Open-World Finite Query Answering Under Number Restrictions

Antoine Amarilli[1,2]

[1]Télécom ParisTech, Paris, France

[2]University of Oxford, Oxford, United Kingdom

August 12, 2014

Open-world query answering

- Evaluate query q over instance I, open-world assumption:
  - The instance I is correct but incomplete
  - Consider all possible completions J satisfying constraints Σ
  - Certain answers to query q among those completions

  $\Rightarrow$ Formally: $I, \Sigma \models q$ if $J \models q$ for all $J \supseteq I$ s.t. $J \models \Sigma$

Open-world query answering

- Evaluate query q over instance I, open-world assumption:
  - The instance I is correct but incomplete
  - Consider all possible completions J satisfying constraints Σ
  - Certain answers to query q among those completions

  $\Rightarrow$ Formally: $I, \Sigma \models q$ if $J \models q$ for all $J \supseteq I$ s.t. $J \models \Sigma$
- Constraints:
  - TGDs, especially inclusion dependencies (ID)

    $\Rightarrow$ Unary inclusion dependencies (UID): $R[A] \subseteq S[B]$
  - Number restrictions, especially functional dependencies (FD)

Open-world query answering

- Evaluate query q over instance I, open-world assumption:
  - The instance I is correct but incomplete
  - Consider all possible completions J satisfying constraints Σ
  - Certain answers to query q among those completions

  $\Rightarrow$ Formally: $I, \Sigma \models q$ if $J \models q$ for all $J \supseteq I$ s.t. $J \models \Sigma$
- Constraints:
  - TGDs, especially inclusion dependencies (ID)

    $\Rightarrow$ Unary inclusion dependencies (UID): $R[A] \subseteq S[B]$
  - Number restrictions, especially functional dependencies (FD)
- Finite vs unrestricted QA

Instance:	List of employees
Constraint 1:	Each employee reviews some employee (UID)
Constraint 2:	At most one reviewer per employee (FD)
Query:	Are all employees reviewed?

Table of Contents

Undecidability barrier

- Entailment of IDs and FDs is undecidable [Mitchell, 1983]
- Already for binary IDs and unary FDs:
  $R[A, B] \subseteq S[C, D]$, $R[A] \rightarrow R[B]$

$\Rightarrow$ QA (finite or not) is also undecidable [Calì et al., 2003]
(Remark: this proof requires constants in the query)

Undecidability barrier

- Entailment of IDs and FDs is undecidable [Mitchell, 1983]
- Already for binary IDs and unary FDs:
  $R[A, B] \subseteq S[C, D]$, $R[A] \rightarrow R[B]$

$\Rightarrow$ QA (finite or not) is also undecidable [Calì et al., 2003]
(Remark: this proof requires constants in the query)

$\Rightarrow$ We can't have everything

Idea 1: Separability

- The chase for IDs: universal model
- Intuition: apply all IDs with fresh elements
- FDs are separable from IDs if they do not impact the chase
- Sufficient conditions for separability, e.g., non-conflicting:
  - $\Rightarrow$ exported positions must not be a strict superset of a key
- When separable, we can ignore FDs (just check them on I)

Idea 1: Separability

- The chase for IDs: universal model
- Intuition: apply all IDs with fresh elements
- FDs are separable from IDs if they do not impact the chase
- Sufficient conditions for separability, e.g., non-conflicting:
  - $\Rightarrow$ exported positions must not be a strict superset of a key
- When separable, we can ignore FDs (just check them on I)

$\Rightarrow$ The chase is infinite in general so it doesn't work in the finite

$\Rightarrow$ Finite QA undecidable for separable IDs/FDs [Rosati, 2006] (intuition: their finite consequences may not be separable)

Idea 2: Finite controllability

- Finite controllability means that finite and infinite QA coincide
- IDs are finitely controllable [Rosati, 2006]
  - ⇒ Construction: finite chase (chase with distant reuses)
- Generalizes to the guarded fragment [Barany et al., 2010]
  - ⇒ (Guarded means that $\forall/\exists$ must be covered by an atom)
  - ⇒ Intuition: query acyclification and cycle blowup
- Generalises to IDs/FDs with foreign keys condition [Rosati, 2006]

Idea 2: Finite controllability

- Finite controllability means that finite and infinite QA coincide
- IDs are finitely controllable [Rosati, 2006]
  - ⇒ Construction: finite chase (chase with distant reuses)
- Generalizes to the guarded fragment [Barany et al., 2010]
  - ⇒ (Guarded means that $\forall/\exists$ must be covered by an atom)
  - ⇒ Intuition: query acyclification and cycle blowup
- Generalises to IDs/FDs with foreign keys condition [Rosati, 2006]

⇒ FDs are not expressible in the guarded fragment.

⇒ IDs/FDs are not finitely controllable!

Idea 3: Arity-two

- Finite and unrestricted QA decidable in arity-two for the two-variable guarded fragment and counting constraints [Pratt-Hartmann, 2009]
  - Intuition: again, encode the acyclic part of the query
  - Satisfiability decidable by reduction to an inequation system
- Explicit construction for DLs [Ibáñez-García et al., 2014]

Idea 3: Arity-two

- Finite and unrestricted QA decidable in arity-two for the two-variable guarded fragment and counting constraints [Pratt-Hartmann, 2009]
    - Intuition: again, encode the acyclic part of the query
    - Satisfiability decidable by reduction to an inequation system
- Explicit construction for DLs [Ibáñez-García et al., 2014]

⇒ Only for arity-two signatures

⇒ No clear way to generalize to higher arity

Table of Contents

Our setting

- So:
  - Finite QA
  - TGDs and EGDs with interaction (not FC)
  - High-arity signatures
  - $\Rightarrow$ Can we have all three?

Our setting

- So:
  - Finite QA
  - TGDs and EGDs with interaction (not FC)
  - High-arity signatures
  - $\Rightarrow$ Can we have all three?

$\Rightarrow$ What if we restrict the language to UIDs and FDs?

Our setting

- So:
  - Finite QA
  - TGDs and EGDs with interaction (not FC)
  - High-arity signatures
  - ⇒ Can we have all three?
- ⇒ What if we restrict the language to UIDs and FDs?
  - No direct encoding to arity-two (unlike UIDs/UKDs...)
  - UIDs are important IDs in practice
  - UIDs match the DL intuition
  - UIDs are less expressive than BIDs
  - and...

