Entailment of Set Constraints is not Feasible

Martin Müller Joachim Niehren

Programming Systems Lab,
Universität des Saarlandes
66041 Saarbrücken, Germany,
{mmueller,niehren}@ps.uni-sb.de

Abstract. Set constraints are inclusions between expressions denoting sets of trees. They have been used extensively for type inference and program analysis. At the lower end of the expressiveness scale there are atomic set constraints and Ines constraints (inclusions over non-empty sets) for both of which a cubic time satisfiability test is known. Recently, there has been increasing interest in entailment tests for set constraints. We prove that the entailment problem of atomic set constraints is coNP-hard. We also show that the entailment problem of Ines constraints is coNP-hard. This corrects a claim of polynomial complexity presented at CP'96. Our results suggest that a complete entailment test is not feasible even for simple classes of set constraints.

Keywords: atomic set constraints, Ines constraints, entailment, complexity.

1 Introduction

Set constraints are inclusions between expressions denoting sets of trees. Such expressions are first-order terms with or without set operators. Set constraints have received much attention in constraint-based type inference and program analysis for different programming languages [32, 24, 28, 21, 5, 15, 19, 7, 8, 2, 1]. Other applications of set constraints include order-sorted unification [37] and constraint logic programming [25].

The complexity of the satisfiability problem for various classes of set constraints has been widely studied [17, 20, 10, 3, 6, 18, 4, 11, 12, 3, 35, 29] and was often found to be very high (*e.g.*, NEXPTIME-complete [35, 3] and DEXPTIME-complete [14]). At the lower end of the expressiveness scale there are atomic set constraints [20] and Ines constraint (inclusions over non-empty set) [29]. These two classes of set constraints allow for an efficient satisfiability test.

Recently, there has been increasing interest in entailment tests for set constraints. Entailment algorithms are useful for constraint simplification (*e.g.*, see [27]; but also [36, 30, 9] for entailment over a domain of trees called "types"). Also the satisfiability problem of set constraints with negation has been considered, which generalizes the entailment problem of atomic set constraints. In program analysis for instance, negation can be used for the analysis of conditional else-branches.

An *atomic set constraint* ψ is a conjunction of inclusions $t_1 \subseteq t_2$ between first-order terms t_1 and t_2 without set operators, and interpreted over arbitrary sets of trees. Atomic set constraints are important, since they are contained in most classes of set constraints. The satisfiability problem of atomic set constraints can be solved in cubic time (see the complete version of [29] for instance). The satisfiability problem of atomic set constraints with negation is decidable and in DEXPTIME [14]. Hardness results for entailment of atomic set constraints have not been presented so far.

We prove that the entailment problem $\psi \models \psi'$ of atomic set constraints is coNP-hard. This result implies that the entailment problem of atomic set constraints with negation is coNP-hard. This in contrast to the claim of polynomial complexity in [14]. For our proof of coNP-hardness, we adapt a recent idea by Henglein and Rehof [22]. They have considered entailment of ordering constraints ("structural subtyping") over a domain of finite trees ("simple types") and show it coNP-complete. We transfer their idea to a class of set constraints.

Ines constraints have been introduced in [29] motivated by type analysis for Oz [31, 34], a concurrent constraint programming language [26, 33]. Syntactically, *Ines constraints* coincide with atomic set constraints, but semantically they are interpreted over non-empty sets of trees (rather than over arbitrary sets). Ines constraints have also attracted some interest because they are supposed to have the technically pleasing independence property [13].

The given complexity results for atomic set constraints do also apply for Ines constraints. The satisfiability problem of Ines constraints can be solved in cubic time [29]. The satisfiability problem of Ines constraints with negation is decidable and in NEXPTIME. We prove that the entailment problem of Ines constraints is coNP-hard. This is in contrast to a polynomial result presented at CP'96 [13]. The flaw in this paper is due to an incomplete entailment algorithm (see below for an example). Note also that the proof of the independence property of Ines constraints given in [13] is not correct since it is based on the incomplete entailment algorithm. We do believe that the independence property holds for Ines constraints, but have not yet been able prove it.

