The Computational Complexity of QoS Measures for Orchestrations *

Joaquim Gabarro, Sergio Leon-Gaixas, Maria Serna
ALBCOM Research group. Comp. Sci. Dept.
UPC Barcelona Tech, Spain
email: {gabarro,mjserna}@cs.upc.edu

Abstract

We consider Web services defined by orchestrations in the Orc language and two natural quality of services measures, the *number of outputs* and a discrete version of the *first response time*. We analyse first those subfamilies of finite orchestrations in which the measures are well defined and consider their evaluation in both reliable and probabilistic unreliable environments. On those subfamilies in which the QoS measures are well defined, we consider a set of natural related problems and analyse its computational complexity. In general our results show a clear picture of the difficulty of computing the proposed QoS measures with respect to the expressiveness of the subfamilies of Orc. Only in few cases the problems are solvable in polynomial time pointing out the computational difficulty of evaluating QoS measures even in simplified models.

Keywords: Computational complexity; Orchestration QoS measures; Number of outputs; First response delay; Fully defined variables model; Semantic models.

1 Introduction

As the number of Web services available on-line increases at high speed, one fundamental question is to analyse their performance and reliability. We are interested in the computational limits of computing quality of services (QoS) measures for Web services and to provide a counterpart to many experimental results about the topic. There are plenty of proposals for quality of service (QoS) and Risk analysis for Web services (see as an example [30, 20, 10, 19, 7, 14, 13, 23, 12, 17]). However in all those papers there are almost no results

*J. Gabarro and M. Serna are partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grant TIN2013-46181-C2-1-R (COMMAS) and also by SGR 2014:1034 (ALBCOM) from AGAUR, Generalitat de Catalunya. M. Serna is also supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) through the "Marìa de Maeztu" Programme for Units of Excellence in R&D under Grant MDM-2014-0445 (BGSMath).

name	sites	operators
ElementaryOrc	$1(x)$, 0, $S(x_1,\ldots,x_n)$	$\mid$, $>x>$, $\gg$
PruningOrc	$1(x)$, 0, $S(x_1,\ldots,x_n)$	$\mid$, $>x>$, $\gg$, $<x<$
IfOrc	$1(x)$, 0, $S(x_1,\ldots,x_n)$, $\mathit{if}(x)$	$\mid$, $>x>$, $\gg$
Orc	$1(x)$, 0, $S(x_1,\ldots,x_n)$, $\mathit{if}(x)$	$\mid$, $>x>$, $\gg$, $<x<$

Table 1: The subfamilies of Orc expressions.

indicating the computational limits of the proposed QoS measures. The aim of this paper is to set some theoretical basis to define formally QoS measures and to study the computational limits of such measures at least for some subfamilies of Web services defined by orchestrations. We adopt the Orc language [21] which was proposed as a minimalistic language to describe orchestrations of Web services. In Orc, services are modeled by sites which have some predefined semantics. A *site* accepts an argument and *publishes* a unique result value. For example, a call to a search engine, $\mathit{Find}(x)$, may publish the set of sites which *currently* offer service x. An orchestration which composes a number of service calls into a complex computation is represented by an *Orc expression.* Thus an orchestration *publishes* a stream of values. We restrict ourselves to the subfamily of finite orchestration described by Orc expressions, i.e. orchestrations without iteration and infinite recursion.

Once the subset of orchestrations is defined we introduce two QoS measures of interest. Our first step is to devise the appropriate semantics that guarantees a correct definition of the intended measures. There are many semantics proposed for orchestrations described in Orc [29, 25, 15, 9, 16, 28]. However, we derive some minimalistic semantics to reason about the QoS measures considered in this paper. We focus on two natural QoS measures related to the *productivity* and the *latency* of the system. The *number of outputs* providing the length of the produced stream, and a delay measure, the *first item delay*, providing the delay incurred in the production of the first output. We analyse on which subfamilies of finite orchestrations those measures are well defined. Our results show that this happens when only *non-blocking* sites are allowed or when blocking sites are allowed but the pruning operator is forbidden. This leads us to consider three subfamilies of orchestrations (See Table 1 and Section 2.1 for formal definitions.)

QoS must be assessed in the presence of misbehaviours or failures, usually inside some probabilistic framework. In our probabilistic model we assume some knowledge on the unreliable behaviour of the sites, which is given by a probability of success or a probability distribution on a discreet subset of values of interest. Our probabilistic model is a generalization of the percentage of success that is usually given by the provider [2, 26, 1]. We extend, as usual, the QoS measures to unreliable environments by computing their expectation or their probability of success.

We analyse the complexity of several computational problems related to the considered QoS measures both in reliable and unreliable environments. Table 2 provides an overview of the results in the paper. Our results show a natural

Problem	ElementaryOrc	PruningOrc	IfOrc
ExistsOut	P [11]	P [11]	NP-complete
Out	P [11]	P [11]	#P-complete
PrPub oblivious	needs EXPSPACE	needs EXPSPACE	needs EXPSPACE
PrPub stable	#P-hard, PSPACE	#P-hard, PSPACE	#P-hard, PSPACE
ExpectedOut oblivious	P	#P-hard	#P-hard
ExpectedOut stable	PSPACE	#P-hard, PSPACE	#P-hard, PSPACE
BoundFirst	P	P	NP-complete
First	P	P	NP-hard, PSPACE
PrBoundFirst oblivious	needs EXPSPACE	needs EXPSPACE	needs EXPSPACE
PrBoundFirst stable	#P-hard, PSPACE	#P-hard, PSPACE	#P-hard, PSPACE
ExpectedFirst oblivious	needs EXPSPACE	needs EXPSPACE	needs EXPSPACE
ExpectedFirst stable	#P-hard, PSPACE	#P-hard, PSPACE	#P-hard, PSPACE

Table 2: Summary of results.

jump of increasing complexity from the reliable to the unreliable setting. In the reliable setting the expressiveness and the computational mechanism used in the construction of the Orc expression have an impact in the computational complexity of the considered problems. We can derive efficient algorithms when sites are non-blocking and the pruning operator is allowed. The problems become intractable when blocking sites are allowed, even when the pruning operator is forbidden. Finally, in the probabilistic model complexity changes depend more on correlations than on the expressiveness of the orchestrations. In general, our membership results make a careful use of variable contexts and of the structure of the orchestration. Hardness results follow from reductions from variations of the satisfiability problem for boolean formulas.

We have found only very few non experimental papers dealing with our topic of interest. In [24] the study of orchestrations in unreliable environments with known probabilities has been undertaken using finite Markov chains. Here we develop an alternative approach based on probabilistic environments.

The paper is organized as follows. In Section 2 we provide an overview of Orc and state our main hypothesis. We also introduce the probabilistic framework used to study QoS measures in unreliable environments. In Section 3 we deal with the number of outputs measure, we first analyse in which Orc subfamilies the measure is well defined. Then, we introduce and study the complexity of the associated computational problems. Section 4 defines the first item delay measure and analyses the Orc subfamilies in which it is well defined. This is complemented with the study of the complexity of the associated problems. We conclude in Section 5 presenting our conclusions and some lines for future research. We assume familiarity with Orc and the computational complexity classes P,NP,#P and PSPACE and EXPPSPACE [6, 22]. We adopt the usual convention that numbers appearing in a problem are either natural or rational. For rational numbers we assume a representation as an irreducible fraction, so they are represented by two numbers in binary.

2 Preliminaries

We start describing in a general way the components of the proposed framework for orchestrations.

2.1 Orc expressions

An orchestration which composes a number of service calls into a complex computation can be described in *Orc* [21]. An orchestrator may utilize any service that is available on the Web. The simplest kind of Orc expression is a *site* (service) call $S(x_1, \ldots, x_n)$, where S is the service's name and $(x_1, \ldots, x_n)$ is a list of formal parameters, for some $n \geq 0$. Thus, when it is executed and the formal parameters hold acceptable arguments $(v_1, \ldots, v_n)$ it will *publish* (return, output) the result value $s(v_1, \ldots, v_n)$. As usual we distinguish the syntactic call $S(x_1, \ldots, x_n)$ on variables from the computed function $s(v_1, \ldots, v_n)$ with an abuse of notation we write $(x_1, \ldots, x_n) = (v_1, \ldots, v_n)$ to express the fact that the formal parameters are assigned to the corresponding value. For sites with 0 parameters we write $S()$ and for sites with just one parameter $S(x)$. A site call is *silent* if it does not publish a result. Orc has two special internal sites, site 1 and site 0. A call to site 0 never publishes a result and thus remains silent. A call to site 1 always returns one signal. Site 1 admits calls with any number of parameters, with the form $1(x_1, \ldots, x_n)$ (or $1(x)$), the call returns a signal whenever all the variables in the parameter list are defined. Usually we do not need to distinguish among the published value of a call to site 1, however when we want that site 1 publishes a boolean value, we denote this fact as $1(\mathtt{1})$ and $1(\mathtt{0})$. In the first case the published value is 1 (true) and in the second 0 (false). We model the output of a site by a string. As usual, we use + to denote string concatenation.

The following table provides a description of some of the sites appearing in other examples. Most of them are taken from [21].

Site	Meaning	Published Value
AddressAlice	publishes Alice's email	`aalice`
AddressBob	publishes Bob's email	`abob`
Email(a, m)	sends m to email a and publishes	`s_` + `a` + `_` + `m`
EmailAlice(m)	sends m to Alice's email and publishes	`s_aalice_` + `m`
EmailBob(m)	sends m to Bob's email and publishes	`s_abob_` + `m`
EmailDad(m)	sends m to my dad's email and publishes	`s_adad_` + `m`
CNN	publishes a summary of the CNN news	`cnn`
BBC	publishes a summary of the BBC news	`bbc`

For instance, the call *Email*(`aalice`, `cnn`) publishes the string `s_aalice_cnn`.

We consider orchestrations, where the orchestrator calls different "external" sites like *CNN* or *Email*(a, m). We assume that all the "external" sites have well-defined behaviours, implementing polynomial time computable functions and acting according to the non-blocking hypothesis and the constant delay hypothesis [11]:

> A site S is *non-blocking* if $S(x_1, \ldots, x_n)$ must publish a result for any well-defined arguments $v_1, \ldots, v_n$; otherwise S is *potentially blocking*.
>
> Non-blocking hypothesis: Every external site is non-blocking.
>
> Constant delay hypothesis: Every site S has associated a delay δ_S and, in any call, S publishes after δ_S time units from the time at which all their parameters are defined.

We consider an exception to the non-blocking hypothesis as we can use an additional internal site, the site $if(x)$. This site was introduced in [21]. A call to $if(b)$ publishes a signal when b gets the value true and remains silent otherwise. Note that site $if(x)$ is potentially blocking. We can think of such a site as being implemented "locally" by the orchestrator. This is the unique blocking site allowed in an orchestration.

In an orchestration the results published by a sub-orchestration can be stored in a local variable that can be used as a parameter in another site call. Observe that some times a variable x remains undefined (written as $x = \perp$). We use upper-case letters for external sites and orchestrations and lower-case letters for variables. Notice that, although a site call publishes at most one result, this is not the case of an orchestration as the combined structure will publish a stream of data. In this paper we deal only with *finite orchestrations* where finite means: excluding iteration and recursion. If P and Q are Orc expressions then the following expressions are also Orc expressions [21].

- Sequence $P > x > Q(x)$: P is evaluated and, for each value v published by P, an instance $Q(v)$ is executed. If P publishes the stream, $v_1, v_2, \ldots v_n$, then $P > x > Q(x)$ publishes some interleaved stream of the outputs of the calls $Q(v_1), Q(v_2), \ldots, Q(v_n)$. When the value of x is not needed we write $P \gg Q$.

- Symmetric Parallelism $P \mid Q$: P and Q are evaluated in parallel. $P \mid Q$ publishes *some* interleaving of the streams published by P and Q.

- Pruning $P(x) < x < Q$: P and Q are evaluated in parallel. Some sub-expressions in P may become blocked by a dependency on x. The first result published by Q is bound to x, the remainder of Q's evaluation is terminated and evaluation of the blocked residue of P is resumed[1].

In the remaining of the paper we use the term Orc to denote the set of finite orchestrations that can be constructed using the previous operators in which all sites, except $if(b)$, follow the non-blocking hypothesis. We also consider three subfamilies of Orc (see Fig.1). In ElementaryOrc the $if(b)$ site and the pruning operator are not allowed. In PruningOrc we forbid the use of the $if(b)$ site

[1]The expression "$P(x) < x < Q$" was encoded as "$P(x)$ **where** $x :\in Q$" and called *asymmetric parallelism* in in [21].

and, symmetrically, in IfOrc the pruning operator is not allowed. Observe that iteration and infinite recursion is not allowed in any of the subfamilies, however our hardness result hold for those cases in which the proposed QoS measures could be defined. We call the operational semantics induced by the definitions and given explicitly in [21] the *Misra-Cook semantics.*

Following we provide some examples of orchestrations. In the examples and in some constructions, we assign names to sub-expressions. You should bear in mind that the representation of the Orc expression (according to the definition) requires the replacement of names by their description in terms of sites.

Example 1. *Let us analyse informally the Misra-Cook semantics of the following orchestrations. Let us start with TwoAlice orchestration:*

$$\mathit{TwoAlice} = (\mathit{CNN} \mid \mathit{BBC}) > x > \mathit{EmailAlice}(x)$$

which for sake of readability is rewritten as:

$$\begin{aligned} &\mathit{TwoNews} = (\mathit{CNN} \mid \mathit{BBC}),\\ &\mathit{TwoAlice} = \mathit{TwoNews} > x > \mathit{EmailAlice}(x) \end{aligned}$$

This orchestration has a very rich behaviour. Initially x is undefined. Suppose that CNN returns first, in this case x stores the value `cnn`, *$\mathit{EmailAlice}(\texttt{cnn})$ is called and x becomes undefined again. If later on BBC publishes some result, x will store* `bbc` *and $\mathit{EmailAlice}(\texttt{bbc})$ will be called.*

Consider now the FullCrash expression:

$$\begin{aligned} &\mathit{NewsEmail}(x) = (\mathit{CNN} \mid \mathit{EmailAlice}(x)),\\ &\mathit{FullCrash} = 0 > x > \mathit{NewsEmail}(x) \end{aligned}$$

As 0 never returns, according to the definition of sequential composition, $\mathit{NewsEmail}(x)$ will never be executed and FullCrash publishes no result.

In $\mathit{OneAlice} = \mathit{EmailAlice}(x) < x < \mathit{TwoNews}$, observe that the variable x in OneAlice can take either the value `cnn` *or* `bbc` *unpredictably as the pruning operator introduces non-determinism.*

Finally, the expression

$$\mathit{OneTwo} = \big(\mathit{if}(b) \gg \mathit{CNN} \mid \mathit{if}(\neg b) \gg (\mathit{BBC} \mid \mathit{FOX})\big) < b < (1(\texttt{1}) \mid 1(\texttt{0}))$$

publishes (non deterministically) either `cnn` *or the stream* `bbc`, `fox`. □

In the following example we provide a more complex expressions where we want to emphasize the role of the variables.

Example 2. *Consider the following orchestration MyNews:*

$$\begin{aligned} &\mathit{TwoNews} = (\mathit{CNN} \mid \mathit{BBC})\\ &\mathit{MyEmails}(a, x) = (\mathit{EmailDad}(x) \mid \mathit{Email}(a, x))\\ &\mathit{SendNews}(a) = (\mathit{TwoNews} > x > \mathit{MyEmails}(a, x))\\ &\mathit{Addresses} = (\mathit{AddressAlice} \mid \mathit{AddressBob})\\ &\mathit{MyNews} = (\mathit{SendNews}(a) < a < \mathit{Addresses}) \end{aligned}$$

The MyNews expression corresponds to

$$\begin{aligned} \mathit{MyNews} \;\;=\;\; & \big((\mathit{CNN} \mid \mathit{BBC}) > x > (\mathit{EmailDad}(x) \mid \mathit{Email}(a,x))\big) \\ & < a < (\mathit{AddressAlice} \mid \mathit{AddressBob}) \end{aligned}$$

where only site calls appear. □

We have to be careful when dealing with iterative descriptions as this might give rise to expressions with exponential size. In the following example we provide one of such constructions.

