Iteration Algebras for UnQL Graphs and Completeness for Bisimulation

Makoto Hamana
Department of Computer Science, Gunma University, Japan
hamana@cs.gunma-u.ac.jp

This paper shows an application of Bloom and Ésik's iteration algebras to model graph data in a graph database query language. About twenty years ago, Buneman et al. developed a graph database query language UnQL on the top of a functional meta-language UnCAL for describing and manipulating graphs. Recently, the functional programming community has shown renewed interest in UnCAL, because it provides an efficient graph transformation language which is useful for various applications, such as bidirectional computation. However, no mathematical semantics of UnQL/UnCAL graphs has been developed. In this paper, we give an equational axiomatisation and algebraic semantics of UnCAL graphs. The main result of this paper is to prove that completeness of our equational axioms for UnCAL for the original bisimulation of UnCAL graphs via iteration algebras. Another benefit of algebraic semantics is a clean characterisation of structural recursion on graphs using free iteration algebra.

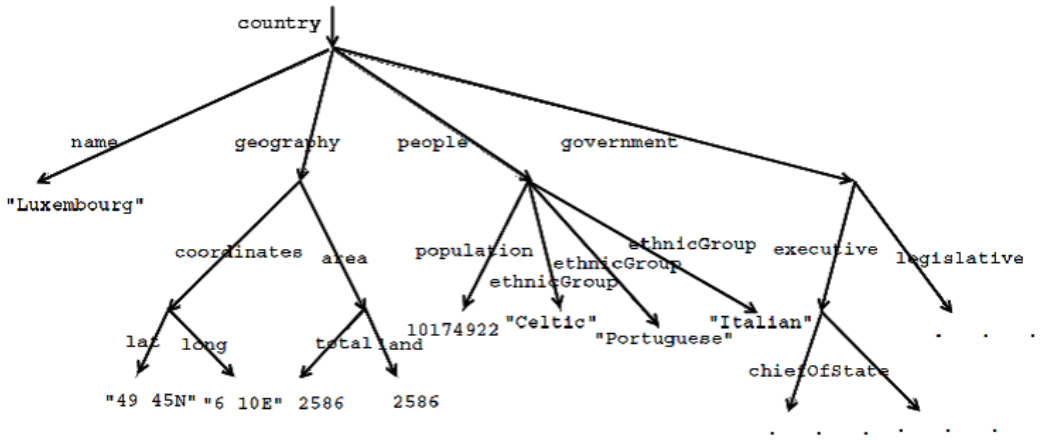
1 Introduction

Graph database is used as a back-end of various web and net services, and therefore it is one of the important software systems in the Internet society. About twenty years ago, Buneman et al. [6, 7, 8] developed a graph database query language UnQL (Unstructured data Query Language) on top of a functional meta-language **UnCAL** (Unstructured Calculus) for describing and manipulating graph data. The term "unstructured" is used to refer to unstructured or semi-structured data, i.e., data having no assumed format in a database (in contrast to relational database). Recently, the functional programming community found a new application area of UnCAL in so-called bidirectional transformations on graph data, because it provides an efficient graph transformation language. The theory and practice of UnCAL have been extended and refined in various directions (e.g. [18, 19, 17, 1]), which has increased the importance of UnCAL.

In this paper, we give a more conceptual understanding of UnCAL using semantics of type theory and fixed points. We give an equational axiomatisation and algebraic semantics of UnCAL graphs. The main result of this paper is to prove completeness of our equational axioms for UnCAL for the original bisimulation of UnCAL graphs via iteration algebras. Another benefit of algebraic semantics is a clean characterisation of the computation mechanism of UnCAL called "structural recursion on graphs" using free iteration algebra.

UnCAL Overview. We begin by introducing UnCAL. UnCAL deals with graphs in a graph database. Hence, it is better to start with viewing how concrete semi-structured data is processed in UnCAL. Consider the semi-structured data `sd` below which is taken from [8].

Submitted to:
FICS 2015



It contains information about country, e.g. geography, people, government, etc. It is depicted as a tree above, in which edges and leaves are labelled. Using UnCAL's term language for describing graphs (and trees), this is defined by `sd` shown at right. Then we can define functions in UnCAL to process data. For example, a function that retrieves all ethnic groups in the graph can be defined simply by

```
sd ◃ country:{name:"Luxembourg",
 geography:{coordinates:{long:"49 45N", lat:"6 10E"},
            area:{total:2586, land:2586}},
 people:{population:425017,
         ethnicGroup:"Celtic",
         ethnicGroup:"Portuguese",
         ethnicGroup:"Italian"},
 government:{executive:{chiefOfState:{name:"Jean",..}}}}
```

```
sfun f1(L:T) = if L = ethnicGroup then (result:T) else f1(T)
```

The keyword `sfun` denotes a function definition by *structural recursion on graphs*, which is the computational mechanism of UnCAL. Executing it, we can certainly extract:

```
f1(sd) ⇝ {result:"Celtic", result:"Portuguese", result:"Italian"}
```

The notation $\{\cdots:\cdots,\cdots\}$ is a part of the UnCAL's term language for representing graphs. It consists of markers x, labelled edges $\ell{:}t$, vertical compositions $s \diamond t$, horizontal compositions $\langle s, t\rangle$, other horizontal compositions $s \cup t$ merging roots, forming cycles $\mathsf{cycle}(t)$, constants $\{\}$,$()$, and definitions $(x \triangleleft t)$. These term constructions have underlying graph theoretic meaning shown at th right. Namely, these are officially defined as operations on the ordinary representations of graphs: (vertices set, edges set, leaves, roots)-tuples $(V, E, \{y_1, \ldots y_m\}, \{x_1, \ldots, x_n\})$, but we do not use the graph theoretic definitions of these operations in this paper.

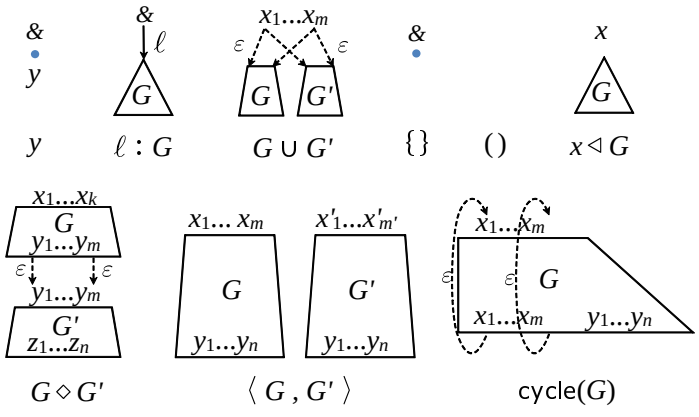


Figure 1: Graph theoretic definitions of constructors [8]
Slightly changed notation. Correspondence between the original and this paper's: $\&y = y$, $@ = \diamond$, $\oplus = \langle -, - \rangle$, $(- := -) = - \triangleleft -$.

UnCAL deals with graphs *modulo bisimulation* (i.e. not only modulo graph isomorphism). An UnCAL graph is directed and have (possibly multiple) root(s) written & (or multiple $x_1 \cdots x_n$) and leaves (written $y_1 \cdots y_m$), and with the roots and leaves drawn pictorially at the top and bottom, respectively. The symbols $x, y_1, y_2, \&$ in the figures and terms are called markers, which are the names of nodes in a graph and are used for references for cycles. Also, they are used as port names to connect two graphs. A dotted line labelled ε is called an ε-edge, which is a "virtual" edge connecting two nodes directly. This is achieved by identifying graphs by *extended bisimulation*, which ignores ε-edges suitably in UnCAL. The UnCAL graph G shown at the left is an example. This is extended bisimilar to a graph that reduces all ε-edges. Using UnCAL's language, G is represented as the following term $\mathbf{t}_G$

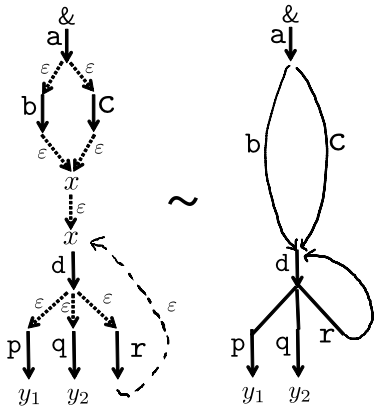


Figure 2: Graph G and bisimilar one

$$\mathbf{t}_G \;=\; \mathtt{a:}(\{\mathtt{b:}\,x\} \cup \{\mathtt{c:}\,x\}) \diamond \mathsf{cycle}(x \triangleleft \mathtt{d:}(\{\mathtt{p:}\,y_1\} \cup \{\mathtt{q:}\,y_2\} \cup \{\mathtt{r:}\,x\})\,).$$

