The AAAI-17 Workshop on
Symbolic Inference and Optimization
WS-17-14

Embedding Tarskian Semantics in Vector Spaces

Taisuke Sato
AI research center AIST / National Institute of Informatics, Japan
email:satou.taisuke@aist.go.jp

Abstract

We propose a new linear algebraic approach to the computation of Tarskian semantics in logic. We embed a finite model $\mathbf{M}$ in first-order logic with N entities in N-dimensional Euclidean space $\mathbb{R}^N$ by mapping entities of $\mathbf{M}$ to N dimensional one-hot vectors and k-ary relations to order-k adjacency tensors (multi-way arrays). Second given a logical formula F in prenex normal form, we compile F into a set Σ_F of algebraic formulas in multi-linear algebra with a nonlinear operation. In this compilation, existential quantifiers are compiled into a specific type of tensors, e.g., identity matrices in the case of quantifying two occurrences of a variable. It is shown that a systematic evaluation of Σ_F in $\mathbb{R}^N$ gives the truth value, 1(true) or 0(false), of F in $\mathbf{M}$. Based on this framework, we also propose an unprecedented way of computing the least models defined by Datalog programs in linear spaces via matrix equations and empirically show its effectiveness compared to state-of-the-art approaches.

Introduction

In this paper, we propose a new linear algebraic approach to the computation of Tarskian semantics, i.e., the standard semantics for first-order logic. Tarskian semantics determines the truth value $[\![F]\!]$ of first-order formulas F in a model $\mathbf{M}$ based on a relational structure comprised of a non-empty domain $\mathscr{D}$ and relations over $\mathscr{D}$, using an interpretation associated with $\mathbf{M}$ that maps constants to entities in $\mathscr{D}$ and predicate symbols to the relations. $[\![F]\!]$ is step-by-step determined in $\mathbf{M}$ along the syntactic structure of F. What we propose here is to carry out this evaluation in another model isomorphically copied to the N-dimensional Euclidean space $\mathbb{R}^N$, when the first-order language $\mathscr{L}$ we use has only N constants and correspondingly $\mathscr{D}$ contains N entities.

More precisely, given a finite model $\mathbf{M}$, we first encode entities in $\mathscr{D}$ into vectors in $\mathbb{R}^N$ where N is the cardinality of $\mathscr{D}$ and also encode k-ary relations in $\mathbf{M}$ to order-k adjacency tensors in multi-linear algebra. Then to evaluate a logical formula F in prenex normal form, starting from atoms, we inductively derive a set Σ_F of algebraic formulas in multi-linear algebra augmented with a nonlinear operation. Evaluating Σ_F in $\mathbb{R}^N$ gives the truth value $[\![F]\!]$ in $\mathbf{M}$, that is, $[\![F]\!] = 1$ if $\mathbf{M} \models F$ else $[\![F]\!] = 0$.

Our proposal is motivated by recent work on logical inference concerning knowledge graphs(KGs). KGs are graphs encoding RDF triples of the form (subject : s, predicate : p, object : o) and can be considered as a set of ground atoms of the form $p(s,o)$. There are huge KGs available such as Freebase(Bollacker et al. 2008). The problem is that although they are good resources of the real world information and logically simple, they are huge, containing tens of millions of nodes and furthermore incomplete; there are lots of inconsistent data and also lots of missing data. To carry out various KG tasks such as computing the truth value (or more generally probability) of $p(s,o)$ while coping with the sheer amount of data and incompleteness, three major approaches are developed (Nickel et al. 2015); one that is based on probabilistic models, one that uses explicit features sampled from the graph and one that learns latent feature vectors from the graph. The last approach, latent feature approach, compiles entities and predicates in the domain into vectors and tensors(Kolda and Bader 2009) respectively and apply various linear algebraic operations, with dimension reduction, to compute the probability of $p(s,o)$.

In the development of these approaches, formulas beyond ground atoms are introduced and investigated such as existentially quantified conjunctions as queries and definite clauses as constraints on KGs (Grefenstette 2013; Rocktäschel, Singh, and Riedel 2015; Krompaß, Nickel, and Tresp 2014; Guu, Miller, and Liang 2015; Yang et al. 2015). However, from a logical point of view, their treatment was confined to propositional logic level and the evaluation of general first-order formulas is left untouched except for the work done by Grefenstette (Grefenstette 2013). Regrettably, while he succeeded in completely embedding the fragment of model theory, model theory of quantifier-free first-order logic, in tensor spaces, quantified formulas were excluded and had to be treated separately by another framework. Nested quantification was not allowed either. So how to evaluate arbitrarily quantified formulas in a vector space still remains open.

We solve this problem by introducing specific tensors for existential quantifiers together with a nonlinear operation. Our contribution is two-fold. First we introduce a single framework for the evaluation of quantified first-order formulas in a vector space, assuming the domain is finite, thus solving the remaining problem.

The second contribution is to present a concrete method, based on our framework, to compute the least model of Datalog programs in a vector space, which opens up a completely new way of evaluating recursive programs, though we have to skip details due to page limitations and only sketch experimental result.

At this point it would be beneficial to ask why evaluating logical formulas in a vector space is an interesting idea. First, there are a rich family of algebraic operations available in a vector space such as inner product, outer product, projection, PCA, SVD and so on that helps analyzing and manipulating vector data. Second, basically they are of polynomial time complexities, so we can expect efficient computation. Last but not least, approximation through various matrix and tensor decomposition potentially leads to logical inference for Web scale symbolic data.

We assume the reader is familiar with basics of logic and linear algebra including matrices and tensors(Kolda and Bader 2009; Cichocki et al. 2009).

Preliminaries

We first review some terminology in logic. We assume our first order language $\mathscr{L}$ contains N constants $\{e_1,\ldots,e_N\}$ and no function symbols.

A model $\mathbf{M} = (\mathscr{D},I)$ is a pair of domain, a nonempty set $\mathscr{D}$ and an interpretation I that maps constants e_i to elements (entities, individuals) $I(e_i) \in \mathscr{D}$ and k-ary predicate symbols r to k-ary relations $I(r) \subseteq \overbrace{\mathscr{D} \times \cdots \times \mathscr{D}}^{k \text{ times}}$. An assignment a is a mapping from variables x to an element $a(x) \in \mathscr{D}$. It provides a way of evaluating formulas containing free variables. Syntactically terms mean variables and/or constants and atomic formulas or atoms $r(t_1,\ldots,t_k)$ are comprised of a k-ary predicate symbol r and k terms $t_1,\ldots,t_k$ some of which may be variables. Formulas F in $\mathscr{L}$ are inductively constructed as usual from atoms using logical connectives (negation $\neg$, conjunction $\wedge$, disjunction $\vee$) and quantifiers ($\exists$,$\forall$).

Now we define free/bound occurrences of variables in F. When F is an atom, all variables in F occur free in F. When F is a negation $\neg F_1$, disjunction $F_1 \vee F_2$ or conjunction $F_1 \wedge F_2$, free variables in F_1 and those in F_2 both occur free in F and vice versa. When F is an existentially quantified formula $\exists x F_1$, free variables in F_1 except x occur free in F and vice versa. Variables in F that do not occur free in F are said to be bound. A formula is closed if it has no free variable whereas it is open if it has no quantification.

