Translating Separation Logic into Dynamic Frames Using Fine-Grained Region Logic

Yuyan Bao, Gary T. Leavens, and Gidon Ernst

CS-TR-13-02a
March 2014

Keywords: Frame axiom, modifies clause, separation logic, dynamic frames, region logic, formal methods, Dafny language, DafnyR language.
2013 CR Categories: D.2.4 [*Software Engineering*] Software/Program Verification — Formal methods, programming by contract; F.3.1 [*Logics and Meanings of Programs*] Specifying and Verifying and Reasoning about Programs — Assertions, logics of programs, pre- and post-conditions, specification techniques;

Submitted for publication.

Computer Science
4000 Central Florida Blvd.
University of Central Florida
Orlando, Florida 32816, USA

Translating Separation Logic into Dynamic Frames Using Fine-Grained Region Logic

Yuyan Bao Gary T. Leavens
Computer Science, University of Central Florida, Orlando, FL 32816 USA
ybao@eecs.ucf.edu, leavens@eecs.ucf.edu

Gidon Ernst
Universität Augsburg, D-86135 Augsburg, Germany
ernst@informatik.uni-augsburg.de

Abstract

Several techniques have been proposed for specification and verification of frame conditions, making it difficult for specification language designers to know which to pick. Ideally there would be a single mechanism that could be used to express specifications written in all techniques. In this paper we provide a single mechanism that can be used to write specifications in the style of both separation logic and dynamic frames. This mechanism shows common characters between the two methodologies.

Categories and Subject Descriptors H.4 [*Information Systems Applications*]: Miscellaneous; D.2.4 [*Software Engineering*]: Software/Program Verification—formal methods, programming by contract

General Terms verification

Keywords Region logic, sequential programming, separation logic, dynamic frames, formal methods, frame axioms, DafnyR language

1. Introduction

In Hoare-style reasoning about sequential, imperative programs, framing is important for verification. A method's frame describes the locations that the method may not change [3]. Framing allows verification to carry properties past statements such as method calls, since properties about unchanged locations will remain valid.

Due to the importance of framing, many authors have focused on methodologies for specification of frame conditions and associated verification techniques

1.1 Separation Logic

Separation logic [8, 18] extends Hoare's logic with reasoning about locations on the heap. The separating conjunction, $P * Q$ denotes that assertions *P* and *Q* hold in separate parts of the heap. A binary tree could be defined in separation logic as follows:

$$tree(t) \stackrel{\text{def}}{=} (t = null \Rightarrow emp) * (t \neq null \Rightarrow t.val \mapsto \_ * tree(t.left) * tree(t.right)).$$

Separation logic is concise because its frame rule allows ignoring separated parts of the heap during reasoning; for example one can ignore the right subtree when reasoning about the left subtree. Therefore, separation logic simplifies reasoning about data structures that consist of isolated substructures, such as acyclic linked lists and binary trees.

On the other hand, one cannot use separation when there is sharing, as in a directed acyclic graph (DAG), where the left and right sides of a DAG may share nodes. Specifying sharing and framing for shared parts of a data structure is challenging and tricky [21], and may need the ramification operator [7].

1.2 Dynamic Frames and Region Logic

Unlike separation logic, *dynamic frames theory* [9, 10] uses regions that are (conceptually) stored in variables to specify frame properties. It defines a *region* as a set of locations. Regions are represented by specification-only variables that vary as a program's state changes. Dafny [13–15] and region logic [1] are two approaches that apply dynamic frames theory.

1.2.1 The idea of Dynamic Frames

Fig. 1 shows code snippets specifying a linked-list program written in Dafny [13–15]. Dafny uses **`modifies`** and **`reads`** clauses to specify frame properties. The dynamic frame is specified by the ghost field `footprint`. It stores a set of object references including **`this`** and its successors in the list. This property is defined in the function `Valid`. `Valid` serves as an invariant that must be satisfied once a `Node` object is created.

```
1  class Node<T> {
2   var list: seq<T>;
3   var footprint: set<Node<T>>;
4   var data: T;
5   var next: Node<T>;
6   // constructor  and other methods omitted
7   function Valid(): bool
8     reads this, footprint;
9   {
10    this in this.footprint && null !in this.footprint &&
11    (next == null ==> list == [data]) &&
12    (next != null ==>
13        next in footprint &&
14        next.footprint <= footprint &&
15        this !in next.footprint &&
16        list == [data] + next.list &&
17        next.Valid())
18  }
19   method Prepend(d: T) returns (r: Node<T>)
20     requires Valid();
21     ensures r != null && r.Valid() &&
22       fresh(r.footprint - old(footprint));
23     ensures r.list == [d] + list;
24   {
25     r := new Node<T>;
26     r.data := d;
27     r.next := this;
28     r.footprint := {r} + this.footprint;
29     r.list := [r.data] + this.list;
30   }
31 }
```

Figure 1. A linked-list code snippet specifying in the styling of dynamic frame. Code snippets are from the Dafny repository [12].

Compared to separation logic's assertions that couple locations and their content, the dynamic frames technique allows one to specify heap locations independently. For example, the `Prepend` method ensures that one adds the new frame (new node) to the ghost variable `footprint`. Although this increases coding tasks, properties can be freely specified by set operations and first-order expressions. Set membership and disjointness can specify isolated structures. For example, line 16 in Fig. 1 requires that `this` can not be in its successor's footprint, which guarantees acyclicity. Moreover, first-order expressions can easily describe arbitrary sharing properties. In the example of specifying Schorr-Waite algorithm shown in Fig. 2, line 11 to 13 specifies a graph with random sharing. This flexibility is convenient for use in verifying a traversal algorithm, where the exact shape is not of concern. Furthermore, in the example of binary trees with sharing defined in Fig. 3, because each subtree stores its own children, the frame of part of unvisited right subtrees can be calculated by $right.ftp - left.ftp$, which is non-trivial in separation logic.

1.3 Motivation

As we have shown, each approach has its own advantages and disadvantages. A single mechanism that can be used to write specification in the style of both approaches can make best use of each one's advantages.

Region logic [1] was conceived as a way to write specifications that mimic those in separation logic, but which only

```
1  class Node {
2   var children: seq<Node>;
3   var marked: bool;
4   var childrenVisited: int; // other fields omitted...
5  }
6  method RecursiveMark(root: Node, ghost S: set<Node>)
7   requires root in S;
8   // S is closed under 'children':
9   requires (forall n :: n in S ==> n != null &&
10            (forall ch :: ch in n.children ==>
11                ch == null || ch in S));
12  requires (forall n :: n in S ==> ! n.marked &&
13             n.childrenVisited == 0);
14  modifies S;
15  ensures root.marked;
16  // nodes reachable from 'root' are marked:
17  ensures (forall n :: n in S && n.marked ==>
18            (forall ch :: ch in n.children &&
19                ch != null ==> ch.marked));
20  ensures (forall n :: n in S ==>
21            n.childrenVisited==old(n.childrenVisited) &&
22            n.children == old(n.children));
23  { /* ... */ }
```

Figure 2. Method specification of Schorr-Waite algorithm in Dafny from its repository [12].

```
class Node {
  var left: Node;    var right: Node;
  var marked: bool;  var ftp: set<Node>;
  predicate Dag()  reads this, ftp;
  {
    null !in ftp && this in ftp &&
    (this.left==null && this.right==null==>
          ftp=={this}) &&
    (this.left!=null && this.right==null==>
          this.left in ftp-{this} &&
          this.left.ftp==ftp-{this} &&
          this.left.Dag()) &&
    (this.left==null && this.right!= null==>
          this.right in ftp-{this} &&
          this.right.ftp == ftp-{this} &&
          this.right.Dag()) &&
    (this.left!=null && this.right!=null ==>
          (this.left!=this.right ==>
          {this.left}+{this.right} == ftp-{this} &&
          (this.left.ftp+this.right.ftp)==ftp-{this} &&
          this.left.Dag() && this.right.Dag()) &&
    (this.left == this.right ==>
          {this.left}+{this.right}==ftp-{this} &&
          this.left.ftp==this.right.ftp &&
          this.left.ftp==ftp-{this} &&
          this.left.Dag() && this.right.Dag()))
} }
```

Figure 3. A DAG code snippet written in Dafny.

require first-order theorem-proving. Since regions are also used in the dynamic frames technique, we believe that regions are a mechanism into which one can translate both separation logic and dynamic frame style specifications. This idea largely works, but for simplicity and better algebraic properties of the region logic operators, we changed the definition of regions to match that used in the dynamic frames theory: sets of locations. We call the result a "fine-grained" region logic. Using sets of locations is also a good match for specification languages such as JML [5, 11].

Separation logic [8, 18] eliminates frame conditions, but requires one to implicitly request access to locations in a method's precondition. Intuitively, we could simply take these locations as the frame condition in dynamic frame specifications. Thus it seems that Dafny [13–15] could be used to simulate separation logic. However, consider the separation logic assertion $(x.f \mapsto v) * (x'.f' \mapsto v')$. The locations named are $\{(x.f), (x'.f')\}$, which are represented by a set of pairs of an object and a field name. But Dafny uses a set of object references to specify frame properties, and those objects need to have a single type. Thus using sets of objects in Dafny is not the best way to encode separation logic. Region logic [1] allows one to specify frame properties at the granularity of an object's fields. However, its region type still represents a set of objects. Region union on sets of locations is not defined. That is a hindrance for computing locations for framing from separation logic assertions.

We consider all allocated memories as a heap, H. Although the frame condition in the dynamic frames technique provides a set of locations that may be changed in a method, the dynamic frames technique does not restrict the subset of $dom(H)$ that programs can access. In separation logic, reasoning about a method is restricted to the part of the heap that is specified in its precondition. Our general approach is to use the footprint of method preconditions from separation logic specifications to obtain a partial heap h such that $dom(h) \subseteq dom(H)$.

1.4 Contributions

The contributions of this paper are as follows:

- We introduce a fine-grained region logic. This fine-grained region logic is used in a variant of Dafny, DafnyR. It allows one to directly translate separation logic's points-to assertions into frame axioms. Our implementation of DafnyR is available from `http://dafnyr.codeplex.com/`.
- We introduce an if-then-else region expression that allows region expressions to more precisely match the footprint of assertions.
- We show how to translate a restricted separation logic into DafnyR in a way that preserves the meaning of assertions.
- We show how to translate proofs of correctness in separation logic into proofs in DafnyR's logic, and show that provability is preserved.

1.5 Overview of the results

In the next section, we present our language, DafnyR. Section 3 introduces a restricted separation logic that we encode into DafnyR. Section 4 shows the translation from the restricted separation logic to DafnyR, and proves the semantics meaning is preserved in the translation. Section 5 discusses the encoding of overlapping conjunction, which is an extension of separation logic, and the backward translation, from DafnyR to separation logic. Section 6 describes related work. Section 7 gives conclusions and future work.

2. The DafnyR Language

DafnyR uses a version of *region logic* in a variant of Dafny [13–15]. To simplify our presentation, we only use a subset of DafnyR's syntax. In particular, we do not allow recursive predicates.

2.1 Syntax of DafnyR

DafnyR adds region expressions to Dafny. Fine-grained regions not only allow us to define the built-in predicate $PointsTo_f$ as later explained, they also allow us to define operations, such as union, on these fine-grained regions. In particular, the conditional region expression (if) allows us to syntactically represent regions that can only be determined dynamically. An assertion of the form $P(ins)$ invokes the predicate P with argument list *ins*.

DEFINITION 2.1 (DafnyR Syntax). *The syntax of* DafnyR *assertions, expressions, and statements is as follows:*

$$
\begin{array}{lll}
\mathit{Assrt} & ::= & \mathit{Expr}_1 \ = \ \mathit{Expr}_2 \ \mid \ \mathit{Assrt}_1 \ \texttt{\&\&} \ \mathit{Assrt}_2 \\
 & \mid & \mathit{Assrt}_1 \ \text{`}\texttt{||}\text{'} \ \mathit{Assrt}_2 \ \mid \ \mathit{Assrt}_1 \ \Rightarrow \ \mathit{Assrt}_2 \\
 & \mid & \exists \ x\,\texttt{.}\mathit{Assrt} \ \mid \ P\,\texttt{(}\mathit{ins}\texttt{)} \ \mid \ \mathit{REAssrt} \\
\mathit{Expr} & ::= & x \ \mid \ \textbf{null} \ \mid \ n \ \mid \ x\,\texttt{.}f \ \mid \ \mathit{RE} \\
\mathit{ins} & ::= & \mathit{Empty} \ \mid \ \mathit{ExprList} \\
\mathit{Empty} & ::= & \\
\mathit{ExprList} & ::= & \mathit{ExprList}\texttt{,} \ \mathit{Expr} \ \mid \ \mathit{Expr} \\
\mathit{RE} & ::= & \textbf{alloc} \ \mid \ \textbf{region}\{\} \ \mid \ \textbf{region}\{\mathit{Expr}\,\texttt{.}f\} \\
 & \mid & \textbf{fpt}\,\texttt{(}\mathit{Expr}\texttt{)} \ \mid \ \textbf{fpt}\,\texttt{(}\mathit{Assrt}\texttt{)} \\
 & \mid & \mathit{RE}_1 \ + \ \mathit{RE}_2 \ \mid \ \mathit{RE}_1 \ * \ \mathit{RE}_2 \\
 & \mid & \textbf{if} \ \mathit{Assrt} \ \textbf{then} \ \mathit{RE}_1 \ \textbf{else} \ \mathit{RE}_2 \\
\mathit{REAssrt} & ::= & \mathit{RE}_1 \ \texttt{!!} \ \mathit{RE}_2 \ \mid \ \mathit{RE}_1 \ \texttt{<=} \ \mathit{RE}_2 \\
\mathit{Stmt} & ::= & x \ \texttt{:=} \ \mathit{Expr} \ \mid \ x\,\texttt{.}f \ \texttt{:=} \ \mathit{Expr} \ \mid \ x_1 \ \texttt{:=} \ x_2\,\texttt{.}f \\
 & \mid & x \ \texttt{:=} \ \textbf{new} \ K \\
 & \mid & \textbf{if}\,\texttt{(}\mathit{Expr}{\neq}\texttt{0)}\,\textbf{then}\{\mathit{Stmt}_1\}\textbf{else}\{\mathit{Stmt}_2\} \\
 & \mid & \textbf{while}\,\texttt{(}\mathit{Expr}{\neq}\texttt{0)}\,\{\mathit{Stmt}\} \ \mid \ \mathit{Stmt}_1\texttt{;}\,\mathit{Stmt}_2
\end{array}
$$

where $x \in Id$ is an identifier, n is a numeric literal, and f is a field name.

We define other logical operators and predicates as follows: $true \equiv (0 = 0)$, $false \equiv (0 = 1)$, $\neg Assrt \equiv (Assrt \Rightarrow false)$, $e \neq e' \equiv \neg(e = e')$, and $\forall x.Assrt \equiv \neg(\exists x.\neg Assrt)$.

For convenience in encoding separation logic's points-to assertions, we assume that, for each field f of type S in each class T, there is a built-in predicate $PointsTo_f$ defined as:

```
predicate PointsTo_f(o: T, v: S)
reads region{o.f}; {  o ≠ null && o.f = v }
```

We define Γ as a type environment that maps variables to types:

$$\Gamma \in TypeEnv = Id \to Type$$

$$\Gamma \vdash x : T \textbf{ where } (\Gamma\, x) = T \qquad \Gamma \vdash \texttt{null} : K \textbf{ where } isClass(K) \qquad \Gamma \vdash n : \texttt{int}$$

$$\frac{\Gamma \vdash x : K}{\Gamma \vdash x.f : T} \begin{array}{l}\textbf{where } isClass(K)\\ \textbf{and } (f : T) \in fields(K)\end{array} \qquad \Gamma \vdash \texttt{alloc} : \texttt{region} \qquad \Gamma \vdash \texttt{region}\{\} : \texttt{region}$$

$$\frac{\Gamma \vdash \mathit{Expr} : K}{\Gamma \vdash \texttt{region}\{\mathit{Expr}.f\} : \texttt{region}} \begin{array}{l}\textbf{where } isClass(K)\\ \textbf{and } (f : T) \in fields(K)\end{array} \qquad \frac{\Gamma \vdash \mathit{Expr} : T}{\Gamma \vdash \texttt{fpt}(\mathit{Expr}) : \texttt{region}}$$

$$\frac{\Gamma \vdash \mathit{Expr}_1 : \texttt{region}, \qquad \Gamma \vdash \mathit{Expr}_2 : \texttt{region}}{\Gamma \vdash \mathit{Expr}_1\ O\ \mathit{Expr}_2 : \texttt{region}} \textbf{ where } O \in \{+, *\} \qquad \frac{\Gamma \vdash Assrt : \texttt{bool}}{\Gamma \vdash \texttt{fpt}(Assrt) : \texttt{region}}$$

$$\frac{\Gamma \vdash Assrt : \texttt{bool}, \qquad \Gamma \vdash RE_1 : \texttt{region}, \qquad \Gamma \vdash RE_2 : \texttt{region}}{\Gamma \vdash \texttt{if } Assrt \texttt{ then } RE_1 \texttt{ else } RE_2 : \texttt{region}}$$

Figure 4. Typing rules for DafnyR expressions. The predicate $isClass$ returns true just when K is either `object` or a declared class name in the program. The auxiliary function $fields$ takes a class name and returns a list of its declared field names and their types.

$$\frac{\Gamma \vdash \mathit{Expr}_1 : T, \qquad \Gamma \vdash \mathit{Expr}_2 : T}{\Gamma \vdash \mathit{Expr}_1 = \mathit{Expr}_2 : \texttt{bool}} \qquad \frac{\Gamma \vdash Assrt_1 : \texttt{bool}, \qquad \Gamma \vdash Assrt_2 : \texttt{bool}}{\Gamma \vdash Assrt_1\ O\ Assrt_2 : \texttt{bool}} \textbf{ where } O \in \{\&\&, ||, \Rightarrow\}$$

$$\frac{\Gamma, (x : T) \vdash Assrt : \texttt{bool}}{\Gamma \vdash \exists x.Assrt : \texttt{bool}} \qquad \frac{\Gamma \vdash RE_1 : \texttt{region}, \qquad \Gamma \vdash RE_2 : \texttt{region}}{\Gamma \vdash RE_1\ O\ RE_2 : \texttt{bool}} \textbf{ where } O \in \{!\,!, <=\}$$

$$\frac{\Gamma \vdash \mathit{Expr}_1 : T_1, \qquad \ldots, \qquad \Gamma \vdash \mathit{Expr}_n : T_n}{\Gamma \vdash P(ins) : \texttt{bool}} \begin{array}{l}\textbf{where } (x_1 : T_1, \ldots, x_n : T_n) = \mathit{formalTypes}(P),\ n \geqslant 0,\\ \textbf{and } (\mathit{Expr}_1, \ldots, \mathit{Expr}_n) = ins\end{array}$$

Figure 5. Typing rules for DafnyR Assertions. The auxiliary function *formalTypes* takes a predicate name and returns the list of its declared formal parameters and their types.

$$\frac{\Gamma \vdash x : T,\ \Gamma \vdash \mathit{Expr} : T}{\Gamma \vdash x\ :=\ \mathit{Expr} : ok(\Gamma)} \qquad \frac{\Gamma \vdash x.f : T,\ \Gamma \vdash \mathit{Expr} : T}{\Gamma \vdash x.f\ :=\ \mathit{Expr} : ok(\Gamma)} \qquad \frac{\Gamma \vdash x : T,\ \Gamma \vdash x'.f : T}{\Gamma \vdash x\ :=\ x'.f : ok(\Gamma)} \qquad \frac{\Gamma \vdash x : K,\ \Gamma \vdash \texttt{new}\ K : K}{\Gamma \vdash x\ :=\ \texttt{new}\ K; : ok(\Gamma)}$$

$$\frac{\Gamma \vdash \mathit{Expr}_1 \neq 0 : \texttt{bool},\ \Gamma \vdash \mathit{Stmt}_1 : ok(\Gamma_1),\ \Gamma \vdash \mathit{Stmt}_2 : ok(\Gamma_2)}{\Gamma \vdash \texttt{if}\ (\mathit{Expr} \neq 0)\ \{\mathit{Stmt}_1\}\texttt{else}\ \{\mathit{Stmt}_2\} : ok(\Gamma)} \qquad \frac{\Gamma \vdash \mathit{Expr} \neq 0 : \texttt{bool},\ \Gamma \vdash \mathit{Stmt} : ok(\Gamma')}{\Gamma \vdash \texttt{while}\ (\mathit{Expr} \neq 0)\ \{\mathit{Stmt}\} : ok(\Gamma)}$$

$$\frac{\Gamma \vdash \mathit{Stmt}_1 : ok(\Gamma''),\ \Gamma'' \vdash \mathit{Stmt}_2 : ok(\Gamma')}{\Gamma \vdash \mathit{Stmt}_1; \mathit{Stmt}_2 : ok(\Gamma')}$$

Figure 6. Typing rules for DafnyR statements.

The typing rules for expressions are defined in Fig. 4, the typing rules for assertions are defined in Fig. 5, the typing rules for statements are defined in Fig. 6.

2.2 Semantics of DafnyR

We now present a semantics of DafnyR expressions and assertions. We introduce a set *Loc*, which represents locations in a heap as pairs of object references and field names. We use a store σ, which is a partial function that maps a variable to its value, and a heap H, which maps from an object reference and a field name to that location's value. A *Value* is either a Boolean, an object reference (which may be $null$), an integer, or a set of locations.

$$\begin{aligned} \textit{Value} = \quad & \textit{Boolean} + \textit{Object} + \textit{Int} + PowerSet(\textit{Loc}) + \\ & \{Error\} \end{aligned}$$

Definition of heap

Heaps (H) are finite maps from Loc to values. Heaps are manipulated using the following operations.

DEFINITION 2.2 (Heap Operations). Lookup *in a heap, written* $H[o, f]$, *is defined when* $(o, f) \in dom(H)$. $H[o, f]$ *is the value that* H *associates to* (o, f).

H_2 extends H_1, *written* $H_1 \subseteq H_2$, *means:*
$\forall(o, f) \in dom(H_1) : (o, f) \in dom(H_2) : H_1[o, f] = H_2[o, f]$.

H_1 is disjoint from H_2, *written* $H_1 \perp H_2$, *means* $dom(H_1) \cap dom(H_2) = \varnothing$.

The combination *of two partial heaps written* $H_1 \cdot H_2$, *is defined when* $H_1 \perp H_2$ *holds, and is the partial heap such that:* $dom(H_1 \cdot H_2) = dom(H_1) \cup dom(H_2)$, *and for all* $(o, f) \in dom(H_1 \cdot H_2)$:

$$(H_1 \cdot H_2)[o, f] = \begin{cases} H_1[o, f], & \textit{if } (o, f) \in dom(H_1), \\ H_2[o, f], & \textit{if } (o, f) \in dom(H_2). \end{cases}$$

2.3 Footprints

2.3.1 Semantic Footprints

Semantically, a footprint is the smallest set of (heap) locations on which the value of an expression or assertion depends. The notion of dependency is formalized by considering the evaluation in two heaps, and finding what locations the heaps must agree on to result in the same value.

DEFINITION 2.3 (Agree on Locations). *Let* H_1 *and* H_2 *be two heaps and let Loc be a set of locations (i.e., of pairs of object references and fields). Two heaps,* H_1 *and* H_2, agree on *Loc, written* $H_1 \overset{\text{Loc}}{\equiv} H_2$ *when* $\forall(o, f) \in Loc :: ((o, f) \in dom(H_1) \cap dom(H_2)) \wedge H_1[o, f] = H_2[o, f]$.

A semantic footprint is the minimal set of locations necessary to evaluate an expression or assertion in a given state. That is, changing the value that the state associates to a location outside the footprint will not change the value of the expression or assertion.

DEFINITION 2.4 (Semantic Footprint of Expressions). *Let Expr be an expression, and* (σ, H) *be a state. Let* $\mathcal{E}$ *be the expression evaluation function. Let* F *be a set of locations. Then* F is the semantic footprint of *Expr* in the state (σ, H) *if and only if:*

1. $\forall H' :: H \overset{\text{F}}{\equiv} H' \Rightarrow (\mathcal{E}[\![Expr]\!]_{\sigma,H} = \mathcal{E}[\![Expr]\!]_{\sigma,H'})$, *and*
2. $(\nexists F' : F' \subset F :$
   $(\forall H' :: H \overset{\text{F}'}{\equiv} H' \Rightarrow (\mathcal{E}[\![Expr]\!]_{\sigma,H} = \mathcal{E}[\![Expr]\!]_{\sigma,H'})))$.

