Towards Weighted Default Reasoning

NATHANIEL WESLEY FILARDO and JASON EISNER
Johns Hopkins University, Baltimore, MD
(*e-mail:* {nwf,jason}@cs.jhu.edu)

submitted 4 May 2017; revised 4 May 2017; accepted ?

Abstract

Our concurrent paper [4] has considered the semantics of an arithmetic circuit solver [3] extended to work on potentially infinite circuits that are described by weighted logic programs. That paper assumed a sound, decidable theory of sets of terms. Unfortunately, the required operations of set subtraction, intersection, union, projection, subset testing, and cardinality counting are not all expressible or decidable in classes of tree automata sufficiently capable to describe runtime quantities within our weighted logic programs. The present work investigates the possibility of eliminating the need for set difference by representing a (piecewise-constant) function on trees as a finite collection of constant functions on varying domains. Functions on narrower domains override the "default" values given for wider domains. The last vestige of set subtraction is encapsulated within a cardinality counting operation (where the difference itself need not be expressed as an automaton).

1 Introduction

Logic programs, written in, e.g., Prolog, consist of inference rules such as "`rs(X) :- r(X,Y), s(Y)`". This rule asserts that `rs(x)` is true if ∃`y` such that `r(x,y)` and `s(y)` are both true. A *weighted* logic program is a definition of a generalized computational circuit. The nodes of such a circuit are termed **items** and constraints on their values may be specified using Prolog-like rules such as `rs(X) ⊕= r(X,Y) ⊗ s(Y)`, which effectively defines a vector `rs` by adding (⊕) the product (⊗) of the matrix `r` and the vector `s`. SLD resolution of Prolog programs [6] can reason about *sets* of terms at once, using free variables, answering some *infinite* questions in finite time. Practical extension of set-at-a-time reasoning to weighted programs remains an open problem.

Our concurrent paper [4], whose introduction we reproduce in summary form here, gives a sketch of an algorithm for *set-at-a-time* reasoning about weighted logic programs, assuming the ability to perform algebraic manipulation of sets of items. In particular, our method for carrying out one step of backward computation, Compute, depended upon *explicit set subtraction* so that, given a query for possibly infinitely many items κ, it could respond with a *finite partition* wherein each (possibly infinite) subset was associated with a *single value*. Unfortunately, set subtraction is difficult to explicitly compute; many classes of tree automata are not closed under set subtraction [2].[1] The present work investigates one possible route to eliminating the need for set subtraction, or, rather, encapsulating it behind a weaker operation. We *encode* partitions of items using a recursive

[1] One can, of course, *represent* set difference by maintaining a pair of sets and interpreting appropriately. Unfortunately, this representation has limitations, not the least of which is that it is not, itself, closed under subtraction.

notion of *defaults*: rather than describing the behavior of each of $\{0\}$, $\mathbb{N} \setminus 0$ and $\mathbb{Z} \setminus \mathbb{N}$ (which together partition $\mathbb{Z}$), we now would describe behaviors of $\{0\}$, $\mathbb{N}$, and $\mathbb{Z}$, with the understanding that *the most precise description available* is the one to use, overriding any "default" behaviors associated with the broader descriptions. That is, 0 would behave as described by $\{0\}$, -1 by $\mathbb{Z}$, and 1 by $\mathbb{N}$.

As in our companion paper, our treatment is intended to be applicable to any weighted logic language. Our particular formalism, μDyna, pronounced "micro-Dyna," is built up of little more than set theory and has a straightforward, declarative semantics, detached from any particular implementation, and devoid of metalogical escapes. After brief reviews of notation (§1.1) and μDyna (§1.2), we describe the particular set-theoretic black box we assume (§1.3) and the above idea of encoding functions by defaults (§2). We motivate our set-at-a-time reasoning with an example (§3.1) which guides discussion (§3.2) and formalization (§3.3-3.6) of default reasoning. We present a pseudocode listing in §4, detailing its execution on our example, and conclude by comparing related work in §5.

1.1 Review of Notation

We use the notation of §1.1 of [4], slightly extended. This section is largely a summary of that earlier work; new notation will be explicitly marked as such.

Sets Sets are manipulated by the standard operators (e.g., $\{\ldots\}$, $\cup$, $\cap$, $\in$, $\subseteq$, $\setminus$). $\wp$ sends a set to its powerset; $\wp_{\text{fin}}$ sends a set to its set of *finite* subsets. $|\sigma|$ denotes the cardinality of σ, one of $\mathbb{N}_\infty \stackrel{\text{def}}{=} \mathbb{N} \cup \{\infty\}$. The *partial* function $\text{selt}(\sigma)$ projects a singleton set to its element: $\text{selt}(\{s\}) \stackrel{\text{def}}{=} s$. We use the shorthands $\bigcup \sigma \stackrel{\text{def}}{=} \bigcup_{s \in \sigma} s$ and $\mathbb{N}_1^n \stackrel{\text{def}}{=} \{1, 2, \ldots, n\} \subset \mathbb{N}$.

Bags $\{\!|s@m|\!\}$ denotes a bag holding exactly m copies of s, also said as s with **multiplicity** $m \in \mathbb{N}_\infty$ (with 0 being identified with absence from the bag). Multiplicities of 1 may be suppressed: $\{\!|s|\!\} = \{\!|s@1|\!\}$. As with sets, comprehension notation may be employed, quantifying over both elements and multiplicities. The traditional symbols of set-theory with a plus sign superimposed will be used for bag operations: $\uplus$, $\biguplus$, $\dot{\in}$, etc., though we overload $\varnothing$ for the empty bag as well. $\wp_+\beta$ denotes the *set* of all sub-bags of bag β. $U_m^{-1}\sigma$ is the bag whose elements are from σ, all with multiplicity m. *New in this paper*, we define $v \sqsubseteq_{=m} \beta$ to mean that v occurs *exactly* m times in β.

Tuples We denote n-tuples as $\langle t_i \rangle_{i \in \mathbb{N}_1^n} \stackrel{\text{def}}{=} \langle t_1, \ldots, t_n \rangle$, and use $\vec{t}$ when n is clear from context. A **pair** is a tuple of length 2. $+\!\!+$ is the associative tuple concatenation operator. The length of a tuple is denoted $\text{tlen}\langle t_1, \ldots, t_n \rangle \stackrel{\text{def}}{=} n$; $\langle\rangle$ is the tuple of length 0. Nested tuples may be written with color-matched brackets for ease of reading, e.g., $\langle\langle\rangle, \langle\langle a \rangle, b\rangle\rangle$. Projection selects an element within a tuple: $\vec{t}\downarrow_k \stackrel{\text{def}}{=} t_k$, for $\vec{t}$ an n-tuple and $k \in \mathbb{N}_1^n$. Nested tuples can be projected along **paths** (a tuple of positive integers): $\vec{t}\downarrow_{\ldots k_2.k_1} \stackrel{\text{def}}{=} \vec{t}\cdots\downarrow_{k_2}\downarrow_{k_1}$.

Sets of Tuples Dependent sums are written $\Sigma_{s \in \sigma} Y_s \stackrel{\text{def}}{=} \{\langle s, t_s \rangle \mid s \in \sigma, t_s \in Y_s\}$, where Y is a σ-indexed collection of sets. As we will often have sets described by tuples of elements sampled from a product of other sets, we introduce a product-forming tuple operator, $\langle \sigma, \tau \rangle \stackrel{\text{def}}{=} \{\langle s, t \rangle \mid s \in \sigma, t \in \tau\}$. Projection is extended to sets, $\sigma\downarrow_\pi \stackrel{\text{def}}{=} \{\sigma\downarrow_\pi \mid s \in \sigma\}$, and a new form is added to preserve cardinalities: $\sigma\downarrow_\pi^{@} \stackrel{\text{def}}{=} \{\!|s\downarrow_\pi \mid s \in \sigma|\!\}$. **Refinement** filters a set of tuples by a projection: $\sigma[\tau/\pi] \stackrel{\text{def}}{=} \{s \in \sigma \mid s\downarrow_\pi \in \tau\}$.

Functions Sets of functions are denoted with the dependent product operator: $\Pi_{s \in \sigma} Y_s$ where Y is a σ-indexed collection of *sets*; when Y is contstant (i.e., $\exists_\tau \forall_{s \in \sigma} Y_s = \tau$), we write $\sigma \to \tau$. The domain of a function $f \in \Pi_{s \in \sigma} Y_s$ is denoted $\text{dom}(f) \stackrel{\text{def}}{=} \sigma$. Functions

may be written using set notation, as $\{s \mapsto t \mid \cdots\}$, where s is in the domain and t the (dependent) codomain; $\mapsto$ is an infix pair constructor ($(a \mapsto b) \stackrel{\text{def}}{=} \langle a, b\rangle$). We also use this notation in quantification, e.g., $\{\varphi(s,t) \mid s \mapsto t \in f\}$, to range over the domain of a function. Functions can be constructed out of other notation by use of an **argument placeholder**, "$\cdot$": e.g., $a_\cdot \stackrel{\text{def}}{=} \{n \mapsto a_n \mid n \in \alpha\}$ for appropriate α.

