University of Amsterdam
Programming Research Group

Data Linkage Algebra, Data Linkage Dynamics, and Priority Rewriting

J.A. Bergstra
C.A. Middelburg

Report PRG0806 May 2008

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

C.A. Middelburg

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
The Netherlands

e-mail: kmiddelb@science.uva.nl

Programming Research Group Electronic Report Series

Data Linkage Algebra, Data Linkage Dynamics, and Priority Rewriting⋆

J.A. Bergstra and C.A. Middelburg

Programming Research Group, University of Amsterdam,
P.O. Box 41882, 1009 DB Amsterdam, the Netherlands
J.A.Bergstra@uva.nl,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, priority rewrite system, meadow, garbage collection.

1998 ACM Computing Classification: D.3.3, D.4.2, F.1.1, F.3.2, F.3.3.

1 Introduction

With the current paper, we carry on the line of research with which a start was made in [5]. The object pursued with that line of research is the development of a theoretical understanding of possible forms of sequential programs, starting from the simplest form of sequential programs, and associated ways of programming. The view is taken that sequential programs in the simplest form are sequences of instructions. PGA (ProGram Algebra), an algebra in which programs are looked upon as sequences of instructions, is taken for the basis of the development aimed at. The work presented in the current paper is primarily concerned with the use of dynamic data structures in programming.

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 [1].

⋆ This research was partly carried out in the framework of the Jacquard-project Symbiosis, which is funded by the Netherlands Organisation for Scientific Research (NWO).

Data linkage dynamics is an upgrade of molecular dynamics. The latter model has been developed in the setting of PGA and was first described in [2]. 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 [13, 14, 10], such as a finite field with zero-totalized division, can be done. The features to reclaim garbage include: full garbage collection, restricted garbage collection (as if reference counts are used), safe disposal of potential garbage, and unsafe disposal of potential garbage.

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 [1]. Further studies of priority rewrite systems can, for example, be found in [25, 28, 26]. 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.

In the line of research carried on, the view is taken that the behaviours exhibited by sequential programs on execution are threads as considered in basic thread algebra.[1] A thread proceeds by performing actions in a sequential fashion. A thread may perform certain actions for the purpose of interacting with some service provided by an execution environment. When processing an action performed by a thread, a service affects that thread by returning a reply value to the thread at completion of the processing of the action. In the setting of basic thread algebra, the use mechanism is introduced in [7] to allow for this kind of interaction. 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 also explain how basic thread algebra can be combined with data linkage dynamics by means of the use mechanism such that the whole can be used for studying issues concerning the use of dynamic data structures in programming.

In [6], 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.

[1] In [5], basic thread algebra is introduced under the name basic polarized process algebra. Prompted by the development of thread algebra [7], which is a design on top of it, basic polarized process algebra has been renamed to basic thread algebra.

This paper is organized as follows. First, we introduce data linkage algebra (Section 2). Next, we present data linkage dynamics (Sections 3, 4, and 5). After that, we review basic thread algebra and the use mechanism (Sections 6 and 7). Then, we explain how basic thread algebra can be combined with data linkage dynamics by means of the use mechanism (Section 8). Following this, we give the alternative description of data linkage dynamics in the world of sets (Sections 9, 10, and 11). Finally, we make some concluding remarks (Section 12).

Some familiarity with term rewriting systems is assumed. The desirable background can, for example, be found in [16, 21, 22].

2 Data Linkage Algebra

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

The elements of the initial algebra of DLA can serve for 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 $\mathbf{DL}$, BTA 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. 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;
  - 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 bears on 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, and n and m stand for arbitrary values from Value.

All closed DLA terms are derivably equal to basic terms over DLA, i.e. closed DLA terms in which the data linkage overriding 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 1 (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. □

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 $a \neq b$

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.

A data linkage $L \in \mathcal{DL}$ is *non-deterministic* if at least one of the following holds:

- $L \oplus (\xrightarrow{s} a) = L \oplus (\xrightarrow{s} b)$ for some $s \in \mathsf{Spot}$ and $a, b \in \mathsf{AtObj}$ with $a \neq b$;

– $L \oplus (a \xrightarrow{f} b) = L \oplus (a \xrightarrow{f} c)$ for some $f \in \mathsf{Field}$ and $a, b, c \in \mathsf{AtObj}$ with $b \neq c$;
– $L \oplus (a \xrightarrow{f} b) = L \oplus (a \xrightarrow{f})$ for some $f \in \mathsf{Field}$ and $a, b \in \mathsf{AtObj}$;
– $L \oplus (a)_n = L \oplus (a)_m$ for some $a \in \mathsf{AtObj}$ and $n, m \in \mathsf{Value}$ with $n \neq m$.

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

In Sections 9, deterministic data linkages are represented by means of functions and data linkage overriding combination is modelled by means of function overriding.

DLA reminds us of frame algebra [8]: links are like transitions and data linkage combination is like frame union. However, there is no counterpart for data linkage overriding combination in frame algebra.

3 Data Linkage Dynamics

DLD (Data Linkage Dynamics) is a simple model of computation that bears on the use of dynamic data structures in programming. It comprises states, basic actions, and the state changes and replies that result from performing the basic actions. The states of DLD are data linkages. In this section, we give an informal explanation of DLD. In Section 4, we will define the state changes and replies that result from performing the basic actions of DLD by means of a term rewriting system with rule priorities. For expository reasons, actions related to reclaiming garbage are treated separately in Section 5.

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, and a fixed but arbitrary finite set AtObj of atomic objects have been given. Unlike in DLA, it is assumed that a fixed but arbitrary finite meadow [13, 14, 10] has been given and that Value consists of the elements of that meadow. 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 prime examples of finite meadows are finite fields with zero-totalized division. The function ch is used whenever a fresh atomic object must be obtained.

Below, we will first explain the features of DLD to structure data dynamically and then the features of DLD to deal with values found in dynamically structured data.

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

– if there exists a unique atomic object a for which $\xrightarrow{s} a$ is contained in L, *the content of spot s* instead of the unique atomic object a for which $\xrightarrow{s} a$ 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;
– if there exists a unique atomic object b for which $a \xrightarrow{f} b$ is contained in L, *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.

In the case where the uniqueness condition is met, the spot or field concerned is called *locally deterministic*.

By means of actions, 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 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 as well.

DLD has the following non-value-related 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 $s, t \in \mathsf{Spot}$ and $f \in \mathsf{Field}$, a *get field action* $s = t.f$.

If only locally deterministic spots and fields are involved, these non-value-related basic actions of DLD can be explained as follows:

- $s\,!$: 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;

– $s = t.f$: 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 only locally deterministic spots and fields are involved in performing a non-value-related action, there is no state change and the reply is F.

The choice is made to deal uniformly with all cases in which not only locally deterministic spots and fields are involved. However, if a field that is not locally deterministic is involved in performing a remove field action or a has field action, there are certainly other imaginable ways to deal with it.

In addition to non-value-related basic actions, DLD has value-related basic actions. By means of the value-related basic actions, calculations in a finite meadow can be done.

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

– *atomic object a has a value assigned* instead of there exists a unique value n for which $(a)_n$ is contained in L;
– if there exists a unique value n for which $(a)_n$ is contained in L, *the value assigned to atomic object a* instead of the unique value n for which $(a)_n$ is contained in L.

