UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Llenguatges
i Sistemes Informàtics

MASTER THESIS

QoS Measures for Orchestrations in Unreliable Environments

STUDENT: Sergio Leon Gaixas
DIRECTORS: Joaquim Gabarro Valles, Maria Serna Igleasias
DATE: September 1, 2013

Abstract

When dealing with Grid applications, there could append a lot of different thinks that could make the final execution differ some times. For example, is over-optimistic to suppose that all the site calls that the Grid application management system does to perform sub-computations are operatives the 100% of the time and no error never happens. Another case, for example is when the final result varies depending on the values returned by the services.
When the user accepts this non-determinism, it may be good to give some estimation of success. Here we will see some approaches to this problem based on the Expected number of published values and the Probability of failure on non-recursive Orc expressions describing the orchestration of sites in Grid applications. We will see that in some cases those problems are only tractable under certain conditions and the special case created by the delay of site calls when there are no restrictions on the Orc expressions.
After that, a new approach based on the probability of delays will be presented. With this approach we will give a new estimations based on the average time required for values to be published and the time required to publish the first or last value.
As an annex, a Java library to compute some of those problems is given and some important points will be seen.

Contents

Chapter 1

Introduction

As time passes, the number of on-line services available increases at high speed, thanks that the distributed model of computation using on-line services is gaining more ground, not only for task for which is essential but also more and more task are migrating from the centralized common model.
Early applications like the SETI@home project [23] where a high workload is distributed between users were the starting point of actual multi-client applications. Nowadays, multi-client applications where some calculations are transferred to secondary servers in order to reduce the workload in the main server have evolved since the first applications showing how powerful decentralized processes can be.

Some applications uses services like on-line spell checkers and translators to solve some problems externally and with the benefit of having always the latest version without the need of install or actualize extra software. Other, like vacation planning, simply work retrieving data from other services or comparing the results searching the best combinations. At present, some applications even have completely migrated to on-line versions, like text editors, slide-show creators or image editors, erasing the need of having high performance computers for some tasks.

As the scope of the distributed applications nowadays is very wide, result a difficult task to simplify and analyse their behaviour in a unique form.

An inherit pattern in most of the on-line distributed programs nowadays is this:

- Do some computations with the current data
- Ask remote services to compute some data
- Acquire new data from remote services
- Start again

This is a simple pattern but it describes quite well how most of this applications works regardless the goal of the application.
For example, in an application like the SETI@home project the main server retrieves the data from satellites, then do some calculations and data partition, then it ask clients (services) to process the data and retrieve the results when clients end their tasks, after that repeat all the process. Other example is the spell-checking in some web-browsers where, when the user

introduces new text, the application select the new text and ask a remote service to check it and retrieve if the text is correct or some possible corrections. Vacation planning is another example that holds to this pattern, first the user gives some restrictions and preferences, then the application ask for flights to some airlines and search for those with good combinations, after that it search for hotels and finally return the best combinations founded.

When you enter the area of distributed system, there are three important words that suddenly appear, *Web services*, *orchestration* and *choreography*. Quoting the glossary of *W3C* [51], `http://www.w3.org/TR/ws-gloss/`, the definitions of *Web service*, *orchestration* and *choreography*. Following a definition of web service:

> *A* Web service *is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Web services is nowadays a well established technology, detailed information can be found in [3, 42]. Such a services are mainly separated into two big parts, orchestrations and choreographies [43]. An orchestration represents control from one party's perspective::

> *An* orchestration *defines the sequence and conditions in which one Web service invokes other Web services in order to realize some useful function. I.e., an orchestration is the pattern of interactions that a Web service agent must follow in order to achieve its goal.*

A choreography considers multi-party sources and:

> *A choreography defines the sequence and conditions under which multiple cooperating independent agents exchange messages in order to perform a task to achieve a goal state.*
>
> *Web Services Choreography concerns the interactions of services with their users. Any user of a Web service, automated or otherwise, is a client of that service. These users may, in turn, be other Web Services, applications or human beings. Transactions among Web Services and their clients must clearly be well defined at the time of their execution, and may consist of multiple separate interactions whose composition constitutes a complete transaction. This composition, its message protocols, interfaces, sequencing, and associated logic, is considered to be a choreography.*

In general we see that there are two main possible perspectives, *orchestrations* and *choreographies*. An orchestrations take the view from a local machine using external services. A choreography coordinate multiple-sites from a global perceptive without a central point.

Orchestration always represents control from one central point trying to accomplish some goal. Here, we are interested in get some goodness measures for the accomplishment of that goal when external services have uncertain behaviours.

For practical issues *WSDL*, Web service Description Language, and *BPEL*, Business Process Execution Language, appears as main languages to describe and implement orchestrations [42, 43]. In order to give a high level description of orchestrations, Jayadev Misra and William R. Cook introduce Orc [38], a minimalistic language to deal with orchestration. The Orc language can be accessed on line through

`http://orc.csres.utexas.edu/research.shtml`

It has been used both, as a programming language [32] and also as a starting point to deal with the semantics of web services. This is a rich topic.

- A first semantic approach using trees was developed in [27].
- A *trace* approach has been undertaken in [30, 49].
- The approach undertaken in [30] has been extended adding *time* to traces to get a *timed trace semantics* [52]. Links between timed trace semantics and timed automata [6] has been developed in [9, 10].
- Other approaches have been undertaken. A description in terms of predicative semantics [24, 25] has been undertaken for Orc in[46]. A real-time rewriting approach appears in [5, 4]. Relation between Orc and the REO model is undertaken in [45].

Although there are many papers dealing with different semantics aspects of Orc [27, 30, 52, 6, 9, 10, 45] the link between the semantic models and computational complexity have not been considered to the best of our knowledge.

Our main goal is to give QoS measures to orchestrations expressed in the Orc language. QoS measures can be designed to analyse "a priori" (before execution) the quality of orchestrations. To do so we first have to develop appropriate semantic models to capture the main aspects of the QoS measure. We consider to semantic approaches *fully defined variable model* and *fully defined variable-time model*.

For these measures, we first study the sub-families of Orc in which the semantics allows the definition of the QoS measure. We analyse first the complexity versus expressibility in a deterministic context. We finally incorporate a priori probabilistic

In general, we are going to give QoS measures from two different perspectives, measures related to the number of items resulting of an execution and measures related to the execution time. Our study linking QoS measures and complexity start from the measures defined in [17, 15]. In this work we extend the study by providing adequate semantics and to more expressive families or Orc.

1.1 Content Outline

This thesis is separated into different sections:

First, Chapter 2 give a brief introduction about the reliability of Web services and shows some points from where we can extract probabilities for further uses. We analyse two different kind of problems. First, problems related to attacks and other causes like natural phenomena or human negligence, with punctual causes and difficult to modelling into probability models.

After that, we see some problems related precisely to the network and server reliability and to the execution of the service.

In Chapter 3, we give a short introduction to of Orc language. The Orc language represent services as sites, we introduce the idea of non-blocking sites in order to get sites easier to analyse. In a similar way, we present the idea of descriptors of values, in order to reduce the big number of possible outputs of services. We introduce some operators of the Orc language, original operators and extensions. Finally, based on descriptors and operators, we present the different sub-families of Orc for which we are going to extract QoS measures.

In Chapters 4 and 5 we introduce two semantic approaches, the *fully defined variable model* and the *fully defined variable-time model*. In Chapter 4, we use the fully defined variable model in order to show possible QoS measures related to the number of items published during the execution of an orchestration, and then we analyse those measures for each sub-family of Orc. After that, in Chapter 5, we use the fully defined variable-time model in order to show possible QoS measures related to the time needed for the items published during the execution of an orchestration, and then we analyse those measures for each sub-family of Orc.

In Chapter 6, we present a probabilistic framework used to analyse orchestrations under failures and errors, and also probabilistic sites. We present different models in order to model different relations and dependencies between site calls and an services and observe the relation between these models and the reality. Finally, in Chapters 4 and 8, we use the recently introduced probabilistic framework to analyse new QoS measures related to the seen previously.

As an appendix, following the ideas and algorithms proposed in this document, we give a little Java library for compute some of the QoS measures related to the number of items published. In the appendix B we will give a brief overview of this library and see all its functions. This library will be not only able to compute these QoS measures but also to execute Orc expressions of the sub-families seen following the model introduced in 4. We also explain how to use this library in order to create real site and do real executions.

1.2 Technical preliminaries

1.2.1 Complexity classes and problems

Following, we resume the complexity classes and related problems appearing in this thesis.

P. Computable in poly-time. We use the complexity class P for both decision and non-decision problems.

PSPACE. Computable in poly-space. We use the complexity class PSPACE for both decision and non-decision problems.

NP. Poly-length membership certificates verifiable in poly-time.
A language $L \in \{0,1\}^*$ is in NP if there exists a polynomial $p(n)$ and a TM running polynomial time such that for every $x \in \{0,1\}^*$ we have:

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^{p(|A|} s.t. (M(< x, y >) = 1)$$

In order to prove NP hardness, we use the $3-SAT$ problem:

- $3-SAT$. Given a $3-CNF$ formula F with n literals $x_1,\ldots,x_n$, determine if there exist an assignation of values to $x_1,\ldots,x_n$ that satisfies F.

♯P [48]. Class of functions $f:\{0,1\}^*\to\aleph$ s.t. there exist a poly-time TM M and a polynomial $p(n)$ such that for every $x\in\{0,1\}^*$ we have:

$$f(x)=|\{yin\{0,1\}^{p(|x|)}:M(<x,y>)=1\}|$$

In order to prove ♯P hardness, we use the *Counting* $3-SAT$ and *Counting MONOTONE* $2-SAT$ problems:

- *Counting* $3-SAT$. Given a $3-CNF$ formula F with n literals $x_1,\ldots,x_n$, count the number of satisfying assignments of values to $x_1,\ldots,x_n$ that satisfies F.
- *Counting MONOTONE* $2-SAT$. Given a MONOTONE $2-CNF$ formula[1] F with n literals $x_1,\ldots,x_n$, count the number of satisfying assignments of values to $x_1,\ldots,x_n$ that satisfies F.

♯.NP [26]. Class of functions $f:\{0,1\}^*\to\aleph$ s.t. there exist a poly-time NTM M and a polynomial $p(n)$ such that for every $x\in\{0,1\}^*$ we have:

$$f(x)=|\{y\in\{0,1\}^{p(|x|)}:M(<x,y>)=1\}|$$

In order to prove ♯.NP hardness, we use the *Counting with* $3-SAT$ *oracle* problem:

- *Counting with* $3-SAT$ *oracle*. Given a $3-CNF$ formula F with n literals $x_1,\ldots,x_n$ and a value $j<n$, count the number of satisfying assignments of values to $x_1,\ldots,x_j$ such that exist an assignation of values to $x_{j+1},\ldots,x_n$ that satisfies F.

Given a CNF formula F with n literals $x_1,\ldots,x_n$ and a string $v=(v_1\ldots v_n)\in\{0,1\}^n$, with an abuse of the language, we write $F(v)=1$ if $F(v_1,\ldots,v_n)=true$, otherwise $F(v)=0$.

1.2.2 Multi-sets

In order to define the *fully defined variable model* and the *fully defined variable-time model* semantics, we need first the notion of multi-set or bag and few properties of that. A multi-set is a non-ordered set allowing repetitions. We use the notation $\lfloor\!\lfloor I_1,\ldots,I_n\rfloor\!\rfloor$, taken from [35], to describe a bag containing the items $I_1,\ldots,I_n$.

When using multi-sets we consider the operations $+$, denoting bag union, following [35, pag 82]. As an example:

$$\lfloor\!\lfloor \mathtt{a},\mathtt{b}\rfloor\!\rfloor+\lfloor\!\lfloor \mathtt{a},\mathtt{c}\rfloor\!\rfloor=\lfloor\!\lfloor \mathtt{a},\mathtt{a},\mathtt{b},c\rfloor\!\rfloor$$

Given multi-sets $M_1,\ldots,M_n$ we note as usual

$$\sum_{1\le i\le n}M_i=M_1+\cdots+M_n$$

[1] A MONOTONE CNF formula is a formula such that all literals never are negated.

We introduce also the "demonic choice" operator $\sqcap$ [35, pag 4] to denote non-deterministic choice between bags. Given two possibilities A and B, a demonic choice between, $A \sqcap B$, implies that either A or B will be chosen but no both. The demonic choice operator is idempotent in the sense that $A \sqcap A = A$ and follows typical distributive laws, so:

$$\begin{aligned}
&(\llbracket \mathtt{x} \rrbracket \sqcap \llbracket \mathtt{y} \rrbracket) + (\llbracket \mathtt{x} \rrbracket \sqcap \llbracket \mathtt{y} \rrbracket \sqcap \llbracket \mathtt{z} \rrbracket) \\
&\quad = \llbracket \mathtt{x}, \mathtt{x} \rrbracket \sqcap \llbracket \mathtt{x}, \mathtt{y} \rrbracket \sqcap \llbracket \mathtt{x}, \mathtt{z} \rrbracket \sqcap \llbracket \mathtt{y}, \mathtt{x} \rrbracket \sqcap \llbracket \mathtt{y}, \mathtt{y} \rrbracket \sqcap \llbracket \mathtt{y}, \mathtt{z} \rrbracket \\
&\quad = \llbracket \mathtt{x}, \mathtt{x} \rrbracket \sqcap \llbracket \mathtt{x}, \mathtt{y} \rrbracket \sqcap \llbracket \mathtt{x}, \mathtt{z} \rrbracket \sqcap \llbracket \mathtt{y}, \mathtt{y} \rrbracket \sqcap \llbracket \mathtt{y}, \mathtt{z} \rrbracket
\end{aligned}$$

Given a demonic choice between multi-sets $M_1, \ldots, M_n$ we note

$$\sqcap_{i=1}^{n} M_i = M_1 \sqcap \cdots \sqcap M_n$$

Chapter 2

Reliability & Probabilities

When we work with web services executed in unreliable environments we need to consider several problems. Remote servers may be off-line at times for multiple reasons. Even if a service is on-line, there maybe problems on the net, the server may need more time that the given to perform its action among other sources of uncertainty.

These problems can be either instantaneous and affect a unique site call or persistent during a certain amount of time. In case of persistent problems, we need also to take into account that maybe there is not only one service affected, but services on the same server or network can be also affected depending on the problem.

With all these possibilities it is clear that we cannot rely on services with deterministic behaviours for measure orchestrations. In this chapter we analyse some cases of uncertain behaviour on sites and how to extract probabilities in order to use then to measure the goodness of orchestrations. We consider three main scenarios: *Attacks and Failures*, *Reliability*, and *Time-outs*. Given the large number of possible problems and the complexity of extract probabilities from them, we are going to focus on searching pessimist measures of the behaviour of services.

Finally, we need to have always in mind that, as we are working with complex systems where almost anything can happen, the probabilities and the related results will be only approaches to the real behaviour.

2.1 Attacks and Problems

The most serious cause of misbehaviours are human attacks and problems with physical components. Sometimes accidents involving facilities or hardware occur, also humans can attack web servers or network components. These problems and attacks may slow servers, make them off-line or even turn off part of a network. This kind of failures is the most widely spoken of network problems because of their nature. Now let us analyse different types and examples of common problems and attacks.

Fire and another problems. Problems on physical components or facilities, commonly occur due to human errors, poor maintenance or external actions. These problems may have different severity, from a short disconnection to a destruction of physical components in an unpredictable way. When the problem only causes a short disconnection there is no additional problems. When some components are destroyed, like a hard disk that may create

a big issue (if there is no backup available), something common for centralized services having all the servers located in the same building. In order to prevent the worst case scenario, it is recommended to have a backup server on a different Data Center or contract a hosting that ensures a periodic backup, but that also mean a higher cost.

The fire on a Data Center in Macomb County, Michigan on April 17th 2013 that caused the loss of a big number of services in the area is a good example of these type of problems [36]. The IT services were hosted in a Data Center located in the basement of an old building that was not prepared and besides they did not have a backup Data Center. Because that, the fire put off-line all the IT services of the Macomb's government and will require millions of dollars worth of repairs and upgrades. These problems would have been easily avoided if the government had originally invested in good facilities, but even so this is still a common problem.

In addition to physical problems, sometimes the service can be disrupted by problems related to the software. These problems normally are caused by bad data updates or license expiration, usually due to human errors, and may stop all the services until they are solved.

The outage of *Windows Azure Cloud* servers on February 22th 2013 was caused by a expired SSL Certificate [37]. This is an interesting example of what an oversight can cause. Something as simple for a company like Microsoft as updating a certificate, would have avoided an outage of more than 12 hours on their services.

Denial of Service Attacks. One of the most common attacks is *Distributed Denial of Service* (*DDoS* for short) [50]. In these attacks, a big number of machines try to saturate some servers with simultaneous connections. These attacks affect not only the attacked server but also the near routers as they need to endure a bigger traffic. There are temporally solutions to this attacks and hostings tend to proceed in a different way depending on the price of the plans, but normally, although the web can be maintained on-line, these attacks create a significant delay in the response from the server.

The *DDoS* attack suffered by the web *Spamhaus* on March 18th 2013, possibly the biggest *DDoS* attack so far is a good example of what a *DDoS* attack can do [29, 44]. This attack put the web off-line during some time until measures where taken. During almost all the attack duration, it saturated the networks of *Cloudflare* (the hosting server *Spamhaus*) and adjacent connecting points. As *Spamhaus* had contracted the costliest plan, the hosting provided the best response against the attack and because that the web was on-line most of the time. However, if they had contracted another plan, the same hosting would have provided a worst response and the web would have been off-line until the attack stopped.

While the last example was a mass attack to a unique server, it is more common to have a series of small attacks to related servers. Those series of attacks are commonly more efficient. The series of *DDoS* attacks suffered by some on-line-banking sites starting in March 5th 2013 [33]. Although smaller than the *Spamhaus* attack, these may be a better example of what attacks can do to on-line services. In this case, unlike in *Spamhaus* where a really big attack saturated the line but the service only was ceased at the beginning, these small but scattered attacks had much more success as, although not always achieved, they stopped momentarily several servers, causing a considerable total off-line time.

Service Inaccessible Attacks. Besides collapsing servers, other attacks make the services inaccessible. Following two common attacks of this kind:

- *Website defacement.* This attack consist in entering into the server and modify the data [53]. Normally there is not a big loss of information as the attacks tend to affect configuration or access files in order to redirect all requests to a new file.

- *DNS cache poisoning.* This attack consist in entering into an intermediate DNS server and add bad entries wanting them to be propagated to other DNS servers [40]. In these cases, servers becomes inaccessible until the correct DNS entries are restored.

Although they have these type of attacks same final goal as *DDoS*, they follow completely different procedure. The attack affects only the attacked server, modifying the data or making it inaccessible, this means that nearby networks are not affected. Another difference is that in order to recover from the attack, actions like restore data or reseed DNS entries are needed. These measures solve the problems from root, although they may need some time. In order to avoid those problems, it is needed to have the servers with the latest versions since they have less known vulnerabilities. That action may be something important to consider if we expect to have high risk of attack, but can affect the cost as highest paid servers care more about security and tend to be more updated.

On November 24th 2012, Google Pakistan's web page was defaced [8]. That attack make the service unavailable until they realized and deleted the redirection from the main page but the all problem took some more time to be solved. Attack like this proves that even the bigger companies can be attacked successfully.

Extracting probabilities. As said, those attacks and problems are unpredictable, but we can try to extract some probability about the likelihood of suffering a misbehaviour. We can get an estimation of this probability from different origins. Big companies like Google, that are often attacked, can have some public data about the status of the services offered like the status page of Google services [19]. Big hostings also have public statistics of the server status, like in Amazon Web Services [7]. Finally, we can also get statistics from similar services.

Example 2.1 In a report from *SecureList* we have that in the second half of 2011 the average duration of a *DDoS* attack on the tested servers was 9 and a half hours [34], and the most attacked received 218 attacks on the second quarter of the same year [39]. For instance, let WS_A and WS_B be two different web services:

- Service WS_A is attacked with high probability, let us say that it gets 22 attacks each quarter (10% compared to most attacked server). With that high risk they have a server contracted with good protection versus *DDoS* and that makes that only 1% of the attacked time really entails problems.

- Service WS_B has less probability of been attacked, let us say that it gets around 2 attacks each quarter (1% compared to most attacked server). With less probability they contract a simple hosting without protection versus *DDoS* and therefore the service is off-line during the attacks.

If the average attack time is 9.5 hours like in the average of all attacks, WS_A will be under attack around 209 hours attacked in 90 days ($9,68\%$ of the time) and WS_B around 19 hours ($0,88\%$ of the time). Given the difference in the services contracted, while WS_B has no protection against attacks and therefore will be on-line only $99,12\%$ of the time, WS_A with

the extra protection contracted will counter most of the attack and will not have problems $99,999\%$ of the time, and that will be a "'perfect"' service as will be shown next.

In general, although we can get an approximate perception of the probability of being under attack for the services. Unless the service has a very bad maintenance, this probability tends to be a small probability with low effect on the goodness of the orchestration using these services.

At this point, in the reports of *SecureList* about attacks in the 2nd quarter of 2011 [39] and 2nd half of 2011 [34], an interesting point is the *Distribution of attacked websites by on-line activity* (figure 2.1). From that distribution we can extract the risk of being attacked depending on the service offered. From this statistics, we may be able to compute some estimated probability of being under attack if we knew the total number of attacks and the number of sites of each kind. Even so, as it can be seen in the graphs, the distribution of attacks by activity has a big variance and, apart of the site activity, there are other causes that lead to attacks. These distribution of attacks by sites may be a good approach for selecting groups of sites by risk when approximating problems by other methods like with competitive games, as we will see in chapter 9 (Note that the difference on number of sites has also importance for considering the risk of attack.).

2.2 Network and Servers Reliability

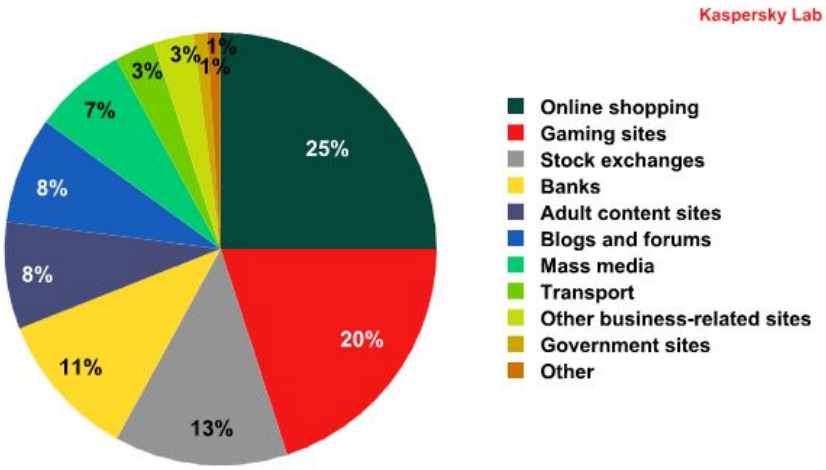
The use of web services implies to suffer all the possible problems on networks: off-line servers, network failures and server congestion among others. In general, we have two distinct problems that can make our orchestration fail, errors related to the *reliability* of the network and the servers and *time-out* on calls. We analyse separately with both cases. Citing an interesting article from New York Times [47]:

> AT&T's dial tone set the all-time standard for reliability. It was engineered so that 99.999 percent of the time, you could successfully make a phone call. Five 9s. That works out to being available all but 5.26 minutes a year.

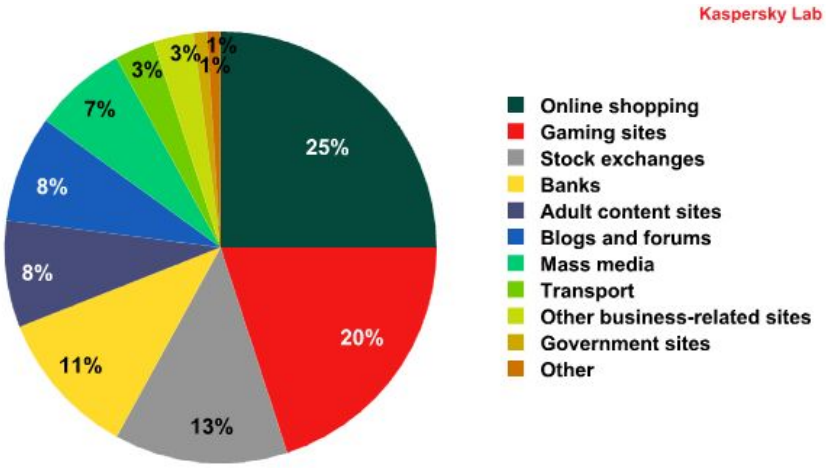
As it is pointed out in[47], while phone calls have a reliability of 99.999%, this standard is almost impossible to achieve in a web service due to its higher complexity. The Five 9s mean that a service can not be off-line more than 5 minutes a year. If the service is located in a single machine, it is clear that Five 9s is not realistic. Either network problems or server maintenance may cause to not reach the Five 9s.

What happens with *big services*? Although the great services can have a myriad of machines, also have to deal with the reliability of the network. This fact makes impossible to reach the 100% reliability, reaching as a maximum the Five 9s. As the senior vice president for operations at Google Urs Hölzle said [47]: *We don't believe Five 9s is attainable in a commercial service, if measured correctly.* After that, he stated that their goal for the major services was Four 9s (99.99%), though their search service is near the 99.999% every year. These Four 9s are the goal for all big Web services, and every year there are more services arriving to it, but in the end, they can not ensure more than those Four 9s.

Let us consider what happen when we go to other services, not only for the bigger ones. In this case we can have two different possibilities *services on Web Hostings* or *services on*



(a) Q2 2011



(b) H2 2011

Figure 2.1: Distribution of DDoS attacks by on-line activity in 2nd quarter and 2nd half of 2011 by Kaspersky Lab (look at [39], [34]).

private networks. The more common case, where the services are hosted in some servers maintained by a third party. In general, there are a lot of combination possible of Web hosting, from dedicated servers completely managed by the contractor to shared servers with big restrictions or even static pages servers, but in the end there is one simple thing that affects directly the reliability of the site, how much is paid. Web hosting is a business and as such the price is what makes the difference.

Consider the hosting *Hostinger*. At the moment of writing this, the hosting offer three different plans. The *free plan* does not have any cost, but ensures only 99% of on-line time. In addition to that, they offer two *paid plans*, one for 2.41€/month ensuring an on-line time of 99.5% and the other for 5.63€/month ensuring 99.9%. The different reliability of each plan is a good example of how the price can make a difference, and that without counting other improvements and protection on paid servers.

Host Name/Page Url	Service	Status	Check period	Last Check	Resp. time	# checks	# outages	# failures	Uptime
server11.000webhost.com	http (80)		3 minutes	12/21/2012	0.215 s	609007	536	1873	**99.692 %**
server12.000webhost.com	http (80)		3 minutes	12/21/2012	0.192 s	609136	430	1772	**99.709 %**
server13.000webhost.com	http (80)		3 minutes	12/21/2012	0.196 s	609365	346	2032	**99.667 %**
server14.000webhost.com	http (80)		3 minutes	12/21/2012	0.213 s	609355	726	2108	**99.654 %**
server15.000webhost.com	http (80)		3 minutes	12/21/2012	0.208 s	609241	294	1093	**99.821 %**
server16.000webhost.com	http (80)		3 minutes	12/21/2012	0.227 s	609138	503	2190	**99.64 %**
server17.000webhost.com	http (80)		3 minutes	12/21/2012	0.227 s	607647	483	1484	**99.756 %**
server18.000webhost.com	http (80)		3 minutes	12/21/2012	0.226 s	609011	433	3367	**99.447 %**
server19.000webhost.com	http (80)		3 minutes	12/21/2012	0.225 s	609007	386	1909	**99.687 %**
server20.000webhost.com	http (80)		3 minutes	12/21/2012	0.269 s	609287	708	1816	**99.702 %**
server21.000webhost.com	http (80)		3 minutes	12/21/2012	0.234 s	609281	956	2536	**99.584 %**
server22.000webhost.com	http (80)		3 minutes	12/21/2012	0.221 s	609274	358	968	**99.841 %**
server23.000webhost.com	http (80)		3 minutes	12/21/2012	0.221 s	609267	401	1551	**99.745 %**
server24.000webhost.com	http (80)		3 minutes	12/21/2012	0.243 s	609256	490	1376	**99.774 %**
server25.000webhost.com	http (80)		3 minutes	12/21/2012	0.223 s	608907	466	2028	**99.667 %**
server26.000webhost.com	http (80)		3 minutes	12/21/2012	0.229 s	609044	471	2235	**99.633 %**
server27.000webhost.com	http (80)		3 minutes	12/21/2012	0.207 s	609237	405	2287	**99.625 %**
server28.000webhost.com	http (80)		3 minutes	12/21/2012	0.214 s	609231	354	1861	**99.695 %**
server29.000webhost.com	http (80)		3 minutes	12/21/2012	0.215 s	609229	290	1122	**99.816 %**
server30.000webhost.com	http (80)		3 minutes	12/21/2012	0.219 s	609213	720	1643	**99.73 %**

Figure 2.2: Servers Log: Uptime report of *000Webhost* hosting by `serviceuptime.com` (look at [13]).

When working with these services a common question is: *can we trust the reliability announced by the provider?*. For paid services it is more or less clear that we can somehow accept their words because if they do not fulfil them, we have the right to complain, and if something happens, as in the case of Windows Azure, the provider tends to compensate for it. But in free servers, though sometimes surprising, we can also trust the reliability ensured.

Let us see the web hosting *000Webhost*. Although they offer paid services, their most important service is their free web hosting. For that service they ensures a 99% reliability. When daily using these services sometimes it seems that they do not accomplish that, because errors are subjectively amplified. In order to check if the 99% reliability is real the best way is to use an external service to test it. In this case, for example there is available a report *Serviceuptime.com* about the servers of *000Webhost* (figure 2.2). From that report we see that, not only all servers have at least 99% uptime, but that almost all servers have more that 99.5% uptime.

Another possibility, although less common, is that the services are hosted in some private network managed by the same service provider. These service providers can be either public institutions, private companies or common users and in the same time we can have servers

with as many varieties as in Web hosting plus the case that the server is used for other purposes at the same time. The last case is the more common when the service provider is a common user. These type of servers do not have any ensured on-line time and, in addition, tend to be highly limited, specially in the number of concurrent connections, and therefore we cannot have a ensured reliability as with Web hosting. Still, in these cases we can extract some probabilities from statistical analysis of the service and therefore get some approximated probabilities. These services are also more prone to problems like in the Macomb case.

Although those can all be good reliabilities, that is only the reliability of the services. Unless the service consist in simply returning static pages, it is almost certain that the services will use some other services like Databases. In that case, the final reliability of the service will be something less that the stated by the provider. Web services using different servers can be simulated by Orc expressions. In the nexts chapters we will use these orchestrations to compute the probability of publishing in order to measure the real reliability of the services.

2.3 Time-out and failures probability

As we have pointed out, the service reliability is affected by the network reliability. However Network protocols tend to have failure correction systems and it is not common to have a service inaccessible if the server is on-line, except in case that the server is under some attack or has some problems. In the end, network reliability ends up affecting mainly the delay in the communications. When working with Web services, it is quite common to establish a maximum time for each call in order to avoid endless calls. Thus a failure is considered when the time-out is triggered, see for example how web browsers manage long queries or scripts [11] [12]. We can set this time-outs as a fixed big time-out for all services, time-outs to groups of similar services or as an specific time-out for each service.

A *fixed time-out* for all services is a simple solution for cases where we do not know too much about the services. In this case, as we do not know if calls will take more or less time, we let a large amount of time to perform any action. This is the worst approach to time-outs, as we can wait a lot of time for fast services when they fail. On the other hand, if we are only interested in a time limit to perform all the orchestration, this method gives a fast and easy way to compute the time-out needed.

Assigning a *group time-out* for similar services is a good idea when working with similar servers and expensive processes. One of the main reasons why this approach is useful is that Web hosting providers tend to limit the execution time of calls, depending on the contracted plan contracted.

We can see that in the plans offered by the hosting *1&1*. They offer four different plans for web hosting. These plans have prices ranging from 1.99€/month to 19.99€/month and giving 20 to 60 seconds on each call depending on the plan contracted. If all the services were contracted in *1&1* or hosting with similar plans, then we could have a fixed time-out of 65 seconds (5 extra seconds in order to avoid communication delays) or group the services in groups with time-outs of 25 to 65 seconds, and with that we ensure that if the service ends we get the response and otherwise do not wait too long.

As the servers time-outs are bigger than any possible network delay, then we can simply fix the time-out as the server time-out plus some margin for network delays (1 extra second for example). This time-out ensures 0 loses of packages, and therefore the probability of the

site calls will come from the probability that the process ends during the time given by the server. As servers can have more or less load, the execution time can vary greatly between executions, but we can extract statistically the probability that a execution ends in time.

As observation, this strategy can be used sometimes in order to reach 0% error probability by time if we can afford to pay a Web hosting with more execution time available that the needed in the worst case scenario.

A *specific time-out* for each service is good especially in the case where we have relatively fast services, as they finalize long before the server cuts the connection. In this case there are three aspects that matter:

- *Estimated processing time.* Each service need different time to process requests, also this time may vary a lot between requests. In general we have static and dynamic sites with similar or varied response length.
- *Ping.* The ping time is the time that a message needs to arrive and return from a server. That time caused mainly by the geographic distance and will be considered unavoidable.
- *Jitter.* The jitter is the average deviation in ping time. If the deviation on ping is great we may need to take it also into account.

In general, we can estimate this time-out from statistics on the services. We propose formulas to compute the time-out for a service in different situations, using a parameter $\delta > 0$ to weight deviation.

- On-line services

  If the service is on-line and we are able to access to it, then we may compute the time-out directly from the real service. We perform n random calls (this number may depend on the stability of the response time, but we propose 10.000 calls) and compute the average time (μ) and his deviation (σ). Then, we compute the time-out as follows:

  $$time\_out(\delta) = \mu + \sigma\delta$$

- Pre-computed

  When we are working with our services, in cases where we are unable to access to the on-line service for testing (restriction on number of calls, service still not uploaded,...), we may want to estimate the time-out based on the server capabilities and location. In order to compute the time-out without having access to the server, we are going to use the average ping (ρ) to the server, the estimated process time (μ') and deviation on process time (σ') for the server capabilities (in order to compute the process time we can do test in our machines and scale results, having into account the speed of the network). The formula is as follows:

  $$time\_out(\delta) = 2\rho + \mu' + \sigma'\delta$$

  Note that in *TCP* we need to establish the connexion, and therefore we need 2 times the ping.

- Pre-computed with high jitter When pre-computing time-outs, in the case where the jitter of the network (ϵ) is important we use the following formula:

$$time\_out(\delta) = 2\rho + \mu' + (\sigma' + \epsilon)\delta$$

In order to compute the average time and deviation, and finally the probability of failure, n calls to each server need to be done. In case of test of on-line services it may be important to perform those calls extended in time, in order to not bias the result. Also each call needs to be done without any data cache and knowing previously the IP direction of location of the service. When we have the n time values $x_1, ..., x_n$ we compute the average delay and deviation:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i, \quad \sigma = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - \mu)^2}$$

and the probability of failure is given by:

$$\mathsf{Pr}(\delta) = \frac{1}{n}\#\{x_i \mid x_i > time\_out(\delta)\}$$

In general, with services of the style *document provider*, which publish a document, a varied length on the response is normally what makes the deviation grow up and requires large time-outs. On the other hand, services of the *solver* style, that solves some problems, usually generate more or less deviation depending on the type of process. Meanwhile, the jitter is commonly neglected unless we deal with networks with very bad behaviour.

Example 2.2 Let us see the difference between three distinct existing sites. In order to compute the average time and deviation, and finally the probability of failure, During a week $n = 10000$ calls to each server were executed, recording the call time of each.

- Site *IPFW* is a service offering static pages [28]. This is static site that return a fixed size table with Spanish verbs conjugations. Each call was done with a random verb from a collection of 100 verbs.
- Service *Stackoverflow* [1] is a dynamic site for asking computing related questions. Each call was done at random from 100 questions chosen from the web page.
- *Yahoo* [2]. Calls where done to the main page, a dynamic page showing the last news from different sources and few other info.

For each server then we had:

	IPFW	*Stackoverflow*	*Yahoo*
μ	250ms	599ms	1585ms
σ	126ms	100ms	522ms

With those values and the test of each site, we compute the probability of failure for each site with a fixed time-out of $800ms$

	IPFW	*Stackoverflow*	*Yahoo*
Pr	0.984	0.973	0

and the time-out and probability of failure for cases with $\delta = 1$ and $\delta = 1.5$.

	IPFW	*Stackoverflow*	*Yahoo*
$Time\_out(1)$	376ms	699ms	2107ms
$\mathsf{Pr}(1)$	0.876	0.86	0.982
$Time\_out(1.5)$	439ms	749ms	2368ms
$\mathsf{Pr}(1.5)$	0.954	0.974	1

With these values we can see the big span between different time-outs. These values where computed with a relatively low number of calls in a short period of time. In order to have a better approximation to the behaviour of the server, a more exhaustive testing will be needed.

In addition to the probability of time-out, we can estimate the distribution of delays (the probability of having delays in different ranges). In chapter 8 this distribution will be used in order to compute an approximation to the distribution of minimum delay. In general, this distribution can be easily extracted from the log of calls.

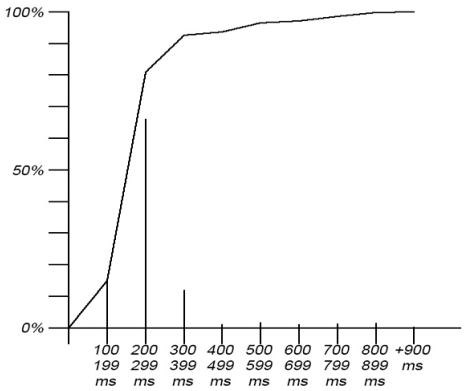
Example 2.3 Let us see which would be the distributions for the sites considered in the Example 2.2. With the same logs used for the probability the following distributions of delays are extracted, discretizing into groups of $100ms$ (See figure 2.3).

Further on, these distributions will be used in order to compute an approximation to the delay on the first published element. Moreover we can use these distribution as another method to compute service time-out in a way that it is easy to compute the time-out for which we lose at most $X\%$ of the good responses.

For computing the probability of failure with time-out, we need a big number of samples in order to have a good estimation. This is even more important when working with sites like *Yahoo* with large deviation on the call time.

Delay	*Prob.*
100 - 199 ms	0.153
200 - 299 ms	0.673
300 - 399 ms	0.120
400 - 499 ms	0.010
500 - 599 ms	0.015
600 - 699 ms	0.01
700 - 799 ms	0.012
800 - 899 ms	0.014
+900 ms	0.002

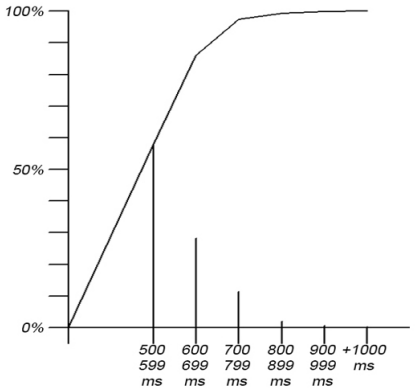
(a) Prob: IPFW



(b) Dist:IPFW

Delay	*Prob.*
500 - 599 ms	0.578
600 - 699 ms	0.282
700 - 799 ms	0.113
800 - 899 ms	0.019
900 - 999 ms	0.006
+1000 ms	0.002

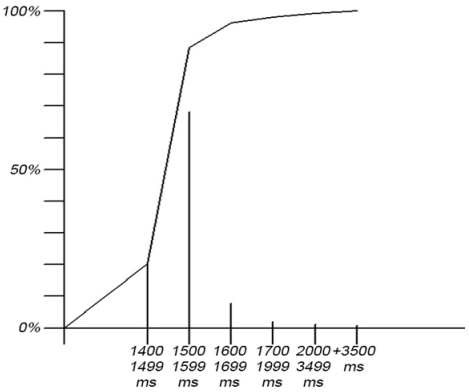
(c) Prob: Stackoverflow



(d) Dist:Stackoverflow

Delay	*Prob.*
1400 - 1499 ms	0.203
1500 - 1599 ms	0.681
1600 - 1699 ms	0.077
1700 - 1999 ms	0.019
2000 - 3499 ms	0.012
+3500 ms	0.008

(e) prob: Yahoo



(f) Dist: Yahoo

Figure 2.3: Probabilitites and Delay distributions for failure

Chapter 3

Orc Language and Sub-families

Orc is a language introduced by Misra and Cook [38] that can give a short and clear representation of orchestrations of sites in Grid applications. Now, let us have a small view of the Orc language.

Definition 3.1 An *Orc expression*, or simply *expression*, is a representation of an orchestration in Orc language. An expression can be represented by a name. Given an expression E and a list of parameters L, $E(L)$ describes an evaluation of the orchestration described by E with parameters L.

. . . . . .

The Orc language can be used as a simple extension to a sequential programming language. If E is the name of an Orc *expression* and L the list of actual parameters, an assignment of the output of E to a variable x can be expressed in the form $x :\in E(L)$. If the evaluation of $E(L)$ outputs one or more result, the first one is assigned to x. In case of no results, the statement execution does not terminate.

