A Theory of Singly-Linked Lists and its Extensible Decision Procedure*

Silvio Ranise
LORIA-INRIA–Lorraine & Università degli Studi di Milano

Calogero Zarba
Universität des Saarlandes

Abstract

The key to many approaches to reason about pointer-based data structures is the availability of a decision procedure to automatically discharge proof obligations in a theory encompassing data, pointers, and the reachability relation induced by pointers. So far, only approximate solutions have been proposed which abstract either the data or the reachability component. Indeed, such approximations cause a lack of precision in the verification techniques where the decision procedures are exploited.

In this paper, we consider the pointer-based data structure of singly-linked lists and define a Theory of Linked Lists *(*TLL*). The theory is expressive since it is capable of precisely expressing both data and reachability constraints, while ensuring decidability. Furthermore, its decidability problem is NP-complete. We also design a practical decision procedure for* TLL *which can be combined with a wide range of available decision procedures for theories in first-order logic.*

1. Introduction

The use of pointer-based data structures, i.e. of structures where an updatable field can be referenced from more than one point, is widespread in programming as well as in other areas of computer science. Many approaches (see, e.g., [5, 12, 2, 7, 26, 6, 20, 24, 19]) to reason about this technique have been studied since the pioneering work of Burstall [10], but the result has been methods that suffer from extreme complexity and serious difficulties to incorporate reasoning in a wealth of decidable theories over data and pointer values, such as integers.

The key to many of these approaches is the availability of a decision procedure to automatically discharge proof obligations in a theory encompassing cells, memories, and a reachability relation induced by following pointers. Since reachability is not a first-order concept, some higher-order feature must be included to precisely cope with it. So, while there exist precise and automatic techniques to reason about pointer reachability (see, e.g., [19]), little has been done to combine such techniques with available decision procedures for theories over data and pointers. As a consequence, approximate solutions have been proposed. Either the structure over data and pointers have been abstracted away so that tools to reason about reachability can be used (see, e.g., [12]), or a first-order approximation of reachability has been found (see, e.g., [26]) so that available decision procedures for the theories of pointers and data can be used. Indeed, this compromise causes a lack of precision in the verification techniques where such reasoning procedures are used. It would be very desirable to build a decision procedure capable of precisely reason about reachability while being extensible with available decision procedures for fragments of first-order logic.

In this paper, we consider the pointer-based data structure of singly-linked lists and we define a *Theory of Linked Lists* (TLL) as a class of structures of many-sorted first-order logic. The theory is quite expressive: we can reason about cells (which are pairs of data and pointers), indexed collections of cells (i.e. memory configurations or heaps), and the reachability of a certain cell from another.

We show the decidability of TLL by proving a small model property. We also prove that the decision problem of TLL is NP-complete. Then, we show that TLL satisfies the hypothesis of a recent combination schema [29] that allows us to incorporate arbitrary (decidable) theories over the elements or the pointers. Given a decision procedure for TLL, we are capable of combining it with a wide range of available decision procedures for various decidable theories in first-order logic. We are left with the problem of building a decision procedure for TLL. In fact, the decision procedure suggested by a naïve exploitation of the small model property is not practical. We view TLL as an extension of a core theory T_{Base} by constructs for reachability. In this way, we devise a reduction of the decision problem for TLL to that of T_{Base} and then we regard this theory as a combination of theories for which available decision procedures exist.

To summarize, the **contributions** of this paper are two. First, we define the decidable theory TLL to reason about singly-linked lists which is precise with respect to both el-

*This research is partially funded by the French projects ACI "GECCOO" and ANR "ARROWS."

ements or pointers and reachability. Second, we design a practical decision procedure for TLL and we show how to extend it with a wide range of available procedures for decidable fragments of first-order logic. The procedure for TLL should improve the precision of the various verification techniques requiring decision procedures for similar theories while ensuring complete automation.

Related Work. For lack of space, we only discuss works which are closely related to ours. The theory TLL is similar to that used in [25] (which is a refinement of that in [8]). The decidability of TLL should explain why Isabelle is so successful in automatically discharging most proof obligations about programs manipulating linked lists considered in [25]. The use of decidable fragments of higher-order logic to reason about pointer-based data structures have received a lot of attention (see, e.g., [19]) and provides a high degree of expressiveness for reachability. However, little is known about the combination of such logics with decidable first-order theories to reason about data and pointer values. In [5], the decidability of a logic for pointer-based data structures is proved by using a small model property. Our proof is simpler since we use basic model-theoretic arguments while [5] exploits known results on reachability in finite trees. However, TLL is limited to linked lists while the logic in [5] covers also trees and graphs. The logic in [5] can only express properties involving the structure of linked data structures and related reachability properties, while TLL considers properties depending also on the data stored in linked lists. A generalization of the logic used in [5] has been recently described in [33] but again the emphasis is on expressing complex shape constraints rather than taking into account theories on data and pointers. In the context of Separation Logic [30], a decision procedure for singly linked lists based on a small-model theorem is described in [6]. The main difference with our work is that [6] abstracts away the theories over data and pointers. In combination with predicate abstractions, [2, 7] describe decision procedures for logics which are similar to TLL. The main difference is in expressiveness: both works abstract away theories over data and the logic in [7] seem more restrictive than ours (for example, it cannot express disjointness of lists). The works in [26, 24, 20] are the most closely related to ours since they all try to combine reachability reasoning with available procedures for decidable fragments of first-order logic. The main difference is in the treatment of reachability as they only provide first-order approximations which are easy to combine with decision procedures but provide limited precision. In contrast, we develop a (full) decision procedure for TLL.

Plan of the paper. Section 2 introduces some background notions. Section 3 formally defines TLL. Section 4 states the small model property, describes how to build a practical decision procedure for TLL, and shows how to combine the procedure with decision procedures for data and pointers. Section 5 discusses some conclusions and the future work. The proofs of the key results and more programs manipulating linked lists annotated with formulae of TLL can be found in the Technical Report RI-310-06 of the Department of Computer Science, Università degli Studi di Milano, available on-line at `http://www.loria.fr/~ranise/pubs/TR-310-06-UNIMI.pdf`.

2. Formal Preliminaries

We assume the usual concepts of many-sorted first-order logic (see, e.g., [16]). A *signature* Σ is a triple (S, F, P) where S is a set of sorts, F is a set of function symbols, and P is a set of predicate symbols endowed with their arities constructed using the sorts in S. If $\Sigma_1 = (S_1, F_1, P_1)$ and $\Sigma_2 = (S_2, F_2, P_2)$ are signatures, their *union* is the signature $\Sigma_1 \cup \Sigma_2 = (S_1 \cup S_2, F_1 \cup F_2, P_1 \cup P_2)$. Given a signature Σ, we assume the standard notions of Σ-*term*, Σ-*literal*, and Σ-*formula*. A Σ-*sentence* is a Σ-formula with no free variables. A literal is *flat* if it is of the form $x = y$, $x \neq y$, $x = f(y_1, \ldots, y_n)$, $p(y_1, \ldots, y_n)$, or $\neg p(y_1, \ldots, y_n)$, where $x, y, y_1, \ldots, y_n$ are variables, f is a function symbol, and p is a predicate symbol. Flattening preserves the satisfiability of sets of literals and can be done efficiently (see, e.g., [1]). If t (φ) is a term (form, resp.), we denote with $vars_\sigma(t)$ ($vars_\sigma(\varphi)$, resp.) the set of variables of sort σ occurring in t (φ, resp.). If φ is either a term or a formula, we denote with $vars(\varphi)$ the set $\bigcup_\sigma vars_\sigma(\varphi)$. Finally, if Φ is a set of terms or a set of formulae, $vars_\sigma(\Phi)$ and $vars(\Phi)$ are defined as obvious. Below, we will identify conjunctions of formulae $\varphi_1 \wedge \cdots \wedge \varphi_n$ with the set $\{\varphi_1, \ldots, \varphi_n\}$.

