Variable Side Conditions and Greatest Relations in Algebraic Separation Logic

Han-Hing Dang and Peter Höfner

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

Abstract. When reasoning within separation logic, it is often necessary to provide side conditions for inference rules. These side conditions usually contain information about variables and their use, and are given within a meta-language, i.e., the side conditions cannot be encoded in separation logic itself. In this paper we discuss different possibilities how side conditions of variables—occurring e.g. in the ordinary or the hypothetical frame rule—can be characterised using algebraic separation logic. We also study greatest relations; a concept used in the soundness proof of the hypothetical frame rule. We provide one and only one level of abstraction for the logic, the side conditions and the greatest relations.

1 Introduction

Over the last years, separation logic (SL) (e.g. [11]) has been established as a formal system that allows reasoning and verification of imperative programs *including* shared mutable data structures. It is a proper extension of Hoare logic and has been used (a) to split data structures into logically connected regions which can then be analysed and reasoned about separately; (b) for the analysis of pointer variables and their update; and (c) for the dynamic assignment of "owners" of data regions under concurrent access to them.

One major instrument of SL is the frame rule. It allows adding arbitrary disjoint storage to the resources which are actually used by a command. Proof rules like the frame rule are often constrained by side conditions on the variables involved. Usually, they are formulated within a meta-language. This complicates reasoning in general and, more particularly, when building tools for automated reasoning (e.g. Smallfoot [1]) two layers have to be considered.

In a companion paper [5], an algebra for separation logic based on a relational semantics of commands has been presented. On this basis a restricted version of the frame rule has been proved in an abstract way. The proof itself is based on three assumptions: safety monotonicity, a frame property and preservation. The first two were also used by Reynolds [11], the third one was intended as an algebraic counterpart of the side condition of the frame rule.

In this paper we show that the concept of preservation as presented in [5] is too strict. Motivated by this observation, we discuss to which extent variable side conditions can be embedded into the algebraic framework. As a result we

characterise various side conditions at an algebraic level and provide one and only one level of abstraction for both the logic and the side conditions. Moreover we bring the hypothetical frame rule into our setting. This rule allows more general reasoning than the original one. As a further application for algebraic separation logic we give pointfree characterisations for greatest relations which play an important role in proving soundness of this particular frame rule.

2 The Frame Rule and the Set of Modified Variables

The frame rule [8] describes that a command can also be executed using a larger storage—as long as the command does not influence the additional storage:

$$\frac{\{p\}C\{q\}}{\{p * r\}C\{q * r\}} \quad MV(C) \cap FV(r) = \emptyset \ . \tag{1}$$

The expression $\{p\}\, C\, \{q\}$ denotes a slightly modified Hoare triple in partial correctness semantics where p and q are predicates about states and C is a command: as usual, the command C establishes the postcondition if the precondition is met. Additionally, the command C can always be executed whenever p is satisfied. The disjoint storage part, characterised by r, will remain unchanged as long as no free variable of r will be touched by any execution of C. This restriction on the usage of variables is described by the formula $MV(C) \cap FV(r) = \emptyset$.

Next to the side condition, two further assumptions on commands C are needed to prove the frame rule in SL: *safety monotonicity* and the *frame property*. The former guarantees that if C is executable from a state, it can also run on a state with a larger heap; the latter states that every execution of C can be tracked back to an execution of C running on states with a possible smaller heap.

Let us now take a closer look at the side condition of the frame rule and on the set $MV(C)$ of variables modified by a command C. Formally, the syntax of a command is given by[1]

$$\begin{aligned} exp &::= var \mid seq.var \mid \mathsf{tail}(seq) \mid \mathsf{head}(seq) \mid \ldots \\ comm &::= var := exp \mid \mathsf{dispose}\ exp \\ &\quad \mid \mathsf{skip} \mid comm\ ;\ comm \mid \mathsf{if}\ bexp\ \mathsf{then}\ comm\ \mathsf{else}\ comm \\ &\quad \mid var := \mathsf{cons}\,(exp, \ldots, exp) \mid var := [exp] \mid [exp] := exp\ , \end{aligned}$$

where *var* denotes variables, *exp* expressions and *bexp* boolean expressions. Moreover *seq* stands for sequences of values, . denotes concatenation and head, tail return the head or tail of a sequence resp. The command $v := \mathsf{cons}\,(e_1, ..., e_n)$ allocates n contiguous fresh cells and places the values of the expressions e_i in the current store as the contents of the i-th cell. The address of the first cell is stored in v while the following cells can be accessed via address arithmetic. The assignment $v := [e]$ dereferences the heap cell at the address given by the value of e and stores its value in v. An execution of $[e_1] := e_2$ assigns the value of e_2

[1] We provide more details on the commands used later on.

to a cell located at the value of e_1. Finally, the command $\mathsf{dispose}\, e$ deallocates the heap cell located at the address which is the value of e.

Based on that, the set $MV(C)$ of modified variables (not heap cells!) of a command C can be determined inductively by the rules given in Table 1.

C	$MV(C)$
$[e_1] := e_2$	$\emptyset$
$v := [e]$	$\{v\}$
$\mathsf{dispose}\ e$	$\emptyset$
$v := \mathsf{cons}\,(e_1, ..., e_n)$	$\{v\}$
$C_1\,;C_2$	$MV(C_1) \cup MV(C_2)$
$\mathsf{if}\ (b)\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2$	$MV(C_1) \cup MV(C_2)$

Table 1. $MV(C)$-set for commands C.

These definitions seem straightforward; however, they depend in an essential way on the syntax and structure of a command and not on its semantics.

For the following commands we assume that two variables x, y are available.

$$C_1 =_{df} (x := y) \qquad \text{and} \qquad C_2 =_{df} (x := y\,;\, y := 3\,;\, y := x)\ .$$

After execution, both commands C_1 and C_2 have set the variable x to y and the value of y is the same as it was before the execution. However, C_2 modifies y during the execution. Hence $MV(C_1) = \{x\}$ and $MV(C_2) = \{x, y\}$. To connect commands with relations, the algebraic approach of [6] describes each command by an input-output relation between states, or, in other words, by a state transformer. (The details will be explained in the following sections). These relations only reflect the overall behaviour, hence cannot look at the syntactic structure of a given command. In particular, the commands C_1 and C_2 are indistinguishable for the algebraic approach. Usually, the set MV of modified variables lists all variables to which values are assigned. However, it would be interesting to determine the set of all variables which are "really changed" by a command as a relation. For the commands C_1 and C_2 this would be the set $\{x\}$.

3 Algebraic Separation Logic

Before looking at side conditions algebraically, we have to recapitulate the foundations of algebraic separation logic and its relational semantics.

A system's *state* is a pair consisting of a store and a heap; stores and heaps are partial functions from variables or addresses to values. To simplify the formal treatment, values and addresses are assumed to be integers.

$$\begin{aligned} \mathit{Values} &= \mathbb{Z}\ , & \mathit{Stores} &= \mathit{Vars} \rightsquigarrow \mathit{Values}\ , \\ \{\mathsf{nil}\} \uplus \mathit{Addresses} &\subseteq \mathit{Values}\ , & \mathit{Heaps} &= \mathit{Addresses} \rightsquigarrow \mathit{Values}\ , \\ & & \mathit{States} &= \mathit{Stores} \times \mathit{Heaps}\ , \end{aligned}$$

where *Vars* is the set of program variables, $\uplus$ denotes the disjoint union of sets and $M \rightsquigarrow N$ denotes the set of partial functions between M and N. Stores and heaps will be denoted by s and h, resp., while σ and τ stand for states. The

store and heap part of a state σ are denoted by s_σ and h_σ, resp. As usual we define a domain operator dom and a range operator cod on relations and partial functions. For example, $\mathsf{dom}(s)$ for a store s returns the set of all variables with defined values; for a relation R a partial identity relation is returned. The constant nil denotes an improper reference and therefore heaps must not map nil to any value.

