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 is dual to satisfiability: validity of φ can be determined using a SAT solver by checking the satisfiability of $\neg\varphi$ and complementing the answer: $\text{VALID}(\varphi) = \neg\text{SAT}(\neg\varphi)$.

With the advent of SAT-solvers based on conflict-directed clause learning (i.e., CDCL SAT solvers) [11] 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 [13]. Stålmarck's method was protected by Swedish, European, and U.S. patents [15], which may have discouraged experimentation by researchers. Indeed, one finds relatively few publications that concern Stålmarck's method—some of the exceptions are by Harrison [9], Cook and Gonthier [2], and Björk [1]. (Kunz and Pradhan [10] discuss a closely related method.)

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 [3]. In particular, we show that Stålmarck's method is based on a

* Supported, in part, by NSF under grants CCF-{0810053, 0904371}, by ONR under grants N00014-{09-1-0510, 10-M-0251, 11-C-0447}, by ARL under grant W911NF-09-1-0413, by AFRL under grants FA9550-09-1-0279 and FA8650-10-C-7088; and by DARPA under cooperative agreement HR0011-12-2-0012.. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors, and do not necessarily reflect the views of the sponsoring agencies.

** T. Reps has an ownership interest in GrammaTech, Inc., which has licensed elements of the technology discussed in this publication.

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 method will lead to

- better program-analysis tools that import principles found in Stålmarck's method into program analyzers
- improvements to Stålmarck-based validity checkers by (i) incorporating domains other than the ones that have been used (implicitly) in previous implementations of the method, or (ii) improving the method in other ways by incorporating additional techniques from the field of abstract interpretation.

There has been one payoff already: in [18], we describe ways in which ideas from Stålmarck's method can be adopted for use in program analysis. The techniques described in [18] are quite different from the huge amount of recent work based on reducing a program path π to a formula φ_π via symbolic execution, and then passing φ_π to a decision procedure to determine whether π is feasible. Instead, we adopted—and adapted—the key ideas from Stålmarck's method to create new algorithms for key program-analysis operations.

In this paper, we use the vantage point of abstract interpretation to describe the elements of the Dilemma Rule—the inference rule that distinguishes Stålmarck's method from other propositional-reasoning approaches—as follows:

Branch of a Proof: In Stålmarck's method, each proof-tree branch is associated with a so-called *formula relation* [13]. In abstract-interpretation terms, each branch is associated with an abstract-domain element.

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* [4] by means of a technique called *local decreasing iterations* [8].

"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 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.

We use these insights to present a parametric framework for propositional validity-checking algorithms. The advantages of our approach are

- We prove correctness at the framework level, once and for all, instead of for each instantiation.
- Instantiations that use 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 relations (k-BAR) and 2-variable Boolean inequality relations (2-BIR), we obtain more powerful decision procedures.

Contributions. The contributions of the paper can be summarized as follows:
- We explain Stålmarck's method in terms of abstract interpretation [3]—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.
- 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.

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 presents preliminary experimental results. §7 discusses related work. Proofs and a discussion of efficiency issues are presented in [17].

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.

2.1 Stålmarck's Method

Consider the tautology $\varphi = (a \wedge b) \vee (\neg a \vee \neg b)$. 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 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 in a set of propositional variables $\mathcal{V}$, and generates a list of *integrity constraints* as shown in Fig. 1. An *assignment* is a function in $\mathcal{V} \to \{0, 1\}$. The

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

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.

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. □

Alg. 1 (Fig. 4) 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 (Fig. 4) 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.

$v_1 \equiv \mathbf{0}$	... 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.

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.

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:

$$\begin{array}{lll} u_1 \Leftrightarrow (u_2 \Leftrightarrow u_3) & u_2 \Leftrightarrow (a \wedge u_4) & u_3 \Leftrightarrow (u_5 \vee u_6) \\ u_4 \Leftrightarrow (b \vee c) & u_5 \Leftrightarrow (a \wedge b) & u_6 \Leftrightarrow (a \wedge c) \end{array}$$

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

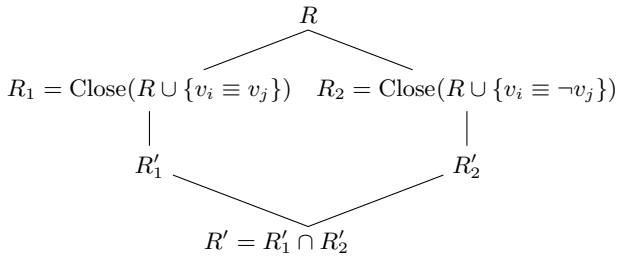


Fig. 5: The Dilemma Rule.