We assume the usual notion of Σ-*interpretation* over a set X of variables as a map which interprets each symbols in Σ (see, e.g., [16]). A Σ-*structure* is a Σ-interpretation over an empty set of variables. A Σ-formula φ over a set X of variables is *satisfiable* if it is true in some Σ-interpretation over X. Two Σ-formulae φ and ψ over a set X of variables are *equivalent* if their truth values (in symbols, $\varphi^{\mathcal{A}}$ and $\psi^{\mathcal{A}}$) are identical (i.e. $\varphi^{\mathcal{A}} = \psi^{\mathcal{A}}$), for all Σ-interpretations over X. Let $\mathcal{A}$ be an Ω-interpretation over some set V of variables. For a signature $\Sigma \subseteq \Omega$ and a set of variables $U \subseteq V$, we denote with $\mathcal{A}^{\Sigma,U}$ the interpretation obtained from $\mathcal{A}$ by restricting it to interpret only the symbols in Σ and the variables in U. Furthermore, we let $\mathcal{A}^{\Sigma} = \mathcal{A}^{\Sigma,\emptyset}$. Two Σ-interpretations $\mathcal{A}$ and $\mathcal{B}$ are *elementary equivalent* iff for each closed Σ-formula ϕ, we have that $\mathcal{A} \models \phi$ iff $\mathcal{B} \models \phi$. With $Mod^{\Sigma}(\Phi)$, we denote the class of many-sorted Σ-structures satisfying all the formulae in the set Φ. We say that a Σ-structure $\mathcal{A}$ is $\Sigma_0 = (S_0, F_0, \emptyset)$-*term generated* (or

simply term-generated when Σ_0 is clear from the context), with $\Sigma_0 \subseteq \Sigma$, iff for each $\sigma \in S_0$ and each $a \in A_\sigma$, there exists a Σ_0-term t such that $a = t^{\mathcal{A}}$, where $t^{\mathcal{A}}$ is the interpretation of t in $\mathcal{A}$. The function symbols in Σ_0 are usually called *constructors*.

A Σ-*theory* is a pair $(\Sigma, \mathbf{A})$ where Σ is a signature and $\mathbf{A}$ is a class of Σ-structures. Given a theory $T = (\Sigma, \mathbf{A})$, a T-*interpretation* is a Σ-interpretation $\mathcal{A}$ such that $\mathcal{A}^\Sigma \in \mathbf{A}$. Given a Σ-theory T, a Σ-formula φ over a set X of variables is T-*satisfiable* if it is true in some T-interpretation over X. We write $\mathcal{A} \models_T \varphi$ when $\mathcal{A}$ is a T-interpretation satisfying φ. Given a Σ-theory T, two Σ-formulae φ and ψ over a set X of variables are T-*equivalent* if $\varphi^{\mathcal{A}} = \psi^{\mathcal{A}}$, for all T-interpretations over X. Given a Σ-theory T, the *quantifier-free satisfiability problem* of T is the problem of deciding, for each quantifier-free Σ-formula φ, whether or not φ is T-satisfiable. We can regard the free variables in φ as Skolem constants when checking for the satisfiability of the quantifier-free formula φ, since φ is equisatisfiable to its existential closure, which, in turn, is equisatisfiable to its Skolemization.

2.1 Combination schemas

Let $T_i = (\Sigma_i, \mathbf{A}_i)$ be a theory, for $i = 1, 2$. The *combination* of T_1 and T_2 is the theory $T_1 \oplus T_2 = (\Sigma, \mathbf{A})$ where $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\mathbf{A} := \{\mathcal{A} \mid \mathcal{A}^{\Sigma_1} \in \mathbf{A}_1 \text{ and } \mathcal{A}^{\Sigma_2} \in \mathbf{A}_2\}$.

If Φ_i is a set of Σ_i-sentences and $T_i = (\Sigma_i, Mod^{\Sigma_i}(\Phi_i))$, for $i = 1, 2$, with $\Sigma_1 \cap \Sigma_2 = \emptyset$, then $T_1 \oplus T_2 = (\Sigma_1 \cup \Sigma_2, Mod^{\Sigma_1 \cup \Sigma_2}(\Phi_1 \cup \Phi_2))$. (For a proof of this fact, see [29].) We recall some classes of theories which will be useful below.

Definition 1 (Finite model property) *Let $\Sigma = (S, F, P)$ be a signature, $S_0 \subseteq S$ be a set of sorts, and T be a Σ-theory; T has the* finite model property *with respect to S_0 if for every T-satisfiable quantifier-free Σ-formula φ there exists a T-interpretation $\mathcal{A}$ satisfying φ such that A_σ is finite, for each sort $\sigma \in S_0$.*

Let $\Sigma = (S, F, P)$ be a signature, $S \subseteq S_0$ be a set of sorts, and T be a Σ-theory; T is *stably infinite* with respect to S_0 if for every T-satisfiable quantifier-free Σ-formula φ there exists a T-interpretation $\mathcal{A}$ satisfying φ such that A_σ is infinite, for each sort $\sigma \in S_0$.

Definition 2 (Smoothness) *Let $\Sigma = (S, F, P)$ be a signature, $S_0 = \{\sigma_1, \ldots, \sigma_n\} \subseteq S$ be a set of sorts, and T be a Σ-theory; T is* smooth *with respect to S_0 if: (*i*) for every T-satisfiable quantifier-free Σ-formula φ, (*ii*) for every T-interpretation $\mathcal{A}$ satisfying φ, (*iii*) for every cardinal number $\kappa_1, \ldots, \kappa_n$ such that $\kappa_i \geq |A_{\sigma_i}|$, there exists a T-interpretation $\mathcal{B}$ satisfying φ such that $|B_{\sigma_i}| = \kappa_i$, for $i = 1, \ldots, n$.*

Definition 3 (Finite witnessability) *Let $\Sigma = (S, F, P)$ be a signature, $S_0 \subseteq S$ be a set of sorts, and T be a Σ-theory; T is* finitely witnessable *with respect to S_0 if there exists a computable function $witness$ that for every quantifier-free Σ-formula φ returns a quantifier-free Σ-formula $\psi = witness(\varphi)$ such that: (*i*) φ and $(\exists \bar{v})\psi$ are T-equivalent, where $\bar{v} = vars(\psi) \setminus vars(\varphi)$; (*ii*) if ψ is T-satisfiable then there exists a T-interpretation $\mathcal{A}$ satisfying ψ such that the domain A_σ interpreting the sort σ in $\mathcal{A}$ is the (finite) set $[vars_\sigma(\psi)]^{\mathcal{A}}$ of elements in $\mathcal{A}$ interpreting the variables of sort σ in ψ (in symbols, $A_\sigma = [vars_\sigma(\psi)]^{\mathcal{A}}$), for each $\sigma \in S_0$.*

Definition 4 (Politeness) *Let $\Sigma = (S, F, P)$ be a signature, $S_0 \subseteq S$ be a set of sorts, and T be a Σ-theory; T is* polite *with respect to S_0 if it is both smooth and finitely witnessable with respect to S_0.*

Let T_i be a $\Sigma_i = (S_i, F_i, P_i)$-theory, for $i = 1, 2$, and let $S = S_1 \cap S_2$. Assume that (*a*) the quantifier-free satisfiability problem of T_i is decidable, for $i = 1, 2$; (*b*) $F_1 \cap F_2 = \emptyset$ and $P_1 \cap P_2 = \emptyset$; and (*c*) T_2 is polite with respect to S. The combination method in [29] consists of four phases.

