Fundamenta Informaticae 173 (2020) 1–31
DOI 10.3233/FI-2020-1913
IOS Press

Higher-Order Rank Functions on Directed Graphs

Kenji Kashiwabara[*]
Department of General Systems Studies
University of Tokyo
3-8-1, Komaba, Meguro-ku, Tokyo, Japan
kashiwa@idea.c.u-tokyo.ac.jp

Ikumi Horie
Faculty of Economics
Dokkyo University
1-1, Gakuen-cho, Soka-shi, Saitama, Japan
horie@dokkyo.ac.jp

Kazunori Yamaguchi
Department of General Systems Studies
University of Tokyo
3-8-1, Komaba, Meguro-ku, Tokyo, Japan
yamaguch@graco.c.u-tokyo.ac.jp

Abstract. We introduce a new higher-order rank function with the capability to completely discriminate non-equivalent nodes. We review the partition lattice and rank functions and situate the existing rank functions and higher-order rank functions within the formalization. We propose a new refining operator and a new rank function that are better than the existing ones in some applications. We also show that the entire topology (graph) can be reconstructed from only our higher-order ranks making it possible to compare nodes in different graphs and to update the equivalence of nodes when an edge is added. Finally, we briefly describe the use of our higher-order rank function in analyzing web pages as a possible application in different domains.

[*]Address for correspondence: Department of General Systems Studies, the University of Tokyo, 3-8-1, Komaba, Meguro-ku, Tokyo, Japan.

Received June 2017; revised October 2019.

1. Introduction

A wide variety of information can be expressed in the form of a directed graph, for example, the web graph of the Web or the state transition graph of a state transition system [1]. A graph can be considered to be a way to represent sets in a non-well-founded set theory when its bisimilarity is guaranteed on the nodes. Here, we will use the Anti-Foundation Axiom [2], which we call *AFA*, as a non-well-founded set theory. Moreover, we will sometimes use set-theoretical notions for mathematical simplicity.

We often have to group nodes according to the bisimilarity in these applications. For example, we might want to identify structurally identical sites on the web [3, 4]. In a transition system, it is often required to identify equivalent states [5, 6]. For AFA, we have to identify the equivalence of nodes based on the Anti-Foundation Axiom to obtain a set represented by the nodes.

A leaf node and a non-leaf node can never be structurally grouped. Thus, whether it is a leaf node or not is an attribute showing that two nodes can never be grouped. A rank function is a function to yield such a value called a *rank value* for a node.

By aggregating the rank values of its children, we can construct richer rank functions. We call such a constructed function *a higher-order rank function* and the original rank function *a base rank function* in this paper. By repeating this child aggregation, we can make richer and richer higher-order rank functions. Higher-order rank functions have been used in algorithms, for example, to improve the speed of the bisimulation calculation, but their general properties have not been studied fully. For example, it is not clear whether we can determine the equivalence simply by the higher-order rank values of nodes or not. Higher-order rank values of only orders 0 and 1 (xrank$'$ in Section 4.1) have been used in the literature [5], but in these cases, the equivalence was determined by the refining algorithm. In the literature [7], a higher-order rank function (arank in Section 4.2) had the refining algorithm as part of it. In this paper, we propose a new scheme for constructing a higher-order rank function from a given base rank function and show that the higher-order rank function determines the equivalence completely if the base rank function satisfies a certain condition.

Our higher-order rank function can be used as its own. The higher-order rank function of order k reflects the structural difference up to depth k and is easy to interpret. In [3], the higher-order rank function was used to investigate the structural similarity of web graphs. Accordingly, we believe that there is a merit to study the higher-order rank functions and clarify their properties.

A rank function induces such an equivalence relation on nodes wherein two nodes are equivalent if and only if their rank values are equal. The equivalence relation induces a partition on the nodes. Thus, the rank function is related to a partition and the higher-order rank function is related to refining. In this paper, we review the properties of partitions and refining. It turns out that important properties of higher-order rank functions can be determined from the properties of the corresponding refining operators.

The contributions of this paper are as follows.

- We introduce a new higher-order rank function ($\mathrm{H}\,^k_r$ in Section 4.3) and show that the higher-order rank function for some base rank function r with sufficiently large k induces the coarsest stable partition (least partitioned without conflict.)

- As a partial answer to the following intuition, we show that the entire topology can be encoded into higher-order ranks if some conditions are met (Section 5.1).

  Our intuition is that, for a given non-well-founded set a, such an optimal notion should somehow encode the entire topology of the well-founded sets in the transitive closure of a. ([5], p. 239)

- We propose a new refining operator (**cpo** in Section 2.5) that is simpler than the previous operators.
- We show which of the rank functions defined in [5, 7, 8] can be used to calculate a coarsest stable partition (in Section 3). We introduce a new rank function (nrank in Section 3.1).
- We reformulate some rank functions in [5, 7] as higher-order rank functions (xrank' and arank in Sections 4).
- The use of our higher-order rank function in analyzing web pages is briefly described (Sections 4.1 and 4.2).
- Our higher-order rank function is uniquely determined by a graph. Thus, it can be used for comparing nodes in two different graphs or for calculating updates. These possibilities are briefly described in Sections 5.3 and 5.4.

This paper focuses on mathematical aspects. Sometimes we will compute the computational complexity, but the development of a faster algorithm is beyond the scope of this paper.

In this paper, we do not consider labels because sets with labels can be simulated by using a set without labels ([9], p. 78).

2. Stable partition

In this section, we study the properties of refining operators and partitions.

2.1. Lattice of partitions

Let V be a non-empty finite set throughout this paper.

A family $\mathcal{Q} \subseteq 2^V$ on V is said to be a *partition* on V if (1) $V = \bigcup \mathcal{Q}$, (2) $A, B \in \mathcal{Q}$ with $A \neq B$ implies $A \cap B = \emptyset$, and (3) $\emptyset \notin \mathcal{Q}$. In the following, $\mathcal{P}$, $\mathcal{Q}, \mathcal{R}$, ... represent partitions of V. An element of a partition is called a *block*.

A relation $\preceq$ between partitions $\mathcal{P}$ and $\mathcal{Q}$ on V is defined so that $\mathcal{P} \preceq \mathcal{Q}$ if and only if for any block $A \in \mathcal{P}$ there exists a block $B \in \mathcal{Q}$ with $A \subseteq B$. We say that $\mathcal{P}$ is a *refinement* of $\mathcal{Q}$.

The join operator $\vee$ is such an operator that $a \vee b$ is the minimal partition c such that $a \preceq c$ and $b \preceq c$, and the meet operator $\wedge$ is such an operator that $a \wedge b$ is the maximal partition c such that $c \preceq a$ and $c \preceq b$. Here, the maximal and minimal are with respect to $\preceq$.

Proposition 2.1. $\mathcal{P} \vee \mathcal{Q} = \min\{C \subseteq V \mid C$ is a union of some blocks in $\mathcal{P}$ and also is a union of some blocks in $\mathcal{Q}$ and $C \neq \emptyset\}$.

Proof:
Let $\mathcal{R} = \{C \subseteq V \mid C$ is the union of some blocks in $\mathcal{P}$, and C is the union of some blocks in $\mathcal{Q}\}$.

We first show that $\min \mathcal{R}$ is a partition. Since $V \in \mathcal{R}$, $\mathcal{R}$ covers V, that is $\bigcup \mathcal{R} = V$. If A and B with $A \subsetneq B$ belong to $\mathcal{R}$, $B - A$ belongs to $\mathcal{R}$ since $\mathcal{P}$ and $\mathcal{Q}$ are partitions. If A and B belong to $\mathcal{R}$, $A \cup B \in \mathcal{R}$. So if A and B belong to $\mathcal{R}$, $A \cap B \in \mathcal{R}$ or $A \cap B = \emptyset$ since $A \cap B = (A \cup B) - ((A \cup B) - A) - ((A \cup B) - B)$. So $\min \mathcal{R}$ is a partition because any $A, B \in \min \mathcal{R}$ satisfies $A = B$ or $A \cap B = \emptyset$.

Since $\mathcal{P} \preceq \min \mathcal{R}$ and $\mathcal{Q} \preceq \min \mathcal{R}$, we have $\mathcal{P} \vee \mathcal{Q} \preceq \min \mathcal{R}$.

For any $A \in \mathcal{P} \vee \mathcal{Q}$, we have $A \in \mathcal{R}$ since $\mathcal{P} \vee \mathcal{Q} \succeq \mathcal{P}$ and $\mathcal{P} \vee \mathcal{Q} \succeq \mathcal{Q}$. Moreover, $A \in \min \mathcal{R}$ because any element in $\mathcal{R}$ can be written as a union of some elements of $\min \mathcal{R}$. So we have $\mathcal{P} \vee \mathcal{Q} \succeq \min \mathcal{R}$. ☐

$\mathcal{P} \wedge \mathcal{Q} = \{A \cap B \mid A \in \mathcal{P}, B \in \mathcal{Q}, A \cap B \neq \emptyset\}$.

All the partitions form a lattice with $\wedge$ and $\vee$. This lattice is called a *partition lattice*. The minimum element of the partition lattice of V is $\{\{a\} \mid a \in V\}$ and the maximum element is $\{V\}$. Because V is assumed to be finite, the partition lattice on V is also finite.

2.2. Directed graph and stable partition

A directed graph G is a pair $G = (V, E)$ of a set V of nodes and a set E of edges such that $E \subseteq V \times V$. Here, a directed graph has no parallel edges in the same direction, but may have loops. Hereafter we fix a graph $G = (V, E)$ and omit it from the notation if there is no ambiguity.

For an edge $(a, b) \in E$, b is called a *child* of a and a is called a *parent* of b. A node with no child is called *a leaf node*[1]. The in-degree of a leaf node is not limited to 1 and a node with in-degree 0 is called a *root* in this paper. The set of all the leaf nodes is denoted as $\Phi = \{b \in V \mid \forall a \in V, (b, a) \notin E\}$. If there are multiple graphs and it is necessary to specify which graph the leaf nodes belong to, we denote the leaf nodes of a graph G as $\Phi(G)$.

Let $E^{-1} : 2^V \to 2^V$ be a map from a set $A \subseteq V$ to its parents defined as $E^{-1}(A) = \{b \in V \mid (b, a) \in E, a \in A\}$. The image of V by f is denoted as $f[V] = \{f(a) \mid a \in V\}$.

In the set theory based on the Anti-Foundation Axiom [2, 10], a cyclic graph can be considered to represent a set. In this sense, we denote an edge $(a, b) \in E$ as $b \in_E a$. We omit E of $\in_E$ if E is obvious. Note that the same set may be represented by different nodes. For a graph, this equivalence notion of nodes is defined as bisimulation.

Definition 2.2. [10] For $G = (V, E)$, a binary relation $\asymp$ on V is a bisimulation over G if and only if, for any nodes $a, b, c, d \in V$,

1. $(a \asymp b$ and $(a, c) \in E) \Rightarrow \exists d(c \asymp d$ and $(b, d) \in E)$,
2. $(a \asymp b$ and $(b, d) \in E) \Rightarrow \exists c(c \asymp d$ and $(a, c) \in E)$.

[1]In some literature, a leaf node is called a sink node.

If there is a bisimulation $\asymp$ such that $a \asymp b$, we say that nodes a and b are *bisimilar* and denote it as $a \equiv_b b$. If every node in G_1 is bisimilar to some node in G_2 and vice versa, we say G_1 is *bisimilar* to G_2.

$\equiv_b$ can be determined using the operator **stab** defined below.

Before the defining **stab** though, we introduce the operator **div**.

Definition 2.3. The operator **div** on a partition $\mathcal{P}$ on V is defined as

$$\mathbf{div}(\mathcal{P}) = (\bigwedge\{\{E^{-1}(A), (E^{-1}(A))^c\} - \{\emptyset\} \mid \ A \in \mathcal{P}\}) \wedge \mathcal{P}.$$

Here, $\bigwedge \mathcal{P}$ is the meet of all the elements in $\mathcal{P}$ and $A^c = V - A$.

Lemma 2.4.

$$\mathcal{P} \succeq \mathbf{div}(\mathcal{P}),$$
$$\mathcal{P} \succeq \mathcal{Q} \Longrightarrow \mathbf{div}(\mathcal{P}) \succeq \mathbf{div}(\mathcal{Q}).$$

Proof:
The first inequality holds by definition. Let's show the second one. Let $A \in \mathcal{P}$. Then there exist $B_i \in \mathcal{Q}$ such that $A = \bigcup_i B_i$. Accordingly, we have $E^{-1}(A) = \bigcup_i E^{-1}(B_i)$. Thus,

$$(\{E^{-1}(A), (E^{-1}(A))^c\} - \{\emptyset\}) \succeq \bigwedge_i (\{E^{-1}(B_i), (E^{-1}(B_i))^c\} - \{\emptyset\}).$$ □

For a directed graph and $A \subseteq V$, a partition $\mathcal{P}$ on V is said to be *stable* with respect to A if any $B \in \mathcal{P}$ satisfies either $B \subseteq E^{-1}(A)$ or $E^{-1}(A) \cap B = \emptyset$.

