CHARACTERIZING TRACES OF PROCESSES DEFINED BY PRECEDENCE AND RESPONSE CONSTRAINTS: AN ORDER THEORY APPROACH

MARK DUKES AND ANTON SOHN

Abstract. In this paper we consider a general system of activities that can, but do not have to, occur. This system is governed by a set of two types of constraints: precedence and response. A precedence constraint dictates that an activity can only occur if it has been preceded by some other specified activity. Response constraints are similarly defined. An execution of the system is a listing of activities in the order they occur and which satisfies all constraints. Such systems naturally arise in areas of theoretical computer science and decision science. An outcome of the freedom with which activities can occur is that there are many different possible executions, and gaining a combinatorial insight into these is a non-trivial problem.

We characterize all of the ways in which such a system can be executed. Our approach uses order theory to provide a classification in terms of the linear extensions of posets constructed from the constraint sets. This characterization is essential in calculating the stakeholder utility metrics that have been developed by the first author that allow for quantitative comparisons of such systems/processes. It also allows for a better understanding of the theoretical backbone to these processes and their deconstruction as a shuffle product of smaller systems.

1. Introduction

The notion of an activity requiring a previous activity to have first occurred is an innate concept in discrete modelling. So too is the notion of an activity occurring as a response to some activity having occurred. Examples are plentiful and include the concurrency one finds in asynchronous systems [2], the precedence diagram method [9] in the area of scheduling/project planning, the precedence graph for testing conflict serializability in database management systems [7, Chpt. 17], and finite-state verification in software systems [6, 1].

Declarative processes [15, 11] have their origins in the area of business process modelling and were introduced as an alternative to the traditional imperative approaches [10]. The declarative paradigm consists of detailing, through a set of constraints on activities, how activities may occur with respect to one another. A declarative process can then happen in any possible way so long as it does not violate any of the specified constraints.

While declarative processes allow for extremely abstract and specialized constraints, the majority of those that appear in a modelling context are a small collection. One such example is the *precedence constraint* that specifies if some activity b were to occur, then another activity a must have preceded it. Another is the *response constraint* the specifies should an activity a occur, then activity b must occur as a response.

These natural constraints can appear in a wide variety of scenarios and are not restricted to the technically oriented examples mentioned above. Recently the first author introduced two utility metrics for declarative processes [5, 4]. These metrics were the proven solutions to a collection of stipulated axioms. The purpose of this was to allow for quantitative comparisons of such processes against one another with respect to prescribed notions of stakeholder satisfaction [4]. It also allowed for the comparison of declarative processes from the viewpoints of multiple stakeholders. Examples of this comparative process analysis in action include a

2020 *Mathematics Subject Classification.* 06A07, 68R05.

Key words and phrases. Order theory; Linear extension; Precedence constraint; Response constraint; Declarative process.

hospital emergency department admission processes [4] and aspects of the Federal Disaster Assistance Policy [3]. When it comes to modelling human-designed processes, the most common constraints are precedence and response constraints.

The ways in which a declarative process may be executed are called the *traces* of the process. A necessary object in the calculation of the utility metrics is a listing of all traces [4]. Determining this listing is a challenging enumeration problem since certain collections of constraints can create subtle trace inter-dependencies that are non-trivial to analyze. Such a characterization is used to calculate the total number of traces along with the refined number of traces that satisfy stakeholder 'satisfaction' constraints.

In this paper we solve the problem of characterizing the traces of declarative processes in which the constraint set contains only precedence and response constraints. These results are readily applicable to those areas of scheduling, asynchronous systems, database management systems, and finite-state verification that were previously mentioned. Our approach is to represent the declarative process constraint set using partially ordered sets. We find that linear extensions of suitably defined posets are at the heart of the trace characterization problem. The previous statement is a somewhat simplified account of our characterizations but represents the essence of the solutions. Partially ordered sets already have many applications in the sciences [8]. This paper enriches that application area by providing a new and unusual use of posets.

In Section 2 we define declarative processes and some of the necessary concepts. In Section 3 we prove a decomposition of declarative processes into smaller 'connected' declarative processes that uses the notion of the shuffle product. In Section 4 we consider declarative processes whose constraint set consists of only *successor* constraints. The main result of that section is Theorem 12 which characterizes the traces in terms of linear extensions of a poset constructed from the constraint set. In Section 5 we consider the more general case of constraint sets that contain precedence and response constraints. We show that the derivation of the trace set is far more involved, but is achievable, and the main result of this section is Theorem 25.

In Section 6 we consider two special cases of the declarative processes studied in Section 5. These special cases occur when constraints are either all precedence constrains, or all response constraints. We are able to leverage properties of the relations induced by the constraint sets to give simpler expressions for the set of all traces for both of these, and these are summarized in Theorems 33 and 37. Finally in Section 7, we consider the computation of some of the sets that appear in the the statements of the theorems and outline efficient methods for their calculation based both on our results and known algorithms in the literature.

2. Preliminaries

A declarative process D is a pair (Σ, Const) where Σ is a set of n activities and Const is a listing of constraints detailing existential and temporal dependencies between activities. These processes were recently defined and studied in a series of papers [5, 4, 3]. In this paper we will focus on two constraints: the precedence constraint prec, and the response constraint resp. The constraint $\mathsf{resp}(a, b)$ indicates that if a occurs then b must occur afterwards as a response to it. Similarly $\mathsf{prec}(a, b)$ indicates that if b occurs then a must have occurred before it.

There is a third constraint succ, termed the *successor constraint*, that represents the logical conjunction of prec and resp:

$$\mathsf{succ}(a, b) \equiv \mathsf{resp}(a, b) \wedge \mathsf{prec}(a, b).$$

Given activities a and b, the constraint $\mathsf{succ}(a, b)$ indicates that a occurs if and only if b occurs, and that a occurs before b. Constraint sets consisting of successor constraints can be seen to be a special type of the more general predecessor and response constraint sets.

A trace of a declarative process $D = (\Sigma, \mathsf{Const})$ is a sequence of elements of Σ that satisfies all constraints in Const. We will represent sequences of elements of a set A as words over the alphabet A in order to avoid confusion between sequences and ordered tuples. For example, the sequence (a, b, c, d) will be represented as the word $abcd$. Note that the empty sequence,

denoted by ϵ, may be a trace of D. In the research so far on this topic, and also in this paper, we will be interested in first-passage traces, which are those traces in which each activity $a \in \Sigma$ occurs at most once, so from here onward we say trace to mean first-passage trace. We denote the set of traces of D by $\mathsf{Traces}(D)$. Throughout this paper we make the reasonable assumption that Σ is always nonempty. Moreover, we will assume that if a constraint between activities a and b, such as $\mathsf{prec}(a,b)$, is in Const, then a and b are distinct elements of Σ.

Example 1. Consider $D = (\Sigma, \mathsf{Const})$ with $\Sigma = \{a,b,c\}$ and $\mathsf{Const} = \{\mathsf{resp}(c,a), \mathsf{prec}(b,a)\}$. Then

$$\mathsf{Traces}(D) = \{\epsilon, b, ba, cba, bca\}.$$

Given a set Σ, let $\mathsf{SubPerms}(\Sigma)$ be the set of all permutations of all subsets of the set Σ. For example,

$$\mathsf{SubPerms}(\{a,b,c\}) = \{\epsilon, a, b, c, ab, ba, ac, ca, bc, cb, \\ abc, acb, bac, bca, cab, cba\}.$$

One straightforward way of determining the set of all traces of a system is to check whether every sequence $\tau \in \mathsf{SubPerms}(\Sigma)$ satisfies all of the constraints and to include it in $\mathsf{Traces}(D)$ if so. A different approach would be to consider the $|\mathsf{Const}|$ different declarative systems $\{(\Sigma, \mathsf{const})\}_{\mathsf{const} \in \mathsf{Const}}$, generate the traces for each of these systems, and then take the intersection of the traces for the $|\mathsf{Const}|$ systems. While not as brute force as the first method, this is not far in the direction of an improvement. Ideally, we would like a method that takes into account the specific structure of Const and can announce the traces of the system as a function of this structure. This is the motivation behind the direction we take in this paper.

Before beginning, there are several concepts best introduced now. For n a positive integer we denote by $[n]$ the set $\{1, \dots, n\}$. Given a sequence $x = x_1 \dots x_k$ we define the image of x, denoted $\mathrm{im}(x)$, to be the set $\{x_1, \dots, x_k\}$ of terms in that sequence. We define the restriction of a sequence x to the set A, denoted $x|_A$, to be the subsequence of x obtained by deleting from x all those terms which are not in A. Given a binary relation R and a set A, define the restriction of R to A, denoted by $R|_A$, to be the relation $R \cap (A \times A) = \{(a,b) \in R : a, b \in A\}$. We denote by $R^{\oplus}$ the reflexive and transitive closure of R on the set on which it is defined.

2.1. **Shuffle product.** The *shuffle product* of two words is the formal sum of all the ways of interleaving (or riffle shuffling) these two words. For example, $ab \sqcup\!\sqcup xy = abxy + axby + xaby + axyb + xayb + xyab$. The shuffle product is associative and commutative. Naturally if $\pi_1, \dots, \pi_m$ are words, we define $\sqcup\!\sqcup_{i=1}^{m} \pi_i := \pi_1 \sqcup\!\sqcup \cdots \sqcup\!\sqcup \pi_m$.

If S is a formal sum then we denote by $\mathsf{Set}[S]$ the set of all terms in S, ignoring coefficients. For example, $\mathsf{Set}[ab \sqcup\!\sqcup xy] = \{abxy, axby, xaby, axyb, xayb, xyab\}$. Any sequences we will be interleaving in this paper will be disjoint, so we state without proof the following intuitive remark.

Remark 2. Let $\pi_1, \dots, \pi_m$ be sequences such that the $\mathrm{im}(\pi_i)$ are pairwise disjoint. Then $\tau \in \mathsf{Set}[\sqcup\!\sqcup_{i=1}^{m} \pi_i]$ if and only if $\mathrm{im}(\tau)$ consists of exactly all the terms in each π_i (that is, $\mathrm{im}(\tau) = \bigcup_{i=1}^{m} \mathrm{im}(\pi_i)$), and each π_i is a subsequence of τ.