Finite closure for UIDs and FDs

- Implication of UIDs/FDs is decidable and PTIME [Cosmadakis et al., 1990]
- Unrestricted and finite do not coincide

Finite closure for UIDs and FDs

- Implication of UIDs/FDs is decidable and PTIME [Cosmadakis et al., 1990]
- Unrestricted and finite do not coincide
- For unrestricted: implication of FDs and UIDs in isolation

Finite closure for UIDs and FDs

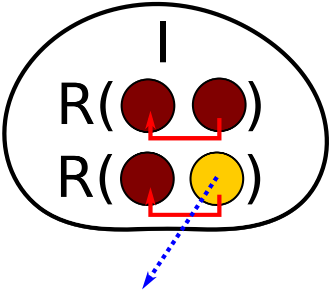
- Implication of UIDs/FDs is decidable and PTIME [Cosmadakis et al., 1990]
- Unrestricted and finite do not coincide
- For unrestricted: implication of FDs and UIDs in isolation
- For finite: add cycle reversal:
    - Consider only unary FDs: $R[i] \to S[j]$
    - When $R[i] \subseteq S[j]$ we have $|R[i]| \leq |S[j]|$
    - When $R[i] \to S[j]$ we have $|R[i]| \geq |S[j]|$
    - Inequality cycles with this encoding

Finite closure for UIDs and FDs

- Implication of UIDs/FDs is decidable and PTIME [Cosmadakis et al., 1990]
- Unrestricted and finite do not coincide
- For unrestricted: implication of FDs and UIDs in isolation
- For finite: add cycle reversal:
  - Consider only unary FDs: $R[i] \rightarrow S[j]$
  - When $R[i] \subseteq S[j]$ we have $|R[i]| \leq |S[j]|$
  - When $R[i] \rightarrow S[j]$ we have $|R[i]| \geq |S[j]|$
  - Inequality cycles with this encoding

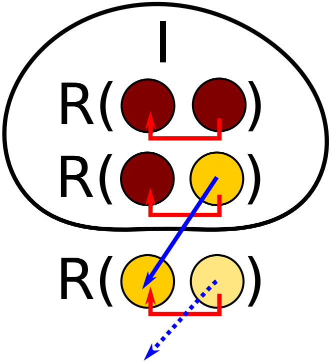
  $\Rightarrow$ In the finite, such cycles must be reversed

Finite closure example



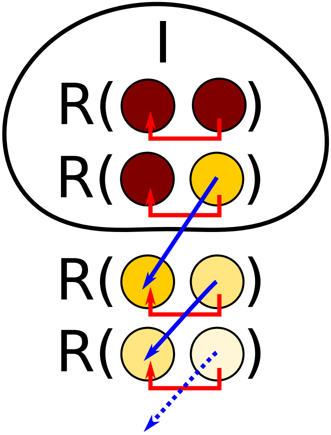
- $R[2] \subseteq R[1]$
- $R[2] \rightarrow R[1]$

Finite closure example



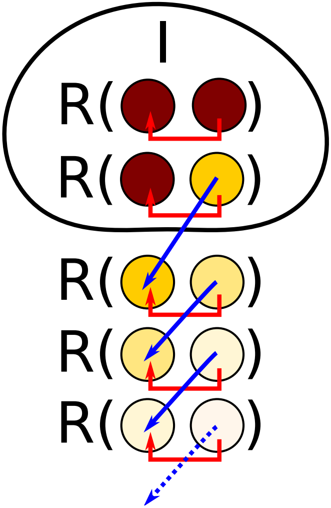
- $R[2] \subseteq R[1]$
- $R[2] \to R[1]$

Finite closure example



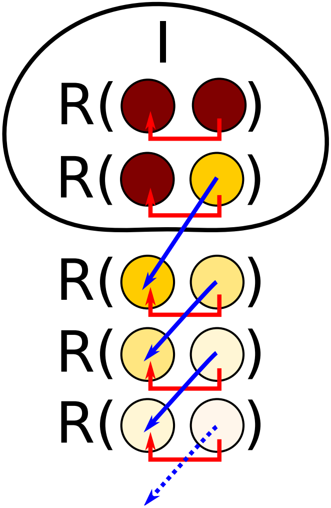
- $R[2] \subseteq R[1]$
- $R[2] \to R[1]$

Finite closure example



- $R[2] \subseteq R[1]$
- $R[2] \to R[1]$

Finite closure example

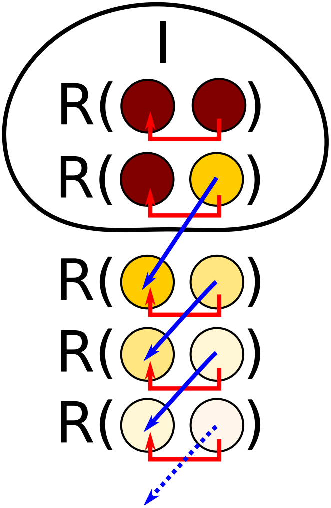


- $R[2] \subseteq R[1]$
- $R[2] \rightarrow R[1]$

$\Rightarrow |R[2]| \leq |R[1]|$

$\Rightarrow |R[1]| \leq |R[2]|$

Finite closure example



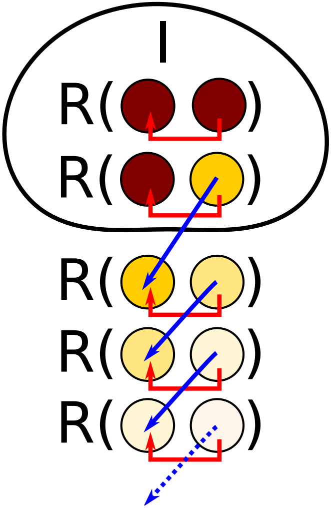
- $R[2] \subseteq R[1]$
- $R[2] \to R[1]$

