On the Complexity of the Bernays-Schönfinkel Class with Datalog

Witold Charatonik and Piotr Witkowski
{wch, pwit}@ii.uni.wroc.pl

Institute of Computer Science
University of Wrocław

Abstract. Bernays-Schönfinkel class with Datalog is a 2-variable fragment of the Bernays-Schönfinkel class extended with least fixed points expressible by certain monadic Datalog programs. It was used in a bounded model checking procedure for programs manipulating dynamically allocated pointer structures, where the bounded model checking problem was reduced to the satisfiability of formulas in this logic. The best known upper bound on the complexity of the satisfiability problem for this logic was 2NEXPTIME.
In this paper we extend the Bernays-Schönfinkel class with Datalog to a more expressive logic — a fragment of two-variable logic with counting quantifiers extended with the same kind of fixed points. We prove that both satisfiability and entailment for the new logic are decidable in NEXPTIME and we give a matching lower bound for the original logic, which establishes NEXPTIME-completeness of the satisfiability and entailment problems for both of them. Our algorithm is based on a translation to 2-variable logic with counting quantifiers.

1 Introduction

Automated verification of programs manipulating dynamically allocated pointer structures is a challenging task. The reachable state space of such programs is infinite and the verification problem is undecidable. Moreover, to reason about sets of nodes reachable from a program variable one often wants to be able to compute transitive closure or least fixed points of some operators. Unfortunately, even quite simple logics like two-variable fragments of first-order logic or even two-variable fragments of Bernays-Schönfinkel class extended with either transitive closure or least fixed points quickly become undecidable [10,13].

In [7] the authors proposed a bounded model checking procedure for imperative programs that manipulate dynamically allocated pointer structures on the heap. Although in this procedure an explicit bound is assumed on the length of the program execution, the size of the initial data structure is not bounded. Therefore, such programs form infinite and infinitely branching transition systems. The procedure is based on the following four observations. First, error conditions like dereference of a dangling pointer, are expressible in the two-variable fragment of Bernays-Schönfinkel class with equality. Second, the fragment is closed under weakest preconditions wrt. finite paths. Third, data structures like trees, lists (singly or doubly linked, or even circular) are expressible in a fragment of monadic Datalog. Finally, the combination of the Bernays-Schönfinkel

fragment with Datalog fragment is decidable. The bounded model checking problem for pointer programs is then reduced to the satisfiability of formulas in Bernays-Schönfinkel class with Datalog. The authors gave an algorithm solving the satisfiability problem in 2NEXPTIME.

In this paper, we improve the logic from [7] in terms of both expressibility and complexity. The two-variable fragment of Bernays-Schönfinkel class (BS_2 for short) is simply a fragment of first-order logic restricted to two universally quantified variables. In the new logic, we are able to use both universal and existential quantifiers and additionally we may use counting in quantifiers. Moreover, the logic in [7] has additional semantic restrictions, some of which we are able to drop here. This gives us an increase of expressibility.

We give an algorithm for testing satisfiability of our logic that runs in NEXPTIME. The algorithm is based on a translation to 2-variable logic with counting quantifiers. Since our logic subsumes the Bernays-Schönfinkel class with Datalog (further called BS_2 + Datalog), we improve the previous complexity upper bound. We also give a matching lower bound and establish NEXPTIME-completeness of both logics. Finally, we also solve the entailment problem, which was left open in [7]. It has further consequences for the applicability of the logic in verification.

The paper is organized as follows. The main results on satisfiability and entailment are proved in Section 3. Section 4 gives the corresponding lower bound. In Section 5 we show that dropping one of the restrictions on sharing data that forbids us to model directed acyclic graphs leads to a very complex logic. The main part of the paper is finished with the discussion on related and future work. In the appendix we give some proofs that did not fit in the page limit and an example showing that our logic can also be used to specify verification conditions of nontrivial program manipulating linked data structures.

2 Preliminaries

Let Σ_E (extensional database vocabulary) be a vocabulary containing relational symbols of arity at most 2 and functional symbols of arity 0 (that is, constants). Let Σ_I (intensional database vocabulary) be a relational vocabulary containing unary symbols only and let Σ_P be a set containing some of Σ_E binary predicates. Assume in addition that a countably infinite set of variables V is available. Σ_E is a vocabulary of some (to be precised later) two-variable logic formula, Σ_I defines symbols that occur in heads of clauses from Datalog programs and Σ_P — binary symbols used in their bodies. Following [7], we are interested in *monadic tree-automaton like* Datalog programs, which are Datalog programs whose clauses are similar to transitions in edge-labeled tree automata.

Definition 1. *A monadic tree-automaton like Datalog program over Σ_I,Σ_E and Σ_P is a conjunction of clauses of the form*

$$p(u) \leftarrow B, r_1(u, v_1), q_1(v_1), \ldots, r_l(u, v_l), q_l(v_l)$$

where

- *B is a conjunction of Σ_E-literals containing constants and possibly the variable u,*

– $p, q_1, \ldots, q_l$ *are* Σ_I*-predicates,*
– $r_1, \ldots, r_l$ *are distinct* Σ_P*-predicates,*
– $l \geq 0, u, v_1, \ldots, v_l$ *are distinct variables from* V.

Monadic tree-automata like Datalog programs are further called (monadic, t.a. like) Datalog programs for short.

Example 1. A binary tree with *left* and *right* pointers, whose leaves point to constant $NULL$ is defined by Datalog program

$$tree(x) \leftarrow x = NULL;\ tree(x) \leftarrow left(x, y), tree(y), right(x, z), tree(z).$$

Let P be a monadic Datalog program. Given a Σ_E-structure M, the *least extension* of M wrt. P is the least $(\Sigma_E \cup \Sigma_I)$-structure that is a model of the clause set P. More formally, we have the following definition.

Definition 2 (Least extension). *Let M be a relational structure over Σ_E, P be a monadic Datalog program over Σ_I,Σ_P and Σ_E whose variables come from V. Let $F^M = \{p(e) \mid p(\cdot) \in \Sigma_I \wedge e \in M\}$. The direct consequence operator T_P^M is a function from $2^{F^M} \rightarrow 2^{F^M}$ defined as follows.*

$$T_P^M(I) = I \cup \{p(e) \mid \exists[p(x) \leftarrow body] \in P. \exists \Theta \in V \rightarrow M.\ \ \Theta(x) = e \wedge M \cup I \models (body\Theta)\}$$

Let $T_P^M = \bigcup_{i=1}^{\infty}(T_P^M)^i(\emptyset)$. Then the least extension of M with respect to P is a structure M_P over $\Sigma_E \cup \Sigma_I$, such that $M_P = M \cup T_P^M$.

Definition 3 (Sliced Datalog program). *A monadic tree-automaton like Datalog program P over Σ_I, Σ_E and Σ_P is called sliced if, for some natural k, $P = \bigwedge_{i=1}^{k} P_i$ where*

– *each P_i is a monadic tree-automaton like Datalog program over vocabularies Σ_I^i ,Σ_P^i and Σ_E,*
– *$\Sigma_I^i \cap \Sigma_I^j = \emptyset$ and $\Sigma_P^i \cap \Sigma_P^j = \emptyset$ for $i \neq j$, $1 \leq i, j \leq k$,*
– *$\bigcup_{i=1}^{k} \Sigma_I^i = \Sigma_I$ and $\bigcup_{i=1}^{k} \Sigma_P^i = \Sigma_P$.*

We sometimes write $\{P_1, \ldots, P_k\}$ instead of $\bigwedge_{i=1}^{k} P_i$. In the following, we will use sliced programs in two ways. The first is to control if some data structures are allowed or are forbidden to share substructures (see the bounded-sharing restriction below). The second way is to solve the entailment problem, where we have to assure that the structures defined by the two formulas may interfere.

The two-variable fragment of Bernays-Schönfinkel class (BS_2 for short) is the set of all formulas of the form $\forall x \forall y\ \phi$, where ϕ is a quantifier-free first-order formula over Σ_E. We specify reachability predicates via sliced Datalog programs.

Definition 4 (BS_2 + Datalog**).** *A formula of Bernays-Schönfinkel class with Datalog is a conjunction $\phi \wedge P \wedge Q$ where*

– *ϕ is a BS_2 formula over Σ_E,*
– *$P = \{P_1, \ldots, P_k\}$ is a sliced monadic tree-automaton like Datalog program*

– *query Q is conjunction of constant positive atoms $\bigwedge_i p_i(c_i)$ (where $p_i \in \Sigma_I$ and $c_i \in \Sigma_E$) and BS_2 class formulas over $\Sigma_E \cup \Sigma_I$ with negative occurrences of $p_i(x)$ atoms only ($p_i \in \Sigma_I$ and x is a variable).*

Example 2. Tree rooted in a constant *root* whose nodes have backward link r to *root* is specified as BS_2 + Datalog formula $\phi \wedge P \wedge Q$, where Q is $tree(root) \wedge \forall_x tree(x) \rightarrow (x \neq NULL \rightarrow r(x, root))$, P is Datalog program from example 1 (with one slice only) and ϕ is simply *true*.

For a given Σ_E-formula ϕ and a query Q, we say that Σ_E-structure M *satisfies* $\phi \wedge P \wedge Q$, denoted by $M \models \phi \wedge P \wedge Q$, if M_P is a model of $\phi \wedge Q$, where M_P is the least extension of M wrt. P. However, this still does not define the semantics of Bernays-Schönfinkel class with Datalog. The logic was designed to model heaps of pointer programs, therefore we are not interested in general satisfiability of formulas in this class, but we impose two additional restrictions on models.

