Science of Computer Programming 35 (1999) 191–221

Science of Computer Programming

www.elsevier.nl/locate/scico

Tractable constraints in finite semilattices

Jakob Rehof[a,*,1], Torben Æ. Mogensen[b]

[a] *Microsoft Research, One microsoft Way, Redmond, WA 98052, USA*

[b] *DIKU, Department of Computer Science, Universitetsparken 1, DK-2100 Copenhagen Ø, Denmark*

Abstract

We introduce the notion of *definite* inequality constraints involving monotone functions in a finite meet-semilattice, generalizing the logical notion of Horn-clauses, and we give a linear time algorithm for deciding satisfiability. We characterize the expressiveness of the framework of definite constraints and show that the algorithm uniformly solves exactly the set of all meet-closed relational constraint problems, running with small linear time constant factors for any fixed problem. We give an alternative technique for reducing inequalities to satisfiability of Horn-clauses (HORNSAT) and study its efficiency. Finally, we show that the algorithm is complete for a maximal class of tractable constraints, by proving that *any* strict extension will lead to **NP**-hard problems in any meet-semilattice. © 1999 Elsevier Science B.V. All rights reserved.

Keywords: Finite semilattices; Constraint satisfiability; Program analysis; Tractability; Algorithms

1. Introduction

It is well known that many program analysis problems can be solved by generating a set of constraints over a finite domain and then solving these. Examples include flow analysis (closure analysis) [24], binding time analysis [15, 4, 5, 11], usage count analysis [23], multiplicity inference for region-size analysis [3], strictness analysis [16, 1]. In many cases such analyses can be specified by annotated-type systems [27], and the analyses can be implemented using non-standard type-inference, where constraints between type annotations are collected from the program. Some more detailed examples taken from flow analysis and usage count analysis are given in Section 6.

In this paper we show how to solve certain classes of constraints over finite domains efficiently, and characterize classes that are not tractable. The solution methods can be used as a tool for analysis designers, and the characterization can help the designer recognize when an analysis may have bad worst-case behaviour.

* Corresponding author. Fax: (+1) 425-705-2404.

E-mail addresses: rehof@microsoft.com (J. Rehof), torbenm@diku.dk (T.Æ. Mogensen)

[1] This work was done while the first author was at DIKU.

Our classification of constraint problems evolves from the notion of *definite inequalities*, which can be seen as a generalization of the logical notion of *Horn clauses* [14, 10], from the two-point boolean lattice of the truth values to arbitrary finite semilattices. Definite inequality constraints are inequalities of the form $\tau \leqslant A$ where τ is an expression built from an arbitrary set of function symbols denoting monotone functions and A is either a variable or a constant. We show that the fixpoint computation technique of Kildall [22] and the decision procedure for Horn-clause satisfiability of Dowling and Gallier [10] can be adapted to the framework of general finite semilattices, leading to an algorithm, called D, for solving sets of definite constraints, with the following properties:

- The algorithm runs in linear time with small constant factors, for any fixed finite semilattice.
- The algorithm operates uniformly over all finite semilattices and all monotone functions, and hence it can be thought of as a general solver for the *uniform problem* which is parameterized on both the domain (semilattice), the monotone functions and the constraint set. The algorithm can therefore serve as a general purpose, off-the-shelf solver, which supports a whole range of program analyses.

We are interested in understanding the *range of applicability* of the framework of definite constraints, i.e., we would like to know, what problems can algorithm D solve efficiently? At a superficial level, the applicability of the algorithm is restricted to the class of definite constraints, since the algorithm assumes its constraint set parameter to be of this form. However, constraint problems which are not given in definite form might be transformed into equivalent definite ones, and hence the question above translates into the problem of determining the expressive power of definite inequality constraints: which problems can be expressed by equivalent sets of definite constraints? In order to demonstrate that generality does not come at the cost of sacrificing efficiency, our investigation of this latter question is accompanied by an analysis of the *cost* of the transformations into definite form. We show that

- Algorithm D solves exactly the set of all *meet-closed* relational constraint problems, modulo linear time transformations, with small constant factors, into definite form. Hence, the generic use of the algorithm is feasible, remaining within the efficient linear time framework.

Here, the notion of a relational constraint problem is a generalization of the logical notion introduced by Schaefer [26] to prove his remarkable Dichotomy Theorem, according to which any relational boolean constraint problem either falls into one of four easily recognizable tractable (polynomial time solvable) classes, or else the problem is **NP**-complete. One of the four tractable classes corresponds exactly to Horn-clauses. For recent generalizations of Schaefer's Dichotomy Theorem in various directions, see [12, 7, 21].

By composing Birkhoff's Representation Theorem for finite lattices with Schaefer's Dichotomy Theorem we get an easy proof of the following completeness property for algorithm D:

– Any strict extension of the class of meet-closed problems solved by algorithm D will lead to an **NP**-complete uniform problem.
– For distributive lattices we can exhibit simple meet-closed problems with a fixed semilattice and a fixed set of relations such that no matter how the problems are extended with non-meet-closed relations the resulting problems will be **NP**-complete.

This shows that algorithm D is complete for a maximal tractable class of problems, namely the meet-closed ones, and the theorems on non-extendability can be used to recognize the borderline between tractable and intractable problems.

Finally, we show that definite constraint problems can be represented by boolean Horn-implications, leading to an alternative linear time solution procedure, by reduction to the procedure of Dowling and Gallier [10]. We compare algorithm D to reduction to Horn-clauses with respect to time and space efficiency.

The paper is organized as follows. Section 2 introduces the framework of constraint problems involving monotone functions on finite semilattices, and Section 3 defines the restricted form of definite inequalities and introduces algorithm D, which appears in Appendix A. Section 4 introduces the generalized framework of relational problems; the expressiveness of definite constraints is studied (Section 4.1) and the alternative method of transformation into logical Horn-clauses is described (Section 4.2). The results on intractability of extensions of the meet-closed framework are given in Section 5. Section 6 contains some more detailed examples of how the results presented in this paper can be used to reason about concrete program analysis problems. Finally, Section 7 compares the behaviour of an implementation of algorithm D and an implementation of constraint solving by reduction to Horn-clauses. Section 8 concludes the paper.

2. Monotone function problems

Let P be a poset and F a finite set of monotone functions $f : P^{a_f} \rightarrow P$ with $a_f \geqslant 1$ the *arity* of f. We call the pair $\Phi = (P, F)$ a *monotone function problem* (MFP). Given $\Phi = (P, F)$ we let T_Φ denote the set of *Φ-terms*, ranged over by τ, σ and given by $\tau ::= \alpha \mid c \mid f(\tau_1, \ldots, \tau_{a_f})$ where c ranges over constants in P, α, β, γ range over a denumerably infinite set $\mathscr{V}$ of variables and f is a function symbol corresponding to $f \in F$. Constants and variables are collectively referred to as *atoms*, and we let A range over atoms. We assume a fixed enumeration of $\mathscr{V}, \mathscr{V} = v_1, v_2, \ldots, v_n, \ldots$ For each number $m > 0$ we let $\mathscr{V}_m$ denote the sequence of the m first variables in the enumeration of $\mathscr{V}$. Let $\rho \in S^m$ for some set S. We write $[\rho]_i$ for the ith coordinate of ρ, $i \in \{1, \ldots, m\}$. Any $\rho \in S^m$ is implicitly considered a mapping $\rho : \mathscr{V}_m \rightarrow S$ by defining $\rho(v_i)$ to be $[\rho]_i$, for $i \in \{1, \ldots, m\}$. For $\rho \in S^m, 1 \leqslant k \leqslant m$ we let $\rho \downarrow k = ([\rho]_1, \ldots, [\rho]_k) \in S^k$.

A *constraint set* C over Φ is a finite set of formal inequalitites of the form $\tau \leqslant \tau'$ with $\tau, \tau' \in T_\Phi$. The set of distinct variables occurring in a term τ is denoted $Var(\tau)$, and if C is a constraint set, then $Var(C)$ denotes the set of distinct variables occurring in C. In this paper, *we always assume that* $Var(C)=\mathcal{V}_m$ *for some* m. If $\rho \in P^m$, τ a (P,F)-term with $Var(\tau)\subseteq \mathcal{V}_m$, then ρ is called a *valuation* of τ in P. A valuation is tacitly lifted to an *interpretation* $[\![\tau]\!]\rho$ of τ, given by $[\![\alpha]\!]\rho=\rho(\alpha)$, $[\![c]\!]\rho=c$, $[\![f(\tau_1,\ldots,\tau_k)]\!]\rho=f([\![\tau_1]\!]\rho,\ldots,[\![\tau_k]\!]\rho)$. Let $Var(\tau)\subseteq\mathcal{V}_m$. Then the *m-ary function denoted by* τ is the map $[\![\tau]\!]:P^m\to P$ given by $[\![\tau]\!](\rho)=[\![\tau]\!]\rho$. Since every function in F is assumed to be monotone, the function $[\![\tau]\!]$ is easily seen to be monotone, for every (P,F)-term τ.

Let τ,τ' be (P,F)-terms. If $Var(\tau)\cup Var(\tau')\subseteq\mathcal{V}_m$, $\rho\in P^m$, we say that ρ *satisfies* the constraint $\tau\leqslant\tau'$, written $P,\rho\models\tau\leqslant\tau'$, iff $[\![\tau]\!]\rho\leqslant[\![\tau']\!]\rho$ is true in P. If C is a constraint set over (P,F), $Var(C)\subseteq\mathcal{V}_m$, $\rho\in P^m$, we say that ρ is a *valuation of* C *in* P; we say that ρ satisfies C, written $P,\rho\models C$, iff $P,\rho\models\tau\leqslant\tau'$ for every constraint $\tau\leqslant\tau'$ in C. We say that C is *satisfiable* if and only if there exists a valuation ρ of C in P such that $P,\rho\models C$. The set of *solutions* to C, denoted $Sol(C)$, is the set $\{\rho\in P^m \mid P,\rho\models C, m=|Var(C)|\}$ ($|Var(C)|$ denotes the size of $Var(C)$).

If $\Phi=(P,F)$ is an MFP, then we define the decision problem Φ-SAT to be the following: *Given a constraint set* C *over* Φ, *determine whether* C *is satisfiable over* P. We assume that $f\in F$ are given by a_f-dimensional *operation matrixes* M_f with $M_f[x_1,\ldots,x_{a_f}]=f(x_1,\ldots,x_{a_f})$. Under this representation, evaluating a function f at given arguments $x_1,\ldots,x_{a_f}$ is a constant time operation, and hence evaluating an arbitrary functional term τ is $O(|\tau|)$ (we measure the size of a term by $|\tau|$, the number of occurrences of symbols (constants, variables and function symbols) in τ; the size of a constraint set C, denoted $|C|$, is the number of occurrences of symbols in C). If P is a lattice, the component P is assumed to be given as the set of elements of P together with an additional operation matrix, $M_\sqcup$, defining the least upper bound of L, i.e., $M_\sqcup[x,y]=x\sqcup y$. From this we can recover the order relation of P, using that $x\leqslant y$ iff $x\sqcup y=y$. This representation will be referred to as the *matrix representation*. As we shall see, this is also an appropriate representation when P is a semilattice, since this case will be reduced to the case where P is a lattice.

There are many problems Φ for which Φ-SAT is **NP**-hard (every problem Φ-SAT is obviously in **NP**, since we can guess and verify a solution non-deterministically in polynomial time). Simple examples of this occur whenever P is a finite lattice and both the least upper bound ($\sqcup$) and the greatest lower bound ($\sqcap$) operations are in the F-component of Φ:

Example 1. For any non-trivial finite lattice L, the problem $\Phi=(L,\{\sqcap,\sqcup\})$ is **NP**-complete, by reduction from CNF-SAT (propositional satisfiability, [13]): We can assume that L contains at least one *atom*, i.e., an element $c\in L$ such that $c\neq\bot$ and for every $x\in L$, $\bot\leqslant x<c$ implies $x=\bot$. Any logical *clause* C (a set of literals representing their disjunction) can be written in the form

$$(P_1\wedge\cdots\wedge P_k)\Rightarrow(Q_1\vee\cdots\vee Q_m)$$

where the P_i are the propositional variables which occur negated in the clause, and the Q_i are the variables which occur unnegated; here the empty disjunction is the logical constant *false*, and the empty conjunction is *true*. Then C is logically satisfiable if and only if

$$(P_1 \sqcap \cdots \sqcap P_k) \leqslant (Q_1 \sqcup \cdots \sqcup Q_m)$$

is satisfiable under the additional constraints that $\alpha \leqslant c$ for every variable α in C, *true* is interpreted as c and *false* as $\perp$, where c is any fixed atom of L.

Also, the structure of the poset P is important for complexity, see [25].

The foregoing observations show that we need to impose restrictions on problems to make them tractable. In the following development, we shall generally assume that P is a meet-semilattice. Note, though, that the whole development transfers to join-semilattices by lattice-theoretic dualization.

3. Definite problems

Let $\Phi = (P, F)$ be an MFP. A constraint set C over Φ in which every inequality is of the form $\tau \leqslant A$, with an atom on the right hand side, is called *definite*. A definite set $C = \{\tau_i \leqslant A_i\}_{i \in I}$ can be written $C = C_{\mathrm{var}} \cup C_{\mathrm{cnst}}$ where $C_{\mathrm{var}} = \{\tau_j \leqslant \beta_j\}_{j \in J}$ are the *variable expressions* in C, having a variable (β_j) on the right-hand side of $\leqslant$, and $C_{\mathrm{cnst}} = \{\tau_k \leqslant c_k\}_{k \in K}$ are the *constant expressions* in C, having a constant (c_k) on the right-hand side. Notice that propositional Horn-clauses are a special case of definite inequalities:

Example 2. Satisfiability of *definite* inequalities over $\Phi = (\mathbf{2}, \{\sqcap\})$, with $\mathbf{2}$ the two-point boolean lattice, is exactly the HORNSAT problem (satisfiability of propositional Horn clauses [14, 10]) since Horn clauses have the form $P_1 \wedge \cdots \wedge P_n \Rightarrow Q$, which is equivalent to the inequality $P_1 \sqcap \cdots \sqcap P_n \leqslant Q$.

