# Context Logic and Tree Update[*]

Cristiano Calcagno    Philippa Gardner    Uri Zarfaty

Department of Computing
Imperial College
University of London

## ABSTRACT

Spatial logics have been used to describe properties of tree-like structures (Ambient Logic) and in a Hoare style to reason about dynamic updates of heap-like structures (Separation Logic). We integrate this work by analyzing dynamic updates to tree-like structures with pointers (such as XML with identifiers and idrefs). Naïve adaptations of the Ambient Logic are not expressive enough to capture such local updates. Instead we must explicitly reason about arbitrary tree contexts in order to capture updates throughout the tree. We introduce Context Logic, study its proof theory and models, and show how it generalizes Separation Logic and its general theory BI. We use it to reason locally about a small imperative programming language for updating trees, using a Hoare logic in the style of O'Hearn, Reynolds and Yang, and show that weakest preconditions are derivable. We demonstrate the robustness of our approach by using Context Logic to capture the locality of term rewrite systems.

## Categories and Subject Descriptors

D.2.4 [**Software/Program verification**]: Correctness proofs, Formal methods, Validation; F.3.1 [**Specifying and Verifying and Reasoning about Programs**]: Logics of programs

## General Terms

Languages, theory, verification

## Keywords

tree update, Hoare Logic, contexts

## 1. INTRODUCTION

We study Hoare logics for reasoning about data update. Hoare logics have been well-explored for heap update, from the original work of Hoare based on first-order logic to the recent work of O'Hearn, Reynolds and Yang [9, 12, 6], with their emphasis on local reasoning using the Separation Logic. Such Hoare reasoning has hardly been explored for other data structures. We show that the techniques for reasoning locally about heap update can be adapted to reason locally about tree update (XML update). This adaptation was by no means straightforward. Surprisingly, the Ambient Logic for trees cannot be used as the basis for the Hoare triples, since the weakest preconditions are not expressible. Instead, we had to fundamentally change the way we reason about structured data, using *Context Logic* to analyse both data and contexts.

Data update typically identifies the portion of data to be replaced, removes it, and inserts the new data *in the same place*. This place of insertion is essential for reasoning about updates. Context Logic has context application—data insertion in a context—as its central construct, plus (adjunct) connectives for reasoning hypothetically about data insertion. This shift of perspective emerged from our study of tree update, in particular realising that the Ambient Logic could not directly describe tree insertion. Given the conceptual nature of our context reasoning, we expect the same approach to apply to a wide range of structured data.

In this paper, we provide a Hoare logic for reasoning locally about a simple imperative language for tree update, using our Context Logic specialised to trees. We prove that the weakest preconditions are derivable. We study the general theory of Context Logics, providing a Hilbert-style proof theory, forcing semantics and models. The Bunched Logic (BI) of O'Hearn and Pym gives the general theory of Separation Logic. We show that BI can be obtained from Context Logic by collapsing some of the structure. We prove soundness of our proof theory with respect to the forcing semantics. Calcagno and Yang have recently proved completeness, by adapting Yang's results for BI [11].

We also adapt our Hoare logic reasoning about trees to heap update and term rewriting. Our reasoning about heap update corresponds precisely to O'Hearn *et al.*'s reasoning using Separation Logic, as one would expect since heap contexts are simple. Our reasoning about term rewriting demonstrates the robustness of our approach[1]. Although terms in rewrite systems can be seen as special cases of trees, there is a crucial difference: terms over a signature do not decompose as a composition of subterms, due to the fixed arity of function symbols. They do however decompose

[*]A preliminary version of this work appeared in an informal proceedings [3].

[1]Many thanks to Peter O'Hearn for suggesting this open problem.

nicely as context/subtree pairs. This example demonstrates the fundamental importance of contexts for reasoning locally about structured data.

## *Heap Update*

O'Hearn and Reynolds have introduced a style of Hoare logic for imperative programs consisting of small *local axioms*, which specify properties about the portion of the heap accessed by a command, and a *frame rule*, which uniformally extends the reasoning to properties about the rest of the heap. The impact of their approach is that the resulting Hoare logic is very simple, and has been applied to several problems—such as pointer arithmetic, concurrent imperative programs and passivity—which escaped reasoning in the traditional Hoare-logic style. Their primary innovation was to base their Hoare logic on the Separation Logic, consisting of standard first-order connectives and additional formulae for directly analysing the heap structure. In particular, two key logical constructs are the *separating conjunction* $*$ for describing disjoint properties about the heap and used to formulate the frame rule, and its adjoint $-\!\!*$ for analysing extensions of the heap and used to express the weakest preconditions.

Consider the heap update command $[n] = v$, which updates address $n$ in the heap with value $v$. The corresponding small axiom for this command is

$$\{n \mapsto -\}\ [n] = v\ \{n \mapsto v\}$$

The precondition states that the heap consists of one cell with address $n$ and an unspecified value, and the postcondition states that the cell now has value $v$. This small axiom only describes properties about the specific cell $n$. To extend to properties about a larger heap, we use the frame rule to derive the triple

$$\{P * (n \mapsto -)\}\ [n] = v\ \{P * (n \mapsto v)\}$$

The assertion $P * (n \mapsto -)$ states that the heap can be split disjointedly into the cell $n$ with an arbitrary value (the ordering of cells does not matter), and the rest of the heap with property $P$ which is unaffected by the update command. The postcondition therefore has the same structure, with the now updated cell $n$ and the rest of the heap satisfying $P$.

The small axioms and frame rule are elegant, and intuitively express the behaviour of commands. In addition, the weakest preconditions are derivable, a natural requirement which is essential for providing verification tools. The weakest preconditions use assertions of the form $R -\!\!* Q$, which states that whenever a heap is extended by a heap satisfying $R$ then the resulting heap satisfies $Q$. For example, the formula $(n \mapsto v) -\!\!* Q$ states that, whenever the heap is extended by a cell with address $n$ and value $v$, then property $Q$ must hold. This formula is used to define the weakest precondition of our update command:

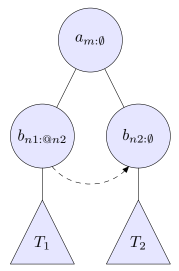
$$\{((n \mapsto v) -\!\!* Q) * (n \mapsto -)\}\ [n] = v\ \{Q\}$$

The precondition states that the heap can be split into the cell $n$ with unspecified value and the rest of the heap which, when extended by a cell $n$ with specific value $v$, satisfies property $Q$.

## *Tree Update*

We apply this style of Hoare logic reasoning about heap update to tree update. It is possible to describe analogous small axioms and frame rule using (an adaptation of) the Ambient Logic. In order to derive the weakest preconditions however, we must work with Context Logic, as we illustrate in Section 5.

Consider the following picture of a tree, the corresponding XML format and the syntactic description we use in this paper:



**XML format:**

```
<a ID=m>
  <b ID=n1 IDREF=n2>
    T1
  </b>
  <b ID=n2>
    T2
  </b>
</a>
```

$$a_{m:\emptyset}[b_{n1:@n2}[T_1] \mid b_{n2:\emptyset}[T_2]]$$

The structure highlights the unique node addresses (XML identifiers) and the arbitrary cross-links (pointers, XML idrefs), and is similar to the trees studied in [1] except that we permit dangling pointers. In [4], Gardner *et al.* use the Ambient Logic to specify static properties of such trees: the vertical path structure (path expressions), the horizontal structure (XML schema) and properties about pointers (types for XML idrefs). Here we reason about tree update.

Consider an update command for trees $[n] = T$, which replaces the subtree at address $n$ with the tree $T$. The small axiom for this command is

$$\{a_{n:v}[\text{true}]\}\ [n] = T\ \{a_{n:v}[T]\}$$

The precondition states that the tree consists of a top node identified by address $n$ with value $v$ and an unspecified subtree. The postcondition says that the subtree has been replaced by $T$. In order to extend the small axiom to triples stating properties about larger trees, we use a generalised frame rule based on context application to derive the triple

$$\{K(a_{n:v}[\text{true}])\}[n] = T\ \{K(a_{n:v}[T])\}$$

The precondition states that the tree can be split disjointedly into a tree with top node $a_{n:v}$, and a context satisfying context formula $K$ which is unaffected by the update command. The postcondition therefore has the same structure, with the subtree at address $n$ updated.

To specify the weakest precondition, we use a context adjunct $P \triangleright Q$ which describes a property about contexts: whenever a tree satisfying $P$ is placed in the context hole, then the resulting tree satisfies $Q$. For example, the adjoint $a_{n:v}[T] \triangleright Q$ states that, whenever a tree with address $n$, value $v$ and subtree $T$ is placed in the context hole, then the property $Q$ must hold. The weakest precondition of our update command is:

$$\{(a_{n:v}[T] \triangleright Q)(a_{n:v}[\text{true}])\}\ [n] = T\ \{Q\}$$

The precondition says that the tree can be split into a subtree with top node $a_{n:v}$, and a context which satisfies $Q$ when tree $a_{n:v}[T]$ is put in the hole. Again, this triple is derivable.