DLD has the following value-related basic actions:

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

If only locally deterministic spots are involved, these value-related basic actions of DLD can be explained as follows:

– $s \mathrel{<=} 0$: 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;
– $s \mathrel{<=} 1$: 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;
– $s \mathrel{<=} t + u$: 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;

- $s \mathrel{<=} t \cdot u$: 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;
- $s \mathrel{<=} -t$: 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 spots t and the reply is T; otherwise, nothing changes and the reply is F;
- $s \mathrel{<=} 1 \mathbin{/} t$: 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 spots 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 only locally deterministic spots are involved in performing a value-related action, 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 $s \mathrel{<=} 0$ and then performing $s \mathrel{<=} s + t$. 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 contents of spot s by first performing $u \mathrel{<=} -u$, next performing $s \mathrel{<=} t + u$ and then performing $u \mathrel{<=} -u$ once again. Division can be done like subtraction.

We write A_{DLD} for the set of all non-value-related and value-related basic actions of DLD.

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.

4 Priority Rewrite System for Data Linkage Dynamics

In this section, we describe the state changes and replies that result from performing the basic actions of DLD by means of a priority rewrite system [1]. For that purpose, we introduce for each basic action α of DLD, the unary *effect* operator eff_α and the unary *yield* operator yld_α. 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.

Before we actually describe the state changes and replies that result from performing the basic actions of DLD by means of a priority rewrite system, we shortly review priority rewrite systems.

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

Let a priority rewrite system $(\mathcal{R}, <)$ be given. Let r be a rewrite rule of $\mathcal{R}$. Then an *r-rewrite* is a closed instance of r. 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 $(\mathcal{R}, <)$ 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. The priority rewrite system for DLD given below is well-defined. This is easily shown by means of Theorem 3.11 from [1].

The priority rewrite system for DLD given below is actually a many-sorted priority rewrite system. In addition to the sort **DL** of data linkages, it has the sort **R** of replies. Terms of sort **R** are built using the reply constants $\mathsf{T} : \mathbf{R}$ and $\mathsf{F} : \mathbf{R}$ and the yield operators $\mathit{yld}_\alpha : \mathbf{DL} \to \mathbf{R}$ for $\alpha \in A_{\mathrm{DLD}}$. The definitions and results concerning term rewriting systems extend easily to the many-sorted case, see e.g. [11], 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 DLD given below, 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 $n_i > n_j$.

In the priority rewrite system for DLD given below, the function *atobj* is used to restrict the basic terms over DLA for which 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:

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

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 and the rewrite rules for the effect and yield operators given in Tables 2 and 3. In these tables, L stands for an arbitrary basic term over DLA, s, t and u stand for arbitrary spots from Spot, f stands for an arbitrary field from Field, 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 2 is incomparable with each of the rewrite rules in Table 3 and each of the rewrite rules in Tables 2 and 3 is incomparable with each of the axioms of DLA that are taken as rewrite rules. Moreover, taken as rewrite rules, the axioms of DLA are mutually incomparable.

Henceforth, we will write AC1 for the set of equations that consists of the associativity, commutativity and identity axioms for $\oplus$.[2] The one-step reduction relation of interest for DLD is the one-step reduction relation modulo AC1 determined by the priority rewrite system for DLD. We write $\twoheadrightarrow_{\text{AC1}}$ for the closure of this reduction relation under transitivity and reflexivity.

What is stated before at the end of Section 3 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

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

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

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

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.

[2] The mnemonic name AC1 for the associativity, commutativity and identity axioms for some operator is taken from [20].

Table 2. Rewrite rules for effect 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]	$\mathit{eff}_{s=t}(X \oplus (\xrightarrow{s} a)) = X$	
[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]	$\mathit{eff}_{s=*}(X \oplus (\xrightarrow{s} a)) = X$	
[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]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = X \oplus (\xrightarrow{s} a)$	
[2]	$\mathit{eff}_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = X \oplus (\xrightarrow{s} a)$	
[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]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (\xrightarrow{t} c)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} c) \oplus (\xrightarrow{t} c)$	
[2]	$\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f}) \oplus (\xrightarrow{t} b)) = X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (\xrightarrow{t} b)$	
[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]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	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]	$\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f}) \oplus (\xrightarrow{s} c)) = X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f})$	
[3]	$\mathit{eff}_{s=t.f}(X) = X$	

Table 2. (Continued)

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

Table 3. Rewrite rules for yield operators

[1]	$yld_{s\,!}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s\,!}(L) = \mathsf{T}$	if $atobj(L) \subset \mathsf{AtObj}$
[2]	$yld_{s\,!}(L) = \mathsf{F}$	if $atobj(L) = \mathsf{AtObj}$
[1]	$yld_{s=t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s=t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s=t}(X) = \mathsf{T}$	
[1]	$yld_{s=*}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s=*}(X) = \mathsf{T}$	
[1]	$yld_{s==t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s==t}(X \oplus (\xrightarrow{t} a) \oplus (\xrightarrow{t} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s==t}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{t} a)) = \mathsf{T}$	
[3]	$yld_{s==t}(X \oplus (\xrightarrow{s} a)) = \mathsf{F}$	
[3]	$yld_{s==t}(X \oplus (\xrightarrow{t} a)) = \mathsf{F}$	
[4]	$yld_{s==t}(X) = \mathsf{T}$	
[1]	$yld_{s==*}(X \oplus (\xrightarrow{s} a)) = \mathsf{F}$	
[2]	$yld_{s==*}(X) = \mathsf{T}$	
[1]	$yld_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[2]	$yld_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{F}$	
[2]	$yld_{s/f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[3]	$yld_{s/f}(X \oplus (\xrightarrow{s} a)) = \mathsf{T}$	
[4]	$yld_{s/f}(X) = \mathsf{F}$	
[1]	$yld_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$yld_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[2]	$yld_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$yld_{s\backslash f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$yld_{s\backslash f}(X) = \mathsf{F}$	
[1]	$yld_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = \mathsf{F}$	if $a \neq b$
[1]	$yld_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f} c)) = \mathsf{F}$	if $b \neq c$
[1]	$yld_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b) \oplus (a \xrightarrow{f})) = \mathsf{F}$	
[2]	$yld_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) = \mathsf{T}$	
[2]	$yld_{s\,\vert\,f}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$yld_{s\,\vert\,f}(X) = \mathsf{F}$	
[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]	$yld_{s.f=t}(X) = \mathsf{F}$	

Table 3. (Continued)

[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]	$yld_{s=t.f}(X \oplus (\xrightarrow{t} a) \oplus (a \xrightarrow{f})) = \mathsf{T}$	
[3]	$yld_{s=t.f}(X) = \mathsf{F}$	
[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]	$yld_{s<=-_t}(X) = \mathsf{F}$	

Table 3. (Continued)

[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]	$yld_{s=?t}(X) = \mathsf{F}$	
[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]	$yld_{s=?*}(X) = \mathsf{F}$	

The priority rewrite system for DLD is used in Section 8 in examples concerning computations in which the basic actions of DLD are involved.

Below, 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 $\mathit{yld}_\alpha(D) \in \mathcal{Y}$.

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

Theorem 2 (Normal forms). *The priority rewrite system for* DLD *has the following properties concerning normal forms with respect to reduction modulo* AC1*:*

1. *all elements of $\mathcal{E}$ have unique normal forms modulo* AC1*;*
2. *the normal forms of the elements of $\mathcal{E}$ are basic terms over* DLA*;*

3. *all elements of $\mathcal{Y}$ have unique normal forms;*
4. *the normal forms of the elements of $\mathcal{Y}$ are* T *and* F.

