TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 19: The Chase

David Carral
Knowledge-Based Systems

TU Dresden, July 3, 2020

Review

Definition 18.7: A dependency is a formula of the form

$$\forall \vec{x}, \vec{y}. \varphi[\vec{x}, \vec{y}] \rightarrow \exists \vec{z}. \psi[\vec{x}, \vec{z}],$$

where $\vec{x}, \vec{y}, \vec{z}$ are disjoint lists of variables, and φ (the body) and ψ (the head) are conjunctions of atoms using variables $\vec{x} \cup \vec{y}$ and $\vec{x} \cup \vec{z}$, respectively. We allow equality atoms $s \approx t$ to occur in dependencies. The variables $\vec{x}$, which occur in body and head, are known as frontier.

- Tuple-generating dependencies (tgds) are not using equality (in the head)
  (Special case: inclusion dependencies)
- Equality-generating dependencies (egds) have single head atoms with the equality predicate
  (Special case: functional dependencies, esp. keys)
- Full dependencies have no $\exists$ (quite similar to Datalog)

Many reasoning tasks on tgds can be reduced to query answering under tgd constraints, but this problem is undecidable in general.

The Chase

Review: Evaluating Datalog

We recall the semi-naive bottom-up evaluation of Datalog in alternative notation:

- We compute sets of facts Δ^i for each step $i = 0, 1, 2, \ldots$
- Let $\Delta^{[i,j]} = \bigcup_{k=i}^{j} \Delta^k$
- A substitution θ is a match of a CQ φ over a set of facts Δ if $\varphi\theta \subseteq \Delta$

We can describe semi-naive evaluation as follows:

```
function chase(Σ, I)
01  i = 0        Δ^0 = I
02  repeat :
03      applyRules(Σ, Δ, i)
04  until Δ^i = ∅
05  return Δ^[0,i]
```

```
function applyRules(Σ, Δ, i)
01  Δ^{i+1} = ∅
02  foreach (φ → ψ) ∈ Σ :
03      foreach match θ of φ over Δ^[0,i] with φθ ∩ Δ^i ≠ ∅ :
04          Δ^{i+1} = (Δ^{i+1} ∪ ψθ) \ Δ^[0,i]
05  i = i + 1
```

Then chase$(\Sigma, \mathcal{I})$ for a set of full tgds Σ corresponds to the least model of Datalog.
(Essentially this chase is simply an alternative version of the T_P operator)

Chasing tgds

Can we perform a bottom-up computation for non-full tgds?

New feature: existential quantifiers in rule heads $\rightsquigarrow$ create new domain elements

Definition 19.1: Let **N** be a countably infinite set of (labelled) nulls, distinct from constants and variables.

We can use (fresh) nulls to satisfy existential quantifiers during the chase:

`function` applyRules(Σ, Δ, i)

01 $\Delta^{i+1} = \emptyset$
02 **foreach** $(\varphi \rightarrow \exists \vec{z}.\psi) \in \Sigma$:
03 **foreach** match θ of φ over $\Delta^{[0,i]}$ with $\varphi\theta \cap \Delta^i \neq \emptyset$:
04 $\theta' = \theta \cup \{\vec{z} \mapsto \vec{n}\}$, where $\vec{n} \subseteq$ **N** are fresh nulls
05 $\Delta^{i+1} = (\Delta^{i+1} \cup \psi\theta') \setminus \Delta^{[0,i]}$
06 $i = i + 1$

The outer loop (function chase(Σ, $\mathcal{I}$)) remains as before.

Termination

When introducing new nulls, the chase may fail to terminate.

Example 19.2: Consider the inclusion dependencies

$$\text{Connect}(x, y, z) \rightarrow \exists v.\text{Lines}(z, v)$$
$$\text{Lines}(x, y) \rightarrow \exists v, w.\text{Connect}(v, w, x)$$

The chase over the database with a single fact Lines(85, bus) produces facts $\Delta^1 = \{\text{Connect}(n_1, n_2, 85)\}$, $\Delta^2 = \{\text{Lines}(85, n_3)\}$, $\Delta^3 = \{\text{Connect}(n_4, n_5, 85)\}$, $\Delta^4 = \{\text{Lines}(85, n_6)\}$, ...

