A First Order Logic with Frames

Christof Löding
Department of Computer Science
RWTH Aachen University
Email: loeding@automata.rwth-aachen.de

P. Madhusudan[*], Adithya Murali[†], and Lucas Peña[‡]
Department of Computer Science
University of Illinois at Urbana Champaign
Email: [*]madhu@illinois.edu, [†]adithya5@illinois.edu,
[‡]lpena7@illinois.edu

Abstract—We propose a novel logic, called *Frame Logic* (FL), that extends first-order logic (with recursive definitions) using a construct $Fr(\cdot)$ that captures the *implicit frames* of formulas— the precise subset of the universe upon which their meaning depends. Using such frames, we formulate proof rules that facilitate frame reasoning elegantly when the underlying model undergoes change. We show that the logic is expressive by expressing several properties of data-structures and also exhibit a translation from a separation logic that defines *precise formulas* to frame logic. Finally, we design a program logic based on frame logic for reasoning with programs that dynamically update heaps that facilitates local specifications and frame reasoning. We present both localized proof rules as well as rules that can derive the weakest tightest preconditions in FL. We show that verification conditions for recursion-free program blocks can be derived also in pure first-order logic with recursive definitions with an appropriate frame rule, facilitating frame reasoning.

I. Introduction

Program logics for expressing and reasoning with programs that dynamically manipulate heaps is an active area of research. The research on separation logic has argued convincingly that it is highly desirable to have *localized logics* that talk about small states (heaplets rather than the global heap), and the ability to do *frame reasoning*. Separation logic achieves this objective by having a tight heaplet semantics and using special operators, primarily a separating conjunction operator $*$ and a separating implication operator (the magic wand $-*$).

In this paper, we ask a fundamental question: can classical logics (such as FOL and FOL with recursive definitions) be extended to support localized specifications and frame reasoning? Can we utilize classical logics for reasoning effectively with programs that dynamically manipulate heaps, with the aid of local specifications and frame reasoning?

The primary contribution of this paper is to endow a classical logic, namely first-order logic with recursive definitions (with least fixpoint semantics) with frames and frame reasoning.

A formula in first-order logic with recursive definitions (FO-RD) can be naturally associated with a *frame*— the subset of the universe that determines its truth. By using a more careful syntax such as guarded quantification (which continue to have a classical interpretation), we can in fact write specifications in FO-RD that have very precise frames. For example, we can write the property that x points to a linked list using a formula $list(x)$ written purely in FO-RD so that its frame is precisely the locations constituting the linked list.

In this paper, we define an extension of FO-RD, called Frame Logic (FL) where we allow a new operator $Fr(\alpha)$ which, for an FO-RD formula α, evaluates to the frame of α. Logical formulas thus have access to frames and can use it to *separate* frames and do frame reasoning. For instance, the logic can now express that two lists are disjoint by asserting that $Fr(list(x)) \cap Fr(list(y)) = \emptyset$. It can then reason that in such a program heap configuration, if the program manipulates only the locations in $Fr(list(y))$ then $list(x)$ would continue to be true, using simple frame reasoning.

The addition of the frame operator to FO-RD yields a very natural logic for expressing specifications. First, formulas in FO-RD have the same meaning as they have in FL, i.e., retain their classical meaning. For example, $f(x) = y$ (written in FO-RD as well as in FL) is true in any model that has x mapped by f to y, instead of a specialized "tight heaplet semantics" that demands that f be a partial function with domain only consisting of the location x. The frame that contains only the location x is important, of course, but is made accessible using the frame operator, i.e., $Fr(f(x) = y)$ gives the set containing the sole element interpreted for x. Second, expressing properties of frames can be naturally expressed using set operations. To state that the lists pointed to be x and y are disjoint, we don't need special operators (such as the $*$ operator in separation logic) but can express this using $Fr(list(x)) \cap Fr(list(y))$ is empty.

Third, when used to annotate programs, pre/post specifications for programs written in FL can be made *implicitly* local by interpreting their frames to be the localized heaplets accessed and modified by programs, yielding frame reasoning akin to separation logic. Finally, as we show in this paper, the weakest precondition of specifications across basic loop-free paths can be expressed in FL, making it an expressive logic for

reasoning with programs. Separation logic, on the other hand, introduces the magic wand operator $-\ast$ in order to add enough expressiveness to be closed under weakest preconditions [35].

We define frame logic (FL) as an extension of FO with recursive definitions (FO-RD) that operates over a multi-sorted universe, with a particular foreground sort (used to model locations on the heap) and several background sorts that are defined using separate theories. Frames for formulas are defined with respect to the foreground sort only. A special background sort of *sets* of elements of the foreground sort is assumed and is used to model the frames for formulas. For any formula φ in the logic, we have a special formula $Fr(\varphi)$ that captures its frame, a set of locations in the foreground sort, that intuitively corresponds to the precise subdomain of functions the value of φ depends on. We then prove a *frame theorem* (Theorem 1) that says that changing a model M by changing the interpretation of functions that are not in the frame of φ, $Fr(\varphi)$, will not affect the truth of the formula. This theorem then directly supports frame reasoning; if a model satisfies φ and the model is changed so that the changes made are disjoint from the frame of φ, then φ will continue to hold. We also show that FL formulae can be translated to vanilla FO-RD logic (without frame operators); in other words, the semantics for the frame of a formula can be captured in FO-RD itself. Consequently, we can use any FO-RD reasoning mechanism (proof systems [20], [19] or heuristic algorithms such as the natural proof techniques [22], [29], [34], [37]) to reason with FL formulas.

We illustrate our logic using several examples drawn from program verification; we show how to express various data-structure definitions and the elements they contain and various measures for them using FL formulas (e.g., linked lists, sorted lists, list segments, binary search trees, AVL trees, lengths of lists, heights of trees, set of keys stored in the datastructure, etc.)

While the sensibilities of our logic are definitely inspired by separation logic, there are some fundamental differences (beyond the fact that our logic extends the syntax and semantics of classical logics with a special frame operator, avoiding operators such that $\ast$ and $-\ast$). In separation logic, there can be many frames of a formula (also called heaplets)— a heaplet for a formula is one that *supports its truth*. For example, a formula of the form $\alpha \vee \beta$ can have a heaplet that supports the truth of α or one that supports the heap of β. However, the philosophy that we follow in our design is to have a *single* frame that supports the truth value of a formula, whether it be *true or false*. Consequently, the frame of the formula $\alpha \vee \beta$ is the *union* of the frames of the formulas α and β.

The above design choice of the frame being *determined* by the formula has several consequences that lead to a deviation from separation logic. For instance, the frame of the negation of a formula φ is the same as the frame of φ. And the frame of the formula $f(x) = y$ and its negation are the same, namely the singleton location interpreted for x. In separation logic, the corresponding formula will have the same heaplet but its negation will include *all* other heaplets. The choice of having determined frames or heaplets is not new, and there have been several variants and sublogics of separation logics that have been explored. For example, the logic DRYAD [34], [29] is a separation logic that insists on determined heaplets to support automated reasoning, and the *precise* fragment of separation logic studied in the literature [27] defines a sublogic that has (essentially) determined heaplets. The second main contribution in this paper is to show that this fragment of separation logic (with slight changes for technical reasons) can be translated to frame logic, such that the unique heaplet that satisfies a precise separation logic formula is its frame of the corresponding formula in FL.

The third main contribution of this paper is a program logic based on frame logic for a simple while-programming language destructively updating heaps. We present two kinds of proof rules for reasoning with such programs annotated with pre- and post-conditions written in frame logic. The first set of rules are local rules that axiomatically define the semantics of the program, using the smallest frames for each command. We also give a frame rule that allows arguing preservation of properties whose frames are disjoint from the heaplet modified by a program. These rules are similar to analogous rules in separation logic. The second class of rules work to give a *weakest tightest precondition* for any postcondition with respect to non-recursive programs. In separation logic, the corresponding rules for weakest preconditions are often expressed using separating implication (the magic-wand operator). Given a small change made to the heap and a postcondition β, the formula $\alpha -\ast \beta$ captures all heaplets H where if a heaplet that satisfies α is joined with H, then β holds. When α describes the change effected by the program, $\alpha -\ast \beta$ captures, essentially, the weakest pre-condition. However, the magic wand is a very powerful operator that calls for quantifications over heaplets and submodels, and hence involves second order quantification. In our logic, we show that we can capture the weakest precondition with only first-order quantification, and hence first-order frame logic is closed under weakest preconditions across non-recursive programs blocks. This immediately means that as long as loop invariants are given also in FL, verification conditions in FL can be derived using our proof rules. By translating these to pure FO-RD formulas, we can

use FO-RD reasoning techniques to prove them valid.

In summary, the contributions of this paper are:

- A logic, called *frame logic* (FL) that extends FO-RD with a frame operator and supports frame reasoning. We illustrate FL with specifications of various data-structures. We also show a translation to equivalent formulas in FO-RD.
- A program logic and proof system based on FL including local rules and rules for computing the weakest tightest precondition. This directly results in a technique to obtain verification conditions for non-recursive programs in FL as well as FO-RD.
- A separation logic fragment that can generate only precise formulas, and a translation from this logic to equivalent FL formulas.

II. BACKGROUND: FIRST ORDER LOGIC WITH RECURSIVE DEFINITIONS AND UNINTERPRETED COMBINATIONS OF THEORIES

The base logic upon which we build frame logic is a first order logic with recursive definitions (FO-RD), where we allow a foreground sort and several background sorts, each with their individual theories (like arithmetic, sets, arrays, etc.). The foreground sort and functions involving the foreground sort are *uninterpreted* (not constrained by theories). This hence can be seen as an uninterpreted combination of theories over disjoint domains. This logic has been defined and used to model heap verification before [21].

We will build frame logic over such a framework where frames are modeled as subsets of elements of the foreground sort. When modeling heaps in program verification using logic, the foreground sort will be used to model *locations of the heap*, uninterpreted functions from the foreground sort to foreground sort will be used to model *pointers*, and uninterpreted functions from the foreground sort to the background sort will model *data fields*. Consequently, frames will be subsets of locations of the heap, which is appropriate as these are the domains of pointers that change when a program updates a heap.

We define a signature as $\Sigma = (S; C; F; \mathcal{R}; \mathcal{I})$, where S is a finite non-empty set of sorts. C is a set of constant symbols, where each $c \in C$ has some sort $\tau \in S$. F is a set of function symbols, where each function $f \in F$ has a type of the form $\tau_1 \times \ldots \times \tau_m \to \tau$ for some m, with $\tau_i, \tau \in S$. The sets $\mathcal{R}$ and $\mathcal{I}$ are (disjoint) sets of relation symbols, where each relation $R \in \mathcal{R} \cup \mathcal{I}$ has a type of the form $\tau_1 \times \ldots \times \tau_m$. The set $\mathcal{I}$ contains those relation symbols for which the corresponding relations are inductively defined using formulas (details are given below), while those in $\mathcal{R}$ are given by the model.

We assume that the set of sorts contains a designated "foreground sort" denoted by σ_f. All the other sorts in S are called background sorts, and for each such background sort σ we allow the constant symbols of type σ, function symbols that have type $\sigma^n \to \sigma$ for some n, and relation symbols have type σ^m for some m, to be constrained using an arbitrary theory T_σ.

A formula in first-order logic with recursive definitions (FO-RD) over such a signature is of the form $(\mathcal{D}, \alpha)$, where $\mathcal{D}$ is a set of recursive definitions of the form $R(\vec{x}) := \rho_R(\vec{x})$, where $R \in \mathcal{I}$ and $\rho_R(\vec{x})$ is a first-order logic formula, in which the relation symbols from $\mathcal{I}$ occur only positively. α is also a first-order logic formula over the signature. We assume $\mathcal{D}$ has at most one definition for any inductively defined relation, and that the formulas ρ_R and α use only inductive relations defined in $\mathcal{D}$.

The semantics of a formula is standard; the semantics of inductively defined relations are defined to be the least fixpoints that satisfies the relational equations, and the semantics of α is the standard one defined using these semantics for relations. We do not formally define the semantics, but we will formally define the semantics of frame logic (discussed in the next section and whose semantics is defined in the Appendix) which is an extension of FO-RD.

III. FRAME LOGIC

We now define Frame Logic, the central contribution of this paper.

We consider a universe with a foreground sort and several background sorts, each restricted by individual theories, as described in Section II. We consider the elements of the foreground sort to be *locations* and consider frames as *sets of locations*, i.e., sets of elements of the foreground sort. We hence introduce a background sort $\sigma_{S(f)}$; the elements of sort $\sigma_{S(f)}$ model sets of elements of sort σ_f. Among the relation symbols in $\mathcal{R}$ there is the relation $\in$ of type $\sigma_f \times \sigma_{S(f)}$ that is interpreted as the usual element relation. The signature includes the standard operations on sets $\cup, \cap$ with the usual meaning, the unary function $\tilde{\ }$ that is interpreted as the complement on sets (with respect to the set of foreground elements), and the constant $\emptyset$. For these functions and relations we assume a background theory $B_{\sigma_{S(f)}}$ that is an axiomatisation of the theory of sets. We further assume that the signature does not contain any other function or relation symbols involving the sort $\sigma_{S(f)}$.

For reasoning about changes of the structure over the locations, we assume that there is a subset $F_m \subseteq F$ of function symbols that are declared mutable. These functions can be used to model mutable pointer fields in the heap that can be manipulated by a program and thus change. Formally, we require that each $f \in F_m$ has at least one argument of sort σ_f.

FL formulas: φ $::=$ $t_\tau = t_\tau \mid R(t_{\tau_1}, \ldots, t_{\tau_m}) \mid \varphi \wedge \varphi \mid \neg\varphi \mid ite(\gamma : \varphi, \varphi) \mid \exists y : \gamma.\ \varphi$
$\qquad\qquad\qquad \tau \in S,\ R \in \mathcal{R} \cup \mathcal{I}$ of type $\tau_1 \times \cdots \times \tau_m$

Guards: γ $::=$ $t_\tau = t_\tau \mid R(t_{\tau_1}, \ldots, t_{\tau_m}) \mid \gamma \wedge \gamma \mid \neg\gamma \mid ite(\gamma : \gamma, \gamma) \mid \exists y : \gamma.\ \gamma$
$\qquad\qquad\qquad \tau \in S \setminus \{\sigma_{\mathsf{S(f)}}\},\ R \in \mathcal{R}$ of type $\tau_1 \times \cdots \times \tau_m$

Terms: t_τ $::=$ $c \mid x \mid f(t_{\tau_1}, \ldots, t_{\tau_m}) \mid ite(\gamma : t_\tau, t_\tau) \mid Fr(\varphi)$ (if $\tau = \sigma_{\mathsf{S(f)}}$) $\mid Fr(t_{\tau'})$ (if $\tau = \sigma_{\mathsf{S(f)}}$)
$\qquad\qquad\qquad \tau, \tau' \in S$ with constants c, variables x of type τ, f of type $\tau_1 \times \cdots \times t_m \to \tau$

Recursive definitions: $R(\vec{x}) := \rho_R(\vec{x})$ with $R \in \mathcal{I}$ of type $\tau_1 \times \cdots \times \tau_m$ with $\tau_i \in S \setminus \{\sigma_{\mathsf{S(f)}}\}$
$\qquad\qquad\qquad$ frame logic formula $\rho_R(\vec{x})$ in which all relation symbols $R' \in \mathcal{I}$ occur only
$\qquad\qquad\qquad$ positively or inside a frame expression.

Fig. 1. Syntax of frame logic: γ for guards, t_τ for terms of sort τ, and general formulas φ. Guards cannot use inductively defined relations or frame expressions.

For variables, let Var_τ denote the set of variables of sort τ, where $\tau \in S$. We let $\overline{x}$ abbreviate tuples $x_1, \ldots, x_n$ of variables.

Our frame logic over uninterpreted combinations of theories is a variant of first-order logic with recursive definitions that has an additional operator $Fr(\varphi)$ that assigns to each formula φ a set of elements (its frame or "heaplet" in the context of heaps) in the foreground universe. So $Fr(\varphi)$ is a term of sort $\sigma_{\mathsf{S(f)}}$.

The intended semantics of $Fr(\varphi)$ (and of the inductive relations) is defined formally as a least fixpoint of a set of equations. This semantics is presented in Section III-C. In the following, we first define the syntax of the logic, then discuss informally the various design decisions for the semantics of frames, before proceeding to a formal definition of the semantics

A. Syntax of Frame Logic (FL)

The syntax of our logic is given is the grammar in Figure 1. This extends FO-RD with the rule for building *frame expressions*, which are terms of sort $\sigma_{\mathsf{S(f)}}$ of the form $Fr(\alpha)$ for a formula α, or $Fr(t)$ for a term t.

The formulas defined by γ are used as *guards* in existential quantification and in the if-then-else-operator, which is denoted by ite. The restriction compared to general formulas is that guards cannot use inductively defined relations (R ranges only over $\mathcal{R}$ in the rule for γ, and over $\mathcal{R} \cup \mathcal{I}$ in the rule for φ), nor terms of sort $\sigma_{\mathsf{S(f)}}$ and thus no frame expressions (τ ranges over $S \setminus \{\sigma_{\mathsf{S(f)}}\}$ in the rules for γ and over S in the rule for φ). The requirement that the guard does not use the inductive relations and frame expressions is used later to ensure the existence of least fixpoints for defining semantics of inductive definitions. The semantics of an ite-formula $ite(\gamma, \alpha, \beta)$ is the same as the one of $(\gamma \wedge \alpha) \vee (\neg\gamma \wedge \beta)$; however, the *frames* of the two formulas will turn out to be different (i.e., $Fr(ite(\gamma : \alpha, \beta))$ and $Fr((\gamma \wedge \alpha) \vee (\neg\gamma \wedge \beta))$ are different), as explained in Section III-B. The same is true for existential formulas, i.e., $\exists y : \gamma.\varphi$ has the same semantics as $\exists y.\gamma \wedge \varphi$ but, in general, has a different frame.