A term τ will be called *simple* if τ is a constant, a variable or has the form $\tau = f(A_1, \ldots, A_m)$, i.e., there are no nested function applications; a constraint set C is called simple if all terms in it are simple. The L-normalization $\rightarrow_L$ transforms a definite set into a simple and definite set: let C be definite, $C = C' \cup \{f(\ldots g(\boldsymbol{\tau}) \ldots) \leqslant A\}$ with $Var(C) = \mathscr{V}_m$ and $\boldsymbol{\tau}$ a tuple of terms, and define $\rightarrow_L$ by

$$C \rightarrow_L C' \cup \{f(\ldots v_{m+1} \ldots) \leqslant A, g(\boldsymbol{\tau}) \leqslant v_{m+1}\}$$

Normalizing a set with respect to $\rightarrow_L$ expands the set with only a small constant factor:

Lemma 1. *The reduction $\rightarrow_L$ is strongly normalizing, and if C^* is a normal form of a definite set C, then C^* is definite with $|C^*| \leqslant 3|C|$.*

Proof. Given definite set C, let $\|C\|$ denote the number of distinct occurrences of function symbols that are nested under a function symbol in C; it is easy to see that if

$C \to_L C'$ then $\|C'\| = \|C\| - 1$, and hence no L-reduction sequence starting from C can have more than $\|C\|$ steps. Also, if $C \to_L C'$, then $|C'| = |C| + 2$. Moreover, we have $\|C\| \leqslant |C|$, since $|C|$ counts the total number of occurrences of symbols in C, whereas $\|C\|$ counts only a subset. It follows that, if C^* is a normal form of C, we must have

$$|C^*| \leqslant 2\|C\| + |C| \leqslant 2|C| + |C| = 3|C|$$

The normal form C^* is obviously definite, since C is.

Monotonicity guarantees that solving an L-normalized set is equivalent to solving the original set:

Lemma 2. *If $C \to_L C'$ then $Sol(C) = \{\rho' \downarrow m \mid \rho' \in Sol(C')\}$ where $Var(C) = \mathscr{V}_m$.*

Proof. Let $C = C'' \cup \{f(\dots g(\tau)\dots) \leqslant A\}$, $Var(C) = \mathscr{V}_m$ with

$$C \to_L C'' \cup \{f(\dots v_{m+1}\dots) \leqslant A, g(\tau) \leqslant v_{m+1}\} = C'$$

Suppose ρ is a solution to C. Define ρ' by $\rho'(v_i) = \rho(v_i)$ for $i \leqslant m$ and $\rho'(v_{m+1}) = [\![g(\tau)]\!]\rho$. Then

$$f(\dots \rho'(v_{m+1})\dots) = f(\dots [\![g(\tau)]\!]\rho \dots) \leqslant [\![A]\!]\rho = [\![A]\!]\rho'$$

(we used $[\![A]\!]\rho = [\![A]\!]\rho'$, since v_{m+1} does not occur in A) and so ρ' solves the first inequality of the reduct. For the second, we have

$$[\![g(\tau)]\!]\rho' = [\![g(\tau)]\!]\rho = \rho(v_{m+1})$$

where we used $v_{m+1} \notin Var(g(\tau))$ at the first equation. So ρ' solves the second inequality also. We have shown the left-to-right inclusion.

Assume, for the other inclusion, that $\rho' \in Sol(C')$. Define $\rho = \rho' \downarrow m$. Then (since ρ' solves C')

$$[\![g(\tau)]\!]\rho' \leqslant \rho'(v_{m+1})$$

and therefore, by monotonicity, we have

$$f(\dots [\![g(\tau)]\!]\rho \dots) = f(\dots [\![g(\tau)]\!]\rho' \dots) \leqslant f(\dots \rho'(v_{m+1})\dots) \leqslant [\![A]\!]\rho' = [\![A]\!]\rho$$

where we used again that v_{m+1} occurs neither in A nor in $g(\tau)$. This shows that $\rho = \rho' \downarrow m$ is a solution to C.

Fig. 4 in Appendix A gives an algorithm, called D, for solving definite constraints over an MFP $\Phi = (L, F)$ with L a finite lattice.[2] Algorithm D exploits L-normalization

[2] Algorithm D is similar to the technique of Kildall [22] for fast fixed point computation in data-flow frameworks and also to the linear time algorithm of Dowling and Gallier [10] for solving the HORNSAT-problem. Compare also the functional dependency closure algorithm of Beeri and Bernstein [2]. In fact, the iteration step of algorithm D is easily seen to be equivalent to a search for the least fixed point of a monotone operator F on a lattice, since the least fixed point of F is identical to the least post-fixed point of F. We already observed (beginning of Section 3) that definite inequalities strictly subsume Horn clauses. See Sections 4.1 and 4.2 for more on the connection to Horn clauses.

to achieve linear time worst case complexity. Later, we shall change D slightly into an algorithm $D^{\top}$ which works for meet-semilattices.

Correctness of algorithm D follows from the properties:

(1) after every update of the current valuation ρ at β, $\rho(\beta)$ increases strictly in the order of L, so in particular, termination follows, since L has finite height;

(2) the iteration step finds the least solution μ to the set of *variable expressions* in C, so if any solution to C exists, then μ must also satisfy the *constant expressions*, by monotonicity of all functions $[\![\tau]\!]$.

Since the correctness proof is a straightforward consequence of the properties mentioned above, we leave out the proof of the following theorem.

Theorem 3 (Correctness). *If C has a solution over Φ, then it has a minimal solution, and algorithm D outputs ρ the minimal solution of C, and if C has no solution, then the algorithm fails.*

We analyse the complexity of algorithm D, showing that it runs in linear time with a small constant factor for fixed problem Φ. Let $h(L)$ denote the height of a finite lattice L, i.e., the maximal length of a chain in L.

Theorem 4 (Complexity). *For fixed MFP Φ algorithm D runs in time* $O(|C|)$ *and performs at most $3h(L) \cdot |C|$ basic computations on input C.*

Proof. *Initialization*: We can assume (Lemma 1) that C is in L-normal form modulo an expansion by a factor 3 (hence $|C|$ below is at worst three times the size of the input set). It is easy to see that initialization can be done in time of the order $O(|C|)$. We use that arrays are assumed to be indexed by variables, and every test of the form $\rho, L \models \tau \leqslant \beta$ (or the negation of such) can be performed in constant time, by L-normal form. A detailed argument is straightforward and is left out.

Iteration: We amortize the number of times the test of the conditional in the **for**-loop, $\rho, L \not\models \sigma \leqslant \gamma$, can be executed *in total* on input C, in the worst case. We observe that each time an item $Clist[\beta][k]$ is inspected in this test the current valuation ρ has increased strictly (w.r.t. the order on L) at β. It follows that no single item $Clist[\beta][k]$ can be inspected more than $h(L)$ times, since ρ cannot increase more than $h(L)$ times at β. Now, the number of entries in $Clist[\beta]$ is equal to the number of occurrences of variable β in C (cf. description of $Clist$, Appendix A). Since the number of items in total in $Clist$ is therefore bounded by $|C|$, it follows that no more than $h(L) \cdot |C|$ single inspections can be performed over a complete run of the algorithm on input C. The bound on the number of tests implies that all the other instructions in the iteration step can at most be executed $h(L) \cdot |C|$ times over a complete run of the algorithm, on input C. By L-normal form, all instructions are constant time (depending only on the arity of function symbols in Φ, which is held fixed). Since the final test at the output step is clearly computable in time $O(|C|)$, we have shown that, at any control point of the algorithm, the instruction at that point can consume at most $h(L) \cdot |C|$ units of time in total on input C.

Algorithm D operates uniformly in Φ, so it can be considered a decision procedure for the *uniform problem* parametric in Φ and C. In this case, taking $h(L) = |L|$ in the worst case, $|L|^{a_{\max}}$ the size of the function matrixes with $a_{\max}$ the maximal arity of functions in Φ, we have input size $N = |L| + |L|^{a_{\max}} + |C|$. With log-cost for a matrix look-up we get a maximum cost of $\mathrm{O}(a_{\max} \cdot \log |L|)$ for a basic computation, resulting in $\mathrm{O}(a_{\max} \cdot \log |L| \cdot |L| \cdot N)$ worst case behaviour for the uniform problem.

Algorithm D generalizes to the cases where the poset is a finite meet-semilattice, as follows. Let $P^{\top}$ denote the lattice obtained from P by adding a top element $\top$, taking $c \leqslant \top$ for all $c \in P$. We change algorithm D by adding the test **if** $\exists \alpha . \rho(\alpha) = \top$ **then** FAIL at the beginning of the output step. We extend the functions in Φ such that $f(x_1, \ldots, x_{a_f}) = \top$ if any $x_i = \top$. This modification of algorithm D will be referred to as algorithm $D^{\top}$. Algorithm $D^{\top}$ is obviously sound, by soundness of algorithm D, and it is a complete decision procedure for semilattices, since if P has no top element and the least solution to the variable expressions in C maps a variable to the top element of $P^{\top}$, then clearly C can have no solutions in P.

4. Relational problems

Inequality constraints are a special case of the much more general framework of relational constraints. A *relational constraint problem* is a pair $\Gamma = (P, \mathscr{S})$ with P a finite poset and $\mathscr{S}$ a finite set of finite relations $R \subseteq P^{a_R}$, with $1 \leqslant a_R$ (the arity of R). Any $R \subseteq P^k$ is called a *relation over* P. A *relational constraint set* C over Γ is a finite set of Γ*-terms* of the form $R(A_1, \ldots, A_{a_R})$ where A ranges over variables in $\mathscr{V}$ and constants drawn from P. The size of a constraint term $t = R(A_1, \ldots, A_{a_R})$ is $|t| = a_R$; the size $|C|$ of a constraint set is the sum of the sizes of all terms in C. We say that a constraint set C is *satisfiable* if there exists a valuation ρ of C in P such that $([\![A_1]\!]\rho, \ldots, [\![A_{a_R}]\!]\rho) \in R$ for every term $R(A_1, \ldots, A_{a_R})$ in C. If $Var(C) = \mathscr{V}_m$, then $Sol(C)$ is the set of valuations $\rho \in P^m$ satisfying C. We define the decision problem Γ-SAT to be: *Given a constraint set* C *over* Γ, *is* C *satisfiable?* If $x \in P^m$ and $1 \leqslant i \leqslant m$ then $[x]_i$ denotes the ith coordinate of x. We use the vector notation $\mathbf{x} = (x_1, \ldots, x_m)$, and if $R \subseteq P^m$ we write $R(\mathbf{x})$ as an abbreviation for $R(x_1, \ldots, x_m)$ also when this expression is considered a term.

4.1. Representability

We are interested in the following question: How many relational constraint problems can be (efficiently) solved using algorithm D? This translates to the question: How many problems can be transformed into definite inequality problems and what is the cost of the transformation?

A relation $R \subseteq P^m$ is called *representable* with respect to a constraint problem $\Gamma = (P, \mathscr{S})$ if $R = Sol(C)$ for some constraint set C over Γ. We say that a problem Γ is a *problem with inequality* if the order relation $\leqslant$ on P is representable with respect to Γ. We say that a problem Γ *has minimal solutions* if $Sol(C)$ has a minimal element with respect to $\leqslant$ for any constraint set C over Γ with $Sol(C) \neq \emptyset$. If $\Gamma = (P, \mathscr{S})$ with P a meet-semilattice and it holds for all $R \in \mathscr{S}$ that $x, y \in R$ implies $x \sqcap y \in R$, then Γ is said to be a *meet-closed problem.* A relational problem $\Gamma = (P, \mathscr{S})$ is *representable in definite form* if P is a meet-semilattice and there exists an MFP Φ such that for all constraint sets C over Γ there exists a definite set C' over Φ with $Sol(C) = Sol(C')$.

Suppose that $R \subseteq P^m$ is a meet-closed relation, P a meet-semilattice. Then define the *partial* function $H_R : P^m \to P^m$ by $H_R(\mathbf{x}) = \bigwedge \uparrow_R (\mathbf{x})$ where $\uparrow_R (\mathbf{x}) = \{\mathbf{y} \in P^m \mid \mathbf{y} \geqslant \mathbf{x}, R(\mathbf{y})\}$ and with H_R *undefined* if $\uparrow_R (\mathbf{x}) = \emptyset$. Then H_R is monotone when defined, i.e., $\forall \mathbf{x}\mathbf{y} \in dom(H_R).\ \mathbf{x} \leqslant \mathbf{y} \Rightarrow H_R(\mathbf{x}) \leqslant H_R(\mathbf{y})$, and if $\uparrow_R (\mathbf{x}) \neq \emptyset$, then $\mathbf{x} \in dom(H_R)$. Moreover, for all $\mathbf{x} \in dom(H_R)$ one has $H_R(\mathbf{x}) \geqslant \mathbf{x}$, since every $\mathbf{y} \in \uparrow_R(\mathbf{x})$ satisfies $\mathbf{y} \geqslant \mathbf{x}$, and so we have $\bigwedge \uparrow_R (\mathbf{x}) \geqslant \mathbf{x}$.

Lemma 5. $\mathbf{x} \in R$ *if and only if* $\mathbf{x} \in dom(H_R)$ *and* $H_R(\mathbf{x}) \leqslant \mathbf{x}$.