We follow the idea of [4, 6] and define based on states, a *command* as a relation $C \in \mathit{Cmds} =_{df} \mathcal{P}(\mathit{States} \times \mathit{States})$. Relations offer a number of operations, including sequential composition ; , choice $\cup$, converse $^\smile$ and complementation $\overline{\ }$. In general, relations and the structure $(\mathit{Cmds}, \subseteq, ;, I)$ in particular form Boolean quantales [2], where I is the identity relation. For the purpose of the paper, we restrict ourselves to relations; although most of the results could be also achieved in the more general quantale setting.

Next we recapitulate the relational semantics for the above mentioned concrete commands. For that we define $FV(e)$ as the set of all free variables occurring in an expression e. The value e^s of an expression is defined for an arbitrary store s only if $FV(e) \subseteq \mathsf{dom}(s)$. Moreover, we define an update operator on partial functions by $f \,|\, f' =_{df} f \cup \{(v,c) : (v,c) \in f' \wedge v \notin \mathsf{dom}(f)\}$ and abbreviate $\{(v,c)\} \,|\, f$ to $(v,c) \,|\, f$. We characterise the commands linking input states (s,h) and output states (s',h') by $R \mathrel{\widehat{=}} P$ to abbreviate the clause $(s,h)\, R\, (s',h') \Leftrightarrow_{df} P$. We require for each of the following commands C and expressions e occurring in C that $FV(e) \subseteq \mathsf{dom}(s)$.

$$
\begin{aligned}
[\![[e_1] := e_2]\!]_c &\mathrel{\widehat{=}} s' = s \wedge e_1^s \in \mathsf{dom}(h) \wedge h' = (e_1^s, e_2^s) \,|\, h \;, \\
[\![v := [e]]\!]_c &\mathrel{\widehat{=}} s' = (v, h(e^s)) \,|\, s \;\wedge e^s \in \mathsf{dom}(h) \wedge\; h' = h \;, \\
[\![\,\mathsf{dispose}\ e]\!]_c &\mathrel{\widehat{=}} s' = s \wedge e^s \in \mathsf{dom}(h) \wedge h' = h - \{(e^s, h(e^s))\} \;, \\
[\![v := \mathsf{cons}\,(e_1, ..., e_n)]\!]_c &\mathrel{\widehat{=}} \exists\, a \in \mathit{Addresses}\,.\; s' = (v,a) \,|\, s \;\wedge \\
&\qquad\qquad a, \ldots, a+n-1 \notin \mathsf{dom}(h) \wedge \\
&\qquad\qquad h' = \{(a, e_1^s), \ldots, (a+n-1, e_n^s)\} \,|\, h \;.
\end{aligned}
$$

If a command is executable on a state, we assume that it can also run on larger states containing more variable declarations. Moreover, we assume that the command does not change the set of defined program variables. We assume $C \in \mathit{Cmds}.\ \sigma\, C\, \tau \Rightarrow \mathsf{dom}(s_\sigma) = \mathsf{dom}(s_\tau)$ and for all $X \subseteq \mathit{Vars}.\ \sigma \in \mathsf{dom}(C) \Rightarrow \exists\, \tau.\ \tau \in \mathsf{dom}(C) \wedge \mathsf{dom}(s_\tau) = \mathsf{dom}(s_\sigma) \cup X$. The semantics of an if–statement is defined as usual. Due to readability we will omit brackets $[\![\ \ ]\!]_c$ and have each statement stand for its semantics.

This forms the basis of algebraic separation logic—except for separating conjunction $*$ which is described in the next section.

4 States: Compatibility and Splitting

The separating conjunction $*$ of SL unites disjoint heap regions and allows reasoning about separate storage. The algebraic approach is more general and lifts this operation to general commands: a command is split into executions run-

ning on disjoint heap parts. By splitting we describe a generalised version of the separating conjunction $*$.

- Two stores s and s' are *compatible* iff $s = s' \vee \mathsf{dom}(s) \cap \mathsf{dom}(s') = \emptyset$.
- Two states $\sigma_1 = (s_1, h_1)$ and $\sigma_2 = (s_2, h_2)$ are *combinable* iff
  $(s_1, h_1) \# (s_2, h_2) \Leftrightarrow_{df} s_1, s_2$ are *compatible* $\wedge\ \mathsf{dom}(h_1) \cap \mathsf{dom}(h_2) = \emptyset$.
- The *split* relation $\lhd$ is defined for states σ, σ_1 and σ_2 as
  $$\sigma \lhd (\sigma_1, \sigma_2) \Leftrightarrow_{df} \sigma_1 \# \sigma_2 \wedge \sigma = \sigma_1 * \sigma_2 \ ,$$
  where $\sigma_1 * \sigma_2 = (s_1 \cup s_2, h_1 \cup h_2)$ if $\sigma_1 = (s_1, h_1)$ and $\sigma_2 = (s_2, h_2)$.
- The *join* relation $\rhd$ is the converse of $\lhd$, i.e., $\rhd = \lhd^{\smile}$.
- The *Cartesian product* $C_1 \times C_2$ of two commands C_1, C_2 is given, as usual, by
  $$(\sigma_1, \sigma_2)\,(C_1 \times C_2)\,(\tau_1, \tau_2) \Leftrightarrow_{df} \sigma_1\, C_1\, \tau_1 \wedge \sigma_2\, C_2\, \tau_2 \ .$$
  It is well known that $\times$ and ; satisfy the exchange property
  $$(R_1 \times R_2)\,;(S_1 \times S_2) \;=\; (R_1\,;S_1) \times (R_2\,;S_2) \ . \tag{2}$$
  We assume for the rest of this paper that ; binds tighter than $\times$.
- The $*$ *composition* is defined by $C_1 * C_2 =_{df} \lhd\,;(C_1 \times C_2)\,;\rhd$.

By store compatibility it is required that both involved stores are either equal or both map from a disjoint set of variables. Therefore when joining and splitting states we are more liberal as in standard separation logic. The standard separating conjunction requires the stores involved to be equal.

For later properties we need to define a relation w.r.t. a command C that only allows store changes of C and excludes heap alteration. For states σ, σ', the *store-change* relation S_C for a command C is defined by

$$\begin{aligned}\sigma\ S_C\ \sigma' \Leftrightarrow_{df}\ & h_\sigma = h_{\sigma'} \wedge \mathit{changed}(C) \subseteq \mathsf{dom}(s_\sigma) \wedge \\ & (\exists\, \sigma_c, \sigma'_c \,.\, \sigma_c\ C\ \sigma'_c \wedge s_{\sigma_c} \subseteq s_\sigma \wedge s_{\sigma'_c} \subseteq s_{\sigma'} \wedge \\ & s_\sigma - s_{\sigma_c} = s_{\sigma'} - s_{\sigma'_c}) \ ,\end{aligned}$$

where $\mathit{changed}(C) =_{df} \bigcup_{(\tau_1,\tau_2)\in C} \mathsf{dom}(s_{\tau_1} - s_{\tau_2})$. This latter definition is motivated by the fact that $x \in \mathsf{dom}(s_{\tau_1} - s_{\tau_2}) \Leftrightarrow x \in \mathsf{dom}(s_{\tau_1}) \wedge s_{\tau_1}(x) \neq s_{\tau_2}(x)$. Given a command C the relation S_C changes each store variable of an input state as C would do. The first line of the definition ensures that all stores involved mention at least all variables that are changed by an arbitrary execution of C. This is necessary to ensure certain preservation properties given later. Next, we briefly sum up some results for the relation S_C needed later on.

