Extending Weakly-Sticky Datalog$^\pm$:

Query-Answering Tractability and Optimizations

Leopoldo Bertossi

Carleton University
Ottawa, Canada

Join work with: Mostafa Milani

Datalog± and Our Goal

- Datalog$^{\pm}$: An extension of Datalog with some useful ontological constructs (the $+$)

  In particular, Datalog^{+} allows for $\exists$-variables in TGDs

  With syntactic restrictions for good computational properties (the $-$)

- We start from sticky and weakly-sticky (WS) Datalog$^{\pm}$ programs [Cali et al. AIJ'12]

- Both well-behaved under the chase procedure in relation to generation and propagation of nulls, QA, etc.

- WS programs appear in our applications to "quality data" specification and extraction [Milani et al. RuleML'15]

- Sticky programs (SP) are defined on the basis of a variable marking procedure

- WS programs extend SPs by applying the notion of weak-acyclicity (WA) (as in DE) (also syntactic class)

  Requires dependency graphs with special edges

  Define Π_F and Π_∞: finite- and infinite-rank positions (the former take finitely many values during the chase)

  Repeated occurrences of a marked variable in a body must appear in at least one position in Π_F

- For both classes there are polynomial-time QA algorithms

  A "theoretical" one for WS [Cali et al., AIJ 2012]

In this work:

- We propose a chase-based QA algorithm
- We consider its optimization using a magic-sets (MS) technique

  Query-driven rewriting of the program, for faster evaluation
- WS is not closed under MS
- We extend WS to a good program class closed under MS

  For which the QA algorithm works
- Process takes us first to "semantic classes" that extend WS

Chase: Stickiness and Weak-Stickiness

- SP and WS are syntactic program classes
- There are "semantic" classes

  They refer to behavior under the chase propagation process (which may involve the extensional data, D)

- In particular, "chase-stickiness" (SCh): [Cali et al., AIJ'12]

  "If a TGD σ is (groundedly) applied with a value v replacing a repeated variable in σ's body, then v is propagated all the way through all possible subsequent chase steps"

- SP $\Rightarrow$ SCh (for any D) ($\not\Leftarrow$)

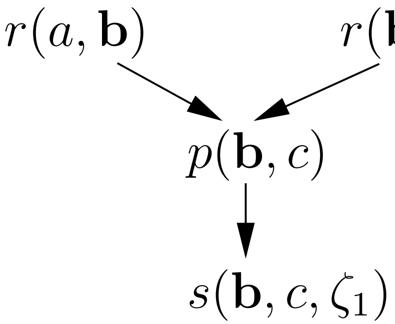
Example: Programs $\mathcal{P}_1$ and $\mathcal{P}_2$ with $D = \{r(a,b), r(b,c)\}$

$\mathcal{P}_1^r$:

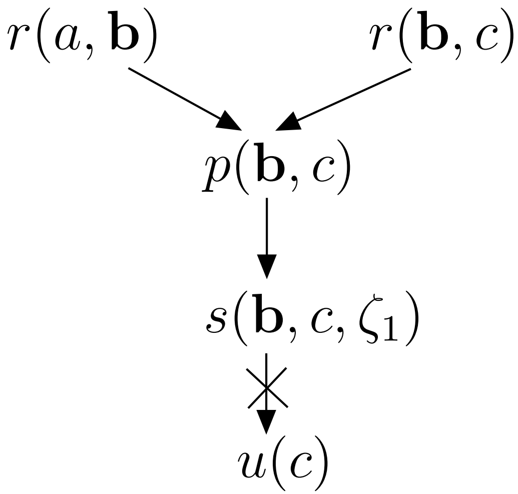
$$r(X,Y), r(Y,Z) \rightarrow p(Y,Z). \qquad p(X,Y) \rightarrow \exists Z\; s(X,Y,Z).$$

$$s(X,Y,Z) \rightarrow u(Y).$$

$\mathcal{P}_2^r$ is $\mathcal{P}_1^r$ w/o third TGD



$\mathcal{P}_2$ is SCh!