The basic unit of this language are the *Sites* in addition with a few number of operators together variables and constants. A *Site* is a the representation of a service or a function. Sites are also the minimum possible expressions of Orc.

3.1 Sites

Sites are the base of Orc expressions. During the evaluation of the expression, sites are called as procedures, returning one response or halting without responding the call.

Sites are represented by site names, starting with upper-case letters, and could be either functions (say, to compute the hash of a string), object methods (say, add a video file to an upload queue), a monitor procedure (say, to check if a net buffer is non-empty) or a web service (say, email web service). Also, an orchestration may involve humans in sites (like when an explicit approval is needed).

Sites can be called with different values as input of certain types, and then publish an output of a certain type, if not halts. If at least one variable of the input does not have a value assigned, a site call waits until all variables have some value assigned, and in the case that some variables never get a value the thread will halt.

Definition 3.2 A parametrised site S is *non-blocking* if $S(x_1, \ldots, x_n)$ must publish a result for all well-defined arguments $(x_1, \ldots, x_n) = (v_1, \ldots, v_n)$ [18], otherwise S is *potentially blocking*.

. . . . . .

Must be taken into account that, while it is possible that, even it the publishing type is be a tuple, a site call only publishes one value or none, but never more than one [38, p. 2]. It is important to remember this as otherwise that would be a source of parallelism, and site calls are not allowed to start new threads of computation inside the orchestration.

For sites performing functions, there is no change in the environment no change of status is caused. On the other hand, there are a lot of sites that may affect the behaviour of future calls to the same site or other sites. For example, two sites performing the actions of *Add* and *Get* some items from a queue, here is clear that *Add* calls affects directly *Get* calls.

For a site S, a site call with input $X = (x_1, \ldots, x_n)$ is described as $S(X)$. If a site S does not has input parameter, we can call it either as $S()$ or S.

3.1.1 Predefined Sites

In the Orc language, there are few predefined sites performing simple operations without the need of calls to external services. The Table 3.1 shows the basic predefined sites of the Orc language.

Site	**Input**	**Description**
0	None	It remains silent without publish any value. It has no other uses than stop execution threads without publishing any value.
1	X:Any	When called with parameters $X = \{x1, \ldots, x_n\}$, it publish the last value x_n. If called without input, it publish a 1.
If	b:Boolean	If $b = true/1$ then publish 1 otherwise halts.
Rtimer	t:Integer$^+$	It publish 1 after an elapse of t ms.

Table 3.1: List of predefined sites used in this document.

In addition, Orc define other predefined sites, like *Signal* an alias to 1. We are going to consider only predefined sites with the same behaviour in any call (given the input parameters). But, in Orc language there are also defined sites like *Clock* that publish the current time in milliseconds of the machine.

3.1.2 Service Sites

In addition to the predefined sites running in the local machine, an Orc expression can use external services as a sites. In general, any service can be represented by a site.

In Table 3.2 a description of some of the sites with *Deterministic Output* appearing in some examples is provided. Most of them are taken from [38].

Site	Meaning	Published Value
True	publishes the *true* value	`true/1`
False	publishes the $false$ value	`false/0`
$And(b_1, \ldots, b_n)$	publishes an and of the input	$b_1 \wedge \ldots \wedge b_n$
$Or(b_1, \ldots, b_n)$	publishes an or of the input	$b_1 \vee \ldots \vee b_n$
AddressAlice	publishes Alice's email address	`aalice`
AddressBob	publishes Bob's email address	`abob`
$Email(a, m)$	sends message m to email address a	`s_` + `a` + `_` + `m`
$EmailDad(m)$	sends message m to my father	`s_adad_` + `m`
$Print(m)$	sends message m to printer	`print_` + `m`
CNN	publishes a summary of the CNN news	`cnn`
BBC	publishes a summary of the BBC news	`bbc`

Table 3.2: Some examples of services sites.

For instance $Email($`aAlice`$,$ `cnn`$)$ publishes the string `s_aalice_cnn`.

3.2 Variables and Constants

Programs, in any languages, need at some point to store and re-use some kind of data during its the execution. Orc expressions are not an exception and they also need some variables and constants to fulfil their functionality. In the Orc language, variable names start with lower-case letters and have block scope, that means that a variable name can be re-used in different sub-expressions[1].

Definition 3.3 During the evaluation of an expression, the *Context* L contains all the actual parameters. It is a mapping from variables to their values. Given a expression E, $E(x = v)$ means evaluate E adding $x = v$ to the context.

As usual, variables and constant must have a type. On Orc, there are no type restriction on the type of variables, and in general, what is assumed is that any type of data can be

[1] In the original language, variables could be re-written, for simplification here we do not allow that possibility.

assigned to a variable. Common types used in Orc are Boolean values, integers and strings. An important type are *Signals*, a special data type with no distinct values, in case of been assigned to variables we consider that equivalent to a *true* value. Apart from these, values like images, video or even sites can be a possible kind, used for example by a discovery service.

As we want to obtain results concerning some performance measures computed beforehand, we are going to focus on orchestrations where the value of the variables is not important (the orchestration behaves the same whether one or other value is used). We also consider expressions where the data types involved are finite and with a relatively "small" domain. As the condition of having small domains is sometimes difficult to achieve, when we analyse orchestrations, sometimes we use *Descriptors* of the values instead of the actual values.

Definition 3.4 A *descriptor* is a short description for a set of values. A descriptor groups together these values that meet certain properties important for the orchestration.

For example, if an orchestration has images as values and some sites are interested in the image type and others in the relative size, then the descriptor `Image` can be a data type with domain in $\{jpgSmall, jpgMedium, jpgLarge, pngSmall, \ldots, gifLarge\}$.

. . . . . .

3.3 Operators

The Orc language is composed by operations between expressions, being sites the minimum form of expression. Since its introduction [38], Orc had 3 basic operators available to compose expressions. These operators are *Sequential composition*, *Parallel composition* and *Pruning*

- *Sequential composition* $A > x > B(x)$ and $A \gg B(x)$

  When evaluating $A > x > B(x)$ for a context L, for each value v published by A we evaluate $B(x = v)$, being the results published by B the results published by the full expression. In case that the values published by A are not used in B, we can write it as $A \gg B$, and in this case, we evaluate B for each value published by A.

- *Parallel composition* | $A \mid B$

  When evaluating $A \mid B$ for a context L, A and B are evaluated in parallel in separate threads, each with context L. The full expression publish the values published by the two sub-expressions.

- *Pruning* $A(x) < x < B$[2]

  When evaluating $A(x) < x < B$, for a context L, A and B are evaluated in parallel in separate threads, each with context L. 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 publish the values published the sub-expressions A.

[2]Initially, it was written as A where $x \in B$, this $A < x < B$ notation is a simplification of this.

The language has been extended with several new operators. We present two of them. The *Otherwise* operator, introduced by [31], and the *Signaling* introduced in [15].

- *Otherwise* $A; B$

  When evaluating $A; B$ for a context L, A is evaluated with context L. In case that A publish at least one value, the full expression publish the values published by A. In case that A does not publish any value, B is evaluated with context L and the full expression publish the values published by B.

  As "A does not publish" can be undecidable, we consider that A does not publish anything if passed some defined time A has not published yet.

- *Signaling* $A <!B$

  When evaluating $A <!B$ for a context L, B is evaluated with context L. When B publishes its first result, all the threads of B are killed and A is evaluated with context L. The full expression publish the values published the sub-expressions A. This operator originated as a synonym to the composition $(1(x) \gg A) < x < B$, where A does not use x

Given these 5 operators, the precedence of operators is the following

$$[> x >,\ \gg,\ |,\ ;,\ < x <,\ <!]$$

For example:

$$A \gg B \mid C < x < E; F \mid D = ((A \gg B) \mid C) < x < (E; (F \mid D))$$

Example 3.1 With these three operators we are able to describe a large variety of possible orchestrations. These are some examples:

$$\begin{array}{rcl} \mathit{LoopM}(k, M) & = & \mathit{If}(k > 0) \gg M() \gg \mathit{LoopM}(k-1, M)^3 \\ \mathit{SendNews} & = & (\mathit{AddressAlice} > a > \mathit{Email}(a, m)) \\ & & \quad < m < (\mathit{CNN} \mid \mathit{BBC}) \\ \mathit{KeepMailing}(t) & = & \mathit{Rtimer}(t) \gg \mathit{KeepMailing}(t) \mid \mathit{SendNews} \\ \mathit{SendSomething} & = & (\mathit{CNN} | \mathit{BBC}) > x > \mathit{EmailDad}(m) \end{array}$$

3.4 Evaluation

Orc expressions can be implemented in multiple ways. One implementation may be, an expression and an output point:

- Create a new thread and start evaluating the expression.
- When a | operator is found, generate two new threads. In one evaluate the left side and in the other the right side.

[3] Observe that is possible to have recursion in expressions.

- When encounter a ; operator, generate a new thread. Add an observer to the left side and execute it in the new thread. Start a timer, then if the left side has not publish anything when the timer stops, kill the thread of the left side (and its children) and evaluate the right side in it, otherwise continue evaluating the left side.[4]
- When encounter $< x <$ (or $<!$), generate a new thread. Add $x = \bot$ to the context of the left side and evaluate, then evaluate the right side with output point x. Add an observer to x and when it takes some value, kill the right thread (and its children).
- When encounter $> x >$ (or $\gg$), evaluate sequentially the right side for each value published by the left side.
- If there is no following expressions, publish the last value in the output point.

This possible implementation and other possibilities have the problem that are complex to analyse and may have poorly defined contexts.

Example 3.2 Let us analyse informally the following expression:

$$OneTwo = (If(b) \gg CNN \mid If(\neg b) \gg (CNN \mid BBC)) < b < (1(true) \mid 1(false))$$

When starting the evaluation, in two parallel threads expressions $(If(b) \gg CNN \mid If(\neg b) \gg (CNN \mid BBC))$ and $(1(1) \mid 1(0))$ are evaluated. The first expression create two new threads with the parallel operator and both wait for the variable b to take value.

The variable b takes value 0 or 1 non deterministically as this depends on what value is published first during the evaluation of $(1(1) \mid 1(0))$. Then, can be either 0 or 1.

When b takes value, the two threads waiting for it resume their execution. In case that b takes value $true$, the first one publish the value `cnn` and the other is killed as $If(\neg true)$ halts. In the other hand, if b takes value $false$, then is the first one the expression killed and the second one publish `cnn` and `bbc` according to the order they are published.

3.5 Orc Sub-families

Orc expressions can be really complex and difficult to analyse. Here, we are going to study only a small part of Orc language for which the behaviour can be somehow predicted. We are going to take into account only *Finite expressions*, those that does not have any recursion or iterations. Inside the Finite expressions, we are going to consider 8 different sub-families that share certain properties seen later on.

- ElementaryOrc. This sub-family of Orc includes expressions with only the operators $|$, $> x >$ and $\gg$.

  All site in the expression are non-blocking[5]. Site *If* is not used and site *Rtimer* must have a constant value as input. As an exception, site 0 can be used even being blocking.

[4] This is a possible implementations assuming that if within a certain time no value is published, then no value will be published with more time.

[5] Remember that a non-blocking site is such that always publish something independently on the input values.

- ExtElementaryOrc. Extension of ElementaryOrc allowing operators <! and ;.

  This sub-family of Orc includes expressions with the operators $|$, $> x >$, $\gg$, <! and ;.

  All site in the expression are non-blocking. Site *If* is not used and site *Rtimer* must have a constant value as input. As an exception, site 0 can be used even being blocking.

- PruningOrc. Extension of ElementaryOrc allowing operator $< x <$.

  This sub-family of Orc includes expressions with the operators $|$, $> x >$, $\gg$ and $< x <$.

  All site in the expression are non-blocking. Site *If* is not used and site *Rtimer* must have a constant value as input. As an exception, site 0 can be used even being blocking.

- ExtPruningOrc. Extension of ExtElementaryOrc and PruningOrc allowing all the operator.

  This sub-family of Orc includes expressions with all the operators seen.

  All site in the expression are non-blocking. Site *If* is not used and site *Rtimer* must have a constant value as input. As an exception, site 0 can be used even being blocking.

- IfOrc. This sub-family of Orc includes expressions with only the operators $|$, $> x >$ and $\gg$.

  Site in the expression are potentially blocking. All predefined sites seen can be used.

- ExtIfOrc. Extension of IfOrc allowing operators <! and ;.

  This sub-family of Orc includes expressions with the operators $|$, $> x >$, $\gg$, <! and ;.

  Site in the expression are potentially blocking. All predefined sites seen can be used.

- MixedOrc. Extension of IfOrc allowing operator $< x <$.

  This sub-family of Orc includes expressions with the operators $|$, $> x >$, $\gg$ and $< x <$. The composition $A(x) < x < B$ with the restriction that given a context L, the stream produced by evaluation of B with context L only produces instances of a value v determined by B and L.

  Given that, we assume that there is a function $F_B(L)$, such that for any context L, $v \in B(L) \Rightarrow v = F_B(L)$.

  Site in the expression are potentially blocking. All predefined sites seen can be used.

- ExtMixedOrc. Extension of MixedOrc allowing the operators <! and ;.

  This sub-family of Orc includes expressions with all the operators seen. Operator $< x <$ has the same restrictions as in MixedOrc.

  Site in the expression are potentially blocking. All predefined sites seen can be used.

Table 3.3 shows a resume of the 8 sub-families. The relation between the 8 sub-families can be seen in figure 3.1.

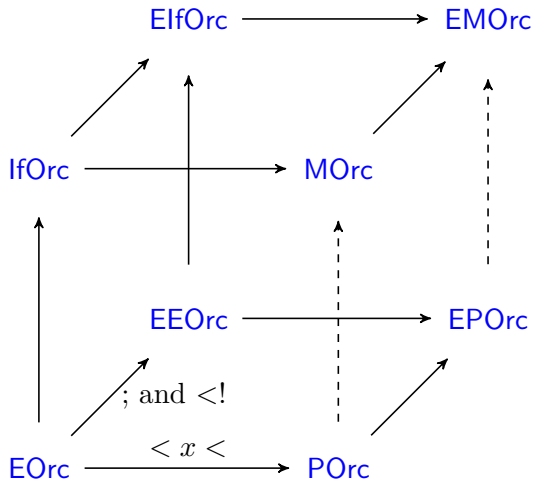


Figure 3.1: Visual representation of the relationships in Orc sub-families. Straight arrows indicates sub-family inclusion and da shed arrows indicate inclusion when restriction on pruning operator usage.

This relation between sub-families give following useful results:

Observation For any problem P, an upper-bound on the complexity of P for ExtMixedOrc implies at most the same upper bound for ExtPruningOrc, ExtIfOrc and MixedOrc. That is also true for MixedOrc to PruningOrc and IfOrc and so for the rest of sub-families.

Observation For any problem P, a lower-bound on the complexity of P for ElementaryOrc implies at least the same lower bound for ExtElementaryOrc, PruningOrc and IfOrc. That is also true for PruningOrc to ExtPruningOrc and MixedOrc and so for the rest of sub-families.

Sub-family	**Operators**	**Sites Available**
ElementaryOrc	$\|, > x >, \gg$	*Predefined sites:* 1, 0 and *Rtimer* (with constant parameters). *Service sites:* Non-blocking.
ExtElementaryOrc	$\|, > x >, \gg, ;, <!$	
PruningOrc	$\|, > x >, \gg, < x <$	
ExtPruningOrc	$\|, > x >, < x <, \gg, ;, <!$	
IfOrc	$\|, > x >, \gg$	*Predefined sites:* 1, 0,*If* and *Rtimer*. *Service sites:* Potentially blocking.
ExtIfOrc	$\|, > x >, \gg, ;, <!$	
MixedOrc	$\|, > x >, \gg, < x <$ (restricted)	
ExtMixedOrc	$\|, > x >, < x <$ (restricted), $\gg, ;, <!$	

Table 3.3: List of Orc sub-families.

Chapter 4

Number of outputs

In this chapter we are focusing in possibility of quantifying the number of values published by an expression when it is executed, provided that it can be defined. That tends to be a good measure of the goodness of an orchestration if we are not interested in get results fast but in get the maximum number of results. Here we introduce the fully defined variables model based on bags of values in order to be able to describe the behaviour of orchestrations abstracting from time. After that we see the relation between this model and the number of outputs and that for all the sub-families of Orc studied the number of outputs is the same in all the executions.

Finally, we are going to analyse some related problems to the number of outputs. On the bad side we are going to see that some of these problems have a high complexity, having NP-Hard and ♯P-Hard problems, but in the other hand we have seen that all the problems belongs to PSPACE, at least for the sub-families of Orc seen. Table 4.1 at the end of the chapter resumes the results found.

4.1 Fully Defined Variables Model

The semantics of Orc has been and is a rich research topic [27, 52, 46]. As we are interested in metrics related to the outputs produced by an orchestration, we consider an operational semantic so called *fully defined variables* model, a variation to the given by Misra and Cook [38]. 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* the possible streams can be described by a multi-set or bag $\lfloor\!\lfloor v_1, v_2, \ldots, v_n \rfloor\!\rfloor$. 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 $[\![E]\!] = \lfloor\!\lfloor v_1, v_2, \ldots, v_n \rfloor\!\rfloor$.

For orchestrations like $TwoCNN = (CNN \mid CNN)$, as we are working with multi-sets, it makes sense to write equalities like the following $[\![TwoCNN]\!] = \lfloor\!\lfloor \texttt{cnn} \rfloor\!\rfloor + \lfloor\!\lfloor cnn \rfloor\!\rfloor = \lfloor\!\lfloor \texttt{cnn}, \texttt{cnn} \rfloor\!\rfloor$. We also have that the pruning operator $< x <$ can give rise to a non deterministic behaviour.

Example 4.1 Remind the orchestrations given in Example 3.2 :

$$OneTwo = \begin{array}{l}(If(b) \gg CNN \mid If(\neg b) \gg (CNN \mid BBC)) \\ < b < (1(true) \mid 1(false))\end{array}$$

For this expression, the variable b could take either $true$ or $false$ as value. Then the $[\![b]\!] = \lfloor\!\lfloor \texttt{true} \rfloor\!\rfloor \sqcap \lfloor\!\lfloor \texttt{false} \rfloor\!\rfloor$. In case that b takes $true$, $OneTwo$ publish $\lfloor\!\lfloor \texttt{cnn} \rfloor\!\rfloor$ otherwise publish $\lfloor\!\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\!\rfloor$. Given that, we have that $[\![OneTwo]\!] = \lfloor\!\lfloor \texttt{cnn} \rfloor\!\rfloor \sqcap \lfloor\!\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\!\rfloor$.

4.1.1 Sites

Recall that we have considered that a call site returns a bag with cardinality 0 or 1. Site calls are dependent on the context, in particular, a site call $S(x_1, \dots x_n)$ need for all variables $x_1, \dots x_n$ to be defined. Let us provide the bags associated to predefined sites:

$$[\![1]\!] = \lfloor\!\lfloor 1 \rfloor\!\rfloor \qquad [\![1(x_1, \dots, x_n)]\!] = \begin{cases} \lfloor\!\lfloor x_n \rfloor\!\rfloor & \text{if } \forall i \in [1, \dots, n]; x_i \neq \bot \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & \text{if otherwise} \end{cases}$$

$$[\![0]\!] = \lfloor\!\lfloor\ \rfloor\!\rfloor \qquad [\![Rtimer]\!] = \lfloor\!\lfloor 1 \rfloor\!\rfloor \qquad [\![If(b)]\!] = \begin{cases} \lfloor\!\lfloor 1 \rfloor\!\rfloor & \text{if } b = \texttt{true} \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & otherwise \end{cases}$$

For non-predefined sites we consider 2 variations. Non-blocking sites and potentially blocking sites (Definition 3.2).

In the case of non-blocking sites. A site call $S(x_1, \dots x_n)$ with all variables define returns $s(x_1, \dots x_n)$, thus we have

$$[\![S(x_1, \dots x_n)]\!] = \lfloor\!\lfloor s(x_1, \dots, x_n) \rfloor\!\rfloor$$

In the case of potentially blocking sites. A site call $S(x_1, \dots x_n)$ with all variables define returns $s(x_1, \dots x_n)$ if $r(x_1, \dots x_n) = true$, thus we have

$$[\![S(x_1, \dots x_n)]\!] = \begin{cases} \lfloor\!\lfloor s(x_1, \dots, x_n) \rfloor\!\rfloor & \text{if } r(x_1, \dots x_n) = true \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & \text{otherwise} \end{cases}$$

In both cases, $s(x_1, \dots, x_n)$ and $r(x_1, \dots, x_n)$ need to be deterministic functions computable in poly-time with respect $(x_1, \dots, x_n)$.

4.1.2 Operations

With the fully defined variables model, now we can express the meaning of expressions defined as operations between bags. Following we describe the behaviour of the operators with the use of bags based on the behaviour on a normal execution.

- *Sequential composition* $> x >$ and $\gg$ $\qquad (A > x > B(x))$

$$[\![A > x > B(x)]\!] = \sqcap_{M \in [\![A]\!]} \sum_{v \in M} [\![B(x = v)]\!]$$

$$[\![A \gg B(x)]\!] = \sqcap_{M \in [\![A]\!]} \sum_{v \in M} [\![B]\!]$$

- *Parallel composition* | $(A \mid B)$

$$[\![A \mid B]\!] = [\![A]\!] + [\![B]\!]$$

- *Pruning* $< x <$ $(A(x) < x < B)$

$$[\![A(x) < x < B]\!] = \sqcap_{M \in [\![B]\!]} \begin{cases} \sqcap_{v \in M} [\![A(x = v)]\!] & \text{if } M \neq \lfloor\!\lfloor \rfloor\!\rfloor \\ [\![A(x = \bot)]\!] & \text{otherwise} \end{cases}$$

- *Otherwise* ; $(A; B)$

$$[\![A; B]\!] = \sqcap_{M \in [\![A]\!]} \begin{cases} [\![M]\!] & \text{if } M \neq \lfloor\!\lfloor \rfloor\!\rfloor \\ [\![B]\!] & \text{otherwise} \end{cases}$$

- *Signaling* $<!$ $(A <!B)$

$$[\![A(x) <!B]\!] = \sqcap_{M \in [\![B]\!]} \begin{cases} [\![A]\!] & \text{if } M \neq \lfloor\!\lfloor \rfloor\!\rfloor \\ \lfloor\!\lfloor\ \rfloor\!\rfloor & \text{otherwise} \end{cases}$$

Example 4.2 Let us see the orchestration *SendNews* (Example 3.1).

$$SendNews = (AddressAlice > a > Email(a, m)) < m < (CNN \mid BBC)$$

The evaluation of *SendNews* will go as follows.
First a call to $(CNN \mid BBC)$ with an empty context give the possible values of m.

$$[\![CNN \mid BBC]\!] = [\![CNN]\!] + [\![BBC]\!] = \lfloor\!\lfloor \texttt{cnn} \rfloor\!\rfloor + \lfloor\!\lfloor \texttt{bbc} \rfloor\!\rfloor = \lfloor\!\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\!\rfloor$$

Then m take a value from $\lfloor\!\lfloor \texttt{cnn}, \texttt{bbc} \rfloor\!\rfloor$, $t = \lfloor\!\lfloor \texttt{cnn} \rfloor\!\rfloor \sqcap \lfloor\!\lfloor \texttt{bbc} \rfloor\!\rfloor$. After that, $(AddressAlice > a > Email(a, m))$ is called.

$$[\![SendNews]\!] = \sqcap_{m \in \lfloor\!\lfloor \texttt{cnn},\texttt{bbc} \rfloor\!\rfloor} [\![(AddressAlice > a > Email(a, m))]\!]$$

- For $m = \texttt{cnn}$:
  $[\![(AddressAlice > a > Email(a, \texttt{cnn}))]\!] =$
  $\sum_{a \in [\![AddressAlice]\!]} [\![Email(a, \texttt{cnn})]\!]$
  $[\![AddressAlice]\!] = \lfloor\!\lfloor \texttt{aalice} \rfloor\!\rfloor$
  $[\![(AddressAlice > a > Email(a, \texttt{cnn}))]\!] =$
  $\sum_{a \in \lfloor\!\lfloor \texttt{aalice} \rfloor\!\rfloor} [\![Email(a, \texttt{cnn})]\!] =$
  $[\![Email(\texttt{aalice}, \texttt{cnn})]\!] = \lfloor\!\lfloor \texttt{s\_aalice\_cnn} \rfloor\!\rfloor$

- For $m = \texttt{bbc}$:
  $[\![(AddressAlice > a > Email(a, \texttt{bbc}))]\!] =$
  $\sum_{a \in [\![AddressAlice]\!]} [\![Email(a, \texttt{bbc})]\!]$
  $[\![AddressAlice]\!] = \{\!|\texttt{aalice}|\!\}$
  $[\![(AddressAlice > a > Email(a, \texttt{bbc}))]\!] =$
  $\sum_{a \in \{\!|\texttt{aalice}|\!\}} [\![Email(a, \texttt{bbc})]\!] =$
  $[\![Email(\texttt{aalice}, \texttt{bbc})]\!] = \{\!|\texttt{s\_aalice\_bbc}|\!\}$

Then $[\![SendNews]\!] = \{\!|\texttt{s\_aalice\_cnn}|\!\} \sqcap \{\!|\texttt{s\_aalice\_bbc}|\!\}$.

From the behaviour of sites and operators, the following holds:

Theorem 4.1.1. *Given a finite orchestration E is holds that $[\![E]\!] = \{\!| \, |\!\}$ or exists a unique non-deterministic finite decomposition in multi-sets $[\![E]\!] = \sqcap_{i=1}^{n} M_i$ being the elements of each M_i values returned by an execution of E, for $1 \leq i \leq n$.*

4.2 Number of outputs

In this chapter we are focusing in the number of values published by an expression when it is executed. Now let us see the relation between the fully defined variables model and the number of outputs.

Definition 4.1 Given a bag M, $\#M$ denotes the cardinality of M.

. . . . . .

As we have seen in Example 4.2, the different bags in $[\![E]\!]$ can have different number of items. We prove in this section that for the Orc sub-families introduced previously, they are guaranteed to produce outputs with the same number of items in all the executions. In order to prove that, first let us analyse the meaning of sub-expression for these families.

Theorem 4.2.1. *The meaning of $[\![E]\!]$ has a unique bag when restricting to expressions E in ExtIfOrc.*

Proof. It can be proved simply by the definitions of sites as generators of multi-sets and operators as operations between them.

Sites are the basic sub-expressions that we can found. By definition, sites with deterministic behaviour publish at most one value determined by the context of the site call and there is no demonic choice involved in that. Therefore, the meaning of a site call has a unique bag.

Now, let us see that no operator introduces non-determinism as they just keep it from their sub-expressions. By induction, as the basic sub-expressions, sites, does not add non-determinism, then the meaning of E does not has any demonic choice.

Given that we have that $[\![E]\!] = M$ for a unique bag M. □

The pruning operator is the unique operator adding demonic choices. Now let us analyse the sub-families that allow this operator.

Theorem 4.2.2. *The meaning of $[\![E]\!]$ is composed by bags with the same number of items when restricting to expressions E in ExtPruningOrc.*

Proof. In order to prove that, first let us remember that non-blocking sites always publish something if all the input values are defined. Given that, as we are considering only the number of items, we can consider all values as signals, only indicating that a variable is defined. In that case, we write $x = \bot$ if the variable x is undefined, and $x \neq \bot$ if it is defined. Now, let us analyse the operator $A(x) < x < B$ under that assumption:

In the base case, B is an ExtElementaryOrc expression. Given that, we have that $[\![B]\!]$ has a unique bag and therefore:

$$[\![A(x) < x < B]\!] = \begin{cases} [\![A(x \neq \bot)]\!] & \text{if } [\![B]\!] \neq \{\!|\ |\!\} \\ [\![A(x = \bot)]\!] & \text{otherwise} \end{cases}$$

Then, in the case that B is an ExtPruningOrc expression we have that as the abstraction from values removes the non-determinism of pruning for the base case, by induction in the general case there is also no non-determinism added with this operator, with respect the number of items.

Then we have that, while pruning introduces non-determinism on the values, if E is a ExtPruningOrc expression with meaning $[\![E]\!] = \sqcap_{i=1}^{n} M_i$, the cardinality of each bag is the same, $\#M_1 = \ldots \#M_n$. □

Theorem 4.2.3. *The meaning of $[\![E]\!]$ has a unique bag when restricting to expressions E in ExtMixedOrc.*

Proof. To prove this, let us analyse the behaviour of the operator $A(x) < x < B$ in ExtMixedOrc.

In the base case, B is an ExtIfOrc expression. Given that, we have that $[\![B]\!]$ has a unique bag. Also, the restriction imposed in ExtMixedOrc in order to allow the pruning operator ensures that either $[\![B]\!] = \{\!|\ |\!\}$ or $[\![B]\!] = \{\!|v, \ldots, v|\!\}$, for a value v determined by B and the context. Therefore, we have:

$$[\![A(x) < x < B]\!] = \begin{cases} [\![A(x = v)]\!] & \text{if } [\![B]\!] = \{\!|v, \ldots, v|\!\} \\ [\![A(x = \bot)]\!] & \text{otherwise} \end{cases}$$

Then, in the case that B is an ExtMixedOrc expression we have that as there is no non-determinism of pruning for the base case, by induction in the general case there is also no non-determinism added with this operator.

Given that we have that $[\![E]\!] = M$ for a unique bag M. □

Having meanings with a unique bag or all bags with the same number of items, now we can give a measure of the number of outputs of an expression.

Definition 4.2 For an expression E in ExtMixedOrc, $\mathsf{out}(E)$ denote the number of outputs produced by an execution of E.

If $[\![E]\!] = \sqcap_{i=1}^{n} M_i$, $\mathsf{out}(E) = \#M_1 = \ldots = \#M_n$.

Related to the number of outputs, we can be interested also in the number of outputs having some specific value.

Definition 4.3 For an expression E, the *domain* of E, $\mathbb{D}(E)$, is defined as the set of possible values $v \in [\![E]\!]$. If an orchestration E has sites $S_1, \ldots, S_n$ as endpoints, then $\mathbb{D}(E) \subseteq \cup_{i=1}^{n} \mathbb{D}(S_i)$.

. . . . . .

Definition 4.4 Given an expression E, we note as E^v to the expression E only allowing it to publish the value v. The expression E^v can be interpreted as $E > x > If(x = v)$. We use the don't care $*$ to denote that that the value does not matter, then $E = E^*$.

We note as $\mathsf{out}(E^v)$ the multiplicity of v in $[\![E]\!]$.

. . . . . .

Observe that given an expression E in ExtMixedOrc, for any value $v \in \mathbb{D}(E)$, $\mathsf{out}(E^v)$ is a well defined value. That is given by the fact that $[\![E]\!]$ is a unique bag $M = \lfloor\!\lfloor v_1, \ldots, v_n \rfloor\!\rfloor$. Given that :

$$\mathsf{out}(E^v) =\mid \{v_i \mid v_i = v;\ i \in [1, \ldots, n]\} \mid$$

4.2.1 Computing the out function

Now, let us see how to compute $\mathsf{out}(E)$ and $\mathsf{out}(E^v)$ for the studied sub-families of Orc.

For ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc we have that all sites are non-blocking. The following algorithm to compute $\mathsf{out}(E)$ comes directly from the evaluation model based on bags and the fact that in these families values does not matter to the number of outputs.

Lemma 4.2.4. *Let E be an ExtPruningOrc expression. For a given context L, $\mathsf{out}(E)$ can be computed recursively as follows.*

$$
\begin{array}{rcl}
\mathit{out}(0) & = & 0 \\
\mathit{out}(1(x)) & = & \begin{cases} 1 & \mathit{if}\ (x \in L) \\ 0 & \mathit{otherwise} \end{cases} \\
\mathit{out}(S(X)) & = & \begin{cases} 1 & \mathit{if}\ (\forall x_i \in X; x_i \in L) \\ 0 & \mathit{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{out}(A \mid B) & = & \mathit{out}(A) + \mathit{out}(B) \\
\mathit{out}(A \gg B) & = & \mathit{out}(A) * \mathit{out}(B) \\
\mathit{out}(A > x > B) & = & \mathit{out}(A) * \mathit{out}(B(x \neq \bot)) \\
\mathit{out}(A < x < B) & = & \begin{cases} \mathit{out}(A(x \neq \bot)) & \mathit{if}\ (\mathit{out}(B) > 0) \\ \mathit{out}(A(x = \bot)) & \mathit{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{out}(A; B) & = & \begin{cases} \mathit{out}(A) & \mathit{if}\ (\mathit{out}(A) > 0) \\ \mathit{out}(B) & \mathit{otherwise} \end{cases} \\
\mathit{out}(A <! B) & = & \begin{cases} \mathit{out}(A) & \mathit{if}\ (\mathit{out}(B) > 0) \\ 0 & \mathit{otherwise} \end{cases}
\end{array}
$$

Example 4.3 Let us consider the PruningOrc expression *SendNews* from the Example 3.1 :

$$
\begin{array}{rcl}
\mathit{SendNews} & = & (\mathit{AddressAlice} > a > \mathit{Email}(a, m)) \\
& & \quad < m < (\mathit{CNN} \mid \mathit{BBC})
\end{array}
$$

The expression can be rewritten as:

$$
\begin{array}{rcl}
A(m) & = & (\mathit{AddressAlice} > a > \mathit{Email}(a, m)) \\
B & = & (\mathit{CNN} \mid \mathit{BBC}) \\
\mathit{SendNews} & = & A < m < B
\end{array}
$$

We compute $\mathsf{out}(\mathit{SendNews})$ as follows:

$$
\mathsf{out}(\mathit{SendNews}) = \begin{cases} \mathsf{out}(A(x \neq \bot)) & \text{if } (\mathsf{out}(B) > 0) \\ \mathsf{out}(A(x = \bot)) & \text{otherwise} \end{cases}
$$

$$
\begin{array}{ll}
\mathsf{out}(B) & = \mathsf{out}(\mathit{CNN}) + \mathsf{out}(\mathit{BBC}) = 2 \\
\mathsf{out}(\mathit{SendNews}) & = \mathsf{out}(A(x \neq \bot)) \\
\mathsf{out}(A(x \neq \bot)) & = \mathsf{out}(\mathit{AddressAlice}) * \mathsf{out}(\mathit{Email}(\not\perp, \not\perp)) = 1 \\
\mathsf{out}(\mathit{SendNews}) & = 1
\end{array}
$$

For IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc, the use of potentially blocking sites makes values important to the number of outputs. In these cases, we can approach $\mathsf{out}(E)$ by recursive calls to $\mathsf{out}(E^v)$.

Lemma 4.2.5. *Let E be an ExtMixedOrc expression. $\mathit{out}(E)$ can be computed recursively as follows:*

$$
\mathit{out}(E) = \sum_{v \in \mathbb{D}(E)} \mathit{out}(E^v)
$$

That can seen directly by the fact that $[\![E]\!]$ is a unique bag and $\mathbb{D}(E)$ contains all the possible values on that bag.

Now, we need to be able to compute $\mathsf{out}(E^v)$. Remember that potentially blocking sites are described by two functions $r(X)$ and $s(X)$ that determine if a site call publish and the value published in that case. The following algorithm to compute $\mathsf{out}(E^v)$ comes directly from the evaluation model based on bags and the fact that in ExtMixedOrc there is no demonic choice involved even having pruning as we restrict it to produce only instances of a determined (Given $A(x) < x < B$ and a context L, we say that $v \in [\![B]\!] \Rightarrow F_B(L) = v$).

Lemma 4.2.6. *Let E be an ExtMixedOrc expression. For a given context L, $\mathsf{out}(E^v)$ can be computed recursively as follows:*

$$
\begin{aligned}
\mathsf{out}(0^v) &= 0 \\
\mathsf{out}(1(x)^v) &= \begin{cases} 1 & \text{if } (x \in L) \text{ and } (x = v) \\ 0 & \text{otherwise} \end{cases} \\
\mathsf{out}(If(b)^v) &= \begin{cases} 1 & \text{if } (b = true) \text{ and } (v = 1) \\ 0 & \text{otherwise} \end{cases} \\
\mathsf{out}(S(X)^v) &= \begin{cases} 1 & \text{if } (s(X) = x),\ (r(X) = true) \\ & \qquad \text{and } (\forall x_i \in X; x_i \in L) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{out}((A \mid B)^v) &= \mathsf{out}(A^v) + \mathsf{out}(B^v) \\
\mathsf{out}((A \gg B)^v) &= \mathsf{out}(A) * \mathsf{out}(B^v) \\
\mathsf{out}((A > x > B)^v) &= \textstyle\sum_{w \in \mathbb{D}(A)} \mathsf{out}(A^w) * \mathsf{out}(B(x = w)^v) \\
\mathsf{out}((A < x < B)^v) &= \begin{cases} \mathsf{out}(A(x = F_B(L))) & \text{if } (\mathsf{out}(B) > 0) \\ \mathsf{out}(A(x = \bot)) & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{out}((A; B)^v) &= \begin{cases} \mathsf{out}(A^v) & \text{if } (\mathsf{out}(A^v) > 0) \\ \mathsf{out}(B^v) & \text{otherwise} \end{cases} \\
\mathsf{out}((A <! B)^v) &= \begin{cases} \mathsf{out}(A^v) & \text{if } \mathsf{out}(B) > 0 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Example 4.4 Let us consider the IfOrc expression *SendSomething* from the Example 3.1 :

$$SendSomething = (CNN|BBC) > x > EmailDad(m)$$

The expression can be rewritten as:

$$
\begin{aligned}
A &= (CNN \mid BBC) \\
B(m) &= EmailDad(m) \\
SendSomething &= A > m > B
\end{aligned}
$$

For $v = \texttt{s\_adad\_cnn}$, we compute $\mathsf{out}(\mathit{SendSomething}^v)$ as follows:

$$\begin{array}{ll}
\mathsf{out}(\mathit{SendSomething}^v) & = \sum_{w \in \mathbb{D}(A)} \mathsf{out}(A^w) * \mathsf{out}(B(x = w)^v) \\
\mathbb{D}(A) = \{\texttt{cnn}, \texttt{bbc}\} & \\
\mathsf{out}(\mathit{SendSomething}^v) & = \mathsf{out}(A^{\texttt{cnn}}) * \mathsf{out}(B(x = \texttt{cnn})^v) \\
 & \quad + \mathsf{out}(A^{\texttt{bbc}}) * \mathsf{out}(B(x = \texttt{bbc})^v) \\
\mathsf{out}(B(x = \texttt{cnn})^v) & = 1 \\
\mathsf{out}(A^{\texttt{cnn}}) & = \mathsf{out}(\mathit{CNN}^{\texttt{cnn}}) + \mathsf{out}(\mathit{BBC}^{\texttt{cnn}}) = 1 + 0 = 1 \\
\mathsf{out}(B(x = \texttt{bbc})^v) & = 0 \\
\mathsf{out}(\mathit{SendSomething}^v) & = 1 * 1 + 0 = 1
\end{array}$$

For $v = \texttt{s\_adad\_bbc}$ we have the same result. Then, it holds that:

$$\mathsf{out}(\mathit{SendSomething}) = 2 =$$

$$\mathsf{out}(\mathit{SendSomething}^{\texttt{s\_adad\_cnn}}) + \mathsf{out}(\mathit{SendSomething}^{\texttt{s\_adad\_bbc}})$$

4.3 Related Problems

Having a well defined output length measure, there are some related computational problems of interest. For a given expressions E in any of the studied sub-families given in chapter 3:

- **Output**. Given an expression E, compute the number of items on a the bags of $[\![E]\!]$ ($\mathsf{out}(E)$).
- **Produces**. Given an expression E, determine if the bags of $[\![E]\!]$ contain at least one value ($\mathsf{out}(E) > 0$).

And for expressions publishing a unique bag:

- **OutputValue**. Given an expression E and a value v, compute the multiplicity of v in $[\![E]\!]$ ($\mathsf{out}(E^v)$).
- **ProducesValue**. Given an expression E and a value v, determine if the value v appear on $[\![E]\!]$ ($\mathsf{out}(E^v) > 0$).

Observer that all these problem have a well defined solution as seen before. Now let us show some complexity bounds for the different problems and sub-families. First, let us define some useful gadgets.

Definition 4.5 For a finite Orc expression E with n operators $|$, $< x <$ and $;$, we define a *trace* $t \in \{0,1\}^{\leq n}$ as an execution path on the evaluation of E.

A trace describe an execution path in the following way. When encountering the i-th operator during the execution, i-th position of the trace decides how to proceed. In case of operators $|$ and $;$, the trace decides if the left side or the right side is evaluated. For the

$< x <$ operator, it decides if the right side is evaluated first, or if the left side is the only one evaluated, and then $x = \bot$.

We denote $\mathsf{paths}(E) \subseteq \{0,1\}^{\leq n}$ the set of valid execution paths. Given $t \in \mathsf{paths}(E)$ we note by $\mathsf{val}(t)$ the return value due to this path. Observe that $\mathsf{val}(t) = \lfloor\lfloor v \rfloor\rfloor$, if the execution path publishes a value v, or $\mathsf{val}(t) = \lfloor\lfloor\ \rfloor\rfloor$, in the case that the execution path returns nothing. Note that both $t \in \mathsf{paths}(E)$ and $\mathsf{val}(t)$ can be computed in poly-time with respect to the size of E.