Given a model $\mathbf{M} = (\mathscr{D},I)$ and an assignment a, the denotation $[\![E]\!]_{I,a}$ in $\mathbf{M}$ of an expression E is inductively defined for terms t and formulas F as follows. $[\![t]\!]_{I,a} = I(t)$ if t is a constant, else $a(t)$. When r is a k-ary relation symbol, $[\![r(t_1,\ldots,t_k)]\!]_{I,a} = 1$(true) if $([\![t_1]\!]_{I,a},\ldots,[\![t_k]\!]_{I,a}) \in I(r)$, else 0(false). Let F_1,F_2 be formulas. For a negation $\neg F_1$, we define $[\![\neg F_1]\!]_{I,a} = 1 - [\![F_1]\!]_{I,a}$. $[\![F_1 \vee F_2]\!]_{I,a} = 1$ if $[\![F_1]\!]_{I,a} = 1$ or $[\![F_2]\!]_{I,a} = 1$, else 0. $[\![\exists x F_1]\!]_{I,a} = 1$ if there exists some $d \in \mathscr{D}$ such that $[\![F_1]\!]_{I,a[x \leftarrow d]} = 1$, else 0. Here $a[x \leftarrow d]$ is a new assignment that is the same as a except that it assigns d to the variable x. Finally $[\![A \wedge B]\!]_{I,a} = [\![\neg(\neg A \vee \neg B)]\!]_{I,a}$ and $[\![\forall x F_1]\!]_{I,a} = [\![\neg \exists x \neg F_1]\!]_{I,a}$. For any formula F, $[\![F]\!]_{I,a} \in \{1,0\}$ and when $[\![F]\!]_{I,a} = 1$, we write $\mathbf{M} \models_a F$. However when F is closed, since $[\![F]\!]_{I,a}$ does not depend on the assignment a, we just write $[\![F]\!]$ and $\mathbf{M} \models F$ if F is true in $\mathbf{M}$. For formulas F,G, we say F and G are equivalent and write $F \equiv G$ if $[\![F]\!]_{I,a} = [\![G]\!]_{I,a}$ for any model $\mathbf{M}$, any interpretation I and any assignment a. In what follows, I,a are omitted when they are clear from the context.

Then recall that a literal is an atom (positive literal) or its negation (negative literal). Suppose F is an open formula. F has an equivalent formula in disjunctive normal form (DNF) $A_1 \vee \cdots \vee A_k$ such that each disjunct A_i is a monomial, i.e., conjunction of literals. Dually F has an equivalent formula in conjunctive normal form (CNF) $A_1 \wedge \cdots \wedge A_k$ such that each conjunct A_i is a clause, i.e., disjunction of literals.

It is known that every formula has an equivalent formula in prenex normal form $Q_1 x_1 \cdots Q_m x_m G$ where $Q_1,\ldots,Q_m$ are quantifiers $\exists,\forall$ and G is open. So to evaluate the truth value $[\![F]\!]$ of a given F in $\mathbf{M}$, since G is equivalent to DNF or CNF, we have only to evaluate a prenex normal form $Q_1 x_1 \cdots Q_m x_m G$ where G is an open DNF or CNF.

Note the subformula $Q_m x_m G$. Since DNF and CNF are convertible to each other, it is equivalent to $\exists$-DNF or $\forall$-CNF where $\exists$-DNF is a class of formulas F such that F is a disjunction of disjuncts $\exists x(L_1 \wedge \cdots \wedge L_M)$ comprised of literals $L_1,\ldots,L_M$, whereas $\forall$-CNF is a class of formulas F such that F is a conjunction of conjuncts $\forall x(L_1 \vee \cdots \vee L_M)$. Suppose $F = Q_1 x_1 \cdots Q_m x_m G$ is given. We may assume, without loss of generality, that if $Q_m = \exists$, $Q_m x_m G$ is in $\exists$-DNF. Otherwise $Q_m x_m G$ is in $\forall$-CNF.

Now we turn to vector spaces. We consider tensors as multi-linear maps as mathematical objects and multi-way arrays as data structure depending on the context(Kolda and Bader 2009; Cichocki et al. 2009). Although tensors are a generalization of vectors and matrices, we specifically say vectors and matrices when their shape needs to be distinguished.

In what follows, scalars are denoted by lower case letters like a. Vectors mean column vectors and we denote them by boldface lower case letters like $\mathbf{a}$ and $\mathbf{a}$'s components by a_i. $\mathscr{D}' = \{\mathbf{e}_1,\ldots,\mathbf{e}_N\}$ stands for the standard basis of N-dimensional Euclidean space $\mathbb{R}^N$ where $\mathbf{e}_i = (0\cdots,1,\cdots,0)^T$ is a vector that has one at the i-th position and zeros elsewhere. Such vectors are called one-hot vectors. $\mathbf{1}$ is a vector of all ones. $(\mathbf{a} \bullet \mathbf{b}) = \mathbf{a}^T\mathbf{b}$ is the inner product of $\mathbf{a}$ and $\mathbf{b}$ whereas $\mathbf{a} \circ \mathbf{b} = \mathbf{a}\mathbf{b}^T$ is their outer product. Matrices are assumed to be square and written by boldface upper case letters like $\mathbf{A}$. In particular $\mathbf{I}$ is an identity matrix. $\mathbb{1} = \mathbf{1} \circ \mathbf{1}$ is a matrix of all ones. $\mathrm{tr}(\mathbf{A})$ stands for the trace of $\mathbf{A}$. Order-p tensors $\in \mathbb{R}^{\overbrace{N \times \cdots \times N}^{p}}$ are denoted by underlined matrices like $\underline{\mathbf{A}}$ or $\{a_{i_1,\ldots,i_p}\}$ $(1 \leq i_1,\ldots,i_p \leq N)$. $\underline{\mathbf{A}}$'s component $a_{i_1,\ldots,i_p}$ is also written as $(\underline{\mathbf{A}})_{i_1,\ldots,i_p}$. Let $\underline{\mathbf{A}} = \{a_{i_1,\ldots,i_p}\}$ and $\underline{\mathbf{B}} = \{b_{k_1,\ldots,k_q}\}$ be tensors. The mode-(n,m) contracted product $\underline{\mathbf{A}} \times_{n,m} \underline{\mathbf{B}}$ of $\underline{\mathbf{A}}$ and $\underline{\mathbf{B}}$ is defined by $(\underline{\mathbf{A}} \times_{n,m} \underline{\mathbf{B}})_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_p,k_1,\ldots,k_{m-1},k_{m+1},\ldots,k_q}$

$= \sum_j a_{i_1,\ldots,i_{n-1},j,\ldots,i_p} b_{k_1,\ldots,k_{m-1},j,\ldots,k_q}$ with the convention that the association is to the left, i.e., $\underline{\mathbf{A}} \times_{n,m} \underline{\mathbf{B}} \times_{p,q} \underline{\mathbf{C}} = (\underline{\mathbf{A}} \times_{n,m} \underline{\mathbf{B}}) \times_{p,q} \underline{\mathbf{C}}$. So $\underline{\mathbf{A}} \bullet_n \mathbf{u}$, the contracted product of $\underline{\mathbf{A}}$ and vector $\mathbf{u}$, which is computed by $(\underline{\mathbf{A}} \bullet_n \mathbf{u})_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_p} = \sum_j a_{i_1,\ldots,i_{n-1},j,\ldots,i_p} u_j$ is equal to $\underline{\mathbf{A}} \times_{n,1} \mathbf{u}$ and the usual n-mode product $\underline{\mathbf{A}} \times_n \mathbf{U}$ of $\underline{\mathbf{A}}$ and matrix $\mathbf{U}$ is equal to $\underline{\mathbf{A}} \times_{n,2} \mathbf{U}$.

