Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

On Logics of Aliasing

Marius Bozga, Radu Iosif and Yassine Lakhnech

Report n^o TR-2004-4

April 23, 2004

On Logics of Aliasing

Marius Bozga, Radu Iosif and Yassine Lakhnech

April 23, 2004

Abstract

In this paper we investigate the existence of a deductive verification method based on a logic that describes pointer aliasing. The main idea of such a method is that the user has to annotate the program with loop invariants, pre- and post-conditions. The annotations are then automatically checked for validity by propagating weakest preconditions and verifying a number of induced implications. Such a method requires an underlying logic which is decidable and has a sound and complete weakest precondition calculus. We start by presenting a powerful logic (**wAL**) which can describe the shapes of most recursively defined data structures (lists, trees, etc.) has a complete weakest precondition calculus but is undecidable. Next, we identify a decidable subset (**pAL**) for which we show closure under the weakest precondition operators. In the latter logic one loses the ability of describing unbounded heap structures, yet bounded structures can be characterized up to isomorphism. For this logic two sound and complete proof systems are given, one based on natural deduction, and another based on the effective method of analytic tableaux. The two logics presented in this paper can be seen as extreme values in a framework which attempts to reconcile the naturally oposite goals of expressiveness and decidability.

Keywords: Deductive verification, pointer aliasing, satisfiability problem, weakest preconditions

Reviewers:

Notes:

How to cite this report:

@techreport { ,
title = { On Logics of Aliasing},
authors = { Marius Bozga, Radu Iosif and Yassine Lakhnech},
institution = { Verimag Technical Report },
number = {TR-2004-4},
year = { },
note = { }
}

1 Introduction

The problem of pointer *aliasing* plays an important role in the fields of static analysis and software model checking. In general, static analysis checks basic properties such as data sharing and circularities in the heap of a program, useful for optimizing object-oriented compilers, while model checking deals with the evolution of heap structures, in both shape and contents, over time. An early result [21] shows that precise may-alias analysis in the presence of loops is undecidable. As a consequence, the approach adopted by the static analysis community, is the abstraction-based *shape analysis* [23]. This method is effective in the presence of loops, since the domain of the analysis is bounded, but often imprecise. In this paper we present an orthogonal solution to the aliasing problem, in that precision is the primary goal. To ensure termination, we use Floyd's method [10] of annotating the program with pre-, post-conditions and loop invariants. The annotations are subsequently verified by a push-button procedure, that computes weakest preconditions expressed using an effectively decidable logic.

The key is to find a logic that can altogether (i) express aliasing and shape properties of the program heap, (ii) is effectively decidable, and moreover, (iii) has a sound and complete weakest precondition calculus with respect to the atomic statements. While the second and third requirements are clear, the first one is still ambiguous: what kind of specifications can we express in a decidable heap logic with weakest preconditions? The contribution of this paper is the definition of a formal framework in which we prove that such logics *can* be found. Our focus is on imperative programs with destructive updating, in which heaps are viewed as shape graphs with labels only on edges i.e., we ignore from the start the internal states of the objects.

As a starting point, we present a general logic Weak Alias Logic (**wAL**) that is expressive enough to describe the recursive data structures of interest (lists, trees, dags etc.) as infinite classes of finite graphs. This logic has also a sound and complete weakest precondition calculus with respect to atomic statements such as new object creation and assignment of pointers. The satisfiability problem of the **wAL** logic is found to be undecidable but recursively enumerable, which motivates further searches for semi-decision procedures and non-trivial decidable subsets.

In the rest of the paper, we define a decidable subset of **wAL**, called Propositional Alias Logic (**pAL**) for describing pointer aliasing that is, moreover, able to characterize arbitrary finite structures and finite classes of structures. The tradeoff in defining **pAL** is losing the ability to describe a number of interesting shape properties such as listness, (non)circularity, etc. For this logic, we give a proof-theoretic system based on natural deduction, and an effective tableau decision method. Both systems are shown to be sound and complete. Moreover, the satisfiability problem for **pAL** is shown to be NP-complete. The last point concerns the definition, in **pAL**, of weakest preconditions for imperative programs with destructive updating. At this point, we use the **wAL** weakest precondition calculus, previously developped in [2]. Our weakest precondition calculus for **pAL** is sound and complete, as a consequence of the soundness and completness of the definitions for **wAL** weakest preconditions.

1.0.1 Related Work

To describe properties of dynamic program stores, various formalisms have been proposed in the literature e.g., L_r [1], BI (Bunched Implications) [13], Separation Logic [22] and PAL (Pointer

Assertion Language) [17]. As a common point with our work, L_r [1] uses regular expressions to describe reachability between two points in the heap and is shown to be decidable, yet the weakest precondition calculus is not developed. On the other hand, BI [13] and Separation Logic [22] produce remarkably simple preconditions and have quite clean proof-theoretic models [18]. Another feature of these formalisms is that they allow for compositional reasoning [19]. As a downside, the quantifier fragment, essential to express weakest preconditions, is undecidable [5], while the ground (propositional) fragment is decidable, a tableau procedure being proposed in [11]. In a later publication [6], a specialization of the ground fragment of BI to tree models is used as a type system for a language, based on λ-calculus, that handles trees. An effectively decidable formalism is PAL [17], an extension of second-order monadic logic on trees that allows to describe a restricted class of graphs, known as "graph types" [16], as opposed to our approach that deals with unrestricted graphs. Programs that manipulate such graphs are restricted to updating only the underlying tree (backbone). The resulting actions can thus be described in monadic second-order logic, and the validity of Hoare triples expressed in PAL can be automatically decided [15].

The decision procedures for both L_r and PAL use Rabin's result on the monadic second order theory of n successors (SnS) [20]. The decision procedure for the satisfiability of SnS is however non-elementary. We show that the decision problem for the **pAL** logic is NP-complete, thus drastically improving the complexity bounds. Also, to the best of our knowledge, no previously published work on the verification of heap properties has the ability to deal with *unrestricted* (destructively updated) data structures, developing a sound and complete weakest precondition calculus on top of a decidable logic for graphs.

2 Weak Alias Logic

In this section we introduce Weak Alias Logic (**wAL**), a logic that is expressive enough for defining recursive data structures (lists, trees, etc) as infinite classes of finite graphs, as well as for defining a weakest precondition calculus of imperative programming languages with destructive updating [2]. This section defines the logic, and Section 5 briefly recalls the weakest precondition calculus that has been developed on top of it.

Before giving the syntax of **wAL**, let us introduce the notion of *heap*, which is central in defining interpretations of **wAL** formulas. Intuitively, a heap is represented by a graph where the nodes model objects and the edges model pointers between objects. The heap edges are labeled with symbols from a given alphabet Σ, which stands for the set of all program pointers, including all program variables and record fields (selectors). It is furthermore required that the graph be deterministic, as a program pointer can only point to one object at a time.

In this paper we adopt the *storeless representation* [2], [12], [14], [8] of a graph, in which each node is associated the language recognized by the automaton whose set of states is given by the set of graph nodes, the transition relation by the set of edges, the initial state is a designated entry point in the heap, and the unique final state, the node itself. The interested reader is referred to [2] for a detailed discussion on the advantages of the storeless representation of heaps, such as compatibility with garbage collection and isomorphic transformations.

Definition 1 (Heap) *A* heap $\mathcal{M} \subseteq \mathcal{P}(\Sigma^+)$ *is either the empty set or a finite set* $\{X_1, X_2, \ldots, X_n\}$ *satisfying the following conditions, for all* $1 \leq i, j \leq n$*:*

(C1) non-emptiness: $X_i \neq \emptyset$*,*

(C2) determinism: $i \neq j \Rightarrow X_i \cap X_j = \emptyset$*,*

(C3) prefix closure and right regularity:

$$\forall x \in X_i \, [\forall y, z \in \Sigma^+ [x = yz \Rightarrow \exists \, 1 \leq k \leq n \, [y \in X_k \wedge X_k z \subseteq X_i]]]$$

One can also think of a heap element as the set of all incoming paths leading to it, paths that start with a program variable. The (C1),(C2) and (C3) restrictions must be imposed on the elements of a heap in order to maintain the correspondence (up to isomorphism) with the graph model [2]. An equivalent approach, taken in [14], [8], is to consider the languages in the heap as equivalence classes of a right-regular relation on $\Sigma^* \times \Sigma^*$. The set of all heaps over an alphabet Σ is denoted in the following by $\mathcal{H}(\Sigma)$.

Figure 1 introduces the abstract syntax (upper part) and semantics (lower part) of the **wAL** logic. The terms of a **wAL** formula are regular expressions ρ over the alphabet Σ with free variables from a set Var. We allow the classical composition operations on regular expressions, together with the left derivate, denoted by $\rho_1^{-1}\rho_2 \triangleq \{\sigma \in \Sigma^* \mid \rho_1\sigma \cap \rho_2 \neq \emptyset\}$. Formulas are built from the atomic propositions $\rho_1 = \rho_2$ (language equivalence) and $\langle X \rangle \rho_1$ (modality) connected with the classical first-order operators $\wedge$, $\neg$ and $\exists$. A less usual requirement is imposed on the syntax of the existential quantifier: the quantified variable need to occur at least once within the angled brackets of a modality in the scope of the quantifier, which is formally captured by the $\varphi\langle X \rangle$. Notice also that only free variables can occur inside the modality brackets. A formula φ is said to be *closed* if no variables occur free i.e., $FV(\varphi) = \emptyset$, where FV is defined recursively on the syntax, as usual. We define $\forall X \, . \, \varphi \triangleq \neg\exists X \, . \, \neg\varphi$, $\varphi_1 \vee \varphi_2 \triangleq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, and $\varphi_1 \rightarrow \varphi_2 \triangleq \neg\varphi_1 \vee \varphi_2$. The set of all **wAL** formulas over the alphabet Σ is formally denoted by **wAL**$[\Sigma]$.

A **wAL** formula is interpreted with respect to a heap $\mathcal{M}$ and a valuation ν assigning free variables to languages. The only non-standard operator is the modality $\langle X \rangle \rho_1$, where X is bound to denote a heap entity which intersects (the interpretation of) ρ_1. As a consequence of the syntactic restriction imposed on the existential quantifier, all variables in a closed formula are bound to heap entities[1]. A heap $\mathcal{M}$ is said to be a *model* for a closed **wAL** formula φ if and only if $[\![\varphi]\!]_{\mathcal{M},\lambda X.\perp} = 1$. In case where φ has at least one model, it is said to be *satisfiable*.

Expressing path properties The following properties describe the existence of paths in the structure. We consider the predicate $reach(X, Y) = \langle Y \rangle X\Sigma^+$ stating that node Y is reachable from node X by some non-empty path. A node Y is said to be *next* to a node X if the following predicate holds: $next(X, Y) = \langle Y \rangle X\Sigma \wedge \forall Y' \, . \, \langle Y' \rangle X\Sigma \rightarrow Y = Y'$. In other words, if Y is the

[1]This syntactic restriction on the quantification domain was mainly suggested by the fact that, allowing quantification over $\mathcal{P}(\Sigma^*)$ makes the logic undecidable even when modalities are not used at all in formulas. A formal proof will be included in an extended version of this paper.

$$\begin{array}{rcl}\rho & ::= & v \in \Sigma \mid X \in Var \mid \Sigma \mid \rho_1 \cdot \rho_2 \mid \rho^* \mid \rho_1 \cup \rho_2 \mid \rho_1 \cap \rho_2 \mid \bar{\rho} \mid \rho_1^{-1}\rho_2 \\ \varphi & ::= & \rho_1 = \rho_2 \mid \langle X \rangle \rho_1 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists X \,.\, \varphi\langle X \rangle\end{array}$$

$$\begin{array}{rcl}\mathcal{M} & \in & \mathcal{H}(\Sigma),\ \nu : Var \to \mathcal{P}(\Sigma^*) \\ [\![\rho]\!]_\nu & \triangleq & \rho[\nu(FV(\rho))/FV(\rho)] \\ [\![\rho_1 = \rho_2]\!]_{\mathcal{M},\nu} = 1 & \iff & [\![\rho_1]\!]_\nu = [\![\rho_2]\!]_\nu \\ [\![\langle X \rangle \rho_1]\!]_{\mathcal{M},\nu} = 1 & \iff & \nu(X) \in \mathcal{M} \text{ and } \nu(X) \cap [\![\rho_1]\!]_\nu \neq \emptyset \\ [\![\exists X\ [\varphi]]\!]_{\mathcal{M},\nu} = 1 & \iff & \exists\, \rho \in \mathcal{P}(\Sigma^*) \,.\, [\![\varphi]\!]_{\mathcal{M},\ [X \to \rho]\nu} = 1\end{array}$$

Figure 1: Weak Alias Logic

only neighbor of X. A path from X to Y is *linear* if there is no branching i.e., if all the nodes on the path have only one successor:

$$\begin{array}{rcl} linear(X, Y) & = & reach(X, Y) \wedge \forall Z \,.\, \neg Z = Y \wedge \big(X = Z \vee reach(X, Z)\big) \wedge \\ & & reach(Z, Y) \to \exists Z' \,.\, \neg Z' = Z \wedge next(Z, Z')\end{array}$$

The existence of a cycle containing both X and Y is given by the following predicate: $cycle(X, Y) = reach(X, Y) \wedge reach(Y, X)$, and if we are only interested by stating the existence of a cycle containing X, the following can be used: $\exists Y \,.\, cycle(X, Y)$.