Even if the chase does not terminate, we may consider the countably infinite structure that it produces in the limit. We define:

$$\text{chase}(\Sigma, \mathcal{I}) = \bigcup_{i \geq 0} \Delta^i$$

The chase is a model

Even an infinite chase is (conceptually) useful to solve reasoning problems.

Theorem 19.3: Consider a set Σ of tgds and a database $\mathcal{I}$.
Then $\mathcal{I} \subseteq \text{chase}(\Sigma, \mathcal{I})$ and $\text{chase}(\Sigma, \mathcal{I}) \models \Sigma$.

(Alternatively, we could consider $\mathcal{I}$ as a set of facts and state that $\text{chase}(\Sigma, \mathcal{I}) \models \Sigma \cup \mathcal{I}$.)

Proof: Easy (suppose for a contradiction that it is not a model; conclude that some tgd must be violated in a specific way; observe that this tgd would be applicable in this way and must therefore already have been applied – contradiction). □

The chase is universal

An essential property of the chase is as follows:

Theorem 19.4: Consider a set Σ of tgds and a database $\mathcal{I}$. If $\mathcal{J} \supseteq \mathcal{I}$ is a (possibly infinite) interpretation with $\mathcal{J} \models \Sigma$ then there is a homomorphism $\mu : \text{chase}(\Sigma, \mathcal{I}) \rightarrow \mathcal{J}$.

Proof: We construct μ inductively along the computation of the chase by building a sequence of finite homomorphisms $\mu_0 \subseteq \mu_1 \subseteq \mu_2 \subseteq \ldots$ such that $\mu_i : \Delta^{[0,i]} \rightarrow \mathcal{J}$.

- Initially, let μ_0 be the identity mapping on $\mathcal{I}$
- In step i, let $\varphi \rightarrow \exists \vec{z}.\psi$ be a rule that contributes facts $\psi\theta'$ to Δ^{i+1} for the match θ (extended to add new nulls)
  - Then $\mu_i(\varphi\theta) \subseteq \mathcal{J}$ by induction hypothesis, hence there is a corresponding substitution $\sigma : x \mapsto \mu_i(x\theta)$ on $\mathcal{J}$ with $\varphi\sigma \subseteq \mathcal{J}$
  - Since $\mathcal{J} \models \Sigma$, we find an extension σ' of σ such that $\psi\sigma' \subseteq \mathcal{J}$
  - If $z\theta' = n$ is a new null introduced in Δ this step, then let $\mu_{i+1}(n) = \sigma'(z)$.
  - Let $\mu_{i+1}(t) = \mu_i(t)$ for all t that occurred in $\Delta^{[0,i]}$

It is not hard to see that $\mu = \bigcup_{i \geq 0} \mu_i$ is the required homomorphism. □

BCQ answering with the chase

Definition 19.5: A model $\mathcal{J}$ of a theory $\mathcal{T}$ that has a homomorphism into every other model of $\mathcal{T}$ is called universal.

Note (for students of universal algebra): Universal models are initial objects in the category of models of $\mathcal{T}$ with homomorphisms.

Observation 19.6: All universal models of $\mathcal{T}$ are homomorphically equivalent.

Theorem 19.7: A BCQ q is entailed from a database $\mathcal{I}$ and set of tgds Σ iff q has a match with chase$(\Sigma, \mathcal{I})$.

Proof: The "only if" direction follows since chase$(\Sigma, \mathcal{I})$ is a model.

