View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Illinois Digital Environment for Access to Learning and Scholarship Repository

First-order Probabilistic Inference Revisited

DRAFT

Rodrigo de Salvo Braz
braz@uiuc.edu

Dan Roth
danr@cs.uiuc.edu

Eyal Amir
eyal@cs.uiuc.edu

Abstract

Following ideas in Poole [Poo03], which we correct, formalize and extend, this paper presents the first provable algorithm for reasoning with probabilistic first-order representations at the *lifted* level. Specifically, the algorithm automates the process of probabilistic reasoning about populations of individuals, their properties and the relations between them, without the need to ground the probabilistic knowledge base. The algorithm makes use of unification to guide an interleaving of variable ordering and first-order variable elimination. Importantly, our contribution includes the formalization of concepts necessary to reason about the algorithm's correctness and its correctness proof.

1 Introduction

In a variety of applications one may want to represent and reason with respect to multiple objects, multiple properties of them and relations among them. Examples include reasoning about collections of papers and authors, reasoning about populations and medical conditions they have or extracting information about terrorist activities and perpetrators from a collection of news articles.

Popular representations of probability distributions over sets of objects, such as Bayes nets and random Markov fields are zeroth-order representations and do not allow, in principle, representing and reasoning without making each individual and each property of an individual into a separate node. In the

past decade there has been intensive work attempting to lift probabilistic representation to first-order representations, borrowing ideas from first-order predicate logic and logic programming [NH95, NS92, Jae97, KR00, KP98, Poo93], mostly based conceptually on Halpern's Probabilistic Logic framework [Hal90]. These methods involve the representation of probabilistic assertions with predicates, logical variables, constant symbols and quantification. First-order probabilistic representations allow for a compact specification of models, less redundancy, and to express queries that quantify over populations or properties, such as the probability that *any* disease kills someone in a specific population, or of *someone* in a population having the measles.

While representations of first-order probabilistic models have been studied extensively in the last few years, there has been little progress in inference with it. Specifically, inference with such a representation still requires grounding – the creation of the underlying propositional network with one node for each property of each object. The most notable exceptions to this are first-order probabilistic inference [Poo03], which is the topic of this paper, and SPOOK [PKMT99], which uses caching in order to avoid repeated grounding. Unfortunately, there has been little study on the application of this caching method to general DAGs, and the savings due to this caching method are not easy to measure or to compare with.

Poole [Poo03] points out that full grounding is unnecessary, especially when we have large subgroups of individuals on which we have identical knowledge. In these cases, it is possible to abstract away from individuals in a principled manner, avoiding grounding. Poole's method uses the insight that computing posteriors and marginals via variable elimination [ZP94] can benefit from *lifting*: we can eliminate sets of random variables, if those are described in a compact parameterized form, which we call a *parameterized random variable*.

To clarify some of the representational and inferential issues, consider the following example, modeling the spreading of 100 possible diseases in a population of 1,000,000 people (with similar syntax to that in [Poo03], explained below):

$$P(epidemic(measles)) = 0.05$$
$$\forall\ D \in disease; D \neq measles \rightarrow P(epidemic(D)) = 0.01$$
$$P(sick(measles, mary)) = 0.9$$
$$\forall\ X \in person, \forall\ D \in disease; X \neq mary \rightarrow P(sick(D, X)|epidemic(D)) = 0.4,$$
$$P(sick(D, X)|\neg epidemic(D)) = 0.001$$

Figure 1: Computing an answer to query $P(deathBy(t)|epidemic(t))$ from our example with a single disease *typhus* (t) and a million people will be expensive for the propositional grounded model (a) as it has a large tree-width, but cheap for the lifted model (b) since it is a linear graph.

$\forall\ X \in person, \forall\ D \in disease; \phi(deathBy(D), sick(D, X))$

This model declares that every disease has a probability 0.01 of being an epidemic, except for measles, for which it is 0.05, that any person has a probability 0.4 of becoming sick of an epidemic disease and 0.001 of other diseases, and that the probability of a person dying of a disease is related to the person having that disease by a potential function ϕ. [1]

An example of a model that benefits from lifted inference can be seen at Fig. 1. Arguably, one can set up a linear-graph propositional model doing the same computation as the lifted model shown, and in fact this is ultimately what our algorithm does internally. However, such a model would not directly represent the first-order model and would require manual encoding and decoding of information, whereas our algorithm does that automatically.

This paper's first contribution is to formalize the ideas for lifted first-order probabilistic inference by Poole which, while containing insightful ideas, has been presented without a full formalization and proof of correctness. This formalization reveals that Poole's method is incorrect, and the origins of its flaws. The algorithm presented there is restricted to eliminating a single parameterized variable at each step; however, sometimes there are no orderings of such elimination for which this results in a correct answer; there is a need to allow for the elimination of groups of variables. In fact, even if one allows the elimination of groups of parameterized variables, only a small fraction of

[1] This paper refers to undirected models expressed by way of potential functions on sets of variables [Pea91]. When these potential functions are conditional probabilities on parameterized random variables, however, we express them as such. In this example, the potential function ϕ on $deathBy(D), sick(D, X)$ is the only one that is not a conditional probability on parameterized variables (the conditional probability of $deathBy(D)$ depends on *all* instances of $sick(D, X)$).

these orderings are valid. This seems to be a fundamental difference between propositional and first-order variable elimination. For the query shown in Fig. 1, Poole's method outputs an incorrect answer if the first variable eliminated is either $epidemic(t)$ or $deathBy(t)$. Our work abstracts a necessary and sufficient condition for determining if a group of parameterized variables can be eliminated. Finally, we show constructively that a valid ordering always exists.

An implementation of the algorithm can be found at
`http://l2r.cs.uiuc.edu/~cogcomp`.

2 Examples and motivation

In the section we present, through examples, the main ideas behind our claim that Poole's method is incorrect, how to fix the problem and the corrected algorithm.

2.1 First-order probabilistic inference

Consider the example given in the introduction: $person$ and $disease$ are types, X and D are typed logical variables, $measles$ and $mary$ are constant terms, $sick(measles, mary)$ is a ground random variable, $sick(D, X)$ is a parameterized random variable (of which D and X are the parameters), $X \neq mary$ is a constraint system (which can involve many of the assertion's logical variables), and $P(epidemic(D))$ and $\phi(deathBy(D), sick(D, X))$ are potential functions on parameterized random variables.

A probabilistic assertion consists of a set of universally quantified typed logical variables, a constraint system on them, and a potential function on parameterized random variables. It stands for all the instantiated assertions obtained by grounding its parameterized random variables. This grounding is determined by assignments to the logical variables that satisfy the constraint system. A first-order probabilistic assertion is therefore a compact way of expressing a large number of structurally identical propositional probabilistic assertions.

Because such a first-order probabilistic model has a corresponding propositional model, we can think of its semantics as the semantics of that proposi-

tional model. We can therefore express the joint distribution induced by it on the set A of all ground random variables it involves (either explicitly or through a parameterized random variable) as a normalization of the products of all the instantiated potential functions. For our original example, this is

$$\begin{aligned} P(A) \quad \propto \quad & P(epidemic(measles)) \times P(sick(measles, mary)) \\ & \times \Big[\prod_{D \neq measles} P(epidemic(D)) \Big] \times \Big[\prod_{D} \phi(deathBy(D)) \Big] \\ & \times \Big[\prod_{X, D \neq mary} P(sick(D, X) | epidemic(D)) \Big]. \end{aligned}$$

2.2 Poole's method

We now give an example of Poole's method and identify situations in which it is incorrect. The method considers the probabilistic model in its first-order form (no grounding is done), and eliminates atoms (parametrized random variables) that are not present in the query until only the query variables remain. To illustrate this, we use a simplified version of our original example, for which we have the query $P(sick(cold, mary), sick(cold, john))$:

$\forall\ X \in person, \forall\ D \in disease;$
$$P(sick(D, X) | epidemic(D)) = 0.4$$
$$P(sick(D, X) | \neg epidemic(D)) = 0.001$$
$\forall\ D \in disease; P(epidemic(D)) = 0.01$

We will use this example throughout the paper and abbreviate *epidemic*, *cold*, *john* and *mary* to *epi*, *c*, *j* and *m*, respectively. We also assume that X and D will always be of types *person* and *disease*.

A possible first parameterized variable to be eliminated is $epi(D)$, The method replaces the assertions in which $epi(D)$ occurs by a new one which is the result of summing it out: [2]

$$\forall\ X \in person,\ D \in disease;$$
$$P(sick(D, X)) = \sum_{epi(D) \in \{false, true\}} P(epi(D)) P(sick(D, X) | epi(D)) = 0.00499. \quad (1)$$

[2] In fact, [Poo03] indicates that we can do this two assertions at a time, but this is incorrect; just like with propositional variable elimination, all assertions involving the eliminated variables must be used.

This new assertion defines the marginal distribution on the remaining random variables:

$$P(\{sick(D,X) : D \in disease, X \in person\}) = \prod_{D,X} P(sick(D,X)). \quad (2)$$

This marginal implies that the ground instances of $sick(D,X)$ are independent once we eliminate $epi(D)$, so $P(sick(c,j), sick(c,m)) = 0.00499 \times 0.00499 = 0.0000249001$.

