAR). By instantiating the framework with other abstract domains, such as k-variable Boolean affine re-

$$v_1 \Leftrightarrow (v_2 \vee v_3) \quad (1)$$
$$v_2 \Leftrightarrow (a \wedge b) \quad (2)$$
$$v_3 \Leftrightarrow (\neg a \vee \neg b) \quad (3)$$

Fig. 1: Integrity constraints corresponding to the formula $\varphi = (a \wedge b) \vee (\neg a \vee \neg b)$. The root variable of φ is v_1.

$$\frac{p \Leftrightarrow (q \vee r) \qquad p \equiv \mathbf{0}}{q \equiv \mathbf{0} \qquad r \equiv \mathbf{0}}\ \textsc{Or1}$$

$$\frac{p \Leftrightarrow (q \wedge r) \qquad q \equiv \mathbf{1} \qquad r \equiv \mathbf{1}}{p \equiv \mathbf{1}}\ \textsc{And1}$$

Fig. 2: Propagation rules.

$v_1 \equiv \mathbf{0}$	... by assumption
$v_2 \equiv \mathbf{0},\ v_3 \equiv \mathbf{0}$	... by rule Or1 using Eqn. (1)
$\neg a \equiv \mathbf{0},\ \neg b \equiv \mathbf{0}$	... by rule Or1 using Eqn. (3)
$a \equiv \mathbf{1},\quad b \equiv \mathbf{1}$	... interpretation of logical negation
$v_2 \equiv \mathbf{1}$	... by rule And1 using Eqn. (2)
$\mathbf{0} \equiv \mathbf{1}$	... $v_2 \equiv \mathbf{0}, v_2 \equiv \mathbf{1}$

Fig. 3: Proof that φ is valid.

lations (k-BAR) and 2-variable Boolean inequality relations (2-BIR), we obtain more powerful decision procedures.

Organization. The remainder of the paper is organized as follows: §2 reviews Stålmarck's algorithm, and presents our generalized framework at a semi-formal level. §3 defines terminology and notation. §4 describes Stålmarck's method using abstract-interpretation terminology and presents the general framework. §5 describes instantiations of the framework that result in new decision procedures. §6 discusses related work. §7 concludes. Proofs are presented in App. A.

2 Overview

In this section, we first review Stålmarck's method with the help of a few examples. We then present our generalized framework at a semi-formal level. The algorithms that we give are intended to clarify the principles behind Stålmarck's method, rather than represent the most efficient implementation. (In the terminology of Datalog, our algorithms use "naive evaluation", whereas more efficient implementations would use "semi-naive evaluation" [15, Algs. 3.3 and 3.4].)

2.1 Stålmarck's Method

Consider the tautology $\varphi = (a \wedge b) \vee (\neg a \vee \neg b)$. It expresses the pigeonhole principle for two pigeons ($a \wedge b$) and one hole ($\neg a \vee \neg b$). Ex. 1 below shows that the simpler component of the two components of Stålmarck's method (application of "simple deductive rules") is sufficient to establish that φ is valid.

Example 1. We use $\mathbf{0}$ and $\mathbf{1}$ to denote the propositional constants false and true, respectively. Propositional variables, negations of propositional variables, and propositional constants are referred to collectively as *literals*. Stålmarck's method manipulates *formula relations*, which are equivalence relations over literals. A formula relation R will be denoted by $\equiv_R$, although we generally omit

Algorithm 1: propagate$(j, R_1, R, \mathcal{I})$

```
1 R_2 = ApplyRule[I](j, R_1)
2 return Close(R ∪ R_2)
```

Algorithm 2: 0-saturation$(R, \mathcal{I})$

```
1 repeat
2     R' ← R
3     foreach j ∈ I, R_1 ⊆ R do
4         R ← propagate(j, R_1, R, I)
5 until (R = R') ∥ contradiction(R)
6 return R
```

Algorithm 3: 1-saturation$(R, \mathcal{I})$

```
1  repeat
2      R' ← R
3      foreach v_i, v_j such that
       v_i ≡ v_j ∉ R and v_i ≡ ¬v_j ∉ R
       do
4          R_1 ← Close(R ∪ {v_i ≡ v_j})
5          R_2 ← Close(R ∪ {v_i ≡ ¬v_j})
6          R'_1 ← 0-saturation(R_1)
7          R'_2 ← 0-saturation(R_2)
8          R ← R'_1 ∩ R'_2
9  until (R = R') ∥ contradiction(R)
10 return R
```

Algorithm 4: k-saturation$(R, \mathcal{I})$

```
1  repeat
2      R' ← R
3      foreach v_i, v_j such that
       v_i ≡ v_j ∉ R and v_i ≡ ¬v_j ∉ R do
4          R_1 ← Close(R ∪ {v_i ≡ v_j})
5          R_2 ← Close(R ∪ {v_i ≡ ¬v_j})
6          R'_1 ← (k–1)-saturation(R_1, I)
7          R'_2 ← (k–1)-saturation(R_2, I)
8          R ← R'_1 ∩ R'_2
9  until (R = R') ∥ contradiction(R)
10 return R
```

Algorithm 5: k-Stålmarck(φ)

```
1 (v_φ, I) ← integrity(φ)
2 R ← {v_φ ≡ 0}
3 R' ← k-saturation(R, I)
4 if R' = 0 ≡ 1 then return valid
5 else return unknown
```

Fig. 4: Stålmarck's method. The operation Close performs transitive closure on a formula relation after new tuples are added to the relation.

the subscript when R is understood. We use $\mathbf{0} \equiv \mathbf{1}$ to denote the universal (and contradictory) equivalence relation $\{l_i \equiv l_j \mid l_i, l_j \in \text{Literals}\}$.

Stålmarck's method first assigns to every subformula of φ a unique Boolean variable, and generates a list of *integrity constraints* as shown in Fig. 1. An *assignment* is a function in $\mathcal{V} \to \{0, 1\}$, where $\mathcal{V}$ is the set of propositional variables. The integrity constraints limit the set of assignments in which we are interested. Here the integrity constraints encode the structure of the formula.

Stålmarck's method establishes the validity of the formula φ by showing that $\neg\varphi$ leads to a contradiction (which means that $\neg\varphi$ is unsatisfiable). Thus, the second step of Stålmarck's method is to create a formula relation that contains the assumption $v_1 \equiv \mathbf{0}$. Fig. 2 lists some *propagation rules* that enable Stålmarck's method to refine a formula relation by inferring new equivalences. For instance, rule Or1 says that if $p \Leftrightarrow (q \vee r)$ is an integrity constraint and $p \equiv \mathbf{0}$ is in the formula relation, then $q \equiv \mathbf{0}$ and $r \equiv \mathbf{0}$ can be added to the formula relation.

Fig. 3 shows how, starting with the assumption $v_1 \equiv \mathbf{0}$, the propagation rules derive the explicit contradiction $\mathbf{0} \equiv \mathbf{1}$, thus proving that φ is valid. □

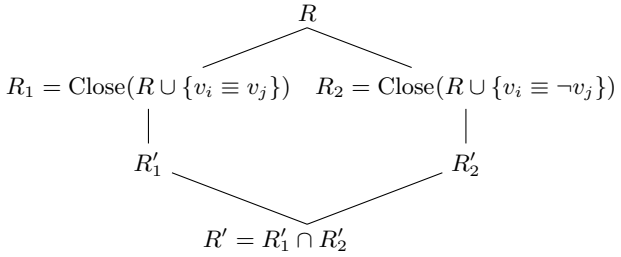

Fig. 5: The Dilemma Rule.

Ex. 1 illustrates several aspects of Stålmarck's method. By starting with the assumption that the formula is equivalent to false, Stålmarck's method derives new equivalences using simple deductive rules. These rules use the currently known equivalences and an integrity constraint to derive new equivalences.