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 (Fig. 4). 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. 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 values of a and c, respectively, each deduce a contradiction on one of their branches. Each contradictory branch is eliminated because the (universal) relation $\mathbf{0} \equiv \mathbf{1}$ is the identity element for intersection, and hence the intersection result is the equivalence relation from the non-contradictory branch. We illustrate this fact in Fig. 6 by eliding the merges with contradictory branches. Finally, splitting on the variable b leads to a contradiction on both branches. □

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 (Fig. 4). Stålmarck's method (Alg. 5 of Fig. 4) 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 $O(|\varphi|^k)$ [13].

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 that might satisfy $\neg\varphi$. In particular, the contradictory equivalence relation $\mathbf{0} \equiv \mathbf{1}$ represents the empty set of assignments. Overall, the proof repre-

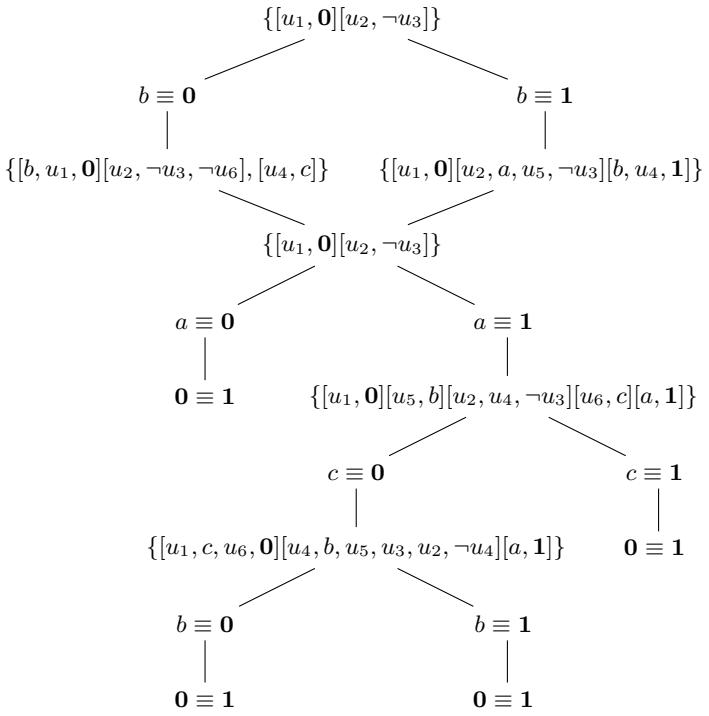


Fig. 6: Sequence of Dilemma Rules in a 1-saturation proof that ψ is valid. (Details of 0-saturation steps omitted.)

sents the set of assignments $\bigcup_i A_i$, which is a superset of the set of assignments that might satisfy $\neg\varphi$. Validity of φ 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 a strictly more expressive method than equivalence relations for abstracting a set of variable assignments. Moreover, Ex. 3 shows that, for some tautologies, replacing equivalence relations with inequality relations enables Stålmarck's method to be able to find a k-saturation proof with a strictly lower value of k.

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

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

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

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.

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 $w_1 \leq \mathbf{0}$. 0-saturation using the propagation rules of Fig. 7 results in the contradiction $\mathbf{1} \leq \mathbf{0}$, as shown in Fig. 8. □

We say that the instantiation of Stålmarck's method with inequality relations is *more powerful than* the instantiation with equivalence relations. In general, Stålmarck's method can be made more powerful by using a more expressive abstraction: when you plug in a more expressive 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 create more powerful instantiations?
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 $\mathbf{0}$ and $\mathbf{1}$, the unary connective $\neg$, and the binary con-

nectives $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, and $\oplus$ (xor). Propositional variables, negations of propositional variables, and propositional constants are referred to collectively as *literals*. $\mathrm{voc}(\varphi)$ denotes the subset of $\mathcal{V}$ that occurs in φ.