The following two examples illustrate the entailment problem of Ines constraints. The entailment proposition (1) holds over Ines (but not over atomic set constraints).

$$x \subseteq f(x') \wedge x' \subseteq a \wedge x \subseteq f(y') \ \models\ x' \subseteq y' \tag{1}$$

A possible argument is as follows: Since the denotation of x is non-empty, the intersection of the denotations of x' and y' must be non-empty. The constraint $x' \subseteq a$ implies that x' denotes the singleton set $\{a\}$. Hence, the denotation of y' must at least contain a. Thus $x' \subseteq y'$ is entailed. By a similar argument, the entailment proposition (2) can be shown valid for Ines.

$$x \subseteq f(x', x'') \wedge x' \subseteq g(y', y'') \wedge y' \subseteq a \wedge x \subseteq f(z', z'') \wedge z' \subseteq g(u', u'') \models\ y' \subseteq u' \tag{2}$$

This example illustrates the incompleteness of the algorithm from [13]: there the entailment in (1) is detected while the entailment in (2) is not. Note that this failure depends on the usage of binary constructors in (2).

In the domain of constraint programming, the results of this paper imply that, without further restrictions, it is impractical to instantiate the CC-scheme with either Ines or atomic set constraints since the CC-scheme relies on an efficient entailment test [26, 33, 23]. In contrast, a corresponding CLP-language is well-conceivable since CLP only requires an efficient satisfiability test [23].

Plan of the Paper. Section 2 contains some technical preliminaries. Section 3 introduces atomic set constraints. Section 4 shows their entailment problem to be coNP-hard. Section 5 presents our complexity results for Ines constraints.

2 Preliminaries

We assume a set of *variables* ranged over by x, y, z and a signature Σ that defines a set of *function symbols* f, g and their respective arity $n \geq 0$. *Constants*, *i.e.*, function symbols of arity 0, are denoted with a and b. We assume that Σ contains at least one binary function symbol and one constant.

A *first-order term* t is a variable or an expression $f(t_1, \ldots, t_n)$ where n is the arity of f. A (finite) *tree* τ is a ground first-order term, *i.e.*, a first-order term without variables. The set of all finite trees over Σ is denoted by Fin. The powerset of Fin is denoted by $\mathsf{P}(\text{Fin})$ and the set of all non-empty subsets of Fin by $\mathsf{P}^+(\text{Fin})$. We freely consider $\mathsf{P}(\text{Fin})$ and $\mathsf{P}^+(\text{Fin})$ as model theoretic structures where a function symbol f of Σ is interpreted as elementwise tree constructor and the relation symbol $\subseteq$ as the subset relation.

We use the following predicate logic notation. Assume a class L of first-order logic formulae φ and a model theoretic structure $\mathcal{A}$ to interpret them. The set of *free variables* in φ is denoted with $\mathsf{V}(\varphi)$ and the set of function symbols used in φ with $\mathsf{C}(\varphi)$. A *solution* of φ is a variable assignment into the domain of $\mathcal{A}$ which makes φ true. A formula φ is *satisfiable* if there exists a solution of φ. A formula φ is *valid* if all variable assignments into the domain of $\mathcal{A}$ are solutions of φ. We say that φ entails φ' if the implication $\varphi \rightarrow \varphi'$ is valid, and write $\varphi \models \varphi'$. The *satisfiability problem of* L is whether a formula $\varphi \in L$ is satisfiable. The *entailment problem of* L, whether an entailment $\varphi \models \varphi'$ holds for formulae $\varphi, \varphi' \in L$. The *satisfiability problem of* L *with negation* is whether a conjunction $\bigwedge_{i=1}^{n} \varphi_i \wedge \bigwedge_{j=1}^{m} \neg\varphi'_j$ of positive and negative formula in L is satisfiable. It is well known that the entailment problem of L is less general than the satisfiability problem of L with negation, because $\varphi \models \varphi'$ if and only if $\varphi \wedge \neg\varphi'$ is unsatisfiable.