For recursive definitions (throughout the paper, we use the terms recursive definitions and inductive definitions

with the same meaning), we require that the relation R that is defined does not have arguments of sort $\sigma_{\mathsf{S(f)}}$. This is another restriction in order to ensure the existence of a least fixpoint model in the definition of the semantics.[1]

B. Semantics of Frame Expressions: Design Decisions

We discuss the design decisions that go behind the semantics of the frame operator Fr in our logic, and then give an example for the frame of an inductive definition. The formal conditions that the frames should satisfy are stated in the equations in Figure 2, and are explained in Section III-C. Here, we start by an informal discussion.

The first decision is to have every formula define uniquely a frame, which roughly captures the subdomain of mutable functions that a formula φ's truthhood depends on, and have $Fr(\varphi)$ evaluate to it.

The choice for frames of atomic formulae are relatively clear. An atomic formula of the kind $f(x) = y$, where x is of the foreground sort and f is a mutable function, has as its frame the singleton set containing the location interpreted for x. And atomic formulas that do not involve mutable functions over the foreground have an empty frame. Frames for terms can also be similarly defined. The frame of a conjunction $\alpha \wedge \beta$ should clearly be the union of the frames of the two formulas.

Remark 1. *In traditional separation logic, each pointer field is stored in a separate location, using integer offsets. However, in our work, we view pointers as references and disallow pointer arithmetic. A more accurate heaplet for such references can be obtained by taking heaplet to be the pair (x, f) (see [28]), capturing the fact that the formula depends only on the field f of x. Such accurate heaplets can be captured in FL as well— we can introduce a non-mutable field lookup pointer L_f and use $x.L_f.f$ in programs instead of $x.f$.*

What should the frame of a formula $\alpha \vee \beta$ be? The choice we make here is that its frame is the *union* of the frames of α and β. Note that in a model where α is true and β is false, we still include the heaplet of β

---

[1]It would be sufficient to restrict formulas of the form $R(t_1, \ldots, t_n)$ for inductive relations R to not contain frame expressions as subterms.

in $Fr(\alpha \vee \beta)$. In a sense, this is an overapproximation of the frame as far as frame reasoning goes, as surely preserving the model's definitions on the frame of α will preserve the truth of α, and hence of $\alpha \vee \beta$.

However, we prefer the frame to be the union of the frames of α and β. We think of the frame as the subdomain of the universe that determines the meaning of the formula, whether it be *true* or *false*. Consequently, we would like the frame of a formula and its negation to be the same. Given that the frame of the negation of a disjunction, being a conjunction, is the union of the frames of α and β, we would like this to be the frame.

Separation logic makes a different design decision. Logical formulas are not associated with tight frames, but rather, the semantics of the formula is defined for models with given frames/heaplets, where the idea of a heaplet is whether it supports the *truthhood* of a formula (and not its falsehood). For example, for a model, the various heaplets that satisfy $\neg(f(x) = y)$ in separation logic would include all heaplets where the location of x is not present, which does not coincide with the notion we have chosen for frames. However, for positive formulas, separation logic handles frames more accurately, as it can associate several frames for a formula, yielding two heaplets for formulas of the form $\alpha \vee \beta$ when they are both true in a model. The decision to have a single frame for a formula compels us to take the union of the frames to be the frame of a disjunction.

There are situations, however, where there are disjunctions $\alpha \vee \beta$, where only *one* of the disjuncts can possibly be true, and hence we would like the frame of the formula to be the frame of the disjunct that happens to be true. We therefore introduce a new syntactical form $ite(\gamma : \alpha, \beta)$ in frame logic, whose heaplet is the union of the frames of γ and α, if γ is true, and the frames of γ and β if γ is false. While the truthhood of $ite(\gamma : \alpha, \beta)$ is the same as that of $(\gamma \wedge \alpha) \vee (\neg\gamma \wedge \beta)$, its frames are potentially smaller, allowing us to write formulas with tighter frames to support better frame reasoning. Note that the frame of $ite(\gamma : \alpha, \beta)$ and its negation $ite(\neg\gamma : \neg\alpha, \neg\beta)$ are the same, as we desired.

Turning to quantification, the frame for a formula of the form $\exists x.\alpha$ is hard to define, as its truthhood could depend on the entire universe. We hence provide a mechanism for *guarded* quantification, in the form $\exists x : \gamma.\ \alpha$. The semantics of this formula is that there exists some location that satisfies the guard γ, for which α holds. The frame for such a formula includes the frame of the guard, and the frames of α when x is interpreted to be a location that satisfies γ. For example, $\exists x : (x = f(y)).\ g(x) = z$ has as its frame the locations interpreted for y and $f(y)$ only.

For a formula $R(\overline{t})$ with an inductive relation R defined by $R(\overline{x}) := \rho_R(\overline{x})$, the frame descends into

the definition, changing the variable assignment of the variables in $\overline{x}$ from the inductive definition to the terms in $\overline{t}$. Furthermore, it contains the elements to which mutable functions are applied in the terms in $\overline{t}$.

Recursive definitions are designed such that the evaluation of the equations for the frame expressions is independent of the interpretation of the inductive relations. The equations mainly depend on the syntactic structure of formulas and terms. Only the semantics of guards, and the semantics of subterms under a mutable function symbol play a role. For this reason, we disallow guards to contain recursively defined relations or frame expressions. We also require that the only functions involving the sort $\sigma_{S(f)}$ are the standard functions involving sets. Thus, subterms of mutable functions cannot contain frame expressions (which are of sort $\sigma_{S(f)}$) as subterms.

These restrictions ensure that there indeed exists a unique simultaneous least solution of the equations for the inductive relations and the frame expressions.

We end this section with an example.

Example 1. *Consider the definition of a predicate* $tree(x)$ *w.r.t. two unary mutable functions ℓ and r:*

$$
\begin{aligned}
tree(x) := \ \ & ite(x = nil : true, \alpha) \ where \\
\alpha = \ \ & tree(\ell(x)) \wedge tree(r(x)) \wedge \\
& Fr(tree(\ell(x))) \cap Fr(tree(r(x))) = \emptyset \wedge \\
& \neg(x \in Fr(tree(\ell(x))) \cup Fr(tree(r(x))))
\end{aligned}
$$

The above inductive definition defines binary trees with pointer fields ℓ and r for left- and right-pointers. The definition says that x points to a tree if either x is equal to nil (and in this case its frame is empty), or $\ell(x)$ and $r(x)$ are trees and its frames are disjoint. The last conjunct says that x does not belong to the frame of the left and right subtrees; this condition is, strictly speaking, not required to define trees (as least fixpoint semantics will ensure this anyway). Note that the access to the frame of formulas eases defining disjointness of heaplets, like in separation logic. The frame of $tree(x)$ turns out to be precisely the nodes that are reachable from x using ℓ and r pointers, as one would desire. Consequently, if a pointer outside this frame changes, we would be able to conclude using frame reasoning that the truth value of $tree(x)$ does not change. ☐

C. Formal Semantics of Frame Logic

Before we explain the semantics of the frame expressions and inductive definitions, we introduce a semantics that treats frame expressions and the symbols from $\mathcal{I}$ as uninterpreted symbols. We refer to this semantics as *uninterpreted semantics*. For the formal definition we need to introduce some terminology first.

An occurrence of a variable x in a formula is free if it does not occur under the scope of a quantifier for x. By renaming variables we can assume that each

$$
\begin{aligned}
\llbracket Fr(c) \rrbracket_M(\nu) &= \llbracket Fr(x) \rrbracket_M(\nu) = \emptyset \text{ for a constant } c \text{ or variable } x \\[4pt]
\llbracket Fr(f(t_1,\ldots,t_n)) \rrbracket_M(\nu) &=
\begin{cases}
\displaystyle\bigcup_{i \text{ with } t_i \text{ of sort } \sigma_{\mathsf f}} \{\llbracket t_i \rrbracket_{M,\nu}\} \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_M(\nu) & \text{if } f \in F_{\mathsf m} \\[6pt]
\displaystyle\bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_M(\nu) & \text{if } f \notin F_{\mathsf m}
\end{cases} \\[4pt]
\llbracket Fr(Fr(\varphi)) \rrbracket_M(\nu) &= \llbracket Fr(\varphi) \rrbracket_M(\nu) \\
\llbracket Fr(Fr(t)) \rrbracket_M(\nu) &= \llbracket Fr(t) \rrbracket_M(\nu) \\
\llbracket Fr(t_1 = t_2) \rrbracket_M(\nu) &= \llbracket Fr(t_1) \rrbracket_M(\nu) \cup \llbracket Fr(t_2) \rrbracket_M(\nu) \\
\llbracket Fr(R(t_1,\ldots,t_n)) \rrbracket_M(\nu) &= \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_M(\nu) \text{ for } R \in \mathcal{R} \\
\llbracket Fr(R(\overline{t})) \rrbracket_M(\nu) &= \llbracket Fr(\rho_R(\overline{x})) \rrbracket_M(\nu[\overline{x} \leftarrow \llbracket \overline{t} \rrbracket_{M,\nu}]) \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_M(\nu) \\
& \quad \text{for } R \in \mathcal{I} \text{ with definition } R(\overline{x}) := \rho_R(\overline{x}), \overline{t} = (t_1,\ldots,t_n), \overline{x} = (x_1,\ldots,x_n) \\
\llbracket Fr(\alpha \wedge \beta) \rrbracket_M(\nu) &= \llbracket Fr(\alpha) \rrbracket_M(\nu) \cup \llbracket Fr(\beta) \rrbracket_M(\nu) \\
\llbracket Fr(\neg\varphi) \rrbracket_M(\nu) &= \llbracket Fr(\varphi) \rrbracket_M(\nu) \\[4pt]
\llbracket Fr(ite(\gamma : \alpha, \beta)) \rrbracket_M(\nu) &= \llbracket Fr(\gamma) \rrbracket_M(\nu) \cup
\begin{cases}
\llbracket Fr(\alpha) \rrbracket_M(\nu) & \text{if } M \models \gamma \\
\llbracket Fr(\beta) \rrbracket_M(\nu) & \text{if } M \not\models \gamma
\end{cases} \\[4pt]
\llbracket Fr(ite(\gamma : t_1, t_2)) \rrbracket_M(\nu) &= \llbracket Fr(\gamma) \rrbracket_M(\nu) \cup
\begin{cases}
\llbracket Fr(t_1) \rrbracket_M(\nu) & \text{if } M \models \gamma \\
\llbracket Fr(t_2) \rrbracket_M(\nu) & \text{if } M \not\models \gamma
\end{cases} \\[4pt]
\llbracket Fr(\exists y : \gamma.\varphi) \rrbracket_M(\nu) &= \bigcup_{u \in D_y} \llbracket Fr(\gamma) \rrbracket_M(\nu[y \leftarrow u]) \cup \bigcup_{u \in D_y; M,\nu[y \leftarrow u] \models \gamma} \llbracket Fr(\varphi) \rrbracket_M(\nu[y \leftarrow u])
\end{aligned}
$$

Fig. 2. Equations for frame expressions.

variable only occurs freely in a formula or is quantified by exactly one quantifier in the formula. We write $\varphi(x_1,\ldots,x_k)$ to indicate that the free variables of φ are among $x_1,\ldots,x_k$. Substitution of a term t for all free occurrences of variable x in a formula φ is denoted $\varphi[t/x]$. Multiple variables are substituted simultaneously as $\varphi[t_1/x_1,\ldots,t_n/x_n]$. We abbreviate this by $\varphi[\overline{t}/\overline{x}]$.

A model is of the form $M = (U; \llbracket \cdot \rrbracket_M)$ where $U = (U_\sigma)_{\sigma \in S}$ contains a universe for each sort, and an interpretation function $\llbracket \cdot \rrbracket_M$. The universe for the sort $\sigma_{\mathsf{S(f)}}$ is the powerset of the universe for $\sigma_{\mathsf f}$.

A variable assignment is a function ν that assigns to each variable a concrete element from the universe for the sort of the variable. For a variable x, we write D_x for the universe of the sort of x (the domain of x). For a variable x and an element $u \in D_x$ we write $\nu[x \leftarrow u]$ for the variable assignment that is obtained from ν by changing the value assigned for x to u.

The interpretation function $\llbracket \cdot \rrbracket_M$ maps each constant c of sort σ to an element $\llbracket c \rrbracket_M \in U_\sigma$, each function symbol $f : \tau_1 \times \ldots \times \tau_m \to \tau$ to a concrete function $\llbracket f \rrbracket_M : U_{\tau_1} \times \ldots \times U_{\tau_m} \to U_\tau$, and each relation symbol $R \in \mathcal{R} \cup \mathcal{I}$ of type $\tau_1 \times \ldots \times \tau_m$ to a concrete relation $\llbracket R \rrbracket_M \subseteq U_{\tau_1} \times \ldots \times U_{\tau_m}$. These interpretations are assumed to satisfy the background theories (see Section II). Furthermore, the interpretation function maps each expression of the form $Fr(\varphi)$ to a function $\llbracket Fr(\varphi) \rrbracket_M$ that assigns to each variable assignment ν a set $\llbracket Fr(\varphi) \rrbracket_M(\nu)$ of foreground elements. The set $\llbracket Fr(\varphi) \rrbracket_M(\nu)$ corresponds to the frame of the formula when the free variables are interpreted by ν. Similarly, $\llbracket Fr(t) \rrbracket_M$ is a function from variable assignments to sets of foreground elements.

Based on such models, we can define the semantics of terms and formulas in the standard way. The only construct that is non-standard in our logic are terms of the form $Fr(\varphi)$, for which the semantics is directly given by the interpretation function. We write $\llbracket t \rrbracket_{M,\nu}$ for the interpretation of a term t in M with variable assignment ν. With this convention, $\llbracket Fr(\varphi) \rrbracket_M(\nu)$ denotes the same thing as $\llbracket Fr(\varphi) \rrbracket_{M,\nu}$. As usual, we write $M, \nu \models \varphi$ to indicate that the formula φ is true in M with the free variables interpreted by ν, and $\llbracket \varphi \rrbracket_M$ denotes the relation defined by the formula φ with free variables $\overline{x}$.

We refer to the above semantics as the *uninterpreted semantics* of φ because we do not give a specific meaning to inductive definitions and frame expressions.

Now let us define the true semantics for FL. The relation symbols $R \in \mathcal{I}$ represent inductively defined relations, which are defined by equations of the form $R(\overline{x}) := \rho_R(\overline{x})$ (see Figure 1).

In the intended meaning, R is interpreted as the least relation that satisfies the equation

$$\llbracket R(\overline{x}) \rrbracket_M = \llbracket \rho_R(\overline{x}) \rrbracket_M.$$

The usual requirement for the existence of a unique least fixpoint of the equation is that the definition of R does not negatively depend on R. For this reason, we require that in $\rho_R(\overline{x})$ each occurrence of an inductive predicate $R' \in \mathcal{I}$ is either inside a frame expression, or it occurs under an even number of negations.[2]

Every frame expression is evaluated on a model to a set of foreground elements (under a given variable assignment ν). Formally, we are interested in models in which the frame expressions are interpreted to be the sets

---

[2] As usual, it would be sufficient to forbid negative occurrences of inductive predicates in mutual recursion.

that correspond to the *smallest solution of the equations given in Figure 2*. The intuition behind these definitions was explained in Section III-B

Example 2. *Consider the inductive definition $tree(x)$ defined in Example 1. To check whether the equations from Figure 2 indeed yield the desired frame, note that the frames of $Fr(x = nil) = Fr(x) = Fr(true) = \emptyset$, and $Fr(\alpha) = Fr(tree(\ell(x))) \cup Fr(tree(r(x)))$ for all variable assignments. Below, we write $[u]$ for a variable assignment that assigns u to x. Then we obtain that $Fr(tree(x))[u] = \emptyset$ if $u = nil$, and $Fr(tree(x))[u] = Fr(tree(\ell(x)))[u] \cup Fr(tree(r(x)))[u]$ if $x \neq nil$. So we need to determine the frames for the formulas $tree(\ell(x))$ and $tree(r(x))$ at u. The equations yield $Fr(tree(\ell(x)))[u] = V \cup Fr(\ell(x))[u]$ with*

$$V = \begin{cases} \emptyset \text{ if } \ell(u) = nil \\ Fr(tree(\ell(x)))[\ell(u)] \cup Fr(tree(r(x)))[\ell(u)] \text{ else} \end{cases}$$

Note that in the second case, the variable assignment has changed from u to $\ell(u)$. Since $Fr(\ell(x))[u] = \{u\}$, the element u enters the frame, and the recursion further descends into the subtree of $\ell(u)$ as desired. The case of $tree(r(x))$ is analogous. □

A *frame model* is a model in which the interpretation of the inductive relations and of the frame expressions corresponds to the least solution of the respective equations (see Appendix VIII-A for a rigorous formalisation).

Proposition 1. *For each model M, there is a unique frame model over the same universe and the same interpretation of the constants, functions, and non-inductive relations.*

D. A Frame Theorem

The frame of a formula can be used for frame reasoning in the following sense: if we modify a model M by changing the interpretation of the mutable functions (e.g., a program modifying pointers), then truth values of formulas do not change if the change happens outside the frame of the formula. This is formalized below.

Given two models M, M' over the same universe, we say that M' is a *mutation of M* if $[\![R]\!]_M = [\![R]\!]_{M'}$, $[\![c]\!]_M = [\![c]\!]_{M'}$, and $[\![f]\!]_M = [\![f]\!]_{M'}$, for all constants c, relations $R \in \mathcal{R}$, and functions $f \in F \setminus F_{\mathsf{m}}$. In other words, M can only be different from M' on the interpretations of the mutable functions, the inductive relations, and the frame expressions.

Given a subset $X \subseteq U_{\sigma_f}$ of the elements from the foreground universe, we say that the *mutation is stable on X* if the values of the mutable functions did not change on arguments from X, that is, $[\![f]\!]_M(u_1, \ldots, u_n) = [\![f]\!]_{M'}(u_1, \ldots, u_n)$ for all mutable functions $f \in F_{\mathsf{m}}$ and all appropriate tuples $u_1, \ldots, u_n$ of arguments with $\{u_1, \ldots, u_n\} \cap X \neq \emptyset$.

