



HAL
open science

Higher Order Bayesian Networks, Exactly

Claudia Faggian, Pautasso Daniele, Vanoni Gabriele

► **To cite this version:**

Claudia Faggian, Pautasso Daniele, Vanoni Gabriele. Higher Order Bayesian Networks, Exactly. Proc. ACM Program. Lang, inPress, POPL 2024, 10.1145/363292 . hal-04337161

HAL Id: hal-04337161

<https://hal.science/hal-04337161>

Submitted on 12 Dec 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Higher Order Bayesian Networks, Exactly

Extended Version

CLAUDIA FAGGIAN, IRIF, CNRS, Université Paris Cité, France

DANIELE PAUTASSO, University of Turin, Italy

GABRIELE VANONI, IRIF, CNRS, Université Paris Cité, France

Bayesian networks are graphical *first-order* probabilistic models that allow for a compact representation of large probability distributions, and for efficient inference, both exact and approximate. We introduce a *higher-order* programming language—in the idealized form of a λ -calculus—which we prove *sound and complete* w.r.t. Bayesian networks: each Bayesian network can be encoded as a term, and conversely each (possibly higher-order and recursive) program of ground type *compiles* into a Bayesian network.

The language allows for the specification of recursive probability models and hierarchical structures. Moreover, we provide a *compositional* and *cost-aware* semantics which is based on factors, the standard mathematical tool used in Bayesian inference. Our results rely on advanced techniques rooted into linear logic, intersection types, rewriting theory, and Girard’s geometry of interaction, which are here combined in a novel way.

CCS Concepts: • **Theory of computation** → **Lambda calculus; Probabilistic computation; Linear logic; Type theory; Denotational semantics.**

Additional Key Words and Phrases: intersection types, bounds, Bayesian networks, probabilistic programming, programming languages, geometry of interaction

ACM Reference Format:

Claudia Faggian, Daniele Pautasso, and Gabriele Vanoni. 2024. Higher Order Bayesian Networks, Exactly: Extended Version. *Proc. ACM Program. Lang.*, 8, POPL, Article 84 (January 2024), 59 pages. <https://doi.org/10.1145/3632926>

1 INTRODUCTION

This paper is a foundational study, taking a cost-aware approach to the semantics of higher-order probabilistic programming languages. Probabilistic models play a crucial role in several fields such as machine learning, cognitive science, and applied statistics, with applications spanning from finance to biology. A prominent example of such models are Bayesian networks (BNs) [Pearl 1988], a (first-order, static) graphical formalism able to represent complex systems in a *compact* way and enabling *efficient* inference algorithms. BNs decompose large joint distributions into smaller *factors*. These are used in inference algorithms, both exact (such as message passing and variable elimination) and approximate (sampling-based). Despite their significant strengths, the task of modeling using Bayesian networks is comparable to the task of programming using logical circuits.

Authors’ addresses: [Claudia Faggian](#), IRIF, CNRS, Université Paris Cité, France, faggian@irif.fr; [Daniele Pautasso](#), University of Turin, Italy, daniele.pautasso@unito.it; [Gabriele Vanoni](#), IRIF, CNRS, Université Paris Cité, France, gabriele.vanoni@irif.fr.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

© 2024 Copyright held by the owner/author(s).

ACM 2475-1421/2024/1-ART84

<https://doi.org/10.1145/3632926>

Probabilistic Programming Languages. A different approach is taken by *probabilistic programming languages* (PPLs), where statistical models are specified as programs. The fundamental idea behind PPLs is to separate the model description—the program—from the computation of the probability distribution specified by the program—the inference task. This separation aims at making stochastic modeling as accessible as possible, hiding the underlying inference engines, which typically encompass various sampling methods such as importance sampling, Markov Chain Monte Carlo, and Gibbs sampling. In this paper, we specifically focus on *functional* PPLs, which allow for first-class higher-order functions and compositional semantics (e.g. Church [Goodman et al. 2008], Anglican [Wood et al. 2014] and Venture [Mansinghka et al. 2014]).

Bayesian Networks and PPLs. Bayesian networks and PPLs are closely interconnected. On the one hand, Bayesian networks can be easily represented as simple, first-order, probabilistic programs [Gilks et al. 1994; Koller et al. 1997]. On the other hand, several *first-order* PPLs (such as BUGS, a widely used declarative language, and Infer.NET) compile programs into a graphical model, specifically a Bayesian network, which is then utilized for performing inference tasks. For a detailed tutorial and an analysis of the involved subtleties, we refer to [van de Meent et al. 2018] (Ch.3).

Towards a Foundational Understanding. The research community is devoting considerable effort to establish a solid foundational understanding of functional PPLs. A central concept is compositionality, which is the key to reason in a modular (and scalable) way about programs. Such an understanding is crucial for the development of robust formal methods to facilitate the analysis of probabilistic programs, the construction of Bayesian models, and the verification of inference correctness. Pioneering works by Jacobs and Zanasi [2016, 2020] have paved the way for a logical and semantical comprehension of Bayesian networks and inference from a categorical perspective. The majority of foundational papers, e.g. [Dahlqvist et al. 2018; Heunen et al. 2017; Ścibior et al. 2018; Stein and Staton 2021; Vákár et al. 2019], have adopted an approach based on category theory, which has yielded remarkable insights, enabling denotational proofs of correctness and compositionality principles. This line of research however does not take into account the *raison d'être* of Bayesian networks, namely the space and time efficiency of inference. In the literature, compositional semantics and efficiency are typically explored as *separate* entities. This dichotomy stands in stark contrast to Bayesian networks, where the representation, its semantics (the defined joint distribution), and the inference algorithms are deeply *intertwined*.

This paper. Our foundational investigation adopts a cost-aware perspective. We introduce a semantical framework that integrates the *efficiency* of Bayesian networks with the *expressiveness* of higher-order functional programming, and the *compositional* nature of type systems. We adopt an idealized functional PPL, namely an untyped λ -calculus enriched with probabilistic primitives. Our language *faithfully* encompasses conventional Bayesian networks (expressed here by first-order terms in normal form), and comes equipped with a semantics and formal methods which are *resources-sensitive*. The core of our approach lies in semantical techniques, including rewriting theory and type systems. These form the groundwork for a compilation scheme that translates higher-order terms into Bayesian networks, which serve as a low-level language. We list below our main contributions:

- *A higher-order language for BNs.* We introduce a probabilistic call-by-push-value λ -calculus that we prove *sound and complete for BNs*: not only any Bayesian network can be encoded as a term (which is standard), but also—conversely—any higher-order (possibly recursive) program of ground type will eventually reduce to a first-order normal form, corresponding to a Bayesian

network (Thm. 4.6 and Thm. 7.1). Our language also supports the encoding of advanced stochastic models, including *template* Bayesian networks, and the specification of *recursive* probability models. Notably, the operational semantics we define corresponds to the process of unrolling the template into an actual BN, in line with the intended semantics of the templates.

- *A factor-based semantics.* We endow each term of ground type with a *factor*-based semantics. Factors, indeed, are the mathematical structure which give semantics to BNs. Technically, computing this factor-based semantics requires tracking the generation and sharing of random variables—this task is achieved using non-idempotent intersection types. This technique allows us to extract a Bayesian network \mathcal{B}_t from the type derivation of a ground term t , and to prove that the semantics (in the sense of Bayesian networks theory) of \mathcal{B}_t coincides with the semantics of t .
- *The factor semantics is compositional.* The main technical achievement of the paper is to establish the compositionality of the factor semantics for typed terms (Sect. 6 and Sect. 7.2). This is particularly noteworthy since operations on factors do not exhibit this property, in general. The design of the type system plays a crucial role in ensuring compositionality, thereby enabling the modular reasoning that one expects in a high-level programming language.
- *The factor semantics is resource-sensitive.* Our semantics takes resource consumption into account—its computational complexity (both in terms of space and time) is similar to that for Bayesian networks. Additionally, the type system offers a precise estimate of the cost associated with computing the semantics of a term (i.e., the cost of calculating the *exact* distribution defined by the term).
- *Proof techniques.* The proof of our results incorporates sophisticated techniques that have their foundation in linear logic, leveraging significant advancements made over the past 15 years. Specifically, we employ intersection types, rewriting theory, linear logic, and concepts inspired by Girard’s geometry of interaction in a novel and synergistic manner.

Proofs and more examples are in the Appendix.

2 ABOUT BAYESIAN NETWORKS, SHARING, AND THE λ -CALCULUS, INFORMALLY

Bayesian Reasoning. In the Bayesian interpretation, probabilities describe *degrees of belief* in events, and inference allows for reasoning under uncertainty. One challenge in Bayesian reasoning is how to represent joint probability distributions. Indeed, these can quickly become very large: in general, a probability distribution over n boolean variables requires storing 2^n values. Bayesian networks are able to express a joint probability distribution over several variables in a compact way (*factorized representation*), allowing for *efficient inference*, without ever needing to reconstruct the full joint distribution.

An Example of Bayesian Network. Let us start with an informal example. We want to model the fact that the lawn being Wet in the morning may depend on either Rain or the Sprinkler being on. In turn, both Rain and the regulation of the Sprinkler depend on being or not in the Dry Season. The dependencies between these four variables (shortened to D, S, R, W) are represented as arrows in Fig. 1, while (for each variable) the strength of the dependencies is quantified by a *conditional probability table* (CPT). Assume we wonder: did it rain last night? Given that we are in DrySeason, our *prior* belief is that Rain happens with probability 0.2. However, if we observe that the lawn is Wet, our confidence increases. The updated belief is called *posterior*. The model in Fig. 1 allows us to infer the posterior probability of Rain, given the *evidence* that the lawn is Wet, i.e. $\Pr(R = t | W = t)$. Posteriors are typical queries which can be answered by *Bayesian inference*,

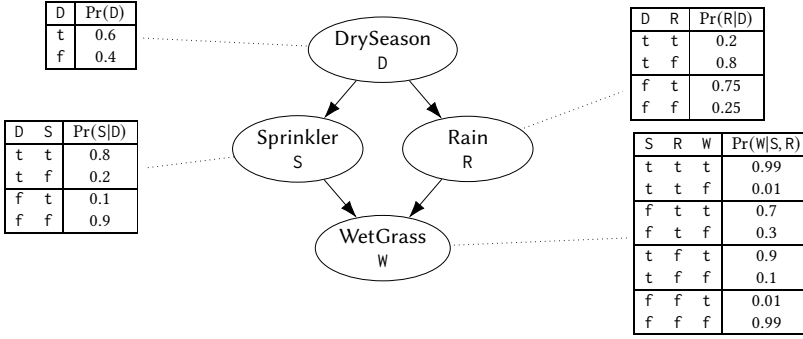


Fig. 1. An example of Bayesian network (from [Darwiche 2009]).

Joint distribution over D, S, R, W					Marginal over R, W		
D	S	R	W	Pr(d, s, r, w)	R	W	Pr(r, w)
t	t	t	t	0.09504	t	t	0.33
t	f	t	t	0.0168	t	f	0.09
f	t	t	t	0.0297
f	f	t	t	0.189			
t	t	t	f	0.00096			
t	f	t	f	0.0072			
f	t	t	f	0.0003			
f	f	t	f	0.081			
...			

⇒
summing out D, S

$$\Pr(r, w) = \sum_{d,s \in \{t,f\}} \Pr(d, s, r, w)$$

Fig. 2. Joint distribution corresponding to the Bayesian network in Fig. 1, and marginalization.

which has at its core Bayes conditioning, often condensed in the following informal formula:

$$\text{Posterior} = \text{Prior} \times \text{Likelihood} \div \text{Evidence}$$

Concretely, in our example:

$$\Pr(\text{Rain} = t | \text{Wet} = t) = \frac{\Pr(\text{Wet} = t | \text{Rain} = t) \Pr(\text{Rain} = t)}{\Pr(\text{Wet} = t)} = \frac{\Pr(\text{Rain} = t, \text{Wet} = t)}{\Pr(\text{Wet} = t)}$$

So, to compute the posterior $\Pr(\text{Rain} = t | \text{Wet} = t)$, we have to compute the *marginal* $\Pr(\text{Rain} = t, \text{Wet} = t)$, which can be obtained by summing out the other variables from the joint probability. *Marginalization* is illustrated in Fig. 2 (all rows which agree on the value of Rain and Wet are merged into a single row, summing up the probabilities). We remark that the joint distribution over D, S, R, W has 2^4 entries, although here we only display a subset of them.

The marginal probability $\Pr(\text{Wet} = t)$ of the evidence is computed in a similar way (yielding 0.69). Then, we obtain the posterior by normalizing: $\Pr(\text{Rain} = t | \text{Wet} = t) = 0.33/0.69 = 0.48$. Further evidence (for example, the Sprinkler is broken) would once again update our belief. In practice, the numerator of Bayes theorem (the unnormalized marginal) often suffices, since the actual posterior is proportional to it:

$$\text{Posterior} \propto \text{Prior} \times \text{Likelihood}$$

Summing up, the key step in *exact inference* is the computation of marginals.

Bayesian Networks as Terms. In probabilistic programming, it is standard to describe a Bayesian network with a term (see e.g. [Gordon et al. 2014] for a brief tutorial). We can encode our initial example into a rather standard probabilistic call-by-value λ -calculus, as follows:

```

let dry      = bernoulli0.6 in
let rain    = case ⟨dry⟩ of {t⇒bernoulli0.8; f⇒bernoulli0.1} in
let sprinkler = case ⟨dry⟩ of {t⇒bernoulli0.2; f⇒bernoulli0.75} in
let wet     = case ⟨rain, sprinkler⟩ of {⟨t, t⟩⇒bernoulli0.99; ⟨t, f⟩⇒bernoulli0.7;
                                         ⟨f, t⟩⇒bernoulli0.9; ⟨f, f⟩⇒bernoulli0.01}
in wet

```

(1)

The idea is that any CPT can be encoded with a case construct, together with a probabilistic primitive sample_d , which returns a value sampled from a (countably supported) probability distribution d . In this example we sample from a Bernoulli distribution. This way, all Bayesian networks can be encoded in a very basic fragment of the simply typed call-by-value λ -calculus (this will be discussed in the next sections). What if we allow for a richer, non linear, λ -calculus?

Beyond Ground Bayesian Networks. Standard Bayesian networks describe probabilistic models in an intuitive and compact way, but have some inherent limitations. They are a first-order model, lacking *modularity and compositionality*. A well-established way to increase the expressive power of Bayesian networks are *templates*¹, which allow for the description of *hierarchical* models and for taking into account *temporality*. Dynamic Bayesian networks [Dean and Kanazawa 1989] (in Fig. 4) are an instance extensively used in real-world applications—e.g. mobile robotics. A more structured approach is indeed essential when models become large and complex. While standard Bayesian networks have a precise mathematical definition, rooted in graph theory and statistics, templates are a more informal notion. We show that techniques from functional programming language theory can provide a neat and mathematically sound framework, founded on the theory of lambda calculus, exactly matching the intended semantics of templates.

Repeated Coin Tosses. Let us consider the following experiment, where repetition is involved.

1. Sample a bias r_i from a discrete distribution (for simplicity, let us assume there are only two possible choices: r_1 or r_2).
2. Toss m times a coin of bias r_i .
3. Return the results of the m (biased) coin tosses.

It is standard to graphically describe such an experiment by means of the *plate* notation [Buntine 1994; Gilks et al. 1994] (see Fig. 3), a graphical meta-formalism for representing models with *repeated structures* and *shared parameters*. A rectangular plate grouping random variables indicates multiple copies of the sub-graph. A number (m) is drawn to represent the number of repetitions. Unrolling the plate m times defines a ground Bayesian network. Please notice that the intended Bayesian network is the unrolled one. In Fig. 3, we show the template which models our experiment (a), and the ground Bayesian network resulting from its unrolling (b).

Repetitions, Sharing, and PPL. How can we describe this experiment as a λ -term? Let us assume $m = 2$. It is tempting to encode the template in the following way:

```

let bias = sampled in
let coin = case ⟨bias⟩ of {r1⇒bernoullir1; r2⇒bernoullir2}
in ⟨coin, coin⟩

```

¹We refer to [Koller and Friedman 2009], Ch. 6, for a detailed presentation and pointers to the vast literature.

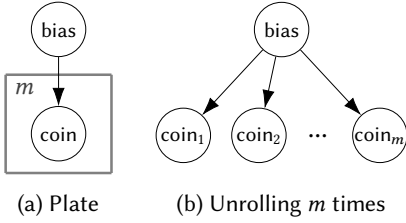
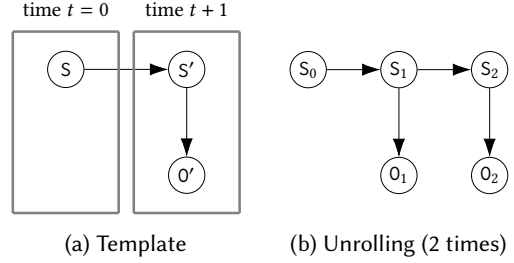
Fig. 3. Modeling m coin tosses.

Fig. 4. Discrete Time Dynamic Bayesian network.

<pre>let bias = sample_d in let coin = !(c⟨bias⟩) in let y_1 = der coin in let y_2 = der coin in ⟨y_1, y_2⟩</pre>	→	<pre>let bias = sample_d in let y_1 = der !(c⟨bias⟩) in let y_2 = der !(c⟨bias⟩) in ⟨y_1, y_2⟩</pre>	→*	<pre>let bias = sample_d in let y_1 = c⟨bias⟩ in let y_2 = c⟨bias⟩ in ⟨y_1, y_2⟩</pre>
---	---	--	----	--

Fig. 5. The λ -term correctly modeling two coin tosses, and its reduction to normal form. $\mathbf{c}\langle\text{bias}\rangle$ stands for the conditional expression case $\langle\text{bias}\rangle$ of $\{r_1 = \text{bernoulli}_{r_1}; r_2 = \text{bernoulli}_{r_2}\}$. The result stored in bias is correctly shared, while coin is copied before performing the toss, thus giving two independent and identically distributed values to y_1 and y_2 .

Unfortunately, the call-by-value policy makes sure that all the instances of coin have the *same* shared value, so the possible outcomes are only $\langle t, t \rangle$ and $\langle f, f \rangle$. Indeed, since only values can be substituted for variables, all expressions, even the probabilistic ones, have to be evaluated before being substituted. Switching the evaluation order to call-by-name does not solve the problem: now *all* the probabilistic primitives are copied *before* being evaluated. This means that all coin tosses are independent, this way not belonging necessarily to the same coin: some could have bias r_1 and some others r_2 . We need a finer evaluation mechanism that allows the programmer to say when values have to be shared, and when instead we want to actually duplicate unevaluated expressions.

Call-by-Push-Value. More than 20 years ago Levy [1999] introduced call-by-push-value as a subsuming paradigm and functional/imperative synthesis, refining the computational λ -calculus by Moggi [1989]. The slogan was: “a value *is*, a computation *does*”, as this language tries to unify call-by-name and call-by-value, providing two new primitives. The former *thinks* a computation inside a value, and the latter *forces* the evaluation of a value as a computation. Similar ideas have been independently developed in the linear logic community, using Girard’s translations. There, linear λ -calculi use the $!$ to thunk, and der (eliction) to force [Benton and Wadler 1996; Egger et al. 2014; Ehrhard 2016; Melliès and Tabareau 2010; Simpson 2005]. Having all of this in mind, we encode our experiment as the leftmost term of Fig. 5.

Operational Semantics. The simplest operational semantics for probabilistic programs is in terms of *sampled* values. A standard approach in probabilistic λ -calculi (including [Ehrhard and Tasson 2019]) is to give the operational semantics via Markov chains: sequential evaluation produces distributions over *execution paths*. Here, we follow a different route, because we want to model the unrolling of a higher-order term into a ground Bayesian Network, as shown in Fig. 5. By firing all the redexes but the probabilistic choices, the term t reduces to a *normal form* that represents a standard, ground Bayesian network, exactly matching the unrolling of the template in Fig. 3.

3 PRELIMINARIES ON CALCULUS AND TYPES

This section presents the probabilistic programming language we are going to use throughout this paper. The language includes constructs for describing *sampling* and *conditioning*. We have already argued why we opted for a language that is able to thunk computations and force values.

3.1 Syntax

Our language, dubbed λ_1 -calculus, is a fragment of [Ehrhard and Tasson 2019] probabilistic call-by-push-value. For ease of presentation, in this paper we limit *ground types* to booleans. This way, all random variables are assumed binary, and as a consequence, we only sample from Bernoulli distributions. Generalizing the language to *discrete* r.v.s is straightforward.

Terms. Let \mathcal{V} be a countable set of variables. λ_1 -terms are defined by the following grammar:

TERMS	t, u	::=	$v \mid \text{sample}_d \mid \text{case } v \text{ of } \{v_i \Rightarrow \text{sample}_{d_i}\} \mid \text{obs}(x = b)$
			$\text{let } x = u \text{ in } t \mid \text{letp } \langle x, y \rangle = v \text{ in } t \mid tv \mid \lambda x.t \mid \text{der } v$
VALUES	v, w	::=	$x \in \mathcal{V} \mid \langle v, w \rangle \mid !t \mid b$
BOOLEANS	b	::=	$\text{t} \mid \text{f}$

Following [Ehrhard 2016; Ehrhard and Tasson 2019], we use Linear Logic inspired notations: $!t$ corresponds to $\text{thunk}(t)$ and $\text{der } t$ to $\text{force}(t)$. The probabilistic primitive sample_d samples a boolean value from a (Bernoulli) distribution d . The case construct is just a generalized if/then/else—please notice that the case expression is restricted, because we reserve it to the encoding of CPT's, as we have informally described in Sect. 2. Observed data (the *evidence*) are specified syntactically using an *observe* construct, written obs —for example $\text{obs}(\text{wet} = \text{t})$; we will give several examples of its use in Sect. 8.

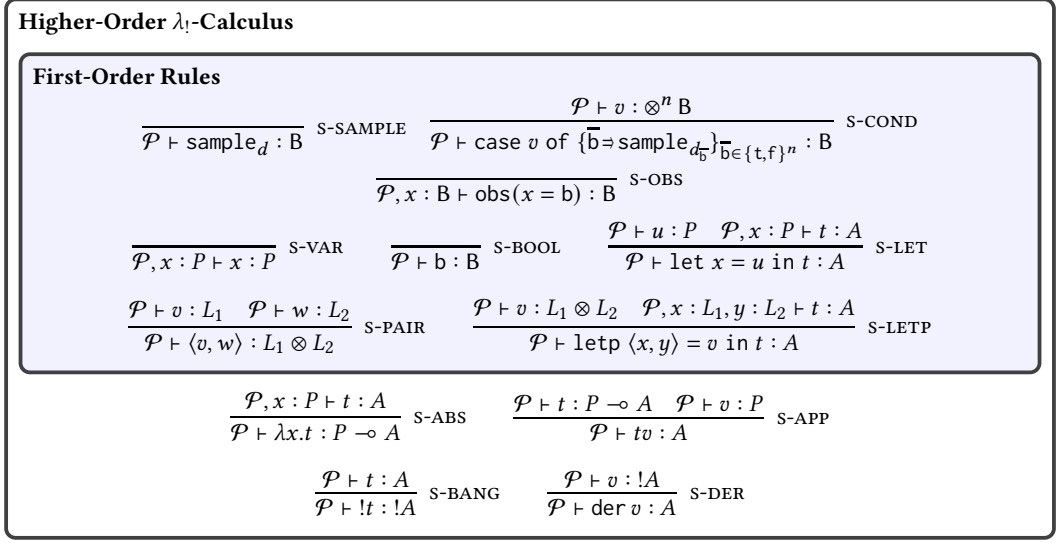
Free and bound variables are defined as usual: $\lambda x.t$ binds x in t , and the same for let and letp . A term is *closed* when there are no free occurrences of variables in it. Terms are considered modulo α -equivalence, and capture-avoiding (meta-level) substitution of all the free occurrences of x for u in t is noted $t\{x \leftarrow u\}$.

Syntactic Sugar. The grammar of the calculus is rather restricted, reminiscent of A-normal forms (and similarly to [Levy 1999]). It is standard to recover general constructs as follows:

tu	\triangleq	$\text{let } z = u \text{ in } tz$
$\langle u_1, u_2 \rangle$	\triangleq	$\text{let } z_1 = u_1 \text{ in let } z_2 = u_2 \text{ in } \langle z_1, z_2 \rangle$
$\text{der } u$	\triangleq	$\text{let } z = u \text{ in der } z$
$\text{letp } \langle y_1, y_2 \rangle = u \text{ in } t$	\triangleq	$\text{let } z = u \text{ in letp } \langle y_1, y_2 \rangle = z \text{ in } t$
$\text{case } u \text{ of } \{v_i \Rightarrow t_i\}$	\triangleq	$\text{let } z = u \text{ in case } z \text{ of } \{v_i \Rightarrow t_i\}$
$\text{obs}(u = b)$	\triangleq	$\text{let } z = u \text{ in obs}(z = b)$

Notation 3.1. We often write $\langle v_1, \dots, v_n \rangle$ for a n -tuple, ignoring the tree order. In particular, we write \bar{b} for tuples of booleans $\langle b_1, \dots, b_n \rangle$.

(Call-by-Push-Value) Simple Types. In the actual technical development of this paper, we will use intersection types. However, we prefer to first give the intuitions about typing in the more familiar setting of simple types. The ground types are (tensors of) booleans. Following Levy [1999] and Ehrhard and Tasson [2019], we then define by mutual induction two kinds of types: *positive* types and *general* types. Only *positive* types can be assigned to variables in the type environment and

Fig. 6. The simply typed $\lambda_!$ -calculus.

can appear in the left hand side of an arrow.

$$\begin{array}{ll} \text{GROUND TYPES} & L, K ::= B \mid L \otimes K \\ \text{POSITIVE TYPES} & P, Q ::= L \mid !A \\ \text{TYPES} & A, B ::= P \mid P \multimap A \end{array}$$

The typing rules are in Fig. 6, where a context \mathcal{P} is a sequence of assignments of positive types P to variables x . As usual, a judgment $\mathcal{P} \vdash t : A$ indicates that t has type A given typing context \mathcal{P} . We write $\pi \triangleright \mathcal{P} \vdash t : A$ to indicate that π is a type derivation of the given judgment. All the rules in Fig. 6 are standard but s-COND. Notice that in rule s-COND by $\{\bar{b} \Rightarrow \text{sample}_{d_{\bar{b}}}\}_{\bar{b} \in \{t, f\}^n}$ we mean that for each possible n -tuple of booleans $\bar{b} \in \{t, f\}^n$ there is a corresponding sample clause. For the sake of brevity, from now on we will often shorten a case expression depending on n variables x_1, \dots, x_n as follows:

$$\mathfrak{c}_{\langle x_1, \dots, x_n \rangle} \triangleq \text{case } \langle x_1, \dots, x_n \rangle \text{ of } \{\bar{b} \Rightarrow \text{sample}_{d_{\bar{b}}}\}_{\bar{b} \in \{t, f\}^n}$$

Remark 3.2 (Additive Contexts). The reader familiar with Linear Logic and calculi based on it (such as [Benton et al. 1993]) may be surprised by the fact that here (as in [Ehrhard 2016; Levy 1999]) the context is managed additively. This is because the only types which are allowed in a context are positive, hence either of the form $!A$, or booleans, coded by additives.² Notice that proper linear types, such as $A \multimap B$, are not allowed in the context (if allowed, their management would be multiplicative).

The Higher-Order and the Low-Level Language. It is standard to encode a ground Bayesian network with a simple let-term of ground type. The reader can easily realize that every ground Bayesian network can be described in a simple, first-order fragment of the $\lambda_!$ -calculus, as we have

²Positive types can be contracted and weakened. This is clear for types of the form $!A$, but holds also for ground types. Indeed a boolean type corresponds to the additive formulas $1 \oplus 1$. Notice that $\perp \& \perp = (1 \oplus 1)^\perp$ can be weakened and contracted.

done in the examples in Sect. 2. In particular, there is no need for the modalities ! and der, which are instead the key to implement higher-order behaviors. Abstraction and application are not necessary, as well. We refer to such a fragment as λ_{low} -calculus. Formally, the grammar for λ_{low} -terms is:

$$\begin{aligned} \text{LOW-LEVEL TERMS } t, u &::= v \mid \text{sample}_d \mid \text{case } v \text{ of } \{\bar{b} \Rightarrow \text{sample}_{d_b}\} \mid \text{obs}(x = b) \mid \\ &\quad \text{let } x = u \text{ in } t \mid \text{letp } \langle x, y \rangle = v \text{ in } t \\ \text{LOW-LEVEL VALUES } v, w &::= x \in \mathcal{V} \mid \langle v, w \rangle \mid t \mid f \end{aligned}$$

It is easy to check that a λ_{low} -term is typable with a ground context if and only if it has ground type and is typable with first-order rules (those highlighted in Fig. 6), only. Going back to our introductory intuitions, we see the $\lambda_!$ -calculus as the target high-level language in which the statistical model is designed by the programmer, and the λ_{low} -calculus as the low-level language, closer to ground Bayesian networks. *Compiling* $\lambda_!$ -terms into λ_{low} -terms is taken care of by *semantical* tools. The first of such tools is the reduction relation, which we introduce next.

3.2 Operational Semantics

In Sect. 2 we have anticipated that the operational semantics of our calculus formalizes the unrolling of a template into a ground Bayesian network, which is its intended meaning. Formally, every $\lambda_!$ -term of ground type compiles (*i.e.*, rewrites) into a λ_{low} -term.

Root Rules. Since the goal is to produce a term describing a Bayesian network, here reduction does not fire probabilistic redexes, *i.e.* we do not actually sample from distributions. As a consequence, a term of shape sample_d never reduces to a value. This feature of our language forces us to opt for a notion of reduction, dubbed *reduction at a distance* [Accattoli and Kesner 2010; Milner 2006], which is a bit more sophisticated than usual, and reminiscent of reduction on graphs, such as proof-nets and bigraphs. Precisely, our reduction is similar to that in [Arrial et al. 2023; Bucciarelli et al. 2020]. Reduction is called *at a distance* because in some of the rules the interacting parts of a redex can be separated by an arbitrarily long (possibly empty) list of let constructs—*i.e.* they are *distant*. Formally, we need the notion of *substitution list*, *i.e.* a sequence of nested let constructors:

$$\text{SUBSTITUTION LISTS } S ::= \langle \cdot \rangle \mid \text{let } x = u \text{ in } S \mid \text{letp } \langle x, y \rangle = v \text{ in } S$$

We are now able to define the rewriting rules which are the base of our reduction relation. We call the term on the left-hand side a *redex*.

$$\begin{array}{l} \text{ROOT RULES} \\ \langle \lambda x.t \rangle S v \mapsto_{\text{db}} \langle t \{x \leftarrow v\} \rangle S \quad \text{let } x = \langle v \rangle S \text{ in } t \mapsto_{\text{dsub}} \langle t \{x \leftarrow v\} \rangle S \\ \text{der } !t \mapsto_{\text{der!}} t \quad \text{letp } \langle x, y \rangle = \langle v, w \rangle \text{ in } t \mapsto_{\text{pm}} t \{x \leftarrow v\} \{y \leftarrow w\} \end{array}$$

$\langle t \rangle S$ stands for the term obtained from S by replacing the hole $\langle \cdot \rangle$ with t (possibly capturing the free variables of t). The rule \mapsto_{db} fires a (possibly distant) beta-redex. The rule \mapsto_{dsub} fires a (possibly distant) let, provided that its argument is a value. The rule $\mapsto_{\text{der!}}$ defrosts a frozen term. The rule \mapsto_{pm} performs pattern matching with pairs. We set $\mapsto \triangleq \mapsto_{\text{db}} \cup \mapsto_{\text{dsub}} \cup \mapsto_{\text{der!}} \cup \mapsto_{\text{pm}}$.