Example 3. *Consider the orchestration Bang_n defined as follows*

$$\begin{aligned} \mathit{Bang}_0 &= \big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) \\ \mathit{Bang}_1 &= (\mathit{Bang}_0 \gg \mathit{Bang}_0) \\ \mathit{Bang}_n &= \big(\mathit{Bang}_{n-1} \gg \mathit{Bang}_{n-1}\big) \;\; \textit{for } n > 0 \end{aligned}$$

The iterative description by sub-expressions is quite succinct. Observe that when $n = 2$ we have:

$$\mathit{Bang}_2 = \Big(\big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) \gg \big(1(\mathtt{1}) \mid 1(\mathtt{1})\big)\Big) \gg \Big(\big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) \gg \big(1(\mathtt{1}) \mid 1(\mathtt{1})\big)\Big)$$

Expanding and replicating the sub-expressions it is easy to see that the Orc *expression corresponding to Bang_n has exponential size in relation to n.* □

2.2 Probabilistic environments:

In order to define the probabilistic model, we assume that a QoS measure m is defined for orchestrations. We assume that, for each participating site S, the number of possible values of m is finite. A *distribution* of m for site S is a lottery assigning a probability to each of those values. Observe that such distributions sometimes can be inferred from vendor (or general) information [2, 26, 1], from system logs, or can be constructed experimentally by the user. Our notation is inspired by [18, p. 20]. We denote a QoS distribution for an orchestration E and a measure m by $P_E = (m_1@p_1 \parallel \cdots \parallel m_k@p_k)$ where m_i are the possible values of m and p_i, $0 \leq p_i \leq 1$, is the probability that the value is m_i and $\sum_{i=1}^{k} p_i = 1$. In $P_E = (m_1@p_1 \parallel \cdots \parallel m_k@p_k)$ we think of "$\parallel$" as a probabilistic choice operator[2]. When only two values are possible we use the shorter description $(m_1 \;{}_p\!\oplus m_2) = \big(m_1@p \parallel m_2@(1-p)\big)$. To deal with such probability distributions, we consider probabilistic orchestrations. $E = (E_1@p_1 \parallel E_2@p_2 \parallel \cdots \parallel E_n@p_n)$ denotes the probabilistic orchestration where E_i is executed with probability p_i. When there are two orchestrations we write $E = (E_1 \;{}_p\!\oplus E_2)$.

Following we provide examples of diverse probabilistic behaviour of services. Those examples have been obtained experimentally using a pre-specified time-out to identify a crash. Whenever the pre-specified time-out is triggered, we

[2]In [18, p. 20] the notation $(\mathit{prog}_1@p_1 \mid \cdots \mid \mathit{prog}_k@p_k)$ is used. We avoid $\mid$, for probabilistic choice, as it represents the Orc operator parallel composition.

assume that the service will never publish. We have considered three services available on the Web. *IPFW* is a service offering static pages [3]. A call to the services returns a fixed size table with Spanish verb conjugations. We took a fixed collection of 100 verbs and perform calls with a selected random verb. *StackOverflow* is a well known dynamic site where you can ask computing related questions [4]. Again, we selected 100 questions, chosen from the web page, and each call to the service was done with a randomly selected question. Finally, we used the news server from *Yahoo*! [5]. In this case the calls were issued to the main page, a dynamic page showing the latest news from different sources and few other info. To get the probability distribution estimation we issued, within a week, $n = 10000$ calls to each server. For each call we recorded the duration (in intervals of 500 ms). The obtained probability distributions, for the first response delay, are the following (we use ω to mean that the "time-out" was triggered):

$$P_{IPFW} = (500@0.955 \parallel 1000@0.053 \parallel \omega@0.002)$$
$$P_{StackOverflow} = (1000@0.998 \parallel \omega@0.002),$$
$$P_{Yahoo!} = (1500@0.203 \parallel 2000@0.777 \parallel 2500@0.006 \parallel 3000@0.004 \parallel \omega@0.01)$$

Observe the big span in the response time within a relatively low number of calls in a short period, for the case of *Yahoo*!

Definition 1. *Given an orchestration E having calls to sites $\{S_1, \ldots, S_n\}$, and a QoS measure m a* probabilistic environment *is a tuple $\mathcal{P} = (P_1, \ldots, P_n)$ where for each $1 \leq i \leq n$, P_i is a QoS failure distribution of m for site S_i, that is $\mathsf{m}(S_i)$ behaves according to distribution P_i.*

We assume that the failure distributions depend only on the site and are independent among sites. As an orchestration can issue many calls to the same site, we assume that, for probabilistic orchestrations, the outputs of any successful call to a particular site remain independent. We consider two models:

- **Oblivious model**. Inside the orchestration the faulty behaviour of any call to a site S is independent from any other call to S.
- **Stable model**. The faulty behaviour of the first call to a site S is replicated in any other call to S.

Given a probabilistic environment $\mathcal{P}$, we are interested in modeling the faulty behaviour of E under $\mathcal{P}$ in the oblivious (o) and the stable (s) models and in computing the probability of some events and their expectation. We use the notation $\Pr^{\alpha}_{\mathcal{P}}$ or $\mathbb{E}^{\alpha}_{\mathcal{P}}$ to denote, respectively, the probability and expectation under probabilistic environment $\mathcal{P}$ and model $\alpha \in \{o, s\}$. We drop some of the indices ($\mathcal{P}$ or α) when they are clear from the context.

3 Number of outputs

In order to deal with measures related to the number of outputs produced by the orchestration, we consider the operational semantics introduced in [21].

In such a model any variable x contains all the possible values before being used. This approach provides a mathematical tool to derive the desired results. Many variables in orchestrations keep a value or a stream of values. When an orchestration E publishes a stream $v_1, v_2, \ldots, v_n$, the relative ordering of the values depends on the relative response time of the sites appearing in E. If we *abstract from time*, which is possible as we are interested only in the length of the stream, the possible streams can be described by a multi-set or bag $\lfloor\lfloor v_1, v_2, \ldots, v_n \rfloor\rfloor$ (notation $\lfloor\lfloor \cdot \rfloor\rfloor$ is taken from [18]). In such a case, the "meaning" of E, denoted by $[\![E]\!]$, is the bag $\lfloor\lfloor v_1, v_2, \ldots, v_n \rfloor\rfloor$ and we write $[\![E]\!] = \lfloor\lfloor v_1, v_2, \ldots, v_n \rfloor\rfloor$. The fact that site 0 never returns is formalized as site 0 returns nothing, that is $[\![0]\!] = \lfloor\lfloor\ \rfloor\rfloor$. When using multi-sets we consider the operatior "+", denoting bag union, following [18, p. 82], as required by the symmetric parallelism composition. As the pruning operator can give rise to a non deterministic behaviour, we introduce also the "demonic choice" operator "$\sqcap$" [18, p. 4] to denote non deterministic choice. Observe that, in the presence of non-deterministic choice, the number of outputs is well defined only in those cases in which we can prove that all the possible bags have the same size. In the same lines we associate a meaning $[\![x]\!]$ to each variable x appearing in an Orc expression.

Example 4. *Consider the following orchestrations whose meaning in the Misra-Cook semantics is commented in Example 1.*

In TwoAlice we have

$$[\![\mathit{TwoNews}]\!] = \lfloor\lfloor \mathtt{cnn}, \mathtt{bbc} \rfloor\rfloor$$
$$[\![x]\!] = [\![\mathit{TwoNews}]\!] = \lfloor\lfloor \mathtt{cnn}, \mathtt{bbc} \rfloor\rfloor$$
$$[\![\mathit{TwoAlice}]\!] = \lfloor\lfloor \mathtt{s\_aalice\_cnn}, \mathtt{s\_aalice\_bnn} \rfloor\rfloor$$

Therefore, TwoAlice publishes short summaries of the two emails sent to Alice.

In FullCrash, NewsEmail(x) *will never be executed and* $[\![\mathit{FullCrash}]\!] = [\![\ ]\!]$. *In the orchestration OneToAlice, the variable* x *can take the value* `cnn` *or* `bbc`*, therefore* $[\![x]\!] = \lfloor\lfloor \mathtt{cnn} \rfloor\rfloor \sqcap \lfloor\lfloor \mathtt{bbc} \rfloor\rfloor$ *and* $[\![\mathit{OneToAlice}]\!] = \lfloor\lfloor \mathtt{s\_aalice\_cnn} \rfloor\rfloor \sqcap \lfloor\lfloor \mathtt{s\_aalice\_bbc} \rfloor\rfloor$. *Note that* $[\![\mathit{OneTwo}]\!] = \lfloor\lfloor \mathtt{cnn} \rfloor\rfloor \sqcap \lfloor\lfloor \mathtt{bbc}, \mathtt{fox} \rfloor\rfloor$ *and the orchestration can, in a non-deterministic way, output a stream having length* 1 *or another with length* 2.

Let us consider another example of pruning $E_1(x) < x < E_2$ *when* $x = \perp$. *Consider the orchestration PartialCrash* $=$ *NewsEmail*$(x) < x < 0$. *Recall that the* 0 *in PartialCrash denotes a failing service. Then, as* 0 *never publishes, we have* $x = \perp$, *thus* $[\![x]\!] = \lfloor\lfloor\ \rfloor\rfloor$ *and* $[\![\mathit{NewsEmail}(\perp)]\!] = \lfloor\lfloor \mathtt{cnn} \rfloor\rfloor$. *Finally* $[\![\mathit{PartialCrash}]\!] = \lfloor\lfloor \mathtt{cnn} \rfloor\rfloor$ *as expected.* □

In order to show the existence of a *normal form* we analyze first the meaning associated to the syntactic constructions in some particular cases. As in the fully defined variable semantics we abstract from the different publishing times the meaning of an execution consists of the bag of published values. We introduce some notation to deal with bags. We use $\lfloor\lfloor \cdots \rfloor\rfloor$ to denote bags [18, p. 82] and $\#\lfloor\lfloor \cdots \rfloor\rfloor$ to denote the number of items in the bag.

Recall that the empty set is denoted as $\lfloor\lfloor\ \rfloor\rfloor$. For instance, $\lfloor\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\rfloor$ denotes a bag containing the values cnn and bbc. We use $+$ to denote multiset union and $\sqcap$ to denote non-deterministic choice. The result of a multi-set union is the multiset formed by the two multiset. As an example $\lfloor\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\rfloor + \lfloor\lfloor \texttt{fox}, \texttt{bbc} \rfloor\rfloor = \lfloor\lfloor \texttt{cnn}, \texttt{bbc}, \texttt{bbc}, \texttt{fox} \rfloor\rfloor$. Observe that bbc appears in both multisets and in consequence appears twice in their multiset union. The result of a non-deterministic choice is one of the multisets, however we cannot control which one. So, $\lfloor\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{fox}, \texttt{bbc} \rfloor\rfloor$ can unpredictably being either $\lfloor\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\rfloor$ or $\lfloor\lfloor \texttt{fox}, \texttt{bbc} \rfloor\rfloor$. We further assume the following "distributive" behaviour of the non-deterministic choice in front of the multi-set union operation

$$\begin{aligned}&(\lfloor\lfloor \texttt{cnn} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc} \rfloor\rfloor) + (\lfloor\lfloor \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{disney} \rfloor\rfloor)\\ &\quad = \lfloor\lfloor \texttt{cnn}, \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{cnn}, \texttt{disney} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc}, \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc}, \texttt{disney} \rfloor\rfloor\end{aligned}$$

Given multi-sets $M_1, \ldots, M_n$ we note as usual $\sum_{1\le i\le n} M_i = M_1 + \cdots + M_n$ and $\sqcap_{1\le i\le n} M_i = M_1 \sqcap \cdots \sqcap M_n$.

Let us now analyze the meaning associated to some syntactic constructions. For the internal sites, we have:

$$[\![0]\!] = \lfloor\lfloor\ \rfloor\rfloor \quad [\![1(x)]\!] = \begin{cases} \lfloor\lfloor v \rfloor\rfloor & \text{if } x = v \\ \lfloor\lfloor\ \rfloor\rfloor & \text{if } x = \bot \end{cases} \qquad [\![\mathit{if}(b)]\!] = \begin{cases} \lfloor\lfloor \texttt{1} \rfloor\rfloor & \text{if } b = \texttt{true} \\ \lfloor\lfloor\ \rfloor\rfloor & \mathit{otherwise} \end{cases} \tag{1}$$

Assuming, as mentioned before, that a site call $S(x_1, \ldots x_n)$ when the parameters are assigned to the values $(v_1, \ldots v_n)$ returns $s(v_1, \ldots v_n)$ and the non-blocking hypothesis, the meaning of the external site calls is

$$[\![S(x_1, \ldots x_n)]\!] = \begin{cases} \lfloor\lfloor s(v_1, \ldots, v_n) \rfloor\rfloor & \text{if } (x_1, \ldots x_n) = (v_1, \ldots v_n) \\ \lfloor\lfloor\ \rfloor\rfloor & \text{if } \exists i : 1 \le i \le n : x_i = \bot \end{cases} \tag{2}$$

In the following we describe, inductively, the multi-set decomposition corresponding to the Orc operators, assuming that the orchestration components verify this property. For doing so, for the two orchestrations E_1 and E_2, we assume, that $[\![E_1]\!] = \sqcap_{1\le i\le n_1} M_i$ and $[\![E_2]\!] = \sqcap_{1\le j\le n_2} M'_j$, for some multisets $M_1, \ldots, M_{n_1}, M'_1, \ldots, M'_{n_2}$. When some of the orchestrations is parameterized we assume that the multisets are parameterized by the same variable in the usual sense: each valid assignment of values to the variable determines a multiset. In such a case we write $M(x)$ to denote this dependency and $M(v)$ for the multiset determined by the assignment $x = v$.

The parallel composition executes both subexpressions in parallel, therefore, we have

$$[\![(E_1 \mid E_2)]\!] = [\![E_1]\!] + [\![E_2]\!] = \sqcap_{1\le i\le n_1} \sqcap_{1\le j\le n_2} (M_i + M'_j). \tag{3}$$

For the sequential composition $E = E_1 > x > E_2(x)$, let us comment first two easy cases. When $n_1 = 0$, $[\![E_1]\!] = \lfloor\lfloor\ \rfloor\rfloor$, and therefore $[\![E]\!] = \lfloor\lfloor\ \rfloor\rfloor$. When

$E_2(x)$ is just a site call $S(x)$, in the case where $[\![E_1]\!] = \lfloor\!\lfloor v_1, \dots v_n \rfloor\!\rfloor$, $[\![x]\!] = [\![E_1]\!]$ and

$$[\![E]\!] = \lfloor\!\lfloor s(v_1), \dots, s(v_n) \rfloor\!\rfloor.$$

But when $[\![E_1]\!] = \lfloor\!\lfloor v_1, \dots v_n \rfloor\!\rfloor \sqcap \lfloor\!\lfloor w_1, \dots w_m \rfloor\!\rfloor$ and $E_2(x) = S(x)$, we have[3] $[\![E]\!] = \lfloor\!\lfloor s(v_1), \dots, s(v_n) \rfloor\!\rfloor \sqcap \lfloor\!\lfloor s(w_1), \dots, s(w_n) \rfloor\!\rfloor$. Let us consider another case, where

$$[\![E_1]\!] = \lfloor\!\lfloor v \rfloor\!\rfloor \text{ and } [\![E_2(x)]\!] = M'_1(x) \sqcap M'_2(x)$$

then[4]

$$[\![E]\!] = M'_1(v) \sqcap M'_2(v)$$

For the general case, first observe that the meaning of a variable is the bag of values that are stored in the variable in some of the execution paths of the expression. In the construction $E_1 > x > E_2(x)$, we have $[\![x]\!] = [\![E_1]\!] = \sqcap_{1 \le i \le n_1} M_i$. According to the definition the meaning of the whole orchestration is

$$[\![E_1 > x > E_2(x)]\!] = \sqcap_{1 \le i \le n_1} \sqcap_{1 \le j \le n_2} \sum_{m \in M_i} M'_j(m). \tag{4}$$

Finally, let us consider the pruning operator $E_1(x) < x < E_2$ recall that as we abstract from time, pruning is modeled through non-determinism. Let us consider first the case $E_1(x) = S(x)$ and $[\![E_2]\!] = \lfloor\!\lfloor v_1, \dots, v_n \rfloor\!\rfloor$. In such a case the meaning of the variable x is a non-deterministic choice of the outputs of E_2, i.e., $[\![x]\!] = \lfloor\!\lfloor v_1 \rfloor\!\rfloor \sqcap \cdots \sqcap \lfloor\!\lfloor v_n \rfloor\!\rfloor = \sqcap_{1 \le i \le n} \lfloor\!\lfloor v_i \rfloor\!\rfloor$. Therefore, the meaning of the expression is $[\![E_1(x) < x < E_2]\!] = \sqcap_{1 \le i \le n} \lfloor\!\lfloor s(v_i) \rfloor\!\rfloor$.
In the general case $[\![E_2]\!] = \sqcap_{1 \le j \le n_2} M'_j$ and the variable x in $E_1(x) < x < E_2$ has meaning $[\![x]\!] = \sqcap_{1 \le j \le n_2} \sqcap_{m' \in M'_j} \lfloor\!\lfloor m' \rfloor\!\rfloor$ and

$$[\![E_1(x) < x < E_2]\!] = \sqcap_{1 \le i \le n_1} \sqcap_{1 \le j \le n_2} \sqcap_{m' \in M'_j} M_i(m'). \tag{5}$$

Using the preceding associations the following theorem can be proved by structural induction. Observe that, in the case of sites, the decomposition trivially holds and that this decomposition is maintained through the different operations.