Theorem 1 (Frame Theorem). *Let M, M' be frame models such that M' is a mutation of M that is stable on $X \subseteq U_{\sigma_f}$, and let ν be a variable assignment. Then $M, \nu \models \alpha$ iff $M', \nu \models \alpha$ for all formulas α with $[\![Fr(\alpha)]\!]_M(\nu) \subseteq X$, and $[\![t]\!]_{M,\nu} = [\![t]\!]_{M',\nu}$ for all terms t with $[\![Fr(t)]\!]_M(\nu) \subseteq X$.*

Proof. See Appendix VIII-B. □

E. Reduction from Frame Logic to FO-RD

The only extension of frame logic compared to FO-RD is the frame operator Fr, which defines a function from interpretations of free variables to sets of foreground elements. The semantics of the frame operator can be captured within FO-RD itself, so reasoning within frame logic can be reduced to reasoning within FO-RD.

A formula $\alpha(\overline{y})$ with $\overline{y} = y_1, \ldots, y_m$ has one frame for each interpretation of the free variables. We capture these frames by an inductively defined relation $Fr_\alpha(\overline{y}, z)$ of arity $m + 1$ such that for each frame model M, we have $(u_1, \ldots, u_m, u) \in [\![Fr_\alpha]\!]_M$ if $u \in [\![Fr(\alpha)]\!]_M(\nu)$ for the interpretation ν that interprets y_i as u_i.

Since the semantics of $Fr(\alpha)$ is defined over the structure of α, we introduce corresponding inductively defined relations Fr_β and Fr_t for all subformulas β and subterms t of either α or of a formula ρ_R for $R \in \mathcal{I}$.

The equations for frames from Figure 2 can be expressed by inductive definitions for the relations Fr_β. The translations are shown in Figure 4 in the Appendix.

For the definitions, we assume that $\overline{y}$ contains all variables that are used in α and in the formulas ρ_R. We further assume that each variable either is used in at most one of the formulas α or ρ_R, and either only occurs freely in it, or is quantified at most once. The relations Fr_β are all of arity $n + 1$, even if the subformulas do not use some of the variables. In practice, one would rather use relations of arities as small as possible, referring only to the relevant variables. In a general definition, this is, however, rather cumbersome to write, so we use this simpler version in which we do not have to rearrange and adapt the variables according to their use in subformulas.

For the definition of $Fr_{R(\overline{t})}$ note that the variables in $\overline{x}$ from the definition of R are contained in $\overline{y}$, and are substituted by the terms in $\overline{t}$ in the first part of the formula. Similarly, the quantified variable x in an existential formula is contained in $\overline{y}$.

It is not hard to see that general frame logic formulas can be translated to FO-RD formulas that make use of these new inductively defined relations.

Proposition 2. *For every frame logic formula there is an equivalent FO-RD formula with the signature extended by auxiliary predicates for recursive definitions of frames.*

Data Structure/Predicate	Definition
$list(x)$ singly linked list	$ite(x = nil : \top, list(next(x)) \wedge x \notin Fr(list(next(x))))$
$dll(x)$ doubly linked list	$ite(x = nil : \top, ite(next(x) = nil : \top,$ $prev(next(x)) = x \wedge dll(next(x)) \wedge x \notin Fr(dll(next(x)))))$
$lseg(x, y)$ list segment	$ite(x = y : \top, lseg(next(x), y) \wedge x \notin Fr(lseg(next(x), y)))$
$length(x, n)$ length of list	$ite(x = nil : n = 0, length(next(x), n - 1))$
$sorted{-}list(x)$ sorted list	$ite(x = nil : \top, ite(next(x) = nil, \top,$ $key(x) \leq key(next(x)) \wedge sorted{-}list(next(x)) \wedge x \notin Fr(sorted{-}list(next(x))))))$
$mkeys(x, M)$ multiset of keys in linked list	$ite(x = nil : M = \emptyset, \exists M_1. \, M = M_1 \cup_m \{key(x)\} \wedge mkeys(next(x), M_1))$
$btree(x)$ binary tree	$ite(x = nil : \top, btree(left(x)) \wedge btree(right(x)) \wedge$ $x \notin Fr(btree(left(x))) \wedge x \notin Fr(btree(right(x))) \wedge$ $Fr(btree(left(x))) \cap Fr(btree(right(x))) = \emptyset)$
$bst(x)$ binary search tree	$ite(x = nil : \top, ite(left(x) = nil \wedge right(x) = nil : \top, ite(left(x) = nil :$ $key(x) \leq key(right(x)) \wedge bst(right(x)) \wedge x \notin Fr(bst(right(x))), ite(right(x) = nil :$ $key(left(x)) \leq key(x) \wedge bst(left(x)) \wedge x \notin Fr(bst(left(x)))))),$ $key(x) \leq key(right(x)) \wedge key(left(x)) \leq key(x) \wedge bst(left(x)) \wedge bst(right(x)) \wedge$ $x \notin Fr(bst(left(x))) \wedge x \notin Fr(bst(right(x))) \wedge$ $Fr(bst(left(x))) \cap Fr(bst(right(x))) = \emptyset)$
$height(x, n)$ height of binary tree	$ite(x = nil : n = 0, \exists n_1, n_2. \, height(left(x), n_1) \wedge height(right(x), n_2) \wedge$ $ite(n_1 > n_2 : n = n_1 + 1, n = n_2 + 1))$
$priorities(x, M)$ set of priorities in a binary tree	$ite(x = nil : M = \emptyset, \exists M_1, M_2. \, M = M_1 \cup M_2 \cup \{priority(x)\} \wedge$ $priorities(left(x), M_1) \wedge priorities(right(x), M_2))$
$treap(x)$ treap	$ite(x = nil : \top,$ $left(x) \neq nil \Rightarrow key(left(x)) < key(x) \wedge priority(left(x)) < priority(x) \wedge$ $right(x) \neq nil \Rightarrow key(right(x)) < key(x) \wedge priority(right(x)) < priority(x) \wedge$ $treap(left(x)) \wedge treap(right(x)) \wedge priorities(left(x)) \cap priorities(right(x)) = \emptyset \wedge$ $x \notin Fr(treap(left(x))) \cup Fr(treap(right(x))) \wedge$ $Fr(treap(left(x))) \cap Fr(treap(right(x))) = \emptyset)$
$balance{-}factor(x, b)$ balance factor (for AVL tree)	$ite(x = nil : 0, \exists n_1, n_2 : height(left(x), n_1) \wedge height(right(x), n_2) \wedge b = n_2 - n_1)$
$avl{-}tree(x)$	$ite(x = nil : \top,$ $avl{-}tree(left(x)) \wedge avl{-}tree(right(x)) \wedge balance{-}factor(x) \in \{-1, 0, 1\} \wedge$ $x \notin Fr(avl{-}tree(left(x))) \cup Fr(avl{-}tree(right(x))) \wedge$ $Fr(avl{-}tree(left(x))) \cap Fr(avl{-}tree(right(x))) = \emptyset)$
$ttree(x)$ threaded tree	$pttree(x, nil)$
$pttree(x)$ threaded tree aux	$ite(x = nil : \top, ((right(x) = nil \wedge tnext(x) = p) \vee$ $(right(x) \neq nil \wedge tnext(x) = right(x))) \wedge$ $pttree(left(x), x) \wedge pttree(right(x), p) \wedge$ $x \notin Fr(pttree(left(x), x)) \cup Fr(pttree(right(x), p)) \wedge$ $Fr(pttree(left(x), x)) \cap Fr(pttree(right(x), p)) = \emptyset)$

F. Expressing Data-structures Properties in FL

We now present the formulation of several data-structures and properties about them in FL. Table 1 depicts formulations of singly- and doubly-linked lists, list segments, lengths of lists, sorted lists, the multiset of keys stored in a list (assuming a background sort of multisets), binary trees, their heights, treaps, and AVL trees. In all these definitions, the frame operator plays a crucial role. We also present a formulation of *threaded trees*, which are binary trees where, apart from tree-edges, there is a pointer $tnext$ that connects every tree node to the inorder successor in the tree; these pointers go from leaves to ancestors arbitrarily far away in the tree, making it a nontrivial definition.

We believe that FL formulas naturally and succinctly express these data-structures and their properties, making it an attractive logic for annotating programs.

IV. PROGRAMS AND PROOFS

In this section, we develop a program logic for a while-programming language that can destructively update heaps. We assume that location variables are denoted by variables of the form x and y, whereas variables that denote other data (which would correspond to the *background* sorts in our logic) are denoted by v. We omit the grammar to construct background terms and formulas, and simply denote such 'background expres-

$$S ::= x := c \mid x := y \mid x := y.f \mid v := be$$
$$\mid x.f := y \mid \mathsf{alloc}(x) \mid \text{if } be \text{ then } S \text{ else } S$$
$$\mid \text{while } be \text{ do } S \mid S \,;\, S$$

Fig. 3. Grammar of while programs. Here c is a constant location and f is a field pointer. be and le are background and location expressions respectively. In our logics, we model every field f as a function $f()$ from locations to the appropriate sort.

sions' with be and clarify the sort when it is needed. The grammar for our programming language is in Figure 3.

Operational Semantics

A configuration $\mathcal{C}$ is of the form (M, H, U) where M contains interpretations for the store and the heap. The store is a partial map that interprets variables, constants, and non-mutable functions (a function from location variables to locations) and the heap is a total map on the domain of locations that interprets mutable functions (a function from pointers and locations to locations). H is a subset of locations denoting the set of allocated locations, and U is a subset of locations denoting *a subset* of unallocated locations that can be allocated in the future (the complement of $H \cup U$ is are deallocated and will never be allocated in the future). We introduce a special configuration $\perp$ that the program transitions to when it dereferences a variable not in H.

A configuration (M, H, U) is *valid* if all variables of the location sort map only to locations not in U, locations in H do not point to any location in U and U is a subset of the complement of H that does not contain nil or the locations mapped to by the variables. We denote this by $valid(M, H, U)$ Initial configurations and reachable configurations of any program will be valid.

The transition of configurations on various commands that manipulate the store and heap are defined in the natural way. Allocation adds a new location from U into A with pointer-fields defaulting to *nil* and default data fields. See the Appendix VIII-D for more details.

Triples and Validity

We express specifications of programs using triples of the form $\{\alpha\}S\{\beta\}$ where α and β are FL formulae and S is a program. The formulae are, however, restricted—we disallow atomic relations on locations, and functions with arity greater than one. We also disallow functions from a background sort to the foreground sort, and require guards in quantification to be of the form $f(z') = z$ or $z \in U$ (z is the quantified variable).

We define a triple to be *valid* if every valid configuration with heaplet being precisely the frame of α, when acted on by the program, yields a configuration with heaplet being the frame of β. More formally, a triple

is valid if for every valid configuration (M, H, U) such that $M \models \alpha$, $H = [\![Fr(\alpha)]\!]_M$:

- it is never the case that the abort state $\perp$ is encountered in the execution on S.
- if (M, H, U) transitions to (M', H', U') on S, then $M' \models \beta$ and $H' = [\![Fr(\beta)]\!]_{M'}$

A. Program Logic

First, we define a set of *local rules* and rules for conditionals, while, sequence, consequence, and framing:

Assignment: $\{true\} \; x := y \; \{x = y\}$
$\qquad\qquad\quad \{true\} \; x := c \; \{x = c\}$

Lookup: $\{f(y) = f(y)\} \; x := y.f \; \{x = f(y)\}$

Mutation: $\{f(x) = f(x)\} \; x.f := y \; \{f(x) = y\}$

Allocation: $\{true\} \; \mathsf{alloc}(x) \; \{\bigwedge_{f \in F} f(x) = def_f\}$

Deallocation: $\{f(x) = f(x)\} \; \mathsf{free}(x) \; \{true\}$

Conditional rule:
$$\frac{\{be \wedge \alpha\} \; S \; \{\beta\} \quad \{\neg be \wedge \alpha\} \; T \; \{\beta\}}{\{\alpha\} \text{ if } be \text{ then } S \text{ else } T \; \{\beta\}}$$

While rule:
$$\frac{\{\alpha \wedge be\} \; S \; \{\alpha\}}{\{\alpha\} \text{ while } be \text{ do } S \; \{\neg be \wedge \alpha\}}$$

Sequence rule:
$$\frac{\{\alpha\} \; S \; \{\beta\} \quad \{\beta\} \; T \; \{\gamma\}}{\{\alpha\} \; S \,;\, T \; \{\gamma\}}$$

Consequence rule:
$$\frac{\begin{array}{cc} \alpha' \implies \alpha & \\ \beta \implies \beta' & \end{array} \{\alpha\} \; S \; \{\beta\} \quad \begin{array}{c} Fr(\alpha) = Fr(\alpha') \\ Fr(\beta) = Fr(\beta') \end{array}}{\{\alpha'\} \; S \; \{\beta'\}}$$

Frame rule:
$$\frac{Fr(\alpha) \cap Fr(\gamma) = \emptyset \quad \{\alpha\} \; S \; \{\beta\}}{\{\alpha \wedge \gamma\} \; S \; \{\beta \wedge \gamma\}} \; vars(S) \cap fv(\gamma) = \emptyset$$

The above rules are intuitively clear and are similar to the local rules in separation logic [35]. The rules for statements capture their semantics using minimal/tight heaplets, and the frame rule allows proving triples with larger heaplets. The soundness of the frame rule relies crucially on the frame theorem for FL (Theorem 1). In the rule for alloc above, the postcondition says that the newly allocated location has default values for all pointer fields and datafields (denoted as def_f).

Theorem 2. *The above rules are sound with respect to the operational semantics.*

Proof. See Appendix VIII-F. Local rules are derived from global rules (see next section). $\qquad\square$

Weakest-precondition proof rules

We now turn to the much more complex problem of designing rules that give weakest preconditions for

arbitrary postconditions, for loop-free programs. In separation logic, such rules resort to using the magic-wand operator $-\ast$ [12], [25], [26], [35], The magic-wand operator, a complex operator whose semantics calls for *second-order quantification* over arbtitrarily large submodels. In our setting, our main goal is to show that FL is itself capable of expressing weakest preconditions of postconditions written in FL.

First, we define a notion of *Weakest Tightest Precondition* (WTP) of a formula β with respect to each command in our operational semantics. To define this notion, we first define the notion of a preconfiguration, and use that definition to define weakest tightest preconditions:

Definition 1. *The preconfigurations corresponding to a configuration (M, H, U) with respect to a program S are a set of valid configurations of the form (M_p, H_p, U_p) (with M_p being a model, H_p and U_p a subuniverse of the locations in M_p, and U_p being unallocated locations) such that when S is executed on M_p with unallocated set U_p it dereferences only locations in H_p and results (using the operational semantics rules) in (M, H, U) or gets stuck. That is:*

$$preconfigurations((M, H, U), S) =$$
$$\{(M_p, H_p, U_p) \mid valid(M_p, H_p, U_p) \ and$$
$$(M_p, H_p, U_p) \overset{S}{\Rightarrow} (M, H, U) \ or$$
$$(M_p, H_p, U_p) \ gets \ stuck \ on \ S\}$$

Definition 2. *The WTP of a formula β with respect to a program S, is a formula α such that*

$$\{(M_p, H_p, U_p) \mid M_p \models \alpha, H_p = [\![Fr(\alpha)]\!]_{M_p},$$
$$valid(M_p, H_p, U_p)\}$$
$$= \{preconfigurations((M, H, U), S) \mid M \models \beta,$$
$$H = [\![Fr(\beta)]\!]_M, valid(M, H, U)\}$$

With the notion of weakest tightest preconditions, we define global program logic rules for each command of our language. In contrast to local rules, global specifications contain heaplets that may be larger than the smallest heap on which one can execute the command.

Intuitively, the WTP of β for lookup states that β must hold in the precondition when x is interpreted as x', where $x' = f(y)$, and further that the location y must belong to the frame of β. The rules for mutation and allocation are more complex. For mutation, we define a transformation $MW^{x.f:=y}(\beta)$ that evaluates a formula β in the pre-state as though it were evaluated in the post-state. We similarly define such a transformation $MW_v^{\mathsf{alloc}(x)}$ for allocation. We will define these in detail later. Finally, the deallocation rule ensures x is not in the frame of the postcondition. The conjunct $f(x) = f(x)$ is provided to satisfy the tightness condition, ensuring

the frame of the precondition is the frame of the post-condition with x added. The rules can be seen below.

Assignment-G:
$$\{\beta[y/x]\} \ x := y \ \{\beta\}, \quad \{\beta[c/x]\} \ x := c \ \{\beta\}$$

Lookup-G:
$$\{\exists x' : \ x' = f(y). \ (\beta \wedge y \in Fr(\beta))[x'/x]\}$$
$$\quad x := y.f$$
$$\{\beta\}$$
where x' does not occur in β

Mutation-G:
$$\{MW^{x.f:=y}(\beta \wedge x \in Fr(\beta))\} \ x.f := y \ \{\beta\}$$

Allocation-G:
$$\{\forall v : (v \in U) . (v \neq nil \Rightarrow MW_v^{\mathsf{alloc}(x)}(\beta))\}$$
$$\quad \mathsf{alloc}(x)$$
$$\{\beta\}$$
for some fresh variable v

Deallocation-G:
$$\{\beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x)\} \ \mathsf{free}(x) \ \{\beta\}$$
where $f \in F_{\mathsf{m}}$ is an arbitrary (unary) mutable function.

Definitions of MW primitives[3]

Recall that the formulas $MW^{x.f:=y}$ and $MW_v^{\mathsf{alloc}(x)}$ need to evaluate a formula β in the pre-state as it would evaluate in the post-state after mutation and allocation statements. The definition of $MW^{x.f:=y}$ is as follows:

$$MW^{x.f:=y}(\beta) =$$
$$\beta[\lambda z. \ ite(z = x : ite(f(x) = f(x) : y, y), f(z))/f]$$

The $\beta[\lambda z.\rho(z)/f]$ notation is shorthand for saying that each occurrence of a term of the form $f(t)$, where t is a term, is substituted (recursively, from inside out) by the term $\rho(t)$. The precondition essentially evaluates β taking into account f's transformation, but we use the *ite* expression with a tautological guard $f(x) = f(x)$ (which has the frame containing the singleton x) in order to preserve the frame (see Appendix VIII-F: Lemma 2).