Reduction. A *reduction step* \rightarrow is the closure of \mapsto under evaluation context. Reduction \rightarrow_r (for $r \in \{\text{db}, \text{dsub}, \text{der!}, \text{pm}\}$) is defined similarly. *Evaluation contexts*, which are terms containing exactly one occurrence of a special symbol—the *hole* $\langle \cdot \rangle$ —are defined as follows:

$$\text{EVALUATION CONTEXTS } E ::= \langle \cdot \rangle \mid Ev \mid \text{let } x = E \text{ in } t \mid \text{let } x = u \text{ in } E$$

$E(t)$ stands for the term obtained from E by replacing the hole $\langle \cdot \rangle$ with t (possibly capturing the free variables of t). As it is standard with programming languages, we adopt a weak notion of reduction,

which here means that we do not reduce inside the scope of a ! (a thunk), nor in the scope of a λ . Please notice that given a term of shape $\text{let } x = u \text{ in } t$, reduction can be performed inside either u or t . This is essential to make possible a reduction such as the one in Fig. 5. As a consequence, reduction is *not* deterministic. However, the choice of redex is irrelevant, in the following sense.

PROPOSITION 3.3 (CONFLUENCE).

1. The reduction \rightarrow is confluent.
2. Every normalizing term is strongly normalizing.
3. All maximal reduction sequences from a term t have the same length.

PROOF. Consequences of a diamond-like property, essentially as in [Bucciarelli et al. 2020]. \square

Progress and BN Normal Forms. We say that a term t is in normal form if no reduction applies ($t \not\rightarrow$). It is well-known that simply typed λ -calculi are strongly normalizing. Here we prove that every $\lambda_!$ -term of ground type reduces to a normal form which is a low-level term, that we dub *BN normal form*. The idea—which we will make formal in Sect. 7.1—is that normal forms of ground type directly correspond to ground Bayesian networks, hence the name.

PROPOSITION 3.4 (PROGRESS). *Let t be a $\lambda_!$ -term in normal form and $\pi \triangleright \mathcal{L} \vdash t : L$ a type derivation, where all types in the context \mathcal{L} are ground. Then t is a λ_{low} -term, and π contains first order rules, only.*

PROOF. By induction on the structure of the derivation π . (In the Appendix). \square

The proposition above allows us to describe the shape of BN normal forms, *i.e.* the normal forms of ground type. If we restrict our attention to closed terms, the BN normal forms are the subset of closed low-level terms generated by the following set of productions³:

$$n ::= \text{let } x = s \text{ in } n \mid v \mid s \qquad s ::= \text{let } x = s \text{ in } s \mid \text{sample}_d \mid \mathbf{c}_{(x_1, \dots, x_n)} \mid \text{obs}(x = b)$$

Please notice that BN normal forms are not values, in general. This is because we do not reduce probabilistic primitives.

4 THE INTERSECTION TYPE SYSTEM

Simple type systems guarantee termination, but are poor in expressiveness. In this work we want to specify rich behaviors, such as recursion, and this is why we switch to the *untyped* λ -calculus. However, this is not enough to obtain the results we want, and we still do need a form of typing. Since we are interested in giving a Bayesian network *semantics* to terms, we need to *keep track* of the random variables defined by a term, and to handle the fact that recursive programs generate a finite but *unbounded number* of random variables. Random variables are associated to the sampling and conditional primitives, so we need to take into account how many times these primitives are duplicated during the reduction. A quite natural option (especially when using a calculus inspired by linear logic, as we do) is to consider a type system that explicitly takes into account how many times a sub-term is copied during the evaluation. *Non-idempotent intersection types* for the *untyped* λ -calculus do precisely that. This approach has several advantages:

- *Tracking random variables:* ground types now correspond precisely to random variables, as we discuss in Sect. 4.1 below.
- *Distinguishing copies:* non-idempotent intersection types intrinsically take into account how many times a (sub-)term is copied. This is of fundamental importance for us, since we want to keep track of all the random variables generated by our typed program. Think for example of a single (thunked) sample_d instruction that is duplicated several times during execution.

³Please notice that the converse is not true: not all terms generated by this grammar are typable.

Non-idempotent intersection type derivation duplicate *in advance* each use of a sub-term, and give each a (possibly different) type.

- *Enforcing termination*: Bayesian networks are representations for *finite* probabilistic models. Therefore we are interested in terminating programs, only. Intersection types provide such a guarantee, still allowing for complex behaviors such as general recursion.
- *Ruling out ill-formed terms*: as a by-product, intersection types act as a traditional type system discarding terms which “go wrong”.

We highlight that we could have proceeded differently. We could have considered a *typed* PCF-like language, as in [Ehrhard and Tasson 2019]. Still—for the reasons described above—we would have needed to add, on top of its type system, an intersection type system, like in [Dal Lago and Gaboardi 2011; Ehrhard 2016]. However, dealing with two layers of types is heavy, and we preferred keeping the syntax as simple as possible. This way, from now on we will consider just the intersection type system for the (untyped) $\lambda_!$ -calculus which we have presented in Sect. 3.1. In Sect. 10 we sketch how the (two-layered) type system for the call-by-push-value PCF would look like.

Remark 4.1 (Intersection Types and Turing Completeness). Type systems ensure safety and desirable properties such as termination, or deadlock-freeness. *Intersection types* for the untyped λ -calculus [Coppo and Dezani-Ciancaglini 1978; Coppo et al. 1981] bring this idea to its extreme consequence. Intersection types not only *guarantee* termination, but also *characterize* it, providing a *compositional* presentation of *all and only* the terminating programs. They can indeed be seen as a *semantical tool* for higher-order languages. Being the untyped λ -calculus Turing-complete, the price to pay is that intersection type systems are inherently undecidable. This is not typically considered an issue because such systems have a semantic nature: they are used to give denotational models. However, please notice that *the first-order fragment* of our system *is decidable*.

4.1 Towards Bayesian Networks: Named Types

Before presenting the intersection type system, we need to introduce one more ingredient, namely *named* booleans, which we use to track random variables. In this subsection we give some simple examples to convey the intuitions, which we then formalize in Sect. 4.2.

Describing a Bayesian network (or a marginal distribution) by means of a term of type $\otimes^n B$ is a standard, easy task. However, retrieving a Bayesian network from a term of type $\otimes^n B$ is less immediate, even with low-level terms. Does a low-level term of ground type define a marginal distribution? If yes, over how many random variables? And what is the underlying Bayesian network, if any?

Example 4.2. Let us consider the term t , where $\mathbf{c}(y) \triangleq \text{case } y \text{ of } \{b = \text{sample}_d\}$.

$$t \triangleq \text{let } x = \text{bernoulli}_{0,2} \text{ in let } y = x \text{ in let } z = \mathbf{c}(y) \text{ in } \langle x, y \rangle : B \otimes B$$

We know that the term t defines two r.v.s (say X and Y), because there are two probabilistic constructs. We also know that the output is a probability distribution over tuples in $B \otimes B$. It is however not obvious which variables are involved in the final marginal distribution. We can track which random variables are involved in the term by *naming* the booleans and assigning a *distinct name* to the subject of each probabilistic axiom (we will formalize this in Sect. 4.2).

$$\frac{\frac{\frac{\frac{x : B_X \vdash x : B_X}{x : B_X \vdash \mathbf{c}(y) : B_Y} \quad \frac{y : B_X \vdash y : B_X}{x : B_X, y : B_X, z : B_Y \vdash \langle x, y \rangle : B_X \otimes B_X}}{x : B_X, y : B_X \vdash \text{let } z = \mathbf{c}(y) \text{ in } \langle x, y \rangle} \quad \frac{x : B_X \vdash x : B_X}{x : B_X \vdash \text{let } y = x \text{ in let } z = \mathbf{c}(y) \text{ in } \langle x, y \rangle : B_X \otimes B_X}}{\vdash \text{let } x = \text{sample}_d \text{ in let } y = x \text{ in let } z = \mathbf{c}(y) \text{ in } \langle x, y \rangle : B_X \otimes B_X}$$

We now realize that the marginal distribution defined by the term t is in fact $\Pr(X = b, X = b)$, for $b \in \{t, f\}$. This is a *redundant* version of $\Pr(X = b)$, the distribution over the single variable X . The probabilities associated to the tuple of values in $B_X \otimes B_X$ are indeed $\langle t, t \rangle \mapsto 0.2$, $\langle f, f \rangle \mapsto 0.8$. Necessarily, the tuples $\langle t, f \rangle$ and $\langle f, t \rangle$ have probability 0. Please contrast the term t above with the term $\text{let } x = \text{bernoulli}_{0.2} \text{ in let } y = \mathbf{c}_{\langle x \rangle} \text{ in } \langle x, y \rangle : B_X \times B_Y$, which instead defines a distribution over *two* variables.

In the following we formalize these ideas, exploiting named types to associate a Bayesian network to a ground term. We write only the names X, Y , but please think of them as named booleans B_X, B_Y .

4.2 The Type System

In this section, we introduce the type system, which will be the base for the factor semantics in Sect. 6. In order to focus on the main ideas, we postpone the treatment of the evidence, namely of the construct $\text{obs}(x = b)$ to Sect. 8. This is because while being conceptually easy, it requires some fine tuning of the type system, that would make its description harder to understand.

The Type System. The grammar of intersection types is derived from the one for simple types, and for this reason we keep the same meta-variables. Indeed, we shall not use simple types anymore in the rest of the paper, so no confusion can occur. We assume a countable set $\text{Names} = \{X, Y, Z, \dots\}$ of symbols, called *names*, which play the role of *atomic types*. The grammar of types includes ground types, multisets of types (the proper intersection types), and functional types.

$$\begin{array}{lll} \text{GROUND TYPES} & K, L & ::= X \in \text{Names} \mid K \otimes L \\ \text{POSITIVE TYPES} & P, Q & ::= L \mid [A_1, \dots, A_n] \\ \text{TYPES} & A, B & ::= P \mid P \multimap A \end{array}$$

Here $[\dots]$ denotes the multiset constructor. Please notice that the empty multiset $[\]$ is a positive type, as well. Two changes are present w.r.t. the definition of simple types:

1. *Named Booleans* : the type B of booleans has been substituted by a countable set of names. This means that each *use* of a boolean variable has now a distinct type (name). Morally, different names correspond to different random variables.
2. *Multisets for Thunks* : types of shape $!A$ have been replaced by multisets of types. As usual in non-idempotent intersection types, the idea is that every type inside a multiset corresponds to a single use of the typed term (see Sect. 4.4 for an example).

The typing rules are in Fig. 7; typing contexts, which we separate in *ground contexts* (denoted by Λ) and multiset contexts (denoted by Γ or Δ), are defined in the next paragraph. Given a type A , we denote by $\text{Nm}(A)$ the *set* of names which appear in A . The definition extends to typing contexts (e.g. $\text{Nm}(\Lambda)$) and type derivations ($\text{Nm}(\pi)$).

Contexts. A *typing context* Σ is a (total) map from variables to positive types such that only finitely many variables are not mapped to the empty multiset $[\]$. The *domain* of Σ is the set $\text{dom}(\Sigma) \triangleq \{x \mid \Sigma(x) \neq [\]\}$. A context Σ is *empty* if $\text{dom}(\Sigma) = \emptyset$. A typing context Σ is denoted by $x_1 : P_1, \dots, x_n : P_n$ if $\text{dom}(\Sigma) \subseteq \{x_1, \dots, x_n\}$ and $\Sigma(x_i) = P_i$ for all $1 \leq i \leq n$. Given two typing contexts Σ_1 and Σ_2 such that $\text{dom}(\Sigma_1) \cap \text{dom}(\Sigma_2) = \emptyset$, the typing context Σ_1, Σ_2 is defined as $(\Sigma_1, \Sigma_2)(x) \triangleq \Sigma_1(x)$ if $x \in \text{dom}(\Sigma_1)$, $(\Sigma_1, \Sigma_2)(x) \triangleq \Sigma_2(x)$ if $x \in \text{dom}(\Sigma_2)$, and $(\Sigma_1, \Sigma_2)(x) \triangleq [\]$ otherwise. Observe that $\Sigma, x : [\]$ is equal to Σ . Given a context Σ , it is convenient to partition it into a ground and a multiset context. We call *ground context* (denoted by Λ) the restriction of Σ to the variables which are mapped to ground types, and we call *multiset context* (denoted by Γ or Δ) its

Higher-Order Calculus	
First-Order Rules	
$\frac{\chi \notin \text{Nm}(\Lambda)}{\Lambda \vdash \text{sample}_d : \chi} \quad \text{I-SAMPLE}$	$\frac{\chi \notin \{Y_1, \dots, Y_n\} \text{ and } \chi \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : Y_1, \dots, y_n : Y_n \vdash \mathbf{c}(y_1, \dots, y_n) : \chi} \quad \text{I-COND}$
$\frac{}{\Lambda, x : P \vdash x : P} \quad \text{I-VAR}$	$\frac{\Lambda, \Gamma_1 \vdash u : P \quad \Lambda, \Gamma_2, x : P \vdash t : A}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \text{let } x = u \text{ in } t : A} \quad \text{I-LET}$
$\frac{\Lambda \vdash v : L_1 \quad \Lambda \vdash w : L_2}{\Lambda \vdash \langle v, w \rangle : L_1 \otimes L_2} \quad \text{I-PAIR}$	$\frac{\Lambda \vdash v : L_1 \otimes L_2 \quad \Lambda, x : L_1, y : L_2, \Gamma \vdash t : A}{\Lambda, \Gamma \vdash \text{letp } \langle x, y \rangle = v \text{ in } t : A} \quad \text{I-LETP}$
$\frac{\Lambda, \Gamma, x : P \vdash t : A}{\Lambda, \Gamma \vdash \lambda x. t : P \multimap A} \quad \text{I-ABS}$	$\frac{\Lambda, \Gamma_1 \vdash t : P \multimap A \quad \Lambda, \Gamma_2 \vdash v : P}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash tv : A} \quad \text{I-APP}$
$\frac{(\Lambda, \Gamma_i \vdash t : A_i)_{i=1}^n}{\Lambda, \uplus_i \Gamma_i \vdash !t : [A_1, \dots, A_n]} \quad \text{I-BANG}$	$\frac{\Lambda, \Gamma \vdash v : [A]}{\Lambda, \Gamma \vdash \text{der } v : A} \quad \text{I-DER}$

Fig. 7. The intersection type system iTypes .

complement, *i.e.* the restriction of Σ to the variables which are mapped to multisets. Multiset union \uplus is extended to multiset contexts point-wise, *i.e.* $(\Gamma \uplus \Delta)(x) \triangleq \Gamma(x) \uplus \Delta(x)$, for each variable x .

Remark 4.3. Please notice that in Fig. 7 we have operated a few simplifications in the presentation of the type system. In particular, we do not type boolean constants anymore—this simplifies the I-COND rule, that now is an axiom. Moreover, notice that tuples always have the multiset context empty. As we aforementioned, we postpone the treatment of $\text{obs}(x = b)$ to Sect. 8.

Type Derivations. We write $\pi \triangleright \Lambda, \Gamma \vdash t : A$ to indicate that π is a type derivation (using the full type system in Fig. 7), proving that t has type A given typing context Λ (ground) and Γ (multiset). We write $\pi \triangleright_{\text{low}} \Lambda \vdash t : L$ for a derivation π which uses first order rules, restricted to ground contexts, only.

Naming Condition. We call *main names* those which type the subject of an I-COND or I-SAMPLE rule. Given a type derivation π , we assume that all the main names are *pairwise distinct*. So, each I-COND or I-SAMPLE rule is uniquely identified by a name χ . This requirement is easy to implement. Indeed, it is a sort of Barendregt convention, but for types. One could consider the name introduced by a probabilistic axiom as the address of the axiom in the type derivation.

The First-Order Fragment. Notice that the first-order fragment in Fig. 7 is the same as the first-order fragment of simple types (Fig. 6), the only difference being that now the booleans are named. Clearly this fragment is *decidable*, since any first-order simply typed derivation of $\mathcal{L} \vdash t : \otimes^n B$ (where every type in the context \mathcal{L} is ground) can easily be named, by assigning a distinct name to the subject of every probabilistic axioms, and to every occurrence of boolean type in \mathcal{L} .

4.3 Properties of the Type System

The intersection type system satisfies all the properties one would expect—proofs are in the Appendix. First, types are stable under reduction and expansion (the latter property not holding for simple types).

PROPOSITION 4.4 (SUBJECT REDUCTION/EXPANSION). *Let t be a $\lambda_!$ -term such that $t \rightarrow u$. Then $\Sigma \vdash t : A$ if and only if $\Sigma \vdash u : A$.*

Subject reduction can be strengthened, showing that there is a measure that decreases along each reduction sequence. This gives a combinatorial proof that typable terms are strongly normalizing.

THEOREM 4.5. *Let t be a λ_1 -term. If t is typable, then t is strongly normalizing.*

Crucially, the progress lemma, stated for simple types, still holds for intersection types. This means that every (higher-order, possibly recursive) λ_1 -term which is typable with ground type in a ground context, eventually reduces to a **BN** normal form.

THEOREM 4.6 (COMPILING INTO THE LOW-LEVEL). *Let t be a λ_1 -term such that $\pi \triangleright \Lambda \vdash t : L$. Then $t \rightarrow^* u$, where u is a λ_{low} -term in normal form (a BN normal form).*

Notice that the normal form u has necessarily a type derivation $\pi' \triangleright_{\text{low}} \Lambda \vdash u : L$ (using first-order rules only). Finally, we highlight that the type system is syntax driven. As a consequence:

PROPOSITION 4.7 (UNIQUE DERIVATION). *Let t be a λ_1 -term, and Λ a ground context. Then there exists at most one type derivation π such that $\pi \triangleright \Lambda \vdash t : L$.*

This means that for each λ_1 -term t and ground context Λ , there exists at most one ground type L such that $\Lambda \vdash t : L$ admits a type derivation. In particular, the type derivation for a term in BN normal form is uniquely determined. By subject expansion we have:

THEOREM 4.8. *Let t be a closed λ_1 -term t . If t reduces to BN normal form, then a type derivation $\pi \triangleright \vdash t : L$ exists and is unique.*

4.4 Putting the Calculus and the Type System at Work

We illustrate with some examples the expressiveness of the calculus, and the use of the type system.

Expressiveness. Since we have access to the full expressiveness of the untyped λ -calculus (Remark 4.1), we can use a standard encoding (in its call-by-push-value flavor) of integers, arithmetic, if/then/else, and fixed point combinators.

Example 4.9 (Encoding Recursive Behavior). We are now able to encode Dynamic Bayesian networks, such as the one depicted in Fig. 4 (from [Koller and Friedman 2009], Ch.6). The idea behind this model is that a system evolves with time in a stochastic way. At each time step, one random variable S_{i+1} , which depends only on the previous state S_i , represents the new state, while the observation O_{i+1} depends only on the current state S_{i+1} . A typical query in these kinds of models is

$$\Pr(S_n = s \mid O_1 = o_1, \dots, O_n = o_n)$$

which intuitively means: after n time steps, what is the probability of being in a certain state, knowing all the observations? We can write the template of Fig. 4 as follows:

```

t  $\triangleq$   $\lambda n$ . let  $s_0 = \text{bernoulli}_p$  in  $u \ n \ s_0$ 
u  $\triangleq$  fix !( $\lambda x$ .  $\lambda n$ .  $\lambda s$ . if isZero( $n$ ) then  $s$ 
                                     else let  $s' = \mathbf{c}^S_{(s)}$  in
                                     let  $o' = \mathbf{c}^O_{(s')}$  in
                                     let  $m = \text{pred}(n)$  in
                                     let  $r = (\text{der } x) \ m \ s'$ 
                                     in  $\langle o', r \rangle$ )

```

Then $t_{\underline{n}}$ —where \underline{n} is an encoding of the integer n —represents the template that is to be unrolled n times. Operationally, we have exactly that the fixed point operator is unfolded n times generating the unrolled Bayesian network. From the point of view of the type system, we have that the

type of t_n depends on the integer n :

$$\vdash t_n : 0_1 \otimes \cdots \otimes 0_n \otimes S_n$$

The Intersection Types, in use. The term which encodes the BN of Fig. 1 is easily typed in the first-order fragment—the reader can find the derivation in Example 7.2. Here we give an example of type derivation where *multiple copies* are involved, so that we need to use the *multiset type*. First, observe that a type of shape $[A_1, \dots, A_n]$ can be thought of as an informative refinement of the thunk type $!A$. Sub-terms are typed several times, once for each copy that will be produced during the reduction. This feature is crucial to handle random variables. We stress that the inability to deal with an unbounded number of random variables is the key issue which limits to first-order the compilation of programs into BNs, or data flow analysis (see e.g. [Gorinova et al. 2022; van de Meent et al. 2018]).

Example 4.10 (Multiple Coin Tosses). Consider a term u similar to that in Fig. 5, modeling two tosses of the same (biased) coin: $u \triangleq \text{let } x = \text{sample}_d \text{ in let } y = !(c(x)) \text{ in } \langle x, \text{der } y, \text{der } y \rangle$. For readability, here we use some syntactic sugar. The (unique) type derivation for u is

$$\frac{\frac{\frac{x : X \vdash c(x) : Y_1 \quad x : X \vdash c(x) : Y_2}{x : X \vdash !(c(x)) : [Y_1, Y_2]} \quad \frac{x : X \vdash x : X \quad x : X, y : [Y_1, Y_2] \vdash \langle \text{der } y, \text{der } y \rangle : Y_1 \otimes Y_2}{x : X, y : [Y_1, Y_2] \vdash \langle x, \text{der } y, \text{der } y \rangle : X \otimes Y_1 \otimes Y_2}}{\vdash \text{sample}_d : X} \quad x : X \vdash \text{let } y = !(c(x)) \text{ in } \langle x, \text{der } y, \text{der } y \rangle : X \otimes Y_1 \otimes Y_2}{\pi \vdash u : X \otimes Y_1 \otimes Y_2}$$

5 THE SEMANTICS OF BAYESIAN NETWORKS

In this section we formally define the semantics of Bayesian networks. First, let us briefly revise the language of Bayesian modeling. For more details, we refer to [Darwiche 2008] for a concise presentation, and to standard texts for an exhaustive treatment [Darwiche 2009; Neapolitan 2003; Pearl 1988].

5.1 Random Variables

Bayesian methods provide a formalism for reasoning about partial beliefs under conditions of uncertainty. Since we cannot determine for certain the state of some features of interest, we settle for determining how *likely* it is that a particular feature is in a particular state. Random variables represent features of the system being modeled. For the purpose of modeling, a random variable can be seen as a *name* for an atomic proposition (e.g. "Wet") which assumes values from a set of states (e.g. $\{t, f\}$). The system is modeled as a *joint* probability distribution on all possible values of the variables of interest – an element in the sample space represents a possible state of the system.

Example 5.1. The canonical sample space sketched in Fig. 2 consists of 2^4 tuples (only some entries are displayed); to each tuple \bar{x} is associated a probability. The event $(R = t)$ contains 2^3 tuples, the event $(R = t, W = t)$ contains 2^2 tuples, and has probability 0.33.

Remark 5.2. Notice that in Bayesian modeling, random variables are identified first, and only implicitly become functions on a sample space. We refer to the excellent textbook by Neapolitan [Neapolitan 2003] (Ch. 1) for a formal treatment relating the notion of random variable as used in Bayesian inference, with the classical definition of function on a sample space.

Given a countable set Names, we associate to each name $X \in \text{Names}$ a set of values, denoted by $\text{Val}(X)$ (typically $\text{Val}(X) = \{t, f\}$). From now on, we silently identify a name X with the pair

$(X, \text{Val}(X))$, which effectively defines a *random variable* (r.v.). A finite set of names $\mathbb{X} = \{X_1, \dots, X_n\}$ defines a "compound" r.v. whose value set $\text{Val}(\mathbb{X})$ is the Cartesian product $\text{Val}(X_1) \times \dots \times \text{Val}(X_n)$.

Notation 5.3. The metavariables $\mathbb{X}, \mathbb{Y}, \mathbb{Z}$ range over finite *sets of names* (r.v.s). As standard, a lowercase letter x denotes a generic value $x \in \text{Val}(X)$, and \bar{x} denotes a tuple in the cartesian product $\text{Val}(\mathbb{X}) \triangleq \text{Val}(X_1) \times \dots \times \text{Val}(X_n)$. Moreover, we use juxtaposition as a tuple constructor, e.g. if $x \in \text{Val}(X)$ and $\bar{y} \in \text{Val}(Y_1) \times \dots \times \text{Val}(Y_n)$, then $x\bar{y} \in \text{Val}(X) \times \text{Val}(Y_1) \times \dots \times \text{Val}(Y_n)$. Given a subset $\mathbb{Y} \subseteq \mathbb{X}$, we denote by $\bar{x}|_{\mathbb{Y}}$ the restriction of \bar{x} to \mathbb{Y} (so, $\bar{x}|_{\mathbb{Y}} \in \text{Val}(\mathbb{Y})$). Given two sets of names \mathbb{X} and \mathbb{Y} , we say that $\bar{x} \in \text{Val}(\mathbb{X})$ and $\bar{y} \in \text{Val}(\mathbb{Y})$ *agree on the common names* ($\bar{x} \sim \bar{y}$ for short) whenever $\bar{x}|_{\mathbb{X} \cap \mathbb{Y}} = \bar{y}|_{\mathbb{X} \cap \mathbb{Y}}$.

5.2 Factors, Sum and Product Operations

Inference algorithms rely on basic operations on a class of functions known as *factors*, which generalize the notions of probability distribution and of conditional distribution. Factors will be the key ingredients also in our semantics.

Definition 5.4 (Factor). A *factor* ϕ over a set of names (r.v.s) \mathbb{X} is a function $\phi : \text{Val}(\mathbb{X}) \rightarrow \mathbb{R}_{\geq 0}$ mapping each tuple $\bar{x} \in \mathbb{X}$ to a non-negative real.

When \mathbb{X} is clear from the context, we simply write ϕ (omitting the superscript \mathbb{X}); then $\text{Nm}(\phi)$ denotes \mathbb{X} . Letters ϕ, ψ range over factors. Please notice that in the literature about BNs, $\phi(\bar{x})$ is often written $\phi_{\bar{x}}$. We adopt this convenient notation when making explicit calculations.

Example 5.5. Factors generalize familiar concepts from probability theory.

- A *joint probability distribution* over the set \mathbb{X} is a factor ϕ which maps each tuple $\bar{x} \in \text{Val}(\mathbb{X})$ to a probability $\phi(\bar{x})$ such that $\sum_{\bar{x} \in \text{Val}(\mathbb{X})} \phi(\bar{x}) = 1$.
- A *conditional probability table* (CPT) for X given \mathbb{Y} is a factor ϕ over $\{X\} \cup \mathbb{Y}$ which maps each tuple $x\bar{y} \in \text{Val}(\{X\} \cup \mathbb{Y})$ to a probability $\phi(x\bar{y})$ such that for each $\bar{y} \in \mathbb{Y}$, $\sum_{x \in \text{Val}(X)} \phi(x\bar{y}) = 1$.⁴

Factors come with two important operations: sum (out) and product. Summing out a name (r.v.) Z from a factor means that we are removing Z , thus obtaining a smaller factor. As depicted in Fig. 2, intuitively we do so by merging all tuples which agree on all the other variables but Z .

Definition 5.6 (Sum Out). The *sum out* of $Z \subseteq \mathbb{X}$ from ϕ is a factor $\sum_Z \phi$ over $\mathbb{Y} \triangleq \mathbb{X} - Z$, defined as:

$$\left(\sum_Z \phi \right) (\bar{y}) \triangleq \sum_{\bar{z} \in \text{Val}(Z)} \phi(\bar{z}\bar{y})$$

Multiplication of factors is defined in such a way that only "coherent" pairs are multiplied.

Definition 5.7 (Product). The *product* of ϕ_1 and ϕ_2 is a factor $\phi_1 \odot \phi_2$ over $Z \triangleq \mathbb{X} \cup \mathbb{Y}$, defined as:

$$(\phi_1 \odot \phi_2)(\bar{z}) \triangleq \phi_1(\bar{x})\phi_2(\bar{y}) \quad \text{where } \bar{x} = \bar{z}|_{\mathbb{X}} \text{ and } \bar{y} = \bar{z}|_{\mathbb{Y}}.$$

We denote n -ary products by $\odot_n \phi_n$. We denote by $\mathbf{1}_{\mathbb{Y}} \triangleq \mathbf{1}$ the factor over the set of names \mathbb{Y} , sending every tuple of $\text{Val}(\mathbb{Y})$ to 1. Observe that $\phi \odot \mathbf{1} = \phi$ if $\mathbb{Y} \subseteq \mathbb{X}$. Factors over an empty set of

⁴Please notice that here, and in the following definition of sum out, we slightly abuse the notation. In fact, as standard, we consider the tuples as sequences indexed by the set of random variables. Every time, we present the tuples ordered in the most convenient way for a compact definition.

variables are allowed, and called trivial. In particular, we write $\mathbf{1}_\emptyset \triangleq \mathbf{1}$ for the trivial factor assigning 1 to the empty tuple. Product and summation are both commutative, product is associative, and—crucially—they distribute *under suitable conditions*:

$$\text{If } \mathbb{Z} \cap \text{Nm}(\phi_1) = \emptyset \text{ then } \sum_{\mathbb{Z}} (\phi_1 \odot \phi_2) = \phi_1 \odot \left(\sum_{\mathbb{Z}} \phi_2 \right) \quad (2)$$

This distributivity is the key property on which exact inference algorithms rely. CPT's being factors, they admit sum and product operations. Please notice that the result of such operations is not necessarily a CPT, but, of course, it is a factor.

Remark 5.8 (Cost of Operations on Factors). Summing out any number of variables from a factor ϕ demands $\mathcal{O}(\exp(w))$ time and space, where w is the number of variables over which ϕ is defined. Multiplying k factors demands $\mathcal{O}(k \cdot \exp(w))$ time and space, where w is the number of variables in the resulting factor.

5.3 The Semantics of Bayesian Networks

Bayesian networks are graph-theoretic objects able to represent large probability distributions compactly, via a *factorized representation*. Inference algorithms then implement *factorized computations*.

Definition 5.9. A Bayesian network \mathcal{B} over the set of r.v.s \mathbb{X} is a pair $(\mathcal{G}, \mathcal{T})$ where:

- \mathcal{G} is a directed acyclic graph (DAG) over the set of nodes \mathbb{X} .
- \mathcal{T} assigns, to each variable $X \in \mathbb{X}$ a *conditional probability table (a CPT)*, which is a factor ϕ^X over variables $\{X\} \cup \text{Pa}(X)$, where $\text{Pa}(X)$ denotes the set of parents of X in \mathcal{G} .

The graph structure and independence assumptions on it yield the correctness of the semantics.