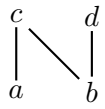
2.2. **Preorders and partial orders.** A *preorder* on a set P is a binary relation $\lesssim$ on P that is reflexive and transitive. We call the ordered pair $(P, \lesssim)$ a *preordered set*. If $\lesssim$ is also antisymmetric then it is a *partial order*, and we call the pair $(P, \lesssim)$ a *partially ordered set* or *poset*. Sometimes we simply refer to the set P as a preordered/partially set.

The *Hasse diagram* of the preorder $\lesssim$ is a transitive reduction of the directed graph $(P, \lesssim)$. The Hasse diagram of a preorder need not be unique, since we can permute the vertices in any cycle to obtain an equally valid Hasse diagram. A preorder is antisymmetric (and hence a partial order) exactly when its Hasse diagram is acyclic. In this case the Hasse diagram is unique, and since it is acyclic we can draw it so that for any $a, b \in P$, a is above b if $a \lesssim b$, thus

eliminating the need for directed edges in the diagram. A subset I of P is called an *down-set* of $(P, \lesssim)$ if for all $a \in P$ and $b \in I$ it satisfies $a \lesssim b \Rightarrow a \in I$. The set of all down-sets of a $(P, \lesssim)$ when ordered by inclusion forms a poset denoted $\mathcal{J}((P, \lesssim))$. An *induced subposet* of a poset $(P, \preceq_P)$ is a poset $(Q, \preceq_Q)$ such that $Q \subseteq P$ and for all $a, b \in Q$, $a \preceq_Q b \iff a \preceq_P b$.

2.3. **Linear extensions.** Let $(P, \preceq)$ be a poset with $P = \{a, \ldots, a_m\}$. A *linear extension* of P is a bijection $\sigma : P \mapsto [m]$ such that $\sigma^{-1}(i) \prec \sigma^{-1}(j) \Rightarrow i < j$. A linear extension of P can be seen as a total ordering of P which respects $\preceq$.

Example 3. For example, let P be the poset on $\{a, b, c, d\}$ with Hasse diagram:



Then $\sigma = (\sigma(a), \sigma(b), \sigma(c), \sigma(d)) = (1, 2, 3, 4)$ is a linear extension of P, whereas $\sigma = (3, 2, 1, 4)$ is not. The set of all linear extensions of P is

$$\{(1, 2, 3, 4), (2, 1, 3, 4), (1, 2, 4, 3), (2, 1, 4, 3), (3, 1, 4, 2)\}.$$

We may identify a linear extension $\sigma : P \to [m]$ with the sequence $\sigma^{-1}(1) \ldots \sigma^{-1}(m)$ of the elements of P. This allows us to write the following expression for the set of all ways that we can totally order the elements of P while respecting $\preceq$.

$$\mathcal{L}(P) = \{\sigma^{-1}(1) \ldots \sigma^{-1}(m) \ : \ \sigma \text{ is a linear extension of } P\}.$$

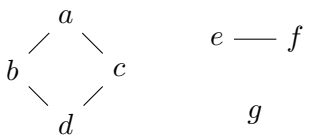
Applying this to the example above, we have

$$\mathcal{L}(P) = \{abcd, bacd, abdc, badc, bdac\}.$$

3. Decomposition of Declarative Processes

Let $D = (\Sigma, \mathsf{Const})$ be a declarative process such that Const can consist of only precedence and response constraints. We refer to such a declarative process as a *precedence-and-response-only* declarative process, and to its constraint set as a *precedence-and-response-only* constraint set. Consider G_D, the simple graph on Σ with edges corresponding to constraints in Const. We call D *connected* if G_D is a connected graph. A partition of G_D into its m connected components induces a partition of D into m declarative processes $D_1 = (\Sigma_1, \mathsf{Const}_1), \ldots, D_m = (\Sigma_m, \mathsf{Const}_m)$, where for each $i \in [m]$, Σ_i is the vertex set of a connected component and Const_i corresponds to the edge set of that connected component. Note that this decomposition is clearly unique up to labeling, and that each D_i is a connected declarative process by design.

Example 4. Let $D = (\{a, b, c, d, e, f, g\}, \{\mathsf{succ}(a, b), \mathsf{prec}(d, c), \mathsf{succ}(b, d), \mathsf{resp}(c, a), \mathsf{resp}(e, f)\})$. Then G_D is the following graph:



This means D can be written as the sum of the three connected declarative processes: $D_1 = (\{a, b, c, d\}, \{\mathsf{succ}(a, b), \mathsf{prec}(d, c), \mathsf{succ}(b, d), \mathsf{resp}(c, a)\})$; $D_2 = (\{e, f\}, \{\mathsf{resp}(e, f)\})$; $D_3 = (\{g\}, \emptyset)$.

Lemma 5. Let D be a precedence-and-response-only declarative process with a decomposition into connected declarative processes $D_1, \ldots, D_m$ as described above. Then

$$\tau \in \mathsf{Traces}(D) \text{ if and only if } \tau \in \mathsf{Set}[\sqcup\!\sqcup_{i=1}^{m} \tau_i] \text{ for some } (\tau_1, \ldots, \tau_m) \in \prod_{i=1}^{m} \mathsf{Traces}(D_i)$$

Proof. Assume $\tau \in \mathsf{Traces}(D)$ and consider $(\tau|_{\Sigma_1}, \ldots, \tau|_{\Sigma_m})$. For all $i \in [m]$ we wish to show that $\tau|_{\Sigma_i} \in \mathsf{Traces}(D_i)$, so let $\mathsf{prec}(a, b), \mathsf{resp}(c, d) \in \mathsf{Const}_i$. Note that $a, b, c, d \in \Sigma_i$ by definition of Σ_i. Suppose that $b \in \mathrm{im}(\tau|_{\Sigma_i})$ since otherwise $\tau|_{\Sigma_i} \vDash \mathsf{prec}(a, b)$ vacuously. Then $b \in \mathrm{im}(\tau)$ since $\tau|_{\Sigma_i}$ is

a subsequence of τ. But $\tau \vDash \mathsf{prec}(a,b)$ so $a \in \mathrm{im}(\tau)$ and hence $a \in \mathrm{im}(\tau|_{\Sigma_i})$. Furthermore b must follow a in τ, so b also follows a in $\tau|_{\Sigma_i}$ since it is a subsequence of τ. Hence $\tau|_{\Sigma_i} \vDash \mathsf{prec}(a,b)$. We can similarly show that $\tau|_{\Sigma_i} \vDash \mathsf{resp}(c,d)$ and so $\tau|_{\Sigma_i} \in \mathsf{Traces}(D_i)$. This implies

$$(\tau|_{\Sigma_1}, \ldots, \tau|_{\Sigma_m}) \in \prod_{i=1}^{m} \mathsf{Traces}(D_i).$$

Next we show that $\tau \in \mathsf{Set}[\sqcup_{i=1}^m \tau|_{\Sigma_i}]$. Clearly $\tau|_{\Sigma_i}$ is a subsequence of τ for all $i \in [m]$. We also have that $\mathrm{im}(\tau) = \mathrm{im}(\tau|_\Sigma) = \cup_{i=1}^m \mathrm{im}(\tau|_{\Sigma_i})$ since $\Sigma = \cup_{i=1}^m \Sigma_i$. By Remark 2, $\tau \in \mathsf{Set}[\sqcup_{i=1}^m \tau|_{\Sigma_i}]$.

Conversely, let $\tau \in \mathsf{Set}[\sqcup_{i=1}^m \tau_i]$ for some $(\tau_1, \ldots, \tau_m) \in \prod_{i=1}^m \mathsf{Traces}(D_i)$.
We wish to show that $\tau \in \mathsf{Traces}(D)$, so let $\mathsf{prec}(a,b), \mathsf{resp}(c,d) \in \mathsf{Const}$. Suppose that $b \in \mathrm{im}(\tau)$ since otherwise $\tau \vDash \mathsf{prec}(a,b)$ vacuously. Now $b \in \mathrm{im}(\tau_i) \subseteq \Sigma_i$ for some $i \in [m]$. Note that $a, b \in \Sigma_i$ and $\mathsf{prec}(a,b) \in \mathsf{Const}_i$ by definition of Σ_i and Const_i. Hence $\tau_i \vDash \mathsf{prec}(a,b)$ so $a \in \mathrm{im}(\tau_i) \subseteq \mathrm{im}(\tau)$. Furthermore b must follow a in τ, so b also follows a in τ_i since it is a subsequence of τ. Therefore $\tau \vDash \mathsf{prec}(a,b)$. We can similarly show that $\tau \vDash \mathsf{resp}(c,d)$ and so $\tau \in \mathsf{Traces}(D)$. □

In other words, $\mathsf{Traces}(D)$ is the set of all possible interleavings of a selection of traces where one trace is chosen from each $\mathsf{Traces}(D_i)$. This decomposition lemma allows for a great simplification of the trace characterization problem when a decomposition into many smaller declarative systems exists. It will prove essential in the following section on successor-only declarative processes.

Example 6. Consider D_1, D_2 and D_3 from Example 4. One can easily verify that $abd \in \mathsf{Traces}(D_1)$, $\epsilon \in \mathsf{Traces}(D_2)$ and $g \in \mathsf{Traces}(D_3)$, so that $(abd, \epsilon, g) \in \prod_{i=1}^3 \mathsf{Traces}(D_i)$. Now the sequence $abgd$ is an element of $\mathsf{Set}[abd \sqcup\!\sqcup \epsilon \sqcup\!\sqcup g]$ (i.e. it is an interleaving of the sequences abd, ϵ and g) so it is also in $\mathsf{Traces}(D)$. Of course this is not the only element of $\mathsf{Traces}(D)$.

