Deciding Functional Lists with Sublist Sets

Thomas Wies[1], Marco Muñiz[2], and Viktor Kuncak[3]

[1] New York University, New York, NY, USA
[2] University of Freiburg, Germany
[3] EPFL, Switzerland

Abstract. Motivated by the problem of deciding verification conditions for the verification of functional programs, we present new decision procedures for automated reasoning about functional lists. We first show how to decide in NP the satisfiability problem for logical constraints containing equality, constructor, selectors, as well as the transitive sublist relation. We then extend this class of constraints with operators to compute the set of all sublists, and the set of objects stored in a list. Finally, we support constraints on sizes of sets, which gives us the ability to compute list length as well as the number of distinct list elements. We show that the extended theory is reducible to the theory of sets with linear cardinality constraints, and therefore still in NP. This reduction enables us to combine our theory with other decidable theories that impose constraints on sets of objects, which further increases the potential of our decidability result in verification of functional and imperative software.

1 Introduction

Specifications using high-level data types, such as sets and algebraic data types have proved effective for describing the behavior of functional and imperative programs [12, 26]. Functional lists are particularly convenient and widespread in both programs and specifications. Efficient decision procedures for reasoning about lists can therefore greatly help automate software verification tasks.

Theories that allow only constructing and decomposing lists correspond to term algebras and have efficient decision procedures for quantifier-free fragments [1, 15]. However, these theories do not support list concatenation or sublists. Adding list concatenation makes the logic difficult because it subsumes the existential problem for word equations [7, 11, 17], which has been well-studied and is known to be difficult.

This motivates us to use as a starting point the logic of lists with a sublist (suffix) relation, which can express some (even if not all) of the properties expressible using list concatenation. We give an axiomatization of this theory where quantifiers can be instantiated in a complete and efficient way, following the methodology of local theory extensions [18]. Although local theory extensions have been applied to term algebras with certain recursive functions [19], they have not been applied to term algebras in the presence of the sublist operation.

The general subterm relation in term algebras was shown to be in NP using different techniques [21], without discussion of practical implementation procedures and without support for set operators. Several expressive logics of linked imperative data structures have been proposed [3, 9, 10, 23]. In these logics, variables range over graph nodes, as opposed to lists viewed as terms. In other words, the theories that we consider have an additional extensionality axiom, which ensures that no two list objects in the universe have identical tail and head. This axiom has non-trivial consequences on the set of satisfiable formulas and requires a new decision procedure. Our logic admits reasoning about set-algebraic constraints, as well as cardinalities of sublist and content sets. Note that the cardinality of the set of sublists of a list xs can be used to express the length xs. Our result is particularly interesting given that the theory of concatenation with word length function is not known to be decidable [4]. Decidable logics that allow reasoning about length of lists have been considered before [20]. However, our set-algebraic constraints are strictly more expressive and capture forms of quantification that are useful for the specification of complex properties.

Contributions. We summarize the contributions of our paper as follows:

- We give a set of local axioms for the theory of lists with sublist relation that admits an efficient implementation in the spirit of [9, 23] and can leverage general implementation methods for local theory extensions [5, 6].
- We show how to extend this theory with an operator to compute the longest common suffix of two lists. We also give local axioms that give the decision procedure for the extended logic.
- We show how to further extend the theory by defining sets of elements that correspond to all sublists of a list, and then stating set algebra and size operations on such sets. Using a characterization of the models of this theory, we establish that the theory admits a reduction to the logic BAPA of sets with cardinality constraints [8]. We obtain a decidable logic that supports reasoning about the contents of lists as well as about the number of elements in a list.

Impact on verification tools. We have found common functions in libraries of functional programming languages and interactive theorem provers that can be verified to meet a detailed specification using our logic. We discuss several examples in the paper. Moreover, the reduction to BAPA makes it possible to combine this logic with a number of other BAPA-reducible logics [16, 20, 24, 25]. Therefore, we believe that our logic will be a useful component of verification tools in the near future.

An extended version of this paper with proofs and additional material is available as a technical report [22].

2 Examples

We describe our contributions through several examples. In the first two examples we show how we use our decision procedure to verify functional correctness

```
def drop[T](n: Int, xs: List[T]): List[T] = {
  if (n ≤ 0) xs
  else xs match {
    case nil ⇒ nil
    case cons(x, ys) ⇒ drop(n−1, ys)
  }
} ensuring (zs ⇒ (n < 0 → zs = xs) ∧ (n ≥ 0 ∧ length(xs) < n → zs = nil) ∧
                 (n ≥ 0 ∧ length(xs) ≥ n → zs ⪯ xs ∧ length(zs) = length(xs) − n))
```

Fig. 1. Function drop that drops the first n elements of a list xs

$$n > 0 \ \wedge \ xs \neq nil \ \wedge \ cons(x, ys) = xs \ \wedge \ zs \preceq ys \wedge$$
$$(n - 1 \geq 0 \wedge length(ys) \geq n - 1 \ \rightarrow length(zs) = length(ys) - (n - 1)) \rightarrow$$
$$n \geq 0 \wedge length(xs) \geq n \rightarrow length(zs) = length(xs) - n$$

Fig. 2. One of the verification conditions for the function drop

of a function written in a functional programming notation similar to the Scala programming language [14]. In our third example we demonstrate the usefulness of our logic to increase the degree of automation in interactive theorem proving. Throughout this section we use the term sublist for a suffix of a list.

Example 1: dropping elements from a list. Our first example, listed in Figure 1, is the function drop of the List class in the Scala standard library (such functions also occur in standard libraries for other functional languages, such as Haskell). The function takes as input an integer number n and a parametrized functional list xs. The function returns a functional list zs which is the sublist obtained from xs after dropping the initial n elements.

The **ensuring** statement specifies the postcondition of the function (a precondition is not required). The postcondition is expressed in our logic FLS^2 of functional lists with sublist sets shown in Figure 8. We consider the third conjunct of the postcondition in detail: it states that if the input n is a positive number and smaller than the length of xs then (1) the returned list zs is a sublist of the input list xs, denoted by $zs \preceq xs$, and (2) the length of zs is equal to the length of xs discounting the n dropped elements.

Deciding verification conditions. To verify the correctness of the drop function, we generate verification conditions and use our decision procedure to decide their validity. Figure 2 shows one of the generated verification conditions, expressed in our logic of Functional Lists with Sublist Sets (FLS^2). This verification condition considers the path through the second case of the **match** expression.

The verification condition can be proved valid using the FLS^2 decision procedure presented in Section 7. The theory FLS^2 is a combination of the theory FLS and the theory of sets with linear cardinality constraints (BAPA). Our decision procedure follows the methodology of [24] that enables the combination of such set-sharing theories via reduction to BAPA. Figure 3 illustrates how this decision procedure proves subgoal G_2. We first negate the subgoal and then eliminate the

FLS fragment:
$Xs = \sigma(\mathsf{xs}) \wedge Ys = \sigma(\mathsf{ys}) \wedge Zs = \sigma(\mathsf{zs}) \wedge$
$\mathsf{xs} \neq \mathsf{nil} \wedge \mathsf{cons}(\mathsf{x}, \mathsf{ys}) = \mathsf{xs} \wedge \mathsf{zs} \preceq \mathsf{ys}$
Projection onto shared sets Xs, Ys, Zs:
$Zs \subseteq Ys \ \wedge \ Ys \subseteq Xs \ \wedge \ \mathsf{card}(Xs) > 1 \ \wedge \ \mathsf{card}(Xs) = \mathsf{card}(Ys) + 1$