Expressing recursive data structures The **wAL** logic can describe the shapes of most typical recursive data structures used in programming languages with dynamic memory allocation: lists, trees, dags, etc. For instance, non-cyclic simply-linked lists pointed to by the $head$ variable and using the $next$ field as forward selector, are being described by the following predicate:

$$nclist(head) \ = \ \forall X \,.\, \langle X \rangle head \to \exists Y \,.\, \langle Y \rangle X next^* \wedge linear(X, Y) \wedge \neg cycle(Y, Y)$$

Doubly-linked lists pointed to by the *head* variable and using the *next* and *prev* field pointers as forward and backward selectors, respectively, can be captured by the following:

$$\begin{array}{rcl} dlist(head, next, prev) & = & \forall X, Y \exists Z \,.\, (\langle X \rangle head \Rightarrow \neg\langle Y \rangle X prev) \wedge \\ & & (\langle Z \rangle X next \Rightarrow X \neq Z \wedge \langle X \rangle Z prev)\end{array}$$

Some data structures, such as trees, require the absence of sharing. A sharing predicate expressing that X and Y belong to two structures that share some node can be given as: $share(X, Y) = \exists Z \,.\, reach(X, Z) \wedge reach(Y, Z)$. A tree structure pointed to by a variable *root* is described by the following formula:

$$tree(root) = \forall X . \langle X \rangle root \to \forall Y, Z \,.\, (reach(X, Y) \wedge reach(X, Z)) \to \neg share(Y, Z))$$

A dag structure in which every node is reachable from a *root* variable is given by the following formula. Note the absence of cycles between any two nodes in the structure.

$$dag(root) = \exists X \,.\, \langle X \rangle root \to \forall Y, Z \,.\, reach(X, Y) \wedge reach(X, Z) \to \neg cycle(Y, Z)$$

2.1 Undecidability of wAL

The result of this section comes with no surprise, in the light of similar undecidability results for logics able to express graph properties such as e.g, the logic of Bunched Implications (BI) [5], and Monadic Second-Order Logic of graphs [7]. Given along the same lines as the undecidability proof for BI [5], our proof for **wAL** relies on a classical result in finite model theory [9], namely that the first order logic interpreted over finite structures is undecidable.

Given a vocabulary $\mathcal{V}$ of relation symbols, let FO$[\mathcal{V}]$ be the set of first-order formulas with symbols from $\mathcal{V}$. For each relation symbol $R \in \mathcal{V}$, let $\#(R)$ denote its *arity* i.e., its number of arguments. Let $\mathcal{V} = \{R_1, \ldots, R_n\}$ for the rest of this section. We interpret first-order formulas over structures $\mathcal{A} = \langle A, R_1^{\mathcal{A}}, \ldots, R_n^{\mathcal{A}} \rangle$, where A is the *universe* and $R_i^{\mathcal{A}} \subseteq A^{\#(R_i)}$, $1 \leq i \leq n$ are the *interpretations* of the relation symbols from $\mathcal{V}$ over A. A structure is said to be *finite* if and only if its universe is finite. Given a valuation $v : FV(\varphi) \to A$ of the free variables in a formula $\varphi \in$ FO$[\mathcal{V}]$, we denote by $[\![\varphi]\!]_{\mathcal{A},v}$ the interpretation of φ in $\mathcal{A}$. We say that $\mathcal{A}$ is a *model* of a closed first-order formula φ if and only if $[\![\varphi]\!]_{\mathcal{A},\lambda X.\perp} = 1$. It is known that the problem of finding a finite model for a closed FO$[\mathcal{V}]$ formula is undecidable [9]:

Theorem 1 (Trahtenbrot's Theorem) *Let $\mathcal{V}$ be a vocabulary with at least one symbol of arity two or more. Then the set $Sat[\mathcal{V}] \triangleq \{\varphi \in FO[\mathcal{V}] \mid FV(\varphi) = \emptyset,\ \varphi$ has a finite model $\}$ is not decidable.*

Given an arbitrary first order formula, we shall translate it into a **wAL** formula such that satisfiability is strongly preserved by the translation. Considering that $\mathcal{V} = \{R_1, \ldots, R_n\}$, we define $\Sigma_{\mathcal{V}} = \{\alpha_{i1}, \ldots, \alpha_{i\#(R_i)}, \beta_i \mid 1 \leq i \leq n\} \cup \{\gamma\}$. That is, for each relation symbol of arity k we consider k different α-symbols and a β-symbol in $\Sigma_{\mathcal{V}}$. The translation is given by the recursive function $\Theta : FO[\mathcal{V}] \to \mathbf{wAL}[\Sigma_{\mathcal{V}}]$, defined as:

$$\Theta(R_k(X_1, \ldots, X_{\#(R_k)})) \triangleq \exists X \,.\, \langle X \rangle \Sigma^* \wedge \bigwedge_{i=1}^{\#(R_k)} \langle X_i \rangle X \alpha_{ki}$$

$$\Theta(X = Y) \triangleq X = Y \qquad \Theta(\varphi_1 \wedge \varphi_2) \triangleq \Theta(\varphi_1) \wedge \Theta(\varphi_2)$$

$$\Theta(\neg\varphi) \triangleq \neg\Theta(\varphi) \qquad \Theta(\exists X \,.\, \varphi) \triangleq \exists X \,.\, \langle X \rangle \Sigma^* \wedge \Theta(\varphi)$$

Note that the translation of a closed first-order formula respects the syntactic constraints of **wAL**, that each quantified variable must occur inside the brackets of a modality, and that only a variable can occur on this position. Moreover, a closed first-order formula translates into a closed **wAL** formula. Now it remains to be shown that the translation strongly preserves satisfiability. We remind that satisfiability for **wAL** is implicitly defined on finite models (Definition 1). Due to space constraints, all proofs are deferred to Appendix **??**.

Lemma 1 *A closed first-order formula φ is finitely satisfiable if and only if $\Theta(\varphi)$ is satisfiable.*

Proof: For clarity, we define here the semantics of first order logic:

$$[\![X = Y]\!]_{\mathcal{A},v} = 1 \iff v(X) = v(Y)$$

$$[\![R(X_1, \ldots, X_{\#(R)})]\!]_{\mathcal{A},v} = 1 \iff (v(X_1), \ldots, v(X_{\#(R)})) \in R^{\mathcal{A}}$$

$$[\![\exists X \,.\, \varphi]\!]_{\mathcal{A},v} = 1 \iff [\![\varphi]\!]_{\mathcal{A},[X \to a]v} = 1 \text{ for some } a \in A$$

The interpretation of negation and conjunction are standard.
"$\Rightarrow$" Let $\mathcal{A} = \langle A, R_1^{\mathcal{A}}, \dots R_n^{\mathcal{A}} \rangle$ be a finite structure and let $A = \{a_1, \dots, a_N\}$. We define a heap $\mathcal{M}_{\mathcal{A}}$ as follows:

- for each $a_i \in A$, $1 \le i \le N$ there is a node $X \in \mathcal{M}_{\mathcal{A}}$ such that $\gamma^i \in X$. Let $\mu : A \to \mathcal{M}_{\mathcal{A}}$ be the injective function that maps a_i to the corresponding X i.e., for all $1 \le i \le N$ we have $\gamma^i \in \mu(a_i)$.
- for each tuple $(a_{i1}, \dots, a_{i\#(R_i)})_j \in R_i^{\mathcal{A}}$ there is a node $\{\beta_i^j\} \in \mathcal{M}_{\mathcal{A}}$. Moreover, we have, for all $1 \le k \le \#(R_i)$, $\beta_i^j \alpha_{ik} \in \mu(a_{ik})$.
- nothing else is in $\mathcal{M}_{\mathcal{A}}$.

It is easily checked that indeed $\mathcal{M}_{\mathcal{A}}$ is a heap, according to Definition 1. Next we prove that, for all formulas $\varphi \in$ FO[$\mathcal{V}$] and for all valuations $v : FV(\varphi) \to A$, $[\![\varphi]\!]_{\mathcal{A},v} = [\![\Theta(\varphi)]\!]_{\mathcal{M}_{\mathcal{A}},\mu\circ v}$, by induction on the structure of φ. The most interesting cases are:

- $\varphi = R_k(X_1, \dots, X_{\#(R_i)})$ for some $R_k \in \mathcal{V}$, $1 \le k \le n$. Then $[\![\varphi]\!]_{\mathcal{A},v} = 1$ iff $(v(X_1), \dots, v(X_{\#(R_k)})) \in R_i^{\mathcal{A}}$. Assuming now that $[\![\varphi]\!]_{\mathcal{A},v} = 1$, from the definition of $\mathcal{M}_{\mathcal{A}}$, we have $\{\beta_i^j\} \in \mathcal{M}_{\mathcal{A}}$ for some j. Now, it is easy to verify that:

  $$[\![\langle X \rangle \Sigma^* \wedge \bigwedge_{i=1}^{\#(R_k)} \langle X_i \rangle X \alpha_{ki}]\!]_{\mathcal{M}_{\mathcal{A}},[X \to \{\beta_i^j\}](\mu \circ v)} = 1$$

  Thus we obtain $[\![\Theta(\varphi)]\!]_{\mathcal{M}_{\mathcal{A}},\mu\circ v} = 1$. For the other direction, assume that $[\![\Theta(\varphi)]\!]_{\mathcal{M}_{\mathcal{A}},\mu\circ v} = 1$. Then, for a certain $X \in \mathcal{M}_{\mathcal{A}}$ we have, for all $1 \le i \le \#(R_k)$:

  $$X \alpha_{ki} \subseteq \mu(v(X_i))$$

  By the definition of $\mathcal{M}_{\mathcal{A}}$, the only choice for X is $\{\beta_k^j\}$ for some j. Hence $(v(X_1), \dots, v(X_{\#(R_k)})) \in R_k^{\mathcal{A}}$, which leads to $[\![\varphi]\!]_{\mathcal{A}} = 1$.
- $\varphi = \exists X \,.\, \psi$. Then $[\![\varphi]\!]_{\mathcal{A},v} = 1$ iff $[\![\psi]\!]_{\mathcal{A},[X \to a_i]v} = 1$ for some $a_i \in A$. By the induction hypothesis, this is equivalent to $[\![\Theta(\psi)]\!]_{\mathcal{A},\mu\circ([X \to a_i]v)} = [\![\Theta(\psi)]\!]_{\mathcal{M}_{\mathcal{A}},[X \to \mu(a_i)](\mu\circ v)} = 1$. From the definition of $\mathcal{M}_{\mathcal{A}}$ we have $\gamma^i \in \mu(a_i) \Rightarrow [\![\langle \mu(a_i) \rangle \Sigma^*]\!]_{\mathcal{M}_{\mathcal{A}},\mu\circ v} = 1$, therefore $[\![\langle X \rangle \Sigma^* \wedge \Theta(\psi)]\!]_{\mathcal{M}_{\mathcal{A}},[X \to \mu(a_i)](\mu\circ v)} = 1$. Thus $[\![\Theta(\varphi)]\!]_{\mathcal{M}_{\mathcal{A}},\mu\circ v} = 1$.

"$\Leftarrow$" Let $\mathcal{M}$ be a heap. We define a structure $\mathcal{A}_{\mathcal{M}} = \langle \mathcal{A}_{\mathcal{M}}, R_1^{\mathcal{A}_{\mathcal{M}}}, \dots, R_n^{\mathcal{A}_{\mathcal{M}}} \rangle$ such that, for all formulas $\varphi \in$ FO[$\mathcal{V}$] and all valuations $\nu : Var \to \mathcal{M}$ we have $[\![\Theta(\varphi)]\!]_{\mathcal{M},\nu} = [\![\varphi]\!]_{\mathcal{A}_{\mathcal{M}},\nu}$. The definition of $\mathcal{A}_{\mathcal{M}}$ is as follows:

- $A_{\mathcal{M}} = \mathcal{M}$, and,
- for all $R_k \in \mathcal{V}$, $1 \le k \le n$, and all $X_1, \dots, X_{\#(R_k)} \in \mathcal{M}$ we have $(X_1, \dots, X_{\#(R_k)}) \in R_k^{\mathcal{A}_{\mathcal{M}}}$ iff $X \alpha_{ki} \subseteq X_i$, $1 \le i \le \#(R_k)$, for some $X \in \mathcal{M}$.

The proof follows by induction on the structure of φ. The most interesting cases are:

- $\varphi = R_k(X_1, \ldots, X_{\#(R_k)})$ for some $R_k \in \mathcal{V}$, $1 \leq k \leq n$. We calculate:

$$\begin{array}{rcl} [\![\Theta(\varphi)]\!]_{\mathcal{M},\nu} & = & [\![\langle X\rangle \Sigma^* \wedge \bigwedge_{i=1}^{\#(R_k)} \langle X_i\rangle X \alpha_{ki}]\!]_{\mathcal{M},\nu} = 1 \\ & \iff & \nu(X)\alpha_{ki} \subseteq \nu(X_i) \quad \forall 1 \leq i \leq \#(R_k), \text{ for some } \mathcal{X} \in \mathcal{M} \\ & \iff & (\nu(X_1), \ldots, \nu(X_\#(R_k))) \in R_k^{\mathcal{A}_\mathcal{M}} \\ & \iff & [\![\varphi]\!]_{\mathcal{A}_\mathcal{M},\nu} = 1 \end{array}$$

- $\varphi = \exists X \,.\, \psi$. We calculate:

$$\begin{array}{rcl} [\![\Theta(\varphi)]\!]_{\mathcal{M},\nu} & = & [\![\exists X \,.\, \langle X\rangle \Sigma^* \wedge \Theta(\psi)]\!]_{\mathcal{M},\nu} = 1 \\ & \iff & [\![\Theta(\psi)]\!]_{\mathcal{M},[X \to \mathcal{X}]\nu} = 1 \quad \text{for some } \mathcal{X} \in \mathcal{M} \\ & \iff & [\![\psi]\!]_{\mathcal{A}_\mathcal{M},[X \to \mathcal{X}]\nu} = 1 \quad \text{(induction hypothesis)} \\ & \iff & [\![\varphi]\!]_{\mathcal{A}_\mathcal{M},\nu} = 1 \end{array}$$

□

Considering for the moment that the alphabet Σ is sufficiently large to code the vocabulary $\mathcal{V}$ of a given first order logic, Theorem 1 and Lemma 1 lead immediately to the following result.

Theorem 2 *For a sufficiently large alphabet Σ, the set $Sat[\Sigma] \triangleq \{\varphi \in$ **wAL**$[\Sigma] \mid FV(\varphi) = \emptyset,\ \varphi$ has a model $\}$ is not recursive.*

Since Theorem 1 holds for vocabularies containing at least one relation symbol of arity two, by the definition of $\Sigma_\mathcal{V}$ it follows that Theorem 2 holds for generic heaps over alphabets of size at least four. Here, a more refined heap model could provide us with more intuition in identifying classes of heaps over which the satisfiability problem becomes decidable. For instance, considering $\Sigma = \Pi \cup \Omega$, $\Pi \cap \Omega = \emptyset$, $\|\Omega\| = 1$ and all heaps of the form $\mathcal{M} \subseteq \mathcal{P}(\Pi \times \Omega^*)$ i.e., heaps consisting only of (possibly circular) singly linked lists. In this simple case, we propose to revisit the decidability of the satisfiability problem for **wAL**.