Given a trace t and $s \in \{0,1\}^{|t|}$, s is a *sub-trace* of t ($s \backslash t$) if s describes the execution of t without considering entering on pruning. We denote $\mathsf{subpaths}(E) \subseteq \{0,1\}^{\leq n}$ the set of valid sub-paths s. Given s and t, $s \backslash t$ can be checked in poly-time simply following the traces.

. . . . . .

4.3.1 The complexity of problems Output & Produces

For ElementaryOrc and PruningOrc these two problems has been previously studied an proved to pertain to P [15]. Now let us see that this results also hold for ExtElementaryOrc and ExtPruningOrc.

Theorem 4.3.1. *The problem Output belongs to P when restricted to ExtPruningOrc.*

Proof. Directly from the formula of how to compute out for these sub-families. With a quick look, we see that each operator ask for the out(E) of its sub-expressions at most one time. As that is true for all operators and for sites we can solve it in poly-time in the number of variables, then we can compute $\mathsf{out}(E)$ in poly-time. □

Theorem 4.3.2. *The problem Produces belongs to P when restricted to ExtPruningOrc.*

Proof. Directly from Output ∈ P, as $E \in \mathsf{Produces} \Leftrightarrow \mathsf{out}(E) > 0$. □

For IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc the results are not as good as in this case. We have that solve Produces is as hard as to solve SAT.

Theorem 4.3.3. *The problem Produces is NP-Complete when restricted to IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc.*

Proof. First, let us see that the problem is NP. With the use of traces, we have that Produces can be seen as the NP problem:

$$E \in \mathsf{Produces} \Leftrightarrow \exists t \in \{0,1\}^{\leq n} \text{ s.t. } (t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) \neq \bot).$$

Now, in order to prove that it is also NP-Hard, let us consider a reduction form 3-SAT.

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$. Let us encode each one of the cause through calls to site *If*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

$$E_{C_i}(y_{i_a}, y_{i_b}, y_{i_c}) = (If(y_{ia}) \mid If(y_{ib}) \mid If(y_{ic}))$$

Taking $True = 1(1)$ and $False = 1(0)$ the orchestration E_F associated to F is:

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

Note that for $(v_1, \ldots, v_n) \in \{0, 1\}^n$ we have $E(v_1, \ldots, v_n) \in \mathsf{Produces} \Leftrightarrow F(v_1, \ldots v_n) = 1$. As $(x_1, \ldots, x_n)$ takes all possible values in $\{0, 1\}^n$ by construction of E_F, we have:

$$\begin{array}{c} F \in 3 - \mathsf{SAT} \Leftrightarrow \exists (v_1, \ldots, v_n) \in \{0, 1\}^n \text{ s.t. } (F(v_1, \ldots, v_n) = 1) \Leftrightarrow \\ \exists (v_1, \ldots, v_n) \in \{0, 1\}^n \text{ s.t. } (E(v_1, \ldots, v_n) \in \mathsf{Produces}) \Leftrightarrow \mathsf{Produces}(E_F) \end{array}$$

Then, as the problem is NP and NP-Hard, it is NP-Complete. □

For Output we have that for IfOrc and MixedOrc that this problem is ♯P-Complete and ♯.NP-Complete.

Theorem 4.3.4. *The problem Output is ♯P-Complete when restricted to IfOrc.*

Proof. In order to prove that Output for an expression E in IfOrc is ♯P-Complete, first let us see that this problem is in ♯P. For that, let us reuse the proof of Produces(E) in NP. For an expression E we had that

$$E \in \mathsf{Produces} \Leftrightarrow \exists t \in \{0, 1\}^n \text{ s.t. } (t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) \neq \llbracket \rrbracket).$$

In this case, each valid certificate corresponds to an output of E, therefore

$$\mathsf{out}(E) = \|\{t \in \{0, 1\} \mid (\ t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) \neq \llbracket \rrbracket\ )\ \}\|$$

and that is a typical ♯P problem.

Now, in order to proof that it is also ♯P-Hard, let us consider a different reduction form 3-SAT to the used for Produces.

Given $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$. Let us encode each one of the cause through calls to site *If*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

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

Taking $True = 1(1)$ and $False = 1(0)$ the orchestration E_F corresponding to the formula F is:

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

Note that now for $(v_1, \ldots, v_n) \in \{0, 1\}^n$ we have $F(v_1, \ldots v_n) = 1 \Leftrightarrow \mathsf{out}(E(v_1, \ldots, v_n)) = 1$. As $(v_1, \ldots, v_n)$ takes all possible values in $\{0, 1\}^n$ by construction of E_F, we have:

$$\begin{array}{c} \sharp F = \sum_{(v_1, \ldots, v_n) \in \{0,1\}^n} F(v_1, \ldots, v_n) = \\ \sum_{(v_1, \ldots, v_n) \in \{0,1\}^n} \mathsf{out}(E(v_1, \ldots, v_n)) = \mathsf{out}(E_F) \end{array}$$

As the problem is ♯P and ♯P-Hard, it is ♯P-Complete. □

Theorem 4.3.5. *The problem Output is ♯.NP-Complete when restricted to MixedOrc.*

Proof. In order to prove that Output for an expression E in MixedOrc is ♯.NP-Complete, first let us see that this problem is in ♯.NP. In this case, the definition of traces used to proof Produces was NP does not work, as we can have multiple traces publishing one value when pruning. What we can do now is use sub-traces in order to count without taking the multiple possibilities of pruning into account. Given that, we can describe the problem Produces as follows:

$$E \in \mathsf{Produces} \Leftrightarrow \exists t \in \{0,1\}^{\leq n}, \exists s \in \{0,1\}^{\leq n} \text{ s.t. } (s\backslash t,\ t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) \neq \llparenthesis\ \rrparenthesis).$$

In this case, each sub-trace s for witch exist trace t publishing such that $s\backslash t$ corresponds to an output of E, therefore

$$\mathsf{out}(E) = \|\{s \in \mathsf{subpaths}(E) \mid \exists t \in \mathsf{paths}(E) \text{ s.t. } (s\backslash t \text{ and } \mathsf{val}(t) \neq \llparenthesis\ \rrparenthesis)\ \}\|$$

and that is a typical ♯.NP problem.

Now, in order to proof that it is also ♯.NP-Hard, let us consider the following reduction from counting with oracle in 3-SAT.

Given $F = C_1 \wedge \ldots \wedge C_m$ with $n+m$ literals $x_1, \ldots, x_n, x'_1, \ldots, x'_m$, 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 $1 \leq i \leq n$. Let us encode each one of the cause through calls to site *If*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

$$E_{C_i}(y_{i_a}, y_{i_b}, y_{i_c}) = (If(y_{ia}) \mid If(y_{ib}) \mid If(y_{ic}))$$

Taking $True = 1(1)$ and $False = 1(0)$ the orchestration E_F corresponding to the formula F is:

$$\begin{aligned}
&Double = (True \mid False)\\
&E(x_1, \ldots, x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \gg \ldots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c})\\
&E' = 1(w) < w < (Double > x'_1 > \cdots > Double > x'_m >\\
&\qquad E(x_1, \ldots, x_n, x'_1, \ldots, x'_m))\\
&E_F = Double > x_1 > \cdots > Double > x_n > E'(x_1, \ldots, x_n)
\end{aligned}$$

Note that now for $(v_1, \ldots, v_n, v'_1, \ldots, v'_m) \in \{0,1\}^{n+m}$ we have $F(v_1, \ldots v_n, v'_1, \ldots, v'_m) = \mathsf{out}(E(v_1, \ldots, v_n, v'_1, \ldots, v'_m))$. As we have that $(v'_1, \ldots, v'_m)$ takes all possible values in $\{0,1\}^m$ inside a pruning expression, we have that for each $(v_1, \ldots, v_n) \in \{0,1\}^n$:

$$\begin{gathered}
\mathsf{out}(E'(v_1, \ldots, v_n)) = 1 \Leftrightarrow\\
\exists (v'_1, \ldots, v'_m) \in \{0,1\}^m \text{ s.t. } (\mathsf{out}(E(v_1, \ldots, v_n, v'_1, \ldots, v'_m)) = 1) \Leftrightarrow\\
\exists (v'_1, \ldots, v'_m) \in \{0,1\}^m \text{ s.t. } (F(v_1, \ldots, v_n, v'_1, \ldots, v'_m) = 1) \Leftrightarrow\\
\text{Given fixing } (x_1, \ldots, x_n) = (v_1, \ldots, v_n),\ F \in 3-\mathsf{SAT}
\end{gathered}$$

As $(v_1, \ldots, v_n)$ takes all possible values in $\{0,1\}^n$ by construction of E_F, we have that $\mathsf{out}(E_F)$ is the result of counting with oracle in 3-SAT.

As the problem is ♯.NP and ♯.NP-Hard, it is ♯.NP-Complete.

□

For ExtIfOrc and ExtMixedOrc, the operator ; does not allow the completeness on the counting problems, as we need to know if the left side publish or not in order to get the out from the left or right side, and that cannot be done by certificates.

Theorem 4.3.6. *The problem Output belongs to PSPACE when restricted to ExtIfOrc and ExtIfOrc.*

Proof. Remember that we can compute $\mathsf{out}(E)$ as the sum of $\mathsf{out}(E^v)$ for each possible value $v \in \mathbb{D}(E)$. From the algorithm to compute $\mathsf{out}(E^v)$ we can see that it is a recursive algorithm with depth inferior to $|E|$ and in each level we only need to store a fixed amount of data. Given that, we can compute recursively $\mathsf{out}(E^v)$ in poly-space on the size of E. Then we can compute $\mathsf{out}(E)$ simply summing recursively $\mathsf{out}(E^v)$ for each value $v \in \mathbb{D}(E)$ within poly-space. □

4.3.2 The complexity of problems OutputValue & ProducesValue

As we have seen for Output and Produces, when values are involved, the problems tend to increment its complexity. I this case, for OutputValue and ProducesValue the filtering to a unique output value increments considerable the complexity in some cases.

For ProducesValue, we have that for all families the problem becomes as difficult to solve SAT.

Theorem 4.3.7. *The problem ProducesValue is NP-Complete when restricted to ElementaryOrc, ExtElementaryOrc, IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc.*

Proof. First, let us see that the problem is NP. With the use of traces, we have that ProducesValue can be seen as the NP problem:

$$< E, v > \in \mathsf{ProducesValue} \Leftrightarrow \exists t \in \mathsf{paths}(E) \text{ s.t. } (\mathsf{val}(t) = \lfloor\!\lfloor v \rfloor\!\rfloor).$$

Now, in order to proof that it is also NP-Hard let us remind the definition of non-blocking sites. For those sites, we have that a site call $S(v_1, \dots v_n)$ will publish always a value determined by $s(v_1, \dots v_n)$. For $E \in \mathsf{Produces}$ and $\mathsf{out}(E)$, the value published does not matter, but in this case that is different.

Given that, let us consider a reduction form 3-SAT. Given a 3-CNF formula F with n literals $x_1, \dots, x_n$, let the site $S_F(x_1, \dots, x_n)$ be defined by the following function $s(x_1, \dots, x_n)$.

$$s(v_1, \dots, v_n) = \begin{cases} success & \text{if } F(v_1, \dots, v_n) = 1 \\ fail & \text{otherwise} \end{cases}$$

Note that s is a valid polynomial function describing a site. Now, define the expression associated to F, E_F, as follows:

$$Double = (True \mid False)$$
$$E_F = Double > x_1 > \cdots > Double > x_n > S_F(x_1, \dots, x_n)$$

$\mathsf{Produces}({E_F}^{success})$ implies that exist a trace $t = (t_1, \dots, t_n) \in \{0,1\}^n$ such that $val(t) = \lfloor\!\lfloor success \rfloor\!\rfloor$. By construction, a trace $t = (t_1, \dots, t_n) \in \{0,1\}^n$ makes the variables $(x_1, \dots, x_n)$ take values $(t_1, \dots, t_n)$. Therefore, $val(t) = \lfloor\!\lfloor success \rfloor\!\rfloor \Leftrightarrow F(t_1, \dots, t_n) = true$.

See that E_F is an ElementaryOrc expression, and for the relation between sub-families, this result is also valid for ExtElementaryOrc, IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc.

Then, as the problem is NP and NP-Hard, it is NP-Complete. □

For OutputValue on ElementaryOrc, IfOrc and MixedOrc we found complexities related to counting problems.

Theorem 4.3.8. *The problem OutputValue is ♯P-Complete when restricted to ElementaryOrc and IfOrc.*

Proof. First, let us see that the problem is ♯P. For that, let us reuse the proof of ProducesValue in NP. For an expression E we had that

$$< E, v > \in \mathsf{ProducesValue} \Leftrightarrow \exists t \in \{0,1\}^n \text{ s.t. } (t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) = \lfloor\!\lfloor v \rfloor\!\rfloor).$$

In this case, each valid certificate corresponds to an output of E, therefore

$$\mathsf{out}(E^v) = \|\{t \in \{0,1\} \mid (\ t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) = \lfloor\!\lfloor v \rfloor\!\rfloor\ )\ \}\|$$

and that is a typical ♯P problem.

Now, in order to proof that it is also ♯P-Hard, let us re-use the expression used in order to prove ProducesValue is NP-Hard

Given a 3-SAT formula F with n literals $x_1, \ldots, x_n$, let the site $S_F(x_1, \ldots, x_n)$ be defined by the following function $s(x_1, \ldots, x_n)$.

$$s(v_1, \ldots, v_n) = \begin{cases} success & \text{if } F(v_1, \ldots, v_n) = 1 \\ fail & \text{otherwise} \end{cases}$$

Now, define the expression E_F as follows:

$$Double = (True \mid False)$$
$$E_F = Double > x_1 > \cdots > Double > x_n > S_F(x_1, \ldots, x_n)$$

By construction, $\mathsf{out}(E_F^{success}) = \#F$ as for each trace $t = (t_1, \ldots, t_n) \in \{0,1\}^n$, $\mathsf{val}(t) = \lfloor\!\lfloor success \rfloor\!\rfloor \Leftrightarrow F(t_1, \ldots, t_n) = true$, and, by construction, each trace describe an output value.

For the relation of sub-families, this result is also valid for IfOrc.

Then, as the problem is ♯P and ♯P-Hard, it is ♯P-Complete. □

Theorem 4.3.9. *The problem OutputValue is ♯.NP-Complete when restricted to MixedOrc.*

Proof. In order to prove OutputValue of MixedOrc is ♯.NP-Complete, first let us see that this problem is in ♯.NP.

With the same double trace t and s used to prove out ∈♯P for MixedOrc, were $t \in \mathsf{paths}(E)$ was a trace of E and s was a sub-trace of t describing the execution without entering on pruning. We can describe the problem ProducesValue as follows:

$$< E, v > \in \mathsf{ProducesValue} \Leftrightarrow$$
$$\exists t \in \{0,1\}^{\leq n}, \exists s \in \{0,1\}^{\leq n} \text{ s.t. } (s\backslash t,\ t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) = \lfloor\!\lfloor v \rfloor\!\rfloor).$$

In this case, each certificate s for witch exist a valid certificate t publishing v such that $s\backslash t$ describes a published value v, therefore

$$\mathsf{out}(E^v) = \|\{s \in \{0,1\}^{\leq n} \mid (\ \exists s \in \{0,1\}^{\leq n} \text{ s.t. } ((s\backslash t,\ t \in \mathsf{paths}(E) \text{ and } \mathsf{val}(t) = \lfloor\!\lfloor v \rfloor\!\rfloor)\ )\ \}\|$$

and that is a typical ♯.NP problem.

Now, in order to proof that it is also ♯.NP-Hard, let us consider the following reduction from out. Given an MixedOrc expression E, define F as follows:

$$F = E \gg 1(success)$$

Then, by construction $\mathsf{out}(F^{success}) = \mathsf{out}(E)$. As $\mathsf{out}(E)$ is ♯.NP-Complete for MixedOrc, then OutputValue is ♯.NP-Hard for MixedOrc.

Then, as the problem is ♯.NP and ♯.NP-Hard, it is ♯.NP-Complete. □

For ExtElementaryOrc, ExtIfOrc and ExtMixedOrc we have the same problems in order to get completeness, as the operator ; need to know if the left side publish or not in order to get the out from the left or right side, and that cannot be done by certificates.

Theorem 4.3.10. *The problem OutputValue belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. From the algorithm to compute $\mathsf{out}(E^v)$ we can see that it is a recursive algorithm with depth inferior to $|E|$ and in each level we only need to store a fixed amount of data. Given that, we can compute recursively $\mathsf{out}(E^v)$ in poly-space on the size of E. □

	Output	**Produces**	**OutputValue**	**ProducesValue**
ElementaryOrc	P	P	♯P-C	NP-C
ExtElementaryOrc	P	P	♯P-H	NP-C
PruningOrc	P	P	*not included*	*not included*
ExtPruningOrc	P	P	*not included*	*not included*
IfOrc	♯P-C	NP-C	♯P-C	NP-C
ExtIfOrc	♯P-H	NP-C	♯P-H	NP-C
MixedOrc	♯.NP-C	NP-C	♯.NP-C	NP-C
ExtMixedOrc	♯.NP-H	NP-C	♯.NP-H	NP-C

Table 4.1: Resume of problems related to the number of outputs. All problems belong to PSPACE.

Chapter 5

Delay of output values

This chapter focuses in the delay of values published by the execution of an expression. That tends to be a good measure of the goodness of an orchestration if we are more interested in get results fast that in get a lot of results. Here we expand the fully defined variables model based on bags described in Chapter 4 into the fully defined variables-delay model in order to take execution time into account. After that we see the relation between this model and the delay of outputs and that for all the sub-families of Orc studied the values are published with the same delays in all the executions.

Finally, we are going to analyse some related problems to the delay of outputs. On the bad side we are going to see that some of these problems have a high complexity, having NP-Hard and ♯P-Hard problems, but in the other hand we have seen that all the problems belongs to PSPACE, at least for the sub-families of Orc seen. Table 5.1 at the end of the chapter resumes the results found.

5.1 Fully Defined Variables-Delay Model

In the fully defined variables model we abstracted from execution time describing the possible output streams as bags of values. Given that model, we where able to describe the output of an execution without the complexity the threads add. However, when we want to define measures related to the execution time, the fully defined variables model is not useful.

In order to deal up to some extent with time, we extend the fully defined variables model adding the required time for publish each value. We name this new model *fully defined variables-delay* model. In this model, bags not only contain the published values but also their associated delay.

We assume delays are measured with milliseconds, and therefore are non-negative integers. We denote "never returns" by the special symbol ω then taking $\delta_i = \omega$ means "infinite" delay. Assume $\delta + \omega = \omega + \omega = \omega$ and $min(\delta, \omega) = \delta$.

When an orchestration E publishes a stream $v_1, v_2, \ldots, v_n$, each value v_i have a delay δ_i from the start of the execution to its publication. The possible streams can be described by a multi-set or bag $\{\!|\langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle|\!\}$. In such a case, with an abuse of the language, the "meaning" of E is the bag $\{\!|\langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle|\!\}$.

We consider the same operation + between multi-sets and the demonic choice operator. Also, we assume that for any bag $M = \{\!|\langle v_1 : \delta_1\rangle, \ldots, \langle v_n : \delta_n\rangle|\!\}$, we remove any value v_i

with delay $\delta_i = \omega$, as having delay ω means that the value never will be published.

We need to introduce a new operator between multi-sets. The "add delay" operator $\oplus$ to denote an addition of delay to each value of a bag. It is defined as:

$$\begin{array}{c} \{\!|\langle v_1 : \delta_1\rangle, \ldots, \langle v_m : \delta_m\rangle|\!\} \oplus \{\!|\langle w_1 : \delta'_1\rangle, \ldots, \langle w_n : \delta'_n\rangle|\!\} \\ = \{\!|\langle w_1 : (\delta_1 + \delta'_1), w_1 : (\delta_2 + \delta'_1)\rangle, \ldots, \langle w_n : (\delta_m + \delta'_n)\rangle|\!\} \end{array}$$

We can also use it to add delay directly as follows:

$$\delta \oplus \{\!|\langle v_1 : \delta_1\rangle, \ldots, \langle v_m : \delta_m\rangle|\!\} = \{\!|\langle v_1 : (\delta_1 + \delta)\rangle, \ldots, \langle v_m : (\delta_m + \delta)\rangle|\!\}$$

As examples:

$$\begin{array}{c} \{\!|\langle \mathtt{v_1} : 100\rangle, \langle \mathtt{v_2} : 200\rangle|\!\} \oplus \{\!|\langle \mathtt{cnn} : 500\rangle, \langle \mathtt{bbc} : 600\rangle|\!\} \\ = \{\!|\langle \mathtt{cnn} : 600\rangle, \langle \mathtt{cnn} : 700\rangle, \langle \mathtt{bbc} : 700\rangle, \langle \mathtt{bbc} : 800\rangle|\!\} \end{array}$$

$$1000 \oplus \{\!|\langle \mathtt{cnn} : 500\rangle, \langle \mathtt{bbc} : 600\rangle|\!\} = \{\!|\langle \mathtt{cnn} : 1500\rangle, \langle \mathtt{bbc} : 1600\rangle|\!\}$$

When abstracting from the values, we define 3 different measures related to the delays:

Definition 5.1 Given a multi-set $M = \{\!|\langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle|\!\}$, we introduce the *first item delay* value $\mathsf{first}(M)$ to denote $min(\delta_1, \delta_2, \ldots, \delta_n)$. As an example

$$\mathsf{first}(\{\!|\langle \mathtt{cnn} : 500\rangle, \langle \mathtt{bbc} : 600\rangle|\!\}) = 500$$

We assume that $\mathsf{first}(\{\!|\,|\!\}) = \omega$

. . . . . .

Definition 5.2 Given a multi-set $M = \{\!|\langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle|\!\}$, we introduce the *last item delay* value $\mathsf{last}(M)$ to denote $max(\delta_1, \delta_2, \ldots, \delta_n)$. As an example

$$\mathsf{last}(\{\!|\langle \mathtt{cnn} : 500\rangle, \langle \mathtt{bbc} : 600\rangle|\!\}) = 600$$

We assume that $\mathsf{last}(\{\!|\,|\!\}) = \omega$

. . . . . .

Definition 5.3 Given a multi-set $M = \{\!|\langle v_1 : \delta_1\rangle, \langle v_2 : \delta_2\rangle, \ldots, \langle v_n : \delta_n\rangle|\!\}$, we introduce the *average item delay* value $\mathsf{avgDelay}(M)$ as $\sum_{i=0}^{n} \delta_i / \#M$. As an example

$$\mathsf{avgDelay}(\{\!|\langle \mathtt{cnn} : 500\rangle, \langle \mathtt{bbc} : 600\rangle|\!\}) = 550$$

We assume that $\mathsf{avgDelay}(\{\!|\,|\!\}) = \omega$

. . . . . .

5.1.1 Sites

Under this semantics, a site call can be seen as an expression publishing one bag either with empty or with one item and delay distinct to ω. Site calls are dependent on the context, in particular, a site call $S(x_1, \ldots x_n)$ need for all variables $x_1, \ldots x_n$ to to be defined in order to publish something.

As variables defined by pruning take value in a parallel execution, they may have an associated delay in case that they are not defined at the moment of a site call. These delays are stored in the context with the value of the variables and may affect the delay of a site call. In particular, a site call $S(x_1, \ldots x_n)$ need for all variables $x_1, \ldots x_n$ to have value in order to publish something, if those variables takes delays $\delta_1, \ldots, \delta_n$ until they are defined, if the site call publishing the value v in time δ, then $[\![S(x_1, \ldots x_n)]\!] = max(\delta_1, \ldots, \delta_n) \oplus \{\!|\, \langle v : \delta \rangle \,|\!\}$ as site S is not called until all its input parameters are defined.

For predefined sites, as they implement simple functions in the main machine, we consider that they have delay depreciable (0[1]) with respect to other sites. Let us provide the bags associated to predefined sites:

$$[\![1]\!] = \{\!|\, \langle 1 : 0 \rangle \,|\!\} \qquad [\![1(x_1, \ldots, x_n)]\!] = \begin{cases} \{\!|\, \langle x_n : 0 \rangle \,|\!\} & \text{if } \forall i \in [1, \ldots, n]; x_i \neq \bot \\ \{\!|\, |\!\} & \text{if otherwise} \end{cases}$$

$$[\![0]\!] = \{\!|\, |\!\} \qquad [\![Rtimer(t)]\!] = \{\!|\, \langle 1 : t \rangle \,|\!\} \qquad [\![If(b)]\!] = \begin{cases} \{\!|\, \langle 1 : 0 \rangle \,|\!\} & \text{if } b = \texttt{true} \\ \{\!|\, |\!\} & otherwise \end{cases}$$

For non-predefined sites we have considered two different behaviours depending on whether these sites are non-blocking sites or potentially blocking sites. For non-blocking sites we assume the *constant delay hypothesis*.

Definition 5.4 The *constant delay hypothesis* assume that every non-blocking site S has associated a delay δ_S. In any successful call S publishes after δ_S time units from the time at which all their parameters are defined.

. . . . . .

Then, a site call $S(v_1, \ldots v_n)$ with all variables define returns the value $s(v_1, \ldots v_n)$ in time δ_S:

$$[\![S(v_1, \ldots v_n)]\!] = \{\!|\, \langle s(v_1, \ldots, v_n) : \delta_S \rangle \,|\!\}$$

In the case of potentially blocking sites we assume the *deterministic delay hypothesis*.

Definition 5.5 The *deterministic delay hypothesis* assume that every potentially blocking site S has associated a delay function r. Any call $S(v_1, \ldots, v_n)$ publishes after $r(v_1, \ldots, v_n)$ time units from the time at which all their parameters are defined.

. . . . . .

[1]The use of delay 0 is only for simplification and we can use any value that we consider appropriate.

We assume that a site call $S(v_1, \ldots v_n)$ with all variables define returns $s(v_1, \ldots v_n)$ if $r(v_1, \ldots v_n) \neq \omega$, then:

$$[\![S(v_1, \ldots v_n)]\!] = \begin{cases} \{\!|\, \langle s(v_1, \ldots, v_n) : r(v_1, \ldots, v_n) \rangle \,|\!\} & \text{if } r(v_1, \ldots v_n) \neq \omega \\ \{\!|\, |\!\} & \text{otherwise} \end{cases}$$

In both cases, $s(x_1, \ldots, x_n)$ and $r(x_1, \ldots, x_n)$ assume them to be computable in poly-time with respect to its input values.

5.1.2 Operations

With the fully defined variables model, now we can express operators as operations between bags. For that, we need to take some things into account.

First, we need to keep in mind that variables have an associated delay until their definition, then these delays need to be decremented according to the time the expressions need to publish each value. Also, in pruning we are going to get only values for which their delay is minimum. Given an expression $E(x)$, we denote as $E(x = \langle v : \delta \rangle)$ to an execution of E where x takes the value v after δ time units. We also consider $E(x = \langle v : 0 \rangle)$, when x takes value before the execution of E.

Also, the operator ; was defined as to execute the right side if the left side does not publish anything. We have assumed that in order to ensure that a expression does not publish, we take a big enough time-out and consider that it does not publish if it does not publish within that time. Now, in order to express that time, we present the notation $A;_t B$ where t is the time for within the expression A can publish, in replacing to the original $A; B$.

Following we describe the behaviour of operators with the use of bags.

- *Sequential composition* $A > x > B(x)$ and $A \gg B$

$$[\![A > x > B(x)]\!] = \sqcap_{M \in [\![A]\!]} \sum_{\langle v:\delta \rangle \in M} \delta \oplus [\![B(x = \langle v : 0 \rangle)]\!]$$

$$[\![A \gg B]\!] = \sqcap_{M \in [\![A]\!]} \sum_{\langle v:\delta \rangle \in M} \delta \oplus [\![B]\!]$$

- *Parallel composition* $A \mid B$

$$[\![A \mid B]\!] = [\![A]\!] + [\![B]\!]$$

- *Pruning* $A(x) < x < B$

$$[\![A(x) < x < B]\!] = \sqcap_{M \in [\![B]\!]} \sqcap_{\langle v:\delta \rangle \in M; \delta = \mathsf{first}(M)} [\![A(x = \langle v : \delta \rangle)]\!]$$

In case of $M = \{\!|\, |\!\}$ we take $x = \bot$.

- *Otherwise* $A ;_t B$

$$[\![A ;_t B]\!] = \sqcap_{M \in [\![A]\!]} \begin{cases} M & \text{if } \mathsf{first}(M) \leq t \\ t \oplus [\![B]\!] & \text{otherwise} \end{cases}$$

- *Signaling* $A <! B$

$$[\![A(x) <! B]\!] = \sqcap_{M \in [\![B]\!]} \mathsf{first}(M) \oplus [\![A]\!]$$

Example 5.1 Let us see the orchestration *SendNews* (Examples 3.14.2).

$$SendNews = (AddressAlice > a > Email(a, m)) < m < (CNN \mid BBC)$$

Let us say that sites *CNN*,*BBC*, *AddressAlice* and *Email* have delay 500, 600, 200 and 1000 respectively.

The evaluation of *SendNews* will go as follows.

First a call to $(CNN \mid BBC)$ with an empty context give the possible values of m.

$$\begin{array}{rcl} [\![CNN \mid BBC]\!] & = & [\![CNN]\!] + [\![BBC]\!] \\ & = & \llparenthesis \langle \texttt{cnn} : 500 \rangle \rrparenthesis + \llparenthesis \langle \texttt{bbc} : 600 \rangle \rrparenthesis \\ & = & \llparenthesis \langle \texttt{cnn} : 500 \rangle, \langle \texttt{bbc} : 600 \rangle \rrparenthesis \end{array}$$

Then m take a value from $\llparenthesis \langle \texttt{cnn} : 500 \rangle, \langle \texttt{bbc} : 600 \rangle \rrparenthesis$. As `cnn` is the only value with minimum delay, $t = \llparenthesis \langle \texttt{cnn} : 500 \rangle \rrparenthesis$.

After that, $(AddressAlice > a > Email(a, \texttt{cnn}))$ is called.

AddressAlice publish `aalice` with delay 200. After publishing it, m is updated as $m = (\texttt{cnn} : 300)$.

Now it call to $Email(\texttt{aalice}, \texttt{cnn})$. This call publish `s_aalice_cnn` with delay 1000, but as it needs m to be defined, in total it takes 1300 time units to publish.

In the end, $[\![SendNews]\!] = \llparenthesis \langle \texttt{s\_aalice\_cnn} : 1500 \rangle \rrparenthesis$. Observe that, in this case, while there is a pruning operator with different values involved, as there is only one with minimum delay, then no demonic choice is involved.

From the behaviour of sites and operators, the following holds:

Theorem 5.1.1. *Given a finite orchestration E is holds that $[\![E]\!] = \llparenthesis \rrparenthesis$ or exists a unique non-deterministic finite decomposition in multi-sets $[\![E]\!] = \sqcap_{i=1}^{n} M_i$ being the elements of each M_i pairs of value and delay returned by an execution of E, for $1 \leq i \leq n$.*

5.2 Delay

In this chapter we are focusing in the delay of values published by an expression when it is executed. Now let us see the relation between the fully defined variables-delay model and the delay of outputs.

Definition 5.6 Given a bag $M = \llbracket \langle v_1 : \delta_1 \rangle, \ldots, \langle v_n : \delta_n \rangle \rrbracket$, the *delay bag* Δ_M denotes the multiset of delays $\llbracket \delta_1, \ldots, \delta_n \rrbracket$ containing the delays of each value of M.

If two bags M_1 and M_2 have the same delay bag $\Delta_{M_1} = \Delta_{M_2}$, we say that M_1 and M_2 are *time-equivalent*.

. . . . . .

As we have seen, for general Orc expressions, the different bags in $[\![E]\!]$ can have different number of items. Even so, we proved that, for the Orc sub-families introduced previously, expressions have the same number of published items in each execution. Now, we prove in this section that, for those sub-families, expressions are guaranteed to produce outputs with the same delays on items in all the executions. In order to prove that, first let us analyse the meaning of sub-expression for these families.

Theorem 5.2.1. *The meaning of $[\![E]\!]$ has a unique bag when restricting to expressions E in ExtIfOrc.*

Proof. As in the fully defined variables model, it can be proved simply by the definitions of sites as generators of multi-sets and operators as operations between them.

Sites are the basic sub-expressions that we can found. By definition, sites with deterministic behaviour publish at most one value with a delay, both determined by the context of the site call and there is no demonic choice involved in that. Therefore, the meaning of a site call has a unique bag.

Now, let us see that no operator introduces non-determinism as they just keep it from their sub-expressions. By induction, as the basic sub-expressions, sites, does not add non-determinism, then the meaning of E does not has any demonic choice.

Given that we have that $[\![E]\!] = M$ for a unique bag M. □

The pruning operator is the unique operator adding demonic choices. Now let us analyse the sub-families that allow this operator.

Theorem 5.2.2. *The meaning of $[\![E]\!]$ is composed by time-equivalent bags with the same delay on items when restricting to expressions E in ExtPruningOrc.*

Proof. In order to prove that, first let us remember that non-blocking sites always publish something if all the input values are defined and, by the constant delay hypothesis, it takes always the same amount of time. Given that, as we are considering only the delay of items, we can consider all values as signals with an associated delay, only indicating that a variable is defined at certain moment. In that case, we write $x = \langle \bot : \omega \rangle$ if the variable x is undefined, and $x = \langle 1 : \delta \rangle$ if it is defined at time δ. Now, let us analyse the operator $A(x) < x < B$ under that assumption:

In the base case, B is an ExtElementaryOrc expression. Given that, we have that $[\![B]\!]$ has a unique bag and therefore:

$$[\![A(x) < x < B]\!] = \begin{cases} [\![A(x = \langle 1 : \mathsf{first}([\![B]\!]) \rangle)]\!] & \text{if } [\![B]\!] \neq \llbracket\ \rrbracket \\ [\![A(x = \langle \bot : \omega \rangle)]\!] & \text{otherwise} \end{cases}$$

Then, in the case that B is an ExtPruningOrc expression we have that as the abstraction from values removes the non-determinism of pruning for the base case, by induction in the

general case there is also no non-determinism added with this operator, with respect the number of items.

Then we have that, while pruning introduces non-determinism on the values, if E is a ExtPruningOrc expression with meaning $[\![E]\!] = \sqcap_{i=1}^{n} M_i$, all bags $M_1 \dots M_n$ are time-equivalent. □

Theorem 5.2.3. *The meaning of $[\![E]\!]$ has a unique bag when restricting to expressions E in ExtMixedOrc.*

Proof. To prove this, let us analyse the behaviour of the operator $A(x) < x < B$ in ExtMixedOrc.

In the base case, B is an ExtIfOrc expression. Given that, we have that $[\![B]\!]$ has a unique bag. Also, the restriction imposed in ExtMixedOrc in order to allow the pruning operator ensures that either $[\![B]\!] = \{\!|\ |\!\}$ or $[\![B]\!] = \{\!|\ \langle v : \delta_1 \rangle, \dots, \langle v : \delta_n \rangle\ |\!\}$, for a value v determined by B and the context. Therefore, we have:

$$[\![A(x) < x < B]\!] = \begin{cases} [\![A(x = \langle v : \mathsf{first}([\![B]\!]) \rangle)]\!] & \text{if } [\![B]\!] = \{\!|\ \langle v : \delta_1 \rangle, \dots, \langle v : \delta_n \rangle\ |\!\} \\ [\![A(x = \langle \bot : \omega \rangle)]\!] & \text{otherwise} \end{cases}$$

Then, in the case that B is an ExtMixedOrc expression we have that as there is no non-determinism of pruning for the base case, by induction in the general case there is also no non-determinism added with this operator.

Given that we have that $[\![E]\!] = M$ for a unique bag M. □

Having meanings with a unique bag or all bags time-equivalents, now we can give a measure of the delay of outputs of an expression.

Definition 5.7 Given an expression E with $[\![E]\!] = \sqcap_{i=1}^{n} M_i$, where $M_1, \dots, M_n$ are time-equivalent, the *delay* of E is denoted as $\Delta(E) = \Delta_{M_1} = \dots = \Delta_{M_n}$.

. . . . . .

5.3 Related Problems

Having a well defined delays, there are some related computational problems of interest.

For a given expressions E in any of the studied sub-families given in chapter 3:

- **first**. Determine the minimum delay for a value published during an execution of E.
- **last**. Determine the maximum delay for a value published during an execution of E.
- **avgDelay**. Compute the average delay for the outputs published by an execution of E.

Observer that all these problem have a well defined solution as seen before. Now let us show some complexity bounds for the different problems and sub-families.

In Chapter 4 we had defined traces and sub-traces of a expression 4.5. For a finite Orc expression E with n operators $|$, $< x <$ and $;$, a *trace* $t \in \{0,1\}^{\leq n}$ was defined as an execution path on the evaluation of E. We noted $\mathsf{paths}(E) \subseteq \{0,1\}^{\leq n}$ the set of valid execution paths, and given $t \in \mathsf{paths}(E)$ we noted by $\mathsf{val}(t)$ the return value due to that path. Now, let us see the relation between delay and traces.

Definition 5.8 Given a trace $t \in \mathsf{paths}(E)$, we note as $\mathsf{time}(t)$ the time needed to publish following the path t. Observe that if $\mathsf{val}(t) = \lfloor\!\lfloor \rfloor\!\rfloor$ then $\mathsf{time}(t) = \omega$. Note that $\mathsf{time}(t)$ can be computed in poly-time with respect to the size of E.

. . . . . .

As in some cases we have to differentiate between the maximum delays and no publish, we introduce the following function between delays.

Definition 5.9 Given two delays δ_1 and δ_2 we define the function "max time", max^* as follows:

$$max^*(\delta_1, \delta_2) = \begin{cases} \delta_1 & \text{if } \delta_2 \leq \delta_1 \neq \omega \\ \delta_2 & \text{if } \delta_1 \leq \delta_2 \neq \omega \\ \delta_2 & \text{if } \delta_2 \leq \delta_1 = \omega \\ \delta_1 & \text{if } \delta_1 \leq \delta_2 = \omega \\ \omega & \text{if } \delta_1 = \delta_2 = \omega \end{cases}$$

. . . . . .

5.3.1 The complexity of problems first

Let us see how to compute first for ElementaryOrc and ExtElementaryOrc. The following algorithm to compute $\mathsf{first}(E)$ comes directly from the evaluation model delay based on bags and the fact that in ElementaryOrc and ExtElementaryOrc there is no distinction in values in order determine the delay of sites.

Lemma 5.3.1. *Let E be an ExtElementaryOrc expression. For a given context L, the first of E can be computed recursively as follows:*

$$
\begin{array}{rcl}
\mathit{first}(0) & = & \omega \\
\mathit{first}(1(x)) & = & \begin{cases} 0 & \mathit{if}\ (x \neq \bot) \\ \omega & \mathit{otherwise} \end{cases} \\
\mathit{first}(S(X)) & = & \begin{cases} \delta_S & \mathit{if}\ (x_i \neq \bot; \forall x_i \in X) \\ \omega & \mathit{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{first}(A|B) & = & min(\mathit{first}(A), \mathit{first}(B)) \\
\mathit{first}(A \gg B) & = & \mathit{first}(A) + \mathit{first}(B) \\
\mathit{first}(A > x > B) & = & \mathit{first}(A) + \mathit{first}(B(x \neq \bot))
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{first}(A ;_t B) & = & \begin{cases} \mathit{first}(A) & \mathit{if}\ (\mathit{first}(A) \leq t) \\ t + \mathit{first}(B) & \mathit{otherwise} \end{cases} \\
\mathit{first}(A <! B) & = & \begin{cases} \mathit{first}(A) + \mathit{first}(B) & \mathit{if}\ (\mathit{out}(B) > 0) \\ \omega & \mathit{otherwise} \end{cases}
\end{array}
$$

Corollary 5.3.2. *This recursive algorithm can be extended to* ExtIfOrc *simply taking values into account, in a similar way as was done for compute* $\mathit{out}(E^v)$.

Example 5.2 Let us consider the ElementaryOrc expression *SendNews* :

$$SendPrint \quad = \quad (CNN \mid BBC) > m > (EmailDad(m) \mid Print(m))$$

The expression can be rewritten as:

$$
\begin{array}{rcl}
A & = & (CNN \mid BBC) \\
B(m) & = & (EmailDad(m) \mid Print(m)) \\
SendPrint & = & A > m > B(m)
\end{array}
$$

Let us say that sites *CNN*,*BBC*, *EmailDad* and *Print* have delay 500, 600, 1000 and 1500 respectively.
We compute $\mathsf{first}(SendNews)$ as follows:

$$
\begin{array}{rcl}
\mathsf{first}(SendPrint) & = & \mathsf{first}(A) + \mathsf{first}(B(m \neq \bot)) \\
\mathsf{first}(A) & = & min(\mathsf{first}(CNN), \mathsf{first}(BBC)) min(500, 600) = 500 \\
\mathsf{first}(B(m \neq \bot)) & = & min(\mathsf{first}(EmailDad(m \neq \bot)), \mathsf{first}(Print(m \neq \bot))) \\
 & = & min(1000, 1500) = 1000 \\
\mathsf{first}(SendPrint) & = & 500 + 1000 = 1500
\end{array}
$$

Now, let us give some complexity bounds for the problem. First, let us see that at least this problem can be solved in poly-space for all the sub-families studied.