3 Atomic Set Constraints

An *inclusion constraint* without set operators ψ is a conjunction of inclusions between first-order terms:

$$\psi \ ::= \ t_1 \subseteq t'_1 \ \mid \ \psi \wedge \psi'$$

An *atomic set constraint* [20] is an inclusion constraint interpreted over sets of trees. As usual, we choose finite trees, i.e. we interpret in the structure $\mathsf{P}(\mathrm{Fin})$. It is easy to see that the hardness result also holds for the alternative interpretation over sets of possibly infinite trees.

Proposition 1. *There exists an algorithm that solves the satisfiability problem of atomic set constraints in cubic time.*

This result is well known. An algorithm and a complete proof can be found in the full version of [29].

Theorem 2. *The entailment problem of atomic set constraints is coNP-hard.*

Proof. The coNP-hardness of the entailment problem follows from Proposition 4 in Section 4. □

Corollary 3. *The satisfiability problem of atomic set constraints with negation is coNP-hard.*

Proof. Follows from Theorem 2 since the entailment problem is less general than the satisfiability problem with negation as argued above in Section 2.

Note that Corollary 3 is in contrast to the claim of [14] that atomic set constraints with negation have a polynomial satisfiability problem.

4 Entailment is coNP-Hard

We prove that the entailment problem of atomic set constraints is coNP-hard as claimed in Theorem 2. Our proof is based on a reduction of the problem SAT (propositional satisfiability [16]) whether a set of propositional clauses is non-satisfiable. The reduction elaborates on an idea by Henglein and Rehof in [22].

We assume an infinite set of boolean variables ranged over by u. A *clause* C a finite disjunction of literals u or $\neg u$. A propositional formula Γ is a conjunction of clauses $\bigwedge_{i=1}^{n} C_i$. A solution of Γ is a function β that assigns boolean values to boolean variables such that Γ evaluates under β to $\mathtt{true}$. The *clause satisfiability problem (SAT)* is whether a given formula Γ has a solution.

Proposition 4. *For every propositional formula Γ and distinct variables x and y there exist existential formulas $\Psi(\Gamma, x)$ and $\hat{\Psi}(y)$ over atomic set constraints whose size is linear in the size of Γ and such that*

$$\Psi(\Gamma, x) \wedge \hat{\Psi}(y) \models x \subseteq y \quad \textit{iff} \quad C \textit{ is unsatisfiable.}$$

Before giving the proof, we note some needed preliminaries on paths and trees.

Trees and Paths. A *path* p or q is a sequence of positive integers ranged over by i, j, n, m. The *empty path* is denoted by ε, and the free-monoid concatenation of paths p and q by pq. We have $\varepsilon p{=}p\varepsilon{=}p$. Given paths p and q, q is called a *prefix of* p if $p = qp'$ for some path p'. A *token* is a pair $(p,\, f)$ where p is a path and f a function symbol. The set of all tokens is denoted with Tok and the powerset of tokens with $\mathsf{P}(\mathsf{Tok})$. Every tree τ corresponds to a unique set of tokens $\gamma(\tau)$ defined as follows:

$$\gamma(f(\tau_1, \ldots, \tau_n)) = \{(\varepsilon,\, f)\} \ \cup\ \{(ip,\, a) \mid (p,\, a) \in \gamma(\tau_i),\ 1 \leq i \leq n\}$$

Proof of Proposition 4. Assume that Γ contains the variables $u_1, \ldots, u_k$. The idea of [22] is to represent every boolean variable assignment β on $\{u_1, \ldots, u_k\}$ as a path p_β:

$$p_\beta = \overline{\beta(u_k)} \ldots \overline{\beta(u_1)} \qquad \text{where} \quad \overline{\mathsf{true}} = 1 \text{ and } \overline{\mathsf{false}} = 2.$$

A set of boolean variable assignment B can be represented as a set $\sigma(B)$ of trees that we define next. Let f be a binary function symbol and a a constant (a binary function symbol exists because of our assumptions on the signature). We say

$$\beta \ \mathit{in}\ \tau \quad \text{if} \quad (p_\beta,\, a) \in \gamma(\tau) \text{ and } (p',\, f) \in \gamma(\tau) \text{ for all proper prefixes } p' \text{ of } p_\beta$$