This solution, however, is incorrect (the reasons are given in 2.4). For comparison, consider the grounding of the model and observe that $sick(cold, mary)$ and $sick(cold, john)$ are two random variables with a single parent, $epi(cold)$. We get that

$$\begin{aligned} &P(sick(c,m), sick(c,j)) \\ &\qquad = \sum_{epi(c)} P(sick(c,m), sick(c,j)|epi(c))P(epi(c)) \\ &\qquad = \sum_{epi(c)} P(sick(c,m)|epi(c))P(sick(c,j)|epi(c))P(epi(c)) \quad (3) \\ &\qquad = 0.4 \times 0.4 \times 0.001 + 0.001 \times 0.001 \times 0.99 = 0.00160099 \end{aligned}$$

2.3 A condition for first-order variable elimination

The reason for the discrepancy seen in the previous subsection is that the parameterized variable being eliminated does not contain, in some sense, the logical variables X and D occurring in the assertions. To explain why this is an important condition, it is instructive to examine, first, an instance in which the method does work, within this same example. This should provide some insight into the method and as to why such a condition is necessary.

Note that the final step in the calculation of $P(sick(cold, mary), sick(cold, john))$ was not a variable elimination step, but simply an observation of an obvious independence expressed by this joint distribution. Algorithmically, this can be done by another variable elimination. Now, however, the situation is slightly subtler since *some* random variables of the form $sick(D,X)$ are in the query (only two, actually: $sick(cold, mary)$

and $sick(cold, john)$) and some are not. We can describe the ones to be eliminated with the constraint $D \neq cold \vee (X \neq mary \wedge X \neq john)$. Instead of directly presenting the answer obtained by Poole's method, we detail its derivation so as to give insight into its working.

We define the instantiations on (D, X) *not* in the query to be $NQ = \{(D, X) : D \neq c \vee (X \neq m \wedge X \neq j\}$, and the ones in the query as $Q = \{(D, X) : (D, X) \notin NQ\} = \{(cold, mary), (cold, john)\}$. We also define $sick_{\mathcal{S}}(D, X) = \{sick(D, X) : (D, X) \in \mathcal{S}\}$ for $\mathcal{S}$ either NQ or Q.

Given the distribution (2) on the instances of $sick(D, X)$, the answer to query $sick(cold, mary), sick(cold, john)$ is given by another marginalization

$$\begin{aligned} P(sick(c, m), sick(c, j)) &= \sum_{sick_{NQ}(D,X)} P(\{sick(D, X) : D \in disease, X \in person\}) \\ &= \sum_{sick_{NQ}(D,X)} \prod_{(D,X)} P(sick(D, X)) \end{aligned}$$

The summation is over all groundings of all eliminated variables, which we want to avoid. To do that, we can first split the product and factor out the assertions on Q:

$$\sum_{sick_{NQ}(D,X)} \prod_{(D,X)} P(sick(D, X)) = \Big[\prod_{(D,X)\in Q} P(sick(D, X))\Big]\Big[\sum_{sick_{NQ}(D,X)} \prod_{(D,X)\in NQ} P(sick(D, X))\Big]$$

The crucial step in eliminating the grounding summation takes place now, when we observe that, for any set of variables $x_1, \ldots, x_n$ and function f,

$$\begin{aligned} \sum_{x_1,\ldots,x_n} \prod_{i=1,\ldots,n} f(x_i) &= \sum_{x_1} \cdots \sum_{x_n} f(x_1) \ldots f(x_n) \\ &= \Big[\sum_{x_1} f(x_1)\Big] \ldots \Big[\sum_{x_n} f(x_n)\Big] \\ &= \prod_{i=1,\ldots,n} \sum_{x_i} f(x_i). \end{aligned} \tag{4}$$

Based on this, we can write

$$P(sick(c,m), sick(c,j)) = \Big[\prod_{(D,X)\in Q} P(sick(D,X))\Big]\Big[\prod_{(D,X)\in NQ}\sum_{sick(D,X)} P(sick(D,X))\Big].$$

In this particular case, the summation equals 1 because $P(sick(D,X))$ is a probability distribution, and the whole second product disappears. This is only true in the cases in which the eliminated variables are children in the conditional probabilities involved. In the general case we may get a new potential on the remaining variables, which is a summation over the parameterized variable's domain, without the need to consider all groundings. We then have

$$\begin{aligned} P(sick(c,m), sick(c,j)) &= \prod_{(D,X)\in Q} P(sick(D,X)) \\ &= P(c,m)P(c,j) = 0.0000249001 \end{aligned}$$

which, as in our informal earlier observation, says that the query's probability is indeed the product of $P(sick(D,X))$ over the random variables in the query. While this numeric value per se is incorrect (according to (3)), it is so due to assertion (1) being incorrect, not because of the elimination just performed.

The most important part of the manipulations above is the application of 4. Notice however that it could only be applied because there was a one-to-one correspondence between the bindings $(D,X) \in NQ$ in the product and $sick_{NQ}(D,X)$ in the sum. Had we had, for example, $(D,X) \in NQ$ in the product and $epi(D)$ in the sum, there would be no such correspondence and we would not be able to apply 4. We can state this condition as follows:

Condition 2.1. *A parameterized random variable can only be eliminated if there is a one-to-one correspondence between the assignments to its logical variables and the assignments to the logical variables of probabilistic assertions in which it occurs (after appropriate splitting of assertions).*

This condition is properly formalized in Theorem 3.4.

2.4 Why Poole's method is incorrect

The need for Condition 2.1 explained above can be better seen by examining the incorrect elimination of $epi(D)$ presented before, and showing how they are not satisfied in it. Our simplified model defines a joint distribution on all its ground random variables, $\{sick(D,X)\} \cup \{epi(D)\}$:

$$P(\{sick(D,X)\} \cup \{epi(D)\}) = \Big[\prod_{D} P(epi(D))\Big]\Big[\prod_{(D,X)} P(sick(D,X)|epi(D))\Big].$$

If we choose to eliminate random variables of the form $epi(D)$ first, we sum over them and obtain the marginal on the remaining variables, $\{sick(D,X)\}$:

$$\begin{aligned} &P(\{sick(D,X)\}) \\ &\qquad = \sum_{epi(D)} P(\{sick(D,X)\} \cup \{epi(D)\}) \\ &\qquad = \sum_{epi(D)} \Big[\prod_{D} P(epi(D))\Big]\Big[\prod_{(D,X)} P(sick(D,X)|epi(D))\Big] \qquad (5) \\ &\qquad = \Big[\prod_{D} P(epi(D))\Big]\Big[\prod_{D} \sum_{epi(D)} \prod_{X} P(sick(D,X)|epi(D))\Big]. \end{aligned}$$

After correct inversions on products indexed by D, the summation cannot "pass through" the product indexed by X because there is no one-to-one correspondence between the groundings of D and X and we cannot apply something like (4). In comparison, as we have seen in (1), Poole's method determines that

$$P(sick(D,X)) = \sum_{epi(D)} P(sick(D,X)|epi(D))P(epi(D)).$$

Obtaining such an assertion is to say, according to the semantics we have defined, that

$$P(\{sick(D,X)\}) = \prod_{D}\prod_{X}\sum_{epi(D)} P(sick(D,X)|epi(D))P(epi(D)),$$

which does not follow from (5) (they provide different numerical answers).

In some cases Condition 2.1 greatly reduce the number of valid orderings. For example, in an assertion with parameterized random variables

$$u_1(X), \ldots, u_m(X), b(X, Y_1), \ldots, b(X, Y_n),$$

only an exponentially small number of orderings (those with all binary variables coming before any unary variables) will be valid. However, it is not difficult to find a valid ordering, as discussed next.

2.5 Keeping the Inversion Condition satisfied

We have calculated $P(sick(c, j), sick(c, m))$ analytically, but it is necessary to show that this can be done by lifted variable elimination. This depends on finding an elimination ordering such that every set of variables being eliminated satisfies Condition 2.1, which is the case for the ordering $sick_{NQ}(D, X) = \{sick(D, X) : (D, X) \in NQ\}$ and then $\{epi(D) : D \in disease\}$. By eliminating $sick_{NQ}(D, X)$ first, we eliminate ground variables with all values of X but j and m, so we can do away with products indexed by X, after which it is possible to soundly eliminate $epi(D)$. We do this by writing

$$\begin{aligned} &P(\{sick(D, X) : (D, X) \in Q\} \cup \{epi(D)\}) \\ &\qquad = \sum_{sick_{NQ}(D,X)} P(\{sick(D, X)\} \cup \{epi(D)\}) \\ &\qquad = \sum_{sick_{NQ}(D,X)} \Big[\prod_D P(epi(D))\Big] \Big[\prod_{(D,X)} P(sick(D, X)|epi(D))\Big] \end{aligned}$$

We again have made use of (4), obtaining a summation without a product inside. This summation is 1 and Q contains only two elements, so we expand it and write

$$\begin{aligned} &P(\{sick(D, X) : (D, X) \in Q\} \cup \{epi(D)\}) = \\ &\qquad \Big[\prod_D P(epi(D))\Big] P(sick(c, m)|epi(c)) P(sick(c, j)|epi(c)). \end{aligned}$$

We now proceed in an analogous fashion by eliminating variables of the form

$epi(D)$.

$$
\begin{aligned}
&P(sick_Q(D,X)) = P(sick(c,m), sick(c,j)) \\
&= \sum_{epi(D)} P(\{sick(D,X) : (D,X) \in Q\} \cup \{epi(D)\}) \\
&= \Big[\sum_{epi(D)} \Big[\prod_{D} P(epi(D)) \Big] P(sick(c,m)|epi(c)) P(sick(c,j)|epi(c)) \Big] \\
&= \sum_{epi(c)} \Big[\sum_{\substack{epi(D) \\ : D \neq c}} \Big[P(epi(c)) \prod_{D \neq c} P(epi(D)) \Big] P(sick(c,m)|epi(c)) P(sick(c,j)|epi(c)) \Big] \\
&= \Big[\sum_{epi(c)} P(sick(c,m)|epi(c)) P(sick(c,j)|epi(c)) P(epi(c)) \Big] \Big[\sum_{\substack{epi(D) \\ : D \neq c}} \prod_{D \neq c} P(epi(D)) \Big] \\
&= \Big[\sum_{epi(c)} P(sick(c,m)|epi(c)) P(sick(c,j)|epi(c)) P(epi(c)) \Big] \Big[\prod_{D \neq c} \sum_{epi(D)} P(epi(D)) \Big].
\end{aligned}
$$

Again we have a case of a summation being evaluated to 1, so we obtain

$$
\begin{aligned}
P(sick_Q(D,X)) &= P(sick(c,m), sick(c,j)) \\
&= \sum_{epi(c)} P(sick(c,m)|epi(c)) P(sick(c,j)|epi(c)) P(epi(c)),
\end{aligned}
$$

which amounts to the calculation (3) resulting in the correct answer 0.00160099.