DEFINITION 2.5 (Semantic Footprint of Assertions). *Let an assertion* $Assrt$, *and a state* (σ, H) *be given. Let* F *be a set of locations. Then* F is the semantic footprint of $Assrt$ in the state (σ, H) *if and only if:*

1. $(\forall H' :: H \overset{\text{F}}{\equiv} H' \Rightarrow$
   $(\sigma, H \models Assrt \iff \sigma, H' \models Assrt))$, *and*
2. $(\nexists F' : F' \subset F :$
   $(\forall H' :: H \overset{\text{F}'}{\equiv} H' \Rightarrow$
   $(\sigma, H \models Assrt \iff \sigma, H' \models Assrt)))$.

To illustrate this definition, consider an implication assertion, $Assrt_1 \Rightarrow Assrt_2$. A program evaluates $Assrt_1$ first by accessing a set of locations, Loc_1. If it is true, $Assrt_2$ is evaluated by accessing a set of locations, Loc_2, otherwise $Assrt_2$ is skipped. Therefore, if the assertion is true in a given state, then the semantic footprint of this implication assertion in that state is $Loc_1 \cup Loc_2$, otherwise it is just Loc_1.

2.3.2 Syntactic representation for footprint

Now we consider a way to statically determine a syntactic representation of the semantic footprint of an assertion.

Naive approaches to obtaining such a syntactic representation can be very imprecise and do not necessarily reflect the meaning of separation logic assertions. For example, consider the assertion: $((b \neq 0) \Rightarrow x.f \mapsto 0) * ((b = 0) \Rightarrow y.f \mapsto 0)$. According to the semantics of separating conjunction, there must be two disjoint heaps, h_1 and h_2, where $(b \neq 0) \Rightarrow x.f \mapsto 0$ and $(b = 0) \Rightarrow y.f \mapsto 0$ are valid, respectively. This assertion depends on the variable b, and thus the assertion neither requires nor prohibits x and y from being aliases. However, a naive syntactic computation of footprints might prohibit x and y from being aliased (if it required that $dom(h_1) = \{(x, f)\}$ and $dom(h_2) = \{(y, f)\}$). Therefore, we need a representation that respects the way assertions (and expressions) are evaluated. For this reason, we added the conditional region expression (if) to DafnyR.

2.3.3 Semantics of DanyR

We now show the semantics of DafnyR's expressions and assertions and show that DanfyR's built-in function **`fpt`** computes a footprint that is equal to the semantic footprint in every state.

In the following semantics, $\mathcal{E}_{DR}$ gives the denotation of an expression, $\mathcal{RE}$ gives the denotation of a region expression, and $\mathcal{A}_{DR}$ gives the Boolean denotation of an assertion. The built-in footprint function **fpt** syntactically maps expressions and assertions to region expressions.

Region expressions, RE (Definition 2.1), are used to manipulate regions; they denote sets of locations. We consider region expressions and the + operator to form a commutative monoid with unit element **region**{}, which denotes the empty region. The region expression **alloc** denotes the domain of the heap, which is all the allocated locations. The region expression **region**$\{Expr.f\}$ denotes a set containing the location of field f in the object that is the value of *Expr* (if *Expr* is not null), and all locations needed to evaluate *Expr*. Operators $+$, $*$, `!!`, and `<=` are set notations, denoting union, intersection, disjointness and subset of regions respectively. For example, RE_1 `!!` RE_2 is true just when the regions RE_1 and RE_2 are disjoint.

The region expression, **if** $Assrt$ **then** RE_1 **else** RE_2, denotes that when the $Assrt$ is true, the region is the meaning of RE_1, otherwise, it is the meaning of RE_2. Note that the **fpt** function is not symmetric with respect to conjunction, disjunction and separating conjunction. For instance, $\mathbf{fpt}(Assrt_1 \,\&\&\, Assrt_2)$ does not necessarily equal $\mathbf{fpt}(Assrt_2 \,\&\&\, Assrt_1)$. Instead, the **fpt** function follows Dafny's left-to-right evaluation order [13]. For example, when checking the assertion $o \neq null \,\&\&\, o.f = 5$, the sub-expression $o \neq null$ is evaluated first.

DEFINITION 2.6 (Semantics of Expressions and Assertions). *Let a fixed set of predicate declarations for a program be given. The meaning of expressions in DafnyR is given by the following, where $\mathcal{N}$ is the standard meaning function for numeric literals.*

$$
\begin{aligned}
&\mathcal{E}_{DR} : Expr \to Store \times Heap \to Value\\
&\mathcal{E}_{DR}[\![x]\!]_{\sigma,H} = \sigma(x) \qquad \mathcal{E}_{DR}[\![\mathbf{null}]\!]_{\sigma,H} = null\\
&\mathcal{E}_{DR}[\![n]\!]_{\sigma,H} = \mathcal{N}[\![n]\!] \quad \mathcal{E}_{DR}[\![RE]\!]_{\sigma,H} = \mathcal{RE}[\![RE]\!]_{\sigma,H}\\
&\mathcal{E}_{DR}[\![x.f]\!]_{\sigma,H} = H[\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f]
\end{aligned}
$$

The semantics of region expressions, $\mathcal{RE}[\![-]\!]_{\sigma,H}$, is shown in Fig. 8.

The semantics of assertions, $\mathcal{A}_{DR}[\![-]\!]_{\sigma,H}$ is defined by:

$$
\begin{aligned}
&\mathcal{A}_{DR} : Assrt \to Store \times Heap \to Boolean\\
&\mathcal{A}_{DR}[\![Assrt]\!]_{\sigma,H} = \begin{cases} true, & \text{if } \sigma, H \models_{DR} Assrt\\ false, & \text{if } \sigma, H \not\models_{DR} Assrt \end{cases}
\end{aligned}
$$

The validity of assertions in DafnyR is defined in Fig. 7.

We now present a denotational semantics for DafnyR's statements. A program state S of the form (σ, H) contains a store and a heap: $State = (Store \times Heap) + \{Error\}$. The $allocate$ function takes the heap and the class name as parameters, and returns a location and a new heap. Also *fieldNames* is a function that takes a class name and returns a list of the names of its declared fields.

$$
\begin{aligned}
&\mathcal{RE}[\![-]\!] = RE \to Store \times Heap \to PowerSet(Loc)\\
&\mathcal{RE}[\![\mathbf{alloc}]\!]_{\sigma,H} = dom(H)\\
&\mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H} = \varnothing\\
&\mathcal{RE}[\![\mathbf{region}\{Expr.f\}]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{fpt}(Expr)]\!]_{\sigma,H}\\
&\quad \cup \{(o,f) \mid o = \mathcal{E}_{DR}[\![Expr]\!]_{\sigma,H}, o \neq null\}\\
&\mathcal{RE}[\![RE_1 + RE_2]\!]_{\sigma,H} = \mathcal{RE}[\![RE_1]\!]_{\sigma,H} \cup \mathcal{RE}[\![RE_2]\!]_{\sigma,H}\\
&\mathcal{RE}[\![RE_1 * RE_2]\!]_{\sigma,H} = \mathcal{RE}[\![RE_1]\!]_{\sigma,H} \cap \mathcal{RE}[\![RE_2]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{if}\ Assrt\ \mathbf{then}\ RE_1\ \mathbf{else}\ RE_2]\!]_{\sigma,H} =\\
&\quad \textbf{if } \mathcal{A}_{DR}[\![Assrt]\!]_{\sigma,H} = true \textbf{ then } \mathcal{RE}[\![RE_1]\!]_{\sigma,H}\\
&\quad \textbf{else } \mathcal{RE}[\![RE_2]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(x)]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(null)]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(n)]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(x.f)]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{region}\{x.f\}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(RE)]\!]_{\sigma,H} =\\
&\quad \begin{cases} \mathcal{RE}[\![\mathbf{fpt}(Assrt)]\!]_{\sigma,H}, & \text{if } RE = \mathbf{if}\ Assrt\ \mathbf{then}\\ & \qquad RE_1\ \mathbf{else}\ RE_2\\ \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}, & \text{otherwise} \end{cases}\\
&\mathcal{RE}[\![\mathbf{fpt}(Expr_1 = Expr_2)]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(Expr_1)]\!]_{\sigma,H} \cup \mathcal{RE}[\![\mathbf{fpt}(Expr_2)]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(Assrt_1 \,\&\&\, Assrt_2)]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(Assrt_1)]\!]_{\sigma,H} \cup \mathbf{if}\ Assrt_1\ \mathbf{then}\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(Assrt_2)]\!]_{\sigma,H}\ \mathbf{else}\ \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(Assrt_1 \,||\, Assrt_2)]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(Assrt_1)]\!]_{\sigma,H} \cup \mathbf{if}\ Assrt_1\ \mathbf{then}\\
&\quad \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}\ \mathbf{else}\ \mathcal{RE}[\![\mathbf{fpt}(Assrt_2)]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(Assrt_1 \Rightarrow Assrt_2)]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(Assrt_1)]\!]_{\sigma,H} \cup \mathbf{if}\ Assrt_1\ \mathbf{then}\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(Assrt_2)]\!]_{\sigma,H}\ \mathbf{else}\ \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(\exists x.Assrt)]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{alloc}]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(P(ins))]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![Frm(P)[ins/formals(P)] + \mathbf{fpt}(ins)]\!]_{\sigma,H}\\
&\quad \text{where } Frm(P) \text{ is the frame of predicate } P \text{ given}\\
&\quad \text{in its declaration, and } formals(P) \text{ is the list of } P\text{'s}\\
&\quad \text{formal parameter names.}\\
&\mathcal{RE}[\![\mathbf{fpt}(RE_1\,!!\,RE_2)]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(RE_1)]\!]_{\sigma,H} \cup \mathcal{RE}[\![\mathbf{fpt}(RE_2)]\!]_{\sigma,H}\\
&\mathcal{RE}[\![\mathbf{fpt}(RE_1 {<=} RE_2)]\!]_{\sigma,H} =\\
&\quad \mathcal{RE}[\![\mathbf{fpt}(RE_1)]\!]_{\sigma,H} \cup \mathcal{RE}[\![\mathbf{fpt}(RE_2)]\!]_{\sigma,H}
\end{aligned}
$$

Figure 8. Semantics of region expressions

DEFINITION 2.7 (The semantics of DafnyR Statements). *The meaning of statements in DafnyR is given by the following, where K is a class name.*

$$
\begin{array}{l}
\bot \ \models_{DR} \mathit{Assrt} \iff \mathit{true} \\
\mathit{Error} \models_{DR} \mathit{Assrt} \iff \mathit{true} \\
\sigma, H \models_{DR} \mathit{Expr}_1 = \mathit{Expr}_2 \iff \mathcal{E}_{DR}[\![\mathit{Expr}_1]\!]_{\sigma,H} = \mathcal{E}_{DR}[\![\mathit{Expr}_2]\!]_{\sigma,H} \\
\sigma, H \models_{DR} P(\mathit{ins}) \iff \mathcal{A}_{RSL}[\![\mathit{Assrt}]\!]_{\sigma',H} \\
\qquad \text{where } \sigma' \text{ is } \sigma[params \mapsto \mathcal{E}_{DR}[\![\mathit{ins}]\!]_{\sigma,H}] \text{ and } P \text{ has body } \mathit{Expr} \text{ and } params \text{ are its formals.} \\
\sigma, H \models_{DR} Assrt_1 \ \&\&\ Assrt_2 \iff \textbf{if } \sigma, H \models_{DR} Assrt_1 \textbf{ then } \sigma, H \models_{DR} Assrt_2 \textbf{ else } \mathit{false} \\
\sigma, H \models_{DR} Assrt_1 \ || \ Assrt_2 \iff \textbf{if } \sigma, H \models_{DR} Assrt_1 \textbf{ then } \mathit{true} \textbf{ else } \sigma, H \models_{DR} Assrt_2 \\
\sigma, H \models_{DR} Assrt_1 \Rightarrow Assrt_2 \iff \textbf{if } \sigma, H \models_{DR} Assrt_1 \textbf{ then } \sigma, H \models_{DR} Assrt_2 \textbf{ else } \mathit{true} \\
\sigma, H \models_{DR} \exists x.Assrt \iff \mathit{exists}\ v.(\sigma[x \mapsto v], H \models_{DR} Assrt) \\
\sigma, H \models_{DR} RE_1 \,!\,!\, RE_2 \iff (\mathcal{RE}[\![RE_1]\!]_{\sigma,H} \cap \mathcal{RE}[\![RE_2]\!]_{\sigma,H}) = \varnothing \\
\sigma, H \models_{DR} RE_1 <= RE_2 \iff \mathcal{RE}[\![RE_1]\!]_{\sigma,H} \subseteq \mathcal{RE}[\![RE_2]\!]_{\sigma,H}
\end{array}
$$

Figure 7. Validity of assertions in DafnyR

$$
\begin{array}{l}
\mathcal{S} : \mathit{State}_\bot \to \mathit{State}_\bot \\
\mathcal{S}[\![(x := \mathit{Expr})]\!]_{\sigma,H} = (\sigma[x \mapsto \mathcal{E}_{DR}[\![\mathit{Expr}]\!]_{\sigma,H}], H) \\
\mathcal{S}[\![(x.f := \mathit{Expr})]\!]_{\sigma,H} = \textbf{\textit{if}}\ \mathcal{E}_{DR}[\![x]\!]_{\sigma,H} \neq \mathit{null} \\
\qquad \textbf{\textit{then}}\ (\sigma, H[(\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f) \mapsto \mathcal{E}_{DR}[\![\mathit{Expr}]\!]_{\sigma,H}]) \\
\qquad \textbf{\textit{else}}\ \mathit{Error} \\
\mathcal{S}[\![(x := x'.f)]\!]_{\sigma,H} = \textbf{\textit{if}}\ \mathcal{E}_{DR}[\![x']\!]_{\sigma,H} \neq \mathit{null} \\
\qquad \textbf{\textit{then}}\ (\sigma[x \mapsto H[\mathcal{E}_{DR}[\![x']\!]_{\sigma,H}, f)], H)\ \textbf{\textit{else}}\ \mathit{Error} \\
\mathcal{S}[\![(x := \texttt{new}\, K)]\!]_{\sigma,H} = \\
\qquad \textbf{\textit{let}}\ (l, H') = \mathit{allocate}(K, H)\ \textbf{\textit{in}} \\
\qquad \textbf{\textit{let}}\ (f_1, \ldots, f_n) = \mathit{fieldNames}(K)\ \textbf{\textit{in}} \\
\qquad \textbf{\textit{let}}\ \sigma' = \sigma[x \mapsto l]\ \textbf{\textit{in}} \\
\qquad\quad (\sigma', H'[(\sigma'(x), f_1) \mapsto 0, \ldots, (\sigma'(x), f_n) \mapsto 0]) \\
\mathcal{S}[\![(\texttt{if}(\mathit{Expr} \neq 0)\{\mathit{Stmt}_1\}\texttt{else}\{\mathit{Stmt}_2\})]\!]_{\sigma,H} = \\
\qquad \textbf{\textit{if}}\ \mathcal{E}_{DR}[\![\mathit{Expr}]\!]_{\sigma,H} \neq 0\ \textbf{\textit{then}}\ \mathcal{S}[\![\mathit{Stmt}_1]\!]_{\sigma,H} \\
\qquad \textbf{\textit{else}}\ \mathcal{S}[\![\mathit{Stmt}_2]\!]_{\sigma,H}
\end{array}
$$

$$
\begin{array}{l}
\mathcal{S}[\![(\texttt{while}(\mathit{Expr} \neq 0)\ \{\mathit{Stmt}\}]\!]_{\sigma,H} = \\
\quad \mathit{fix}(\lambda g\,.\,\underline{\lambda} s\,. \\
\qquad \textbf{\textit{if}}\ \mathcal{E}_{DR}[\![\mathit{Expr}]\!]_{\sigma,H} \neq 0 \\
\qquad \textbf{\textit{then}}\ \textbf{\textit{let}}\ s' = \mathcal{S}[\![\mathit{Stmt}]\!]_{\sigma,H}\ \textbf{\textit{in}}\ gs' \\
\qquad \textbf{\textit{else}}\ s)(\sigma, H) \\
\mathcal{S}[\![(\mathit{Stmt}_1; \mathit{Stmt}_2)]\!]_{\sigma,H} = \\
\qquad \textbf{\textit{let}}\ (\sigma', H') = \mathcal{S}[\![\mathit{Stmt}_1]\!]_{\sigma,H}\ \textbf{\textit{in}}\ \mathcal{S}[\![\mathit{Stmt}_2]\!]_{\sigma',H'}
\end{array}
$$

Next we show that the denotation of the syntactic footprint is the semantic footprint.

LEMMA 2.8. *Let (σ, H) be a state. For all assertions $Assrt$ and expressions Expr, the semantic footprint of $Assrt$ equals $\mathcal{RE}[\![\texttt{fpt}(Assrt)]\!]_{\sigma,H}$ and the semantic footprint of Expr equals $\mathcal{RE}[\![\texttt{fpt}(\mathit{Expr})]\!]_{\sigma,H}$.*

Proof: We prove it by simultaneous induction on the structure of expressions and assertions.

The first base cases are expressions, where *Expr* is of the form x, **null**, n, or the region expressions **region**$\{\}$. In each of these cases $R = \mathcal{RE}[\![\texttt{fpt}(\mathit{Expr})]\!]_{\sigma,H} = \varnothing$, by Def. 2.6. For these cases, by the definition (2.5) the semantic footprint is also $\varnothing$.

The second base case is region expression **alloc**. In this case, $R = \mathcal{RE}[\![\texttt{fpt}(\mathit{Expr})]\!]_{\sigma,H} = \mathit{dom}(H)$, by Def. 2.6. By definition of semantic footprint is also $\mathit{dom}(H)$.

The inductive hypothesis is that for all subexpressions Expr_i, all subassertions $Assrt_i$, for each subexpression, its semantic footprint, F_i, equals either $\mathcal{RE}[\![\texttt{fpt}(\mathit{Expr}_i)]\!]_{\sigma,H}$ (for a subexpression) or $\mathcal{RE}[\![\texttt{fpt}(Assrt_i)]\!]_{\sigma,H}$ (for an subassertion).

The first inductive case is when *Expr* is of the form $\mathit{Expr}_1.f$. In this case, $R_1 = \mathcal{RE}[\![\texttt{fpt}(\mathit{Expr}_1)]\!]_{\sigma,H}$ and thus

$$
\begin{aligned}
R_1 = \ & \mathcal{RE}[\![\texttt{fpt}(\mathit{Expr}_1)]\!]_{\sigma,H} \\
& \cup \{(o, f) \mid o = \mathcal{E}_{DR}[\![\mathit{Expr}_1]\!]_{\sigma,H}, o \neq \mathit{null}\},
\end{aligned}
$$

by Def. 2.6. By the inductive hypothesis, the semantic footprint of Expr_1 is also R_1. There are two subcases; for both of these let o be $\mathcal{E}_{DR}[\![\mathit{Expr}_1]\!]_{\sigma,H}$. One case is if $o \neq \mathit{null}$, in which case the semantic footprint includes the location (o, f), because (o, f) is in the value of the expression. The other case is if $o = \mathit{null}$, in which case the semantic footprint does not include (o, f), and is thus just R_1. Thus in both cases the result follows.

The second inductive case is when $Assrt$ is of the form $\mathit{Expr}_1 = \mathit{Expr}_2$. By the inductive hypothesis, Expr_1's semantic footprint is $F_1 = \mathcal{RE}[\![\texttt{fpt}(\mathit{Expr}_1)]\!]_{\sigma,H}$, and Expr_2's semantic footprint is $F_2 = \mathcal{RE}[\![\texttt{fpt}(\mathit{Expr}_2)]\!]_{\sigma,H}$. By the semantics of DafnyR (Def. 2.6), $\mathcal{RE}[\![\texttt{fpt}(Assrt)]\!]_{\sigma,H}$ is $F_1 \cup F_2$. Since the validity of $\mathit{Expr}_1 = \mathit{Expr}_2$ depends on the value of both Expr_1 and Expr_2, its semantic footprint is also $F_1 \cup F_2$.

Another inductive case is when $Assrt$ is of the form $REAssrt$. By the inductive hypothesis, let F_1 be RE_1's semantic footprint, such that $F_1 = \mathcal{RE}[\![\texttt{fpt}(REAssrt_1)]\!]_{\sigma,H}$, and let F_2 be RE_2's semantic footprint, such that $F_2 = \mathcal{RE}[\![\texttt{fpt}(REAssrt_2)]\!]_{\sigma,H}$. By the semantics of DafnyR (Def. 2.6), let $R = F_1 \cup F_2$. And the validity of $REAssrt$ depends on RE_1 and RE_2. Therefore its semantic footprint is also $F_1 \cup F_2$.

Another inductive case is when $Expr$ is **if** $Assrt$ **then** RE_1 **else** RE_2. By definition $\mathcal{RE}[\![\texttt{fpt}(\mathit{Expr})]\!]_{\sigma,H}$ is $\mathcal{RE}[\![\texttt{fpt}(Assrt)]\!]_{\sigma,H}$. By the inductive hypothesis, $Assrt$'s semantic footprint is $\mathcal{RE}[\![\texttt{fpt}(Assrt)]\!]_{\sigma,H}$, which is the semantic footprint of the entire expression.

Another inductive case is when $Assrt$ is of the form $Assrt_1 \,\&\&\, Assrt_2$, where $F_{a1} = \mathcal{RE}[\![\texttt{fpt}(Assrt_1)]\!]_{\sigma,H}$

and $F_{a2} = \mathcal{RE}[\![\texttt{fpt}(Assrt_2)]\!]_{\sigma,H}$ We prove it by two cases according to whether $Assrt_1$ is valid or invalid.

Case 1. $Assrt_1$ is valid in the state (σ, H). By the definition of DafnyR's footprint and semantics (Def.2.6), let $R = \mathcal{RE}[\![\texttt{fpt}(Assrt_1 \,\&\,\& Assrt_2)]\!]_{\sigma,H} = F_{a1} \cup F_{a2}$. By definition of semantic footprint (Def. 2.5), $\forall H'_1.H_1 \overset{\mathrm{F_{a1}}}{\equiv} H'_1 \Rightarrow \sigma, H_1 \models_{DR} Assrt_1 \iff \sigma, H'_1 \models_{DR} Assrt_1$, and $\forall H'_2.H_2 \overset{\mathrm{F_{a2}}}{\equiv} H'_2 \Rightarrow \sigma, H_2 \models_{DR} Assrt_2 \iff \sigma, H'_2 \models_{DR} Assrt_2$. By assumption, $Assrt_1$ is valid in the state (σ, H), whether $Assrt_1 \,\&\,\& Assrt_2$ is valid or not depends on $Assrt_2$. By our analysis above, $\forall H'.H \overset{\mathrm{F_{a2}}}{\equiv} H' \Rightarrow (\sigma, H \models_{DR} Assrt_2 \iff \sigma, H' \models_{DR} Assrt_2)$. Moreover, by inductive hypothesis, there does not exists F'_{a2}, such that $F'_{a2} \subset F_{a2}$ and $H_2 \overset{\mathrm{F'_{a2}}}{\equiv} H'_2 \Rightarrow (\sigma, H_2 \models_{DR} Assrt_2 \iff \sigma, H'_2 \models_{DR} Assrt_2)$. So $F_{a1} \cup F_{a2}$ is minimal. Hence $F_{a1} \cup F_{a2}$ is the semantic footprint of $Assrt_1 \,\&\,\& Assrt_2$. Therefore $R = F_{a1} \cup F_{a2}$.

Case 2. $Assrt_1$ is invalid in the state (σ, H). By the semantics of DafnyR (Def. 2.6), let $R = \mathcal{RE}[\![\texttt{fpt}(Assrt_1)]\!]_{\sigma,H}$. By inductive hypothesis, $R = F_{a1}$. By the semantics of DafnyR and $Assrt$ is invalid, we have $Assrt_1 \,\&\,\& Assrt_2$ is invalid in the given state no matter what locations that $Assrt_2$ asserts. Therefore F_{a1} is the semantic footprint of $Assrt_1 \,\&\,\& Assrt_2$. Therefore $R = F_{a1}$.