Functionality. We require that all binary relations are functional, i. e., for all predicates r in Σ_E, structure M (and thus also M_P must be a model of $\forall u, v_1, v_2(r(u, v_1) \wedge r(u, v_2) \rightarrow v_1 = v_2)$. This ensures that every pointer at any given moment points to at most one heap cell.

Bounded-Sharing. We require that the binary relations occurring in each slice P_i of the Datalog program $P = \{P_1, \ldots, P_k\}$ represent pointers in data structures that do not share memory with other data structures defined by P_i. That is, structure M_P must be a model of all sentences of the form $\forall u_1, u_2, v\ (s_1(u_1, v) \wedge s_1(u_2, v) \wedge u_1 \neq u_2 \rightarrow const(v))$ and $\forall u_1, u_2, v(s_1(u_1, v) \wedge s_2(u_2, v) \rightarrow const(v))$, where s_1 and s_2 are two distinct predicates occurring in Σ_P^i and $const(v)$ is a shorthand for the disjunction $\bigvee_{c \in \Sigma_E} v = c$. Note that the bounded-sharing restriction is not imposed on all binary predicates but just on the ones occurring in the Datalog program P.

As an example consider two programs defining lists

$$list_1(x) \leftarrow x = NULL;\ list_1(x) \leftarrow next_1(x, y), list_1(y).$$
$$list_2(x) \leftarrow x = NULL;\ list_2(x) \leftarrow next_2(x, y), list_2(y).$$

together with a formula $\forall x \neg next_1(NULL, x) \wedge \neg next_2(NULL, x)$. If the two programs are in the same slice then, with the exception of constants, they are not allowed to have common nodes. On the other hand, if they are in different slices, they may freely share nodes.

The functionality and bounded-sharing restrictions are expressible in the Bernays-Schönfinkel class with equality, but they require more than two variables, so we cannot add them to the formula ϕ. Each Datalog slice P_i can define multiple structures. Constant elements are intended to model nodes pointed to by program variables. Such nodes are allowed to be pointed to by elements of different structures, even from the same slice.

Definition 5 (Semantics of Bernays-Schönfinkel class with Datalog). *Let $\varphi = \phi \wedge P \wedge Q$ and M be a finite structure over Σ_E such that*

– *Structure M_P, that is the least extension of M wrt. P, obeys functionality and bounded-sharing restrictions,*

– $M_P \models \phi \wedge Q$,

Then M is said to satisfy φ, in symbols $M \models \varphi$.

Remark 1. Contrary to our definition of Σ_E, in [7] vocabulary of ϕ doesn't contain unary relational symbols. One can however get rid of them by introducing additional constant (say d) and replacing each occurrence of atom $r(x)$ by $r'(x, d)$, where r' is fresh binary symbol. Thus presence of unary symbols in Σ_E incurs no increase in asymptotic complexity of satisfiability problem. Furthermore in [7] queries Q are limited to be conjunction of constant positive atoms, only one-sliced Datalog programs were allowed and admissible models obeyed additional semantic restriction called intersection-freeness. We removed here this last restriction since it is expressible as a query $\bigwedge_{p(\cdot)\in\Sigma_I} \bigwedge_{q(\cdot)\in\Sigma_I, q\neq p} \forall v(p(v) \wedge q(v) \rightarrow const(v))$. These extensions didn't however increase asymptotical complexity of satisfiability and entailment problems compared to [7]. Lower bound proof from section 4 holds as is for logic defined in [7].

Definition 6 (Derivation). *Let M be a structure over Σ_E satisfying the bounded-sharing restriction, let P be a monadic sliced t.a. like Datalog program, let $p \in \Sigma_I$ and $e \in M$. A* derivation *$deriv(p(e))$ is a tree labeled with atoms of the form $q(a)$ with $q \in \Sigma_I$ and $a \in M$ such that*

– *the root of $deriv(p(e))$ is labeled with $p(e)$,*
– *if a leaf of $deriv(p(e))$ is labeled with $q(a)$ then there exists a clause $[q(x) \leftarrow B] \in P$ and a valuation Θ such that $\Theta(x) = a$ and $M \models B\Theta$,*
– *for every inner node of $deriv(p(e))$ labeled with $q(a)$ there exists a clause $[q(u) \leftarrow B \wedge \bigwedge_{i=1}^{k}(r_i(u, v_i) \wedge q_i(v_i))] \in P$ and a valuation Θ, such that $\Theta(u) = a$, $M \models B\Theta \wedge \bigwedge_{i=1}^{k} r_i(a, \Theta(v_i))$ and the children of $q(a)$ are roots of derivations $deriv(q_i(\Theta(v_i))$ for all $1 \leq i \leq k$.*

Note that since vocabularies of different slices are disjoint, the whole tree $deriv(p(e))$ contains only symbols from one slice.

A constant atom is an atom of the form $p(c)$ where c is an element interpreting a constant symbol. A constant derivation is a derivation of a constant atom. A minimal derivation is a derivation whose all subderivations have the minimal possible height. Unless otherwise specified, all derivations we consider are minimal.

Remark 2. Without loss of generality we assume that all leaves in derivations are constant atoms, that is, for all clauses of the form $p(u) \leftarrow B$ in P there exists a constant $c \in \Sigma_E$ such that $u = c$ is a conjunct in B. If $p(u) \leftarrow B$ does not satisfy this requirement, then it is replaced by $p(u) \leftarrow B, c(u, y), c(y)$ and $c(x) \leftarrow x = c$, where c, $c(\cdot)$ and $c(\cdot, \cdot)$ are fresh constant, fresh unary Σ_I-predicate and fresh binary Σ_P-predicate, respectively. By definition of vocabularies $c(\cdot, \cdot)$ is then also a fresh Σ_E-predicate.

Definition 7 ($p(e)$-tree). *A $p(e)$-tree is a maximal subtree of a derivation $deriv(p(e))$ with root labeled $p(e)$ and no inner nodes labeled with constant atoms.*

By remark 2 leaves of each finite derivation are labeled by constant atoms, so are leaves of any $p(e)$-tree. The least extension (Definition 2) gives a model-theoretic semantics of Datalog programs over Σ_E-structures; derivations (Definition 6) gives a proof-theoretic semantics. By a simple inductive proof it is not difficult to see that they are equivalent.

Proposition 1. *Let M be a finite structure satisfying the bounded-sharing restriction. There exists a finite derivation of $p(e)$ if and only if $p(e) \in T^P_M$.*

3 NEXPTIME upper bound

In this section we prove that the satisfiability and entailment problem for Bernays-Schönfinkel class with Datalog is decidable in NEXPTIME. We give a satisfiability (resp. entailment) preserving polynomial translation from Bernays-Schönfinkel class with Datalog to two-variable logic with counting quantifiers.

The two variable logic with counting (C^2) is a fragment of first order logic containing formulas whose all subformulas have at most two free variables, but these formulas may contain counting quantifiers $\exists^{\leq k}$, $\exists^{\geq k}$, $\exists^{=k}$. Decidability of the satisfiability problem was discovered independently in [11] and [26]. First NEXPTIME results, under unary encoding of counts were obtained in [26], [27] and under binary coding in [28]. While C^2 doesn't enjoy the finite model property it is natural to ask for finite satisfiability. This question was positively answered in [11], NEXPTIME complexity was established by [28] even under binary encoding. The (finite) satisfiability of C^2 is NEXPTIME-hard as a consequence of NEXPTIME-hardness of the two variable fragment of first-order logic.

Our translation is done in two steps. First we translate the input formula to an intermediate logic C^2_r + Datalog + bsr (restricted C^2 with Datalog and bounded-sharing restriction), and then we translate the resulting formula to C^2. The reason to introduce this intermediate logic is that it is more expressive than Bernays-Schönfinkel class with Datalog and may be of its own interest. In this logic the Bernays-Schönfinkel part of the formula (that is, the ϕ conjunct in a formula of the form $\phi \wedge P \wedge Q$) is relaxed to use arbitrary quantifiers (not only the $\forall$ quantifier), even with counting; the query Q is incorporated into the conjunct ϕ and relaxed to be arbitrary formula with constant atoms and restricted occurrences of non-constant Σ_I-atoms. This is a good step forward in ability to express more complicated verification conditions of pointer programs. Moreover, we skip one out of two semantic restrictions on models, namely functionality which is expressible in C^2. The bounded-sharing restriction could not be dropped — it is the restriction that allows to keep the complexity of Datalog programs under control. In Section 5 we show that the complexity of unrestricted C^2_r with Datalog is much worse than NEXPTIME.

Let ϕ be a C^2 formula over $\Sigma_E \cup \Sigma_I$ and let ϕ' be its negational normal form. We say that a Σ_I-atom $p(x)$ has a restricted occurrence in ϕ if either $p(x)$ occurs positively in ϕ' and only in in scope of existential quantifiers or $p(x)$ occurs negatively in ϕ' and only in scope of universal quantifiers. For example $p(x)$ has a restricted occurrence in formulas $\forall x\ p(x) \rightarrow \psi$, $\forall x\ (p(x) \wedge q(x)) \rightarrow \psi$, $\exists_x\ p(x) \wedge \psi$ or $\exists_x\ p(x) \wedge q(x) \wedge \psi$, where ψ is some C^2 formula with one free variable x and $q(\cdot)$ is some Σ_I-predicate. An occurrence of atom $p(x)$ in formula $\forall_y \exists_x p(x) \wedge \psi$ is not restricted, because $p(x)$ is occurs positively and in scope of a $\forall$ quantifier.