Terms The items and values of our language, **ground terms**, are from a Herbrand universe $\mathcal{H}$ over $\mathcal{F}$. $\mathcal{H}$ is the smallest set such that $\mathtt{f}/n \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{H}$ implies that $\mathtt{f}\langle t_1, \ldots, t_n\rangle \in \mathcal{H}$. A **non-ground term** (synonymously, **type**) is a subset of $\mathcal{H}$. We use the product-forming tuple operator as a shorthand for non-ground terms: $\mathtt{f}\langle \tau_1, \ldots, \tau_n\rangle \stackrel{\text{def}}{=} \{\mathtt{f}\langle t_1, \ldots, t_n\rangle \mid \forall_i t_i \in \tau_i\}$. Projection is extended to work on trees as well as on tuples by ignoring any functors along the path. The symbol $\text{NULL} \notin \mathcal{H}$ will indicate the absence of a value assigned to an item. As shorthands, we define $\mathcal{H}' \stackrel{\text{def}}{=} \{\text{NULL}\} \cup \mathcal{H}$, $\mathcal{H}^+ \stackrel{\text{def}}{=} \wp_+ U_\infty^{-1} \mathcal{H}$ (all bags of terms), and $\mathcal{H}'^+ \stackrel{\text{def}}{=} \wp_+ U_\infty^{-1} \mathcal{H}'$ (and NULL).

Aggregation Functions Aggregation of multiple values associated with an item (**aggregands**) to a single value is carried out by **aggregators**, functions $f \in \mathcal{H}'^+ \to \mathcal{H}'$ which obey $f(\varnothing) = f(\{\!|\text{NULL}|\!\}) = f(\{\!|\text{NULL@}\infty|\!\}) = \text{NULL}$, $\forall_{a \in \mathcal{H}} f(\{\!|a|\!\}) = a$, and $\forall_{\sigma,\sigma'} f(\sigma \uplus \sigma') = f(\{\!|f(\sigma)|\!\} \cup \sigma')$. NULL is a unit of every aggregator.

1.2 μDyna Normal-Form Programs

We review our formalism for programs; this is largely condensed from §1.2 and §3.2 of [4].

Rules within a μDyna program are indexed by a finite set Ξ and are sets of nested tuples over terms. The base case of the nesting is a **kv-pair**, which pairs together a key and a value (in that order, i.e., $\langle \text{key}, \text{value}\rangle$), each from $\mathcal{H}$ (not $\mathcal{H}'$). Each entry within a rule's set of tuples, ρ_r, is a pair of the **head/result kv-pair** (at $\text{HR} \stackrel{\text{def}}{=} 1$; additionally, $\text{HEAD} \stackrel{\text{def}}{=} 1.1$ and $\text{RES} \stackrel{\text{def}}{=} 1.2$) and a n_r-tuple of **subgoal** kv-pairs (at $\text{SG} \stackrel{\text{def}}{=} 2$). Our example of a weighted logic language rule from above, `rs(X) ⊕= r(X,Y) ⊗ s(Y)`, is, in μDyna, $\{\langle\langle \mathtt{rs}\langle x\rangle, z\rangle, \langle \mathtt{r}\langle x, y\rangle \mapsto r, \mathtt{s}\langle y\rangle \mapsto s, \otimes\langle r, s\rangle \mapsto z\rangle\rangle \mid r, s, x, y, z \in \mathcal{H}\}$. Each element reads as an instruction: "contribute the result to the head (for aggregation) if each subgoal's key has been assigned the corresponding value;" e.g., $\langle\langle \mathtt{rs}\langle 1\rangle, 0\rangle, \langle\langle \mathtt{r}\langle 1,3\rangle, 2\rangle, \langle \mathtt{s}\langle 3\rangle, 5\rangle, \langle \otimes\langle 2,5\rangle, 0\rangle\rangle\rangle$ says to contribute 0 to $\mathtt{rs}\langle 1\rangle$ if $\mathtt{r}\langle 1,3\rangle \mapsto 2$, $\mathtt{s}\langle 3\rangle \mapsto 5$, and $\otimes\langle 2,5\rangle \mapsto 0$.

Formally, sets ρ_r used as μDyna rules obey five constraints: ① projections along HEAD, RES, and SG are defined for all elements of the set; ② the head and result are terms, i.e., $\forall_{t \in \rho_r, \pi \in \{\text{HEAD,RES}\}} t\downarrow_\pi \in \mathcal{H}$; ③ the number of subgoals in r, denoted n_r, is *constant* across all groundings of the rule, i.e., $\forall_{r \in \Xi, s \in \rho_r\downarrow_{\text{SG}}} \text{tlen}(s) = n_r$; ④ each subgoal is itself a pair of two terms, i.e., $\forall_{t \in \rho_r, i \in \mathbb{N}_1^{n_r}, j \in \{1,2\}} t\downarrow_{\text{SG}.i.j} \in \mathcal{H}$; and ⑤ the subgoals and head determine the grounding, i.e., $\forall_{\alpha \subseteq \rho_r} |\alpha\downarrow_{\text{SG}}| = |\alpha\downarrow_{\text{HEAD}}| = 1 \Rightarrow |\alpha| = 1$. A **rule query** $\vec{t}$ for a rule r is a n_r-tuple of items. A rule query gives rise to a set of **pre-answers**, $\theta_r^{\vec{t}} \stackrel{\text{def}}{=} \rho_r[\{t_1\}/\text{SG}.1.1] \cdots [\{t_{n_r}\}/\text{SG}.n_r.1]$. $\vec{t}$ is trivial if $\theta_r^{\vec{t}} = \varnothing$. Given valuations v_i for each t_i, one can filter pre-answers to a set of **rule answers**, $\epsilon_{r,\vec{v}}^{\vec{t}} \stackrel{\text{def}}{=} \theta_r^{\vec{t}}[\{v_1\}/\text{SG}.1.2] \cdots [\{v_{n_r}\}/\text{SG}.n_r.2]$. If any $v_i = \text{NULL}$, then ϵ is $\varnothing$. The constraints on μDyna rules imply that $\forall_{h \in \mathcal{I}, r, \vec{t}, \vec{v}} |\epsilon_{r,\vec{v}}^{\vec{t}}[\{h\}/\text{HEAD}]| \leq 1$. **Non-ground rule queries** are, similarly, tuples of *sets of* items. The corresponding **non-ground pre-answers** are unions of the ground pre-answers: $\theta_r^{\vec{\tau}} \stackrel{\text{def}}{=} \bigcup_{\vec{t} | \forall_i t_i \in \tau_i} \theta_r^{\vec{t}} = \rho_r[\tau_1/\text{SG}.1.1] \cdots [\tau_{n_r}/\text{SG}.n_r.1]$. **Non-ground rule answers** ϵ are properly defined using *valuation functions* f_i for each subgoal; however, in this paper, we restrict to *constant valuation functions* and will simply give their output v_i for each. That is, $\epsilon_{r,\vec{v}}^{\vec{\tau}} \stackrel{\text{def}}{=} \bigcup_{\vec{t} | \forall_i t_i \in \tau_i} \epsilon_{r,\vec{v}}^{\vec{t}} = \rho_r[\{v_1\}/\text{SG}.1.2] \cdots [\{v_{n_r}\}/\text{SG}.n_r.2] \subseteq \theta_r^{\vec{\tau}}$.

A μDyna program as a whole consists of several components: ① its set of items, $\mathcal{I} \subseteq \mathcal{H}$; ② a *bag* of ($\mu$Dyna) rules as defined, $\{\!|\rho_r \mid r \in \Xi|\!\}$, where Ξ is a *finite set*; and ③ an assignment of items to aggregation operators, $\text{aggr}(\cdot) \in \mathcal{I} \to (\mathcal{H}'^{+} \to \mathcal{H}')$.

1.3 Required Set Theory Operations

The development of this paper aims to eliminate the need to compute and represent *differences* of sets of trees, as part of a more general program of lowering the requirements of algorithmic representations of sets. In general, our requirements will remain quite high. We require the ability to describe sets that correspond to all instantiations of a non-ground term (which may contain repeated variables). We also require our family of representable sets to be closed under finite unions and intersections. Finally, we require the ability to count cardinalities of set differences (without representing the differences themselves). This last implies that we can also perform subset testing and equality testing.

Ultimately, we have achieved our modest goal of eliminating explicit set difference; the catch, however, is that we still need the ability to extract *cardinalities* of subtractions. In [4], we ultimately required that all non-ground rule answers ϵ were *uniform* in their contributions to their heads: each $h \in \epsilon\downarrow_{\text{HEAD}}$ had to be associated with *the same* bag of aggregands. Now, we will require a similar kind of uniformity of rule answers, in particular, uniformity on a "surviving" subset. Recognition of this uniformity, with a little irony, encapsulates *two* set subtractions yet appears to be a *slightly* lower requirement. In particular, given a non-ground rule answer $\epsilon \subseteq \rho$, an "obstructed head" set $\omega \subseteq \mathcal{I}$, and a "masked" set of rule groundings $\mu \subseteq \rho$, we define $\textsc{AnswerFor}(\epsilon, \omega, \mu)$ to equal β iff $\forall_{h \in (\epsilon\downarrow_{\text{HEAD}} \setminus \omega)} (\epsilon \setminus \mu)[\{h\}/\text{HEAD}]\downarrow^{@}_{\text{RES}} = \beta$. We can read this as "every head in $\epsilon\downarrow_{\text{HEAD}}$, excepting the obstructed ω, is, after masked groundings are removed, associated with the same bag-view RES projection, β." For simplicity, we will restrict to βs of the form $\{\!|v@m|\!\}$ for some $v \in \mathcal{H}$ and $m \in \mathbb{N}_\infty$.

2 Encoding Functions with Finite Ranges

In a large subset of μDyna programs, we need to find and manipulate functions $f \in \kappa \to \mathcal{H}^+$ which have possibly infinite domain κ but only *finite range*, where the range is only known at runtime. We review the encoding used by [4] before exhibiting a "default"-based encoding which avoids the need for set subtraction in construction and may be interpreted with only cardinality of subtractions.