The "if" direction follows since chase$(\Sigma, \mathcal{I})$ is universal: a match is a homomorphism from variable of q into chase$(\Sigma, \mathcal{I})$, which can be extended by a homomorphism μ : chase$(\Sigma, \mathcal{I}) \to \mathcal{J}$ to any other model $\mathcal{J}$ of $\Sigma \cup \mathcal{I}$. □

Semi-deciding BCQ entailment

The previous result yields a semi-decision procedure even if the chase does not terminate:

Corrolary 19.8: BCQ entailment under constraints defined by sets of tgds is semi-decidable.

Proof: If a BCQ is entailed by chase$(\Sigma, \mathcal{I})$, then it has a match to a finite part $\Delta^{[0,i]} \subseteq$ chase$(\Sigma, \mathcal{I})$ for some chase step i. A semi-decision procedure therefore can check for such matches while computing the chase. □

By the undecidability result, this is the best we can hope for in the general case.

However, one might hope for a finite chase in practical cases.

The chase revisited

The chase we specified so far is the oblivious chase. It tends to not terminate even in simple cases.

A better approach is the restricted chase that includes an additional check (line 4):

`function` applyRules(Σ, Δ, i)

01 $\Delta^{i+1} = \emptyset$
02 **foreach** $(\varphi \to \exists \vec{z}\psi) \in \Sigma$:
03 **foreach** match θ of φ over $\Delta^{[0,i]}$ with $\varphi\theta \cap \Delta^i \neq \emptyset$:
04 **if** $\Delta^{[0,i]} \not\models \exists \vec{z}.\psi\theta$:
05 $\theta' = \theta \cup \{\vec{z} \mapsto \vec{n}\}$, where $\vec{n} \subseteq$ **N** are fresh nulls
06 $\Delta^{i+1} = (\Delta^{i+1} \cup \psi\theta') \setminus \Delta^{[0,i]}$
07 $i = i + 1$

Example

Example 19.9: Consider the inclusion dependencies as before:

$$\text{Connect}(x, y, z) \to \exists v.\text{Lines}(z, v)$$
$$\text{Lines}(x, y) \to \exists v, w.\text{Connect}(v, w, x)$$

The restricted chase over the database with a single fact Lines(85, bus) first produces facts $\Delta^1 = \{\text{Connect}(n_1, n_2, 85)\}$. In the next iteration, however, we find that $\{\text{Lines}(85, \text{bus})\} \models \exists v.\text{Lines}(85, v)$, and hence $\Delta^2 = \emptyset$ and the chase terminates.

There are still cases where even the restricted chase does not terminate.

How can we know if this is the case?

Chase Termination

Chase Termination

If the chase terminates on a set Σ of tgds, then we can use it to decide query entailment over Σ.

- Termination may happen for some database instances but not for others
- Termination on all instances is called universal termination

We cannot expect this good property to hold in general:

Theorem 19.10: The tgd $p(x, y) \rightarrow \exists z.p(y, z)$ over the database instance with fact $p(a, b)$ does not have a finite universal model.

Corollary: no chase that produces finite models can terminate in this case.

Proof: Any finite structure either has (a) p-chains of bounded length, or (b) a p-cycle. In case (a), there is a BCQ for a longer p-chain that does not match (but that is entailed); in case (b) there is a BCQ for a cycle that does match (but that is not entailed). Hence no finite model can be universal. □

Chase termination is difficult

Theorem 19.11: The question if the (restricted or oblivious) chase terminates on a set of tgds Σ and database $\mathcal{I}$ is undecidable.

Proof: By the same proof as for the undecidability of BCQ entailment (Theorem 18.17):

- Simulate a deterministic TM in tgds
- The simulation will terminate iff the DTM halts □

Note 1: If the DTM does not halt, the universal model must contain an infinite forward-directed chain of successor configurations. Then, as in Example 19.10, we find that no finite model can be universal. Hence the argument extends to any chase procedure.

Note 2: Termination is clearly semi-decidable (just run the chase).

Universal chase termination

The situation for universal termination is more complicated.

