Data Linkage Algebra, Data Linkage Dynamics, and Priority Rewriting

Bergstra, J.; Middelburg, C.A.

DOI
10.3233/FI-2013-950

Publication date
2013

Document Version
Final published version

Published in
Fundamenta Informaticae

Fundamenta Informaticae 128 (2013) 367–412
DOI 10.3233/FI-2013-950
IOS Press

Data Linkage Algebra, Data Linkage Dynamics, and Priority Rewriting

J.A. Bergstra, C.A. Middelburg*
Informatics Institute, University of Amsterdam
Science Park 904, 1098 XH Amsterdam, the Netherlands
{*J.A.Bergstra,C.A.Middelburg*}*@uva.nl*

Abstract. We introduce an algebra of data linkages. Data linkages are intended for modelling the states of computations in which dynamic data structures are involved. We present a simple model of computation in which states of computations are modelled as data linkages and state changes take place by means of certain actions. We describe the state changes and replies that result from performing those actions by means of a term rewriting system with rule priorities. The model in question is an upgrade of molecular dynamics. The upgrading is mainly concerned with the features to deal with values and the features to reclaim garbage.

Keywords: data linkage algebra, data linkage dynamics, garbage collection, priority rewrite system

1. Introduction

The data structures involved in programming are quite often dynamic data structures, i.e. data structures that may vary in size and shape. Dynamic data structures are data structures whose constituent parts are linked in one way or another to facilitate the insertion and deletion of constituent parts. Scientific work on dynamic data structure is generally concerned with specific dynamic data structures that show a simple shape, such as linked lists and various kinds of tree structures, or with dynamic data structures in general. In the former case, the dynamic data structures concerned are regularly considered in abstraction from their representation by means of pointers. In the latter case, however, dynamic data structures are

*Address for correspondence: Informatics Institute, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, the Netherlands

primarily considered at the level of their representation by means of pointers. Although it is often useful to abstract from the representation by means of pointers, it seems that no serious attempts have been made to provide a setting in which this is possible. The aim of the current paper is to provide such a setting.

We introduce an algebra, called data linkage algebra, of which the elements are intended for modelling the states of computations in which dynamic data structures are involved. We also present a simple model of computation, called data linkage dynamics, in which states of computations are modelled as elements of data linkage algebra and state changes take place by means of certain actions. We describe the state changes and replies that result from performing those actions by means of a term rewriting system with rule priorities [2].

Term rewriting systems take an important place in theoretical computer science. Moreover, because term rewriting is a practical mechanism for doing calculations, term rewriting systems have many applications in software engineering. Term rewriting systems with rule priorities, also called priority rewrite systems, were first proposed in [2]. Further studies of priority rewrite systems can, for example, be found in [23, 37, 40, 42]. Applications of priority rewrite systems are found in various areas, see e.g. [18, 31, 45]. The rule priorities add expressive power: the reduction relation of a priority rewrite system is not decidable in general. It happens that it is quite convenient to describe the state changes and replies that result from performing the actions of data linkage dynamics by means of a priority rewrite system. Moreover, the priority rewrite system in question turns out computationally unproblematic: its reduction relation is decidable.

We take the view that the behaviours produced by sequential programs under execution are threads as considered in basic thread algebra [7] (see also [12, Chapter 2]).[1] These threads represent in a direct way the behaviours produced by instruction sequences under execution. A thread proceeds by performing actions in a sequential fashion. Upon each action performed by a thread, a reply from the execution environment, which takes the action as an instruction to be processed, determines how the thread proceeds. In [13], basic thread algebra has been extended with services, which represent an abstract view on the behaviours exhibited by the components of an execution environment that are capable of processing particular instructions independently, and use and apply operators, which have to do with the effects of the interaction between threads and services that takes place during instruction processing (see also [12, Chapter 3]).

The state changes and replies that result from performing the actions of data linkage dynamics can be achieved by means of services. In the current paper, we explain how basic thread algebra extended with services and use operators can be combined with data linkage dynamics such that the whole can be used for studying issues concerning the use of dynamic data structures in programming.

Data linkage dynamics is an upgrade of molecular dynamics, which was first described in [3]. The name molecular dynamics refers to the molecule metaphor used to explain the model. By that, there is no clue in the name itself to what it stands for. To remedy this defect, the upgrade has been renamed to data linkage dynamics. The upgrading is mainly concerned with the features to deal with values and the features to reclaim garbage. In data linkage dynamics, calculations in a non-trivial finite meadow [5, 16, 17], such as a finite field with the multiplicative inverse operation made total by imposing that the multiplicative inverse of zero is zero, can be done. The features to reclaim garbage include: full garbage collection,

[1] In [7], basic thread algebra is introduced under the name basic polarized process algebra. Prompted by the development of thread algebra [9], which is a design on top of it, basic polarized process algebra has been renamed to basic thread algebra.

restricted garbage collection (as if reference counts are used), safe disposal of potential garbage, and unsafe disposal of potential garbage.

In [11], a description of the state changes and replies that result from performing the actions of molecular dynamics was given in the world of sets. In the current paper, we relate this description to the description based on data linkage algebra by widening the former to a description for data linkage dynamics and showing that the widened description agrees with the description based on data linkage algebra.

Data linkage dynamics in itself is meant to convey a theoretical understanding of the pragmatic concept of a dynamic data structure as exploited in the practice of programming. Such a theoretical understanding is a valuable complement of the understanding of how specific programs use dynamic data structures, which is acquired by means of dynamic analysis tools that analyze how programs build and modify them (see e.g. [39]). We expect that a theoretical understanding will become increasingly important to the development of successful software systems. Below, we give a first impression of what is to be expected from data linkage dynamics as a setting in which issues concerning dynamic data structures are studied.

In work on dynamic data structures, on the one hand issues concerning specific dynamic data structures, in particular performance issues and computational complexity issues, are studied (see e.g. [22, 34, 38, 44]). In the studies in question, the dynamic data structures concerned are mostly considered in abstraction from their representation by means of pointers. We believe that issues like these ones can also be studied in the setting of data linkage dynamics, and we expect that the use of a single setting facilitates uniformity in the way in which the same issue is approached for different dynamic data structures.

In work on dynamic data structures, on the other hand issues concerning dynamic data structures in general, particularly issues related to optimization and parallelization of sequential programs that make use of dynamic data structures, are studied (see e.g. [19, 20, 27, 41]). In the studies in question, dynamic data structures are mostly considered at the level of their representation by means of pointers. We consider it likely that, for issues like these ones, more general results can be obtained by studying the issues concerned in the setting of data linkage dynamics, where they can be considered in abstraction from their representation by means of pointers.

This paper is organized as follows. First, we introduce data linkage algebra (Section 2). Next, after a short review of priority rewrite systems (Section 3), we present data linkage dynamics (Sections 4, 5, 6, and 7). After that, we review basic thread algebra and its extension with services and use operators (Sections 8 and 9) and explain how this extension of basic thread algebra can be combined with data linkage dynamics (Section 10). Following this, we give the alternative description of data linkage dynamics in the world of sets (Sections 11, 12, and 13). Finally, we make some concluding remarks (Section 14).

Some familiarity with term rewriting systems is assumed. The desirable background can, for example, be found in [21, 32, 33]. For convenience, the basic definitions and results regarding term rewriting systems are collected in an appendix.

2. Data Linkage Algebra

In this section, we introduce the algebraic theory DLA (Data Linkage Algebra).

The elements of the initial algebra of DLA are intended for modelling the states of computations in which dynamic data structures are involved. These states resemble collections of molecules composed of atoms. An atom can have fields and each of those fields can contain an atom. An atom together with the ones it has links to via fields can be viewed as a sub-molecule, and a sub-molecule that is not contained in a larger sub-molecule can be viewed as a molecule. Thus, the collection of molecules that make up a state can be viewed as a fluid. To make atoms reachable, there are spots and each spot can contain an atom.

Disengaging from the molecule metaphor, atoms will henceforth be called atomic objects. Moreover, sub-molecules, molecules and fluids will henceforth not be distinguished and commonly be called data linkages.

In DLA, it is assumed that a fixed but arbitrary finite set Spot of *spots*, a fixed but arbitrary finite set Field of *fields*, a fixed but arbitrary finite set AtObj of *atomic objects*, and a fixed but arbitrary finite set Value of *values* have been given.

DLA has one sort: the sort $\mathbf{DL}$ of *data linkages*. To build terms of sort $\mathbf{DL}$, DLA has the following constants and operators:

- for each $s \in \mathsf{Spot}$ and $a \in \mathsf{AtObj}$, the *spot link* constant $\xrightarrow{s} a : \mathbf{DL}$;
- for each $a \in \mathsf{AtObj}$ and $f \in \mathsf{Field}$, the *partial field link* constant $a \xrightarrow{f} : \mathbf{DL}$;
- for each $a, b \in \mathsf{AtObj}$ and $f \in \mathsf{Field}$, the *field link* constant $a \xrightarrow{f} b : \mathbf{DL}$;
- for each $a \in \mathsf{AtObj}$ and $n \in \mathsf{Value}$, the *value association* constant $(a)_n : \mathbf{DL}$;
- the *empty data linkage* constant $\emptyset : \mathbf{DL}$;
- the binary *data linkage combination* operator $\oplus : \mathbf{DL} \times \mathbf{DL} \to \mathbf{DL}$;
- the binary *data linkage overriding combination* operator $\oplus' : \mathbf{DL} \times \mathbf{DL} \to \mathbf{DL}$.

Terms of sort $\mathbf{DL}$ are built as usual (see e.g. [43, 46]). Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{DL}$, including X, Y, Z. We use infix notation for data linkage combination and data linkage overriding combination.

Let L and L' be closed DLA terms. Then the constants and operators of DLA can be explained as follows:

- $\xrightarrow{s} a$ is the atomic data linkage that consists of a link via spot s to atomic object a;
- $a \xrightarrow{f}$ is the atomic data linkage that consists of a partial link from atomic object a via field f;
- $a \xrightarrow{f} b$ is the atomic data linkage that consists of a link from atomic object a via field f to atomic object b;
- $(a)_n$ is the atomic data linkage that consists of an association of the value n with atomic object a;
- $\emptyset$ is the data linkage that does not contain any atomic data linkage;
- $L \oplus L'$ is the union of the data linkages L and L';
- $L \oplus' L'$ differs from $L \oplus L'$ as follows:
  - if L contains spot links via spot s and L' contains spot links via spot s, then the former links are overridden by the latter ones;

Table 1. Axioms of DLA

$X \oplus Y = Y \oplus X$	
$X \oplus (Y \oplus Z) = (X \oplus Y) \oplus Z$	
$X \oplus X = X$	
$X \oplus \emptyset = X$	
$\emptyset \oplus' X = X$	
$X \oplus' \emptyset = X$	
$X \oplus' (Y \oplus Z) = (X \oplus' Y) \oplus (X \oplus' Z)$	
$(X \oplus (\xrightarrow{s} a)) \oplus' (\xrightarrow{s} b) = X \oplus' (\xrightarrow{s} b)$	
$(X \oplus (a \xrightarrow{f})) \oplus' (a \xrightarrow{f}) = X \oplus' (a \xrightarrow{f})$	
$(X \oplus (a \xrightarrow{f} b)) \oplus' (a \xrightarrow{f}) = X \oplus' (a \xrightarrow{f})$	
$(X \oplus (a \xrightarrow{f})) \oplus' (a \xrightarrow{f} b) = X \oplus' (a \xrightarrow{f} b)$	
$(X \oplus (a \xrightarrow{f} b)) \oplus' (a \xrightarrow{f} c) = X \oplus' (a \xrightarrow{f} c)$	
$(X \oplus (a)_n) \oplus' (a)_m = X \oplus' (a)_m$	
$(X \oplus (\xrightarrow{s} a)) \oplus' (\xrightarrow{t} b) = (X \oplus' (\xrightarrow{t} b)) \oplus (\xrightarrow{s} a)$	if $s \neq t$
$(X \oplus (a \xrightarrow{f})) \oplus' (\xrightarrow{s} b) = (X \oplus' (\xrightarrow{s} b)) \oplus (a \xrightarrow{f})$	
$(X \oplus (a \xrightarrow{f} b)) \oplus' (\xrightarrow{s} c) = (X \oplus' (\xrightarrow{s} c)) \oplus (a \xrightarrow{f} b)$	
$(X \oplus (a)_n) \oplus' (\xrightarrow{s} b) = (X \oplus' (\xrightarrow{s} b)) \oplus (a)_n$	
$(X \oplus (\xrightarrow{s} a)) \oplus' (b \xrightarrow{f}) = (X \oplus' (b \xrightarrow{f})) \oplus (\xrightarrow{s} a)$	
$(X \oplus (a \xrightarrow{f})) \oplus' (b \xrightarrow{g}) = (X \oplus' (b \xrightarrow{g})) \oplus (a \xrightarrow{f})$	if $a \neq b \vee f \neq g$
$(X \oplus (a \xrightarrow{f} b)) \oplus' (c \xrightarrow{g}) = (X \oplus' (c \xrightarrow{g})) \oplus (a \xrightarrow{f} b)$	if $a \neq c \vee f \neq g$
$(X \oplus (a)_n) \oplus' (b \xrightarrow{f}) = (X \oplus' (b \xrightarrow{f})) \oplus (a)_n$	
$(X \oplus (\xrightarrow{s} a)) \oplus' (b \xrightarrow{f} c) = (X \oplus' (b \xrightarrow{f} c)) \oplus (\xrightarrow{s} a)$	
$(X \oplus (a \xrightarrow{f})) \oplus' (b \xrightarrow{g} c) = (X \oplus' (b \xrightarrow{g} c)) \oplus (a \xrightarrow{f})$	if $a \neq b \vee f \neq g$
$(X \oplus (a \xrightarrow{f} b)) \oplus' (c \xrightarrow{g} d) = (X \oplus' (c \xrightarrow{g} d)) \oplus (a \xrightarrow{f} b)$	if $a \neq c \vee f \neq g$
$(X \oplus (a)_n) \oplus' (b \xrightarrow{f} c) = (X \oplus' (b \xrightarrow{f} c)) \oplus (a)_n$	
$(X \oplus (\xrightarrow{s} a)) \oplus' (b)_n = (X \oplus' (b)_n) \oplus (\xrightarrow{s} a)$	
$(X \oplus (a \xrightarrow{f})) \oplus' (b)_n = (X \oplus' (b)_n) \oplus (a \xrightarrow{f})$	
$(X \oplus (a \xrightarrow{f} b)) \oplus' (c)_n = (X \oplus' (c)_n) \oplus (a \xrightarrow{f} b)$	
$(X \oplus (a)_n) \oplus' (b)_m = (X \oplus' (b)_m) \oplus (a)_n$	if $a \neq b$

- if L contains partial field links and/or field links from atomic object a via field f and L' contains partial field links and/or field links from atomic object a via field f, then the former partial field links and/or field links are overridden by the latter ones;
- if L contains value associations with atomic object a and L' contains value associations with atomic object a, then the former value associations are overridden by the latter ones.

Following the introduction of DLA, we will present a simple model of computation that pertains to the use of dynamic data structures in programming. DLA provides a notation that enables us to get a clear picture of computations in the context of that model.

The axioms of DLA are given in Table 1. In this table, s and t stand for arbitrary spots from Spot, f and g stand for arbitrary fields from Field, a, b, c and d stand for arbitrary atomic objects from AtObj,

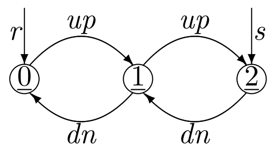

Figure 1. Graphical representation of a data linkage

and n and m stand for arbitrary values from Value.

In the examples given in this paper, we take the set $\{\underline{n} \mid n \in \{0, \dots, 9\}\}$ for AtObj.

Example 2.1. We consider the following closed DLA term:

$$((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn} \underline{0}) \oplus (\underline{1} \xrightarrow{up} \underline{2}) \oplus (\underline{2} \xrightarrow{dn} \underline{0}) \oplus (\xrightarrow{s} \underline{2})) \oplus' (\underline{2} \xrightarrow{dn} \underline{1}) \; .$$

Using the axioms of DLA, we can establish that this term denotes the same data linkage as the following term:

$$(\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn} \underline{0}) \oplus (\underline{1} \xrightarrow{up} \underline{2}) \oplus (\underline{2} \xrightarrow{dn} \underline{1}) \oplus (\xrightarrow{s} \underline{2}) \; .$$

The data linkage concerned is represented graphically in Figure 1.

All closed DLA terms are derivably equal to basic terms over DLA, i.e. closed DLA terms in which the data linkage overriding combination operator does not occur.

The set $\mathcal{B}$ of *basic terms* over DLA is inductively defined by the following rules:

- $\emptyset \in \mathcal{B}$;
- if $s \in \mathsf{Spot}$ and $a \in \mathsf{AtObj}$, then $\xrightarrow{s} a \in \mathcal{B}$;
- if $a \in \mathsf{AtObj}$ and $f \in \mathsf{Field}$, then $a \xrightarrow{f} \in \mathcal{B}$;
- if $a, b \in \mathsf{AtObj}$ and $f \in \mathsf{Field}$, then $a \xrightarrow{f} b \in \mathcal{B}$;
- if $a \in \mathsf{AtObj}$ and $n \in \mathsf{Value}$, then $(a)_n \in \mathcal{B}$;
- if $L_1, L_2 \in \mathcal{B}$, then $L_1 \oplus L_2 \in \mathcal{B}$.

Theorem 2.2. (Elimination)
For all closed DLA terms L, there exists a basic term $L' \in \mathcal{B}$ such that $L = L'$ is derivable from the axioms of DLA.

Proof:
This is easily proved by induction on the structure of L. In the case where $L \equiv L_1 \oplus' L_2$, we use the fact that for all basic terms $L'_1, L'_2 \in \mathcal{B}$, there exists a basic term $L'' \in \mathcal{B}$ such that $L'_1 \oplus' L'_2 = L''$ is derivable from the axioms of DLA. This is easily proved by induction on the structure of L'_2. □

We are only interested in the initial model of DLA. We write $\mathcal{DL}$ for the set of all elements of the initial model of DLA.

$\mathcal{DL}$ consists of the equivalence classes of basic terms over DLA with respect to the equivalence induced by the axioms of DLA. In other words, modulo equivalence, $\mathcal{B}$ is $\mathcal{DL}$. Henceforth, we will identify basic terms over DLA and their equivalence classes.

Let $L \in \mathcal{DL}$. Then L is *locally deterministic* with regard to a spot $s \in \mathsf{Spot}$ if the following holds:

- $L \oplus (\stackrel{s}{\to} a) = L$ for some $a \in \mathsf{AtObj}$;
- $L \oplus (\stackrel{s}{\to} a) = L \oplus (\stackrel{s}{\to} b)$ implies $a = b$ for all $a, b \in \mathsf{AtObj}$;

and L is *locally deterministic* with regard to a field $f \in \mathsf{Field}$ for an atomic object $a \in \mathsf{AtObj}$ if the following holds:

- $L \oplus (a \stackrel{f}{\to} b) = L$ for some $b \in \mathsf{AtObj}$ or $L \oplus (a \stackrel{f}{\to}) = L$;
- $L \oplus (a \stackrel{f}{\to} b) = L \oplus (a \stackrel{f}{\to} c)$ implies $b = c$ for all $b, c \in \mathsf{AtObj}$;
- $L \oplus (a \stackrel{f}{\to} b) \neq L \oplus (a \stackrel{f}{\to})$ for all $b \in \mathsf{AtObj}$;

and L is *locally deterministic* with regard to the value assignment for an atomic object $a \in \mathsf{AtObj}$ if the following holds:

- $L \oplus (a)_n = L$ for some $n \in \mathsf{Value}$;
- $L \oplus (a)_n = L \oplus (a)_m$ implies $n = m$ for all $n, m \in \mathsf{Value}$;

and L is *deterministic* if the following holds:

- for all $s \in \mathsf{Spot}$, L is locally deterministic with regard to s;
- for all $f \in \mathsf{Field}$ and all $a \in \mathsf{AtObj}$, L is locally deterministic with regard to f for a;
- for all $a \in \mathsf{AtObj}$, L is locally deterministic with regard to the value assignment for a.