THEOREM 5.10 (PEARL [1986]). A Bayesian network \mathcal{B} over the set of r.v.s \mathbb{X} defines a unique probability distribution over \mathbb{X} (its semantics):

$$\llbracket \mathcal{B} \rrbracket \triangleq \bigodot_{X \in \mathbb{X}} \phi^X.$$

Please notice also that, given a Bayesian network \mathcal{B} defining a probability distribution over \mathbb{X} , the *marginal distribution* of $\llbracket \mathcal{B} \rrbracket$ over a subset $\mathbb{Y} \subseteq \mathbb{X}$ is defined by $\sum_{\mathbb{X}-\mathbb{Y}} \llbracket \mathcal{B} \rrbracket$.

6 THE SEMANTICS OF TYPED TERMS

This section contains the main result of this paper, namely the fact that we can endow terms of ground type with a factor-based semantics that reflects the probabilistic behavior of the term. Moreover, we prove that the semantics is compositional, in the sense that it can be computed following the structure of intersection type derivations.

Semantics of the Probabilistic Axioms. The intuition guiding the definition of our semantics is the very definition of the semantics of a BN, as we have just seen in Sect. 5.3. In general, this depends only on the CPT's which are assigned to each random variable. We apply the same principle in the realm of (intersection) typed $\lambda_!$ -terms. The idea is that we can associate a CPT to each probabilistic axiom in a type derivation, and then we define the semantics of the typed term as their product (as factors), summing out the names not occurring in the final type judgment to obtain the marginal which is specified by the term. We start by formally defining the factor which is associated to a probabilistic axiom. We give an *example* of how the following definitions work in Fig. 8.

Definition 6.1 (Probabilistic Axioms). Recall that we identify each name X with the pair $(X, \text{Val}(X))$, effectively defining a r.v. (see Sect. 5.1).

- I-SAMPLE. We associate to the axiom $\Lambda \vdash \text{sample}_d : X$ the factor ϕ over the r.v. $\{X\}$ such that $\phi(x) \triangleq d(x)$ for each $x \in \text{Val}(X)$.
- I-COND. Let us consider the following instance of the I-COND axiom.

$$\frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : Y_1, \dots, y_n : Y_n \vdash \text{case } \langle y_1, \dots, y_n \rangle \text{ of } \{\bar{b} \Rightarrow \text{sample}_{d_{\bar{b}}}\}_{\bar{b} \in \{t, f\}^n} : X} \text{ I-COND}$$

We associate to this axiom the factor ϕ over the set of r.v.s $\{Y_1, \dots, Y_n, X\}$, such that $\phi(\bar{b}x) \triangleq d_{\bar{b}}(x)$, for each $\bar{b}x \in \text{Val}(Y_1) \times \dots \times \text{Val}(Y_n) \times \text{Val}(X)$ ⁵.

Semantics of a Type Derivation. Once given the interpretation of the probabilistic axioms, it is straightforward to extend the interpretation to any type derivation of ground type. We multiply all the factors associated to the axioms, and then sum out all the names not appearing in the conclusion.

Definition 6.2 (Semantics of a Type Derivation). Let $\pi \triangleright J$ be a type derivation, $\text{Cpts}(\pi)$ the set of factors associated to its probabilistic axioms, and $\mathbb{X} = \bigcup_{\phi \in \text{Cpts}(\pi)} \text{Nm}(\phi)$. Then the semantics of π is:

$$\llbracket \pi \rrbracket \triangleq \sum_{\mathbb{X} - \text{Nm}(J)} \left(\bigodot_{\phi \in \text{Cpts}(\pi)} \phi \right)$$

Remark 6.3. If π is a type derivation such that $\text{Cpts}(\pi) = \emptyset$, then $\llbracket \pi \rrbracket = \mathbf{1}_\emptyset$. Notice, in particular, that this is the case for each I-VAR axiom.

On Compositionality. One immediately notices that the above definition of semantics does *not* look compositional. Indeed, the semantics of a type derivation π is computed looking *globally* at π , in particular at its probabilistic axioms. We ask ourselves, and we answer in the positive, if it is possible to give a more local, modular way, of computing the very same semantics. Informally, given a type derivation π obtained from the composition of π_1, \dots, π_n , such as

$$\pi \triangleq \frac{\pi_1 \quad \dots \quad \pi_n}{J}$$

⁵Please notice that we are a bit informal here, because we assume that Y_1, \dots, Y_n are pairwise distinct. This is not an obligation, so the actual definition is more involved. This is, however, just a technical point that does not affect the meaning of the definition. For the sake of completeness, we provide the technically precise definition in the Appendix.

$\frac{}{\Lambda \vdash \text{bernoulli}_{0,2} : R} \text{ I-SAMPLE}$ <div style="margin-left: 40px;"> <table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th>R</th><th>Pr(R)</th></tr> <tr><td>t</td><td>0.2</td></tr> <tr><td>f</td><td>0.8</td></tr> </table> $\phi_1 \triangleq$ </div>	R	Pr(R)	t	0.2	f	0.8	$\frac{}{\Lambda, r : R \vdash \text{case } r \text{ of } \{t \Rightarrow \text{bernoulli}_{0,7}; f \Rightarrow \text{bernoulli}_{0,01}\} : W} \text{ I-COND}$ <div style="margin-left: 40px;"> <table border="1" style="border-collapse: collapse; text-align: center;"> <tr><th>R</th><th>W</th><th>Pr(W R)</th></tr> <tr><td>t</td><td>t</td><td>0.7</td></tr> <tr><td>t</td><td>f</td><td>0.3</td></tr> <tr><td>f</td><td>t</td><td>0.01</td></tr> <tr><td>f</td><td>f</td><td>0.99</td></tr> </table> $\phi_2 \triangleq$ </div>	R	W	Pr(W R)	t	t	0.7	t	f	0.3	f	t	0.01	f	f	0.99
R	Pr(R)																					
t	0.2																					
f	0.8																					
R	W	Pr(W R)																				
t	t	0.7																				
t	f	0.3																				
f	t	0.01																				
f	f	0.99																				

Fig. 8. The factor associated to this I-SAMPLE axiom is ϕ_1 . The factor associated to this I-COND axiom is ϕ_2 . For example, $\phi_2(tf) \triangleq \text{bernoulli}_{0,7}(f)$.

we would like to obtain $\llbracket \pi \rrbracket$ from $\llbracket \pi_1 \rrbracket, \dots, \llbracket \pi_n \rrbracket$. In particular, following the pattern of Definition 6.2, we could write:

$$\llbracket \pi \rrbracket = \sum_{\mathbb{Z}} \left(\bigodot_i \llbracket \pi_i \rrbracket \right) \quad \text{where } \mathbb{Z} = \bigcup_i \text{Nm}(\llbracket \pi_i \rrbracket) - \text{Nm}(J) \quad (3)$$

In words, composition is obtained by *first* performing the product $\bigodot_i \llbracket \pi_i \rrbracket$ —which yields a factor over the names $\bigcup_i \text{Nm}(\llbracket \pi_i \rrbracket)$ —and *then* marginalizing, by summing out the names which do not appear in the conclusion J . Such a notion of composition is yet a variant of the pervasive paradigm

$$\text{composition} = \text{parallel composition} + \text{hiding}.$$

The problem now is that the equation 3 is not a priori true. This is because sum out and product do not distribute in general, but only under suitable conditions. The type system design is crucial to guarantee that the factors semantics is indeed compositional. We illustrate this fact with an example.

Example 6.4 (Types and Compositionality). Consider the following derivation, which is *non well-typed* because the names introduced by the first and third probabilistic axiom are not distinct. Below, $\mathbf{c}^1, \mathbf{c}^3$ are sample terms, and $\mathbf{c}^2(z), \mathbf{c}^4(z')$ are case expressions. The terms t, u are those typed by π_1, ρ_1 , respectively. The premise ρ_2 is the obvious derivation of $x : X, y : Y \vdash \langle x, y \rangle : X \otimes Y$.

$$\frac{\frac{\frac{\vdash \mathbf{c}^1 : Z \diamond \phi^1 \quad \{Z\}}{\pi_1 \triangleright \text{let } z = \mathbf{c}^1 \text{ in } \mathbf{c}^2(z) : X} \quad \{Z, X\}}{\vdash \mathbf{c}^1 : Z \diamond \phi^1 \quad z : Z \vdash \mathbf{c}^2(z) : X \diamond \phi^2} \quad \frac{\frac{x : X \vdash \mathbf{c}^3 : Z \diamond \phi^3 \quad \{Z\} \quad x : X, z' : Z \vdash \mathbf{c}^4(z') : Y \diamond \phi^4 \quad \{Z, Y\}}{\rho_1 \triangleright x : X \vdash \text{let } z' = \mathbf{c}^3 \text{ in } \mathbf{c}^4(z') : Y} \quad \rho_2}{\pi_2 \triangleright x : X \vdash \text{let } y = u \text{ in } \langle x, y \rangle : X \otimes Y}}{\pi \triangleright \text{let } x = t \text{ in let } y = u \text{ in } \langle x, y \rangle : X \otimes Y}$$

We annotate each of the four probabilistic axioms with the corresponding CPT. It is easy to check that compositionality (Eq. (3)) does not hold, because $\llbracket \pi \rrbracket = \sum_Z (\phi^1 \odot \phi^2 \odot \phi^3 \odot \phi^4) \neq (\sum_Z \phi^1 \odot \phi^2) \odot (\sum_Z \phi^3 \odot \phi^4) = \llbracket \pi_1 \rrbracket \odot \llbracket \pi_2 \rrbracket$. Observe that $\llbracket \pi \rrbracket$ is a factor over $\{X, Y\}$, $\llbracket \pi_1 \rrbracket$ over $\{X\}$, and $\llbracket \pi_2 \rrbracket$ over $\{Y\}$. So for example we have:

$$\llbracket \pi \rrbracket_{tt} = \phi_t^1 \cdot \phi_{tt}^2 \cdot \phi_t^3 \cdot \phi_{tt}^4 + \phi_f^1 \cdot \phi_{ft}^2 \cdot \phi_f^3 \cdot \phi_{ft}^4 \quad \llbracket \pi_1 \rrbracket_t = \phi_t^1 \cdot \phi_{tt}^2 + \phi_f^1 \cdot \phi_{ft}^2 \quad \llbracket \pi_2 \rrbracket_t = \phi_t^3 \cdot \phi_{tt}^4 + \phi_f^3 \cdot \phi_{ft}^4$$

$$\text{Therefore } \llbracket \pi \rrbracket_{tt} \neq \left(\phi_t^1 \cdot \phi_{tt}^2 + \phi_f^1 \cdot \phi_{ft}^2 \right) \cdot \left(\phi_t^3 \cdot \phi_{tt}^4 + \phi_f^3 \cdot \phi_{ft}^4 \right) = \llbracket \pi_1 \rrbracket_t \cdot \llbracket \pi_2 \rrbracket_t = (\llbracket \pi_1 \rrbracket \odot \llbracket \pi_2 \rrbracket)_{tt}.$$

Proving Compositionality. We are ready to prove that $\llbracket \pi \rrbracket$ can be compositionally defined for every typed derivation $\pi \triangleright \Lambda \vdash t : L$. The crucial property—guaranteed by the type system—is that π is well-formed, in the technical sense given below. Such a property is the key ingredient in the proof of compositionality, because it ensures the distributivity of the sum over the product.

Definition 6.5 (Well-Formedness).

1. The type derivations $\pi_1 \triangleright J_1, \dots, \pi_n \triangleright J_n$ are *compatible* if

$$\text{exists } j \text{ s.t. } Z \in \text{Nm}(\pi_j) \text{ and } Z \notin \text{Nm}(J_j) \quad \Rightarrow \quad Z \notin \text{Nm}(\pi_i) \text{ for any } i \neq j.$$

2. A type derivation π is *well-formed* if for every rule in π , its premises are compatible.

LEMMA 6.6 (WELL-FORMED DERIVATIONS). *Every type derivation $\pi \triangleright \Lambda \vdash t : L$ is well-formed.*

We postpone to Sect. 7 the discussion of the proof, which relies on a fine analysis of the flow of the computation. Using this result, we are able to prove that the semantics—that we have defined in a global way—can indeed be computed compositionally, validating Eq. (3).

PROPOSITION 6.7 (COMPOSITIONALITY). *Let $\pi \triangleright J$ be the following type derivation:*

$$\frac{\pi_1 \triangleright J_1 \quad \dots \quad \pi_n \triangleright J_n}{\pi \triangleright J}$$

If π_1, \dots, π_n are compatible, then:

$$\llbracket \pi \rrbracket = \sum_{\mathbb{Z}} \left(\bigodot_i \llbracket \pi_i \rrbracket \right) \quad \text{where } \mathbb{Z} \triangleq \bigcup_i \text{Nm}(\llbracket \pi_i \rrbracket) - \text{Nm}(J)$$

PROOF. Wlog, consider $n = 2$. Let us set $\Phi_\pi \triangleq \bigodot_{\phi \in \text{Cpts}(\pi)} \phi$ for each derivation π . By Def. 6.2, we can write $\llbracket \pi_i \rrbracket = \sum_{\mathbb{W}_i} \Phi_{\pi_i}$, where $\mathbb{W}_i = \text{Nm}(\Phi_{\pi_i}) - \text{Nm}(J_i)$ ($i \in \{1, 2\}$). Crucially, since π_1 and π_2 are compatible, $\mathbb{W}_1 \cap \text{Nm}(\pi_2) = \emptyset$ and $\mathbb{W}_2 \cap \text{Nm}(\pi_1) = \emptyset$. Hence, by Eq. (2), sum and product distribute, and we have:

$$\llbracket \pi_1 \rrbracket \odot \llbracket \pi_2 \rrbracket = \sum_{\mathbb{W}_1} \Phi_{\pi_1} \odot \sum_{\mathbb{W}_2} \Phi_{\pi_2} = \sum_{\mathbb{W}_1} \sum_{\mathbb{W}_2} (\Phi_{\pi_1} \odot \Phi_{\pi_2}) = \sum_{\mathbb{W}_1} \sum_{\mathbb{W}_2} \Phi_\pi \quad (4)$$

where we used the fact that $\text{Cpts}(\pi) = \text{Cpts}(\pi_1) \cup \text{Cpts}(\pi_2)$. Now let $\mathbb{Y}_i \triangleq \text{Nm}(\llbracket \pi_i \rrbracket)$; since $\text{Nm}(\Phi_{\pi_i}) = \mathbb{Y}_i \uplus \mathbb{W}_i$ and, by the compatibility of π_1 and π_2 , $\mathbb{W}_i \cap \text{Nm}(\Phi_{\pi_j}) = \emptyset$ for $i \neq j$, we have

$$\text{Nm}(\Phi_\pi) = (\mathbb{Y}_1 \cup \mathbb{Y}_2) \uplus (\mathbb{W}_1 \uplus \mathbb{W}_2). \quad (5)$$

Let $\mathbb{Z} \triangleq (\mathbb{Y}_1 \cup \mathbb{Y}_2) - \text{Nm}(J)$; by Eq. (5) and compatibility, $\text{Nm}(\Phi_\pi) - \text{Nm}(J) = \mathbb{Z} \uplus \mathbb{W}_1 \uplus \mathbb{W}_2$. Therefore

$$\sum_{\mathbb{Z}} (\llbracket \pi_1 \rrbracket \odot \llbracket \pi_2 \rrbracket) = \sum_{\mathbb{Z}} \sum_{\mathbb{W}_1} \sum_{\mathbb{W}_2} \Phi_\pi = \sum_{\text{Nm}(\Phi_\pi) - \text{Nm}(J)} \Phi_\pi \triangleq \llbracket \pi \rrbracket$$

where we sum out \mathbb{Z} from both sides of Eq. (4). \square

Inductive Interpretation of Type Derivations. Now that we have proved that our semantics is compositional, we are able to compute it inductively, starting from the axioms, and then following the structure of the type derivation. Probabilistic axioms are assigned a CPT as indicated in Def. 6.1. The I-VAR axioms are assigned the trivial factor 1_\emptyset (see Remark 6.3). Then the semantics of each sub-derivation is inductively obtained following Prop. 6.7. In Fig. 9 we decorate the intersection type system with factors, according to this process. A decorated type judgment is written $\Sigma \vdash t : A \diamond \psi$, where ψ is the inductively computed factor.

LEMMA 6.8. *Let $\pi \triangleright J \diamond \psi$ be a well-formed type derivation. Then $\llbracket \pi \rrbracket = \psi$.*

PROOF. By induction on the derivation. The property trivially holds for all axioms. Assume that $\pi \triangleright J \diamond \psi_i$ is obtained from derivations π_i ($1 \leq i \leq n$). Since π is well-formed, by definition of well-formedness also the derivations π_i are well-formed. Then by *i.h.*, $\psi_i = \llbracket \pi_i \rrbracket$. Thus, by letting $\mathbb{Z} \triangleq \bigcup_i \text{Nm}(\psi_i) - \text{Nm}(J)$, we have

$$\psi := \sum_{\mathbb{Z}} \left(\bigodot_i \psi_i \right) = \sum_{\mathbb{Z}} \left(\bigodot_i \llbracket \pi_i \rrbracket \right) = \llbracket \pi \rrbracket$$

where the last equality follows from Prop. 6.7. \square

Since derivations of ground type are always well-formed (Lemma 6.6), we have proved that:

THEOREM 6.9. *Let $\pi \triangleright \Lambda \vdash t : L \diamond \psi$ be a derivation of ground type. Then $\llbracket \pi \rrbracket = \psi$.*

This theorem states that the semantics $\llbracket \pi \rrbracket$ (as in Def. 6.2) of a type derivation π can be *inductively computed*, as described in Fig. 9.

Higher-Order $\lambda_!$-Calculus	
First-Order Rules	
$\frac{X \notin \text{Nm}(\Lambda) \quad (\phi \text{ as in Def. 6.1})}{\Lambda \vdash \text{sample}_d : X \diamond \phi} \text{I-SAMPLE}$	$\frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda) \quad (\phi \text{ as in Def. 6.1})}{\Lambda, y_1 : Y_1, \dots, y_n : Y_n \vdash \mathbf{c}\langle y_1, \dots, y_n \rangle : X \diamond \phi} \text{I-COND}$
$\frac{}{\Lambda, x : P \vdash x : P \diamond \mathbf{1}} \text{I-VAR}$	$\frac{\Lambda, \Gamma_1 \vdash u : P \diamond \psi_1 \quad \Lambda, \Gamma_2, x : P \vdash t : A \diamond \psi_2 \quad Z = (Y_1 \cup Y_2) - \text{Nm}(\Lambda, \Gamma_1, \Gamma_2, A)}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \text{let } x = u \text{ in } t : A \diamond \sum_Z \psi_1 \odot \psi_2} \text{I-LET}$
$\frac{\Lambda \vdash v : L_1 \diamond \mathbf{1} \quad \Lambda \vdash w : L_2 \diamond \mathbf{1}}{\Lambda \vdash \langle v, w \rangle : L_1 \otimes L_2 \diamond \mathbf{1}} \text{I-PAIR}$	$\frac{\Lambda \vdash v : L_1 \otimes L_2 \diamond \mathbf{1} \quad \Lambda, \Gamma, x : L_1, y : L_2 \vdash t : A \diamond \psi}{\Lambda, \Gamma \vdash \text{letp } \langle x, y \rangle = v \text{ in } t : A \diamond \psi} \text{I-LETP}$
$\frac{\Lambda, \Gamma, x : P \vdash t : A \diamond \psi}{\Lambda, \Gamma \vdash \lambda x. t : P \rightarrow A \diamond \psi} \text{I-ABS}$	$\frac{\Lambda, \Gamma_1 \vdash t : P \rightarrow A \diamond \psi_1 \quad \Lambda, \Gamma_2 \vdash v : P \diamond \psi_2 \quad Z = (Y_1 \cup Y_2) - \text{Nm}(\Lambda, \Gamma_1, \Gamma_2, A)}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash tv : A \diamond \sum_Z \psi_1 \odot \psi_2} \text{I-APP}$
$\frac{(\Lambda, \Gamma_i \vdash t : A_i \diamond \psi_i)_{i=1}^n \quad n \geq 0}{\Lambda, \uplus_{i=1}^n \Gamma_i \vdash !t : [A_1, \dots, A_n] \diamond \bigcirc_i \psi_i} \text{I-BANG}$	$\frac{\Lambda, \Gamma \vdash v : [A] \diamond \psi}{\Lambda, \Gamma \vdash \text{der } v : A \diamond \psi} \text{I-DER}$

Fig. 9. Inductive Interpretation of typed terms (judgments are annotated—in blue—with their interpretation).

Invariance of the Semantics. The semantics is invariant under reduction and expansion. This is due to the fact that probabilistic axioms are stable w.r.t. reduction and expansion. Indeed, non-idempotent intersection derivations somehow internalize the process of rewriting.

THEOREM 6.10 (INVARIANCE). *Let t be a $\lambda_!$ -term and $t \rightarrow u$. Then: $\Lambda \vdash t : L \diamond \psi \Leftrightarrow \Lambda \vdash u : L \diamond \psi$.*

Semantics Completion. We conclude with a remark. The reader may expect that the interpretation of a type derivation $\pi \triangleright J$ were a factor over $\text{Nm}(J)$. For example, one could expect the interpretation of an identity axiom to be $y : Y \vdash y : Y \diamond \mathbf{1}^{\{Y\}}$ instead of $y : Y \vdash y : Y \diamond \mathbf{1}$. The fact is that our semantics focuses only on the *probabilistic content* of the derivation $\pi \triangleright J$. Please notice that the non-probabilistic information is already fully contained in the type judgment J , because intersection types carry such information. Indeed, an interpretation of $\pi \triangleright J$ as a factor over $\text{Nm}(J)$ is easily obtained by a form of *completion*. We give more details in the Appendix. We mention also that the completed interpretation is a needed step to bridge the gap between our semantics and *weighted relational models/probabilistic coherence spaces* such as [Ehrhard et al. 2014; Ehrhard and Tasson 2019; Laird et al. 2013].

7 BAYESIAN NETWORKS GO WITH THE FLOW

In this section we prove two results which we have already anticipated:

- we show that every (closed) term t of ground type corresponds to a Bayesian network \mathcal{B}_t , and that the two have the *same semantics*.⁶
- we prove the central result making the semantics of terms compositional, namely that every type derivation $\pi \triangleright \Lambda \vdash t : L$ is well-formed (Lemma 6.6).

The key ingredient underlying both results is the same: we associate to each type derivation π a directed graph— $\text{flow}(\pi)$ —which essentially describes the *flow of the computation* in π . The crucial fact is proving that $\text{flow}(\pi)$ is *acyclic*. Our argument exploits sophisticated techniques borrowed from the theory of Linear Logic, and in particular from Girard’s Geometry of Interaction [Accattoli et al. 2020a, 2021a,b; Dal Lago et al. 2017; Girard 1989]. In order to convey more clearly the basic ideas, here we give the definitions only for the low-level fragment. Recall that this fragment suffices to type every BN normal form. The development for the full higher-order calculus is in the Appendix.

The Flow Graph of a Type Derivation is a DAG. Intuitively, the graph we are going to build tracks the occurrences of atomic types (*i.e.* the random variables) throughout the type derivation. We indicate a specific occurrence of an atom inside a ground type L by means of a (type) context, *i.e.* a type with a hole, as follows:

$$\text{GROUND TYPE CTXS } \underline{K}, \underline{L} ::= (\cdot) \mid \underline{K} \otimes L \mid K \otimes \underline{L}$$

So \underline{L} denotes an occurrence of atom (here X) inside the type $L \triangleq \underline{L}(X)$. Notice for example that the type $(X \otimes X) \otimes Y$ contains three occurrences of atoms. Given a type derivation π , we assume given a distinct label to each occurrence of atom appearing in each judgment of π . We call such a label a *position*. We can now build a graph that has all the positions of π as vertices, and that tracks the flow of each name X .

Definition 7.1 (Flow Graph). Let $\pi \triangleright \Lambda \vdash t : L$. The flow graph $\text{flow}(\pi)$ of π is the *directed graph* which has as *vertices* all the positions occurring in π , and *edges* as indicated in Fig. 10.

Example 7.2. We show the type derivation for the term (1) encoding our initial example, annotated with the flow graph. The reader can already notice that the flow exactly matches the DAG corresponding to the Bayesian network in Fig. 1.

$$\begin{array}{c}
 \frac{\frac{\frac{\frac{\vdash \text{bernoulli}_{0,6} : D}{d : D \vdash \text{c}^S(d) : S} \quad d : D \vdash \text{c}^R(d) : R}{d : D, s : S \vdash \text{let } r = \text{c}^R(d) \text{ in let } w = \text{c}^W(s,r) \text{ in } w : W} \quad s : S, r : R \vdash \text{c}^W(s,r) : W}{s : S, r : R \vdash \text{let } w = \text{c}^W(s,r) \text{ in } w : W} \quad w : W \vdash w : W}{d : D \vdash \text{let } s = \text{c}^S(d) \text{ in let } r = \text{c}^R(d) \text{ in let } w = \text{c}^W(s,r) \text{ in } w : W} \quad s : S, r : R \vdash \text{c}^W(s,r) : W}{\vdash \text{let } d = \text{bernoulli}_{0,6} \text{ in let } s = \text{c}^S(d) \text{ in let } r = \text{c}^R(d) \text{ in let } w = \text{c}^W(s,r) \text{ in } w : W}
 \end{array}$$

We will exploit the fact that the flow graph of a type derivation of ground type is acyclic.

PROPOSITION 7.3 (THE FLOW IS ACYCLIC). *Let $\pi \triangleright \Lambda \vdash t : L$. Then $\text{flow}(\pi)$ is a DAG.*

PROOF SKETCH. If t is in normal form, it is immediate to verify (by induction on the derivation π) that $\text{flow}(\pi)$ is *acyclic*. If t is not in normal form, by Thm. 4.6 we know that there exists a term u in normal form such that $t \rightarrow^* u$. It is then enough to prove that cycles are preserved along a reduction sequence, which we do by strengthening the subject reduction statement. \square

⁶Precisely, the joint distribution underlying t and \mathcal{B}_t is exactly the same. We then have to take into account that, in general, the term t encodes also a query, defining a *marginal* of the full joint distribution (Thm. 7.1).

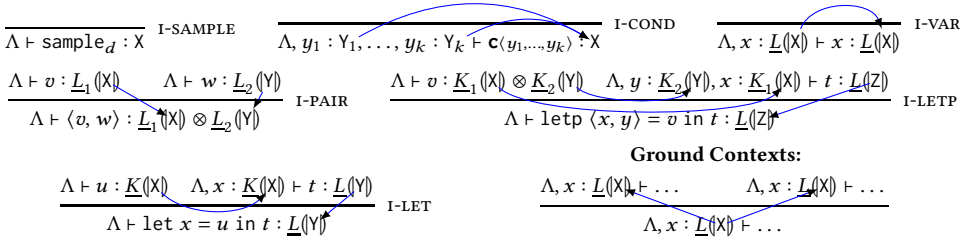


Fig. 10. Flow graph on the low-level fragment. The flow graph for the full calculus is defined in the Appendix.

7.1 Terms and Bayesian Networks: Soundness and Completeness

We can encode any Bayesian network \mathcal{B} with a low-level term $t_{\mathcal{B}}$, in the standard way; it is immediate to check that \mathcal{B} and $t_{\mathcal{B}}$ have the same semantics. Conversely, the flow graph gives us a way to extract a Bayesian network \mathcal{B}_t from every term t of ground type. Indeed, each probabilistic axiom corresponds to a random variable, and the dependencies between probabilistic axioms in the flow graph correspond to the edges in the Bayesian network. We have already illustrated this process in Example 7.2. More formally, we extract a Bayesian network as follows. For clarity, we focus on *closed* terms. If the term is not closed, what we would extract is a *conditional* Bayesian network.

Definition 7.4 (BN Extraction). Let t be a closed λ_1 -term of ground type. The derivation $\pi \triangleright t : L$ defines a Bayesian network $\mathcal{B}_t = (\mathcal{G}, \mathcal{T})$ over $\text{Nm}(\pi)$ where:

- \mathcal{T} maps each name $X \in \text{Nm}(\pi)$ to the factor which is associated to the axiom introducing X .
- \mathcal{G} is obtained from $\text{flow}(\pi)$ by collapsing all the vertices labeled by the same name.

This extraction process is correct, in the following sense (the proof is straightforward by unfolding the definition of $\llbracket \pi \rrbracket$ and $\llbracket \mathcal{B}_t \rrbracket$).

THEOREM 7.1 (FROM GROUND TYPE TERMS TO BNs). *Let $\pi \triangleright t : L$ be a derivation of ground type, and \mathcal{B}_t the Bayesian network associated to it, as in Def. 7.4. Then*

$$\llbracket \pi \rrbracket = \sum_{\text{Nm}(\pi) - \text{Nm}(L)} \llbracket \mathcal{B}_t \rrbracket.$$

7.2 From DAGs to Compositionality

We use the technology of the flow graph also to prove the fundamental result which we used to determine compositionality, namely that all derivations of ground type are well-formed (Lemma 6.6). The crucial property is the following, which is remarkably reminiscent of the characterizing property of jointrees, the data structure underlying the *message passing* algorithm for exact inference on BNs. The proof (obtained from the fact that the flow graph is a DAG) is in the Appendix.

LEMMA 7.5 (NAMED PATHS). *Let $\pi \triangleright \Lambda \vdash t : L$, and let X be the main name of a probabilistic axiom α^X . Then in $\text{flow}(\pi)$, each position with name X is connected to the occurrence of X in α^X by a path where all positions have name X .*

Then, we are finally able to give the proof of Lemma 6.6.

PROOF OF LEMMA 6.6. We prove that each rule in $\pi \triangleright \Lambda \vdash t : L$ has compatible premises. Let $\pi_1 \triangleright J_1, \dots, \pi_n \triangleright J_n$ be the premises of a rule in π . Assume there exists a π_j such that $X \in \text{Nm}(\pi_j)$ and $X \notin \text{Nm}(J_j)$. It is easy to see that X is necessarily the main name of a probabilistic axiom α^X in π_j . By Lemma 7.5, each position with name X in $\text{flow}(\pi)$ is connected to the occurrence of X in α^X by a path where all positions have name X . Since $X \notin \text{Nm}(J_j)$, it is impossible that $X \in \text{Nm}(\pi_i)$ for $i \neq j$. \square

8 DEALING WITH EVIDENCE, AKA COMPUTING THE POSTERIOR

Quoting [Gordon et al. 2014], inference is the task of “computing an explicit representation of the probability distribution implicitly specified by a probabilistic program”. So far, we have focused on the computation of marginals, without explicitly dealing with evidence. Admittedly, the most interesting inference problem is to compute a *posterior* distribution, such as $\Pr(\text{Rain}|\text{Wet} = \text{t})$. We know that this is the same as $\Pr(\text{Rain}, \text{Wet} = \text{t})/\Pr(\text{Wet} = \text{t})$ (as recalled in Sect. 2). In standard practice, to compute a posterior means to compute the unnormalized posterior

$$\Pr(\text{Rain}, \text{Wet} = \text{t}). \quad (6)$$

We can then normalize (6) by dividing for $\Pr(\text{Wet} = \text{t})$, which is also available—*for free*—from (6), just by adding up all the probabilities appearing in it. Indeed, $\Pr(\text{Wet} = \text{t}) = \sum_{\text{Rain}} \Pr(\text{Rain}, \text{Wet} = \text{t})$.

