A Note on Context Logic

Philippa Gardner

Imperial College London

This note describes joint work with Cristiano Calcagno and Uri Zarfaty. It introduces the general theory of Context Logic, and has been written for the Appsem Summer School 2005. It should be read in conjunction with Reynold's paper on Separation Logic: A Logic for shared Mutable data Structures, and our paper on Context Logic and Tree Update. We acknowledge support from EPSRC.

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 systems. 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 however hardly been studied for other examples of update, such as tree update, and there has been little attempt at a unified theory.

Calcagno, Gardner and Zarfaty have studied Hoare reasoning about data update, using Context Logic for reasoning *locally* about structured data [CGZ05]. Our work arose from the work of O'Hearn, Reynolds and Yang on reasoning locally about heap update using Separation Logic [Rey02,YO02,IO01]. Their pioneering idea is that, if an update only accesses part of the data, leaving the rest unchanged, then this locality property should also be reflected in the reasoning. The original Hoare Logic based on First-order Logic did not have this property. In contrast, Separation Logic, a Hoare Logic based on Bunched Logic of O'Hearn and Pym [OP99,Pym02], does have it. With Separation Logic, they are able to reason about examples of heap update which had escaped modular reasoning in the traditional Hoare-logic style for over thirty years: pointer arithmetic, concurrent imperative programs and passivity.

We have recently shown that the techniques for reasoning locally about heap update can be adapted to reason locally about tree update (XML update) [CGZ05]. This work was by no means straightforward. We initially assumed that we could develop a Hoare logic based on Cardelli and Gordon's Ambient Logic, a logic for reasoning locally about static trees (ambients, firewalls, XML). However, Ambient Logic does not have enough expressive power to reason about local update. Instead, we had to fundamentally change the way we reason about structured data. We introduced Context Logic, which analyses both data and contexts. Local 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.

In this note, we provide a gentle introduction to the fundamental theory of Context Logic. We first introduce the logic using examples of structured data, emphasising the common nature of the logical reasoning. We use these examples to illustrate the differences and similarities between Context Logic, Bunched Logic and Ambient Logic. These examples also motivate our general theory of Context Logic. We provide the general models, the forcing semantics and a Hilbert-style proof theory for the basic logic. Calcagno and Yang have shown completeness [YC05]. We then incorporate an additional zero formula, since many of our example models have a zero data element, and show that it yields interesting logical structure. Again, we provide the general models and proof theory.

This note illustrates that, for our examples of structured data, there is an intuitive–but adhoc–way of presenting the Context Logic for that data structure. One future challenge is to capture the spirit of this informal intuition, by identifying an appropriate general account of structured data. As a first step, we will look at Fiore's generalised species of structures [Fio05]. In [CGZ05], we give preliminary examples of local Hoare reasoning about data update based on Context Logic reasoning: tree update, heap update which is exactly analogous to reasoning using Separation Logic, and term rewriting which had escaped reasoning using Separation Logic. In each of these examples, the results and proofs associated with our Hoare reasoning are remarkably similar. They give us some assurance that, in the long-term, we will be able to find a unified theory of data update using Context Logic.

2 Using Context Logic

We explore several examples of using Context Logic to reason about structured data. Each example illustrates a different aspect of Context Logic reasoning. We have purposely unified these examples as much as possible, as a way of introducing the general theory of Context Logic given in section 3.

2.1 Context Logic for Sequences

As our first example of structured data, we use the set of sequences. It provides a simple example to illustrate the difference between reasoning in Context Logic and reasoning in Bunched Logic. We define a data set of sequences $\mathcal{D}$ and set of contexts $\mathcal{C}$, constructed from a signature set of names Σ:

$$\begin{array}{ll} \text{sequences} & d ::= 0 \mid a \in \Sigma \mid d \cdot d \\ \text{contexts} & c ::= \_ \mid d \cdot c \mid c \cdot d \end{array}$$

We declare the following equations on sequences and contexts, which state that the join operator $\_ \cdot \_$ is associative with identity 0:

$$\begin{array}{cc} 0 \cdot d = d \cdot 0 = d & 0 \cdot c = c \cdot 0 = c \\ d_1 \cdot (d_2 \cdot d_3) = (d_1 \cdot d_2) \cdot d_3 & d_1 \cdot (d_2 \cdot c) = (d_1 \cdot d_2) \cdot c \\ (c \cdot d_1) \cdot d_2 = c \cdot (d_1 \cdot d_2) & (d_1 \cdot c) \cdot d_2 = d_1 \cdot (c \cdot d_2) \end{array}$$

The application function $ap : \mathcal{C} \times \mathcal{D} \rightarrow \mathcal{D}$ is defined in the obvious way:

$$\begin{aligned} ap\,(\_, d) &= d \\ ap\,(d' \cdot c, d) &= d' \cdot ap\,(c, d) \\ ap\,(c \cdot d', d) &= ap\,(c, d) \cdot d' \end{aligned}$$

We often write $c(d)$ for $ap\,(c, d)$.

Definition 1 (Formulae for Sequences). *The Context Logic for Sequences consists of a set of* data formulae $\mathcal{P}_{\mathcal{D}}$ *for describing properties of sequences and a set of* context formulae $\mathcal{K}_{\mathcal{C}}$ *for describing properties of contexts. These formulae are constructed from signature set Σ and variable set $\mathcal{V}$, and are given by the grammars:*[1]

$$\begin{array}{lrll} \textit{data formulae} & P ::= & K(P) \mid K \lhd P & \textit{structural formulae} \\ & & P \Rightarrow P \mid \textit{false} & \textit{additive formulae} \\ & & 0 \mid u \mid P \cdot P & u \in \Sigma \cup \mathcal{V},\ \textit{specific data formulae} \\ & & \exists x.P & x \in \mathcal{V},\ \textit{quantification} \\ \textit{context formulae} & K ::= & I \mid P \rhd P & \textit{structural formulae} \\ & & K \Rightarrow K \mid \textit{False} & \textit{additive formulae} \\ & & P \cdot K \mid K \cdot P & \textit{specific context formulae} \\ & & \exists x.K & x \in \mathcal{V},\ \textit{quantification} \end{array}$$

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 a sequence can be split into a context satisfying K applied to a subsequence satisfying P. For example, if we define the context formula True $\triangleq \neg$False, then the formula True(P) states that there is a subsequence satisfying property P. The next two formulae are both (right) adjoints of application. The formula $K \lhd P$ is satisfied by a given sequence if, whenever we insert the sequence into a context satisfying K, then the result satisfies P. For example, the formula True $\lhd P$ states that, when the sequence is put in any context, the resulting sequence 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 a sequence 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,

[1] It is also simple and natural to include a formula for context composition and the corresponding adjoints. We choose to omit these formulae since, so far, they have not been necessary for our results. For simplicity, we also choose to work with one-holed contexts rather than multi-holed contexts. It is simple to extend to the more general setting.

regardless of what data is put in the context hole, the resulting sequence satisfies property P_2. This adjoint is essential for expressing weakest preconditions for update commands. The context formula I specifies that a context equals the empty context.