In contrast, we believe that it is not possible to express such weakest preconditions using (a minor adaptation of) the Ambient Logic to the trees-with-pointers model. We can prove that the Ambient Logic cannot express the weakest preconditions without some form of recursion. The somewhere modality does provide a limited form of context reasoning, but in Section 5 we illustrate that this reasoning is not enough to describe the parametric weakest preconditions. It remains future work to pin down this inexpressivity result.

## 2. TREE DATA MODEL

We present a data model for semi-structured data (XML) based on Cardelli, Gardner and Ghelli's trees-with-pointers model [4] (see also [2]). Our formalism integrates the simple tree structure of Cardelli and Gordon's ambient calculus [5] with the pointer structure of heaps. In particular, it permits dangling pointers, which are essential for local reasoning about such linked structures.

### Trees with Pointers

Our trees-with-pointers model consists of a labelled tree structure with uniquely identified nodes, unstructured data (values, text) and graphical links (pointers) between nodes. Node identifiers allow us to update trees locally, while pointers allow us to model arbitrary graph structures. We also define *linear contexts*, which are trees with a unique hole[2]. Given infinite sets of names $n \in \mathcal{N}$ and labels $a \in \mathcal{A}$, the sets of values, trees and contexts are defined in Figure 1. The insertion of a tree $T$ in a context $C$, denoted $C(T)$, is defined in the standard way. A *well-formed tree* or *context* is one where the node identifiers are unique. Note that composition, node formation and tree insertion are *partial* when restricted to well-formed trees.

The *structural congruence* $\equiv$ between trees is the smallest congruence on trees which satisfies axioms (i)-(iii) in Figure 1. This corresponds to tree isomorphism under a multiset interpretation. We trivially extend this to a structural congruence on contexts using axioms (iv)-(vi). Note that the result is also a congruence with respect to tree insertion: that is, $C_1(T_1) \equiv C_2(T_2)$ whenever $C_1 \equiv C_2$ and $T_1 \equiv T_2$. The definitions of well-formed trees and contexts are consistent with respect to structural congruence.

## 3. TREE UPDATE LANGUAGE

We present a core update language for directly manipulating trees with pointers. The language is simple, yet expressive enough to illustrate the subtleties of tree update. The atomic commands are similar to the basic operations for XML update studied in [10]. Our core language serves as a starting point for a full update language for XML.

### Variables and Expressions

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 latter is a finite partial function defined on both 'tree variables', which are mapped to trees, and 'value variables', which are mapped to values (see Figure 2). This approach allows us to break down complex operations, such as moving and copying trees, into smaller ones that deal with only one area of the working tree at a time and can hence be analysed locally.

We employ two types of *expressions* in our update language: tree expressions and value expressions. Tree expressions are just tree variables, since these are enough to present our ideas. In fact, our results hold for more complex tree expressions, such as those for describing path-based reasoning. Value expressions consist of value variables or constants. Both forms of expressions are determined by valuations $[\![E]\!]s$ on the store $s$, as shown in Figure 2.

### Commands

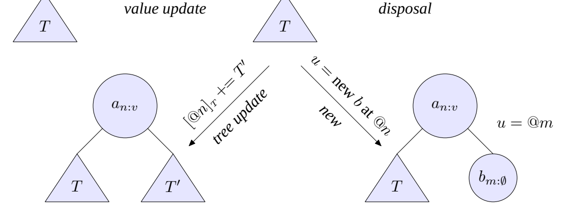
The core system consists of a simple imperative programming language for altering state, with assignment, lookup, update, new and disposal commands:

$$\mathbb{C} ::= \begin{array}{lll} \text{tree operations} & \text{value operations} & \\ x = E_{\mathcal{T}} & u = E_{\mathcal{V}} & \text{assignment} \\ x = [E_{\mathcal{V}}]_T & u = [E_{\mathcal{V}}]_V & \text{lookup} \\ [E_{\mathcal{V}}]_T \mathrel{+}= F_{\mathcal{T}} & [E_{\mathcal{V}}]_V = F_{\mathcal{V}} & \text{update} \\ u = \text{new } a \text{ at } E_{\mathcal{V}} & & \text{new} \\ \text{dispose } E_{\mathcal{V}} & & \text{dispose} \end{array}$$

These commands are split into tree operations and value operations. Assignment, lookup and update have closely related tree and value forms. Assignment assigns to a variable $x$ or $u$ the value of an expression $E_{\mathcal{T}}$ or $E_{\mathcal{V}}$; lookup assigns to $x$ or $u$ the tree or value at the location specified (as a pointer) by the value of $E_{\mathcal{V}}$; and update goes to the location specified by $E_{\mathcal{V}}$ and either *appends* to the subtree there the value of $F_{\mathcal{T}}$, or *replaces* the value there by the value of $F_{\mathcal{V}}$. We chose to append in the case of tree updates as it is a natural operation on trees which, together with appropriate disposal, can express tree replacement.

The new command creates a new tree node, with label $a$, a fresh identifier, the nil value and an empty subtree, appends the node to the subtree at the location specified by $E_{\mathcal{V}}$, and assigns to $u$ a pointer back to the newly created identifier. The dispose command deletes the tree specified by $E_{\mathcal{V}}$.

The complete operational semantics of the commands is given in Figure 3, and is similar to the operational semantics for updating heaps given in [12]. It uses an evaluation relation $\rightsquigarrow$ defined on configuration triples $\mathbb{C}, s, t$, terminal states $s, t$, and faults. Note that the specification of the semantics requires contexts $C$ to isolate the subtrees affected by the commands. The diagram below illustrates the behaviour of some of the commands on a simple tree of the form $a_{n:v}[T]$, with arrows pointing to the results:



[2]We believe it is routine to adapt the results presented in this paper to multi-holed contexts. We limit ourselves to linear contexts, since these are enough to derive our weakest preconditions.

| | | |
|---|---|---|
| values $V$ ::= | nil | null value |
| | $@n$ | a pointer |
| | data | data |
| trees $T$ ::= | $0$ | empty tree |
| | $a_{n:V}[T]$ | node: label $a$, name $n$, value $V$ |
| | $T \mid T$ | composition of trees |
| contexts $C$ ::= | $-$ | empty context |
| | $a_{n:V}[C]$ | node: label $a$, name $n$, value $V$ |
| | $C \mid T$ | right-composition with tree |
| | $T \mid C$ | left-composition with tree |

*Structural congruence*

(i) $T_1 \mid T_2 \equiv T_2 \mid T_1$
(ii) $T \mid 0 \equiv T$
(iii) $T_1 \mid (T_2 \mid T_3) \equiv (T_1 \mid T_2) \mid T_3$

(iv) $C \mid T \equiv T \mid C$
(v) $C \mid 0 \equiv C$
(vi) $C \mid (T_1 \mid T_2) \equiv (C \mid T_1) \mid T_2$

**Figure 1: Data Model**

| | | | |
|---|---|---|---|
| tree variables $\mathrm{Var}_\mathcal{T} =$ | $\{x, y, \dots\}$ | tree expressions $E_\mathcal{T}$ ::= | $\mathrm{Var}_\mathcal{T}$ |
| value variables $\mathrm{Var}_\mathcal{V} =$ | $\{u, v, \dots\}$ | value expressions $E_\mathcal{V}$ ::= | $\mathrm{Var}_\mathcal{V} \mid V$ |

stores $s \in (\mathrm{Var}_\mathcal{T} \rightharpoonup_{\mathrm{fin}} T) \times (\mathrm{Var}_\mathcal{V} \rightharpoonup_{\mathrm{fin}} V)$ valuations $[\![E]\!]s : [\![x]\!]s = s(x)$ $[\![u]\!]s = s(u)$, $[\![V]\!]s = V$

**Figure 2: Variables and Expressions**

The lookup, update and dispose commands rely on the expression $E_\mathcal{V}$ evaluating to a pointer which identifies a node in the working tree[3]. If it does not, they will fault. A different error occurs when the tree update operation tries to insert a tree with a node identifier that clashes with one already in 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 (see Section 5). 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 update operation is to rename the node identifiers of the tree being inserted with fresh identifiers. Our simpler operation is enough for this paper.

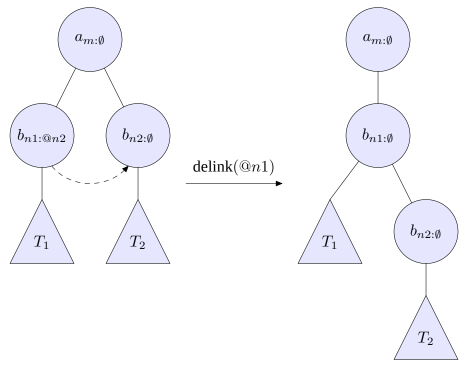
### Example Program

We present a simple program *delink* that takes a pointer from a given location, and collapses it by moving its target to that location:

$$\begin{aligned}\mathrm{delink}(E_\mathcal{V}) \triangleq\ & u = [E_\mathcal{V}]_V;\\ & x = [u]_T;\\ & \text{dispose } u;\\ & [E_\mathcal{V}]_T \mathrel{+}= x;\\ & [E_\mathcal{V}]_V = \mathrm{nil}\end{aligned}$$

[3]Well-formedness of the working tree ensures that the pointer target is always unique.