The definition of $MW_v^{\mathsf{alloc}(x)}$ is much more complex and uses the set of unallocated nodes U:

$$MW_v^{\mathsf{alloc}(x)}(\beta) =$$
$$\beta[v/x][\lambda z. \ ite(z = v : def_f, f(z))/f]_{f \in F}[U \setminus \{v\}/U]$$

This transformation removes x from the frame of the formula (note $MW^{x.f:=y}$ preserves the frame).

Theorem 3. *The rules above suffixed with **-G** are sound w.r.t the operational semantics. And, each precondition corresponds to the weakest tightest precondition of β.*

Proof. See Appendix VIII-F. □

---

[3]The acronym MW is a shout-out to the Magic-Wand operator, as these serve a similar function, except that they are in FL itself.

V. Expressing a Precise Separation Logic

In this section, we show the FL is expressive by capturing a fragment of separation logic in frame logic; the fragment is a syntactic fragment of separation logic that defines only *precise formulas*— formulas that can be satisfied in at most one heaplet for any store. The translation also shows that frame logic can naturally and compactly capture such separation logic formulas.

A. A Precise Separation Logic

As discussed in Section I, a crucial difference between separation logic and frame logic is that formulas in separation logic have uniquely determined frames/heaplets, while this is not true in separation logic. However, it is well known that in verification, determined heaplets are very natural (most uses of separation logic in fact are precise) and sometimes desirable. For instance, see [8] where precision is used crucially to give sound semantics to concurrent separation logic and [27] where precise formulas are proposed in verifying modular programs as imprecision causes ambiguity in function contracts.

We define a fragment of separation logic that defines precise formulas (more accurately, we handle a slightly larger class inductively: formulas that when satisfiable have unique minimal heaplets for any given store). The fragment we capture is similar to the notion of precise predicates seen in [27]:

Definition 3. *PSL Fragment:*
- *sf: formulas over the stack only (nothing dereferenced). Includes isatom?(), $m(x) = y$ for immutable m, true, background formulas, etc.*
- $x \xrightarrow{f} y$
- $ite(sf, \varphi_1, \varphi_2)$ *where sf is from the first bullet*
- $\varphi_1 \wedge \varphi_2$
- $\varphi_1 * \varphi_2$
- $\mathcal{I}$ *where $\mathcal{I}$ contains all unary inductive definitions I that have unique heaplets inductively (list, tree, etc.). In particular, the body ρ_I of I is a formula in the PSL fragment ($\rho_I[I \leftarrow \varphi]$ is in the PSL fragment provided φ is in the PSL fragment). Additionally, for all x, if $s, h \models I(x)$ and $s, h' \models I(x)$, then $h = h'$.[4]*
- $\exists y. (x \xrightarrow{f} y) * \varphi_1$

Note that in the fragment negation and disjunction are disallowed, but mutually exclusive disjunction using *ite* is allowed. Existential quantification is only present when the topmost operator is a $*$ and where one of the formulas guards the quantified variable uniquely.

The semantics of this fragment follows the standard semantics of separation logic [12], [25], [26], [35], with

[4]While we only assume unary inductive definitions here, we can easily generalize this to inductive definitions with multiple parameters.

the heaplet of $x \xrightarrow{f} y$ taken to be $\{x\}$. See Remark 1 in Section III-B for a discussion of a more accurate heaplet for $x \xrightarrow{f} y$ being the set containing the pair (x, f), and how this can be modeled in the above semantics by using field-lookups using non-mutable pointers.

Theorem 4 (Minimum Heap). *For any formula φ in the PSL fragment, if there is an s and h such that $s, h \models \varphi$ then there is a h_φ such that $s, h_\varphi \models \varphi$ and for all h' such that $s, h' \models \varphi$, $h_\varphi \subseteq h'$.*

Proof. See Appendix VIII-G $\qquad \square$

B. Translation to Frame Logic

For a separation logic store and heap s, h (respectively), we define the corresponding interpretation $\mathcal{M}_{s,h}$ such that variables are interpreted according to s and values of pointer functions on $dom(h)$ are interpreted according to h. For φ in the PSL fragment, we first define a formula $P(\varphi)$, inductively, that captures whether φ is precise. φ is a precise formula iff, when it is satisfiable with a store s, there is exactly one h such that $s, h \models \varphi$. The formula $P(\varphi)$ is in separation logic and will be used in the translation.

Definition 4. *Precision predicate P:*
- $P(sf) = \bot$
- $P(x \xrightarrow{f} y) = \top$
- $P(ite(sf, \varphi_1, \varphi_2)) = (sf \wedge P(\varphi_1)) \vee (\neg sf \wedge P(\varphi_2))$
- $P(\varphi_1 \wedge \varphi_2) = P(\varphi_1) \vee P(\varphi_2)$
- $P(\varphi_1 * \varphi_2) = P(\varphi_1) \wedge P(\varphi_2)$
- $P(I) = \top$ *where $I \in \mathcal{I}$ is an inductive predicate*
- $P(\exists y. (x \xrightarrow{f} y) * \varphi_1) = P(\varphi_1)$

Note that this definition captures precision within our fragment since stack formulae are imprecise and pointer formulae are precise. The argument for the rest of the cases follow by simple structural induction.

Now we define the translation T inductively:

Definition 5. *Translation from PSL to Frame Logic:*
- $T(sf) = sf$
- $T(x \xrightarrow{f} y) = (f(x) = y)$
- $ite(sf, \varphi_1, \varphi_2) = ite(T(sf), T(\varphi_1), T(\varphi_2))$
- $T(\varphi_1 \wedge \varphi_2) =$

$$T(\varphi_1) \wedge T(\varphi_2)$$
$$\wedge\, T(P(\varphi_1)) \implies Fr(T(\varphi_2)) \subseteq Fr(T(\varphi_1))$$
$$\wedge\, T(P(\varphi_2)) \implies Fr(T(\varphi_1)) \subseteq Fr(T(\varphi_2))$$

- $T(\varphi_1 * \varphi_2) = T(\varphi_1) \wedge T(\varphi_2) \wedge Fr(T(\varphi_1)) \cap Fr(T(\varphi_2)) = \emptyset$
- $T(I) = T(\rho_I)$ *where ρ_I is the definition of the inductive predicate I as in Section 3.*
- $T(\exists y. (x \xrightarrow{f} y) * \varphi_1) = \exists y : [f(x) = y]. [T(\varphi_1) \wedge x \notin Fr(T(\varphi_1))]$

11

Recall that any formula φ in the PSL fragment has a unique minimal heap (Theorem 4). With this, we have the following theorem, which captures the correctness of the translation:

Theorem 5. *For any formula φ in the PSL fragment, we have the following implications:*

$$s, h \models \varphi \implies \mathcal{M}_{s,h} \models T(\varphi)$$
$$\mathcal{M}_{s,h} \models T(\varphi) \implies s, h' \models \varphi$$
$$where\ h' \equiv \mathcal{M}_{s,h}(Fr(T(\varphi)))$$

Here, $\mathcal{M}_{s,h}(Fr(T(\varphi)))$ is the interpretation of $Fr(T(\varphi))$ in the model $\mathcal{M}_{s,h}$. Note h' is minimal and is equal to h_φ as in Theorem 4.

Proof. See Appendix VIII-G. ☐

VI. Related Work

The frame problem [13] is an important problem in many different domains of research. In the broadest form, it concerns representing and reasoning about the effects of a local action without requiring explicit reasoning regarding static changes to the global scope. For example, in artificial intelligence one wants a logic that can seamlessly state that if a door is opened in a lit room, the lights continue to stay switched on. This issue is present in the domain of verification as well, specifically with heap-manipulating programs.

There are many solutions that have been proposed to this problem. The most prominent proposal in the verification context is separation logic [12], [25], [26], [35], which introduces special symbols $*$ and $-*$ (magic wand) representing separating conjunction and separating implication, with a tight frame semantics for each statement; formulas hence implicitly define frames.

In contrast to separation logic, the work on Dynamic Frames [18], [17] and similarly inspired approaches such as Region Logic [4], [2], [3] allow methods to explicitly specify the portion of the frame that may be modified. This allows finer grained control over the modifiable section and avoids special symbols like $*$ and $-*$. However, explicitly writing out frame annotations can become verbose and tedious. The work on Implicit Dynamic Frames [36] is similar to ours in attempts to counteract this and allows the syntactic inference of an 'access set' that aids frame reasoning; however, it resorts to special constructs like $*$. Our work, in contrast, gives access to the frame of formulas using a special construct, and the user to reason with these frames using set theory.

Another distinction involves the discrepancy between non-unique heaplets in separation logic and unique heaplets in our work. The use of determined heaplets has been seen in [34], [29], [27] as it can be more amenable to automated reasoning, and in particular a subset of separation logic with determined heaplets known as

precise predicates is captured in [27], which we also capture in Section V.

There is also a rich literature on reasoning with these logics for programs. Decidability is an important dimension and there is a lot of work on decidable logics for heaps with separation logic specifications [7], [11], [24], [30], [5], [6]. The work based on EPR (Effectively Propositional Reasoning) for specifying heap properties [14], [15], [16] provides decidability, as does some of the work that translates separation logic specifications into classical logic [31].

Translating separation logic into classical logics and reasoning with them is a solution pursued in a lot of recent efforts [10], [29], [32], [31], [33]. Work on natural proofs [22], [29], [34], [37] convert the special operators $*$ and $-*$ to first-order logic or first-order logic variants. Techniques such as [21] include foundations for natural proofs with an emphasis on reasoning about recursive definitions. These techniques perform sound but incomplete reasoning, but not decidable procedures.

Other techniques including recent work on cyclic proofs [9], [38] use heuristics for reasoning about recursive definitions. We believe the above tools and techniques can be adapted in the future to the Frame Logic introduced in this paper.

VII. Conclusions

Our main contribution is to show that classical first-order logic can be endowed with frame reasoning using a logical construct that recovers the implicit frames of formulas, and to develop a program logic based on it. The program logic supports local heap reasoning, frame reasoning, supports weakest tightest preconditions across loop-free programs, and we have argued its efficacy by expressing properties of data-structures naturally and succinctly, and showing that it can express a precise fragment of separation logic. The above, combined with the fact that it has a classical semantics and can be rewritten into first-order logic with recursive definitions, makes it an attractive logic for program verification.

Our results show that when loop invariants can be expressed in Frame Logic, the weakest precondition rules can be used, automatically, to reduce verification to checking validity of frame logic formulas. These can then be reduced to pure first-order with recursive definition reasoning, which can be effected using interactive theorem provers like Coq [23] or by automated first-order mechanisms [22], [29], [34], [37], [19]. A practical realization of this in a tool for verifying programs in a standard programming language is the most compelling future work. Given that our logic is a first-order logic, we believe that it could be a more tractable setting to find decidable fragments. Finally, an extension of our work to *higher order logics* is also intriguing.

REFERENCES

[1] Krzysztof R. Apt. Ten years of hoare's logic: A survey—part i. *ACM Trans. Program. Lang. Syst.*, 3(4):431–483, October 1981.

[2] Anindya Banerjee and David Naumann. Local reasoning for global invariants, part ii: Dynamic boundaries. 60, 06 2013.

[3] Anindya Banerjee, David A. Naumann, and Stan Rosenberg. Regional logic for local reasoning about global invariants. In Jan Vitek, editor, *ECOOP 2008 – Object-Oriented Programming*, pages 387–411, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[4] Anindya Banerjee, David A. Naumann, and Stan Rosenberg. Local reasoning for global invariants, part i: Region logic. *J. ACM*, 60(3):18:1–18:56, June 2013.

[5] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *Proceedings of the 4th International Conference on Formal Methods for Components and Objects*, FMCO'05, pages 115–137, Berlin, Heidelberg, 2006. Springer-Verlag.

[6] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. A decidable fragment of separation logic. In *Proceedings of the 24th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'04, pages 97–109, 2004.

[7] Josh Berdine, Cristiano Calcagno, and Peter W. O'Hearn. Symbolic execution with separation logic. In *Proceedings of the Third Asian Conference on Programming Languages and Systems*, APLAS'05, pages 52–68, 2005.

[8] Stephen Brookes. A semantics for concurrent separation logic. *Theor. Comput. Sci.*, 375(1-3):227–270, April 2007.

[9] James Brotherston, Dino Distefano, and Rasmus Lerchedahl Petersen. Automated cyclic entailment proofs in separation logic. In *Proceedings of the 23rd International Conference on Automated Deduction*, CADE'11, pages 131–146, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] W. N. Chin, C. David, H. H. Nguyen, and S. Qin. Automated verification of shape, size and bag properties. In *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, pages 307–320, 2007.

[11] Byron Cook, Christoph Haase, Joël Ouaknine, Matthew Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *Proceedings of the 22Nd International Conference on Concurrency Theory*, CONCUR'11, pages 235–249, 2011.

[12] Stéphane Demri and Morgan Deters. Separation logics and modalities: a survey. *Journal of Applied Non-Classical Logics*, 25:50–99, 2015.

[13] Patrick J. Hayes. The frame problem and related problems in artificial intelligence. In Bonnie Lynn Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 223 – 230. Morgan Kaufmann, 1981.

[14] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Ori Lahav, Aleksandar Nanevski, and Mooly Sagiv. Modular reasoning about heap paths via effectively propositional formulas. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 385–396, New York, NY, USA, 2014. ACM.

[15] Shachar Itzhaky, Anindya Banerjee, Neil Immerman, Aleksandar Nanevski, and Mooly Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 756–772, Berlin, Heidelberg, 2013. Springer-Verlag.

[16] Shachar Itzhaky, Nikolaj BjØrner, Thomas Reps, Mooly Sagiv, and Aditya Thakur. Property-directed shape analysis. In *Proceedings of the 16th International Conference on Computer Aided Verification*, CAV'14, pages 35–51, Berlin, Heidelberg, 2014. Springer-Verlag.

[17] I. T. Kassios. The dynamic frames theory. *Form. Asp. Comput.*, 23(3):267–288, May 2011.

[18] Ioannis T. Kassios. Dynamic frames: Support for framing, dependencies and sharing without restrictions. In Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, editors, *FM 2006: Formal Methods*, pages 268–283, Berlin, Heidelberg, 2006. Springer-Verlag.

[19] Laura Kovács, Simon Robillard, and Andrei Voronkov. Coming to terms with quantified reasoning. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL '17, pages 260–270, New York, NY, USA, 2017. ACM.

[20] Laura Kovács and Andrei Voronkov. First-order theorem proving and vampire. In *CAV '13*, pages 1–35, 2013.

[21] Christof Löding, P. Madhusudan, and Lucas Peña. Foundations for natural proofs and quantifier instantiation. *PACMPL*, 2(POPL):10:1–10:30, 2018.

[22] P. Madhusudan, Xiaokang Qiu, and Andrei Stefanescu. Recursive proofs for inductive tree data-structures. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 123–136, New York, NY, USA, 2012. ACM.

[23] The Coq development team. The coq proof assistant reference manual, 2018. Version 8.8.2.

[24] Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic + superposition calculus = heap theorem prover. In *Proceedings of the 32Nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '11, pages 556–566, New York, NY, USA, 2011. ACM.

[25] Peter W. O'Hearn. A primer on separation logic (and automatic program verification and analysis). In *Software Safety and Security*, 2012.

[26] Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, CSL '01, pages 1–19, London, UK, UK, 2001. Springer-Verlag.

[27] Peter W. O'Hearn, Hongseok Yang, and John C. Reynolds. Separation and information hiding. In *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '04, pages 268–280, New York, NY, USA, 2004. ACM.

[28] Matthew Parkinson and Gavin Bierman. Separation logic and abstraction. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '05, pages 247–258, New York, NY, USA, 2005. ACM.

[29] Edgar Pek, Xiaokang Qiu, and P. Madhusudan. Natural proofs for data structure manipulation in c using separation logic. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, pages 440–451, New York, NY, USA, 2014. ACM.

[30] Juan Antonio Navarro Pérez and Andrey Rybalchenko. Separation logic modulo theories. In *Programming Languages and Systems (APLAS)*, pages 90–106, Cham, 2013. Springer International Publishing.

[31] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using smt. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV'13, pages 773–789, Berlin, Heidelberg, 2013. Springer-Verlag.

[32] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic with trees and data. In *Proceedings of the 16th International Conference on Computer Aided Verification*, CAV'14, pages 711–728, Berlin, Heidelberg, 2014. Springer-Verlag.

[33] Ruzica Piskac, Thomas Wies, and Damien Zufferey. Grasshopper. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 124–139, 2014.

[34] Xiaokang Qiu, Pranav Garg, Andrei Ştefănescu, and P. Madhusudan. Natural proofs for structure, data, and separation. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 231–242, New York, NY, USA, 2013. ACM.

[35] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, pages 55–74, 2002.

[36] Jan Smans, Bart Jacobs, and Frank Piessens. Implicit dynamic frames. *ACM Trans. Program. Lang. Syst.*, 34(1):2:1–2:58, May 2012.

[37] Philippe Suter, Mirco Dotta, and Viktor Kuncak. Decision procedures for algebraic data types with abstractions. In *Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '10, pages 199–210, New York, NY, USA, 2010. ACM.

[38] Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. Automated mutual explicit induction proof in separation logic. In *FM*, 2016.

VIII. APPENDIX

A. Frame Models

As explained in Section III-C, a frame model is a model in which the inductively defined relations and the frame expressions are interpreted by the simultaneous least solution of the corresponding equations.

In order to make the definition of frame models precise, we need a bit of terminology.

