Local Reasoning about Data Update

Cristiano Calcagno, Philippa Gardner and Uri Zarfaty

Department of Computing
Imperial College London
London, UK
{ccris,pg,udz}@doc.ic.ac.uk

Abstract

We present local Hoare reasoning about data update, using Context Logic for analysing structured data. We apply our reasoning to tree update, heap update which is analogous to local Hoare reasoning using Separation Logic, and term rewriting.

Keywords: context logic, local reasoning, tree update, heap update

1   Introduction

Structured data update is pervasive in computer systems: examples include heap update on local machines, information storage on hard disks, the update of distributed XML databases, and general term rewriting. Programs for manipulating such dynamically-changing data are notoriously difficult to write correctly. Hoare Logics provide a standard technique for reasoning about update. Such reasoning has been widely studied for heap update. Hoare reasoning has hardly been studied for other examples of update, such as tree update, and there has been little attempt at a unified theory.

We present local Hoare reasoning about data update, using Context Logic (CL) for reasoning directly about structured data. CL arose from two independent bodies of work: Separation Logic (SL) [13,15,7] for reasoning locally about heap update, based on the general theory of Bunched Logic (BL) [9]; and Ambient Logic (AL) [3,4] for reasoning locally about static trees. A natural question is whether AL can be used to reason about tree update. We demonstrate that this is not possible. Instead, we fundamentally change the way we reason about structured data, by introducing CL. Data update typically identifies the portion of data to be replaced, removes it, and inserts the new data *in the same place*. With CL, we reason about both data and this place of insertion (contexts).

Our local Hoare reasoning follows the style of local reasoning first introduced in [10] using SL. The motivating idea is that the atomic commands tend to operate in a local way, by only accessing a small part of the current data, called the *footprint* of the command. Local Hoare reasoning reflects this local operation of the commands, in both the specifications and the proofs of the commands' behaviour. It uses *small axioms* to specify the behaviour of atomic commands on their footprints, and the *frame rule* to specify that the rest of the data (the context) remains unchanged. Our local Hoare reasoning relies crucially on CL-reasoning, whose structural connectives specifically mirror this separation of the footprint from the surrounding context.

This paper is the full version of our conference paper [1]. The work presented in [1] focussed on reasoning about tree update. Here, we present a more general account of CL, and a local Hoare reasoning framework for reasoning about data update which then uniformly applies to tree update, heap update and term rewriting.

Context Logic

CL extends the standard propositional connectives with additional *structural* connectives for reasoning directly about subdata. The *structural application* $K(P)$ specifies that data can be split into subdata satisfying formula P and a context satisfying formula K. There are two corresponding *structural adjoints*: the data formula $K \triangleleft P$ denoting that, whenever the given data is placed in a context satisfying K, then the result must satisfy P; and the context formula $P \triangleright Q$ denoting that, whenever data satisfying P is applied to the given context, then the result satisfies formula Q. The structural application and the $\triangleleft$ adjoint have analogues in AL; the $\triangleright$ adjoint does not. This additional adjoint is essential for describing

weakest preconditions of our Hoare reasoning.

In this paper, we give the basic definition of CL, its proof theory and models. We prove completeness in [2]. We also give an extension of CL, called CL_0, which consists of an additional zero formula 0 and accompanying axioms for specifying empty data. CL_0 has interesting additional logical structure, including a derived composition formula and accompanying right adjoints which generalise the structural connectives of BL and SL. We demonstrate that CL_0-reasoning collapses to a variant of BL-reasoning for certain models. Our variant of BL permits a non-commutative structural composition on data, so that we can analyse sequences. We show that CL_0-reasoning is identical to BL-reasoning for multisets and heaps, whose contexts have the same structure as the data, but is more powerful for sequences and trees, whose contexts are more complex than the data. We also study CL-reasoning for terms. Although terms can be seen as special cases of trees, there is a crucial difference: terms over a signature do not have a natural empty term, and do not decompose as a composition of subterms due to the fixed arity of the function symbols. They do however decompose nicely as context/subtree pairs. We can therefore apply CL-reasoning, but not BL-reasoning, to terms. These examples demonstrate the generality of our CL-reasoning.

Local Hoare Reasoning

We present a framework for local Hoare reasoning about commands acting locally on data, by describing a general interpretation of the Hoare triples, a general definition of local command, and general inference rules for Hoare triples based on such local commands. We then apply our general Hoare reasoning to three examples of data update: tree update, heap update which is exactly analogous to the reasoning based on SL, and term rewriting which had previously escaped reasoning using SL. We illustrate our Hoare reasoning using the tree dispose command $[n]_\mathrm{T} := 0$, which disposes a subtree with top node identified by node variable n. The small axiom for this command is

$$\{n[\mathrm{true}]\}\ [n]_\mathrm{T} := 0\ \{0\}$$

The precondition $n[\mathrm{true}]$ is a data formula defined specifically for trees. It specifies a tree with top node given by n, whose subforest is unspecified. This precondition describes properties about the footprint of the command, in this case the subtree identified by n, and is the minimal safety condition necessary for the command to execute. The postcondition only describes the result of the action of the command on the footprint, in this case the empty tree specified by formula 0. To extend the small-axiom reasoning to properties about larger trees, we use a generalised frame rule to derive

$$\{K(n[\mathrm{true}])\}\ [n]_\mathrm{T} := 0\ \{K(0)\}$$

The precondition states that the tree can be split disjointedly into a subtree with top node n, and a context satisfying context formula K. The definition of local commands and the rules ensure that this context is unaffected by the update command. The postcondition thus has the same structure, with K now applied to 0.

The general rules and the small axioms for our examples are complete for

straight-line code, which we demonstrate by showing that the *weakest precondition axioms* are derivable. As well as providing an important sanity check, such axioms play a fundamental role in the design of verification tools. In our example of tree dispose, the derivable weakest precondition axiom is:

$$\{(0 \rhd P)(n[\text{true}])\}[n]_{\text{T}} := 0\{P\}$$

This weakest precondition simply specifies that the tree can be split into a subtree with top node n, and a context which satisfies P when the empty tree is put in the hole. Although it is possible to define small axioms and frame rules using AL instead of CL, it is not possible to define the weakest preconditions.

Our local Hoare reasoning about heap update is identical to that given [10] using SL. In fact, our Hoare reasoning about tree update and heap update is remarkably similar. This similarity does not just occur with the small axioms and weakest precondition axioms, but also at the level of the proofs that the weakest precondition axioms are derivable. The comparison with our Hoare reasoning for term rewriting is less immediate, since the nature of term rewriting is different from our languages for tree and heap update. However, it is clear that the same general principles apply, both at the level of specification and at the level of proof. Our Hoare reasoning seems to be robust with respect to the style of update language chosen. Indeed, our examples suggest that it might be possible to develop a general theory of local Hoare reasoning about data update.

2 Context Logic

We give the basic definition of CL, its proof theory and models (section 2.1). We extend this basic CL to include an additional zero formula for specifying empty data (section 2.2), which provides interesting additional logical structure and a comparison with BL (section 2.2.2). Finally, we study the application of CL reasoning to four example models of structured data: sequences, multisets, trees and terms (section 2.3).

2.1 Basic Context Logic

CL consists of data formula denoted by P and context formulae denoted by K. These formulae are constructed from standard propositional connectives, and less familiar structural connectives for directly analysing the data and context structure.

Definition 2.1 [CL-Formulae] The set of CL formulae consists of disjoint sets of *data formulae* $\mathcal{P}$ and *context formulae* $\mathcal{K}$, constructed from the grammars:
data formulae

$$P ::= K(P) \mid K \lhd P \qquad \text{structural formulae}$$
$$P \lor P \mid \neg P \mid \text{false} \qquad \text{boolean additive formulae}$$

context formulae

$$K ::= I \mid P \rhd P \qquad \text{structural formulae}$$
$$K \lor K \mid \neg K \mid \text{False} \qquad \text{boolean additive formulae}$$

We use boolean additive formulae, since this is the standard choice when using a logic to analyse specific models. It would also be interesting to study intuitionistic CL. The key formulae are the structural formulae $K(P)$, $K \lhd P$, $P_1 \rhd P_2$ and I. The *application formula* $K(P)$ specifies that the given data element can be split into a context satisfying K applied to data satisfying P. For example, if we define the context formula True $\triangleq \neg$False, then the formula True(P) states that some subdata satisfies property P. The next two formulae are both (right) adjoints of application. The formula $K \lhd P$ is satisfied by the given data if, *whenever* we insert the data into a context satisfying K, then the result satisfies P. For example, the formula (True $\lhd P$) states that, when the data is put in any context, the result satisfies property P. Meanwhile, $P_1 \rhd P_2$ is a statement on contexts. It is satisfied by a given context if, *whenever* we insert in the context some data satisfying P_1, then the result satisfies P_2. Given the derived data formula true $\triangleq \neg$false, the context formula (true$\rhd P_2$) states that, regardless of what data is put in the context hole, the resulting data satisfies property P_2. This adjoint is essential for expressing weakest preconditions for update commands (section 3). The context formula I specifies that a context equals the empty context.

We give a simple Hilbert-style proof theory, following the style for BL in [12]. The axioms and rules for the structural operators state that $K \lhd P_2$ and $P_1 \rhd P_2$ are right adjoints of $K(P_1)$, and that I is the identity of application.

Definition 2.2 [CL-Proof Theory] The Hilbert-style CL-proof theory consists of the standard axioms and rules for the boolean additive connectives (including cut), and the following axioms and rules for the structural connectives:

$$P \dashv\vdash_\mathcal{P} I(P) \qquad \frac{K_1 \vdash_\mathcal{K} K_2 \qquad P_1 \vdash_\mathcal{P} P_2}{K_1(P_1) \vdash_\mathcal{P} K_2(P_2)}$$

$$\frac{K(P_1) \vdash_\mathcal{P} P_2}{K \vdash_\mathcal{K} P_1 \rhd P_2} \qquad \frac{K \vdash_\mathcal{K} P_1 \rhd P_2 \qquad P \vdash_\mathcal{P} P_1}{K(P) \vdash_\mathcal{P} P_2}$$

$$\frac{K(P_1) \vdash_\mathcal{P} P_2}{P_1 \vdash_\mathcal{P} K \lhd P_2} \qquad \frac{P_1 \vdash_\mathcal{P} K \lhd P_2 \qquad K_1 \vdash_\mathcal{K} K}{K_1(P_1) \vdash_\mathcal{P} P_2}$$

We sometimes omit the subscripts in $\vdash_\mathcal{P}$ and $\vdash_\mathcal{K}$, and sometimes write $\vdash_{\mathrm{CL}}$ to refer explicitly to this CL-proof theory.

We use the standard derived classical formulae for both data and context formulae: true, $P \wedge P$ and $P \Rightarrow P$; similarly for contexts, writing True for the context formula that is always true. We also use the following derived structural formulae.

Definition 2.3 (CL-Derived Formulae)

- $\diamond P \triangleq \mathit{True}(P)$ *specifies that somewhere property P holds and* $\Box P \triangleq \neg(\diamond\neg P)$ *specifies that everywhere property P holds.*

- $P_1 \blacktriangleright P_2 \triangleq \neg(P_1 \rhd \neg P_2)$ *specifies that there exists some data element satisfying property P_1 such that, when it is put in the hole of the given context, the resulting data satisfies P_2.*

- $K \blacktriangleleft P_2 \triangleq \neg(K \lhd \neg P_2)$ *specifies that there exists a context satisfying property K such that, when the given data element is put in the hole, the resulting data satisfies P_2.*

The order of binding precedence is: $\neg$, $\wedge$, $\vee$, $\{\lhd, \rhd, \blacktriangleleft, \blacktriangleright\}$ and $\Rightarrow$, with no precedence between the elements in $\{\lhd, \rhd, \blacktriangleleft, \blacktriangleright\}$.

The simple presentation given here emphasises the right adjoint properties of $\lhd$ and $\rhd$. In [2], we show that this proof theory is equivalent to the standard classical ML-proof theory plus an additional set of axioms specific to CL. This alternative ML-formulation emphasises the derived connectives $\blacktriangleright$ and $\blacktriangleleft$ instead of the adjoints $\rhd$ and $\lhd$. We have hardly studied the CL-proof theory. It should be possible to prove a cut-elimination result, following the analogous result for BL [12]. Also, Simpson's proof theory for intuitionistic modal logic [14] must surely yield a proof theory for intuitionistic CL.

We give some basic properties of validity: the $\vee$–connective distributes over application and the $\wedge$–connective partially distributes, these results are typical for this style of logical reasoning; the interplay between the structural adjoints $\rhd$, $\lhd$ and

application is rather like standard modus ponens for the additive connectives; and our derived somewhere and everywhere modalities for data also satisfy the T-axiom of modal logic.

Lemma 2.4 (Basic properties of validity)

$$\vee\text{-}distributivity \qquad (K_1 \vee K_2)(P) \dashv\vdash_{\mathcal{P}} K_1(P) \vee K_2(P)$$

$$K(P_1 \vee P_2) \dashv\vdash_{\mathcal{P}} K(P_1) \vee K(P_2)$$

$$\wedge\text{-}semi\text{-}distributivity \qquad (K_1 \wedge K_2)(P) \vdash_{\mathcal{P}} K_1(P) \wedge K_2(P)$$

$$K(P_1 \wedge P_2) \vdash_{\mathcal{P}} K(P_1) \wedge K(P_2)$$

$$modus\text{-}ponens\ results \qquad (P \rhd P')(P) \vdash_{\mathcal{P}} P' \wedge \mathrm{True}(P)$$

$$K(K \lhd P) \vdash_{\mathcal{P}} P \wedge K(\mathrm{true})$$

$$P' \wedge \mathrm{True}(P) \vdash_{\mathcal{P}} (P \blacktriangleright P')(P)$$

$$P \wedge K(\mathrm{true}) \vdash_{\mathcal{P}} K(K \blacktriangleleft P)$$

$$T\text{-}axiom \qquad \vdash_{\mathcal{P}} P \Rightarrow \Diamond P$$

$$\vdash_{\mathcal{P}} \Box P \Rightarrow P$$

$\wedge$ only partially distributes, since the left-hand sides specify the splitting of the data into a context and subdata, whereas the right-hand sides specify two splittings that need not coincide. Although the T-axiom of ML is derivable, the 4-axiom stating that $\Diamond\Diamond P \Rightarrow \Diamond P$ is not; see the *Step* model in Example 2.6 for a counter-model. In Zarafty's thesis [16], he studies CL extended with a context connective corresponding to context composition and the two right adjoints. The 4-axiom does hold in this extension, and the somewhere modality corresponds more closely to our structured-data intuition.

We now define the CL-models and satisfaction relation, which are sound and complete with respect to the CL-proof theory.

Definition 2.5 [CL-Model] A *CL-model Mod* is a tuple $(\mathcal{D}, \mathcal{C}, \mathrm{ap}, \mathbf{I})$ such that

(i) $\mathcal{D}$ and $\mathcal{C}$ are sets;

(ii) $\mathrm{ap} \subseteq (\mathcal{C} \times \mathcal{D}) \times \mathcal{D}$ is relation, called *application*: we use the notation $\mathrm{ap}(c, d_1) = d_2$ for $((c, d_1), d_2) \in \mathrm{ap}$;

(iii) $\mathbf{I} \subseteq \mathcal{C}$ acts as a left identity to ap: that is,
  - $\forall d \in \mathcal{D}, \exists i \in \mathbf{I}, d' \in \mathcal{D}.\ \mathrm{ap}(i, d) = d'$;
  - $\forall d, d' \in \mathcal{D}, \forall i \in \mathbf{I}.\ \mathrm{ap}(i, d) = d'$ implies $d = d'$.

We often call $\mathcal{D}$ the *data set* and $\mathcal{C}$ the *context set*, because of the form of our motivating examples. Of course, there are models which do not fit this structured data intuition.

Example 2.6 [Example CL-Models]

- $Mon_{\mathcal{D}} = (\mathcal{D}, \mathcal{D}, \cdot, \{e\})$ where $\mathcal{D}$ is a set with monoidal operator $\cdot : (\mathcal{D} \times \mathcal{D}) \to \mathcal{D}$ and unit $e \in \mathcal{D}$; also, $Part\_Mon_{\mathcal{D}} = (\mathcal{D}, \mathcal{D}, \cdot, \{e\})$ which is like $Mon_{\mathcal{D}}$ except that $\cdot$ is a partial monoid. These models correspond to the BL-models (definition 2.26), and CL-reasoning collapses to BL-reasoning for these models (theorem 2.29).

- $Heap$ is an example of $Part\_Mon_{\mathcal{D}}$ where $\mathcal{H} = \mathbb{N}^+ \rightharpoonup_{fin} \mathbb{N}$ is the set of finite partial functions denoting the heaps. The domain $\mathbb{N}^+ = \mathbb{N} - \{0\}$ does not include 0 as it is reserved for the nil value. Heap composition $h \cdot h'$, for heaps $h, h' \in \mathcal{H}$, is function union and is only defined when $dom(h) \cap dom(h') = \emptyset$.

- $Term_{\Sigma} = (\mathcal{T}_{\Sigma}, \mathcal{C}_{\Sigma}, ap, \{\_\})$ where $\mathcal{T}_{\Sigma}$ is the data set of terms constructed from the r-ary function symbols $f : r$ in signature Σ, $\mathcal{C}_{\Sigma}$ is the corresponding set of contexts, ap denotes the standard application of contexts to terms, and $\_$ denotes the empty context.

- $Seq_A = (\mathcal{S}_A, \mathcal{C}_A, ap, \{\_\})$ where $\mathcal{S}_A$ is the set of sequences constructed from the elements in alphabet A, $\mathcal{C}_A$ is the corresponding set of contexts, ap and $\_$ are as for $Term_{\Sigma}$; let $Mult_A$ denote the analogous CL-model constructed from multisets;

- $Tree_A = (\mathcal{T}_A, \mathcal{C}_A, ap, \{\_\})$ is an example of $Term_{\Sigma_A}$ with an additional equality relation on terms and contexts. The set of terms $\mathcal{T}_A$ is generated by the signature Σ_A constructed from the sets $\Sigma_0 = \{0\}$, $\Sigma_1 = A$ where A is the set of node labels and $\Sigma_2 = \{|\}$, where Σ_i denotes the function symbols of arity i. We use the notation $t \mid t'$ for $|(t, t')$ and $a[t]$ for $a(t)$, $a \in A$. Terms are considered modulo an equality relation generated by the axioms $0 \mid t \equiv t$, $t \mid t' \equiv t' \mid t$, $(t \mid t') \mid t'' \equiv t \mid (t' \mid t'')$, and closed by the obvious structural rules for the function symbols. The set of contexts $\mathcal{C}_A$ is constructed similarly with an analogous equality relation on contexts.

- Loc_Tree_N is similar to $Tree_N$ with $\Sigma_0 = \{0\}$, $\Sigma_1 = N$ and $\Sigma_2 = \{|\}$ as before. This time however the set N denotes a set of *unique* node identifiers, and $|$ is therefore partial like the partial heap composition. We shall use this model for illustrating our ideas about tree update.