We have shown in our example that not every ordering is valid but have found a valid one. It is important to know whether it is always possible to do so for any query. The answer is positive, as we see in the next section.

3 Formalization

In this section we formalize the problem and the notions necessary to solve it, presenting the algorithm in technical detail and the proof of its correctness.

3.1 Basic Definitions

3.1.1 Syntax

Definition 3.1. *Let $\mathcal{F}$ be an infinite set of constant symbols, $\mathcal{X}$ an infinite set of variable symbols, $\mathcal{P}$ be an infinite set of predicate symbols, $\mathcal{T}$ the set of terms equal to $\mathcal{F} \cup \mathcal{X}$, and $\mathcal{A}$ the set of logical atoms (or simply atoms) formed with a predicate symbol in $\mathcal{P}$ and terms in $\mathcal{T}$. Each variable X has a type, a set of possible values for X, which is provided by a function $T(X)$.*

Let $\mathcal{T}^*$ be the set of possible sequences of symbols in $\mathcal{T}$. Then the function $pred : \mathcal{A} \to \mathcal{P}$ maps atoms to their predicate symbols and the function $args : \mathcal{A} \to \mathcal{T}^*$ maps atoms to the sequence formed by their arguments.

In our introductory example, *mary* and *measles* are (constant) terms, X and D are variable terms, *hadContact* and *sick* are predicate symbols and $sick(D, X)$ and $epidemic(D)$ are atoms.

Definition 3.2. *An* equality constraint *is an equation between terms, and a* disequality constraint *is a disequation (that is, a negated equation) between terms. A* constraint *is either an equality constraint or a disequality constraint. A boolean formula on constraints ($\top$, $\bot$ or a combination of constraints using $\wedge$,$\vee$ and $\neg$ connectives) is called a* constraint formula.

A constraint system *is a pair $(F, \mathbf{V})$ where F is a constraint formula and $\mathbf{V}$ is a set of logical variables, called* the base *of $(F, \mathbf{V})$. We often use "," to denote $\wedge$ in constraint systems.*

In our example, $X \neq mary$ is a disequation and also a constraint system, when coupled with the set $\{X_1, X_2\}$.

Definition 3.3. *An* assignment *is a function $\sigma : \mathcal{X} \to (\mathcal{F} \cup \mathcal{X})$. The* domain *of σ, $Dom(\sigma)$, is the set of variables being mapped to something else, $\{X : \sigma(X) \neq X\}$.*

Let $\mathbf{V}$ be a set of logical variables. A grounding assignment wrt $\mathbf{V}$ *is an assignment σ mapping all variables in $\mathbf{V}$ to ground terms, that is, $\{X \in \mathbf{V} : \sigma(X) \in \mathcal{F}\}$.*

Definition 3.4. *A* random variable *is denoted by a ground atom, while a* parameterized random variable *is denoted by an atom in general.*

A sequence of parameterized random variables*, or simply* parseq*, is a pair* $(\mathbf{A}, C)$ *where* $\mathbf{A}$ *is a sequence of parameterized random variables and* C *is a constraint system on the logical variables occurring in* $\mathbf{A}$*. The constraint system is required to contain all logical variables occurring in* $\mathbf{A}$ *(and possibly more). For simplicity of notation, for any parseq* $(\mathbf{A}, C)$ *we define* $ParVar((\mathbf{A}, C)) = \mathbf{A}$ *and* $C((\mathbf{A}, C)) = C$*.*

A parfactor *is a pair* (ϕ, Arg) *of a parseq* $Arg = (\mathbf{A}, C)$ *and a non-negative real function* ϕ *on* $\mathbf{A}$ *called the* potential function *of the parfactor. The constraint system is required to contain all logical variables occurring in* Arg *(and possibly more). For simplicity's sake we define, for any parfactor* $g = (\phi', Arg')$*, that* $\phi_g = \phi'$*,* $Arg(g) = Arg'$*,* $ParVar(g) = ParVar(Arg(g))$*, and* $C(g) = C(Arg(g))$*.*

Probabilistic assertions are represented as parfactors. For example,

$$\forall X_1 : person, X_2 : person, D : disease;$$
$$X_1 \neq X_2 \rightarrow P(sick(D, X_1) | hadContact(X_1, X_2), sick(D, X_2)) = 0.2$$

can be represented as the parfactor

$$\big(\phi, (\{sick(D, X_1), hadContact(X_1, X_2), sick(D, X_2)\}, (X_1 \neq X_2, \{X_1, X_2, D\}))\big)$$

where ϕ is the function

$$\phi\big(sick(D, X_1), hadContact(X_1, X_2), sick(D, X_2)\big) = \begin{cases} 0.2, & \text{if } hadContact(X_1, X_2) \wedge sick(D, X_2), \\ 0.5, & \text{otherwise.} \end{cases}$$

The ϕ is a potential function as used in undirected graphical models. The seemingly arbitrary value of 0.5 for when the condition is false causes, when this is the case, the total joint distribution to be computed as if the assertion were not present, which is the expected effect. This is for convenience only, and the specification of conditional probabilities for each assignment to parents is also possible.

We will often talk about multiple parfactors or parseqs. In these cases they will always be standardized away, that is, logical variables present in separate constraint systems are assumed to be distinct even if they have the same name. This can always be made more clear by renaming them if necessary.

Definition 3.5. *A* syntactic construct *is any of the constructs defined above (a term, an atom, a parseq, a parfactor, a constraint, a constraint system, an assignment), a set of those, or a sequence of those.*

Let S be a syntactic construct. Then $LVar(S)$ is the set of logical variables occurring in S and $ParVar(S)$ is the set of parameterizes random variables occurring in S (as we had already defined for parseqs and parfactors).

Let S be a syntactic construct and σ an assignment wrt $LVar(S)$. Then $S\sigma$ is the syntactic construct resulting from the replacement of every occurrence of a logical variable in S by its corresponding value in σ. For any set of assignments Θ, $S\Theta$ is the sequence $\{A\sigma\}_{\sigma\in\Theta}$.

3.1.2 Semantics – Extensions

Definition 3.6. *An assignment σ is* consistent *with a constraint formula F if $F\sigma$ is true.*

For every constraint system $C = (F, \mathbf{V})$ the extension of C, *$[C]$, denotes the set of all grounding assignments wrt $\mathbf{V}$ which are consistent with F. The* extension of a constraint formula *F is defined as $[(F, LVar(F)]$.*

Two constraint systems C and D are equivalent *if $[C] = [D]$.*

For any formula F and syntactic construct S, $S[F]$ is an abbreviation for $S[(F, Var(S))]$.

An analogous operation is defined for parseqs. If E is a parseq, then $[E]$ is $ParVar(E)[C(E)]$ (all possible substitutions according to grounding assignments in $[C(E)]$. Two parseqs D and E are called equivalent *if $[D] = [E]$.*

A notation remark is in order. It is usual in probability texts to denote the mathematical variables representing the assignments to random variables (not be be confused with the assignments to logical variables about which we have been talking about) by their small letter counterparts. For example, $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ represents an assignment to $X_1, \ldots, X_{i-1}$, $X_{i+1}, \ldots, X_n$. In our case, this is made a bit more complex since we deal with sets of random variables represented by potentially complex expressions, involving constraint systems, set operations and indexing. We follow the small letter convention by establishing that expressions involving small letter versions of known variables and functions represent assignments to the set of

random variables determined by the same expression having the small letter variable names replaced by their capital letter counterparts. So, for example, $a[\top]$, $a\Theta_1 - a\Theta_2$ and $parVar(g)$ represent assignments to $A[\top]$, $A\Theta_1 - A\Theta_2$ and $ParVar(g)$ (the arguments of ϕ_g), respectively. This is an abuse of notation in the sense that the operations applied to small letter variables are really meant to be applied to the objects denoted by their capital letter versions, but the meaning will always be clear from the context.

Definition 3.7. *The* grounding *of parfactor $g = (\phi, Arg)$, for $Arg = (\mathbf{A}, C)$, is the function $[g]([arg]) = \prod_{\sigma \in [C]} \phi(\mathbf{a}\sigma)$, defined on assignments to random variables $[Arg]$.*

Because $[g]$ is always applied to $[arg]$ which is defined by g itself, we reduce redundancy by abusing the notation and writing simply $[g]$ instead of $[g]([arg])$ when it is clear that we mean the value on a given assignment to $[Arg]$ rather than the function itself.

Note that the operations $[\cdot]$ for constraint systems and $[\cdot]$ for parfactors, although not technically the same, represent the same idea of considering all of their possible groundings. The grounding of a parfactor, however, is a real function, while the grounding of constraint systems are sets of grounding assignments and the grounding of parseqs are sets of ground random variables.