We encode a set B of boolean variable assignments as the following set of trees $\sigma(B)$:

$$\sigma(B) = \{\tau \mid \text{if } \beta \in B \text{ then } \beta \ \mathit{in}\ \tau\}$$

Let τ^k be the complete binary tree of depth k built from function symbols f and a. We now define the formulas $\Psi(\Gamma, x)$ and $\hat{\Psi}(y)$ such that

$$\alpha \text{ is a solution of } \Psi(\Gamma, x) \ \text{ iff } \ \alpha(x) \subseteq \sigma(Sol(\neg\Gamma)) \tag{3}$$
$$\alpha \text{ is a solution of } \hat{\Psi}(x) \ \text{ iff } \ \alpha(x) = \{\tau^k\} \tag{4}$$

where $Sol(\neg\Gamma) = \{\beta \mid \beta \text{ is a solution of } \neg\Gamma\}$). Before doing so, we verify that the above two properties justify Proposition 4:

$$\begin{array}{lll} \Psi(\Gamma, x) \wedge \hat{\Psi}(y) \models x{\subseteq}y & \text{iff} & \forall\alpha : \alpha \models \Psi(\Gamma, x) \wedge \hat{\Psi}(y)\text{: } \alpha(x) \subseteq \alpha(y) \\ & \text{iff} & \forall\alpha : \alpha \models \Psi(\Gamma, x) \wedge \hat{\Psi}(y)\text{: } \alpha(x) \subseteq \{\tau^k\} \quad (*) \\ & \text{iff} & \sigma(Sol(\neg\Gamma)) \subseteq \{\tau_k\} \\ & \text{iff} & \Gamma \text{ is unsatisfiable} \end{array}$$

For the equivalence marked $(*)$ note that by (3) in particular $\sigma(Sol(\neg\Gamma))$ is a solution of $\Psi(C, x)$ and $\hat{\Psi}(y)$. We define the existential formulae $\Psi(C, x)$ and $\hat{\Psi}(y)$ as follows. For every $1{\leq}i{\leq}k$ and every clause C with variables in $\{u_1, \ldots u_k\}$ let

$$\delta_i(C) = 1 \text{ if } \neg u_i \text{ in } C, \qquad \delta_i(C) = 2 \text{ if } u_i \text{ in } C, \qquad \delta_i(C) = 3 \text{ otherwise.}$$

Without loss of generality we assume that there exists no clause C and boolean variable u_i such that both u_i in C and $\neg u_i$ in C. Every clause C corresponds to a path $p(C)$ given by $p(C) = \delta_k(C)\ldots\delta_1(C)$. Let $\Gamma = \bigwedge_{i=1}^n C_i$. The formulae $\Psi(\Gamma, x)$ and $\hat{\Psi}(y)$ are defined by recursion over $p(C_i)$.

$$
\begin{array}{lcl}
\Psi(\Gamma, x) & = & \bigwedge_{i=1}^n \Psi'(p(C_i), x) \\
\quad \Psi'(\varepsilon, x) & = & x{=}a \\
\quad \Psi'(1p, x) & = & \exists x' \exists x'' (x{=}f(x', x'') \wedge \Psi'(p, x')) \\
\quad \Psi'(2p, x) & = & \exists x' \exists x'' (x{=}f(x', x'') \wedge \Psi'(p, x'')) \\
\quad \Psi'(3p, x) & = & \exists x' \exists x'' \exists y\, (x{=}f(x', x'') \wedge x' {\subseteq} y \wedge x'' {\subseteq} y \wedge \Psi'(p, y)) \\
\hat{\Psi}(y) & = & \hat{\Psi}'(3^k, y) \\
\quad \hat{\Psi}'(3p, x) & = & \exists y\, (x{=}f(y, y) \wedge \hat{\Psi}'(p, y))
\end{array}
$$

Here, 3^k stand for a path $3\ldots3$ of length k. Since every step of this definition introduces at most three new variables, these formulae have size linear in the number of variables k and clauses n. It is easy to verify that every solution α of $\hat{\Psi}(y)$ satisfies (4), *i.e.*, $\alpha(x) = \{\tau^k\}$. In Section A we show that $\Psi(\Gamma, x)$ satisfies (3). □

5 Ines Constraints

An *Ines constraint* [29] is an inclusion constraint interpreted over non-empty sets of trees, i.e. in the structure $\mathsf{P}^+(Fin)$.