A data linkage $L \in \mathcal{DL}$ is *non-deterministic* if it is not deterministic.

In Section 11, deterministic data linkages are represented by means of functions and data linkage overriding combination is modelled by means of function overriding.

3. Interlude: Priority Rewrite Systems

In Sections 4 and 5, we will present data linkage dynamics, a model of computation in which states of computations are modelled as data linkages and state changes take place by means of certain actions. We will describe the state changes and replies that result from performing these actions by means of a priority rewrite system. Therefore, we shortly review priority rewrite systems first. A comprehensive account of priority rewrite systems can be found in [2]. For convenience, the basic definitions and results regarding term rewriting systems are collected in an appendix.

A *priority rewrite system* is a pair $(\mathcal{R}, <)$, where $\mathcal{R}$ is a term rewriting system and $<$ is a partial order on the set of rewrite rules of $\mathcal{R}$.

Informally, the procedural meaning of the partial order on the set of rewrite rules of a priority rewrite system is that a term of the form $f(t_1, \ldots, t_n)$ is allowed to be rewritten according to some applicable rewrite rule only if it cannot be rewritten to a term $f(t'_1, \ldots, t'_n)$ to which a rewrite rule of a higher priority is applicable.

Let $(\mathcal{R}, <)$ be a priority rewrite system, and let R be a set of closed instances of rewrite rules of $\mathcal{R}$. Then an *R-reduction* is a reduction of $\mathcal{R}$ that belongs to the closure of R under closed contexts, transitivity and reflexivity. Let r be a rewrite rule. Then an *r-rewrite* is a closed substitution instance of r and an *r-redex* is the left-hand side of a substitution instance of r.

Let $(\mathcal{R}, <)$ be a priority rewrite system. Assume that there exists a unique set R of closed instances of rewrite rules of $\mathcal{R}$ such that an r-rewrite $t \to s \in R$ if there does not exist an R-reduction $t \twoheadrightarrow t'$ that leaves the head symbol of t unaffected and an r'-rewrite $t' \to s' \in R$ with $r < r'$. Then $(\mathcal{R}, <)$ determines a *one-step reduction relation* as follows: $\to$ is the closure of R under closed contexts. Moreover, let E be a set of equations between terms over the signature of $\mathcal{R}$. Then $(\mathcal{R}, <)$ determines a *one-step reduction relation modulo E* as follows: $t \to_E s$ if and only if $t' \to s'$ for some t' and s' such that $t = t'$ and $s = s'$ are derivable from E (where $\to$ denotes the one-step reduction relation determined by $(\mathcal{R}, <)$). If a unique R as described above exists, $(\mathcal{R}, <)$ is called *well-defined*. If a priority rewrite system is not well-defined, then it does not determine a one-step reduction relation.

Let $(\mathcal{R}, <)$ be a priority rewrite system. Then $\mathcal{R}$ is called the underlying term rewriting system of $(\mathcal{R}, <)$.

The priority rewrite system for data linkage dynamics is actually a many-sorted priority rewrite system. The definitions and results concerning term rewriting systems extend easily to the many-sorted case, see e.g. [14], and likewise for priority rewrite systems.

Equations can serve as rewrite rules. Taken as rewrite rules, equations are only used in the direction from left to right. In the priority rewrite system for data linkage dynamics, equations that serve as axioms of DLA are taken as rewrite rules.

Henceforth, all rewrite rules will be written as equations. Moreover, the notation

$$\begin{array}{ll} \overline{[n_1]} & r_1 \\ \vdots & \\ \underline{[n_k]} & r_k \end{array}$$

will be used in a table of rewrite rules to indicate that each of the rewrite rules $r_1, \ldots, r_k$ is incomparable with each of the other rewrite rules in the table and, for $i, j \in \{1, \ldots, k\}$, $r_i < r_j$ if and only if $n_i > n_j$.

4. The Kernel of Data Linkage Dynamics

DLD (Data Linkage Dynamics) is a simple model of computation that pertains to the use of dynamic data structures in programming. It comprises states, basic actions, i.e. indivisible actions, and the state changes and replies that result from performing the basic actions. The states of DLD are data linkages.

In connection with the value-related basic actions of DLD, it is assumed that a fixed but arbitrary model of a certain set of equations has been given and that the set Value consists of the elements of that algebra. A different set of equations would give rise to a variant of DLD that includes a large part of DLD, namely the part with its features to structure data dynamically. In this section, we introduce this part of DLD, called DLD-K (DLD Kernel). In Section 5, we will introduce the remaining part of DLD, i.e. the part with its features to deal with values found in dynamically structured data. Basic actions related to reclaiming garbage are treated separately in Section 7.

Like in DLA, it is assumed that a fixed but arbitrary finite set Spot of spots, a fixed but arbitrary finite set Field of fields, a fixed but arbitrary finite set AtObj of atomic objects, and a fixed but arbitrary set Value of values have been given. It is also assumed that a fixed but arbitrary *choice* function ch : $(\mathcal{P}(\mathsf{AtObj}) \setminus \emptyset) \to \mathsf{AtObj}$ such that, for all $A \in \mathcal{P}(\mathsf{AtObj}) \setminus \emptyset$, $ch(A) \in A$ has been given. The function ch is used whenever a fresh atomic object must be obtained.

By means of the basic actions of DLD-K, fresh atomic objects can be obtained, fields can be added to and removed from atomic objects, and the contents of fields of atomic objects can be examined and modified. A few basic actions use a spot to put an atomic object in or to get an atomic object from. The contents of spots can be compared and modified.

DLD-K has the following basic actions:

- for each $s \in \mathsf{Spot}$, a *get fresh atomic object action* $s\,!$;
- for each $s, t \in \mathsf{Spot}$, a *set spot action* $s = t$;
- for each $s \in \mathsf{Spot}$, a *clear spot action* $s = *$;
- for each $s, t \in \mathsf{Spot}$, an *equality test action* $s == t$;
- for each $s \in \mathsf{Spot}$, an *undefinedness test action* $s == *$;
- for each $s \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *add field action* s/f;
- for each $s \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *remove field action* $s\backslash f$;
- for each $s \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *has field action* $s\,|\,f$;
- for each $s, t \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *set field action* $s.f = t$;
- for each $s \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *clear field action* $s.f = *$;
- for each $s, t \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *get field action* $s = t.f$.

We write $A_{\mathrm{DLD\text{-}K}}$ for the set of all basic actions of DLD-K.

The intuition is that performing a basic action may cause a state change and will produce a reply. The possible replies are T (standing for true) and F (standing for false), and the actual reply is state-dependent. Basic actions intended for examining the state do not cause a state change, and basic actions intended for changing the state produce the reply F only if something precludes a state change.

When speaking informally about a state L of DLD-K, we say:

- if L is locally deterministic with regard to spot s, *the content of spot* s instead of the unique atomic object a for which $\xrightarrow{s} a$ is contained in L;

- if L is locally deterministic with regard to field f for atomic object a, *the content of field* f *of atomic object* a instead of the unique atomic object b for which $a \xrightarrow{f} b$ is contained in L;
- *the fields of atomic object* a instead of the set of all fields f such that either $a \xrightarrow{f}$ is contained in L or there exists an atomic object b such that $a \xrightarrow{f} b$ is contained in L.

The basic actions of DLD-K can be explained as follows if all spots and fields involved in performing them are spots and fields with regard to which the current state is locally deterministic:

- $s\,!$: if a fresh atomic object can be allocated, then the content of spot s becomes that fresh atomic object and the reply is T; otherwise, nothing changes and the reply is F;
- $s = t$: the content of spot s becomes the same as the content of spot t and the reply is T;
- $s = *$: the content of spot s becomes undefined and the reply is T;
- $s == t$: if the content of spot s equals the content of spot t, then nothing changes and the reply is T; otherwise, nothing changes and the reply is F;
- $s == *$: if the content of spot s is undefined, then nothing changes and the reply is T; otherwise, nothing changes and the reply is F;
- s/f: if the content of spot s is an atomic object and f does not yet belong to the fields of that atomic object, then f is added (with undefined content) to the fields of that atomic object and the reply is T; otherwise, nothing changes and the reply is F;
- $s\backslash f$: if the content of spot s is an atomic object and f belongs to the fields of that atomic object, then f is removed from the fields of that atomic object and the reply is T; otherwise, nothing changes and the reply is F;
- $s\,|\,f$: if the content of spot s is an atomic object and f belongs to the fields of that atomic object, then nothing changes and the reply is T; otherwise, nothing changes and the reply is F;
- $s.f = t$: if the content of spot s is an atomic object and f belongs to the fields of that atomic object, then the content of that field becomes the same as the content of spot t and the reply is T; otherwise, nothing changes and the reply is F;
- $s.f = *$: if the content of spot s is an atomic object and f belongs to the fields of that atomic object, then the content of that field becomes undefined and the reply is T; otherwise, nothing changes and the reply is F;
- $s = t.f$: if the content of spot t is an atomic object and f belongs to the fields of that atomic object, then the content of spot s becomes the same as the content of that field and the reply is T; otherwise, nothing changes and the reply is F.

In the explanation given above, wherever we say that the content of a spot or field becomes the same as the content of another spot or field, this is meant to imply that the former content becomes undefined if the latter content is undefined. If not all spots and fields involved in performing a basic action of DLD-K are spots and fields with regard to which the current state is locally deterministic, there is no state change and the reply is F.

The choice is made to deal uniformly with all cases in which not all spots and fields involved in performing a basic action of DLD-K are spots and fields with regard to which the current state is locally deterministic. However, if the field involved in performing a remove field action or a has field action is not a field with regard to which the current state is locally deterministic, there are other imaginable ways to deal with it. For example, the state change and reply could be the same as in the case where the field involved is a field with regard to which the current state is locally deterministic.

Recall that, in the examples given in this paper, we take the set $\{\underline{n} \mid n \in \{0,\dots,9\}\}$ for AtObj. Moreover, we take the choice function ch such that $ch(A) = \underline{n}$ if and only if $\underline{n} \in A$ and there does not exist an $n' < n$ such that $\underline{n'} \in A$.

Example 4.1. We consider the closed DLA term

$$(\overset{r}{\to}\underline{0}) \oplus (\underline{0} \overset{up}{\to} \underline{1}) \oplus (\underline{1} \overset{dn}{\to} \underline{0}) \oplus (\underline{1} \overset{up}{\to} \underline{2}) \oplus (\underline{2} \overset{dn}{\to} \underline{1}) \oplus (\overset{s}{\to}\underline{2})$$

from Example 2.1 again. The data linkage denoted by this term can be obtained from the empty data linkage by performing

- first $r\,!$ and $s = r$ in that order, yielding $(\overset{r}{\to}\underline{0}) \oplus (\overset{s}{\to}\underline{0})$;
- next $t = s$, $s\,!$, t/up, s/dn, $t.up = s$ and $s.dn = t$ in that order, yielding $(\overset{r}{\to}\underline{0}) \oplus (\overset{s}{\to}\underline{1}) \oplus (\overset{t}{\to}\underline{0}) \oplus (\underline{0} \overset{up}{\to} \underline{1}) \oplus (\underline{1} \overset{dn}{\to} \underline{0})$;
- then again $t = s$, $s\,!$, t/up, s/dn, $t.up = s$ and $s.dn = t$ in that order, yielding $(\overset{r}{\to}\underline{0}) \oplus (\overset{s}{\to}\underline{2}) \oplus (\overset{t}{\to}\underline{1}) \oplus (\underline{0} \overset{up}{\to} \underline{1}) \oplus (\underline{1} \overset{dn}{\to} \underline{0}) \oplus (\underline{1} \overset{up}{\to} \underline{2}) \oplus (\underline{2} \overset{dn}{\to} \underline{1})$;
- finally $t = *$.

The priority rewrite system for DLD-K given below is a many-sorted priority rewrite system. In addition to the sort $\mathbf{DL}$ of data linkages, it has the sort $\mathbf{R}$ of replies. Because this priority rewrite system is used to describe the state changes and replies that result from performing the basic actions of DLD-K, it has for each basic action $\alpha \in A_{\text{DLD-K}}$, the unary *effect* operator $\mathit{eff}_\alpha : \mathbf{DL} \to \mathbf{DL}$ and the unary *yield* operator $\mathit{yld}_\alpha : \mathbf{DL} \to \mathbf{R}$. The intuition is that these operators stand for operations that give, for each state L, the state and reply, respectively, that result from performing basic action α in state L. Moreover, the priority rewrite system has the following two constants of sort $\mathbf{R}$: T and F.

In the priority rewrite system for DLD-K given below, the function $atobj$ is used to restrict the basic terms over DLA for which the syntactical variable L stands. This function gives, for each basic term L over DLA, the set of atomic objects occurring in L. It is defined as follows:

$$\begin{aligned}
&atobj(\overset{s}{\to} a) = \{a\}\ , && atobj((a)_n) = \{a\}\ ,\\
&atobj(a \overset{f}{\to}) = \{a\}\ , && atobj(\emptyset) = \emptyset\ ,\\
&atobj(a \overset{f}{\to} b) = \{a, b\}\ , && atobj(L \oplus L') = atobj(L) \cup atobj(L')\ .
\end{aligned}$$

The priority rewrite system for DLD-K consists of the axioms of DLA, with the exception of the associativity, commutativity and identity axioms for $\oplus$, taken as rewrite rules and the rewrite rules for the effect and yield operators given in Table 2. In this table, L stands for an arbitrary basic term over

Table 2. Rewrite rules for effect and yield operators

[1]	$\mathit{eff}_{s!}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s!}(L) = L \oplus' (\xrightarrow{s} a)$ where $a = ch(\mathsf{AtObj} \setminus atobj(L))$	if $atobj(L) \subset \mathsf{AtObj}$
[2]	$\mathit{eff}_{s!}(L) = L$	if $atobj(L) = \mathsf{AtObj}$
[1]	$\mathit{eff}_{s=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s=t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s=t}(X \oplus (\xrightarrow{t} a)) = (X \oplus (\xrightarrow{t} a)) \oplus' (\xrightarrow{s} a)$	
[3]	$\mathit{eff}_{s=t}(X \oplus (\xrightarrow{s} a)) = \mathit{eff}_{s=t}(X)$	
[4]	$\mathit{eff}_{s=t}(X) = X$	
[1]	$\mathit{eff}_{s=*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s=*}(X \oplus (\xrightarrow{s} a)) = \mathit{eff}_{s=*}(X)$	
[3]	$\mathit{eff}_{s=*}(X) = X$	
[1]	$\mathit{eff}_{s==t}(X) = X$	
[1]	$\mathit{eff}_{s==*}(X) = X$	
[1]	$\mathit{eff}_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)$	
[2]	$\mathit{eff}_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})$	
[3]	$\mathit{eff}_{s/f}(X \oplus (\xrightarrow{s} a)) = (X \oplus (\xrightarrow{s} a)) \oplus' (a \xrightarrow{f})$	
[4]	$\mathit{eff}_{s/f}(X) = X$	
[1]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[2]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a))$	
[2]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a))$	
[3]	$\mathit{eff}_{s\backslash f}(X) = X$	
[1]	$\mathit{eff}_{s\mid f}(X) = X$	
[1]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[1]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (a \xrightarrow{f} c)) = (X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b)) \oplus' (a \xrightarrow{f} b)$	
[2]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (a \xrightarrow{f})) = (X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b)) \oplus' (a \xrightarrow{f} b)$	
[3]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = (X \oplus (\xrightarrow{s} a)) \oplus' (a \xrightarrow{f})$	
[4]	$\mathit{eff}_{s.f=t}(X) = X$	
[1]	$\mathit{eff}_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[2]	$\mathit{eff}_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = (X \oplus (\xrightarrow{s} a)) \oplus' (a \xrightarrow{f})$	
[3]	$\mathit{eff}_{s.f=*}(X) = X$	

Table 2. (Continued)

[1]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[2]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b)) = (X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b)) \oplus' (\xrightarrow{s} b)$	
[2]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f}) \oplus (\xrightarrow{s} c)) = \mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f}))$	
[3]	$\mathit{eff}_{s=t.f}(X) = X$	
[1]	$\mathit{yld}_{s\,!}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$\mathit{yld}_{s\,!}(L) = \mathsf{T}$	if $\mathit{atobj}(L) \subset \mathsf{AtObj}$
[2]	$\mathit{yld}_{s\,!}(L) = \mathsf{F}$	if $\mathit{atobj}(L) = \mathsf{AtObj}$
[1]	$\mathit{yld}_{s=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$\mathit{yld}_{s=t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[2]	$\mathit{yld}_{s=t}(X) = \mathsf{T}$	
[1]	$\mathit{yld}_{s=*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$\mathit{yld}_{s=*}(X) = \mathsf{T}$	
[1]	$\mathit{yld}_{s==t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$\mathit{yld}_{s==t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[2]	$\mathit{yld}_{s==t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} a)) = \mathsf{T}$	
[3]	$\mathit{yld}_{s==t}(X \oplus (\xrightarrow{s} a)) = \mathsf{F}$	
[3]	$\mathit{yld}_{s==t}(X \oplus (\xrightarrow{t} a)) = \mathsf{F}$	
[4]	$\mathit{yld}_{s==t}(X) = \mathsf{T}$	
[1]	$\mathit{yld}_{s==*}(X \oplus (\xrightarrow{s} a)) = \mathsf{F}$	
[2]	$\mathit{yld}_{s==*}(X) = \mathsf{T}$	
[1]	$\mathit{yld}_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$\mathit{yld}_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{F}$	
[2]	$\mathit{yld}_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[3]	$\mathit{yld}_{s/f}(X \oplus (\xrightarrow{s} a)) = \mathsf{T}$	
[4]	$\mathit{yld}_{s/f}(X) = \mathsf{F}$	
[1]	$\mathit{yld}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$\mathit{yld}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$\mathit{yld}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[2]	$\mathit{yld}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$\mathit{yld}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$\mathit{yld}_{s\backslash f}(X) = \mathsf{F}$	
[1]	$\mathit{yld}_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$\mathit{yld}_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$\mathit{yld}_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[2]	$\mathit{yld}_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$\mathit{yld}_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$\mathit{yld}_{s\,\vert\,f}(X) = \mathsf{F}$	

Table 2. (Continued)

$\overline{[1]}$	$yld_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$yld_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[1]	$yld_{s.f=t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$yld_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$yld_{s.f=t}(X) = \mathsf{F}$	
$\overline{[1]}$	$yld_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$yld_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[2]	$yld_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$yld_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$yld_{s.f=*}(X) = \mathsf{F}$	
$\overline{[1]}$	$yld_{s=t.f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$yld_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[2]	$yld_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$yld_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
$\underline{[3]}$	$yld_{s=t.f}(X) = \mathsf{F}$	

DLA, s and t stand for arbitrary spots from Spot, f stands for an arbitrary field from Field, and a, b and c stand for arbitrary atomic objects from AtObj. Each of the rewrite rules in Table 2 is incomparable with each of the axioms of DLA that are taken as rewrite rules. Moreover, the axioms of DLA that are taken as rewrite rules are mutually incomparable.