First phase: variable abstraction. Let Γ be a conjunction of $(\Sigma_1 \cup \Sigma_2)$-literals. The output of the variable abstraction phase is a conjunction $\Gamma_1 \cup \Gamma_2$ satisfying the following properties: (*a*) each literal in Γ_i is a Σ_i-literal, for $i = 1, 2$; and (*b*) $\Gamma_1 \cup \Gamma_2$ is $(T_1 \oplus T_2)$-satisfiable if and only if Γ is $(T_1 \oplus T_2)$-satisfiable. (Note that properties (a) and (b) can be effectively enforced with the help of fresh variables.) We call $\Gamma_1 \cup \Gamma_2$ a conjunction of literals in *separate* form.

Second phase: witness introduction. Let $\Gamma_1 \cup \Gamma_2$ be a conjunction of literals in separate form returned in the variable abstraction phase. In the witness introduction phase we compute $\psi_2 = witness_{T_2}(\Gamma_2)$, and we output $\Gamma_1 \cup \{\psi_2\}$. Intuitively, this phase introduces the fresh variables in $vars(\psi_2) \setminus vars(\Gamma)$, whose role is to witness that certain facts hold for the polite theory T_2.

Third phase: decomposition. Let $\Gamma_1 \cup \{\psi_2\}$ be the conjunction obtained in the witness introduction phase. Let $V_\sigma = vars_\sigma(\psi_2)$ for each $\sigma \in S$, and let $V = \bigcup_{\sigma \in S} V_\sigma$. In the decomposition phase we nondeterministically guess a family E of equivalence relations $E = \{E_\sigma \subseteq V_\sigma \times V_\sigma \mid \sigma \in S\}$. Then, we construct the *arrangement* of V induced by E, defined by

$$arr(V, E) = \{x = y \mid (x, y) \in E_\sigma \text{ and } \sigma \in S\} \cup \{x \neq y \mid (x, y) \in (V_\sigma \times V_\sigma) \setminus E_\sigma \text{ and } \sigma \in S\},$$

and we output the conjunction $\Gamma_1 \cup \{\psi_2\} \cup arr(V, E)$.

Fourth phase: check. Let $\Gamma_1 \cup \{\psi_2\} \cup arr(V, E)$ be a conjunction obtained in the decomposition phase. The check phase consists in the following steps: **1.** if $\Gamma_1 \cup arr(V, E)$ is T_1-satisfiable go to step 2; otherwise output *fail*. **2.** If $\{\psi_2\} \cup arr(V, E)$ is T_2-satisfiable go to step 3; otherwise output *fail*. **3.** output *succeed*.

The correctness of the method is proved in [29]. If we assume that the theories being combined are stably infinite and share only sort symbols, the combination schema above reduces to the (many-sorted version of the) Nelson-Oppen combination schema [31] by dropping the witness introduction phase and guessing an arrangement only over the set of shared variables.

3. The theory TLL

We formally define the theory TLL to reason about cells, memory configurations, and the reachability on pointers.

Definition 5 (The theory TLL) *Let* $\mathsf{TLL} := (\Sigma_{\mathsf{TLL}}, \mathbf{TLL})$ *where* $\Sigma_{\mathsf{TLL}} := \Sigma_{\mathsf{Cells}} \cup \Sigma_{\mathsf{Memory}} \cup \Sigma_{\mathsf{Reachability}} \cup \Sigma_{\mathsf{Sets}} \cup \Sigma_{\mathsf{Bridge}}$ *(see Figure 1) and* **TLL** *is the class of* Σ_{TLL}*-structures satisfying the conditions in Figure 2.*

The interpretations of the sort symbols elem and addr is left unspecified. So, we take $\Sigma_{\mathsf{addr}} = (\emptyset, \emptyset, \mathsf{addr})$ and $\Sigma_{\mathsf{elem}} = (\emptyset, \emptyset, \mathsf{elem})$. We will see how to overcome this limitation in Section 4.3.

The interpretation of the function symbols in Σ_{Cells} or Σ_{Memory} is standard: the former models pairs of elements (data) and addresses (pointers), which are the content of each cell in a list; the latter models memory configurations, i.e. snap-shots of the memory where the linked lists are stored. In particular, the interpretation of the symbols in $\Sigma_{\mathsf{Cells}} \cup \Sigma_{\mathsf{Memory}}$ is closely related to that proposed by Burstall [10].

Property 6 $A_{\mathsf{cell}}^{A_{\mathsf{addr}}}$ *is isomorphic to* $A_{\mathsf{elem}}^{A_{\mathsf{addr}}} \times A_{\mathsf{addr}}{}^{A_{\mathsf{addr}}}$.

So, we can regard a memory configuration as a pair of arrays indexed by addresses: the former storing elements and the latter storing addresses, which is exactly the view of memory in [10].

Paths cannot be considered as finite sequences of addresses equipped with the usual constructors (e.g., cons) and operations (e.g., concatenation). In fact, any path $[i_1, ..., i_n]$ must be such that its addresses must be non-repeating (cf. Figure 2). While this invariant is trivially satisfied for the empty and the singleton sequence (for which we provide the constructors ϵ and $[\_]$), it may be violated by the concatenation of two sequences. This is the reason why we introduce the predicate $append$, which holds when concatenating two paths sharing no address so that the result is still a path.

There are infinite and finite lists. We are interested in studying only finite lists which can be furtherly classified in *acyclic* (terminating with $null$) and *cyclic* lists, as depicted in Figure 3. We introduce the following predicates to formalize each types of finite lists and their union:

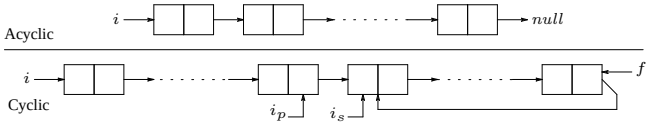

Figure 3. The two types of (finite) lists in TLL

$$\mathsf{acyclic}(m,i) \quad \Leftrightarrow \quad null \in addr2set(m,i) \tag{1}$$

$$\mathsf{cyclic}(m,i) \quad \Leftrightarrow \quad \exists i_s, f, i_p. \left(\begin{array}{ll} i_s \in addr2set(m,i) & \wedge \\ f \in addr2set(m,i) & \wedge \\ i_p \in addr2set(m,i) & \wedge \\ i_p \neq f & \wedge \\ m[i_p].next = i_s & \wedge \\ m[f].next = i_s & \end{array} \right) \tag{2}$$

$$\mathsf{isfinite}(m,i) \quad \Leftrightarrow \quad \mathsf{acyclic}(m,i) \vee \mathsf{cyclic}(m,i). \tag{3}$$

For cyclic lists, the cell pointed to by i_s (cf. Figure 3) is said to be (*heap*) *shared* since there are two next-fields (namely, that of the cells pointed to by f and by i_p) whose value is i_s. Notice also that isfinite recognizes only finite lists, in the sense that if $\mathsf{isfinite}(m,i)$ holds then there exists an integer $n \geq 0$ such that the cardinality of $addr2set(m,i)$ is equal to n.

