Footprints in Local Reasoning

Mohammad Raza and Philippa Gardner

Department of Computing,
Imperial College London, 180 Queen's Gate, London SW7 2AZ, UK
{mraza,pg}@doc.ic.ac.uk

Abstract. Local reasoning about programs exploits the natural local behaviour common in programs by focussing on the footprint - that part of the resource accessed by the program. We address the problem of formally characterising and analysing the footprint notion for abstract local functions introduced by Calcagno, O'Hearn and Yang. With our definition, we prove that the footprints are the only essential elements required for a complete specification of a local function. We also show that, for well-founded models (which is usually the case in practice), a smallest specification always exists that only includes the footprints, thus formalising the notion of small axioms in local reasoning. We also present results for the non-well-founded case, and introduce the natural class of one-step local functions for which the footprints are the smallest safe states.

Keywords: Footprints, Hoare Logic, Local Reasoning, Separation Logic.

1 Introduction

Local reasoning about programs focusses on the collection of resources directly acted upon by the program. It has recently been introduced and used to substantial effect in *local* Hoare reasoning about memory update. Researchers previously used Hoare reasoning based on First-order Logic to specify how programs interacted with the *whole* memory. O'Hearn, Reynolds and Yang instead introduced local Hoare reasoning based on Separation Logic [13,10]. The idea is to reason only about the local parts of the memory—the *footprints*—that are directly accessed by a program. Intuitively, the footprints form the pre-conditions of the *small* axioms, which provide the smallest complete specification of the program. All the true Hoare triples are derivable from the small axioms and the general Hoare rules. In particular, the *frame rule* extends the reasoning to properties about the rest of the heap which has not been changed by the command.

O'Hearn, Reynolds and Yang originally introduced Separation Logic to solve the problem of how to reason about the mutation of data structures in memory. They have applied their reasoning to several memory models, including heaps based on pointer arithmetic [13], heaps with permissions [4], and the combination of heaps with variable stacks which views variables as resource [5,15]. In each case, the basic soundness and completeness results for local Hoare reasoning are essentially the same. For this reason, Calcagno, O'Hearn and Yang [9] recently introduced abstract local functions over abstract resource models (separation algebras), generalising their specific examples of local imperative commands for manipulating memory models. They develop Abstract

R. Amadio (Ed.): FOSSACS 2008, LNCS 4962, pp. 201–215, 2008.

Separation Logic to provide local Hoare reasoning about such local functions, and give general soundness and completeness results.

We believe that the general concept of a local function is a fundamental step towards establishing the theoretical foundations of local reasoning, and Abstract Separation Logic is an important generalisation of the local Hoare reasoning systems now widely studied in the literature. However, Calcagno, O'Hearn and Yang do not characterise the footprints and small axioms in this general theory, which is a significant omission. O'Hearn, Reynolds and Yang, in one of their first papers on the subject [13], state the local reasoning viewpoint as:

> 'to understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged.'

A complete understanding of the foundations of local Hoare reasoning therefore requires a formal characterisation of the footprint notion. O'Hearn tried to formalise footprints in his work on Separation Logic (personal communication with O'Hearn). His intuition was that the footprints should be the smallest states on which the program is safe, and the *small axioms* arising from these footprints should give rise to a complete specification using the general rules for local Hoare reasoning. However, Yang discovered this notion of footprint does not work, since it does not always yield complete specifications. In this paper, we resolve this problem, providing a definition of footprint which does give rise to complete specifications.

Consider the local program[1]

$$AD ::= \quad x := new(); dispose(x)$$

This *allocate-deallocate* program allocates a new cell, stores its address value in the stack variable x, and then deallocates the cell. It is local because all its atomic constituents are local. This tiny example captures the essence of a common type of program; there are many programs which, for example, create a list, work on the list, and then destroy the list.

The smallest heap on which the *AD* program is safe is the empty heap emp. The specification using this pre-condition is:

$$\{emp\} \quad AD \quad \{emp\} \tag{1}$$

We can extend our reasoning to larger heaps by applying the frame rule: for example, extending to a one-cell heap with arbitrary address l and value v gives

$$\{l \mapsto v\} \quad AD \quad \{l \mapsto v\} \tag{2}$$

However, axiom (1) does not give the complete specification of the *AD* program. In fact, it captures very little of the spirit of allocation followed by de-allocation. For example, the following triple is also true:

$$\{l \mapsto v\} \quad AD \quad \{l \rightarrow v \wedge x \neq l\} \tag{3}$$

[1] Yang's example was the 'allocate-deallocate-test' program *ADT* ::= 'x := new();dispose(x); if (x=1) then z:=0 else z:=1;x=0'. Our *AD* program provides a more standard example of program behaviour.

This triple (3) is true because, if l is already allocated, then the new address cannot be l and hence x cannot be l. It cannot be derived from (1). However, the combination of axiom (1) and axiom (3) for arbitrary one-cell heaps does provide the smallest complete specification. This example illustrates that O'Hearn's intuitive view of the footprints as the minimal safe states just does not work for common imperative programs.

In this paper, we introduce the definition of the footprint of a local function. For our *AD* example, our definition identifies emp and the arbitrary one-cell heaps $l \mapsto v$ as footprints, as expected. We prove the general result that, for any local function, the footprints are the only elements which are *essential* to specify completely the behaviour of this function. For well-founded resource, which is almost always the case in practice, we show that the footprints are also always *sufficient*: that is, a complete specification always exists that only uses the footprints. We also explore results for non-well-founded resource: for models without negativity (no inverse elements except the identity), such as heaps with permissions, either the footprints are sufficient (such as for the *write* command in the permissions model) or there is no smallest complete specification (such as for the *read* command in the permissions model); for models with negativity, we show that there can exist smallest complete specifications based on elements which are not essential and hence not footprints. Finally, we identify a natural class of local functions, which we call *one-step* local functions, which satisfy O'Hearn's original intuition regarding footprints. For well-founded resource, the one-step local functions are precisely the local functions whose footprints are the smallest safe states.