Technically, in the setting of BNs (and in our setting), there is no essential difference between computing a marginal like $\Pr(\text{Rain}, \text{Wet} = \text{t})$ or a marginal like $\Pr(\text{Rain}, \text{Wet})$. The key to deal with evidence (such as $\text{Wet} = \text{t}$) is to update each CPT in the model, by zeroing out those rows that are inconsistent with the observed data. The resulting factors are no longer normalized, but are still valid factors, and their product gives the unnormalized posterior distribution. We illustrate this with an example, referring to [Darwiche 2009] (Ch. 6.7) or [Koller and Friedman 2009] (Ch. 9.3.2) for the details.

Example 8.1 (Basic). Consider a simple BN of graph $\text{Rain} \rightarrow \text{Wet}$. The factors associated to Rain and Wet are respectively ϕ_1 and ϕ_2 , as given in Fig. 8. We know (Sect. 5.3) that $\Pr(\text{Rain}, \text{Wet}) = \phi_1 \odot \phi_2$. Assume we have *evidence* $e \triangleq (\text{Wet} = \text{t})$. The (unnormalized) posterior distribution $\Pr(\text{Rain}, \text{Wet} = \text{t})$ is again the product of the factors associated to the nodes of the BN (as in Thm. 5.10) but this time starting from the updated factors ϕ_1^e and ϕ_2^e . In our case, $\phi_1^e = \phi_1$, while ϕ_2^e is modified as in Fig. 11. So $\Pr(\text{Rain}, \text{Wet} = \text{t}) = \phi_2^e \odot \phi_1^e$ (the result is in Fig. 11). Hence, we

$$\phi_2^e = \begin{array}{|c|c|c|} \hline \text{R} & \text{W} & \Pr(\text{W} = \text{t}|\text{R}) \\ \hline \text{t} & \text{t} & 0.7 \\ \hline \text{f} & \text{t} & 0.01 \\ \hline \end{array} \quad \phi_2^e \odot \phi_1^e = \begin{array}{|c|c|c|} \hline \text{R} & \text{W} & \Pr(\text{R}, \text{W} = \text{t}) \\ \hline \text{t} & \text{t} & 0.14 \\ \hline \text{f} & \text{t} & 0.008 \\ \hline \end{array} \quad \frac{\phi_2^e \odot \phi_1^e}{0.148} = \begin{array}{|c|c|c|} \hline \text{R} & \text{W} & \Pr(\text{R}|\text{W} = \text{t}) \\ \hline \text{t} & \text{t} & 0.14/0.148 = 0.946 \\ \hline \text{f} & \text{t} & 0.008/0.148 = 0.054 \\ \hline \end{array}$$

Fig. 11. Dealing with evidence

immediately have:

- $\Pr(\text{Wet} = \text{t}) = 0.148$, which is obtained by adding up all the entries in $\Pr(\text{Rain}, \text{Wet} = \text{t})$;
- $\Pr(\text{Rain}|\text{Wet} = \text{t})$, which is obtained by normalizing $\Pr(\text{Rain}, \text{Wet} = \text{t})$, as depicted in Fig. 11.

So, for example, we have the posterior $\Pr(\text{Rain} = \text{t}|\text{Wet} = \text{t}) = 0.14/0.148 = 0.946$, a huge increase from our prior belief $\Pr(\text{Rain} = \text{t}) \triangleq \phi_1(\text{t}) = 0.2$.

8.1 The Semantics of Terms, with Evidence

The main ingredients being the same, our approach is easily adapted to deal with evidence, syntactically encoded by the construct $\text{obs}(x = b)$. To express evidence, we enrich the definition of intersection types with two more sets of atomic types, namely $\{X^t \mid X \in \text{Names}\}$ and $\{X^f \mid X \in \text{Names}\}$.

$$\begin{array}{l} \text{ATOMIC TYPES} \quad \mathcal{X} ::= X \mid X^t \mid X^f \\ \text{GROUND TYPES} \quad K, L ::= X \mid K \otimes L \end{array} \quad \left| \quad \begin{array}{l} \text{POSITIVE TYPES} \quad P, Q ::= L \mid [A_1, \dots, A_n] \\ \text{TYPES} \quad A, B ::= P \mid P \multimap A \end{array} \right.$$

Technically, we require the atomic types appearing in a type derivation to be pairwise consistent, meaning that if two atomic types X_1, X_2 have the same name X , then they are both either X or X^t

or X^f . It is easy to check that for this property to hold for a type derivation $\pi \triangleright J$, it just suffices to require that it holds for the atomic types in the conclusion J .

Given a derivation π and a name X , we define $\text{Val}(X) = \{t\}$ if X occurs in π as X^t , $\text{Val}(X) = \{f\}$ if X occurs in π as X^f , and $\text{Val}(X) = \{t, f\}$ otherwise. Essentially, we treat an observed r.v. as unary, akin to factor reduction in [Koller and Friedman 2009].

Types and Semantics. We can now complete Fig. 7 (and Fig. 9) with the typing rule for $\text{obs}(x = b)$:

$$\frac{}{\Lambda, x : X^b \vdash \text{obs}(x = b) : X^b \diamond \mathbf{1}^0} \text{I-OBS}$$

where we annotated the rule with the semantics indicated by Def. 6.2 (a remark similar to 6.3 applies). The typing rules for I-SAMPLE and I-COND in Fig. 7 (and Fig. 9) are updated as follows:

$$\frac{X \notin \text{Nm}(\Lambda)}{\Lambda \vdash \text{sample}_d : X \diamond \phi^{\{X\}}} \text{I-SAMPLE} \quad \frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : \mathcal{Y}_1, \dots, y_n : \mathcal{Y}_n \vdash \mathbf{c}\langle y_1, \dots, y_n \rangle : X \diamond \phi^{\{Y_1, \dots, Y_n, X\}}} \text{I-COND}$$

where we annotate the rules with the factor ϕ indicated by Def. 6.1, taking into account the definition of $\text{Val}(X)$ given above (see Example 8.2). Everything else stays the same: all definitions and results in Sections 6 and 7 remain valid. An example will clarify what happens, and how the evidence is reflected into the semantics, via the type derivation.

Example 8.2 (Basic). Consider a term modeling the BN of Example 8.1, i.e. $\text{Rain} \rightarrow \text{Wet}$:

$$\text{let rain} = \text{bernoulli}_{0.2} \text{ in let wet} = \mathbf{c}\langle \text{rain} \rangle \text{ in } \langle \text{rain}, \text{wet} \rangle : R \otimes W.$$

where $\mathbf{c}\langle \text{rain} \rangle \triangleq \text{case rain of } \{t \Rightarrow \text{bernoulli}_{0.7}, f \Rightarrow \text{bernoulli}_{0.01}\}$. To express the evidence $W = t$, we use the construct $\text{obs}(\text{wet} = t)$, yielding the (sugared) term

$$\text{let rain} = \text{bernoulli}_{0.2} \text{ in let wet} = \mathbf{c}\langle \text{rain} \rangle \text{ in } \langle \text{rain}, \text{obs}(\text{wet} = t) \rangle : R \otimes W^t$$

Let us see how the evidence is reflected into the semantics, thanks to the type derivation π .

$$\frac{\frac{\frac{}{r : R \vdash r : R \diamond \mathbf{1}_0} \text{I-COND} \quad \frac{}{w : W^t \vdash \text{obs}(w = t) : W^t \diamond \mathbf{1}_0} \text{I-OBS}}{r : R, w : W^t \vdash \langle r, \text{obs}(w = t) \rangle : R \otimes W^t \diamond \mathbf{1}_0} \text{I-LET}}{\vdash \text{bern}_{0.2} : R \diamond \phi_1^{\{R\}} \quad \frac{}{r : R \vdash \mathbf{c}\langle r \rangle : W^t \diamond \psi^{\{R, W\}}} \text{I-COND}}{\pi \triangleright \text{let } r = \text{bern}_{0.2} \text{ in let } w = \mathbf{c}\langle r \rangle \text{ in } \langle r, \text{obs}(w = t) \rangle : R \otimes W^t \diamond \phi_1 \diamond \psi} \text{I-LET}$$

The I-OBS axiom forces the type W^t . In turn, W^t is propagated to the I-COND axiom, via the I-LET rule. What is the semantics of the I-COND axiom? Spelling out Def. 6.1, and recalling that here $\text{Val}(W) = \{t\}$, we have that the factor ψ associates a scalar to each of the *two tuples* in $\text{Val}(R) \times \text{Val}(W)$:

$$\psi(tt) \triangleq \text{bernoulli}_{0.7}(t) \text{ and } \psi(ft) \triangleq \text{bernoulli}_{0.01}(t).$$

That is, ψ is exactly the factor ϕ_2^e in Fig. 11. Please observe that the difference in the interpretation is entirely due to the type W^t . With the same definitions as in Sect. 6, we have $\llbracket \pi \rrbracket = \phi_1 \diamond \psi$, which is exactly $\phi_1^e \diamond \phi_2^e = \text{Pr}(\text{Rain}, \text{Wet} = t)$.

A very similar reasoning would apply to express $\text{Pr}(\text{Rain} | \text{Wet} = t)$ in a model with several variables, like the one in Fig. 1; marginalization is taken care by the semantics exactly as before. It is more interesting to revisit Example 4.10, to illustrate how we deal with evidence together with *copies*.

Example 8.3 (Coin Tosses). Consider again the term u of Example 4.10, modeling two tosses of the same (biased) coin. Assume we want to *infer (learn) the bias of the coin* from the result of the tosses. For example, if we observe that both the coin tosses yield t , this would increase our confidence that the coin is biased toward t . Indeed the semantics of the updated term changes in this

sense, as we show here. The following term⁷ u' expresses the evidence, given the model encoded by u . Notice the *modularity* in the encoding.

$$u' \triangleq \text{letp } \langle x, y_1, y_2 \rangle = u \text{ in } \langle x, \text{obs}(y_1 = t), \text{obs}(y_2 = t) \rangle$$

Once again, each observation $\text{obs}(y_i = t)$ forces the type Y_i^t , so recording the evidence t for each Y_i ; the type derivation takes care of propagating the evidence, as shown below:

$$\frac{\frac{\frac{x : X \vdash x : X}{\pi' \triangleright u : X \otimes Y_1^t \otimes Y_2^t} \quad \frac{\frac{y_1 : Y_1^t \vdash \text{obs}(y_1 = t) : Y_1^t \quad y_2 : Y_2^t \vdash \text{obs}(y_2 = t) : Y_2^t}{y_1 : Y_1^t, y_2 : Y_2^t \vdash \langle \text{obs}(y_1 = t), \text{obs}(y_2 = t) \rangle : Y_1^t \otimes Y_2^t}}{x : X, y_1 : Y_1^t, y_2 : Y_2^t \vdash \langle x, \text{obs}(y_1 = t), \text{obs}(y_2 = t) \rangle : X \otimes Y_1^t \otimes Y_2^t}}{\rho \triangleright \text{letp } \langle x, y_1, y_2 \rangle = u \text{ in } \langle x, \text{obs}(y_1 = t), \text{obs}(y_2 = t) \rangle : X \otimes Y_1^t \otimes Y_2^t} \text{I-LETP}}$$

The derivation $\pi' \triangleright u$ (highlighted in red) is the same as the derivation $\pi \triangleright u$ of Example 4.10, but with each type Y_i replaced by Y_i^t (as forced by the I-LETP typing rule). The semantics changes as expected, reflecting that the evidence increases our confidence that the coin is biased toward t . The reader can find the computation of the semantics developed in full in the Appendix.

9 A COST-AWARE SEMANTICS

In this section we examine the cost of computing the semantics of a term t of ground type, that is the cost of inferring the marginal distribution defined by t . We assume π to be an arbitrary derivation of a judgment J of shape $\Lambda \vdash t : L$. Looking at Def. 6.2 one easily realizes the following:

PROPOSITION 9.1 (COST UPPER BOUND). *The cost of computing $\llbracket \pi \rrbracket$ according to Def. 6.2 is $O(m_\pi \cdot 2^n)$, where $n = |\text{Nm}(\text{Cpts}(\pi))|$ is the number of names which appear in $\text{Cpts}(\pi)$, and $m_\pi \leq n$ is the number of probabilistic axioms in π .*

This is the *upper bound* to the cost of computing the semantics of π . But there is more to the story. Non-idempotent type systems are known to provide quantitative information about the typed term, such as bounds on the execution time. There is a rich literature on systems which capture (possibly tight) complexity bounds in different styles of computation [Accattoli et al. 2022, 2020b; de Carvalho 2018], including *the expected runtime* of probabilistic computations [Dal Lago et al. 2021]. However, in the setting of Bayesian modeling and exact inference, the runtime, and even the expected runtime, has little relevance, because the *dominating cost* (in time and space) is due to the computation of the probability distribution. Such a cost is what interests us—our type system is able to provide accurate bounds.

On the Cost of Inductively Computing $\llbracket \pi \rrbracket$. Let us point out the differences, *computationally*, between the definition of $\llbracket \pi \triangleright J \rrbracket$ according to Def. 6.2—which only considers the probabilistic axioms and the names occurring in the conclusion J —and the inductive definition illustrated in Fig. 9. The latter, which computes the semantics following the structure of the type derivation, allows for a *more efficient* computation, in general. This is because summing out step-by-step yields intermediate factors of smaller size. This indeed is exactly the way algorithms for exact inference work.

Example 9.2. Consider a Bayesian network whose DAG is a chain $X_1 \rightarrow X_2 \rightarrow \dots \rightarrow X_n$. The following term defines the marginal over the single variable X_n .

$$\pi \triangleright \text{let } x_1 = \text{sample}_d \text{ in let } x_2 = \mathbf{c}^2_{\langle x_1 \rangle} \text{ in } \dots \text{ let } x_n = \mathbf{c}^n_{\langle x_{n-1} \rangle} \text{ in } x_n : X_n$$

⁷As usual, we use some syntactic sugar.

- Computing $\llbracket \pi \rrbracket$ according to Def. 6.2 has cost $O(n \cdot 2^n)$. Indeed, we first compute the product of n factors—that is *the full joint distribution* over X_1, \dots, X_n —and then sum out all the variables but X_n .
- Regardless of the number of variables in the chain, the cost of computing the semantics *following the structure of the derivation* is $O(n \cdot 2^3)$, as one can easily verify by trying to write down the derivation: we do not actually need to explicitly build a distribution over n random variables.

Clearly, there is no guarantee that the cost of inductively computing the semantics of π is always *strictly less* than computing $\llbracket \pi \rrbracket$ directly.

If we examine the cost of computing the semantics of π inductively, we obtain a better upper bound than in Prop. 9.1—please observe that the latter essentially corresponds to computing the full joint distribution underlying the model.

PROPOSITION 9.3 (INDUCTIVE COST). *Let π be a type derivation, m_π the number of probabilistic axioms in π , and $n = |\text{Nm}(\text{Cpts}(\pi))|$ the number of names which appear in $\text{Cpts}(\pi)$ (as in Prop. 9.1). The cost of inductively computing the semantics of π following the structure of the derivation is*

$$O(m_\pi \cdot 2^W)$$

where $W \leq n$ is the maximal cardinality of any set of names \mathbb{Y} appearing in the derivation, when decorated as in Fig. 9.

Remark 9.4. The reader familiar with inference algorithms will realize that inductively computing the semantics following the structure of the derivation implements inference by a form of the Variable Elimination algorithm, and indeed such an observation can be made formal. The needed technical details, however, are beyond the scope of this paper.

Remark 9.5 (Observed Variables). Both bounds—in Prop. 9.1 and Prop. 9.3— can be made tighter by restricting n to the set $\hat{\mathbb{Y}}$ of non observed names. The set \mathbb{E} of observed variables does not actually contribute to the number of entries in a factor, since each one has a single possible value. So $\text{Val}(\hat{\mathbb{Y}}) \times \text{Val}(\mathbb{E})$ is isomorphic to $\text{Val}(\hat{\mathbb{Y}})$.

Exact Bounds. Given that the type system is resource-sensitive by design, it is not difficult to refine type derivations by decorating π with the number of elementary steps needed to compute its semantics. We stress that this information can be retrieved *without* performing calculations on factors—it is enough to keep track of the relevant names occurring in each judgment. It is straightforward to fine-tune the complexity measure and provide bounds with different granularity.

In Fig. 12 we give a minimalist example, limited to the *decidable* first-order system, where the weight on the turnstile counts the number of multiplications performed during the computation (the cost of sums being negligible w.r.t. the cost of products). Here, given a judgment $J \diamond \mathbb{X}$, we write $\hat{\mathbb{X}}$ for the restriction of \mathbb{X} to the names that are not observed in J : this is to take into account that observed names do not actually contribute to the factor size, hence to the overall cost, as remarked above.

Same Model, Different Cost. Since the cost of computing the semantics (*i.e.* the cost of exact inference) depends on the structure of the term itself, different terms describing the very same marginal distribution may have different costs. Indeed, a term encodes not only a Bayesian network and a marginal, but also a way to compute it, in agreement with the ideas which underlay exact inference *on terms*, as proposed by Koller and Pfeffer [1997]. Our type system detects such a difference. The following example illustrates this point.

Example 9.6 (Same BN, Different Cost). Consider the following two terms, both in normal form, and both corresponding to the same Bayesian network whose graph is the chain $X \rightarrow Y \rightarrow Z$.

$$\begin{array}{c}
\frac{X \notin \text{Nm}(\Lambda)}{\Lambda \vdash \text{sample}_d : \mathcal{X} \diamond \{X\}} \text{ I-SAMPLE} \quad \frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : \mathcal{Y}_1, \dots, y_n : \mathcal{Y}_n \vdash \mathbf{c}\langle y_1, \dots, y_n \rangle : \mathcal{X} \diamond \{Y_1, \dots, Y_n, X\}} \text{ I-COND} \\
\frac{}{\Lambda, x : P \vdash x : P \diamond \emptyset} \text{ I-VAR} \quad \frac{}{\Lambda, x : \mathcal{X}^b \vdash \text{obs}(x = b) : \mathcal{X}^b \diamond \emptyset} \text{ I-OBS} \\
\frac{\Lambda \vdash^{n_1} u : P \diamond Y_1 \quad \Lambda, x : P \vdash^{n_2} t : A \diamond Y_2 \quad Z = (Y_1 \cup Y_2) - \text{Nm}(\Lambda, A)}{\Lambda \vdash^{n_1+n_2+m} \text{let } x = u \text{ in } t : A \diamond (Y_1 \cup Y_2) - Z} \text{ I-LET} \quad m = \begin{cases} 2^{|\widehat{Y}_1 \cup \widehat{Y}_2|} & \text{if } Y_1, Y_2 \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \\
\frac{\Lambda \vdash^0 v : L_1 \diamond \emptyset \quad \Lambda \vdash^0 w : L_2 \diamond \emptyset}{\Lambda \vdash^0 \langle v, w \rangle : L_1 \otimes L_2 \diamond \emptyset} \text{ I-PAIR} \quad \frac{\Lambda \vdash^0 v : L_1 \otimes L_2 \diamond \emptyset \quad \Lambda, x : L_1, y : L_2 \vdash^n t : A \diamond Y}{\Lambda \vdash^n \text{letp } \langle x, y \rangle = v \text{ in } t : A \diamond Y} \text{ I-LETP}
\end{array}$$

Fig. 12. First-order type system annotated with the *cost* of computing the factor.

1. $t_1 \triangleq \text{let } x = \text{sample}_d \text{ in let } y = \mathbf{c}^Y_{\langle x \rangle} \text{ in let } z = \mathbf{c}^Z_{\langle y \rangle} \text{ in } z : Z$
2. $t_2 \triangleq \text{let } z = (\text{let } y = (\text{let } x = \text{sample}_d \text{ in } \mathbf{c}^Y_{\langle x \rangle}) \text{ in } \mathbf{c}^Z_{\langle y \rangle}) \text{ in } z : Z$

Both terms define the same marginal distribution over Z . However, inductively computing such a distribution has a different *cost*⁸: 12 multiplications for t_1 , and 8 for t_2 .

Cost and Reduction. The upper bound in Prop. 9.1 is invariant by reduction, because the number of probabilistic axioms in a type derivation is invariant. On the other hand, the cost of *inductively* computing the semantics of a type derivation π (Prop. 9.3) is *not* stable by reduction, because the structure of the derivation changes, and so the size W of the largest factor to be inductively computed may grow or shrink. This is indeed the rationale behind the program transformations which correspond to the Variable Elimination algorithm performed on terms [Ehrhard et al. 2023b]. Starting from a normal form, the efficiency may be improved via expansion (the reverse of reduction).

Cost of Factors Product vs. Matrices Product. We stress how much the product of factors *differs* from the product of matrices. In this difference lies the efficiency of a factors-based semantics w.r.t. to a categorical [Jacobs and Zanasi 2020] or relational [Ehrhard et al. 2014; Ehrhard and Tasson 2019] one, where a central role is played by a product \otimes which behaves as the tensor product of matrices.

Example 9.7. Let t_1, t_2 be case expressions, respectively encoding two CPTs $\phi^{X_1} = \text{Pr}(X_1 | Y_1, Y_2, Y_3)$ and $\phi^{X_2} = \text{Pr}(X_2 | Y_1, Y_2, Y_3)$, where two distinct r.v.s (X_1 and X_2 , respectively) are conditioned to the *same* set of r.v.s Y_1, Y_2, Y_3 . We can see each ϕ^{X_i} interpreting t_i as a stochastic matrix (of size 2^4). One easily realizes that computing the tensor product $\phi^{X_1} \otimes \phi^{X_2}$ of the two matrices, requires to *compute and store* $2^4 \cdot 2^4 = 2^8$ entries. In contrast, the factor product $\phi^{X_1} \odot \phi^{X_2}$ computes 2^5 entries. Indeed, in a categorical or relational model, to compute the semantics of the term $\langle t_1, t_2 \rangle$ will (in general) pass via $\phi^{X_1} \otimes \phi^{X_2}$. On this basis, it is easy to build a term which encodes a BN over the 5 variables X_1, X_2, Y_1, Y_2, Y_3 , and whose inductive interpretation (in a categorical or relational model) requires to compute and store 2^8 values—one such a term is given in the Appendix. This is somehow weird—from the point of view of BNs—given that the *full* joint distribution over 5 variables has size 2^5 .

⁸The reader can find the explicit computations in the Appendix.

$$\begin{array}{c}
\frac{X \notin \text{Nm}(\Lambda)}{\Lambda \vdash \text{sample}_d : \mathbf{B} : \mathbf{X}} \text{ P-SAMPLE} \quad \frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : \mathbf{B} : Y_1, \dots, y_n : \mathbf{B} : Y_n \vdash \mathbf{c}\langle y_1, \dots, y_n \rangle : \mathbf{B} : \mathbf{X}} \text{ P-COND} \\
\\
\frac{\Lambda, \Gamma_1 \vdash u : \mathbf{P} : \mathbf{Q} \quad \Lambda, \Gamma_2, x : \mathbf{P} : \mathbf{Q} \vdash t : \mathbf{A} : \mathbf{C}}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \text{let } x = u \text{ in } t : \mathbf{A} : \mathbf{C}} \text{ P-LET} \\
\\
\frac{}{\Lambda \vdash \underline{n} : \mathbf{N} : \bar{n}} \quad \frac{\Lambda, \Gamma_1 \vdash v : \mathbf{N} : \bar{0} \quad \Lambda, \Gamma_2 \vdash t : \mathbf{A} : \mathbf{C}}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \text{ifZero } v \text{ t } u : \mathbf{A} : \mathbf{C}} \quad \frac{\Lambda, \Gamma_1 \vdash v : \mathbf{N} : \bar{n+1} \quad \Lambda, \Gamma_2 \vdash u : \mathbf{A} : \mathbf{C}}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \text{ifZero } v \text{ t } u : \mathbf{A} : \mathbf{C}} \\
\\
\frac{\Lambda, \Gamma_0, x : \mathbf{!A} : [\mathbf{C}_1, \dots, \mathbf{C}_n] \vdash t : \mathbf{A} : \mathbf{C} \quad (\Lambda, \Gamma_i \vdash \text{fix } x.t : \mathbf{A} : \mathbf{C}_i)_{i=1}^n}{\Lambda, \uplus_{i=0}^n \Gamma_i \vdash \text{fix } x.t : \mathbf{A} : \mathbf{C}}
\end{array}$$

Fig. 13. The type system for call-by-push-value PCF (selected rules).

10 LIFTING THE TYPE SYSTEM TO CALL-BY-PUSH-VALUE PCF

To streamline the presentation, we carried out our analysis in the setting of the untyped λ -calculus, on top of which we have defined an intersection type system. As mentioned in Sect. 4.2, all our methods can be lifted to a more user friendly PCF-like syntax, similar to that in [Ehrhard and Tasson 2019]. Terms of a call-by-push-value PCF are those of the λ -calculus in Sect. 3 plus some operations needed to handle natural numbers (including a constant \underline{n} for each number n) and a fixed point combinator:

$$\begin{array}{l}
\text{TERMS } t, u ::= \dots \mid \text{succ } v \mid \text{pred } v \mid \text{ifZero } v \text{ t } u \mid \text{fix } x.t \\
\text{VALUES } v, w ::= \dots \mid \underline{n}
\end{array}$$

The reduction rules are as expected. As standard, PCF types (represented in red) are the simple types of Sect. 3 extended with the ground type of natural numbers:

$$\text{GROUND TYPES } L, K ::= \dots \mid \mathbf{N}$$

The fact that PCF is a *typed* calculus does *not* mean that we can get rid of intersection types (here represented in blue) to give semantics to the terms. Indeed, we still need to keep track of the names which are generated during the computation. As usual with PCF, ground *intersection types* need to be extended with a type constant \bar{n} for each natural number n :

$$\text{GROUND INTERSECTION TYPES } L, K ::= \dots \mid \bar{n}$$

At this point we need to type PCF terms. A PCF term will come with *two* types: a standard type, and an intersection type, as in [Dal Lago and Gaboardi 2011; Ehrhard 2016; Ehrhard et al. 2014]. For example, the term sample_d is typed as $\vdash \text{sample}_d : \mathbf{B} : \mathbf{X}$. The idea is that in a judgment $\vdash t : \mathbf{A} : \mathbf{C}$, the intersection type \mathbf{C} refines the standard PCF type \mathbf{A} . In the example, \mathbf{X} refines \mathbf{B} , being \mathbf{X} the name of a boolean variable. In Figure 13, we report a selection of the typing rules for this language. One can notice that the rules P-SAMPLE, P-COND, and P-LET are obtained by overlapping the ones for simple types with the ones for intersection types. The typing rules which are specific to PCF constructs are standard, and have been adapted from [Ehrhard 2016].

11 RELATED WORK

Theoretical research on functional PPLs, pioneered by [Saheb-Djahromi 1978] is a very active area. Early investigation [Friedman et al. 1998; Koller et al. 1997; Park 2003; Park et al. 2005; Pless and Luger 2001; Ramsey and Pfeffer 2002] has evolved in a large body of work, to support the growing development of software. Landmark foundational work aiming at providing sound and compositional methods for *higher-order* Bayesian inference includes [Borgström et al. 2015] and [Ścibior et al.

2018]. The former is based on operational techniques, while the latter on a modular denotational semantics. Our work is inspired by both. The importance of capturing the data flow of *higher-order* probabilistic programs, is stressed in [Castellan and Paquet 2019; Paquet 2021], which rely on event structures. Relevantly, Bayesian networks are at the core of Paquet [2021] game semantics proposal, motivated by the fact that—in practice—most modern inference engines do not manipulate directly the program syntax but rather some graph representation of it. We share this view, and the goal of providing reasoning tools for such implementations. We also mention [Gorinova et al. 2022], which introduces an information flow type system, in a *first-order*, imperative setting. As already recalled in the introduction, the theory of Bayesian networks has been investigated extensively from a categorical viewpoint by Jacobs and Zanasi [Jacobs 2023; Jacobs et al. 2019; Jacobs and Zanasi 2016, 2020].

The foundational studies based on Bayesian networks which we have cited above focus on expressiveness and compositionality. However, they do not take into account space and time consumption of probabilistic reasoning, which is the very motivation for the introduction [Pearl 1986] and development of BNs. As a matter of fact, the cost of *actually computing the semantics* explodes when taking a categorical [Jacobs and Zanasi 2020] or relational [Ehrhard et al. 2014; Ehrhard and Tasson 2019] approach, because of the product \otimes , which behaves as the tensor product of matrices (see Example 9.7); inductively computing the semantics of n binary r.v.s easily leads to intermediate computations whose size is much larger than 2^n (the size of the full joint distribution). This fact has already been pointed out in a recent paper by Ehrhard et al. [2023a], which advocates the need for a new approach to quantitative semantics, more attentive to the resource consumption. That paper imports factors techniques into the setting of *multiplicative linear logic*, essentially a *linear* λ -calculus with tuples (roughly, our first-order fragment)—the authors leave as an open challenge the treatment of linear logic exponentials (roughly, the calculus of Fig. 6). Our framework, indeed, is able to deal with a fully-fledged λ -calculus, thanks to the intersection type system and the flow-graph techniques built on top of it. Actually, our factor-based semantics can be seen as an optimized version of semantics based on the weighted relational model, such as [Laird et al. 2013] and Probabilistic Coherence Space [Danos and Ehrhard 2011; Danos and Harmer 2002; Ehrhard et al. 2014; Ehrhard and Tasson 2019]. Finally, we mention that Chiang et al. [2023] also use techniques inspired from linear logic to provide a denotational semantics and exact inference procedures to recursive probabilistic programs.

The idea that programming languages provide a way to overcome the limitations of standard Bayesian networks has been actively propounded by Koller and Pfeffer [1997]; Pfeffer and Koller [2000], which introduced a modeling framework based on Object-Oriented programming. This approach has eventually led to the object-oriented language Figaro [Pfeffer 2016].

12 CONCLUSIONS

We have presented a *higher-order* probabilistic programming language which allows for the specification of recursive probability models and hierarchical structures. Higher-order programs are *compiled into standard Bayesian networks* operationally, via rewriting, and denotationally, via an intersection type system. The novelty of our contribution is that (1) the compositional semantics is based on *factors*, the very mathematical notion which is used to give semantics to Bayesian networks, and which is the basis of *exact* inference algorithms, and (2) our semantics and type system are *resource-sensitive*. The notion of *resource* appears here in two forms: (1) we precisely track generation and sharing of *random variables*, and (2) we account for the actual *cost* of inference—the cost of computing the semantics of a typed term. We obtain these quantitative results relying on advanced semantic techniques rooted into linear logic, intersection types, rewriting theory, and Girard’s geometry of interaction, which are here combined in a novel way.

The fact that our semantics is factor-based implies that standard algorithms for exact inference (which are acting on factors) can be applied. A further natural direction is to investigate inference techniques based on term transformations, in the spirit of [Ehrhard et al. 2023b; Koller et al. 1997].

ACKNOWLEDGMENTS

The authors are in debt with Thomas Ehrhard and Michele Pagani for many insightful discussions. We thank Ugo Dal Lago, Beniamino Accattoli, Delia Kesner for their useful remarks. We are also grateful to the anonymous referees, whose valuable comments have improved the presentation of this work. This research was supported by the ANR project PPS: ANR-19-CE48-0014. The third author is also supported by the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101034255.