Theorem 5.3.3. *The problem* first *belongs to* PSPACE *when restricted to* MixedOrc.

Proof. Given an expression E, we can solve $\mathsf{first}(E)$ as :

$$\mathsf{first}(E) = \min_{t \in \mathsf{paths}(E)} \mathsf{time}(t)$$

□

Theorem 5.3.4. *The problem first belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. As the operator $A <!B$ is an special alias for the composition $(1(x) \gg A) < x < B$, we can do the same process for compute first as for PruningOrc and MixedOrc families.

In the other hand, the operator $A;_k B$ adds the problem that while we can compute $\mathsf{time}(t)$ in poly-time for any trace t, we cannot compute if $\mathsf{val}(t)$ will be a published value. In case that t describes an execution path through A, we need only need to know if the part of t relative to A takes time $\leq k$, as we consider are seeking for the first value published. In the other hand, in case that t describes an execution path through B, we do not need to check anything, as it will take always more time that the fastest execution through A if there is one within time k. □

After having that the problem belongs to PSPACE for all the sub-families, let us search for lower bounds on the complexity. For ElementaryOrc and ExtElementaryOrc, as for the output related problems, we have that first can be solved in poly-time.

Theorem 5.3.5. *The problem first belongs to P when restricted to ExtElementaryOrc.*

Proof. Directly from the algorithm to compute first for these sub-families. With a quick look, we see that each operator ask for the first of its sub-expressions at most one time. As that is true for all operators and for sites we can solve it in poly-time in the number of variables, then we can solve first in poly-time. □

Given the relation between publish and the delay on publishing, when adding values we encounter the same NP hardness as for Produces.

Theorem 5.3.6. *The problem first is NP-Hard when restricted to IfOrc.*

Proof. Observe that given a finite Orc expression E, the problem $E \in \mathsf{Produces}$ can be solved as $\mathsf{first}(E) \neq \omega$. That means that if an evaluation publish something, then that is done in less than infinite time, otherwise we can wait forever for the first published value.

Given that reduction, we have that as Produces is NP-Complete for IfOrc, the problem first is at least NP-Hard. □

For PruningOrc and ExtPruningOrc the complexity lower-bound of first remains as an open question.

5.3.2 The complexity of problems last

Let us see how to compute last for ElementaryOrc and ExtElementaryOrc. The following algorithm to compute last comes directly from the evaluation model delay based on bags and the fact that in ElementaryOrc and ExtElementaryOrc there is no distinction in values in order determine the delay of sites.

Lemma 5.3.7. *Let E be an ExtElementaryOrc expression. For a given context L, last of E can be computed recursively as follows:*

$$
\begin{array}{rcl}
\mathit{last}(0) & = & \omega \\
\mathit{last}(1(x)) & = & \begin{cases} 0 & \mathit{if}\ (x \neq \bot) \\ \omega & \mathit{otherwise} \end{cases} \\
\mathit{last}(S(X)) & = & \begin{cases} \delta_S & \mathit{if}\ (x_i \neq \bot; \forall x_i \in X) \\ \omega & \mathit{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{last}(A|B) & = & max^*(\mathit{last}(A), \mathit{last}(B)) \\
\mathit{last}(A \gg B) & = & \mathit{last}(A) + \mathit{last}(B) \\
\mathit{last}(A > x > B) & = & \mathit{last}(A) + \mathit{last}(B(x \neq \bot))
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{last}(A;_t B) & = & \begin{cases} \mathit{last}(A) & \mathit{if}\ (\mathit{first}(A) \leq t) \\ t + \mathit{last}(B) & \mathit{otherwise} \end{cases} \\
\mathit{last}(A <! B) & = & \begin{cases} \mathit{last}(A) + \mathit{first}(B) & \mathit{if}\ (\mathit{out}(B) > 0) \\ \omega & \mathit{otherwise} \end{cases}
\end{array}
$$

Corollary 5.3.8. *This recursive algorithm can be extended to* ExtIfOrc *simply taking values into account, in a similar way as was done for compute* $\mathit{out}(E^v)$.

Example 5.3 Let us consider the ElementaryOrc expression *SendNews* :

$$\mathit{SendPrint} \quad = \quad (\mathit{CNN} \mid \mathit{BBC}) > m > (\mathit{EmailDad}(m) \mid \mathit{Print}(m))$$

The expression can be rewritten as:

$$
\begin{array}{rcl}
A & = & (\mathit{CNN} \mid \mathit{BBC}) \\
B(m) & = & (\mathit{EmailDad}(m) \mid \mathit{Print})(m) \\
\mathit{SendPrint} & = & A > m > B(m)
\end{array}
$$

Let us say that sites *CNN*,*BBC*, *EmailDad* and *Print* have delay 500, 600, 1000 and 1500 respectively.
We compute $\mathsf{last}(\mathit{SendNews})$ as follows:

$$
\begin{array}{rcl}
\mathsf{last}(\mathit{SendPrint}) & = & \mathsf{last}(A) + \mathsf{last}(B(m \neq \bot)) \\
\mathsf{last}(A) & = & max^*(\mathsf{last}(\mathit{CNN}), \mathsf{last}(\mathit{BBC})) = max^*(500, 600) = 600 \\
\mathsf{last}(B(m \neq \bot)) & = & max^*(\mathsf{last}(\mathit{EmailDad}(m \neq \bot)), \mathsf{last}(\mathit{Print}(m \neq \bot))) \\
 & = & max^*(1000, 1500) = 1500 \\
\mathsf{last}(\mathit{SendPrint}) & = & 600 + 1500 = 2100
\end{array}
$$

Now, let us give some complexity bounds for the problem. First, let us see that at least this problem can be solved in poly-space for all the sub-families studied.

Theorem 5.3.9. *The problem* last *belongs to* PSPACE *when restricted to* IfOrc.

Proof. Given an expression E, we can solve last as :

$$\mathsf{last}(E) = max^*_{t \in \mathsf{paths}(E)} \mathsf{time}(t)$$

□

Theorem 5.3.10. *The problem last belongs to PSPACE when restricted to MixedOrc.*

Proof. The pruning operator adds the complexity that it only takes the fastest value published. When proving out in ♯.NP for MixedOrc we found a similar problem. In that case we solve it using the sub-traces introduced in 4.5.

Given an expression E, we can solve last as :

$$\mathsf{last}(E) = max^*_{s \in \mathsf{subpaths}(E)} min_{t \in \mathsf{paths}(E);\, s \backslash t} \mathsf{time}(t)$$

□

Theorem 5.3.11. *The problem last belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. As the operator $A <!B$ is an special alias for the composition $(1(x) \gg A) < x < B$, we can do the same process for compute first as for PruningOrc and MixedOrc families.

In the other hand, the operator $A;_k B$ adds the problem that while we can compute $\mathsf{time}(t)$ in poly-time for any trace t, we cannot compute if $\mathsf{val}(t)$ will be a published value. In case that t describes an execution path through A, we need to ensure that $\mathsf{first}(A) \leq k$, as we consider are seeking for the last value published and it can go through A if there is at least one value published through A in needing lees than time t. In the other hand, in case that t describes an execution path through B, we need to ensure that $\mathsf{first}(A) > k$, as in the other hand does not matter if $\mathsf{time}(t)$ is the maximum time as $\mathsf{val}(t)$ will not be published. □

After having that the problem belongs to PSPACE for all the sub-families, let us search for lower bounds on the complexity. For ElementaryOrc and ExtElementaryOrc, as for the output related problems, we have that last can be solved in poly-time.

Theorem 5.3.12. *The problem last belongs to P when restricted to ExtElementaryOrc.*

Proof. Directly from the algorithm to compute last for these sub-families. With a quick look, we see that each operator ask for the last of its sub-expressions at most one time. As that is true for all operators and for sites we can solve it in poly-time in the number of variables, then we can solve last in poly-time. □

As for first, when adding values we encounter the same NP hardness as for Produces.

Theorem 5.3.13. *The problem last is NP-Hard when restricted to IfOrc.*

Proof. Observe that given a finite Orc expression E, the problem $E \in$ Produces can be solved as $\mathsf{last}(E) \neq \omega$. That means that if an evaluation publish something, then that is done in less than infinite time, otherwise we can wait forever for that to publish any value.

Given that reduction, we have that as Produces is NP-Complete for IfOrc, the problem last is at least NP-Hard. □

For PruningOrc and ExtPruningOrc the complexity lower-bound of last remains as an open question.

5.3.3 The complexity of problems avgDelay

Let us see how to compute avgDelay for ElementaryOrc and ExtElementaryOrc. The following algorithm to compute avgDelay comes directly from from modify the algorithms for first and last. Taking into account that for parallel composition each sub-expression return a different number of outputs, we simply need to weigh the contribution of each part, in this case if $\mathsf{out}(A) = \mathsf{out}(B) = 0$, then $\mathsf{avgDelay}(A|B) = \omega$. For sequential compositions, it holds that the average time is the sum of the average of each sub-expression, as all published value bye the left side trigger the publication of the same number of values for the right side. Finally, for ; and <! operators, as we have always at most one average and one fixed time, then the formulas comes direct from first and last.

Lemma 5.3.14. *Let E be an ElementaryOrc or ExtElementaryOrc expression. For a given context L, avgDelay of E can be computed recursively as follows:*

$$
\begin{array}{rcl}
\mathit{avgDelay}(0) & = & \omega \\
\mathit{avgDelay}(1(x)) & = & \begin{cases} 0 & \text{if } (x \neq \bot) \\ \omega & \text{otherwise} \end{cases} \\
\mathit{avgDelay}(S(X)) & = & \begin{cases} \delta_S & \text{if } (x_i \neq \bot; \forall x_i \in X) \\ \omega & \text{otherwise} \end{cases} \\
\\
\mathit{avgDelay}(A|B) & = & \dfrac{\mathit{avgDelay}(A) * \mathit{out}(A) + \mathit{avgDelay}(B) * \mathit{out}(B)}{\mathit{out}(A) + \mathit{out}(B)} \\
\mathit{avgDelay}(A \gg B) & = & \mathit{avgDelay}(A) + \mathit{avgDelay}(B) \\
\mathit{avgDelay}(A > x > B) & = & \mathit{avgDelay}(A) + \mathit{avgDelay}(B(x \neq \bot)) \\
\\
\mathit{avgDelay}(A ;_t B) & = & \begin{cases} \mathit{avgDelay}(A) & \text{if } (\mathit{first}(A) \leq t) \\ t + \mathit{avgDelay}(B) & \text{otherwise} \end{cases} \\
\mathit{avgDelay}(A <! B) & = & \begin{cases} \mathit{avgDelay}(A) + \mathit{first}(B) & \text{if } (\mathit{out}(B) > 0) \\ \omega & \text{otherwise} \end{cases}
\end{array}
$$

Corollary 5.3.15. *This recursive algorithm can be extended to ExtIfOrc simply taking values into account, in a similar way as was done for compute $\mathsf{out}(E^v)$.*

Example 5.4 Let us consider the ElementaryOrc expression *SendNews* :

$$\mathit{SendPrint} \quad = \quad (\mathit{CNN} \mid \mathit{BBC}) > m > (\mathit{EmailDad}(m) \mid \mathit{Print}(m))$$

The expression can be rewritten as:

$$
\begin{array}{rcl}
A & = & (\mathit{CNN} \mid \mathit{BBC}) \\
B(m) & = & (\mathit{EmailDad}(m) \mid \mathit{Print}(m)) \\
\mathit{SendPrint} & = & A > m > B(m)
\end{array}
$$

Let us say that sites *CNN*,*BBC*, *EmailDad* and *Print* have delay 500, 600, 1000 and 1500 respectively.

We compute $\mathsf{last}(SendNews)$ as follows:

$$\begin{array}{rcl}
\mathsf{avgDelay}(SendPrint) & = & \mathsf{avgDelay}(A) + \mathsf{avgDelay}(B(m \neq \bot)) \\
\mathsf{avgDelay}(A) & = & (\mathsf{avgDelay}(CNN) + \mathsf{avgDelay}(BBC))/2 = (500 + 600)/2 = 550 \\
\mathsf{avgDelay}(B(m \neq \bot)) & = & (\mathsf{avgDelay}(EmailDad(m \neq \bot)) + \mathsf{avgDelay}(Print)(m \neq \bot))/2 \\
 & = & (1000 + 1500)/2 = 1250 \\
\mathsf{avgDelay}(SendPrint) & = & 550 + 1250 = 1800
\end{array}$$

Now, let us give some complexity bounds for the problem. First, let us see that at least this problem can be solved in poly-space for all the sub-families studied.

Theorem 5.3.16. *The problem avgDelay belongs to PSPACE when restricted to IfOrc.*

Proof. Given an expression E. In case that $\mathsf{out}(E) = 0 \Rightarrow \mathsf{avgDelay}(E) = \omega$. Otherwise, we can solve $\mathsf{avgDelay}(E)$ as :

$$\mathsf{avgDelay}(E) = \frac{\sum_{t \in \mathsf{paths}(E); \mathsf{time}(t) \neq \omega} \mathsf{time}(t)}{\mathsf{out}(E)}$$

□

Theorem 5.3.17. *The problem avgDelay belongs to PSPACE when restricted to MixedOrc.*

Proof. The pruning operator adds the complexity that it only takes the fastest value published. When proving last belongs PSPACE for MixedOrc we found the similar problem. Let us take now a similar approach based on sub-traces.

Given an expression E, we can solve avgDelay as :

$$\mathsf{avgDelay}(E) = \frac{\sum_{s \in \mathsf{subpaths}(E)} min_{t \in \mathsf{paths}(E);\ s \backslash t} \mathsf{time}(t)}{\mathsf{out}(E)}$$

□

Theorem 5.3.18. *The problem avgDelay belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. As the operator $A <! B$ is an special alias for the composition $(1(x) \gg A) < x < B$, we can do the same process for compute first as for PruningOrc and MixedOrc families.

In the other hand, the operator $A ;_k B$ adds the problem that while we can compute $\mathsf{time}(t)$ in poly-time for any trace t, we cannot compute if $\mathsf{val}(t)$ will be a published value. In case that t describes an execution path through A, we need to ensure that $\mathsf{first}(A) \leq k$, as $\mathsf{val}(t)$ will be published only if there is at least one value published through A in needing lees than time t. In the other hand, in case that t describes an execution path through B, we need to ensure that $\mathsf{first}(A) > k$, as in the other hand $\mathsf{val}(t)$ will not be published. □

After having that the problem belongs to PSPACE for all the sub-families, let us search for lower bounds on the complexity. For ElementaryOrc and ExtElementaryOrc, as for the output related problems, we have that avgDelay can be solved in poly-time.

Theorem 5.3.19. *The problem avgDelay belongs to P when restricted to ExtElementaryOrc.*

Proof. Directly from the algorithm to compute avgDelay for these sub-families. With a quick look, we see that each operator ask for the avgDelay of its sub-expressions at most one time. For the operator | it also ask for out, but as we have seen previously, that can be done in poly-time for these two sub-families or Orc. Then, as for sites we can solve it in poly-time in the number of variables, then we can solve avgDelay in poly-time. □

When adding values, we encounter new complexity issues, as for example we can use time to weigh publishing an non-publishing paths with different delays and then use those delays to compute the number of publishing paths. Given that, we encounter ♯P and ♯.NP hardness on families that take values into account.

Theorem 5.3.20. *The problem avgDelay is ♯P-Hard when restricted to IfOrc.*

Proof. Let us consider the following reduction form 3-SAT

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$. Let us encode each one of the cause through calls to site *If*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

$$
\begin{aligned}
E_{C_i}(y_{i_a}, y_{i_b}, y_{i_c}) = \quad & \mathit{If}(y_{ia}) \mid \\
& \mathit{If}(\neg y_{ia}) \gg \Big(\\
& \quad \mathit{If}(y_{ib}) \mid \\
& \quad \mathit{If}(\neg y_{ib}) \gg (\\
& \quad\quad \mathit{If}(y_{ic}) \mid \\
& \quad\quad \mathit{If}(\neg y_{ic}) \gg 1(0)) \Big)
\end{aligned}
$$

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

$$
\begin{aligned}
& \mathit{Double} = (\mathit{True} \mid \mathit{False}) \\
& T_z(z_1, \ldots, z_m) = \mathit{If}(\neg z_1) \mid \\
& \qquad \mathit{If}(z_1) \gg \Big(\\
& \qquad\quad \mathit{If}(\neg z_2) \gg \mathit{Rtimer}(1) \mid \\
& \qquad\quad \mathit{If}(z_2) \gg (\\
& \qquad\qquad \ldots \\
& \qquad\qquad\quad \mathit{If}(z_m) \gg \mathit{Rtimer}(1) \mid \\
& \qquad\qquad\quad \mathit{If}(\neg z_m) \ldots) \Big) \\
& E(x_1, \ldots, x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) > z_1 > \ldots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c}) \\
& \qquad\qquad > z_m > T_z(z_1, \ldots, x_m) \\
& E_F = \mathit{Double} > x_1 > \cdots > \mathit{Double} > x_n > E(x_1, \ldots, x_n)
\end{aligned}
$$

Note that now for $(v_1, \ldots, v_n) \in \{0, 1\}^n$ we have that E makes a call to T with $(z_1, \ldots, z_m)$, where $z_i = 1$ if $(v_1, \ldots, v_n)$ satisfies C_i and $z_i = 0$ otherwise. For its part, given $(z_1, \ldots, z_m) \in \{0, 1\}^m$, $T_z(z_1, \ldots, z_m)$ produces always 1 output, with delay 1 if all $z_1, \ldots, z_m$ are 1, and 0 otherwise.

Given that, for each $(v_1, \ldots, v_n) \in \{0, 1\}^n$ we have that $F(v_1, \ldots v_n) = \mathsf{avgDelay}(E(v_1, \ldots, v_n))$. As $(v_1, \ldots, v_n)$ takes all possible values in $\{0, 1\}^n$ by construction of E_F, we have:

$$\sharp F = \sum_{(v_1,\ldots,v_n)\in\{0,1\}^n} F(v_1,\ldots,v_n) =$$
$$\sum_{(v_1,\ldots,v_n)\in\{0,1\}^n} \mathsf{avgDelay}(E(v_1,\ldots,v_n))/2^n = \mathsf{avgDelay}(E_F)/2^n$$

□

Theorem 5.3.21. *The problem avgDelay is ♯.NP-Hard when restricted to MixedOrc.*

Proof. In order to proof that it is also ♯.NP-Hard, let us a similar reduction from counting with oracle in 3-SAT as the used for out on MixedOrc.

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with $n+m$ literals $x_1, \ldots, x_n, x'_1, \ldots, x'_m$, 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 $1 \leq i \leq n$. Let us encode each one of the cause through calls to site *If*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

$$E_{C_i}(y_{i_a}, y_{i_b}, y_{i_c}) = (If(y_{ia}) \mid If(y_{ib}) \mid If(y_{ic}))$$

Taking $True = 1(1)$ and $False = 1(0)$ the orchestration E_F associated to F is:

$$\begin{array}{l} Double = (True \mid False) \\ E(x_1,\ldots,x_n) = E_{C_1}(x_{1_a}, x_{1_b}, x_{1_c}) \gg \ldots \gg E_{C_m}(x_{m_a}, x_{m_b}, x_{m_c}) \\ E' = 1(w) < w < (Rtimer(1) \mid \\ \qquad\qquad (Double > x'_1 > \cdots > Double > x'_m > \\ \qquad\qquad\quad E(x_1,\ldots,x_n,x'_1,\ldots,x'_m))) \\ E_F = Double > x_1 > \cdots > Double > x_n > E'(x_1,\ldots,x_n) \end{array}$$

Note that now for $(v_1,\ldots,v_n,v'_1,\ldots,v'_m) \in \{0,1\}^{n+m}$ we have $F(v_1,\ldots v_n,v'_1,\ldots,v'_m) = \mathsf{out}(E(v_1,\ldots,v_n,v'_1,\ldots,v'_m))$. As we have that $(v'_1,\ldots,v'_m)$ takes all possible values in $\{0,1\}^m$ inside a pruning expression, we have that for each $(v_1,\ldots,v_n) \in \{0,1\}^n$:

$$\mathsf{avgDelay}(E(v_1,\ldots,v_n)) = 0 \Leftrightarrow$$
$$\exists (v'_1,\ldots,v'_m) \in \{0,1\}^m \text{ s.t. } (\mathsf{out}(E(v_1,\ldots,v_n,v'_1,\ldots,v'_m)) = 1) \Leftrightarrow \exists (v'_1,\ldots,v'_m) \in \{0,1\}^m$$
$$\text{s.t. } (F(v_1,\ldots,v_n,v'_1,\ldots,v'_m) = 1) \Leftrightarrow$$
$$\text{Fixing } (x_1,\ldots,x_n) = (v_1,\ldots,v_n),\ F \in 3-\mathsf{SAT}$$

As $(v_1,\ldots,v_n)$ takes all possible values in $\{0,1\}^n$ by construction of E_F, we have that $(1 - \mathsf{avgDelay}(E_F)) * 2^n$ is the result of counting with oracle in 3-SAT.

□

For PruningOrc and ExtPruningOrc the complexity lower-bound of avgDelay remains as an open question.

	first	last	avgDelay
ElementaryOrc	P	P	P
ExtElementaryOrc	P	P	P
PruningOrc	—	—	—
ExtPruningOrc	—	—	—
IfOrc	NP-H	NP-H	♯P-C
ExtIfOrc	NP-H	NP-H	♯P-H
MixedOrc	NP-H	NP-H	♯.NP-C
ExtMixedOrc	NP-H	NP-H	♯.NP-H

Table 5.1: Resume of problems related to the delay of outputs. All problems belong to PSPACE.

Chapter 6

Probabilistic Frameworks

In Chapter 2 we had seen that deterministic sites like the one studied until now are not very realistic. Even so, we have seen that sometimes sites can be described in terms like "is on-line at least 99% of the time" or "publish a random bit". In these cases, we are able to extract some probabilities about the behaviour of sites, and now we are going to use these probabilities to predict the behaviour of expressions using those sites. In this chapter we are going to extend the definition of sites adding the *probabilistic behaviour* in order to give an estimation of the internal non-determinism, performance and execution environment of services.

6.1 Probabilistic Sites

Without taking failures into account, the goal main goal of a service can make it non-deterministic. Sometimes the goal of the service is generate random data, perform some random or probabilistic computation or they can simply map real life facts. In this case we can have both a probability for return and a probability for the published value.

For these sites, sometimes we are able to extract some kind of probability about the behaviour of site calls either by its own internal behaviour or from logs of previous calls. We call these sites *probabilistic sites*.

Definition 6.1 For a disjoint group of behaviours, a *behaviour distribution* gives the probability for each behaviour. Taking a notation inspired in [35, page 20], a behaviour distribution is denoted as a tuple $(B_1@p_1 \parallel \ldots \parallel B_n@p_n)$, where p_i is the probability of having the behaviour B_i, for $1 \leq i \leq n$.

Given a way to express probabilistic behaviours, now we can give a more complete definition of probabilistic sites.

Definition 6.2 A *probabilistic site* is such that, without taking errors and delays into account, a site call publish different values (or halt) following a behaviour distribution. For a site call $S(X)$, its behaviour distribution is denoted as $P_S(X) = (v_1@p_1(X) \parallel \ldots \parallel v_n@p_n(X) \parallel \perp@q(X))$ where $\{v_1, \ldots, v_n\} = \mathbb{D}(S)$ and $p_i(X)$ denotes the probability of publish v_i given input X for $1 \leq i \leq n$, $q(X)$ denotes the probability of halting on input X.

The probabilistic behaviour can come in multiple ways. A site S can have a random execution independent on the input, in that case P_S is a unique distribution for each site call.

Example 6.1 Let us consider two different sites mimicking random choices. A call to *Coin* publishes $true$ with probability 1/2 otherwise publish $false$. A call to *BlockingCoin* publishes a signal with probability 1/2 otherwise halts. Then we have $P_{Coin} = (true@1/2 \parallel false@1/2)$ and $P_{BlockingCoin} = (1@1/2 \parallel \bot@1/2)$.

On the other hand, can be also possible to have a certain behaviour depending on the input values.

Example 6.2 Let us define a site $DiceX(x)$ as a generator that, for an input $x \in [1, 6]$ generates a number between 1 and 6, then if the number is bigger that x halts, otherwise publish the value. For this site then we have:

$$\begin{aligned}
P_{DiceX}(1) &= (1@1/6 \parallel \bot@5/6) \\
P_{DiceX}(2) &= (1@1/6 \parallel 2@1/6 \parallel \bot@4/6) \\
P_{DiceX}(3) &= (1@1/6 \parallel 2@1/6 \parallel 3@1/6 \parallel \bot@3/6) \\
P_{DiceX}(4) &= (1@1/6 \parallel 2@1/6 \parallel 3@1/6 \parallel 4@1/6 \parallel \bot@2/6) \\
P_{DiceX}(5) &= (1@1/6 \parallel 2@1/6 \parallel 3@1/6 \parallel 4@1/6 \parallel 5@1/6 \parallel \bot@1/6) \\
P_{DiceX}(6) &= (1@1/6 \parallel 2@1/6 \parallel 3@1/6 \parallel 4@1/6 \parallel 5@1/6 \parallel 6@1/6)
\end{aligned}$$

Definition 6.3 We denote probabilistic sites as *input independent* in case where the behaviour distribution does not depend on the input values and *input dependent* in case where the behaviour distribution depends in some way on the values of the call.

Observe that deterministic sites are a special case of probabilistic sites where the behaviour distribution for a call only have one possible outcome.

Example 6.3 Let us consider some deterministic sites defined in the Table 3.2. *True*, *False*, $And(b_1, \ldots, b_n)$ and $Or(b_1, \ldots, b_n)$. We consider these sites as a probabilistic sites with the following behaviours:

$$\begin{aligned}
P_{True} &= (true@1) \\
P_{False} &= (false@1) \\
P_{And}(b_1, \ldots, b_n) &= \begin{cases} (true@1) & \text{if } (b_1 \wedge \ldots \wedge b_n) = true \\ (false@1) & \text{otherwise} \end{cases} \\
P_{Or}(b_1, \ldots, b_n) &= \begin{cases} (true@1) & \text{if } (b_1 \vee \ldots \vee b_n) = true \\ (false@1) & \text{otherwise} \end{cases}
\end{aligned}$$

6.2 Site Call Profiles

We have seen that problems like network congestion or server failures can make site calls take different execution time or even fail. Also, we have that probabilistic sites does have different behaviour, even if failures are not considered, and that behaviour can or cannot be affected by the input values.

Now, we are going to consider only behaviour related to independent site calls. Given a site S, we assume that the possible output values belong to $\mathbb{D}(S)$ and the possible delays to $\Delta(S)$, being $\mathbb{D}(S)$ and $\Delta(S)$ finite and well defined domains.

We can describe a site taking into account both the probabilities of delay/failure and probabilistic behaviours of sites with a unique profile. In that case, the probability of halting in a successful execution is added to the probability of having delay ω or fail.

In general, for a probabilistic site $S(X)$ we have that things like the execution time or the probability of failure may depend in the input values and also in the published value.

Example 6.4 Let $Video(cat)$ be a site that given a video category, $cat \in \{music, funny, trailer\}$, returns a random video of that category. Let us assume that we describe the videos as *short*, *normal* and *long* given the duration.

As each category has different videos, for each there are different probabilities for each duration. Also, is clear that a *short* video will take less time to be transmitted that a *long* video.

Definition 6.4 Given a site $S(X)$, a *delay&value profile* $DV_S(X)$ describes define the distribution of $\langle value : delay \rangle$ for the output for a successful call to S with input X and the probability of halting.

$$DV_S(X) = (\langle v_1 : \delta_1 \rangle \,@p_{1,1}(X) \parallel \ldots \parallel \langle v_n : \delta_m \rangle \,@p_{n,m}(X) \parallel \langle \perp : \omega \rangle \,@q(X))$$

For $\{v_1, \ldots, v_n\} = \mathbb{D}(S)$ and $\{\delta_1, \ldots, \delta_m\} = \Delta(S)$, $p_{i,j}(X)$ is the probability of publish the value v_i with delay δ_j in a site call with input X, for $1 \leq i \leq n$,$1 \leq j \leq m$.

A *halting&value profile* HV_S describes the distribution of values for the output for a successful call to S with input X and the probability of halting.

$$HV_S(X) = (v_1@p_1(X) \parallel \ldots \parallel v_n@p_n(X))$$

For $\{v_1, \ldots, v_n\} = \mathbb{D}(S)$, $p_i(X)$ is the probability of publish the value v_i is the probability of publish the value v_i in a site call with input X, for $1 \leq i \leq n$.

The abstraction from time on halting &value profiles gives the relation $p_i(X) = \sum_{j=1}^{m} p_{i,j}$ for $1 \leq i \leq n$. In both cases, $q(X)$ is the probability of halting with input X.

. . . . . .

We assume that without matter the encoding of a site S, we can compute $DV_S(X)$ and $HV_S(X)$ in polynomial time on the size of X.

With an abuse of language, for a site S we write directly:

$$S(X) = (\langle v_1 : \delta_1 \rangle \,@p_{1,1}(X) \parallel \ldots \parallel \langle v_n : \delta_m \rangle \,@p_{n,m}(X) \parallel \langle \perp : \omega \rangle \,@q(X))$$

or abstracting from time:

$$S(X) = (v_1@p_1(X) \parallel \ldots \parallel v_n@p_n(X))$$

There are cases where the behaviour of sites does not depend on the input values neither on the published value.

Example 6.5 Let *Noise* be a site that returns a random $1Kb$ string when called (like the offered by *Random.org* [21, 22]), for example for cryptographic purposes. As *Noise* does not have any parameter, it is clear that the behaviour does not depend on the input. Also, we may assume that the value published will not influence the delay for publish it.

Then, we have that the site *Noise* has a distribution of delay and probability of halting independent on the input and the published value.

Now, in a similar way as we had non-blocking sites in case of deterministic sites without failures 3.2, let us describe these special sites independent on values.

Definition 6.5 A site $S(X)$ is *non-dependent on values* if the distribution of delays for site call to S does not depend on X neither on the published value.

. . . . . .

If $S(X) = (\langle v_1 : \delta_1 \rangle @p_{1,1}(X) \parallel \ldots \parallel \langle v_n : \delta_m \rangle @p_{n,m}(X) \parallel \langle \perp : \omega \rangle @q(X))$, we have that for any pair X, X', $p_{i,j}(X) = p_{i,j}(X')$ for $1 \leq i \leq n$, $1 \leq j \leq m$ and also $p_{i,j}(X) = p_{i',j}(X)$ for $1 \leq i \leq i' \leq n$, $1 \leq j \leq m$.

Given that, we can define the behaviour of delay/halting and published values with simple profiles:iour with the use of different profiles.

Definition 6.6 Given a non-dependent on values site $S(X)$, a *delay profile* D_S describes time needed for a site call to $S(X)$ to publish.

$$D_S = (\delta_1@p_1 \parallel \ldots \parallel \delta_m@p_m \parallel \omega@q)$$

For $\{\delta_1, \ldots, \delta_m\} = \Delta(S)$, p_j is the probability of publish with delay δ_j in a site call, for $1 \leq j \leq m$.

A *halting profile* H_S gives the probability that a site call to S succeeds. $H_S = (1 - q)$

In both cases, q is the probability of halting.

. . . . . .

Definition 6.7 Given a non-dependent on values site $S(X)$, a *value profile* $V_S(X)$ describes distribution of output values call to $S(X)$, given that $S(X)$ publish a value.

$$V_S(X) = (v_1@p_1^v(X) \parallel \ldots \parallel v_n@p_n^v(X))$$

For $\{v_1, \ldots, v_n\} = \mathbb{D}(S)$, $p_i^v(X)$ is the probability of publish the value v_i in a site call to $S(X)$ given that the site call succeeds, for $1 \leq i \leq n$.

. . . . . .

With an abuse of language, for a site S we write directly:

$$S(X) = \langle (v_1@p_1^v(X) \parallel \ldots \parallel v_n@p_n^v(X)) : (\delta_1@p_1 \parallel \ldots \parallel \delta_m@p_m \parallel \omega@q) \rangle$$

or abstracting from time:

$$S(X) = (v_1@p_1(X) \parallel \ldots \parallel v_n@p_n(X) \parallel \bot@q)$$

In cases where the output value is not important, we can write it as:

$$S(X) = \langle 1 : (\delta_1@p_1 \parallel \ldots \parallel \delta_m@p_m \parallel \omega@q) \rangle$$

or abstracting from time:

$$S(X) = (1@(1-q) \parallel \bot@q)$$

6.3 Environment Profiles

We have seen that site calls can have a probabilistic behaviour where we can consider all site calls as independent. But, as we have seen also in Chapter 2, site calls are not completely independent as problems like an attacked server affect to all site calls. In a similar way, the behaviour of a different site calls can be the same during all the execution, for example if site calls depend on some stored data.

Given that, in order to get better QoS measures, we may want to analyse each case separately.

Definition 6.8 We define as *environment* all the set of data, hardware, networks... that may affect the execution of an orchestration.

An environment can have different *states*, describing status of the environment for which site calls follow an specific behaviour.

. . . . . .

Example 6.6 Let us consider a possible interpretation of the site *BBC* (Table 3.2). Let the service BBC_{spain} be a news service that returns the number of news published in the BBC page [20] in the last hour related to the topic "Spain".

If we take $\{0, 13, 3^+\} = \mathbb{D}(BBC_{spain})$, to describe no new news, between 1 and 3 and more than 3, depending on the number of stored news the behaviour of BBC_{spain} may be:

$$BBC_{spain} = \begin{cases} (0@(1-q) \parallel \bot@q) & \text{no new news} \\ (13@(1-q) \parallel \bot@q) & \text{between 1 and 3 news} \\ (3^+@(1-q) \parallel \bot@q) & \text{more than 3 news} \end{cases}$$

As in the last example. sometimes there is a part of the environment that only affect a single sites.

Definition 6.9 Given a site S in an orchestration E, we define the *site environment* of S as the environment that may affect the behaviour of S and only site S.

. . . . . .

We assume that, for a site environment, we can describe the behaviour of independent site calls with the site call profiles seen previously (delay&value, halting&value, delay, halting and value profiles).

Definition 6.10 Given a site S, the *state profile* $\mathcal{E}_S = (Q_1@p_1 \parallel \ldots \parallel Q_n@p_n)$ define the probability distribution for the states of environment related to the site S. The environment is in the state Q_i with probability p_i, for $1 \leq i \leq n$.

For each state Q_i, there are site call profile associated for site S.

. . . . . .

Example 6.7 Let us consider the site BBC_{spain} (Example 6.6). This site has 3 possible states, 0 news, 1 to 3 news or more that 3 news, let us call them Q_A, Q_B and Q_C. Given that site calls may take different delays or even fail, the following could be a possible stable profile for BBC_{spain} if we are going to take into account the call time.

$$\begin{array}{rcl} \mathcal{E}_{BBC_{spain}} & = & (Q_A@0.2 \parallel Q_B@0.4 \parallel Q_C@0.4) \\ Q_A & = & (\langle 0:500\rangle @0.5 \parallel \langle 0:700\rangle @0.499 \parallel \langle \perp:\omega\rangle @0.001) \\ Q_B & = & (\langle 13:500\rangle @0.5 \parallel \langle 13:800\rangle @0.499 \parallel \langle \perp:\omega\rangle @0.001) \\ Q_C & = & (\langle 3^+:500\rangle @0.5 \parallel \langle 3^+:1000\rangle @0.499 \parallel \langle \perp:\omega\rangle @0.001) \end{array}$$

With the status profile we are considered the case where each site has its own environment and states, but sometimes we can have sites with common environments. For example, two sites in the same server, both sites go off-line if the server is disconnected. Given that, we are going to consider also the case where sites can have common environments.

Definition 6.11 Given site $S_1, \ldots, S_n$ in an orchestration E, we define the *common environment* of $S_1, \ldots, S_n$ as the environment that may affect the behaviour of these sites and only these site.

. . . . . .

We assume that, for a common environment, we can describe the behaviour of independent site calls with the site call profiles seen previously (delay&value, halting&value, delay, halting and value profiles).

Definition 6.12 Given site $S_1, \ldots, S_n$, the *type profile* $\mathcal{E}_{S_1,\ldots,S_n} = (C_1@p_1 \parallel \ldots \parallel C_n@p_n)$ define the probability distribution for the states of the common environment related to the sites $S_1, \ldots, S_n$. The environment is in the state C_i with probability p_i, for $1 \leq i \leq n$.

For each state C_i, there is an state profile $Q_{i,j}$ associated for each site S_j.

. . . . . .

A site S only can only follow one type profile.

Example 6.8 Let us consider the site $Email(a, m)$ and $EmailDad(m)$ (Table 3.2). We can assume that both sites use the same mail server, and therefore if that server is off-line then both sites fail, otherwise both sites succeeds. Then, the following is a possible correlated profile for those sites:

$$\begin{aligned} \mathcal{E}_{Email,EmailDad} &= (C_{OK}@0.999 \parallel C_{FAIL}@0.001) \\ C_{OK} &= Q^{OK}_{Email}, Q^{OK}_{EmailDad} \\ C_{FAIL} &= Q^{FAIL}_{Email}, Q^{FAIL}_{EmailDad} \end{aligned}$$

Where Q^{OK}_{Email} and $Q^{OK}_{EmailDad}$ are the stable profiles for $Email(a, m)$ and $EmailDad(m)$ when the mail server is on-line and Q^{FAIL}_{Email} and $Q^{FAIL}_{EmailDad}$ are the stable profiles when it is off-line.

Example 6.9 Given 4 sites $Site_1$, $Site_2$, $Site_3$ and $Site_4$, with a common environment, the following can be a possible description of the environments and the behaviour of $Site_1$:

Type Profile for sites $Site_1$, $Site_2$, $Site_3$ and $Site_4$:

	$Common_1@p_1$	$Common_2@p_2$	$Common_3@p_3$
$Site_1$	$Stable_{11}$	$Stable_{12}$	$Stable_{13}$
$Site_2$	$Stable_{21}$	$Stable_{22}$	$Stable_{23}$
$Site_3$	$Stable_{31}$	$Stable_{32}$	$Stable_{33}$
$Site_4$	$Stable_{41}$	$Stable_{42}$	$Stable_{43}$

State profiles of site S_1 for the states of the common environment $Common_1$:

$Stable_{11}$	$Behaviour_{111}@p_{111} \parallel Behaviour_{112}@p_{112} \parallel Behaviour_{113}@p_{113}$
$Stable_{12}$	$Behaviour_{121}@p_{121} \parallel Behaviour_{122}@p_{122} \parallel Behaviour_{123}@p_{123}$
$Stable_{13}$	$Behaviour_{131}@p_{131} \parallel Behaviour_{132}@p_{132}$

Delay&value and halting&value profiles of site $Site_1$ for the state $Behaviour_{111}$ of the site environment $Stable_{11}$:

	$Behaviour_{111}$
$Delay\&value$	$\langle v_1 : \delta_1 \rangle @pr_{11} \parallel \langle v_1 : \delta_2 \rangle @pr_{12} \parallel \langle v_2 : \delta_1 \rangle @pr_{21} \parallel \langle \bot : \omega \rangle @q$
$Halting\&value$	$v_1@(pr_{11} + pr_{12}) \parallel v_2@pr_21 \parallel \bot@q$

6.4 Models

Definition 6.13 Given an orchestration E having calls to sites $S_1, \ldots, S_n$, a *probabilistic environment* $\mathcal{P}$ describe probabilistic profiles of the environments and sites involved in E.

. . . . . .

Given the different possible profiles, we introduce four different models:

- ***Deterministic Model***. All sites have deterministic behaviours.

  The environment only has one state and all site calls have the same behaviour in all the executions.

- ***Oblivious Model***. During an execution, all site calls are independent one from another.

  The environment only has one state. All sites follow site call profiles (delay&value, halting&value,... profiles).

- ***Stable Model***. During an execution, all site calls are independent from calls to other sites, but can maintain certain degree of dependency with other site calls to the same site given the state site environment. A state for the site environment of each site is selected before the execution.

  Sites can have a site environment with more than one state. There are no common environment between sites.

- ***Typed Model***. During an execution, all site calls are independent from calls to other sites out of a common environment, but can maintain certain degree of dependency with other site calls given the states of the common and site environments.

  Sites can have a common and site environments with more than one state each. A state for the common environment and site environment of each site is selected before the execution.

Observe that the *Deterministic Model* is the model seen until now. A deterministic behaviour can be seen as a special case of site call profile where for any input values there is one possible behaviour with probability 1 and the others have probability 0.

Given that, the relation of these models is the following:

$$Deterministic \rightarrow Oblivious \rightarrow Stable \rightarrow Typed$$

Example 6.10 Let us consider the following expression:

$$\begin{aligned} WinWinWin &= BCoin > a > If(a) \gg BCoinBCoin \\ &\quad > b > If(b) \gg BCoin > c > If(c) \end{aligned}$$

This expression publish if the 3 calls to *BCoin* publish *true*. Let *BCoin* be a site that selects with probability 1/2 between a unbiased coin and a biased coin with probability 4/5 of publish true. Let us analyse this expression for both the oblivious and stable model.

