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

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

Calogero Zarba
Universität der 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 \(\text{NP}\)-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 \(\text{NP}\)-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 naive exploitation of the small model property is not practical. We view TLL as an extension of a core theory \(T_{\text{Base}}\) by constructs for reachability. In this way, we devise a reduction of the decision problem for TLL to that of \(T_{\text{Base}}\) 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 “GEC-COO” and ANR “ARROWS.”
ments 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 $\Sigma$ 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 $\Sigma_1 \cup \Sigma_2 = (S_1 \cup S_2, F_1 \cup F_2, P_1 \cup P_2)$. Given a signature $\Sigma$, we assume the standard notions of $\Sigma$-term, $\Sigma$-literal, and $\Sigma$-formula. A $\Sigma$-sentence is a $\Sigma$-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 $t(\varphi)$ is a term (form, resp.), we denote with $\varsigma(t)$ ($\varsigma(\varphi)$, resp.) the set of variables of sort $\sigma$ occurring in $t$ ($\varphi$, resp.). If $\varphi$ is either a term or a formula, we denote with $\varsigma(\varphi)$ the set $\bigcup_{\varsigma} \varsigma(\varphi)$. Finally, if $\Phi$ is a set of terms or a set of formulæ, $\varsigma(\Phi)$ and $\varsigma(\Phi)$ are defined as obvious. Below, we will identify conjuctions of formulæ $\varphi_1 \land \cdots \land \varphi_n$ with the set $\{\varphi_1, \ldots, \varphi_n\}$.

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

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

### 2.1 Combination schemas

Let $T_i = (\Sigma_i, A_i)$ be a theory, for $i = 1, 2$. The combination of $T_1$ and $T_2$ is the theory $T_1 \oplus T_2 = (\Sigma, A)$ where $\Sigma = \Sigma_1 \cup \Sigma_2$ and $A := \{ A \mid A^{\Sigma_1} \subseteq A_1 \text{ and } A^{\Sigma_2} \subseteq A_2 \}$.

If $\Phi_i$ is a set of $\Sigma_i$-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 $\Sigma$-theory; $T$ has the finite model property with respect to $S_0$ if for every $T$-satisfiable quantifier-free $\Sigma$-formula $\varphi$ there exists a $T$-interpretation $\mathcal{A}$ satisfying $\varphi$ such that $A_{ \sigma}$ 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 $\Sigma$-theory; $T$ is stably infinite with respect to $S_0$ if for every $T$-satisfiable quantifier-free $\Sigma$-formula $\varphi$ there exists a $T$-interpretation $\mathcal{A}$ satisfying $\varphi$ such that $A_{ \sigma}$ 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 $\Sigma$-theory; $T$ is smooth with respect to $S_0$ if: (i) for every $T$-satisfiable quantifier-free $\Sigma$-formula $\varphi$, (ii) for every $T$-interpretation $\mathcal{A}$ satisfying $\varphi$, (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 $\varphi$ 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 $\Sigma$-theory; $T$ is finitely witnessable with respect to $S_0$ if there exists a computable function witness that for every quantifier-free $\Sigma$-formula $\varphi$ returns a quantifier-free $\Sigma$-formula $\psi = \text{witness}(\varphi)$ such that: (i) $\varphi$ and $(\exists \bar{v})\psi$ are $T$-equivalent, where $\bar{v} = \text{vars}(\varphi) \setminus \text{vars}(\psi)$; (ii) if $\psi$ is $T$-satisfiable then there exists a $T$-interpretation $\mathcal{A}$ satisfying $\varphi$ such that the domain $A_{ \sigma}$ interpreting the sort $\sigma$ in $\mathcal{A}$ is the (finite) set $\{ \text{vars}_{ \sigma}(\psi)^{\mathcal{A}} \}$ of elements in $A$ interpreting the variables of sort $\sigma$ in $\psi$ (in symbols, $A_{ \sigma} = [\text{vars}_{ \sigma}(\psi)^{\mathcal{A}}]$), 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 $\Sigma$-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 $\Gamma$ 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 $\Gamma_1$ is a $\Sigma_1$-literal, for $i = 1, 2$; and (b) $\Gamma_1 \cup \Gamma_2$ is $(T_1 \cup T_2)$-satisfiable if and only if $\Gamma$ is $(T_1 \cup T_2)$-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 $\psi_2 = \text{witness}(\Gamma_2)$, and we output $\Gamma_1 \cup \{ \psi_2 \}$. Intuitively, this phase introduces the fresh variables in $\text{vars}(\psi_2) \setminus \text{vars}(\Gamma)$, 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 $V_{ \sigma} = \text{vars}_{ \sigma}(\psi_2)$ 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