Alg. 1 implements the propagation rules of Fig. 2. Given an integrity constraint $j \in \mathcal{I}$ and a set of equivalences $R_1 \subseteq R$, line (1) calls the function ApplyRule, which instantiates and applies the derivation rules of Fig. 2 and returns the deduced equivalences in R_2. The new equivalences in R_2 are incorporated into R and the transitive closure of the resulting equivalence relation is returned. We implicitly assume that if Close derives a contradiction then it returns $\mathbf{0} \equiv \mathbf{1}$. Alg. 2 describes *0-saturation*, which calls propagate repeatedly until no new information is deduced, or a contradiction is derived. If a contradiction is derived, then the given formula is proved to be valid.

Unfortunately, 0-saturation is not always sufficient.

Example 2. Consider the tautology $\psi = (a \wedge (b \vee c)) \Leftrightarrow ((a \wedge b) \vee (a \wedge c))$, which expresses the distributivity of $\wedge$ over $\vee$. The integrity constraints for ψ are:

$$u_1 \Leftrightarrow (u_2 \Leftrightarrow u_3) \qquad u_2 \Leftrightarrow (a \wedge u_4) \qquad u_3 \Leftrightarrow (u_5 \vee u_6)$$
$$u_4 \Leftrightarrow (b \vee c) \qquad u_5 \Leftrightarrow (a \wedge b) \qquad u_6 \Leftrightarrow (a \wedge c)$$

The root variable of ψ is u_1. Assuming $u_1 \equiv \mathbf{0}$ and then performing 0-saturation does not result in a contradiction; all we can infer is $u_2 \equiv \neg u_3$.

To prove that ψ is a tautology, we need to use the Dilemma Rule, which is a special type of branching and merging rule. It is shown schematically in Fig. 5. After two literals v_i and v_j are chosen, the current formula relation R is split into two formula relations, based on whether we assume $v_i \equiv v_j$ or $v_i \equiv \neg v_j$, and transitive closure is performed on each variant of R. Next, the two relations are 0-saturated, which produces the two formula relations R'_1 and R'_2. Finally, the two proof branches are merged by intersecting the set of tuples in R'_1 and R'_2. The correctness of the Dilemma Rule follows from the fact that equivalences derived from *both* of the (individual) assumptions $v_i \equiv v_j$ and $v_i \equiv \neg v_j$ hold irrespective of whether $v_i \equiv v_j$ holds or whether $v_i \equiv \neg v_j$ holds.

The Dilemma Rule is applied repeatedly until no new information is deduced by a process called *1-saturation*, shown in Alg. 3. 1-saturation uses two literals v_i and v_j, and splits the formula relation with respect to $v_i \equiv v_j$ and $v_i \equiv \neg v_j$ (lines (4) and (5)). 1-saturation finds a contradiction when both 0-saturation branches identify contradictions (in which case $R = R'_1 \cap R'_2$ equals $\mathbf{0} \equiv \mathbf{1}$). The formula ψ in Ex. 2 can be proved valid using 1-saturation, as shown in

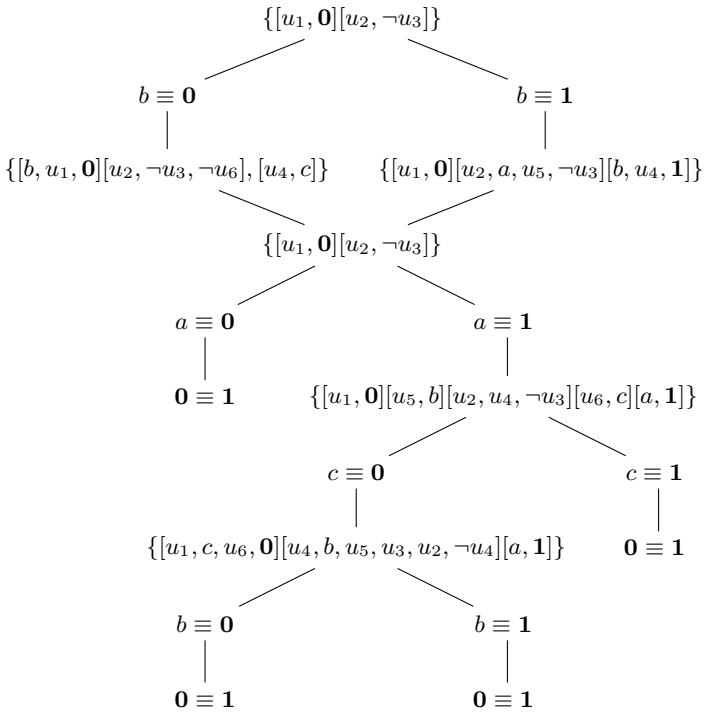

Fig. 6: Sequence of Dilemma Rules in a 1-saturation proof that ψ is valid. (Details of 0-saturation steps omitted.)

Fig. 6. The first application of the Dilemma Rule, which splits on the value of b, does not make any progress; i.e., no new information is obtained after the intersection. The next two applications of the Dilemma Rule, which split on the value of a and c, each eliminate one of their proof branches. Finally, splitting on the variable b leads to a contradiction on both branches. □

There are many possible strategies for choosing which equivalence to split on for the Dilemma Rule during 1-saturation. The choices made by a strategy have an impact on the completeness and efficiency of the decision procedure. For expository purposes, Alg. 3 adopts a simple yet complete strategy. We revisit this issue in App. B.

Unfortunately 1-saturation may not be sufficient to prove certain tautologies. The 1-saturation procedure can be generalized to the k-saturation procedure shown in Alg. 4. Stålmarck's method (Alg. 5) is structured as a semi-decision procedure for validity checking. The actions of the algorithm are parameterized by a certain parameter k that is fixed by the user. For a given tautology, if k is large enough Stålmarck's method can prove validity, but if k is too small the answer returned is "unknown". In the latter case, one can increment k and try again. However, for each k, $(k+1)$-saturation is significantly more expensive than k-saturation: the running time of Alg. 5 as a function of k is $O(|\varphi|^k)$ [12].

$$\frac{a \Leftrightarrow (b \Rightarrow c)}{c \leq a \qquad \neg a \leq b}\ \textsc{Imp1} \qquad\qquad \frac{a \Leftrightarrow (b \Leftrightarrow c) \qquad a \leq \mathbf{0}}{b \leq \neg c \qquad \neg c \leq b}\ \textsc{Iff1}$$

$$\frac{a \Leftrightarrow (b \Rightarrow c) \qquad \mathbf{1} \leq b \qquad c \leq \mathbf{0}}{a \leq 0}\ \textsc{Imp2}$$

Fig. 7: Examples of propagation rules for inequality relations on literals.

$w_1 \leq \mathbf{0}$	... by assumption
$w_2 \leq \neg w_3,\ \neg w_3 \leq w_2$	... Rule IFF1 on $w_1 \Leftrightarrow (w_2 \Leftrightarrow w_3)$
$q \leq w_2,\quad \neg w_2 \leq p$	... Rule IMP1 on $w_2 \Leftrightarrow (p \Rightarrow q)$
$q \leq \neg w_3$	... $q \leq w_2, w_2 \leq \neg w_3$
$w_3 \leq p$	... $w_2 \leq \neg w_3$ implies $w_3 \leq \neg w_2, \neg w_2 \leq p$
$\neg p \leq w_3,\ \neg w_3 \leq \neg q$	... Rule IMP1 on $w_3 \Leftrightarrow (\neg q \Rightarrow \neg p)$
$q \leq \mathbf{0}$	... $\neg w_3 \leq \neg q$ implies $q \leq w_3, q \leq \neg w_3$
$\mathbf{1} \leq p$	... $w_3 \leq p, \neg p \leq w_3$ implies $\neg w_3 \leq p$
$w_2 \leq \mathbf{0},$	... Rule IMP2 on $w_2 \Leftrightarrow (p \Rightarrow q)$
$w_3 \leq \mathbf{0}$	... Rule IMP2 on $w_3 \Leftrightarrow (\neg q \Rightarrow \neg p)$
$\mathbf{1} \leq \mathbf{0}$	... $w_2 \leq \neg w_3, \neg w_3 \leq w_2, w_2 \leq \mathbf{0}, w_3 \leq \mathbf{0}$

Fig. 8: 0-saturation proof that χ is valid, using inequality relations on literals.

