Confluence in Probabilistic Rewriting

Alejandro Díaz-Caro\textsuperscript{a,2} Guido Martínez\textsuperscript{b}

\textsuperscript{a} Universidad Nacional de Quilmes \& CONICET. Bernal, Buenos Aires, Argentina  
\texttt{alejandro.diaz-caro@unq.edu.ar}

\textsuperscript{b} Universidad Nacional de Rosario \& CIFASIS-CONICET. Rosario, Santa Fe, Argentina  
\texttt{martinez@cifasis-conicet.gov.ar}

Abstract

Driven by the interest of reasoning about probabilistic programming languages, we set out to study a notion of unicity of normal forms for them. To provide a tractable proof method for it, we define a property of \textit{distribution confluence} which is shown to imply the desired unicity (even for infinite sequences of reduction) and further properties. We then carry over several criteria from the classical case, such as Newman’s lemma, to simplify proving confluence in concrete languages. Using these criteria, we obtain simple proofs of confluence for $\lambda_1$, an affine probabilistic $\lambda$-calculus, and for $Q^*$, a quantum programming language for which a related property has already been proven in the literature.

\textbf{Keywords:} abstract rewriting system, probabilistic rewriting, confluence

1 Introduction

In the formal study of programming languages, modelling execution via a small-step operational semantics is a popular choice. Such a semantics is given by an \textit{abstract rewriting system} (ARS) which, mathematically, is no more than a binary relation on abstract terms specifying whether one term can rewrite to another. This relation is not required to be a function, and can thus allow for a program to rewrite in two different ways. In such a case, it is important that the different execution paths for a given program reach the same final value (if any); thus guaranteeing that any two determinizations of the semantics (e.g. execution machines) assign the same meaning to every program.

This correctness property is expressible at the level of relations, and is known as \textit{unicity of normal forms} (UN): any two irreducible terms reachable from a common starting point must be equal. For non-trivial languages, such as the $\lambda$-calculus, it can be hard to prove this property directly. Fortunately, the property of \textit{confluence} can serve as a proof method for it since UN is a trivial corollary of it while its proof

\footnote{1 This is basically the work done during the second author’s Licenciatura thesis\cite{17}.}

\footnote{2 Partially supported by projects STICAmSud 16STIC05 FoQCoSS and PICT-PRH 2015-1208.}
tends to be more tractable. This was the approach followed by Church and Rosser in [6, Corollary 2], where UN and confluence were first proven for the \( \lambda \)-calculus in 1936. Nowadays, confluence is widely used to show the adequacy of operational semantics in many kinds of programming languages.

In the past decades there has been a growing interest in programming languages with probabilistic behaviours (e.g. [1,2,4,9,10,13,21,22]), which cannot be modelled as a mere relation between terms. Example features include a probabilistic choice operator [10] and quantum measurement [9]. In these settings, the same need of showing the correctness of the semantics is present.

At a first glance, it may seem as if neither UN nor confluence could ever hold in these cases, since the possible results of, say, rolling a die or measuring a qubit are irreconcilably distinct. The key observation is that, in a probabilistic language, a notion of equivalence of programs should be about distributions of values, and not about punctual values [16]. Indeed, a single program might evaluate to different values if it is run twice, but equality should certainly be reflexive. Thus, the expectation should be that the different reduction paths do not impact the final distribution of results. This is precisely the property we set out to study in this paper, developing an associated notion of confluence for it.

For a concrete example, take a hypothetical language for representing dice rolls, where \( \Box \) represents an unrolled die which can reduce to any element in \( \{1, \ldots, 6\} \) with equal probability. For a pair of dice, represented by \((\Box, \Box)\), we should be allowed to choose which die to roll first. Rolling the first die can result in any term in the set \( \{(i, \Box)\}_{i=1,6} \) with equal probability; and similarly for the second. When continuing the rolls, both branches will end up in the same uniform distribution. However, this could be not so: consider the term \((\lambda x.(x,x)) \Box\). If the die is rolled before the abstraction is applied, only pairs with equal components can be obtained, which is not the case if we apply the abstraction first, obtaining \((\Box, \Box)\), which can reduce to any pair of results. We shall later come back to a similar language and provide conditions to avoid this divergence.

In Section 2 we introduce the problem of unicity of distributions for probabilistic rewriting. In Section 3 we define a rewriting system over distributions giving rise to our notion of distribution confluence and prove its adequacy. In Section 4 we give some criteria for proving distribution confluence, simplifying the burden of proof for concrete languages. In Section 5 we extend our unicity of distributions result to terms which are only asymptotically terminating. In Section 6 we prove confluence for two concrete programming languages: a simple, affine, probabilistic calculus, dubbed \( \lambda_1 \) and the quantum lambda calculus \( Q^* \) [9]. Finally, in Section 7 we conclude, give some insights on future directions, and analyse some related work.

## 2 Probabilistic Rewriting

### 2.1 Preliminaries

We assume familiarity with the study of abstract rewriting. We adopt the terminology from [6] and call a sequence of expansions followed by a sequence of reductions...
a peak. When the order is reversed, such sequence is called a valley. The property of confluence can then be expressed as “there is a valley for every peak”. When an ARS \( A \) is confluent, we note it by \( A \models \text{CR} \), and similarly for UN and other properties. The relation \( R \mod E \), noted as \( R/E \), is defined for any equivalence relation \( E \) as \( E \cdot R \cdot E \) [15, 20].

We denote by \( \mathcal{L}(X) \) the type of finite lists with elements in \( X \), where \([[]] \), ‘:’ and ‘++’ denote the empty list, the list constructor, and list concatenation, respectively. We also use the notation \([a,b,c] \) for \( a:b:c:[] \).

We define \( \mathcal{D}(A) = \mathcal{L}(\mathbb{R}^+ \times A) \), used as an explicit representation of (finitely-supported) distributions. There is no further restriction on \( \mathcal{D}(A) \). In particular, any given element might appear more than once, as in \([(1/3, a), (1/2, b), (1/6, a)] \). On a node \((p,a)\) of the distribution, \( p \) is called the weight and \( a \) the element. The weight of a distribution is defined to be the sum of all weights of its nodes, and can be any positive real number. When we require a normalised distribution, we use the type \( \mathcal{D}_1(A) \), defined as the set of those \( d \in \mathcal{D}(A) \) with unit weight. We abbreviate the distribution \([(p_1,a_1), (p_2,a_2), \ldots, (p_n,a_n)] \) by \([(p_i,a_i)]_i \), where \( n \) should be clear from the context. We also write \( \alpha D \) for the distribution obtained by scaling every weight in \( D \) by \( \alpha \); that is, \( \alpha [(p_i,a_i)]_i = [(\alpha p_i,a_i)]_i \).

