The Sticky Path to Expressive Querying:

Decidability of Navigational Queries under Existential Rules

Piotr Ostropolski-Nalewaja[1,2] and Sebastian Rudolph[1]

[1] TU Dresden [2] University of Wrocław

Introduction

Basic Setup

In this talk, we are going to discuss:

Instances

Relational structures

Basic Setup

In this talk, we are going to discuss:

Instances

Relational structures

Queries

Regular path queries

Basic Setup

In this talk, we are going to discuss:

Instances

Relational structures

Queries

Regular path queries

Constraints

Used to restrict instances in the form of *existential rules*

Basic Setup

In this talk, we are going to discuss:

Instances

Relational structures

Queries

Regular path queries

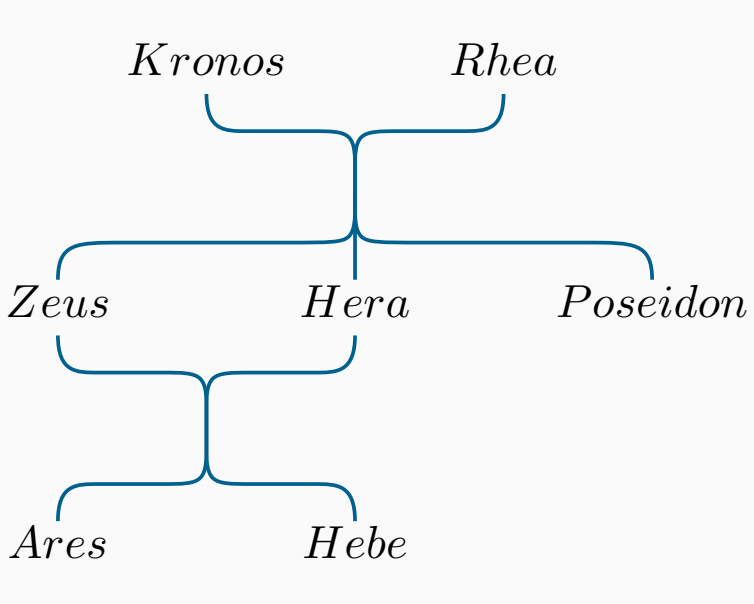
Constraints

Used to restrict instances in the form of *existential rules*

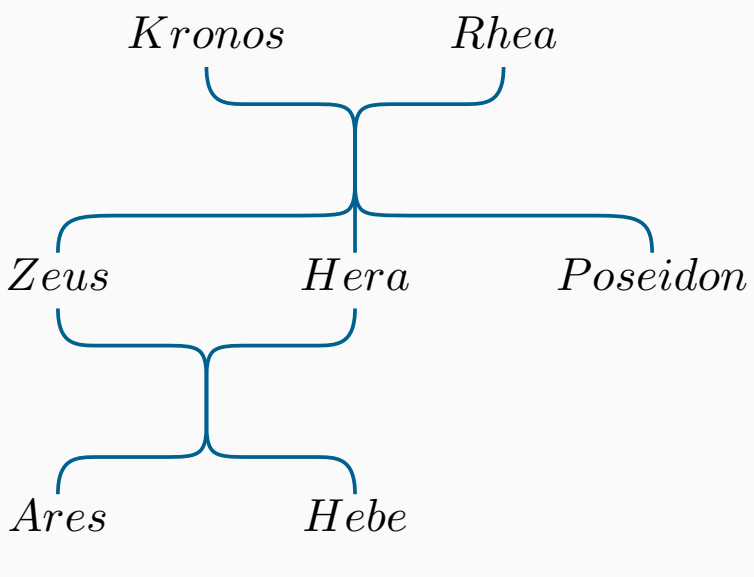
Entailment

Of Regular Path Queries under existential rules

Logic Based Knowledge Representation

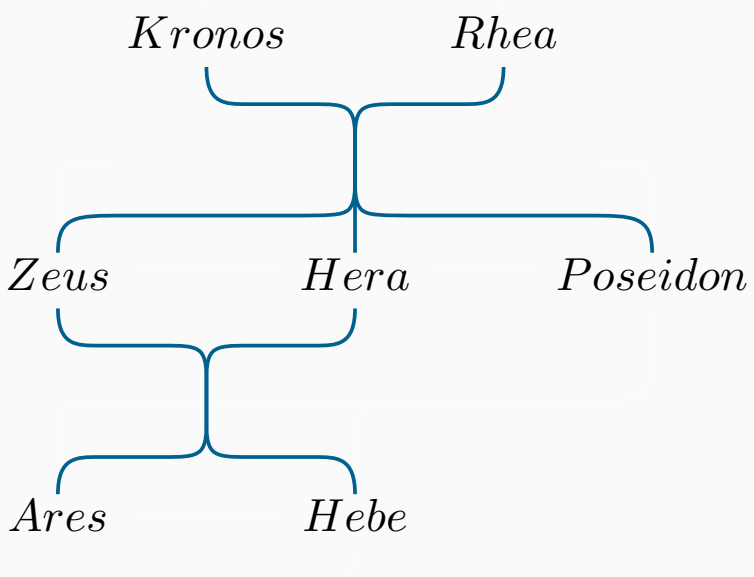


Logic Based Knowledge Representation



Usually we know more:

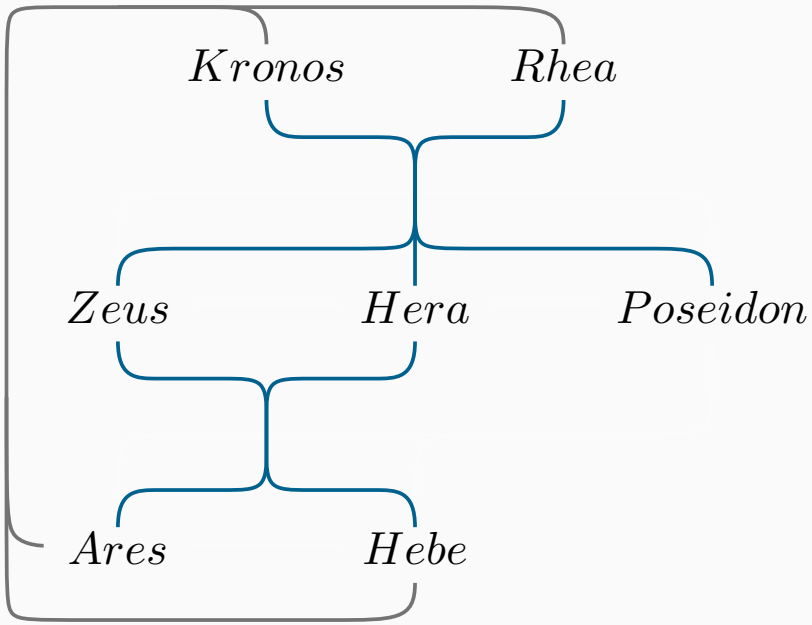
Logic Based Knowledge Representation



Usually we know more:

Grandparents

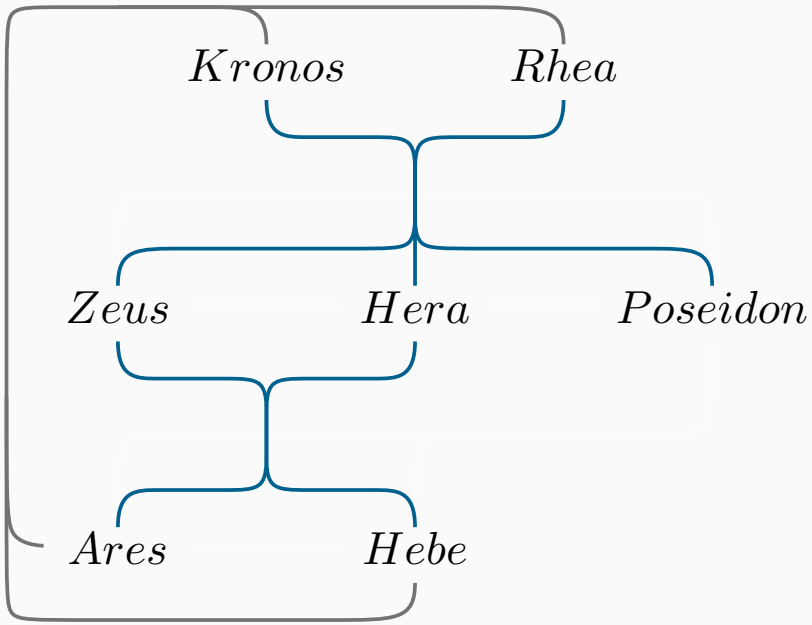
Logic Based Knowledge Representation



Usually we know more:

Grandparents

Logic Based Knowledge Representation

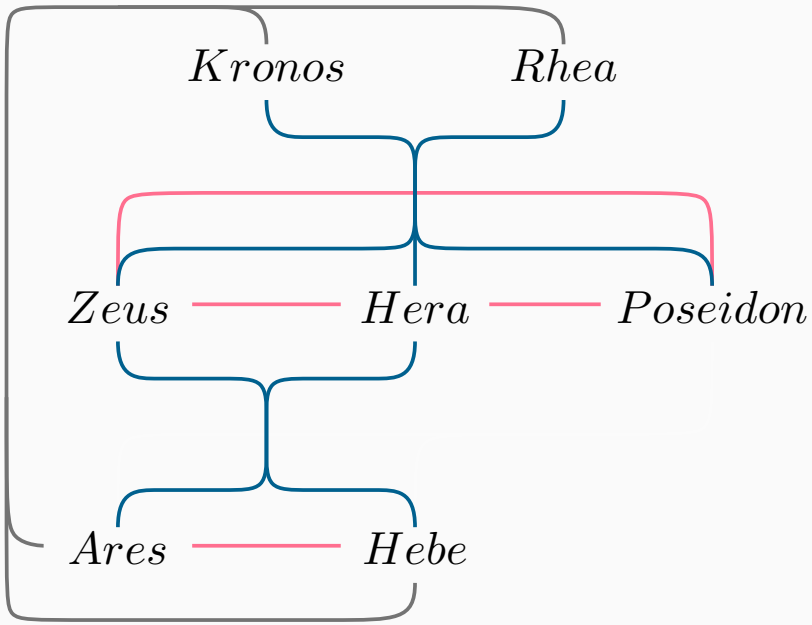


Usually we know more:

Grandparents

Siblings

Logic Based Knowledge Representation

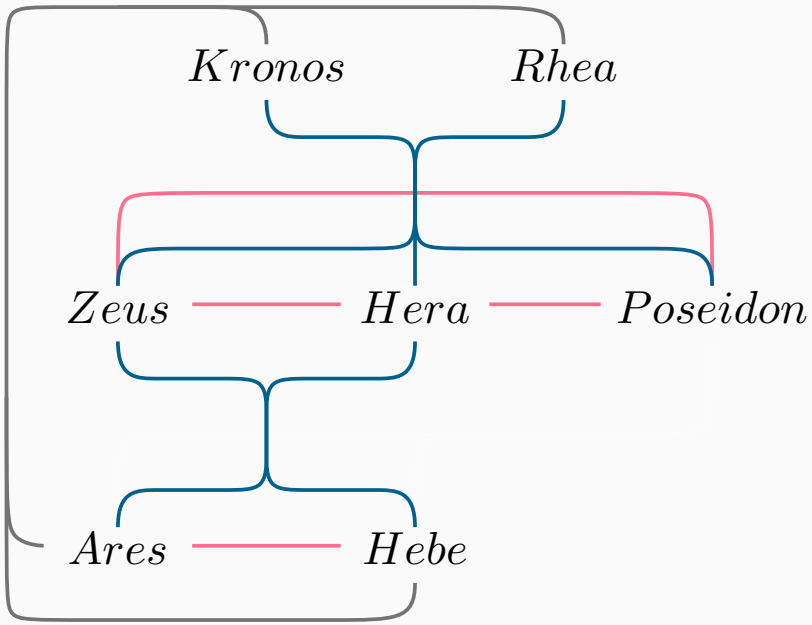


Usually we know more:

Grandparents

Siblings

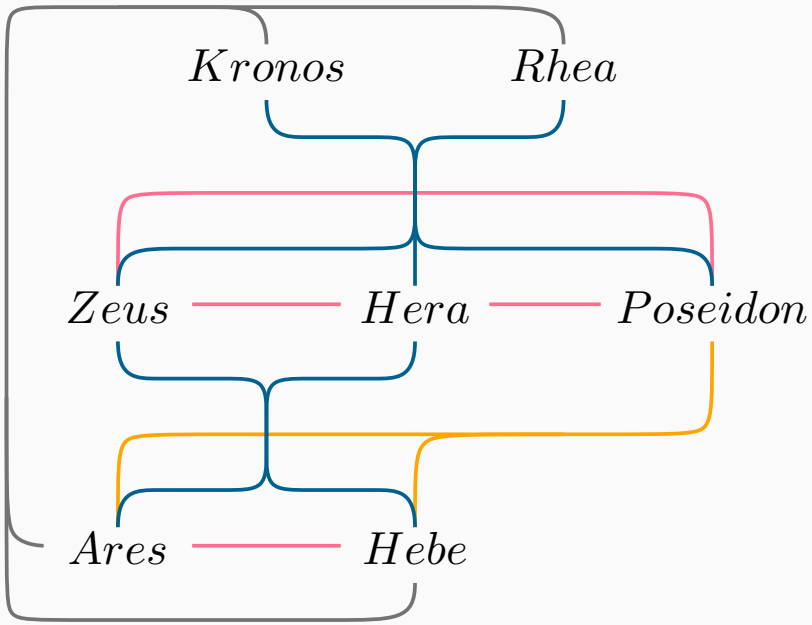
Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles

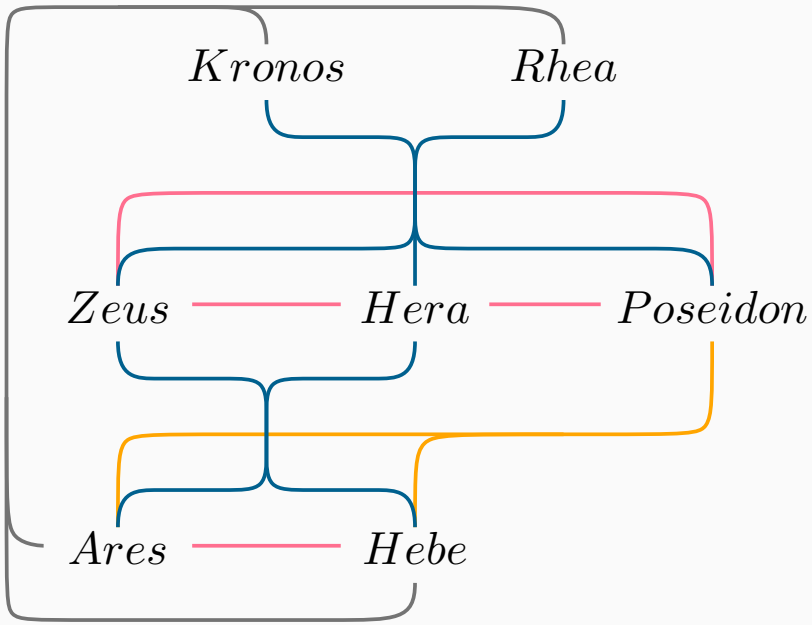
Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles

Logic Based Knowledge Representation

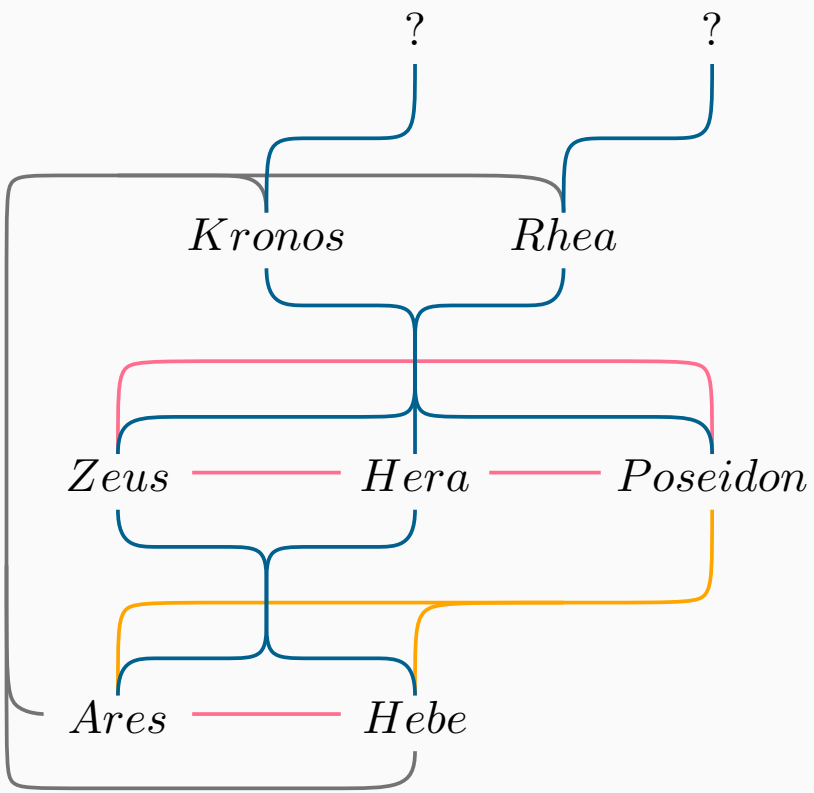


Usually we know more:

- Grandparents
- Siblings
- Uncles

Everyone has a mother

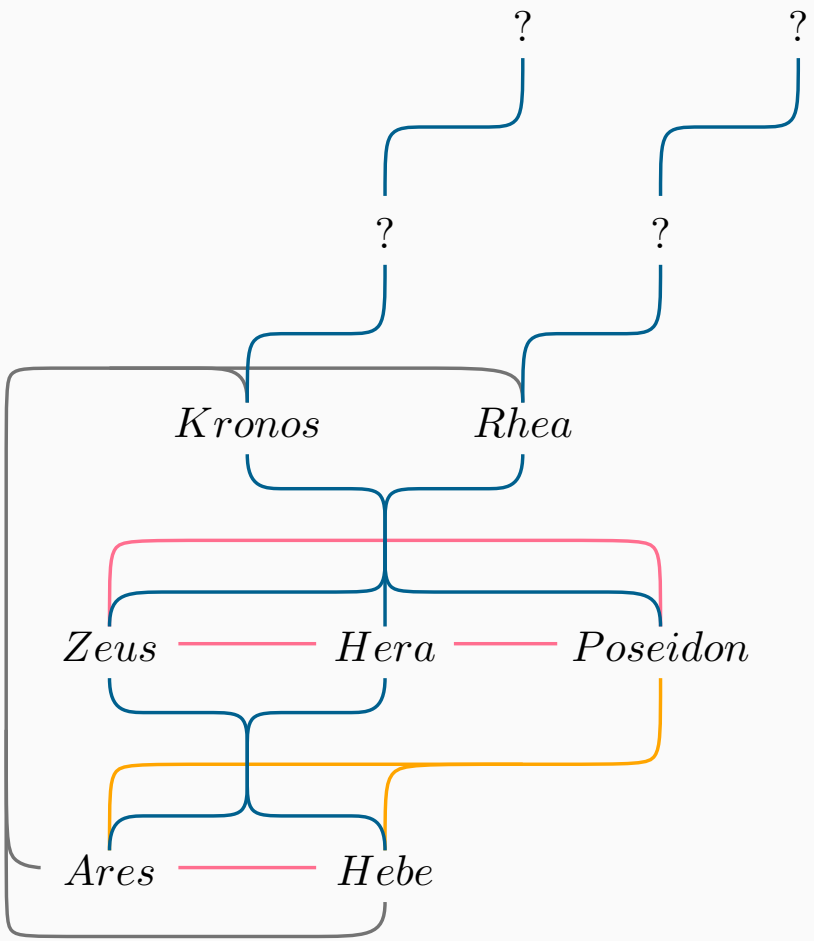
Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles
- Everyone has a mother

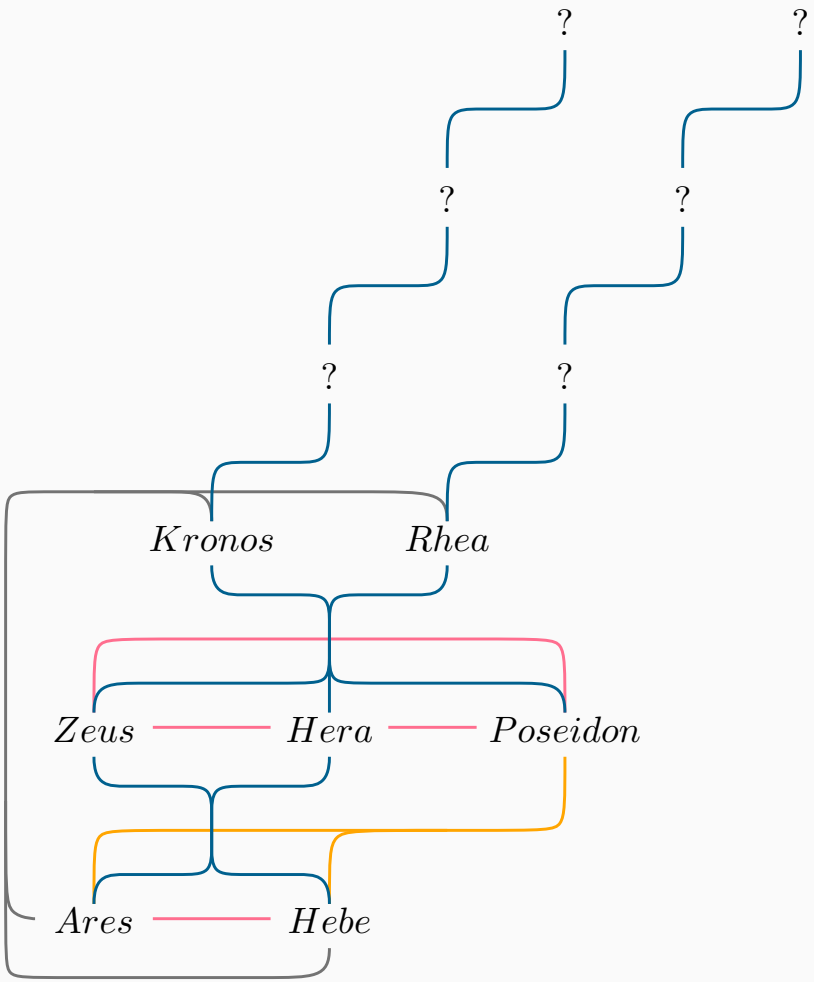
Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles
- Everyone has a mother

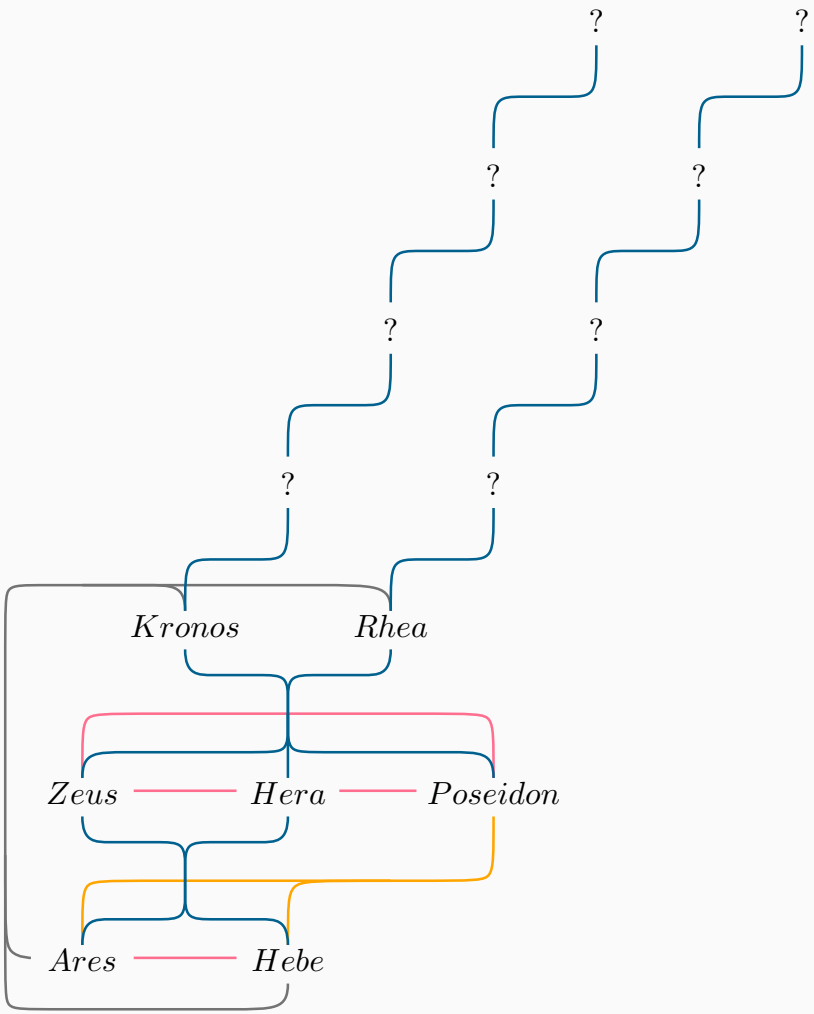
Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles
- Everyone has a mother