Encoding by Domain Partition A piecewise-constant function with finite range, $f \in \kappa \to \alpha$, can be described by a finite set of pairs $\langle \kappa_i, a_i \rangle$ with disjoint $\kappa_i \subseteq \kappa$ and $a_i \in \alpha$. This, in turn, can be made into a function $f' = \bigcup\{\kappa_i \mapsto a_i\}$ so that $f(k) = f'(\kappa_i)$ with κ_i the *unique* element of $\text{dom}(f')$ which contains t. Given two such encodings, f'_i, of functions with a common domain, $f_i \in \kappa \to \alpha$, and a binary operator $\oplus$ on α, it is easy to build the **$\oplus$-join** of g_1 and g_2: $g_1 \wedge^{\oplus} g_2 \in \kappa \to \alpha$; it is $\{(\tau_1 \cap \tau_2 \neq \varnothing) \mapsto g'_1(\tau_1) \oplus g'_2(\tau_2) \mid \tau_i \in \text{dom}(g'_i)\}$. If, however, the g_i have different domains, $g_i \in \kappa_i \to \alpha_i$, and we wish for the join to have $\kappa_1 \cup \kappa_2$ as its domain, we must first specify values for the novel part of the domain for each g_i. Specifically, we must add (partitions of) $\kappa_2 \setminus \kappa_1$ to $\text{dom}(g_1)$, and $\kappa_1 \setminus \kappa_2$ to $\text{dom}(g_2)$, associated with identity elements of $\oplus$. The need to compute set differences is unavoidable.

The non-ground reasoning algorithm of [4] maintains its collections of aggregands to head items using exactly this encoding; its DISJOIN function, which adds the collection of aggregands β to each head $h \in \eta$ across the encoding f' and yields an appropriately

adjusted encoding, is $f' \wedge^{\uplus} \{\eta \mapsto \beta\}$ disguise:

```
def DISJOIN(f', η, β)  % Add all of β to each h ∈ η across all of f'
  return {(κ ∩ η) ↦ β ⊎ τ | (κ ↦ τ) ∈ f', κ ∩ η ≠ ∅}  % common domain elements
  ∪ {(η ∖ ⋃(dom(f'))) ↦ β | η ∉ ⋃(dom(c))} ∪ {(κ ∖ η) ↦ τ | (κ ↦ τ) ∈ f', κ ∉ η}
```

(enlarge f''s domain; enlarge novel domain)

Encoding by Defaults A **pointwise-constant backed-off function** (BF), $\mathcal{F} \in \kappa \hat{\to} \alpha$, encodes any function $f \in \kappa \to \alpha$, with α finite, using a function on *potentially overlapping* subsets of $\wp\kappa$. The **base** of $\mathcal{F}$, $T_{\mathcal{F}} \subseteq \wp\kappa$, is the domain of this encoding function, i.e., $\mathcal{F}$ is of the form $T_{\mathcal{F}} \to \alpha$. A BF has three preconditions on its base: it must be **intersection-closed** (i.e., $\forall_{\tau_i \in T_{\mathcal{F}}} \tau_1 \cap \tau_2 \in T_{\mathcal{F}}$), it must be a **cover** of α (i.e., $\bigcup T_{\mathcal{F}} = \alpha$),[2] and it must contain $\varnothing \in T_{\mathcal{F}}$.[3] $\cap$-closure allows us to define the **encloser** of $\sigma \subseteq \kappa$ in $\mathcal{F}$: $\lceil\sigma\rceil_{\mathcal{F}}$ (or just $\lceil\sigma\rceil$, when clear) is the *smallest* $\tau \in T_{\mathcal{F}}$ such that $\sigma \subseteq \tau$, if it uniquely exists.[4] The BF $\mathcal{F}$ encodes f by taking $f(k) = \mathcal{F}(\lceil\{k\}\rceil)$. We extend application notation to $\mathcal{F}(k)$ (since $k \in \kappa$ and not $\wp\kappa$, this is unambiguous). If $T_{\mathcal{F}}$ is finite, $\mathcal{F} \in \kappa \hat{\to}_{\text{fin}} \alpha$ ($\subseteq \kappa \hat{\to} \alpha$).

The $\wedge^{\oplus}$ of two $\mathcal{F}_i \in \kappa \hat{\to} \alpha$ is much as before: $\mathcal{F}_1 \wedge^{\oplus} \mathcal{F}_2 \in \kappa \to \alpha$ has base $T = \{\sigma_1 \cap \sigma_2 \mid \sigma_i \in T_{\mathcal{F}_i}\}$ and sends $\tau \in T$ to $\mathcal{F}_1(\lceil\tau\rceil_{\mathcal{F}_1}) \oplus \mathcal{F}_2(\lceil\tau\rceil_{\mathcal{F}_2})$.[5] Given a BF $\mathcal{F} \in \kappa \hat{\to} \alpha$, we can enlarge its domain to ensure that it is a superset of η by ensuring that η and $\{\eta \cap \sigma \mid \sigma \in T_{\mathcal{F}}\}$ exist in the enlarged base. No longer needing to partition, we have no need for set subtraction.

Of course, as BFs are just an *encoding* of a function, just as is the partition-based scheme above, it is possible, and hopefully illustrative, to consider converting a BF $\mathcal{F}$ to a partition encoding f'. The subset of κ strictly enclosed by $\tau \in T_{\mathcal{F}}$ (and not by some subset of τ) is $u(\tau) = \tau \setminus \bigcup\{\sigma \in T_{\mathcal{F}} \mid \sigma \subseteq \tau\}$. Thus, $K = \{u(\tau) \mid \tau \in T_{\mathcal{F}}\}$ is a partition of κ, and $f'(\kappa') = \mathcal{F}(\kappa')$ is constant on each $\kappa' \in K$.

Example 1. Consider the three rules $\{\langle\langle \mathtt{r}\langle x, y\rangle, 2\rangle, \langle\rangle\rangle \mid x, y \in \mathbb{Z}\}$, $\{\langle\langle \mathtt{r}\langle x, x\rangle, 1\rangle, \langle\rangle\rangle \mid x \in \mathbb{Z}\}$, and $\{\langle\langle \mathtt{r}\langle 0, 0\rangle, 1\rangle, \langle\rangle\rangle\}$.[6] If aggregated by sum, their combined contributions form a three-way partition of $\mathtt{r}\langle\mathbb{Z}, \mathbb{Z}\rangle$: $\{\mathtt{r}\langle 0, 0\rangle \mapsto 4\} \cup \{\mathtt{r}\langle x, x\rangle \mapsto 3 \mid x \in \mathbb{Z} \setminus \{0\}\} \cup \{\mathtt{r}\langle x, y\rangle \mapsto 2 \mid x, y \in \mathbb{Z}, x \neq y\}$. This same result can be readily encoded as a pointwise-constant BF without the need for set subtraction: $\{\{\mathtt{r}\langle 0, 0\rangle\} \mapsto 4, \{\mathtt{f}\langle x, x\rangle \mid x \in \mathbb{Z}\} \mapsto 3, \mathtt{r}\langle\mathbb{Z}, \mathbb{Z}\rangle \mapsto 2\}$. ◊

Example 2. A rule in which values covary with the head, such as $\{\langle\langle \mathtt{f}\langle x\rangle, x\rangle, \langle\rangle\rangle \mid x \in \mathcal{H}\}$ does not give rise to contributions amenable to pointwise-constant BFs. We defer to future work relaxing the pointwise-constancy requirement. ◊

3 Default Reasoning

3.1 A Motivating Example

To motivate and guide our discussion of default reasoning using BFs, we tell again, in Figure 1 the story from [4] of computing the product of an infinite matrix with an infinite

[2] This ensures that $\mathcal{F}$ still acts as a total function on its domain. Practically, this often means that $X \in T_{\mathcal{F}}$, usually with $\mathcal{F}(\tau)(t) = \text{NULL}$ when $\text{NULL} \in \alpha$.

[3] $\mathcal{F}(\varnothing)$ cannot influence the decoded meaning of $\mathcal{F}$; $\varnothing \in T_{\mathcal{F}}$ just for simplicity of definitions.

[4] Not having required that $T_{\mathcal{F}}$ be closed under $\cup$, i.e., having required that it be only a *semi*-lattice under $\subseteq$, we are not ensured that an arbitrary subset of κ has an encloser. Enclosers are, however, certainly defined for any subsets of any $\sigma \in T_{\mathcal{F}}$ and for all singleton subsets of κ.

[5] The use of $\lceil\tau\rceil$ as arguments to $\mathcal{F}_i$ is perhaps surprising, but necessary: there may be multiple pairs $\langle\tau_1, \tau_2\rangle$ with $\tau_i \in T_{\mathcal{F}_i}$ that have the same intersection τ.

[6] Prolog-style syntax does not obviously offer a convenient way to write the μDyna rule $\{\langle\langle \mathtt{r}\langle x, y\rangle, 2\rangle, \langle\rangle\rangle \mid x, y \in \mathbb{Z}\}$; introducing explicit annotations might allow something like `r(X :int, Y :int) += 2`.

