Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 225 (2009) 341–360
www.elsevier.com/locate/entcs

A Random Bag Preserving Product Operation

Michel Schellekens [1,2]

Department of Computer Science/CEOL
National University of Ireland, Cork[3]

Abstract

The author's current research programme is the development of a modular calculus for the average-cost of data structuring. This modular calculus provides a novel foundation for the analysis of algorithms. Its applicability to the analysis of algorithms has been demonstrated at the Center for Efficiency-Oriented Languages (CEOL) through the design of the novel programming language $\mathcal{MOQA}$ and the associated average-case analysis tool DISTRI-TRACK [8,4,5,2,3]. Modular computations of the average cost of data structuring are possible through the fundamental notion of random bag preservation. Random bag preserving operations enable the constructive tracking of the data *and* the distribution of the data states during a $\mathcal{MOQA}$ computation. This in turn enables the (semi-)automated derivation of the average cost of the operations. Two fundamental $\mathcal{MOQA}$ operations enable the creation and destruction of data structures: the $\mathcal{MOQA}$ product operation, which is the subject of this paper, and the $\mathcal{MOQA}$ delete operation, which forms the subject of [3]. The introduction of the entire $\mathcal{MOQA}$ language is well beyond the scope of this paper and will be reported in a book [2]. The language has been implemented at CEOL and automated derivations of average-cost of data structuring are under way. Here we report on a (simplified) version of the fundamental notion of random bag preservation and demonstrate that the central $\mathcal{MOQA}$ product operation possesses this crucial property.

Keywords: Algorithms, Random Bags, Compositionality, Modularity, Languages.

1 Introduction

$\mathcal{MOQA}$ is a domain specific high-level language. The language has extensive programming capacity in the sense that it includes for-loops, (terminating) recursion and conditionals. This approach enables the programming of a wide variety of data restructuring algorithms, including most sorting and searching algorithms.

The crucial property of $\mathcal{MOQA}$ is that it preserves "regularity" in a certain way during its operations. This regularity is captures by the notions of random structures and random bags. All $\mathcal{MOQA}$ operations are guaranteed to preserve

[1] Science Foundation Ireland Principal Investigator, 07/IN.1/I977
[2] Email: m.schellekens@cs.ucc.ie
[3] Centre for Efficiency-Oriented Languages

doi:10.1016/j.entcs.2008.12.085

random bags. This enables the tracking of data structures during the computations, which in turn facilitates the modular derivation of the average-case time of $\mathcal{MOQA}$ programs.

We focus on the product operation in this paper since the operation serves two purposes: first it enables one to insert an element ("data structure of cardinality one") into a data structure and second, in general, it enables the merging of two data structures into a larger one. Such an operation arises for instance in Insertion Sort or Merge sort. The operation is formulated in such a way here that it applies to data structures determined by arbitrary finite partial orders, incorporating a very wide variety of data structures.

2 States and Random Structures

We proceed with formal definitions. The first one defines the concept of a state. Note that $\mathcal{L}$ forms a set of labels, which we consider to be a subset of the natural numbers equipped with their standard order, $(\mathcal{N}, \leq)$.

Definition 2.1 A *labeling* of a finite partial order $(X, \sqsubseteq)$ from the countable set of labels $\mathcal{N}$ is an increasing injection from X to $\mathcal{N}$, paired with the partial order $(X, \sqsubseteq)$. A *state* of a finite partial order $(X, \sqsubseteq)$ *from a set of labels* $\mathcal{L}$, where $|X| = |\mathcal{L}|$, is an increasing injection $F\colon X \to \mathcal{L}$, paired with the partial order $(X, \sqsubseteq)$.

Note that we consider two labelings, say $(F_1, (X, \sqsubseteq_1))$ and $(F_2, (X_2, \sqsubseteq_2))$, to be different when their underlying partial orders differ. This includes the case where $X_1 = X_2$ and where $\sqsubseteq_1$ and $\sqsubseteq_2$ differ, but where the functions F_1 and F_2 coincide. In practice, and with abuse of notation, when the underlying partial order is unambiguous, we will refer to a labeling F as opposed to a state $(F, (X, \sqsubseteq))$.

Of course, it follows from the above definition that states are bijections from X to $\mathcal{L}$.

Omitting the order in the following notations consists of a slight abuse of notation, which will not cause ambiguities in the work. Let $(X, \sqsubseteq)$ be a finite partial order. We let $m(Y)$ denote the set of minimal elements of $(Y, \sqsubseteq)$ and $M(Y)$ denote the set of maximal elements of $(Y, \sqsubseteq)$. Let F be a state of this partial order. We let $m(F)$ denote the labels for F of minimal elements of $(X, \sqsubseteq)$, i.e. $F(m(X))$, and we let $M(F)$ denote the labels for F of maximal elements of $(X, \sqsubseteq)$, i.e. $F(M(X))$. For any subset $\mathcal{A}$ of the set of labels $\mathcal{L}$, we let $m(\mathcal{A})$ denote the labels in $\mathcal{A}$ of minimal elements of $(F^{-1}(\mathcal{A}), \sqsubseteq)$, i.e. $F(m(F^{-1}(\mathcal{A})))$, and we let $M(\mathcal{A})$ denote the labels for F of maximal elements of $(F^{-1}(\mathcal{A}), \sqsubseteq)$, i.e. $F(M(F^{-1}(\mathcal{A})))$. Finally we use the following notation: $\vee\mathcal{A}$ denotes the maximum label of the set $\mathcal{A}$ while $\wedge\mathcal{A}$ denotes the minimum label of $\mathcal{A}$.

Remark 2.2 *It is quite evident that the greatest (least) label of a state must occur at a maximal (minimal) element.*

Definition 2.3 The *Random Structure* on a finite partial order $(X, \sqsubseteq)$, with respect to a set of labels $\mathcal{L}$ where $|X| = |\mathcal{L}|$, is the set of all states from $\mathcal{L}$ of the partial

order. We denote this random structure by: $\mathcal{R}_{\mathcal{L}}(X, \sqsubseteq)$.

Notation: We frequently denote a random structure $\mathcal{R}_{\mathcal{L}}(X, \sqsubseteq)$ by R and in that case refer to the underlying set X and set of labels $\mathcal{L}$ as X_R and $\mathcal{L}_R$.

We remark that the definition of a random structure does *not* require the underlying partial order to be connected.

Remark 2.4 *Random structures, $\mathcal{R}_{\mathcal{L}_1}(X, \sqsubseteq)$ and $\mathcal{R}_{\mathcal{L}_2}(X, \sqsubseteq)$, of a given partial order $(X, \sqsubseteq)$ and obtained for two different sets of labels, $\mathcal{L}_1$ and $\mathcal{L}_2$, can easily be seen to be* label-isomorphic*, i.e. there exists an order preserving bijection $\Psi_{(\mathcal{L}_1,\mathcal{L}_2)}$ from the linear order $(\mathcal{L}_1, \leq)$ to the linear order $(\mathcal{L}_2, \leq)$, where $\leq$ is the usual order on the natural numbers, such that $\mathcal{R}_{\mathcal{L}_2}(X, \sqsubseteq) = \{\Psi_{(\mathcal{L}_1,\mathcal{L}_2)} \circ F | \, F \in \mathcal{R}_{\mathcal{L}_1}(X, \sqsubseteq)\}$. So if $\mathcal{L}_1 = \{a_1, \ldots, a_n\}$ and $\mathcal{L}_2 = \{b_1, \ldots, b_n\}$ where $\forall i \in \{1, \ldots, n-1\}. \, a_i < a_{i+1}$ and $b_i < b_{i+1}$ then $\forall i \in \{1, \ldots, n\}. \, \Psi_{(\mathcal{L}_1,\mathcal{L}_2)}(a_i) = b_i$. We refer to the unique equivalence class for the equivalence relation "label-isomorphic" as* the *random structure $\mathcal{R}(X, \sqsubseteq)$ of a partial order $(X, \sqsubseteq)$*, where the label set $\mathcal{L}$ is no longer indicated.