BAPA fragment:
$xs\_length = \mathsf{card}(Xs) - 1 \ \wedge \ ys\_length = \mathsf{card}(Ys) - 1 \ \wedge \ zs\_length = \mathsf{card}(Zs) - 1 \wedge$
$\mathsf{n} > 0 \ \wedge \ (\mathsf{n} - 1 \geq 0 \ \wedge \ ys\_length \geq \mathsf{n} - 1 \rightarrow zs\_length = ys\_length - (\mathsf{n} - 1)) \wedge$
$\mathsf{n} \geq 0 \ \wedge \ xs\_length \geq \mathsf{n} \ \wedge \ zs\_length \neq xs\_length - \mathsf{n}$
Projection onto shared sets Xs, Ys, Zs: $\mathsf{card}(Xs) \neq \mathsf{card}(Ys) + 1$

Fig. 3. Separated conjuncts for the negated subgoal G_2 of the VC in Figure 2 with the projections onto shared sets

length function. For every list xs we encode its length length(xs) using sublist sets as follows. We introduce a set variable Xs and define it as the set of all sublists of xs: $\{l.\, l \preceq \mathsf{xs}\}$, which we denote by $\sigma(\mathsf{xs})$. We then introduce an integer variable xs_length that denotes the length of xs by defining $xs\_length = \mathsf{card}(Xs) - 1$, where $\mathsf{card}(Xs)$ denotes the cardinality of set Xs. Note that we have to subtract 1, since nil is also a sublist of xs. We then purify the resulting formula and separate it into two conjuncts for the individual fragments. These two conjuncts are shown in Figure 3. The two separated conjuncts share the set variables Xs, Ys, and Zs. After the separation the underlying decision procedure of each fragment computes a projection of the corresponding conjunct onto the shared set variables. These projections are the strongest BAPA consequences that are expressible over the shared sets in the individual fragments. After the projections have been computed, we check satisfiability of their conjunction using the BAPA decision procedure. In our example the conjunction of the two projections is unsatisfiable, which proves that G_2 is valid. In Section 7 we describe how to construct these projections onto set variables for the FLS^2 theory.

Example 2: greatest common suffix. Figure 4 shows our second example, a Scala function gcs, which takes as input two functional lists xs, ys and their corresponding lengths lxs, lys. This precondition is specified by the **require** statement. The function returns a pair (zs,lzs) such that zs is the greatest common suffix of the two input lists and lzs its length. This is captured by the postcondition. Our logic provides the operator $\mathsf{xs} \sqcap \mathsf{ys}$ that denotes the greatest common suffix of two lists xs and ys. Thus, we can directly express the desired property.

Figure 5 shows one of the verification conditions that are generated for the function gcs. This verification condition captures the case when the lists xs, ys are not empty, their lengths are equal, their head elements x, y are equal, and lz1s is equal to length(xs)-1. The verification condition can again be split into two subgoals. We focus on subgoal G_1. Figure 6 shows the separated conjuncts for this subgoal and their projections onto the shared set variables Xs, Ys, Zs, and $Z1s$. Using the BAPA decision procedure, we can again prove that the conjunction of the two projections is unsatisfiable.

```
def gcs[T](xs: List[T], lxs: Int, ys: List[T], lys: Int): (List[T], Int)
require (length(xs)=lxs ∧ length(ys)=lys) =
  (xs,ys) match {
    case (nil, _) ⇒ (nil, 0)
    case (_, nil) ⇒ (nil, 0)
    case (cons(x, x1s), cons(y, y1s)) ⇒
      if (lxs > lys) gcs(x1s, lxs−1, ys, lys)
      else if (lxs < lys) gcs(xs, lxs, y1s, lys−1)
      else {
        val (z1s, lz1s) = gcs(x1s, lxs−1, y1s, lys−1)
        if (x = y ∧ lz1s = (lxs − 1)) (cons(x, z1s), lz1s+1) else (z1s, lz1s)
      }
  } ensuring ((zs, lzs) ⇒ length(zs) = lzs ∧ zs = xs ⊓ ys)
```

Fig. 4. Function gcs that computes the greatest common suffix of two lists

$$\begin{array}{l} \mathsf{length}(\mathsf{xs}) = \mathsf{lxs} \ \wedge \ \mathsf{length}(\mathsf{ys}) = \mathsf{lys} \ \wedge \ \mathsf{xs} \neq \mathsf{nil} \ \wedge \ \mathsf{ys} \neq \mathsf{nil} \ \wedge \ \mathsf{lxs} = \mathsf{lys} \wedge \mathsf{x} = \mathsf{y} \wedge \\ \mathsf{cons}(\mathsf{x}, \mathsf{x1s}) = \mathsf{xs} \ \wedge \ \mathsf{cons}(\mathsf{y}, \mathsf{y1s}) = \mathsf{ys} \ \wedge \ \mathsf{lz1s} = \mathsf{lxs} - 1 \wedge \\ \mathsf{length}(\mathsf{z1s}) = \mathsf{lz1s} \ \wedge \ \mathsf{z1s} = \mathsf{xs1} \sqcap \mathsf{y1s} \ \wedge \ \mathsf{zs} = \mathsf{cons}(\mathsf{x}, \mathsf{z1s}) \ \wedge \ \mathsf{lzs} = \mathsf{lz1s} + 1 \ \rightarrow \\ \quad \underbrace{\mathsf{length}(\mathsf{zs}) = \mathsf{lzs}}_{G_1} \ \wedge \ \underbrace{\mathsf{zs} = \mathsf{xs} \sqcap \mathsf{ys}}_{G_2} \end{array}$$

Fig. 5. One of the verification conditions for the function gcs

Example 3: interactive theorem proving. Given a complete specification of functions such as drop and gcd in our logic, we can use our decision procedure to automatically prove more complex properties about such functions. For instance, the function drop is not just defined in the Scala standard library, but also in the theory List of the Isabelle/HOL interactive theorem prover [13]. Consider the following property of function drop:

$$n \leq m \rightarrow \tau(\mathsf{drop}(n, \mathsf{xs})) \subseteq \tau(\mathsf{drop}(m, \mathsf{xs}))$$

where the expression $\tau(\mathsf{xs})$ denotes the *content set* of a list xs, i.e., $\tau(\mathsf{xs}) = \{\mathsf{head}(l).\, l \preceq \mathsf{xs}\}$. This property corresponds to Lemma set_drop_subset_set_drop stated and proved in the Isabelle theory List. Using the postcondition of function drop to eliminate all occurrences of this function in Lemma set_drop_subset_set_drop yields the formula shown in Figure 7. This formula belongs to our logic. The proof of lemma set_drop_subset_set_drop that is presented inside the Isabelle theory is not fully automated, and involves the statement of an intermediate auxiliary lemma. Using our decision procedure, the main lemma can be proved directly and fully automatically, without requiring an auxiliary lemma. Our logic is, thus, useful to increase the degree of automation in interactive theorem proving.