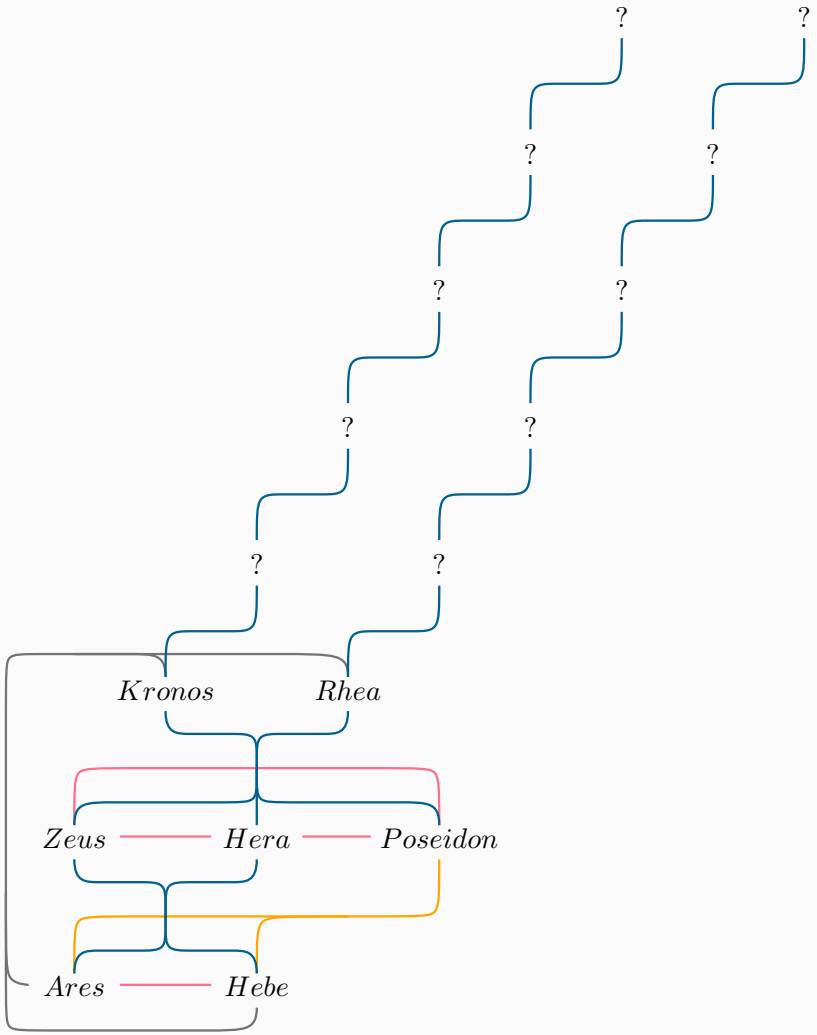
Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles
- Everyone has a mother

Logic Based Knowledge Representation



Usually we know more:

- Grandparents
- Siblings
- Uncles
- Everyone has a mother

General Question

In general, we are interested about *entailment*

$$Database,\ Knowledge \models Query$$

General Question

In general, we are interested about *entailment*

$$Database,\ Knowledge \models Query$$

Example ontology languages (Knowledge):

- First/Second Order Logic (and numerous fragments),
- Modal/Description Logics.

General Question

In general, we are interested about *entailment*

$$Database,\ Knowledge \models Query$$

Example ontology languages (Knowledge):

- First/Second Order Logic (and numerous fragments),
- Modal/Description Logics.

Example query languages:

- Conjunctive Queries (SQL),
- Regular Path Queries (Cypher, Gremlin and SPARQL),
- Datalog.

Introduction: Existential Rules

Existential Rules

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

$$\forall \bar{x}\bar{y} \;\; \alpha(\bar{x}, \bar{y}) \quad \rightarrow \quad \exists \bar{z} \;\; \beta(\bar{y}, \bar{z}).$$

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

$$\forall \bar{x}\bar{y}\ \alpha(\bar{x},\bar{y}) \rightarrow \exists \bar{z}\ \beta(\bar{y},\bar{z}).$$

Body

Head

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

$$\forall \bar{x}\bar{y} \;\; \alpha(\bar{x}, \bar{y}) \quad \rightarrow \quad \exists \bar{z} \;\; \beta(\bar{y}, \bar{z}).$$

Everyone has a mother

$$\texttt{Human}(x) \rightarrow \exists y \;\; \texttt{Mother}(x, y)$$
$$\texttt{Mother}(x, y) \rightarrow \texttt{Human}(y)$$

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

$$\forall \bar{x}\bar{y} \;\; \alpha(\bar{x}, \bar{y}) \quad \rightarrow \quad \exists \bar{z} \;\; \beta(\bar{y}, \bar{z}).$$

Everyone has a mother

$$\texttt{Human}(x) \rightarrow \exists y \;\; \texttt{Mother}(x, y)$$
$$\texttt{Mother}(x, y) \rightarrow \texttt{Human}(y)$$

R is transitive

$$\texttt{R}(x, y), \texttt{R}(y, z) \rightarrow \texttt{R}(x, z)$$

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

$$\forall \bar{x}\bar{y} \;\; \alpha(\bar{x}, \bar{y}) \quad \rightarrow \quad \exists \bar{z} \;\; \beta(\bar{y}, \bar{z}).$$

Everyone has a mother

$$\texttt{Human}(x) \rightarrow \exists y \;\; \texttt{Mother}(x, y)$$
$$\texttt{Mother}(x, y) \rightarrow \texttt{Human}(y)$$

R is transitive

$$\texttt{R}(x, y), \texttt{R}(y, z) \rightarrow \texttt{R}(x, z)$$

Everything is connected

$$\forall xy \quad \rightarrow \texttt{R}(x, y)$$

Existential Rules

Existential Rule

An *existential rule* is an FO formula of the form:

$$\forall \bar{x}\bar{y} \;\; \alpha(\bar{x}, \bar{y}) \quad \rightarrow \quad \exists \bar{z} \;\; \beta(\bar{y}, \bar{z}).$$

Everyone has a mother

$$\texttt{Human}(x) \rightarrow \exists y \;\; \texttt{Mother}(x, y)$$
$$\texttt{Mother}(x, y) \rightarrow \texttt{Human}(y)$$

R is transitive

$$\texttt{R}(x, y), \texttt{R}(y, z) \rightarrow \texttt{R}(x, z)$$

Everything is connected

$$\forall xy \quad \rightarrow \texttt{R}(x, y)$$

Models are gridlike

$$\texttt{H}(x, y) \rightarrow \exists z \;\; \texttt{H}(y, z)$$
$$\texttt{H}(x, y) \rightarrow \exists x' \;\; \texttt{V}(x, x')$$
$$\texttt{H}(x, y), \;\; \texttt{V}(x, x'), \;\; \texttt{V}(y, y') \rightarrow \texttt{H}(x', y')$$

Universal models

Usually, given a database $\mathcal{D}$,

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$,

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

Quite Difficult :(

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

For existential rules there always exists a *universal model*.

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

For existential rules there always exists a *universal model*. Such model $\mathcal{U}$ homomorphically maps into every model of the database $\mathcal{D}$ and the ruleset $\mathcal{R}$!

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

For existential rules there always exists a *universal model*. Such model $\mathcal{U}$ homomorphically maps into every model of the database $\mathcal{D}$ and the ruleset $\mathcal{R}$!