We often reason about equivalence of distributions. For that purpose, we define a relation ‘\( \sim \)’ as the congruence closure of the rules in Fig. 1 (i.e. the smallest relation satisfying the rules and such that \( D \sim D' \) implies \( E_1 + D + E_2 \sim E_1 + D' + E_2 \)). We distinguish some subsets of ‘\( \sim \)’ by limiting the rules that may be used. We note by \( S \) the congruence closure of \( \text{SPLIT} \), by \( (FJ) \) the congruence closure of both \( \text{FLIP} \) and \( \text{JOIN} \), and similarly for other subsets.

We call two distributions equivalent if they are related by the reflexive-transitive closure of \( \sim \), noted \( \approx \). Two distributions are equivalent, then, precisely when they assign the same total weight to every \( a \in A \), regardless of order and duplication.

Arguably, using such a definition of distributions with lists is cumbersome and less clear than using a more semantic definition. However, we feel this is outweighed by the degree of rigour attained in later proofs (especially as we found some of them to be quite error-prone). As a secondary benefit, most of our development should be straightforwardly mechanizable.

A downside of the choice of lists is that we can only represent finitely-supported distributions. This restriction is present in other works as well (e.g. [11, 23]) and it seems to not be severe for modelling programming languages.

2.2 Probabilistic Abstract Rewriting Systems (PARS)

To model probabilistic rewriting, we need to move away from a simple relation between terms as used in ARSes. We shall then relate elements to distributions...
of elements, which introduce probability. In favour of expressivity, we allow for a single element to be related to multiple distributions (or none)\(^3\).

**Definition 2.1** A probabilistic abstract rewriting system (PARS) is a pair \((A,\mapsto)\) where \(A\) is a set (called the “carrier”) and \(\mapsto\) a relation of type \(\mathcal{P}(A \times \mathcal{D}_1(A))\) (called the “pointwise evolution relation”).

It should be clear that every ARS is also a PARS by taking Dirac distributions (i.e. normalised distributions with a single element). We can provide a simple example of a PARS by extensionally listing \(\mapsto\), as is commonly done for ARSes.

**Example 2.2** Let \(A\) be the PARS given by

\[
\begin{align*}
a &\mapsto [(2/3, b), (1/3, c)] & a &\mapsto [(2/5, d), (3/5, e)] \\
b &\mapsto [(1/2, c), (1/2, d)] & c &\mapsto [(1, d)]
\end{align*}
\]

Here, \(a\) is the only non-deterministic element. We call \(d\) a terminal element since it has no successor distributions.

As more significant examples, in Section 6 we describe two probabilistic λ-calculi with operational semantics modellable by a PARS.

Execution in a PARS is a mixture of non-deterministic and probabilistic choices. The first kind, corresponding to the \(\mathcal{P}\) operator, occur when the machine chooses a successor distribution for the current element. The second kind, corresponding to the \(\mathcal{D}_1\) operator, is a random choice between the elements of the chosen successor distribution. To model such execution, we introduce the notion of computation tree.

**Definition 2.3** Given a PARS \((A,\mapsto)\), we define the set of its (finite) “computation trees” with root a (noted \(\mathcal{T}(a)\)) inductively by the following rules.

\[
a \in \mathcal{T}(a) \\
a \mapsto [(p_1, a_1), \ldots, (p_n, a_n)] \quad t_i \in \mathcal{T}(a_i) \\
\begin{align*}
[a; (p_1, t_1); \ldots; (p_n, t_n)] &\in \mathcal{T}(a) \\
&\xrightarrow{1/2} c \\
&\xrightarrow{1/2} d \\
&\xrightarrow{1/2} t_i \quad \cdots 
\end{align*}
\]

A graphical representation of an example tree for the PARS in Definition 2.2 is given in the right. We also sometimes consider infinite computation trees, by taking the coinductively defined set instead.

So, any tree \(T \in \mathcal{T}(a)\) represents one possible (uncertain) evolution of the system after starting on \(a\). There is no further assumption about trees: in particular, if an element \(a\) is expanded many times in a tree, different successor distributions may be used at each node\(^4\).

When all the leaves of a tree are terminal elements, we call the tree maximal (as there is no proper supertree of it). A computation tree naturally assigns to each of its leaves a probability, taken as the product of the \(p_i\) in the path from the

---

\(^3\) Note that limiting it to a single distribution would make it impossible to model the term \((□, □)\) without fixing a strategy.

\(^4\) In some works, such as [3, 4], it is assumed that execution is Markovian, meaning the choice of distribution depends only on the current state. We do not make such assumption and allow for a more general execution.
It is clear that collecting the leaves of a tree (along with their assigned probabilities) gives rise to a normalised list distribution. We call such distribution the support of $T$ and note it as $\text{supp}(T)$.

We can now state our property of interest.

**Definition 2.4 (UTD)** A PARS $A$ has “unique terminal distributions” when for every $a$ and $T_1, T_2 \in \mathcal{T}(a)$ maximal, we have $\text{supp}(T_1) \approx \text{supp}(T_2)$.

We said before that proving UN (for ARSes) directly is usually hard. Since PARSes subsume ARSes, the same difficulties arise for proving UTD directly. Therefore, we seek a property akin to confluence, providing a more tractable proof method.

One idea is to set up a rewriting over the supports of computation trees (expanding leaves as a reduction) and study its confluence. However, this is bound to be too rigid, as the behaviours of computation trees are not identified by equivalence of their support distributions, forbidding reasoning modulo equivalence. For example, a tree with two leaves $(\frac{1}{2}, a)$ can exhibit more terminal distributions than a tree with a single $(1, a)$ leaf. Therefore we would have to deem the following, morally confluent, system:

$$
\begin{align*}
    a &\rightarrow [(\frac{1}{2}, b), (\frac{1}{2}, c)] \\
    a &\rightarrow [(1, c)] \\
    b &\rightarrow [(1, c)]
\end{align*}
$$

as non-confluent, since $[(\frac{1}{2}, c), (\frac{1}{2}, c)] \neq [(1, c)]$.