Definition 8 (Syntax of C^2_r + Datalog + bsr). *A formula of C^2_r + Datalog + bsr is a conjunction of the form $\phi \wedge P$ such that*

– *ϕ is a formula of the two-variable logic with counting quantifiers over the signature $\Sigma_E \cup \Sigma_I$,*
– *all Σ_I-literals occurring in ϕ are either constant literals or have only restricted occurrences in ϕ, and*
– *P is a sliced monadic tree-automaton like Datalog program.*

Definition 9 (Semantics of C_r^2 + Datalog + bsr). *Let $\varphi = \phi \wedge P$ be a formula of C_r^2 + Datalog + bsr and let M be a finite structure over Σ_E such that*

– *Structure M_P, that is the least extension of M wrt. P, obeys bounded-sharing restriction, and*
– *$M_P \models \phi$.*

Then M is said to satisfy φ, in symbols $M \models \varphi$.

The following proposition reduces satisfiability of BS_2 + Datalog to satisfiability of C_r^2 + Datalog + bsr.

Proposition 2. *For every formula $\varphi = \phi \wedge P \wedge Q$ in Bernays-Schönfinkel class with Datalog there exists an equivalent formula ψ of C_r^2 + Datalog + bsr of size polynomial in the size of φ.*

Proof. Let ψ be the conjunction of the following formulas

1. $\phi \wedge Q$,
2. $\bigwedge_{r(\cdot,\cdot)\in\Sigma_E} \forall u \exists^{\leq 1} v\ r(u,v)$,
3. P.

Then ψ is a formula of C_r^2 + Datalog + bsr, because ϕ is a BS_2 formula and all Σ_I-atoms in $\phi \wedge Q$ are either constant atoms or have restricted occurrences only. Furthermore conjunct 2 expresses the functionality restriction, so ψ is equivalent to φ. □

We say that a formula φ is entails a formula φ' (in symbols $\varphi \models \varphi'$) if for all structures M we have that $M \models \varphi$ implies $M \models \varphi'$. Below we show that entailment problem of BS_2 + Datalog is reducible to satisfiability of C_r^2 + Datalog + bsr. We start with an observation that names of predicates defined by Datalog programs may be arbitrarily changed. By a renaming function we mean here any bijection between different vocabularies.

Lemma 1. *Let $\varphi = \phi \wedge P \wedge Q$ be a BS_2 + Datalog class formula, where P is a sliced Datalog program over Σ_I and Σ_P. Let φ_{ren} be a formula φ where Σ_I-predicates were arbitrarily renamed. Then $M \models \varphi$ if and only if $M \models \varphi_{ren}$.*

Proposition 3 (entailment). *For every formulas $\varphi = \phi \wedge P \wedge Q$ and $\varphi' = \phi' \wedge P' \wedge Q'$ in the BS_2 + Datalog class there exists a formula ψ of C_r^2 + Datalog + bsr of size polynomial in the size of φ and φ', such that $\varphi \models \varphi'$ if and only if ψ is unsatisfiable.*

Proof. Let Σ_I,Σ_P be vocabularies of sliced monadic t.a. like Datalog program P, let $\Sigma_{I'}$,$\Sigma_{P'}$ be vocabularies of P'. By previous lemma it may be w.l.o.g assumed, that $\Sigma_I \cap \Sigma_{I'} = \emptyset$. Furthermore if $r(\cdot,\cdot) \in \Sigma_P \cap \Sigma_{P'}$ then replacing each occurrence of $r(\cdot,\cdot)$ in P by a fresh predicate $r'(\cdot,\cdot)$ and adding conjunct $\forall_x \forall_y r(x,y) \iff r'(x,y)$ to ϕ

decreases cardinality of $\Sigma_P \cap \Sigma_{P'}$ while preserving entailment. By iteratively applying this procedure it can be ensured, that also $\Sigma_P \cap \Sigma_{P'} = \emptyset$. Let $P = \{P_1, \ldots, P_k\}$ and $P' = \{P'_1, \ldots, P'_l\}$. It follows that $\{P_1, \ldots, P_k, P'_1, \ldots, P'_l\}$ is also a sliced monadic Datalog program. Formula ψ is defined to be $(\phi \wedge Q \wedge \neg(\phi' \wedge Q)) \wedge \{P_1, \ldots, P_k, P'_1, \ldots, P'_l\}$. Notice also, that by definition of BS_2 + Datalog formulas, all Σ_I-atoms in $\neg(\phi' \wedge Q')$ are either constant atoms or have restricted occurrences, so ψ is a correct C_r^2 + Datalog + bsr formula. We now prove, that $\varphi \models \varphi'$ if and only if ψ is unsatisfiable. Let Σ be a dictionary of formulas ϕ and ϕ' (in particular $\Sigma_P \cup \Sigma_{P'} \subseteq \Sigma$). Let M be an arbitrary structure over Σ. Then $M \models \phi \wedge P \wedge Q$ if and only if the least extension of M wrt. P models $\phi \wedge P$: $M_P \models \phi \wedge Q$. Since $\Sigma_P \cap \Sigma_{P'} = \emptyset$, this is equivalent to $M_{P \wedge P'} \models \phi \wedge Q$. Similarly, $M \models \phi' \wedge P' \wedge Q'$ is equivalent to $M_{P \wedge P'} \models \phi' \wedge Q'$. Therefore, by definition of entailment, $\phi \wedge P \wedge Q \not\models \phi' \wedge P' \wedge Q'$ if and only if there exists a structure M, such that $M_{P \wedge P'} \models \phi \wedge Q$ and $M_{P \wedge P'} \not\models \phi' \wedge Q'$, that is if $(\phi \wedge Q \wedge \neg(\phi' \wedge Q)) \wedge (P \wedge P')$ is satisfiable. □

We now give a polynomial translation from satisfiability of C_r^2 + Datalog + bsr to finite satisfiability of C^2. First the problem is simplified a bit by removing positive occurrences of non-constant Σ_I- atoms.

Proposition 4. *Let $\varphi = \phi \wedge P$ be a C_r^2 + Datalog + bsr formula. Then there exists C_r^2 + Datalog + bsr formula $\varphi' = \phi' \wedge P$, such that φ' is satisfiable if and only if φ is satisfiable and no atom of the form $p(x)$ ($p(\cdot) \in \Sigma_I$ and x is a variable) occurs positively in ϕ'.*

Proof. Assume w.l.o.g. that ϕ is in negational normal form. By definition of C_r^2 + Datalog + bsr, each positive occurrence of atom $p(x)$ is in scope of $\exists$ quantifiers only. In particular variable x is existentially quantified, so it can be skolemized. This process replaces x by a fresh constant. By iteration a required formula ϕ' can be obtained. □

Definition 10 (Translation from C_r^2 + Datalog + bsr to C^2). *Let $\varphi = \phi \wedge P$ be a formula of C_r^2 + Datalog + bsr over the vocabulary $\Sigma_E \cup \Sigma_I$, such that no atom of the form $p(x)$ ($p(\cdot) \in \Sigma_I$ and x is a variable) occurs positively in ϕ. Let Σ_P be the set of binary relations from $P = \{P_1, \ldots, P_k\}$ and let Σ be a vocabulary containing $\Sigma_E \cup \Sigma_I$ and predicates $const(\cdot), \{p(c)\text{-}node_q(\cdot), p(c)\text{-}edge_q(\cdot,\cdot), p(c)\text{-}leaf_q(\cdot) \mid p, q \in \Sigma_I, c \in \Sigma_E\}$. The translation $\tau(\varphi)$ of φ to C^2 over Σ is the conjunction of the following formulas.*

1. ϕ
2. $\forall x const(x) \leftrightarrow \bigvee_{c \in \Sigma_E} x = c$
3. $$\bigwedge_{i=1}^{k} \forall v(\neg const(v) \rightarrow \exists^{\leq 1} u(\bigvee_{r(\cdot,\cdot) \in \Sigma_P^i} r(u,v)))$$
   where $\Sigma_P^1, \ldots, \Sigma_P^k$ are vocabularies of binary symbols used by clauses from slices $P_1, \ldots, P_k$ respectively,
4. $$\bigwedge_{i=1}^{k} \bigwedge_{r(\cdot,\cdot) \in \Sigma_P^i} \bigwedge_{s(\cdot,\cdot) \in \Sigma_P^i, s \neq r} \forall u \forall v(r(u,v) \wedge s(u,v) \rightarrow const(v))$$

5.
$$\forall u\ p(u) \leftarrow \bigwedge_{i=1}^{l} \exists v(r_i(u,v) \wedge q_i(v)) \wedge B(u)$$
for all Datalog clauses $p(u) \leftarrow B(u), r_1(u,v_1), q_1(v_1), \ldots, r_l(u,v_l), q_l(v_l)$ *from P,*

6.
$$(q(c) \rightarrow \bigvee_{j=1}^{m}(\bigwedge_{i=1}^{l_j} \exists v(r_i^j(c,v) \wedge q_i^j(v) \wedge p(c)\text{-}edge_{q^j}(c,v)) \wedge B^j(c)))$$
and

$$\forall u(q(u) \wedge \neg const(u) \wedge p(c)\text{-}node_q(u) \rightarrow \bigvee_{j=1}^{m}(\bigwedge_{i=1}^{l_j} \exists v(r_i^j(u,v) \wedge q_i^j(v) \wedge p(c)\text{-}edge_{q^j}(u,v)) \wedge B^j(u)))$$
for all $q \in \Sigma_I$*, all* $c \in \Sigma_E$ *and all* $p \in \Sigma_I$ *where* $\{q(u) \leftarrow B^j(u) \wedge \bigwedge_{i=1}^{l_j}(r_i^j(u,v_i) \wedge q_i^j(v_i))\}_{j=1}^{m}$ *is the set of all clauses from P defining the predicate q,*

7.
$$\forall x \forall y\ p(c)\text{-}edge_q(x,y) \wedge \neg const(y) \rightarrow p(c)\text{-}node_q(y)$$
for all $p(\cdot), q(\cdot) \in \Sigma_I$ *and all* $c \in \Sigma_E$