UnCAL's structural recursive function works also on cycle. For example, define another function

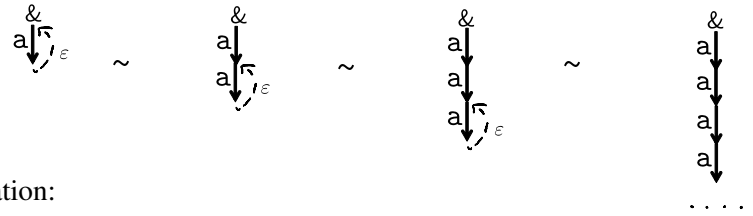
```
sfun f2(L:T) = a:f2(T)
```

that replaces every edge with a. As expected,

$$\mathtt{f2}(\mathbf{t}_G) \;\rightsquigarrow\; \mathtt{a:}(\{\mathtt{a:}\,x\} \cup \{\mathtt{a:}\,x\}) \diamond \mathsf{cycle}(x \triangleleft \mathtt{a:}(\{\mathtt{a:}\,y_1\} \cup \{\mathtt{a:}\,y_2\} \cup \{\mathtt{a:}\,x\})\,)$$

where all labels are changed to a.

Another characteristic role of bisimulation is that it identifies expansion of cycles. For example, a term $\mathsf{cycle}(\& \triangleleft \mathtt{a:}\&)$ corresponds to the graph shown below at the leftmost. It is bisimilar to the right ones, especially the infinitely expanded graph shown at the rightmost, which has no cycle.



These are in term notation:

$$\mathsf{cycle}(\& \triangleleft \mathtt{a:}\&) \;\sim\; \mathtt{a:}\mathsf{cycle}(\& \triangleleft \mathtt{a:}\&) \;\sim\; \mathtt{a:a:}\mathsf{cycle}(\& \triangleleft \mathtt{a:}\&)$$

Problems. There have been no algebraic laws that establish the above expansion of cycle. Namely, these are merely bisimilar, and not a consequence of any algebraic law. But obviously, we expect that it should be a consequence of the algebraic law of *fixed point property* of cycle.

In the original and subsequent formulation of UnCAL [8, 17, 18, 1], there are complications of this kind. The relationship between terms and graphs in UnCAL is not a one-to-one correspondence. No term notation exits for ε-edges and infinite graphs (generated by the cycle construct), thus the rightmost infinite graphs of the above expansion cannot be expressed in syntax. But such an infinite graph is allowed as a possible graph in the original formulation of UnCAL. Consequently, instead of terms, one must use graphs and graph theoretic reasoning with care of bisimulation to reason about UnCAL. Therefore, a property in UnCAL could not be established only using induction on terms. That fact sometime makes some proofs about UnCAL quite complicated.

Because UnCAL graphs are identified by bisimulation, it is necessary to use a procedure or algorithm to check the bisimilarity as in the cycle example above. Listing some typical valid equations for the bisimulation can be a shortcut [8, 19], but it was only sound and *not complete* for bisimulation.

Hence, we give an algebraic and type-theoretic formulation of UnCAL by giving equational axioms of UnCAL graphs. In this paper, we prove completeness of our proposed axioms using iteration algebra [4]. Thus we have a *complete* syntactic axiomatisations of the equality on UnQL/UnCAL graphs, as a set of axioms capturing the original bisimulation, without touching graphs, ε-edges, and the notion of bisimulation explicitly. We prove it by connecting it with the algebraic axiomatisations of bisimulation [3, 12].

How to model UnCAL and structural recursion. The first idea to understand UnCAL is to interpret it as a categorical structure. We can regard edges as *morphisms* (of the opposite directions), the vertical composition $\diamond$ as the *composition of arrows*, and cycle as a *fixpoint operator* in a suitable category. Thus the target categorical structure should have a notion of fixpoint, which has been studied in iteration theories of Bloom and Ésik [3]. In particular, iteration categories [10] are suitable, which are traced cartesian categories [20] (monoidal version is used in Hasegawa's modelling of cyclic sharing theories [16, 15]) additionally satisfying the commutative identities axiom [3] (see also [25] Section 2 for a useful overview around this).

We also need to model UnCAL's computational mechanism: "structural recursion on graphs". The general form of the definition of structural recursive function is

$$\texttt{sfun}\ F(\ell : t)\ =\ e \qquad (\bigstar)$$

where e can involve $F(t)$. The graph algorithm in [8] provide a transformation of graphs that produces some computed graphs using the definition ($\bigstar$). It becomes a function F satisfying the equations ([8] Prop. 3):

$$\begin{array}{lll} F(\,y_i\,) = y_i & F(\,(x \triangleleft t)\,) = (x \triangleleft F(t)) & F(\quad \ell\!:\!t \quad) = e \\ F(\,()\,) = () & F(\quad s \cup t \quad) = F(s) \cup F(t) & F(\quad s \diamond t \quad) = F(s) \diamond F(t) \quad \cdots(\bowtie) \\ F(\,\{\}\,) = \{\} & F(\ \langle s, t\rangle\ ) = \langle F(s), F(t)\rangle & F(\ \mathsf{cycle}(t)\ ) = \mathsf{cycle}(F(t)) \quad \cdots(\bowtie) \end{array} \qquad (1)$$

when e does not depend[1] on t. This is understandable naturally as the example `f2` recurses structurally the term $\mathbf{t}_G$. Combining the above categorical viewpoint, F can be understood as a functor that preserves cycle and products (thus a traced cartesian functor). A categorical semantics of UnCAL can be given along this idea, which will be reported elsewhere. This idea works for simple cases of structural recursion such as `f2`.

However, there is a critical mismatch between the above categorical view and UnCAL's structural recursion of more involved cases. Buneman et al. mentioned a condition that the above nine equations hold only when *e does not* depend on t in ($\bigstar$). Two equations marked ($\bowtie$) do not hold in general if *e does* depend on t (other seven equations do hold). Crucially, `f1` is already this case, where `T` appears as not of the form `f1(T)`. The following another example shows why ($\bowtie$) do not hold: the structural recursive function `aa?` tests whether the argument contains "`a:a:`".

```
sfun a?(L:T)  = if L=a then true:{} else {}
sfun aa?(L:T) = if L=a then a?(T) else aa?(T)
```

The definition of `aa?` *does* depend on `T` at the "then"-clause. Then we have the inequalities:

[1] Here "e depends on t" means that e contains t other than the form $F(t)$.

aa?((a:&)⋄(a:{})) = aa?(a:a:{}) = true:{} ≠ {} = {}⋄{} = aa?(a:&) ⋄ aa?(a:{})
aa?(cycle(a:&)) = aa?(a:a:cycle(a:&)) = true:{} ≠ {} = cycle({}) = cycle(aa?(a:&))

This means that F does not preserve cycle in general, and even *is not functorial*, thus the categorical view seems not helpful to understand this pattern of recursion.

In this paper, we consider *algebraic semantics* of UnCAL using the notion of iteration Σ-algebras [4, 12] in §3. It solve the problem mentioned above, i.e. we derive the structural recursion even when the case that e depends on t within the algebraic semantics.

Organisation. This paper is organised as follows. We first give a framework of equational theory for UnCAL graphs by reformulating UnCAL graph data in a type theoretic manner in Section 2. We then give algebraic semantics of UnCAL using iteration Σ-algebras in Section 3. We prove completeness of our axioms for UnCAL graphs for bisimulation in Section 3.3. We further derive structural recursion on UnCAL graphs in Section 3.5. Finally, in Section 3.6. we show several examples how structural recursive functions on graphs are modeled.

2 UnCAL and its Equational Theory

We give a framework of equational theory for UnCAL graphs. We reformulate UnCAL graph data in a type theoretic manner. We do not employ the graph theoretic and operational concepts (such as ε-edges, bisimulation, and the graph theoretic definitions in Fig. 1). Instead, we give an algebraic axiomatisation of UnCAL graphs following the tradition of categorical type theory [9]. The syntax in this paper is slightly modified from the original presentation [8] to reflect the categorical idea, which may be more readable for the reader familiar with categorical type theory.