In the Oblivious model *BCoin* choose between the two coins in each call, then we have probability $1/2(1/2 + 4/5) = 13/20$ of publish true in each call.

$$\begin{aligned} \mathcal{P} &= (P_{BCoin}) \\ P_{BCoin} &= (true@13/20 \parallel false@7/20) \end{aligned}$$

As we need 3 consecutive *true* outputs, *WinWinWin* will publish with probability $2197/8000 \approx 0.27$

For the Stable model, *BCoin* first choose between the two coins, then do all calls with the chosen coin.

$$\begin{aligned} \mathcal{P} &= (P_{BCoin}) \\ P_{BCoin} &= (Q_{unbiased}@1/2 \parallel Q_{biased}@1/2) \\ Q_{unbiased} &= (true@1/2 \parallel false@1/2) \\ Q_{biased} &= (true@4/5 \parallel false@1/5) \end{aligned}$$

If the unbiased coin is selected *WinWinWin* will publish with probability 1/8 and if it is the biased one, then it will publish with probability 64/125. As each case has the same probability, *WinWinWin* will publish with probability $1/2(1/8 + 64/125) = 637/2000 \approx 0.32$.

The relation between the different models give useful results:

Lemma 6.4.1. *For any problem P and a sub-family of Orc, an upper-bound on the complexity of P given a probabilistic environment $\mathcal{P}$ in Typed Model implies at most the same upper bound given a probabilistic environment $\mathcal{P}$ in Stable Model. That is also true for Stable and Oblivious models and Oblivious and Deterministic models.*

Lemma 6.4.2. *For any problem P and a sub-family of Orc, a lower-bound on the complexity of P given a probabilistic environment $\mathcal{P}$ in Deterministic Model implies at least the same lower bound given a probabilistic environment $\mathcal{P}$ inOblivious Model. That is also true for Oblivious and Stable models and Stable and Typed models.*

In the Stable and Typed models, the selection of the states of each site is done before the execution. As we define these profiles to be encoded in a way that each state is defined independently from another, then we can describe the selection of a state with a boolean string of length inferior to encoding of the profile.

Definition 6.14 Given probabilistic environment $\mathcal{P}$ for the Stable or Typed models, we define a *global state* as a selection of states for each environment. The *set of global states*, $\mathsf{G}(\mathcal{P})$, is the set of all global states.

Given $Q \in \mathsf{G}(\mathcal{P})$, $|Q| \leq |\mathcal{P}|$ and we can compute the probability of selecting that global state $\Pr(Q)$ in poly-time. Also, we denote as $(E|Q)$ the execution of an expression E following the global state Q.

. . . . . .

In chapter 3 we had defined 8 different non-recursive sub-families of Orc based on non-blocking/potentially blocking sites and the use of operator $< x <$ or ; and <!. Now, given a probabilistic environment, we redefine these sub-families replacing non-blocking sites for non-dependent on values and potentially blocking sites for general probabilistic sites with the posibility of failures.

Chapter 7

Number of outputs in probabilistic frameworks

In Chapter 4 we have seen some problems related to the number of outputs for the *Deterministic Model*. In that model, and the studied sub-families of Orc, the number of outputs when evaluate an expression is a well defined value, even if there are non-deterministic choices involved. Now, in the *Oblivious*, *Stable* and *Typed* models that is not true, and therefore we cannot study the same problems but a related ones.

Given a expression E and a probabilistic environment $\mathcal{P}$:

- **PrProduction** Compute the probability of publish at least one value during an execution of E ($\Pr(\mathsf{out}(E) > 0)$).
- **ExpOut**. Compute the expected number of outputs of an execution of E ($\mathbb{E}(\mathsf{out}(E))$).
- **PrOut**. Given $k \geq 0$, compute the probability of publish at least k values during an execution of E ($\Pr(\mathsf{out}(E) \geq k)$).

Table 7.1 at the end of the chapter resumes the results found for the different families and models.

7.1 The complexity of problems PrProduction

The problem PrProduction returns,for a given an Orc expression, the probability of publishing at least one value during an execution. Let us analyse it under the Oblivious, Stable and Typed models.

7.1.1 Oblivious model

In order to compute the probability of return, first we need to introduce additional tools.

Definition 7.1 Given a sub-orchestration A in an orchestration E with a probabilistic environment $\mathcal{P}$ in Oblivious Model. the post probability of A is the probability that for each value published by A, this value triggers the publication of at least one value in E. Finally $\Pr(A, \mathsf{post})$ denotes the probability to get at least one output on a given post.

Taking $\mathsf{post} = 1$ we have that $\Pr(E, \mathsf{post}) = \Pr(\mathsf{out}(E) > 0)$. From the definitions we get the following result.

Let us see how to compute PrProduction for ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc. The following algorithm to compute $\Pr(out(E) > 0)$ comes directly from the evaluation model based on bags and the fact that in these families values does not matter to the number of outputs.

Lemma 7.1.1. *Let E be an ExtPruningOrc expression. For a given context L, $\Pr(E, 1)$ can be computed recursively as follows.*

$$
\begin{aligned}
\Pr(0, \mathit{post}) &= 0 \\
\Pr(1(x), \mathit{post}) &= \begin{cases} \mathit{post} & \mathit{if}\ (x \in L) \\ 0 & \mathit{otherwise} \end{cases} \\
\Pr(S(X), \mathit{post}) &= \begin{cases} F_S(1) * \mathit{post} & \mathit{if}\ (\forall x_i \in X; x_i \in L) \\ 0 & \mathit{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\Pr(A \mid B, \mathit{post}) &= 1 - (1 - \Pr(A, \mathit{post})) * (1 - \Pr(B, \mathit{post})) \\
\Pr(A \gg B, \mathit{post}) &= \Pr(A, \Pr(B, \mathit{post})) \\
\Pr(A > x > B, \mathit{post}) &= \Pr(A, \Pr(B(x \neq \bot), \mathit{post})) \\
\Pr(A < x < B, \mathit{post}) &= \Pr(A(x \neq \bot, \mathit{post}) * \Pr(B, 1) \\
&\quad + \Pr(A(x = \bot), \mathit{post}) * (1 - \Pr(B, 1))
\end{aligned}
$$

$$
\begin{aligned}
\Pr(A; B, \mathit{post}) &= \Pr(A, \mathit{post}) + \Pr(B, \mathit{post}) * (1 - \Pr(A, \mathit{post})) \\
\Pr(A <\!! B, \mathit{post}) &= \Pr(A, \mathit{post}) * \Pr(B, 1)
\end{aligned}
$$

For ElementaryOrc and ExtElementaryOrc, as for the output related problems, we have that PrProduction(E) can be solved in poly-time.

Theorem 7.1.2. *In the Oblivious model, the problem PrProduction belongs to P when restricted to ExtElementaryOrc.*

Proof. Directly from the algorithm to compute $\Pr(E, \mathsf{post})$ for these sub-families. With a quick look, we see that each operator ask for the $\Pr(E, \mathsf{post})$ of its sub-expressions at most one time. As that is true for all operators and for sites we can solve it in poly-time in the number of variables, then we can solve $\Pr(\mathsf{out} > 0)$ in poly-time. □

For PruningOrc or ExtPruningOrc the algorithm does not give the same polynomial bound because the pruning operator involves check two different possibilities. Then, let us see what happen in these cases.

Theorem 7.1.3. *In the Oblivious model, the problem PrProduction belongs to PSPACE when restricted to ExtPruningOrc.*

Proof. Directly from the algorithm to compute $\Pr(E, \mathsf{post})$ for these sub-families. With a quick look, we see that this algorithm is a common recursive algorithm with polynomial

depth and a fixed amount of stored data for each step. Given that, we can solve $\Pr(\mathsf{out} > 0)$ in poly-space. □

Theorem 7.1.4. *In the Oblivious model, the problem PrProduction is ♯P-Hard when restricted to PruningOrc.*

Proof. Let us consider the following reduction from the counting MONOTONE 2-SAT. Let F be a MONOTONE 2-CNF formula as follows: $F = C_1 \wedge \cdots \wedge C_m$ 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 the site *BlockingCoin* with a halting profile

$$H_{BlockingCoin} = (success@1/2 \parallel failure@1/2)$$

Let us define the orchestration E_F associated to F as follows:

$$\begin{array}{l} \text{for } 1 \leq i \leq m,\ E_{C_i}(y_{i_a}, y_{i_b}) = \big(1(y_{i_a}) | 1(y_{i_b})\big) \\ E(x_1, \ldots, x_n) = 1(s) < s < (E_{C_1}(y_{1_a}, y_{1_b}) \gg \cdots \gg E_{C_m}(y_{m_a}, y_{m_b})) \\ E_F = (\cdots (E(x_1, \ldots, x_n) < x_1 < BlockingCoin) \cdots) < x_n < BlockingCoin \end{array}$$

During a evaluation of E_F, all variable $x_1, \ldots, x_n$ will be defined or not each with probability $1/2$ independently one of another. Then, we can describe the variables as a string $s = s_1, \ldots, s_n$, where $s_i = 1$ if $x_i \neq \bot$ and $s_i = 0$ if $x_i = \bot$, for $1 \leq i \leq n$. If we denote the execution of E_F following s as $E_F|s$, by construction of E_F we have that $F(s) = \mathsf{out}(E_F|s)$ Then, as each evaluation has probability $1/2^n$:

$$\Pr(\mathsf{out}(E_F) > 0) = \textstyle\sum_{s \in \{0,1\}^n} \mathsf{out}(E_F|s)/2^n = \sum_{s \in \{0,1\}^n} F(s)/2^n = \#F/2^n$$

Then, $2^n \Pr(\mathsf{out}(E_F) > 0)$ is the number of satisfying assignments for F. □

Now, let us analyse the case of IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc. Without loss of generality, let us focus on Boolean expressions. With a simple modification of $\Pr(A, \mathsf{post})$ used for ExtPruningOrc, now $\Pr(A, \mathsf{post}_0, \mathsf{post}_1)$ denotes the probability to get at least one output on a given post_0 when publishing 0 and post_1 when publishing 1.

Taking $\mathsf{post}_0 = \mathsf{post}_1 = 1$ we have that $\Pr(E, \mathsf{post}_0, \mathsf{post}_1) = \Pr(\mathsf{out}(E) > 0)$. From the definitions we get the following result.

Let us see how to compute PrProduction for IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc. The following algorithm to compute $\Pr(out(E) > 0)$ is a modification of the previous algorithm taking into account published values into a Boolean domain. Also, it is easy to see that we can extend it for any finite domain of values.

Lemma 7.1.5. *Let E be an ExtMixedOrc expression. For a given context L, $\Pr(E, 1, 1)$ can be computed recursively as follows.*

$$
\begin{array}{rcl}
\Pr(0, \mathit{post}_0, \mathit{post}_1) & = & 0 \\
\Pr(1(x), \mathit{post}_0, \mathit{post}_1) & = & \begin{cases} \mathit{post}_x & \mathit{if}\ (x \in L) \\ 0 & \mathit{otherwise} \end{cases} \\
\Pr(S(X), \mathit{post}_0, \mathit{post}_1) & = & \begin{cases} F_S(X,1)* & \\ \quad (V_S(X,0)\mathit{post}_0 & \\ \quad +V_S(X,1)\mathit{post}_1) & \mathit{if}\ (\forall x_i \in X; x_i \in L) \\ 0 & \mathit{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\Pr(A \mid B, \mathit{post}_0, \mathit{post}_1) & = & 1 - (1 - \Pr(A, \mathit{post}_0, \mathit{post}_1)) \\
 & & \quad *(1 - \Pr(B, \mathit{post}_0, \mathit{post}_1)) \\
\Pr(A \gg B, \mathit{post}_0, \mathit{post}_1) & = & \Pr(A, \Pr(B, \mathit{post}_0, \mathit{post}_1), \\
 & & \quad \Pr(B, \mathit{post}_0, \mathit{post}_1)) \\
\Pr(A > x > B, \mathit{post}_0, \mathit{post}_1) & = & \Pr(A, \Pr(B(x=0), \mathit{post}_0, \mathit{post}_1), \\
 & & \quad \Pr(B(x=1), \mathit{post}_0, \mathit{post}_1)) \\
\Pr(A < x < B, \mathit{post}_0, \mathit{post}_1) & = & \Pr(A(x = F_B(L), \mathit{post}_0, \mathit{post}_1) \\
 & & \quad * \Pr(B, 1, 1) \\
 & & + \Pr(A(x \neq F_B(L)), \mathit{post}_0, \mathit{post}_1) \\
 & & \quad *(1 - \Pr(B, 1, 1)) \\
 & & \\
\Pr(A; B) > 0) & = & \Pr(A, \mathit{post}_0, \mathit{post}_1) \\
 & & + \Pr(B, \mathit{post}_0, \mathit{post}_1) \\
 & & \quad *(1 - \Pr(A, \mathit{post}_0, \mathit{post}_1)) \\
\Pr(A <\!!B, \mathit{post}_0, \mathit{post}_1) & = & \Pr(A, \mathit{post}_0, \mathit{post}_1) \\
 & & \quad * \Pr(B, 1, 1)
\end{array}
$$

Now, let us give some complexity bounds for PrProduction on these families.

Theorem 7.1.6. *In the Oblivious model, the problem PrProduction belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Directly from the formula of how to compute $\Pr(E, \mathsf{post}_0, \mathsf{post}_1)$ for these sub-families. With a quick look, we see that this algorithm is a common recursive algorithm with polynomial depth and a fixed amount of stored data for each step. In case of bigger value domain, would need to repeat that for more values. Given that, we can solve $\Pr(\mathsf{out} > 0)$ in poly-space. □

Theorem 7.1.7. *In the Oblivious model, the problem PrProduction is ♯P-Hard when restricted to IfOrc.*

Proof. Let us consider the following reduction from the counting 3-SAT.

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$. Let us encode each one of the cause through calls to site *If*. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

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

Taking *Coin* as a site following a value profile $V_{Coin} = (true@1/2 \parallel false@1/2)$ the orchestration E_F associated to the formula F is:

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

Note that now for $(x_1, \ldots, x_n) \in \{0, 1\}^n$ we have $F(x_1, \ldots x_n) = 1 \Leftrightarrow \mathsf{out}(E(x_1, \ldots, x_n)) = 1$. As, by construction of E_F, $(x_1, \ldots, x_n)$ takes any value in $\{0, 1\}^n$ with the same probability $(1/2^n)$, we have:

$$\Pr(\mathsf{out}(E_F) > 0) \sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} \mathsf{out}(E(x_1, \ldots, x_n))/2^n = =$$
$$\sum_{(x_1,\ldots,x_n)\in\{0,1\}^n} F(x_1, \ldots, x_n)/2^n = \sharp F/2^n$$

Given that, we have that $2^n \Pr(\mathsf{out}(E_F) > 0)$ is the number of satisfying assignments for F. □

7.1.2 Stable and Typed models

For the Oblivious model we have that PrProduction can be computed in poly-space for all the sub-families studied, now let us see what happens in case of expressions in the Stable and Typed models.

Theorem 7.1.8. *In the Typed model, the problem PrProduction belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. We can compute $\Pr(\mathsf{out}(E) > 0)$ in the following way:

$$\Pr(\mathsf{out}(E) > 0) = \sum_{Q\in G(\mathcal{P})} \Pr(\mathsf{out}(E) > 0|Q) * \Pr(Q)$$

$E|Q$ is an expression following a Oblivious probabilistic environment, then we can compute $\Pr(\mathsf{out}(E|Q) > 0)$ in poly-space. Given that, we can simply compute $\Pr(\mathsf{out}(E) > 0)$ computing $\Pr(\mathsf{out}(E|Q) > 0)$ recursively for each $Q \in G(\mathcal{P})$. □

Theorem 7.1.9. *In the Stable model, the problem PrProduction is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Let us consider the following reduction from counting MONOTONE 2-SAT.

Given a MONOTONE 2-CNF 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 $1 \leq i \leq m$.

For $1 \leq i \leq n$, let X_i be a site following the Stable behaviour P_{X_i}:

$$\begin{aligned} P_{X_i} &= (Q_{itrue}@1/2 \parallel Q_{ifalse}@1/2) \\ Q_{itrue} &= (1@1) \\ Q_{ifalse} &= (\bot@1) \end{aligned}$$

Then for $1 \leq i \leq m$ define $E_{C_i} = (X_{i_a}|X_{i_b})$, where i_a and i_b are the indexes of the variables used in clause C_i. Then we define E_F associated to F as follows

$$E_F = E_{C_1} \gg \cdots \gg E_{C_m}$$

Given a global profile $Q = (q_1 \cdots q_n)$ by construction $\mathsf{out}((E_F)_{|\mathtt{Q}}) > 0) \Leftrightarrow F(\mathtt{Q}) = 1$.

As for any $Q \in \{0, 1\}^n$, $\Pr(Q) = 1/2^n$ it holds $\Pr(\mathsf{out}(E_F) > 0) = \sum_{Q\in\{0,1\}^n} \Pr(\mathsf{out}(E_F|Q) > 0) = (1/2^n) \sum_{\mathtt{Q}\in\{0,1\}^n} F(\mathtt{Q})$.

Then, $2^n \Pr(\mathsf{out}(E_F) > 0)$ is the number of satisfying assignments for F. □

7.2 The complexity of problems ExpOut

The problem ExpOut returns, given an Orc expression, the expected number of outputs of an execution. Let us analyse it under the Oblivious, Stable and Typed models.

7.2.1 Oblivious model

Let us see how to compute ExpOut for ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc. The following algorithm to compute $\mathbb{E}(out(E))$ is modification of the equivalents for out in the Deterministic model. In this case, we taken into account that sites may fail even if all variables are defined. Also, given that all calls are independent, for parallel and sequential compositions the expected output is the sum or multiplication of expected outputs as was the sum or multiplication of the number of outputs in deterministic cases. Finally, in cases where its needed that one sub-expression publish or not in order to evaluate in one way or another, as the two cases (publish or not) are disjoint, we only need to evaluate each case times the probability of each case (note that for the operator $A; B$, in $\mathbb{E}(\mathsf{out}(A))$ it is already included the probability that A publish.).

Lemma 7.2.1. *Let E be an ExtPruningOrc expression. For a given context L, $\mathbb{E}(\mathsf{out}(E))$ can be computed recursively as follows.*

$$
\begin{aligned}
\mathbb{E}(\mathsf{out}(0)) &= 0 \\
\mathbb{E}(\mathsf{out}(1(x))) &= \begin{cases} 1 & \text{if } (x \in L) \\ 0 & \text{otherwise} \end{cases} \\
\mathbb{E}(\mathsf{out}(S(X))) &= \begin{cases} F_S(1) & \text{if } (\forall x_i \in X; x_i \in L) \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}(\mathsf{out}(A \mid B)) &= \mathbb{E}(\mathsf{out}(A)) + \mathbb{E}(\mathsf{out}(B)) \\
\mathbb{E}(\mathsf{out}(A \gg B)) &= \mathbb{E}(\mathsf{out}(A)) * \mathbb{E}(\mathsf{out}(B)) \\
\mathbb{E}(\mathsf{out}(A > x > B)) &= \mathbb{E}(\mathsf{out}(A)) * \mathbb{E}(\mathsf{out}(B(x \neq \bot))) \\
\mathbb{E}(\mathsf{out}(A < x < B)) &= \mathbb{E}(\mathsf{out}(A(x \neq \bot))) * \Pr(\mathsf{out}(B) > 0) \\
&\quad + \mathbb{E}(\mathsf{out}(A(x = \bot))) * (1 - \Pr(\mathsf{out}(B) > 0))
\end{aligned}
$$

$$
\begin{aligned}
\mathbb{E}(\mathsf{out}(A; B)) &= \mathbb{E}(\mathsf{out}(A)) + \mathbb{E}(\mathsf{out}(B)) * (1 - \Pr(\mathsf{out}(A) > 0)) \\
\mathbb{E}(\mathsf{out}(A <! B)) &= \mathbb{E}(\mathsf{out}(A)) * \Pr(\mathsf{out}(B) > 0)
\end{aligned}
$$

For ElementaryOrc and ExtElementaryOrc, as for the output related problems, we have that ExpOut can be solved in poly-time.

Theorem 7.2.2. *In the Oblivious model, the problem ExpOut belongs to P when restricted to ExtElementaryOrc.*

Proof. Directly from the formula of how to compute $\mathbb{E}(\mathsf{out})$ for these sub-families. With a quick look, we see that each operator ask for the $\mathbb{E}(\mathsf{out})$ of its sub-expressions at most one time, and other calls as seen to be polynomial. As that is true for all operators and for sites we can solve it in poly-time in the number of variables, then we can solve $\mathbb{E}(\mathsf{out})$ in poly-time. □

For PruningOrc or ExtPruningOrc the algorithm does not give the same polynomial bound because the pruning operator involves check two different possibilities. Then, let us see what happen in these cases.

Theorem 7.2.3. *In the Oblivious model, the problem ExpOut belongs to PSPACE when restricted to ExtPruningOrc..*

Proof. Directly from the algorithm to compute $\mathbb{E}(\mathsf{out}(E))$ for these sub-families. With a quick look, we see that this algorithm is a common recursive algorithm with polynomial depth and a fixed amount of stored data for each step. Given that, we can solve $\mathbb{E}(\mathsf{out}(E))$ in poly-space. □

Theorem 7.2.4. *In the Oblivious model, the problem ExpOut is ♯P-Hard when restricted to PruningOrc.*

Proof. Let us consider the following reduction from the counting MONOTONE 2-SAT. Let F be a MONOTONE 2-CNF formula as follows: $F = C_1 \wedge \cdots \wedge C_m$ 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 the site *BlockingCoin* with a Halting profile

$$H_{BlockingCoin} = (success@1/2 \parallel failure@1/2)$$

Let us define the orchestration E_F as follows:

$$\begin{array}{l} \text{for } 1 \leq i \leq m,\ E_{C_i}(y_{i_a}, y_{i_b}) = \big(1(y_{i_a}) | 1(y_{i_b})\big) \\ E(x_1, \ldots, x_n) = 1(s) < s < (E_{C_1}(y_{1_a}, y_{1_b}) \gg \cdots \gg E_{C_m}(y_{m_a}, y_{m_b})) \\ E_F = (\cdots (E(x_1, \ldots, x_n) < x_1 < BlockingCoin) \cdots) < x_n < BlockingCoin \end{array}$$

During a evaluation of E_F, all variable $x_1, \ldots, x_n$ will have or not value each with probability $1/2$ independently one of another. Then, we can describe an execution as a string $s = s_1, \ldots, s_n$, where $s_i = 1$ if $x_i \neq \bot$ and $s_i = 0$ if $x_i = \bot$, for $1 \leq i \leq n$. If we denote the execution of E_F following s as $E_F|s$, by construction of E_F we have that $F(s) = \mathsf{out}(E_F|s)$ Then, as each evaluation has probability $1/2^n$:

$$\mathbb{E}(\mathsf{out}(E_F)) = \textstyle\sum_{s \in \{0,1\}^n} \mathsf{out}(E_F|s)/2^n = \sum_{s \in \{0,1\}^n} F(s)/2^n = \#F/2^n$$

Then, $2^n\, \mathbb{E}(\mathsf{out}(E_F))$ is the number of satisfying assignments for F. □

Now, let us analyse the case of IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc. Let us do a similar approach as for the Deterministic model where we computed $\mathsf{out}(E)$ as the sum $\mathsf{out}(E^v)$ for each possible value v. In this case, we are going to compute $\mathbb{E}(\mathsf{out}(E))$ as follows:

$$\mathbb{E}(\mathsf{out}(E)) = \sum_{v \in \mathbb{D}(E)} \mathbb{E}(\mathsf{out}(E^v))$$

The following algorithm to compute $\mathbb{E}(out(E^v))$ is a modification the algorithm for computing $\mathsf{out}(E^v)$ in the Deterministic model. In this case, as for the previous algorithm for solve $\mathbb{E}(\mathsf{out})$, we take into account the probability that a site fails, but now we add the distinction between the different values published. We use $\Pr(\mathsf{out}(A^v) > 0)$ describing the probability that

A publish at least one value v, that can be seen as an alias to $\Pr(\mathsf{out}(A > x > If(x = v) > 0)$ and therefore we know how to compute it. In the same way, we use $\mathbb{E}(\mathsf{out}(A))$ that can be computed as the sum of $\mathbb{E}(\mathsf{out}(A^v))$ for each possible value v as we have seen.

Lemma 7.2.5. *Let E be an IfOrc, ExtIfOrc, MixedOrc or ExtMixedOrc expression. For a given context L, $\mathbb{E}(\mathsf{out}(E^v))$ can be computed recursively as follows.*

$$
\begin{array}{rcl}
\mathbb{E}(\mathsf{out}(0^v)) & = & 0 \\
\mathbb{E}(\mathsf{out}(1(x)^v)) & = & \begin{cases} 1 & \text{if } (x \in L) \text{ and } (x = v) \\ 0 & \text{otherwise} \end{cases} \\
\mathbb{E}(\mathsf{out}(If(b)^v)) & = & \begin{cases} 1 & \text{if } (b = true) \text{ and } (v = 1) \\ 0 & \text{otherwise} \end{cases} \\
\mathbb{E}(\mathsf{out}(S(X)^v)) & = & \begin{cases} F_S(X,1) * V_S(X,v) & \text{if } (\forall x_i \in X; x_i \in L) \\ 0 & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathbb{E}(\mathsf{out}((A \mid B)^v)) & = & \mathbb{E}(\mathsf{out}(A^v)) + \mathbb{E}(\mathsf{out}(B^v)) \\
\mathbb{E}(\mathsf{out}((A \gg B)^v)) & = & \mathbb{E}(\mathsf{out}(A)) * \mathbb{E}(\mathsf{out}(B^v)) \\
\mathbb{E}(\mathsf{out}((A > x > B)^v)) & = & \sum_{w \in \mathbb{D}(A)} \mathbb{E}(\mathsf{out}(A^w)) * \mathbb{E}(\mathsf{out}(B^v(x = w))) \\
\mathbb{E}(\mathsf{out}((A < x < B)^v)) & = & \mathbb{E}(\mathsf{out}(A^v(x = F_B(L)))) * \Pr(\mathsf{out}(B) > 0) \\
 & & + \mathbb{E}(\mathsf{out}(A^v(x = \bot))) * (1 - \Pr(\mathsf{out}(B) > 0))
\end{array}
$$

$$
\begin{array}{rcl}
\mathbb{E}(\mathsf{out}((A; B)^v)) & = & \mathbb{E}(\mathsf{out}(A^v)) \\
 & & + \mathbb{E}(\mathsf{out}(B^v)) * (1 - \Pr(\mathsf{out}(A^v) > 0)) \\
\mathbb{E}(\mathsf{out}((A <! B)^v)) & = & \mathbb{E}(\mathsf{out}(A^v)) * \Pr(\mathsf{out}(B) > 0)
\end{array}
$$

Now, let us give some complexity bounds for ExpOut on these families.

Theorem 7.2.6. *In the Oblivious model, the problem ExpOut belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Directly from the formula of how to compute $\mathbb{E}(\mathsf{out}(E^v))$ for these sub-families. With a quick look, we see that this algorithm is a common recursive algorithm with polynomial depth and a fixed amount of stored data for each step. In case of bigger value domain, would need to repeat that for more values. Given that, we can solve $\mathbb{E}(\mathsf{out})$ in poly-space. □

Theorem 7.2.7. *In the Oblivious model, the problem ExpOut is ♯P-Hard when restricted to IfOrc.*

Proof. Direct from computing $\mathsf{out}(E)$ in IfOrc and ExtIfOrc. out in the Deterministic model is the equivalent problem as $\mathbb{E}(\mathsf{out})$ for the Oblivious model. Given that, as $\mathsf{out}(E)$ was proved to be ♯P-Complete, we have that $\mathbb{E}(\mathsf{out})$ is at least ♯P-Hard. □

Theorem 7.2.8. *In the Oblivious model, the problem ExpOut is ♯.NP-Hard when restricted to MixedOrc.*

Proof. Direct from computing $\mathsf{out}(E)$ in MixedOrc and ExtMixedOrc out in the Deterministic model is the equivalent problem as $\mathbb{E}(\mathsf{out})$ for the Oblivious model. Given that, as $\mathsf{out}(E)$ was proved to be ♯.NP-Complete, we have that $\mathbb{E}(\mathsf{out})$ is at least ♯.NP-Hard. □

7.2.2 Stable and Typed models

For the Oblivious model we have that ExpOut can be computed in poly-space for all the sub-families studied, now let us see what happens in case of expressions in the Stable and Typed models.

Theorem 7.2.9. *In the Typed model, the problem ExpOut belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. We can compute $\mathbb{E}(\mathsf{out}(E))$ in the following way:

$$\mathbb{E}(\mathsf{out}(E)) = \sum_{Q \in G(\mathcal{P})} \mathbb{E}(\mathsf{out}(E|Q)) * \Pr(Q)$$

$E|Q$ is an expression following a Oblivious probabilistic environment, then we can compute $\mathbb{E}(\mathsf{out}(E|Q))$ in poly-space. Given that, we can simply compute $\mathbb{E}(\mathsf{out}(E))$ computing $\mathbb{E}(\mathsf{out}(E|Q))$ recursively for each $Q \in G(\mathcal{P})$. □

Theorem 7.2.10. *In the Typed model, the problem ExpOut is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Let us consider the following reduction from counting 3-SAT.

Given 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$.

Let us see the encoding used for prove $\mathsf{out} \in$ ♯P-Complete for IfOrc.

Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

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

Now, for each E_{C_i} let us replace $If(x_j)$ for a call to X_j and $If(\neg x_j)$ for a call to $\overline{X_j}$. For example:

$$E_{C_i}(x_1, x_2, x_3) = (If(x_1) \mid (If(\neg x_1) \gg (If(\neg x_2) \mid (If(x_2) \gg If(x_3)))))$$

$$\Rightarrow E_{C_i} = (X_1 \mid (\overline{X_1} \gg (\overline{X_2} \mid (X_2 \gg X_3))))$$

Let us define each pair of sites $(X_i, \overline{X_i})$ as a pair of sites following a type profile:

$$\begin{array}{l} \mathcal{E}_{(X_i, \overline{X_i})} = (C_{true}@1/2 \parallel C_{false}@1/2) \\ C_{true} = ((success@1), (failure@1)) \\ C_{false} = ((failure@1), (success@1)) \end{array}$$

With probability $1/2$ all calls to site X_i succeeds and calls to $\overline{X_i}$ fail, and with probability $1/2$ otherwise. Then the orchestration E_F associated to the formula F is:

$$E_F = E_{C_1} \gg \ldots \gg E_{C_m}$$

We can describe a global profile for E_F as a string $Q = q_1, \ldots, q_n \in \{0, 1\}$, where $q_i = 1$ means X_i success and $\overline{X_i}$ fails and otherwise for $q_i = 0$ By construction of E_F we have that $F(Q) = \mathsf{out}(E_F|Q)$ Then, as each evaluation has probability $1/2^n$:

$$\mathbb{E}(\mathsf{out}(E_F)) = \textstyle\sum_{Q \in \{0,1\}^n} \mathsf{out}(E_F|Q)/2^n = \sum_{Q \in \{0,1\}^n} F(Q)/2^n = \#F/2^n$$

Then, $2^n \Pr(\mathsf{out}(E_F) > 0)$ is the number of satisfying assignments for F. □

For ElementaryOrc and ExtElementaryOrc In Stable model, the complexity lower-bound of ExpOut remains as an open question.

We have an interesting relation in cases where an evaluation can publish at most one value.

Lemma 7.2.11. *If an expression E publish only 0 or 1 value during any of its evaluations, then $\mathbb{E}(\mathit{out}(E)) = \Pr(\mathit{out}(E) > 0)$.*

That is because in these cases,

$$\mathbb{E}(\mathit{out}(E)) = \sum_{i=0}^{1} i * \Pr(\mathit{out}(E) = i) = 1 * \Pr(\mathit{out}(E) = 1) = \Pr(\mathit{out}(E) > 0)$$

This result can be extended for any sub-family and probabilistic environment $\mathcal{P}$ in Oblivious, Stable or Typed model.

7.3 The complexity of problems PrOut

The problem $\Pr(\mathsf{out} > k)$ is an extension of the problem $\Pr(\mathsf{out} > 0)$, where we ask not only that the expression publish some values, but that the number of published values is more k. Let us analyse it under the Oblivious, Stable and Typed models.

7.3.1 Oblivious model

In order to compute PrOut, first we need to introduce some tools.

Definition 7.2 Given a sub-orchestration A in an orchestration E with a probabilistic environment $\mathcal{P}$ in Oblivious Model. The postK distribution of A is the probability that for each value published by A, this value triggers the publication of $\mathsf{postK}[i]$ values in E, for $0 \leq i < k$. Finally $\Pr(A, \mathsf{post})$ denotes the distribution of probability to get i values on a given postK, $0 \leq i < k$.

. . . . . .

Taking $\mathsf{postK} = [0, 1, 0, \ldots, 0]$, if $P = \Pr(E, \mathsf{postK})$ we have that:

$$\Pr(\mathsf{out}(E) > 0) = 1 - \sum_{i=0}^{k-1} P[i]$$

Definition 7.3 Given two arrays A and B, we define $Prod(A, B)$ in the following way

$Prod(A, B) =$
  Array $V[0 \ldots k]$
  $fori : 0 \to k$
    $forj : 0 \to k - i$
      $V[i + j] = V[i + j] + A[i] * B[j]$
  $Return \quad V$

. . . . . .

Now, let us see how to compute PrOut for ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc. The following algorithm to compute $\Pr(E, \mathsf{postK})$ comes directly is a modification of the algorithm to compute $\Pr(out(E) > 0)$, using postK for the probability of ending publishing $0, 1, \ldots, k-1$ values. Then, we only need to modify the parallel composition to get the probabilities of having $0, 1, \ldots, k-1$ outputs, coming each output from one side or another. We use $value * Array$ and $Array + Array$ functions in the common way.

Lemma 7.3.1. *Let E be an ElementaryOrc, ExtElementaryOrc, PruningOrc or ExtPruningOrc expression. For a given context L, $\Pr(E, [0, 1, 0, \ldots, 0])$ can be computed recursively as follows:*

$$
\begin{array}{rcl}
\Pr(0, \mathit{postK}) & = & 0 \\
\Pr(1(x), \mathit{postK}) & = & \begin{cases} \mathit{postK} & \mathit{if}\ (x \in L) \\ 0 & \mathit{otherwise} \end{cases} \\
\Pr(S(X), \mathit{postK}) & = & \begin{cases} F_S(1) * \mathit{postK} & \mathit{if}\ (\forall x_i \in X; x_i \in L) \\ 0 & \mathit{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\Pr(A \mid B, \mathit{postK}) & = & \mathit{Prod}(\Pr(A, \mathit{postK}), \Pr(B, \mathit{postK})) \\
\Pr(A \gg B, \mathit{postK}) & = & \Pr(A, \Pr(B, \mathit{postK})) \\
\Pr(A > x > B, \mathit{postK}) & = & \Pr(A, \Pr(B(x \neq \bot), \mathit{postK})) \\
\Pr(A < x < B, \mathit{postK}) & = & \Pr(A(x \neq \bot, \mathit{postK}) * \Pr(\mathit{out}(B) > 0) \\
 & & + \Pr(A(x = \bot), \mathit{post}) * \Pr(\mathit{out}(B) = 0)
\end{array}
$$

$$
\begin{array}{rcl}
\Pr(A; B, \mathit{postK}) & = & \Pr(A, \mathit{postK}) - [\Pr(\mathit{out}(A) = 0), 0, \ldots, 0] \\
 & & + \Pr(B, \mathit{postK}) * \Pr(\mathit{out}(A) = 0) \\
\Pr(A <! B, \mathit{postK}) & = & \Pr(A, \mathit{postK}) * \Pr(\mathit{out}(B) > 0) \\
 & & + [\Pr(\mathit{out}(B) = 0), 0, \ldots, 0]
\end{array}
$$

Corollary 7.3.2. *This recursive algorithm can be extended to ExtMixedOrc simply taking values into account, in a similar way as was done for compute PrProduction(E).*

In this case, for ElementaryOrc and ExtElementaryOrc with a probabilistic environment $\mathcal{P}$ in Oblivious model, is open if the problem belongs to P or if it is NP-Hard or ♯P-Hard. In case of limiting k to a polynomial value on $|E|$, the presented algorithm can be used to compute $\Pr(\mathsf{out}(E) \geq k)$ in poly-time.

In order to give upper-bounds on the complexity, we need to introduce additional tools.

Definition 7.4 Given an expression E, in a similar way that the *post* probability was defined, we define $Pt(A)$ for a sub-expression A in E as the remainder part of E for evaluate after evaluate A. We call it *Post Expression.*

For example, given $E = A \gg ((B \gg C) \mid D) \gg F$, we have that $Pt(A) = ((B \gg C) \mid D) \gg F$, $Pt(B) = C \gg F$, $Pt(C) = F$, $Pt(D) = F$.

. . . . . .

Theorem 7.3.3. *In the Oblivious model, the problem PrOut belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Following the algorithm presented, is clear that post arrays may be exponentially big, and therefore are not a valid approximation to get poly-space. Now, let us see that we can compute $\Pr(\mathsf{out}(E) > k)$ as $\Pr(\mathsf{out}(E) > k) = 1 - \sum_{i=0}^{k} \Pr(\mathsf{out}(E) = i)$.

Following the algorithm, we can ask, not for the full post array but only for a unique value of this array. In that way, we can compute $\Pr(\mathsf{out}(E) = k)$ through recursive calls for different values on each sub-expression. For example, for $Pr(\mathsf{out}(A|B) = k)$ we can ask $Pr(\mathsf{out}(A \gg Pt(A)) = 0) * Pr(\mathsf{out}(B \gg Pt(B)) = k) + \ldots + Pr(\mathsf{out}(A \gg Pt(A)) = k) * Pr(\mathsf{out}(B \gg Pt(B)) = 0)$. A similar thing can be done for the operator ;. For $Pr(\mathsf{out}(A \gg B) = k)$, we can ask for $Pr(\mathsf{out}(A \gg Pt(A)(x \neq \bot) = k) * Pr(\mathsf{out}(B) > 0) + Pr(\mathsf{out}(A \gg Pt(A)(x = \bot)) = k) * Pr(\mathsf{out}(B) = 0)$ or with values in case of IfOrc and MixedOrc.

In general, what we can do is compute it recursively walking for all the execution tree. Given that, we claim that the problem PrOut belongs to PSPACE. □

Theorem 7.3.4. *In the Oblivious model, the problem PrOut is ♯P-Hard when restricted to PruningOrc and IfOrc.*

Proof. Direct from computing $\Pr(\mathsf{out} > 0)$. $\Pr(\mathsf{out} > 0)$ is a special case of $\Pr(\mathsf{out} > k)$ where $k = 0$, given that, as that problem is ♯P-Hard for these sub-families, $\Pr(\mathsf{out} > k)$ is also ♯P-Hard. □

For ElementaryOrc and ExtElementaryOrc In Oblivious model, the complexity lower-bound of PrOut remains as an open question.

7.3.2 Stable and Typed models

Now, let us consider the Stable and Typed models.

Theorem 7.3.5. *In the Typed model, the problem PrOut belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. We can compute $\Pr(\mathsf{out}(E) \geq k)$ in the following way:

$$\Pr(\mathsf{out}(E) \geq k) = \sum_{Q \in G(\mathcal{P})} \Pr(\mathsf{out}(E|Q) \geq k) * \Pr(Q)$$

$E|Q$ is an expression following a Oblivious probabilistic environment, then we can compute $\mathbb{E}(\mathsf{out}(E|Q))$ in poly-space. Given that, we can simply compute $\Pr(\mathsf{out}(E) \geq k)$ computing $\Pr(\mathsf{out}(E|Q) \geq k)$ recursively for each $Q \in G(\mathcal{P})$. □

Theorem 7.3.6. *In the Stable model, the problem PrOut is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Direct from computing $\Pr(\mathsf{out} > 0)$. $\Pr(\mathsf{out} > 0)$ is a special case of $\Pr(\mathsf{out} > k)$ where $k = 0$, given that, as that problem is ♯P-Hard, $\Pr(\mathsf{out} > k)$ is also ♯P-Hard. □

Problem	*Model*	**ElemOrc**	**EElemOrc**	**PruOrc**	**EPruOrc**	**IfOrc**	**EIfOrc**	**MixOrc**	**EMixOrc**
PrProduction	*Deterministic*	P	P	P	P	NP-C	NP-C	NP-C	NP-C
	Oblivious	P	P	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
ExpOut	*Deterministic*	P	P	P	P	♯P-C	♯P-H	♯.NP-C	♯.NP-H
	Oblivious	P	P	♯P-H	♯P-H	♯P-H	♯P-H	♯.NP-H	♯.NP-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯.NP-H	♯.NP-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯.NP-H	♯.NP-H
PrOut	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	———	———	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H

Table 7.1: Resume of problems related to the number of outputs with probabilistic environments. All problems belong to PSPACE.