Each equivalence relation that arises during Stålmarck's method can be viewed as an *abstraction* of a set of variable assignments. More precisely, at any moment during a proof there are some number of open branches. Each branch B_i has its own equivalence relation R_i, which represents a set of variable assignments A_i. Overall, the proof represents the set of assignments $\bigcup_i A_i$, which is a superset of the set of variable assignments that can falsify the formula. Validity is established by showing that the set $\bigcup_i A_i$ equals $\emptyset$.

2.2 Generalizing Stålmarck's Method

Instead of computing an equivalence relation $\equiv$ on literals, let us compute an inequality relation $\leq$ between literals. Fig. 7 shows a few of the propagation rules that deduce inequalities. Because (i) an equivalence $a \equiv b$ can be represented using two inequality constraints, $a \leq b$ and $b \leq a$, (ii) an inequivalence $a \not\equiv b$ can be treated as an equivalence $a \equiv \neg b$, and (iii) $a \leq b$ cannot be represented with any number of equivalences, inequality relations are strictly more expressive than equivalence relations. Thus, we might expect that using inequality relations in Stålmarck's method would result in a more powerful proof procedure. Ex. 3 confirms that this is the case.

Example 3. Consider the formula $\chi = (p \Rightarrow q) \Leftrightarrow (\neg q \Rightarrow \neg p)$. The corresponding integrity constraints are $w_1 \Leftrightarrow (w_2 \Leftrightarrow w_3)$, $w_2 \Leftrightarrow (p \Rightarrow q)$, and $w_3 \Leftrightarrow (\neg q \Rightarrow \neg p)$. The root variable of χ is w_1. Using formula relations (i.e., equivalence relations over literals), Stålmarck's method finds a 1-saturation proof that χ is valid. In contrast, using inequality relations, a Stålmarck-like algorithm finds a 0-saturation proof. The proof starts by assuming that $w_1 \leq \mathbf{0}$. As shown in Fig. 8, 0-saturation using the propagation rules of Fig. 7 results in the contradiction $\mathbf{1} \leq \mathbf{0}$. □

The preceding example illustrates that Stålmarck's method can be made more powerful by using a more precise abstraction. That is, when you plug in a more precise abstraction, a proof may be possible with a lower value of k. This observation raises the following questions:

1. What other abstractions can be used to increase precision?
2. Given an abstraction, how do we come up with the propagation rules?
3. How do we split the current abstraction at the start of the Dilemma Rule?
4. How do we perform the merge at the end of the Dilemma Rule?
5. How do we guarantee that the above operations result in a sound and complete decision procedure?

Abstract interpretation provides the appropriate tools to answer these questions.

3 Terminology and Notation

3.1 Propositional Logic

We write propositional formulas over a set of propositional variables $\mathcal{V}$ using the propositional constants **0** and **1**, the unary connective $\neg$, and the binary connectives $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\oplus$. Propositional variables, negations of propositional variables, and propositional constants are referred to collectively as *literals*.

The semantics of propositional logic is defined in the standard way:

Definition 1. *An* **assignment** *a is a (finite) function in $\mathcal{V} \to \{0,1\}$. Given a formula φ over the propositional variables $x_1, \ldots, x_n$ and an assignment a that is defined on (at least) $x_1, \ldots, x_n$, the* **meaning** *of φ with respect to a, denoted by $[\![\varphi]\!](a)$, is the truth value in $\{0,1\}$ defined inductively as follows:*

$$
\begin{array}{lll}
[\![\mathbf{0}]\!](a) = 0 & [\![\neg\varphi]\!](a) = 1 - [\![\varphi]\!](a) & [\![\varphi_1 \Rightarrow \varphi_2]\!](a) = ([\![\varphi_1]\!](a) \leq [\![\varphi_2]\!](a)) \\
[\![\mathbf{1}]\!](a) = 1 & [\![\varphi_1 \wedge \varphi_2]\!](a) = \min([\![\varphi_1]\!](a), [\![\varphi_2]\!](a)) & [\![\varphi_1 \Leftrightarrow \varphi_2]\!](a) = ([\![\varphi_1]\!](a) = [\![\varphi_2]\!](a)) \\
[\![x_i]\!](a) = a(x_i) & [\![\varphi_1 \vee \varphi_2]\!](a) = \max([\![\varphi_1]\!](a), [\![\varphi_2]\!](a)) & [\![\varphi_1 \oplus \varphi_2]\!](a) = ([\![\varphi_1]\!](a) \neq [\![\varphi_2]\!](a))
\end{array}
$$

Assignment a **satisfies** *φ, denoted by $a \models \varphi$, iff $[\![\varphi]\!](a) = 1$. Formula φ is* **satisfiable** *if there exists a such that $a \models \varphi$; φ is* **valid** *if for all a, $a \models \varphi$.*

We overload the notation $[\![\cdot]\!]$ as follows: $[\![\varphi]\!]$ means $\{a \mid a : \mathcal{V} \to \{0,1\} \wedge a \models \varphi\}$. Thus, we can think of φ as a device for accepting a set of assignments. Given a finite set of formulas $\Phi = \{\varphi_i\}$, $[\![\Phi]\!]$ means $\bigcap_i [\![\varphi_i]\!]$.

3.2 Abstract Domains

In this paper, the concrete domain $\mathcal{C}$ is $\mathbb{P}(\mathcal{V} \to \{0,1\})$. We will work with several abstract domains $\mathcal{A}$, each of which abstracts $\mathcal{C}$ by a Galois connection $\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$. We assume that the reader is familiar with the basic terminology of abstract interpretation [4] ($\bot$, $\top$, $\sqcup$, $\sqcap$, $\sqsubseteq$, α, γ, monotonicity, distributivity, etc.), as well as with the properties of a Galois connection $\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$.

Definition 2. *An element R of the domain of* **equivalence relations** *(Equiv) over the set Literals$[\mathcal{V}]$ formed from Boolean variables $\mathcal{V}$ and their negations represents a set of assignments in $\mathbb{P}(\mathcal{V} \to \{0,1\})$. The special value $\bot_{Equiv}$ represents the empty set of assignments, and will be denoted by "$\mathbf{0} \equiv \mathbf{1}$". Each*

other value $R \in Equiv$ is an equivalence relation on Literals[$\mathcal{V}$]; the concretization $\gamma(R)$ is the set of all assignments that satisfy all the equivalences in R. The ordering $a_1 \sqsubseteq_{Equiv} a_2$ means that equivalence relation a_1 is a coarser partition of Literals[$\mathcal{V}$] than a_2. The value $\top_{Equiv}$ is the identity relation, $\{(v,v)|v \in \mathcal{V}\}$, and thus represents the set of all assignments. $R_1 \sqcup R_2$ is the coarsest partition that is finer than both R_1 and R_2.