2 Background

This is a background section that describes the separation algebras, local functions and Hoare reasoning introduced in [9]. Further details and motivation can be found in [9].

Definition 1 (Separation Algebra). *A* **separation algebra** *is a cancellative, partial commutative monoid* $(\Sigma, \bullet, u)$. Σ *is a set, and* $\bullet$ *is a partial binary operator with unit* u *which satisfies the familiar axioms of associativity, commutativity and unit, using a partial equality on* Σ *where either both sides are defined and equal, or both are undefined. It also satisfies the cancellative property stating that, for each* $\sigma \in \Sigma$*, the partial function* $\sigma \bullet (\cdot) : \Sigma \mapsto \Sigma$ *is injective.* **Separateness** *(#) and* **substate** *(*$\preceq$*) relations are given by* $\sigma_0 \# \sigma_1$ *iff* $\sigma_0 \bullet \sigma_1$ *is defined and* $\sigma_0 \preceq \sigma_2$ *iff* $\exists \sigma_1 . \, \sigma_2 = \sigma_0 \bullet \sigma_1$.

Examples of separation algebras include multisets under union (with unit $\emptyset$), the natural numbers with addition (with unit 0), heaps as finite partial functions from locations to values ([9] and example 1), heaps with permissions [9,4], and the combination of heaps and variable stacks enabling us to model programs with variables as local functions ([9], [15] and example 1). These example all have an intuition of resource, with $\sigma_1 \bullet \sigma_2$ intuitively giving more resource than just σ_1 and σ_2 for $\sigma_1, \sigma_2 \neq u$. However, there are also examples that do not conform to this resource intuition, such as $\{a, u\}$ with $a \bullet a = u$. We shall overload notation, using Σ to denote $(\Sigma, \bullet, u)$.

Lemma 1 (Subtraction). *For* $\sigma_1, \sigma_2 \in \Sigma$*, if* $\sigma_1 \preceq \sigma_2$ *then there exists a unique element* $\sigma_2 - \sigma_1 \in \Sigma$ *such that* $(\sigma_2 - \sigma_1) \bullet \sigma_1 = \sigma_2$.

Following [9], we model commands on separation algebras as functions of the form $f : \Sigma \to P(\Sigma)^\top$, where $\top$ is an extra top element added to the powerset. The range $P(\Sigma)^\top$ is used to model non-determinism and faulting: elements can map either to a set of elements (to allow for non-determinism) or $\top$ (which represents faulting and returning an error). Mapping to the empty set represents divergence (non-termination).

Definition 2. *Define the set $P(\Sigma)^\top = P(\Sigma) \cup \{\top\}$ with the standard subset relation extended with a new greatest element $\top$: that is, $p \sqsubseteq \top$ for all $p \in P(\Sigma)^\top$. It is a total commutative monoid with $\{u\}$ as the unit and a binary operator $*$ given by:*

$$\begin{aligned} p * q &= \{\sigma_0 \bullet \sigma_1 \mid \sigma_0 \# \sigma_1 \wedge \sigma_0 \in p \wedge \sigma_1 \in q\} \quad \textit{if } p, q \in P(\Sigma) \\ &= \top \quad \textit{otherwise} \end{aligned}$$

For functions $f : \Sigma \to P(\Sigma)^\top$, $f \sqsubseteq g$ iff $f(\sigma) \sqsubseteq g(\sigma)$ for all $\sigma \in \Sigma$.

Intuitively, we think of a command acting on resource to be *local* if whenever the command executes safely on any resource element, then any more resource that may be added will not be affected by the command. This intuition was first formalised in [19] (for the RAM model) with the following constraints:

- *Safety monotonicity*: if the command is safe on some resource element, then it does not fault when more resource is added.
- *Frame property*: if the command is safe on some resource element, then any additional resource will remain unchanged after execution if the execution terminates.

In the abstract setting of [9] which we use in this paper, these two restrictions were amalgamated into the following succinct definition of a local function.

Definition 3 (Local Function). *A* **local function on** *Σ is a total function $f : \Sigma \to P(\Sigma)^\top$ which satisfies the* **locality condition***:*

$$\sigma \# \sigma' \ \ \textit{implies} \ \ f(\sigma' \bullet \sigma) \sqsubseteq \{\sigma'\} * f(\sigma)$$

f faults on σ when it is not sufficient resource for safe execution ($f(\sigma) = \top$). Adding more resource may make the execution safe ($f(\sigma' \bullet \sigma) \sqsubseteq f(\sigma) = \top$). Safety monotonicity follows since if f is safe on σ ($f(\sigma) \sqsubset \top$) then $f(\sigma' \bullet \sigma)$ is also safe ($f(\sigma' \bullet \sigma) \sqsubseteq \{\sigma'\} * f(\sigma) \sqsubset \top$). The frame property follows by the fact that σ' is preserved by f.

Example 1 (Local functions on separation algebras).