DATA AVAILABILITY STATEMENT

Missing proofs and more examples are available in the Appendix. Please notice that while in the paper we postpone to Sect. 8 the treatment of types and semantics for the observe construct, the proofs in the Appendix directly integrate it.

REFERENCES

- Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2020a. The Machinery of Interaction. In *Proceedings of the 22nd International Symposium on Principles and Practice of Declarative Programming (Bologna, Italy) (PPDP ’20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 15 pages. <https://doi.org/10.1145/3414080.3414108>
- Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2021a. The (In)Efficiency of Interaction. *Proc. ACM Program. Lang.* 5, POPL, Article 51 (Jan. 2021), 33 pages. <https://doi.org/10.1145/3434332>
- Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2021b. The Space of Interaction. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 1–13. <https://doi.org/10.1109/LICS52264.2021.9470726>
- Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. 2022. Multi types and reasonable space. *Proc. ACM Program. Lang.* 6, ICFP (2022), 799–825. <https://doi.org/10.1145/3547650>
- Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. 2020b. Tight typings and split bounds, fully developed. *J. Funct. Program.* 30 (2020), e14. <https://doi.org/10.1017/S095679682000012X>
- Beniamino Accattoli and Delia Kesner. 2010. The Structural λ -Calculus. In *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6247)*, Anuj Dawar and Helmut Veith (Eds.). Springer, 381–395. https://doi.org/10.1007/978-3-642-15205-4_30
- Victor Arrial, Giulio Guerrieri, and Delia Kesner. 2023. Quantitative Inhabitation for Different Lambda Calculi in a Unifying Framework. *Proc. ACM Program. Lang.* 7, POPL (2023), 1483–1513. <https://doi.org/10.1145/3571244>
- P. N. Benton, Gavin M. Bierman, Valeria de Paiva, and Martin Hyland. 1993. A Term Calculus for Intuitionistic Linear Logic. In *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA ’93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings (Lecture Notes in Computer Science, Vol. 664)*, Marc Bezem and Jan Friso Groote (Eds.). Springer, 75–90. <https://doi.org/10.1007/BFb0037099>
- P. N. Benton and Philip Wadler. 1996. Linear Logic, Monads and the Lambda Calculus. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*. IEEE Computer Society, 420–431. <https://doi.org/10.1109/LICS.1996.561458>
- Johannes Borgström, Ugo Dal Lago, Andrew D. Gordon, and Marcin Szymczak. 2015. A Lambda-Calculus Foundation for Universal Probabilistic Programming. *CoRR abs/1512.08990* (2015). arXiv:1512.08990 <http://arxiv.org/abs/1512.08990>
- Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. 2020. The Bang Calculus Revisited. In *Functional and Logic Programming - 15th International Symposium, FLOPS 2020, Akita, Japan, September 14-16, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12073)*, Keisuke Nakano and Konstantinos Sagonas (Eds.). Springer, 13–32. https://doi.org/10.1007/978-3-030-59025-3_2
- Wray L. Buntine. 1994. Operations for Learning with Graphical Models. *J. Artif. Int. Res.* 2, 1 (dec 1994), 159–225. <https://doi.org/10.1613/jair.62>
- Simon Castellan and Hugo Paquet. 2019. Probabilistic Programming Inference via Intensional Semantics. In *Programming Languages and Systems - 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11423)*, Luís Caires (Ed.). Springer, 322–349. https://doi.org/10.1007/978-3-030-17184-1_12

- David Chiang, Colin McDonald, and Chung-chieh Shan. 2023. Exact Recursive Probabilistic Programming. *Proc. ACM Program. Lang.* 7, OOPSLA1 (2023), 665–695. <https://doi.org/10.1145/3586050>
- Mario Coppo and Mariangiola Dezani-Ciancaglini. 1978. A new type assignment for lambda-terms. *Archiv für mathematische Logik und Grundlagenforschung* 19, 1 (1978), 139–156. <https://doi.org/10.1007/BF02011875>
- Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. 1981. Functional Characters of Solvable Terms. *Math. Log.* Q. 27, 2-6 (1981), 45–58. <https://doi.org/10.1002/malq.19810270205>
- Fredrik Dahlqvist, Alexandra Silva, Vincent Danos, and Ilias Garnier. 2018. Borel Kernels and their Approximation, Categorically. In *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018, Dalhousie University, Halifax, Canada, June 6-9, 2018 (Electronic Notes in Theoretical Computer Science, Vol. 341)*, Sam Staton (Ed.). Elsevier, 91–119. <https://doi.org/10.1016/j.entcs.2018.11.006>
- Ugo Dal Lago, Claudia Faggian, and Simona Ronchi Della Rocca. 2021. Intersection types and (positive) almost-sure termination. *Proc. ACM Program. Lang.* 5, POPL (2021), 1–32. <https://doi.org/10.1145/3434313>
- Ugo Dal Lago, Claudia Faggian, Benoît Valiron, and Akira Yoshimizu. 2017. The geometry of parallelism: classical, probabilistic, and quantum effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, Giuseppe Castagna and Andrew D. Gordon (Eds.). ACM, 833–845. <https://doi.org/10.1145/3009837.3009859>
- Ugo Dal Lago and Marco Gaboardi. 2011. Linear Dependent Types and Relative Completeness. *Log. Methods Comput. Sci.* 8, 4 (2011). [https://doi.org/10.2168/LMCS-8\(4:11\)2012](https://doi.org/10.2168/LMCS-8(4:11)2012)
- Vincent Danos and Thomas Ehrhard. 2011. Probabilistic coherence spaces as a model of higher-order probabilistic computation. *Inf. Comput.* 209, 6 (2011), 966–991. <https://doi.org/10.1016/j.ic.2011.02.001>
- Vincent Danos and Russell Harmer. 2002. Probabilistic game semantics. *ACM Trans. Comput. Log.* 3, 3 (2002), 359–382. <https://doi.org/10.1145/507382.507385>
- Adnan Darwiche. 2008. Bayesian Networks. In *Handbook of Knowledge Representation*, Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter (Eds.). Foundations of Artificial Intelligence, Vol. 3. Elsevier, 467–509. [https://doi.org/10.1016/S1574-6526\(07\)03011-8](https://doi.org/10.1016/S1574-6526(07)03011-8)
- Adnan Darwiche. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.
- Daniel de Carvalho. 2018. Execution time of λ -terms via denotational semantics and intersection types. *Math. Str. in Comput. Sci.* 28, 7 (2018), 1169–1203. <https://doi.org/10.1017/S0960129516000396>
- Thomas Dean and Keiji Kanazawa. 1989. A model for reasoning about persistence and causation. *Computational Intelligence* 5, 2 (1989), 142–150. <https://doi.org/10.1111/j.1467-8640.1989.tb00324.x> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8640.1989.tb00324.x>
- Jeff Egger, Rasmus Ejlers Møgelberg, and Alex Simpson. 2014. The enriched effect calculus: syntax and semantics. *J. Log. Comput.* 24, 3 (2014), 615–654. <https://doi.org/10.1093/logcom/exs025>
- Thomas Ehrhard. 2016. Call-By-Push-Value from a Linear Logic Point of View. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9632)*, Peter Thiemann (Ed.). Springer, 202–228. https://doi.org/10.1007/978-3-662-49498-1_9
- Thomas Ehrhard, Claudia Faggian, and Michele Pagani. 2023a. The Sum-Product Algorithm For Quantitative Multiplicative Linear Logic. In *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023, July 3-6, 2023, Rome, Italy (LIPIcs, Vol. 260)*, Marco Gaboardi and Femke van Raamsdonk (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:18. <https://doi.org/10.4230/LIPIcs.FSCD.2023.8>
- Thomas Ehrhard, Claudia Faggian, and Michele Pagani. 2023b. The Variable Elimination Algorithm as a Let-Term Rewriting. Talk presented at LAFI 2023.
- Thomas Ehrhard, Michele Pagani, and Christine Tasson. 2014. Probabilistic Coherence Spaces are Fully Abstract for Probabilistic PCF. In *The 41th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL14, San Diego, USA*, P. Sewell (Ed.). ACM.
- Thomas Ehrhard and Christine Tasson. 2019. Probabilistic call by push value. *Log. Methods Comput. Sci.* 15, 1 (2019). [https://doi.org/10.23638/LMCS-15\(1:3\)2019](https://doi.org/10.23638/LMCS-15(1:3)2019)
- Claudia Faggian, Daniele Pautasso, and Gabriele Vanoni. 2023. Higher-Order Bayesian Networks, Exactly (Extended version). *CoRR* 2311.04651 (2023). <https://arxiv.org/abs/2311.04651>
- Nir Friedman, Daphne Koller, and Avi Pfeffer. 1998. Structured Representation of Complex Stochastic Systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*, Jack Mostow and Chuck Rich (Eds.). AAAI Press / The MIT Press, 157–164. <http://www.aaai.org/Library/AAAI/1998/aaai98-022.php>
- W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. 1994. A Language and Program for Complex Bayesian Modelling. *Journal of the Royal Statistical Society. Series D (The Statistician)* 43, 1 (1994), 169–177. <http://www.jstor.org/stable/2348941>

- Jean-Yves Girard. 1989. Geometry of Interaction 1: Interpretation of System F. In *Logic Colloquium '88*, R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo (Eds.). Studies in Logic and the Foundations of Mathematics, Vol. 127. Elsevier, 221 – 260. [https://doi.org/10.1016/S0049-237X\(08\)70271-4](https://doi.org/10.1016/S0049-237X(08)70271-4)
- Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *UAI 2008, Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, Helsinki, Finland, July 9-12, 2008*, David A. McAllester and Petri Myllymäki (Eds.). 220–229.
- Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, James D. Herbsleb and Matthew B. Dwyer (Eds.). ACM, 167–181. <https://doi.org/10.1145/2593882.2593900>
- Maria I. Gorinova, Andrew D. Gordon, Charles Sutton, and Matthijs Vákár. 2022. Conditional Independence by Typing. *ACM Trans. Program. Lang. Syst.* 44, 1 (2022), 4:1–4:54. <https://doi.org/10.1145/3490421>
- Chris Heunen, Ohad Kammar, Sam Staton, and Hongseok Yang. 2017. A convenient category for higher-order probability theory. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*. IEEE Computer Society, 1–12. <https://doi.org/10.1109/LICS.2017.8005137>
- Bart Jacobs. 2023. Structured Probabilistic Reasoning. (2023). <http://www.cs.ru.nl/B.Jacobs/PAPERS/ProbabilisticReasoning.pdf>
- Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. 2019. Causal Inference by String Diagram Surgery. In *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11425)*, Mikolaj Bojanczyk and Alex Simpson (Eds.). Springer, 313–329. https://doi.org/10.1007/978-3-030-17127-8_18
- Bart Jacobs and Fabio Zanasi. 2016. A Predicate/State Transformer Semantics for Bayesian Learning. In *The Thirty-second Conference on the Mathematical Foundations of Programming Semantics, MFPS 2016, Carnegie Mellon University, Pittsburgh, PA, USA, May 23-26, 2016 (Electronic Notes in Theoretical Computer Science, Vol. 325)*, Lars Birkedal (Ed.). Elsevier, 185–200. <https://doi.org/10.1016/j.entcs.2016.09.038>
- Bart Jacobs and Fabio Zanasi. 2020. The Logical Essentials of Bayesian Reasoning. In *Foundations of Probabilistic Programming*. Cambridge University Press, 295 – 332.
- Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.
- Daphne Koller, David A. McAllester, and Avi Pfeffer. 1997. Effective Bayesian Inference for Stochastic Programs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence 1997, Providence, Rhode Island, USA, Benjamin Kuipers and Bonnie L. Webber (Eds.)*. AAAI Press / The MIT Press, 740–747.
- Daphne Koller and Avi Pfeffer. 1997. Object-Oriented Bayesian Networks. In *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997*, Dan Geiger and Prakash P. Shenoy (Eds.). Morgan Kaufmann, 302–313. https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=324&proceeding_id=13
- Jim Laird, Giulio Manzonetto, Guy McCusker, and Michele Pagani. 2013. Weighted Relational Models of Typed Lambda-Calculi. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, 301–310. <https://doi.org/10.1109/LICS.2013.36>
- Paul Blain Levy. 1999. Call-by-Push-Value: A Subsuming Paradigm. In *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings (Lecture Notes in Computer Science, Vol. 1581)*, Jean-Yves Girard (Ed.). Springer, 228–242. https://doi.org/10.1007/3-540-48959-2_17
- Vikash Mansinghka, Daniel Selsam, and Yura Perov. 2014. Venture: a higher-order probabilistic programming platform with programmable inference. arXiv:1404.0099 [cs.AI]
- Paul-André Mellies and Nicolas Tabareau. 2010. Resource modalities in tensor logic. *Ann. Pure Appl. Log.* 161, 5 (2010), 632–653. <https://doi.org/10.1016/j.apal.2009.07.018>
- Robin Milner. 2006. Local Bigraphs and Confluence: Two Conjectures: (Extended Abstract). In *Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26, 2006 (Electronic Notes in Theoretical Computer Science, Vol. 175)*, Roberto M. Amadio and Iain Phillips (Eds.). Elsevier, 65–73. <https://doi.org/10.1016/j.entcs.2006.07.035>
- Eugenio Moggi. 1989. Computational Lambda-Calculus and Monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE Computer Society, 14–23.
- Richard E. Neapolitan. 2003. *Learning Bayesian Networks*. Prentice Hal.
- Hugo Paquet. 2021. Bayesian strategies: probabilistic programs as generalised graphical models. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12648)*, Nobuko Yoshida (Ed.). Springer, 519–547. https://doi.org/10.1007/978-3-030-72019-3_19

- Sungwoo Park. 2003. A calculus for probabilistic languages. In *Proceedings of TLDI'03: 2003 ACM SIGPLAN International Workshop on Types in Languages Design and Implementation, New Orleans, Louisiana, USA, January 18, 2003*. 38–49.
- Sungwoo Park, Frank Pfenning, and Sebastian Thrun. 2005. A Probabilistic Language based upon Sampling Functions. In *Conference Record of the 32nd Symposium on Principles of Programming Languages (POPL'05)*, M. Abadi (Ed.). ACM Press, Long Beach, California, 171–182.
- Judea Pearl. 1986. Fusion, Propagation, and Structuring in Belief Networks. *Artif. Intell.* 29, 3 (1986), 241–288. [https://doi.org/10.1016/0004-3702\(86\)90072-X](https://doi.org/10.1016/0004-3702(86)90072-X)
- Judea Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Avi Pfeffer. 2016. *Practical Probabilistic Programming* (1st ed.). Manning Publications Co., USA.
- Avi Pfeffer and Daphne Koller. 2000. Semantics and Inference for Recursive Probability Models. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA*, Henry A. Kautz and Bruce W. Porter (Eds.). AAAI Press / The MIT Press, 538–544. <http://www.aaai.org/Library/AAAI/2000/aaai00-082.php>
- Daniel Pless and George F. Luger. 2001. Toward General Analysis of Recursive Probability Models. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001*. 429–436.
- Norman Ramsey and Avi Pfeffer. 2002. Stochastic lambda calculus and monads of probability distributions. In *Conference Record of POPL 2002: The 29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, OR, USA, January 16-18, 2002*. 154–165. <https://doi.org/10.1145/503272.503288>
- N. Saheb-Djahromi. 1978. Probabilistic LCF. In *Mathematical Foundations of Computer Science 1978, Proceedings, 7th Symposium, Zakopane, Poland, September 4-8, 1978 (Lecture Notes in Computer Science, Vol. 64)*, Józef Winkowski (Ed.). Springer, 442–451. https://doi.org/10.1007/3-540-08921-7_92
- Adam Scibior, Ohad Kammar, Matthijs Vákár, Sam Staton, Hongseok Yang, Yufei Cai, Klaus Ostermann, Sean K. Moss, Chris Heunen, and Zoubin Ghahramani. 2018. Denotational validation of higher-order Bayesian inference. *Proc. ACM Program. Lang.* 2, POPL (2018), 60:1–60:29. <https://doi.org/10.1145/3158148>
- Alex K. Simpson. 2005. Reduction in a Linear Lambda-Calculus with Applications to Operational Semantics. In *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3467)*, Jürgen Giesl (Ed.). Springer, 219–234. https://doi.org/10.1007/978-3-540-32033-3_17
- Dario Stein and Sam Staton. 2021. Compositional Semantics for Probabilistic Programs with Exact Conditioning. *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS) (2021)*, 1–13.
- Matthijs Vákár, Ohad Kammar, and Sam Staton. 2019. A domain theory for statistical probabilistic programming. *Proc. ACM Program. Lang.* 3, POPL (2019), 36:1–36:29. <https://doi.org/10.1145/3290349>
- Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. 2018. An Introduction to Probabilistic Programming. *CoRR* abs/1809.10756 (2018). arXiv:1809.10756 <http://arxiv.org/abs/1809.10756>
- Frank Wood, Jan Willem Meent, and Vikash Mansinghka. 2014. A New Approach to Probabilistic Programming Inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 33)*, Samuel Kaski and Jukka Corander (Eds.). PMLR, Reykjavik, Iceland, 1024–1032. <https://proceedings.mlr.press/v33/wood14.html>

APPENDIX

We include here proofs that have been omitted in the body of the article, and some more examples.

Reduction, in the notation of Explicit Substitutions

Since let-terms can be quite heavy to manage, in the Appendix we usually employ **the notation of explicit substitutions**, which allows for more concise proofs:

$$\begin{aligned} t[x \leftarrow u] &\triangleq \text{let } x = u \text{ in } t, \\ t[\langle x, y \rangle \leftarrow v] &\triangleq \text{letp } \langle x, y \rangle = v \text{ in } t. \end{aligned}$$

Substitution lists S , evaluation contexts E , and root reduction rules can therefore be written as follows. As before:

- $\langle t \rangle S$ stands for the term obtained from S by replacing the hole $\langle \cdot \rangle$ with t ,
 - $E(t)$ stands for the term obtained from E by replacing the hole (\cdot) with t ,
- possibly capturing the free variables of t .

$$\begin{aligned} \text{SUBSTITUTION LISTS } S &::= \langle \cdot \rangle \mid [x \leftarrow u]S \mid [\langle x, y \rangle \leftarrow v]S \\ \text{EVALUATION CONTEXTS } E &::= (\cdot) \mid Ev \mid t[x \leftarrow E] \mid E[x \leftarrow u] \end{aligned}$$

ROOT RULES

$$\begin{aligned} \langle \lambda x. t \rangle S \ v &\mapsto_{\text{db}} \langle t[x \leftarrow v] \rangle S & t[x \leftarrow \langle v \rangle S] &\mapsto_{\text{dsub}} \langle t\{x \leftarrow v\} \rangle S \\ \text{der } !t &\mapsto_{\text{der}} t & t[\langle x, y \rangle \leftarrow \langle v, w \rangle] &\mapsto_{\text{pm}} t[x \leftarrow v][y \leftarrow w] \end{aligned}$$

Please notice that here we use a formulation for the \mapsto_{db} and \mapsto_{pm} rules which is slightly different but *equivalent* to that in Sect. 3.2. The advantage (at the level of proofs) is that actual substitution is now carried by the \mapsto_{dsub} rule only.

A COMPLEMENTS TO SECT. 3 (THE CALCULUS)

We write $\pi \triangleright_{\text{low}}$ when π contains first-order rules, only.

LEMMA (3.4, PROGRESS). *Let t be a λ_1 -term in normal form such that $\pi \triangleright \mathcal{L} \vdash t : L$, where L and all types in the context \mathcal{L} are ground. Then π only uses first-order rules, i.e. $\pi \triangleright_{\text{low}} \mathcal{L} \vdash t : L$.*

PROOF. We will prove a more general result, namely that if t is a λ_1 -term in *normal* form and $\pi \triangleright \mathcal{L} \vdash t : C$, where all types in the context \mathcal{L} are ground, then the following properties hold:

1. if $C = !A$ then $t = \langle !u \rangle S$;
2. if $C = P \multimap A$ then $t = \langle \lambda x. u \rangle S$;
3. if $C = L_1 \otimes L_2$ then $t = \langle v \rangle S$;
4. if $C = L$ then t is a low-level term, i.e. $\pi \triangleright_{\text{low}} \mathcal{L} \vdash t : L$.

The proof proceeds by induction on the type derivation π , considering its last rule.

- Case s-VAR is immediate. Since we assume that the typing context only contains ground types, necessarily:

$$\pi \triangleright \mathcal{L}', x : L \vdash x : L$$

- Case s-PAIR. Immediate by *i.h.*:

$$\frac{\mathcal{L} \vdash v_1 : L_1 \quad \mathcal{L} \vdash v_2 : L_2}{\pi \triangleright \mathcal{L} \vdash \langle v_1, v_2 \rangle : L_1 \otimes L_2} \text{S-PAIR}$$

- Case s-SAMPLE and s-COND are immediate.
- Case s-OBS is immediate.

- Case s-BANG is immediate:

$$\frac{\mathcal{L} \vdash u : A}{\pi \triangleright \mathcal{L} \vdash !u : !A} \text{S-BANG}$$

- Case s-ABS. Immediate:

$$\frac{\mathcal{L}, x : P \vdash u : A}{\pi \triangleright \mathcal{L} \vdash \lambda x. u : P \multimap A} \text{S-ABS}$$

- Case s-LET. The derivation π has shape:

$$\frac{\mathcal{L} \vdash s : P \quad \mathcal{L}, x : P \vdash u : C}{\pi \triangleright \mathcal{L} \vdash u[x \leftarrow s] : C} \text{S-LET}$$

Observe that P is necessarily a ground type. Indeed, $P = !A$ is not possible, because by *i.h.* we would have $s = \langle !r \rangle S$, making $u[x \leftarrow s]$ a redex. By *i.h.*, all the claims hold for both u and s , hence they hold for $u[x \leftarrow s]$.

- Case s-LETP. Necessarily, $t \triangleq u[\langle x, y \rangle \leftarrow v]$ in normal form implies $v = z$ (a single variable), because $u[\langle x, y \rangle \leftarrow \langle v_1, v_2 \rangle]$ would be a redex. So we have

$$\frac{\frac{\mathcal{L} \vdash z : L_1 \otimes L_2}{\pi \triangleright \mathcal{L} \vdash u[\langle x, y \rangle \leftarrow z] : C} \text{S-VAR} \quad \mathcal{L}, x : L_1, y : L_2 \vdash u : C}{\pi \triangleright \mathcal{L} \vdash u[\langle x, y \rangle \leftarrow z] : C} \text{S-LETP}$$

By *i.h.*, the claim holds for u , and therefore for $u[\langle x, y \rangle \leftarrow z]$.

The following cases never apply because of the assumption that t is normal:

- Case s-APP. We have

$$\frac{\mathcal{L} \vdash u : P \multimap C \quad \mathcal{L} \vdash v : P}{\pi \triangleright \mathcal{L} \vdash uv : C} \text{S-APP}$$

where by *i.h.* $u = \langle \lambda x. s \rangle S$, which is not possible, because uv would be a redex.

- Case s-DER. This case does not apply, because by *i.h.* we would have $u = !s$, making $der u$ a redex.

$$\frac{\mathcal{L} \vdash u : !A}{\pi \triangleright \mathcal{L} \vdash der u : A} \text{S-DER}$$

□

By Progress and the fact that typable terms are strongly normalizing, we have

COROLLARY A.1. *If $\pi \triangleright \mathcal{L} \vdash t : L$, then $t \rightarrow^* u$, where u is a low-level term in normal form (with type derivation $\pi' \triangleright_{\text{LOW}} \mathcal{L} \vdash u : L$).*

B COMPLEMENTS TO SECT. 4 (THE TYPE SYSTEM)

B.1 The Full Type System

For convenience, we give explicitly the full type system, here denoted iTypes^+ , which takes into account also the construct $\text{obs}(x = b)$, as we have described in Sect. 8.

The grammar of types is as follows, where $X \in \text{Names}$:

$$\begin{aligned} \text{ATOMIC TYPES} \quad X &::= X \mid X^t \mid X^f \\ \text{GROUND TYPES} \quad K, L &::= X \mid K \otimes L \\ \text{POSITIVE TYPES} \quad P, Q &::= L \mid [A_1, \dots, A_n] \\ \text{TYPES} \quad A, B &::= P \mid P \multimap A \end{aligned}$$

The typing rules for the full system iTypes^+ are in Fig. 14.

Higher-order calculus	
First-order rules	
$\frac{X \notin \text{Nm}(\Lambda)}{\Lambda \vdash \text{sample}_d : \mathcal{X}} \text{I-SAMPLE}$	$\frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : \mathcal{Y}_1, \dots, y_n : \mathcal{Y}_n \vdash \mathbf{c}\langle y_1, \dots, y_n \rangle : \mathcal{X}} \text{I-COND}$
$\frac{b \in \{t, f\}}{\Lambda, x : \mathcal{X}^b \vdash \text{obs}(x = b) : \mathcal{X}^b} \text{I-OBS}$	$\frac{}{\Lambda, x : P \vdash x : P} \text{I-VAR} \qquad \frac{\Lambda, \Gamma_1 \vdash u : P \quad \Lambda, \Gamma_2, x : P \vdash t : A}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \text{let } x = u \text{ in } t : A} \text{I-LET}$
$\frac{\Lambda \vdash v : L_1 \quad \Lambda \vdash w : L_2}{\Lambda \vdash \langle v, w \rangle : L_1 \otimes L_2} \text{I-PAIR}$	$\frac{\Lambda \vdash v : L_1 \otimes L_2 \quad \Lambda, x : L_1, y : L_2, \Gamma \vdash t : A}{\Lambda, \Gamma \vdash \text{letp } \langle x, y \rangle = v \text{ in } t : A} \text{I-LETP}$
$\frac{\Lambda, \Gamma, x : P \vdash t : A}{\Lambda, \Gamma \vdash \lambda x. t : P \multimap A} \text{I-ABS}$	$\frac{\Lambda, \Gamma_1 \vdash t : P \multimap A \quad \Lambda, \Gamma_2 \vdash v : P}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash tv : A} \text{I-APP}$
$\frac{(\Lambda, \Gamma_i \vdash t : A_i)_{i=1}^n}{\Lambda, \uplus_i \Gamma_i \vdash !t : [A_1, \dots, A_n]} \text{I-BANG}$	$\frac{\Lambda, \Gamma \vdash v : [A]}{\Lambda, \Gamma \vdash \text{der } v : A} \text{I-DER}$

Fig. 14. The full intersection type system: iTypes+

B.2 Properties of the Type System

Subject Reduction, Subject Expansion, Termination and Progress. The extensive proofs of Subject Reduction and Subject Expansion are in Appendix G and Appendix H, respectively. The statements are strengthened to obtain several other properties we need. The proof of Progress is very similar to the proof in Appendix A. Please notice that the results mentioned above hold both for the system in Fig. 7 and for the full system iTypes+ in Fig. 14. The proofs are carried out in the latter.

Unique type derivation. A separate treatment is needed for the property of unique type derivation in Prop. 4.7: this property—in the way it is stated in Sect. 4—only holds for the system in Fig. 7, that is a system with *no* observed types, but *not* for the full system (Fig. 14). In Appendix B.4 we generalize the property of "unique derivation", providing a form which holds in general.

B.3 Unique Derivation (Prop. 4.7), holding for the system in Fig. 7

The interest of Prop. 4.7 (and its immediate corollary Thm. 4.8) is that given a term t in BN normal form, then its *semantics is uniquely determined*, because its type derivation π is uniquely determined. The same holds for any term t which reduces to BN normal form. In fact, if t is closed, we can simply (and *uniquely*) write $\llbracket t \rrbracket$ for $\llbracket \pi \rrbracket$.

To prove Prop. 4.7 we need a preliminary lemma.

LEMMA B.1. *Let t be a λ_{low} -term, and Λ a ground context. If there exists a type derivation $\pi \triangleright_{\text{low}} \Lambda \vdash t : L$, then it is unique.*

PROOF. The proof proceeds by induction on t . Remark that π does not contain I-OBS rules, nor observed types.

- Case $t = x$. Necessarily $\Lambda = \Lambda', x : L$, therefore the derivation $\pi \triangleright \Lambda', x : L \vdash x : L$ is uniquely determined.
- Cases $t = \text{sample}_d$ and $t = \mathbf{c}\langle x_1, \dots, x_n \rangle$ are immediate, as the derivation π is uniquely determined by the choice of Λ .

- Case $t = \langle v_1, v_2 \rangle$. By *i.h.* there are at most one L_i such that $\pi_i \triangleright \Lambda \vdash v_i : L_i$ ($i \in \{1, 2\}$); moreover, if they exist, π_1 and π_2 are both unique. Then there is only one way to obtain the derivation $\pi \triangleright \Lambda \vdash \langle v_1, v_2 \rangle : L$, where $L = L_1 \otimes L_2$, by rule I-PAIR.
- Case $t = \text{let } x = u \text{ in } s$. By *i.h.* there is at most one K such that $\pi_1 \triangleright \Lambda \vdash u : K$. Similarly, by *i.h.* there is at most one L such that $\pi_2 \triangleright \Lambda, x : K \vdash s : L$. Moreover, if they exist, both π_1 and π_2 are unique. Then there is only one way to obtain the derivation $\pi \triangleright \Lambda \vdash \text{let } x = u \text{ in } s : L$, by rule I-LET.
- Case $t = \text{letp } \langle y_1, y_2 \rangle = v \text{ in } u$. Similar to the previous case. □

PROPOSITION (4.7, UNIQUE DERIVATION). *Let t be a λ_1 -term, and Λ a ground context. Then there exists at most one type derivation π such that $\pi \triangleright \Lambda \vdash t : L$.*

PROOF. Let us suppose t is typable as $\pi \triangleright \Lambda \vdash t : L$. We prove that π is unique. By Thm. 4.5, we have that t is strongly normalizing. Then we argue by induction on the number of steps n needed to reach its normal form.

- If $n = 0$, then t is in normal form. In particular, by Thm. 4.6, we have that t is low-term and therefore by Lemma B.1 satisfies the claim.
- If $n > 0$, then we have $t \rightarrow u \rightarrow^{n-1} r$, where r is in normal form. By Prop. 4.4 t is typable as $\pi \triangleright \Lambda \vdash t : L$ if and only if u is typable as $\pi' \triangleright \Lambda \vdash u : L$. Then, we apply the *i.h.* to u , obtaining that if there exists $\pi' \triangleright \Lambda \vdash u : L$, then π' is unique. We conclude by Thm. H.3. □

B.4 Unique General Derivation, holding for the full type system (with observe)

When we consider the full system `iTypes+` (in Fig. 14), which includes *observed types*, then Subject Reduction, Subject Expansion, Progress, and Thm. 4.6 still hold, in the same formulation of Sect. 4.3. The *unique derivation* property in Prop. 4.7 instead does not hold as stated. For example, the term `sampled` can be typed in three valid ways:

$$\vdash \text{sample}_d : X \quad \vdash \text{sample}_d : X^t \quad \vdash \text{sample}_d : X^f$$

We notice however that all the three derivations above have the same shape $\vdash \text{sample}_d : X$, for $X \in \{X, X^t, X^f\}$, and that the type X can be seen as “*more general*” than X^t and X^f . This remark turns out to be a property of all derivations of ground type, and allows us to generalize Prop. 4.7 in a way that holds for the full type system.