The diagram below illustrates the behaviour of the program on the sample tree given in the introduction:



Note that the program would also yield a result if the pointer referred to a subtree inside $T_1$, but would clearly fault if the pointer referred back to the top node. In Section 5, we show how our program reasoning copes quite naturally with these different cases.

## 4. CONTEXT LOGIC

In [4], Gardner *et al.* amalgamated ideas from the Ambient Logic and Separation Logic to provide a logic for analysing the trees-with-pointers model. In Section 5, we show that this logic is not expressive enough to describe the weakest preconditions for our update language. This observation led to Context Logic. Beginning with a concrete application, we first introduce the Context Logic specialised to analyse our trees. We then study the general theory of Context Logic, providing the proof theory, models and completeness results. We also show how to collapse its structure to obtain the Bunched Logic of O'Hearn and Pym as a special instance. In Section 5, we show that the Context Logic for the trees-with-pointers model is indeed expressive enough to describe weakest preconditions.

$$\frac{[\![E_{\mathcal{T}}]\!]s \equiv T'}{x = E_{\mathcal{T}}, s, T \rightsquigarrow [s|x \leftarrow T'], T} \qquad \frac{[\![E_{\mathcal{V}}]\!]s = V}{u = E_{\mathcal{V}}, s, T \rightsquigarrow [s|u \leftarrow V], t}$$

$$\frac{[\![E_{\mathcal{V}}]\!]s = @n \quad T \equiv C(a_{n:V}[T'])}{x = [E_{\mathcal{V}}]_T, s, T \rightsquigarrow [s|x \leftarrow a_{n:V}[T']], T} \qquad \frac{[\![E_{\mathcal{V}}]\!]s = @n \quad T \equiv C(a_{n:V}[T'])}{u = [E_{\mathcal{V}}]_V, s, T \rightsquigarrow [s|u \leftarrow V], T}$$

$$\frac{[\![E_{\mathcal{V}}]\!]s = @n \quad T \equiv C(a_{n:V}[T']) \quad [\![F_{\mathcal{T}}]\!]s \equiv T'' \quad C(a_{n:V}[T' \,|\, T'']) \text{ well-formed}}{[E_{\mathcal{V}}]_T \mathrel{+}= F_{\mathcal{T}}, s, T \rightsquigarrow s, C(a_{n:V}[T' \,|\, T''])}$$

$$\frac{[\![E_{\mathcal{V}}]\!]s = @n \quad T \equiv C(a_{n:V}[T']) \quad [\![F_{\mathcal{V}}]\!]s = V'}{[E_{\mathcal{V}}]_V = F_{\mathcal{V}}, s, T \rightsquigarrow s, C(a_{n:V'}[T'])} \qquad \frac{[\![E_{\mathcal{V}}]\!]s = @m \quad T \equiv C(b_{m:V}[T']) \quad n \notin \text{Free Names}(T)}{u = \text{new } a \text{ at } E_{\mathcal{V}}, s, T \rightsquigarrow [s|u \leftarrow @n], C(b_{m:V}[T' \,|\, a_{n:\text{nil}}[0]])}$$

$$\frac{[\![E_{\mathcal{V}}]\!]s = @n \quad T \equiv C(a_{n:V}[T'])}{\text{dispose } E_{\mathcal{V}}, s, T \rightsquigarrow s, C(0)} \qquad \frac{[\![E_{\mathcal{V}}]\!]s \neq @n \vee T \not\equiv C(a_{n:V}[T']))}{\left.\begin{array}{cc} x = [E_{\mathcal{V}}]_T & u = [E_{\mathcal{V}}]_V \\ [E_{\mathcal{V}}]_T \mathrel{+}= F_{\mathcal{T}} & [E_{\mathcal{V}}]_V = F_{\mathcal{V}} \\ u = \text{new } a \text{ at } E_{\mathcal{V}} & \text{dispose } E_{\mathcal{V}} \end{array}\right\}, s, T \rightsquigarrow \text{fault}}$$

$$\frac{\mathbb{C}_1, s, T \rightsquigarrow \mathbb{C}_1', s', T'}{(\mathbb{C}_1; \mathbb{C}_2), s, T \rightsquigarrow (\mathbb{C}_1'; \mathbb{C}_2), s', T'} \qquad \frac{\mathbb{C}_1, s, T \rightsquigarrow s', T'}{(\mathbb{C}_1; \mathbb{C}_2), s, T \rightsquigarrow \mathbb{C}_2, s', T'} \qquad \frac{\mathbb{C}_1, s, T \rightsquigarrow \text{fault}}{(\mathbb{C}_1; \mathbb{C}_2), s, T \rightsquigarrow \text{fault}}$$

$[s|x \leftarrow v]$ means the partial function $s$ overwritten with $s(x) = v$

**Figure 3: Operational Semantics**

## 4.1 A Context Logic for Trees

The Context Logic for analysing trees with pointers consists of tree assertions denoted by $P$, and context assertions denoted by $K$. In each case, these include standard assertions from classical first-order logic, novel structural assertions for analysing the tree and context structure, and specialised assertions associated with our tree model: basic tree and context assertions, and quantification over the variables of our update language:

| | | |
|---|---|---|
| $P ::=$ | $K(P) \mid K \lhd P$ | structural assertions |
| | $0 \mid E_{\mathcal{T}} \mid E_{\mathcal{V}} = E_{\mathcal{V}}$ | basic assertions |
| | $P \Rightarrow P \mid \text{false}$ | classical assertions |
| | $\exists x.P \mid \exists u.P \mid \exists a.P \mid \exists n.P$ | quantifiers |
| $K ::=$ | $P \rhd P$ | structural assertions |
| | $- \mid a_{n:E_{\mathcal{V}}}[K] \mid P \,|\, K$ | basic assertions |
| | $K \Rightarrow K \mid \text{False}$ | classical assertions |
| | $\exists x.K \mid \exists u.K \mid \exists a.K \mid \exists n.K$ | quantifiers |

The full semantics is given in Figure 4 by two satisfaction relations: the judgement $s, T \vDash_{\mathcal{T}} P$ says that the tree assertion $P$ holds for a given store and tree, while judgement $s, C \vDash_{\mathcal{K}} K$ states that the context assertion $K$ holds for a given store and context.

The key assertions are the structural assertions $K(P)$, $K \lhd P$ and $P \rhd Q$. The *application assertion* $K(P)$ specifies that a tree can be split into a context satisfying $K$ applied to a subtree satisfying $P$. The other two assertions are both (right) adjoints of application. The assertion $K \lhd P$ is adjoint to $K(-)$. It is satisfied by a given tree if, whenever we insert the tree into a context satisfying $K$, then the result satisfies $P$. We shall see that this adjoint is analogous to the magic wand of Separation Logic, and the composition and branch adjoints of the Ambient Logic. Meanwhile $P \rhd Q$ is adjoint to $-(P)$ and is therefore a statement on *contexts*. It is satisfied by a given context if, whenever we insert in the context a tree satisfying $P$, then the result satisfies $Q$. Note that this assertion has no analogue in a naïve adaptation of the Ambient Logic, and provides much of the power of our context approach. It does however correspond to the magic wand of Separation Logic, as the context structure collapses in the heap case. It is used throughout the paper and is essential for expressing weakest preconditions for update commands.

The basic tree assertions are specific to our tree application. They consist of the empty tree 0, the assertion $E_{\mathcal{T}}$ which holds whenever the working tree has the same value as $E_{\mathcal{T}}$ (up to structural congruence), and the assertion $E_{\mathcal{V}} = F_{\mathcal{V}}$ which holds when the values of the expressions are equal. The basic context assertions consist of the empty context '$-$' and two basic context connectives: the *subtree context* $a_{n:E_{\mathcal{V}}}[K]$, satisfied by the context $a_{n:[\![E_{\mathcal{V}}]\!]s}[C]$ for some $C$ satisfying $K$, and the *parallel context* $P \,|\, K$, satisfied by a tree satisfying $P$ in parallel with a context satisfying $K$.

For ease of reading, we often write $P \,|\, Q$ and $a_{n:v}[P]$ instead of $(P \,|\, -)(Q)$ and $(a_{n:v}[-])(P)$. This causes no ambiguity.