A *pre-model* $\hat{M}$ is defined like a model with the difference that a pre-model does not interpret the inductive relation symbols and the frame expressions $Fr(\varphi)$ and $Fr(t)$. A pre-model $\hat{M}$ spans a class of models $\mathsf{Mod}(\hat{M})$, namely those that simply extend $\hat{M}$ by an interpretation of the inductive relations and the frame expressions.

The inductive definitions of relations from $\mathcal{I}$ can have negative references to frame expressions. For example, the tree definition from Example 1 uses frame expressions in the subformula $Fr(tree(\ell(x))) \cap Fr(tree(r(x))) = \emptyset$. This formula is true if there does not exist an element in the intersection $Fr(tree(\ell(x))) \cap Fr(tree(r(x)))$, and hence negatively refers to these frame expressions. For this reason, we need to define two partial orders that correspond to first taking the least fixpoint for the frame expression, and then the least fixpoint for the inductive predicates.

In the following, we refer to the equations for the frame expressions from Figure 2 as *frame equations*, and to the equations $[\![R(\overline{x})]\!]_M = [\![\rho_R(\overline{x})]\!]_M$ for the inductive definitions as the *inductive equations*.

For $M_1, M_2 \in \mathsf{Mod}(\hat{M})$ we let $M_1 \leq_{\mathrm{f}} M_2$ if

- $[\![Fr(\varphi)]\!]_{M_1}(\nu) \subseteq [\![Fr(\varphi)]\!]_{M_2}(\nu)$ as well as $[\![Fr(t)]\!]_{M_1}(\nu) \subseteq [\![Fr(t)]\!]_{M_2}(\nu)$ for all frame expressions and all variable assignments ν.

Note that $\leq_{\mathrm{f}}$ is not a partial order but only a preorder: for two models M_1, M_2 that differ only in their interpretations of the inductive relations, we have $M_1 \leq_{\mathrm{f}} M_2$ and $M_2 \leq_{\mathrm{f}} M_1$. We write $M_1 <_{\mathrm{f}} M_2$ if $M_1 \leq_{\mathrm{f}} M_2$ and not $M_2 \leq_{\mathrm{f}} M_1$.

We further define $M_1 \leq_{\mathrm{i}} M_2$ if

- $[\![Fr(\varphi)]\!]_{M_1} = [\![Fr(\varphi)]\!]_{M_2}$ as well as $[\![Fr(t)]\!]_{M_1} = [\![Fr(t)]\!]_{M_2}$ for all frame expressions, and

- $[\![I]\!]_{M_1} \subseteq [\![I]\!]_{M_2}$ for all inductive relations $I \in \mathcal{I}$.

The relation $\leq_{\mathrm{i}}$ is a partial order.

We say that $M \in \mathsf{Mod}(\hat{M})$ is a *frame model* if its interpretation function $[\![\cdot]\!]_M$ satisfies the inductive equations and the frame equations, and furthermore

1) each $M' \in \mathsf{Mod}(\hat{M})$ with $M' <_{\mathrm{f}} M$ does not satisfy the frame equations, and
2) each $M' \in \mathsf{Mod}(\hat{M})$ with $M' <_{\mathrm{i}} M$ does not satisfy the inductive equations.

For proving the existence of a unique frame model, we use the following lemma for dealing with guards and terms with mutable functions.

Lemma 1. *Let $\hat{M}$ be a pre-model, $M_1, M_2 \in \mathsf{Mod}(\hat{M})$, and ν be a variable assignment.*

1) *If φ is formula that does not use inductive relations and frame expressions, then $M_1, \nu \models \varphi$ iff $M_2, \nu \models \varphi$.*
2) *If t is a term that has no frame expressions as subterms, then $[\![t]\!]_{M_1,\nu} = [\![t]\!]_{M_2,\nu}$.*
3) *If $t = f(t_1, \ldots, t_n)$ is a term with a mutable function symbol $f \in F_{\mathsf{m}}$, then $[\![t_i]\!]_{M_1,\nu} = [\![t_i]\!]_{M_2,\nu}$ for all i.*

Proof. Parts 1 and 2 are immediate from the fact that M_1 and M_2 only differ in the interpretation of the inductive relations and frame expressions. For the third claim, note that we assumed that the only functions involving arguments of sort $\sigma_{\mathsf{S}(\mathsf{f})}$ are the standard functions for set manipulation. Hence, a term build from a mutable function symbol cannot have frame expressions as subterms. Therefore, the third claim follows from the second one. □

The following proposition is the formalization of Proposition 1 in Section III-C.

Proposition 3. *For each pre-model $\hat{M}$, there is a unique frame model in $\mathsf{Mod}(\hat{M})$.*

Proof. The frame equations define an operator μ_{f} on $\mathsf{Mod}(\hat{M})$. This operator μ_{f} is defined in a standard way, as explained in the following. Let $M \in \mathsf{Mod}(\hat{M})$. Then $\mu_{\mathrm{f}}(M)$ is a model in $\mathsf{Mod}(\hat{M})$ where $[\![Fr(\varphi)]\!]_{\mu_{\mathrm{f}}(M)}$, resp. $[\![Fr(t)]\!]_{\mu_{\mathrm{f}}(M)}$, is obtained by taking the right-hand side of the corresponding equation. For example, $[\![Fr(\varphi_1 \wedge \varphi_2)]\!]_{\mu_{\mathrm{f}}(M)}(\nu) = [\![Fr(\varphi_1)]\!]_M(\nu) \cup [\![Fr(\varphi_2)]\!]_M(\nu)$. The interpretation of the inductive predicates is left unchanged by μ_{f}.

We can show that μ_{f} is a monotonic operator on $(\mathsf{Mod}(\hat{M}), \leq_{\mathrm{f}})$, that is, for all $M_1, M_2 \in \mathsf{Mod}(\hat{M})$ with $M_1 \leq_{\mathrm{f}} M_2$ we have hat $\mu_{\mathrm{f}}(M_1) \leq \mu_{\mathrm{f}}(M_2)$. It is routine to check monotonicity of μ_{f} by induction on the structure of the frame expressions. We use Lemma 1 for the only cases in which the semantics of formulas

and terms is used in the frame equations, namely *ite*-formulas, existential formulas, and terms $f(t_1, \ldots, t_n)$ with mutable function f. Consider, for example, the frame equation

$$\llbracket Fr(f(t_1, \ldots, t_n)) \rrbracket_M(\nu)$$
$$= \bigcup_{i \text{ with } t_i \text{ of sort } \sigma_{\mathsf{f}}} \{\llbracket t_i \rrbracket_{M,\nu}\} \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_M(\nu)$$

for $f \in F_{\mathsf{m}}$, and let $M_1 \leq_{\mathsf{f}} M_2$ be in $\mathsf{Mod}(\hat{M})$ and ν be a variable assignment. Then

$$\llbracket Fr(f(t_1, \ldots, t_n)) \rrbracket_{\mu_{\mathsf{f}}(M_1)}(\nu)$$
$$= \bigcup_{i \text{ with } t_i \text{ of sort } \sigma_{\mathsf{f}}} \{\llbracket t_i \rrbracket_{M_1,\nu}\} \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_{M_1}(\nu)$$
$$\overset{(1)}{=} \bigcup_{i \text{ with } t_i \text{ of sort } \sigma_{\mathsf{f}}} \{\llbracket t_i \rrbracket_{M_2,\nu}\} \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_{M_1}(\nu)$$
$$\overset{(2)}{\subseteq} \bigcup_{i \text{ with } t_i \text{ of sort } \sigma_{\mathsf{f}}} \{\llbracket t_i \rrbracket_{M_2,\nu}\} \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_{M_2}(\nu)$$
$$= \llbracket Fr(f(t_1, \ldots, t_n)) \rrbracket_{\mu_{\mathsf{f}}(M_2)}(\nu)$$

where (1) holds because of Lemma 1, and (2) holds because $M_1 \leq_{\mathsf{f}} M_2$.

As a further case, consider the frame equation for $R(\overline{t})$ where R is an inductively defined relation and $t = (t_1, \ldots, t_n)$.

$$\llbracket Fr(R(\overline{t})) \rrbracket_{\mu_{\mathsf{f}}(M_1)}(\nu)$$
$$= \llbracket Fr(\rho_R(\overline{x}) \rrbracket_{M_1}(\nu[\overline{x} \leftarrow \llbracket \overline{t} \rrbracket_{M_1,\nu}]) \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_{M_1}(\nu)$$
$$\overset{(*)}{\subseteq} \llbracket Fr(\rho_R(\overline{x}) \rrbracket_{M_2}(\nu[\overline{x} \leftarrow \llbracket \overline{t} \rrbracket_{M_2,\nu}]) \cup \bigcup_{i=1}^{n} \llbracket Fr(t_i) \rrbracket_{M_2}(\nu)$$
$$= \llbracket Fr(R(\overline{t})) \rrbracket_{\mu_{\mathsf{f}}(M_2)}(\nu)$$

For the inclusion $(*)$ we use the fact that the t_i do not contain frame expressions as subterms by our restriction of the type of inductively defined relations. Hence, by Lemma 1, $\llbracket t_i \rrbracket_{M_1,\nu} = \llbracket t_i \rrbracket_{M_2,\nu}$.

Similarly, one can show the inclusion for the other frame equations.

We also obtain an operator μ_{i} from the inductive equations, which leaves the interpretation of the frame expressions unchanged. The operator μ_{i} is monotonic on $(\mathsf{Mod}(\hat{M}), \leq_{\mathsf{i}})$ because inductive predicates can only be used positively in the inductive definitions, and furthermore $\leq_{\mathsf{i}}$ only compares models with the same interpretation of the frame expressions.

In order to obtain the unique frame model, we first consider the subset of $\mathsf{Mod}(\hat{M})$ in which all inductive predicates are interpreted as empty set. On this set of models, $\leq_{\mathsf{f}}$ is a partial order and forms a complete lattice (the join and meet for the lattice are obtained by taking the pointwise union, respectively intersection, of the interpretations of the frame expression). By the Knaster-Tarski theorem, there is a unique least fixpoint of μ_{f}. This fixpoint can be obtained by iterating μ_{f} starting from the model in $\mathsf{Mod}(\hat{M})$ that interprets all inductive

relations and the frame expression by the empty set (in general, this iteration is over the ordinal numbers, not just the natural numbers). Let M_f be this least fixpoint.

The subset of $\mathsf{Mod}(\hat{M})$ in which the frame expressions are interpreted as in M_f forms a complete lattice with the partial order $\leq_{\mathsf{i}}$. Again by the Knaster-Tarski Theorem, there is a unique least fixpoint. This least fixpoint can be obtained by iterating the operator μ_{i} starting from M_f (again, the iteration is over the ordinals).

Denote the resulting model by $M_{f,i}$. It interprets the frame expressions in the same way as M_f, and thus $M_f \leq_{\mathsf{f}} M_{f,i}$ and $M_{f,i} \leq_{\mathsf{i}} M_i$. By monotonicity of μ_{f}, $M_{f,i}$ is also a fixpoint of μ_{f} and thus satisfies the frame equations. Hence $M_{f,i}$ satisfies the inductive equations and the frame equations. It can easily be checked that $M_{f,i}$ also satisfies the other conditions of a frame model: Let $M \in \mathsf{Mod}(\hat{M}$ with $M <_{\mathsf{f}} M_{f,i}$. Then also $M \leq_{\mathsf{f}} M_f$ and assuming that M satisfies the frame equations yields a smaller fixpoint of μ_{f}, and thus a contradiction. Similarly, a model $M <_{\mathsf{i}} M_{f,i}$ cannot satisfy the inductive equations.

It follows that $M_{f,i}$ is a frame model in $\mathsf{Mod}(\hat{M})$. Uniqueness follows from the uniqueness of the least fixpoints of μ_{f} and μ_{i} as used in the construction of $M_{f,i}$. □

B. Frame Theorem Proof

Theorem 1 (Frame Theorem). *Let M, M' be frame models such that M' is a mutation of M that is stable on $X \subseteq U_{\sigma_{\mathsf{f}}}$, and let ν be a variable assignment. Then $M, \nu \models \alpha$ iff $M', \nu \models \alpha$ for all formulas α with $\llbracket Fr(\alpha) \rrbracket_M(\nu) \subseteq X$, and $\llbracket t \rrbracket_{M,\nu} = \llbracket t \rrbracket_{M',\nu}$ for all terms t with $\llbracket Fr(t) \rrbracket_M(\nu) \subseteq X$.*

Proof. The intuition behind the statement of the theorem should be clear. The frame of a formula/term contains the elements on which mutable functions are dereferenced in order to evaluate the formula/term. If the mutable functions do not change on this set, then the evaluation does not change.

For a formal proof of the Frame Theorem, we refer to the terminology and definitions introduced in Appendix VIII-A, and to the proof of Proposition 3 in Appendix VIII-A, in which the unique frame model is obtained by iterating the operators μ_{f} and μ_{i}, which are defined by the frame equations and the inductive equations.

In general, this iteration of the operators ranges over ordinals (not just natural numbers). For an ordinal η, let M_η and M'_η be the models at step η of the fixpoint iteration for obtaining the frame models M and M'. So the sequence of the M_η have monotonically increasing interpretations of the inductive relations and frame expressions, and are equal to M on the interpretation of the other relations and functions. The frame model M

is obtained at some stage ξ of the fixpoint iteration, so $M = M_\xi$. More precisely, the frame model is contructed by first iterating the operator μ_{f} until the fixpoint of the frame expressions is reached. During this iteration, the inductive relations are interpreted as empty. Then the operator μ_{i} is iterated until also the inductive relations reach their fixpoint. Below, we do an induction on η. In that induction, we do not explicitly distinguish these two phases. because it does not play any role for the arguments (only in one place and we mention it explicitly there).

By induction on η, we can show that $M_\eta, \nu \models \varphi \Leftrightarrow M'_\eta, \nu \models \varphi$, and $[\![t]\!]_{M_\eta, \nu} = [\![t]\!]_{M'_\eta, \nu}$ for all variable assignments ν and all formulas φ with $[\![Fr(\varphi)]\!]_M(\nu) \subseteq X$, respectively terms t with $[\![Fr(t)]\!]_M(\nu) \subseteq X$. For each η, we furthermore do an induction on the structure of the formulas, respectively terms.

Note that the assumption that the frame is contained in X refers to the frame in M. So when applying the induction, we have to verify that the condition on the frame of a formula/term is satisfied in M (and not in M_η).

For the formulas, the induction is straight forward, using Lemma 1 (see Appendix VIII-A) in the cases of existential formulas and ite-formulas. Consider, for example, the case of an existential formula $\psi = \exists y : \gamma.\varphi$ with $[\![Fr(\psi)]\!]_M \subseteq X$.

$$
\begin{aligned}
& M_\eta, \nu \models \exists y : \gamma.\varphi \\
\Leftrightarrow \quad & \text{exists } u \in D_y : M_\eta, \nu[y \leftarrow u] \models \gamma \\
& \text{and } M_\eta, \nu[y \leftarrow u] \models \varphi \\
\overset{(*)}{\Leftrightarrow} \quad & \text{exists } u \in D_y : M'_\eta, \nu[y \leftarrow u] \models \gamma \\
& \text{and } M'_\eta, \nu[y \leftarrow u] \models \varphi \\
\Leftrightarrow \quad & M'_\eta, \nu \models \exists y : \gamma.\varphi
\end{aligned}
$$

where $(*)$ holds by induction on the structure of the formula. We only have to verify that $[\![Fr(\gamma)]\!]_M(\nu[y \leftarrow u]) \subseteq X$ and $[\![Fr(\varphi)]\!]_M(\nu[y \leftarrow u]) \subseteq X$ in order to use the induction hypothesis.

Since γ is a guard of an existential formula, it satisfies the condition of Lemma 1 (see Appendix VIII-A), and therefore its truth value is the same in $(M_\eta, \nu[y \leftarrow u])$ for all ordinals η (Lemma 1 applies because all the models M_η differ only in the interpretations of the frame expressions and inductive relations, and thus have the same pre-model). In particular, $M, \nu[y \leftarrow u] \models \gamma$ since $M = M_\xi$ for some ordinal ξ. From the equations for the frames we obtain $[\![Fr(\gamma)]\!]_M(\nu[y \leftarrow u]) \subseteq [\![Fr(\psi)]\!]_M(\nu)$ and $[\![Fr(\varphi)]\!]_M(\nu[y \leftarrow u]) \subseteq [\![Fr(\psi)]\!]_M(\nu)$. The desired claim now follows from the fact that $[\![Fr(\psi)]\!]_M(\nu) \subseteq X$.

For inductive relations R with definition $R(\overline{x}) := \rho_R(\overline{x})$, we have to use the induction on the ordinal η. Assume that $\varphi = R(\overline{t})$ for $\overline{t} = (t_1, \ldots, t_n)$, and that $[\![Fr(\varphi)]\!]_M(\nu) \subseteq X$. Then $[\![Fr(\rho_R(\overline{x}))]\!]_M(\nu[\overline{x} \leftarrow \overline{t}]) \subseteq X$ and $[\![Fr(t_i)]\!]_M(\nu) \subseteq X$ for all i by the frame equations.

For the case of a limit ordinal η, the inductive relations of M_η, resp. M'_η, are obtained by taking union of the interpretations of the inductive relations for all M_ζ, resp. M'_ζ, for all $\zeta < \eta$. So the claim follows directly by induction.

For a successor ordnial $\eta + 1$, we can assume that we are in the second phase of the construction of the frame model (the iteration of the operator μ_{i}). For the first phase the claim trivially holds because all the inductive relations are interpreted as empty. Thus, we have

$$
\begin{aligned}
M_{\eta+1}, \nu \models R(\overline{t}) \quad \Leftrightarrow \quad & M_\eta, \nu \models \rho_R(\overline{t}) \\
\Leftrightarrow \quad & M_\eta, \nu[\overline{x} \leftarrow [\![\overline{t}]\!]_{M_\eta, \nu}] \models \rho_R(\overline{x}) \\
\overset{(*)}{\Leftrightarrow} \quad & M'_\eta, \nu[\overline{x} \leftarrow [\![\overline{t}]\!]_{M'_\eta, \nu}] \models \rho_R(\overline{x}) \\
\Leftrightarrow \quad & M'_{\eta+1}, \nu \models R(\overline{t})
\end{aligned}
$$

where $(*)$ holds by induction on η.