$\mathcal{P}_1$ not SCh

b replacing Y not fully propagated in $\mathcal{P}_1$, but it is with $\mathcal{P}_1$

- Syntactic stickiness condition is characterized using a two-step marking procedure:

1. Preliminary step: Mark body variables that do not appear in the heads

$$\mathcal{P}^r: \quad r(\hat{X}, Y), r(Y, Z) \rightarrow p(Y, Z). \qquad p(X, Y) \rightarrow \exists Z\, s(X, Y, Z).$$
$$s(\hat{X}, Y, \hat{Z}) \rightarrow u(Y).$$

2. Propagation step: Mark body variables that appear in the heads only in positions where marked variables occur in other rules

$$\mathcal{P}^r: \quad r(\hat{X}, \hat{Y}), r(\hat{Y}, Z) \rightarrow p(Y, Z). \qquad p(\hat{X}, Y) \rightarrow \exists Z\, s(X, Y, Z).$$
$$s(\hat{X}, Y, \hat{Z}) \rightarrow u(Y).$$

- A program is sticky if it does not have a marked join variable

$\mathcal{P}^r$ is not sticky due to Y in the first rule

- SCh is a semantic notion .... depends on chase and data

  Still well-behaved wrt. QA

- More general semantic classes? Well behaved?

  Generalized stickiness of the chase (GSCh): Relax condition on SCh:

  "... repeated variable that do not appear in finite positions (i.e. only in infinite positions) ..."

  Infinite (finite) position: unlimited (finite) number of values appear in the chase

- "Position (in)finiteness" is undecidable

  So as chase termination ...

  GSCh of a program also undecidable

- Π_F " $\subsetneqq$ " finite positions

  "finite-rank positions" sound but incomplete wrt. finite positions

  Π_F "selects" some finite positions

- We want to investigate different "selection functions" and the corresponding program classes

  That is, programs for which values in join variables *not appearing in selected finite positions* are ...

Finite Positions and Program Classes

- Set-valued function $\mathcal{S}$ returning a subset of a program's finite positions

  SCh($\mathcal{S}$): subclass of GSCh replacing "finite position" by "$\mathcal{S}$-finite position"

  - For $\mathcal{S}^{\perp}$ returning the empty set of finite positions: SCH:=SCh($\mathcal{S}^{\perp}$) is a semantic version of the class of sticky programs
  - For $\mathcal{S}^{rank} := \Pi_F$ (finite-rank positions): WSCh:=SCh($\mathcal{S}^{rank}$) is the semantic class of programs with weak-stickiness of the chase

    WSCh is the semantic version of WS
  - For $\mathcal{S}^{\top}$ returning all the (semantically) finite positions: SCh($\mathcal{S}^{\top}$) is the class GSCh

- A range of semantic classes and their syntactic versions:

$$
\begin{array}{ccccccc}
\mathcal{S}^{\perp} & \underset{(a)}{\subsetneqq} & \mathcal{S}^{rank} & \underset{(b)}{\subsetneqq} & \mathcal{S}^{\exists} & \underset{(c)}{\subsetneqq} & \mathcal{S}^{\top} \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
\mathit{SCh}{:=}\mathit{SCh}(\mathcal{S}^{\perp}) & \underset{(d)}{\subsetneqq} & \mathit{WSCh}{:=}\mathit{SCh}(\mathcal{S}^{rank}) & \underset{(e)}{\subsetneqq} & \mathit{SCh}(\mathcal{S}^{\exists}) & \underset{(f)}{\subsetneqq} & \mathit{GSCh}{:=}\mathit{SCh}(\mathcal{S}^{\top}) \\
\underset{(g)}{\cup\nparallel} & & \underset{(h)}{\cup\nparallel} & & \underset{(i)}{\cup\nparallel} & & \\
\mathit{Sticky} & \underset{(j)}{\subsetneqq} & \mathit{WS} & \underset{(k)}{\subsetneqq} & \mathit{JWS} & &
\end{array}
$$

$\mathit{SCh}(\mathcal{S})$ grows monotonically with $\mathcal{S}$: the more finite positions identified by $\mathcal{S}$, the more general the subclass of GSCh (this one undecidable)

- A computable $\mathcal{S}$ defines a "decidable class" SCh($\mathcal{S}$), i.e. BCQ answering is decidable. E.g. $\mathcal{S}^{\perp}$ and $\mathcal{S}^{rank}$

Joint-Weak-Stickiness

- A new selection function $\mathcal{S}^{\exists}$
- Use joint-acyclicity and existential dependency graph (EDG)

  [Rudolph et al., IJCAI'11]
- EDGs allow to define $\exists$-finite positions, a subset $\Pi_F^{\exists}$ of finite positions
- Use selection function $\mathcal{S}^{ext} := \Pi_F^{\exists}$

  Computable and more general than $\mathcal{S}^{rank}$
- New semantic and syntactic classes: $SCh(\mathcal{S}^{\exists})$ and joint-weakly-sticky (JWS) programs, resp.
- $SCh(\mathcal{S}^{\exists})$ extends $SCh(\mathcal{S}^{rank})$ and JWS extends WS

QA for $SCh(\mathcal{S})$

- QA algorithm QA($SCh(\mathcal{S})$) for SCh($\mathcal{S}$) with computable $\mathcal{S}$

  Takes as input: program $\mathcal{P}$ and a CQ $\mathcal{Q}$

  (not necessarily efficient, see later ...)