The specific data and context formulae arise from the specific data model under consideration, in this case sequences. Since we do not start from a general description of data, but rather just look at examples, this part of the logic is adhoc. We will see however that the formulae are generated in a uniform way from the grammars. In the case of sequences, the specific data formulae consist of the formula 0 specifying that the sequence is empty, the formula u specifying individual names and name variables, and the join formula $P_1 \cdot P_2$ specifying that the sequence can be split into two parts, the first satisfying P_1 and the second P_2. This join formula is in fact logically equivalent to $(P_1 \cdot \_)(P_2)$. We choose to define it directly, since the construction of the specific data formulae then follows the data grammar. The construction of the specific context formulae $P \cdot K$ and $K \cdot P$ follows the grammar of contexts: the formula $P \cdot K$ for example denotes that a context can be split into a sequence satisfying P and a context satisfying K.

The forcing semantics is given in definition 2 using two satisfaction relations: the judgement $\sigma, d \vDash_D P$ states that data formula P holds for a given sequence d and a substitution σ that evaluates variables to names, whilst judgement $\sigma, c \vDash_K K$ states that context formula K holds for context c and substitution σ.

Definition 2 (Satisfaction for Sequences). *We define two satisfaction relations $\sigma, d \vDash_D P$ and $\sigma, c \vDash_K K$, by induction on the structure of data and context formulae respectively:*

$$\begin{array}{ll}
\sigma, d \vDash_D K(P) & \textit{iff } \exists c \in \mathcal{C}, d' \in \mathcal{D}.\ ap\,(c, d') = d \textit{ and } \sigma, c \vDash_K K \textit{ and } \sigma, d' \vDash_D P \\
\sigma, d \vDash_D K \lhd P & \textit{iff } \forall c \in \mathcal{C}.(\sigma, c \vDash_K K \textit{ implies } \sigma, ap\,(c, d) \vDash_D P) \\
\sigma, d \vDash_D P_1 \Rightarrow P_2 & \textit{iff } \sigma, d \vDash_D P_1 \textit{ implies } \sigma, d \vDash_D P_2 \\
\sigma, d \nvDash_D \textit{false} & \\
\sigma, d \vDash_D 0 & \textit{iff } d = 0 \\
\sigma, d \vDash_D u & \textit{iff } \sigma(u) = d \\
\sigma, d \vDash_D P_1 \cdot P_2 & \textit{iff } \exists d_1, d_2.\ \sigma, d_1 \vDash_D P_1 \textit{ and } \sigma, d_2 \vDash_D P_2 \\
\sigma, d \vDash_D \exists x.P & \textit{iff } \exists a \in \Sigma.\ \sigma\{a/x\}, d \vDash_D P \\
\sigma, c \vDash_K I & \textit{iff } c = \_ \\
\sigma, c \vDash_K P_1 \rhd P_2 & \textit{iff } \forall\, d \in \mathcal{D}.(\sigma, d \vDash_D P_1 \textit{ implies } \sigma, ap\,(c, d) \vDash_D P_2) \\
\sigma, c \vDash_K K_1 \Rightarrow K_2 & \textit{iff } \sigma, c \vDash_K K_1 \textit{ implies } \sigma, c \vDash_k K_2 \\
\sigma, c \nvDash_K \textit{False} & \\
\sigma, c \vDash_K P \cdot K & \textit{iff } \exists d,\ c_1.\quad c = d \cdot c_1 \textit{ and } \sigma, d \vDash_D P \textit{ and } \sigma, c_1 \vDash_K K \\
\sigma, c \vDash_K K \cdot P & \textit{iff } \exists c_1, d.\quad c = c_1 \cdot d \textit{ and } \sigma, c_1 \vDash_K K \textit{ and } \sigma, d \vDash_D P \\
\sigma, c \vDash_K \exists x.K & \textit{iff } \exists a \in \Sigma.\ \sigma\{a/x\}, c \vDash_K K
\end{array}$$

We omit the subscripts D and K when the intended meaning is clear.

The cases for the structural formulae have been discussed already. The cases for the additive formulae and quantifiers are standard. We use the standard derived classical formulae for both data and context formulae: $\neg P$, true, $P \wedge P$, $P \vee P$ and $\forall x.\, P$; for contexts, we write True rather than true. We shall also use the following derived formulae arising from the structural formula:

– $\diamond P \triangleq \mathrm{True}(P)$ specifies that somewhere property P holds;
– $P_1 * P_2 \triangleq (0 \triangleright P_1)(P_2)$ specifies that it is possible to remove a subsequence satisfying P_2, and put in the empty sequence in the hole to obtain a sequence satisfying P_1;
– $P_1 \blacktriangleright P_2 \triangleq \neg(P_1 \triangleright \neg P_2)$ specifies that *there exists* a sequence satisfying property P_1 such that, when it is put in the hole of the given context, the resulting sequence satisfies P_2;
– $K \blacktriangleleft P_2 \triangleq \neg(K \triangleleft \neg P_2)$ specifies that *there exists* a context satisfying property K such that, when the given sequence is put in the hole, the resulting sequence satisfies P_2.

The order of binding precedence is: $\neg$, $\diamond$, $\{\cdot, *\}$, $\wedge$, $\vee$, $\{\triangleleft, \triangleright, \blacktriangleleft, \blacktriangleright\}$, $\Rightarrow$, $\exists$, $\forall$. There is no precedence between the elements in $\{\cdot, *\}$ and $\{\triangleleft, \triangleright, \blacktriangleleft, \blacktriangleright\}$, since it would be impossible to remember!

Notice that the cases for the specific formulae are given up to the equational theory of sequences. In fact, it is a typical result of this style of logic that there is always a formula that can distinguish between data terms that are not equal. This strength of analysis is exactly what is required to reason about heap and tree update.

Proposition 3. *Logical equivalence corresponds to sequence and context equality:*

$$\begin{array}{lll} d_1 = d_2 & \textit{iff} & \forall P, \sigma.\ (\sigma, d_1 \vDash_D P \ \ \textit{iff}\ \ \sigma, d_2 \vDash_D P) \\ c_1 = c_2 & \textit{iff} & \forall K, \sigma.\ (\sigma, c_1 \vDash_D K \ \ \textit{iff}\ \ \sigma, c_2 \vDash_D K) \end{array}$$

The proof uses the so-called characteristic data and context formulae: for example, given sequence d, the characteristic data formula P_d is a formula satisfying, for all d_1, $\sigma, d_1 \vDash_D P_d$ if and only if $d = d_1$. The proof is left as an exercise.

We shall see that derived formula $P_1 * P_2$ plays an important role in the general theory of Context Logic. It is essential for reasoning about update, such as the removal of a subsequence. We shall fully explore this derived formula in section 3.2, in particular describing the derived adjoints. At this stage, we just give a few sample formulae using $*$, to illustrate its expressive power:

$\mathrm{true} * a$	contains a
$(\mathrm{true} * a) * a$	contains two as
$\mathrm{true} * (a * a)$	contains two consecutive as
$a * \mathrm{true}$	a at the beginning or the end of the sequence

The first formula simply states that it is possible to take an a out of the sequence. Contrast the second and third formulae. The second states that two separate as can be removed, but nothing about their relative positions in the sequence. The third formula states that two adjacent as can be removed. Thus, the derived $*$-connective is not associative. The last formula is quite an odd property. It states that a subsequence can be removed to leave an a. This can only happen if there is an a either at the beginning or the end of the sequence, since only one subsequence can be removed. The first and last formulae illustrate that, for sequences, the $*$ is not commutative.