Chapter 8

Probabilistic Delay

In Chapter 5 we have seen some problems related to the delay of outputs for the *Deterministic Model*. In that model, and the studied sub-families of Orc, the delay of outputs when evaluate an expression is a well defined bag, even if there are non-deterministic choices involved. Now, in the *Oblivious*, *Stable* and *Typed* models that is not true, and therefore we cannot study the same problems but a related ones.

Given a expression E and a probabilistic environment $\mathcal{P}$:

- **PrFirst**. Given $k \geq 0$, Compute the probability of publish the first value in at most time k during an execution of E ($\Pr(\mathsf{first}(E) \leq k)$).
- **ExpFirst**. Compute the expected delay for the first published value of an execution ($\mathbb{E}(\mathsf{first}(E))$). Only executions publishing at least one value are considered.
- **MinFirst**. Minimum first. Compute the minimum possible delay for a published value of an execution ($min(\mathsf{first}(E))$).
- **PrLast**. Given $k \geq 0$, Compute the probability of publish the last value after time k during an execution ($\Pr(\mathsf{last}(E) > k)$).
- **ExpLast**. Compute the expected delay for the last published value of an execution ($\mathbb{E}(\mathsf{last}(E))$). Only executions publishing at least one value are considered.
- **MaxLast**. Compute the maximum possible delay for a published value of an execution ($max(\mathsf{last}(E))$).

Tables 8.1 and 8.2 at the end of the chapter resumes the results found for the different families and models.

8.1 PrFirst & ExpFirst

The problem PrFirst returns, given an Orc expression and a value $k \geq 0$, the probability of publishing the first value within time k during an execution. The problem ExpFirst returns, given an Orc expression, the expected time of publishing the first value published during an execution, taking only into account executions publishing some values. Let us analyse them under the Oblivious, Stable and Typed models.

8.1.1 Oblivious model

In order give bounds to PrFirst and ExpFirst, first let us see that we can compute the probability that the first value is published at time k. This problem is defined as, given an expression E and a value $k \geq 0$, compute $\Pr(\mathsf{first}(E) = k)$.

Theorem 8.1.1. *In the Oblivious model, the problem* $\Pr(\mathit{first}(E) = k)$ *belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. We are going to use the *post expression* (Pt) defined previously 7.4. Then, following the evaluation of the expression we ask for the probabilities of having delay k. As some sub-families can have pruning operators, we are going to use the same context with time on variables as used in order to compute first for the Deterministic model, in that case, we write $\Pr(\mathsf{first}(A(x = \langle v : \delta \rangle))) = k)$ meaning that we ask for the probability that $\mathsf{first}(A) = k$ given that x will take the value v after time δ.

This time, we are not going to give the full algorithm but a brief description of the behaviour of each.

- $\Pr(\mathsf{first}((A|B)(x = \langle v : t \rangle))) = k)$.

  For A and B, we can ask $\Pr(\mathsf{first}(A(x = \langle v : t \rangle))) = k')$ and $\Pr(\mathsf{first}(B(x = \langle v : t \rangle))) = k')$ for any k'. Then, we can solve $\Pr(\mathsf{first}((A|B)(x = \langle v : t \rangle))) = k)$ with a recursive algorithm like the following:

$$\begin{aligned} prAlK &= \textstyle\sum_{i=0}^{k-1} \Pr(\mathsf{first}(A(x = \langle v : t \rangle))) = i) \\ prBlK &= \textstyle\sum_{i=0}^{k-1} \Pr(\mathsf{first}(B(x = \langle v : t \rangle))) = i) \\ prAK &= \Pr(\mathsf{first}(A(x = \langle v : t \rangle))) = k) \\ prBK &= \Pr(\mathsf{first}(B(x = \langle v : t \rangle))) = k) \\ Pr(\mathsf{first}((A|B)(x = \langle v : t \rangle))) = k) &= (1 - prAlK) * prBK \\ &\quad + prAK * (1 - prBlK - prBK) \end{aligned}$$

- $\Pr(\mathsf{first}((S \gg A)(x = \langle v : t \rangle))) = k)$.

  For a site following a sub-expression, we can ask for each possible delay δ of S inferior to k ($\delta \in D(S)$, $\delta \leq k$). Then, for each δ, we can ask for the probability that A publish in time $k - \delta$, $\Pr(\mathsf{first}(A(x = \langle v : t - \delta \rangle))) = k - \delta)$.

  In case that the call to S needs at least one variable undefined at that point, we can simply discount the time needed for that variable to take value. In the same way, if S defines a variable y with value v', then we will ask for $\Pr(\mathsf{first}(A(x = \langle v : t - \delta \rangle, y = \langle v' : 0 \rangle))) = k - \delta)$, for each possible value v'.

  In this case, A is always $Pt(S)$, as we consider all the following calls.

- $\Pr(\mathsf{first}((A < y < B)(x = \langle v : t \rangle))) = k)$.

  For each time $t' \in [0, k]$ we can ask for $\Pr(\mathsf{first}(B(x = \langle v : t \rangle))) = t')$. Then, we can ask to A for $\Pr(\mathsf{first}(A(x = \langle v : t \rangle, y = \langle v' : t' \rangle))) = k)$, with $v' = F_B(L)$ in case of MixedOrc and ExtMixedOrc or $v' \neq \bot$ for PruningOrc and ExtPruningOrc.

  Then, we can solve $\Pr(\mathsf{first}((A < y < B)(x = \langle v : t \rangle))) = k)$ with a recursive algorithm like the following:

$$\begin{array}{rcl} ST & = & \sum_{t'=0}^{k} \Pr(\mathsf{first}(B(x = \langle v : t\rangle)) = t') \\ \Pr(\mathsf{first}((A < y < B)(x = \langle v : t\rangle)) = k) & = & \sum_{t'=0}^{k} \Pr(\mathsf{first}(B(x = \langle v : t\rangle)) = t') \\ & & * \Pr(\mathsf{first}(A(x = \langle v : t\rangle, y = \langle v' : t'\rangle)) = k) \\ & & +(1 - ST) \\ & & * \Pr(\mathsf{first}(A(x = \langle v : t\rangle, y = \langle \bot : \omega\rangle)) = k) \end{array}$$

- $\Pr(\mathsf{first}((A <!B)(x = \langle v : t\rangle)) = k)$.

  In this case, we can ask to A and B for a pair of delays that make k in total Then, we can solve $\Pr(\mathsf{first}((A <!B)(x = \langle v : t\rangle)) = k)$ with a recursive algorithm like the following:

  $$\begin{array}{rcl} \Pr(\mathsf{first}((A <!B)(x = \langle v : t\rangle)) = k) & = & \sum_{i=0}^{k} \left(\Pr(\mathsf{first}(B(x = \langle v : t\rangle)) = 0) \right. \\ & & \left. * \Pr(\mathsf{first}(A(x = \langle v : t - i\rangle)) = (k - i)) \right) \end{array}$$

- $\Pr(\mathsf{first}((A;_q B)(x = \langle v : t\rangle)) = k)$.

  In this case, we need to consider possible cases into account.

  If $k < q$, then the value never will appear from B, simply because we assume that B will not be executed until time q. In that case, $\Pr(\mathsf{first}((A;_q B)(x = \langle v : t\rangle)) = k) = \Pr(\mathsf{first}(A(x = \langle v : t\rangle)) = k)$

  In case that $k > q$, then there is the possibility that $\Pr(\mathsf{first}(A(x = \langle v : t\rangle)) \leq q)$. We can solve it computing recursively the probability of $\Pr(\mathsf{first}(A(x = \langle v : t\rangle)) = i)$ for $0 \leq i \leq q$, then $\Pr(\mathsf{first}((A;_q B)(x = \langle v : t\rangle)) = k) = (1 - \Pr(\mathsf{first}(A(x = \langle v : t\rangle)) \leq q)) * \Pr(\mathsf{first}(B(x = \langle v : t - q\rangle)) = k - q)$.

  For $k = q$ we have the limit case. We have that $\Pr(\mathsf{first}(A(x = \langle v : t\rangle)) = k)$ return the probability that the first value is published by A in time k, the in a similar way as for $k > q$, we have that B will publish the first value if A takes times $> q$ for its first value and then B takes time 0 to publish its first value.

Given this outline of algorithm, we claim that $\Pr(\mathsf{first}(A) = k)$ belongs to PSPACE. □

Given this outline of algorithm, we claim that $\Pr(\mathsf{first}(A) = k)$ belongs to PSPACE. Then,

if we can compute $\Pr(\mathsf{first}(E) = k)$ in poly-space, we can compute also $\Pr(\mathsf{first}(E) \leq k)$ and $\mathbb{E}(\mathsf{first}(E))$.

Theorem 8.1.2. *In the Oblivious model, the problem PrFirst belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Given an expression E and a value k, we can compute $\Pr(\mathsf{first}(E) \leq k)$ with $k + 1$ recursive calls to $\Pr(\mathsf{first}(E) = i)$. Then, we can solve PrFirst at follows:

$$\Pr(\mathsf{first}(E) \leq k) = \sum_{i=0}^{k} \Pr(\mathsf{first}(E) = i)$$

As $\Pr(\mathsf{first}(E) = i)$ can be computed in poly-space, also $\Pr(\mathsf{first}(E) \leq k)$ can be done in poly-space. □

Theorem 8.1.3. *In the Oblivious model, the problem ExpFirst belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Given an expression E with n site calls, if the all sites publish in at most δ_{max} and otherwise does not publish, then any value published by E will take at most $n * \delta_{max}$ time. Given that, we can compute $\mathbb{E}(\mathsf{first}(E))$ with recursive calls to $\Pr(\mathsf{first}(E) = i)$ for $0 \leq i \leq n * \delta_{max}$. Then, we can solve ExpFirst at follows:

$$\mathbb{E}(\mathsf{first}(E)) = \sum_{i=0}^{n*\delta_{max}} i * \Pr(\mathsf{first}(E) = i) / \Pr(\mathsf{out}(E) > 0)$$

As both $\Pr(\mathsf{out}(E) > 0)$ and $\Pr(\mathsf{first}(E) = i)$ can be computed in poly-space, then $\mathbb{E}(\mathsf{first}(E))$ can be computed in in poly-space.

In case that $\Pr(\mathsf{out}(E) > 0) = 0$ we say that $\mathbb{E}(\mathsf{first}(E)) = \omega$. □

Now that we know that both PrFirst and ExpFirst can be computed in poly-space, let see some lower bound on their complexity for the different sub-families. First, let us see an interesting relation between PrFirst and PrProduction.

Lemma 8.1.4. *In the Oblivious Stable or Typed model, the problem PrProduction reduces to PrFirst when restricted to a finite Orc expression.*

Proof. We have seen that for any expression E with n site calls, if δ_{max} is the maximum time needed for any site when it publish, then any value published by an execution of E will be published within time $n * \delta_{max}$. Given that, is easy to see that:

$$\Pr(\mathsf{out}(E) > 0) = \Pr(\mathsf{first}(E) \leq n * \delta_{max})$$

□

Given that relation the we have that PrFirst is at least as hard as PrProduction.

Theorem 8.1.5. *In the Oblivious model, the problem PrFirst is ♯P-Hard when restricted to PruningOrc and IfOrc.*

Proof. Direct from Lemma 8.1.4, as PrProduction is ♯P-Hard for these sub-families of Orc, PrFirst is also ♯P-Hard. □

For $\mathsf{ExpFirst}(E)$ we cannot take the same approach from PrProduction as executions that not publish are not considered while computing ExpFirst, but we can find similar results.

Theorem 8.1.6. *In the Oblivious model, the problem ExpFirst is ♯P-Hard when restricted to PruningOrc.*

Proof. Let us consider the following reduction from counting MONOTONE 2-SAT.

Given a MONOTONE 2-CNF 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 *DelayCoin* be a site with delay profile $(0@1/2 \parallel 1@1/2)$. Let us define the orchestration E_F as follows:

for $1 \leq i \leq m$ do $E_{C_i}(y_{i_a}, y_{i_b}) = (1(y_{i_a})|1(y_{i_b}))$
$E(x_1, \ldots, x_n) = (E_{C_1}(y_{1_a}, y_{1_b}) \gg \cdots \gg E_{C_m}(y_{m_a}, y_{m_b}))$
$E_F = (\cdots(E(x_1, \ldots, x_n) < x_1 < \mathit{DelayCoin})\cdots) < x_n < \mathit{DelayCoin}$

Given $v = (v_1, \ldots v_n)$ such that v_i has delay $\in \{0, 1\}$ let us define map a boolean version $b(v_1, \ldots v_n) = (b(v_1), \ldots b(v_n))$ such that for $1 \leq i \leq n$

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

$$\mathsf{first}(E(v_1, \ldots, v_n)) = \begin{cases} 0 & \text{if } F(b(v_1), \ldots b(v_n)) = 1, \\ 1 & \text{otherwise} \end{cases}$$

As *DelayCoin* follows a delay profile $(0@1/2 \parallel 1@1/2)$, each $v \in \{0, 1\}^n$ occurs with probability $1/2^n$, then $\mathbb{E}(\mathsf{first}(E_F)) = (1/2^n) * \sum_{v \in \{0,1\}^n} \mathsf{first}(E(v)) = (1/2^n) * \sum_{v \in \{0,1\}^n} (1 - F(b(v))) = 1 - \sharp F/2^n$. Therefore, $2^n(1 - \mathbb{E}(\mathsf{first}(E_F)))$ solves counting MONOTONE 2-SAT. □

Theorem 8.1.7. *In the Oblivious model, the problem ExpFirst is ♯P-Hard when restricted to IfOrc.*

Proof. Let us consider the following reduction from counting 3-SAT.

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$. 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 us encode clause failures as follow. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

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

Taking *Coin* as a site following a value profile $V_{Coin} = (\mathit{true}@1/2 \parallel \mathit{false}@1/2)$, the orchestration E_F associated to the formula F is:

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

Given $v = (v_1, \ldots v_n)$ such that $v_i \in \{0, 1\}$ we have that:

$$\mathsf{first}(E(v)) = \begin{cases} 0 & \text{if } F(v_1, \ldots v_n) = 1, \\ 1 & \text{otherwise} \end{cases}$$

As each $v \in \{0, 1\}^n$ occurs with probability $1/2^n$, then:

$$\mathbb{E}(\mathsf{first}(E_F)) = (1/2^n) * \textstyle\sum_{v \in \{0,1\}^n} \mathsf{first}(E(v)) = \\ (1/2^n) * \textstyle\sum_{v \in \{0,1\}^n} (1 - F(v)) = 1 - \#F/2^n$$

Therefore, $2^n(1 - \mathbb{E}(\mathsf{first}(E_F)))$ solves counting 3-SAT. □

For ElementaryOrc and ExtElementaryOrc In Oblivious model, the complexity lower-bound of PrFirst and ExpFirst remains as open questions.

8.1.2 Stable and Typed models

For the Oblivious model we have that PrFirst and ExpFirst can be computed in poly-space for all the sub-families studied, now let us see what happens in case of expressions in the Stable and Typed models.

Theorem 8.1.8. *In the Typed model, the problem PrFirst belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. For the Stable and Typed models we can compute $\Pr(\mathsf{first}(E) \leq k)$ in the following way:

$$\Pr(\mathsf{first}(E) \leq k) = \sum_{Q \in G(\mathcal{P})} \Pr(\mathsf{first}(E|Q) \leq k) * \Pr(Q)$$

$E|Q$ is an expression following a Oblivious probabilistic environment, then we can compute $\Pr(\mathsf{out}(E|Q) > 0)$ in poly-space. Given that, we can simply compute $\Pr(\mathsf{out}(E) > 0)$ computing $\Pr(\mathsf{out}(E|Q) > 0)$ recursively for each $Q \in G(\mathcal{P})$. □

Theorem 8.1.9. *In the Typed model, the problem ExpFirst belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. For the Stable and Typed models we can compute $\mathbb{E}(\mathsf{first}(E))$ in the following way:

$$\mathbb{E}(\mathsf{first}(E)) = \sum_{Q \in G(\mathcal{P});\ \Pr(\mathsf{out}(E|Q)>0)} \mathbb{E}(\mathsf{first}(E|Q)) * \Pr(Q)$$

□

Theorem 8.1.10. *In the Stable model, the problem PrFirst is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Direct from Lemma 8.1.4, as PrProduction is ♯P-Hard for ElementaryOrc, PrFirst is also ♯P-Hard. □

Theorem 8.1.11. *In the Stable model, the problem ExpFirst is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Let us consider the following reduction from counting MONOTONE 2-SAT.

Given a MONOTONE 2-CNF 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 $1 \leq i \leq m$.

For $1 \leq i \leq n$, let X_i be a site following the status profile P_{X_i}:

$$\begin{array}{rcl} P_{X_i} & = & (P_{itrue}@1/2 \parallel P_{ifalse}@1/2) \\ P_{itrue} & = & (\langle 1:0 \rangle @1) \\ P_{ifalse} & = & (\langle \bot : \omega \rangle @1) \end{array}$$

Then for $1 \leq i \leq m$ define $E_{C_i} = (X_{i_a}|X_{i_b})$, where i_a and i_b are the indexes of the variables used in clause C_i. Then we define E_F as follows

$$E_F = (E_{C_1} \gg \cdots \gg E_{C_m}) \mid Rtimer(1)$$

Given a global profile $Q = (q_1 \cdots q_n)$ by construction $\mathbb{E}(\mathsf{first}(E_F|Q)) = 1 - F(s)$.

As for any $Q \in \{0,1\}^n$, $\Pr(Q) = 1/2^n$ it holds $\mathbb{E}(\mathsf{first}(E_F)) = \sum_{Q \in \{0,1\}^n} \mathbb{E}(\mathsf{first}(E_F|Q))/2^n = (1/2^n) \sum_{\mathsf{Q} \in \{0,1\}^n} (1 - F(Q)) = (1 - \#F)/2^n$.

Then, $2^n(1 - \mathbb{E}(\mathsf{first}(E_F)))$ is the number of satisfying assignments for F. □

8.2 MinFirst

The problem MinFirst returns, given an Orc expression, the minimum possible delay of a value published during an execution. Let us analyse it under the Oblivious, Stable and Typed models.

8.2.1 Oblivious model

Let us see an interesting relation between compute MinFirst in the Oblivious model and compute first in the Deterministic model.

Theorem 8.2.1. *In the Oblivious model, the problem MinFirst reduces to first in Deterministic model when restricted to ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc.*

Proof. For these sub-families, MinFirst can be seen as compute first given that all sites publish always and with its minimum delay. That is because, as the values of variables does not matter to the delay of sites, is clear that the minimum possible delay will correspond to an execution path when all sites take their minimum delay. □

Theorem 8.2.2. *In the Oblivious model, the problem MinFirst reduces to first in Deterministic model when restricted to IfOrc and MixedOrc.*

Proof. Given a site $S(x_1, \ldots, x_n)$, for each value $v_i \in \mathbb{D}(S)$ we define $V_{Si}(x_1, \ldots, x_n)$ as a site publishing v_i with the minimum delay for which a call to $S(w_1, \ldots, w_n)$ can publish v_i. Then, given an expression E, we define E' as a copy of E replacing each site S by a sub-expression $(V_{S1} \mid V_{S1} \mid \ldots)$.

Now, we can solve MinFirst of E as first of E', given that E' follow all the possible execution paths of E for all the possible values with their minimum delay. □

Observe that for ExtIfOrc and ExtMixedOrc we cannot take the same approach, as for $A;_t B$ it is possible that an execution through B trigger the publication of the value with minimum delay, and we can lose that value if there is a path publishing through A in lees than time t. We cannot replace either $A;_t B$ by $A|(Rtimer(t) \gg B)$ as, while in that case we will find if there is a fastest path through B, it is possible that there is always a path from A needing less that time t and therefore B never will be executed.

In Theorem 8.1.2 we have seen that $\Pr(\mathsf{first}(E) \leq t)$ can be computed in poly-space for all the sub-families seen. Now let us use that to give upper bounds to MinFirst.

Theorem 8.2.3. *In the Oblivious model, the problem MinFirst belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Given a expression E, we can compute MinFirst recursively with calls to $\Pr(\mathsf{first}(E) \leq t)$. For that, starting with $i = 0$ we check $\Pr(\mathsf{first}(E) \leq i) = 0$ and go incrementing i until $\Pr(\mathsf{first}(E) \leq i) > 0$. The first delay i for which $\Pr(\mathsf{first}(E) \leq i) > 0$ will be MinFirst of E. □

8.2.2 Stable and Typed models

For the Oblivious model we have that MinFirst can be computed in poly-space for all the sub-families studied, now let us see what happens in case of expressions in the Stable and Typed models.

Theorem 8.2.4. *In the Typed model, the problem MinFirst belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. For the Stable and Typed models we can compute MinFirst in the following way:

$$min(\mathsf{first}(E)) = min_{Q \in G(\mathcal{P})}\mathsf{MinFirst}(E|Q)$$

□

Theorem 8.2.5. *In the Typed model, the problem MinFirst is NP-Hard when restricted to ElementaryOrc.*

Proof. Let us remind the reduction from counting 3-SAT used to prove $\mathbb{E}(\mathsf{out})$ in ♯P-Hard also for the Typed model (Theorem 7.2.10).

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, we had transformations of clauses into sub-expressions using sites as variables, in the form:

$$C_i = (x_1 \vee x_2 \vee x_3) \Leftrightarrow E_{C_i} = (X_1 \mid (\overline{X_1} \gg (X_2 \mid (\overline{X_2} \gg X_3))))$$

Each pair of sites $(X_i, \overline{X_i})$ follow a status profile where only one site succeeds on its calls, now let us add time to this profile:

$$S_{(X_i,\overline{X_i})} = \quad \Big((\{\langle \bot : \omega \rangle @1\}, \{\langle 1 : 0 \rangle @1\})@1/2 \parallel (\{\langle 1 : 0 \rangle @1\}, \{\langle \bot : \omega \rangle @1\})@1/2\Big)$$

Then the orchestration E_F associated to F was:

$$E_F = E_{C_1} \gg \ldots \gg E_{C_m}$$

Describing a global profile for E_F as a string $Q = q_1, \ldots, q_n \in \{0,1\}$, where $q_i = 1$ means X_i success and $\overline{X_i}$ fails and otherwise for $q_i = 0$ By construction of E_F we have that $F(Q) = 1 \Leftrightarrow \mathsf{MinFirst}(E_F|Q) = 0$. Then, we have that:

$$\min(\mathsf{first}(E_F)) = 0 \Leftrightarrow \exists Q \in \{0,1\}^n \text{ s.t. } (\mathsf{out}(E_F|Q) = 1) \Leftrightarrow$$
$$\exists Q \in \{0,1\}^n \text{ s.t. } (F(Q) = 1) \Leftrightarrow F \in 3 - SAT$$

□

For ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc In Stable model, the complexity lower-bound of MinFirst remain as an open questions.

8.3 PrLast & ExpLast

The problem PrLast returns, given an Orc expression and a value $k \geq 0$, the probability of publishing the last value after time k during an execution. The problem ExpLast returns, given an Orc expression, the expected time of publishing the last value published during an execution, taking only into account executions publishing some values. Let us analyse them under the Oblivious, Stable and Typed models.

8.3.1 Oblivious model

In order give bounds to PrLast and ExpLast, first let us see that we can compute the probability that the last value is published at time k. This problem is defined as, given an expression E and a value $k \geq 0$, compute $\Pr(\mathsf{last}(E) = k)$.

Theorem 8.3.1. *In the Oblivious model, the problem* $\Pr(\mathit{last}(E) = k)$ *belongs to PSPACE when restricted to MixedOrc.*

Proof. We are going to use the *post expression* (Pt) defined previously 7.4. Then, following the evaluation of the expression we ask for the probabilities of having delay k. As some sub-families can have pruning operators, we are going to use the same context with time on variables as used in order to compute last for the Deterministic model, in that case, we write $\Pr(\mathsf{last}(A(x = \langle v : \delta \rangle)) = k)$ meaning that we ask for the probability that $\mathsf{last}(A) = k$ given that x will take the value v after time δ.

This time, we are not going to give the full algorithm but a brief description of the behaviour of each.

- $\Pr(\mathsf{last}((A|B)(x = \langle v : t \rangle)) = k)$.

  For A and B, we can ask $\Pr(\mathsf{last}(A(x = \langle v : t \rangle)) = k')$ and $\Pr(\mathsf{last}(B(x = \langle v : t \rangle)) = k')$ for any k'. Then, can solve $\Pr(\mathsf{last}((A|B)(x = \langle v : t \rangle)) = k)$ with a recursive algorithm like the following:

$$\begin{aligned}
prAlK &= \textstyle\sum_{i=0}^{k-1} \Pr(\mathsf{last}(A(x = \langle v : t \rangle)) = i) \\
prBlK &= \textstyle\sum_{i=0}^{k-1} \Pr(\mathsf{last}(B(x = \langle v : t \rangle)) = i) \\
prAK &= \Pr(\mathsf{last}(A(x = \langle v : t \rangle)) = k) \\
prBK &= \Pr(\mathsf{last}(B(x = \langle v : t \rangle)) = k) \\
prAO0 &= \Pr(\mathsf{last}(A(x = \langle v : t \rangle)) = \omega) \\
prBO0 &= \Pr(\mathsf{last}(B(x = \langle v : t \rangle)) = \omega) \\
Pr(\mathsf{last}((A|B)(x = \langle v : t \rangle)) = k) &= prAK * (prBlK + prBO0) \\
&\quad + prBK * (prAlK - prAK + prAO0)
\end{aligned}$$

- $\Pr(\mathsf{last}((S \gg A)(x = \langle v : t \rangle)) = k)$.

  For a site following a sub-expression, we can ask for each possible delay δ of S inferior to k ($\delta \in D(S)$, $\delta \leq k$). Then, for each δ, we can ask for the probability that A publish in time $k - \delta$, $\Pr(\mathsf{last}(A(x = \langle v : t - \delta \rangle)) = k - \delta)$.

  In case that the call to S needs at least one variable undefined at that point, we can simply discount the time needed for that variable to take value. In the same way, if S defines a variable y with value v', then we will ask for $\Pr(\mathsf{last}(A(x = \langle v : t - \delta \rangle, y = \langle v' : 0 \rangle)) = k - \delta)$, for each possible value v'.

  In this case, A is always $Pt(S)$, as we consider all the following calls.

- $\Pr(\mathsf{last}((A < y < B)(x = \langle v : t \rangle)) = k)$.

  For each time $t' \in [0, k]$ we can ask for $\Pr(\mathsf{first}(B(x = \langle v : t \rangle)) = t')$. Then, we can ask to A for $\Pr(\mathsf{last}(A(x = \langle v : t \rangle, y = \langle v' : t' \rangle)) = k)$, with $v' = F_B(L)$ in case of MixedOrc and ExtMixedOrc or $v' \neq \bot$ for PruningOrc and ExtPruningOrc.

If max_B is an upper-bound to the maximum possible delay of B, for example $max_B = n * \delta_{max}$ where n is the number of sites in B and δ_{max} the maximum delay of a site call for the sites of B, then we can solve $\Pr(\mathsf{last}((A < y < B)(x = \langle v : t\rangle)) = k)$ with the following recursive algorithm:

$$\begin{array}{rcl} ST & = & \sum_{t'=0}^{max_B} \Pr(\mathsf{first}(B(x = \langle v : t\rangle)) = t') \\ \Pr(\mathsf{last}((A < y < B)(x = \langle v : t\rangle)) = k) & = & \sum_{t'=0}^{max_B} \Pr(\mathsf{first}(B(x = \langle v : t\rangle)) = t') \\ & & * \Pr(\mathsf{last}(A(x = \langle v : t\rangle, y = \langle v' : t'\rangle)) = k) \\ & & +(1 - ST) \\ & & * \Pr(\mathsf{last}(A(x = \langle v : t\rangle, y = \langle \bot : \omega\rangle)) = k) \end{array}$$

Given this outline of algorithm, we claim that $\Pr(\mathsf{last}(A) = k)$ belongs to PSPACE. □

Theorem 8.3.2. *In the Oblivious model, the problem* $\Pr(\mathit{last}(E) = k)$ *belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. We can extend the algorithm to compute $\Pr(\mathsf{last}(E) = k)$ for MixedOrc expressions adding for the operators $<!$ and $;_t$:

- $\Pr(\mathsf{last}((A <!B)(x = \langle v : t\rangle)) = k)$.

  In this case, we can ask to A and B for a pair of delays that make k in total Then, we can solve $\Pr(\mathsf{last}((A <!B)(x = \langle v : t\rangle)) = k)$ with a recursive algorithm like the following:

  $$\begin{array}{rcl} \Pr(\mathsf{last}((A <!B)(x = \langle v : t\rangle)) = k) & = & \sum_{i=0}^{k} \left(\Pr(\mathsf{first}(B(x = \langle v : t\rangle)) = 0) \right. \\ & & \left. * \Pr(\mathsf{last}(A(x = \langle v : t - i\rangle)) = (k - i)) \right) \end{array}$$

- $\Pr(\mathsf{last}((A;_q B)(x = \langle v : t\rangle)) = k)$.

  In this case, we need to consider possible cases into account.

  If $k < q$, then the value never will appear from B, simply because we assume that B will not be executed until time q. In that case, $\Pr(\mathsf{last}((A;_q B)(x = \langle v : t\rangle)) = k) = \Pr(\mathsf{last}(A(x = \langle v : t\rangle)) = k)$.

  In case that $k \geq q$, then there is the possibility that $\Pr(\mathsf{first}(A(x = \langle v : t\rangle)) \leq q)$. We can solve it computing recursively the probability as $\Pr(\mathsf{last}((A;_q B)(x = \langle v : t\rangle)) = k) = \Pr(\mathsf{first}(A(x = \langle v : t\rangle)) \leq q) * \Pr(\mathsf{last}(A(x = \langle v : t - q\rangle)) = k | \mathsf{first}(A(x = \langle v : t - q\rangle)) \leq q) + (1 - \Pr(\mathsf{first}(A(x = \langle v : t\rangle)) \leq q)) * \Pr(\mathsf{last}(B(x = \langle v : t - q\rangle)) = k - q)$.

  For $k = q$ we have the limit case. We have that $\Pr(\mathsf{first}(A(x = \langle v : t\rangle)) = k)$ return the probability that the first value is published by A in time k, the in a similar way as for $k > q$, we have that B will publish the first value if A takes times $> q$ for its first value and then B takes time 0 to publish its first value.

We claim that $\Pr(\mathsf{last}(E) = k | \mathsf{first}(E) \leq t)$ can be computed in poly-space with a similar algorithm asking in parallel composition the probability that one side has $\mathsf{first} \leq t$ and to the other the probability that $\mathsf{last} = k$ and asking to each side the probability that $\mathsf{last} = k$ given that $\mathsf{first} \leq t$. □

Given this outline of algorithm, we claim that $\Pr(\mathit{last}(A) = k)$ *belongs to PSPACE.*

Then, if we can compute $\Pr(\mathsf{last}(E) = k)$ in poly-space, we can compute also $\Pr(\mathsf{last}(E) > k)$ and $\mathbb{E}(\mathsf{last}(E))$.

Theorem 8.3.3. *In the Oblivious model, the problem PrLast belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Given an expression E and a value k, we can compute $\Pr(\mathsf{last}(E) > k)$ with $k + 1$ recursive calls to $\Pr(\mathsf{last}(E) = i)$. Then, we can solve PrLast at follows:

$$\Pr(\mathsf{last}(E) > k) = 1 - \sum_{i=0}^{k} \Pr(\mathsf{last}(E) = i)$$

As $\Pr(\mathsf{last}(E) = i)$ can be computed in poly-space, also $\Pr(\mathsf{last}(E) > k)$ can be done in poly-space. □

Theorem 8.3.4. *In the Oblivious model, the problem ExpLast belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Given an expression E with n site calls, if the all sites publish in at most δ_{max} and otherwise does not publish, then any value published by E will take at most $n * \delta_{max}$ time. Given that, we can compute $\mathbb{E}(\mathsf{first}(E))$ with recursive calls to $\Pr(\mathsf{last}(E) = i)$ for $0 \leq i \leq n * \delta_{max}$. Then, we can solve $\mathsf{ExpLast}(E)$ at follows:

$$\mathbb{E}(\mathsf{last}(E)) = \sum_{i=0}^{n*\delta_{max}} i * \Pr(\mathsf{last}(E) = i) / \Pr(\mathsf{out}(E) > 0)$$

As both $\Pr(\mathsf{out}(E) > 0)$ and $\Pr(\mathsf{last}(E) = i)$ can be computed in poly-space, then $\mathbb{E}(\mathsf{last}(E))$ can be computed in in poly-space.

In case that $\Pr(\mathsf{out}(E) > 0) = 0$ we say that $\mathbb{E}(\mathsf{last}(E)) = \omega$. □

Now that we know that both PrLast and ExpLast can be computed in poly-space, let see some lower bound on their complexity for the different sub-families. First, let us see an interesting relation between PrLast and PrProduction.

Lemma 8.3.5. *In the Oblivious Stable or Typed model, the problem PrProduction reduces to PrLast when restricted to a finite Orc expression.*

Proof. We have seen that for any expression E with n site calls, if δ_{max} is the maximum time needed for any site when it publish, then any value published by an execution of E will be published within time $n * \delta_{max}$. Given that, is easy to see that:

$$\Pr(\mathsf{out}(E) > 0) = 1 - \Pr(\mathsf{last}(E) > n * \delta_{max})$$

□

Given that relation the we have that PrLast is at least as hard as PrProduction.

Theorem 8.3.6. *In the Oblivious model, the problem PrLast is ♯P-Hard when restricted to PruningOrc and IfOrc.*

Proof. Direct from Lemma 8.3.5, as PrProduction is ♯P-Hard for these sub-families of Orc, PrLast is also ♯P-Hard. □

For ExpLast(E) we cannot take the same approach from PrProduction as executions that not publish are not considered while computing ExpLast, but we can find similar results.

Theorem 8.3.7. *In the Oblivious model, the problem ExpLast is ♯P-Hard when restricted to PruningOrc.*

Proof. Let us consider the following reduction from counting MONOTONE 2-SAT.

Given a MONOTONE 2-CNF 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 *DelayCoin* be a site with delay profile $(0@1/2 \parallel 1@1/2)$. Let us define the orchestration E_F as follows:

for $1 \leq i \leq m$ do $E_{C_i}(y_{i_a}, y_{i_b}) = (1(y_{i_a})|1(y_{i_b}))$
$E(x_1, \ldots, x_n) = (E_{C_1}(y_{1_a}, y_{1_b}) \gg \cdots \gg E_{C_m}(y_{m_a}, y_{m_b}))$
$E_F = (\cdots(E(x_1, \ldots, x_n) < x_1 < \mathit{DelayCoin}) \cdots) < x_n < \mathit{DelayCoin}$

Given $v = (v_1, \ldots v_n)$ such that v_i has delay $\in \{0, 1\}$ let us define map a boolean version $b(v_1, \ldots v_n) = (b(v_1), \ldots b(v_n))$ such that for $1 \leq i \leq n$

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

$$\mathsf{last}(E(v_1, \ldots, v_n)) = \begin{cases} 1 & \text{if } F(b(v_1), \ldots b(v_n)) = 1, \\ 0 & \text{otherwise} \end{cases}$$

As *DelayCoin* follows a delay profile $(0@1/2 \parallel 1@1/2)$, each $v \in \{0,1\}^n$ occurs with probability $1/2^n$, then $\mathbb{E}(\mathsf{last}(E_F)) = (1/2^n) * \sum_{v \in \{0,1\}^n} \mathsf{last}(E(v)) = (1/2^n) * \sum_{v \in \{0,1\}^n} F(b(v)) = \sharp F/2^n$. Therefore, $2^n\, \mathbb{E}(\mathsf{last}(E_F))$ solves counting MONOTONE 2-SAT. □

Theorem 8.3.8. *In the Oblivious model, the problem ExpLast is ♯P-Hard when restricted to IfOrc.*

Proof. Let us consider the following reduction from counting 3-SAT.

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, 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 $1 \leq i \leq n$. 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 us encode clause failures as follow. Given $C_i = y_{i_a} \vee y_{i_b} \vee y_{i_c}$ define

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

Taking *Coin* as a site following a value profile $V_{Coin} = (\mathit{true}@1/2 \parallel \mathit{false}@1/2)$, the orchestration E_F associated to F is:

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

Given $v = (v_1, \ldots v_n)$ such that $v_i \in \{0, 1\}$ we have that:

$$\mathsf{last}(E(v)) = \begin{cases} 1 & \text{if } F(v_1, \ldots v_n) = 1, \\ 0 & \text{otherwise} \end{cases}$$

As each $v \in \{0,1\}^n$ occurs with probability $1/2^n$, then:

$$\mathbb{E}(\mathsf{last}(E_F)) = (1/2^n) * \textstyle\sum_{v\in\{0,1\}^n} \mathsf{last}(E(v)) = \\ (1/2^n) * \textstyle\sum_{v\in\{0,1\}^n} F(v) = \#F/2^n$$

Therefore, $2^n\,\mathbb{E}(\mathsf{last}(E_F))$ solves counting 3-SAT. □

Now, let us see an interesting relation between PrLast and PrFirst.

Lemma 8.3.9. *Given a expression E, if for any execution of E, all values are produce at the same time, then $\mathbb{E}(\mathit{first}(E)) = \mathbb{E}(\mathit{last}(E))$ and $\Pr(\mathit{first}(E) \leq k) = 1 - \Pr(\mathit{last}(E) > k)$*

That is because if all values take the same delay during a evaluation then that delay is both the first and the last time.

For ElementaryOrc and ExtElementaryOrc In Oblivious model, the complexity lower-bound of PrLast and ExpLast remain as open questions.

8.3.2 Stable and Typed models

For the Oblivious model we have that PrLast and ExpLast(E) can be computed in poly-space for all the sub-families studied, now let us see what happens in case of expressions in the Stable and Typed models.

Theorem 8.3.10. *In the Typed model, the problem PrLast belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. For the Stable and Typed models we can compute $\Pr(\mathsf{last}(E) > k)$ in the following way:

$$\Pr(\mathsf{last}(E) > k) = \sum_{Q\in G(\mathcal{P})} \Pr(\mathsf{last}(E|Q) > k) * \Pr(Q)$$

$E|Q$ is an expression following a Oblivious probabilistic environment, then we can compute $\Pr(\mathsf{out}(E|Q) > 0)$ in poly-space. Given that, we can simply compute $\Pr(\mathsf{out}(E) > 0)$ computing $\Pr(\mathsf{out}(E|Q) > 0)$ recursively for each $Q \in G(\mathcal{P})$. □

Theorem 8.3.11. *In the Typed model, the problem ExpLast belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. For the Stable and Typed models we can compute $\mathbb{E}(\mathsf{last}(E))$ in the following way:

$$\mathbb{E}(\mathsf{last}(E)) = \sum_{Q\in G(\mathcal{P});\ \Pr(\mathsf{out}(E|Q)>0)} \mathbb{E}(\mathsf{last}(E|Q)) * \Pr(Q)$$

□

Theorem 8.3.12. *In the Stable model, the problem PrLast is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Direct from Lemma 8.3.5, as PrProduction is ♯P-Hard for ElementaryOrc sub-families of Orc, PrLast is also ♯P-Hard. □

Theorem 8.3.13. *In the Stable model, the problem ExpLast is ♯P-Hard when restricted to ElementaryOrc.*

Proof. Let us consider the following reduction from counting MONOTONE 2-SAT.

Given a MONOTONE 2-CNF 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 $1 \leq i \leq m$.

For $1 \leq i \leq n$, let X_i be a site following the status profile $\mathcal{E}_{X_i}$:

$$\begin{array}{rcl} \mathcal{E}_{X_i} & = & (P_{itrue}@1/2 \parallel P_{ifalse}@1/2) \\ \mathcal{E}_{itrue} & = & (\langle 1:0\rangle\,@1) \\ \mathcal{E}_{ifalse} & = & (\langle \bot:\omega\rangle\,@1) \end{array}$$

Then for $1 \leq i \leq m$ define $E_{C_i} = (X_{i_a}|X_{i_b})$, where i_a and i_b are the indexes of the variables used in clause C_i. Then we define the expression E_F associated to F as follows:

$$E_F = (E_{C_1} \gg \cdots \gg E_{C_m}) \mid Rtimer(1)$$

Given a global profile $Q = (q_1 \cdots q_n)$ by construction $\mathbb{E}(\mathsf{first}(E_F|s)) = 1 - F(s)$.

As for any $Q \in \{0,1\}^n$, $\Pr(Q) = 1/2^n$ it holds $\mathbb{E}(\mathsf{first}(E_F)) = \sum_{Q\in\{0,1\}^n} \mathbb{E}(\mathsf{first}(E_F|Q))/2^n = (1/2^n)\sum_{\mathtt{Q}\in\{0,1\}^n}(1 - F(Q)) = (1 - \#F)/2^n$.

Then, $2^n(1 - \mathbb{E}(\mathsf{first}(E_F)))$ is the number of satisfying assignments for F. □

8.4 MaxLast

The problem MaxLast returns, given an Orc expression, the maximum possible delay of a value published during an execution. Let us analyse it under the Oblivious, Stable and Typed models.

8.4.1 Oblivious model

Let us see an interesting relation between compute MaxLast in the Oblivious model and compute last in the Deterministic model.