Proof. If $\mathbf{x} \in R$ then $\mathbf{x} \in \uparrow_R(\mathbf{x})$, hence $\uparrow_R (\mathbf{x}) \neq \emptyset$, $\mathbf{x} \in dom(H_R)$, and $H_R(\mathbf{x}) \leqslant \mathbf{x}$. On the other hand, if $\mathbf{x} \in dom(H_R)$ and $H_R(\mathbf{x}) \leqslant \mathbf{x}$, then one has $H_R(\mathbf{x}) = \mathbf{x}$, and since R is meet-closed we have $R(H_R(\mathbf{x}))$, hence $R(\mathbf{x})$. □

We can now characterize the class of relational problems which can be solved using algorithm D, i.e., the problems which can be expressed by definite inequalities.

Theorem 6 (Representability). (1) *Let* $\Gamma = (P, \mathscr{S})$ *with* P *a meet-semilattice. Then* Γ *is representable in definite form if and only if* Γ *is meet-closed. In particular, if* Γ *is meet-closed, then any constraint set* C *over* Γ *can be represented by a definite and simple constraint set* C' *with* $|C'| \leqslant m(m+2) \cdot |C|$ *where* m *is the maximal arity of a relation in* $\mathscr{S}$.

(2) *Let* $\Gamma_{\leqslant} = (P, \mathscr{S})$ *be a relational constraint problem with inequality,* P *arbitrary poset. Then the following conditions are equivalent*:
(i) $\Gamma_{\leqslant}$ *has minimal solutions.*
(ii) $\Gamma_{\leqslant}$ *is meet-closed and* P *is a meet-semilattice.*
(iii) $\Gamma_{\leqslant}$ *is representable in definite form.*

Proof. (1). ($\Rightarrow$) If Γ is representable in definite form, then for $R \in \mathscr{S}$, $R \subseteq P^k$ one has $Sol(C_R) = R$ with $C_R = \{R(\alpha_1, \ldots, \alpha_k)\}$. By assumption there is a definite set C over a problem Φ with $Sol(C) = Sol(C_R) = R$. But for $\tau \in T_\Phi[\![\tau]\!]$ is monotone, hence for any $\rho, \rho' \in Sol(C)$ and any definite inequality $\tau \leqslant A$ in C one has $[\![\tau]\!](\rho \sqcap \rho') \leqslant [\![\tau]\!]\rho \sqcap [\![\tau]\!]\rho' \leqslant [\![A]\!]\rho \sqcap [\![A]\!]\rho' = [\![A]\!](\rho \sqcap \rho')$, hence any definite set C has $Sol(C)$ meet-closed, hence R must be meet-closed.

($\Leftarrow$) If Γ is meet-closed, then (Lemma 5) Γ is representable in *definite* form, hence it can be solved using algorithm D or $D^\top$, as follows. For every $R \in \mathscr{S}$ a term $t = R(A_1, \ldots, A_m)$ is equivalent, by Lemma 5, to $H_R(A_1, \ldots, A_m) \leqslant (A_1, \ldots, A_m)$, which can be written in definite form as $C_t = \{H_R^i(A_1, \ldots, A_m) \leqslant A_i \mid i = 1 \ldots m\}$, with

$H^i_R(\mathbf{x}) = [H_R(\mathbf{x})]_i$, all monotone (when defined), by Lemma 5. Note that $|C_t| = a_R(|t| + 2) = a_R(a_R + 2)$. If P is not a lattice, we use algorithm $D^\top$ starting with a set of extended relations $\mathscr{S}^\top = \{R \cup \top^{a_R} \mid R \in \mathscr{S}\}$ with $\top^{a_R}$ the top element in $(P^\top)^{a_R}$; then all functions $H^i_R : P^m \to P$ are total and monotone. Hence, solving a set C over a meet-closed problem $\Gamma = (P, \mathscr{S})$ is equivalent to solving the definite and simple constraint set $C' = \bigcup_{t \in C} C_t$ over $\Phi = (P, F)$ with $F = \bigcup_{R \in \mathscr{S}} \{H^i_R \mid \in \{1, \ldots, a_R\}\}$.

(2). (i) ⇒ (ii) Since $\Gamma_\leqslant$ has inequality, we can assume w.l.o.g. that $\mathscr{S}$ contains the order relation $\leqslant$ on P. Then $Sol(\{\alpha \leqslant \alpha\}) = P$, hence (by minimality of solutions) P has a bottom element, $\bot$. For any $x, y \in P$, consider $C = \bigcup_{z \in \downarrow(\{x,y\})} \{z \leqslant \alpha\}$, where $\downarrow(\{x, y\}) = \{z \in P \mid z \leqslant x, z \leqslant y\}$. Since P has a bottom element, we have $\downarrow(\{x, y\}) \neq \emptyset$, and since $x, y \in Sol(C)$ it follows by minimality of solutions that $Sol(C)$ has a least element, which is $\bigvee \downarrow(\{x, y\}) = x \sqcap y$. This shows that P is a meet-semilattice. To see that Γ is meet-closed, let $R \in \mathscr{S}$ and $x, y \in R$, and consider $C_{x,y} = (\bigcup_{j=1}^{a_R} \{\alpha_j \leqslant [x \sqcap y]_j\}) \cup \{R(\alpha_1, \ldots, \alpha_{a_R})\}$. Since $x, y \in Sol(C_{x,y})$, $Sol(C_{x,y})$ has a least element, z. Then $z \geqslant x \sqcap y$ by construction of $C_{x,y}$, but also $z \leqslant x \sqcap y$ because $x, y \in Sol(C_{x,y})$. Moreover, $z \in R$ by construction of $C_{x,y}$, and hence $x \sqcap y = z \in R$. We have shown (i) ⇒ (ii). The implication (ii) ⇒ (iii) follows from item (1) of the present theorem, and (iii) ⇒ (i) is true, since by Theorem 3 any definite problem with solutions has minimal solutions.

Observe that property 1 of Theorem 6 can be seen as a strict generalization of the well-known fact that a set $R \subseteq \mathbf{2}^k$ of boolean vectors is definable by a set of propositional Horn-clauses if and only if R is closed under conjunction.[3] Under this view, the notion of *definite* inequalities generalizes the notion of Hornpropositions from the boolean case of **2** to an arbitrary meet-semilattice.[4]

We have seen that any meet-closed problem can be represented by definite, functional constraints. Conversely, consider *distributive constraint sets* over an MFP, i.e., constraint sets C where all $\tau \leqslant \tau' \in C$ have the *right-hand side* τ' built from distributive functions only.[5] Distributive sets strictly include the definite ones, but every distributive set can be represented by a relational set over a meet-closed problem, since the functions in F can be regarded as relations (graphs), hence (Theorem 6(1)) algorithm D can solve any Φ-SAT problem restricted to distributive constraint sets. In practice, it may be convenient to translate a distributive set directly into a definite one, using the auxiliary functions H_g, defined thus: let $g : P^m \to P$ with P a meet-semilattice; then

[3] The definability condition for propositional Horn-clauses is a special case of a much more general model theoretic characterization of Horn-definability in first-order predicate logic, by which an arbitrary first-order sentence ϕ is logically equivalent to a Horn-sentence if and only if ϕ preserves reduced products of models; see [6] or [17] for in-depth treatments of this result. See also [9].

[4] Note that in definite inequalities we are allowed to use any monotone functions, whereas in the special case of Horn-implications we may use only one function, the meet operation. It is easy to see that one cannot, in general, define an arbitrary meet-closed relation using only the meet operation of the semilattice, since, for instance, any set C of inequalities in one variable using only the meet function must be *convex* (i.e., $x, y \in Sol(C)$ and $x \leqslant z \leqslant y$ imply $z \in Sol(C)$.) So, for instance, taking L to be the three element chain $0 < 1 < \infty$, the subset $\{\infty, 0\}$ is meet-closed but not convex and hence it cannot be defined using just the meet operation.

[5] A function f is distributive if $f(x \sqcap y) = f(x) \sqcap f(y)$.

$H_g : P \to P^m$ is the *partial* function given by $H_g(x) = \bigwedge\{y \in P^m \mid x \leqslant g(y)\}$ with H_g *undefined* if no $y \in P^m$ satisfies $y \leqslant g(x)$. If P is a lattice, then H_g is a total function, and it is always monotone. We have

Lemma 7. *Let $f : P^n \to P$, $g : P^m \to P$ with g distributive. Then $f(x) \leqslant g(y)$ if and only if $f(x) \in dom(H_g)$ and $H_g(f(x)) \leqslant y$.*

Proof. Assume $f(x) \leqslant g(y)$. Then $y \in \{z \mid f(x) \leqslant g(z)\}$, hence $f(x) \in dom(H_g)$ with $H_g(f(x)) \leqslant y$. Assume $f(x) \in dom(H_g)$ with $H_g(f(x)) \leqslant y$. Then $(g \circ H_g)(f(x)) \leqslant g(y)$, since g is (distributive and therefore) monotone. For any $w \in dom(H_g)$ one has, by distributivity of g, that $(g \circ H_g)(w) = \bigwedge\{g(z) \mid w \leqslant g(z)\} \geqslant w$. Hence we have $f(x) \leqslant (g \circ H_g)(f(x)) \leqslant g(y)$.

The transformation $\to_R$ given by

$$C \cup \{A \leqslant g(\ldots h(\tau) \ldots)\} \to_R C \cup \{A \leqslant g(\ldots v_{m+1} \ldots), v_{m+1} \leqslant h(\tau)\}$$

is analogous to L-normalization and satisfies properties corresponding to Lemmas 1 and 2. For MFP $\Phi = (L, F)$, L meet-semilattice, let $\Phi' = (L, F')$ with $F' = F \cup \bigcup_{g \in F_\mathrm{d}} \{H_g^i \mid i = 1 \ldots a_g\}$ where F_d is the set of distributive functions in F and $H_g^i(x) = [H_g(x)]_i$. Using $\to_L$ and $\to_R$ one has by Lemma 7

Proposition 8. *Let $\Phi = (P, F)$ be an MFP, P a meet-semilattice. Then for any distributive constraint set C over Φ there exists a definite and simple constraint set C' over Φ' with $Sol(C) = \{\rho' \downarrow k \mid \rho \in Sol(C')\}$, where $Var(C) = \mathscr{V}_k$, and with $|C'| \leqslant 3|C| + 2n(m+2)$ where n is the number of inequalities in C and m is the maximal arity of a function in F.*

Proof. We describe a transformation from C to C' in several steps, as follows. Let $C = \bigcup_{i=1}^{n}\{\tau_i \leqslant \tau_i'\}$ be a given distributive set over Φ with $Var(C) = \mathscr{V}_{\|}$.
1. Split C into a definite set C_L and an anti-definite C_R, with $C_L = \bigcup_{i=1}^{n}\{\tau_i \leqslant v_{k+i}\}$ and $C_R = \bigcup_{i=1}^{n}\{v_{k+i} \leqslant \tau_i'\}$. Then we have $|C_L| + |C_R| = |C| + 2n$.
2. Transform C_L to L-normal form, C_L^*, and transform C_R to R-normal form, C_R^*. We have $|C_L^*| \leqslant 3|C_L|$ and $|C_R^*| \leqslant 3|C_R|$, by Lemma 1 and corresponding properties for R-normalization.
3. For every inequality $\alpha \leqslant g(A_1, \ldots, A_m)$ in C_R^* we have, by Lemma 7, the equivalent inequality $H_g(\alpha) \leqslant (A_1, \ldots, A_m)$, which in turn can be transformed into the equivalent set

   $$\bigcup_{i=1}^{m}\{H_g^i(\alpha) \leqslant A_i\}$$

   with $H_g^i(x) = [H_g(x)]_i$. Let C_R' denote the set which results from performing this transformation on every inequality of the form $\alpha \leqslant g(A_1, \ldots, A_m)$ in C_R^*. Then $|C_R'| \leqslant 2n(m-1) + |C_R^*|$.

Taking $C' = C_L^* \cup C_R'$, it is clear that C' is a definite and simple constraint set over the monotone problem $\Phi = (P, F')$. By Lemma 2 (and corresponding properties for R-normalization), we clearly have $Sol(C) = \{\rho' \downarrow k \mid \rho' \in Sol(C')\}$. As for the size of C', we have

$$\begin{aligned}
|C'| &= |C_L^*| + |C_R'| \\
&\leqslant 3|C_L| + 2n(m-1) + 3|C_R| \\
&= 3(|C_L| + |C_R|) + 2n(m-1) \\
&= 3(|C| + 2n) + 2n(m-1) \\
&= 3|C| + 2n(m+2)
\end{aligned}$$

4.2. Boolean representation

We show how sets of definite inequalities over finite lattices can be translated into propositional formulae, such that there is a direct correspondence between solutions to the propositional system and solutions to the lattice inequalities.

Given a lattice L with $n+1$ elements, we represent each element of L by an element in 2^n. First we number the elements in $L \backslash \{\bot\} = \{l_1, \ldots, l_n\}$. We then represent each element x in L by a vector of boolean values $\phi(x) : L \to 2^n$ where

$$\phi(x) = (b_1, \ldots, b_n) \quad \text{where } b_i = 1 \text{ iff } l_i \leqslant x$$

We also define a mapping $\psi : 2^n \to L$:

$$\psi((b_1, \ldots, b_n)) = \bigvee \{l_i \mid b_i = 1\}$$

It is clear that $x = \psi(\phi(x))$ and $v \leqslant \phi(\psi(v))$. Moreover, both are monotone. Hence, ϕ and ψ form a Galois connection between L and 2^n. We will translate definite inequalities over lattice terms into sets of definite inequalities over 2^n. We will assume that we have already transformed the constraints to the form $f(A_1, \ldots, A_{a_f}) \leqslant A_0$. We translate constraints over the L-variables $v_1, \ldots, v_k$ into sets of constraints over the boolean variables $v_{11}, \ldots, v_{1n}, \ldots, v_{k1}, \ldots, v_{kn}$. We extend ϕ to variables by setting $\phi(v_i) = (v_{i1}, \ldots, v_{in})$ and define $\phi_i(x)$ to be the ith component of $\phi(x)$.