An alternative way to define the same domain is to consider it as the domain of **two-variable Boolean affine relations** *(2-BAR) over $\mathcal{V}$. Each element $R \in$ 2-BAR is a conjunction of Boolean affine constraints, where each constraint has one of the following forms:*

$$v_i \oplus v_j = \mathbf{0} \quad v_i \oplus v_j \oplus \mathbf{1} = \mathbf{0} \quad v_i = \mathbf{0} \quad v_i \oplus \mathbf{1} = \mathbf{0},$$

which correspond to the respective equivalences

$$v_i \equiv v_j \qquad v_i \equiv \neg v_j \qquad v_i \equiv \mathbf{0} \qquad v_i \equiv \mathbf{1}.$$

The value $\bot_{2\text{-}BAR}$ is any set of unsatisfiable constraints. The value $\top_{2\text{-}BAR}$ is the empty set of constraints. The concretization function $\gamma_{2\text{-}BAR}$, and abstraction function $\alpha_{2\text{-}BAR}$ are:

$$\gamma_{2\text{-}BAR}(R) = \{c \in (\mathcal{V} \to \{0,1\}) \mid R = \textstyle\bigwedge_i r_i \text{ and for all } i, c \models r_i\}$$
$$\alpha_{2\text{-}BAR}(C) = \textstyle\bigwedge\{r \mid \text{for all } c \in C, c \models r\}$$

For convenience, we will continue to use equivalence notation ($\equiv$) in examples that use 2-BAR, rather than giving affine relations ($\oplus$).

Definition 3. *An element of the* **Cartesian domain** *represents a set of assignments in $\mathbb{P}(\mathcal{V} \to \{0,1\})$. The special value $\bot_{Cartesian}$ denotes the empty set of assignments; all other values can be denoted via a 3-valued assignment in $\mathcal{V} \to \{0,1,*\}$. The third value "$*$" denotes an unknown value, and the values 0, 1, $*$ are ordered so that $0 \sqsubset *$ and $1 \sqsubset *$.*

The partial ordering $\sqsubseteq$ on 3-valued assignments is the pointwise extension of $0 \sqsubset *$ and $1 \sqsubset *$, and thus $\top_{Cartesian} = \lambda w.*$ and $\sqcup_{Cartesian}$ is pointwise join. The concretization function $\gamma_{Cartesian}$, and abstraction function $\alpha_{Cartesian}$ are:

$$\gamma_{Cartesian}(A) = \{c \in (\mathcal{V} \to \{0,1\}) \mid c \sqsubseteq A\}$$
$$\alpha_{Cartesian}(C) = \lambda w. \bigsqcup\{c(w) \mid c \in C\}$$

We will denote an element of the Cartesian domain as a mapping, e.g., $[p \mapsto 0, q \mapsto 1, r \mapsto *]$, or $[0,1,*]$ if p, q, and r are understood.

Definition 4. *Given an abstract domain $\mathcal{A}$, an element $a \in \mathcal{A}$ is a* **co-atom** *if for all $A \in \mathcal{A}$ such that $a \sqsubseteq A \sqsubseteq \top_{\mathcal{A}}$ either $a = A$ or $A = \top_{\mathcal{A}}$.*

The co-atoms for 2-BAR are all elements that consist of a single constraint. Let $A[v \leftarrow b]$ be the assignment-update operation that maps A to $\lambda w.(w = v)\,?\,b : A(w)$. The set of co-atoms for the Cartesian domain is the set of "single-point" partial assignments $\{\top_{\text{Cartesian}}[v \leftarrow 0]\} \cup \{\top_{\text{Cartesian}}[v \leftarrow 1]\}$. In fact, the "single-point" partial assignments are a subset of the co-atoms for many abstract domains; we use $\top_{\mathcal{A}}[v \leftarrow 0]$ and $\top_{\mathcal{A}}[v \leftarrow 1]$ to denote the co-atoms in $\mathcal{A}$ that correspond to "single-point" partial assignments that set v to 0 and 1, respectively.

Local Decreasing Iterations. Local decreasing iterations [7] is a technique that is ordinarily used for improving precision during the abstract interpretation of a program. During an iterative fixed-point-finding analysis, the technique of local decreasing iterations is applied at particular points in the program, such as, e.g., the interpretation of the true branch of an if-statement whose branch condition is φ. The operation that needs to be performed is the application of the abstract transformer for assume(φ). As the name "local decreasing iterations" indicates, a purely local iterative process repeatedly applies the operator assume(φ) either until some precision criterion or resource bound is attained, or a (local) fixed point is reached. The key theorem is stated as follows:

Theorem 1. *([7, Thm. 2]) An operator τ is a* lower closure operator *if it is monotonic, idempotent ($\tau \circ \tau = \tau$), and reductive ($\tau \sqsubseteq \lambda x.x$).*

Let τ be a lower closure operator on $\mathcal{A}$; let $(\tau_1, \ldots, \tau_k)$ be a k-tuple of reductive operators on $\mathcal{A}$, each of which over-approximates ($\sqsupseteq$) τ; and let $(u_n)_{n \in N}$ be a sequence of elements in $[1, \ldots, k]$. Then the sequence of reductive operators on $\mathcal{A}$ defined by:

$$\eta_0 = \tau_{u_0} \qquad \eta_{n+1} = \tau_{u_{n+1}} \circ \eta_n$$

is decreasing and each of its elements over-approximates τ.

Example 4. The propagation rules of Fig. 2 can be recast in terms of reductive operators that refine an element R of the 2-BAR domain as follows:

Operator	Derived from
$\tau_1(R) = R \cup ((v_1 \equiv \mathbf{0} \in R)\ ?\ \{v_2 \equiv \mathbf{0}, v_3 \equiv \mathbf{0}) : \emptyset)$	$v_1 \Leftrightarrow (v_2 \vee v_3) \in \mathcal{I}$
$\tau_2(R) = R \cup ((\{a \equiv \mathbf{1}, b \equiv \mathbf{1}\} \subseteq R)\ ?\ \{v_2 \equiv \mathbf{1}) : \emptyset)$	$v_2 \Leftrightarrow (a \wedge b) \in \mathcal{I}$
$\tau_3(R) = R \cup ((\{v_3 \equiv \mathbf{0}) \in R)\ ?\ \{a \equiv \mathbf{1}, b \equiv \mathbf{1}\} : \emptyset)$	$v_3 \Leftrightarrow (\neg a \wedge \neg b) \in \mathcal{I}$
$\tau_4(R) = (\{v_2 \equiv \mathbf{0}, v_2 \equiv \mathbf{1}) \subseteq R)\ ?\ \mathbf{0} \equiv \mathbf{1} : R$	

The operators τ_1, τ_2, and τ_3 correspond to the three integrity constraints shown in Fig. 1. The derivation described in Fig. 3 can now be stated as $\tau_4(\tau_2(\tau_3(\tau_1(\{v_1 \equiv \mathbf{0}\})))) = (\tau_4 \circ \tau_2 \circ \tau_3 \circ \tau_1)(\{v_1 \equiv \mathbf{0}\})$, which results in the abstract state $\mathbf{0} \equiv \mathbf{1}$.

Note that the operators given above are specialized for the given integrity constraints, while the rules in Fig. 2 are stated in a parametric fashion. We will revisit this issue in App. B. □

4 The Generalized Framework

In this section, we map the concepts used in Stålmarck's method to concepts used in abstract interpretation, as summarized in Tab. 1. The payoff is that we obtain a parametric framework for propositional validity-checking algorithms (Alg. 9) that can be instantiated in different ways by supplying different abstract domains. The assumptions of our framework are rather minimal:

1. There is a Galois connection $\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$ between $\mathcal{A}$ and the concrete domain of assignments $\mathcal{C} = \mathbb{P}(\mathcal{V} \to \{0, 1\})$.

2. $\mathcal{A}$ is at least as expressive as the Cartesian domain (Defn. 3); that is, for all $A_c \in$ Cartesian, there exists $A \in \mathcal{A}$ such that $\gamma_{\text{Cartesian}}(A_c) = \gamma_{\mathcal{A}}(A)$. In particular, all co-atoms of the Cartesian domain are co-atoms in $\mathcal{A}$, and we begin validity checking from the element $\top_{\mathcal{A}}[v_\varphi \leftarrow 0]$.
3. There is an algorithm to perform the join of arbitrary elements of the domain; that is, for all $A_1, A_2 \in \mathcal{A}$, there is an algorithm that produces $A_1 \sqcup A_2$.
4. The set of co-atoms in $\mathcal{A}$ is closed under negation; that is, for each co-atom $a_1 \in \mathcal{A}$ there exists a co-atom $a_2 \in \mathcal{A}$ such that $\gamma(a_2) = \gamma(\top_{\mathcal{A}}) - \gamma(a_1)$. We denote abstract negation by $\neg^\sharp$; i.e., $a_2 = \neg^\sharp a_1$ and $a_1 = \neg^\sharp a_2$.
5. There is an algorithm to perform meet with a co-atom; that is, for all $A \in \mathcal{A}$ and each co-atom $a \in \mathcal{A}$, there is an algorithm that produces $A \sqcap a$.