$\Rightarrow$ $|R[2]| \leq |R[1]|$

$\Rightarrow$ $|R[1]| \leq |R[2]|$

$\Rightarrow$ $|R[2]| = |R[1]|$

Finite closure example



- $R[2] \subseteq R[1]$
- $R[2] \to R[1]$

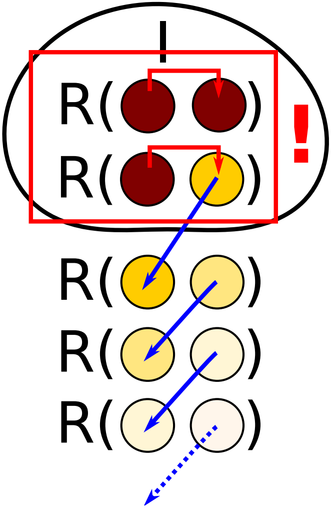
$\Rightarrow$ $|R[2]| \leq |R[1]|$

$\Rightarrow$ $|R[1]| \leq |R[2]|$

$\Rightarrow$ $|R[2]| = |R[1]|$

- Add $R[1] \subseteq R[2]$
- Add $R[1] \to R[2]$

Finite closure example



- $R[2] \subseteq R[1]$
- $R[2] \to R[1]$

$\Rightarrow$ $|R[2]| \leq |R[1]|$

$\Rightarrow$ $|R[1]| \leq |R[2]|$

$\Rightarrow$ $|R[2]| = |R[1]|$

- Add $R[1] \subseteq R[2]$
- Add $R[1] \to R[2]$

$\Rightarrow$ No finite model!

Finite controllability up to closure

- In arity-two, UIDs/UFDs finitely controllable up to finite closure [Rosati, 2008, Ibáñez-García et al., 2014]

$\Rightarrow$ To perform finite QA on instance I, UIDs/UFDs Σ:

  - Compute Σ^* the finite closure of Σ
  - Check if I satisfies the UFDs of Σ^*
  - Perform unrestricted QA with I and Σ^*
  - Easy because UIDs/UFDs are non-conflicting so separable

Finite controllability up to closure

- In arity-two, UIDs/UFDs finitely controllable up to finite closure [Rosati, 2008, Ibáñez-García et al., 2014]

$\Rightarrow$ To perform finite QA on instance I, UIDs/UFDs Σ:

  - Compute Σ^* the finite closure of Σ
  - Check if I satisfies the UFDs of Σ^*
  - Perform unrestricted QA with I and Σ^*
  - Easy because UIDs/UFDs are non-conflicting so separable

$\Rightarrow$ Does this also hold with higher-arity relations and FDs?

The result

Theorem

UIDs and FDs, though not finitely controllable, are finitely controllable up to finite closure, on arbitrary arity signatures.

The result

Theorem

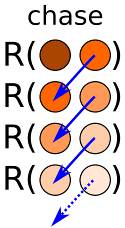
UIDs and FDs, though not finitely controllable, are finitely controllable up to finite closure, on arbitrary arity signatures.

$\Rightarrow$ It suffices to show that for any k, I, and Σ^*, there is a finite completion of I by Σ^* which is universal for queries of size $\leq k$

Table of Contents

Quotienting the chase

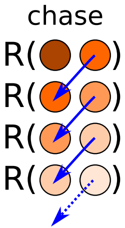
$R[2] \subseteq R[1]$



- Consider k-neighborhood equivalence

Quotienting the chase

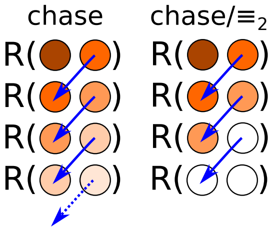
R[2] ⊆ R[1]



- Consider k-neighborhood equivalence
- Quotient the chase by this relation

Quotienting the chase

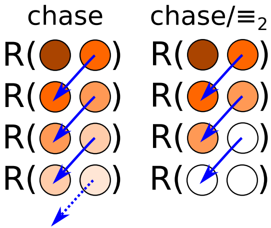
$R[2] \subseteq R[1]$



- Consider k-neighborhood equivalence
- Quotient the chase by this relation

Quotienting the chase

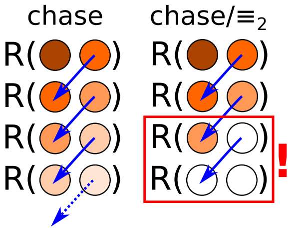
$R[2] \subseteq R[1]$



- Consider k-neighborhood equivalence
- Quotient the chase by this relation
- May violate FDs

Quotienting the chase

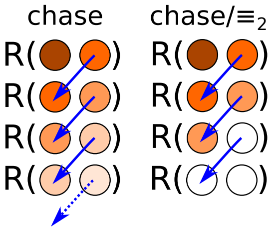
$R[2] \subseteq R[1]$



- Consider k-neighborhood equivalence
- Quotient the chase by this relation
- May violate FDs

Quotienting the chase

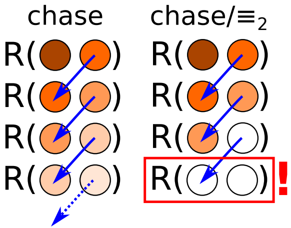
$R[2] \subseteq R[1]$



- Consider k-neighborhood equivalence
- Quotient the chase by this relation
- May violate FDs
- Not universal (cycles) even for $\leq k$

Quotienting the chase

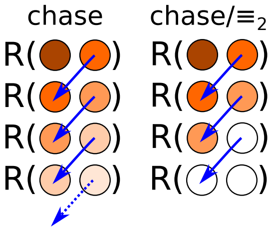
$R[2] \subseteq R[1]$



- Consider *k*-neighborhood equivalence
- Quotient the chase by this relation
- May violate FDs
- Not universal (cycles) even for $\leq k$

Quotienting the chase

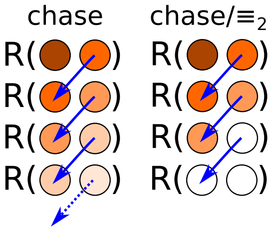
$R[2] \subseteq R[1]$



- Consider k-neighborhood equivalence
- Quotient the chase by this relation
- May violate FDs
- Not universal (cycles) even for $\leq k$
- Yet universal for $\leq k$ acyclic queries