Proof. Properties 1 and 3 are proved combined by showing that all closed DLD terms are weakly confluent modulo AC1 and strongly normalizing modulo AC1.

In the proof of the weak confluence modulo AC1 of all closed DLD terms, 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. The following holds for the ordering $<$ on the rewrite rules of DLD:

- $r' < r$ if and only if the left hand side of r is a substitution instance of the left hand side of r';
- for all critical pairs of rewrites $(t \to s, t' \to s')$ that arise from overlap modulo AC1 of a rewrite rule on an incomparable rewrite rule, s and s' have a common reduct;
- for all critical pairs of rewrites $(t \to s, t' \vdash\!\dashv s')$ 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;
- overlaps between comparable rewrite rules are overlaps at the outermost occurrence only.

From this, the weak confluence modulo AC1 of all closed DLD terms follows straightforwardly, following the same line of reasoning as in the proof of Theorem 4.8 from [1] and using Theorem 16 from [19].

In the proof of the strong normalization modulo AC1 of all closed DLD terms, we write AC for the set of equations that consists of the associativity and commutativity axioms for $\oplus$. Moreover, we write $\to'_{\mathrm{AC}}$ and $\to'_{\mathrm{AC1}}$ for the one-step reduction relations modulo AC and AC1, respectively, determined by the underlying term rewriting system of the priority rewrite system for DLD. First, it is proved that all closed DLD terms are strongly normalizing modulo AC in the underlying term rewriting system of the priority rewrite system for DLD. This is easily proved by means of the reduction ordering induced by the integer polynomials $\phi(D)$ associated with DLD terms D as follows:

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

where it is assumed that, for each variable X over data linkages, there is a variable $\underline{X}$ over integers. Next, it is proved by means of a function θ on closed DLD terms that all closed DLD terms are strongly normalizing modulo AC1 in the underlying term rewriting system of the priority rewrite system for DLD.

The function θ, which transforms closed DLD terms to ones whose one-step reductions modulo AC1 do not depend on the identity axiom for $\oplus$, is defined by $\theta(D) = \theta_1(\theta_0(D))$, where

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

and

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

It is easy to see 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 not all closed DLD terms are strongly normalizing modulo AC1 in the underlying term rewriting system of the priority rewrite system for DLD. 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, not all closed DLD terms are strongly normalizing modulo AC in the underlying term rewriting system of the priority rewrite system for DLD. However, the contrary was proved above. Hence, all closed DLD terms are strongly normalizing modulo AC1 in the underlying term rewriting system of the priority rewrite system for DLD. From this, it follows immediately that all closed DLD terms are strongly normalizing modulo AC1 in the priority rewrite system for DLD.

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

In Tables 2 and 3, 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 2 which is relevant to this point because, modulo AC1, the number of normal forms of the priority rewrite system for DLD is finite.

Corollary 1 (Equivalent priority rewrite system). *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 normal forms with respect to reduction modulo* AC1 *determines the same one-step reduction relation modulo* AC1 *as the priority rewrite system of* DLD.

Let a priority rewrite system $(\mathcal{R}, <)$ be given. Let r be 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 1 (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 interesting corollary of Theorem 2, Corollary 1, and Proposition 1.

Corollary 2 (Decidability of reduction relation). *The one-step reduction relation modulo* AC1 *determined by the priority rewrite system for* DLD *is decidable.*

5 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:

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

If only locally deterministic spots and fields are involved, these reclamation-related actions of data linkage dynamics can be explained as follows:

- $s \backslash !$: if a fresh atomic object can be allocated, then the content of spot s becomes that fresh atomic object, the old content of spot s is reclaimed in case it has become an unreachable atomic object, and the reply is T; otherwise, nothing changes and the reply is F;
- $s \backslash\!= t$: the content of spot s becomes the same as the content of spot t, the old content of spot s is reclaimed in case it has become an unreachable atomic object, and the reply is T;
- $s \backslash\!= *$: the content of spot s becomes undefined, the old content of spot s is reclaimed in case it has become an unreachable atomic object, and the reply is T;
- $s.f \backslash\!= t$: 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, the old content of the field is reclaimed in case it has become an unreachable atomic object, and the reply is T; otherwise, nothing changes and the reply is F;
- $s \backslash\!= t.f$: 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, the old content of spot s is reclaimed in case it has become an unreachable atomic object, and the reply is T; otherwise, nothing changes and the reply is F;
- $s \backslash\backslash !$: if a fresh atomic object can be allocated, then the content of spot s becomes that fresh atomic object, the content of everything containing the old content of spot s becomes undefined, the old content of spot s is reclaimed, and the reply is T; otherwise, nothing changes and the reply is F;
- $s \backslash\backslash\!= t$: the content of spot s becomes the same as the content of spot t, the content of everything containing the old content of spot s becomes undefined, the old content of spot s is reclaimed, and the reply is T;
- $s \backslash\backslash\!= *$: the content of spot s becomes undefined, the content of everything containing the old content of spot s becomes undefined, the old content of spot s is reclaimed, and the reply is T;
- $s.f \backslash\backslash\!= t$: 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, the content of everything containing the old content of the field becomes undefined, the old content of the field is reclaimed, and the reply is T; otherwise, nothing changes and the reply is F;
- $s \backslash\backslash\!= t.f$: 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, the content of everything containing the old content of spot s becomes undefined, the old content of spot s is reclaimed, and the reply is T; otherwise, nothing changes and the reply is F;
- rgc: all unreachable atomic objects that do not occur in a cycle are reclaimed, and the reply is T;
- fgc: all unreachable atomic objects are reclaimed, and the reply is T.

If not only locally deterministic spots and fields are involved in performing a reclamation-related action, there is no state change and the reply is F.

Table 4. Rewrite rules for full garbage collection

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

The differences between the actions with safe disposal and their counterparts without disposal turn out to be insufficiently uniform to allow for the actions with safe disposal to be explained by means of a general remark about the differences. A similar remark applies to the actions with unsafe disposal.

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 C# [17, 15], but also in historic programming languages such as LISP [23, 24]. In [23], the term reclamation is used instead of garbage collection.

The rewrite rules for full garbage collection and restricted garbage collection are given in Tables 4 and 5, respectively. The rewrite rules for safe disposal and unsafe disposal are given in Table 6. In these tables, 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.