The other cases for formulas are similar (or simpler).

Concerning the terms, we also present some cases only, the other cases being similar or simpler.

We start with the case $t = f(t_1, \ldots, t_n)$ for a mutable function f. Let ν be a variable assignment with $[\![t]\!]_{M,\nu} \subseteq X$. By the frame equations, $[\![t_i]\!]_{M,\nu} \subseteq X$ for all i. We have $[\![t]\!]_{M_\eta, \nu} = [\![f]\!]_M([\![t_1]\!]_{M_\eta, \nu}, \ldots, [\![t_n]\!]_{M_\eta, \nu})$. By induction on the structure of terms, we have $[\![t_i]\!]_{M_\eta, \nu} = [\![t_i]\!]_{M'_\eta, \nu} =: u_i$. By Lemma 1 (see Appendix VIII-A), we conclude that $[\![t_i]\!]_{M_\eta, \nu} = [\![t_i]\!]_{M, \nu}$. Since f is mutable, it contains at least one argument of sort σ_{f}, say t_j. Then $[\![t_j]\!]_{M,\nu} \in [\![Fr(t)]\!]_M(\nu) \subseteq X$, and the mutation did not change the function value of f on the tuple $(u_1, \ldots, u_n)$. So we obtain in summary that $[\![t]\!]_{M',\nu} = [\![f]\!]_{M'}(u_1, \ldots, u_n) = [\![f]\!]_M(u_1, \ldots, u_n) = [\![t]\!]_{M,\nu}$.

Now consider terms of the form $Fr(\varphi)$. We need to proceed by induction on the structure of φ. We present the case of $\varphi = ite(\gamma : \varphi_1, \varphi_2)$. Let ν be a variable assignment with $[\![Fr(\varphi)]\!]_M(\nu) \subseteq X$. Assume that $M_\eta, \nu \models \gamma$. By the condition on guards, Lemma 1 yields that $M, \nu \models \gamma$ and thus $[\![Fr(\gamma)]\!]_M(\nu) \subseteq X$ and $[\![Fr(\varphi_1)]\!]_M(\nu) \subseteq X$. We obtain

$$
\begin{aligned}
[\![Fr(\varphi)]\!]_{M_\eta}(\nu) \quad = \quad & [\![Fr(\gamma)]\!]_{M_\eta}(\nu) \cup [\![Fr(\varphi_1)]\!]_{M_\eta}(\nu) \\
\overset{(*)}{=} \quad & [\![Fr(\gamma)]\!]_{M'_\eta}(\nu) \cup [\![Fr(\varphi_1)]\!]_{M'_\eta}(\nu) \\
= \quad & [\![Fr(\varphi)]\!]_{M'_\eta}(\nu)
\end{aligned}
$$

where $(*)$ follows by induction on the structure of the formula inside the frame expression. The case $M_\eta, \nu \not\models \gamma$ is analogous.

Now consider $Fr(\varphi)$ with $\varphi = R(\overline{t})$ for an inductively defined relation R with definition $R(\overline{x}) = \rho_R(\overline{x})$ and $\overline{t} = (t_1, \ldots, t_n)$. Let ν be a variable assignment with $[\![Fr(\varphi)]\!]_M(\nu) \subseteq X$. By the frame equations, $[\![Fr(\rho_R(\overline{x}))]\!]_M(\nu[\overline{x} \leftarrow [\![\overline{t}]\!]_{M,\nu}]) \subseteq X$ and $[\![Fr(t_i)]\!]_M(\nu) \subseteq X$.

Let $\eta + 1$ be a successor ordinal. Then

$$
\begin{aligned}
& [\![Fr(R(\overline{t}))]\!]_{M_{\eta+1}}(\nu) \\
&= [\![Fr(\rho_R(\overline{x}))]\!]_{M_\eta}(\nu[\overline{x} \leftarrow [\![\overline{t}]\!]_{M_\eta,\nu}]) \cup \bigcup_{i=1}^{n} [\![Fr(t_i)]\!]_{M_\eta}(\nu) \\
&\overset{(*)}{=} [\![Fr(\rho_R(\overline{x}))]\!]_{M'_\eta}(\nu[\overline{x} \leftarrow [\![\overline{t}]\!]_{M'_\eta,\nu}]) \cup \bigcup_{i=1}^{n} [\![Fr(t_i)]\!]_{M'_\eta}(\nu) \\
&= [\![Fr(R(\overline{t}))]\!]_{M'_{\eta+1}}(\nu)
\end{aligned}
$$

where $(*)$ holds by induction on η. We can apply the induction hypothesis because the terms t_i do not contain frame expressions by the restriction on the type of inductive relations, and thus $[\![\overline{t}]\!]_{M'_\eta,\nu} = [\![\overline{t}]\!]_{M_\eta,\nu} = [\![\overline{t}]\!]_{M,\nu}$ by Lemma 1.

The proof of the other cases works in a similar fashion. $\square$

C. Translation of Frame equations to FO-RD

The translation from Frame Logic to FO-RD is detailed in Section III-E. The crucial element is the translation of the semantics of the *Fr* operator to FO-RD which is shown in Figure 4.

D. Operational Semantics

The full operational semantics are in Figure 5. We represent a configuration as $\langle s, h, A, U \rangle$ here instead of (M, H, U) as in the main paper. Here, s represents the store, h represents the heap, A denotes the set of allocated locations, and U denotes unallocated locations.

The pointer lookup rule changes the store where the variable x now maps to $f(y)$, provided y had been allocated. The pointer modification rule modifies the heap on the function f, where the store's interpretation for x now maps to the store's interpretation for y. The allocation rule is actually the only nondeterministic rule in the operational semantics, as there is a transition for each $a \notin A$. For each such a, provided x had not been allocated ($s(x) \notin A$), the store is modified where x now points to a. Additionally, the heap is modified for each function f where the newly allocated a maps to the default value for each f. All other rules are straightforward.

Note that when side conditions are violated as in the lookup rule or pointer modification rule, the configuration transitions to $\bot$, which denotes an abort or fault configuration. These faulting transitions are crucial for the soundness of the frame rule. However, in the allocation rule, there is no transition to bottom if a side condition is violated. Instead, no transition occurs at all, and the configuration gets stuck. This is because for the weakness condition of the allocation program logic rule, we want to include states in which no location can be allocated.

E. Definitions of MW primitives

We have already seen the definition of $MW^{x.f:=y}$ in Section IV-A. We will detail the construction of $MW_v^{\mathsf{alloc}(x)}$ in this section.

$MW_v^{\mathsf{alloc}(x)}$, like $MW^{x.f:=y}$, is also meant to evaluate a formula in the pre-state as though it were evaluated in the post-state. However, note that the frame of this formula must not contain the allocated location (say x). Since we know from the operational semantics of allocation that the allocated location is going to point to default values, we can proceed similarly as we did for the previous definition, identify terms evaluating to $f(x)$ and replace them with the default value (under f). This has the intended effect of evaluating to the same value as in the post-state while removing x from the frame.

However, this approach fails when we apply it to frame expressions (since removing x from the frame guarantees that we would no longer compute the 'same' value as of that in the post-state). In particular, a sub-formula of the form $t \in Fr(\gamma)$ may be falsified by that transformation. To handle this, we identify when x might be in the frame of a given expression and replace it with v (which is given as a parameter) such that neither x nor v is dereferenced, and will not be in the frame of the resulting transformation. For the program logic rule we then interpret v to any of the locations outside the allocated set in the pre-state and demand that weakest-pre satisfy the transformed formula for any such location (as it must, since we do not know which of the hitherto unallocated locations might be allocated as a result of the command).

We define $MW_v^{\mathsf{alloc}(x)}$ inductively. We first consider the case where β does not contain any subformulas involving frame expressions or inductive definitions. Then, we have $MW_v^{\mathsf{alloc}(x)}$ defined as follows:

$$
\begin{aligned}
MW_v^{\mathsf{alloc}(x)}(\beta) = \\
\beta[v/x][\lambda z.\ ite(z = v : def_f, f(z))/f]_{f \in F}[U/U \setminus \{v\}]
\end{aligned}
$$

where this means for each instance of a (mutable or immutable) function f in β, we replace $f(x)$ with a default value. We also replace all free instances of x in β with v, as x should not appear free in the precondition since it had not yet been allocated. Further we must transform all the unallocated sets to be used later in the program to not contain v (which contains the value we intend to allocate in the current step).

If $Fr(\gamma)$ is a subterm of β, we translate it to a term $Fr_\gamma^{x;v}$ inductively as in Figure 6. This definition is very similar to the translation of FL formulae to FO-RD in Figure 4 where we replace free instances of x with v. Since this is a relation, we must transform membership to evaluation, i.e, transform expressions of the form $t \in Fr(\gamma)$ to $Fr_\gamma^{x;v}(\overline{y}, MW_v^{\mathsf{alloc}(x)}(t))$ where $\overline{y}$ are the

$$
\begin{aligned}
Fr_c(\overline{y}, z) &:= \text{false} && \text{for a constant } c \\
Fr_x(\overline{y}, z) &:= \text{false} && \text{for a variable } x \\[6pt]
Fr_{f(t_1,\ldots,t_n)}(\overline{y}, z) &:=
\begin{cases}
\left(\displaystyle\bigvee_{t_i \text{ of sort } \sigma_f} z = t_i \right) \vee \displaystyle\bigvee_{i=1}^{n} Fr_{t_i}(\overline{y}, z) & \text{if } f \in F_{\mathsf{m}} \\[12pt]
\displaystyle\bigvee_{i=1}^{n} Fr_{t_i}(\overline{y}, z) & \text{if } f \notin F_{\mathsf{m}}
\end{cases} \\[6pt]
Fr_{Fr(\beta)}(\overline{y}, z) &:= Fr_\beta(\overline{y}, z) \\
Fr_{R(t_1,\ldots,t_n)}(\overline{y}, z) &:= \textstyle\bigvee_{i=1}^{n} Fr_{t_i}(\overline{y}, z) \text{ for } R \in \mathcal{R} \\
Fr_{(t_1 = t_2)}(\overline{y}, z) &:= Fr_{t_1}(\overline{y}, z) \vee Fr_{t_2}(\overline{y}, z) \\
Fr_{R(\overline{t})}(\overline{y}, z) &:= Fr_{\rho_{R(\overline{x})}}(\overline{y}, z)[\overline{t}/\overline{x}] \vee \textstyle\bigvee_{i=1}^{n} Fr_{t_i}(\overline{y}, z) \\
&\quad\ \text{for } R \in \mathcal{I} \text{ with definition } R(\overline{x}) := \rho_R(\overline{x}) \\
Fr_{\beta_1 \wedge \beta_2}(\overline{y}, z) &:= Fr_{\beta_1}(\overline{y}, z) \vee Fr_{\beta_1}(\overline{y}, z) \\
Fr_{\neg\beta}(\overline{y}, z) &:= Fr_\beta(\overline{y}, z) \\
Fr_{ite(\gamma:\beta_1,\beta_2)}(\overline{y}, z) &:= Fr_\gamma(\overline{y}, z) \vee ite(\gamma : Fr_{\beta_1}(\overline{y}, z), Fr_{\beta_2}(\overline{y}, z)) \\
Fr_{ite(\gamma:t_1,t_2)}(\overline{y}, z) &:= Fr_\gamma(\overline{y}, z) \vee ite(\gamma : Fr_{t_1}(\overline{y}, z), Fr_{t_2}(\overline{y}, z)) \\
Fr_{\exists x:\gamma.\beta}(\overline{y}, z) &:= \exists x.(Fr_\gamma(\overline{y}, z) \vee (\gamma \wedge Fr_\beta(\overline{y}, z)))
\end{aligned}
$$

Fig. 4. Translation of frame equations to FO-RD.

$$
\begin{aligned}
&\langle s, h, A, U \rangle \xRightarrow{x := y} \langle s[x \mapsto y], h, A \rangle \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{x := c} \langle s[x \mapsto c], h, A \rangle \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{v := be} \langle s[v \mapsto be], h, A \rangle \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{x := y.f} \langle s[x \mapsto f(y)], h, A \rangle, \text{ if } s(y) \in A \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{x := y.f} \bot, \text{ if } s(y) \notin A \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{\text{if } be \text{ then } S \text{ else } T} \langle s', h', A', U' \rangle, \text{ if } s \models be \text{ and } \langle s, h, A, U \rangle \xRightarrow{S} \langle s', h', A' \rangle \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{\text{if } be \text{ then } S \text{ else } T} \langle s', h', A', U' \rangle, \text{ if } s \not\models be \text{ and } \langle s, h, A, U \rangle \xRightarrow{T} \langle s', h', A' \rangle \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{\text{while } be \text{ do } S} \langle s', h', A', U' \rangle, \text{ if } s \models be \text{ and } \langle s, h, A, U \rangle \xRightarrow{S \,;\, \text{while } be \text{ do } S} \langle s', h', A' \rangle \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{\text{while } be \text{ do } S} \langle s, h, A, U \rangle, \text{ if } s \not\models be \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{x.f := y} \langle s, h[f \mapsto f[s(x) \mapsto s(y)]], A \rangle, \text{ if } s(y) \in A \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{x.f := y} \bot, \text{ if } s(y) \notin A \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{\text{alloc}(x)} \langle s[x \mapsto a], h[f \mapsto f[a \mapsto def_f]], A \cup \{a\}, U \setminus \{a\} \rangle, \text{ for all } f \in F \\
&\qquad\quad \text{if } s(x) \notin A, a \notin A, \text{ and } a \in U \\[4pt]
&\langle s[x \mapsto a], h, A \rangle \xRightarrow{\text{free}(x)} \langle s, h, A, U' \setminus \{s(x)\} \rangle, \text{ if } s(x) \in A \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{\text{free}(x)} \bot, \text{ if } s(x) \notin A \\[4pt]
&\langle s, h, A, U \rangle \xRightarrow{S \,;\, T} \langle s'', h'', A'' \rangle, \text{ if } \langle s, h, A, U \rangle \xRightarrow{S} \langle s', h', A' \rangle \text{ and } \langle s', h', A', U' \rangle \xRightarrow{T} \langle s'', h'', A'', U'' \rangle
\end{aligned}
$$

Fig. 5. Operational Semantics of Frame Logic Programming Language

$$
\begin{aligned}
Fr^{x;v}_c(\overline{y},z) &:= \text{false} \quad\text{for a constant } c\\
Fr^{x;v}_w(\overline{y},z) &:= \text{false} \quad\text{for a variable } w\\
Fr^{x;v}_{f(t)}(\overline{y},z) &:= \begin{cases} \Big(\big(z = MW^{\mathsf{alloc}(x)}_v(t)\big) \vee Fr^{x;v}_t(\overline{y},z) \wedge \big(MW^{\mathsf{alloc}(x)}_v(f(t) = f(t))\big)\Big) & \text{if } f \in F_{\mathsf{m}}\\ Fr^{x;v}_t(\overline{y},z) & \text{if } f \notin F_{\mathsf{m}} \end{cases}\\
Fr^{x;v}_{Fr(\beta)}(\overline{y},z) &:= Fr^{x;v}_\beta(\overline{y},z)\\
Fr^{x;v}_{(t_1=t_2)}(\overline{y},z) &:= Fr^{x;v}_{t_1}(\overline{y},z) \vee Fr^{x;v}_{t_2}(\overline{y},z)\\
Fr^{x;v}_{R(\overline{t})}(\overline{y},z) &:= Fr^{x;v}_{\rho_R(\overline{w})}(\overline{y},z)[MW^{\mathsf{alloc}(x)}_v(\overline{t})/\overline{w}] \vee \bigvee^n_{i=1} Fr^{x;v}_{t_i}(\overline{y},z)\\
&\quad\ \text{for } R \in \mathcal{I} \text{ with definition } R(\overline{w}) := \rho_R(\overline{w})\\
Fr^{x;v}_{\beta_1 \wedge \beta_2}(\overline{y},z) &:= Fr^{x;v}_{\beta_1}(\overline{y},z) \vee Fr^{x;v}_{\beta_1}(\overline{y},z)\\
Fr^{x;v}_{\neg\beta}(\overline{y},z) &:= Fr^{x;v}_\beta(\overline{y},z)\\
Fr^{x;v}_{ite(\gamma:\beta_1,\beta_2)}(\overline{y},z) &:= Fr^{x;v}_\gamma(\overline{y},z) \vee ite(MW^{\mathsf{alloc}(x)}_v(\gamma) : Fr^{x;v}_{\beta_1}(\overline{y},z), Fr^{x;v}_{\beta_2}(\overline{y},z))\\
Fr^{x;v}_{ite(\gamma:t_1,t_2)}(\overline{y},z) &:= Fr^{x;v}_\gamma(\overline{y},z) \vee ite(MW^{\mathsf{alloc}(x)}_v(\gamma) : Fr^{x;v}_{t_1}(\overline{y},z), Fr^{x;v}_{t_2}(\overline{y},z))\\
Fr^{x;v}_{\exists w:\gamma.\beta}(\overline{y},z) &:= \exists w : \Big(MW^{\mathsf{alloc}(x)}_v(\gamma)\Big) . \Big(Fr^{x;v}_\gamma(\overline{y},z) \vee \big(MW^{\mathsf{alloc}(x)}_v(\gamma) \wedge Fr^{x;v}_\beta(\overline{y},z)\big)\Big)
\end{aligned}
$$

Fig. 6. Definition of $Fr^{x;v}$ for use in $MW^{\mathsf{alloc}(x)}_v$.

free variables (we transform inductively— at the highest level free variables will be program/ghost variables). We also transform union of frame expressions to disjunction of the corresponding relations, equality to (quantified) double implication, etc.