Theorem 1. *Given an* Orc *expression E it holds that either $[\![E]\!] = \lfloor\!\lfloor\ \rfloor\!\rfloor$ or there is a unique non-deterministic finite decomposition in multi-sets $[\![E]\!] = \sqcap_i M_i$, elements in M_i corresponds to the possible values returned by site calls.*

Example 5. *Let us compute the bags corresponding to MyNews introduced in the Example 2. Clearly $[\![TwoNews]\!] = \lfloor\!\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\!\rfloor$. Let us consider the meaning of the variables. As x appears in a sequential composition $[\![x]\!] = \lfloor\!\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\!\rfloor$. Let us consider a, note that $[\![Addresses]\!] = \lfloor\!\lfloor \texttt{aalice}, \texttt{abob} \rfloor\!\rfloor$. As a*

[3]This definition can be seen as an adaptation of the law $(prog_1 \sqcap prog_2); prog_3 = prog_1; prog_3 \sqcap prog_2; prog_3$ given in [18, p. 323] to orchestrations.

[4]As we are interested in counting problems we adapt the rule $prog_1; (prog_2 \sqcap prog_3) \sqsubseteq prog_1; prog_2 \sqcap prog_1; prog_3$ given in [18, p. 324] replacing $\sqsubseteq$ by an equality.

appears in a pruning operation $[\![a]\!] = \lfloor\lfloor \texttt{aalice} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{abob} \rfloor\rfloor$. *The orchestration* $\mathit{SendNews}(a)$ *verifies* $[\![\mathit{SendNews}(a)]\!] = \lfloor\lfloor m_1(a), m_2(a), m_3(a), m_4(a)]\!]$ *where the elements* $m_1(a) = \texttt{s\_adad\_cnn}$ *an* $m_2(a) = \texttt{s\_adad\_bbc}$ *are independent of* a. *In* $m_3(a) = \texttt{s\_}a\texttt{\_cnn}$, $m_4(a) = \texttt{s\_}a\texttt{\_bbc}$ *the variable* a *has to be replaced by an address. We shortly write*

$$[\![\mathit{SendNews}(a)]\!] = \lfloor\lfloor \texttt{s\_adad\_cnn}, \texttt{s\_adad\_bbc}, \texttt{s\_}a\texttt{\_cnn}, \texttt{s\_}a\texttt{\_bbc} \rfloor\rfloor.$$

Finally, as $[\![\mathit{MyNews}]\!] = [\![\mathit{SendNews}(a) < a < \mathit{Addresses}]\!]$, *using* (5) *we get*

$$\begin{aligned} [\![\mathit{MyNews}]\!] &= [\![\mathit{SendNews}(\texttt{aalice})]\!] \sqcap [\![\mathit{SendNews}(\texttt{abob})]\!] \\ &= \lfloor\lfloor \texttt{s\_adad\_cnn}, \texttt{s\_adad\_bbc}, \texttt{s\_aalice\_cnn}, \texttt{s\_aalice\_bbc} \rfloor\rfloor \\ &\quad \sqcap \lfloor\lfloor \texttt{s\_adad\_cnn}, \texttt{s\_adad\_bbc}, \texttt{s\_abob\_cnn}, \texttt{s\_abob\_bbc} \rfloor\rfloor. \end{aligned}$$

□

Example 6. *In the following orchestration:*

$$\mathit{OneAndOne} = \Big(\big(1(x) < x < (\mathit{CNN} \mid \mathit{BBC})\big) \mid \big(1(x) < x < (\mathit{FOX} \mid \mathit{DISNEY})\big) \Big)$$

the meaning of the base orchestrations are:

$$\begin{aligned} [\![(\mathit{CNN} \mid \mathit{BBC})]\!] &= \lfloor\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\rfloor \\ [\![(\mathit{FOX} \mid \mathit{DISNEY})]\!] &= \lfloor\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\rfloor. \end{aligned}$$

Therefore,

$$\begin{aligned} [\![(1(x) < x < (\mathit{CNN} \mid \mathit{BBC})))]\!] &= \lfloor\lfloor \texttt{cnn} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc} \rfloor\rfloor \\ [\![(1(x) < x < (\mathit{FOX} \mid \mathit{DISNEY})]\!] &= \lfloor\lfloor \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{disney} \rfloor\rfloor, \end{aligned}$$

and

$$\begin{aligned} [\![\mathit{OneAndOne}]\!] &= (\lfloor\lfloor \texttt{cnn} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc} \rfloor\rfloor) + (\lfloor\lfloor \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{disney} \rfloor\rfloor) \\ &= \lfloor\lfloor \texttt{cnn}, \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{cnn}, \texttt{disney} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc}, \texttt{fox} \rfloor\rfloor \sqcap \lfloor\lfloor \texttt{bbc}, \texttt{disney} \rfloor\rfloor \end{aligned}$$

. □

As we have seen, orchestration *OneTwo* in Example 4, different multi-sets in $[\![E]\!]$ can have different sizes. The following result shows which Orc subfamilies are guaranteed to produce outputs with a unique output length.

Theorem 2. *All the output streams produced by an execution of an* Orc *expression* E *in* ElementaryOrc, PruningOrc *or* IfOrc *have the same number of items denoted as* out(E).

Proof. The proof of follows from an inductive structural reasoning over the different allowed operations on orchestrations. We have structured it through several claims.

Claim 1. *Any orchestration E in* ElementaryOrc *publishes a unique output stream. Assuming that this output stream contains the values $v_1, \ldots, v_n$, then $\lfloor\lfloor E \rfloor\rfloor = \lfloor\lfloor v_1, \ldots, v_n \rfloor\rfloor$ (the case $\lfloor\lfloor E \rfloor\rfloor = \lfloor\lfloor\ \rfloor\rfloor$ corresponds to $n = 0$). The number of published outputs is well defined and* $\mathsf{out}(E) = n$.

The proof of this claim is straightforward in the case of ElementaryOrc because there is no indeterminism. Observe that the claim is true when E is a site call.

For the parallel composition $E_1 \mid E_2$ where $E_1 = \lfloor\lfloor v_1, \ldots, v_n \rfloor\rfloor$ and $E_2 = \lfloor\lfloor w_1, \ldots, w_m \rfloor\rfloor$, according to Equation (3) we have

$$[\![E_1 \mid E_2]\!] = \lfloor\lfloor v_1, \ldots, v_n, w_1, \ldots, w_m \rfloor\rfloor.$$

Therefore, the meaning is an unique multi-set and the number of outputs is well defined.

Consider the sequential composition $E_1 > x > E_2(x)$ with $E_1 = \lfloor\lfloor v_1, \ldots, v_n \rfloor\rfloor$ and $E_2(x) = \lfloor\lfloor w_1(x), \ldots, w_m(x) \rfloor\rfloor$, then from Equation (5) we have

$$[\![E_1 > x > E_2(x)]\!] = \lfloor\lfloor w_i(v_j) \mid w_i(x) \in [\![E_2(x)]\!], v_j \in [\![E_1]\!] \rfloor\rfloor,$$

and, again, the meaning is formed by a unique bag.

Claim 2. *For any orchestration E in* PruningOrc*, $\lfloor\lfloor E \rfloor\rfloor$ has a non-deterministic behaviour represented by a set of bags. All the bags in $\lfloor\lfloor E \rfloor\rfloor$ contain the same number of items.*

Observe that the claim is true in the case of site calls. For the other constructions assume that $[\![E_1]\!] = \sqcap_{1 \leq i \leq n_1} M_i$ and $\# M_i = k$, for $1 \leq i \leq n_1$, and that $[\![E_2]\!] = \sqcap_{| \leq j \leq n_2} M'_j$ and $\# M'_j = k'$, for $1 \leq j \leq n_2$. In the case of a parameter dependency we assume also that the size of a bag is independent of the value assigned to the parameter.

For the parallel composition $E_1 \mid E_2$, from (3) each pair i, j contributes to the meaning with the multiset $M_i + M'_j$ which has $k + k'$ elements. Thus all those multi-sets have the same cardinality.

For the sequential composition $E_1 > x > E_2(x)$. According to (4) each pair i, j contributes with the multi-set $\sum_{m \in M_i} M'_j(m)$ having $k\ k'$ items. As before, all the multi-sets have the same cardinality.

Consider now the pruning operation, that is $E_1(x) < x < E_2$. According to (5), for each i, j, each choice $m' \in M'_j$ provides the multi-set $M_i(m')$ with $\# M_i(m) = k$ and the claim follows.

Claim 3. *Any orchestration $E \in$* IfOrc *publishes a unique output stream thus having a well defined length.*

In this case, as in ElementaryOrc, there is no indeterminism and therefore any call to the if site is done in a deterministic context and their meaning is a unique multi-set. The presented case analysis concludes the proof. □ □

3.1 Computational problems:

For those Orc subfamilies in which the measure out is well defined, we consider the following computational problems:

> ExistOut: Given an Orc expression E, decide whether $\mathsf{out}(E) > 0$.
> Out: Given an Orc expression E compute $\mathsf{out}(E)$.

Let us analyse the complexity of both problems for ElementaryOrc, PruningOrc and IfOrc.

Theorem 3 ([11]). *The problems* ExistOut *and* Out *restricted to ElementaryOrc or PruningOrc can be solved in polynomial time.*

Proof. Let us start considering the case of ElementaryOrc orchestrations. In this case, the proof that both ExistOut and Out belong to P follows from the following recursive definition. $\mathsf{out}(0) = 0$, $\mathsf{out}(1) = 1$

$$\mathsf{out}(1(v)) = \begin{cases} 1 & \text{if the parameter } v \text{ is defined} \\ 0 & \text{otherwise} \end{cases}$$

Whenever E_1, E_2 are ElementaryOrc expressions it holds that

$$\mathsf{out}(E_1|E_2)) = \mathsf{out}(E_1) + \mathsf{out}(E_2)$$
$$\mathsf{out}(E_1 > z > E_2(z)) = \mathsf{out}(E_1)\mathsf{out}(E_2(z \neq \perp))$$

where $E_2(z \neq \perp)$ denotes the orchestration $E_2(z)$ when variable z is well defined. For instance given $E_2(x) = (\mathit{CNN} \mid \mathit{EmailDad}(x))$, the case $E_2(x \neq \perp)$ corresponds with a defined value for x and $\mathsf{out}(E_2(x \neq \perp)) = 2$.

Let us consider the number of operations needed to compute $\mathsf{out}(E)$. Given an Orc expression E, the number of operators is bounded by the size of the expression E. The Orc operators in E can be "|", $\gg$, or "$> x >$". A way to to compute $\mathsf{out}(E)$ consists to map the expression E into an *arithmetic expression*. We do that mapping the Orc operator operator "|" into the arithmetic operator"sum" and operators "$\gg$", "$> x >$" are mapped into "product". Finally, any site S is mapped into $\mathsf{out}(S)$. This arithmetic expression can be evaluated polynomially in the size of the expression E.

In the second place, let us consider the case of PruningOrc. In this case we have to extend the recursive definition to the additional operator.

$$\mathsf{out}(E_1(z) < z < E_2) = \begin{cases} \mathsf{out}(E_1(z \neq \perp)) & \text{if } \mathsf{out}(E_2) > 0 \\ \mathsf{out}(E_1(z = \perp)) & \text{if } \mathsf{out}(E_2) = 0 \end{cases}$$

where $E_1(z = \perp)$ denotes the orchestration $E_1(z)$ when variable z is undefined. For instance $E_1(x = \perp) = (\mathit{CNN} \mid \mathit{EmailDad}(x = \perp)) = (\mathit{CNN} \mid 0)$ and therefore $\mathsf{out}(E_1(x = \perp)) = 1$.

Let us consider the number of operations needed to compute $\mathsf{out}(E)$ in the case of PruningOrc. We need to deal with the asymmetric parallel composition operator the "$< x <$". Consider the syntactic tree corresponding to the Orc expression E. In such a tree, the nodes corresponds to the Orc operators. In the case of nodes corresponding to "$< x <$" we associate the variable x to this node. We are interested to know if $x = \perp$ or $x \neq \perp$. As we can have nested asymmetric parallel compositions like

$$E = \big(\cdots < x < \cdots\big(\cdots(\cdots < y < \cdots)\cdots\big)\cdots\big)$$

We evaluate the tree from the right to the left in order to get the value of the variables corresponding to the rightmost asymmetric parallel composition first. According to that we need to know if $y = \perp$ before to consider if $x = \perp$. This concludes the proof. □ □

In the following example we compute $\mathsf{out}(\mathit{Bang}_2)$ according to Theorem 3.

Example 7. *Consider a $\mathsf{ElementaryOrc}$ case. Let us continue with the Example 3 and compute $\mathsf{out}(\mathit{Bang}_2)$. We have*

$$\mathit{Bang}_2 = \Big(\big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) \gg \big(1(\mathtt{1}) \mid 1(\mathtt{1})\big)\Big) \gg \Big(\big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) \gg \big(1(\mathtt{1}) \mid 1(\mathtt{1})\big)\Big)$$

As $\mathsf{out}(1(\mathtt{1})) = 1$, we obtain the following arithmetic expression

$$\mathsf{out}(\mathit{Bang}_2)) = \big((1+1)\ (1+1)\big)\ \big((1+1)\ (1+1)\big) = 16 = 2^{2^2}.$$

In general $\mathsf{out}(\mathit{Bang}_n) = 2^{2^n}$. As the size of Bang_n is $O(2^{n+1})$ a polynomial number of bits, in relation to the size of Bang_n, are enough to compute the number of outputs.