In order to show that the satisfiability problem for **wAL** is recursively enumerable, let us first consider the *model checking* problem. The model checking problem asks whether a given heap $\mathcal{M}$ is a model for a formula ψ. This problem is decidable, by the fact that any heap model is finite. The interested reader is referred to [4] for an algorithm. But the set $\mathcal{H}(\Sigma)$ of all heaps over a finite alphabet is enumerable. Hence, if a given formula ψ is satisfiable, an algorithm that enumerates all models $\mathcal{M}_1, \mathcal{M}_2, \ldots$, testing whether each $\mathcal{M}_i$ is a model of ψ, will eventually stop.

Lemma 2 *For every finite Σ, the set $Sat[\Sigma]$ is recursively enumerable.*

An interesting open problem is then how to find useful semi-decision procedures for **wAL**.

3 Propositional Alias Logic

The negative result from the previous section motivates the search for decidable subsets of **wAL** that are yet able to express meaningful properties of heaps. One basic property encountered in many applications is *data sharing*. In this section we define a simpler logic based directly on the notion of *aliasing* of finite heap access paths. The rest of this paper is concerned with the study of Propositional Alias Logic (**pAL**) from three perspectives: proof theory, automated reasoning and program logic. The ability of **pAL** to express other heap properties besides aliasing, is also investigated.

$$
\begin{array}{rcl}
w & := & v \in \Sigma \mid w_1 \cdot w_2 \mid w_1^{-1} w_2 \\
\varphi & := & w_1 \leq w_2 \mid w_1 \Diamond w_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bot
\end{array}
$$

$$
[\![w_1 \Diamond w_2]\!]_{\mathcal{M}} \;=\; 1 \iff \exists X \in \mathcal{M} \,.\, w_1, w_2 \in X
$$

Figure 2: Propositional Alias Logic

Figure 2 defines the abstract syntax (upper part) and the semantics (lower part) of **pAL**. The terms are finite words over an alphabet Σ, with $w_1^{-1}w_2$ being the suffix of w_2 that, concatenated with w_1, yields w_2, if such suffix exists, or the empty word ϵ, otherwise. The atomic propositions are the *prefix test* ($w_1 \leq w_2$) and the *alias proposition* ($w_1 \Diamond w_2$). Formulas are built from atomic propositions connected with the propositional operators $\wedge$ and $\neg$. In the syntax definition, $\bot$ denotes the *false* literal[2]. The set of all **pAL** formulas over the alphabet Σ is formally denoted by $\mathbf{pAL}[\Sigma]$. We define the *size* of a **pAL** formula to be:

$$
|w_1 \leq w_2| = |w_1 \Diamond w_2| \triangleq 1, \quad |\varphi_1 \wedge \varphi_2| \triangleq |\varphi_1| + |\varphi_2| + 1, \quad |\neg\varphi| \triangleq |\varphi| + 1
$$

The semantics of **pAL** is defined with respect to a heap $\mathcal{M}$. An alias proposition $w_1 \Diamond w_2$ is true if and only if there exists an element of $\mathcal{M}$ such that both terms w_1, w_2 belong to it. Note that, since $\mathcal{M} \subseteq \mathcal{P}(\Sigma^+)$, if either one of the terms is ϵ, the alias proposition is false. The intended meaning of $w \Diamond w$ for some $w \in \Sigma^+$, is to say that w is a well-defined path in the heap. The following semantic equivalence is a trivial check:

$$
w_1 \Diamond w_2 \iff \exists X \,.\, \langle X \rangle w_1 \wedge \langle X \rangle w_2 \tag{1}
$$

The prefix relation $w_1 \leq w_2$ can be encoded in **wAL** as $w_1^{-1}w_2 \neq \emptyset^*$, where $\epsilon \triangleq \emptyset^*$ is a possible definition of the empty word in **wAL**. These considerations justify the fact that **pAL** is a subset of **wAL**. The embedding is proper ($\mathbf{pAL}[\Sigma] \subset \mathbf{wAL}[\Sigma]$), since e.g. reachability and linearity are not expressible in **pAL**.

[2]False could have been defined as $\varphi \wedge \neg\varphi$ for an arbitrary formula φ. However an explicit definition is preferred for the purposes of the proof theoretic system of Section 3.1.

3.1 Natural Deduction System

This section introduces a natural deduction system [25] for **pAL** that proves to be a useful tool in reasoning about aliases. Although later in this paper we adopt the automated reasoning view, as opposed to the proof theoretic, a number of results from this sections are used in the rest of the paper. The system (Figure 3) is that of propositional calculus à la Gentzen (rules ∧E, ∧I, ¬E, ¬I, ⊥E, ⊥I) to which we add three rules concerning only alias propositions (sufE, sufI and sym). For these rules we take $\Gamma \subseteq \mathbf{pAL}[\Sigma]$, $x, y, z \in \Sigma^+$ and $t \in \Sigma^*$.

$$\frac{xt\diamond y}{x\diamond x}\ (\text{sufE}) \qquad \frac{x\diamond y \quad yt\diamond z}{xt\diamond z}\ (\text{sufI}) \qquad \frac{x\diamond y}{y\diamond x}\ (\text{sym})$$

$$\frac{\varphi \wedge \psi}{\varphi}\ (\wedge E) \qquad \frac{\varphi \quad \psi}{\varphi \wedge \psi}\ (\wedge I) \qquad \frac{\bot}{\varphi}\ (\bot E) \qquad \frac{\varphi \quad \neg\varphi}{\bot}\ (\bot I)$$

$$\frac{\Gamma, \neg\varphi \vdash \bot}{\Gamma \vdash \varphi}\ (\neg E) \qquad \frac{\Gamma, \varphi \vdash \bot}{\Gamma \vdash \neg\varphi}\ (\neg I)$$

Figure 3: Natural Deduction System for **pAL**

The natural deduction system presented in Figure 3 exhibits a number of interesting properties: it is sound, complete and, all proofs of alias propositions can be given in a normal form. To formalize these notions, we need further notation. Given a set of formulas Γ and ψ a formula we say that $\Gamma \models \psi$ if and only if, for each heap $\mathcal{M}$ we have that, if $[\![\varphi]\!]_{\mathcal{M}} = 1$ for all $\varphi \in \Gamma$, then $[\![\psi]\!]_{\mathcal{M}} = 1$. If p is an alias proposition, we say that $\Gamma \vdash_{PA} p$ if and only if there exists a derivation of p with premises in Γ that uses *only* the (sufI), (sufE) and (sym) rules. Otherwise, if ψ is any formula, we say that $\Gamma \vdash \psi$ if and only if there exists a derivation of ψ with premises in Γ. The set Γ is said to be *consistent* if and only if $\Gamma \nvdash \bot$. By $Th(\Gamma)$ we denote the *theory* of Γ i.e., the set of all formulas that can be deduced from it. Formally, we denote $Th(\Gamma) \triangleq \{\varphi \mid \Gamma \vdash \varphi\}$.

Lemma 3 *Given a set of formulas Γ and an alias proposition p, a proof for $\Gamma \vdash_{PA} p$ is said to be in* normal form *if and only if no* (sufI) *rule is used to derive the premise of a* (sufE) *rule. If $\Gamma \vdash_{PA} p$ then there exists an normalized proof with premises in Γ and conclusion p.*

Proof: We shall provide a procedure to replace every derivation in which the (sufI) rule is used in the premise of the (sufE) rule. It is sufficient to consider subtrees in which the rules are adjacent and iterate the procedure to modify the entire derivation. There are two cases to be considered, based on whether a (sym) rule occurs or not on the branch leading from the premise of (sufI) to the conclusion of (sufE). Since the (sufI) and (sufE) rules are adjacent, at most one occurrence need to be considered (two or more subsequent (sym) rules are redundant). Let us first consider the case in which the (sym) rule does occur:

$$\cfrac{\cfrac{\cfrac{x\diamond y \quad yt\diamond z}{xt\diamond z}\ (\text{sufI})}{z\diamond xt}\ (\text{sym})}{z'\diamond z'}\ (\text{sufE})$$

where $z' \leq z$. This proof is replaced by:

$$\dfrac{\dfrac{yt\diamond z}{z\diamond yt}\,(\text{sym})}{z'\diamond z'}\,(\text{sufE})$$

Second, consider the following proof:

$$\dfrac{\dfrac{x\diamond y \quad yt\diamond z}{xt\diamond z}\,(\text{sufI})}{x'\diamond x'}\,(\text{sufE})$$

where $x' \leq xt$. There are two cases:

- if $x' \leq x$ then the proof can be replaced by

$$\dfrac{x\diamond y}{x'\diamond x'}\,(\text{sufE})$$

- else, if $x < x'$, let $x' = xt_0$ and $t = t_0 t_1$ for some $t_0, t_1 \in \Sigma^*$. The proof is replaced by:

$$\dfrac{\dfrac{\dots}{xt_0\diamond yt_0}\,(\text{sufI}) \qquad \dfrac{\dfrac{x\diamond y \quad \dfrac{yt\diamond z}{yt_0\diamond yt_0}\,(\text{sufE})}{xt_0\diamond yt_0}\,(\text{sufI})}{yt_0\diamond xt_0}\,(\text{sym})}{x'\diamond x'}\,(\text{sufI})$$

  Notice that, in the second case we need to duplicate the entire proof tree for $xt_0\diamond yt_0$. For clarity reasons this has been omitted.

$\square$

Given a finite set of alias propositions, one attempts to construct a heap that satisfies the entire set. The existence of such a model is guaranteed by the following lemma. Note that, for arbitrary sets of formulas, the existence of a model occurs as a consequence of the consistency or downward closure[3] properties.

Lemma 4 *Let Γ be a set of formulas containing a finite number of alias propositions, $\approx_\Gamma \subseteq \Sigma^+ \times \Sigma^+$ be a relation on finite sequences, defined as $x \approx_\Gamma y$ if and only if $\Gamma \vdash_{PA} x\diamond y$, and H_Γ be the set $\{x \mid x \approx_\Gamma x\}$. Then $\approx_\Gamma$ is a total equivalence relation on H_Γ, and the quotient $H_{\Gamma/\approx_\Gamma}$ is a heap. Moreover, $\|H_{\Gamma/\approx_\Gamma}\| \leq k \cdot \|\Gamma\|$, where $k \in \mathbb{N}$ is a constant.*

Proof: First, $\approx_\Gamma$ is total on H_Γ by definition. Reflexivity follows immediately, also by definition of H_Γ. Symmetry is ensured by rule (sym) and transitivity by (sufI) taking $t = \epsilon$. The following claim is sufficient to show index finiteness.

[3]Definition 3 in Section 4.

Claim 1 *For any $x, y \in \Sigma^+$, if $x \approx_\Gamma y$ then there exists $z, t \in H_\Gamma$ such that $x \approx_\Gamma z$, $z \leq t$ and $t \Diamond u \in \Gamma$ for some $u \in \Sigma^+$.*

Proof: By induction on the normalized proof tree for $x \approx_\Gamma y$. The existence of such a proof is guaranteed by Lemma 3.

- if $x \approx_\Gamma y$ because $x \Diamond y \in \Gamma$, trivially take $z = t = x$.
- if $x \approx_\Gamma y$ because $y \approx_\Gamma x$ (sym), by induction hypothesis there exists $z, t \in H$ such that $y \approx_\Gamma z$, $z \leq t$ and $t \Diamond u \in S$. Since $x \approx_\Gamma y$, we have $x \approx_\Gamma z$.
- if $x \approx_\Gamma y$ because $x = x'x''$, $x' \approx_\Gamma y'$ and $y'x'' \approx_\Gamma y$ (sufI), by induction hypothesis we have $y'x'' \approx_\Gamma z$, $z \leq t$ and $t \Diamond u \in \Gamma$ for some $z, t \in H$. Hence $y \approx_\Gamma z$ and, since $x \approx_\Gamma y$, $x \approx_\Gamma z$.
- if $x = y$ and $x \approx_\Gamma x$ because $xx' \approx_\Gamma y'$ (sufE). Since we have considered a normalized proof tree, all rules used above are either (sufE) or (sym). Then t can be found following the unique path to the assumption $t \Diamond u \in \Gamma$. Depending on whether (sym) has been applied an even (odd) number of times along this branch, take $z = x$ ($z = y$ where $x \approx_\Gamma y$ is the premise of (sym)).

□

Since the number of alias propositions is finite, and since each word occurring in a formula is finite, the number of equivalence classes of $\approx_\Gamma$ is finite. From this observation we immediately obtain that $\|H_{\Gamma/\approx_\Gamma}\| \leq k \cdot \|\Gamma\|$, taking $k = \max\{|x|, |y| \mid x \Diamond y \in \Gamma\}$. Next, we need to show that $H_{\Gamma/\approx_\Gamma}$ satisfies the three conditions from Definition 1.

(C1,2) By the fact that $\approx_\Gamma$ is an equivalence relation and $H_{\Gamma/\approx_\Gamma}$ is a partition of H_Γ.

(C3) Let $x \in X$ for some $X \in H_{\Gamma/\approx_\Gamma}$ and $x = yz$. If $x = yz \in H_\Gamma$ then $\Gamma \vdash_{PA} yz \Diamond yz$ therefore, by (sufE) we have $y \in H_\Gamma$. Hence $y \in Y$ for some $Y \in H_{\Gamma/\approx_\Gamma}$. Let $y' \in Y$. Then $y' \Diamond y$ and, by (sufI), we have $y'z \Diamond yz$. Hence $y'z \approx_\Gamma yz$ which leads to $y'z \in X$.

□

3.1.1 Completeness of the Natural Deduction System

This section is concerned with showing soundness and completeness of the natural deduction system for **pAL**. The following lemma proves soundness for the rules in Figure 3.

Lemma 5 (Soundness) *Given Γ a set of formulas and ψ a formula, $\Gamma \vdash \psi$ implies $\Gamma \models \psi$.*

Proof: By induction on the derivation tree with premises in Γ and conclusion ψ. We consider only the rules (sufI), (sufE) and (sym). Let $\mathcal{M}$ be any heap.

(sufI) Suppose that $[\![x \Diamond y]\!]_\mathcal{M} = [\![yt \Diamond z]\!]_\mathcal{M} = 1$ and prove that $[\![xt \Diamond z]\!]_\mathcal{M} = 1$. From the definition of alias, there exists sets $X, Y \in \mathcal{M}$ such that $x, y \in X$ and $yt, z \in Y$. By (C3) we have $Xt \subseteq Y$, hence $xt \in Y$. This leads to $[\![xt \Diamond z]\!]_\mathcal{M} = 1$.