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

Definition 1. *An* **assignment** σ *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* σ *that is defined on (at least)* $x_1, \ldots, x_n$, *the* **meaning** *of* φ *with respect to* σ, *denoted by* $[\![\varphi]\!](\sigma)$, *is the truth value in* $\{0,1\}$ *defined inductively as follows:*

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

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

We overload the notation $[\![\cdot]\!]$ *as follows:* $[\![\varphi]\!]$ *means* $\{\sigma \mid \sigma : \mathcal{V} \to \{0,1\} \wedge \sigma \models \varphi\}$. *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 [3] ($\bot$, $\top$, $\sqcup$, $\sqcap$, $\sqsubseteq$, 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* $\mathcal{V}$, *their negations, and Boolean constants 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:*

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

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. □

Local Decreasing Iterations. Local decreasing iterations [8] 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. *([8, 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 instantiate the rules of Fig. 2 for 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 $\mathbf{0} \equiv \mathbf{1}$. □

Stålmarck's Method	Abstract-Interpretation Concept
Equivalence relation	Abstract-domain element
Propagation rule	Sound reductive operator
0-saturation	Local decreasing iterations
Split	Meet ($\sqcap$) in each proof-tree branch: one with a splitting-set element a and one with a's companion
Intersection ($\cap$)	Join ($\sqcup$)

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

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 proofs of all theorems stated in this section are found in [17].

Definition 4. *Given a Galois connection $\mathcal{C} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{A}$ between abstract domain $\mathcal{A}$ and concrete domain $\mathcal{C} = \mathbb{P}(\mathcal{V} \to \{0,1\})$, an **acceptable splitting set** S for $\mathcal{A}$ satisfies*

1. *$S \subseteq \mathcal{A}$*
2. *For every $a \in S$, there exists $b \in S$ such that $\gamma(a) \cup \gamma(b) = \gamma(\top)$. Two elements $a, b \in S$ such that $\gamma(a) \cup \gamma(b) = \gamma(\top)$ are called **companions**.*
3. *For every assignment $C \in \mathcal{V} \to \{0,1\}$ there exists $M_C \subseteq S$ such that $\gamma(\bigsqcap M_C) = C$. We call M_C the **cover** of C.* □

Example 5. The set of "single-point" partial assignments $\{\top[v \leftarrow 0]\} \cup \{\top[v \leftarrow 1]\}$ is an acceptable splitting set for both the Cartesian domain and the 2-BAR domain. Another acceptable splitting set for the 2-BAR domain is the set consisting of all 2-BAR elements that consist of a single constraint. □

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 $\gamma_{\text{Cartesian}}(A_c) = \gamma_{\mathcal{A}}(A)$.
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. There is an algorithm to perform the meet of arbitrary elements of the domain; that is, for all $A_1, A_2 \in \mathcal{A}$, there is an algorithm that produces $A_1 \sqcap A_2$.
5. There is an acceptable splitting set S for $\mathcal{A}$ (Defn. 4).

Assumption 2 ensures that any instantiation that satisfies assumptions 1–4 will satisfy assumption 5: the set of "single-point" partial assignments inherited from the Cartesian domain (Ex. 5) is always an acceptable splitting set.

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

```
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 a, b that are companions
       such that a ⋢ A and b ⋢ A do
4          A_1 ← A ⊓ a
5          A_2 ← A ⊓ b
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
```

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 algorithm derives that $\bar{A}' = \bot_{\mathcal{A}}$, then the formula φ is proved valid.

Generalized Propagation Rules. The propagation rules aim to refine the abstract state by assuming a single integrity constraint $J \in \mathcal{I}$. It is possible to list all the propagation rules in the style of Fig. 2 for the 2-BAR domain; for brevity, Alg. 6 is stated in terms of the semantic properties that an individual propagation rule satisfies, expressed using the operations α, γ, and $\sqcap$ of abstract domain $\mathcal{A}$. This procedure is sound if the abstract value $\bar{A}$ returned satisfies $\gamma(\bar{A}) \supseteq [\![\mathcal{I}]\!] \cap \gamma(A)$. Furthermore, to guarantee progress we have to show that Alg. 6 implements a reductive operator, i.e., $\bar{A} \sqsubseteq A$.

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

Example 6. 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 single integrity constraint (line 3), until no new information is derived or a contradiction is found (line 5); $\mathrm{voc}(\varphi)$ denotes the set of φ's propositional variables.