### *Auxiliary Definitions*

We use the standard derived classical connectives and quantifiers for both trees and contexts: $\neg P$, true, $P \wedge P$, $P \vee P$ and $\forall \bullet \,.\, P$, where $\bullet$ is $x$, $u$, $a$ or $n$. (For contexts we write True rather than true.) We also use the derived tree equality $E_{\mathcal{T}} = F_{\mathcal{T}}$, which is satisfied whenever the two tree expressions have the same value:

$$E_{\mathcal{T}} = F_{\mathcal{T}} \triangleq \text{true} \,|\, ((E_{\mathcal{T}} \rhd F_{\mathcal{T}})(0) \wedge 0)$$

$$\begin{array}{ll}
s,T \vDash_{\mathcal{T}} K(P) \text{ iff } \exists\, C,T' \text{ s.t. } T \equiv C(T'), & s,C \vDash_{\mathcal{K}} P \triangleright Q \text{ iff } \forall\, T_1(s,T_1 \vDash_{\mathcal{T}} P \text{ and} \\
\qquad s,C \vDash_{\mathcal{K}} K \text{ and } s,T' \vDash_{\mathcal{T}} P & \qquad C(T_1) \text{ well-formed } \Rightarrow s,C(T_1) \vDash_{\mathcal{T}} Q) \\
s,T \vDash_{\mathcal{T}} K \lhd P \text{ iff } \forall\, C_1(s,C_1 \vDash_{\mathcal{K}} K \text{ and} & s,C \vDash_{\mathcal{K}} - \text{ iff } C \equiv - \\
\qquad C_1(T) \text{ well-formed } \Rightarrow s,C_1(T) \vDash_{\mathcal{T}} P) & s,C \vDash_{\mathcal{K}} a_{n:E_{\mathcal{V}}}[K] \text{ iff } \exists\, C' \text{ s.t. } C \equiv a_{n:[\![E_{\mathcal{V}}]\!]s}[C'] \\
s,T \vDash_{\mathcal{T}} 0 \text{ iff } T \equiv 0 & \qquad \text{and } s,C' \vDash_{\mathcal{K}} K \\
s,T \vDash_{\mathcal{T}} E_{\mathcal{T}} \text{ iff } T \equiv [\![E_{\mathcal{T}}]\!]s & s,C \vDash_{\mathcal{K}} P \mid K \text{ iff } \exists\, T,C' \text{ s.t. } C \equiv T \mid C', \\
s,T \vDash_{\mathcal{T}} E_{\mathcal{V}} = F_{\mathcal{V}} \text{ iff } [\![E_{\mathcal{V}}]\!]s = [\![F_{\mathcal{V}}]\!]s & \qquad s,T \vDash_{\mathcal{T}} P \text{ and } s,C' \vDash_{\mathcal{K}} K \\
\\
s,T \vDash_{\mathcal{T}} P \Rightarrow Q \text{ iff } s,T \vDash_{\mathcal{T}} P \text{ implies } s,T \vDash_{\mathcal{T}} Q & s,C \vDash_{\mathcal{K}} K_1 \Rightarrow K_2 \text{ iff } s,C \vDash_{\mathcal{K}} K_1 \text{ implies } s,C \vDash_{\mathcal{K}} K_2 \\
s,T \vDash_{\mathcal{T}} \text{false} \quad \text{never} & s,C \vDash_{\mathcal{K}} \text{False} \quad \text{never} \\
s,T \vDash_{\mathcal{T}} \exists x.P \text{ iff } \exists\, T' \text{ s.t. } [s|x \mapsto T'],T \vDash_{\mathcal{T}} P & s,C \vDash_{\mathcal{K}} \exists x.K \text{ iff } \exists\, T \text{ s.t. } [s|x \mapsto T],C \vDash_{\mathcal{K}} K \\
s,T \vDash_{\mathcal{T}} \exists u.P \text{ iff } \exists\, V \text{ s.t. } [s|u \mapsto V],T \vDash_{\mathcal{T}} P & s,C \vDash_{\mathcal{K}} \exists u.K \text{ iff } \exists\, V \text{ s.t. } [s|u \mapsto V],C \vDash_{\mathcal{K}} K \\
s,T \vDash_{\mathcal{T}} \exists a.P \text{ iff } \exists\, a' \in \mathcal{A} \text{ s.t. } s,T \vDash_{\mathcal{T}} P[a'/a] & s,C \vDash_{\mathcal{K}} \exists a.K \text{ iff } \exists\, a' \in \mathcal{A} \text{ s.t. } s,C \vDash_{\mathcal{K}} K[a'/a] \\
s,T \vDash_{\mathcal{T}} \exists n.P \text{ iff } \exists\, n' \in \mathcal{N} \text{ s.t. } s,T \vDash_{\mathcal{T}} P[n'/n] & s,C \vDash_{\mathcal{K}} \exists n.K \text{ iff } \exists\, n' \in \mathcal{N} \text{ s.t. } s,C \vDash_{\mathcal{K}} K[n'/n]
\end{array}$$

**Figure 4: Semantics of Formulas**

A useful property of equality assertions for values and trees is their context-insensitivity:

$$\begin{array}{lcl}
K((E_{\mathcal{T}} = F_{\mathcal{T}}) \wedge Q) & \Leftrightarrow & (E_{\mathcal{T}} = F_{\mathcal{T}}) \wedge K(Q) \\
K((E_{\mathcal{V}} = F_{\mathcal{V}}) \wedge Q) & \Leftrightarrow & (E_{\mathcal{V}} = F_{\mathcal{V}}) \wedge K(Q)
\end{array}$$

We say that an assertion is *exact* if it is satisfied by at most one tree or context for any given store. The following useful property is reversible whenever $P$ is exact.

$$(P \triangleright Q)(P) \underset{\text{exact P}}{\overset{\text{always}}{\rightleftarrows}} Q \wedge \text{True}(P)$$

## 4.2 General Theory of Context Logic

In this section, we develop the underlying general theory of Context Logic, giving the proof theory, the models and the forcing semantics. We give soundness and completeness results. Finally, we compare our Context Logic with the Bunched Logic of O'Hearn and Pym.

For the purpose of this paper, we deal only with boolean Context Logics, where the law of excluded middle holds.

Definition 1 (Context Logic Assertions). *Context Logic consists of a set of* data assertions *and a set of* context assertions, *described by the grammars:*

*data assertions*

$$\begin{array}{lll}
P ::= & K(P) \mid K \lhd P & \textit{structural assertions} \\
& P \Rightarrow P \mid \textit{false} & \textit{additive assertions}
\end{array}$$

*context assertions*

$$\begin{array}{lll}
K ::= & I \mid P \triangleright P & \textit{structural assertions} \\
& K \Rightarrow K \mid \textit{False} & \textit{additive assertions}
\end{array}$$

Definition 2 (Proof Theory). *The Hilbert-style proof theory for Context Logic consists of the standard axioms and rules for the additive assertions plus those for the structural assertions given in Figure 5.*

Definition 3 (Model). *A* model $\mathbf{M}$ *for Context Logic is given by two sets,* $\mathbf{K}$ *and* $\mathbf{D}$*, with a partial application function* $app : \mathbf{K} \times \mathbf{D} \rightharpoonup \mathbf{D}$ *and a nonempty set* $I \subseteq \mathbf{K}$ *that acts as a left identity:*

$$\forall d \in \mathbf{D}.app(I, \{d\}) = \{d\}$$

*where app is extended to sets in the obvious way.*

The forcing semantics for the structural assertions with respect to a model $\mathbf{M}$ is also given in Figure 5, using two satisfaction relations $\mathbf{M}, c \vDash_{\mathcal{K}} K$ and $\mathbf{M}, d \vDash_{\mathcal{D}} P$. The semantics for both sets of additive connectives is standard.

Theorem 4 (Soundness and Completeness). *The proof theory is sound and complete with respect to the forcing semantics: that is,*

$$\begin{array}{l}
K \vdash_{\mathcal{K}} K' \Leftrightarrow (\mathbf{M}, K \vDash_{\mathcal{K}} K' \textit{ for all models } \mathbf{M}) \\
P \vdash_{\mathcal{D}} P' \Leftrightarrow (\mathbf{M}, P \vDash_{\mathcal{D}} P' \textit{ for all models } \mathbf{M})
\end{array}$$

*where* $\mathbf{M}, K \vDash_{\mathcal{K}} K' \Leftrightarrow \forall c \in \mathbf{K}. \mathbf{M}, c \vDash_{\mathcal{K}} K \Rightarrow \mathbf{M}, c \vDash_{\mathcal{K}} K'$ *and* $\mathbf{M}, P \vDash_{\mathcal{D}} P' \Leftrightarrow \forall d \in \mathbf{D}. \mathbf{M}, d \vDash_{\mathcal{D}} P \Rightarrow \mathbf{M}, d \vDash_{\mathcal{D}} P'$ *when* $\mathbf{M} = (\mathbf{K}, \mathbf{D}, app, I)$.

Proof. Soundness follows by easy induction on the derivation of $K \vdash_{\mathcal{K}} K'$ and $P \vdash_{\mathcal{D}} P'$ respectively. Completeness requires techniques based on maximally consistent sets of formulas and bisimulation, recently developed in [11]. □

### *Context Logic with Zero*

A natural assertion to add is the zero assertion 0, corresponding to the empty heap assertion in Separation Logic and the empty tree assertion in the Ambient Logic. We shall however see that this extra assertion is not present in the term rewriting case. When it is present, the zero assertion allows us to derive extra structure: specifically, $K(0)$ is a *projection* from contexts onto data, and $0 \triangleright P$ is an (adjoint) *embedding* in the other direction.

Definition 5 (Context Logic Assertions with Zero). *The* Context Logic with Zero *consists of data and context assertions as in Definition 1, with an additional data assertion* 0.

Definition 6 (Proof theory with Zero). *The proof theory for Context Logic with Zero extends Figure 5 with the following axioms:*

$$(0 \triangleright P)(0) \dashv\vdash_{\mathcal{D}} P \qquad \neg(0 \triangleright P) \dashv\vdash_{\mathcal{K}} 0 \triangleright \neg P$$

$$I \dashv\vdash_{\mathcal{K}} 0 \triangleright 0$$

It is simple to check that these axioms hold in our concrete tree model. We give further explanation after we introduce the models.