Definition 3.8. *We define the function $GVar(g)$ on a parfactor g to be $[Arg(g)]$ and $GVar(G) = \bigcup_{g \in G} GVar(g)$ for any set of parfactors G.*

3.1.3 Semantics – Joint distribution

We are now ready to define the joint probability described by a set of parfactors.

Definition 3.9. *Given a set of parfactors G defined on a set of parameterized random variables (atoms) A, the joint distribution on all the random variables defined by it is given by*

$$P(gVar(G)) \propto \prod_{g \in G} [g]$$

where, according to the definitions above, $gVar(G)$ is an assignment to $GVar(G)$, the set of all random variables obtained by grounding all atoms in G.

It should be clear that the definition above is equivalent to

$$P(gVar(G)) \propto \prod_{g \in G} \prod_{\sigma \in [C(g)]} \phi_g(parVar(Arg(g))\sigma)$$

where $parVar(Arg(g))\sigma$ is, according to our notation convention, an assignment to $ParVar(Arg(g))\sigma$ and is simply expressing that the joint distribution on random variables (the ground atoms) is determined by the product of all groundings of all parfactors.

In order to further simplify notation, we make the taking of paramaterized random variables from a parseq unnecessary inside arguments to potential functions. This is unambiguous since parsets themselves cannot be such arguments. This allows us to write

$$\phi_g(arg(g)\sigma) = \phi_g(parVar(Arg(g))\sigma) \tag{6}$$

3.2 First-Order Variable Elimination

We now present the method of First-order variable elimination. The problem is to find the marginal distribution on a set of random variables given a model and evidence (an assignment to another set of random variables). Just as in the propositional case, we can incorporate evidence by integrating it into the model, so we restrict ourselves to answering queries given a model only. Also, for didactic purposes, we restrict ourselves for now to answering ground queries only.

Given a set of parfactors G and a query Q, Theorem 3.3 guarantees that either G is defined on Q only or there are random variables $[E]$, such that $[E] \cap Q = \emptyset$, which can be eliminated. It also provides the sets of parfactors H' and H'' such that $H' \cup H''$ is equivalent to G, parfactors in H'' do not involve any ground random variables in $[E]$ while all parfactors in H' do. These properties can be used to simplify the marginalization with a sum

based on the assignments to $[E]$ to a marginalization without a summation:

$$\begin{aligned} P(gVar(G) \setminus [e]) &= \sum_{[e]} P(gVar(G)) \\ &= \sum_{[e]} P(gVar(H' \cup H'')) \\ &= \sum_{[e]} \Big[\prod_{h'' \in H''} [h''] \Big] \Big[\prod_{h' \in H'} [h'] \Big] \\ &= \Big[\prod_{h'' \in H''} [h''] \Big] \sum_{[e]} \Big[\prod_{h' \in H'} [h'] \Big] \\ &= \Big[\prod_{h'' \in H''} [h''] \Big] [g'], \end{aligned} \tag{7}$$

where g' can be calculated according to Theorem 3.4 and whose grounding does not involve the random variables in $[E]$. By repeating this procedure, we eventually obtain a marginal distribution on Q only.

We now proceed to show the lower-level results justifying this method.

3.3 Splitting a model

We define the notion of a *splitting unifier*, which describes a set of random variables to be eliminated from a model. As we have discussed before, not every set of variables can be eliminated and, accordingly, there is also a notion of a *valid* splitting unifier. By finding a valid splitting unifier, we determine the setting for a step of first-order variable elimination.

We present the definitions and theorems here, leaving the proofs for appendix D.

3.3.1 Splitting unifiers

Splitting unification is an operation where, given a set of parseqs $\mathbf{R}$ and a "seed" parseq E (meant to be everything *not* in the query), one tries to find a parseq E' (meant to be the variables eliminated in the next step) such that $[E'] \subseteq [E]$ and, for each original parseq $R \in \mathbf{R}$, $[E']$ is either completely included in it ($[E'] \subseteq [R]$) or completely disjoint from it ($[E'] \cap [R] = \emptyset$).

Definition 3.10. *Let* $\mathbf{R}$ *be a set of parseqs and* E *a parseq. A* splitting unifier of $\mathbf{R}$ *given a* seed E *is a parseq* E' *such that* $[E'] \subseteq [E]$, $[E'] \neq \emptyset$, *for any* $S \in \mathbf{R}$, *either* $[E'] \subseteq [S]$ *or* $[E'] \cap [S] = \emptyset$ *and there is at least one* $R \in \mathbf{R}$ *for which* $[E'] \subseteq [R]$. *A splitting unification of* $\mathbf{R}$ *given* E *is said to* fail *if there is no splitting unifier of* $\mathbf{R}$ *given* E, *and to* succeed *otherwise.*

If there is a non-empty intersection between $[E]$ and $[R]$ for some $R \in \mathbf{R}$, then we can find a splitting unifier for $\mathbf{R}$ given E. Intuitively, this is done by first unifying some part of E to some part of R, and then using this resulting intersection as a seed for the remaining elements in $\mathbf{R}$.

Theorem 3.1. *Let* $\mathbf{R}$ *be a set of parseqs and* E *a parseq. There is at least one element* $R \in \mathbf{R}$ *such that* $[E] \cap [R] \neq \emptyset$ *if and only if there is a splitting unifier* E' *of* $\mathbf{R}$ *given* E.

3.3.2 Valid splitting unifiers

We define a *valid* splitting unifier of a set of parseqs wrt a ground query. To do that, we first need the notion *uncovered* logical variables.

The next definition uses the notion of *partial unifier*, treated in appendix D. Intuitively, a partial unifier describes the unification between subsets of parseqs. For example, the partial unifier of parseqs

$$R_1 = (\{p(X, Y), q(Y, Z)\}, (Y \neq Z, \{X, Y, Z\}))$$

and

$$R_2 = (\{q(V, a), r(W, W)\}, (\top, \{V, W\}))$$

is

$$(\{q(Y', Z')\}, (Z' = a \wedge Y' \neq a, \{Y', Z'\})).$$

Furthermore, this partial unifier defines assignments $\sigma_{R_1} = \{Y \rightarrow Y', Z \rightarrow Z'\}$ and $\sigma_{R_2} = \{V \rightarrow Y'\}$.

Definition 3.11. *Let* E *be a splitting unifier of a set of parseqs* $\mathbf{R}$ *with some seed. For any* $R \in \mathbf{R}$ *such that* $[R] \cap [E] \neq \emptyset$, *let* σ_R *be the assignment defined by the partial unifier of* E *and* R *(as determined by Theorem D.2). A logical variable* Y *in* $LVar(R)$ *is* covered by E wrt $\mathbf{R}$ *if* Y *is in the domain of* σ_R, *and* uncovered *otherwise.*

Definition 3.12. *A splitting unifier E of a set of parseqs $\mathbf{R}$ (with some seed) is* valid *if there are no uncovered logical variables by E wrt $\mathbf{R}$.*

Theorem 3.2. *For any set $\mathbf{R}$ of parseqs and Q a set of ground atoms such that $GVar(\mathbf{R}) \setminus Q \neq \emptyset$ there is a valid splitting unifier E of $\mathbf{R}$ with seed $GVar(R) \setminus Q$.*

Finally, we present the main result which is the basis for first-order variable elimination.

Theorem 3.3. *Let G be a set of parfactors and Q a ground query. If $GVar(G) \neq Q$ then there exists*

- *a parseq E such that $[E] \cap Q = \emptyset$,*
- *a set of parfactors H' such that $C(h') = C(E)$ and $E \subset Arg(h')$ for each $h' \in H'$, and*
- *a set of parfactors H'' (called* residuals*) such that $[Arg(h'')] \cap [E] = \emptyset$ for each $h'' \in H''$.*

Moreover, $H = H' \cup H''$ defines the same distribution as G, i.e., $\prod_{h \in H}[h] = \prod_{g \in G}[g]$.

Proof. According to Theorem 3.2 applied to the parseqs $P = \{Arg(g) : g \in G\}$ and Q, there is a valid splitting unifier E of $\mathbf{R}$ wrt Q. Let $G' \subseteq G$ be the set of parfactors such that $[E] \subseteq [Arg(g')]$ for every $g' \in G'$, and $G'' \subseteq G$ be the set of parfactors such that $[E] \not\subseteq [Arg(g'')]$ for every $g'' \in G''$ defined by the splitting.

Because this splitting is valid, for each $g' \in G'$ there is an assignment $\sigma_{g'}$ involving all logical variables in g' such that $C(g')\sigma = C(E)$; let $Arg^+_{g'} = (ParVar(g')\sigma, C(E))$ and $Arg^-_{g'} = (ParVar(g')\sigma, C(g')\sigma \wedge \neg C(E))$. According to this constructions, $[Arg^+_{g'}]$ and $[Arg^-_{g'}]$ form a partition of $[Arg(g')]$, so the two parfactors $h^+_{g'} = (\phi_{g'}, Arg^+)$ and $h^-_{g'}(\phi_{g'}, Arg^-)$ are equivalent to g'. Also, because of this construction, $[E] \subseteq GVar(h^+_{g'})$ and since $C(h^+_{g'}) = C(E)$ it is also true that $E \subseteq Arg(h^+_{g'})$. Furthermore, $[E] \cap GVar(h^-_{g'})$ because of the construction of $C(Arg^-_{g'})$ which excludes $C(E)$.