Even though we do not use an index for $\bot$ in our representation, we will for convenience assign it index 0 and define $\phi_0(x) = 1$ for all $x \in L$. This corresponds to extending the representation vector with an extra bit, which is always 1 (since $\bot$ is $\leqslant$ all elements in L).

We first generate, for each variable v_i, the set of constraints $v_{ik} \leqslant v_{ij}$ whenever $l_j \leqslant l_k$ (we actually need only do so for a set of pairs $l_j \leqslant l_k$ whose transitive and reflexive closure yields the ordering on L). These constraints will ensure that any solution to the constraint set will be in the image of ϕ. Note that this only works because we are in a lattice, since whenever we have two solutions to the constraints for a variable, the meet and join of these are also solutions. Even if we use more general Horn-clauses to model the ordering relation, it will be meet-closed, so at best we can extend the construction to meet semilattices.

Let the *ith frontier of f*, $F_i(f)$ be the smallest subset of L^{a_f} such that $\phi_i(f(y_1,\dots,y_{a_f}))=1$ iff there exist $(x_1,\dots,x_{a_f})\in F_i(f),(x_1,\dots,x_{a_f})\leqslant(y_1,\dots,y_{a_f})$. This is well defined, because the set of all $(x_1,\dots,x_{a_f})$ such that $\phi_i(f(x_1,\dots,x_{a_f}))=1$ is one such, and since the intersection of two such sets is also one. While $F_i(f)$ in the worst case may be of size $O(|L^{a_f}|)$, it is often smaller. If f is distributive, $|F_i(f)|\leqslant 1$.

For each i between 1 and n and for each $(x_1,\dots,x_{a_f})\in F_i(f)$, we generate from the constraint $f(A_1,\dots,A_{a_f})\leqslant A_0$ a new constraint:

$$\phi_{j_1}(A_1)\sqcap\cdots\sqcap\phi_{j_{a_f}}(A_{a_f})\leqslant\phi_i(A_0)$$

where j_k is the index of x_k in L.

The translation of $f(A_1,\dots,A_{a_f})\leqslant A_0$ is the set of all these new constraints.

Theorem 9. *The constraint $f(A_1,\dots,A_{a_f})\leqslant A_0$ is satisfiable iff all the constraints in the translation and all the ordering constraints between the components of the variables are. Moreover, any solution to the translation is in the image of ϕ and maps by ψ to a solution of $f(A_1,\dots,A_{a_f})\leqslant A_0$. Since ϕ and ψ are order-preserving, least solutions map to least solutions.*

Proof. Let us assume that $f(A_1,\dots,A_{a_f})\leqslant A_0$ has a solution ρ. Hence, we have $f(\rho(A_1),\dots,\rho(A_{a_f}))\leqslant\rho(A_0)$, and thus either the right-hand size $\phi_i(\rho(A_0))=1$ or the left-hand side $\phi_i(f(\rho(A_1),\dots,\rho(A_{a_f}))=0$. In the first case, the constraint

$$\phi_{j_1}(\rho(A_1))\sqcap\cdots\sqcap\phi_{j_{a_f}}(\rho(A_{a_f}))\leqslant\phi_i(\rho(A_0))$$

is trivially true. In the second case, there is by the definition of $F_i(f)$ no $(x_1,\dots,x_{a_f})\in F_i(f),(x_1,\dots,x_{a_f})\leqslant(\rho(A_1),\dots,\rho(A_{a_f}))$. Hence, there must be a k such that $\neg(x_k\leqslant\rho(A_k))$. Hence, $\phi_{j_k}(\rho(A_k))=0$, and hence the entire left hand side of the inequality evaluates to 0 which satisfies the inequality. The constraints that ensure that a solution is in the image of ϕ are trivially satisfied, so the complete set of constraints are.

Conversely, let us assume that we have a solution ρ' to all the constraints that ensure each variable-tuple is in the image of ϕ and all constraints of the form

$$\phi_{j_1}(A_1)\sqcap\cdots\sqcap\phi_{j_n}(A_{a_f})\leqslant\phi_i(A_0)$$

We define a mapping ρ of the variables in the original constraint: $\rho(v_i)=\psi(\rho'(v_{i1}),\dots,\rho'(v_{in}))$. Since ρ' maps variable-tuples to images of ϕ, we have $\rho'(\phi_j(A))=\phi_j(\rho(A))$ for any j and any atom A in the original constraint set.

Let us further assume that ρ is *not* a solution to the original constraint. Then for some i, $\phi_i(f(\rho(A_1),\dots,\rho(A_{a_f}))=1$ and $\phi_i(\rho(A_0))=0$. By the above, we have $\rho'(\phi_i(A_0))=\phi_i(\rho(A_0))=0$. Since, by assumption, $\phi_i(f(\rho(A_1),\dots,\rho(A_{a_f}))=1$, there must be an $(x_1,\dots,x_{a_f})\in F_i(f),(x_1,\dots,x_{a_f})\leqslant(\rho(A_1),\dots,\rho(A_{a_f})$. This means that for all $k\in\{1,\dots,a_f\},\phi_{j_k}(\rho(A_k))=1$. Hence, $\phi_{j_1}(\rho(A_1))\sqcap\cdots\sqcap\phi_{j_n}(\rho(A_{a_f}))=1$ and the constraint

$$\rho'(\phi_{j_1}(A_1))) \sqcap \cdots \sqcap \rho'(\phi_{j_n}(A_{a_f})) \leqslant \rho'(\phi_i(A_0))$$

is not solved, which is in contradiction with our first assumption.

Complexity. We have translated a set C of constraints over an $(n+1)$-point lattice L into a set C' of constraints over the boolean 2-point lattice. The size of the translated constraint set can be calculated as follows:

For any variable v_i in C, we introduce n variables $v_{i1},\dots,v_{in}$ in C'. For each v_i, C' contains at most n^2 constraints to ensure that any solution to C' will map $v_{i1},\dots,v_{in}$ to an image of an element in L.

For any constraint $f(A_1,\dots,A_{a_f}) \leqslant A_0$, we introduce a number of constraints, each of size a_f+1. The number of constraints is $\sum_{i=1}^{n} |F_i(f)| \leqslant n \times n^{a_f}$. Since the size of the constraint $f(A_1,\dots,A_{a_f}) \leqslant A_0$ is (a_f+1), the size of the translation is less than n^{1+a_f} times the size of the original constraint.

Bringing these together, we get that $|C'| \leqslant n^{1+a_{\max}} \times |C| + n^2 \times |V|$, where $a_{\max}$ is the maximal arity of a function symbol in C and $|V|$ is the number of variables in C. For a fixed lattice and set of function symbols, this is a linear expansion. For the uniform problem, the input is given as operation matrices for the function symbols plus the constraints. The size of an operation matrix for a function with arity a is n^a, so the size of the input is greater than $n^{a_{\max}} + |C|$. The size of the output is $n^{1+a_{\max}} \times |C| + n^2 \times |V|$. The size of the input is hence the sum of two values and the size of the output is (approximately) the product of these. Hence, we get a quadratic worst-case expansion for the uniform problem.

The exponential dependence on the arity of the function symbols may seem bad, but it can be argued that any reasonable translation to boolean constraints will expand non-polynomially in the arity of the function symbols. We assume that each constraint is translated individually. Hence, each occurrence of a function symbol f in the constraint set will in the translation contain enough information to uniquely represent f. The number of monotone functions over a lattice L^a can be n^{n^a}, where $n = \mathrm{O}(|L|)$. Hence, a uniform representation of functions from $L^a \to L$ using a fixed alphabet will in the worst case use at least $\mathrm{O}(n^a)$ symbols. Since constraints contain variables as well as the fixed boolean symbols, the actual figure is a bit smaller, but given a reasonable limit on the number of variables used, it is still more than polynomial. Comparing with algorithm D (see Theorem 4) we see that D runs in time linearly dependent on $a_{\max}$ for the uniform case, hence boolean representation should, in general, only be used in case arities are known to be small.

Equivalence. The translation of constraint sets produce equivalent instances of the HORNSAT-problem: Each constraint in the translation is of the form $A_1 \sqcap \cdots \sqcap A_m \leqslant A_0$, where the A_i are variables or constants ranging over the lattice $\{0,1\}$. These constraints are isormorphic to Horn-clauses, and can hence be solved in time linear in the size of the constraint set using a HORNSAT-procedure [10].

5. Intractability of extensions

We have seen that algorithm D efficiently decides the *uniform* satisfiability problem (i.e., uniform in both Γ and C), when instances Γ are restricted to be meet-closed. It is relevant to ask whether this can be extended to cover more relations than the meet-closed ones (perhaps by finding an algorithm entirely different from D). The main purpose of this section is to demonstrate that, unless $\mathbf{P}=\mathbf{NP}$, *no* such extension is possible for any meet-semilattice L. This shows that algorithm D is complete for a maximal tractable class of problems, namely the meet-closed ones. We proceed to give some technical definitions which will be needed to prove this result.

If L is a meet-semilattice, we say that a problem $\Gamma=(L,\mathscr{S})$ is a *maximal* meet-closed problem if Γ is meet-closed and for any $R\subseteq L^k$ which is not meet-closed, the (particular, non-uniform) satisfiability problem $(L,\mathscr{S}\cup\{R\})$-SAT is NP-complete. We first show that *any distributive lattice has a maximal meet-closed problem* and deal with the general case afterwards. The proof is by composing Birkhoff's Representation Theorem for finite lattices[6] with Schaefer's Dichotomy Theorem [26] for the complexity of logical satisfiability problems.

For lattice L, let $\mathbf{Idl}(L)$ be the set of *order ideals* of L and $\mathbf{Irr}(L)$ the set of *join-irreducible* elements of L. If L is a finite, *distributive* lattice with $\mathbf{Irr}(L)=c_1,\ldots,c_n$ any fixed enumeration of $\mathbf{Irr}(L)$, then Birkhoff's Representation Theorem entails that, with $\eta:L\to\mathbf{Idl}(\mathbf{Irr}(L))$ defined by $\eta(x)=\{y\in\mathbf{Irr}(L)\mid y\leqslant x\}$, the map $\varphi:L\to\mathbf{2}^n$ becomes an *order-embedding* by setting $[\varphi(x)]_i=1$, if $c_i\in\eta(x)$, and $[\varphi(x)]_i=0$, if $c_i\notin\eta(x)$, for $i=1,\ldots,n$. We refer to φ as the *canonical embedding* of L. For $R\subseteq L^k$ we let[7] $\varphi(R)=\{\langle\varphi(x_1),\ldots,\varphi(x_k)\rangle\mid(x_1,\ldots,x_k)\in R\}$, so $\mathbf{x}\in R\Leftrightarrow\varphi(\mathbf{x})\in\varphi(R)$. With $\Gamma=(L,\mathscr{S})$, L distributive, define the problem $\varphi(\Gamma)=(\mathbf{2},\varphi(\mathscr{S}))$ with $\varphi(\mathscr{S})=\{\varphi(R)\mid R\in\mathscr{S}\}$. We use $\varphi(R)$ to denote the relation symbol corresponding to the relation $\varphi(R)$. If C is a constraint set over $\varphi(\Gamma)$, one has $\varphi(R)\subseteq\mathbf{2}^{kn}$ for all relations $\varphi(R)\in\varphi(\mathscr{S})$, where $n=|\mathbf{Irr}(L)|$ and k is the arity of R.

Now, we are interested in problems Γ such that $\varphi(\Gamma)$-SAT becomes polynomial time reducible to Γ-SAT. The problem here is that such reduction is not possible in general, because the constraint language of $\varphi(\Gamma)$ is more expressive than that of Γ. For instance, a unary relation symbol R may get translated into a symbol $\varphi(R)$ of arity $n=|\mathbf{Irr}(L)|$ with $n>1$, so we can write (taking $n=3$) constraints with patterns like $\{\varphi(R)(x,y,z),\varphi(R)(z,y,x)\}$ expressing that there exists $\mathbf{b}\in\mathbf{2}^3$ such that both it and its reversal is in $\varphi(R)$; in general this cannot be expressed in the constraint language

[6] See [8]. We recall also that, for lattice L, an *order ideal* in L is a down-closed subset of L; $x\in L$ is *join-irreducible* if $x\neq\bot$ and $x=y\sqcup z$ implies $x=y$ or $x=z$ for all $y,z\in L$; L is *distributive* if $(x\sqcup y)\sqcap z=(x\sqcap z)\sqcup(y\sqcap z)$ for all $x,y,z\in L$; an *order-embedding* of lattices is an injective map preserving meet and join.