Another inductive case is when $Assrt$ is of the form $Assrt_1 \Rightarrow Assrt_2$. We prove it by two cases according to whether $Assrt_1$ is valid or invalid.

Case 1. $Assrt_1$ is valid in state (σ, H). Let

$$R = \mathcal{RE}[\![\texttt{fpt}(Assrt_1) + \texttt{fpt}(Assrt_2)]\!]_{\sigma,H}.$$

By the inductive hypothesis, $R = F_{a1} \cup F_{a2}$, where each F_{ai} is the semantic footprint of the corresponding $Assrt_i$. By definition of semantic footprint (Def. 2.5), $\forall H'.H \overset{\mathrm{F_{a1}}}{\equiv} H' \Rightarrow \sigma, H \models_{DR} Assrt_1 \iff \sigma, H' \models_{DR} Assrt_1$, and $\forall H'.H \overset{\mathrm{F_2}}{\equiv} H' \Rightarrow \sigma, H \models_{DR} Assrt_2 \iff \sigma, H' \models_{DR} Assrt_2$. Therefore, $\forall H'.\ H \overset{\mathrm{F_{a1} \cup F_2}}{\equiv} H' \Rightarrow \sigma, H \models_{DR} Assrt_1 \iff \sigma, H' \models_{DR} Assrt_1$ *and* $\sigma, H \models_{DR} Assrt_2 \iff \sigma, H' \models_{DR} Assrt_2$. By assumption, $Assrt_1$ is valid in the state (σ, H), whether $Assrt_1 \Rightarrow Assrt_2$ is valid or not depends on $Assrt_2$. By our analysis above, $\forall H'.H \overset{\mathrm{F_1 \cup F_2}}{\equiv} H' \Rightarrow (\sigma, H \models_{DR} Assrt_2 \iff \sigma, H' \models_{DR} Assrt_2)$. Moreover, by inductive hypothesis, there does not exists F'_1 and F'_2, such that $F'_1 \subset F_1$, $F'_2 \subset F_2$ and $\forall H'.H \overset{\mathrm{F'_1}}{\equiv} H' \Rightarrow (\sigma, H \models_{DR} Assrt_1 \iff \sigma, H' \models_{DR} Assrt_1)$, and $H \overset{\mathrm{F'_2}}{\equiv} H' \Rightarrow (\sigma, H \models_{DR} Assrt_2 \iff \sigma, H' \models_{DR} Assrt_2)$. So $F_1 \cup F_2$ is minimal. Hence $F_1 \cup F_2$ is the semantic footprint of $Assrt_1 \Rightarrow Assrt_2$. Therefore $R = F_1 \cup F_2$.

Case 2. $Assrt_1$ is invalid in the state (σ, H). By the semantics of DafnyR (Def. 2.6),let $R = \mathcal{RE}[\![\texttt{fpt}(Assrt_1)]\!]_{\sigma,H}$. By inductive hypothesis, $R = F_1$. By the semantics of DafnyR and $Assrt_1$ is invalid, we have $Assrt_1 \Rightarrow Assrt_2$ is invalid in the given state no matter what locations that $Assrt_2$ asserts. Therefore F_1 is the semantic footprint of $Assrt_1 \Rightarrow Assrt_2$. Therefore $R = F_1$.

Another inductive case is when $Assrt$ is of the form $\exists\ x.\ Assrt$. By the semantics of DafnyR (Def. 2.6), let $R = \mathcal{RE}[\![\texttt{fpt}(\exists\, x.\, Assrt)]\!]_{\sigma,H} = dom(H)$. It is trivial true.

Another inductive case is when $Assrt$ is of the form $P(ins)$. By the semantics of DafnyR (Def. 2.6),let $R = \mathcal{RE}[\![Frm(P)[ins/formals(P)]]\!]_{\sigma,H} \cup \mathcal{RE}[\![\texttt{fpt}(ins)]\!]_{\sigma,H}$. By assumption, $\mathcal{RE}[\![Frm(P)[ins/formals(P)]]\!]_{\sigma,H}$ equals the semantic footprint of P's body. By the inductive hypothesis $\mathcal{RE}[\![\texttt{fpt}(ins)]\!]_{\sigma,H}$ equals ins's semantic footprint. Therefore R equals its semantic footprint.

The inductive case of disjunction is similar. ∎

2.4 Verification Logic

The validity of a Hoare-formula $\{-\}Stmt\{-\}[-]$ means that it is partially correct and respects the specified frame (given by the region expression after the postcondition).

DEFINITION 2.9 (Valid Hoare-formula). *Let Stmt be a statement, let P and Q be assertions, let ε be a region expression, and let (σ, H) be a state. Then $\{P\}$ Stmt $\{Q\}[\varepsilon]$ is valid in (σ, H), written $\sigma, H \models_{DR} \{P\}$ Stmt $\{Q\}[\varepsilon]$, if and only if whenever $\sigma, H \models_{DR} P$ and $(\sigma', H') = \mathcal{S}[\![Stmt]\!]_{\sigma,H}$, then $\sigma', H' \models_{DR} Q$ and for all $(o, f) \in dom(H)$,*

$$H'[o, f] \neq H[o, f] \Rightarrow (o, f) \in \mathcal{RE}[\![\varepsilon]\!]_{\sigma,H}.$$

A Hoare-formula $\{P\}Stmt\{Q\}[\varepsilon]$ is valid*, written $\models_{DR} \{P\}Stmt\{Q\}[\varepsilon]$, if and only if for all states (σ, H), $\sigma, H \models_{DR} \{P\}$ Stmt $\{Q\}[\varepsilon]$.*

The proof axioms and rules for DafnyR are adapted from various papers [1, 6].

DEFINITION 2.10 (Proof rules and axioms for DafnyR). *The axioms and inference rules for the partial correctness of DanfyR statements are shown in Fig. 9.*

3. Restricted Separation Logic

In this section we introduce a slightly restricted version of separation logic, which we call RSL, and show how to translate it into DafnyR.

3.1 Syntax of RSL

Our syntax for RSL follows Parkinson and Summers [17] in restricting existential assertions so that they can only quantify over values stored in the heap. Without such a restriction separation logic tools are not complete [17]. In addition, we exclude the emp predicate and separating implication, for reasons that we will explain in the discussion.

DEFINITION 3.1 (Restricted Separation Logic). *The syntax of* restricted separation logic *has assertions (a) and expressions (e) defined as follows:*

$(ALLOC_{DR}) \vdash_{DR} \{true\}\ x := \mathbf{new}\ K\ \{\&\&_{i=1}^{n} PointsTo_{f_i}(x, 0)\}\ \ [\mathbf{region}\{\}]$ where $(f_1, \ldots, f_n) = \mathit{fieldNames}(K)$

$(ASGN_{DR}) \vdash_{DR} \{true\}\ x := \mathit{Expr}\ \{x = \mathit{Expr}\}\ [\mathbf{region}\{\}]$ where $x \notin \mathtt{FV}(\mathit{Expr})$

$(UPD_{DR}) \vdash_{DR} \{x \neq null\}\ x.f := \mathit{Expr}\ \{x.f = \mathit{Expr}\}\ [\mathbf{region}\{x.f\}]$ where $x \notin \mathtt{FV}(\mathit{Expr})$

$(ACC_{DR}) \vdash_{DR} \{x' \neq null\ \&\&\ x'.f = \mathit{Expr}\}\ x := x'.f\ \{x = \mathit{Expr}]\}\ [\mathbf{region}\{\}]$ where $x \neq x'$ and $x \notin \mathtt{FV}(\mathit{Expr})$

$$(IF_{DR})\ \frac{\vdash_{DR} \{P\ \&\&\ \mathit{Expr} \neq 0\}\ \mathit{Stmt}_1\{Q\}\ [\varepsilon],\quad \vdash_{DR} \{P\ \&\&\ \mathit{Expr} = 0\}\ \mathit{Stmt}_2\{Q\}\ [\varepsilon]}{\vdash_{DR} \{P\}\ \mathbf{if}(\mathit{Expr} \neq 0)\{\mathit{Stmt}_1\}\mathbf{else}\{\mathit{Stmt}_2\}\{Q\}\ [\varepsilon]}$$

$$(WHILE_{DR})\ \frac{\vdash_{DR} \{I\ \&\&\ \mathit{Expr} \neq 0\}\ \mathit{Stmt}\{I\}\ [\varepsilon]}{\vdash_{DR} \{I\}\ \mathbf{while}(\mathit{Expr} \neq 0)\ \{\mathit{Stmt}\}\ \{I\ \&\&\ \mathit{Expr} = 0\}\ [\varepsilon]}$$

$$(SEQ_{DR})\ \frac{\vdash_{DR} \{P\}\ \mathit{Stmt}_1\ \{Q'\}\ [\varepsilon_1],\ \vdash_{DR} \{Q'\}\ \mathit{Stmt}_2\{Q\}\ [\varepsilon_2]}{\vdash_{DR} \{P\}\ \mathit{Stmt}_1; \mathit{Stmt}_2\{Q\}\ [\varepsilon_1 + \varepsilon_2]}$$

$$(SubEff_{DR})\ \frac{\vdash_{DR} \{P\}\mathit{Stmt}\{Q\}\ [\varepsilon],\quad P \vdash \varepsilon <= \varepsilon'}{\vdash_{DR} \{P\}\mathit{Stmt}\{Q\}\ [\varepsilon']}$$

$$(CON_{DR})\ \frac{\vdash P \Rightarrow P',\ \vdash Q' \Rightarrow Q,\ \vdash_{DR} \{P'\}\mathit{Stmt}\{Q'\}[\varepsilon]}{\vdash_{DR} \{P\}\mathit{Stmt}\{Q\}\ [\varepsilon]}$$

$$(FRM_{DR})\ \frac{\vdash_{DR} \{P\}\ \mathit{Stmt}\ \{Q\}}{\vdash_{DR} \{P\ \&\&\ R\}\ \mathit{Stmt}\ \{Q\ \ \&\&\ \ R\}\ [\varepsilon]}\quad \textbf{where } (\mathtt{Modify}(\mathit{Stmt})\ \cap\ \mathtt{FV}(R)) = \varnothing \textbf{ and } \varepsilon * \mathbf{fpt}(R) = \mathbf{region}\{\}$$

$\mathtt{Modify}(-)$ computes the set of (stack) variables that may be updated by a statement. It is defined as follows:

$\mathtt{Modify}(x := \mathit{Expr}) = \{x\}$
$\mathtt{Modify}(x.f := \mathit{Expr}) = \{\}$
$\mathtt{Modify}(x := x'.f) = \{x\}$
$\mathtt{Modify}(x := \mathbf{new}K) = \{x\}$
$\mathtt{Modify}(\mathbf{if}(\mathit{Expr} \neq 0)\{\mathit{Stmt}_1\}\mathbf{else}\{\mathit{Stmt}_2\}) = \mathtt{Modify}(\mathit{Stmt}_1) \cup \mathtt{Modify}(\mathit{Stmt}_2)$
$\mathtt{Modify}(\mathbf{while}(\mathit{Expr} \neq 0)\{\mathit{Stmt}\}) = \mathtt{Modify}(\mathit{Stmt})$
$\mathtt{Modify}(\mathit{Stmt}_1; \mathit{Stmt}_2) = \mathtt{Modify}(\mathit{Stmt}_1) \cup \mathtt{Modify}(\mathit{Stmt}_2)$

Let ε and η be frames (region expressions). We define sub-frame rules as follows:

$$\vdash \varepsilon <= \varepsilon \qquad \vdash \mathbf{region}\{\} <= \varepsilon \qquad \vdash \varepsilon + \eta <= \eta + \varepsilon \qquad \frac{\vdash \varepsilon_1 <= \varepsilon_2}{\vdash \varepsilon_1 + \eta <= \varepsilon_2 + \eta} \qquad \frac{\vdash \varepsilon_1 <= \varepsilon_2 \quad \vdash \varepsilon_2 <= \varepsilon_3}{\vdash \varepsilon_1 <= \varepsilon_3}$$

Figure 9. Proof rules and axioms in DafnyR

$$a\ ::=\ e_1\ =\ e_2\ \mid\ x.f \mapsto e\ \mid\ a_1\ *\ a_2\ \mid\ a_1\ \wedge\ a_2\ \mid\ a_1\ \vee\ a_2\ \mid\ a_1\ \Rightarrow\ a_2\ \mid\ \exists x'.x.f\ \mapsto\ x'\ *\ a$$
$$e\ ::=\ x\ \mid\ \mathbf{null}\ \mid\ n$$

We use the same abbreviations in RSL as in DafnyR for $true$, $false$, $\neg$, and $\forall$. In addition, we write $\exists x'.x.f \mapsto x'$ as an abbreviation for $\exists x'.x.f \mapsto x' * true$.

3.2 Semantics of RSL

The semantics of RSL is given using states that consist of a pair, (σ, h), of a store and a heap, as in DafnyR's semantics. Stores (σ), heaps (h), and *Value*s are also as in DafnyR.

We adapt the Reynolds's classical semantics for Separation Logic [18], because it is more expressive than the intuitionistic semantics [8].

DEFINITION 3.2 (RSL Semantics). *Assuming that $\mathcal{N}$ is the standard meaning function for numeric literals and σ is a store, then the semantics of expressions in separation logic is:*

$$\mathcal{E}_{RSL} : e \to \mathit{Store} \to \mathit{Value}$$
$$\mathcal{E}_{RSL}[\![x]\!]_\sigma = \sigma(x) \quad \mathcal{E}_{RSL}[\![n]\!]_\sigma = \mathcal{N}[\![n]\!]$$
$$\mathcal{E}_{RSL}[\![\mathbf{null}]\!]_\sigma = null$$

The semantics of assertions, $\mathcal{A}_{RSL}[\![-]\!]_{\sigma,h}$ is defined by:

$$\mathcal{A}_{RSL} : a \to \mathit{Store} \times \mathit{Heap} \to \mathit{Boolean}$$
$$\mathcal{A}_{RSL}[\![a]\!]_{\sigma,h} = \begin{cases} true, & \text{if } \sigma, h \models_{RSL} a \\ false, & \text{if } \sigma, h \not\models_{RSL} a \end{cases}$$

The validity of assertions in RSL is defined in Fig. 10.

3.3 Verification Logic

To allow comparison with DafnyR's logic, we use DafnyR statements in a verification logic that uses RSL assertions. The meaning of Hoare triples $\{-\}Stmt\{-\}$ is defined as follows

DEFINITION 3.3 (Validity of Hoare Triples). *Let $Stmt$ be a DafnyR statement, a_1 and a_2 be RSL assertions, and let (σ, h) be a program state. Then the Hoare triple $\{a_1\}\mathit{Stmt}\{a_2\}$* is valid in (σ, h), *written $\sigma, h \models_{RSL} \{a_1\}\ \mathit{Stmt}\ \{a_2\}$, if and only if whenever $\sigma, h \models_{RSL} a_1$ and $(\sigma', h') = \mathcal{S}[\![\mathit{Stmt}]\!]_{\sigma,h}$, then $\sigma', h' \models_{RSL} a_2$.*

$\{a_1\}\ \mathit{Stmt}\ \{a_2\}$ is valid*, written $\models_{RSL} \{a_1\}\ \mathit{Stmt}\ \{a_2\}$, if and only if, for all states (σ, h), $\sigma, h \models_{RSL} \{a_1\}\ \mathit{Stmt}\ \{a_2\}$.*

3.3.1 Provability Relation

Our proof axioms and rules for DafnyR statements, using RSL, are adapted from various papers [6, 18]. Note that con-

$$
\begin{array}{l}
\sigma, h \vDash_{RSL} e = e' \iff \mathcal{E}_{RSL}[\![e]\!]_\sigma = \mathcal{E}_{RSL}[\![e']\!]_\sigma \\
\sigma, h \vDash_{RSL} x.f \mapsto e \iff dom(h) = \{(\mathcal{E}_{RSL}[\![x]\!]_\sigma, f)\} \; and \; \mathcal{E}_{RSL}[\![x]\!]_\sigma \neq null \; and \; h[\mathcal{E}_{RSL}[\![x]\!]_\sigma, f] = \mathcal{E}_{RSL}[\![e]\!]_\sigma \\
\sigma, h \vDash_{RSL} a_1 * a_2 \iff exists\; h_1, h_2.(h_1 \bot h_2 \; and \; h = h_1 \cdot h_2 \; and \; \textbf{if} \;\; \sigma, h_1 \vDash_{RSL} a_1 \; \textbf{then} \;\; \sigma, h_2 \vDash_{RSL} a_2 \; \textbf{else} \;\; false) \\
\sigma, h \vDash_{RSL} a_1 \wedge a_2 \iff \textbf{if} \;\; \sigma, h \vDash_{RSL} a_1 \; \textbf{then} \;\; \sigma, h \vDash_{RSL} a_2 \; \textbf{else} \;\; false \\
\sigma, h \vDash_{RSL} a_1 \vee a_2 \iff \textbf{if} \;\; \sigma, h \vDash_{RSL} a_1 \; \textbf{then} \;\; true \; \textbf{else} \;\; \sigma, h \vDash_{RSL} a_2 \\
\sigma, h \vDash_{RSL} a_1 \Rightarrow a_2 \iff \textbf{if} \;\; \sigma, h \vDash_{RSL} a_1 \; \textbf{then} \;\; \sigma, h \vDash_{RSL} a_2 \; \textbf{else} \; true \\
\sigma, h \vDash_{RSL} \exists x'.\, x.f \mapsto x' * a \iff exists\; v.\, (\sigma[x' \mapsto v], h \vDash_{RSL} x.f \mapsto x' * a)
\end{array}
$$

Figure 10. Validity of assertions in RSL

ventionally, predicate *emp* is used to specify the precondition of allocation. However, since RSL does not have *emp*, we use $true$ instead.

DEFINITION 3.4 (Proof rules and axioms in RSL). *Let P and Q be assertions in RSL. Let Stmt be a well-formed statement in DafnyR. Then the form $\vdash_{RSL}$ $\{P\}Stmt\{Q\}$ is a partial correctness judgment for DafnyR programs in RSL. It is defined in Fig. 11.*

$\texttt{Modify}(-)$ computes the set of (stack) variables that may be updated by a statement. It is defined in Fig. 9.

4. Translation from RSL to DafnyR

The translation from RSL assertions to DafnyR assertions is syntactic and local.

The syntactic mapping $\texttt{TR}[\![-]\!]$ is overloaded. It operates on both RSL expressions and assertions.

4.1 Translation of Expressions

The translation for expressions is trivial.

DEFINITION 4.1. *The syntactic mapping from RSL expressions to DafnyR expressions is defined as follows:*

$$\texttt{TR}[\![x]\!] = x \quad \texttt{TR}[\![null]\!] = null \quad \texttt{TR}[\![n]\!] = n$$

This preserves the meaning of RSL expressions.

LEMMA 4.2. *Let e be an RSL expression, σ be a store and H be a heap. Then $\mathcal{E}_{RSL}[\![e]\!]_\sigma = \mathcal{E}_{DR}[\![\texttt{TR}[\![e]\!]]\!]_{\sigma,H}$.*

Proof: By the semantics of RSL, the meaning of an expression solely depends on σ. Therefore, the heap H is irrelevant, and thus the values of the expression in both semantics are equal. ∎

4.2 Translation of Assertions

The translation for assertions is more interesting.

DEFINITION 4.3. *The syntactic mapping from RSL assertions to DafnyR assertions is defined in Fig. 12*

4.3 Footprint of assertions in RSL and results about the translation

To show that the syntactic mapping in Definition 4.3 preserves their meanings, we must show that (1) the translation preserves the semantic footprints of assertions in each state, and (2) the translation preserves validity of assertions. Therefore we first give a hypothetical footprints of assertions in terms of RSL's syntax and region expressions, and prove that it is the semantic footprint. Then we prove that the meaning of both hypothetical footprints and assertions are preserved by the translation. Finally we define the syntactical footprint of assertions of RSL in terms of region expressions in DafnyR's syntax.

4.3.1 Hypothetical footprint of assertions in RSL

We want to give a syntactical definition of assertions' semantic footprint in RSL in terms of region expressions in DafnyR's syntax. However, some assertions' semantic footprints need to be expressed with conditional region expressions ($\texttt{\textbf{if}}\; Assrt \;\texttt{\textbf{then}}\; RE_1 \;\texttt{\textbf{else}}\; RE_2$). For example, the semantic footprint of $a_1 * a_2$ is the union of the semantic footprint of a_1 and the semantic footprint of a_2 if a_1 is true, otherwise, it is just the semantic footprint of a_1. However, we cannot use $\texttt{TR}[\![a_1]\!]$ in defining its semantic footprint, because we do not know if $\texttt{TR}[\![a_1]\!]$ semantically equals a_1; indeed, that is what we want to prove.

Therefore, we temporarily presume that region expressions support the syntax $\texttt{\textbf{if}}\; a \;\texttt{\textbf{then}}\; RE_1 \;\texttt{\textbf{else}}\; RE_2$. Its semantics is defined in formula (1) below:

$$
\begin{array}{l}
\mathcal{RE}[\![\texttt{\textbf{if}}\; a \;\texttt{\textbf{then}}\; RE_1 \;\texttt{\textbf{else}}\; RE_2]\!]_{\sigma,h} = \\
\quad \textbf{if}\; \mathcal{A}_{RSL}[\![a]\!]_{\sigma,h} = true \;\textbf{then}\; \mathcal{RE}[\![RE_1]\!]_{\sigma,h} \\
\quad \textbf{else}\; \mathcal{RE}[\![RE_2]\!]_{\sigma,h}
\end{array} \qquad (1)
$$

Using these presumed region expressions, we define a hypothetical footprint of assertions in RSL, and prove our translation of assertions of RSL preserves their meanings.