Tensors can be constructed by outer products; $(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})_{ijk} = a_i b_j c_k$ is an order-3 tensor and $(\underline{\mathbf{A}} \circ \mathbf{B})_{i_1,\ldots,i_p,k_1,\ldots,k_q} = a_{i_1,\ldots,i_p} b_{k_1,\ldots,k_q}$ is the outer product of $\underline{\mathbf{A}} = \{a_{i_1,\ldots,i_p}\}$ and $\underline{\mathbf{B}} = \{b_{k_1,\ldots,k_q}\}$.

Embedding a model into a vector space

Let $\{e_1,\ldots,e_N\}$ be the set of constants in $\mathscr{L}$ and $\mathbf{M} = (\mathscr{D}, I)$ a model where $\mathscr{D} = \{e_1,\ldots,e_N\}$ (we here identify $I(e_i)$ and $I(r)$ in $\mathbf{M}$ with e_i and r respectively to avoid notational complications). We show how to replace the evaluation $[\![F]\!]$ of a prenex formula F in $\mathbf{M}$ with the evaluation of Σ_F, a set of tensors compiled from F, in N-dimensional Euclidean space $\mathbb{R}^N$. The compilation of F into Σ_F starts from literals then proceeds to compound formulas and quantifications.

Entities, literals, logical connectives and existential quantifier

First we isomorphically map $\mathbf{M}$ to a model $\mathbf{M}'$ in $\mathbb{R}^N$. We map entities $e_i \in \mathscr{D}$ to one-hot vectors $\mathbf{e}_i$. So $\mathscr{D}$ is mapped to $\mathscr{D}' = \{\mathbf{e}_1,\ldots,\mathbf{e}_N\}$, the basis of $\mathbb{R}^N$. We next map a k-ary relation r in $\mathbf{M}$ to a k-ary relation r' over $\mathscr{D}'$ which is computed by an order-k tensor $\underline{\mathbf{R}} = \{r_{i_1,\ldots,i_k}\}$. $\underline{\mathbf{R}}$ is designed to retain the truth value $[\![r(e_{i_1},\ldots,e_{i_k})]\!]$ in $\mathbf{M}$ and given by the equation

$$\begin{aligned} & [\![r(e_{i_1},\ldots,e_{i_k})]\!] \\ &= \underline{\mathbf{R}}(\mathbf{e}_{i_1},\ldots,\mathbf{e}_{i_k}) \quad \text{as multi-linear map} \\ &= \underline{\mathbf{R}} \times_{1,1} \mathbf{e}_{i_1} \times_{1,2} \cdots \times_{1,i_k} \mathbf{e}_{i_k} \\ &= r_{i_1,\ldots,i_k} \in \{1,0\} \ \ (\forall i_1,\ldots,i_k \in \{1,\ldots,N\}). \quad (1) \end{aligned}$$

We identity r' with $\underline{\mathbf{R}}$ for simplicity and say $\underline{\mathbf{R}}$ encodes the $\mathbf{M}$-relation r. Let $\mathbf{M}'$ be a model $(\mathscr{D}', I')$ in $\mathbb{R}^N$ such that I' interprets entities by $I'(e_i) = \mathbf{e}_i$ $(1 \le i \le N)$ and relations r by $I'(r) = \underline{\mathbf{R}}$ introduced by (1).

We next inductively define the evaluation $[\![F]\!]'_{I',a'}$ of a formula F in $\mathbf{M}'$. Let a be an assignment in $\mathbf{M}$ and a' the corresponding assignment in $\mathbf{M}'$, i.e., $a(x) = e_i$ if-and-only-if $a'(x) = \mathbf{e}_i$. For a ground atom $r(e_{i_1},\ldots,e_{i_k})$, we define

$$[\![r(e_{i_1},\ldots,e_{i_k})]\!]' = \underline{\mathbf{R}}(\mathbf{e}_{i_1},\ldots,\mathbf{e}_{i_k}) \ (\forall i_1,\ldots,i_k \in \{1,\ldots,N\}) \tag{2}$$

where $\underline{\mathbf{R}} = \{r_{i_1,\ldots,i_k}\}$ is the tensor that encodes the $\mathbf{M}$-relation r in $\mathbf{M}$ (see (1)). By definition $[\![F]\!]'_{I',a'} = [\![F]\!]_{I,a}$ holds for any atom F. Negative literals are evaluated specifically in $\mathbf{M}'$ using tensors $\neg\underline{\mathbf{R}}$ introduced by

$$\begin{aligned} [\![\neg r(e_{i_1},\ldots,e_{i_k})]\!]' &= \neg\underline{\mathbf{R}}(\mathbf{e}_{i_1},\ldots,\mathbf{e}_{i_k}) \\ &= 1 - r_{i_1,\ldots,i_k} \\ \text{where } \neg\underline{\mathbf{R}} &\stackrel{\text{def}}{=} \overbrace{\mathbf{1} \circ \cdots \circ \mathbf{1}}^{k} - \underline{\mathbf{R}} \end{aligned} \tag{3}$$

We say $\neg\underline{\mathbf{R}}$ encodes an $\mathbf{M}$-relation $\neg r$[1]. Negation other than negative literals and conjunction and disjunction are evaluated in $\mathbf{M}'$ as follows.

$$\begin{aligned} [\![\neg F]\!]'_{I',a'} &= 1 - [\![F]\!]'_{I',a'} & (4) \\ [\![F_1 \wedge \cdots \wedge F_h]\!]'_{I',a'} &= [\![F_1]\!]' \cdots [\![F_h]\!]'_{I',a'} & (5) \\ [\![F_1 \vee \cdots \vee F_h]\!]'_{I',a'} &= \min_1([\![F_1]\!]'_{I',a'} + \cdots + [\![F_h]\!]'_{I',a'}) & (6) \\ [\![\exists y F]\!]'_{I',a'} &= \min_1(\sum_{i=1}^{N} [\![F_{y \leftarrow e_i}]\!]'_{I',a'}) & (7) \end{aligned}$$

Here $\min_1(x) = \min(x,1) = x$ if $x < 1$ else 1 and when applied to tensors, it means componentwise application. $F_{y \leftarrow e_i}$ is a formula obtained from F by replacing every free occurrence of y in F with e_i. Universal quantification is treated as $\forall x F = \neg\exists x \neg F$.