2.1 Syntax

Markers and contexts. We assume an infinite set of symbols called *markers*, denoted by typically $x, y, z, \ldots$. One can understand markers as variables in a type theory. The marker denoted by & is called the default marker, which is just a default choice of a marker having no special property. Let L be a set of *labels*. A *label* ℓ is a symbol (e.g. $\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots$ in Fig. 2). A *context*, denoted by $\langle\langle x_1, x_2, \ldots\rangle\rangle$, is a sequence of pairwise distinct markers . We typically use $X, Y, Z, \ldots$ for contexts. We use $\langle\langle\rangle\rangle$ for the empty contexts, X, Y for the concatenation, and $|X|$ for its length. We may use the vector notation $\vec{x}$ for sequence $x_1, \ldots, x_n$. The outermost bracket $\langle\langle\ \rangle\rangle$ of a context may be omitted. We may use the abbreviations for the empty context $0 = \langle\langle\rangle\rangle$. Note that the concatenation may need suitable renaming to satisfy pairwise distinctness of markers.

Raw terms.

$$t \ ::= \ y_Y \ \mid \ \ell{:}t \ \mid \ s \diamond t \ \mid \ \langle s, t\rangle \ \mid \ \mathsf{cycle}^X(t) \ \mid \ \{\}_Y \ \mid \ ()_Y \ \mid \ \curlywedge \ \mid \ (x \triangleleft t)$$

We assume several conventions to simplify the presentation of theory. We often omit subscripts or superscripts such as Y when they are unimportant or inferable. We identify $\langle\langle s, t\rangle, u\rangle$ with $\langle s, \langle t, u\rangle\rangle$; thus we will freely omit parentheses as $\langle t_1, \ldots, t_n\rangle$. A constant $\curlywedge$ express a branch in a tree, and we call the symbol $\curlywedge$ a *man*, because it is similar to the shape of a kanji or Chinese character meaning a man, which is originated from the figure of a man having two legs (and the top is a head).

$$\text{(Nil)}\ \frac{}{Y \vdash \{\}_Y\ :\&} \qquad \text{(Emp)}\ \frac{}{Y \vdash ()_Y\ :\langle\langle\rangle\rangle} \qquad \text{(Man)}\ \frac{}{y_1, y_2 \vdash \curlywedge_{\langle\langle y_1,y_2\rangle\rangle}\ :\&}$$

$$\text{(Com)}\ \frac{Y \vdash s\ :Z \qquad X \vdash t\ :Y}{X \vdash s \diamond t\ :Z} \qquad \text{(Label)}\ \frac{\ell \in L \quad Y \vdash t\ :\&}{Y \vdash \ell{:}t\ :\&} \qquad \text{(Mark)}\ \frac{Y = \langle\langle y_1, \ldots, y_n\rangle\rangle}{Y \vdash y_{iY}\ :\&}$$

$$\text{(Pair)}\ \frac{Y \vdash s\ :X_1 \qquad Y \vdash t\ :X_2}{Y \vdash \langle s, t\rangle\ :X_1, X_2} \qquad \text{(Cyc)}\ \frac{Y, X \vdash t\ :X}{Y \vdash \mathsf{cycle}^X(t)\ :X} \qquad \text{(Def)}\ \frac{Y \vdash t\ :\&}{Y \vdash (x \triangleleft t)\ :x}$$

Figure 3: Typing rules

Abbreviations. We use the following abbreviations.

$$\begin{array}{llllll} \{s\} \cup \{t\} & \triangleq & \curlywedge \diamond \langle s, t\rangle & s \times t & \triangleq & \langle s \diamond \pi_1, t \diamond \pi_2\rangle \\ \pi_1 & \triangleq & x_{\langle\langle x,y\rangle\rangle} & \mathsf{id}_{\langle\langle x\rangle\rangle} & \triangleq & x_{\langle\langle x\rangle\rangle} \\ \pi_2 & \triangleq & y_{\langle\langle x,y\rangle\rangle} & \mathsf{id}_{\langle\langle x_1,\ldots,x_n\rangle\rangle} & \triangleq & x_1\,\langle\langle x_1\rangle\rangle \times \cdots \times x_n\,\langle\langle x_n\rangle\rangle \end{array}$$

$$\Delta_X \triangleq \langle \mathsf{id}_X, \mathsf{id}_X\rangle \qquad \mathsf{c} \triangleq \langle \pi_2, \pi_1\rangle$$

Inheriting the convention of $\langle -, -\rangle$, we also identify $(s \times t) \times u$ with $s \times (t \times u)$, thus we omit parentheses as $t_1 \times \ldots \times t_n$.

2.2 Typed syntax

For contexts X, Y, we inductively define a judgment relation $Y \vdash t\ :X$ of terms by the typing rules in Fig. 3. We call a marker *free* in t when it occurs in t other than the left hand-side of a definition $(x \triangleleft s)$. In a judgment, free markers in t are always taken from Y. Thus Y is a variable context (which we call the *source context*) in ordinary type theory, and X is the roots (which we call the *target context* or *type*). For example, the term $\mathbf{t_G}$ in §1 is well-typed $y_1, y_2 \vdash \mathbf{t_G}\ :\&$, which corresponds a graph in Fig. 2, where the marker &is the name of the root. When t is well-typed by the typing rules, we call t a (well-typed UnCAL) term. We identify t of type & with $(\& \triangleleft t)$.

Definition 2.1 (Substitution) Let $Y = \langle\langle y_1 \cdots, y_k\rangle\rangle$, W be contexts such that $|Y| \leq |W|$ and Y can be embedded into W in an order-preserving manner, and Y' is the subsequence of W deleting all of Y (NB. $|W| = |Y| + |Y'|$, Y' is possibly empty). Suppose $W \vdash t\ :X$, $Z \vdash s_i\ :\langle\langle y_i\rangle\rangle$ $(1 \leq i \leq k)$. Then a substitution $Z, Y' \vdash t\,[\vec{y} \mapsto \vec{s}]\ :X$ is inductively defined as follows.

$$\begin{array}{rcl@{\qquad}rcl} y_i\,[\vec{y} \mapsto \vec{s}] & \triangleq & s_i & (t_1 \diamond t_2)\,[\vec{y} \mapsto \vec{s}] & \triangleq & t_1 \diamond (t_2\,[\vec{y} \mapsto \vec{s}]) \\ x\,[\vec{y} \mapsto \vec{s}] & \triangleq & x\ (\text{if } x \text{ in } Y') & \langle t_1, t_2\rangle\,[\vec{y} \mapsto \vec{s}] & \triangleq & \langle (t_1\,[\vec{y} \mapsto \vec{s}]), (t_2\,[\vec{y} \mapsto \vec{s}])\rangle \\ \{\}_Y\,[\vec{y} \mapsto \vec{s}] & \triangleq & \{\}_{Z+Y'} & \mathsf{cycle}(t)\,[\vec{y} \mapsto \vec{s}] & \triangleq & \mathsf{cycle}(t\,[\vec{y} \mapsto \vec{s}]) \\ ()_Y\,[\vec{y} \mapsto \vec{s}] & \triangleq & ()_{Z+Y'} & (x \triangleleft t)\,[\vec{y} \mapsto \vec{s}] & \triangleq & (x \triangleleft t\,[\vec{y} \mapsto \vec{s}]) \\ (\ell{:}t)\,[\vec{y} \mapsto \vec{s}] & \triangleq & \ell{:}(t\,[\vec{y} \mapsto \vec{s}]) & & & \end{array}$$

$$\curlywedge_{\langle\langle y_1,y_2\rangle\rangle}\,[y_1 \mapsto s_1, y_2 \mapsto s_2] \triangleq \curlywedge_{\langle\langle y_1,y_2\rangle\rangle} \diamond (s_1, s_2)$$

Note that $t\,[\vec{y} \mapsto \vec{s}]$ denotes a meta-level substitution operation, not an explicit substitution.

2.3 Equational theory

For terms $Y \vdash s\ :X$ and $Y \vdash t\ :X$, an *(UnCAL) equation* is of the form $Y \vdash s = t\ :X$. Hereafter, for simplicity, we often omit the source X and target Y contexts, and simply write $s = t$ for an equation, but even such an abbreviated form, we assume that it has implicitly suitable source and target contexts and is of the above judgemental form.