We define H' to be $\{h^+_{g'} : g' \in G'\}$ and H'' to be $\{h^-_{g'} : g' \in G'\} \cup G''$ which, along with E, are the objects guaranteed by the theorem to exist. □

3.4 Doing away with summation

Another important step, used to calculate g' in (7), is the operation which, given a set of parfactors sharing the variables to be eliminated and a constraint system, determines a new parfactor without the eliminated variables.

Note that in the theorem below we use the operation of subtracting a set of logical variables from a constraint system and the function Dim. These concepts are defined in appendix A on constraint systems, but intuitively the former is meant to be the removal of unnecessary variables from a constraint system and the latter how many times the number of assignments is reduced by this elimination.

Theorem 3.4. *Let $C = (F, \mathbf{V})$ be a constraint system and G be a set of parfactors such that, for every $g \in G$, $C(g) = C$. Also, let E be a parseq such that $LVar(E) = \mathbf{V}$ and $ParVar(E) \in ParVar(Arg(g))$ for every $g \in G$. Let $\mathbf{R}$ (for* remaining*) be $LVar(Arg')$, $\mathbf{D}$ (for* disappearing*) be $LVar(G) \setminus \mathbf{R}$ and Arg' be the parseq $(\cup_{g \in G} ParVar(Arg(g))) \setminus ParVar(E), C - \mathbf{D})$. Then*

$$\sum_{[e]} \prod_{g \in G} [g] = [g']$$

where $g' = (\phi_{g'}, Arg')$, with

$$\phi_{g'}(arg') = \Big(\sum_{d \in \{0,1\}} \prod_{g \in G} \phi(d, arg(g) \setminus e)) \Big)^{Dim(C, \mathbf{D})}.$$

Note that we are using our notation convention as in (6). Strictly speaking, the last equation should have been written as

$$\phi_{g'}(parVar(Arg')) = \Big(\sum_{d \in \{0,1\}} \prod_{g \in G} \phi(d, parVar(Arg(g)) \setminus parVar(E))) \Big)^{Dim(C, \mathbf{D})}.$$

Proof.

$$\sum_{[e]} \prod_{g \in G} [g] = \sum_{[e]} \prod_{g \in G} \prod_{\sigma \in [C(g)]} \phi_g(e\sigma, (arg(g) \setminus e)\sigma)$$

but $C(g) = C$, so

$$= \sum_{[e]} \prod_{g \in G} \prod_{\sigma \in [C]} \phi_g(e\sigma, (arg(g) \setminus e)\sigma)$$
$$= \sum_{e\sigma_1} \cdots \sum_{e\sigma_n} \prod_{g \in G} \phi_g(e\sigma_1, (arg(g) \setminus e)\sigma_1) \cdots \prod_{g \in G} \phi_g(e\sigma_n, (arg(g) \setminus e)\sigma_n)$$

for $\sigma_1, \ldots, \sigma_n$ all the grounding assignments in $[C]$. Note that the ability to expand both $[e]$ and $\prod_{\sigma \in [C]}$ into corresponding elements depends on the hypothesis that $LVar(E) = \mathbf{V}$; if $\mathbf{V}$ had some variable not in $LVar(E)$, then we would have $e\sigma_i = e\sigma_j$ for some $i \neq j$. Now, because the parfactors are shattered, no $e\sigma_i$ occurs in any
$\prod_{g \in G} \phi_g(e\sigma_j, (arg(g) \setminus e)\sigma_j)$ for $j \neq i$, so we can factor things out and obtain

$$= \Big[\sum_{e\sigma_1} \prod_{g \in G} \phi_g(e\sigma_1, (arg(g) \setminus e)\sigma_1)\Big] \ldots \Big[\sum_{e\sigma_n} \prod_{g \in G} \phi_g(e\sigma_n, (arg(g) \setminus e)\sigma_n)\Big]$$

By renaming $e\sigma_i$ in each summation, we obtain

$$= \Big[\sum_{d \in \{0,1\}} \prod_{g \in G} \phi_g(d, (arg(g) \setminus e)\sigma_1)\Big] \ldots \Big[\sum_{d \in \{0,1\}} \prod_{g \in G} \phi_g(d, (arg(g) \setminus e)\sigma_n)\Big]$$
$$= \prod_{\sigma \in [C]} \sum_{d \in \{0,1\}} \prod_{g \in G} \phi_g(v, (arg(g) \setminus e)\sigma)$$

We now do not have the variables in $\mathbf{D}$ in the arguments to the ϕ_g functions. Therefore, the application of each assignment $\sigma \in [C]$ agreeing on $\mathbf{V} \setminus \mathbf{D} = \mathbf{R}$ will result in the same expression. We can therefore replace all these identical assignment applications by a single one application by the assignment in $[C - \mathbf{D}]$ that agrees with them on $\mathbf{R}$, powering it to the number of original assignments, which Theorem A.6 says to be $Dim(C, \mathbf{D})$, writting

$$= \prod_{\sigma \in [C - \mathbf{D}]} \Big(\sum_{d \in \{0,1\}} \prod_{g \in G} \phi_g(d, (arg(g) \setminus e)\sigma)\Big)^{Dim(C, \mathbf{D})}$$
$$= [\phi_{g'}]$$

□

4 Conclusion

First-order probabilistic inference is important as a knowledge representation. So far is has not lived up to expectations, however, due to the lack of inference algorithms taking concrete advantage from it. Following and correcting the ideas in [Poo03], we have formalized an algorithm that benefits from the representation by avoiding full grounding of the model.

Future work includes determining what model measures are relevant in complexity analysis (for example determining how a tree-width cost analysis applies to the first-order case). Other directions are to find orderings that may lead to a small cost, to perform bounded approximation, and applications (we are currently applying it to a human-computer interface application).

References

[Bar98] Roman Barták. Constructive negation in CLP(H). Technical Report No. 98/6, Charles University, July 1998.

[Hal90] Joseph Y. Halpern. An analysis of first-order logics of probability. In *Proceedings of IJCAI-89, 11th International Joint Conference on Artificial Intelligence*, pages 1375–1381, Detroit, US, 1990.

[Jae97] Manfred Jaeger. Relational Bayesian networks. In Morgan Kaufmann, editor, *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 266–273, 1997.

[KP98] Daphne Koller and Avi Pfeffer. Probabilistic frame-based systems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 580–587, 1998.

[KR00] Kristian Kersting and Luc De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.

[NH95] Liem Ngo and Peter Haddawy. Probabilistic logic programming and bayesian networks. In *Asian Computing Science Conference*, pages 286–300, 1995.

[NS92] Raymond T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[Pea91] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo (Calif.), 1991. Harold Cohen Library Liverpool.

[PKMT99] Avi Pfeffer, Daphne Koller, Brian Milch, and Ken T. Takusagawa. Spook: A system for probabilistic object oriented knowledge representation. In *Proceedings of the 14th Annual Conference on Uncertainty in AI (UAI-99)*, pages 541–550, 1999.

[Poo93] David Poole. Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[Poo03] D. Poole. First-order probabilistic inference. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 985–991, 2003.

[ZP94] N.L. Zhang and D. Poole. A simple approach to bayesian network computations. In *Proceedings of the Tenth Biennial Canadian Artificial Intelligence Conference*, 1994.

A Constraint Systems

The algorithm and proof presented in this paper require intensive manipulation of constraint systems. This subsection defines several properties and operations used on them.

It is convenient to keep constraint systems of parfactors in a normalized form. A constraint formula F is *normalized* if it is one of $\top$, $\bot$ and a conjunction of constraints in which each logical variable appears in at most one left-hand side of an equality constraint, no constraint appears twice, there are no equality constraints of the form $t = t$ for any term t and no disequality constraints of the form $c_1 \neq c_2$ for any two distinct constants c_1 and c_2.

This is a normal form as described in [Bar98]. A *normalized constraint system* is a constraint system whose constraint formula component is normalized. We define $normal(F)$ to be the normal form of a constraint conjunction F.

Theorem A.1 ([Bar98]). *Every constraint conjunction is equivalent to a normalized constraint conjunction.*

The following theorem shows that a constraint system is always equivalent to a set of normalized constraint systems.

Theorem A.2. *For every constraint formula F, let D be the set of disjuncts in the DNF of F and F_G, for each $G \in (2^D - \emptyset)$, be the normalized constraint conjunction normal$\big(\bigwedge_{d\in D, d\in G} d \bigwedge_{d\in D, d\notin G} \neg d\big)$. Then $\{[F_G] : G \in (2^D - \emptyset)\}$ is a partition of $[F]$.*

Proof. Let H be $\{F_G : G \in (2^D - \emptyset)\}$. We show that the extensions of elements in H form a partition of $[F]$ by showing that an assignment satisfies F if and only if it is in the extension of one and only one element of H. For each assignment σ satisfying F take $G_\sigma = \{d \in D : d\sigma \text{ is true}\}$, which is in $2^D - \emptyset$. By the definition of F_G, σ must be in $[F_{G_\sigma}]$. F_{G_σ} is the only element in H to whose extension σ belongs to. This is so because for any $G' \in H, G' \neq G_\sigma$ since $G' \neq G$ there must exist $d \in D$ that is not in G_σ and that therefore it not satisfied by σ, so $\sigma \notin [F_{G'}]$. Conversely, if σ does not satisfy F, it does not satisfy F' or any $d \in D$. Since all assignments in the extension of every $F_G \in H$ must satisfy some $d \in D$, σ is not in any of them. □

Let $C = (F, \mathbf{V})$ be a normalized constraint system and $X \in \mathbf{V}$ be a variable. C is *shattered wrt* X if, for every pair of distinct constraints $X\ rel_1\ \alpha$ and $X\ rel_2\ \beta$ in F, where $rel_1, rel_2 \in \{=, \neq\}$, either $\forall\sigma \in [C] : \alpha\sigma = \beta\sigma$ or $\forall\sigma \in [C] : \alpha\sigma \neq \beta\sigma$.