[7] We use the notation $\langle\mathbf{y}_1,\ldots,\mathbf{y}_k\rangle$, for $\mathbf{y}_i\in L^n$, to denote the "flattened" kn-vector $\mathbf{z}$ obtained by concatenating the tuples $\mathbf{y}_1,\ldots,\mathbf{y}_k$ into a single tuple, in that order; in detail, $\mathbf{z}=(z_1,\ldots,z_{kn})$ with $z_1=[\mathbf{y}_1]_1,\ldots,z_n=[\mathbf{y}_1]_n,\ldots,z_{(k-1)n+1}=[\mathbf{y}_k]_1,\ldots,z_{kn}=[\mathbf{y}_k]_n$. For $\mathbf{x}=(x_1,\ldots,x_k)\in L^k$ we write $\varphi(\mathbf{x})$ for $\langle\varphi(x_1),\ldots,\varphi(x_k)\rangle$.

of Γ. However, if Γ has a certain kind of relations, then this becomes possible. Given distributive lattice L with canonical embedding φ and $|\mathbf{Irr}(L)| = n$ we let Π_L denote the set of "projection relations" $\pi_{ij} \subseteq L^2$, $i, j \in \{1, \ldots, n\}$, defined as follows:

$$\pi_{ij} = \{(x, y) \in L^2 \mid [\varphi(x)]_i = [\varphi(x)]_j\}$$

It is easy to check that π_{ij} are all meet-closed relations, since φ is an order-embedding. If L is non-trivial, there must be $a, b \in L$ with $a < b$, so that $\varphi(a) < \varphi(b)$, and hence $[\varphi(a)]_j = 0$ and $[\varphi(b)]_j = 1$ for some j; we can therefore express that $[\varphi(x)]_i = 0$ and $[\varphi(x)]_i = 1$ by $\pi_{ij}(x, a)$ and $\pi_{ij}(x, b)$, respectively. Constraints of the form $\pi_{ij}(x, y)$ will be written as $[\varphi(x)]_i = [\varphi(y)]_j$ or $[\varphi(x)]_i = b$ ($b \in \{0, 1\}$). One does not, of course, need explicit reference to φ in order to define the projection relations, so long as one can talk about the join-irreducible elements in an appropriate way:

Example 3. Let $\Phi = (L, F)$, $\mathbf{Irr}(L) = c_1, \ldots, c_n$ and suppose we have a function $f_{c_i} \in F$ for each $c_i \in \mathbf{Irr}(L)$ where $f_{c_i}(x) = \top$ if $c_i \leqslant x$ and $f_{c_i}(x) = \bot$ otherwise. Then all f_{c_i} are distributive functions, and since the condition $[\varphi(x)]_i = [\varphi(y)]_j$ is equivalent to the condition $c_i \leqslant x \Leftrightarrow c_j \leqslant y$, the first mentioned condition can be expressed by distributive inequalities over Φ, because $c_i \leqslant x \Rightarrow c_j \leqslant y$ is equivalent to the distributive constraint $f_{c_i}(x) \leqslant f_{c_j}(y)$.

If $\Gamma = (L, \mathscr{S})$ and C is a constraint set over $\varphi(\Gamma)$, we describe a translation of C, called $\varphi^{-1}(C)$, to a constraint set over $\Gamma' = (L, \mathscr{S} \cup \Pi_L)$, as follows. Let $t_1, \ldots, t_m$ be an enumeration of the constraint terms in C. Each t_s can be written as a term of the form $t_s = \varphi(R)(\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle)$, where each $\mathbf{A}_i$ is a vector of n atomic terms, $n = |\mathbf{Irr}(L)|$, k the arity of R. We let $t_s \Downarrow (p, i) = [\mathbf{A}_p]_i$ ($1 \leqslant p \leqslant k, 1 \leqslant i \leqslant n$.) For each term t_s we let $\alpha_1^s, \ldots, \alpha_k^s$ be k unique, fresh variables to be used in the translation of term t_s. The relational term $t_s = \varphi(R)(\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle)$ then gets translated into the constraint set $C_s = C_{s,1} \cup C_{s,2} \cup C_{s,3}$, where

$$\begin{aligned} C_{s,1} &= \{R(\alpha_1^s, \ldots, \alpha_k^s)\} \\ C_{s,2} &= \{[\varphi(\alpha_j^s)]_i = b \mid [\mathbf{A}_j]_i = b \text{ with } b \in \{0, 1\}\} \\ C_{s,3} &= \{[\varphi(\alpha_p^s)]_i = [\varphi(\alpha_q^u)]_j \mid t_s \Downarrow (p, i) \text{ is variable } x \wedge \exists t_u \in C.\, t_u \Downarrow (q, j) = x\} \end{aligned}$$

For $C = t_1, \ldots, t_m$ we define $\varphi^{-1}(C) = \bigcup_{s=1}^{m} C_s$. The constraints in $\varphi^{-1}(C)$ using π_{ij} can simulate all patterns in $\varphi(C)$, leading to

Lemma 10. *Let $\Gamma = (L, \mathscr{S})$ with L non-trivial distributive lattice, and let C be a constraint set over $\varphi(\Gamma)$. Then C is satisfiable over $\varphi(\Gamma)$ if and only if $\varphi^{-1}(C)$ is satisfiable over $\Gamma' = (L, \mathscr{S} \cup \Pi_L)$.*

Proof. ($\Rightarrow$) Assume C satisfiable over $\varphi(\Gamma)$. Let ρ be a solution to C and define the valuation ρ' by $\rho'(\alpha_q^u) = \varphi^{-1}(\rho(\mathbf{A}_q))$, where $\mathbf{A}_q$ is the qth argument vector in the term t_u in C (t_u is the unique component of $\varphi^{-1}(C)$ where the variable α_q^u is introduced, and this variable has exactly one occurrence in $\varphi^{-1}(C)$). This defines ρ' on all the variables in $\varphi^{-1}(C)$.

We now consider the possible constraints in $\varphi^{-1}(C)$, demonstrating that ρ' satisfies them all.

(1) Let $t_s = R(\alpha_1^s, \ldots, \alpha_k^s)$ be a term coming from the component $C_{s,1}$ in $\varphi^{-1}(C)$. Then $\varphi(R)(\langle \mathbf{A}_1, \ldots, \mathbf{A}_k \rangle)$ is a term in C, and so $\langle \rho(\mathbf{A}_1), \ldots, \rho(\mathbf{A}_k)\rangle \in \varphi(R)$, hence

$$(\varphi^{-1}(\rho(\mathbf{A}_1)), \ldots \varphi^{-1}(\rho(\mathbf{A}_k))) \in R, \quad \text{i.e., } (\rho'(\alpha_1^s), \ldots, \rho'(\alpha_k^s)) \in R$$

(2) Consider a $C_{s,2}$-constraint of the form $[\varphi(\alpha_j^s)]_i = b$ in $\varphi^{-1}(C)$. Then $t_s = \varphi(R)(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ is a term in C with $[\mathbf{A}_j]_i = b$, hence $\rho([\mathbf{A}_j]_i) = b$; moreover, $\rho'(\alpha_j^s) = \varphi^{-1}(\rho(\mathbf{A}_j))$, hence $[\varphi(\rho'(\alpha_j^s))]_i = [\varphi(\varphi^{-1}(\rho(\mathbf{A}_j)))]_i = [\rho(\mathbf{A}_j)]_i = \rho([\mathbf{A}_j]_i) = b$, as desired.

(3) Consider a $C_{s,3}$-constraint of the form $[\varphi(\alpha_p^s)]_i = [\varphi(\alpha_q^u)]_j$ in $\varphi^{-1}(C)$. Then $t_s = \varphi(R)(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ and $t_u = \varphi(R')(\mathbf{B}_1, \ldots, \mathbf{B}_r)$ are terms in C with $[\mathbf{A}_p]_i = x = [\mathbf{B}_q]_j$, two different occurrences of variable x. Then $\rho([\mathbf{A}_p]_i) = \rho([\mathbf{B}_q]_j) = \rho(x)$, and so we have $\rho'(\alpha_p^s) = \varphi^{-1}(\rho(\mathbf{A}_p))$ and $\rho'(\alpha_q^u) = \varphi^{-1}(\rho(\mathbf{B}_q))$, hence $[\varphi(\alpha_p^s)]_i = [\rho(\mathbf{A}_p)]_i = \rho(x) = \rho([\mathbf{B}_q]_j) = [\varphi(\alpha_q^u)]_j$, as desired.

($\Leftarrow$) Assume $\varphi^{-1}(C)$ is satisfiable, and let ρ be a solution. We now define a valuation ρ' for C, using the following auxiliary definition. If $x \in \mathit{Var}(C)$, we write (with slight abuse of notation) $\varphi^{-1}(x) = (\alpha_p^s, i)$ where α_p^s is an arbitrary but fixed chosen variable in $\mathit{Var}(\varphi^{-1}(C))$ such that the term $t_s = R(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ is in C with $x = [\mathbf{A}_p]_i$. By construction of $\varphi^{-1}(C)$ such a variable α_p^s and index i can always be found in $C_{s,1}$. Now define valuation ρ' by taking $\rho'(x) = [\varphi(\rho(\alpha_q^u))]_j$ where $\varphi^{-1}(x) = (\alpha_q^u, j)$.

Now let $t_s = \varphi(R)(\mathbf{A}_1, \ldots, \mathbf{A}_k)$ be an arbitrary constraint in C. Then the term $R(\alpha_1^s, \ldots, \alpha_k^s)$ is in $C_{s,1}$, and hence $(\rho(\alpha_1^s), \ldots, \rho(\alpha_k^s)) \in R$, from which it follows that

$$\langle \varphi(\rho(\alpha_1^s)), \ldots, \varphi(\rho(\alpha_k^s))\rangle \in \varphi(R)$$

We now *claim* that

$$\langle \rho'(\mathbf{A}_1), \ldots, \rho'(\mathbf{A}_k)\rangle = \langle \varphi(\rho(\alpha_1^s)), \ldots, \varphi(\rho(\alpha_k^s))\rangle \tag{1}$$

from which it follows that t_s is satisfied under ρ'. To see that the *claim* is true, we consider an arbitrary component $[\mathbf{A}_p]_i$ in $\langle \mathbf{A}_1, \ldots, \mathbf{A}_k\rangle$. There are two cases to consider:

(1) $[\mathbf{A}_p]_i = x$ is a variable. Here we have $\rho'([\mathbf{A}_p]_i) = \rho'(x) = [\varphi(\rho(\alpha_q^u))]_j$ for some u, q, j with t_u a term in C and variable x occurring as the jth component of the qth argument vector of t_u. Therefore, the constraint $[\varphi(\alpha_p^s)]_i = [\varphi(\alpha_q^u)]_j$ is in $C_{s,3}$ (and also, in fact, in $C_{u,3}$) which ρ satisfies, and therefore we get $\rho'(x) = [\varphi(\rho(\alpha_p^s))]_i$, as desired.

(2) $[\mathbf{A}_p]_i = b$ is a logical constant, $b \in \{0, 1\}$. Then $\rho'([\mathbf{A}_p]_i) = b$. Moreover, the constraint $[\varphi(\alpha_p^s)]_i = b$ is in $C_{s,2}$, hence $[\varphi(\rho(\alpha_p^s))]_i = b = \rho'([\mathbf{A}_p]_i)$, as desired.

Items (1) and (2) establish claim (1), and the proof is complete.

The translation of constraint sets incurs an expansion which is at most polynomial in the size of the original set:

Lemma 11. *For every constraint set C over $\varphi(\Gamma)$ one has $|\varphi^{-1}(C)| \leqslant |C|^2 \cdot (2|C| + 3)$.*

Proof. Consider the definition of $C_s = C_{s,1} \cup C_{s,2} \cup C_{s,3}$. The size of $C_{s,1}$ is clearly bounded by $|C|$. The number of constraints in $C_{s,2}$ is clearly bounded by $|C|$, and the size of each of these constraints is 2, hence the size of $C_{s,2}$ is bounded by $2|C|$. The third set $C_{s,3}$ contributes at most $|C|^2$ new constraints, since each new constraint is determined by a distinct pair of occurrences of variables in C. Each of the new constraint terms has size 2, so the size of $C_{s,3}$ is bounded by $2|C|^2$. We then get, for each term $t_s \in C$, that $|C_s| \leqslant 3|C| + 2|C|^2$. Since there are at most $|C|$ terms in C, we get at most $|C|$ sets C_s in the translated set, and so we have $|\varphi^{-1}(C)| \leqslant |C|(3|C| + 2|C|^2) = |C|^2(2|C| + 3)$.

We now recall the contents of Schaefer's Dichotomy Theorem [26]. It yields the following very powerful classification (see also [19]):

Theorem 12 (Schaefer [26]). *Let $\Gamma = (\mathbf{2}, \mathcal{S})$ be any boolean problem. Then the satisfiability problem Γ-SAT is polynomial time decidable, if one of the following four conditions are satisfied: either* (1) *every relation in $\mathcal{S}$ is closed under disjunction, or* (2) *every relation in $\mathcal{S}$ is closed under conjunction, or* (3) *every relation in $\mathcal{S}$ is bijunctive, i.e., it satisfies the closure condition $\forall x, y, z \in \mathbf{2}^k .\ x, y, z \in R \Rightarrow (x \wedge y) \vee (y \wedge z) \vee (z \wedge x) \in R$, or* (4) *every relation in $\mathcal{S}$ is affine, i.e., it satisfies the closure condition $\forall x, y, z \in \mathbf{2}^k .\ x, y, z \in R \Rightarrow x \oplus y \oplus z \in R$, where $\oplus$ is the exclusive disjunction.*

Otherwise, if none of the above conditions are satisfied, the problem Γ-SAT is **NP**-*complete under log-space reductions.*

If L is a meet-semilattice which is not a lattice, we let $L^\top$ denote the extension of L to a lattice under addition of a top element, as earlier. If L is already a lattice, we let $L^\top = L$, by definition. We say that L is a *distributive meet-semilattice* if L is a meet-semilattice such that $L^\top$ is a distributive lattice. If L is a distributive meet-semilattice we know by previous remarks that $L^\top$ has a canonical embedding into $\mathbf{2}^n$, and this map φ will be referred to as the canonical embedding of L also. For any meet-semilattice L we let $M_L = \{(x, y, z) \in L^3 \mid z = x \sqcap y\}$, the meet-relation of L.