Lemma 4.1 *For an arbitrary test r and commands C, D we have*

1. $S_r \subseteq I$,
2. $C * S_C \subseteq C * I$. *In particular,* $C * (\mathsf{emp}\,;\, S_C\,) \subseteq C$ *where* $\sigma\ \mathsf{emp}\ \sigma' \Leftrightarrow_{df} s_\sigma = s'_\sigma \wedge h_\sigma = \emptyset = h_{\sigma'}$,
3. *If* $C \subseteq D$ *then* $C * (r\,;\, S_D\,) \subseteq C * (r\,;\, S_C\,)$.
4. *If* $r; S_{C;D} \subseteq r; S_C\,; r; S_D$ *then* $(C;D)*(r; S_{C;D}\,) \subseteq (C*(r; S_C\,));(D*(r; S_D\,))$

Proof. (1) follows immediate from the definition. For (2) consider $\sigma\ (C * S_C\ )\ \sigma'$. Then by definition there exist states $\sigma_c, \sigma'_c, \sigma_S, \sigma'_S$ with $\sigma = \sigma_c * \sigma_S \wedge \sigma_c \# \sigma_S \wedge \sigma' = \sigma'_c * \sigma'_S \wedge \sigma'_c \# \sigma'_S \wedge \sigma_c\ C\ \sigma'_c \wedge \sigma_S\ S_C\ \sigma'_S$. By the definition of S_C this implies $s_{\sigma_c} = s_{\sigma_S}$ and $s_{\sigma'_c} = s_{\sigma'_S}$. Now set $\sigma_I =_{df} (\emptyset, h_{\sigma_S})$, then $\sigma = \sigma_c * \sigma_I \wedge \sigma_c \# \sigma_I \wedge \sigma_c\ C\ \sigma'_c \wedge \sigma_I \in I \wedge \sigma'_c \# \sigma_I \wedge \sigma' = \sigma'_c * \sigma_I$ holds.

Furthermore assume $\sigma\ C * (r\ ;\ S_D\ )\ \sigma'$. Again there exist $\sigma_c, \sigma'_c, \sigma_r, \sigma_S$ with $\sigma = \sigma_c * \sigma_r \wedge \sigma_c \# \sigma_r \wedge \sigma_r \in r \wedge \sigma_r\ S_D\ \sigma_S \wedge \sigma_c\ C\ \sigma'_c \wedge \sigma' = \sigma'_c * \sigma_S \wedge \sigma'_c \# \sigma_S$ which implies $s_{\sigma_c} = s_{\sigma_r}$ and $s_{\sigma'_c} = s_{\sigma_S}$. Therefore, by $C \subseteq D$ and the definition of S_C also $\sigma_r\ S_C\ \sigma_S$ holds. Finally to prove (4), we calculate

$$
\begin{array}{ll}
 & (C\ ;D) * (r\ ;\ S_{C;D}\ ) = \lhd\ ;((C\ ;D) \times (r\ ;\ S_{C;D}\ ))\ ;\rhd \\
\subseteq & \lhd\ ;((C\ ;D) \times (r\ ;\ S_C\ \ ;r\ ;\ S_D\ ))\ ;\rhd \\
= & \lhd\ ;((C \times (r\ ;\ S_C\ ))\ ;(D \times (r\ ;\ S_D\ )))\ ;\rhd \\
\subseteq & \lhd\ ;((C \times (r\ ;\ S_C\ ))\ ;\rhd\ ;\lhd\ ;(D \times (r\ ;\ S_D\ ))\ ;\rhd \\
= & (C * (r\ ;\ S_C\ ))\ ;(D * (r\ ;\ S_D\ ))\ .
\end{array}
$$

This uses the definition of $*$, assumption, associativity, Exchange (2), neutrality of $(I \times I)$ and $(I \times I) \subseteq \rhd\ ;\lhd$. □

Informally, considering any execution of C then by $C * S_C$ only disjoint heap cells are added to the states involve, i.e., S_C changes exactly the same variables as C does. In particular in $C * (r;\ S_D\ )$, S_D alters the same variables of r as C does if $C \subseteq D$. Finally, the assumption $r\ ;\ S_{C;D} \subseteq r\ ;\ S_C\ \ ;r\ ;\ S_D$ states that r is not changed after all variable assignments of C.

5 The Frame Rule Algebraically

Besides commands the frame rule uses slightly modified Hoare triples $\{p\}\,C\,\{q\}$ (see Section 2). In the algebraic setting predicates (pre- and postconditions) can be modelled by tests as e.g. in [7]. In *Cmds*, tests are given by partial identity relations of the form $\{(\sigma,\sigma) \mid \sigma \in p\}$ for some set $p \in \mathit{States}$. We further abbreviate $(\sigma,\sigma) \in p$ to $\sigma \in p$. Using tests, an if–statement $\mathsf{if}\ p\ \mathsf{then}\ C\ \mathsf{else}\ C'$ is described by $p\,;C \cup \neg p\,;C'$, where $\neg p =_{df} \overline{p} \cap I$. It has further been shown that a (standard) Hoare triple $\{p\}\,C\,\{q\}$ is equivalent to $p\,;C \subseteq C\,;q$, where p, q are test elements. It expresses that q is reached under all C-transitions from p, i.e., the precondition p guarantees the postcondition q. The modified Hoare triples are characterised by

$$\{p\}C\{q\} \Leftrightarrow_{df} p \subseteq \mathsf{dom}(C) \wedge p\,;C \subseteq C\,;q\ . \tag{3}$$

Note that $\mathsf{dom}(C)$ for a relation C denotes the corresponding partial identity relation. This means that $\mathsf{dom}(C)$ is a subidentity of states. Informally, $p \subseteq \mathsf{dom}(C)$ states that C can be executed in all states satisfying p. By these definitions, the frame rule turns into the implication

$$p\,;C \subseteq C\,;q \Rightarrow (p * r)\,;C \subseteq C\,;(q * r)\ . \tag{4}$$

The frame rule as well as its algebraic counterpart (4) do not hold in general. As mentioned before, three assumptions are made to prove the frame rule: safety monotonicity, the frame property and the side condition.

Formally, a command C is *safety monotone* iff for all $\sigma, \sigma' \in \mathit{States}.\ \sigma \# \sigma' \wedge \sigma \in \mathsf{dom}(C) \Rightarrow \sigma * \sigma' \in \mathsf{dom}(C)$; a command C satisfies the *frame property* iff for all $\sigma, \sigma_{S_C}, \sigma_1 \in \mathit{States}.\ \sigma \in \mathsf{dom}(C) \wedge \sigma_{S_C} \in \mathsf{dom}(S_C) \wedge \sigma \# \sigma_{S_C} \wedge (\sigma * \sigma_I)\ C\ \sigma_1 \Rightarrow \exists \sigma_2, \sigma_S.\ \sigma\ C\ \sigma_2 \wedge \sigma_{S_C}\ S_C\ \sigma_S \wedge \sigma_2 \# \sigma_S \wedge \sigma_1 = \sigma_2 * \sigma_S$ [6]. These definitions can be given pointfree and purely algebraically:

- C is *safety-monotonic* iff $\mathsf{dom}(C) * I \subseteq \mathsf{dom}(C)$;
- C has the *frame property* iff $(\mathsf{dom}(C) \times \mathsf{dom}(S_C)) ; \rhd ; C \subseteq (C \times S_C) ; \rhd$.

Equivalence proofs[2] can be found in [6]. In the next section, we will have a closer look on the third assumption, namely the side condition.