In order to not reject cases like this, we could change the previous notion to allow for an equivalence when closing the diagram. In this case, it is hard to reason compositionally about the property.

### 3 Rewriting Distributions and Confluence

To arrive at a notion of confluence that avoids the previously mentioned drawbacks, we shall define a rewriting over distributions that is more liberal than that of computation trees (Definition 3.4).

**Definition 3.1** Given a PARS $A = (A, \rightarrow)$, we define the relation $\rightarrow_P$ (of type $\mathcal{P}(\mathcal{D}(A) \times \mathcal{D}(A))$) (called “parallel evolution”) by the rules:

$$
\begin{align*}
    a &\rightarrow A & \text{ds} \rightarrow_P \text{ds}' \\
    (p, a) : \text{ds} \rightarrow_P pA + + \text{ds}' & \text{ds} \rightarrow_P \text{ds}' \\
    (p, a) : \text{ds} \rightarrow_P (p, a) : \text{ds}' & \text{ds} \rightarrow_P \text{ds}' \\
\end{align*}
$$

Note that without using the first rule, this is just the identity relation on distributions. We note the subset of this relation where the first rule must be used at least once in a step as $\rightarrow_P^1$, and call it proper evolution. Note that $\rightarrow_P$ is enough to simulate computation trees in this system, since it can be used to rewrite between their supports in the following sense.

**Definition 3.2** We call a relation $\rightarrow$ “compositional” when, if $D_1 \rightarrow E_1$ and $D_2 \rightarrow E_2$, then $\alpha D_1 + + \beta D_2 \rightarrow \alpha E_1 + + \beta E_2$ for all $\alpha, \beta \in \mathbb{R}^+$.

**Lemma 3.3** If $T \in \mathcal{T}(a)$, then $[(1, a)] \rightarrow_P^* \text{supp}(T)$

**Proof.** First, note that $\rightarrow_P$ is compositional. The result then follows by induction.
on $T$, using compositionality.

We now define an ARS over distributions, combining both parallel evolution and equivalence steps. Our definition of confluence for a PARS $A$ is then simply the usual confluence of that relation.

**Definition 3.4** Given a PARS $A = (A, \rightarrow)$, we define an associated ARS $\text{Det}(A)$ (called the “determinisation” of $A$) over the set $\mathcal{D}(A)$ by the relation $\rightarrow = (\rightarrow_p \cup \approx)$.

**Definition 3.5 (Distribution confluence)** We say a PARS $A$ is “distribution confluent” (or simply “confluent”) when $\text{Det}(A)$ is confluent in the classical sense.

Note that reduction in $\text{Det}(A)$ is more liberal than the expansion of trees, since it allows for “partial” evolutions. Indeed, if $a \rightarrow D$, we have $[(1/2, a)] \rightarrow D + [(1/2, a)]$. Nevertheless, its confluence is adequate for proving UTD, as Lemma 3.7 shows.

**Lemma 3.6** If $D_1$ is terminal and $D_1 \rightarrow^* D_2$, then $D_1 \approx D_2$ and $D_2$ is terminal.

**Proof.** It is clear, from the definition of $\rightarrow_p$, that if $D_1$ is terminal and $D_1 \rightarrow_p D'$, then $D_1 = D'$ (that is, exactly equal). The result then follows by induction on the number of steps, and the transitivity and reflexivity of $\approx$. \hfill \ensuremath{\square}

**Lemma 3.7** If $A \models \text{CR}$, then $A \models \text{UTD}$.

**Proof.** Take $T_1, T_2 \in \mathcal{T}(a)$ maximal. We know from Lemma 3.3 that $\text{supp}(T_2) \equiv^* [(1, a)] \rightarrow^* \text{supp}(T_1)$. By confluence, there must exist $C$ such that $\text{supp}(T_2) \rightarrow^* C \equiv^* \text{supp}(T_1)$. Since $T_1, T_2$ are maximal, their supports are terminal. Then, from two applications of Lemma 3.6, we get that $\text{supp}(T_2) \approx C \approx \text{supp}(T_1)$, as needed. \hfill \ensuremath{\square}

Furthermore, beyond UTD, distribution confluence implies that diverging computations (with no terminal distribution) can also be joined. As a consequence of that, confluence gives a neat method of proving the consistency of the equational theory induced by $\rightarrow$, as long as two distinct terminal elements exist.

**Lemma 3.8** If $D_1, D_2$ are terminal distributions, then $D_1 \equiv^* D_2$ if and only if $D_1 \approx D_2$.

**Proof.** The way back is trivial, so we detail the way forward. From confluence (repeatedly), $D_1$ and $D_2$ must have a common reduct. The result then follows from Lemma 3.6. \hfill \ensuremath{\square}

So, if $a$ and $b$ are distinct terminal elements, we know that $\rightarrow$-convertibility is a consistent theory as $[(1, a)] \not\approx [(1, b)]$. Summarizing, in a confluent PARS, reasoning about equivalence of programs is simplified and there is a strong consistency guarantee about convertibility, much like in the classical case.

### 4 Proving confluence

#### 4.1 Introduction

In the previous section, we have introduced our definition of confluence and argued that it is correct and sufficient for studying programming languages. For the prop-
Díaz-Caro and Martínez

property to be useful in practice, it should be also amenable to be proven. In this section we provide several simplified criteria for this task, obtaining analogues to many of the usual methods for classical confluence.

Since distribution confluence is no more than the classical confluence of \( \rightarrow \), every existing classical criteria (such as the diamond property or Newman’s lemma) is valid in this setting. However, very few of those are useful. Indeed, Det(\( \mathcal{A} \)) is never strongly (or even weakly) normalising regardless of \( \mathcal{A} \), and therefore Newman’s lemma does not apply. With respect to the diamond property, consider a system with \( a \rightarrow D \), then the following reductions are possible:

\[
[(1, a)] \leftrightarrow [(1/2, a), (1/2, a)] \rightarrow 1/2D \leftrightarrow [(1/2, a)]
\]

and these two distributions cannot be joined in a single step (unless we make further assumptions on \( \mathcal{A} \)). Also, as evidenced by this example, we need to prove confluence for every distribution, not just for Dirac ones; and take into account equivalence steps as well.