As it is particularly useful to talk about sets of addresses, e.g., the set of addresses of the cells belonging to a given linked lists, we have included the signature Σ_{Sets} of a simple theory of sets whose elements can only be addresses. Indeed, we need bridge functions either to map paths of addresses to sets of addresses (cf. $path2set$) or to compute the set of addresses which can be reached by following a chain of next-fields from a given address in a given memory configuration (cf. $getp$ and $addr2set$). Being able to characterize sets of addresses is not only convenient but it also enables us to avoid the use of quantifiers. For example, consider the case where we want to express the fact that the lists identified by two variables i and j do not share any non-empty segment of list as depicted in Figure 4. With the help of the predicate $reach$, it is easy to see that the only way to represent the situation requires the use of quantifiers (see, e.g., [2]):

$$\begin{array}{l} \mathsf{acyclic}(m,i) \wedge \mathsf{acyclic}(m,j) \wedge \\ \forall k_i, k_j. \exists p_i, p_j. \left(\begin{array}{ll} reach(m,i,k_i,p_i) & \wedge \\ reach(m,j,k_j,p_j) & \wedge \\ k_i \neq null \wedge k_j \neq null & \end{array} \right) \Rightarrow k_i \neq k_j \end{array}$$

where k_i, k_j are variables of sort addr, and p_i, p_j are variables of sort path. Instead, by using sets of addresses, we can more compactly specify (without resorting to quantifiers) that the intersection of the two sets of addresses reachable from i and j, respectively, is the singleton set contain-

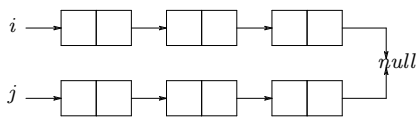

Figure 4. Two separate linked lists

Signature	Sorts	Functions	Predicates
Σ_{Cells}	cell elem	$error$: cell $mkcell$: elem, addr → cell $\_.data$: cell → elem $\_.next$: cell → addr	—
Σ_{Memory}	mem addr cell	$null$: addr $\_[\_]$: mem, addr → cell upd : mem, addr, cell → mem	—
$\Sigma_{\mathsf{Reachability}}$	mem addr path	ϵ : path $[\_]$: addr → path	$append$: path, path, path $reach$: mem, addr, addr, path
Σ_{Sets}	addr set	$\emptyset$: set $\{\_\}$: addr → set $\cup, \cap, \setminus$: set, set → set	$\in$: addr, set $\subseteq$: set, set
Σ_{Bridge}	mem addr set path	$path2set$: path → set $addr2set$: mem, addr → set $getp$: mem, addr, addr → path	—

Note: As usual, we model fields as unary functions returning the various pieces of information stored in the field of a cell. To emphasize the fact that a function f represents a field, we will write $x.f$ in place of $f(x)$, where x is a cell.

Figure 1. The signature of the theory TLL

Interpretation of sort symbols: cell, mem, set, and path	
Each sort σ in Σ_{TLL} is mapped to a non-empty set A_σ such that: (a) $A_{\mathsf{cell}} = A_{\mathsf{elem}} \times A_{\mathsf{addr}}$; (b) $A_{\mathsf{mem}} = A_{\mathsf{cell}}^{A_{\mathsf{addr}}}$; (c) A_{set} is the power-set of A_{cell}; (d) A_{path} is the set of all finite sequences of (pairwise) distinct elements of A_{cell}.	
Signature	Interpretation
Σ_{Cells}	— $mkcell^{\mathcal{A}}(e,i) = \langle e,i\rangle$, for each $e \in A_{\mathsf{elem}}$ and $i \in A_{\mathsf{addr}}$; — $\langle e,i\rangle.data^{\mathcal{A}} = e$, for each $e \in A_{\mathsf{elem}}$ and $i \in A_{\mathsf{addr}}$; — $\langle e,i\rangle.next^{\mathcal{A}} = i$, for each $e \in A_{\mathsf{elem}}$ and $i \in A_{\mathsf{addr}}$; — $error^{\mathcal{A}}.next^{\mathcal{A}} = null^{\mathcal{A}}$.
Σ_{Memory}	— $a[i]^{\mathcal{A}} = a(i)$, for each $a \in A_{\mathsf{mem}}$ and $i \in A_{\mathsf{addr}}$; — $upd^{\mathcal{A}}(a,i,u) = a_{i \mapsto u}$, for each $a \in A_{\mathsf{mem}}$, $i \in A_{\mathsf{addr}}$, and $u \in A_{\mathsf{cell}}$; — $a^{\mathcal{A}}(null^{\mathcal{A}}) = error^{\mathcal{A}}$, for each $a \in A_{\mathsf{mem}}$
$\Sigma_{\mathsf{Reachability}}$	— $\epsilon^{\mathcal{A}}$ is the empty sequence; — $[i]^{\mathcal{A}}$ is the sequence containing $i \in A_{\mathsf{addr}}$ as the only element; — $([i_1,...,i_n],[j_1,...,j_m],[i_1,...,i_n,j_1,...,j_m]) \in append^{\mathcal{A}}$ iff i_k and j_l are all distinct; — $(a,i,j,p) \in reach^{\mathcal{A}}$ iff $i = j$ and $p = \epsilon$, or there exist addresses $i_1,\ldots,i_n \in A_{\mathsf{addr}}$ such that: (*i*) $p = [i_1,\ldots,i_n]$; (*iii*) $a(i_r).next^{\mathcal{A}} = i_{r+1}$, for $1 \le r < n$; (*ii*) $i_1 = i$; (*iv*) $a(i_n).next^{\mathcal{A}} = j$.
Σ_{Sets}	The symbols $\emptyset$, $\{\_\}$, $\cup$, $\cap$, $\setminus$, $\in$, and $\subseteq$ are interpreted according to their standard interpretation over sets (of addresses).
Σ_{Bridge}	— $addr2set^{\mathcal{A}}(a,i) = \left\{ j \in A_{\mathsf{addr}} \mid \exists p \in A_{\mathsf{path}} \text{ s.t. } (a,i,j,p) \in reach^{\mathcal{A}} \right\}$; — $path2set^{\mathcal{A}}(p) = \{i_1,...,i_n\}$ for $p = [i_1,...,i_n] \in A_{\mathsf{path}}$; — $getp^{\mathcal{A}}(a,i,j) = \begin{cases} p & \text{if } (a,i,j,p) \in reach^{\mathcal{A}} \\ \epsilon & \text{otherwise} \end{cases}$ for each $a \in A_{\mathsf{mem}}$, $p \in A_{\mathsf{path}}$, and $i,j \in A_{\mathsf{addr}}$.

Note: $a_{i \mapsto u}$ abbreviates the function a' with domain A_{addr} and co-domain A_{cell} such that $a'(x) = u$, if $x = i$, and $a'(x) = a(x)$, otherwise. We denote the set of functions with domain X and co-domain Y as Y^X.

Figure 2. Characterization of a Σ_{TLL}-interpretation $\mathcal{A}$

ing $null$:

$$\mathsf{acyclic}(m,i) \wedge \mathsf{acyclic}(m,j) \wedge$$
$$addr2set(m,i) \cap addr2set(m,j) = \{null\}.$$

To further illustrate the expressiveness of TLL, we consider how TLL-formulae can be used to annotate the program for in-place list reversal of acyclic lists:[1]

```
o := null;
WHILE (!i ≠ null) DO
           t := M[!i].next;
  M[!i].next := !o;
           o := !i;
           i := !t
END
```

The mutable variable `M` stores memory configurations. The list before the execution of the program is identified by the mutable variable `i`. At any iteration of the loop, `i` points to the suffix of the initial list (still to be reversed) while the mutable variable `o` points to the reverse of the prefix of the initial list. After the execution of the program, `i` points to `null` and `o` to the reverse of the initial list. The mutable variable `t` serves only to swap pointers.