FLS fragment:
$Xs = \sigma(\mathsf{xs}) \wedge Ys = \sigma(\mathsf{ys}) \wedge Zs = \sigma(\mathsf{zs}) \wedge Z1s = \sigma(\mathsf{z1s}) \wedge$
$\mathsf{cons}(\mathsf{x}, \mathsf{x1s}) = \mathsf{xs} \wedge \mathsf{cons}(\mathsf{y}, \mathsf{y1s}) = \mathsf{ys} \wedge \mathsf{xs} \neq \mathsf{nil} \wedge \mathsf{x} = \mathsf{y} \wedge \mathsf{ys} \neq \mathsf{nil} \wedge$
$\mathsf{z1s} = \mathsf{x1s} \sqcap \mathsf{y1s} \wedge \mathsf{zs} = \mathsf{cons}(\mathsf{x},\mathsf{z1s})$
Projection onto shared sets Xs, Ys, Zs, $Z1s$:
$\mathsf{card}(Xs) > 1 \wedge \mathsf{card}(Ys) > 1 \wedge \mathsf{card}(Zs) > 1 \wedge$
$Z1s \subseteq Zs \wedge \mathsf{card}(Zs) = \mathsf{card}(Z1s) + 1 \wedge$
$((\mathsf{card}(Z1s) = \mathsf{card}(Xs) - 1 \vee \mathsf{card}(Z1s) = \mathsf{card}(Ys) - 1) \rightarrow Zs = Xs = Ys)$

BAPA fragment:
$xs\_length = \mathsf{card}(Xs) - 1 \wedge ys\_length = \mathsf{card}(Ys) - 1 \wedge$
$zs\_length = \mathsf{card}(Zs) - 1 \wedge z1s\_length = \mathsf{card}(Z1s) - 1 \wedge$
$xs\_length = \mathsf{lxs} \wedge ys\_length = \mathsf{lys} \wedge z1s\_length = \mathsf{lz1s} \wedge$
$\mathsf{lxs} = \mathsf{lys} \wedge \mathsf{lz1s} = \mathsf{lxs} - 1 \wedge \mathsf{lzs} = \mathsf{lz1s} + 1 \wedge zs\_length \neq \mathsf{lzs}$
Projection onto shared sets Xs, Ys, Zs, $Z1s$:
$\mathsf{card}(Z1s) = \mathsf{card}(Xs) - 1 \wedge \mathsf{card}(Z1s) = \mathsf{card}(Ys) - 1 \wedge \mathsf{card}(Zs) \neq \mathsf{card}(Z1s) + 1$

Fig. 6. Separated conjuncts for the negated subgoal G_1 of the VC in Figure 5 with projections onto the shared sets

$(n < 0 \rightarrow \mathsf{zs}_n = \mathsf{xs}) \wedge (n \geq 0 \wedge \mathsf{length}(\mathsf{xs}) < n \rightarrow \mathsf{zs}_n = \mathsf{nil}) \wedge$
$(n \geq 0 \wedge \mathsf{length}(\mathsf{xs}) \geq n \rightarrow \mathsf{zs}_n \preceq \mathsf{xs} \wedge \mathsf{length}(\mathsf{zs}_n) = \mathsf{length}(\mathsf{xs}) - n) \wedge$
$(m < 0 \rightarrow \mathsf{zs}_m = \mathsf{xs}) \wedge (m \geq 0 \wedge \mathsf{length}(\mathsf{xs}) < m \rightarrow \mathsf{zs}_m = \mathsf{nil}) \wedge$
$(m \geq 0 \wedge \mathsf{length}(\mathsf{xs}) \geq m \rightarrow \mathsf{zs}_m \preceq \mathsf{xs} \wedge \mathsf{length}(\mathsf{zs}_m) = \mathsf{length}(\mathsf{xs}) - m) \wedge$
$n \leq m \rightarrow \tau(\mathsf{zs}_n) \subseteq \tau(\mathsf{zs}_m)$

Fig. 7. Lemma set_drop_subset_set_drop expressed in our logic

3 Logic FLS^2 of Functional Lists with Sublists Sets

The grammar of our logic of functional lists with sublist sets is shown in Figure 8. It supports reasoning about lists built from list constructors and selectors, sublists, the length of lists, and cardinality and set algebraic constraints over the sets of sublists of lists $\sigma(l)$ as well their content sets $\tau(l)$.

The remainder of the paper is structured as follows. In Section 5 we first present the fragment FLS of the logic FLS^2 that allows formulas over lists and sublists, but not sets, cardinalities, or length constraints. We then formally define the semantics of the logic FLS and give a decision procedure for its satisfiability problem in Section 6. Finally, in Section 7 we show how to use this decision procedure for a BAPA reduction that decides the full logic FLS^2.

4 Preliminaries

In the following, we define the syntax and semantics of formulas. We also review the notions of partial structures and local theory extensions from [18].

Sorted logic. We present our problem in sorted logic with equality. A *signature* Σ is a tuple (S, Ω), where S is a countable set of sorts and Ω is a countable set of

$$
\begin{aligned}
F &::= A_L \mid A_S \mid F_1 \wedge F_2 \mid F_1 \vee F_2 \mid \neg F \\
A_L &::= T_L \preceq T_L \mid T_L = T_L \mid T_H = T_H \\
T_L &::= v_L \mid \mathsf{nil} \mid \mathsf{cons}(T_H, T_L) \mid \mathsf{tail}(T_L) \mid T_L \sqcap T_L \\
T_H &::= v_H \mid \mathsf{head}(T_L) \\
A_S &::= B_L = B_L \mid B_L \subseteq B_L \mid T_I = T_I \mid T_I < T_I \\
B_L &::= s_L \mid \emptyset \mid \{T_L\} \mid \sigma(T_L) \mid B_L \cup B_L \mid B_L \setminus B_L \\
B_H &::= s_H \mid \emptyset \mid \{T_H\} \mid \tau(T_L) \mid \mathsf{head}[B_L] \mid B_H \cup B_H \mid B_H \setminus B_H \\
T_I &::= v_I \mid K \mid T_I + T_I \mid K \cdot T_I \mid \mathsf{card}(B_L) \mid \mathsf{card}(B_H) \mid \mathsf{length}(T_L) \\
K &::= \ldots -2 \mid -1 \mid 0 \mid 1 \mid 2 \ldots
\end{aligned}
$$

Fig. 8. Logic FLS^2 of lists, sublist, sublist sets, list contents, and size constraints

function symbols f with associated arity $n \geq 0$ and associated sort $s_1 \times \cdots \times s_n \to s_0$ with $s_i \in S$ for all $i \leq n$. Function symbols of arity 0 are called *constant symbols*. We assume that all signatures contain the sort bool and for every other sort $s \in S$ a dedicated equality symbol $=_s \in \Omega$ of sort $s \times s \to \mathsf{bool}$. Note that we generally treat predicate symbols of sort $s_1, \ldots, s_n$ as function symbols of sort $s_1 \times \ldots \times s_n \to \mathsf{bool}$. Terms are built as usual from the function symbols in Ω and (sorted) variables taken from a countably infinite set X that is disjoint from Ω. A term t is said to be *ground*, if no variable appears in t. We denote by $\mathsf{Terms}(\Sigma)$ the set of all ground Σ-terms.

A Σ-*atom* A is a Σ-term of sort bool. We use infix notation for atoms built from the equality symbol. A Σ-*formula* F is defined via structural recursion as either one of A, $\neg F_1$, $F_1 \wedge F_2$, or $\forall x : s.F_1$, where A is a Σ-atom, F_1 and F_2 are Σ-formulas, and $x \in X$ is a variable of sort $s \in S$. We typically drop the sort annotation (both for quantified variables and the equality symbols) if this does not cause any ambiguity. We use syntactic sugar for Boolean constants (true, false), disjunctions ($F_1 \vee F_2$), implications ($F_1 \to F_2$), and existential quantification ($\exists x.F_1$). We define literals and clauses as usual. A clause C is called *flat* if no term that occurs in C below a predicate symbol or the symbol $=$ contains nested function symbols. A clause C is called *linear* if (i) whenever a variable occurs in two non-variable terms in C that do not start with a predicate or the equality symbol, the two terms are identical, and if (ii) no such term contains two occurrences of the same variable.