Thus, a priori, it would seem as if distribution confluence is hard to prove. To relieve that, we shall prove various syntactic lemmas about the relation \( \rightarrow \), allowing us to decompose it into more manageable forms. We then show how we can limit our reasoning to Dirac distributions, ignore equivalence steps in the peaks and allow to use them freely in the valleys. Lastly, we carry over classical criteria for confluence into this setting, such as the aforementioned diamond property and Newman’s lemma.

4.2 Syntactic lemmas about the relation \( \rightarrow \)

Since both \( \rightarrow_p \) and \( \approx \) are reflexive, we have \((\rightarrow_p \cup \approx)^* = (\rightarrow_p / \approx)^*\). Thus, since confluence is a property over the reflexive-transitive closure of a relation, it suffices to study the confluence of \( \rightarrow_p / \approx \), where equivalence steps do not have a cost, but are pervasive.

Given the precise syntactic definition for both relations, we can prove by analysis on the reductions that any step of \( \rightarrow_p / \approx \) can be made by splitting first, then evolving, and then joining back elements, as Lemma 4.3 states. We first introduce the following notion of commutation.

**Definition 4.1 (Sequential commutation)** We say that a relation \( R \) “commutes over” \( S \) when \( S \cdot R \subseteq R \cdot S \), and note it as \( R \vdash S \). The property can be expressed by the diagram on the right.

\[
\begin{array}{c}
S \\
\downarrow \\
\downarrow \\
\downarrow \\
\downarrow \\
S \\
\end{array}
\]

A key property of sequential commutation is that if \( R \vdash S \), then \((R \cup S)^* = R^* \cdot S^*\). It is also preserved when taking the \( n \)-fold composition (i.e. “\( n \) steps”) or reflexive-transitive closures on each side. We now prove some commutations relating evolution and equivalence steps (the last one needs some “administrative” steps).

**Lemma 4.2** We have \( \rightarrow_p \vdash (FJ)^*; S \vdash (FJ)^* \) and \( \rightarrow_p \cdot S \subseteq S \cdot \rightarrow_p \cdot (FJ)^* \).

\[\text{Note that this is not the usual notion of commuting relations, defined as } S^{-1} \cdot R \subseteq R^{-1} \cdot S, \text{ which is a symmetric property and could be called “parallel”}.
\]
Proof. By induction on the shape of the reductions. \[\square\]

Lemma 4.3 The relations \(\rightarrow_P / \approx\) and \(S^* \cdot \rightarrow_P \cdot (FJ)^*\) coincide.

Proof. The backwards inclusion is trivial, so we detail only the forward direction. By making use of the second commutation in Lemma 4.2 we get that \(\approx = S^* \cdot (FJ)^*\). Thus, we need to show \(S^* \cdot (FJ)^* \cdot \rightarrow_P \cdot S^* \cdot (FJ)^* \subseteq S^* \cdot \rightarrow_P \cdot (FJ)^*\). The proof then proceeds by using the other two commutations to reorder the relations. \[\square\]

Furthermore, this equivalence extends to \(n\)-fold compositions.

Lemma 4.4 The relations \((\rightarrow_P / \approx)^n\) and \(S^* \cdot \rightarrow_P^n \cdot (FJ)^*\) coincide.

Proof. By induction on \(n\), and using the previous lemma and commutations. \[\square\]

4.3 Simplifying diagrams

With the previous decompositions, we can now prove a very generic result about diagram simplification with a specific root \(D\), which then easily generalizes to the whole system.

Definition 4.5 We say a pair of relations \((\gamma, \delta)\) “closes” another pair \((\alpha, \beta)\) “on \(a\)” if whenever \(b \leftarrow_\alpha a \rightarrow_\beta c\) then there exists \(d\) such that \(b \rightarrow_\gamma d \leftarrow_\delta c\). The diagram for the property can be seen on the right. When this occurs for all \(a\), we simply say “\((\gamma, \delta)\) closes \((\alpha, \beta)\)”. Note that \(\rightarrow\) is confluent precisely when \((\rightarrow^*, \rightarrow^*)\) closes \((\rightarrow^*, \rightarrow^*)\).

Definition 4.6 We call a relation \(\rightarrow\) “local”, when if \(\alpha D_1 \parallel \beta D_2 \rightarrow E\), then there exist \(E_1, E_2\) such that \(E = \alpha E_1 \parallel \beta E_2\) and \(D_i \rightarrow E_i\). (Note that \(\rightarrow_P\) and \(S\) are local).

Theorem 4.7 Let \(\alpha, \beta\) be local relations and \(\gamma, \delta\) compositional relations. If \((\gamma / \approx, \delta / \approx)\) closes \((\alpha, \beta)\) for the Dirac distributions of \(D\), then \((\gamma / \approx, \delta / \approx)\) closes \((S^* \cdot \alpha, S^* \cdot (FJ)^*)\) for \(D\).

Proof. We give a sketch of the proof, for a more explanatory development please refer to [17]. We need to close \((S^* \cdot \alpha, S^* \cdot (FJ)^*)\). First, note that closing \((S^* \cdot \alpha, S^* \cdot (FJ)^*)\) is enough since we can revert the \((FJ)^*\) steps with \((FS)^*\) steps. Now, since \(\alpha\) and \(\beta\) are local, \(S^* \cdot \alpha\) and \(S^* \cdot \beta\) are as well. Thus, we can limit ourselves to closing the Dirac distributions of \(D\), and combine the reductions since \(\gamma, \delta\) are compositional. We now need to close \((S^* \cdot \alpha, S^* \cdot \beta)\) when starting from some \([[(1, a)]\]. Note that the left (right) branch is then of the form \(p_1 D_1 \parallel \cdots \parallel p_n D_n\) \((q_1 E_1 \parallel \cdots \parallel q_m E_m)\), where \(a\) reduces via \(\alpha (\beta)\) to each \(D_i (E_j)\). We can apply our hypothesis to get a \(C_{i,j}\) closing each \(D_i, E_j\). By first splitting each branch appropriately, we can close them in \(p_1 q_1 C_{1,1} \parallel \cdots \parallel p_1 q_m C_{1,m} \parallel \cdots \parallel p_n q_m C_{n,m}\), thus we conclude. \[\square\]

From this theorem, we get as corollaries several simplified criteria for confluence, applicable at the level of a particular distribution or to the whole system.

Criterion 4.8 (Dirac confluence) If for every element \(a\) of \(D\) and distributions \(E, F\) such that \(E \rightarrow_P [(1, a)] \rightarrow_P^* F\) there is a \(C\) such that \(E \rightarrow^* C \leftarrow^* F\), then \(D\)
is confluent.