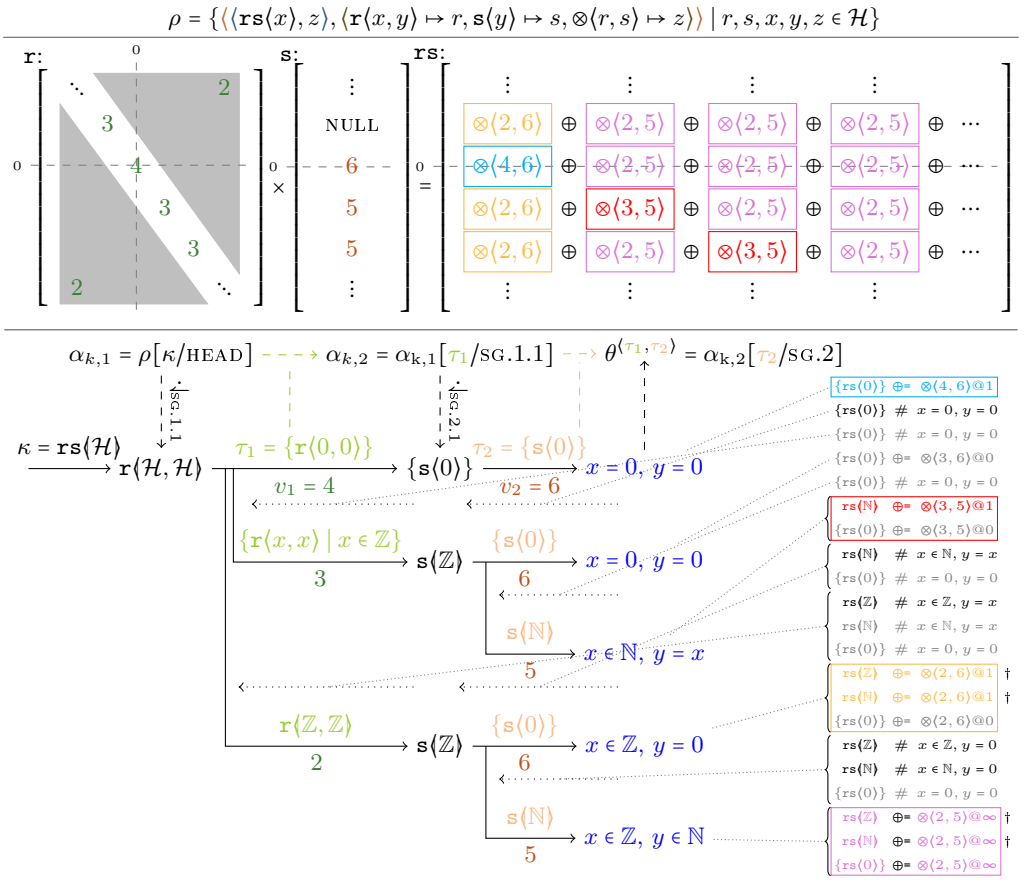
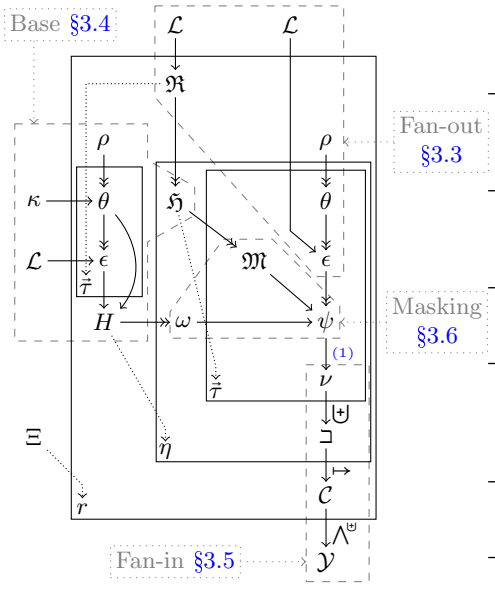


Fig. 1: Our rule `rs(X) ⊕= r(X,Y) ⊗ s(Y)`, shown in μDyna form at the top, can perform the computation shown in the middle—the product of an infinite matrix (all of whose off-diagonal elements are 2) with an infinite vector. To obtain the answer, we call COMPUTE with query $\kappa = \mathtt{rs}\langle\mathcal{H}\rangle$. The bottom of the figure shows a search tree that computes the aggregands of the answers. The root represents the initial query of the first subgoal `r`, and the edges from the root correspond to the *overlapping* branches of answers returned by LOOKUP. Each such edge leads to a new node with some refined query of the second subgoal `s`, and the edges from that node again correspond to answers. (We elide the handling of the third subgoal, ⊗, as it is only queried on singleton sets and so no branching is possible.) Thus, each leaf corresponds to a non-ground rule query $\langle\tau_1, \tau_2\rangle$, which can be read off from the root-leaf path. The attendant (pre-)answer sets are central to further reasoning; for reasons of space we show only θ at each leaf and describe it by the variables used in the comprehension definition of ρ at the top. Each rule answer *may* contribute values to subsets of their heads, subject to *masking* (§3.6). To the right of the search tree is a trace (to be read top-to-bottom, and discussed in §4.1) of the execution of the algorithm of §4, which visits overrides before visiting defaults (i.e., sets are visited before their proper supersets). This order is why, for example, the top-most answer deriving from the pre-answer corresponding to $x = 0$, $y = 0$ contributes to $\{\mathtt{rs}\langle 0\rangle\}$ but the second does not. Densely dotted lines connect regions of the trace with their causative object in the search tree. Value contributions from leaves are denoted with ⊕= in this log; masks, derived from θ upon returns up the search tree (dotted arrows), are denoted # and continue to use the variable notation from the leaves. We elide the last set of masking operations. Gray entries do not change the algorithm's state; entries marked with daggers (†) rely on *obstruction*. The shape of the search tree is determined by LOOKUP's returns, namely $\{\{\mathtt{r}\langle 0,0\rangle\} \mapsto 4, \{\mathtt{r}\langle x,x\rangle \mid x \in \mathbb{Z}\} \mapsto 3, \mathtt{r}\langle\mathbb{Z},\mathbb{Z}\rangle \mapsto 2\}$ and $\{\{\mathtt{s}\langle 0\rangle\} \mapsto 6, \mathtt{s}\langle\mathbb{N}\rangle \mapsto 5\}$, BFs encoding the `r` and `s` items that have non-NULL weights.



	Mnemonic	Type
Ξ	Program Rule Indices	Set
$\mathcal{L}$	*L*ookup Results	BF
κ	Query *K*ey Set	$\subseteq \mathcal{I}$
r	*R*ule index	$\in \Xi$
η	*H*ead under consideration	$\in H$
$\vec{\tau}$	Rule query	$\in \mathfrak{R}$
$\mathfrak{R}$	*R*eplacements	Set of tuples of subsets of $\mathcal{I}$
$\mathfrak{H}$	Replacements for *H*ead	
$\mathfrak{M}$	*M*asking replacements	
H	Induced *H*eads	$\subseteq \mathcal{I}$
ω	*O*bstructing heads	
ρ	*R*ule groundings	μDyna
θ	Unevaluated, *th*ese keys	
ϵ	Answers are *E*valuated	
ψ	*S*urviving answers	
ν	Keys and *V*alues	Bag of kv-pairs
$\sqsupset$	*B*ag of result kv-pairs	
$\mathcal{C}$	*C*ontribution per rule	BF
$\mathcal{Y}$	*Y*ield	

Fig. 2: Plate notation for the various characters involved in inference with defaults. Plates (solid rectangles) indicate quantification of structure over the variable in their lower-left corner; domains of quantification point to their elements with dotted arrows. Solid arrows entering a plate are fanned-out to each instantiation; those exiting are fanned-in using the mechanism indicated. Double-tipped arrows indicate that the target is, in addition to being derived from the source, also *a subset* thereof. To reduce visual clutter, indices have been omitted (they can be uniquely recovered from context) and $\mathcal{L}$ and ρ have been repeated in the diagram, representing the same value at each occurrence. The $\vec{\tau}$ plate on the left derives the "induced heads" (see §3.4), which form the base of the contribution to the answer ($\mathcal{C}$) for each rule ($r \in \Xi$) given the query head (κ) and input BF ($\mathcal{L}$). The η plate derives the value for each point therein. Our assumptions on the program (1) (in §3.6) are seen to be the bridge between the system's fan-in (§3.5) and its masking post-processing (§3.6) of fan-out (§3.3).

vector via `rs(X) ⊕= r(X,Y) ⊗ s(Y)`. To find the answer, we invoke one step of backward reasoning by calling `Compute(rs⟨ℤ⟩)`, which, internally, uses `Lookup` to obtain values for subgoals' items. `Lookup` will continue to return a finite map with elements $\tau \mapsto v$, where $\tau \subseteq \mathcal{I}$ is a *set* of items and $v \in \mathcal{H}'$ is the value assigned to each $t \in \tau$. The centerpiece of this paper is that the τs in this map now *overlap*, forming the base of a BF that *encodes* the partition of old. Thus, the non-ground rule answers obtained (at the leaves of the tree) in answer to the non-ground rule queries (the root-leaf path through the tree) will overlap in that their *pre-answer* sets may have non-empty intersection. The *answer* sets may not intersect due to ascribing different values to the same keys.

3.2 An Intuitive View of Default Reasoning

Let us attempt to give an intuition before we dive into the formalities. We will use $\tilde{\cdot}$ over a symbol to mean that it has a more rigorous definition, given in the indicated section. As there are rather many derived quantities in flight, we proffer the map in Figure 2 for orientation as we proceed. Quantities involved will come to have unusually many indices on them; subscripts will be global in flavor while superscripts will be more local.

Every contribution to a head h can be *named* by a pair of a rule, r, and a *ground* rule query $\vec{t}$ thereof. If the responses from `Lookup` are partitions of its argument, as they were

in [4], then each such name will be associated with at most one value: within the search tree analogous to that of Figure 1, t_1 occurs on at most one edge from the root node, and, thereunder, t_2 again occurs on at most one edge, and so on. Whatever values are assigned to each t_i, the rule answers ϵ will be contained within the pre-answers θ: $\theta_r^{\vec{t}}$ is the *set of all possible answers for a given name* (for all possible heads, at that).