6 Variable Preservation and Variable Side Conditions

Side conditions are often used to restrict the behaviour of commands. In the frame rule, $MV(C) \cap FV(r) = \emptyset$ guarantees that r still holds in the postcondition. In other words, C preserves r.

A first attempt to tackle this variable side conditions of the frame rule algebraically is given in [5]:

$$\lhd ; (C \times r) \subseteq C ; \lhd . \tag{5}$$

This equation states that whenever a state σ can be split into two states σ_c and σ_r such that C can be executed on σ_c, and σ_r satisfies r then each execution of C ends up in a state where σ_r can be retained completely unchanged. Using this, it is possible to prove the frame rule (4).

Unfortunately, Equation (5) is very restrictive and makes the algebraic approach incomplete. This means that not all instances satisfying the original frame rule satisfy the additional assumption (5). For example taking $C = (x := 1)$ and $r = (\mathsf{true})$ yields $\lhd ; (C \times r)$, which is not contained in $C ; \lhd$. In particular, consider a pair $(\sigma, (\tau_1, \tau_2)) \in \lhd ; (C \times r)$. We can assume that $s_{\tau_2}(x) = 2$. But there exists no such splitting in $C ; \lhd$ since each τ_i has to satisfy $s_{\tau_i}(x) = 1$. This means the domain as well as the image of C need to be checked for combinability.

This yields another version of preservation. For arbitrary states $\sigma_1, \sigma_2, \tau_1$ and τ_2, the *combinability (joinability) relation* $\between$ on pairs of states is defined by

$$(\sigma_1, \sigma_2) \between (\tau_1, \tau_2) \Leftrightarrow_{df} \sigma_1 \# \sigma_2 \wedge \sigma_1 = \tau_1 \wedge \sigma_2 = \tau_2 .$$

The relation $\between$ is a test (a partial identity relation) that characterises those pairs of states that are combinable w.r.t. $\#$. We will use this relation to obtain the subcommand of a command that maintains combinability with a test r.

Using combinability, we can now define another version of side conditions. A command C *weakly preserves* r iff

$$\between ; (C \times r) ; \between \neq \emptyset . \tag{6}$$

[2] In this paper we deviate slightly from [6]. But the proofs can be adapted.

Pointwise this means that $\exists\, \sigma_1, \sigma_2, \sigma_r.\ \sigma_1 \# \sigma_r \wedge \sigma_1\, C\, \sigma_2 \wedge \sigma_r \in r \wedge \sigma_2 \# \sigma_r$. Informally, there is at least one execution of C such that an input state σ_1 as well as an output state σ_2 are combinable with a state σ_r. More precisely the relation $\between\, ;(C \times r)\, ;\between$ is removing all executions in C that do not maintain r. If the set is empty then r will definitely be changed by C. Simple consequences of the definition are that if C_1 and C_2 preserve r then $C_1 \cup C_2$ preserves r as well. Moreover, I and emp preserve r provided $r \neq \emptyset$.

In some sense, Equation (6) behaves really angelically: for an assertion r it searches for one "execution-path" in C that preserves r. Let us give an example: with $C =_{df}\ \mathsf{if}\,(x = 0)\,\mathsf{then\ skip\ else}\, x := 2$ and $r =_{df}\ (x \neq 2)$ we have $\between\, ;(C \times r)\, ;\between\ = \between\, ;((x = 0) \times r)\, ;\between\ \neq \emptyset$ since $\between\, ;((x \neq 0\, ;\, x := 2) \times r)\, ;\between\ = \emptyset$.

Therefore this approach is not strong enough to capture the side conditions of the frame rule. Another disadvantage of Equation (6) is that algebraically inequalities cannot easily be used for equational (automated) reasoning.

Moreover, weak preservation is not closed under composition. Consider commands $C_1 =_{df}\ (x := 1), C_2 =_{df}\ (y := 2)$ and an assertion $r =_{df}\ (x = 3 \vee y = 4)$. Then $\between\, ;(C_i \times r)\, ;\between\ \neq \emptyset$ but $\between\, ;(C_1\, ;\, C_2 \times r)\, ;\between\ = \emptyset$. It is not possible to force weak preservation to be closed under composition. This is based on the fact that relations cannot distinguish commands like $y := x$ from $y := x\, ;\, x = 0\, ;\, x := y$ (cf. Section 2). By equivalence of these commands this would imply that $y := x$ does not preserve $r =_{df}\ (x = 1)$.

To give a more appropriate definition we look at downward closure: a command C *downward-preserves* a test r iff $r \neq \emptyset$ and

$$\forall\, C' \subseteq C\ :\ \between\, ;(C' \times r)\, ;\between\ = \emptyset \Rightarrow C' = \emptyset\ . \tag{7}$$

Obviously, every command $C \neq \emptyset$ that downward-preserves r also weakly preserves r. But this definition is much more restrictive. Informally, the definition states that C preserves r only if all possible "executions of C" already preserve r. By this side condition, the above problem of the if-statement can be avoided: the problem of weak preservation is that if there is a choice between two execution paths, it suffices if one of these preserves r. If there is a choice of execution paths "inside" a command C, it can be split into two subcommands C_1 and C_2 with $C = C_1 \cup C_2$. Both commands have now to preserve r, i.e., $\between\, ;(C_i \times r)\, ;\between\ \neq \emptyset$ $(i \in \{1, 2\})$. Moreover the definition is closed under $\cup$, but still not under $;$.

Now, we present a fourth possibility for algebraic side conditions. It is discussed in more detail since we will apply this condition in a small case study.

A command C *preserves* a test r iff

$$C * (r\ ;\ S_C\ ) \subseteq C * r\ . \tag{8}$$

Pointwise this spells out for all $\sigma, \sigma' \in \mathit{States}$:

$$\exists\, \sigma_c, \sigma_r, \sigma_c', \sigma_S.\ \sigma = \sigma_c * \sigma_r \wedge \sigma_c\ C\ \sigma_c' \wedge \sigma_r \in r \wedge \sigma_r\, S_C\ \sigma_S \wedge \sigma' = \sigma_c' * \sigma_S$$
$$\Rightarrow\ \exists\, \sigma_c, \sigma_r, \sigma_c'.\ \sigma = \sigma_c * \sigma_r \wedge \sigma_c\ C\ \sigma_c' \wedge \sigma_r \in r \wedge \sigma' = \sigma_c' * \sigma_r$$

assuming $\sigma_c \# \sigma_r$, $\sigma_c' \# \sigma_S$ and $\sigma_c' \# \sigma_r$. Informally, this inequation characterises commands that only modify variables which do not influence r.

Lemma 6.1 *For arbitrary commands C_1, C_2 and test r:*

- *If C_1, C_2 preserve r then $C_1 \cup C_2$ preserves r.*
- *Assume C, D preserves r. If $r\,;\, S_{C;D} \subseteq r\,;\, S_C\,;r\,;\, S_D$ and $(C * r)\,;(D * r) \subseteq (C\,;\, D) * r$ then $C\,;\, D$ preserves r.*
- *I and emp preserve r, i.e., $I * (r\,;\, S_I) \subseteq I * r$ and $\mathsf{emp} * (r\,;\, S_{\mathsf{emp}}) \subseteq r$.*
- *C preserves I and emp.*

The proof is by straightforward calculations and by Lemma 4.1.

Using this definition of preservation (together with safety monotonicity and the frame property), it is again possible to prove the frame rule purely algebraically. Therefore the algebraic approach is still sound, but maybe still not complete. However, these new definitions make the algebraic frame rule more widely applicable and the restrictions are far smaller than before.

We use these results to model more complex side conditions of SL in the following section. In particular, we look at a variant of the frame rule and describe to which extent its side conditions can be included in the relational framework.