4. Successor-only constraint sets

In this section we will consider declarative processes whose constraint set consists of only successor constraints. Since $\mathsf{succ}(a,b)$ can be rewritten as $\mathsf{resp}(a,b) \wedge \mathsf{prec}(a,b)$, such systems are special cases of the more general systems we will consider later. Our reason for considering this special case is that it has a special property which we can leverage. We have chosen to present this before our investigation of the more general constraint set so as to familiarize the reader with the basic concepts and arguments we will use in a less complicated and slightly less formal setting, while also showcasing the type of result we hope to achieve in greater generality. As a consequence, the main result of this section could be derived as a corollary of the results in the following section, but we choose to prove it independently regardless.

Our approach to determining $\mathsf{Traces}(D)$ will involve first determining $\mathsf{PossIm}(D) := \{\mathrm{im}(\tau) : \tau \in \mathsf{Traces}(D)\}$, the collection of all possible sets that occur as the image of some trace of D, and then arranging each of these sets in all possible ways that the constraint set allows. In order to achieve this, we will need a certain relation $\rightsquigarrow$ which satisfies the following intrinsic property:

Property 7. If $a \rightsquigarrow b$ for some $a, b \in \Sigma$, then b must certainly follow a in any trace in which both activities occur.

Definition 8. Let $D = (\Sigma, \mathsf{Const})$ be a precedence-and-response-only declarative process. Then the *order-preserving relation* of D is the relation

$$\rightsquigarrow := \{(a,b) \in \Sigma^2 : \mathsf{prec}(a,b) \in \mathsf{Const} \text{ or } \mathsf{resp}(a,b) \in \mathsf{Const}\}$$

on Σ.

The fact that this relation satisfies Property 7 follows immediately from the definitions of the precedence and response constraints.

For the remainder of this section, consider a declarative process $D = (\Sigma, \mathsf{Const})$ where Const consists only of successor constraints. We refer to such a declarative process as a *successor-only* declarative process and to its constraint set as a *successor-only* constraint set. Lemma 5 allows

us to restrict our attention to the case where D is connected, which turns out to be useful in this instance, so suppose that D is connected.

Remark 9. Since $\mathsf{succ}(a,b) \equiv \mathsf{resp}(a,b) \wedge \mathsf{prec}(a,b)$, the order-preserving relation $\rightsquigarrow$ of a successor-only declarative process is $D = (\Sigma, \mathsf{Const})$ is just the relation $\{(a,b) \in \Sigma^2 : \mathsf{succ}(a,b) \in \mathsf{Const}\}$ on Σ.

Clearly $\rightsquigarrow^{\oplus}$ is a preorder on Σ by construction but is not necessarily a partial order. This can be seen from the following example.

Example 10. Consider the declarative processes $D^{(1)} = (\{a,b,c\}, \{\mathsf{succ}(b,a), \mathsf{succ}(c,a)\})$ and $D^{(2)} = (\{a,b,c\}, \{\mathsf{succ}(a,b), \mathsf{succ}(b,c), \mathsf{succ}(c,a)\})$. The Hasse diagrams of their corresponding preorders, $\rightsquigarrow_1^{\oplus}$ and $\rightsquigarrow_2^{\oplus}$, are shown in Figures 1 and 2. We can see that $\rightsquigarrow_1^{\oplus}$ is a partial order

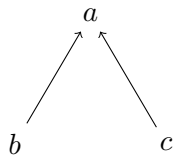


FIGURE 1. $(\{a,b,c\}, \rightsquigarrow_1^{\oplus})$

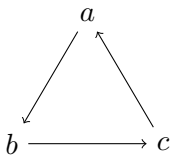


FIGURE 2. $(\{a,b,c\}, \rightsquigarrow_2^{\oplus})$

while $\rightsquigarrow_2^{\oplus}$ is not, since its Hasse diagram contains a directed cycle.

Let us now consider $\mathsf{PossIm}(D)$. Since, by assumption, D is connected and every constraint is a successor constraint, the occurrence of any activity in a trace implies the occurrence of every other activity in that trace. This means that the image of every trace is either $\emptyset$ or Σ. However, notice that if $\rightsquigarrow^{\oplus}$ is not antisymmetric then there is a directed cycle in the Hasse diagram of $(\Sigma, \rightsquigarrow^{\oplus})$. By Property 7, an activity in that cycle must be followed by all activities in the cycle, including itself. Thus a trace with image Σ will feature repeated activities, which is a contradiction since we are deadling with first-passage traces in which an activity can occur at most once. From the above discussion it follows that if $\rightsquigarrow^{\oplus}$ is not antisymmetric, then $\mathsf{PossIm}(D) = \{\emptyset\}$. Otherwise if $\rightsquigarrow^{\oplus}$ is antisymmetric, then it is not difficult to see that there will exist some trace with image Σ (namely a linear extension of the poset $(\Sigma, \rightsquigarrow^{\oplus})$) and so $\mathsf{PossIm}(D) = \{\emptyset, \Sigma\}$.

Example 11. Consider again the declarative processes $D^{(1)}$ and $D^{(2)}$ from Example 10. If a occurs in a trace of $D^{(1)}$ then the constraint set implies that the other two activities must also occur. The same is true for b and c. Hence we have that $\mathsf{PossIm}(D^{(1)}) = \{\emptyset, \{a,b,c\}\}$. If a occurs in a trace of $D^{(2)}$ then the constraint set implies that it must be succeeded by b, which in turn must be succeeded by c, which in turn must be succeeded by a, meaning that a occurs twice; a contradiction. The same is true for b and c. Hence $\mathsf{PossIm}(D^{(2)}) = \{\emptyset\}$.

Now that we know what images a trace can have, we wish to arrange the elements of each possible image I in all possible ways that the constraint set allows to form traces. By construction of $\rightsquigarrow$, this turns out to be equivalent to generating the set $\mathcal{L}((I, \preceq_I))$ where $(\preceq_I) := (\rightsquigarrow|_I)^{\oplus}$ as we will show now for the successor-only case.

Theorem 12. Let $D = (\Sigma, \mathsf{Const})$ be a successor-only declarative process. Recall the description of $\rightsquigarrow$ as given in Remark 9 and that $\rightsquigarrow^{\oplus}$ is its reflexive and transitive closure.

(i) In the event that D is connected,
  (a) If $\rightsquigarrow^{\oplus}$ is not antisymmetric then $\mathsf{Traces}(D) = \{\epsilon\}$.
  (b) If $\rightsquigarrow^{\oplus}$ is antisymmetric then $\mathsf{Traces}(D) = \{\epsilon\} \cup \mathcal{L}((\Sigma, \rightsquigarrow^{\oplus}))$.

(ii) In the event that D is not connected, let $D_1, \ldots, D_m$ be its decomposition into connected declarative processes as described in Section 3. Then

$$\mathsf{Traces}(D) = \bigcup_{(\tau_1,\ldots,\tau_m) \in \prod_{i=1}^{m} \mathsf{Traces}(D_i)} \mathsf{Set}\left[\sqcup\!\sqcup_{i=1}^{m} \tau_i\right],$$

where each $\mathsf{Traces}(D_i)$ can be found using part (i) of this proposition.

Proof. (i) Let $n = |\Sigma|$.

(a) In this case $\mathsf{PossIm}(D) = \{\emptyset\}$, as previously discussed. The sequence ϵ vacuously satisfies any successor constraint, so it must be in $\mathsf{Traces}(D)$, and the result follows since the image of no other sequence is $\emptyset$.

(b) In this case $\mathsf{PossIm}(D) = \{\emptyset, \Sigma\}$, as previously discussed.

$\subseteq$: Let $\tau = a_1 \dots a_n \in \mathsf{Traces}(D)$. If $\mathrm{im}(\tau) = \emptyset$ then $\tau = \epsilon$ and we are done so suppose instead that $\mathrm{im}(\tau) = \Sigma$. Let $\sigma : \Sigma \to [n]$ be the bijection defined by $a_i \mapsto i$, so that $\tau = \sigma^{-1}(1) \dots \sigma^{-1}(n)$. We wish to show that σ is a linear extension of $(\Sigma, \rightsquigarrow^{\oplus})$. Let i_1, i_2 be distinct elements of $[n]$ such that $\sigma^{-1}(i_1) = a_{i_1} \rightsquigarrow^{\oplus} a_{i_2} = \sigma^{-1}(i_2)$. Since $\rightsquigarrow^{\oplus}$ satisfies Property 7, a_{i_2} must follow a_{i_1} in τ. That is, $i_1 < i_2$. Hence σ is a linear extension of $(\Sigma, \rightsquigarrow^{\oplus})$ and so $\tau \in \mathcal{L}((\Sigma, \rightsquigarrow^{\oplus}))$.

$\supseteq$: As before, ϵ must be an element of $\mathsf{Traces}(D)$. Let $\tau = a_1 \dots a_n \in \mathcal{L}((\Sigma, \rightsquigarrow^{\oplus}))$. Then $\tau = \sigma^{-1}(1) \dots \sigma^{-1}(n)$ for some linear extension σ of $(\Sigma, \rightsquigarrow^{\oplus})$. We wish to show that $\tau \in \mathsf{Traces}(D)$ so let $\mathsf{succ}(a, b) \in \mathsf{Const}$. Now $\mathrm{im}(\tau) = \Sigma$ so both a and b must appear in τ. This implies $a = \sigma^{-1}(i)$ and $b = \sigma^{-1}(j)$ for some distinct $i, j \in [n]$. From the definition of $\rightsquigarrow$, we have that $a \rightsquigarrow b$ so $a \rightsquigarrow^{\oplus} b$ and finally $\sigma^{-1}(i) \rightsquigarrow^{\oplus} \sigma^{-1}(j)$. Since σ is a linear extension of $(\Sigma, \rightsquigarrow^{\oplus})$ this shows $i < j$, meaning that b follows a in τ. Therefore $\tau \vDash \mathsf{succ}(a, b)$ and so $\tau \in \mathsf{Traces}(D)$.