Note that because the concrete domain $\mathcal{C}$ is over a finite set of Boolean variables, the abstract domain $\mathcal{A}$ has no infinite descending chains. It is not hard to show that 2-BAR meets assumptions (1)–(5). The standard version of Stålmarck's method (§2.1) is the instantiation of the framework presented in this section with the abstract domain 2-BAR.

At any moment during our generalization of Stålmarck's method, each open branch B_i represents a set of variable assignments $C_i \in \mathcal{C}$ such that $\bigcup_i C_i \supseteq [\![\neg\varphi]\!]$. That is, each branch B_i represents an abstract state $A_i \in \mathcal{A}$ such that $\bigcup_i \gamma(A_i) \supseteq [\![\neg\varphi]\!]$. Let $\bar{A} = \bigsqcup_i A_i$. Then $\bar{A}$ is sound, i.e., $\gamma(\bar{A}) \supseteq \bigcup_i \gamma(A_i) \supseteq [\![\neg\varphi]\!]$. The net result of the proof rules is to derive a *semantic reduction* $\bar{A}'$ of $\bar{A}$ with respect to the integrity constraints $\mathcal{I}$; that is, $\gamma(\bar{A}') \cap [\![\mathcal{I}]\!] = \gamma(\bar{A}) \cap [\![\mathcal{I}]\!]$, and $\bar{A}' \sqsubseteq \bar{A}$. If the abstract state $\bot_{\mathcal{A}}$ is derived, then the formula is proved valid.

It is possible to give a variant of our generalized Stålmarck's method that only needs to perform abstract negation of the co-atoms inherited from the Cartesian domain. In this situation, we can drop Assumption (4): because the concretizations of $\top_{\mathcal{A}}[v \leftarrow 0]$ and $\top_{\mathcal{A}}[v \leftarrow 1]$ are disjoint, Assumption (2) already ensures that the co-atoms inherited from the Cartesian domain are closed under negation.

Generalized Propagation Rules. The Propagation Rules aim to refine the abstract state by assuming a subset J of the integrity constraints $\mathcal{I}$. It is possible to list all the propagation rules in the style of Fig. 2 for the 2-BAR domain, but this manual approach may not be feasible when dealing with more complicated domains. Alg. 6 describes the generalized procedure for implementing propagation rules for an abstract domain $\mathcal{A}$ in terms of α, γ, and $\sqcap$. This procedure is sound if the abstract value $\bar{A}$ returned satisfies $\gamma(\bar{A}) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$. Further-

Stålmarck's Method	Abstract-Interpretation Concept
Equivalence relation	Abstract-domain element
Propagation rule	Sound reductive operator
0-saturation	Local decreasing iterations
Split	$\sqcap$ with a co-atom a and its abstract negation $\neg^\sharp a$
Intersection ($\cap$)	join ($\sqcup$)

Table 1: Abstract-interpretation account of Stålmarck's method.

Algorithm 6: $\text{propagate}_{\mathcal{A}}(J, A_1, A, \mathcal{I})$

```
1 requires(J ⊆ I ∧ A_1 ⊒ A)
2 return A ⊓ α([[J]] ∩ γ(A_1))
```

Algorithm 7: $0\text{-saturation}_{\mathcal{A}}(A, \mathcal{I})$

```
1 repeat
2     A' ← A
3     foreach J ⊆ I, A_1 ⊒ A such
      that |voc(J) ∪ voc(A_1)| < ε) do
4         A ← propagate_A(J, A_1, A, I)
5 until (A = A') ∥ A = ⊥_A
6 return A
```

Algorithm 8: $\text{k-saturation}_{\mathcal{A}}(A, \mathcal{I})$

```
1  repeat
2      A' ← A
3      foreach co-atom a ∈ A such
       that a ⋣ A and ¬♯a ⋣ A do
4          A_1 ← A ⊓ a
5          A_2 ← A ⊓ ¬♯a
6          A'_1 ← (k–1)-saturation_A(A_1, I)
7          A'_2 ← (k–1)-saturation_A(A_2, I)
8          A ← A'_1 ⊔ A'_2
9  until (A = A') ∥ A = ⊥_A
10 return A
```

Fig. 9: The algorithms that make up the parametric framework for propositional validity checking. Alg. 7 is parameterized on ϵ, which bounds the number of variables considered in each propagation step.

more, to guarantee progress we have to show that Alg. 6 implements a reductive operator, i.e., $\bar{A} \sqsubseteq A$.

Theorem 2. *Let* $\bar{A} := \text{propagate}_{\mathcal{A}}(J, A_1, A, \mathcal{I})$ *with* $J \subseteq \mathcal{I}$ *and* $A_1 \sqsupseteq A$. *Then* $\gamma(\bar{A}) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$ *and* $\bar{A} \sqsubseteq A$. □

Example 5. Let us apply Alg. 6 with $J = \{v_1 \Leftrightarrow (v_2 \vee v_3)\}$, $A_1 = \{v_1 \equiv \mathbf{0}\}$ and $A = \{v_1 \equiv \mathbf{0}, v_4 \equiv \mathbf{0}\}$. To save space, we use 3-valued assignments to represent the concrete states of assignments to $v_1, \ldots, v_4$.

$$
\begin{aligned}
[\![J]\!] &= \{(1,0,1,*), (1,1,0,*), (1,1,1,*), (0,0,0,*)\} \\
\gamma(A_1) &= \{(0,*,*,*)\} \\
C = [\![J]\!] \cap \gamma(A_1) &= \{(0,0,0,*)\} \\
\alpha(C) &= \{v_1 \equiv \mathbf{0}, v_2 \equiv \mathbf{0}, v_3 \equiv \mathbf{0}\} \\
\bar{A} = A \sqcap \alpha(C) &= \{v_1 \equiv \mathbf{0}, v_2 \equiv \mathbf{0}, v_3 \equiv \mathbf{0}, v_4 \equiv \mathbf{0}\}
\end{aligned}
$$

Thus, the value $\bar{A}$ computed by Alg. 6 is exactly the abstract value that can be deduced by propagation rule Or1 of Fig. 2. □

Generalized 0-Saturation. Alg. 7 shows the generalized 0-saturation procedure that repeatedly applies the propagation rules (line (4)) using a subset of the integrity constraints (line (3)), until no new information is derived or a contradiction is found (line (5)); $\text{voc}(\varphi)$ denotes the set of φ's propositional variables. To prove soundness we show that the abstract value $\bar{A}$ returned by Alg. 7 satisfies $\gamma(\bar{A}) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$.