Quotienting the chase

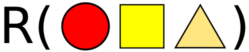
$R[2] \subseteq R[1]$



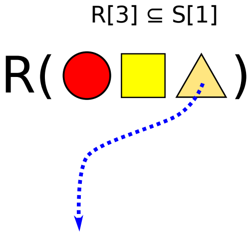
- Consider k-neighborhood equivalence
- Quotient the chase by this relation
- May violate FDs
- Not universal (cycles) even for $\leq k$
- Yet universal for $\leq k$ acyclic queries
- Keep a homomorphism to this quotient

Frugal chase steps

- Follow the chase

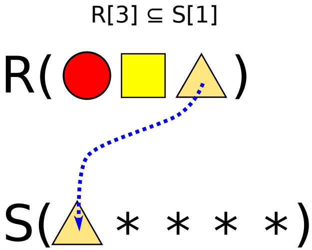


Frugal chase steps



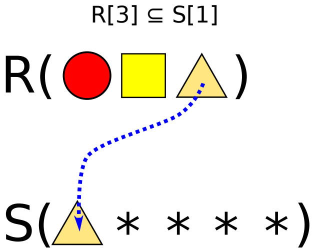
- Follow the chase

Frugal chase steps



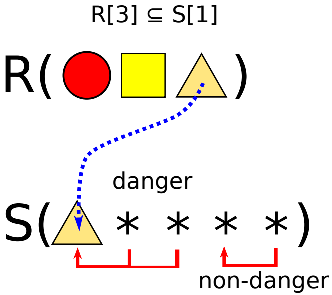
- Follow the chase

Frugal chase steps



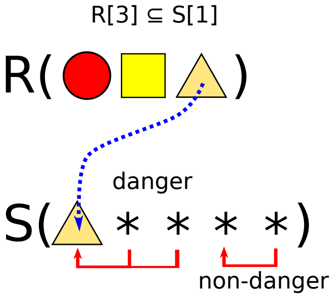
- Follow the chase
- Partition positions:
  - Exported position
  - Dangerous positions (determiners for an UFD)
  - Non-dangerous positions (the rest)

Frugal chase steps



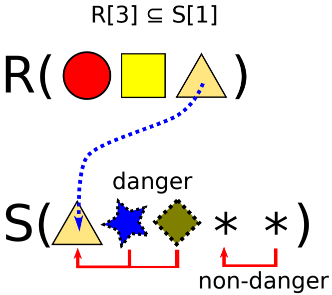
- Follow the chase
- Partition positions:
  - Exported position
  - Dangerous positions (determiners for an UFD)
  - Non-dangerous positions (the rest)

Frugal chase steps



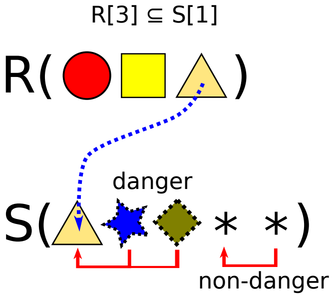
- Follow the chase
- Partition positions:
  - Exported position
  - Dangerous positions (determiners for an UFD)
  - Non-dangerous positions (the rest)
- Create fresh elements for dangerous

Frugal chase steps



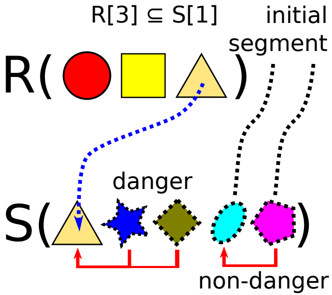
- Follow the chase
- Partition positions:
  - Exported position
  - Dangerous positions (determiners for an UFD)
  - Non-dangerous positions (the rest)
- Create fresh elements for dangerous

Frugal chase steps



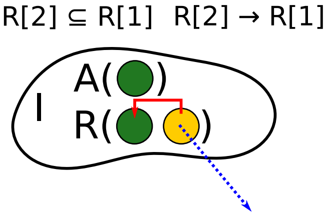
- Follow the chase
- Partition positions:
  - Exported position
  - Dangerous positions (determiners for an UFD)
  - Non-dangerous positions (the rest)
- Create fresh elements for dangerous
- Reuse clusters for non-dangerous created by initial chasing

Frugal chase steps



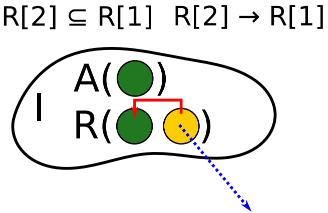
- Follow the chase
- Partition positions:
  - Exported position
  - Dangerous positions (determiners for an UFD)
  - Non-dangerous positions (the rest)
- Create fresh elements for dangerous
- Reuse clusters for non-dangerous created by initial chasing

Blueprint



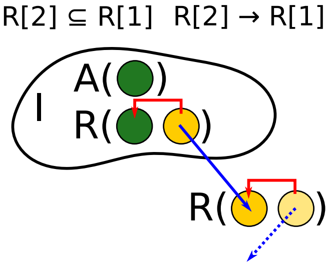
Blueprint

- Infinite functional paths...



Blueprint

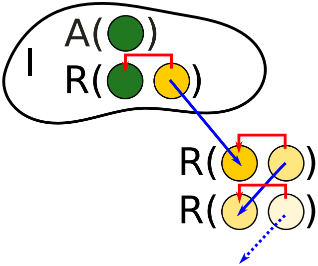
- Infinite functional paths…



Blueprint

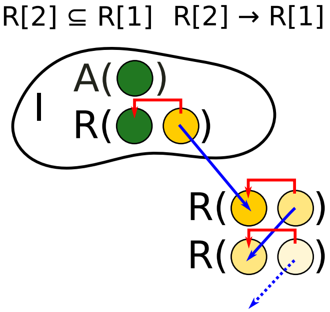
- Infinite functional paths...