DEFINITION 4.4 (Hypothetical footprint for RSL). *The* hypothetical footprint function for expressions *maps all expressions to the empty region: $\texttt{\textbf{fp}}_{Hy}(e) = \texttt{\textbf{region}}\{\}$.*

The hypothetical footprint function for assertions *maps assertions to region expressions as follows:*

$$(ALLOC_{RSL}) \vdash_{RSL} \{true\}\ x := \textbf{new}\ K\ \{\circledast_{i=1}^{n} x.f_i \mapsto 0\}\ \textbf{where}\ (f_1, \ldots, f_n) = fields(K)$$
$$(ASGN_{RSL}) \vdash_{RSL} \{true\}\ x := Expr\ \{x = Expr\}\ \textbf{where}\ x \notin \texttt{FV}(Expr)$$
$$(UPD_{RSL}) \vdash_{RSL} \{\exists v.x.f \mapsto v\}\ x.f := Expr\ \{x.f \mapsto Expr\}$$
$$(ACC_{RSL}) \vdash_{RSL} \{x'.f \mapsto Expr\}x := x'.f\{x = Expr \wedge x'.f \mapsto Expr\}\ \textbf{where}\ x \neq x'\ \text{and}\ x \notin \texttt{FV}(Expr)$$
$$(IF_{RSL})\ \frac{\vdash_{RSL} \{P \wedge Expr \neq 0\}\ Stmt_1\ \{Q\}, \qquad \vdash_{RSL} \{P \wedge Expr = 0\}\ Stmt_2\ \{Q\}}{\vdash_{RSL} \{P\}\ \texttt{if}(Expr \neq 0)\{Stmt_1\}\texttt{else}\{Stmt_2\}\ \{Q\}}$$
$$(WHILE_{RSL})\ \frac{\vdash_{RSL} \{I \wedge Expr \neq 0\}\ Stmt\ \{I\}}{\vdash_{RSL} \{I\}\ \texttt{while}(Expr \neq 0)\{Stmt\}\ \{I \wedge Expr = 0\}} \qquad (SEQ_{RSL})\ \frac{\vdash_{RSL} \{P\}\ Stmt_1\ \{Q'\}, \quad \vdash_{RSL} \{Q'\}\ Stmt_2\ \{Q\}}{\vdash_{RSL} \{P\}\ Stmt_1; Stmt_2\ \{Q\}}$$
$$(CON_{RSL})\ \frac{\vdash P \Rightarrow P', \qquad \vdash_{RSL} \{P'\}\ Stmt\ \{Q'\}, \qquad \vdash Q' \Rightarrow Q}{\vdash_{RSL} \{P\}\ Stmt\ \{Q\}} \qquad (FRM_{RSL})\ \frac{\vdash_{RSL} \{P\}\ Stmt\ \{Q\}}{\vdash_{RSL} \{P * R\}\ Stmt\ \{Q * R\}}\ \textbf{where}\ (\texttt{Modify}(Stmt) \cap \texttt{FV}(R)) = \varnothing$$

Figure 11. Axioms and inference rules for verification of statements using RSL.

$$\begin{array}{ll}
\texttt{TR}[\![e_1 = e_2]\!] = \texttt{TR}[\![e_1]\!] = \texttt{TR}[\![e_2]\!] & \texttt{TR}[\![x.f \mapsto e]\!] = PointsTo_f(\texttt{TR}[\![x]\!], \texttt{TR}[\![e]\!]) \\
\multicolumn{2}{l}{\texttt{TR}[\![a_1 * a_2]\!] = \texttt{TR}[\![a_1]\!]\,\texttt{\&\&}\,\texttt{TR}[\![a_2]\!]\,\texttt{\&\&}\ (\textbf{fpt}(\texttt{TR}[\![a_1]\!])\,\texttt{!!}\,\textbf{fpt}(\texttt{TR}[\![a_2]\!]))} \\
\texttt{TR}[\![a_1 \wedge a_2]\!] = \texttt{TR}[\![a_1]\!]\,\texttt{\&\&}\,\texttt{TR}[\![a_2]\!] & \texttt{TR}[\![a_1 \vee a_2]\!] = \texttt{TR}[\![a_1]\!]\,\texttt{||}\,\texttt{TR}[\![a_2]\!] \\
\texttt{TR}[\![a_1 \Rightarrow a_2]\!] = \texttt{TR}[\![a_1]\!] \Rightarrow \texttt{TR}[\![a_2]\!] & \texttt{TR}[\![\exists x'.x.f \mapsto x' * a]\!] = \exists x'.\texttt{TR}[\![x.f \mapsto x' * a]\!]
\end{array}$$

Figure 12. Syntactic mapping from RSL assertions to DafnyR assertions

$$\begin{array}{l}
\textbf{\textit{fp}}_{Hy}(e_1 = e_2) = \textbf{\textit{region}}\{\} \\
\textbf{\textit{fp}}_{Hy}(x.f \mapsto e) = \textbf{\textit{region}}\{x.f\} \\
\textbf{\textit{fp}}_{Hy}(a_1 * a_2) = \textbf{\textit{fp}}_{Hy}(a_1) + \\
\quad \textbf{\textit{if}}\ a_1\ \textbf{\textit{then}}\ \textbf{\textit{fp}}_{Hy}(a_2)\ \textbf{\textit{else}}\ \textbf{\textit{region}}\{\} \\
\textbf{\textit{fp}}_{Hy}(a_1 \wedge a_2) = \textbf{\textit{fp}}_{Hy}(a_1) + \\
\quad \textbf{\textit{if}}\ a_1\ \textbf{\textit{then}}\ \textbf{\textit{fp}}_{Hy}(a_2)\ \textbf{\textit{else}}\ \textbf{\textit{region}}\{\} \\
\textbf{\textit{fp}}_{Hy}(a_1 \vee a_2) = \textbf{\textit{fp}}_{Hy}(a_1) + \\
\quad \textbf{\textit{if}}\ a_1\ \textbf{\textit{then}}\ \textbf{\textit{region}}\{\}\ \textbf{\textit{else}}\ \textbf{\textit{fp}}_{Hy}(a_2) \\
\textbf{\textit{fp}}_{Hy}(a_1 \Rightarrow a_2) = \textbf{\textit{fp}}_{Hy}(a_1) + \textbf{\textit{if}}\ a_1\ \textbf{\textit{then}} \\
\quad \textbf{\textit{fp}}_{Hy}(a_2)\ \textbf{\textit{else}}\ \textbf{\textit{region}}\{\} \\
\textbf{\textit{fp}}_{Hy}(\exists x'.x.f \mapsto x' * a) = \\
\quad \textbf{\textit{region}}\{x.f\} + \textbf{\textit{fp}}_{Hy}(a)[x.f/x'].
\end{array}$$

Next we show our semantic evaluation function $\mathcal{RE}$ of the built-in syntactic footprint function $\textbf{fp}_{Hy}$ gives the semantic footprint.

LEMMA 4.5. *Let (σ, h) be a state. Let a be an assertion in RSL. Let F be the semantic footprint in state (σ, h). Let $R = \mathcal{RE}[\![\textbf{\textit{fp}}_{Hy}(a)]\!]_{\sigma,h}$. Then $R = F$.*

Proof: We prove the theorem by the induction on the assertion's structure. One base case is when a is of the form $e_1 = e_2$. By definition of hypothetical footprint (Def. 4.4) and semantics of DafnyR (Def. 2.6), $R = \mathcal{RE}[\![\textbf{fp}_{Hy}(e_1 = e_2)]\!]_{\sigma,h} = \mathcal{RE}[\![\textbf{region}\{\}]\!]_{\sigma,h} = \varnothing$. By definition of semantic footprint (Def. 2.5), $F = \varnothing$. Therefore $R = F$.

The second base case is when a is of the form $x.f \mapsto e$. By definition of hypothetical footprint (Def. 4.4) and semantics of DafnyR (Def. 2.6), $R = \mathcal{RE}[\![\textbf{fp}_{Hy}(x.f \mapsto e)]\!]_{\sigma,h} = \mathcal{RE}[\![\textbf{region}\{x.f\}]\!]_{\sigma,h} = \{(\mathcal{E}_{RSL}[\![x]\!]_{\sigma,h}, f)\}$. By definition of semantic footprint (Def. 2.5), $(\mathcal{E}_{RSL}[\![x]\!]_{\sigma,h}, f)$ is the only location whose value can affect the assertion's validity. Therefore $R = F$.

The inductive hypothesis is that for each subassertion a_i, if its semantic footprint in (σ, h) is F_i, and if $R_i = \mathcal{RE}[\![\textbf{fp}_{Hy}(a_i)]\!]_{\sigma,h}$, then $R_i = F_i$.

The first inductive case is when a is of the form $a_1 * a_2$. By semantics of RSL, the current heap h can be divided into two disjoint sub-heaps, h_1 and h_2, where a_1 and a_2 hold separately. We prove $R = F$ by two cases according to whether a_1 is valid or invalid.

Case 1. a_1 is valid in the state (σ, h). By the definition of hypothetical footprint (Def. 4.4), we have $R = \mathcal{RE}[\![\textbf{fp}_{Hy}(a_1) + \textbf{fp}_{Hy}(a_2)]\!]_{\sigma,h}$. By the inductive hypothesis, $R = F_1 \cup F_2$, where each F_i is the semantic footprint of the corresponding a_i. By definition of semantic footprint (Def. 2.5), $\forall h'_1.h_1 \stackrel{\mathrm{F_1}}{\equiv} h'_1 \Rightarrow \sigma, h_1 \vDash_{RSL} a_1 \iff \sigma, h'_1 \vDash_{RSL} a_1$, and $\forall h'_2.h_2 \stackrel{\mathrm{F_2}}{\equiv} h'_2 \Rightarrow \sigma, h_2 \vDash_{RSL} a_2 \iff \sigma, h'_2 \vDash_{RSL} a_2$. By set theory, $F_1 \subseteq (F_1 \cup F_2)$ and $F_2 \subseteq (F_1 \cup F_2)$. By definition of heap (Def. 2.2), $\forall h'_1, h'_2.h_1 \cdot h'_2 \stackrel{\mathrm{F_1 \cup F_2}}{\equiv} h'_1 \cdot h'_2 \Rightarrow \sigma, h_1 \cdot h'_2 \vDash_{RSL} a_1 \iff \sigma, h'_1 \cdot h'_2 \vDash_{RSL} a_1$, and $\forall h'_1, h'_2.h_2 \cdot h'_1 \stackrel{\mathrm{F_1 \cup F_2}}{\equiv} h'_2 \cdot h'_1 \Rightarrow \sigma, h_2 \cdot h'_1 \vDash_{RSL} a_2 \iff \sigma, h'_2 \cdot h'_1 \vDash_{RSL} a_2$. Therefore, since $h = h_1 \cdot h_2$, we conclude $\forall h'.h \overset{F_1 \cup F_2}{\equiv} h' \Rightarrow (\sigma, h \models_{RSL} a_1 \iff \sigma, h' \models_{RSL} a_1)$ *and* $(\sigma, h \models_{RSL} a_2 \iff \sigma, h' \models_{RSL} a_2)$. By assumption, a_1 is valid in the state (σ, h), whether $a_1 * a_2$ is valid or not depends on a_2. By our analysis above, $\forall h'.h \overset{F_1 \cup F_2}{\equiv} h' \Rightarrow (\sigma, h \models_{RSL} a_2 \iff \sigma, h' \models_{RSL} a_2)$. Moreover, by inductive hypothesis, there does not exist F_1' and F_2', such that $F_1' \subset F_1$, $F_2' \subset F_2$ and $\forall h_1', h_2'.h_1 \overset{F_1'}{\equiv} h_1' \Rightarrow (\sigma, h_1 \models_{RSL} a_1 \iff \sigma, h_1' \models_{RSL} a_1)$, and $h_2 \overset{F_2'}{\equiv} h_2' \Rightarrow (\sigma, h_2 \models_{RSL} a_2 \iff \sigma, h_2' \models_{RSL} a_2)$. So $F_1 \cup F_2$ is minimal. Hence $F_1 \cup F_2$ is the semantic footprint of $a_1 * a_2$. Therefore $R = F_1 \cup F_2$.

Case 2. a_1 is invalid in the state (σ, h). By the definition of hypothetical footprint, $R = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h}$. By the inductive hypothesis, the semantic footprint of a_1 is F_1. By the semantics of RSL and a_1 is invalid, we have $a_1 * a_2$ is invalid in the given state no matter what locations that a_2 asserts. Therefore F_1 is the semantic footprint of $a_1 * a_2$. Therefore $R = F_1$.

The second inductive case is when a is of the form $a_1 \Rightarrow a_2$. We prove it by two cases according to whether a_1 is valid or invalid.

Case 1. a_1 is valid in state (σ, h). By the definition of hypothetical footprint, $R = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1) + \mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h}$. By the inductive hypothesis, $R = F_1 \cup F_2$, where each F_i is the semantic footprint of the corresponding a_i. By definition of semantic footprint (Def. 2.5), $\forall h'.h \overset{F_1}{\equiv} h' \Rightarrow \sigma, h \models_{RSL} a_1 \iff \sigma, h' \models_{RSL} a_1$, and $\forall h'.h \overset{F_2}{\equiv} h' \Rightarrow \sigma, h \models_{RSL} a_2 \iff \sigma, h' \models_{RSL} a_2$. Therefore, $\forall h'.\ h \overset{F_1 \cup F_2}{\equiv} h' \Rightarrow \sigma, h \models_{RSL} a_1 \iff \sigma, h' \models_{RSL} a_1$ *and* $\sigma, h \models_{RSL} a_2 \iff \sigma, h' \models_{RSL} a_2$. By assumption, a_1 is valid in the state (σ, h), whether $a_1 \Rightarrow a_2$ is valid or not depends on a_2. By our analysis above, $\forall h'.h \overset{F_1 \cup F_2}{\equiv} h' \Rightarrow (\sigma, h \models_{RSL} a_2 \iff \sigma, h' \models_{RSL} a_2)$. Moreover, by inductive hypothesis, there does not exist F_1' and F_2', such that $F_1' \subset F_1$, $F_2' \subset F_2$ and $\forall h'.h \overset{F_1'}{\equiv} h' \Rightarrow (\sigma, h \models_{RSL} a_1 \iff \sigma, h' \models_{RSL} a_1)$, and $h \overset{F_2'}{\equiv} h' \Rightarrow (\sigma, h \models_{RSL} a_2 \iff \sigma, h' \models_{RSL} a_2)$. So $F_1 \cup F_2$ is minimal. Hence $F_1 \cup F_2$ is the semantic footprint of $a_1 \Rightarrow a_2$. Therefore $R = F_1 \cup F_2$.

Case 2. a_1 is invalid in the state (σ, h). By the definition of hypothetical footprint, $R = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h}$. By the inductive hypothesis, the semantic footprint of a_1 is F_1. By the semantics of RSL, since a_1 is invalid, $a_1 \Rightarrow a_2$ is invalid in the given state no matter what locations a_2 asserts. Therefore F_1 is the semantic footprint of $a_1 \Rightarrow a_2$, and thus $R = F_1$.

The third inductive case is when a is of the form $\exists x'.x.f \mapsto x' * a$. By the definition of hypothetical footprint, the footprint of existential assertions do not depend on the existential variables. Therefore this case is the same as the case of separating conjunction.

The other inductive cases, conjunction and disjunction, are similar. ∎

We have shown that the hypothetical footprint is the semantic footprint of assertions of RSL. So, from now on, we use *hypothetical footprint* as a synonym for the semantic footprint of RSL assertions. Using Lemma 4.2, we can show that the hypothetical footprint of each assertion of RSL is always a subset of the domain of the current heap corresponding to the definition of RSL's semantics in Definition 3.2.

LEMMA 4.6. *Let (σ, h) be a state. Let a be an assertion of RSL, and F be its hypothetical footprint in state (σ, h). Then $\sigma, h \models_{RSL} a \Rightarrow F \subseteq dom(h)$.*

Proof: By induction on the structure of assertions.

Let a and (σ, h) be given. Let F be a's hypothetical footprint in (σ, h). Assume $\sigma, h \models_{RSL} a$. We proceed by induction on the structure of a.

One base case is when a is $e_1 = e_2$. By the semantics of RSL (Def. 4.4), each expression's footprint is an empty set, $\varnothing$. By set theory, $\varnothing \subseteq dom(h)$.

The second base case is when a is $x.f \mapsto e$. By the semantics of RSL (Def. 4.4), $dom(h) = \{(\mathcal{E}_{RSL}[\![x]\!]_h, f)\}$. And by definition, this is also the hypothetical footprint of a.

The inductive hypothesis is that for all subassertions a_i, the heap h, for each subassertion a_i, its hypothetical footprint, F_i, is a subset of *dom*(h).

The first inductive case is when a is of the form $a_1 * a_2$. By the semantics of RSL (Def. 3.2), there exists h_1 and h_2, such that $h_1 \cdot h_2 = h$. Let a_1's footprint be F_1, and a_2's footprint be F_2. Let us consider the set, $F_1 \cup F_2$. By inductive hypothesis, $F_1 \subseteq dom(h_1)$ and $F_2 \subseteq dom(h_2)$. Thus by set theory, $(F_1 \cup F_2) \subseteq (dom(h_1) \cup dom(h_2))$. By definition of heap (Def. 2.2), $dom(h_1) \cup dom(h_2) = dom(h)$. Therefore $(F_1 \cup F_2) \subseteq dom(h)$.

The second case is when a is of the form $a_1 \Rightarrow a_2$. Let a_1's footprint be F_1, and a_2's footprint be F_2. Let us consider the set, $F_1 \cup F_2$. By inductive hypothesis, $F_1 \subseteq dom(h)$ and $F_2 \subseteq dom(h)$. By set theory, $F_1 \cup F_2 \subseteq dom(h)$.

The third inductive case is when a is of the form $\exists x'.x.f \mapsto x' * a$. By the definition of hypothetical footprint, the footprint of existential assertions do not depend on the existential variables. So the result follows by the same reasoning as in the separating conjunction case.

The cases for conjunction and disjunction of assertions are similar. ∎

COROLLARY 4.7. *Let a_1 and a_2 be assertions in RSL, then $\forall(\sigma, h).\ \sigma, h \models_{RSL} a_1 * a_2 \Rightarrow \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cap \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h} = \varnothing$.*

Proof: Let σ and h be given. Assume $\sigma, h \models_{RSL} a_1 * a_2$. Then by semantics of RSL (Def. 3.2), there exists h_1 and h_2, such that $h_1 \perp h_2$. Thus by definition of heap (Def. 2.2), $dom(h_1) \cap dom(h_2) = \varnothing$. By lemma 4.6, we have that $\mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \subseteq dom(h_1)$ and also that $\mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h} \subseteq dom(h_2)$. Therefore, by set theory, they are disjoint. ∎

4.3.2 Results about the assertion translation

Now we prove the semantic meaning of RSL assertions is preserved by the syntactic mapping function, $\mathtt{TR}$. The key to this proof is showing that in a given state, the hypothetical footprint of a RSL assertion, a, is also the semantic footprint of its translated assertion, $\mathtt{TR}[\![a]\!]$. Then a's validity can be preserved in the translation by the definition of footprints. The proof also uses the following technical lemma.

LEMMA 4.8. *Let σ be a store, and h and H be heaps. Let a be a RSL assertion and $Assrt$ be a DafnyR assertion. If $\sigma, h \models_{RSL} a \iff \sigma, H \models_{DR} Assrt$, then $\mathcal{A}_{RSL}[\![a]\!]_{\sigma,h} = \mathcal{A}_{DR}[\![Assrt]\!]_{\sigma,H}$.*

Proof: For a given state (σ, h) and RSL assertion a, by the semantics of RSL, $\mathcal{A}_{RSL}[\![a]\!]_{\sigma,h}$ is *true*, if $\sigma, h \models_{RSL} a$, otherwise it is *false*. Similarly, for a given state (σ, H) and DafnyR assertion $Assrt$, by the semantics of DafnyR, $\mathcal{A}_{DR}[\![Assrt]\!]_{\sigma,H}$ is $true$, if $\sigma, H \models_{DR} Assrt$, otherwise it is *false*. Therefore, by assumption $\sigma, h \models_{RSL} a \iff \sigma, H \models_{DR} Assrt$, we can achieve the conclusion $\mathcal{A}_{RSL}[\![a]\!]_{\sigma,h} = \mathcal{A}_{DR}[\![Assrt]\!]_{\sigma,H}$. ∎

THEOREM 4.9. *Let a be an assertion in RSL. Let σ be a store, h and H be heaps, and $F = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a)]\!]_{\sigma,h}$. If $h \overset{F}{\equiv} H$, then $F = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a]\!])]\!]_{\sigma,H}$, and $\sigma, h \models_{RSL} a \iff \sigma, H \models_{DR} \mathtt{TR}[\![a]\!]$.*

Proof: We prove this theorem by induction on the structure of the assertion a. The proof is found in Appendix A. ∎

According to this theorem, a valid assertion in RSL is translated to the corresponding assertion in DanfyR that is also valid. Conversely, an invalid assertion in RSL is translated to the corresponding assertion in DafnyR that is also invalid. Thus, the translation preserves assertion validity.

4.3.3 Footprint of RSL

By Theorem 4.9, RSL assertions a and the translated assertions $\mathtt{TR}[\![a]\!]$ are semantically equivalent on the states (σ, h) and (σ, H), where h and H agree on a's hypothetical footprint. Therefore we can replace a with $\mathtt{TR}[\![a]\!]$ in the definition of the hypothetical footprint function for RSL.

DEFINITION 4.10 (Footprint of RSL). *The* footprint function for expressions *maps all expressions to the empty region: $\mathbf{fp}_{RSL}(e) = \mathbf{region}\{\}$. The* footprint function for assertions *maps assertions to regions, as shown in Fig. 13.*

4.4 Translation of Proofs

In this section, we explore a syntactical mapping on proof rules and show that it also preserves proofs.

4.4.1 Results about the proof translation

We consider mapping assertions and Hoare-tripes in RSL to those in DafnyR by syntactically translating assertions. The trouble is that the mapping for the field-update, field-acc and frame rules do not seem obvious. For example, according to definition 4.3, $\exists v.\ x.f \mapsto v$ in RSL maps to the predicate $\exists v.\ PointsTo_f(x, v)$ in DafnyR. However, UPD_{RSL} requires a precondition, $\exists v.\ x.f \mapsto v$, but UPD_{DR} requires a precondition, $x \neq null$, not the predicate $\exists v.\ PointsTo_f(x, v)$. Recall that $PointsTo_f(x, Expr')$ is defined as $x \neq null\ \&\&\ x.f = Expr'$. That entails $x \neq null$. Therefore we derive a new rule, $DUPD_{DR}$, shown in Fig. 14. Similarly, we relax the precondition for ACC_{DR}. But we encounter another trouble that ACC_{DR} seems to miss a corresponding postcondition, $x'.f \mapsto Expr$ of ACC_{RSL}. Actually, this is entailed by DafnyR's frame condition, $[\mathbf{region}\{\}]$, which means the heap is not changed by the statement, and the value of $x'.f$ is preserved before and after executing it. Therefore we can derive a relaxed proof rule $DACC_{DR}$ shown in Fig. 15. Note that in the last step, we change the frame to $\mathbf{region}\{x'.f\}$, this is justified because the postcondition specifies the desired value at location $\mathbf{region}\{x'.f\}$.

Finally we derive a relaxed frame rule $DFRM_{DR}$ shown in Fig. 16. Note that we put $\epsilon * \mathbf{fpt}(R) = \mathbf{region}\{\}$ as a side condition. Therefore, if the side condition appears in a proof translated form RSL, then it will hold in the translation.

Now we use the derived rules to define a syntactic mapping between RSL and DafnyR.