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;

Table 5. Rewrite rules for restricted garbage collection

[1]	$\mathit{eff}_{\mathsf{rgc}}(X) = \mathit{rgc}(\emptyset, X)$
[1]	$\mathit{yld}_{\mathsf{rgc}}(X) = \mathsf{T}$
[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]	$\mathit{rgc}(X, Y) = \mathit{rgc}(\emptyset, X)$

- 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 operators $\mathit{eff}_{s\,\backslash !}$, $\mathit{eff}_{s \backslash = t}$, $\mathit{eff}_{s \backslash = *}$, $\mathit{eff}_{s.f \backslash = t}$, $\mathit{eff}_{s \backslash = t.f}$, $\mathit{eff}_{s\,\backslash\backslash !}$, $\mathit{eff}_{s \backslash\backslash = t}$, $\mathit{eff}_{s \backslash\backslash = *}$, $\mathit{eff}_{s.f \backslash\backslash = t}$ and $\mathit{eff}_{s \backslash\backslash = t.f}$ are described using auxiliary operators disp_a and clr_a ($a \in \mathsf{AtObj}$). 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;
- 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 that $\mathit{clr}_a(L)$ removes spot links to a from L and and replaces field links to a by partial field links. Carrying out safe disposal of a on $\mathit{clr}_a(L)$ amounts

Table 6. 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]	$\mathit{eff}_{s\backslash =t.f}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_b(\emptyset, \mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{s} a)))$	
[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]	$\mathit{eff}_{s.f\backslash\backslash =t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b)) =$ $\mathit{disp}_b(\emptyset, \mathit{clr}_a(\mathit{eff}_{s.f=t}(X \oplus (\xrightarrow{s} a) \oplus (a \xrightarrow{f} b))))$	
[3]	$\mathit{eff}_{s.f\backslash\backslash =t}(X) = \mathit{eff}_{s.f=t}(X)$	
[1]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)) = X \oplus (\xrightarrow{s} a) \oplus (\xrightarrow{s} b)$	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]	$\mathit{eff}_{s\backslash\backslash =t.f}(X \oplus (\xrightarrow{s} a)) = \mathit{disp}_b(\emptyset, \mathit{clr}_a(\mathit{eff}_{s=t.f}(X \oplus (\xrightarrow{s} a))))$	
[3]	$\mathit{eff}_{s\backslash\backslash =t.f}(X) = \mathit{eff}_{s=t.f}(X)$	

Table 6. (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]	$yld_{s\backslash =t.f}(X) = yld_{s=t.f}(X)$	
[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]	$yld_{s\backslash\backslash =t.f}(X) = yld_{s=t.f}(X)$	
[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]	$disp_d(X \oplus (a \xrightarrow{f} d), (b \xrightarrow{f} d) \oplus Y) = disp_d(X \oplus (a \xrightarrow{f} d) \oplus (b \xrightarrow{f} d), Y)$	
[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 $a \neq d \land b \neq d$
[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]	$clr_d(X \oplus (a)_n) = clr_d(X) \oplus (a)_n$	

to the same thing as removing all links and value associations in which a is involved from L, i.e. carrying out unsafe disposal of a on L.

$\mathrm{DLD}_{\mathrm{recl}}$ is DLD extended with the reclamation features introduced above. The priority rewrite system of $\mathrm{DLD}_{\mathrm{recl}}$ consists of the rewrite rules of DLD and the rewrite rules given in Tables 4, 5 and 6. 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 $\mathrm{DLD}_{\mathrm{recl}}$, the set of effect terms and the set of yield terms can be defined like for DLD. Theorem 2 and Proposition 1 go through for $\mathrm{DLD}_{\mathrm{recl}}$. 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 $\mathrm{DLD}_{\mathrm{recl}}$ is decidable as well.

In the following examples concerning the reclamation features introduced above, we take the set $\{\underline{n} \mid n \in \{0,\dots,9\}\}$ for AtObj.

Example 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 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})) \to_{\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}))) \to_{\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})) \to_{\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})) \to_{\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})) \to_{\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})) \to_{\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})) \to_{\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) \to_{\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.

6 Basic Thread Algebra

In this section, we review BTA (Basic Thread Algebra), a form of process algebra which is tailored to the description of the behaviour of 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 $\mathsf{tau} \notin \mathcal{A}$, has been given. We write $\mathcal{A}_{\mathsf{tau}}$ for $\mathcal{A} \cup \{\mathsf{tau}\}$. The members of $\mathcal{A}_{\mathsf{tau}}$ are referred to as *actions*.

The intuition is that each basic action performed by a thread is taken as a command to be processed by some service provided by an execution environment. The processing of a command may involve a change of state of the service concerned. At completion of the processing of the command, the service produces a reply value. This reply is either T or F and is returned to the thread concerned.

Although BTA is one-sorted, we make this sort explicit. The reason for this is that we will extend BTA with an additional sort in Section 7.

The algebraic theory 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 *deadlock* 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 (see e.g. [27, 29]). Throughout the paper, we assume that there are infinitely many variables of sort $\mathbf{T}$, including x, y, z.

Table 7. Axiom of BTA

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

We use infix notation for postconditional composition. We introduce *action prefixing* as an abbreviation: $\alpha \circ p$, where p is a term of sort $\mathbf{T}$, abbreviates $p \trianglelefteq \alpha \trianglerighteq p$.

Let p and q be closed terms of sort $\mathbf{T}$ and $\alpha \in \mathcal{A}_{\mathsf{tau}}$. Then $p \trianglelefteq \alpha \trianglerighteq q$ will perform action α, and after that proceed as p if the processing of α leads to the reply T (called a positive reply), and proceed as q if the processing of α leads to the reply F (called a negative reply). The action tau plays a special role. It is a concrete internal action: performing tau will never lead to a state change and always lead to a positive reply, but notwithstanding all that its presence matters.

BTA has only one axiom. This axiom is given in Table 7. 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 of sort $\mathbf{T}$ denotes a finite thread, i.e. a thread of which the length of the sequences of actions that it can perform is bounded. Guarded recursive specifications give rise to infinite threads.

A *guarded recursive specification* over BTA is a set of recursion equations $E = \{X = p_X \mid X \in V\}$, 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 [3].

7 The Use Mechanism

A thread may perform certain actions for the purpose of interacting with some service provided by an execution environment. When processing an action performed by a thread, a service affects that thread by returning a reply value to the thread at completion of the processing of the action. In this section, we introduce the use mechanism, which is concerned with this kind of interaction between threads and services.[3]

It is assumed that a fixed but arbitrary finite set $\mathcal{F}$ of *foci* and a fixed but arbitrary finite set $\mathcal{M}$ of *methods* have been given. Each focus plays the role of a name of some service provided by an execution environment that can be requested to process a command. Each method plays the role of a command proper. For the set $\mathcal{A}$ of actions, we take the set $\{f.m \mid f \in \mathcal{F}, m \in \mathcal{M}\}$. Performing an action $f.m$ is taken as making a request to the service named f to process command m.

[3] This version of the use mechanism was first introduced in [7]. In later papers, the original use mechanism is also called thread-service composition.

Table 8. Axioms for use operators

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

We introduce yet another sort: the sort **S** of *services*. However, we will not introduce constants and operators to build terms of this sort. **S** is considered to stand for the set of all services. We identify services with functions $H : \mathcal{M}^+ \to \{\mathsf{T}, \mathsf{F}, \mathsf{B}\}$ that satisfy the following condition:

$$\forall \gamma \in \mathcal{M}^+, m \in \mathcal{M} \bullet (H(\gamma) = \mathsf{B} \Rightarrow H(\gamma \frown \langle m \rangle) = \mathsf{B}) .$$[4]

Given a service H and a method $m \in \mathcal{M}$, the *derived service* of H after processing m, written $\frac{\partial}{\partial m} H$, is defined by $\frac{\partial}{\partial m} H(\gamma) = H(\langle m \rangle \frown \gamma)$.

A service H can be understood as follows:

- if $H(\langle m \rangle) \neq \mathsf{B}$, then the request to process m is accepted by the service, the reply is $H(\langle m \rangle)$, and the service proceeds as $\frac{\partial}{\partial m} H$;
- if $H(\langle m \rangle) = \mathsf{B}$, then the request to process m is rejected by the service.