Oblivious chase termination over arbitrary database instance is equivalent to termination over a special critical instance:

Proposition 19.12: Consider a set Σ of tgds. The oblivious chase terminates over Σ on all instances iff it terminates over the following critical instance $\mathcal{I}_\star$:

- $\Delta^{\mathcal{I}_\star} = \{\star\}$,
- for every n-ary predicate symbol p in Σ, $p^{\mathcal{I}_\star} = \star^n$.

Proof: Based on the following simple observation:

Any homomorphism $\mu : \mathcal{I} \rightarrow \mathcal{J}$ between two database instances extends to a homomorphism μ : chase$(\Sigma, \mathcal{I}) \rightarrow$ chase$(\Sigma, \mathcal{J})$ such that $\mu^-(\delta)$ is a finite set for all $\delta \in \Delta^{\mathcal{J}}$.

This can be shown by inductive construction of this extended homomorphism along the chase.

In particular, if chase$(\Sigma, \mathcal{J})$ is finite, so is chase$(\Sigma, \mathcal{I})$.

The claim follows since every instance has a (unique, obvious) homomorphism to $\mathcal{I}_\star$. □

Deciding universal termination?

Deciding universal oblivious chase termination could be easier than deciding chase termination on arbitrary instances:

- The reduction from DTM halting does not work, since the encoding is not correct on $\mathcal{I}_\star$.
- Indeed, in common DTM encodings, $\mathcal{I}_\star$ will fail to terminate, even if a particular input is accepted.

However, showing undecidability merely turns out to be much more work:

Theorem 19.13 (Gogacz & Marcinkowski 2014): Universal termination of the oblivious chase is undecidable.

As before, we obtain semi-decidability (just run the chase on the critical instance)

Sufficient conditions for termination

Undecidability of termination:
There is no algorithm that recognises exactly the cases where a chase terminates.

Can we still find algorithms that recognises many of the "relevant" cases?
⇝ Acyclicity conditions

Observation 19.14: If a set of tgds is non-recursive (in the sense that the predicate-dependency graph has no cycles), then the chase universally terminates.

One also speaks of acyclic tgds in this case.

Weak acyclicity

Acyclicity is very restrictive. A more liberal condition is based on tracing dependencies between predicate positions:

Definition 19.15: A predicate position is a pair $\langle p, i\rangle$ where p is a predicate symbol and $i \in \{1, \dots, \text{arity}(p)\}$. Given an atom $p(t_1, \dots, t_n)$, the term at position $\langle p, i\rangle$ is t_i. The dependency graph of a tgd set Σ has the set of all positions in predicates of Σ as its nodes.

For every rule ρ, and every variable x at position $\langle p, i\rangle$ in the head of ρ, the graph contains the following edges:

- If x is universally quantified and occurs at position $\langle q, j\rangle$ in the body of ρ, then there is an edge $\langle q, j\rangle \to \langle p, i\rangle$.
- If x is existentially quantified and another variable y occurs at position $\langle q, j\rangle$ in the body of ρ, then there is a special edge $\langle q, j\rangle \Rightarrow \langle p, i\rangle$.

Σ is weakly acyclic if its dependency graph does not contain a cycle that involves a special edge.

Weak acyclicity (2)

Example 19.16: The following tgd set is weakly acyclic yet recursive:

$$q(x) \to \exists v.p(x, v)$$
$$p(x, y) \to q(x)$$

Weak acyclicity is easy to check:

Theorem 19.17: Whether Σ is weakly acyclic is NL-complete.

Proof: The problem can be solved in NL using a variation of the usual NL reachability algorithm. It is NL-hard by an easy reduction from directed graph reachability. □

Chase termination under weak acyclicity

Theorem 19.18: If Σ is weakly acyclic then the oblivious (and the restricted) chase universally terminates over Σ.