Theorem 5. *The satisfiability problem of Ines constraints over* $\mathsf{P}^+(\mathrm{Fin})$ *can be decided in time* $O(n^3)$.

Proof. This has been shown in [29]. □

Theorem 6. *The entailment problem of Ines constraints is coNP-hard.*

Proof. The proof for atomic set constraints in Section 4 applies unchanged. □

This result in in contrast to a claim of polynomial complexity presented at CP'96 [13]. The flaw in this paper was the incompleteness of the entailment algorithm given (see the introduction for an example). As a consequence, also the proof that Ines constraints have the independence property must be considered incorrect since it is based on the incomplete entailment algorithm. The independence property claimed for Ines constraints is the following:

$$\text{if} \quad \psi \models \bigvee_{i=1} \psi_i \quad \text{then exists } i, 1 \leq i \leq n, \text{ such that} \quad \psi \models \psi_i$$

We believe that the independence property holds for Ines constraints but have not yet been able to prove it.

Corollary 7. *The satisfiability problem of Ines constraints with negation is coNP-hard.*

Acknowledgments This research reported here has been supported by the Esprit Working Group CCL II (EP 22457) and the SFB 378 at the Universität des Saarlandes.

References

1. A. Aiken and M. Fähndrich. Making Set-Constraint Program Analysis Scale. In *Proc. Set-Constraint Workshop at CP'96*, 1996.
2. A. Aiken and N. Heintze. Constraint-Based Program Analysis, 1995. Invited Lecture at the 22^{nd} ACM Symposium on Principles of Programming Languages.
3. A. Aiken, D. Kozen, M. Vardi, and E. Wimmers. The Complexity of Set Constraints. In *Conference on Computer Science Logic*, pp. 1–17, 1993.
4. A. Aiken, D. Kozen, and E. Wimmers. Decidability of Systems of Set Constraints with Negative Constraints. *Information and Computation*, 122(1):30–44, 1995.
5. A. Aiken and B. Murphy. Static Typing in a Dynamically Typed Language. In *18^{th} ACM Symposium on Principles of Programming Languages*, pp. 279–290, Jan. 1991.
6. A. Aiken and E. Wimmers. Solving Systems of Set Constraints. In *7^{th} IEEE Symposium on Logic in Computer Science*, pp. 329–340, June 1992.
7. A. Aiken and E. Wimmers. Type Inclusion Constraints and Type Inference. In *6^{th} Conf. on Functional Programming and Computer Architecture*, pp. 31–41, 1993.
8. A. Aiken, E. L. Wimmers, and T. Lakshman. Soft Typing with Conditional Types. In *21^{st} ACM Symposium on Principles of Programming Languages*, Jan. 1994.
9. A. Aiken, E. Wimmers, and J. Palsberg. Optimal Representation of Polymorphic Types with Subtyping. Tech. Rep. UCB/CSD-96-909, University of California, Berkeley, 1996.
10. L. Bachmair, H. Ganzinger, and U. Waldmann. Set Constraints are the Monadic Class. In *8^{th} IEEE Symposium on Logic in Computer Science*, pp. 75–83, 1993.
11. W. Charatonik and L. Pacholski. Negative set constraints with equality. In *9^{th} IEEE Symposium on Logic in Computer Science*, pp. 128–136, 1994.
12. W. Charatonik and L. Pacholski. Set Constraints with Projections are in NEXPTIME. In 35^{th} *Symposium on Foundations of Computer Science*, pp. 642–653, 1994.
13. W. Charatonik and A. Podelski. The Independence Property of a Class of Set Constraints. In *International Conference on Principles and Practice of Constraint Programming*, LNCS 1118, Springer, 1996.
14. W. Charatonik and A. Podelski. Set Constraints with Intersection. In *12^{th} IEEE Symposium on Logic in Computer Science*, Warsaw, Poland, 1997. to appear.
15. T. Frühwirth, E. Shapiro, M. Y. Vardi, and E. Yardeni. Logic Programs as Types for Logic Programs. *6^{th} IEEE Symposium on Logic in Computer Science*, pp. 300–309, 1991.
16. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.
17. R. Gilleron, S. Tison, and M. Tommasi. Solving Systems of Set Constraints using Tree Automata. In *10^{th} TACS*, LNCS 665, pp. 505–514. Springer, 1993.
18. R. Gilleron, S. Tison, and M. Tommasi. Solving Systems of Set Constraints with Negated Subset Relationships. In *34^{th} FOCS*, pp. 372–380, 1993.
19. N. Heintze. *Set Based Program Analysis.* PhD thesis, Carnegie Mellon Univ., 1992.