Next, consider the following $\mathsf{PruningOrc}$ expression:

$$E = 1(x) < x < \Big(\big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) > y > \big((1(z) \mid (1(\mathtt{1})) < z < (1(t) < t < 0)\big)\Big)$$

We need to consider the variables appearing in asymmetric parallel composition in the order, t first, z second and x third. As $t = \perp$ we have to compute $\mathsf{out}(1(t = \perp))$. As $\mathsf{out}(1(t = \perp)) = 0$, we have $z = \perp$ and we compute $\mathsf{out}\big(1(z = \perp) \mid 1(\mathtt{1})\big) = 1$ and

$$\mathsf{out}\big[\big(1(\mathtt{1}) \mid 1(\mathtt{1})\big) > y > \big(1(z = \perp) \mid 1(\mathtt{1})\big)\big] = 2.$$

As the number of outputs is 2, the variable x is defined and, finally we have $\mathsf{out}(E) = \mathsf{out}\big(1(x \neq \perp)\big) = 1$. □

In order to prove complexity bounds for orchestrations in IfOrc we introduce some notation. When E has n symmetric parallel operators, any valid execution path can be identified with a *trace*, a string in $\mathtt{t} \in \{\mathtt{l}, \mathtt{r}\}^{\leq n}$. Assume that the i-th execution of a parallel operator corresponds to subexpressions $E_1 \mid E_2$. We codify the path following the call to E_1 with label $\mathtt{l}$ and the one following the call to E_2 with label $\mathtt{r}$. Thus, in the i-th position of the trace there will

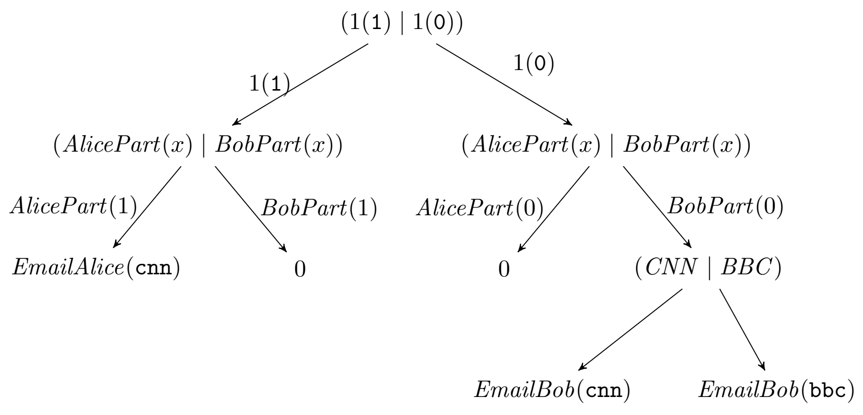

Figure 1: The valid execution paths of *SelectiveReaders*.

be a corresponding r or l symbol denoting which of the two suborchestrations has been executed. Observe that, for a given expression E, not all the strings in $\{\mathtt{l}, \mathtt{r}\}^{\leq n}$ describe valid execution paths. We note $\mathsf{paths}(E) \subseteq \{\mathtt{l}, \mathtt{r}\}^{\leq n}$ the set of traces corresponding to valid execution paths. Any execution path gives potentially one or zero values, for $\mathtt{t} \in \mathsf{paths}(E)$, $\mathsf{val}(\mathtt{t})$ denotes the value returned by this execution path. Observe that $\mathsf{val}(\mathtt{t}) = \lfloor\lfloor v \rfloor\rfloor$, when the execution path publishes a value v, or $\mathsf{val}(\mathtt{t}) = \lfloor\lfloor\ \rfloor\rfloor$, otherwise.

Example 8. *Consider the orchestration SelectiveReaders defined as follows:*

$$\begin{aligned}
&\mathit{SelectiveReaders} = (1(\mathtt{1}) \mid 1(\mathtt{0})) > x > (\mathit{AlicePart}(x) \mid \mathit{BobPart}(x)),\\
&\mathit{AlicePart}(x) = \mathit{if}(x) \gg \mathit{CNN} > y > \mathit{EmailAlice}(y),\\
&\mathit{BobPart}(x) = \mathit{if}(\neg x) \gg \mathit{NewsToBob}(x),\\
&\mathit{NewsToBob}(x) = (\mathit{CNN} \mid \mathit{BBC}) > z > \mathit{EmailBob}(z))
\end{aligned}$$

SelectiveReaders contains 3 *operators of type* $|$*, the corresponding valid execution paths are depicted in a tree-like form in Fig. 1 and we have that*

$$\mathsf{paths}(\mathit{SelectiveReaders}) = \{\mathtt{ll}, \mathtt{lr}, \mathtt{rl}, \mathtt{rrl}, \mathtt{rrr}\}.$$

For $\mathtt{t} = \mathtt{ll}$*, in the first two* $|$*, we take the left part thus leaving expression*

$$\begin{aligned}
&\big(1(\mathtt{1}) > x > \mathit{AlicePart}(x)\big)\\
&\quad = \big(1(\mathtt{1}) > x > \mathtt{if}(x) \gg \mathit{CNN} > y > \mathit{EmailAlice}(y)\big)
\end{aligned}$$

and thus $\mathsf{val}(\mathtt{ll}) = \lfloor\lfloor \mathtt{s\_aalice\_cnn} \rfloor\rfloor$*. When* $\mathtt{t} = \mathtt{rl}$*, we have*

$$\begin{aligned}
&\big(1(\mathtt{0}) > x > \mathit{AlicePart}(x)\big)\\
&\quad = \big(1(\mathtt{0}) > x > \mathtt{if}(x) \gg \mathit{CNN} > y > \mathit{EmailAlice}(y)\big)
\end{aligned}$$

as this path behaves as 0*, it returns nothing and we have* $\mathsf{val}(\mathtt{rl}) = \lfloor\lfloor\ \rfloor\rfloor$*.* □

The following result follows from the definitions.

Lemma 1. *Let E be an* IfOrc *expression. We can decide in polynomial time (in the size of E) whether* $\mathtt{t} \in \mathsf{paths}(E)$ *and, when* $\mathtt{t} \in \mathsf{paths}(E)$*, whether* $\mathsf{val}(t) \neq \lfloor\lfloor\ \rfloor\rfloor$*. Furthermore,* $[\![E]\!] = \sum_{\mathtt{t} \in \mathsf{paths}(\mathtt{E})} \mathsf{val}(\mathtt{t})$ *and* $\#\lfloor\lfloor E \rfloor\rfloor = \#\{\mathtt{t} \in \mathsf{paths}(E) \mid \mathsf{val}(\mathtt{t}) \neq \lfloor\lfloor\ \rfloor\rfloor\}$*.*

Our next result establishes the computational complexity for IfOrc.

Theorem 4. *The problem* $\mathsf{ExistOut}$ *is* NP*-complete and the problem* Out *is* $\#\mathsf{P}$*-complete when restricted to* IfOrc*.*

Proof. Recall that, from Lemma 1, given an Orc expression E having n parallel operators, each published value can be retrieved following a valid execution path described by $\mathtt{t} \in \{\mathtt{l}, \mathtt{r}\}^{\leq n}$. To check whether $\mathsf{out}(E) > 0$, we can guess a trace and check, in polynomial time, that it corresponds to a productive execution path. Thus, $\mathsf{ExistOut}$ belongs to NP. Observe that also we have $\#\lfloor\lfloor E \rfloor\rfloor = \#\{\mathtt{t} \in \mathsf{paths}(E) \mid \mathsf{val}(\mathtt{t}) \neq \lfloor\lfloor\ \rfloor\rfloor\}$ which proves that Out belongs to $\#\mathsf{P}$.

To prove hardness, we consider a reduction form the 3-SAT problem. Given a Boolean formula in 3CNF $F = C_1 \wedge \ldots \wedge C_m$ over n variables, $x_1, \ldots, x_n$, where

$$C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c},\ \{y_{1_a}, y_{1_b}, y_{1_c}\} \subseteq \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\},$$

for $1 \leq i \leq n$, are the clauses. Let us see how to encode a clause through an IfOrc expression. Assume that $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$, over variables $x_{i_a}, x_{i_b}, x_{i_c}$, define

$$E_{C_i}(x_{i_a}, x_{i_b}, x_{i_c}) = (\mathit{if}(y_{ia}) \mid (\mathit{if}(\neg y_{ia}) \gg (\mathit{if}(y_{ib}) \mid (\mathit{if}(\neg y_{ib}) \gg \mathit{if}(y_{ic}))))) \gg 1,$$

where double negations are eliminated. We associate to the formula F the following orchestration E_F:

$$E(x_1, \ldots, x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \gg \ldots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c})$$
$$E_F = (\mathit{True} \mid \mathit{False}) > x_1 > \cdots > (\mathit{True} \mid \mathit{False}) > x_n > E(x_1, \ldots, x_n)$$

Where, $\mathit{True} = 1(\mathtt{1})$ and $\mathit{False} = 1(\mathtt{0})$. Observe that a description of E_F can be computed in linear time in the size of F. Note that, for $(x_1, \ldots, x_n) \in \{\mathtt{0}, \mathtt{1}\}^n$, we have $[\![E(x_1, \ldots, x_n)]\!] = \lfloor\lfloor 1 \rfloor\rfloor$, when $F(x_1, \ldots x_n) = 1$, and $[\![E(x_1, \ldots, x_n)]\!] = \lfloor\lfloor\ \rfloor\rfloor$, otherwise. As $[\![x_i]\!] = \lfloor\lfloor \mathtt{1}, \mathtt{0} \rfloor\rfloor$, for $1 \leq i \leq n$, $[\![E_F]\!] = \lfloor\lfloor \underbrace{1, \ldots \ldots, 1}_{\#\{x \mid F(x)=1\}} \rfloor\rfloor$ and $\mathsf{out}(E_F) = \#\{x \mid F(x) = 1\}$. Therefore, the reduction works correctly for both problems and the theorem holds. □ □

3.2 Probabilistic environments and problems:

Now we assume a *crash* failure model in which a call to an external site S succeeds (produces an output) with probability p_S. In the next definition we adapt Definition 1 to the crash model. Observe that we have to keep only the probability of success p_S from which it can be derived the probability of crash.

Definition 2. *Given an orchestration E having calls to sites $\{S_1, \dots, S_n\}$, a* probabilistic out environment *for E is a set $\mathcal{P} = \{p_1, \dots, p_n\}$ where, for each $1 \le i \le n$, $0 \le p_i \le 1$ and p_i gives the probability that S_i produces an output.*

Example 9. *Assume that a call to the CNN site succeeds with probability p. that is probabilistic environment $\mathcal{P} = \{p\}$. Let us consider the expected number of outputs of TwiceCNN = CNN $\gg$ CNN in the probabilistic environment $\mathcal{P} = \{p\}$ under the oblivious and the stable models. In the* oblivious *model, both calls to the CNN are independent (we toss the coin two times) the probability to get an output is p^2, formally $\mathbb{E}^o_{\mathcal{P}}(\mathit{out}(\mathit{TwiceCNN}))) = p^2$. In the* stable *model, we just toss the coin once and $\mathbb{E}^s_{\mathcal{P}}((\mathit{out}(\mathit{TwiceCNN}))) = p$.* $\square$

Let us define the two computational problems associated to the probabilistic case. Given an an Orc expression E, a probabilistic out environment $\mathcal{P}$, and an a model $\alpha \in \{o, s\}$ where o denotes oblivious ans s stable models:

> PrPub: compute the probability of publication, i.e. $\Pr^\alpha_{\mathcal{P}}(\mathsf{out}(E) > 0)$.
> $\mathsf{ExpectedOut}$: compute $\mathbb{E}^\alpha_{\mathcal{P}}(\mathsf{out}(E))$.

First, let us consider the *oblivious* case. We extend the bag semantics to probabilistic environments. Let us start with $\mathsf{ElementaryOrc}$. For a site S publishing with probability p_s, let $q_\mathsf{s} = 1 - p_\mathsf{s}$, we define

$$[\![S(x_1, \dots x_n)]\!] = \begin{cases} \lfloor\!\lfloor s(v_1, \dots, v_n)\rfloor\!\rfloor @ p_\mathsf{s} \parallel \lfloor\!\lfloor\ \rfloor\!\rfloor @ q_\mathsf{s} & \text{if } (x_1, \dots x_n) = (v_1, \dots v_n) \\ \lfloor\!\lfloor\ \rfloor\!\rfloor @ 1 & \text{if } \exists i : 1 \le i \le n : x_i = \bot \end{cases} \tag{6}$$

Observe that we assume that the probability of success depends on the site not on the particular assignment to the variables.

Assuming that, for $\mathsf{ElementaryOrc}$ expressions E and E' or $E'(x)$ we have $[\![E]\!] = \big(\parallel_{1\le i\le k} M_i @ p_i\big)$ and $[\![E']\!] = \big(\parallel_{1\le j\le k'} M'_j @ p'_j\big)$ or $[\![E'(x)]\!] = \big(\parallel_{1\le j\le k'} M'_j(x) @ p'_j\big)$ where, for $1 \le i \le k$ and $1 \le j \le k'$, M_i and $M'_j(M'_j(x))$ are multi-sets, we define

$$[\![E \mid E']\!] = \Big(\parallel_{1\le i\le k} \parallel_{1\le j\le k'} (M_i + M'_j) @ p_i p'_j\Big) \tag{7}$$

$$[\![E > x > E'(x)]\!] = \Big(\parallel_{1\le i\le k} \parallel_{1\le j\le k'} \big(\sum_{m\in M_i} M'_j(m)\big) @ p_i (p'_j)^{\# M_i}\Big) \tag{8}$$

As a consequence, in the *oblivious* model, any E in $\mathsf{ElementaryOrc}$ in a probabilistic environment factorizes uniquely as

$$[\![E]\!] = \big(M_1 @ p_1 \parallel \cdots \parallel M_k @ p_k\big) = \big(\parallel_{1\le i\le k} M_i @ p_i\big)$$

where M_i, $1 \le i \le k$, are pairwise different bags and $\sum_k p_k = 1$. Note that each M_i models a possible output stream.

Recall that the *probability generating function* of random variable X, $g(X)(z)$ is a polynomial over a variable z, defined as

$$g(X)(z) = \hat{p}_0 + \hat{p}_1 z^1 + \cdots \hat{p}_{\ell_{\max}} z^{\ell_{\max}} = \sum_\ell \hat{p}_\ell z^\ell \tag{9}$$

where $\hat{p}_i = P(X = i)$. We will make use of two well known properties: $\Pr(X = 0) = g(X)(0)$ and $\mathbb{E}(X) = g'(X)(1)$ were $g'(X)$ is the derivative of $g(X)$. In order to analyse the probability that an Orc expression E publishes, we consider the generating function associated to the probability distribution of the random variable $\mathsf{out}(E)$. We use $g(E)$ as an abbreviation of $g(\mathsf{out}(E))$.

From the bag semantics of E, defining $\ell_{\max} = \max\{\#M_i \mid 1 \leq i \leq k\}$, we have that the probability of getting an output of length ℓ, $0 \leq \ell \leq \ell_{\max}$, is $\hat{p}_\ell = \sum_{\#M_i=\ell} p_i$. We provide the tools to evaluate $g(E)(0) = \Pr(\mathsf{out}(E) = 0)$ and $g'(E)(1) = \mathbb{E}(\mathsf{out}(E))$ without having to compute explicitely the coefficients of $g(E)$. From the the definitions and the extension of the bag semantics we have the following recursive expressions.

Lemma 2. *For local sites* 1 *and* 0, $g(1)(z) = z$ *and* $g(0)(z) = 1$. *For a site* S *publishing with probability* p_{s} *and remaining silent with probability* $q_{\mathsf{s}} = 1 - p_{\mathsf{s}}$, $g(S)(z) = p_{\mathsf{s}}z + q_{\mathsf{s}}$. *For* ElementaryOrc *expressions* E *and* F $(F(x))$, $g(E \mid F)(z) = g(E)(z)\ g(F)(z)$, $g(E \gg F)(z) = g(E)(g(F)(z))$, *when* $[\![E]\!] \neq \lfloor\!\lfloor\ \rfloor\!\rfloor$, $g(E > x > F(x))(z) = g(E)(g(F(x \neq \bot)(z))$.