**Proof.** A corollary of Theorem 4.7, taking $\alpha = \beta = \gamma = \delta = \rightarrow^*_p$. □

**Criterion 4.9 (Semi-confluence)** If for every element $a$ of $D$ and distributions $E, F$ such that $a \mapsto E$ and $[(1, a)] \rightarrow^*_p F$ there is a $C$ such that $E \rightarrow^* C \leftarrow^* F$, then $D$ is semi-confluent for $\rightarrow_p / \approx$.

**Proof.** A corollary of Theorem 4.7, taking $\alpha = \rightarrow_p$ and $\beta = \gamma = \delta = \rightarrow^*_p$. □

**Criterion 4.10 (Diamond property)** If for every element $a$ of $D$ and distributions $E, F$ such that $E \leftrightarrow a \mapsto F$ there is a $C$ such that $E \rightarrow_p \approx C \leftarrow_p \approx F$, then $D$ has the diamond property for $\rightarrow_p / \approx$.

**Proof.** A corollary of Theorem 4.7, taking $\alpha = \beta = \gamma = \delta = \rightarrow_p$. □

Note that in all these criteria, we need not consider any equivalence in the peak, and can use them freely in the valley, both before and after evolving. Also, proving any of these criteria for every element $a$ entails the confluence of the system.

In the classical case, a common tool for proving confluence is switching to another rewriting relation with equal reflexive-transitive closure (and thus an equivalent confluence) but which might be easier to analyse. For distribution confluence, a similar switch is allowed, slightly simplified by Lemma 4.12.

**Definition 4.11** Given two PARS over the same carrier set $A$, with relations $\rightarrow_1$ and $\rightarrow_2$, if for every $a \rightarrow_1 D$, we have $[(1, a)] \rightarrow^*_2 D$ we say that $\rightarrow_1$ is simulated by $\rightarrow_2$.

**Lemma 4.12** If $\rightarrow_1$ is simulated by $\rightarrow_2$, then $\rightarrow_1 \subseteq \rightarrow^*_2$. Hence, if both relations simulate each other, we have $\rightarrow^*_1 = \rightarrow^*_2$, and their confluences are therefore equivalent.

**Proof.** The first part follows by case analysis on the reduction $\rightarrow_1$. The second part is then trivial. □

### 4.4 Newman’s lemma

Newman’s lemma [18] states that, for a strongly normalising system, local confluence and confluence are equivalent properties, yet we have remarked previously that $\rightarrow_p$ is never a strongly normalising relation. To get an analogue to Newman’s lemma, we thus provide a specialized notion of strong normalisation.

**Definition 4.13** A infinite sequence $D_i$ such that $D_1 \rightarrow D_2 \rightarrow D_3 \rightarrow \cdots$ is called an “infinite $\rightarrow$-chain” (of root $D_1$).

**Definition 4.14** We call a distribution $D$ “strongly normalising” when there is no infinite $\rightarrow^*_p$-chain of root $D$. We call a PARS strongly normalising when every distribution is strongly normalising.

There are indeed systems which do satisfy this requirement, and it is intuitively what one would expect. Now a probabilistic analogue to Newman’s lemma can be obtained, following a proof style very similar to that of [14].

---

6 Note that infinite ($\rightarrow^*_p / \approx$)-chains always exist because of partial evolution.
**Definition 4.15** We say that a distribution $D$ is “locally confluent” when $E \leftarrow_P^1 D \rightarrow_P^1 F$ implies that there exists $C$ such that $E \rightarrow_P^* C \leftarrow_P^* F$.

(Note that strong normalisation over Dirac distributions implies it for all distributions, and likewise for local confluence.)

**Lemma 4.16 (Newman’s)** If a PARS is locally confluent and strongly normalising, then it is confluent.

**Proof.**
We shall prove, by well-founded induction over $\rightarrow_P^1$, that every distribution is confluent. For a particular distribution, it suffices to show that any peak of proper evolutions can be closed by $\rightarrow_P^*$. Then, by Corollary 4.8 (and since $\rightarrow_P^* = \rightarrow_P^{1*}$), confluence follows. We want to close a diagram of shape $E \leftarrow_P^1 D \rightarrow_P^1 F$. If either of the branches is zero steps long, then we trivially conclude. If not, we can form the diagram on the right, completing the proof by local confluence and the induction hypotheses for $E'$ and $F'$. □

5 Limit distributions

In classical abstract rewriting, an element can either be non-normalising, weakly normalising or strongly normalising (corresponding to the situations where it will not, may, and will normalise, respectively). In probabilistic rewriting, the story is not as simple. Consider the following PARS, where $b$ is a terminal element.

$$a \mapsto [(1/2, a), (1/2, b)]$$

Is $a$ normalising? One could say “no” since, indeed, it does not have a finite maximal computation tree, as there is always some probability for the system to be in the non-terminal $a$ state. However, such a probability will be made arbitrarily small by taking sufficient steps, and the distribution $[(1, b)]$ is reached in the limit. In this case $a$ is called almost surely terminating [3]. Certainly, a desirable fact is that almost-surely-terminating elements have a unique final distribution. We will prove that distribution confluence guarantees such unicity.

We first introduce a notion of distance between mathematical distributions, i.e. normalised functions of type $A \rightarrow [0, 1]^7$. We note with $[D]$ the mathematical distribution obtained from the list distribution $D$ (with the expected definition). We also extend definitions over mathematical distributions to list distributions by applying $[\cdot]$ where appropriate.

**Definition 5.1** Given $D, E$ mathematical distributions, we define the distance between them as $d(D, E) = \sum_{a \in A} |D(a) - E(a)|$.

**Definition 5.2 (Limit of a sequence)** Given an infinite sequence of mathematical distributions $D_0, D_1, \ldots$ we say $L$ is a limit for the sequence if for every $\varepsilon > 0$, $\ldots$

---

7 We move away from list distributions to allow for infinitely supported distributions on the limit.
there exists \( N > 0 \) such that for any \( i \geq N \), \( d(D_i, L) < \varepsilon \).

Note that this distance is the \( L_1 \) distance and the definition of limit is the usual one for metric spaces. It is then well known that there is at most one limit for a given sequence. We are interested in limits composed of terminal elements, representing a distribution of values. For that, the following definition is useful.