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, $p /_f H$ is the thread that results from processing all actions performed by thread p that are of the form $f.m$ by the service H. When an action of the form $f.m$ performed by thread p is processed by the service H, that action is turned into the internal action tau and postconditional composition is removed in favour of action prefixing on the basis of the reply value produced.

The axioms for the use operators are given in Table 8. In this table, f and g stand for an arbitrary foci from $\mathcal{F}$, m stands for an arbitrary method from $\mathcal{M}$, and H is a variable of sort **S**. Axioms TSU3 and TSU4 express that the action tau and actions of the form $g.m$, where $f \neq g$, are not processed. Axioms TSU5 and TSU6 express that a thread is affected by a service as described above when an action of the form $f.m$ performed by the thread is processed by the service.

[4] We write S^* for the set of all finite sequences with elements from set S and S^+ for the set of all non-empty finite sequences with elements from set S. We use the following notation for finite sequences: $\langle\, \rangle$ for the empty sequence, $\langle e \rangle$ for the sequence having e as sole element, $\sigma \frown \sigma'$ for the concatenation of finite sequences σ and σ', and $\mathrm{len}(\sigma)$ for the length of finite sequence σ.

Axiom TSU7 expresses that deadlock takes place when an action to be processed is not accepted.

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 8.

8 Thread Algebra and Data Linkage Dynamics Combined

The state changes and replies that result from performing the 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 the use mechanism such that the whole can be used for studying issues concerning the use of dynamic data structures in programming.

First, we introduce the state-based approach to describe families of services that will be used. This approach is similar to the approach to describe state machines introduced in [9].

In this approach, a family of services is described by

- a set of states S;
- an effect function $\mathit{eff} : \mathcal{M} \times S \to S$;
- a yield function $\mathit{yld} : \mathcal{M} \times S \to \{\mathsf{T}, \mathsf{F}, \mathsf{B}\}$;

satisfying the following conditions:

$$\begin{aligned} &\exists s \in S \bullet \forall m \in \mathcal{M} \bullet \\ &\quad (\mathit{yld}(m, s) = \mathsf{B} \land \forall s' \in S \bullet (\mathit{yld}(m, s') = \mathsf{B} \ \Rightarrow\ \mathit{eff}(m, s') = s)) \ . \end{aligned}$$

The set S contains the states in which the services may be, and the functions eff and yld give, for each method m and state s, the state and reply, respectively, that result from processing m in state s.

We define, for each $s \in S$, a cumulative effect function $\mathit{ceff}_s : \mathcal{M}^* \to S$ in terms of s and eff as follows:

$$\begin{aligned} &\mathit{ceff}_s(\langle\,\rangle) = s \ , \\ &\mathit{ceff}_s(\gamma \frown \langle m\rangle) = \mathit{eff}(m, \mathit{ceff}_s(\gamma)) \ . \end{aligned}$$

We define, for each $s \in S$, a service H_s in terms of ceff_s and yld as follows:

$$H_s = \mathit{yld}(m, \mathit{ceff}_s(\gamma)) \ .$$

H_s is called the service with *initial state* s described by S, eff and yld. We say that $\{H_s \mid s \in S\}$ is the *family of services* described by S, eff and yld.

The condition that is imposed on S, eff and yld implies that, for each $s \in S$, H_s is a service indeed. It is worth mentioning that $\frac{\partial}{\partial m} H_s = H_{\mathit{eff}(m,s)}$ and $H_s(\langle m\rangle) = \mathit{yld}(m, s)$.

We proceed with explaining how basic thread algebra can be combined with data linkage dynamics by means of the use mechanism 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.

Take $\mathcal{M}$ such that $A_{\mathrm{DLD}} \subseteq \mathcal{M}$. Let $L \in \mathcal{DL}$. Then we write $\mathcal{DLD}(L)$ for the service with initial state L described by $S = \mathcal{DL} \cup \{\uparrow\}$, where $\uparrow \notin \mathcal{DL}$, and the functions *eff* and *yld* defined as follows:

$$
\begin{array}{llll}
\mathit{eff}(m, L) = \mathit{eff}_m(L) & \text{if } m \in A_{\mathrm{DLD}}\ , & \mathit{yld}(m, L) = \mathit{yld}_m(L) & \text{if } m \in A_{\mathrm{DLD}}\ , \\
\mathit{eff}(m, L) = \uparrow & \text{if } m \notin A_{\mathrm{DLD}}\ , & \mathit{yld}(m, L) = \mathsf{B} & \text{if } m \notin A_{\mathrm{DLD}}\ , \\
\mathit{eff}(m, \uparrow) = \uparrow\ , & & \mathit{yld}(m, \uparrow) = \mathsf{B}\ . &
\end{array}
$$

By means of threads and the family of data linkage dynamics services introduced above, we can give a clear picture of computations in which dynamic data structures are involved.

In the following examples concerning computations of threads, we take the set $\{\underline{n} \mid n \in \{0, \dots, 9\}\}$ for AtObj. Moreover, we take the choice function ch such that $ch(A) = \underline{n}$ iff $\underline{n} \in A$ and there does not exist an $n' < n$ such that $\underline{n'} \in A$. In order to represent computations, we use the binary relation $\xrightarrow{\mathsf{tau}}$ on closed terms of $\mathrm{BTA}_{\mathrm{use}}$ defined by $p \xrightarrow{\mathsf{tau}} q$ iff $p = \mathsf{tau} \circ q$. Thus, $p \xrightarrow{\mathsf{tau}} q$ indicates that p can perform a concrete internal action and then proceed as q. Moreover, for each method $\alpha \in A_{\mathrm{DLD}}$, we write (α) instead of $\mathsf{dld}.\alpha$.

Example 3. We consider a simple thread in which non-value-related basic actions of DLD occur:

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

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

$$
\begin{array}{l}
((r\,!) \circ (t\,!) \circ (r/up) \circ (t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}(\emptyset) \xrightarrow{\mathsf{tau}} \\
((t\,!) \circ (r/up) \circ (t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}(\xrightarrow{r} \underline{0}) \xrightarrow{\mathsf{tau}} \\
((r/up) \circ (t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{r} \underline{0}) \oplus (\xrightarrow{t} \underline{1})) \xrightarrow{\mathsf{tau}} \\
((t/dn) \circ (r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{r} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{0} \xrightarrow{up})) \xrightarrow{\mathsf{tau}} \\
((r.up = t) \circ (t.dn = r) \circ (t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{r} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{0} \xrightarrow{up}) \oplus (\underline{1} \xrightarrow{dn})) \xrightarrow{\mathsf{tau}} \\
((t.dn = r) \circ (t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{r} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn})) \xrightarrow{\mathsf{tau}} \\
((t = *) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{r} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn} \underline{0})) \xrightarrow{\mathsf{tau}} \\
\mathsf{S} /_{\mathsf{dld}} \mathcal{DLD}((\xrightarrow{r} \underline{0}) \oplus (\underline{0} \xrightarrow{up} \underline{1}) \oplus (\underline{1} \xrightarrow{dn} \underline{0}))\ .
\end{array}
$$