In Section 6, we will state some properties of the priority rewrite system for DLD. It is obvious from the proofs that the properties concerned are properties of the priority rewrite system for DLD-K as well. Among the latter properties is the well-definedness of the priority rewrite system for DLD-K. If it would not have this property, the priority rewrite system for DLD-K would not determine a one-step reduction relation.

Henceforth, we will write AC1 for the set of equations that consists of the associativity, commutativity and identity axioms for $\oplus$.[2] Because there are equal DLD-K terms that cannot be rewritten to the same term once the equations in AC1 are only used in one direction, reduction modulo AC1 is of importance to DLD-K. Thus, the one-step reduction relation of interest for DLD-K is the one-step reduction relation modulo AC1 determined by the priority rewrite system for DLD-K. We will write $\twoheadrightarrow_{\mathrm{AC1}}$ for the closure of this reduction relation under transitivity and reflexivity. Notice that AC1 does not contain the idempotency axiom for $\oplus$. This axiom is taken as rewrite rule because it is only needed in the direction from left to right.

[2]The mnemonic name AC1 for the associativity, commutativity and identity axioms for some operator is taken from [30].

Example 4.2. The statement that the data linkage denoted by

$$(\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn} \underline{0})$$

can be obtained from the empty data linkage by performing $r\,!$, $t\,!$, r/up, t/dn, $r.up = t$, $t.dn = r$ and $t = *$ in that order is substantiated by the priority rewrite system for DLD-K, where it is provable that

$$\begin{aligned}&\mathit{eff}_{t=*}(\mathit{eff}_{t.dn=r}(\mathit{eff}_{r.up=t}(\mathit{eff}_{t/dn}(\mathit{eff}_{r/up}(\mathit{eff}_{t\,!}(\mathit{eff}_{r\,!}(\emptyset))))))) \twoheadrightarrow_{\mathrm{AC1}}\\ &(\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn} \underline{0})\ .\end{aligned}$$

5. Data Linkage Dynamics

In this section, we extend DLD-K with features to deal with values found in dynamically structured data, resulting in DLD. That is, we add basic actions by means of which calculations can be done with values that are associated with atomic objects to the basic actions of DLD-K.

Unlike in DLD-K, it is assumed that a fixed but arbitrary finite meadow has been given and that Value consists of the elements of that meadow. A meadow is a commutative ring with a multiplicative identity element 1 and a total multiplicative inverse operation ${}^{-1}$ satisfying the reflexivity equation $(u^{-1})^{-1} = u$ and the restricted inverse equation $u \cdot (u \cdot u^{-1}) = u$. Thus, a meadow has an additive identity element 0, a multiplicative identity element 1, an addition operation $+$, a multiplication operation $\cdot$, an additive inverse operation $-$, and a multiplicative inverse operation ${}^{-1}$ that satisfies $0^{-1} = 0$. Meadows were defined for the first time in [16] and elaborated in several subsequent papers (see e.g. [5, 17]). The prime examples of finite meadows are finite fields with the multiplicative inverse operation made total by imposing that the multiplicative inverse of zero is zero.

DLD has the basic actions of DLD-K and in addition the following basic actions:

- for each $s \in \mathsf{Spot}$, an *assign zero action* $s <= 0$;
- for each $s \in \mathsf{Spot}$, an *assign one action* $s <= 1$;
- for each $s, t, u \in \mathsf{Spot}$, an *assign sum action* $s <= t + u$;
- for each $s, t, u \in \mathsf{Spot}$, an *assign product action* $s <= t \cdot u$;
- for each $s, t \in \mathsf{Spot}$, an *assign additive inverse action* $s <= -\,t$;
- for each $s, t \in \mathsf{Spot}$, an *assign multiplicative inverse action* $s <= 1\ /\ t$;
- for each $s, t \in \mathsf{Spot}$, a *value equality test action* $s =?\ t$;
- for each $s \in \mathsf{Spot}$, a *value undefinedness test action* $s =?\ *$.

We write A_{DLD} for the set of all basic actions of DLD.

When speaking informally about a state L of DLD, we also say:

- if L is locally deterministic with regard to the value assignment for a, *the value assigned to atomic object* a instead of the unique value n for which $(a)_n$ is contained in L;
- *atomic object* a *has a value assigned* instead of L is locally deterministic with regard to the value assignment for a.

The value-related basic actions of DLD can be explained as follows if all spots and value assignments involved in performing them are spots and value assignments with regard to which the current state is locally deterministic:

- $s <= 0$: if the content of spot s is an atomic object, then the value assigned to that atomic object becomes 0 and the reply is T; otherwise, nothing changes and the reply is F;
- $s <= 1$: if the content of spot s is an atomic object, then the value assigned to that atomic object becomes 1 and the reply is T; otherwise, nothing changes and the reply is F;
- $s <= t + u$: if the content of spot s is an atomic object and the contents of spots t and u are atomic objects that have values assigned, then the value assigned to the content of spot s becomes the sum of the values assigned to the contents of spots t and u and the reply is T; otherwise, nothing changes and the reply is F;
- $s <= t \cdot u$: if the content of spot s is an atomic object and the contents of spots t and u are atomic objects that have values assigned, then the value assigned to the content of spot s becomes the product of the values assigned to the contents of spots t and u and the reply is T; otherwise, nothing changes and the reply is F;
- $s <= -\,t$: if the content of spot s is an atomic object and the content of spot t is an atomic object that has a value assigned, then the value assigned to the content of spot s becomes the additive inverse of the value assigned to the content of spot t and the reply is T; otherwise, nothing changes and the reply is F;
- $s <= 1\,/\,t$: if the content of spot s is an atomic object and the content of spot t is an atomic object that has a value assigned, then the value assigned to the content of spot s becomes the multiplicative inverse of the value assigned to the content of spot t and the reply is T; otherwise, nothing changes and the reply is F;
- $s =?\, t$: if the contents of spots s and t are atomic objects that have values assigned and the value assigned to the content of spot s equals the value assigned to the content of spot t, then nothing changes and the reply is T; otherwise, nothing changes and the reply is F;
- $s =?\, *$: if the content of spot s is an atomic object that has no value assigned, then nothing changes and the reply is T; otherwise, nothing changes and the reply is F.

If not all spots and value assignments involved in performing a value-related basic action are spots and value assignments with regard to which the current state is locally deterministic, there is no state change and the reply is F.

Notice that copying, subtraction, and division can be done with the value-related basic actions available in DLD. If the content of spot s is an atomic object and the content of spot t is an atomic object that has a value assigned, then that value can be assigned to the content of spot s by first performing $s <= 0$ and then performing $s <= s + t$. If the content of spot s is an atomic object and the contents of spots t and u are atomic objects that have values assigned, then the difference of those values can be assigned to the content of spot s by first performing $u <= -\,u$, next performing $s <= t + u$ and then performing $u <= -\,u$ once again. Division can be done like subtraction.

Example 5.1. We consider the data linkage denoted by the following closed DLA term:

$$(\xrightarrow{s} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{1})_7 \oplus (\xrightarrow{u} \underline{2}) \oplus (\underline{2})_3 \;.$$

The data linkage obtained from this data linkage by first performing $u <= -\,u$, next performing $s <= t + u$ and then performing $u <= -\,u$ once again is the data linkage denoted by the term

$$(\xrightarrow{s} \underline{0}) \oplus (\underline{0})_4 \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{1})_7 \oplus (\xrightarrow{u} \underline{2}) \oplus (\underline{2})_3 \;.$$

In DLD, finite meadows are taken as the basis for the features to deal with values. This allows for calculations in finite fields. The approach followed is generic: take the algebras that are the models of some set of equational axioms and introduce value-related basic actions for the operations of those algebras.

The priority rewrite system for DLD consists of the axioms of DLA, with the exception of the associativity, commutativity and identity axioms for $\oplus$, taken as rewrite rules, the rewrite rules from the priority rewrite system for DLD-K, and the rewrite rules given in Table 3. In this table, s, t and u stand for arbitrary spots from Spot, a, b and c stand for arbitrary atomic objects from AtObj, and n and m stand for arbitrary values from Value. Each of the rewrite rules in Table 3 is incomparable with each of the axioms of DLA that are taken as rewrite rules and each of the rewrite rules from the priority rewrite system for DLD-K.

The total number of rewrite rules for DLD is quite large. This is fully attributable to the fact that DLD has 19 different kinds of basic actions. The number of rewrite rules for the effect operator eff_α for a basic action α is on average about 4, and the number of rewrite rules for the yield operator yld_α for a basic action α is on average about 5. Moreover, the fraction of the rewrite rules that deal with the case in which not all spots, fields and/or value assignments involved in performing the basic action concerned are ones with regard to which the data linkage concerned is locally deterministic is on average somewhat greater than $1/2$. Due to the uniform treatment of this case, the rules in question have the forms $\mathit{eff}_\alpha(L) = L$ and $\mathit{yld}_\alpha(L) = \mathsf{F}$. They are labelled by 1, i.e. they have the highest priority. The remaining rewrite rules reflect the informal explanations of the basic actions of DLD given before in a direct way.

Example 5.2. What is stated before about copying and subtraction with the value-related basic actions of DLD is substantiated by the priority rewrite system for DLD. Let $L = M \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n$ be a closed DLA term. Then there exists a basic term N over DLA such that

$$\begin{aligned} \mathit{eff}_{s<=s+t}(\mathit{eff}_{s<=0}(L)) &\twoheadrightarrow_{\mathrm{AC1}} N \;, \\ L \oplus' (a)_n &\twoheadrightarrow_{\mathrm{AC1}} N \;. \end{aligned}$$

In other words, by first performing $s <= 0$ and then performing $s <= s + t$, the value assigned to the content of spot s becomes the same as the value assigned to the content of spot t. Let $L' = M' \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m$ be a closed DLA term. Then there exists a basic term N' over DLA such that

$$\begin{aligned} \mathit{eff}_{u<=-\,u}(\mathit{eff}_{s<=t+u}(\mathit{eff}_{u<=-\,u}(L'))) &\twoheadrightarrow_{\mathrm{AC1}} N' \;, \\ L' \oplus' (a)_{n-m} &\twoheadrightarrow_{\mathrm{AC1}} N' \;. \end{aligned}$$

In other words, by first performing $u <= -\,u$, next performing $s <= t + u$ and then performing $u <= -\,u$ once again, the value assigned to the content of spot s becomes the difference of the values assigned to the contents of spots t and u.

Table 3. Rewrite rules for additional effect and yield operators

$\overline{[1]}$	$\mathit{eff}_{s<=0}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} a)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=0}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[2]	$\mathit{eff}_{s<=0}(X \oplus (\xrightarrow{s} a)) = (X \oplus (\xrightarrow{s} a)) \oplus' (a)_0$	
$\underline{[3]}$	$\mathit{eff}_{s<=0}(X) = X$	
$\overline{[1]}$	$\mathit{eff}_{s<=1}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} a)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=1}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[2]	$\mathit{eff}_{s<=1}(X \oplus (\xrightarrow{s} a)) = (X \oplus (\xrightarrow{s} a)) \oplus' (a)_1$	
$\underline{[3]}$	$\mathit{eff}_{s<=1}(X) = X$	
$\overline{[1]}$	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[1]	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[1]	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{u} a) \oplus (\xrightarrow{u} b)) = X \oplus (\xrightarrow{u} a) \oplus (\xrightarrow{u} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{u} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{u} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[2]	$\mathit{eff}_{s<=t+u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m) =$ $(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m) \oplus' (a)_{n+m}$	
$\underline{[3]}$	$\mathit{eff}_{s<=t+u}(X) = X$	
$\overline{[1]}$	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[1]	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[1]	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{u} a) \oplus (\xrightarrow{u} b)) = X \oplus (\xrightarrow{u} a) \oplus (\xrightarrow{u} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{u} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{u} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[2]	$\mathit{eff}_{s<=t \cdot u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m) =$ $(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m) \oplus' (a)_{n \cdot m}$	
$\underline{[3]}$	$\mathit{eff}_{s<=t \cdot u}(X) = X$	
$\overline{[1]}$	$\mathit{eff}_{s<=-\,t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=-\,t}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[1]	$\mathit{eff}_{s<=-\,t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=-\,t}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[2]	$\mathit{eff}_{s<=-\,t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n) = (X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n) \oplus' (a)_{-n}$	
$\underline{[3]}$	$\mathit{eff}_{s<=-\,t}(X) = X$	
$\overline{[1]}$	$\mathit{eff}_{s<=1/t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=1/t}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[1]	$\mathit{eff}_{s<=1/t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s<=1/t}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m$	if $n \neq m$
[2]	$\mathit{eff}_{s<=1/t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n) = (X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n) \oplus' (a)_{n^{-1}}$	
$\underline{[3]}$	$\mathit{eff}_{s<=1/t}(X) = X$	
$\underline{\overline{[1]}}$	$\mathit{eff}_{s=?t}(X) = X$	
$\underline{\overline{[1]}}$	$\mathit{eff}_{s=?*}(X) = X$	

Table 3. (Continued)

[1]	$yld_{s<=0}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=0}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s<=0}(X \oplus (\xrightarrow{s} a)) = \mathsf{T}$	
[3]	$yld_{s<=0}(X) = \mathsf{F}$	
[1]	$yld_{s<=1}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=1}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s<=1}(X \oplus (\xrightarrow{s} a)) = \mathsf{T}$	
[3]	$yld_{s<=1}(X) = \mathsf{F}$	
[1]	$yld_{s<=t+u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=t+u}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s<=t+u}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=t+u}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s<=t+u}(X \oplus (\xrightarrow{u} a) \oplus (\xrightarrow{u} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=t+u}(X \oplus (\xrightarrow{u} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s<=t+u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m) = \mathsf{T}$	
[3]	$yld_{s<=t+u}(X) = \mathsf{F}$	
[1]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{u} a) \oplus (\xrightarrow{u} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{u} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s<=t\cdot u}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n \oplus (\xrightarrow{u} c) \oplus (c)_m) = \mathsf{T}$	
[3]	$yld_{s<=t\cdot u}(X) = \mathsf{F}$	
[1]	$yld_{s<=-\,t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=-\,t}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s<=-\,t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=-\,t}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s<=-\,t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n) = \mathsf{T}$	
[3]	$yld_{s<=-\,t}(X) = \mathsf{F}$	
[1]	$yld_{s<=1/t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=1/t}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s<=1/t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s<=1/t}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s<=1/t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} b) \oplus (b)_n) = \mathsf{T}$	
[3]	$yld_{s<=1/t}(X) = \mathsf{F}$	
[1]	$yld_{s=?t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s=?t}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[1]	$yld_{s=?t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s=?t}(X \oplus (\xrightarrow{t} a) \oplus (a)_n \oplus (a)_m) = \mathsf{F}$	if $n \neq m$
[2]	$yld_{s=?t}(X \oplus (\xrightarrow{s} a) \oplus (a)_n \oplus (\xrightarrow{t} b) \oplus (b)_n) = \mathsf{T}$	
[3]	$yld_{s=?t}(X) = \mathsf{F}$	

Table 3. (Continued)

[1]	$yld_{s=?*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s=?*}(X \oplus (\xrightarrow{s} a) \oplus (a)_n) = \mathsf{F}$	
[3]	$yld_{s=?*}(X \oplus (\xrightarrow{s} a)) = \mathsf{T}$	
[4]	$yld_{s=?*}(X) = \mathsf{F}$	

It is easy to check that for all $\alpha \in A_{\mathrm{DLD}}$, for all closed DLA terms L and L' such that $\mathit{eff}_\alpha(L) \twoheadrightarrow_{\mathrm{AC1}} L'$, L is deterministic if and only if L' is deterministic. In other words, both determinism and non-determinism are properties of data linkages that are preserved by the basic actions of DLD. Because of this, the data linkage obtained from another data linkage by performing a number of basic actions of DLD in succession is deterministic if and only if the latter data linkage is determistic.

With much effort, a few applications of non-deterministic data linkages could be thought up, but these applications would require a variant of DLD with basic actions of which we have not yet a clear image. This raises the question why DLA and DLD cover non-deterministic data linkages. The answer is that we have devised several variants of the pair of DLA and DLD that are restricted to deterministic data linkages, but they turned out to be more complicated than the pair of DLA and DLD.

The priority rewrite system for DLD is used in Section 10 in examples concerning computations in which the basic actions of DLD are involved.

6. Properties of the Priority Rewrite System for DLD

In this section, we state some properties of the priority rewrite system for DLD. For the purpose of stating the properties in question rigorously, we introduce the set $\mathcal{E}$ of *effect terms* and the set $\mathcal{Y}$ of *yield terms*. They are inductively defined by the following rules:

- $\emptyset \in \mathcal{E}$;
- if $s \in \mathsf{Spot}$ and $a \in \mathsf{AtObj}$, then $\xrightarrow{s} a \in \mathcal{E}$;
- if $a \in \mathsf{AtObj}$ and $f \in \mathsf{Field}$, then $a \xrightarrow{f} \in \mathcal{E}$;
- if $a, b \in \mathsf{AtObj}$ and $f \in \mathsf{Field}$, then $a \xrightarrow{f} b \in \mathcal{E}$;
- if $a \in \mathsf{AtObj}$ and $n \in \mathsf{Value}$, then $(a)_n \in \mathcal{E}$;
- if $D_1, D_2 \in \mathcal{E}$, then $D_1 \oplus D_2 \in \mathcal{E}$;
- if $D_1, D_2 \in \mathcal{E}$, then $D_1 \oplus' D_2 \in \mathcal{E}$;
- if $\alpha \in A_{\mathrm{DLD}}$ and $D \in \mathcal{E}$, then $\mathit{eff}_\alpha(D) \in \mathcal{E}$;
- if $\alpha \in A_{\mathrm{DLD}}$ and $D \in \mathcal{E}$, then $yld_\alpha(D) \in \mathcal{Y}$.

Clearly, $\mathcal{B}$ is a proper subset of $\mathcal{E}$. Below we will prove that effect terms have normal forms that are basic terms over DLA.

As a preparation, we state an important property of the underlying term rewriting system of the priority rewrite system for DLD.

Proposition 6.1. (Strongly normalizing underlying term rewriting system)
The underlying term rewriting system of the priority rewrite system for DLD is strongly normalizing modulo AC1.

Proof:
We will write $\mathcal{R}'_{\mathrm{DLD}}$ for the underlying term rewriting system of the priority rewrite system for DLD, we will write AC for the set of equations that consists of the associativity and commutativity axioms for $\oplus$, and we will write $\to'_{\mathrm{AC}}$ and $\to'_{\mathrm{AC1}}$ for the one-step reduction modulo AC relation of $\mathcal{R}'_{\mathrm{DLD}}$ and the one-step reduction modulo AC1 relation of $\mathcal{R}'_{\mathrm{DLD}}$, respectively.