Lemma 2.5. For $A \subseteq V$, a partition $\mathcal{P}$ is stable with respect to A if and only if $E^{-1}(A)$ is the union of some blocks in $\mathcal{P}$.

Proof:
We assume that $\mathcal{P}$ is stable with respect to A. Now let us assume, on the contrary, that there exists $x \in E^{-1}(A) - \bigcup\{B \subseteq E^{-1}(A) \mid B \in \mathcal{P}\}$. Then there exists a block $B' \in \mathcal{P}$ containing x. Since $\mathcal{P}$ is stable with respect to A, the non-emptiness of $E^{-1}(A) \cap B'$ implies $B' \subseteq E^{-1}(A)$, a contradiction to $x \notin \bigcup\{B \subseteq E^{-1}(A) \mid B \in \mathcal{P}\}$, and we have $\bigcup\{B \subseteq E^{-1}(A) \mid B \in \mathcal{P}\} = E^{-1}(A)$.

Conversely, when $E^{-1}(A)$ is the union of some blocks in $\mathcal{P}$, any $B \in \mathcal{P}$ satisfies either $B \subseteq E^{-1}(A)$ or $E^{-1}(A) \cap B = \emptyset$. □

$\mathcal{P}$ is said to be *stable* if $\mathcal{P}$ is stable with respect to any block in $\mathcal{P}$.

Lemma 2.6. $\mathcal{P} = \mathbf{div}(\mathcal{P})$ if and only if $\mathcal{P}$ is stable.

Proof:
Assume that $\mathcal{P} = \mathbf{div}(\mathcal{P})$. Then we have $\{E^{-1}(A), (E^{-1}(A))^c\} - \{\emptyset\} \succeq \mathcal{P}$ for any $A \in \mathcal{P}$. This means that $\mathcal{P}$ is stable by Lemma 2.5.

Conversely, assume that $\mathcal{P}$ is stable. Then, by Lemma 2.5, for $A \in \mathcal{P}$, we have

$$\{E^{-1}(A), (E^{-1}(A))^c\} - \{\emptyset\} \succeq \mathcal{P}.$$

Thus, we have $\mathcal{P} = \mathbf{div}(\mathcal{P})$. □

Now let us estimate the complexity of the calculation of the operator **div**. In [11], it was shown that the computational complexity of the refinement by block B is $O(|B| + \sum_{y \in B} |E^{-1}(\{y\})|)$. To calculate $\mathbf{div}(\mathcal{P})$, we have to scan all the blocks $B \in \mathcal{P}$ resulting in a computational complexity of $O(|E| + |V|)$.

2.3. Stabilization operator

By applying the operator **div** iteratively starting from a partition $\mathcal{P}$, the partition gets finer. Eventually it reaches a stable partition since $|V|$ is finite. We denote such a partition by $\mathbf{stab}(\mathcal{P})$ and call the operator **stab** the *stabilization operator*.

Formally, we can define the operator **stab** as follows. Define $\mathbf{div}^{k+1}(\mathcal{P}) = \mathbf{div}(\mathbf{div}^k(\mathcal{P}))$ for $k \in \mathbf{N}$ and $\mathbf{div}^0(\mathcal{P}) = \mathcal{P}$. $\mathbf{div}^{k+1}(\mathcal{P}) \preceq \mathbf{div}^k(\mathcal{P})$ follows from Lemma 2.4. So this sequence converges. We define $\mathbf{stab}(\mathcal{P})$ to be $\mathbf{div}^l(\mathcal{P})$ for the smallest $l \in \mathbf{N}$ such that $\mathbf{div}^{l+1}(\mathcal{P}) = \mathbf{div}^l(\mathcal{P})$. Obviously $\mathbf{div}(\mathbf{stab}(\mathcal{P})) = \mathbf{stab}(\mathcal{P})$ holds.

stab has monotonicity and idempotency, as shown in the next lemma.

Lemma 2.7.

$$\mathcal{P} \succeq \mathbf{stab}(\mathcal{P}),$$
$$\mathcal{P} \succeq \mathcal{Q} \Longrightarrow \mathbf{stab}(\mathcal{P}) \succeq \mathbf{stab}(\mathcal{Q}),$$
$$\mathbf{stab}(\mathcal{P}) = \mathbf{stab}(\mathbf{stab}(\mathcal{P})).$$

Lemma 2.8. $\mathcal{P}$ is stable if and only if $\mathbf{stab}(\mathcal{P}) = \mathcal{P}$.

Proof:
Assume that $\mathcal{P}$ is stable. By Lemma 2.6, $\mathbf{div}(\mathcal{P}) = \mathcal{P}$. So we have $\mathbf{stab}(\mathcal{P}) = \mathcal{P}$.

Conversely, assume that $\mathbf{stab}(\mathcal{P}) = \mathcal{P}$. From the definition of **stab**, we have $\mathbf{div}(\mathbf{stab}(\mathcal{P})) = \mathbf{stab}(\mathcal{P})$. So, from the assumption, we have $\mathbf{div}(\mathcal{P}) = \mathcal{P}$. By Lemma 2.6, $\mathcal{P}$ is stable. □

Lemma 2.9. If $\mathcal{P}$ and $\mathcal{Q}$ are stable, $\mathcal{P} \vee \mathcal{Q}$ is also stable.

Proof:
Since $\mathcal{P}$ and $\mathcal{Q}$ are stable, we have $\mathbf{stab}(\mathcal{P}) = \mathcal{P}$ and $\mathbf{stab}(\mathcal{Q}) = \mathcal{Q}$.

$\mathcal{P} \vee \mathcal{Q} \succeq \mathbf{stab}(\mathcal{P} \vee \mathcal{Q})$ follows from the first statement in Lemma 2.7.

$\mathcal{P} \vee \mathcal{Q} \succeq \mathcal{P}$ implies $\mathbf{stab}(\mathcal{P} \vee \mathcal{Q}) \succeq \mathbf{stab}(\mathcal{P}) = \mathcal{P}$. $\mathbf{stab}(\mathcal{P} \vee \mathcal{Q}) \succeq \mathcal{P} \vee \mathcal{Q}$ follows from $\mathbf{stab}(\mathcal{P} \vee \mathcal{Q}) \succeq \mathcal{P}$ and $\mathbf{stab}(\mathcal{P} \vee \mathcal{Q}) \succeq \mathcal{Q}$.

So by the anti-symmetric law, we have $\mathbf{stab}(\mathcal{P} \vee \mathcal{Q}) = \mathcal{P} \vee \mathcal{Q}$. □

Even if $\mathcal{P}$ and $\mathcal{Q}$ are stable, $\mathcal{P} \wedge \mathcal{Q}$ is not always stable.

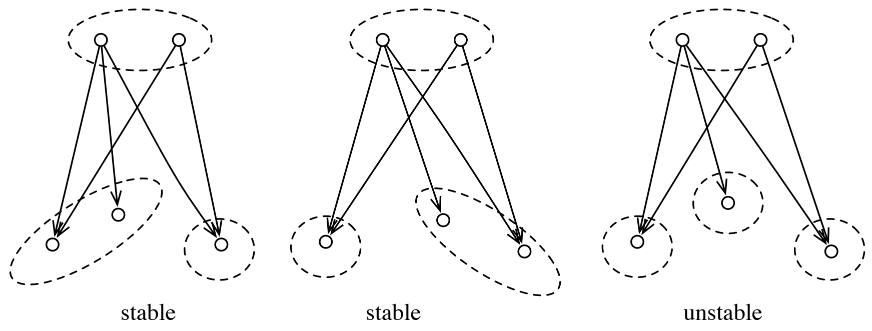

Figure 1. The meet of stable partitions is not always stable.

Example 2.10. Figure 1 shows three partitions on the same directed graph on five nodes. The left and middle partitions are stable partitions. But the right partition, which is the meet of the left and middle partitions, is unstable.

All the stable partitions form a lattice with $\vee$ and $\wedge^*$ where $\mathcal{P} \wedge^* \mathcal{Q} = \textbf{stab}(\mathcal{P} \wedge \mathcal{Q})$.

Lemma 2.11. $\textbf{stab}(\mathcal{P}) = \bigvee\{\mathcal{Q} \mid \mathcal{P} \succeq \mathcal{Q}, \mathcal{Q} = \textbf{div}(\mathcal{Q})\}$, where $\bigvee \mathcal{P}$ is the join of all the elements in $\mathcal{P}$.

Proof:
Let $\mathcal{R} = \bigvee\{\mathcal{Q} \mid \mathcal{P} \succeq \mathcal{Q}, \mathcal{Q} = \textbf{div}(\mathcal{Q})\}$. Then $\mathcal{R} \succeq \textbf{stab}(\mathcal{P})$ since $\textbf{div}(\textbf{stab}(\mathcal{P})) = \textbf{stab}(\mathcal{P})$.

On the other hand, we have $\mathcal{P} \succeq \mathcal{R}$ by definition and $\mathcal{R}$ is a stable partition by Lemma 2.6 and Lemma 2.9. Thus, $\textbf{stab}(\mathcal{P}) \succeq \textbf{stab}(\mathcal{R}) = \mathcal{R}$ by the monotonicity in Lemma 2.7. Therefore, from the anti-symmetric law, we have $\textbf{stab}(\mathcal{P}) = \mathcal{R}$. □

Since V is finite, the lattice of the stable partitions has the maximum element. We call the maximum element in all the stable partitions the *coarsest stable partition*.

Theorem 2.12. $\textbf{stab}(\{V\})$ is the coarsest stable partition.

Proof:
Let $\mathcal{P}$ be a stable partition. Then $\{V\} \succeq \mathcal{P}$. So $\textbf{stab}(\{V\}) \succeq \textbf{stab}(\mathcal{P})$ by the second statement in Lemma 2.7. Since $\textbf{stab}(\mathcal{P}) = \mathcal{P}$ by Lemma 2.8, we have $\textbf{stab}(\{V\}) \succeq \mathcal{P}$. Therefore, $\textbf{stab}(\{V\})$ is the coarsest stable partition. □

2.4. Relation of coarsest stable partition and AFA

In AFA, the equality of sets is determined by the bisimulation defined by the directed graph of $\in$. The coarsest stable partition of the relation coincides with the quotient set of V determined by the bisimulation [12]. So, the set corresponding to node x equals the set corresponding to node y if and only if x and y belong to the same block in the coarsest stable partition. Formally, $\equiv_b$ on $G = (V, E)$ is the same as $\sim_{\textbf{stab}(\{V\})}$. Here, $\sim_{\mathcal{R}}$ is the equivalence relation induced by partition $\mathcal{R}$.

2.5. Child partition operator

Here, we introduce a refining operator **cpo** that is simpler than **div**. **cpo** corresponds to some higher-order rank function in Section 4.3 and is useful for analyzing the properties of the higher-order rank function.

Definition 2.13. **cpo** is defined as
$\mathbf{cpo}(\mathcal{P}) = \bigwedge\{\{E^{-1}(A), (E^{-1}(A))^c\} - \{\emptyset\} \mid A \in \mathcal{P}\}$.

$\mathbf{cpo}(\mathcal{P}) \succeq \mathbf{div}(\mathcal{P})$ holds since $\mathbf{div}(\mathcal{P}) = \mathbf{cpo}(\mathcal{P}) \wedge \mathcal{P}$ by definition. Note that $\mathcal{P} \succeq \mathbf{cpo}(\mathcal{P})$ may not hold.

For a directed graph and a partition $\mathcal{R}$, define $\mathbf{cpo}^k(\mathcal{R})$ as $\mathbf{cpo}^0(\mathcal{R}) = \mathcal{R}$ and $\mathbf{cpo}^{k+1}(\mathcal{R}) = \mathbf{cpo}(\mathbf{cpo}^k(\mathcal{R}))$.

Example 2.14. Consider the directed graph in Figure 2 with nodes $\{a, b, c, d\}$. Let the initial partition $\mathcal{R}$ be $\{\{a, b, c\}, \{d\}\}$ as in the left figure. The middle figure shows $\mathbf{cpo}(\mathcal{R}) = \{\{a\}, \{b, c, d\}\}$ which is not finer than $\mathcal{R}$. The right figure shows $\mathbf{cpo}^2(\mathcal{R}) = \{\{a\}, \{b, c\}, \{d\}\}$. This is the coarsest stable partition.