Entailment, existential rules

$$\mathcal{D}, \mathcal{R} \models Q \iff Q \text{ holds in } \mathcal{U}.$$

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

For existential rules there always exists a *universal model*. Such model $\mathcal{U}$ homomorphically maps into every model of the database $\mathcal{D}$ and the ruleset $\mathcal{R}$!

Entailment, existential rules

$$\mathcal{D}, \mathcal{R} \models Q \iff Q \text{ holds in } \mathcal{U}.$$

Much better :)

Universal models

Usually, given a database $\mathcal{D}$, a logical theory $\mathcal{T}$, and query Q the *entailment* is defined as:

Entailment, general case

$$\mathcal{D}, \mathcal{T} \models Q \iff \bigwedge_{\mathcal{I} \text{ is model of } \mathcal{D},\mathcal{T}} Q \text{ holds in } \mathcal{I}.$$

For existential rules there always exists a *universal model*. Such model $\mathcal{U}$ homomorphically maps into every model of the database $\mathcal{D}$ and the ruleset $\mathcal{R}$!

Entailment, existential rules

$$\mathcal{D}, \mathcal{R} \models Q \iff Q \text{ holds in } \mathcal{U}.$$

Universal models can be computed with *the chase* algorithm!

Chase example I

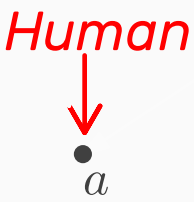
$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \;\; \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$

Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \;\; \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$

Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \;\; \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$

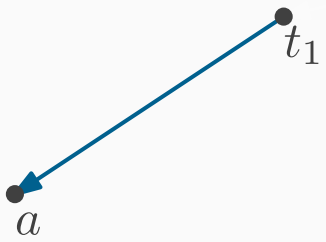


Chase example I

$$\texttt{Human}(x) \rightarrow \exists y \ \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$

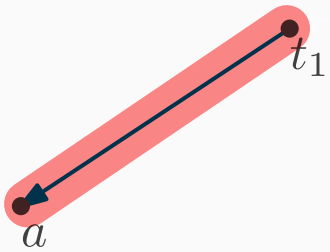
Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \ \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$



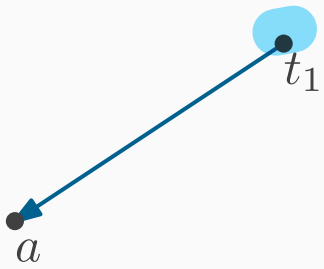
Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \ \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$



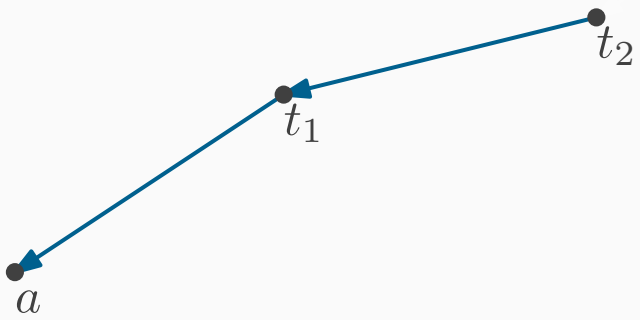
Chase example I

$$\texttt{Human}(x) \rightarrow \exists y\ \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$



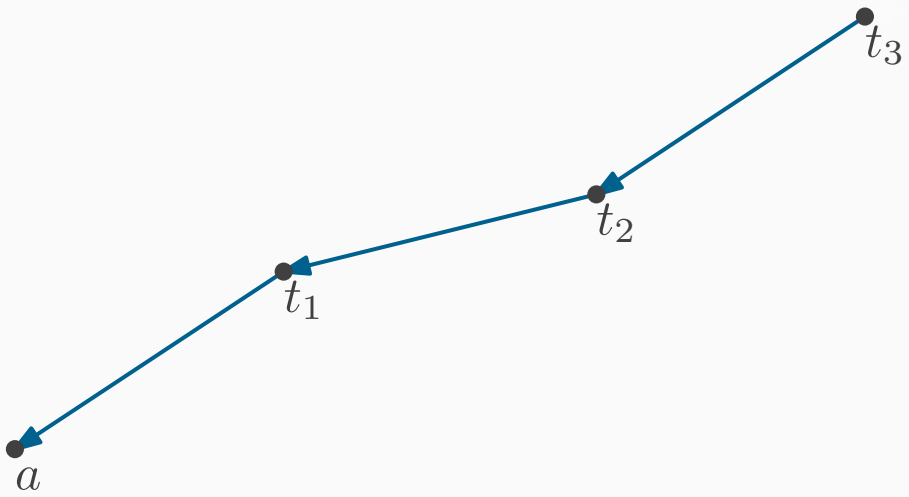
Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \;\; \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$



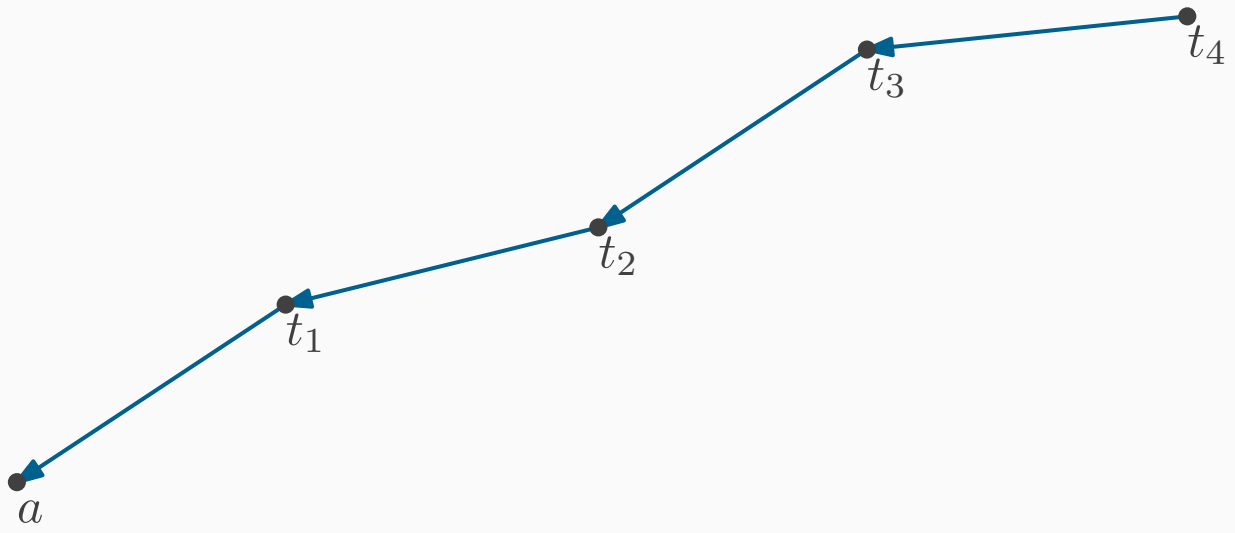
Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \;\; \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$



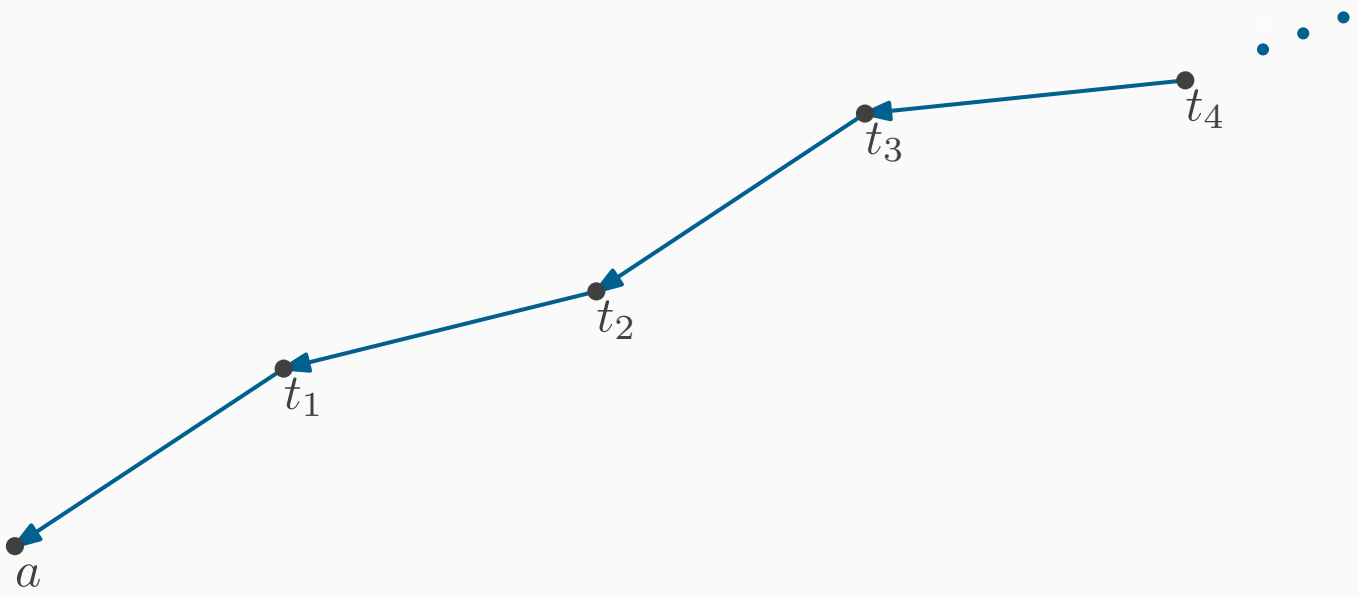
Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \;\; \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$