First, it is proved that $\mathcal{R}'_{\mathrm{DLD}}$ is strongly normalizing modulo AC. This is easily proved by means of the reduction ordering induced by the integer polynomials $\phi(D)$ associated with DLD terms D as follows:

$$
\begin{array}{lll}
\phi(X) = \underline{X}\;, & \phi(\xrightarrow{s} a) = 3\;, & \phi(\emptyset) = 2\;,\\
 & \phi(a \xrightarrow{f}) = 3\;, & \phi(D_1 \oplus D_2) = \phi(D_1) + \phi(D_2) + 1\;,\\
 & \phi(a \xrightarrow{f} b) = 3\;, & \phi(D_1 \oplus' D_2) = \phi(D_1) \cdot \phi(D_2)\;,\\
 & \phi((a)_n) = 3\;, & \phi(\mathit{eff}_\alpha(D)) = 4 \cdot \phi(D)\;,\\
 & \phi(\mathsf{T}) = 3\;, & \phi(\mathit{yld}_\alpha(D)) = 4 \cdot \phi(D)\;,\\
 & \phi(\mathsf{F}) = 3\;, &
\end{array}
$$

where it is assumed that, for each variable X over data linkages, there is a variable $\underline{X}$ over integers. Here, it is crucial that $\phi(D_1 \oplus D_2) = \phi(D_2 \oplus D_1)$ and $\phi(D_1 \oplus (D_2 \oplus D_3)) = \phi((D_1 \oplus D_2) \oplus D_3)$ for all DLD terms D_1, D_2, D_3 (see e.g. Section 6.2.3 from [47]).

Next, it is proved by means of a function θ on DLD terms that $\mathcal{R}'_{\mathrm{DLD}}$ is strongly normalizing modulo AC1. The function θ, which transforms DLD terms to ones for which the applicable reduction modulo AC1 steps of $\mathcal{R}'_{\mathrm{DLD}}$ do not depend on the identity axiom for $\oplus$, is defined by $\theta(D) = \theta_1(\theta_0(D))$, where

$$
\begin{array}{lll}
\theta_0(X) = X\;, & \theta_0(\xrightarrow{s} a) = \xrightarrow{s} a\;, & \theta_0(\emptyset) = \emptyset\;,\\
 & \theta_0(a \xrightarrow{f}) = a \xrightarrow{f}\;, & \theta_0(\emptyset \oplus D) = \theta_0(D)\;,\\
 & \theta_0(a \xrightarrow{f} b) = a \xrightarrow{f} b\;, & \theta_0(D \oplus \emptyset) = \theta_0(D)\;,\\
 & \theta_0((a)_n) = (a)_n\;, & \theta_0(D_1 \oplus D_2) = \theta_0(D_1) \oplus \theta_0(D_2) \;\; \text{if } D_1 \not\equiv \emptyset \land D_2 \not\equiv \emptyset\;,\\
 & & \theta_0(D_1 \oplus' D_2) = \theta_0(D_1) \oplus' \theta_0(D_2)\;,\\
 & & \theta_0(\mathit{eff}_\alpha(D)) = \mathit{eff}_\alpha(\theta_0(D))\;,\\
 & \theta_0(\mathsf{T}) = \mathsf{T}\;, & \theta_0(\mathit{yld}_\alpha(D)) = \mathit{yld}_\alpha(\theta_0(D))\;,\\
 & \theta_0(\mathsf{F}) = \mathsf{F}\;, &
\end{array}
$$

$$
\begin{array}{lll}
\theta_1(X) = X \;, & \theta_1(\xrightarrow{s} a) = \xrightarrow{s} a \;, & \theta_1(\emptyset) = \emptyset \;, \\
 & \theta_1(a \xrightarrow{f}) = a \xrightarrow{f} \;, & \theta_1(D_1 \oplus D_2) = \theta_1(D_1) \oplus \theta_1(D_2) \;, \\
 & \theta_1(a \xrightarrow{f} b) = a \xrightarrow{f} b \;, & \theta_1(D_1 \oplus' D_2) = (\emptyset \oplus \theta_1(D_1)) \oplus' \theta_1(D_2) \;, \\
 & \theta_1((a)_n) = (a)_n \;, & \theta_1(\mathit{eff}_\alpha(D)) = \mathit{eff}_\alpha(\emptyset \oplus \theta_1(D)) \;, \\
 & \theta_1(\mathsf{T}) = \mathsf{T} \;, & \theta_1(\mathit{yld}_\alpha(D)) = \mathit{yld}_\alpha(\emptyset \oplus \theta_1(D)) \;, \\
 & \theta_1(\mathsf{F}) = \mathsf{F} \;. &
\end{array}
$$

By checking all rewrite rules, it is easily established that $t \to'_{\mathrm{AC1}} s$ only if $\theta(t) \to'_{\mathrm{AC}} \theta(s)$. From this it follows that, for each reduction sequence with respect to $\to'_{\mathrm{AC1}}$, the sequence obtained by replacing each term t in the reduction sequence by $\theta(t)$ is a reduction sequence with respect to $\to'_{\mathrm{AC}}$. Now assume that $\mathcal{R}'_{\mathrm{DLD}}$ is not strongly normalizing modulo AC1. Then there exists an infinite reduction sequence with respect to $\to'_{\mathrm{AC1}}$. Consequently, there exists an infinite reduction sequence with respect to $\to'_{\mathrm{AC}}$ as well. In other words, $\mathcal{R}'_{\mathrm{DLD}}$ is not strongly normalizing modulo AC. However, the contrary was proved above. Hence, $\mathcal{R}'_{\mathrm{DLD}}$ is strongly normalizing modulo AC1. □

A consequence of Theorem 3.11, Proposition 3.13, and Proposition 3.14 from [2] is that a priority rewrite system is well-defined if its underlying term rewriting system is strongly normalizing. Moreover, a term rewriting system is strongly normalizing modulo AC1 only if it is strongly normalizing as well. So we have an important corollary of Proposition 6.1.

Corollary 6.2. (Well-definedness)
The priority rewrite system for DLD is well-defined.

If it would not be well-defined, the priority rewrite system for DLD would not determine a one-step reduction relation and the following theorem would not make sense.

Theorem 6.3. (Normal forms)
The priority rewrite system for DLD has the following properties concerning normal forms with respect to reduction modulo AC1:

1. each element of $\mathcal{E}$ has a unique normal form modulo AC1;
2. the normal forms of the elements of $\mathcal{E}$ are exactly the basic terms over DLA;
3. each element of $\mathcal{Y}$ has a unique normal form;
4. the normal forms of the elements of $\mathcal{Y}$ are exactly the constants of sort $\mathbf{R}$.

Proof:
Properties 1 and 3 are proved combined. It follows immediately from Proposition 6.1 that the priority rewrite system for DLD is strongly normalizing modulo AC1 on all closed DLD terms. Therefore, it remains to be shown that the priority rewrite system for DLD is weakly confluent modulo AC1 on all closed DLD terms.

In this weak confluence proof, we use the *one-step equality relation* $\vdash\!\dashv$. This relation is defined as the closure of the set of all closed instances of the equations in AC1 under symmetry and closed contexts. A critical pair $(t \to s, t \to s')$ in which t, s, and s' are closed terms is called a *closed critical pair*.

The following holds for the priority rewrite system for DLD:

1. $r' < r$ if and only if the left hand side of r is a substitution instance of the left hand side of r';
2. for all closed critical pairs $(t \to s, t \to s')$ that arise from overlap modulo AC1 of a rewrite rule on an incomparable rewrite rule, s and s' have a common reduct modulo AC1;
3. for all closed critical pairs $(t \to s, t \vdash\!\dashv s')$ that arise from overlap modulo AC1 of a rewrite rule on an equation from AC1, there exists a one-step reduction $s' \to_{\mathrm{AC1}} s''$ that consists of the contraction of a redex modulo AC1 such that s and s'' have a common reduct modulo AC1;
4. overlaps between comparable rewrite rules are overlaps at the outermost positions only.

From this, the weak confluence modulo AC1 of the priority rewrite system for DLD on all closed DLD terms follows straightforwardly, following a line of reasoning similar to the one followed in the proof of Theorem 4.8 from [2], using Theorems 5 and 16 from [29]. The proof of Theorem 4.8 from [2] is concerned with showing that the one-step reduction relation determined by a priority rewrite system does not give rise to critical pairs while the current proof is concerned with showing that the one-step reduction relation determined by the priority rewrite system of DLD does not give rise to critical pairs that are not convergent. We use Theorems 5 and 16 from [29] because we consider reduction modulo AC1. These theorems allow for confluence modulo a set of equations to be established by convergence checks on critical pairs. In the use of the second theorem, it suffices to let $L = \emptyset$. Because we can add to point 3 above that the redex concerned occurs 'less deep than the overlap', this theorem can be used in spite of the fact that AC1-equivalence classes are infinite (see the second remark following the proof of the theorem).

Properties 2 and 4 are easily proved combined by structural induction. □

In Table 2, L stands for an arbitrary basic term over DLA. This means that each rewrite rule schema in which L occurs represents an infinite number of rewrite rules. We have a corollary of Theorem 6.3 which is relevant to this point because, modulo AC1, the number of basic terms over DLA is finite.

Corollary 6.4. (Equivalent priority rewrite system)
Let $\sim$ be the equivalence relation on $\mathcal{B}$ induced by AC1, let $repr : \mathcal{B}/{\sim} \to \mathcal{B}$ be such that $repr(\mathcal{B}') \in \mathcal{B}'$ for all $\mathcal{B}' \in \mathcal{B}/{\sim}$, and let $\mathcal{B}^*$ be the image of $\mathcal{B}/{\sim}$ under $repr$. Take the priority rewrite system obtained from the priority rewrite system for DLD by restricting the basic terms over DLA that L stands for to the elements of $\mathcal{B}^*$. This adapted priority rewrite system determines the same one-step reduction relation modulo AC1 as the priority rewrite system of DLD.

Corollary 6.4 is material to Corollary 6.6.

Let a priority rewrite system $(\mathcal{R}, <)$ be given. Let r be a rewrite rule from $\mathcal{R}$, and let $t \to s$ be an r-rewrite. Then r is *enabled* for t if $t \to s$ belongs to the one-step reduction relation determined by $(\mathcal{R}, <)$.

Proposition 6.5. (Enabled rewrite rules)
Let r be a rewrite rule from the priority rewrite system for DLD, and let D be a closed r-redex. Then

1. if $D \not\equiv \mathit{eff}_\alpha(D')$ and $D \not\equiv \mathit{yld}_\alpha(D')$ for all $\alpha \in A_{\mathrm{DLD}}$ and $D' \in \mathcal{E}$, then r is enabled for D;
2. if $D \equiv \mathit{eff}_\alpha(D')$ or $D \equiv \mathit{yld}_\alpha(D')$ for some $\alpha \in A_{\mathrm{DLD}}$ and $D' \in \mathcal{E}$, then r is enabled for D if and only if D is not a closed r'-redex for some rewrite rule r' with $r < r'$.

Proof:
This follows immediately from the definition of the one-step reduction relation determined by a priority rewrite system and the priority rewrite system for DLD. □

We have an important corollary of Theorem 6.3, Corollary 6.4, and Proposition 6.5.

Corollary 6.6. (Decidability of reduction relation)
The one-step reduction modulo AC1 relation determined by the priority rewrite system for DLD is decidable.

7. Reclaiming Garbage in Data Linkage Dynamics

Atomic objects that are not reachable via spots and fields can be reclaimed. Reclamation of unreachable atomic objects is relevant because the set AtObj of atomic objects is finite. There are various ways to achieve reclamation of unreachable atomic objects. In this section, we introduce some of them.

Data linkage dynamics has the following reclamation-related actions:

- a *full garbage collection action* fgc;
- a *restricted garbage collection action* rgc;
- for each $s \in \mathsf{Spot}$, a *get fresh atomic object action with safe disposal* $s \setminus !$;
- for each $s, t \in \mathsf{Spot}$, a *set spot action with safe disposal* $s \setminus = t$;
- for each $s \in \mathsf{Spot}$, a *clear spot action with safe disposal* $s \setminus = *$;
- for each $s, t \in \mathsf{Spot}$, $f \in \mathsf{Field}$, a *set field action with safe disposal* $s.f \setminus = t$;
- for each $s \in \mathsf{Spot}$, $f \in \mathsf{Field}$, a *clear field action with safe disposal* $s.f \setminus = *$;
- for each $s, t \in \mathsf{Spot}$, $f \in \mathsf{Field}$, a *get field action with safe disposal* $s \setminus = t.f$;
- for each $s \in \mathsf{Spot}$, a *get fresh atomic object action with unsafe disposal* $s \setminus\setminus !$;
- for each $s, t \in \mathsf{Spot}$, a *set spot action with unsafe disposal* $s \setminus\setminus = t$;
- for each $s \in \mathsf{Spot}$, a *clear spot action with unsafe disposal* $s \setminus\setminus = *$;
- for each $s, t \in \mathsf{Spot}$, $f \in \mathsf{Field}$, a *set field action with unsafe disposal* $s.f \setminus\setminus = t$;
- for each $s \in \mathsf{Spot}$, $f \in \mathsf{Field}$, a *clear field action with unsafe disposal* $s.f \setminus\setminus = *$;
- for each $s, t \in \mathsf{Spot}$, $f \in \mathsf{Field}$, a *get field action with unsafe disposal* $s \setminus\setminus = t.f$.

These reclamation-related actions of DLD can be explained as follows if all spots and fields involved in performing them are spots and fields with regard to which the current state is locally deterministic:

- fgc: all unreachable atomic objects are reclaimed, and the reply is T;
- rgc: all unreachable atomic objects that do not occur in a cycle are reclaimed, and the reply is T;

- $s \backslash !$, $s \backslash = t$, $s \backslash = *$, $s.f \backslash = t$, $s.f \backslash = *$, and $s \backslash = t.f$: like $s\,!$, $s = t$, $s = *$, $s.f = t$, $s.f = *$, and $s = t.f$, but followed by the reclamation of the old content of the spot or field whose content has been replaced if it has become an unreachable atomic object;
- $s \backslash\backslash !$, $s \backslash\backslash = t$, $s \backslash\backslash = *$, $s.f \backslash\backslash = t$, $s.f \backslash\backslash = *$, and $s \backslash\backslash = t.f$: like $s\,!$, $s = t$, $s = *$, $s.f = t$, $s.f = *$, and $s = t.f$, but followed by the reclamation of the old content of the spot or field whose content has been replaced after the content of everything containing it has been made undefined.

If not all spots and fields involved in performing a reclamation-related action are spots and fields with regard to which the current state is locally deterministic, there is no state change and the reply is F.

Full or restricted garbage collection can be made automatic by treating each $s\,!$ as if it is fgc or rgc followed by $s\,!$.

Garbage collection originates from programming languages that support the use of dynamic data structures. It is not only found in contemporary object-oriented programming languages such as Java [1, 24] and C# [25, 26], but also in historic programming languages such as LISP [35, 36]. In [35], the term reclamation is used instead of garbage collection.

The rewrite rules for full garbage collection and restricted garbage collection are given in Table 4. In this table, s stands for an arbitrary spot from Spot, f and g stand for arbitrary fields from Field, a, b, c and d stand for arbitrary atomic objects from AtObj, and n stand for an arbitrary value from Value.

The operators $\mathit{eff}_{\mathsf{fgc}}$ and $\mathit{eff}_{\mathsf{rgc}}$ are described using the auxiliary operators fgc and rgc, respectively. We mention that in $\mathit{fgc}(L, L')$:

- $L \oplus L'$ is the data linkage on which full garbage collection is carried out;
- all atomic objects found in L are already known to be reachable;
- if all atomic objects found in L' are unreachable, then L is the result of full garbage collection on $L \oplus L'$.

We mention that in $\mathit{rgc}(L, L')$:

- $L \oplus L'$ is the data linkage on which removal of all links from and value associations with atomic objects that have a reference count equal to zero is carried out repeatedly until this is no longer possible (the reference count of an atomic object is the number of links to that atomic object);
- all atomic objects found in L are already known to have a reference count greater than zero;
- if all atomic objects found in L' have a reference count equal to zero, then L is the result of removing all links from and value associations with atomic objects that have a reference count equal to zero from $L \oplus L'$;
- atomic objects found in L that have a reference count greater than zero in $L \oplus L'$ may have a reference count equal to zero in L.

It is striking that the description of restricted garbage collection by means of rewrite rules with priorities is more complicated than the description of full garbage collection by means of rewrite rules with priorities.

The rewrite rules for safe disposal and unsafe disposal are given in Table 5. In this table, s and t stand for arbitrary spots from Spot, f and g stand for arbitrary fields from Field, a, b, c and d stand for arbitrary atomic objects from AtObj, and n stand for an arbitrary value from Value.