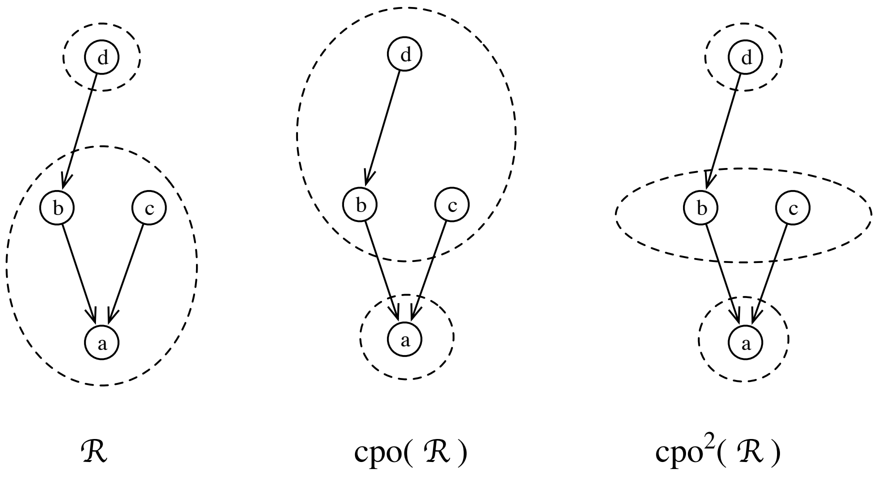

Figure 2. Example application of child partition operators: $\mathcal{R}$, $\mathbf{cpo}(\mathcal{R})$, and $\mathbf{cpo}^2(\mathcal{R})$.

Thus, **cpo** is not monotonic in general. However, it is monotonic under certain conditions as will be shown later in Theorem 2.20.

Lemma 2.15. If $\mathcal{P}_1 \succeq \mathcal{P}_2$, $\mathbf{cpo}(\mathcal{P}_1) \succeq \mathbf{cpo}(\mathcal{P}_2)$ holds.

Proof:
Since $|V|$ is finite, the length of sequence $\mathcal{P}_1 \succeq \cdots \succeq \mathcal{P}_2$ is also finite. Therefore, we have only to show the case that $\mathcal{P}_2$ is covered by $\mathcal{P}_1$. In this case, every block in $\mathcal{P}_1$ belongs to $\mathcal{P}_2$ except

one block, and $|\mathcal{P}_2| = |\mathcal{P}_1| + 1$. Let $A_1, A_2 \in \mathcal{P}_2$ and $A_1 \cup A_2 \in \mathcal{P}_1$. To prove this case of the lemma, we have only to show that $(\{E^{-1}(A_1), (E^{-1}(A_1))^c\} - \{\emptyset\}) \wedge (\{E^{-1}(A_2), (E^{-1}(A_2))^c\} - \{\emptyset\}) \preceq \{E^{-1}(A_1 \cup A_2), (E^{-1}(A_1 \cup A_2))^c\} - \{\emptyset\}$ for disjoint A_1 and A_2. This follows from the fact that $(\{E^{-1}(A_1), (E^{-1}(A_1))^c\} - \{\emptyset\}) \wedge (\{E^{-1}(A_2), (E^{-1}(A_2))^c\} - \{\emptyset\}) = \{E^{-1}(A_1) \cap E^{-1}(A_2), E^{-1}(A_1) \cap (E^{-1}(A_2))^c, (E^{-1}(A_1))^c \cap E^{-1}(A_2), (E^{-1}(A_1))^c \cap (E^{-1}(A_2))^c\} - \{\emptyset\}$ and $\{E^{-1}(A_1 \cup A_2), (E^{-1}(A_1 \cup A_2))^c\} = \{E^{-1}(A_1) \cup E^{-1}(A_2), (E^{-1}(A_1))^c \cap (E^{-1}(A_2))^c\}$. □

Lemma 2.16. If $\mathcal{P} \succeq \mathbf{cpo}(\mathcal{P})$, $\mathbf{cpo}(\mathcal{P}) = \mathbf{div}(\mathcal{P})$.

Proof:
$\mathbf{div}(\mathcal{P}) = \mathbf{cpo}(\mathcal{P}) \wedge \mathcal{P} = \mathbf{cpo}(\mathcal{P})$ when $\mathcal{P} \succeq \mathbf{cpo}(\mathcal{P})$. □

Lemma 2.17. If $\mathcal{R} \succeq \mathbf{cpo}(\mathcal{R})$, $\mathbf{cpo}^k(\mathcal{R}) \succeq \mathbf{cpo}^{k+1}(\mathcal{R})$ for $k \in \mathbf{N}$. Moreover, $\mathbf{div}^k(\mathcal{R}) = \mathbf{cpo}^k(\mathcal{R})$ holds for $k \in \mathbf{N}$ when $\mathcal{R} \succeq \mathbf{cpo}(\mathcal{R})$.

Proof:
We prove the former statement by induction on k. The base case follows from the assumption. The inductive step follows from the fact that $\mathbf{cpo}^k(\mathcal{R}) \succeq \mathbf{cpo}^{k+1}(\mathcal{R})$ implies $\mathbf{cpo}^{k+1}(\mathcal{R}) \succeq \mathbf{cpo}^{k+2}(\mathcal{R})$ by Lemma 2.15.

Next, we prove the latter statement by induction on k. By Lemma 2.16,

$$\mathbf{div}(\mathbf{cpo}^k(\mathcal{R})) = \mathbf{cpo}(\mathbf{cpo}^k(\mathcal{R})) = \mathbf{cpo}^{k+1}(\mathcal{R}).$$

By the inductive hypothesis,

$$\mathbf{div}(\mathbf{cpo}^k(\mathcal{R})) = \mathbf{div}(\mathbf{div}^k(\mathcal{R})) = \mathbf{div}^{k+1}(\mathcal{R}).$$

Now we have $\mathbf{div}^{k+1}(\mathcal{R}) = \mathbf{cpo}^{k+1}(\mathcal{R})$. □

Corollary 2.18. $\mathbf{div}^k(\{V\}) = \mathbf{cpo}^k(\{V\})$ holds for any $k \in \mathbf{N}$.

Proof:
The assumption of Lemma 2.17 is satisfied since $\{V\} \succeq \{\Phi, \Phi^c\} = \mathbf{cpo}(\{V\})$. □

Lemma 2.19. $\mathbf{cpo}(\mathbf{stab}(\{V\})) = \mathbf{stab}(\{V\})$.

Proof:
For a sufficiently large k, $\mathbf{stab}(\{V\}) = \mathbf{div}^k(\{V\})$. $\mathbf{cpo}(\mathbf{stab}(\{V\}) = \mathbf{cpo}(\mathbf{div}^k(\{V\}))$. From Corollary 2.18, $\mathbf{cpo}(\mathbf{div}^k(\{V\})) = \mathbf{cpo}^{k+1}(\{V\}) = \mathbf{div}^{k+1}(\{V\}) = \mathbf{stab}(\{V\})$. □

The following theorem is one of the main theorems in this paper.

Theorem 2.20. If $\mathcal{R} \succeq \mathbf{stab}(\{V\})$, for a sufficiently large k, $\mathbf{cpo}^k(\mathcal{R}) = \mathbf{stab}(\{V\})$. We can use $|V| - 1$ as a sufficiently large k.

Proof:
We have $\{V\} \succeq \mathcal{R} \succeq \mathbf{stab}(\{V\})$. By Lemma 2.17, for any k, we have $\mathbf{cpo}^k(\{V\}) \succeq \mathbf{cpo}^k(\mathcal{R}) \succeq \mathbf{cpo}^k(\mathbf{stab}(\{V\}))$. $\mathbf{cpo}^k(\mathbf{stab}(\{V\})) = \mathbf{stab}(\{V\})$ by Lemma 2.19. By Corollary 2.18, $\mathbf{div}^k(\{V\}) = \mathbf{cpo}^k(\{V\}) \succeq \mathbf{stab}(\{V\})$. For a sufficiently large k, $\mathbf{div}^k(\{V\}) = \mathbf{stab}(\{V\})$. So, for a sufficiently large k, $\mathbf{cpo}^k(\{V\}) = \mathbf{stab}(\{V\})$.

If $\mathbf{cpo}^k(\{V\}) = \mathbf{cpo}^{k+1}(\{V\})$, we have $\mathbf{cpo}^{k+1}(\{V\}) = \mathbf{cpo}^{k+2}(\{V\})$. Thus, $\mathbf{cpo}^k(\{V\}) = \mathbf{cpo}^{k+1}(\{V\})$ for $k \geq |V| - 1$. So, we can use $|V| - 1$ as a sufficiently large k. □

3. Rank function

We call a function from the nodes V to some domain D *a rank function* and call the value returned by the function *the rank value*. We assume that rank functions on isomorphic directed graphs are identical. The rank function $r : V \to D$ induces an equivalence relation $\sim_r$ such that $a \sim_r b \Leftrightarrow r(a) = r(b)$.

First, we define the most fundamental rank function **0**.

Definition 3.1. The rank function **0** is a rank function such that $\mathbf{0}(a) = 0$ for $a \in V$.

3.1. nrank

The length of a shortest path to some leaf node can be used as a rank, but in a cyclic case, there may exist a node which has no path to any leaf node. In such a case, we use ∞ as a rank value. This rank function nrank : $V \to \mathbf{N} \cup \{\infty\}$ is defined as follows.

$$\mathrm{nrank}(a) = \begin{cases} \text{the length of the shortest path from } a & \text{if there exists a path} \\ \quad \text{to some leaf node,} & \text{to some leaf node,} \\ \infty, & \text{otherwise.} \end{cases}$$

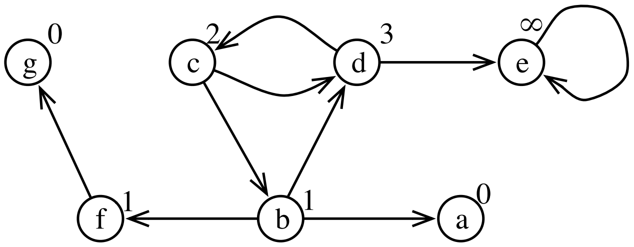

Figure 3. nrank for sample graph

In other words, nrank gives the minimum distance from the leaves.

Figure 3 shows an example of nrank.

The next proposition follows from the definition.

Proposition 3.2. For an integer $r \geq 0$,

$$\text{nrank}^{-1}(r) = E^{-1^{(r)}}(\Phi) - \bigcup_{0 \leq i \leq r-1} E^{-1^{(i)}}(\Phi)$$

where $E^{-1^{(r)}}$ denotes the composition of E^{-1} by r times.

Proposition 3.3. $V/\!\sim_{\text{nrank}} \succeq \mathbf{cpo}(V/\!\sim_{\text{nrank}})$.

Proof:
N_k denotes the block consisting of the nodes of rank k determined by the nrank function. Let $A \in \mathbf{cpo}(V/\sim_{nrank})$. $A = (\bigcap_{k \in I} E^{-1}(N_k)) \cap (\bigcap_{k \notin I} E^{-1}(N_k^c))$ for some $I \subseteq \mathbf{N} \cup \{\infty\}$. Let k_0 be the minimum number in I. Then $A \subseteq N_{k_0+1}$. □

Proposition 3.4. $V/\!\sim_{\text{nrank}} \succeq \mathbf{stab}(\{V\})$.

Proof:
Suppose, on the contrary, that the partition of the rank function is not coarser than or equal to the coarsest stable partition. Then there exists a block in the coarsest stable partition that has nodes of different ranks. Let A be a block which has a node of the least rank among such blocks. Let a be a node of the minimum rank in A. Let b be a node which does not have the minimum rank in A. Since the set of leaf nodes forms a block of the coarsest stable partition, we can assume that the rank of a is greater than 0. Then there exists an edge from node a to a node of rank $(\text{nrank}(a)-1)$. But there exists no edge from b to some node of rank $(\text{nrank}(a) - 1)$ because b has greater rank than a. Because of the choice of node a, any block in the coarsest stable partition that contains a node of rank $(\text{nrank}(a) - 1)$ cannot intersect with two blocks in $V/\!\sim_{\text{nrank}}$. So a and b belong to different blocks in the coarsest stable partition, a contradiction. □

$V/\!\sim_{\text{nrank}}$ may not be the coarsest stable partition but any two elements in the same block of the coarsest stable partition have the same rank by this lemma.

Corollary 3.5. $\mathbf{div}^k(V/\!\sim_{\text{nrank}}) = \mathbf{stab}(\{V\})$ for a sufficiently large k.

Proof:
The statement follows from Lemma 2.16, Propositions 3.3 and 3.4, and Theorem 2.20. □

The complexity of computing nrank is $O(|V| + |E|)$.