$R[2] \subseteq R[1]$ $R[2] \rightarrow R[1]$



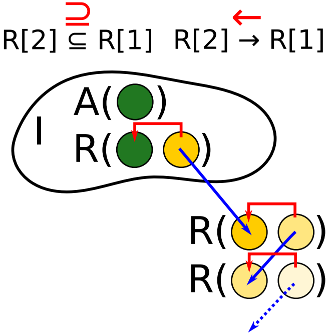
Blueprint

- Infinite functional paths...
- ... but only within a cycle



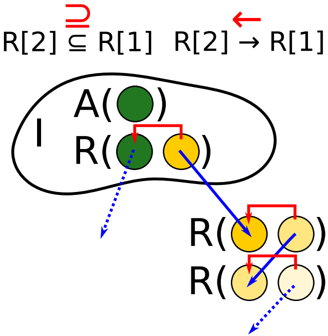
Blueprint

- Infinite functional paths…
- … but only within a cycle



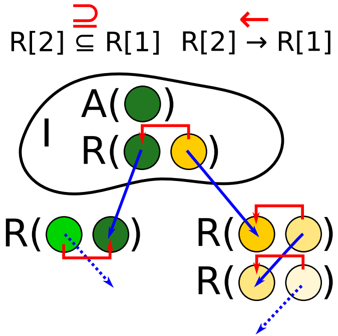
Blueprint

- Infinite functional paths...
- ... but only within a cycle



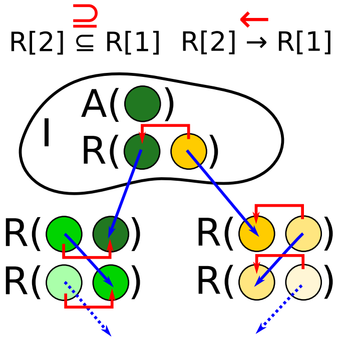
Blueprint

- Infinite functional paths...
- ... but only within a cycle



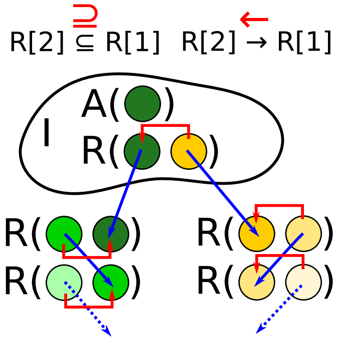
Blueprint

- Infinite functional paths...
- ... but only within a cycle



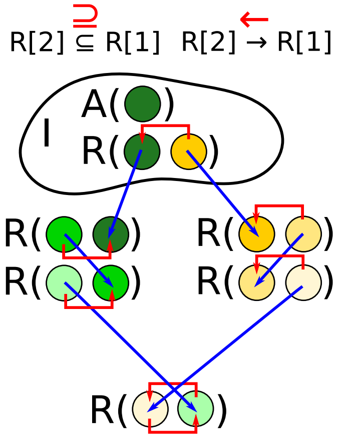
Blueprint

- Infinite functional paths...
- ... but only within a cycle
- Connect it back (match elements)



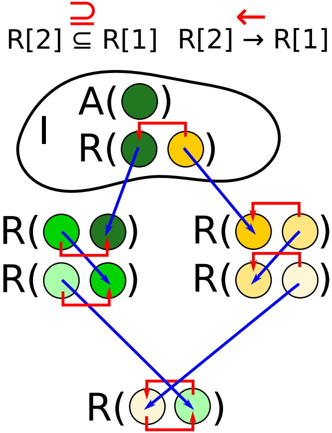
Blueprint

- Infinite functional paths...
- ... but only within a cycle
- Connect it back (match elements)



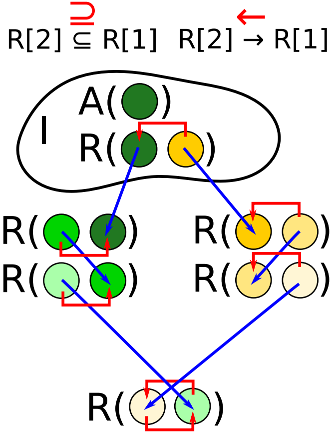
Blueprint

- Infinite functional paths...
- ... but only within a cycle
- Connect it back (match elements)
- More complex if many positions of many relations are involved...



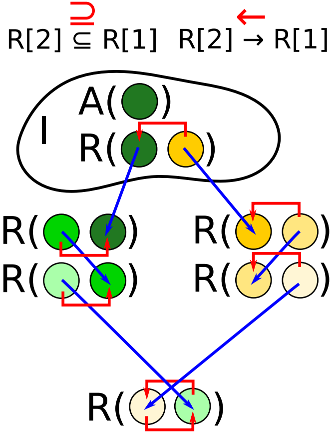
Blueprint

- Infinite functional paths...
- ... but only within a cycle
- Connect it back (match elements)
- More complex if many positions of many relations are involved...
- Uses cardinality along cycles...



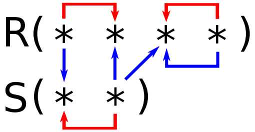
Blueprint

- Infinite functional paths...
- ... but only within a cycle
- Connect it back (match elements)
- More complex if many positions of many relations are involved...
- Uses cardinality along cycles...
- ... but also initial chasing to force "generic neighborhoods"

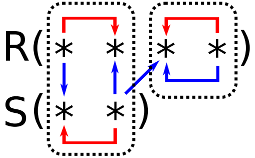


Dependency graph

- Build a DAG on the dependency cycles

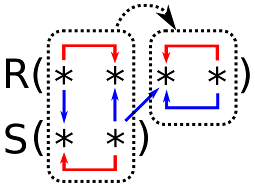


Dependency graph



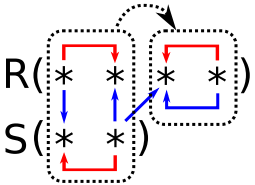
- Build a DAG on the dependency cycles

Dependency graph



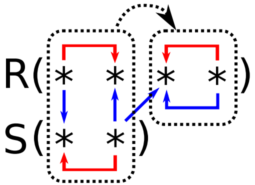
- Build a DAG on the dependency cycles

Dependency graph



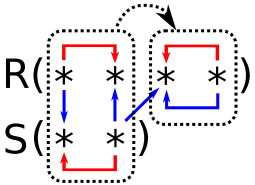
- Build a DAG on the dependency cycles
- We never create fresh elements
  for a higher dependency in the DAG

Dependency graph



- Build a DAG on the dependency cycles
- We never create fresh elements
  for a higher dependency in the DAG
- Satisfy cycles along a topological sort

Dependency graph

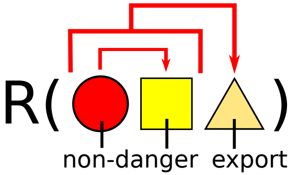


- Build a DAG on the dependency cycles
- We never create fresh elements for a higher dependency in the DAG
- Satisfy cycles along a topological sort