The following TLL-formulae ensure that the set of reachable cells remains the same (no *memory leakage*:

Pre-cond.:	$\mathsf{acyclic}(M,i) \wedge R_i^0 = addr2set(M,i)$
Post-cond.:	$\mathsf{acyclic}(M,o) \wedge R_i^0 = addr2set(M,o)$
Invariant:	$\mathsf{acyclic}(M,i) \wedge \mathsf{acyclic}(M,o) \wedge$
	$addr2set(M,i) \cap addr2set(M,o) = \{null\} \wedge$
	$addr2set(M,i) \cup addr2set(M,o) = R_i^0.$

The pre-condition states that `i` points to a (possibly) empty acyclic list and it defines the set R_i^0 of reachable cells in the memory configuration before the execution of the program. The post-condition requires that the set of memory locations reachable before the execution of the program (namely, R_i^0) must be the same of the one reachable afterwards. The invariant of the loop states that the lists `i` and `o` are acyclic, that the sets of reachable cells from `i` and `o` are disjoint, and that their union is equal to the set of cells which is reachable from `i` before the execution of the loop. By using a verification condition generator such as [14], it is possible to generate three quantifier-free formulae whose TLL-unsatisfiability implies the absence of memory leakage.

4. An extensible decision procedure for TLL

We design a decision procedure for the satisfiability problem of TLL. We do this in two steps. First, we prove that TLL enjoys the small-model property (cf. Definition 1) with respect to the domains of elem and addr. From this, it immediately follows the decidability of TLL by enumerating Σ_{TLL}-structures up to a certain cardinality. Indeed, such a brute force procedure is not usable in practice. The second step consists of showing how to use the small-model property to build a *usable* decision procedure for the satisfiability problem of TLL by harnessing the recent advances in first-order decision procedures.

[1] We use a dialect of ML with immutable and mutable variables as the one used in [14].

4.1 Small model property

Let Γ be a conjunction of TLL-literals. To simplify the proof of the small model property, we pre-process the set Γ of literals by performing flattening (see Section 2) and then, identifying a set of *normalized* TLL-*literals* obtained by exhaustively applying some simple syntactic transformations to a set of flat literals.

Definition 7 *A* TLL-*literal is* normalized *if it is a flat literal of the form:*

$e_1 \neq e_2\,,$	$i \neq j\,,$	
$i = null\,,$	$u = error\,,$	
$u = mkcell(e,i)\,,$	$u = rd(a,i)\,,$	$a = upd(b,i,u)\,,$
$x = \{i\}\,,$	$x = y \cup z\,,$	$x = y \setminus z\,,$
$p \neq q\,,$	$p = [i]\,,$	
$x = path2set(p)\,,$	$append(p_1,p_2,p_3)\,,$	$\neg append(p_1,p_2,p_3)\,,$
$x = addr2set(a,i)\,,$	$p = getp(a,i,j)\,,$	

where e_1, e_2, e *are* elem-*variables,* i, j *are* addr-*variables,* u, v *are* cell-*variables,* a, b *are* mem-*variables,* x, y, z *are* set-*variables, and* p, q, p_1, p_2, p_3 *are* path-*variables.*

Lemma 8 *Deciding the* TLL-*satisfiability of quantifier-free* TLL-*formulae is equivalent to checking the* TLL-*satisfiability of normalized* TLL-*literals.*

The small-model property of TLL is shown by considering an arbitrary TLL-interpretation $\mathcal{A}$ satisfying a conjunction of normalized literals Γ and reducing A_{elem} and A_{addr} to finite sets whose cardinalities are bounded by a certain polynomial in the size of the constants in Γ. This is done in two phases. First, we identify certain sub-terms of sort elem and addr in Γ that are used to identify finite sub-sets A'_{elem} and A'_{addr} of A_{elem} and A_{addr}, respectively. Second, we navigate through the pointer structure induced by $\mathcal{A}$ so that the various paths only contain addresses in A'_{addr}. At this point we have obtained a TLL-interpretation $\mathcal{A}'$ and we are left with the problem of showing that it satisfies Γ. This is done by case analysis on normalized literals.

Lemma 9 (Small model property) *Let* Γ *be a conjunction of normalized* TLL-*literals, and let* $V_\tau = vars_\tau(\Gamma)$*, for each sort* τ*. Also, let* $n = |V_{\mathsf{elem}}|$*,* $m = |V_{\mathsf{addr}}|$*,* $r = |V_{\mathsf{mem}}|$*, and* $s = |V_{\mathsf{path}}|$*. Then the following are equivalent:*

1. Γ *is* TLL-*satisfiable;*
2. Γ *is true in a* TLL-*interpretation* $\mathcal{A}$ *such that* $|A_{\mathsf{elem}}| \leq n + r \cdot |A_{\mathsf{addr}}|$ *and* $|A_{\mathsf{addr}}| \leq m + 1 + rm + s^2 + s^3$.

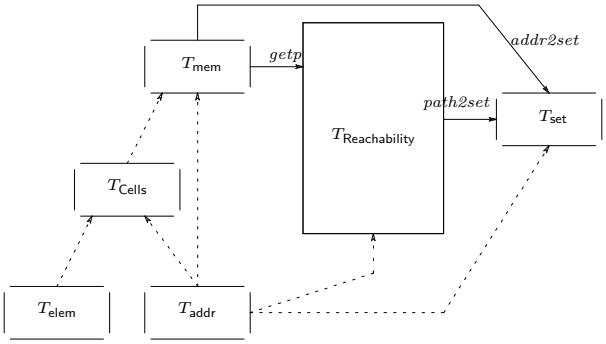

Figure 5. TLL **as a combination of theories**

The small model property yields a brute force method to check the TLL-satisfiability of normalized literals. One must enumerate all sets A_{addr} and A_{elem} with cardinalities bounded by the polynomials of Lemma 9 in search of an TLL-interpretation satisfying the conjunctions of normalized literals. With Lemma 8, this yields the decidability of the TLL-satisfiability problem.

Theorem 10 *The problem of deciding the* TLL*-satisfiability of quantifier-free* TLL*-formulae is decidable and NP-complete.*

4.2 A usable decision procedure for TLL

Our goal is to build a decision procedure for TLL by leveraging the vast literature on decision procedures for fragments of first-order logic and their combinations (see, e.g., [1] for further pointers to the literature). In this respect, it is particularly useful to regard TLL as a combination of theories, as depicted in Figure 5 (where dashed arrows denote the 'import' relation and solid arrows mark the presence of a bridge function, labelling the arrow, between two theories). Let $T_{\Sigma_x} := (\Sigma_x, \mathbf{X})$, where $x \in \{\mathsf{elem}, \mathsf{addr}, \mathsf{cell}, \mathsf{mem}, \mathsf{set}, \mathsf{Reachability}\}$ and $\mathbf{X} := \{\mathcal{A}^{\Sigma_X} \mid \mathcal{A}^{\Sigma_{\mathsf{TLL}}} \in \mathbf{TLL}\}$. More precisely, we want to combine the decision procedures for the various theories by using a many-sorted variant of the Nelson-Oppen combination schema [31]. To this end, we must show (*i*) the availability of decision procedures for all sub-theories and (*ii*) that the hypotheses (signature disjointness except for sort symbols and stably-infiniteness) for applying the combination schema are satisfied.

Requirements (*i*) and (*ii*) are unproblematic for the theories T_{elem} and T_{addr} of equality over elem and addr, respectively. There are many decision procedures available in the literature for such a theory (see, e.g., [27, 1]) and it is well-known that they are stably-infinite. It is also easy to meet requirements (*i*) and (*ii*) for the theories T_{Cells}, T_{mem}, and T_{set} (cf. Figure 5). In fact, for each of these theories, it is possible to build decision procedures by adapting existing techniques. Furthermore, it is known that the theories T_{Cells}, T_{mem}, and T_{set} are stably-infinite (see, e.g., [29]) and only share sort symbols.