Example 4. We 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 at the end of Section 3. We use the rewrite rules of DLD and axiom TSU5 of the use mechanism to obtain the following picture of a computation of this thread:

$$\begin{array}{l}
((u <= - u) \circ (s <= t + u) \circ (u <= - u) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{s} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{1})_7 \oplus (\xrightarrow{u} \underline{2}) \oplus (\underline{2})_3) \xrightarrow{\mathsf{tau}} \\
((s <= t + u) \circ (u <= - u) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{s} \underline{0}) \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{1})_7 \oplus (\xrightarrow{u} \underline{2}) \oplus (\underline{2})_{-3}) \xrightarrow{\mathsf{tau}} \\
((u <= - u) \circ \mathsf{S}) /_{\mathsf{dld}} \\
\qquad \mathcal{DLD}((\xrightarrow{s} \underline{0}) \oplus (\underline{0})_4 \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{1})_7 \oplus (\xrightarrow{u} \underline{2}) \oplus (\underline{2})_{-3}) \xrightarrow{\mathsf{tau}} \\
\mathsf{S} /_{\mathsf{dld}}\, \mathcal{DLD}((\xrightarrow{s} \underline{0}) \oplus (\underline{0})_4 \oplus (\xrightarrow{t} \underline{1}) \oplus (\underline{1})_7 \oplus (\xrightarrow{u} \underline{2}) \oplus (\underline{2})_3) \,.
\end{array}$$

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

A hierarchy of program notations rooted in PGA is presented in [5]. 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 [5], threads that are definable by finite guarded recursive specifications over BTA are taken as the behaviours of PGA programs. The combination of basic thread algebra and data linkage dynamics by means of the use mechanism 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 $\mathrm{PGLD}_{\mathsf{dld}}$ be an instance of PGLD in which all basic actions of DLD are available as basic instructions. For each program P from $\mathrm{PGLD}_{\mathsf{dld}}$, we write $|P|$ for the thread that is the behaviour of P according to [5]. 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 abovementioned 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 PGLD$_{\mathsf{dld}}$ to a program Q from PGLD$_{\mathsf{dld}}$ such that, for all natural numbers n, $|P| \,/_{\mathsf{dld}}\, \mathcal{DLD}^{n}_{\mathsf{afgc}}(\emptyset) = |Q| \,/_{\mathsf{dld}}\, \mathcal{DLD}^{c \cdot n + c'}(\emptyset)$?

9 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 actions of molecular dynamics that was given in [6]. In Section 10, we will show that the alternative description agrees with the description based on data linkage algebra.

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

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

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 Tables 9 and 10.[5] In these tables, $\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$ abbreviates $\sigma(s) \neq \bot$ and $\mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(s)$ abbreviates $\sigma(s) \neq \bot \land \xi(\sigma(s)) \neq \bot$.

[5] 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 $\mathrm{dom}(h) = \mathrm{dom}(f) \cup \mathrm{dom}(g)$ 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 $f \mathbin{\lhd\!\!\!-} S$ for the function g with $\mathrm{dom}(g) = \mathrm{dom}(f) \setminus S$ such that for all $e \in \mathrm{dom}(g)$, $g(e) = f(e)$.

Table 9. Definition of effect operations

$\mathit{eff}'_{s\,!}(\sigma,\zeta,\xi) =$	
$\quad(\sigma \dagger [s \mapsto \mathit{ch}(\mathsf{AtObj} \setminus \mathrm{dom}(\zeta))],$	
$\quad\;\zeta \dagger [\mathit{ch}(\mathsf{AtObj} \setminus \mathrm{dom}(\zeta)) \mapsto [\,]],$	
$\quad\;\xi \dagger [\mathit{ch}(\mathsf{AtObj} \setminus \mathrm{dom}(\zeta)) \mapsto \bot])$	if $\mathrm{dom}(\zeta) \subset \mathsf{AtObj}$
$\mathit{eff}'_{s\,!}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	if $\mathrm{dom}(\zeta) = \mathsf{AtObj}$
$\mathit{eff}'_{s=t}(\sigma,\zeta,\xi) = (\sigma \dagger [s \mapsto \sigma(t)],\zeta,\xi)$	
$\mathit{eff}'_{s=*}(\sigma,\zeta,\xi) = (\sigma \dagger [s \mapsto \bot],\zeta,\xi)$	
$\mathit{eff}'_{s==t}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s==*}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s/f}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \dagger [f \mapsto \bot]],\xi)$	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 $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \notin \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s\setminus f}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \lhd \{f\}],\xi)$	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 $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s\,\vert\,f}(\sigma,\zeta,\xi) = (\sigma,\zeta,\xi)$	
$\mathit{eff}'_{s.f=t}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta \dagger [\sigma(s) \mapsto \zeta(\sigma(s)) \dagger [f \mapsto \sigma(t)]],\xi)$	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 $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land f \in \mathrm{dom}(\zeta(\sigma(s))))$
$\mathit{eff}'_{s=t.f}(\sigma,\zeta,\xi) =$	
$\quad(\sigma \dagger [s \mapsto \zeta(\sigma(t))(f)],\zeta,\xi)$	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 $\neg\,\mathrm{def}^{\mathrm{o}}_{\sigma}(s)$
$\mathit{eff}'_{s<=t+u}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto \xi(\sigma(t)) + \xi(\sigma(u))])$	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 $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u))$
$\mathit{eff}'_{s<=t\cdot u}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto \xi(\sigma(t)) \cdot \xi(\sigma(u))])$	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 $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(u))$
$\mathit{eff}'_{s<=-\,t}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto -\xi(\sigma(t))])$	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 $\neg\,(\mathrm{def}^{\mathrm{o}}_{\sigma}(s) \land \mathrm{def}^{\mathrm{v}}_{\sigma,\xi}(t))$
$\mathit{eff}'_{s<=1/t}(\sigma,\zeta,\xi) =$	
$\quad(\sigma,\zeta,\xi \dagger [\sigma(s) \mapsto \xi(\sigma(t))^{-1}])$	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 10. Definition of yield operations

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

10 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 9 agrees with the one given in Section 4, provided only deterministic data linkages are considered. In the case where data linkages are taken for states, the effect operations preserve determinism. 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 9 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 [18]. 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:

$$
\begin{aligned}
&\mathit{retr}(\sigma, \zeta, \xi) = \\
&\quad \textstyle\bigoplus_{s \in \{s' \in \mathrm{dom}(\sigma) \mid \sigma(s') \neq \bot\}} (\xrightarrow{s} \sigma(s)) \oplus \\
&\quad \textstyle\bigoplus_{a \in \mathrm{dom}(\zeta)} \big(\bigoplus_{f \in \{f' \in \mathrm{dom}(\zeta(a)) \mid \zeta(a)(f') = \bot\}} (a \xrightarrow{f})\big) \oplus \\
&\quad \textstyle\bigoplus_{a \in \mathrm{dom}(\zeta)} \big(\bigoplus_{f \in \{f' \in \mathrm{dom}(\zeta(a)) \mid \zeta(a)(f') \neq \bot\}} (a \xrightarrow{f} \zeta(a)(f))\big) \oplus \\
&\quad \textstyle\bigoplus_{a \in \{a' \in \mathrm{dom}(\xi) \mid \xi(a') \neq \bot\}} (a)_{\xi(a)} \, .^{6}
\end{aligned}
$$

[6] We write $\bigoplus_{i \in \mathcal{I}} D_i$, where $\mathcal{I}$ is a finite set and D_i is a DLD term for each $i \in \mathcal{I}$, 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 $\oplus$). The convention is that $\bigoplus_{i \in \mathcal{I}} D_i$ stands for $\emptyset$ if $\mathcal{I} = \emptyset$.

This function can be thought of as regaining the abstract deterministic data linkages from their concrete representations by means of functions.