Part (ii) is a restatement of Lemma 5. □

Example 13. Applying Proposition 12(i) to the declarative processes from Example 10, we find that $\mathsf{Traces}(D^{(1)}) = \{\epsilon\} \cup \mathcal{L}((\{a, b, c\}, \rightsquigarrow_1^{\oplus})) = \{\epsilon, bca, cba\}$ and $\mathsf{Traces}(D^{(2)}) = \{\epsilon\}$.

Example 14. Consider the declarative process $D = (\{a, b, c, d\}, \{\mathsf{succ}(a, b), \mathsf{succ}(c, d)\})$. The decomposition of D as described in Section 3 gives us the two connected declarative processes

$$D_1 = (\{a, b\}, \{\mathsf{succ}(a, b)\}) \text{ and } D_2 = (\{c, d\}, \{\mathsf{succ}(c, d)\}).$$

We use Theorem 12 to find $\mathsf{Traces}(D_1) = \{\epsilon, ab\}$ and $\mathsf{Traces}(D_2) = \{\epsilon, cd\}$. Now we compute $\sqcup\!\sqcup_{i=1}^{2} \tau_i = \tau_1 \sqcup\!\sqcup \tau_2$ for each

$$(\tau_1, \tau_2) \in \prod_{i=1}^{2} \mathsf{Traces}(D_i) = \mathsf{Traces}(D_1) \times \mathsf{Traces}(D_2) = \{(\epsilon, \epsilon), (\epsilon, cd), (ab, \epsilon), (ab, cd)\}.$$

The shuffle products are:

$$\begin{aligned} \epsilon \sqcup\!\sqcup \epsilon &= \epsilon \\ \epsilon \sqcup\!\sqcup cd &= cd \\ ab \sqcup\!\sqcup \epsilon &= ab \\ ab \sqcup\!\sqcup cd &= abcd + acbd + cabd + cadb + cdab. \end{aligned}$$

Applying Lemma 5 we find that

$$\mathsf{Traces}(D) = \{\epsilon, ab, cd, abcd, acbd, cabd, cadb, cdab\}.$$

5. Precedence-and-response-only constraint sets

In this section we investigate the most general case of this paper. Let $D = (\Sigma, \mathsf{Const})$ be a precedence-and-response-only declarative process. Our main result is a description for $\mathsf{Traces}(D)$ in terms of a order theoretic structures related to the interdependencies stated in the constraint set. The approach used will be the same as in Section 4: we first determine $\mathsf{PossIm}(D)$ and then arrange each of its sets in all possible ways that the constraint set allows.

Determining $\mathsf{PossIm}(D)$ is not as straightforward as it was in the successor-only case. The difficulty arises because if D is connected we cannot make the same argument to show that the image of every trace is either $\emptyset$ or Σ. In order to overcome this, we need to define a certain relation which satisfies the following property:

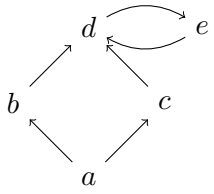


FIGURE 3. Hasse diagram for the preordered set $(\Sigma, \lesssim)$ from Example 17

Property 15. If $a \lesssim b$ for some $a, b \in \Sigma$, then in any trace in which b occurs, a must also occur.

Definition 16. Let $D = (\Sigma, \mathsf{Const})$ be a precedence-and-response-only declarative process. Then the *implied-occurrence relation* of D is the relation

$$\lesssim \; := \; \{(a, b) \in \Sigma^2 : \mathsf{prec}(a, b) \in \mathsf{Const} \text{ or } \mathsf{resp}(b, a) \in \mathsf{Const}\}^{\oplus}$$

on Σ.

It can be easily verified that this relation does in fact satisfy Property 15 using the definitions of the precedence and response constraints.

Example 17. Consider the declarative process $D = (\Sigma, \mathsf{Const})$ where $\Sigma = \{a, b, c, d, e\}$ and $\mathsf{Const} = \{\mathsf{resp}(b, a), \mathsf{resp}(c, a), \mathsf{resp}(d, e), \mathsf{resp}(e, c), \mathsf{prec}(a, d), \mathsf{prec}(b, d), \mathsf{prec}(d, e)\}$. The Hasse diagram of the preordered set $(\Sigma, \lesssim)$ is given in Figure 3. The down-sets of $(\Sigma, \lesssim)$ can be seen to be $\emptyset, \{a\}, \{a, b\}, \{a, c\}, \{a, b, c\}$ and Σ. We also have $\rightsquigarrow = \{(b, a), (c, a), (d, e), (e, c), (a, d), (b, d)\}$.

Lemma 18. If $\tau \in \mathsf{Traces}(D)$ then $\mathrm{im}(\tau)$ is a down-set of $(\Sigma, \lesssim)$.

Proof. Let $a, b \in \Sigma$ be such that $b \in \mathrm{im}(\tau)$ and $a \lesssim b$. Then $a \in \mathrm{im}(\tau)$ since $\lesssim$ satisfies Property 15. □

The converse of the above lemma does not hold, that is, a down-set of $(\Sigma, \lesssim)$ need not be the image of some trace. The following construction will allow us to characterize the down-sets which *are* the image of some trace. This will then be used to determine $\mathsf{Traces}(D)$.

Definition 19. Let $D = (\Sigma, \mathsf{Const})$ be a precedence-and-response-only declarative process. Recall that $\rightsquigarrow$, the order-preserving relation of D, is the relation $\{(a, b) \in \Sigma^2 : \mathsf{prec}(a, b) \in \mathsf{Const} \text{ or } \mathsf{resp}(a, b) \in \mathsf{Const}\}$ on Σ, and that $\lesssim$, the implied-occurence relation of D, is the relation $\{(a, b) \in \Sigma^2 : \mathsf{prec}(a, b) \in \mathsf{Const} \text{ or } \mathsf{resp}(b, a) \in \mathsf{Const}\}^{\oplus}$ on Σ. Now if I is a down-set of $(\Sigma, \lesssim)$ then define $\preceq_I$ to be the relation $(\rightsquigarrow |_I)^{\oplus}$ on I, where $(\rightsquigarrow |_I)^{\oplus}$ is the reflexive and transitive closure of the relation $(\rightsquigarrow |_I)$.

Naturally, we write $a \prec_I b$ when $a \preceq_I b$ and $a \neq b$.

Lemma 20. Let $\tau \in \mathsf{Traces}(D)$ and let $I = \mathrm{im}(\tau)$. If $a_0, a_m \in I$ are such that $a_0 \prec_I a_m$, then a_m follows a_0 in τ.

Proof. By definition of $\preceq_I$ we have that $a_0 \rightsquigarrow a_1 \rightsquigarrow \ldots \rightsquigarrow a_m$ for some $a_1, \ldots, a_{m-1} \in I$. Since $\rightsquigarrow$ satisfies Property 7, we have that a_i follows a_{i-1} in τ for all $i \in [m]$. Hence a_m must follow a_0 in τ. □

So, by construction, $\preceq_I$ describes the ways in which we can arrange the elements of I to form a trace. In the definition of $\preceq_I$ we restrict to I *before* taking the transitive closure. The following example illustrates why this is necessary.

Example 21. Consider the declarative process $D = (\{a, b, c\}, \{\mathsf{prec}(a, b), \mathsf{resp}(b, c)\})$. We have that $\lesssim = \{(a, a), (b, b), (c, c), (a, b), (c, b)\}$ and $\rightsquigarrow = \{(a, b), (b, c)\}$. Recall that $\rightsquigarrow$ is constructed to have the property that if $a \rightsquigarrow b$ for some $a, b \in \Sigma$, then b must certainly follow a in any trace in which both activities occur. It would seem reasonable to assume that $\rightsquigarrow^{\oplus}$ would still respect this property, since if c follows b which in turn follows a, then c would have to follow a,

but there is an error in this reasoning. In our example, $\rightsquigarrow^{\oplus}$ contains (a, c), and $I = \{a, c\}$ is a down-set of $(\Sigma, \lesssim)$ and so is a contender for the image of a trace. If we restrict $\rightsquigarrow^{\oplus}$ to I, it still contains (a, c), which means that the sequence ca cannot be in $\mathsf{Traces}(D)$. Again this seems reasonable, but by examining the constraint set, note that if b does not occur in a trace then there is no restriction on the order in which a and c must occur. This shows that ca is, in fact, in $\mathsf{Traces}(D)$.

Before classifying $\mathsf{PossIm}(D)$ and subsequently $\mathsf{Traces}(D)$ we will need the following lemma.

Lemma 22. Let I be a down-set of $(\Sigma, \lesssim)$ such that $\preceq_I$ is antisymmetric. Then $\mathcal{L}((I, \preceq_I)) \subseteq \mathsf{Traces}(D)$.

Proof. First note that $(I, \preceq_I)$ is a poset since $\preceq_I$ is already reflexive and transitive by construction. Let $m = |I|$ and let $\tau \in \mathcal{L}((I, \preceq_I))$. Then $\mathrm{im}(\tau) = I$ and $\tau = \sigma^{-1}(1) \ldots \sigma^{-1}(m)$ for some linear extension σ of $(I, \preceq_I)$. If $m = 0$ then σ is the empty function and $\tau = \epsilon \in \mathsf{Traces}(D)$. Suppose now that $m \neq 0$. We wish to show that $\tau \in \mathsf{Traces}(D)$ so let $\mathsf{prec}(a, b), \mathsf{resp}(c, d) \in \mathsf{Const}$.

Suppose that $b \in \mathrm{im}(\tau)$ since otherwise $\tau \vDash \mathsf{prec}(a, b)$ vacuously. From the definition of $\lesssim$, we have that $a \lesssim b$ which implies that $a \in I = \mathrm{im}(\tau)$ since I is a down-set of $(\Sigma, \lesssim)$. Now $a = \sigma^{-1}(i)$ and $b = \sigma^{-1}(j)$ for some distinct $i, j \in [m]$. From the definition of $\rightsquigarrow$, we have that $a \rightsquigarrow b$ so $a \rightsquigarrow |_I b$ and finally $a \prec_I b$. Hence $\sigma^{-1}(i) \prec_I \sigma^{-1}(j)$ which implies that $i < j$ since σ is a linear extension of $(I, \preceq_I)$. Thus b follows a in τ so $\tau \vDash \mathsf{prec}(a, b)$. By exactly the same reasoning we can show that $\tau \vDash \mathsf{resp}(c, d)$. Therefore $\tau \in \mathsf{Traces}(D)$. □

We may now state and prove the classification of $\mathsf{PossIm}(D)$.