- Some notations used in QA($SCh(\mathcal{S})$):

A pair rule/assignment σ and θ is applicable over I if:

1. $I \models (body(\sigma))[\theta]$
2. $\theta'(head(\sigma))$ is not isomorphic to any atom in I (via nulls)
   (θ' extends θ with mapping for $\exists$-variables of σ into fresh nulls)

Freezing a null is moving it into the set of constants

Resumption is freezing every null in an instance and continuing the algorithm

The QA($SCh(\mathcal{S})$) algorithm:

Step 1: Initialize an instance I with the the extensional database D

Step 2: Chose an applicable σ and θ over I, add $head(\sigma)[\theta']$ into I (θ' extends θ with mappings for $\exists$-variables in σ to fresh nulls)

Step 3: Freeze new nulls that appear in a position of $\mathcal{S}(\mathcal{P})$ (relevant for isom test)

Step 4: Iteratively apply Steps 2 and 3 until saturation

Step 5: Resume Step 2, that is, freeze every null in I and continue with Step 2, repeat resumption $M_{\mathcal{Q}}$ times, the number of variables in $\mathcal{Q}$ (now nulls in positions "outside" $\mathcal{S}$ are being frozen)

Step 6: Return the tuples in $\mathcal{Q}(I)$ that do not have null values (including frozen nulls)

In Step 2 each pair is applied once

Example: Program $\mathcal{P}$ with $D = \{s(a,b,c), v(b), u(c)\}$ and $\mathcal{P}^r$:

$\sigma_1:\ s(X,Y,Z) \rightarrow \exists W\ s(Y,Z,W).$ $\sigma_3: s(X,Y,Z), v(X), s(Y,Z,W) \rightarrow p(Y,Z).$

$\sigma_2:\ u(X) \rightarrow \exists Y,Z\ s(X,Y,Z).$

A BCQ $\mathcal{Q}$: $p(c,Y) \rightarrow ans_{\mathcal{Q}}$ $\mathcal{S}^{rank}$ as selection function

$\mathcal{P}$ is *WS* and then also in *SCh*$(\mathcal{S}^{rank})$

QA($SCh(\mathcal{S})$) runs as follows:

- Step 1: $I = \{s(a,b,c), v(b), u(c)\}$
- Steps 2-3: σ_1,$\theta_1 = \{X \rightarrow a, Y \rightarrow b, Z \rightarrow c\}$ are applicable and add $s(b,c,\zeta_1)$
  ζ_1 is not frozen $s[3] \notin \mathcal{S}^{rank}(\mathcal{P})$ Step 4: repeat Steps 2 and 3
- Steps 2-3: σ_2, $\theta_2 = \{X \rightarrow c\}$ are applicable and add $s(c,\zeta_2,\zeta_3)$
  ζ_2 and ζ_3 are not frozen (same reason)

- Step 2: No more applicable rule-assignment, σ_1 and $\theta_3 = \{X \rightarrow b, Y \rightarrow c, Y \rightarrow \zeta_1\}$ are not applicable, because $s(c, \zeta_1, \zeta_4)$ is isomorphic to $s(c, \zeta_2, \zeta_3)$

- Step 5: Freeze ζ_1, ζ_2, and ζ_3 and resume Step 2

  $\mathcal{Q}$ has one variable and we resume once (here "infinite" nulls are frozen)

- Steps 2-3: Now σ_1 and θ_3 become applicable and add $s(c, \zeta_1, \zeta_4)$

  ζ_4 is not frozen (not isom anymore)

- Steps 2-3: σ_3 and $\theta_4 = \{X \rightarrow b, Y \rightarrow c, Z \rightarrow \zeta_1, W \rightarrow \zeta_4\}$ are applicable and add $p(c, \zeta_1)$ ζ_1 is already frozen

- No applicable rule-assignment, no more resumptions

- Step 6: $I = D \cup \{s(b, c, \zeta_1), s(c, \zeta_2, \zeta_3), s(c, \zeta_1, \zeta_4), p(c, \zeta_1)\}$