It is easy to see that random structures allow one to incorporate traditional labeled data structures, such as heaps, unordered lists and sorted lists, as long as the labelings respect the underlying order. We illustrate this in the next example, where for each data structure, i.e. partial order, all possible data states are represented.

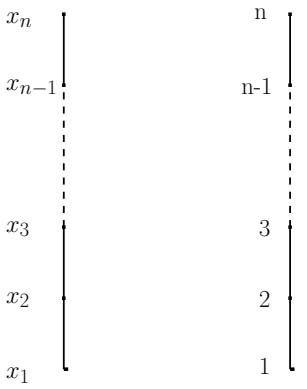
Example 2.5 *In each part of the example, we display the Hasse Diagram of the given partial order on the left and the states on the right. In each case the underlying set consists of elements $\{x_1, \ldots, x_n\}$, while the labels are the set of indices $\{1, \ldots, n\}$. Part a) illustrates that random structures incorporate the case of lists in a natural way.*

a) Consider the partial order $(X, \sqsubseteq)$ over the set $\{x_1, x_2, \ldots, x_n\}$ equipped with the discrete order. The random structure $\mathcal{R}_{\mathcal{L}}(X, \sqsubseteq)$ consists of all $n!$ permutations of labels on the elements of X and can be interpreted as the set of lists of size n. We will denote in the following such a random structure by $\mathcal{A}_n$ where $\mathcal{A}$ stands for "Atomic".

$$\begin{array}{cccccc} \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ x_1 & x_2 & x_3 & & x_{n-1} & x_n \end{array}$$

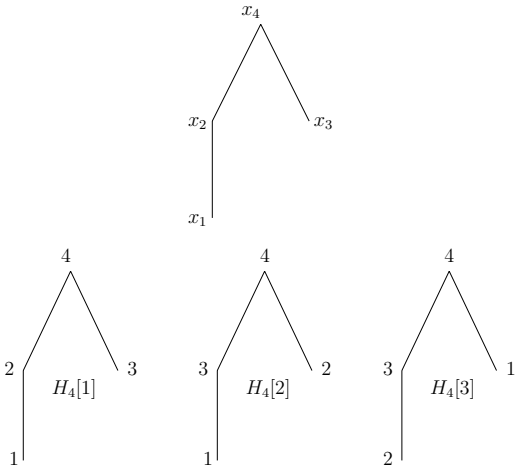
The probability for labelings to be in one of these states is $\frac{1}{n!}$.

b) Consider the partial order $(X, \sqsubseteq)$ over the set $\{x_1, x_2, \ldots, x_n\}$ equipped with a linear order. The random structure $\mathcal{R}_{\mathcal{L}}(X, \sqsubseteq)$ consists of a single state, denoted by $\mathcal{S}_n$, which can be interpreted as the sorted list.



The probability for a labeling over this partial order to be in this state is 1.

c) Finally, we remark that heaps can be represented as random structures over a partial order which has a tree as Hasse Diagram. Heaps of size n are denoted by $\mathcal{H}_n$. For instance, the random structure $\mathcal{H}_3$ determined by the following Hasse Diagram and label set $\{1,2,3,4\}$ consists exactly of the states $H_4[1], H_4[2]$ and $H_4[3]$ displayed below.



The probability for a labeling to be in one of these states is $\frac{1}{3}$.

Similarly, Heap Ordered Trees in general can be represented in this way.

It is obvious that the cardinality of Random Structures over partial orders with n elements lies between 1 and $n!$ included.

3 Floor and Ceiling Functions

We introduce "floor" and "ceiling" functions for elements of partial orders, which will be useful to specify the pseudo-code for our product operation. For a partial

order $(X, \sqsubseteq)$ and an element $x \in X$, we define $\lceil x \rceil$ to be the set of all elements immediately and strictly above x, i.e. $\lceil x \rceil = \{y | \, y \sqsupset_1 x\}$. Similarly, we define: $\lfloor x \rfloor = \{y | \, y \sqsubset_1 x\}$. For a discrete subset Y of X, we define: $\lceil Y \rceil = \cup_{y \in Y} \lceil y \rceil$ and $\lfloor Y \rfloor = \cup_{y \in Y} \lfloor y \rfloor$. Given a state F with range $\mathcal{L}$, the floor and ceiling of a label $a \in \mathcal{L}$ and of a set of labels, is defined as follows: $\lceil a \rceil = F(\lceil F^{-1}(a) \rceil)$, $\lfloor a \rfloor = F(\lfloor F^{-1}(a) \rfloor)$. For a subset $\mathcal{A}$ of $\mathcal{L}$ we define: $\lceil \mathcal{A} \rceil = F(\lceil F^{-1}(\mathcal{A}) \rceil)$, $\lfloor \mathcal{A} \rfloor = F(\lfloor F^{-1}(\mathcal{A}) \rfloor)$. Of course: $a \in \lceil b \rceil \Rightarrow a > b$ and $a \in \lfloor b \rfloor \Rightarrow a < b$.

4 Random Structure and Random Bag preserving functions

The notion of Random Structure Preservation involves the notion of a random bag which is defined below.

Definition 4.1 A random bag is a finite bag of pairs, $\{(R_1, K_1), \ldots, (R_n, K_n)\}$, each of which consists of a random structure R paired with a multiplicity K.

Remark 4.2 In case $n = 1$ and $K = 1$ we identify the random bag with the random structure R_1. i.e. we interpret a random structure in the context of random bags as a random bag of size one and multiplicity one.

We will define operations that transform a random structure $\mathcal{R}_{\mathcal{L}}(X, \sqsubseteq)$ into a bag of random structures $\{(\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1), K_1), \ldots, (\mathcal{R}_{\mathcal{L}_n}(X_n, \sqsubseteq_n), K_n)\}$, where

$$\forall i \in \{1, \ldots, n\}. \, \mathcal{L}_i \subseteq \mathcal{L}.$$

Such operations will be called random bag preserving operations, or RB-preserving operations. The label sets $\mathcal{L}_i$ are subsets of the original label set $\mathcal{L}$ since the deletion operation which we will consider may remove some labels.

We introduce the notion of a *refinement* in the following. Random Bag Preserving operations "refine" the original partial order in that the newly created partial orders of the resulting collection have underlying sets X_i that are subsets of the original set X and have orders $\sqsubseteq_i$ that are finer than, i.e. include, the restriction of the original partial order $\sqsubseteq$ to the new set X_i under consideration. We formalize this below.

Definition 4.3 Let $R = \mathcal{R}_{\mathcal{L}}(X, \sqsubseteq)$ and $\forall i \in \{1, \ldots, n\}. \, R_i = \mathcal{R}_{\mathcal{L}_i}(X_i, \sqsubseteq_i)$, where $\forall i \in \{1, \ldots, n\}. \, \mathcal{L}_i \subseteq \mathcal{L}$ and $\forall i \in \{1, \ldots, n\}. \, X_i \subseteq X$ and $\forall x, y \in X_i. \, x \sqsubseteq y \Rightarrow x \sqsubseteq_i y$. We call any collection of random structures $\{R_1, \ldots, R_n\}$ satisfying this condition *a refinement* of the random structure R. We also refer to $\mathcal{L}_i$ as a refinement of the label set $\mathcal{L}$ and to each $(X_i, \sqsubseteq_i)$ as a refinement of the partial order $(X, \sqsubseteq)$.