**Definition 5.3** For a mathematical distribution \( D \), we define its “liveness” as the sum of weights for non-terminal elements. That is, \( \text{Liv}(D) = \sum_{a \in \text{dom}(-\rightarrow)} D(a) \)

Note that the liveness of a list distribution cannot increase by evolution, and that \( \text{Liv}(D) = 0 \) iff \( D \) is terminal. Moreover, since the normalised part of a distribution cannot evolve, liveness provides an upper bound on the possible distance to be attained by evolution, as the following lemma states.

**Lemma 5.4** If \( D \rightarrow^* E \), then \( d(D, E) \leq 2 \cdot \text{Liv}(D) \).

**Proof.** By Lemma 4.4 there exist \( D' \) and \( E' \) such that \( D \approx D' \rightarrow^*_p E' \approx E \). Because of the equivalences, it suffices to show the result for \( D' \) and \( E' \). Assume, without loss of generality, that \( D' = D_l + D_t \), where all elements of \( D_l \) are not terminal, and all those of \( D_t \) are. Since parallel evolution is local and terminal elements cannot evolve, we know that \( E' = E'' + D_t \) for some \( E'' \). Then, \( d(D', E') \) is simply \( d(D_l, E'') \). Note that \( \text{Liv}(D') \) is the weight of \( D_l \) and of \( E'' \). Since distance is bounded by total weight, it follows that it is at most \( 2 \cdot \text{Liv}(D') = 2 \cdot \text{Liv}(D) \). \( \square \)

Now, we can extend our notion of unicity of terminal distributions to limit distributions of terminal elements, accounting for an infinite sequence of reduction steps. We call this property *unicity of limit distributions* (ULD) and show it to be a consequence of distribution confluence.

**Lemma 5.5** If \( A \) is confluent, and a distribution \( D \) is the root of two infinite \( \rightarrow \)-chains \( E_i \) and \( F_j \) with respective limits \( E_\infty \) and \( F_\infty \) terminal distributions, then \( E_\infty = F_\infty \).

**Proof.**
Take \( \varepsilon > 0 \). By the definition of limit, we know there are \( i, j \) such that \( d(E_i, E_\infty) < \varepsilon/6 \) and \( d(F_j, F_\infty) < \varepsilon/6 \). Since \( E_\infty \) and \( F_\infty \) are terminal, \( \text{Liv}(E_i) \) and \( \text{Liv}(F_j) \) must be less than \( \varepsilon/6 \). The distributions \( E_i \) and \( F_j \) are reachable by a finite amount of \( \rightarrow \) steps, so by confluence there exists a distribution \( C \) such that \( E_i \rightarrow^* C \rightarrow^* F_j \). From Lemma 5.4, we get that \( d(E_i, C) < \varepsilon/3 \) and likewise for \( F_j \). From these four bounds and the triangle inequality we get that \( d(E_\infty, F_\infty) < \varepsilon \). Since this is the case for any positive \( \varepsilon \), this distance must be exactly 0, and therefore \( E_\infty = F_\infty \). \( \square \)
6 Case studies

6.1 A linear probabilistic λ-calculus: $\lambda_1$

In our introductory example, we used the term $(\lambda x. (x, x)) \Box$ as an example of a non-confluent computation. The tension apparently arises between “binding the result” (CBV) and “binding the computation” (CBN), which makes a difference in the probabilistic case when the binding is duplicated (as already pointed out in [10]). There seem to be three ingredients needed for this failure of confluence of a term $(\lambda x. M)N$: (1) $x$ appears free more than once in $M$ (2) $N$ has a non-Dirac terminal distribution (3) both call-by-name and call-by-value reductions are possible.

In this section we define a probabilistic λ-calculus, dubbed $\lambda_1$, that prevents the combination of these three features by providing two kinds of abstractions, one restricting duplication and one restricting evaluation order. We show $\lambda_1$ to be confluent (by a diamond property), giving evidence that little more than linearity of probabilistic arguments is required to achieve a confluent probabilistic programming language.

The calculus is heavily based on the one defined in [24]. The set of pre-terms is given by the following grammar

$$M, N ::= x \mid MN \mid \lambda x. M \mid !M \mid M \oplus_p N$$

where the main novelty is the probabilistic choice operator $\oplus_p$, for any real number $p$ in the open interval $(0, 1)$. Abstractions ($\lambda$) are affine, that is, in the scope of $\lambda x$, there can be at most one free occurrence of $x$. Non-linear abstractions ($\lambda!$) have no such restriction. Affinity is enforced by a well-formedness judgment, whose definition is straightforward and which we thus omit. We work only with well-formed pre-terms, which form the set of terms.

The operational semantics is provided as a PARS in Fig. 2. A non-linear abstraction can only $\beta$-reduce with an argument of the form $!N$. Such terms cannot reduce, and are called thunks. This effectively implies that non-linear abstractions follow a fixed strategy (which is, morally, CBV until the argument is reduced to a thunk and CBN afterwards)\(^9\).

Fig. 2. Full semantics for $\lambda_1$
To prove the diamond property for $\lambda_1$, we first need two substitution lemmas. When $D = [(p_i, a_i)]_i$, we write $D[M/x]$ for the distribution $[(p_i, a_i[M/x])]_i$. Similarly, $M[D/x]$ denotes $[(p_i, M[a_i/x])]_i$.

**Lemma 6.1** If $M \rightsquigarrow D$, then $M[N/x] \rightsquigarrow D[N/x]$.

**Proof.** By induction on $M \rightsquigarrow D$. \hfill $\square$

**Lemma 6.2** If $M \rightsquigarrow D$, and $x$ is linear in $N$, then $N[M/x] \rightsquigarrow N[D/x]$.

**Proof.** By induction on the well-formedness of $N$. \hfill $\square$

Lemmas 6.1 and 6.2 are analogous to both statements of [24, Lemma 3.1]. Armed with both, we can prove the following theorem, which implies the diamond property.

**Theorem 6.3** If $D \leftarrow M \rightsquigarrow E$ then there exist $C, C'$ such that $D \rightarrow_{\alpha}^p C$ and $E \rightarrow_{\alpha}^p C'$ with $C \approx C'$.

**Proof.** By induction on the shape of $M \rightsquigarrow D$ and $M \rightsquigarrow E$. \hfill $\square$

By Corollary 4.10 we conclude that the calculus $\lambda_1$ is confluent, and thus enjoys both UTD and ULD.