- $Rel_{\mathcal{D}} = (\mathcal{D}, \mathcal{P}(\mathcal{D} \times \mathcal{D}), \mathrm{ap}, \{i\})$ where $\mathcal{D}$ is an arbitrary set, $\mathcal{P}(\mathcal{D} \times \mathcal{D})$ denotes the set of binary relations on $\mathcal{D}$, ap is relational application, and i is the identity relation; also $Fun_{\mathcal{D}} = (\mathcal{D}, \mathcal{D} \to \mathcal{D}, \mathrm{ap}, \{i\})$ where $\mathcal{D} \to \mathcal{D}$ is the set of total functions from $\mathcal{D}$ to $\mathcal{D}$, ap is function application and i is the identity function.

- $Step = (\mathbb{N}, \{0, 1\}, +, \{0\})$ where the data set is the natural numbers, the context set is $\{0, 1\}$, application is normal addition, and the unit is zero. This model demonstrates that the 4-axiom of ML is not derivable, since the number 2 satisfies $\Diamond\Diamond 0$ but not $\Diamond 0$.

- $\mathcal{M}_1 + \mathcal{M}_2 = (\mathcal{D}_1 \cup \mathcal{D}_2, \mathcal{C}_1 \cup \mathcal{C}_2, \mathrm{ap}, I_1 \cup I_2)$ where $\mathcal{M}_1 = (\mathcal{D}_1, \mathcal{C}_1, \mathrm{ap}_1, I_1)$ and $\mathcal{M}_2 = (\mathcal{D}_2, \mathcal{C}_2, \mathrm{ap}_2, I_2)$ are CL-models and, for arbitrary $\mathsf{d}_i \in \mathcal{D}_i, \mathsf{c}_i \in \mathcal{C}_i$, $\mathrm{ap}(\mathsf{c}_i, \mathsf{d}_j) = \mathrm{ap}_i(\mathsf{c}_i, \mathsf{d}_j)$ if $i = j$ and is undefined otherwise.

Definition 2.7 [CL-Satisfaction Relation] Given a CL-model $Mod = (\mathcal{D}, \mathcal{C}, \mathrm{ap}, \mathbf{I})$, the CL-satisfaction relation $\vDash_{CL}$ consists of two relations $Mod, d \vDash_{\mathcal{P}} P$ and $Mod, c \vDash_{\mathcal{K}} K$ where $d \in \mathcal{D}$ and $c \in \mathcal{C}$. The two relations are defined by induction on the struc-

ture of the formulae. The cases for the boolean additive connectives are standard; the cases for the structural connectives are:

$$Mod, d \vDash_{\mathcal{P}} K(P) \quad \text{iff} \quad \exists c \in \mathcal{C}, d' \in \mathcal{D}. \ \text{ap}(c, d') = d \wedge Mod, c \vDash_{\mathcal{K}} K \wedge Mod, d' \vDash_{\mathcal{P}} P$$

$$Mod, d \vDash_{\mathcal{P}} K \triangleleft P \quad \text{iff} \quad \forall c \in \mathcal{C}, d' \in \mathcal{D}. \ Mod, c \vDash_{\mathcal{K}} K \wedge \ \text{ap}(c, d) = d' \Rightarrow Mod, d' \vDash_{\mathcal{P}} P$$

$$Mod, c \vDash_{\mathcal{K}} I \quad \text{iff} \quad c \in \mathbf{I}$$

$$Mod, c \vDash_{\mathcal{K}} P_1 \triangleright P_2 \quad \text{iff} \quad \forall d, d' \in \mathcal{D}. \ Mod, d \vDash_{\mathcal{P}} P_1 \wedge \ \text{ap}(c, d) = d' \Rightarrow Mod, d' \vDash_{\mathcal{P}} P_2$$

We sometimes omit the subscripts $\mathcal{P}$ and $\mathcal{K}$. In section 2.3, we explore the satisfaction relations for Seq_A, $Mult_A$ and $Tree_A$ in depth, giving many examples to illustrate the expressivity of CL-reasoning on these models.

Definition 2.8 *A formula P or K is* valid *for a given model $\mathcal{M} = (\mathcal{D}, \mathcal{C}, ap, I)$, written $\mathcal{M} \vDash_{\mathcal{P}} P$ or $\mathcal{M} \vDash_{\mathcal{C}} K$, if it is satisfied by all data or contexts in the model:*

$$\mathcal{M} \vDash_{\mathcal{P}} P \triangleq \forall \mathsf{d} \in \mathcal{D}. \ \mathcal{M}, \mathsf{d} \vDash_{\mathcal{P}} P$$

$$\mathcal{M} \vDash_{\mathcal{C}} K \triangleq \forall \mathsf{c} \in \mathcal{C}. \ \mathcal{M}, \mathsf{c} \vDash_{\mathcal{C}} K$$

Theorem 2.9 (CL-soundness and completeness) *The* CL*-proof theory is sound and complete with respect to the* CL*-satisfaction relation:*

$$\text{true} \vdash_{\mathcal{P}} P \Leftrightarrow \forall \mathcal{M}.(\mathcal{M} \vDash_{\mathcal{P}} P)$$

$$\text{True} \vdash_{\mathcal{C}} K \Leftrightarrow \forall \mathcal{M}.(\mathcal{M} \vDash_{\mathcal{C}} K)$$

We prove completeness in [2] via an interpretation of the structural connectives of CL as modalities in modal logic (ML). This interpretation is not straight-forward, as it uses the negation duals of the structural adjoints as the fundamental connectives. We present additional axioms for these CL-modalities to give a precise correspondence between the original CL-presentation and its ML-interpretation. These axioms are well-behaved, in that they satisfy the conditions necessary for us to apply a general completeness result about ML (Sahlqvist's theorem). We thus prove that the CL-proof theory is sound and complete with respect to the set of CL-models. We have analogous results for BL. This work follows previous work by Calcagno and Yang, who proved completeness for BL and CL from first principles in unpublished work. They also extend their result to show completeness for CL-models where application is a function; their technique breaks for BL due to associativity. The extension to the functional case is given by Zarfaty in [16]. We use the relational definition here, to relate certain CL-models with the BL-models (theorem 2.29).

2.2   Context Logic with Zero

Notice that many of the CL-models given in Example 2.6 describe structured data with a natural element corresponding to empty data. We extend CL with a zero

formula 0 and additional axioms, to capture natural properties of empty data. The resulting logic, denoted CL_0, has interesting logical structure and allows for a precise correspondence with BL (section 2.2.2).

Definition 2.10 (CL_0-**Formulae**) *The set of* CL_0-*formulae consists of data and context formulae as in Definition 2.1 plus an additional data formula* 0, *called the zero formula.*

Definition 2.11 (CL_0-**Proof theory**) *The* CL_0-*proof theory extends Definition 2.2 with the zero axioms:*

$$\text{True} \vdash_{\mathcal{C}} 0 \blacktriangleright \text{true} \qquad \text{true} \vdash_{\mathcal{P}} \text{True}(0) \qquad 0 \blacktriangleright P \vdash_{\mathcal{C}} 0 \rhd P \qquad 0 \rhd 0 \vdash_{\mathcal{C}} I$$

These zero axioms state intuitive properties about data elements regarded as empty data. Their converses are all derivable. The first axiom specifies a totality condition that every context can be applied to empty data. Recall that we have example models where the application is not total, so this property is not derivable. The second axiom is a surjectivity condition specifying that all data can be split into a context and empty data. The third axiom states that all the zero elements behave in a similar fashion when applied to a context. The fourth axiom identifies the context $0 \rhd 0$, returning empty data when applied to empty data, with the empty context I. Whilst we believe these zero axioms describe important properties of empty data, we do not know whether we have captured the full essence of empty data. For example, another sensible property would be $0 \vdash_{\mathcal{D}} \neg(True(\neg 0))$ specifying that empty data cannot be split into a context and non-empty data. We know that this property is not derivable, since its converse holds in the CL_0-model $Mon_{\mathcal{D}_a}$ given just after Example 2.13. We have chosen to work with the zero axioms given, as they are already enough to prove some interesting properties.

Definition 2.12 [CL_0-Model] A CL_0-model is a tuple $(\mathcal{C}, \mathcal{D}, ap, \mathbf{I}, \mathbf{0})$ where

(i) $(\mathcal{C}, \mathcal{D}, ap, \mathbf{I})$ is a CL-model;

(ii) $\mathbf{0} \subseteq \mathcal{D}$;

(iii) the projection $p : \mathcal{C} \to \mathcal{D}$ defined by

$$p(c) = d \Leftrightarrow \exists\, o \in \mathbf{0}.\ ap(c, o) = d$$

   is a total surjective function;

(iv) $\forall c \in \mathcal{C}, \forall o \in \mathbf{0}.\ p(c) = o \Rightarrow c \in \mathbf{I}$.

The projection function p maps every element in $\mathcal{C}$ to a unique element in $\mathcal{D}$, by applying it to a zero element. The projection function is surjective, meaning that every data element has a zero element as a sub-element. Condition iv places a strong connection between $\mathbf{I}$ and $\mathbf{0}$: from Definition 2.5, we have $p(i) \in \mathbf{0}$ for all $i \in \mathbf{I}$; from condition iv, we also have $p^{-1}(0) \subseteq \mathbf{I}$ for all $o \in \mathbf{0}$.

Example 2.13 [Example CL_0-models] The following extensions to the CL-models given in Example 2.6 are all CL_0-models.

- $Mon_\mathcal{D} = (\mathcal{D}, \mathcal{D}, \cdot, \{e\}, \{e\})$; similarly for $Part\_Mon_\mathcal{D}$.

- $Heap$ with the empty function as the zero element.

- Seq_A, $Mult_A$, $Tree_A$ and Loc_Tree_N with the empty sequence, empty multiset and the empty tree as the appropriate zero elements.

- $Mod_1 + Mod_2 = (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{D}_1 \cup \mathcal{D}_2, \mathsf{ap}, \mathsf{I}_1 \cup \mathsf{I}_2, \mathbf{0}_1 \cup \mathbf{0}_2)$ where $\mathcal{M}_1 = (\mathcal{C}_1, \mathcal{D}_1, \mathsf{ap}_1, \mathsf{I}_1, \mathbf{0}_1)$ and $\mathcal{M}_2 = (\mathcal{C}_2, \mathcal{D}_2, \mathsf{ap}_2, \mathsf{I}_2, \mathbf{0}_2)$ are CL_0-models and ap is defined in Example 2.6.

The CL_0-model $Mon_{\mathcal{D}_\mathsf{a}}$ with monoid $\mathcal{D}_a = \{a, e\}$, and $a \cdot a = e$ illustrates that we have not captured all our intuition regarding the behaviour of empty data. For example, it shows that the entailment $0 \vdash_\mathcal{P} \neg(\mathrm{True}(\neg 0))$ is not derivable using the CL_0-proof theory. Now consider the derived formulae $0^+ \triangleq \mathrm{true}$, $(n+1)^+ \triangleq (\neg I)(n^+)$ and $n \triangleq n^+ \wedge \neg(n+1)^+$ for all $n \in \mathbb{N}$. In some CL_0-models, such as the sequence model, these formulae capture the size of data: for example, formula n^+ specifies sequences with at least n elements, and formula n specifies sequences with precisely n elements. However, the $Mon_{\mathcal{D}_\mathsf{a}}$-model shows that this sort of analysis is not always possible.

The CL-model $Step$ does not have a zero set, since there is no surjective function from contexts (0 and 1) to data ($\mathbb{N}$). Also, the CL-model $Term_\Sigma$ with signature set $\Sigma = \{f : 1, g_1 : 0, g_2 : 0\}$ does not have a zero set. Assume for contradiction that it has zero set $\mathbf{0}$: if $\mathbf{0}$ does not contain g_2 (or g_1), then g_2 cannot be in the image of p contradicting surjectivity; however g_1 and g_2 cannot both be in $\mathbf{0}$ since then $p(\_) = g_1$ and $p(\_) = g_2$, contradicting well-formedness of function p. Notice that $Term_{\Sigma'}$ where $\Sigma' = \{f : 1, g : 0\}$ is a CL_0-model with zero set $\{g : 0\}$. The CL-model $Rel_\mathcal{D}$ has no zero set since the everywhere undefined relation contradicts totality of the projection function. Also $Fun_\mathcal{D}$ has no zero set: the only possible choice would be a singleton $\{d\}$ for the projection to be a function, but it contradicts condition iv since the identity is not the only function mapping d to itself.

Definition 2.14 (CL_0-**Satisfaction Relation**) *Let $Mod_0 = (\mathcal{D}, \mathcal{C}, ap, I, \mathbf{0})$ be an arbitrary CL_0-model. The CL_0-satisfaction relation extends the relations $Mod_0, d \vDash_\mathcal{P} P$ and $Mod_0, c \vDash_\mathcal{K} P$ given in Definition 2.7 with*

$$Mod_0, d \vDash_\mathcal{P} 0 \quad iff \quad d \in \mathbf{0}$$

Theorem 2.15 (CL_0-**Soundness and Completeness**) *The CL_0-proof theory is sound and complete with respect to the CL_0-satisfaction relation.*

As for Thm. 2.9, completeness can either by proved from first principles, or by using the ML-interpretation of CL_0 and appealing to Salqvist's theorem.

2.2.1 Derived formulae

We explore some derived CL_0-formulae. First, we derive a binary $*$-connective on data and its accompanying right adjoints, which are generalised version of $*$ and $-\!*$ of BL and SL. We also show that a natural embedding relation to the projection function of CL_0-models suggests embedding/projection formulae with interesting

logical structure.

∗-formulae

Definition 2.16 (Derived ∗-formulae) *We derive the following* CL_0-*formulae:*

$$P_1 * P_2 \triangleq (0 \triangleright P_1)(P_2)$$
$$P_1 \mathbin{*\!\!-} P_3 \triangleq (0 \triangleright P_1) \triangleleft P_3$$
$$P_2 \mathbin{-\!\!*} P_3 \triangleq \neg(\neg(P_2 \triangleright P_3)(0))$$

We assume the binding precedence $\neg$, $*$, $\wedge$, $\vee$, $\{\triangleleft, \triangleright, \blacktriangleleft, \blacktriangleright, \mathbin{*\!\!-}, \mathbin{-\!\!*}\}$. The derived formula $P_1 * P_2$ specifies that the given data can be split into subdata satisfying property P_2 and a context with the property that, when a zero element is put in the hole, satisfies property P_1. For example, the formula $\neg 0 * \neg 0$ specifies that the given data can be split into two disjoint, non-empty parts. The $*$-connective is neither commutative nor associative in general. It is not associative or commutative in the sequence or tree model for example, but is in the heap model. We therefore have two right adjoints. The first adjoint $P_1 \mathbin{*\!\!-} P_3$ is straightforward. It states that, whenever a context applied to a zero element satisfies P_1, then the context applied to the given data element satisfies P_3. The second adjoint is more complicated. It states that, whenever data satisfying P_2 replaces empty subdata of the given data, then the resulting data satisfies P_3: for trees, this data satisfying P_2 can be inserted at the leaves or by any node; for sequences, this data can be inserted at any point in the sequence; for heaps, the data just extends the heap provided there is no clash of heap addresses.

Lemma 2.17 (Properties of ∗-formulae)

(i) $P \mathbin{*\!\!-} \_$ *and* $P \mathbin{-\!\!*} \_$ *are the right adjoints of* $P * \_$ *and* $\_ * P$ *respectively: that is,*

$$\frac{P_1 * P_2 \vdash_{\mathcal{P}} P_3}{P_1 \vdash_{\mathcal{P}} P_2 \mathbin{-\!\!*} P_3} \qquad \frac{P_1 * P_2 \vdash_{\mathcal{P}} P_3}{P_2 \vdash_{\mathcal{P}} P_1 \mathbin{*\!\!-} P_3}$$

(ii) *The zero formula* 0 *is the right and left unit of* $*$*: that is,* $P * 0 \Leftrightarrow P \Leftrightarrow 0 * P$*.*

These $*$-formulae relate closely to a derivable $*$-relation in the CL_0-model.

Definition 2.18 (Derived ∗-relation) *The relation* $* \subseteq (\mathcal{D} \times \mathcal{D}) \times \mathcal{D}$ *is defined by*

$$* = \{((\mathsf{d}_1, \mathsf{d}_2), \mathsf{d}_3) \mid \exists \mathsf{c} \in \mathcal{C}, \mathsf{o} \in 0.\ \mathsf{d}_1 = ap(\mathsf{c}, \mathsf{o}) \wedge \mathsf{d}_3 = ap(\mathsf{c}, \mathsf{d}_2)\}$$

We let $\mathsf{d}_1 * \mathsf{d}_2$ *denote the set* $\{\mathsf{d}_3 \mid ((\mathsf{d}_1, \mathsf{d}_2), \mathsf{d}_3) \in *\}$*.*

With CL_0-models $Mon_{\mathcal{D}}$ and $Part\_Mon_{\mathcal{D}}$, the relation $*$ is a (partial) function and corresponds to $\cdot$. With the model $Tree_A$, the relation $*$ is more complicated with $t_1 * t_2$ describing the set of trees obtained by inserting t_2 into an arbitrary location inside t_1: for example, $(\mathsf{a}_1[0] \mid \mathsf{a}_2[0]) * \mathsf{b}[0] = \{\mathsf{a}_1[\mathsf{b}[0]] \mid \mathsf{a}_2[0],\ \mathsf{a}_1[0] \mid \mathsf{a}_2[\mathsf{b}[0]],\ \mathsf{a}_1[0] \mid \mathsf{a}_2[0] \mid \mathsf{b}[0]\}$.

Lemma 2.19 (CL$_0$-**satisfaction for** $*$-**formulae**) *Given* CL$_0$-*model Mod, there is a direct connection between the $*$-formulae and the $*$-connective on data given by:*

$$\mathcal{M}, \mathsf{d} \vDash_{\mathcal{P}} P_1 * P_2 \;\Leftrightarrow \exists \mathsf{d}_1, \mathsf{d}_2 \in \mathcal{D}. (\mathsf{d} \in \mathsf{d}_1 * \mathsf{d}_2 \wedge \mathcal{M}, \mathsf{d}_1 \vDash_{\mathcal{P}} P_1 \wedge \mathcal{M}, \mathsf{d}_2 \vDash_{\mathcal{P}} P_2)$$

$$\mathcal{M}, \mathsf{d} \vDash_{\mathcal{P}} P_1 \ast\!\!- P_3 \Leftrightarrow \forall \mathsf{d}_1, \mathsf{d}_3 \in \mathcal{D}. (\mathsf{d}_3 \in \mathsf{d} * \mathsf{d}_1 \wedge \mathcal{M}, \mathsf{d}_1 \vDash_{\mathcal{P}} P_1 \Rightarrow \mathcal{M}, \mathsf{d}_3 \vDash_{\mathcal{P}} P_3)$$

$$\mathcal{M}, \mathsf{d} \vDash_{\mathcal{P}} P_2 -\!\!\ast P_3 \Leftrightarrow \forall \mathsf{d}_2, \mathsf{d}_3 \in \mathcal{D}. (\mathsf{d}_3 \in \mathsf{d}_2 * \mathsf{d} \wedge \mathcal{M}, \mathsf{d}_2 \vDash_{\mathcal{P}} P_2 \Rightarrow \mathcal{M}, \mathsf{d}_3 \vDash_{\mathcal{P}} P_3)$$

e, p-formulae

We motivate our study of derived projection and embedding formulae by first showing that there is a natural embedding relation in the CL$_0$-models.