$\Rightarrow$ Finite extension that satisfies UIDs/UFDs with a homomorphism to the quotient

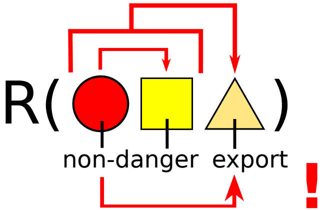
Higher-arity FDs

- Ignored so far
- May only be triggered at non-dangerous reuses
- Idea: if non-dangerous but dangerous for higher-arity FD then no unary key



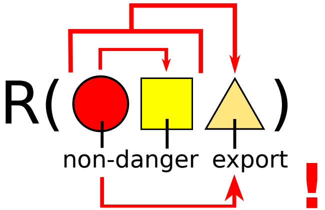
Higher-arity FDs

- Ignored so far
- May only be triggered at non-dangerous reuses
- Idea: if non-dangerous but dangerous for higher-arity FD then no unary key



Higher-arity FDs

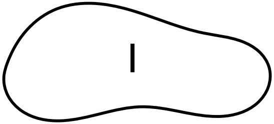
- Ignored so far
- May only be triggered at non-dangerous reuses
- Idea: if non-dangerous but dangerous for higher-arity FD then no unary key



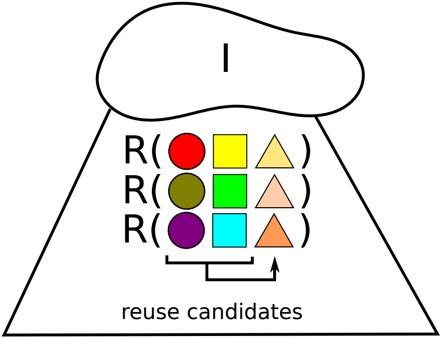
- Idea:
  - create many reuse candidates
  - combine them in different patterns