Composition

(sub) $t \diamond \langle s_1, \cdots, s_k, \mathsf{id}_{Y'} \rangle = t\,[\vec{y} \mapsto \vec{s}]$

(SP) $\langle \pi_1 \diamond t, \pi_2 \diamond t \rangle = t$

Parameterised fixpoint

(fix) $\mathsf{cycle}(t) = t \diamond \langle \mathsf{id}_Y, \mathsf{cycle}(t) \rangle$

(Bekič) $\mathsf{cycle}(\langle t, s \rangle) = \langle \pi_2, \mathsf{cycle}(s) \rangle \diamond \langle \mathsf{id}_Y, \mathsf{cycle}(t \diamond \langle \mathsf{id}_{YX}, \mathsf{cycle}(s) \rangle) \rangle$

(nat$_Y$) $\mathsf{cycle}(t) \diamond s = \mathsf{cycle}(t \diamond (s \times \mathsf{id}_X))$

(nat$_X$) $\mathsf{cycle}(s \diamond t) = s \diamond \mathsf{cycle}(t \diamond (\mathsf{id}_Y \times s))$

(CI) $\mathsf{cycle}(\langle t \diamond (\mathsf{id}_X \times \rho_1), \ldots, t \diamond (\mathsf{id}_X \times \rho_m) \rangle) = \Delta_m \diamond \mathsf{cycle}(t \diamond (\mathsf{id}_X \times \Delta_m))$

Deleting trivial cycle

(c2) $\mathsf{cycle}(\curlywedge) = \mathsf{id}$

Commutative monoid

(unitL$\curlywedge$) $\curlywedge \diamond (\{\}_0 \times \mathsf{id}) = \mathsf{id}$

(assoc$\curlywedge$) $\curlywedge \diamond (\mathsf{id} \times \curlywedge) = \curlywedge \diamond (\curlywedge \times \mathsf{id})$

(com$\curlywedge$) $\curlywedge \diamond \mathsf{c} = \curlywedge$

Degenerated bialgebra

(compa) $\Delta \diamond \curlywedge = (\curlywedge \times \curlywedge) \diamond (\mathsf{id} \times \mathsf{c} \times \mathsf{id}) \diamond (\Delta \times \Delta)$

(degen) $\curlywedge \diamond \Delta = \mathsf{id}$

Figure 4: Axioms AxGr for UnCAL graphs

Fig. 4 shows *our proposed axioms* AxGr to characterise UnCAL graphs. These axioms are chosen to soundly and completely represent the original bisimulation of graphs by the equality of this logic. Actually, it is sound: for every axiom $s = t$, s and t are bisimilar. But completeness is not clear only from the axioms. We will show it in §3.

The axiom (sub) is similar to the β-reduction in the λ-calculus, which induces the axioms for cartesian product (cf. the **derived theory** below). The cartesian structure provides a canonical commutative comonoid with comultiplication Δ.

Two terms are paired with a common root by $\{s\} \cup \{t\} = \curlywedge \diamond (s, t)$. The commutative monoid axioms states that this pairing $\{-\} \cup \{-\}$ can be parentheses free in nested case. The degenerate bialgebra axioms state the compatibility between the commutative monoid and comonoid structures. The degenerated bialgebra is suitable to model directed acyclic graphs (cf. [14] §4.5), where it is stated within a PROP [21]. The monoid multiplication $\curlywedge$ expresses a branch in a tree, while the comultiplication Δ expresses a sharing. Commutativity expresses that there is no order between the branches of a node, cf. (commu$\cup$) in the **derived theory** below, and degeneration expresses that the branches of a node form a set (not a sequence), cf. (degen').

Parameterised fixpoint axioms axiomatise a fixpoint operator. They (minus (CI)) are known as the axioms for Conway operators of Bloom and Ésik [3], which ensures that all equalities that holds in cpo semantics do hold. It is also arisen in work independently of Hyland and Hasegawa [15], who established a connection with the notion of traced cartesian categories [20]. There are equalities that Conway operators do not satisfy, e.g. $\mathsf{cycle}(t) = \mathsf{cycle}(t \diamond t)$ does not hold only by the Conway axioms. The axiom (CI) fills this gap, which corresponds to the commutative identities of Bloom and Ésik [3]. This form is taken from [25] and adopted to the UnCAL setting, where $\Delta_m \triangleq \langle \mathsf{id}_\&, \cdots, \mathsf{id}_\& \rangle$, $Y = \langle\langle y_1, \ldots, y_m \rangle\rangle$, $\& \vdash \Delta_m : Y$, $X + Y \vdash t : \&$, $Y \vdash \rho_i : Y$ such that $\rho_i = \langle q_{i1}, \ldots, q_{im} \rangle$ where each q_{ij} is one of $Y \vdash \pi_i : \&$ for $i = 1, \ldots, m$. The axiom (c2) (and derived (c1) below) have been taken as necessary ones for completeness for bisimulation used in several axiomatisations, e.g. [23, 5, 12].

The equational logic EL-UnCAL for UnCAL is a logic to deduce formally proved equations, called *(UnCAL) theorems*. The equational logic is almost the same as ordinary one for algebraic terms. The inference rule of the logic consists of reflexivity, symmetricity, transitivity, congruence rules for all constructors, with the following axiom and the substitution rules.

$$\text{(Ax)}\ \frac{(Y \vdash s = t : X) \in E}{Y \vdash s = t : X} \qquad \text{(Sub)}\ \frac{W \vdash t = t' : X \qquad Z \vdash s_i = s_i' : y_i\ (1 \le i \le k)}{Z + Y' \vdash t\,[\vec{y} \mapsto \vec{s}] = t'\,[\vec{y} \mapsto \vec{s'}] : X}$$

The set of all theorems deduced from the axioms AxGr is called a *(UnCAL) theory*.

Derived theory. The following are formally derivable from the axioms, thus are theorems.

$$\begin{array}{llllll}
\text{(tmnl)} & t = ()_Y \quad \text{for all } Y \vdash t : \langle\langle\rangle\rangle & & \text{(dpair)} & \langle t_1, t_2\rangle \diamond s & = & \langle t_1 \diamond s, t_2 \diamond s\rangle \\
\text{(fst)} & \pi_1 \diamond \langle s, t\rangle = s & & \text{(fsi)} & \langle \pi_1, \pi_2\rangle & = & \mathsf{id} \\
\text{(snd)} & \pi_2 \diamond \langle s, t\rangle = t & & \text{(SP)} & \langle \pi_1 \diamond t, \pi_2 \diamond t\rangle & = & t \\
\text{(bmul)} & ()_\& \times ()_\& = ()_\& \diamond \curlywedge & & \text{(bcomul)} & \Delta \diamond \{\}_0 & = & (\{\}_0 \times \{\}_0) \\
\text{(unitR}\curlywedge\text{)} & \curlywedge \diamond (\mathsf{id} \times \{\}_0) = \mathsf{id} & & \text{(bunit)} & ()_\& \diamond \{\}_0 & = & \mathsf{id} \\
\text{(c1)} & \mathsf{cycle}(\mathsf{id}) = \{\}_0 & & \text{(comm}\cup\text{)} & \{s\} \cup \{t\} & = & \{t\} \cup \{s\} \\
\text{(unR}\diamond\text{)} & t \diamond \mathsf{id} = t & & \text{(unit}\cup\text{)} & \{\{\}\} \cup \{t\} & = & t = \{t\} \cup \{\{\}\} \\
\text{(unL}\diamond\text{)} & \mathsf{id} \diamond t = t & & \text{(assoc}\cup\text{)} & \{\{s\} \cup \{t\}\} \cup \{u\} & = & \{s\} \cup \{\{t\} \cup \{u\}\} \\
\text{(assoc}\diamond\text{)} & (s \diamond t) \diamond u = s \diamond (t \diamond u) & & \text{(degen')} & \{t\} \cup \{t\} & = & t
\end{array}$$

Because of the first three lines, UnCAL has the cartesian products. For (c1), the proof is

$$\mathsf{cycle}(\mathsf{id}) =^{(\text{unitL}\curlywedge)} \mathsf{cycle}(\curlywedge \diamond (\{\}_0 \times \mathsf{id})) =^{(\text{nat}_Y)} \mathsf{cycle}(\curlywedge) \diamond \{\}_0 =^{(\text{c2})} \mathsf{id} \diamond \{\}_0. = \{\}_0.$$

3 Algebraic Semantics of UnCAL

In this section, we consider algebraic semantics of UnCAL. We also give a complete characterisation of the structural recursion, where e can depend on t in (★).