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

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

**Fourth phase: check.** Let $\Gamma_1 \cup \{ \psi_2 \} \cup \text{arr}(V, E)$ be a conjunction obtained in the decomposition phase. The check phase consists in the following steps: 1. if $\Gamma_1 \cup \text{arr}(V, E)$ is $T_1$-satisfiable go to step 2; otherwise output fail. 2. If $\{ \psi_2 \} \cup \text{arr}(V, E)$ 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 TLL := (ΣTLL, TLL) where ΣTLL := ΣCells ∪ ΣMemory ∪ ΣReachability ∪ ΣSets ∪ ΣBridge (see Figure 1) and TLL is the class of ΣTLL-structures satisfying the conditions in Figure 2.

The interpretations of the sort symbols elem and addr is left unspecified. So, we take Σaddr = (0, 0, addr) and Σelem = (0, 0, elem). We will see how to overcome this limitation in Section 4.3.

The interpretation of the function symbols in ΣCells or ΣMemory 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., snapshots of the memory where the linked lists are stored. In particular, the interpretation of the symbols in ΣCells ∪ ΣMemory is closely related to that proposed by Burstall [10].

Property 6 A_{addr}^{A_{cell}} is isomorphic to A_{elem}^{A_{cell}} × A_{addr}^{A_{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 [i1, ..., in] 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 e 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:

acyclic(m, i) ⇔ null ∈ addr2set(m, i) (1)

cyclic(m, i) ⇔ ∃i, f, i_p. \left\{ \begin{array}{l l} i_s \in addr2set(m, i) \land f \in addr2set(m, i) \land i_p \in addr2set(m, i) \land i_p \neq f \land m[i_p].next = i_s \land m[f].next = i_s \end{array} \right. (2)

isfinite(m, i) ⇔ acyclic(m, i) ∨ cyclic(m, i). (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 isfinite(m, i) holds then there exists an integer n ≥ 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 ΣSets 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. Figure 3) 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]):

acyclic(m, i) ∧ acyclic(m, j) ∧ ∀k_i, k_j. \exists p_i, p_j. \left\{ \begin{array}{l l} \text{reach}(m, i, k_i, p_i) \land \text{reach}(m, j, k_j, p_j) \land k_i \neq null \land k_j \neq null \end{array} \right. \Rightarrow k_i \neq k_j

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-
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

