Abstract Dynamic Frames

Han-Hing Dang

Institut für Informatik, Universität Augsburg, D-86159 Augsburg, Germany
h.dang@informatik.uni-augsburg.de

Abstract. Based on a former relation-algebraic approach to separation logic we present an abstraction of the theory of dynamic frames and algebraically describe concepts, properties and behaviour of that theory in a pointfree fashion. Moreover, relationships to abstract concepts of separation logic are given to pave the way for a unified treatment of both approaches. In particular, we also sketch the main ideas within the framework of local actions.

Keywords: Frame problem, local actions, relational semantics, separation algebra

1 Introduction

For obtaining a methodology that guarantees modularity and hence scalability in specification and correctness proofs of computer programs, an adequate solution to the frame problem [MH69] is required. The frame problem asks for a methodology that allows specifying which resources of a program can be changed and which ones are left unchanged without naming them explicitly. A popular approach to this problem is the theory of dynamic frames [Kas11] that provides the mentioned modularity while still being expressive enough to handle a variety of useful programs. Further variations of the theory address the automation of program verification (e.g., [SJP09, Lei10, GGN11]).

Another approach to the frame problem is given by separation logic [Rey02] which allows, due to its popular frame rule, modular reasoning about parts of a program without the need to construct a program proof in a larger context anew. For this logic there exist a few abstract and algebraic approaches that are used to extract and formalise general behaviour [COY07, DHM11, HHM+11, DM12], also in the case of concurrency. Unfortunately, for the theory of dynamic frames such approaches and considerations barely exist.

In the present paper we revisit a former relation-based algebraic calculus [DHM11, DM12] that was used as a formal base for pointfree proofs of inference rules of separation logic and combine it with the theory of dynamic frames.

The contributions of this work comprise an abstract treatment of the resources and locations dealt with in that theory, based on separation algebras. Moreover we give point-free characterisations and proofs of crucial concepts within the extended relational approach of [DM12] and explain their concrete
meaning. By this, the relational calculus extends towards a unifying approach for dynamic frames and separation logic.

The structure of this paper is as follows. First, we present all required basic definitions of separation algebras and the extended relational structure. In Section 3 we give pointfree variants of framing requirements and consequences of this. By this, Section 4 abstractly clarifies the relationship between locality principles and their application in accumulating frames. We conclude this work with a discussion on the relationship to so-called local actions.

2 Basics of the Algebraic Structure

This section provides the formal background to abstractly characterise dynamic frames. Framing requirements are defined in [Kas11] using a relational style. This motivates the idea to use the relation-algebraic structures of [DHM11, DM12] as an abstract base.

2.1 Separation Algebras

Before giving basic definitions and direct consequences of the algebra we start with the concept of separation algebras that provides a general way to characterise the structure and properties of resources [COY07].

Definition 2.1 A separation algebra is a cancellative and partial commutative monoid that we denote by $(\Sigma, \bullet, u)$. Elements of the algebra are called states and denoted by $\sigma, \tau, \ldots \in \Sigma$. Due to partiality two terms are defined to be equal iff both are defined and equal or both terms are undefined. This induces a combinability relation $\#$ defined by

$$\sigma_0 \# \sigma_1 \iff \sigma_0 \bullet \sigma_1 \text{ is defined}$$

and a substate relation given for $\sigma_0, \sigma_1 \in \Sigma$ by

$$\sigma_0 \preceq \sigma_1 \iff \exists \sigma_2. \sigma_0 \bullet \sigma_2 = \sigma_1.$$

When writing $\sigma \bullet \tau$ for states $\sigma, \tau$ we will implicitly assume $\sigma \# \tau$ in the following.

The empty state $u$ is the unit of the partial binary operator $\bullet$ which, additionally satisfies cancellativity, i.e., $\sigma_1 \bullet \tau = \sigma_2 \bullet \tau \Rightarrow \sigma_1 = \sigma_2$ for arbitrary states $\sigma_1, \sigma_2, \tau$.

A concrete instance of a separation algebra can be found in the dynamic frames setting. Resources or states in that approach are finite mappings from an infinite set of locations $\text{Loc}$ to an infinite set of values $\text{Val}$ that comprises at least integers and Booleans. Formally, we use the concrete dynamic frames separation algebra $\text{DFSA} =_{df} (\text{Loc} \rightarrow \cup, \emptyset)$ where $\cup$ denotes union of location-disjoint functions, $\emptyset$ the completely undefined function and $\sigma \# \tau \iff \text{dom}(\sigma) \cap \text{dom}(\tau) = \emptyset$. We write $\text{dom}(\sigma)$ for a mapping or state $\sigma$ to denote its domain or more concretely all of its allocated locations, i.e., a subset of $\text{Loc}$. Moreover, we define the substate $\sigma|_X$ that restricts the domain of the state $\sigma$ to a set of locations $X$. 

2
Lemma 2.2 For a state $\tau$ assume $\text{dom}(\tau) = X$. Then for arbitrary $\sigma$ we have $(\sigma \cdot \tau)|_X = \tau$.