The notion of shattering is useful when calculating the number of possible values for X in $[C]$, for if we know that different constraints on X will always be either equivalent or contradictory to each other, then the number of possible values for X will depend solely on the number and type of constraints on it. This will be shown in Theorem A.4.

The next theorem shows that we can always replace a parfactor by a set of parfactors, each with a constraint system based on a normalized constraint system shattered wrt a certain variable.

Theorem A.3. *Let $C = (F, \mathbf{V})$ be a normalized constraint system, X a variable and J the set of conjuncts in F. Then $S(C)$, defined as*

$$\{(F', \mathbf{V}) : F' = normal(F \wedge \bigwedge_{\substack{\alpha,\ rel,\ \beta: \\ (X\ rel_1\ \alpha)\in J,(X\ rel_2\ \beta)\in J, \\ \alpha\neq\beta,rel_1,rel_2,rel\in\{=,\neq\}}} \alpha\ rel\ \beta), F' \neq \bot\}$$

is a set of normalized constraint systems shattered wrt X whose extensions form a partition of $[C]$.

Moreover, each of the resulting constraint systems preserves previous shattering, i.e., if C is shattered wrt a variable Y, then so are the elements of $S(C)$.

Proof. Every constraint system $(F', \mathbf{V}) \in S(C)$ is shattered wrt X because, for every pair of distinct constraints $X\ rel_1\ \alpha$ and $X\ rel_2\ \beta$ in F, α and β are restricted to be either equal or different according to every assignment in $[F']$.

Every assignment σ in the extension of a constraint system $(F', \mathbf{V}) \in S$ is also in $[(F, \mathbf{V})]$, because if $F'\sigma$ is true, then it satisfies all conjuncts used in the construction of F', one of which is F itself.

Given distinct constraint systems $(F', \mathbf{V}), (F'', \mathbf{V}) \in S$, their extensions are disjoint because their construction involve opposite constraints for at least one pair of terms α and β, so the assignments in one's extensions will never satisfy the other's.

Every assignment $\sigma \in [(F, \mathbf{V})]$ is in the extension of at least one element of $S(C)$. To see this, we consider that for each pair of disequalities $X = \alpha$ and $X = \beta$, either $\alpha\sigma = \beta\sigma$ or $\alpha\sigma \neq \beta\sigma$, and by the construction of $S(C)$ there will always be an element $(F', \mathbf{V})$ in it such that $F'\sigma$ is true. Note that F' is necessarily different from $\bot$ because it is equivalent to $normal(F \wedge \bigwedge_{\alpha,\beta:X=\alpha\in J,X=\beta\in J,\alpha\neq\beta,\sigma(\alpha)=\sigma(\beta)} \alpha = \beta \wedge \bigwedge_{\alpha,\beta:X=\alpha\in J,X=\beta\in J,\alpha\neq\beta,\sigma(\alpha)\neq\sigma(\beta)} \alpha \neq \beta)$ so at least σ is in $[F']$, by this definition and from the fact that $\sigma \in [F]$.

The preservation of shattering can be observed from the fact that the extensions of elements of $S(C)$ are subsets of $[C]$. If it is true that C is shattered wrt Y, then for every pair of distinct constraints $Y\ rel_1\ \alpha$ and $Y\ rel_2\ \beta$ in F, where $rel_1, rel_2 \in \{=, \neq\}$, either $\forall\sigma \in [C] : \alpha\sigma = \beta\sigma$ or $\forall\sigma \in [C] : \alpha\sigma \neq \beta\sigma$, and consequently, for every $C' \in S(C)$, either $\forall\sigma \in [C'] : \alpha\sigma = \beta\sigma$ or

$\forall \sigma \in [C'] : \alpha\sigma \neq \beta\sigma$. This shows that every $C' \in S(C)$ remains shattered wrt Y. □

The notion of shattered constraint systems is useful for calculating the number of assignments eliminated when a logical variable is eliminated from the system. For this we need to define the notions of eliminations of logical variables from constraint systems, and well as the dimensionality of a constraint system wrt a logical variable.

We define the elimination of logical variables from a normalized constraint system through a $-$ operation between a constraint system and a set of variables. For every $C = (F, \mathbf{V})$ a normalized constraint system and $W \subseteq \mathbf{V}$ a set of variables, $C - W$ is the constraint system $(F', \mathbf{V} - W)$, where F' is the conjunction of all conjuncts in F that do not involve any variables in W.

Let $C = (F, \mathbf{V})$ be a normalized constraint system and X a logical variable. Then the *dimensionality of C with respect to X*, denoted as $Dim(C, X)$, is a natural number such that for every assignment $\sigma \in [C - \{X\}]$ there are exactly $Dim(C, X)$ distinct assignments in $[C]$ agreeing with σ on $\mathbf{V} - \{X\}$.

We now show how to calculate the dimensionality of constraint system wrt a variable.

Theorem A.4. *Let $C = (F, \mathbf{V})$ be a normalized constraint system shattered wrt a variable X. Then $Dim(C, X)$ is 1 if F contains a conjunct $X = t$ for some grounded term t and $|T(X)| - n$ if there is no such equality conjunct and there are $n \in \{0, \ldots, |T(X)|\}$ disequality conjuncts $X = t_i$ for grounded terms $t_i, i = 1, \ldots, n$.*

Proof. Let C' be $C - \{X\}$. For every $\sigma' \in [C']$, make σ identical to σ' but for mapping X to itself, and take $C'' = (F\sigma, \{X\})$. Because C is shattered wrt X, $F\sigma'$ contains at most one equality $X = t$ for a grounded term t and zero or more disequalities $X \neq t_i$ for $t_i, i = 1, \ldots, n$ grounded terms (distinct from t if there is an equality). These terms must all be grounded because X is the only variable left in $F\sigma$. If there is one equality $X = t$, then there is only one assignment $\sigma'' \in [C'']$ and agreeing with σ' on $\mathbf{V} - \{X\}$, which is the one mapping X to t and Y to $\sigma'(Y)$ for every $Y \neq X$. If there is no equality $X = t$, then for every value x in X's type $T(X)$ that is different from all t_i's (there will be $|T(X)| - n$ such values) there is an assignment $\sigma'' \in [C'']$ and agreeing with σ' on $\mathbf{V} - \{X\}$, which is the one mapping X to

x and Y to $\sigma'(Y)$ for every $Y \neq X$. In either case, there is a natural number r such that there are r assignments satisfying C''.

Now, for each $\sigma'' \in [C'']$, we build the assignment σ''' that agrees with σ' on $\mathbf{V} - \{X\}$ and with σ'' on X. $\sigma''' \in [C]$ because $F\sigma''' \equiv F\sigma'\sigma''$, which, since $\sigma'' \in [C'']$, must be true, and σ''', by its construction, agrees with σ'. Because we have r such assignments for each $\sigma'' \in [C''] = [C - \{X\}]$, the definition of $Dim(C, X)$ tells us that $Dim(C, X) = r$.

Now notice that the particular values to which σ' maps variables are unimportant in the above argument. Therefore the number $Dim(C, X)$ of assignments satisfying $C - \{X\}$ and agreeing with σ' on $\mathbf{V} - \{X\}$ will be the same for any $\sigma' \in [C']$. □

We now generalize these results from a single variable to a *set* of variables.

Given a constraint system $C = (F, \mathbf{V})$ shattered wrt all logical variables in a set $\mathbf{X} \subseteq \mathbf{V}$, the *dimensionality of C wrt $\mathbf{X}$* , denoted as $Dim(C, \mathbf{X})$, is a natural number such that for every assignment $\sigma \in [C - \mathbf{X}]$ there are exactly $Dim(C, \mathbf{X})$ distinct assignments in $[C]$ agreeing with σ on $\mathbf{V} - \mathbf{X}$.

We can calculate $Dim(C, \mathbf{X})$ for a normalized constraint system shattered wrt a set of logical variables $\mathbf{X}$ from its dimensionality wrt each variable in $\mathbf{X}$.

Theorem A.5. *Let C be a constraint system $(F, \mathbf{V})$ shattered wrt all logical variables in a set $\mathbf{X} \subseteq \mathbf{V}$. If $|\mathbf{X}| = 1$, then $Dim(C, \mathbf{X}) = Dim(C, X)$. If $|\mathbf{X}| > 1$, then for any variable $X \in \mathbf{X}$, with $\mathbf{Y} = \mathbf{X} - X$, $Dim(C, \mathbf{X}) = Dim(C - \mathbf{Y}, X) \times Dim(C, \mathbf{Y})$.*

Proof. We prove this by induction on $|\mathbf{X}|$. For the case base where $|\mathbf{X}| = 1$, we observe that the definitions for dimensionality wrt a single variable and wrt a set of variables with a single variable turn out to be the same, so $Dim(C, \mathbf{X}) = Dim(C, X)$. For $|\mathbf{X}| > 1$, by induction there are $Dim(C, \mathbf{Y})$ assignments in $[C]$ agreeing with each assignment in $[C - \mathbf{Y}]$ on $\mathbf{V} - \mathbf{Y}$, and, from the definition of dimensionality wrt a variable, there are $Dim(C - \mathbf{Y}, X)$ assignments in $[C - \mathbf{Y}]$ agreeing with each assignment in $[(C - \mathbf{Y}) - X] = [C - \mathbf{X}]$ on $\mathbf{V} - \mathbf{X}$. Note that an assignment σ agreeing with an assignment σ' on $\mathbf{V} - \mathbf{Y}$ and σ' agreeing with σ'' on $\mathbf{V} - \mathbf{X}$ implies that σ agrees with σ'' on $\mathbf{V} - \mathbf{X}$, because $\mathbf{V} - \mathbf{X} \subseteq \mathbf{V} - \mathbf{Y}$. Therefore, there are $Dim(C, \mathbf{Y}) \times$

$Dim(C-\mathbf{Y}, X)$ assignments in $[C]$ agreeing with each assignment in $[C-\mathbf{X}]$ on $\mathbf{V}-\mathbf{X}$, which means that $Dim(C, \mathbf{X}) = Dim(C, \mathbf{Y}) \times Dim(C-\mathbf{Y}, X)$. □

Finally, we show how to calculate $Dim(C, \mathbf{X})$ without the restriction on $C = (F, \mathbf{V})$ being shattered wrt $\mathbf{X}$.