To improve efficiency the quantities J and A_1 are chosen so that $|\mathrm{voc}(J) \cup \mathrm{voc}(A_1)|$ is small (line 3). 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. Because J in Alg. 6 is a single integrity constraint, there are only a bounded number of Boolean operators involved in each propagation step. By limiting the size of $\mathrm{voc}(J) \cup \mathrm{voc}(A_1)$ (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.

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

Generalized k-Saturation. Alg. 8 describes the generalized k-saturation procedure that repeatedly applies the generalized Dilemma Rule. By requirement 5, there is an acceptable splitting set S for $\mathcal{A}$. The generalized Dilemma Rule, shown schematically in Fig. 9, splits the current abstract state A into two abstract states A_1 and A_2 using companions $a, b \in S$. Using the fact that $\gamma(a) \cup \gamma(b) = \gamma(\top)$ (Defn. 4), we can show that $\gamma(A_1) \cup \gamma(A_2) = \gamma(A)$. This fact is essential for proving the soundness of the generalized Dilemma Rule. To merge the two branches of the generalized Dilemma Rule, we perform a join of the abstract states derived in each branch. The dashed arrows from A to A', A_1 to A'_1, and A_2 to A'_2 in Fig. 9 indicate that, in each case, the target value is a semantic reduction of the source value. The next theorem proves that Alg. 8, which utilizes the generalized Dilemma Rule, is sound.

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

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

```
1 (v_φ, I) ← integrity(φ)
2 A ← ⊤_A[v_φ ← 0]
3 A' ← k-saturation_A(A, I)
4 if A' = ⊥_A then return valid
5 else return unknown
```

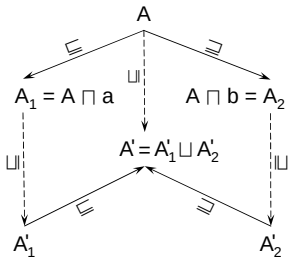


Fig. 9: Generalized Dilemma Rule.

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. [Soundness of Alg. 9]
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. To prove completeness we make use of item 3 of Defn. 4. After performing k-saturation, Alg. 9 has considered all assignments C that have a cover of size k. Let MinCover$[C] = \texttt{min}\{|M| \mid M \subseteq S$ is a cover of $C\}$, and let $m = \texttt{max}_{C \in \mathcal{C}}$MinCover$[C]$. m-Stålmarck$_{\mathcal{A}}(\varphi)$ will consider all assignments, and thus is complete; that is, if m-Stålmarck$_{\mathcal{A}}(\varphi)$ returns *unknown*, then φ is definitely not valid. The efficiency of our generalization of Stålmarck's Method is discussed in [17].

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. As observed in §4, any instantiation that satisfies the first four assumptions of the framework has an acceptable splitting set; hence, we only consider the first four assumptions in the discussion below.

Cartesian Domain. The original version of Stålmarck's method [16] 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.

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 $a_i, b \in \{\mathbf{0}, \mathbf{1}\}$.

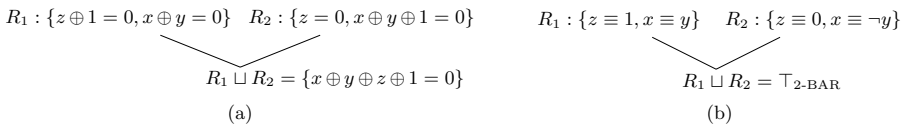


Fig. 10: 3-BAR (a) retains more information at the join than 2-BAR (b).

1. The definitions of the γ and α functions of the Galois connection $\mathbb{P}(\mathcal{V} \to \{0,1\}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}}$ 3-BAR are identical to those stated in Defn. 2.
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. $A_1 \sqcap A_2$ can be implemented by unioning the two sets of constraints.

Example 7. Fig. 10 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. □

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. The definitions of γ and α are again identical to those given in Defn. 2.
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. $A_1 \sqcap A_2$ can be implemented by unioning the two sets of constraints.

Example 8. Fig. 11 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. □

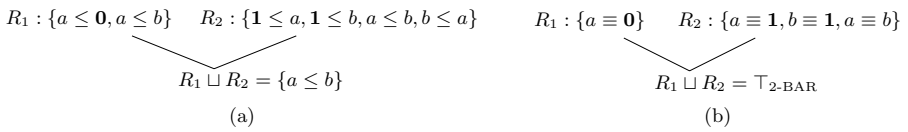


Fig. 11: 2-BIR (a) retains more information at the join than 2-BAR (b).