B.4.1 Unique General Derivation. Recall that an atomic type X may assume two forms: either as observed type (X^t, X^f), or as unobserved type X . Since the type system is *syntax driven*, the term itself can stipulate that some types are observed, via the construct `obs(x = b)`, which is reflected in a I-OBS-rule. We prove that if a closed term t is typable, it admits *exactly one* derivation π which is general, in the sense that (intuitively) it contains *no more information than that provided by the term t* (Prop. B.3). Moreover, all the other valid type derivations for the same term t are obtained as refinement of π (Prop. B.4).

Let us first formalize the intuitive notion that a type derivation is general if it declares as observed all and only the types that the term prescribes as observed.

Definition B.2 (General Derivation). A type derivation π in system `iTypes+` is *general* if each atomic type which appears in π and which does not type the subject of any I-OBS rule, is *unobserved* (i.e., has form X and not X^b).

The proposition below (that we prove in Sect. B.4.2) allows us to recover uniqueness. Please notice that any derivation in the type system of Fig. 7 is general. This way, Prop. 4.7 is a special case of Prop. B.3.

PROPOSITION B.3 (UNIQUE GENERAL DERIVATION). *Let t be a closed λ_1 -term. In system $i\text{Types}^+$, if there exists a derivation $\pi \triangleright \vdash t : L$, then there exists a unique general derivation $\pi' \triangleright \vdash t : L'$.*

If t is a closed term, and $\pi \triangleright \vdash t : L$ is its general derivation, then any other type derivation for t is obtained by refinement, in the following sense. We define an order relation on atomic types:

$$X \geq X^b \text{ (} X \text{ is more general than } X^b \text{), for each } X \in \text{Names.}$$

The order extends to all types, and to type derivations, in the natural way, *point-wise*.

PROPOSITION B.4 (MOST GENERAL DERIVATION). *Let t be a closed λ_1 -term and $\pi \triangleright \vdash t : L$ its general derivation. Then $\pi \geq \pi'$ for any derivation $\pi' \triangleright \vdash t : L'$.*

B.4.2 Proof of Prop. B.3.

Existence. We first establish the existence of a general derivation, for every term which has a type derivation. The following property is immediate to check by inspecting the typing rules.

LEMMA B.5 (GENERALIZATION). *Let $\pi \triangleright \Sigma \vdash t : A$ be a type derivation, and let $\text{Obs}(\pi)$ be the set of the atomic types which occur as the subject of an I -OBS rule. Replacing in π all occurrences of atom X^b with X , for each atom which appears in π but does not belong to $\text{Obs}(\pi)$ yields a valid type derivation, written $\pi^* \triangleright \Sigma^* \vdash t : A^*$. Moreover, the derivation π^* is general; we call π^* the generalization of π .*

Uniqueness. To prove that a closed term of ground type has a *unique* general derivation, we rely on Prop. 4.7. We first need the notion of skeleton, and some technical lemmas.

We write \widehat{A} for the type obtained from A by replacing all occurrences of the atom X^b with X , for each $X \in \text{Names}$. We write \widehat{t} for the term obtained from t by replacing all occurrence of subterm $\text{obs}(x = b)$ with x , for each $x \in \mathcal{V}$.

LEMMA B.6 (SKELETON). *Let $\pi \triangleright \Sigma \vdash t : A$ be a type derivation. Replacing in π each judgment $\Sigma_u \vdash u : A_u$ with the judgment $\widehat{\Sigma}_u \vdash \widehat{u} : \widehat{A}_u$ yields a type derivation $\widehat{\pi} \triangleright \widehat{\Sigma} \vdash \widehat{t} : \widehat{A}$, which we call the skeleton of π .*

By construction, all derivations with the same skeleton are *identical*, modulo quotienting X and X^b . In fact, we can be more precise.

LEMMA B.7 (MAIN). *Let $\pi_1 \triangleright \Sigma_1 \vdash t : L_1$ and $\pi_2 \triangleright \Sigma_2 \vdash t : L_2$ be type derivations with the same skeleton $\widehat{\pi}_1 = \widehat{\pi}_2$. The two derivations may only differ on the atoms which do not type an I -OBS-rule.*

PROOF. By induction on the term t . □

COROLLARY B.8. *Two type derivations with the same skeleton have the same generalization:*

$$\widehat{\pi}_1 = \widehat{\pi}_2 \Rightarrow \pi_1^* = \pi_2^*.$$

By using Prop. 4.7, we have

PROPOSITION B.9 (UNIQUENESS). *Let t be a closed λ_1 -term. If t has two general derivations of ground type $\pi_1 \triangleright \vdash t : L_1$ and $\pi_2 \triangleright \vdash t : L_2$ then $\pi_1 = \pi_2$.*

PROOF. By observing that the skeleton of a type derivation only uses the rules in Fig. 7, and therefore satisfies Prop. 4.7, we have that all the *ground type* derivations of the same term have the same skeleton. Hence the claim (notice that if π_i is general, then $\pi_i^* = \pi_i$). □

C COMPLEMENTS TO SECT. 6 (THE SEMANTICS OF TYPED TERMS)

C.1 Interpretation of the Probabilistic Axioms, Formally

In the paper, we are a slightly informal when describing how to associate a factor to a I-COND axiom. Let us describe how to associate a CPT to each probabilistic axiom, formally, and what are the subtleties.

- To a I-SAMPLE axiom of shape:

$$\frac{}{\Lambda \vdash \text{sample}_d : X \diamond \phi^{\{X\}}}$$

we associate the factor ϕ over the singleton $\{X\}$ such that, for all $x \in \text{Val}(X)$:

$$\phi(x) = d(x)$$

- To a I-COND axiom of shape:

$$\frac{X \notin \{Y_1, \dots, Y_n\} \text{ and } X \notin \text{Nm}(\Lambda)}{\Lambda, y_1 : Y_1, \dots, y_n : Y_n \vdash \text{case } \langle y_1, \dots, y_n \rangle \text{ of } \{\bar{b} \Rightarrow \text{sample}_{d_{\bar{b}}}\}_{\bar{b} \in \{t, f\}^n} : X \diamond \phi^{\mathbb{Y} \cup \{X\}}}$$

we associate a factor ϕ over the set of names $\mathbb{Y} \cup \{X\}$, where $\mathbb{Y} = \{Y_1, \dots, Y_n\}$. Remark that Y_1, \dots, Y_n need not to be pairwise distinct, hence \mathbb{Y} is a set whose cardinality $|\mathbb{Y}| = k$ may be less than n . The factor ϕ is defined as follows, for all $x \in \text{Val}(X)$ and $\bar{y} \in \text{Val}(\mathbb{Y})$ (please observe that \bar{y} is a tuple of k elements):

$$\phi(\bar{y}x) = d_{\bar{b}}(x)$$

where $\bar{b} = \langle \bar{y}|_{Y_1}, \dots, \bar{y}|_{Y_n} \rangle$, recalling that (by Notation 5.3) $\bar{y}|_{Y_i} \in \text{Val}(Y_i)$ is the value that Y_i assumes in $\bar{y} \in \text{Val}(\mathbb{Y})$.

The subtlety in the definition is that if the same name Y occurs several times in the context (say, $Y_1 = Y_2 = Y_4$) then the *same value* will be repeated several times in the tuple \bar{b} , at the corresponding positions. We give two examples to clarify.

Example C.1. Taking the Bayesian network in Fig. 1 as reference, we provide an example of a case expression and the associated CPT. In the term, $x : S$ and $y : R$ are two variables of suitable type.

case $\langle x, y \rangle$ of	S	R	W	Pr(W S, R)
$\langle t, t \rangle \Rightarrow \text{bernoulli}_{0.99}$	t	t	t	0.99
$\langle f, t \rangle \Rightarrow \text{bernoulli}_{0.7}$	t	t	f	0.01
$\langle t, f \rangle \Rightarrow \text{bernoulli}_{0.9}$	f	t	t	0.7
$\langle f, f \rangle \Rightarrow \text{bernoulli}_{0.01}$	f	t	f	0.3
$\langle f, f \rangle \Rightarrow \text{bernoulli}_{0.01}$	t	f	t	0.9
$\langle f, f \rangle \Rightarrow \text{bernoulli}_{0.01}$	t	f	f	0.1
	f	f	t	0.01
	f	f	f	0.99

Example C.2. Let us also give an example where the same name appear several times. If we have $y_1 : Y, y_2 : Y \vdash \text{case } \langle y_1, y_2 \rangle \text{ of } \{\bar{b} \Rightarrow \text{sample}_{d_{\bar{b}}}\}_{\bar{b} \in \{t, f\}^2} : X$ the factor over the names $\{Y, X\}$ associates to $\langle t, t \rangle \in \text{Val}(Y) \times \text{Val}(Y)$ the value $d_{t,t}(t)$, to $\langle f, t \rangle \in \text{Val}(Y) \times \text{Val}(Y)$ the value $d_{f,t}(t)$, and so on.

C.2 Invariance of the Semantics

The factor semantics we have defined is invariant under reduction and expansion, as expected. This is due to the fact that probabilistic axioms are stable w.r.t. reduction and expansion.

THEOREM (6.10, INVARIANCE). *Let t be a $\lambda_!$ -term and $t \rightarrow u$. Then $\Lambda \vdash t : L \diamond \psi$ if and only if $\Lambda \vdash u : L \diamond \psi$.*

PROOF. Invariance of the semantics is a consequence of Thm. 6.9, together with the Subject Reduction and Subject Expansion (Appendix G, Appendix H): in the proof of such properties it suffices to observe that $\text{Cpts}(\pi) = \text{Cpts}(\pi')$. Then $\llbracket \pi \rrbracket = \llbracket \pi' \rrbracket$ by Thm. 6.9. \square

Example C.3. To better understand the invariance of $\text{Cpts}(\pi)$ under reduction, recall that *only values can be duplicated and deleted*. One can see the consequences of this fact by considering the two following terms, and their type derivation. Observe that the first term is in normal form, precisely because only values can be deleted. By contrast, the subterm !sample_d , which is going to be deleted, can only be assigned an empty type.

$$\begin{array}{c}
 \frac{x : X, y : Y \vdash y : Y}{\vdash \text{sample}_d : X \diamond \overset{\{x\}}{\phi} \quad x : X \vdash \lambda y. y : Y \multimap Y \overset{0}{\diamond} \mathbf{1}} \\
 \hline
 \vdash \text{let } x = \text{sample}_d \text{ in } \lambda y. y : Y \multimap Y \overset{0}{\diamond} \mathbf{1} \quad \rightarrow
 \end{array}$$

$$\begin{array}{c}
 \frac{y : Y \vdash y : Y}{\vdash \text{!sample}_d : [] \overset{0}{\diamond} \mathbf{1} \quad x : [] \vdash \lambda y. y : Y \multimap Y \overset{0}{\diamond} \mathbf{1}} \\
 \hline
 \vdash \text{let } x = \text{!sample}_d \text{ in } \lambda y. y : Y \multimap Y \overset{0}{\diamond} \mathbf{1} \quad \rightarrow \quad \frac{y : Y \vdash y : Y}{\vdash \lambda y. y : Y \multimap Y \overset{0}{\diamond} \mathbf{1}}
 \end{array}$$

C.3 Completion of the Semantics

As briefly discussed in Sect. 6, the reader may expect that the interpretation of a type derivation $\pi \triangleright J$ were a factor over $\text{Nm}(J)$. For example, one would expect to interpret an identity axiom α as

$$y : Y \vdash y : Y \overset{\{Y\}}{\diamond} \mathbf{1} \quad \text{instead of} \quad y : Y \vdash y : Y \overset{0}{\diamond} \mathbf{1}$$

The fact is that our semantics focuses on the *probabilistic content* of a derivation $\pi \triangleright J$, only. Please notice that the non-probabilistic information is already fully contained in the type judgment J (because intersection types carry such information). Indeed, an interpretation of $\pi \triangleright J$ as a factor over $\text{Nm}(J)$ is easily obtained by a form of *completion*, which is simply a multiplication for the trivial factor:

$$\overline{\llbracket \pi \rrbracket} \triangleq \llbracket \pi \rrbracket \odot \overset{\text{Nm}(J)}{\mathbf{1}}$$

In the case of the identity axiom, this gives $\overline{\llbracket \alpha \rrbracket} = \overset{0}{\mathbf{1}} \odot \overset{Y}{\mathbf{1}} = \overset{Y}{\mathbf{1}}$. This completion commutes with composition. We give some more details below.

Definition C.4 (Completion). Given a type derivation $\pi \triangleright J$, the completion of $\llbracket \pi \rrbracket$ is the factor over the set of names $\text{Nm}(J)$ defined as:

$$\overline{\llbracket \pi \rrbracket} \triangleq \llbracket \pi \rrbracket \odot \overset{\text{Nm}(J)}{\mathbf{1}}$$

Example C.5 (Completed factor). The example below shows, respectively, the factor semantics and its completion for I-VAR and I-SAMPLE axioms.

$$\frac{}{\pi_1 \triangleright y : Y \vdash y : Y} \qquad \frac{}{\pi_2 \triangleright y : Y \vdash \text{bernoulli}_p : X}$$

$$\llbracket \pi_1 \rrbracket = \mathbf{1}_\emptyset \qquad \begin{array}{c|c} Y & \llbracket \pi_1 \rrbracket_y \\ \hline f & 1 \\ t & 1 \end{array} \qquad \begin{array}{c|c} X & \llbracket \pi_2 \rrbracket_x \\ \hline f & p \\ t & (1-p) \end{array} \qquad \begin{array}{c|c} Y & X & \llbracket \pi_2 \rrbracket_{yx} \\ \hline f & f & p \\ f & t & (1-p) \\ t & f & p \\ t & t & (1-p) \end{array}$$

Example C.6 (Completed derivation). Below we annotate the same derivation π with $\llbracket \pi \rrbracket$ (l.h.s.) and with its completion $\overline{\llbracket \pi \rrbracket}$ (r.h.s.).

$$\frac{\frac{}{\vdash \text{sample}_d : X \diamond \phi^{\{X\}} \quad x : X, y : Y \vdash y : Y \diamond \mathbf{1}^0} \quad x : X \vdash \lambda y. y : Y \multimap Y \diamond \mathbf{1}^0}{\vdash \text{let } x = \text{sample}_d \text{ in } \lambda y. y : Y \multimap Y \diamond \mathbf{1}^0}}{\vdash \text{let } x = \text{sample}_d \text{ in } \lambda y. y : Y \multimap Y \diamond \mathbf{1}^0} \qquad \frac{\frac{}{\vdash \text{sample}_d : X \diamond \phi^{\{X\}} \quad x : X, y : Y \vdash y : Y \diamond \mathbf{1}^{\{X,Y\}}} \quad x : X \vdash \lambda y. y : Y \multimap Y \diamond \mathbf{1}^{\{X,Y\}}}{\vdash \text{let } x = \text{sample}_d \text{ in } \lambda y. y : Y \multimap Y \diamond \mathbf{1}^{\{Y\}}}}{\vdash \text{let } x = \text{sample}_d \text{ in } \lambda y. y : Y \multimap Y \diamond \mathbf{1}^{\{Y\}}}$$

C.3.1 Bridging with the Relational Models. The completed interpretation is a key step in order to bridge the gap between our semantics and *weighted relational models/probabilistic coherence spaces* such as [Ehrhard et al. 2014; Ehrhard and Tasson 2019; Laird et al. 2013].

The other observation we need is the following, suggested by Example 4.2.

Remark C.7 (On Redundancy). Looking at Example 4.2 one notice an interesting fact, which then shines in our semantics: to memorize the probability distribution over the (four) tuples in $B_X \otimes B_X$, we only need the distribution over the (two) values of B_X , since (as one see in the example) reconstructing the former from the latter is trivial. For example:

- $\text{Val}(B_X) \rightarrow \mathbb{R}: \quad t \mapsto 0.2, f \mapsto 0.8.$
- $\text{Val}(B_X) \otimes \text{Val}(B_X) \rightarrow \mathbb{R}: \quad \langle t, t \rangle \mapsto 0.2, \langle f, f \rangle \mapsto 0.8, \langle t, f \rangle \mapsto 0, \langle f, t \rangle \mapsto 0.$

A similar process allows us to retrieve the relational model from the completed semantics.

D COMPLEMENTS TO SECT. 7 (THE FLOW GRAPH)

D.1 The Flow Graph

To define the flow graph for a *general* type derivation in *iTypes*, we need to specify what are the positions (*i.e.* the vertexes), what are the edges, and how the edges are oriented.

Orientation and Polarity. The edges of the flow graph are oriented according to the input/output polarity of the positions: the orientation is upwards on inputs, and downwards on outputs.

In a judgment of ground type $\Lambda \vdash t : L$, all occurrence of atoms in Λ are seen as input, and all occurrences of atoms in L are seen as output. However, when considering a generic type derivation in *iTypes*, we need to take into account arrow types: note for example that the occurrence Y is an input in the judgment $\vdash t : Y \multimap X$. The definition of polarity given below is rather standard in proof-theory.

Definition D.1 (Input and Output Polarity). To each occurrence of atom $X \in \{X, X^t, X^f\}$ in a judgment is associated either an input polarity (denoted X^\uparrow) or output popularity (denoted X^\downarrow), according to the following definition.

$$\text{Pol}(x : P_1, \dots, x_n : P_n \vdash t : A) \triangleq x : (P_1)^-, \dots, x_n : (P_n)^- \vdash t : (A)^+$$

$$\begin{array}{ll}
(\mathcal{X})^- \triangleq \mathcal{X}^\uparrow & (\mathcal{X})^+ \triangleq \mathcal{X}^\downarrow \\
(A \otimes B)^- \triangleq (A)^- \otimes (B)^- & (A \otimes B)^+ \triangleq (A)^+ \otimes (B)^+ \\
(A \multimap B)^- \triangleq (A)^+ \multimap (B)^- & (A \multimap B)^+ \triangleq (A)^- \multimap (B)^+ \\
([A_1, \dots, A_n])^- \triangleq [(A_1)^-, \dots, (A_n)^-] & ([A_1, \dots, A_n])^+ \triangleq [(A_1)^+, \dots, (A_n)^+]
\end{array}$$

Polarized Positions. Similarly to what we have already done for ground types, we indicate a specific occurrence of an atom inside a type A by means of a (type) context, *i.e.* types with a hole, as follows:

$$\begin{array}{ll}
\text{GROUND TYPE CTXS} & \underline{K}, \underline{L} ::= (\cdot) \mid \underline{K} \otimes \underline{L} \mid \underline{K} \otimes \underline{L} \\
\text{POSITIVE TYPE CTXS} & \underline{P}, \underline{Q} ::= \underline{L} \mid [A_1, \dots, \underline{A}, \dots, A_n] \\
\text{TYPE CTXS} & \underline{A}, \underline{B} ::= \underline{P} \mid \underline{P} \multimap \underline{A} \mid \underline{P} \multimap \underline{A}
\end{array}$$

The definition can be extended to ground and exponential contexts in a straightforward way; we will use $\underline{\Delta}(\mathcal{X})$ and $\underline{\Gamma}(\mathcal{X})$ to refer to a specific atom occurrence in Δ and Γ , respectively. We assume that each atom occurrence appearing in a sequent of a derivation π is given a distinct label. We call such a label a *position*; each position has the polarity of the corresponding atom.

Flow Graph. The flow graph associated to a type derivation π is the *directed graph* that has for *vertexes* the positions of π , and *edges* as indicated in Fig. 15. The *orientation* of the edges is given by the polarity of the positions: each edge enters and exits an *input position* \mathcal{X}^\uparrow *going upwards* (*i.e.*, going from the conclusion of a rule to its premises). Similarly, edges enter and exit an *output position* \mathcal{X}^\downarrow *going downwards* (*i.e.*, going from the premises to the conclusion).

PROPOSITION (7.3, THE FLOW IS ACYCLIC). *Let $\pi \triangleright \Delta \vdash t : L$. Then $\text{flow}(\pi)$ is a DAG.*

PROOF. If t is in normal form, it is immediate to verify, by induction on the derivation π , that $\text{flow}(\pi)$ is *acyclic*. If t is not in normal form, by Thm. 4.6 we know that there exists a term u in normal form such that $t \rightarrow^* u$. Therefore it is enough to prove that cycles are preserved along a reduction sequence; this can be done by strengthening the subject reduction statement (see Appendix G). \square

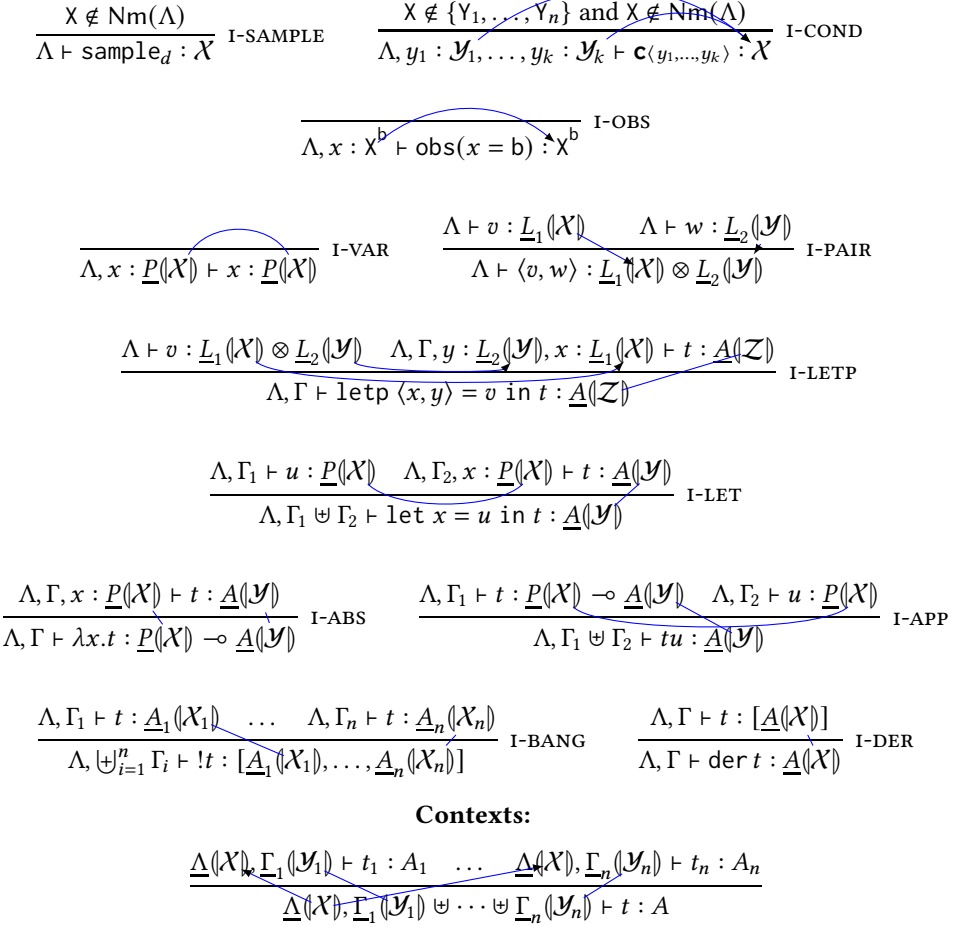
D.2 From DAGs to Compositionality

We say that a position p has underlying name X (resp. underlying atom \mathcal{X}) if p denotes an occurrence of the atom $\mathcal{X} \in \{X, X^\dagger, X^\ddagger\}$. The main position of a probabilistic axiom is the one corresponding to the type of the subject.

LEMMA (7.5, NAMED PATHS). *Let $\pi \triangleright \Delta \vdash t : L$, and let X be the main name of a probabilistic axiom α^X . Then in $\text{flow}(\pi)$, each position with underlying name X is connected to the main position in α^X by a path in which all positions have underlying name X .*

PROOF. Let us start with two easy observations. Notice that both also holds by replacing "underlying name" with "underlying atom"

1. By inspecting Fig. 15 and taking into account the orientation of the edges, we realize that in $\text{flow}(\pi)$ each position has *at most one parent with the same underlying name (resp., the same atom)*. Observe that the only positions which do not have any parent with the same name (resp., the same atom), are either the main position of a probabilistic axiom, or those in the context of the conclusion $\Delta \vdash t : L$ (which have no parent at all).

Fig. 15. Flow graph of a *iTypes* derivation.

2. From (1), by recalling that a DAG where each node has at most one parent is a tree, we have the following key observation: if we partition $\text{flow}(\pi)$ into maximal connected subgraphs whose vertexes are *positions with the same name* X , then each such subgraph is a *directed tree*, whose root corresponds either to the main type of a probabilistic axiom, or to a position in the context Λ of the conclusion.

The same holds true if we partition $\text{flow}(\pi)$ into maximal connected subgraphs whose vertexes are *positions with the same atom* X ,

Now let X be the main name of a probabilistic axiom. One can easily verify that $X \notin \text{Nm}(\Lambda)$ (because ground contexts are additive, and because of the conditions on the axioms names). Since we require the main names of π to be pairwise distinct, we conclude that in $\text{flow}(\pi)$ there is *exactly one* connected component of positions with the same name X . \square

An immediate consequence is that if a name X is summed out, then necessarily it is a main name.

We isolate two useful properties which we have shown as part of the proof of Lemma 7.5, because we will use in the following.

Fact D.2 (Main Names). Let $\pi \triangleright \Lambda \vdash t : L$ be a type derivation, and let X be the main name of a probabilistic axiom. One can easily verify that $X \notin \text{Nm}(\Lambda)$.

Fact D.3 (Atoms). Let $\pi \triangleright \Lambda \vdash t : L$ be a type derivation. Then in $\text{flow}(\pi)$, each position with underlying atom X is connected either to the main type X of a probabilistic axiom or to an atom X occurring in Λ (in the conclusion of π), by a path in which all positions have underlying atom X .

E COMPLEMENTS TO SECT. 8 (DEALING WITH EVIDENCE)

Let us complete the coin tosses example (Example 8.3), using actual probabilities. Assume that the tossed coin is biased either towards t (bias 0.7), or towards false (bias 0.4). Since we lack knowledge, our prior belief is that either coin could be tossed with equal probability. We want to infer the bias of the coin from the result of the tosses.

EXAMPLE (8.3, CONT.). In the term u of Example 4.10 modeling two tosses of the same (biased) coin

$$u \triangleq \text{let } x = \text{sample}_d \text{ in let } y = !(\mathbf{c}(x)) \text{ in } \langle x, \text{der } y, \text{der } y \rangle$$

we assume $\mathbf{c}(x)$ stands for the conditional expression $\text{case } \langle x \rangle$ of $\{r_1 = \text{bernoulli}_{0.7}; r_2 = \text{bernoulli}_{0.4}\}$ and $\text{sample}_d = \text{bernoulli}_{0.5}$.

Assume that we observe that both the coin tosses yield t , which is encoded as follows (Example 4.10)

$$u' \triangleq \text{letp } \langle x, y_1, y_2 \rangle = u \text{ in } \langle x, \text{obs}(y_1 = t), \text{obs}(y_2 = t) \rangle$$

Let us compare the type derivation π for the term u with the type derivation ρ for the term u' and see how the semantics changes. In both cases, the semantics is the product of the factors associated to the probabilistic axioms.

- $\llbracket \pi \triangleright u : X \otimes Y_1 \otimes Y_2 \rrbracket$. The probabilistic axioms in π (see Example 4.10) are the following

$$\vdash \text{sample}_d : X \diamond \psi \qquad x : X \vdash \mathbf{c}(x) : Y_1 \diamond \psi_1 \qquad x : X \vdash \mathbf{c}(x) : Y_2 \diamond \psi_2$$

where we have annotated the respective interpretations, which are

X	Pr(X)
t	0.5
f	0.5

X	Y _i	Pr(Y _i X)
t	t	0.7
t	f	0.3
f	t	0.4
f	f	0.6

- $\llbracket \rho \triangleright u : X \otimes Y_1^t \otimes Y_2^t \rrbracket$. The probabilistic axioms in ρ (see Example 8.3) are the following three

$$\vdash \text{sample}_d : X \diamond \psi \qquad x : X \vdash \mathbf{c}(x) : Y_1^t \diamond \psi_1^e \qquad x : X \vdash \mathbf{c}(x) : Y_2^t \diamond \psi_2^e$$

where

$$\psi_i^e =$$

X	Y _i	Pr(Y _i = t X)
t	t	0.7
f	t	0.4

Proceeding like in Example 8.2, $\llbracket \rho \rrbracket = \psi_1^e \odot \psi_2^e \odot \psi$ is exactly $\text{Pr}(X, Y_1 = t, Y_2 = t)$. From it, we have $\text{Pr}(Y_1 = t, Y_2 = t) = 0.325$ and $\text{Pr}(X|Y_1 = t, Y_2 = t)$.

X	Y ₁	Y ₂	Pr(X, Y ₁ = t, Y ₂ = t)
t	t	t	0.245
f	t	t	0.08

X	Y ₁	Y ₂	Pr(X Y ₁ = t, Y ₂ = t)
t	t	t	0.753
f	t	t	0.246

Notice how the evidence has increased our confidence that the coins is biased towards t from 0.5 to 0.753.

- $fcts(J) = 1$ if J is conclusion of I-SAMPLE or I-COND.
- Case of J conclusion of a rule with $h \geq 1$ premises: $\frac{J_1 \dots J_h}{J} R$
 - $fcts(J)$ is the number of premises J_i such that $fcts(J_i) \neq 0$.

LEMMA F.2 (NODES VS LEAVES, WEIGHTED). *Let $\pi \triangleright J_\pi$ be a type derivation, and m_π the number of probabilistic axioms in π . Let*

$$r_\pi \triangleq \sum_J (fcts(J) - 1)$$

where J ranges over all judgments in π such that $fcts(J) \neq 0$. Then

$$\begin{aligned} r_\pi &= 0 = m_\pi, & \text{if } fcts(J_\pi) = 0 \\ r_\pi &= m_\pi - 1, & \text{otherwise.} \end{aligned}$$

PROOF. By induction on the structure of π . If π consists of a single axiom, the property holds trivially: for I-VAR $m_\pi = 0 = r_\pi$, for a probabilistic axiom $m_\pi = 1$ and $r_\pi = 0$.

Assume $\pi \triangleright J_\pi$ has last rule R :

$$\frac{\pi_1 \triangleright J_1 \dots \pi_h \triangleright J_h}{\pi \triangleright J_\pi} R$$

- Case $fcts(J_\pi) = 0$ is immediate, observing that $fcts(J_\pi) = 0$ iff $fcts(J_i) = 0$ for each i , and so $fcts(J_\pi) = 0$ iff $m_\pi = 0$ (i.e., the derivation of J_π contains no probabilistic axiom).
- Case $fcts(J_\pi) = k > 0$. By assumption, there are k premises such that $fcts(J_i) \neq 0$. By i.h., for each such premise $r_{\pi_i} = m_{\pi_i} - 1$. Hence $r_\pi = (\sum_1^k r_{\pi_i}) + (k - 1) = (\sum_1^k m_{\pi_i}) - k + (k - 1) = m_\pi - 1$. \square

LEMMA F.3 (MULTIPLICATIONS). *With the same notations as above, the total number of multiplications to compute the semantics of a derivation π is*

$$O(m_\pi \cdot 2^W).$$

PROOF. Immediate consequence of Lemma F.2, recalling that to multiply k factors requires $(k - 1) \cdot 2^W$ multiplications, where w is the number of names in the resulting factor.