Total and partial structures. Given a signature $\Sigma = (S, \Omega)$, a *partial* Σ-*structure* α is a function that maps each sort $s \in S$ to a non-empty set $\alpha(s)$ and each function symbol $f \in \Omega$ of sort $s_1 \times \cdots \times s_n \to s_0$ to a partial function $\alpha(f) : \alpha(s_1) \times \cdots \times \alpha(s_n) \rightharpoonup \alpha(s_0)$. If α is understood, we write just t instead of $\alpha(t)$ whenever this is not ambiguous. We assume that all partial structures interpret the sort bool by the two-element set of Booleans $\{0, 1\}$. We further assume that all structures α interpret the symbol $=_s$ by the equality relation on $\alpha(s)$. A partial structure α is called *total structure* or simply *structure* if it interprets all function symbols by total functions. For a Σ-structure α where

Σ extends a signature Σ_0 with additional sorts and function symbols, we write $\alpha|_{\Sigma_0}$ for the Σ_0-structure obtained by restricting α to Σ_0.

Given a total structure α and a *variable assignment* $\beta : X \to \alpha(S)$, the evaluation $[\![t]\!]_{\alpha,\beta}$ of a term t in α, β is defined as usual. For a ground term t we typically write just $[\![t]\!]_\alpha$. A quantified variable of sort s ranges over all elements of $\alpha(s)$. From the interpretation of terms the notions of satisfiability, validity, and entailment of atoms, formulas, clauses, and sets of clauses in total structures are derived as usual. In particular, we use the standard interpretations for propositional connectives of classical logic. We write $\alpha, \beta \models F$ if α satisfies F under β where F is a formula, a clause, or a set of clauses. Similarly, we write $\alpha \models F$ if F is valid in α. In this case we also call α a *model* of F. The interpretation $[\![t]\!]_{\alpha,\beta}$ of a term t in a partial structure α is as for total structures, except that if $t = f(t_1, \ldots, t_n)$ for $f \in \Omega$ then $[\![t]\!]_{\alpha,\beta}$ is undefined if either $[\![t_i]\!]_{\alpha,\beta}$ is undefined for some i, or $([\![t_1]\!]_{\alpha,\beta}, \ldots, [\![t_n]\!]_{\alpha,\beta})$ is not in the domain of $\alpha(f)$. We say that a partial structure α *weakly satisfies* a literal L under β, written $\alpha, \beta \models_w L$, if (i) L is an atom A and either $[\![A]\!]_{\alpha,\beta} = 1$ or $[\![A]\!]_{\alpha,\beta}$ is undefined, or (ii) L is a negated atom $\neg A$ and either $[\![A]\!]_{\alpha,\beta} = 0$ or $[\![A]\!]_{\alpha,\beta}$ is undefined. The notion of weak satisfiability is extended to clauses and sets of clauses as for total structures. A clause C (respectively, a set of clauses) is *weakly valid* in a partial structure α if α weakly satisfies α for all variable assignments β. We then call α a *weak partial model* of C.

Theories and local theory extensions. A *theory* $\mathcal{T}$ for a signature Σ is simply a set of Σ-formulas. We consider theories $\mathcal{T}(\mathcal{M})$ defined as a set of Σ-formulas that are valid in a given set of models $\mathcal{M}$, as well as theories $\mathcal{T}(\mathcal{K})$ defined as a set of Σ-formulas that are consequences of a given set of formulas $\mathcal{K}$. In the latter case, we call $\mathcal{K}$ the *axioms* of the theory $\mathcal{T}(\mathcal{K})$ and we often identify $\mathcal{K}$ and $\mathcal{T}(\mathcal{K})$.

In what follows, we consider theories that are defined by a set of axioms. Let $\Sigma_0 = (S, \Omega_0)$ be a signature and assume that signature $\Sigma_1 = (S, \Omega_0 \cup \Omega_1)$ extends Σ_0 by new function symbols Ω_1. We call the function symbols in Ω_1 *extension symbols* and terms starting with extension symbols *extension terms*. Now, a theory $\mathcal{T}_1$ over Σ_1 is an *extension* of a theory $\mathcal{T}_0$ over Σ_0, if $\mathcal{T}_1$ is obtained from $\mathcal{T}_0$ by adding a set of (universally quantified) clauses $\mathcal{K}$. In the following, when we refer to a set of ground clauses G, we assume they are over the signature $\Sigma_1^c = (S, \Omega_0 \cup \Omega_1 \cup \Omega_c)$ where Ω_c is a set of new constant symbols. Let $\mathcal{K}$ be a set of (universally quantified) clauses. We denote by $\mathsf{st}(\mathcal{K}, G)$ the set of all ground subterms that appear in $\mathcal{K}$ or G and by $\mathcal{K}[G]$ the set of all instantiations of clauses in $\mathcal{K}$ where variables appearing below extension terms have been instantiated by the terms in $\mathsf{st}(\mathcal{K}, G)$. Then an extension $\mathcal{T}_1 = \mathcal{T}_0 \cup \mathcal{K}$ is a *local extension* if it satisfies condition (Loc):

(Loc) For every finite set of ground clauses G, $G \cup \mathcal{T}_1 \models \mathsf{false}$ iff there is no partial Σ_1^c-structure α such that $\alpha_{|\Sigma_0}$ is a total model of $\mathcal{T}_0$, all terms in $\mathsf{st}(\mathcal{K}, G)$ are defined in α, and α weakly satisfies $\mathcal{K}[G] \cup G$.

$$\begin{array}{lll} \mathsf{nil} : \mathsf{list} & \mathsf{tail} : \mathsf{list} \to \mathsf{list} & \preceq : \mathsf{list} \times \mathsf{list} \to \mathsf{bool} \\ \mathsf{cons} : \mathsf{data} \times \mathsf{list} \to \mathsf{list} & \mathsf{head} : \mathsf{list} \to \mathsf{data} & \sqcap : \mathsf{list} \times \mathsf{list} \to \mathsf{list} \end{array}$$

Fig. 9. Sorts of function symbols in the signature Σ_{FLS}

$$\begin{array}{rcl} \alpha_{\mathsf{FLS}}(\mathsf{list}) & \stackrel{\text{def}}{=} & \mathsf{L} \stackrel{\text{def}}{=} \{\, t \in \mathsf{Terms}(\Sigma_{\mathsf{L}}) \mid t : \mathsf{list} \,\} \\ \alpha_{\mathsf{FLS}}(\mathsf{data}) & \stackrel{\text{def}}{=} & \mathsf{D} \stackrel{\text{def}}{=} \{\, t \in \mathsf{Terms}(\Sigma_{\mathsf{L}}) \mid t : \mathsf{data} \,\} \\ \alpha_{\mathsf{FLS}}(\mathsf{cons}) & \stackrel{\text{def}}{=} & \mathsf{cons}_{\mathsf{L}} \stackrel{\text{def}}{=} \lambda(d, l).\, \mathsf{cons}(d, l) \\ \alpha_{\mathsf{FLS}}(\mathsf{nil}) & \stackrel{\text{def}}{=} & \mathsf{nil} \\ \alpha_{\mathsf{FLS}}(\mathsf{tail}) & \stackrel{\text{def}}{=} & \mathsf{tail}_{\mathsf{L}} \stackrel{\text{def}}{=} \lambda l.\, \text{if } l = \mathsf{nil} \text{ then } \mathsf{nil} \text{ else } l' \text{ where } l = \mathsf{cons}(d, l') \\ \alpha_{\mathsf{FLS}}(\mathsf{head}) & \stackrel{\text{def}}{=} & \mathsf{head}_{\mathsf{L}} \stackrel{\text{def}}{=} \lambda l.\, \text{if } l = \mathsf{nil} \text{ then } d_1 \text{ else } d \text{ where } l = \mathsf{cons}(d, l') \\ \alpha_{\mathsf{FLS}}(\preceq) & \stackrel{\text{def}}{=} & \lambda(l_1, l_2).\, l_1 \preceq_{\mathsf{L}} l_2 \\ \alpha_{\mathsf{FLS}}(\sqcap) & \stackrel{\text{def}}{=} & \lambda(l_1, l_2).\, l_1 \sqcap_{\mathsf{L}} l_2 \end{array}$$