Remark 4.4 (For the Semantics oriented reader) One can verify that the class of Random Bags with the refining order form a CPO.

Summary 1 We use the following notation: $\mathcal{U}$, referred to as the *universe*, is a countable list of variables, say $\mathcal{U} = \{u_n | \, n \in \mathcal{N}\}$. We denote the set of all finite

partial orders over $\mathcal{U}$ by

$$\mathcal{PO}_{fin}(\mathcal{U}) = \{(X, \sqsubseteq) |\, X \subseteq \mathcal{U} \text{ and } (X, \sqsubseteq) \text{ is a finite partial order.}\}.$$

The set of all labelings over partial orders from $\mathcal{PO}_{fin}(\mathcal{U})$ is denoted by $\mathcal{F}$, i.e.:

$$\mathcal{F} = \{F |\, F\colon (X, \sqsubseteq) \to \mathcal{N}, (X, \sqsubseteq) \in \mathcal{PO}_{fin}(\mathcal{U}) \text{ and } F \text{ is a labeling}\}.$$

Definition 4.5 A function $\phi\colon \mathcal{F} \to \mathcal{F}$ is *refining on* R if there exists a refinement $\{R_1, \ldots, R_n\}$ of R such that $\phi\colon R \to R_1 \cup \ldots \cup R_n$ is surjective.

The operations we will consider typically transform random structures R into a refinement $\{R_1, \ldots, R_n\}$ of R; more precisely they determine refining functions.

Definition 4.6 In case we have determined a refinement $\{R_1, \ldots, R_n\}$ of R, based on which we can establish that the function ϕ is refining on R, then we refer to ϕ in combination with this particular selection of a refinement as a *representation for* ϕ. Such a representation is denoted as follows: $\phi\colon R \to \{R_1, \ldots, R_n\}$.

The following definition formalizes the notion of Random Structure Preservation.

Definition 4.7 A function $\mu\colon \mathcal{F} \to \mathcal{F}$ is *Random Structure preserving on a random structure R (RS-preserving on a random structure R)* iff there exists a partition $\mathcal{F}_1, \ldots, \mathcal{F}_n$ of R, a refinement $\{R_1, \ldots, R_n\}$ of R and non-zero natural numbers $K_1, \ldots, K_n$ such that

$$\forall F \in R_i . |\mu^{-1}(F) \cap \mathcal{F}_i| = K_i.$$

The function μ is *RS-preserving* iff it is RS-preserving on every random structure.

The function μ is called *strongly RS-preserving* if and only if $n = 1$.

Note that we will demonstrate that the product operation is *strongly* RS-preserving.

Remark 4.8 *1) Since refining functions are surjective, we have in the above definition automatically that for each $i \in \{1, \ldots, n\}$. $\mu(\mathcal{F}_i) = R_i$.*
2) The definition of RS-preservation is more general than the informal use of randomness preservation in the literature. The informal use of randomness preservation only regards the preservation of the uniform distribution, where a random structure is mapped to a single random structure, as is the case for the Backwards Analysis of [7] and for the cases discussed in [6], and no non-trivial multiplicity is involved (i.e. $K = 1$). This is captured in our context by the notion of a strongly RS-preserving function with multiplicity one. Representations of RS-preserving functions in our context, map a random structure to a bag of random structures.
3) When we have a particular representation in mind for an RS-preserving function, we will, with abuse of terminology, refer to the image of the RS-preserving function as a random bag. In practice of course an RS-preserving function could have more than one representation.

Remark 4.9 *It is clear that the definition of RS-preservation could be simplified in case the random structures $R_1, \ldots, R_n$ have pairwise disjoint underlying partial orders. In that case the definition is equivalent to the following:*

$$\forall F \in R_i.\, |\mu^{-1}(F)| = K_i.$$

Of course, one can always guarantee that the random bag $\{R_1, \ldots, R_n\}$ is such that the underlying partial orders are pairwise disjoint by identifying random structures with the same, i.e. order-isomorphic, underlying partial orders and by adjusting the multiplicities accordingly. We prefer to keep the more general version of RS-preservation at this time, since identification of order-isomorphic partial orders in practice in general is costly and the time analysis may not require such an identification.

Definition 4.10 In case we have determined a refinement $\{R_1, \ldots, R_n\}$ of R with multiplicities $K_1, \ldots, K_n$ with respect to some partition $\mathcal{F}_1, \ldots, \mathcal{F}_n$, based on which we can establish that the function μ is RS-preserving on R, then we refer to μ in combination with this particular selection of a refinement, partition and multiplicities as an *RS-representation for* μ. Such an RS-representation for μ is denoted as follows:

$$\mu_{(\mathcal{F}_1,\ldots,\mathcal{F}_n)}\colon R \to \{(R_1, K_1), \ldots, (R_n, K_n)\}.$$

Summary 2 Typically, and with some abuse of notation, we will not mention the partition involved for RS-representations:

$$\mu\colon R \to \{(R_1, K_1), \ldots, (R_n, K_n)\}.$$

The motivation behind this shorter notation is that once our choice for the refining collection, the partition and the corresponding multiplicities have been determined, we only need the resulting random bag in order to determine the average-case time.

5 The Random Product

The random product is a fundamental $\mathcal{MOQA}$ data structuring operation which enables the joining of two data structures into a larger data structure. Here our aim is to illustrate that a random bag preserving product operation can be obtained and we present a proof of this result. Research is ongoing at CEOL on alternative versions of the product and efficiency comparisons. The techniques for verifying random bag preservation as outlined below are however standard approaches and serve to illustrate that the most common version of the product satisfies this property.

In order to define the random product, we first define the product of two finite partial orders. The definition is similar to the one given in [1]. Then we define the product of two labelings and we extend this definition to sets of labelings. Finally,

we define the random product on a random structure as a unary operation, which performs an operation on two substructures of the given random structure and reproduces a new random structure.

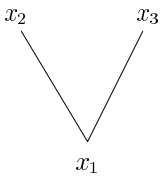
5.1 The product of two finite partial orders

Definition 5.1 Given two finite *disjoint* partial orders $(X_1, \sqsubseteq_1)$ and $(X_2, \sqsubseteq_2)$.

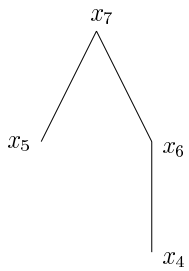
The set $X_1 \bigotimes X_2$ is defined to be the union of the disjoint sets X_1 and X_2. The relation $\sqsubseteq_1 \bigotimes \sqsubseteq_2$ is defined to be the least partial order on $X_1 \bigotimes X_2$ containing $\sqsubseteq_1$ and $\sqsubseteq_2$ and $X_1 \times X_2$.

It is easy to verify that the partial order $\sqsubseteq_1 \bigotimes \sqsubseteq_2$ is the transitive closure of the binary relations $\sqsubseteq_1$, $\sqsubseteq_2$ and the set of pairs $\{(M, m) | \, M$ is a maximal element of $(X_1, \sqsubseteq_1)$, m is a minimal element of $(X_2, \sqsubseteq_2)\}$.