  Return: $\mathcal{Q}$ is true

- QA($SCh(\mathcal{S})$) is applicable for any program in Datalog$^+$:

  It terminates returning sound answers ($\mathcal{S}$ computable)

- QA($SCh(\mathcal{S})$) also complete if applied with $\mathcal{S}$ to a program in SCh($\mathcal{S}$)

- It runs in polynomial-time in data when polynomially many values appear in $\mathcal{S}$-finite positions (in data)

- In particular it is tractable for SP, WS, JWS and their semantic classes

Magic-Sets Rewriting

- We consider a magic-sets rewriting for Datalog$^+$: MagicD$^+$

- Extension of the rewriting in [Alviano et al., Datalog 2.0'12] for Datalog$^\exists$
 programs

- Quite similar to MS for Datalog, but:

  1. MagicD$^+$ does not bound variables if they are existentially quantified

     Nothing like this: $\exists Z\ r^{fb}(X, Z) \leftarrow\ r^{ff}(X, Y)$

  2. Allows intentional predicates with extensional data

- MagicD$^+$ takes a program $\mathcal{P}$ and a CQ $\mathcal{Q}$; returns $\mathcal{P}_m$, $\mathcal{Q}_m$

- $ans(\mathcal{Q}, \mathcal{P}) = ans(\mathcal{Q}_m, \mathcal{P}_m)$ and "faster evaluation" of $\mathcal{Q}_m$ over $\mathcal{P}_m$

Example: A program $\mathcal{P}$ with $D = \{u(a), r(a,b)\}$ and $\mathcal{P}^r$:

$$r(X,Y), r(Y,Z) \ \rightarrow\ p(X,Z) \qquad\qquad u(Y), r(X,Y) \ \rightarrow \exists Z\ r(Y,Z)$$

A BCQ $\mathcal{Q}:\ p(a,Y)\ \rightarrow\ ans_{\mathcal{Q}}$

MagicD$^+$:

Step 1: (adorned rules generation)

Adorned version of the query: $p^{bf}(a,Y)\ \rightarrow\ ans_{\mathcal{Q}}$

Adorned versions of the rules in $\mathcal{P}^r$: (depends on SWIP)

$$r^{bf}(X,Y), r^{bf}(Y,Z)\ \rightarrow p^{bf}(X,Z). \qquad u(Y), r^{fb}(X,Y)\ \rightarrow \exists Z\ r^{bf}(Y,Z)$$

The first rule is not adorned with fb since Z is a $\exists$-variable

Step 2: Add magic predicates to the adorned rules:

$$mg\_p^{bf}(X),\ r^{bf}(X,Y), r^{bf}(Y,Z)\ \rightarrow p^{bf}(X,Z).$$

$$mg\_r^{bf}(Y),\ u(Y), r^{fb}(X,Y)\ \rightarrow \exists Z\ r^{bf}(Y,Z).$$

Step 3: Add the definition of the magic predicates:

$$mg\_p^{bf}(X)\ \rightarrow mg\_r^{bf}(X). \qquad mg\_r^{bf}(X), r^{bf}(X,Y)\ \rightarrow mg\_r^{bf}(Y).$$

And a fact $mg\_p^{bf}(a)$

Step 4: Load the extensional data of the adorned predicates:

$$mg\_r^{bf}(X), r(X,Y)\ \rightarrow r^{bf}(X,Y). \qquad mg\_r^{fb}(Y), r(X,Y)\ \rightarrow r^{fb}(X,Y).$$

- If $\mathcal{P}$ is WS, $\mathcal{P}_m$ is not necessarily WS or in SCh($\mathcal{S}^{rank}$)
- So WS and SCh($\mathcal{S}^{rank}$) are not closed under MagicD$^+$

  This is due to the new join variables in Step 2

  They might be marked and appear in infinite rank positions
- JWS and SCh($\mathcal{S}^{\exists}$) are closed under MagicD$^+$

  That is because the positions of the new joins are bounded

  $\mathcal{S}^{\exists}$ always characterizes them as finite (unlike $\mathcal{S}^{rank}$)
- QA($SCh(\mathcal{S}^{\exists})$) can be optimized using MagicD$^+$ for JWS programs