### 6.2 $Q^*$

The $Q^*$ calculus [9] is a quantum programming language with quantum measurement, an inherently probabilistic operation. Reduction occurs between configurations, which are terms coupled with a quantum state, and which we will not detail further. Its semantics does not fix a strategy and, as $\lambda_1$, is also based on [24]. Reduction steps are paired with a label indicating which type of reduction occurred (e.g. which qubit was measured). In $Q^*$, terms are linear and not affine: variables representing quantum data cannot be duplicated nor discarded, as per the no-cloning [26] and no-erasure [19] properties of quantum physics.

The authors prove a property called strong confluence which asserts that any two maximal (possibly infinite) computation trees with a common root have an equivalent support when restricted to normal forms, and, further, that any normal form appears in an equal amount of leaves on both trees.

To prove such property, the authors prove a crucial lemma called quasi-one-step confluence, which is morally a diamond property but with slightly different behaviours according to the reductions taken. The reductions are distinguished between two sets $\mathcal{N}$, $\mathcal{K}$ and those of the form $\text{meas}_\alpha$ (measurements). We will not describe these sets nor $Q^*$’s semantics (its full description can be found in [9]), and will only state the lemma about its reductions. The notation $C \rightarrow_{\alpha}^p D$ means “$C$ reduces to $D$ with probability $p$ via the label $\alpha$”; and $C \rightarrow_{\alpha}^{\mathcal{N}} D$ means $C \rightarrow_{\alpha}^p D$ for some $\alpha \in \mathcal{N}$ (idem $\mathcal{K}$).

**Lemma 6.4** (Quasi-one-step Confluence for $Q^*$ [8, Proposition 4])

Let $C, D, E$ be configurations and $C \rightarrow_{\alpha}^p D$, $C \rightarrow_{\alpha}^{\mathcal{N}} E$, then:

1. $C \rightarrow_{\alpha}^p D 
2. C \rightarrow_{\alpha}^{\mathcal{N}} E 
3. C \rightarrow_{\alpha}^p D 
4. C \rightarrow_{\alpha}^{\mathcal{N}} E 

at the same non-confluence.
(i) If $\alpha \in \mathcal{K}$ and $\beta \in \mathcal{K}$, then either $D = E$ or there is $F$ with $D \rightarrow_{\mathcal{K}}^1 F$ and $E \rightarrow_{\mathcal{K}}^1 F$.

(ii) If $\alpha \in \mathcal{K}$ and $\beta \in \mathcal{N}$, then either $D \rightarrow_{\mathcal{N}}^1 E$ or there is $F$ with $D \rightarrow_{\mathcal{N}}^1 F$ and $E \rightarrow_{\mathcal{N}}^1 F$.

(iii) If $\alpha \in \mathcal{K}$ and $\beta = \text{meas}$, then there is $F$ with $D \rightarrow_{\text{meas}}^s F$ and $E \rightarrow_{\mathcal{N}}^1 F$.

(iv) If $\alpha \in \mathcal{N}$ and $\beta \in \mathcal{N}$, then either $D = E$ or there is $F$ with $D \rightarrow_{\mathcal{N}}^1 F$ and $E \rightarrow_{\mathcal{N}}^1 F$.

(v) If $\alpha \in \mathcal{N}$ and $\beta = \text{meas}$, then there is $F$ with $D \rightarrow_{\text{meas}}^s F$ and $E \rightarrow_{\mathcal{N}}^1 F$.

(vi) If $\alpha = \text{meas}_r$ and $\beta = \text{meas}_q$ (with $r \neq q$), then there are $t, u \in [0, 1]$ and an $F$ such that $pt = su$, $D \rightarrow_{\text{meas}_q}^t F$ and $E \rightarrow_{\text{meas}_r}^u F$.

From this lemma, the fact that there are no infinite $\mathcal{K}$ sequences, and a “probabilistic strip lemma”, the authors prove strong confluence [9, Theorem 5.4].

For distribution confluence, a simpler proof can be obtained. After modelling $Q^*$ as a PARS (without labeled reductions, but sets of distributions instead) we can readily reinterpret Lemma 6.4 to prove the diamond property for it (by Corollary 4.10). From this result, distribution confluence follows, and therefore also unicity of both terminal and limit distributions. Notably, neither the normalisation requirement for $\mathcal{K}$ nor the “probabilistic strip lemma” are needed for this fact.

Our obtained distribution confluence is similar, but neither weaker nor stronger than strong confluence. It is not weaker as distribution confluence guarantees that divergences of computations without any normal form can be joined, which strong confluence does not. It is also not stronger as it implies nothing of limit distributions that are not terminal, while it follows from strong confluence that they must coincide in their normalized part. It also does not imply the equality between the amount of leaves on each tree.\(^\text{10}\).

7 Conclusions

We have studied the problem of showing that an operational semantics for a probabilistic language is not affected by the choice of strategy. For this purpose, we provided a definition of confluence for probabilistic systems by defining a classical relation over distributions. We showed our property of distribution confluence to be appropriate as, in particular, it implies a unicity of terminal distributions, both for finite and infinite reductions, and gives an equational consistency guarantee.

We believe this development demonstrates that distribution confluence provides a reasonable “sweet spot” for proving the correctness of probabilistic semantics, as it provides the expected guarantees about execution while allowing tractable proofs. Concretely, the provided proofs for $\lambda_1$ and $Q^*$ are in line with what one would expect for linear calculi.

The proof about $Q^*$ also partially answers the conjecture posed in [9, Section 8] (“any rewriting system enjoying properties like Proposition 4 [our Lemma 6.4]...\(^\text{10}\).
enjoys confluence in the same sense as the one used here”) positively. The answer is partial since distribution confluence is not strictly equivalent.

Looking ahead, there are several interesting directions to explore. First, a study of confluence dealing with terms (and not just abstract elements) should provide more insights applicable to concrete languages, and we expect concepts such as orthogonality to have probabilistic analogues. As a generalization, it seems possible to take distribution weights from any mathematical field and not only the positive reals: even if interpreting such systems is not obvious, it seems most of our results would hold. Finally, a quantitative notion of confluence could also be explored, where a distribution is considered confluent if any divergence of it can be joined “up to ε”; in particular, obtaining useful simplified criteria for said property seems difficult.