Fig. 10. The canonical model α_{FLS} of functional lists with sublists

5 Logic FLS of Functional Lists with Sublists

We now define the logic of functional lists with sublists (FLS) and its accompanying theory. The logic FLS is given by all quantifier-free formulas over the signature $\Sigma_{\mathsf{FLS}} = (S_{\mathsf{FLS}}, \Omega_{\mathsf{FLS}})$. The signature Σ_{FLS} consists of sorts $S_{\mathsf{FLS}} = \{\mathsf{bool}, \mathsf{list}, \mathsf{data}\}$ and function symbols $\Omega_{\mathsf{FLS}} = \{\mathsf{nil}, \mathsf{cons}, \mathsf{head}, \mathsf{tail}, \sqcap, \preceq\}$. The sorts of the function symbols in Ω_{FLS} are shown in Figure 9. We use infix notation for the symbols $\sqcap$ and $\preceq$.

The theory of functional lists with sublist relationship $\mathcal{T}_{\mathsf{FLS}}$ is the set of all formulas in FLS that are true in the canonical model of lists. We denote this canonical model by α_{FLS}. The structure α_{FLS} is the term algebra generated by the signature $\Sigma_{\mathsf{L}} = (S_{\mathsf{FLS}}, \{\mathsf{cons}, \mathsf{nil}, d_1, d_2, \dots\})$, where $d_1, d_2, \dots$ are infinitely many constant symbols of sort data. The complete definition of α_{FLS} is given in Figure 10. The canonical model interprets the sort list as the set of all Σ_{L}-terms of sort list. We denote this set by L. Likewise, the sort data is interpreted as the set of all Σ_{L}-terms of sort data. We denote this set by D. The function symbols cons and nil are interpreted as the corresponding term constructors. The function symbols head and tail are interpreted as the appropriate selectors $\mathsf{head}_{\mathsf{L}}$ and $\mathsf{tail}_{\mathsf{L}}$. The predicate symbol $\preceq$ is interpreted as the sublist relation $\preceq_L \subseteq \mathsf{L} \times \mathsf{L}$ on lists. The sublist relation is defined as the inverse of the reflexive transitive closure of the tail selector function:

$$l_1 \preceq_{\mathsf{L}} l_2 \iff^{\text{def}} (l_2, l_1) \in \{\, (l, \mathsf{tail}_{\mathsf{L}}(l)) \mid l \in \mathsf{L} \,\}^*$$

The relation $\preceq_{\mathsf{L}}$ is a partial order on lists. In fact, it induces a meet-semilattice on the set L. We denote by $\sqcap_{\mathsf{L}}$ the meet operator of this semilattice. Given two lists l_1 and l_2, the list $l_1 \sqcap_{\mathsf{L}} l_2$ denotes the greatest common suffix of l_1 and l_2. The structure α_{FLS} interprets the function symbol $\sqcap$ as the operator $\sqcap_{\mathsf{L}}$.

We further define the theory of all finite substructures of α_{FLS}. Let Σ_{FLSf} be the signature Σ_{FLS} without the function symbol cons and let α_{FLSf} be the

structure α_{FLS} restricted to the signature Σ_{FLSf}. We call a finite subset L of L sublist closed if for all $l \in L$, $l' \in \mathsf{L}$, $l' \preceq_{\mathsf{L}} l$ implies $l' \in L$. For a finite sublist closed subset L of L, the structure α_L is the finite total substructure of α_{L} induced by the restricted support sets $\alpha_L(\mathsf{list}) \stackrel{\text{def}}{=} L$ and $\alpha_L(\mathsf{data}) \stackrel{\text{def}}{=} \{\, \mathsf{head}_{\mathsf{L}}(l) \mid l \in L \,\}$. We denote by $\mathcal{M}_{\mathsf{FLSf}}$ the set of all such finite total substructures α_L of α_{FLSf}. The theory $\mathcal{T}_{\mathsf{FLSf}}$ is the set of all FLS formulas that are true in all structures $\mathcal{M}_{\mathsf{FLSf}}$.

6 Decision Procedure for FLS

In the following, we show that the theory $\mathcal{T}_{\mathsf{FLS}}$ is decidable. For this purpose we reduce the decision problem for $\mathcal{T}_{\mathsf{FLS}}$ to the decision problem of the theory $\mathcal{T}_{\mathsf{FLSf}}$. We then give a finite first-order axiomatization of the theory $\mathcal{T}_{\mathsf{FLSf}}$ and show that it is a local theory extension. In total, this implies that deciding satisfiability of a ground formula F with respect to the theory $\mathcal{T}_{\mathsf{FLS}}$ can be reduced to deciding satisfiability of F conjoined with finitely many ground instances of the first-order axioms of $\mathcal{T}_{\mathsf{FLSf}}$.

Reducing FLS to FLSf. We first note that satisfiability of an FLS formula F in the canonical model can be reduced to checking satisfiability in the finite substructures, if the function symbol cons does not occur in F.

Proposition 1. *Let F be a quantifier-free Σ_{FLSf}-formula. Then F is satisfiable in α_{FLS} if and only if F is satisfiable in some structure $\alpha \in \mathcal{M}_{\mathsf{FLSf}}$.*

We can now exploit the fact that, in the term algebra α_{FLS}, the constructor $\mathsf{cons}_{\mathsf{L}}$ is uniquely determined by the functions $\mathsf{head}_{\mathsf{L}}$ and $\mathsf{tail}_{\mathsf{L}}$. Let F be an FLS formula. Then we can eliminate an occurrence $F(\mathsf{cons}(t_d, t_l))$ of function symbol cons in a term of F by rewriting $F(\mathsf{cons}(t_d, t_l))$ into:

$$x \neq \mathsf{nil} \wedge \mathsf{head}(x) = t_d \wedge \mathsf{tail}(x) = t_l \wedge F(x)$$

where x is a fresh variable of sort list that does not appear elsewhere in F. Let $\mathsf{elimcons}(F)$ be the formula that results from rewriting recursively all appearances of function symbol cons in F. Clearly, in the canonical model α_{FLS}, the formulas F and $\mathsf{elimcons}(F)$ are equisatisfiable. Thus, with Proposition 1 we can conclude.

Lemma 2. *Let F be an FLS formula. Then F is satisfiable in α_{FLS} if and only if $\mathsf{elimcons}(F)$ is satisfiable in some structure $\alpha \in \mathcal{M}_{\mathsf{FLSf}}$.*

Axiomatizing FLSf. We next show that there exists a first-order axiomatization $\mathcal{K}_{\mathsf{FLSf}}$ of the theory $\mathcal{T}_{\mathsf{FLSf}}$. The axioms $\mathcal{K}_{\mathsf{FLSf}}$ are given in Figure 11. The free variables appearing in the formulas are implicitly universally quantified.