We continue to characterise and manage several central properties of the dynamic frames approach within the abstraction to separation algebras. For this we require additional assumptions given in [DHA09] and basically follow the approach of that work. A separation algebra $(\Sigma, \cdot, u)$ satisfies disjointness iff for all $\sigma, \tau$

$$\sigma \cdot \sigma = \tau \Rightarrow \sigma = \tau$$

and it satisfies cross-split iff for arbitrary $\sigma_i$ with $i \in \{1, 2, 3, 4\}$

$$\sigma_1 \cdot \sigma_2 = \sigma_3 \cdot \sigma_4 \Rightarrow \exists \sigma_{13}, \sigma_{14}, \sigma_{23}, \sigma_{24}. \sigma_1 = \sigma_{13} \cdot \sigma_{14} \land \sigma_2 = \sigma_{23} \cdot \sigma_{24} \land \sigma_3 = \sigma_{13} \cdot \sigma_{23} \land \sigma_4 = \sigma_{14} \cdot \sigma_{24} .$$

Disjointness in the presence of cancellativity implies that the only element that can be combined with itself is the neutral element $u$, i.e.,

$$\sigma \# \sigma \Rightarrow \sigma = u .$$

Equivalently, non-unit elements cannot be combined with themselves since any allocated resources will overlap in such products. Therefore, the condition of (1) is called disjointness.

For a proof of (3) assume a state $\sigma$ that satisfies $\sigma \# \sigma$. By definition of $\#$, Equation (1), and a logic step:

$$\sigma \# \sigma \iff (\exists \tau. \sigma \cdot \sigma = \tau) \Rightarrow (\tau = \sigma) \Rightarrow (\sigma \cdot \sigma = \sigma) .$$

Now, by cancellativity we can infer $u \cdot \sigma = \sigma \cdot \sigma \Rightarrow u = \sigma$.

To explain the idea of the cross-split assumption, assume that a state can be combined in two ways or that there exist two possible splits of a state. Then there need to exist four substates that represent a partition of the original state w.r.t. the mentioned splits. The partitions of the state can be depicted as follows:

For the remaining sections we assume separation algebras that satisfy disjointness and cross-split. A concrete example of such a separation algebra can be found in [HV13]. The assumptions are required there to establish basic properties of operators for reasoning about sharing within data structures. Note that the separation algebra $\text{DFSA}$ also satisfies disjointness and cross-split.

2.2 The Relational Structure

In what follows we define a relational structure enriched by an operator $\ast$ that is also called separating conjunction. It ensures disjointness of program states or executions on disjoint states (cf. [DM12]).
**Definition 2.3** Assume a separation algebra \((\Sigma, \bullet, u)\). A command is a relation \(P \subseteq \Sigma \times \Sigma\). Relational composition of commands is denoted by \(\circ\). Its unit \(\text{skip} =_df \{(\sigma, \sigma) : \sigma \in \Sigma\}\) is the identity relation while the universal relation is denoted by \(\top\). Tests are special commands \(p, q, r\) that satisfy \(p \subseteq \text{skip}\). As particular tests we define \(\text{emp} =_df \{(u, u)\}\) that characterises the empty state \(u\) and \(\lceil P \rceil\) that represents the domain of a command \(P\). It is characterised by the universal property

\[
\lceil P \rceil \subseteq q \iff P \subseteq q ; P
\]

where \(q\) is an arbitrary test. In particular, \(P \subseteq \lceil P \rceil ; P\) and hence \(P = \lceil P \rceil ; P\). Moreover, we have \(\lceil P \rceil = (P ; \top) \cap \text{skip}\).

Note that tests form a Boolean algebra with \(\text{skip}\) as its greatest and \(\emptyset\) as its least element w.r.t. \(\subseteq\). Moreover, on tests \(\cup\) coincides with join and \(\circ\) with meet. In particular, tests are idempotent and commute under composition, i.e., \(p ; p = p\) and \(p ; q = q ; p\).

We now come to the definitions to introduce state separation relationally. Separation of commands can be interpreted either as their parallel execution on disjoint portions of states or, in the special case of tests as assertions characterising disjoint resources. For both cases we need a concept to be able to reason independently on disjoint portions of resources relationally.

First we introduce the Cartesian product \(P \times Q\) of commands \(P, Q\) by

\[
(\sigma_1, \sigma_2) \ (P \times Q) \ (\tau_1, \tau_2) =_df \sigma_1 P \tau_1 \land \sigma_2 Q \tau_2.
\]

Union, inclusion and intersection of such relations on pairs are straightforward while composition is defined componentwise.

We assume that \(\circ\) binds tighter than \(\times\) and \(\cap\). It is clear that \(\text{skip} \times \text{skip}\) is the identity of \(\circ\) on products. Note that \(\times\) and \(\circ\) satisfy an *equational* exchange law:

\[
P ; Q \times R ; S = (P \times R) ; (Q \times S).
\]

Pairs of tests are subidentities w.r.t. \(\text{skip} \times \text{skip}\) and thus are idempotent and commute under \(\circ\). A special test is given by the combinability check \(\#\) [DM12], on pairs of states:

\[
(\sigma_1, \sigma_2) \ # (\tau_1, \tau_2) =_df \sigma_1 \ # \sigma_2 \land \sigma_1 = \tau_1 \land \sigma_2 = \tau_2.
\]

It main usage is to rule out pairs of incompatible states that can occur within products \(P \times Q\) for arbitrary commands \(P, Q\).

As in [DHM11], we connect the pairs of states with single states using the so-called *split* relation \(\triangleleft\) and its converse *join* \(\triangleright\) defined by

\[
\sigma \triangleleft (\sigma_1, \sigma_2) =_df (\sigma_1, \sigma_2) \triangleright \sigma =_df \sigma_1 \# \sigma_2 \land \sigma = \sigma_1 \bullet \sigma_2.
\]

**Corollary 2.4** \(\# ; \triangleright = \triangleright\) and symmetrically \(\triangleleft ; \# = \triangleleft\). Moreover, \(\# \subseteq \triangleright ; \triangleleft\).
Corollary 2.5 (Forward/Backward Compatibility) For tests $p, q$ we have
$\# ; (p \times q) = (p \times q) ; \#$.