The situation is more complex for $T_{\mathsf{Reachability}}$ (cf. Figure 5) for which we must solve two problems. First, there is no available decision procedure for $T_{\mathsf{Reachability}}$. Below, instead of building such a procedure, we show how to replace $T_{\mathsf{Reachability}}$ with another theory (preserving satisfiability) for which it is easier to re-use existing techniques. Second, we must transform the input set of literals to be checked for satisfiability in order to eliminate the bridge functions in Figure 5 while preserving satisfiability, so to meet requirement (*ii*) above.

Distinctness & Uniqueness:

$$\begin{array}{rcl} \forall i, l.cons(i,l) & \neq & nil \\ \forall i, l.hd(cons(i,l)) & = & i \\ \forall i, l.tl(cons(i,l)) & = & l \\ \forall l.l \neq nil \Rightarrow cons(hd(l), tl(l)) & = & l \end{array}$$

k-Acyclicity: $\forall i_1, ..., i_k, , l.cons(i_1, ..., cons(i_k, l))) \neq l$

Generatedness: $\forall l.(l = nil \vee \exists i, l'.l = cons(i, l'))$

Note: $i, i_1, ..., i_k$ are variables of sort addr and l, l' are variables of sort fseq.

Figure 6. The axioms for T_{fseq}

Replacing $T_{\mathsf{Reachability}}$ with T_{path}. So far, to mechanize the reasoning in $T_{\mathsf{Reachability}}$ and similar theories, the approach has been to identify a suitable set of first-order axioms approximating reachability (see, e.g., [26, 20]) and use it in combination with other decision procedures. In this way, only sound but potentially incomplete procedures have been designed. The main problem is that reachability can only be axiomatized in extensions of first-order logic with operators to compute the reflexive and transitive closure of the one-step reachability relation induced by the next-field (see, e.g., [22] for a detailed discussion on the limitations of approximating reachability in first-order logic). In the following, we build a decision procedure for $T_{\mathsf{Reachability}}$ by using an alternative characterization of paths and then exploiting the small-model property.

We want to represent paths by using finite sequences of addresses rather than with the operators in $\Sigma_{\mathsf{Reachability}}$ (cf. Figures 1 and 2). So, we introduce the theory T_{fseq} of finite (arbitrary) sequences of addresses. The signature Σ_{fseq} contains the sort symbols addr and fseq, the function symbols nil : fseq, $cons$: addr $\times$ fseq $\rightarrow$ fseq, hd : fseq $\rightarrow$ elem, tl : fseq $\rightarrow$ fseq, and the empty set of predicate symbols. Its axioms are depicted in Figure 6. We abbreviate with DU the set containing the axioms for

distinctness and uniqueness of the constructors $cons$ and nil, with $Acyclic$ the infinite set of axioms of k-acyclicity (for $k \geq 1$), and with Gen the singleton set containing the axiom expressing the fact that any finite sequence must be generated by the constructors nil and $cons$. In the following, let $FSeq := DU \cup Acyclic \cup Gen$. It is possible to show that [23, 32] any two models $\mathcal{A}$ and $\mathcal{B}$ of $FSeq$ are elementary equivalent. Hence, we are free to choose any particular sub-class of $Mod^{\Sigma_{\mathsf{fseq}}}(FSeq)$ to define the theory T_{fseq} and use $FSeq$ to check the satisfiability in the particular class of models. We define T_{fseq} as the pair $(\Sigma_{\mathsf{fseq}}, \mathsf{TGen})$ where TGen is the class of term-generated structures in $Mod^{\Sigma_{\mathsf{fseq}}}(FSeq)$.

Property 11 *Let Γ be a set of Σ_{fseq}-literals. Then, Γ is T_{fseq}-satisfiable iff Γ is $(\Sigma_{\mathsf{fseq}}, Mod^{\Sigma_{\mathsf{fseq}}}(FSeq))$-satisfiable.*

This Property is an immediate consequence of the completeness of $FSeq$. There exist decision procedures in the literature for similar theories of lists (see, e.g., [15, 28]) which can easily be adapted for T_{fseq}. We notice that T_{fseq} is stably infinite since all the structures in TGen have an infinite domain for fseq (see, e.g., [17]).

Now, we define the theory T_{path} by extending the theory T_{fseq} with some auxiliary function and predicate symbols so that the symbols of $T_{\mathsf{Reachability}}$ appearing in normalized literals (cf. Definition 7) can easily be defined. The additional (with respect to T_{fseq}) symbols and axioms of T_{path} are depicted in Figure 7. Let Σ_{path} be the signature consisting of all the symbols in Figures 6 and 7, and $PATH$ be the set of axioms in Figures 6 and 7. The theory T_{path} is the pair $(\Sigma_{\mathsf{path}}, \mathsf{ETGen})$, where $\mathsf{ETGen} := \{\mathcal{A}^{\Sigma_{\mathsf{path}}} \mid \mathcal{A}^{\Sigma_{\mathsf{path}}} \models PATH$ and $\mathcal{A}^{\Sigma_{\mathsf{fseq}}} \in \mathsf{TGen}\}$. It is possible to extend the theory

$$T_{\mathsf{Base}} \quad := \quad T_{\mathsf{addr}} \oplus T_{\mathsf{elem}} \oplus T_{\mathsf{cell}} \oplus T_{\mathsf{mem}} \oplus T_{\mathsf{path}} \oplus T_{\mathsf{set}}$$

with definitions in such a way that the missing functions and predicates of TLL (namely, ϵ, $[\_]$, $path2set$, $append$, and $addr2set$) can be defined as $\epsilon = nil$, $[i] = cons(i, nil)$, $ispath(p) \Rightarrow path2set(p) = fseq2set(p)$,

$$\left(\begin{array}{ll} ispath(p_1) \wedge ispath(p_2) & \wedge \\ fseq2set(p_1) \cap fseq2set(p_2) = \emptyset & \wedge \\ app(p_1, p_2) = p_3 & \end{array}\right) \quad \Leftrightarrow \quad append(p_1, p_2, p_3)$$

$$isreachablep(m, i, j, p) \Rightarrow getp(m, i, j) \quad = \quad p$$

$$\neg isreachablep(m, i, j, p) \Rightarrow getp(m, i, j) \quad = \quad nil.$$

Let $\Sigma_{\widehat{\mathsf{TLL}}} := \Sigma_{\mathsf{path}} \cup \{getp, append, path2set\}$ and GAP be the definitions above. The theory $\widehat{\mathsf{TLL}}$ is the pair $(\Sigma_{\widehat{\mathsf{TLL}}}, \widehat{\mathsf{ETGen}})$, where

$$\widehat{\mathsf{ETGen}} \quad := \quad \{\mathcal{A}^{\Sigma_{\widehat{\mathsf{TLL}}}} \mid \mathcal{A}^{\Sigma_{\widehat{\mathsf{TLL}}}} \models GAP \text{ and } \mathcal{A}^{\Sigma_{\mathsf{path}}} \in \mathsf{ETGen}\}.$$

By using the formulae in GAP and a simple induction (recall that the class TGen of structures is term generated), it is possible to prove the following Lemma.