Theorem A.6. *Given a set of logical variables* $\mathbf{X}$ *and a normalized constraint system* $C = (F, \mathbf{V})$ *(not necessarily shattered wrt* $\mathbf{X}$*), let* S *be the set of constraint systems shattered wrt* $\mathbf{X}$ *such that its element's extensions form a partition of* $[C]$*, according to Theorem A.3. Then there are* $Dim(C, \mathbf{X}) = \sum_{C' \in S} Dim(S, \mathbf{X})$ *assignments in* $[C]$ *agreeing with each assignment* $[C-\mathbf{X}]$ *on* $\mathbf{V}-\mathbf{X}$*.*

Proof. We know that there are $Dim(C', \mathbf{X})$ assignments in each $C' \in S$ agreeing with each assignment in $[C-\mathbf{X}]$ on $\mathbf{V}-\mathbf{X}$. Because $\{[C'] : C' \in S\}$ is a partition of $[C]$, it exhausts $[C]$ and guarantees that $[C'] \cap [C''] = \emptyset$ for $C', C'' \in S, C' \neq C''$, so there are $\sum_{C' \in S} Dim(S, \mathbf{X})$ assignments in $[C]$ agreeing with each assignment $[C-\mathbf{X}]$ on $\mathbf{V}-\mathbf{X}$. □

B Parseq Shattering

As seen in the previous subsection, constraint systems can be broken down into normalized constraint systems shattered wrt some variable, with disjoint extensions. This creates individual systems whose extension is uniform in some sense and therefore more predictable without the need to analyse their grounding. In this subsection we show that a similar notion can be applied to parseqs.

When a parseq is defined on atoms with the same predicate, some instances of it may contain more random variables than others. For example, a parseq $(\{q(X, Y), q(X, Z)\}, \top)$ has instances in which $Y = Z$ and others in which $Y \neq Z$. If this parseq is the argument sequence for a parfactor $q(X, Y) = q(X, Z)$ means that the grounded factor will be defined on a single random variable, while $q(X, Y) \neq q(X, Z)$ means that the grounded factor will be defined over two random variables. This is undesirable because sometimes we rely on the structural uniformity of groundings to manipulate them without the need to consider individual cases.

This motivates the definition of a shattered parseq, that is, a parseq whose all groundings contain the same number of grounded atoms.

Definition B.1. *A parseq* $(\mathbf{A}, C)$ *is* shattered *if every pair of parameterized random variables in* $\mathbf{A}$ *is constrained by* C *to be either identical or different from each other.*

A normalized parseq *is a parseq whose constraint system is normalized and whose equality conjuncts have been eliminated by substituting variables in the parseq's parameterized random variables according to them. This is to say that their constraint system's constraint formula is a conjunction of disequalities. The function normal maps parseqs into their normalized versions.*

Analogously to constraint systems, for any normalized parseq there is a set of normalized shattered parseqs whose constraint systems' extensions form a partition of its constraint system's extension. In order to show how this can be computed, we need to introduce convenient ways of expressing that two atoms must be different or equal, which we call *equality (or disequality) constraints on atoms*, as defined below.

The *equality constraints of atoms* B and C is

$$Eq(B, C) = \bigwedge_{i=1}^{|args(B)|} args(B)_i = args(C)_i$$

if $pred(B) = pred(C) \wedge |args(B)| = |args(C)|$, or $Eq(B, C) = \bot$ otherwise.

The *disequality constraints of atoms* B and C is

$$Dis(B, C) = \bigvee_{i=1}^{|args(B)|} args(B)_i \neq args(C)_i$$

if $pred(B) = pred(C) \wedge |args(B)| = |args(C)|$, or $Dis(B, C) = \top$ otherwise.

More generally, we define $AtConstr$ to be

$$AtConstr(B, C, =) = Eq(B, C)$$

and

$$AtConstr(B, C, \neq) = Dis(B, C).$$

We can now show how to replace a normalized parseq by a set of shattered normalized parseqs.

Theorem B.1. *Let $E = (\mathbf{A}, (F, \mathbf{V}))$ be a normalized parseq. Then $NNS(E)$ (the non-normalized shattering of E) is defined as*

$$\{(\mathbf{A}, (F', \mathbf{V})) : F' = normal(F \wedge \bigwedge_{B,C \in \mathbf{A}, rel \in \{=,\neq\}} AtConstr(B, C, rel))\}$$

and is a set of shattered parseqs such that $\{[(F', \mathbf{V})] : (\mathbf{A}, (F', \mathbf{V})) \in NNS(g)\}$ is a partition of $[(F, \mathbf{V})]$.

Note that while the formulas *in this set are normalized, the* parseqs *are not. We also define $S(E) = \{normal(E') : E' \in NNS(E)\}$ as the version of $NNS(E)$ with normalized parseqs (that is, without equality conjuncts in their constraint systems).*

Proof. This proof is similar to the proof of Theorem A.3.

Every assignment $\sigma \in [(F, \mathbf{V})]$ is in $[(F', \mathbf{V})]$ for some $(\mathbf{A}, (F', \mathbf{V})) \in NNS(E)$, namely the one with F' equal to the normalized form of

$$F \wedge \bigwedge_{B,C \in \mathbf{A}: B\sigma = C\sigma} AtConstr(B, C, =) \wedge \bigwedge_{B,C \in \mathbf{A}: B\sigma \neq C\sigma} AtConstr(B, C, \neq)$$

since $F\sigma$ is true and there will be an element of $NNS(E)$ constructed with constraints on the atoms of $\mathbf{A}$ exactly reflecting whether σ makes them equal or different.

Every assignment $\sigma \in [(F', \mathbf{V})]$ for some $(\mathbf{A}, (F', \mathbf{V})) \in NNS(E)$ is also in $[(F, \mathbf{V})]$ because $F'\sigma$ is true and since F is a conjunct of F', $F\sigma$ must also be true.

$[(F', \mathbf{V})]$ form a partition of $[(F, \mathbf{V})]$ because no assignment σ can be in both $[(F', \mathbf{V})]$ and $[(F'', \mathbf{V})]$ for two distinct F' and F'' such that $(\mathbf{A}, (F', \mathbf{V}))$, $(\mathbf{A}, (F'', \mathbf{V})) \in NNS(E)$. This is so because if F' and F'' are distinct then there is at least one pair of atoms $B, C \in \mathbf{A}$ on which one imposes equality and the other disequality, and σ can only make one of those true. □

C Parseq operations

We show here that operations on sets of ground random variables described by parseqa can be performed on the parseqs themselves.

It is important to show that the extension of a parseq minus a set of ground atoms can be described by the extension of another parseq.

Theorem C.1. *Let R be a parseq and Q be a set of ground atoms. Then*

$$[R] \setminus Q = [(\bigcup_{V \in ParVar(R)} V\sigma_V, \bigwedge_{V \in ParVar(R)} \bigwedge_{W \in Q} (C(R)\sigma_V \wedge \neg C_{V,W}))],$$

where σ_V is an assignment of logical variables $LVar(V)$ to unique names and $C_{V,W}$ is constraint system with the equalities necessary to make $V\sigma_V$ equal to W.

Proof. Let A be a ground atom in $[R] \setminus Q$. Then $A \in [(V\sigma_V, C(R)\sigma_V)]$ for some $V \in ParVar(R)$ but $A \neq W$ for any $W \in Q$, so

$$A \in [(\bigcup_{V \in ParVar(R)} V\sigma_V, \bigwedge_{V \in ParVar(R)} \bigwedge_{W \in Q} (C(R)\sigma_V \wedge \neg C_{V,W}))]$$

by construction of the latter.

Conversely, if $A \in [(\bigcup_{V \in ParVar(R)} V\sigma_V, \bigwedge_{V \in ParVar(R)} \bigwedge_{W \in Q} (C(R)\sigma_V \wedge \neg C_{V,W}))]$, A is a instance of some $V \in ParVar(R)$ but not equal to any $W \in Q$, so it must be in $[R] \setminus Q$. □

D Unifications

There are three types of unifications needed in first-order variable elimination: parseq unification, partial unification and splitting unification.

D.1 Parseq Unification

The most basic of the unifications we use is parseq unification. Parseq unification unifies sets of parameterized random variables, which are just logic

atoms, so it resembles the regular unification used in resolution. However, this is a more general case since it may involve disequalities and logical variables may already be constrained prior to it.