Assume we are inductively computing the semantics of π , following Fig. 9. Let $W \leq n_\pi$ be the largest number of names involved in the computation of any factor. To compute the semantics of a sub-derivation $\pi' \triangleright J$ given the interpretation of its premises, we need to compute the product of $fcts(J)$ factors, which has cost 0 if $fcts(J) = 0$, and otherwise has cost at most $(fcts(J) - 1) \cdot 2^W$. So (using Lemma F.2) the total number of multiplications to compute the semantics of π is bounded by

$$r_\pi \cdot 2^W = (m_\pi - 1) \cdot 2^W$$

\square

Counting Additions. The cost of summing out any number of (binary) r.v.s from a factor of size W is $O(2^W)^9$. Notice that, when computing the semantics of π following Fig. 9, we perform a sum out when (in a rule I-LET or I-APP) some names appear in the premisses, but not in the conclusion. By the observations in Appendix D.2, a name can be summed out at most once in the derivation π , and moreover, only main names are summed out. Therefore the number of possible sum outs is bounded by the number m of probabilistic axioms in π . We immediately have the following.

⁹Summing out observed variable does not imply any actual computation, because—as we have already remarked in Sect. 9—the set \mathbb{E} of observed variable does not actually contribute to the number of entries in a factor: $\text{Val}(\widehat{\mathbb{Y}}) \times \text{Val}(\mathbb{E})$ is isomorphic to $\text{Val}(\widehat{\mathbb{Y}})$.

LEMMA F.4 (ADDITIONS). *With the same notations as above, the total number of additions to compute the semantics of a derivation π is*

$$O(m_\pi \cdot 2^W).$$

Remark F.5 (Reduction and Cost of the Semantics). The upper bound in Prop. 9.1 is invariant by reduction (because the number of probabilistic axioms axiom in a type derivation is invariant). Instead, the cost of *inductively* computing the semantics of a type derivation π is *not* stable by reduction, because as the derivation changes, the size W of the largest factor to be inductively computed may grow or decrease. Notice that both cases are possible, so given a term t , the cost of inductively computing the semantics of its normal form may be larger than the cost for t .

F.2 On the Product of Factors vs. Product of Matrices.

We stress how much the product of factors is *different* from the product of matrices. In this difference lies the efficiency of a factors-based semantics w.r.t. to a categorical [Jacobs and Zanasi 2020] or relational [Ehrhard et al. 2014; Ehrhard and Tasson 2019] one, where the tensor product \otimes (which behaves as the tensor product of matrices) plays instead a central role.

Example F.6 (Factors are compact). Let t_1, t_2 be case expressions, respectively encoding two CPTs $\phi^{X_1} = \Pr(X_1|Y_1, Y_2, Y_3)$ and $\phi^{X_2} = \Pr(X_2|Y_1, Y_2, Y_3)$ conditioned to the *same* variables. We can see each ϕ^{X_i} interpreting t_i as a stochastic matrix (of size 2^4). One easily realizes that computing the tensor product of matrices $\phi^{X_1} \otimes \phi^{X_2}$ requires to *compute and store* $2^4 \cdot 2^4 = 2^8$ entries, while the factor product $\phi^{X_1} \circ \phi^{X_2}$ computes 2^5 entries. In a categorical or relational model, to compute the semantics of the term $\langle t_1, t_2 \rangle$ will (in general) require to compute $\phi^{X_1} \otimes \phi^{X_2}$. On this basis, it is easy to build a term t such as the following one

$$\begin{aligned} &\text{let } y_1 = \text{sample}_{d_1} \text{ in let } y'_1 = y_1 \text{ in} \\ &\text{let } y_2 = \text{sample}_{d_2} \text{ in let } y'_2 = y_2 \text{ in} \\ &\text{let } y_3 = \text{sample}_{d_3} \text{ in let } y'_3 = y_3 \text{ in} \\ &\text{let } x_1 = \mathbf{c}^{X_1}_{\langle y_1, y_2, y_3 \rangle} \text{ in} \\ &\text{let } x_2 = \mathbf{c}^{X_2}_{\langle y'_1, y'_2, y'_3 \rangle} \text{ in } \langle x_1, x_2 \rangle. \end{aligned}$$

which encodes a BN over the 5 variables X_1, X_2, Y_1, Y_2, Y_3 , where computing the inductive interpretation of t in a categorical or relational model requires to compute and store 2^8 values. This is somehow weird, since the full joint distribution over 5 variables has size 2^5 . In contrast, as we have stressed in this section, the cost of computing our semantics is never larger than the cost of computing the joint distribution.

G SUBJECT REDUCTION

We strengthen Subject Reduction statement, in order to have the properties we need to prove also the strong normalization of typable terms, the invariance of the factor-based semantics, and the acyclicity of the flow graph of a ground derivation. In the present section and in the following one we use the symbol \mathbf{m} to refer to a (possibly empty) multiset without explicitly naming its elements.

Definition G.1. The measure of a derivation, denoted $\text{meas}(\pi)$, is obtained by counting the number of rules I-LET, I-DER, I-APP, I-LETP occurring in π , giving them weight 1, 1, 2, 3 respectively. All the other rules have 0 weight.

LEMMA G.2. *If $\pi \triangleright \Lambda, \Gamma \vdash v : L$ then Γ is an empty context and $\text{meas}(\pi) = 0$. Moreover π contains no probabilistic axioms, and $\text{flow}(\pi)$ is acyclic.*

PROOF. Since $v : L$, then either $v = x$ or $v = \langle v_1, v_2 \rangle$. We reason by induction on π .

- Case $v = x$. Then $\pi \triangleright \Lambda \vdash x : L$ where $\Lambda = \Lambda', x : L$, and the result immediately follows.
- Case $v = \langle v_1, v_2 \rangle$. By inductive hypothesis the property holds both for $\pi_1 \triangleright \Lambda \vdash v_1 : L_1$ and $\pi_2 \triangleright \Lambda \vdash v_2 : L_2$; then it immediately follows that $\pi \triangleright \Lambda \vdash v : L_1 \otimes L_2$ satisfies the property, as the flow goes towards the axioms in Λ , and towards the conclusion in $L_1 \otimes L_2$.

□

LEMMA G.3 (SPLIT). *Let $\pi \triangleright \Lambda, \Gamma \vdash v : \mathbf{m}_1 \uplus \dots \uplus \mathbf{m}_n$. Then there exist $\pi_i \triangleright \Lambda, \Gamma_i \vdash v : \mathbf{m}_i$ ($1 \leq i \leq n$) such that $\Gamma = \uplus_{i=1}^n \Gamma_i$ and $\text{meas}(\pi) = \sum_{i=1}^n \text{meas}(\pi_i)$. Moreover, if $\pi \triangleright \Lambda, \Gamma \vdash v : []$, then Γ is empty, $\text{meas}(\pi) = 0$ and $\text{flow}(\pi)$ is acyclic.*

PROOF. By observing that the subject can be assigned a multiset type either by a rule I-VAR or by a rule I-BANG; in both cases we can choose a suitable partition of the multiset. Note that $v = !u : []$ can only be obtained by rule I-BANG with $n = 0$ premises, hence Γ is empty and $\text{flow}(\pi)$ is trivially acyclic. □

LEMMA G.4 (SUBSTITUTION LEMMA). *Let $\pi^v \triangleright \Lambda, \Gamma \vdash v : P$ and $\pi \triangleright \Lambda, \Delta, x : P \vdash t : A$. Then there exists $\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash t\{x \leftarrow v\} : A$ such that $\text{meas}(\pi') = \text{meas}(\pi^v) + \text{meas}(\pi)$. Moreover, consider the rewriting:*

$$\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \pi \triangleright \Lambda, \Delta, x : P \vdash t : A}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash t\{x \leftarrow v\} : A} \text{I-LET} \rightsquigarrow \pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash t\{x \leftarrow v\} : A$$

The following properties hold:

1. π^{let} and π' contain the same probabilistic axioms, modulo renaming of free variables;
2. if $\text{flow}(\pi^{\text{let}})$ contains a directed cycle, then $\text{flow}(\pi')$ contains a directed cycle;
3. if there is a directed path between two positions i and j in the conclusion of π^{let} , then there is a path between i and j in the conclusion of π' .

PROOF. By induction on the derivation π , considering its last rule.

- Case I-VAR. Two subcases:
 1. If $t = x$, then $A = P$, that is $\pi \triangleright \Lambda, x : P \vdash x : P$. Observe that Δ is empty. We have:

$$\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{\pi \triangleright \Lambda, x : P \vdash x : P}{\pi \triangleright \Lambda, x : P \vdash x : P} \text{I-VAR}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \vdash x[x \leftarrow v] : P} \text{I-LET} \rightsquigarrow \pi' := \pi^v \triangleright \Lambda, \Gamma \vdash v : P$$

Clearly $\text{meas}(\pi') = \text{meas}(\pi^v) = \text{meas}(\pi^v) + \text{meas}(\pi)$ and the probabilistic axioms contained in π^{let} and π' are the same. Lastly, even if we delete π , the flow remains essentially unaltered, as π contains no cyclic path.

2. If $t = y \neq x$ then $\pi \triangleright \Lambda', x : P, y : A \vdash y : A$, and $\Lambda = \Lambda', y : A$ if A is ground, $\Lambda = \Lambda'$ otherwise. It is immediate to check that $\text{meas}(\pi^v) = 0$, Γ is empty, π^v contains no probabilistic axioms and $\text{flow}(\pi^v)$ is acyclic: indeed, either $P = L$ is ground, and we use Lemma G.2, or $P = []$, and we resort to Lemma G.3. Therefore we have:

$$\frac{\pi^v \triangleright \Lambda \vdash v : P \quad \frac{\pi \triangleright \Lambda', x : P, y : A \vdash y : A}{\pi \triangleright \Lambda', x : P, y : A \vdash y : A} \text{I-VAR}}{\pi^{\text{let}} \triangleright \Lambda', y : A \vdash y[x \leftarrow v] : A} \text{I-LET} \rightsquigarrow \frac{\pi' \triangleright \Lambda', y : A \vdash y : A}{\pi' \triangleright \Lambda', y : A \vdash y : A} \text{I-VAR}$$

Clearly $\text{meas}(\pi') = 0 = \text{meas}(\pi^v) + \text{meas}(\pi)$. Observe that erasing π^v has no effect on the flow, because π^v does not contain cyclic paths; similarly, there are no probabilistic axioms in π^{let} nor in π' , as π^v contains no probabilistic axioms.

- Case I-SAMPLE is similar to the second subcase of I-VAR.

- Case I-COND. Let $A = X$; the only interesting subcase that differs from the second subcase of I-VAR is:

$$\frac{\frac{\pi^v \triangleright \Lambda', y : \mathcal{Y} \vdash y : \mathcal{Y} \quad \text{I-VAR} \quad \frac{\pi \triangleright \Lambda', y : \mathcal{Y}, x : \mathcal{Y} \vdash \mathbf{c}\langle \dots, x, \dots \rangle : X \quad \text{I-COND}}{\pi \triangleright \Lambda', y : \mathcal{Y} \vdash \mathbf{c}\langle \dots, x, \dots \rangle [x \leftarrow y] : X \quad \text{I-LET}}}{\pi^{\text{let}} \triangleright \Lambda', y : \mathcal{Y} \vdash \mathbf{c}\langle \dots, x, \dots \rangle [x \leftarrow y] : X} \quad \rightsquigarrow}{\frac{\pi' \triangleright \Lambda', y : \mathcal{Y} \vdash \mathbf{c}\langle \dots, y, \dots \rangle : X \quad \text{I-COND}}{\pi' \triangleright \Lambda', y : \mathcal{Y} \vdash \mathbf{c}\langle \dots, y, \dots \rangle : X} \quad \text{I-COND}}$$

Note that both Γ and Δ are empty contexts, and that the free variable x has been renamed. Again we have $\text{meas}(\pi') = 0 = \text{meas}(\pi^v) + \text{meas}(\pi)$; moreover the requirements on cyclic paths and probabilistic axioms are trivially satisfied, as π^v contains neither of them.

- Case I-OBS. Similarly to what happens with case I-VAR, we distinguish two subcases. The first one is $\pi \triangleright \Lambda, x : X^b \vdash \text{obs}(x = b) : X^b$, that is $A = P = X^b$; observe that necessarily $\pi^v \triangleright \Lambda \vdash y : X^b$, so that $\pi' \triangleright \Lambda, y : X^b \vdash \text{obs}(y = b) : X^b$ is immediately obtained by variable renaming. The other subcase is $\pi \triangleright \Lambda', x : P, y : X^b \vdash \text{obs}(y = b) : X^b$, that is $\Lambda = \Lambda', y : X^b$; the proof proceeds as in the second subcase of I-VAR, yielding $\pi' \triangleright \Lambda', y : X^b \vdash \text{obs}(y = b) : X^b$. Clearly in both cases one has $\text{meas}(\pi') = 0 = \text{meas}(\pi) + \text{meas}(\pi^v)$, no probabilistic axiom is involved, and the flow, after rewriting π^{let} into π' , is essentially unchanged because π^v contains no cyclic paths.
- Case I-PAIR. Posing $A = L_1 \otimes L_2$, we have:

$$\frac{\frac{\pi^v \triangleright \Lambda \vdash v : P \quad \frac{\pi_1 \triangleright \Lambda, x : P \vdash w_1 : L_1 \quad \pi_2 \triangleright \Lambda, x : P \vdash w_2 : L_2}{\pi \triangleright \Lambda, x : P \vdash \langle w_1, w_2 \rangle : L_1 \otimes L_2} \quad \text{I-PAIR}}{\pi^{\text{let}} \triangleright \Lambda \vdash \langle w_1, w_2 \rangle [x \leftarrow v] : L_1 \otimes L_2} \quad \text{I-LET} \quad \rightsquigarrow}{\frac{\frac{\pi^v \triangleright \Lambda \vdash v : P \quad \Lambda, x : P \vdash w_1 : L_1}{\pi_1^{\text{let}} \triangleright \Lambda \vdash w_1 [x \leftarrow v] : L_1} \quad \text{I-LET} \quad \frac{\pi^v \triangleright \Lambda \vdash v : P \quad \Lambda, x : P \vdash w_2 : L_2}{\pi_2^{\text{let}} \triangleright \Lambda \vdash w_2 [x \leftarrow v] : L_2} \quad \text{I-LET}}{\frac{\frac{\pi_1^{\text{let}} \triangleright \Lambda \vdash w_1 \{x \leftarrow v\} : L_1 \quad \text{i.h.} \quad \pi_2^{\text{let}} \triangleright \Lambda \vdash w_2 \{x \leftarrow v\} : L_2 \quad \text{i.h.}}{\pi' \triangleright \Lambda \vdash \langle w_1, w_2 \rangle \{x \leftarrow v\} : L_1 \otimes L_2} \quad \text{I-PAIR}}{\pi' \triangleright \Lambda \vdash \langle w_1, w_2 \rangle \{x \leftarrow v\} : L_1 \otimes L_2} \quad \text{I-PAIR}}$$

By Lemma G.2, P is either ground or the empty multiset $[\]$; we also deduce that $\text{meas}(\pi^v) = \text{meas}(\pi) = \text{meas}(\pi') = 0$, that both Γ and Δ are empty, and that π^{let} contains no probabilistic axiom. Since all the types involved are ground (with the possible exception of $P = [\]$), both $\text{flow}(\pi^{\text{let}})$ and $\text{flow}(\pi')$ are clearly acyclic, and paths between positions in the conclusion are trivially preserved.

- Case I-ABS. By letting $A = A_1 \multimap A_2$, we have:

$$\frac{\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{\pi_1 \triangleright \Lambda, \Delta, x : P, y : A_1 \vdash u : A_2}{\pi \triangleright \Lambda, \Delta, x : P \vdash \lambda y. u : A_1 \multimap A_2} \quad \text{I-ABS}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash \lambda y. u [x \leftarrow v] : A_1 \multimap A_2} \quad \text{I-LET} \quad \rightsquigarrow}{\frac{\frac{\Lambda, \Gamma \vdash v : P \quad \Lambda, \Delta, x : P, y : A_1 \vdash u : A_2}{\pi_1^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta, y : A_1 \vdash u [x \leftarrow v] : A_2} \quad \text{I-LET}}{\frac{\pi_1^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta, y : A_1 \vdash u \{x \leftarrow v\} : A_2 \quad \text{i.h.}}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash \lambda y. u \{x \leftarrow v\} : A_1 \multimap A_2} \quad \text{I-ABS}}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash \lambda y. u \{x \leftarrow v\} : A_1 \multimap A_2} \quad \text{I-ABS}}$$

Alpha conversion ensures $y \notin \text{dom}(\Gamma)$, and the *i.h.* guarantees there exists $\pi'_1 \triangleright \Lambda, \Gamma \uplus \Delta, y : A_1 \vdash u\{x \leftarrow v\} : A_2$ containing the same probabilistic axioms as π_1^{let} and such that $\text{meas}(\pi'_1) = \text{meas}(\pi^v) + \text{meas}(\pi_1)$; observe that $\text{meas}(\pi_1) = \text{meas}(\pi)$ and $\text{meas}(\pi'_1) = \text{meas}(\pi')$. The *i.h.* also ensures that cycles and paths between positions of the conclusion are preserved when one rewrites π_1^{let} into π'_1 .

- Case I-APP. We have:

$$\begin{array}{c}
 \frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{\pi_1 \triangleright \Lambda, \Delta_1, x : P_1 \vdash u_1 : Q \multimap A \quad \pi_2 \triangleright \Lambda, \Delta_2, x : P_2 \vdash u_2 : Q}{\pi \triangleright \Lambda, \Delta, x : P \vdash u_1 u_2 : A} \text{I-APP}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash u_1 u_2 [x \leftarrow v] : A} \text{I-LET} \\
 \rightsquigarrow \\
 \frac{\frac{\pi_1^v \triangleright \Lambda, \Gamma_1 \vdash v : P_1 \quad \Lambda, \Delta_1, x : P_1 \vdash u_1 : Q \multimap A}{\pi_1^{\text{let}} \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash u_1 [x \leftarrow v] : Q \multimap A} \text{I-LET} \quad \frac{\pi_2^v \triangleright \Lambda, \Gamma_2 \vdash v : P_2 \quad \Lambda, \Delta_2, x : P_2 \vdash u_2 : Q}{\pi_2^{\text{let}} \triangleright \Lambda, \Gamma_2 \uplus \Delta_2 \vdash u_2 [x \leftarrow v] : Q} \text{I-LET}}{\frac{\pi'_1 \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash u_1 \{x \leftarrow v\} : Q \multimap A \quad \text{i.h.} \quad \pi'_2 \triangleright \Lambda, \Gamma_2 \uplus \Delta_2 \vdash u_2 \{x \leftarrow v\} : Q}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash u_1 u_2 \{x \leftarrow v\} : A} \text{I-APP}} \text{i.h.}
 \end{array}$$

We distinguish to subcases:

- case $P = P_1 = P_2$ is ground, i.e. $\pi^v = \pi_1^v = \pi_2^v$. The result follows by Lemma G.2, which guarantees $\text{meas}(\pi^v) = 0$ and Γ empty, and by *i.h.*, ensuring there exist $\pi'_1 \triangleright \Lambda, \Delta_1 \vdash u_1 \{x \leftarrow v\} : Q \multimap A$ and $\pi'_2 \triangleright \Lambda, \Delta_2 \vdash u_2 \{x \leftarrow v\} : Q$ such that $\text{meas}(\pi'_i) = \text{meas}(\pi^v) + \text{meas}(\pi_i) = \text{meas}(\pi_i)$, for $i \in \{1, 2\}$. Indeed, $\text{meas}(\pi') = \text{meas}(\pi'_1) + \text{meas}(\pi'_2) + 2 = \text{meas}(\pi^v) + \text{meas}(\pi)$.
- case $P_1 = \mathbf{m}_1, P_2 = \mathbf{m}_1$ and $P = \mathbf{m}_1 \uplus \mathbf{m}_2$. By Lemma G.3, there exist $\pi_i^v \triangleright \Lambda, \Gamma_i \vdash v : \mathbf{m}_i$ ($i \in \{1, 2\}$) such that $\Gamma = \Gamma_1 \uplus \Gamma_2$ and $\text{meas}(\pi^v) = \text{meas}(\pi_1^v) + \text{meas}(\pi_2^v)$. Moreover, by *i.h.* there are $\pi'_1 \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash u_1 \{x \leftarrow v\} : Q \multimap A$ and $\pi'_2 \triangleright \Lambda, \Gamma_2 \uplus \Delta_2 \vdash u_2 \{x \leftarrow v\} : Q$ such that $\text{meas}(\pi'_i) = \text{meas}(\pi_i^v) + \text{meas}(\pi_i)$ for $i \in \{1, 2\}$. Therefore $\text{meas}(\pi') = \text{meas}(\pi'_1) + \text{meas}(\pi'_2) + 2 = \text{meas}(\pi_1^v) + \text{meas}(\pi_2^v) + \text{meas}(\pi_1) + \text{meas}(\pi_2) + 2 = \text{meas}(\pi^v) + \text{meas}(\pi)$.

In both cases, the *i.h.* ensures that π_1^{let} and π'_i ($i \in \{1, 2\}$) contain the same probabilistic axioms.

Lastly we examine the flow. Observe that:

- any path or cycle in π^v appears in π_1^v or in π_2^v .
- any (possibly cyclic) path in π^{let} can be partitioned into paths in π^v , paths in π_1 , paths in π_2 , the edges between π_1 and π_2 via Q , the paths between π^v and π_i via P_i ($i \in \{1, 2\}$), plus the trivial paths between the border of π^{let} and the borders of π^v, π_1 and π_2 .

Consider a (possibly cyclic) path γ in π^{let} :

- If γ does not use Q , then necessarily it is composed of subpaths $\{\gamma_j\}_{j \in I_1}$ in π_1^v and π_1 , possibly via P_1 , or paths $\{\gamma_j\}_{j \in I_2}$ in π_2^v and π_2 , possibly via P_2 . In the first case, for each γ_j ($j \in I_1$), a path with the same behaviour appears in π_1^{let} and, by *i.h.*, in π'_1 ; consequently it appears also in π' . Similarly for the second case.
 - If γ does use Q , we consider the subpaths using π_1^v and π_1 , possibly via P_1 , and those using π_2^v and π_2 , possibly via P_2 . By *i.h.*, we find all of them in π_1^{let} and π_2^{let} , and therefore in π'_1 and π'_2 . Hence in π' , using the edges via Q , we find again a cycle if γ was a cycle, or a path between the positions i and j in the conclusion of π' , if γ was a path between the positions i and j in the conclusion of π^{let} .
- Case I-LET. We have:

$$\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{\pi_1 \triangleright \Lambda, \Delta_1, x : P_1 \vdash s : Q \quad \pi_2 \triangleright \Lambda, \Delta_2, x : P_2, y : Q \vdash u : A}{\pi \triangleright \Lambda, \Delta, x : P \vdash u[y \leftarrow s] : A} \text{I-LET}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash u[y \leftarrow s][x \leftarrow v] : A} \text{I-LET}$$

$$\begin{array}{c}
\frac{\pi_1^v \triangleright \Lambda, \Gamma_1 \vdash v : P_1 \quad \Lambda, \Delta_1, x : P_1 \vdash s : Q}{\pi_1^{\text{let}} \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash s\{x \leftarrow v\} : Q} \text{I-LET} \quad \frac{\pi_2^v \triangleright \Lambda, \Gamma_2 \vdash v : P_2 \quad \Lambda, \Delta_2, x : P_2, y : Q \vdash u : A}{\pi_2^{\text{let}} \triangleright \Lambda, \Gamma_2 \uplus \Delta_2, y : Q \vdash u\{x \leftarrow v\} : A} \text{I-LET} \\
\sim \\
\frac{\frac{\pi_1^v \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash s\{x \leftarrow v\} : Q}{\pi_1^{\text{let}} \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash s\{x \leftarrow v\} : Q} \text{i.h.} \quad \frac{\pi_2^v \triangleright \Lambda, \Gamma_2 \uplus \Delta_2, y : Q \vdash u\{x \leftarrow v\} : A}{\pi_2^{\text{let}} \triangleright \Lambda, \Gamma_2 \uplus \Delta_2, y : Q \vdash u\{x \leftarrow v\} : A} \text{i.h.}}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash u[y \leftarrow s]\{x \leftarrow v\} : A} \text{I-LET}
\end{array}$$

The reasoning is similar to case I-APP, with two subcases:

- case $P = P_1 = P_2$ is ground. Again Lemma G.2 assures $\text{meas}(\pi^v) = 0$ and Γ empty. By *i.h.* there exist $\pi_1' \triangleright \Lambda, \Delta_1 \vdash s\{x \leftarrow v\} : Q$ and $\pi_2' \triangleright \Lambda, \Delta_2, y : Q \vdash u\{x \leftarrow v\} : A$ such that $\text{meas}(\pi_i') = \text{meas}(\pi^v) + \text{meas}(\pi_i) = \text{meas}(\pi_i)$, for $i \in \{1, 2\}$; the result follows observing that $\text{meas}(\pi') = \text{meas}(\pi_1') + \text{meas}(\pi_2') + 1 = \text{meas}(\pi^v) + \text{meas}(\pi)$.
- case $P_1 = \mathbf{m}_1, P_2 = \mathbf{m}_1$ and $P = \mathbf{m}_1 \uplus \mathbf{m}_2$. We use Lemma G.3 and the inductive hypothesis to obtain $\pi_1' \triangleright \Lambda, \Gamma_1 \uplus \Delta_1 \vdash s\{x \leftarrow v\} : Q$ and $\pi_2' \triangleright \Lambda, \Gamma_2 \uplus \Delta_2, y : Q \vdash u\{x \leftarrow v\} : A$ such that $\text{meas}(\pi_i') = \text{meas}(\pi_i^v) + \text{meas}(\pi_i)$ for $i \in \{1, 2\}$. Therefore $\text{meas}(\pi') = \text{meas}(\pi_1') + \text{meas}(\pi_2') + 1 = \text{meas}(\pi_1^v) + \text{meas}(\pi_2^v) + \text{meas}(\pi_1) + \text{meas}(\pi_2) + 1 = \text{meas}(\pi^v) + \text{meas}(\pi)$.

As in case I-APP, the *i.h.* ensures that π_i^{let} and π_i' ($i \in \{1, 2\}$) contain the same probabilistic axioms; the analysis of the flow is also analogous to the aforementioned case.

- Case I-LETP.

$$\begin{array}{c}
\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{\pi_1 \triangleright \Lambda, x : P_1 \vdash w : K_1 \otimes K_2 \quad \pi_2 \triangleright \Lambda, \Delta, x : P_2, y_2 : K_2, y_1 : K_1, \vdash u : A}{\pi \triangleright \Lambda, \Delta, x : P \vdash u\langle y_1, y_2 \rangle \leftarrow w \rangle : A} \text{I-LETP}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash u\langle y_1, y_2 \rangle \leftarrow w \rangle : A} \text{I-LET} \\
\sim \\
\frac{\pi_1^v \triangleright \Lambda \vdash v : P_1 \quad \Lambda, x : P_1 \vdash w : K_1 \otimes K_2}{\pi_1^{\text{let}} \triangleright \Lambda \vdash w\{x \leftarrow v\} : K_1 \otimes K_2} \text{I-LET} \quad \frac{\pi_2^v \triangleright \Lambda, \Gamma \vdash v : P_2 \quad \Lambda, \Delta, x : P_2, y_1 : K_1, y_2 : K_2 \vdash u : A}{\pi_2^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta, y : K_1 \otimes K_2 \vdash u\{x \leftarrow v\} : A} \text{I-LET} \\
\sim \\
\frac{\frac{\pi_1^v \triangleright \Lambda \vdash w\{x \leftarrow v\} : K_1 \otimes K_2}{\pi_1^{\text{let}} \triangleright \Lambda \vdash w\{x \leftarrow v\} : K_1 \otimes K_2} \text{i.h.} \quad \frac{\pi_2^v \triangleright \Lambda, \Gamma \uplus \Delta, y_2 : K_2, y_1 : K_1 \vdash u\{x \leftarrow v\} : A}{\pi_2^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta, y_2 : K_2, y_1 : K_1 \vdash u\{x \leftarrow v\} : A} \text{i.h.}}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash u\langle y_1, y_2 \rangle \leftarrow w \rangle\{x \leftarrow v\} : A} \text{I-LETP}
\end{array}$$

There are two subcases:

- case $P_1 = P_2 = P$ is ground. Lemma G.2 assures $\text{meas}(\pi^v) = 0$ and Γ empty. By *i.h.* there are $\pi_1' \triangleright \Lambda \vdash w\{x \leftarrow v\} : K_1 \otimes K_2$ and $\pi_2' \triangleright \Lambda, \Delta, y_1 : K_1, y_2 : K_2 \vdash u\{x \leftarrow v\} : A$ such that $\text{meas}(\pi_i') = \text{meas}(\pi^v) + \text{meas}(\pi_i) = \text{meas}(\pi_i)$, for $i \in \{1, 2\}$; observe that $\text{meas}(\pi_1) = \text{meas}(\pi_1') = 0$, therefore $\text{meas}(\pi') = \text{meas}(\pi_2') + 3 = \text{meas}(\pi^v) + \text{meas}(\pi)$.
- case $P_1 = []$ and $P_2 = P$. By Lemma G.3 and the *i.h.* we obtain $\pi_1' \triangleright \Lambda \vdash w\{x \leftarrow v\} : K_1 \otimes K_2$ and $\pi_2' \triangleright \Lambda, \Gamma \uplus \Delta, y_1 : K_1, y_2 : K_2 \vdash u\{x \leftarrow v\} : A$ such that $\text{meas}(\pi_i') = \text{meas}(\pi_i^v) + \text{meas}(\pi_i)$ for $i \in \{1, 2\}$. Remark that $\text{meas}(\pi_1) = \text{meas}(\pi_1') = \text{meas}(\pi_1^v) = 0$, hence $\text{meas}(\pi') = \text{meas}(\pi_2') + 3 = \text{meas}(\pi_2^v) + \text{meas}(\pi_2) + 3 = \text{meas}(\pi^v) + \text{meas}(\pi)$.

The analysis of both the probabilistic axioms and of the flow proceeds as usual.