(sufE) Suppose that $[\![xt\diamond y]\!]_{\mathcal{M}} = 1$. Then there exists a set $X \in \mathcal{M}$ such that $xt \in X$. By (C3) there exists a set $Y \in \mathcal{M}$ such that $x \in Y$. Hence $[\![x\diamond x]\!]_{\mathcal{M}} = 1$.

(sym) Immediate from $[\![x\diamond y]\!]_{\mathcal{M}} = [\![y\diamond x]\!]_{\mathcal{M}}$.

□

The rest of this section is concerned with the dual implication. We recall the notion of consistency, which plays here a central role.

Definition 2 (Maximally Consistent Set) *A set of formulas* Γ *is said to be* maximally consistent *if and only if:*

- Γ *is consistent.*
- *If* $\Gamma \subseteq \Delta$ *and* Δ *is consistent then* $\Gamma = \Delta$.

There are three classical properties of maximally consistent sets [25]. Let Γ be a maximally consistent set. Then for all formulas φ we have $\varphi \in \Gamma$ if and only if $\neg\varphi \notin \Gamma$. Second, all maximally consistent sets are closed under derivation i.e. if $\Gamma \vdash \varphi$ then $\varphi \in \Gamma$. Third and last, each consistent set is contained in at least one maximally consistent set.

Next, we define the *closure* of Γ to be the theory of Γ, to which we have added the negation of all alias propositions that cannot be deduced from Γ. Formally we have $Cl(\Gamma) \triangleq Th(\Gamma \cup \{\neg(x\diamond y) \mid \Gamma \not\vdash x\diamond y\})$.

Lemma 6 *Given a consistent set* Γ*, we have that* $Cl(\Gamma)$ *is maximally consistent.*

Proof: We first show that $Cl(\Gamma)$ is consistent. This is done by showing the consistency of $S = \Gamma \cup \{\neg(x\diamond y) \mid \Gamma \not\vdash x\diamond y\}$. Let $x_1\diamond\, y_1, x_2\diamond\, y_2 \ldots$ be a numbering of alias formulas. Then $S = \bigcup_{n\geq 1} S_n$ where S_i is $\Gamma\cup\{x_i\diamond\, y_i\}$ if $\Gamma \not\vdash x_i\diamond\, y_i$ and Γ otherwise. Since each S_i is consistent, we have that S is consistent. Since $Cl(\Gamma)$ is consistent, it is contained in a maximally consistent set Δ. Suppose $Cl(\Gamma) \subset \Delta$ and let $\varphi \in \Delta \setminus Cl(\Gamma)$ be a formula such that its size $|\varphi|$ is minimal. We distinguish the following cases:

- $\varphi = x\diamond y$ for some $x, y \in \Sigma^+$. Two situations are possible:
  - if $\Gamma \vdash x\diamond y$ then $\varphi \in Cl(\Gamma)$, contradiction.
  - else, if $\Gamma \not\vdash x\diamond y$ then $\neg\varphi \in Cl(\Gamma) \subset \Delta$, hence Δ is not consistent, contradiction.
- $\varphi = \neg(x\diamond y)$ is analogous.
- $\varphi = \varphi_1 \wedge \varphi_2$. Since Δ is maximally consistent, thus closed under derivation, we have $\varphi_1, \varphi_2 \in \Delta$. Since $Cl(\Gamma) \not\vdash \varphi_1 \wedge \varphi_2$ then either $Cl(\Gamma) \not\vdash \varphi_1$ or $Cl(\Gamma) \not\vdash \varphi_2$. Hence either $\varphi_1 \in \Delta \setminus Cl(\Gamma)$ or $\varphi_2 \in \Delta \setminus Cl(\Gamma)$. But $|\varphi_1| < |\varphi|$ and $|\varphi_2| < |\varphi|$, which contradicts with the choice of φ.

- $\varphi = \neg(\varphi_1 \wedge \varphi_2)$. Since Δ is maximally consistent, hence closed under derivation, we obtain that either $\neg\varphi_1 \in \Delta$ or $\neg\varphi_2 \in \Delta$. To see this, assume $\neg\varphi_1, \neg\varphi_2 \notin \Delta$. Since Δ is maximally consistent, we have $\varphi_1, \varphi_2 \in \Delta$. Hence $\varphi_1 \wedge \varphi_2 \in \Delta$, which contradicts the fact that Δ is consistent. Also since $Cl(\Gamma) \not\vdash \neg(\varphi_1 \wedge \varphi_2)$ we have $Cl(\Gamma) \not\vdash \neg\varphi_1$ and $Cl(\Gamma) \not\vdash \neg\varphi_2$. Hence either $\neg\varphi_1$ or $\neg\varphi_2$ belongs to $\Delta \setminus Cl(\Gamma)$, and since $|\neg\varphi_1| < |\varphi|$ and $|\neg\varphi_2| < |\varphi|$, this contradicts with the choice of φ.

- $\varphi = \neg\neg\varphi_1$. Since Δ is maximally consistent, thus closed under derivation, we have $\varphi_1 \in \Delta$. Next we show $\varphi_1 \notin Cl(\Gamma)$, which contradicts the choice for φ, since $|\varphi_1| < |\varphi|$ Suppose $\varphi_1 \in Cl(\Gamma)$. Since $Cl(\Gamma)$ is closed under derivation we have $\neg\neg\varphi_1 \in Cl(\Gamma)$. But this contradicts with the fact that $\varphi \notin Cl(\Gamma)$.

□

Theorem 3 *Any consistent set of formulas containing a finite number of alias propositions is satisfiable.*

Proof: Let Γ be a consistent set and $H_{\Gamma/\approx_\Gamma}$ be the heap defined in the text of Lemma 4. We prove a stronger claim: for any formula φ, $[\![\varphi]\!]_{H_{\Gamma/\approx_\Gamma}} = 1 \iff \varphi \in Cl(\Gamma)$. Since $\Gamma \subseteq Cl(\Gamma)$ we have the result. By induction on the structure of φ we have:

- $\varphi = x\Diamond y$: $[\![x\Diamond y]\!]_{H_{\Gamma/\approx_\Gamma}} = 1 \iff \exists X \in H_{\Gamma/\approx_\Gamma}\ x, y \in X \iff x \approx y \iff x\Diamond y \in Cl(\Gamma)$.

- $\varphi = \neg\psi$: $[\![\neg\psi]\!]_{H_{\Gamma/\approx_\Gamma}} = 1 \iff [\![\psi]\!]_{H_{\Gamma/\approx_\Gamma}} = 0$. By induction hypothesis, this is equivalent to $\psi \notin Cl(\Gamma)$ which is further equivalent to $\neg\psi \in Cl(\Gamma)$.

- $\varphi = \psi_1 \wedge \psi_2$: $[\![\psi_1 \wedge \psi_2]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$ if and only if $[\![\psi_1]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$ and $[\![\psi_2]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$. By induction hypothesis $\psi_1, \psi_2 \in Cl(\Gamma)$, hence $\psi_1 \wedge \psi_2 \in Cl(\Gamma)$ by ($\wedge$I).

□

Corollary 1 *Given a set of formulas Γ and a formula ψ, we have $\Gamma \models \psi$ if and only if $\Gamma \vdash \psi$.*

Proof: "$\Leftarrow$" By Lemma 5. "$\Rightarrow$" Assume $\Gamma \not\vdash \psi$. Then $\Gamma \cup \{\neg\psi\}$ is consistent. By Theorem 3, there exists a model $\mathcal{M}$ such that $[\![\varphi]\!]_{\mathcal{M}} = 1$ for all $\varphi \in \Gamma \cup \{\neg\psi\}$. Then for all $\varphi \in \Gamma$, $[\![\varphi]\!]_{\mathcal{M}} = 1$ and $[\![\psi]\!]_{\mathcal{M}} = 0$. Hence $\Gamma \not\models \psi$. □

3.2 Expressiveness of pAL

In this section we investigate the expressiveness of the **pAL** language. We show that any finite heap structure over a finite alphabet can be uniquely characterized by a **pAL** formula. As a consequence, any finite class of heap structures can be defined in **pAL**[4]. This extends our previous result in [2], that **pAL** has the power to distinguish between any two non-isomorphic heap

[4]Even if a **pAL** formula, e.g $x\Diamond y$, is in general satisfied by an infinite number of heaps.

configurations. However, the far more interesting question, of whether and how could **pAL** be extended to describe recursive data structures and still preserve decidability, is subject to ongoing and future work.

For the rest of this section, let $\mathcal{M} = \{X_1, \dots, X_n\}$ be a given heap. We shall define a formula $\phi_{\mathcal{M}}$ such that $[\![\phi_{\mathcal{M}}]\!]_{\mathcal{M}} = 1$ and, for any other heap $\mathcal{M}'$ such that $[\![\phi_{\mathcal{M}}]\!]_{\mathcal{M}'} = 1$, we have $\mathcal{M} = \mathcal{M}'$. For a finite word $w \in \Sigma^+$, we denote by $Pref(w)$ the set of all its prefixes, including w. For a set $X \in \mathcal{M}$, a word $w \in X$ is *elementary* if and only if it has at most two prefixes in X and at most one prefix in any other set $Y \in \mathcal{M}$, $Y \neq X$. Formally, we have $Elem_{\mathcal{M}}(X) \triangleq \{w \in X \mid \|Pref(w) \cap X\| \leq 2 \text{ and } \forall Y \neq X \ . \ \|Pref(w) \cap Y\| \leq 1\}$. An important property of the sets of elementary words is finiteness. This results as a consequence of the fact that both $\mathcal{M}$ and Σ are finite, since the length of any $w \in Elem_{\mathcal{M}}(X)$ is $|w| \leq \|\mathcal{M}\| + 1$, thus $\|Elem_{\mathcal{M}}(X)\| \leq \|\Sigma\|^{\|\mathcal{M}\|+1}$. A *dangling* word is a minimal undefined path in $\mathcal{M}$. Formally, we define $Dang_{\mathcal{M}}(X) = \{wa \mid w \in Elem_{\mathcal{M}}(X),\ a \in \Sigma,\ wa \notin \bigcup \mathcal{M}\}$. Since $Elem_{\mathcal{M}}$ and Σ are finite, so is $Dang_{\mathcal{M}}(X)$. With this notation, we define:

$$\begin{aligned}
\Gamma_{\mathcal{M}} \ \triangleq \ & \bigcup_{X \in \mathcal{M}} \{w \diamond w' \mid w, w' \in Elem_{\mathcal{M}}(X)\} \ \cup & (2)\\
& \bigcup_{X,Y \in \mathcal{M}, X \neq Y} \{\neg(w \diamond w') \mid w \in Elem_{\mathcal{M}}(X), w' \in Elem_{\mathcal{M}}(Y)\} \ \cup & (3)\\
& \bigcup_{X \in \mathcal{M}} \{\neg(w \diamond w) \mid w \in Dang_{\mathcal{M}}(X)\} \ \cup \{\neg(a \diamond a) \mid a \in \Sigma \setminus \bigcup \mathcal{M}\} & (4)
\end{aligned}$$

This set is constructed as follows: the first component (2) describes each object as a set of alias propositions composed of elementary sequences, the second component (3) distinguishes between objects using negated alias propositions and the third and fourth components (4) describe the dangling sequences. Notice that $\Gamma_{\mathcal{M}}$ is not minimal, since for instance in (3) it is sufficient to choose only one $w \in Elem_{\mathcal{M}}(X)$ and one $w' \in Elem_{\mathcal{M}}(Y)$. However, it is finite, according to our previous considerations. Intuitively, $\Gamma_{\mathcal{M}}$ contains all the necessary information to characterize $\mathcal{M}$, thus we shall take $\phi_{\mathcal{M}} \triangleq \bigwedge \Gamma_{\mathcal{M}}$. To show that $\mathcal{M}$ is a model of $\phi_{\mathcal{M}}$ is a trivial but tedious check. That it is indeed the only model, will be shown in the rest of this section.

Lemma 7 *Let $\mathcal{M}$ be a heap with $X \in \mathcal{M}$, and $\Gamma_{\mathcal{M}}$ be the characteristic set defined in the previous. Then the following hold:*

1. *for each $w \in X$ there exists $w_0 \in Elem_{\mathcal{M}}(X)$ such that $\Gamma_{\mathcal{M}} \vdash w \diamond w_0$.*

2. *for all $w \notin \bigcup \mathcal{M}$ we have $\Gamma_{\mathcal{M}} \vdash \neg(w \diamond w)$.*

3. *for any $x, y \in \Sigma^+$ we have $[\![x \diamond y]\!]_{\mathcal{M}} = 1 \Rightarrow \Gamma_{\mathcal{M}} \vdash x \diamond y$ and $[\![x \diamond y]\!]_{\mathcal{M}} = 0 \Rightarrow \Gamma_{\mathcal{M}} \vdash \neg(x \diamond y)$.*

Proof:

1. Let $w \in X$ be a sequence. For the purposes of this proof, we define the rank of w as:

$$rk(w) \triangleq \max(\|Pref(w) \cap X\| - 2, 0) + \sum_{Y \neq X} \max(\|Pref(w) \cap Y\| - 1, 0)$$

   The proof is by induction on $rk(w)$. For the base case $rk(w) = 0$ we have $w \in Elem_{\mathcal{M}}(X)$. Hence $w\Diamond w \in \Gamma_{\mathcal{M}}$. For the induction step, if $rk(w) > 0$ then either $\|Pref(w) \cap X\| > 2$ or $\|Pref(w) \cap Y\| > 1$ for some $Y \neq X$. In either case there exists two strict prefixes $x, y < w$ such that $x, y \in Z$ for some $Z \in \mathcal{M}$, possibly equal to X. W.l.o.g. assume $x < y$ and put $w = yw'$. Now $rk(x), rk(y), rk(xw') < rk(w)$ hence, by the inductive hypothesis there exists $x_0, y_0 \in Elem_{\mathcal{M}}(Z)$ and $w_0 \in Elem_{\mathcal{M}}(X)$ such that $\Gamma_{\mathcal{M}} \vdash x\Diamond x_0$, $\Gamma_{\mathcal{M}} \vdash y\Diamond y_0$ and $\Gamma_{\mathcal{M}} \vdash xw'\Diamond w_0$. Since, by definition of $\Gamma_{\mathcal{M}}$ we have $x_0\Diamond y_0 \in \Gamma_{\mathcal{M}}$, we obtain $\Gamma_{\mathcal{M}} \vdash w\Diamond w_0$ as follows:

$$\dfrac{\dfrac{\dfrac{x\Diamond x_0 \qquad \dfrac{x_0\Diamond y_0 \qquad \dfrac{y\Diamond y_0}{y_0\Diamond y}}{x_0\Diamond y}}{x\Diamond y}}{y\Diamond x} \qquad xw'\Diamond w_0}{w = yw'\Diamond w_0}$$

2. There are two possibilities: there exists $x < w$ such that $x \in X$ for some $X \in \mathcal{M}$. Let x be the longest such prefix. Then $w = xaw'$ for some $a \in \Sigma$, with $xa \notin \bigcup \mathcal{M}$. Using the first point, there exists $x_0 \in Elem_{\mathcal{M}}$ such that $\Gamma_{\mathcal{M}} \vdash x\Diamond x_0$. By definition of $\Gamma_{\mathcal{M}}$ we have $\neg(x_0a\Diamond x_0a) \in \Gamma_{\mathcal{M}}$. The result is obtained by the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x\Diamond x_0}{x_0\Diamond x} \qquad [xaw'\Diamond xaw']}{x_0aw'\Diamond xaw'}}{x_0a\Diamond x_0a} \qquad \neg(x_0a\Diamond x_0a)}{\bot}}{\neg(w = xaw'\Diamond xaw')}$$

   In the other case, when w has no prefix in $\bigcup \mathcal{M}$ we have $w = aw'$ with $a \in \Sigma \setminus \bigcup \mathcal{M}$. From the definition of $\Gamma_{\mathcal{M}}$ we have $\neg(a\Diamond a) \in \Gamma_{\mathcal{M}}$ and $\Gamma_{\mathcal{M}} \vdash \neg(w\Diamond w)$ follows.