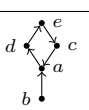
Proposition 23. If D is a precedence-and-response-only declarative process, then $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ for which $\preceq_I$ is antisymmetric.

Proof. We prove that $I \in \mathsf{PossIm}(D)$ if and only if I is a down-set of $(\Sigma, \lesssim)$ and $\preceq_I$ is antisymmetric. Let $I \in \mathsf{PossIm}(D)$. Then $I = \mathrm{im}(\tau)$ for some $\tau \in \mathsf{Traces}(D)$. Hence I is a down-set of $(\Sigma, \lesssim)$ by Lemma 20. Suppose for a contradiction that $\preceq_I$ is not antisymmetric, and let $a, b \in I$ be such that $a \prec_I b$ and $b \prec_I a$. By Lemma 20, a must follow b which in turn must follow a in τ. So a appears more than once in τ, a contradiction. Therefore I is a down-set of $(\Sigma, \lesssim)$ and $\preceq_I$ is antisymmetric.

Conversely, let I be a down-set of $(\Sigma, \lesssim)$ such that $\preceq_I$ is antisymmetric. Since $\preceq_I$ is antisymmetric the pair $(I, \preceq_I)$ is a poset. Let $\tau \in \mathcal{L}((I, \preceq_I))$. By Lemma 22 we have $\tau \in \mathsf{Traces}(D)$. But $\mathrm{im}(\tau) = I$ so $I \in \mathsf{PossIm}(D)$. □

An efficient method for the generation of $\mathsf{PossIm}(D)$ along with all of the relations $\preceq_I$s will be described in Section 7.

Example 24. Consider again the declarative process D from Example 17. The relations $\preceq_I$ corresponding to each of the down-sets I of $(\Sigma, \lesssim)$, as well as their Hasse Diagrams, are given in the table below.

k	I_k	$\preceq_{I_k}$	Hasse diagram
1	$\emptyset$	$\emptyset$	
2	$\{a\}$	$\{(a, a)\}$	
3	$\{a, b\}$	$\{(a, a), (b, b), (b, a)\}$	
4	$\{a, c\}$	$\{(a, a), (c, c), (c, a)\}$	
5	$\{a, b, c\}$	$\{(a, a), (b, b), (c, c), (b, a), (c, a)\}$	
6	Σ	$\{(a, a), (b, b), (c, c), (d, d), (e, e), (b, a), (c, a), (d, e), (e, c), (a, d), (b, d), (a, e), (a, c), (d, c), (d, a), (e, a), (e, d), (c, d), (c, e), (b, e), (b, c)\}$	

From the Hasse diagrams in this table we can see that only $\preceq_\Sigma$ is not antisymmetric, and so we use Proposition 23 to find that

$$\mathsf{PossIm}(D) = \{I_1, I_2, I_3, I_4, I_5\} = \{\emptyset, \{a\}, \{a,b\}, \{a,c\}, \{a,b,c\}\}.$$

Now if $I \in \mathsf{PossIm}(D)$, then it is the image of at least one trace of D, and the order in which activities can occur in these traces is described by $\preceq_I$. This means that arranging I in all possible ways that the constraint set allows turns out to be equivalent to generating the set $\mathcal{L}((I, \preceq_I))$. This leads to the following classification of $\mathsf{Traces}(D)$.

Theorem 25. Recall the definitions of $\preceq_I$ and $\lesssim$ as given in Definition 19. If D is a precedence-and-response-only declarative process, then

$$\mathsf{Traces}(D) = \bigcup_{I \in \mathsf{PossIm}(D)} \mathcal{L}((I, \preceq_I)),$$

where $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ such that $\preceq_I$ is antisymmetric.

Proof. The fact that $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ such that $\preceq_I$ is antisymmetric is proven in Proposition 23.

We will first show that $\mathsf{Traces}(D) \subseteq \bigcup_{I \in \mathsf{PossIm}(D)} \mathcal{L}((I, \preceq_I))$. Let $\tau = a_1 \dots a_m \in \mathsf{Traces}(D)$. By Lemma 20, $\mathrm{im}(\tau) = I$ for some down-set I of $(\Sigma, \lesssim)$. The set $I \in \mathsf{PossIm}(D)$ so $\preceq_I$ is antisymmetric and hence $(I, \preceq_I)$ is a poset. Let $\sigma : I \to [m]$ be the bijection defined by $a_i \mapsto i$ so that $\tau = \sigma^{-1}(1) \dots \sigma^{-1}(m)$. We wish to show that σ is a linear extension of $(I, \preceq_I)$. Let $i_1, i_2 \in [m]$ be such that $\sigma^{-1}(i_1) = a_{i_1} \prec_I a_{i_2} = \sigma^{-1}(i_2)$. By Lemma 20, a_{i_2} must follow a_{i_1} in τ. That is, $i_1 < i_2$. Hence σ is a linear extension of $(I, \preceq_I)$, so $\tau \in \mathcal{L}((I, \preceq_I))$.

Conversely, to show $\bigcup_{I \in \mathsf{PossIm}(D)} \mathcal{L}((I, \preceq_I)) \subseteq \mathsf{Traces}(D)$, let $I \in \mathsf{PossIm}(D)$. Then $\preceq_I$ is antisymmetric, so $\mathcal{L}((I, \preceq_I)) \subseteq \mathsf{Traces}(D)$ by Lemma 22. □

To generate each $\mathcal{L}((I, \preceq_I))$ in practice we may apply known algorithms such as that of Pruesse and Ruskey [12]. This algorithm, when given a poset P, generates the set $\mathcal{L}(P)$ in time $\mathcal{O}(e(P))$ where $e(P)$ is the number of linear extensions of P.

Example 26. Applying Theorem 25 to the declarative process D from Example 17, we find that

$$\mathsf{Traces}(D) = \bigcup_{k=1}^{5} \mathcal{L}((I_k, \preceq_{I_k})) = \{\epsilon, a, ba, ca, bca, cba\}.$$

Remark 27. Note that we do not require D to be connected in Theorem 25. However, we may still decompose D if it is not connected and apply the theorem to each declarative process in the decomposition if this turns out to be more computationally efficient.

6. Special Cases

In this section we will consider declarative processes that either consist only of precedence constraints, or only of response constraints.

6.1. Precedence-only constraint sets.

Consider a declarative process $D = (\Sigma, \mathsf{Const})$ where Const consists only of precedence constraints. We refer to such a declarative process as a *precedence-only* declarative process, and to its constraint set as a *precedence-only* constraint set. In this case $\rightsquigarrow$ is the relation $\{(a,b) \in \Sigma^2 : \mathsf{prec}(a,b) \in \mathsf{Const}\}$ on Σ and $\lesssim$ is the relation $\{(a,b) \in \Sigma^2 : \mathsf{prec}(a,b) \in \mathsf{Const}\}^\oplus$ on Σ. Notice that $\lesssim$ is the same as $\rightsquigarrow^\oplus$, a fact that we will leverage in order to optimize our classification of $\mathsf{Traces}(D)$.

Example 28. Consider the declarative process $D = (\Sigma, \mathsf{Const})$ where $\Sigma = \{a,b,c,d,e,f\}$ and $\mathsf{Const} = \{\mathsf{prec}(a,c), \mathsf{prec}(b,c), \mathsf{prec}(c,d), \mathsf{prec}(d,e), \mathsf{prec}(e,d), \mathsf{prec}(d,f)\}$. The Hasse diagram of the corresponding preordered set $(\Sigma, \lesssim)$ is given in Figure 4. The down-sets of $(\Sigma, \lesssim)$ can be seen to be $\emptyset, \{a\}, \{b\}, \{a,b\}, \{a,b,c\}, \{a,b,c,d,e\}$, and Σ.

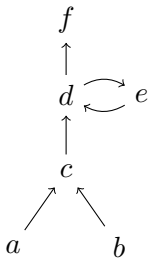


FIGURE 4. $(\Sigma, \lesssim)$

As before, for each down-set I of $(\Sigma, \lesssim)$ we define $\preceq_I$ to be the relation $(\rightsquigarrow|_I)^\oplus$ on I. The following lemma demonstrates that, unlike in the precedence-and-response-only case, we may equivalently take the transitive closure before restricting to I in the definition of $\preceq_I$.

Lemma 29. Let $D = (\Sigma, \mathsf{Const})$ be a precedence-only declarative process. If I is a down-set of $(\Sigma, \lesssim)$, then $(\preceq_I) := (\rightsquigarrow|_I)^\oplus = (\rightsquigarrow^\oplus)|_I$.

Proof. Suppose $(a, b) \in (\rightsquigarrow|_I)^\oplus$. We note that $a, b \in I$. Since (a, b) is in the transitive closure there must exist $a_1, \ldots a_m \in I$ such that $a(\rightsquigarrow|_I)a_1(\rightsquigarrow|_I)\ldots(\rightsquigarrow|_I)a_m(\rightsquigarrow|_I)b$ which implies $a \rightsquigarrow a_1 \rightsquigarrow \ldots \rightsquigarrow a_m \rightsquigarrow b$. This means $a \rightsquigarrow^\oplus b$ and so $a(\rightsquigarrow^\oplus)|_I b$.