Table 4. Rewrite rules for full and restricted garbage collection

[1]	$\mathit{eff}_{\mathsf{fgc}}(X) = \mathit{fgc}(\emptyset, X)$
[1]	$\mathit{eff}_{\mathsf{rgc}}(X) = \mathit{rgc}(\emptyset, X)$
[1]	$\mathit{yld}_{\mathsf{fgc}}(X) = \mathsf{T}$
[1]	$\mathit{yld}_{\mathsf{rgc}}(X) = \mathsf{T}$
[1]	$\mathit{fgc}(X, (\xrightarrow{s} a) \oplus Y) = \mathit{fgc}(X \oplus (\xrightarrow{s} a), Y)$
[1]	$\mathit{fgc}(X \oplus (\xrightarrow{s} a), (a \xrightarrow{f}) \oplus Y) = \mathit{fgc}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f}), Y)$
[1]	$\mathit{fgc}(X \oplus (a \xrightarrow{f} b), (b \xrightarrow{g}) \oplus Y) = \mathit{fgc}(X \oplus (a \xrightarrow{f} b) \oplus (b \xrightarrow{g}), Y)$
[1]	$\mathit{fgc}(X \oplus (\xrightarrow{s} a), (a \xrightarrow{f} b) \oplus Y) = \mathit{fgc}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b), Y)$
[1]	$\mathit{fgc}(X \oplus (a \xrightarrow{f} b), (b \xrightarrow{g} c) \oplus Y) = \mathit{fgc}(X \oplus (a \xrightarrow{f} b) \oplus (b \xrightarrow{g} c), Y)$
[1]	$\mathit{fgc}(X \oplus (\xrightarrow{s} a), (a)_n \oplus Y) = \mathit{fgc}(X \oplus (\xrightarrow{s} a) \oplus (a)_n, Y)$
[1]	$\mathit{fgc}(X \oplus (a \xrightarrow{f} b), (b)_n \oplus Y) = \mathit{fgc}(X \oplus (a \xrightarrow{f} b) \oplus (b)_n, Y)$
[2]	$\mathit{fgc}(X, Y) = X$
[1]	$\mathit{rgc}(X, (\xrightarrow{s} a) \oplus Y) = \mathit{rgc}(X \oplus (\xrightarrow{s} a), Y)$
[1]	$\mathit{rgc}(X \oplus (\xrightarrow{s} a), (a \xrightarrow{f}) \oplus Y) = \mathit{rgc}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f}), Y)$
[1]	$\mathit{rgc}(X \oplus (a \xrightarrow{f} b), (b \xrightarrow{g}) \oplus Y) = \mathit{rgc}(X \oplus (a \xrightarrow{f} b) \oplus (b \xrightarrow{g}), Y)$
[1]	$\mathit{rgc}(X, (a \xrightarrow{f} b) \oplus (b \xrightarrow{g}) \oplus Y) = \mathit{rgc}(X \oplus (b \xrightarrow{g}), (a \xrightarrow{f} b) \oplus Y)$
[1]	$\mathit{rgc}(X \oplus (\xrightarrow{s} a), (a \xrightarrow{f} b) \oplus Y) = \mathit{rgc}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b), Y)$
[1]	$\mathit{rgc}(X \oplus (a \xrightarrow{f} b), (b \xrightarrow{g} c) \oplus Y) = \mathit{rgc}(X \oplus (a \xrightarrow{f} b) \oplus (b \xrightarrow{g} c), Y)$
[1]	$\mathit{rgc}(X, (a \xrightarrow{f} b) \oplus (b \xrightarrow{g} c) \oplus Y) = \mathit{rgc}(X \oplus (b \xrightarrow{g} c), (a \xrightarrow{f} b) \oplus Y)$
[1]	$\mathit{rgc}(X \oplus (\xrightarrow{s} a), (a)_n \oplus Y) = \mathit{rgc}(X \oplus (\xrightarrow{s} a) \oplus (a)_n, Y)$
[1]	$\mathit{rgc}(X \oplus (a \xrightarrow{f} b), (b)_n \oplus Y) = \mathit{rgc}(X \oplus (a \xrightarrow{f} b) \oplus (b)_n, Y)$
[1]	$\mathit{rgc}(X, (a \xrightarrow{f} b) \oplus (b)_n \oplus Y) = \mathit{rgc}(X \oplus (b)_n, (a \xrightarrow{f} b) \oplus Y)$
[1]	$\mathit{rgc}(X, \emptyset) = X$
[2]	$\mathit{rgc}(X, Y) = \mathit{rgc}(\emptyset, X)$

The operators $\mathit{eff}_{s\backslash!}$, $\mathit{eff}_{s\backslash=t}$, $\mathit{eff}_{s\backslash=*}$, $\mathit{eff}_{s.f\backslash=t}$, $\mathit{eff}_{s.f\backslash=*}$, $\mathit{eff}_{s\backslash=t.f}$, $\mathit{eff}_{s\backslash\backslash!}$, $\mathit{eff}_{s\backslash\backslash=t}$, $\mathit{eff}_{s\backslash\backslash=*}$, $\mathit{eff}_{s.f\backslash\backslash=t}$, $\mathit{eff}_{s.f\backslash\backslash=*}$ and $\mathit{eff}_{s\backslash\backslash=t.f}$ are described using auxiliary operators disp_a and clr_a ($a \in \mathsf{AtObj}$). Indeed, disp_a deals with safe disposal of atomic object a and clr_a deals with making the content of everything containing a undefined. Carrying out safe disposal of a on $\mathit{clr}_a(L)$ amounts to the same thing as carrying out unsafe disposal of a on L. We mention that in $\mathit{disp}_a(L, L')$:

- $L \oplus L'$ is the data linkage on which safe disposal of a is carried out;
- all atomic objects found in L are already known not to be involved in the safe disposal of a;
- links and value associations are moved from L' to L in stages as follows:
  - first, all links and value associations that make up the reachable part of $L \oplus L'$ are moved from L' to L,
  - next, all links to a and value associations with a are moved from L' to L if a is found in L,
  - finally, all links and value associations in which a is not involved are moved from L' to L;

Table 5. Rewrite rules for safe and unsafe disposal

[1]	$\mathit{eff}_{s\,\backslash !}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{eff}_{s\,!}(X \oplus (\xrightarrow{s} a)))$	
[2]	$\mathit{eff}_{s\,\backslash !}(X) = \mathit{eff}_{s\,!}(X)$	
[1]	$\mathit{eff}_{s\backslash =t}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{eff}_{s=t}(X \oplus (\xrightarrow{s} a)))$	
[2]	$\mathit{eff}_{s\backslash =t}(X) = \mathit{eff}_{s=t}(X)$	
[1]	$\mathit{eff}_{s\backslash =*}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{eff}_{s=*}(X \oplus (\xrightarrow{s} a)))$	
[2]	$\mathit{eff}_{s\backslash =*}(X) = \mathit{eff}_{s=*}(X)$	
[1]	$\mathit{eff}_{s.f\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathit{disp}_b(\emptyset, \mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)))$	
[2]	$\mathit{eff}_{s.f\backslash =t}(X) = \mathit{eff}_{s.f=t}(X)$	
[1]	$\mathit{eff}_{s.f\backslash =*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathit{disp}_b(\emptyset, \mathit{eff}_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)))$	
[2]	$\mathit{eff}_{s.f\backslash =*}(X) = \mathit{eff}_{s.f=*}(X)$	
[1]	$\mathit{eff}_{s\backslash =t.f}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{s} a)))$	
[2]	$\mathit{eff}_{s\backslash =t.f}(X) = \mathit{eff}_{s=t.f}(X)$	
[1]	$\mathit{eff}_{s\,\backslash\backslash !}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s\,\backslash\backslash !}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{clr}_a(\mathit{eff}_{s\,!}(X \oplus (\xrightarrow{s} a))))$	
[3]	$\mathit{eff}_{s\,\backslash\backslash !}(X) = \mathit{eff}_{s\,!}(X)$	
[1]	$\mathit{eff}_{s\backslash\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s\backslash\backslash =t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s\backslash\backslash =t}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{clr}_a(\mathit{eff}_{s=t}(X \oplus (\xrightarrow{s} a))))$	
[3]	$\mathit{eff}_{s\backslash\backslash =t}(X) = \mathit{eff}_{s=t}(X)$	
[1]	$\mathit{eff}_{s\backslash\backslash =*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s\backslash\backslash =*}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{clr}_a(\mathit{eff}_{s=*}(X \oplus (\xrightarrow{s} a))))$	
[3]	$\mathit{eff}_{s\backslash\backslash =*}(X) = \mathit{eff}_{s=*}(X)$	
[1]	$\mathit{eff}_{s.f\backslash\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s.f\backslash\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s.f\backslash\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[1]	$\mathit{eff}_{s.f\backslash\backslash =t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[2]	$\mathit{eff}_{s.f\backslash\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathit{disp}_b(\emptyset, \mathit{clr}_b(\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b))))$	
[3]	$\mathit{eff}_{s.f\backslash\backslash =t}(X) = \mathit{eff}_{s.f=t}(X)$	
[1]	$\mathit{eff}_{s.f\backslash\backslash =*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s.f\backslash\backslash =*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s.f\backslash\backslash =*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[2]	$\mathit{eff}_{s.f\backslash\backslash =*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathit{disp}_b(\emptyset, \mathit{clr}_b(\mathit{eff}_{s.f=*}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b))))$	
[3]	$\mathit{eff}_{s.f\backslash\backslash =*}(X) = \mathit{eff}_{s.f=*}(X)$	
[1]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)$	if $a \neq b$
[1]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)$	if $b \neq c$
[1]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})$	
[2]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_a(\emptyset, \mathit{clr}_a(\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{s} a))))$	
[3]	$\mathit{eff}_{s\backslash\backslash =t.f}(X) = \mathit{eff}_{s=t.f}(X)$	

Table 5. (Continued)

[1]	$yld_{s\,\backslash !}(X) = yld_{s\,!}(X)$	
[1]	$yld_{s\backslash =t}(X) = yld_{s=t}(X)$	
[1]	$yld_{s\backslash =*}(X) = yld_{s=*}(X)$	
[1]	$yld_{s.f\backslash =t}(X) = yld_{s.f=t}(X)$	
[1]	$yld_{s.f\backslash =*}(X) = yld_{s.f=*}(X)$	
[1]	$yld_{s\backslash =t.f}(X) = yld_{s=t.f}(X)$	
[1]	$yld_{s\,\backslash\backslash !}(X) = yld_{s\,!}(X)$	
[1]	$yld_{s\backslash\backslash =t}(X) = yld_{s=t}(X)$	
[1]	$yld_{s\backslash\backslash =*}(X) = yld_{s=*}(X)$	
[1]	$yld_{s.f\backslash\backslash =t}(X) = yld_{s.f=t}(X)$	
[1]	$yld_{s.f\backslash\backslash =*}(X) = yld_{s.f=*}(X)$	
[1]	$yld_{s\backslash\backslash =t.f}(X) = yld_{s=t.f}(X)$	
[1]	$disp_d(X, (\xrightarrow{s} a) \oplus Y) = disp_d(X \oplus (\xrightarrow{s} a), Y)$	
[1]	$disp_d(X \oplus (\xrightarrow{s} a), (a \xrightarrow{f}) \oplus Y) = disp_d(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f}), Y)$	
[1]	$disp_d(X \oplus (a \xrightarrow{f} b), (b \xrightarrow{g}) \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} b) \oplus (b \xrightarrow{g}), Y)$	
[1]	$disp_d(X \oplus (\xrightarrow{s} a), (a \xrightarrow{f} b) \oplus Y) = disp_d(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b), Y)$	
[1]	$disp_d(X \oplus (a \xrightarrow{f} b), (b \xrightarrow{g} c) \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} b) \oplus (b \xrightarrow{g} c), Y)$	
[1]	$disp_d(X \oplus (\xrightarrow{s} a), (a)_n \oplus Y) = disp_d(X \oplus (\xrightarrow{s} a) \oplus (a)_n, Y)$	
[1]	$disp_d(X \oplus (a \xrightarrow{f} b), (b)_n \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} b) \oplus (b)_n, Y)$	
[2]	$disp_d(X \oplus (\xrightarrow{s} d), (a \xrightarrow{f} d) \oplus Y) = disp_d(X \oplus (\xrightarrow{s} d) \oplus (a \xrightarrow{f} d), Y)$	
[2]	$disp_d(X \oplus (a \xrightarrow{f} d), (b \xrightarrow{f} d) \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} d) \oplus (b \xrightarrow{f} d), Y)$	
[2]	$disp_d(X \oplus (\xrightarrow{s} d), (d)_n \oplus Y) = disp_d(X \oplus (\xrightarrow{s} d) \oplus (d)_n, Y)$	
[2]	$disp_d(X \oplus (a \xrightarrow{f} d), (d)_n \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} d) \oplus (d)_n, Y)$	
[3]	$disp_d(X, (a \xrightarrow{f}) \oplus Y) = disp_d(X \oplus (a \xrightarrow{f}), Y)$	if $a \neq d$
[3]	$disp_d(X, (a \xrightarrow{f} b) \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} b), Y)$	if $a \neq d \wedge b \neq d$
[3]	$disp_d(X, (a)_n \oplus Y) = disp_d(X \oplus (a)_n, Y)$	if $a \neq d$
[4]	$disp_d(X, Y) = X$	
[1]	$clr_d(X \oplus (\xrightarrow{s} d)) = clr_d(X)$	
[1]	$clr_d(X \oplus (\xrightarrow{s} a)) = clr_d(X) \oplus (\xrightarrow{s} a)$	if $a \neq d$
[1]	$clr_d(X \oplus (a \xrightarrow{f})) = clr_d(X) \oplus (a \xrightarrow{f})$	
[1]	$clr_d(X \oplus (a \xrightarrow{f} d)) = clr_d(X) \oplus (a \xrightarrow{f})$	
[1]	$clr_d(X \oplus (a \xrightarrow{f} b)) = clr_d(X) \oplus (a \xrightarrow{f} b)$	if $b \neq d$
[1]	$clr_d(X \oplus (a)_n) = clr_d(X) \oplus (a)_n$	

- if all atomic objects found in L' are involved in the safe disposal of a, then L is the result of safe disposal of a on $L \oplus L'$.

We mention further that $clr_a(L)$ removes spot links to a from L and and replaces field links to a by partial field links.

DLD-R is DLD extended with the reclamation features introduced above. The priority rewrite system of DLD-R consists of the rewrite rules from the priority rewrite system for DLD and the rewrite rules given in Tables 4 and 5. Each of the rewrite rules of DLD is incomparable with each of the additional rewrite rules. Moreover, additional rewrite rules in different tables are incomparable.

For DLD-R, the set of effect terms and the set of yield terms can be defined like for DLD. Theorem 6.3 and Proposition 6.5 go through for DLD-R. Moreover, the enabledness of rewrite rules for the auxiliary operators can be characterized like for the effect and yield operators. This means that the one-step reduction relation modulo AC1 determined by the priority rewrite system for DLD-R is decidable as well.

Example 7.1. We consider a data linkage in which $\underline{2}$ and $\underline{3}$ occur as unreachable atomic objects:

$$(\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})\ .$$

We use the rewrite rules for full garbage collection to obtain the following picture of its application to this data linkage:

$$\begin{array}{l}
\mathit{eff}_{\mathsf{fgc}}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{fgc}(\emptyset, (\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{fgc}((\xrightarrow{r} \underline{0}), (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{fgc}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}), (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
(\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1})\ .
\end{array}$$

We use the rewrite rules for restricted garbage collection to obtain the following picture of its application to the same data linkage:

$$\begin{array}{l}
\mathit{eff}_{\mathsf{rgc}}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{rgc}(\emptyset, (\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{rgc}((\xrightarrow{r} \underline{0}), (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{rgc}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}), (\underline{2} \xrightarrow{up} \underline{3}) \oplus (\underline{3} \xrightarrow{dn} \underline{2})) \to_{\mathrm{AC1}} \\
\mathit{rgc}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{3} \xrightarrow{dn} \underline{2}), (\underline{2} \xrightarrow{up} \underline{3})) \to_{\mathrm{AC1}} \\
\mathit{rgc}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{3} \xrightarrow{dn} \underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3}), \emptyset) \to_{\mathrm{AC1}} \\
(\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{3} \xrightarrow{dn} \underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})\ .
\end{array}$$

The effect of restricted garbage collection is different because it does not reclaim atomic objects that occur in a cycle.

Example 7.2. We consider a data linkage in which atomic object $\underline{0}$ is reachable in different ways:

$$(\xrightarrow{r} \underline{0}) \oplus (\xrightarrow{s} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t} \underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})\ .$$

We use the rewrite rules for set spot with safe disposal to obtain the following picture of its application to this data linkage:

$$
\begin{array}{l}
\mathit{eff}_{s\backslash{=}t}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{eff}_{s=t}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3}))) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, (\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{r}\underline{0}), (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}), (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}), (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}), (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3}), \emptyset) \rightarrow_{\mathrm{AC1}} \\
(\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})\ .
\end{array}
$$

We use the rewrite rules for set spot with unsafe disposal to obtain the following picture of its application to the same data linkage:

$$
\begin{array}{l}
\mathit{eff}_{s\backslash\backslash{=}t}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{clr}_{\underline{0}}(\mathit{eff}_{s=t}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})))) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{clr}_{\underline{0}}((\xrightarrow{r}\underline{0}) \oplus (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3}))) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{clr}_{\underline{0}}((\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3}))) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{clr}_{\underline{0}}((\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2})) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{clr}_{\underline{0}}((\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1})) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, \mathit{clr}_{\underline{0}}(\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}(\emptyset, (\xrightarrow{s}\underline{2}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{s}\underline{2}), (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{s}\underline{2}) \oplus (\xrightarrow{t}\underline{2}), (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{2} \xrightarrow{up} \underline{3})) \rightarrow_{\mathrm{AC1}} \\
\mathit{disp}_{\underline{0}}((\xrightarrow{s}\underline{2}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3}), (\underline{0} \xrightarrow{up} \underline{1})) \rightarrow_{\mathrm{AC1}} \\
(\xrightarrow{s}\underline{2}) \oplus (\xrightarrow{t}\underline{2}) \oplus (\underline{2} \xrightarrow{up} \underline{3})\ .
\end{array}
$$

The effect of set spot with unsafe disposal is different because it reclaims an atomic object irrespective of its reachability.

8. Basic Thread Algebra

In this section, we review BTA (Basic Thread Algebra), a form of process algebra which is tailored to the behaviours that are produced by deterministic sequential programs under execution. The behaviours concerned are called *threads*.

In BTA, it is assumed that a fixed but arbitrary finite set $\mathcal{A}$ of *basic actions*, with $\mathtt{tau} \notin \mathcal{A}$, has been given. Besides, $\mathtt{tau}$ is a special basic action. We write $\mathcal{A}_{\mathtt{tau}}$ for $\mathcal{A} \cup \{\mathtt{tau}\}$.

A thread is a behaviour which consists of performing basic actions in a sequential fashion. Upon each basic action performed, a reply from an execution environment determines how the thread proceeds. The

Table 6. Axiom of BTA

$x \trianglelefteq \mathsf{tau} \trianglerighteq y = x \trianglelefteq \mathsf{tau} \trianglerighteq x$	T1

possible replies are the Boolean values T and F. Performing tau, which is considered performing an internal action, will always lead to the reply T.

BTA has one sort: the sort $\mathbf{T}$ of *threads*. To build terms of sort $\mathbf{T}$, BTA has the following constants and operators:

- the *inaction* constant $\mathsf{D} : \mathbf{T}$;
- the *termination* constant $\mathsf{S} : \mathbf{T}$;
- for each $\alpha \in \mathcal{A}_{\mathsf{tau}}$, the binary *postconditional composition* operator $\_ \trianglelefteq \alpha \trianglerighteq \_ : \mathbf{T} \times \mathbf{T} \to \mathbf{T}$.

Terms of sort $\mathbf{T}$ are built as usual. Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{T}$, including x, y, z.

We use infix notation for postconditional composition. We introduce *basic action prefixing* as an abbreviation: $\alpha \circ p$, where p is a term of sort $\mathbf{T}$, abbreviates $p \trianglelefteq \alpha \trianglerighteq p$. We identify expressions of the form $\alpha \circ p$ with the BTA term they stand for.

The thread denoted by a closed term of the form $p \trianglelefteq \alpha \trianglerighteq q$ will first perform α, and then proceed as the thread denoted by p if the reply from the execution environment is T and proceed as the thread denoted by q if the reply from the execution environment is F. The thread denoted by D will become inactive and the thread denoted by S will terminate.

Example 8.1. Some simple examples of closed BTA terms are

$$a \circ (\mathsf{S} \trianglelefteq b \trianglerighteq \mathsf{D})\ , \qquad (b \circ \mathsf{S}) \trianglelefteq a \trianglerighteq \mathsf{D}\ .$$

The first term denotes the thread that first performs basic action a, next performs basic action b, if the reply from the execution environment on performing b is T, after that terminates, and if the reply from the execution environment on performing b is F, after that becomes inactive. The second term denotes the thread that first performs basic action a, if the reply from the execution environment on performing a is T, next performs the basic action b and after that terminates, and if the reply from the execution environment on performing a is F, next becomes inactive.

BTA has only one axiom. This axiom is given in Table 6. Using the abbreviation introduced above, axiom T1 can be written as follows: $x \trianglelefteq \mathsf{tau} \trianglerighteq y = \mathsf{tau} \circ x$.

Each closed BTA term denotes a finite thread, i.e. a thread with a finite upper bound to the number of basic actions that it can perform. Infinite threads, i.e. threads without a finite upper bound to the number of basic actions that it can perform, can be described by guarded recursion.

A *guarded recursive specification* over BTA is a set of recursion equations $\{X = p_X \mid X \in V\}$, where V is a set of variables of sort $\mathbf{T}$ and each p_X is a term of the form D, S or $p \trianglelefteq \alpha \trianglerighteq q$ with p and q BTA terms of sort $\mathbf{T}$ that contain only variables from V. We are only interested in models of BTA in which guarded recursive specifications have unique solutions, such as the projective limit model of BTA presented in [4].