3. For the first implication, if $[\![x\Diamond y]\!]_{\mathcal{M}} = 1$ then $x, y \in X$ for some $X \in \mathcal{M}$. Using the first point, there exist $x_0, y_0 \in Elem_{\mathcal{M}}(X)$ such that $\Gamma_{\mathcal{M}} \vdash x\Diamond x_0$ and $\Gamma_{\mathcal{M}} \vdash y\Diamond y_0$. Since $x_0\Diamond y_0 \in \Gamma_{\mathcal{M}}$, by definition of $\Gamma_{\mathcal{M}}$, the result follows immediately. For the second implication, we distinguish three cases:

   - if $x, y \in \bigcup \mathcal{M}$ then there exists $X, Y \in \mathcal{M}$, $X \neq Y$ such that $x \in X$ and $y \in Y$. By the first point, there exists $x_0 \in Elem_{\mathcal{M}}(X)$, $y_0 \in Elem_{\mathcal{M}}(Y)$ such that $\Gamma_{\mathcal{M}} \vdash x\Diamond x_0$ and $\Gamma_{\mathcal{M}} \vdash y\Diamond y_0$. Moreover, by definition of $\Gamma_{\mathcal{M}}$, we have $\neg(x_0\Diamond y_0) \in \Gamma_{\mathcal{M}}$. We obtain the result as follows:

$$\dfrac{\dfrac{\dfrac{x\Diamond x_0}{x_0\Diamond x} \quad \dfrac{[x\Diamond y] \quad y\Diamond y_0}{x\Diamond y_0}}{x_0\Diamond y_0} \quad \neg(x_0\Diamond y_0)}{\dfrac{\perp}{\neg(x\Diamond y)}}$$

- if $x \in \bigcup\mathcal{M}$ and $y \notin \bigcup\mathcal{M}$ then, by the first implication we have $\Gamma_{\mathcal{M}} \vdash x\Diamond x$ and, by the second point of this lemma we have $\Gamma_{\mathcal{M}} \vdash \neg(y\Diamond y)$. It is easy now to infer that $\Gamma_{\mathcal{M}} \vdash \neg(x\Diamond y)$.
- if $x, y \notin \bigcup\mathcal{M}$, by the second point of this lemma we have $\Gamma_{\mathcal{M}} \vdash \neg(x\Diamond x)$ and $\Gamma_{\mathcal{M}} \vdash \neg(y\Diamond y)$. Then $\Gamma_{\mathcal{M}} \vdash \neg(x\Diamond y)$ follows easily.

□

Notice that, from the third point of Lemma 7, and since $\Gamma_{\mathcal{M}}$ is satisfiable, hence consistent, we obtain that $[\![\varphi]\!]_{\mathcal{M}} = 1$ if and only if $\Gamma_{\mathcal{M}} \vdash \varphi$. Thus, the set of formulas that are satisfied by $\mathcal{M}$ is finitely axiomatisable since $Th(\Gamma_{\mathcal{M}}) = \{\varphi \mid [\![\varphi]\!]_{\mathcal{M}} = 1\}$, and $\Gamma_{\mathcal{M}}$ is finite by definition.

Theorem 4 *Let $\mathcal{M}$ be a heap and $\phi_{\mathcal{M}}$ be the formula $\bigwedge\Gamma_{\mathcal{M}}$. If $[\![\phi_{\mathcal{M}}]\!]_{\mathcal{M}'} = 1$, then $\mathcal{M} = \mathcal{M}'$.*

Proof: Using that $\{\varphi \mid [\![\varphi]\!]_{\mathcal{M}} = 1\} = \{\varphi \mid [\![\varphi]\!]_{\mathcal{M}} = 1\} = Th(\Gamma_{\mathcal{M}})$ we obtain $[\![x\Diamond y]\!]_{\mathcal{M}} = [\![x\Diamond y]\!]_{\mathcal{M}'}$ for any $x, y \in \Sigma^+$. Now choose $x \in \bigcup\mathcal{M}$. Then $[\![x\Diamond x]\!]_{\mathcal{M}} = [\![x\Diamond x]\!]_{\mathcal{M}'} = 1$, hence $x \in \bigcup\mathcal{M}'$. Dually, if $x \notin \bigcup\mathcal{M}$, then $[\![x\Diamond x]\!]_{\mathcal{M}} = [\![x\Diamond x]\!]_{\mathcal{M}'} = 0$, therefore $x \notin \bigcup\mathcal{M}'$. So $\bigcup\mathcal{M} = \bigcup\mathcal{M}'$. Next, we show that the partitioning of $\bigcup\mathcal{M}$ is the same for both $\mathcal{M}$ and $\mathcal{M}'$. For all $x, y \in \Sigma^+$, we have $x, y \in X$ for some $X \in \mathcal{M}$ if and only if $[\![x\Diamond y]\!]_{\mathcal{M}} = [\![x\Diamond y]\!]_{\mathcal{M}'} = 1$, if and only if $x, y \in Y$ for some $Y \in \mathcal{M}'$. This concludes our proof. □

Example Given $\Sigma = \{a, b, c\}$, the heap $\mathcal{M} = \{ab^*\}$ composed of one element pointed to by a with a b self loop is characterized by the formula $a\Diamond ab \wedge \neg c\Diamond c \wedge \neg ac\Diamond ac$.

Provided that the satisfiability problem for **pAL** is decidable, we obtain a (possibly inefficient) method for model checking, as a byproduct of the above theorem. To answer the question whether a heap $\mathcal{M}$ is a model of a formula ψ, it is sufficient to construct the characteristic formula $\phi_{\mathcal{M}}$ and decide whether $\phi_{\mathcal{M}} \to \psi$ is satisfiable.

4 Tableau Decision Procedure for pAL

A proof that uses natural deduction is mainly based on manually adding assumptions in order to reach contradictions (and deleting them afterwards). This makes, in general, natural deduction unsuitable for automated reasoning and motivates our preference for the method of *analytic tableaux* [24], an elegant and efficient proof procedure for propositional logic, which we subsequently extend to **pAL**. Traditionally, a tableau for a propositional formula φ is a tree having φ as the root node and subformulas of φ or negations of subformulas of φ as nodes. A tableau branch is said to be *closed* if it contains a formula together with its negation, and *open* otherwise. A tableau is said to be closed if and only if all its branches are closed. To check whether a formula φ is a tautology one builds the tableau for $\neg\varphi$, and infers that φ is a tautology if and only if the tableau eventually closes. In case at least one branch remains open, a counterexample for φ can be extracted.

$$\frac{\neg(xt\Diamond z)\ldots yt\Diamond z}{\neg(x\Diamond y)}\ (T_1) \qquad \frac{xt\Diamond y}{x\Diamond x}\ (T_2) \qquad \frac{x\Diamond y}{y\Diamond x}\ (T_3) \qquad \frac{\neg(x\Diamond y)}{\neg(y\Diamond x)}\ (T_4)$$

$$\frac{\varphi_1 \wedge \varphi_2}{\varphi_1;\varphi_2}\ (T_5) \qquad \frac{\neg(\varphi_1 \wedge \varphi_2)}{\neg\varphi_1 \quad | \quad \neg\varphi_2}\ (T_6) \qquad \frac{\neg\neg\varphi}{\varphi}\ (T_7) \qquad \frac{\varphi\ldots\neg\varphi}{\bot}\ (T_8)$$

Figure 4: Tableau Expansion Rules

Figure 4 shows the tableau expansion rules for **pAL**. We consider that $x, y, z \in \Sigma^+$ and $t \in \Sigma^*$ that is, we can apply the rules also for an empty suffix ($t = \epsilon$). The tableau is constructed top-down. A rule whose hypothesis are of the form $\varphi \ldots \psi$ (namely T_1 and T_8) can be applied at a node, as soon as both φ and ψ are on the path from the root to the current node, order independent. Rule (T_5) expands by putting both φ_1 and φ_2 on the same branch of the tableau, while rule (T_6) creates two new branches, one containing φ_1 and the other one containing φ_2. All other rules expand by appending their conclusion to the current branch. We use rule (T_8) to close a branch, since $\bot$ does not expand any further. Each rule can only be applied provided that its conclusion does not already appear on the current branch, otherwise the procedure runs the risk of looping forever (for instance, applying one of rules $T_{3,4}$), without introducing any new formulas[5].

Example Figure 5 presents a sample run of the tableau procedure whose goal is to prove that, for some *given* $k \in \mathbb{N}$, $\phi_k \triangleq a\Diamond ab \to a\Diamond ab^k$ is a tautology. First, we eliminate the implication: $\phi_k = \neg(a\Diamond ab \wedge \neg(a\Diamond ab^k))$ and start the tableau procedure with $\neg\phi_k$ as the root node. To the right of each node occurs the number of the node(s) used in the hypothesis, followed by the name of the rule applied in order to obtain that node. In this example, the tableau closes after $k + 6$ steps. Branching lacks in this tableau because the rule (T_6) is never applied. $\Box$

The tableau expansion rules can be easily understood with the natural deduction rules in mind. For instance, rule (T_1) can be derived using (sufI), ($\bot$I) and ($\neg$I). Rules (T_2) and (T_3) are (sufE) and (sufI), respectively, while (T_4) is easily derived using (sym) and ($\neg I$). The rest of the rules correspond to the purely propositional part of the natural deduction system and are an easy check. This, and Corollary 1 ensure that the tableau rules are sound i.e., if a tableau started with $\neg\varphi$ closes, then φ is a tautology. The dual implication, if φ is a tautology then every tableau started with $\neg\varphi$ will eventually close, will be dealt with in the following.

Note that the rules in Figure 4 do not cover the entire **pAL** syntax from Figure 2: the atomic propositions of the form $x \leq y$ are not considered. The reason is that such propositions trivially evaluate to either true or false and could be eliminated from a formula a priori. However, to be

[5]The definition of a finer notion of redundancy is planned in the full version.

[1]	$\neg\neg(a\Diamond ab \wedge \neg(a\Diamond ab^k))$			[7]	$\neg(ab^{k-1}\Diamond a)$	$(5, 6, T_1)$
[2]	$a\Diamond ab \wedge \neg(a\Diamond ab^k))$	$(1, T_7)$			$\vdots$	
[3]	$a\Diamond ab$	$(2, T_5)$				
[4]	$\neg(a\Diamond ab^k))$	$(2, T_5)$				
[5]	$ab\Diamond a$	$(3, T_3)$		[k+5]	$\neg(ab\Diamond a)$	$(5, k+4, T_1)$
[6]	$\neg(ab^k\Diamond a)$	$(4, T_4)$		[k+6]	$\perp$	$(5, k+5, T_8)$

Figure 5: Tableau Example

pedantic we could introduce the following rules:

$$\frac{[x \le xt]}{} \ (T_9) \qquad \frac{x \le y}{\perp} \text{ when } x \not\le y \ (T_{10})$$

$$\frac{\neg(x \le xt)}{\perp} \ (T_{11}) \qquad \frac{[\neg(x \le y)]}{} \text{ when } x \not\le y \ (T_{12})$$

Rule (T_9) simply deletes its hypothesis from the current branch, while rule (T_{10}) closes the branch, when its hypothesis is false. Rules $(T_{11,12})$ are needed to deal with the dual situations.

The rest of this section is concerned with proving that the tableau method is both complete and effective. To handle the tableau rules in an uniform way, we use the unified notation of [24]: let an α-rule be one of the rules $(T_{1\ldots5})$ and β-rule be the rule (T_6). We denote the premises of a R-rule by $R_1 \ldots R_n$ and its conclusions by $\widetilde{R}_1, \ldots \widetilde{R}_m$, where $R = \alpha, \beta$.

Definition 3 *A set of formulas* Γ *is said to be* downward closed *if and only if it respects the following conditions:*

- *for no* $x, y \in \Sigma^+$*, we have* $x\Diamond y, \neg(x\Diamond y) \in \Gamma$*,*
- *for any* α*-rule, if* $\alpha_1, \ldots, \alpha_n \in \Gamma$*, then* $\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_m \in \Gamma$*,*
- *for any* β*-rule, if* $\beta_1, \ldots, \beta_n \in \Gamma$*, then either* $\widetilde{\beta}_1 \in S$ *or ... or* $\widetilde{\beta}_m \in \Gamma$.

A tableau branch is said to be *complete* if no more rules can be applied to expand it. A tableau is said to be complete if and only if each of its branches is complete. It is manifest that an open complete tableau branch is a downward closed set. The following technical lemma is key to showing satisfiability of downward closed sets. We recall here the definition of the $\approx_\Gamma$ relation from Lemma 4. The following theorem is the main result of this section.