Conversely, suppose $(a, b) \in (\rightsquigarrow^\oplus)|_I$. Again, we note that this means $a, b \in I$. As $a \rightsquigarrow^\oplus b$ there must exist $a_1, \ldots a_m \in \Sigma$ such that $a \rightsquigarrow a_1 \rightsquigarrow \ldots \rightsquigarrow a_m \rightsquigarrow b$. This implies $a_i \rightsquigarrow^\oplus b$ for all $i \in [m]$, which in turn implies $a_i \lesssim b$ for all $i \in [m]$. Now since I is a down-set that contains b and $a_i \lesssim b$ for all $i \in [m]$, we have $a_i \in I$ again for all such i. This allows us to write

$$a(\rightsquigarrow|_I)a_1(\rightsquigarrow|_I)\ldots(\rightsquigarrow|_I)a_m(\rightsquigarrow|_I)b,$$

i.e. $a(\rightsquigarrow|_I)^\oplus b$. □

We can use this lemma to prove the following proposition concerning a 'maximal' element of $\mathsf{PossIm}(D)$, of which all other elements of $\mathsf{PossIm}(D)$ are not only subsets, but induced subposets.

Proposition 30. Let $D = (\Sigma, \mathsf{Const})$ be a precedence-only declarative process and define M to be the set $\Sigma \backslash \{a \in \Sigma : b \lesssim c \lesssim b \lesssim a$ for some distinct $b, c \in \Sigma\}$. Then the following hold.

(i) $\mathsf{PossIm}(D)$ is the set of down-sets of $(\Sigma, \lesssim)$ which are also subsets of M.
(ii) $M \in \mathsf{PossIm}(D)$.
(iii) If $I \in \mathsf{PossIm}(D)$ then $(I, \preceq_I)$ is an induced subposet of $(M, \preceq_M)$.

Proof.

(i) Let $I \in \mathsf{PossIm}(D)$. We have that I is a down-set of $(\Sigma, \lesssim)$ by Proposition 23. Let $a \in I$ and suppose for a contradiction that $a \notin M$. Then there exist distinct $b, c \in \Sigma$ such that $b \lesssim c \lesssim b \lesssim a$. Now $b, c \in I$ since I is a down-set of $(\Sigma, \lesssim)$. Recall that $\lesssim$ and $\rightsquigarrow^\oplus$ are the same relation and that $(\preceq_I) = (\rightsquigarrow|_I)^\oplus = (\rightsquigarrow^\oplus)|_I$ so $b \preceq_I c \preceq_I b$. Hence $\preceq_I$ is not antisymmetric, contradicting Proposition 23. Therefore $I \subseteq M$.

Let I be a down-set of $(\Sigma, \lesssim)$ such that $I \subseteq M$. Suppose for a contradiction that $I \notin \mathsf{PossIm}(D)$. By Proposition 23, $\preceq_I$ must not be antisymmetric, so there exist distinct $b, c \in I$ such that $b \preceq_I c \preceq_I b$. Again recall that $\lesssim$ and $\rightsquigarrow^\oplus$ are the same relation and that $(\preceq_I) = (\rightsquigarrow|_I)^\oplus = (\rightsquigarrow^\oplus)|_I$ so $b \lesssim c \lesssim b$. Notice that $b \in M$ and $b \lesssim c \lesssim b \lesssim b$, contradicting the definition of M.

(ii) By part (i) of this proposition we need only show that M is a down-set of $(\Sigma, \lesssim)$, so let $a \in \Sigma$ and $x \in M$ such that $a \lesssim x$. Suppose for a contradiction that $a \notin M$. Then there exist distinct $b, c \in \Sigma$ such that $b \lesssim c \lesssim b \lesssim a$. Hence $b \lesssim c \lesssim b \lesssim x$, contradicting $x \in M$.

(iii) Let $I \in \mathsf{PossIm}(D)$ and let $a, b \in I$. Then $I \subseteq M$ and $a \preceq_I b \iff a\ (\rightsquigarrow^\oplus)|_I\ b \iff a\ (\rightsquigarrow^\oplus)|_M\ b \iff a \preceq_M b$.

□

M should be thought of as the largest down-set of $(\Sigma, \lesssim)$ which induces a partial order (i.e. on which $\lesssim$ is antisymmetric). Proposition 30 greatly simplifies the generation of the set $\mathsf{PossIm}(D)$ in that we need only check that a given down-set I is a subset of M, rather than having to check whether its corresponding order $\preceq_I$ is antisymmetric.

Example 31. Consider again the declarative process D from Example 28. Let us examine the Hasse diagram of $(\Sigma, \lesssim)$ in Figure 4 in order to determine M. In the context of Hasse diagrams, $M := \Sigma \backslash \{a \in \Sigma : b \lesssim c \lesssim b \lesssim a \text{ for some distinct } b, c \in \Sigma\}$ is just Σ without those elements which are part of, or greater than, a directed cycle. In our example, this is readily seen to be the set $\{a, b, c\}$. The down-sets of $(\Sigma, \lesssim)$ which are subsets of $M = \{a, b, c\}$ are $\emptyset, \{a\}, \{b\}, \{a, b\}$ and $\{a, b, c\}$ so by Proposition 30, these are the sets which constitute $\mathsf{PossIm}(D)$.

Considering Theorem 25 in the context of Proposition 30 part (iii) we raise the following question: Given an induced subposet Q of a poset P, can we efficiently determine $\mathcal{L}(Q)$ given $\mathcal{L}(P)$? It turns out that the answer is yes, which means that we can also greatly improve/simplify the generation of $\mathsf{Traces}(D)$.

Lemma 32. Let P be a poset and let Q be an induced subposet of P. Then

$$\mathcal{L}(Q) \;=\; \{\pi|_Q \;:\; \pi \in \mathcal{L}(P)\}.$$

Proof. Let us suppose $\tau = a_1 \dots a_{|P|} \in \mathcal{L}(Q)$. Denote by $\preceq_T$ the total order implied by τ, that is, the relation $\{(a_i, a_j) \in Q^2 : i \leq j\}$, and denote by T the poset $(Q, \preceq_T)$. It can be shown that in this case $(P, (\preceq_P \cup \preceq_T)^{\oplus})$ is a poset, which we denote by $P + T$. Since $(\preceq_P) \subseteq (\preceq_P \cup \preceq_T)^{\oplus}$, we have that $\mathcal{L}(P + T) \subseteq \mathcal{L}(P)$. Now if $\pi \in \mathcal{L}(P + T)$ then $\pi \in \mathcal{L}(P)$ and clearly $\pi|_Q = \tau$.

Conversely, suppose $\pi \in \mathcal{L}(P)$. Then $\pi = \sigma^{-1}(1) \dots \sigma^{-1}(|P|)$ for some linear extension σ of P. Hence $\pi|_Q = \sigma^{-1}(i_1) \dots \sigma^{-1}(i_{|Q|})$ for some $i_1, \dots, i_{|Q|} \in [|P|]$ with $i_1 < \dots < i_{|Q|}$. Define the bijection $\rho : Q \to [|Q|]$ by $\sigma^{-1}(i_k) \mapsto k$, so that $\rho^{-1}(k) = \sigma^{-1}(i_k)$ for all $k \in [|Q|]$. Now suppose that $\rho^{-1}(r) = \sigma^{-1}(i_r) \preceq_Q \sigma^{-1}(i_s) = \rho^{-1}(s)$ for some $r, s \in [|Q|]$. Then $\sigma^{-1}(i_r) \prec_P \sigma^{-1}(i_s)$ so $i_r < i_s$ since σ is a linear extension of P. Hence $r < s$ so ρ is a linear extension of Q. Clearly $\pi|_Q = \rho^{-1}(1) \dots \rho^{-1}(|Q|)$ so $\pi|_Q \in \mathcal{L}(Q)$. □

Now we need only apply a linear extension generating algorithm to the poset $(M, \preceq_M)$ instead of to each and every one of a potentially large number of posets. Once we have generated $\mathcal{L}((M, \preceq_M))$, the remaining $\mathcal{L}((I, \preceq_I))$ are very easily found using Lemma 32. Finally we prove the classification of the set of traces of a precedence-only declarative process.

Theorem 33. Recall the definitions of $\preceq_I$ and $\lesssim$ as given in Definition 19. If D is a precedence-only declarative process, then

$$\mathsf{Traces}(D) = \{\pi|_I : \pi \in \mathcal{L}((M, \preceq_M)) \text{ and } I \in \mathsf{PossIm}(D)\},$$

where $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ which are subsets of $M := \Sigma \backslash \{a \in \Sigma : b \lesssim c \lesssim b \lesssim a \text{ for some distinct } b, c \in \Sigma\}$.

Proof. The fact that $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ which are subsets of M is proven in Proposition 30(i). By Theorem 25 we have that

$$\mathsf{Traces}(D) = \bigcup_{I \in \mathsf{PossIm}(D)} \mathcal{L}((I, \preceq_I)).$$

But if $I \in \mathsf{PossIm}(D)$ then $(I, \preceq_I)$ is an induced subposet of $(M, \preceq_M)$ by Proposition 30(iii), and so $\mathcal{L}((I, \preceq_I)) = \{\pi|_I : \pi \in \mathcal{L}((M, \preceq_M))\}$ by Lemma 32. The result follows. □

Example 34. Consider again the declarative process D from Example 28. We have previously calculated that $M = \{a, b, c\}$ and $\mathsf{PossIm}(D) = \{\emptyset, \{a\}, \{b\}, \{a, b\}, \{a, b, c\}\}$. We may use Pruesse and Ruskey's algorithm to find that $\mathcal{L}((M, \preceq_M)) = \{abc, bac\}$. Applying Theorem 33, we find that

$$\begin{aligned}\mathsf{Traces}(D) =& \left\{abc|_\emptyset, bac|_\emptyset, abc|_{\{a\}}, bac|_{\{a\}}, abc|_{\{b\}}, bac|_{\{b\}}, abc|_{\{a,b\}}, bac|_{\{a,b\}}, abc|_{\{a,b,c\}}, bac|_{\{a,b,c\}}\right\} \\ =& \{\epsilon, a, b, ab, ba, abc, bac\}.\end{aligned}$$

6.2. **Response-only constraint sets.** Consider a declarative process $D = (\Sigma, \mathsf{Const})$ where Const consists only of response constraints. We refer to such a declarative process as a *response-only* declarative process, and to its constraint set as a *response-only* constraint set. This subsection differs only slightly from the previous subsection in that $\rightsquigarrow$ is the relation $\{(a,b) \in \Sigma^2 : \mathsf{resp}(a,b) \in \mathsf{Const}\}$ on Σ and $\lesssim$ is the relation $\{(a,b) \in \Sigma^2 : \mathsf{resp}(b,a) \in \mathsf{Const}\}^{\oplus}$. In other words, we have that $\rightsquigarrow^{\oplus}$ and $\lesssim$ are each others transpose, rather than being equal. We define $(\preceq_I) := (\rightsquigarrow|_I)^{\oplus}$ as before. The results turn out to be identical to the previous subsection, and the proofs very similar.