Definition 2.20 [CL$_0$-embedding relation] Given CL$_0$-model $Mod_0 = (\mathcal{C}, \mathcal{D}, ap, \mathbf{I}, \mathbf{0})$, the embedding relation $e : \mathcal{D} \times \mathcal{C}$ is defined by

$$e(\mathsf{d}, \mathsf{c}) \text{ if and only if } p(\mathsf{c}) = \mathsf{d}.$$

We write $e(\mathsf{d})$ for $\{\mathsf{c} \in \mathcal{K} : e(\mathsf{d}, \mathsf{c})\}$, denoting the set of contexts which give d when applied to a zero element.

The relation e is not necessarily a function: for example, in the tree model $e(b[0]) = \{b[\_], \; b[0] \,|\, \_\}$. The pair (e, p) is an embedding-projection pair: that is $\forall d \in \mathcal{D}. \{p(c) \mid c \in e(d)\} = \{d\}$. They also give an elegant connection between I and 0: $p(i) \subseteq \mathbf{0}$ for all $i \in \mathbf{I}$ and $e(o) \subseteq \mathbf{I}$ for all $o \in \mathbf{0}$. Since this (e,p)-pair is such a natural structure in the models, we explore the corresponding CL$_0$-formulae.

Definition 2.21 (Derived e, p-formulae) *We define the following derived* CL$_0$-*formulae:*

> *projection formula* $\quad K^p \triangleq K(0)$
>
> *embedding formula* $\quad P^e \triangleq 0 \triangleright P$

The data formula K^p specifies that the given data is a natural projection of a context satisfying property K with a zero element put in the hole. The context formula P^e specifies that a given context satisfies property P whenever a zero element is put in the hole.

Lemma 2.22 (CL$_0$-**satisfaction for** e, p-**formulae**) *Given* CL$_0$-*model Mod, the connection between the e, p-formulae and the embedding-projection pair (e, p) on data is given by*

$$\mathcal{M}, \mathsf{d} \vDash_{\mathcal{P}} K^p \Leftrightarrow \exists \mathsf{c} \in \mathcal{C}. \; p(\mathsf{c}) = \mathsf{d} \wedge \mathcal{M}, \mathsf{c} \vDash_{\mathcal{C}} K$$

$$\mathcal{M}, \mathsf{c} \vDash_{\mathcal{C}} P^e \;\Leftrightarrow \forall \mathsf{d} \in \mathcal{D}. \; \mathsf{c} \in e(\mathsf{d}) \Rightarrow \mathcal{M}, \mathsf{d} \vDash_{\mathcal{P}} P$$

Lemma 2.23 (Properties of e, p-formulae) *The following entailments are derivable:*

$$P \vdash_D P^{ep}$$
$$\neg(P^e) \dashv\vdash_D (\neg P)^e$$
$$0^e \vdash_D I$$

The first entailment follows from the second and third axioms given in Definition 2.11, and corresponds to (e, p) being an embedding-projection pair. The second entailments specify that negation distributes over $(\_)^e$, or equivalently that $(\_)^e$ has a right adjoint given by $\neg((\neg\_)^p)$. The third entailment specifies that $(\_)^e$ lifts 0 to I, and coincides with the fourth axiom in Definition 2.11.

Lemma 2.24 *If we replace the zero axioms given in Definition 2.11 by the entailments in lemma 2.23, then the zero axioms are derivable.*

These results suggest that our initial choice of zero axioms was a natural choice. They also indicate that we do not yet understand the full significance of the embedding-projection formulae. In [16], Zarfaty explores this further, by suggesting an alternative presentation of CL_0 consisting of context composition, the accompanying adjoints and these embedding/projection formulae.

2.2.2 Comparison with Bunched Logic

We also present (a variant of) BL [11], its models and satisfaction relation, and compare it to CL. We use the notation $\circ$ and $\multimap$, instead of the standard $*$ and $\twoheadrightarrow$ for the multiplicative conjunction and its adjoint. We reserve the notation $*$ and $\twoheadrightarrow$ for our generalised versions given in Definition 2.16. Our variation of standard BL does not require $\circ$ to be commutative, since one of our key example models is sequences where $\circ$ denotes concatenation.

Definition 2.25 [BL-Formulae] The set of BL-formulae $\mathcal{P}_{BL}$ is defined by:

$$P ::= 0 \mid P \circ P \mid P \multimapinv P \mid P \multimap P \qquad \text{structural formulae}$$
$$P \vee P \mid \neg P \mid \text{false} \qquad\qquad \text{boolean additive formulae}$$

The key formulae are the structural formulae 0, $P_1 \circ P_2$, $P_1 \multimapinv P_2$ and $P_1 \multimap P_2$. The *zero* formula 0 specifies empty data. The *composition* formula splits the given data into two parts: the first satisfying P_1 and the second P_2. Unlike the original BL, we have two right adjoints, due to $\circ$ being non-commutative: $P_1 \multimapinv P_2$ specifies that, *whenever* some data satisfying P_1 is placed to the left of the given data, then the result satisfies P_2; the other adjoint $P_1 \multimap P_2$ places data to the right. This distinction has no effect in the heap model, but is important in the sequence model. As in CL, we define the negation duals of the adjoints as $P_1 \multimapdotinv P_2 \triangleq \neg(P_1 \multimap \neg P_2)$ and $P_1 \multimapdot P_2 \triangleq \neg(P_1 \multimapinv \neg P_2)$.

Definition 2.26 [BL-Model] A *BL-model Mod* is a tuple $(\mathcal{D}, \cdot, \mathbf{e})$ such that

(i) $\mathcal{D}$ is a set;

(ii) $\cdot \subseteq (\mathcal{D} \times \mathcal{D}) \times \mathcal{D}$ is an associative relation: we use the notation $\cdot(\mathsf{d}_1, \mathsf{d}_2) = \mathsf{d}_3$ for $((\mathsf{d}_1, \mathsf{d}_2), \mathsf{d}_3) \in \cdot$;

(iii) $\mathbf{e} \subseteq \mathcal{D}$ acts as a left and right identity to $\cdot$: that is,
- $\forall \mathsf{d} \in \mathcal{D}, \exists \mathsf{e} \in \mathbf{e}, \mathsf{d}' \in \mathcal{D}. \quad \cdot(\mathsf{e}, \mathsf{d}) = \mathsf{d}'$
- $\forall \mathsf{d} \in \mathcal{D}, \exists \mathsf{e} \in \mathbf{e}, \mathsf{d}' \in \mathcal{D}. \quad \cdot(\mathsf{d}, \mathsf{e}) = \mathsf{d}'$
- $\forall \mathsf{d}, \mathsf{d}' \in \mathcal{D}, \forall \mathsf{e} \in \mathbf{e}. \quad \cdot(\mathsf{e}, \mathsf{d}) = \mathsf{d}'$ or $\cdot(\mathsf{d}, \mathsf{e}) = \mathsf{d}'$ implies $\mathsf{d} = \mathsf{d}'$.

Notice that any BL-model $\mathcal{M} = (\mathcal{D}, \cdot, \mathbf{e})$ can be lifted to a CL_0-model $\mathcal{M}_{\mathrm{BL}} = (\mathcal{D}, \mathcal{D}, \cdot, \mathbf{e}, \mathbf{e})$ with the derived $*$-relation 2.18 coinciding with $\cdot$. In fact, we shall see that the CL_0-satisfaction relation collapses to the BL-satisfaction relation for such models (theorem 2.29). We highlight specific BL-models for heaps, sequences and trees, since we will use them in this paper. Contrast these BL-models with the analogous CL-models given in Example 2.6, which also emphasise the context structure.

Example 2.27 [Some BL-models]
- $Seq_A = (\mathcal{S}_A, \cdot, \{0\})$ where $\mathcal{D}_A$ is the set of sequences constructed from the elements in set A, $\cdot$ is sequence concatenation, and 0 is the empty sequence; let $Mult_A$ denote the analogous CL_0-model constructed from multisets.
- $Tree_A = (\mathcal{T}_A, |, \{0\})$ where $\mathcal{T}_A$ is the set of trees in Example 2.6, $|$ is horizontal tree composition, and 0 is the empty tree.
- $Heap = (\mathcal{D}, \cdot, \{e\})$ where $\mathcal{D}, \cdot$ and e are as in Example 2.6.

Notice the difference between the CL_0-model of sequences Seq_A and the CL-lifting of the BL-model Seq_A: in the CL_0-model, the contexts can be regarded as having the form $s_1 \cdot \_ \cdot s_2$ and application replaces the hole $\_$ by a sequence; in the CL_0-lifting of BL-model Seq_A, the contexts are an isomorphic copy of the sequence set and application corresponds to concatenation of sequences. A similar difference occurs with trees. By contrast, the CL_0-model $Mult_A$ corresponds to the CL_0-lifting of the BL-model $Mult_A$ due to the commutativity of the multiset union. This similarity also occurs with heaps.

Definition 2.28 [BL-Satisfaction Relation] Given a BL-model $Mod = (\mathcal{D}, \cdot, \mathbf{e})$, the BL-satisfaction relation is of the form $Mod, d \vDash P$ where $d \in \mathcal{D}$ and $P \in \mathcal{P}_{\mathrm{BL}}$. As before, it is defined by induction on the structure of formulae. The structural connectives are given by:

$$Mod, \mathsf{d} \vDash_{\mathrm{BL}} 0 \qquad \text{iff} \quad \mathsf{d} \in \mathbf{e}$$

$$Mod, \mathsf{d} \vDash_{\mathrm{BL}} P_1 \circ P_2 \quad \text{iff} \quad \exists \mathsf{d}_1, \mathsf{d}_2 \in \mathcal{D}. \ \cdot(\mathsf{d}_1, \mathsf{d}_2) = \mathsf{d} \wedge Mod, \mathsf{d}_1 \vDash_{\mathrm{BL}} P_1 \wedge Mod, \mathsf{d}_2 \vDash_{\mathrm{BL}} P_2$$

$$Mod, \mathsf{d} \vDash_{\mathrm{BL}} P_1 \circ\!\!- P_3 \quad \text{iff} \quad \forall \mathsf{d}_1, \mathsf{d}_3 \in \mathcal{D}. \ \cdot(\mathsf{d}, \mathsf{d}_1) = \mathsf{d}_3 \wedge Mod, \mathsf{d}_1 \vDash_{\mathrm{BL}} P_1 \Rightarrow Mod, \mathsf{d}_3 \vDash_{\mathrm{BL}} P_3$$

$$Mod, \mathsf{d} \vDash_{\mathrm{BL}} P_2 \multimap P_3 \quad \text{iff} \quad \forall \mathsf{d}_2, \mathsf{d}_3 \in \mathcal{D}. \ \cdot(\mathsf{d}_2, \mathsf{d}) = \mathsf{d}_3 \wedge Mod, \mathsf{d}_2 \vDash_{\mathrm{BL}} P_2 \Rightarrow Mod, \mathsf{d}_3 \vDash_{\mathrm{BL}} P_3$$

We assume that $\circ$, $\circ\!\!-$ and $\multimap$ have the same binding precedence as $*$, $\ast\!\!-$ and $-\!\ast$.

The Hilbert-style BL-proof theory consists of analogous rules to those given for the CL-proof theory (Definition 2.2), with an additional axiom for the associativity of $\circ$. As for CL, we can prove a completeness result for BL, either from first princi-

ples, or by interpreting the structural connectives as ML-modalities and appealing to Sahlqvist's theorem [2]. We do not know how to extend this result to BL-models restricted to those where $\cdot$ is a function. This is due to the associativity of $\cdot$, which breaks our construction of a functional model from a relational one.

Theorem 2.29 (Collapse to BL) *Given BL-model $Mod = (\mathcal{D}, \cdot, \mathbf{e})$ and corresponding CL_0-model $Mod_{\mathrm{BL}} = (\mathcal{D}, \mathcal{D}, \cdot, \mathbf{e}, \mathbf{e})$, we may define translations $|\_|_{\mathcal{P}} : \mathcal{P} \to \mathcal{P}_{\mathrm{BL}}$ and $|\_|_{\mathcal{K}} : \mathcal{K} \to \mathcal{P}_{\mathrm{BL}}$ from CL_0-formulae to BL-formulae, and a translation $|\_|_{\mathrm{BL}} : \mathcal{P}_{\mathrm{BL}} \to \mathcal{P}$ from BL-formulae to CL-data formulae, such that*

- *for $\mathsf{d} \in \mathcal{D}$, $P \in \mathcal{P}$ and $K \in \mathcal{K}$,*

$$Mod_{\mathrm{CL}}, \mathsf{d} \vDash_{\mathcal{P}} P \Leftrightarrow Mod, \mathsf{d} \vDash_{\mathrm{BL}} |P|_{\mathcal{P}}$$
$$Mod_{\mathrm{CL}}, \mathsf{d} \vDash_{\mathcal{K}} K \Leftrightarrow Mod, \mathsf{d} \vDash_{\mathrm{BL}} |K|_{\mathcal{K}}$$

- *for $\mathsf{d} \in \mathcal{D}$ and $P \in \mathcal{P}_{\mathrm{BL}}$,*

$$Mod, \mathsf{d} \vDash_{\mathrm{BL}} P \Leftrightarrow Mod_{\mathrm{CL}}, \mathsf{d} \vDash_{\mathcal{P}} |P|_{\mathrm{BL}}$$

Proof. The translations are defined by induction on the structure of the formulae. The cases for the additive connectives follow the structure of the connectives; the cases for the structural connectives are:
the translations from CL_0-formulae to BL-formulae:

$$|0|_{\mathcal{P}} = 0 \qquad\qquad |I|_{\mathcal{K}} = 0$$
$$|K(P)| = |K|_{\mathcal{K}} \circ |P|_{\mathcal{P}} \qquad |P \rhd Q|_{\mathcal{K}} = |P|_{\mathcal{P}} \multimap |Q|_{\mathcal{P}}$$
$$|K \lhd P|_{\mathcal{P}} = |K|_{\mathcal{K}} \multimapinv |P|_{\mathcal{P}}$$

the translation from BL-formulae to CL-data formulae:

$$|0|_{\mathrm{BL}} = 0$$
$$|P \circ Q|_{\mathrm{BL}} = |P|_{\mathrm{BL}} * |Q|_{\mathrm{BL}}$$
$$|P \multimapinv Q|_{\mathrm{BL}} = |P|_{\mathrm{BL}} \mathbin{\ast\mkern-8mu-} |Q|_{\mathrm{BL}}$$
$$|P \multimap Q|_{\mathrm{BL}} = |P|_{\mathrm{BL}} \mathbin{-\mkern-8mu\ast} |Q|_{\mathrm{BL}}$$

The proof follows by a simple induction on the structure of formulae. $\square$

Recall that the CL_0-models for multisets and heaps are the same as the CL_0-liftings of their analogous BL-models. This theorem shows that the CL_0-reasoning and the BL-reasoning coincide for these models.

2.3   Applications of CL

We study four applications of CL-reasoning to sequences, multisets, trees and terms: sequences provide a simple example to illustrate that CL_0-reasoning and BL-

reasoning is different; multisets provide an example where CL_0- and BL-reasoning is the same; trees provide a more substantial example where the reasoning is different; and terms provide an example where BL-reasoning is not possible. In each case, the application involves extending the CL-formulae with formulae for analysing the specific structure of the data and contexts arising from the model. Here, we work with specific constants associated with the model; in section 3, we work with variables and quantification.

2.3.1   Sequences

CL for sequences generated by alphabet A is CL_0 extended by specific connectives for specifically analysing the CL_0–model Seq_A presented in Example 2.6. The additional connectives specify the one-element sequence $a \in A$, and analyse additional structure of the sequence contexts. We write $s_1 \cdot s_2$ to denote the concatenation of two sequences s_1 and s_2, and $s_1 \cdot \_ \cdot s_2$ to denote a context with the context hole $\_$ between the sequences s_1 and s_2.

Definition 2.30 [CL for Seq_A] CL applied to the sequence model Seq_A, denoted CL_{Seq_A}, consists of CL_{Seq_A}-formulae constructed by extending the CL_0-formulae defined inductively by the grammars in definition 2.14 with the following additional cases:

context formulae

$$K ::= P \bowtie P \quad \text{specific context formulae}$$

data formulae

$$P ::= a \quad \text{specific data formulae } a \in A$$

The CL_{Seq_A}-satisfaction relation extends the CL_0-satisfaction relation (definition 2.14) with the additional cases:

$$Seq_A, c \vDash_{\mathcal{K}} P_1 \bowtie P_2 \quad \text{iff} \quad \exists s_1, s_2 \in \mathcal{S}. \ c = s_1 \cdot \_ \cdot s_2 \wedge Seq_A, s_1 \vDash_{\mathcal{P}} P_1 \wedge Seq_A, s_2 \vDash_{\mathcal{P}} P_2$$

$$Seq_A, s \vDash_{\mathcal{P}} a \qquad \text{iff} \quad s = a$$

It is easy to show that logical equivalence for CL_{Seq_A} corresponds to sequence equality. The strength of this analysis is typical for this style of logical reasoning. We can derive a formula for sequence composition $P_1 \circ P_2 \triangleq (P_1 \bowtie 0)(P_2)$ which specifies that a sequence can be split into two sequences, the left one satisfying P_1 and the right one P_2. This is logically equivalent to $(0 \bowtie P_2)(P_1)$. Contrast this with the derived $*$-formula $P_1 * P_2$ which holds for a given sequence if it is possible to remove a subsequence satisfying P_2 to leave the remaining sequence satisfying P_1. We also derive the two corresponding right adjoints: the formula $P_1 \multimapinv P_2 \triangleq (P_1 \bowtie 0) \triangleleft P_2$ specifies that, whenever a sequence satisfying property P_1 is joined to the left of the given sequence, then the result satisfies P_2; similarly for $P_1 \multimap P_2 \triangleq (0 \bowtie P_1) \triangleleft P_2$. We give some additional derived formulæ which are specific to this CL_{Seq_A}-model.

Example 2.31 [CL$_{Seq_A}$- derived formulæ]

(i) $a \circ b \circ a \triangleq a \circ (b \circ a)$, the formula specifying the sequence $a \cdot b \cdot a$.

(ii) $a \circ \text{true}$, the formula specifying any sequence beginning with an a.

(iii) $a * \text{true}$, a sequence that either begins or ends with an a.

(iv) $\Diamond a$ and $\text{true} * a$ and $\text{true} \circ a \circ \text{true}$, any sequence that contains an a.

(v) $(a \multimap P) \circ b$, a sequence ending in b that satisfies P if this b is replaced by a.