It is instructive to compare the specific formula $P_1 \cdot P_2$ with the general formula $P_1 * P_2$:

$\mathrm{true} \cdot a$	a at the end of the sequence
$\mathrm{true} \cdot a \cdot \mathrm{true} \cdot a \cdot \mathrm{true}s$	contains two as
$\mathrm{true} \cdot a \cdot \mathrm{true}$	contains an a
$a \cdot \mathrm{true}$	a at the beginning of the sequence

We can also derive the two right adjoints for $\_ \cdot \_$: the formula $P_1 \mathbin{\cdot\!\!-} P_2 \triangleq (P_1 \cdot \_) \triangleleft P_2$ 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, the formula $P_1 \mathbin{-\!\!\cdot} P_2 \triangleq (\_ \cdot P_2) \triangleleft P_1$ specifies that, whenever a sequence satisfying property P_1 is joined to the right of the given sequence, then the result satisfies P_2. For example, the formula $a \mathbin{-\!\!\cdot} P$ specifies that, when a is joined to the right of the sequence, then the resulting sequence satisfies P. Similarly for $a \mathbin{\cdot\!\!-} P$. In contrast, notice that the formula $(a \triangleright P)(0)$ specifies that, whenever an a is put *somewhere* in the given sequence, then the result satisfies property P.

A natural question is whether the Context Logic for sequences is more powerful than a natural adaptation of Bunched Logic to sequences. This question can be formulated within our setting, by asking whether every data formula for the Context Logic for sequences is as expressive as the subset of data formulae generated by the additive formulae, the specific data formulae, quantification, and different structural data formulae given by

$$P \cdot P \mid P \mathbin{-\!\!\cdot} P \mid P \mathbin{\cdot\!\!-} P.$$

The satisfaction relation for these data formulae is defined as for the derived formulae described above.

It takes some care to formulate the question of whether the sublogic is less powerful than the full Context Logic for sequences. Consider the Context Logic formula $P * a$. We expect that, for every specific example of $P * a$, there is an equivalent data formula in the sublogic: that is, logical equivalence for the sublogic corresponds to logical equivalence of the full logic. For example, the formula $(b \cdot c) * a$ is equivalent to $a \cdot b \cdot c \vee b \cdot a \cdot c \vee b \cdot c \cdot a$. However, we also believe that it is not possible to give a formula in the sublogic which is equivalent to $P * a$ and is *parametric* in P: that is, does not depend on the structure of P. For example, the formula $\mathrm{True}(b) * a$ is equivalent to $\mathrm{true} \cdot b \cdot \mathrm{true} \cdot a \cdot \mathrm{true} \vee \mathrm{true} \cdot a \cdot \mathrm{true} \cdot b \cdot \mathrm{true}$, which has very different structure to the previous example. This parametricity property is essential for expressing the weakest preconditions, and is on-going work.

2.2 Context Logic for Multisets

The set of multisets provides the simplest example of a data model where reasoning using Context Logic and Bunched Logic is essentially the same. Multisets can be described in a similar fashion to sequences, with the extra condition that $\_\cdot\_$ is commutative. With commutativity, notice that there is a bijection between multisets and contexts, given by $p : \mathcal{C} \to \mathcal{D}$ where $p(c) = c(0)$. In contrast, the corresponding p for sequences is surjective, but not injective. We shall see that this property has a significant effect on Context Logic: for multisets (and heaps), the logic collapses to Bunched Logic; for sequences (and trees), it does not.

Definition 4 (Context Logic for Multisets). *The formulae and satisfaction relation for multisets is defined as for sequences, with the satisfaction relation now depending on the stronger multiset equality.*

In this case, we have $P_1 \cdot P_2$ logically equivalent to $P_1 * P_2$ as well as $(P_1 \cdot \_)P_2$. In addition, $K \cdot P$ and $P \cdot K$ are logically equivalent to $K(P)$.

With multisets, the structure of Context Logic collapses to the Bunched Logic case. In particular, it is enough to work with a subset of the data formulae consisting of the additive formulae, the specific data formulae, quantification and a subset of the structural data formulae defined by[2]

$$P \cdot P \mid P \mathbin{-\!\cdot} P$$

We let $\mathcal{D}_f$ denote this fragment of data formulae.

Proposition 5. *There exist translations* $\widehat{\_} : \mathcal{K}_{\mathcal{C}} \to \mathcal{P}_{\mathcal{D}_f}$ *and* $\widehat{\_} : \mathcal{P}_{\mathcal{D}} \to \mathcal{P}_{\mathcal{D}_f}$ *such that*

$$\sigma, c \vDash_K K \Leftrightarrow \sigma, c(0) \vDash_D \widehat{K}$$
$$\sigma, d \vDash_D P \Leftrightarrow \sigma, d \vDash_D \widehat{P}$$

Proof The translations are defined inductively on the structure of data and context formulae. We just give the cases for the structural formulae and the specific formulae; the cases for the additive formulae and quantification just follow the structure of the formulae:

$$\begin{array}{ll} \widehat{K(P)} \triangleq \widehat{K} \cdot \widehat{P} & \widehat{I} \triangleq 0 \\ \widehat{K \lhd P} \triangleq \widehat{K} \mathbin{-\!\cdot} \widehat{P} & \widehat{P_1 \rhd P_2} \triangleq \widehat{P_1} \mathbin{-\!\cdot} \widehat{P_2} \\ \widehat{0} \triangleq 0 & \widehat{u} \triangleq u \end{array}$$

The results follow by induction on the structure of the data and context formulae. We leave the proof as an exercise.

[2] The formula $P \mathbin{\cdot\!-} P$ is logically equivalent to $P \mathbin{-\!\cdot} P$.

2.3 Context Logic for Heaps

We now introduce our heap model. Our Context Logic reasoning for heaps has much in common with the multiset example. The difference is due to the *partiality* of the application operation, since location names must be unique.

We consider unary heaps, which are finite partial functions from locations to values. We present them as a collection of unary cells $a \mapsto b$, where a, b are elements from signature set $\Sigma \cup \{\text{nil}\}$: the $a \in \Sigma$ denotes the unique location name of the cell, and $b \in \Sigma \cup \{\text{nil}\}$ the value it contains. The adaptation of this case to n-ary cells is trivial. We define a data set of heaps, denoted $\mathcal{D}$, and a set of contexts $\mathcal{C}$, using $l(d)$ to denote the set of locations in heap d and similarly $l(c)$ for contexts:

$$\begin{array}{ll} \text{heaps} & d = 0 \mid a \mapsto b \mid d_1 \cdot d_2, \quad l(d_1) \cap l(d_2) = \emptyset \\ \text{contexts} & c = \_ \mid c \cdot d \mid d \cdot c, \quad l(d) \cap l(c) = \emptyset \end{array}$$

Heaps and contexts satisfy analogous equations to those satisfied by multisets. The key difference between heaps and multisets is the enforced uniqueness of the location names, and therefore the partiality of the $\_ \cdot \_$ operation and application. This partiality is factored into the definition of satisfaction.