Lemma 13. *Let L be a non-trivial distributive meet-semilattice, and let φ be its canonical embedding. Then the relation $\varphi(M_L)$ satisfies: $\varphi(M_L)$ is closed under conjunction, $\varphi(M_L)$ is not closed under disjunction, $\varphi(M_L)$ is not bijunctive, and $\varphi(M_L)$ is not affine.*

Proof. Let L be a non-trivial distributive meet-semilattice with canonical embedding φ, $\mathbf{Irr}(L^\top) = c_1, \ldots, c_n$. Let $a \in L$ be any *atom*, i.e., such that $\bot < a$ and for any $x \in L$ with $\bot < x \leqslant a$ one has $x = a$. Then a is join-irreducible, hence $a = c_i$, for some i, in the enumeration of $\mathbf{Irr}(L^\top)$. Inspection of φ shows that $\varphi(\bot) = \mathbf{0}$ and that $\varphi(a) = \mathbf{0}[1/i]$, where we use the convention, for $\mathbf{x} \in \mathbf{2}^n$, $y \in \mathbf{2}$, of writing $\mathbf{x}[y/j]$ for the vector $\mathbf{z}$ such that $[\mathbf{z}]_k = [\mathbf{x}]_k$ for $k \neq j$ and $[\mathbf{z}]_j = y$. Fixing an atom a in L, define the relation $R_a \subseteq \mathbf{2}^3$ by

$$R_a = \{(x, y, z) \in \mathbf{2}^3 \mid \langle \mathbf{0}[x/i], \mathbf{0}[y/i], \mathbf{0}[z/i] \rangle \in \varphi(M_L)\}$$

where i is the unique index such that $[\varphi(a)]_i = 1$. We *claim* that $R_a = M_\mathbf{2}$. To see that $R_a \subseteq M_\mathbf{2}$, consider that, since φ is an order embedding preserving meets, one has by definition of M_L that for any $\langle \mathbf{0}[x/i], \mathbf{0}[y/i], \mathbf{0}[z/i]\rangle \in \varphi(M_L)$ it will be the case that $z = x \wedge y$. To see that $M_\mathbf{2} \subseteq R_a$, consider that $\varphi(a) = \mathbf{0}[1/i]$ and $\varphi(\bot) = \mathbf{0}$, hence $\{[\varphi(a)]_i, [\varphi(\bot)]_i\} = \{0, 1\}$, and therefore any combination $(x, y) \in \mathbf{2}^2$ can be written as $(x, y) = ([u]_i, [v]_i)$ for $u, v \in \{\varphi(a), \varphi(\bot)\}$ and with $(u, v, u \wedge v) \in R_a$. It follows that, for any $(x, y) \in \mathbf{2}^2$ one has $(x, y, x \wedge y) \in R_a$.

We now show that $M_\mathbf{2}$ is neither closed under disjunction, nor bijunctive, nor affine. It is obvious that $M_\mathbf{2}$ is not closed under disjunction, since $(0,1,0), (1,0,0) \in M_\mathbf{2}$ but $(1,1,0) \notin M_\mathbf{2}$. To see that $M_\mathbf{2}$ is not bijunctive, let $\mathbf{x} = (0,1,0)$, $\mathbf{y} = (1,1,1)$, $\mathbf{z} = (1,0,0)$. We have $\mathbf{x}, \mathbf{y}, \mathbf{z} \in M_\mathbf{2}$, but $(\mathbf{x} \sqcap \mathbf{y}) \vee (\mathbf{y} \sqcap \mathbf{z}) \vee (\mathbf{z} \sqcap \mathbf{x}) = (1,1,0) \notin M_\mathbf{2}$. To see that $M_\mathbf{2}$ is not affine, consider that $\mathbf{x}, \mathbf{y}, (0,0,0) \in M_\mathbf{2}$ (with $\mathbf{x}, \mathbf{y}$ as above) but $\mathbf{x} \oplus \mathbf{y} \oplus (0,0,0) = (1,1,0) \notin M_\mathbf{2}$.

Since $M_\mathbf{2} = R_a$ and $M_\mathbf{2}$ satisfies none of conditions (1), (3), (4) of Theorem 12, it follows from the definition of R_a that $\varphi(M_L)$ does not satisfy any of these conditions. On the other hand, it is evident that $\varphi(M_L)$ is closed under conjunction, because M_L is meet-closed and φ preserves meets.

Let L be distributive and $\mathscr{S} = \Pi_L \cup \{M_L\}$; then the problem $\tilde{\Gamma} = (L, \mathscr{S})$ is meet-closed. We can show that, for *any* relation R over L which is not meet-closed, the problem Γ^+-SAT with $\Gamma^+ = (L, \mathscr{S} \cup \{R\})$ is **NP**-hard, by reduction from $\varphi(\Gamma)$-SAT, which, in turn, can be shown to be **NP**-hard by Lemmas 10, 11 and 13 together with Schaefer's Dichotomy Theorem:

Theorem 14 (Intractability of extensions, distributive case). *For any non-trivial, distributive meet-semilattice L the problem $\tilde{\Gamma} = (L, \Pi_L \cup \{M_L\})$ is maximal meet-closed.*

Proof. Let L be a non-trivial, distributive meet-semilattice. Let φ be its canonical embedding, mapping $L^\top$ into $\mathbf{2}^n$ with $n = |\mathbf{Irr}(L^\top)|$. By Lemma 13 the relation $\varphi(M_L)$ is meet-closed but not join-closed, not bijunctive and not affine. Let $\mathscr{S} = \Pi_L \cup \{M_L\}$; then the problem $\tilde{\Gamma} = (L, \mathscr{S})$ is meet-closed. We *claim* that, for *any* relation R over L which is not meet-closed, the problem Γ^+-SAT with $\Gamma^+ = (L, \mathscr{S} \cup \{R\})$ is **NP**-hard.

First assume $L = L^\top$. To prove Γ^+-SAT **NP**-hard, we first show that, with $\Gamma = (L, \{R, M_L\})$, the problem $\varphi(\Gamma)$-SAT is **NP**-complete, for *any* $R \subseteq L^k$ which is not meet-closed. We then show that Γ^+-SAT is **NP**-complete by reduction from $\varphi(\Gamma)$-SAT.

To see that $\varphi(\Gamma)$-SAT is **NP**-complete, note that, since $\varphi(R)$ cannot be meet-closed (since R is not meet-closed and φ is order-embedding), and (by Lemma 13) $\varphi(M_L)$ is meet-closed but in no other of the tractable classes defined by Schaefer, the set $\{\varphi(R), \varphi(M_L)\}$ cannot belong to any of Schaefer's tractable classes. It then follows from Theorem 12 that $\varphi(\Gamma)$-SAT is **NP**-complete. Now to reduce $\varphi(\Gamma)$-SAT to Γ^+-SAT, let C be a constraint set over $\varphi(\Gamma)$. Then $\varphi^{-1}(C)$ is a constraint set over Γ^+, and by Lemma 11 we have $|\varphi^{-1}(C)|$ polynomially bounded by $|C|$; moreover, Lemma 10 shows that $\varphi^{-1}(C)$ is equivalent to C, hence φ^{-1} defines a polynomial time reduction from $\varphi(\Gamma)$-SAT to Γ^{+}-SAT. This proves the theorem for $L = L^{\top}$.

Assume $L \neq L^{\top}$. Since $R \subseteq L^k$, $M_L \subseteq L^6$ and φ is an order-embedding, $\varphi(\top)$ cannot be a component in any of $\varphi(R)$ or $\varphi(M_L)$. Therefore, the translation of constraint sets in Lemma 10 using projection relations over the lattice $L^{\top}$ can equally well be carried out using the projection relations Π over L, yielding a set $\varphi^{-1}(C)$ over Γ^{+} equivalent to C over $\varphi(\Gamma)$. From this it follows that the problem Γ^{+}-SAT is also **NP**-complete in the case where L is not a lattice.

Example 4. The problem $\Phi = (\mathbf{2}, \{\sqcap\})$ is maximal meet-closed. To see this, represent Φ as relational problem $\Gamma = (\mathbf{2}, \{M_{\mathbf{2}}, \leqslant\})$. It then follows from Theorem 14 that Γ is maximal, because all relations in the set $\Pi_{\mathbf{2}}$ can already be defined in terms of $M_{\mathbf{2}}$, by Horn-definability (cf. comment after Theorem 6) together with the fact that all relations in $\Pi_{\mathbf{2}}$ are meet-closed.

By Birkhoff's Theorem we know that L embeds into $\mathbf{2}^n$ if *and only* if L is distributive, hence the method used to prove Theorem 14 will not work for arbitrary finite lattices. However, if L is an arbitrary meet-semilattice we have the following weaker result:

Theorem 15 (Intractability of extensions, general case). *Let L be any non-trivial meet-semilattice and let $R \subseteq L^k$ be any relation over L which is not meet-closed. Then there exists a meet-closed problem $\Gamma = (L, \mathscr{S})$ such that the problem $\tilde{\Gamma}$-SAT is* **NP**-*complete with $\tilde{\Gamma} = (L, \mathscr{S} \cup \{R\})$.*

Proof. It is possible to obtain the theorem by exploiting Theorem 14. However, it may be instructive to see that the result can be obtained by a simple, direct reduction from CNF-SAT, and this we proceed to give now. We first show how certain relations can be defined when an arbitrary non-meet-closed relation is given, and after that we define Γ, $\tilde{\Gamma}$ and give a polynomial time reduction.

Let $R \subseteq L^k$ be any relation which is not meet-closed. Then there are $a, b \in R$ with $a \sqcap b \notin R$. Let $X = \{a, b, a \sqcap b\}$, let $Y = \{\langle a, b\rangle, \langle b, a\rangle, \langle a \sqcap b, a \sqcap b\rangle\}$, and let Z be the least meet-closed subset of L^{3k} which contains the set $A = \{\langle a, b, a\rangle, \langle a, a, a\rangle, \langle b, a, a\rangle, \langle b, b, b\rangle\}$; then X, Y and Z are all meet-closed relations over L. Given relations U, V over L we write $U \times V = \{\langle x, y\rangle \mid x \in U, y \in V\}$. We can then define the relation N given by $N = (R \times R) \cap Y = \{\langle a, b\rangle, \langle b, a\rangle\}$ and the relation T given by $T = R \cap X = \{a, b\}$; finally, note that we have A definable in terms of Z and R, since $A = Z \cap (R \times R \times R)$.

We now claim that CNF-SAT can be encoded using constraints over the problem $\tilde{\Gamma} = (L, \{R, X, Y, Z\})$. First, observe that the three relations T, A, N are defined in terms of the relations R, X, Y, Z using only the operations $\cap$ and $\times$. It follows that the relations T, A, N can be represented using constraints with relation symbols R, X, Y, Z since the solution set of the constraint set $\{U(x), V(y)\}$ is just $U \times V$ and the solution set of $\{U(x), V(x)\}$ is just $U \cap V$. Next, observe that, if we interpret a as the truth value **false** and b as the truth value **true**, then $T(x)$ holds iff $x \in \{\mathbf{true}, \mathbf{false}\}$, $N(\langle x, y\rangle)$ holds iff $y = \neg x$ and $A(\langle x, y, z\rangle)$ holds iff $z = x \wedge y$. It easily follows that any instance of CNF-SAT can be represented, in polynomial time, by an equivalent constraint set over $\tilde{\Gamma}$. Take $\Gamma = (L, \{X, Y, Z\})$, then Γ is a meet-closed problem with the desired property.

Theorem 15 entails that the uniform satisfiability problem restricted to meet-closed relations becomes **NP**-hard no matter how it is extended, since the theorem says that there will always be a particular (non-uniform) problem over such an extension which is **NP**-hard; here the hard problem depends on the given extension. In contrast, Theorem 14 asserts the existence of a particular (non-uniform) problem which becomes hard no matter how it is extended. The results given here extend, in the case of finite domains, the results of [18], which considers totally ordered domains. See also [20].

6. Applications in program analysis

One important application of the constraint solving technology studied in this paper is constraint-based program analysis over finite domains. Constraints are extracted from a program which typically describe run-time properties of the program. Solving the constraints is the main part of the analysis.

Several such analyses fall within the framework studied here. In many applications in program analysis the semilattice L will be thought of as a domain of abstract program properties with lower elements representing more information than higher elements. [8] If an analysis can be implemented as a constraint problem with minimal solutions it will have the desirable property that it is guaranteed to yield a uniquely determined piece of information which is optimal relative to the abstraction of the analysis. Theorem 6 says that *all and only* inequality constraint problems with this natural property can be represented as definite inequalities and hence can be solved in linear time using algorithm D.

In the remainder of this section, we give some examples of how our results can be used to reason about concrete program analysis problems. We consider set constraint based flow analysis and usage count analyses. Hopefully, it should be possible to see how the principles used in these examples could work in other contexts also.

6.1. Set constraints and flow analysis

Flow analysis (or closure analysis) is an important, pervasive program analysis, which can be presented as a constraint satisfaction problem over a finite domain. Consider set constraint based flow analysis, as defined by Palsberg and O'Keefe [24]. Here, the domain ranged over by constraint variables is the power set lattice over a finite set

[8] Alternatively, elements with more information may sit higher in the semilattice. Our results still apply, since the whole development in this paper can of course be dualized in the lattice-theoretic sense to encompass join-semilattices and join-closed problems over such.

of *labels*, where each label typically designates an occurrence of some object in the program. Suppose $L=\{c_1,\dots,c_n\}$ is the set of labels, then the lattice domain is the set $\wp(L)$ of subsets of L. Constraints are of the following forms:

1. $c_i \subseteq X$
2. $X \subseteq Y$
3. $c_i \subseteq X \Rightarrow Y \subseteq Z$

Here, c_i denotes the singleton $\{c_i\}$, and X, Y, Z are constraint variables ranging over subsets of L. The formal inequality ($\subseteq$) is interpreted as set inclusion (also denoted $\subseteq$). The constraints shown in the third class above are called *conditional constraints*; a valuation ρ assigning subsets of L to variables satisfies a conditional constraint $c_i \subseteq X \Rightarrow Y \subseteq Z$ if and only if $c_i \in \rho(X)$ implies $\rho(Y) \subseteq \rho(Z)$.