(vi) $(a \multimap* P) * b$, a sequence containing a b that satisfies P if this b removed and an a added *anywhere* in the sequence.

(vii) $(a \rhd P)(b)$, a sequence containing a b that satisfies P if this b is replaced by a added *in the same place*.

(viii) $\Box(1 \Rightarrow a)$, any sequence containing just as.

(ix) $(a \circ b)^* \triangleq 0 \vee (a \circ \text{true} \wedge \text{true} \circ b \wedge \Box(2 \Rightarrow a \circ b \vee b \circ a))$, the formula specifies the Kleene star $(a \cdot b)^*$ denoting either the empty sequence or sequences with alternating as and bs, starting from a and ending in b.

By contrast with this last example (example 2.34ix), the Kleene star $(a \cdot a)^*$ is not expressible in CL$_{Seq_A}$. These two examples are key examples for illustrating the difference between regular languages and $*$-free regular languages. In unpublished work, we have recently shown that CL$_{Seq_A}$ does indeed specify the $*$-free regular languages.

We now contrast CL$_{Seq_A}$-reasoning with BL-reasoning about the BL-model Seq_A (definition 2.27).

Definition 2.32 [BL for Seq_A] BL applied to the BL-model Seq_A, denoted BL$_{Seq_A}$, consists of BL$_{Seq_A}$-formulae constructed by extending the BL-formulae defined inductively in definition 2.25 with the following additional case:

data formulae

$$P ::= a \quad \text{specific data formulae } a \in A$$

The BL$_{Seq_A}$-satisfaction relation extends the BL-satisfaction relation (definition 2.28) with the additional cases:

$$Seq_A, s \vDash a \text{ iff } s = a$$

BL$_{Seq_A}$ is clearly a sublogic of CL$_{Seq_A}$, since $\circ, \circ\!-$ and $\multimap$ are derivable. Recall that the CL$_0$-model Seq_A is not the same as the CL$_0$-lifting of the BL-model. This suggests that the reasoning will be different. However, the question of whether CL$_{Seq_A}$ is more expressive than BL$_{Seq_A}$ is subtle. Consider the CL$_{Seq_A}$-formula $(0 \rhd b \circ c)(a)$. It is logically equivalent to $a \circ b \circ c \vee b \circ a \circ c \vee b \circ c \circ a$. Now consider the CL$_{Seq_A}$-formula $(0 \rhd \text{True}(b))(a)$. It is equivalent to $\text{true} \circ b \circ \text{true} \circ a \circ \text{true} \vee \text{true} \circ a \circ \text{true} \circ b \circ \text{true}$, but it has very different structure to the previous example. In fact, we have shown that CL$_{Seq_A}$ and BL$_{Seq_A}$ are equality expressive using the standard definition of logical expressivity. However, they are not *parametrically* as expressive, as these examples suggest. We believe this parametric nature of CL is important for the modular Hoare reasoning given in section 3.

2.3.2   Multisets

The CL_0- model of multisets is the simplest model in which the CL-reasoning collapses to BL-reasoning. This collapse is due to the commutativity of multiset union. CL for multisets generated by alphabet A, denoted CL_{Mult_A}, is CL_0 extended by formulae for determining the one-element sequences $a \in A$. In this case, we do not require additional formulae for analysing the structure of the sequence contexts.

Definition 2.33 [CL for $Mult_A$] CL applied to the sequence model $Mult_A$, denoted CL_{Mult_A}, consists of CL_{Mult_A}-formulae constructed by extending the CL_0-formulae defined inductively by the grammars in definition 2.14 with the following additional data formula:

$$P ::= \mathsf{a} \quad \text{specific data formulae } \mathsf{a} \in A$$

The CL_{Mult_A}-satisfaction relation extends the CL_0-satisfaction relation (definition 2.14) with the additional a case given as for sequences (definition 2.32).

We again have a formula for data composition, this time given by $P_1 \circ P_2 \triangleq (0 \rhd P_1)(P_2)$ which specifies that a multiset can be split into two parts, one satisfying P_1 and the other P_2. This is logically equivalent to $(0 \rhd P_2)(P_1)$. We also derive the corresponding right adjoint: the formula $P_1 \multimap P_2 \triangleq (0 \rhd P_1) \lhd P_2$ specifies that, whenever a multiset satisfying property P_1 is joined to the given multiset, then the result satisfies P_2. It is logically equivalent to $(P_1 \rhd P_2)(0)$. This time we only have one right adjoint due to the commutativity of multiset union. We give some additional derived formulæ which are specific to this CL_{Mult_A}-model.

Example 2.34 [CL_{Mult_A}- derived formulæ]

 (i) $\mathsf{a} * \mathsf{b} * \mathsf{a}$, the multiset $\{\mathsf{a}, \mathsf{a}, \mathsf{b}\}$.

 (ii) $\Diamond \mathsf{a}$ and $\text{true} * \mathsf{a}$ and $\mathsf{a} * \text{true}$, any multiset that contains an a.

 (iii) $(\mathsf{a} \twoheadrightarrow P) * \mathsf{b}$ and $(\mathsf{a} \rhd P)(\mathsf{b})$, a multiset containing a b that satisfies P if a b is removed and an a added.

Definition 2.35 [BL for $Mult_A$] BL applied to the BL-multiset model $Mult_A$, denoted BL_{Mult_A}, consists of BL_{Mult_A}-formulae extended with specific data formulae $\mathsf{a} \in A$ and the BL_{Mult_A}-satisfaction relation extended in the obvious way (as in definition 2.32 for BL for Seq_A).

This time the CL_{Mult_A}-reasoning collapses to BL_{Mult_A}-reasoning, since the CL_0-model $Mult_a$ is the same as the CL_0-lifting of the BL-model $Mult_A$ and we can use the translations in definition 2.29 extended with the trivial case for $\mathsf{a} \in A$.

2.3.3   Trees

CL applied to model $Tree_A$ is CL_0 extended by specific formulae which can be interpreted in the CL_0-model $Tree_A$ presented in Example 2.6. The additional formulae specify ways of analysing tree contexts.

Definition 2.36 [CL for $Tree_A$] CL applied to the tree model $Tree_A$, denoted CL_{Tree_A}, consists of CL_{Tree_A}-formulae constructed by extending the CL_0-formulae

defined inductively by the grammars in definition 2.10 with the following additional context formulae:

$$K ::= \mathsf{a}[K] \mid K \circ P \quad \text{specific context formulae } \mathsf{a} \in A$$

The CL_{Tree_A}-satisfaction relation is defined by extending the CL_0-satisfaction relation (definition 2.14) with the additional cases:

$$Tree_A, \mathsf{c} \vDash_{\mathcal{K}} \mathsf{a}[K] \quad \text{iff } \exists \mathsf{c}' \in \mathcal{C}. \quad \mathsf{c} = \mathsf{a}[\mathsf{c}'] \wedge Tree_A, \mathsf{c}' \vDash_{\mathcal{K}} K$$

$$Tree_A, \mathsf{c} \vDash_{\mathcal{K}} K \circ P \text{ iff } \exists \mathsf{c}' \in \mathcal{C}, \mathsf{d} \in \mathcal{D}. \quad \mathsf{c} = \mathsf{c}' \mid \mathsf{d} \wedge Tree_A, \mathsf{c}' \vDash_{\mathcal{K}} K \wedge Tree_A, \mathsf{d} \vDash_{\mathcal{P}} P$$

These additional specific formulae describe two ways of analysing tree contexts: either tree contexts consist of a top node labelled a with a subcontext underneath the node, or they can be split into a context and data. We have the derived formulae $\mathsf{a}[P] \triangleq (\mathsf{a}[P \circ I])(0)$, $P_1 \circ P_2 \triangleq$ and $(P_1 \circ I)(P_2)$, with their adjoints $\hat{\mathsf{a}}[P] \triangleq \mathsf{a}[I] \triangleleft P$ and $P_1 \multimap P_2 \triangleq (P_1 \circ I) \triangleleft P_2$. Here are some other derived formulae.

Example 2.37 [CL_{Tree_A}-derived formulæ]

(i) $\mathsf{a}[0]$, the tree $\mathsf{a}[0]$.

(ii) $\mathsf{a}[\text{true}]$, a tree with root node labelled a.

(iii) $\Diamond \mathsf{a}[\text{true}]$ and $\text{true} * \mathsf{a}[\text{true}]$, a tree containing a node a.

(iv) $\mathsf{a}[0] * \text{true}$ and $(0 \triangleright \mathsf{a}[0])(\text{true})$ and $\mathsf{a}[\text{true}] \vee (\mathsf{a}[0] \circ \text{true})$, a tree with root node a and either a subforest and no siblings, or siblings and an empty subforest.

(v) $(\mathsf{a}[\text{true}] \blacktriangleright P)(0)$, a tree into which it is possible to add a subtree with root node a to obtain a tree satisfying P.

(vi) $(0 \triangleright P)(\mathsf{a}[\text{true}])$ and $P * \mathsf{a}[\text{true}]$, a tree containing a that satisfies P if the subtree at a is replaced by a 0.

We define BL for trees, corresponding to the static Ambient Logic without quantifiers.

Definition 2.38 [BL for $Tree_A$] BL applied to the BL-model $Tree_A$, denoted BL_{Tree_A}, consists of BL_{Tree_A}-formulae constructed by extending the BL-formulae defined inductively in definition 2.25 with the following additional data formulae:

$$P ::= \mathsf{a}[P] \mid \hat{\mathsf{a}}[P] \mid \Diamond P \quad \text{specific data formulae } \mathsf{a} \in A$$

The BL_{Tree_A}-satisfaction relation is defined by extending the BL-satisfaction rela-

tion (definition 2.28) with the additional cases:

$$Tree_A, \mathsf{d} \vDash \mathsf{a}[P] \text{ iff } \exists \mathsf{d}' \in \mathcal{D}. \quad \mathsf{d} = \mathsf{a}[\mathsf{d}'] \wedge Tree_A, \mathsf{d}' \vDash P$$

$$Tree_A, \mathsf{d} \vDash \hat{\mathsf{a}}[P] \text{ iff } Tree_A, \mathsf{a}[\mathsf{d}] \vDash P$$

$$Tree_A, \mathsf{d} \vDash \diamond P \quad \text{iff } (\exists \mathsf{d}', \mathsf{d}'' \in \mathcal{D}. \mathsf{d} = \mathsf{d}' \,|\, \mathsf{d}'' \wedge Tree_A, \mathsf{d}' \vDash P) \vee$$
$$(\exists \mathsf{a} \in A, \mathsf{d}', \mathsf{d}'' \in \mathcal{D}. \mathsf{d} = \mathsf{a}[\mathsf{d}'] \,|\, \mathsf{d}'' \wedge Tree_A, \mathsf{d}' \vDash \diamond P)$$

BL_{Tree_A} is a sublogic of CL_{Tree_A}. The comparison between CL_{Tree_A} and BL_{Tree_A} is subtle. Consider the CL-formula $(0 \rhd \mathsf{b}_1[\mathsf{b}_2[0]])(\mathsf{a}[\mathrm{true}])$, which specifies that we can remove a subtree with root label a to obtain a tree $\mathsf{b}_1[\mathsf{b}_2[0]]$. It corresponds to the BL-formula $\mathsf{b}_1[\mathsf{b}_2[\mathsf{a}[\mathrm{true}]]] \vee \mathsf{b}_1[\mathsf{b}_2[0] \,|\, \mathsf{a}[\mathrm{true}]] \vee (\mathsf{b}_1[\mathsf{b}_2[0]] \,|\, \mathsf{a}[\mathrm{true}])$. Now consider the CL-formula $(0 \rhd \diamond \mathsf{b}_2[\mathrm{true}])(\mathsf{a}[\mathrm{true}])$. It corresponds to the BL-formula $\diamond \mathsf{b}_2[\mathrm{true}] \wedge \diamond \mathsf{a}[\neg \diamond \mathsf{b}_2[\mathrm{true}]]$. The structure of the CL-formulae is very similar, and the implication

$$(0 \rhd \mathsf{b}_1[\mathsf{b}_2[0]])(\mathsf{a}[\mathrm{true}]) \Rightarrow (0 \rhd \diamond \mathsf{b}_2[\mathrm{true}])(\mathsf{a}[\mathrm{true}])$$

is immediate. By contrast, the structure of the BL-formulae is very different, and the corresponding implication is much less obvious. We believe that the parametric nature of our CL-reasoning is essential for our local Hoare reasoning about tree update given in section 3.

2.3.4 Terms

Terms over a signature do not decompose as a parallel composition of subterms, due to the fixed arity of function symbols. They do however decompose nicely as context/subtree pairs. The term model is therefore an interesting model for us to explore. We apply CL-reasoning to terms in the term model $Term_\Sigma$ (Example 2.6), extending CL with specific formulae which can be interpreted in the CL-model $Tree_A$. The additional formulae specify the function symbols of arity 0, and ways of analysing term contexts.

Definition 2.39 [CL for $Term_\Sigma$] CL applied to the term model $Term_\Sigma$, denoted $\mathrm{CL}_{Term_\Sigma}$, consists of $\mathrm{CL}_{Term_\Sigma}$-formulae constructed by extending the CL-formulae defined inductively by the grammars in definition 2.10 with the following additional cases:

data formulae:

$$P ::= f \quad \text{specific data formulae}, f : 0 \in \Sigma$$

context formulae

$$K ::= f(P_1, \ldots, K, \ldots, P_r) \quad \text{specific context formulae}, f : r \in \Sigma$$

The $\mathrm{CL}_{Term_\Sigma}$-satisfaction relation is defined by extending the CL-satisfaction rela-

tion (definition 2.14) with the additional cases:

$$Term_\Sigma, \mathsf{t} \vDash_\mathcal{P} \mathsf{f} \qquad\qquad\qquad \text{iff } \mathsf{t} = \mathsf{f}, \quad \mathsf{f} : 0 \in \Sigma$$

$$Term_\Sigma, \mathsf{t} \vDash_\mathcal{K} \mathsf{f}(P_1, \dots, K, \dots, P_r) \text{ iff } \mathsf{t} = \mathsf{f}(\mathsf{t_1}, \dots, \_, \dots, \mathsf{t_r}) \wedge$$

$$Term_\Sigma, \mathsf{t_i} \vDash_\mathcal{P} P_i, \ \mathsf{f} : r \in \Sigma$$

From these additional formulae, we can derive a formula for specifying a specific term. For example, consider the signature $\Sigma = \{\mathsf{f} : 1, \mathsf{g}_1 : 0, \mathsf{g}_2 : 0\}$. The term $\mathsf{f}(\mathsf{f}(\mathsf{g}_1))$ can be specified by the term $(\mathsf{f}(\mathsf{f}(\_)))(\mathsf{g}_1)$ which is the application of a context, where a 0-ary function symbol has been replaced by a context hole, to the 0-ary function symbol.

Example 2.40 [$\mathrm{CL}_{Term_\Sigma}$-derived formulæ] We give some additional derived formula for $\mathrm{CL}_{Term_\Sigma}$ when $\Sigma = \{\mathsf{f} : 1, \mathsf{g}_1 : 0, \mathsf{g}_2 : 0\}$.

(i) $\mathsf{f}[\text{true}]$, a term starting with function symbol f.

(ii) $\Diamond\mathsf{f}[\text{true}]$, a term containing function symbol f.

(iii) $(\mathsf{g}_1 \triangleright \mathsf{f}[\mathsf{g}_1])(\text{true})$, a term starting with function symbol f.

(iv) $(\mathsf{f}[\text{true}] \blacktriangleright P)(g_1)$, a tree into which it is possible to replace a leaf with function symbol g_1 by a term starting with function symbol f to obtain a term satisfying P.

(v) $(\mathsf{g}_1 \triangleright P)(\mathsf{f}[\text{true}])$, a term containing a subterm starting with f that satisfies P if this subterm is replaced by g_1.

For the choice of Σ above, recall that $Term_\Sigma$ cannot have a zero element (discussion after Example 2.13). We therefore cannot derive the $*$-composition. We also cannot derive the $\circ$-composition, since terms cannot be split into two disjoint parts at the top level. We thus cannot apply BL-reasoning for this model.

3 Local Hoare Reasoning

We present a local Hoare reasoning framework for reasoning about commands acting on the data set of a CL-model. We present a general interpretation of Hoare triples for such commands based on the CL-satisfaction relation. We give a general definition of local command based on application in the CL-model, and present general inference rules for the Hoare triples on local commands. We then apply our general Hoare reasoning to three examples of data update: tree update, heap update, and term rewriting.

3.1 General Hoare Reasoning

Hoare Triples

Consider the Hoare triple $\{P\} \, \mathbb{C} \, \{Q\}$, where $\mathbb{C}$ is a program command, and P and Q are two CL-data formulae extended with variables and quantification. It has a non-standard fault-avoiding partial interpretation: it is partial in that the triple only holds if the data model satisfies P; it is fault-avoiding in that P specifies the presence of the resources necessary for the command to succeed. In our examples, the resources are the subdata in the working data determined by unique locations. Any attempt to access a location which is not currently available causes the program to fault. An immediate consequence of this interpretation is that, when $\mathbb{C}$ is run in a state satisfying P, it will only refer to the locations guaranteed to exist by P.

The behaviour of the commands $\mathbb{C}$ on a data set $\mathcal{D}$ is given by an operational semantics describing a relation $\rightsquigarrow$:

- $\mathbb{C}, \mathsf{s}, \mathsf{d} \rightsquigarrow \mathsf{s}', \mathsf{d}'$ specifies that the execution of command $\mathbb{C}$ starting with variable store s and data $\mathsf{d} \in \mathcal{D}$ will terminate successfully producing updated store s' and data $\mathsf{d}' \in \mathcal{D}$;

- $\mathbb{C}, \mathsf{s}, \mathsf{d} \rightsquigarrow$ fault specifies that $\mathbb{C}$ has attempted to access a resource not present in s, d and hence yields a fault.

Definition 3.1 (Interpretation of Hoare Triples) *Given a command* $\mathbb{C}$, *CL-models Mod with data set* $\mathcal{D}$, *and two CL-data formulae* P *and* Q, *a Hoare triple* $\{P\} \, \mathbb{C} \, \{Q\}$ *is said to hold iff whenever* $\mathsf{s}, \mathsf{d} \vDash_{\mathcal{P}} P$ *for* $d \in \mathcal{D}$ *then:*

(i) $\mathbb{C}, \mathsf{s}, \mathsf{d} \rightsquigarrow \mathsf{s}', \mathsf{d}' \Rightarrow \mathsf{s}', \mathsf{d}' \vDash_{\mathcal{P}} Q$ *(*partial interpretation*)*

(ii) $\mathbb{C}, \mathsf{s}, \mathsf{d} \not\rightsquigarrow$ fault *(*fault-avoiding interpretation*)*

Inference Rules

Our general inference rules for Hoare triples are given in Figure 1. In our examples of reasoning about update in the following sections, we will extend these general rules with axioms for specifying specific atomic commands. We assume standard sequential composition of commands. We do not consider any other general command constructs: for example, extending our reasoning to the while command just follows standard techniques.