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

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

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

$$\mathbf{M} = (\mathbf{K}, \mathbf{D}, app, I)$$

$$\begin{aligned}
&\mathbf{M}, d \vDash_{\mathcal{D}} K(P) \text{ iff } \exists c \in \mathbf{K}, d' \in \mathbf{D}. \text{ s.t. } app(c, d') = d,\\
&\qquad \mathbf{M}, c \vDash_{\mathcal{K}} K \text{ and } \mathbf{M}, d' \vDash_{\mathcal{D}} P\\
&\mathbf{M}, d \vDash_{\mathcal{D}} K \lhd P \text{ iff } \forall c \in \mathbf{K}.(\mathbf{M}, c \vDash_{\mathcal{K}} K \text{ and}\\
&\qquad app(c, d) \text{ defined } \Rightarrow \mathbf{M}, app(c, d) \vDash_{\mathcal{D}} P)\\
&\mathbf{M}, c \vDash_{\mathcal{K}} I \text{ iff } c \in I\\
&\mathbf{M}, c \vDash_{\mathcal{K}} P_1 \rhd P_2 \text{ iff } \forall d \in \mathbf{D}.(\mathbf{M}, d \vDash_{\mathcal{D}} P_1 \text{ and}\\
&\qquad app(c, d) \text{ defined } \Rightarrow \mathbf{M}, app(c, d) \vDash_{\mathcal{D}} P_2)
\end{aligned}$$

**Figure 5: Context Logic Proof Theory and Forcing Semantics**

DEFINITION 7 (MODEL WITH ZERO). *A* model $\mathbf{M}$ *for Context Logic with Zero is a model* $(\mathbf{K}, \mathbf{D}, app, I)$ *for Context Logic with an additional set* $\mathbf{0} \subseteq \mathbf{D}$*, and a relation* $R_{app} \subseteq \mathbf{K} \times \mathbf{D}$ *defined by*

$$(c, d) \in R_{app} \Leftrightarrow d \in app(c, \mathbf{0})$$

*which satisfies the following conditions:*

*(i)* $R_{app}$ *is a total surjective function* $\mathbf{K} \to \mathbf{D}$*;*

*(ii)* ${R_{app}}^{-1}(\mathbf{0}) = I$.

The first Axiom in Definition 6 implies that $R_{app}$ is surjective, and the second implies that it is a total function. The third axiom corresponds to condition (ii).

Notice that the action of $R_{app}$ on sets of contexts is representable as an operator $-^{p} : K \mapsto K(0)$. The inverse relation is also representable, as the operator $-^{e} : P \mapsto (0 \rhd P)$. Using this derived notation, we can give a new reading to the axioms in the proof theory. The first axiom becomes $P^{ep} \dashv\vdash_{\mathcal{D}} P$, and implies that $(-^{e}, -^{p})$ is an embedding-projection pair. The second axiom becomes $\neg(P^{e}) \dashv\vdash_{\mathcal{K}} (\neg P)^{e}$, and says that the embedding preserves and reflects negations, or equivalently that $-^{e}$ has a right adjoint $\neg((\neg -)^{p})$. The third axiom becomes $I \dashv\vdash_{\mathcal{K}} 0^{e}$, and says that 0 embeds to $I$.

DEFINITION 8 (FORCING SEMANTICS WITH ZERO). *The forcing semantics for Context Logic with Zero extends Figure 5 with*

$$\mathbf{M}, d \vDash_{\mathcal{D}} 0 \quad \textit{iff} \quad d \in \mathbf{0}$$

THEOREM 9 (SOUNDNESS AND COMPLETENESS WITH ZERO). *The proof theory for Context Logic with Zero is sound and complete with respect to the forcing semantics with Zero.*

Using the derived embedding/projection notation, there is a natural way to define a structural operation $- * -$ on data: an embedding followed by an application. The definition of $*$ and its two right adjoints (fixing either argument) is as follows:

$$\begin{aligned}
P * Q &\triangleq P^{e}(Q)\\
P \ast\!\!- \, Q &\triangleq P^{e} \lhd Q\\
P \mathrel{-\!\!*} Q &\triangleq \neg((\neg(P \rhd Q))^{p})
\end{aligned}$$

LEMMA 10 (ADJOINTS). $P \ast\!\!- -$ *and* $P \mathrel{-\!\!*} -$ *are the right adjoints of* $P * -$ *and* $- * P$*, respectively.*

PROOF. The former is immediate since $K \lhd -$ is right adjoint of $K(-)$. The latter follows from the adjunction properties of $\rhd$ and $-^{e}$, together with excluded middle and the second axiom in Definition 6. □

LEMMA 11 (UNIT). $P * 0 \Leftrightarrow P \Leftrightarrow 0 * P$

PROOF. The left logical equivalence follows immediately from the first axiom in Definition 6. The right one follows from the third axiom and $I$ being the left identity of application. □

However, $*$ does not have a monoid structure in general. This is exemplified in the tree model: $P * Q$ holds of a tree if it can be decomposed into a context whose underlying tree satisfies $P$, and a tree satisfying $Q$. Therefore $*$ is neither commutative nor associative. Moreover, the interpretation of $*$ in the tree model is a relation $R \subseteq (\mathbf{D} \times \mathbf{D}) \times \mathbf{D}$, rather than a partial function.

### Comparison with BI

The Bunched Logic of O'Hearn and Pym [8] is a special case of Context Logic, where the two sets of assertions are identical, application corresponds to $- * -$, the two adjoints are the same due to the commutativity of $*$, and I corresponds to 0. We formalise the correspondence. When the model $\mathbf{M}$ comes from a BI model[4], then our derived $*$-connective coincides with BI's $*$-connective. Moreover, in these BI models, the Context Logic is exactly as expressive as BI.

THEOREM 12 (COLLAPSE TO BI). *For each data formula* $P$*, there exists an equivalent BI formula*[5] $\phi$*: that is,* $\mathbf{M}, true \vDash_{\mathcal{D}} (P \Leftrightarrow \phi)$ *whenever* $\mathbf{M}$ *comes from a BI model.*

In this section, we have concentrated on a specific presentation of boolean Context Logic. We are currently working on the categorical semantics and completeness for several variants of the logic, where context composition and the embedding/projection pair described here play a prominent role. This work will appear in a future paper.

## 5. PROGRAM LOGIC FOR TREE UPDATE

We present a Hoare logic for reasoning about our tree update language, based on the fault-avoiding interpretation and small axioms of O'Hearn, Reynolds and Yang [7]. We express and derive the weakest preconditions for our commands.

DEFINITION 13 (HOARE TRIPLES). *A triple* $\{P\}\,\mathbb{C}\,\{Q\}$ *holds iff whenever* $\mathbb{C}$ *is run in a state* $s, T$ *satisfying* $P$*: (i) it does not generate a fault (we say* $\mathbb{C}, s, T$ *is* safe*), and (ii) if it terminates then the resulting state satisfies* $Q$*.*

[4]That is, both $\mathbf{K}$ and $\mathbf{D}$ being the underlying set of the BI monoid, with *app* and $I$ the binary operation and unit.

[5]A formula in the fragment $\phi ::= 0 \mid \phi * \phi \mid \phi \mathrel{-\!\!*} \phi \mid \phi \Rightarrow \phi \mid \text{false}$.

$$\begin{array}{rll}
\{(x = x_0) \wedge 0\} & x = E_{\mathcal{T}} & \{(x = E_{\mathcal{T}}[x_0/x]) \wedge 0\} \\
\{(u = u_0) \wedge 0\} & u = E_{\mathcal{V}} & \{(u = E_{\mathcal{V}}[u_0/u]) \wedge 0\} \\
\{(E_{\mathcal{V}} = @n) \wedge (x = x_0) \wedge (a_{n:v}[\mathrm{true}] \wedge y)\} & x = [E_{\mathcal{V}}]_T & \{(E_{\mathcal{V}}[x_0/x] = @n) \wedge (x = y) \wedge (a_{n:v}[\mathrm{true}] \wedge y)\} \\
\{(E_{\mathcal{V}} = @n) \wedge (u = u_0) \wedge a_{n:v}[y]\} & u = [E_{\mathcal{V}}]_V & \{(E_{\mathcal{V}}[u_0/u] = @n) \wedge (u = v) \wedge a_{n:v}[y]\} \\
\{(E_{\mathcal{V}} = @n) \wedge a_{n:v}[y]\} & [E_{\mathcal{V}}]_T \mathrel{+}= F_{\mathcal{T}} & \{(E_{\mathcal{V}} = @n) \wedge a_{n:v}[y \mid F_{\mathcal{T}}]\} \\
\{(E_{\mathcal{V}} = @n) \wedge a_{n:v}[y]\} & [E_{\mathcal{V}}]_V = F_{\mathcal{V}} & \{(E_{\mathcal{V}} = @n) \wedge a_{n:F_{\mathcal{V}}}[y]\} \\
\{(E_{\mathcal{V}} = @m) \wedge (u = u_0) \wedge b_{m:v}[y]\} & u = \text{new } a \text{ at } E_{\mathcal{V}} & \{(E_{\mathcal{V}}[u_0/u] = @m) \wedge \exists n.((u = @n) \wedge b_{m:v}[y \mid a_{n:\mathrm{nil}}[0]])\} \\
\{(E_{\mathcal{V}} = @n) \wedge a_{n:v}[\mathrm{true}]\} & \text{dispose } E_{\mathcal{V}} & \{(E_{\mathcal{V}} = @n) \wedge 0\}
\end{array}$$

The variables $x, x_0, y$ and $u, u_0, v$ are assumed to be distinct.