20. N. Heintze and J. Jaffar. A Decision Procedure for a Class of Set Constraints. In *5th IEEE Symposium on Logic in Computer Science*, 1990.
21. N. Heintze and J. Jaffar. A Finite Presentation Theorem for Approximating Logic Programs. *17th ACM Symposium on Principles of Programming Languages*, pp. 197–209, 1990.
22. F. Henglein and J. Rehof. The Complexity of Subtype Entailment for Simple Types. In *12th IEEE Symposium on Logic in Computer Science*, Warsaw, Poland, 1997.
23. J. Jaffar and M. J. Maher. Constraint Logic Programming: A Survey. *The Journal of Logic Programming*, 19/20:503–582, May-July 1994.
24. N. D. Jones and S. S. Muchnik. Flow Analysis and Optimization of Lisp-like Structures. In *6th ACM Symposium on Principles of Programming Languages*, pp. 244–256, 1979.
25. D. Kozen. Set Constraints and Logic Programming. In *1st International Conference on Constraints in Computational Logics*, LNCS 845, Springer, 1994.
26. M. J. Maher. Logic semantics for a class of committed-choice programs. In *Logic Programming, Proceedings of the 4th International Conference*, pp. 858–876, 1987. MIT Press.
27. S. Marlow and P. Wadler. A Practical Subtyping System for Erlang. In *2nd ACM SIGPLAN International Conference on Functional Programming*, 1997.
28. P. Mishra and U. Reddy. Declaration-free Type Checking. In *12th ACM Symposium on Principles of Programming Languages*, pp. 7–21, Jan. 1985.
29. M. Müller, J. Niehren, and A. Podelski. Inclusion Constraints over Non-Empty Sets of Trees. In International Joint Conference on Theory and Practice of Software Development (TAPSOFT), LNCS, Springer, 1997. A complete version can be found at `http://www.ps.uni-sb.de/~mmueller`
30. F. Pottier. Simplifying Subtyping Constraints. In *ACM SIGPLAN International Conference on Functional Programming*, pp. 122–133. May 1996.
31. Programming Systems Lab, Universität des Saarlandes. The Oz Programming System, 1996. `http://www.ps.uni-sb.de/www/oz/`.
32. J. Reynolds. Automatic Computation of Data Set Definitions. *Information Processing*, 68, 1969.
33. V. A. Saraswat. *Concurrent Constraint Programming*. MIT Press, 1993.
34. G. Smolka. The Oz Programming Model. In J. van Leeuwen, ed., *Computer Science Today*, LNCS 1000, pp. 324–343. Springer, 1995.
35. K. Stefansson. Systems of Set Constraints with Negative Constraints are NEXPTIME-Complete. *9th IEEE Symposium on Logic in Computer Science*, pp. 137–141, 1994.
36. V. Trifonov and S. Smith. Subtyping Constrained Types. In *International Static Analysis Symposium*, LNCS 1145, pp. 349–365. Springer, 1996.
37. T. E. Uribe. Sorted Unification Using Set Constraints. In *Conference on Automated Deduction*, pp. 163–177, 1992.

A Proofs missing from Section 4

In this Section we prove that

$$\alpha \text{ is a solution of } \Psi(\Gamma, x) \text{ iff } \alpha(x) \subseteq \sigma(\{\beta \mid \beta \text{ is a solution of } \neg\Gamma\})$$

The main Lemma is 10. First let us define some technical notions.