1. **Heap dispose command.** We model heaps by the separation algebra $(H, \bullet, u_H)$, where $H = L \rightharpoonup_{\mathit{fin}} Val$ are finite partial functions from a set of locations to a set of values, the partial operator $\bullet$ is the union of partial functions with disjoint domains, and the unit u_H is the empty function. For $h \in H$, let $dom(h)$ be the domain of h. We write $l \mapsto v$ for the partial function with domain $\{l\}$ that maps l to v. For $h_1, h_2 \in H$, if $h_2 \preceq h_1$ then $h_1 - h_2 = h_1 \mid_{dom(h_1) - dom(h_2)}$, and is undefined otherwise. For $h \in H$, the $dispose[l]$ command that deletes the cell at location l in h is given by

$$dispose[l](h) = \begin{cases} \{h - (l \mapsto v)\} & h \succeq (l \mapsto v) \text{ for some } v \\ \top & \text{otherwise} \end{cases}$$

The function is local: if $h \not\preceq (l \mapsto v)$ then $dispose[l](h) = \top$, and $dispose[l](h' \bullet h) \sqsubseteq \top$. Otherwise, $dispose[l](h' \bullet h) = \{(h' \bullet h) - (l \mapsto v)\} \sqsubseteq \{h'\} * \{h - (l \mapsto v)\} = \{h'\} * dispose[l](h)$.

2. ***AD* command.** The *AD* command $x := new(); dispose(x)$ described in the introduction can be modelled as a local function on a separation algebra that includes the heap and the variable stack. We use the algebra $H \times S$ with H as before and $S = Var \rightharpoonup_{fin} Val$. The $\bullet$ operator in this case combines states with disjoint heap and stack domains, and is undefined otherwise. The unit is (u_H, u_S), where u_S is the empty stack. Although this approach is limited to disjoint reference to stack variables, this constraint can be lifted by enriching the separation algebra with *permissions* [4]. However, this added complexity can be avoided for the discussion in this paper. For a state $h \in H \times S$, let $loc(h)$ denote the set of heap locations in h.

$$AD(h) = \begin{cases} \{h' \bullet (u_H, x \mapsto w) \mid w \notin loc(h')\} & h = h' \bullet (u_H, x \mapsto v) \text{ for some } v \\ \top & \text{otherwise} \end{cases}$$

The function is local: if $h \neq h' \bullet (u_H, x \mapsto v)$ for some h' and v, then $AD(h) = \top$, and for any h'', $AD(h'' \bullet h) \sqsubseteq \top$. Otherwise, $h = h' \bullet (u_H, x \mapsto v)$ for some h' and v. Then for any h'', $AD(h'' \bullet h) = \{h'' \bullet h' \bullet (u_H, x \mapsto w) \mid w \notin loc(h'' \bullet h')\} \sqsubseteq \{h''\} * \{h' \bullet (u_H, x \mapsto w) \mid w \notin loc(h')\} = \{h''\} * AD(h)$.

3. **Operations on Integers.** We consider the separation algebra of integers under addition with identity 0. It can be seen that any ‘adding’ function $f(x) = \{x + c\}$ that adds a constant c is local, while a function that multiplies by a constant c, $f(x) = \{cx\}$, is non-local.

We now present the local Hoare reasoning framework for local functions on separation algebras. We treat predicates simply as subsets of the separation algebra.

Definition 4. *A* **predicate** *p over Σ is an element of the powerset $P(\Sigma)$.*

Note that the top element $\top$ is not a predicate and that the $*$ operator, although defined on $P(\Sigma)^\top \times P(\Sigma)^\top \rightarrow P(\Sigma)^\top$, acts as a binary connective on predicates. We have the distributive law for union: $(\bigsqcup X) * p = \bigsqcup\{x * p \mid x \in X\}$ where $X \subseteq P(\Sigma)$. The same is not true for intersection in general, but does hold for *precise* predicates.

Definition 5 (Precise Predicates). *A predicate $p \in P(\Sigma)$ is* **precise** *iff, for every $\sigma \in \Sigma$, there exists at most one $\sigma_p \in p$ such that $\sigma_p \preceq \sigma$.*

$\{l \mapsto v \mid v \in Val\}$ for some l is precise, while $\{l \mapsto v \mid l \in Loc\}$ for some v is not. Also, any singleton predicate $\{\sigma\}$ is precise.

Lemma 2 (Precision Characterization). *A predicate p is precise iff, for all $X \subseteq P(\Sigma)$, $(\bigsqcap X) * p = \bigsqcap\{x * p \mid x \in X\}$.*

In the introduction we discussed the intuition of how the footprints are expected to correspond to the *smallest* safe states. We will return to this point in section 6, employing the notion of a *saturated* predicate, which is one that is always falsified on bigger states. Saturated can also be called ‘anti-intuitionistic’, because an intuitionistic predicate is one that is never falsified on bigger states. Every precise predicate is also saturated.

Definition 6 (Saturated predicate). *A predicate $p \in P(\Sigma)$ is* **saturated** *if for every $\sigma \in p$, there is no σ' in p such that $\sigma \prec \sigma'$.*

Our Hoare reasoning system is a slight adaptation of Abstract Separation Logic [9], the difference being that we emphasise the notion of a specification as a tuple of pre- and post- conditions, rather than the usual Hoare triples that include the function. A triple is then equivalent to saying that a function f *satisfies* a tuple (p, q), written $f \models (p, q)$. This approach is very similar to the notion of the *specification statement* (a Hoare triple with a 'hole') introduced in [11], which is used in refinement calculi, and was also used to prove completeness of a local reasoning system in [19].

Definition 7 (Specification). *Let Σ be a separation algebra. A* **statement** *on Σ is a tuple (p, q), where $p, q \in P(\Sigma)$ are predicates representing pre- and post- conditions. A* **specification** *ϕ on Σ is a set of statements. We let $\Phi_\Sigma = P(P(\Sigma) \times P(\Sigma))$ be the set of all specifications on Σ. We shall exclude the subscript when it is clear from the context. The* **domain** *of a specification is defined as $D(\phi) = \bigsqcup\{p \mid (p, q) \in \phi\}$.* **Domain equivalence** *is defined as $\phi \cong_D \psi$ iff $D(\phi) = D(\psi)$.*

Thus the domain is the union of the preconditions of all the statements in the specification. It is one possible measure of *size*: how much of Σ the specification is referring to. We also adapt the notions of precise and saturated predicates to specifications.