DEFINITION 4.11 (Syntactic Mapping from RSL to DafnyR). *Let P and Q be assertions in RSL. We define a syntactic mapping $\mathtt{TR}_{RSL}[\![-]\!]$ from RSL's assertions and Hoare-triples to DafnyR's as:*

$$
\begin{aligned}
&\mathtt{TR}_{RSL}[\![P]\!] \;=\; P \\
&\mathtt{TR}_{RSL}[\![\{P\}\ Stmt\ \{Q\}]\!] \;= \\
&\qquad \{\mathtt{TR}_{RSL}[\![P]\!]\}\ Stmt\ \{\mathtt{TR}_{RSL}[\![Q]\!]\}\ [\mathbf{fpt}(\mathtt{TR}[\![P]\!])].
\end{aligned}
$$

Let $h_1, \ldots, h_n$ be hypothesis and c be conclusion in RSL's inference rules and axioms. The syntactic mapping from them to DafnyR's rules and axioms are defined as:

$$
\mathtt{TR}_{RSL}[\![\frac{h_1,\ \ldots,\ h_n}{c}]\!] \;=\; \frac{\mathtt{TR}_{RSL}[\![h_1]\!],\ \ldots,\ \mathtt{TR}_{RSL}[\![h_n]\!]}{\mathtt{TR}_{RSL}[\![c]\!]}
$$

Preservation of Provability

Now we prove that proofs are preserved by the syntactic mapping $\mathtt{TR}_{RSL}[\![-]\!]$, i.e., proofs done in RSL can be converted into DafnyR proofs. This result gives in practice the ability to use existing approaches or decision procedures for RSL and apply them to the more general world of dynamic frames.

$$
\begin{array}{l}
\mathbf{fp}_{RSL}(e_1 = e_2) = \mathbf{region}\{\} \qquad \mathbf{fp}_{RSL}(x.f \mapsto e) = \mathbf{region}\{x.f\} \\
\mathbf{fp}_{RSL}(a_1 * a_2) = \mathbf{fp}_{RSL}(a_1) \ + \ \mathbf{if}\ \mathtt{TR}[\![a_1]\!]\ \mathbf{then}\ \mathbf{fp}_{RSL}(a_2)\ \mathbf{else}\ \mathbf{region}\{\} \\
\mathbf{fp}_{RSL}(a_1 \wedge a_2) = \mathbf{fp}_{RSL}(a_1) \ + \ \mathbf{if}\ \mathtt{TR}[\![a_1]\!]\ \mathbf{then}\ \mathbf{fp}_{RSL}(a_2)\ \mathbf{else}\ \mathbf{region}\{\} \\
\mathbf{fp}_{RSL}(a_1 \vee a_2) = \mathbf{fp}_{RSL}(a_1) \ + \ \mathbf{if}\ \mathtt{TR}[\![a_1]\!]\ \mathbf{then}\ \mathbf{region}\{\}\ \mathbf{else}\ \mathbf{fp}_{RSL}(a_2) \\
\mathbf{fp}_{RSL}(a_1 \Rightarrow a_2) = \mathbf{fp}_{RSL}(a_1) + \mathbf{if}\ \mathtt{TR}[\![a_1]\!]\ \mathbf{then}\ \mathbf{fp}_{RSL}(a_2)\ \mathbf{else}\ \mathbf{region}\{\} \\
\mathbf{fp}_{RSL}(\exists x'.x.f \mapsto x' * a) = \ \mathbf{region}\{x.f\} + \mathbf{fp}_{RSL}(a)[x.f/x'].
\end{array}
$$

Figure 13. Footprint function for RSL assertions

$$
(CON_{DR})\ \dfrac{
(CON_{DR})\ \dfrac{
\vdash \begin{array}{c} x.f = Expr \Rightarrow \\ x.f = Expr \end{array}
\quad
\vdash \begin{array}{c} PointsTo_f(x, Expr') \\ \Rightarrow\ x \neq null \end{array}
\quad
\begin{array}{c}(UPD_{DR})\\ \vdash_{DR} \begin{array}{c} \{x \neq null\} \\ x.f := Expr \\ \{x.f = Expr\} \\ [\mathbf{region}\{x.f\}] \end{array}\end{array}
}{
\vdash_{DR} \begin{array}{c} \{PointsTo_f(x, Expr')\} \\ x.f := Expr \\ \{x.f = Expr\}\ [\mathbf{region}\{x.f\}] \end{array}
}
\quad
\begin{array}{l}
\vdash \begin{array}{c} x \neq null\,\&\,\&\,x.f = Expr \\ \Rightarrow PointsTo_f(x, Expr) \end{array} \\
\vdash \begin{array}{c} x.f = Expr \Rightarrow \\ x.f = Expr \end{array}
\end{array}
}{
\vdash_{DR} \begin{array}{c} \{PointsTo_f(x, Expr')\} \\ x.f := Expr \\ \{PointsTo_f(x, Expr)\}\ [\mathbf{region}\{x.f\}] \end{array}
}
$$

Figure 14. Derivation of the $DUPD_{DR}$ rule

$$
(SubEff_{DR})\ \dfrac{
(FRM_{DR})\ \dfrac{
(CON_{DR})\ \dfrac{
\begin{array}{l}
\vdash \begin{array}{c} PointsTo_f(x', Expr) \iff \\ x' \neq null\ \&\&\ x'.f = Expr \end{array} \\
\vdash \begin{array}{c} x = Expr \Rightarrow \\ x = Expr \end{array}
\end{array}
\quad
\begin{array}{c}(ACC_{DR})\\ \vdash_{DR} \begin{array}{c} \{x' \neq null\ \&\&\ x'.f = Expr\} \\ x := x'.f \\ \{x = Expr\}\ [\mathbf{region}\{\}] \end{array}\end{array}
}{
\vdash_{DR} \begin{array}{c} \{PointsTo_f(x', Expr)\} \\ x := x'.f \\ \{x = Expr\}\ [\mathbf{region}\{\}] \end{array}
}
}{
\vdash_{DR} \begin{array}{c} \{PointsTo_f(x', Expr)\} \\ x := x'.f \\ \{x = Expr\ \&\&\ PointsTo_f(x', Expr)\} \\ [\mathbf{region}\{\}] \end{array}
}
\quad
\begin{array}{c} PointsTo_f(x', Expr) \vdash_{DR} \\ \mathbf{region}\{\} <= \mathbf{region}\{x'.f\} \end{array}
}{
\vdash_{DR} \begin{array}{c} \{PointsTo_f(x', Expr)\} \\ x := x'.f \\ \{x = Expr\ \&\&\ PointsTo_f(x', Expr)\} \\ [\mathbf{region}\{x'.f\}] \end{array}
}
$$

$$
\textbf{where } x \notin \mathtt{FV}(Expr) \textbf{ and } \{x\} \cap \mathtt{FV}(PointsTo_f(x', Expr)) = \varnothing
$$

Figure 15. Derivation of the $DACC_{DR}$ rule

$$
(SubEff_{DR})\ \dfrac{
(FRM_{DR})\ \dfrac{\vdash_{DR} \{P\}\ Stmt\ \{Q\}[\varepsilon]}{\vdash_{DR} \{P\ \&\&\ R\}\ Stmt\ \{Q\ \&\&\ R\}[\varepsilon]}
\quad
P \vdash \varepsilon <= \varepsilon + \mathbf{fpt}(R)
}{
\vdash_{DR} \{P\ \&\&\ R\}\ Stmt\ \{Q\ \&\&\ R\}[\varepsilon + \mathbf{fpt}(R)]
}
$$

$$
\begin{array}{l}
\textbf{where } (\mathtt{Modify}(Stmt) \cap \mathtt{FV}(R)) = \varnothing \\
\textbf{and } \varepsilon * \mathbf{fpt}(R) = \mathbf{region}\{\}
\end{array}
$$

Figure 16. Derivation of the $DFRM_{DR}$ rule

In the theorem below, the assumption $\vdash x \neq null \Rightarrow \exists v.PointsTo_f(x, v)$ is implicit in DafnyR, since DafnyR assumes arbitrary values for un-initialized variables.

THEOREM 4.12. *Assume that for all variables x and fields f of x's type, $\vdash x \neq null \Rightarrow \exists v.PointsTo_f(x, v)$. Let x be a triple in RSL, then if $\vdash_{RSL} x$, then $\vdash_{DR}$ $\mathtt{TR}_{RSL}[\![x]\!]$.*

Proof: The proof strategy is to syntactically translate the proof of x into DafnyR, using $\mathtt{TR}_{RSL}[\![-]\!]$ (Definition 4.11). Then we show in each case that the translated proof is a proof in DafnyR by induction.

Assume $\vdash_{RSL} x$. We prove it by induction on the structure of the RSL proof.

1. ($ALLOC$) One base case is when x has the form of $\{true\}x := \mathbf{new}K\{\circledast_{i=1}^{n} x.f_i \mapsto 0\}$, where $\{f_1, \ldots, f_n\} = fields(K)$.

$$\begin{array}{ll}
 & \mathtt{TR}_{RSL}[\![\{true\}\ x := \mathbf{new}\ K\ \{\circledast_{i=1}^{n} x.f_i \mapsto 0\}]\!] \\
= & \langle \text{by rule mapping (Def. 4.11)} \rangle \\
 & \mathtt{TR}[\![\{true\}]\!] \\
 & x := \mathbf{new}\ K \\
 & \mathtt{TR}[\![\{\circledast_{i=1}^{n} x.f_i \mapsto 0\}]\!][\mathbf{fpt}(\mathtt{TR}[\![true]\!])] \\
= & \langle \text{by assertion mapping (Def. 4.3)} \rangle \\
 & \{true\} \\
 & x := \mathbf{new}\ K \\
 & \{\&\&_{i=1}^{n} PointsTo_{f_i}(x, 0)\ \&\& \\
 & !!_{i=1}^{n}\mathbf{fpt}(PointsTo_{f_i}(x, 0))\}[\mathbf{fpt}(true)] \\
= & \langle \text{by semantics of RSL (Def. 2.6)} \rangle \\
 & \{true\} \\
 & x := \mathbf{new}\ K \\
 & \{\&\&_{i=1}^{n} PointsTo_{f_i}(x, 0)\ \&\& \\
 & !!_{i=1}^{n}\mathbf{fpt}(PointsTo_{f_i}(x, 0))\}[\mathbf{region}\{\}]
\end{array}$$

The translated form is a derived rule $DALLOC_{DR}$ shown in Fig. 17.

2. ($ASGN$) The second base case is when x has the form $\{true\}\ x := Expr\ \{x = Expr\}$

$$\begin{array}{ll}
 & \mathtt{TR}_{RSL}[\![\{true\}\ x := Expr\ \{x = Expr\}]\!] \\
= & \langle \text{by rule mapping (Def. 4.11)} \rangle \\
 & \{\mathtt{TR}[\![true]\!]\} \\
 & x := Expr \\
 & \{\mathtt{TR}[\![x = Expr]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![true]\!])] \\
= & \langle \text{by assertion mapping (Def. 4.3)} \rangle \\
 & \{true\} \\
 & x := Expr \\
 & \{x = Expr\}\ [\mathbf{fpt}(true)] \\
= & \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \{true\} \\
 & x := Expr \\
 & \{x = Expr\}\ [\mathbf{region}\{\}]
\end{array}$$

The form above is the $ASGN_{DR}$ rule in DafnyR.

3. (UPD) The third base case is when x has the form $\{\exists v.x.f \mapsto v\}\ x.f := Expr\ \{x.f \mapsto Expr\}$

$$\begin{array}{ll}
 & \mathtt{TR}_{RSL}[\![\begin{array}{c}\{\exists v.x.f \mapsto v\} \\ x.f := Expr \\ \{x.f \mapsto Expr\}\end{array}]\!] \\
= & \langle \text{by rule mapping (Def. 4.11)} \rangle \\
 & \{\mathtt{TR}[\![\exists v.x.f \mapsto v]\!]\} \\
 & x.f := Expr \\
 & \{\mathtt{TR}[\![x.f \mapsto Expr]\!]\} \\
 & [\mathbf{fpt}(\mathtt{TR}[\![x.f \mapsto Expr']\!])] \\
= & \langle \text{by assertion mapping 4.3} \rangle \\
 & \{\exists v.PointsTo_f(x, v)\} \\
 & x.f := Expr \\
 & \{PointsTo_f(x, Expr)\} \\
 & [\mathbf{fpt}(PointsTo_f(x, Expr'))] \\
= & \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \{\exists v.PointsTo_f(x, v)\} \\
 & x.f := Expr \\
 & \{PointsTo_f(x, Expr)\} \\
 & [\mathbf{region}\{x.f\}]
\end{array}$$

This form is the derived rule $DUPD_{DR}$ shown in Fig. 14.

4. (ACC) The fourth base case is when x has the form $\{x'.f \mapsto Expr\}\ x := x'.f\ \{x = Expr \wedge x'.f \mapsto Expr\}$

$$\begin{array}{ll}
 & \mathtt{TR}_{RSL}[\![\begin{array}{c}\{x'.f \mapsto Expr\} \\ x := x'.f \\ \{x = Expr\ \wedge\ x'.f \mapsto Expr\}\end{array}]\!] \\
= & \langle \text{by rule mapping (Def. 4.11)} \rangle \\
 & \{\mathtt{TR}[\![x'.f \mapsto Expr]\!]\} \\
 & x := x'.f \\
 & \{\mathtt{TR}[\![x = Expr\ \wedge\ x'.f \mapsto Expr]\!]\} \\
 & [\mathbf{fpt}(\mathtt{TR}[\![x'.f \mapsto Expr]\!])] \\
= & \langle \text{by assertion mapping (Def. 4.3)} \rangle \\
 & \{PointsTo_f(x', Expr)\} \\
 & x := x'.f \\
 & \{x = Expr\ \&\&\ PointsTo_f(x', Expr)\} \\
 & [\mathbf{fpt}(PointsTo_f(x', Expr))] \\
= & \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \{PointsTo_f(x', Expr)\} \\
 & x := x'.f \\
 & \{x = Expr\ \&\&\ PointsTo_f(x', Expr)\} \\
 & [\mathbf{region}\{x'.f\}]
\end{array}$$

The formula the derived rule $DACC_{DR}$ shown in Fig. 15.

Now we have proven all the base cases. Next we prove inductive cases. The inductive hypothesis is that for all hypothesis rules x, if $\vdash_{RSL} x$, then $\vdash_{DR}$ $\mathtt{TR}_{RSL}[\![x]\!]$.

5. (IF) In this case, x has the form

$$\frac{\begin{array}{c}\{P \wedge Expr \neq 0\}\ Stmt_1\ \{Q\}, \\ \{P \wedge Expr = 0\}\ Stmt_2\ \{Q\}\end{array}}{\{P\}\ \mathbf{if}(Expr \neq 0)\{Stmt_1\}\ \mathbf{else}\{Stmt_2\}\ \{Q\}}$$

By the inductive hypothesis, this is derivable in DafnyR.

$$
CON_{DR}\ \frac{\begin{array}{l}\vdash \begin{array}{l}(\&\&_{i=1}^{n} PointsTo_{f_i}(x,0)\,\&\,\& \\ \;!!_{i=1}^{n}\mathbf{fpt}(PointsTo_{f_i}(x,0))) \\ \;\Rightarrow \&\&_{i=1}^{n} PointsTo_{f_i}(x,0),\end{array} \\ \vdash true \Rightarrow true,\end{array} \qquad \begin{array}{l}(ALLOC_{DR}) \\ \vdash_{DR} \begin{array}{l}\{true\} \\ x := \mathbf{new}\, K \\ \{\&\&_{i=1}^{n} PointsTo_{f_i}(x,0) \\ \&\,\&(!!_{i=1}^{n}\mathbf{fpt}(PointsTo_{f_i}(x,0)))\}\ [\mathbf{region}\{\}]\end{array}\end{array}}{\begin{array}{l}\vdash_{DR} \{true\}\ x := \mathbf{new}\, K\ \{\&\&_{i=1}^{n} PointsTo_{f_i}(x,0)\}\ [\mathbf{region}\{\}] \\ \textbf{where } (f_1,\ldots,f_n) = \textit{fieldNames}(K)\end{array}}
$$

Figure 17. $DALLOC_{DR}$ rule

6. ($WHILE$) In this case, x is

$$\frac{\vdash_{SL} \{I \wedge Expr \neq 0\}\ Stmt\ \{I\}}{\vdash_{SL} \{I\}\ \mathbf{while}(Expr \neq 0)\{Stmt\}\ \{I \wedge Expr = 0\}}$$

   By the inductive hypothesis, this is derivable in DafnyR.

7. (SEQ) In this case x is

$$\frac{\vdash_{SL} \{P\}\ Stmt_1\ \{Q'\} \qquad \vdash_{SL} \{Q'\}\ Stmt_2\ \{Q\}}{\vdash_{SL} \{P\}\ Stmt_1; Stmt_2\ \{Q\}}$$

   By the inductive hypothesis, this is derivable in DafnyR.

8. ($CONSEQ$) In this case, x is

$$\frac{\vdash P \Rightarrow P' \qquad \vdash_{SL} \{P'\} Stmt\ \{Q'\} \qquad \vdash Q' \Rightarrow Q}{\vdash_{SL} \{P\}\ Stmt\ \{Q\}}$$

   By the inductive hypothesis, this is derivable in DafnyR.

9. (FRM) In this case, x is

$$\frac{\vdash_{SL} \{P\}\ Stmt\ \{Q\}}{\vdash_{SL} \{P * R\}\ Stmt\ \{Q * R\}}$$

   The calculation is shown in Fig. 18, where the condition $\mathbf{fpt}(\mathtt{TR}[\![P]\!])!!\mathbf{fpt}(\mathtt{TR}[\![Q]\!])$ satisfies the side-condition of DafnyR's frame rule, therefore, the rule above can be simplified as:

$$\frac{\{\mathtt{TR}[\![P]\!]\}\ Stmt\ \{\mathtt{TR}[\![Q]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![P]\!])]}{\begin{array}{c}\{\mathtt{TR}[\![P]\!]\,\&\,\&\,\mathtt{TR}[\![R]\!]\} \\ Stmt \\ \{\mathtt{TR}[\![Q]\!]\,\&\,\&\,\mathtt{TR}[\![R]\!] \\ [\mathbf{fpt}(\mathtt{TR}[\![P]\!])\ +\ \mathbf{fpt}(\mathtt{TR}[\![R]\!])]\end{array}}$$

   This formula is the derived rule, $DFRM_{DR}$, shown in Fig. 16.

∎

Conservatism of the Translation

According to Theorem 4.12, an axiom or provable Hoare-triple in RSL is translated to the corresponding axiom, provable Hoare-formula or provable derived Hoare-formula in DafnyR.

For the converse, we can combine theorem 4.9 with the soundness of DafnyR's verification logic to show that the translation cannot translate invalid Hoare triples in RSL into provable Hoare-formula in DafnyR.

THEOREM 4.13. *Suppose a_1 and a_2 are RSL assertions and $\{a_1\}$ Stmt $\{a_2\}$ is an invalid Hoare triple. Then its translation, $\mathtt{TR}_{RSL}[\![\{a_1\}\ Stmt\ \{a_2\}]\!]$, is not provable in DafnyR's verification logic.*

Proof: Let a_1 and a_2 be RSL assertions. Suppose that $\{a_1\}$ *Stmt* $\{a_2\}$ is invalid. By the semantics of partial correctness Hoare-triples (Def. 3.3) this means that there is some state σ, h such that $\sigma, h \models_{RSL} a_1$ and $(\sigma', h') = \mathcal{S}[\![Stmt]\!]_{\sigma,h}$, but $\sigma', h' \not\models_{RSL} a_2$. We will show that $\vdash_{DR} \mathtt{TR}_{RSL}[\![\{a_1\}\ Stmt\ \{a_2\}]\!]$ is a contradiction to the soundness of DafnyR's verification logic (see Appendix B). By definition, the translation is

$$\{\mathtt{TR}[\![a_1]\!]\}\ Stmt\ \{\mathtt{TR}[\![a_2]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])].$$

Let $F_1 = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h}$. By definition $h \stackrel{\mathrm{F_1}}{\equiv} h$, thus by Theorem 4.9, $F_1 = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,h}$ and $\sigma, h \models_{DR} \mathtt{TR}[\![a_1]\!]$. Let $F_2 = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma',h'}$. Since by definition $h' \stackrel{\mathrm{F_2}}{\equiv} h'$, and we are assuming that $\sigma', h' \not\models_{RSL} a_2$, by Theorem 4.9 again it follows that $\sigma', h' \not\models_{DR} \mathtt{TR}[\![a_2]\!]$. Thus the translation is invalid as a Hoare-formula. However, the DafnyR verification logic is sound [2], so this is a contradiction to the provability of the translation in DafnyR's verification logic. ∎

Therefore, our translation makes DafnyR's logic a conservative extension of RSL.

5. Discussion

In this section we discuss issues related to separation logic features.

5.1 Other Assertions in Standard Separation Logic

In section 3, we introduced a restricted separation logic (RSL) that excludes the *emp* predicate and separating implication. We discuss these excluded assertion forms in this section.

5.1.1 The emp predicate

The semantics of the emp predicate given by Reynolds [18] is: $\sigma, h \models emp \iff dom(h) = \emptyset$. This asserts that the heap, h, is empty. It is used in specifying memory allocation

$$\mathtt{TR}_{RSL}[\![\frac{\{P\}\ \mathit{Stmt}\ \{Q\}}{\{P * R\}\ \mathit{Stmt}\ \{Q * R\}}]\!]$$

$= \quad \langle$by rule mapping (Def. 4.11)$\rangle$

$$\frac{\{\mathtt{TR}[\![P]\!]\}\ \mathit{Stmt}\ \{\mathtt{TR}[\![Q]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![P]\!])]}{\{\mathtt{TR}[\![P * R]\!]\}\ \mathit{Stmt}\ \{\mathtt{TR}[\![Q * R]\!]\}}$$

$= \quad \langle$by assertion translation (Def. 4.3)$\rangle$

$$\frac{\{\mathtt{TR}[\![P]\!]\}\ \mathit{Stmt}\ \{\mathtt{TR}[\![Q]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![P]\!])]}{\begin{array}{c}\{\mathtt{TR}[\![P]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![P]\!])!!\mathbf{fpt}(\mathtt{TR}[\![Q]\!]))\}\ \mathit{Stmt}\quad \{\mathtt{TR}[\![Q]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![Q]\!])!!\mathbf{fpt}(\mathtt{TR}[\![R]\!]))\}\\ [\mathbf{fpt}(\mathtt{TR}[\![P]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![P]\!])!!\mathbf{fpt}(\mathtt{TR}[\![R]\!])))]\end{array}}$$

$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$

$$\frac{\{\mathtt{TR}[\![P]\!]\}\ \mathit{Stmt}\ \{\mathtt{TR}[\![Q]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![P]\!])]}{\begin{array}{c}\{\mathtt{TR}[\![P]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![P]\!])!!\mathbf{fpt}(\mathtt{TR}[\![Q]\!]))\}\ \mathit{Stmt}\quad \{\mathtt{TR}[\![Q]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![Q]\!])!!\mathbf{fpt}(\mathtt{TR}[\![R]\!]))\}\\ [\mathbf{fpt}(\mathtt{TR}[\![P]\!]\,\&\&\,\mathtt{TR}[\![R]\!])]\end{array}}$$

$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$

$$\frac{\{\mathtt{TR}[\![P]\!]\}\ \mathit{Stmt}\ \{\mathtt{TR}[\![Q]\!]\}[\mathbf{fpt}(\mathtt{TR}[\![P]\!])]}{\begin{array}{c}\{\mathtt{TR}[\![P]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![P]\!])!!\mathbf{fpt}(\mathtt{TR}[\![Q]\!]))\}\ \mathit{Stmt}\quad \{\mathtt{TR}[\![Q]\!]\,\&\&\,\mathtt{TR}[\![R]\!]\,\&\&\,(\mathbf{fpt}(\mathtt{TR}[\![Q]\!])!!\mathbf{fpt}(\mathtt{TR}[\![R]\!]))\}\\ [\mathbf{fpt}(\mathtt{TR}[\![P]\!])\ +\ \mathbf{fpt}(\mathtt{TR}[\![R]\!])]\end{array}}$$

Figure 18. Calculation on Frame formulas

and deallocation as in the following axioms:

$$\{emp\}\ x := \mathbf{new}\ K\ \{\circledast_{i=1}^{n} x.f_i \mapsto 0\} \tag{2}$$

$$\{\circledast_{i=1}^{n} x.f_i \mapsto 0\}\ \mathbf{free}\ x\ \{emp\} \tag{3}$$

In definition 3.4, we defined the $ALLOC_{RSL}$ rule as

$$\{true\}\ x := \mathbf{new}\ K\ \{\circledast_{i=1}^{n} x.f_i \mapsto 0\} \tag{4}$$

When specifying heap allocation, the precondition of (2) is stronger than that of (4). However, this is not problematic, since in practice, emp will be applied to an empty heap, and both emp and $true$ will hold for an empty heap.

However, the story is different for deallocation, which in the verification logic of DafnyR could be specified as:

$$\{\circledast_{i=1}^{n} x.f_i \mapsto 0\}\ \mathbf{free}\ (x)\ \{true\} \tag{5}$$

Note that the postcondition of (3) is stronger than the postcondition of (5). This shows an advantage of separation logic over dynamic frames. Consider a method, $dispose(lst)$ that disposes a linked-list, lst, by iteratively freeing each node in lst. The postcondition of (5) does not have any proof obligation. That means it is always satisfied even if a statement in the implementation of $dispose$ does not free some nodes in lst. But such an incorrect implementation could not be verified in RSL using (3), which specifies that the storage must be deallocated.

Because of the semantics of DafnyR, which works with the entire heap, not just the part requested by a precondition, DafnyR lacks the expressiveness to encode emp.

5.1.2 Separating implication

The separating implication (or "magic wand") operator poses problems for our translation, because we are unable to determine a suitable footprint for it. The semantics of separating implication assertions is [18]:

$$\begin{array}{rl}\sigma, h \models a_1 \mathrel{-\!\!*} a_2 \iff & \forall h'.(h' \perp h\ and\ \sigma, h' \models a_1)\\ & implies\ \sigma, h \cdot h' \models a_2)\end{array}$$

The trouble with creating a definition of the footprint of such an assertion is that the footprint of the antecedent (a_1) is not necessarily a subset of the domain of the current heap. But that would contradict Lemma 4.6, which says that in any state, the footprint should be a subset of the current heap's domain.

5.2 Intuitionistic semantics of SL

We defined the semantics of RSL classically [18]. Separation logic can also be given an intuitionistic semantics [8]. In the intuitionistic semantics, *emp* is omitted, and there is a monotonicity condition: which says that if $\forall\, h', h : h \subseteq h' : \sigma, h \models a \Rightarrow \sigma, h' \models a$. Furthermore, the semantics of point-to assertions and implication are defined as follows:

$$\begin{array}{l}\sigma, h \models_{SL} x.f \mapsto e \iff \{(\mathcal{E}_{RSL}[\![x]\!]_\sigma, f)\} \in dom(h)\\ \qquad\qquad and\ \mathcal{E}_{RSL}[\![x]\!]_\sigma \neq null\ and\\ \qquad\qquad h[\mathcal{E}_{RSL}[\![x]\!]_\sigma, f] = \mathcal{E}_{RSL}[\![x]\!]_\sigma\\ \sigma, h \models_{SL} a_1 \Rightarrow a_2 \iff \forall h' : h' \supseteq h :\\ \qquad\qquad \sigma, h' \models a_1 implies\ \sigma, h' \models a_2\end{array}$$

Since semantic footprints are minimal sets of locations, the points-to assertions cause no problems in this semantics.