Lemma 8 *For any downward closed set of formulas* Γ*,* $\neg(x\Diamond y) \in \Gamma$ *implies* $x \not\approx_\Gamma y$*.*

Proof: Assume $x \approx_\Gamma y$. By induction on the normalized proof for $x \approx_\Gamma y$ we show $\neg(x\Diamond y) \notin \Gamma$. The existence of such a proof is guaranteed by Lemma 3. For the (sufE) part of the proof we show $x \approx_\Gamma y \Rightarrow x\Diamond y \in \Gamma$. Since Γ is downward closed, we have $\neg(x\Diamond y) \notin \Gamma$.

- if $x \approx_\Gamma y$ because $x\Diamond y \in \Gamma$ we are done.
- if $x \approx_\Gamma y$ because $y \approx_\Gamma x$, we have $y\Diamond x \in \Gamma$ by induction hypothesis, then $x\Diamond y \in \Gamma$, since Γ is downward closed.
- if $x \approx_\Gamma x$ because $xy \approx_\Gamma z$, we have $xy\Diamond z \in \Gamma$ by induction hypothesis, then $x\Diamond x \in \Gamma$ since Γ is downward closed.

Next, for the (sufI) part we show directly $x \approx_\Gamma y \Rightarrow \neg(x\Diamond y) \notin \Gamma$.

- for the premises of the (sufI) part, we have $x\Diamond y \in \Gamma$ from the previous point, hence $\neg(x\Diamond y) \notin \Gamma$, since Γ is downward closed.
- if $x \approx_\Gamma y$ because $y \approx_\Gamma x$ and $\neg(y\Diamond x) \notin \Gamma$ by the induction hypothesis, we have $\neg(x\Diamond y) \notin \Gamma$, or else $\neg(y\Diamond x) \in \Gamma$ would be true, by the fact that Γ is downward closed.
- if $x \approx_\Gamma y$ because $x = x'x''$, $x' \approx_\Gamma y'$ and $y'x'' \approx_\Gamma y$, we have $\neg(x'\Diamond y')$, $\neg(y'x''\Diamond y) \notin \Gamma$, by induction hypothesis. Suppose $\neg(x\Diamond y) \in \Gamma$ was true. If $y'x''\Diamond y \in \Gamma$ then $\neg(x'\Diamond y') \in \Gamma$, since Γ is downward closed. But this contradicts the induction hypothesis, hence $y'x''\Diamond y \notin \Gamma$. Analogously, $x'\Diamond y' \notin \Gamma$. Moreover, $y\Diamond y'x'', y'\Diamond x' \notin \Gamma$. Repeating the reasoning, we obtain that for no nodes $u \approx_\Gamma v$ in the (sufI) part of the proof we have $u\Diamond v \in \Gamma$. However, this contradicts with the previous point, since at least for the premises $u \approx_\Gamma v$ of the (sufI) part we have $u\Diamond v \in \Gamma$. Thus, the assumption $\neg(x\Diamond y) \in \Gamma$ cannot be true.

$\square$

Theorem 5 *Any downward closed set of formulas containing a finite number of alias propositions is satisfiable.*

Proof: Let Γ be such a set, and let $H_{\Gamma/\approx_\Gamma}$ be a heap constructed as in Lemma 4. We need to prove that $[\![\varphi]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$ for all $\varphi \in \Gamma$. The proof proceeds by induction on the size of φ:

- if $\varphi = x\Diamond y$ then $x \approx_\Gamma y$ by definition of $\approx_\Gamma$, hence $x, y \in X$ for some $X \in H_{\Gamma/\approx_\Gamma}$. Therefore $[\![x\Diamond y]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$.
- if $\varphi = \varphi_1 \wedge \varphi_2$, and since, by induction hypothesis, $[\![\varphi_1]\!]_{H_{\Gamma/\approx_\Gamma}} = [\![\varphi_2]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$ we have $[\![\varphi]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$.
- if $\varphi = \neg\varphi'$ we distinguish three cases:
  - if $\varphi' = x\Diamond y$ then, by Lemma 8, we have $x \not\approx_\Gamma y$. Hence $[\![x\Diamond y]\!]_{H_{\Gamma/\approx_\Gamma}} = 0$ which leads to the result.
  - if $\varphi' = \varphi_1 \wedge \varphi_2$ then, since Γ is downward closed, we have either $\neg\varphi_1 \in \Gamma$ or $\neg\varphi_2 \in \Gamma$. By the induction hypothesis, since the size of φ is greater than both the sizes of $\neg\varphi_1$ and $\neg\varphi_2$, we have either $[\![\neg\varphi_1]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$ or $[\![\neg\varphi_2]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$. Both cases lead to $[\![\neg(\varphi_1 \wedge \varphi_2)]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$.

– if $\varphi' = \neg\varphi_1$ then, since Γ is downward closed, we have $\varphi_1 \in \Gamma$. By induction hypothesis we have $[\![\varphi_1]\!]_{H_{\Gamma/\approx_\Gamma}} = 1$, which leads to the desired conclusion.

□

The proof of the above theorem uses the model construction technique from Lemma 4. The same method can be moreover used to derive a counterexample of a non-valid formula, starting from an open tableau branch. Before stating our completeness result for the tableau method, let us show that the method is effective. That is, each tableau procedure started with a finite formula as the root node, using the rules from Figure 4, eventually terminates.

Lemma 9 *The tableau of a finite formula is finite.*

Proof: It is sufficient to show that each branch of a tableau with a finite formula as the root node is finite. Finiteness of the entire tableau follows by an application of König's Lemma, since the branching degree in our case is 2. For the purposes of this proof we define the degree of a formula to be:

$$\begin{array}{rcl} \|x\Diamond y\| & \triangleq & \max(|x|,|y|) \\ \|\varphi_1 \wedge \varphi_2\| & \triangleq & \max(\|\varphi_1\|,\|\varphi_2\|)+1 \\ \|\neg\varphi\| & \triangleq & \begin{cases} \|\varphi\| & \text{if } \varphi = x\Diamond y \\ \|\varphi\|+1 & \text{otherwise} \end{cases} \end{array}$$

where $|x|$ denotes the length of $x \in \Sigma^+$. It is easy to see that the number of formulas having the same degree is finite. A tableau branch is an ordered sequence of formulas $\varphi_1, \varphi_2, \ldots$ where each φ_i is obtained from one or more φ_j, $j < i$ by an application of one of the rules $(T_{1\ldots7})$. We show next that $\|\varphi_i\| \leq \max_{j\leq i}(\|\varphi_j\|)$. It is sufficient to show, for each rule R with premises $R_1, \ldots, R_n$ and conclusions $\widetilde{R}_1, \ldots, \widetilde{R}_m$, that $\max_{1\leq i\leq n}(\|R_i\|) \geq \|\widetilde{R}_j\|$, for all $1 \leq j \leq m$. The most interesting case is rule T_1:

$$\begin{array}{rcl} \max(\|\neg(xt\Diamond z)\|,\|yt\Diamond z\|) & = & \max(|xt|,|yt|,|z|) \\ & \geq & \max(|x|,|y|,|z|) \\ & \geq & \max(|x|,|y|) \\ & = & \|\neg(x\Diamond y)\| \end{array}$$

This guarantees that, on a branch starting with φ such that $\|\varphi\| = k$, each formula can be of degree at most k. Assume now that the branch is infinite. Since the number of formulas with the same degree is finite, and we have at most k degrees on the branch, the number of formulas must be finite. Hence, at least one formula occurs twice, which contradicts the previously stated proviso, that each rule is applied only if the conclusion does not already appears on the branch. This concludes our proof. □

Besides showing termination of the tableau procedure, the above lemma, together with Theorem 5 ensure that the tableau approach is complete.

Corollary 2 *If a formula φ is a tautology then every complete tableau starting with $\neg\varphi$ eventually closes.*

Proof: Suppose that there exists a complete tableau starting with $\neg\varphi$ with a branch left open. Such a branch is a downward closed set and, since it is finite, by Lemma 9, we obtain that it is also satisfiable, by Theorem 5. Hence φ cannot be a tautology. $\square$ As a side remark, notice that our logic is not *compact* i.e., for a denumerable infinite set of formulas Γ, if each finite subset $\Delta \subset \Gamma$ is satisfiable, this however does not imply the satisfiability of Γ. Consider for instance the set $\Gamma = \{a^k \Diamond a^k, \neg(a \Diamond a^{k+1}) \ldots \neg(a^k \Diamond a^{k+1}) \mid k \geq 1\}$ defining an infinite non-cyclic list with selector a. Each finite subset of Γ is satisfiable, yet Γ is not, since we do not interpret formulas over infinite heaps. Further study of this issue is planned in order to extend **pAL** towards describing infinite classes of recursive data structures. In the light of the decidability result concerning **pAL**, we are next investigating the time complexity of the above satisfiability problem, and find that it is NP-complete.

Lemma 10 *The satisfiability problem for* **pAL** *is in NP.*

Proof: Given a **pAL** formula ϕ, we need to show that:

- If ϕ is satisfiable, then it admits a model $\mathcal{M}$ such that $\|\mathcal{M}\| \leq P(|\phi|)$, where P is a unary polynomial integer function. Let $\phi = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{m_i} l_{ij}$ be the disjunctive normal form of ϕ, where each l_{ij}, $1 \leq i \leq n, 1 \leq j \leq n$ is either an alias formula or the negation of an alias formula occurring in ϕ. Thus $m_i \leq |\phi|$, for all $1 \leq i \leq n$. If ϕ is satisfiable, then at least one conjunctive clause, say the k-th, $1 \leq k \leq n$ must be satisfiable. Let Γ be the set $\{l_{kj} \mid 1 \leq j \leq m_k\}$. If Γ is satisfiable, then, by Theorem 3 it must be consistent. Since it is consistent then, by the proof of Theorem 3, $H_{\Gamma/\approx_\Gamma}$ defined in Lemma 4, is a model. However, according to Lemma 4, we have $\|H_{\Gamma/\approx_\Gamma}\| \leq k \cdot \|\Gamma\| = k \cdot m_k \leq k \cdot |\phi|$.

- Given a heap $\mathcal{M}$, checking whether $[\![\phi]\!]_{\mathcal{M}} = 1$ takes time at most $Q(\|\mathcal{M}\|, |\phi|)$, where Q is a binary polynomial integer function. Each alias proposition occurring within ϕ can be evaluated in time at most $k \cdot \|\mathcal{M}\|$, where k is the constant from Lemma 4. Moreover, the number of alias propositions is bounded by $|\phi|$. Thus the time required to check whether $[\![\phi]\!]_{\mathcal{M}} = 1$ is at most $k \cdot \|\mathcal{M}\| \cdot |\phi| + E(|\phi|)$, where $E(|\phi|)$ is a polynomial bound on the time needed to evaluate ϕ, provided with a valuation the atomic propositions occurring within it.

$\square$

Theorem 6 *The satisfiability problem for* **pAL** *is NP-complete.*

Proof: To show NP-hardness, let $\varphi = \bigwedge_{i=1}^{n}(l_{i1} \vee l_{i2} \vee l_{i3})$, where each $l_{ij} \in \{x, \neg x\}$ for some binary variable x, be an instance of 3-SAT. We define the following translation function:

$$\Theta(\varphi) \triangleq t \Diamond t \wedge f \Diamond f \wedge \neg(t \Diamond f) \wedge \bigwedge_{i=1}^{n} \Theta(l_{i1}) \vee \Theta(l_{i2}) \vee \Theta(l_{i3})$$

$$\Theta(l) \triangleq \begin{cases} x \Diamond t & \text{if } l = x \\ x \Diamond f & \text{if } l = \neg x \end{cases}$$

It is obvious that computing Θ takes time bounded by a polynomial in the size of ϕ. It remains to be shown that φ is satisfiable if and only if $\Theta(\varphi)$ is. For the left to right direction, assume that $[\![\varphi]\!]_v = 1$ for some boolean valuation $v : FV(\varphi) \rightarrow \mathbb{B}$ of the free variables of φ. Let $\mathcal{M}_v$ be the heap $\{T, F\}$ such that $t \in T$, $f \in F$ and for each $x \in FV(\varphi)$, we have $x \in T$ if $v(x) = 1$, and $x \in F$ otherwise. It is easily checked that $\mathcal{M}$ is indeed a heap according to Definition 1, and that $[\![\Theta(\varphi)]\!]_{\mathcal{M}_v} = 1$. Viceversa, for each heap $\mathcal{M}$ such that $[\![\Theta(\varphi)]\!]_{\mathcal{M}_v} = 1$ we define a valuation $v_{\mathcal{M}} : FV(\phi) \rightarrow \mathbb{B}$ by $v(x) = 1$ if $[\![x \Diamond t]\!]_{\mathcal{M}} = 1$ and $v(x) = 0$ otherwise. It is easily checked that v is a well-defined function and that indeed $[\![\phi]\!]_v = 1$. The NP-completeness result follows, from the above considerations, together with Lemma 10. $\Box$

5 An Effective Program Logic

In this section we demonstrate the possibility of using **pAL** as a weakest precondition calculus for imperative programs with destructive updating. Otherwise stated, we show that **pAL** is closed under applications of the weakest preconditions predicate transformers. Our proof of closure is constructive i.e., we define weakest preconditions in terms as predicate transformers directly on **pAL**. This is achieved by means of the sound and complete program logic defined on top of **wAL** [2]. Moreover, soundness and completness of the **pAL** weakest precondition axioms are consequences of soundness and completness in the case of **wAL**.

We consider a simple imperative language consisting of the following three atomics statements. Note that the statements of most object-oriented languages can be precompiled in this form, possibly by introducing fresh temporary variables:

$$Stmnt \quad := \quad uv = \text{null} \mid uv = \text{new} \mid uv = w \;\; (\text{where } uv \not\leq w)$$

Here $v, w \in \Sigma$ denote pointer variables, and $u \in \Sigma^*$ is a (possibly empty) dereferencing path. The first statement resets the v field of the object pointed to by u, if $u \neq \epsilon$, or the v top-level variable, otherwise. This may cause the builtin garbage collector recall all non-reachable objects. The second statement allocates a fresh object for further uses, and the third statement assigns its left-hand side the object pointed to by the right-hand side variable. The syntactic constraint that comes with the last statement is due to the following technical problem. The semantics of the assignment is given as the composition of two primitive operations: first one removes the v arc from the node pointed to by u, and then it assigns it to w. If $uv \leq w$ and there are no other paths to the cell pointed to by w, the garbage collection caused by the first operation removes the unreachable cell before the assignment is finished. The requirement $uv \not\leq w$ is however sufficient to ensure that, in practice, this situation never occurs.