Definition 8. *A specification is saturated iff its domain is saturated. It is precise iff its domain is precise.*

Definition 9 (Satisfaction). *A local function f satisfies a statement (p, q), written $f \models (p, q)$, iff, for all $\sigma \in p$, $f(\sigma) \sqsubseteq q$. f satisfies a specification $\phi \in \Phi$, written $f \models \phi$, iff $f \models (p, q)$ for all $(p, q) \in \phi$.*

Definition 10 (Semantic consequence). *Let $p, q, r, s \in P(\Sigma)$ and $\phi, \psi \in \Phi$. Each judgement $(p, q) \models (r, s), \phi \models (p, q)$, $(p, q) \models \phi$, and $\phi \models \psi$ holds iff all local functions that satisfy the left hand side also satisfy the right hand side.*

Proposition 1 (Order Characterization). *$f \sqsubseteq g$ iff, for all $p, q \in P(\Sigma)$, $g \models (p, q)$ implies $f \models (p, q)$.*

For every specification ϕ, there is a 'best' local function satisfying ϕ (lemma 3). This is of the same nature as the best local action of [9], except that we generalise to specifications rather than statements (single pre- and post-condition pairs).

Definition 11 (Best local action). *For a specification $\phi \in \Phi$, the* **best local action** *of ϕ, written $bla[\phi]$, is the function of type $\Sigma \to P(\Sigma)^\top$ defined by*

$$bla[\phi](\sigma) = \bigsqcap_{\sigma' \preceq \sigma, \sigma' \in p, (p,q) \in \phi} \{\sigma - \sigma'\} * q$$

Lemma 3. *Let $\phi \in \Phi$. The following hold:*

- *$bla[\phi]$ is local*
- *$bla[\phi] \models \phi$*
- *if f is local and $f \models \phi$ then $f \sqsubseteq bla[\phi]$.*

$$\frac{(p,q)}{(p*r,q*r)}\;\textit{Frame}\qquad \frac{p'\sqsubseteq p\quad (p,q)\quad q\sqsubseteq q'}{(p',q')}\;\textit{Consequence}\qquad \frac{(p_i,q_i),\ \text{all } i\in I}{(\bigsqcup_{i\in I}p_i,\bigsqcup_{i\in I}q_i)}\;\textit{Union}\qquad \frac{(p_i,q_i),\ \text{all } i\in I, I\neq\emptyset}{(\bigsqcap_{i\in I}p_i,\bigsqcap_{i\in I}q_i)}\;\textit{Intersection}$$

Fig. 1. Inference Rules for local Hoare reasoning

Lemma 4. *For $\phi \in \Phi$ and $p, q \in P(\Sigma)$, $bla[\phi] \models (p, q) \Leftrightarrow \phi \models (p, q)$.*

The inference rules of the proof system are given in figure 1. The system is sound and complete with respect to the satisfaction relation; the proof uses lemmas 3 and 4.

Definition 12 (Proof-theoretic consequence). *For statements p, q, r, s and specifications ϕ, ψ, each of the judgements $(p, q) \vdash (r, s), \phi \vdash (p, q)$, $(p, q) \vdash \phi$, and $\phi \vdash \psi$ holds iff the right-hand side is derivable from the left-hand side by the rules in figure 1.*

Theorem 1 (Soundness and Completeness). $\phi \models (p, q) \Leftrightarrow \phi \vdash (p, q)$.

3 Properties of Specifications

We discuss certain properties of specifications as a prerequisite for our main discussion on footprints in Section 4. We introduce the notion of a *complete* specification for a local function: a specification from which all properties that hold for the function can be derived in the proof system. However, a function may have many complete specifications, so we introduce a canonical form for specifications. We show that of all the complete specifications of a local function, there exists a unique canonical complete specification for every domain. As discussed in the introduction, an important notion of local reasoning is the *small specification* which completely describes the behaviour of a local function by mentioning only the footprint. Thus, as a prerequisite to investigating their existence, we formalise small specifications as complete specifications with the smallest possible domain. Similarly, we define *big* specifications as complete specifications with the biggest domain.

Definition 13 (Complete Specification). *A specification $\phi \in \Phi$ is a* **complete specification** *for f, written $complete(\phi, f)$, iff, for all $p, q \in P(\Sigma), f \models (p, q) \Leftrightarrow \phi \models (p, q)$. Let $\Phi_{comp(f)}$ be the set of all complete specifications of f.*

ϕ is complete for f whenever the tuples that hold for f are *exactly* the tuples that follow from ϕ. This also means that any two complete specfications ϕ and ψ for a local function are semantically equivalent, that is, $\phi \dashv\models \psi$. The following proposition illustrates how the notions of best local action and complete specification are closely related.

Proposition 2. *For all $\phi \in \Phi$ and local functions f, $complete(\phi, f) \Leftrightarrow f = bla[\phi]$.*

Any specification is therefore only complete for a unique local function, which is its best local action. However, a local function may have lots of complete specifications. We therefore introduce a canonical form for specifications.

Definition 14 (Canonicalisation). *The* **canonicalisation** *of a specification ϕ is defined as $\phi_{can} = \{(\{\sigma\}, bla[\phi](\sigma)) \mid \sigma \in D(\phi)\}$. A specification is in* **canonical** *form if it is equal to its canonicalisation. Let $\Phi_{can(f)}$ denote the set of all canonical complete specifications of f.*

Proposition 3. *For any specification ϕ, we have $\phi \dashv\vDash \phi_{can}$.*

Thus, the canonicalisation of a specification is logically equivalent to the specification. The following corollary shows that all complete specifications that have the same domain have a unique canonical form, and specifications of different domains have different canonical forms.