7 Variable Conditions in Information Hiding

In this section we present an approach to include more complex variable preservation conditions into the relational setting. For such side conditions we consider the *hypothetical frame rule* introduced in [9]. It uses the concept of information hiding. We give only some key concepts of that rule and present how reasoning with this proof rule can be captured by our relational approach. For more details concerning the frame rule we refer to [9].

We only treat a special case of the hypothetical frame rule. The ideas given can be easily generalised. The inference rule reads

$$\frac{\{p_1\}k_1\{q_1\}[X_1],\ \{p_2\}k_2\{q_2\}[X_2]\ \vdash\ \{p\}C\{q\}}{\{p_1 * r\}k_1\{q_1 * r\}[X_1, Y],\ \{p_2 * r\}k_2\{q_2 * r\}[X_2, Y]\ \vdash\ \{p * r\}C\{q * r\}}\ ,$$

where the side conditions are skipped for the moment. They will be given below. The semantics of $\vdash$ is as follows: if the triples on the left hand side of $\vdash$ hold, then C satisfies $\{p\}C\{q\}$. To explain the new type of triples above we consider $\{p_i\}k_i\{q_i\}[X_i]$. k_i denotes an identifier, i.e., a placeholder for a *local* command C_i, that is, a command that satisfies safety monotonicity and the frame property. Such commands are determined by environments η, i.e., mappings from identifiers to local commands. In particular the premise is quantified over all environments η that make $\vdash$ holds. The sets X_i in the triples list the variables which each k_i is allowed to change. Replacing k_i in the triple $\{p_i\}k_i\{q_i\}$ with a concrete $C_i = \eta(k_i)$, the triple can be interpreted with usual semantics.

The general hypothetical frame rule only considers an arbitrary number of $\{p_i\}k_i\{q_i\}[X_i]$ triples and therefore does not introduce any new concepts. To get an idea for the usage of this proof rule we consider again its premise. The command C denotes a command that uses during its execution the local commands $\eta(k_i)$ for an actual considered environment η. Now the hypothetical frame rule

allows us to infer triples with more information. The pre- and postconditions of k_i now come with additional disjoint heap cells satisfying the predicate r. Moreover the sets of variables that the k_i may modify are extended by a common set Y. Intuitively this means that all k_i together can be seen as a module or package providing some functionality through its public methods, namely the k_i. Usually the concrete implementation remains hidden by an import of such module.

The premise of the proof rule expresses the described situation. The consequent of the rule gives a view of the module from its inside. It reveals all internally used variables and heap cells used by k_i. All k_i share some private variables and storage. In particular, to be able to work correctly on those resources, each k_i has to maintain a resource invariant r. Due to this behaviour the inference rule comes with more complex variable conditions than the ordinary frame rule. However, it is much more flexible than the ordinary frame rule and allows reasoning in a more realistic setting. The following side conditions come with the hypothetical frame rule to restrict the behaviour for k_i and C

(a) C does not modify any free variables of r, except through k_1 and k_2
(b) Y is disjoint from X_1, X_2, $FV(p_i)$, $FV(q_i)$, $FV(p)$, $FV(q)$ and $MV(C)$.

By these conditions module variables can only be modified within a module.

Before tackling these side conditions we first show that the hypothetical frame rule can be included in our relational approach. The first construct we consider are the triples of the form $\{p_i\}k_i\{q_i\}[X_i]$. For such triples we first define for an arbitrary set X of variables a command C_X by

$$C_X =_{df} \{(\sigma,\sigma') : X \subseteq \mathsf{dom}(s_\sigma) \cap \mathsf{dom}(s_{\sigma'}),\ s_\sigma|_{\overline{X}} = s_{\sigma'}|_{\overline{X}},\ h_\sigma = h_{\sigma'}\}$$

where $\overline{X} = \mathit{Vars} - X$. This command is used to ensure that all variables which are not in a set X have to preserve their value from the input to the output states. Each variable in X can be changed to an arbitrary value. In analogy to Equation (8) we say that a command k *preserves* C_X if $k * (C_X\,;\, S_k\,) \subseteq k * C_X$ By this we further define the new triples by

$$\{p_i\}k_i\{q_i\}[X_i] \Leftrightarrow_{df} \{p_i\}k_i\{q_i\} \wedge k_i \text{ preserves } C_{X_i}\ .$$

Note again that we cannot restrict variable modifications "inside" a command (cf. the example given at the end of Section 2). This is a major restriction of our relational approach. To verify the side condition (a) within *Cmds* we assume a syntactically given command C by the grammar of Section 2. The idea is to split the command C into subsequences C_k of C where no k_i occurs and verify preservation of $[\![C_k]\!]_c$ relationally. To ensure that each C_k of C before and after a k_i preserves r, we apply the following routine to C. We start by a set $Z =_{df} \{C\}$.

1. Repeat until no $C_j \in Z$ contains $\mathsf{if\ then\ else}$: if there exists C_i with $C \equiv C_1;(\,\mathsf{if}\, p \,\mathsf{then}\, C_2 \,\mathsf{else}\, C_3)\,;C_4$ then $Z := Z - \{C\} \cup \{C_1\,;p\,;C_2\,;C_4,\ C_1\,;\neg p\,;C_3\,;C_4\}$
2. Repeat until no $C_j \in Z$ contains a k_i : if $\exists C_i\,.\ C \equiv C_1\,;k_i\,;C_2$ then $Z := Z - \{C\} \cup \{C_1,\ \mathsf{cod}(C_1\,;k_i)\,;C_2\}$

$\equiv$ denotes syntactical equivalence. Since C_2 is reached after $C_1\,;k_i$ it remains to consider $\mathsf{cod}(C_1\,;k_i)\,;C_2$. To consider the right command relation in C_j the

routine appends tests to commands e.g. in $p\,;C_j$ although p is syntactically not a command. A concrete example is given in the next section.

Now each of these subcommands has to preserve r, i.e., it has to maintain the values of all free variables of r. By knowing the concrete structure of a command, Assumption (a) can be checked completely at the relational level.

Next we introduce an approach to characterise Assumption (b) relationally. We constrain the use of the internal variables Y of a module by the following inequations which are to be add to the premises of the proof rule.

$$k_i \text{ preserves } v_Y \ , \quad i \in \{1,2\} \tag{9}$$

$$C \text{ preserves } v_Y \ , \tag{10}$$

where $v_Y =_{df} \{(\sigma,\sigma) : \sigma = (s,h),\ Y \subseteq \mathsf{dom}(s),\ h \in \mathit{Heaps}\}$.

By Assumption (9) no variable in Y is changed by any execution of k_i. Therefore also each execution of k_i starting from p_i and finishing in q_i preserves Y which the intention of requiring Y to be disjoint from $FV(p_i)$ and $FV(q_i)$. The same argumentation holds for requiring each X_i to be is disjoint from Y. By the second assumption also C does not modify any variables of Y.

In summary, the intended restrictions by all meta-level variable conditions of the hypothetical frame rule can be checked pointfree and purely relationally when knowing the concrete structure and syntax of commands. To demonstrate and clarify these conditions we present a short example in the next section.