6 Experiments

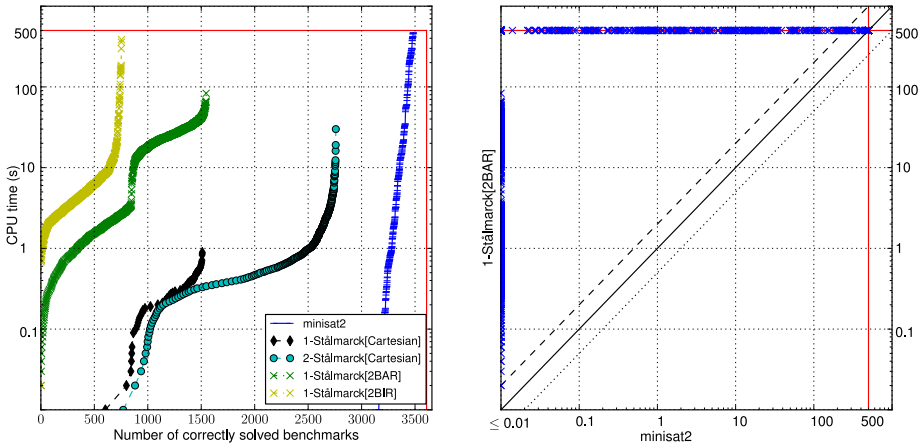
As discussed in §1, a validity-checking algorithm can be used for checking satisfiability. In this section, we present preliminary experimental results for the following instantiations of our parametric framework:

- *1*-Stålmarck[Cartesian]: uses 1-saturation and the Cartesian domain.
- *1*-Stålmarck[2-BAR]: uses 1-saturation and the 2-BAR domain.
- *1*-Stålmarck[2-BIR]: uses 1-saturation and the 2-BIR domain.
- *2*-Stålmarck[Cartesian]: uses 2-saturation and the Cartesian domain.

We compared the above algorithms with the mature SAT solver, MiniSat (v2.2.0) solver [7]. For our evaluation, we used the Small, Difficult Satisfiability Benchmark (SDSB) suite, which contains 3,608 satisfiability benchmarks that have up to 800 literals, and have been found to be difficult for solvers [14]. We used a time-out limit of 500 seconds. If an algorithm could not determine whether a benchmark was satisfiable or unsatisfiable, then the solver is recorded as taking the full 500 seconds for that benchmark.

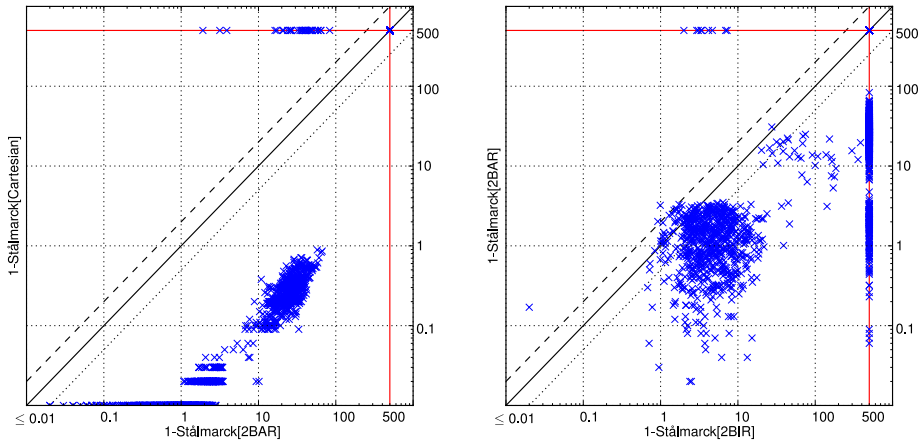
For each of the five algorithms, Fig. 12(a) is a semi-log plot in which each point (n, t) means that there were n benchmarks that were each solved correctly in no more than t seconds. Fig. 12(b) and Fig. 13 give log-log scatter plots of the time taken (in seconds) for each of the benchmarks, for several combinations of the five algorithms. As seen in Fig. 12(a), MiniSat correctly solves 3,484 of 3,608 benchmarks, and is significantly faster than *1*-Stålmarck[2-BAR] (Fig. 12(b)).