Now, however, we are assuming that the response is described by a BF, $\mathcal{L} \in \mathcal{I} \hat{\to} \mathcal{H}'$, and so there may be several branches containing t_1, each of which may contain several branches containing t_2, etc. We must *post-process* the results of the search so that only values from *enclosing branches* are considered. As there is only one encloser for t_i, this will again associate at most one value with the query $\vec{t}$, and the definition of BFs ensure that this will be the value *as if* the result had been partitioned. How are we to do this post-processing?

We can identify each leaf of the search tree with a non-ground rule *query* $\vec{\tau}$, e.g., $\langle \tau_1, \tau_2 \rangle$. Given a rule query $\vec{\tau}$ for r, the pre-answer set $\theta_r^{\vec{\tau}}$ is the *set of all possible answers* for all ground rule queries $\{\vec{t} \mid t_i \in \tau_i\}$. Thus, given *two* non-ground rule queries $\vec{\sigma}$ and $\vec{\tau}$, in which $\forall_i \sigma_i \subseteq \tau_i$, the set of possible answers *unique to* $\vec{\tau}$ is $\theta_r^{\vec{\tau}} \setminus \theta_r^{\vec{\sigma}}$, which is a superset of whatever *actual* rule answers are licensed by $\vec{\tau}$. If we collect *all* such $\vec{\sigma}$ from the search tree for each $\vec{\tau}$ into $\tilde{\mathfrak{M}}^{\vec{\tau}}$ (§3.6), then we see that $\theta_r^{\vec{\tau}} \setminus \bigcup_{\vec{\sigma} \in \tilde{\mathfrak{M}}^{\vec{\tau}}} \theta_r^{\vec{\sigma}}$ is the set of possible answers at the leaf identified with $\vec{\tau}$, and, given a value v_i for each τ_i, $\tilde{\psi}^{\vec{\tau}} = \epsilon_{r,\vec{v}}^{\vec{\tau}} \setminus \bigcup_{\vec{\sigma} \in \tilde{\mathfrak{M}}^{\vec{\tau}}} \theta_r^{\vec{\sigma}}$ (§3.6) is the set of *un-masked answers* from this leaf, just as if each τ_i had been a partition element. $\tilde{\nu}^{\vec{\tau}} = \tilde{\psi}^{\vec{\tau}} \downarrow_{\text{HR}}^{@}$ (§3.5) is then a bag of pairs of heads and associated aggregands from this leaf, which should be split by head and then aggregated, along with $\tilde{\nu}$ from other leaves.

Unfortunately, this $\tilde{\nu}^{\vec{\tau}}$ is difficult to compute and manipulate. Its computation clearly requires set subtraction. More worrying, even, it may overlap with other leaves' $\tilde{\nu}^{\vec{\tau}'}$ in complex ways reminiscent of why we needed to DISJOIN, but with yet more difficulty: there is no reason to believe that $\tilde{\nu}$ can be described as a bag product between a set of heads η and a bag of aggregands β. We require some kind of *uniformity* of $\tilde{\nu}$, and so must require something of its progenitor, $\tilde{\psi}$. Let us put this question on hold.

If our goal is to not just *interpret* BF encodings of item answers but to *produce* one as well, in response to an item query, there is the question of *what base* (i.e., what sets of heads) $\tilde{H}$ the output will have (§3.4). Assuming the kind of uniformity suggested above, we might speculate that $\epsilon \downarrow_{\text{HEAD}}$ for each choice of rule query $\vec{\tau}$ as likely candidates. It will turn out that we need $\theta \downarrow_{\text{HEAD}}$ as well in some cases, and, of course, we must ensure that the output base is $\cap$-closed. Producing a BF also allows us to revisit the uniformity requirement on $\tilde{\nu}$: *it is OK to be wrong* on a head h when generating the answer for base point $\eta \in H$, *so long as* η is not $\lceil \{h\} \rceil$. That is, so long as a smaller head set gets it right, our error will go un-noticed! Thus, we require uniformity of ψ only at $\eta \setminus \{\eta' \in H \mid \eta' \subseteq \eta\}$.

Let us now investigate details before turning our attention to algorithmic description.

3.3 Replacements

For any rule r, non-ground rule queries $\vec{\tau}$, pre-answers $\theta^{\vec{\tau}}$, and answers $\epsilon^{\vec{\tau}}$ are defined for *any* choice of sets of terms $\vec{\tau} = \langle \tau_1, \ldots, \tau_{n_r} \rangle$. However, given that LOOKUP acts as a BF, $\mathcal{L}$, rule queries formed from $T_{\mathcal{L}}$ have special significance, describing root-leaf paths in the search tree. We call these **$\mathcal{L}$-replacements** (or just replacements, when clear): $\mathfrak{R}_{r,\mathcal{L}} \stackrel{\text{def}}{=} \{\vec{\tau} = \langle \lceil \tau_i \rceil \rangle_{i \in \mathbb{N}_1^{n_r}} \mid \sigma_i \in T_{\mathcal{L}}, \tau_i = \sigma_i \cap \rho_r \downarrow_{\text{SG}.i.1}, \theta_r^{\vec{\tau}} \neq \varnothing\}$. We have, without semantic consequence, restricted to non-trivial rule queries formed from sets which are not

completely overridden within $\mathcal{L}$. Our idea to order $\vec{\tau}$ by $\subseteq$ on components is formalized as a partial order termed **specificity**: $\vec{\tau} \preceq \vec{\tau}' \Leftrightarrow \forall_{i \in \mathbb{N}_1^{n_r}} \tau_i \subseteq \tau_i'$, and $\vec{\tau} \prec \vec{\tau}' \Leftrightarrow \vec{\tau} \preceq \vec{\tau}' \wedge \exists_i \tau_i \neq \tau_i'$.

The BF $\mathcal{L}$, encoding a valuation function, specifies more than just its base. There is, thus, a natural choice not only for the non-ground rule *queries* but also for the values $\vec{v}$ determining a set of non-ground rule answers: the values assigned by $\mathcal{L}$ to each τ_i, and in particular $v_i = \mathcal{L}(\lceil \tau_i \rceil_{\mathcal{L}})$. Thus, we define the non-ground rule answer set for $\vec{\tau} \in \mathfrak{R}_{r,\mathcal{L}}$: $\epsilon_{r,\mathcal{L}}^{\vec{\tau}} \stackrel{\text{def}}{=} \epsilon_{r,\vec{v}}^{\vec{\tau}} = \theta_r^{\vec{\tau}}[v_1/\text{SG}.1.2]\cdots[v_{n_r}/\text{SG}.n_r.2]$. Recall that elements of these ϵ *are no longer necessarily all true*: overriding (more-specific) replacements cause elements to not accurately reflect the assignment of values to items as by the *interpretation* of $\mathcal{L}$.

We augment the definitions of θ and ϵ with another index, allowing concise notation for additional refinement by HEADs: $\theta_r^{\eta,\vec{\tau}} \stackrel{\text{def}}{=} \theta_r^{\vec{\tau}}[\eta/\text{HEAD}]$ and similarly for ϵ. Several set-theoretic properties of θ and ϵ are worth noting: ① $\theta_r^{\alpha,\vec{\tau}} = \theta_r^{\eta,\vec{\tau}}[\alpha/\text{HEAD}]$ and $\epsilon_{r,\mathcal{L}}^{\alpha,\vec{\tau}} = \epsilon_{r,\mathcal{L}}^{\eta,\vec{\tau}}[\alpha/\text{HEAD}]$ for any pair of sets $\alpha \subseteq \eta$. Thus, θ and ϵ are $\subseteq$-homomorphic in their head index. (This would not be true of ϵ if we had used $\lceil \cdot \rceil$ in its definition; instead, values are determined directly by $\vec{\tau}$.) ② $\theta_r^{\cdot}$ sends $\vec{\tau} \preceq \vec{\tau}'$ to $\theta_r^{\vec{\tau}} \subseteq \theta_r^{\vec{\tau}'}$, but no similar homomorphism holds for $\epsilon_{r,\mathcal{L}}^{\cdot}$ due to the evaluation of $\mathcal{L}$ in the latter.

3.4 Induced Heads

For a given r, valuation $\mathcal{L}$, and query head κ, a $\mathcal{L}$-replacement $\vec{\tau} \in \mathfrak{R}_{r,\mathcal{L}}$ gives rise to two **induced non-ground heads**, entries in the base of the BF encoding the results of `COMPUTE`-ing κ on the rule r. The head projection of both the (pre-)answer sets are potentially meaningful quantities, representing sets of items which behave similarly under the rule r. We define $H_{r,\kappa,\mathcal{L}}$ as the $\cap$-closure of the head projections of (pre)-answers for all replacements $\mathfrak{R}_{r,\mathcal{L}}$, i.e., of $\{\theta_r^{\kappa,\vec{\tau}}\downarrow_{\text{HEAD}}, \epsilon_{r,\mathcal{L}}^{\kappa,\vec{\tau}}\downarrow_{\text{HEAD}} \mid \vec{\tau} \in \mathfrak{R}_{r,\mathcal{L}}\}$.

A key insight is that the $\mathcal{L}$-replacements that might potentially influence an *entire* head α, as opposed to merely a subset thereof, are those $\mathcal{L}$-replacements whose *pre*-answers do not refine the head to a proper subset of α: $\mathfrak{H}_{r,\mathcal{L}}^{\alpha} \stackrel{\text{def}}{=} \{\vec{\tau} \in \mathfrak{R}_{r,\mathcal{L}} \mid \alpha = \theta_r^{\alpha,\vec{\tau}}\downarrow_{\text{HEAD}}\}$.