8.
$$\forall x\ p(c)\text{-}edge_q(x,d) \rightarrow p(c)\text{-}leaf_q(d)$$
for all $p(\cdot), q(\cdot) \in \Sigma_I$ *and all* $c, d \in \Sigma_E$*,*

9.
$$p(c)\text{-}leaf_q(d) \wedge q(d)\text{-}leaf_r(e) \rightarrow p(c)\text{-}leaf_r(e)$$
for all $p(\cdot), q(\cdot), r(\cdot) \in \Sigma_I$ *and all* $c, d, e \in \Sigma_E$

10.
$$\neg p(c)\text{-}leaf_p(c)$$
for all $p(\cdot) \in \Sigma_I$ *and all* $c \in \Sigma_E$*.*

The intuition behind this translation is the following. First, observe that the conjuncts 3 and 4 express the bounded-sharing restriction. Then we want to rewrite Datalog clauses to implications (conjunct 5), but we need also the reverse directions of these implications — this is done with conjunct 6, separately for elements interpreting and not interpreting constants. Now we are almost done, but this construction possibly leads to cyclic structures (that would correspond to infinite derivations), but thanks to the bounded-sharing restriction it is possible to forbid cycles containing constant elements, which is enough for correctness since only formulas from C_r are allowed. Let $M \models \tau(\varphi)$. Then for any given $p(c)$, such that $M \models p(c)$, the set $\{p(c)\text{-}node_q(e) \mid q \in \Sigma_I \wedge M \models p(c)\text{-}node_q(e)\}$ forms a superset of non-constant nodes of a $p(c)-tree$, the set $\{p(c)\text{-}edge_q(e_1,e_2) \mid q \in \Sigma_I \wedge M \models p(c)\text{-}edge_q(e_1,e_2)\}$ a superset of its edges and finally the set $\{p(c)\text{-}leaf_q(d) \mid q \in \Sigma_I \wedge M \models p(c)\text{-}leaf_q(d)\}$ is a superset of its leaves. This is enforced by conjuncts 6–8. Irreflexivity (conjunct 10) of transitive closure (conjunct 9) of "being a leaf of" relation assures inexistence of cycles. More formally, we have the following proposition that we prove in the appendix.

Proposition 5. *Let $\varphi = \phi \wedge P$ be a C^2_r + Datalog + bsr formula. Then φ is satisfiable if and only if the C^2 formula $\tau(\varphi)$ is finitely satisfiable.*

Observe that the size of $\tau(\varphi)$ is polynomial in the size of φ, which together with Proposition 4 gives us the following corollary.

Corollary 1. *The satisfiability problem for C^2_r + Datalog + bsr is in NEXPTIME.*

Together with Proposition 2 and Proposition 3 this gives us our main theorem.

Theorem 1. *The satisfiability and entailment problems for Bernays-Schönfinkel class with Datalog are in NEXPTIME.*

4 NEXPTIME lower bound

In this section we prove that the satisfiability problem for Bernays-Schönfinkel class with Datalog is NEXPTIME-hard. This is done by a reduction from the following version of a tiling problem. It is known [9] that this version of the tiling problem is NEXPTIME-complete.

Definition 11 (Tiling problem).
***Instance**: a tuple $\langle T, H, V, k\rangle$ where T is a finite set of tile colors, $H, V \subseteq T \times T$ are binary relations and k is a positive integer encoded in binary (on $\lceil \log k \rceil$ bits).*
***Question**: does there exist a tiling of $k \times k$ square by 1×1 tiles such that colors of each pair of horizontally adjacent tiles satisfy the relation H and colors of each pair of vertically adjacent tiles satisfy V?*

If the question in the tiling problem has a positive answer then we say that the instance $\langle T, H, V, k\rangle$ has a solution or that there exists a good tiling for this instance. In the following, for a given instance $\langle T, H, V, k\rangle$ we define a formula φ of the Bernays-Schönfinkel class with Datalog such that φ is satisfiable if and only if there exists an appropriate tiling.

The idea is that any model of the formula describes a good tiling and any such tiling defines a model of φ. The size of the formula will be polynomial in $|T| + |H| + |V| + \lceil \log k \rceil$. Using simple Datalog program and simulating two counters by $\forall\forall$ formulas it is enforced that any model of φ has $k \times k$ square as a substructure. Is is then enough to constrain pairs of adjacent nodes according to H and V. Conversely any good tiling can be labelled by Σ_E predicates to form model of φ. The detailed description is given in the appendix.

Theorem 2. *The satisfiability problem for Bernays-Schönfinkel class with Datalog is NEXPTIME-hard.*

The monadic Datalog program of the BS_2 + Datalog formula constructed here has only one slice and its query Q is simply a constant Σ_I-atom. There are also no two distinct Σ_I-predicates. Hence, this reduction also establishes NEXPTIME lower bound for the satisfiability and entailment problems for the logic defined in [7].

5 Hardness of C_r^2 + Datalog

In Section 3 we have shown that the logic C_r^2 + Datalog + bsr is decidable in NEXPTIME. In this section we show that if we skip the bounded-sharing restriction then the obtained logic C_r^2 + Datalog becomes much harder. Dropping this restriction is quite tempting, because it would allow to reason about data structures that do share substructures, like DAG representations of trees. Here we prove that with the possibility to share structure, the logic becomes powerful enough to express vector addition systems (VAS). The reduction in Section 3 relied on the fact that in order to conform with the least fixed point semantics of Datalog we have to keep cycles on the heap under control, and with the bounded-sharing restriction we have to worry only about cycles with occurrences of constants. Without this restriction we would have to additionally handle cycles that avoid constants, which is out of scope of the C^2 logic.

Vector addition systems [25] is a very simple formalism equivalent to Petri Nets. It is known that its reachability problem is decidable [18,23,21] and EXPSPACE-hard [22], but precise complexity is not known, and after almost 40 years of research it is even not known if the problem is elementary. Below we give an exponential reduction from the reachability problem for VAS to the satisfiability problem for C_r^2 + Datalog. Although we believe that C_r^2 + Datalog is decidable, it is very unlikely that it has an elementary decision algorithm since existence of such an algorithm implies existence of an elementary algorithm for VAS reachability.

Now we recall definitions from [25]. An n-dimensional vector addition system is an ordered pair $\langle \boldsymbol{v}, W \rangle$ where $\boldsymbol{v}$ is an n-tuple of non-negative integers and W is a finite set of n-tuples of integers. Given two n-tuples $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ and $\boldsymbol{y} = \langle y_1, \ldots, y_n \rangle$ we write $\boldsymbol{x} \leq \boldsymbol{y}$ if $x_i \leq y_i$ for all $i = 1, \ldots, n$ and we define $\boldsymbol{x}+\boldsymbol{y}$ to be the tuple $\langle x_1+y_1, \ldots, x_n+y_n \rangle$. The reachability problem for VAS is the problem whether for a given VAS $\langle \boldsymbol{v}, W \rangle$ there exists a finite sequence $\boldsymbol{w_1}, \ldots, \boldsymbol{w}_m$ such that

- $\boldsymbol{w}_i \in W$ for all $i = 1, \ldots, m$,
- $\boldsymbol{v} + \boldsymbol{w}_1 + \ldots + \boldsymbol{w}_i \geq \boldsymbol{0}$ for all $i = 1, \ldots, m$, and
- $\boldsymbol{v} + \boldsymbol{w}_1 + \ldots + \boldsymbol{w}_m = \boldsymbol{0}$

where $\boldsymbol{0}$ is the n-tuple $\langle 0, \ldots, 0 \rangle$.

To reduce VAS reachability problem to satisfiability in C_r^2 + Datalog we first represent solutions to the reachability problem as directed acyclic graphs and then write a formula that describes such graphs.

Definition 12 (Transition sequence dag). *Let $\langle \boldsymbol{v}, W \rangle$ be an n-dimensional VAS. A transition sequence dag for $\langle \boldsymbol{v}, W \rangle$ is a directed acyclic multigraph G such that*

- *nodes of G are $n_0, n_1, \ldots, n_m$ for some positive integer m,*
- *node n_0 corresponds to vector $\boldsymbol{v}$, each node n_i for $i = 1, \ldots, m$ corresponds so some vector in W,*
- *edges of G are colored with colors numbered from 1 to n (the dimension of the VAS)*
- *if an edge of G starts in n_i and ends in n_j then $i < j$,*
- *if $\boldsymbol{v} = \langle v_1, \ldots, v_n \rangle$ then for all $i = 1, \ldots, n$ there are v_i edges of color i starting in n_0,*

– *if a node n corresponds to a vector* $\boldsymbol{w} = \langle w_1, \ldots, w_n \rangle$ *and* $w_i \geq 0$ *then there are* w_i *edges of color i starting in n, and there are no edges of color i ending in n,*
– *if a node n corresponds to a vector* $\boldsymbol{w} = \langle w_1, \ldots, w_n \rangle$ *and* $w_i \leq 0$ *then there are* w_i *edges of color i ending in n, and there are no edges of color i starting in n.*

Figure 1 shows an example of a transition sequence dag for the vector addition system $\langle\langle 4, 4\rangle, \{\langle -1, -2\rangle, \langle -1, 2\rangle\}\rangle$. Edges of color 1 are drawn with solid arrows while edges of color 2 are drawn with dotted arrows. Nodes n_1, n_3 and n_4 correspond to vector $\langle -1, -2\rangle$; node n_2 corresponds to vector $\langle -1, 2\rangle$.