Example 10. *Take* $BlockingCoin = (1(\mathtt{1})\ {}_{1/2}\!\oplus 0)$ *and define:*

$$ParBlockingCoin_n = \underbrace{(BlockingCoin \mid \cdots \mid BlockingCoin)}_{n\ \mathtt{times}}$$
$$ManyTosses_n = ParBlockingCoin_n \gg BlockingCoin$$

Remind that $g(BlockingCoin)(z) = q_{\mathtt{BlockingCoin}} + p_{\mathtt{BlockingCoin}}z = 1/2 + z/2$. *From this we get that* $g(ParBlockingCoin_n)(z) = (g(BlockingCoin)(z))^n$ *To compute* PrPub *we apply* $\Pr(\mathit{out}(Tosses_n) > 0) = 1 - g(ManyTosses_n)(0)$. *The generating function verifies*

$$\begin{aligned} g(Tosses_n)(0) &= g(ParBlockingCoin_n \gg BlockingCoin)(0) \\ &= g(ParBlockingCoin_n)(g(BlockingCoin)(0)) \\ &= g(ParBlockingCoin_n)(1/2) = \big(g(BlockingCoin)(1/2)\big)^n = (3/4)^n \end{aligned}$$

because $g(BlockingCoin)(0) = 1/2$ *and* $g(BlockingCoin)(1/2) = 3/4$. *Then we have that* $\Pr(\mathit{out}(Tosses_n) > 0) = 1 - (3/4)^n$. *As expected, the probability is close to* 1 *for* n *large* [5]. *Now consider the* Orc *expression*

$$ManyOutputs_n = \underbrace{((1(\mathtt{1}) \mid 1(\mathtt{1})) \gg \cdots \gg (1(\mathtt{1}) \mid 1(\mathtt{1})))}_{n\ \mathtt{times}}$$
$$ExpTosses_n = ManyOutputs_n \gg ParBlockingCoin_n$$

Observe that $g(ManyOutputs)(z) = z^{2^n}$. *Thus*

$$\begin{aligned} g(ExpTosses_n)(0) &= g(ManyOutputs_n \gg ParBlockingCoin_n)(0) \\ &= g(ManyOutputs_n)(g(ParBlockingCoin_n)(0)) \\ &= g((ManyOutputs_n)(1/2^n) = (1/2^n)^{2^n}. \end{aligned}$$

[5]We would like to warn the reader against the following, clearly false, "intuitive" approach: $\Pr(\mathsf{out}(Tosses_n) > 0) = Pr(\mathsf{out}(ParBlockingCoin_n) > 0)\ \Pr(\mathsf{out}(BlockingCoin_n) > 0) = ((1 - (1/2)^n)1/2 < 1/2$.

and $\Pr(\textsf{out}(\mathit{ExpTosses}_n) > 0) = 1 - (1/2^n)^{2^n} = (2^{n2^n} - 1)/2^{n2^n}$. *As* $2^{n2^n} - 1$ *is an odd number the fraction is irreducible and it requires* $n2^n$ *bits.* $\square$

In order to get complexity results we need to fix the representation of a probabilistic environment $\mathcal{P}$ for an orchestration E where $S_1, \ldots, S_n$ are the sites appearing in E. We use the standard representation for rational numbers. For any $1 \leq i \leq n$ we write explicitly the probability of success of S_i as a fraction of two natural numbers $p_i = x_i/y_i$, encoded as a pair (x_i, y_i). The length (or size) of $\mathcal{P}$ denoted as $|\ \mathcal{P}\ |$ is $O\big(\sum_{1\leq i\leq n}(\log x_i + \log y_i)\big)$. Observe that we have restricted ourselves to rational valued probabilities, which is needed in order to get an upperbound on the complexity of the considered problems. The hardness results, as usual in complexity theory when dealing with function problems, rely on the output size in the fixed representation.

Let us consider the ExpectedOut problem in the *oblivious* case. It is well known the relation between the expected number of outputs and the derivative g' (in relation to z) of the generating function. We have $\mathbb{E}(\textsf{out}(E)) = g'(E)(1)$.

Lemma 3. *For ElementaryOrc the following holds. Given a site S with probability of success p_{s}, $\mathbb{E}(\textsf{out}(S)) = p_{\mathsf{s}}$. For internal sites 1 and 0 we have $\mathbb{E}(\textsf{out}(1)) = 1$ and $\mathbb{E}(\textsf{out}(0)) = 0$. For the parallel composition, $\mathbb{E}(\textsf{out}(E \mid F)) = \mathbb{E}(\textsf{out}(E)) + \mathbb{E}(\textsf{out}(F))$. For the sequential composition, $\mathbb{E}(\textsf{out}(E \gg F)) = \mathbb{E}(\textsf{out}(E))\ \mathbb{E}(\textsf{out}(F))$. When $[\![E]\!] \neq \lfloor\!\lfloor\ \rfloor\!\rfloor$ we have $\mathbb{E}(\textsf{out}(E > x > F(x))) = \mathbb{E}(\textsf{out}(E))\ \mathbb{E}(\textsf{out}(F(x \neq \perp)))$.*

Proof. For parallel composition $g'(E \mid F)(z) = g'(E)(z)\ g(F)(z) + g(E)(z)\ g'(F)(z)$, as $g(F)(1) = g(E)(1) = 1$ we get $\mathbb{E}(\textsf{out}(E \mid F)) = g'(E)(1) + g'(F)(1) = \mathbb{E}(\textsf{out}(E)) + \mathbb{E}(\textsf{out}(F))$. For sequential composition, applying the chain rule we get $g'(E \gg F)(z) = g'(E)(g(F)(z))\ g'(B)(z)$ and therefore $\mathbb{E}(\textsf{out}(E \gg F)) = \mathbb{E}(\textsf{out}(E))\ \mathbb{E}(\textsf{out}(F))$. When $[\![E]\!] \neq \lfloor\!\lfloor\ \rfloor\!\rfloor$ we have $\mathbb{E}(\textsf{out}(E > x > F(x))) = \mathbb{E}(\textsf{out}(E))\ \mathbb{E}(\textsf{out}(F(x \neq \perp)))$. $\square$ $\square$

Let us continue with Example 10.

Example 11. *We compute the expected number of outputs of $\mathit{ManyTosses}_n$ and $\mathit{ExpTosses}_n$ based on their syntactic structure.*

$$\begin{aligned}\mathbb{E}(\textsf{out}(\mathit{ManyTosses}_n)) &= \mathbb{E}(\textsf{out}(\mathit{ParBlockingCoin}_n))\ \mathbb{E}(\textsf{out}(\mathit{BlockingCoin})) \\ &= \mathbb{E}(\textsf{out}(\mathit{ParBlockingCoin}_n))/2 = n\,\mathbb{E}(\textsf{out}(\mathit{BlockingCoin}))/2 = n/4\end{aligned}$$

Let us consider $\mathit{ExpTosses}_n$. As $\mathbb{E}\big(\textsf{out}(1(1))\big) = 1$ we have $\mathbb{E}\big(\textsf{out}(1(1) \mid 1(1)\big) = 2$ and $\mathbb{E}\big(\textsf{out}(\mathit{ManyOutputs}_n)\big) = 2^n$, as before $\mathbb{E}\big(\textsf{out}(\mathit{ParBlockingCoin}_n)\big) = n/2$. Finally $\mathbb{E}\big(\textsf{out}(\mathit{ExpTosses}_n)\big) = n2^{n-1}$. The number of bits required to write the expectation is, in this case, polynomial with respect to the size of $\mathit{ExpTosses}_n$ expression. $\square$

Theorem 5. *In the* oblivious *model restricted to ElementaryOrc, the problem* PrPub *requires exponential space while the* ExpectedOut *problem can be solved in polynomial time.*

Proof. The analysis of the $\mathit{ExpTosses}_n$ given in Example 10 shows that computing the probability of publications requires an exponential number of bits with respect to the size of the given expression. This occurs because

$$\Pr(\mathsf{out}(\mathit{ExpTosses}_n) > 0) = (2^{n2^n} - 1)/2^{n2^n}$$

is an irreducible fraction and therefore the encoding as a pair $(2^{n2^n} - 1, 2^{n2^n})$ needs exponential space.

Let us consider the complexity bounds of the recursive approach suggested in Lemma 3. Let $S_1, \ldots, S_n$ the sites appearing in $\mathcal{P}$. For any S_i the probability of success is $p_i = x_i/y_i$, encoded as a pair (x_i, y_i). Define $b = \max\{\log x_i, \log y_i \mid 1 \le i \le n\}$. Each p_i is written as a pair (x_i, y_i) needing each one at most b bits. As usual the $p_i + p_j$ is described by the pair $(x_iy_j + x_jy_i, y_iy_j)$ and $p_i\ p_j$ is given by (x_ix_j, y_iy_j). Remind that to multiply 2 numbers of at most b bits $2b$ bits are enough. To sum two numbers of at most $2b$ bits we need at most $2b + 1$. Therefore, $p_i + p_j$ can be encoded with a pair needing $2b + 1$ bits for the first component and $2b$ bits by the second. In the case of a product $p_i\ p_j$ the encoding pair is (x_ix_j, y_i, y_j) and $2b$ bits are enough for each component. Taking $2(b + 1)$ we have a common bound for both, sums and products. In order to compute the expectations, consider the syntactic tree associated to the expression E (unfolding the recursive calls until calls to a sites). Remark that, as we are working with Orc expressions, the size of the syntactic tree is similar to the size of the expression. This tree has operators "$|$", "$\gg$" and "$> x >$" as internal nodes. This tree can be directly transformed into a tree corresponding to an arithmetic expression having fractions (the success probabilities) as leaves. As in the case of out '$|$" is mapped into "sum" and the sequential operators "$\gg$" , "$> x >$" into "product". Leaves correspond to site's expectation, that is $\mathbb{E}(S_i) = x_i/y_i$.

The number of operations (sums or products) is bounded by the size of E denoted as $\mid E \mid$ We have to add or multiply at most $\mid E \mid$ times fractions of b bits. Just taking into account a bound to the number of bits needed to perform an operation we have the following. The operation between 2 leaves need at most $2b+1$ bits. The operation between 4 leaves needs $2(2b+1)+1$. In the case of 8 leaves we need $2\big(2(2b + 1) + 1\big) + 1 = 2^3b + 2^3 - 1 \le 2^3(b + 1) = 8(b + 1)$. An upper bound to the number of bits needed to perform any operation is

$$(b + 1) \mid E \mid \le (\mid \mathcal{P} \mid) + 1) \mid E \mid = O\big((\mid E \mid + \mid \mathcal{P} \mid)^2\big)$$

and the expectations can be computed in polynomial space. To prove a polynomial time bound is straightforward. The number of operations is upper bounded by $\mid E \mid$. The most time consuming operation is a product of two numbers of at most $O\big((\mid E \mid + \mid \mathcal{P} \mid)^2\big)$ bits. This can be done in time $O\big((\mid E \mid + \mid \mathcal{P} \mid)^4\big)$. This give us a time bound $O\big(\ \mid E \mid (\mid E \mid + \mid \mathcal{P} \mid)^4)\big) = O\big(\ \mid E \mid + \mid \mathcal{P} \mid\ )^{O(1)}\big)$. $\square$ $\square$

By reductions from the #MONOTONE 2-SAT problem which is known to be #P-hard [27] we get the following result.

Theorem 6. *In the* oblivious *model, the problem* ExpectedOut *is #P-hard when restricted to PruningOrc or IfOrc expressions.*

The proof of Theorem 6 is done in two separate lemmas. We provide a separate hardness proof for each of the different subfamilies of orchestrations.

Lemma 4. *For PruningOrc expressions in the oblivious model, the* ExpectedOut *problems is #P-hard.*

Proof. We provide a reduction from the #MONOTONE 2-SAT problem: Given a MONOTONE 2-SAT formula $F = C_1 \wedge \cdots \wedge C_m$ with n variables $x_1, \ldots, x_n$ where $C_i = y_{i_a} \vee y_{i_b}$, $\{y_{i_a}, y_{i_b}\} \subseteq \{x_1, \ldots, x_n\}$ for $1 \le i \le m$, compute the number of satisfying assignments of F. We define the orchestration E_F as follows:

$$\begin{aligned}
&\mathit{BlockingCoin} = (1(\mathtt{1})\ {}_{1/2}{\oplus}\ 0)\\
&\text{for } 1 \le i \le m \text{ do } E_{C_i}(y_{i_a}, y_{i_b}) = \big(1(y_{i_a}) | 1(y_{i_b})\big)\\
&E(x_1, \ldots, x_n) = 1(s) < s < (E_{C_1}(y_{1_a}, y_{1_b}) \gg \cdots \gg E_{C_m}(y_{m_a}, y_{m_b}))\\
&E_F = (\cdots (E(x_1, \ldots, x_n) < x_1 < \mathit{BlockingCoin}) < x_2 < \cdots) < x_n < \mathit{BlockingCoin}
\end{aligned}$$

Observe that E_F can be computed in polynomial time in the size of F. Given $v = (v_1, \ldots v_n)$ with $v_i \in \{1, \perp\}$, we associate a Boolean interpretation to the tuple, $b((v_1, \ldots v_n) = (b(v_1), \ldots b(v_n))$ where, for $1 \le i \le n$,

$$b(v_i) = \begin{cases} 1 & \text{if } v_i = 1, \\ 0 & \text{if } v_i = \perp. \end{cases}$$

Using this association we have that

$$[\![E(v_1, \ldots, v_n)]\!] = \begin{cases} \lfloor\lfloor 1 \rfloor\rfloor & \text{if } F(b(v_1), \ldots b(v_n)) = 1, \\ \lfloor\lfloor\ \rfloor\rfloor & \text{otherwise.} \end{cases}$$

For a fixed $(v_2, \ldots, v_n)$, let us compute the meaning of $E(x_1, v_2, \ldots, v_n) < x_1 < \mathit{BlockingCoin}$. As $[\![x_1]\!] = [\![\mathit{BlockingCoin}]\!] = (\lfloor\lfloor 1 \rfloor\rfloor @1/2 \parallel \lfloor\lfloor\ \rfloor\rfloor @1/2)$, thus we get

$$\begin{aligned}
&[\![E(x_1, v_2, \ldots, v_n) < x_1 < \mathit{BlockingCoin}]\!]\\
&\qquad = ([\![E(1, v_2, \ldots, v_n)]\!] @1/2 \parallel [\![E(\perp, v_2, \ldots, v_n)]\!] @1/2)
\end{aligned}$$

Assuming $(B@p_1 \parallel B@p_2 \parallel \cdots) = (B@(p_1 + p_2) \parallel \cdots)$ and defining $p = \#\{x \mid F(x_1, \ldots x_n) = 1\}/2^n$, we have

$$[\![E_F]\!] = \Big(\parallel_{(v_1, \ldots, v_v) \in \{1, \perp\}^n} [\![E(v_1, \ldots, v_n)]\!] @(1/2)^n \Big) = \big(\lfloor\lfloor 1 \rfloor\rfloor @p \parallel \lfloor\lfloor\ \rfloor\rfloor @(1-p)\big).$$

From this last equality it follows that $\mathbb{E}(E_F) = \big(\sum_{x \in \{0,1\}^n} F(x)\big)/2^n$. Therefore, $\mathbb{E}(E_F)2^n$ is the solution to the #MONOTONE 2-SAT problem. □ □

Lemma 5. *For IfOrc expressions in the oblivious model, the* ExpectedOut *problems is #P-hard.*

Proof. To deal with hardness we consider the following variation of the reduction given in Theorem 4:

$$\begin{aligned}
&\mathit{BoolCoin} = \big(1(\mathtt{1})\ {}_{1/2}{\oplus}\ 1(\mathtt{0})\big)\\
&E(x_1,\ldots,x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \gg \cdots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c})\\
&E_F = \mathit{BoolCoin} > x_1 > \cdots > \mathit{BoolCoin} > x_n > E(x_1,\ldots,x_n)
\end{aligned}$$

It holds that $\mathbb{E}(\mathsf{out}(E_F)) = \{x \mid x \in \{0,1\}^n \text{ such that } F(x) = 1\}/2^n$ and the #P-hardness of ExpectedOut follows. □ □

In the *stable model*, an orchestration E with calls to n different sites $\{S_1,\ldots S_n\}$ and probabilistic out environment $\mathcal{P} = (p_1,\ldots,p_n)$, in an execution, follows a *success profile* $\mathsf{s} = s_1\cdots s_n \in \{0,1\}^n$. In such a profile, $s_i = 1$ means that all the calls to site S_i succeed and $s_i = 0$ means that all the calls to site S_i fail. According to $\mathcal{P}$ a success profile s occurs with probability $\Pr(\mathsf{s}) = \prod_{i=1}^{n}\big(p_i\cdot s_i+(1-p_i)\cdot(1-s_i)\big)$. Let us call $E_{|\mathsf{s}}$ the orchestration obtained after replacing all occurrences of sites S_i having $s_i = 0$ with site 0. We can compute PrPub as $\sum_{\{\mathsf{s}\in\{0,1\}^n|\mathsf{out}(E_{|\mathsf{s}}>0)\}}\Pr(\mathsf{s})$ and ExpectedOut as $\sum_{\mathsf{s}\in\{0,1\}^n}\Pr(\mathsf{s})\mathsf{out}(E_{|\mathsf{s}})$.