The intuition for this inequation is that tests do not change states as subidentities
and hence starting from compatible states they return the same compatible ones.
Finally, the $*$-composition of commands $P, Q$ is defined by

$$P \ast Q =_{df} \langle ; (P \times Q) ; \rangle.$$

For states $\sigma, \tau$ we have $\sigma (P \ast Q) \tau$ iff $\sigma$ can be split into states $\sigma_P, \sigma_Q$ on
which $P$ and $Q$ can act and produce results $\tau_P, \tau_Q$ that are again combinable to
$\tau = \tau_P \bullet \tau_Q$. Hence $P \ast Q$ also provides a possibility to characterise the structure
of commands and hence their behaviour on parts of a state.

Moreover, $P \ast Q$ can also be interpreted as the concurrent execution of pro-
grams $P, Q$ running on combinable or disjoint sets of resources [DM12, DM13].

Note that for tests $p, q$ the command $p \ast q$ is also a test and in particular,
$\text{skip} \ast \text{skip} = \text{skip}$. Moreover, $*$ is associative and commutative and $\text{emp}$ is its unit.

For readers familiar with fork algebras (e.g., [FBH97]) we remark that using
the pairing operation $\star (\sigma, \tau) = (\sigma, \tau)$ one has the relationship

$$\langle = (\langle = \Sigma \langle = \rangle) ; = \rangle \cap = \bullet$$

where $\Sigma$ denotes the fork operator, $\geq$ the converse of $\leq$ and $\sigma (\bullet)(\sigma_1, \sigma_2) \leftrightarrow \sigma = \sigma_1 \bullet \sigma_2$. Moreover, the Cartesian products coincides with the direct product,
i.e., $P \times Q = P \otimes Q$. For the sake of simplicity we stay with the above given
definitions, since the additional constructs provided by fork algebras are not
required for our purposes.

3 Abstracting Dynamic Frames

Dynamic frames are represented in concrete program specifications as specification
variables, i.e., variables that serve only for verification purposes and hence
are not physically visible in the program itself. Their usage is to cover a set of
locations of a state $\sigma$ ranging over variables or allocated objects. By this mech-
anism one obtains the expressiveness to specify what a program or a method is
allowed to modify and what remains untouched during its execution.

For an abstraction of the theory of dynamic frames we start by considering
the concrete separation algebra $\text{DFSA}$. Frequently used examples in the theory
of dynamic frames are the auxiliary specification variables

$$\text{used} = \text{used}_\sigma =_{df} \text{dom}(\sigma) \quad \text{and} \quad \text{unused} =_{df} \text{Loc} - \text{used}.$$

The former denotes the set of locations to which the state $\sigma$ assigns values while
the latter corresponds to all unallocated ones in that state. A dynamic frame $f$
at a state $\sigma$ is defined as a subset of $\text{Loc}$ satisfying $f \subseteq \text{used}$. Hence, dynamic
frames are state dependent and may vary with state transitions, i.e., considering
$\sigma P \sigma'$ for a command $P$ and a dynamic frame $f$ in $\sigma$ then generally $f$ in $\sigma'$ will
capture a different set of locations. Following the notation in [Kas11] a dynamic frame \( f \) in a final state \( \sigma' \) is denoted by \( f' \), i.e., it would correspond to \( f_{\sigma'} \).

Our central goal is to derive an abstract and pointfree relational treatment of dynamic frames. Therefore, we are mainly interested in extracting behavioural patterns and aspects or effects of these. For a relational treatment we use a constant set of locations representing an initial dynamic frame \( f \). The dynamic behaviour within state transitions \( \sigma P \sigma' \) will be represented by relational and pointfree formalisations rather than using functions or expressions that depend on the states \( \sigma \) or \( \sigma' \). This will allow more concise structural characterisations and pointfree proofs of basic properties involving dynamic frames.

Concretely, assuming an initial dynamic frame \( f \) to be a fixed set of locations we define

\[
[f] =_df \{ (\sigma, \sigma') : f = \text{dom}(\sigma) \},
\]

i.e., embedding \( f \) as a relation yields a subidentity which characterises all states where the allocated set of locations equals \( f \). Note that \([f]\) \(\neq\) \(\emptyset\), even if \( f = \emptyset \), because then \([f]\) = \(\{(u, u)\}\). For better readability we will omit the \([\ ]\) brackets in the following. The context will disambiguate the usage.

This embedding of \( f \) implies that the corresponding test satisfies a special behaviour which coincides with a pointfree characterisation of so-called precise tests [DM13]:

\[
(f \times \text{skip}) : \triangleright ; \triangleleft ; (f \times \text{skip}) \subseteq f \times \text{skip}.
\] (7)

In a pointwise form it reads for arbitrary states \( \sigma, \sigma_1, \sigma_2 \)

\[
(\sigma_1 \in f \land \sigma_2 \in f \land \sigma_1 \preceq \sigma \land \sigma_2 \preceq \sigma) \Rightarrow \sigma_1 = \sigma_2,
\]

where \( \tau \in f \Leftrightarrow \_df \tau \in f \tau \) for arbitrary states \( \tau \) and test \( f \). This means that in any state \( \tau \) a unique substate w.r.t. \( \preceq \) that contains exactly the locations of \( f \) can always be pointed out.

As the next step we introduce pointfree relational variants of framing requirements that are crucial for the theory of dynamic frames [Kas11].

**Definition 3.1 (Framing Requirements)** Assume a dynamic frame \( f \). Then the modification command \( \Delta(\_\) and preservation command \( \Xi(\_\) are defined by

\[
\Delta f =_df \{ (\sigma, \sigma') : \sigma|_{\text{used}-f} = \sigma'|_{\text{used}-f} \},
\]

\[
\Xi f =_df \{ (\sigma, \sigma') : \sigma|_f = \sigma'|_f \}.
\]

The modification requirement \( \Delta f \) intuitively asserts that at most resources captured by the frame \( f \) can be changed while any other resources remain untouched and hence are not modified. In particular, \( \Delta f \) allows the allocation of fresh storage. Conversely, \( \Xi f \) asserts that at least the state parts characterised by \( f \) are not changed while anything else can be changed arbitrarily.