Lemma 35. Let $D = (\Sigma, \mathsf{Const})$ be a response-only declarative process. If I is a down-set of $(\Sigma, \lesssim)$, then $(\preceq_I) := (\rightsquigarrow|_I)^{\oplus} = (\rightsquigarrow^{\oplus})|_I$.

Proof. If $(a,b) \in \preceq_I$, then the same argument used in the first part of the proof of Lemma 29 shows that this implies $(a,b) \in (\rightsquigarrow^{\oplus})|_I$.

Conversely, suppose that $(a,b) \in (\rightsquigarrow^{\oplus})|_I$ and note that this implies $a, b \in I$. By the same reasoning as before, since $a \rightsquigarrow^{\oplus} b$ there must exists $b_1, \dots b_m \in \Sigma$ such that $a \rightsquigarrow b_1 \rightsquigarrow \dots \rightsquigarrow b_m \rightsquigarrow b$. This implies $a \rightsquigarrow^{\oplus} b_i$ for all $i \in [m]$ from which we have $b_i \lesssim a$, again for all $i \in [m]$. Now since I is a down-set this gives us that $b_i \in I$ for all $i \in [m]$. These properties show

$$a(\rightsquigarrow|_I)b_1(\rightsquigarrow|_I)\dots(\rightsquigarrow|_I)b_m(\rightsquigarrow|_I)b$$

from which we get $a(\rightsquigarrow|_I)^{\oplus}b$. □

Proposition 36. Let $D = (\Sigma, \mathsf{Const})$ be a response-only declarative process and define M to be the set $\Sigma \backslash \{a \in \Sigma : b \lesssim c \lesssim b \lesssim a \text{ for some distinct } b, c \in \Sigma\}$. Then

(i) $\mathsf{PossIm}(D)$ is the set of down-sets of $(\Sigma, \lesssim)$ which are also subsets of M.
(ii) $M \in \mathsf{PossIm}(D)$.
(iii) If $I \in \mathsf{PossIm}(D)$ then $(I, \preceq_I)$ is an induced subposet of $(M, \preceq_M)$.

Proof. (i) Let $I \in \mathsf{PossIm}(D)$. The set I is a down-set of $(\Sigma, \lesssim)$ by Proposition 23. Let $a \in I$ and suppose for a contradiction that $a \notin M$. Then there exist distinct $b, c \in \Sigma$ such that $b \lesssim c \lesssim b \lesssim a$. Now $b, c \in I$ since I is a down-set of $(\Sigma, \lesssim)$. Recall that $\lesssim$ and $\rightsquigarrow^{\oplus}$ are each others transpose and that $(\preceq_I) = (\rightsquigarrow|_I)^{\oplus} = (\rightsquigarrow^{\oplus})|_I$ so $b \preceq_I c \preceq_I b$. Hence $\preceq_I$ is not antisymmetric, contradicting Proposition 23. Therefore $I \subseteq M$.

Let I be a down-set of $(\Sigma, \lesssim)$ such that $I \subseteq M$. Suppose for a contradiction that $I \notin \mathsf{PossIm}(D)$. By Proposition 23, $\preceq_I$ must not be antisymmetric, so there exist distinct $b, c \in I$ such that $b \preceq_I c \preceq_I b$. Again recall that $\lesssim$ and $\rightsquigarrow^{\oplus}$ are each others transpose and that $(\preceq_I) = (\rightsquigarrow|_I)^{\oplus} = (\rightsquigarrow^{\oplus})|_I$ so $b \lesssim c \lesssim b$. Notice that $b \in M$ and $b \lesssim c \lesssim b \lesssim b$, contradicting the definition of M.

The proofs of parts (ii) and (iii) are precisely the same as those of Proposition 30. □

Theorem 37. Recall the definitions of $\preceq_I$ and $\lesssim$ as given in Definition 19. If D is a response-only declarative process, then

$$\mathsf{Traces}(D) = \{\pi|_I : \pi \in \mathcal{L}((M, \preceq_M)) \text{ and } I \in \mathsf{PossIm}(D)\},$$

where $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ which are subsets of $M := \Sigma \backslash \{a \in \Sigma : b \lesssim c \lesssim b \lesssim a \text{ for some distinct } b, c \in \Sigma\}$.

Proof. The fact that $\mathsf{PossIm}(D)$ is the set of all down-sets I of $(\Sigma, \lesssim)$ which are subsets of M is proven in Proposition 36(i). By Theorem 25 we have that

$$\mathsf{Traces}(D) = \bigcup_{I \in \mathsf{PossIm}(D)} \mathcal{L}((I, \preceq_I)).$$

But if $I \in \mathsf{PossIm}(D)$ then $(I, \preceq_I)$ is an induced subposet of $(M, \preceq_M)$ by Proposition 36(iii), and so $\mathcal{L}((I, \preceq_I)) = \{\pi|_I : \pi \in \mathcal{L}((M, \preceq_M))\}$ by Lemma 32, giving the result. □

7. Matters of computation

In this section we provide some comments relating to the practical application of the theorems presented in this paper and the sets that need to be computed in order to derive the trace sets. The main combinatorial objects that the theorems rely on are the set $\mathsf{PossIm}(D)$ and the collection of partial orders $\preceq_I$ for each $I \in \mathsf{PossIm}(D)$. First we need generate the set of down-sets of the preordered set $(\Sigma, \lesssim)$. Next, and in no particular order, we need to determine which of these down-sets I are in $\mathsf{PossIm}(D)$, and generate $\preceq_I$ for each of those which are members. We will tackle each of these problems in a separate subsection. Finally we will comment on the generation of each $\mathcal{L}((I, \preceq_I))$.

7.1. **The down-sets.** It is known that a preorder $\lesssim$ on Σ induces a partial order as follows. Define the relation $\sim$ on Σ by $a \sim b$ if $a \lesssim b$ and $b \lesssim a$, and denote by $[a]_\sim$ the $\sim$ equivalence class of $a \in \Sigma$. Define the relation $\leq$ on $\Sigma/\sim$ by $[a]_\sim \leq [b]_\sim$ if $a \lesssim b$.

Lemma 38. $\leq$ is well defined, and a partial order on $\Sigma/\sim$.

Proof. First we show that $\leq$ is well defined. Suppose that $a \lesssim b$ for some $a, b \in \Sigma$ and let $a' \sim a$ and $b' \sim b$. Then $a' \lesssim a \lesssim b \lesssim b'$ so $a' \lesssim b'$. Hence the definition of $\leq$ is independent of equivalence class representative.

Now we show that $\leq$ is a partial order on $\Sigma/\sim$. Let $[a]_\sim \in \Sigma/\sim$. We have that $a \lesssim a$ so $[a]_\sim \leq [a]_\sim$. Hence $\leq$ is reflexive. Let $[a]_\sim, [b]_\sim \in \Sigma/\sim$ be such that $[a]_\sim \leq [b]_\sim$ and $[b]_\sim \leq [a]_\sim$. Then $a \lesssim b$ and $b \lesssim a$ so $a \sim b$, meaning that $[a]_\sim = [b]_\sim$. Hence $\leq$ is antisymmetric. Let $[a]_\sim, [b]_\sim, [c]_\sim \in \Sigma/\sim$ be such that $[a]_\sim \leq [b]_\sim \leq [c]_\sim$. Then $a \lesssim b \lesssim c$, yielding $a \lesssim c$, and so $[a]_\sim \leq [c]_\sim$. Hence $\leq$ is transitive. □

Now that $(\Sigma/\sim, \leq)$ is a poset, it is also known that $\mathcal{J}((\Sigma/\sim, \leq))$ is graded of rank $k = |\Sigma/\sim|$ and has rank function $\rho' : \mathcal{J}((\Sigma/\sim, \leq)) \to \{0, 1, \ldots, k\}$ defined by $I \mapsto |I|$ [14, Prop 3.4.5].

Proposition 39. The posets $\mathcal{J}((\Sigma/\sim, \leq))$ and $\mathcal{J}((\Sigma, \lesssim))$ are isomorphic under the map $\psi : \mathcal{J}((\Sigma/\sim, \leq)) \to \mathcal{J}((\Sigma, \lesssim))$ defined by

$$I \mapsto \begin{cases} \emptyset & \text{if } I = \emptyset \\ \bigcup_{S \in I} S & \text{if } I \neq \emptyset. \end{cases}$$

Proof. First we show that ψ is a well-defined function. Let I be a down-set of $(\Sigma/\sim, \leq)$. We wish to show that $\psi(I)$ is a down-set of $(\Sigma, \lesssim)$. If $I = \emptyset$ then we are done so suppose otherwise. Let $b \in \psi(I)$ and $a \in \Sigma$ such that $a \lesssim b$. Then $[b]_\sim \in I$ and $[a]_\sim \leq [b]_\sim$ so $[a]_\sim \in I$. Hence $a \in \psi(I)$.

Next we show that ψ is a bijection. Let I and J be distinct down-sets of $(\Sigma/\sim, \leq)$. Without loss of generality let $[a]_\sim \in I \backslash J$. Then $a \in \psi(I) \backslash \psi(J)$ so $\psi(I) \neq \psi(J)$. Therefore ψ is injective. Let I be a down-set of $(\Sigma, \lesssim)$. If $I = \emptyset$ then $\psi(\emptyset) = I$ so suppose that I is nonempty and consider $J := \{[a]_\sim : a \in I\}$. Then $\psi(J) = \bigcup_{S \in J} S = \bigcup_{a \in I} [a]_\sim$ so clearly $I \subseteq \psi(J)$. If $b \in \psi(J)$ then $b \in [a]_\sim$ for some $a \in I$ so $b \sim a$. This shows that $b \lesssim a$, and so $b \in I$ as I is a down-set of $(\Sigma, \lesssim)$. Hence $\psi(J) \subseteq I$ and we have $\psi(J) = I$. Therefore ψ is surjective.