Theorem 3. *For all* $A \in \mathcal{A}$, $\gamma(0\text{-saturation}_{\mathcal{A}}(A, \mathcal{I})) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$. □

Generalized k-Saturation. Alg. 8 describes the generalized k-saturation procedure that repeatedly applies the generalized Dilemma Rule. The generalized

Algorithm 9: k-Stålmarck$_{\mathcal{A}}(\varphi)$

1 $(v_\varphi, \mathcal{I}) \leftarrow \text{integrity}(\varphi)$
2 $A \leftarrow \top_{\mathcal{A}}[v_\varphi \leftarrow 0]$
3 $A' \leftarrow \text{k-saturation}_{\mathcal{A}}(A, \mathcal{I})$
4 **if** $A' = \bot_{\mathcal{A}}$ **then return** *valid*
5 **else return** *unknown*

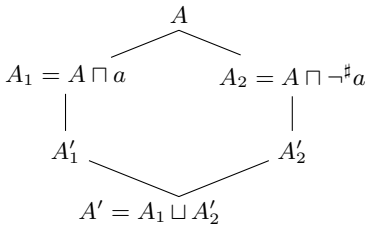

Fig. 10: Generalized Dilemma Rule

Dilemma Rule, shown schematically in Fig. 10, splits the current abstract state A into two abstract states A_1 and A_2 using a co-atom a and its abstract negation $\neg^\sharp a$. To merge the two branches of the generalized Dilemma Rule, we perform a join of the abstract states derived in each branch. The next theorem proves that Alg. 8, which utilizes this generalized Dilemma Rule, is sound.

Theorem 4. *For all* $A \in \mathcal{A}$, $\gamma(\text{k-saturation}_{\mathcal{A}}(A, \mathcal{I})) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$. □

Generalized k-Stålmarck. Alg. 9 describes our generalization of Stålmarck's method, which is parameterized by an abstract domain $\mathcal{A}$. Line (1) converts the formula φ into the integrity constraints $\mathcal{I}$, with v_φ representing φ. We have to prove that Alg. 9 returns *valid* when the given formula φ is indeed valid.

Theorem 5. *If* k*-Stålmarck*$_{\mathcal{A}}(\varphi)$ *returns* valid, *then* $[\![\neg\varphi]\!] = \emptyset$. □

Completeness. As we saw in §2, Alg. 9 is not complete for all values of k. However, Alg. 9 is complete if k is large enough. In particular, let $m = |\text{voc}(\mathcal{I})|$. Because the abstract domain $\mathcal{A}$ is at least as expressive as the Cartesian abstract domain, m-Stålmarck$_{\mathcal{A}}(\mathcal{I})$ will eventually try out all 2^m variable assignments. Consequently, m-Stålmarck$_{\mathcal{A}}(\varphi)$ is complete; that is, if m-Stålmarck$_{\mathcal{A}}(\varphi)$ returns *unknown*, then φ is definitely not valid. App. B discusses the efficiency of our generalization of Stålmarck's Method.

5 Instantiations

Stålmarck's method is the instantiation of the framework from §4 with the abstract domain 2-BAR. In this section, we present the details for a few other instantiations of the framework from §4.

Cartesian Domain. The original version of Stålmarck's method [14] did not use equivalence classes of propositional variables (i.e., the abstract domain 2-BAR). Instead, it was based on a weaker abstract domain of partial assignments, or equivalently, the Cartesian domain. It is easy to see that the Cartesian domain meets the requirements of the framework.

1. There is a Galois connection $\mathbb{P}(\mathcal{V} \to \{0,1\}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}}$ Cartesian. The α and γ functions are given in Defn. 3.
2. The Cartesian domain obviously meets the expressiveness condition of being as precise as the Cartesian domain.

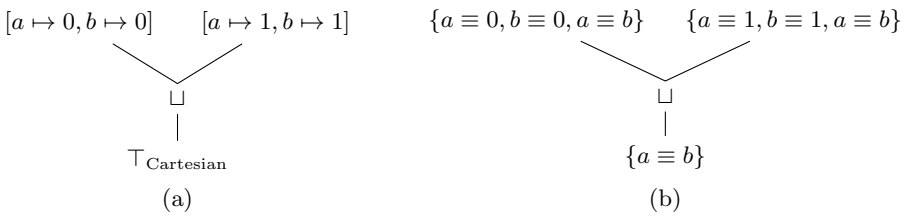

Fig. 11: An example in which the Cartesian domain and 2-BAR start with equivalent information in the respective branches, but (a) the Cartesian domain loses all information at the join, whereas (b) 2-BAR retains the equivalence $a \equiv b$.

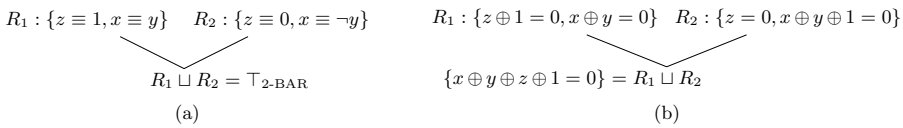

Fig. 12: 3-BAR retains more information at the join than 2-BAR.

3. The partial ordering $\sqsubseteq$ on 3-valued assignments is the pointwise extension of $0 \sqsubset *$ and $1 \sqsubset *$. Thus, $\sqcup_{\text{Cartesian}}$ is pointwise join.
4. The set of co-atoms for the Cartesian domain is the set of "single-point" partial assignments $\{\top_{\text{Cartesian}}[v \leftarrow 0]\} \cup \{\top_{\text{Cartesian}}[v \leftarrow 1]\}$. The abstract negation of $\top_{\text{Cartesian}}[v \leftarrow 0]$ is $\top_{\text{Cartesian}}[v \leftarrow 1]$, and vice versa. Their concretizations are disjoint.
5. Let $a = \top_{\text{Cartesian}}[v \leftarrow b]$ be a co-atom, where $b \in \{0, 1\}$. For all $A \in$ Cartesian, $A \sqcap a = (A(v) = 1 - b)\ ?\ \bot_{\text{Cartesian}} : A[v \leftarrow (A(v) \sqcap b)]$.

Example 6. Fig. 11 presents an example in which the Cartesian domain and 2-BAR start with equivalent information in the respective branches, but the Cartesian domain loses all information at a join, whereas 2-BAR retains an equivalence. Consequently, the original version of Stålmarck's method [14] is less powerful than the standard version of Stålmarck's method that came later [13]. □

Three-Variable Boolean Affine Relations (3-BAR). The abstract domain 3-BAR is defined almost identically to 2-BAR (Defn. 2). In general, a non-bottom element of 3-BAR is a satisfiable conjunction of constraints of the form $\bigoplus_{i=1}^{3}(a_i \wedge x_i) \oplus b = \mathbf{0}$, where $a_i, b \in \{\mathbf{0}, \mathbf{1}\}$.

1. The α and γ functions of the Galois connection $\mathbb{P}(\mathcal{V} \to \{0,1\}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}}$ 3-BAR are stated almost identically to Defn. 2.
$$\gamma_{\text{3-BAR}}(R) = \{c \in (\mathcal{V} \to \{0,1\}) \mid R = \textstyle\bigwedge_i r_i \text{ and for all } i, c \models r_i\}$$
$$\alpha_{\text{3-BAR}}(C) = \textstyle\bigwedge\{r \mid \text{for all } c \in C, c \models r\}$$
2. 3-BAR generalizes 2-BAR, and so is more precise than the Cartesian domain.
3. $A_1 \sqcup A_2$ can be implemented by first extending A_1 and A_2 with all implied constraints, and then intersecting the extended sets.
4. The co-atoms of 3-BAR are the elements that consist of a single constraint.
5. $A \sqcap a$ can be implemented by unioning a's one constraint to those of A.

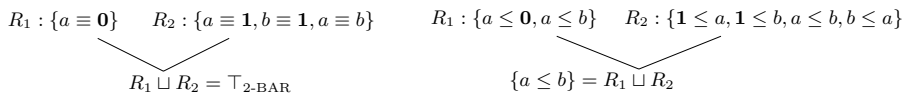

Fig. 13: 2-BIR retains more information at the join than 2-BAR.