- Case I-BANG. Let $A = [A_1, \dots, A_n]$; then we have:

$$\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{(\pi_i \triangleright \Lambda, \Delta_i, x : P_i \vdash u : A_i)_{i=1}^n}{\pi \triangleright \Lambda, \Delta, x : P \vdash !u : [A_1, \dots, A_n]} \text{I-BANG}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash !u\{x \leftarrow v\} : [A_1, \dots, A_n]} \text{I-LET}$$

\sim

$$\begin{array}{c}
\left(\frac{\pi_i^v \triangleright \Lambda, \Gamma_i \vdash v : P_i \quad \Lambda, \Delta_i, x : P_i \vdash u : A_i}{\pi_i^{\text{let}} \triangleright \Lambda, \Gamma_i \uplus \Delta_i \vdash u[x \leftarrow v] : A_i} \text{I-LET} \right)_{i=1}^n \\
\rightsquigarrow \\
\frac{\frac{\pi'_i \triangleright \Lambda, \Gamma_i \uplus \Delta_i \vdash u\{x \leftarrow v\} : A_i}_{i=1}^n \text{ i.h.}}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash !u\{x \leftarrow v\} : [A_1, \dots, A_n]} \text{I-BANG}
\end{array}$$

As always, consider the two subcases:

- case $P = P_i$ is ground, i.e. $\pi^v = \pi_i^v$ for all i . By Lemma G.2 we know that $\text{meas}(\pi^v) = 0$ and Γ is an empty context. Moreover, by *i.h.* there exist $\pi'_i \triangleright \Lambda, \Delta_i \vdash u\{x \leftarrow v\} : A_i$ such that $\text{meas}(\pi'_i) = \text{meas}(\pi^v) + \text{meas}(\pi_i) = \text{meas}(\pi_i)$ ($1 \leq i \leq n$).
- case $P_i = \mathbf{m}_i$ and $P = \uplus_{i=1}^n \mathbf{m}_i$. Then by Lemma G.3 there exist $\Lambda, \Gamma_i \vdash v : \mathbf{m}_i$ ($1 \leq i \leq n$) such that $\uplus_{i=1}^n \Gamma_i = \Gamma$ and $\text{meas}(\pi) = \sum_{i=1}^n \text{meas}(\pi_i)$. By *i.h.* there are $\pi'_i \triangleright \Lambda, \Gamma_i \uplus \Delta_i \vdash u\{x \leftarrow v\} : A_i$ such that $\text{meas}(\pi'_i) = \text{meas}(\pi_i^v) + \text{meas}(\pi_i)$ ($1 \leq i \leq n$).

In both cases, *i.h.* guarantees that π_i^{let} and π'_i ($1 \leq i \leq n$) contain the same probabilistic axioms. The analysis of the flow is straightforward, as every path between the positions of P in π^{let} can be recovered via the paths between the positions of P_i in π_i^{let} .

- Case I-DER.

$$\begin{array}{c}
\frac{\pi^v \triangleright \Lambda, \Gamma \vdash v : P \quad \frac{\pi_1 \triangleright \Lambda, \Delta, x : P \vdash u : [A]}{\pi \triangleright \Lambda, \Delta, x : P \vdash \text{der } u : A} \text{I-DER}}{\pi^{\text{let}} \triangleright \Lambda, \Gamma \uplus \Delta \vdash \text{der } u[x \leftarrow v] : [A]} \text{I-LET} \\
\rightsquigarrow \\
\frac{\Lambda, \Gamma \vdash v : P \quad \Lambda, \Delta, x : P \vdash u : [A]}{\pi'_1 \triangleright \Lambda, \Gamma \uplus \Delta \vdash u[x \leftarrow v] : [A]} \text{I-LET} \\
\rightsquigarrow \\
\frac{\pi'_1 \triangleright \Lambda, \Gamma \uplus \Delta \vdash u\{x \leftarrow v\} : [A]}{\pi' \triangleright \Lambda, \Gamma \uplus \Delta \vdash \text{der } u\{x \leftarrow v\} : A} \text{I-DER} \text{ i.h.}
\end{array}$$

By *i.h.* there exists $\pi'_1 \triangleright \Lambda, \Gamma \uplus \Delta \vdash u\{x \leftarrow v\} : [A]$ such that $\text{meas}(\pi'_1) = \text{meas}(\pi^v) + \text{meas}(\pi_1)$; note that $\text{meas}(\pi_1) = \text{meas}(\pi)$ and $\text{meas}(\pi'_1) = \text{meas}(\pi')$. Moreover, *i.h.* guarantees that the probabilistic axioms contained in π_1^{let} and π'_1 are the same, and that both cycles and path between positions of the conclusion are preserved when transforming the former derivation into the latter. □

LEMMA G.5 (SUBJECT REDUCTION). *If $\pi \triangleright \Lambda, \Gamma \vdash t : A$ and $t \mapsto t'$, then $\pi' \triangleright \Lambda, \Gamma \vdash t' : A$. Moreover:*

1. $\text{meas}(\pi) > \text{meas}(\pi')$;
2. π and π' contain the same probabilistic axioms, modulo renaming of free variables;
3. if $\text{flow}(\pi)$ contains a directed cycle, then $\text{flow}(\pi')$ contains a directed cycle;
4. if there is a directed path between two positions i and j in the conclusion of π , then there is a path between i and j in the conclusion of π' .

PROOF. The proof is by induction on the reduction context E in which the reduction takes place. For the sake of space we focus on the base case $E = \langle \cdot \rangle$, the inductive cases being easy to check using the *i.h.*. There is one case per rewriting rule.

- Rule db. In this case we have $t = (\lambda x.s)Su$ and $t' = \langle s[x \leftarrow u] \rangle S$. Thus we proceed by induction on S .
 - Case $S = \langle \cdot \rangle$. The derivation π has shape:

$$\frac{\frac{\Lambda, \Gamma_1, x : P \vdash s : A}{\Lambda, \Gamma_1 \vdash \lambda x.s : P \multimap A} \text{I-ABS} \quad \Lambda, \Gamma_2 \vdash u : P}{\pi \triangleright \Lambda, \Gamma \vdash (\lambda x.s)u : A} \text{I-APP}$$

We conclude with

$$\frac{\Lambda, \Gamma_2 \vdash u : P \quad \Lambda, \Gamma_1, x : P \vdash s : A}{\pi' \triangleright \Lambda, \Gamma \vdash s[x \leftarrow u] : A} \text{I-LET}$$

Observe that $\text{meas}(\pi) = \text{meas}(\pi') + 1$. Clearly the rewriting rule does not affect probabilistic axioms nor the paths between positions in the conclusion; cyclic paths are trivially preserved too.

- Case $S = S'[y \leftarrow r]$. The derivation π is:

$$\frac{\frac{\pi_1 \triangleright \Lambda, \Gamma_1 \vdash r : P \quad \pi_2 \triangleright \Lambda, \Gamma_2, y : P \vdash \langle \lambda x.s \rangle S' : Q \multimap A}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \langle \lambda x.s \rangle S : Q \multimap A} \text{I-LET} \quad \pi_3 \triangleright \Lambda, \Gamma_3 \vdash u : Q}{\pi \triangleright \Lambda, \Gamma \vdash \langle \lambda x.s \rangle S u : A} \text{I-APP}$$

Looking at π , we can safely assume that $y \notin \text{dom}(\Gamma_3)$; then one can build

$$\frac{\pi_2 \triangleright \Lambda, \Gamma_2, y : P \vdash \langle \lambda x.s \rangle S' : Q \multimap A \quad \pi_3 \triangleright \Lambda, \Gamma_3 \vdash u : Q}{\rho \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle \lambda x.s \rangle S' u : A} \text{I-APP}$$

By *i.h.* there exists $\rho' \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle s[x \leftarrow u] \rangle S' : A$ preserving probabilistic axioms and such that $\text{meas}(\rho) > \text{meas}(\rho')$; hence one can build

$$\frac{\pi_1 \triangleright \Lambda, \Gamma_1 \vdash r : P \quad \rho' \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle s[x \leftarrow u] \rangle S' : A}{\pi' \triangleright \Lambda, \Gamma \vdash \langle s[x \leftarrow u] \rangle S : A} \text{I-LET}$$

Note that $\text{meas}(\pi) = \text{meas}(\rho) + \text{meas}(\pi_1) + 1 > \text{meas}(\rho') + \text{meas}(\pi_1) + 1 = \text{meas}(\pi')$. We now examine the flow in detail. Any (possibly cyclic) path γ in π can be partitioned into paths inside π_1, π_2, π_3 , those connecting π_2 and π_3 via Q , the edges connecting π_1 and π_2 via P , and the trivial edges between the border of π and the borders of the three subderivations. Any path in π which does not use P nor Γ_1 appears in ρ and consequently, by *i.h.*, in ρ' . Therefore if π has a cycle or a path between two positions i and j in its conclusion, we are able to recover a cycle or a path between positions i and j also in π' , possibly by using the edges between P in π' .

- Rule *dsusb*. Then $t = s[x \leftarrow \langle v \rangle S]$ and $t' = \langle s[x \leftarrow v] \rangle S$. Again, we proceed by induction on S .
 - $S = \langle \cdot \rangle$. Then π is:

$$\frac{\pi_1 \triangleright \Lambda, \Gamma_1 \vdash v : Q \quad \pi_2 \triangleright \Lambda, \Gamma_2, x : Q \vdash s : A}{\pi \triangleright \Lambda, \Gamma \vdash s[x \leftarrow v] : A} \text{I-LET}$$

and we conclude by Lemma G.4, assuring there exists $\pi' \triangleright \Lambda, \Gamma \vdash s[x \leftarrow v] : A$ such that $\text{meas}(\pi') = \text{meas}(\pi_1) + \text{meas}(\pi_2)$. The same Lemma guarantees that probabilistic axioms, paths between positions in the conclusion and cycles are preserved.

- $S = S'[y \leftarrow r]$. The derivation π has shape:

$$\frac{\frac{\pi_1 \triangleright \Lambda, \Gamma_1 \vdash r : P \quad \pi_2 \triangleright \Lambda, \Gamma_2, y : P \vdash \langle v \rangle S' : Q}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \langle v \rangle S : Q} \text{I-LET} \quad \pi_3 \triangleright \Lambda, \Gamma_3, x : Q \vdash s : A}{\pi \triangleright \Gamma \vdash s[x \leftarrow \langle v \rangle S] : A} \text{I-LET}$$

We start by building

$$\frac{\pi_2 \triangleright \Lambda, \Gamma_2, y : P \vdash \langle v \rangle S' : Q \quad \pi_3 \triangleright \Lambda, \Gamma_3, x : Q \vdash s : A}{\rho \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash s[x \leftarrow \langle v \rangle S'] : A} \text{I-LET}$$

By *i.h.* there exists $\rho' \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle s\{x \leftarrow v\} \rangle S' : A$ preserving probabilistic axioms and such that $\text{meas}(\rho) > \text{meas}(\rho')$. Therefore we can conclude with

$$\frac{\pi_1 \triangleright \Lambda, \Gamma_1 \vdash r : P \quad \rho' \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle s\{x \leftarrow v\} \rangle S' : A}{\pi' \triangleright \Lambda, \Gamma \vdash \langle s\{x \leftarrow v\} \rangle S : A} \text{I-LET}$$

It is easy to check that $\text{meas}(\pi) = \text{meas}(\rho) + \text{meas}(\pi_1) + 1 > \text{meas}(\rho') + \text{meas}(\pi_1) + 1 = \text{meas}(\pi')$. The reasoning on the flow is similar to case db.

- Rule der!. In this case $t = \text{der } !s$ and $t' = s$. We simply have:

$$\frac{\frac{\pi' \triangleright \Gamma \vdash s : A}{\Gamma \vdash !s : [A]} \text{I-BANG}}{\pi \triangleright \Gamma \vdash \text{der } (!s) : A} \text{I-DER}$$

Clearly $\text{meas}(\pi) = \text{meas}(\pi') + 1$, and the probabilistic axioms contained in π and π' are the same. Moreover, since cyclic paths can only be in π' , they are trivially preserved; a similar argument applies to the paths between positions in the conclusion.

- Rule pm. In this case t is $\text{letp } \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle$ in s and t' is $s[x_1 \leftarrow v_1][x_2 \leftarrow v_2]$. The derivation π is

$$\frac{\frac{\Lambda \vdash v_1 : L_1 \quad \Lambda \vdash v_2 : L_2}{\Lambda \vdash \langle v_1, v_2 \rangle : L_1 \otimes L_2} \text{I-PAIR} \quad \Lambda, \Gamma, x_1 : L_1, x_2 : L_2 \vdash s : A}{\pi \triangleright \Lambda, \Gamma \vdash \text{letp } \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle \text{ in } s : A} \text{I-LETP}$$

and it is easy to build

$$\frac{\Lambda \vdash v_2 : L_2 \quad \frac{\Lambda \vdash v_1 : L_1 \quad \Lambda, \Gamma, x_1 : L_1, x_2 : L_2 \vdash s : A}{\Lambda, \Gamma, x_2 : L_2 \vdash s[x_1 \leftarrow v_1] : A} \text{I-LET}}{\pi' \triangleright \Lambda, \Gamma \vdash s[x_1 \leftarrow v_1][x_2 \leftarrow v_2] : A} \text{I-LET}$$

Observe that $\text{meas}(\pi) = \text{meas}(\pi') + 1$. Since we only need to rearrange the subderivations, it is easy to verify that probabilistic axioms, the paths between positions in the conclusion, and cyclic paths, are indeed preserved. \square

H SUBJECT EXPANSION

We strengthen the statement of Subject Expansion, in order to have the elements we need to prove also Prop. 4.7.

LEMMA H.1 (ANTI-SPLIT). *Let $\pi_i \triangleright \Lambda, \Gamma_i \vdash v : \mathbf{m}_i$ ($1 \leq i \leq n$). Then there exists $\pi \triangleright \Lambda, \Gamma \vdash v : \mathbf{m}$ such that $\Gamma = \biguplus_{i=1}^n \Gamma_i$ and $\mathbf{m} = \biguplus_{i=1}^n \mathbf{m}_i$. Moreover, if all π_i ($1 \leq i \leq n$) are unique, then π is unique.*

PROOF. We distinguish two cases:

- $v = x$. Then $\pi_i \triangleright \Lambda, x : \mathbf{m}_i \vdash x : \mathbf{m}_i$, and there is only one way to build $\pi \triangleright \Lambda, x : \mathbf{m} \vdash x : \mathbf{m}$, by using rule var. Observe that Γ is empty.
- $v = !t$. Let $\mathbf{m}_i = [A_{i1}, \dots, A_{ik_i}]$; then π_i is obtained from k_i subderivations of shape $\pi_{ij} \triangleright \Lambda, \Gamma_{ij} \vdash t : A_{ij}$ ($1 \leq j \leq k_i$), and it is possible to build π by combining all of the π_{ij} via rule I-BANG. Therefore the uniqueness of all π_i implies the uniqueness of π . \square

LEMMA H.2 (ANTI-SUBSTITUTION). *If $\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash t\{x \leftarrow v\} : A$, then there exist $\pi^t \triangleright \Lambda, \Gamma, x : P \vdash t : A$ and $\pi^v \triangleright \Lambda, \Delta \vdash v : P$. Moreover, if π is unique, both π^t and π^v are unique.*

PROOF. By induction on t .

1. Case $t = x$. Then $P = A$, the derivation π^t is necessarily $\pi^t \triangleright \Lambda, x : A \vdash x : A$ (observe that Γ is empty), and $\pi^v = \pi$.
2. Case $t = y \neq x$. Then π^t is necessarily $\pi^t \triangleright \Lambda', y : A \vdash y : A$ where $\Lambda = \Lambda', y : A$ if A is ground, $\Lambda = \Lambda'$ otherwise. We distinguish two subcases:
 - $\Lambda' = \Lambda'', x : P$, i.e. P is ground. Then by Lemma G.2 Δ is empty; observe that there is a unique way of constructing $\pi^v \triangleright \Lambda \vdash v : P$, using rules I-VAR and I-PAIR only.
 - otherwise $P = []$, therefore $v = !u$ and $\pi^v \triangleright \Lambda \vdash v : P$ can only be obtained by rule I-BANG posing $n = 0$.
3. Case $t = \text{sample}_d$. Similar to case (2).
4. Case $t = \mathbf{c}\langle y_1, \dots, y_n \rangle$ where $x = y_i$ ($1 \leq i \leq n$). This scenario is similar to case (1), with $\pi^t \triangleright \Lambda, x : \mathcal{Y}_i \vdash \mathbf{c}\langle y_1, \dots, y_n \rangle : \mathcal{X}$ and $\pi^v \triangleright \Lambda \vdash v : \mathcal{Y}_i$, where $v = z$ is a variable.
5. Case $t = \mathbf{c}\langle y_1, \dots, y_n \rangle$ where $x \neq y_i$ ($1 \leq i \leq n$). Similar to case (2).
6. Case $t = \text{obs}(x = b)$. Similar to case (1): necessarily $\pi^t \triangleright \Lambda, x : \mathcal{X}^b \vdash \text{obs}(x = b) : \mathcal{X}^b$ and $\pi^v \triangleright \Lambda \vdash v : \mathcal{X}^b$, where $v = z$ is a variable.
7. Case $t = \text{obs}(y = b)$ where $y \neq x$. Necessarily $\pi^t \triangleright \Lambda, x : P \vdash \text{obs}(y = b) : \mathcal{X}^b$, and for $\pi^v \triangleright \Lambda \vdash v : P$ we distinguish the subcases P ground or $P = []$, as in case (2).
8. Case $t = \langle w_1, w_2 \rangle$. Then π has shape:

$$\frac{\Lambda \vdash w_1 \{x \leftarrow v\} : L_1 \quad \Lambda \vdash w_2 \{x \leftarrow v\} : L_2}{\pi \triangleright \Lambda \vdash \langle w_1, w_2 \rangle \{x \leftarrow v\} : A} \text{ I-PAIR}$$

Observe that since $A = L_1 \otimes L_2$, all exponential contexts are empty (by Lemma G.2). By *i.h.* there are $\pi^{w_1} \triangleright \Lambda, x : P \vdash w_1 : L_1$ and $\pi^{w_2} \triangleright \Lambda \vdash v : P$, together with $\pi^{w_2} \triangleright \Lambda, x : P \vdash w_2 : L_2$ and $\pi_2^v \triangleright \Lambda \vdash v : P$; remark that P is ground (again, by Lemma G.2). Since all the aforementioned derivations are built from rules I-VAR and I-PAIR only, they are necessarily unique. Clearly π^t is obtained from π^{w_1} and π^{w_2} by rule I-PAIR, and $\pi^v = \pi_1^v = \pi_2^v$.

9. Case $t = \lambda y. u$. The derivation π has shape:

$$\frac{\Lambda, \Gamma \uplus \Delta, y : Q \vdash u \{x \leftarrow v\} : B}{\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash \lambda y. u \{x \leftarrow v\} : A} \text{ I-ABS}$$

where $A = Q \multimap B$. Note that, by definition of substitution, y cannot occur free in v ; therefore the *i.h.* assures there exist $\pi^u \triangleright \Lambda, \Gamma, y : Q, x : P \vdash u : B$ and $\pi^v \triangleright \Lambda, \Delta \vdash v : P$. One can easily construct π^t starting from π^u , via rule I-ABS. Lastly, note that π unique implies its premise is unique; in turn, by *i.h.*, this implies π^u (and consequently π^t) and π^v are unique.

10. Case $t = su$. Then π is:

$$\frac{\Lambda, \Gamma_1 \uplus \Delta_1 \vdash s \{x \leftarrow v\} : Q \multimap A \quad \Lambda, \Gamma_2 \uplus \Delta_2 \vdash u \{x \leftarrow v\} : Q}{\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash su \{x \leftarrow v\} : A} \text{ I-APP}$$

By *i.h.* there are $\pi^s \triangleright \Lambda, \Gamma_1, x : P_1 \vdash s : Q \multimap A$ and $\pi_1^v \triangleright \Lambda, \Delta_1 \vdash v : P_1$, together with $\pi^u \triangleright \Lambda, \Gamma_2, x : P_2 \vdash u : Q$ and $\pi_2^v \triangleright \Lambda, \Delta_2 \vdash v : P_2$. Clearly we can build the π^t from π^s and π^u via rule I-APP; for π^v we distinguish two cases:

- If $P = P_1 = P_2$ is ground, then Δ is empty; consequently $\pi^v = \pi_1^v = \pi_2^v$.
- Otherwise $P = P_1 \uplus P_2$ is a multiset; then by Lemma H.1 we can build $\pi^v \triangleright \Lambda, \Delta \vdash v : P$, where $\Delta = \Delta_1 \uplus \Delta_2$, starting from π_1^v and π_2^v .

Remark that π unique means that both its premises are unique; in turn, by *i.h.*, this implies π^s , π^u (and consequently π^t), π_1^v , π_2^v (and consequently π^v) are unique.

11. Case $t = \text{let } y = u \text{ in } s$. The derivation π has shape:

$$\frac{\Lambda, \Gamma_1 \uplus \Delta_1 \vdash u \{x \leftarrow v\} : Q \quad \Lambda, \Gamma_2 \uplus \Delta_2, y : Q \vdash s \{x \leftarrow v\} : A}{\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash (\text{let } y = u \text{ in } s) \{x \leftarrow v\} : A} \text{ I-LET}$$

Observe that y cannot occur free in v , by definition of substitution. Then by *i.h.* there are $\pi^u \triangleright \Lambda, \Gamma_1, x : P_1 \vdash u : Q$ and $\pi^v \triangleright \Lambda, \Delta_1 \vdash v : P_1$, together with $\pi^s \triangleright \Lambda, \Gamma_2, y : Q, x : P_2 \vdash s : A$ and $\pi_2^v \triangleright \Lambda, \Delta_2 \vdash v : P_2$. Again, π^t is obtained from π^u and π^s via rule I-LET; for what concerns π^v and the claims about uniqueness, the same considerations we made in case (10) apply.

12. Case $t = \text{letp } \langle y_1, y_2 \rangle = w$ in s . The derivation π is:

$$\frac{\Lambda \vdash w\{x \leftarrow v\} : L_1 \otimes L_2 \quad \Lambda, \Gamma \uplus \Delta, y_1 : L_1, y_2 : L_2 \vdash s\{x \leftarrow v\} : A}{\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash (\text{letp } \langle y_1, y_2 \rangle = w \text{ in } s)\{x \leftarrow v\} : A} \text{I-LETP}$$

By definition of substitution, y_1 and y_2 cannot occur free in v . Then by *i.h.* there are $\pi^w \triangleright \Lambda, x : P_1 \vdash w : L_1 \otimes L_2$ and $\pi_1^v \triangleright \Lambda \vdash v : P_1$, together with $\pi^s \triangleright \Lambda, \Gamma, y_1 : L_1, y_2 : L_2, x : P_2 \vdash s : A$ and $\pi_2^v \triangleright \Lambda, \Delta \vdash v : P_2$. Clearly π^t is obtained from π^w and π^s via rule I-LETP. Regarding π^v and uniqueness, the reasoning is again similar to case (10), the only relevant difference being that either $P = P_1 = P_2$ is ground or $P_1 = []$ and $P = P_2$.

13. Case $t = !u$. The derivation π is:

$$\frac{(\Lambda, \Gamma_i \uplus \Delta_i \vdash u\{x \leftarrow v\} : A_i)_{i=1}^n}{\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash (!u)\{x \leftarrow v\} : A} \text{I-BANG}$$

where $\Gamma \uplus \Delta = \uplus_{i=1}^n (\Gamma_i \uplus \Delta_i)$ and $A = [A_1, \dots, A_n]$. By *i.h.* we have $\pi_i^u \triangleright \Lambda, \Gamma_i, x : P_i \vdash u : A_i$ and $\pi_i^v \triangleright \Lambda, \Delta_i \vdash v : P_i$ for $1 \leq i \leq n$. Clearly π^t can be obtained from all π_i^u ($1 \leq i \leq n$) by rule I-BANG, while for π^v we follow the same reasoning of case (10):

- If $P = P_i$, i.e. P is ground, then Δ is empty; consequently $\pi^v = \pi_i^v$ for all i .
- Otherwise $P = \uplus_{i=1}^n P_i$ is a multiset; then by Lemma H.1 we can build $\pi^v \triangleright \Lambda, \Delta \vdash v : P$, where $\Delta = \uplus_{i=1}^n \Delta_i$, starting from all π_i^v ($1 \leq i \leq n$).

The result on uniqueness follows by generalizing the argument presented in case (10) to n premises. In the special case $n = 0$, the derivation $\pi^t \triangleright \Lambda, x : P \vdash !u : []$ must be obtained by using no premises; therefore both π^t and $\pi^v \triangleright \Lambda \vdash v : P$ (observe that P is necessarily ground) are uniquely determined.

14. Case $t = \text{der } u$. Then π has shape:

$$\frac{\Lambda, \Gamma \uplus \Delta \vdash u\{x \leftarrow v\} : [A]}{\pi \triangleright \Lambda, \Gamma \uplus \Delta \vdash (\text{der } u)\{x \leftarrow v\} : A} \text{I-DER}$$

The result immediately follows by *i.h.*: indeed, there exist $\pi^u \triangleright \Lambda, \Gamma, x : P \vdash u : A$, from which we obtain π^t via rule I-DER, and $\pi^v \triangleright \Lambda, \Delta \vdash v : P$. The considerations on uniqueness are analogous to case (9). □

THEOREM H.3 (SUBJECT EXPANSION). *If $\pi' \triangleright \Lambda, \Gamma \vdash t' : A$ and $t \mapsto t'$, then there exists $\pi \triangleright \Lambda, \Gamma \vdash t : A$. Moreover, if π' is unique, then π is unique.*

PROOF. The proof is by induction on the reduction context E in which the reduction takes place. Here we examine the base case, in which the reduction context is empty; the inductive cases follow by *i.h.*. There is one subcase per rewriting rule.

- Rule db. We have $t = (\lambda x.s)Su$ and $t' = \langle s[x \leftarrow u] \rangle S$; we proceed by induction on S .
 - Case $S = \langle \cdot \rangle$. The derivation π' has shape:

$$\frac{\Lambda, \Gamma_2 \vdash u : P \quad \Lambda, \Gamma_1, x : P \vdash s : A}{\pi' \triangleright \Lambda, \Gamma \vdash s[x \leftarrow u] : A} \text{I-LET}$$

clearly it is possible to build π as follows:

$$\frac{\frac{\Lambda, \Gamma_1, x : P \vdash s : A}{\Lambda, \Gamma_1 \vdash \lambda x.s : P \multimap A} \text{I-ABS} \quad \Lambda, \Gamma_2 \vdash u : P}{\pi \triangleright \Lambda, \Gamma \vdash (\lambda x.s)u : A} \text{I-APP}$$

Observe that π' unique means that the two premises of the I-LET rule are unique; from this we immediately get that π is unique too.

- Case $S = S'[y \leftarrow r]$. The derivation π' has shape:

$$\frac{\Lambda, \Gamma_1 \vdash r : P \quad \rho' \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle s[x \leftarrow u] \rangle S' : A}{\pi' \triangleright \Lambda, \Gamma \vdash \langle s[x \leftarrow u] \rangle S : A} \text{I-LET}$$

By *i.h.* there exists a derivation ρ of shape:

$$\frac{\Lambda, \Gamma_2, y : P \vdash \langle \lambda x.s \rangle S' : Q \multimap A \quad \Lambda, \Gamma_3 \vdash u : Q}{\rho \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle \lambda x.s \rangle S' u : A} \text{I-APP}$$

where we can safely assume that $y \notin \text{dom}(\Gamma_3)$ by hypothesis of rule db. Therefore one can construct π as follows:

$$\frac{\frac{\Lambda, \Gamma_1 \vdash r : P \quad \Lambda, \Gamma_2, y : P \vdash \langle \lambda x.s \rangle S' : Q \multimap A}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \langle \lambda x.s \rangle S : Q \multimap A} \text{I-LET} \quad \Lambda, \Gamma_3 \vdash u : Q}{\pi \triangleright \Lambda, \Gamma \vdash \langle \lambda x.s \rangle S u : A} \text{I-APP}$$

Observe that π' unique means that its two premises, one of which is ρ' , are unique; this by *i.h.* implies that ρ is unique, and the uniqueness of π follows.

- Rule dsub. Then $t = s[x \leftarrow \langle v \rangle S]$ and $t' = \langle s\{x \leftarrow v\} \rangle S$. Again, we proceed by induction on S .
 - $S = \langle \cdot \rangle$. Starting from $\pi' \triangleright \Lambda, \Gamma \vdash s\{x \leftarrow v\} : A$, Lemma H.2 guarantees there exist π_1 and π_2 from which we can construct π :

$$\frac{\pi_1 \triangleright \Lambda, \Gamma_1 \vdash v : Q \quad \pi_2 \triangleright \Lambda, \Gamma_2, x : Q \vdash s : A}{\pi \triangleright \Lambda, \Gamma \vdash s[x \leftarrow v] : A} \text{I-LET}$$

The same Lemma provides the result about uniqueness.

- $S = S'[y \leftarrow r]$. Then π' is:

$$\frac{\Lambda, \Gamma_1 \vdash r : P \quad \rho' \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash \langle s\{x \leftarrow v\} \rangle S' : A}{\pi' \triangleright \Lambda, \Gamma \vdash \langle s\{x \leftarrow v\} \rangle S : A} \text{I-LET}$$

and by *i.h.* there exists ρ :

$$\frac{\Lambda, \Gamma_2, y : P \vdash \langle v \rangle S' : Q \quad \Lambda, \Gamma_3, x : Q \vdash s : A}{\rho \triangleright \Lambda, \Gamma_2 \uplus \Gamma_3, y : P \vdash s[x \leftarrow \langle v \rangle S'] : A} \text{I-LET}$$

where $y \notin \text{dom}(\Gamma_3)$ by hypothesis of rule dsub. Hence we conclude by building:

$$\frac{\frac{\Lambda, \Gamma_1 \vdash r : P \quad \Lambda, \Gamma_2, y : P \vdash \langle v \rangle S' : Q}{\Lambda, \Gamma_1 \uplus \Gamma_2 \vdash \langle v \rangle S : Q} \text{I-LET} \quad \Lambda, \Gamma_3, x : Q \vdash s : A}{\pi \triangleright \Gamma \vdash s[x \leftarrow \langle v \rangle S] : A} \text{I-LET}$$

The reasoning about uniqueness is similar to case db.

- Rule der!. In this case $t = \text{der } !s$ and $t' = s$. Starting from $\pi' \triangleright \Lambda, \Gamma \vdash s : A$, we can easily build:

$$\frac{\frac{\pi' \triangleright \Lambda, \Gamma \vdash s : A}{\Lambda, \Gamma \vdash !s : [A]} \text{I-BANG}}{\pi \triangleright \Lambda, \Gamma \vdash \text{der } (!s) : A} \text{I-DER}$$

One immediately sees that π' unique implies π unique.

- Rule pm. In this case t is $\text{let } \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle$ in s and t' is $s[x_1 \leftarrow v_1][x_2 \leftarrow v_2]$. The derivation π' has shape:

$$\frac{\frac{\Lambda \vdash v_1 : L_1 \quad \Lambda, \Gamma, x_1 : L_1, x_2 : L_2 \vdash s : A}{\Lambda, \Gamma, x_2 : L_2 \vdash s[x_1 \leftarrow v_1] : A} \text{I-LET}}{\Lambda \vdash v_2 : L_2} \text{I-LET} \quad \frac{}{\pi' \triangleright \Lambda, \Gamma \vdash s[x_1 \leftarrow v_1][x_2 \leftarrow v_2] : A} \text{I-LET}$$

By using the very same subderivations, and only changing the rules, it is easy to obtain π :

$$\frac{\frac{\Lambda \vdash v_1 : L_1 \quad \Lambda \vdash v_2 : L_2}{\Lambda \vdash \langle v_1, v_2 \rangle : L_1 \otimes L_2} \text{I-PAIR} \quad \Lambda, \Gamma, x_1 : L_1, x_2 : L_2 \vdash s : A}{\pi \triangleright \Lambda, \Gamma \vdash \text{letp } \langle x_1, x_2 \rangle = \langle v_1, v_2 \rangle \text{ in } s : A} \text{I-LETP}$$

Again, it easy to check that π' unique implies π unique.

□

Received 2023-07-11; accepted 2023-11-07