Definition 6 (Formulae for Heaps). *The Context Logic for heaps consists of data formulae and context formulae constructed from the signature set $\Sigma \cup \{nil\}$ and variable set $\mathcal{V}$. They are defined as for sequences, except that the specific data and context formulae are defined by*

$$\begin{array}{lll} \textit{specific data formulae} & 0 \mid u \mapsto v \mid P \cdot P & u, v \in \Sigma \cup \{nil\} \cup \mathcal{V} \\ \textit{specific context formulae} & P \cdot K \mid K \cdot P & \end{array}$$

Again, the formulae $P \cdot P$, $P \multimap P$, $P \cdot\!\!- P$, $P \cdot K$ and $K \cdot P$ are derivable. We assume that $\mapsto$ binds more strongly than the other operators.

Definition 7 (Satisfaction for Heaps). *As for the sequence case (definition 2), we define two satisfaction relations $\sigma, d \vDash_D P$ and $\sigma, c \vDash_K K$, where in this case d denotes a heap and c denotes a heap context. These relations are defined by induction on the structure of heap and context formulae respectively. We just give the cases which differ from definition 2:*

$$\begin{array}{l} \sigma, d \vDash_D K \lhd P \;\; \textit{iff} \\ \qquad \forall c \in \mathcal{C}.(\sigma, c \vDash_K K \textit{ and } c(d') \textit{ defined implies } \sigma, c(d) \vDash_D P) \\ \sigma, d \vDash_D u \mapsto v \quad \textit{iff } d = \sigma(u) \mapsto \sigma(v) \\ \sigma, c \vDash_K P_1 \rhd P_2 \textit{ iff} \\ \qquad \forall d \in \mathcal{D}.(\sigma, d \vDash_D P_1 \textit{ and } c(d') \textit{ defined implies } \sigma, c(d) \vDash_D P_2) \end{array}$$

In section 2.2, we showed how the structure of Context Logic collapses in the multiset case. This collapse also occurs for heaps.

The formula $u \mapsto v$ specifies that the given heap has just *one* cell, with location given by $\sigma(u)$ and value given by $\sigma(v)$. Here are some additional definitions and properties about locations and pointers:

$u \mapsto v \wedge u \mapsto v'$	one cell with location $\sigma(u)$ and value $\sigma(v) = \sigma(v')$
$u \mapsto \_ \triangleq \exists x.\ u \mapsto x$	one cell with location $\sigma(u)$ and any value
$u \mapsto \_ \wedge v \mapsto \_$	one cell with location $\sigma(u) = \sigma(v)$
$u \mapsto \_ * v \mapsto \_$	two different cells with locations $\sigma(u) \neq \sigma(v)$
$u \hookrightarrow \_ \triangleq u \mapsto \_ * \mathrm{true}$	at least a cell with address $\sigma(u)$
$u \hookrightarrow \_ * v \hookrightarrow \_$	at least two different cells
$u \hookrightarrow v * v \hookrightarrow u$	at least two different cells which point to each other

2.4 Context Logic for Trees

We discovered Context Logic by studying trees, in particular recognising that the Ambient Logic does not have enough expressive power to reason about tree update. Consider the data set of trees $\mathcal{D}$ and set of contexts $\mathcal{C}$, again constructed from a signature set of names $a \in \Sigma$:

$$\begin{array}{ll} \text{trees} & d ::= 0 \mid a[d] \mid d \cdot d \\ \text{contexts} & c ::= \_ \mid a[c] \mid d \cdot c \mid c \cdot d \end{array}$$

with the analogous equations on trees and contexts as for the sequence case. The application function is defined as before, with the additional case

$$ap\,(a[c], d) = a[ap\,(c, d)]$$

We have also studied the cases when $\_ \cdot \_$ is commutative for trees and contexts (as in the Ambient Calculus), and when the names are unique resulting in application being partial (viewing the names as node identifiers). Even when $\_ \cdot \_$ is commutative, Context Logic reasoning is stronger than Bunched Logic reasoning.

Definition 8 (Formulae for Trees). *The Context Logic for trees consists of* data formulae *and* context formulae *constructed from the signature set Σ and variable set $\mathcal{V}$. They are defined as for sequences, except that the specific data and context formulae are defined by*

$$\begin{array}{lll} \textit{specific data formulae} & 0 \mid u[P] \mid P \cdot P & u \in \Sigma \cup \mathcal{V} \\ \textit{specific context formulae} & u[K] \mid P \cdot K \mid K \cdot P & u \in \Sigma \cup \mathcal{V} \end{array}$$

The only formulae requiring explanation are $u[P]$ and $u[K]$, which states that a tree has a top node denoted by $\sigma(u)$ and subtree satisfying property P; similarly for $u[K]$.

Definition 9 (Satisfaction for trees). *The satisfaction relations for trees are defined analogously to the relations given in definition 2 for sequences. The only case not covered by that definition are*

$$\begin{array}{l} \sigma, d \vDash_D u[P] \ \textit{iff} \ \exists d' \in \mathcal{D}.\ d = \sigma(u)[d'] \ \textit{and} \ \sigma, d' \vDash_D P \\ \sigma, c \vDash_K u[K] \ \textit{iff} \ \exists c' \in \mathcal{C}.\ c = \sigma(u)[c'] \ \textit{and} \ \sigma, c' \vDash_K K \end{array}$$

As before, $P_1 \cdot P_2$, $P_1 -\!\!\cdot P_2$ and $P_1 \cdot\!\!- P_2$ are derivable from the other constructs. The join formula allows us to reason about the horizontal structure of tree: for example, $b[\mathrm{true}] \cdot a[\mathrm{true}]$ denotes a tree with two top nodes b and a, with b to the left of a. When $\_ \cdot \_$ is commutative, the ordering between b and a is lost. The adjoints $\_ -\!\!\cdot \_$ and $\_ \cdot\!\!- \_$ let us reason about properties that are satisfied when given trees are added horizontally. When $\_ \cdot \_$ is commutative, then $P_1 -\!\!\cdot P_2$ is equivalent to $P_1 \cdot\!\!- P_2$. Notice that $u[P]$ is logically equivalent to $(u[\_])(P)$, and states that a tree has top node denoted by u and subtree satisfying property P. The node formulae allow us to express vertical path information: for example, the formula $\mathrm{true} \cdot b[\mathrm{true} \cdot c[\mathrm{true}] \cdot \mathrm{true}] \cdot \mathrm{true}$ specifies that the given tree has a path $b \backslash c$. It has a derivable right adjoint $u \propto P' \triangleq u[\_] \lhd P'$ which states that, when a tree is put under a top node u, then the resulting tree satisfies property P'. For example, the formula $b \propto (a[\mathrm{true}] -\!\!\cdot P_1)$ states that when a given tree is put in a context $b[\_] \cdot a[d]$ for an arbitrary tree d then the resulting tree satisfies property P_1. These derived formulae are all present in the (static) Ambient Logic.

Recall the derived formula $P_1 * P_2 \triangleq (0 \rhd P_1)(P_2)$. In the tree case, this specifies that a tree can be split into a subtree satisfying P_2 and a context such that, when 0 is put in the hole, then the resulting tree satisfies P_1. Even when $\_ \cdot \_$ is commutative, this formula is not equivalent to $P_1 \cdot P_2$, which only splits the tree at the top level. For example, the formula $b[c[0]] \cdot a[\mathrm{true}]$ describes a tree with two top nodes b and a. In contrast, with the formula $b[c[0]] * a[\mathrm{true}]$, the a node can be inside the b node. Just as for the sequence case, there is a natural question of whether Context Logic for trees with commutativity is as expressive as the (static) Ambient Logic. Using a size argument, we can prove that this is not the case without some form of recursion. With the somewhere modality $\diamond P$ added to the Ambient Logic, we believe we are in a similar situation to the sequence case. We can explore the expressivity of Context Logic for trees compared with the data subformulae constructed from the join and node formulae, their corresponding adjoints, and a modality operator. Given $P * a[0]$, we conjecture that, for each P, there is an equivalent formula expressible in the sublogic, but that this formula is not parametric in P.