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

Theorem 3 (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:*

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

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

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

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:

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

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

11 Another Description of Garbage Reclamation

In this section, we describe of 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 and yield operations for reclamation on $\mathcal{DL}$, the effect and yield operations for reclamation on $\mathcal{DLR}$ are defined using auxiliary functions.

The auxiliary functions $reach : SS \times AS_1 \to \mathcal{P}(\mathsf{AtObj})$ and $incycle : AS_1 \to \mathcal{P}(\mathsf{AtObj})$ are defined as follows:

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

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

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

Table 11. Definition of effect operations for reclamation

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

The auxiliary functions $\mathit{sd}, \mathit{ud} : \mathsf{AtObj} \times \mathcal{DLR} \to \mathcal{DLR}$ are defined as follows:

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

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

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 Tables 11 and 12.

Theorem 3 goes through for $\mathrm{DLD}_{\mathrm{recl}}$. The additional cases to be considered involve proofs by induction over the definition of $\mathit{reach}(a,\zeta)$ for appropriate a and ζ.

12 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 rewrite system with rule priorities.

Table 12. Definition of yield operations for reclamation

$yld'_{\mathsf{fgc}}(\sigma, \zeta, \xi) = \mathsf{T}$
$yld'_{\mathsf{rgc}}(\sigma, \zeta, \xi) = \mathsf{T}$
$yld'_{s\,\backslash !}(\sigma, \zeta, \xi) = yld'_{s\,!}(\sigma, \zeta, \xi)$
$yld'_{s \backslash = t}(\sigma, \zeta, \xi) = yld'_{s = t}(\sigma, \zeta, \xi)$
$yld'_{s \backslash = *}(\sigma, \zeta, \xi) = yld'_{s = *}(\sigma, \zeta, \xi)$
$yld'_{s.f \backslash = t}(\sigma, \zeta, \xi) = yld'_{s.f = t}(\sigma, \zeta, \xi)$
$yld'_{s \backslash = t.f}(\sigma, \zeta, \xi) = yld'_{s = t.f}(\sigma, \zeta, \xi)$
$yld'_{s\,\backslash\backslash !}(\sigma, \zeta, \xi) = yld'_{s\,!}(\sigma, \zeta, \xi)$
$yld'_{s \backslash\backslash = t}(\sigma, \zeta, \xi) = yld'_{s = t}(\sigma, \zeta, \xi)$
$yld'_{s \backslash\backslash = *}(\sigma, \zeta, \xi) = yld'_{s = *}(\sigma, \zeta, \xi)$
$yld'_{s.f \backslash\backslash = t}(\sigma, \zeta, \xi) = yld'_{s.f = t}(\sigma, \zeta, \xi)$
$yld'_{s \backslash\backslash = t.f}(\sigma, \zeta, \xi) = yld'_{s = t.f}(\sigma, \zeta, \xi)$

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 [12] 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 particular, we managed to give a clear idea of the features related to reclamation of garbage.

We believe that the description of the dynamic aspects of the presented model of computation is rather sizable because of the start from first principles. This is substantiated by the size of the alternative description in the world of sets that we have given in this paper as well. We also believe that the use of conditional term rewriting [4] instead of priority rewriting would give rise to a less compact description.

The presented model of computation and its description are well-thought out, but the choices made can only be justified by applications. Together with thread algebra and program algebra, we hold the model of computation as described in this paper to be a suitable starting-point for investigations into issues concerning the interplay between programs and dynamic data structures, including issues concerning reclamation of garbage.

References

1. J. C. M. Baeten, J. A. Bergstra, J. W. Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *Theoretical Computer Science*, 67:283–301, 1989.
2. J. A. Bergstra and I. Bethke. Molecular dynamics. *Journal of Logic and Algebraic Programming*, 51:193–214, 2002.
3. J. A. Bergstra and I. Bethke. Polarized process algebra and program equivalence. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proceedings 30th ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 2003.

4. J. A. Bergstra and J. W. Klop. Conditional rewrite rules: Confluence and termination. *Journal of Computer and System Sciences*, 32:323–362, 1986.
5. J. A. Bergstra and M. E. Loots. Program algebra for sequential code. *Journal of Logic and Algebraic Programming*, 51(2):125–156, 2002.
6. J. A. Bergstra and C. A. Middelburg. A thread calculus with molecular dynamics. Computer Science Report 06-24, Department of Mathematics and Computer Science, Eindhoven University of Technology, August 2006.
7. J. A. Bergstra and C. A. Middelburg. Thread algebra for strategic interleaving. *Formal Aspects of Computing*, 19(4):445–474, 2007.
8. J. A. Bergstra and A. Ponse. Frame algebra with synchronous communication. In R. J. Wieringa and R. B. Feenstra, editors, *Information Systems – Correctness and Reusability*, pages 3–15. World Scientific, 1995.
9. J. A. Bergstra and A. Ponse. Combining programs and state machines. *Journal of Logic and Algebraic Programming*, 51(2):175–192, 2002.
10. J. A. Bergstra and A. Ponse. A generic basis theorem for cancellation meadows. `arXiv:0803.3969v2 [math.RA]` at `http://arxiv.org/`, March 2008.
11. J. A. Bergstra and J. V. Tucker. Equational specifications, complete term rewriting, and computable and semicomputable algebras. *Journal of the ACM*, 42(6):1194–1230, 1995.
12. J. A. Bergstra and J. V. Tucker. Elementary algebraic specifications of the rational complex numbers. In K. Futatsugi et al., editors, *Goguen Festschrift*, volume 4060 of *Lecture Notes in Computer Science*, pages 459–475. Springer-Verlag, 2006.
13. J. A. Bergstra and J. V. Tucker. The rational numbers as an abstract data type. *Journal of the ACM*, 54(2):Article 7, 2007.
14. I. Bethke and P. H. Rodenburg. Some properties of finite meadows. `arXiv:0712.0917v1 [math.RA]` at `http://arxiv.org/`, December 2007.
15. J. Bishop and N. Horspool. *C# Concisely*. Addison-Wesley, Reading, MA, 2004.
16. N. Dershowitz and J. P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. Elsevier, Amsterdam, 1990.
17. A. Hejlsberg, S. Wiltamuth, and P. Golde. *C# Language Specification*. Addison-Wesley, Reading, MA, 2003.
18. C.B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, second edition, 1990.
19. J. P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. *SIAM Journal of Computing*, 15(4):1155–1194, 1986.
20. J. P. Jouannaud and C. Marché. Termination and completion modulo associativity, commutativity and identity. *Theoretical Computer Science*, 104:29–51, 1992.
21. J. W. Klop. Term rewriting systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, Oxford, 1992.
22. J. W. Klop and R. de Vrijer. First-order term rewriting systems. In Terese, editor, *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*, pages 24–59. Cambridge University Press, Cambridge, 2003.
23. J. McCarthy. Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 3(4):184–195, 1960.
24. J. McCarthy, P. W. Abrahams, D. J. Edwards, T. P. Hart, and M. I. Levin. *LISP 1.5 Programmer's Manual*. MIT Press, Cambridge, MA, 1962.
25. C. K. Mohan. Priority rewriting: Semantics, confluence, and conditionals. In N. Dershowitz, editor, *RTA'89*, volume 355 of *Lecture Notes in Computer Science*, pages 278–291. Springer-Verlag, 1989.