3.2. xrank

The literature [5] introduced a rank function, which we call xrank in this paper. xrank is defined as follows.

For a directed graph $G = (V, E)$, let $G^{scc} = (V^{scc}, E^{scc})$ be the graph defined as follows.

$V^{scc} = \{c \mid c \text{ is a strongly connected component in } G\}$,

$E^{scc} = \{(c_1, c_2) \mid c_1, c_2 \in V^{scc}, c_1 \neq c_2, (\exists a_1 \in c_1)(\exists a_2 \in c_2)((a_1, a_2) \in E)\}$.

G^{scc} is acyclic. For a node $a \in V$, we denote the strongly connected component to which a belongs as $c(a) \in V^{scc}$. For $a \in V$, let $G(a)$ be the graph restricted to the nodes reachable from $a \in V$ of G. $WF(G) = \{a \in V \mid G(a) \text{ is acyclic}\}$.

Definition 3.6. xrank(a) is defined as:

$$\text{xrank}(a) = \begin{cases} 0, & \text{if } a \in \Phi(G), \\ -\infty, & \text{if } c(a) \in \Phi(G^{scc}), \\ & a \notin \Phi(G), \\ \max(\{1 + \text{xrank}(b) \mid (c(a), c(b)) \in E^{scc}, b \in WF(G)\} & \\ \cup\{\text{xrank}(b) \mid (c(a), c(b)) \in E^{scc}, b \notin WF(G)\}), & \text{otherwise.} \end{cases}$$

$V/\sim_{\text{xrank}} \succeq \textbf{stab}(\{V\})$ was proved in [5].

The complexity of calculating strongly connected components is $O(|E|+|V|)$; xrank has the same complexity.

3.3. xrank*

For the equivalence of states in some transition systems, simulation, which is a slightly weaker condition than bisimulation, is used.

Definition 3.7. Let $G = (V, E)$. A binary relation $\leq$ is called a simulation over G when ($a \leq b$ and $(a, c) \in E) \Rightarrow \exists d(c \leq d$ and $(b, d) \in E)$.

a and b are sim-equivalent ($a \equiv_s b$) if there exist two simulations $\leq_1$ and $\leq_2$ such that $a \leq_1 b$ and $b \leq_2 a$.

For deciding sim-equivalence, another type of rank (called rank* in [8]) is defined on xrank (rank). We call this rank xrank*.

Definition 3.8. xrank$^*(a)$ is defined as:

$$\text{xrank}^*(a) = \begin{cases} 0, & \text{if } a \in \Phi(G), \\ \max(\{1 + \text{xrank}(b) \mid (c(a), c(b)) \in E^{scc}\}), & \text{if } a \in WF(G), a \notin \Phi(G), \\ +\infty, & \text{otherwise.} \end{cases}$$

Proposition 3.9. $V/\sim_{\text{xrank}^*} \succeq \textbf{stab}(\{V\})$.

Proof:
We show that two nodes that have different xrank* belong to different blocks in the coarsest stable partition. First we show that any node a with xrank$^*(a) = k$ belongs to a different block from that of any node b with xrank$^*(b) < k$ in the coarsest stable partition by induction on k.

For the base case, $\mathrm{xrank}^*(a) = 1$ belongs to a different block from that of b to $\mathrm{xrank}^*(b) = 0$ because b with $\mathrm{xrank}^*(b) = 0$ must be a leaf node.

Next, we show the inductive step. Any node $a \in WF(G) - \Phi(G)$ with $\mathrm{xrank}^*(a) = k$ has a child node c of xrank^* $k - 1$. On the other hand, a node b with $\mathrm{xrank}^*(b) < k$ cannot have a child d with $\mathrm{xrank}^*(d) = k - 1$. By the inductive hypothesis, node c belongs to a different block from any child d of b on the coarsest stable partition. Therefore a and b belong to different blocks on the coarsest stable partition. So any node a with $\mathrm{xrank}^*(a) = k$ belongs to a different block from that of any node b with $\mathrm{xrank}^*(b) < k$ in the coarsest stable partition for any natural number k.

Finally, we show that node a with $\mathrm{xrank}^*(a) = \infty$ belongs to a different block from that of any node b with $\mathrm{xrank}^*(b) = m < \infty$ by induction on m because b has a child c with $\mathrm{xrank}^*(c) = m - 1$ and a does not. □

Proposition 3.10. $V/\sim_{\mathrm{xrank}^*} \succeq \mathbf{cpo}(V/\sim_{\mathrm{xrank}^*})$ holds.

Proof:
Assume that a and b are in the same block in $\mathbf{cpo}(V/\sim_{\mathrm{xrank}^*})$. Then for each child a' of a, there is a child b' in b such that a' and b' are in the same block of $V/\sim_{\mathrm{xrank}^*}$. If there is no such child, then $a, b \in \Phi(G)$ and belong to the same block in $V/\sim_{\mathrm{xrank}^*}$. If $a' \notin WF(G)$, then $\mathrm{xrank}^*(a) = \mathrm{xrank}^*(b) = \infty$, and a and b are in the same block in $V/\sim_{\mathrm{xrank}^*}$. Consider the case that $a, b \in WF(G)$ and $a, b \notin \Phi(G)$. There is a child a' of a such that $\mathrm{xrank}^*(a) = 1 + \mathrm{xrank}(a')$. Because a and b are in the same block of $\mathbf{cpo}(V/\sim_{\mathrm{xrank}^*})$, there must be a child b' of b in the same block to a' in $V/\sim_{\mathrm{xrank}^*}$ and a' and b' have the same xrank*. So, $\mathrm{xrank}^*(a) \leq \mathrm{xrank}^*(b)$. By exchanging a and b, we have $\mathrm{xrank}^*(a) = \mathrm{xrank}^*(b)$. So, a and b are in the same block in $V/\sim_{\mathrm{xrank}^*}$. This exhausts the cases. □

The complexity of calculating xrank* is $O(|E| + |V|)$.

3.4. xrank$_s$

Another type of rank (called rank$_s$ in [8]) is defined for calculating simulation. We call this rank xrank$_s$.

Definition 3.11. $\mathrm{xrank}_s(a)$ is defined as:

$$\mathrm{xrank}_s(a) = \begin{cases} 0, & \text{if } c(a) \in \Phi(G^{scc}), \\ \max(\{1 + \mathrm{xrank}_s(b) \mid (c(a), c(b)) \in E^{scc}\}), & \text{otherwise.} \end{cases}$$

The complexity of calculating xrank$_s$ is $O(|E| + |V|)$.

Example 3.12. $V/\sim_{\mathrm{xrank}_s} \succeq \mathbf{cpo}(V/\sim_{\mathrm{xrank}_s})$ does not hold. See Figure 4. $xrank_s(a) = 1$ and $xrank_s(b) = 2$, but a and b are in the same block of $\mathbf{cpo}(V/\sim_{\mathrm{xrank}_s})$.

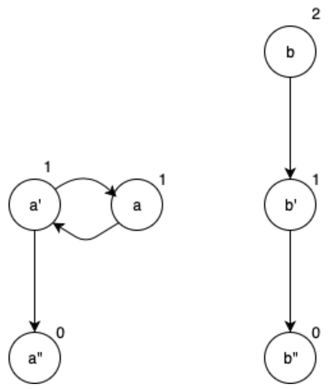

Figure 4. Example in which **cpo** does not refine partitions for $xrank_s$ and xrank'.

3.5. crank

Here, we explore other possibilities of using strongly connected components to calculate a rank.

Definition 3.13. crank(a) is defined as:

$$\text{crank}(a) = \begin{cases} 0, & \text{if } a \in \Phi(G), \\ -\infty, & \text{if } c(a) \in \Phi(G^{scc}) \\ & \text{and } a \notin \Phi(G), \\ \max(\{1 + \text{crank}(b) \mid (c(a), c(b)) \in E^{scc}\}), & \text{otherwise.} \end{cases}$$

Example 3.14. $V/\sim_{\text{crank}} \succeq$ **stab**$(\{V\})$ does not hold in general. Consider the directed graph shown in Figure 5. The numbers in the figure are the cranks of the nodes. In the coarsest stable partition, the above two nodes are in a single block, but nodes in the block have different cranks: 1 and 2. Moreover, we can see that $V/\sim_{crank} \succeq$ **cpo**$(V/\sim_{crank})$ does not hold from the figure.

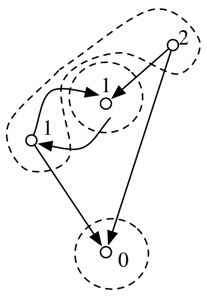

Figure 5. Crank and the coarsest stable partition. Dotted ovals enclose the coarsest stable partition.

Example 3.15. nrank, xrank, crank, xrank*, and xrank$_s$ of a sample directed graph are shown in Figure 6.

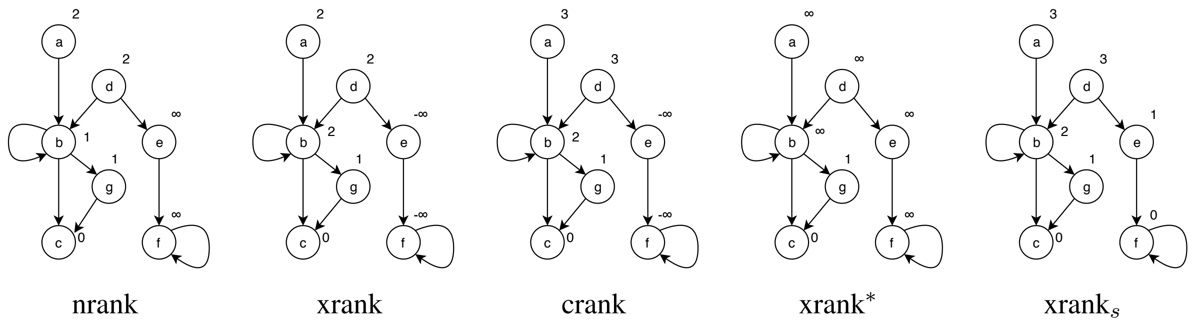

Figure 6. nrank (left), xrank (middle left), crank (center), xrank* (middle right), and xrank$_s$ (right) of a sample directed graph.

$V/\sim_{\text{nrank}} \succeq$ **stab**$(\{V\})$ by Proposition 3.4, $V/\sim_{\text{xrank}} \succeq$ **stab**$(\{V\})$ by [5], $V/\sim_{\text{xrank}^*} \succeq$ **stab**$(\{V\})$ by Proposition 3.9, and $V/\sim_{\mathbf{0}} = \{V\} \succeq$ **stab**$(\{V\})$. In Figure 6, e and f are bisimilar, but $\text{xrank}_s(e) \neq \text{xrank}_s(f)$. So, $V/\sim_{\text{xrank}_s} \succeq$ **stab**$(\{V\})$ does not hold.

4. Higher-order rank function

Suppose that each node a is assigned a rank value $r(a)$ by the rank function described in Section 3. Then, we can calculate a new rank value $r'(a)$ for node a by aggregating to it the rank values of its children by some method. We call a rank function constructed from another rank function a *higher-order rank function* and the initial rank function the *base rank function*. The return value of the higher-order rank function is called *a higher-order rank value*.

The higher-order rank value $r'(a)$ may differentiate more nodes than $r(a)$ does. But when does this happen? The same method can be used to calculate another higher-order rank value $r''(a)$ for node a by aggregating the higher-order rank value $r'(b)$ of its child b. Hence, there are infinitely many higher-order rank values. We make a distinction among the higher-order rank values by the order. The base rank value is the higher-order rank value of order 0. A higher-order rank value constructed from higher-order rank values of order k is of order $k+1$. By increasing the order, how far we can differentiate nodes? In this section, we will answer these questions.

4.1. xrank′

The rank function defined in Definition 6.3 of [5], where it is denoted as rank', will be denoted as xrank′ in this paper. xrank′ is defined as

$$\text{xrank}'(a) = \begin{cases} \{0\}, & \text{if } a \in \Phi(G), \\ \{-\infty\}, & \text{if } c(a) \in \Phi(G^{scc}) \\ & \text{and } a \notin \Phi(G), \\ \{1 + m' \mid (c(a), c(m)) \in E^{scc}, & \text{otherwise} \\ \quad m \in WF(G), m' \in \text{xrank}'(m)\} & \\ \cup\{m' \mid (c(a), c(m)) \in E^{scc}, & \\ \quad m \notin WF(G), m' \in \text{xrank}'(m)\}. & \end{cases}$$

In xrank, $\{1 + m' \mid (c(a), c(m)) \in E^{scc}, m \in WF(G), m' \in \text{xrank}'(m)\} \cup \{m' \mid (c(a), c(m)) \in E^{scc}, m \notin WF(G), m' \in \text{xrank}'(m)\}$ is summarized by $\max$. In xrank′, information other than the maximum is represented as sets. In [5], it was proposed to use a bit vector for efficient handling the sets. The computational complexity of xrank′ is $O(|E| + |V|)$, neglecting $O(|V|)$ overhead incurred by handling the bit vector.