Theorem 8.4.1. *In the Oblivious model, the problem MaxLast reduces to last in Deterministic model when restricted to ElementaryOrc.*

Proof. For this sub-families, MaxLast can be seen as compute last given that all sites publish always and with their maximum delay. That is because, as the values of variables does not matter to the delay of sites, is clear that the maximum possible delay will correspond to an execution path when all sites take their maximum delay. □

Theorem 8.4.2. *In the Oblivious model, the problem MaxLast reduces to last in Deterministic model when restricted to IfOrc.*

Proof. Given a site $S(x_1, \ldots, x_n)$, for each value $v_i \in \mathbb{D}(S)$ we define $V_{Si}(w_1, \ldots, w_n)$ as a site publishing v_i with the maximum delay for which a call to $S(w_1, \ldots, w_n)$ can publish v_i (without consider ω, unless v_i cannot be published). Then, given an expression E, we define E' as a copy of E replacing each site S by a sub-expression $(V_{S1} \mid V_{S1} \mid \ldots)$.

Now, we can solve MaxLast of E as compute last of E', given that E' follow all the possible execution paths of E for all the possible values with their maximum delay. □

Observe that for ExtIfOrc and ExtMixedOrc we cannot take the same approach, as for $A;_t B$ it is possible that an execution through A trigger the publication of the value with maximum delay, and we can lose that value if there are no paths publishing through A in lees than time t.

In a similar way, for PruningOrc, ExtPruningOrc, MixedOrc and ExtMixedOrc, the pruning operator has the problem that with nested pruning the maximum delay comes either from an execution where some sites fail at some point.

In Theorem 8.3.3 we have seen that $\Pr(\mathsf{last}(E) > t)$ can be computed in poly-space for all the sub-families seen. Now let us use that to give upper bounds to MaxLast.

Theorem 8.4.3. *In the Oblivious model, the problem MaxLast belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Given a expression E, we can compute MaxLast recursively with calls to $\Pr(\mathsf{last}(E) > t)$. First, let $prOut = \Pr(\mathsf{out}(E) \neq 0)$ For that, starting with $i = 0$ we check $\Pr(\mathsf{last}(E) > i) > prOut$ and go incrementing i until $\Pr(\mathsf{last}(E) > i) > prOut$. The previous delay to the i for which $\Pr(\mathsf{last}(E) > i) = prOut$ will be MaxLast. □

For ExtElementaryOrc, PruningOrc and ExtPruningOrc In Oblivious model, the complexity lower-bound of $\mathsf{MaxLast}(E)$ remain as an open questions.

8.4.2 Stable and Typed models

For the Oblivious model we have that MaxLast can be computed in poly-space for all the sub-families studied, now let us see what happens in case of expressions in the Stable and Typed models.

Theorem 8.4.4. *In the Typed model, the problem MaxLast belongs to PSPACE when restricted to ExtMixedOrc.*

Proof. Using the function max^* (Definition 5.9), for the Stable and Typed models we can compute MaxLast in the following way:

$$\max(\mathsf{last}(E_F)) = max^*_{Q \in G(\mathcal{P})} \mathsf{MaxLast}(E|Q)$$

□

Theorem 8.4.5. *In the Typed model, the problem MaxLast is NP-Hard when restricted to ElementaryOrc.*

Proof. Let us remind the reduction from counting 3-SAT used to prove $\mathbb{E}(\mathsf{out})$ in ♯P-Hard also for the Typed model (Theorem 7.2.10).

Given a 3-CNF formula $F = C_1 \wedge \ldots \wedge C_m$ with n literals $x_1, \ldots, x_n$, we had transformations of clauses into sub-expressions using sites as variables, in the form:

$$C_i = (x_1 \vee x_2 \vee x_3) \Leftrightarrow E_{C_i} = (X_1 \mid (\overline{X_1} \gg (X_2 \mid (\overline{X_2} \gg X_3))))$$

Each pair of sites $(X_i, \overline{X_i})$ follow a type profile where only one site succeeds on its calls, now let us add time to this profile:

$$\mathcal{E}_{(X_i,\overline{X_i})} = \quad \Big((\{\langle \bot : \omega \rangle \, @1\}, \{\langle 1:0 \rangle \, @1\})@1/2 \parallel (\{\langle 1:0 \rangle \, @1\}, \{\langle \bot : \omega \rangle \, @1\})@1/2\Big)$$

Then the orchestration E_F corresponding to the formula F was:

$$E_F = E_{C_1} \gg \ldots \gg E_{C_m}$$

Describing a global profile for E_F as a string $Q = q_1, \ldots, q_n \in \{0,1\}$, where $q_i = 1$ means X_i success and $\overline{X_i}$ fails and otherwise for $q_i = 0$ By construction of E_F we have that $F(Q) = 1 \Leftrightarrow \mathsf{MaxLast}(E_F|Q) = 0$. Then, we have that:

$$\max(\mathsf{last}(E_F)) = 0 \Leftrightarrow \exists Q \in \{0,1\}^n \text{ s.t. } (\mathsf{out}(E_F|Q) = 1) \Leftrightarrow$$
$$\exists Q \in \{0,1\}^n \text{ s.t. } (F(Q) = 1) \Leftrightarrow F \in 3-SAT$$

□

For ElementaryOrc, ExtElementaryOrc, PruningOrc and ExtPruningOrc In Stable model, the complexity lower-bound of MaxLast remain as an open questions.

Problem	*Model*	**ElemOrc**	**EElemOrc**	**PruOrc**	**EPruOrc**	**IfOrc**	**EIfOrc**	**MixOrc**	**EMixOrc**
PrFirst	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	____	_____	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
ExpFirst	*Deterministic*	P	P	P	P	♯P-H	♯P-H	♯P-H	♯P-H
	Oblivious	____	_____	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
MinFirst	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Stable	____	_____	___	____	NP-H	NP-H	NP-H	NP-H
	Typed	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H

Table 8.1: Resume of problems related to the first delay of outputs with probabilistic environments. All problems belong to PSPACE.

Problem	*Model*	**ElemOrc**	**EElemOrc**	**PruOrc**	**EPruOrc**	**IfOrc**	**EIfOrc**	**MixOrc**	**EMixOrc**
PrLast	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	———	———	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
ExpLast	*Deterministic*	P	P	P	P	♯P-H	♯P-H	♯P-H	♯P-H
	Oblivious	———	———	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
MaxLast	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	P	———	———	———	NP-H	NP-H	NP-H	NP-H
	Stable	———	———	———	———	NP-H	NP-H	NP-H	NP-H
	Typed	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H

Table 8.2: Resume of problems related to the last delay of outputs with probabilistic environments. All problems belong to PSPACE.

Chapter 9

Conclusions & Future Work

Applications based on the use of external web-services in order to reduce the computational burden are becoming more important as cloud computing becomes a common model, especially for inexpensive devices. The Orc Language is an easy but flexible language for describe the orchestration of those services, that abstracts from the functionality of the services used, basing on the connection between them.

In order to build better and more efficient orchestrations, we need some measure of the goodness of an orchestration. For that, we have focused on measures related to the number of values published and the delay from the start of the execution until the publication of values.

As Orc language can be a really powerful language, for which a simple analysis may not even be enough to know even if an execution finishes or not, we focus on few sub-families of Orc for which we ensure not only that the execution will end, but that if the services have a deterministic behaviour and no failure, the number of outputs published will be always the same in each execution. We called those sub-families ElementaryOrc, ExtElementaryOrc, PruningOrc, ExtPruningOrc, IfOrc, ExtIfOrc, MixedOrc and ExtMixedOrc and those were created by adding functionality to the base sub-family ElementaryOrc, a finite sub-family where only operators $|$, $> x >$ and $\gg$ were allowed and sites do not depend on values (see figure 9.1). For example, PruningOrc added the operator $< x <$ to ElementaryOrc and IfOrc added the use of values.

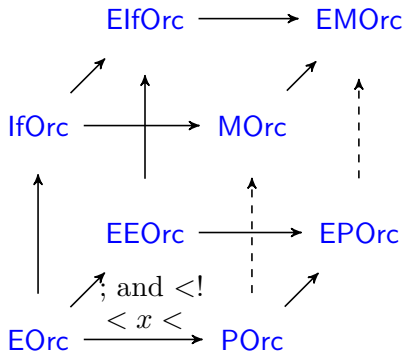


Figure 9.1: Visual representation of the relationships in Orc sub-families.

Given that, for orchestrations where sites where deterministic, we studied the following problems:

- Output. Compute the number of outputs of an execution.
- Produces. Solve if an execution will or not publish some values during an execution.
- OutputValue. Compute the number of outputs of a values $= v$ during an execution.
- ProducesValue. Solve if an execution will or not publish some values $= v$ during an execution.

We have also seen that if site calls are also deterministic in the time needed, for those families, the results published follows a certain publication times. In those cases, the following problems where studied:

- first. Time required for the first value to be published.
- last. Time required for the last value to be published.
- avgDelay. Average time required for a value to be published.

The assumption of sites with deterministic behaviour was not too realistic, as we have seen that web-services does not have a 100% reliability, sometimes can perform some random computation, can be affected by real life actions,... Given that, we proposed a new framework based on probabilistic sites. For those sites, we defined different profiles for independent calls:

- Delay&Value Profile. Describing the probability of publish a certain value with certain delay or halt.
- Halting&Value Profile. Describing the probability of publish a certain value or halt.
- Delay Profile. Describing the probability that a site call take some time.
- Halting Profile. Describing the probability that a site call succeeds.
- Value Profile. Describing the probability of publish a certain value on a successful call.

We also define 2 different profiles in order to describe the environment of sites:

- Status Profile. Describe the behaviour of a site, given a status of the environment related only to it.
- Type Profile. Describe the behaviour of sites, given a status of the environment related only to them.

With those different probabilistic profiles, then we are able to represent more realistic sites, with probabilities of failures or on random computations an also describe some behaviours of related sites. Then, we extend the deterministic model with 3 new probabilistic models based in probabilistic and independent site calls.

- Oblivious model. The case where the environment has only one possible state.

- Stable model. The case where the environment of each site is independent.
- Typed model. The case where some sites have common environments.

Given these models, we see some QoS measures related to the seen before. For output related problems, we studied the following problems:

- ExpOut. Compute the expected number of outputs of an execution.
- PrProduction. Compute the probability of publish at least one value during an execution.
- PrOut. Compute the probability of publish more than k values during an execution.

And for time related problems:

- ExpFirst. Compute the expected delay of the first published value.
- PrFirst. Compute the probability of publish the first value in time $\leq k$.
- minDelay. Compute the minimum possible delay for the first published item.
- maxDelay. Compute the maximum possible delay for the last published item.
- ExpLast. Compute the expected delay of the last published value.
- PrLast. Compute the probability of publish the last value in time $> k$.
- avgDelay. Compute the average delay of all published values.

We show that all those problems belongs to PSPACE, with the assumption of having "small" domains of values, but different lower bound on complexity were found. As the solution of these problems are not integer or boolean values but decimal values, we need to take into account that there will be some precision errors according to the type of data used, the expression and the probabilistic profile. The following tables resume the results found:

Problem	*Model*	**ElemOrc**	**EElemOrc**	**PruOrc**	**EPruOrc**	**IfOrc**	**EIfOrc**	**MixOrc**	**EMixOrc**
PrProduction	*Deterministic*	P	P	P	P	NP-C	NP-C	NP-C	NP-C
	Oblivious	P	P	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
ExpOut	*Deterministic*	P	P	P	P	♯P-C	♯P-H	♯.NP-C	♯.NP-H
	Oblivious	P	P	♯P-H	♯P-H	♯P-H	♯P-H	♯.NP-H	♯.NP-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯.NP-H	♯.NP-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯.NP-H	♯.NP-H
PrOut	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	———	———	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H

Table 9.1: Resume of problems related to the number of outputs with probabilistic environments. All problems belong to PSPACE.

Problem	*Model*	**ElemOrc**	**EElemOrc**	**PruOrc**	**EPruOrc**	**IfOrc**	**EIfOrc**	**MixOrc**	**EMixOrc**
PrFirst	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	———	———	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
ExpFirst	*Deterministic*	P	P	P	P	♯P-H	♯P-H	♯P-H	♯P-H
	Oblivious	———	———	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
MinFirst	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Stable	———	———	———	———	NP-H	NP-H	NP-H	NP-H
	Typed	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H

Table 9.2: Resume of problems related to the first delay of outputs with probabilistic environments. All problems belong to PSPACE.

Problem	*Model*	**ElemOrc**	**EElemOrc**	**PruOrc**	**EPruOrc**	**IfOrc**	**EIfOrc**	**MixOrc**	**EMixOrc**
PrLast	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	____	_____	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
ExpLast	*Deterministic*	P	P	P	P	♯P-H	♯P-H	♯P-H	♯P-H
	Oblivious	____	_____	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Stable	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
	Typed	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H	♯P-H
MaxLast	*Deterministic*	P	P	P	P	NP-H	NP-H	NP-H	NP-H
	Oblivious	P	_____	____	_____	NP-H	NP-H	NP-H	NP-H
	Stable	____	_____	____	_____	NP-H	NP-H	NP-H	NP-H
	Typed	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H	NP-H

Table 9.3: Resume of problems related to the last delay of outputs with probabilistic environments. All problems belong to PSPACE.

9.1 Future Work

We have seen some QoS measures for orchestrations, both for deterministic sites without failures and for probabilistic sites with failures, but in some cases we were unable to get lower bounds on the complexity of those problems. The unknown lower bounds, all appear on classes where the value does not affect the result to the number of outputs published or the time of their publication. In those cases, it will be interesting to find if there are some lower-bound bigger than P or if they are actually P.

We selected families were the non-determinism on the values defined by pruning does not matter. The next direct step is to extend the problems seen to all the finite Orc expressions, for deterministic and probabilistic models.

Different approaches can be done, but we propose an approach based on delay and value profiles. In that case, we can try to get the multiplicity of values published with minimum delay and get then the probability for each delay. This approach is more or less easy to do for the Deterministic model, but for probabilistic models that may be a bit more complex.

In addition to the problems seen, there can be other interesting QoS measures related to the seen. For example, for the Deterministic model:

- Determine if an expression publish at least k/half of its values within some time t.
- Compute the deviation on delay of published values.
- Compute the probability that the first published value has some value v.

Or for probabilistic models:

- Compute the probability of publishing at least k values.
- Compute the probability that an expression publish at least k/half of its values within some time t.
- Compute the average delay of published values.
- Compute the probability that the first published value has some value v.

On the other hand, besides of study new problems, we can see the studied problems with the addition of new operators extending the language. For example, we propose the following:

- Operator : **Besides** $A : B$

  This operator would be the opposite of otherwise $(A; B)$. In this case, the sub-expression B will start its execution when A publish its first value. In case that A does not publish anything, then B will not be executed.

  In the Deterministic model, if only consider QoS measures for the orchestration, it will be equivalent to $A|(B <!A)$. That is not the case for the probabilistic models.

- Operator : **Last pruning** $A(x) < x|B$

  Modification of the pruning operator where x is not defined by the first value published by B, but by the last value published. As the operator otherwise, sometimes it can be

undecidable if there will be or not new values published by B, in that case we can add a time-out t for within all the values of B need to be published ($A < x|_t B$)

Having measures of the goodness of orchestrations, the next step will be to optimize those orchestrations to get the maximum number of outputs, minimum delays, minimum cost for at least some probability of publish... Optimization problems tend to have high cost when multiple variables are involved, in this case sites. Also, we have that only compute the "values" for a defined set of sites is in most cases practically incommutable, having in many cases complexities from NP-Hard onwards. Even so, orchestrations commonly do not tend to be "really big", as in those case too many things could fail, therefore we find interesting the optimization "small" of orchestrations for which we can compute some goodness measure.

Some approaches based on Game Theory [41] has been done related to sites with deterministic behaviours [14, 16] under site failures. There, a competitive game named *Angel-Daemon* game was proposed. In that game, two players, the *Angel* and the *Daemon*, decided a which fail one in order to increases the number of outputs and other in order to decrease it.

Basing on this idea, we propose new similar games based on the angel-daemon behaviour of players to optimize get orchestrations that, although without optimum, will have a good behaviour under certain assumptions.

1. *Angel-Daemon* with probabilistic sites.

   First, we extend the *Angel-Daemon* for sites with probabilistic behaviour. In this case, instead of selecting the set of sites that will fail, each player has to select a set of "bad" status profiles for his set of sites.

2. *Security-Vs-Attack*.

   As we have seen, an increase on the security contracted affects directly to the behaviour of sites when they are under attack. In this game, the angelic player will select a number of sites to have high security. Then, the daemonic player will select a set of "bad" status profiles for the sites.

   This game goal is to search for the critical services that can cause the worst behaviour under attack in order to avoid the worst case scenario.

3. *Replicate-Vs-Attack*.

   Another way of increasing the goodness of orchestrations is to replicate sites in order to have more security under failures, avoid instantaneous problems,... In this game, the angelic player will add some copies of sites in order to increase the goodness of the orchestration. Then, the daemonic player will select a set of "bad" status profiles for the sites of the resulting orchestration.

   This game goal is to search for the critical services that can cause the worst behaviour under attack for which replication can appease those problems.

For these different games, we can focus either on measures related to the number of outputs as for related to the delay. Also, these games can be extended with different rules as be within a maximum cost, sites that are related under attack or that can not be improved

to a certain extent of without the improvement of another site... Given that, the study of some of these problems can be a good starting point to create best orchestrations of sites, now that cloud computing is on the rise.

Bibliography

[1] Stackoverflow.
http://stackoverflow.com.

[2] Yahoo.
http://yahoo.com.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, Berlin, 2004.

[4] M. AlTurki and J. Meseguer. Real-time rewriting semantics of orc. In *Proceedings of the 9th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 14-16, 2007, Wroclaw, Poland*, pages 131–142. ACM, 2007.

[5] M. AlTurki and J. Meseguer. Reduction Semantics and Formal Analysis of Orc Programs. *Electr. Notes Theor. Comput. Sci.*, 200(3), 2008.

[6] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.

[7] Amazon. AWS Service Health Dashboard.
http://status.aws.amazon.com.

[8] W. Desk. Turkish hackers deface Google Pakistan's webpage, Nov 2012.
http://tribune.com.pk/story/470594/
turkish-hackers-deface-google-pakistans-webpage/.

[9] J. S. Dong, Y. Liu, J. Sun, and X. Zhang. Verification of Computation Orchestration Via Timed Automata. In *Formal Methods and Software Engineering, 8th International Conference on Formal Engineering Methods, ICFEM 2006*, volume 4260 of *LNCS*, pages 226–245. Springer, 2006.

[10] J. S. Dong, Y. Liu, J. Sun, and X. Zhang. Towards verification of computation orchestration. *Formal Aspects of Computing.*, 2013.

[11] Error 118 (net::ERR_CONNECTION_TIMED_OUT): The operation timed out.
http://productforums.google.com/forum/#!topic/chrome/kPkVzx_HQWI[1-25-false].

[12] A script running on this page is taking a loooong time to do its job.
https://support.google.com/chrome/answer/96804.

[13] serviceuptime - Services Statistics : 000webhost.com.
http://www.serviceuptime.com/users/monitoring.php?S=6c8ef11fcca35bab3d971cdce3d7179b/.

[14] J. Gabarro, A. Garcia, M. Clint, P. Kilpatrick, and A. Stewart. Bounded Site Failures: an Approach to Unreliable Grid Environments. In *Making Grids Work*, pages 175–187. Springer-Verlag, Berlin, 2008.

[15] J. Gabarro, A. Garcia, and M. Serna. Computational aspects of uncertainty profiles and angel-daemon games. 2013. *Theory of Computing Systems*, in press.

[16] J. Gabarro, A. Garcia, M. Serna, P. Kilpatrick, and A. Stewart. Analysing Orchestrations with Risk Profiles and Angel-Daemon Games. In *Grid Computing Achievements and Propects*, pages 121–132. Springer-Verlag, Berlin, 2008.

[17] J. Gabarro, M. Serna, , and A. Stewart. Analysing Web-Orchestrations Under Stress Using Uncertainty Profiles. 2013. *The Computer Journal*, in press.

[18] J. Gabarro, M. Serna, and A. Stewart. Analysing web-orchestrations under stress using uncertainty profiles. *The Computer Journal*, 2013. In press.

[19] Google. Google Apps Status Dashboard.
http://www.google.com/appsstatus .

[20] BBC.
http://www.bbc.co.uk/.

[21] Random.org.
http://www.random.org.

[22] Random.org : Random Byte Generator.
http://www.random.org/cgi-bin/randbyte?nbytes=1024&format=b.

[23] SETI@home.
http://setiathome.berkeley.edu/.

[24] E. C. R. Hehner. Predicative Programming, Part I. *Commun. ACM*, 27(2):134–143, 1984.

[25] E. C. R. Hehner. Predicative Programming, Part II. *Commun. ACM*, 27(2):144–151, 1984.

[26] L. A. Hemaspaandra and H. Vollmer. The satanic notations: counting classes beyond #P and other definitional adventures. *SIGACT News*, 26(1):2–13, Mar. 1995.
http://doi.acm.org/10.1145/203610.203611.

[27] C. Hoare, G. Menzel, and J. Misra. A Tree Semantics of an Orchestration Language. In *Engineering Theories of Software Intensive Systems*, volume 195, pages 331–350. Springer, 2005.

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

[29] N. P. John Markoff. Firm Is Accused of Sending Spam, and Fight Jams Internet, Mar 2013.
http://www.nytimes.com/2013/03/27/technology/internet/online-dispute-becomes-internet-snarling-attack.html.

[30] D. Kitchin, W. R. Cook, and J. Misra. A Language for Task Orchestration and Its Semantic Properties. In *CONCUR 2006 - Concurrency Theory, 17th International Conference*, volume 4137 of *LNCS*, pages 477–491. Springer, 2006.

[31] D. Kitchin, A. Quark, W. Cook, and J. Misra. The Orc Programming Language. In D. Lee, A. Lopes, and A. Poetzsch-Heffter, editors, *Formal Techniques for Distributed Systems, Joint 11th IFIP WG 6.1 International Conference FMOODS 2009 and 29th IFIP WG 6.1 International Conference FORTE 2009, Lisboa, Portugal, June 9-12, Proceedings*, volume 5522 of *LNCS*, pages 1–25. Springer-Verlag, Berlin, 2009.

[32] D. Kitchin, A. Quark, and J. Misra. Quicksort: Combining Concurrency, Recursion, and Mutable Data Structures. In *Reflections on the Work of C.A.R. Hoare", a Festschrift in honor of his 75th birthday*, pages 229–254. Springer-Verlag, 2010.

[33] T. Kitten. DDoS Attacks on Banks: No Break In Sight, Apr 2013.
http://www.bankinfosecurity.com/ddos-attacks-on-banks-no-break-in-sight-a-5708.

[34] Y. N. Maria Garnaeva. DDoS attacks in H2 2011, Feb 2012.
http://www.securelist.com/en/analysis/204792189/DDoS_attacks_in_Q2_2011.

[35] A. McIver and C. Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Springer, Berlin, 2005.

[36] R. Miller. Michigan County Offline After Data Center Fire, Apr 2013.
http://www.datacenterknowledge.com/archives/2013/04/19/michigan-county-offline-after-data-center-fire.

[37] R. Miller. Windows Azure Cloud Crashed by Expired SSL Certificate, Feb 2013.
http://www.datacenterknowledge.com/archives/2013/02/25/windows-azure-cloud-crashed-by-expired-ssl-certificate/.

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

[39] Y. Namestnikov. DDoS attacks in Q2 2011, Aug 2011.
http://www.securelist.com/en/analysis/204792189/DDoS_attacks_in_Q2_2011.

[40] T. Olzak. DNS cache poisoning: Definition and Prevention. 2006.
http://adventuresinsecurity.com/Papers/DNS_Cache_Poisoning.pdf.

[41] M. Osborne. *An Introductions to Game Theory.* Oxford University Press, New York and Oxford, 2004.

[42] M. Papazoglou. *Web Services & SOA: Principles and Technology.* Pearson, London, 2012.

[43] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.

[44] M. Prince. The DDoS That Almost Broke the Internet, Mar 2013. `http://blog.cloudflare.com/the-ddos-that-almost-broke-the-internet`.

[45] J. Proença and D. Clarke. Coordination Models Orc and Reo Compared. *Electr. Notes Theor. Comput. Sci.*, 194(4):57–76, 2008.

[46] A. Stewart, J. Gabarro, and A. Keenan. Reasoning about orchestrations of web services using partial correctness, 2012. *Formal Aspects of Computing*, only electronic access.

[47] R. Stross. 99.999% Reliable? Don't Hold Your Breath, Jan 2011. `http://www.nytimes.com/2011/01/09/business/09digi.html`.

[48] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[49] D. Vardoulakis and M. Wand. A Compositional Trace Semantics for Orc. In *Coordination Models and Languages, 10th International Conference, COORDINATION 2008, Oslo, Norway, June 4-6, Proceedings*, volume 5052 of *LNCS*, pages 331–346, Berlin, 2008. Springer-Verlag, Berlin.

[50] W3C. Denial-of-service attack. `http://www.w3.org/Security/Faq/wwwsf6.html`.

[51] W3C. W3C, Web Services Glossary. `http://www.w3.org/TR/ws-gloss/`.

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

[53] Wikipedia. Website defacement. `http://en.wikipedia.org/wiki/Website_defacement`.

Appendix A

Java Library

Together with the text of the thesis, we provide 3 little Java libraries based on the algorithms seen. In this chapter we provide a description of each part of the libraries and how to use them. The 3 libraries are the following:

- **ProbOrcSignal**. Library for compute Output and Produces of ExtPruningOrc expressions in the oblivious model.
- **ProbOrcBool**. Library for compute Output and Produces of ExtMixedOrc expressions in the oblivious model.
- **VModelOrc**. Library for execute Orc expressions by the fully defined variables model.

A.1 Working with expressions

All 3 libraries are based on a class singleton class *ORC* that is the responsible of define and store expressions along with to manage the computations.

In order to define expression we use the functions *AddExpression* and *AddSite*. During the definition, we can refer to sites and expressions not yet available, given that, it is possible to define recursive expressions. We can also redefine expressions and sites.

Example A.1 Let us consider the PruningOrc expression *SendNews* (Example 3.1) :

$$\begin{aligned} SendNews &= (AddressAlice > a > Email(a,m)) \\ &\quad < m < (CNN \mid BBC) \end{aligned}$$

This expression can be codified as:

```
ORC o = ORC.Singleton();
o.AddExpression("SendNews() = ( AddressAlice() >a> Email(a,m) )
                                                <m< (CNN() | BBC() ) ");
```

In Example 4.3, we have seen this expression rewritten as:

$$\begin{aligned} A(m) &= (AddressAlice > a > Email(a,m)) \\ B &= (CNN \mid BBC) \\ SendNews &= A < m < B \end{aligned}$$

```
ORC o = ORC.Singleton();
o.AddExpression("A() = AddressAlice() >a> Email(a,m)");
o.AddExpression("B() = CNN() | BBC() ");
o.AddExpression("SendNews() = A(m) <m< B() ");
```

When writing expressions, calls to sites and other expressions are codified using the notation $Name(x_1, \ldots, x_n)$ in case of a call with parameters $x_1, \ldots, x_n$ and $Name()$ for a call without parameters.

A.2 Defining sites

Each library has is own kind of site.

- **ProbOrcSignal : SignalSite**.
- **ProbOrcBool : BoolSite**.
- **VModelOrc : Site**.

A.2.1 SignalSite

Sites used to compute Output and Produces on ExtPruningOrc expressions. These sites are described by the probability of success given when initializing it. That probability can be given as a double or preferably as a fraction, with the numerator and denominator.

Example A.2 Let us consider that sites *CNN* and *BBC* have probabilities of success 9/10 and 19/20, then we define these sites as follows:

```
ORC o = ORC.Singleton();
o.AddSite("CNN", SignalSite(9,10));
o.AddSite("BBC", SignalSite(19,20));
```

A.2.2 BoolSite

Sites used to compute Output and Produces on Boolean ExtMixedOrc expressions. The base class is abstract and we need to generate our own sites extending it.

```
public abstract class BoolSite {
    abstract public BigFraction P(Boolean[] vals, BigFraction postF,
                                                  BigFraction postT);
    abstract public BigFraction E(Boolean[] vals, Boolean r);
}
```

We need to implement the methods $BigFractionP$ and $BigFractionE$.

The method P takes the array *vals* as input and *postF* and *postT* as the post probability if produced $false$ or *true*. When implementing it, we need to ensure that it never returns a value bigger than 1.

The method E takes the array $vals$ and a value r as input. It returns the probability of publish the value r in case that $r = true$ or $r = false$ or the probability of publish something in case that $r = null$. When implementing it, we need to ensure that it never returns a value bigger than 1.

Example A.3 Let us consider that sites *Coin* (Example 6.1), then we define this site as follows:

```
public class Coin extends BoolSite{
    @Override
    public BigFraction P(Boolean[] vals, BigFraction postF, BigFraction postT) {
        return postT.add(postF).divide(2);
    }

    @Override
    public BigFraction E(Boolean[] vals, Boolean r) {
        if(r==null){
            return BigFraction.ONE;
        } else {
            return BigFraction.ONE.divide(2);
        }
    }
}
```

In order to compose expressions, there are some operators available:

- Sequential $A >> B$.
- Define $A > x > B$.
- Parallel $A|B$.
- NParallel $|nA$.

  Equivalent to $A|\ldots|A$, for n parallel calls to A.
- Pruning $A < x < B$.
- Signaling $A <!B$.

Site 1() is a default site for all the libraries. Also, site $If(a\ op\ b)$ for *ProbOrcBool* (where a and b can be variables or $true/false$ and op can be $=$, $!=$, $<$, $<=$, $>$ and $>=$).

A.2.3 Site

Sites used to execute expressions with the fully defined variables model. The base class is abstract and we need to generate our own sites extending it.

```
public abstract class Site {
    abstract public ArrayList Ex(Object[] vals);
}
```

We need to implement the method *ArrayListEx*.

The method *Ex* takes the array *vals* as input. It returns an empty list if the site call does not publish and otherwise publish a list with the value published.

Example A.4 Let us consider that sites *Coin* (Example 6.1), then we define this site as follows:

```
public class Coin extends Site{
    @Override
    public ArrayList Ex(Object[] os) {
        ArrayList ret = new ArrayList();
        Random rand = new Random();
        if(rand.nextBoolean()){
            ret.add(true);
        } else {
            ret.add(false);
        }
        return ret;
    }
}
```

As the implementation of sites is open to the user, they can be implemented publishing any number of values, unlike in the standard. In the same way, they can be implemented as local functions or with calls to external services.

A.3 Computing QoS measures and Executing

ProbOrcSignal and *ProbOrcBool* are libraries thought to compute two different QoS measures, the probability of publish something and the expected number of items published during an execution. In both cases, the manager class *ORC* is the responsible of computing these measures.

In order to compute Produces we use the function P, returning the probability as a *BigFraction*:

```
public BigFraction P(String name) throws Exception
public BigFraction P(String name, Boolean[] pars) throws Exception
```

The parameter *name* is the name of an expression or site. The extra parameter *pars* is an array containing the parameters to the call, used for cases of expressions and sites with input parameters.

In order to compute Output we use the function E, returning the expected number of items as a *BigFraction*:

```
public BigFraction E(String name) throws Exception
public BigFraction E(String name, Boolean[] pars) throws Exception
```

The parameter *name* is the name of an expression or site. The extra parameter *pars* is an array containing the parameters to the call, used for cases of expressions and sites with input parameters.

VModelOrc is a library thought to execute Orc expressions using the fully defined variables model. The manager class *ORC* is the responsible of start the execution.

In order to start the execution we use the function *Ex*, returning an array of output values:

```
public ArrayList Ex(String name) throws Exception
public ArrayList Ex(String name, Object[] pars) throws Exception
```

The execution is done sequentially and because that we need to be careful with big outputs and time expensive sites in parallel compositions.

A.4 Code and Dependencies

The libraries and the code can be found on `https://github.com/gaixas1/ProbOrc`. All the code is open and free to modify and use.

There are also two dependencies needed to use it (both included in the folder "dependencies/"):

- **ANTLR v3**. Library used for parsing the Orc expressions.
- **Appache Commons Math**. Library containing the *BigFraction* class used to compute results (only needed for *ProbOrcSignal* and *ProbOrcBool*).

Appendix B

Working examples

Now that we have the models and problems defined, let us see some examples with simple expressions.

Following we have the list of examples and case studies:

1. ***All4All***

$$\begin{array}{ll} TwoNews = & (CNN \mid BBC) \\ SendNews(a) = & (TwoNews > x > Email(a,x)) \\ Addresses = & (AddressAlice \mid AddressBob) \\ All4All = & (Addresses > a > SendNews(a)) \end{array}$$

An ElementaryOrc expression.

Problems:
Deterministic model : Output, Produces, first, last and avgDelay.
Oblivious model : ExpOut, PrProduction, ExpFirst, PrFirst, ExpLast and PrLast.
Typed model : ExpOut and PrProduction.

2. ***MyNews***

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

A PruningOrc expression.

Problems:
Deterministic model : Output and Produces.
Oblivious model : ExpOut and PrProduction.
Typed model : ExpOut and PrProduction.

3. ***OneOrTwo***

$$\begin{array}{ll} OneOrTwo = & Coin > b > \\ & (If(b) \gg CNN \mid If(\neg b) \gg (CNN \mid BBC)) \end{array}$$

An IfOrc expression.

Problems:
Oblivious model : ExpOut, PrProduction, ExpFirst and PrFirst.
Stable model : ExpOut and PrProduction.

4. ***Cron***

$$\begin{array}{lcl} Backup & = & DB.new > x > BackupServer.send(x) \\ & & \quad > d > BackupDB.save(d) \\ DBActions & = & (Backup \mid DB.day()) \\ Cron & = & Server.access() \gg DBActions > y > Mail.send(y) \end{array}$$

A site composed by different servers expressed by an ElementaryOrc expression.

Problems:
Halting profiles under the Typed Model.

B.1 *All4All*

$$\begin{array}{ll} TwoNews = & (CNN \mid BBC) \\ SendNews(a) = & (TwoNews > x > Email(a, x)) \\ Addresses = & (AddressAlice \mid AddressBob) \\ All4All = & (Addresses > a > SendNews(a)) \end{array}$$

All4All is an ElementaryOrc expression, therefore we can consider that all sites publish a signal. Let us analyse it under Deterministic, Oblivious and Typed models.

B.1.1 Deterministic Model

For the deterministic model all sites success in any call, also we say that sites have the following delays:

Site	Delay
CNN	500
BBC	600
AddressAlice	200
AddressBob	200
Email	800

Let us compute first $\mathsf{out}(All4All)$:

$$\begin{array}{ll} \mathsf{out}(All4All) = & \mathsf{out}(Addresses) * \mathsf{out}(SendNews) \\ \mathsf{out}(Addresses) = & \mathsf{out}(AddressAlice) + \mathsf{out}(AddressBob) = 2 \\ \mathsf{out}(SendNews) = & \mathsf{out}(TwoNews) * \mathsf{out}(Email) \\ & \quad = \mathsf{out}(TwoNews) * 1 \\ \mathsf{out}(TwoNews) = & \mathsf{out}(CNN) + \mathsf{out}(BBC) = 2 \\ \mathsf{out}(SendNews) = & 2 \\ \mathsf{out}(All4All) = & 2 * 2 = 4 \end{array}$$

As $\mathsf{out}(All4All) = 4$, $All4All \in$ Produces.

Now, let us compute $\mathsf{first}(All4All)$ and $\mathsf{last}(All4All)$:

$$
\begin{array}{ll}
\mathsf{first}(All4All) = & \mathsf{first}(Addresses) + \mathsf{first}(SendNews) \\
\mathsf{first}(Addresses) = & Min(\mathsf{first}(AddressAlice), \mathsf{first}(AddressBob)) \\
& \quad = Min(200, 200) = 200 \\
\mathsf{first}(SendNews) = & \mathsf{first}(TwoNews) + \mathsf{first}(Email) \\
& \quad = \mathsf{first}(TwoNews) + 800 \\
\mathsf{first}(TwoNews) = & Min(\mathsf{first}(CNN), \mathsf{first}(BBC)) \\
& \quad = Min(500, 600) = 500 \\
\mathsf{first}(SendNews) = & 500 + 800 = 1300 \\
\mathsf{first}(All4All) = & 200 + 1300 = 1500
\end{array}
$$

$$
\begin{array}{ll}
\mathsf{last}(All4All) = & \mathsf{first}(Addresses) + \mathsf{first}(SendNews) \\
\mathsf{last}(Addresses) = & Max(\mathsf{first}(AddressAlice), \mathsf{first}(AddressBob)) \\
& \quad = Max(200, 200) = 200 \\
\mathsf{last}(SendNews) = & \mathsf{first}(TwoNews) + \mathsf{first}(Email) \\
& \quad = \mathsf{first}(TwoNews) + 800 \\
\mathsf{last}(TwoNews) = & Max(\mathsf{first}(CNN), \mathsf{first}(BBC)) \\
& \quad = Max(500, 600) = 600 \\
\mathsf{last}(SendNews) = & 600 + 800 = 1400 \\
\mathsf{last}(All4All) = & 200 + 1400 = 1600
\end{array}
$$

Finally, let us compute $\mathsf{avgDelay}(All4All)$:

$$
\begin{array}{ll}
\mathsf{avgDelay}(All4All) = & \mathsf{avgDelay}(Addresses) + \mathsf{avgDelay}(SendNews) \\
\mathsf{avgDelay}(Addresses) = & AVG(\mathsf{first}(AddressAlice), \mathsf{first}(AddressBob)) \\
& \quad = AVG(200, 200) = 200 \\
\mathsf{avgDelay}(SendNews) = & \mathsf{avgDelay}(TwoNews) + \mathsf{avgDelay}(Email) \\
& \quad = \mathsf{first}(TwoNews) + 800 \\
\mathsf{avgDelay}(TwoNews) = & AVG(\mathsf{first}(CNN), \mathsf{first}(BBC)) \\
& \quad = AVG(500, 600) = 550 \\
\mathsf{avgDelay}(SendNews) = & 550 + 800 = 1450 \\
\mathsf{avgDelay}(All4All) = & 200 + 1450 = 1550
\end{array}
$$

It holds that $\mathsf{first}(All4All) \leq \mathsf{avgDelay}(All4All) \leq \mathsf{last}(All4All)$ $(1500 < 1550 < 1600)$.

B.1.2 Oblivious Model

For the Oblivious model all sites fail in independent calls with certain probability and also can take different times to publish. The following table shows the probability of each delay, we take failure as delay ω:

Site	Delay
CNN	$500@1/2 \parallel 700@1/3 \parallel \omega@1/6$
BBC	$600@3/4 \parallel 700@1/6 \parallel \omega@1/12$
AddressAlice	$200@1/2 \parallel 250@9/20 \parallel \omega@1/20$
AddressBob	$200@1/2 \parallel 250@9/20 \parallel \omega@1/20$
Email	$800@1/2 \parallel 1000@1/2$

Let us compute first $\mathbb{E}(\mathsf{out})$ and $\Pr(\mathsf{out} > 0)$.

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{All4All})) = & \mathbb{E}(\mathsf{out}(\mathit{Addresses})) * \mathbb{E}(\mathsf{out}(\mathit{SendNews})) \\
\mathbb{E}(\mathsf{out}(\mathit{Addresses})) = & \mathbb{E}(\mathsf{out}(\mathit{AddressAlice})) + \mathbb{E}(\mathsf{out}(\mathit{AddressBob})) \\
 & \quad = 19/20 + 19/20 = 19/10 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews})) = & \mathbb{E}(\mathsf{out}(\mathit{TwoNews})) * \mathbb{E}(\mathsf{out}(\mathit{Email}) \\
 & \quad = \mathsf{out}(\mathit{TwoNews})) * 1 \\
\mathbb{E}(\mathsf{out}(\mathit{TwoNews})) = & \mathbb{E}(\mathsf{out}(\mathit{CNN})) + \mathbb{E}(\mathsf{out}(\mathit{BBC})) \\
 & \quad = 5/6 + 11/12 = 7/4 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews})) = & 7/4 \\
\mathbb{E}(\mathsf{out}(\mathit{All4All})) = & (19/10) * (7/4) = 133/40 = 3.325
\end{array}
$$

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{All4All}) > 0) = & \Pr(\mathit{All4All}, 1) \\
\Pr(\mathit{All4All}, 1) = & \Pr(\mathit{Addresses}, \Pr(\mathit{SendNews}, 1)) \\
\Pr(\mathit{SendNews}, 1) = & \Pr(\mathit{TwoNews}, \Pr(\mathit{Email}, 1)) \\
 & \quad = \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{TwoNews}, 1) = & 1 - (1 - \Pr(\mathit{CNN}, 1)) * (1 - \Pr(\mathit{BBC}, 1)) \\
 & \quad = 1 - (1/6) * (1/12) = 71/72 \\
\Pr(\mathit{SendNews}, 1) = & 71/72 \\
\Pr(\mathit{All4All}, 1) = & \Pr(\mathit{Addresses}, 71/72) \\
\Pr(\mathit{Addresses}, 71/72) = & 1 - (1 - \Pr(\mathit{AddressAlice}, 71/72)) * (1 - \Pr(\mathit{AddressBob}, 71/72)) \\
 & \quad = 1 - (1 - 1349/1440) * (1 - 1349/1440) = 2065319/2073600 \\
\Pr(\mathit{All4All}, 1) = & 2065319/2073600 \approx 0.996
\end{array}
$$

For $\mathbb{E}(\mathsf{first})$ and $\Pr(\mathsf{first} \leq k)$, let us compute the delay distribution of $\mathsf{first}(\mathit{All4All})$[1]

[1] We are going to use + and ∗ to describe sequential and parallel composition of distributions.