Example 7. Fig. 12 presents an example in which 2-BAR and 3-BAR start with equivalent information in the respective branches, but 2-BAR loses all information at a join, whereas 3-BAR retains an affine relation. Consequently, the instantiation of our framework with the 3-BAR domain provides a more powerful proof procedure than the standard version of Stålmarck's method. □

Example 8. 3-BAR can use two integrity constraints and a single co-atom to deduce new information:

$$\frac{p \Leftrightarrow (q \wedge r) \qquad q \Leftrightarrow (s \wedge t) \qquad r \oplus s \oplus t = 0}{p = 0}\ \textsc{And2}$$

$$\frac{p \Leftrightarrow (q \Leftrightarrow r) \qquad q \Leftrightarrow (s \Leftrightarrow t) \qquad r \oplus s \oplus t = 1}{p = 1}\ \textsc{Iff2}$$

In Rule And2 $r \oplus s \oplus t = 0$ implies that the variables r, s, and t cannot be 1 simultaneously, thus ensuring that $p = 0$. □

Two-Variable Boolean Inequality Relations (2-BIR). 2-BIR is yet another constraint domain, and hence defined similarly to 2-BAR and 3-BAR. A non-bottom element of 2-BIR is a satisfiable conjunction of constraints of the form $x \leq y$, $x \leq b$, or $b \leq x$, where $x, y \in \mathcal{V}$ and $b \in \{\mathbf{0}, \mathbf{1}\}$.

1. γ and α are again defined similarly to the ones given in Defn. 2:
$$\gamma_{\text{2-BIR}}(R) = \{c \in (\mathcal{V} \to \{0,1\}) \mid R = \textstyle\bigwedge_i r_i \text{ and for all } i, c \models r_i\}$$
$$\alpha_{\text{2-BIR}}(C) = \textstyle\bigwedge\{r \mid \text{for all } c \in C, c \models r\}$$
2. An equivalence $a \equiv b$ can be represented using two inequality constraints, $a \leq b$ and $b \leq a$, and hence 2-BIR is more precise than 2-BAR, which in turn is more precise than the Cartesian domain.
3. $A_1 \sqcup A_2$ can be implemented by first extending A_1 and A_2 with all implied constraints, and then intersecting the extended sets.
4. The co-atoms of 2-BIR are the elements that consist of a single constraint.
5. $A \sqcap a$ can be implemented by unioning a's one constraint to those of A.

Example 9. Fig. 13 presents an example in which 2-BAR and 2-BIR start with equivalent information in the respective branches, but 2-BAR loses all information at a join, whereas 2-BIR retains a Boolean inequality. Consequently, the instantiation of our framework with the 2-BIR domain provides a more powerful proof procedure than the standard version of Stålmarck's method. □

6 Related Work

Stålmarck's method was patented under Swedish, U.S., and European patents [13]. Up until Jan. 4, 2011, when the U.S. patent expired, researchers interested

in experimenting with Stålmarck's method in the U.S. would have needed to obtain a license for the technology. According to Wikipedia [16],

> "In 2002, the Court of Appeals for the Federal Circuit dramatically limited the scope of the research exemption [for performing research on patented methods without infringing] in Madey v. Duke University, 307 F.3d 1351, 1362 (Fed. Cir. 2002). The court did not reject the defense, but left only a 'very narrow and strictly limited experimental use defense' for 'amusement, to satisfy idle curiosity, or for strictly philosophical inquiry.' The court also precludes the defense . . . [for research] done 'in furtherance of the alleged infringer's legitimate business.' In the case of a research university . . . the court held that . . . the defense was inapplicable."

The tutorial by Sheeran and Stålmarck [12] gives a lucid presentation of the algorithm. Björk [2] explored extensions of Stålmarck's method to first-order logic. (Björk credits Stålmarck with making the first extension of the method in that direction, and mentions an unpublished manuscript of Stålmarck's.)

We recently became aware that Haller and D'Silva [6, 8] are (jointly) engaged in research with similar goals to ours, but in a somewhat different domain. They have given an abstract-interpretation-based account of CDCL SAT solvers [10, 1, 11]. Their work and our work were performed independently and contemporaneously. The only element that we adopted after hearing Haller's presentation at NSAD 2011 [8] was the terminology of co-atoms to formalize the split operation (line (3) of Alg. 8). However, even that requirement could be stated without using co-atoms, because all we require are two elements $A_1, A_2 \in \mathcal{A}$ such that $\gamma(A_1) \cup \gamma(A_2) = \gamma(\top_{\mathcal{A}})$ and $\gamma(A_1) \cap \gamma(A_2) \neq \gamma(\top_{\mathcal{A}})$.

7 Conclusion

As mentioned in §3.2, the original version of Stålmarck's method [14] was based on partial assignments, rather than equivalence classes of propositional variables. In abstract-interpretation terminology, it used the Cartesian domain (Defn. 3); later, the Cartesian domain was replaced with the abstract domain 2-BAR.

In this paper, we elevated this kind of plug-compatibility to a principle: we developed a parametric framework for creating algorithms for propositional validity checking—in which each instantiation has the same basic structure as Stålmarck's algorithm.

We provided answers to the five questions listed in §2.2:

1. We showed that instantiating the framework with either 2-BIR or 3-BAR creates a validity checker that is more powerful than either the original version [14] or the slightly later standard version of Stålmarck's method [13].
2. The propagation rules are sound reductive operators that can be computed using α, γ, and $\sqcap$, as shown in Alg. 6.
3. The split performed at the start of the Dilemma Rule performs a meet with a co-atom a and its abstract negation $\neg^\sharp a$.

4. At the end of the Dilemma Rule, the abstract states from the two branches of the proof are merged by performing a join.
5. In §4, we defined what an abstract domain $\mathcal{A}$ has to provide for it to be used to instantiate our framework.

References

1. R. Bayardo, Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *AAAI*, 1997.
2. M. Björk. First order Stålmarck. *J. Autom. Reasoning*, 42(1):99–122, 2009.
3. B. Cook and G. Gonthier. Using Stålmarck's algorithm to prove inequalities. In *Int. Conf. on Formal Engineering Methods*, 2005.
4. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, 1977.
5. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, 1979.
6. V. D'Silva. Abstractions in satisfiability solvers, 2011. Talk presented at Microsoft Research Cambridge, Sept. 9, 2011.
7. P. Granger. Improving the results of static analyses programs by local decreasing iteration. In *FSTTCS*, 1992.
8. L. Haller. Satisfiability solving as abstract interpretation, 2011. Talk presented at NSAD 2011 (Sept. 13, 2011).
9. J. Harrison. Stålmarck's algorithm as a HOL Derived Rule. In *Int. Conf. on Theorem Proving in Higher Order Logics*, 1996.
10. J. Marques Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD*, 1996.
11. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, 2001.
12. M. Sheeran and G. Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. *FMSD*, 16(1):23–58, 2000.
13. G. Stålmarck. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula, 1989. Swedish Patent No. 467,076 (approved 1992); U.S. Patent No. 5,276,897 (approved 1994); European Patent No. 403,454 (approved 1995).
14. G. Stålmarck and M. Säflund. Modeling and verifying systems and software in propositional logic. In *Int. Conf. on Safety of Computer Controls Systems*, 1990.
15. J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I: Classical Database Systems*. Comp. Sci. Press, Rockville, MD, 1988.
16. Research exemption, Sept. 2011. en.wikipedia.org/wiki/Research_exemption.

A Proofs

Theorem 2 (Soundness of Alg. 6) *Let* $\bar{A} := \text{propagate}_{\mathcal{A}}(J, A_1, A, \mathcal{I})$ *with* $J \subseteq \mathcal{I}$ *and* $A_1 \sqsupseteq A$. *Then* $\gamma(\bar{A}) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$ *and* $\bar{A} \sqsubseteq A$.