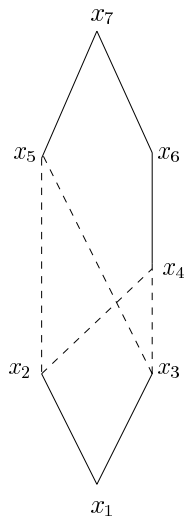
Example 5.2 *If we consider the sets* $X_1 = \{x_1, x_2, x_3\}$ *and* $X_2 = \{x_4, x_5, x_6, x_7\}$ *then* $X_1 \bigotimes X_2 = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$. *We indicate the new pairs added via the operation* $\bigotimes$ *via dashed lines.*



$(X_1, \sqsubseteq_1)$



$(X_2, \sqsubseteq_2)$



$(X_1 \otimes X_2, \sqsubseteq_1 \otimes \sqsubseteq_2)$

We define the product of two labelings as a first step towards the definition of the random product of two random structures.

5.2 The product of two labelings

Let F_1,F_2 be labelings on finite partial orders $(X_1, \sqsubseteq_1)$ and $(X_2, \sqsubseteq_2)$ respectively. We call F_1 and F_2 *disjoint* when their domains X_1 and X_2 are disjoint and their ranges $F_1(X_1)$ and $F_2(X_2)$ are disjoint.

Pseudo-code for the product $\bigotimes$ on labelings
Let F_1, F_2 be disjoint labelings which are provided as inputs.

We define the product of the two labelings. To avoid technicalities, we assume in the following pseudo-code that the labelings F_1 and F_2 of which the product is taken are (implicitly) processed first to retrieve a new function F, consisting of the join of the labelings F_1 and F_2. The creation of F will be indicated in the final pseudo-code for the random product by the initial code line: $F = F_1 \cup F_2$, where we consider the graph union of these functions.

We will also assume the implicit generation of the restrictions of this function F, i.e. $F \restriction X_1$ and $F \restriction X_2$, to the sets X_1 and X_2 respectively and hence won't specify the detailed implementation of these restrictions in the pseudo code. The function F and its restrictions $F \restriction X_1$ and $F \restriction X_2$ will freely be referred to in the pseudo-code for P.

The pseudo-code to generate a labeling from $F = F_1 \cup F_2$ is based on a generalization of the procedures Push-Down and Push-Up used in the pseudo-code of the Heapsort Algorithm in Section 2. We will provide pseudo-code for Williams versions of the Push operations and remark that it is straightforward to specify Floyd versions of these procedures [4]. We omit the details but will refer to these generalizations as F-Push-Down and F-Push-Up in the following. We provide pseudo-code for generalized versions of Williams' Push operations:

W-Push-Down(b, F)
while $\lfloor b \rfloor \neq \emptyset$ **and** $b < \vee\lfloor b \rfloor$
  swap$(b, \vee\lfloor b \rfloor, F)$

W-Push-Up(a, F)
while $\lceil a \rceil \neq \emptyset$ **and** $a > \wedge\lceil a \rceil$
  swap$(a, \wedge\lceil a \rceil, F)$

As before, we will use Push-Down and Push-Up freely in the pseudo-code, without specifying which version we use since this is a matter of choice of implementation.

We provide the pseudo-code for the Labeling-Product Algorithm where the inputs for the algorithm are the disjoint labelings F_1 and F_2. We denote the function F returned by the Labeling-Product algorithm as $F_1 \bigotimes F_2$.

[4] Cf. [9] for a discussion of both versions.

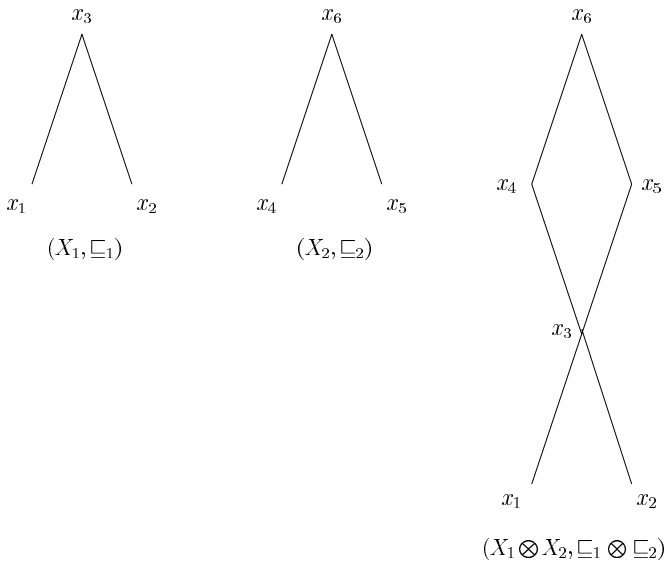
Pseudo-code for the Labeling-Product Algorithm

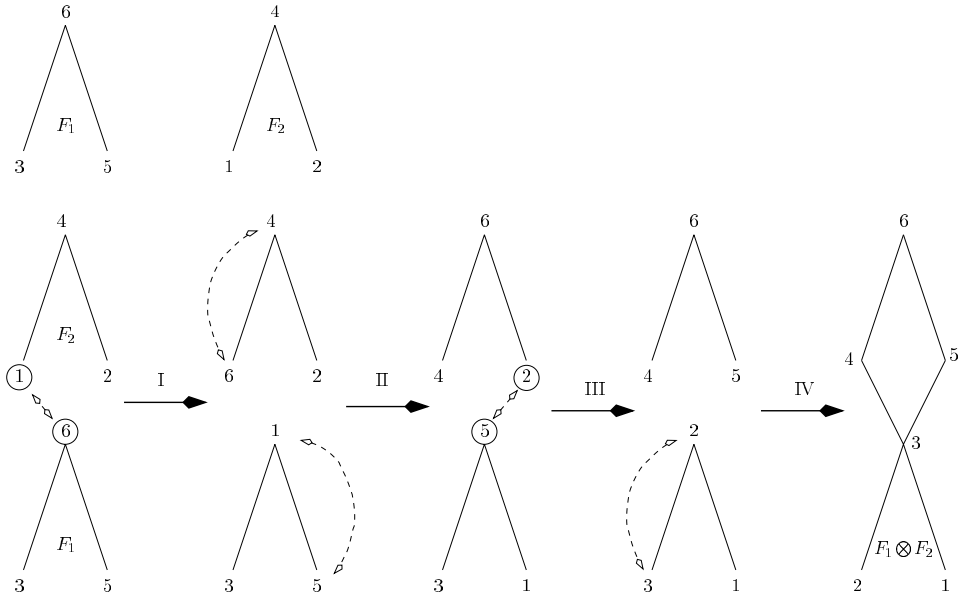
$F := F_1 \cup F_2$;
while $\vee M(F \restriction X_1) > \wedge m(F \restriction X_2)$ **do**
  $a := \vee M(F \restriction X_1); b := \wedge m(F \restriction X_2)$;
  swap (a, b, F);
  Push-Down(b, F);
  Push-Up(a, F)
Return F

Lemma 5.3 *If F_1 and F_2 are disjoint labelings then $F_1 \bigotimes F_2$ is a labeling.*

Proof. This follows via straightforward yet technically lengthy verifications from the pseudo-code of the random product algorithm. We omit the details. □

Example 5.4 *In the example given below, we consider two labelings F_1 and F_2 for the partial orders displayed below and illustrate the steps involved in executing the Labeling-Product algorithm.*





We indicated the selection of labels of extremal elements by full circles and these elements occur swapped in the following picture. For each while loop execution, initiated by an original swap of labels of two extremal elements, the other pairs of elements to be swapped are linked in the picture via a double arrow (in dashed line display). These elements occur swapped in the picture. The final picture illustrates the end result of the computation, i.e. $F_1 \bigotimes F_2$.