Example 8.2. A simple example of a guarded recursive specification is the one consisting of following two equations:

$$x = x \trianglelefteq a \trianglerighteq y\ , \qquad y = y \trianglelefteq b \trianglerighteq \mathsf{S}\ .$$

The x-component of the solution of this guarded recursive specification is the thread that first performs basic action a repeatedly until the reply from the execution environment on performing a is F, next performs basic action b repeatedly until the reply from the execution environment on performing b is F, and after that terminates.

9. Services and Use Operators

A thread may perform a basic action for the purpose of requesting a named service provided by an execution environment to process a method and to return a reply to the thread at completion of the processing of the method. In this section, we review the extension of BTA with services and operators which are concerned with this kind of interaction between threads and services.

It is assumed that a fixed but arbitrary finite set $\mathcal{F}$ of *foci* has been given. Foci play the role of names of the services provided by an execution environment. It is also assumed that a fixed but arbitrary finite set $\mathcal{M}$ of *methods* has been given. For the set $\mathcal{A}$ of basic actions, we take the set $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Performing a basic action $f.m$ is taken as making a request to the service named f to process command m.

A service is able to process certain methods. The processing of a method may involve a change of the service. The reply value produced by the service at completion of the processing of a method is either T, F or B. The special reply B, standing for blocked, is used to deal with the situation that a service is requested to process a method that it is not able to process.

Example 9.1. A simple example of a service is one that is able to process methods for pushing a natural number on a stack ($\texttt{push:}n$), testing whether the top of the stack equals a natural number ($\texttt{topeq:}n$), and popping the top element from the stack ($\texttt{pop}$). Processing of a pushing method or a popping method changes the service, because it changes the stack with which it deals, and produces the reply value T if no stack overflow or stack underflow occurs and F otherwise. Processing of a testing method does not change the service, because it does not changes the stack with which it deals, and produces the reply value T if the test succeeds and F otherwise. Attempted processing of a method that the service is not able to process changes the service into one that is not able to process any method and produces the reply B.

The following is assumed with respect to services:

- a signature $\Sigma_{\mathcal{S}}$ has been given that includes the following sorts:
  - the sort $\mathbf{S}$ of *services*;
  - the sort $\mathbf{R}'$ of *replies*;

  and the following constants and operators:
  - the *empty service* constant $\delta : \mathbf{S}$;
  - the *reply* constants $\mathsf{T}, \mathsf{F}, \mathsf{B} : \mathbf{R}'$;

Table 7. Axioms for use operators

$\mathsf{S} /_f S = \mathsf{S}$		TSU1
$\mathsf{D} /_f S = \mathsf{D}$		TSU2
$(\mathsf{tau} \circ x) /_f S = \mathsf{tau} \circ (x /_f S)$		TSU3
$(x \unlhd g.m \unrhd y) /_f S = (x /_f S) \unlhd g.m \unrhd (y /_f S)$	if $f \neq g$	TSU4
$(x \unlhd f.m \unrhd y) /_f S = \mathsf{tau} \circ (x /_f \frac{\partial}{\partial m}(S))$	if $\varrho_m(S) = \mathsf{T}$	TSU5
$(x \unlhd f.m \unrhd y) /_f S = \mathsf{tau} \circ (y /_f \frac{\partial}{\partial m}(S))$	if $\varrho_m(S) = \mathsf{F}$	TSU6
$(x \unlhd f.m \unrhd y) /_f S = \mathsf{tau} \circ \mathsf{D}$	if $\varrho_m(S) = \mathsf{B}$	TSU7

  - for each $m \in \mathcal{M}$, the *derived service* operator $\frac{\partial}{\partial m} : \mathbf{S} \to \mathbf{S}$;
  - for each $m \in \mathcal{M}$, the *service reply* operator $\varrho_m : \mathbf{S} \to \mathbf{R}'$;

- a minimal $\Sigma_{\mathcal{S}}$-algebra $\boldsymbol{S}$ has been given in which T, F, and B are mutually different, and
  - $\bigwedge_{m \in \mathcal{M}} \frac{\partial}{\partial m}(z) = z \wedge \varrho_m(z) = \mathsf{B} \Rightarrow z = \delta$ holds;
  - for each $m \in \mathcal{M}$, $\frac{\partial}{\partial m}(z) = \delta \Leftrightarrow \varrho_m(z) = \mathsf{B}$ holds.

The intuition concerning $\frac{\partial}{\partial m}$ and ϱ_m is that on a request to service S to process method m:

- if $\varrho_m(S) \neq \mathsf{B}$, S processes m, produces the reply $\varrho_m(S)$, and then proceeds as $\frac{\partial}{\partial m}(S)$;
- if $\varrho_m(S) = \mathsf{B}$, S is not able to process method m and proceeds as δ.

The empty service δ itself is unable to process any method.

We introduce the following additional operators:

- for each $f \in \mathcal{F}$, the binary *use* operator $\_ /_f \_ : \mathbf{T} \times \mathbf{S} \to \mathbf{T}$.

We use infix notation for the use operators.

Intuitively, the thread denoted by a closed term of the form $p /_f S$ is the thread that results from processing all basic actions performed by thread denoted by p that are of the form $f.m$ by service S. When a basic action of the form $f.m$ performed by a thread is processed by a service, the service changes in accordance with the method concerned and affects the thread as follows: the basic action is turned into the internal action tau and the two ways to proceed reduce to one on the basis of the reply value produced by the service.

The axioms for the use operators are given in Table 7. In this table, f and g stand for arbitrary foci from $\mathcal{F}$, m stands for an arbitrary method from $\mathcal{M}$, and S stands for an arbitrary term of sort $\mathbf{S}$. Axioms TSU3 and TSU4 express that the internal action tau and basic actions of the form $g.m$, where $f \neq g$, are not processed by the service. Axioms TSU5 and TSU6 express that a thread is affected by a service as described above when a basic action of the form $f.m$ performed by the thread is processed by the service. Axiom TSU7 expresses that inaction takes place when a basic action of the form $f.m$ performed by the thread cannot be processed by the service.

Example 9.2. We consider the stack services described in Example 9.1. For each sequence σ of natural numbers, we take $\mathcal{NNS}(\sigma)$ as a constant for the stack service that deals with a stack whose content is represented by σ. Provided a precise description of the stack services has been given, axioms TSU1–TSU7 can be used to prove the following equations for all σ whose size is less than the maximal stack size:

$$
\begin{aligned}
&(\mathtt{nns.push}{:}n \circ x) /_{\mathtt{nns}} \mathcal{NNS}(\sigma) = \mathsf{tau} \circ (x /_{\mathtt{nns}} \mathcal{NNS}(n\sigma)) \;, \\
&(x \trianglelefteq \mathtt{nns.pop} \trianglerighteq \mathsf{S}) /_{\mathtt{nns}} \mathcal{NNS}(\epsilon) = \mathsf{tau} \circ \mathsf{S} \;,^{3} \\
&(x \trianglelefteq \mathtt{nns.pop} \trianglerighteq \mathsf{S}) /_{\mathtt{nns}} \mathcal{NNS}(n\sigma) = \mathsf{tau} \circ (x /_{\mathtt{nns}} \mathcal{NNS}(\sigma)) \;.
\end{aligned}
$$

Henceforth, we write $\mathrm{BTA}_{\mathrm{use}}$ for BTA, taking the set $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$ for $\mathcal{A}$, extended with the use operators and the axioms from Table 7.

10. Thread Algebra and Data Linkage Dynamics Combined

The state changes and replies that result from performing the basic actions of data linkage dynamics can be achieved by means of services. In this section, we explain how basic thread algebra can be combined with data linkage dynamics by means of services and use operators such that the whole can be used for studying issues concerning the use of dynamic data structures in programming.

Recall that we write $\mathcal{DL}$ for the set of elements of the initial model of DLA. It is assumed that $\mathtt{dld} \in \mathcal{F}$ and $A_{\mathrm{DLD}} \subseteq \mathcal{M}$.

For $\Sigma_{\mathcal{S}}$, we take the signature that consists of the sorts, constants and operators that are mentioned in the assumptions with respect to services made in Section 9 and a constant $\mathcal{DLD}(L)$ of sort $\mathbf{S}$ for each $L \in \mathcal{DL}$.

For $\boldsymbol{\mathcal{S}}$, we take a minimal $\Sigma_{\mathcal{S}}$-algebra that satisfies the conditions that are mentioned in the assumptions with respect to services made in Section 9 and the following conditions for each $L \in \mathcal{DL}$:

$$
\begin{aligned}
&\tfrac{\partial}{\partial m}(\mathcal{DLD}(L)) = \mathcal{DLD}(\mathit{eff}_m(L)) && \text{if } m \in A_{\mathrm{DLD}} \;, \\
&\tfrac{\partial}{\partial m}(\mathcal{DLD}(L)) = \delta && \text{if } m \notin A_{\mathrm{DLD}} \;, \\[1ex]
&\varrho_m(\mathcal{DLD}(L)) = \mathit{yld}_m(L) && \text{if } m \in A_{\mathrm{DLD}} \;, \\
&\varrho_m(\mathcal{DLD}(L)) = \mathsf{B} && \text{if } m \notin A_{\mathrm{DLD}} \;.
\end{aligned}
$$

Note that $\boldsymbol{\mathcal{S}}$ is unique up to isomorphism. The elements of the interpretation of the sort $\mathbf{S}$ in $\boldsymbol{\mathcal{S}}$ are called *data linkage dynamics services*.

By means of threads and data linkage dynamics services, we can give a precise picture of computations in which dynamic data structures are involved.

In order to represent computations, we use the binary relation $\stackrel{\mathsf{tau}}{\to}$ on closed terms of $\mathrm{BTA}_{\mathrm{use}}$ defined by $p \stackrel{\mathsf{tau}}{\to} q$ if and only if $p = \mathsf{tau} \circ q$. Thus, $p \stackrel{\mathsf{tau}}{\to} q$ indicates that p can perform an internal action and then proceed as q. Moreover, for each method $\alpha \in A_{\mathrm{DLD}}$, we write (α) instead of $\mathtt{dld}.\alpha$.

Example 10.1. We consider a simple thread in which non-value-related basic actions of DLD occur:

$$(r\,!) \circ (t\,!) \circ (r/\mathit{up}) \circ (t/\mathit{dn}) \circ (r.\mathit{up} = t) \circ (t.\mathit{dn} = r) \circ (t = *) \circ \mathsf{S} \;.$$

[3] We use the notation ϵ for the empty sequence.

We use the rewrite rules of DLD and axiom TSU5 from the axioms for the use operators to obtain the following picture of the computation of this thread in the case where the initial state is the empty data linkage:

$$
\begin{aligned}
&((r\,!) \circ (t\,!) \circ (r/up) \circ (t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}(\emptyset) \overset{\mathsf{tau}}{\to} \\
&((t\,!) \circ (r/up) \circ (t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}(\overset{r}{\to} \underline{0}) \overset{\mathsf{tau}}{\to} \\
&((r/up) \circ (t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{r}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1})) \overset{\mathsf{tau}}{\to} \\
&((t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{r}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{0} \overset{up}{\to})) \overset{\mathsf{tau}}{\to} \\
&((r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{r}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{0} \overset{up}{\to}) \oplus (\underline{1} \overset{dn}{\to})) \overset{\mathsf{tau}}{\to} \\
&((t.dn = r) \circ (t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{r}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{0} \overset{up}{\to} \underline{1}) \oplus (\underline{1} \overset{dn}{\to})) \overset{\mathsf{tau}}{\to} \\
&((t = *) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{r}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{0} \overset{up}{\to} \underline{1}) \oplus (\underline{1} \overset{dn}{\to} \underline{0})) \overset{\mathsf{tau}}{\to} \\
&\mathsf{S} \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{r}{\to} \underline{0}) \oplus (\underline{0} \overset{up}{\to} \underline{1}) \oplus (\underline{1} \overset{dn}{\to} \underline{0}))\ .
\end{aligned}
$$

Example 10.2. We also consider a simple thread in which value-related basic actions of DLD occur:

$$(u <= -\,u) \circ (s <= t + u) \circ (u <= -\,u) \circ \mathsf{S}\ .$$

This is a thread for calculating the difference of two values as described in Section 5. We use the rewrite rules of DLD and axiom TSU5 from the axioms for the use operators to obtain the following picture of a computation of this thread:

$$
\begin{aligned}
&((u <= -\,u) \circ (s <= t + u) \circ (u <= -\,u) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{s}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{1})_7 \oplus (\overset{u}{\to} \underline{2}) \oplus (\underline{2})_3) \overset{\mathsf{tau}}{\to} \\
&((s <= t + u) \circ (u <= -\,u) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{s}{\to} \underline{0}) \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{1})_7 \oplus (\overset{u}{\to} \underline{2}) \oplus (\underline{2})_{-3}) \overset{\mathsf{tau}}{\to} \\
&((u <= -\,u) \circ \mathsf{S}) \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{s}{\to} \underline{0}) \oplus (\underline{0})_4 \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{1})_7 \oplus (\overset{u}{\to} \underline{2}) \oplus (\underline{2})_{-3}) \overset{\mathsf{tau}}{\to} \\
&\mathsf{S} \mathbin{/_{\mathtt{dld}}} \mathcal{DLD}((\overset{s}{\to} \underline{0}) \oplus (\underline{0})_4 \oplus (\overset{t}{\to} \underline{1}) \oplus (\underline{1})_7 \oplus (\overset{u}{\to} \underline{2}) \oplus (\underline{2})_3)\ .
\end{aligned}
$$

These examples show that DLA provides a notation that enables us to get a clear picture of the successive states of a computation.

In [7], PGA (ProGram Algebra) and a hierarchy of program notations rooted in PGA are presented. Included in this hierarchy are very simple program notations which are close to existing assembly languages up to and including simple program notations that support structured programming by offering a rendering of conditional and loop constructs. In [7], threads that are definable by finite guarded recursive specifications over BTA are taken as the behaviours of programs represented by closed PGA terms. The combination of basic thread algebra and data linkage dynamics by means of services and use operators can be used for studying issues concerning the use of dynamic data structures in programming at the level of program behaviours. Together with one of the program notations rooted in PGA, this combination can be used for studying issues concerning the use of dynamic data structures in programming at the level of programs.

We mention one such issue. In general terms, the issue is whether we can do without automatic garbage collection by program transformation at the price of a linear increase of the number of available atomic objects. Below we phrase this issue more precisely for PGLD, but it can be studied using any other program notation rooted in PGA as well. PGLD is close to existing assembly languages and has absolute jump instructions.

Let $\mathrm{PGLD}_{\mathtt{dld}}$ be an instance of PGLD in which all basic actions of DLD are available as basic instructions. For each program P from $\mathrm{PGLD}_{\mathtt{dld}}$, we write $|P|$ for the thread that is the behaviour of P according to [7]. Let $\mathrm{DLD}_{\mathsf{afgc}}$ be the variation of DLD in which all basic actions of the form $s\,!$ are treated as if they are preceded by fgc and let, for each $L \in \mathcal{DL}$, $\mathcal{DLD}_{\mathsf{afgc}}(L)$ be the corresponding data linkage dynamics service with initial state L. Data linkage dynamics services have the cardinality of the set AtObj of atomic objects as parameter. We write $\mathrm{DLD}^n(L)$ and $\mathrm{DLD}^n_{\mathsf{afgc}}(L)$ to indicate that the actual cardinality is n. The above-mentioned issue can now be phrased as follows: for which natural numbers c and c' does there exist a program transformation that transforms each program P from $\mathrm{PGLD}_{\mathtt{dld}}$ to a program Q from $\mathrm{PGLD}_{\mathtt{dld}}$ such that, for all natural numbers n, $|P| /_{\mathtt{dld}}\, \mathcal{DLD}^n_{\mathsf{afgc}}(\emptyset) = |Q| /_{\mathtt{dld}}\, \mathcal{DLD}^{c \cdot n + c'}(\emptyset)$?

11. Another Description of Data Linkage Dynamics

In this section, we describe the state changes and replies that result from performing the basic actions of DLD in the world of sets. This alternative description is a widening of the description of the state changes and replies that result from performing the basic actions of molecular dynamics that was given in [11]. In Section 12, we will demonstrate that the alternative description agrees with the description based on DLA. Thus, we will show the connection between molecular dynamics and the upgrade of it presented in the current paper.

We define sets SS, AS_1, AS_2 and $\mathcal{DLR}$ as follows:

$$
\begin{aligned}
SS &= \mathsf{Spot} \to (\mathsf{AtObj} \cup \{\bot\})\ , \\
AS_1 &= \textstyle\bigcup_{A \in \mathcal{P}(\mathsf{AtObj})} (A \to \bigcup_{F \in \mathcal{P}(\mathsf{Field})} (F \to (\mathsf{AtObj} \cup \{\bot\})))\ , \\
AS_2 &= \textstyle\bigcup_{A \in \mathcal{P}(\mathsf{AtObj})} (A \to (\mathsf{Value} \cup \{\bot\}))\ , \\
\mathcal{DLR} &= \{(\sigma, \zeta, \xi) \in SS \times AS_1 \times AS_2 \mid \\
&\qquad \mathrm{dom}(\zeta) = \mathrm{dom}(\xi) \land \mathrm{rng}(\sigma) \subseteq \mathrm{dom}(\zeta) \cup \{\bot\} \land \\
&\qquad \forall a \in \mathrm{dom}(\zeta)\ (\mathrm{rng}(\zeta(a)) \subseteq \mathrm{dom}(\zeta) \cup \{\bot\})\}\ .
\end{aligned}
$$

The elements of $\mathcal{DLR}$ can be considered representations of deterministic data linkages. Let $(\sigma, \zeta, \xi) \in \mathcal{DLR}$, let $s \in \mathsf{Spot}$, let $a \in \mathrm{dom}(\zeta)$, and let $f \in \mathrm{dom}(\zeta(a))$. Then $\sigma(s)$ is the content of spot s if $\sigma(s) \neq \bot$, f is a field of atomic object a, $\zeta(a)(f)$ is the content of field f of atomic object a if $\zeta(a)(f) \neq \bot$, and $\xi(a)$ is the value assigned to atomic object a if $\xi(a) \neq \bot$. The content of spot s is undefined if $\sigma(s) = \bot$, the content of field f of atomic object a is undefined if $\zeta(a)(f) = \bot$, and the value assigned to atomic object a is undefined if $\xi(a) = \bot$. Notice that $\mathrm{dom}(\zeta)$ is taken for the set of all atomic objects that are in use. Therefore, the content of each spot, i.e. each element of $\mathrm{rng}(\sigma)$, must be in $\mathrm{dom}(\zeta)$ if the content is defined and, for each atomic object a that is in use, the content of each of its fields, i.e. each element of $\mathrm{rng}(\zeta(a))$, must be in $\mathrm{dom}(\zeta)$ if the content is defined.