To show this, we make a small change of perspective:

Definition 19.19: The naive skolemisation of a tgd $\forall \vec{x}, \vec{y}.\varphi[\vec{x}, \vec{y}] \rightarrow \exists \vec{z}.\psi[\vec{x}, \vec{z}]$ is obtained by replacing each existential variable $z \in \vec{z}$ with a function term $f_z(\vec{x}, \vec{y})$, where f_z is a fresh skolem function symbol of arity $|\vec{x}| + |\vec{y}|$.

The chase can be performed using the naive skolemisation of all tgds, and using function terms instead of labelled nulls. The result is isomorphic to the oblivious chase.

Proof (of Thm. 19.18): If the oblivious chase is infinite, then so is the skolem chase. Then the skolem chase uses infinitely many functional terms.

Hence there is a term of the form $f_z(t_1, \ldots, t_\ell)$ where f_z occurs nested within one of the terms t_i. It is easy to show that the sequence of rule applications that leads to such a term induces a cycle in the dependency graph through a special edge. □

Side remark: The Skolem chase

Using a less naive skolemisation improves the chase:

- For skolemisation, replace existential variables z by functional terms $f_z(\vec{x})$ that depend only on frontier variables $\vec{x}$
- Use the resulting skolem terms during the oblivious chase instead of named nulls
  $\rightsquigarrow$ skolem chase
- Weak acyclicity can be adapted to the skolem chase: in case 2, do not create special edges from positions of non-frontier variables

Example 19.20: Consider the tgd $p(x, y) \rightarrow \exists z.p(x, z)$. It is weakly acyclic with the modified definition, but not with our initial one. Over a fact $p(a, b)$, the oblivious chase would not terminate, but produce facts $p(a, b), p(a, n_1), p(a, n_2), \ldots$

In contrast, the skolem chase is based on the (non-naive) skolemisation $p(x, y) \rightarrow p(x, f(x))$, and therefore produces only the facts $p(a, b)$ and $p(a, f(a))$.

In general, the skolem chase terminates in more cases than the oblivious chase, and the restricted chase terminates in more cases than the skolem chase. All chases produce universal models.

Reasoning in weakly acyclic tgds

The chase can be used for reasoning as it will terminate.

However, this may still take a while:

Theorem 19.21 (Calí, Gottlob, Pieris 2010): The chase on a set Σ of weakly acyclic tgds may be of double exponential size in the size Σ. Deciding BCQ entailment over weakly acyclic tgds is 2ExpTime-complete for combined complexity and P-complete for data complexity.

Proof (sketch): Termination within doubly-exponential time (polynomial time for data complexity) can be obtained from the proof of termination (estimate maximal number of function terms without repeated symbols; estimate maximal number of unique facts that can be derived; conclude upper bound).

2ExpTime-hardness can be shown by simulating a doubly exponentially time bounded deterministic TM using weakly acyclic tgds. The key is to define a chain of doubly exponential length that can then be used for modelling time and space in a standard TM simulation (as seen in this course).

P-hardness follows from the P-hardness of full tgds (Datalog). □

Building long chains with weakly acyclic tgds

Key idea [Calí, Gottlob, Pieris 2010]: For $k \geq 0$, we construct a tape of length 2^{2^k} using a tgd set of size proportional to k. An initial chain of two elements is defined by the facts:

$$r_0(c_0), r_0(c_1), \mathsf{succ}_0(c_0, c_1), \mathsf{min}_0(c_0), \mathsf{max}_0(c_1).$$