We assume sets of free variables free($\mathbb{C}$) and modified variables mod($\mathbb{C}$) of a

$$
\text{Consequence} \qquad \frac{P' \Rightarrow P \quad \{P\}\,\mathbb{C}\,\{Q\} \quad Q \Rightarrow Q'}{\{P'\}\,\mathbb{C}\,\{Q'\}}
$$

$$
\begin{array}{l} \text{Auxiliary Variable} \\ \quad\text{Elimination} \end{array} \qquad \frac{\{P\}\,\mathbb{C}\,\{Q\}}{\{\exists x.P\}\,\mathbb{C}\,\{\exists x.Q\}}\; x \notin \text{free}(\mathbb{C})
$$

$$
\text{Frame} \qquad \frac{\{P\}\,\mathbb{C}\,\{Q\}}{\{K(P)\}\,\mathbb{C}\,\{K(Q)\}}\; \text{mod}(\mathbb{C}) \cap \text{free}(K) = \emptyset
$$

$$
\text{Sequencing} \qquad \frac{\{P\}\,\mathbb{C}_1\,\{Q\} \quad \{Q\}\,\mathbb{C}_2\,\{R\}}{\{P\}\,\mathbb{C}_1\,;\,\mathbb{C}_2\,\{R\}}
$$

Fig. 1. General Inference Rules for Hoare Triples

command. Intuitively, the set free($\mathbb{C}$) is the set of variables that may affect the execution of $\mathbb{C}$, and mod($\mathbb{C}$) the set of variables that $\mathbb{C}$ may modify. Typically, these sets are defined using a simple syntactic check, as we shall see in our examples. We require the following natural properties on free($\mathbb{C}$) and mod($\mathbb{C}$) for our rules to be sound; we use the notation $[s \mid x \leftarrow v]$ to denote stores updated with variable x assigned value v:

- if $x \notin \text{free}(\mathbb{C})$ then
  - $\cdot$ $\mathbb{C}, \mathsf{s}, \mathsf{d} \rightsquigarrow \text{fault}$ implies $\mathbb{C}, [\mathsf{s}|x \leftarrow \mathsf{v}], \mathsf{d} \rightsquigarrow \text{fault}$
  - $\cdot$ $\mathbb{C}, \mathsf{s}, \mathsf{d} \rightsquigarrow \mathsf{s}', \mathsf{d}'$ implies $\mathbb{C}, [\mathsf{s}|x \leftarrow \mathsf{v}], \mathsf{d} \rightsquigarrow [\mathsf{s}'|x \leftarrow \mathsf{v}], \mathsf{d}'$
- if $x \notin \text{mod}(\mathbb{C})$ then
  - $\cdot$ $\mathbb{C}, \mathsf{s}, \mathsf{d} \rightsquigarrow \mathsf{s}', \mathsf{d}'$ implies $\mathsf{s}(x) = \mathsf{s}'(x)$

The rules of consequence, auxiliary variable elimination and sequencing are standard Hoare logic rules. The Frame rule is non-standard, and generalises the Frame rule introduced in [7]. It relies on our assumption that commands behave locally (definition 3.2). If this is the case then, due to our interpretation of the Hoare triples, the premise implies that $\mathbb{C}$ only requires the resources specified by P, and therefore any additional data specified by K will be unaltered by that command. In addition, the side-condition guarantees that, although the store will be modified by the command, it will not affect any of the variables in K.

The soundness of the Frame Rule relies on the commands behaving locally. A command is local if it satisfies two properties, which were initially introduced in [7]. Intuitively, these properties are: the *safety-monotonicity* property which specifies that, if a command is safe in a state (that is, it does not fault in that state), then it

is safe in a larger state; and the *frame property* which specifies that if a command is safe in a given state, then any execution of the command on a larger state implies that it can be tracked to an execution on the smaller state. We now give a formal definition of local commands.

Definition 3.2 (Local Commands) *A command* $\mathbb{C}$ *is* local *for* CL-*model* $Mod = (\mathcal{D}, \mathcal{C}, ap, I)$ *if and only if it satisfies the following two properties:*

(i) safety-monotonicity property*:* $\forall \mathsf{d} \in \mathcal{D}, \mathsf{c} \in \mathcal{C}.\ \mathbb{C}, \mathsf{s}, \mathsf{d} \not\leadsto \text{fault} \wedge ap(\mathsf{c}, \mathsf{d})\downarrow \Rightarrow \mathbb{C}, \mathsf{s}, ap(\mathsf{c}, \mathsf{d}) \not\leadsto \text{fault}$;

(ii) frame property*:* $\forall \mathsf{d}, \mathsf{d}' \in \mathcal{D}, \mathsf{c} \in \mathcal{C}.\ \mathbb{C}, \mathsf{s}, \mathsf{d} \not\leadsto \text{fault} \wedge ap(\mathsf{c}, \mathsf{d})\downarrow \wedge \mathbb{C}, \mathsf{s}, ap(\mathsf{c}, \mathsf{d}) \leadsto \mathsf{s}', \mathsf{d}' \Rightarrow \exists \mathsf{d}'' \in \mathcal{D}. \mathbb{C}, \mathsf{s}, \mathsf{d} \leadsto \mathsf{s}', \mathsf{d}'' \wedge \mathsf{d}' = ap(\mathsf{c}, \mathsf{d}'')$.

Theorem 3.3 (Soundness) *The rules in Figure 3.1 are sound.*

Proof. The proof of soundness for the rules is routine, except in the Frame Rule case which requires the commands to be local. Assume that $\{P\}\mathbb{C}\{Q\}$, $\mathsf{s}, \mathsf{c} \vDash_\mathcal{K} K$, $\mathsf{s}, \mathsf{d} \vDash_\mathcal{P} P$ and $ap(\mathsf{c}, \mathsf{d})\downarrow$. We know that $\mathbb{C}, \mathsf{s}, \mathsf{d} \not\leadsto \text{fault}$ by the premise and our interpretation of the Hoare triples. We have $\mathbb{C}, \mathsf{s}, ap(\mathsf{c}, \mathsf{d}) \not\leadsto \text{fault}$ by safety monotonicity of $\mathbb{C}$. If $\mathbb{C}, \mathsf{s}, ap(\mathsf{c}, \mathsf{d}) \leadsto \mathsf{s}', \mathsf{d}'$ then by the frame property $\exists \mathsf{d}''. \mathbb{C}, \mathsf{s}, \mathsf{d} \leadsto \mathsf{s}', \mathsf{d}''$ and $\mathsf{d}' = ap(\mathsf{c}, \mathsf{d}'')$. By the premise, we have $\mathsf{s}', \mathsf{d}'' \vDash_\mathcal{P} Q$. Since $\text{mod}(\mathbb{C}) \cap \text{free}(K) = \emptyset$, we also have $\mathsf{s}', \mathsf{c} \vDash_\mathcal{K} K$. We may therefore conclude that $\mathsf{s}', \mathsf{d}' \vDash_\mathcal{P} K(Q)$ as required. $\square$

Weakest Preconditions and Small Axioms
The weakest precondition axioms for the atomic commands are a standard ingredient of Hoare reasoning. They imply completeness of the Hoare triple inference system for straight-line code. They are also typically used in verification tools based on Hoare reasoning.

Definition 3.4 (Weakest Preconditions) *The weakest precondition of a command* $\mathbb{C}$ *with respect to a postcondition* P *is a set of states* $wp(\mathbb{C}, P)$ *where* $(\mathsf{s}, \mathsf{d}) \in wp(\mathbb{C}, P)$ *iff* $\mathbb{C}, \mathsf{s}, \mathsf{d} \not\leadsto \text{fault}$ *and* $\mathbb{C}, \mathsf{s}, \mathsf{d} \leadsto \mathsf{s}', \mathsf{d}' \Rightarrow \mathsf{s}', \mathsf{d}' \vDash_\mathcal{D} P$.

If the weakest precondition of a command $\mathbb{C}$ with respect to P is expressible as a formula P_{wp} then the weakest precondition triple $\{P_{\text{wp}}\}\mathbb{C}\{P\}$ holds and, whenever $\{P'\}\mathbb{C}\{P\}$ for some data formulae P', then $P' \Rightarrow P_{wp}$. The weakest precondition for the sequencing command $\mathbb{C}_1 \ ; \ \mathbb{C}_2$ is the standard formula $wp(\mathbb{C}_1, wp(\mathbb{C}_2, P))$. The weakest preconditions of the atomic commands in our examples of data update are given in the following sections.

In [10] an alternative style of axioms was introduced, called the *small axioms*. These are triples of the form $\{P_{fp}\}\mathbb{C}\{Q\}$, where P_{fp} just describes the footprint of command $\mathbb{C}$ and Q describes the result of $\mathbb{C}$ on that footprint. From our intuition regarding local commands, it should be possible to derive the weakest precondition axioms, and hence all Hoare triples, from the small axioms. In the following sections, we will see that this is indeed the case for our examples.

3.2 Hoare Reasoning about Tree Update

We describe a core language for manipulating trees, give small axioms and weakest precondition axioms for the atomic commands, and show that the weakest precondition axioms are derivable from the small axioms using the general inference rules given in figure 3.1. We use $LocTree_N$ from example 2.6 as our tree model. Recall that local Hoare reasoning only works for local commands (definition 3.2). In the tree case, this means we must use a model in which the nodes are precisely identified. For example, we shall use the local dispose command $[n]_T := 0$, which disposes the subtree with top node given by variable n. For this command to be local, the node value of n must be precisely identified. If the value of n did not describe a unique node, then the frame property would fail. The tree model $LocTree_N$ therefore provides the simplest tree model for illustrating our ideas, since its only node structure is the unique identifiers. In [1,6], we explore a trees-with-pointers model and associated update language which correspond much more closely to XML and XML update. This model has a much more complicated node structure, consisting of labels (XML tags), unique node identifiers (XML identifiers) and cross-pointers (XML idrefs). It is easy to adapt the results presented here to this more complicated setting.

Tree Update Language

Our data storage model resembles that of traditional imperative languages, except that trees are first-class objects. It consists of two components: a working tree t (analogous to a heap) and a store s. The store is a function defined on both node variables and tree variables which are mapped to values:

$$
\begin{aligned}
\text{node variables} \quad & Var_N = \{n, m, \dots\} \\
\text{tree variables} \quad & Var_{\mathcal{T}_N} = \{x, y, \dots\} \\
\text{stores} \quad & s \in (Var_N \to N) \times (Var_{\mathcal{T}_N} \to \mathcal{T}_N)
\end{aligned}
$$

This approach of storing trees allows us to break down complex operations, such as moving trees, into smaller ones that deal with only one area of the working tree at a time and can hence be analysed locally.

We present a core update language for directly manipulating trees. Our language is simple, yet expressive enough to illustrate the subtleties of tree update. The commands consist of variable assignment, updates and sequencing. The update commands are analogous to standard commands used for updating heaps: dispose, append, lookup and new. There are however subtle differences. First, our update commands manipulate whole tree values, not just integers as for heaps. Second, there are two location choices for where to update, either next to the distinguished node or just below the node. We use the notation $[n]_T$ to denote a subtree with top node n, and $[n]_{SF}$ to denote the subforest underneath n. Finally, the new command creates a new node which, unlike the heap case, needs to be specifically located at a particular node n.

Definition 3.5 (Commands for Tree Update) *The commands of our tree update language are given by the grammar:*

$$\mathbb{C} ::= n := n' \mid x := x' \quad \textit{variable assignment}$$

$$\mathbb{C}_{up}(n) \qquad\qquad \textit{update at location } n$$

$$\mathbb{C} \,;\, \mathbb{C} \qquad\qquad\quad \textit{sequencing}$$

The tree updates $\mathbb{C}_{up}(n)$ acting at location n are defined as follows, with each update command having two variants corresponding to updating at the identified node or at its subforest:

$$\mathbb{C}_{up}(n) ::= [n]_{\mathrm{T}} := 0 \qquad\quad [n]_{\mathrm{SF}} := 0 \qquad\quad \textit{dispose}$$

$$[n]_{\mathrm{T}} \mathbin{*=} x \qquad\quad [n]_{\mathrm{SF}} \mathbin{*=} x \qquad\quad \textit{append}$$

$$x := [n]_{\mathrm{T}} \qquad\quad x := [n]_{\mathrm{SF}} \qquad\quad \textit{lookup}$$

$$n' := \textit{new } [n]_{\mathrm{T}} \quad\; n' := \textit{new } [n]_{\mathrm{SF}} \quad\; \textit{new}$$

The set free($\mathbb{C}$) is the set of variables occurring in $\mathbb{C}$. The set mod($\mathbb{C}$) is $\{n\}$ for node variable assignment, $\{x\}$ for tree variable assignment and lookup, $\{n'\}$ for new, $\emptyset$ for the other atomic commands, and $mod(\mathbb{C}_1) \cup mod(\mathbb{C}_2)$ for $\mathbb{C}_1 \,;\, \mathbb{C}_2$.

The left-hand dispose command replaces the subtree with top node n by the empty tree 0; the right-hand dispose command replaces the subforest underneath node n by the empty tree. The append commands are analogous: the right-hand command adds the tree value given by x next to node n; the left-hand command adds it underneath node n. The lookup command assigns either the subtree with top node n or the subforest underneath n to the variable x. The new commands create a new tree node with a fresh identifier and an empty subforest, either adds this fresh node next to the node n or underneath it, and stores the new node identifier in variable n'.

These update commands all rely on the node identified by variable n to be in the working tree. If it is not, they will fault. A different error occurs when an append command tries to insert a tree with a node identifier that clashes with the working tree. In this case, the rule diverges, returning no result. This choice to diverge rather than fault is necessary in order to keep the command local (definition 3.2). In fact, our current choice of update is somewhat unnatural, precisely because of its dependence on the global state of the tree. A more realistic append operation is to rename the node identifiers of the tree being inserted with fresh identifiers [6]. Our simpler operation is enough for this paper.

Definition 3.6 (Operational Semantics for Tree Update) *The operational semantics of the tree update language is given in Figure 2, using an evaluation relation $\rightsquigarrow$ defined on configuration triples $\mathbb{C}, \mathsf{s}, \mathsf{t}$, terminal states s, t and faults* fault.

Example 3.7 [Move] We present a simple program move(n, n') which takes a sub-

$$\frac{\mathsf{s}(n') = \mathsf{n}'}{n := n', \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|n \leftarrow \mathsf{n}'], \mathsf{t}} \quad \frac{\mathsf{s}(x') = \mathsf{t}'}{x := x', \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|x \leftarrow \mathsf{t}'], \mathsf{t}}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{[n]_{\mathrm{T}} := 0, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}, \mathrm{ap}(\mathsf{c}, 0)} \quad \frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{[n]_{\mathrm{SF}} := 0, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}, \mathrm{ap}(\mathsf{c}, \mathsf{n}[0])}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}']) \quad \mathsf{s}(x) \equiv \mathsf{t}'' \quad \mathsf{t}'' \# \mathsf{t}}{[n]_{\mathrm{T}} *= x, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}' \mid \mathsf{t}''])}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}']) \quad \mathsf{s}(x) \equiv \mathsf{t}'' \quad \mathsf{t}'' \# \mathsf{t}}{[n]_{\mathrm{SF}} *= x, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}, \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}' \mid \mathsf{t}''])}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{x := [n]_{\mathrm{T}}, \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|x \leftarrow \mathsf{n}[\mathsf{t}']], \mathsf{t}} \quad \frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{x := [n]_{\mathrm{SF}}, \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|x \leftarrow \mathsf{t}'], \mathsf{t}}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}']) \quad \mathsf{n}' \# \mathsf{t}}{n' := \mathrm{new}\ [n]_{\mathrm{T}}, \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|n' \leftarrow \mathsf{n}'], \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}' \mid \mathsf{n}'[0]])}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}']) \quad \mathsf{n}' \# \mathsf{t}}{n' := \mathrm{new}\ [n]_{\mathrm{SF}}, \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s}|n' \leftarrow \mathsf{n}'], \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}' \mid \mathsf{n}'[0]])}$$

$$\frac{\mathbb{C}_1, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathbb{C}', \mathsf{s}', \mathsf{t}'}{(\mathbb{C}_1\ ;\ \mathbb{C}_2), \mathsf{s}, \mathsf{t} \rightsquigarrow (\mathbb{C}'\ ;\ \mathbb{C}_2), \mathsf{s}', \mathsf{t}'} \quad \frac{\mathbb{C}_1, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}', \mathsf{t}'}{(\mathbb{C}_1\ ;\ \mathbb{C}_2), \mathsf{s}, \mathsf{t} \rightsquigarrow \mathbb{C}_2, \mathsf{s}', \mathsf{t}'}$$

$$\frac{\mathsf{s}(n) = \mathsf{n} \quad \mathsf{t} \not\equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])}{\mathbb{C}_{\mathrm{up}}(n), \mathsf{s}, \mathsf{t} \rightsquigarrow \mathrm{fault}} \quad \frac{\mathbb{C}_1, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathrm{fault}}{(\mathbb{C}_1\ ;\ \mathbb{C}_2), \mathsf{s}, \mathsf{t} \rightsquigarrow \mathrm{fault}}$$

$[s|x \leftarrow \mathsf{v}]$ denotes the store s updated with $s(x) = \mathsf{v}$; similarly for $[s|n \leftarrow \mathsf{n}]$.
$t'' \# t$ specifies that the node identifiers of t'' and t are disjoint.
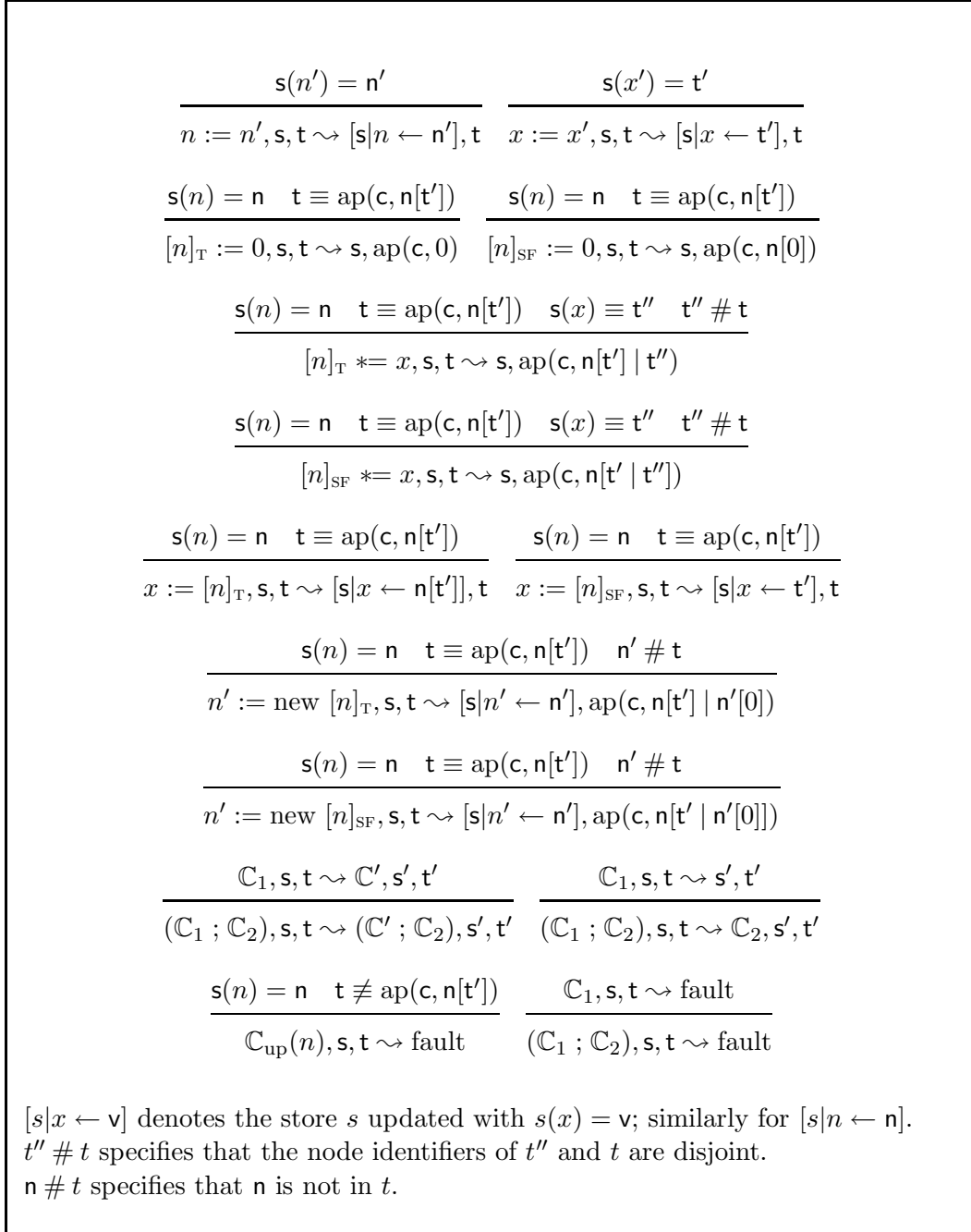$\mathsf{n} \# t$ specifies that n is not in t.