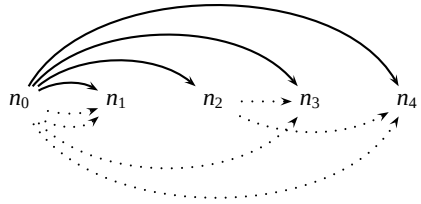

Fig. 1. Transition sequence dag for the vector addition system $\langle\langle 4, 4\rangle, \{\langle -1, -2\rangle, \langle -1, 2\rangle\}\rangle$

It is not difficult to see that the following lemma holds.

Lemma 2. *The reachability problem for a vector addition system* $\langle \boldsymbol{v}, W \rangle$ *has a solution if and only if there exists a transition sequence dag for* $\langle \boldsymbol{v}, W \rangle$.

Now we are ready to give the reduction. Let $\langle \boldsymbol{v}, W \rangle$ be an n-dimensional vector addition system. We construct a C_r^2 + Datalog formula $\varphi = \phi \wedge P$ as follows. The vocabulary Σ_E is $\{e_i(\cdot,\cdot) \mid 1 \leq i \leq n\} \cup \{e_{i,j}(\cdot,\cdot) \mid 1 \leq i \leq n \wedge 1 \leq j \leq \max_j\} \cup \{tosink(\cdot,\cdot), src, sink\}$. We use the predicate e_i for edges of color i. Additionally, to capture multiple edges of the same color starting in the same node we shall use touches of colors: the predicate $e_{i,j}$ is used for touch j of color i. Here j ranges from 1 to the maximum number occurring in a vector in $W \cup \{v\}$ as i-th component, denoted $\max_i$. The constant *src* is used to model the starting node of a transition sequence dag. To have exactly one ending node we introduce additional *sink* node and connect all nodes of outdegree 0 to it.

The vocabulary Σ_I is $\{node^{\boldsymbol{w}}(\cdot) \mid \boldsymbol{w} \in W\} \cup \{color_i(\cdot) \mid 1 \leq i \leq n\} \cup \{start(\cdot), end(\cdot)\}$.

First we give Datalog program P. It consists of the following clauses.

1. $$start(x) \leftarrow x = src \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{v_i} e_{i,j}(x, y_{i,j}) \wedge color_i(y_{i,j})$$
   where $\boldsymbol{v} = \langle v_1, \ldots, v_n \rangle$,
2. $color_i(x) \leftarrow e_i(x, y), node^{\boldsymbol{w}}(y)$ for all $\boldsymbol{w} \in W$,
3. $$node^{\boldsymbol{w}}(x) \leftarrow x \neq src \wedge \bigwedge_{k:w_k>0} \bigwedge_{j=1}^{w_k} e_{k,j}(x, y_{k,j}) \wedge color_k(y_{k,j})$$
   for all $\boldsymbol{w} \in W$ such that $\{k \mid w_k > 0\} \neq \emptyset$

4.
$$node^{\boldsymbol{w}}(x) \leftarrow x \neq src \wedge tosink(x,y), end(y)$$
for all $\boldsymbol{w} \in W$ such that $\{k \mid w_k > 0\} = \emptyset$
5. $end(x) \leftarrow x = sink$.

To define the formula ϕ we use two macro-definitions: $forbid\text{-}out(x, S)$ is an abbreviation for $\forall_y(\bigwedge_{r(\cdot,\cdot)\in S} \neg r(x,y))$ and $forbid\text{-}in(S, x)$ is $\forall_y(\bigwedge_{r(\cdot,\cdot)\in S} \neg r(y,x))$. Then ϕ is defined as the conjunction of

1. Functionality restriction and intersection freeness (expressed as in the proof of Proposition 2 in Section 3 and as in Remark 1 in Section 2 respectively),
2.
$$\forall x\ node^{\boldsymbol{w}}(x) \rightarrow forbid\text{-}out(x, \Sigma_E \setminus \{e_{i,j}(\cdot,\cdot) \mid w_i > 0 \wedge j \leq w_i\}),$$
$$\forall x\ node^{\boldsymbol{w}}(x) \rightarrow forbid\text{-}in(\Sigma_E \setminus \{e_i(\cdot,\cdot) \mid w_i < 0\}, x)$$
$$\forall x\ node^{\boldsymbol{w}}(x) \rightarrow (\bigwedge_{i:w_i<0} (\exists_y^{=-w_i} e_i(y,x)))$$
for all $\boldsymbol{w} \in W$
3. $\forall y\ color_i(y) \rightarrow \exists_x^{=1}(\bigvee_{j=1}^{\max_i}(e_{i,j}(x,y) \wedge forbid\text{-}in(\Sigma_E \setminus \{e_{i,j}\}, y)))$ and $\forall y\ color_i(x) \rightarrow forbid\text{-}out(x, \Sigma_E \setminus \{e_i\})$
4. $\forall y\ end(y) \rightarrow forbid\text{-}out(x, \Sigma_E) \wedge forbid\text{-}in(\Sigma_E \setminus \{tosink\}, y)$
5. $\forall_x start(x) \rightarrow forbid\text{-}in(\Sigma_E, x) \wedge forbid\text{-}out(x, \Sigma_E \setminus \{e_{i,j}(\cdot,\cdot) \mid v_i > 0 \wedge j \leq v_i\})$ where $\boldsymbol{v} = \langle v_1, \ldots, v_n \rangle$
6. $start(src)$.

Proposition 6. *Let $\langle \boldsymbol{v}, W \rangle$ be an n-dimensional vector addition system and let φ be the C_r^2 +* Datalog *formula constructed above. The reachability problem for $\langle \boldsymbol{v}, W \rangle$ has a solution if and only if φ is satisfiable.*

Proof (sketch). ($\Rightarrow$) Assume that $\langle \boldsymbol{v}, W \rangle$ has a solution. Let G be a transition sequence dag for $\langle \boldsymbol{v}, W \rangle$. For every edge $e = \langle n_i, n_j \rangle$ of color k in G introduce an intermediate node n_e; label the node n_e and the edge $\langle n_e, n_j \rangle$ with color k; if e was the l-th edge of color k starting in the node n_i then label the edge $\langle n_i, n_e \rangle$ with touch l of color k. Remove e from G. Add a new node *sink* to the graph and connect all nodes that have no successors in G to it via edges labeled *tosink*. Label node n_0 by constant *src*. Compute the least extension of the constructed structure wrt P. The obtained graph is a model of φ.

($\Leftarrow$) Take a model of φ. By clauses 3 and 4 each element labeled $node^{\boldsymbol{w}}$ has at least required by the vector $\boldsymbol{w}$ number of successors of each color, by conjuncts 2 in ϕ it has exactly required number of successors and predecessors. Clause 2 of P together with conjuncts 3 of ϕ ensures that all elements labeled $color_i$ are intermediate elements on edges of color i. By removing these intermediate elements and the element interpreting the *sink* constant we obtain some graph G. Since G comes from a model of a Datalog program, it is acyclic. A topological sorting of this graph shows that it is a transition sequence dag for $\langle \boldsymbol{v}, W \rangle$, so $\langle \boldsymbol{v}, W \rangle$ has a solution. □

6 Related work

Automated verification of programs manipulating dynamically allocated pointer structures is a challenging and important problem. It has received a lot of attention recently. We refer the reader to [7] for a discussion of relations between our approach and work based on abstract interpretation and shape analysis [15,34,31], fragments of first-order logic with transitive closure or least fixed point [14,32], reachability logic [1], monadic second order logic [17,24], graph logics based on C^2 [19,29], separation logic [30,4] and bounded model checking [5,16,8,12]. More recent work includes [2,33,20,3,6]. The last three papers focus on reasoning about lists and cannot express properties like reachability in a tree between a pair of program variables. In [2] the authors are able to analyze trees without sharing by simulating them by a set of converging lists. They then apply abstraction refinement framework and verify a finite-state over-approximation of the input program. In contrast, we allow bounded sharing and in our bounded-model-checking application we check properties of an under-approximation (due to a bounded execution of a program) of an infinite-state system. It seems that in terms of expressibility the logic most related to ours is the one from [33]. The exact difference in expressive needs to be investigated, but the two logics differ in terms of complexity and underlying decision procedures. The satisfiability problem for the logic in [33] has NEXPTIME lower bound and elementary upper bound and it is based on a translation to monadic second-order logic on trees (the authors say that they have another doubly-exponential procedure, but it is not published).

7 Conclusion and future work

We extended the work from [7] on bounded model checking of imperative programs that manipulate dynamically allocated pointer structures on the heap. We improved the complexity of this method and increased the expressibility of the logic. We solved the entailment problem which opens the possibilities to use the logic not only in bounded model checking but also in verification.

Our algorithms are based on a translation to two-variable logic with counting quantifiers (C^2). One may ask why we do not use directly this logic. The most important reason is that in Datalog it is relatively easy to express common data structures; the semantics based on least fixed points allows to control in a simple way (a)cyclicity of these structures. Trying to express it in C^2 leads to formulas like our translation, which is of course too complicated to be used.

There are several possibilities for future work. An obvious one is to implement the method. Another one is investigation of applications in verification, like analysis whether counterexamples generated by tools based on abstraction-refinement are spurious. Still another one is further extension of the expressibility. Although we have relaxed some restrictions from [7] (now we are able to express lists that may share some nodes or that are forbidden to share nodes), we are still not able to express DAG representations of trees. In a separate work (not published) we have developed a direct algorithm (not based on a translation to C^2) for satisfiability of Bernays-Schönfinkel class with Datalog; we believe that it will be more suitable for implementation and that it will allow us to express quantitative properties like a tree being balanced.