When comparing among the instantiations of our framework, we expect more benchmarks to be solved correctly as we move to more expressive abstract domains. For instance, *1*-Stålmarck[2-BAR] (1,545 benchmarks) solves 36 bench-



(a) (b)

Fig. 12: (a) Semi-log plot showing the number of benchmarks that were each solved correctly in no more than t seconds. (b) Log-log scatter plot of the time taken (in seconds) by MiniSat versus *1*-Stålmarck[2-BAR].



(a) (b)

Fig. 13: Log-log scatter plots of the time taken (in seconds) by (a) *1*-Stålmarck[2-BAR] versus *1*-Stålmarck[Cartesian], and (b) *1*-Stålmarck[2-BIR] versus *1*-Stålmarck[2-BAR].

marks that *1*-Stålmarck[Cartesian] (1,509 benchmarks) was unable to solve. On the other hand, *1*-Stålmarck[2-BAR] is slower than *1*-Stålmarck[Cartesian], as seen in Fig. 13(a), because the join operation of the 2-BAR domain is more expensive than that of the Cartesian domain. The complexity of the join operation plays an even greater role for the 2-BIR domain: although *1*-Stålmarck[2-BIR] solves 9 benchmarks that *1*-Stålmarck[2-BAR] was unable to solve, overall *1*-Stålmarck[2-BIR] is only able to solve 754 benchmarks in the 500-second time limit. We are currently investigating more efficient implementations of the join algorithms for the various domains.

Using 2-saturation allows Stålmarck's method instantiated with Cartesian domain to correctly solve 2,758 benchmarks (Fig. 12(a)), including 1,213 benchmarks that *1*-Stålmarck[2-BAR] was unable to solve and 1,774 benchmarks that *1*-Stålmarck[2-BIR] was unable to solve.

7 Related Work

Stålmarck's method was patented under Swedish, U.S., and European patents [15]. Sheeran and Stålmarck [13] give a lucid presentation of the algorithm. Björk [1] explored extensions of Stålmarck's method to first-order logic.

CDCL/DPLL solvers [12] are alternatives to Stålmarck's method for validity checking and SAT. D'Silva et al. [5] give an abstract-interpretation-based account of CDCL/DPLL SAT solvers. Thus, though having similar goals, our work and that of D'Silva et al. are complementary. Our work and that of D'Silva et al. were performed independently and contemporaneously. They have also lifted their

technique from a propositional SAT solver to a floating-point decision procedure that makes use of floating-point intervals [6].

References

1. M. Björk. First order Stålmarck. *J. Autom. Reasoning*, 42(1):99–122, 2009.
2. B. Cook and G. Gonthier. Using Stålmarck's algorithm to prove inequalities. In *ICFEM*, 2005.
3. 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.
4. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *POPL*, 1979.
5. V. D'Silva, L. Haller, and D. Kroening. Satisfiability solvers are static analyzers. In *SAS*, 2012.
6. V. D'Silva, L. Haller, D. Kroening, and M. Tautschnig. Numeric bounds analysis with conflict-driven learning. In *TACAS*, 2012.
7. N. Eén and N. Sjørensson. The MiniSat solver, 2006. minisat.se/MiniSat.html.
8. P. Granger. Improving the results of static analyses programs by local decreasing iteration. In *FSTTCS*, 1992.
9. J. Harrison. Stålmarck's algorithm as a HOL Derived Rule. In *TPHOLs*, 1996.
10. W. Kunz and D. Pradhan. Recursive learning: A new implication technique for efficient solutions to CAD problems–test, verification, and optimization. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 13(9):1143–1158, 1994.
11. J. Marques Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *ICCAD*, 1996.
12. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, 2001.
13. M. Sheeran and G. Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. *FMSD*, 16(1):23–58, 2000.
14. I. Spence. tts: A SAT-solver for small, difficult instances. *Journal on Sat., Boolean Modeling and Computation*, 2008. Benchmarks available from http://www.cs.qub.ac.uk/ i.spence/sdsb.
15. 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).
16. G. Stålmarck and M. Säflund. Modeling and verifying systems and software in propositional logic. In *Int. Conf. on Safety of Computer Controls Systems*, 1990.
17. A. Thakur and T. Reps. A generalization of Stålmarck's method. TR 1699 (revised), CS Dept., Univ. of Wisconsin, Madison, WI, June 2012. See "www.cs.wisc.edu/wpis/papers/tr1699r.pdf".
18. A. Thakur and T. Reps. A method for symbolic computation of precise abstract operations. In *CAV*, 2012.