Fig. 2. Operational Semantics for Tree Update

tree with top node n and moves it underneath the node n'. The program consists
of three commands: the assignment of the subtree to variable x, the disposal of the

subtree, and the append of the value of x (the subtree) underneath node n':

$$move(n, n') \triangleq x := [n]_{\text{T}} \; ;$$
$$[n]_{\text{T}} := 0 \; ;$$
$$[n']_{\text{SF}} *= x$$

We will use this example to illustrate our program logic reasoning.

CL for Tree Update

Our application of CL for $LocTree_N$ is essentially the same as that for CL_{Tree_A} given in definition 3.13. The differences are that we use node variables, rather than node constants, and tree variables in our formulae, and allow quantification over these variables. These differences are necessary for our Hoare reasoning about our tree-update commands, which are based on node and tree variables whose values are determined by a store.

Definition 3.8 [CL for Tree Update] CL applied to the tree model $LocTree_N$, denoted $\text{CL}_{LocTree_N}$, consists of $\text{CL}_{LocTree_N}$-formulae constructed by extending the CL_0-formulae defined inductively by the grammars in definition 2.10 with the following additional cases:

data formulae

$$P ::= x \qquad\qquad \text{specific data formulae, } x \in Var_{\mathcal{T}_N}$$
$$\exists n. P \mid \exists x. P \quad \text{quantification, } n \in Var_N, \; x \in Var_{\mathcal{T}_N}$$

context formulae

$$K ::= n[K] \mid K \circ P \quad \text{specific context formulae, } n \in Var_N$$
$$\exists n. K \mid \exists x. K \quad \text{quantification, } n \in Var_N, \; x \in Var_{\mathcal{T}_N}$$

The $\text{CL}_{LocTree_N}$-satisfaction relation consists of two relations of the form

$$LocTree_N, \mathsf{s}, \mathsf{t} \vDash_{\mathcal{P}} P \quad \text{and} \quad LocTree_N, \mathsf{s}, \mathsf{c} \vDash_{\mathcal{K}} K$$

where $\mathsf{t} \in \mathcal{T}_N$ and $\mathsf{c} \in \mathcal{C}_N$, with the extra store component used to determine the values of the variables. They are defined analogously to the CL_{Tree_A}-satisfaction relation (definition 3.13) with a standard treatment of the variables and quantification

given by:

$$LocTree_N, \mathsf{s}, \mathsf{t} \vDash_{\mathcal{P}} x \qquad \text{iff } \mathsf{s}(x) = \mathsf{t}$$

$$LocTree_N, \mathsf{s}, \mathsf{t} \vDash_{\mathcal{P}} \exists x.\, P \quad \text{iff } \exists \mathsf{t}' \in \mathcal{T}_N.\, LocTree_N, [\mathsf{s} \mid x \leftarrow \mathsf{t}'], \mathsf{t} \vDash_{\mathcal{P}} P$$

$$LocTree_N, \mathsf{s}, \mathsf{t} \vDash_{\mathcal{P}} \exists n.\, P \quad \text{iff } \exists \mathsf{n} \in N.\, LocTree_N, [\mathsf{s} \mid n \leftarrow \mathsf{n}], \mathsf{t} \vDash_{\mathcal{P}} P$$

$$LocTree_N, \mathsf{s}, \mathsf{c} \vDash_{\mathcal{K}} n[K] \quad \text{iff } \exists \mathsf{c}' \in \mathcal{C}_N. \quad \mathsf{c} \equiv \mathsf{s}(n)[\mathsf{c}'] \wedge LocTree_N, \mathsf{s}, \mathsf{c}' \vDash_{\mathcal{K}} K$$

$$LocTree_N, \mathsf{s}, \mathsf{c} \vDash_{\mathcal{K}} K \circ P \quad \text{iff } \exists \mathsf{c}' \in \mathcal{C}_N, \mathsf{t} \in \mathcal{T}_N. \quad \mathsf{c} \equiv \mathsf{c}' \mid \mathsf{t} \wedge$$
$$LocTree_N, \mathsf{s}, \mathsf{c}' \vDash_{\mathcal{K}} K \wedge LocTree_N, \mathsf{s}, \mathsf{t} \vDash_{\mathcal{P}} P$$

$$LocTree_N, \mathsf{s}, \mathsf{c} \vDash_{\mathcal{K}} \exists x.\, K \quad \text{iff } \exists \mathsf{t} \in \mathcal{T}_N.\, LocTree_N, [\mathsf{s} \mid x \leftarrow \mathsf{t}], \mathsf{c} \vDash_{\mathcal{P}} K$$

$$LocTree_N, \mathsf{s}, \mathsf{c} \vDash_{\mathcal{K}} \exists n.\, K \quad \text{iff } \exists \mathsf{n} \in N.\, LocTree_N, [\mathsf{s} \mid n \leftarrow \mathsf{n}], \mathsf{c} \vDash_{\mathcal{P}} K$$

Local Hoare Reasoning about Tree Update
We show that the local Hoare reasoning framework described in Section 3.1 can be applied to our tree update language.

Lemma 3.9 (Locality for Tree Update) *All the commands in our tree update language are local.*

Proof. Variable assignment is trivially local as it is independent of the tree. If $\mathbb{C}_1$, $\mathbb{C}_2$ are local then $\mathbb{C}_1$; $\mathbb{C}_2$ is local by definition. Now consider the update commands at location n. These only fault if n is not in the tree. Hence, the safety monotonicity condition holds for these commands, since the successful application of a context to the given tree still has n in the large tree. Notice that all the operational rules in Figure 2 for the update commands describe the transformation of a well-defined tree $\mathsf{t} \equiv \mathrm{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}'])$ to a tree $\mathsf{t}_1 \equiv \mathrm{ap}(\mathsf{c}, \mathsf{t}'')$ for varying values of t''. The frame property holds, since an update on a larger well-defined tree $\mathrm{ap}(\mathsf{c}', \mathsf{t}) \equiv \mathrm{ap}(\mathsf{c}; \mathsf{c}', \mathsf{n}[\mathsf{t}'])$, where ; denotes standard context composition, yields a transformation to the tree $\mathrm{ap}(\mathsf{c}; \mathsf{c}', \mathsf{t}'') \equiv \mathrm{ap}(\mathsf{c}', \mathsf{t}_1)$. □

Definition 3.10 (Small Axioms for Tree Update) *The small axioms for the atomic tree update commands are given in Figure 3.*

It is straightforward to see that these axioms are indeed small: the preconditions precisely describe the footprint of the command, and the postconditions just describe the immediate effect of the command on the footprint. The weakest precondition axioms in Figure 4 are also easy to read. For example, the formula describing the weakest precondition of the dispose command $[n]_{\mathrm{T}} := 0$ just states that the tree contains a subtree identified by variable n, which when replaced by the empty tree gives property P.

Theorem 3.11 (Weakest Precondition Axioms for Tree Update) *The axioms for the atomic tree update commands given in Figure 4 are weakest precondition axioms.*

$$\{(n' = n_1) \wedge 0\} \; n := n' \qquad\qquad \{(n = n_1) \wedge 0\}$$

$$\{(x' = x_1) \wedge 0\} \; x := x' \qquad\qquad \{(x = x_1) \wedge 0\}$$

$$\{n[\text{true}]\} \; [n]_{\text{T}} := 0 \qquad \{0\}$$

$$\{n[\text{true}]\} \; [n]_{\text{SF}} := 0 \qquad \{n[0]\}$$

$$\{n[y]\} \; [n]_{\text{T}} \mathbin{*=} x \qquad \{n[y] \mid x\}$$

$$\{n[y]\} \; [n]_{\text{SF}} \mathbin{*=} x \qquad \{n[y \mid x]\}$$

$$\{y \wedge n[\text{true}]\} \; x := [n]_{\text{T}} \qquad \{y \wedge (x = y)\}$$

$$\{n[y]\} \; x := [n]_{\text{SF}} \qquad \{n[y] \wedge (x = y)\}$$

$$\{n[y]\} \; n' := \text{new} \; [n]_{\text{T}} \quad \{n[y] \mid n'[0]\}$$

$$\{n[y]\} \; n' := \text{new} \; [n]_{\text{SF}} \quad \{n[y \mid n'[0]]\}$$

$$\text{where } n_1, x_1, y \notin \text{mod}(\mathbb{C})$$

Fig. 3. Small Axioms for Tree Update

Proof. The proof for each atomic command is a simple application of the definitions. For example, consider the update command $[n]_{\text{T}} := 0$. We have $\text{wp}([n]_{\text{T}} := 0, P) = \{(\mathsf{s}, \mathsf{t}') \mid \exists \mathsf{c}, \mathsf{t}, \mathsf{n}.\, \mathsf{s}(n) = \mathsf{n} \wedge \mathsf{t}' \equiv \text{ap}(\mathsf{c}, \mathsf{n}[\mathsf{t}]) \wedge \mathsf{s}, \text{ap}(\mathsf{c}, \mathsf{n}[0]) \vDash_{\mathcal{P}} P\}$ by definition. We must prove that $\mathsf{s}, \mathsf{t}' \vDash_{\mathcal{P}} (0 \triangleright P)(n[\text{true}]) \Leftrightarrow (\mathsf{s}, \mathsf{t}') \in \text{wp}([n]_{\text{T}} := 0, P)$. This follows directly from the definition of the CL-satisfaction relation. The proofs for the other atomic commands are similar. $\qquad\square$

Lemma 3.12 (Derivability of Weakest Precondition Axioms) *The weakest precondition axioms in Figure 5 are derivable from the small axioms in Figure 3 and the proof rules in Figure 1.*

Proof. See Figure 5. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Using the move program $move(n, n')$ from Example 3.7, we demonstrate our local Hoare reasoning. By calculating the weakest precondition of the program with respect to the postcondition true, we can derive the necessary condition for non-faulting execution. The following derivation applies the weakest precondition axioms backwards, and simplifies the formulae at each step.

$$\{P[n'/n]\}\ n := n' \qquad \{P\}$$

$$\{P[x'/x]\}\ x := x' \qquad \{P\}$$

$$\{(0 \rhd P)(n[\text{true}])\}\ [n]_{\text{T}} := 0 \qquad \{P\}$$

$$\{(n[0] \rhd P)(n[\text{true}])\}\ [n]_{\text{SF}} := 0 \qquad \{P\}$$

$$\{\exists y.\,((n[y]\mid x) \rhd P)(n[y])\}\ [n]_{\text{T}} \ast= x \qquad \{P\}$$

$$\{\exists y.\,(n[y\mid x] \rhd P)(n[y])\}\ [n]_{\text{SF}} \ast= x \qquad \{P\}$$

$$\{\exists y.\,\Diamond(y \wedge n[\text{true}]) \wedge P[y/x]\}\ x := [n]_{\text{T}} \qquad \{P\}$$

$$\{\exists y.\,\Diamond n[y] \wedge P[y/x]\}\ x := [n]_{\text{SF}} \qquad \{P\}$$

$$\{\exists y.\forall n'.\,((n[y]\mid n'[0]) \rhd P)(n[y])\}\ n' := \text{new } [n]_{\text{T}}\ \{P\}$$

$$\{\exists y.\forall n'.\,(n[y\mid n'[0]] \rhd P)(n[y])\}\ n' := \text{new } [n]_{\text{SF}}\ \{P\}$$

$$\text{where } y \notin \text{free}(\mathbb{C}) \cup \text{free}(P)$$

Fig. 4. Weakest Preconditions for Tree Update

$$\{(0 \rhd \text{True}(n'[\text{true}]))(n[\text{true}])\}$$

$$x := [n]_{\text{T}}$$

$$\{(0 \rhd \text{True}(n'[\text{true}]))(n[\text{true}])\}$$

$$[n]_{\text{T}} := 0$$

$$\{\text{True}(n'[\text{true}])\}$$

$$[n']_{\text{SF}} \ast= x$$

$$\{\text{true}\}$$

Hence, the safety precondition of a non-faulting execution of $move(n, n')$ is $(0 \rhd$ True$(n'[\text{true}]))(n[\text{true}])$. This assertion expresses exactly what we would expect: the current tree must contain nodes n' and n, but n' cannot be underneath n since the subtree with root n can be inserted in a context containing n'. Furthermore, we can now easily derive a general specification for the command, using tree variables u, v as place-holders:

$$\{(0 \rhd \text{True}(n'[u]))(n[v])\}$$

$$move(n, n')$$

$$\{\text{True}(n'[u\mid n[v]])\}$$

Variable Assignment

$$\frac{\{(n' = n_1) \wedge 0\} \; n := n' \; \{(n = n_1) \wedge 0\}}{\{(0 \triangleright P[n_1/n])((n' = n_1) \wedge 0)\} \; n := n' \; \{(0 \triangleright P[n_1/n])((n = n_1) \wedge 0)\}} \; \text{FRAME}$$

$$\frac{}{\{(n' = n_1) \wedge (0 \triangleright P[n_1/n])(0)\} \; n := n' \; \{(n = n_1) \wedge (0 \triangleright P[n_1/n])(0)\}} \; \text{CONS}$$

$$\frac{}{\{(n' = n_1) \wedge P[n_1/n]\} \; n := n' \; \{(n = n_1) \wedge P[n_1/n]\}} \; \text{CONS}$$

$$\frac{}{\{P[n'/n]\} \; n := n' \; \{P\}} \; \text{CONS/VARS}$$

$$\frac{\{(x' = x_1) \wedge 0\} \; x := x' \; \{(x = x_1) \wedge 0\}}{\{(0 \triangleright P[x_1/x])((x' = x_1) \wedge 0)\} \; x := x' \; \{(0 \triangleright P[x_1/x])((x = x_1) \wedge 0)\}} \; \text{FRAME}$$

$$\frac{}{\{(x' = x_1) \wedge (0 \triangleright P[x_1/x])(0)\} \; x := x' \; \{(x = x_1) \wedge (0 \triangleright P[x_1/x])(0)\}} \; \text{CONS}$$

$$\frac{}{\{(x' = x_1) \wedge P[x_1/x]\} \; x := x' \; \{(x = x_1) \wedge P[x_1/x]\}} \; \text{CONS}$$

$$\frac{}{\{P[x'/x]\} \; x := x' \; \{P\}} \; \text{CONS/VARS}$$

Dispose

$$\frac{\{n[\text{true}]\} \; [n]_\text{T} := 0 \; \{0\}}{\{(0 \triangleright P)(n[\text{true}])\} \; [n]_\text{T} := 0 \; \{(0 \triangleright P)(0)\}} \; \text{FRAME}$$

$$\frac{}{\{(0 \triangleright P)(n[\text{true}])\} \; [n]_\text{T} := 0 \; \{P\}} \; \text{CONS}$$

$$\frac{\{n[\text{true}]\} \; [n]_\text{SF} := 0 \; \{n[0]\}}{\{(n[0] \triangleright P)(n[\text{true}])\} \; [n]_\text{SF} := 0 \; \{(n[0] \triangleright P)(n[0])\}} \; \text{FRAME}$$

$$\frac{}{\{(n[0] \triangleright P)(n[\text{true}])\} \; [n]_\text{SF} := 0 \; \{P\}} \; \text{CONS}$$

Append

$$\frac{\{n[y]\} \; [n]_\text{T} \mathrel{*}= x \; \{n[y] \mid x\}}{\{((n[y] \mid x) \triangleright P)(n[y])\} \; [n]_\text{T} \mathrel{*}= x \; \{((n[y] \mid x) \triangleright P)(n[y] \mid x)\}} \; \text{FRAME}$$

$$\frac{}{\{((n[y] \mid x) \triangleright P)(n[y])\} \; [n]_\text{T} \mathrel{*}= x \; \{P\}} \; \text{CONS}$$

$$\frac{}{\{\exists y. \, ((n[y] \mid x) \triangleright P)(n[y])\} \; [n]_\text{T} \mathrel{*}= x \; \{P\}} \; \text{VARS}$$

$$\frac{\{n[y]\} \; [n]_\text{SF} \mathrel{*}= x \; \{n[y \mid x]\}}{\{(n[y \mid x] \triangleright P)(n[y])\} \; [n]_\text{SF} \mathrel{*}= x \; \{(n[y \mid x] \triangleright P)(n[y \mid x])\}} \; \text{FRAME}$$

$$\frac{}{\{(n[y \mid x] \triangleright P)(n[y])\} \; [n]_\text{SF} \mathrel{*}= x \; \{P\}} \; \text{CONS}$$

$$\frac{}{\{\exists y. \, (n[y \mid x] \triangleright P)(n[y])\} \; [n]_\text{SF} \mathrel{*}= x \; \{P\}} \; \text{VARS}$$