  This includes every program in WS and SP

EXTRA SLIDES

Example: A program $\mathcal{P}$ with, $D = \{r(a,b), v(b)\}$, a BCQ $\mathcal{Q}$: $r(Y,a) \rightarrow ans_{\mathcal{Q}}$ and $\mathcal{P}^r$:

$$r(X,Y) \rightarrow \exists Z\, r(Y,Z). \qquad r(X,Y) \rightarrow \exists Z\, r(Z,X).$$

$$r(X,Y), r(Y,Z), v(Y) \rightarrow r(Y,X).$$

$\mathcal{P}$ is *WS* the only repeated marked variable is Y that appears in $v[1] \in \Pi_F(\mathcal{P}^r)$

- The adorned query rule is $\mathcal{Q}_m$: $r^{fb}(Y,a) \rightarrow ans_{\mathcal{Q}}$
- $\mathcal{P}^m$ has adorned rules:

$$mg\_r(Y), r^{fb}(X,Y) \rightarrow \exists Z\, r^{bf}(Y,Z).$$
$$mg\_r(X), r^{bf}(X,Y) \rightarrow \exists Z\, r^{fb}(Z,X).$$
$$mg\_r(X), r^{bf}(X,Y), r^{bf}(Y,Z), v(Y) \rightarrow r^{fb}(Y,X).$$
$$mg\_r(Y), r^{fb}(X,Y), r^{bf}(Y,Z), v(Y) \rightarrow r^{bf}(Y,X).$$

And the magic rules (and a fact $mg\_r(a)$):

$$mg\_r(X), r^{bf}(X,Y) \rightarrow mg\_r(Y). \quad (1)$$

$$mg\_r(Y), r^{fb}(X,Y) \rightarrow mg\_r(X). \quad (2)$$

Every body variable in $\mathcal{P}^m$ is marked

mg_r^{fb} and mg_r^{bf} are equivalent and are replaced with mg_r

$\mathcal{P}_m$ is *not WS*, $r^{fb}[1], r^{fb}[2], r^{bf}[1], r^{bf}[2]$, and $mg\_r[1]$ are have infinite rank

The new join variables with magic predicates break the syntactic property of *WS* and also the SCh($\mathcal{S}^{rank}$)-stickiness property

Joint-Acyclicity and ∃-finite positions

- A set of rules is standardized apart if no variable appears in more than one rule

- For a variable X, $B(X)$ and $H(X)$ are the sets of all positions where X occurs in the body and the head of a rule resp.

- Target positions of a ∃-variable Z $(T(Z))$ is the smallest set of positions such that

  1. $H(Z) \subseteq T(Z)$
  2. $H(X) \subseteq T(Z)$ for every ∀-variable X with $B(X) \subseteq T(Z)$

Roughly $T(Z)$ is the set of positions where the null values invented by Z may appear during the chase

- An *existential dependency graph* (*EDG*) of $\mathcal{P}^r$ is a directed graph with:

  - $\exists$-variables of $\mathcal{P}^r$ as its nodes
  - There is an edge from Z to Z' if there is variable X in the body of the rule containing Z' such that $B(X) \subseteq T(Z)$

- Intuitively the edge shows that the values invented by Z might appear in the body of the rule of Z'

- A cycle including Z and Z' shows the possibility of cyclic and infinite value invention Z and Z'

- A program is *joint-acyclic* (JA) if its *EDG* is acyclic

[Rudolph et al., IJCAI'11]

- They properly extend *WA* programs

- They have polynomial size chase w.r.t the size of the extensional data

Example: A program $\mathcal{P}$ with $\mathcal{P}^r$:

$$u(Y), r(X,Y) \rightarrow \exists Z\, r(Y,Z) \qquad\qquad r(X,Y), r(Y,Z) \rightarrow p(X,Z)$$

- $B(Y) = \{u[1], r[2]\}$ and $H(Y) = \{r[1]\}$ and $T(Z) = \{r[2]\}$

- The *EDG* of $\mathcal{P}^r$ has Z as its node

There is no edge since $B(X)$ and $B(Y)$ are not subsets of $T(Z)$

- So $\mathcal{P}^r$ is JA but not WA ($r[1]$ and $r[2]$ have infinite rank)

- We define set of $\exists$-finite of $\mathcal{P}^r$ denoted by $\Pi_F^{\exists}(\mathcal{P}^r)$ as:

  The set of positions that are not in the target set of any $\exists$-variable in a cycle in *EDG*$(\mathcal{P}^r)$

Intuitively, a position in $\Pi_F^{\exists}(\mathcal{P}^r)$ is not in the target of any $\exists$-variable that may invent infinite null values

In the example in page 26 every position is $\exists$-finite since there is no cycle in the EDG