They showed $V/\!\sim_{\text{xrank}'} \succeq \textbf{stab}(\{V\})$. To get the coarsest stable partition from $V/\sim_{xrank'}$, they employed an additional refining operation described in Algorithm 4 of [5]. The overall complexity they claimed was $O(|E| \log |V|)$.

Example 4.1. $V/\!\sim_{\text{xrank}'} \succeq \textbf{cpo}(V/\!\sim_{\text{xrank}'})$ does not hold. See Figure 4. $xrank'(a) = \{1\}$ and $xrank'(b) = \{2\}$, but a and b are in the same block of $\textbf{cpo}(V/\!\sim_{\text{xrank}'})$.

4.2. arank

The literature [7] proposed to use a rank $<\text{xrank}(a), \mathbb{A}(a)>$ for node a. $\mathbb{A}(a)$ is a unique id assigned to the node a to distinguish it from other nodes with the same xrank with a. $\mathbb{A}(a)$ is determined as follows. Let $\gtrdot_i (a)$ be a function such that $\gtrdot_i (a)(b)$ is 1 if b is a child of a and 0 otherwise. $\mathbb{A}(a)$ is a unique id given to $\gtrdot_i (a)$. If $\gtrdot_i (a)$ and $\gtrdot_i (a')$ are the same as a function, they are identified and given the same id. This identification is the most difficult part of this algorithm, and OBDD (Ordered Binary Decision Diagram) is used for this purpose. Then, let $\text{arank}(a) =< \text{xrank}(a), \mathbb{A}(a) >$.

$V/\!\sim_{\text{arank}} = \textbf{stab}(\{V\})$ holds because $\mathbb{A}(a)$ is assigned in such a way as to distinguish a from other nodes with the same xrank by using the coarsest stable partition algorithm of [6].

4.3. H_r^k

How high a rank can we achieve with a higher-order rank function? Section 4.2 showed that some higher-order rank values can induce a partition coarser than the coarsest stable partition. In this section, we show that some higher-order rank values for sufficiently large order k induce the coarsest stable partition. We show this in a purely mathematical framework. The higher-order rank value is uniquely determined from the graph structure and the base rank function. This is different from the encoding in Section 4.2 and is desirable when we compare the higher-order rank values as in Sections 5.3 and 5.4.

Our higher-order rank function H_r^k for a base rank function r with order k is defined as follows.

Definition 4.2. H_r^k is a higher-order rank function on V with order k and base rank function r such that $\mathrm{H}_r^k(a) = \{\mathrm{H}_r^{k-1}(b) \mid (a, b) \in E\}$ for $k > 0$ and $\mathrm{H}_r^0(a) = r(a)$ for node $a \in V$.

For example, H_r^1 of a node whose children's H_r^0 are 0 and 1 is $\{0, 1\}$.

Note that $\mathrm{H}_r^k(a)$ is a well-founded set.

Example 4.3. Figure 7 shows $\mathrm{H}_{\text{nrank}}^1$ of the graph in Figure 3.

If $V/\!\sim_r \succeq \textbf{stab}(\{V\})$, the partition of a higher-order rank function H_r^k for a sufficiently large k coincides with the coarsest stable partition. This will be proved in Corollary 4.5.

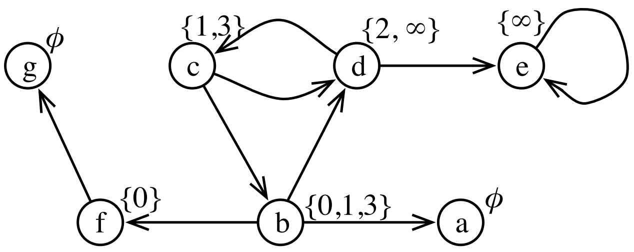

Figure 7. Higher-order rank $\mathrm{H}^{\,1}_{\mathrm{nrank}}$ for the graph in Figure 3.

Theorem 4.4. $V/\sim_{\mathrm{H}^{\,k}_{r}} = \mathbf{cpo}^k(V/\sim_r)$ holds for any $k \geq 0$.

Proof:
We prove the statement by induction on k. For $k = 0$, the statement reduces to $V/\sim_r = \mathbf{cpo}^0(V/\sim_r)$ and trivially holds by definition.

The inductive hypothesis is $V/\sim_{\mathrm{H}^{\,k-1}_{r}} = \mathbf{cpo}^{k-1}(V/\sim_r)$.

Take $a, b \in V$ so that $\mathrm{H}^{\,k}_{r}(a) = \mathrm{H}^{\,k}_{r}(b)$. $\mathrm{H}^{\,k}_{r}(a) = \{\mathrm{H}^{\,k-1}_{r}(a')|(a,a') \in E\}$ by definition. For each $\mathrm{H}^{\,k-1}_{r}(a')$, there exists b' such that $\mathrm{H}^{\,k-1}_{r}(a') = \mathrm{H}^{\,k-1}_{r}(b')$ and $(b,b') \in E$. From the inductive hypothesis, a' and b' are in the same block in $\mathbf{cpo}^{k-1}(V/\sim_r)$. This is true for every a'. So, a and b are in the same block in $\mathbf{cpo}^k(V/\sim_r)$. This proves $V/\sim_{\mathrm{H}^{\,k}_{r}} \preceq \mathbf{cpo}^k(V/\sim_r)$.

Now assume that a and b are in the same block of $\mathbf{cpo}^k(V/\sim_r)$. Then, for every child a' of a, there is a child b' of b such that a' and b' are in the same block of $\mathbf{cpo}^{k-1}(V/\sim_r)$. From the inductive hypothesis, $\mathrm{H}^{k-1}_{r}(a') = \mathrm{H}^{k-1}_{r}(b')$. So, $\mathrm{H}^{k}_{r}(a)$ and $\mathrm{H}^{k}_{r}(b)$ have the same children and $\mathrm{H}^{k}_{r}(a) = \mathrm{H}^{k}_{r}(b)$. This proves $V/\sim_{\mathrm{H}^{\,k}_{r}} \succeq \mathbf{cpo}^k(V/\sim_r)$. Now we have $V/\sim_{\mathrm{H}^{\,k}_{r}} = \mathbf{cpo}^k(V/\sim_r)$, which completes the induction step. □

Corollary 4.5. Assume that a directed graph (V, E) is given with a rank function r on V. If $V/\sim_r \succeq \mathbf{stab}(\{V\})$, $V/\sim_{H^k_r} = \mathbf{stab}(\{V\})$ for a sufficiently large k. $|V| - 1$ is sufficient for such k.

Proof:
If $V/\sim_r \succeq \mathbf{stab}(\{V\})$, for sufficiently large k, $\mathbf{cpo}^k(V/\sim_r) = \mathbf{stab}(\{V\})$ by Theorem 2.20. Theorem 4.4 yields $V/\sim_{H^k_r} = \mathbf{cpo}^k(V/\sim_r)$. □

The minimum such k is called the *minimum separating order* (*MSO* for short) of G. The MSO depends on the base rank function, and we will mention whose MSO it is if necessary.

$V/\sim_{\mathrm{H}^{\,0}_{r}} \succeq V/\sim_{\mathrm{H}^{\,1}_{r}}$ does not hold in general. Example 2.14 is a counterexample. However, the following holds.

Corollary 4.6. If $V/\sim_r \succeq \mathbf{cpo}(V/\sim_r)$, $V/\sim_{\mathrm{H}^{\,k}_{r}} = \mathbf{div}^k(V/\sim_r)$ holds for any $k \geq 0$. So, $V/\sim_{\mathrm{H}^{\,k}_{r}} = \mathbf{stab}(\{V\})$ for sufficiently large k.

Proof:
The statement follows from Theorem 4.4 and Lemma 2.17. □

Example 4.7. Consider the directed graph shown in Figure 8. H^{0}_{nrank}, $\mathrm{H}^{1}_{\mathrm{nrank}}$, and $\mathrm{H}^{2}_{\mathrm{nrank}}$ are shown in Figure 8 left, middle, and right, respectively. The nodes with the same rank are enclosed by dashed ovals. In this example, $V/\sim_{\mathrm{H}^{2}_{nrank}} = \mathbf{stab}(\{V\})$, and MSO = 2.

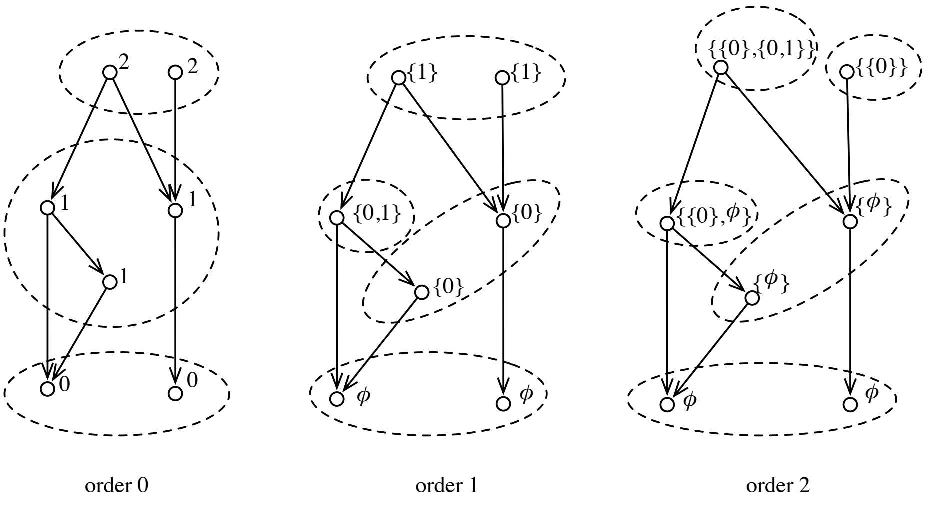

Figure 8. Partitions according to orders 0, 1, and 2 induced by nrank.

Example 4.8. Figure 9 shows the graph in Figure 2 of [5].

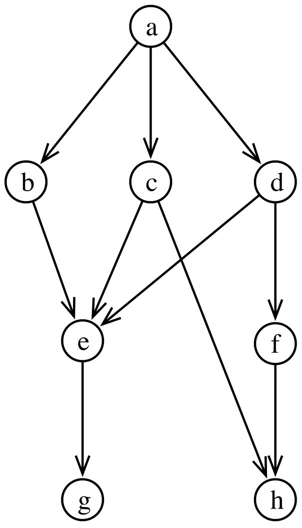

Figure 9. The graph in Figure 2 of [5].

$\mathrm{H}^{k}_{\mathrm{nrank}}$ for $k = 0, 1, 2$ are shown below.

node	nrank	H^1_{nrank}	H^2_{nrank}
a	2	$\{1,2\}$	$\{\{1\},\{1,0\}\}$
b	2	$\{1\}$	$\{\{0\}\}$
c	1	$\{1,0\}$	$\{\{0\},\emptyset\}$
d	2	$\{1\}$	$\{\{0\}\}$
e	1	$\{0\}$	$\{\emptyset\}$
f	1	$\{0\}$	$\{\emptyset\}$
g	0	$\emptyset$	$\emptyset$
h	0	$\emptyset$	$\emptyset$

As can be seen, each pair of nodes b and d, nodes e and f, and nodes g and h has the same H^1_{nrank} and no pair can be separated by advancing to the next order. This confirms that each pair of b and d, e and f, and g and h is bisimilar.

$V/\sim_{H^k_0}=$ **stab**$(\{V\})$ for sufficiently large k from Corollary 4.5.

Example 4.9. Shown below are $H^k_{\text{xrank}^*}$ in Figure 6 for $k = 0, 1, 2$.

node	xrank*	$H^1_{\text{xrank}^*}$	$H^2_{\text{xrank}^*}$
a	∞	$\{\infty\}$	$\{\{0,1\}\}$
b	∞	$\{0,1,\infty\}$	$\{\{0,1,\infty\},\{0\},\emptyset\}$
c	0	$\emptyset$	$\emptyset$
d	∞	$\{\infty\}$	$\{\{0,1,\infty\},\{\infty\}\}$
e	∞	$\{\infty\}$	$\{\{\infty\}\}$
f	∞	$\{\infty\}$	$\{\{\infty\}\}$
g	1	$\{0\}$	$\{\emptyset\}$

From Proposition 3.10 and Theorem 4.6, $V/\sim_{H^k_{\text{xrank}^*}} =$ **stab**$(\{V\})$.