Chase example I

$$\texttt{Human}(x) \quad \rightarrow \quad \exists y \ \texttt{Mother}(y, x) \wedge \texttt{Human}(y)$$

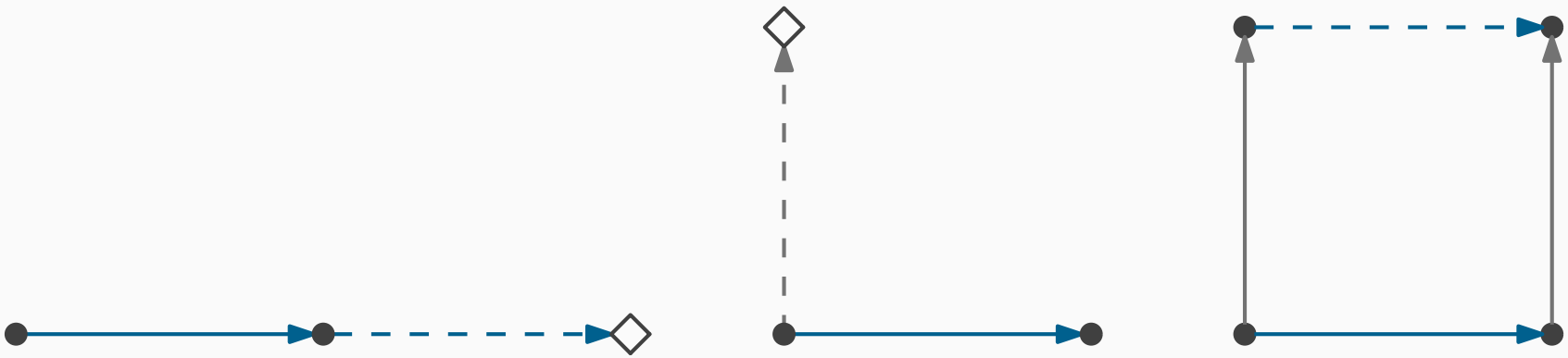


Chase example II

$$
\begin{aligned}
\mathtt{H}(x, y) &\rightarrow \exists z \ \ \mathtt{H}(y, z) \\
\mathtt{H}(x, y) &\rightarrow \exists x' \ \ \mathtt{V}(x, x') \\
\mathtt{H}(x, y) \wedge \mathtt{V}(x, x') \wedge \mathtt{V}(y, y') &\rightarrow \mathtt{H}(x', y')
\end{aligned}
$$

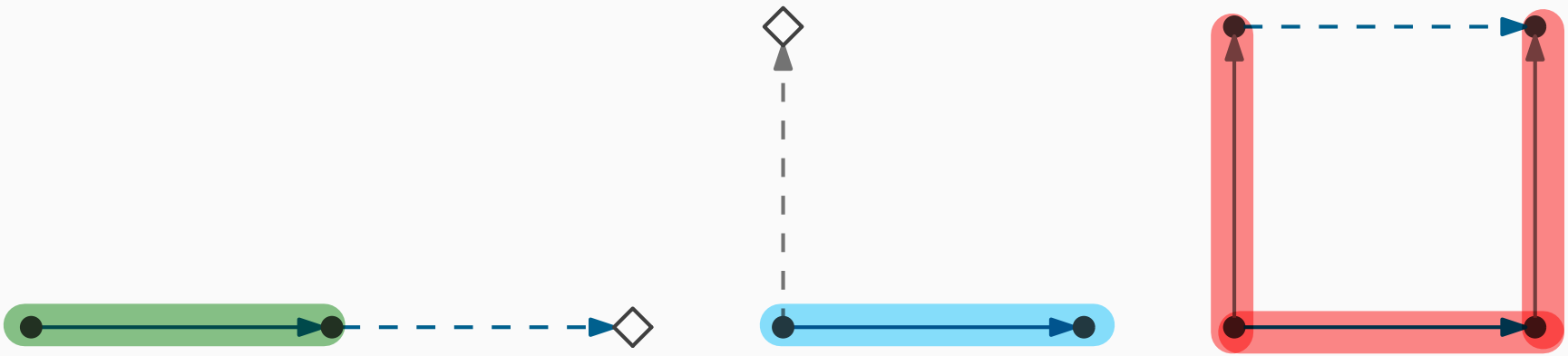
Chase example II

$$
\begin{aligned}
\mathtt{H}(x,y) &\rightarrow \exists z\ \ \mathtt{H}(y,z) \\
\mathtt{H}(x,y) &\rightarrow \exists x'\ \ \mathtt{V}(x,x') \\
\mathtt{H}(x,y) \wedge \mathtt{V}(x,x') \wedge \mathtt{V}(y,y') &\rightarrow \mathtt{H}(x',y')
\end{aligned}
$$



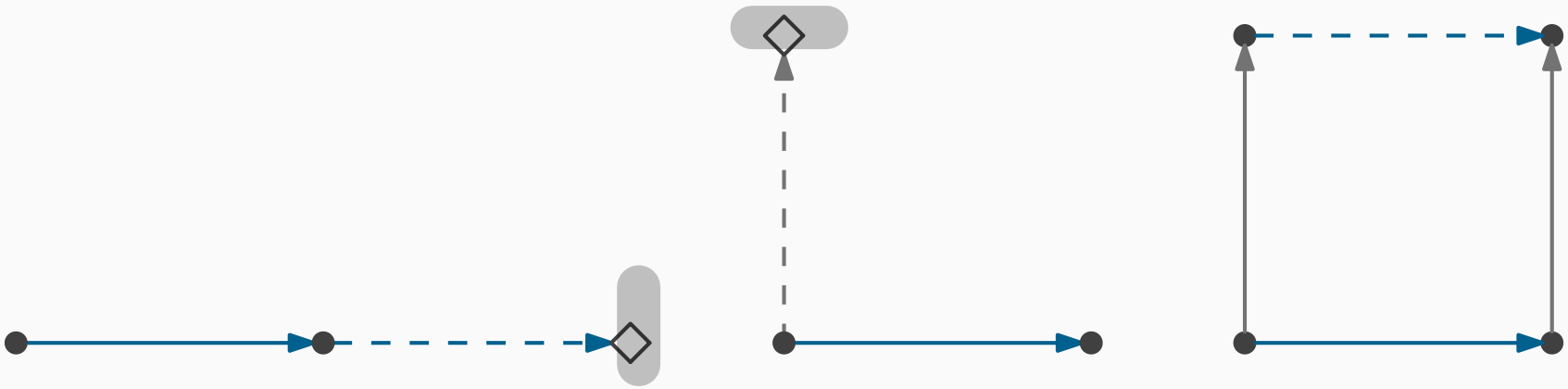
Chase example II

$$\begin{aligned}
\mathtt{H}(x,y) &\rightarrow \exists z \;\; \mathtt{H}(y,z) \\
\mathtt{H}(x,y) &\rightarrow \exists x' \;\; \mathtt{V}(x,x') \\
\mathtt{H}(x,y) \wedge \mathtt{V}(x,x') \wedge \mathtt{V}(y,y') &\rightarrow \mathtt{H}(x',y')
\end{aligned}$$