References

1. N. Alechina and N. Immerman. Reachability logic: An efficient fragment of transitive closure logic. *Logic Journal of IGPL*, 8:325–337, 2000.
2. Ittai Balaban, Amir Pnueli, and Lenore D. Zuck. Shape analysis of single-parent heaps. In *Proc. of the* 8^{th} *Int. Conference on Verification, Model Checking, and Abstract Interpretation*, Lect. Notes in Comp. Sci. Springer, 2007.
3. Kshitij Bansal, Rémi Brochenin, and Etienne Lozes. Beyond shapes: Lists with ordered data. In *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures: Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009*, pages 425–439, Berlin, Heidelberg, 2009. Springer-Verlag.
4. J. Berdine, C. Calcagno, and P. O'Hearn. A decidable fragment of separation logic. In *Proc. FSTTCS'04*, LNCS 3328, pages 97–109. Springer, 2004.
5. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDD. In *Proc. TACAS'99*, LNCS 1579, pages 193–207. Springer, 1999.
6. Ahmed Bouajjani, Cezara Drăgoi, Constantin Enea, and Mihaela Sighireanu. A logic-based framework for reasoning about composite data structures. In *CONCUR 2009: Proceedings of the 20th International Conference on Concurrency Theory*, pages 178–195, Berlin, Heidelberg, 2009. Springer-Verlag.
7. Witold Charatonik, Lilia Georgieva, and Patrick Maier. Bounded model checking of pointer programs. In *Proceedings of the 19th Annual Conference of the European Association for Computer Science Logic (CSL'05)*, pages 397–412, 2005.
8. E. Clarke, D. Kroening, and F. Lerda. A tool for checking ANSI-C programs. In *Proc. TACAS'04*, LNCS 2988, pages 168–176. Springer, 2004.
9. Martin Fürer. The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In *Proceedings of the Symposium "Rekursive Kombinatorik" on Logic and Machines: Decision Problems and Complexity*, pages 312–319, London, UK, 1984. Springer-Verlag.
10. Erich Grädel, Martin Otto, and Eric Rosen. Undecidability results on two-variable logics. In *STACS '97: Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, pages 249–260, London, UK, 1997. Springer-Verlag.
11. Erich Graedel, Martin Otto, and Eric Rosen. Two-variable logic with counting is decidable. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 306, Washington, DC, USA, 1997. IEEE Computer Society.
12. M. Huth and S. Pradhan. Consistent partial model checking. *Electronic Notes in Theoretical Computer Science*, 23, 2003.
13. N. Immerman, A. Rabinovich, T. Reps, M.Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *Proceedings of the 18th Annual Conference of the European Association for Computer Science Logic (CSL'04)*, pages 160–174, 2004.
14. N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. The boundary between decidability and undecidability for transitive-closure logics. In *Proc. CSL'04*, LNCS 3210, pages 160–174. Springer, 2004.
15. N. Immerman, A. Rabinovich, T. Reps, M. Sagiv, and G. Yorsh. Verification via structure simulation. In *Proc. CAV'04*, LNCS 3114, pages 281–294. Springer, 2004.
16. D. Jackson and M. Vaziri. Finding bugs with a constraint solver. In *Proc. ISSTA'00*, pages 14–25, 2000.
17. N. Klarlund and M. I. Schwartzbach. Graph types. In *Proc. POPL'93*, pages 196–205, 1993.

18. S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, pages 267–281, 1982.
19. V. Kuncak and M. Rinard. On role logic. Technical Report 925, MIT Computer Science and Artificial Intelligence Laboratory, 2003.
20. Shuvendu Lahiri and Shaz Qadeer. Back to the future: revisiting precise program verification using smt solvers. In *POPL '08: Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 171–182, New York, NY, USA, 2008. ACM.
21. Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science*, pages 4–13, 2009.
22. R. J. Lipton. The reachability problem requires exponential space. 62, New Haven, Connecticut: Yale University, Department of Computer Science, Research, Jan, 1976.
23. Ernst W. Mayr. An algorithm for the general petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.
24. A. Møller and M. I. Schwartzbach. The pointer assertion logic engine. In *Proc. PLDI'01*, pages 221–231, 2001.
25. B. O. Nas. Reachability problems in vector addition systems. *The American Mathematical Monthly*, 80(3):292–295, 1973.
26. Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity of two-variable logic with counting. In *LICS '97: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, page 318, Washington, DC, USA, 1997. IEEE Computer Society.
27. Leszek Pacholski, Wieslaw Szwast, and Lidia Tendera. Complexity results for first-order two-variable logic with counting. *SIAM J. Comput.*, 29(4):1083–1117, 2000.
28. Ian Pratt-Hartmann. Complexity of the two-variable fragment with counting quantifiers. *J. of Logic, Lang. and Inf.*, 14(3):369–395, 2005.
29. A. Rensink. Canonical graph shapes. In *Proc. ESOP'04*, LNCS 2986, pages 401–415. Springer, 2004.
30. J. Reynolds. Intuitionistic reasoning about shared mutable data structure, 1999. Proc. of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare.
31. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape-analysis problems via 3-valued logic. *ACM TOPLAS*, 24(2):217–298, 2002.
32. T. Wies. Symbolic shape analysis. Master's thesis, MPI Informatik, Saarbrücken, Germany, 2004.
33. G. Yorsh, A. Rabinovich, M. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data-structures. *Journal of Logic and Algebraic Programming*, 73(1-2):111 – 142, 2007. Foundations of Software Science and Computation Structures 2006 (FOSSACS 2006).
34. G. Yorsh, T. Reps, and M. Sagiv. Symbolically computing most-precise abstract operations for shape analysis. In *Proc. TACAS'04*, LNCS 2988, pages 530–545. Springer, 2004.

A Proof of Proposition 5

Proof. ($\Rightarrow$) Assume that φ is satisfiable. Take a structure M, such that $M \models \varphi$. We extend M to a structure M' over Σ by adding some facts. Initially $M' = M$. Conjunct 1 of $\tau(\varphi)$ is satisfied by definition. Label each constant node $e \in M'$ by $const(e)$ in order to satisfy 2. Conjuncts 3 and 4 express bounded-sharing restriction and are obviously satisfied by definition of semantics of our logic. Conjunct 5 is satisfied by definition of the least extension. Consider the set $\{p(c) \mid p \in \Sigma_I \wedge c \in \Sigma_E\} \cap T_M^P$. For each its

element $p(c)$ take arbitrary $p(c)-tree$. Let i be its height. If $i > 0$ then consider each $p(e)-subtree$ of $p(c)-tree$, where e is a non-constant. Assume $p(e)$ was derived by ground rule $[p(e) \leftarrow B \wedge \bigwedge_{j=1}^{k}(r_j(e,e_j) \wedge q_j(e_j))]$. To a newly constructed structure M' add the facts

- $p(c)\text{-}edge_{q_j}(e,e_j)$ for $1 \leq j \leq k$,
- $p(c)\text{-}node_{q_j}(e_j)$ for each $1 \leq j \leq k$, such that e_j is non-constant,
- $p(c)\text{-}leaf_{q_j}(e_j)$ for each $1 \leq j \leq k$, such that e_j is a constant.

This process leads to satisfiability of conjuncts 6, 7 and 8. In order to satisfy conjunct 9 compute the transitive closure of "being a leaf of" relation. That is saturate M' according to conjunct 9. By definition of T_M^P and $p(c)-derivation$ no $p(c)$ is its own leaf. This leads to satisfaction of conjunct 10. That is $M' \models \tau(\varphi)$ as required.

($\Leftarrow$) Assume that $\tau(\varphi)$ is satisfiable. Let $M \models \tau(\varphi)$. Let M' be a structure M stripped to vocabulary Σ_E and let M'_P be the least extension of M' with respect to P. Conjunct 5 of $\tau(\varphi)$ defines some superset of $T_{M'}^P$, so all negative Σ_I-facts true in M remain true in M'_P. To show that that $M'_P \models \phi$ it is then enough to prove equivalence that $p(c) \in T_{M'}^P$ if and only if $M \models p(c)$ for all constant literals $p(c)$ occurring in ϕ.

Again by Conjunct 5, implication $p(c) \in T_{M'}^P \Rightarrow M \models p(c)$ holds . We now show that the reverse implication is also true. Take any $p(c)$, such that $M \models p(c)$. By conjuncts 6 a derivation $deriv(p(c))$ of $p(c)$ in structure M can be constructed. While M' is identical to M on Σ_E predicates, $deriv(p(c))$ is also a correct derivation in M'. If $deriv(p(c))$ were infinite, then since M' satisfies the bounded-sharing restriction, some path of $deriv(p(c))$ would contain multiple occurrences of some fact $q(d)$ and , (with $q \in \Sigma_I$ and $d \in \Sigma_E$). But it is impossible since conjunct 10 ensures $M \models \neg q(d)\text{-}leaf_q(d)$. So $deriv(p(c))$ is finite and by Proposition 1 $p(c) \in T_{M'}^P$. □

B NEXPTIME lower bound

In this section we prove that the satisfiability problem for Bernays-Schönfinkel class with Datalog is NEXPTIME-hard. This is done by a reduction from the following version of a tiling problem. It is known [9] that this version of the tiling problem is NEXPTIME-complete.

Definition 13 (Tiling problem).
***Instance**: a tuple $\langle T, H, V, k\rangle$ where T is a finite set of tile colors, $H, V \subseteq T \times T$ are binary relations and k is a positive integer encoded in binary (on $\lceil \log k \rceil$ bits).*
***Question**: does there exist a tiling of $k \times k$ square by 1×1 tiles such that colors of each pair of horizontally adjacent tiles satisfy the relation H and colors of each pair of vertically adjacent tiles satisfy V?*

If the question in the tiling problem has a positive answer then we say that the instance $\langle T, H, V, k\rangle$ has a solution or that there exists a good tiling for this instance. In the following, for a given instance $\langle T, H, V, k\rangle$ we define a formula φ of the Bernays-Schönfinkel class with Datalog such that φ is satisfiable if and only if there exists an appropriate tiling.