8 A Relational Treatment of a Queue Module

In this section, we exemplarily replay the queue module example given in [9]. This module offers two methods to enqueue and to dequeue elements from a list which represents the shared and hidden storage of the module procedures. From the outside of the module the list cannot be seen, i.e., from that point of view only values will be cut off from or appended to an abstract sequence α stored in a variable Q. The following two interface specifications[3] are given

$$\{Q = \alpha \wedge z = n \wedge \mathsf{emp}\}\,\mathtt{enq}\,\{Q = \alpha \cdot n \wedge \mathsf{emp}\}[\{Q\}] \ ,$$
$$\{Q = n \cdot \alpha \wedge \mathsf{emp}\}\,\mathtt{deq}\,\{Q = \alpha \wedge z = n \wedge \mathsf{emp}\}[\{Q\}]$$

and will be part of the antecedent of the hypothetical frame rule. The precondition for `enq` ensures that Q stores the sequence α of the hidden list while z stores an arbitrary value n to be appended at the end of the sequence. An environment η_1 could e.g. return $\eta_1(\mathtt{enq}) = (Q := Q \cdot z)$. In `deq`, the head of the sequence α is assigned to z and then deleted from α. A possible local command for that could be $\eta_1(\mathtt{deq}) = (z := \mathsf{head}(\alpha)\ ;\ Q := \mathsf{tail}(\alpha))$.

These specifications can be embedded into the relational framework by simply requiring $\eta_1(\mathtt{enq}) * (C_{\{Q\}}\,;\, S_{\eta_1(\mathtt{enq})}\,) \subseteq \eta_1(\mathtt{enq}) * C_{\{Q\}}$ and $(Q = \alpha\,; z = n\,; \mathsf{emp})\,;$ $\eta_1(\mathtt{enq}) \subseteq \eta_1(\mathtt{enq})\,; (Q = \alpha \cdot n\,; \mathsf{emp})$. The same can be done for `deq`.

The conclusion of the hypothetical frame rule reveals the resource invariant of the module. Concretely the resource invariant r for this module is $\mathsf{listseg}(Q,x,y)*$

[3] In [9] the specifications are used parametrical; for simplicity reasons we use values.

$(y \mapsto -, -)$ which ensures a list segment from x to y representing a sequence stored in Q. The predicate $\mathsf{listseg}(Q, x, y)$ is defined by $(x = y \wedge \alpha = \varepsilon \wedge \mathsf{emp}) \vee (x \neq y \wedge \exists v, z.\ x \mapsto v, z * \mathsf{listseg}(\mathsf{tail}(Q), z, y))$. The last two cells starting from y reserve storage for a value that an execution of `enq` will append. By this an internal implementation mapped by η_2, i.e., local commands for `enq` and `deq` might look as follows

$$\begin{array}{ll} Q := Q \cdot z; & Q := \mathsf{tail}(Q); \\ t := \mathsf{cons}(-, -); & z := [x]\,;\, t := x\,;\, x := [x+1]; \\ [y] := z\,;\, [y+1] := t\,;\, y := t & \mathsf{dispose}\ t \end{array}$$

In particular the following triples are inferred by the hypothetical frame rule.

$$\{(Q = \alpha \wedge z = n \wedge \mathsf{emp}) * r\}\, \mathtt{enq}\, \{(Q = \alpha \cdot n \wedge \mathsf{emp}) * r\}[\{Q\}, \{t, x, y\}]$$
$$\{(Q = n \cdot \alpha \wedge \mathsf{emp}) * r\}\, \mathtt{deq}\, \{(Q = \alpha \wedge z = n \wedge \mathsf{emp}) * r\}[\{Q\}, \{t, x, y\}]$$

Again we can formulate these triples relationally with the variable condition modelled by $\eta_2(\mathtt{enq}) * (C_{\{Q\}\cup\{t,x,y\}}\ ;\ S_{\eta_2(\mathtt{enq})}) \subseteq \eta_2(\mathtt{enq}) * C_{\{Q\}\cup\{t,x,y\}}$. Next we consider for the hypothetical frame rule the command

$$C = \mathsf{if}\ b\ \mathsf{then}\ (\eta_2(\mathtt{deq})\,;\, z = 1\,;\, \eta_2(\mathtt{enq}))\ \mathsf{else}\ (\eta_2(\mathtt{deq})\,;\, z = 0\,;\, \eta_2(\mathtt{enq}))\ .$$

It is split into $C_1 = (b = \mathsf{true})\,;\, (\eta_2(\mathtt{deq})\,;\, z := 1\,;\, \eta_2(\mathtt{enq}))$ and $C_2 = (b = \mathsf{false})\,;\, (\eta_2(\mathtt{deq})\,;\, z := 0\,;\, \eta_2(\mathtt{enq}))$. Consequently we split both commands recursively at $\eta_2(\mathtt{deq})$ and $\eta_2(\mathtt{enq})$. This results in commands $C_{1_1} = (b = \mathsf{true})$, $C_{1_2} = (\mathsf{cod}(b = \mathsf{true}\,;\, \eta_2(\mathtt{deq}))\,;\, z := 1)$, $C_{1_3} = \mathsf{cod}(b = \mathsf{true}\,;\, \eta_2(\mathtt{deq})\,;\, z := 1)$, $C_{2_1} = (b = \mathsf{false})$, $C_{2_2} = (\mathsf{cod}(b = \mathsf{false}\,;\, \eta_2(\mathtt{deq}))\,;\, z := 0)$ and $C_{2_3} = \mathsf{cod}(b = \mathsf{false}\,;\, \eta_2(\mathtt{deq})\,;\, z := 0)$. All commands C_{i_j} preserve r. For example it can be shown for C_{1_2} that $C_{1_2} * (r\ ;\ S_{C_{1_2}}) \subseteq C_{1_2} * r$.

This shows that restrictions by side conditions of the hypothetical frame rule and the rule itself can be expressed in the presented relational framework. Only the subcommands as given by the structure of a command is needed for a relation based argumentation which facilitates algebraic reasoning with this rule.

9 Greatest Local Relations

To conclude this work, we introduce the concept of *greatest local relations*[4]. As discussed before, the premise and the conclusion of the hypothetical frame rule are quantified over all possible environments η. According to [9], proving soundness of the inference rule can be simplified. It suffices to consider so-called greatest environments since those environments already capture any other environment. A greatest environments maps an identifier k_i of $\{p_i\}k_i\{q_i\}[X_i]$ to the greatest local relation satisfying it. For further details see [9].

Our basic intention in this work is to include greatest local relations and their variable conditions into our setting. This underpins that the relational approach is also able to capture concepts used in the hypothetical frame rule. We give a pointfree characterisation for these commands in the relational setting and explain how their variable conditions can be included.

[4] A local relation of [9] is a local command in our approach.

Definition 9.1 ([9]) Consider a triple $\{p\}k_i\{q\}[X]$. We define the greatest local relation $\mathsf{great}(p,q,X)$ satisfying that triple by the following conditions assuming $\sigma = (s,h)$ and $\sigma' = (s',h')$

1. σ is safe for $\mathsf{great}(p,q,X) \iff \sigma \in p * \mathsf{true}$
2. $\sigma\ \mathsf{great}(p,q,X)\ \sigma' \iff$
   (a) $\forall x \notin X.\ s(x) = s'(x)$
   (b) $\forall h_p, h_I.\ (h = h_p \cup h_I \wedge h_p \cap h_I = \emptyset \wedge (s,h_p) \in p)$
   $\Rightarrow \exists h_q.\ h_q \cap h_I = \emptyset \wedge h' = h_q \cup h_I \wedge (s',h_q) \in q$

By this definition every state $\sigma = (s,h)$ which $\mathsf{great}(p,q,X)$ is executed at can be split into two disjoint states $(s,h_p) \in p$ and an arbitrary state (s,h_I). Every variable of $\mathsf{dom}(s) - X$ cannot be modified by $\mathsf{great}(p,q,X)$ and a resulting state σ' can be split again into states (s',h_I) and $(s',h_q) \in q$.

We implicitly treat the variable condition (2a) by an extra assumption. Hence we omit X in $\mathsf{great}(p,q)$ and include the variable condition (2a) by requiring for the rest of this section $\mathsf{great}(p,q)$ preserves C_X with C_X as in Section 4.