<table>
<thead>
<tr>
<th>Signature</th>
<th>Sorts</th>
<th>Functions</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Sigma _{\text{Cells}} )</td>
<td>cell, elem</td>
<td>( \text{error} : \text{cell} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{mkcell} : \text{elem}, \text{addr} \to \text{cell} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( _\text{data} : \text{cell} \to \text{elem} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( _\text{next} : \text{cell} \to \text{addr} )</td>
<td>—</td>
</tr>
<tr>
<td>( \Sigma _{\text{Memory}} )</td>
<td>mem, addr, cell</td>
<td>( \text{null} : \text{addr} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( __ : \text{mem}, \text{addr} \to \text{cell} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{upd} : \text{mem}, \text{addr}, \text{cell} \to \text{mem} )</td>
<td>—</td>
</tr>
<tr>
<td>( \Sigma _{\text{Reachability}} )</td>
<td>mem, addr, path</td>
<td>( \epsilon : \text{path} )</td>
<td>append : \text{path}, \text{path}, \text{path}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( __ : \text{addr} \to \text{path} )</td>
<td>reach : \text{mem}, \text{addr}, \text{addr}, \text{path}</td>
</tr>
<tr>
<td>( \Sigma _{\text{Sets}} )</td>
<td>addr, set</td>
<td>( \emptyset : \text{set} )</td>
<td>( \in : \text{addr}, \text{set} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( {,} : \text{addr} \to \text{set} )</td>
<td>( \subseteq : \text{set}, \text{set} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \cup, \cap, \setminus : \text{set}, \text{set} \to \text{set} )</td>
<td>—</td>
</tr>
<tr>
<td>( \Sigma _{\text{Bridge}} )</td>
<td>mem, addr, set, path</td>
<td>( \text{path2set} : \text{path} \to \text{set} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{addr2set} : \text{mem}, \text{addr} \to \text{set} )</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \text{getp} : \text{mem}, \text{addr}, \text{addr} \to \text{path} )</td>
<td>—</td>
</tr>
</tbody>
</table>

**Interpretation of sort symbols:** cell, mem, set, and path

Each sort \( \sigma \) in \( \Sigma _{\text{TLL}} \) is mapped to a non-empty set \( A _{\sigma} \) such that:

- \( A _{\text{cell}} = A _{\text{elem}} \times A _{\text{addr}} \);
- \( A _{\text{mem}} = A _{\text{cell}} \);
- \( A _{\text{addr}} \) is the power-set of \( A _{\text{cell}} \);
- \( A _{\text{path}} \) is the set of all finite sequences of \( (\text{pairwise}) \) distinct elements of \( A _{\text{cell}} \).

**Signature | Interpretation**

- \( \Sigma _{\text{Cells}} \)
  - \( \text{mkcell} : \text{cell} \times \text{cell} \times \text{cell} \)
  - \( \text{data} : \text{cell} \to \text{cell} \times \text{cell} \)
- \( \Sigma _{\text{Memory}} \)
  - \( \text{null} : \text{cell} \times \text{cell} \times \text{cell} \)
- \( \Sigma _{\text{Reachability}} \)
  - \( \epsilon : \text{cell} \times \text{cell} \times \text{cell} \)
- \( \Sigma _{\text{Sets}} \)
  - \( \emptyset : \text{cell} \times \text{cell} \times \text{cell} \)
  - \( \{,\} : \text{cell} \times \text{cell} \times \text{cell} \)
  - \( \cup, \cap, \\setminus : \text{cell} \times \text{cell} \times \text{cell} \)
- \( \Sigma _{\text{Bridge}} \)
  - \( \text{path2set} : \text{cell} \times \text{cell} \times \text{cell} \)
  - \( \text{addr2set} : \text{cell} \times \text{cell} \times \text{cell} \)
  - \( \text{getp} : \text{cell} \times \text{cell} \times \text{cell} \)

**Note:** \( a_{i \mapsto u} \) abbreviates the function \( a' \) with domain \( A _{\text{addr}} \) and co-domain \( A _{\text{cell}} \) 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 \( \Sigma _{\text{TLL}} \)-interpretation \( A \)**
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 \( n \) ull 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.:** \( \text{acyclic}(M, i) \land R^0_i = \text{addr2set}(M, i) \)
- **Post-cond.:** \( \text{acyclic}(M, o) \land R^0_o = \text{addr2set}(M, o) \)
- **Invariant:** \( \text{addr2set}(M, i) \land \text{addr2set}(M, o) = \{ \text{null} \} \land \text{addr2set}(M, i) \cup \text{addr2set}(M, o) = R^0_i \)

The pre-condition states that \( i \) points to a (possibly) empty acyclic list and it defines the set \( R^0_i \) 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, \( R^0_i \)) 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 \( \Sigma_{\text{TLL}} \)-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.