2.5 Context Logic for Terms

So far, all our examples of structured data have had a zero element, and a horizontal join operator. As our final example, we study the set of terms arising from function symbols of fixed arity, with no zero element. We shall see that the zero element has an important role to play in our development of Context Logic (section 3.2). We therefore find the term example interesting, since it is a natural example which does not have this structure. Consider the data set of terms $\mathcal{D}$ and set of contexts $\mathcal{C}$, constructed from a signature Σ consisting of n-ary function symbols:

$$\begin{array}{lll} \text{terms} & d ::= f(d_1, \ldots, d_n), & f : n \in \Sigma \\ \text{contexts} & c ::= \_ \mid f(d_1, \ldots, c, \ldots, d_n), & f : n \in \Sigma \end{array}$$

The application function is

$$ap\,(\_, d) = d$$
$$ap\,(f(d_1, \ldots, c, \ldots, d_n), d) = f(d_1, \ldots, ap\,(c, d), \ldots, d_n)$$

Definition 10 (Formulae for Terms). *The Context Logic for terms consists of* data formulae *and* context formulae *constructed from the signature set Σ and variable set $\mathcal{V}$. They are defined as for sequences, except that the specific data and context formulae are defined by*

specific data formulae	$f(P_1, \ldots, P_n)$,	$f : n \in \Sigma$
specific context formulae	$f(P_1, \ldots, K, \ldots, P_n)$,	$f : n \in \Sigma$

In an analogous fashion to the tree case, we can derive the formula $f(P_1, \ldots, P_n) \triangleq f(P_1, \ldots, \_, \ldots, P_n)(P_i)$, with its corresponding right adjoints.

Definition 11 (Satisfaction for Terms). *The satisfaction relations for terms are defined analogously to the tree case with*

$$\sigma, d \vDash_D f(P_1, \ldots, P_n) \text{ iff } d = f(d_1, \ldots, d_n) \text{ and } d_i \vDash_D P_i$$
$$\sigma, c \vDash_D f(P_1, \ldots, K, \ldots, P_n) \text{ iff}$$
$$\exists c' \in \mathcal{C}, d_1, ..., d_n \in \mathcal{D}.\ c = f(d_1, \ldots, c', \ldots, d_n) \wedge \sigma, d_i \vDash_D P_i \wedge \sigma, c' \vDash_K K$$

Our Context Logic reasoning about terms is a trivial adaptation of our reasoning about other examples studied in this section, since terms decompose nicely into context/subterm pairs. Unlike the other examples, it is not possible to define a derived join formula $P_1 \cdot P_2$, due to the fixed arity structure of the function symbols. There is also no formula 0 specifying a zero data element, since such an element does not exist with terms. Our simple reasoning about terms demonstrates the full generality of our Context Logic approach, compared with Bunched Logic and Ambient Logic whose fundamental structure depends on the join connective and zero formula.

3 General Theory of Context Logic

We develop the underlying general theory of Context Logic, giving the proof theory, the models and the forcing semantics. We do this in two stages: first we look at the basic theory of Context logic, restricting our attention to the structural and additive formulae; then we look at Context Logic with an additional zero formula 0, since many of our examples include a zero element and it yields interesting logical structure.

3.1 Context Logic

We study the basic theory of Context Logic, focusing on the structural and additive formulae.

Definition 12 (Formulae). *The basic Context Logic consists of a set of* data formulae *and a set of* context formulae*, described by the grammars:*

data formulae	$P ::= K(P) \mid K \lhd P$	*structural formulae*
	$P \Rightarrow P \mid \mathit{false}$	*additive formulae*
context formulae	$K ::= I \mid P \rhd P$	*structural formulae*
	$K \Rightarrow K \mid \mathit{False}$	*additive formulae*

The models for Context Logic generalise the specific examples we gave in section 2. Each of the specific examples consisted of a data set $\mathcal{D}$, a context set $\mathcal{C}$ containing the empty context $\_$, and an application function applying contexts to data. In the general models, we have abstract sets $\mathcal{D}$ and $\mathcal{C}$, which we still call the data and context sets respectively, a partial application function, and a set I of distinguished elements of $\mathcal{C}$ which play the role of the empty context.

Definition 13 (Models). *A* model $\mathcal{M}$ *for Context Logic is a tuple* $(\mathcal{D}, \mathcal{C}, ap, \mathbf{I})$ *such that*

1. $\mathcal{D}$ *and* $\mathcal{C}$ *are sets*
2. $ap : \mathcal{C} \times \mathcal{D} \rightharpoonup \mathcal{D}$ *is a partial function, called the* application function
3. $\mathbf{I} \subseteq \mathcal{C}$ *acts as a left identity to* ap*: that is,*
   - $\forall d \in \mathcal{D}, \exists i \in \mathbf{I}.\ ap(i, d)$ *defined;*
   - $\forall d \in \mathcal{D}, \forall i \in \mathbf{I}.\ ap(i, d)$ *defined implies* $ap(i, d) = d$.

The forcing semantics for an arbitrary model is just an abstract version of the forcing semantics for sequences (definition 2).

Definition 14 (Satisfaction for Models). *Given a model* $\mathcal{M}$*, we define two satisfaction relations* $\mathcal{M}, d \vDash_D P$ *and* $\mathcal{M}, c \vDash_K K$ *by induction on the structure of data and context formulae respectively. The cases for the additive connectives are standard. The cases for the structural connectives are*

$$\mathcal{M}, d \vDash_D K(P) \text{ iff } \exists c \in \mathcal{C}, d' \in \mathcal{D}.\ (ap(c, d') = d \wedge \mathcal{M}, c \vDash_K K \wedge \mathcal{M}, d' \vDash_D P)$$

$$\mathcal{M}, d \vDash_D K \lhd P \text{ iff } \forall c \in \mathcal{C}.(\mathcal{M}, c \vDash_K K \wedge\ ap(c, d) \text{ defined } \Rightarrow \mathcal{M}, ap(c, d) \vDash_D P)$$

$$\mathcal{M}, c \vDash_K I \text{ iff } c \in \mathbf{I}$$

$$\mathcal{M}, c \vDash_K P_1 \rhd P_2 \text{ iff } \forall d \in \mathcal{D}.(\mathcal{M}, d \vDash_D P_1 \wedge\ ap(c, d) \text{ defined } \Rightarrow \mathcal{M}, ap(c, d) \vDash_D P_2)$$

In the application case, we assume that $ap(c, d') = d$ means $ap(c, d')$ is defined and equals d. We use the same derived formulae as those given in section 2.