However, the intuitionistic semantics of implication assertions is similar to the semantics of magic wand discussed in section 5.1.2. Thus we are unable to extend our result to this semantics.

5.3 Encoding Ramifications

Hobor and Villard [7] extend separation logic with *overlapping conjunctions*, of the form $a_1 \circledast a_2$, which are use the "ramification" ($\circledast$) operator. They define the semantics of such assertions as follows:

$$\begin{aligned}\sigma, h \models_{SL} a_1 \circledast a_2 \iff & \; exists\, h_1, h_2, h_3 . h_1 \bot h_2 \bot h_3 \\ & and\, h_1 \cdot h_2 \cdot h_3 = h \; and\; \sigma, h_1 \cdot h_2 \models a_1 \; and \\ & \sigma, h_2 \cdot h_3 \models a_2.\end{aligned}$$

Overlapping conjunction can be used to express assertions about shared data structures.

Ramifications can be added to RSL without causing problems with our results. This can be done by extending the definition of hypothetical footprint as follows.

DEFINITION 5.1. *The hypothetical footprint for a ramification assertion is given by:*

$$\begin{aligned}\mathbf{fp}_{Hy}(a_1 \circledast a_2) = \quad & \mathbf{fp}_{Hy}(a_1) + \mathbf{if}\; a_1 \;\mathbf{then} \\ & \mathbf{fp}_{Hy}(a_2) \;\mathbf{else\; region}\{\}.\end{aligned}$$

The translation into DafnyR assertions would also be simple:

DEFINITION 5.2.

$$\mathtt{TR}[\![a_1 \circledast a_2]\!] = \mathtt{TR}[\![a_1]\!] \,\&\&\, \mathtt{TR}[\![a_2]\!].$$

We can then adapt our proofs and show that the semantics of assertions is preserved by this translation.

LEMMA 5.3. *Let a_1 and a_2 be an assertions in RSL, and (σ, h) and (σ, H) be states. Let F be $a_1 \circledast a_2$'s hypothetical footprint $F = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1 \circledast a_2)]\!]_{\sigma,H}$. If $h \stackrel{F}{\equiv} H$, then $F = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1 \circledast a_2]\!])]\!]_{\sigma,H}$ and $\sigma, h \models_{RSL} a_1 \circledast a_2 \iff \sigma, H \models_{DR} \mathtt{TR}[\![a_1 \circledast a_2]\!]$.*

Proof: We consider it as another inductive case in the proof of Theorem 4.9 . The inductive hypothesis is that for all subassertions a_i, heaps h_i and H', for each subassertion a_i, the footprint is $F_i = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_i)]\!]_{\sigma,H'}$. If $h \stackrel{F_i}{\equiv} H'$, then $F_i = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_i]\!])]\!]_{\sigma,H'}$, and $\sigma, h_i \models_{SL} a_i \iff \sigma, H' \models_{DR} \mathtt{TR}[\![a_i]\!]$.

We first prove $\mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1 \circledast a_2]\!])]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1 \circledast a_2)]\!]_{\sigma,H}$ as follows:

$$\begin{aligned}
& \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1 \circledast a_2)]\!]_{\sigma,h} \\
= \; & \langle \text{by the hypothetical footprint (Def. 5.4)} \rangle \\
& \mathcal{RE}\left[\!\!\left[\begin{array}{l} \mathbf{fp}_{Hy}(a_1) + \mathbf{if}\; a_1 \;\mathbf{then}\; \mathbf{fp}_{Hy}(a_2) \\ \mathbf{else\; region}\{\} \end{array} \right]\!\!\right]_{\sigma,h} \\
= \; & \langle \text{by the presumed semantics (formula (1)), twice} \rangle \\
& \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup \mathbf{if}\; \mathcal{A}_{RSL}[\![a_1]\!]_{\sigma,h} = true \\
& \mathbf{then}\; \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h} \mathbf{else}\; \emptyset \\
= \; & \left\langle \begin{array}{l} \text{by inductive hypothesis, } \sigma, h \models_{RSL} a_i \iff \\ \sigma, H \models_{DR} \mathtt{TR}[\![a_i]\!] \text{, and Lemma 4.8} \end{array} \right\rangle \\
& \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup \\
& \mathbf{if}\; \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = true \;\mathbf{then} \\
& \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \;\mathbf{else}\; \emptyset \\
= \; & \left\langle \begin{array}{l} \text{by inductive hypothesis,} \\ \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_i)]\!]_{\sigma,h} = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_i]\!])]\!]_{\sigma,H}, \\ \text{twice} \end{array} \right\rangle \\
& \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup \\
& \mathbf{if}\; \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = true \;\mathbf{then} \\
& \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H} \;\mathbf{else}\; \emptyset \\
= \; & \langle \text{by semantics of DafnyR (Def. 2.6), twice} \rangle \\
& \mathcal{RE}\left[\!\!\left[\begin{array}{l} \mathbf{fpt}(\mathtt{TR}[\![a_1]\!]) + (\mathbf{if}\; \mathtt{TR}[\![a_1]\!] \;\mathbf{then} \\ \mathbf{fpt}(\mathtt{TR}[\![a_2]\!]) \;\mathbf{else\; region}\{\}) \end{array} \right]\!\!\right]_{\sigma,H} \\
= \; & \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
& \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!] \&\& \mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H} \\
= \; & \langle \text{by syntactic mapping (Def. 5.2)} \rangle \\
& \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1 \circledast a_2]\!])]\!]_{\sigma,H}
\end{aligned}$$

Next we prove $\sigma, h \models_{RSL} a_1 \circledast a_2 \iff \sigma, H \models_{DR} \mathtt{TR}[\![a_1 \circledast a_2]\!]$.

We first prove it from the left side to the right side. Assume $\sigma, h \models_{RSL} a_1 \circledast a_2$. By the semantics of ramification, there exists three disjoint sub-heaps h_1, h_2, h_3, such that a_1 asserts $h_1 \cdot h_2$, and a_2 asserts on $h_2 \cdot h_3$. Therefore $F_1 = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h_1 \cdot h_2}$ and $F_2 = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h_2 \cdot h_3}$. We calculate as follows:

$$\begin{aligned}
& \sigma, h \models_{RSL} a_1 \circledast a_2 \\
\iff & \langle \text{by semantics of ramification} \rangle \\
& exists\, h_1, h_2, h_3 . h_1 \bot h_2 \bot h_3 \; and\; h_1 \cdot h_2 \cdot h_3 = h \; and \\
& \sigma, h_1 \cdot h_2 \models_{SL} a_1 \; and\; \sigma, h_2 \cdot h_3 \models_{SL} a_2 \\
\Rightarrow \; & \langle \text{by Theorem 4.9} \rangle \\
& \sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!] \; and\; \sigma, H \models_{DR} \mathtt{TR}[\![a_2]\!] \\
\iff & \langle \text{by semantics of DafnyR (Def: 2.6)} \rangle \\
& \sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!] \;\&\&\; \mathtt{TR}[\![a_2]\!] \\
\iff & \langle \text{by syntactical mapping (Def. 5.2)} \rangle \\
& \sigma, H \models_{DR} \mathtt{TR}[\![a_1 \circledast a_2]\!]
\end{aligned}$$

Next we prove it from the right side to the left side. Assume $\sigma, H \models_{DR} \mathtt{TR}[\![a_1 \circledast a_2]\!]$, where the footprints of a_1 is $F_1 = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H}$, and the footprint of a_2 is $F_2 = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}$. We calculate it as follows:

$$\begin{aligned}
& \sigma, H \models_{DR} \mathtt{TR}[\![a_1 \circledast a_2]\!] \\
\iff & \langle \text{by syntactical mapping (Def. 5.2)} \rangle \\
& \sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!] \;\&\&\; \mathtt{TR}[\![a_2]\!] \\
\iff & \langle \text{by semantics of DafnyR (Def: 2.6)} \rangle \\
& \sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!] \; and\; \sigma, H \models_{DR} \mathtt{TR}[\![a_2]\!] \\
\Rightarrow \; & \left\langle \begin{array}{l} \text{by definition of } F_1 \text{ and } F_2 \text{, we construct heaps} \\ h_1, h_2 \text{ and } h_3 \text{, such that } dom(h_2) = F_1 \cap F_2, \\ dom(h_1) = F_1 - dom(h_2) \text{ and } dom(h_3) = \\ F_2 - dom(h_2) \text{ and } h_1 \cdot h_2 \stackrel{F_1}{\equiv} H \text{ and } h_2 \cdot h_3 \stackrel{F_2}{\equiv} H \end{array} \right\rangle
\end{aligned}$$

$$
\begin{array}{ll}
& \mathit{exists}\, h_1, h_2, h_3.\, F_1 = \mathit{dom}(h_1) \cup \mathit{dom}(h_2)\ \mathit{and} \\
& F_2 = \mathit{dom}(h_2) \cup \mathit{dom}(h_3)\ \mathit{and}\ \sigma, H \vDash_{DR} \mathtt{TR}[\![a_1]\!] \\
& \mathit{and}\ \sigma, H \vDash_{DR} \mathtt{TR}[\![a_2]\!]\ \mathit{and}\ \sigma, h_1 \cdot h_2 \vDash_{DR} \mathtt{TR}[\![a_1]\!] \\
& \mathit{and}\ \sigma, h_2 \cdot h_3 \vDash_{DR} \mathtt{TR}[\![a_2]\!] \\
\iff & \langle \text{by set theory} \rangle \\
& \mathit{exists}\, h_1, h_2, h_3.\, \mathit{dom}(h_1) \cap \mathit{dom}(h_2) = \varnothing\ \mathit{and} \\
& \mathit{dom}(h_2) \cap \mathit{dom}(h_3) = \varnothing\ \mathit{and} \\
& \mathit{dom}(h_1) \cap \mathit{dom}(h_3) = \varnothing\ \mathit{and} \\
& F_1 = \mathit{dom}(h_1) \cup \mathit{dom}(h_2)\ \mathit{and} \\
& F_2 = \mathit{dom}(h_2) \cup \mathit{dom}(h_3)\ \mathit{and} \\
& \sigma, H \vDash_{DR} \mathtt{TR}[\![a_1]\!]\ \mathit{and}\ \sigma, H \vDash_{DR} \mathtt{TR}[\![a_2]\!]\ \mathit{and} \\
& \sigma, h_1 \cdot h_2 \vDash_{DR} \mathtt{TR}[\![a_1]\!]\ \mathit{and}\ \sigma, h_2 \cdot h_3 \vDash_{DR} \mathtt{TR}[\![a_2]\!] \\
\iff & \langle \text{by definition of heap (Def. 2.2)} \rangle \\
& \mathit{exists}\, h_1, h_2, h_3.\, h_1 \bot h_2 \bot h_3\ \mathit{and} \\
& F_1 = \mathit{dom}(h_1) \cup \mathit{dom}(h_2)\ \mathit{and} \\
& F_2 = \mathit{dom}(h_2) \cup \mathit{dom}(h_3)\ \mathit{and}\ \sigma, H \vDash_{DR} \mathtt{TR}[\![a_1]\!] \\
& \mathit{and}\ \sigma, H \vDash_{DR} \mathtt{TR}[\![a_2]\!]\ \mathit{and} \\
& \sigma, h_1 \cdot h_2 \vDash_{DR} \mathtt{TR}[\![a_1]\!]\ \mathit{and}\ \sigma, h_2 \cdot h_3 \vDash_{DR} \mathtt{TR}[\![a_2]\!] \\
\iff & \langle \text{by Theorem 4.9, twice} \rangle \\
& \mathit{exists}\, h_1, h_2, h_3.\, h_1 \bot h_2 \bot h_3 \\
& F_1 = \mathit{dom}(h_1) \cup \mathit{dom}(h_2)\ \mathit{and} \\
& F_2 = \mathit{dom}(h_2) \cup \mathit{dom}(h_3)\ \mathit{and} \\
& \sigma, h_1 \cdot h_2 \vDash_{DR} \mathtt{TR}[\![a_1]\!]\ \mathit{and} \\
& \sigma, h_2 \cdot h_3 \vDash_{RSL} \mathtt{TR}[\![a_2]\!]\ \mathit{and}\ \sigma, h'_1 \vDash_{RSL} a_1\ \mathit{and} \\
& \sigma, h'_2 \vDash_{RSL} a_2 \\
\Rightarrow & \left\langle \begin{array}{l}\text{by Theorem 4.9, twice. And } h_1 \cdot h_2 \overset{F_1}{\equiv} h'_1 \text{ and} \\ h_2 \cdot h_3 \overset{F_2}{\equiv} h'_2\end{array} \right\rangle \\
& \mathit{exists}\, h_1, h_2, h_3.\, h_1 \bot h_2 \bot h_3\ \mathit{and} \\
& \sigma, h_1 \cdot h_2 \vDash_{RSL} a_1\ \mathit{and}\ \sigma, h_2 \cdot h_3 \vDash_{RSL} a_2 \\
\Rightarrow & \langle \text{by construction } h = h_1 \cdot h_2 \cdot h_3 \rangle \\
& \mathit{exists}\, h_1, h_2, h_3.\, h_1 \bot h_2 \bot h_3\ \mathit{and}\ h = h_1 \cdot h_2 \cdot h_3 \\
& \mathit{and}\ \sigma, h_1 \cdot h_2 \vDash_{RSL} a_1\ \mathit{and}\ \sigma, h_2 \cdot h_3 \vDash_{RSL} a_2 \\
\iff & \langle \text{by semantics of ramification} \rangle \\
& \sigma, h \vDash_{RSL} a_1 \circledast a_2
\end{array}
$$

∎

By the result of Theorem 4.9 and Lemma 5.3, we can conclude RSL assertions a and the translated assertions $\mathtt{TR}[\![a]\!]$ are semantically equivalent on the states (σ, h) and (σ, H), where h and H agree on a's footprint. Thus, following our earlier development, we can replace a with $\mathtt{TR}[\![a]\!]$ in definition 5.1, and redefine the footprint of ramification assertion in terms of region expressions in DafnyR's syntax.

DEFINITION 5.4. *The definition for ramification assertions' footprints is:*

$$
\begin{array}{rl}
\boldsymbol{fp}_{RSL}(a_1 \circledast a_2) = & \boldsymbol{fp}_{RSL}(a_1) + \textbf{\textit{if}}\ \mathit{TR}[\![a_1]\!]\ \textbf{\textit{then}} \\
& \boldsymbol{fp}_{RSL}(a_2)\ \textbf{\textit{else region}}\{\}
\end{array}
$$

5.4 Translation of DafnyR to RSL

In this section, we show our attempt to encode dynamic frames by translating DafnyR to the restricted separation logic.

DafnyR uses specification-only or ghost variables with type **`region`** to dynamically calculate frames as a program proceeds. This calculation can also be achieved by a pure method, such as in Smans' work [20], which returns a set of locations. Such a method is analogous to a DafnyR function that returns a region.

In the translation, we assume RSL also has predicates. We translate predicate invocations and declarations separately.

DEFINITION 5.5. *(Syntactic Mapping DafnyR to Separation Logic). Let DafnyR expressions and assertions be given in definition 2.1. We define a syntactic mapping* $\mathit{TR}_s[\![-]\!]$ *from DafnyR expressions and assertions to separation logic expressions and assertions shown in Fig. 19.*

Note that the translation of region expression, **`alloc`**, is not clear. And a region union expression could also be translated as: $\mathtt{TR}_s[\![RE_1 + RE_2]\!] = \mathtt{TR}_s[\![RE_1]\!] \circledast \mathtt{TR}_s[\![RE_2]\!]$, using ramification.

However, as mentioned in the background, the region expressions given in definition 2.1 are only subset of DafnyR's region expressions. We omit some region expressions that allow one to manipulate a region in a first class way.

$$
\begin{array}{l}
\mathit{RE}\ ::=\ \ldots \\
\quad\ \mid\ \mathbf{filter}\{\mathit{RE},\ K\}\ \mid\ \mathbf{filter}\{\mathit{RE},\ K,\ f\} \\
\mathit{Assrt}\ ::=\ \ldots \\
\quad\ \mid\ \exists x \in \mathit{RE}.\mathit{Assrt}\ \mid\ \mathbf{fresh}\{\mathrm{RE}\}\ \mid\ \mathbf{old}\{\mathit{Expr}\}
\end{array}
$$

It is not clear how to translate these other expressions and assertions to separation logic, since separation logic assertions couples locations and their contents, and do not provide a way to extract locations or to express types. Moreover, the dynamic frames technique commonly declares region variables as ghost fields or ghost variables. These seem difficult to translate into separation logic.

In a SMT based verifier, such as Dafny, DafnyR and VERL [19], method calls are verified with respect to the called method's specification. At the method call site, the method's precondition is checked, and the locations specified in the frame condition are allowed to take on arbitrary values (with havoc), then its postcondition is assumed. Therefore one must always gives desirable values to those havoced locations in its postcondition. If the frame condition is precise, which means it specifies a minimal set of locations that may be changed, then one mentions fewer locations in the postcondition, compared to less precise frame condition, which make one specify post-state properties of a bigger set of locations. In other words, if the frame contains more than the necessary locations, one needs to specify that values in those unnecessary locations are preserved. That could be done by old expression in Dafny, DafnyR and VERL or by logical variables in VeriFast. Therefore although these additional expressions provide a way to minimize the locations in the frame condition, they do not necessarily increase DafnyR's expressiveness.

$$\mathtt{TR}_s[\![x]\!] = x \quad \mathtt{TR}_s[\![null]\!] = null \quad \mathtt{TR}_s[\![n]\!] = n$$
$$\mathtt{TR}_s[\![Expr_1 = Expr_2]\!] = \begin{cases} \mathtt{TR}_s[\![x]\!].f \mapsto \mathtt{TR}_s[\![Expr_2]\!] & \text{if } Expr_1 = x.f \\ \mathtt{TR}_s[\![Expr_1]\!] = \mathtt{TR}_s[\![Expr_2]\!] & \text{otherwise} \end{cases}$$
$$\mathtt{TR}_s[\![Assrt_1 \&\& Assrt_2]\!] = \mathtt{TR}_s[\![Assrt_1]\!] \wedge \mathtt{TR}_s[\![Assrt_2]\!] \quad \mathtt{TR}_s[\![Assrt_1 || Assrt_2]\!] = \mathtt{TR}_s[\![Assrt_1]\!] \vee \mathtt{TR}_s[\![Assrt_2]\!]$$
$\mathtt{TR}_s[\![Assrt_1 \Rightarrow Assrt_2]\!] = \mathtt{TR}_s[\![Assrt_1]\!] \Rightarrow \mathtt{TR}_s[\![Assrt_2]\!] \quad \mathtt{TR}_s[\![\exists x.a]\!] = \exists x.\mathtt{TR}_s[\![a]\!] * true$, where x is not a region variable.
$\mathtt{TR}_s[\![P(ins)]\!] = P(\mathtt{TR}_s[\![ins]\!])$, where we overload $\mathtt{TR}_s[\![-]\!]$ for lists of actual arguments,ins.
$\mathtt{TR}_s[\![P(decls)\{Assrt\}]\!] = P(\mathtt{TR}_s[\![decls]\!]\{\mathtt{TR}[\![Assrt]\!]\})$, where we overload $\mathtt{TR}_s[\![-]\!]$ for lists of declarations, $decls$.
$$\mathtt{TR}_s[\![\mathbf{region}\{x.f\}]\!] = \exists x'.(\mathtt{TR}_s[\![x]\!].f \mapsto x' * true) \quad \mathtt{TR}_s[\![RE_1 + RE_2]\!] = \mathtt{TR}_s[\![RE_1]\!] * (\mathtt{TR}_s[\![RE_1]\!] -\!\!* \mathtt{TR}_s[\![RE_2]\!])$$
$$\mathtt{TR}_s[\![RE_1 * RE_2]\!] = (\mathtt{TR}[\![RE_2]\!] -\!\!* \mathtt{TR}[\![RE_1]\!]) -\!\!* \mathtt{TR}[\![RE_1]\!] \quad \mathtt{TR}_s[\![RE_1 !! RE_2]\!] = \mathtt{TR}_s[\![RE_1]\!] * \mathtt{TR}_s[\![RE_2]\!]$$

Figure 19. Syntactic mapping from DafnyR expressions and assertions to RSL's

6. Related Work

In this section, we discuss related work.

6.1 Dynamic Frames

The theory of dynamic frames is due to Kassios [9, 10]. The theory is based on sets of locations, as in DafnyR, so our translation from separation logic could perhaps also be adapted to target other verification systems that use dynamic frames [20]. These works do not show how to translate separation logic into the dynamic frames technique.

6.2 Dafny

Leino's Dafny system [13, 14] adopts the dynamic frames technique, but uses variables that store sets of objects. In Dafny it is not easy to specify frame properties at the level of locations (fields), instead one must strengthen postconditions, by using *old* expressions to specify which fields of threatened objects must not change. DafnyR can specify frames at the level of locations directly.

Because in Dafny one writes frame conditions using sets of objects, it would be difficult to precisely translate separation logic's points-to assertions into a predicate. By contrast, since DafnyR has regions that are sets of locations, it is easy to specify the frame conditions of DafnyR's built-in $PointsTo_f$ predicates.

6.3 Region Logic and VERL

The region logic of Banerjee, Naumann, and Rosenberg [1] is the source of DafnyR's region expressions. Region logic defines regions as sets of objects. But region logic can use **wr** (writes) and **rd** (reads) clauses to specify frame properties at the granularity of individual fields. Hence, as in DafnyR a points-to assertion could be translated using predicates with precise frames. However, it would be more difficult to deal with the translation of separating conjunction, because in region logic one cannot directly express disjointness of regions that contain locations. Expressing such tests directly on fine-grained regions is an advantage of DafnyR.

Rosenberg also defined a tool based on Dafny, VERL [19], that adds region logic to Dafny. Like Dafny and region logic, VERL uses sets of objects for regions. Furthermore, that work did not address the connection between region logic and separation logic.

6.4 Parkinson and Summers

Recently Parkinson and Summers [17], have shown a relationship between separation logic and the methodology of implicit dynamic frames as used in concurrent languages such as Chalice [16]. The methodology of implicit dynamic frames for such languages uses permissions [4]. Parkinson and Summers used "permission masks" to derive the partial heaps used in the semantics of separation logic from the permissions specified in the implicit dynamic frames technique. They use a Total Heaps Permission Logic to bridge the gap between the two logics. Our work was inspired by their approach. Instead of using permissions, DafnyR uses fine-grained regions containing locations, but these regions also can be thought of as determining partial heaps. The work of Parkinson and Summers is based on the intuitionistic semantics of separation logic [8], while ours is based on the more expressive classical semantics [18]. Moreover, their work did not present the connection between separation logic and the dynamic frames technique.

For the connection between implicit dynamic frames and dynamic frames, one can consider a location (o, f) with a positive permission in Parkinson and Summers' work as a singleton region $\{(o, f)\}$. In general a partial heap obtained by a permission mask can be obtained by the corresponding region. In this way one can draw many connections between their work and our work on DafnyR. On the other hand, their work does not use conditional permissions, which would be the analogue of DafnyR's conditional region expressions, and they did not show that their translation preserves proofs of correctness, as we have done.

7. Conclusion

We have shown that a restricted form of separation logic can be translated into a fine-grained region logic in a way that preserves the validity of assertions and proofs of partial correctness. The translation is precise in the sense that it translates invalid separation logic assertions into invalid region logic assertions. The translation is based on a semantic notion of footprint, which we have shown can be computed statically, due to the use of conditional region expressions. Thus DafnyR's fine-grained region logic can be used to write specifications both the style of separation logic and in the style of the dynamic frames technique.

Future work includes relaxing the restrictions on the form of separation logic used in the technical results. In particular we would like to treat separating implication (or equivalently, separation logic's intitionistic semantics).

Future work includes incorporating these ideas into JML.