26. M. Sakai and Y. Toyama. Semantics and strong sequentiality of priority term rewriting systems. *Theoretical Computer Science*, 208:87–110, 1998.
27. D. Sannella and A. Tarlecki. Algebraic preliminaries. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specification*, pages 13–30. Springer-Verlag, Berlin, 1999.
28. J. van der Pol. Operational semantics of rewriting with priorities. *Theoretical Computer Science*, 200:289–312, 1998.
29. M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 675–788. Elsevier, Amsterdam, 1990.

Electronic Reports Series of the Programming Research Group

Within this series the following reports appeared.

[PRG0805] J.A. Bergstra, S. Nolst Trenite, and M.B. van der Zwaag, *UvA Budget Allocatie Model,* Programming Research Group - University of Amsterdam, 2008.

[PRG0804] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Sequential Poly-Threading,* Programming Research Group - University of Amsterdam, 2008.

[PRG0803] J.A. Bergstra and C.A. Middelburg, *Thread Extraction for Polyadic Instruction Sequences,* Programming Research Group - University of Amsterdam, 2008.

[PRG0802] A. Barros and T. Hou, *A Constructive Version of AIP Revisited,* Programming Research Group - University of Amsterdam, 2008.

[PRG0801] J.A. Bergstra and C.A. Middelburg, *Programming an Interpreter Using Molecular Dynamics,* Programming Research Group - University of Amsterdam, 2008.

[PRG0713] J.A. Bergstra, A. Ponse, and M.B. van der Zwaag, *Tuplix Calculus,* Programming Research Group - University of Amsterdam, 2007.

[PRG0712] J.A. Bergstra, S. Nolst Trenite, and M.B. van der Zwaag, *Towards a Formalization of Budgets,* Programming Research Group - University of Amsterdam, 2007.

[PRG0711] J.A. Bergstra and C.A. Middelburg, *Program Algebra with a Jump-Shift Instruction,* Programming Research Group - University of Amsterdam, 2007.

[PRG0710] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences with Dynamically Instantiated Instructions,* Programming Research Group - University of Amsterdam, 2007.

[PRG0709] J.A. Bergstra and C.A. Middelburg, *Instruction Sequences with Indirect Jumps,* Programming Research Group - University of Amsterdam, 2007.

[PRG0708] B. Diertens, *Software (Re-)Engineering with PSF III: an IDE for PSF,* Programming Research Group - University of Amsterdam, 2007.

[PRG0707] J.A. Bergstra and C.A. Middelburg, *An Interface Group for Process Components,* Programming Research Group - University of Amsterdam, 2007.

[PRG0706] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Skew Meadows,* Programming Research Group - University of Amsterdam, 2007.

[PRG0705] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker, *Meadows,* Programming Research Group - University of Amsterdam, 2007.

[PRG0704] J.A. Bergstra and C.A. Middelburg, *Machine Structure Oriented Control Code Logic (Extended Version),* Programming Research Group - University of Amsterdam, 2007.

[PRG0703] J.A. Bergstra and C.A. Middelburg, *On the Operating Unit Size of Load/Store Architectures,* Programming Research Group - University of Amsterdam, 2007.

[PRG0702] J.A. Bergstra and A. Ponse, *Interface Groups and Financial Transfer Architectures,* Programming Research Group - University of Amsterdam, 2007.

[PRG0701] J.A. Bergstra, I. Bethke, and M. Burgess, *A Process Algebra Based Framework for Promise Theory,* Programming Research Group - University of Amsterdam, 2007.

[PRG0610] J.A. Bergstra and C.A. Middelburg, *Parallel Processes with Implicit Computational Capital,* Programming Research Group - University of Amsterdam, 2006.

[PRG0609] B. Diertens, *Software (Re-)Engineering with PSF II: from architecture to implementation,* Programming Research Group - University of Amsterdam, 2006.

[PRG0608] A. Ponse and M.B. van der Zwaag, *Risk Assessment for One-Counter Threads,* Programming Research Group - University of Amsterdam, 2006.

[PRG0607] J.A. Bergstra and C.A. Middelburg, *Synchronous Cooperation for Explicit Multi-Threading,* Programming Research Group - University of Amsterdam, 2006.

[PRG0606] J.A. Bergstra and M. Burgess, *Local and Global Trust Based on the Concept of Promises,* Programming Research Group - University of Amsterdam, 2006.

[PRG0605] J.A. Bergstra and J.V. Tucker, *Division Safe Calculation in Totalised Fields,* Programming Research Group - University of Amsterdam, 2006.

[PRG0604] J.A. Bergstra and A. Ponse, *Projection Semantics for Rigid Loops,* Programming Research Group - University of Amsterdam, 2006.

[PRG0603] J.A. Bergstra and I. Bethke, *Predictable and Reliable Program Code: Virtual Machine-based Projection Semantics (submitted for inclusion in the Handbook of Network and Systems Administration),* Programming Research Group - University of Amsterdam, 2006.

[PRG0602] J.A. Bergstra and A. Ponse, *Program Algebra with Repeat Instruction,* Programming Research Group - University of Amsterdam, 2006.

[PRG0601] J.A. Bergstra and A. Ponse, *Interface Groups for Analytic Execution Architectures,* Programming Research Group - University of Amsterdam, 2006.

[PRG0505] B. Diertens, *Software (Re-)Engineering with PSF,* Programming Research Group - University of Amsterdam, 2005.

[PRG0504] P.H. Rodenburg, *Piecewise Initial Algebra Semantics,* Programming Research Group - University of Amsterdam, 2005.

[PRG0503] T.D. Vu, *Metric Denotational Semantics for BPPA,* Programming Research Group - University of Amsterdam, 2005.

[PRG0502] J.A. Bergstra, I. Bethke, and A. Ponse, *Decision Problems for Pushdown Threads,* Programming Research Group - University of Amsterdam, 2005.

[PRG0501] J.A. Bergstra and A. Ponse, *A Bypass of Cohen's Impossibility Result,* Programming Research Group - University of Amsterdam, 2005.

[PRG0405] J.A. Bergstra and I. Bethke, *An Upper Bound for the Equational Specification of Finite State Services,* Programming Research Group - University of Amsterdam, 2004.

[PRG0404] J.A. Bergstra and C.A. Middelburg, *Thread Algebra for Strategic Interleaving,* Programming Research Group - University of Amsterdam, 2004.

[PRG0403] B. Diertens, *A Compiler-projection from PGLEc.MSPio to Parrot,* Programming Research Group - University of Amsterdam, 2004.

[PRG0402] J.A. Bergstra and I. Bethke, *Linear Projective Program Syntax,* Programming Research Group - University of Amsterdam, 2004.

[PRG0401] B. Diertens, *Molecular Scripting Primitives,* Programming Research Group - University of Amsterdam, 2004.

[PRG0302] B. Diertens, *A Toolset for PGA,* Programming Research Group - University of Amsterdam, 2003.

[PRG0301] J.A. Bergstra and P. Walters, *Projection Semantics for Multi-File Programs,* Programming Research Group - University of Amsterdam, 2003.

[PRG0201] I. Bethke and P. Walters, *Molecule-oriented Java Programs for Cyclic Sequences,* Programming Research Group - University of Amsterdam, 2002.

The above reports and more are available through the website: www.science.uva.nl/research/prog/

Electronic Report Series

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ Amsterdam
the Netherlands

www.science.uva.nl/research/prog/