Example 15. Our examples of structured data—sequences, multisets, heaps, trees, terms—are models of Context Logic. Other models include

- $\mathit{Fun}_{\mathcal{D}} = (\mathcal{D}, \mathcal{D} \rightharpoonup \mathcal{D}, ap, \{i\})$ where $\mathcal{D}$ is an arbitrary set, $\mathcal{D} \rightharpoonup \mathcal{D}$ denotes the set of partial functions from $\mathcal{D}$ to $\mathcal{D}$, ap is standard function application, and i is the identity function;

– $Mon_{\mathcal{D}} = (\mathcal{D}, \mathcal{D}, ap, \{0\})$ where $\mathcal{D}$ is a monoid consisting of monoidal operator $\_ \cdot \_ : \mathcal{D} \times \mathcal{D} \to \mathcal{D}$ with unit 0, and application is defined by

$$ap(d_1, d_2) \triangleq d_1 \cdot d_2;$$

a specific example is when $\mathcal{D} = \{0, a\}$ with $a \cdot a = a$;

– two trivial models: $Triv_0 = (\emptyset, \emptyset, \emptyset : \emptyset \to \emptyset, \emptyset)$ and $Triv_1 = (\mathcal{D}, \mathcal{C}, ap, \mathcal{C})$ with $ap(c, d) = d$ for all $c \in \mathcal{C}$, $d \in \mathcal{D}$;

– given two models $\mathcal{M}_1 = (\mathcal{D}_1, \mathcal{C}_1, ap_1, I_1)$ and $\mathcal{M}_2 = (\mathcal{D}_2, \mathcal{C}_2, ap_2, I_2)$ such that $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{C}_1, \mathcal{C}_2$ are disjoint, then define model $\mathcal{M}_1 + \mathcal{M}_2 = (\mathcal{D}_1 \cup \mathcal{D}_2, \mathcal{C}_1 \cup \mathcal{C}_2, ap, I_1 \cup I_2)$ with

$$\begin{array}{rll} ap(c, d) = & ap_1(c, d), & d \in \mathcal{D}_1,\ c \in \mathcal{C}_1 \\ & ap_2(c, d), & d \in \mathcal{D}_2, c \in \mathcal{C}_2 \\ & \text{undefined}, & \text{otherwise} \end{array}$$

We give a simple Hilbert-style proof theory, in which the axioms and rules for the structural operators just state that $K \lhd P_2$ and $P_1 \rhd P_2$ are right adjoints of $K(P_1)$, and I is the identity of application. Calcagno and Yang have shown that this proof theory is sound and complete with respect to the satisfaction relation of our models [YC05].

Definition 16 (Proof Theory). *The Hilbert-style proof theory for Context Logic consists of the following axioms and rules* [3]*:*

the axioms and rules for the additive formulae (using the derived connectives $\wedge$ and $\neg$ to simplify the presentation):

$$\begin{array}{cc}
P \vdash_D P & \mathit{false} \vdash_D P \\[1ex]
\dfrac{P \vdash_D P_1 \qquad P \vdash_D P_2}{P \vdash_D P_1 \wedge P_2} & \dfrac{P \vdash_D P_1 \wedge P_2}{P \vdash_D P_i} \\[2ex]
\dfrac{P_1 \wedge P_2 \vdash_D P_3}{P_1 \vdash_D P_2 \multimap P_3} & \dfrac{P \vdash_D P_1 \multimap P_2 \qquad P \vdash_D P_1}{P \vdash_D P_2} \\[2ex]
\neg\neg P \vdash_D P & \neg\neg K \vdash_K K \\[1ex]
K \vdash_K K & \mathit{False} \vdash_K K \\[1ex]
\dfrac{K \vdash_K K_1 \qquad K \vdash_K K_2}{K \vdash_K K_1 \wedge K_2} & \dfrac{K \vdash_K K_1 \wedge K_2}{K \vdash_K K_i} \\[2ex]
\dfrac{K_1 \wedge K_2 \vdash_K K_3}{K_1 \vdash_K K_2 \multimap K_3} & \dfrac{K \vdash_K K_1 \multimap K_2 \qquad K \vdash_K K_1}{K \vdash_K K_2}
\end{array}$$

[3] We concentrate on boolean Context Logic for this note.

the axioms and rules for the structural formulae:

$$P \dashv\vdash_D I(P) \qquad \frac{K_1 \vdash_K K_2 \qquad P_1 \vdash_D P_2}{K_1(P_1) \vdash_D K_2(P_2)}$$

$$\frac{K(P_1) \vdash_D P_2}{K \vdash_K P_1 \rhd P_2} \qquad \frac{K \vdash_K P_1 \rhd P_2 \qquad P \vdash_D P_1}{K(P) \vdash_D P_2}$$

$$\frac{K(P_1) \vdash_D P_2}{P_1 \vdash_D K \lhd P_2} \qquad \frac{P_1 \vdash_D K \lhd P_2 \qquad K_1 \vdash_K K}{K_1(P_1) \vdash_D P_2}$$

In order to state the soundness and completeness theorem, we introduce some notation. Given model $\mathcal{M} = (\mathcal{D}, \mathcal{C}, ap, I)$, we define:

$$\mathcal{M}, P \vDash_D P' \triangleq \forall d \in \mathcal{D}.\, \mathcal{M}, d \vDash_D P \Rightarrow \mathcal{M}, d \vDash_D P'$$
$$\mathcal{M}, K \vDash_K K' \triangleq \forall c \in \mathcal{C}.\, \mathcal{M}, c \vDash_K K \Rightarrow \mathcal{M}, c \vDash_K K'$$

Theorem 17 (Soundness and Completeness). *The proof theory is sound and complete with respect to the satisfaction relations of the models: that is,*

$$P \vdash_D P' \Leftrightarrow (\mathcal{M}, P \vDash_D P' \textit{ for all models } \mathcal{M})$$
$$K \vdash_K K' \Leftrightarrow (\mathcal{M}, K \vDash_K K' \textit{ for all models } \mathcal{M})$$

Proof. Soundness follows by easy induction on the derivation of $K \vdash_K K'$ and $P \vdash_D P'$ respectively. Completeness is proved by Calcagno and Yang [YC05] in two steps. First, they show completeness for models where the application is a *relation*. The proof is an adaptation of a standard technique used in modal logic: the construction of a canonical model built from maximal consistent sets of formulae. Second, they show that each relational model is bisimilar to a functional model, and satisfies the same formulae. This implies completeness for the functional models.

A key impact of the soundness and completeness result is that, in order to demonstrate that a property does not hold, it is enough to find a counter-example in a specific model. Consider the following general properties of Context Logic:

$$\vdash_D (K_1 \wedge K_2)(P) \Rightarrow K_1(P) \wedge K_2(P)$$
$$\vdash_D (P_1 \rhd P_2)(P_1) \Rightarrow P_2 \wedge \diamond P_1$$
$$\vdash_D (P_2 \wedge \diamond P_1) \Rightarrow (P_1 \blacktriangleright P_2)(P_1)$$

The proofs of these properties are left as an exercise. The reverse implications do not hold. For the first case, let $K_1 \triangleq \neg I$ and $K_2 \triangleq I$ and $P \triangleq$ true. The reverse implication is not satisfied by most models and data elements. For example, consider the multiset model $Mult$ with element a. Then $Mult, a \vDash I(\text{true})$ since $a = (\_)(a)$ and $Mult, a \vDash (\neg I)(\text{true})$ since $a = (\_ \cdot a)(0)$, but clearly $Mult, a \nvDash (\neg I \wedge I)(\text{true})$ since no context satisfies $\neg I \wedge I$. Notice however that this reverse implication does hold for the models $Triv_i$, since $(\neg I)(\text{true})$ cannot be satisfied by a data element in either model. The other cases are left as an exercise. It is possible to show that both reverse implications are false by appealing to the same (very simple) instances of P_1 and P_2.