Corollary 1. *$\Phi_{can(f)}$ is isomorphic to the quotient set $\Phi_{comp(f)}/\cong_D$, under the equality-preserving isomorphism that maps $[\phi]_{\cong_D}$ to ϕ_{can}.*

Definition 15 (Small and Big specifications). *ϕ is a* **small specification** *for f iff $\phi \in \Phi_{comp(f)}$ and there is no $\psi \in \Phi_{comp(f)}$ such that $D(\psi) \sqsubset D(\phi)$. A* **big specification** *is defined similarly.*

Small and *big* specifications are thus the specifications with the smallest and biggest domains respectively. The question is if/when small and big specifications exist. The following result shows that a canonical big specification exists for every local function.

Proposition 4 (Big Specification). *For any local function f, the canonical big specification for f is given by $\phi_{big(f)} = \{(\{\sigma\}, f(\sigma)) \mid f(\sigma) \sqsubset \top\}$.*

Small specifications are used in local reasoning to completely specify the behaviour of an update command by only mentioning the behaviour of the command on the part of the resource that is affected by the command [13,4,7]. The question of the existence of small specifications is therefore strongly related to the concept of footprints. Finding a small specification is about finding the complete specification with the smallest possible domain, and therefore enquiring about which elements of Σ are essential and sufficient for a complete specification. This requires a formal characterisation of the footprint notion, which we shall now present.

4 Footprints

In the introduction we discussed how the *AD* program demonstrates that the footprints of a local function do not correspond simply to the smallest safe states, as these states alone do not always yield complete specifications. In this section we introduce the definition of footprint that does yield complete specifications. In order to understand what the footprint of a local function should be, we begin by analysing the definition of locality. Recall that the locality definition 3 says that the action on a certain state σ_1 imposes a *limit* on the action on a bigger state $\sigma_2 \bullet \sigma_1$. This limit is $\{\sigma_2\} * f(\sigma_1)$, that is, we have $f(\sigma_2 \bullet \sigma_1) \sqsubseteq \{\sigma_2\} * f(\sigma_1)$. A reformulation of this definition is that for any state σ, the action on that state has to be within the limit imposed by *every* state σ' that is smaller than it, and we therefore have

$$f(\sigma) \sqsubseteq \bigsqcap_{\sigma' \prec \sigma} \{\sigma - \sigma'\} * f(\sigma')$$

We define this overall constraint imposed on σ by all the smaller states as the *local limit* of f on σ, and show that the locality of a function is equivalent to it satisfying the local limit constraint.

Definition 16 (Local limit). *For a local function f on Σ, and $\sigma \in \Sigma$, the* **local limit** *of f on σ is defined as*

$$L_f(\sigma) = \bigsqcap_{\sigma' \prec \sigma} \{\sigma - \sigma'\} * f(\sigma')$$

Proposition 5. *f is local $\quad \Leftrightarrow \quad f(\sigma) \sqsubseteq L_f(\sigma)$ for all $\sigma \in \Sigma$*

With this intuition that the locality of f is determined by the local limit restricting the action of f, we define the footprints as those elements which further reduce this limit: σ is a footprint of f if and only if $f(\sigma) \sqsubset L_f(\sigma)$. If the result of the function is equal to the local limit on a certain element, then this can be determined just by the fact that f is a local function. If the result is strictly smaller, then this behaviour is additional to f being local. The intuition is that this characteristic property of the function would have to be explicitly stated in a complete specification of the function, which would make footprints the essential elements required to describe this function's behaviour. We formally prove this central result after stating the definition and illustrating it with examples.

Definition 17 (Footprint). *For a local function f and $\sigma \in \Sigma$, σ is a footprint of f, written $F_f(\sigma)$, iff $f(\sigma) \sqsubset L_f(\sigma)$. We denote the set of footprints of f by $F(f)$.*

Note that an element σ is therefore not a footprint iff the action of f on σ is at the local limit, that is $f(\sigma) = L_f(\sigma)$. With this definition of footprint, the smallest elements on which f is safe are always footprints. This is because f faults on everything smaller, and thus the local limit is $\top$. However, these elements are not always the only footprints. An example is the *AD* command discussed in the introduction. The empty heap is a footprint as it is the smallest safe heap, but the heap cell $l \mapsto v$ is also a footprint.

Example 2 (Dispose command). The footprints of the $dispose[l]$ command are the cells at location l. We check this by considering the following cases

1. The empty heap, u_H, is not a footprint since $L_{dispose[l]}(u_H) = \top = dispose[l](u_H)$
2. Every cell $l \mapsto v$ for some v is a footprint

$$L_{dispose[l]}(l \mapsto v) = \{l \mapsto v\} * dispose[l](u_H) = \{l \mapsto v\} * \top = \top$$
$$dispose[l](l \mapsto v) = \{u_H\} \sqsubset L_{dispose[l]}(l \mapsto v)$$

3. Every state σ such that $\sigma \succ (l \mapsto v)$ for some v is not a footprint

$$L_{dispose[l]}(\sigma) \sqsubseteq \{\sigma - (l \mapsto v)\} * dispose[l](l \mapsto v) = \{\sigma - (l \mapsto v)\} = dispose[l](\sigma)$$

By proposition 5, we have $L_{dispose[l]}(\sigma) = dispose[l](\sigma)$. The intuition is that σ does not characterise any 'new' behaviour of the function: its action on σ is just a consequence of its action on the cells at location l and the locality property of the function.

4. Every state σ such that $\sigma \not\succ (l \mapsto v)$ for some v is not a footprint

$$L_{dispose[l]}(\sigma) \sqsubseteq \{\sigma\} * dispose[l](u_H) = \{\sigma\} * \top = \top = dispose[l](\sigma)$$

   Again by proposition 5, $L_{dispose[l]}(\sigma) = dispose[l](\sigma)$.