**Figure 6: Small Axioms**

$$\begin{array}{rl}
\textbf{Frame Rule:} & \dfrac{\{P\}\ \mathbb{C}\ \{Q\}}{\{K(P)\}\ \mathbb{C}\ \{K(Q)\}} \quad \mathrm{Mod}(\mathbb{C}) \cap \mathrm{FV}(K) = \{\} \\[2ex]
\textbf{Auxiliary Variable Elimination:} & \dfrac{\{P\}\ \mathbb{C}\ \{Q\}}{\{\exists \bullet .P\}\ \mathbb{C}\ \{\exists \bullet .Q\}} \quad \bullet = x|u|a|n \notin \mathrm{FV}(\mathbb{C}) \\[2ex]
\textbf{Consequence:} & \dfrac{P' \Rightarrow P \quad \{P\}\ \mathbb{C}\ \{Q\} \quad Q \Rightarrow Q'}{\{P'\}\ \mathbb{C}\ \{Q'\}} \\[2ex]
\textbf{Sequencing:} & \dfrac{\{P\}\ \mathbb{C}_1\ \{Q\} \quad \{Q\}\ \mathbb{C}_2\ \{R\}}{\{P\}\ \mathbb{C}_1; \mathbb{C}_2\ \{R\}}
\end{array}$$

**Figure 7: Inference Rules**

## *Small Axioms*

We give a set of "small axioms", one for each of the commands that alter state. The axioms are small, in that they mention only those areas of data that are accessed by the corresponding command. They are presented in Figure 6 and are self-explanatory.

THEOREM 14. *The Small Axioms are sound.*

PROOF. The result follows directly from the operational semantics of the commands. □

## *Inference Rules*

There are four inference rules, given in Figure 7. The key rule is the Frame Rule, which is a direct generalisation of the Frame Rule in [12]. It captures the idea of local behaviour, by stating that if a working tree is sufficient for the faultless execution of a command, then any additional tree structure will remain unaltered by that command. We use context assertions to separate out this extra structure.

The set $\mathrm{Mod}(\mathbb{C})$ in the side-condition contains those store variables modified by a command $\mathbb{C}$: i.e. $\{x\}$ for tree assignments and lookups, $\{u\}$ for value assignments, lookups and new, $\{\}$ for update and disposal, and the usual union for sequences. We define $\mathrm{FV}(P)$, $\mathrm{FV}(K)$ and $\mathrm{FV}(\mathbb{C})$ to be the set of free variables in $P$, $K$ and $\mathbb{C}$, with variables bound only by existential quantifiers.

The Auxiliary Variable Elimination, Consequence and Sequencing rules are standard.

The soundness of the inference rules is routine, except for the Frame Rule which depends on the commands being local.

DEFINITION 15 (LOCALITY). *A command $\mathbb{C}$ is* local *if it satisfies these properties:*

*Safety Monotonicity: if $\mathbb{C}, s, T$ is safe (i.e. executes without fault) and $C(T)$ is well-formed then $\mathbb{C}, s, C(T)$ safe.*

*Frame Property: if $\mathbb{C}, s, T$ is safe, $C(T)$ is well-formed, and $\mathbb{C}, s, C(T) \leadsto^* s', T'$ then $\exists T''$ such that $\mathbb{C}, s, T \leadsto^* s', T''$ and $T' \equiv C(T'')$.*

These conditions characterise which commands act, and can therefore be reasoned about, locally. They are generalisations of the conditions given in [12]. For example, a tree update command resulting in a non-wellformed tree must diverge (as mentioned in Section 3). This divergence is necessary since the well-formedness check is a global one. Using fault instead of divergence would break safety monotonicity and make local reasoning impossible. This situation is analogous to reasoning about a heap command trying to allocate a fixed location, say 7. This command would diverge when 7 is already allocated.

THEOREM 16. *The Inference Rules given in Figure 7 are sound.*

PROOF. The only interesting case is the Frame Rule. All the commands are local (i.e. satisfy Safety Monotonicity and the Frame Property). Soundness then follows from a direct argument as in [12]. □

## *Weakest Preconditions*

The weakest precondition with respect to a predicate $P$ and a command $\mathbb{C}$ is a predicate characterizing all pre-states from which every terminating execution of $\mathbb{C}$ leads to a state satisfying $P$.

Being able to express the weakest preconditions in the logic and derive them from the small axioms is an important sanity check and immediately provides a completeness

$$
\begin{array}{rll}
\{P[E_{\mathcal{T}}/x]\} & x = E_{\mathcal{T}} & \{P\} \\
\{P[E_{\mathcal{V}}/u]\} & u = E_{\mathcal{V}} & \{P\} \\
\{\exists y, v, a, n.\ (E_{\mathcal{V}} = @n) \wedge \mathrm{True}(a_{n:v}[\mathrm{true}] \wedge y) \wedge P[y/x]\} & x = [E_{\mathcal{V}}]_T & \{P\} \\
\{\exists v, a, n.\ (E_{\mathcal{V}} = @n) \wedge \mathrm{True}(a_{n:v}[\mathrm{true}]) \wedge P[v/u]\} & u = [E_{\mathcal{V}}]_V & \{P\} \\
\{\exists y, v, a, n.\ (E_{\mathcal{V}} = @n) \wedge (a_{n:v}[y \mid F_{\mathcal{T}}] \rhd P)(a_{n:v}[y])\} & [E_{\mathcal{V}}]_T \mathrel{+}= F_{\mathcal{T}} & \{P\} \\
\{\exists y, v, a, n.\ (E_{\mathcal{V}} = @n) \wedge (a_{n:F_{\mathcal{V}}}[y] \rhd P)(a_{n:v}[y])\} & [E_{\mathcal{V}}]_V = F_{\mathcal{V}} & \{P\} \\
\{\exists y, v, b, m.(E_{\mathcal{V}} = @m) \wedge (\forall n.\ (b_{m:v}[y \mid a_{n:\mathrm{nil}}[0]] \rhd P[@n/u]))(b_{m:v}[y])\} & u = \mathrm{new}\ a\ \mathrm{at}\ E_{\mathcal{V}} & \{P\} \\
\{\exists v, a, n.\ (E_{\mathcal{V}} = @n) \wedge ((0 \rhd P)(a_{n:v}[\mathrm{true}]))\} & \mathrm{dispose}\ E_{\mathcal{V}} & \{P\}
\end{array}
$$

The variables $y, v, a, b, m, n$ (except $a$ for new) are assumed not to occur free in the command or the postcondition.

**Figure 8: Weakest Preconditions**

result for straightline code: all true triples are derivable. Furthermore, a standard design of a verification tool is to use a verification generator on code annotated with pre- and postconditions to derive the weakest precondition of a given postcondition, and then verify, using a theorem prover (or automatic decision procedure when possible), that the corresponding given precondition implies the weakest one.

Figure 8 contains the weakest preconditions expressed as Hoare triples. These triples are analogous to those given in [12] for heap update, as we show in Section 6. As an example, consider the weakest precondition for the tree update command $[E_{\mathcal{V}}]_T \mathrel{+}= F_{\mathcal{T}}$:

$$\exists y, v, a, n.\ (E_{\mathcal{V}} = @n) \wedge (a_{n:v}[y \mid F_{\mathcal{T}}] \rhd P)(a_{n:v}[y])$$

This property is satisfied if $E_V$ evaluates to a pointer $@n$, and the tree can be split into a context $C$ and a subtree $a_{n:v}[y]$ for some fresh tree variable $y$, such that when $a_{n:v}[y \mid F_{\mathcal{T}}]$ is inserted into the context and the result is well-formed, then the result satisfies $P$. In another words, the result of appending $F_{\mathcal{T}}$ under the node at $n$ satisfies $P$, as we would expect.