Example 12. *Let us consider the expressions $\mathit{ManyTosses}_n$ and $\mathit{ExpTosses}_n$ (see Example 10) in the* stable *model. Remind that:*

$$\begin{aligned}
&\mathit{BlockingCoin} = (1(\mathtt{1})\ {}_{1/2}{\oplus}\ 0)\\
&\mathit{ParBlockingCoin}_n = \underbrace{(\mathit{BlockingCoin} \mid \cdots \mid \mathit{BlockingCoin})}_{n\ \mathtt{times}}\\
&\mathit{ManyTosses}_n = \mathit{ParBlockingCoin}_n \gg \mathit{BlockingCoin}
\end{aligned}$$

The probabilistic behaviour is given by $S_1 = \mathit{BlockingCoin}$, and $\mathcal{P} = (p_1) = (1/2)$. The success profile contains just one site, $\mathsf{s} = s_1 \in \{0,1\}$. Consider the successful case $\mathsf{s} = 1$, then $\mathit{BlockingCoin}_{|\mathsf{s}=1} = 1(\mathtt{1})$ and

$$\begin{aligned}
\big(\mathit{ParBlockingCoin}_n\big)_{|\mathsf{s}=1} &= \underbrace{(1(\mathtt{1}) \mid \cdots \mid 1(\mathtt{1}))}_{n\ \mathtt{times}}\\
\big(\mathit{ManyTosses}_n\big)_{|\mathsf{s}=1} &= \underbrace{(1(\mathtt{1}) \mid \cdots \mid 1(\mathtt{1}))}_{n\ \mathtt{times}} \gg 1(\mathtt{1})
\end{aligned}$$

and $\mathsf{out}\Big(\big(\mathit{ManyTosses}_n\big)_{|\mathsf{s}=1}\Big) = n$. *When $\mathsf{s} = s_1 = 0$ the expression $\mathit{ManyTosses}_n$ completelly fails and therefore behaves like 0 site, $\big(\mathit{ManyTosses}_n\big)_{|\mathsf{s}=0} = 0$. The* PrPub *is $1/2$ and* ExpectedOut *is $n/2$. In the case of $\mathit{ExpTosses}_n$ we have:*

$$\begin{aligned}
&\big(\mathit{ExpTosses}_n\big)_{|\mathsf{s}=0}\\
&\quad = \underbrace{((1(\mathtt{1}) \mid 1(\mathtt{1})) \gg \cdots \gg (1(\mathtt{1}) \mid 1(\mathtt{1})))}_{n\ \mathtt{times}} \gg \underbrace{(0 \mid \cdots \mid 0)}_{n\ \mathtt{times}} = 0\\
&\big(\mathit{ExpTosses}_n\big)_{|\mathsf{s}=1}\\
&\quad = \underbrace{((1(\mathtt{1}) \mid 1(\mathtt{1})) \gg \cdots \gg (1(\mathtt{1}) \mid 1(\mathtt{1})))}_{n\ \mathtt{times}} \gg \underbrace{(1(\mathtt{1}) \mid \cdots \mid 1(\mathtt{1}))}_{n\ \mathtt{times}}
\end{aligned}$$

The PrPub *is also* $1/2$ *and* ExpectedOut *is* $(1/2)n2^n$. □

We summarize our complexity results for the stable model in the following theorem:

Theorem 7. *In the* stable *model, the problem* PrPub *is #P-hard when restricted to ElementaryOrc. Both problems,* PrPub *and* ExpectedOut*, are #P-hard when restricted to PruningOrc or IfOrc. Both problems,* PrPub *and* ExpectedOut*, belongs to PSPACE when restricted to ElementaryOrc, PruningOrc or IfOrc expressions.*

Next, we provide a proof through several lemmas.

Lemma 6. *In the stable model, the problems* PrPub *and* ExpectedOut *belong to PSPACE when restricted to ElementaryOrc, PruningOrc or IfOrc.*

Proof. Let us consider the ElementaryOrc case. In the case of the PrPub problem we need to compute $\sum_{\{\mathsf{s}\in\{0,1\}^n \mid \mathsf{out}(E_{|\mathsf{s}}>0)\}} \Pr(\mathsf{s})$. Let b the number of bits needed to encode each of the integers in the probabilities given in $\mathcal{P}$. Any probability $\Pr(\mathsf{s})$ can be encoded with a pair of numbers of $n2(b+1)$ bits. In the case of PrPub we need to sum at most 2^n such fractions, but this can be done allocating $O(n)$ extra bits and we keep in polynomial space. In the case of expectations, we need to compute $\sum_{\mathsf{s}\in\{0,1\}^n} \Pr(\mathsf{s})\mathsf{out}(E_{|\mathsf{s}})$. As $\mathsf{out}(E_{|\mathsf{s}})$ can be computed in polynomial time we keep the whole computation in polynomial space. The other cases are similar. □ □

Lemma 7. *For ElementaryOrc expressions in the stable model, the problem* PrPub *is #P-hard.*

Proof. Let us consider the following reduction from #MONOTONE 2-SAT. Given a MONOTONE 2-SAT formula $F = C_1 \wedge \cdots \wedge C_m$ with n variables $x_1, \ldots, x_n$ where $C_i = y_{i_a} \vee y_{i_b}$, $\{y_{i_a}, y_{i_b}\} \subseteq \{x_1, \ldots, x_n\}$ for $1 \leq i \leq m$. Let us define the orchestration E_F as follows. First, define $X_i = (1(1)\ {}_{1/2}\!\oplus 0)$, for $1 \leq i \leq n$, second, for each $1 \leq i \leq m$, define $E_{C_i} = \big(X_{i_a} | X_{i_b}\big)$ and finally, set the expression $E_F = E_{C_1} \gg \cdots \gg E_{C_m}$. Given $\mathsf{s} = s_1 \cdots s_n$ by construction $\mathsf{out}\big((E_F)_{|\mathsf{s}}\big) > 0)$ iff $F(\mathsf{s}) = 1$. As for any $\mathsf{s} \in \{0,1\}^n$, $\Pr(\mathsf{s}) = 2^{-n}$ it holds $\Pr(\mathsf{out}(E_F) > 0) = \sum_{\mathsf{s}\in\{0,1\}^n} \Pr\big(\mathsf{out}\big((E_F)_{|\mathsf{s}}\big) > 0\big) = 2^{-n} \sum_{\mathsf{s}\in\{0,1\}^n} F(\mathsf{s})$. □ □

Lemma 7 proves #P-hardness for the simplest family and therefore the #P-hardness for PrPub restricted to IfOrc or PruningOrc is inherited from ElementaryOrc. In the same way, for ExpectedOut we inherit the #P- hardness of Out for IfOrc, because a probabilistic environment includes the non-faulty scenarios and we conclude the proof of Theorem 7.

4 First item delay

We analyse now for which Orc families the delay needed to get the first published value is a well defined measure. We assume that delays are *measured within* *milliseconds*, and therefore they are non-negative integers. Symbol ω means "infinite" delay or "never". As usual that $\delta+\omega = \omega+\omega = \omega$ and $\min\{\delta, \omega\} = \delta$.

According to the constant delay hypothesis each site S (different from 0) has an associated finite delay δ_S. Note that in orchestrations with pruning the delay of the *first item* is not always unique as it is shown in the following example.

Example 13. *Depends* $= \big(\mathit{if}(b) \gg \mathit{CNN} \mid \mathit{if}(\neg b) \gg \mathit{BBC}\big) < b < \big(1(\mathtt{1}) \mid 1(\mathtt{0})\big)$ *and assume that* $\delta_{CNN} \neq \delta_{BBC}$*. The delay of* $1(x \neq \bot)$ *is* δ_1 *independently of the value taken by* x*. The expression Depends has a non-deterministic behaviour, it returns* `cnn` *with delay* $\delta_{CNN} + \delta_1$ *or* `bbc` *with delay* $\delta_{BBC} + \delta_1$*.* $\square$

We introduce a refinement of our semantics, a timed version of the fully defined variables model. Assume that an orchestration E publishes a stream $v_1, v_2, \ldots, v_n$, where each value v_i is published with delay δ_i after E is called. When time is abstracted, we have noted this behaviour as $\lfloor\!\lfloor v_1, v_2, \ldots, v_n \rfloor\!\rfloor$. If we take into account the different publication times we write $\lfloor\!\lfloor \langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle \rfloor\!\rfloor$. We call such a bag a *timed multiset*. Observe that, under this semantics, the behaviour of 0 is again described by $\lfloor\!\lfloor\rfloor\!\rfloor$. Taking into account the constant-delay hypothesis the behaviour of $S(x_1, \ldots, x_n)$ is $\lfloor\!\lfloor \langle S(v_1, \ldots, v_n) : \delta_S\rangle \rfloor\!\rfloor$ when all variables are defined ($x_i = v_i$, for $1 \le i \le n$) and $\lfloor\!\lfloor\rfloor\!\rfloor$ otherwise. Let us provide the meanings associated to the different syntactic constructions in some particular cases. For the internal sites, we have:

$$[\![0]\!] = \lfloor\!\lfloor\ \rfloor\!\rfloor \quad [\![1(x)]\!] = \begin{cases} \lfloor\!\lfloor \langle v : \delta_1\rangle \rfloor\!\rfloor & \text{if } x = v \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & \text{if } x = \bot \end{cases} \quad [\![\mathit{if}(b)]\!] = \begin{cases} \lfloor\!\lfloor \langle \mathtt{1} : \delta_{\mathit{if}}\rangle \rfloor\!\rfloor & \text{if } b = \mathtt{true} \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & \textit{otherwise} \end{cases} \tag{10}$$

Assuming that a site call $S(x_1, \ldots x_n)$ with assigned parameters $(v_1, \ldots v_n)$ returns $s(v_1, \ldots v_n)$ and the non-blocking and time-delay hypothesis, we have

$$[\![S(x_1, \ldots x_n)]\!] = \begin{cases} \lfloor\!\lfloor \langle s(v_1, \ldots, v_n) : \delta_S\rangle \rfloor\!\rfloor & \text{if } (x_1, \ldots x_n) = (v_1, \ldots v_n) \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & \text{if } \exists i : 1 \le i \le n : v_i = \bot \end{cases} \tag{11}$$

In the following we describe the timed multi-set decomposition corresponding to the Orc operators assuming that the meaning of the component orchestrations is described by a non deterministic choice of timed multi-sets. Let E_1 and E_2 be two orchestrations. We assume that $[\![E_1]\!] = \sqcap_{1 \le i \le n_1} M_i$ and $[\![E_2]\!] = \sqcap_{\le j \le n_2} M'_j$ for some timed multisets $M_1, \ldots, M_{n_1}, M'_1, \ldots, M'_{n_2}$. As before when the orc expressions are parameterized we assume that the bags in the decomposition are parameterized. The parallel composition executes both subexpressions in parallel, therefore we have

$$[\![(E_1 \mid E_2)]\!] = [\![E_1]\!] + [\![E_2]\!] = \sqcap_{1 \le i \le n_1} \sqcap_{1 \le j \le n_2} (M_i + M'_j). \tag{12}$$

To express the meaning of the sequential composition $E = E_1 > x > E_2(x)$ we need additional definitions. Given $M = \lfloor\!\lfloor \langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle \rfloor\!\rfloor$ and a delay δ we note

$$\delta \oplus M = \lfloor\!\lfloor \langle v_1 : \delta + \delta_1\rangle, \langle v_2 : \delta + \delta_2\rangle, \ldots, \langle v_n : \delta + \delta_n\rangle \rfloor\!\rfloor$$

As $[\![E_1]\!] = \sqcap_i M_i$, in the fully defined variables approach $[\![x]\!] = \sqcap_i M_i$. Assuming that a pair $\langle v : \delta\rangle \in M_i$ launches a call $E_2(x)$, at some time δ, it will generate $M'_j(v)$ with a delay of δ, i.e., $\delta \oplus M'_j(v)$. The output associated to elements in M_i belong to the same multibag, thus providing a timed multibag $\sum_{\langle v:\delta\rangle \in M_i} \delta \oplus M'_j(v)$ for each possible selection of j. As each pair (i, j) indexing $(M_i, M'_j(x))$ determines a possible multibag in the non-deterministic choice we have:

$$[\![E_1 > x > E_2(x)]\!] = \sqcap_{1\le i\le n_1} \sqcap_{1\le j\le n_2} \left(\sum_{(v:\delta)\in M_i} \delta \oplus M'_j(v) \right) \qquad (13)$$

Let us consider the asymmetric parallelism $E_1(x) < x < E_2$. Firstly, we need to introduce another operation the *first arrivals* of a timed multi-set $M = \lfloor\!\lfloor \langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle \rfloor\!\rfloor$ which is defined as

$$\mathsf{first}(M) = [\![(v : \delta) \in M \mid \delta = \min(\delta_1, \ldots, \delta_n)]\!]$$

Observe that $\#\mathsf{first}(M)$ can be greater than 1 as M can have different values arriving at the same minimum time. In the case of an asymmetric parallelism composition $E_1(x) < x < E_2$, the meaning of the variable x is the non-deterministic choice among the first arrivals of E_2. When $[\![E_2]\!] = \sqcap_{1\le j\le n_2} M'_j$ we have

$$[\![x]\!] = \sqcap_{1\le j\le n_2} \big(\sqcap_{\langle v,\delta\rangle \in \mathsf{first}(M'_j)} [\![\langle v : \delta\rangle]\!] \big).$$

Note that $[\![x]\!]$ may contain many different time values. Assuming again that $[\![E_1(x)]\!] = \sqcap_{1\le i\le n_1} M_i(x)$, then

$$[\![E_1(x) < x < E_2]\!] = \sqcap_{1\le i\le n_1} \sqcap_{\langle v:\delta\rangle \in [\![x]\!]} \delta \oplus M_i(v) \qquad (14)$$

As for Theorem 1, the proof of the following result follows from the previous results and structural induction.

Theorem 8. *In a timed fully defined variables semantics, an* Orc *expression E verifies $[\![E]\!] = [\![\ ]\!]$ or has a unique finite decomposition $[\![E]\!] = \sqcap_i M_i$ where M_i is a timed multiset.*

Given $M = \lfloor\!\lfloor \langle v_1 : \delta_1\rangle, \ldots, \langle v_n : \delta_n\rangle \rfloor\!\rfloor$, we define the associated *delays-multiset*, $\Delta(M) = \lfloor\!\lfloor \delta_1, \ldots, \delta_n \rfloor\!\rfloor$. Timed multisets M_1 and M_2 are called *time-equivalent* iff $\Delta(M_1) = \Delta(M_2)$. The following result identifies the families of Orc where the measure first can be defined. In the preceding constructions it is shown that, when E belongs to ElementaryOrc or IfOrc, $[\![E]\!]$ has a unique timed bag and when E belongs to PruningOrc, all the timed bags in $[\![E]\!]$ are time-equivalent.

Theorem 9. *For an* Orc *expression E in* ElementaryOrc, PruningOrc *or* IfOrc, *the delay of the first published output is well defined and we note this quantity as* first(E).

4.1 Computational problems

Given an Orc expression E with $S_1, \ldots, S_n$, $\delta_{S_1}, \ldots, \delta_{S_n}$ we consider the following computational problems associated with the first measure:

> BoundFirst: Given an integer k decide whether $\mathsf{first}(E) < k$.
> First: Compute $\mathsf{first}(E)$.

We denote $\mathsf{first}(0) = \omega$. According to the constant delay hypothesis:

$$\mathsf{first}(S(x_1, \ldots, x_n)) = \begin{cases} \delta_S & \text{if } x_i \neq \bot \text{ for } 1 \leq i \leq n \\ \omega & \text{otherwise.} \end{cases}$$

Let us start considering the complexity in the case of PruningOrc expressions.

Lemma 8. *The problems* BoundFirst *and* First *are solvable in polynomial time when restricted to* PruningOrc.

Proof. We show how to compute efficiently the first measure when pruning is allowed. For doing so we have to keep a timed context for variables. The context will keep track of the time at which a variable will be defined in the evaluation of a sub-expression. Within a context, time will be relative to the time at which the sub-expression is called. A *context* for a set of variables X is a mapping from the variables in X to a delay t. The time delay associated to a variable indicates the time at which the variable will be defined. We use functional notation, so that $C(x)$ means the delay associated to x in context C. We use the notation $\mathsf{first}_C(E)$ to denote the first delay item of expression E assuming that it is evaluated within context C. From the time-delay hypothesis, for a site call $S(X)$ executed within context C, as the site has to wait until all variables are defined, we have that $\mathsf{first}_C(S(X)) = \delta_S + \max_{x \in X} C(x)$.