3.1 Iteration Σ-Algebras

We first review the notion of iteration Σ-algebras and various characterisation results by Bloom and Ésik. Let Σ be a signature, i.e. a set of function symbols equipped with arities. We define μ-terms by

$$t ::= x \mid f(t_1, \ldots, t_n) \mid \mu x.t,$$

where x is a variable. We use the convention that a function symbol $f^{(n)} \in \Sigma$ denotes n-ary. For a set V of variables, we denote by $\mathrm{T}(V)$ the set of all μ-terms generated by V. We define ConwayCI as the set of following equational axioms:

Conway equations

$$\mu x.t[s/x] = t[\mu x.\, s[t/x]\,/x], \qquad \mu x.\mu y.t = \mu x.t[x/y]$$

Group equations associated with a group G

$$\mu x.(t[1 \cdot x/x], \ldots, t[n \cdot x/x])_1 = \mu y.(x[y/x], \ldots, [y/x])$$

Note that *the fixed point law*

$$\mu x.t = t[\mu x.t/x]$$

is an instance the first axiom of Conway equations by taking $s = x$. The group equations [11] known as an alternative form of the commutative identities, are an axiom schema parameterised by a finite group $(G, \cdot)$ of order n, whose elements are natural numbers from 1 to n. We also note that the μ-notation is here extended on vectors $(t_1, \ldots, t_n)$, and $(-)_1$ denotes the first component of a vector. Given a vector $x = (x_1, \ldots, x_n)$ of distinct variables, the notation $i \cdot x = (x_{i\cdot 1}, \ldots, x_{i\cdot n})$ is used.

Definition 3.1 ([4]) A *pre-iteration* Σ*-algebra* $(A, (\!|-|\!)_A)$ consists of an nonempty set A and an interpretation function $(\!|-|\!)_A^{(-)} : \mathrm{T}(V) \times A^V \to A$ satisfying

(i) $(\!|x|\!)_A^\rho = \rho(x)$ for each $x \in V$ (iii) $(\!|t|\!)_A = (\!|t'|\!)_A \Longrightarrow (\!|\mu x.t|\!)_A = (\!|\mu x.t'|\!)_A$.

(ii) $(\!|t[t_1/x_1, \cdots, t_n/x_n]|\!)_A^\rho = (\!|t|\!)_A^{\rho'}$ with $\rho'(x_i) = (\!|t_i|\!)_A^\rho$, $\rho'(x) = \rho(x)$ for $x \neq x_i$

A pre-iteration Σ-algebra can be seen as a Σ-algebra $(A, \{f_A \mid f \in \Sigma\})$ with extra operations $(\!|\mu x.t|\!)_A$ for all t. A pre-iteration Σ-algebra A *satisfies* an equation $s = t$ over μ-terms, if $(\!|s|\!)_A = (\!|t|\!)_A$. Let E be a set of equations over μ-terms. An *iteration* Σ*-algebra* is a pre-iteration Σ-algebra that satisfies all equations in ConwayCI. An *iteration* (Σ, E)*-algebra* is an iteration Σ-algebra that satisfies all equations in E. A homomorphism of iteration Σ-algebras $h : A \to B$ is a function such that $h \circ (\!|t|\!)_A = (\!|t|\!) \circ h^V$ for all t. Since the variety of iteration Σ-algebras is exactly the variety of all continuous Σ-algebras ([4] Introduction), the interpretation of $\mu x.t$ in an iteration Σ-algebra can be determined through it.

We now regard each label $\ell \in L$ as an unary function symbol. Then we consider an iteration $L \cup \{0^{(0)}, +^{(2)}\}$-algebra. We define the axiom set AxBR by

$$\begin{array}{lll} s+(t+u) = (s+t)+u & s+t = t+s & t+0 = t \\ \mu x.\, x = 0 & \mu x.\,(x+y) = y & \text{for } y \text{ not containing } x \end{array}$$

and $\mathsf{AxCBR} \triangleq \mathsf{ConwayCI} \cup \mathsf{AxBR}$. We write $\mathsf{AxCBR} \vdash_\mu s = t$ if an equation $s = t$ is derivable from AxCBR by the standard equational logic EL-μ for μ-terms. For example, idempotency is derivable:

$$\mathsf{AxCBR} \ \vdash_\mu \ t + t = t$$

The proof is $t = \mu x.(x+t) = (\mu x.(x+t)) + t = t + t$, which uses the last axiom in AxBR and the fixed point law. Since μ-terms can be regarded as a representation of process terms of regular behavior as Milner shown in [23] (or synchronization trees [3]), the standard notion of strong bisimulation between two μ-terms can be defined. We write $s \sim t$ if they are bisimilar.

Theorem 3.2 ([3, 4, 12, 13])

(i) The axiom set AxCBR completely axiomatises the bisimulation, i.e., $\mathsf{AxCBR} \vdash_\mu s = t \iff s \sim t$

(ii) The set $\mathrm{T}(V)$ *of all* μ*-terms forms a free pre-iteration* Σ*-algebra over* V*.*

(iii) The set $\mathcal{BR}$ *of all regular* L*-labeled trees having* V*-leaves modulo bisimulation forms a free iteration* $(L \cup \{0, +\}, \mathsf{AxBR})$*-algebra over* V *([12] below Lemma 2, [24] Thm. 2).*

Note that $\mathcal{BR}$ stands for **R**egular trees modulo **B**isimulation, and AxBR stands for the axioms for regular trees modulo bisimulation.

3.2 Characterising UnCAL Normal Forms

UnCAL normal forms. Given an UnCAL term t of type &, we compute the *normal form* of t by the following three rewrite rules (N.B. we do not here use the other axioms) as a rewrite system [2], which are oriented equational axioms taken from the derived theory, AxGr and abbreviations.

$$\begin{array}{lrcl} \text{(sub)} & t \diamond \langle s_1, \cdots, s_k, \mathsf{id} \rangle & = & t\,[\vec{y} \mapsto \vec{s}] \\ \text{(Bekič)} & \mathsf{cycle}(\langle t, s \rangle) & = & \langle \pi_2, \mathsf{cycle}(s) \rangle \diamond \langle \mathsf{id}_A, \mathsf{cycle}(t \diamond \langle \mathsf{id}_{A \times V}, \mathsf{cycle}(s) \rangle) \rangle \\ \text{(union)} & \curlywedge \diamond (s, t) & = & \{s\} \cup \{t\} \end{array}$$

Let $\mathcal{M}$ be the set of all rewriting normal forms by the above rules, which finally erases all $\langle -, - \rangle$ and $\diamond$ in a given t. Normal forms are uniquely determined because the rewrite rules are confluent and terminating, hence have the unique normal form property [2]. Then by induction on terms we have that terms in $\mathcal{M}$ follow the grammar

$$\mathcal{M} \ni t ::= \quad y \quad | \quad \ell\!:\!t \quad | \quad \mathsf{cycle}^X(t) \quad | \quad \{\} \quad | \quad \{s\} \cup \{t\} \quad | \quad (x \triangleleft t).$$

Any outermost definition must be of the form $(\& \triangleleft t')$ by the assumption that the original given t is of type $\&$, thus we identity it with t'. Other definitions appear inside of t, as the following cases:

- Case $\{(x_1 \triangleleft t_1)\} \cup \{(x_2 \triangleleft t_2)\}$. We identify it with merely $\{t_1\} \cup \{t_2\}$, because marker names x_1, x_2 are hidden by this construction.
- Case $Y \vdash \mathsf{cycle}^x(x \triangleleft t') : x$. We identify it with merely $\mathsf{cycle}^{\&}(t')$, because these are equivalent by renaming of free maker x.