A prototype DafnyR system can be obtained from `http://dafnyr.codeplex.com`.

Appendix

A. Proof of Theorem 4.9

Theorem 4.9 is as follows.

Theorem 4.9: Let a be an assertion in RSL. Let σ be a store, h and H be heaps, and $F = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a)]\!]_{\sigma,h}$. If $h \overset{\mathrm{F}}{\equiv} H$, then $F = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a]\!])]\!]_{\sigma,H}$, and $\sigma, h \models_{RSL} a \iff \sigma, H \models_{DR} \mathtt{TR}[\![a]\!]$.

Proof: Assume $h \overset{\mathrm{F}}{\equiv} H$. We prove the theorem by induction on the structure of the assertion a.

One base case is when a is $e_1 = e_2$.

We first prove $\mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![e_1 = e_2]\!])]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{fp}_{Hy}(e_1 = e_2)]\!]_{\sigma,h}$ as follows:

$$
\begin{array}{ll}
 & \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![e_1 = e_2]\!])]\!]_{\sigma,H} \\
= & \quad \langle \text{by syntactic mapping (Def. 4.3)} \rangle \\
 & \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![e_1]\!] = \mathtt{TR}[\![e_2]\!])]\!]_{\sigma,H} \\
= & \quad \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![e_1]\!]) + \mathbf{fpt}(\mathtt{TR}[\![e_2]\!])]\!]_{\sigma,H} \\
= & \quad \langle \text{by semantics of DafnyR (Def. 2.6), twice} \rangle \\
 & \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,H} \\
= & \quad \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \varnothing \\
= & \quad \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \mathcal{RE}[\![\mathbf{region}\{\}]\!]_{\sigma,h} \\
= & \quad \langle \text{by footprint in RSL (Def. 4.4)} \rangle \\
 & \mathcal{RE}[\![\mathbf{fp}_{Hy}(e_1 = e_2)]\!]_{\sigma,h}
\end{array}
$$

Next we prove $\sigma, h \models_{RSL} e_1 = e_2 \iff \sigma, H \models_{DR} \mathtt{TR}[\![e_1 = e_2]\!]$ as follows:

$$
\begin{array}{ll}
 & \sigma, h \models_{RSL} e_1 = e_2 \\
\iff & \langle \text{by semantics of RSL in Definition 3.2} \rangle \\
 & \mathcal{E}_{RSL}[\![e_1]\!]_\sigma = \mathcal{E}_{RSL}[\![e_2]\!]_\sigma \\
\iff & \langle \text{by lemma 4.2} \rangle \\
 & \mathcal{E}_{DR}[\![e_1]\!]_{\sigma,H} = \mathcal{E}_{DR}[\![e_2]\!]_{\sigma,H} \\
\iff & \langle \text{by syntactic mapping (Def. 4.1), twice} \rangle \\
 & \mathcal{E}_{DR}[\![\mathtt{TR}[\![e_1]\!]]\!]_{\sigma,H} = \mathcal{E}_{DR}[\![\mathtt{TR}[\![e_2]\!]]\!]_{\sigma,H} \\
\iff & \langle \text{by semantics of DafnyR (Def. 2.6)} \rangle \\
 & \sigma, H \models_{DR} \mathtt{TR}[\![e_1]\!] = \mathtt{TR}[\![e_2]\!] \\
\iff & \langle \text{by syntactic mapping (Def 4.3)} \rangle \\
 & \sigma, H \models_{DR} \mathtt{TR}[\![e_1 = e_2]\!]
\end{array}
$$

The second base case is when a is of the form $x.f \mapsto e$, and $F = \mathcal{RE}[\![\mathbf{fp}_{Hy}(x.f \mapsto e)]\!]_{\sigma,h}$.

We first prove $\mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![x.f \mapsto e]\!])]\!]_{\sigma,h} = \mathcal{RE}[\![\mathbf{fp}_{Hy}(x.f \mapsto e)]\!]_{\sigma,H}$ as follows:

$$
\begin{array}{ll}
 & \mathcal{RE}[\![\mathbf{fp}_{Hy}(x.f \mapsto e)]\!]_{\sigma,h} \\
= & \quad \langle \text{by hypothetical footprint (Def. 4.4)} \rangle \\
 & \mathcal{RE}[\![\mathbf{region}\{x.f\}]\!]_{\sigma,h} \\
= & \quad \langle \text{by semantics of region expression (Def. 2.6)} \rangle \\
 & \{(\mathcal{E}_{DR}[\![x]\!]_{\sigma,h}, f)\} \\
= & \quad \langle \text{by Lemma 4.2, twice} \rangle \\
 & \{(\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f)\} \\
= & \quad \langle \text{by semantics of region expression (Def. 2.6)} \rangle \\
 & \mathcal{RE}[\![\mathbf{region}\{x.f\}]\!]_{\sigma,H}
\end{array}
$$

$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fpt}(PointsTo_f(x,e))]\!]_{\sigma,H}$
$= \quad \langle$by syntactic mapping (Def. 4.3)$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![x.f \mapsto e]\!])]\!]_{\sigma,H}$

Next we prove $\sigma, h \models_{RSL} x.f \mapsto e \iff \sigma, H \models_{DR} \mathtt{TR}[\![x.f \mapsto e]\!]$ under the assumption that $h \overset{\mathrm{F}}{\equiv} H$.

We first prove it from the left side to the right side. Assume $\sigma, h \models_{RSL} x.f \mapsto e$, where $F = \{(\mathcal{E}_{RSL}[\![x]\!]_\sigma, f)\}$. We calculate it as follows:

$\quad \sigma, h \models_{RSL} x.f \mapsto e$
$\iff \langle$by semantics of RSL (Def. 3.2)$\rangle$
$\quad dom(h) = \{(\mathcal{E}_{RSL}[\![x]\!]_\sigma, f)\}$ *and*
$\quad \mathcal{E}_{RSL}[\![x]\!]_\sigma \neq null$ *and* $h[\mathcal{E}_{RSL}[\![x]\!]_\sigma, f] = \mathcal{E}_{RSL}[\![e]\!]_\sigma$
$\Rightarrow \quad \langle$by assumption: $h \overset{\mathrm{F}}{\equiv} H\rangle$
$\quad H[\mathcal{E}_{RSL}[\![x]\!]_\sigma, f] = \mathcal{E}_{RSL}[\![e]\!]_\sigma$ *and*
$\quad \mathcal{E}_{RSL}[\![x]\!]_\sigma \neq null$
$\iff \langle$by $\mathcal{E}_{RSL}[\![e]\!]_\sigma = \mathcal{E}_{DR}[\![e]\!]_{\sigma,H}$ (Lemma: 4.2), three times$\rangle$
$\quad H[\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f] = \mathcal{E}_{DR}[\![e]\!]_{\sigma,H}$ *and*
$\quad \mathcal{E}_{DR}[\![x]\!]_{\sigma,H} \neq null$
$\iff \langle$by syntactic mapping (Def. 4.1), three times$\rangle$
$\quad H[\mathcal{E}_{DR}[\![\mathtt{TR}[\![x]\!]]\!]_{\sigma,H}, f] = \mathcal{E}_{DR}[\![\mathtt{TR}[\![e]\!]]\!]_{\sigma,H}$ *and*
$\quad \mathcal{E}_{DR}[\![\mathtt{TR}[\![x]\!]]\!]_{\sigma,H} \neq null$
$\iff \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$\quad \sigma, H \models_{DR} \mathtt{TR}[\![x]\!] \neq null \;\&\&$
$\quad \mathtt{TR}[\![x]\!].f = \mathtt{TR}[\![e]\!]$
$\iff \langle$by definition of $PointsTo_f$ predicate$\rangle$
$\quad \sigma, H \models_{DR} PointsTo_f(\mathtt{TR}[\![x]\!], \mathtt{TR}[\![e]\!])$
$\iff \langle$by syntactic mapping (Def. 4.3) $\rangle$
$\quad \sigma, H \models_{DR} \mathtt{TR}[\![x.f \mapsto e]\!]$

Then we prove it from the right side to the left side. Assume $\sigma, H \models_{DR} \mathtt{TR}[\![x.f \mapsto e]\!]$, where its hypothetical footprint is $F = \{(\mathcal{E}_{DR}[\![x]\!]_{\sigma,h}, f)\}$. We calculate it as follows:

$\quad \sigma, H \models_{DR} \mathtt{TR}[\![x.f \mapsto e]\!]$
$\iff \langle$by syntactic mapping (Def. 4.3) $\rangle$
$\quad \sigma, H \models_{DR} PointsTo_f(\mathtt{TR}[\![x]\!], \mathtt{TR}[\![e]\!])$
$\iff \langle$by definition of $PointsTo_f$ predicate$\rangle$
$\quad \sigma, H \models_{DR} \mathtt{TR}[\![x]\!] \neq null \;\&\&\; \mathtt{TR}[\![x]\!].f = \mathtt{TR}[\![e]\!]$
$\iff \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$\quad H[\mathcal{E}_{DR}[\![\mathtt{TR}[\![x]\!]]\!]_{\sigma,H}, f] = \mathcal{E}_{DR}[\![\mathtt{TR}[\![e]\!]]\!]_{\sigma,H}$ *and*
$\quad \mathcal{E}_{DR}[\![\mathtt{TR}[\![x]\!]]\!]_{\sigma,H} \neq null$
$\iff \langle$by syntactic mapping (Def. 4.1), three times$\rangle$
$\quad H[\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f] = \mathcal{E}_{DR}[\![e]\!]_{\sigma,H}$ *and*
$\quad \mathcal{E}_{DR}[\![x]\!]_{\sigma,H} \neq null$
$\Rightarrow \langle$by definition of F, we can construct heap h, such that $dom(h) = F$ and $h \overset{\mathrm{F}}{\equiv} H\rangle$
$\quad dom(h) = \{(\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f)\}$ *and*
$\quad \mathcal{E}_{DR}[\![x]\!]_{\sigma,H} \neq null$ *and*
$\quad h[\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f] = \mathcal{E}_{DR}[\![e]\!]_{\sigma,H}$
$\iff \langle$by $\mathcal{E}_{RSL}[\![e]\!]_\sigma = \mathcal{E}_{DR}[\![e]\!]_{\sigma,H}$ (Lemma: 4.2), four times$\rangle$
$\quad dom(h) = \{(\mathcal{E}_{RSL}[\![x]\!]_\sigma, f)\}$ *and* $\mathcal{E}_{RSL}[\![x]\!]_\sigma \neq null$
$\quad$ *and* $h[\mathcal{E}_{RSL}[\![x]\!]_\sigma, f] = \mathcal{E}_{RSL}[\![e]\!]_\sigma$
$\iff \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$\quad \sigma, h \models_{RSL} x.f \mapsto e$

The inductive hypothesis is that for all subassertions a_i, heaps h_i and H', for each subassertion a_i, the footprint is $F_i = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_i)]\!]_{\sigma,h_i}$. If $h \overset{\mathrm{F_i}}{\equiv} H'$, then $F_i = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_i]\!])]\!]_{\sigma,H'}$, and $\sigma, h_i \models_{RSL} a_i \iff \sigma, H' \models_{DR} \mathtt{TR}[\![a_i]\!]$.

The first inductive case is when a is of the form $a_1 * a_2$. we first prove $\mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1 * a_2]\!])]\!]_{\sigma,H} = \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1 * a_2)]\!]_{\sigma,h}$ as follows:

$\quad \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1 * a_2)]\!]_{\sigma,h}$
$= \quad \langle$by hypothetical footprint (Def. 4.4)$\rangle$
$\quad \mathcal{RE}\left[\!\!\left[\begin{array}{l} \mathbf{fp}_{Hy}(a_1) + \mathbf{if}\ a_1\ \mathbf{then}\ \mathbf{fp}_{Hy}(a_2) \\ \mathbf{else}\ \mathbf{region}\{\} \end{array} \right]\!\!\right]_{\sigma,h}$
$= \quad \langle$by the presumed semantics (formula (1)), twice$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup \mathbf{if}\ \mathcal{A}_{RSL}[\![a_1]\!]_{\sigma,h} = true$
$\quad \mathbf{then}\ \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h} \mathbf{else}\ \varnothing$
$= \quad \langle$by inductive hypothesis, $\sigma, h \models_{RSL} a_i \iff \sigma, H \models_{DR} \mathtt{TR}[\![a_i]\!]$, and Lemma 4.8$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup$
$\quad \mathbf{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = true$
$\quad \mathbf{then}\ \mathcal{RE}[\![\mathbf{fp}_{Hy}(a_2)]\!]_{\sigma,h}\ \mathbf{else}\ \varnothing$
$= \quad \langle$by inductive hypothesis, $\mathcal{RE}[\![\mathbf{fp}_{Hy}(a_i)]\!]_{\sigma,h} = \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_i]\!])]\!]_{\sigma,H}$, twice$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$
$\quad \mathbf{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = true$
$\quad \mathbf{then}\ \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}\ \mathbf{else}\ \varnothing$
$= \quad \langle$by set theory$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$
$\quad \mathbf{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = true$
$\quad \mathbf{then}\ \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H} \cup \varnothing \mathbf{else}\ \varnothing$
$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$
$\quad \mathbf{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a]\!]]\!]_{\sigma,H} = true$
$\quad \mathbf{then} \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H} \cup$
$\quad \mathcal{RE}\left[\!\!\left[\begin{array}{l} (\mathbf{if}\ \mathtt{TR}[\![a_2]\!]\ \mathbf{then}\ \mathbf{region}\{\} \\ \mathbf{else}\ \mathbf{region}\{\}) \end{array} \right]\!\!\right]_{\sigma,H}$
$\quad \mathbf{else}\ \varnothing$
$= \quad \langle$by semantics of Dafny (Def. 2.6)$\rangle$
$\quad \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$
$\quad \mathbf{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a]\!]]\!]_{\sigma,H} = true$
$\quad \mathbf{then} \mathcal{RE}[\![\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H} \cup$
$\quad \mathcal{RE}\left[\!\!\left[\begin{array}{l} (\mathbf{if}\ \mathtt{TR}[\![a_2]\!]\ \mathbf{then}\ \mathbf{region}\{\} \\ \mathbf{else}\ \mathbf{region}\{\}) \end{array} \right]\!\!\right]_{\sigma,H}$
$\quad \mathbf{else}\ \varnothing$
$= \quad \langle$by semantics of Dafny (Def. 2.6)$\rangle$
$\quad \mathcal{RE}\left[\!\!\left[\begin{array}{l} \mathbf{fpt}(\mathtt{TR}[\![a_1]\!]) + \\ \mathbf{if}\ \mathtt{TR}[\![a_1]\!]\ \mathbf{then}\ \mathbf{fpt}(\mathtt{TR}[\![a_2]\!]) + \\ \mathbf{if}\ \mathtt{TR}[\![a_2]\!]\ \mathbf{then} \\ \mathbf{fpt}(\mathbf{fpt}(\mathtt{TR}[\![a_1]\!])!!\mathbf{fpt}(\mathtt{TR}[\![a_2]\!])) \\ \mathbf{else}\ \mathbf{region}\{\}\mathbf{else}\ \mathbf{region}\{\} \end{array} \right]\!\!\right]_{\sigma,H}$
$= \quad \langle$by semantics of Dafny (Def. 2.6)$\rangle$

$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket) + (\mathbf{if}\ \mathtt{TR}\llbracket a_1\rrbracket\\ \mathbf{then}\ \mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket \&\& \\ (\mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)\ !!\ \mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket)))\\ \mathbf{else\ region}\{\})\end{array}\right]\!\!\right]_{\sigma,H}$$

$$= \quad \langle\text{by semantic of Dafny (Def. 2.6)}\rangle$$

$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket\ \&\&\ \mathtt{TR}\llbracket a_2\rrbracket\ \&\&\\ \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)\ !!\ \mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket))\end{array}\right]\!\!\right]_{\sigma,H}$$

$$= \quad \langle\text{by syntactic mapping (Def. 4.3)}\rangle$$

$$\mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_1 * a_2\rrbracket)\rrbracket_{\sigma,H}$$

Next we prove $\sigma, h \models_{RSL} a_1 * a_2 \iff \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1 * a_2\rrbracket$.

We first prove it from the left side to the right side. Assume $\sigma, h \models_{RSL} a_1 * a_2$, by the semantic of separating conjunction, h is divided into two sub-heaps, h_1 and h_2, where $F_1 = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h_1}$ and $F_2 = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_2)\rrbracket_{\sigma,h_2}$. Since $h_1 \subseteq h$ and $h_2 \subseteq h$, by definition of heap, $F_1 = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h}$ and $F_2 = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_2)\rrbracket_{\sigma,h}$.We calculate as follows:

$$\begin{array}{ll}
& \sigma, h \models_{RSL} a_1 * a_2\\
\iff & \left\langle\begin{array}{l}\text{by the semantics of RSL (Def. 3.2), and Corol-}\\ \text{lary 4.7}\end{array}\right\rangle\\
& exists\ h_1, h_2.(h_1 \perp h_2\ and\ h = h_1 \cdot h_2\ and\\
& \mathbf{if}\ \sigma, h_1 \models_{RSL} a_1\ \mathbf{then}\ \sigma, h_2 \models_{RSL} a_2\ \mathbf{else}\ false)\\
& and\ F_1 \cap F_2 = \varnothing\\
\Rightarrow & \left\langle\begin{array}{l}\text{by inductive the hypothesis,}\\ \sigma, h_i \models_{RSL} a_i \iff \sigma, H \models_{DR} \mathtt{TR}\llbracket a_i\rrbracket,\\ \text{twice, and } h_i \overset{\mathrm{F_i}}{\equiv} H\end{array}\right\rangle\\
& \mathbf{if}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \mathbf{then}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_2\rrbracket\\
& \mathbf{else}\ false\ and\ F_1 \cap F_2 = \varnothing\\
\iff & \left\langle\begin{array}{l}\text{by } F_1 = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h} \text{ and}\\ F_2 = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_2)\rrbracket_{\sigma,h}\end{array}\right\rangle\\
& \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,H} \cap \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_2)\rrbracket_{\sigma,H} = \varnothing\\
& and\ \mathbf{if}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \mathbf{then}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_2\rrbracket\\
& \mathbf{else}\ false\\
\iff & \left\langle\begin{array}{l}\text{by inductive hypothesis } \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_i)\rrbracket_{\sigma,h} =\\ \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_i\rrbracket)\rrbracket_{\sigma,H}\text{, twice}\end{array}\right\rangle\\
& \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)\rrbracket_{\sigma,H} \cap\\
& \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket)\rrbracket_{\sigma,H} = \varnothing\ and\\
& \mathbf{if}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \mathbf{then}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_2\rrbracket\\
& \mathbf{else}\ false\\
\iff & \langle\text{by semantics of DafnyR (Def. 2.6), twice}\rangle\\
& \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \&\&\ \mathtt{TR}\llbracket a_2\rrbracket\ \&\&\\
& \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)!!\mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket)\\
\iff & \langle\text{by syntactic mapping (Def. 4.3)}\rangle\\
& \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1 * a_2\rrbracket
\end{array}$$

Then we prove it from the right side to the left side. Assume $\sigma, H \models_{DR} \mathtt{TR}\llbracket a_1 * a_2\rrbracket$, where the footprint of a_1 is $F_1 = \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)\rrbracket_{\sigma,H}$, and the footprint of a_2 is $F_2 = \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)\rrbracket_{\sigma,H}$.We calculate as follows:

$$\begin{array}{ll}
& \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1 * a_2\rrbracket\\
\iff & \langle\text{by syntactic mapping (Def. 4.3)}\rangle\\
& \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \&\&\ \mathtt{TR}\llbracket a_2\rrbracket\ \&\&\\
& \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)!!\mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket)\\
\iff & \langle\text{by semantics of DafnyR (Def. 2.6), twice}\rangle\\
& \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_1\rrbracket)\rrbracket_{\sigma,H} \cap \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_2\rrbracket)\rrbracket_{\sigma,H}\\
& = \varnothing\ and\ \mathbf{if}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \mathbf{then}\\
& \sigma, H \models_{DR} \mathtt{TR}\llbracket a_2\rrbracket\ \mathbf{else}\ false\\
\Rightarrow & \left\langle\begin{array}{l}\text{by definition of } F_1 \text{ and } F_2\text{, we can construct}\\ \text{heaps } h_1 \text{ and } h_2\text{, such that } dom(h_1) = F_1 \text{ and}\\ dom(h_2) = F_2 \text{ and } h_1 \overset{\mathrm{F_1}}{\equiv} H \text{ and } h_2 \overset{\mathrm{F_2}}{\equiv} H \text{ and}\\ \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_i)\rrbracket_{\sigma,h} = \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_i\rrbracket)\rrbracket_{\sigma,H}\text{,}\\ \text{twice}\end{array}\right\rangle\\
& exists\ h_1, h_2.\ dom(h_1) \cap dom(h_2) = \varnothing\ and\\
& \mathbf{if}\ \sigma, h_1 \models_{RSL} a_1\ \mathbf{then}\ \sigma, h_2 \models_{RSL} a_2\\
& \mathbf{else}\ false\ and\ \mathbf{if}\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_1\rrbracket\ \mathbf{then}\\
& \sigma, H \models_{DR} \mathtt{TR}\llbracket a_2\rrbracket\ \mathbf{else}\ false\\
= & \left\langle\begin{array}{l}\text{by inductive hypothesis, } \sigma, h \models_{RSL} a_i \iff\\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_i\rrbracket\text{, twice}\end{array}\right\rangle\\
& exists\ h_1, h_2.\ dom(h_1) \cap dom(h_2) = \varnothing\ and\\
& \mathbf{if}\ \sigma, h_1 \models_{RSL} a_1\ \mathbf{then}\ \sigma, h_2 \models_{RSL} a_2\ \mathbf{else}\\
& false\ and\ \mathbf{if}\ \sigma, h_1' \models_{RSL} a_1\ \mathbf{then}\ \sigma, h_2' \models_{RSL} a_2\\
& \mathbf{else}\ false\\
\Rightarrow & \left\langle\begin{array}{l}\text{by inductive hypothesis } \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_i)\rrbracket_{\sigma,h} =\\ \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_i\rrbracket)\rrbracket_{\sigma,H}\text{, twice. And } h_1 \overset{\mathrm{F_1}}{\equiv} h_1'\\ \text{and } h_2 \overset{\mathrm{F_2}}{\equiv} h_2'\end{array}\right\rangle\\
& exists\ h_1, h_2.\ dom(h_1) \cap dom(h_2) = \varnothing\ and\\
& \mathbf{if}\ \sigma, h_1 \models_{RSL} a_1\ \mathbf{then}\ \sigma, h_2 \models_{RSL} a_2\ \mathbf{else}\ false\\
\Rightarrow & \langle\text{by construction } h = h_1 \cdot h_2\rangle\\
& exists\ h_1, h_2.\ dom(h_1) \cap dom(h_2) = \varnothing\ and\\
& \mathbf{if}\ \sigma, h_1 \models_{RSL} a_1\ \mathbf{else}\ \sigma, h_2 \models_{RSL} a_2\ \mathbf{then}\ false\\
& and\ h = h_1 \cdot h_2\\
\iff & \langle\text{by definition of heap (Def. 2.2)}\rangle\\
& exists\ h_1, h_2.\ h_1 \perp h_2\ and\\
& \mathbf{if}\ \sigma, h_1 \models_{RSL} a_1\ \mathbf{else}\ \sigma, h_2 \models_{RSL} a_2\ \mathbf{then}\ false\\
& and\ h = h_1 \cdot h_2\\
\iff & \langle\text{by semantics of RSL (Def. 3.2)}\rangle\\
& \sigma, h \models a_1 * a_2
\end{array}$$