For a subterm of β of the form $I(\overline{t})$ where I is an inductive definition with body ρ_I, we translate it to $I'(MW^{\mathsf{alloc}(x)}_v(\overline{t}))$ where the body of I' is defined as $MW^{\mathsf{alloc}(x)}_v(\rho_{I'})$.

The above cases can be combined with boolean operators and if-then-else, which $MW^{\mathsf{alloc}(x)}_v$ distributes over.

F. Program Logic Proofs

This section contains the soundness proofs for all global rules in Section IV. Since only the allocation rule modifies U, we represent a configuration as (M, H) for all rules in this section besides the allocation rule.

Theorem 6 (Lookup Soundness). *Let M be a model and H a sub-universe of locations such that*

$$M \models \exists x' : x' = f(y). (\beta \wedge y \in Fr(\beta))[x'/x]$$
$$H = [\![Fr(\exists x' : x' = f(y). (\beta \wedge y \in Fr(\beta)))[x'/x]]\!]_M$$

Then $(M,H) \xrightarrow{x:=y.f} (M',H')$, $M' \models \beta$, and $H' = [\![Fr(\beta)]\!]_{M'}$.

Proof. Observe that $[\![y]\!]_M \in H$ since y is in the frame of the precondition. Therefore we know $(M,H) \xrightarrow{x:=y.f} (M',H')$ where $M' = M[x \mapsto [\![f(y)]\!]_M]$ and $H' = H$. Next, note that if there is a formula α (or term t) where x is not a free variable of α (or t), then M and M' have the same valuation of α (or t). This is true because the semantics of lookup only changes the valuation for

x on M. In particular, $M' \models \exists x' : x' = f(y). (\beta \wedge y \in Fr(\beta))[x'/x]$. Thus,

$$
\begin{aligned}
&M' \models \exists x' : x' = f(y). (\beta \wedge y \in Fr(\beta))[x'/x]\\
&\implies M'[x' \mapsto c]\\
&\qquad \models x' = f(y) \wedge (\beta \wedge y \in Fr(\beta))[x'/x]\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(for some } c)\\
&\implies M'[x' \mapsto [\![f(y)]\!]_{M'}] \models (\beta \wedge y \in Fr(\beta))[x'/x]\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(since } f \text{ is a function)}\\
&\implies M'[x' \mapsto [\![x]\!]_{M'}] \models (\beta \wedge y \in Fr(\beta))[x'/x]\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(operational semantics)}\\
&\implies M'[x' \mapsto [\![x]\!]_{M'}]\\
&\qquad \models ((\beta \wedge y \in Fr(\beta))[x'/x])[x/x']\\
&\implies M'[x' \mapsto [\![x]\!]_{M'}] \models \beta \wedge y \in Fr(\beta)\\
&\implies M' \models \beta \wedge y \in Fr(\beta) \quad (\beta \text{ does not mention } x')\\
&\implies M' \models \beta
\end{aligned}
$$

The heaplet condition follows from a similar argument. Specifically

$$
\begin{aligned}
H' &= H\\
&= [\![Fr(\exists x' : x' = f(y). (\beta \wedge y \in Fr(\beta)))[x'/x]]\!]_M\\
&= [\![Fr(\exists x' : x' = f(y). (\beta \wedge y \in Fr(\beta)))[x'/x]]\!]_{M'}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(does not mention } x)\\
&= \{[\![y]\!]_{M'}\} \cup\\
&\quad\ [\![Fr((\beta \wedge y \in Fr(\beta))[x'/x])]\!]_{M'}(x' \mapsto [\![f(y)]\!]_{M'})\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(def of } Fr)\\
&= \{[\![y]\!]_{M'}\} \cup\\
&\quad\ [\![Fr((\beta \wedge y \in Fr(\beta))[x'/x])]\!]_{M'}(x' \mapsto [\![x]\!]_{M'})\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(operational semantics)}\\
&= \{[\![y]\!]_{M'}\} \cup [\![Fr(\beta \wedge y \in Fr(\beta))]\!]_{M'}\\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(similar reasoning as above)}
\end{aligned}
$$

19

$$= [\![Fr(\beta)]\!]_{M'} \qquad \text{(since } M' \models y \in Fr(\beta) \text{ from above)}$$

$\square$

Theorem 7 (WTP Lookup)**.** *Let M, M' be models with H, H' sub-universes of locations (respectively) such that $(M, H) \xRightarrow{x := y.f} (M', H')$, $M' \models \beta$ and $H' = [\![Fr(\beta)]\!]_{M'}$. Then*

$$M \models \exists x' : x' = f(y). \, (\beta \wedge y \in Fr(\beta))[x'/x]$$
$$\text{(weakest-pre)}$$
$$H = [\![Fr(\exists x' : x' = f(y). \, (\beta \wedge y \in Fr(\beta)))]\!]_M$$
$$\text{(tightest-pre)}$$

Proof. Both parts follow by simply retracing steps in the above proof. The weakness claim follows from the first part of the proof above, where all implications can be made bidirectional (using operational semantics rules, definition of existential quantifier, etc.). The tightness claim follows immediately from the second part of the proof above as all steps involve equalities. $\square$

For soundness of the pointer modification rules, we prove the following lemma:

Lemma 2. *Given a formula β (term t) and configurations (M, H) and (M', H') such that (M, H) transforms to (M', H') on the command $x.f := y$, then $[\![MW^{x.f := y}(\beta)]\!]_M = [\![\beta]\!]_{M'}$. Additionally, $[\![Fr(MW^{x.f := y}(\beta))]\!]_M = [\![Fr(\beta)]\!]_{M'}$. Both equalities hold for terms t as well.*

Proof. Induction on the structure of β, unfolding $MW^{x.f := y}(\beta)$ accordingly. We discuss one interesting case here, namely when β has a subterm of the form $f(t)$. Now, we have two cases, depending on whether $[\![MW^{x.f := y}]\!]_M = [\![x]\!]_M$. If it does, then

$$[\![MW^{x.f := y}(f(t))]\!]_M$$
$$= [\![MW^{x.f := y}(f)]\!]_M([\![MW^{x.f := y}(t)]\!]_M) \quad \text{(definition)}$$
$$= [\![MW^{x.f := y}(f)]\!]_M([\![x]\!]_M) \quad \text{(assumption)}$$
$$= [\![MW^{x.f := y}(f(x))]\!]_M \quad \text{(defintion)}$$
$$= [\![y]\!]_M = [\![y]\!]_{M'} \quad \text{(def of } MW^{x.f := y})$$
$$= [\![f(x)]\!]_{M'} = [\![f]\!]_{M'}([\![x]\!]_{M'}) \quad \text{(def of } f \text{ on } M')$$
$$= [\![f]\!]_{M'}([\![x]\!]_M) \quad \text{(definition of } M, M')$$
$$= [\![f]\!]_{M'}([\![MW^{x.f := y}(t)]\!]_M) \quad \text{(assumption)}$$
$$= [\![f]\!]_{M'}([\![t]\!]_{M'}) \quad \text{(induction hypothesis)}$$
$$= [\![f(t)]\!]_{M'} \quad \text{(definition)}$$

The proof for the cases when $[\![MW^{x.f := y}(t)]\!]_M \neq [\![x]\!]_M$ and the heaplet equality claims are similar, and all other cases are trivial. $\square$

Theorem 8 (Mutation Soundness)**.** *Let M be a model and H a sub-universe of locations such that*

$$M \models MW^{x.f := y}(\beta \wedge x \in Fr(\beta))$$
$$H = [\![Fr(MW^{x.f := y}(\beta \wedge x \in Fr(\beta)))]\!]_M$$

Then $(M, H) \xRightarrow{x.f := y} (M', H'), M' \models \beta$, and $H' = [\![Fr(\beta)]\!]_{M'}$

Proof. From the definition of the transformation $MW^{x.f := y}$, we have that $MW^{x.f := y}(\beta \wedge x \in Fr(\beta))$ will be transformed to the same formula as $MW^{x.f := y}(\beta) \wedge x \in Fr(MW^{x.f := y}(\beta))$, the heaplet of which, since the formula holds on M, contains x. Therefore $x \in H$ and from the operational semantics we have that $(M, H) \xRightarrow{x.f := y} (M', H')$ for some (M', H') such that $H = H'$.

From Lemma 2 we have that $M' \models \beta \wedge x \in Fr(\beta)$, since M models the same. In particular $M' \models \beta$. Moreover we have

$$H' = H \qquad \text{(operational semantics)}$$
$$= [\![Fr(MW^{x.f := y}(\beta \wedge x \in Fr(\beta)))]\!]_M \quad \text{(given)}$$
$$= [\![Fr(\beta \wedge x \in Fr(\beta))]\!]_{M'} \quad \text{(Lemma 2)}$$
$$= [\![Fr(\beta)]\!]_{M'} \quad \text{(semantics of H operator)}$$

Therefore $M' \models \beta$ and $H' = [\![Fr(\beta)]\!]_{M'}$ which makes our pointer mutation rule sound. $\square$

Theorem 9 (WTP Mutation)**.** *Let M, M' be models with H, H' sub-universes of locations (respectively) such that $(M, H) \xRightarrow{x.f := y} (M', H')$, $M' \models \beta$ and $H' = [\![Fr(\beta)]\!]_{M'}$. Then*

$$M \models MW^{x.f := y}(\beta \wedge x \in Fr(\beta)) \quad \text{(weakest-pre)}$$
$$H = [\![Fr(MW^{x.f := y}(\beta \wedge x \in Fr(\beta)))]\!]_M$$
$$\text{(tightest-pre)}$$

Proof. From the operational semantics, we have that $(M, H) \xRightarrow{x.f := y} (M', H')$ only if $x \in H$ and $H = H'$. Therefore $x \in H' = [\![Fr(\beta)]\!]_{M'}$ (given) which in turn implies that $M' \models \beta \wedge x \in Fr(\beta)$ as well as $H' = [\![Fr(\beta \wedge x \in Fr(\beta))]\!]_{M'}$. Applying Lemma 2 yields the result. $\square$

Lemma 3. *Given a formula β (or term t) and configurations (M, H, U) and (M', H', U') such that (M, H, U) transforms to (M', H', U') on the command $\mathsf{alloc}(x)$, then $[\![Fr^{x;v}(\overline{y}, MW_v^{\mathsf{alloc}(x)}(t))]\!]_{M[v \mapsto a]}$ iff $[\![t \in Fr(\beta)]\!]_{M'}$, where $a = [\![x]\!]_{M'}$ and $\overline{y}$ are the free variables in $MW_v^{\mathsf{alloc}(x)}(\beta)$. Additionally, $[\![Fr(Fr_\beta^{x;v}(\overline{y}, z))]\!]_{M[v \mapsto a]} = [\![Fr(\beta) \setminus \{x\}]\!]_{M'}$ where z is a free variable. Both equalities hold for terms t as well.*

Proof. Induction on the structure of β and using the construction in Figure 6. For the second claim about

the frame of $Fr^{x;v}$, the fact that we only allow specific kinds of guards is crucial in the inductive case of the existential quantifier. $\qquad\square$

Lemma 4. *Given a formula β (or term t) and configurations (M, H, U) and (M', H', U') such that (M, H, U) transforms to (M', H', U') on the command* alloc(x), *then* $[\![MW_v^{\mathsf{alloc}(x)}(\beta)]\!]_{M[v \mapsto a]} = [\![\beta]\!]_{M'}$, *where* $a = [\![x]\!]_{M'}$. *Additionally,* $[\![Fr(MW_v^{\mathsf{alloc}(x)}(\beta))]\!]_{M[v \mapsto a]} = [\![Fr(\beta) \backslash \{x\}]\!]_{M'}$. *Both equalities hold for terms t as well.*

Proof. First, we split on the structure of β, as the definition of $MW_v^{\mathsf{alloc}(x)}$ differs depending on the form of β. For subformulas with no frame expressions or inductive definitions, the proof follows from the syntactic definition of $MW_v^{\mathsf{alloc}(x)}$ and is very similar to Lemma 2. Subformulas with frame expressions follow by construction using Lemma 3, and formulas with inductive definitions follow by construction as well. Boolean combinations and if-then-else follow using the inductive hypothesis. $\qquad\square$

We are now ready to prove the soundness and WTP property of the allocation rule. This will be different from the other soundness theorems because it reasons only about configurations reachable by a program or a valid initial state. This strengthening of the premise is not an issue since we will only ever execute commands on such states. We shall first prove a lemma.

Lemma 5. *Let β be any formula within our restricted fragment (Section IV) and (M, H, U) be a valid configuration. Then, for any locations $a_1, a_2 \in U$:*

$$[\![MW_v^{\mathsf{alloc}(x)}(\beta)]\!]_{M[v \mapsto a_1]} = [\![MW_v^{\mathsf{alloc}(x)}(\beta)]\!]_{M[v \mapsto a_2]}$$
and
$$[\![Fr(MW_v^{\mathsf{alloc}(x)}(\beta))]\!]_{M[v \mapsto a_1]} =$$
$$[\![Fr(MW_v^{\mathsf{alloc}(x)}(\beta))]\!]_{M[v \mapsto a_2]}$$

Proof. The proof follows by a simple inductive argument on the structure of β. First observe that in any model if v is interpreted to an unallocated location (more generally a location outside of H) it is never contained in $MW_v^{\mathsf{alloc}(x)}(\beta)$ since it is never dereferenced. Therefore, all we are left to prove is that the actual value of v (between choices in U) influences neither the truth value nor the frame of the formula. The key case is that of *ite* expressions where the value of v can influence the truth of the guard. This case can be resolved using the observation that since (M, H, U) is a valid configuration, the value of any unallocated location can never equal that of a program variable. Since we have no atomic relations either in our restricted fragment, any two values in U are indistinguishable by a formula in this fragment.

In particular, any *ite* expressions that depend on the value of v either compare it with a term over a program variable —which is never equal, or compare it with a quantified variable — which itself only takes on values allowed by the guard of the quantification that, inductively, does not distinguish between values in U. $\qquad\square$

Theorem 10 (Allocation Soundness). *Let (M, H, U) be a valid configuration such that*

$$M \models \forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right)$$

$$H = [\![Fr(\forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right))]\!]_M$$

$$(M, H, U) \xrightarrow{\mathsf{alloc}(x)} (M', H', U \backslash [\![x]\!]_{M'})$$

Then $M' \models \beta$ and $H' = [\![Fr(\beta)]\!]_{M'}$

Proof. Let a be the actual location allocated, i.e., $a = [\![x]\!]_{M'}$. Clearly $a \in U$ by the operational semantics. Then, we have:

$$M \models \forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right)$$
$$\implies M[v \mapsto a] \models (v \in U) \Rightarrow \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right)$$
$$\implies M[v \mapsto a] \models MW_v^{\mathsf{alloc}(x)}(\beta)$$
$$(a \in U \text{ by operational semantics})$$
$$\implies M' \models \beta \qquad \text{(Lemma 4)}$$

For the frame claim, we have:

$$H = [\![Fr(\forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right))]\!]_M$$
$$= \bigcup_{s \in U} [\![Fr(MW_v^{\mathsf{alloc}(x)}(\beta))]\!]_{M[v \mapsto s]}$$
$$\text{(definition of } Fr \text{ operator)}$$
$$= [\![Fr(MW_v^{\mathsf{alloc}(x)}(\beta))]\!]_{M[v \mapsto a]} \qquad \text{(Lemma 5)}$$
$$= [\![Fr(\beta) \backslash \{x\}]\!]_{M'} \qquad \text{(Lemma 4)}$$

Now $H' = H \cup \{[\![x]\!]_{M'}\}$ (by operational semantics) $= [\![Fr(\beta) \backslash \{x\}]\!]_{M'} \cup \{[\![x]\!]_{M'}\} = [\![Fr(\beta)]\!]_{M'}$, as desired. $\qquad\square$

Theorem 11 (WTP Allocation). *Let (M, H, U) and $(M', H', U \backslash [\![x]\!]_{M'})$ be valid configurations such that*

$$(M, H, U) \xrightarrow{\mathsf{alloc}(x)} (M', H', U \backslash [\![x]\!]_{M'})$$
$$M' \models \beta \text{ and } H' = [\![Fr(\beta)]\!]_{M'}$$

Then

$$M \models \forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right)$$

$$H = [\![Fr(\forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right))]\!]_M$$

Proof. The first claim follows easily from an application of Lemma 4 followed by an application of Lemma 5. For the second claim, observe that as done in the proof above for Theorem 10 we can prove that $[\![Fr(\beta) \backslash \{x\}]\!]_{M'} = [\![Fr(\forall v : (v \in U) . \left(MW_v^{\mathsf{alloc}(x)}(\beta) \right))]\!]_M$. The proof

concludes by observing that by the operational semantics we have $H = H' \setminus \{[\![x]\!]_{M'}\} = [\![Fr(\beta)]\!]_{M'} \setminus \{[\![x]\!]_{M'}\} = [\![Fr(\beta) \setminus \{x\}]\!]_{M'}$ $\qquad\square$

Theorem 12 (Deallocation Soundness)**.** *Let M be a model and H a sub-universe of locations such that*

$$M \models \beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x)$$
$$H = [\![Fr(\beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x))]\!]_M$$

Then $(M, H) \xrightarrow{\text{free}(x)} (M', H'), M' \models \beta$, *and* $H' = [\![Fr(\beta)]\!]_{M'}$

Proof. Observe that $x \in Fr(\beta \wedge x \notin Fr\beta) \wedge f(x) = f(x))$, i.e., $[\![x]\!]_M \in H$. Therefore we have from the operational semantics that $(M, H) \xrightarrow{\text{free}(x)} (M', H')$ such that $M' = M$ and $H' = H \setminus \{[\![x]\!]_M\}$. Since $M \models \beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x)$, we know $M \models \beta$, which implies $M' \models \beta$. Similarly, we have:

$$\begin{aligned}
H' &= H \setminus \{[\![x]\!]_M\} &\text{(operational semantics)}\\
&= [\![Fr(\beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x))]\!]_M \setminus \{[\![x]\!]_M\}\\
&= [\![Fr(\beta)]\!]_M \cup \{[\![x]\!]_M\} \setminus \{[\![x]\!]_M\} &\text{(def of } Fr\text{)}\\
&= [\![Fr(\beta)]\!]_M\\
&= [\![Fr(\beta)]\!]_{M'} &(M = M')
\end{aligned}$$
$\qquad\square$

Theorem 13 (WTP Deallocation)**.** *Let M, M' be models with H, H' sub-universes of locations (respectively) such that $(M, H) \xrightarrow{\text{free}(x)} (M', H')$, $M' \models \beta$ and $H' = [\![Fr(\beta)]\!]_{M'}$. Then*

$$M \models \beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x) \quad \text{(weakest-pre)}$$
$$H = [\![Fr(\beta \wedge x \notin Fr(\beta) \wedge f(x) = f(x))]\!]_M$$
$$\text{(tightest-pre)}$$

Proof. For the first part, note that the operational semantices ensures $[\![x]\!]_{M'} \notin H' = [\![Fr(\beta)]\!]_{M'}$ and $M = M'$. So $M' \models x \notin Fr(\beta)$ which implies $M \models x \notin Fr(\beta)$. Similarly, $M \models \beta$, and $M \models f(x) = f(x)$ as it is a tautology. Tightness follows from similar arguments as in Theorem 12, again noting that $H' = H \setminus \{[\![x]\!]_M\}$ as per the operational semantics. $\qquad\square$

Theorem 2. *The four local rules (for assignment, lookup, mutation, allocation, and deallocation) given in Section IV are sound given the global rules.*

Proof. The validity of assignment follows immediately setting β to be $x = y$ (or $x = c$). Instantiating with this and the precondition becomes $y = y$ which is equivalent to $true$ (the heaplet of both is empty)

The validity of the next (lookup) follows since

$$\begin{aligned}
&wtp(x = f(y), x := y.f)\\
&= \exists x' : x' = f(y).\\
&\quad (x = f(y) \wedge y \in Fr(x = f(y)))[x'/x]\\
&= \exists x' : x' = f(y). \, x' = f(y) \wedge y \in Fr(x' = f(y))\\
&= f(y) = f(y) \wedge y \in Fr(f(y) = f(y))
\end{aligned}$$

This is a tautology, so it is clearly implied by any precondition, in particular the precondition $f(y) = f(y)$. Similarly, the frame of the resulting formula is the singleton $\{y\}$ which is also the frame of $f(y) = f(y)$ as needed.