Lemma 3. *The axioms $\mathcal{K}_{\mathsf{FLSf}}$ are sound, i.e., for all $\alpha \in \mathcal{M}_{\mathsf{FLSf}}$, $\alpha \models \mathcal{K}_{\mathsf{FLSf}}$.*

$$\textsf{Pure:}\ \ \mathsf{head}(x) = \mathsf{head}(y) \wedge \mathsf{tail}(x) = \mathsf{tail}(y) \rightarrow x = y \vee x = \mathsf{nil} \vee y = \mathsf{nil}$$

$$\textsf{NoCycle1:}\ \ \mathsf{nil} \preceq x \qquad \textsf{UnfoldL:}\ \ \mathsf{tail}(x) \preceq x$$

$$\textsf{NoCycle2:}\ \ \mathsf{tail}(x) = x \rightarrow x = \mathsf{nil} \qquad \textsf{UnfoldR:}\ \ x \preceq y \rightarrow x = y \vee x \preceq \mathsf{tail}(y)$$

$$\textsf{Refl:}\ \ x \preceq x \qquad \textsf{GCS1:}\ \ x \sqcap y \preceq x$$

$$\textsf{Trans:}\ \ x \preceq y \wedge y \preceq z \rightarrow x \preceq z \qquad \textsf{GCS2:}\ \ x \sqcap y \preceq y$$

$$\textsf{AntiSym:}\ \ x \preceq y \wedge y \preceq x \rightarrow x = y \qquad \textsf{GCS3:}\ \ z \preceq x \wedge z \preceq y \rightarrow z \preceq x \sqcap y$$

$$\textsf{Total:}\ \ y \preceq x \wedge z \preceq x \rightarrow y \preceq z \vee z \preceq y$$

Fig. 11. First-order axiomatization $\mathcal{K}_{\mathsf{FLSf}}$ of the theory $\mathcal{T}_{\mathsf{FLSf}}$

As a prerequisite for proving completeness of the axioms, we next show that the finite models of the axioms $\mathcal{K}_{\mathsf{FLSf}}$ are structurally equivalent to the finite substructures of the canonical model of functional lists.

Proposition 4. *Every finite model of $\mathcal{K}_{\mathsf{FLSf}}$ is isomorphic to some structure in $\mathcal{M}_{\mathsf{FLSf}}$.*

Locality of FLSf. We will now prove that the theory $\mathcal{K}_{\mathsf{FLSf}}$ can be understood as a local theory extension and, at the same time, prove that $\mathcal{K}_{\mathsf{FLSf}}$ is a complete axiomatization of the theory $\mathcal{T}_{\mathsf{FLSf}}$.

In what follows, the signature Σ_{FLSf} is the signature of the theory extension $\mathcal{K}_{\mathsf{FLSf}}$. We also have to determine the signature Σ_0 of the base theory $\mathcal{T}_0$ by fixing the extension symbols. We treat the function symbols $\Omega_e \stackrel{\text{def}}{=} \{\mathsf{head}, \mathsf{tail}, \sqcap\}$ as extension symbols, but the sublist relation $\preceq$ as a symbol in the signature of the base theory, i.e. $\Sigma_0 \stackrel{\text{def}}{=} (S_{\mathsf{FLS}}, \{\mathsf{nil}, \preceq\})$. The base theory itself is given by the axioms that define the sublist relation, but that do not contain any of the extension symbols, i.e., $\mathcal{T}_0 \stackrel{\text{def}}{=} \{\mathsf{NoCycle1}, \mathsf{Refl}, \mathsf{Trans}, \mathsf{AntiSym}, \mathsf{Total}\}$. We further denote by $\mathcal{K}_e \stackrel{\text{def}}{=} \mathcal{K}_{\mathsf{FLSf}} \setminus \mathcal{T}_0$ the extension axioms.

We now show that $\mathcal{K}_{\mathsf{FLSf}} = \mathcal{T}_0 \cup \mathcal{K}_e$ is a local theory extension. As in the definition of local theory extensions in Section 4, for a set of ground clauses G, we denote by $\mathcal{K}_e[G]$ all instances of axioms $\mathcal{K}_e$ where the variables occurring below extension symbols Ω_e are instantiated by all ground terms $\mathsf{st}(\mathcal{K}_e, G)$ that appear in $\mathcal{K}_e$ and G. Furthermore, we denote by Σ^c_{FLSf} the signature Σ_{FLSf} extended with finitely many new constant symbols Ω_c.

Lemma 5. *For every finite set of Σ^c_{FLSf} ground clauses G, if α is a partial Σ^c_{FLSf}-structure such that $\alpha_{|\Sigma_0}$ is a total model of $\mathcal{T}_0$, all terms in $\mathsf{st}(\mathcal{K}_e, G)$ are defined in α, and α weakly satisfies $\mathcal{K}_e[G] \cup G$ then there exists a finite total Σ^c_{FLSf}-structure that satisfies $\mathcal{K}_{\mathsf{FLSf}} \cup G$.*

We sketch the proof of Lemma 5. Let α be a partial Σ^c_{FLSf}-structure as required in the lemma. We can obtain a finite partial substructure α' from α by restricting the interpretations of sorts data and list to the elements that are used in the interpretations of the ground terms $\mathsf{st}(\mathcal{K}_e, G)$. Then α' is still a total model of $\mathcal{T}_0$ and still weakly satisfies $\mathcal{K}_e[G] \cup G$, since all axioms in $\mathcal{K}_{\mathsf{FLSf}}$ are universal.

We can then complete α' to a finite total model of $\mathcal{K}_{\mathsf{FLSf}} \cup G$ as follows. First, for every $u \in \alpha'(\mathsf{list})$ where $\alpha'(\mathsf{head})$ is not defined, we can extend $\alpha'(\mathsf{data})$ by a fresh element d_u and define $\alpha'(\mathsf{head})(u) = d_u$. Now, let $u \in \alpha'(\mathsf{list})$ such that $\alpha(\mathsf{tail})$ is not defined on u. If $u = \alpha'(\mathsf{nil})$, we define $\alpha'(\mathsf{tail})(u) = u$. Otherwise, from the fact that α' satisfies axioms NoCycle1, AntiSym, and Total we can conclude that there exists a maximal element $v \in \alpha'(\mathsf{list}) \setminus \{u\}$ such that $(v, u) \in \alpha'(\preceq)$. However, we cannot simply define $\alpha'(\mathsf{tail})(u) = v$. The resulting structure would potentially violate axiom Pure. Instead, we extend $\alpha'(\mathsf{list})$ with a fresh element w and $\alpha'(\mathsf{data})$ with a fresh element d_w, and define: $\alpha'(\mathsf{head})(w) = d_w$, $\alpha'(\mathsf{tail})(w) = v$, and $\alpha'(\mathsf{tail})(u) = w$. We further extend the definition of $\alpha'(\preceq)$ for the newly added element w, as expected. The completion of $\alpha'(\sqcap)$ to a total function is then straightforward.

From Lemma 5 we can now immediately conclude that the theory $\mathcal{K}_{\mathsf{FLSf}}$ satisfies condition (Loc). Completeness of the axioms follows from Proposition 4 and Lemma 5.

Theorem 6. *$\mathcal{K}_{\mathsf{FLSf}}$ is a local theory extension of the theory $\mathcal{T}_0$.*

Theorem 7. *$\mathcal{K}_{\mathsf{FLSf}}$ is an axiomatization of the theory $\mathcal{T}_{\mathsf{FLSf}}$, i.e., $\mathcal{T}(\mathcal{K}_{\mathsf{FLSf}}) = \mathcal{T}_{\mathsf{FLSf}}$.*

Deciding FLS. We now describe the decision procedure for deciding satisfiability of FLS formulas. Given an FLS input formula F, the decision procedure proceeds as follows: (1) compute $\hat{F} = \mathsf{elimcons}(\neg F)$, replace all variables in $\hat{F}$ with fresh constant symbols, and transform the resulting formula into a set of ground clauses G; and (2) use Theorem 6 and the reduction scheme for reasoning in local theory extensions [18], to reduce the set of clauses $\mathcal{K}_{\mathsf{FLSf}} \cup G$ to an equisatisfiable formula in the Bernays-Schönfinkel-Ramsey class, which is decidable. The reduction scheme computes the set of clauses $\mathcal{T}_0 \cup \mathcal{K}_e[G] \cup G$ and then eliminates all occurrences of extension functions Ω_e in literals of clauses in this set. The resulting set of clauses contains only universally quantified variables, constants, relation symbols, and equality, i.e., it belongs to the Bernays-Schönfinkel-Ramsey class. Soundness and completeness of the decision procedure follows from Lemma 2, Theorems 6 and 7, and [18, Lemma 4].