⇒ Lemma: if no UKD then $O(n^{>1})$ patterns for $O(n)$ elements

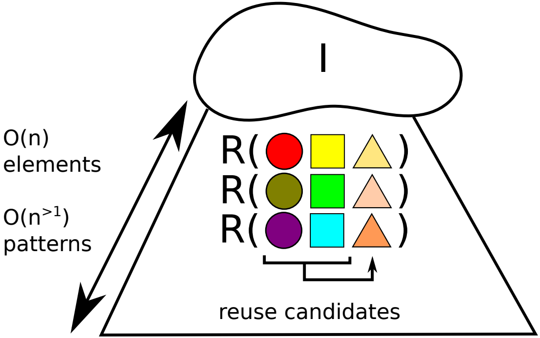
Higher-arity FDs



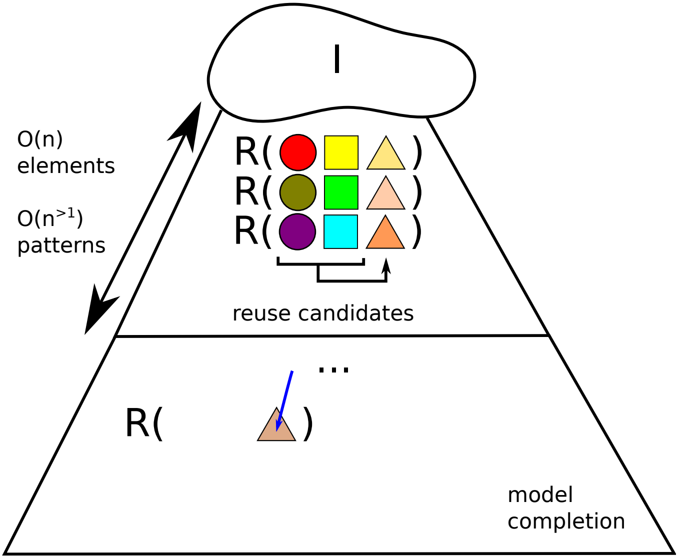
Higher-arity FDs



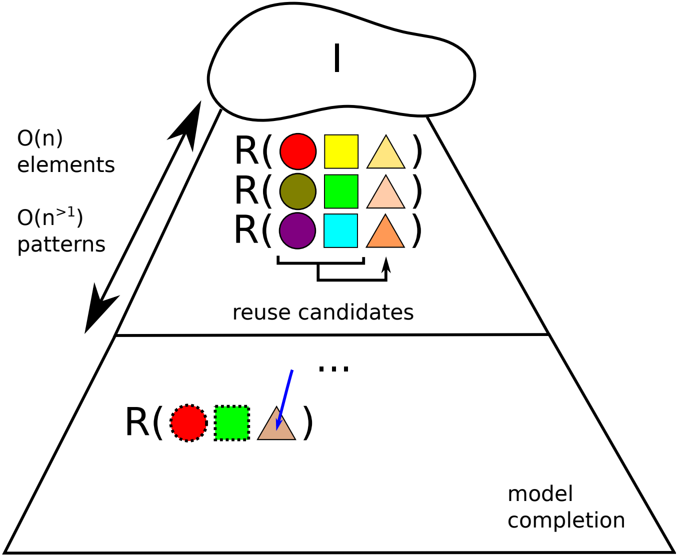
Higher-arity FDs



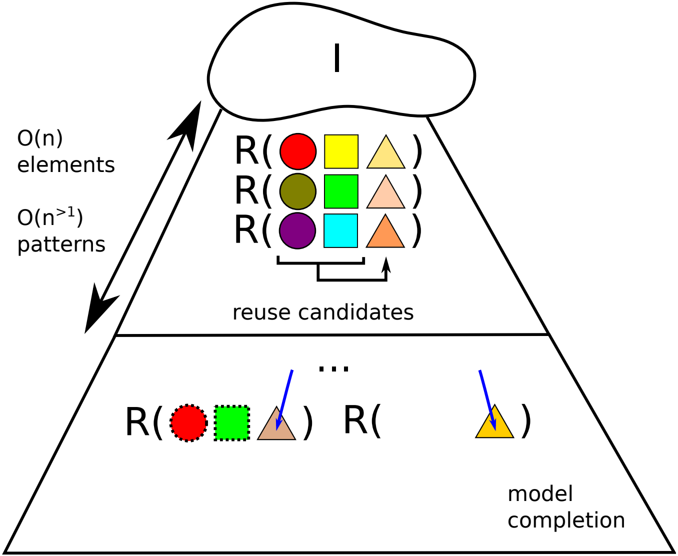
Higher-arity FDs



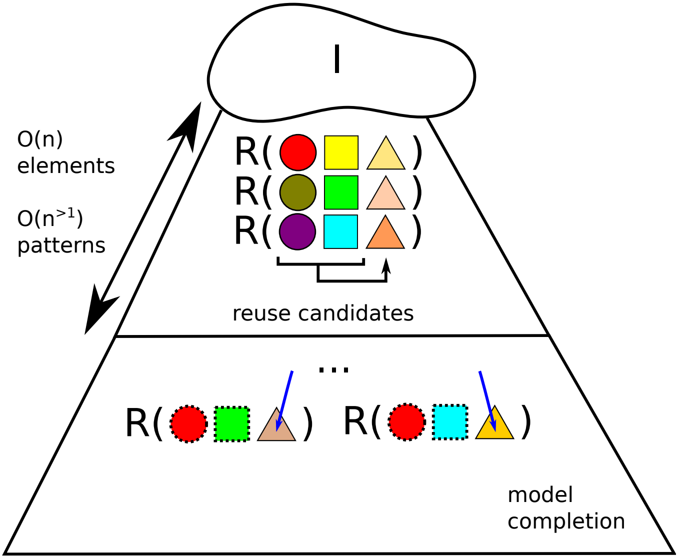
Higher-arity FDs



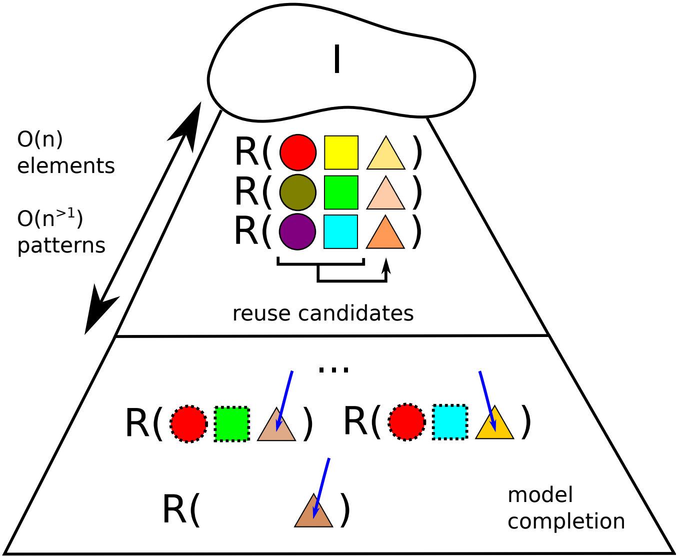
Higher-arity FDs



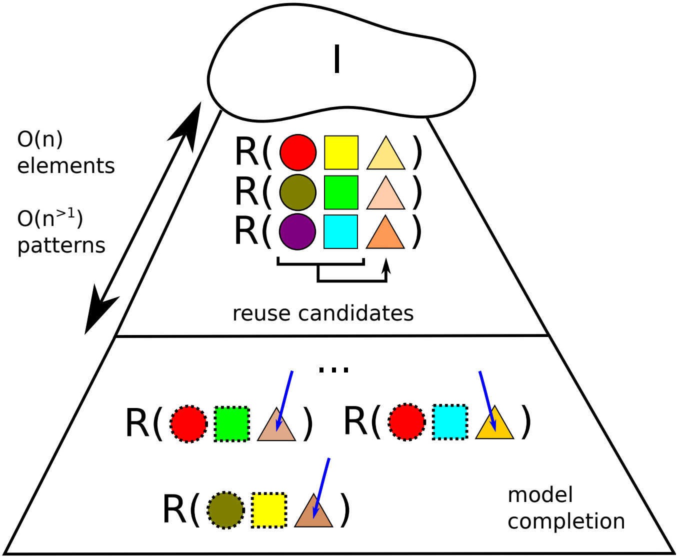
Higher-arity FDs



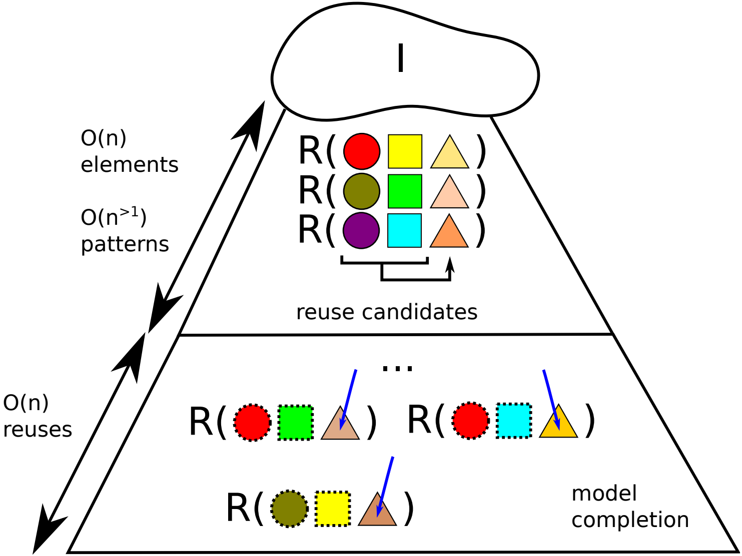
Higher-arity FDs



Higher-arity FDs



Higher-arity FDs

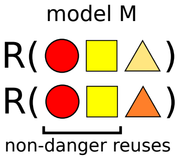


Blowing up cycles

- Usually: product with a group of high girth [Otto, 2002]
- Ensures all non-instance cycles are large
- We cannot do it directly as it would violate the FDs

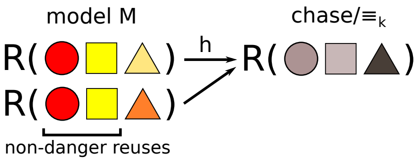
Blowing up cycles

- Usually: product with a group of high girth [Otto, 2002]
- Ensures all non-instance cycles are large
- We cannot do it directly as it would violate the FDs
- However, intuition:
  - FD violations can only appear on non-dangerous reuses...
  - ... and those facts are mapped to the same quotient fact
  - ... so cycles on them are self-homomorphic in the quotient