Definition 5.5 Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be disjoint sets of labels. The *label-product function*

$$\bigotimes : \mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1) \times \mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2) \to \mathcal{R}_{\mathcal{L}_1 \cup \mathcal{L}_2}(X_1 \otimes X_2, \sqsubseteq_1 \otimes \sqsubseteq_2)$$

is defined by: $\bigotimes(F_1, F_2) = F_1 \bigotimes F_2$.

The following result is important to obtain that the Random Product is an RS-preserving operation.

Theorem 5.6 *The label-product function is a bijection.*

Proof. Consider two disjoint partial orders $(X_1, \sqsubseteq_1)$ and $(X_2, \sqsubseteq_2)$.

We present a proof for Williams's versions of the Push operations. The proof for Floyd's version is similar.

We view the execution of the labeling product algorithm as a series of swaps along chains of $X_1 \bigotimes X_2$. For a given pair of disjoint labelings, F_1 and F_2, each such chain is determined by a single run of the two push operations in the code of the random product. We recall that at the start of the while loops, labels a and b are involved in the swaps, where in terms of the pseudo-code, $a = \vee M(F \restriction X_1)$ and $b = \wedge m(F \restriction X_2)$. We refer to these labels as the *extremal labels*. The label b is swapped downwards along a unique chain in the partial order $(X_1, \sqsubseteq_1)$ labeled by F_1 and a is swapped upwards along a unique chain in the partial order $(X_2, \sqsubseteq_2)$

labeled by F_2. The result of appending these two paths forms a chain in the product partial order $(X_1 \bigotimes X_2, \sqsubseteq_1 \bigotimes \sqsubseteq_2)$.We will show that each such swap sequence along such a unique chain is injective. It follows that the labeling-product function $\bigotimes$ is injective.

In order to show the result, we assume that we have two labelings $F_1, F_1^{'}$ of the partial order $(X_1, \sqsubseteq_1)$ and two labelings F_2, F_2' of the partial order $(X_2, \sqsubseteq_2)$ such that F_1 and F_2 are disjoint, F_1' and F_2' are disjoint and $F_1 \bigotimes F_2 = F_1' \bigotimes F_2'$. We show that $F_1 = F_1'$ and $F_2 = F_2'$.

We will display the labels on the chain determined by the swap sequence arising from the call to $F_1 \bigotimes F_2$, by:

$$[a_1, a_2, \ldots, a_m], [b_1, b_2, \ldots, b_k],$$

where (a, b) is the first pair which is swapped by the algorithm, $a_m = a, b_1 = b$, the sequence $[a_1, a_2, \ldots, a_m]$ consists of the labels in the labeled partial order $(X_1, \sqsubseteq_1 , F_1)$ which are respectively swapped with b and the sequence $[b_1, b_2, \ldots, b_k]$ consists of the labels in the labeled partial order $(X_2, \sqsubseteq_2, F_2)$ which are respectively swapped with a.

In the above, we allow the case where $m = 0$ and $k = 0$, i.e. no swap occurs.

Similarly,we display the labels on the chain determined by the swap sequence arising from the call to $F_1' \bigotimes F_2'$, by:

$$[a_1', a_2', \ldots, a_n'], [b_1^{'}, b_2^{'}, \ldots, b_l'],$$

where (a', b') is the first pair which is swapped by the algorithm, $a_n' = a', b_1' = b'$, the sequence $[a_1', a_2', \ldots, a_m']$ consists of the labels in the labeled partial order $(X_1, \sqsubseteq_1, F_1')$ which are respectively swapped with b' and the sequence $[b_1^{'}, b_2^{'}, \ldots, b_k^{'}]$ consists of the labels in the labeled partial order $(X_2, \sqsubseteq_2, F_2')$ which are respectively swapped with a'.

In the above, we again allow the case were $n = 0$ and $l = 0$, i.e. no swap occurs.

We remark that $Ra(F_1) = Ra(F_1') = \mathcal{L}_1$ and that $Ra(F_2) = Ra(F_2') = \mathcal{L}_2$. This implies that $a = a'$ and $b = b'$.

We show that $a = a'$. The case $b = b'$ is similar. The algorithm selects the maximal label a at depth 0 in the labeled partial order $(X_1, \sqsubseteq_1, F_1)$ and the maximal label a' in the labeled partial order $(X_1, \sqsubseteq_1, F_1')$. Since $Ra(F_1) = Ra(F_1') = \mathcal{L}_1$ and labelings are increasing, we know that the maximum label of $\mathcal{L}_1$ must occur as a label of a maximal element and thus $a = a' = maximum(\mathcal{L}_1)$.

We remark that this fact does not alter, even after the first two push operations in the algorithm have been run through a number of times. Inductively one can show that $Ra(F_1) = Ra(F_1')$ remains true. Indeed, in case $a < b$ no swaps will occur and the result holds trivially. Otherwise, after the first series of swaps has happened for the first two while loops, we obtain that in $Ra(F_1)$, the label a simply has been replaced by the label b and in F_1' the same has taken place. Hence we preserve the fact that the ranges of the respective labelings coincide, which suffices

to yield the desired property.

It follows by the fact that $a = a'$ and $b = b'$ at the start of each swap sequence, the number of non-trivial swap sequences induced by $F_1 \bigotimes F_2$ is identical to the number of non-trivial swap sequences induced by $F_1' \bigotimes F_2'$.

Hence we can focus on the *last* swap sequences induced by $F_1 \bigotimes F_2$ and $F_1' \bigotimes F_2'$ respectively and assume that both swap sequences, by the above, must start with a swap on the same pair of elements, a and b. Since the labelings of course have changed during the previous swap sequences, we denote the labelings at the start of the final swap sequences by G_1, G_2 and G_1', G_2' respectively.

Consider these final chains along which the labels are swapped, i.e. the chain

$$[G_1^{-1}(a_1), G_1^{-1}(a_2), \ldots, G_1^{-1}(a_m)], [G_2^{-1}(b_1), G_2^{-1}(b_2), \ldots, G_2^{-1}(b_k)]$$

and the chain

$$[(G_1')^{-1}(a_1'), (G_1')^{-1}(a_2'), \ldots, (G_1')^{-1}(a_n')], [(G_2')^{-1}(b_1'), (G_2')^{-1}(b_2'), \ldots, (G_2')^{-1}(b_l')].$$

To show injectivity for the final swap sequences, it suffices that these chains must be identical.

Indeed, assume that these paths are the same, say a path denoted by P. Since $F_1 \bigotimes F_2 = F_1' \bigotimes F_2'$ and the swap sequence on P does of course not affect labels of $X_1 - P$, the labelings G_1 and G_1' must coincide on the set $X_1 - P$. Moreover, since the net result of the Push-Down operation is to move the label of the maximal element of P to the element originally labeled with b in F_2 and to move every other label of an element of P to the element immediately above it on P, we obtain that G_1 must be identical to G_1'.

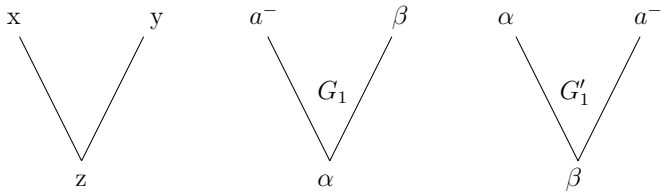
We claim that it is always the case that the swap sequences corresponding to b must be the same for G_1 and G_1' and hence, by the above, the final swap operations form an injective operation.

We recall that since $F_1 \bigotimes F_2 = F_1' \bigotimes F_2'$, we must have that at the end of both Push-Down operations the label b is a label of the same element in the partial order.