Finally we show that ψ is a poset isomorphism. That is, we show that for all down-sets I, J of $(\Sigma/\sim, \leq)$, $I \subseteq J \iff \psi(I) \subseteq \psi(J)$. Suppose that $I \subseteq J$. Now if $a \in \psi(I)$ then $[a]_\sim \in I$ so $[a]_\sim \in J$ which implies that $a \in \psi(J)$. Thus $\psi(I) \subseteq \psi(J)$. Conversely, suppose that $\psi(I) \subseteq \psi(J)$. Now if $[a]_\sim \in I$ then $a \in \psi(I)$ so $a \in \psi(J)$ which implies that $[a]_\sim \in J$. Thus $I \subseteq J$. □

As a result $\mathcal{J}((\Sigma, \lesssim))$ is also graded of rank k and has rank function $\rho : \mathcal{J}((\Sigma, \lesssim)) \to \{0, 1, \ldots, k\}$ defined by $I \mapsto \rho'(\psi^{-1}(I))$. With all of this in mind we now have the following method for generating the necessary down-sets:

Method 40. To generate the down-sets of $(\Sigma, \lesssim)$ along with their respective ranks in $\mathcal{J}((\Sigma, \lesssim))$.

(i) Generate the poset $(\Sigma/\sim, \leq)$.
(ii) Apply to $(\Sigma/\sim, \leq)$ a known down-set generating algorithm, such as that of Squire [13].
(iii) Assign to each down-set of $(\Sigma/\sim, \leq)$ its rank, which is just its cardinality.

(iv) Apply the isomorphism ψ to each down-set of $(\Sigma/\sim, \preceq)$, leaving its rank unchanged, to yield the down-sets of $(\Sigma, \lesssim)$ along with their respective ranks.

Having generated the down-sets of $(\Sigma, \lesssim)$, one could naively generate $\preceq_I$ for each down-set I one-by-one and apply Proposition 23 to determine whether or not it is an element of $\mathsf{PossIm}(D)$. This turns out to involve a lot of redundancy which we can avoid by considering the down-sets in some order of non-decreasing rank.

7.2. **The partial orders.** To start with, the rank of an element of $\mathcal{J}((\Sigma, \lesssim))$ must be greater than the rank of all of its subsets in $\mathcal{J}((\Sigma, \lesssim))$. This fact can be seen to be true of any collection of sets which, when ordered by inclusion, forms a graded poset. As a result, generating the $\preceq_I$s in order some order of non-decreasing rank allows us to make use of the already generated $\preceq_I$s to make computation more efficient.

Suppose that we wish to generate $\preceq_I$ having already generated $\preceq_J$ for some $J \subseteq I$ in $\mathcal{J}((\Sigma, \lesssim))$. This involves nothing more than finding the transitive and reflexive closure of $\rightsquigarrow|_I$. But we have already done some of the work when we generated $\preceq_J$, since $(\preceq_J) = (\rightsquigarrow|_J)^{\oplus} \subseteq (\rightsquigarrow|_I)^{\oplus} = (\preceq_I)$. This means that we can add all of the elements of $\preceq_J$ to $\rightsquigarrow|_I$ before computing its transitive closure in order to eliminate the redundancy of performing the same computations twice, and still obtain the same result. This is demonstrated in the following identity.

$$\preceq_I = (\preceq_I)^{\oplus} = ((\preceq_I) \cup (\preceq_J))^{\oplus} = ((\rightsquigarrow|_I)^{\oplus} \cup (\preceq_J))^{\oplus} = ((\rightsquigarrow|_I) \cup (\preceq_J))^{\oplus}.$$

Supposing that we have already generated $\preceq_J$ for every $J \subset I$ in $\mathcal{J}((\Sigma, \lesssim))$, we may avoid any possibility of the kind of redundancy described above by adding all of the elements of every $\preceq_J$ to $\rightsquigarrow|_I$ before computing its transitive closure. By the same reasoning as before we have:

$$\preceq_I = \left((\rightsquigarrow|_I) \cup \left(\bigcup_{J \subset I \text{ in } \mathcal{J}((\Sigma,\lesssim))} \preceq_J \right) \right)^{\oplus}. \qquad (\dagger)$$

One could further refine this identity by letting J range only over elements covered by I in $\mathcal{J}((\Sigma, \lesssim))$.

7.3. **The possible images.** Some more redundant computation can be avoided by use of the following lemma.

Lemma 41. $\mathsf{PossIm}(D)$ is a down-set of the poset $\mathcal{J}((\Sigma, \lesssim))$.

Proof. Let $I \in \mathsf{PossIm}(D)$ and let $J \in \mathcal{J}((\Sigma, \lesssim))$ be such that $J \subseteq I$. Suppose for a contradiction that $J \notin \mathsf{PossIm}(D)$. By Proposition 23, we must have that $\preceq_J$ is not antisymmetric. But $(\preceq_J) = (\rightsquigarrow|_J)^{\oplus} \subseteq (\rightsquigarrow|_I)^{\oplus} = (\preceq_I)$ so $\preceq_I$ is also not antisymmetric, contradicting $I \in \mathsf{PossIm}(D)$. □

Note that $\mathsf{PossIm}(D)$ is a collection of down-sets of $(\Sigma, \lesssim)$, but what this lemma tells us is that as a subset of $\mathcal{J}((\Sigma, \lesssim))$, $\mathsf{PossIm}(D)$ is itself a down-set. What this means is that if we find a down-set of $(\Sigma, \lesssim)$ which is not in $\mathsf{PossIm}(D)$, we know that all of its supersets in $\mathcal{J}((\Sigma, \lesssim))$ are not either. If we consider the down-sets of $(\Sigma, \lesssim)$ in order of non-decreasing rank, then we ensure that we do not needlessly generate $\preceq_I$ for any down-set I which we could have otherwise discarded by these means.

Finally we describe our method for determining the elements of $\mathsf{PossIm}(D)$ and generating their respective partial orders, which involves checking the down-sets I of $(\Sigma, \lesssim)$ one-by-one, computing the $\preceq_I$s as we go and 'marking' those I which are not in $\mathsf{PossIm}(D)$.

Method 42. To generate $\mathsf{PossIm}(D)$:

(i) Use Method 40 to generate the down-sets of $(\Sigma, \lesssim)$ along with their respective ranks.
(ii) In order of non-decreasing rank, check the down-sets of $(\Sigma, \lesssim)$, skipping over marked elements.

(iii) Let I be the down-set which is being checked. Since we are checking in order of non-decreasing rank, we have already computed $\preceq_J$ for all $J \subset I$ in $\mathcal{J}((\Sigma, \lesssim))$, so we may compute $\preceq_I$ using (†). If $\preceq_I$ is not antisymmetric then $I \notin \mathsf{PossIm}(D)$ by Proposition 23, so we mark it. Furthermore, by Lemma 41, no superset of I in $\mathcal{J}((\Sigma, \lesssim))$ is in $\mathsf{PossIm}(D)$, so we mark all of these as well.

(iv) Repeat step 2 until we have exhausted all elements of $\mathcal{J}((\Sigma, \lesssim))$, at which point the unmarked elements I will make up $\mathsf{PossIm}(D)$, and we will have generated all of the respective $\preceq_I$s.

Acknowledgements

M.D. wishes to acknowledge the financial support from University College Dublin (OBRSS Research Support Scheme 16426) for this work.

References

[1] L. Aceto, A. Ingólfsdóttir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification.* Cambridge University Press (2007).

[2] M. Dien, A. Genitrini, and F. Peschanski. A Combinatorial Study of Async/Await Processes. *ICTAC 2022*, pages 170–187 (2022).

[3] M. Dukes. A simple declarative model of the Federal Disaster Assistance Policy – modelling and measuring transparency. *EURO Journal on Decision Processes* **11**, 100035 (2023).

[4] M. Dukes. Stakeholder utility measures for declarative processes and their use in process comparisons. *IEEE Transactions on Computational Social Systems* **9**, no. 2, 546–558 (2022).

[5] M. Dukes and A.A. Casey. Combinatorial diversity metrics for declarative processes: an application to policy process analysis. *International Journal of General Systems* **50**, no. 4, 367–387 (2021).

[6] M.B. Dwyer, G.S. Avrunin, J.C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the Twenty-First International Conference on Software Engineering*, pages 411–420, Los Angeles, USA (1999).

[7] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley Longman Publishing, 5th edition (2006).

[8] M. Fattore and R. Bruggemann. *Partial Order Concepts in Applied Sciences.* Springer Cham (2016).

[9] H. Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling.* Wiley, 10th edition (2009).

[10] Object Management Group: *Business Process Modeling Notation Version 2.0.* Technical report, Object Management Group Final Adopted Specification (2011).

[11] M. Pesic, H. Schonenberg, W.M.P. van der Aalst. Declare: Full support for loosely-structured processes. In: *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 287–300, Annapolis, Maryland, USA (2007).

[12] G. Pruesse and F. Ruskey. Generating Linear Extensions Fast. *SIAM Journal on Computing* **23**, no. 2, 373–386 (1994).

[13] M.B. Squire. Enumerating the Ideals of a Poset. *Preprint.* North Carolina State University (1995).

[14] R.P. Stanley. *Enumerative Combinatorics Vol. 1.* Cambridge University Press, 2nd edition (2011).

[15] W. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development* **23**, no. 2, 99–113 (2009).

UCD School of Mathematics and Statistics, University College Dublin, Dublin 4, Ireland.
Email address: mark.dukes@ucd.ie, anton.sohn@ucdconnect.ie