**Theorem 3.2** Assume a dynamic frame \( f \). Then

\[
\Delta f = (f ; \top) * \text{skip} \quad \text{and} \quad \Xi f = f * \top.
\]
Proof. By definition of $\Delta$, definition of skip, by set theory and definition of $\top$, using $f$ is a test, definition of $;$, and definition of $*$:

\[
\begin{align*}
\sigma (\Delta f) \sigma' \\
\iff \sigma|_{\text{used} - f} = \sigma'|_{\text{used} - f} \\
\iff \sigma|_{\text{used} - f} \text{ skip } \sigma'|_{\text{used} - f} \\
\iff \sigma|_{\text{used} - f} \text{ skip } \sigma'|_{\text{used} - f} \wedge \sigma|_{\top f} \sigma'|_{\text{used} - f} - (\text{used} - f) \\
\iff \sigma|_{\text{used} - f} \text{ skip } \sigma'|_{\text{used} - f} \wedge \sigma|_{\top f} \sigma'|_{\text{used} - f} - (\text{used} - f) \wedge \sigma|_{\top f} \sigma|_{\text{used} - f} \\
\iff \sigma|_{\text{used} - f} \text{ skip } \sigma'|_{\text{used} - f} \wedge \sigma|_{f ; \top f \sigma'}_{\text{used} - f} \\
\Rightarrow \sigma (f ; \top \text{ skip}) \sigma' .
\end{align*}
\]

For the reverse implication assume states $\sigma_f, \sigma_{\text{skip}}, \sigma_{\top}$ with $\sigma_f \in f \wedge \sigma = \sigma_f \bullet \sigma_{\text{skip}} \wedge \sigma' = \sigma_{\top} \bullet \sigma_{\text{skip}}$. Using Lemma 2.2 we get $\sigma|_f = (\sigma_f \bullet \sigma_{\text{skip}})|_f = \sigma_f$.

Hence, $\sigma = \sigma_f \bullet \sigma|_{\text{used} - f}$ and cancellativity implies $\sigma_{\text{skip}} = \sigma|_{\text{used} - f}$. Moreover, we can infer $\sigma'|_{\text{used} - f} = (\sigma_{\top} \bullet \sigma_{\text{skip}})|_{\text{used} - f} = (\sigma_{\top} \bullet \sigma|_{\text{used} - f})|_{\text{used} - f} = \sigma|_{\text{used} - f}$.

Now Lemma 2.2 implies $\sigma_{\top} = \sigma'|_{\text{used} - f}$.

By definition of $\Xi$, $f$ is a test, set theory and definition of $\top$, and definition of $*$:

\[
\begin{align*}
\sigma (\Xi f) \sigma' \\
\iff \sigma|_f = \sigma'|_f \\
\iff \sigma|_f f \sigma'|_f \\
\iff \sigma|_f f \sigma'|_f \wedge \sigma|_{\text{used} - f} \top \sigma'|_{\text{used} - f} \\
\Rightarrow \sigma (f \ast \top \text{ skip}) \sigma' .
\end{align*}
\]

The reverse implication can be proved analogously to the above case. \hfill \Box

The algebraic embedding of dynamic frames as precise tests and their use in pointfree characterisations of the framing requirements yields the abstraction from the concrete DFSA separation algebra to arbitrary ones mentioned in Section 2.1. Moreover this allows calculational proofs of fundamental properties that establish the theory as a solution to tackle the frame problem (cf. Section 1).

We begin with the following result: Assume two initial disjoint sets of locations $f, g$ where only locations of $f$ can be modified, then all locations of $g$ will remain unchanged. The general idea of this is that expressions depending on locations of $f$ will not affect expressions that depend only on locations in $g$.

**Lemma 3.3** Assume dynamic frames $f, g$. Then

\[(f \ast g \ast \text{skip}) : \Delta f \subseteq g \ast \Delta f .\]

**Proof.** By Theorem 3.2, definition of $\ast$, neutrality of skip and Equation (5), $f$ is precise (Equation (7)), skip is neutral and Equation (5), definition of $\ast$, commutativity of $\ast$ and Theorem 3.2,

\[
\begin{align*}
(f \ast g \ast \text{skip}) : \Delta f \\
= (f \ast g \ast \text{skip}) : ((f ; \top) \ast \text{skip})
\end{align*}
\]
= \llcorner; (f \times (g \ast \text{skip})) ; \lrcorner; \llcorner; (f ; \top \times \text{skip}) ; \lrcorner
= \llcorner; (\text{skip} \times (g \ast \text{skip})) ; (f \times \text{skip}) ; \lrcorner; \llcorner; (\text{skip} ; (\top \times \text{skip}) ; \lrcorner
\subseteq \llcorner; (f ; \top \times (g \ast \text{skip})) ; (f \times \text{skip}) ; \lrcorner; \llcorner; (f ; \top \times \text{skip}) ; \lrcorner
= \llcorner; (f ; \top \times (g \ast \text{skip})) ; (f \times \text{skip}) ; \lrcorner
= (f ; \top) \ast g \ast \text{skip}
= g \ast \Delta f .