It is straightforward to check that the evaluation $[\![F]\!]'_{I',a'}$ of a formula F in $\mathbf{M}'$ by (2), (3), (4), (5), (6) and (7) coincides with $[\![F]\!]_{I,a}$ in $\mathbf{M}$. However, although this evaluation is carried out in a vector space, i.e. $\mathbb{R}^N$, it is based on the reduction of quantification to the ground level as (7) indicates and contains a lot of redundancy. We next show how to do the same thing without grounding quantifications.

$\exists$-DNF and $\forall$-CNF as tensors

Now we come to the crucial point of our proposal, evaluating quantified formulas without grounding. Consider a prenex formula $F = Q_1 x_1 \cdots Q_m x_m G$. For the moment we assume the inner most quantified subformula $Q_m x_m G$ is in $\exists$-DNF.

Let $\exists y (L_1 \wedge \cdots \wedge L_M)$ be an arbitrary disjunct of $Q_m x_m G$ where $L_1,\ldots,L_M$ are literals. We further assume the *variable condition* that y occurs once in each literal $L_m = r^\circ_m(x^m_1,\ldots,x^m_{N_m})$ $(1 \le m \le M)$. Here $r^\circ_m = r_m$ if L_m is a positive literal else $r^\circ_m = \neg r_m$. r°_m in $\mathbf{M}$ is called the $\mathbf{M}$-relation contained in L_m. Let $\underline{\mathbf{R}}^\circ_m (1 \le m \le M)$ be a tensor encoding the $\mathbf{M}$-relation r°_m defined respectively by (1) or (3). So $\underline{\mathbf{R}}^\circ_m(\mathbf{x}^m_1,\ldots,\mathbf{x}^m_{N_m}) = [\![L_m[x^m_1,\ldots,x^m_{N_m}]]\!]$ holds where x^m_i $(1 \le i \le N_m)$ range over the domain of constants $\mathscr{D} = \{e_1,\ldots,e_N\}$ while $\mathbf{x}^m_i$ correspondingly range over the domain of the standard basis $\mathscr{D}' = \{\mathbf{e}_1,\ldots,\mathbf{e}_N\}$. The notation $L_m[x^m_1,\ldots,x^m_{N_m}]$ emphasizes that $x^m_1,\ldots,x^m_{N_m}$ occur in L_m.

Suppose $L_m = r^\circ_m(x^m_1,\ldots,x^m_{N_m})$ $(1 \le m \le M)$ has y as the j_m-th argument. Remove y, the j_m-th argument, from $(x^m_1,\ldots,x^m_{N_m})$. Write the remaining arguments (with order preserved) collectively as $x^{(m)}_{-y}$ and consider $[\![L_m]\!] = [\![L_m[x^{(m)}_{-y}]]\!] = \underline{\mathbf{R}}^\circ_m(\mathbf{x}^{(m)}_{-y})$ as a function of $x^{(m)}_{-y}$ or a function of the corresponding arguments $\mathbf{x}^{(m)}_{-y}$ over $\mathscr{D}'^{N_m-1}$ parameterized with y. Then consider $[\![\exists y(L_1 \wedge \cdots \wedge L_M)]\!] \in \{1,0\}$ as a relation combined with arguments (free variables, possibly duplicate) $(x^{(1)}_{-y},\ldots,x^{(M)}_{-y})$ over $\mathscr{D}^{\Sigma_m(N_m-1)}$, or equivalently, a function applied to $(\mathbf{x}^{(1)}_{-y},\ldots,\mathbf{x}^{(M)}_{-y})$ over $\mathscr{D}'^{\Sigma_m(N_m-1)}$. We seek a tensor $\underline{\mathbf{R}}^{\text{new}}$ that encodes this function, i.e., $\underline{\mathbf{R}}^{\text{new}}$ such

[1] $\mathbf{1} \circ \cdots \circ \mathbf{1}$ is an order-k tensor. $\mathbf{1} \circ \cdots \circ \mathbf{1}(\mathbf{e}_{i_1},\ldots,\mathbf{e}_{i_k}) = (\mathbf{1} \bullet \mathbf{e}_{i_1}) \cdots (\mathbf{1} \bullet \mathbf{e}_{i_k}) = 1$ holds.

that $[\![\exists y(L_1 \wedge \cdots \wedge L_M)]\!] = \underline{\mathbf{R}}^{\text{new}}(\mathbf{x}_{-y}^{(1)}, \ldots, \mathbf{x}_{-y}^{(M)})$ holds. Look at

$$\begin{aligned}
&[\![\exists y(L_1 \wedge \cdots \wedge L_M)]\!] \\
&= [\![\exists y(r_1^\circ(x^{(1)}) \wedge \cdots \wedge r_M^\circ(x^{(M)}))]\!] \\
&= \min_1\Big(\sum_{k=1}^{N} \prod_{m=1}^{M} \underline{\mathbf{R}}_m^\circ(\mathbf{x}^{(m)})_{y\leftarrow \mathbf{e}_k}\Big) \\
&= \min_1\Big(\sum_{k=1}^{N} \prod_{m=1}^{M} (\underline{\mathbf{R}}_m^\circ \bullet_{j_m} \mathbf{e}_k)(\mathbf{x}_{-y}^{(m)})\Big) \\
&= \min_1\Big(\sum_{k=1}^{N} \Big\{\big((\underline{\mathbf{R}}_1^\circ \bullet_{j_1} \mathbf{e}_k) \circ \cdots \circ (\underline{\mathbf{R}}_M^\circ \bullet_{j_M} \mathbf{e}_k)\big)(\mathbf{x}_{-y}^{(1)}, \ldots, \mathbf{x}_{-y}^{(M)})\Big\}\Big) \\
&= \min_1\Big(\sum_{k=1}^{N} \big((\overbrace{\mathbf{e}_k \circ \cdots \circ \mathbf{e}_k}^{M}) \times_{1,j_1} \underline{\mathbf{R}}_1^\circ \times_{1,j_2} \cdots \times_{1,j_M} \underline{\mathbf{R}}_M^\circ\big)\Big) \\
&\quad (\mathbf{x}_{-y}^{(1)}, \ldots, \mathbf{x}_{-y}^{(M)}) \\
&= \min_1\big(\underline{\mathbf{Q}}^{\exists,M} \times_{1,j_1} \underline{\mathbf{R}}_1^\circ \times_{1,j_2} \cdots \times_{1,j_M} \underline{\mathbf{R}}_M^\circ\big)(\mathbf{x}_{-y}^{(1)}, \ldots, \mathbf{x}_{-y}^{(M)})
\end{aligned}$$

Here

$$\underline{\mathbf{Q}}^{\exists,M} \stackrel{\text{def}}{=} \sum_{k=1}^{N} \overbrace{\mathbf{e}_k \circ \cdots \circ \mathbf{e}_k}^{M} \tag{8}$$

is a tensor representing the existential quantifier $\exists y$.