**4.1 Small model property**

Let \( \Gamma \) be a conjunction of TLL-literals. To simplify the proof of the small model property, we pre-process the set \( \Gamma \) 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:

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

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

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

1. \( \Gamma \) is TLL-satisfiable;
2. \( \Gamma \) is true in a TLL-interpretation \( A \) such that \( |A_{\text{elem}}| \leq n + r \cdot |A_{\text{addr}}| \) and \( |A_{\text{addr}}| \leq m + 1 + rm + s^2 + s^3 \).
The small model property yields a brute force method to check the TLL-satisfiability of normalized literals. One must enumerate all sets of 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 $x \in \{\text{elem}, \text{addr}, \cell, \mem, \set, \Reachability\}$ and $\mathbf{X} := \{\mathcal{A}_{\Sigma_X} \mid \mathcal{A}_{\Sigma_{\text{null}}} \in \text{TLL}\}$. 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-infinity) for applying the combination schema are satisfied.

Requirements (i) and (ii) are unproblematic for the theories $T_{\text{elem}}$ and $T_{\text{addr}}$ 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 $T_{\text{Cells}}$, $T_{\text{mem}}$, 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 $T_{\text{Cells}}$, $T_{\text{mem}}$, and $T_{\set}$ are stably-infinite (see, e.g., [29]) and only share sort symbols.

The situation is more complex for $T_{\text{Reachability}}$ (cf. Figure 5) for which we must solve two problems. First, there is no available decision procedure for $T_{\text{Reachability}}$. Below, instead of building such a procedure, we show how to replace $T_{\text{Reachability}}$ 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.

**Replacing $T_{\text{Reachability}}$ with $T_{\text{path}}$.** So far, to mechanize the reasoning in $T_{\text{Reachability}}$ 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 $T_{\text{Reachability}}$ 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 $\Sigma_{\text{reachability}}$ (cf. Figures 1 and 2). So, we introduce the theory $T_{\text{sseq}}$ of finite (arbitrary) sequences of addresses. The signature $\Sigma_{\text{sseq}}$ contains the sort symbols addr and fseq, the function symbols $\text{nil} : \text{fseq}$, $\text{cons} : \text{addr} \times \text{fseq} \rightarrow \text{fseq}$, $\text{hd} : \text{fseq} \rightarrow \text{elem}$, $\text{tl} : \text{fseq} \rightarrow \text{fseq}$, and the empty set of predicate symbols. Its axioms are depicted in Figure 6. We abbreviate with $DU$ the set containing the axioms for

**Figure 6. The axioms for $T_{\text{sseq}}$**
distinctness and uniqueness of the constructors \textit{cons} and \textit{nil}, with \textit{Acyclic} the infinite set of axioms of \textit{k}-acyclicity (for \(k \geq 1\)), and with \textit{Gen} the singleton set containing the axiom expressing the fact that any finite sequence must be generated by the constructors \textit{nil} and \textit{cons}. In the following, let \(FSeq := DU \cup \text{Acyclic} \cup \text{Gen}\). It is possible to show that \([23, 32]\) any two models \(A\) and \(B\) of \(FSeq\) are elementarily equivalent. Hence, we are free to choose any particular sub-class of \(\text{Mod}_{\Sigma_{\text{fseq}}} (FSeq)\) to define the theory \(T_{\text{fseq}}\) and use \(FSeq\) to check the satisfiability in the particular class of models. We define \(T_{\text{fseq}}\) as the pair \((\Sigma_{\text{fseq}}, \text{TGen})\) where \(\text{TGen}\) is the class of term-generated structures in \(\text{Mod}_{\Sigma_{\text{fseq}}} (FSeq)\).