7.1 Related work

In [7], similar definitions of rewriting of distributions and confluence are introduced. A key difference is that, in that work, equivalent distributions are identified and there is no partial evolution. In particular, this implies that the relation is not compositional, introducing a very subtle error in Lemma 10 (which, basically, states that a diamond property on Dirac distributions implies that of the whole system). The error is not severe for their development, but highlights the non-triviality of the matter.

In [12], a notion of confluence is defined and proven for an extension of $\lambda_q$ [25] (a quantum $\lambda$-calculus) with measurements (thus introducing probabilistic behaviour). The proposed confluence is basically a confluence on computation trees, and the one we study in this paper is strictly weaker, yet sufficient for UTD and consistency.

In [9], already amply discussed, the notion of confluence introduced is a strong confluence over maximal trees (either finite or infinite) which is related, but neither weaker nor stronger than distribution confluence.

Finally, in [5], a property of confluence is defined and studied over probabilistic rewriting systems which do not contain any non-determinism (i.e. where $\rightarrow$ is a partial function). This is a very different notion of confluence, speaking about punctual final results instead of distributions (indeed, distribution confluence trivially holds as there is no non-determinism).

References

[1] S. Andowa, Process algebra with probabilistic choice. In J.-P. Katoen, editor, Formal Methods for Real-Time and Probabilistic Systems, volume 1601 of Lecture Notes in Computer Science, pages 111–129, 1999.

[2] J. Borgström, U. Dal Lago, A. D. Gordon, and M. Szymczak. A lambda-calculus foundation for universal probabilistic programming. In Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, pages 33–46, New York, NY, USA, 2016. ACM.

[3] O. Bournez and F. Garnier. Proving positive almost-sure termination. In J. Giesl, editor, Proceedings of the 16th International Conference on Term Rewriting and Applications (RTA’05), volume 3467 of Lecture Notes in Computer Science, pages 323–337. Springer-Verlag, 2005.

[4] O. Bournez and M. Hoyrup. Rewriting logic and probabilities. In R. Nieuwenhuis, editor, Rewriting Techniques and Applications, volume 2706 of Lecture Notes in Computer Science, pages 61–75, 2003.
[5] O. Bournez and C. Kirchner. Probabilistic Rewrite Strategies. Applications to ELAN. In S. Tison, editor, Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA’02), volume 2378 of Lecture Notes in Computer Science, pages 252–266. Springer-Verlag, 2002.

[6] A. Church and J. B. Rosser. Some properties of conversion. Transactions of the American Mathematical Society, 39(3):472–482, 1936.

[7] U. Dal Lago, C. Faggian, B. Valiron, and A. Yoshimizu. The geometry of parallelism: Classical, probabilistic, and quantum effects. In Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, pages 833–845, New York, NY, USA, 2017. ACM.

[8] U. Dal Lago, A. Masini, and M. Zorzi. Confluence results for a quantum lambda calculus with measurements. arXiv:0905.4567. Extended version of [9].

[9] U. Dal Lago, A. Masini, and M. Zorzi. Confluence results for a quantum lambda calculus with measurements. In B. Coecke, P. Panangaden, and P. Selinger, editors, Proceedings of the Joint 5th International Workshop on Quantum Physics and Logic and 4th Workshop on Developments in Computational Models (QPL/DCM’08), volume 270.1 of Electronic Notes in Theoretical Computer Science, pages 59–74. Elsevier, 2011.

[10] U. Dal Lago and M. Zorzi. Probabilistic operational semantics for the lambda calculus. RAIRO Theoretical Informatics and Applications, 46(3):413–450, 2012.

[11] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic \( \lambda \)-calculus and quantitative program analysis. Journal of Logic and Computation, 15(2):159–179, 2005.

[12] A. Díaz-Caro, P. Arrighi, M. Gadella, and J. Grattage. Measurements and confluence in quantum lambda calculi with explicit qubits. In B. Coecke, I. Mackie, P. Panangaden, and P. Selinger, editors, Proceedings of the 5th International Workshop on Quantum Physics and Logic (QPL’09), volume 270.2 of Electronic Notes in Theoretical Computer Science, pages 59–74. Elsevier, 2011.

[13] O. M. Herescu and C. Palamidessi. Probabilistic asynchronous \( \pi \)-calculus. In J. Tiuryn, editor, Foundations of Software Science and Computation Structures, volume 1784 of Lecture Notes in Computer Science, pages 146–160, 2000.

[14] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. Journal of the ACM, 27(4):797–821, 1980.

[15] J.-P. Jouannaud and H. Kirchner. Completion of a set of rules modulo a set of equations. In Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL ’84, pages 83–92. ACM, 1984.

[16] D. Kozen. Semantics of probabilistic programs. Journal of Computer and System Sciences, 22(3):328–350, 1981.

[17] G. Martínez. Confluencia en sistemas de reescritura probabilista. Master’s thesis, Universidad Nacional de Rosario, Argentina, Mar. 27, 2017. Available at https://dcc.fceia.unr.edu.ar/es/lcc/tesinas-grado/tesinas/confluencia-en-sistemas-de-reescritura-probabilista.

[18] M. H. A. Newman. On theories with a combinatorial definition of “equivalence”. Annals of Mathematics, 43(2):223–243, 1942.

[19] A. K. Pati and S. L. Braunstein. Impossibility of deleting an unknown quantum state. Nature, 404:164–165, 2000.

[20] G. E. Peterson and M. E. Stickel. Complete sets of reductions for some equational theories. Journal of the ACM, 28(2):233–264, 1981.

[21] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’02, pages 154–165, New York, NY, USA, 2002. ACM.

[22] D. Scott. Stochastic \( \lambda \)-calculus: An extended abstract. Journal of Applied Logic, 12(3):360–376, 2014.

[23] P. Selinger and B. Valiron. A lambda calculus for quantum computation with classical control. In P. Urzyczyn, editor, Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA’05), volume 3461 of Lecture Notes in Computer Science, pages 354–368. Springer-Verlag, 2005.

[24] A. Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In J. Giesl, editor, Proceedings of the 16th International Conference on Term Rewriting and Applications (RTA’05), volume 3467 of Lecture Notes in Computer Science, pages 219–234. Springer-Verlag, 2005.

[25] A. van Tonder. A lambda calculus for quantum computation. SIAM Journal on Computing, 33:1109–1135, 2004.

[26] W. K. Wootters and W. H. Zurek. A single quantum cannot be cloned. Nature, 299:802–803, 1982.