The *UnCAL normal forms* $\mathcal{N}$ are obtained from $\mathcal{M}$ by these identifications. It is of the form

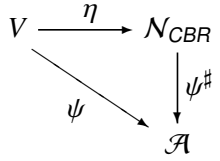
$$\begin{array}{lllllllll} \mathcal{N} \ni & t ::= & y & | & \ell\!:\!t & | & \mathsf{cycle}^X(t) & | & \{\} \quad | \quad \{s\} \cup \{t\} \\ \mathrm{T}(V) \ni & t ::= & y & | & \ell(t) & | & \mu x_1 \ldots \mu x_n . t & | & 0 \quad | \quad s + t \end{array}$$

Every normal form bijectively corresponds to a μ-term in $\mathrm{T}(V)$, i.e. $\mathcal{N} \cong \mathrm{T}(V)$, because each the above construct corresponds to the lower one, where $X = \langle\langle x_1, \ldots, x_n \rangle\rangle$. Hereafter, we may identify normal forms and μ-terms as above. Define the pair of signature and axioms by

$$\mathsf{UnC} \triangleq (L \cup \{0, +\}, \ \mathsf{AxBR}).$$

We regard an arbitrary UnC-algebra $\mathcal{A}$ as an *algebraic model* of UnCAL graphs. First, we show the existence of a free model. Define $\mathcal{N}_{\mathsf{CBR}}$ to be the quotient of $\mathcal{N}$ by the congruence generated by AxCBR.

Proposition 3.3
$\mathcal{N}_{CBR}$ forms a free iteration UnC-algebra over V. Thus for any function $\psi : V \to \mathcal{A}$, there exists an unique UnC-algebra homomorphism $\psi^\sharp$ such that the right diagram commutes, where η is an embedding of variables.



Proposition 3.4 $\mathcal{N}_{CBR} \cong \mathcal{BR}$.

Proof. By Theorem 3.2 (iii). □

3.3 Completeness of the Axioms for Bisimulation

Buneman et al. formulated that UnCAL graphs were identified by *extended bisimulation*, which is a bisimulation on graphs involving ε-edges. As discussed in §1, since our approach is to use only UnCAL terms, it suffices to consider only the standard (strong) bisimulation between UnCAL terms, as done in [23, 3, 12, 13]. We denote by $\sim$ bisimulation for UnCAL term.

In this subsection, we show the completeness of AxGr for bisimulation, using the following Lemma 3.5 that reduces the problem of $\mathsf{EL\text{-}UnCAL}$ to that of EL-μ through UnCAL normal forms. AxCBR has been shown to be complete for the bisimulation [3].

Lemma 3.5 *For UnCAL normal forms $n, m \in \mathcal{N}$, $\mathsf{AxCBR} \vdash_\mu n = m \iff Y \vdash n = m : X$ is derivable from AxGr in $\mathsf{EL\text{-}UnCAL}$.*

Proof. [⇒] : By induction on proofs of EL-μ. For every axiom in AxCBR, there exists the corresponding axiom in AxGr or an EL-UnCAL theorem, hence it can be emulated.
[⇐] : By induction on proofs of EL-UnCAL. Let $s = t$ is an axiom of EL-UnCAL. It easy to see that taking normal forms of both side, they are equal term, or correspond to an axiom in AxCBR or EL-μ theorem. □

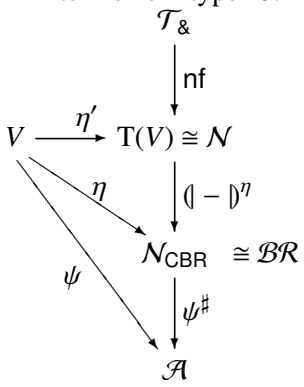
Theorem 3.6 (Completeness) *AxGr is sound and complete for the bisimulation, i.e.,*
$Y \vdash s = t \ : X$ *is derivable from AxGr in EL-UnCAL iff* $s \sim t$.

Proof. [⇒] : Because every axiom in AxGr is bisimilar, and the bisimulation is closed under contexts and substitutions [8].
[⇐] : Suppose $s \sim t$. Since for each rewrite rule for the normalisation function nf, both sides of the rule is bisimilar, nf preserves the bisimilarity. So we have $s \sim \mathsf{nf}(s) \sim \mathsf{nf}(t) \sim t$. Since AxCBR is complete axioms of bisimulation [3, 12], $\mathsf{AxCBR} \vdash_\mu \mathsf{nf}(s) = \mathsf{nf}(t)$. By Lemma 3.5, we have a theorem $Y \vdash \mathsf{nf}(s) = \mathsf{nf}(t) \ : X$. Thus $s = t$ is derivable. □

3.4 Interpretation in Algebraic Models

To interpret UnCAL terms and equations, we connect two freeness results in Thm. 3.2. Since UnCAL normal forms $\mathcal{N}$ is isomorphic to a free pre-iteration algebra $\mathrm{T}(V)$, it has the universal property. Define $\mathcal{T}_\&$ to be the set of all well-typed UnCAL terms of type &. We define $\mathsf{nf} : \mathcal{T}_\& \to \mathcal{N}$ by the function to compute the UnCAL normal form of a term. Then for any derivable equation $Y \vdash s = t \ : X$ in EL-UnCAL, we have $\mathsf{AxCBR} \vdash \mathsf{nf(s)} = \mathsf{nf(t)}$ by Lemma 3.5, thus for all assignment $\psi : V \to \mathcal{A}$,

$$\psi^\sharp(\!|\, \mathsf{nf}(s) \,|\!)^\eta = \psi^\sharp(\!|\, \mathsf{nf}(t) \,|\!)^\eta$$



where η and η' are embedding of variables.

Since $\mathcal{N}_{\mathsf{CBR}} \cong \mathcal{BR}$, we name the isomorphisms $\underline{(-)} : \mathcal{N}_{\mathsf{CBR}} \to \mathcal{BR}$ and $\overline{(-)} : \mathcal{BR} \to \mathcal{N}_{\mathsf{CBR}}$. We write simply a normal form t to denote a representative $[t]$ in $\mathcal{N}_{\mathsf{CBR}}$. Thus given a normal form t (which is a syntactic term, always finite), $\underline{t}$ is a (possibly infinite) regular tree by obtained by expanding cycles in t using fixpoints. Conversely, notice that since $\underline{t}$ is a tree, there are no cycles and the original cycles in t are infinitely expanded. Since $\mathcal{N} \cong \mathrm{T}(V)$, the functions $\underline{(-)}$ may also be applied to μ-terms. The iteration UnC-algebra $\mathcal{BR}$ has operations $0_{\mathcal{BR}} = \underline{\{\,\}}$, $+_{\mathcal{BR}}(r,s) = \underline{\{\overline{r}\} \cup \{\overline{s}\}}$, $\ell_{\mathcal{BR}}(r) = \underline{\ell(\overline{r})}$.

3.5 Deriving structural recursion of involved case

Next we model UnCAL's structural recursion of graphs. We use pairs of "the recursive computation" and the history of data structure. This is similar to the technique of paramorphism [22], which is a way to represent primitive recursion in terms of "fold" in functional programming. Our universal characterisation of graphs is the key to make this possible by the unique homomorphism from the free pre-iteration UnC-algebra $\mathcal{N}$ using the above analysis.

We take a term $X \vdash e_\ell(v,r) \ : X$ involving metavariables v and r, where $e_\ell(F(t),t)$ is the right-hand side e of $F(\ell\!:\!t)$ in (★) . For example, in case of the example `f1` in Introduction (see also Example 3.9), we take

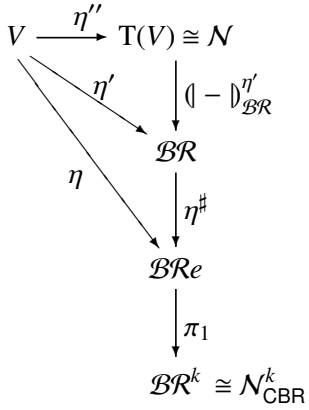
$$\begin{array}{llll} e_\ell(v,r) \triangleq \texttt{result:}r, & e_\ell(F(t),t) = \texttt{result:}t & \text{if } \ell = \texttt{ethnicGroup} \\ e_\ell(v,r) \triangleq v, & e_\ell(F(t),t) = F(t) & \text{if } \ell \neq \texttt{ethnicGroup} \end{array}$$

We construct a *specific* iteration UnC-algebra $\mathcal{BR}e$ for $\{e_\ell(v,r)\}_{\ell\in L}$. Let $k \triangleq |X|$. Without loss of generality, we can assume that $e_\ell(v,r)$ is of the form $\langle t_1, \cdots, t_k\rangle$ where every t_i is a normal form. We define the iteration UnC-algebra $\mathcal{BR}e = \mathcal{BR}^k \times \mathcal{BR}$ having operation

$$\ell_{\mathcal{BR}e}(v,r) = (\underline{e_\ell(v,r)},\ \underline{\ell\!:\!\bar{r}}), \qquad 0_{\mathcal{BR}e} = (\underline{\vec{\{\}}}, \underline{\{\}})$$

and $+_{\mathcal{BR}e}$ is an obvious tuple extensions of $+_{\mathcal{BR}}$. Here $\vec{\{\}}$ is the k-tuple of $\{\}$. Hereafter, we will use this convention $\vec{o}$ of tuple extension of an operator o.