Summing up, the $\mathbf{M}$-relation extracted from $\exists y(L_1 \wedge \cdots \wedge L_M)$, which solely depends on the free variables in it, is encoded by

$$\underline{\mathbf{R}}^{\text{new}} = \min_1\big(\underline{\mathbf{Q}}^{\exists,M} \times_{1,j_1} \underline{\mathbf{R}}_1^\circ \times_{1,j_2} \cdots \times_{1,j_M} \underline{\mathbf{R}}_M^\circ\big) \tag{9}$$

where $\underline{\mathbf{R}}_m^\circ$ encodes the $\mathbf{M}$-relation contained in L_m ($1 \leq m \leq M$) and the existential quantifier $\exists y$ that quantifies M free occurrences of y in $L_1 \wedge \cdots \wedge L_M$ is encoded by an order-M tensor $\underline{\mathbf{Q}}^{\exists,M}$ introduced by (8). We call the equation (9) a definition for $\underline{\mathbf{R}}^{\text{new}}$.

Similarly, if $Q_m x_m G$ is a $\forall$-CNF formula $\forall y(L_1 \vee \cdots \vee L_M)$, the relation in $\mathbf{M}$ extracted from $\forall y(L_1 \vee \cdots \vee L_M)$ is encoded by

$$\underline{\mathbf{R}}^{\text{new}} = \overbrace{\mathbf{1} \circ \cdots \circ \mathbf{1}}^{\Sigma_m N_m - 1} - \min_1\big(\underline{\mathbf{Q}}^{\exists,M} \times_{1,j_1} \underline{\mathbf{R}}_1^\circ \times_{1,j_2} \cdots \times_{1,j_M} \underline{\mathbf{R}}_M^\circ\big) \tag{10}$$

where $\underline{\mathbf{R}}_m^\circ$ encodes the $\mathbf{M}$-relation contained in $\neg L_m$ ($1 \leq m \leq M$) (details omitted).

Compiling prenex formulas

We now compile a prenex formula $F = Q_1 x_1 \cdots Q_m x_m G$, using (9) and (10), into an associated set Σ_F of tensor definitions which computes $[\![F]\!]$ without grounding. However there is one problem to solve before compilation; (9), for example, is derived from $\exists y(L_1 \wedge \cdots \wedge L_M)$ under the the variable condition. When this condition is violated, we need to somehow recover it.

There are two cases where the condition is violated. The first case is that some atom $r_m(x^{(m)})$ in G has duplicate occurrences of variables in the arguments $x^{(m)}$. In this case, let $\mathbf{R}_m$ be a tensor encoding the $\mathbf{M}$-relation r_m which is given by (1). Let $r_m^{\text{new}}(x'^{(m)})$ be a new atom defined by $r_m^{\text{new}}(x'^{(m)}) \Leftrightarrow r_m(x^{(m)})$ where $x'^{(m)}$ is an enumeration of $x^{(m)}$ without duplication. It is apparent that a new relation r_m^{new} stands for in $\mathbf{M}$ is encoded by a tensor $\underline{\mathbf{R}}_m^{\text{new}}$ such that $\underline{\mathbf{R}}_m^{\text{new}}(\mathbf{x}'^{(m)}) = \underline{\mathbf{R}}_m(\mathbf{x}^{(m)})$ where variables $\mathbf{x}'^{(m)}$ and $\mathbf{x}^{(m)}$ run over $\mathscr{D}'$. We replace every atom in G that violates the variable condition with a new atom $r_m^{\text{new}}(x'^{(m)})$ described above so that $\underline{\mathbf{R}}_m^{\text{new}}$ encodes the new $\mathbf{M}$-relation r_m^{new}. Let the result be G^* and consider $F^* = Q_1 x_1 \cdots Q_m x_m G^*$. Obviously when evaluated in $\mathbf{M}'$, F^* and F give the same result, i.e., $[\![F^*]\!]' = [\![F]\!]'(= [\![F]\!])$ holds. So in the first case, we compile F^* instead of F.

The second case is that, for example, some L_is in $D = \exists y(L_1 \wedge \cdots \wedge L_M)$ have no occurrence of y. In this case, we

---

Input: A model $\mathbf{M}$ for a first-order language $\mathscr{L}$ with finitely many constants and a first-order closed formula $F = Q_1 x_1 \cdots Q_m x_m G$ in $\mathscr{L}$ in prenex normal form such that no atom has duplicate variables and G is an open DNF or CNF

Procedure:

[Step 1] Set $\Sigma_F = \{\}$, $G_m = G$ and $F_m = Q_m x_m G_m$;

[Step 2]
- **For** i = m **down-to** 1 **Do**
  - Write $F_i = Q_i x_i G_i$;
  - If $Q_i = \forall$ then goto **[Step 2-B]**;
  - **[Step 2-A]**
  - Convert $Q_i x_i G_i$ to $\exists$-DNF G_i^*;
  - **For each** disjunct D **in** G_i^* **Do**
    - Write $D = \exists y(L_1 \wedge \cdots \wedge L_M) \wedge D'$ where y occurs once in each L_i ($1 \leq i \leq M$) and has no occurrence in D';
    - Let x_{free} be an enumeration without duplication of free variables in $D'' = \exists y(L_1 \wedge \cdots \wedge L_M)$;
    - Define a new atom by $r^{\text{new}}(x_{\text{free}}) \Leftrightarrow D''$;
    - Replace D in G_i^* with $r^{\text{new}}(x_{\text{free}}) \wedge D'$;
    - Introduce a new tensor $\underline{\mathbf{R}}^{\text{new}}$ by (9) encoding the new relation r^{new} in $\mathbf{M}$;
    - Add to Σ_F the tensor definition for $\underline{\mathbf{R}}^{\text{new}}$;
    - **endDo**
  - Set $F_{i-1} = Q_{i-1} x_{i-1} G_i^*$;
  - **[Step 2-B]**
  - Convert $Q_i x_i G_i$ to $\forall$-CNF C_i^*;
    - (the rest is dual to **[Step 2-A]** and omitted)
- **endDo**

[Step 3] If $F_0 = r_1 \wedge \cdots \wedge r_h$ then put $F^{\text{tensor}} = r_1 \cdots r_h$;
- Else $F_0 = r_1 \vee \cdots \vee r_h$ and put $F^{\text{tensor}} = \min_1(r_1 + \cdots + r_h)$;
  - (r_i's are atoms with no arguments, equated with true or false, and hence with $\{1, 0\}$)

Output: F^{tensor} with a set Σ_F of tensor definitions. Σ_F Σ_F encodes new $\mathbf{M}$-relations appearing in S and F_0 gives $[\![F]\!]$ in $\mathbf{M}$.

Figure 1: Compilation procedure of prenex formulas

just shrink the scope of $\exists y$ and rewrite D like $D = \exists y(L_1 \wedge \cdots \wedge L_h) \wedge L_{h+1} \wedge \cdots \wedge L_M$.

Taking these modifications into account, we summarize our compilation procedure in Figure 1. When a model $\mathbf{M}$ and a closed prenex formula F are given, the compilation procedure returns an algebraic formula F^{tensor} and a set Σ_F of tensor definitions. Evaluating F^{tensor} using Σ_F gives $[\![F]\!]$, the truth value of F in $\mathbf{M}$.