*Example 3 (*AD *command).* The *AD* (Allocate-Deallocate) command was defined in example 1. We have the following cases for σ.

1. $\sigma \not\succeq (u_H, x \mapsto v_1)$ for some v_1 is not a footprint, since $L_{AD}(\sigma) = \top = AD(\sigma)$.
2. $\sigma = (u_H, x \mapsto v_1)$ for some v_1 is a footprint since $L_{AD}(\sigma) = \top$ (by case (1)) and $AD(\sigma) = \{(u_H, x \mapsto w) \mid w \in L\} \sqsubset L_{AD}(\sigma)$.
3. $\sigma = (l \mapsto v_1, x \mapsto v_2)$ for some l, v_1, v_2 is a footprint.

$$\begin{aligned} L_{AD}(\sigma) &= \{(l \mapsto v_1, u_S)\} * AD((u_H, x \mapsto v_2)) \\ &\qquad \text{(AD faults on all other elements strictly smaller than } \sigma\text{)} \\ &= \{(l \mapsto v_1, u_S)\} * \{(u_H, x \mapsto w) \mid w \in Loc\} \\ &= \{(l \mapsto v_1, x \mapsto w) \mid w \in Loc\} \end{aligned}$$

$$AD(\sigma) = \{(l \mapsto v_1, x \mapsto w) \mid w \in Loc, w \neq l\} \sqsubset L_{AD}(\sigma)$$

4. $\sigma = (h, x \mapsto v_1)$ for some v_1, and where $|loc(h)| > 1$, is not a footprint.

$$L_{AD}(\sigma) \sqsubseteq \bigsqcap_{(l \mapsto v) \prec h} \{(h - (l \mapsto v), u_S)\} * AD((l \mapsto v, x \mapsto v_1))$$

$$= \{(h, x \mapsto w) \mid w \notin loc(h)\} = AD(\sigma)$$

   By proposition 5 we get $L_{AD}(\sigma) = AD(\sigma)$.
5. $\sigma = (h, s \bullet (x \mapsto v_1))$ for some v_1 and $s \neq u_S$, is not a footprint.

$$L_{AD}(\sigma) \sqsubseteq \{(u_H, s)\} * AD((h, x \mapsto v_1)) = AD(\sigma)$$

   By proposition 5 we get $L_{AD}(\sigma) = AD(\sigma)$.

Our footprint definition therefore works properly for these specific examples. Now we give the formal general result which captures the underlying intuition of local reasoning that the footprints of a local function are the only essential elements for a complete specification of the function.

Theorem 2 (Essentiality). *The footprints of a local function are the essential domain elements for any complete specification of that function, that is,*

$$F_f(\sigma) \quad \Leftrightarrow \quad \forall \phi \in \Phi_{comp(f)}.\, \sigma \in D(\phi).$$

5 Sufficiency and Small Specifications

We know that the footprints are the only elements that are *essential* for a complete specification of a local function in the sense that every complete specification must include them. Now we ask when a set of elements is *sufficient* for a complete specification of a local function, in the sense that there exists a complete specification of the function that only includes these elements. In particular, we wish to know if the footprints alone are sufficient.

Definition 18 (Local limit imposed by a set). *For a subset A of Σ, the* **local limit** *imposed by A on the action of f on σ is defined by*

$$L_{A,f}(\sigma) = \bigsqcap_{\sigma' \preceq \sigma, \sigma' \in A} \{\sigma - \sigma'\} * f(\sigma')$$

Sometimes, the local limit imposed by A is enough to determine the complete behaviour of f. In this case, we call A a *basis* for f.

Definition 19 (Basis). *$A \sqsubseteq \Sigma$ is a* **basis** *for f, written $basis(A, f)$, iff $L_{A,f} = f$.*

This means that, when given the action of f on elements in A alone, we can determine the action of f on any element in Σ by just using the locality property of f. Every local function has at least one basis, namely the trivial basis Σ itself. We next show the correspondence between the bases and complete specifications of a local function.

Lemma 5. *Let $\phi_{A,f} = \{(\{\sigma\}, f(\sigma)\} \mid \sigma \in A, f(\sigma) \sqsubset \top\}$. Then we have $basis(A, f) \Leftrightarrow complete(\phi_{A,f}, f)$.*

For every canonical complete specification $\phi \in \Phi_{can(f)}$, we have $\phi = \phi_{D(\phi),f}$. By the previous lemma it follows that $D(\phi)$ forms a basis for f. The lemma therefore shows that every basis determines a complete canonical specification, and vice versa. This correspondence also carries over to all complete specifications for f by the fact that every domain-equivalent class of complete specifications for f is represented by the canonical complete specification with that domain (corollary 1). It is a simple consequence of the Essentiality theorem (2) that the footprints are present in every basis for a local function.

Lemma 6. *The footprints of f are included in every basis of f.*

The question of sufficiency is about how small the basis can get. Given a local function, we wish to know if it has a smallest basis. We know that every basis contains the footprints, but we would also like to know whether the footprints alone form a basis. This would mean that the function would have a complete specification that just mentions the footprints, so it would be a *smallest* complete specification. We find that for well-founded resource models, this is indeed the case.

Theorem 3 (Sufficiency I). *If a separation algebra Σ is well-founded under the $\preceq$ relation, then the footprints of any local function form a basis for it, that is, $f = L_{F(f),f}$.*

In section 3, the notions of big and small specifications were introduced, and the existence of a big specification was shown. We are now in a position to show the existence of the small specification for well-founded resource. If Σ is well-founded, then every local function has a small specification whose domain is the footprints of the function.