Then, two freeness results in Thm. 3.2 are depicted in the right diagram, where $\eta(x) = (\underline{x_1, \cdots, x_k}, \underline{x})$. Since $\mathrm{T}(V) \cong \mathcal{N}$, the interpretation in $\mathcal{BR}e$ is described as



$$(\!|x|\!)^\eta_{\mathcal{BR}e} = \eta(x), \quad (\!|\{\}|\!)^\eta_{\mathcal{BR}e} = 0_{\mathcal{BR}e}, \quad (\!|\{s\}\cup\{t\}|\!)^\eta_{\mathcal{BR}e} = (\!|s|\!)_{\mathcal{BR}e} +_{\mathcal{BR}e} (\!|t|\!)_{\mathcal{BR}e}$$
$$(\!|\ell\!:\!t|\!)^\eta_{\mathcal{BR}e} = \ell_{\mathcal{BR}e}((\!|t|\!)^\eta_{\mathcal{BR}e}), \quad (\!|\mathsf{cycle}(t)|\!)^\eta_{\mathcal{BR}e} = \eta^\sharp(\underline{\mathsf{cycle}(t)})$$

Now $(\!|-|\!)^\eta_{\mathcal{BR}e}$ is characterised as the unique pre-iteration $L\cup\{0,+\}$-algebra homomorphism from $\mathrm{T}(V)$ that extends η. Defining

$$\phi \triangleq \pi_1 \circ (\!|-|\!)^\eta_{\mathcal{BR}e} : \mathcal{N} \longrightarrow \mathcal{BR}^k \cong \mathcal{N}^k_{\mathsf{CBR}},$$

it is the unique function satisfying

$$\phi(x) = (\underline{x_1}, \ldots, \underline{x_k}), \qquad \phi(\{\}) = \underline{\vec{\{\}}}, \qquad \phi(\{s\}\cup\{t\}) = \underline{\phi(s)\ \vec{\cup}\ \phi(t)},$$
$$\phi(\ell\!:\!t) = \underline{e_\ell(v,t)}, \qquad \phi(\mathsf{cycle}(t)) = \pi_1 \circ \eta^\sharp(\underline{\mathsf{cycle}(t)})$$

The function ϕ takes normal forms of the type $\&$. For non-normal forms, just precompose nf, i.e., define the function $\Phi : \mathcal{T}_\& \to \mathcal{N}^k_{\mathsf{CBR}}$ by $\Phi(s) \triangleq \phi(\mathsf{nf}(s))$, thus, $\Phi^{|X|} : \mathcal{T}_X \to {\mathcal{N}_{\mathsf{CBR}}}^{k|X|} \to \mathcal{T}^k_X$, because $\mathcal{T}_X \cong \mathcal{T}^{|X|}_\&$. In summary, we have the following, where s is a possibly non-normal form

$$\begin{array}{llll}
\Phi(s) & = \phi(\mathsf{nf}(s)) & \phi(x) & = \langle x_1, \cdots, x_k\rangle \qquad \phi(\{\}) = \vec{\{\}} \\
\Phi^{|X|+|Y|}(\langle t_1, t_2\rangle) & = \Phi^{|X|}(t_1) \mathbin{\vec{\times}} \Phi^{|Y|}(t_2) & \phi(\ell\!:\!t) & = e_\ell(\phi(t), t) \qquad \phi(t_1 \cup t_2) = \phi(t_1)\ \vec{\cup}\ \phi(t_2) \\
\Phi^0(()) & = () & \phi(\mathsf{cycle}(t)) & = \pi_1 \circ \eta^\sharp(\underline{\mathsf{cycle}(t)})
\end{array} \tag{2}$$

where $\vec{\times}$ is the "zip" operator of two tuples. Here we use a map $\mathcal{N}_{\mathsf{CBR}} \to \mathrm{Tm}(V)$ to regard a normal form modulo AxCBR as a term, for which any choise of representative is harmless, because UnCAL graphs are identified by bisimulation and AxCBR axiomatises it. Identifying three kinds functions $\Phi, \Phi^{|X|}, \phi$ as a single function (also denoted by Φ, by abuse of notion) on $\mathrm{Tm}(V)$, this Φ is essentially what Buneman et al. [8] called the structural recursion on graphs for the case that e depends on t. Actually, we could make the characterisation more precise than [8], i.e., we obtain also the laws for the cases of $\diamond$ (by the case $\Phi(s) = \phi(\mathsf{nf}(s))$) and cycle, which tells how to compute them.

This is not merely rephrasing the known result, but also a stronger characterisation, which gives precise understanding of the structural recursion on graphs:

(i) Buneman et al. stated that (1) without $(\bowtie)$ is a *property* ([8] Prop. 3) of a "structural recursive function on graphs" defined by the algorithms in [8]. This property (i.e. soundness) is desirable, but unfortunately, no completeness was given. There may be many functions that satisfy the property. In contrast to it, our characterisation is sound and *complete*: (2) determines a *unique* function by the universality.

(ii) This derivation does not entail $\Phi(s \diamond t) = \Phi(s) \diamond \Phi(t)$. It tells us that the only way to compute $\Phi(s \diamond t)$ is to compute the normal form of $s \diamond t$ and then apply ϕ.

(iii) This analysis does not entail $\Phi(\mathsf{cycle}(t)) = \mathsf{cycle}(\Phi(t))$ either. The iteration algebra structure tells us that the homomorphism ϕ maps a term $\mathsf{cycle}(t)$ to its interpretation in $\mathcal{BR}e$ where the cycles are expanded in a regular tree and at the same time, labels ℓ are interpreted using the operations of $\mathcal{BR}e$.

(iv) The structure preserved by structural recursion is the *(pre-)iteration algebra structure*. The structural recursive function ϕ is the composition of a pre-iteration algebra homomorphism, an a iteration algebra homomorphism and a projection.

3.6 Examples

We may use the notation $\{t_1, t_2, \ldots\}$ as the abbreviation of $\{t_1\} \cup \{t_2\} \cup \cdots$.

Example 3.7 ([8] Replace all labels with a) This is the example considered in Introduction.

```
sfun f2(L:T) = a:f2(T)
```

In this case, the recursion does *not* depend on `T` (because the right-hand side uses merely `f2(T)`). We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\ell_{\mathcal{BR}e}(v, r) = (\mathtt{a}\!:\!v,\ \ell\!:\!r).$$

(We may omit over and underlines to denote the isomorphisms for simplicity). Then Φ is the desired structural recursive function `f2`. E.g.

$$\Phi(\mathtt{b}\!:\!\mathsf{cycle}(\mathtt{c}\!:\!\&)) \;=\; \mathtt{a}\!:\!\phi(\mathsf{cycle}(\mathtt{c}\!:\!\&)) \;=\; \mathtt{a}\!:\!\pi_1 \circ \eta^\sharp(\mathtt{c}\!:\!\mathtt{c}\!:\!\cdots) \;=\; \mathtt{a}\!:\!(\overline{\mathtt{a}\!:\!\mathtt{a}\!:\!\cdots}) \;=\; \mathtt{a}\!:\!\mathsf{cycle}(\mathtt{a}\!:\!\&)$$

Example 3.8 ([8] Double the children of each node)

```
sfun f4(L:T) = {a:f4(T)} ∪ {b:f4(T)}
```

Example of execution.

```
f4(a:b:c:{})
⇝ {a:{ a:{a:{}, b:{}}, b:{a:{}, b:{}} }} ∪ {b:{ a:{a:{}, b:{}}, b:{a:{}, b:{}} }}
```

This case does *not* depend on `T`. We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\ell_{\mathcal{BR}e}(v, r) = (\{\mathtt{a}\!:\!v\} \cup \{\mathtt{b}\!:\!v\},\ \ell\!:\!r).$$

Then Φ gives the structural recursive function defined by `f4`.