Lemma 12 *Let Γ be a set of normalized literals (cf. Definition 7). Then, Γ is TLL-satisfiable (cf. Figures 1 and 2) iff Γ is $\widehat{\mathsf{TLL}}$-satisfiable, after renaming the sort symbol* fseq *with* path *(and vice-versa).*

From $\widehat{\mathsf{TLL}}$ to T_{Base}. By Lemma 12, we know that we can use $\widehat{\mathsf{TLL}}$ in place of TLL for satisfiability checking. Hence, we are left with the problem of reducing the $\widehat{\mathsf{TLL}}$-satisfiability of normalized literals to the T_{Base}-satisfiability of quantifier-free formulae, which can be solved by using a many-sorted variant of the Nelson-Oppen schema [31] since, as we argued above, the theories $T_{\mathsf{addr}}, T_{\mathsf{elem}}, T_{\mathsf{cell}}, T_{\mathsf{mem}}, T_{\mathsf{path}}$, and T_{set} are stably infinite and share only sort symbols. To see how to perform the reduction, two observations are crucial. First, the symbolic execution (or, equivalently, the unfolding of the definitions in GAP and $PATH$) of the function and predicates in $\Sigma_{\widehat{\mathsf{TLL}}}$ over ground terms yields terms built out of the symbols in Σ_{fseq}. In the following, let $\overline{\Sigma_{\widehat{\mathsf{TLL}}}} := \Sigma_{\widehat{\mathsf{TLL}}} \setminus \Sigma_{\mathsf{fseq}}$ and assume that the sort fseq is appropriately renamed to path and vice-versa.

Property 13 *For every ground $\overline{\Sigma_{\widehat{\mathsf{TLL}}}}$-term, there exists an equivalent ground Σ_{fseq}-term.*

(The proof is a simple induction.) Second, by the small-model property (cf. Lemma 9), it is always possible to enumerate the finitely many ground terms of sort path and of sort elem. Hence, by substituting all possible ground terms of sort path (and of sort elem) in the set of normalized literals and symbolically executing the definitions of the symbols in $\overline{\Sigma_{\widehat{\mathsf{TLL}}}}$, it is always possible to reduce the $\widehat{\mathsf{TLL}}$-satisfiability problem of normalized literals to the T_{Base}-satisfiability of quantifier-free formulae. Let Γ be a set of normalized $\Sigma_{\mathsf{Reachability}}$-literals, N be the bound on the cardinality of the domain of addr, according to the small-model property (cf. Lemma 9). Let $D := \{a_1, ..., a_N\}$ be a set of fresh variables of sort addr, and

$$\delta := \bigwedge_{1 \leq k \neq l \leq N} a_k \neq a_l \qquad \eta := \bigwedge_{a:\mathsf{addr} \in \Gamma} \bigvee_{i=1}^{N} a = a_i,$$

so that $\delta \wedge \eta$ constrains the cardinality of the domain of addr to be N. Let $D_p := \{a_1^p, ..., a_N^p\}$ be a finite set of fresh variables for each variable p of sort path in Γ, let $FSeq(D_p) := \{t \mid t$ is a ground term built out of the symbols in $\{nil, cons\} \cup D_p$ and of depth at most $N\}$, and

$$\eta_a \quad := \quad \bigwedge_{p:\mathsf{path} \in \Gamma} \bigvee_{i=1}^{N} a_i^p = a_i$$

$$\eta_p \quad := \quad \bigwedge_{p:\mathsf{path} \in \Gamma} \bigvee_{q \in FSeq(D_p)} (p = q \wedge [[ispath(q)]])$$

so that $\eta_a \wedge \eta_p$ constrains the set of support of path to span over the paths of addresses (i.e. non-repeating sequences of addresses) of length up to N, where $[[ispath(q)]]$ denotes the unfolding of $ispath(q)$. With the notation introduced above, we have the following property.

Property 14 *Γ is $\widehat{\mathsf{TLL}}$-satisfiable iff $\bigvee_{j=1}^{m} [[\tilde{\Gamma}_j]]$ is T_{Base}-satisfiable, where*

- *$\bigvee_{j=1}^{m} \Gamma_j$ is the disjunctive normal form of $\Gamma \wedge (\delta \wedge \eta) \wedge (\eta_a \wedge \eta_p))$,*

app : fseq, fseq $\rightarrow$ fseq	$fseq2set$: fseq $\rightarrow$ set
$app(nil, l) = l$ $app(cons(i, l), l') = cons(i, app(l, l'))$	$fseq2set(nil) = \emptyset$ $fseq2set(cons(i, l)) = \{i\} \cup fseq2set(l)$
$ispath$: fseq	$last$: fseq $\rightarrow$ elem
$ispath(nil)$ $ispath(cons(i, nil))$ $\{i\} \not\subseteq fseq2set(l) \wedge ispath(l) \Rightarrow ispath(cons(i, l))$	$last(cons(i, nil)) = i$ $l \neq nil \Rightarrow last(cons(i, l)) = last(l)$
$isreachable$: mem $\times$ addr $\times$ addr	$isreachablep$: mem $\times$ addr $\times$ addr $\times$ path
$isreachable(m, i, i)$ $m[i].next = i' \wedge isreachable(m, i', j) \Rightarrow isreachable(m, i, j)$	$isreachablep(m, i, i, nil)$ $m[i].next = i' \wedge isreachablep(m, i', j, p)$ $\Rightarrow isreachablep(m, i, j, cons(i, p))$

Note: i, i' are implicitly universally quantified variables of sort addr and l, l', p are implicitly universally quantified variables of sort fseq.

Figure 7. Signature and axioms of T_{path} (extending T_{fseq})

- Γ_j *contains a literal of the form* $p = t$ *for each* p *variable of sort* path *in* Γ *and* t *a term in* $FSeq(D_p)$*,*
- $\tilde{\Gamma_j}$ *is the result of substituting all variables* p *of sort* path *in* Γ *with a term* t *in* $FSeq(D_p)$ *when* $p = t$ *is in* Γ_j *and eliminating trivial equalities,*
- $[[\tilde{\Gamma_j}]]$ *is the result of unfolding all applications of the symbols in* $\overline{\Sigma_{\widehat{\mathsf{TLL}}}}$ *occurring in* $\tilde{\Gamma_j}$*. (Notice that* $[[\tilde{\Gamma_j}]]$ *contains only* $\overline{\Sigma}_{\mathsf{Base}}$*-terms.)*

This is a consequence of Property 13, the small-model property (Lemma 9), and simple properties of the transformation into disjunctive normal form.

The reader may be concerned about the usability of the decision procedure suggested by Property 14. In fact, the formula $\bigvee_{j=1}^{m}[[\tilde{\Gamma_j}]]$ to be checked for T_{Base}-satisfiability can be quite large. Fortunately, a new generation of tools to check the Satisfiability of quantifier-free formulae Modulo a Theory (SMT) based on an integration of SAT solving and (combinations of) decision procedures for sets of literals is available (e.g., [9, 4, 13, 21]). Such tools have been proven quite successful [3] to handle large formulae and so they seem to be able to efficiently check the T_{Base}-satisfiability of $\bigvee_{j=1}^{m}[[\tilde{\Gamma_j}]]$.

4.3 Modular Extensions of TLL

So far, we have assumed that $\Sigma_{\mathsf{addr}} = (\emptyset, \emptyset, \mathsf{addr})$ and $\Sigma_{\mathsf{elem}} = (\emptyset, \emptyset, \mathsf{elem})$. We now show how to extend TLL with some other theories about data and/or pointers. For example, consider the problem of modeling xor-linked lists which are particular kind of singly-linked lists allowing one to simulate doubly-linked lists by storing in the next field of each cell the exclusive-or of two contiguous addresses of cells.[2] If we want to take into account that addresses are encoded by fixed-size bit vectors when modeling xor-linked lists, we need to extend TLL with a theory $\mathbf{FSBV}$ of finite-size bit-vectors such as [11]. But then we are in trouble, since $\mathbf{FSBV}$ does not satisfy the requirement of stably-infiniteness which is necessary to apply (a many-sorted version of) the Nelson and Oppen schema [27]. Fortunately, as said in Section 2.1, there exists a combination method allowing us to overcome this difficulty via the concept of polite theory (cf. Definition 4) that can be combined with any arbitrary theory, not necessarily stably-infinite (such as $\mathbf{FSBV}$).