Definition D.1. *Let R and S be parseqs. A normalized constraint system $U = (F, LVar(R) \cup LVar(S))$ is a* unifier of R and S *if $[U] \subseteq [C(R)]$, $[U] \subseteq [C(S)]$ and, for any substitution $\theta \in [U]$, $set(R\theta) = set(S\theta)$, where set is a function returning a set of parameterized random variables of a parseq (which is a sequence and therefore ordered – this is to say that the equality here is defined by regarding the parseqs as sets, not sequences).*

The unification between R and S is said to fail *if there is no unifier of R and S and to* succeed *otherwise.*

[Bar98] shows that there is a unifier of R and S if and only if $[R] \cap [S] \neq \emptyset$, for the case where they are unary parseqs, and how to find it.

A unifier describes the set of instantiations common to both parseqs R and S, and its constraint system is defined on the logical variables of both of them. However, we can choose one of them, say, R, and transform the unifier into an equivalent parseq whose constraint system is defined on $LVar(R)$ alone, as the next theorem shows.

Theorem D.1. *Let U be the unifier of normalized parseqs R and S. The set $[(ParVar(R), U)]$, which is by definition equal to $[(ParVar(S), U)]$, is also equal to a set $[(ParVar(R), U')]$ where U' is a constraint system containing logical variables in $LVar(R)$ only.*

Proof. Consider the constraint system U and assume it in a normalized form. Because it is a product of unification, the logical variables in $LVar(S)$ can only have disequality constraints in relation to constants and other variables in $LVar(S)$, and can only have equality constraints with constants or variables in $LVar(R)$. In particular, there cannot be disequalities between variables in $LVar(R)$ and $LVar(S)$, as mentioned in [Bar98]. We then construct a new constraint system on $LVar(R)$ as follows:

$$U' = \bigcup_{c' \in \mathcal{C}} c'$$

where

$$\mathcal{C} = \bigcup_{c \text{ is a conjunct of } U} s(c)$$

and $s(c)$ maps each conjunction in U to either a unary set or the empty set in the following manner:

$$s(c) = \begin{cases} \{c\}, & \text{if } c \text{ does not involve variables in } LVar(S), \\ \{Y = \alpha\}, & \text{if } c \text{ is of the form } X = \alpha \text{ for } X \in LVar(S) \text{ and} \\ & X = Y \text{ is a conjunct of } U \text{ for some } Y \in LVar(R), \\ \emptyset, & \text{if } c \text{ is an equality } X = \alpha \text{ for } X \in LVar(S) \text{ and} \\ & \text{there is no other conjunct} \\ & X = Y \text{ in } U, \text{ for } Y \in LVar(R). \end{cases}$$

From this transformations one can verify, since the variables in $LVar(S)$ do not occur in R, that $[(ParVar(R), U')] = [(ParVar(R), U)]$. □

Theorem D.2. *Let U be a unifier of parseqs R and S. U* defines assignments *σ_R and σ_S such that $R\sigma_R = U$ and $S\sigma_S = U$.*

Proof. The existence of these assignments is a consequence of the fact that the unification in [Bar98] guarantees an equality, for each logical variable in the parseqs, to either a constant or logical variable in the other parseq. sig_R (sig_S), then, will map each variable in R (S) to either a constant, itself, or a logical variable in S (R), depending on the logical variables actually being used in U. □

D.2 Partial unification

The second type of unification is partial unification between parseqs. Sometimes it is not possible to find a unifier of two parseqs, but a unifier on a subset of the parameterized random variables of those parseqs, which is a useful operation.

Definition D.2. *Let R and S be two parseqs. A* partial unifier *of R and S is a parseq E such that $[E] \neq \emptyset$, $[E] \subseteq [R] \cap [S]$ and $LVar(C(E)) = LVar(E)$ (i.e., the constraint system of E is not allowed to contain logical variables not in $LVar(E)$).*

The partial unification of R and S is said to fail *if there is no partial unifier of R and S and to* succeed *otherwise.*

Note that a unifier is a constraint system but a partial unifier is a parseq. The reason for this difference is that the partial unifier must contain the information on what part of the original parseqs the unification has been performed.

Theorem D.3. *Let R and S be two parseqs. There is a partial unifier of R and S if and only if $[R] \cap [S] \neq \emptyset$.*

Proof. Suppose there is a partial unifier E of R and S. Then $[E]$ is not empty and $[E] \subseteq [R] \cap [S]$, therefore $[R] \cap [S] \neq \emptyset$. Conversely, suppose $[R] \cap [S] \neq \emptyset$. Then there is a ground random variable A both in $[R]$ and $[S]$ which is an instance of $V \in ParVar(R)$ and of $W \in ParVar(S)$. Therefore, we can find a unifier U of parseqs $(V, C(R))$ and $(W, C(S))$, accordingl to [Bar98]. Let us assume U to be normalized, and define U' to be the conjunction of conjuncts of U that only contain logical variables in $LVar(V) \cup LVar(W)$. All assignments to the variables in $LVar(V) \cup LVar(W)$ in U will thus be preserved in U', so U' is also a unifier of parseqs $(V, C(R))$ and $(W, C(S))$. According to Theorem D.1, there is another unifier U'' with logical variables from $LVar(V)$ only (or from $LVar(W)$ only). From this, by definition, (V, U'') (or (W, U'')) is a partial unifier of R and S. □

D.3 Splitting unification

We have already defined splitting unification in section 3.3.1 and stated some theorems about it, which we re-state and prove here.

Proof of Theorem 3.1:

Theorem D.4. *Let $\mathbf{R}$ be a set of parseqs and E a parseq. There is at least one element $R \in \mathbf{R}$ such that $[E] \cap [R] \neq \emptyset$ if and only if there is a splitting unifier E' of $\mathbf{R}$ given E.*

Proof. We prove this by induction on $|\mathbf{R}|$. If $|\mathbf{R}| = 0$ then both directions of the theorem are trivial, based on the observation that there is no element $R \in \mathbf{R}$ whose extension intersects $[E]$ since $\mathbf{R}$ is empty.

If $|\mathbf{R}| \geq 1$, the "if" part of the theorem is tru by the definition of splitting unifier. For the "only if" part, let $R \in \mathbf{R}$ such that $[E] \cap [R] \neq \emptyset$. Then, according to Theorem D.3, there is a partial unifier E' of E and R. Let us

now consider, by induction, the splitting unification of $\mathbf{R}' = \mathbf{R} \setminus \{R\}$ given E' as seed. If this splitting unification fails then for every element $R' \in \mathbf{R}'$, $[R'] \cap [E'] = \emptyset$. This means, by definition, that E' is a splitting unifier for $\mathbf{R}$ given seed E. If this splitting unification succeeds, there is a splitting unifier E'' of $\mathbf{R}'$ such that $[E''] \subseteq [E'] \subseteq [E]$ and $[E''] \subseteq [R]$. Because E'' is a splitting unifier for $\mathbf{R}'$ and $[E'']$ intersects $[R]$, E'' is a splitting unifier for $\mathbf{R}$ given seed E. □

In order to prove Theorem 3.2, we first prove the following lemma:

Lemma D.1. *Let* $\mathbf{R}$ *be a set of parseqs and* Q *a set of ground atoms. If there is a splitting unifier of* $\mathbf{R}$ *with seed* $GVar(G) \setminus Q$*, then there is a valid splitting unifier of* $\mathbf{R}$ *with seed* $GVar(G) \setminus Q$*.*

Proof. Let E be a splitting unifier of $\mathbf{R}$ with seed $GVar(G) \setminus Q$. We prove the theorem by induction on the number of logical variables in $\mathbf{R}$ left uncovered by E. If this number is 0, then E is a valid splitting unifier wrt $GVar(G) \setminus Q$.

If $E > 0$, then let V be a parameterized random variable in $ParVar(R)$, for some $R \in \mathbf{R}$ such that $[R] \cap [E] \neq \emptyset$, such that a logical variable X uncovered by E occurs in V. Let V' be the parseq describing $[(V, C(R))] \setminus Q$ according to Theorem C.1. $[V']$ is not an empty set because the query is ground and V is parameterized. (V' represents ground random variables that need to be eliminated, because they are not in the query, while coming from the grounding of a logical variable uncovered by E.)

Let $W \subseteq ParVar(R)$ such that there is a parseq (W, C) with $[(W, C)] = [E]$. These are simply the parameterized random variables in R that have been unified to parameterized random variables in E. Let E' be the parseq $(W \cup \{V\}, C(E) \wedge C(R))$. By the construction of E', $[E'] \cap R \neq \emptyset$, so, according to Theorem 3.1, there is a splitting unifier E'' of $\mathbf{R}$ for seed E'. In particular, because V is unified to some part of E'', X is covered by E''. Moreover, any logical variable Y that was already covered by E is covered by E'' as well. This is so because any parseq $S \in \mathbf{R}$ such that $[S]$ intersects $[E'']$ must have a set of parameterized random variables T unifying with $ParVar(E)$, so the logical variables in T must have unified to logical variables in E and W, so they are also unified to logical variables in E' and thus covered by E''. By induction, then, there must be a valid splitting unifier for $\mathbf{R}$ with seed $GVar(G) \setminus Q$. □

Theorem D.5. *For any set* $\mathbf{R}$ *of parseqs and* Q *a set of ground atoms such that* $GVar(\mathbf{R}) \setminus Q \neq \emptyset$ *there is a valid splitting unifier* E *of* $\mathbf{R}$ *with seed* $GVar(R) \setminus Q$*.*

Proof. Because $GVar(\mathbf{R}) \setminus Q \neq \emptyset$, there is a parseq D such that $[D] \subseteq GVar(\mathbf{R})$ and $[D] \cap Q \neq \emptyset$ (by Theorem C.1). This means, by Theorem D.4, that there is a splitting unifier D' of $\mathbf{R}$. From Lemma D.1, it follows that there is a valid splitting unifier. □