Corollary 2 (Small specification). *For well-founded separation algebras, every local function has a small specification given by $\phi_{F(f),f}$.*

Thus for well-founded resource, the footprints are always essential and sufficient, and specifications need not consider any other elements. In practice, small specifications may not always be in canonical form even though they always have the same domain as the canonical form. For example, the heap dispose command can have the specification $\{(\{l \mapsto v \mid v \in Val\}, \{u_H\})\}$ rather than $\{(\{l \mapsto v\}, \{u_H\}) \mid v \in Val\}$. Well-founded resource is usually the case in practice. One notable exception is the fractional permissions model [4] in which the resource includes permissions that can be indefinitely divided. In analysing the non-well-founded case, we found it important to distinguish between models that do or do not have *negativity*.

Definition 20 (Negativity). *A separation algebra Σ has* **negativity** *iff there exists a non-unit element $\sigma \in \Sigma$ that has an inverse, that is, $\sigma \neq u$ and $\sigma \bullet \sigma' = u$ for some $\sigma' \in \Sigma$. We say that Σ is* **non-negative** *if no such element exists.*

If a model has negativity then it is non-well-founded, because for elements σ and σ' such that $\sigma \bullet \sigma' = u$, the set $\{\sigma, u\}$ forms an infinite descending chain (there is no least element). All well-founded models are therefore non-negative. In the general non-negative case, we find that either the footprints form a basis, or there is no smallest basis.

Theorem 4 (Sufficiceny II). *If Σ is non-negative then, for any local f, either the footprints form a smallest basis or there is no smallest basis for f.*

Corollary 3 (Small Specification). *If Σ is non-negative, then every local function either has a small specification given by $\phi_{F(f),f}$ or there is no smallest complete specification for that function.*

Example 4 (Permissions). The fractional permissions model [4] is non-well-founded and non-negative. It can be represented by the separation algebra $HPerm = L \rightharpoonup_{fin} Val \times P$ where L and Val are as in example 1, and P is the interval (0, 1] of rational numbers. Elements of P represent 'permissions' to access a heap cell. A permission of 1 for a cell means both read and write access, while any permission less than 1 is read-only access. $\bullet$ joins disjoint heaps and adds the permissions together for any cells that are present in both heaps only if the resulting permission for each heap cell does not exceed 1, and the operation is undefined otherwise. In this case, the write function that updates the value at a location requires a permission of at least 1 and faults on any smaller permission. It therefore has a small specification with precondition being the cell with permission 1. The read function, however, can execute safely on any positive permission, no matter how small. Thus this function can be completely specified with a specification that has a precondition given by the cell with permission z, for all $0 < z \leq 1$. However, this is not a *smallest* specification, as a smaller one can be given by further restricting $0 < z \leq 0.5$. We can therefore always find a smaller specification by reducing the value of z but keeping it positive.

For resource with negativity, we find that it is possible to have small specifications that include non-essential elements (which by theorem 2 are not footprints). These elements are non-essential in the sense that complete specifications exist that do not include them, but there is no complete specification that includes only essential elements.

Example 5 (Integers). An example of a model with negativity is the separation algebra of integers $(\mathbb{Z}, +, 0)$. In this case there can be local functions which can have small specifications that contain non-footprints. Let $f : \mathbb{Z} \rightarrow P(\mathbb{Z})^{\top}$ be defined as $f(n) = \{n + c\}$ for some constant c, as in example 1. f is local, but it has no footprints. This is because for any n, $f(n) = 1 + f(n - 1)$, and so n is not a footprint of f. However, f does have small specifications, for example, $\{(\{0\}, \{c\})\}$, $\{(\{5\}, \{5 + c\})\}$, or indeed $\{(\{n\}, \{n + c\})\}$ for any $n \in \mathbb{Z}$. So although every element is non-essential, some element is required to give a complete specification.

6 One-Step Local Functions

In the introduction, we described a common, but mistaken, intuition that the footprint should correspond to the minimal resource on which the function is safe, and that the behaviour of the function on larger states should be derivable from these minimal states. The intuition fails due to the subtle nature of the locality condition as shown by the *AD* example. However, based on our investigation of footprints, we are now in a position to determine a natural class of local functions for which this basic intuition indeed holds. We call these the *one-step* local functions.

Definition 21 (One-step local function). *A local function is* **one-step** *if it has a saturated basis. It is* **precise** *if it has a precise basis.*

Note that every precise local function is one-step, because every precise predicate is saturated. For any local function, the footprints are the 'stepping points' in the sense that if we start from the bottom element of Σ and go up any ascending chain, the footprints are the points which *add* to the locality constraint by restricting the action on elements above them. For one-step local functions, for any such ascending chain, there is at most a *single* such point along any chain (which is actually the point at which the function becomes safe as shown by proposition 6), and beyond that point, the action of the function is just given by the local limit. So there is at most a single footprint on any ascending chain. Hence the name 'one-step' in analogy with the *unit (or heaviside)* step functions that act as 'on/off switches'. Along any ascending chain in the partial order of Σ, there is a single point at which the function 'turns on', that is, becomes safe.

Precise local functions are one-step functions with the added property that those ascending chains never intersect, that is, any two footprints would never have a common ancestor. So for example, the dispose function is one-step as its behaviour changes at a single point, which is the cell being deleted. It is also precise because there is no heap that contains two cells with location l pointing to different values. For the *AD* function, there are two stepping points (u_H and the single heap cells) along an ascending chain, so it is not a one-step function.

Example 6 (One-step local functions)

1. Dispose. The heap command $dispose[l]$ is a precise local function. This is because the heap algebra is well-founded, so the footprints form a basis and, as shown in section 4, the set of footprints is $\{l \mapsto v \mid v \in Val\}$ which is a precise predicate.