We assume by way of contradiction that the paths are not identical and hence diverge at one point. Because b must end up at the end of the final swap sequences in the same position, we know there is a first time, after the sequences diverge, that the label b ends up as a label of the same element z of X. Say that prior to these swaps we had: $H_1^{-1}(x) = b$ and $H_1'^{-1}(y) = b$ where $x \neq y$ and where H_1 and H_1' are the labelings obtained from G_1 and G_1' by carrying out the swaps on G_1 and G_1' up to the point prior to the first convergence of the paths.

We clarify the situation for both labelings H_1 and H_1' in the following figure. In H_1 the label b will be swapped with a label α while in H_1' the label b will be swapped with a label β.

Since after these swaps the labels of x and y will not be changed again, the labels as displayed in the figure below, are the only ones possible in order to guarantee that the final results of the Push-Down calls are identical.



We now obtain a contradiction since from labeling H_1 it is clear that $\alpha < \beta$ while from labeling H_1' we obtain that $\beta < \alpha$.

Hence we cannot have divergence of the path and the result follows.

Since the same argument holds for a, we obtain that both swap paths must be identical.

The proof can now be concluded by an inductive argument remarking that the same must hold for every pair of swap sequences, when run through in reverse order of their occurrence. Since on elements outside the swap paths, no labels are ever swapped, we obtain that $F_1 = F_1'$ and $F_2 = F_2'$.

Finally we need to verify that the label-product function is surjective. It suffices to verify that $|\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1)| \times |\mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2)| = |\mathcal{R}_{\mathcal{L}_1 \cup \mathcal{L}_2}(X_1 \bigotimes X_2, \sqsubseteq_1 \bigotimes \sqsubseteq_2)|$. We remark that $|\mathcal{R}_{\mathcal{L}_1 \cup \mathcal{L}_2}(X_1 \bigotimes X_2, \sqsubseteq_1 \bigotimes \sqsubseteq_2)| = |\mathcal{R}_{\mathcal{L}_1'}(X_1, \sqsubseteq_1)| \times |\mathcal{R}_{\mathcal{L}_2'}(X_2, \sqsubseteq_2)|$, where $\mathcal{L}_1'$ consists of the first $|X_1|$ elements in the sorted version of $\mathcal{L}$ while $\mathcal{L}_2'$ consists of the last $|X_2|$ elements in the sorted version of $\mathcal{L}$. This follows by the fact that the sets X_1 and X_2 are completely connected in the partial order $(X_1 \bigotimes X_2, \sqsubseteq_1 \bigotimes \sqsubseteq_2)$. Since we can identify labelings up to order-isomorphism, it is clear that $|\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1)| = |\mathcal{R}_{\mathcal{L}_1'}(X_1, \sqsubseteq_1)|$ and that $|\mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2)| = |\mathcal{R}_{\mathcal{L}_2'}(X_2, \sqsubseteq_2)|$. Hence the result follows.

□

We obtain the following immediate corollary.

Corollary 5.7 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ form a partition of the set of labels $\mathcal{L}$. Then:*

$$|\mathcal{R}_{\mathcal{L}}(X_1 \otimes X_2, \sqsubseteq_1 \otimes \sqsubseteq_2)| = |\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1)| \times |\mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2).|$$

Example 5.8 *In the example on pages 14 and 15, we illustrate that the creation of the random product of labelings is an injective process. We do not display all cases, but restrict our attention to the case of a fixed set of labelings which can be used on the first partial order ($\{1,2,3,4\}$) and a fixed set of labelings which can be used on the second partial order ($\{5,6,7\}$). It is easy to verify that the number of possible combinations of labelings for the given partial orders from the set of labels $\{1,2,3,4,5,6,7\}$ is $\binom{7}{4} \times 5 \times 2 = 350$, which prevents a complete illustration of all cases. The first five combinations of pairs of labelings are displayed in bold design at the top of the following page, followed by the computation steps, while the next five combinations are displayed again on the next page in bold design, followed by the computation steps.*

We define the binary random product below, which may be the first type of

product that comes to mind, followed by the unary random product which is the one that will be used in the applications.

5.3 The binary random product

Definition 5.9 Let $\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1)$ and $\mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2)$ be two disjoint random structures. We define *the binary random product*, $\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1) \bigotimes \mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2)$, by $\mathcal{R}_{\mathcal{L}_1 \cup \mathcal{L}_2}(X_1 \bigotimes X_2, \sqsubseteq_1 \bigotimes \sqsubseteq_2)$.

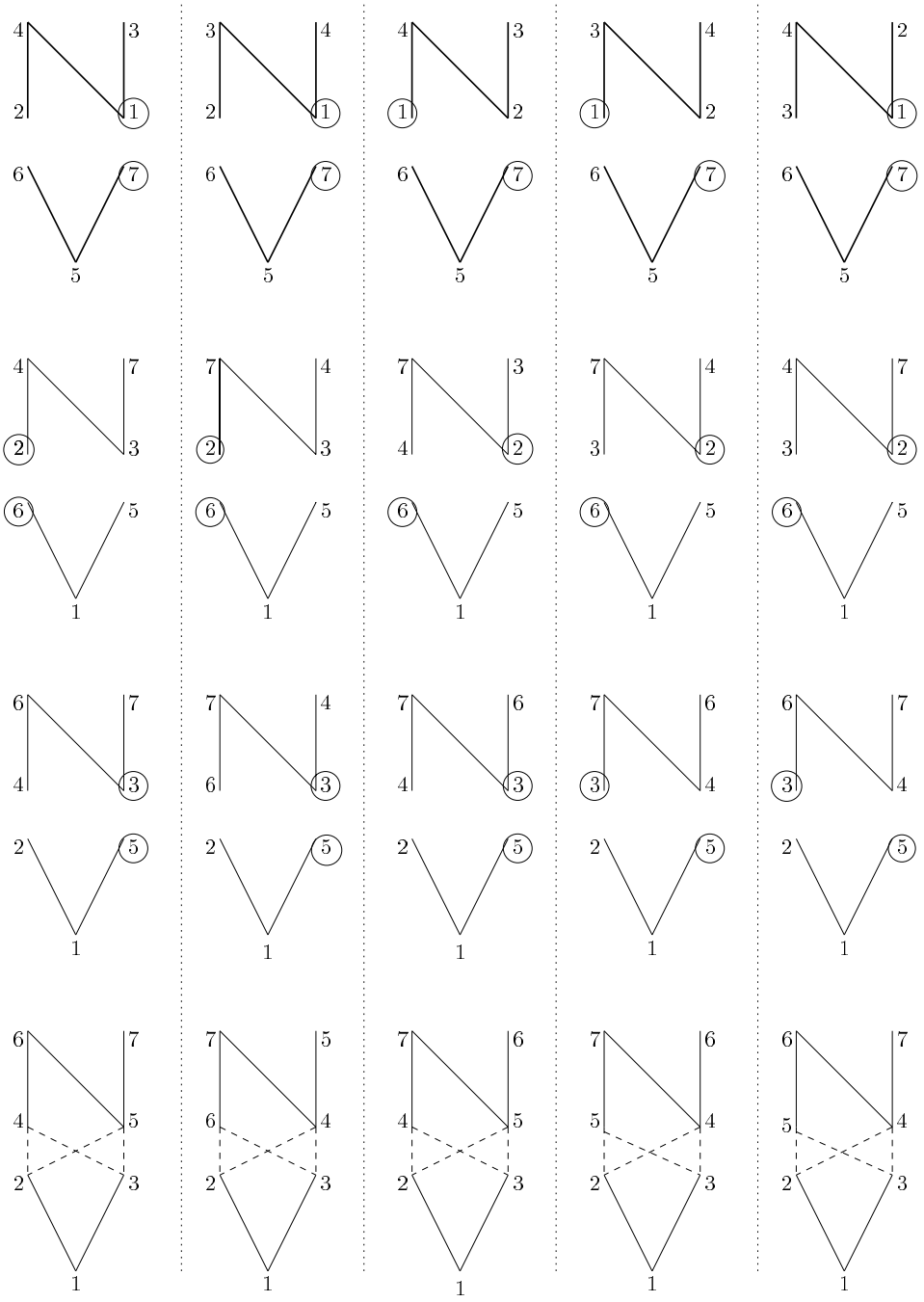
Lemma 5.10 *The binary random product is RS-preserving.*