A compilation example

Let $F_{ABCD} = \forall x \exists y((A(x,y) \wedge B(x)) \vee (C(x,y) \wedge D(y))$. We compile F_{ABCD} into a set $\Sigma_{F_{ABCD}}$ of tensor definitions along the compilation procedure in Figure 1. Let $\underline{\mathbf{A}}$, $\underline{\mathbf{B}}$, $\underline{\mathbf{C}}$ and $\underline{\mathbf{D}}$ respectively be tensors encoding $\mathbf{M}$-relations A, B, C and D.

Set $\Sigma_{F_{ABCD}} = \{\}$. First we convert F_{ABCD}'s innermost subformula F_2 into $\exists$-DNF:

$$\begin{aligned} F_2 &= \exists y((A(x,y) \wedge B(x)) \vee (C(x,y) \wedge D(y))) \\ &= \exists y(A(x,y) \wedge B(x)) \vee \exists y(C(x,y) \wedge D(y)) \\ &= (\exists y A(x,y) \wedge B(x)) \vee \exists y(C(x,y) \wedge D(y)). \end{aligned}$$

Next we introduce new atoms and rewrite F_2 to G_2^*:

$$\begin{aligned} r_A^{\text{new}}(x) &\Leftrightarrow \exists y A(x,y) \\ r_{CD}^{\text{new}}(x) &\Leftrightarrow \exists y(C(x,y) \wedge D(y)) \\ G_2^* &= (r_A^{\text{new}}(x) \wedge B(x)) \vee r_{CD}^{\text{new}}(x). \end{aligned}$$

Correspondingly to these new atoms, we construct tensors below which encode the corresponding relations in $\mathbf{M}$ and add them to Σ_F:

$$\underline{\mathbf{R}}_A^{\text{new}} = \min_1(\underline{\mathbf{Q}}^{\exists,1} \times_{1,2} \underline{\mathbf{A}}) \quad (11)$$

$$\underline{\mathbf{R}}_{CD}^{\text{new}} = \min_1(\underline{\mathbf{Q}}^{\exists,2} \times_{1,2} \underline{\mathbf{C}} \times_{1,1} \underline{\mathbf{D}}). \quad (12)$$

We put $F_1 = \forall x F_2 = \forall x G_2^*$ and continue compilation. We convert F_1 to $\forall$-CNF:

$$\begin{aligned} F_1 &= \forall x((r_A^{\text{new}}(x) \wedge B(x)) \vee r_{CD}^{\text{new}}(x)) \\ &= \forall x(r_A^{\text{new}}(x) \vee r_{CD}^{\text{new}}(x)) \wedge \forall x(B(x) \vee r_{CD}^{\text{new}}(x)). \end{aligned}$$

We introduce new atoms and rewrite F_1 to G_1^*:

$$\begin{aligned} r_{ACD}^{\text{new}} &\Leftrightarrow \forall x\, r_A^{\text{new}}(x) \vee r_{CD}^{\text{new}}(x) \\ r_{BCD}^{\text{new}} &\Leftrightarrow \forall x\, r_B(x) \vee r_{CD}^{\text{new}}(x) \\ G_1^* &= r_{ACD}^{\text{new}} \wedge r_{BCD}^{\text{new}}. \end{aligned}$$

We construct tensors (scalars) for r_{ACD}^{new} and r_{BCD}^{new}:

$$\underline{\mathbf{R}}_{ACD}^{\text{new}} = 1 - \min_1(\underline{\mathbf{Q}}^{\exists,2} \times_{1,1} \neg \underline{\mathbf{R}}_A^{\text{new}} \times_{1,1} \neg \underline{\mathbf{R}}_{CD}^{\text{new}}) \quad (13)$$

$$\underline{\mathbf{R}}_{BCD}^{\text{new}} = 1 - \min_1(\underline{\mathbf{Q}}^{\exists,2} \times_{1,1} \neg \underline{\mathbf{B}} \times_{1,1} \neg \underline{\mathbf{R}}_{CD}^{\text{new}}) \quad (14)$$

and add them to $\Sigma_{F_{ABCD}}$. Now $\Sigma_{F_{ABCD}} = \{(11),(12),(13),(14)\}$. Finally we put

$$\begin{aligned} F_0 &= G_1^* = r_{ACD}^{\text{new}} \wedge r_{BCD}^{\text{new}}. \\ F_{ABCD}^{\text{tensor}} &= \underline{\mathbf{R}}_{ACD}^{\text{new}} \cdot \underline{\mathbf{R}}_{BCD}^{\text{new}}. \end{aligned}$$

So $[\![F_{ABCD}]\!]$ in $\mathbf{M}$ is evaluated without grounding by computing F_{ABCD}^{tensor} using $\Sigma_{F_{ABCD}} = \{(11),(12),(13),(14)\}$.

Binary predicates: matrix compilation

The compilation procedure in Figure 1 is general. It works for arbitrary prenex formulas F with arbitrary predicates. However when r is a binary predicate, the corresponding tensor $\underline{\mathbf{R}}$ is a bilinear map and represented by an adjacency matrix $\mathbf{R}$ as follows.

$$[\![r(e_i, e_j)]\!] = (\mathbf{e}_i \bullet \mathbf{R}\mathbf{e}_j) = \mathbf{e}_i^T \mathbf{R} \mathbf{e}_j = r_{ij} \in \{1,0\} \quad (15)$$

In such binary cases, we can often "optimize" compilation by directly compiling F using matrices without introducing Σ_F. This is quite important in processing KGs logically as they are a set of ground atoms with binary predicates. Hence we here derive some useful compilation patterns using matrices defined by (15) for formulas with binary predicates. We specifically adopt $[\![F]\!]_{\mathbf{Mat}}$ to denote the result of compilation using matrices that faithfully follows (2), (3), (4), (5), (6) and (7) in Subsection .

$$\begin{aligned} &[\![\exists y\, r_1(x,y) \wedge r_2(y,z)]\!]_{\mathbf{Mat}} \\ &= [\![(r_1(x,e_1) \wedge r_2(e_1,z)) \vee \cdots \vee (r_1(x,e_N) \wedge r_2(e_N,z))]\!]_{\mathbf{Mat}} \\ &= \min_1\Big(\sum_{j=1}^{N} [\![r_1(x,e_j) \wedge r_2(e_j,z)]\!]_{\mathbf{Mat}}\Big) \\ &= \min_1\Big(\sum_{j=1}^{N} \mathbf{x}^T \mathbf{R}_1 \mathbf{e}_j \mathbf{e}_j{}^T \mathbf{R}_2 \mathbf{z}\Big) \\ &= \mathbf{x}^T \min_1\Big(\mathbf{R}_1 \big(\sum_{j=1}^{N} \mathbf{e}_j \mathbf{e}_j{}^T\big) \mathbf{R}_2\Big) \mathbf{z} \\ &= \mathbf{x}^T \min_1\big(\mathbf{R}_1 \mathbf{R}_2\big) \mathbf{z} \end{aligned} \quad (16)$$

Here $\mathbf{x}$ and $\mathbf{z}$ run over $\mathscr{D}' = \{\mathbf{e}_1, \ldots, \mathbf{e}_N\}$. Hence the synthesized relation $r_{12}(x,y) \stackrel{\text{def}}{=} \exists y\, r_1(x,y) \wedge r_2(y,z)$ is encoded by a matrix $\mathbf{R}_{12} = \min_1(\mathbf{R}_1 \mathbf{R}_2)$. What is important with this example, or with binary predicates in general, is the fact that $\mathbf{Q}^{\exists,2} = \sum_{j=1}^{N} \mathbf{e}_j \mathbf{e}_j{}^T = I$, an identity matrix, holds.