For a relational characterisation of the remaining properties of Definition 9.1 we use the concept of residuals. In REL, the *right residual* $R\backslash S$ is defined as $\overline{R^\smile ; \overline{S}}$ (e.g., [12]). Residuals can equally be defined by the Galois connection $x \leq a\backslash b \Leftrightarrow_{df} a \cdot x \leq b$ in the general setting of quantales [2]. Therefore the presented theory lifts to a more abstract setting. The definition entails $a \cdot (a\backslash b) \leq b$.

Moreover, for this section we define $S =_{df} S_{C_X}$ and let $\top$ denote the universal relation. By these definitions, we characterise in our relational terms $\mathsf{great}(p,q)$ satisfying the properties (1) and (2b) as follows:

Lemma 9.2 *For tests p, q the greatest local relation $\mathsf{great}(p,q)$ can be characterised by*

$$\mathsf{great}(p,q) = (p * I)\ ;\ \mathsf{res}(p,q)$$

where $\mathsf{res}(p,q) = ((p \times \mathsf{dom}(S))\ ;\ \rhd) \setminus ((\top\ ;\ q \times S)\ ;\ \rhd)$.

Proof. We show that (2b) is satisfied by $\mathsf{res}(p,q)$. We have for arbitrary σ and τ

$$\begin{aligned}
& \sigma\ \overline{((p \times \mathsf{dom}(S))\ ;\ \rhd) \setminus ((\top\ ;\ q \times S)\ ;\ \rhd)}\ \tau \\
\Leftrightarrow\ & \sigma\ \lhd\ ;\ (p \times \mathsf{dom}(S))\ ;\ \overline{(\top\ ;\ q \times S)\ ;\ \rhd}\ \tau \\
\Leftrightarrow\ & \neg(\exists \sigma_p, \sigma_S.\ \sigma = \sigma_p * \sigma_S \wedge \sigma_p \# \sigma_S \wedge \sigma_p \in p \wedge \sigma_S \in \mathsf{dom}(S) \\
& \wedge \neg((\sigma_p, \sigma_S)\ (\top\ ;\ q \times S)\ ;\ \rhd\ \tau)) \\
\Leftrightarrow\ & \neg(\exists \sigma_p, \sigma_S.\ \sigma = \sigma_p * \sigma_S \wedge \sigma_p \# \sigma_S \wedge \sigma_p \in p\ \wedge \sigma_S \in \mathsf{dom}(S) \\
& \wedge \neg(\exists \sigma_q, \sigma'_S.\ \sigma_p\ \top\ \sigma_q \wedge \sigma_q \in q \wedge \sigma_S\, S\, \sigma'_S \wedge \tau = \sigma_q * \sigma'_S \wedge \sigma_q \# \sigma'_S)) \\
\Leftrightarrow\ & \forall \sigma_p, \sigma_S.\ \sigma = \sigma_p * \sigma_S \wedge \sigma_p \# \sigma_S \wedge \sigma_p \in p \\
& \Rightarrow \exists \sigma_q, \sigma'_S.\ \sigma_q \in q \wedge \sigma_S\, S\, \sigma'_S \wedge \tau = \sigma_q * \sigma'_S \wedge \sigma_q \# \sigma'_S
\end{aligned}$$

Moreover for (1) in Definition 9.1 we know $\mathsf{dom}(\mathsf{great}(p,q)) \subseteq p * I$ and by pointwise calculations above we have $p * I \subseteq \mathsf{dom}(\mathsf{res}(p,q))$. Then $p * I = (p * I) * (p * I) \subseteq (p * I)\ ;\ \mathsf{dom}(\mathsf{res}(p,q)) = \mathsf{dom}((p * I)\ ;\ \mathsf{res}(p,q)) = \mathsf{dom}(\mathsf{great}(p,q))$. □

The assumption $\sigma_S \, S \, \sigma'_S$ is used to change all variables of X in s_{σ_S}. Notice also that $\mathsf{res}(p,q)$ does not work on pairs of states, i.e., $\mathsf{res}(p,q) \subseteq \mathit{States} \times \mathit{States}$. With this characterisation, we can give relational proofs for the following results.

Lemma 9.3 *For all p, q, we have that $\mathsf{great}(p,q)$ satisfies $\{p\}\mathsf{great}(p,q)\{q\}$.*

Proof. For the proof we need a standard result from relation algebra: if R_1, R_2 and S_1 are subidentities with $R_1 \subseteq S_1$ then $(R_1 \times R_2) \,;\, (S_1 \times I) = R_1 \times R_2$.

To show $p \,;\, \mathsf{great}(p,q) \subseteq \mathsf{great}(p,q) \,;\, q$ it remains to prove $p \,;\, \mathsf{great}(p,q) \subseteq \top \,;\, q$. Both inequations are equivalent by standard semiring theory. We calculate assuming $p = p * \mathsf{emp} = p * (\mathsf{emp}; \mathsf{dom}(S))$

$$
\begin{array}{ll}
 & p \,;\, \mathsf{great}(p,q) \\
= & \lhd \,;\, (p \times \mathsf{emp}) \,;\, \rhd \,;\, \mathsf{res}(p,q) \\
= & \lhd \,;\, (p \times \mathsf{emp}) \,;\, (p \times \mathsf{dom}(S)) \,;\, \rhd \,;\, \mathsf{res}(p,q) \\
\subseteq & \lhd \,;\, (p \times \mathsf{emp}) \,;\, (\top \,;\, q \times S) \,;\, \rhd \\
= & \lhd \,;\, (p \,;\, \top \,;\, q) \times (\mathsf{emp} \,;\, S) \,;\, \rhd \\
\subseteq & p \,;\, \top \,;\, q \subseteq \top \,;\, q
\end{array}
$$

Moreover, $p \subseteq \mathsf{dom}(\mathsf{great}(p,q))$ follows from $p \subseteq p * I = \mathsf{dom}(\mathsf{great}(p,q))$. □

Next we characterise $\mathsf{great}(p,q)$ also as a local command. Informally, the inequation below is similar to the frame property and also characterises arbitrary storage not needed for $\mathsf{great}(p,q)$ to remain untouched.

Lemma 9.4 *For arbitrary tests p and q we have*

$$((p * I) \times \mathsf{dom}(S)) \,;\, \rhd \,;\, \mathsf{great}(p,q) \subseteq ((p * I) \,;\, \mathsf{great}(p,q) \times S) \,;\, \rhd$$

Proof. We calculate for all $\sigma_p \in p, \sigma_2 \in \mathsf{dom}(S)$ and arbitrary σ_1, σ'

$$
\begin{array}{ll}
 & \sigma_p \# \sigma_1 \wedge (\sigma_p * \sigma_1, \sigma_2) \; ((p * I) \times \mathsf{dom}(S)) \,;\, \rhd \,;\, \mathsf{great}(p,q) \; \sigma' \\
\Leftrightarrow & \sigma_p \# \sigma_1 \wedge \sigma_p * \sigma_1 \in p * I \wedge (\sigma_p * \sigma_1) \# \sigma_2 \wedge (\sigma_p * \sigma_1) * \sigma_2 \; \mathsf{great}(p,q) \; \sigma' \\
\Leftrightarrow & \sigma_p \# \sigma_1 \wedge \sigma_p \in p \wedge \sigma_p \# (\sigma_1 * \sigma_2) \wedge \sigma_p * (\sigma_1 * \sigma_2) \; \mathsf{great}(p,q) \; \sigma' \\
\Rightarrow & \sigma_p \# \sigma_1 \wedge \sigma_p \in p \wedge (\sigma_p * \sigma_1) \# \sigma_2 \wedge (\sigma_p * \sigma_1) * \sigma_2 \; \mathsf{great}(p,q) * S \; \sigma' \\
\Rightarrow & \exists \sigma'_1, \sigma_S. \; \sigma_p \# \sigma_1 \wedge \sigma_p \in p \wedge (\sigma_p * \sigma_1) \# \sigma_2 \wedge (\sigma_p * \sigma_1) \; \mathsf{great}(p,q) \; \sigma'_1 \wedge \\
 & \sigma_2 S \sigma_S \wedge \sigma'_1 \# \sigma_S \wedge \sigma'_1 * \sigma_S = \sigma' \\
\Leftrightarrow & \exists \sigma'_1. \; (\sigma_p * \sigma_1, \sigma_2) \; ((p * I) \,;\, \mathsf{great}(p,q) \times S) \,;\, \rhd \; \sigma'
\end{array}
$$