The idea is that any model of the formula describes a good tiling and any such tiling defines a model of φ. The size of the formula will be polynomial in $|T| + |H| + |V| + \lceil \log k \rceil$. Using simple Datalog program and simulating two counters by $\forall\forall$ formulas it is enforced that any model of φ has $k \times k$ square as a substructure. Is is then enough to constrain pairs of adjacent nodes according to H and V. Conversely any good tiling can be labelled by Σ_E predicates to form model of φ. Below we give a detailed description.

We start by defining the vocabulary $\Sigma_I = \{element(\cdot)\}$ and $\Sigma_E = \{start, end, next(\cdot,\cdot), adjacent^v(\cdot,\cdot), adjacent^h(\cdot,\cdot)\} \cup \bigcup_{i=1}^{\lceil \log k \rceil} \{b_i^v(\cdot), b_i^h(\cdot)\} \cup \bigcup_{i=1}^{|T|} \{t_i(\cdot)\}$.

The formula φ is a conjunction $\phi \wedge P \wedge Q$ where the Datalog program P has only one slice and consists of the following two clauses.

$$element(x) \leftarrow x = end$$
$$element(x) \leftarrow next(x,y), element(y)$$

and the query Q is $element(start)$.

Before we define the formula ϕ, we need some more explanation. We shall use predicates $b_1^v(\cdot), \ldots, b_{\lceil \log k \rceil}^v(\cdot)$ (and $b_1^h(\cdot), \ldots, b_{\lceil \log k \rceil}^h(\cdot)$) in ϕ to encode in binary vertical (respectively horizontal) positions of structure nodes. These positions are numbers from $\{0, \ldots, k-1\}$, we use b_1^v and b_1^h to represent the most significant bit. Predicates $t_1(\cdot), \ldots, t_{|T|}(\cdot)$ define node's color. Program P and query Q enforce existence of a path made of *next* edges between *start* and *end* in the structure. The purpose of formula ϕ is to ensure that this path traverses a $k \times k$ square node by node, that nodes have correct horizontal and vertical positions, precisely one tile color and that horizontal (respectively, vertical) adjacent nodes satisfy H (respectively, V). To simplify the definition of ϕ we use some macro-definitions.

- Let $z^v(e)$ denote that e's vertical position is 0. Simply $z^v(x)$ is an abbreviation for $\bigwedge_{i=1}^{\lceil \log k \rceil} \neg b_i^v(x)$.
- Let $k(e)$ denote that e's vertical position is k-1. So $k^v(x)$ is an abbreviation for $\bigwedge_{i=1}^{\lceil \log k \rceil} (b_i^v(x) \leftrightarrow b_i)$, where $b_1, \ldots, b_{\lceil \log k \rceil}$ is the binary representation of $k-1$ (that is, b_i is the constant *true* if the i-th bit of $k-1$ is 1, and *false* otherwise).
- Let $lessk^v(e)$ denote that e's vertical position is less than $k-1$. So $lessk^v(x)$ is an abbreviation for
$$\bigvee_{j=1}^{\lceil \log k \rceil} ((\bigwedge_{i=1}^{j-1} (b_i^v(x) \leftrightarrow b_i) \wedge (\neg b_j^v(x) \wedge b_j)),$$
where $b_1, \ldots, b_{\lceil \log k \rceil}$ is binary representation of $k-1$. As usual empty conjunction is the constant *true*.
- Let $same^v(e_1, e_2)$ denote that both its arguments are in the same row. So $same^v(x,y)$ is an abbreviation for $\bigwedge_{i=1}^{\lceil \log k \rceil} (b_i^v(x) \leftrightarrow b_i^v(y))$.
- For each $j \in \{1, \ldots, \lceil \log k \rceil\}$ let $switch_j^v(e_1, e_2)$ denote that e_1's vertical position incremented by one gives e_2's vertical position and j is the first position of a bit that differs in both positions. Macro $switch_j^v(x,y)$ is an abbreviation for
$$\bigwedge_{i=1}^{j-1} (b_i^v(x) \leftrightarrow b_i^v(y)) \wedge (\neg b_j^v(x) \wedge b_j^v(y)) \wedge (\bigwedge_{i=j+1}^{\lceil \log k \rceil} b_i^v(x) \wedge \neg b_i^v(y).$$

– Let $neighbor^v(e_1, e_2)$ denote that e_1's vertical position is less than $k-1$ and incremented by one gives e_2's vertical position. So $neighbor^v(x,y)$ is an abbreviation for $lessk^v(x) \wedge \bigvee_{i=1}^{\lceil \log k \rceil} switch_i^v(x,y)$.
– Macros $z^h(\cdot), k^h(\cdot), same^h(\cdot,\cdot), switch_j^h(\cdot,\cdot), neighbor^h(\cdot,\cdot), lessk^h(\cdot)$ are defined analogously.

Now we are ready to define the formula ϕ. It is a conjunction of

1. $z^v(start) \wedge z^h(start) \wedge k^v(end) \wedge k^h(end)$
2. $\forall x \forall y\ adjacent^v(x,y) \leftrightarrow same^h(x,y) \wedge neighbor^v(x,y)$
3. $\forall x \forall y\ adjacent^h(x,y) \leftrightarrow same^v(x,y) \wedge neighbor^h(x,y)$
4. $\forall x \forall y\ next(x,y) \leftrightarrow adjacent^h(x,y) \vee (k^h(x) \wedge z^h(y) \wedge neighbor^v(x,y))$
5. $\forall x \bigvee_{i=1}^{|T|} t_i(x)$
6. $\forall x \bigvee_{i=1}^{|T|} t_i(x) \rightarrow (\bigwedge_{j=1}^{i-1} \neg t_i(x)) \wedge (\bigwedge_{j=i+1}^{|T|} \neg t_i(x))$
7. $\forall x \forall y\ adjacent^h(x,y) \rightarrow \bigvee_{i=1}^{m} t_{h_i}(x) \wedge t_{h'_i}(y)$, where $H = \{\langle t_{h_1}, t_{h'_1}\rangle, \ldots, \langle t_{h_m}, t_{h'_m}\rangle\}$
8. $\forall x \forall y\ adjacent^v(x,y) \rightarrow \bigvee_{i=1}^{n} t_{v_i}(x) \wedge t_{v'_i}(y)$, where $V = \{\langle t_{v_1}, t_{v'_1}\rangle, \ldots, \langle t_{v_n}, t_{v'_n}\rangle\}$.

Think of *start* as an element in the upper left corner of a square and *end* as its lower right corner. The element *start* has coordinates $\langle 0,0\rangle$ and *end* has coordinates $\langle k-1, k-1\rangle$ as defined by conjunct 1. Conjunct 2 ensures that the predicate $adjacent^v(e_1,e_2)$ holds if and only if e_1 and e_2 are in the same column and $e'_1 s$ vertical coordinate is less than $k-1$ and this coordinate incremented by one gives e'_2s vertical coordinate; that is e_1 and e_2 are vertically adjacent. Conjunct 3 does the same for the predicate $adjacent^h(x,y)$. Conjunct 4 defines the predicate $next(e_1,e_2)$ to hold if and only if

– e_1 and e_2 are horizontally adjacent or
– e_1 is the last element of some row and e_2 is the first element in next row.

Conjuncts 5 and 6 state that each node has has precisely one tile color. Conjuncts 7 and 8 enforce respectively relations H and V between horizontally and vertically adjacent elements.

Notice that because of the functionality restriction none of the binary macros can be turned into a predicate. Observe that the size of the formula φ is polynomial in $|T| + |H| + |V| + \lceil \log k \rceil$.

Lemma 3. *Let $\langle T, H, V, k\rangle$ be an instance of the tiling problem and φ be the Bernays-Schönfinkel formula with Datalog defined above. Then the instance $\langle T, H, V, k\rangle$ is solvable if and only if φ is satisfiable.*

Proof. ($\Rightarrow$) Assume that the instance $\langle T, H, V, k\rangle$ has a solution. This means that there exists a relational structure M on $k * k$ nodes $\{e_{i,j} \mid 0 \le i, j < k\}$ such that the following holds.

– For each $e_{i,j} \in M$ there exists precisely one $t \in \{t_1(\cdot), \ldots, t_{|T|}(\cdot)\}$ such that $M \models t(e_{i,j})$. That is each tile has precisely one color.
– Let $M \models t_1(e_{i,j})$ and $M \models t_2(e_{i+1,j})$ where $0 \le i < k-1$ and $0 \le j \le k-1$. Then $\langle t_1, t_2\rangle \in V$, that is, colors of vertical neighbors satisfy V.
– Let $M \models t_1(e_{i,j})$ and $M \models t_2(e_{i,j+1})$ where $0 \le j < k-1$ and $0 \le i \le k-1$. Then $\langle t_1, t_2\rangle \in H$, that is, colors of horizontal neighbors satisfy H.

We shall modify M by adding some facts to form a model of φ. First we add facts $\{next(e_{i,j}, e_{i,j+1}) \mid 0 \leq i < k-1, 0 \leq j < k\}$ and facts $\{next(e_{j,k-1}, e_{j+1,0}) \mid 0 \leq j < k-1\}$ to M. The element $e_{0,0} \in M$ interprets the constant *start* and $e_{k-1,k-1}$ interprets the constant *end*. For each $e_{i,j}$ encode j on $\lceil \log k \rceil$ bits using predicates $\{b^h_1(\cdot), \ldots, b^h_{\lceil \log k \rceil}(\cdot)\}$ and encode i on $\lceil \log k \rceil$ bits using predicates $\{b^v_1(\cdot), \ldots, b^v_{\lceil \log k \rceil}(\cdot)\}$. It can now be checked by case inspection that the macro-definitions have indeed their intended meaning and that conjuncts 1–8 are satisfied. Finally, observe that the functionality and bounded-sharing restrictions are satisfied: every element points to at most one horizontally adjacent, at most one one vertically adjacent and at most one next element; bounded-sharing is required only for the *next* predicate (the only predicate occurring in program P) and there is no no element pointed *next* by two different elements.