2. *AD*. We showed in section 4 that the set of footprints is $\{(u_H, x \mapsto u) \mid u \in Val\} \cup \{(l \mapsto u, x \mapsto v) \mid u, v \in Val\}$. This is a non-saturated predicate, so there is no saturated basis for the *AD* function. It is therefore not one-step.
3. Multiple Dispose. This is the command $dispose[l_1, l_2]$ that faults if neither l_1 or l_2 are present, disposes l_1 if it is present and l_2 is not, disposes l_2 if it is present and l_1 is not, and diverges if both are present. Note that it has to diverge if both are present, otherwise it would not be local. This is an example of a one-step local function that is not precise: the set of footprints is $\{l_1 \mapsto v \mid v \in Val\} \cup \{l_2 \mapsto v \mid v \in Val\}$ and this is a saturated but imprecise predicate.

Proposition 6. *If f is a one-step local function, then its footprints are the smallest states on which the function is safe: $F(f) = min(\{\sigma \mid f(\sigma) \sqsubset \top\})$.*

One-step and precise local functions also have advantageous properties with respect to their specifications. We can determine whether a function is one-step or precise by just looking at its specifications (checking specifications to be saturated or precise is easier than checking functions to be one-step or precise).

Proposition 7. *A local function f is one-step iff it has at least one saturated complete specification. It is precise iff it has at least one precise complete specification.*

We can also sometimes determine the footprints of local functions directly from the specifications, without knowing what the function itself is. If the resource is non-negative and f has a saturated complete specification, then the domain of this specification is the set of footprints of f.

Proposition 8. *If Σ is non-negative and ϕ is a saturated complete specification for f, then the domain of ϕ is the set of footprints of f: that is, $F(f) = D(\phi)$.*

7 Conclusion and Future Work

We have defined the footprints of a local function and demonstrated that they are the only essential elements necessary to obtain complete specifications for local Hoare reasoning. For well-founded resource, the footprints are also sufficient, meaning that they do indeed yield the smallest complete specification. We have therefore solved the footprint problem highlighted in the introduction. We have also given results for the non-well-founded models and have introduced the natural class of one-step local functions for which the footprints are the smallest safe states.

Although we now know what footprints are, there is still much to investigate about their properties. Two important questions for future work are how footprints behave under the sequential composition of functions (we know from examples that it is not simply the union of the sets of footprints) and how the footprints impact on concurrent reasoning where identifying the minimal resource is of paramount importance.

Another question of interest is whether we can regain the simple intuition of footprints as minimal safe states by refining the semantics. We have already identified the natural class of one-step local functions for which the footprints do indeed have this property. However, these one-step functions are not compositional in general, as our *AD* program on the standard heap model illustrates. We are currently exploring an adapted heap model, where the state keeps track of the allocated cells, aiming for the result that sequential composition does preserve our one-step property for this model. More generally, we will seek conditions under which one-step functions do indeed compose; such conditions should be satisfied by our alternative heap model.

Acknowledgements. We thank Calcagno, O'Hearn and Yang for detailed discussions on footprints. Raza acknowledges support of an ORS award. Gardner acknowledges support of a Microsoft/Royal Academy of Engineering Senior Research Fellowship.

References

1. Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P., Wies, T., Yang, H.: Shape Analysis for Composite Data Structures. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, Springer, Heidelberg (2007)
2. Berdine, J., Calcagno, C., OHearn, P.W.: Smallfoot: Automatic modular assertion checking with separation logic. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2005. LNCS, vol. 4111, Springer, Heidelberg (2006)
3. Birkedal, L., Yang, H.: Relational parametricity and separation logic. In: Seidl, H. (ed.) FOSSACS 2007. LNCS, vol. 4423, Springer, Heidelberg (2007)
4. Bornat, R., Calcagno, C., O'Hearn, P., Parkinson, M.: Permission accounting in separation logic. In: 32nd POPL (2005)
5. Bornat, R., Calcagno, C., Yang, H.: Variables as resource in separation logic. In: 21st MFPS (2005)
6. Brookes, S.D.: A semantics for concurrent separation logic. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, Springer, Heidelberg (2004)
7. Calcagno, C., Gardner, P., Zarfaty, U.: Context logic and tree update. In: 32nd POPL (2005)
8. Calcagno, C., Gardner, P., Zarfaty, U.: Local Reasoning about Data Update. In: Gordon Plotkin's festschrift, ENTCS (2007)
9. Calcagno, C., O'Hearn, P., Yang, H.: Local Action and Abstract Separation Logic (Longer version). In: LICS (2007)
10. Isthiaq, S., O'Hearn, P.W.: BI as an assertion language for mutable data structures. In: 28th POPL (2001)
11. Morgan, C.C.: The specification statement. In: ACM Transactions on Programming Languages and Systems (1988)
12. O'Hearn, P.: Resources, concurrency and local reasoning. Theoretical Computer Science, Preliminary version appeared in CONCUR 2004 (2007)
13. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL 2001. LNCS, vol. 2142, Springer, Heidelberg (2001)
14. O'Hearn, P.W., Pym, D.J.: The logic of bunched implications. In: Bulletin of Symbolic Logic (1999)
15. Parkinson, M., Bornat, R., Calcagno, C.: Variables as resource in Hoare logics. In: 21st LICS (2006)
16. Pym, D., O'Hearn, P., Yang, H.: Possible worlds and resources: The semantics of BI. In: Theoretical Computer Science (2004)
17. Pym, D.J.: The Semantics and Proof Theory of the Logic of Bunched Implications. Applied Logic Series. Kluwer Academic Publishers, Dordrecht (2002)
18. Reynolds, J.C.: Separation logic: A logic for shared mutable data structures. In: 17th LICS (2002)
19. Yang, H., O'Hearn, P.: A semantic basis for local reasoning. In: Nielsen, M., Engberg, U. (eds.) FOSSACS 2002. LNCS, vol. 2303, Springer, Heidelberg (2002)