Proof.

$$\begin{aligned} J \subseteq \mathcal{I}, A_1 \sqsupseteq A &\Rightarrow [\![J]\!] \cap \gamma(A_1) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A_1) \\ &\Rightarrow \alpha([\![J]\!] \cap \gamma(A_1)) \sqsupseteq \alpha([\![\mathcal{I}]\!] \cap \gamma(A_1)) \\ &\Rightarrow A \sqcap \alpha([\![J]\!] \cap \gamma(A_1)) \sqsupseteq A \sqcap \alpha([\![\mathcal{I}]\!] \cap \gamma(A_1)) \\ &\Rightarrow \gamma(A \sqcap \alpha([\![J]\!] \cap \gamma(A_1))) \supseteq \gamma(A \sqcap \alpha([\![\mathcal{I}]\!] \cap \gamma(A_1))) \\ &\Rightarrow \gamma(A \sqcap \alpha([\![J]\!] \cap \gamma(A_1))) \supseteq \gamma(A_1) \cap \gamma(\alpha([\![\mathcal{I}]\!] \cap \gamma(A_1)))) \\ &\Rightarrow \gamma(A \sqcap \alpha([\![J]\!] \cap \gamma(A_1))) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A_1) \\ &\Rightarrow \gamma(\bar{A}) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A_1) \end{aligned}$$

This argument proves that the value computed by Alg. 6 is sound.

Furthermore, $A \sqcap \alpha([\![J]\!] \cap \gamma(A_1)) \sqsubseteq A$; therefore, $\bar{A} \sqsubseteq A$. This observation proves that Alg. 6 implements a reductive operator. □

Theorem 3 (Soundness of Alg. 7) *For all* $A \in \mathcal{A}$, $\gamma(\text{0-saturation}_{\mathcal{A}}(A, \mathcal{I})) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$.

Proof. In Alg. 7, the propagate procedure is repeatedly called on line (4) in a loop. By Thm. 2, we know that the call to propagate is a sound reductive operator. Thus, by directly applying Thm. 1 we can conclude that the value computed at the end of the loop on lines (3)–(4) is sound.

Termination of the outermost loop follows from the fact that $\mathcal{A}$ has no infinite descending chains and propagate is reductive. □

Theorem 4 (Soundness of Alg. 8) *For all* $A \in \mathcal{A}$, $\gamma(\text{k-saturation}_{\mathcal{A}}(A, \mathcal{I})) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$.

Proof. We prove this via induction on k. Thm. 3 proves the base case when $k = 0$.

To prove the inductive case, assume that Alg. 8 is sound for $k-1$, i.e.,

$$\text{for all } A \in \mathcal{A}, \gamma((\text{k–1})\text{-saturation}_{\mathcal{A}}(A, \mathcal{I})) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A) \tag{4}$$

Let A^0 be the value of A passed as input to Alg. 8, and A^i be the value of A computed at the end of the ith iteration of the loop body consisting of lines (3)–(8); that is, at line (8).

We show via induction that, for each i, $\gamma(A^i) \supseteq [\![\mathcal{I}]\!] \cup \gamma(A^0)$. We first prove the base case for $i = 1$; that is, $A^1 \supseteq [\![\mathcal{I}]\!] \cup \gamma(A^0)$.

From line (4) of Alg. 8, we have $A_1 = A^0 \sqcap a$, which implies $\gamma(A_1) \supseteq \gamma(A^0) \cap \gamma(a)$. Similarly, by line (5), we have $\gamma(A_2) \supseteq \gamma(A^0) \cap \gamma(\neg^\sharp a)$.

Now $\gamma(A_1) \cup \gamma(A_2) \supseteq \gamma(A^0) \cap (\gamma(a) \cup \gamma(\neg^\sharp a))$. Because $\mathcal{A}$ is closed under negation for co-atoms, we have $\gamma(a) \cup \gamma(\neg^\sharp a) = \gamma(\top_{\mathcal{A}})$. Thus, we get

$$\gamma(A_1) \cup \gamma(A_2) \supseteq \gamma(A^0) \tag{5}$$

By lines (6) and (7), we have $A'_1 = (\text{k–1})\text{-saturation}_{\mathcal{A}}(A_1)$ and $A'_2 = (\text{k–1})\text{-saturation}_{\mathcal{A}}(A_2)$.

By the induction hypothesis (Eqn. (4)), we have

$$\begin{aligned}
&\gamma(A'_1) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A_1), \gamma(A'_2) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A_2) \\
&\Rightarrow \gamma(A'_1) \cup \gamma(A'_2) \supseteq [\![\mathcal{I}]\!] \cap (\gamma(A_1) \cup \gamma(A_2)) \\
&\Rightarrow \gamma(A'_1) \cup \gamma(A'_2) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A^0) \text{ (using Eqn. (5))} \\
&\Rightarrow \gamma(A'_1 \sqcup A'_2) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A^0)
\end{aligned}$$

Thus, at line (8), we have $\gamma(A^1) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A^0)$.

An almost identical proof can be used to prove the inductive case for the induction on i. Thus, for all i, $\gamma(A^i) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A^0)$.

Because the value returned by Alg. 8 is the final value computed at line (8), we have proven the inductive case for the induction on k. Thus, by induction, we have shown that for all $A \in \mathcal{A}$, $\gamma(\text{k-saturation}_{\mathcal{A}}(A, \mathcal{I})) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$. □

Theorem 5 (Soundness of Alg. 9) *If k-Stålmarck*$_{\mathcal{A}}(\varphi)$ *returns* valid*, then* $[\![\neg\varphi]\!] = \emptyset$.

Proof. The abstract state A created at line (2) corresponds to the assumption that there exists an assignment that falsifies φ. Thus, $[\![\neg\varphi]\!] = [\![\mathcal{I}]\!] \cap \gamma(A)$. By Thm. 4, we have $\gamma(A') \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$ at line (3). Thus, if $A' = \bot_{\mathcal{A}}$ we know that $[\![\mathcal{I}]\!] \cap \gamma(A) = \emptyset$. Therefore, if Alg. 9 returns *valid*, $[\![\neg\varphi]\!] = \emptyset$. □

B Efficiency

We now describe how some of the efficient implementation techniques that can be used with 2-BAR, as described in [12], can be generalized to other abstract domains. In each case, we express the essence of each technique in terms of abstract interpretation.

On line (3) of Alg. 7, the subsets J and A_1 are chosen so that $|\text{voc}(J) \cup \text{voc}(A_1)|$ is small. Such a choice enables efficient symbolic implementations of the operations used in Alg. 6, viz., implementing truth-table semantics on the limited vocabulary of size ϵ. For instance, it is sufficient to use only a single integrity constraint $j \in \mathcal{I}$ in the propagation rules for 2-BAR. Because J in Alg. 6 consists of a bounded number of integrity constraints, there are only a bounded number of Boolean operators involved in each propagation step. By limiting the size of $\text{voc}(J) \cup \text{voc}(A_1)$ (line (3) of Alg. 7), it is possible to generate automatically a bounded number of propagation-rule schemas to implement line (2) of Alg. 6.

Alg. 7 should choose J and A_1 such that $\text{voc}(J) \cap \text{voc}(A_1) \neq \emptyset$ to avoid useless calls to propagate that do not lead to new information being deduced. Furthermore, 0-saturation can be made more efficient by using semi-naive evaluation [15, Alg. 3.4].

In Alg. 8, if the co-atom a used in the Dilemma Rule is one for which $a \sqsupseteq A$, then $A \sqcap \neg^\sharp a = \bot_{\mathcal{A}}$, and if $\neg^\sharp a \sqsupseteq$, then $A \sqcap a = \bot_{\mathcal{A}}$. For cases when the abstract

domain supports an exact meet (i.e., for all A_1, A_2, $\gamma(A_1 \sqcap A_2) = \gamma(A_1) \cap \gamma(A_2)$), the property $a \not\sqsupseteq A$ and $\neg^\sharp a \not\sqsupseteq A$ ensures that neither $A \sqcap a$ nor $A \sqcap \neg^\sharp A$ is $\bot_{\mathcal{A}}$, thus avoiding a redundant split.

Assuming that all of the required abstract-domain operations can be performed in polynomial time, Alg. 9 retains the property from the standard version of Stålmarck's method that it runs in polynomial time in the size of the input formula.