Example 3. Continuing our running example (§3.1) and considering $\kappa = \texttt{rs}\langle\mathcal{H}\rangle$, it is easy to check that all induced heads each come from both a θ and an ϵ, as refinement of subgoal values does not alter HEAD projections. All told, $H_{r,\kappa,\mathcal{L}} = \{\{\texttt{rs}\langle 0\rangle\}, \texttt{rs}\langle\mathbb{N}\rangle, \texttt{rs}\langle\mathbb{Z}\rangle\}$. ◊

Example 4. It may seem that θ-derived heads serve no purpose; after all, only subsets of ϵ are plausibly related to the values defined by the semantics of the language. However, consider a rule which constrains the *value* of a subgoal whose key covaries with the head, such as $\{\langle\langle \texttt{a}\langle x\rangle, 1\rangle, \langle\langle \texttt{b}\langle x\rangle, 4\rangle\rangle\rangle \mid x \in \mathcal{H}\}$, with $\mathcal{L} = \{\texttt{b}\langle\mathcal{H}\rangle \mapsto 4, \{\texttt{b}\langle 3\rangle\} \mapsto 2\}$. The θ-derived heads are $\{\texttt{a}\langle\mathcal{H}\rangle, \{\texttt{a}\langle 3\rangle\}\}$, while the ϵ-derived heads are $\{\texttt{a}\langle\mathcal{H}\rangle, \varnothing\}$. Thus, were we to solely consider ϵ-derived heads, we would fail to assert that $\{\texttt{a}\langle 3\rangle\} \mapsto \text{NULL}$. Put another way: a replacement may have answers while a more-specific replacement may not. ◊

Example 5. (Reproduced from [4]) The rule $\{\langle\langle \texttt{f}\langle a\rangle, 1\rangle, \langle\langle \texttt{a}\langle\rangle, a\rangle\rangle\rangle \mid a \in \mathcal{H}\}$ has covariance between the head and a subgoal *value*, contributing the value 1 to an item determined by *the value of* $\texttt{a}\langle\rangle$. While the previous example showed that θ-derived heads are essential, this example shows that ϵ-derived heads are as well, as the sole θ-derived head is $\texttt{f}\langle\mathcal{H}\rangle$. ◊

3.5 Collecting Contributions

For the moment, let us *assume the existence* of a bags of kv-pairs, $\nu_{r,\mathcal{L}}^{\eta,\vec{\tau}} \Subset \epsilon_{r,\mathcal{L}}^{\eta,\vec{\tau}}\downarrow_{\text{HR}}^{@}$, which capture the contribution of each $\vec{\tau} \in \mathfrak{H}_{r,\mathcal{L}}^{\eta}$ to η and omit any contributions from $\vec{\tau}$ overridden by some more-specific $\vec{\tau}' \in \mathfrak{H}$. So armed, it is easy to define the bag of contributions to η across all relevant replacements $\vec{\tau}$: $\sqsupset_{r,\mathcal{L}}^{\eta} \stackrel{\text{def}}{=} \biguplus_{\vec{\tau} \in \mathfrak{H}_{r,\mathcal{L}}^{\eta}} \nu_{r,\mathcal{L}}^{\eta,\vec{\tau}}$. These, then, can be combined in

a BF with base $H_{r,\kappa,\mathcal{L}}$: $\mathcal{C}_{r,\kappa,\mathcal{L}} \stackrel{\text{def}}{=} \{\eta \mapsto \{h \mapsto \{\!\{v@m \mid \langle h, v\rangle \Subset_{=m} \beth^{\eta}_{r,\mathcal{L}}\}\!\} \mid h \in \eta\} \mid \eta \in H_{r,\kappa,\mathcal{L}}\}$. The core of this definition simply partitions the bag $\beth^{\eta}_{r,\mathcal{L}}$ by key and collects together all of the values within a bag for each key in η. This operation is done for each induced head $\eta \in H$; by construction, the induced heads cover all possible results from the current rule. The last step, then, is to repeat this exercise for all rules $r \in \Xi$ and merge the results, which is easily enough done using the $\uplus$-join of all such functions: $\mathcal{Y}_{\Xi,\kappa,\mathcal{L}} \stackrel{\text{def}}{=} \bigwedge^{\uplus}_{r\in\Xi} \mathcal{C}_{r,\kappa,\mathcal{L}}$. (As $\uplus$ is associative-commutative, there is no concern of ordering here.) To ensure that the domain of $\mathcal{Y}$ is $\mathcal{I}$, we may further $\uplus$-join with $\{\mathcal{I} \mapsto \varnothing\}$ as well.

Unfortunately, while theoretically straightforward, this collection mechanism requires iteration over H and each η therein, since the νs are arbitrary. If we are to have hope of Compute-ing in finite time, we must impose structure on ν.

3.6 Masking and Obstruction

We now focus in on the middle of the system, on the task of deriving ν from ϵ. Recall that $\tilde{\psi}^{\vec{\tau}}$ was the set of rule answers from $\vec{\tau}$, having removed all pre-answers for all $\vec{\sigma} \prec \vec{\tau}$. We could formalize this by defining $\tilde{\mathfrak{M}}^{\vec{\tau}}_{r,\mathcal{L}} = \{\vec{\sigma} \in \mathfrak{R}_{r,\mathcal{L}} \mid \vec{\sigma} \prec \vec{\tau}\}$ and taking $\tilde{\psi}^{\eta,\vec{\tau}}_{r,\mathcal{L}} = \epsilon^{\eta,\vec{\tau}}_{r,\mathcal{L}} \setminus \bigcup_{\vec{\sigma}\in\tilde{\mathfrak{M}}^{\vec{\tau}}_{r,\mathcal{L}}} \theta^{\eta,\vec{\sigma}}_{r}$. Then $\tilde{\nu}^{\eta,\vec{\tau}}_{r,\mathcal{L}} = \tilde{\psi}^{\eta,\vec{\tau}}_{r,\mathcal{L}}\!\downarrow^{@}_{\text{HR}}$ is a bag of kv-pairs which are produced by $\vec{\tau}$ and survive masking, which should be aggregated with the results of other replacements and other rules. All told, given Ξ and $\mathcal{L}$, the value assigned to $h \in \mathcal{I}$ by the above, item-at-a-time mechanism is $\text{aggr}(h)(\biguplus_{r\in\Xi} \biguplus_{\vec{\tau}\in\mathfrak{R}_{r,\mathcal{L}}} (\tilde{\psi}^{\eta,\vec{\tau}}_{r,\mathcal{L}}\!\downarrow^{@}_{\text{RES}}))$.

Returning to set-at-a-time reasoning, and, in particular, to generating a BF, we can find two refinements of the system so far described; recall that, for a given output base point η, it is okay to be wrong when describing $h \in \eta$ so long as $\eta \neq \lceil\{h\}\rceil$. Specifically, this means that: ① There is no need to consider masking $\epsilon^{\vec{\tau}}$ by $\theta^{\vec{\sigma}}$, even if $\vec{\sigma} \prec \vec{\tau}$, if $\theta^{\vec{\sigma}}\!\downarrow_{\text{HEAD}} \subsetneq \epsilon^{\vec{\tau}}\!\downarrow_{\text{HEAD}}$. ② We can ignore uniformity requirements of ν on heads $\omega^{\eta}_{r,\mathcal{L}} \stackrel{\text{def}}{=} \bigcup\{\eta' \in H_{r,\kappa,\mathcal{L}} \mid \eta' \subsetneq \eta\}$. The first point means, concretely, that there is no need to consider something quite so large as $\tilde{\mathfrak{M}}$ above; when computing the base point η, we need only consider $\mathfrak{M}^{\alpha,\vec{\tau}}_{r,\mathcal{L}} \stackrel{\text{def}}{=} \{\vec{\sigma} \in \mathfrak{H}^{\alpha}_{r,\mathcal{L}} \mid \vec{\sigma} \prec \vec{\tau}\} = \{\vec{\sigma} \in \mathfrak{R}_{r,\mathcal{L}} \mid \vec{\sigma} \prec \vec{\tau}, \alpha = \theta^{\alpha,\vec{\sigma}}_{r}\!\downarrow_{\text{HEAD}}\}$, i.e., the set of more-specific $\vec{\tau}$ *which do not shrink the pre-answer head.*

Combining both of these observations, we can define (r, $\mathcal{L}$ and η are used three times in this definition, $\vec{\tau}$ twice): $\psi^{\eta,\vec{\tau}}_{r,\mathcal{L}} \stackrel{\text{def}}{=} (\epsilon^{\eta,\vec{\tau}}_{r,\mathcal{L}} \setminus \bigcup_{\vec{\sigma}\in\mathfrak{M}^{\eta,\vec{\tau}}_{r,\mathcal{L}}} \theta^{\vec{\sigma}}_{r})[(\mathcal{I} \setminus \omega^{\eta}_{r,\mathcal{L}})/\text{HEAD}]$. It is at this point that we must appeal to our oracular test of uniformity (§1.3); we require that every computed ψ be such that every head $h \in \psi\!\downarrow_{\text{HEAD}}$ be associated be associated with the same bag of values $\psi[\{h\}/\text{HEAD}]\!\downarrow^{@}_{\text{RES}}$. If this holds, then, at long last, we have a viable ν:

$$\nu^{\eta,\vec{\tau}}_{r,\mathcal{L}} \stackrel{\text{def}}{=} \{\!\{\langle h, v\rangle@m \mid h \in \eta, v \Subset_{=m} \texttt{AnswerFor}(\epsilon^{\eta,\vec{\tau}}_{r,\mathcal{L}}, \omega^{\eta}_{r,\mathcal{L}}, \textstyle\bigcup_{\vec{\sigma}\in\mathfrak{M}^{\eta,\vec{\tau}}_{r,\mathcal{L}}} \theta^{\vec{\sigma}}_{r})\}\!\}. \tag{1}$$