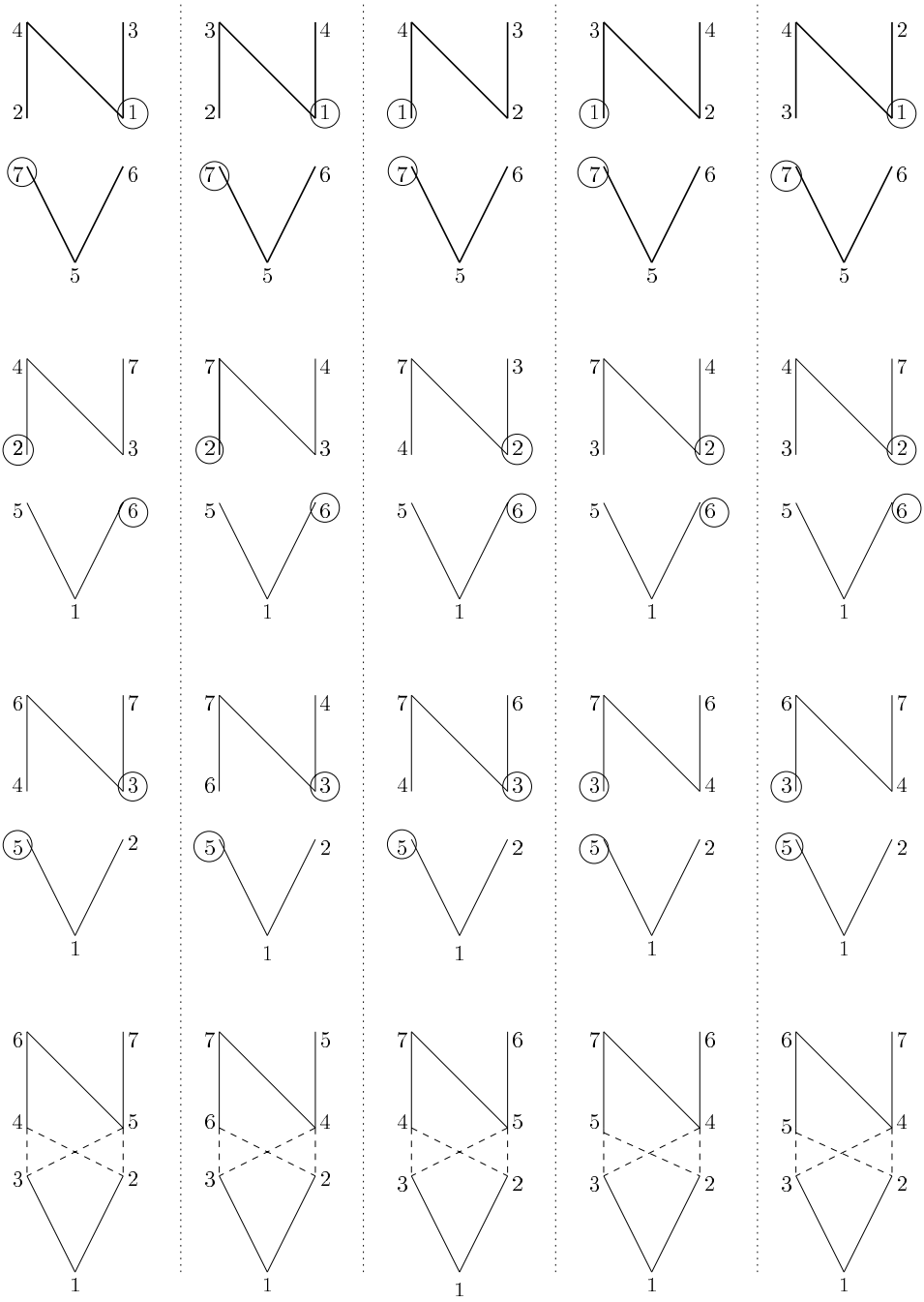
Proof. This follows directly from Theorem 5.6.

□

Remark 5.11 *The binary random product leads to complications regarding the determination of average-time. Indeed, the binary random product has an average time which is a function of the label sets $\mathcal{L}_1$ and $\mathcal{L}_2$. Consider the case where the set $\mathcal{L}_1$ happens to consist of labels which are smaller than the least label of $\mathcal{L}_2$. In this case the binary random product will require no push-downs nor push-ups. At the other extreme, consider label sets $\mathcal{L}'_1$ and $\mathcal{L}'_2$ for which the labels of $\mathcal{L}'_2$ are greater than the largest label of $\mathcal{L}'_1$. In that case clearly the binary random product $\mathcal{R}_{\mathcal{L}'_1}(X_1, \sqsubseteq_1) \bigotimes \mathcal{R}_{\mathcal{L}'_2}(X_2, \sqsubseteq_2)$ will require many push-down and push-up operations. Hence its average time will be strictly greater in general than the average time of $\mathcal{R}_{\mathcal{L}_1}(X_1, \sqsubseteq_1) \bigotimes \mathcal{R}_{\mathcal{L}_2}(X_2, \sqsubseteq_2)$. The dependency of the binary product operation on the label sets involved leads to complications regarding the determination of its average time. Hence the $\mathcal{MOQA}$ language does not contain the binary random product at this stage. Instead we include the unary random product as described below, which avoids the above problem and for which formulas have been derived expressing the average-case time. There is always the option to include the binary random product operation and let the analysis tool return a black-box type message for this particular operation, where the user needs to supply their own time analysis for this particular operation. Such an approach*[5]

[5] Continued on page 16





may be feasible for simple data structures or for the case where the tool is extended with a method to directly compute the average-case time for bounded data structure size through dynamic analysis of the binary random product.

5.4 The unary random product

We describe the unary random product which has been implemented in $\mathcal{MOQA}$-Java at CEOL. This type of product is useful in implementations which involve the joining of data structures, such as for instance for the merge operation in the Mergesort algorithm and the insertion operation in the Insertion sort algorithm.

Definition 5.12 Consider a random structure $\mathcal{R}(X, \sqsubseteq)$ and distinct components I_1 and I_2 of X. We define *the unary random product* of the partial order $(X, \sqsubseteq)$ with respect to components I_1, I_2 of X to be the partial order $(X, \sqsubseteq_{I_1 \bigotimes I_2})$ where $\sqsubseteq_{I_1 \bigotimes I_2}$ is the least partial order containing $\sqsubseteq \cup ((\sqsubseteq \restriction I_1) \bigotimes (\sqsubseteq \restriction I_2))$.

We define the unary random product to be the function:

$$\mu_{I_1 \bigotimes I_2}(X) \colon \mathcal{R}(X, \sqsubseteq) \to \mathcal{R}(X, \sqsubseteq_{I_1 \bigotimes I_2})$$

where $\forall F \in \mathcal{R}(X, \sqsubseteq).\ \mu_{I_1 \bigotimes I_2}(X, I)(F) \restriction (I_1 \bigotimes I_2) = (F \restriction I_1) \bigotimes (F \restriction I_2)$ and
$\mu(F) \restriction (X - (I_1 \cup I_2)) = F \restriction (X - (I_1 \cup I_2))$.