($\Leftarrow$) Let M be a model of φ. Let M' be a substructure of M generated by elements $e \in M$ such that e is reachable from *start* via *next* edges and *end* is reachable from e via *next* edges. By conjuncts 1, 4 and the definition of P and Q the nodes of M' are $\{e_{i,j} \mid 0 \leq i < k \wedge 0 \leq j < k\}$, where j coordinate of $e_{i,j}$ is defined by its $\{b^h_1(\cdot), \ldots, b^h_{\lceil \log k \rceil}(\cdot)\}$, and i coordinate by $\{b^v_1(\cdot), \ldots, b^v_{\lceil \log k \rceil}(\cdot)\}$ predicates. By conjuncts 6 and 5 each tile has precisely one color and by conjuncts 7 and 8 constraints H and V are preserved. □

As a direct corollary from Lemma 3 and the construction above, we get Theorem 2.

C Verification Example

While BS_2 + Datalog was designed to solve bounded model checking problem for pointer programs (see [7] for details), it can also be used to specify verification conditions of nontrivial program manipulating linked data structures. Decidability of entailment problem for this class leads to automated verification of partial correctness for such programs. We found it interesting, because: 1) our logic allows at most two variables 2) we use only universal quantifiers 3) ability of specifying reachability properties is rather limited, e.g. Σ_I predicates cannot be used in formulas in an unrestricted way, 4) satisfiability as well as entailment of our logic are $NEXPTIME$-complete via a translation to a decidable fragment of first order logic (here C^2).

We encode the verification problem (pre-, post-conditions, loop invariants) as conjunctions of BS_2 + Datalog formulas with assumptions. Assumptions state properties of additional instrumentation predicates and have the following property: each structure over vocabulary of the original problem can be extended to a structure interpreting instrumentation predicates, such that the assumptions are satisfied. Different assumptions state properties of different instrumentation predicates, so conjunction of assumptions is also an assumption. Instrumentation predicates can be Σ_E or Σ_I predicates and are denoted by primed symbols ($f'(\cdot,\cdot), g'(\cdot,\cdot), reach'(\cdot)$ etc.). To improve readability we also use macros, which are shorthands for BS_2 + Datalog formulas with assumptions and are intended to be used in conjunction with BS_2 + Datalog formulas and other macros. In the following, when we say that some predicate is fresh, we mean that each time when a macro is unfolded, a new predicate of appropriate arity is added to appropriate (Σ_E or Σ_I) vocabulary. Each Datalog program occurring in the formulas constructed below is put into separate slice. To verify that ϕ with assumption ϕ' entails ψ with assumption ψ'

it is then checked if $\phi \wedge \phi' \wedge \psi' \models \psi$. Since BS_2 + Datalog is closed under conjunction (after an appropriate renaming of vocabularies), this problem can be solved by the using results from Section 3.

C.1 Four useful macros

We start by defining some macros. The first two of them are assumptions to be used by the last two. An f-path is a path on the heap whose consecutive elements are chained by the predicate f. Note that due to the functionality restriction a maximal f-path starting in a given element is uniquely determined.

- the macro $revpath_{a,f,b,f'}$ states that for all elements e_1, e_2 of the f-path starting in a, if $f(e_1, e_2)$ and $e_1 \neq b$ holds then also $f'(e_2, e_1)$ holds. The formula $revpath_{a,f,b,f'}$ is made of conjuncts

$$\begin{array}{l} \forall_x \forall_y f'(y,x) \rightarrow f(x,y) \\ \{reach'(x) \leftarrow x = a; reach'(x) \leftarrow f'(x,y), reach'(y)\} \\ (a \neq b) \rightarrow (\forall_y \neg f'(y,b) \wedge \forall_x \neg f'(a,x)) \\ \forall_x reach'(x) \rightarrow \forall_y((x \neq b \wedge f(x,y)) \rightarrow f'(y,x)) \end{array}$$

  The predicate $reach'(\cdot)$ is fresh here. It labels all elements reachable from a via an f-path up to b (where b may but need not to occur on the path from a).
- the macro $samepath_{a,f,b,g'}$ states that for all elements e_1, e_2 of the f-path starting in a, if $f(e_1, e_2)$ and $e_1 \neq b$ holds then also $g'(e_1, e_2)$ holds. It consists of all conjuncts of the macro $revpath_{a,f,b,f'}$ in addition to the conjunct

$$\forall_x reach'(x) \rightarrow \forall_y(g'(y,x) \iff f'(x,y)).$$

- the macro $inverse_{a,f,b,g}$ states that there is a doubly linked list between a and b with forward edge f and backward edge g. More formally: constant b is reachable from constant a by an f-path and for all elements e_1, e_2 on this path such that $e_1 \neq b$, if $f(e_1, e_2)$, then $g(e_2, e_1)$. Predicate $reach(\cdot)$ is a fresh Σ_I predicate and $f'(\cdot,\cdot)$ is a fresh Σ_E predicate.

$$\{reach(x) \leftarrow x = a; reach(x) \leftarrow f'(x,y), reach(y)\},$$

  the query is $reach(b) \wedge \forall_x reach(x) \rightarrow \forall_y(f'(y,x) \iff g(y,x))$. This formula uses the assumption $revpath_{a,f,b,f'}$.
- $same_{a,f,b,g}$ states that there is a singly linked list between a and b with forward edge f and that edges f and g on this list are parallel. Formally, b is reachable from constant a by an f-path and for all elements e_1, e_2 on this path such that $e \neq b$, if $f(e, e')$ then also $g(e, e')$. The definition of this formula is very similar to the definition of $inverse_{a,f,b,g}$. The predicate $reach(\cdot)$ is a fresh Σ_I predicate and $f'(\cdot,\cdot)$ is a fresh Σ_E predicate.

$$\{reach(x) \leftarrow x = a; reach(x) \leftarrow f'(x,y), reach(y)\},$$

  the query is $reach(b) \wedge \forall_x reach(x) \rightarrow \forall_y(f'(y,x) \iff g(x,y))$. This formula uses the assumption $revpath_{a,f,b,f'}$.

C.2 List reversal

Now we are ready to show that we are able to formulate verification conditions in our logic. The following example is inspired by an example in Section 4.2 in [33].

```
Node reverse(Node x) {
[0] Node y = NULL;
[1] while (x != NULL) {
[2]   Node t = x->next;
[3]   x->next = y;
[4]   y = x;
[5]   x = t;
[6] }
[7] return y;
```

The procedure *reverse* performs reversal of a singly linked list. Similarly to[33] we introduce new symbols to the vocabulary: for each program point $p \in \{0, \ldots 7\}$ and each program variable $v \in \{x, y, z\}$ we introduce a new constant symbol v^p, new predicate $next^p$ and new data structure $list^p$.

The precondition requires that on entry to the procedure *reverse*, at program point 0, the variable x points to an acyclic list.

$pre_{reverse} = \{list^0(u) \leftarrow u = NULL; list^0(u) \leftarrow next^0(u, v), list^0(y)\} \wedge list^0(x^0)$.

The postcondition ensures that the result is an acyclic list pointed to by y, furthermore it ensures that each edge of the original list is reversed in the returned list.

$post_{reverse} = \{list^7(u) \leftarrow u = NULL; list^7(u) \leftarrow next^7(u, v), list_7(v)\} \wedge$
$\wedge list^7(y^7) \wedge inverse_{x^0,next^0,y^7,next^7} \wedge x^0 \neq NULL \rightarrow next^7(x^0, NULL)$.

The most interesting formula is the the loop invariant. It relates the state of the heap at the beginning of each iteration to the state of the heap at the beginning of the procedure. It states that both y^1 and x^1 point to singly linked lists and that $append(reverse(y^1), x^1) = x^0$. The invariant is encoded as the formula $\varphi = \{list^1(u) \leftarrow u = NULL; list^1(u) \leftarrow next^1(u, v), list^1(v)\} \wedge list^1(x^1) \wedge list^1(y^1) \wedge same_{x^0,next^0,NULL,next'}$. The assumption of the formula φ is $revpath_{y^1,next^1,NULL,next'} \wedge next'(y^1, x^1) \wedge samepath_{x^1,next^1,NULL,next'}$. Here $next'(\cdot, \cdot)$ is a fresh instrumentation predicate. The assumption encodes $append(rev(y^1), x^1)$ as a path from *NULL* following *next′* edges. A subpath of this path should contain the original list. This is encoded by the conjunct $same_{x^0,next^0,NULL,next'}$ in φ, which states that on the path from x^0 to *NULL* the edges $next^0$ and *next′* are parallel. The following entailments must now be proven

- $pre_{reverse} \wedge y^0 = NULL \models \varphi$,
- $\varphi \wedge (x^1 \neq NULL) \wedge (y^6 = x^1) \wedge next^1(x^1, x^6) \wedge next^6(x^1, y^1) \wedge samepath_{y^1,n^1,NULL,next^6} \wedge$
  $\wedge samepath_{x^6,next^1,NULL,next^6} \models \varphi^6$ and
- $\varphi^7 \wedge x^7 = NULL \models post_{reverse}$.

Here φ^6(resp. φ^7 and φ^0) is a formula φ where all predicates and constants indexed by 1 were replaced by corresponding predicates indexed by 6 (resp. 7 and 0).