The second inductive case is when a is of the form $a_1 \wedge a_2$. We first prove $\mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_1 \wedge a_2\rrbracket)\rrbracket_{\sigma,H} = \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1 \wedge a_2)\rrbracket_{\sigma,h}$ as follows:

$$\begin{array}{ll}
& \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1 \wedge a_2)\rrbracket_{\sigma,h}\\
= & \langle\text{by hypothetical footprint of RSL (Def. 4.4)}\rangle\\
& \mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathbf{fp}_{Hy}(a_1)\ +\ \mathbf{if}\ a_1\ \mathbf{then}\\ \mathbf{fp}_{Hy}(a_1)\ \mathbf{else\ region}\{\}\end{array}\right]\!\!\right]_{\sigma,h}\\
= & \langle\text{by the presumed semantics (formula (1)), twice}\rangle\\
& \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h} \cup\\
& \mathbf{if}\ \mathcal{A}_{RSL}\llbracket a_1\rrbracket_{\sigma,h} = true\ \mathbf{then}\\
& \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h}\ \mathbf{else}\ \varnothing\\
= & \left\langle\begin{array}{l}\text{by inductive hypothesis, } \sigma, h \models_{RSL} a_i \iff\\ \sigma, H \models_{DR} \mathtt{TR}\llbracket a_i\rrbracket\text{, and Lemma 4.8}\end{array}\right\rangle\\
& \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h} \cup\\
& \mathbf{if}\ \mathcal{A}_{DR}\llbracket \mathtt{TR}\llbracket a_1\rrbracket\rrbracket_{\sigma,H} = true\ \mathbf{then}\\
& \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_1)\rrbracket_{\sigma,h}\ \mathbf{else}\ \varnothing\\
= & \left\langle\begin{array}{l}\text{by inductive hypothesis,}\\ \mathcal{RE}\llbracket \mathbf{fp}_{Hy}(a_i)\rrbracket_{\sigma,h} = \mathcal{RE}\llbracket \mathbf{fpt}(\mathtt{TR}\llbracket a_i\rrbracket)\rrbracket_{\sigma,H}\text{,}\\ \text{twice}\end{array}\right\rangle
\end{array}$$

$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$$
$$\mathtt{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = \mathit{true}\ \mathtt{then}$$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}\ \mathtt{else}\ \emptyset$$
$= \quad \langle$by semantics of DafnyR (Def. 2.6), twice$\rangle$
$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathtt{fpt}(\mathtt{TR}[\![a_1]\!]) + (\mathtt{if}\ \mathtt{TR}[\![a_1]\!]\ \mathtt{then}\\ \mathtt{fpt}(\mathtt{TR}[\![a_2]\!])\ \mathtt{else}\ \mathtt{region}\{\})\end{array}\right]\!\!\right]_{\sigma,H}$$
$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1]\!]\&\&\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}$$
$= \quad \langle$by syntactic mapping (Def. 4.3)$\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1 \wedge a_2]\!])]\!]_{\sigma,H}$$

Next we prove $\sigma, h \models_{RSL} a_1 \wedge a_2 \iff \sigma, H \models_{DR} \mathtt{TR}[\![a_1 \wedge a_2]\!]$ as follows:

$$\sigma, h \models_{RSL} a_1 \wedge a_2$$
$\iff \langle$by semantics of RSL (Def. 3.2)$\rangle$
if $\sigma, h \models_{RSL} a_1$ **then** $\sigma, h \models_{RSL} a_2$ **else** false
$\iff \langle$by inductive hypothesis$\rangle$
if $\sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!]$ **then** $\sigma, H \models_{DR} \mathtt{TR}[\![a_2]\!]$
else false
$\iff \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$$\sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!]\,\&\,\&\,\mathtt{TR}[\![a_2]\!]$$

The third inductive case is when a is of the form $a_1 \vee a_2$. We first prove $\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1 \vee a_2]\!])]\!]_{\sigma,H} = \mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1 \vee a_2)]\!]_{\sigma,h}$ as follows:

$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1 \vee a_2)]\!]_{\sigma,h}$$
$= \quad \langle$by hypothetical footprint of RSL (Def. 4.4)$\rangle$
$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathtt{fp}_{Hy}(a_1) + (\mathtt{if}\ a_1\ \mathtt{then}\\ \mathtt{region}\{\}\ \mathtt{else}\ \mathtt{fp}_{Hy}(a_2))\end{array}\right]\!\!\right]_{\sigma,h}$$
$= \quad \langle$by semantics of region expressions (Def: 2.6)$\rangle$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup$$
$$\mathtt{if}\ \mathcal{A}_{RSL}[\![a_1]\!]_{\sigma,h} = \mathit{true}\ \mathtt{then}\ \emptyset$$
$$\mathtt{else}\ \mathcal{RE}[\![\mathtt{fp}_{Hy}(a_2)]\!]_{\sigma,h}$$
$= \quad \left\langle \begin{array}{l}\text{by inductive hypothesis, } \sigma, h \models_{RSL} a_i \iff \\ \sigma, H \models_{DR} \mathtt{TR}[\![a_i]\!]\text{, and Lemma 4.8}\end{array}\right\rangle$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup$$
$$\mathtt{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = \mathit{true}\ \mathtt{then}\ \emptyset$$
$$\mathtt{else}\ \mathcal{RE}[\![\mathtt{fp}_{Hy}(a_2)]\!]_{\sigma,h}$$
$= \quad \left\langle \begin{array}{l}\text{by inductive hypothesis,}\\ \mathcal{RE}[\![\mathtt{fp}_{Hy}(a_i)]\!]_{\sigma,h} = \mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_i]\!])]\!]_{\sigma,H},\\ \text{twice}\end{array}\right\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$$
$$\mathtt{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = \mathit{true}\ \mathtt{then}\ \emptyset$$
$$\mathtt{else}\ \mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}$$
$= \quad \langle$by semantics of DafnyR (Def. 2.6), twice$\rangle$
$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathtt{fpt}(\mathtt{TR}[\![a_1]\!]) + (\mathtt{if}\ \mathtt{TR}[\![a_1]\!]\ \mathtt{then}\\ \mathtt{region}\{\}\ \mathtt{else}\ \mathtt{fpt}(\mathtt{TR}[\![a_2]\!]))\end{array}\right]\!\!\right]_{\sigma,H}$$
$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(\mathtt{TR}[\![a_1]\!]\,||\,\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}$$
$= \quad \langle$by syntactic mapping (Def. 4.3)$\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1 \vee a_2]\!])]\!]_{\sigma,H}$$

Next we prove $\sigma, h \models_{RSL} a_1 \vee a_2 \iff \sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!]\,||\,\mathtt{TR}[\![a_2]\!]$ as follows:

$$\sigma, h \models_{RSL} a_1 \vee a_2$$
$\iff \langle$by semantics of RSL (Def. 3.2)$\rangle$
if $\sigma, h \models_{RSL} a_1$ **then** true **else** $\sigma, h \models_{RSL} a_2$
$\iff \langle$by inductive hypothesis$\rangle$
if $\sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!]$ **then** true **else**
$\sigma, H \models_{DR} \mathtt{TR}[\![a_2]\!]$
$\iff \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$$\sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!]\,||\,\mathtt{TR}[\![a_2]\!]$$

The fourth inductive case is when a is of the form $a_1 \Rightarrow a_2$. We first prove $\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1 \Rightarrow a_2]\!])]\!]_{\sigma,H} = \mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1 \Rightarrow a_2)]\!]_{\sigma,H}$ as follows:

$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1 \Rightarrow a_2)]\!]_{\sigma,h}$$
$= \quad \langle$by hypothetical footprint of RSL (Def. 4.4)$\rangle$
$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathtt{fp}_{Hy}(a_1) + (\mathtt{if}\ a_1\ \mathtt{then}\\ \mathtt{fp}_{Hy}(a_2)\ \mathtt{else}\ \mathtt{region}\{\})\end{array}\right]\!\!\right]_{\sigma,h}$$
$= \quad \langle$by semantics of region expression (Def. 2.6)$\rangle$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup$$
$$\mathtt{if}\ \mathcal{A}_{RSL}[\![a_1]\!]_{\sigma,h} = \mathit{true}\ \mathtt{then}$$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_2)]\!]_{\sigma,h}\ \mathtt{else}\ \emptyset$$
$= \quad \left\langle \begin{array}{l}\text{by inductive hypothesis, } \sigma, h \models_{RSL} a_i \iff \\ \sigma, H \models_{DR} \mathtt{TR}[\![a_i]\!]\text{, and Lemma 4.8}\end{array}\right\rangle$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_1)]\!]_{\sigma,h} \cup$$
$$\mathtt{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = \mathit{true}\ \mathtt{then}$$
$$\mathcal{RE}[\![\mathtt{fp}_{Hy}(a_2)]\!]_{\sigma,h}\ \mathtt{else}\ \emptyset$$
$= \quad \left\langle \begin{array}{l}\text{by inductive hypothesis,}\\ \mathcal{RE}[\![\mathtt{fp}_{Hy}(a_i)]\!]_{\sigma,h} = \mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_i]\!])]\!]_{\sigma,H},\\ \text{twice}\end{array}\right\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1]\!])]\!]_{\sigma,H} \cup$$
$$\mathtt{if}\ \mathcal{A}_{DR}[\![\mathtt{TR}[\![a_1]\!]]\!]_{\sigma,H} = \mathit{true}\ \mathtt{then}$$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}\ \mathtt{else}\ \emptyset$$
$= \quad \langle$by semantics of DafnyR (Def. 2.6), twice$\rangle$
$$\mathcal{RE}\left[\!\!\left[\begin{array}{l}\mathtt{fpt}(\mathtt{TR}[\![a_1]\!]) + (\mathtt{if}\ \mathtt{TR}[\![a_1]\!]\ \mathtt{then}\\ \mathtt{fpt}(\mathtt{TR}[\![a_2]\!])\ \mathtt{else}\ \mathtt{region}\{\})\end{array}\right]\!\!\right]_{\sigma,H}$$
$= \quad \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1]\!] \Rightarrow \mathtt{TR}[\![a_2]\!])]\!]_{\sigma,H}$$
$= \quad \langle$by syntactic mapping (Def. 4.3)$\rangle$
$$\mathcal{RE}[\![\mathtt{fpt}(\mathtt{TR}[\![a_1 \Rightarrow a_2]\!])]\!]_{\sigma,H}$$

Next we prove $\sigma, h \models_{RSL} a_1 \Rightarrow a_2 \iff \sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!] \Rightarrow \mathtt{TR}[\![a_2]\!]$ as follows:

$$\sigma, h \models_{RSL} a_1 \Rightarrow a_2$$
$\iff \langle$by semantics of RSL (Def. 3.2)$\rangle$
if $\sigma, h \models_{RSL} a_1$ **then** $\sigma, h \models_{RSL} a_2$ **else** true
$\iff \langle$by inductive hypothesis$\rangle$
if $\sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!]$ **then** $\sigma, H \models_{DR} \mathtt{TR}[\![a_2]\!]$
else true
$\iff \langle$by semantics of DafnyR (Def. 2.6)$\rangle$
$$\sigma, H \models_{DR} \mathtt{TR}[\![a_1]\!] \Rightarrow \mathtt{TR}[\![a_2]\!]$$

The fifth inductive case is when a is of the form $\exists x'.x.f \mapsto x' * a$. By the definition of the hypothetical footprint, the footprint of existential assertions do not depend on the existential variables. Therefore it is just a form of separating conjunction case. ∎.

B. Soundness of DafnyR's logic

THEOREM B.1 (Soundness of inference rules). *Let P, Q and I be type correct assertions, and let Stmt be a type correct DafnyR statement. Let ε be a type correct region expression. The axioms and rules for DafnyR are valid. That is:*

$$if \vdash_{DR} \{P\}\, Stmt\, \{Q\}\, [\varepsilon],\ then \vDash_{DR} \{P\}\, Stmt\, \{Q\}\, [\varepsilon].$$

Proof: We prove this by induction on the structure of the proof of $\{P\}$ *Stmt* $\{Q\}$ $[\varepsilon]$. Let (σ, H) be an arbitrary state, and without loss of generality, let $(\sigma', H') = \mathcal{S}[\![Stmt]\!]_{\sigma,H}$. We assume $\vdash_{DR} \{P\}\, Stmt\, \{Q\}\, [\varepsilon]$, and $\sigma, H \vDash_{DR} P$. Then we must prove $\sigma', H' \vDash_{DR} Q$, and that all the changed locations are in ε.

1. ($ALLOC_{DR}$) In this case, *Stmt* is x := **new** K, P is $true$, Q is $\{\&\&_{i=1}^{n} PointsTo_{f_i}(x, 0)\}$ and $\varepsilon =$ **region**$\{\}$. We derive Q as below:

   By the semantics, $(\sigma', H') = (\textbf{let}\ (\sigma'', H'') = alloc(H)$ $in\ (\sigma'', H''[\overline{(\sigma''[x], f_i) \mapsto 0}]))$, which entails Q.

   For the frame condition, *Stmt* only updates newly allocated locations, therefore $\varepsilon =$ **region**$\{\}$ is a correct frame.

2. ($ASGN_{DR}$) In this case, *Stmt* is x := *Expr*, P is $true$, Q is $\{x = Expr\}$ and $\varepsilon =$ **region**$\{\}$, where$x \notin \mathtt{FV}(Expr)$. We derive Q as below:

   By the semantics, $(\sigma', H') = (\sigma[x \mapsto \mathcal{E}_{DR}[\![Expr]\!]_{\sigma,H}], H)$, which entails Q.

   For the frame condition, this statement only updates variable x in the store. So nothing is changed in the heap. Therefore $\varepsilon =$ **region**$\{\}$ is a correct frame.

3. (UPD_{DR}) In this case, *Stmt* is $x.f$:= *Expr*, P is $x \neq null$, Q is $x.f = Expr$ and $\varepsilon =$ **region**$\{x.f\}$. We derive Q as below:

   By the semantics, $(\sigma', H') = (\sigma, H[(\mathcal{E}_{DR}[\![x]\!]_{\sigma,H}, f) \mapsto \mathcal{E}_{DR}[\![Expr]\!]_{\sigma,H}])$, which entails Q.

   For the frame condition, this statement changes the singleton heap location (x, f). Therefore $\varepsilon =$ **region**$\{x.f\}$ is a correct frame.

4. (ACC_{DR}) In this case, *Stmt* is x := $x'.f$, P is $x' \neq null$ && $x'.f = Expr$, Q is $x = Expr$, and $\varepsilon =$ **region**$\{\}$. We derive Q as below:

   By the semantics,

   $(\sigma', H') = (\sigma[x \mapsto H[(\mathcal{E}_{DR}[\![x']\!]_{\sigma,H}, f)]], H)$, which entails Q.

   For the frame condition, this statement only updates variable x in the store. So nothing is changed in the heap. Therefore $\varepsilon =$ **region**$\{\}$ is a correct frame.

5. (SEQ_{DR}) In this case, *Stmt* is *Stmt*;*Stmt*. By the inductive hypothesis for $Stmt_1$ and $Stmt_2$, $(\sigma'', H'') = \mathcal{S}[\![Stmt_1]\!]_{\sigma,H}$, and $\sigma'', H'' \vDash_{DR} Q''$.

   By the second premise and the semantics, $(\sigma', H') = \mathcal{S}[\![Stmt_2]\!]_{\sigma'',H''}$. Hence $\sigma', H' \vDash_{DR} Q$.

   For the frame condition, by the two premises, let ε_1 and ε_2 be the frame conditions of $Stmt_1$ and $Stmt_2$. Then the frame condition of the sequential statements is $\varepsilon = \varepsilon_1 + \varepsilon_2$.

6. (IF_{DR}) In this case, *Stmt* is **if**$(Expr \neq 0)\{Stmt_1\}$**else** $\{Stmt_2\}$.

   There are two cases:

   Case1: $Expr \neq 0$. By the inductive hypothesis, $(\sigma', H') = \mathcal{S}[\![Stmt_1]\!]_{\sigma,H}$, which entails Q.

   Case2: $Expr = 0$. By the inductive hypothesis, $(\sigma', H') = \mathcal{S}[\![Stmt_2]\!]_{\sigma,H}$, which entails Q.

   For the frame condition, by the induction hypothesis, ε is a correct frame.

7. ($WHILE_{DR}$) In this case, *Stmt* is **while**$(Expr \neq 0)\{Stmt\}$. $P = I$, $Q = I$ && $Expr \neq 0$ and the frame conditions is ε. The premise is $\vDash_{DR}$ $\{I$ && $Expr \neq 0\}$ *Stmt* $\{I\}[\varepsilon]$.

   By the semantics of this statement, let g be a recursive point function, such that

   $g = \underline{\lambda} s$. **if** $\mathcal{E}_{RSL}[\![Expr \neq 0]\!]_{\sigma}$ **then let** $s' = \mathcal{S}[\![Stmt]\!]_{\sigma,H}$ **in** gs' **else** s.

   By definition, fix is a fixed point function, so $fix(g) = g$. Then we prove

   $fix(g)(\sigma, H) \vDash_{DR} I$ by fixed-point induction.

   Base Case: $\perp\ \vDash_{DR} I$ holds vacuously. It requires to prove all members in $\perp$ implies I, but there is nothing in $\perp$. Hence it is vacuously true.

   Inductive Case: Let $\sigma'', H'' \vDash_{DR} I$ hold for an arbitrary iteration of g, and ε is the frame condition. Then we prove that $fix(g)(\sigma'', H'') \vDash_{DR} I$ holds, and the changed locations on the heap is ε.

   There are two cases:

   Case 1: $Expr \neq 0$. By the semantics, $fix(g)(\sigma'', H'') = g(\mathcal{S}[\![Stmt]\!]_{\sigma'',H''})$. By the inductive hypothesis,

   $g(\mathcal{S}[\![Stmt]\!]_{\sigma'',H''}) \vDash_{DR} I$ holds. Hence

   $fix(g)(\sigma'', H'') \vDash_{DR} I$ holds. For the frame condition, since the fixed point function always returns the same function g, which is framed by ε by the induction hypothesis, therefore ε is the frame condition for an arbitrary iteration.

   Case 2: $Expr = 0$. By the semantics, $fix(g)(\sigma'', H'') = (\sigma'', H'')$. Therefore by the inductive hypothesis,

   $fix(g)(\sigma'', H'') \vDash_{DR} I$ holds. For the frame condition, since the state does not change, the frame is **region**$\{\}$, which is the subset of ε.

Now we conclude that if the loop exits, which means that $Expr = 0$ holds, the loop invariant I holds. Therefore, Q holds and ε is its frame condition.

8. ($SubEff_{DR}$) In this case, by the inductive hypothesis, $\models_{DR}$ $\{P\}Stmt\{Q\}[\varepsilon]$. Hence when applying the frame condition $\varepsilon' \geqslant \varepsilon$, the locations that may be changed are also contained in ε'. Therefore ε' is a correct frame.

9. (CON_{DR}) In this case, by the inductive hypothesis, $\{P'\}Stmt\{Q'\}[\varepsilon] \models_{DR}$. By the premise, $P \Rightarrow P'$ and $Q' \Rightarrow Q$. Hence $\models_{DR} \{P\}Stmt\{Q\}[\varepsilon]$ is valid.

10. (FRM_{DR}) In this case, the premise is

    $\{P\}Stmt\{Q\}[\varepsilon] \models_{DR}$.

    By the inductive hypothesis of DafyR, the side condition $\varepsilon !! \mathbf{fpt}(R)$ means R's footprint is disjoint with the locations where side effects take place. That means the values in $\mathbf{fpt}(R)$, which are outside ε, are not changed. Thus, by definition of semantic footprint 2.5, the validity of R is preserved after executing *Stmt*.

    For the frame condition, since R is unchanged, locations that may be changed must be in ε. ∎

LEMMA B.2 (Soundness of sub-frame rules). *Let ε and η be frames. if $\vdash \varepsilon <= \eta$, then $\sigma, H \models \varepsilon <= \eta$ for all heaps H and stores σ.*

Proof: By induction on the derivation of $\vdash \varepsilon <= \eta$. The semantics of $<=$ and $*$ maps to the operations $\subseteq$ and $\cap$ on sets, which have the required properties. ∎

Acknowledgments

The work of both authors is supported in part by US NSF under grant CCF-0916715. Thanks to David Naumann for discussions about region logic and comments on an earlier work. Thanks to Rustan Leino for discussions about Dafny and Boogie and help with their implementation.

References

[1] A. Banerjee, D. A. Naumann, and S. Rosenberg. Regional logic for local reasoning about global invariants. In J. Vitek, editor, *European Conference on Object-Oriented Programming (ECOOP)*, volume 5142 of *Lecture Notes in Computer Science*, pages 387–411, New York, NY, 2008. Springer-Verlag.

[2] Y. Bao, G. T. Leavens, and G. Ernst. Translating separation logic into dynamic frames using fine-grained region logic. Technical Report CS-TR-13-02a, Computer Science, University of Central Florida, Orlando, Florida, Mar. 2014.

[3] A. Borgida, J. Mylopoulos, and R. Reiter. On the frame problem in procedure specifications. *IEEE Transactions on Software Engineering*, 21(10):785–798, Oct. 1995.

[4] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis (SAS)*, volume 2694 of *Lecture Notes in Computer Science*, pages 55–72, Berlin, 2003. Springer-Verlag.

[5] P. Chalin, J. R. Kiniry, G. T. Leavens, and E. Poll. Beyond assertions: Advanced specification and verification with JML and ESC/Java2. In *Formal Methods for Components and Objects (FMCO) 2005, Revised Lectures*, volume 4111 of *Lecture Notes in Computer Science*, pages 342–363, Berlin, 2006. Springer-Verlag.

[6] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10):576–580,583, Oct. 1969.

[7] A. Hobor and J. Villard. The ramifications of sharing in data structures. In *Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '13, pages 523–536, New York, NY, USA, 2013. ACM.

[8] S. S. Ishtiaq and P. W. O'Hearn. BI as an assertion language for mutable data structures. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '01, pages 14–26, New York, NY, USA, 2001. ACM.

[9] I. T. Kassios. Dynamic frames: Support for framing, dependencies and sharing without restrictions. In E. S. J. Misra, T. Nipkow, editor, *Formal Methods (FM)*, volume 4085 of *Lecture Notes in Computer Science*, pages 268–283, Berlin, 2006. Springer-Verlag.

[10] I. T. Kassios. The dynamic frames theory. *Formal Aspects of Computing*, 23(3):267–288, May 2011.

[11] G. T. Leavens. JML's rich, inherited specifications for behavioral subtypes. In Z. Liu and H. Jifeng, editors, *Formal Methods and Software Engineering: 8th International Conference on Formal Engineering Methods (ICFEM)*, volume 4260 of *Lecture Notes in Computer Science*, pages 2–34, New York, NY, Nov. 2006. Springer-Verlag.

[12] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. Web page at https://dafny.codeplex.com/.

[13] K. R. M. Leino. Specification and verification of object-oriented software. Lecture notes from Marktoberdorf Internation Summer School, available at `http://research.microsoft.com/en-us/um/people/leino/papers/krml190.pdf`, 2008.

[14] K. R. M. Leino. Dafny: An automatic program verifier for functional correctness. In *Logic for Programming, Artificial Intelligence, and Reasoning, 16th International Conference, LPAR-16*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer-Verlag, 2010.

[15] K. R. M. Leino and R. Monahan. Dafny meets the verification benchmarks challenge. In *Proceedings of the Third international conference on Verified software: theories, tools, experiments*, volume 6217 of *Lecture Notes in Computer Science*, pages 112–126, Berlin, 2010. Springer-Verlag.

[16] K. R. M. Leino and P. Müller. A basis for verifying multithreaded programs. In G. Castagna, editor, *Programming Languages and Systems, 18th European Symposium on Programming, ESOP 2009*, volume 5502 of *Lecture Notes in Computer Science*, pages 378–393, Berlin, Mar. 2009. Springer-Verlag.

[17] M. J. Parkinson and A. J. Summers. The relationship between separation logic and implicit dynamic frames. In *Proceedings of the 20th European conference on Programming languages*

and systems: part of the joint European conferences on theory and practice of software, ESOP'11/ETAPS'11, pages 439–458, Berlin, Heidelberg, 2011. Springer-Verlag.

[18] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the Seventeenth Annual IEEE Symposium on Logic in Computer Science*, pages 55–74, Los Alamitos, California, 2002. IEEE Computer Society Press.

[19] S. Rosenberg. Verifier for region logic. Web page at http://www.cs.stevens.edu/ naumann/pub/VERL/., 2011.

[20] J. Smans, B. Jacobs, F. Piessens, and W. Schulte. An automatic verifier for Java-like programs based on dynamic frames. In *Fundamental Approaches to Software Engineering*, volume 4961 of *Lecture Notes in Computer Science*, pages 261–275, Berlin, Apr. 2008. Springer-Verlag.

[21] H. Yang. *Local reasoning for stateful programs*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2001. AAI3023240.