Complexity. For formulas in the Bernays-Schönfinkel-Ramsey class that have a bounded number of universal quantifiers, the satisfiability problem is known to be NP-complete [2, page 258]. The only quantified variables appearing in the set of clauses obtained after the reduction step of the decision procedure are those that come from the axioms in $\mathcal{K}_{\mathsf{FLSf}}$, more precisely, the axioms in $\mathcal{T}_0$ and the (partial) instantiations of the axioms in $\mathcal{K}_e$. In fact, we can write the clauses for these axioms in such a way that they use exactly 3 quantified variables. Finally, from the parametric complexity considerations in [18] follows that the size of the set of clauses obtained in the final step of our decision procedure is polynomial in the size of the input formula. It follows that the satisfiability problem for FLS is decidable in NP. NP-hardness follows immediately from the fact that FLS can express arbitrary propositional formulas.

Theorem 8. *The decision problem for the theory $\mathcal{T}_{\mathsf{FLS}}$ is NP-complete.*

7 Extension with Sets of Sublists and Content Sets

We next show decidability of the logic that extends FLS with constraints on sets of sublists and the contents of lists. We do this by reducing the extended logic to constraints on sets. For this we need a normal form of formulas in our logic. To obtain this normal form, we start from partial models of FLS, but refine them further to be able to reduce them to constraints on disjoint sets. We then give a BAPA reduction [24] for each of these refined models.

Predecessor-Refined Partial Structures. Our normal form of an FLS formula F is given by a disjunction of certain partial models α of $\mathcal{K}_{\mathsf{FLSf}}$. We call these models *predecessor-refined partial models*.

Definition 9. *α is a* predecessor-refined partial *(*PRP*)* structure *if it is a partial substructure of a structure in $\mathcal{M}_{\mathsf{FLSf}}$ and the following conditions hold in α*

1. *$\preceq$ is totally defined on $\alpha(\mathsf{list})$*
2. *for all $x, y \in \alpha(\mathsf{list})$, $(x \sqcap y) \in \alpha(\mathsf{list})$. Moreover, if $x, y, (x \sqcap y)$ are three distinct elements, then there exists $x_1 \in \alpha(\mathsf{list})$ such that $x_1 \preceq x$ and $\mathsf{tail}(x_1) = (x \sqcap y)$.*
3. *for all $x, y \in \alpha(\mathsf{list})$, if $x \neq y$ and $\mathsf{tail}(x)$ and $\mathsf{tail}(y)$ are defined and equal, then both $\mathsf{head}(x)$ and $\mathsf{head}(y)$ are defined.*

With each PRP *structure α we associate the conjunction of literals that are (strongly) satisfied in α. We call this formula a* PRP *conjunction.*

Theorem 10. *Each* FLS *formula is equivalent to an existentially quantified finite disjunction of* PRP *conjunctions.*

We can compute the PRP structures for an FLS formula F by using a simple modification of the decision procedure for FLS presented in Section 6: instead of instantiating the axioms $\mathcal{K}_e$ of the theory extension only with the ground subterms $\mathsf{st}(K_e, G)$ appearing in the extension axioms K_e and the clauses G generated from F, we instantiate the axioms with a larger set of ground terms Ψ defined as follows:

$$\Psi_0 = \mathsf{st}(K_e, G) \cup \{\, t_1 \sqcap t_2 \mid t_1, t_2 \in \mathsf{st}(K_e, G) \,\}$$
$$\Psi = \Psi_0 \cup \{\, \mathsf{head}(t) \mid t \in \Psi_0 \,\} \cup \{\, \mathsf{pre}(t_1, t_2), \mathsf{tail}(\mathsf{pre}(t_1, t_2)) \mid t_1, t_2 \in \Psi_0 \,\}$$

Here pre is a fresh binary function symbol, which we introduce as a Skolem function for the existential variable x_1 in Property 2 of PRP structures, i.e., we constrain pre using the following axiom:

$$\mathsf{Pre} : \forall xy.\, x \neq y \wedge x \neq x \sqcap y \wedge y \neq x \sqcap y \rightarrow \mathsf{pre}(x, y) \preceq x \wedge \mathsf{tail}(\mathsf{pre}(x, y)) = x \sqcap y$$

The PRP structures for F are then given by the partial models of $\mathcal{T}_0 \cup (K_e \cup \{\mathsf{Pre}\})[\Psi] \cup G$ in which all terms in Ψ and $\preceq$ are totally defined. These partial models can be computed using a tool such as H-PILoT [5].

Input: a PRP structure α. **Output:** a set constraint G_α.

Step 1: Define the relation $\preceq_1$ as irreflexive transitive reduct of $\preceq$ without the tail relation. Formally, for all $x, y \in \alpha(\mathsf{list})$, define $x \preceq_1 y$ iff all of the following conditions hold: (1) $x \preceq y$, (2) $x \neq y$, (3) $\mathsf{tail}(y)$ is undefined, and (4) there is no z in $\alpha(\mathsf{list})$ such that x, y, z are distinct, $x \preceq z$, and $z \preceq y$.

Step 2: Introduce sets $S_{x,y}$ with the meaning $S_{x,y} = (\sigma(y) \setminus \sigma(x)) \setminus \{y\}$ and define $\mathsf{Segs} = \{S_{x,y} \mid x \preceq_1 y\}$.

Step 3: Generate the conjunction $\hat{G}_\alpha$ of the following constraints:

1. $\sigma(\mathsf{nil}) = \{\mathsf{nil}\}$
2. $\sigma(y) = \{y\} \cup \sigma(x)$, for each x, y such that α satisfies $\mathsf{tail}(y) = x$
3. $\sigma(y) = \{y\} \cup S_{x,y} \cup \sigma(x)$, for each x, y such that α satisfies $x \preceq_1 y$
4. $\mathsf{disjoint}((S)_{S \in \mathsf{Segs}}, (\{x\})_{x \in \alpha(\mathsf{list})})$

Step 4: Existentially quantify over all Segs variables in $\hat{G}_\alpha$. If the goal is to obtain a formula without Segs variables, replace each variable $S_{x,y}$ with $(\sigma(y) \setminus \sigma(x)) \setminus \{y\}$.

Step 5: Return the resulting formula G_α.

Fig. 12. Generation of set constraints from a PRP structure

Constraints on Sets of Sublists. Define $\sigma(y) = \{x.\ x \preceq y\}$. Our goal is to show that extending FLS with the $\sigma(\_)$ operator and the set algebra of such sets yields in a decidable logic. To this extent, we consider an FLS formula F with free variables $x_1, \ldots, x_n$ and show that the defined relation on sets $\rho = \{(\sigma(x_1), \ldots, \sigma(x_n)).\ F(x_1, \ldots, x_n)\}$ is definable as $\rho = \{(s_1, \ldots, s_n).\ G(s_1, \ldots, s_n)\}$ for some quantifier-free BAPA [8] formula G. By Theorem 10, it suffices to show this property when F is a PRP conjunction, given by some PRP structure α. Figure 12 shows the generation of set constraints from a PRP structure. By replacing each $\sigma(x)$ with a fresh set variable s_x in the resulting constraint we obtain a formula in set algebra. We can check the satisfiability of such formulas following the algorithms in [8].