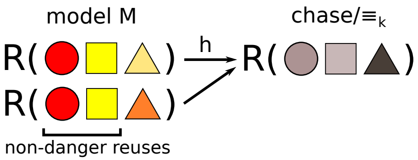
Blowing up cycles

- Usually: product with a group of high girth [Otto, 2002]
- Ensures all non-instance cycles are large
- We cannot do it directly as it would violate the FDs
- However, intuition:
  - FD violations can only appear on non-dangerous reuses...
  - ... and those facts are mapped to the same quotient fact
  - ... so cycles on them are self-homomorphic in the quotient



Blowing up cycles

- Usually: product with a group of high girth [Otto, 2002]
- Ensures all non-instance cycles are large
- We cannot do it directly as it would violate the FDs
- However, intuition:
  - FD violations can only appear on non-dangerous reuses...
  - ... and those facts are mapped to the same quotient fact
  - ... so cycles on them are self-homomorphic in the quotient



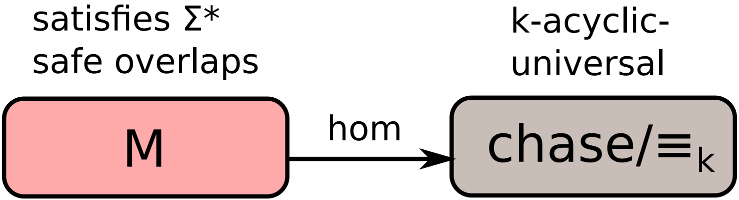
⇒ Blow up cycles, but not those cycles

Blowing up cycles via the product

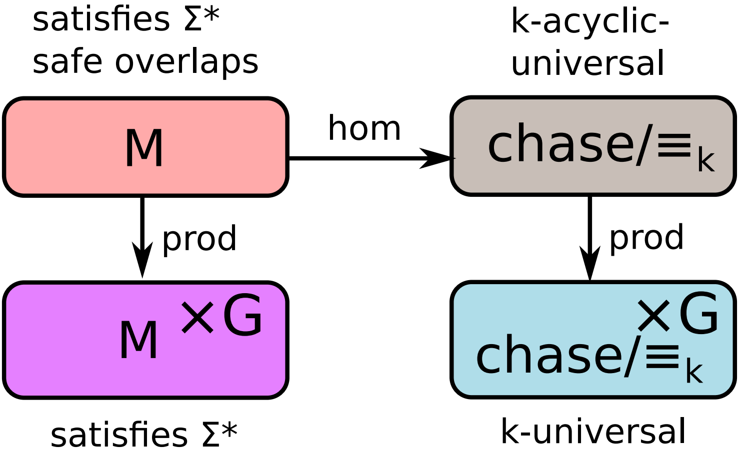
k-acyclic-universal

$\boxed{\text{chase}/\equiv_k}$

Blowing up cycles via the product



Blowing up cycles via the product



Blowing up cycles via the product

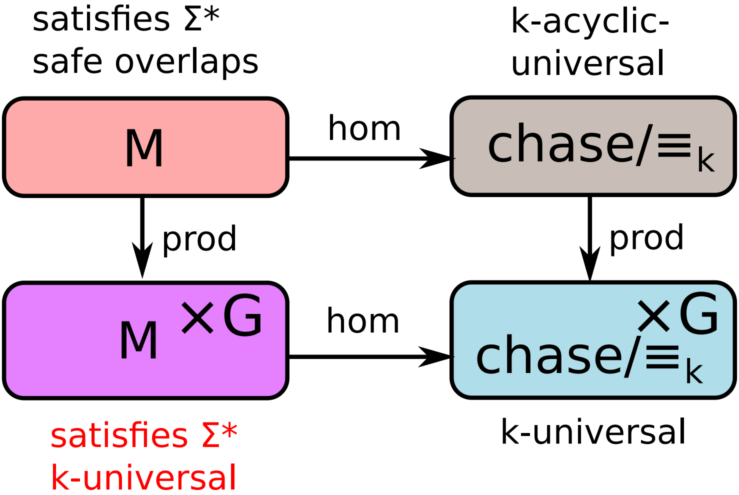


Table of Contents

Summary

- UIDs/FDs finitely controllable up to closure
- Main differences with arity-two:
  - Clusters for non-dangerous reuses
  - Combinations for higher-arity FDs
  - More complex reconnexions along cycles
  - More elaborate cycle elimination (via the quotient)

$\Rightarrow$ Generalize to richer unary languages for high arity?

  - Need construction for finite implication
  - Does the finite model construction adapt?

Summary

- UIDs/FDs finitely controllable up to closure
- Main differences with arity-two:
  - Clusters for non-dangerous reuses
  - Combinations for higher-arity FDs
  - More complex reconnexions along cycles
  - More elaborate cycle elimination (via the quotient)

$\Rightarrow$ Generalize to richer unary languages for high arity?

  - Need construction for finite implication
  - Does the finite model construction adapt?

Thanks for your attention!

References I

- Barany, V., Gottlob, G., and Otto, M. (2010).
  Querying the guarded fragment.
  In *LICS*.
- Calì, A., Lembo, D., and Rosati, R. (2003).
  On the decidability and complexity of query answering over inconsistent and incomplete databases.
  In *PODS*.
- Cosmadakis, S. S., Kanellakis, P. C., and Vardi, M. Y. (1990).
  Polynomial-time implication problems for unary inclusion dependencies.
  JACM, 37(1).

References II

Ibáñez-García, Y., Lutz, C., and Schneider, T. (2014).
Finite model reasoning in horn description logics.
In *KR*.

Mitchell, J. C. (1983).
The implication problem for functional and inclusion dependencies.
Information and Control, 56(3).

Otto, M. (2002).
Modal and guarded characterisation theorems over finite transition systems.
In *LICS*.

References III

Pratt-Hartmann, I. (2009).
Data-complexity of the two-variable fragment with counting quantifiers.
Inf. Comput., 207(8).

Rosati, R. (2006).
On the decidability and finite controllability of query processing in databases with incomplete information.
In *PODS*.

Rosati, R. (2008).
Finite model reasoning in DL-Lite.
In *ESWC*.