The effect and yield operations on $\mathcal{DL}$ are modelled by the effect and yield operations on $\mathcal{DLR}$ that are defined in Table 8. In these tables, $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$ abbreviates $\sigma(s) \neq \bot$ and $\mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(s)$ abbreviates $\sigma(s) \neq \bot \land \xi(\sigma(s)) \neq \bot$. We use the following notation for functions: $[\,]$ for the empty function; $[e \mapsto e']$ for the function f with $\mathrm{dom}(f) = \{e\}$ such that $f(e) = e'$; $f \dagger g$ for the function h with

Table 8. Definition of effect and yield operations

$\mathit{eff}'_{s\,!}(\sigma,\zeta,\xi) = (\sigma \dagger [s \mapsto ch(\mathsf{AtObj} \setminus \mathrm{dom}(\zeta))], \zeta \dagger [ch(\mathsf{AtObj} \setminus \mathrm{dom}(\zeta)) \mapsto [\,]], \xi \dagger [ch(\mathsf{AtObj} \setminus \mathrm{dom}(\zeta)) \mapsto \bot])$	if $\mathrm{dom}(\zeta) \subset \mathsf{AtObj}$
$\mathit{eff}'_{s\,!}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\mathrm{dom}(\zeta) = \mathsf{AtObj}$
$\mathit{eff}'_{s=t}(\sigma,\zeta,\xi) = (\sigma \dagger [s \mapsto \sigma(t)], \zeta, \xi)$	
$\mathit{eff}'_{s=*}(\sigma,\zeta,\xi) = (\sigma \dagger [s \mapsto \bot], \zeta, \xi)$	
$\mathit{eff}'_{s==t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s==*}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s/f}(\sigma,\zeta,\xi) = (\sigma, \zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \dagger [f \mapsto \bot]], \xi)$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \notin \mathrm{dom}(\zeta(\sigma(s)))$
$\mathit{eff}'_{s/f}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \notin \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s\setminus f}(\sigma,\zeta,\xi) = (\sigma, \zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \lhd \{f\}], \xi)$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s)))$
$\mathit{eff}'_{s\setminus f}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s\,\mid\,f}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s.f=t}(\sigma,\zeta,\xi) = (\sigma, \zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \dagger [f \mapsto \sigma(t)]], \xi)$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s)))$
$\mathit{eff}'_{s.f=t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s.f=*}(\sigma,\zeta,\xi) = (\sigma, \zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \dagger [f \mapsto \bot]], \xi)$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s)))$
$\mathit{eff}'_{s.f=*}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s=t.f}(\sigma,\zeta,\xi) = (\sigma \dagger [s \mapsto \zeta(\sigma(t))(f)], \zeta, \xi)$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(t) \land f \in \mathrm{dom}(\zeta(\sigma(t)))$
$\mathit{eff}'_{s=t.f}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(t) \land f \in \mathrm{dom}(\zeta(\sigma(t))))$
$\mathit{eff}'_{s<=0}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto 0])$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$\mathit{eff}'_{s<=0}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$\mathit{eff}'_{s<=1}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto 1])$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$\mathit{eff}'_{s<=1}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$\mathit{eff}'_{s<=t+u}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto \xi(\sigma(t)) + \xi(\sigma(u))])$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u)$
$\mathit{eff}'_{s<=t+u}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u))$
$\mathit{eff}'_{s<=t\cdot u}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto \xi(\sigma(t)) \cdot \xi(\sigma(u))])$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u)$
$\mathit{eff}'_{s<=t\cdot u}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u))$
$\mathit{eff}'_{s<=-\,t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto -\xi(\sigma(t))])$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t)$
$\mathit{eff}'_{s<=-\,t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t))$
$\mathit{eff}'_{s<=1/t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto \xi(\sigma(t))^{-1}])$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t)$
$\mathit{eff}'_{s<=1/t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t))$
$\mathit{eff}'_{s=?t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s=?*}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	

Table 8. (Continued)

$yld'_{s\,!}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{dom}(\zeta) \subset \mathsf{AtObj}$
$yld'_{s\,!}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\mathrm{dom}(\zeta) = \mathsf{AtObj}$
$yld'_{s=t}(\sigma,\zeta,\xi) = \mathsf{T}$	
$yld'_{s=*}(\sigma,\zeta,\xi) = \mathsf{T}$	
$yld'_{s==t}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\sigma(s) = \sigma(t)$
$yld'_{s==t}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\sigma(s) \neq \sigma(t)$
$yld'_{s==*}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\neg\, \mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$yld'_{s==*}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$yld'_{s/f}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \notin \mathrm{dom}(\zeta(\sigma(s)))$
$yld'_{s/f}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \notin \mathrm{dom}(\zeta(\sigma(s))))$
$yld'_{s\backslash f}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s)))$
$yld'_{s\backslash f}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s))))$
$yld'_{s\,\mid\,f}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s)))$
$yld'_{s\,\mid\,f}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s))))$
$yld'_{s.f=t}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s)))$
$yld'_{s.f=t}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s))))$
$yld'_{s.f=*}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s)))$
$yld'_{s.f=*}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge f \in \mathrm{dom}(\zeta(\sigma(s))))$
$yld'_{s=t.f}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(t) \wedge f \in \mathrm{dom}(\zeta(\sigma(t)))$
$yld'_{s=t.f}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(t) \wedge f \in \mathrm{dom}(\zeta(\sigma(t))))$
$yld'_{s<=0}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$yld'_{s<=0}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\, \mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$yld'_{s<=1}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$yld'_{s<=1}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\, \mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$yld'_{s<=t+u}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u)$
$yld'_{s<=t+u}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u))$
$yld'_{s<=t\cdot u}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u)$
$yld'_{s<=t\cdot u}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u))$
$yld'_{s<=-\,t}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t)$
$yld'_{s<=-\,t}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t))$
$yld'_{s<=1/t}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t)$
$yld'_{s<=1/t}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t))$
$yld'_{s=?t}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{o}}_{\sigma}(t) \wedge \xi(\sigma(s)) = \xi(\sigma(t))$
$yld'_{s=?t}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \mathrm{def}^{\mathrm{o}}_{\sigma}(t) \wedge \xi(\sigma(s)) = \xi(\sigma(t)))$
$yld'_{s=?*}(\sigma,\zeta,\xi) = \mathsf{T}$	if $\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \neg\, \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(s)$
$yld'_{s=?*}(\sigma,\zeta,\xi) = \mathsf{F}$	if $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \wedge \neg\, \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(s))$

$\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ such that for all $e \in \mathrm{dom}(h)$, $h(e) = f(e)$ if $e \notin \mathrm{dom}(g)$ and $h(e) = g(e)$ otherwise; $f \lhd S$ for the function g with $\mathrm{dom}(g) = S$ such that for all $e \in \mathrm{dom}(g)$, $g(e) = f(e)$; and $f \ntriangleleft S$ for the function g with $\mathrm{dom}(g) = \mathrm{dom}(f) \setminus S$ such that for all $e \in \mathrm{dom}(g)$, $g(e) = f(e)$.

12. Correctness of the Alternative Description

In this section, we show that the description of the state changes and replies that result from performing the basic actions of DLD given in Section 11 agrees with the one given in Sections 4 and 5, provided only deterministic data linkages are considered. Recall that the basic actions of DLD preserve determinism of data linkages. This means that non-deterministic data linkages are not relevant to DLD if only deterministic data linkages are allowed as initial state.

The step from deterministic data linkages to the concrete states introduced in Section 11 is an instance of a data refinement:

- the concrete states are considered representations of deterministic data linkages;
- the effect and yield operations on deterministic data linkages are modelled by the effect and yield operations on the concrete states.

This data refinement is correct in the sense that:

- there is a representation of each deterministic data linkage;
- for each $\alpha \in A_{\mathrm{DLD}}$, for each deterministic data linkage, the result of applying eff'_α to the representation of that data linkage is the representation of the result of applying eff_α to that data linkage;
- for each $\alpha \in A_{\mathrm{DLD}}$, for each deterministic data linkage, the result of applying yld'_α to the representation of that data linkage is the result of applying yld_α to that data linkage.

Correctness in this sense agrees with the notion of correctness for data refinements as used in, for example, the software development method VDM [28]. Following the terminology of VDM, the three aspects of correctness might be called representation adequacy, action effect modelling and action reply modelling.

For the purpose of stating the above-mentioned correctness rigorously, we introduce a *retrieve function* that relates the concrete states to deterministic data linkages:

$$
\begin{aligned}
&\mathit{retr}(\sigma, \zeta, \xi) = \\
&\quad \textstyle\bigoplus_{s \in \{s' \in \mathrm{dom}(\sigma) \mid \sigma(s') \neq \bot\}} (\xrightarrow{s} \sigma(s)) \oplus \\
&\quad \textstyle\bigoplus_{a \in \mathrm{dom}(\zeta)} \big(\bigoplus_{f \in \{f' \in \mathrm{dom}(\zeta(a)) \mid \zeta(a)(f') = \bot\}} (a \xrightarrow{f})\big) \oplus \\
&\quad \textstyle\bigoplus_{a \in \mathrm{dom}(\zeta)} \big(\bigoplus_{f \in \{f' \in \mathrm{dom}(\zeta(a)) \mid \zeta(a)(f') \neq \bot\}} (a \xrightarrow{f} \zeta(a)(f))\big) \oplus \\
&\quad \textstyle\bigoplus_{a \in \{a' \in \mathrm{dom}(\xi) \mid \xi(a') \neq \bot\}} (a)_{\xi(a)} \;,
\end{aligned}
$$

where $\bigoplus_{i \in \mathcal{I}} D_i$, with $\mathcal{I}$ a finite set and D_i a DLD term for each $i \in \mathcal{I}$, stands for a term $D_{i_1} \oplus \ldots \oplus D_{i_n}$ such that $\mathcal{I} = \{i_1, \ldots, i_n\}$ (all such terms are equal by associativity and commutativity of $\oplus$) if $\mathcal{I}$ is not an empty set, and for the term $\emptyset$ otherwise. The function retr can be thought of as regaining the abstract deterministic data linkages from their concrete representations.

The correctness of the step from deterministic data linkages to the concrete states is stated rigorously in the following theorem.

Theorem 12.1. (Correctness)
$\mathcal{DL}$ and the effect and yield operations on $\mathcal{DL}$ are related to $\mathcal{DLR}$ and the effect and yield operations on $\mathcal{DLR}$ as follows:

1. for all $L \in \mathcal{DL}$ that are deterministic, there exists a $(\sigma, \zeta, \xi) \in \mathcal{DLR}$ such that:

$$L = \mathit{retr}(\sigma, \zeta, \xi) \;;$$

2. for all $\alpha \in A_{\mathrm{DLD}}$, for all $(\sigma, \zeta, \xi) \in \mathcal{DLR}$:

$$\begin{aligned} \mathit{retr}(\mathit{eff}'_\alpha(\sigma, \zeta, \xi)) &= \mathit{eff}_\alpha(\mathit{retr}(\sigma, \zeta, \xi)) \;, \\ \mathit{yld}'_\alpha(\sigma, \zeta, \xi) &= \mathit{yld}_\alpha(\mathit{retr}(\sigma, \zeta, \xi)) \;. \end{aligned}$$

Proof:
This is straightforwardly proved by case distinction on the basic action α using elementary laws for $\dagger$ and $\lhd$. The following facts about the connection between deterministic data linkages and their representations are useful in the proof:

$$\begin{aligned} &\mathit{retr}(\sigma \dagger [s \mapsto a], \zeta, \xi) = \mathit{retr}(\sigma, \zeta, \xi) \oplus (\xrightarrow{s} a) \;, \\ &\mathit{retr}(\sigma, \zeta \dagger [a \mapsto \zeta(a) \dagger [f \mapsto \bot]], \xi) = \mathit{retr}(\sigma, \zeta, \xi) \oplus (a \xrightarrow{f}) \;, \\ &\mathit{retr}(\sigma, \zeta \dagger [a \mapsto \zeta(a) \dagger [f \mapsto b]], \xi) = \mathit{retr}(\sigma, \zeta, \xi) \oplus (a \xrightarrow{f} b) \;, \\ &\mathit{retr}(\sigma, \zeta, \xi \dagger [a \mapsto n]) = \mathit{retr}(\sigma, \zeta, \xi) \oplus (a)_n \;. \end{aligned}$$

They follow from the definition of retr and elementary laws for $\dagger$. $\square$

13. Another Description of Garbage Reclamation

In this section, we describe the state changes and replies that result from performing the reclamation-related actions of data linkage dynamics in the world of sets. Like the effect operations for reclamation on $\mathcal{DL}$, the effect operations for reclamation on $\mathcal{DLR}$ are defined using auxiliary functions. The auxiliary function $\mathit{reach}{:}\, SS \times AS_1 \to \mathcal{P}(\mathsf{AtObj})$ is used to define both $\mathit{eff}'_{\mathsf{fgc}}$ and $\mathit{eff}'_{\mathsf{rgc}}$, and the auxiliary function $\mathit{incycle} : AS_1 \to \mathcal{P}(\mathsf{AtObj})$ is used to define $\mathit{eff}'_{\mathsf{rgc}}$. These auxiliary functions are defined as follows:

$$\begin{aligned} &\mathit{reach}(\sigma, \zeta) = \textstyle\bigcup_{a \in \mathrm{rng}(\sigma)} \mathit{reach}(a, \zeta) \;, \\ &\mathit{incycle}(\zeta) \;= \{a \in \mathrm{dom}(\zeta) \mid \exists a' \in \mathrm{rng}(\zeta(a))\, (a \in \mathit{reach}(a', \zeta))\} \;, \end{aligned}$$

where $\mathit{reach}(a, \zeta) \subseteq \mathsf{AtObj}$ is inductively defined by the following rules:

- $a \in \mathit{reach}(a, \zeta)$;
- if $a' \in \mathit{reach}(a, \zeta)$ and $a'' \in \mathrm{rng}(\zeta(a'))$, then $a'' \in \mathit{reach}(a, \zeta)$.

Table 9. Definition of effect and yield operations for reclamation

$\mathit{eff}'_{\mathsf{fgc}}(\sigma,\zeta,\xi) = (\sigma, \zeta \lhd \mathit{reach}(\sigma,\zeta), \xi \lhd \mathit{reach}(\sigma,\zeta))$	$\mathit{yld}'_{\mathsf{fgc}}(\sigma,\zeta,\xi) = \mathsf{T}$
$\mathit{eff}'_{\mathsf{rgc}}(\sigma,\zeta,\xi) =$ $(\sigma, \zeta \lhd (\mathit{reach}(\sigma,\zeta) \cup \mathit{incycle}(\zeta)), \xi \lhd (\mathit{reach}(\sigma,\zeta) \cup \mathit{incycle}(\zeta)))$	$\mathit{yld}'_{\mathsf{rgc}}(\sigma,\zeta,\xi) = \mathsf{T}$
$\mathit{eff}'_{s\,\backslash !}(\sigma,\zeta,\xi) = \mathit{sd}(\sigma(s), \mathit{eff}'_{s\,!}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\,\backslash !}(\sigma,\zeta,\xi) = \mathit{yld}'_{s\,!}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\backslash = t}(\sigma,\zeta,\xi) = \mathit{sd}(\sigma(s), \mathit{eff}'_{s=t}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\backslash = t}(\sigma,\zeta,\xi) = \mathit{yld}'_{s=t}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\backslash = *}(\sigma,\zeta,\xi) = \mathit{sd}(\sigma(s), \mathit{eff}'_{s=*}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\backslash = *}(\sigma,\zeta,\xi) = \mathit{yld}'_{s=*}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s.f\backslash = t}(\sigma,\zeta,\xi) = \mathit{sd}(\zeta(\sigma(s))(f), \mathit{eff}'_{s.f=t}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s.f\backslash = t}(\sigma,\zeta,\xi) = \mathit{yld}'_{s.f=t}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s.f\backslash = *}(\sigma,\zeta,\xi) = \mathit{sd}(\zeta(\sigma(s))(f), \mathit{eff}'_{s.f=*}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s.f\backslash = *}(\sigma,\zeta,\xi) = \mathit{yld}'_{s.f=*}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\backslash = t.f}(\sigma,\zeta,\xi) = \mathit{sd}(\sigma(s), \mathit{eff}'_{s=t.f}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\backslash = t.f}(\sigma,\zeta,\xi) = \mathit{yld}'_{s=t.f}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\,\backslash\backslash !}(\sigma,\zeta,\xi) = \mathit{ud}(\sigma(s), \mathit{eff}'_{s\,!}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\,\backslash\backslash !}(\sigma,\zeta,\xi) = \mathit{yld}'_{s\,!}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\backslash\backslash = t}(\sigma,\zeta,\xi) = \mathit{ud}(\sigma(s), \mathit{eff}'_{s=t}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\backslash\backslash = t}(\sigma,\zeta,\xi) = \mathit{yld}'_{s=t}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\backslash\backslash = *}(\sigma,\zeta,\xi) = \mathit{ud}(\sigma(s), \mathit{eff}'_{s=*}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\backslash\backslash = *}(\sigma,\zeta,\xi) = \mathit{yld}'_{s=*}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s.f\backslash\backslash = t}(\sigma,\zeta,\xi) = \mathit{ud}(\zeta(\sigma(s))(f), \mathit{eff}'_{s.f=t}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s.f\backslash\backslash = t}(\sigma,\zeta,\xi) = \mathit{yld}'_{s.f=t}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s.f\backslash\backslash = *}(\sigma,\zeta,\xi) = \mathit{ud}(\zeta(\sigma(s))(f), \mathit{eff}'_{s.f=*}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s.f\backslash\backslash = *}(\sigma,\zeta,\xi) = \mathit{yld}'_{s.f=*}(\sigma,\zeta,\xi)$
$\mathit{eff}'_{s\backslash\backslash = t.f}(\sigma,\zeta,\xi) = \mathit{ud}(\sigma(s), \mathit{eff}'_{s=t.f}(\sigma,\zeta,\xi))$	$\mathit{yld}'_{s\backslash\backslash = t.f}(\sigma,\zeta,\xi) = \mathit{yld}'_{s=t.f}(\sigma,\zeta,\xi)$

The auxiliary function $\mathit{sd} : \mathsf{AtObj} \times \mathcal{DLR} \to \mathcal{DLR}$ is used to define eff'_α for the actions α in which a basic action of DLD is combined with safe disposal, and the auxiliary function $\mathit{ud} : \mathsf{AtObj} \times \mathcal{DLR} \to \mathcal{DLR}$ is used to define eff'_α for the actions α in which a basic action of DLD is combined with unsafe disposal. These auxiliary functions are defined as follows:

$$
\begin{aligned}
\mathit{sd}(a, (\sigma,\zeta,\xi)) &= (\sigma, \zeta ⩤ \{a\}, \xi ⩤ \{a\}) && \text{if } a \notin \mathit{reach}(\sigma,\zeta)\ , \\
\mathit{sd}(a, (\sigma,\zeta,\xi)) &= (\sigma,\zeta,\xi) && \text{if } a \in \mathit{reach}(\sigma,\zeta)\ ,
\end{aligned}
$$

$$
\mathit{ud}(a, (\sigma,\zeta,\xi)) = \mathit{sd}(a, (\mathit{clr}_{\mathrm{s}}(a,\sigma), \mathit{clr}_{\mathrm{f}}(a,\zeta), \xi))\ ,
$$

where $\mathit{clr}_{\mathrm{s}} : \mathsf{AtObj} \times \mathit{SS} \to \mathit{SS}$ and $\mathit{clr}_{\mathrm{f}} : \mathsf{AtObj} \times \mathit{AS}_1 \to \mathit{AS}_1$ are defined as follows:

$$
\begin{aligned}
\mathit{clr}_{\mathrm{s}}(a,\sigma)(s) &= \sigma(s) && \text{if } \sigma(s) \neq a\ , \\
\mathit{clr}_{\mathrm{s}}(a,\sigma)(s) &= \bot && \text{if } \sigma(s) = a\ ,
\end{aligned}
$$

$$
\begin{aligned}
\mathit{clr}_{\mathrm{f}}(a,\zeta)(a')(f) &= \zeta(a')(f) && \text{if } \zeta(a')(f) \neq a\ , \\
\mathit{clr}_{\mathrm{f}}(a,\zeta)(a')(f) &= \bot && \text{if } \zeta(a')(f) = a\ .
\end{aligned}
$$

The effect and yield operations for reclamation on $\mathcal{DLR}$ are defined in Table 9.