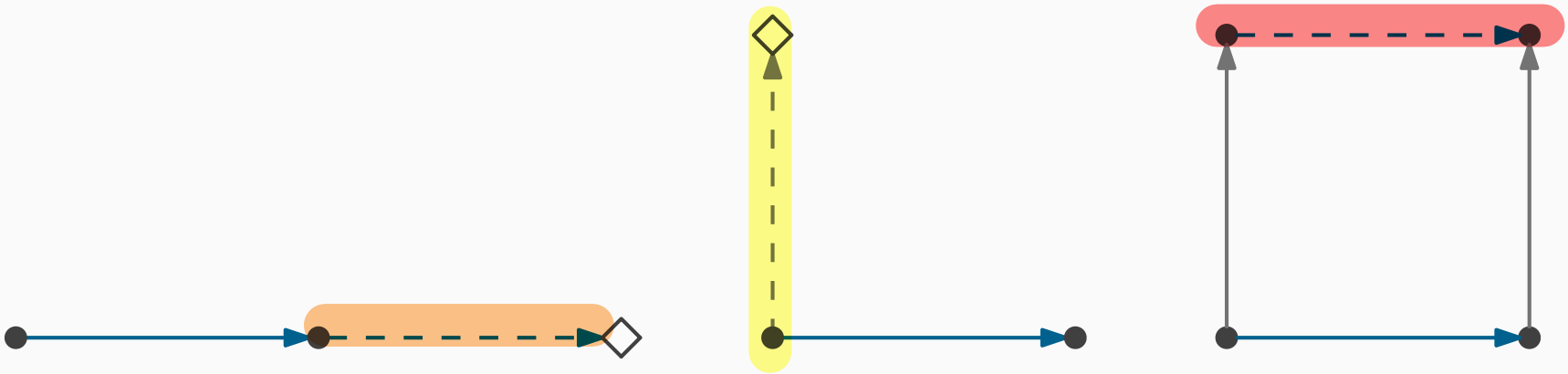
Chase example II

$$\begin{aligned}
\mathtt{H}(x,y) &\rightarrow \exists z\ \mathtt{H}(y,z)\\
\mathtt{H}(x,y) &\rightarrow \exists x'\ \mathtt{V}(x,x')\\
\mathtt{H}(x,y) \wedge \mathtt{V}(x,x') \wedge \mathtt{V}(y,y') &\rightarrow \mathtt{H}(x',y')
\end{aligned}$$