Since a dynamic frame \( f \) covers a set of locations on a state, it can be concluded that as long as \( f \) is not changed then all variables and expressions that depend on its locations will also remain unchanged. Expressions \( E \) can be abstracted relationally to tests that only include the states that assign values to at least all free variables occurring in \( E \). Abstractly we define that a dynamic frame \( f \) frames a test \( E \) iff

\[ (E \ast \text{skip}) ; \Xi f \subseteq E \ast \top . \quad (8) \]

\( \Xi f \) states that dynamic frame \( f \) is preserved while the test \( E \ast \text{skip} \) assumes a starting state \( \sigma \) that contains at least the required locations of \( E \). Now by the relation \( E \ast \top \) we can conclude that these locations will not be modified in a final state \( \sigma' \) since \( E \) is a test.

Altogether we can now prove a central theorem of the dynamic frames theory, stating that a dynamic frame will preserve its values while modifications on a disjoint frame are performed.

**Lemma 3.4 (Value preservation)** Assume dynamic frames \( f, g \). If \( g \) frames a test \( E \) then

\[ (E \ast \text{skip}) ; (f \ast g \ast \text{skip}) ; \Delta f \subseteq E \ast \top . \]

**Proof.** By Lemma 3.3, isotony, Theorem 3.2 and \( g \) frames \( E \) (Equation (8)),

\[ (E \ast \text{skip}) ; (f \ast g \ast \text{skip}) ; \Delta f \subseteq (E \ast \text{skip}) ; (g \ast \Delta f) \subseteq (E \ast \text{skip}) ; (g \ast \top) \subseteq E \ast \top . \]

The abstraction of dynamic frames to sets of locations and representing them relationally as precise tests implies that they already come with the so-called self-framing property. It is used in the program specifications of [Kas11] to maintain that initial disjointness of dynamic frames is preserved in final states. Concretely it characterises a dynamic frame to be preserved whenever the environment does not change its value.

**Lemma 3.5** Dynamic frames are self-framing.

**Proof.** Follows directly from \( f \ast \text{skip} \subseteq \text{skip} \), isotony of \( ; \) and Theorem 3.2. \( \Box \)

Basically, dynamic frames in concrete verification applications are always defined to be self-framing. Hence, this does not impose a restriction on the theory.

We continue with an auxiliary result that is required for later calculations.

**Lemma 3.6** For a dynamic frame \( f \) we have \( \llcorner (\Delta f) = f \ast \text{skip} = \llcorner (\Xi f) \).

A proof can be found in the appendix.
4 Locality and Frame Accumulation

The relational structure of modification commands (cf. Theorem 3.2) reveals that they are related to so-called local commands [DM12, HHM+11]. These commands have the following special behaviour: at most resources in the footprint\(^1\) of such a command are modified while all other resources are left unchanged. Relationally, local commands \(P\) are simply characterised by the equation \(P \ast \text{skip} = P\) [DM12]. For modifications we can immediately conclude

**Lemma 4.1** Modifications \(\Delta f\) are local commands.

**Proof.** By Theorem 3.2, associativity of \(\ast\), \(\text{skip} \ast \text{skip} = \text{skip}\),
\[
\Delta f \ast \text{skip} = ((f ; \top) \ast \text{skip}) \ast \text{skip} = (f ; \top) \ast (\text{skip} \ast \text{skip}) = (f ; \top) \ast \text{skip} = \Delta f.
\]

\(\square\)

Basically, pairs of commands within \(\ast\)-compositions operate separately on disjoint portions of states. In [DM12] it turned out that due to the angelic behaviour of relations, an additional assumption is required for pointfree calculations on the footprint and the resources that remain untouched in \(\ast\)-products. The assumption can be encoded relationally by the frame property [DHM11], i.e.,
\[
\langle P \times \text{skip} \rangle ; \triangleright ; P \subseteq (P \times \text{skip}) ; \triangleright . \quad (9)
\]

In pointwise form it reads as follows, considering arbitrary \(\sigma_P, \sigma_{\text{skip}}, \sigma'\) in a pair \(((\sigma_P, \sigma_{\text{skip}}), \sigma')\) of the left-hand side:
\[
\sigma_P \in \lceil P \ast \sigma_{\text{skip}} \rceil \land (\sigma_P \bullet \sigma_{\text{skip}}) P \sigma' \Rightarrow \exists \sigma'_P. \quad \sigma_P P \sigma'_P \land \sigma' = \sigma'_P \bullet \sigma_{\text{skip}}.
\]

This implies that the state portion \(\sigma_{\text{skip}}\) above does not contain any resources that \(P\) would need for a successful execution and hence is not affected by the execution of \(P\). Equation (9) is named after the frame property of separation logic, since it basically reflects similar behaviour [DHM11]. It can be shown that local commands with a precise footprint satisfy this inequation as, e.g., in the case of modifications \(\Delta f\).

**Lemma 4.2** Modifications \(\Delta f\) have the frame property.

A proof can be found in the appendix.

In the present work, Equation (9) will be applied to prove a relational version of the frame accumulation law of [Kas11]. For a better intuition we start by providing the logical version of that law and describe its semantics. It is originally given as an imperative specification, i.e., a Boolean expression that is relationally evaluated on arbitrary pairs \((\sigma, \sigma')\) where \(\sigma\) denotes the initial and \(\sigma'\) the final state of an arbitrary execution. The accumulation law reads as follows
\[
(\Delta f \ast g' \subseteq f \cup \text{unused}) ; \Delta g \Rightarrow \Delta f . \quad (10)
\]

\(^1\) The minimal set of resources required for non-aborting executions.
The relational version of the accumulation law is to be understood pointwise on arbitrary pairs \((\sigma, \sigma')\) by

\[(\exists \sigma'' . \sigma \Delta f \sigma'' \land g(\sigma'') \subseteq f(\sigma) \cup \text{unused}(\sigma) \land \sigma'' \Delta g \sigma') \Rightarrow \sigma \Delta f \sigma'
\]

where \(\sigma\) denotes an initial state and \(\sigma'\) a final state. Note that the dynamic frame \(g'\) of Equation (10) denotes the final value of \(g\) on the intermediate state \(\sigma''\) instead of \(\sigma'\).