The axiomatic semantics of this language has been introduced in [2], by defining a weakest precondition operator $\widetilde{pre}$ on **wAL** formulas, and is briefly recalled here. For any transition relation over a sequence of statements $\omega \in Stmnt^+$, $\widetilde{pre}$ distributes over conjunction and universal quantification i.e., $\widetilde{pre}(\omega, \varphi_1 \wedge \varphi_2) = \widetilde{pre}(\omega, \varphi_1) \wedge \widetilde{pre}(\omega, \varphi_2)$ and $\widetilde{pre}(\omega, \forall X \,.\, \varphi) = \forall X \,.\, \widetilde{pre}(\omega, \varphi)$. For total transition relations we have $\widetilde{pre}(\omega, \varphi) \Rightarrow \neg\widetilde{pre}(\omega, \neg\varphi)$. If, moreover, the transition relation is total and deterministic, we have that $\widetilde{pre}$ is its own dual i.e., $\widetilde{pre}(\omega, \varphi) \Leftrightarrow \neg\widetilde{pre}(\omega, \neg\varphi)$. In the latter case, $\widetilde{pre}$ distributes over disjunction and existential quantification too. These properties of $\widetilde{pre}$ for total, deterministic programs allow us to define general inference rules for the precondition inductively on the structure of the postcondition. In particular, it is sufficient to define preconditions only for modalities, the rest of the atomic propositions in **wAL** being pure i.e., having model-independent denotations. Figure 6 (upper part) gives the precondition of primitive storeless operations add, rem and new for arbitrary modalities. This is generalized to the statements defined in the previous (lower part).

$$\{\exists X . X \setminus Sv\Sigma^* = T \wedge \langle X\rangle(\sigma \setminus Sv\Sigma^*)\}\ \textbf{rem(S,v)}\ \{\langle T\rangle\sigma\}$$
$$\{\exists X . \chi^v(S,T,X) = U \wedge \bigvee_{i=1,2} \psi_i^{\sigma,\sigma}(X,X)\}\ \textbf{add(S,v,T)}\ \{\langle U\rangle\sigma\}$$

$$\begin{aligned} \text{where } \chi^v(S,T,X) &\triangleq X \cup Sv((T^{-1}S)v)^*(T^{-1}X) \\ \psi_1^{x,y}(X,Y) &\triangleq Sv((T^{-1}S)v)^*(T^{-1}X) \cap x = \emptyset \wedge \langle Y\rangle y \\ \psi_2^{x,y}(X,Y) &\triangleq Sv((T^{-1}S)v)^*(T^{-1}X) \cap x \neq \emptyset \wedge \langle Y\rangle \Sigma^* \end{aligned} \quad (4)$$

$$\{(T = Sv \wedge \sigma \cap Sv \neq \emptyset) \vee \langle T\rangle\sigma\}\ \textbf{new(S,v)}\ \{\langle T\rangle\sigma\}$$

$$\{\exists S . \langle S\rangle u \wedge \widetilde{pre}(rem(S,v),\varphi)\}\ \textbf{uv = null}\ \{\varphi\}$$
$$\{\exists S . \langle S\rangle u \wedge \widetilde{pre}(rem(S,v),\widetilde{pre}(new(S,v),\varphi))\}\ \textbf{uv = new}\ \{\varphi\}$$
$$\{\exists S \exists T . \langle S\rangle u \wedge \langle T\rangle w \wedge \widetilde{pre}(rem(S,v),\widetilde{pre}(add(S,v,T),\varphi))\}\ \textbf{uv = w}\ \{\varphi\}$$

Figure 6: **wAL** Weakest Preconditions

For the rest of this section, let $\sigma, \tau, \theta, u, v, w$ denote constant words, and x, y, z denote variables ranging over words. We introduce the following notation: $\exists x \leq \sigma \,.\, \varphi(x) \triangleq \bigvee_{\tau \in Pref(\sigma)} \varphi(\tau)$. Since σ is a finite word, so is the formula on the right. To start, we recall the equivalence (1), which allows the proper embedding of **pAL** into **wAL**. Figure 7 introduces a number of syntactic shorthands, providing context-dependent translations from **wAL** to **pAL** for them. That is, we do not translate the shorthands individually, but rather in an existentially closed context.

We assert that all translations defined in Figure 7 preserve logical equivalence. To convince ourselves of this fact, let us perform the step-by-step derivation for the positive form of α_σ. The rest of the formulas are translated along the same lines.

$$\exists S \,.\, \langle S\rangle u \wedge \alpha_\sigma \equiv \exists S \,.\, \langle S\rangle u \wedge \sigma \in Sv\Sigma^* \iff$$
$$\exists S \,.\, \exists x \leq \sigma \,.\, \langle S\rangle u \wedge \langle S\rangle x \wedge xv \leq \sigma \iff \exists x \leq \sigma \,.\, x\Diamond u \wedge xv \leq \sigma$$

The goal of this section is to prove that the logic **pAL** is expressive enough to characterize the destructive updating program statements considered in the previous. The following theorem captures the result.

Theorem 7 *For any sequence of statements $\omega \in Stmnt^*$ and any formula $\varphi \in$ **pAL**$[\Sigma]$, we have $\widetilde{pre}(\omega,\varphi) \in$ **pAL**$[\Sigma]$.*

Proof: We give the proof for $\varphi = \sigma\Diamond\tau$ and $\omega = uv =$ null $|$ $uv =$ new $|$ $\omega = uv = w$. The statement of the theorem is a generalization using the distributivity properties for $\widetilde{pre}$ and induction on the length of ω.

Definition	**wAL**	**pAL**
$\alpha_\sigma \;:\; \sigma \in Sv\Sigma^*$	$\exists S \,.\, \langle S \rangle u \wedge \alpha_\sigma$ $\exists S \,.\, \langle S \rangle u \wedge \neg\alpha_\sigma$	$\exists x \leq \sigma \,.\, x \diamond u \wedge xv \leq \sigma$ $u \diamond u \wedge \neg(\exists x \leq \sigma \,.\, x \diamond u \wedge xv \leq \sigma)$
$\beta_\sigma \;:\; \sigma \in Sv(T^{-1}X)$	$\exists S \exists T \exists X \,.\, \langle S \rangle u \wedge \langle T \rangle w \wedge$ $\langle X \rangle \theta \wedge \beta_\sigma$ $\exists S \exists T \exists X \,.\, \langle S \rangle u \wedge \langle T \rangle w \wedge$ $\langle X \rangle \theta \wedge \neg\beta_\sigma$	$\exists x \leq \sigma \,.\, x \diamond u \wedge xv \leq \sigma \wedge$ $w((xv)^{-1}\sigma) \diamond \theta$ $u \diamond u \wedge w \diamond w \wedge \theta \diamond \theta \wedge$ $\neg(\exists x \leq \sigma \,.\, x \diamond u \wedge xv \leq \sigma \wedge$ $w((xv)^{-1}\sigma) \diamond \theta)$
$\gamma_\sigma \;:\; \sigma \in Sv(T^{-1}S)v\Sigma^*$	$\exists S \exists T \,.\, \langle S \rangle u \wedge \langle T \rangle w \wedge \gamma_\sigma$ $\exists S \exists T \,.\, \langle S \rangle u \wedge \langle T \rangle w \wedge \neg\gamma_\sigma$	$\exists x \leq \sigma \; \exists y \leq (xv)^{-1}\sigma \,.\, x \diamond u \wedge$ $xv \leq \sigma \wedge wy \diamond u \wedge yv \leq (xv)^{-1}\sigma$ $u \diamond u \wedge w \diamond w \wedge$ $\neg(\exists x \leq \sigma \; \exists y \leq (xv)^{-1}\sigma \,.\, x \diamond u \wedge$ $xv \leq \sigma \wedge wy \diamond u \wedge yv \leq (xv)^{-1}\sigma)$

Figure 7: **wAL** to **pAL** translation shorthands

- **[uv = null]** Let Φ be the formula $\exists S \,.\, \langle S \rangle u \wedge \widetilde{pre}(rem(S, v), \sigma \diamond \tau)$ given in Figure 6. Since $\sigma \diamond \tau \iff \exists X \,.\, \langle X \rangle \sigma \wedge \langle X \rangle \tau$, we have:

$$\begin{aligned} \Phi &= \exists S \exists X \,.\langle S \rangle u \wedge \quad \widetilde{pre}(rem(S, v), \langle X \rangle \sigma) \wedge \widetilde{pre}(rem(S, v), \langle X \rangle \tau) \\ &= \exists S \exists X \,.\, \langle S \rangle u \wedge \quad (\exists Y' \,.\, Y' \setminus Sv\Sigma^* = X \wedge \langle Y' \rangle(\sigma \setminus Sv\Sigma^*)) \wedge \\ &\qquad\qquad\qquad\qquad (\exists Y'' \,.\, Y'' \setminus Sv\Sigma^* = X \wedge \langle Y'' \rangle(\tau \setminus Sv\Sigma^*)) \end{aligned}$$

  If $\langle X \rangle \sigma$ is true, then $X \neq \emptyset$, thus $Y' \setminus Sv\Sigma^* \cap Y'' \setminus Sv\Sigma^* = X \neq \emptyset$, which leads to $Y' \cap Y'' \neq \emptyset$. Now since Y' and Y'' denote heap entities, by Definition 1 (C2) we have $Y' = Y''$. Thus we can rewrite:

$$\begin{aligned} \Phi &= \exists S \exists X \exists Y \,.\, \langle S \rangle u \wedge Y \setminus Sv\Sigma^* = X \wedge \langle Y \rangle(\sigma \setminus Sv\Sigma^*) \wedge \langle Y \rangle(\tau \setminus Sv\Sigma^*) \\ &= \exists S \exists X \,.\, \langle S \rangle u \wedge \langle X \rangle(\sigma \setminus Sv\Sigma^*) \wedge \langle X \rangle(\tau \setminus Sv\Sigma^*) \\ &= \exists S \exists X \,.\, \langle S \rangle u \wedge \langle X \rangle \sigma \wedge \langle X \rangle \tau \wedge \neg\alpha_\sigma \wedge \wedge \neg\alpha_\tau \end{aligned}$$

  The last expression can be now easily translated into **pAL** using the scheme from Figure 7.

- **[uv = new]** Let Φ be the formula $\exists S \,.\, \langle S \rangle u \wedge \widetilde{pre}(rem(S, v), \widetilde{pre}(new(S, v), \sigma \wedge \tau))$ given in Figure 6. Since the weakest precondition transformer distributes with all the operators of first-order logic, we obtain:

$$\begin{aligned} \Phi &= \exists S \exists X \,.\, \langle S \rangle u \wedge \quad \widetilde{pre}(rem(S, v), \widetilde{pre}(new(S, v), \langle X \rangle \sigma)) \wedge \\ &\qquad\qquad\qquad\qquad \widetilde{pre}(rem(S, v), \widetilde{pre}(new(S, v), \langle X \rangle \tau)) \end{aligned}$$

Next, we have:

$$\begin{array}{l}
\widetilde{pre}(rem(S,v), \widetilde{pre}(new(S,v), \langle X\rangle\sigma)) = \\
\widetilde{pre}(rem(S,v), (X = Sv \wedge \sigma \cap Sv \neq \emptyset) \vee \langle X\rangle\sigma) = \\
\exists Y \,.\, Y \setminus Sv\Sigma^* = X \wedge ((X = Sv \wedge \sigma \cap Sv \neq \emptyset) \vee \langle Y\rangle(\sigma \setminus Sv\Sigma^*)) = \\
\exists Y \,.\, Y \setminus Sv\Sigma^* = X \wedge ((Y \setminus Sv\Sigma^* = Sv \wedge \sigma \cap Sv \neq \emptyset) \vee \langle Y\rangle(\sigma \setminus Sv\Sigma^*)) = \\
\exists Y \,.\, Y \setminus Sv\Sigma^* = X \wedge \langle Y\rangle(\sigma \setminus Sv\Sigma^*)
\end{array}$$

Replacing the result into Φ, we obtain the formula from the previous point. Note that, allocating a new cell to a pointer has the same effect on an alias proposition as setting the pointer to null.