An interesting question is under what conditions the reverse implications are satisfied. This is not just a theoretical game. Such conditions are used in the proofs of the existence of weakest preconditions for Hoare triples based on Context Logic (and Bunched Logic). We consider two definitions: a data formula P is *exact* if at most one data element (up to equality) satisfies it; a data formula P is *precise* if, for an arbitrary data element, it specifies unique subdata.

Definition 18. *Define a data formula P to be exact for model $\mathcal{M}$ iff $\forall d_1, d_2$.*

$$\mathcal{M}, d_1 \vDash P \textit{ and } \mathcal{M}, d_2 \vDash P \textit{ implies } d_1 = d_2.$$

Define a data formula P to be precise for model $\mathcal{M}$ iff $\forall d_1, d_2, c_1, c_2$.

$$\mathit{ap}\,(c_1, d_1) = \mathit{ap}\,(c_2, d_2) \textit{ and } \mathcal{M}, d_1 \vDash P \wedge \mathcal{M}, d_2 \vDash P \textit{ implies } (c_1 = c_2 \wedge d_1 = d_2)$$

We point out some examples of exactness and preciseness. In the sequence model, the formula a is exact, but not precise since the sequence $a{\cdot}a = (a{\cdot}\_)(a) = (\_{\cdot}a)(a)$. In the multiset model, formula a is exact and precise. In the heap model, $a \mapsto b$ is exact and precise, whereas $a \mapsto \_$ is precise but not exact. In the tree model, $a[0]$ is exact and not precise, and $a[\text{true}]$ is neither exact or precise. In contrast, the tree model with unique identifiers is like the heap model, with $a[0]$ exact and precise, and $a[\text{true}]$ is precise but not exact. The justification for these facts are left as a simple exercise for the reader. Using exactness and preciseness, we can prove the reverse implications of the general properties given above:

– given an arbitrary model $\mathcal{M}$, if P is precise then

$$\mathcal{M}, K_1(P) \wedge K_2(P) \vDash_D (K_1 \wedge K_2)(P)$$

– given an arbitrary model $\mathcal{M}$, if P_1 is exact then

$$\mathcal{M}, (P_1 \blacktriangleright P_2)(P_1) \vDash_D P_2 \wedge \diamond P_1 \;\text{ and }\; \mathcal{M}, P_2 \wedge \diamond P_1 \vDash_D (P_1 \triangleright P_2)(P_1)$$

The proof of these results is simple.

3.2 Context Logic with Zero

Many of the examples in section 2.1 include a specific data formula 0, corresponding for example to the empty heap or tree. In this section, we include a zero formula as a structural data formula, viewing it as a fundamental part of the logical structure.

Definition 19 (Formulae with Zero). *The Context Logic with Zero consists of data and context formulae as in Definition 12, with the additional structural data formula 0.*

Definition 20 (Model with Zero). *A model $\mathcal{M}$ for Context Logic with Zero is a tuple $(\mathcal{D}, \mathcal{C}, ap, \mathbf{I}, \mathbf{0})$ where*

1. *$(\mathcal{D}, \mathcal{C}, ap, \mathbf{I})$ is a model of Context Logic;*
2. *$\mathbf{0} \subseteq \mathcal{D}$;*
3. *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 from $\mathcal{C}$ to $\mathcal{D}$;*
4. *$\forall c \in \mathcal{C}, \forall o \in \mathbf{0}.\ p(c) = o \Rightarrow c \in \mathbf{I}$.*

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, which means that every data element has a zero element as a sub-element (such a projection function exists for all the examples in section 2.1, except terms). Notice that we can define an embedding relation $e : \mathcal{D} \times \mathcal{C}$ with $e(d, c)$ if and only if $p(c) = d$. We write $e(d)$ for $\{c \in \mathcal{C} : e(d, c)\}$. The relation e is not necessarily a function: for example, in the tree model $e(b[0]) = \{b[\_],\ \_ \cdot b[0],\ b[0] \cdot \_\}$. The pair (e, p) is an embedding-projection pair: that is $\forall d \in \mathcal{D}. \{p(c) \mid c \in e(d)\} = \{d\}$. Condition 4 places a strong connection between $\mathbf{I}$ and $\mathbf{0}$: from definition 13, we have $p(i) \subseteq \mathbf{0}$ for all $i \in \mathbf{I}$; we also have $e(o) \subseteq \mathbf{I}$ for all $o \in \mathbf{0}$.

Definition 21 (Satisfaction for Models with Zero). *Given a model with zero $\mathcal{M}$, the two satisfaction relations $\mathcal{M}, d \vDash_D P$ and $\mathcal{M}, c \vDash_K K$ are defined by induction on the structure of data and context formulae respectively as in definition 14, with the additional case*

$$\mathcal{M}, d \vDash_D 0 \ \ \textit{iff} \ \ d \in \mathbf{0}$$

Sequences, multisets, heaps and trees are all models of Context Logic with Zero. The term model with signature set $\{f : 1, g_1 : 0, g_2 : 0\}$ is not: if $\mathbf{0}$ does not contain g_2 (and similarly for g_1), then g_2 cannot be in the image of p contradicting surjectivity; however g_1, 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, if the signature set is $\{f : 1, g : 0\}$, then the term model with zero set $\{g : 0\}$ is a model for Context Logic with zero.

Now consider the models of Context Logic introduced in example 15.

- The model $Fun_{\mathcal{D}} = (\mathcal{D}, \mathcal{D} \rightharpoonup \mathcal{D}, ap, \{i\})$ cannot be extended to include a zero set in general. Assume it can, with the zero set denoted by 0. If 0 contains more than one element, say d_1 and d_2, then $p(i) = \{d_1, d_2\}$ which contradicts condition 3. If 0 contains just one element d, then the constant function f_d which always returns the answer d is a counterexample to condition 4.
- The tuple $Mon_{\mathcal{D}} = (\mathcal{D}, \mathcal{D}, ap, \{0\}, \{0\})$ is a model of Context Logic with Zero.
- $Triv_0 = (\emptyset, \emptyset, \emptyset : \emptyset \to \emptyset, \emptyset, \emptyset)$ is a trivial model of Context Logic with zero. It is not possible to extend $Triv_1 = (\mathcal{D}, \mathcal{C}, ap, \mathcal{C})$ with a zero set in general. For example, consider $D = \{d_1, d_2\}$. Since p must be surjective, the zero set much be D, but this implies that p is not well-defined.

– Given two models $\mathcal{M}_1 = (\mathcal{D}_1, \mathcal{C}_1, ap_1, \mathbf{I}_1, \mathbf{0}_1)$ and $\mathcal{M}_2 = (\mathcal{D}_2, \mathcal{C}_2, ap_2, \mathbf{I}_2, \mathbf{0}_2)$ such that $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{C}_1, \mathcal{C}_2$ are disjoint, then define model $\mathcal{M}_1 + \mathcal{M}_2 = (\mathcal{D}_1 \cup \mathcal{D}_2, \mathcal{C}_1 \cup \mathcal{C}_2, ap, I_1 \cup I_2, \mathbf{0}_1 \cup \mathbf{0}_2)$ with ap defined as in example 15.