**Property 11** Let \(\Gamma\) be a set of \(\Sigma_{\text{fseq}}\)-literals. Then, \(\Gamma\) is \(T_{\text{fseq}}\)-satisfiable iff \(\Gamma\) is \((\Sigma_{\text{fseq}}, \text{Mod}_{\Sigma_{\text{fseq}}} (FSeq))\)-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., \([15, 28]\)) that \(T_{\text{fseq}}\) is stably infinite since all the structures in \(\text{TGen}\) are depicted in Figure 7. Let \(\Sigma\) 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 \(T_{\text{path}}\) is the pair \((\Sigma_{\text{path}}, \text{ETGen})\), where \(\text{ETGen} := \{A^{\Sigma_{\text{path}}} \models PATH\text{ and } A^{\Sigma_{\text{path}}} \in \text{TGen}\}\). It is possible to extend the theory \(T_{\text{Base}} := T_{\text{addr}} \oplus T_{\text{elem}} \oplus T_{\text{cell}} \oplus T_{\text{mem}} \oplus T_{\text{path}} \oplus T_{\text{set}}\) with definitions in such a way that the missing functions and predicates of \(TLL\) (namely, \(\epsilon, [\ldots], \text{path2set}, \text{append}, \text{and addr2set}\)) can be defined as \(\epsilon = \text{nil}, [\ldots] = \text{cons}(i, \text{nil})\).

\[
\left(\text{isp}(p_1) \land \text{isp}(p_2) \land \text{fseq2set}(p_1) \land \text{fseq2set}(p_2) = \emptyset \land \text{app}(p_1, p_2) = p_3\right) \iff \text{append}(p_1, p_2, p_3)
\]

Let \(\Sigma_{\text{TLL}} := \Sigma_{\text{path}} \cup \{\text{getp}, \text{append}, \text{path2set}\}\) and \(\text{GAP}\) be the definitions above. The theory \(\text{TLL}\) is the pair \((\Sigma_{\text{TLL}}, \text{ETGen})\), where \(\text{ETGen} := \{A^{\Sigma_{\text{TLL}}} \models \text{GAP}\text{ and } A^{\Sigma_{\text{TLL}}} \in \text{ETGen}\}\).

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

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

From \(\text{TLL}\) to \(T_{\text{Base}}\). By Lemma 12, we know that we can use \(\text{TLL}\) in place of \(\text{TLL}\) for satisfiability checking. Hence, we are left with the problem of reducing the \(\text{TLL}\)-satisfiability of normalized literals to the \(T_{\text{Base}}\)-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 \(T_{\text{addr}}, T_{\text{elem}}, T_{\text{cell}}, T_{\text{mem}}, T_{\text{path}}, \) and \(T_{\text{set}}\) 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 \(\text{GAP}\) and \(PATH\)) of the function and predicates in \(\Sigma_{\text{TLL}}\) over ground terms yields terms built out of the symbols in \(\Sigma_{\text{fseq}}\). In the following, let \(\Sigma_{\text{TLL}} := \Sigma_{\text{TLL}} \setminus \Sigma_{\text{fseq}}\) and assume that the sort \(\text{fseq}\) is appropriately renamed to path and vice-versa.

**Property 13** For every ground \(\Sigma_{\text{TLL}}\)-term, there exists an equivalent ground \(\Sigma_{\text{fseq}}\)-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 \(\Sigma_{\text{TLL}}\), it is always possible to reduce the \(\text{TLL}\)-satisfiability problem of normalized literals to the \(T_{\text{Base}}\)-satisfiability of quantifier-free formulae. Let \(\Gamma\) be a set of normalized \(\Sigma_{\text{Reachability}}\)-literals. To see how to perform the reduction, two observations are crucial. First, the symbolic execution (or, equivalently, the unfolding of the definitions in \(\text{GAP}\) and \(PATH\)) of the function and predicates in \(\Sigma_{\text{TLL}}\) over ground terms yields terms built out of the symbols in \(\Sigma_{\text{fseq}}\). In the following, let \(\Sigma_{\text{TLL}} := \Sigma_{\text{TLL}} \setminus \Sigma_{\text{fseq}}\) and assume that the sort \(\text{fseq}\) is appropriately renamed to path and vice-versa.