In order to show the recursion that allow us to compute the measure first in polynomial time we need two additional operations for contexts. Let C be a context for variable set X

- For $\delta > 0$, $C + \delta$ denotes the context obtained from C when the initial time is advanced to the time step δ. The new context is defined over the same set X of variables. For $x \in X$, $(C + \delta)(x) = \max\{0, C(x) - \delta\}$.
- For a pair z, δ with $z \notin X$, $C[z \leftarrow \delta]$ denotes the context in which a new variable time pair is added. Formally, $C[z \leftarrow \delta]$ is defined on the set $X \cup \{z\}$ with $C[z \leftarrow \delta](x) = C(x)$ for $x \in X$ and $C[z \leftarrow \delta](z) = \delta$.

The initial context C for the complete orchestration E will be empty. Let us analyse the recursive steps depending on the construction.

- *Parallel composition.* When evaluating $A \mid B$ for a context C, A and B are evaluated in parallel in separate threads, each within context C. The full expression publishes the values published by the two sub-expressions, therefore $\mathsf{first}_C(A \mid B) = \min\{\mathsf{first}_C(A), \mathsf{first}_C(B)\}$.

- *Pruning.* When evaluating $A(x) < x < B$, for a context C, A and B are evaluated in parallel in separate threads, each within context C. When B publishes its first result, it is assigned to x on A and all the threads of B are killed. If some threads of A are waiting for x to have value, they are resumed. The full expression publishes the values published by the sub-expression A. Therefore, $\mathsf{first}_C(A(x) < x < B) = \mathsf{first}_{C[x \leftarrow \mathsf{first}_C(B)]}(A)$.

- *Sequential composition.* When evaluating $A > x > B(x)$ within a context C, for each value-time pair $\langle v : \delta \rangle$ published by A, we evaluate $B(x)$ at time δ. The results published by B are the results published by the full expression. We have to take into account the additional delay due to the elapsed time before the call to B is issued and readjust the context to this initial time. Thus, we have that

$$\mathsf{first}_C(A > x > B(x)) = \mathsf{first}_C(A) + \mathsf{first}_{(C+\mathsf{first}_C(A))[x \leftarrow 0]}(B).$$

Observe that this simple recursion allows us to compute $\mathsf{first}(E)$ in polynomial time and the lemma follows. □ □

The proof of Lemma 8 gives us a method to compute first. We develop this approach in the following example.

Example 14. *Let us continue with the Example 2, remind*

$$\begin{aligned}
&\mathit{TwoNews} = (\mathit{CNN} \mid \mathit{BBC})\\
&\mathit{MyEmails}(a,x) = (\mathit{EmailDad}(x) \mid \mathit{Email}(a,x))\\
&\mathit{SendNews}(a) = (\mathit{TwoNews} > x > \mathit{MyEmails}(a,x))\\
&\mathit{Addresses} = (\mathit{AddressAlice} \mid \mathit{AddressBob})\\
&\mathit{MyNews} = (\mathit{SendNews}(a) < a < \mathit{Addresses})
\end{aligned}$$

or displaying as Orc *expression in order to get a clear view of the variables x and a positions:*

$$\begin{aligned}
\mathit{MyNews} \quad = \quad & \big((\mathit{CNN} \mid \mathit{BBC}) > x > (\mathit{EmailDad}(x) \mid \mathit{Email}(a,x))\big)\\
& < a < (\mathit{AddressAlice} \mid \mathit{AddressBob})
\end{aligned}$$

First, we follow an intuitive approach. When MyNews is launched, two threads are spawn to deal in parallel with $\mathit{SendNews}(a)$ and Addresses. The variables x and a become defined respectively at times

$$C(x) = \min\{\delta_{CNN}, \delta_{BBC}\},\ C(a) = \min\{\delta_{AddressAlice}, \delta_{AddressBob}\}$$

Let us consider the publication times of sites in $\mathit{MyEmails}(a,x)$. Once x becomes defined $\mathit{EmailDad}(x)$ takes $\delta_{EmailDad}$ extra time to publish a result. The publishing time is $C(x) + \delta_{EmailDad}$. In the case of $\mathit{MyEmail}(a,x)$, $\max\{C(x), C(y)\}$ is needed to get values for a and x and the publication time is $\max\{C(x), C(y)\} + \delta_{Email}$. Therefore,

$$\mathit{first}(\mathit{MyNews}) = \min\{C(x) + \delta_{EmailDad},\ \max\{C(x), C(a)\} + \delta_{Email}\}.$$

Second, let us take the recursive approach. Starting with an empty context $C_0 = \langle\ \rangle$ *we write*

$$\mathsf{first}(\mathit{MyNews}) = \mathsf{first}_{C_0}(\mathit{SendNews}(a) < a < \mathit{Addresses})$$

Defining $C_1 = C_0[a \leftarrow \mathsf{first}(\mathit{Addresses})]$ *we have* $C_1(a) = C(a)$ *and:*

$$\begin{aligned}&\mathsf{first}(\mathit{MyNews})\\ &\quad = \mathsf{first}_{C_1}(\mathit{SendNews}(a)) = \mathsf{first}_{C_1}(\mathit{TwoNews} > x > \mathit{MyEmails}(a,x))\\ &\quad = \mathsf{first}_{C_1}(\mathit{TwoNews}) + \mathsf{first}_{C_1 + \mathsf{first}_{C_1}(\mathit{TwoNews})[x\to 0]}(\mathit{MyEmails}(a,x))).\end{aligned}$$

As $\mathsf{first}_{C_1}(\mathit{TwoNews}) = C(x)$ *defining* $C_2 = C_1 + \mathsf{first}_{C_1}(\mathit{TwoNews})[x \leftarrow 0]$ *such that* $C_2(a) = \max\{0, C(a) - C(x)\}$ *and* $C_2(x) = 0$ *we have*

$$\begin{aligned}&\mathsf{first}_{C_2}(\mathit{MyEmails}(a,x))\\ &\quad = \mathsf{first}_{C_2}(\mathit{EmailDad}(x) \mid \mathit{Email}(a,x)) = \min\{\delta_{\mathit{EmailDad}},\ \delta_{\mathit{Email}} + C_2(a)\}\end{aligned}$$

As $C(x) + C(a) = C(x) + \max\{0, C(a) - C(x)\} = \max\{C(x), C(a)\}$ *we get:*

$$\begin{aligned}&\mathsf{first}(\mathit{MyNews})\\ &\quad = C(x) + \min\{\delta_{\mathit{EmailDad}},\ \delta_{\mathit{Email}} + C_2(a)\}\\ &\quad = \min\{C(x) + \delta_{\mathit{EmailDad}}, C(x) + C_2(a) + \delta_{\mathit{Email}}\}\\ &\quad = \min\{C(x) + \delta_{\mathit{EmailDad}},\ \max\{C(x), C(a)\} + \delta_{\mathit{Email}}\}\end{aligned}$$

as in the intuitive approach □

We need to consider also the IfOrc expressions. Remind that,

$$\mathsf{first}(\mathit{if}(b)) = \begin{cases}\delta_{\mathit{if}} & \text{if } x = \texttt{true}\\ \omega & \text{otherwise}\end{cases}$$

Given E in IfOrc and $\mathtt{t} \in \mathsf{paths}(E)$ (see Section 3) $\delta(\mathtt{t})$ denotes the delay associated to this execution path. Such a delay is computed adding the delays of sites encountered along the path, we have $\delta(\mathtt{t}) < \omega$ iff $\mathsf{val}(t) \neq \lfloor\!\lfloor\ \rfloor\!\rfloor$.

Example 15. *Let us consider the delays associated to SelectiveReaders given in Example 8. For* $\mathtt{t} = \mathtt{ll}$*, the calls corresponds to*

$$\begin{aligned}&\big(1(\mathtt{1}) > x > \mathit{AlicePart}(x)\big)\\ &\quad = \big(1(\mathtt{1}) > x > \mathtt{if}(x) \gg \mathit{CNN} > y > \mathit{EmailAlice}(y)\big)\end{aligned}$$

and thus $\delta(\mathtt{ll}) = \delta_1 + \delta_{\mathit{if}} + \delta_{\mathit{CNN}} + \delta_{\mathit{EmailAlice}} < \omega$*. When* $\mathtt{t} = \mathtt{rl}$*, we have*

$$\begin{aligned}&\big(1(\mathtt{0}) > x > \mathit{AlicePart}(x)\big)\\ &\quad = \big(1(\mathtt{0}) > x > \mathtt{if}(x) \gg \mathit{CNN} > y > \mathit{EmailAlice}(y)\big)\end{aligned}$$

identifying 0 *with* `false` *we have* $\delta(\mathtt{rl}) = \delta_1 + \omega + \cdots = \omega$. □

The proof of Lemma 9 is inspired in Lemma 1 and Theorem 4.

Lemma 9. BoundFirst *is* NP*-complete when restricted to* IfOrc*.* First *is* NP*-hard and belongs to* PSPACE *when restricted to* IfOrc*.*

Proof. Let us consider the BoundFirst problem. To prove hardness, we consider a reduction form the 3-SAT problem given in Theorem 10. Given

$$\begin{aligned}&E_{C_i}(x_{i_a}, x_{i_b}, x_{i_c})\\ &\quad = (\mathit{if}(y_{ia}) \mid (\mathit{if}(\neg y_{ia}) \gg (\mathit{if}(y_{ib}) \mid (\mathit{if}(\neg y_{ib}) \gg \mathit{if}(y_{ic}))))) \gg 1\end{aligned}$$

it holds $\mathsf{out}(E_{C_i}) > 0$ iff $\mathsf{first}(E_{C_i}) \leq 3\delta_{\mathit{if}} + \delta_1$. Consider delays in the orchestration

$$\begin{aligned}&E(x_1, \dots, x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \gg \dots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c})\\ &E_F = (\mathit{True} \mid \mathit{False}) > x_1 > \dots > (\mathit{True} \mid \mathit{False}) > x_n > E(x_1, \dots, x_n).\end{aligned}$$

As $\mathit{True} = 1(\mathtt{1})$, $\mathit{False} = 1(\mathtt{0})$ we have $\delta_{\mathit{True}} = \delta_{\mathit{False}} = \delta_1$ and

$$(\mathit{True} \mid \mathit{False}) > x_1 > \dots > (\mathit{True} \mid \mathit{False}) > x_n > \cdots$$

has a delay $n\delta_1$. The following equivalences hold F is satisfiable iff $\mathsf{out}(E_F) > 0$ iff $[\![E_F]\!] \neq [\![\;]\!]$ iff $\mathsf{first}(E_F) \leq n\delta_1 + m(3\delta_{\mathit{if}} + \delta_1) < \omega$. To prove NP hardness in the case of IfOrc expressions we note that given F in 3-SAT, F is satisfiable iff $\mathsf{first}(E_F) \neq \omega$. To prove membership in PSPACE observe that, given E in IfOrc we have $\mathsf{first}(E) = \min\{\delta(\mathtt{t}) \mid \mathtt{t} \in \mathsf{paths}(E)\}$. Let $S_1, \dots, S_n$ be the sites appearing in E, as the computation of $\delta(\mathtt{t})$ can be computed in polynomial time in function of $\mid E \mid + \sum_{1\leq i\leq n} \log \delta_{S_i}$ and the min can be computed as a loop. □ □

The preceding results can be summarized into the following result.

Theorem 10. *The problems* BoundFirst *and* First *restricted to* ElementaryOrc *or* PruningOrc *can be solved in polynomial time.* BoundFirst *is* NP*-complete when restricted to* IfOrc*.* First *is* NP*-hard and belongs to* PSPACE *when restricted to* IfOrc*.*

4.2 Probabilistic first environments

In order to analyse the first measure instead of assuming a crash failure model we assume a probability distribution on a finite set of possible delays.

Definition 3. *Given an orchestration E having calls to sites $\{S_1, \dots, S_n\}$, a* probabilistic first environment *for E is a set $\mathcal{P} = \{P_1, \dots, P_n\}$ where, for each $1 \leq i \leq n$, P_i gives the probability distribution of S_i's delay in producing their output.*

Environment $\mathcal{P}$ is given explicitly. That is, given $S \in \{S_1, \dots, S_n\}$ with $P = (\delta_1 @ p_1 \parallel \cdots \parallel \delta_k @ p_k)$. Site S returns at time δ_i with probability p_i, for $1 \leq i \leq n$. We assume again that $p_i = x_i/y_i$ being x_i, y_i natural numbers. Any $\delta_i < \omega$ is a natural number. We encode ω as short string, when $\delta_i = \omega$ we define (to simplify notations) $\log \delta_i = \mid \omega \mid$. Then $\mid P \mid = \sum_{1 \leq i \leq k} (\log \delta_i + \log x_i + \log y_i)$ and $\mid \mathcal{P} \mid = \sum_{1 \leq i \leq n} \mid P_i \mid$.

Let us define the two computational problems associated to the first measure in probabilistic environments. Given an Orc expression E, a first probability environment $\mathcal{P}$ for E, a model $\alpha \in \{o, s\}$ where o denotes oblivious and s stable:

> $\mathsf{PrBoundFirst}$: and an integer, k compute $\Pr^{\alpha}_{\mathcal{P}}(\mathsf{first}(E) \leq k)$.
> $\mathsf{ExpectedFirst}$: compute $\mathbb{E}^{\alpha}_{\mathcal{P}}(\mathsf{first}(E))$.

Let us consider an example of such problems in the oblivious model.

Example 16. *Let Coin be a equiprobable coin with two return times, $\delta_{\mathtt{short}} < \delta_{\mathtt{long}}$. The probabilistic first environment is $P_{Coin} = (\delta_{\mathtt{short}} @ 1/2 \parallel \delta_{\mathtt{long}} @ 1/2)$. Consider $ExpTosses_n$ given in Example 10 where BlockingCoin is replaced by Coin:*

$$\begin{aligned}
ParCoin_n &= \underbrace{(Coin \mid \cdots \mid Coin)}_{n\ \mathtt{times}} \\
ManyOutputs_n &= \underbrace{((1(\mathtt{1}) \mid 1(\mathtt{1})) \gg \cdots \gg (1(\mathtt{1}) \mid 1(\mathtt{1})))}_{n\ \mathtt{times}} \\
ExpTosses_n &= ManyOutputs_n \gg ParCoin_n.
\end{aligned}$$

Note that $\mathsf{first}(ParCoin_n) > \delta_{\mathtt{short}}$ occurs only when all the sites return with delay $\delta_{\mathtt{long}}$, this happens with probability $(1/2)^n$ therefore $\Pr(\mathsf{first}(ParCoin_n) > \delta_{\mathtt{short}}) = (1/2)^n$.

Let us consider $ManyOutputs_n$. Assume that $1(\mathtt{1})$ has a delay distribution $(\delta_1 @ 1)$, that is, it returns $\mathtt{1}$ in δ_1 units with probilibity 1. Therefore, $ManyOutputs_n$ outputs 2^n values $\mathtt{1}$ with delay $n\delta_1$ and probability 1. To compute $\Pr(\mathsf{first}(ExpTosses_n) > \delta_{\mathtt{short}} + n\delta_1)$ remark that we need to fulfill the constraint at any of the 2^n publications of $ManyOutputs_n$, this give us a probability $(1/2)^{n2^n}$ and therefore

$$\Pr(\mathsf{first}(ExpTosses_n) \leq \delta_{\mathtt{short}} + n\delta_1) = 1 - \frac{1}{2^{n2^n}}$$

and, as we have seen before this needs an exponential quantity of bits to be encoded. In order to compute the expectation of $\mathsf{first}(ExpTosses_n)$ note that it can take only the values $n\delta_1 + \delta_{\mathtt{long}}$ and $n\delta_1 + \delta_{\mathtt{short}}$ and

$$\begin{aligned}
\mathbb{E}\left(\mathsf{first}(ExpTosses_n)\right) &= n\delta_1 + \delta_{\mathtt{long}} \frac{1}{2^{n2^n}} + \delta_{\mathtt{short}} \Big(1 - \frac{1}{2^{n2^n}}\Big) \\
&= n\delta_1 + \delta_{\mathtt{short}} + \frac{1}{2^{n2^n}} \big(\delta_{\mathtt{long}} - \delta_{\mathtt{short}}\big).
\end{aligned}$$

Taking $\delta_1 = \delta_{\texttt{short}} = 1$ and $\delta_{\texttt{long}} = 2$ we have

$$\mathbb{E}\left(\textsf{\textit{first}}(\mathit{ExpTosses}_n)\right) = \frac{2^{n2^n}(n+1)+1}{2^{n2^n}}.$$

As $2^{n2^n}(n+1)$ is an even number, $2^{n2^n}(n+1)+1$ is odd. As 2^{n2^n} is even the integer pair $(2^{n2^n}(n+1)+1, 2^{n2^n})$ is irreducible and therefore needs an exponential number of bits to be encoded. □

The Example 16 gives us an exponential lower bound for ElementaryOrc. Therefore, we get the following result.