The law means that whenever \(g\) in the intermediate state is bounded by \(f\) and can only increase by initially unallocated resources then the overall effect is that at most locations in \(f\) are changed in the composition \(\Delta f \cdot \Delta g\). Or equivalently, all allocated resources initially from \(f\) disjoint are preserved. For an algebraic proof we need a pointfree variant to characterise bounds for dynamic frames within modifications, which is of course not trivial to achieve since dynamic frames are state-dependent.

**Definition 4.3** For dynamic frames \(f, g\) we say that \(g\) is bounded by \(f\) iff

\[
\# ; (f ; T \times \text{skip}) ; \triangleright ; (g ; \text{skip}) \subseteq (f ; (g \times \text{skip}) \times \text{skip}) ; \triangleright .
\]

To understand the intuition of this formula within the dynamic frames theory we describe its meaning in the concrete separation algebra \(\text{DFS}A\). Of course it can be interpreted in other adequate separation algebras, too. Assume an arbitrary pair \(((\sigma_f, \sigma_{\text{skip}}), \sigma')\) from the left-hand side of the above inequation. In a pointwise form the premise then reads

\[
\exists \sigma, \sigma_g, \tau_{\text{skip}} . \sigma_f \in f \land \sigma_f \# \sigma_{\text{skip}} \land \sigma_g \cdot \sigma_{\text{skip}} = \sigma_g \cdot \tau_{\text{skip}} = \sigma' \land \sigma_g \in g .
\]

Intuitively the substate \(\sigma_f\) represents that part of the complete state \(\sigma_f \cdot \sigma_{\text{skip}}\) that can be changed while \(\sigma_{\text{skip}}\) corresponds to the untouched part in which any changes to resources are not permitted. By assuming \(\exists \sigma' . \sigma' = \sigma_f \cdot \sigma_{\text{skip}}\) we also know \(\sigma_f \# \sigma_{\text{skip}}\) and hence \(\sigma_{\text{skip}}\) is also disjoint from any additionally allocated resources, i.e., \(\text{dom}(\sigma_{\text{skip}})\) is disjoint from any locations of \(\text{unused}(\sigma_f \cdot \sigma_{\text{skip}})\).

Now, the right-hand side states that

\[
\exists \sigma_{\text{rem}} . \sigma_f \in f \land \sigma' = (\sigma_g \cdot \sigma_{\text{rem}}) \cdot \sigma_{\text{skip}} \land \sigma_g \in g .
\]

This means by cancellativity of the underlying separation algebra that \(\sigma' = \sigma_g \cdot \sigma_{\text{rem}} \text{ and } \tau_{\text{skip}} = \sigma_{\text{rem}} \cdot \sigma_{\text{skip}}\). Hence, \(\sigma_g \leq \sigma_f\) and \(\sigma_{\text{skip}} \leq \tau_{\text{skip}}\). In particular, we get \(\sigma_g \# \sigma_{\text{skip}}\), i.e., \(\sigma_g\) is disjoint from \(\sigma_{\text{skip}}\) which in turn implies that its allocated locations can only cover locations of \(f\) and initially unallocated ones in \(\text{unused}(\sigma_f \cdot \sigma_{\text{skip}})\). The above state partitions can be depicted as follows:

Conversely, we can show using cross-split and disjointness that the underlying separation algebra satisfies the inequation of Definition 4.3, assuming \(\sigma_g \# \sigma_{\text{skip}}\).
To see this, note that the premise asserts $\sigma \top \cdot \sigma_{\text{skip}} = \sigma_g \cdot \tau_{\text{skip}}$ and hence $\sigma \top \# \sigma_{\text{skip}}$. By cross-split, i.e., Equation (2) we infer

$$\exists \sigma_1, \sigma_2, \sigma_3, \sigma_4. \quad \sigma \top = \sigma_1 \cdot \sigma_2 \land \sigma_{\text{skip}} = \sigma_3 \cdot \sigma_4 \land \sigma_g = \sigma_1 \cdot \sigma_3 \land \tau_{\text{skip}} = \sigma_2 \cdot \sigma_4.$$  

Thus, $\sigma_g \# \sigma_{\text{skip}} \iff \sigma_1 \cdot \sigma_3 \# \sigma_3 \cdot \sigma_4$ and Equation (3) implies that $\sigma_3 = u$. By this we immediately have $\sigma_g = \sigma_1 \land \sigma_{\text{skip}} = \sigma_4$ and therefore $\sigma \top = \sigma_1 \cdot \sigma_2 \land \tau_{\text{skip}} = \sigma_2 \cdot \sigma_{\text{skip}}$. Since $\sigma \top \# \sigma_{\text{skip}}$ we can instantiate $\sigma_{\text{rem}}$ as $\sigma_2$.

Unfortunately, Definition 4.3 is more complex than its logical variant which is due to implicitly expressing the particular restriction of $g$ to unallocated resources w.r.t. $f$. However, with Definition 4.3 we now have the possibility to abstractly relate dynamic frames among each other and can continue by reasoning in an (in)equational style. By this we can summarise a central result of dynamic frames within modifications.