For the second local rule (mutation), we first notice that

$$\begin{aligned}
&MW^{x.f := y}(f(x) = y)\\
&= (f(x) = y)\\
&\quad [ite(z = x : ite(f(x) = f(x) : y, y), f(z))/f(x)]\\
&= ite(x = x : ite(f(x) = f(x) : y, y), f(x)) = y
\end{aligned}$$

Then,

$$\begin{aligned}
&wtp(f(x) = y, x.f := y)\\
&= ite(x = x : ite(f(x) = f(x) : y, y), f(x)) = y\\
&\wedge x \in Fr(ite(x = x :\\
&\quad ite(f(x) = f(x) : y, y), f(x)) = y)
\end{aligned}$$

The first conjunct is clearly true since it is equivalent to $y = y$. The second conjunct is also true because $Fr(ite(x = x : ite(f(x) = f(x) : y, y), f(x)) = y) = \{x\}$. Thus, this formula is also a tautology, and it is implied by the precondition $f(x) = f(x)$. Additionally the frame of the resulting formula and the frame of $f(x) = f(x)$ is $\{x\}$ as needed.

For the next local rule (allocation), observe that the postcondition does not have any frame expressions or inductive definitions. Therefore, we have that:

$$\begin{aligned}
&MW^{\text{alloc}(x)}_v(f(x) = def_f)\\
&= ite(x = x : def_f, f(x)) = def_f
\end{aligned}$$

Observe that the frame of the above expression is $\emptyset$. The frame of a conjunction of such expressions is also $\emptyset$. This and the fact that $MW^{\text{alloc}(x)}_v$ distributes over $\wedge$ gives us:

$$\begin{aligned}
&wtp\left(\bigwedge_{f \in F}(f(x) = def_f), x := alloc()\right)\\
&= \forall v : v \notin \emptyset \implies MW^{\text{alloc}(x)}_v\left(\bigwedge_{f \in F} f(x) = def_f\right)\\
&= \forall v : \bigwedge_{f \in F}\left(ite(x = x : def_f, f(x)) = def_f\right)
\end{aligned}$$

which is a tautology (as it is equivalent to $def_f = def_f$) and its frame is $\emptyset$ as desired.

Finally the last local rule (deallocation) follows directly from the global rule for deallocation by setting $\beta = true$. $\square$

Theorem 14 (Conditional, While Soundness)**.**

Proof. See any classical proof of the soundness of these rules, as in [1]. $\square$

Theorem 15 (Sequence Soundness)**.** *The Sequence rule is sound.*

Proof. Follows directly from the operational semantics. $\square$

Theorem 16 (Consequence Soundness)**.** *The Consequence rule is sound.*

Proof. First, note if we can't execute S then the triple is vacuously valid. Next, assume $M \models \alpha'$. Then, because $\alpha' \implies \alpha$, we know $M \models \alpha$. So, if we execute S and result in M', we know $M' \models \beta$ since $\{\alpha\}S\{\beta\}$ is a valid triple. Then, $M' \models \beta'$ since $\beta \implies \beta'$. Finally, since the frames of α and α' as well as β and β' are equal, the validity of the Hoare triple holds. $\square$

Theorem 17 (Frame Rule Soundness)**.** *The Frame rule is sound.*

Proof. First, we establish that for any (M, H) such that $M \models \alpha \wedge \gamma$ and $H = [\![Fr(\alpha \wedge \gamma)]\!]_M$ we never reach $\bot$. Consider $(M, [\![Fr(\alpha)]\!]_M) \overset{S}{\Rightarrow}^*$ and $(M, H) \overset{S}{\Rightarrow}^*$. Let these executions be expressed as a sequence of configurations P_1 and P_2. We can show that for each step in P_2, there exists a corresponding step in P_1 such that:

1) at any corresponding step the allocated set on P_2 is a superset of the allocated set on P_1
2) the executions allocate and deallocate the same locations

The claim as well as the first item is easy to show by structural induction on the program. Given that, the second is trivial since a location available to allocate on P_2 is also available to allocate on P_1. Any location that is deallocated on P_2 that is unavailable on P_1 would cause P_1 to reach $\bot$ which is disallowed since we are given that $\{\alpha\}S\{\beta\}$ is valid.

Thus if we abort on the former we must abort on the latter, which is a contradiction since we are given that $\{\alpha\}S\{\beta\}$ is valid. From the second item above, we can also establish that all mutations of the model are outside of $Fr(\gamma)$ since it is unavailable on P_1 (we start with $Fr(\alpha)$ and allocate only outside $Fr(\alpha \wedge \gamma) = Fr(\alpha) \cup Fr(\gamma)$, and we are also given that the frames of α and γ are disjoint in any model). Therefore, if there exists a configuration (M', H') such that $(M, H) \overset{S}{\Rightarrow}^* (M', H')$

it must be the case that M' is a mutation of M that is stable on $Fr(\gamma)$. Since $\{\alpha\}S\{\beta\}$ is valid we have that $M' \models \beta$. Lastly, we conclude from the Frame Theorem (Theorem 1) that since $M \models \gamma$, $M' \models \gamma$ which gives us $M' \models \beta \wedge \gamma$.

We must also show that $H' = [\![Fr(\beta \wedge \gamma)]\!]_{M'}$. To show this, we can strengthen the inductive invariant above with the fact that at any corresponding step the allocated set on P_2 is not simply a superset of that on P_1, but in fact differs exactly by $Fr(\gamma)$. This invariant establishes the desired claim, which concludes the proof of the frame rule. $\square$

G. Frame Logic Can Capture the PSL fragment

Lemma 6. *For any formula φ in the PSL fragment, if there is an s and h such that $s, h \models \varphi$ and we can extend h by some nonempty h' such that $s, h \cup h' \models \varphi$, then for any h'', $s, h \cup h'' \models \varphi$.*

Proof. If a stack formula holds then it holds on any heap. Pointer formulas and inductive definitions as defined can never have an extensible heap so this is vacuously true.

For $ite(sf, \varphi_1, \varphi_2)$, assume WLOG $s, h \models sf$. Then for any h', $s, h' \models \varphi_1 \Leftrightarrow s, h' \models ite(sf, \varphi_1, \varphi_2)$. Then use the induction hypothesis.

For $\varphi_1 \wedge \varphi_2$, for any h', $s, h' \models \varphi_1 \wedge \varphi_2 \Leftrightarrow s, h' \models \varphi_1$ and $s, h' \models \varphi_2$. If the conjoined formula can be extended, both subformulas can be extended, and then we apply the induction hypothesis.

For separating conjunction, the nature of the proof is similar to conjunction, noting that the heap can be extended iff the heap of *either* subformula can be extended.

For existential formulas in our form, the proof is again similar, noting the heap is extensible iff the heap of φ_1 is extensible. $\square$

Theorem 4 (Minimal Heap)**.** *For any formula φ in the PSL fragment, if there is an s and h such that $s, h \models \varphi$ then there is a unique h_φ such that $s, h_\varphi \models \varphi$ and for all h' such that $s, h' \models \varphi$, $h_\varphi \subseteq h'$.*

Proof. The minimal heaplets for stack formulas are empty. For $x \xrightarrow{f} y$ the heaplet is uniquely $\{x\}$.

For conjunction, there are three cases depending on if φ_1 or φ_2 or both have extensible heaplets. We cover the most difficult case where they both have extensible heaplets here. By definition we know $s, h \models \varphi_1$ and $s, h \models \varphi_2$. By induction, we know there are unique h_{φ_1} and h_{φ_2} such that h_{φ_1} and h_{φ_2} model φ_1 and φ_2 respectively and are minimal. Thus, $h_{\varphi_1} \subseteq h$ and $h_{\varphi_2} \subseteq h$, so $h_{\varphi_1} \cup h_{\varphi_2} \subseteq h$. By Lemma 6, $h_{\varphi_1} \cup h_{\varphi_2}$ is a valid heap for both φ_1 and φ_2. Thus, $s, h_{\varphi_1} \cup h_{\varphi_2} \models \varphi_1 \wedge \varphi_2$ and $h_{\varphi_1} \cup h_{\varphi_2}$ is minimal.

For separating conjunction the minimal heaplet is (disjoint) union. For ite we pick the heaplet of either

case depending on the truth of the guard. By definition, inductive defintions will have minimal heaplets.

Inductive definitions have unique heaplets by the choice we made above and therefore vacuously satisfy the given statement.

For existentials, we know from the semantics of separation logic that every valid heap on a store s for the original existential formula is a valid heap for $\psi \equiv (x \xrightarrow{f} y) * \varphi_1$ on a modified store $s' \equiv s[y \mapsto v]$ for some v. Since the constraint $(x \xrightarrow{f} y)$ forces the value v to be unique, we can then invoke the induction hypothesis to conclude that the minimal heaplets of the existential formula on s and of ψ on s' are the same. In particular, this means that existential formulae in our fragment also have a minimal heaplet. $\square$

Lemma 7. *For any s, h such that $s, h \models \varphi$ we have $\mathcal{M}_{s,h}(Fr(T(\varphi))) = h_\varphi$ where h_φ is as above.*

Proof. Structural induction on φ.

If φ is a stack formula, $h_\varphi = Fr(T(\varphi)) = \emptyset$. If $\varphi \equiv x \xrightarrow{f} y$, $h_\varphi = Fr(T(\varphi)) = \{x\}$.

For $\varphi \equiv ite(sf, \varphi_1, \varphi_2)$, because, $s, h \models \varphi$, we know either $s, h \models \varphi_1$ or $s, h \models \varphi_2$ depending on the truth of sf. WLOG assume $s, h \models sf$, then $h_\varphi = h_{\varphi_1}$. Similarly, $Fr(T(\varphi)) = Fr(sf) \cup Fr(T(\varphi_1)) = Fr(T(\varphi_1))$ (heaplet of stack formulas is empty) and then we apply the induction hypothesis. Similarly if $s, h \not\models sf$.

For $\varphi \equiv \varphi_1 \wedge \varphi_2$, we know from the proof of Lemma 4 that $h_\varphi = h_{\varphi_1} \cup h_{\varphi_2} = \mathcal{M}_{s,h}(Fr(T(\varphi_1))) \cup \mathcal{M}_{s,h}(Fr(T(\varphi_2)))$. The guard parts of the translation $Fr(\varphi)$ since they are all precise formulas which have empty heaplets.

For $\varphi \equiv \varphi_1 * \varphi_2$, the proof is the same to the previous case, again from the proof of Lemma 4.

For an inductive definition I, recall that $\rho_I[I \leftharpoonup \varphi]$ is in the PSL fragment (and crucially does not mention I). Assume φ is fresh and does not occur in ρ_I. Define $\rho'_I \equiv \rho_I[I \leftharpoonup \varphi]$ and note that $\rho_I = \rho'_I[\varphi \leftharpoonup I]$. This means that $h_{\rho_I} = h_{\rho'_I}[h_\varphi \leftharpoonup h_I]$. We also see that $Fr(T(\rho_I)) = Fr(T(\rho'_I))[Fr(T(\varphi)) \leftharpoonup Fr(T(\rho_I))]$. Because $h_{\rho'_I} = Fr(T(\rho'_I))$ (by the other cases in this proof and since ρ'_I does not mention I), we see the heaplets are related by the same sets of recursive equations and we are done.

For existentials, we have from the definition of the *Fr* operator that the frame of the translation of the existential formula is the same as that of $\{x\} \cup Fr(T(\varphi_1))$. The claim then follows from the definition of heaplet of existentials in separation logic as well as the inductive hypothesis for φ_1. $\square$

Theorem 5. *For any formula φ in the PSL fragment, we*

have the following implications:

$$s, h \models \varphi \implies \mathcal{M}_{s,h} \models T(\varphi)$$
$$\mathcal{M}_{s,h} \models T(\varphi) \implies s, h' \models \varphi$$
$$\text{where } h' \equiv \mathcal{M}_{s,h}(Fr(T(\varphi)))$$

Here, $\mathcal{M}_{s,h}(Fr(T(\varphi)))$ is the interpretation of $Fr(T(\varphi))$ in the model $\mathcal{M}_{s,h}$. Note h' is minimal and is equal to h_φ as in Theorem 4.

Proof. First implication: Structural induction on φ.

If φ is a stack formula or a pointer formula, this is true by construction. If φ is an if-then-else formula the claim is true by construction and the induction hypothesis.

If $\varphi = \varphi_1 \wedge \varphi_2$, we know by the induction hypothesis that $\mathcal{M}_{s,h} \models T(\varphi_1)$ and $\mathcal{M}_{s,h} \models T(\varphi_2)$. Further, from the semantics of separation logic, we have that if φ_1 is precise, then $h_{\varphi_1} = h$. Therefore, $h_{\varphi_2} \subseteq h_{\varphi_1}$ (by Lemma 4). Therefore, from Lemma 7, we have that $\mathcal{M}_{s,h} \models Fr(T(\varphi_2)) \subseteq Fr(T(\varphi_1))$. Similarly if φ_2 is precise. This justifies the two latter conjuncts of the translation.

If $\varphi = \varphi_1 * \varphi_2$, we know there exist h_1, h_2 such that $h_1 \cap h_2 = \emptyset$ and $s, h_1 \models \varphi_1$ and $s, h_2 \models \varphi_2$. Then, from Lemma 4, we have that $h_{\varphi_1} \subseteq h_1$ and $h_{\varphi_2} \subseteq h_2$. Thus, by Lemma 7, we have that $\mathcal{M}_{s,h} \models Fr(T(\varphi_1)) \cap Fr(T(\varphi_2)) = \emptyset$. The other conjuncts follow from the induction hypothesis.

Similarly to the proof of Lemma 7, we can show that the translation of the inductive definition satisfies the same recursive equations as the original inductive definition and we are done.

If φ is an existential, the result follows from definition and the induction hypothesis.

Second implication: Structural induction on φ.

By construction, induction hypotheses, and Lemma 7, all cases can be discharged besides conjunction and inductive predicates.

For conjunction, if $\varphi = \varphi_1 \wedge \varphi_2$, we have from the induction hypothesis that $s, h_{\varphi_1} \models \varphi_1$ and $s, h_{\varphi_2} \models \varphi_2$. If φ_1 is precise, we know $\mathcal{M}_{s,h} \models Fr(T(\varphi_2)) \subseteq Fr(T(\varphi_1))$ and therefore $h_{\varphi_2} \subseteq h_{\varphi_1}$ (from Lemma 7). Similarly, if φ_2 is precise, then $\mathcal{M}_{s,h} \models Fr(T(\varphi_1)) \subseteq Fr(T(\varphi_2))$ as well as $h_{\varphi_1} \subseteq h_{\varphi_2}$. In particular, if they are both precise, their frames (and therefore minimal heaplets) are equal, and $h' = h_{\varphi_1} \cup h_{\varphi_2}$ (from the proof of Lemma 4) $= h_{\varphi_1} = h_{\varphi_2}$, and we are done. If only φ_1 is precise (similarly if only φ_2 is precise), then we have as above that $h_{\varphi_2} \subseteq h_{\varphi_1}$ and $h_{\varphi_1} = h'$. Moreover, we know by Lemma 6 that $s, h_{\varphi_1} \models \varphi_2$ and we are done. If neither is precise, both heaps are extensible, so we know by Lemma 6 that $s, h_{\varphi_1} \cup h_{\varphi_2} \models \varphi_1$ and $s, h_{\varphi_1} \cup h_{\varphi_2} \models \varphi_2$ and we are done.

For φ an inductive predicate, we know that $\mathcal{M}_{s,h} \big| Fr(T(\varphi)) \models T(\varphi)$. The remainder follows since,

because we restrict the form of inductive predicates to have a unique heap at each level, the translated inductive predicate will satisfy the same recursive equations as φ.

□