There are a few things to note about this definition. ① It can happen that $\eta' = \epsilon^{\eta,\vec{\tau}}_{r,\mathcal{L}}\!\downarrow_{\text{HEAD}} \subsetneq \eta$. When this happens, it implies that $\eta' \subseteq \omega^{\eta}_{r,\mathcal{L}}$, so $\psi^{\eta,\vec{\tau}}_{r,\mathcal{L}} = \varnothing$ and thus $\nu^{\eta,\vec{\tau}}_{r,\mathcal{L}}$. We can see this clearly in Example 5; for the purpose of the example, assume $\mathcal{L}(\{\mathtt{a}\langle\rangle\}) \mapsto 7$, so that $\epsilon^{\mathtt{f}\langle\mathcal{H}\rangle,\langle\{\mathtt{a}\langle\rangle\}\rangle}_{r,\mathcal{L}}\!\downarrow_{\text{HEAD}} = \epsilon^{\mathtt{f}\langle 7\rangle\},\langle\{\mathtt{a}\langle\rangle\}\rangle}_{r,\mathcal{L}}\!\downarrow_{\text{HEAD}} = \{\mathtt{f}\langle 7\rangle\}$. Thus, because the head covaries with a subgoal value, the default head $\mathtt{f}\langle\mathcal{H}\rangle$ has associated aggregands $\varnothing$. ② This is a constraint only on the (masked, un-obstructed) rule answers. Other than their role in masking, groundings inconsistent with $\mathcal{L}$ need not be considered. The set of inconsistent groundings may even be outside the collection of sets possessing description within an implementation (though the masks, derived from θ and therefore inclusive of both consistent and

inconsistent groundings, must still be within the system). ③ Given ground inputs and a range-restricted program, this algorithm behaves essentially as any other ground solver: ω and $\mathfrak{M}$ will always both be $\varnothing$ and $|\epsilon\downarrow^{@}_{\text{HR}}| = 1$, so AnswerFor is trivial. The need for AnswerFor to be defined thus replaces prior algorithms' (including [4]) appeal to range restriction [1].

A Closure of Computation At long last, we can get computational traction:

Lemma 1. *If (1) holds, $\mathcal{Y}_{\Xi,\kappa,\mathcal{L}}$ is pointwise-constant.*

Proof. The definition in (1) gives $\nu^{\eta,\vec{\tau}}_{r,\mathcal{L}}$ which are *bag products* between $U_1^{-1}\eta$ and values. That is, for any choice of r, $\mathcal{L}$, η, and $\vec{\tau}$ for which ν is defined, $\{h \mapsto \{\!\{v@m \mid \langle h, v\rangle \Subset_{=m} \nu^{\eta,\vec{\tau}}_{r,\mathcal{L}}\}\!\} \mid h \in \eta\}$ is a constant function. As $\beth^{\eta}_{r,\mathcal{L}}$ is a union (over $\vec{\tau}$) of ν bags, it follows that each $\{h \mapsto \{\!\{v@m \mid \langle h, v\rangle \Subset_{=m} \beth^{\eta}_{r,\mathcal{L}}\}\!\} \mid h \in \eta\}$ must, also, be a constant function. The definition of $\mathcal{C}_{r,\kappa,\mathcal{L}}$, which maps each η to a function of the above form, then implies that it is pointwise-constant. $\mathcal{Y}_{\Xi,\kappa,\mathcal{L}}$ is just a join of pointwise-constant BFs, and so must itself be pointwise-constant. □

Thus, to obtain a first practical algorithm, we restrict to pointwise-constant $\mathcal{L}$ and assume (1). The above lemma means, having started with pointwise-constant answers $\mathcal{L}$, that we will *always obtain pointwise-constant answers*, which we can merge into $\mathcal{L}$ as part of our solver's fixed-pointing execution, and not violate our precondition. While pointwise-constancy of $\mathcal{L}$ does not imply (1), it may, speculatively, nevertheless simplify the proof obligation to be met by static analyses.

4 Pseudocode

At last, we come to a procedural description of the system given above; pseudocode is shown in Listing 1. Compute simply wraps computeRule for each rule, using $\uplus$-join and a notion of *bulk aggregation* to turn bags of aggregands into the final answer, $\mathcal{Y}$.[7] computeRule forms nested loops using refineRuleSuffix, which now tracks its position in the rule's subgoals, i; a superset of ϵ, α_{v}; and a superset of θ, α_{k}. These supersets are exact at the leaves of the search tree. refineRuleSuffix calls applyV to contribute the answers it finds (at the leaves) and applyM whenever it returns to add to masks. Within refineRuleSuffix, $\mathcal{C}$ is a BF which stores both values and masks; the masks are removed upon return. capclose adds its argument to, and enforces the $\cap$-closure property of, $\mathcal{C}$.

Mask Estimates The algorithm, as is typical of search algorithms, only enumerates successes (i.e., replacements with non-empty ϵ), while the theory of §3 seems to depend on enumeration of both ϵ and θ for every $\mathcal{L}$-replacement. While the algorithm does derive masks from successes (the rightmost left-facing dotted arrows in Figure 1), it also derives masks from other α_{k} sets during its execution, when it unwinds the recursion of refineRuleSuffix. These α_{k} sets correspond to θs derived from a *prefix* of a rule query, $\vec{\tau} = \langle \tau_1, \ldots, \tau_{i-1}, \mathcal{H}, \ldots, \mathcal{H}\rangle$, which therefore *subsume* the θs of any possible query $\preceq \vec{\tau}$. While these are over-estimates of the sets tracked by the theory, they are not too large, in the sense of improperly masking later results, because they will only mask later results that share the prefix, which has just been exhaustively searched.

[7] To ensure that bulk aggregation is defined, we require that each rule r specify an aggregator consistent with all its possible heads: all items $\mathcal{I} \cap \rho_r\downarrow_{\text{HEAD}}$ must use this aggregator. This ensures that $\text{aggr}(\eta) \stackrel{\text{def}}{=} \text{selt}(\{\text{aggr}(h) \mid h \in \eta\})$ is well-defined when we use it. In practice, we will not compute all of $\mathcal{Y}$ and then reduce it; instead, we will exploit the required properties of aggregators to directly aggregate while computing $\mathcal{C}$ and then again to obtain results equivalent to aggregating across $\mathcal{Y}$.

```
1  def COMPUTE(κ ⊆ 𝓘) ∈ (κ ⇀̂ 𝓗′)
2    𝒴 ← {κ ↦ ∅}  % running union across all rules
3    foreach r ∈ Ξ do let 𝒞 = COMPUTERULE(r, κ) in 𝒴 ← 𝒴 ∧^⊎ 𝒞
4    return {τ ↦ aggr(τ)(β) | (τ ↦ β) ∈ 𝒴}  % bulk aggregate
5
6  def COMPUTERULE(r ∈ Ξ, κ ⊆ 𝓘) ∈ ((ρ_r↓HEAD ∩ κ) ⇀̂_fin 𝓗⁺)
7    𝒞 ← {κ ↦ ⟨∅, ∅⟩}  % initialize results and masks
8    let α = ρ_r[κ/HEAD] in REFINERULESUFFIX(α, α, 1)  % populate 𝒞
9    return {η ↦ x↓1 | (η ↦ x) ∈ 𝒞}  % values w/o masks
10
11   def REFINERULESUFFIX(α_k ⊆ ρ_r, α_v ⊆ α_k, i ∈ ℕ_1^{n_r+1}) ∈ ⟨⟩
12     if α_k = ∅ then return ⟨⟩  % incompatible keys
13     elif α_v ≠ ∅ then
14       if i = n_r + 1 then APPLYV(α_v)  % end of rule
15       else foreach (σ ↦ v) ∈ LOOKUP(α_v↓SG.i) toposorted by ⊆ ascending on σ
16               REFINERULESUFFIX(α_k[σ/SG.i.1], α_v[⟨σ, {v}⟩/SG.i], i + 1)
17     APPLYM(α_k)  % before returning, mask
18   def APPLYV(∅ ⫋ ϵ ⊆ ρ_r) ∈ ⟨⟩  % accumulate values
19     CAPCLOSE(ϵ↓HEAD)
20     foreach (η ↦ ⟨β, μ⟩) ∈ 𝒞 where η ⊆ ϵ↓HEAD
21       let {|v@m|} = ANSWERFOR(ϵ, ω, μ) where ω = ⋃{η′ ∈ dom(𝒞) | η′ ⊆ η}  % §1.3
22       𝒞(η) ← ⟨{|v@m|} ⊎ β, μ⟩
23   def APPLYM(∅ ⫋ α ⊆ ρ_r) ∈ ⟨⟩  % accumulate masks
24     CAPCLOSE(α↓HEAD)
25     foreach (η ↦ ⟨β, μ⟩) ∈ 𝒞 where η ⊆ α↓HEAD do 𝒞(η) ← ⟨β, α ∪ μ⟩
26   def CAPCLOSE(η ⊆ κ) ∈ ⟨⟩  % put η ∈ T_𝒞, ∩-closed
27     if η ∈ T_𝒞 then return
28     else foreach (τ ↦ _) ∈ 𝒞 let η′ = τ ∩ η in
29       if η′ ∉ (T_𝒞 ∪ {∅}) then
30         let A = {β ∈ T_𝒞 | η′ ⊆ β, ∀_{β′∈T_𝒞} β′ ⊆ β ⇒ η′ ⊈ β′}  % smallest T_𝒞s containing η′
31         𝒞(η′) ← ⟨⊎_{α∈A} 𝒞(α)↓1, ⋃_{α∈A} 𝒞(α)↓2⟩
32
33 LOOKUP ∈ Π_{α⊆⟨𝓗,𝓗⟩}((α↓1 ∩ 𝓘) ⇀̂_fin ({NULL} ∪ α↓2))
```

Listing 1: Default-based COMPUTE, ignoring any interaction with the agenda, and assuming ANSWERFOR from §1.3 as a primitive operation. REFINERULESUFFIX now tracks *two* subsets of ρ: the first, α_k, *ignores values* and gradually refines down to θ, while the second, α_v, is the more typical gradual refinement down to ϵ. During its operation, its nested loops each advance in topologically sorted order so that calls to COLLECT are made on monotonically non-$\preceq$-decreasing $\mathcal{L}$-replacements.