Lookup

$$\frac{\{y \wedge n[\text{true}]\} \; x := [n]_\text{T} \; \{y \wedge n[\text{true}] \wedge (x = y)\}}{\{((y \wedge n[\text{true}]) \triangleright P[y/x])(y \wedge n[\text{true}])\} \; x := [n]_\text{T} \; \{((y \wedge n[\text{true}]) \triangleright P[y/x])(y \wedge n[\text{true}] \wedge (x = y))\}} \; \text{FRAME}$$

$$\frac{}{\{((y \wedge n[\text{true}]) \triangleright P[y/x])(y \wedge n[\text{true}])\} \; x := [n]_\text{T} \; \{((y \wedge n[\text{true}]) \triangleright P[y/x])(y \wedge n[\text{true}]) \wedge (x = y)\}} \; \text{CONS}$$

$$\frac{}{\{\Diamond(y \wedge n[\text{true}]) \wedge P[y/x]\} \; x := [n]_\text{T} \; \{P[y/x] \wedge (x = y)\}} \; \text{CONS}$$

$$\frac{}{\{\exists y. \, \Diamond(y \wedge n[\text{true}]) \wedge P[y/x]\} \; x := [n]_\text{T} \; \{P\}} \; \text{CONS/VARS}$$

$$\frac{\{n[y]\} \; x := [n]_\text{SF} \; \{n[y] \wedge (x = y)\}}{\{(n[y] \triangleright P[y/x])(n[y])\} \; x := [n]_\text{SF} \; \{(n[y] \triangleright P[y/x])(n[y] \wedge (x = y))\}} \; \text{FRAME}$$

$$\frac{}{\{(n[y] \triangleright P[y/x])(n[y])\} \; x := [n]_\text{SF} \; \{((n[y] \triangleright P[y/x])(n[y])) \wedge (x = y)\}} \; \text{CONS}$$

$$\frac{}{\{\Diamond n[y] \wedge P[y/x]\} \; x := [n]_\text{SF} \; \{(x = y) \wedge P[y/x]\}} \; \text{CONS}$$

$$\frac{}{\{\exists y. \, \Diamond n[y] \wedge P[y/x]\} \; x := [n]_\text{SF} \; \{P\}} \; \text{CONS/VARS}$$

New

$$\frac{\{n[y]\} \; n' := \text{new} \; [n]_\text{T} \; \{n[y] \mid n'[0]\}}{\{(\forall n'. \, (n[y] \mid n'[0]) \triangleright P)(n[y])\} \; n' := \text{new} \; [n]_\text{T} \; \{(\forall n'. \, (n[y] \mid n'[0]) \triangleright P)(n[y] \mid n'[0])\}} \; \text{FRAME}$$

$$\frac{}{\{\exists y. \forall n'. ((n[y] \mid n'[0]) \triangleright P)(n[y])\} \; n' := \text{new} \; [n]_\text{T} \; \{P\}} \; \text{CONS/VARS}$$

$$\frac{\{n[y]\} \; n' := \text{new} \; [n]_\text{SF} \; \{n[y] \mid n'[0]\}}{\{(\forall n'. \, (n[y] \mid n'[0]) \triangleright P)(n[y])\} \; n' := \text{new} \; [n]_\text{SF} \; \{(\forall n'. \, (n[y] \mid n'[0]) \triangleright P)(n[y] \mid n'[0])\}} \; \text{FRAME}$$

$$\frac{}{\{\exists y. \forall n'. ((n[y] \mid n'[0]) \triangleright P)(n[y])\} \; n' := \text{new} \; [n]_\text{SF} \; \{P\}} \; \text{CONS/VARS}$$

Fig. 5. Derivations of the Weakest Precondition Axioms for Tree Update

Comparison with BL-*reasoning*

We claim that CL is essential for local Hoare reasoning about tree update. In particular, we believe that it is not possible to do this style of Hoare reasoning based on BL. The small axioms for tree update in Figure 3 are expressible in BL for $LocTree_N$. However, the weakest preconditions are not.

Definition 3.13 [BL for Tree Update] BL applied to the tree model $LocTree_N$, denoted $BL_{LocTree_N}$, consists of $BL_{LocTree_N}$-formulae constructed by extending the BL-formulae in definition 2.25 with the following additional data formulae:

$$P ::= n[P] \mid \hat{n}[P] \mid x \mid \diamond P \quad \text{specific data formulae}, n \in Var_N, x \in Var_{\mathcal{T}_N}$$

$$\exists n.\, P \mid \exists x.\, P \qquad \text{quantification}, n \in Var_N,\ x \in Var_{\mathcal{T}_N}$$

The $BL_{LocTree_N}$-satisfaction relation $LocTree_N, \mathsf{s}, \mathsf{t} \vDash_{\mathcal{P}_{\mathrm{BL}}} P$ is the obvious adaptation of the BL_{Tree_A}-satisfaction relation given in definition 2.38.

$BL_{LocTree_N}$ can express updates at the top level of trees by using the composition and branch adjoints to build contexts around the tree. What it cannot do is reason directly about update in an arbitrary context. Consider the weakest precondition of $[n]_{\mathrm{T}} := 0$ given by CL-formula $(0 \triangleright P)(n[\mathsf{true}])$. Even simple postconditions require a case-by-case analysis using $BL_{LocTree_N}$. For example, recall from section 2.3.3 that CL-formula $(0 \triangleright \mathsf{m}_1[\mathsf{m}_2[0]])(n[\mathsf{true}])$ corresponds to BL-formulae $\mathsf{m}_1[\mathsf{m}_2[n[\mathsf{true}]]] \vee \mathsf{m}_1[\mathsf{m}_2[0] \mid n[\mathsf{true}]] \vee (\mathsf{m}_1[\mathsf{m}_2[0]] \mid n[\mathsf{true}])$, whereas CL-formula $(0 \triangleright \diamond \mathsf{m}_2[\mathsf{true}])(n[\mathsf{true}])$ corresponds to BL-formula $\diamond \mathsf{m}_2[\mathsf{true}] \wedge \diamond n[\neg \diamond \mathsf{m}_2[\mathsf{true}]]$. These BL-preconditions are clearly not parametric in the postcondition. In [2], we show that it is not possible to give the weakest precondition for $[n]_{\mathrm{T}} := 0$ using BL-reasoning. This involves extending the logics with propositional variables, and proving that the CL-formula $(0 \triangleright p)(n[\mathsf{true}])$ for propositional variable p cannot be expressed in $BL_{LocTree_N}$.

3.3 Hoare Reasoning about Heap Update

We now describe local Hoare reasoning about heap update. We have already shown that CL-reasoning for heaps is the same as BL-reasoning for heaps (Theorem 2.29). The connection is much deeper, in that our Hoare reasoning is exactly analogous to local Hoare reasoning based on SL [10]. In addition, we shall see that the small axioms, weakest preconditions and the derivations of the weakest preconditions from the small axioms are analogous to the tree case.

Heap Update Language

The data storage model is the RAM model used in SL [10]. It consists of a working heap $\mathsf{h} \in \mathcal{H}$ where $\mathcal{H}$ is the data set of the CL_0-model $Heap$ given in Example 2.6, and a store s. We use the notation $h \cdot h'$ for the composition of heaps, and $\mathsf{n} \mapsto \mathsf{n}'$ for unary cell with address n and value n'. The store maps node variables to natural

numbers.

$$\text{node variables} \quad Var_{\mathbb{N}} = \{n, m, \dots\}$$

$$\text{expressions} \quad E, F ::= n \mid \text{nil}$$

$$\text{stores} \quad s \in (Var_{\mathbb{N}} \to \mathbb{N})$$

We write $[s \mid n \leftarrow \mathsf{n}]$ to denote the store s updated with $s(n) = \mathsf{n}$. In this paper, the expressions E are just variables and nil. This is enough to illustrate our reasoning about the update commands. Extending the expression language to include, for example, arithmetic, does not our reasoning. is given by:

$$[\![n]\!]\mathsf{s} = \mathsf{s}(n) \qquad [\![\text{nil}]\!]\mathsf{s} = 0$$

Definition 3.14 (Commands for Heap Update) *The commands of our heap update language are given by the grammar:*

$$
\begin{aligned}
\mathbb{C} ::= \; & n := E && \textit{variable assignment} \\
& \mathbb{C}_{up}(E) && \textit{update at location } E \\
& n := new() && \textit{new} \\
& \mathbb{C} \; ; \; \mathbb{C} && \textit{sequencing}
\end{aligned}
$$

The heap update commands $\mathbb{C}_{up}(E)$ *acting at location* E *are defined as follows:*

$$
\begin{aligned}
\mathbb{C}_{up}(E) ::= \; & dispose(E) && \textit{dispose} \\
& [E] := F && \textit{mutation} \\
& n := [E] && \textit{lookup}
\end{aligned}
$$

The set free($\mathbb{C}$) is the set of variables occurring in $\mathbb{C}$. The set mod($\mathbb{C}$) is $\{n\}$ for variable assignment and lookup and new, $\emptyset$ for the other atomic commands, and $mod(\mathbb{C}_1) \cup mod(\mathbb{C}_2)$ for $\mathbb{C}_1 \; ; \; \mathbb{C}_2$.

The heap update commands are similar to the tree update commands. The main difference is that heap locations contain unstructured values whilst tree locations contain trees. The heap dispose command corresponds to the tree dispose command operating at the tree level, since in both cases the node itself is deleted. The other heap update commands correspond to tree update commands operating at the subforest level, since subforests play the role of the contents of a node. Heap cell mutation $[E] := F$ corresponds to dispose of the subforest $[n]_{\mathrm{SF}} := 0$ followed by append $[n]_{\mathrm{SF}} \mathrel{*}= x$. Heap lookup $n' := [E]$ corresponds to tree lookup $x := [n]_{\mathrm{SF}}$. The new command for trees differs from the one for heaps in that it specifies the location where the new tree is added. Keeping these differences in mind, we shall see that there are remarkable similarities in the small axioms, weakest preconditions, and derivations for the heap commands and the corresponding tree commands.

$$\frac{[\![E]\!]\mathsf{s} = \mathsf{n}}{n := E, \mathsf{s}, \mathsf{h} \rightsquigarrow [\mathsf{s}|n \leftarrow \mathsf{n}], \mathsf{h}}$$

$$\frac{[\![E]\!]\mathsf{s} = \mathsf{n} \quad \mathsf{h} = \mathsf{h}' \cdot \mathsf{n} \mapsto \mathsf{n}'}{\mathrm{dispose}(E), \mathsf{s}, \mathsf{h} \rightsquigarrow \mathsf{s}, \mathsf{h}'} \qquad \frac{[\![E]\!]\mathsf{s} = \mathsf{n} \quad \mathsf{h} = \mathsf{h}' \cdot \mathsf{n} \mapsto \mathsf{n}' \quad [\![F]\!]\mathsf{s} = \mathsf{n}''}{[E] := F, \mathsf{s}, \mathsf{h} \rightsquigarrow \mathsf{s}, \mathsf{h}' \cdot \mathsf{n} \mapsto \mathsf{n}''}$$

$$\frac{[\![E]\!]\mathsf{s} = \mathsf{n} \quad \mathsf{h} = \mathsf{h}' \cdot \mathsf{n} \mapsto \mathsf{n}'}{n := [E], \mathsf{s}, \mathsf{h} \rightsquigarrow [\mathsf{s}|n \leftarrow \mathsf{n}'], \mathsf{h}} \qquad \frac{\mathsf{n} \notin \mathrm{dom}(\mathsf{h})}{n := \mathrm{new}(), \mathsf{s}, \mathsf{h} \rightsquigarrow [\mathsf{s}|n \leftarrow \mathsf{n}], \mathsf{h} \cdot \mathsf{n} \mapsto \mathsf{nil}}$$

$$\frac{\mathbb{C}_1, \mathsf{s}, \mathsf{h} \rightsquigarrow \mathbb{C}', \mathsf{s}', \mathsf{h}'}{(\mathbb{C}_1 \,;\, \mathbb{C}_2), \mathsf{s}, \mathsf{h} \rightsquigarrow (\mathbb{C}' \,;\, \mathbb{C}_2), \mathsf{s}', \mathsf{h}'} \qquad \frac{\mathbb{C}_1, \mathsf{s}, \mathsf{h} \rightsquigarrow \mathsf{s}', \mathsf{h}'}{(\mathbb{C}_1 \,;\, \mathbb{C}_2), \mathsf{s}, \mathsf{h} \rightsquigarrow \mathbb{C}_2, \mathsf{s}', \mathsf{h}'}$$

$$\frac{[\![E]\!]\mathsf{s} = \mathsf{n} \quad \mathsf{h} \neq \mathsf{h}' \cdot \mathsf{n} \mapsto \mathsf{n}'}{\mathbb{C}_{\mathrm{up}}(E), \mathsf{s}, \mathsf{h} \rightsquigarrow \mathrm{fault}} \qquad \frac{\mathbb{C}_1, \mathsf{s}, \mathsf{h} \rightsquigarrow \mathrm{fault}}{(\mathbb{C}_1 \,;\, \mathbb{C}_2), \mathsf{s}, \mathsf{h} \rightsquigarrow \mathrm{fault}}$$

$[s \mid n \leftarrow \mathsf{n}]$ denotes the store s updated with $s(n) = \mathsf{n}$.
$h \cdot h'$ denotes the composition of heaps and $\mathsf{n} \mapsto \mathsf{n}'$ denotes a unary heap.

Fig. 6. Operational Semantics for Heap Update

Definition 3.15 (Operational Semantics for Heap Update) *The operational semantics of the heap update language is given in Figure 6.*

CL for Heap Update

CL-reasoning for heap update is similar to CL-reasoning for multisets given in section 2.3.2, except that we now incorporate unary heaps, expressions, expression equality and quantification since they are necessary for our Hoare reasoning.

Definition 3.16 [CL for Heap Update] CL for heap update, denoted CL_{Heap}, consists of CL_{Heap}-formulae constructed by extending the CL_0-formulae defined inductively by the grammars in definition 2.10 with the following additional cases:
data formulae

$$P ::= E \mapsto F \mid E = F \quad \text{specific data formulae}$$
$$\exists n.\, P \qquad\qquad \text{quantification, } n \in Var_{\mathbb{N}}$$

context formulae

$$K ::= \exists n.\, K \quad \text{quantification, } n \in Var_{\mathbb{N}}$$

$$\{(E = n_1) \land 0\} \ n := E \qquad \{(n = n_1) \land 0\}$$

$$\{E \mapsto \_ \} \ \text{dispose}(E) \ \{0\}$$

$$\{E \mapsto \_ \} \ [E] := F \quad \{E \mapsto F\}$$

$$\{(E = n_1) \land (n_1 \mapsto n_2)\} \ n := [E] \quad \{(n = n_2) \land (n_1 \mapsto n_2)\}$$

$$\{0\} \ n := \text{new}() \ \{n \mapsto \text{nil}\}$$

$$\text{where } n_1, n_2 \notin \text{mod}(\mathbb{C})$$

Fig. 7. Small Axioms for Heap Update

The extension of the CL_0-satisfaction relation is given by:

$$Heap, \mathsf{s}, \mathsf{h} \vDash_{\mathcal{P}} E \mapsto F \text{ iff } \mathsf{h} = [\![E]\!]\mathsf{s} \mapsto [\![F]\!]\mathsf{s}$$

$$Heap, \mathsf{s}, \mathsf{h} \vDash_{\mathcal{P}} E = F \text{ iff } [\![E]\!]\mathsf{s} = [\![F]\!]\mathsf{s}$$

We define derived formulae used in the Hoare reasoning. We write $E \mapsto \_$ for $\exists n. E \mapsto n$, and $E \hookrightarrow F$ for $(E \mapsto F) * \text{true}$, where the CL_0-derived formula $P * Q$ is given in Definition 2.16 and corresponds to $*$ in SL for the heap model.

Local Hoare Reasoning about Heap Update
We give the small axioms, weakest precondition axioms, and derivations of the weakest preconditions axioms from the small axioms for the heap commands.

Lemma 3.17 (Locality for Heap Update) *All the commands in our heap update language are local.*

Definition 3.18 (Small Axioms for Heap Update) *The Small Axioms for the atomic heap update commands are given in Figure 7.*

Notice the similarity between the small axioms for the heap update commands and the small axioms for tree update commands given in Figure 3, bar the obvious variations due to the variation in the commands. We shall see that this similarity also occurs in the weakest preconditions and derivations. We illustrate this similarity for the dispose heap command $\text{dispose}(E)$ which is analogous to the dispose tree command $[n]_{\text{T}} := 0$. The small axioms for tree and heap dispose are:

$$\{E \mapsto \_ \} \ \text{dispose}(E) \ \{0\} \qquad \{n[\text{true}]\} \ [n]_{\text{T}} := 0 \ \{0\}$$

With the heap command $\text{dispose}(E)$, the precondition states that the heap is unary with expression E denoting the node address and $\_$ denoting the value of the address

37

$$\{P[E/n]\} \; n := E \qquad \{P\}$$

$$\{P * (E \mapsto \_)\} \; \mathrm{dispose}(E) \; \{P\}$$

$$\{((E \mapsto F) \ast\!\!-\!\!\ast P) * (E \mapsto \_)\} \; [E] := F \qquad \{P\}$$

$$\{\exists n_2.((E \hookrightarrow n_2) \wedge P[n_2/n])\} \; n := [E] \qquad \{P\}$$

$$\{\forall n.\, (n \mapsto \mathrm{nil}) \ast\!\!-\!\!\ast P\} \; n := \mathrm{new}() \; \{P\}$$

$$\text{where } n_2 \notin \mathrm{free}(\mathbb{C}) \cup \mathrm{free}(P)$$

Fig. 8. Weakest Preconditions for Heap Update

which is not important. With the tree command $[n]_{\mathrm{T}} := 0$, the precondition states that the tree has top node n and the subforest underneath is not important.

Theorem 3.19 (Weakest Precondition Axioms for Heap Update) *The weakest precondition axioms for the atomic heap update commands are given in Figure 8.*

The weakest precondition axioms for the heap and tree dispose commands are:

$$\{P * (E \mapsto \_)\} \; \mathrm{dispose}(E) \; \{P\} \qquad \{(0 \rhd P)(n[\mathrm{true}])\} \; [n]_{\mathrm{T}} := 0 \; \{P\}$$

The connection is immediate, since $P * Q$ in CL_0 is a shorthand for $(0 \rhd P)(Q)$ and $E \mapsto \_$ is analogous to $n[\mathrm{true}]$. Not only that, but the derivations of these weakest precondition axioms are exactly analogous.

Lemma 3.20 (Derivability of Weakest Precondition Axioms) *The weakest preconditions in Thm. 3.19 are derivable from the small axioms in Figure 7 and the proof rules in Figure 1.*

Proof. See Figure 9. □

3.4 Hoare Reasoning about Term Rewriting

We now apply our local Hoare reasoning to term rewriting systems. We consider rewrite rules as atomic commands. Rewrite rules are not typically regarded as local commands since they may apply to a number of redexes. They are however local in the sense that once the redex has been identified, then only the redex is affected by the rewrite. We formalize this local behaviour by considering *located terms*, where each occurrence of a function symbol f is annotated with a unique location $n \in N$ for N defined as for $LocTree_N$.