Similarly by applying (16), we can compile a doubly quantified formula $\exists x \exists y\, r_1(x,y) \wedge r_2(x,y)$ as follows[2].

$$\begin{aligned} &[\![\exists x \exists y\, r_1(x,y) \wedge r_2(x,y)]\!]_{\mathbf{Mat}} \\ &= [\![(\exists y\, r_1(e_1,y) \wedge r_2(e_1,y)) \vee \cdots \vee (\exists y\, r_1(e_N,y) \wedge r_2(e_N,y))]\!]_{\mathbf{Mat}} \\ &= \min_1\big(\sum_{i=1}^{N} \mathbf{e}_i{}^T \min_1(\mathbf{R}_1 \mathbf{R}_2^T) \mathbf{e}_i\big) \\ &= \min_1(\mathrm{tr}(\mathbf{R}_1 \mathbf{R}_2^T)) \end{aligned} \quad (17)$$

Hence, a Horn formula $\forall x \forall y\, r_1(x,y) \Rightarrow r_2(x,y)$ is compiled into

$$[\![\forall x \forall y\, r_1(x,y) \Rightarrow r_2(x,y)]\!]_{\mathbf{Mat}} \quad (18)$$

$$\begin{aligned} &= [\![\neg \exists x \exists y\, r_1(x,y) \wedge \neg r_2(x,y)]\!]_{\mathbf{Mat}} \\ &= 1 - \min_1(\mathrm{tr}(\mathbf{R}_1 \neg \mathbf{R}_2{}^T)). \end{aligned} \quad (19)$$

Note that $\mathrm{tr}(\mathbf{R}_1 \neg \mathbf{R}_2{}^T)$ gives the number of pairs (x,y) that do not satisfy $r_1(x,y) \Rightarrow r_2(x,y)$. Consequently $\mathrm{tr}(\mathbf{R}_1 \neg \mathbf{R}_2{}^T) =$

[2] When $r(x,y)$ is encoded by $\mathbf{R}$ as $(\mathbf{x} \bullet \mathbf{R}\mathbf{y})$, $r(y,x)$ is encoded by $\mathbf{R}^T$ because $(\mathbf{y} \bullet \mathbf{R}\mathbf{x}) = (\mathbf{x} \bullet \mathbf{R}^T \mathbf{y})$ holds.

0 implies every pair (x,y) satisfies $r_1(x,y) \Rightarrow r_2(x,y)$ and vice versa. Our compilation is thus confirmed correct.

Another, typical, Horn formula $\exists y\, r_1(x,y) \wedge r_2(y,z) \Rightarrow r_3(x,z)$ is compiled into

$$\begin{aligned} & [\![\forall x \forall z \left(\exists y\, r_1(x,y) \wedge r_2(y,z) \Rightarrow r_3(x,z)\right)]\!]_{\mathbf{Mat}} \\ = \quad & 1 - \min_1\left(\mathrm{tr}(\min_1(\mathbf{R}_1\mathbf{R}_2)\neg\mathbf{R}_3{}^T)\right). \end{aligned} \tag{20}$$

Again $\mathrm{tr}\left(\min_1(\mathbf{R}_1\mathbf{R}_2)\neg\mathbf{R}_3{}^T\right)$ is the total number of (x,z)s that do not satisfy $\exists y\, r_1(x,y) \wedge r_2(y,z) \Rightarrow r_3(x,z)$. So our compilation is correct.

Recursive matrix equations

Our non-grounding linear-algebraic approach yields tensor equations from logical equivalence, and this property provides a new approach to the evaluation of Datalog programs. We sketch it using a small example. Consider the following Datalog program that computes the transitive closure `r2` of a binary relation `r1`.

```
r2(X,Z):- r1(X,Z).
r2(X,Z):- r1(X,Y),r2(Y,Z).
```

This program defines the least Hearbrand model $\mathbf{M}$ where `r1` is interpreted as r_1 and `r2` as r_1. $r_2(x_1,x_h)$ holds true if-and-only-if there is a chain $x_1,x_2,\ldots,x_h \in \mathbf{M}$ $(h \geq 1)$ such that $r_1(x_1,x_2), r_1(x_2,x_3), \ldots, r_1(x_{h-1},x_h)$ are all true in $\mathbf{M}$. Then we see the logical equivalence

$$r_2(x,z) \quad \Leftrightarrow \quad r_1(x,z) \vee \exists y (r_1(x,y) \wedge r_2(y,z)) \tag{21}$$

holds for all x,z in $\mathbf{M}$. That means

$$[\![r_2(x,z)]\!] \quad = \quad [\![r_1(x,z) \vee \exists y (r_1(x,y) \wedge r_2(y,z))]\!] \tag{22}$$

also holds for any x,z. Let $\mathbf{R}_1$ and $\mathbf{R}_2$ be adjacency matrices encoding r_1 and r_2 in $\mathbf{M}$ respectively. We translate (22) in terms of $\mathbf{R}_1$ and $\mathbf{R}_2$ as follows.

$$\begin{aligned} \mathbf{x}^T\mathbf{R}_2\mathbf{z} \quad &= \quad [\![r_2(x,z)]\!] \\ &= \quad [\![r_1(x,z) \vee \exists y (r_1(x,y) \wedge r_2(y,z)]\!] \\ &= \quad \min_1(\mathbf{x}^T\mathbf{R}_1\mathbf{z} + \mathbf{x}^T\min_1(\mathbf{R}_1\mathbf{R}_2)\mathbf{z}) \\ &= \quad \mathbf{x}^T\min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2)\mathbf{z} \end{aligned}$$

Since $\mathbf{x},\mathbf{z} \in \mathscr{D}'$ are arbitrary, we reach a recursive equation

$$\mathbf{R}_2 \quad = \quad \min_1(\mathbf{R}_1 + \mathbf{R}_1\mathbf{R}_2). \tag{23}$$

It is to be noted that when considered an equation for unknown $\mathbf{R}_2$, (23) may have more than one solution[3] but we can prove that the transitive closure is the "least" solution of (23) in the sense of matrix ordering[4](proof omitted).