Tractability of flow analysis. Using the framework developed in the present paper we can easily verify that constraint-based flow analysis is indeed tractable. To do so, we present the problem relationally, by introducing a binary relation $R^{\subseteq}$ such that $R^{\subseteq}(X,Y)$ holds if and only if $X \subseteq Y$ and n ternary relations $R_i^{\Rightarrow}$, one for each constant c_i, such that $R_i^{\Rightarrow}(X,Y,Z)$ holds if and only if $c_i \subseteq X \Rightarrow Y \subseteq Z$. It is easy to verify that these relations are all meet-closed: the relation $R^{\subseteq}$ is meet-closed, because intersection is monotone with respect to set inclusion; and for the $R_i^{\Rightarrow}$, suppose that $c_i \subseteq X_1 \Rightarrow Y_1 \subseteq Z_1$ and $c_i \subseteq X_2 \Rightarrow Y_2 \subseteq Z_2$; we must show that

$$c_i \cap c_i \subseteq X_1 \cap X_2 \Rightarrow Y_1 \cap Y_2 \subseteq Z_1 \cap Z_2$$

But $c_i \subseteq X_1 \cap X_2$ implies $Y_1 \subseteq Z_1$ and $Y_2 \subseteq Z_2$ by the assumed conditionals, and therefore we have $Y_1 \cap Y_2 \subseteq Z_1 \cap Z_2$, thereby proving meet-closure of the relations $R_i^{\Rightarrow}$. By Theorem 6 we conclude that set constraint-based flow analysis can be represented in definite form and it can be solved by algorithm D.

Efficient flow analysis. It is instructive to consider the efficiency of the flow analysis obtained so far in the present framework. If we stick to the easy, abstract consideration above, which establishes tractability of the analysis, we will get an algorithm which runs in linear time in the size of the constraint set, *provided* the lattice $\wp(L)$ is *fixed* and the evaluation of a simple inequality $\tau \subseteq \tau'$ is a constant time operation. However, the conditions mentioned here are typically *not* satisfied in flow analysis. Firstly, the lattice is not fixed, because the size of the label set depends linearly on the size of the program being analysed (and program size is the interesting complexity parameter for flow analysis). Thus, the height of the lattice grows linearly, and for this reason our algorithm uses at least time n^2 (n is program size). Secondly, it is no longer fair to regard the evaluation of simple inequalities (subset inclusions) as constant time operations; they, too, grow linearly with the size of L. We must now conclude that the algorithm is at least $O(n^3)$. Finally, the size of the constraint set extracted from a program in the flow analysis under consideration is in fact quadratic (in program size) in the worst case. We are left with an algorithm running in worst-case time $O(n^4)$. This is in contrast to the well-known result (see [24] for references) that constraint-based flow analysis can be done in cubic time.

Our observations above raise the question whether the cubic time result for flow analysis can be understood in the present framework. It turns out that it can, modulo the following representation trick. Suppose that we represent subsets of L as bitvectors of length n (the size of L). We can then give a faithful representation of flow constraints over $\wp(L)$ as definite inequalities over **2**, the two point lattice of bits $\{0,1\}$, with order $0 \leqslant 1$, as follows. To each set variable X in the flow constraint system we introduce n variables X_i ranging over **2**, with the intention that $X_i = 1$ if and only if $c_i \in X$. The representation of individual flow constraints is:

1. For each constraint $c_i \subseteq X$, use the constraint $X_i = 1$.
2. For each constraint $X \subseteq Y$, use the constraints $X_i = Y_i$ for $i = 1 \ldots n$.
3. For each constraint $c_i \subseteq X \Rightarrow Y \subseteq Z$, use the constraints $X_i \sqcap Y_j \leqslant Z_j$ for $j = 1 \ldots n$.

Here the operation $\sqcap$ is the greatest lower bound in **2**. The constraints over **2** can be read as propositional Horn-clauses (recall Example 2), where $X_i \leqslant Y_i$ represents the proposition $c_i \in X \Rightarrow c_i \in Y$, and the clause $X_i \sqcap Y_j \leqslant Z_j$ represents the proposition $c_i \in X \wedge c_j \in Y \Rightarrow c_j \in Z$.

If n is the size of L (which is of the order of program size), we see that a flow constraint set C gets translated into a set over **2** of size at most $n \cdot |C|$. Since, as we have previously noted, the size of the flow constraint set C is of the order n^2, in the worst case, the flow analysis problem gets translated into a set of Horn-constraints of size $O(n^3)$ (worst case). Since Horn-constraints are solved in linear time by algorithm D, we get back cubic time flow analysis, thereby explaining it within our general framework.

From this example one may conclude that, for some representable (meet-closed) problems, one has to think in order to arrive at an efficient representation in definite form. On the other hand, we see that the present framework may lead one to discover a good algorithm by a rather natural representation shift (in this case, moving from the lattice $\wp(L)$ to the lattice **2** via a bitvector representation), while the underlying constraint solver remains the same.

6.2. *Usage count analyses*

We turn to some examples taken from usage count analyses.

Example 5. The *usage count analysis* from [23] has annotations in the lattice $\mathbf{3} = \{0, 1, \infty\}$ with $0 \leqslant 1 \leqslant \infty$. The constraints use the following binary operators:

$k_1 + k_2$

$k_1 \backslash k_2$	0	1	∞
0	0	1	∞
1	1	∞	∞
∞	∞	∞	∞

$k_1 \cdot k_2$

$k_1 \backslash k_2$	0	1	∞
0	0	0	0
1	0	1	∞
∞	0	∞	∞

$k_1 \rhd k_2$

$k_1 \backslash k_2$	0	1	∞
0	0	0	0
1	0	1	∞
∞	0	1	∞

Table 1

Count	Translation	Constraint	Translation
0	00	$k_1 = k_2$	$l_1 = l_2,\ r_1 = r_2$
1	01	$1 \leqslant k$	$r = 1$
∞	11	$k_1 \leqslant k_2$	$l_1 \leqslant l_2,\ r_1 \leqslant r_2$
		$k_1 + k_2 \leqslant k_3$	$1_1 \leqslant l_3,\ r_1 \leqslant r_3,\ l_2 \leqslant l_3,\ r_2 \leqslant r_3,\ r_1 \wedge r_2 \leqslant l_3$
		$\infty \cdot k_1 \leqslant k_2$	$r_1 \leqslant l_2$
		$k_1 \cdot k_2 \leqslant k_3$	$r_1 \wedge r_2 \leqslant r_3,\ l_1 \wedge r_2 \leqslant l_3,\ l_2 \wedge r_1 \leqslant l_3$
		$k_1 \rhd k_2 \leqslant k_3$	$r_1 \wedge l_2 \leqslant l_3,\ r_1 \wedge r_2 \leqslant r_3$

$+$ and $\cdot$ are addition and multiplication over counts. $k_1 \rhd k_2$ is 0 if $k_1 = 0$, otherwise $k_1 \rhd k_2 = k_2$.

The constraints are of one of the forms $k_1 = k_2$, $1 \leqslant k$, $k_1 \leqslant k_2$, $k_1 + k_2 \leqslant k_3$, $\infty \cdot k_1 \leqslant k_2$, $k_1 \cdot k_2 \leqslant k_3$ or $k_1 \rhd k_2 \leqslant k_3$. Hence (noting that $k_1 = k_2$ is equivalent to $k_1 \leqslant k_2,\ k_2 \leqslant k_1$), they are of the kind that can be solved by the methods shown, either by the direct method or by translation into boolean constraints.

To illustrate the translation, we can use the mapping shown below for lattice elements. We replace each constraint variable k_i by two variables l_i and r_i over the binary domain, with the constraint $l_i \leqslant r_i$. The constraints are translated using the translation shown in Section 4.2. Table 1 shows the result after reduction has been made for the constraints that involve constants.

Example 6. The distributive lattice **3** (see previous example) occurs in many program analyses involving usage counting, e.g., [23, 28, 3].

Let *pred* denote the predecessor function on **3**, with $pred(0) = 0$, $pred(1) = 0$, $pred(\infty) = 1$. Then *pred* is a distributive function. Consider the problem $\Phi = (\mathbf{3}, \{\sqcap, pred\})$, it is clearly meet-closed (viewed relationally), and, moreover, it is maximal. To see this, first note that the join-irreducible elements are $\{1, \infty\}$. Now define the function f_1 by setting $f_1(x) = 1 \sqcap x$ and define the function f_∞ by setting $f_\infty(x) = pred(x)$. Then $f_1(x) = 1$ if $1 \leqslant x$ and $f_1(x) = 0$ otherwise; moreover, $f_\infty(x) = 1$ if $\infty \leqslant x$ and $f_\infty(x) = 0$ otherwise. Recalling Example 3 we see that the functions f_1 and f_∞ are sufficient to represent the projection relations in $\Pi_{\mathbf{3}}$, since we can express the condition $c_i \leqslant x \Rightarrow c_j \leqslant y$, for $c_i, c_j \in \mathbf{Irr}(\mathbf{3})$, by the distributive constraint $f_{c_i}(x) \leqslant f_{c_j}(y)$. It then follows from Theorem 14 that Φ is a maximal problem. Contrast this with Example 7 below.

Example 7. *Multiplicity inference* approximates the number of store operations performed on *regions* in order to enhance the *region inference*-based memory management of [3], where an efficient equational (unification based) multiplicity inference algorithm is described. Attempts (made by Tofte and the first author of the present paper) at generalizing multiplicity inference by using non-equational constraints of the form considered in the present paper all turned out to require the simultaneous use of *multiplication* and *subtraction* of multiplicities, which are elements of the lattice **3** of

Example 6. Here multiplication ($x \cdot y$) is defined as in Example 5), and subtraction, $x - y$, is given by setting $0 - x = 0$, $\infty - 0 = \infty - 1 = \infty$, $x - x = 0$. Moreover, it appeared to be necessary to generate constraints which may contain arbitrary subtractions on the right-hand side of inequalities, thus leading to a *non-distributive* problem (in the sense of Section 4.1). Unfortunately, this mixture of operators yields an intractable satisfiability problem, which can be seen as follows: subtraction is not distributive and $\cdot$ restricted to the two-point chain $\{0, 1\}$ equals the greatest lower bound operation on that chain; this shows that the problem $(\mathbf{3}, \{\cdot, -\})$ corresponds to a relational problem containing a non-meet closed extension of a maximal problem, viz., that of Example 4.

7. Implementations

We have tested the constraint solution algorithm D described in Appendix A on two different lattices: the 3-point lattice used in Example 5 and a 5-point lattice of height 4. For the 3-point lattice we have used five binary operators: the three shown in Example 5 plus meet and join. For the 5-point lattice we have used three binary operators: meet, join and an irregular (but still monotone) operator.

For the 3-point lattice we have also tested the method that translates the constraints to boolean constraints and solve these by a Horn-clause solver. We have used a variant of the first algorithm from [10] to solve the Horn clauses.

We have implemented the algorithms in the C programming language and tested it on a number of randomly generated constraint sets. While it is well known that randomly generated test sets rarely show the worst-case asymptotic complexity of algorithms, this will not matter for these tests as the asymptotic worst-case complexity is the same as the asymptotic best case, i.e. linear.

With randomly generated constraints, the number of variables relative to the total number of constraints will affect how constrained each variable is. If the number of constraints is low compared to the number of variables, most variables will be unconstrained, and if the number of constraints is high, most variables will be constrained to the top element of the lattice (we generate only constraints with variables on the right-hand side). We have run tests where the number of variables is equal to the number of constraints (which means most are unconstrained), where the number of variables is 25 of constraints (which sends most to the top element) and one with 50 variables, which spreads the solution roughly evenly across the lattice.

In all cases we have tested four different constraint sets of each size and averaged the running times. The programs are run on a HP9000/735 and timed using the 'time' command. The compiler is the HP C compiler, using standard optimization (`-O`).

The results for the 3-point lattice using the algorithm in Appendix A is shown in Fig. 1. Timings for the same constraint sets when translating to Horn-clauses is shown in Fig. 2. Timings for the 5-point lattice is shown in Fig. 3.

Constraints	Variables	Time
10000	2500	0.09s
10000	5000	0.08s
10000	10000	0.07s
100000	25000	0.89s
100000	50000	0.86s
100000	100000	0.60s
1000000	250000	11.94s
1000000	500000	11.31s
1000000	1000000	7.26s

Fig. 1. Running times for algorithm D on 3-point lattice.

Constraints	Variables	Time
10000	2500	0.11s
10000	5000	0.11s
10000	10000	0.09s
100000	25000	1.18s
100000	50000	1.00s
100000	100000	0.61s
1000000	250000	16.86s
1000000	500000	13.36s
1000000	1000000	7.12s

Fig. 2. Running times for algorithm D on 5-point lattice.

Constraints	Variables	Time
10000	2500	0.08s
10000	5000	0.07s
10000	10000	0.06s
100000	25000	1.13s
100000	50000	1.04s
100000	100000	0.57s
1000000	250000	13.96s
1000000	500000	12.53s
1000000	1000000	6.26s

Fig. 3. Running times for 3-point lattice when translating to Horn clauses.