4.1 Discussion of the Example Trace

Figure 1 includes a trace of COMPUTERULE of Listing 1 running on the example of §3.1 on inputs described in the former's caption. The root node of the search tree corresponds to the outermost REFINERULESYNTAX call's LOOKUP of $\mathtt{r}\langle\mathcal{H}, \mathcal{H}\rangle$.

The first entry in the log (cyan box) arises from the discovery of aggregands, $\{|\otimes\langle 4, 6\rangle@1|\}$ for $\{\mathtt{rs}\langle 0\rangle\}$ from the replacement $\langle\{\mathtt{r}\langle 0, 0\rangle\}, \{\mathtt{s}\langle 0\rangle\}\rangle$. The second entry adds this replacement's pre-answer to the mask for $\{\mathtt{rs}\langle 0\rangle\}$, ensuring that any later-discovered values assigned to the *ground* rule query $\langle\mathtt{r}\langle 0, 0\rangle, \mathtt{s}\langle 0\rangle\rangle$ are discarded. Indeed, we see exactly this

case for the replacement $\langle\{\mathtt{r}\langle x, x\rangle \mid x \in \mathbb{Z}\}, \{\mathtt{s}\langle 0\rangle\}\rangle$, on the third line of the log (corresponding to the second leaf of the tree): the algorithm revisits this ground replacement, due to the unification of x and y (in ρ) by the diagonal $\mathtt{r}$, and must mask its incorrect contribution of $\otimes\langle 3, 6\rangle$. As the head here is still just $\{\mathtt{rs}\langle 0\rangle\}$, ϵ needs no further processing.

The third leaf of the tree generates contributions which claim themselves to be applicable to all $\mathtt{rs}\langle\mathbb{N}\rangle$ (red box). However, when the head is restricted to $\{\mathtt{rs}\langle 0\rangle\}$, we find ourselves *again* revisiting *only* the ground rule query $\langle\mathtt{r}\langle 0, 0\rangle, \mathtt{s}\langle 0\rangle\rangle$, and so, in fact, these contributions are destined to $\mathtt{rs}\langle\mathbb{N} \setminus \{0\}\rangle$, which is encoded by contributing to $\mathtt{rs}\langle\mathbb{N}\rangle$ but not $\{\mathtt{rs}\langle 0\rangle\}$. The algorithm then proceeds to mask off the diagonal entries in all heads $\{\mathtt{rs}\langle\mathbb{Z}\rangle, \mathtt{rs}\langle\mathbb{N}\rangle, \{\mathtt{rs}\langle 0\rangle\}\}$, though there is some redundancy in its efforts (gray log lines).

The fourth leaf of the tree generates contributions nominally for $\mathtt{rs}\langle\mathbb{Z}\rangle$ (yellow box), having constrained $y = 0$ in ρ. Again $\{\mathtt{rs}\langle 0\rangle\}$ masks this contribution (the 0 from the head and from y being sufficient to send the first subgoal to $\{\mathtt{r}\langle 0, 0\rangle\}$). The application of these contributions to the output heads $\mathtt{rs}\langle\mathbb{Z}\rangle$ and $\mathtt{rs}\langle\mathbb{N}\rangle$ rely on the set difference on heads (rather than groundings) in AnswerFor in a way that no prior entry has: it is *not the case* that the masked ϵs for the non-ground rule query $\langle\mathtt{r}\langle\mathbb{Z}, \mathbb{Z}\rangle, \{\mathtt{s}\langle 0\rangle\}\rangle$ and for these heads are uniform: every $\mathtt{rs}\langle\mathbb{N}\rangle$ *other than* $\{rs\}0$ has one contribution of $\otimes\langle 2, 6\rangle$. Once again, the diagonal strikes, but now *from the masks*: the $\mathtt{rs}\langle 0\rangle$ entry in these heads is associated with $\mathtt{r}\langle 0, 0\rangle$, which is already masked! However, $\{\mathtt{rs}\langle 0\rangle\}$ is already an answer in the output BF, and so we may *obstruct* it from AnswerFor's consideration. The same phenomenon recurs in the lavender box from the next and final answer in the search tree.

5 Related Work

5.1 Default Logics

Reiter [9] defines a logic for reasoning with defaults in the light of incomplete data. This logic can capture assertions like "most birds fly" and "emus are birds but do not fly" and will deduce, given a proof that Tweety is a bird, in the absence of a proof that Tweety is an emu, that Tweety can fly. As this is a boolean logic, it does not concern itself with multiplicities beyond "zero or non-zero", as truth is an absorbing element of disjunction. Jaeger [5] extends default logics to handle probabilistic reasoning. The BFs of this paper use defaults not as a compensation for incomplete knowledge—indeed, we must have complete knowledge of the contributions for items in order to assure that we obtain the correct answer—but rather as an encoding of structured sparsity without set subtraction.

5.2 Lifted Explanations for Problog

Problog [8] is a probabilistic extension of Prolog, assigning probabilistic weights to items. Like μDyna, Problog encounters the need to aggregate over unique outcomes,and could benefit from *counting* rather than *enumerating* outcomes. Nampally and Ramakrishnan [7] consider the construction of "lifted explanation graphs" which use the structure of the Problog program and existential quantification (over finite domains) to compactly summarize the support sets of results for rules. The cardinality of the supports are then extracted by solving recurrences on these structures. As noted in that work, these structures are generalizations of binary decision diagrams and likely can be further extended to encode multi-valued decision diagrams over infinite domains, in which case it may be possible to use them as an implementation of our set theory for (a subset of?) μDyna programs. (For both flavors of decision diagrams, see, e.g., [10].)

5.3 Lifted Inference in Statistical Relational Models

Van den Broeck [11] discusses the use of "weighted first-order model counting" as a building block for "lifted" exact (as well as approximate) probabilistic inference. If the sets of our system admit description in first-order logic, then our $\textsc{AnswerFor}(\cdot,\cdot,\cdot)$ oracle can likely be implemented using the same "knowledge compilation" mechanism of that work, which, amusingly, reduces the problem to *weighted circuit solving*.

Conclusion

We have presented a theory and algorithmic strategy of *default-based* set-at-a-time reasoning within a weighted logic program solver. We have lowered the set-theoretic requirements of the task from those of the algorithm of [4] by eliminating set subtraction in favor of the test for uniform cardinality in §1.3.

Acknowledgements

We are deeply indebted to the editorial proficiencies and intellects of Rachael Bennett, Dr. Thomas Filardo, Dr. Nora Zorich, Dr. Scott Smith, Tim Vieira, and Matthew Francis-Landau, who all read numerous early drafts of this document and kindly contributed countless structural, prosodic, and grammatical suggestions to the text.

References

[1] Stefan Brass. "Range Restriction for General Formulas". In: *Proceedings of the 23rd Workshop on (Constraint) Logic Programming*. 2009.

[2] Hubert Comon et al. *Tree Automata Techniques and Applications*. Online publication. 2007. URL: http://tata.gforge.inria.fr/.

[3] Nathaniel Wesley Filardo and Jason Eisner. "A Flexible Solver for Finite Arithmetic Circuits". In: *Technical Communications of the 28th ICLP*. Ed. by Agostino Dovier and Vítor Santos Costa. Vol. 17. Leibniz International Proceedings in Informatics (LIPIcs). 2012, pp. 425–438.

[4] Nathaniel Wesley Filardo and Jason Eisner. "Set-at-a-Time Solving in Weighted Logic Programs". In: In submission to ICLP'17; see http://www.cs.jhu.edu/~nwf/ilcp17-1.1.pdf. 2017.

[5] Manfred Jaeger. "A Logic for Default Reasoning About Probabilities". In: *Proceedings of the Tenth International Conference on Uncertainty in Artificial Intelligence*. UAI'94. Morgan Kaufmann Publishers Inc., 1994, pp. 352–359.

[6] Robert Kowalski. *Predicate Logic as Programming Language*. Memo 70. Department of Artificial Intelligence, Edinburgh University, 1974.

[7] Arun Nampally and C. R. Ramakrishnan. "Inference in Probabilistic Logic Programs using Lifted Explanations". In: (2016).

[8] L. De Raedt, A. Kimmig, and H. Toivonen. "ProbLog: A Probabilistic Prolog and its Application in Link Discovery". In: *Proc. of IJCAI*. 2007, pp. 2462–2467.

[9] R. Reiter. "A logic for default reasoning". In: *Artificial Intelligence* 13.1 (1980), pp. 81–132. DOI: 10.1016/0004-3702(80)90014-4.

[10] A. Srinivasan et al. "Algorithms for discrete function manipulation". In: *1990 IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*. 1990, pp. 92–95. DOI: 10.1109/ICCAD.1990.129849.

[11] Guy Van den Broeck. "Lifted Inference and Learning in Statistical Relational Models". PhD thesis. KU Leuven, 2013, pp. xx + 264.