Path Reachability.

$$\begin{array}{ll} \varphi \vdash x \rightsquigarrow^{\varepsilon} y & \text{if } x \subseteq y \textit{ in } \varphi \\ \varphi \vdash x \rightsquigarrow^{i} y_i & \text{if } x \subseteq f(\ldots y_i \ldots) \textit{ in } \varphi \\ \varphi \vdash x \rightsquigarrow^{pq} y & \text{if ex. } z : \varphi \vdash x \rightsquigarrow^{p} z \text{ and } \varphi \vdash z \rightsquigarrow^{q} y \\ \varphi \vdash x \rightsquigarrow^{p} f & \text{if ex. } \overline{y}, z : x \rightsquigarrow^{p} z \wedge z \subseteq f(\overline{y}) \textit{ in } \varphi \end{array}$$

Path reachability for existential formulas Φ is defined by reference to the matrix of Φ, *e.g.*, $\Phi \vdash x \rightsquigarrow^{p} f$:iff $\varphi \vdash x \rightsquigarrow^{p} f$ for the matrix of Φ. For an arbitrary formula Φ and a variable x we define a $P_a(\varphi, x)$ the set of paths of length k leading from x through f's to a in φ:

$$P_a(\Phi, x) \;=\; \left\{ p \in \{1,2\}^k \;\middle|\; \begin{array}{l} \Phi \vdash x \rightsquigarrow^{p} a \text{ and} \\ \Phi \vdash x \rightsquigarrow^{p'} f \text{ if } p' \text{ proper prefix of } p \end{array} \right\}$$

For ease of writing, we also denote for a clause C or a SAT problem Γ with $P_a(C, x)$ the set of paths of length k leading from x through f's to a in the encoding of C (resp. Γ) wrt. x:

$$P_a(C, x) \;=\; P_a(\Psi'(p(C), x), x) \qquad P_a(\Gamma, x) \;=\; P_a(\Psi(\Gamma, x), x)$$

Observe that the following two Lemmas hold.

Lemma 8. $P_a(\Gamma, x) \;=\; \{p \mid \textit{ex. } f \textit{ such that } \Psi(\Gamma, x) \vdash x \rightsquigarrow^{p} f\}$.

Proof. By induction over the number of variables occurring in Γ, as done in more detail in the proof of Lemma 10.

Lemma 9. *If* $\Gamma = \bigwedge_{i=1}^{n} C_i$, *then* $P_a(\Gamma, x) \;=\; \bigcup_{i=1}^{n} P_a(C_i, x)$.

Proof. The inclusion from right to left is obvious. The inverse inclusion can be shown by induction on paths which exploits that $P_a(C_i, x)$ and $P_a(C_j, x)$ are variable-disjoint up to x. □

Further, we denote with $Sol(\neg\Gamma)$ the set of valuations p which make Γ false:

$$Sol(\neg C) \;=\; \{p \in \{1,2\}^k \mid p \models \neg C\}\,.$$

For ease of writing, we conflate boolean valuations β and the paths p_β associated with them and simply write p for both. Now the argument is as follows.

$$\alpha \text{ is a solution of } \Psi(\Gamma, x) \quad \text{iff} \quad \alpha(x) \subseteq \sigma(P_a(\Gamma, x)) \quad \text{iff} \quad \alpha(x) \subseteq \sigma(Sol(\neg\Gamma))$$

To see that step (1) is valid first note: Whenever α is a solution of $\Psi(\Gamma, x)$ and α' is coincides with α on x, then α' is a solution of $\Psi(\Gamma, x)$. Now, if α is a solution of $\Psi(\Gamma, x)$ then certainly $p \in \tau$ for all $\tau \in \alpha(x)$ and $p \in P_a(\Gamma, x)$, hence $\alpha(x) \subseteq \sigma(P_a(\Gamma, x))$. The converse follows from Lemma 8. The second step follows from Lemma 11.

Lemma 10. *For every clause C over variables $\{x_1, \ldots, x_k\}$, and for arbitrary x: $P_a(C, x) = Sol(\neg C)$*

Proof. By induction over k.