Example 4.10. Shown below are $H^k_{\text{xrank}_s}$ in Figure 6 for $k = 0, 1, 2$.

node	xrank_s	$H^1_{\text{xrank}_s}$	$H^2_{\text{xrank}_s}$
a	3	$\{2\}$	$\{\{0,1,2\}\}$
b	2	$\{0,1,2\}$	$\{\emptyset,\{0\},\{0,1,2\}\}$
c	0	$\emptyset$	$\emptyset$
d	3	$\{1,2\}$	$\{\{0,1,2\},\{0\}\}$
e	1	$\{0\}$	$\{\{0\}\}$
f	0	$\{0\}$	$\{\{0\}\}$
g	1	$\{0\}$	$\{\emptyset\}$

$\mathrm{H}^0_{\mathrm{xrank}_s}(e) \neq \mathrm{H}^0_{\mathrm{xrank}_s}(f)$ and $\mathrm{H}^1_{\mathrm{xrank}_s}(e) = \mathrm{H}^1_{\mathrm{xrank}_s}(f)$. So, $V/\sim_{\mathrm{xrank}_s} \not\succeq \mathbf{cpo}(V/\sim_{\mathrm{xrank}_s})$. So, Corollary 4.6 can not be applied to xrank_s.

Example 4.11. $V/\sim_{\mathrm{xrank}} \succeq \mathbf{cpo}(V/\sim_{\mathrm{xrank}})$ does not hold. An example is shown in Figure 10.

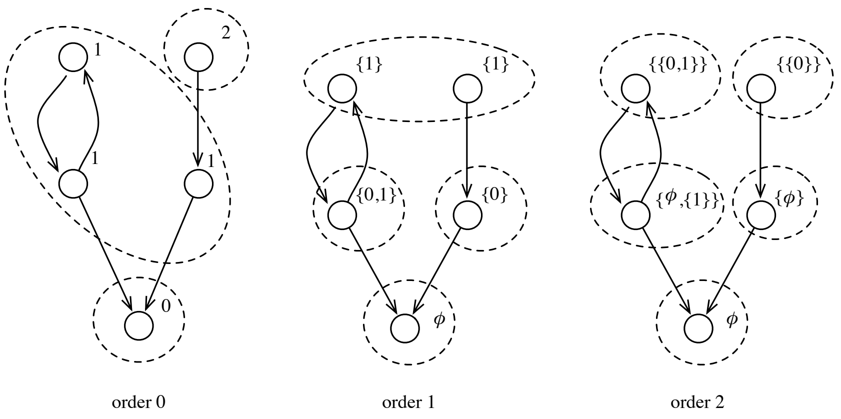

Figure 10. Partitions according to orders 0, 1, and 2 induced by xrank.

The above results are summarized as follows.

base rank function	$V/\sim_r$ $\succeq \mathbf{cpo}(V/\sim_r)$	$V/\sim_r$ $\succeq \mathbf{stab}(\{V\})$	$V/\sim_{H^k_r}$ $= \mathbf{stab}(\{V\})$
0	yes	yes, Example 3.15	yes
nrank	yes, Proposition 3.3	yes, Proposition 3.4	yes, Corollary 4.5 or Corollary 4.6 (Example 4.7)
xrank	no, Example 4.11	yes, [5]	yes, Corollary 4.5
xrank*	yes, Proposition 3.10	Proposition 3.9	yes, Corollary 4.5 or Corollary 4.6 (Example 4.9)
xrank_s	no, Example 3.12	no, Example 3.15	no, Example 4.10
crank	no, Example 3.14	no, Example 3.14	no, Example 4.12
xrank'	no, Example 4.1	yes, [5]	yes, Corollary 4.5
arank	yes	yes	yes, [7]

Example 4.12. For crank and xrank_s, there is a case that $V/\sim_{H^k_r} = \mathbf{stab}(\{V\})$ does not hold for any k. In Figure 11, e and f are in the same block of the coarsest stable partition. The crank of e is 2 and that of f is 3, $H^1_{crank}(e) = \{1,2\}$ and $H^1_{crank}(f) = \{2,3\}$, and $H^2_{crank}(e) = \{\{\{1\},\emptyset\},\{1,2\}\}$ and $H^2_{crank}(f) = \{\{\{1\},\emptyset\},\{2,3\}\}$. Because e and f each have an edge to themselves, $H^k_{crank}(e) \neq$

$H^k_{crank}(f)$ for any k. For xrank$_s$, $H^2_{xrank_s}(e) = \{\{0,1\},\{1\}\}$ and $H^2_{xrank_s}(e) = \{\{0,1\},\{2\}\}$, and $H^k_{xrank_s}(e) \neq H^k_{xrank_s}(f)$ for any k.

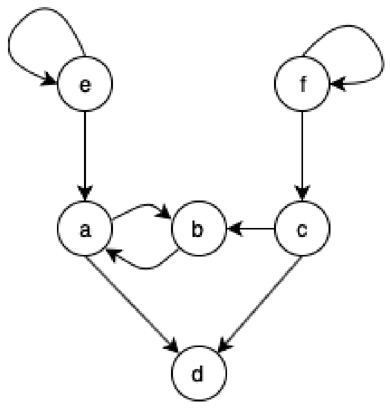

Figure 11. Counterexample of $V/\sim_{H^k_r} =$ **stab**$(\{V\})$ for crank and xrank$_s$.

To get a higher-order rank values up to order k it can be obtained with a computation of complexity $O(k(|E|+|V|))$. For calculating **cpo**, we refine V by using the blocks in P. This operation is simpler than **div** but its computational complexity is still $O(|E|+|V|)$.

In the worst case, the least k in Theorem 2.20 is $|V|-1$. So, the computational complexity of the coarsest stable partition using **stab** is $O(|V|(|E|+|V|))$. This is worse than $O(|E|\log|V|)$ of [6].

We think that the order at which most nodes are separated is much smaller than $|V|$ and that an on-demand calculation of higher-order rank values may save most of the useless calculations. However, an elaboration of this algorithm is beyond the scope of this paper.

5. Applications

5.1. Reconstruction of G from H^k_r

Can the entire topology be encoded into H^k_r? We will rephrase this question as "How can the graph G can be reconstructed from H^k_r"?

Lemma 5.1. If G has no root, then $\mathrm{H}^k_r[V]$ can be reconstructed from $\mathrm{H}^{k'}_r[V]$ for $k' > k$.

Proof:
$\mathrm{H}^{k-1}_r[V] = \bigcup \mathrm{H}^k_r[V]$. □

Definition 5.2. For a subset s in $\{\mathrm{H}^1_{nrank}(a) | a \in V\}$, define $g(s) = \begin{cases} 0, & \text{if } s = \emptyset, \\ (\min s) + 1, & \text{otherwise.} \end{cases}$

Note that $g(\{\infty\}) = \infty$ because $\infty + 1 = \infty$.

Lemma 5.3. If G has no root, $g(\mathrm{H}\,^{1}_{nrank}(a)) = \mathrm{H}\,^{0}_{nrank}(a)$ for $a \in V$.

Proof:
If $a \in \Phi(G)$, $g(H^1_{nrank}(a)) = g(\emptyset) = 0 = H^0_{nrank}(a)$. If no leaf is reachable from a, no leaf is reachable from a's children. $g(H^1_{nrank}(a)) = g(\{\infty\}) = \infty = H^0_{nrank}(a)$. Otherwise, the minimum length to a leaf from a is the minimum length to a leaf from one of the children of a plus 1. $g(H^1_{nrank}(a)) = g(\{\infty, n_1, n_2, ..., n_m\}) = \min(\{n_1, n_2, ..., n_m\}) + 1 = H^0_{nrank}(a)$. □

This is a good property of nrank.

Lemma 5.4. If G has no root, we can extend the function g to $\{\mathrm{H}\,^{i}_{nrank}(a) | a \in V\}$ such that $g(\mathrm{H}\,^{i}_{nrank}(a)) = \mathrm{H}\,^{i-1}_{nrank}(a)$ for $a \in V$ and $1 \leq i \leq k$.

Proof:
We can recursively construct g by defining $g(x) = \{g(y) \mid y \in x\}$ for $x \in H^i_{nrank}(a)$ for $i > 1$. The function g is well-defined because if $H^i_{nrank}(a) = H^i_{nrank}(b)$ then $H^{i-1}_{nrank}(a) = H^{i-1}_{nrank}(b)$ from Lemma 2.17 and Theorem 4.4 for $i > 1$ and from the definitions of g and nrank for $i = 1$. □

This function g is called the Mostowski's collapse in the literature [10].

Theorem 5.5. Suppose that G has no root. If the MSO is k with respect to nrank, then the graph $G' = (V', E')$ with $V' = \mathrm{H}\,^{k}_{nrank}[V]$ and $E' = \{(g(v), v') \mid v \in \mathrm{H}\,^{k+1}_{nrank}[V], v' \in v\}$ is bisimilar to the original graph G.

Proof:
Suppose that $(a, b) \in E$, then for $v = H^{k+1}_{nrank}(a)$, $g(v) = H^k_{nrank}(a)$ by Lemma 5.4. For $v' = H^k_{nrank}(b)$, $v' \in v = H^{k+1}_{nrank}(a)$. So, $(g(v), v') = (H^k_{nrank}(a), H^k_{nrank}(b)) \in E'$. Define the bisimulation relation $\asymp$ on $G \times G'$ as $a \asymp H^k_{nrank}(a)$. For $(a, b) \in E$, $(H^k_{nrank}(a), H^k_{nrank}(b)) \in E'$ and $b \asymp H^k_{nrank}(b)$. Conversely, suppose that $(H^k_{nrank}(a), c) \in E'$. Suppose that no e exists such that $(a, e) \in E$ and $e \asymp c$. Then $H^{k+1}_{nrank}(a)$ does not have c as an element, a contradiction. So, there exists e such that $(a, e) \in E$ and $e \asymp c$. This completes the proof that G and G' are bisimilar. □

Example 5.6. The MSO of G in Figure 3 is 1. So, G can be reconstructed from $\mathrm{H}\,^{2}_{\mathrm{nrank}}[V]$.

node	nrank	$\mathrm{H}\,^{1}_{\mathrm{nrank}}$	$\mathrm{H}\,^{2}_{\mathrm{nrank}}$
a	0	$\emptyset$	$\emptyset$
b	1	$\{0, 1, 3\}$	$\{\emptyset, \{0\}, \{2, \infty\}\}$
c	2	$\{1, 3\}$	$\{\{0, 1, 3\}, \{2, \infty\}\}$
d	3	$\{2, \infty\}$	$\{\{1, 3\}, \{\infty\}\}$
e	∞	$\{\infty\}$	$\{\{\infty\}\}$
f	1	$\{0\}$	$\{\emptyset\}$
g	0	$\emptyset$	$\emptyset$

Suppose that we just have $\mathrm{H}^{2}_{\mathrm{nrank}}[V] = \{\emptyset, \{\emptyset, \{0\}, \{2, \infty\}\}, \{\{0, 1, 3\}, \{2, \infty\}\}, \{\{1, 3\}, \{\infty\}\}, \{\{\infty\}\}, \{\emptyset\}\}$.

First, we reconstruct $\mathrm{H}^{k}_{\mathrm{nrank}}[V] (k < 2)$ from $\mathrm{H}^{2}_{\mathrm{nrank}}[V]$. $\mathrm{H}^{1}_{\mathrm{nrank}}[V] = \bigcup \mathrm{H}^{2}_{\mathrm{nrank}}[V] = \{\emptyset, \{0, 1, 3\}, \{1, 3\}, \{2, \infty\}, \{\infty\}, \{0\}\}$. $\mathrm{H}^{0}_{\mathrm{nrank}}[V] = \bigcup \mathrm{H}^{1}_{\mathrm{nrank}}[V] = \{0, 1, 2, 3, \infty\}$.

By applying g to each element of $\mathrm{H}^{1}_{\mathrm{nrank}}[V]$, we get $g(\emptyset) = 0$, $g(\{0, 1, 3\}) = 1$, $g(\{1, 3\}) = 2$, $g(\{2, \infty\}) = 3$, $g(\{\infty\}) = \infty$, $g(\{0\}) = 1$, and $g(\emptyset) = 0$. g is a function from $H^1_r(a)$ to $H^0_r(a)$ for $a \in V$. We construct a function g from $H^2_r(a)$ to $H^1_r(a)$ for $a \in V$ as follows. $g(\emptyset) = \emptyset$, $g(\{\emptyset, \{0\}, \{2, \infty\}\}) = \{g(\emptyset), g(\{0\}), g(\{2, \infty\})\} = \{0, 1, 3\}$, $g(\{\{0, 1, 3\}, \{2, \infty\}\}) = \{g(\{0, 1, 3\}), g(\{2, \infty\})\} = \{1, 3\}$, and so on. Now we have $V' = \mathrm{H}^{1}_{\mathrm{nrank}}[V] = \{\emptyset, \{0, 1, 3\}, \{1, 3\}, \{2, \infty\}, \{\infty\}, \{0\}\}$, and $E' = \{(g(v), v') | v \in \mathrm{H}^{2}_{\mathrm{nrank}}[V], v' \in v\} = \{(\{0, 1, 3\}, \emptyset), (\{0, 1, 3\}, \{0\}), (\{0, 1, 3\}, \{2, \infty\}), ...\}$. Then, $G' = (V', E')$ is bisimilar to G.