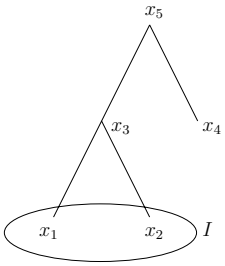
Theorem 5.13 *Consider a random structure $\mathcal{R}(X, \sqsubseteq)$ and distinct components I_1 and I_2 of X. The unary random product $\mu_{I_1 \bigotimes I_2}(X)$ is RS-preserving with multiplicity $\binom{|I_1|+|I_2|}{|I_1|}$.*

Proof. We sketch the proof. Let $\mathcal{L}$ be a set of labels for $I_1 \cup I_2$. From Corollary 5.7 we obtain that for any partition $(\mathcal{L}_1, \mathcal{L}_2)$ of $\mathcal{L}$: $|\mathcal{R}_{\mathcal{L}}(I_1 \bigotimes I_2, \sqsubseteq_1 \bigotimes \sqsubseteq_2)| = |\mathcal{R}_{\mathcal{L}_1}(I_1, \sqsubseteq_1)| \times |\mathcal{R}_{\mathcal{L}_2}(I_2, \sqsubseteq_2)|$. The result follows since there are $\binom{|I_1|+|I_2|}{|I_1|}$ such partitions. □

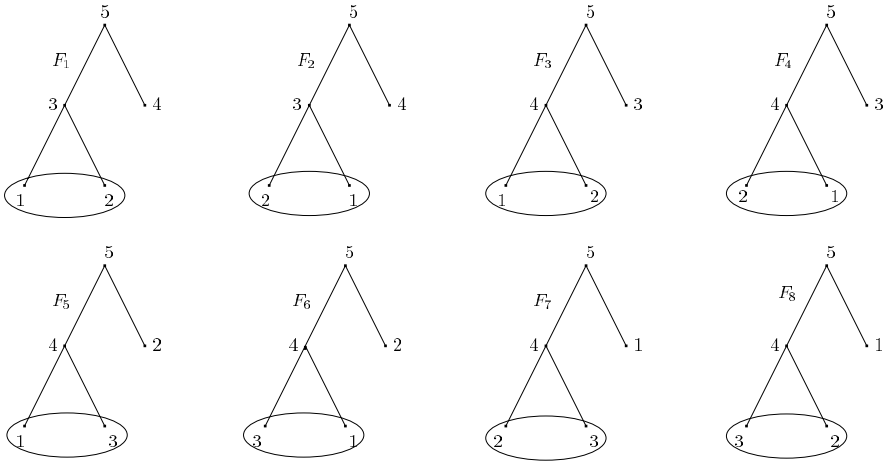
We remark that we can extend the unary random product to operate on random sub-structures of a given random structure as outlined in [2]. The details are technical and are omitted here. Suffices it to say that in case the unary random product is applied to a so-called *isolated* subset of a partial order, the operation is still guaranteed to be RS-preserving. This is illustrated in the example below. Again, we do not provide the formal definition of an isolated subset due to space restrictions. The increased generality of the application of the operations greatly increases its use in the $\mathcal{MOQA}$ language. The reader is referred to [2] for a comprehensive discussion.

We provide an example of the unary random product.

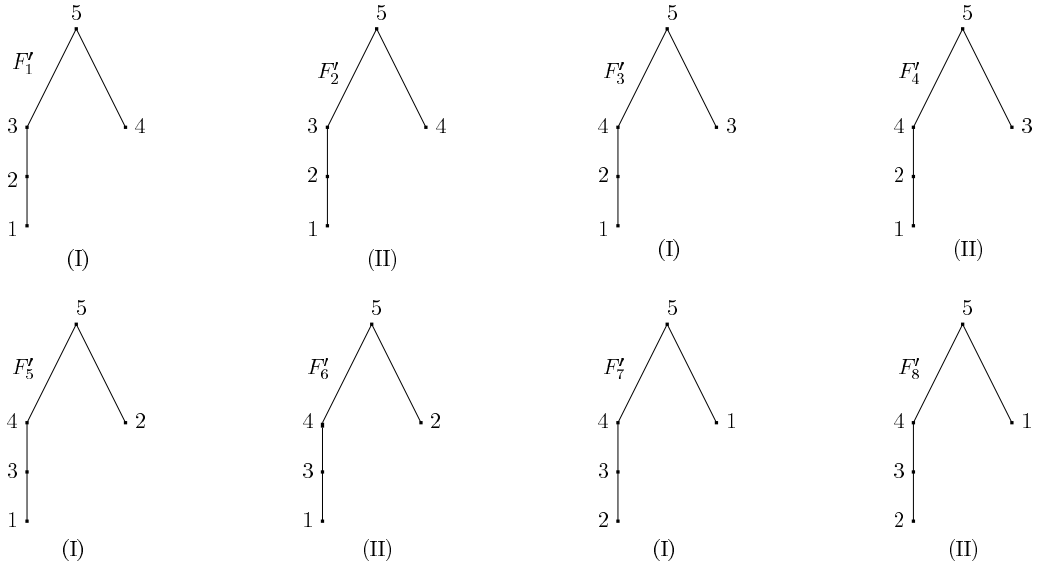
Example 5.14 *Consider the Hasse Diagram of the following tree:*



We display the eight labelings of the tree, where we selected the two leaves at the deepest level, i.e. x_1 and x_2, to form the atomic isolated subset I and labels for this set have been indicated as below.



We apply the unary random product to the isolated subset $I = \{x_1, x_2\}$ and we use the components $I_1 = \{x_1\}$ and $I_2 = \{x_2\}$. The result is displayed below. The multiplicity involved is $\binom{|I_1+I_2|}{|I_1|} = \binom{2}{1} = 2$. Indeed, we obtain two copies of a random structure, a first copy consisting of the labelings marked by (I), i.e. the labelings with odd indices, and a second copy consisting of the labels marked by (II), i.e. the labelings with even indices.



References

[1] B. A. Davey, H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, 1990.

[2] M. Schellekens, A Modular Calculus for the Average Cost of Data Structuring, Springer book to appear, 250 pages, May 2008 (to appear).

[3] M. Schellekens, Compositional Average-Case Analysis, preprint, under review, 2006.

[4] M. Boubekeur, D. Hickey, J. Mc Enery and M. Schellekens, A new Approach for Modular Average-Case Timing of Real-Time Java Programs, Accepted for publication in WSEAS Transactions on Computers, 2007.

[5] M. Boubekeur, D. Hickey, J. Mc Enery and M. Schellekens, Towards Modular Average-Case Timing in Real-Time Languages: An Application to Real-Time Java, Accepted for publication on the 6th WSEAS International Conference on APPLIED COMPUTER SCIENCE (ACS'06), Tenerife, December, 2006.

[6] S. Edelkamp. *Weak-Heapsort, ein schnelles sortierverfahren*. Diplomarbeit Universität Dortmund, 1996.

[7] D. Knuth, *The art of computer programming* vol.**3**, Addison-Wesley, 1973.

[8] M.Schellekens, D. Hickey and G. Bollella, ACETT, a Linearly-Compositional Programming Language for (semi-)automated Average-Case analysis, IEEE Real-Time Systems Symposium - Work In Progress Session, 2004.

[9] M. Li and P. Vitanyi. An introduction to Kolmogorov Complexity and its applications, Texts and Monographs in Computer Science. Springer Verlag, 1993.