Since (23) is a nonlinear equation due to $\min_1$ operation, it looks impossible to apply a matrix inverse to obtain $\mathbf{R}_2$. However we found a way to circumvent this difficulty and proved that it is possible to obtain $\mathbf{R}_2$ by computing (24) and (25) as follows.

$$\begin{aligned} \mathbf{R}_2 \quad &= \quad (\mathbf{R}_2^\dagger) > 0 && (24) \\ \mathbf{R}_2^\dagger \quad &= \quad (\mathbf{I} - \varepsilon\mathbf{R}_1)^{-1}\varepsilon\mathbf{R}_1 && (25) \\ & \quad\quad \text{where } \varepsilon = (1 + \|\mathbf{R}_1\|_\infty)^{-1} \end{aligned}$$

Here $(\mathbf{R}_2^\dagger) > 0$ means to threshold all elements in $\mathbf{R}_2$ at 0, i.e, positive ones are set to 1, o.w. to 0[5].

Experiment with transitive closure computation

We compared our linear algebraic approach to Datalog evaluation with state-of-the-art symbolic approaches using two tabled Prolog systems (B-Prolog (Zhou, Kameya, and Sato 2010) and XSB (Swift and Warren 2012)) and two ASP systems (DLV (Alviano et al. 2010) and Clingo (Gebser et al. 2014)). Although we conducted a number of experiments computing various programs with artificial and real data, due to space limitations, we here pick up one example that computes the transitive closure of random matrices. In the experiment[6], we generate random adjacency matrices by specifying the number of dimension N and the probability p_e of each entry being 1 and compute their transitive closure matrices using (24) and (25). We set $N = 1000$ and vary p_e from 0.0001 to 1.0 and measure the average computation time over five runs (details omitted).

p_e	Matrix	B-Prolog	XSB	DLV	Clingo
0.0001	0.096	0.000	0.000	0.000	0.000
0.001	0.094	0.004	0.003	0.293	0.038
0.01	0.117	2.520	1.746	10.657	14.618
0.1	0.105	18.382	16.296	75.544	125.993
1.0	0.100	188.280	137.903	483.380	1,073.301

Table 1: Average computation time for transitive closure computation (sec)

Table 1 shows the result. Our approach is termed "Matrix" in the table. Two observations are clear. First the computation time of our approach, Matrix, is almost constant while others seem linear w.r.t. p_e. Second, when p_e is small, $p_e = 0.0001 \sim 0.001$ and matrices are sparse, the Matrix method takes more time than existing systems but when p_e gets bigger, it runs orders of magnitude faster than them. The same observation is made with other programs (details omitted).

Related work

There is not much literature concerning first-order logic embedded in vector spaces. The most related work to ours is a formalization of first-order logic in tensor spaces by Grefenstette(Grefenstette 2013). He actually proposed two formalizations. The first one represents entities by one-hot

[3]For example, $\mathbf{R}_2 = \mathbf{1} \circ \mathbf{1}$ is a solution.

[4]Matrices $\mathbf{A} = \{a_{ij}\}$ and $\mathbf{B} = \{b_{ij}\}$ are ordered by $\mathbf{A} \leq \mathbf{B}$: $\mathbf{A} \leq \mathbf{B}$ if-and-only-if $a_{ij} \leq b_{ij}$ for all i,j.

[5]The proof and details are stated in an accompanying paper submitted for publication.

[6]All experiments are carried out on a PC with Intel(R) Core(TM) i7-3770@3.40GHz CPU, 28GB memory.

vectors, predicates by adjacency tensors and truth values by two-dimensional vectors (true by $\top = [1,0]^T$ false by $\bot = [0,1]^T$). AND and OR are order-3 order tensors whereas NOT is a 2×2 matrix that maps $\top$ to $\bot$ and vice versa. The first formalization can completely formalize a quantifier-free fragment of first-order logic in finite domains. The second formalization represents a finite set by a vector of multiple ones (and zeros) and can deal with single quantification by $\exists$ and $\forall$, but nested quantification is out of scope. The unification of the first and second formalizations remains an open problem to his tensor approach.

Krompass et al. (Krompaß, Nickel, and Tresp 2014) proposed a way of answering existential queries of the form $\exists x\, Q_1 \wedge Q_2$ in the context of low-dimensional embeddings. Their approach however does not assign an independent representation to existential quantifiers and is limited to a narrow class of the form $\exists x\, Q_1 \wedge Q_2$.

We found no literature on computing the least model of Datalog programs via solving recursive matrix equations. So the transitive closure computation presented in this paper is possibly the first example of this kind.

Conclusion

We proposed a general approach to evaluate first-order formulas F in prenex normal form in vector spaces. Given a finite model $\mathbf{M}$ with N entities, we compile F into a set Σ_F of hierarchical tensor definitions (equations) with a nonlinear operation. Computing Σ_F in $\mathbb{R}^N$ yields the truth value $[\![F]\!]$ in $\mathbf{M}$. In this compilation process, tensor representation $\underline{\mathbf{Q}}^{\exists,M}$ is introduced to existential quantifiers themselves for the first time as far as we know. Since our approach does not rely on propositionalization of first-order formulas, it can derive tensor equations from logical equivalences. We exploited this property to derive recursive matrix equations to evaluated Datalog programs. We empirically demonstrated the effectiveness of our linear algebraic approach by showing that it runs orders of magnitude faster than existing symbolic approaches when matrices are not too sparse.

References

Alviano, M.; Faber, W.; Leone, N.; Perri, S.; Pfeifer, G.; and Terracina, G. 2010. The Disjunctive Datalog System DLV. In de Moor, O.; Gottlob, G.; Furche, T.; and Sellers, A., eds., *Datalog Reloaded, LNCS 6702*. Springer. 282–301.

Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; and Taylor, J. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of data*, 1247–1250.

Cichocki, A.; Zdunek, R.; Phan, A.-H.; and Amari, S. 2009. *Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. John Wiley & Sons, Ltd.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2014. Clingo = ASP + Control: Preliminary Report. volume arXiv:1405.3694v1, 1–9.

Grefenstette, E. 2013. Towards a Formal Distributional Semantics: Simulating Logical Calculi with Tensors. *Proceedings of the Second Joint Conference on Lexical and Computational Semantics* 1–10.

Guu, K.; Miller, J.; and Liang, P. 2015. Traversing Knowledge Graphs in Vector Space. In *Proceedings of the 2015 Empirical Methods in Natural Language Processing (EMNLP)*, 318–327.

Kolda, T. G., and Bader, B. W. 2009. Tensor decompositions and applications. *SIAM REVIEW* 51(3):455–500.

Krompaß, D.; Nickel, M.; and Tresp, V. 2014. Querying Factorized Probabilistic Triple Databases. In *Proceedings of the 13th International Semantic Web Conference(ISWC'14)*, 114–129.

Nickel, M.; Murphy, K.; Tresp, V.; and Gabrilovich, E. 2015. A Review of Relational Machine Learning for Knowledge Graphs: From Multi-Relational Link Prediction to Automated Knowledge Graph Construction. *CoRR* abs/1503.00759.

Rocktäschel, T.; Singh, S.; and Riedel, S. 2015. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Swift, T., and Warren, D. 2012. XSB: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming (TPLP)* 12(1-2):157–187.

Yang, B.; Yih, W.; He, X.; Gao, J.; and Deng, L. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*.

Zhou, N.-F.; Kameya, Y.; and Sato, T. 2010. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *Proceedings of the 22th International Conference on Tools with Artificial Intelligence (OCTAL-2010)*, 213–218.