Politeness of TLL. We now prove that TLL is polite so that we can modularly extend TLL with an arbitrary theory on data and/or pointer values. It is not difficult to show that TLL is smooth (Definition 2) and finitely witnessable (Definition 3) with respect to $\{\mathsf{elem}, \mathsf{addr}\}$. Hence (by Definition 4), we have that

Lemma 15 TLL *is polite with respect to* $\{\mathsf{elem}, \mathsf{addr}\}$.

A consequence of this Lemma and the correctness of the combination method in Section 2.1 is

Theorem 16 *Let* T *be any theory satisfying the following properties:* (i) *for any quantifier-free formula* φ *of* T*, it is possible to decide whether or not* φ *is* T*-satisfiable;* (ii) T *does not share logical symbols with* TLL *besides equality and it does not contain the sorts* mem, set, *and* path*. Then, it is possible to decide the satisfiability of any quantifier-free formula* φ *in the combination of* TLL *and* T*.*

[2] http://en.wikipedia.org/wiki/Xor_linked_list

5. Conclusion and Future Work

We introduced the theory TLL to annotate programs manipulating linked lists. We showed that the satisfiability problem of TLL is decidable, it is NP-complete, and that TLL can be extended with arbitrary (decidable) theories modeling data or pointer values. Finally, we described how to adapt available decision procedures and methods for their combination to build a usable decision procedure for TLL.

There are two main lines of future work. The former is to use extensions of the theories of arrays [18] which allows one to express local reasoning *à la* Separation Logic [30]. The latter consists of adapting our approach to other pointer data structures such as trees and direct acyclic graphs. In fact, we believe it possible to model nodes with n outgoing pointers by using a cell constructor of arity $n+1$ (n pointer fields plus one data field), then generalize the small model property (cf. Lemma 9), and use this to build a practical decision procedure along the lines of Section 4.2.

References

[1] A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Information and Computation*, 183(2):140–164, 2003.

[2] I. Balaban, A. Pnueli, and L. Zuck. Shape analysis by predicate abstraction. In *Proc. of the Int. Conf. on Verif., Model Checking, and Abstract Interpr. (VMCAI'05)*, LNCS, 2005.

[3] C. Barrett, L. de Moura, and A. Stump. SMT-COMP: Satisfiability modulo theories competition. In *Proc. of Conf. on Comp. Aided Verif. (CAV '05)*, volume 3576 of *LNCS*, 2005.

[4] C. W. Barrett and S. Berezin. CVC Lite: a new implementation of the Cooperating Validity Checker. In R. Alur and D. A. Peled, editors, *Proc. CAV-16*, volume 3114 of *LNCS*, pages 515–518. Springer, 2004.

[5] M. Benedikt, T. W. Reps, and S. Sagiv. A decidable logic for describing linked data structures. In *ESOP*, pages 2–19, 1999.

[6] J. Berdine, C. Calcagno, and P. W. O'Hearn. A decidable fragment of separation logic. In *Proc. of FSTTCS*, 2004.

[7] J. D. Bingham and Z. Rakamaric. A logic and decision procedure for predicate abstraction of heap-manipulating programs. In *Verification, Model Checking, and Abstract Interpretation, 7th International Conference (VMCAI'06)*, 2006.

[8] R. Bornat. Proving pointer programs in hoare logic. In *Mathem. of Program Constr. (MPC'00)*, 2000.

[9] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani. An incremental and layered procedure for the satisfiability of linear arithmetic logic. In *Proc. TACAS-11*, volume 3440 of *LNCS*, 2005.

[10] R. M. Burstall. Some techniques for proving correctness of programs which alter data structures. *Machine Intelligence*, 7:23–50, 1972.

[11] D. Cyrluk, O. Möller, and H. Rueß. An efficient decision procedure for the theory of fixed-sized bit-vectors. In *9th Int. Conf. on CAV*, pages 60–71, 1997.

[12] D. Dams and K. S. Namjoshi. Shape analysis through predicate abstraction and model checking. In *Proc. of VMCAI'03*, LNCS, 2003.

[13] D. Déharbe and S. Ranise. Light-Weight Theorem Proving for Debugging and Verifying Units of Code. In *Int. Conf. on Software Eng. and Formal Methods (SEFM03)*, 2003.

[14] J.-C. Filliâtre. Why: a multi-language multi-prover verification tool. Research Report 1366, LRI, U. Paris Sud, 2003.

[15] P. Fontaine, S. Ranise, and C. G. Zarba. Combining lists with non-stably infinite theories. In *Logic for Progr., Artif. Intell., and Reas. (LPAR'04)*, volume 3452 of *LNCS*, 2005.

[16] J. H. Gallier. *Logic for Computer Science*. Wiley, 1986.

[17] S. Ghilardi. Model theoretic methods in combined constraint satisfiability. *J. of Aut. Reas.*, 33(3–4):221–249, 2005.

[18] S. Ghilardi, E. Nicolini, S. Ranise, and D. Zucchelli. Deciding Extensions of the Theory of Arrays by Integrating Decision Procedures and Instantiation Strategies. In *Proc. of Logics in Artificial Intelligence (Jelia'06)*, 2006. To appear.

[19] J. L. Jensen, M. E. Jorgensen, N. Klarlund, and M. I. Schwartzbach. Automatic Verification of Pointer Programs using Monadic Second-Order Logic. In *SIGPLAN Conf. on Progr. Lang. Design and Impl.*, pages 226–236, 1997.

[20] S. K. Lahiri and S. Qaader. Verifying properties of well-founded linked-lists. In *POPL'06*, 2006.

[21] S. K. Lahiri and S. A. Seshia. The UCLID decision procedure. In R. Alur and D. A. Peled, editors, *Proc. CAV-16*, volume 3114 of *LNCS*, 2004.

[22] T. Lev-Ami, N. Immerman, T. W. Reps, S. Sagiv, S. Srivastava, and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. In *Conf. on Automated Deduction (CADE'05)*, 2005.

[23] M. J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *4th Annual Symp. on Logic in Computer Science (LICS'88)*, 1988.

[24] S. McPeak and G. Necula. Data structures specification via local equality axioms. In *Proc. of Conf. on Computer Aided Verification (CAV'05)*, LNCS, 2005.

[25] F. Mehta and T. Nipkow. Proving pointer programs in higher-order logic. *Inform. and Comp.*, 2005. To appear.

[26] G. Nelson. Verifying reachability invariants of linked structures. In *POPL*, pages 38–47, 1983.

[27] G. Nelson and D. C. Oppen. Simplifications by cooperating decision procedures. *ACM Trans. on Prog. Lang. and Sys.*, 1(2):245–257, 1979.

[28] D. C. Oppen. Reasoning about recursively defined data structures. *J. of the ACM*, 27(3):403–411, 1980.

[29] S. Ranise, C. Ringeissen, and C. G. Zarba. Combining container data structures with non-stably infinite theories of elements. In *Proc. of the Int. Workshop on Frontiers of Combining Systems (FroCos'05)*, LNCS, 2005.

[30] J. Reynolds. Separation logic: a logic for shared mutable data structures, 2002. Invited Paper at LICS'02.

[31] C. Tinelli and C. G. Zarba. Combining decision procedures for sorted theories. In *Logics in Artificial Intelligence (Jelia'04)*, number 3229 in LNCS, 2004.

[32] D. van Dalen. *Logic and Structure*. Springer-Verlag, 1989. Second edition.

[33] G. Yorsh, A. Rabinovich, M. Sagiv, A. Meyer, and A. Bouajjani. A logic of reachable patterns in linked data-structures. In *Proc. of FOSSACS'06*, LNCS, 2006.