5.2. Binary encoding

Example 5.7. The MSO of G in Figure 3 with respect to the rank function **0** is 4 as shown below.

node	**0**	$\mathrm{H}^{1}_{\mathbf{0}}$	$\mathrm{H}^{2}_{\mathbf{0}}$	$\mathrm{H}^{3}_{\mathbf{0}}$	$\mathrm{H}^{4}_{\mathbf{0}}$
a	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
b	0	$\{0\}$	$\{\emptyset, \{0\}\}$	$\{\{\emptyset\}, \{\{0\}\}, \emptyset\}$	$\{\emptyset, \{\{\{0\}\}\}, \{\emptyset\}\}$
c	0	$\{0\}$	$\{\{0\}\}$	$\{\{\emptyset\}, \{\{0\}\}\}$	$\{\{\{\{0\}\}\}, \{\{\emptyset\}, \{\{0\}\}, \emptyset\}\}$
d	0	$\{0\}$	$\{\{0\}\}$	$\{\{\{0\}\}\}$	$\{\{\{\{0\}\}\}, \{\{\{0\}\}, \{\emptyset\}\}\}$
e	0	$\{0\}$	$\{\{0\}\}$	$\{\{\{0\}\}\}$	$\{\{\{\{0\}\}\}\}$
f	0	$\{0\}$	$\{\emptyset\}$	$\{\emptyset\}$	$\{\emptyset\}$
g	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$\mathrm{H}^{k}_{\mathbf{0}}$ is acyclic, and we can construct an encoding for higher-order rank values in a similar way to the Ackermann encoding.

Definition 5.8. A higher-order encoding of order k for the rank function r is defined as follows. $A^k_r(a) = \begin{cases} r(a), \text{ if } k = 0, \\ \vee_{b \in a} 2^{A^{k-1}_r(b)}, \text{ otherwise }. \end{cases}$ Here, $\vee$ is the bitwise logical OR of numbers in the binary representation.

Example 5.9. Higher-order encodings of G with the base rank function **0** in Figure 3 are shown below up to $k = 4$.

node	**0**	A_0^1	A_0^2	A_0^3	A_0^4
a	0	0	00	0000	0000000000000000
b	0	1	11	0111	0000000000010011
c	0	1	10	1100	0000000010010000
d	0	1	10	0100	0001000000010000
e	0	1	10	0100	0000000000010000
f	0	1	01	0001	0000000000000001
g	0	0	00	0000	0000000000000000

Theorem 5.10. $A_r^k(a) = A_r^k(b) \Leftrightarrow \mathrm{H}\,_r^k(a) = \mathrm{H}\,_r^k(b)$.

Proof:
This statement follows from the fact that A_r^k is a faithful binary representation of $\mathrm{H}\,_r^k$ for $k > 0$. □

Note that this higher-order encoding is uniquely determined while the encoding $\mathbb{A}(a)$ in Section 4.2 is not uniquely determined.

5.3. Merging graphs

One can compare two nodes in different graphs by using their higher-order rank values.

If $H_r^k(a) \neq H_r^k(b)$, a and b never become bisimilar.

What can we say if $H_r^k(a) = H_r^k(b)$ holds? Are a and b are bisimilar in $G_1 \cup G_2$? Unfortunately, even if $H_r^k(a) = H_r^k(b)$ holds, $H_r^{k'}(a) = H_r^{k'}(b)$ may not hold for some $k' \gg k$.

Example 5.11. Figure 12 shows an example that (MSO of $G_1 \cup G_2$) $\gg$ (MSO of G_1) and (MSO of G_2).

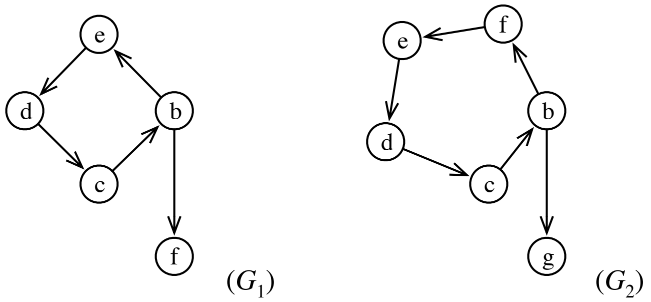

Figure 12. Sample graphs G_1 and G_2 such that (MSO of $G_1 \cup G_2$) $\gg$ (MSO of G_1) and (MSO of G_2).

$\mathrm{H}\,_{\mathrm{nrank}}^k$ for Figure 12 (G_1) is shown below. The MSO is 0, but for later use, we show $\mathrm{H}\,_{\mathrm{nrank}}^k$ up to $k = 4$.

	H^0_{nrank}	H^1_{nrank}	H^2_{nrank}	H^3_{nrank}	H^4_{nrank}
b	1	$\{0,4\}$	$\{\emptyset,\{3\}\}$	$\{\emptyset,\{\{2\}\}\}$	$\{\emptyset,\{\{\{1\}\}\}\}$
c	2	$\{1\}$	$\{\{0,4\}\}$	$\{\{\emptyset,\{3\}\}\}$	$\{\{\emptyset,\{\{2\}\}\}\}$
d	3	$\{2\}$	$\{\{1\}\}$	$\{\{\{0,4\}\}\}$	$\{\{\{\emptyset,\{3\}\}\}\}$
e	4	$\{3\}$	$\{\{2\}\}$	$\{\{\{1\}\}\}$	$\{\{\{\{0,4\}\}\}\}$
f	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

H^k_{nrank} for Figure 12 (G_2) is shown below. The MSO is again 0, but for later use, we show H^k_{nrank} up to $k = 4$.

	H^0_{nrank}	H^1_{nrank}	H^2_{nrank}	H^3_{nrank}	H^4_{nrank}
b	1	$\{0,5\}$	$\{\emptyset,\{4\}\}$	$\{\emptyset,\{\{3\}\}\}$	$\{\emptyset,\{\{\{2\}\}\}\}$
c	2	$\{1\}$	$\{\{0,5\}\}$	$\{\{\emptyset,\{4\}\}\}$	$\{\{\emptyset,\{\{3\}\}\}\}$
d	3	$\{2\}$	$\{\{1\}\}$	$\{\{\{0,5\}\}\}$	$\{\{\{\emptyset,\{4\}\}\}\}$
e	4	$\{3\}$	$\{\{2\}\}$	$\{\{\{1\}\}\}$	$\{\{\{\{0,5\}\}\}\}$
f	5	$\{4\}$	$\{\{3\}\}$	$\{\{\{2\}\}\}$	$\{\{\{\{1\}\}\}\}$
g	0	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

The MSOs of G_1 and G_2 are 0. However, e in G_1 and G_2 has the same H^k_{nrank} for even $k = 3$, and they can be eventually separated at $k = 4$. So, the MSO of $G_1 \cup G_2$ is 4.

Suppose that we want to determine whether a node of graph G_1 and a node of G_2 are bisimilar. We assume that the higher-order rank of G_1 and that of G_2 are already calculated for the order $k \geq$ MSO of G_1 and G_2. For simplicity we additionally assume that G_1 has no two bisimilar nodes and G_2 has no two bisimilar nodes. Then for $k \geq$ MSO of G_1 and G_2, $H^k_{\text{nrank}}(a) \neq H^k_{\text{nrank}}(b)$ if $a, b \in G_1$ and $a \neq b$. This also holds for nodes in G_2. So, a node of G_1 has the same H^k_{nrank} with at most one node of G_2. The partition of H^k_{nrank} for $G_1 \cup G_2$ is such a pair or a singleton. Now, the problem is to decide whether we should divide the pair or not in $G_1 \cup G_2$ which can be solved with a simple partition algorithm, for example, the one shown in Algorithm 1.

Algorithm 1 is a variant of the minimization algorithm of DFA [11]. The computational complexity [2] of the minimization algorithm of DFA for a set of symbols Σ is $O(|\Sigma||V|\log|V|)$. In our case, the possible pairs are limited by $|V_1|$ and $|V_2|$ while those of the edges are limited by $|E_1|$ and $|E_2|$. As such, the computational complexity is $O(|E_1| + |E_2|)$. This is smaller than the computational complexity $O((|E_1| + |E_2|)\log(|V_1| + |V_2|))$ of calculating the coarsest stable partition of the entire graph $G_1 \cup G_2$ by employing (from scratch) the algorithm in [5] and [6]. H^k_{nrank} gets complicated

[2]There have been some improvements in the original algorithm, but the worst-case complexity is still $O(|\Sigma||V|\log|V|)$.

Algorithm 1 Separating pairs algorithm

```
1: $P \leftarrow$ node pairs with the same $H_r^k$. {To process pairs}
2: $P' \leftarrow \emptyset$. {Processed pairs}
3: for $p \in P$ do
4:     move $p$ from $P$ to $P'$.
5:     $(a, b) \leftarrow p$.
6:     for $a' \in a$ do
7:         if there does not exist $b' \in b$ s.t. $(a', b') \in P \cup P'$ or $a' = b'$ then
8:             Transitively divide all the pairs in the list associated with $(a, b)$.
9:             Remove $p$ and the pairs from $P'$.
10:            Break
11:        end if
12:    end for
13:    Do the same for $b$.
14:    if $p \in P'$ then
15:        Put $(a, b)$ into a list associated with $(a', b')$.
16:    end if
17: end for
```

for larger k. The cost to compare such complicated higher-order rank values is not $O(1)$. However $|V|$ is much smaller than the number of possible return values of H_{nrank}^k. So, if we can find a proper hash function, we can effectively compare the return values of H_{nrank}^k in $O(1)$.

Example 5.12. For Example 5.11, the MSO of G_1 and G_2 is 0. Thus, c of G_1 and c of G_2 have the same H_{nrank}^0. H_{nrank}^0 of b, c, d, and e are 1, 2, 3 and 4 for both G_1 and G_2. Now, $P = \{(d, d'), (c, c'), (b, b'), (e, e'), (f, g')\}$, where the nodes in G_2 are apostrophized to make a distinction to the nodes in G_1.

Suppose (d, d') is processed first. Then for child c of d, there exists c' of d' such that $(c, c') \in P$. So, (d, d') is put into the list associated with (c, c') to be processed later. Suppose (c, c') is processed next. Then for child b of c, b' of c' exists such that $(b, b') \in P$. So, (c, c') is put into the list associated to (b, b') to be processed later. Suppose (b, b') is processed next. Then for child e of b, no child x' of b' exists such that $(e, x') \in P$ nor $e = x'$. So we divide (b, b') and following the list associated with (b, b'), we transitively divide (c, c') and (d, d'). Suppose (e, e') is processed next. Then for child d of e, no child d' of e' exists such that $(d, d') \in P$ because (d, d') was already divided. So, we divide (e, e'). Suppose (f, g') is processed last. Because f and g' do not have a child, (f, g') is left in P'. Now we know that c, d, and e of G_1 and those of G_2 are distinct and f and g' are bisimilar.

5.4. Edge increment

An incremental algorithm to maintain the coarsest stable partition was proposed in [13]. Here, we show how to incrementally calculate the coarsest stable partition by using the higher-order rank. We use the example in Figure 13 of [13].

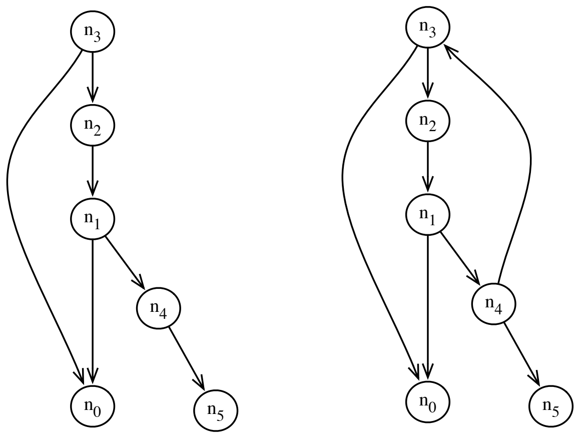

Figure 13. Sample increment of an edge $n_4 \to n_3$ in Figure 10 of the graph in [13]. Before (left) and After (right).