\[
\delta := \bigwedge_{1 \leq k, l \leq N} a_k \neq a_l \quad \eta := \bigwedge_{a \in \text{addr}} \bigvee_{i = 1}^{N} a = a_i,
\]

so that \(\delta \land \eta\) constrains the cardinality of the domain of \(\text{addr}\) to be \(N\).

\[
\eta_a := \bigwedge_{p : \text{path} \in \Gamma} \bigvee_{i = 1}^{N} a_i^p = a_i
\]

\[
\eta_p := \bigwedge_{p : \text{path} \in \Gamma} \bigvee_{q \in \text{FSeq}(D_p)} (p = q \land [\text{isp}(q)])
\]

so that \(\eta_a \land \eta_p\) 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 \([\text{isp}(q)]\) denotes the unfolding of \(\text{isp}(q)\). With the notation introduced above, we have the following property.

**Property 14** \(\Gamma\) is \(\text{TLL}\)-satisfiable iff \(\bigvee_{j=1}^{m} [F_j]\) is \(T_{\text{Base}}\)-satisfiable, where

- \(\bigvee_{j=1}^{m} F_j\) is the disjunctive normal form of \(\Gamma \land (\delta \land \eta) \land (\eta_a \land \eta_p)\).
of all the symbols in \( \sum \) occurring in \( \Gamma_j \). (Notice that \( \Gamma_j \) contains only \( \Sigma_{Base} \)-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 \( \bigvee_{j=1}^{m} \Gamma_j \) 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 \( \bigvee_{j=1}^{m} \Gamma_j \).

### 4.3 Modular Extensions of TLL

So far, we have assumed that \( \Sigma_{addr} = (\emptyset, \emptyset, addr) \) and \( \Sigma_{elem} = (\emptyset, \emptyset, elem) \). 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. 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 \( \text{FSBV} \) of finite-size bit-vectors such as [11]. But then we are in trouble, since \( \text{FSBV} \) 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 \( \text{FSBV} \)).

#### 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 \{elem, addr\}. Hence (by Definition 4), we have that

**Lemma 15** TLL is polite with respect to \{elem, 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 \( \varphi \) of \( T \), it is possible to decide whether or not \( \varphi \) 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 \( \varphi \) in the combination of TLL and \( T \).

---

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

<table>
<thead>
<tr>
<th>( \text{app} : \text{fseq, fseq} \rightarrow \text{fseq} )</th>
<th>( \text{fseq2set} : \text{fseq} \rightarrow \text{set} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{app}(\text{nil}, l) = l )</td>
<td>( \text{fseq2set}(\text{nil}) = \emptyset )</td>
</tr>
<tr>
<td>( \text{app}(\text{cons}(i, l), l') = \text{cons}(i, \text{app}(l, l')) )</td>
<td>( \text{fseq2set}(\text{cons}(i, l)) = {i} \cup \text{fseq2set}(l) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \text{ispath} : \text{fseq} )</th>
<th>( \text{last} : \text{fseq} \rightarrow \text{elem} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ispath}(\text{nil}) )</td>
<td>( \text{last}(\text{cons}(i, \text{nil})) = i )</td>
</tr>
<tr>
<td>( \text{ispath}(\text{cons}(i, l)) )</td>
<td>( i \neq \text{nil} \Rightarrow \text{last}(\text{cons}(i, l)) = \text{last}(l) )</td>
</tr>
<tr>
<td>( {i} \notin \text{fseq2set}(l) \land \text{ispath}(l) \Rightarrow \text{ispath}(\text{cons}(i, l)) )</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** \( i, i' \) are implicitly universally quantified variables of sort \( \text{addr} \) and \( l, l', p \) are implicitly universally quantified variables of sort \( \text{fseq} \).
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 last to build a practical decision procedure along the lines of Section 4.2.

References