Example 3.9 ([8] Retrieve all ethnic groups) We revisit the example given in §1.
For the structural recursive recursive definition of `f1`,

```
sfun f1(L:T) = if L = ethnicGroup then (result:T) else f1(T)
```

This case *does* depend on `T`. Example of execution:

```
f1(sd) ⇝ {result:"Celtic":{}, result:"Portuguese":{}, result:"Italian":{}}
```

We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\begin{aligned}\texttt{ethnicGroup}_{\mathcal{BR}e}(v,r) &\triangleq (\texttt{result:}r,\ \texttt{ethnicGroup:}r)\\ \ell_{\mathcal{BR}e}(v,r) &\triangleq (v,\ \ell\texttt{:}r) \text{ for } \ell \neq \texttt{ethnicGroup}\end{aligned}$$

Then Φ is the structural recursive function defined by `f1`:

```
Φ (sd) = {result:"Celtic":{}, result:"Portuguese":{}, result:"Italian":{}}
```

Example 3.10 Consider another example in §1 of `aa?`. This case *does* depend on `T`. We define the iteration UnC-algebra $\mathcal{BR}e$ by

$$\begin{aligned}\texttt{a}_{\mathcal{BR}e}(v,r) &\triangleq (\texttt{a?}(r),\ \texttt{a:}r)\\ \ell_{\mathcal{BR}e}(v,r) &\triangleq (v,\ \ell\texttt{:}r) \quad \text{for } \ell \neq \texttt{a}.\end{aligned}$$

Then Φ gives the structural function `aa?`

$$\Phi(\texttt{(a:\&)@(a:\{\})}) = \phi(\,\mathsf{nf}(\texttt{(a:\&)@(a:\{\})})\,) = \phi(\texttt{a:a:\{\}}) = \texttt{true:\{\}}$$

$$\Phi(\texttt{cycle(a:\&)}) = \pi_1 \circ \eta^\sharp(\underline{\texttt{cycle(a:\&)}}) = \pi_1 \circ \eta^\sharp(\texttt{a:a:}\cdots) = \pi_1(\texttt{a?(a:}\cdots\texttt{),a:}\cdots) = \texttt{true} : \{\}$$

4 Conclusion

In this paper, we have shown an application of Bloom and Ésik's iteration algebras to model graph data used in UnQL/UnCAL for describing and manipulating graphs. We have formulated UnCAL and given an axiomatisation of UnCAL graphs that characterises the original bisimulation. We have given algebraic semantics using Bloom and Ésik's iteration iteration algebras. The main result of this paper was to show that completeness of our equational axioms for UnCAL for the original bisimulation of UnCAL graphs via iteration algebras. As a consequence, we have given a clean characterisation of the computation mechanism of UnCAL, called "structural recursion on graphs" using free iteration algebra.

Acknowledgments. I am grateful to Kazutaka Matsuda and Kazuyuki Asada for discussions about UnCAL and its interpretation, and their helpful comments on a draft of the paper. A part of this work was done while I was visiting National Institute of Informatics (NII) during 2013 – 2014.

References

[1] K. Asada, S. Hidaka, H. Kato, Z. Hu & K. Nakano (2013): *A parameterized graph transformation calculus for finite graphs with monadic branches*. In: *Proc. of PPDP '13*, pp. 73–84, doi:10.1145/2505879.2505903.

[2] F. Baader & T. Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press.

[3] S. L. Bloom & Z. Ésik (1993): *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science, Springer.

[4] S. L. Bloom & Z. Ésik (1994): *Solving Polynomial Fixed Point Equations*. In: *Proc. of MFCS'94*, LNCS 841, pp. 52–67, doi:10.1007/3-540-58338-6_58.

[5] S. L. Bloom, Z. Ésik & D. Taubner (1993): *Iteration Theories of Synchronization Trees*. *Inf. Comput.* 102(1), pp. 1–55, doi:10.1006/inco.1993.1001.

[6] P. Buneman, S. Davidson, G. Hillebrand & D. Suciu (1996): *A query language and optimization techniques for unstructured data* . In: *Proc. of ACM-SIGMOD'96*, doi:10.1145/233269.233368.

[7] P. Buneman, S. B. Davidson, M. F. Fernandez & D. Suciu (1997): *Adding Structure to Unstructured Data*. In: *Proc. of ICDT '97*, pp. 336–350, doi:10.1007/3-540-62222-5_55.

[8] P. Buneman, M. F. Fernandez & D. Suciu (2000): *UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion*. *VLDB J.* 9(1), pp. 76–110, doi:10.1007/s007780050084.

[9] R.L. Crole (1993): *Categories for Types*. Cambridge Mathematical Textbook.

[10] Z. Ésik (1999): *Axiomatizing Iteration Categories*. *Acta Cybernetica* 14, pp. 65–82.

[11] Z. Ésik (1999): *Group Axioms for Iteration*. *Inf. Comput.* 148(2), pp. 131–180, doi:10.1006/inco.1998.2746.

[12] Z. Ésik (2000): *Axiomatizing the Least Fixed Point Operation and Binary Supremum*. In: *Proc. of Computer Science Logic 2000*, LNCS 1862, pp. 302–316, doi:10.1007/3-540-44622-2_20.

[13] Z. Ésik (2002): *Continuous Additive Algebras and Injective Simulations of Synchronization Trees*. *J. Log. Comput.* 12(2), pp. 271–300, doi:10.1093/logcom/12.2.271.

[14] M. P. Fiore & M. D. Campos (2013): *The Algebra of Directed Acyclic Graphs*. In: *Computation, Logic, Games, and Quantum Foundations*, LNCS 7860, pp. 37–51, doi:10.1007/978-3-642-38164-5_4.

[15] M. Hasegawa (1997): *Models of Sharing Graphs: A Categorical Semantics of* let *and* letrec. Ph.D. thesis, University of Edinburgh. Distinguished Dissertation Series, Springer-Verlag, 1999.

[16] M. Hasegawa (1997): *Recursion from Cyclic Sharing: Traced Monoidal Categories and Models of Cyclic Lambda Calculi*. In: *Proc. of TLCA'97*, pp. 196–213, doi:10.1007/3-540-62688-3_37.

[17] S. Hidaka, K. Asada, Z. Hu, H. Kato & K. Nakano (2013): *Structural recursion for querying ordered graphs*. In: *Proc. of ACM SIGPLAN ICFP'13*, pp. 305–318, doi:10.1145/2500365.2500608.

[18] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda & K. Nakano (2010): *Bidirectionalizing graph transformations*. In: *Proc. of ICFP 2010*, pp. 205–216, doi:10.1145/1863543.1863573.

[19] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, K. Nakano & I. Sasano (2011): *Marker-Directed Optimization of UnCAL Graph Transformations*. In: *Proc. of LOPSTR'11*, pp. 123–138, doi:10.1007/978-3-642-32211-2_9.

[20] A. Joyal, R. Street & D. Verity (1996): *Traced monoidal categories*. *Mathematical Proceedings of the Cambridge Philosophical Society* 119(3), pp. 447–468, doi:10.1017/S0305004100074338.

[21] S. Mac Lane (1971): *Categories for the Working Mathematician*. *Graduate Texts in Mathematics* 5, Springer-Verlag, doi:10.1007/978-1-4612-9839-7.

[22] L. G. L. T. Meertens (1992): *Paramorphisms*. *Formal Asp. Comput.* 4(5), pp. 413–424, doi:10.1007/BF01211391.

[23] R. Milner (1984): *A Complete Inference System for a Class of Regular Behaviours*. *J. Comput. Syst. Sci.* 28(3), pp. 439–466, doi:10.1016/0022-0000(84)90023-0.

[24] P. M. Sewell (1995): *The Algebra of Finite State Processes*. Ph.D. thesis, University of Edinburgh. Dept. of Computer Science technical report CST-118-95, also published as LFCS-95-328.

[25] A. K. Simpson & G. D. Plotkin (2000): *Complete Axioms for Categorical Fixed-Point Operators*. In: *Proc. of LICS'00*, pp. 30–41, doi:10.1109/LICS.2000.855753.