Among the consequences of this reduction is NP-completeness of a logic containing atomic formulas of FLS, along with formulas $s = \sigma(x)$, set algebra expressions containing $\subseteq, \cap, \cup, \setminus, =$ on sets, and the cardinality operator $\mathsf{card}(s)$ that computes the size of the set s along with integer linear arithmetic constraints on such sizes. Because the length of the list x is equal to $\mathsf{card}(\sigma(x)) - 1$, this logic also naturally supports reasoning about list lengths. We note that such a logic can also support a large class of set comprehensions of the form $S = \{x.\ F(x, y_1, \ldots, y_n)\}$ when the atomic formulas within F are of the form $u \preceq v$ and at least one atomic formula of the form $x \preceq y_i$ occurs positively in disjunctive normal form of F. Because $\forall x.F$ is equivalent to $\mathsf{card}(\{x.\ \neg F\}) = 0$, sets give us a form of universal quantification on top of FLS.

Additional Constraints on List Content. We next extend the language of formulas to allow set constraints not only on the set of sublists $\sigma(x)$ but also on the images of such sets under the head function. We define the list content func-

Input: a PRP structure α and an image constraint C.
Output: a set constraint C_α without $\mathsf{head}[s]$ expressions
Step 1: Replace each $\tau(x)$ in C with $\mathsf{head}[\sigma(x) \setminus \{\mathsf{nil}\}]$.
Step 2: Let P_i be all sets of the form $\{x_i\}$ or S_{x_i,x_j} from Figure 12. If s is a Boolean combination of expressions of the form $\sigma(x)$, $\{x\}$, let $J(s)$ be such that $s = \bigcup_{i \in J(s)} P_i$ is the decomposition of s into disjoint sets, derived from set equalities in Figure 12. Then replace each expression $\mathsf{head}[s]$ with $\bigcup_{i \in J(s)} \mathsf{head}[P_i]$.
Step 3: Replace each $\mathsf{head}[P_i]$ with a fresh set variable Q_i and conjoin the result with the following constraints on the image sets Q_i:
1. $\mathsf{card}(Q_i) \leq \mathsf{card}(P_i)$
2. $Q_i = \emptyset \rightarrow P_i = \emptyset$
3. $Q_i \cap Q_j = \emptyset$, for each $x, y \in \alpha(\mathsf{list})$ such that $P_i = \{x\}$, $P_j = \{y\}$, $x \neq y$ and $\mathsf{tail}(x) = \mathsf{tail}(y)$.

Step 4: Existentially quantify over all Q_i and return the resulting formula C_α.

Fig. 13. Eliminating $\mathsf{head}[s]$ from image constraints by introducing additional constraints on top of Figure 12.

tion by $\tau(x) = \mathsf{head}[\sigma(x) \setminus \{\mathsf{nil}\}]$ where we define $\mathsf{head}[s] = \{\mathsf{head}(x) \mid x \in s\}$. We then obtain our full logic FLS^2 shown in Figure 8 that introduces constraints of the form $\mathsf{head}[s] = v$ on top of FLS and constraints on sets of sublists. To show decidability of this logic, we use techniques inspired by [25] to eliminate the image constraints. The elimination procedure is shown in Figure 13. We use the properties of PRP structures that the elements for which $\mathsf{tail}(x_L) = \mathsf{tail}(x_R)$ holds have defined values $\mathsf{head}(x_L)$ and $\mathsf{head}(x_R)$. This allows us to enforce sufficient conditions on sets of sublists and sets of their heads to ensure that the axiom Pure can be enforced. The elimination procedure assumes that we have $\mathsf{head}(s)$ expressions only in the cases where s is a combination of sets of the form $\sigma(x)$ and $\{x\}$, which ensures that s is a disjoint combination of polynomially many partitions. This restriction is not necessary [25], but is natural in applications and ensures the membership in NP.

8 Conclusion

We presented a new decidable logic that can express interesting properties of functional lists and has a reasonably efficient decision procedure. We showed that this decision procedure can be useful to increase the degree of automation in verification tools and interactive theorem provers.

References

1. C. Barrett, I. Shikanian, and C. Tinelli. An abstract decision procedure for satisfiability in the theory of recursive data types. *ENTCS*, 174(8):23–37, 2007.

2. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
3. A. Bouajjani, C. Dragoi, C. Enea, and M. Sighireanu. A logic-based framework for reasoning about composite data structures. In *CONCUR*, 2009.
4. C. A. Furia. What's decidable about sequences? In *ATVA*, pages 128–142, 2010.
5. C. Ihlemann and V. Sofronie-Stokkermans. System description: H-PILoT. In *CADE*, pages 131–139, 2009.
6. S. Jacobs. Incremental instance generation in local reasoning. In *CAV*, pages 368–382, 2009.
7. J. Jaffar. Minimal and complete word unification. *J. ACM*, 37(1):47–85, 1990.
8. V. Kuncak and M. Rinard. Towards efficient satisfiability checking for Boolean Algebra with Presburger Arithmetic. In *CADE-21*, 2007.
9. S. Lahiri and S. Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In *POPL*, 2008.
10. T. Lev-Ami, N. Immerman, T. Reps, M. Sagiv, S. Srivastava, and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. In *CADE-20*, 2005.
11. G. Makanin. The problem of solvability of equations in a free semigroup. *Math. USSR Sbornik*, pages 129–198, 1977. (In AMS, (1979)).
12. H. H. Nguyen, C. David, S. Qin, and W.-N. Chin. Automated verification of shape, size and bag properties via separation logic. In *VMCAI*, 2007.
13. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
14. M. Odersky, L. Spoon, and B. Venners. *Programming in Scala: a comprehensive step-by-step guide*. Artima Press, 2008.
15. D. C. Oppen. Reasoning about recursively defined data structures. In *POPL*, pages 151–157, 1978.
16. R. Piskac, P. Suter, and V. Kuncak. On decision procedures for ordered collections. Technical Report LARA-REPORT-2010-001, EPFL, 2010.
17. W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *J. ACM*, 51(3), 2004.
18. V. Sofronie-Stokkermans. Hierarchic reasoning in local theory extensions. In *CADE*, pages 219–234, 2005.
19. V. Sofronie-Stokkermans. Locality results for certain extensions of theories with bridging functions. In *CADE*, 2009.
20. P. Suter, M. Dotta, and V. Kuncak. Decision procedures for algebraic data types with abstractions. In *POPL*, 2010.
21. K. N. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *Journal of the ACM (JACM)*, 34(2):492–510, 1987.
22. T. Wies, M. Muñiz, and V. Kuncak. On deciding functional lists with sublist sets. Technical Report EPFL-REPORT-148361, EPFL, 2010. available at http://cs.nyu.edu/~wies/publ/on_deciding_functional_lists_with_sublist_sets.pdf.
23. T. Wies, M. Muñiz, and V. Kuncak. An efficient decision procedure for imperative tree data structures. In *CADE*, 2011.
24. T. Wies, R. Piskac, and V. Kuncak. Combining theories with shared set operations. In *FroCoS*, 2009.
25. K. Yessenov, V. Kuncak, and R. Piskac. Collections, cardinalities, and relations. In *VMCAI*, 2010.
26. K. Zee, V. Kuncak, and M. Rinard. Full functional verification of linked data structures. In *PLDI*, 2008.