Lemma 17. *The axioms in Figure 8 express the weakest preconditions.*

Proof. The result follows directly from the operational semantics of the commands. □

Theorem 18 (Derivation of WP's). *The weakest preconditions are derivable.*

Proof. See extended version of the paper online. □

## *Program Logic Example*

Using the example program from Section 3, we can demonstrate Hoare reasoning using Context Logic. By calculating the weakest precondition of program with respect to the postcondition {true}, we can derive the necessary condition for non-faulting execution. We show this for delink($@n$), simplifying the expressions as we go:

$$
\begin{array}{l}
\left\{ \begin{array}{l} \exists a, b, m, v_2. \\ (0 \rhd \mathrm{True}(a_{n:@m}[\mathrm{true}]))(b_{m:v_2}[\mathrm{true}]) \end{array} \right\} \\
\quad u = [v]_V; \\
\left\{ \begin{array}{l} \exists a, b, m, v_1, v_2.\ (u = @m) \wedge \\ (0 \rhd \mathrm{True}(a_{n:v_1}[\mathrm{true}]))(b_{m:v_2}[\mathrm{true}]) \end{array} \right\} \\
\quad x = [u]_T \\
\left\{ \begin{array}{l} \exists a, b, m, v_1, v_2.\ (u = @m) \wedge \\ (0 \rhd \mathrm{True}(a_{n:v_1}[\mathrm{true}]))(b_{m:v_2}[\mathrm{true}]) \end{array} \right\} \\
\quad \mathrm{dispose}\ u \\
\{\exists a, v_1.\ \mathrm{True}(a_{n:v_1}[\mathrm{true}]\} \\
\quad [@n]_T \mathrel{+}= x; \\
\{\exists a, v_1.\ \mathrm{True}(a_{n:v_1}[\mathrm{true}]\} \\
\quad [@n]_V = \mathrm{nil} \\
\{\mathrm{true}\}
\end{array}
$$

Hence the precondition of a non-faulting execution must satisfy $(0 \rhd \mathrm{True}(a_{n:@m}[\mathrm{true}]))(b_{m:v_2}[\mathrm{true}])$ for some $a$,$b$,$m$ and $v_2$. This assertion expresses exactly what we would expect: the pointer at $a_n$ must point somewhere, but not to a direct ancestor. Furthermore, we can now easily derive a general specification for the command, using tree variables as placeholders:

$$
\begin{array}{l}
\{(0 \rhd \mathrm{True}(a_{n:@m}[x]))(b_{m:v}[y])\} \\
\quad \mathrm{delink}(@n) \\
\{\mathrm{True}(a_{n:\mathrm{nil}}[x * b_{m:v}[y]])\}
\end{array}
$$

## *Relation to Ambient Logic*

A natural question is whether the weakest preconditions are expressible using (a variation of) the Ambient Logic to trees with pointers. The Ambient Logic can express updates at the top level of trees by using the parallel composition and branch adjoints to build contexts around the tree. What it cannot do directly is reason about update in an arbitrary context. Using a size argument, we can prove that the Ambient Logic cannot express the weakest preconditions without some form of recursion. By introducing the somewhere modality $\Diamond$, we obtain a limited form of context reasoning. However, we believe that this by itself is still not enough to express the weakest preconditions.

This is best illustrated with an example. Expressing the weakest precondition of dispose $@n$ with even a simple postcondition requires case by case analysis using the Ambient Logic with $\Diamond$. For example, the postcondition $b_{m_1:v_1}[b_{m_2:v_2}[0]]$ has weakest precondition

$$\{\exists a, v.\ (b_{m_1:v_1}[b_{m_2:v_2}[P] \mid P] \mid P) \wedge \Diamond a_{n:v}[\mathrm{true}]\}$$

where $P$ is $(a_{n:v}[\text{true}] \vee 0)$. Meanwhile, the precondition for the weakened postcondition $\Diamond b_{m_2:v_2}[\text{true}]$ is

$$\{\exists a, v.\, \Diamond a_{n:v}[\neg\Diamond b_{m_2:v_2}[\text{true}]] \wedge \Diamond b_{m_2:v_2}[\text{true}]\}$$

These preconditions are clearly not parametric in the postcondition. Moreover, we know that the first of these preconditions must imply the second, since weakest preconditions are monotonic with respect to the postcondition, and $b_{m_1:v_1}[b_{m_2:v_2}[0]]$ implies $\Diamond b_{m_2:v_2}[\text{true}]$. It is clearly not straightforward to prove this implication. In contrast, the implication for the weakest preconditions in Context Logic is trivial (where $\Diamond$ is just the context True):

$$\begin{array}{c}\{\exists a, v.\, (0 \rhd b_{m_1:v_1}[b_{m_2:v_2}[0]])(a_{n:v}[\text{true}])\} \\ \Downarrow \\ \{\exists a, v.\, (0 \rhd \Diamond b_{m_2:v_2}[\text{true}])(a_{n:v}[\text{true}])\}\end{array}$$

## 6. RELATION TO SEPARATION LOGIC

We adapt our logic to reasoning about heap structures: that is finite partial functions from locations to values. We represent heaps as a collection of unary cells $m \mapsto n$, where $m$ is the location of the cell and $n$ the value it contains. Values are locations and *nil*. By analogy with our tree model, we present heaps and contexts with the grammars

$$\begin{array}{rcl} H & = & 0 \mid m \mapsto n \mid H * H \\ C & = & - \mid C * H \mid H * C \end{array}$$

together with a structural congruence on heaps and contexts, such that $*$ is a commutative monoid with unit 0, and a well-formedness condition stipulating unique locations. A variable store $s$ maps variables $u$ to locations, and expressions $E$, $F$ take as values either locations or variables.

The key step in adapting our logic is to change the model-specific data and context assertions to fit the heap structure:

$$\begin{array}{rcl} P & ::= & K(P) \mid K \lhd P \mid 0 \mid E \mapsto F \mid \\ & & P \Rightarrow P \mid \text{false} \mid \exists u.\, P \\ K & ::= & - \mid P \rhd P \mid \\ & & K \Rightarrow K \mid \text{False} \mid \exists u.\, K \end{array}$$

Here we have two atomic heap assertions: 0 and $E \mapsto F$, which holds for the corresponding cell $[\![E]\!]s \mapsto [\![F]\!]s$ (cell $[\![E]\!]s$ containing value $[\![F]\!]s$). Note that we do not need special context assertions, since all the contexts are already expressible using the insertion adjoint $P \rhd Q$. In particular, a context of the form $P * -$ is expressible as $(0 \rhd P)$, as there is only one way to add a hole to a heap.

The resulting logic is essentially the assertion language of Separation Logic. The presence of a 0 allows us to define $*$ and $-\!\!*$ as described in Section 4. These coincide with the ones of Separation Logic. Furthermore, our Collapse result for BI (Theorem 12) gives us a translation of Context Logic assertions into the fragment defined by $*$ and $-\!\!*$. Thus, the Context Logic for heaps has the same expressive power as Separation Logic.

The connection between reasoning about heaps and trees goes further. Our storage model and commands for manipulating trees can be collapsed onto the heap to produce essentially the commands used in [12]. To do this, we discard tree variables but keep value variables, view pointers @$n$ as locations $n$, and restrict ourselves to nodes of the form $a_{n:V}[0]$, which we view as heap pointers $n \mapsto V$ (ignoring the tag $a$). The value assignment, lookup, update and disposal commands translate into their heap equivalents, as do the relevant small axioms, weakest preconditions, and even the derivation proofs [12]. For example, the weakest preconditions become:

$$\begin{array}{rll} \{P[E/u]\} & u = E & \{P\} \\ \{\exists v.(true * (E \mapsto v)) \wedge P[v/u]\} & u = [E]_V & \{P\} \\ \{\exists v.((E \mapsto F) -\!\!* P) * (E \mapsto v)\} & [E]_V = F & \{P\} \\ \{\exists v.P * (E \mapsto v)\} & \text{dispose } E & \{P\} \end{array}$$

The only command that does not translate neatly is *new*. This is unsurprising. For heaps, we do not need to specify *where* a new cell goes (there is only one place it can go), but for trees we do. This is because, unlike heaps, trees have no single location that is expressible locally.

## 7. APPLICATION TO TERM REWRITING

We now adapt our Context Logic to reason about term rewriting systems. Terms over a signature do not decompose as parallel composition of subterms, due to the fixed arity of function symbols. They do however decompose nicely as context/subtree pairs, showing that a $*$-like composition operator and empty term are not the essential primitives for all kinds of spatial reasoning. The simplicity of this adaptation to term rewriting demonstrates the generality of our Context Logic approach.

In this application we treat rewrite rules as atomic commands, and capture their locality by giving small axioms and showing that the weakest preconditions are derivable. Intuitively, rewrite rules act locally since, once the redex is identified, only the corresponding subterm is affected. We formalize this by considering located terms, where each occurrence of a function symbol $f$ is annotated with a location $n \in \mathcal{N}$. Terms and contexts are defined in Figure 9. As before, we require uniqueness of identifiers for well-formedness.

Our storage model consists of an environment, a variable store, and a working term. The environment maps term variables $x$ to terms, and is used only in the logic. The store maps location variables $u$ to locations. Expressions $E$ are terms containing term variables and function symbols annotated with location variables — see Figure 9 for the full description. We write $FV_{\mathcal{T}}(E)$ for the free term variables and $FV_L(E)$ for the free location variables of $E$. Finally, we define the command language consisting of atomic rewrite commands $E \to F$ (with $E$ and $F$ subject to restrictions described later) and sequencing:

$$\mathbb{C} = E \to F \mid \mathbb{C};\mathbb{C}$$

The command $E \to F$ finds matchings for the term variables in $E$, such that $E$ evaluates to a subterm of the working term, and then replaces that subterm with the one generated by substituting the matches into $F$, with fresh values assigned to $F$'s location variables. For example, the execution of the rewrite rule $f_u(x, y) \to g_v(x, h_w(y))$ in a store where $u = n$ will turn $f_m(f_n(c_p, c_{p'}))$ into $f_m(g_k(c_p, h_l(c_{p'})))$ with $k$ and $l$ fresh, and will assign $k$ to $v$ and $l$ to $w$.