Theorem 12.1 goes through for DLD-R. The additional cases to be considered involve proofs by induction over the definition of $\mathit{reach}(a,\zeta)$ for appropriate a and ζ.

14. Conclusions

We have presented an algebra of which the elements are intended for modelling the states of computations in which dynamic data structures are involved. We have also presented a simple model of computation in which states of computations are modelled as elements of this algebra and state changes take place by means of certain actions. We have described the state changes and replies that result from performing those actions by means of a term rewriting system with rule priorities.

We followed a rather fundamental approach. Instead of developing the model of computation on top of an existing theory or model, we started from first principles by giving an elementary algebraic specification [15] of the states of computations in which dynamic data structures are involved. We found out that term rewriting with priorities is a convenient technique to describe the dynamic aspects of the model in an appealing mechanical way. In particular, we managed to give a clear idea of the features related to reclamation of garbage.

It stands out that the description of the dynamic aspects of the presented model of computation by means of a term rewriting system with priorities is rather sizable. However, an alternative description in the world of sets that, unlike the one that we have given in this paper, covers both deterministic and non-deterministic data linkages, would be rather sizable as well. Moreover, we believe that the use of conditional term rewriting [6] instead of priority rewriting would give rise to a less compact and more complicated description.

The presented model of computation and its description are well-thought out, but the choices made can only be justified by applications. Applications to that effect should not only include applications as a setting in which theoretical issues concerning dynamic data structures are studied. They should also include applications as a setting in which practical programming problems that involve dynamic data structures are studied. In [8], we have studied the programming of an interpreter for a program notation that is close to existing assembly languages in this setting.

Together with thread algebra and program algebra [7], we hold the model of computation as described in this paper to be a suitable starting-point for investigations into theoretical issues concerning the interplay between programs and dynamic data structures, including issues concerning reclamation of garbage. We have studied one such issue in [10], namely the feasibility of automatically making everything garbage as soon as it can be viewed as garbage. For the study in question, the abstraction from the representation of dynamic data structures by means of pointers turned out to be really useful.

Acknowledgements

This research has been partly carried out in the framework of the Jacquard-project Symbiosis, which is funded by the Netherlands Organisation for Scientific Research (NWO). We thank two anonymous referees for carefully reading a preliminary version of this paper and for suggesting improvements of the presentation of the paper.

References

[1] Arnold, K., Gosling, J., Holmes, D.: *The Java Programming Language*, Fourth edition, Addison-Wesley, Reading, MA, 2005.

[2] Baeten, J. C. M., Bergstra, J. A., Klop, J. W., Weijland, W. P.: Term-Rewriting Systems with Rule Priorities, *Theoretical Computer Science*, **67**(2–3), 1989, 283–301.

[3] Bergstra, J. A., Bethke, I.: Molecular Dynamics, *Journal of Logic and Algebraic Programming*, **51**(2), 2002, 193–214.

[4] Bergstra, J. A., Bethke, I.: Polarized Process Algebra and Program Equivalence, *Proceedings 30th ICALP* (J. C. M. Baeten, J. K. Lenstra, J. Parrow, G. J. Woeginger, Eds.), 2719, Springer-Verlag, 2003.

[5] Bergstra, J. A., Bethke, I., Ponse, A.: Cancellation Meadows: A Generic Basis Theorem and Some Applications, *Computer Journal*, **56**(1), 2013, 3–14.

[6] Bergstra, J. A., Klop, J. W.: Conditional Rewrite Rules: Confluence and Termination, *Journal of Computer and System Sciences*, **32**(3), 1986, 323–362.

[7] Bergstra, J. A., Loots, M. E.: Program Algebra for Sequential Code, *Journal of Logic and Algebraic Programming*, **51**(2), 2002, 125–156.

[8] Bergstra, J. A., Middelburg, C. A.: Programming an Interpreter Using Molecular Dynamics, *Scientific Annals of Computer Science*, **17**, 2007, 47–81.

[9] Bergstra, J. A., Middelburg, C. A.: Thread Algebra for Strategic Interleaving, *Formal Aspects of Computing*, **19**(4), 2007, 445–474.

[10] Bergstra, J. A., Middelburg, C. A.: Data Linkage Dynamics with Shedding, *Fundamenta Informaticae*, **103**(1–4), 2010, 31–52.

[11] Bergstra, J. A., Middelburg, C. A.: A Thread Calculus with Molecular Dynamics, *Information and Computation*, **208**(7), 2010, 817–844.

[12] Bergstra, J. A., Middelburg, C. A.: *Instruction Sequences for Computer Science*, vol. 2 of *Atlantis Studies in Computing*, Atlantis Press, Amsterdam, 2012.

[13] Bergstra, J. A., Ponse, A.: Combining Programs and State Machines, *Journal of Logic and Algebraic Programming*, **51**(2), 2002, 175–192.

[14] Bergstra, J. A., Tucker, J. V.: Equational Specifications, Complete Term Rewriting, and Computable and Semicomputable Algebras, *Journal of the ACM*, **42**(6), 1995, 1194–1230.

[15] Bergstra, J. A., Tucker, J. V.: Elementary Algebraic Specifications of the Rational Complex Numbers, *Goguen Festschrift* (K. Futatsugi, et al., Eds.), 4060, Springer-Verlag, 2006.

[16] Bergstra, J. A., Tucker, J. V.: The Rational Numbers as an Abstract Data Type, *Journal of the ACM*, **54**(2), 2007, Article 7.

[17] Bethke, I., Rodenburg, P. H.: Some Properties of Finite Meadows, `arXiv:0712.0917v1 [math.RA]`, December 2007.

[18] van den Brand, M. G. J., Klint, P., Verhoef, C.: Term Rewriting for Sale, *Electronic Notes in Theoretical Computer Science*, **15**, 1998, 218–241.

[19] Carlisle, M. C.: *Olden: Parallelizing Programs with Dynamic Data Structures on Distributed-Memory Machines*, Ph.D. Thesis, Department of Computer Science, Princeton University, Princeton, 1996.

[20] Corbera, F., Asenjo, R., Zapata, E. L.: A Framework to Capture Dynamic Data Structures in Pointer-Based Codes, *IEEE Transactions on Parallel and Distributed Systems*, **15**(2), 2004, 151–166.

[21] Dershowitz, N., Jouannaud, J. P.: Rewrite Systems, in: *Handbook of Theoretical Computer Science* (J. van Leeuwen, Ed.), vol. B, Elsevier, Amsterdam, 1990, 243–320.

[22] Flajolet, P., Francon, J., Vuillemin, J.: Towards Analysing Sequences of Operations for Dynamic Data Structures, *Foundations of Computer Science 1979*, IEEE Computer Society Press, 1979.

[23] Gnaedig, I.: Termination of Priority Rewriting, *LATA 2009* (A. H. Dediu, A. M. Ionescu, C. Martín-Vide, Eds.), 5457, Springer-Verlag, 2009.

[24] Gosling, J., Joy, B., Steele, G., Bracha, G.: *The Java Language Specification*, Third edition, Addison-Wesley, Reading, MA, 2005.

[25] Hejlsberg, A., Torgersen, M., Wiltamuth, S., Golde, P.: *The C# Programming Language*, Fourth edition, Addison-Wesley, Reading, MA, 2011.

[26] Hejlsberg, A., Wiltamuth, S., Golde, P.: *C# Language Specification*, Addison-Wesley, Reading, MA, 2003.

[27] Hummell, J., Hendren, L. J., Nicolau, A.: A General Data Dependence Test for Dynamic, Pointer-Based Data Structures, *ACM SIGPLAN Notices*, **29**(6), 1994, 218–229.

[28] Jones, C. B.: *Systematic Software Development Using VDM*, second edition, Prentice-Hall, 1990.

[29] Jouannaud, J. P., Kirchner, H.: Completion of a Set of Rules Modulo a Set of Equations, *SIAM Journal of Computing*, **15**(4), 1986, 1155–1194.

[30] Jouannaud, J. P., Marché, C.: Termination and Completion Modulo Associativity, Commutativity and Identity, *Theoretical Computer Science*, **104**(1), 1992, 29–51.

[31] Kirchner, C., Kirchner, H., de Oliveira, A. S.: Analysis of Rewrite-Based Access Control Policies, *Electronic Notes in Theoretical Computer Science*, **234**, 2009, 55–75.

[32] Klop, J. W.: Term Rewriting Systems, in: *Handbook of Logic in Computer Science* (S. Abramsky, D. Gabbay, T. Maibaum, Eds.), vol. 2, Oxford University Press, Oxford, 1992, 1–116.

[33] Klop, J. W., de Vrijer, R.: First-Order Term Rewriting Systems, in: *Term Rewriting Systems* (Terese, Ed.), vol. 55 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, Cambridge, 2003, 24–59.

[34] Knuth, D. E.: Deletions That Preserve Randomness, *IEEE Transactions on Software Engineering*, **3**(5), 1977, 351–359.

[35] McCarthy, J.: Recursive Functions of Symbolic Expressions and Their Computation by Machine, *Communications of the ACM*, **3**(4), 1960, 184–195.

[36] McCarthy, J., Abrahams, P. W., Edwards, D. J., Hart, T. P., Levin, M. I.: *LISP 1.5 Programmer's Manual*, MIT Press, Cambridge, MA, 1962.

[37] Mohan, C. K.: Priority Rewriting: Semantics, Confluence, and Conditionals, *RTA'89* (N. Dershowitz, Ed.), 355, Springer-Verlag, 1989.

[38] Overmars, M. H., van Leeuwen, J.: Dynamic Multi-Dimensional Data Structures Based on Quad- and K-D Trees, *Acta Informatica*, **17**(3), 1982, 267–285.

[39] Pheng, S., Verbrugge, C.: Dynamic Data Structure Analysis for Java Programs, *Program Comprehension 2006*, IEEE Computer Society Press, 2006.

[40] van der Pol, J.: Operational Semantics of Rewriting with Priorities, *Theoretical Computer Science*, **200**(1–2), 1998, 289–312.

[41] Rogers, A., Carlisle, M. C., Reppy, J. H., Hendren, L. J.: Supporting Dynamic Data Structures on Distributed-Memory Machines, *ACM Transactions on Programming Languages and Systems*, **17**(2), 1995, 233–263.

[42] Sakai, M., Toyama, Y.: Semantics and Strong Sequentiality of Priority Term Rewriting Systems, *Theoretical Computer Science*, **208**(1–2), 1998, 87–110.

[43] Sannella, D., Tarlecki, A.: Algebraic Preliminaries, in: *Algebraic Foundations of Systems Specification* (E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner, Eds.), Springer-Verlag, Berlin, 1999, 13–30.

[44] Tamassia, R.: A Dynamic Data Structure for Planar Graph Embedding, *ICALP'88* (T. Lepistö, A. Salomaa, Eds.), 317, Springer-Verlag, 1988.

[45] Ulidowski, I., Yuen, S.: Generating Priority Rewrite Systems for OSOS Process Languages, *Information and Computation*, **207**(2), 2009, 120–145.

[46] Wirsing, M.: Algebraic Specification, in: *Handbook of Theoretical Computer Science* (J. van Leeuwen, Ed.), vol. B, Elsevier, Amsterdam, 1990, 675–788.

[47] Zantema, H.: Termination, in: *Term Rewriting Systems* (Terese, Ed.), vol. 55 of *Cambridge Tracts in Theoretical Computer Science*, Cambridge University Press, Cambridge, 2003, 181–259.

A. Term Rewriting Systems

In this appendix, the basic definitions and results regarding term rewriting systems are collected.

We assume that a set of constants, a set of operators with fixed arities, and a set of variables have been given; and we consider term rewriting systems for terms that can be built from the constants, operators, and variables in these sets.

A *rewrite rule* is a pair of terms $t \to s$, where t is not a variable and each variable occurring in s occurs in t as well. A *term rewriting system* is a set of rewrite rules.

Let $\mathcal{R}$ be a term rewriting system. Then a *reduction step* of $\mathcal{R}$ is a pair $t \to s$ such that for some substitution instance $t' \to s'$ of a rewrite rule of $\mathcal{R}$, t' is a subterm of t, and s is t with t' replaced by s'. Here, t' is called the *redex* of the reduction step, s' is called the *contractum* of the reduction step, and the replacement of the redex is called the *contraction* of the redex. The *one-step reduction relation* $\to$ of $\mathcal{R}$ is the set of all reduction steps of $\mathcal{R}$. This means that the one-step reduction relation $\to$ is the closure of $\mathcal{R}$ under substitutions and contexts.

Let $\mathcal{R}$ be a term rewriting system. Then a *reduction sequence* of $\mathcal{R}$ is a finite sequence $t_1 \to t_2$, $\ldots$, $t_n \to t_{n+1}$ of consecutive reduction steps of $\mathcal{R}$ or an infinite sequence $t_1 \to t_2$, $t_2 \to t_3, \ldots$ of consecutive reduction steps of $\mathcal{R}$. A *reduction* of $\mathcal{R}$ is a pair $t \twoheadrightarrow s$ such that either t is syntactically equal to s or there exists a finite reduction sequence $t_1 \to t_2, \ldots, t_n \to t_{n+1}$ of $\mathcal{R}$ such that t and s are syntactically equal to t_1 and t_{n+1}, respectively. Here, s is called a *reduct* of t. The *reduction relation* $\twoheadrightarrow$ of $\mathcal{R}$ is the set of all reductions of $\mathcal{R}$. This means that the reduction relation $\twoheadrightarrow$ is the closure of the one-step reduction step relation $\to$ under transitivity and reflexivity.

Let $\mathcal{R}$ be a term rewriting system. Then a term t *is a normal form of* $\mathcal{R}$ if there does not exist a term s such that $t \to s$ is a reduction step of $\mathcal{R}$. A term t *has a normal form in* $\mathcal{R}$ if there exists a term s such that $t \twoheadrightarrow s$ is a reduction of $\mathcal{R}$ and s is a normal form of $\mathcal{R}$. $\mathcal{R}$ *is strongly normalizing* on term t if there does not exist an infinite reduction sequence $t \to t_1$, $t_1 \to t_2$, $t_2 \to t_3, \ldots$ of $\mathcal{R}$. $\mathcal{R}$ *is strongly normalizing* if $\mathcal{R}$ is strongly normalizing on all terms. $\mathcal{R}$ *is weakly confluent* if for each two reduction steps $t \to s_1$ and $t \to s_2$ of $\mathcal{R}$ there exist two reductions $s_1 \twoheadrightarrow s$ and $s_2 \twoheadrightarrow s$ of $\mathcal{R}$. If $\mathcal{R}$ is strongly normalizing and weakly confluent, all terms have a unique normal form in $\mathcal{R}$.

Let $\mathcal{R}$ be a term rewriting system. Then a *reduction ordering* for $\mathcal{R}$ is a well-founded ordering on terms that is closed under substitutions and contexts. $\mathcal{R}$ is strongly normalizing if and only if there exists a reduction ordering $>$ for $\mathcal{R}$ such that $t > s$ for each rewrite rule $t \to s$ of $\mathcal{R}$.

Let $\mathcal{R}$ be a term rewriting system. Then a *critical pair* of $\mathcal{R}$ is a pair $(t' \to s'_1, t' \to s'_2)$ of different reduction steps of $\mathcal{R}$ such that $t' \to s'_2$ is a substitution instance of a rewrite rule $t \to s$ of $\mathcal{R}$ and the redex of $t' \to s'_1$ is a substitution instance of a non-variable subterm of t. A critical pair $(t' \to s'_1, t' \to s'_2)$ of

$\mathcal{R}$ *is convergent* if s'_1 and s'_2 have a common reduct. $\mathcal{R}$ is weakly confluent if and only if all critical pairs of $\mathcal{R}$ are convergent.

Let $\mathcal{R}$ be a term rewriting system and let E be a set of equations between terms. Then a *reduction modulo E step* of $\mathcal{R}$ is a pair $t \to_E s$ such that there exist a reduction step $t' \to s'$ of $\mathcal{R}$ such that $t = t'$ and $s = s'$ are derivable from E. The *one-step reduction modulo E relation* $\to_E$ of $\mathcal{R}$ is the set of all reduction modulo E steps of $\mathcal{R}$. A *reduction modulo E sequence* of $\mathcal{R}$ is a finite sequence $t_1 \to_E t_2, \ldots, t_n \to_E t_{n+1}$ of consecutive reduction modulo E steps of $\mathcal{R}$ or an infinite sequence $t_1 \to_E t_2$, $t_2 \to_E t_3$, $\ldots$ of consecutive reduction modulo E steps of $\mathcal{R}$. A *reduction modulo E* of $\mathcal{R}$ is pair $t \twoheadrightarrow_E s$ such that either $t = s$ is derivable from E or there exists a finite reduction modulo E sequence $t_1 \to_E t_2, \ldots, t_n \to_E t_{n+1}$ of $\mathcal{R}$ such that t and s are syntactically equal to t_1 and t_{n+1}, respectively. The *reduction modulo E relation* $\twoheadrightarrow_E$ of $\mathcal{R}$ is the set of all reductions modulo E of $\mathcal{R}$.

Let $\mathcal{R}$ be a term rewriting system and let E be a set of equations between terms. Then a term t *is a normal form of $\mathcal{R}$ with respect to reduction modulo E* if there does not exist a term s such that $t \to_E s$ is a reduction modulo E step of $\mathcal{R}$. A term t *has a normal form in $\mathcal{R}$ with respect to reduction modulo E* if there exists a term s such that $t \twoheadrightarrow_E s$ is a reduction modulo E of $\mathcal{R}$ and s is a normal form of $\mathcal{R}$ with respect to reduction modulo E. $\mathcal{R}$ *is strongly normalizing modulo E* on term t if there does not exist an infinite sequence $t \to_E t_1$, $t_1 \to_E t_2$, $t_2 \to_E t_3$, $\ldots$ of reduction modulo E steps of $\mathcal{R}$. $\mathcal{R}$ *is strongly normalizing modulo E* if $\mathcal{R}$ is strongly normalizing modulo E on all terms. $\mathcal{R}$ *is weakly confluent modulo E* if for each reduction modulo E step $t \to_E s_1$ of $\mathcal{R}$ and each reduction step $t \to s_2$ of $\mathcal{R}$ there exist reductions modulo E $s_1 \twoheadrightarrow_E s$ and $s_2 \twoheadrightarrow_E s$ of $\mathcal{R}$. If $\mathcal{R}$ is strongly normalizing modulo E and $\mathcal{R}$ is weakly confluent modulo E, all terms have a unique normal form modulo E in $\mathcal{R}$.

Let $\mathcal{R}$ be a term rewriting system and let E be a set of equations between terms. A reduction ordering $>$ for $\mathcal{R}$ *is E-compatible* if $t > s$ implies $t' > s'$ for all terms t, t', s, and s' for which $t = t'$ and $s = s'$ are derivable from E. $\mathcal{R}$ is strongly normalizing modulo E if and only if there exists an E-compatible reduction ordering $>$ for $\mathcal{R}$ such that $t > s$ for each rewrite rule $t \to s$ of $\mathcal{R}$.