$\mathsf{first}(\mathit{All4All}, [0]) = \mathsf{first}(\mathit{Addresses}, \mathsf{first}(\mathit{SendNews}, [0]))$
$\mathsf{first}(\mathit{SendNews}, [0]) = \mathsf{first}(\mathit{TwoNews}, \mathsf{first}(\mathit{Email}, [0])) =$
$\quad \mathsf{first}(\mathit{TwoNews}, (800@1/2 \parallel 1000@1/2))$
$\mathsf{first}(\mathit{TwoNews}, (800@1/2 \parallel 1000@1/2)) =$
$\quad \mathsf{first}(\mathit{CNN}, (800@1/2 \parallel 1000@1/2)) + \mathsf{first}(\mathit{BBC}, (800@1/2 \parallel 1000@1/2))$
$\mathsf{first}(\mathit{CNN}, (800@1/2 \parallel 1000@1/2)) =$
$\quad (500@1/2 \parallel 700@1/3 \parallel \omega@1/6) * (800@1/2 \parallel 1000@1/2) =$
$\quad (1300@1/4 \parallel 1500@5/12 \parallel 1700@1/6 \parallel \omega@1/6)$
$\mathsf{first}(\mathit{BBC}, (800@1/2 \parallel 1000@1/2)) =$
$\quad (600@3/4 \parallel 700@1/6 \parallel \omega@1/12) * (800@1/2 \parallel 1000@1/2) =$
$\quad (1400@3/8 \parallel 1500@1/12 \parallel 1600@3/8 \parallel 1700@1/12 \parallel \omega@1/12)$
$\mathsf{first}(\mathit{TwoNews}, (800@1/2 \parallel 1000@1/2)) =$
$\quad (1300@1/4 \parallel 1500@5/12 \parallel 1700@1/6 \parallel \omega@1/6) +$
$\qquad (1400@3/8 \parallel 1500@1/12 \parallel 1600@3/8 \parallel 1700@1/12 \parallel \omega@1/12) =$
$\quad (1300@1/4 \parallel 1400@9/32 \parallel 1500@83/288 \parallel 1600@1/8 \parallel 1700@1/24 \parallel \omega@1/72)$
$\mathsf{first}(\mathit{Addresses}, (1300@1/4 \parallel \ldots \parallel \omega@1/72)) =$
$\quad \mathsf{first}(\mathit{AddressAlice}, (1300@1/4 \parallel \ldots \parallel \omega@1/72)) +$
$\qquad \mathsf{first}(\mathit{AddressBob}, (1300@1/4 \parallel \ldots \parallel \omega@1/72))$
$\mathsf{first}(\mathit{AddressAlice}, (1300@1/4 \parallel \ldots \parallel \omega@1/72)) =$
$\quad (200@1/2 \parallel 250@9/20 \parallel \omega@1/20) * (1300@1/4 \parallel \ldots \parallel \omega@1/72) =$
$\quad (1500@1/8 \parallel 1550@9/80 \parallel 1600@9/64$
$\qquad \parallel 1650@81/640 \parallel 1700@83/576 \parallel 1750@83/640$
$\qquad \parallel 1800@1/16 \parallel 1850@9/160 \parallel 1900@1/48$
$\qquad \parallel 1950@3/160 \parallel \omega@91/1440)$
$\mathsf{first}(\mathit{AddressBob}, (1300@1/4 \parallel \ldots \parallel \omega@1/72)) =$
$\quad \mathsf{first}(\mathit{AddressAlice}, (1300@1/4 \parallel \ldots \parallel \omega@1/72))$
$\mathsf{first}(\mathit{Addresses}, (1300@1/4 \parallel \ldots \parallel \omega@1/72)) =$
$\quad (1500@1/8 \parallel \ldots \parallel \omega@91/1440) * (1500@1/8 \parallel \ldots \parallel \omega@91/1440) \approx$
$\quad (1500@0.2344 \parallel 1550@0.1842 \parallel 1600@0.1947$
$\qquad \parallel 1650@0.1414 \parallel 1700@0.122 \parallel 1750@0.0743$
$\qquad \parallel 1800@0.0238 \parallel 1850@0.0147 \parallel 1900@0.0038$
$\qquad \parallel 1950@0.0027 \parallel \omega@0.004)$

Given this delay distribution, we can see that it holds $\Pr(\mathsf{out} > 0) = \Pr(\mathsf{first} \neq \omega)$. Now, from this distribution we have: $\mathbb{E}(\mathsf{first}(\mathit{All4All})) = \sum_{(\delta,p)\in\mathsf{first}(\mathit{All4All})} \delta * p / \Pr(\mathsf{out}(\mathit{All4All}) > 0) \approx 1608.34$

B.1.3 Typed Model

For the Typed model we are going to compute $\mathbb{E}(\mathsf{out})$ and $\Pr(\mathsf{out} > 0)$. For this, we need type profiles for the sites. In this case, we say that *CNN* and *BBC* are related, having similar behaviours in time and sites *AddressAlice* and *AddressBob* are in the same server. Given that, the following table show the different type profiles for each site:

Sites	Type Profile
(CNN, BBC)	((1@1), (1 @1/2 $\parallel$ 0 @1/2))@5/6 $\parallel$ ((0@1), (0 @1))@1/6
$(AddressAlice,$ $AddressBob)$	((1 @1), (1 @1))@19/20 $\parallel$ ((0@1), (0 @1))@1/20
Email	(1@1)

In this case we have 4 global profiles.

Sites	Global Profile A	Global Profile B
CNN	(1@1)	(0@1)
BBC	(1 @1/2 $\parallel$ 0 @1/2)	(0@1)
AddressAlice	(1 @1)	(1 @1)
AddressBob	(1 @1)	(1 @1)
Email	(1@1)	(1@1)
Probability	**19/24**	**19/120**
Sites	**Global Profile C**	**Global Profile D**
CNN	(1@1)	(0@1)
BBC	(1 @1/2 $\parallel$ 0 @1/2)	(0@1)
AddressAlice	(0 @1)	(0 @1)
AddressBob	(0 @1)	(0 @1)
Email	(1@1)	(1@1)
Probability	**1/24**	**1/120**

In this case, is clear that with profiles B, C and D, *All4All* does not publish anything, as each execution path goes for *CNN* or *BBC* and for *AddressAlice* or *AddressBob*. Then, let us see what happens with the profile A. As *AddressAlice*, *AddressBob* and *Email* never fail in this case, then :

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{All4All})|A) = & \mathbb{E}(\mathsf{out}(\mathit{Addresses})|A) * \mathbb{E}(\mathsf{out}(\mathit{SendNews})|A) \\
\mathbb{E}(\mathsf{out}(\mathit{Addresses})|A) = & \mathbb{E}(\mathsf{out}(\mathit{AddressAlice})|A) + \mathbb{E}(\mathsf{out}(\mathit{AddressBob})|A) = 2 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews})|A) = & \mathbb{E}(\mathsf{out}(\mathit{TwoNews})|A) * \mathbb{E}(\mathsf{out}(\mathit{Email}|A) \\
 & \quad = \mathsf{out}(\mathit{TwoNews})|A) * 1 \\
\mathbb{E}(\mathsf{out}(\mathit{TwoNews})|A) = & \mathbb{E}(\mathsf{out}(\mathit{CNN})|A) + \mathbb{E}(\mathsf{out}(\mathit{BBC})|A) \\
 & \quad = 1 + 1/2 = 3/2 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews})|A) = & 3/2 \\
\mathbb{E}(\mathsf{out}(\mathit{All4All})|A) = & 2 * (3/2) = 3
\end{array}
$$

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{All4All}) > 0|A) = & \Pr(\mathit{All4All}|A, 1) \\
\Pr(\mathit{All4All}|A, 1) = & \Pr(\mathit{Addresses}|A, \Pr(\mathit{SendNews}|A, 1)) \\
\Pr(\mathit{SendNews}|A, 1) = & \Pr(\mathit{TwoNews}|A, \Pr(\mathit{Email}|A, 1)) \\
 & \quad = \Pr(\mathit{TwoNews}|A, 1) \\
\Pr(\mathit{TwoNews}|A, 1) = & 1 - (1 - \Pr(\mathit{CNN}|A, 1)) * (1 - \Pr(\mathit{BBC}|A, 1)) \\
 & \quad = 1 - (1 - 1) * (1 - 1/2) = 1 \\
\Pr(\mathit{SendNews}|A, 1) = & 1 \\
\Pr(\mathit{All4All}|A, 1) = & \Pr(\mathit{Addresses}|A, 1) \\
\Pr(\mathit{Addresses}|A, 1) = & 1 - (1 - \Pr(\mathit{AddressAlice}|A, 1)) \\
 & \qquad *(1 - \Pr(\mathit{AddressBob}|A, 1)) \\
 & \quad = 1 - (1 - 1) * (1 - 1) = 1 \\
\Pr(\mathit{All4All}|A, 1) = & 1
\end{array}
$$

As the profile A has probability 19/24, we have that $\mathbb{E}(\mathsf{out}(\mathit{All4All})) = \mathbb{E}(\mathsf{out}(\mathit{All4All})|A) * 19/24 = 2.375$ and $\Pr(\mathsf{out}(\mathit{All4All}) > 0) = \Pr(\mathit{All4All}|A, 1) * 19/24 \approx 0.7917$.

B.2 *MyNews*

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

MyNews is an PruningOrc expression, therefore we can consider that all sites publish a signal. Let us analyse it under Deterministic, Oblivious and Typed models.

B.2.1 Deterministic Model

Let us compute first $\mathsf{out}(MyNews)$:

$$
\begin{array}{ll}
\mathsf{out}(MyNews) = & \mathsf{out}(SendNews) * \mathsf{hasOut}(Addresses) \\
\mathsf{out}(Addresses) = & \mathsf{out}(AddressAlice) + \mathsf{out}(AddressBob) = 2 \\
\mathsf{hasOut}(Addresses) = & (\mathsf{out}(Addresses) > 0) = (2 > 0) = 1 \\
\mathsf{out}(SendNews) = & \mathsf{out}(TwoNews) * \mathsf{out}(MyEmails) \\
\mathsf{out}(MyEmails) = & \mathsf{out}(EmailDad) + \mathsf{out}(Email) = 2 \\
\mathsf{out}(TwoNews) = & \mathsf{out}(CNN) + \mathsf{out}(BBC) = 2 \\
\mathsf{out}(SendNews) = & 2 * 2 = 4 \\
\mathsf{out}(MyNews) = & 4 * 1 = 4
\end{array}
$$

As $\mathsf{out}(MyNews) = 4$, then $MyNews \in \mathsf{Produces}$.

B.2.2 Oblivious Model

For the Oblivious model all sites fail in independent calls with certain probability. The following table shows the probability of success/failure for each site:

Site	**Success**	**Failure**
CNN	5/6	1/6
BBC	11/12	1/12
AddressAlice	19/20	1/20
AddressBob	19/20	1/20
Email	1	0
EmailDad	1	0

Let us compute $\Pr(\mathsf{out}(MyNews) > 0)$ and $\mathbb{E}(\mathsf{out}(MyNews))$.

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{All4All}) > 0) = & \Pr(\mathit{MyNews}, 1) \\
\Pr(\mathit{MyNews}, 1) = & \Pr(\mathit{SendNews}(a \neq \bot), 1) * \Pr(\mathit{Addresses}, 1) \\
 & \quad + \Pr(\mathit{SendNews}(a = \bot), 1) * (1 - \Pr(\mathit{Addresses}, 1)) \\
\Pr(\mathit{Addresses}, 1) = & 1 - (1 - \Pr(\mathit{AddressAlice}, 1))(1 - \Pr(\mathit{Bob}, 1)) \\
 & \quad +1 - (1 - 19/20)(1 - 19/20) = 399/400 \\
\Pr(\mathit{SendNews}(a \neq \bot), 1) = & \Pr(\mathit{TwoNews}, \Pr(\mathit{MyEmails}(a \neq \bot), 1)) \\
\Pr(\mathit{MyEmails}(a \neq \bot), 1) = & 1 - (1 - \Pr(\mathit{EmailDad}))(1 - \Pr(\mathit{Email}(a \neq \bot))) \\
 & \quad = 1 - (1 - 1)(1 - 1) = 1 \\
\Pr(\mathit{SendNews}(a \neq \bot), 1) = & \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{SendNews}(a = \bot), 1) = & \Pr(\mathit{TwoNews}, \Pr(\mathit{MyEmails}(a = \bot), 1)) \\
\Pr(\mathit{MyEmails}(a = \bot), 1) = & 1 - (1 - \Pr(\mathit{EmailDad}))(1 - \Pr(\mathit{Email}(a = \bot))) \\
 & \quad = 1 - (1 - 1)(1 - 0) = 1 \\
\Pr(\mathit{SendNews}(a = \bot), 1) = & \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{MyNews}, 1) = & \Pr(\mathit{TwoNews}, 1) * \Pr(\mathit{Addresses}, 1) \\
 & \quad + \Pr(\mathit{TwoNews}, 1) * (1 - \Pr(\mathit{Addresses}, 1)) \\
 & \quad = \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{TwoNews}, 1) = & 1 - (1 - \Pr(\mathit{CNN}, 1)) * (1 - \Pr(\mathit{BBC}, 1)) \\
 & \quad = 1 - (1/6) * (1/12) = 71/72 \\
\Pr(\mathit{MyNews}, 1) = & (71/72) \approx 0.986
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = & \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot))) \Pr(\mathit{Addresses}, 1) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot)))(1 - \Pr(\mathit{Addresses}, 1)) \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot))) = & \mathbb{E}(\mathsf{out}(\mathit{TwoNews})) * \mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a \neq \bot)) \\
\mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a \neq \bot) = & \mathbb{E}(\mathsf{out}(\mathit{EmailDad})) + \mathbb{E}(\mathsf{out}(\mathit{Email}(a \neq \bot))) = 2 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot))) = & \mathbb{E}(\mathsf{out}(\mathit{TwoNews})) * \mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a = \bot)) \\
\mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a \neq \bot) = & \mathbb{E}(\mathsf{out}(\mathit{EmailDad})) + \mathbb{E}(\mathsf{out}(\mathit{Email}(a = \bot))) = 1 \\
\mathbb{E}(\mathsf{out}(\mathit{TwoNews})) = & \mathbb{E}(\mathsf{out}(\mathit{CNN})) + \mathbb{E}(\mathsf{out}(\mathit{BBC})) \\
 & \quad = 5/6 + 11/12 = 7/4 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot))) = & 7/2 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot))) = & 7/4 \\
\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = & (7/2) * (399/400) + (7/4) * (1/400) = 5593/1600 \approx 3.496
\end{array}
$$

B.2.3 Typed Model

For the Typed case, let us consider the same profiles as for *All4All*. [2]

Sites	Global Profile A	Global Profile B
CNN	(1@1)	(0@1)
BBC	(1 @1/2 ∥ 0 @1/2)	(0@1)
AddressAlice	(1 @1)	(1 @1)
AddressBob	(1 @1)	(1 @1)
Email	(1@1)	(1@1)
EmailDad	(1@1)	(1@1)
Probability	**19/24**	**19/120**

[2] *EmailDad* is an alias of *Email* with a fixed address, therefore have the same probability, in this case 1.

Sites	Global Profile C	Global Profile D
CNN	(1@1)	(0@1)
BBC	(1 @1/2 $\parallel$ 0 @1/2)	(0@1)
AddressAlice	(0 @1)	(0 @1)
AddressBob	(0 @1)	(0 @1)
Email	(1@1)	(1@1)
EmailDad	(1@1)	(1@1)
Probability	**1/24**	**1/120**

Like for *All4All*, is clear that with profiles B, and D, *MyNews* does not publish anything as there are no news in those cases. Now, let us see what happens with the profile A. As *AddressAlice*, *AddressBob* and *Email* never fail in this case, then :

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{MyNews}) > 0) = & \Pr(\mathit{MyNews}, 1) \\
\Pr(\mathit{Addresses}, 1) = & 1 - (1 - \Pr(\mathit{AddressAlice}, 1))(1 - \Pr(\mathit{Bob}, 1)) \\
 & \quad +1 - (1 - 1)(1 - 1) = 1 \\
\Pr(\mathit{MyNews}, 1) = & \Pr(\mathit{SendNews}(a \neq \bot), 1) * \Pr(\mathit{Addresses}, 1) \\
 & \quad + \Pr(\mathit{SendNews}(a = \bot), 1) * (1 - \Pr(\mathit{Addresses}, 1)) \\
 & \quad = \Pr(\mathit{SendNews}(a \neq \bot), 1) + 0 \\
\Pr(\mathit{SendNews}(a \neq \bot), 1) = & \Pr(\mathit{TwoNews}, \Pr(\mathit{MyEmails}(a \neq \bot), 1)) \\
\Pr(\mathit{MyEmails}(a \neq \bot), 1) = & 1 - (1 - \Pr(\mathit{EmailDad}))(1 - \Pr(\mathit{Email}(a \neq \bot))) \\
 & \quad = 1 - (1 - 1)(1 - 1) = 1 \\
\Pr(\mathit{SendNews}(a \neq \bot), 1) = & \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{MyNews}, 1) = & \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{TwoNews}, 1) = & 1 - (1 - \Pr(\mathit{CNN}, 1)) * (1 - \Pr(\mathit{BBC}, 1)) \\
 & \quad = 1 - (1) * (1/2) = 1 \\
\Pr(\mathit{MyNews}, 1) = & 1
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = & \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot)))\Pr(\mathit{Addresses}, 1) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot)))(1 - \Pr(\mathit{Addresses}, 1)) \\
 & \quad = \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot)))\Pr(\mathit{Addresses}, 1) + 0 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot))) = & \mathbb{E}(\mathsf{out}(\mathit{TwoNews})) * \mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a \neq \bot)) \\
\mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a \neq \bot) = & \mathbb{E}(\mathsf{out}(\mathit{EmailDad})) + \mathbb{E}(\mathsf{out}(\mathit{Email}(a \neq \bot))) = 2 \\
\mathbb{E}(\mathsf{out}(\mathit{TwoNews})) = & \mathbb{E}(\mathsf{out}(\mathit{CNN})) + \mathbb{E}(\mathsf{out}(\mathit{BBC})) \\
 & \quad = 1 + 1/2 = 3/2 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot))) = & 3/2 * 2 = 3 \\
\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = & 3 * 1 = 3
\end{array}
$$

Now, in the profile C. As *AddressAlice* and *AddressBob* always fail in this case, then :

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{MyNews}) > 0) = & \Pr(\mathit{MyNews}, 1) \\
\Pr(\mathit{Addresses}, 1) = & 1 - (1 - \Pr(\mathit{AddressAlice}, 1))(1 - \Pr(\mathit{Bob}, 1)) \\
& \quad +1 - (1 - 0)(1 - 0) = 0 \\
\Pr(\mathit{MyNews}, 1) = & \Pr(\mathit{SendNews}(a \neq \bot), 1) * \Pr(\mathit{Addresses}, 1) \\
& \quad + \Pr(\mathit{SendNews}(a = \bot), 1) * (1 - \Pr(\mathit{Addresses}, 1)) \\
& \quad = 0 + \Pr(\mathit{SendNews}(a = \bot), 1) \\
\Pr(\mathit{SendNews}(a = \bot), 1) = & \Pr(\mathit{TwoNews}, \Pr(\mathit{MyEmails}(a = \bot), 1)) \\
\Pr(\mathit{MyEmails}(a = \bot), 1) = & 1 - (1 - \Pr(\mathit{EmailDad}))(1 - \Pr(\mathit{Email}(a = \bot))) \\
& \quad = 1 - (1 - 1)(1 - 0) = 1 \\
\Pr(\mathit{SendNews}(a = \bot), 1) = & \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{MyNews}, 1) = & \Pr(\mathit{TwoNews}, 1) \\
\Pr(\mathit{TwoNews}, 1) = & 1 - (1 - \Pr(\mathit{CNN}, 1)) * (1 - \Pr(\mathit{BBC}, 1)) \\
& \quad = 1 - (1) * (1/2) = 1 \\
\Pr(\mathit{MyNews}, 1) = & 1
\end{array}
$$

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = & \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot)))\Pr(\mathit{Addresses}, 1) \\
& \quad + \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot)))(1 - \Pr(\mathit{Addresses}, 1)) \\
& \quad = \mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot))) + 0 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a = \bot))) = & \mathbb{E}(\mathsf{out}(\mathit{TwoNews})) * \mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a = \bot)) \\
\mathbb{E}(\mathsf{out}(\mathit{MyEmails}(a = \bot) = & \mathbb{E}(\mathsf{out}(\mathit{EmailDad})) + \mathbb{E}(\mathsf{out}(\mathit{Email}(a = \bot))) = 1 \\
\mathbb{E}(\mathsf{out}(\mathit{TwoNews})) = & \mathbb{E}(\mathsf{out}(\mathit{CNN})) + \mathbb{E}(\mathsf{out}(\mathit{BBC})) \\
& \quad = 1 + 1/2 = 3/2 \\
\mathbb{E}(\mathsf{out}(\mathit{SendNews}(a \neq \bot))) = & 3/2 * 1 = 3/2 \\
\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = & 3/2 * 1 = 1.5
\end{array}
$$

As the profile A has probability $19/24$ and profile C probability $1/24$, we have that $\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = \mathbb{E}(\mathsf{out}(\mathit{MyNews})|A) * 19/24 + \mathbb{E}(\mathsf{out}(\mathit{MyNews})|C) * 1/24 \approx 2.4375$ and $\Pr(\mathsf{out}(\mathit{MyNews}) > 0) = \Pr(\mathit{MyNews}|A, 1) * 19/24 + \Pr(\mathit{MyNews}|C, 1) * 19/24 \approx 0.8333$.

B.3 *OneOrTwo*

$$
\begin{array}{ll}
\mathit{OneOrTwo} = & \mathit{Coin} > b > \\
& \quad (\mathit{If}(b) \gg \mathit{CNN} \mid \mathit{If}(\neg b) \gg (\mathit{CNN} \mid \mathit{BBC}))
\end{array}
$$

OneOrTwo is an IfOrc expression, therefore we can consider that all sites publish a boolean value or a signal (true). Site *Coin* publish *true* or false each with probability $1/2$. Let us analyse it under Oblivious and Stable models.

For simplification on the algorithm, let us rewrite it as

$$
\begin{array}{ll}
\mathit{CIFT} = & \mathit{If}(b) \gg \mathit{CNN} \\
\mathit{CIFF} = & \mathit{If}(\neg b) \gg (\mathit{CNN} \mid \mathit{BBC}) \\
\mathit{CIF} = & \mathit{CIFT} \mid \mathit{CIFF} \\
\mathit{OneOrTwo} = & \mathit{Coin} > b > \mathit{CIF}
\end{array}
$$

B.3.1 Oblivious Model

For the Oblivious model all sites fail in independent calls with certain probability and also can take different times to publish. The following table shows the probability of each delay, we take failure as delay ω:

Site	Delay
CNN	$500@1/2 \parallel 700@1/3 \parallel \omega@1/6$
BBC	$600@3/4 \parallel 700@1/6 \parallel \omega@1/12$

Let us compute first $\mathbb{E}(\mathsf{out})$ and $\Pr(\mathsf{out} > 0)$.

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{OneOrTwo})) = & \mathbb{E}(\mathsf{out}(\mathit{Coin}^0)) * \mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{Coin}^1)) * \mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) \\
\mathbb{E}(\mathsf{out}(\mathit{Coin}^0)) = & 1/2 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) = & \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=0)) + \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) \\
\mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) = & \mathit{If}(0=0) * \mathbb{E}(\mathsf{out}(\mathit{CNN})) = 1 * 5/6 = 5/6 \\
\mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=0)) = & \mathit{If}(0=1) * (\mathbb{E}(\mathsf{out}(\mathit{CNN})) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{BBC}))) = 0 * (5/6 + 11/12) = 0 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) = & 5/6 + 0 = 5/6 \\
\mathbb{E}(\mathsf{out}(\mathit{Coin}^1)) = & 1/2 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) = & \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) + \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=1)) \\
\mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=1)) = & \mathit{If}(1=0) * \mathbb{E}(\mathsf{out}(\mathit{CNN})) = 0 * 5/6 = 0 \\
\mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) = & \mathit{If}(1=1) * (\mathbb{E}(\mathsf{out}(\mathit{CNN})) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{BBC}))) = 1 * (5/6 + 11/12) = 7/4 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) = & 0 + 7/4 = 7/4 \\
E(\mathsf{out}(\mathit{OneOrTwo})) = & (1/2)(5/6) + (1/2)(7/4) = 31/24 \approx 1.292
\end{array}
$$

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{OneOrTwo}) > 0) = & \Pr(\mathit{OneOrTwo}, 1) \\
\Pr(\mathit{OneOrTwo}, 1) = & \Pr(\mathit{Coin}, \Pr(\mathit{CIF}(b=0), 1), \Pr(\mathit{CIF}(b=1), 1)) \\
\Pr(\mathit{CIF}(b=0), 1) = & 1 - (1 - \Pr(\mathit{CIFF}(b=0), 1))(1 - \Pr(\mathit{CIFT}(b=0), 1)) \\
\Pr(\mathit{CIFF}(b=0), 1) = & \mathit{If}(0=0) * \Pr(\mathit{CNN} = 1 * 5/6 = 5/6 \\
\Pr(\mathit{CIFT}(b=0), 1) = & \mathit{If}(0=1) * (1 - (1 - \Pr(\mathit{CNN})(1 - \Pr(\mathit{BBC})) \\
 & \quad = 0 * (71/72) = 0 \\
\Pr(\mathit{CIF}(b=0), 1) = & 5/6 \\
\Pr(\mathit{CIF}(b=1), 1) = & 1 - (1 - \Pr(\mathit{CIFF}(b=0), 1))(1 - \Pr(\mathit{CIFT}(b=0), 1)) \\
\Pr(\mathit{CIFF}(b=1), 1) = & \mathit{If}(1=0) * \Pr(\mathit{CNN} = 0 * 5/6 = 0 \\
\Pr(\mathit{CIFT}(b=1), 1) = & \mathit{If}(1=1) * (1 - (1 - \Pr(\mathit{CNN})(1 - \Pr(\mathit{BBC})) \\
 & \quad = 1 * (71/72) = 71/72 \\
\Pr(\mathit{CIF}(b=1), 1) = & 71/72 \\
\Pr(\mathit{Coin}, 5/6, 71/72) = & (1/2)(5/6) + (1/2)(71/72) = 131/144 \\
\Pr(\mathsf{out}(\mathit{OneOrTwo}) > 0) = & 131/144 \approx 0.910
\end{array}
$$

For $\mathbb{E}(\mathsf{first})$ and $\Pr(\mathsf{first} \leq k)$, let us compute the delay distribution of $\mathsf{first}(\mathit{OneOrTwo})$ in a fast way.

$$
\begin{aligned}
&\mathsf{first}(\mathit{OneOrTwo}, [0]) = \mathsf{first}(\mathit{Coin}, \mathsf{first}(\mathit{CIF}(b=0), [0]), \mathsf{first}(\mathit{CIF}(b=1), [0])) \\
&\mathsf{first}(\mathit{CIF}(b=0), [0]) = \mathsf{first}(\mathit{CIFF}(b=0), [0]) + \mathsf{first}(\mathit{CIFT}(b=0), [0]) \\
&\quad = \mathsf{first}(\mathit{CIFF}(b=0), [0]) = \mathsf{first}(\mathit{CNN}, [0]) \\
&\quad = (500@1/2 \parallel 700@1/3 \parallel \omega@1/6) \\
&\mathsf{first}(\mathit{CIF}(b=1), [0]) = \mathsf{first}(\mathit{CIFF}(b=1), [0]) + \mathsf{first}(\mathit{CIFT}(b=1), [0]) \\
&\quad = \mathsf{first}(\mathit{CIFT}(b=1), [0]) = \mathsf{first}(\mathit{CNN}, [0]) + \mathsf{first}(\mathit{BBC}, [0]) \\
&\quad = (500@1/2 \parallel 700@1/3 \parallel \omega@1/6) + (600@3/4 \parallel 700@1/6 \parallel \omega@1/12) \\
&\quad = (500@1/2 \parallel 600@3/8 \parallel 700@1/9 \parallel \omega@1/72) \\
&\mathsf{first}(\mathit{Coin}, (500@1/2 \parallel 700@1/3 \parallel \omega@1/6), \\
&\quad (500@1/2 \parallel 600@3/8 \parallel 700@1/9 \parallel \omega@1/72)) \\
&\quad = (500@1/2 \parallel 600@3/16 \parallel 700@2/9 \parallel \omega@13/144)) \\
&\mathsf{first}(\mathit{OneOrTwo}) = (500@1/2 \parallel 600@3/16 \parallel 700@2/9 \parallel \omega@13/144))
\end{aligned}
$$

Given this delay distribution, we can see that it holds $\Pr(\mathsf{out} > 0) = \Pr(\mathsf{first} \neq \omega)$. Now, from this distribution we have : $\mathbb{E}(\mathsf{first}(\mathit{OneOrTwo})) = \sum_{(\delta,p)\in\mathsf{first}(\mathit{OneOrTwo})} \delta * p / \Pr(\mathsf{out}(\mathit{OneOrTwo}) > 0) \approx 569.47$

B.3.2 Typed Model

For the Typed case, let us consider two profiles.

Sites	**Global Profile A**	**Global Profile B**
CNN	(1@1)	(0@1)
BBC	(1 @11/12 $\parallel$ 0 @1/12)	(1 @11/12 $\parallel$ 0 @1/12)
Probability	**5/6**	**1/6**

Let us see what happens with the profile A. In this case calls to *CNN* never fail in this case, then :

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{OneOrTwo})|A) = & \mathbb{E}(\mathsf{out}(\mathit{Coin}^0)) * \mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{Coin}^1)) * \mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) \\
\mathbb{E}(\mathsf{out}(\mathit{Coin}^0)) = & 1/2 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) = & \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=0)) + \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) \\
 & \quad \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) \\
\mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) = & \mathit{If}(0=0) * \mathbb{E}(\mathsf{out}(\mathit{CNN})) = 1 * 1 = 1 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) = & 1 \\
\mathbb{E}(\mathsf{out}(\mathit{Coin}^1)) = & 1/2 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) = & \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) + \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=1)) \\
 & \quad \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) \\
\mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) = & \mathit{If}(1=1) * (\mathbb{E}(\mathsf{out}(\mathit{CNN})) \\
 & \quad + \mathbb{E}(\mathsf{out}(\mathit{BBC}))) = 1 * (1 + 11/12) = 23/12 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) = & 23/12 \\
E(\mathsf{out}(\mathit{OneOrTwo})) = & (1/2)1 + (1/2)(23/12) = 35/24 \approx 1.458
\end{array}
$$

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{OneOrTwo}|A) > 0) = & \Pr(\mathit{OneOrTwo}, 1) \\
\Pr(\mathit{OneOrTwo}, 1) = & \Pr(\mathit{Coin}, \Pr(\mathit{CIF}(b=0), 1), \Pr(\mathit{CIF}(b=1), 1)) \\
\Pr(\mathit{CIF}(b=0), 1) = & 1 - (1 - \Pr(\mathit{CIFF}(b=0), 1))(1 - \Pr(\mathit{CIFT}(b=0), 1)) = \\
 & \quad \Pr(\mathit{CIFF}(b=0), 1) \\
\Pr(\mathit{CIFF}(b=0), 1) = & \mathit{If}(0=0) * \Pr(\mathit{CNN} = 1 * 1 = 1 \\
\Pr(\mathit{CIF}(b=0), 1) = & 1 \\
\Pr(\mathit{CIF}(b=1), 1) = & 1 - (1 - \Pr(\mathit{CIFF}(b=0), 1))(1 - \Pr(\mathit{CIFT}(b=0), 1)) = \\
 & \quad \Pr(\mathit{CIFT}(b=1), 1) \\
\Pr(\mathit{CIFT}(b=1), 1) = & \mathit{If}(1=1) * (1 - (1 - \Pr(\mathit{CNN})(1 - \Pr(\mathit{BBC})) \\
 & \quad = 1 * 1 = 1 \\
\Pr(\mathit{CIF}(b=1), 1) = & 1 \\
\Pr(\mathit{Coin}, 5/6, 71/72) = & (1/2)1 + (1/2)1 = 1 \\
\Pr(\mathsf{out}(\mathit{OneOrTwo}) > 0) = & 1
\end{array}
$$

In the other hand, for the profile B, all calls to CNN fail, then :

$$
\begin{array}{ll}
\mathbb{E}(\mathsf{out}(\mathit{OneOrTwo})|B) = & \mathbb{E}(\mathsf{out}(\mathit{Coin}^0)) * \mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) \\
& \quad + \mathbb{E}(\mathsf{out}(\mathit{Coin}^1)) * \mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) \\
\mathbb{E}(\mathsf{out}(\mathit{Coin}^0)) = & 1/2 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) = & \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=0)) + \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) \\
& \quad \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) \\
\mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=0)) = & \mathit{If}(0=0) * \mathbb{E}(\mathsf{out}(\mathit{CNN})) = 1 * 0 = 0 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=0))) = & 0 \\
\mathbb{E}(\mathsf{out}(\mathit{Coin}^1)) = & 1/2 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) = & \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) + \mathbb{E}(\mathsf{out}(\mathit{CIFF}(b=1)) \\
& \quad \mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) \\
\mathbb{E}(\mathsf{out}(\mathit{CIFT}(b=1)) = & \mathit{If}(1=1) * (\mathbb{E}(\mathsf{out}(\mathit{CNN})) \\
& \quad + \mathbb{E}(\mathsf{out}(\mathit{BBC}))) = 1 * (0 + 11/12) = 11/12 \\
\mathbb{E}(\mathsf{out}(\mathit{CIF}(b=1))) = & 11/12 \\
\mathbb{E}(\mathsf{out}(\mathit{OneOrTwo})) = & (1/2)0 + (1/2)(11/12) = 11/24 \approx 0.458
\end{array}
$$

$$
\begin{array}{ll}
\Pr(\mathsf{out}(\mathit{OneOrTwo}) > 0) = & \Pr(\mathit{OneOrTwo}, 1) \\
\Pr(\mathit{OneOrTwo}|B, 1) = & \Pr(\mathit{Coin}, \Pr(\mathit{CIF}(b=0), 1), \Pr(\mathit{CIF}(b=1), 1)) \\
\Pr(\mathit{CIF}(b=0), 1) = & 1 - (1 - \Pr(\mathit{CIFF}(b=0), 1))(1 - \Pr(\mathit{CIFT}(b=0), 1)) = \\
& \quad \Pr(\mathit{CIFF}(b=0), 1) \\
\Pr(\mathit{CIFF}(b=0), 1) = & \mathit{If}(0=0) * \Pr(\mathit{CNN} = 1 * 0 = 0 \\
\Pr(\mathit{CIF}(b=0), 1) = & 1 \\
\Pr(\mathit{CIF}(b=1), 1) = & 1 - (1 - \Pr(\mathit{CIFF}(b=0), 1))(1 - \Pr(\mathit{CIFT}(b=0), 1)) = \\
& \quad \Pr(\mathit{CIFT}(b=1), 1) \\
\Pr(\mathit{CIFT}(b=1), 1) = & \mathit{If}(1=1) * (1 - (1 - \Pr(\mathit{CNN})(1 - \Pr(\mathit{BBC})) \\
& \quad = 1 * 11/12 = 11/12 \\
\Pr(\mathit{CIF}(b=1), 1) = & 1/12 \\
\Pr(\mathit{Coin}, 5/6, 71/72) = & (1/2)0 + (1/2)(11/12) = 11/24 \\
\Pr(\mathsf{out}(\mathit{OneOrTwo}) > 0) = & 11/24 \approx 0.458
\end{array}
$$

As the profile A has probability $5/6$ and profile B probability $1/6$, we have that $\mathbb{E}(\mathsf{out}(\mathit{MyNews})) = \mathbb{E}(\mathsf{out}(\mathit{MyNews})|A) * 5/6 + \mathbb{E}(\mathsf{out}(\mathit{MyNews})|C) * 1/6 \approx 1.292$ and $\Pr(\mathsf{out}(\mathit{MyNews}) > 0) = \Pr(\mathit{MyNews}|A, 1) * 5/6 + \Pr(\mathit{MyNews}|C, 1) * 1/6 \approx 0.910$.

Observe that $\mathbb{E}(\mathsf{out}(\mathit{MyNews}))$ and $\Pr(\mathsf{out}(\mathit{MyNews}) > 0)$ are the same for the Oblivious and Stable Model. That is a special case that happens when all sites are called at most 1 time each during a evaluation and sites are not related.

B.4 *Cron*

Now lets see a real example with errors and randomness. For that we will see the backup service of one of my personal web sites. The service goal is to backup some of the new data and send a report by mail of the data and new comments.
The service use the following servers and functions:

- *Server*. Main server an access point to the web site database. Functions allowed:
  - *access*(). Query to the web server.
- *BackupServer*. Server hosting the backup page and access point to the backup database. Functions allowed:
  - *send*(x). Query to the web server with new data as input. Output the data into the database language.
- *DB*. Database of the main site holding the data and comments. Functions allowed:
  - *new*(). Query for the last 24 hours data. Output the data.
  - *day*(). Query for the last 24 hours comments. Output the comments.
- *BackupDB*. Database of the backup site holding the backup data. Functions allowed:
  - *save*(x). Quuery for save the data given as input. Output the data.
- *Mail*. Google e-mail server. Functions allowed:
  - *send*(y). Send message y by e-mail.

Given these sites, the following expression represents the service:

$$\begin{array}{lcl} Backup & = & DB.new > x > BackupServer.send(x) \\ & & \quad > d > BackupDB.save(d) \\ DBActions & = & DB.connect() \gg (Backup \mid DB.day()) \\ Cron & = & Server.access() \gg DBActions > y > Mail.send(y) \end{array}$$

The mail provider is Google Mail with reliability 99.99% and the other server are all from the same plan that ensures a 99% online time, but all in independent machines.

Now in each server we have some services available. For Web servers we have the access function that always return if the server is online, the same does the function *send* of Mail and *save* of the databases. In the other hand, functions *new* and *day* returns only if there are new entries or new comments. The service is called daily and by statistic we know that each day we have probability 0.2 to have a new entry and 0.8 of having at least a new comment.

Given that, the following table shows the global profiles for this service:

Sites	**Global Profile A**	**Global Profile B**
Server.access	(1@0.99 ‖ 0@0.01)	(1@0.99 ‖ 0@0.01)
BackupServer.send	(1@0.99 ‖ 0@0.01)	(1@0.99 ‖ 0@0.01)
DB.new	(1@0.2 ‖ 0@0.8)	(0@1)
DB.day	(1@0.8 ‖ 0@0.2)	(0@1)
BackupDB.save	(1@0.99 ‖ 0@0.01)	(1@0.99 ‖ 0@0.01)
Mail.send	(1@0.9999 ‖ 0@0.0001)	(1@0.9999 ‖ 0@0.0001)
Probability	**0.99**	**0.01**

Is clear that for the profile B there will be no value published during the execution, as the DB server will be off-line. Only profile A will publish.

Then, for profile A, we compute the probability of produce at least one value:

$$\begin{aligned}
\Pr(\mathsf{out}(\mathit{Cron}) > 0) &= \Pr(\mathit{Cron}, 1) \\
\Pr(\mathit{Cron}, 1) &= \Pr(\mathit{Server.access}() \gg \mathit{DBActions}, \\
&\qquad \Pr(\mathit{Mail.send}(y \neq \bot), 1)) \\
\Pr(\mathit{Mail.send}(y \neq \bot), 1) &= 0.9999 \\
\Pr(\mathit{DBActions}, 0.9999) &= 1 - (1 - \Pr(\mathit{Backup}, 0.9999)) * (1 - \Pr(\mathit{DB.day}, 0.9999)) \\
\Pr(\mathit{DB.day}, 0.9999) &= 0.8 * 0.9999 = 0,79992 \\
\Pr(\mathit{Backup}, 0.9999) &= 0.2 * 0.99 * 0.99 * 0.9999 \approx 0.196 \\
\Pr(\mathit{DBActions}, 0.9999) &= 1 - (1 - 0.79992) * (1 - 0.196) \approx 0.839 \\
\Pr(\mathsf{out}(\mathit{Cron}) > 0) &= \Pr(\mathit{Server.access}, 0.839) = 0.99 * 0.839 \approx 0.831
\end{aligned}$$

As the profile A has probability 0.99, then $\Pr(\mathsf{out}(\mathit{Cron}) > 0) \approx 0.822$. Given that, the service *Cron* can be used as a site following the halting profile $(\mathit{success}@0.822 \parallel \mathit{failure}@0.178)$.