The results show a lightly more than linear increase in running time, but this can be explained by an increase in cache misses when the data sets grow. For the constraints used in the test, the methods solve roughly 100 000 constraints per second. This is fast enough that the problem is not time, but space. Roughly 30 bytes are used per constraint in the current implementation, so solving a million constraints can be done in 11 s, but it uses about 30 MB of memory, most which is randomly accessed. This makes the wall-clock time increase dramatically when the data set size exceeds the available physical memory, as swapping dominates. While constraints over the 3-point lattice show little difference in time when solved by translation to Horn-clauses, the latter uses roughly twice as much memory, which makes it less practical for large data sets. This will be even more of a problem when larger lattices or higher-arity operators are used.

The space problem can to an extent be reduced by using incremental versions of the algorithm. It is fairly easy to modify the algorithm to keep a minimal solution which is incrementally updated as more constraints are added. If a variable reaches the top of the lattice, all constraints with the variable on the right-hand side can be removed from the constraint set, as no monotonic changes can cause these to be unsatisfied. The worst-case space cost is, however, unchanged, as variables may never reach the top element of the lattice.

8. Conclusion

We have studied efficient solution methods for the following classes of constraint problems over finite meet-semilattices:
– Definite inequalities involving monotone functions.
– Distributive inequalities.
– Boolean inequalities (Horn clauses).
– Meet-closed relational constraints.
We have shown that these classes are equivalent modulo linear time transformations for any fixed problem, and we have estimated the constant factors involved. For any fixed lattice and any fixed set of function/relation symbols the methods are linear time in the size of the constraint set, but, in the case of boolean representation, the time depends non-linearly on the maximal arity of relation symbols used, and memory consumption may become rather large. Boolean representation should therefore be used only when arities of functions involved are known to be small.

We have characterized the expressiveness of the framework of definite constraints, and we have shown that this framework captures a class of problems which is maximal, in the sense that adding any non-meet-closed relation to any non-trivial problem yields **NP**-hard uniform problems.

Acknowledgements

We wish to thank Fritz Henglein, Neil Jones, Helmut Seidl, Mads Tofte and our referees for helpful discussions and comments.

Appendix A. Algorithm

We assume the following structures and operations, together with some invariant properties these items are required to satisfy:

(1) A list *Ilist* of records representing the inequalities in C, the given definite constraint set; there is one record for each inequality in C. Each record holds a pointer to an inequality record in addition to the inequality itself. Each record also holds a boolean variable, called *inserted*. The intention is that the records in *Ilist* can be inserted into a structure called *NS* (see below), and the field *inserted* must be set to true, if the inequality represented by the record is inserted in *NS*; it must be set to false, if the record is not in *NS*. The operations POP, INSERT and DROP (for which see below) will update the *inserted* field.

(2) An array $Clist[\beta]$ indexed by variables in $Var(C)$. Each entry $Clist[\beta]$ holds an array of pointers to inequalitites in *Ilist*, one entry for each inequality in C in which β occurs. Thus, to each item in $Clist[\beta]$ there corresponds a unique occurrence of β in C. Hence, the number of distinct items $Clist[\beta][k]$ is bounded by $|C|$. Moreover, for each inequality in C_{var} there is at least one entry in *Clist* pointing to it.

(3) The structure *NS* is a doubly linked list of pointers to inequalitites in *Ilist*. Pointers in *NS* point *only* to inequalities in *Ilist* which are among the inequalities in C_{var}, i.e., only those inequalities in *Ilist* which have a variable on the right-hand side can be pointed to by an element in *NS*. Each inequality in *Ilist* is represented by at most one item in *NS*, but it may be the case that items in *Ilist* are not represented in *NS*. Each pointer in *NS* points to the item in *Ilist* which it represents, and that item in *Ilist* has its pointer set to the representing item in *NS*. There is a pointer to the head element of *NS*. The idea is that *NS* holds pointers to those inequalities in C_{var} which are *not satisfied* under the current interpretation ρ (see below.)

(4) A finite map ρ mapping each distinct variable of C to an element in L. The map ρ holds the current "guess" at a satisfying valuation for C_{var}. The map $\bot_{Var(C)\to L}$ is the map which sends every variable of C to the bottom element of L. The map ρ is implemented as an array of lattice elements indexed by the variables. We write $\rho(\beta)$ for $\rho[\beta]$. Evaluating $\rho(\beta)$ is a constant time operation, and so is the operation of updating ρ at β, meaning that $\rho(\beta)$ is changed into some new constant of L.

(5) An operation POP which removes the head element of *NS* and returns it (i.e., POP returns a pointer to an inequality in *Ilist*.) Operation POP sets the *inserted* field to false in the inequality record in *Ilist* represented by the popped element of *NS*. This is a constant time operation (since there is a pointer to the head element of *NS* which is maintained by POP).

(6) An operation INSERT(i) which inserts an inequality pointer i at the front of *NS*. Here i is an inequality pointer in *Clist*, and it is assumed that INSERT first checks to see if the flag *inserted* associated with the *Ilist*-element pointed to by i is set to true. If this is the case, then insert does nothing. If the flag is set to false, then INSERT sets the flag *inserted* to true, and then it inserts the pointer in the front of the list *NS*, updating the head pointer of *NS*. This is a constant time operation.

```
1. Input
   A finite set $C = \{\tau_i \le A_i\}_{i \in I}$ of definite inequalities over $\Phi = (L, F)$, $F =$
   $\{f : L^{a_f} \to L\}$ with each $f$ monotone, $L$ finite lattice, $Var(C) = \mathcal{V}_m$ for some
   $m$.
2. Initially
   $C :=$ L-NORMALIZE$(C)$
   $\rho := \bot_{\mathcal{V}_m \to L}$
   $C_{var} := \{\tau \le A \in C \mid A$ is a variable$\}$
   Initialize the lists $Clist[\alpha]$, for every distinct variable $\alpha$ in $C$.
   Initialize $Ilist$ to hold the inequalities in $C$.
   $NS := \{\tau \le \beta \in C_{var} \mid \rho, L \not\models \tau \le \beta\}$
3. Iteration
   while $NS \ne \emptyset$ do
           $\tau \le \beta :=$ POP$(NS)$;

           $\rho(\beta) := [\![\tau]\!]\rho \sqcup \rho(\beta)$;

           for $\sigma \le \gamma \in Clist[\beta]$ do
                  if $\rho, L \not\models \sigma \le \gamma$
                  then INSERT$(\sigma \le \gamma)$
                  else DROP$(\sigma \le \gamma)$
           end; (* for *)

     end;(* while *)
4. Output
   If $\rho, L \models \tau \le c$ for all $\tau \le c \in C_{cnst}$ then output $\rho$ else FAIL.
```

Fig. 4. Algorithm D for satisfiability of definite constraints over Φ.

(7) An operation DROP(i), where i is a pointer to an element of *NS*. Here i is an inequality pointer in *Clist*, and DROP first checks to see if the flag *inserted* associated with the inequality in *Ilist* pointed to by i is set to true; if this is the case, then DROP removes the pointer from *NS*, and if the flag *inserted* is set to false, then DROP does nothing. This is a constant time operation (since *NS* is a doubly linked list.)

(8) An operation L-NORMALIZE which transforms a definite set into L-normal form. This operation can be assumed to run in time $O(|C|)$, and the size of L-NORMALIZE(C) is $O(|C|)$, by Lemma 1. Moreover, solving L-NORMALIZE(C) is equivalent to solving C, by Lemma 2.

In the algorithm we have followed the convention of writing the pattern of an inequality pointed to by an inequality pointer instead of the pointer itself. Thus, the operation $\tau \leqslant \beta :=$ POP(NS) is a shorthand for popping a pointer to the inequality $\tau \leqslant \beta$ from *NS* and loading that pointer into a temporary variable while at the same time pulling the inequality (pointed to by the pointer) apart by pattern matching. This all comes down to leaving a few indirections and a simple form of analysis of inequalities implicit, in passing from elements in *Clist* t elements in *Ilist* and in *NS*. Also, in the **for**-loop the pattern $\sigma \leqslant \gamma$ is really a pointer to an element of *Ilist* containing the inequality $\sigma \leqslant \gamma$. Likewise, the set notation for the initialization of *NS* really means that *NS* is initialized to a doubly linked list of pointers to the inequalities of C satisfying the condition of the set.

References

[1] T. Amtoft, Local type reconstruction by means of symbolic fixed point iteration, in: Proc. 5th European Symp. on Programming (ESOP), Edinburgh, Scotland, Lecture Notes in Computer Science, Vol. 788, Springer, Berlin, April 1994, pp. 43–57.

[2] C. Beeri, P. Bernstein, Computational problems related to the design of normal form relation schemes, ACM Trans. Database Systems (TODS) 4 (1) (1979) 30–59.

[3] L. Birkedal, M. Tofte, M. Vejlstrup, From region inference to von Neumann machines via region representation inference, in: Proc. 23rd Annual ACM Symp. on Principles of Programming Languages (POPL), ACM Press, New York, January 1996, pp. 171–183.

[4] L. Birkedal, M. Welinder. Binding-time analysis for Standard ML, Lisp and Symbolic Comput. 8 (3) (1995) 191–208.

[5] A. Bondorf, J. Jørgensen, Efficient analyses for realistic off-line partial evaluation, J. Funct. Programming 3 (3) (1993) 315–346.

[6] C.C. Chang, H.J. Keisler, Model Theory, Studies in Logic and the Foundation of Mathematics, 3rd Edition, Vol. 73, North-Holland, Amsterdam, 1990.

[7] N. Creignou, A dichotomy theorem for maximum generalized satisfiability problems, J. Comput. System Sci. 51 (3) (1995) 511–522.

[8] B.A. Davey, H.A. Priestley, Introduction to Lattices and Order, Cambridge Mathematical Textbooks, Cambridge University Press, Cambridge, MA, 1990.

[9] R. Dechter, J. Pearl, Structure identification in relational data, Artificial Intell. 58 (1992) 237–270.

[10] William F. Dowling, Jean H. Gallier, Linear-time algorithms for testing the satisfiability of propositional horn formulae, J. Logic Programming 3 (1984) 267–284.

[11] D. Dussart, F. Henglein, C. Mossin, Polymorphic recursion and subtype qualifications: polymorphic binding-time analysis in polynomial time, in: Proc. 2nd Internat. Static Analysis Symp. (SAS), Glasgow, Scotland, Lecture Notes in Computer Science. Springer, Berlin, September 1995.

[12] T. Feder, M.Y. Vardi, Monotone monadic SNP and constraint satisfaction, in: 25th Annual Symp. on the Theory of Computing (STOC), ACM, New York, 1993, pp. 612–622.

[13] M. Garey, D. Johnson, Computers and Intractability – A Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

[14] R. Greenlaw, H.J. Hoover, W. Ruzzo, Limits to Parallel Computation. P-Completeness Theory, Oxford University Press, Oxford, 1995.

[15] F. Henglein, Efficient type inference for higher-order binding-time analysis, in: J. Hughes (Ed.), Functional Programming Languages and Computer Architecture, Cambridge, Massachusetts, August 1991, Lecture Notes in Computer Science, Vol. 523, ACM, Springer, Berlin, 1991, pp. 448–472.

[16] F. Henglein, Iterative fixed point computation for type-based strictness analysis, in: Baudouin Le Charlier (Ed.), 1st Internat. Static Analysis Symp. (SAS), Namur, Belgium, Lecture Notes in Computer Science, Vol. 864, Springer, Berlin, September 1994, pp. 395–407. Also DIKU Semantics Report D-192.

[17] W. Hodges, Model Theory. Encyclopedia of Mathematics and its Applications, Vol. 42, Cambridge University Press, Cambridge, 1993.

[18] P. Jeavons, M. Cooper, Tractable constraints on ordered domains, Artificial Intell. 79 (1995) 327–339.

[19] P. Jeavons, D. Cohen, An algebraic characterization of tractable constraints, in: First Annual Conf. on Computing and Combinatorics (COCOON), Lecture Notes in Computer Sciences, Vol. 959, Springer, Berlin, 1995, pp. 633–642.

[20] P. Jeavons, D. Cohen, M. Gyssens, A unifying framework for tractable constraints, in: 1st Internat. Conf. on Principles and Practice of Constraint Programming, Lecture Notes in Computer Sciences, Vol. 976, Springer, Berlin, 1995, pp. 276–291.

[21] S. Khanna, M. Sudan, D.P. Williamson, A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction, in: Proc. 29th Annual ACM Symp. on Theory of Computing (STOC), ACM, New York, May 1997.

[22] G. Kildall, A unified approach to global program optimization, in: Proc. ACM Symp. on Principles of Programming Languages (POPL), 1973.

[23] T.Æ. Mogensen, Types for 0, 1 or many uses, in: Proc. 9th Internat. Workshop on Implementation of Functional Languages, St. Andrews, Scotland, September 1997.

[24] J. Palsberg, P. O'Keefe, A type system equivalent to flow analysis, ACM Trans. Programming Languages Systems 17 (4) (1995) 576–599.

[25] V. Pratt, J. Tiuryn, Satisfiability of inequalities in a poset, Fund. Inform. 28 (1996) 165–182.

[26] T.J. Schaefer, The complexity of satisfiability problems, in: 10th Annual Symp. on the Theory of Computing (STOC), ACM, New York, 1978, pp. 216–226.

[27] K.L. Solbjerg, Annotated type systems for program analysis, Ph.D. Thesis, DAIMI, Department of Computer Science, University of Aarhus, Denmark, November 1995.

[28] M. Tofte, J.-P. Talpin, Implementation of the typed call-by-value λ-calculus using a stack of regions, in: Proc. 21st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL), Portland, Oregon, ACM, ACM Press, New York, January 1994.