Definition 3.21 (Located Terms) *The* CL-*model* $LocTerm_{\Sigma_N}$ *of located terms generated from signature Σ and node identifiers N is the tuple* $(\mathcal{T}_{\Sigma_N}, \mathcal{C}_{\Sigma_N}, ap, \{\_\})$ *where $\mathcal{T}_{\Sigma_N}$ denotes the data set of* located *terms constructed from indexed function symbols $f_n : r$ where $f : r \in \Sigma$ and $n \in N$ is unique in the terms, $\mathcal{C}_{\Sigma_N}$ is the*

corresponding set of contexts, ap denotes the partial application of contexts to terms, and _ denotes the empty context.

Term Rewriting Update Language

Our data storage model consists of a working term t and a store s. The store is again a total function from node variables and term variables to values:

$$\text{node variables} \quad Var_N = \{n, m, \dots\}$$

$$\text{term variables} \quad Var_{\mathcal{T}_{\Sigma_N}} = \{x, y, \dots\}$$

$$\text{stores} \quad s \in (Var_N \to N) \times (Var_{\mathcal{T}_{\Sigma_N}} \to \mathcal{T}_{\Sigma_N})$$

We use the notation $[s \mid n \leftarrow n]$ and $[s \mid x \leftarrow t]$ as before. We also give expressions which will be used to define the rewrite commands:

$$\text{pre-expressions} \quad E, F ::= x \mid f_n(E, \dots, E)$$

The sets $free_N(E)$ and $free_{\mathcal{T}}(E)$ denote the free node variables and tree variables in E, and are standard. The set of *expressions* are those pre-expressions with linear occurrences of the node variables. The semantics of expressions is a partial function due to uniqueness of node identifiers:

$$[\![x]\!]s = s(x)$$

$$[\![f_n(E_1, \dots, E_r)]\!]s = \begin{cases} f_{s(n)}([\![E_1]\!]s, \dots, [\![E_r]\!]s) & \text{if a well-defined located term} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Definition 3.22 (Commands for Term Rewriting) *The commands for our term rewriting language are given by the grammar:*

$$\mathbb{C} ::= E \to F \quad \text{rewriting rule}$$

$$\mathbb{C} \,;\, \mathbb{C} \quad \text{sequencing}$$

with the following restrictions:

(i) *rewrite rule restrictions: E is not a variable and $free_{\mathcal{T}}(F) \subseteq free_{\mathcal{T}}(E)$;*

(ii) *linearity restriction on term variables due to uniqueness of location names: each $x \in free_{\mathcal{T}}(E)$ occurs once in E and each $x \in free_{\mathcal{T}}(F)$ occurs once in F.*

The set $free\,(E \to F)$ is $free_N(E) \cup free_N(F)$ and $mod(\mathbb{C})$ is $free_N(F) - free_N(E)$. Since term variables are only used internally for pattern matching with the rewrite command, they are neither modified nor free.

The command $E \to F$ acts on the working term t and the store s by finding values for the term variables in E, such that E with those values evaluates to a subterm of t, and then replacing that subterm with the one generated by substituting the values

into F, with fresh values assigned to F's fresh location variables. For example, the execution of the rewrite command $f_n(x, y) \rightarrow g_n(x, h_m(y))$ on the working term $h_{n_1}(f_{n_2}(c_{n_3}, c_{n_4}))$ in a store where $n = n_2$ results in the term $h_{n_1}(g_{n_2}(c_{n_3}, h_{n_5}(c_{n_4})))$ where n_5 is a fresh node value assigned to m in the store. Notice that n_5 must be fresh for the resulting term to be well-formed.

Definition 3.23 (Operational Semantics for Term Rewriting) *The operational semantics of the term rewriting commands is given in Figure 10.*

Going back to our specific rewriting example, notice that the operational semantics only temporarily assigns term variable x to c_{n_3} and y to c_{n_4}. This is because the term variables in a rewriting command are bound in that command, in the sense that they are only used to describe the matching of that command to a subterm of the working term.

CL for Term Rewriting
CL for term rewriting is similar to CL for terms (section 2.3.4), except that it is extended by expressions and quantification.

Definition 3.24 [CL for Term Rewriting] CL applied to the term model $LocTerm_{\Sigma_N}$, denoted $\mathrm{CL}_{LocTerm_{\Sigma_N}}$, consists of $\mathrm{CL}_{LocTerm_{\Sigma_N}}$-formulae constructed by extending the CL-formulae defined inductively by the grammars in definition 2.1 with the following additional cases:
 data formulae

$$P ::= E \qquad\qquad \text{specific data formulae}$$
$$\exists n.\, P \mid \exists x.\, P \quad \text{quantification, } n \in Var_N,\ x \in Var_{\mathcal{T}_N}$$

 context formulae

$$K ::= f_n(P_1, \ldots, K, \ldots, P_r) \quad \text{specific context formulae, } f : r \in \Sigma, n \in N$$
$$\exists n.\, K \mid \exists x.\, K \qquad\qquad \text{quantification, } n \in Var_N,\ x \in Var_{\mathcal{T}_N}$$

The extension of the satisfaction relation is given by:

$$LocTerm_{\Sigma_N}, s, t \vDash_{\mathcal{P}} E \qquad\qquad\qquad \text{iff } \llbracket E \rrbracket s = t$$
$$LocTerm_{\Sigma_N}, s, t \vDash_{\mathcal{K}} f_n(P_1, \ldots, K, \ldots, P_r) \text{ iff } t = f_{\llbracket n \rrbracket s}(t_1, \ldots, \_, \ldots, t_r) \text{ and}$$
$$LocTerm_{\Sigma_N}, s, t_i \vDash_{\mathcal{P}} P_i,\ i \in \{1, \ldots, r\}$$

Local Hoare Reasoning about Term Rewriting
We give the small axiom and weakest precondition axiom for the atomic term rewriting command, and derive the latter from the former as with our previous update examples.

Lemma 3.25 (Locality for Term Rewriting) *The commands in our term rewriting language are local.*

Definition 3.26 (Small Axiom for Term Rewriting) *The small axiom for command $E \to F$ is simply*

$$\{E\}\ E \to F\ \{F\}$$

Theorem 3.27 (Weakest Precondition Axiom for Term Rewriting) *The weakest precondition axiom for command $E \to F$ is*

$$\{\exists \tilde{x}'.(\forall \tilde{m}.(F[\tilde{x}'/\tilde{x}] \rhd P))(E[\tilde{x}'/\tilde{x}])\}\ E \to F\ \{P\}$$

where $\{\tilde{x}\} = \mathit{free}_{\mathcal{T}}(E)$, $\{\tilde{m}\} = \mathit{free}_N(F) - \mathit{free}_N(E)$, and $\{\tilde{x}'\} \cap \mathit{free}_{\mathcal{T}}(P) = \emptyset$.

The substitution $\tilde{x}'/\tilde{x}$ reflects the fact that the term variables $\tilde{x}$ in E are bound in the rewrite command, and can hence be renamed in the logic. The universal quantification for the $\tilde{m}$ is necessary, because the $\tilde{m}$ can be assigned any fresh value with freshness being guaranteed by the well-formedness of terms.

Lemma 3.28 (Derivability of Weakest Precondition Axiom) *The weakest precondition in Thm. 3.27 is derivable from the small axiom in Defn. 3.26 and the proof rules in Figure 1.*

Proof. See Figure 11.   □

Variable Assignment

$$\frac{\dfrac{\{(E = n_1) \wedge 0\}\ n := E\ \{(n = n_1) \wedge 0\}}{\{P[n_1/n] * ((E = n_1) \wedge 0))\}\ n := E\ \{P[n_1/n] * ((n = n_1) \wedge 0)\}}\ \textsc{Frame}}{\{P[E/n] \wedge E = n_1\}\ n := E\ \{P\}}\ \textsc{Cons}$$

$$\frac{}{\{P[E/n]\}\ n := E\ \{P\}}\ \textsc{Cons/Vars}$$

Dispose

$$\frac{\dfrac{\{E \mapsto \_\}\ \mathrm{dispose}(E)\ \{0\}}{\{P * (E \mapsto \_)\}\ \mathrm{dispose}(E)\ \{P * 0\}}\ \textsc{Frame}}{\{P * (E \mapsto \_)\}\ \mathrm{dispose}(E)\ \{P\}}\ \textsc{Cons}$$

Mutation

$$\frac{\dfrac{\{E \mapsto \_\}\ [E] := F\ \{E \mapsto F\}}{\{((E \mapsto F) \mathbin{-\!\!*} P) * (E \mapsto \_)\}\ [E] := F\ \{((E \mapsto F) \mathbin{-\!\!*} P) * (E \mapsto F)\}}\ \textsc{Frame}}{\{((E \mapsto F) \mathbin{-\!\!*} P) * (E \mapsto \_)\}\ [E] := F\ \{P\}}\ \textsc{Cons}$$

Lookup

$$\frac{\{(E = n_1) \wedge (n_1 \mapsto n_2)\}\ n := [E]\ \{(n = n_2) \wedge (n_1 \mapsto n_2)\}}{\left\{\begin{array}{l}((n_1 \mapsto n_2) \mathbin{-\!\!*} P[n_2/n]) \\ * ((E = n_1) \wedge (n_1 \mapsto n_2))\end{array}\right\}\ n := [E]\ \left\{\begin{array}{l}((n_1 \mapsto n_2) \mathbin{-\!\!*} P[n_2/n]) \\ * ((n = n_2) \wedge (n_1 \mapsto n_2))\end{array}\right\}}\ \textsc{Frame}$$

$$\frac{\{(n_1 \hookrightarrow n_2) \wedge P[n_2/n] \wedge (E = n_1)\}\ n := [E]\ \{P[n_2/n] \wedge (n = n_2)\}}{\{(E \hookrightarrow n_2) \wedge P[n_2/n] \wedge (E = n_1)\}\ n := [E]\ \{P\}}\ \textsc{Cons}$$

$$\frac{}{\{\exists n_2.((E \hookrightarrow n_2) \wedge P[n_2/n])\}\ n := [E]\ \{P\}}\ \textsc{Cons/Vars}$$

New

$$\frac{\dfrac{\{0\}\ n := \mathrm{new}()\ \{n \mapsto \mathrm{nil}\}}{\{(\forall n.\ (n \mapsto \mathrm{nil}) \mathbin{-\!\!*} P) * 0\}\ n := \mathrm{new}()\ \{(\forall n.\ (n \mapsto \mathrm{nil}) \mathbin{-\!\!*} P) * (n \mapsto \mathrm{nil})\}}\ \textsc{Frame}}{\{\forall n.\ (n \mapsto \mathrm{nil}) \mathbin{-\!\!*} P)\}\ n := \mathrm{new}()\ \{P\}}\ \textsc{Cons}$$

Fig. 9. Derivations of the Weakest Preconditions for Heap Update

$$\frac{\mathsf{t} = \mathrm{ap}(\mathsf{c}, \mathsf{t}_1) \quad \mathsf{t}_1 = [\![E]\!][\mathsf{s} \mid \tilde{x} \leftarrow \tilde{\mathsf{t}}] \quad \mathsf{t}_2 = [\![F]\!][\mathsf{s} \mid \tilde{x} \leftarrow \tilde{\mathsf{t}}, \tilde{m} \leftarrow \tilde{\mathsf{m}}] \quad \mathsf{t}' = \mathrm{ap}(\mathsf{c}, \mathsf{t}_2)}{E \to F, \mathsf{s}, \mathsf{t} \rightsquigarrow [\mathsf{s} | \tilde{m} \leftarrow \tilde{\mathsf{m}}], \mathsf{t}'}$$

$$\frac{\mathbb{C}_1, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathbb{C}', \mathsf{s}', \mathsf{t}'}{(\mathbb{C}_1 \; ; \; \mathbb{C}_2), \mathsf{s}, \mathsf{t} \rightsquigarrow (\mathbb{C}' \; ; \; \mathbb{C}_2), \mathsf{s}', \mathsf{t}'} \qquad \frac{\mathbb{C}_1, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathsf{s}', \mathsf{t}'}{(\mathbb{C}_1 \; ; \; \mathbb{C}_2), \mathsf{s}, \mathsf{t} \rightsquigarrow \mathbb{C}_2, \mathsf{s}', \mathsf{t}'}$$

$$\frac{\mathsf{t} \neq \mathrm{ap}(\mathsf{c}, [\![E]\!][\mathsf{s} \mid \tilde{x} \leftarrow \tilde{\mathsf{t}}])}{E \to F, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathrm{fault}} \qquad \frac{\mathbb{C}_1, \mathsf{s}, \mathsf{t} \rightsquigarrow \mathrm{fault}}{(\mathbb{C}_1 \; ; \; \mathbb{C}_2), \mathsf{s}, \mathsf{t} \rightsquigarrow \mathrm{fault}}$$

where $\{\tilde{x}\} = \mathrm{free}_{\mathcal{T}}(E)$ and $\tilde{m} = \mathrm{free}_N(F) - \mathrm{free}_N(E)$.

Fig. 10. Operational Semantics for Term Rewriting

Rewrite

$$\frac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\{E\} \ E \to F \ \{F\}}{\{(I \wedge \tilde{x}'{=}\tilde{x})(E)\} \ E \to F \ \{(I \wedge \tilde{x}'{=}\tilde{x})(F)\}}}{\{\tilde{x}'{=}\tilde{x} \wedge E[\tilde{x}'/\tilde{x}]\} \ E \to F \ \{\tilde{x}'{=}\tilde{x} \wedge F[\tilde{x}'/\tilde{x}]\}}}{\{E[\tilde{x}'/\tilde{x}]\} \ E \to F \ \{F[\tilde{x}'/\tilde{x}]\}}}{\{(\forall \tilde{m}.(F[\tilde{x}'/\tilde{x}] \rhd P))(E[\tilde{x}'/\tilde{x}])\} \ E \to F \ \{(\forall \tilde{m}.(F[\tilde{x}'/\tilde{x}] \rhd P))(F[\tilde{x}'/\tilde{x}])\}}}{\{\exists \tilde{x}'.(\forall \tilde{m}.(F[\tilde{x}'/\tilde{x}] \rhd P))(E[\tilde{x}'/\tilde{x}])\} \ E \to F \ \{P\}}}$$

$$\begin{array}{l} \text{Frame} \\ \text{Cons} \\ \text{Vars/Cons} \\ \text{Frame} \\ \text{Cons/Vars} \end{array}$$

Fig. 11. Derivation of the Weakest Precondition Axiom for Term Rewriting

4   Conclusions

We have given a detailed account of CL for reasoning about structured data, and have compared CL_0-reasoning with BL-reasoning. We have analysed several examples of structured data: sequences and trees where CL_0-reasoning is stronger than BL-reasoning, multisets and heaps where CL_0- and BL-reasoning is the same, and terms where only CL-reasoning is feasible. We believe these examples show that reasoning about contexts is important for reasoning about structured data. We have chosen to present the simplest version of CL necessary in order to present our local Hoare reasoning. There are of course several natural extensions such as context composition [16], multi-holed contexts, and binding contexts, which we will study as the application demands.

We have presented a framework for local Hoare reasoning about data update using CL, which we have applied to tree update, heap update and term rewriting. This work is a straightforward adaptation of the local reasoning agenda first established by O'Hearn, Reynolds and Yang in [10], giving us confidence that our CL-reasoning is the right approach. For tree update, it is possible to describe the small axioms and frame rule using BL for trees (static AL). It is however not possible to define the weakest preconditions. In this paper, we illustrate this by example. We show that the weakest precondition for tree dispose cannot be expressed using BL-reasoning, since even simple postconditions require a case-by-case analysis which result in a very different structure for each case. In [2], we prove the formal inexpressivity result.

Our original motivation for reasoning about tree update was to reason about XML update. In this paper, we focus on a simple imperative language for manipulating a simple tree model, which is expressive enough to illustrate the subtleties of reasoning about tree update. In [6], Gardner and Zarfaty study local Hoare reasoning for a more substantial tree-update language which combines update commands with queries. They describe local Hoare reasoning for these more complex commands, but only at the expense of loosing the small axioms. Small-axiom reasoning should still be possible, since these complex commands have well-defined footprints. It may be possible to regain the small-axiom approach, by using more complex contexts involving a mixture of multi-holed contexts and wiring. Indeed, Sassone *et al.* have highlighted Milner's bigraphs [8] as a good model for XML precisely because of the additional context structure. They study BiLog [5], a logic for static bigraphs influenced in part by CL, but do not extend their reasoning to tree update.

In our examples of data update, the CL-reasoning and local Hoare reasoning are intriguingly similar. This suggests the possibility of unified reasoning about data update. For data defined inductively by a grammar, we intuitively know how to give the corresponding CL-theory. We should be able to formalise this intuition, by providing a uniform way of generating data, contexts and CL-formulae from the same underlying signature. It remains to be seen whether this idea can be expanded to generate a unified theory of local Hoare reasoning.

References

[1] Calcagno, C., P. Gardner and U. Zarfaty, *Context logic and tree update*, in: *POPL*, 2005.

[2] Calcagno, C., P. Gardner and U. Zarfaty, *Context logic as modal logic: Completeness and parametric expressivity* (2006), submitted.

[3] Cardelli, L. and A. Gordon, *Anytime, anywhere: Modal logics for mobile ambients*, in: *POPL*, 2000.

[4] Cardelli, L. and A. Gordon, *Logical properties of name restriction*, in: *TLCA*, LNCS 2044, 2001.

[5] Conforti, G., D. Macedonio and V. Sassone, *Spatial logics for bigraphs*, in: *ICALP, LNCS 3520*, 2005.

[6] Gardner, P. and U. Zarfaty, *Local reasoning about tree update*, in: *MFPS*, 2006.

[7] Ishtiaq, S. and P. O'Hearn, *BI as an assertion language for mutable data structures*, in: *POPL*, 2001.

[8] Jensen, O. and P. Milner, *Bigraphs and mobile processes (revised)*, Technical report, University of Cambridge (2004).

[9] O'Hearn, P. and D. Pym, *Logic of bunched implications*, Bulletin of Symbolic Logic **5** (1999), pp. 215–244.

[10] O'Hearn, P., J. Reynolds and H. Yang, *Local reasoning about programs that alter data structures*, in: L. Fribourg, editor, *CSL 2001* (2001), pp. 1–19, LNCS 2142.

[11] Pym, D., P. O'Hearn and H. Yang, *Possible worlds and resources: The semantics of BI*, Theoretical Computer Science **315** (2004), pp. 257–305.

[12] Pym, D. J., "The Semantics and Proof Theory of the Logic of Bunched Implications," Applied Logic Series **26**, Kluwer Academic Publishers, 2002.

[13] Reynolds, J., *Separation logic: a logic for shared mutable data structures* (2002), invited Paper, LICS'02.

[14] Simpson, A., "The Proof Theory and Semantics of Intuitionistic Modal Logic," Ph.D. thesis, PhD Thesis, University of Edinburgh (1993).

[15] Yang, H. and P. O'Hearn, *A semantic basis for local reasoning* (2002), fOSSACS.

[16] Zarfaty, U., *Context logic and tree update*, PhD thesis. In preparation.