- **[uv = w]** Let $\Phi \triangleq \exists S \exists T \,.\, \langle S\rangle u \wedge \langle T\rangle w \wedge \widetilde{pre}(rem(S,v), \widetilde{pre}(add(S,v,T), \sigma\Diamond\tau))$, as in Figure 6. Since the weakest precondition transformer distributes with all the operators of first-order logic, we obtain:

$$\begin{array}{rll}
\Phi = & \exists S & \exists T \,.\, \langle S\rangle u \wedge \langle T\rangle w \wedge \\
& \exists X & .\, \widetilde{pre}(rem(S,v), \widetilde{pre}(add(S,v,T), \langle X\rangle\sigma)) \\
& & \widetilde{pre}(rem(S,v), \widetilde{pre}(add(S,v,T), \langle X\rangle\tau))
\end{array}$$

Applying the weakest precondition for $add(S,v,T)$ we further obtain:

$$\begin{array}{rll}
\Phi = & \exists S & \exists T \,.\, \langle S\rangle u \wedge \langle T\rangle w \wedge \\
& \exists X & .\, \widetilde{pre}(rem(S,v), \exists Y' \,.\, \chi^v(S,T,Y') = X \wedge \bigvee\limits_{i=1,2} \psi_i^{\sigma,\sigma}(Y',Y')) \\
& & \widetilde{pre}(rem(S,v), \exists Y'' \,.\, \chi^v(S,T,Y'') = X \wedge \bigvee\limits_{i=1,2} \psi_i^{\tau,\tau}(Y'',Y''))
\end{array}$$

Next we apply the weakest precondition with respect to $rem(S,v)$:

$$\begin{array}{rcl}
\Phi & = & \exists S \exists T \exists X \exists Y' \exists Z' \exists Y'' \exists Z'' \,.\, \langle S\rangle u \wedge \langle T\rangle w \wedge \\
& & Z' \setminus Sv\Sigma^* = Y' \wedge \chi^v(S,T,Y') = X \wedge Z'' \setminus Sv\Sigma^* = Y'' \wedge \chi^v(S,T,Y'') = X \wedge \\
& & \bigvee\limits_{i,j=1,2} \psi_i^{\sigma,\sigma\setminus Sv\Sigma^*}(Y',Z') \wedge \psi_j^{\tau,\tau\setminus Sv\Sigma^*}(Y'',Z'')
\end{array}$$

Provided that the choices of Y', Y'' and Z', Z'' are pairwise the same, it would be possible to eliminate X, Y', Y'' and rename Z', Z'' by X since the variables are all quantified. The resulting formula is, in this case:

$$\begin{array}{rcl}
\Phi & = & \exists S \exists T \exists X \,.\, \langle S\rangle u \wedge \langle T\rangle w \wedge \\
& & \bigvee\limits_{i,j=1,2} \psi_i^{\sigma,\sigma\setminus Sv\Sigma^*}(X \setminus Sv\Sigma^*, X) \wedge \psi_j^{\tau,\tau\setminus Sv\Sigma^*}(X \setminus Sv\Sigma^*, X)
\end{array}$$

We calculate, for Y denoting Y', Y'' and Z denoting Z', Z'', respectively:

$$
\begin{array}{rcl}
\chi^v(S,T,Y) = \chi(S,T,Z \setminus Sv\Sigma^*) & = & (Z \setminus Sv\Sigma^*) \cup Sv((T^{-1}S)v)^*(T^{-1}(Z \setminus Sv\Sigma^*)) \\
\text{[since } A^{-1}(B \setminus C) \subseteq A^{-1}B \setminus A^{-1}C] & \subseteq & (Z \setminus Sv\Sigma^*) \cup Sv((T^{-1}S)v)^*(T^{-1}Z \setminus T^{-1}(Sv\Sigma^*)) \\
& = & (Z \setminus Sv\Sigma^*) \cup (Sv(T^{-1}Z) \setminus Sv((T^{-1}S)v)^+\Sigma^*)
\end{array}
$$

Suppose now that $Z' \neq Z''$. Since Z', Z'' denote heap entities, by Definition 1 (C2) we have $Z' \cap Z'' = \emptyset$. As a consequence, $T^{-1}Z' \cap T^{-1}Z'' = \emptyset$ and, since Sv is a prefix set, we have $Sv(T^{-1}Z') \cap Sv(T^{-1}Z'') = \emptyset$. One easily verifies now that $\chi^v(S,T,Y') \cap \chi^v(S,T,Y'') = \emptyset$. This however contradicts with the fact that $\chi^v(S,T,Y') \cap \chi^v(S,T,Y'') = X$ and $X \neq \emptyset$, as a result of $\langle X \rangle \sigma$. Hence, the hypothesis $Z' \neq Z''$ was false, and therefore also $Y' = Y''$. Note that $Sv((T^{-1}S)v)^*(T^{-1}(X \setminus Sv\Sigma^*)) \ \cap \ \sigma \neq \emptyset$ is equivalent to $\sigma \in Sv(T^{-1}X)$ and $\sigma \notin Sv((T^{-1}S)v)\Sigma^*$ that is, to $\beta_\sigma \wedge \neg\gamma_\sigma$. We have used that $T^{-1}(Sv\Sigma^*) = (T^{-1}S)v\Sigma^*$ and that $L^+\Sigma^* = L\Sigma^*$, for an arbitrary language $L \subseteq \Sigma^*$. With the notation of Figure 7, the result is:

$$
\begin{aligned}
\Phi \ = \ & \exists S \exists T \exists X \,.\, \langle S \rangle u \wedge \langle T \rangle w \wedge \Big(\langle X \rangle (\Sigma^* \setminus Sv\Sigma^*) \wedge \beta_\sigma \wedge \neg\gamma_\sigma \wedge \beta_\tau \wedge \neg\gamma_\tau \bigvee & (5)\\
& \langle X \rangle (\sigma \setminus Sv\Sigma^*) \wedge \langle X \rangle (\tau \setminus Sv\Sigma^*) \wedge \neg(\beta_\sigma \wedge \neg\gamma_\sigma) \wedge \neg(\beta_\tau \wedge \neg\gamma_\tau) \bigvee & (6)\\
& \langle X \rangle (\sigma \setminus Sv\Sigma^*) \wedge \langle X \rangle (\Sigma^* \setminus Sv\Sigma^*) \wedge \neg(\beta_\sigma \wedge \neg\gamma_\sigma) \wedge \beta_\tau \wedge \neg\gamma_\tau \bigvee & (7)\\
& \langle X \rangle (\Sigma^* \setminus Sv\Sigma^*) \wedge \langle X \rangle (\tau \setminus Sv\Sigma^*) \wedge \beta_\sigma \wedge \neg\gamma_\sigma \wedge \neg(\beta_\tau \wedge \neg\gamma_\tau) \Big) & (8)
\end{aligned}
$$

We can substitute $\langle X \rangle (\sigma \setminus Sv\Sigma^*)$ and $\langle X \rangle (\tau \setminus Sv\Sigma^*)$ by the equivalent $\langle X \rangle \sigma \ \wedge \neg\alpha_\sigma$ and $\langle X \rangle \tau \wedge \neg\alpha_\tau$, respectively. Notice that $\langle X \rangle \sigma \setminus Sv\Sigma^* \Rightarrow \langle X \rangle (\Sigma^* \setminus Sv\Sigma^*)$ and, analogously, $\langle X \rangle \tau \setminus Sv\Sigma^* \Rightarrow \langle X \rangle (\Sigma^* \setminus Sv\Sigma^*)$, thus we can eliminate the term $\langle X \rangle (\Sigma^* \setminus Sv\Sigma^*)$ from the formulas (7) and (8). The last step before giving the result is eliminating the term $\langle X \rangle (\Sigma^* \setminus Sv\Sigma^*)$ from (5). This proceeds as follows. Suppose that $X \subseteq Sv\Sigma^*$. Then we have:

$$
T^{-1}X \subseteq T^{-1}(Sv\Sigma^*) = (T^{-1}S)v\Sigma^* \Rightarrow Sv(T^{-1}X) \subseteq Sv(T^{-1}S)v\Sigma^*
$$

The last relation however contradicts $\beta_\sigma \wedge \neg\gamma_\sigma$. Thus $\beta_\sigma \wedge \neg\gamma_\sigma \ \Rightarrow \ X \not\subseteq Sv\Sigma^* \ \Rightarrow \langle X \rangle (\Sigma^* \setminus Sv\Sigma^*)$, and therefore we can eliminate $\langle X \rangle (\Sigma^* \setminus Sv\Sigma^*)$ from (5). The resulting expression is:

$$
\begin{aligned}
\Phi \ = \ & \exists S \exists T \exists X \,.\, \langle S \rangle u \wedge \langle T \rangle w \wedge \Big(\beta_\sigma \wedge \neg\gamma_\sigma \wedge \beta_\tau \wedge \neg\gamma_\tau \bigvee \\
& \langle X \rangle \sigma \wedge \langle X \rangle \tau \wedge \neg\alpha_\sigma \wedge \neg\alpha_\tau \wedge \neg(\beta_\sigma \wedge \neg\gamma_\sigma) \wedge \neg(\beta_\tau \wedge \neg\gamma_\tau) \bigvee \\
& \langle X \rangle \sigma \wedge \neg\alpha_\sigma \wedge \neg(\beta_\sigma \wedge \neg\gamma_\sigma) \wedge \beta_\tau \wedge \neg\gamma_\tau \bigvee \\
& \langle X \rangle \tau \wedge \neg\alpha_\tau \wedge \beta_\sigma \wedge \neg\gamma_\sigma \wedge \neg(\beta_\tau \wedge \neg\gamma_\tau) \Big)
\end{aligned}
$$

This form can be now easily translated into **pAL**, using the scheme of Figure 7.

□

The proof proceeds by deriving the weakest precondition for an arbitrary alias proposition $\sigma\Diamond\tau$ (equivalently written in **wAL** using the embedding rule 1) using the rules in Figure 6. The result is then translated back from **wAL** to **pAL** using the shorthands from Figure 7. Then we can extend the result to arbitrary post-conditions using the distributivity properties for $\widetilde{pre}$, and to arbitrary sequences of statements by induction on the length of the sequence.

It is important to notice that the translations from **pAL** to **wAL** and back are logical equivalences. Since the $\widetilde{pre}$ operators defined on **wAL** formulas are sound and complete, according to the development in [2], we can infer the existence of a sound and complete weakest precondition calculus also for **pAL**.

6 Conclusions and Future Work

This paper concerns a deductive verification method for aliasing properties in imperative programming languages with destructive updating. Starting from previous work on storeless semantics and alias logic with a weakest precondition calculus **wAL**, we show that the satisfiability problem is undecidable but recursively enumerable. Next, we focus on a decidable subset **pAL** that allows to express sound and complete weakest preconditions. The kind of properties expressible in this logic are related to pointer aliasing, but also arbitrary finite heaps can be defined. We give two sound and complete proof systems for **pAL**, one based on natural deduction, and another based on analytic tableaux. The satisfiability problem for **pAL** is shown to be NP-complete. A tool based on the **pAL** framework is planned in the near future.

The main question related to the existence of a decidable program logic that can express non-trivial shape properties of heap is not fully answered. Although undecidable, the **wAL** logic offers a reach framework in which one can define decidable fragments having complete weakest precondition calculi. One such example is **pAL**. A still open question is the existence of a fragment of **wAL** that encompasses **pAL**, in which one can express properties such as reachability, circularity, etc. One such extension, called **kAL**, is currently under investigation. This logic is obtained from **pAL**, by considering words (over the heap alphabet) with integer counters (parameters indicating the repetition of a finite subword) and first order quantification over the counters. In this way we can express for instance the existence of an unbounded $next$-path between two pointers $head$ and $tail$: $\exists k\ .\ head.\{next\}^k\Diamond tail$, a property that is not expressible in **pAL**. We plan an extensive study of this logic, in order to cover both aspects of satisfiability and expressiveness.

References

[1] Benedikt, M., Reps, T., and Sagiv, M.: A decidable logic for describing linked data structures. European Symposium on Programming, (1999) LNCS, Vol. 1576, 2–19. 1.0.1

[2] M. Bozga, R. Iosif and Y. Lakhnech: Storeless Semantics and Alias Logic. Proc. ACM SIGPLAN 2003 Workshop on Partial Evaluation and Semantics Based Program Manipulation, 55 – 65. 1, 2, 2, 3.2, 5, 5

[3] M. Bozga, R. Iosif and Y. Lakhnech: On Logics of Aliasing. Technical Report TR-2004-4, VERIMAG `http://www-verimag.imag.fr/~iosif/TR-2004-4.ps`

[4] M. Bozga, R. Iosif: On Model Checking Generic Topologies. Technical Report TR-2004-10, VERIMAG `http://www-verimag.imag.fr/~iosif/TR-2004-10.ps` 2.1

[5] C. Calcagno, H. Yang and P.W. O'Hearn: Computability and Complexity Results for a Spatial Assertion Language for Data Structures. In Foundations of Software Technology and Theoretical Computer Science, LNCS, Volume 2245 (2001), 108–119 1.0.1, 2.1

[6] C. Calcagno, L. Cardelli, and A. Gordon: Deciding Validity in a Spatial Logic of Trees. In ACM Workshop on Types in Language Design and Implementation (2003) 62–73 1.0.1

[7] B. Courcelle: The expression of graph properties and graph transformations in monadic second-order logic, Chapter 5 of the "Handbook of graph grammars and computing by graph transformations, Vol. 1 : Foundations" (1997) 313–400 2.1

[8] A. Deutsch: A storeless model of aliasing and its abstractions using finite representations of right-regular equivalence relations. In Proceedings of the IEEE 1992 Conference on Computer Languages (1992) 2–13 2, 2

[9] H.D Ebbinghaus and J. Flum: Finite Model Theory. Springer-Verlag (1999) 2.1

[10] R.W. Floyd: Assigning meaning to programs, Proc. Symposium on Applied Mathematics, American Mathematical Society, 1967, Vol. 1, 19–32. 1

[11] D. Galmiche and D. Mery: Semantic Labelled Tableaux for propositional BI (without bottom). Journal of Logic and Computation, vol. 13, n. 5 (2003) 1.0.1

[12] C.A.R Hoare and He Jifeng: A Trace Model for Pointers and Objects. In Proc. ECOOP'99, LNCS, Vol. 1628 (1999) 1–18 2

[13] S. Ishtiaq and P. O'Hearn: BI as an Assertion Language for Mutable Data Structures. Proc. of 28th ACM-SIGPLAN Symposium on Principles of Programming Languages (2001) 1.0.1

[14] H. B. M. Jonkers. Abstract Storage Structures. Algorithmic Languages, North-Holland (1981) 321–343 2, 2

[15] N. Klarlund and M. I. Schwartzbach: Graphs and Decidable Transductions Based on Edge Constraints, In Proc. 19th Colloquium on Trees and Algebra in Programming, LNCS, Volume 787 (1994) 187–201 1.0.1

[16] N. Klarlund and M. I. Schwartzbach: Graph Types. In Proc. 20th Annual Symposium on Principles of Programming Languages (1993) 196–205 1.0.1

[17] A. Moeller and M. I. Schwartzbach: The Pointer Assertion Logic Engine. In Proc. ACM SIGPLAN Conference on Programming Languages Design and Implementation, (2001). 1.0.1

[18] P.W. O'Hearn and D.J. Pym: The Logic of Bunched Implications. Bulletin of Symbolic Logic, 5(2) (1999) 215–244 1.0.1

[19] P.W. O'Hearn, J.C. Reynolds and H. Yang: Local reasoning about programs that alter data structures. Computer Science Logic, LNCS, Volume 2142 (2001) 1–19 1.0.1

[20] M. O. Rabin: Decidability of second order theories and automata on infinite trees, Trans. Amer. Math. Soc. vol 141 (1969) 1.0.1

[21] G. Ramalingam: The Undecidability of Aliasing. ACM Transactions on Programming Languages and Systems, Vol 16, No 5 (1994) 1467–1471. 1

[22] John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. Proc 17th IEEE Symposium on Logic in Computer Science (2002) 1.0.1

[23] M. Sagiv, M., T. Reps and R. Wilhelm: Parametric Shape Analysis via 3-Valued Logic. ACM Transactions on Programming Languages and Systems, Vol 24, No 3 (2002), 217–298 1

[24] R. M. Smullyan: First-Order Logic. Dover Publications (1993) 4, 4

[25] D. van Dalen: Logic and Structure. Springer-Verlag (1997) 3.1, 3.1.1