Lemma 10. *In the* oblivious model*, the problems* PrBoundFirst *and* Expected-First *require exponential space for* ElementaryOrc *expressions.*

Let us consider the orchestration of Example 16 in the stable model.

Example 17. *Let us reconsider the orchestration of the Example 16 under the stable model. In the stable model we roll Coin once and therefore $ParCoin_n$ behaves like Coin and*

$$\textsf{\textit{first}}(ParCoin_n) = \textsf{\textit{first}}(Coin) = (\delta_{\texttt{short}}@1/2 \parallel \delta_{\texttt{long}}@1/2)$$

Orchestration $ManyOutputs_n$ will generate 2^n activations of a "frozen" version of $ParCoin_n$ therefore $\Pr(\textsf{\textit{first}}(ExpTosses_n) \leq \delta_{\texttt{short}} + n\delta_1) = 1/2$ and the expectation is $\mathbb{E}\left(\textsf{\textit{first}}(ExpTosses_n)\right) = n\delta_1 + (\delta_{\texttt{short}} + \delta_{\texttt{long}})/2$. □

Lemma 11. *In the stable model, the problem* PrBoundFirst *is #P-hard when restricted to PruningOrc and IfOrc.*

Proof. We provide a reduction from the problem PrPub of PruningOrc and IfOrc expressions to PrBoundFirst with delay distributions in the stable model. Let E be a PruningOrc or IfOrc expression. Each site S_i appearing in E has probability p_i of success and probability q_i of be silent. Now, let us change the sites in E, for sites S_i' following delay distributions $\delta_{S_i'} = (1@p_i \parallel \omega@q_i)$, let us name this new expression E'. Then $Pr(\mathsf{out}(E) > 0) = Pr(\mathsf{first}(E) \leq |E|)$, as each execution publishing in E has an execution publishing in E' within time $|E|$, and when not publishing the execution will take time ω. This reduces the problem PrPub in one family to the problem PrBoundFirst in the same family. □ □

Lemma 12. *In the stable model, the problem* ExpectedFirst *is #P-hard when restricted to PruningOrc.*

Proof. Let us consider the following reduction from #MONOTONE 2-SAT. Given a MONOTONE 2-SAT formula $F = C_1 \wedge \cdots \wedge C_m$ with n variables $x_1, \ldots, x_n$ where $C_i = y_{i_a} \vee y_{i_b}$, $\{y_{i_a}, y_{i_b}\} \subseteq \{x_1, \ldots, x_n\}$ for $1 \leq i \leq m$. Given a $DelayCoin$ site with delay distribution $(0@1/2 \parallel 1@1/2)$ consider n independent copies (like proxies) called $\mathit{DelayCoin}_1, \ldots, \mathit{DelayCoin}_n$. Let us redefine the orchestration E_F, in the oblivious model, as follows:

$$\begin{aligned}
&\text{for } 1 \le i \le m \text{ do } E_{C_i}(y_{i_a}, y_{i_b}) = \big(1_1(y_{i_a}) | 1_2(y_{i_b})\big)\\
&E(x_1,\ldots,x_n) = (E_{C_1}(y_{1_a}, y_{1_b}) \gg \cdots \gg E_{C_m}(y_{m_a}, y_{m_b}))\\
&E_F = (\cdots(E(x_1,\ldots,x_n) < x_1 < \mathit{DelayCoin}_1) < x_2 < \cdots) < x_n < \mathit{DelayCoin}_n
\end{aligned}$$

Given $x = (x_1, \ldots x_n)$ such that x_i has delay $\in \{0,1\}$ let us define a Boolean map $b(x_1,\ldots x_n) = (b(x_1), \ldots b(x_n))$ such that, for $1 \le i \le n$,

$$b(v_i) = \begin{cases} 1 & \text{if } x_i \text{ has delay } 0\\ 0 & \text{if } x_i \text{ has delay } 1\end{cases}$$

Observe that,

$$\mathsf{first}(E(x_1,\ldots,x_n)) = \begin{cases} 0 & \text{if } F(b(x_1), \ldots b(x_n)) = 1,\\ 1 & \text{otherwise}\end{cases}$$

As $\mathit{DelayCoin}_i$ follows a delay distribution $(0@1/2 \mathbin{[\!]} 1@1/2)$, each $x \in \{0,1\}^n$ occurs with probability $1/2^n$, then

$$\begin{aligned}
\mathbb{E}(\mathsf{first}(E_F)) &= (1/2^n) \sum_{x\in\{0,1\}^n} \mathsf{first}(E(x))\\
&= (1/2^n) \sum_{x\in\{0,1\}^n} F(b(x)) = \#\{x \mid F(x) = 1\}/2^n
\end{aligned}$$

Therefore, $\mathbb{E}(\mathsf{first}(E_F))/2^n$ solves #MONOTONE 2-SAT. □ □

Lemma 13. *In the stable model,* ExpectedFirst *is #P-hard when restricted to IfOrc.*

Proof. Let us consider the following reduction from 3-SAT. Given a 3-SAT formula $F = C_1 \wedge \ldots \wedge C_m$ over n variables $x_1, \ldots, x_n$, where $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$, $\{y_{1_a}, y_{1_b}, y_{1_c}\} \subseteq \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$, for $1 \le i \le n$. Let us encode clauses through calls to site *if*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

$$E_{C_i}(y_{i_a}, y_{i_b}, y_{i_c}) = (\mathit{if}(y_{ia}) \mid (\mathit{if}(\neg y_{ia}) \gg (\mathit{if}(y_{ib}) \mid (\mathit{if}(\neg y_{ib}) \gg \mathit{if}(y_{ic}))))) \gg 1$$

Let site *Delay* publish 1 with delay 1. Now, let us encode clause failures as follow. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

$$F_{C_i}(y_{i_a}, y_{i_b}, y_{i_c}) = \mathit{if}(\neg y_{ia}) \gg \mathit{if}(\neg y_{ib}) \gg \mathit{if}(\neg y_{ic}) \gg \mathit{Delay}$$

Taking $\mathit{True} = 1(\mathtt{1})$ and $\mathit{False} = 1(\mathtt{0})$ the orchestration E_F corresponding to the formula F is:

$$\begin{aligned}
&\mathit{BoolCoin}_i = \big(1_{i,1}(\mathtt{1}) \;{}_{1/2}\!\oplus 1_{i,2}(\mathtt{0})\big),\ 1 \le i \le n\\
&E(x_1,\ldots,x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \gg \ldots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c})\\
&F(x_1,\ldots,x_n) = F_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \mid \ldots \mid F_{C_m}(x_{m_a}, x_{m_b}, x_{m_c})\\
&E_F = \mathit{BoolCoin}_1 > x_1 > \cdots > \mathit{BoolCoin}_n > x_n > (E(x_1,\ldots,x_n) \mid F(x_1,\ldots,x_n))
\end{aligned}$$

Given $x = (x_1, \dots x_n)$ such that $x_i \in \{0,1\}$ we have that:

$$\mathsf{first}(E(x)|F(x)) = \begin{cases} 0 & \text{if } F(x_1, \dots x_n) = 1, \\ 1 & \text{otherwise} \end{cases}$$

As each $x \in \{0,1\}^n$ occurs with probability $1/2^n$, we have that $\mathbb{E}(\mathsf{first}(E_F)) = (1/2^n) \sum_{x \in \{0,1\}^n} \mathsf{first}(E(x)) = (1/2^n) \sum_{x \in \{0,1\}^n} F(b(x)) = \#F/2^n$. Therefore, $\mathbb{E}(\mathsf{first}(E_F))/2^n$ solves the #3-SAT problem. □ □

Finally, let us consider the *stable model*. The following Lemma follows the lines given in the proof of Theorem 7.

Lemma 14. *In the stable model,* PrBoundFirst *and* ExpectedFirst *belongs to PSPACE for ElementaryOrc, PruningOrc and IfOrc.*

Proof. Consider E with calls to n different sites $\{S_1, \dots S_n\}$ and probabilistic first environment $\mathcal{P} = \{P_1, \dots, P_n\}$, where $P_i = (\delta_{i_1} @ p_{i_1} \parallel, \dots, \parallel \delta_{i_k} @ p_{i_k})$ to denote thatS_i returns at time δ_{i_j} with probability p_{i_j}. As Site S_i has $\{1, \dots, i_k\}$ probabilistic choices, we define a *first delay* profile as an element of the Cartesian product $\mathcal{D} = \prod_{1 \leq i \leq n} \{1, \dots, i_k\}$. The probability of $\mathtt{f} \in \mathcal{D}$ such that $\mathtt{f} = f_1 \dots f_n$ is $\Pr(\mathtt{t}) = p_{f_1} \dots p_{f_n}$. Moreover, for a given $\mathtt{f}$, we abstract the probabilistic behaviour of $E_{|\mathtt{f}}$ and $\mathsf{first}(E_{|\mathtt{t}})$ can be computed in polynomial space according to Theorem 10. (look also at Lemmas 8,9). We solve ExpectedFirst iterativelly computing $\mathbb{E}(\mathsf{first}(E)) = \sum_{\mathtt{f} \in \mathcal{D}} \Pr(\mathtt{f}) \mathsf{first}(E_{|\mathtt{f}})$ and to solve PrBoundFirst we compute $\sum_{\{\mathtt{f} \in \mathcal{D} | \mathsf{out}(E_{|\mathtt{f}} \leq k)\}} \Pr(\mathtt{s})$. □ □

We sumarize the preceding results into the following Theorem:

Theorem 11. *In the* oblivious model*, the problems* PrBoundFirst *and* ExpectedFirst *requires exponential space for ElementaryOrc expressions. In the* stable model*, both problems are #P-hard in PruningOrc or IfOrc and belong to PSPACE in ElementaryOrc, PruningOrc or IfOrc.*

5 Conclusions and open questions

In this paper we have developed formally the appropriate semantics to reason about QoS measures based on productivity and latency. From those semantics we have been able to isolate subfamilies of finite orchestrations where the intended measures are well defined. For those subfamilies where the QoS measures are well defined, we have analysed the impact of the expressiveness of the family versus the computational complexity of several problems related to their computation in both reliable and unreliable environments (see Table 2). Some of our hardness results rely on a natural coding for rational numbers. Our results do not rule out having a different complexity classification under other explicit or implicit codings.

Taking into account the decomposition of the meaning in timed fully defined variable semantics, as given in Theorem 8, there are other well defined QoS measures. In particular one could consider the *delay of the last published item* or the *average delay of the produced items*.

With respect to the probabilistic model, we have contemplated only two independence models for repetitive calls, complete independence (oblivious model) and a complete correlation (stable model). There remain many levels of correlation of interest. For example correlation depending on the state of the network or some joint hypothesis over sets of sites or site types. Another way to deal with correlations is to consider faulty behaviours as strategic situations modeled by angel-daemon games as it was done partially for the `out` measure in [11]. This approach has been extended to periodic orchestrations in [8].

Acknowledgement

The authors want to thank the anonymous referees for the careful reading that help us to improve the readability of the paper and correct several inaccuracies.

References

[1] Aws service health dashboard. `http://status.aws.amazon.com`

[2] Google apps status dashboard. `http://www.google.com/appsstatus`

[3] IPFW, Personal page at Purdue University Fort Wayne. `http://users.ipfw.edu/jehle/courses/verbs/VERB.HTM`

[4] Stackoverflow. `http://stackoverflow.com`

[5] Yahoo. `http://yahoo.com`

[6] Balcazar, J.L., Diaz, J., Gabarro, J.: Structural Complexity I. EATCS Series Texts in Theoretical Computer Science. Springer (1995)

[7] Cardoso, J., Sheth, A.P., Miller, J.A., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. J. Web Sem. **1**(3), 281–308 (2004)

[8] Castro, J., Gabarro, J., Serna, M., Stewart, A.: The robustness of periodic orchestrations in uncertain evolving environments. In: Symbolic and Quantitative Approaches to Reasoning with Uncertainty - 13th European Conference, ECSQARU 2015, Compiegne, France, July 15-17 (2015)

[9] Dong, J.S., Liu, Y., Sun, J., Zhang, X.: Towards verification of computation orchestration. Formal Asp. Comput. **26**(4), 729–759 (2014)

[10] Fu, Y., Vahdat, A., Cherkasova, L., Tang, W.: Ete: Passive end-to-end internet service performance monitoring. In: Proceedings of the General Track of the annual conference on USENIX Annual Technical Conference, ATEC '02, pp. 115–130. USENIX Association, Berkeley, CA, USA (2002)

[11] Gabarro, J., Serna, M., Stewart, A.: Analysing web-orchestrations under stress using uncertainty profiles. The Computer Journal **57**(11), 1591–1615 (2014)

[12] Gao, H., Miao, H., Zeng, H.: Predictive web service monitoring using probabilistic model checking. Appl. Math. Inf. Sci. **7**(1), 139–148 (2013)

[13] Gilly, K., Quesada-Granja, C., Alcaraz, S., Juiz, C., Puigjaner, R.: A statistically customisable web benchmarking tool. Electr. Notes Theor. Comput. Sci. **232**, 89–99 (2009)

[14] Greifeneder, J., Frey, G.: Probabilistic delay time analysis in networked automation systems. In: Proceedings of 10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2005. IEEE (2005)

[15] Hoare, T., Menzel, G., Misra, J.: A tree semantics of an orchestration language. In: G.J.H.D.H.T. Broy M. (ed.) Engineering Theories of Software Intensive Systems, *NATO Science*, vol. 195, pp. 331–350. Springer (2005)

[16] Kitchin, D., Cook, W.R., Misra, J.: A language for task orchestration and its semantic properties. In: C. Baier, H. Hermanns (eds.) CONCUR 2006 - Concurrency Theory, 17th International Conference, *LNCS*, vol. 4137, pp. 477–491. Springer (2006)

[17] Krushevskaja, D., Sandler, M.: Understanding latency variations of black box services. In: Proceedings of the 22nd international conference on World Wide Web, WWW '13, pp. 703–714. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland (2013)

[18] McIver, A., Morgan, C.: Abstraction, Refinement and Proof for Probabilistic Systems. Springer, Berlin (2005)

[19] Menasce, D.A.: Qos issues in web services. IEEE Internet Computing **6**(6), 72–75 (2002)

[20] Menasce, D.A., Almeida, V.: Capacity Planning for Web Services: metrics, models, and methods, 1st edn. Prentice Hall PTR, Upper Saddle River, NJ, USA (2001)

[21] Misra, J., Cook, W.: Computation orchestration: A basis for wide-area computing. Software and Systems Modeling **6**(1), 83–110 (2007)

[22] Papadimitriou, C.H.: Computational Complexity. Addison Wesley (1994)

[23] Rosario, S., Benveniste, A., Jard, C.: Flexible probabilistic QoS management of orchestrations. Int. J. Web Service Res. **7**(2), 21–42 (2010)

[24] Stewart, A., Clint, M., Harmer, T., Kilpatrick, P., Perrott, R., Gabarro, J.: Assessing the reliability and cost of web and grid orchestrations. In: Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, Technical University of Catalonia, Barcelona, Spain, pp. 428–433. IEEE (2008)

[25] Stewart, A., Gabarro, J., Keenan, A.: Reasoning about orchestrations of web services using partial correctness. Formal Aspects of Computing **25**(6), 833–846 (2013)

[26] Stross, R.: 99.999% reliable? don't hold your breath (2011). `http://www.nytimes.com/2011/01/09/business/09digi.html`

[27] Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM J. Comput. **8**(3), 410–421 (1979)

[28] Vardoulakis, D., Wand, M.: A compositional trace semantics for orc. In: D. Lea, G. Zavattaro (eds.) Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Oslo, *LNCS*, vol. 5052, pp. 331–346. Springer (2008)

[29] Wehrman, I., Kitchin, D., Cook, W., Misra, J.: A timed semantics of Orc. Theor. Comput. Sci. **402**(2-3), 234–248 (2008)

[30] Wheeler, E.: Security Risk Management. Elsevier, Amsterdam (2011)