For the graph in Figure 13 (left), the higher-order ranks are as follows.

	H^0_{nrank}	H^1_{nrank}	H^2_{nrank}
n_0	0	$\emptyset$	$\emptyset$
n_1	1	$\{0,1\}$	$\{\emptyset,\{0\}\}$
n_2	2	$\{1\}$	$\{\{0,1\}\}$
n_3	1	$\{0,2\}$	$\{\emptyset,\{1\}\}$
n_4	1	$\{0\}$	$\{\emptyset\}$
n_5	0	$\emptyset$	$\emptyset$

The MSO of this graph is 1, but we show H^2_{nrank} for comparison later.

By adding an edge from $n_4 \to n_3$ as in Figure 13 (right), the higher-order ranks slightly change. The changed higher-order ranks are shown in bold in the table below.

	H^0_{nrank}	H^1_{nrank}	H^2_{nrank}
n_0	0	$\emptyset$	$\emptyset$
n_1	1	$\{0,1\}$	$\{\emptyset,\{\mathbf{0},\mathbf{1}\}\}$
n_2	2	$\{1\}$	$\{\{0,1\}\}$
n_3	1	$\{0,2\}$	$\{\emptyset,\{1\}\}$
n_4	1	$\{\mathbf{0},\mathbf{1}\}$	$\{\emptyset,\{\mathbf{0},\mathbf{2}\}\}$
n_5	0	$\emptyset$	$\emptyset$

The higher-order rank value of n_4 changes and the higher-order rank values depending on it change. Now $\mathrm{H}^1_{\text{nrank}}(n_4)$ is $\{0, 1\}$ being identical to $\mathrm{H}^1_{\text{nrank}}(n_1)$. This means that the local structure around n_4 is the same as that of n_1. n_1's children are n_0 with nrank 0 and n_4 with nrank 1, and n_4's children are n_5 with nrank 0 and n_3 with nrank 1. So, we have to calculate $\mathrm{H}^2_{\text{nrank}}$. In $\mathrm{H}^2_{\text{nrank}}$, n_1 and n_4 have different higher-order rank values and all nodes are separated.

At an edge increment, at most only one pair of nodes has the same higher-order rank at the order of the original MSO. So, we only have to calculate the higher-order rank at the order of the original MSO + 1 for the nodes in the pair. The complexity in this case is proportional to the outdegrees of the nodes and $O(|E|/|V|)$ on average.

5.5. Web graph analysis

Here, we briefly explain an application to web site analysis.

It is said that if the structure of a web site is inconsistent, users tend to get lost in it [14, 15]. By detecting and restructuring irregular structures in a web site, we can reduce such inconsistencies. By viewing web pages as nodes and links between them as edges, we can picture a web site as a directed graph. This graph is called a *web graph*. For this web graph, a higher-order rank function can be considered to produce a summary of the local link structure around each page. Each block induced by the function contains nodes with the same local link structure up to the specified order.

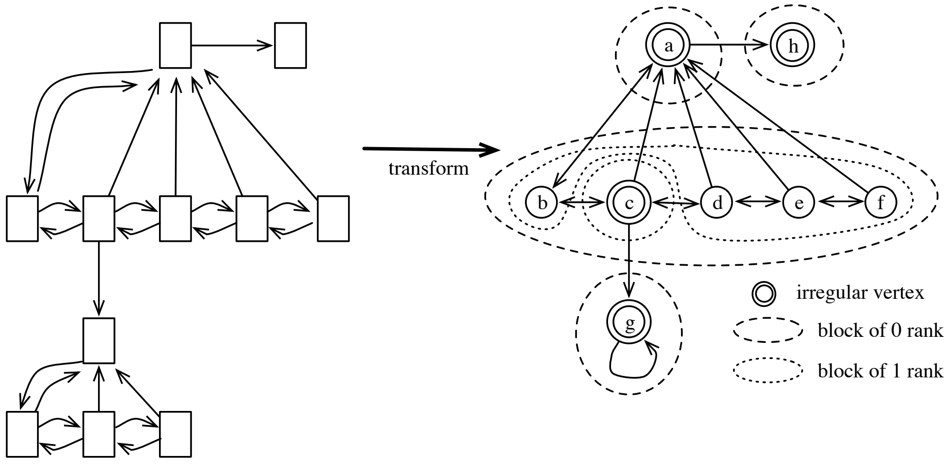

Figure 14. Web pages with a linear substructure and AFA structure with higher-order rank.

Figure 14 shows an example of a web site with pages and links (A) and the corresponding web graph (B). $V/\sim_{\mathrm{H}^0_{\text{nrank}}}$ consists of $\{b, c, d, e, f\}$, $\{a\}$, $\{h\}$, and $\{g\}$. Because the blocks $\{a\}$, $\{h\}$, and $\{g\}$ are smaller than $\{b, c, d, e, f\}$, we can consider them to be irregular. On the original web site (A), they consist of an index, an external page, and a substructure, and compared with the large linear structure of $\{b, c, d, e, f\}$, they are considered to be irregular. Next, $V/\sim_{\mathrm{H}^1_{\text{nrank}}}$ has blocks

$\{b, d, e, f\}$, $\{c\}$, $\{a\}$, $\{h\}$, and $\{g\}$. Here, $\{c\}$ can be considered to be irregular compared with the relatively large block of $\{b, d, e, f\}$. On the original web site (A), a page c is an entry to the substructure, and compared with the still large linear structure, it is considered to be irregular. In [3], it was reported that several irregular structures could be detected by using the higher-order rank function in this way.

A hub is well characterized as a node with many outgoing links in the graph theory [16]. Certain hub structures can be analyzed in more detail by using a higher-order rank function H^k_r. For example, a hub h with small $|H^i_{nrank}(h)|$ can be considered to have homogeneous children, whereas one with large $|H^i_{nrank}(h)|$ can be considered to have heterogeneous (so, irregular) children.

6. Concluding remarks

In this paper, we reviewed the properties of the partition lattice and refining operators **div**, **stab**, and **cpo**. We introduced **cpo**, which is simpler than **div** and directly corresponds to higher-order rank functions. **div** divides the current partition, so the partition depends on the previous history as well as the division caused by the children. **cpo** depends only on the children, and the partition is more homogeneous. We showed that **cpo** is not monotonic, in general, but it becomes so if certain conditions are met. For example, if $V/\sim_{\text{xrank}} \succeq$ **cpo**$(V/\sim_{\text{xrank}})$, successive application of **cpo** on $V/\sim_{\text{xrank}}$ is monotonic (Lemma 2.17). This condition is satisfied by many ranks, including nrank (Proposition 3.3). We also showed that by applying **cpo** to a partition coarser than the stable one, we eventually get the stable partition (Theorem 2.20).

We summarized the rank functions: xrank defined in [5], arank defined in [7], and xrank* and xrank$_s$ defined in [8], and showed their properties. We also proposed nrank which is closely related to **cpo** in the context of the higher-order rank.

We introduced the higher-order rank function H^k_r for order k and a rank function r. Some higher-order rank functions were proposed in [5] and [7] for specific data structures. We separated the mathematical model from the data representation and focused on the mathematical aspects of xrank' (Section 4.1) and arank (Section 4.2). In so doing, we gain flexibility in implementing a mathematical model with guaranteed convergence. H^i_r may be as complex as a well-founded set can be. However, it is defined purely mathematically, so we can employ various implementation techniques. For example, we can use a hash function suitable for the target domain in order to reduce the number of exact comparisons of H^i_r, as explained in Section 5.3.

The equivalence $\sim_{\mathrm{H}^k_r}$ directly corresponds to the refining operator **cpo**^{k} (Theorem 4.4.) This means that a higher-order rank function with sufficiently large order induces the coarsest stable partition if $V/\sim_r \succeq$ **stab**$(\{V\})$ (Corollary 4.5). We may explore a possible implementation without worrying about the limit behavior (Section 4.3). (For obtaining just the coarsest stable partition, it is better to switch to the algorithm in [6].)

cpo is more homogeneous than **div** and has better understandability. In the web application described in Section 5.5, the distinction made by **cpo**^{k} is caused by the difference at the depth[3] k and can be easily interpreted.

[3]the path length to some leaf

We also showed that the entire topology can be reconstructed from H_r^k if some conditions are met (Section 5.1).

The higher-order rank $\mathrm{H}\,_r^k(a)$ depends on the local structure of a, but is independent of the entire structure. Thus, we can compare the nodes $n \in V$ and $n' \in V'$ in different graphs $G = (V, E)$ and $G' = (V', E')$ by their higher-order ranks as shown in Section 5.3.

There are variants of the bisimulation relation. For example, simulation is used in some applications. It is said that $\preceq_S$ used in the simulation does not have a direct counter-part among set-theoretical notion, but the higher-order rank may be useful for analyzing an algorithm on the simulation. We plan to pursue this idea in the future.

This paper focused on a mathematical aspects. To obtain the exact computational complexity, we have to determine the data structures precisely. The worst-case complexity is $O(|E| \log |V|)$, as it includes pathological cases; the average complexity may be smaller. Also, for some popular graphs with good properties such as those manifested by the small-world phenomenon, the complexity may be much smaller still. We also plan to explore these possibilities in our future work.

Acknowledgements

This work was partially supported by JSPS KAKENHI Grant Number 17K01143.

References

[1] Milner R. An Algebraic Definition of Simulation Between Programs. In: Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK, September 1-3, 1971. 1971 pp. 481–489. URL `http://ijcai.org/Proceedings/71/Papers/044.pdf`.

[2] Aczel P. Non-well-founded Sets. Center for the Study of Language and Information Publication Lecture Notes. Cambridge University Press, 1988. ISBN: 9780937073223. URL `https://books.google.co.jp/books?id=9yaRQgAACAAJ`.

[3] Horie I, Yamaguchi K, Kashiwabara K. Higher-order rank analysis for web structure. In: HYPERTEXT 2005, Proceedings of the 16th ACM Conference on Hypertext and Hypermedia, September 6-9, 2005, Salzburg, Austria. 2005 pp. 98–106. doi:10.1145/1083356.1083375.

[4] Horie I, Yamaguchi K, Kashiwabara K. Pattern detection from web using AFA set theory. In: 9th ACM International Workshop on Web Information and Data Management (WIDM 2007), Lisbon, Portugal, November 9, 2007. 2007 pp. 113–120. doi:10.1145/1316902.1316921.

[5] Dovier A, Piazza C, Policriti A. An efficient algorithm for computing bisimulation equivalence. *Theor. Comput. Sci.*, 2004. **311**(1-3):221–256. doi:10.1016/S0304-3975(03)00361-X.

[6] Paige R, Tarjan RE. Three Partition Refinement Algorithms. *SIAM J. Comput.*, 1987. **16**(6):973–989. doi:10.1137/0216062. URL `https://doi.org/10.1137/0216062`.

[7] Piazza C, Policriti A. Ackermann Encoding, Bisimulations, and OBDDs. *TPLP*, 2004. **4**(5-6):695–718. doi:10.1017/S1471068404002091.

[8] Gentilini R, Piazza C, Policriti A. From Bisimulation to Simulation: Coarsest Partition Problems. *J. Autom. Reasoning*, 2003. **31**(1):73–103. doi:10.1023/A:1027328830731.

[9] Gentilini R, Piazza C, Policriti A. Rank and simulation: the well-founded case. *J. Log. Comput.*, 2015. **25**(6):1331–1349. doi:10.1093/logcom/ext066.

[10] Omodeo EG, Policriti A, Tomescu AI. On Sets and Graphs: Perspectives on Logic and Combinatorics. Springer, 2017. ISBN 978-3-319-54981-1. URL `https://link.springer.com/book/10.1007%2F978-3-319-54981-1`.

[11] Hopcroft JE. An N Log N Algorithm for Minimizing States in a Finite Automaton. Technical report, Stanford, CA, USA, 1971.

[12] Kanellakis PC, Smolka SA. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Inf. Comput.*, 1990. **86**(1):43–68. doi:10.1016/0890-5401(90)90025-D.

[13] Saha D. An Incremental Bisimulation Algorithm. In: FSTTCS. 2007. doi:10.1007/978-3-540-77050-3_17.

[14] Krug S. Don'T Make Me Think!: A Common Sense Approach to Web Usability. Que Corp., Indianapolis, IN, USA, 1st edition, 2000. ISBN: 0789723107.

[15] Nielsen J. Designing Web Usability: The Practice of Simplicity. New Riders Publishing, Thousand Oaks, CA, USA, 1999. ISBN: 156205810X.

[16] David E, Jon K. Networks, Crowds, and Markets: Reasoning About a Highly Connected World. Cambridge University Press, New York, NY, USA, 2010. ISBN:0521195330, 9780521195331.