□

Lemma 9.5 *The relation $\mathsf{great}(p,q)$ is local, i.e., it satisfies the frame property and safety monotonicity.*

Proof. To see that $\mathsf{great}(p,q)$ satisfies the frame property, we know by Lemma 9.4

$$((p * I) \times \mathsf{dom}(S)) \,;\, \rhd \,;\, \mathsf{great}(p,q) \subseteq ((p * I) \,;\, \mathsf{great}(p,q) \times S) \,;\, \rhd.$$

By $\mathsf{dom}(\mathsf{great}(p,q)) = p * I$ and isotony the claim follows. Finally for safety monotonicity: $\mathsf{dom}(\mathsf{great}(p,q)) * I = (p * I) * I = p * I = \mathsf{dom}(\mathsf{great}(p,q))$. □

Lemma 9.6 *Consider an arbitrary local command C that satisfies the triple $\{p\}C\{q\}$ and $S_C \subseteq S$, i.e., C modifies variables in X. Then $p;C \subseteq \mathsf{great}(p,q)$.*

Proof. First we show $C \subseteq ((p\times\mathsf{dom}(S));\rhd) \setminus ((\top;q\times S);\rhd)$ which is equivalent to $(p \times \mathsf{dom}(S)) ; \rhd ; C \subseteq (\top ; q \times S) ; \rhd$. We calculate

$$\begin{array}{ll} & (p \times \mathsf{dom}(S)) ; \rhd ; C \\ \subseteq & (p \times \mathsf{dom}(S)) ; (\mathsf{dom}(C) \times \mathsf{dom}(S)) ; \rhd ; C \\ \subseteq & (p \times \mathsf{dom}(S)) ; (C \times S_C) ; \rhd \\ \subseteq & (p ; C \times S) ; \rhd \\ \subseteq & (C ; q \times S) ; \rhd \subseteq (T ; q \times S) ; \rhd \end{array}$$

Then $p ; C \subseteq (p * I) ; C \subseteq (p * I) ; \mathsf{res}(p,q) = \mathsf{great}(p,q)$. □

It can be seen that if a command C satisfies $\{p\}\, C\, \{q\}[X]$ then $p ; C$ is a subset of $\mathsf{great}(p,q)$, i.e., $\mathsf{great}(p,q)$ captures every such local command C.

10 Related Work

There exist different approaches to the issue of variable preservation in proof rules. We briefly sum up some proposals of [3, 10]. One idea is to omit the store and to put variable declarations on the heap, i.e., treat variables the same way as heap cells. Although this would give a uniform treatment, the logic itself would become more complicated, especially Hoare's variable assignment axiom. For our purpose this seems inadequate since the presented case study requires variable declarations to be present for disjoint states; such a resource-based treatment of variables would exacerbate treating standard examples.

The main approach taken in [3, 10] was to introduce ownership predicates for variables that remain unchanged by variable substitutions. It ensures the permission to write to certain variables. This approach tracks ownership rights of variables in states and treats them by a special $*$ operation. This also shifts the treatment of variable conditions to the logic layer. Therefore the conditions can be verified within the logic. Our goal is rather constraining our relational approach by equations that model the side conditions. This allows including plain the store-heap model into our abstract and pointfree treatment without adding additional information like ownership rights. In addition our treatment also enables reasoning purely at the relational level.

11 Conclusion and Outlook

We have studied to which extent side conditions that naturally appear in the frame rule of separation logic can be handled with the same mechanism as the logic itself. The approach presented expresses the side conditions in the same algebra as the logic. In classical logic, side conditions require the introduction of a meta-language; in our algebraic version this is not the case.

As a further application of our approach we have formulated the hypothetical frame rule and its more complex variable side conditions in our relational setting.

The result of this formulation is that our abstract approach can also be used for information hiding proof rules. Especially, greatest local relations can be characterised in a pointfree way in our setting.

As further work we will try to overcome the deficit of the relational approach by working on strings to represent the concrete structure of commands. Moreover, an abstract treatment of the hypothetical frame rule seems to be possible since its meta-level conditions can be lifted to an abstract level. As a consequence a wider range of models can be considered.

Acknowledgements: We are grateful to Bernhard Möller for many valuable remarks and comments. We are also most grateful to the referees who pointed out several flaws and helped to significantly increase the quality.

References

1. Berdine, J., Calcagno, C., O'Hearn, P.W.: Smallfoot: Modular Automatic Assertion Checking with Separation Logic. In: de Boer, F., Bonsangue, M.M., Graf, S., de Roever, W. (eds.) Formal Methods for Components and Objects (FMCO2005). LNCS, vol. 4111, pp. 115–137. Springer (2006)
2. Birkhoff, G.: Lattice Theory, Colloquium Publications, vol. XXV. American Mathematical Society, 3rd edn. (1967)
3. Bornat, R., Calcagno, C., Yang, H.: Variables as resource in separation logic. Electronic Notes in Theoretical Computer Science 155, 247–276 (2006)
4. Dang, H.H., Höfner, P., Möller, B.: Towards algebraic separation logic. In: Berghammer, R., Jaoua, A., Möller, B. (eds.) Relations and Kleene Algebra in Computer Science. LNCS, vol. 5827, pp. 59–72. Springer (2009)
5. Dang, H.H., Höfner, P., Möller, B.: Algebraic Separation Logic. Tech. Rep. 2010-06, Institute of Computer Science, University of Augsburg (2010)
6. Dang, H.H., Höfner, P., Möller, B.: Algebraic Separation Logic. J. Logic and Algebraic Programming (2011), (accepted)
7. Kozen, D.: On Hoare logic, Kleene algebra, and types. In: Gärdenfors, P., Woleński, J., Kijania-Placek, K. (eds.) In the Scope of Logic, Methodology, and Philosophy of Science: Volume One of the 11th Int. Congress Logic, Methodology and Philosophy of Science, Studies in Epistemology, Logic, Methodology, and Philosophy of Science, vol. 315, pp. 119–133. Kluwer (2002)
8. O'Hearn, P.W., Reynolds, J.C., Yang, H.: Local reasoning about programs that alter data structures. In: Fribourg, L. (ed.) CSL '01: 15th International Workshop on Computer Science Logic. LNCS, vol. 2142, pp. 1–19. Springer (2001)
9. O'Hearn, P.W., Reynolds, J.C., Yang, H.: Separation and information hiding. ACM Trans. Program. Lang. Syst. 31(3), 1–50 (2009)
10. Parkinson, M., Bornat, R., Calcagno, C.: Variables as Resource in Hoare logics. In: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science. pp. 137–146. IEEE Computer Society (2006)
11. Reynolds, J.C.: An introduction to separation logic. In: Broy, M. (ed.) In Engineering Methods and Tools for Software Safety and Security. pp. 285–310. IOS Press (2009)
12. Schmidt, G., Ströhlein, T.: Relations and Graphs: Discrete Mathematics for Computer Scientists. Springer (1993)