We must place the following restrictions on $E$ and $F$ in the command $E \to F$. For $\vec{x} = FV_{\mathcal{T}}(E)$, $\vec{u} = FV_L(E)$, $\vec{y} = FV_{\mathcal{T}}(F)$, and $\vec{v} = FV_L(F)$, we require that:

1. each $x \in \vec{x}$ occurs exactly once in $E$ and $\vec{y} \subseteq \vec{x}$, which corresponds to the definition of a rewrite rule;

2. each $y \in \vec{y}$ occurs exactly once in $F$, a linearity condition needed to preserve well-formedness;

$$\begin{array}{rcl} \text{terms } T & ::= & f_n(T_1, T_2, \ldots T_k) \quad k \geq 0 \\ \text{contexts } C & ::= & - \mid f_n(T_1, \ldots, C, \ldots T_k) \quad k \geq 1 \end{array}$$

$$\begin{array}{rcl} \text{term variables Var}_{\mathcal{T}} & = & \{x, y, \ldots\} \\ \text{location variables Var}_{\mathcal{L}} & = & \{u, v, \ldots\} \end{array} \qquad \text{expressions } E ::= \text{Var}_{\mathcal{T}} \mid f_u(E_1, E_2, \ldots E_k)$$

$$\begin{array}{l} \text{environment } \eta \in (\text{Var}_{\mathcal{T}} \rightharpoonup_{\text{fin}} T) \\ \text{store } s \in (\text{Var}_{\mathcal{L}} \rightharpoonup_{\text{fin}} \mathcal{N}) \end{array} \qquad \text{valuations } [\![E]\!]\eta s : \begin{array}{l} [\![x]\!]\eta s = \eta(x) \\ [\![f_u(E_1 \ldots E_k)]\!]\eta s = f_{s(u)}([\![E_1]\!]\eta s \ldots [\![E_k]\!]\eta s) \end{array}$$

**Figure 9: Term Rewrite: Data & Storage Models**

$$\textbf{Command:} \quad \frac{T = C(T_E) \quad T_E = E[\vec{T}/\vec{x}][s(\vec{u})/\vec{u}] \quad T_F = F[\vec{T}/\vec{x}][\vec{m}/\vec{v}] \quad C(T_F) \text{ well-formed}}{E \to F, s, T \rightsquigarrow [s|\vec{v} \leftarrow \vec{m}], C(T_F)}$$

$$\text{where } \vec{x} = FV_{\mathcal{T}}(E), \vec{u} = FV_L(E) \text{ and } \vec{v} = FV_L(F)$$

$$\textbf{Logic:} \quad \begin{array}{l} \eta, s, T \vDash_{\mathcal{T}} E \Leftrightarrow T = [\![E]\!]\eta s \\ \eta, s, C \vDash_{\mathcal{K}} f_u(P_1 \ldots, K, \ldots P_k) \Leftrightarrow \exists C'. C = f_{s(u)}(T_1 \ldots, C', \ldots T_k),\ \eta, s, T_i \vDash_{\mathcal{T}} P_i,\ \eta, s, C' \vDash_{\mathcal{K}} K \end{array}$$

**Figure 10: Term Rewrite: Command & Logic Semantics**

3. $E$ is of the form $f_u(\ldots)$ and not a term variable $\text{Var}_{\mathcal{T}}$, so that the rewriting acts locally at $u$; and

4. each $v \in \vec{v}$ occurs exactly once in $F$, allowing fresh location values to be assigned in parallel.

The operational semantics of $E \to F$ is given in Figure 10. The free variables of $E \to F$ are $\vec{u} \cup \vec{v}$, whilst the modified variables are just $\vec{v}$. Since term variables are only used internally for pattern matching, they are neither modified nor free, which explains why the environment is not a parameter of the operational semantics.

We can now easily adapt our context logic approach to reasoning about terms and rewrites. As before, the key is to change the model-specific assertions, in this case given by term expressions $E$ and term contexts $f_u(P \ldots, K, \ldots P)$. The tree and context assertions are given by the grammars:

$$\begin{array}{l} P ::= K(P) \mid K \lhd P \mid E \mid \\ \qquad P \Rightarrow P \mid \text{false} \mid \exists u. P \\ K ::= - \mid P \rhd P \mid f_u(P, \ldots, K, \ldots P) \mid \\ \qquad K \Rightarrow K \mid \text{False} \mid \exists u. K \end{array}$$

The semantics for the model-specific assertions is given in Figure 10.

Our Hoare logic for term rewriting consists of small axioms for each rewriting command, the Frame Rule, and the other inference rules. The small axiom for $E \to F$ is simply

$$\{E[\vec{x'}/\vec{x}]\} \quad E \to F \quad \{F[\vec{x'}/\vec{x}]\}$$

The substitution $\vec{x'}/\vec{x}$ reflects the fact that the term variables $\vec{x}$ are bound in the command, and can hence be renamed in the logic. The weakest precondition axiom is

$$\{(\forall \vec{v'}. (F[\vec{v'}/\vec{v}] \rhd P[\vec{v'}/\vec{v}]))(E)\} \quad E \to F \quad \{P\}$$

where $\vec{x} = FV_{\mathcal{T}}(E)$, $\vec{v} = FV_L(F)$, $\vec{x} \cap FV_L(P) = \{\}$ and $\vec{v'} \cap FV_L(F, P) = \{\}$. The derivation from the small axiom is straightforward, using the Frame Rule with context $\forall \vec{v'}.(F[\vec{v'}/\vec{v}] \rhd P[\vec{v'}/\vec{v}])$, and the Rule of Consequence.

## 8. CONCLUSIONS AND FUTURE WORK

Although here we concentrated on a specific presentation of boolean Context Logic, there is ongoing work on categorical semantics and completeness for other variants that include more structure on contexts – context composition – and where embedding and projection take a more prominent role. This will appear in a future paper.

We have presented a Hoare logic for reasoning locally about trees, introducing Context Logic and deriving weakest preconditions. It is possible to describe our Hoare triples using the Ambient Logic. However, it is not possible to derive the weakest preconditions, since although we have given such conditions for specific cases in Section 5, it seems clear that we cannot do it parametrically. We have an inexpressivity result for the Ambient Logic without recursion, but it remains future work to pin down such a result for the Ambient Logic with a somewhere modality.

Our original motivation for reasoning about tree update was to reason about update languages for XML. In this paper, we focus on a simple imperative language for manipulating trees, which is expressive enough to illustrate the subtleties of tree update. In future, we will develop a full language for XML update, incorporating features such as commands for updating nodes identified by path expressions, and a local update command that renames node identifiers to avoid name clashes.

We provide a general theory of boolean Context Logic, with a Hilbert-style proof theory, forcing semantics and models. We have shown soundness, and Yang and Calcagno have recently proved completeness. We also extend our results to the Context Logic with an additional 0 data element. This element is present in the heap and tree model (empty heap or tree), but is not present in term rewriting. With the 0 element, we can derive a $*$-like composition on data with two corresponding right adjoints. When $*$ is associative and commutative, then the corresponding two right adjoints coincide and we obtain BI. We are only at the beginning of a general study of Context Logic. We are currently studying the categorical semantics, and the corresponding models and completeness results (with Yang).

We have shown the generality of our Hoare logic reasoning, by adapting our reasoning about tree update to heap update and term rewriting. The heap update example illustrates that not only does our Context Logic collapse to BI, but also that our Hoare reasoning about trees collapses to that of heaps. Our term rewriting example illustrates that our focus on contexts is more fundamental than the multiplicative structure of BI ($0$ and $*$), and suggests that our Hoare logic reasoning is robust with respect to the style of update language chosen. These examples demonstrate the generality of our Context Logic reasoning. In future, we will extend this work to reason more generally about data update. For example, a natural next step is to extend our work to more complex data structures involving name binding.

*Acknowledgments.* We would like to thank Peter O'Hearn, Pino Rosolini and Hongseok Yang for their insightful comments. This work was supported by EPSRC.

## 9. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. Data on the Web: from relations to semistructured data and XML. Morgan Kaufmann, 1999.

[2] N. Biri and D. Galmiche. A separation logic for resource distribution. In *23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2003.

[3] C. Calcagno, P. Gardner, and U. Zarfaty. A context logic for tree update. In *Proceedings of Workshop on Logics for Resources, Processes and Programs (LRPP'04)*, 2004.

[4] L. Cardelli, P. Gardner, and G. Ghelli. Querying trees with pointers. Unpublished Notes, 2003; talk at APPSEM 2001.

[5] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Comput. Sci.*, 240:177–213, 2000.

[6] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *28th POPL*, pages 14–26, London, January 2001.

[7] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In L. Fribourg, editor, *Computer Science Logic (CSL'01)*, pages 1–19. Springer-Verlag, 2001. LNCS 2142.

[8] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications.* Applied Logic Series. Kluwer Academic Publishers, 2002. http://www.dcs.qmul.ac.uk/~pym/Papersage/bunch.ps.

[9] J. Reynolds. Separation logic: a logic for shared mutable data structures. Invited Paper, LICS'02, 2002.

[10] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. SIGMOD 2001, Santa Barbara, CA.

[11] H. Yang and C. Calcagno. Completeness results for Context Logic and BI. In preparation, 2004.

[12] H. Yang and P. O'Hearn. A semantic basis for local reasoning. Proceedings of FOSSACS, 2002.