**Theorem 4.4** Assume dynamic frames where $g$ is bounded by $f$ then

$$\Delta f ; \Delta g \sqsubseteq (f ; \top ; \Delta g) \ast \text{skip}.$$ 

**Proof.** By Theorem 3.2, Corollary 2.4 and Lemma 3.6, $g$ is bounded by $f$, $\text{skip} = \text{skip} ; \text{skip}$ and Equation (5), Lemma 3.6, $\Delta g$ has the frame property and Equation (5) again, and definition of $\ast$:

$$\begin{align*}
\Delta f ; \Delta g &= \langle \sigma ; (f ; \top \times \text{skip}) ; \sigma \rangle ; \Delta g \\
&\subseteq \langle \sigma ; \# ; (f ; \top \times \text{skip}) ; \sigma \rangle ; (g \ast \text{skip}) ; \Delta g \\
&\subseteq \langle \sigma ; (f ; \top ; (g \ast \text{skip}) \times \text{skip}) ; \sigma \rangle ; (\Delta g \times \text{skip}) ; \sigma ; \Delta g \\
&\quad \subseteq \langle \sigma ; (f ; \top ; \Delta g \times \text{skip}) ; \sigma \rangle ; \sigma \ast \Delta g \\
&= (f ; \top ; \Delta g) \ast \text{skip}.
\end{align*}$$

This characterises the behaviour that only the changes on the execution within $f$ need to be considered for $\Delta g$ if $g$ is bounded by $f$, while all other allocated locations w.r.t. a starting state will remain unchanged.

**Corollary 4.5** (Frame Accumulation) Assume dynamic frames $f, g$ where $g$ is bounded by $f$. Then

$$\Delta f ; \Delta g \sqsubseteq \Delta f.$$ 

**Proof.** By Theorem 4.4, isotony and definition of $\top$, and Theorem 4.4:

$$\Delta f ; \Delta g \sqsubseteq (f ; \top ; \Delta g) \ast \text{skip} \sqsubseteq (f ; \top) \ast \text{skip} = \Delta f.$$ 

11
This result can be interpreted as a pointfree variant of the frame accumulation theorem of [Kas11] (cf. Equation (10)). Its general application is to simplify correctness proofs of specifications by eliminating occurrences of sequential composition in combination with framing requirements.

In [Kas11] the concept of strong dynamic frames is also defined. Such frames \( f \) come with the additional restriction on a final state \( \sigma' \) that \( f(\sigma') \) can only contain locations of \( f(\sigma) \) for a starting state \( \sigma \) or unallocated ones w.r.t. \( \sigma \). Since the given abstractions of dynamic frames in this work imply that they are always self-framing, the modifications \( \Delta f \) are only able to extend \( f \) in \( \sigma' \) by previously unallocated locations as in [Kas11]. Hence, simple modifications \( \Delta f \) already coincide with the stronger variant within our abstraction.

As a final result we present another treatment of the abstracted theory in the context of related work.

5 A Related Approach: Local Actions

In [COY07] an abstract approach to separation logic was presented that is built on separation algebras and provides a model of programs in terms of so-called local actions. By contrast with the relational approach of Section 2.2 this concept works pointwise. We show in the following by the use of previous ideas about abstracting dynamic frames that formalisations about modifications in that approach satisfy a similar locality condition and allow a calculational proof of the frame accumulation law within the separation algebra \( \text{DFSA} \).

Basically, local actions are special state transformers, i.e., special functions that map from states to sets of states or to a distinguished element \( \top \). The element \( \top \) is used to denote program abortion, e.g., due to dereferencing of non-allocated resources.

There is also an order \( \sqsubseteq \) defined on sets of states and \( \top \). For arbitrary sets of states \( p, q \in \mathcal{P}(\Sigma) \) it is defined by \( p \sqsubseteq q =_{df} p \subseteq q \). Moreover, \( \top \) is the greatest element w.r.t. the order \( \sqsubseteq \), i.e., for arbitrary \( p \in \mathcal{P}(\Sigma) \cup \{\top\} \) we have \( p \sqsubseteq \top \). One can extend \( \sqsubseteq \) pointwise to state transformers \( f, g \) by \( f \sqsubseteq g \iff_{df} \forall \sigma. f(\sigma) \subseteq g(\sigma) \).

Separating conjunction \( * \) on sets of states is given by

\[
p \ast q =_{df} \left\{ \begin{array}{ll}
\{ \sigma_1 \bullet \sigma_2 : \sigma_1 \# \sigma_2, \sigma_1 \in p, \sigma_2 \in q \} & \text{if } p, q \in \mathcal{P}(\Sigma) \\
\top & \text{otherwise.}
\end{array} \right.
\]

A proper definition of \( * \) on strongest postcondition state transformers might lead to problems with associativity. Hence we stay with the definitions of the original approach. A state transformer definition for modifications can be obtained for a fixed set of locations \( f \) with the same ideas as in Section 3 by

\[
(\Delta f)(\sigma) =_{df} \left\{ \begin{array}{ll}
\Sigma \ast \{ \sigma \}_{\text{used} - f} & \text{if } f \subseteq \text{used}(\sigma) \\
\top & \text{otherwise.}
\end{array} \right.
\]

\(\top\) does not denote the universal relation in this context.
Intuitively, whenever all locations of \( f \) are allocated then all other used locations in \( \sigma \) are preserved. Otherwise, an erroneous execution is signalled by the output \( \top \). Analogously, in the case of \( \Xi f \) we can define

\[
(\Xi f)(\sigma) =_{df} \begin{cases}
\Sigma^* \{ \sigma_f \} & \text{if } f \subseteq \text{used}(\sigma) \\
\top & \text{otherwise}.
\end{cases}
\]

According to Lemma 4.2, the relational version of \( \Delta f \) satisfies the frame property, i.e., Equation (9). Similar behaviour is obtained for the state transformer definition of \( \Delta f \) by the locality property of [COY07], i.e.,

\[
\sigma_1 \# \sigma_2 \Rightarrow (\Delta f)(\sigma_1 \bullet \sigma_2) \subseteq (\Delta f)(\sigma_1) \ast \{ \sigma_2 \}.
\] (11)

State transformers that satisfy Equation (11) are called local actions. The locality property has similar behaviour as the relational version of the frame property. The state \( \sigma_2 \) represents that part of the state \( \sigma_1 \bullet \sigma_2 \) that will remain unchanged while \( \sigma_1 \) contains the footprint of \( \Delta f \).