The following rules are organised in layers $i \in \{0, \ldots, k-1\}$, where each layer combines already constructed elements to construct larger chains:

$$r_i(x) \wedge r_i(y) \rightarrow \exists z.s_i(x, y, z)$$
$$s_i(x, y, z) \rightarrow r_{i+1}(z)$$
$$s_i(x, y, z) \wedge s_i(x, y', z') \wedge \mathsf{succ}_i(y, y') \rightarrow \mathsf{succ}_{i+1}(z, z')$$
$$s_i(x, y, z) \wedge s_i(x', y', z') \wedge \mathsf{max}_i(y) \wedge \mathsf{min}_i(y') \wedge \mathsf{succ}_i(x, x') \rightarrow \mathsf{succ}_{i+1}(z, z')$$
$$\mathsf{min}_i(x) \wedge s_i(x, x, y) \rightarrow \mathsf{min}_{i+1}(y)$$
$$\mathsf{max}_i(x) \wedge s_i(x, x, y) \rightarrow \mathsf{max}_{i+1}(y)$$

Beyond weak acyclicity

Weak acyclicity is not perfect . . .

Example 19.22: The following tgd is not weakly acyclic:

$$q(x) \wedge p(x) \rightarrow \exists v.r(x, v) \wedge q(v)$$

Nevertheless, the oblivious chase terminates universally.

Intuitively, the example terminates since the new null for v does not jointly occur in $\langle p, 1\rangle$ and $\langle q, 1\rangle$.

Many generalisations of weak acyclicity were proposed:

- joint acyclicity (directly addressing the above)
- super-weak acyclicity
- acyclic graph of rule dependencies
- model-summarising and model-faithful acyclicity
- (and many less general conditions in specific areas)

Other acyclicities

Advanced acyclicity conditions are hard to check:

- P-complete: joint acyclicity and super-weak acyclicity
- NP-complete: acyclic graph of rule dependencies
- ExpTime-complete: model-summarising
- 2ExpTime-complete: model-faithful acyclicity

and they capture more and more tgd sets.

Nevertheless, all of these increasingly general notions lead to classes of tgds with the same expressivity (lossless transformation to weakly acyclic tgds possible)

Complexity of reasoning is also the same in all cases (2ExpTime combined and P data).

Restricted Chase Termination

Restricted chase termination

Many results also apply to the restricted chase:

- Termination on a specific instance is undecidable (same reduction)
- Query answering under common acyclicty conditions is 2ExpTime-complete (same reduction)

However, there is a new complication:

Example 19.23: Consider an instance $\mathcal{I}$ with fact $p(a, b)$ and the following tgds:

$$p(x, y) \rightarrow p(y, x) \qquad p(x, y) \rightarrow \exists v.p(y, v)$$

Applying the first tgd to $\mathcal{I}$, we obtain the set $\{p(a, b), p(b, a)\}$, which is a universal model. The restricted chase terminates.

However, if we apply the second tgd before the first, we can generate facts $p(a, b), p(b, n_1), p(b, a), p(n_1.n_2), p(n_1, b), p(n_2, n_3), \ldots$
The restricted chase does not terminate.

Observation: restricted chase termination depends on the order of rule applications

Universal restricted chase termination

The critical instance is not useful for universal termination of the restricted chase:

Observation 19.24: The restricted chase always terminates over the critical instance, for any set of tgds.

It seems to be open if universal restricted chase is semi-decidable or even decidable.

Nevertheless, there are sufficient criteria [Carral, Dragoste, K, 2017] – idea:

- Perform a chase based on the critical instance ...
- but weaken the restricted chase conditions on tgd applications:
  - When a tgd is applied, under-estimate a set of premises that the tgd is using in an arbitrary chase (not just in the critical-instance chase)
    (start from given body, rename different occurrences of constants apart, retrace provenance of nulls in premise)
  - Extend this set of premises by applying full tgds to saturation
  - Use the resulting set of facts to check if the rule should be applied

$\rightsquigarrow$ If this process can be shown to terminate, then the restrcited chase terminates under all instances and all tgd orders that prioritise full tgds

Summary and Outlook

The chase is an important approach for reasoning over dependencies

The oblivious chase introduces more redundant nulls than the restricted chase

Chase termination allows us to decide BCQ answering

Several sufficient acyclcity conditions can be used for showing chase termination

Next topics:

- More languages of tgds with decidable entailment problems
- Outlook