Definition 22 (Proof Theory with Zero). *The proof theory for Context Logic with Zero extends definition 16 with the following zero axioms (we use the derived* $\blacktriangleright$ *to provide a more intuitive presentation of the axioms):*

$$P \vdash_D (0 \blacktriangleright P)(0) \qquad 0 \blacktriangleright P \dashv\vdash_K 0 \triangleright P$$

$$0 \blacktriangleright 0 \vdash_K I$$

The first axiom states that every data element has a zero element as subdata. It implies that $\diamond 0$ is always true, and corresponds to the projection in the models being surjective. In the second axiom, the implication from left to right states that all the zero elements behave in a similar fashion when put in a context. It corresponds to the projection being a function. The implication from right to left implies the existence of a zero element (except in a trivial case such as $Triv_0$). It corresponds to the property that, if a context satisfies $0 \triangleright P$, then it cannot be for the vacuous reason that there are no zero elements, and implies that the projection function is total. Recall the general property $\vdash_D (P_1 \triangleright P_2)(P_1) \Rightarrow P_2 \wedge \diamond P_1$, and the fact that the reverse implication holds when P_1 is exact. Although zero formula is not exact for all models (for example, we can combine two disjoint models), the zero axioms imply $(0 \triangleright P)(0) \dashv\vdash_D P$. From the third axiom, we can derive $I \dashv\vdash_D 0 \blacktriangleright 0 \dashv\vdash_D 0 \triangleright 0$. It corresponds to the condition 4 of definition 20.

There is another way to think about these axioms. Consider the derived formulae: $K^p \triangleq K(0)$ and $P^e \triangleq 0 \triangleright P$. The following entailments are derivable:

$$\begin{aligned} P^{ep} &\vdash_D P \\ \neg(P^e) &\dashv\vdash_D (\neg P)^e \\ 0^e &\vdash_D I \end{aligned}$$

These entailments are equivalent to the axioms given in definition 22. The first entailment implies that the pair $((\_)^e, (\_)^p)$ in a model is an embedding-projection pair; the second that negation distributes over $(\_)^e$, or alternatively that $(\_)^e$ has a right adjoint given by $\neg((\neg\_)^p)$; the third that $(\_)^e$ lifts 0 to I.

Theorem 23 (Soundness and Completeness with Zero). *The proof theory for Context Logic with Zero is sound and complete with respect to the satisfaction relations of the models.*

Proof. The proof by Calcagno and Yang is analogous to the proof for Context Logic without a zero formula. Note that, when application is a relation (used in the first part of the proof), the projection remains a total surjective function due to the zero axioms of definition 22.

Recall that we introduced the derived data formula

$$P_1 * P_2 \triangleq (0 \triangleright P_1)(P_2)$$

for sequences in section 2.1, and showed that $*$ is neither associative nor commutative. Using the embedding/projection notation, so that $P_1 * P_2$ can be viewed as $P_1^e(P_2)$, there is a natural way to define the two right adjoints of $*$ in Context logic with zero:

$$\begin{aligned} P_1 \mathrel{*\!\!-} P &\triangleq P_1^e \lhd P \\ P_2 \mathrel{-\!\!*} P &\triangleq \neg((\neg(P_2 \triangleright P))^p) \end{aligned}$$

The first adjoint 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. The second adjoint is more complicated, although observe the similarities with the right adjoint of $(\_)^e$ given above. It states that, whenever data satisfying P_2 replaces a zero element of the given data, then the resulting data satisfies P. For example, for sequences we have the derived formulae:

$a \mathrel{-\!\!*} P$	inserting a anywhere in the given sequence makes P hold
$a \mathrel{*\!\!-} P$	adding a to either side of the given sequence makes P hold

Lemma 24 (Adjoints). *$P_1 \mathrel{*\!\!-} \_$ and $P_2 \mathrel{-\!\!*} \_$ are the right adjoints of $P_1 * \_$ and $\_ * P_2$, respectively: that is.*

$$\begin{aligned} P_1 * P_2 \vdash_D P &\Leftrightarrow P_2 \vdash_D P_1 \mathrel{*\!\!-} P \\ P_1 * P_2 \vdash_D P &\Leftrightarrow P_1 \vdash_D P_2 \mathrel{-\!\!*} P \end{aligned}$$

Proof. The first result is immediate since $K \lhd \_$ is right adjoint of $K(\_)$. The second follows from the adjunction properties of $\triangleright$ and $(\_)^e$, together with excluded middle and the second axiom in definition 22.

Lemma 25 (Unit). $P * 0 \Leftrightarrow P \Leftrightarrow 0 * P$

Proof. The first equivalence follows immediately from the first axiom in definition 22. The second equivalence follows from the third axiom and I being the left identity of application.

This derived connective $*$ also has a derived counterpart in the underlying models for zero. Define the relation $* \subseteq (\mathcal{D} \times \mathcal{D}) \times \mathcal{D}$ by $d_1 * d_2 \triangleq \{ap\,(c, d_2) : \exists c.\ g(c) = d_1\}$. For example, in the tree model with a commutative join operator $\_ \cdot \_$, we have $b[a[0]] * c[0] = \{b[a[c[0]]], b[a[0] \cdot c[0]], b[a[0]] \cdot c[0]\}$. The connection between this $*$-relation on data and the $*$-connective on data formulae is given by

$$d \vDash_D P_1 * P_2 \text{ iff } \exists d_1, d_2 \in \mathcal{D}.\ d = d_1 * d_2 \wedge d_1 \vDash_D P_1 \wedge d_2 \vDash_D P_2.$$

When the projection function p in the model with zero is a bijection, then the $*$-operator on data is a function. In fact, we can prove a stronger result.

Given a model $\mathcal{M} = (\mathcal{D}, \mathcal{C}, ap, \mathbf{I})$, we define a relation $\sim \subseteq \mathcal{C} \times \mathcal{C}$ by $c_1 \sim c_2$ iff $\forall d \in \mathcal{D}.\, ap\,(c_1, d) = ap\,(c_2, d)$. We say that $\mathcal{M}$ is *extensional* iff $\forall c_1, c_2 \in \mathcal{C}.\ c_1 \sim c_2 \Rightarrow c_1 = c_2$. All the models in section 2.1 are extensional; the model $Triv_1$ is not. Given an extensional model with zero, then the projection function p is an isomorphism if and only if $*$ is a function.

References

[CGZ05] C. Calcagno, P. Gardner, and U. Zarfaty. Context logic and tree update. In *POPL*, 2005.

[Fio05] Marcelo Fiore. Mathematical models of computational and combinatorial structures. In *FOSSACS*, volume LNCS 3441, pages 25–45. Springer, 2005.

[IO01] S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *POPL*, 2001.

[OP99] P. O'Hearn and D. Pym. Logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[Pym02] D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications.* Applied Logic Series. Kluwer Academic Publishers, 2002.

[Rey02] J.C. Reynolds. Separation logic: a logic for shared mutable data structures. Invited Paper, LICS'02, 2002.

[YC05] Hongseok Yang and Cristiano Calcagno. Completeness results for Context Logic and BI. In preparation, 2005.

[YO02] H. Yang and P. O'Hearn. A semantic basis for local reasoning. FOSSACS, 2002.