For a proof of Equation (11) a case distinction is needed. First assume \( \sigma_1 \# \sigma_2 \). If \( f \nsubseteq \text{used}(\sigma_1) \) then

\[
(\Delta f)(\sigma_1) \ast \{ \sigma_2 \} = \top \ast \{ \sigma_2 \} = \top
\]
otherwise.

Next we show that a treatment of the frame accumulation law is also possible using local actions. For a translation of the frame accumulation law into that setting we need to define a local action that models the restricted modification given in its logical variant (cf. Equation (10))

\[
\Delta(f,g) =_{df} \Delta f \land g' \subseteq f \cup \text{unused}(\sigma).
\]

Note that \( g' = g(\sigma') \) generally implies the existence of a set of locations \( g \) in each state \( \sigma' \) in the result set \( (\Delta f)(\sigma) \), interpreting modifications as a local action. By this we need to restrict the local action definition of modification \( \Delta f \) as follows to get a local action for \( \Delta(f,g) \)

\[
(\Delta(f,g))(\sigma) =_{df} \begin{cases}
\{ \sigma' : \text{used}(\sigma') = g \} \ast \Sigma^* \{ \sigma |_{\text{used} - f} \} & \text{if } f \subseteq \text{used}(\sigma) \\
\top & \text{otherwise}.
\end{cases}
\]

The general idea with this is to restrict the output of \( \Delta f \) to involve a fixed set of locations \( g \). Another possibility would be to define another local action that sequentially composed with \( \Delta f \) restricts its output adequately. The above local action for \( \Delta(f,g) \) includes the behaviour described in Definition 4.3 in which a bounding between dynamic frames \( g \) and \( f \) is characterised. Analogously to \( \Delta f \), the state transformer is also a local action. Now, the frame accumulation law in that setting can be stated as follows

\[
\forall \sigma. \ (\Delta(f,g) ; \Delta g)(\sigma) \subseteq \Delta f(\sigma),
\]
where for arbitrary local actions \(f, g\) one pointwise lifts \((f; g)(\sigma) =_{df} \bigcup \{ g(\sigma') : \sigma' \in f(\sigma) \}\) if \(f(\sigma) \neq \top\) and otherwise \(f ; g\) also equals \(\top\). For a proof of the above inequation we assume \(f \subseteq \text{used}(\sigma)\) and \(g \subseteq f \cup \text{unused}(\sigma)\) and calculate

\[
(\Delta(f, g) ; \Delta g)(\sigma) = \bigcup \{ \Delta g(\sigma') : \sigma' \in \{ \sigma' : \text{used}(\sigma') = g \} \} \\
= \bigcup \{ \Delta g(\sigma' \cdot \tau \cdot \sigma|_{\text{used}-f}) : \text{used}(\sigma') = g, \tau \in \Sigma \} \\
\subseteq \bigcup \{ \Delta g(\sigma') \ast \{ \tau \cdot \sigma|_{\text{used}-f} \} : \text{used}(\sigma') = g, \tau \in \Sigma \} \\
= \bigcup \{ \Sigma \ast \{ \tau \cdot \sigma|_{\text{used}-f} \} : \tau \in \Sigma \} \\
\subseteq \bigcup \{ \Sigma \ast \{ \sigma|_{\text{used}-f} \} \} \\
= \Sigma \ast \{ \sigma|_{\text{used}-f} \} \\
= \Delta f(\sigma).
\]

6 Conclusion and Outlook

We explored algebraic and abstract calculi for the theory of dynamic frames. It turned out that an extended relational approach, originally used as an algebraic base for separation logic, can also be used to generally formalise effects of the dynamic frames theory. Since definitions in that theory were given in [Kas11] in a relational style, a direct translation to relational pointfree variants was possible by a few abstractions. This yields a step towards a unifying calculus for abstractly capturing crucial behaviours of dynamic frames and separation logic.

As further work it would be interesting to include the overlapping conjunction of [HV13] into this setting. Applied to assertions it allows an unspecified portion of resources to be shared among two predicates. For the presented calculus, it would enable an abstract treatment of dynamic frames that share certain parts of their locations as e.g., in the situation when two iterators are attached to the same list as described in [Kas11]. Another possibility for this can be considering separation algebras that involve permissions [BCOP05].

Moreover, the relationships to concrete approaches [DYDG+10, PS11, JB12] and their integration into this framework has to be investigated.

Acknowledgements: I am grateful to Eric C. R. Hehner and Bernhard Möller for drawing my attention to this particular topic. Moreover, I thank Bernhard Möller for valuable remarks and all anonymous reviewers for their helpful feedback and comments that significantly helped to improve the paper. This research was partially funded by the DFG project MO 690/9-1 ALGSEP — Algebraic Calculi for Separation Logic.

References


Appendix: Deferred Proofs

Proof of Lemma 3.6.

By Theorem 3.2 and def. of $\ast$, $f$ is a test, $\text{skip} = \text{skip} ; \text{skip}$ and Equation (5), Corollary 2.4 and Corollary 2.5, again Corollary 2.4, Theorem 3.2, def. of $\ast$, $\Delta f = \langle f ; (f \times \text{skip}) ; \triangleright \rangle$

Proof of Lemma 4.2.