$k = 0$: Since the empty disjunction is $false$, we have to show that

$$P_a(C, x) = \{\varepsilon\} = Sol(\neg false)$$

Consider $\Psi'(p(false), x) = \Psi'(\varepsilon, x) = x \subseteq a$ and note that $P_a(C, a) = \{\varepsilon\}$. Further, $Sol(\neg false) = Sol(true) = \{\epsilon\}$ is the set of *all* valuations of null variables.

$k > 0$: We distinguish three cases.

$C = x_k \vee C'$: First note that

$$\Psi'(p(C), x) = \Psi'(2p(C'), x) = \exists y \exists y' (x \subseteq f(y, y') \wedge \Psi(p(C'), y'))\ .$$

Pick y, y' fresh. Then we have the following argument (where we suppress the paths $x \rightsquigarrow^{p'} f$ for ease of reading):

$$\begin{aligned}
P_a(C, x) &= \{p \in \{1,2\}^k \mid \Psi'(p(C), x) \vdash x \rightsquigarrow^p a\} \\
&= \{p \in \{1,2\}^k \mid x \subseteq f(y, y') \wedge \Psi(p(C'), y') \vdash x \rightsquigarrow^p a\} \\
&= \{2q \in \{1,2\}^k \mid \Psi(p(C'), y') \vdash y' \rightsquigarrow^q a\} \\
&= \{2q \mid q \in \{1,2\}^{k-1},\ \Psi(p(C'), y') \vdash y' \rightsquigarrow^q a\} \\
&= \{2q \mid q \in P_a(C', y')\} \\
&= \{2q \mid q \in Sol(\neg C')\} \\
&= Sol(\neg(x_k \vee C')) \;=\; Sol(\neg C)
\end{aligned}$$

$C = \neg x_k \vee C'$: Symmetric.

$C = C'$: First note that

$$\begin{aligned}
&\Psi'(p(C), x) = \Psi'(3p(C'), x) = \\
&\qquad \exists y \exists y' \exists y'' (x \subseteq f(y, y) \wedge y \subseteq y' \wedge y \subseteq y'' \wedge \Psi'(p(C'), y''))\ .
\end{aligned}$$

Pick y, y', y'' fresh. Then we have:

$$\begin{aligned} P_a(C,x) &= \{p \in \{1,2\}^k \mid \Psi'(p(C),x) \vdash x \rightsquigarrow^p a\} \\ &= \{p \in \{1,2\}^k \mid x \subseteq f(y,y) \wedge y \subseteq y' \wedge y \subseteq y'' \wedge \Psi'(p(C'),y'') \vdash x \rightsquigarrow^p a\} \\ &= \{1q \mid q \in \{1,2\}^{k-1},\ y' \subseteq y'' \wedge \Psi'(p(C'),y'') \vdash y' \rightsquigarrow^q a\} \\ &\quad \cup \{2q \mid q \in \{1,2\}^{k-1},\ y \subseteq y'' \wedge \Psi'(p(C'),y'') \vdash y \rightsquigarrow^q a\} \\ &= \{1q \mid q \in \{1,2\}^{k-1},\ \Psi'(p(C'),y') \vdash y' \rightsquigarrow^q a\} \\ &\quad \cup \{2q \mid q \in \{1,2\}^{k-1},\ \Psi'(p(C'),y) \vdash y \rightsquigarrow^q a\} \\ &= \{1q \mid q \in Sol(\neg C')\} \cup \{2q \mid q \in Sol(\neg C')\} \\ &= \{p \in \{1,2\}^k \mid q \in Sol(\neg C'), p = 1q \text{ or } p = 2q\}\} \\ &= Sol(\neg C) \end{aligned}$$

□

Lemma 11. *For every SAT problem Γ over variables $\{x_1, \ldots, x_k\}$, and for arbitrary x: $P_a(\Gamma, x) = Sol(\neg\Gamma)$*

Proof. Let $\Gamma = \bigwedge_{i=1}^{n} C_i$ and observe that

$$Sol(\neg\Gamma) \ = \ \bigcup_{i=1}^{n} Sol(\neg C_i) \ = \ \bigcup_{i=1}^{n} P_a(C_i, x) \ = \ P_a(\Gamma, x)$$

by Lemmas 9 and 10. □