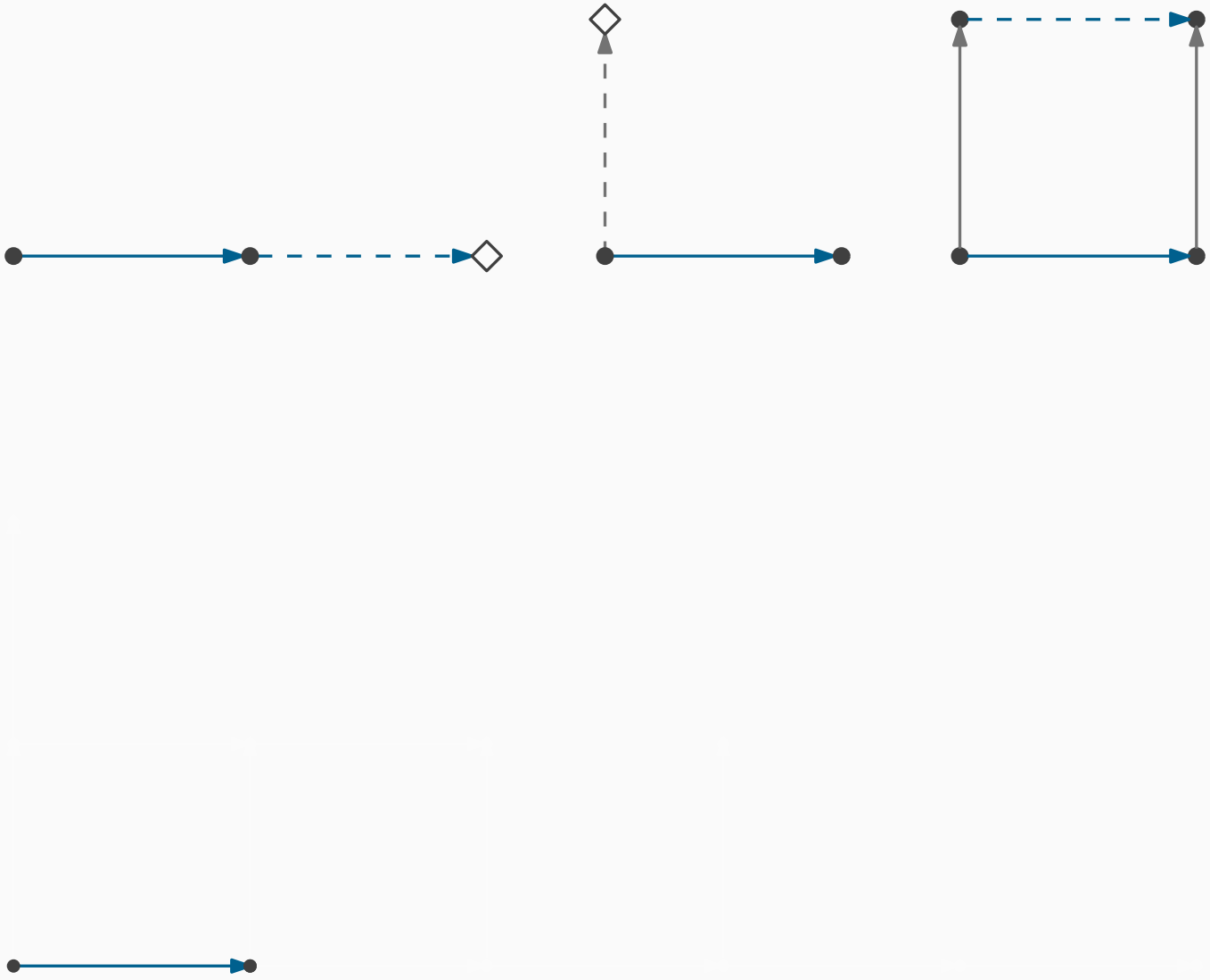


Chase example II

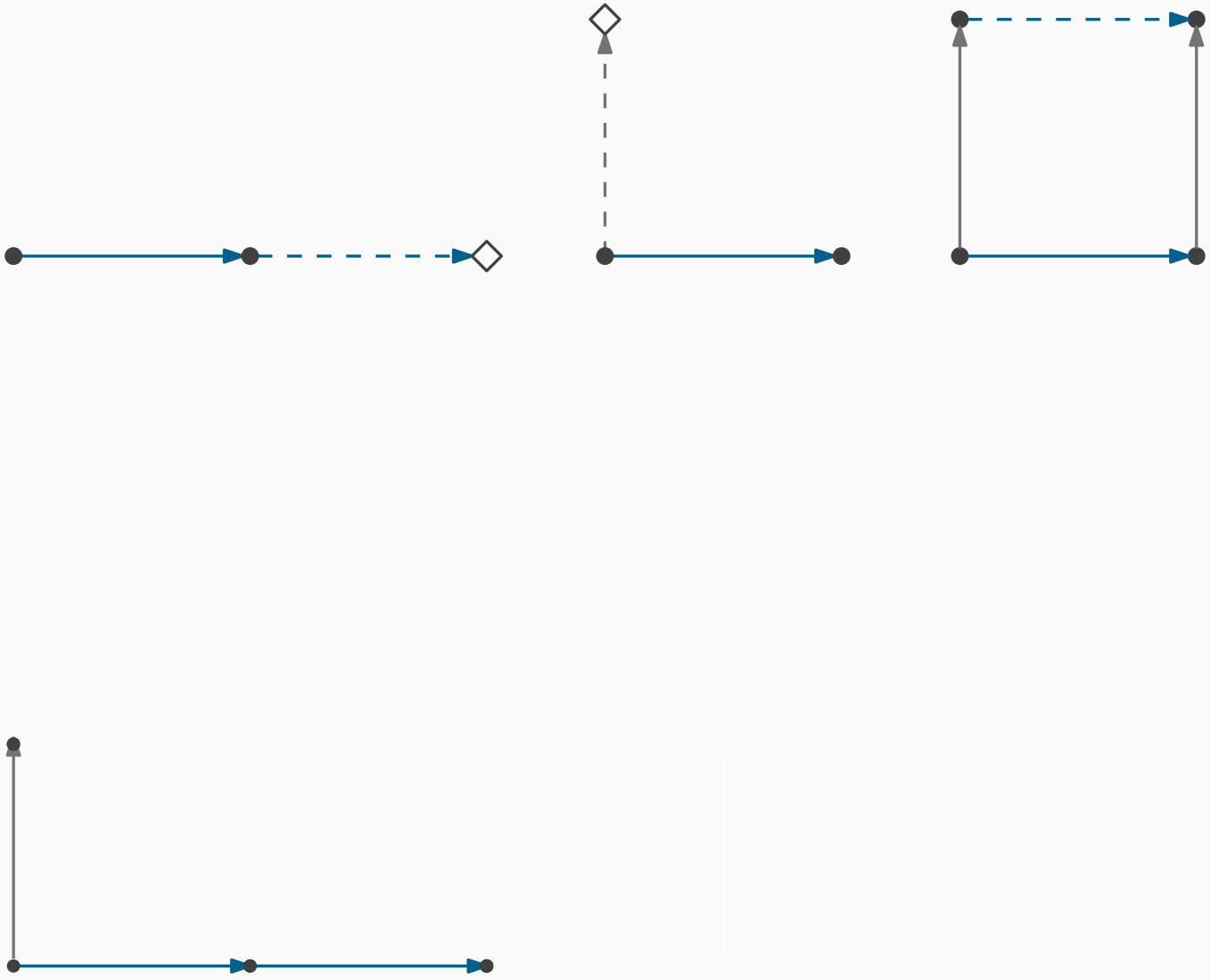
$$\begin{aligned}
\mathtt{H}(x,y) &\rightarrow \exists z\ \ \mathtt{H}(y,z) \\
\mathtt{H}(x,y) &\rightarrow \exists x'\ \ \mathtt{V}(x,x') \\
\mathtt{H}(x,y) \wedge \mathtt{V}(x,x') \wedge \mathtt{V}(y,y') &\rightarrow \mathtt{H}(x',y')
\end{aligned}$$



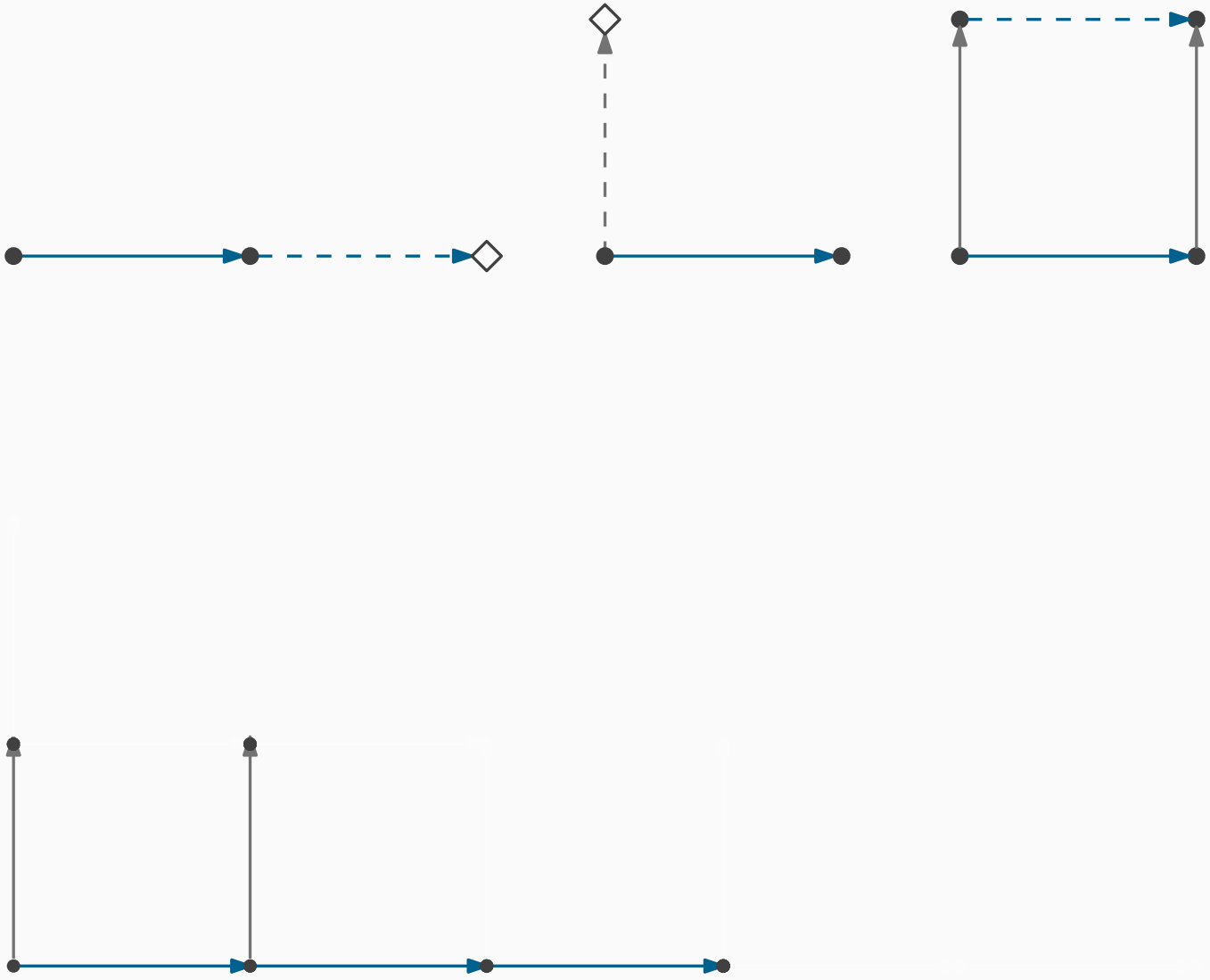
Chase example II



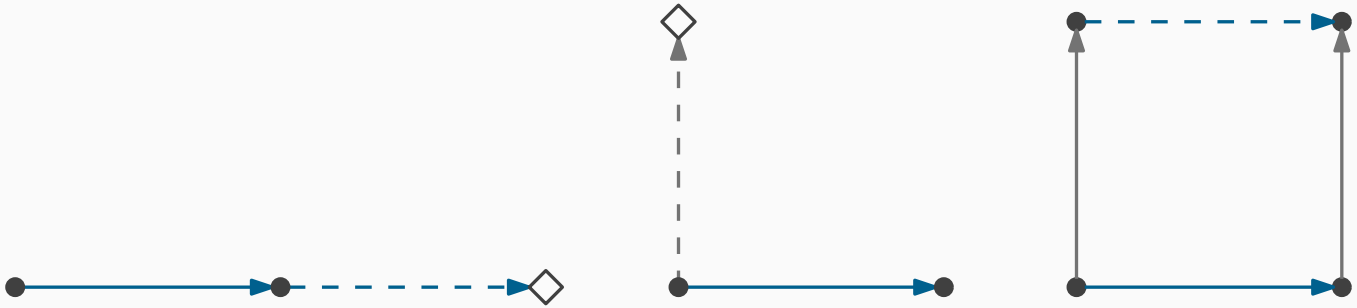
Chase example II

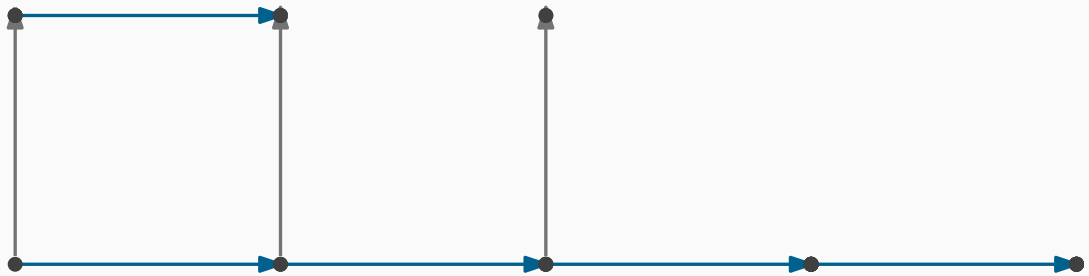


Chase example II

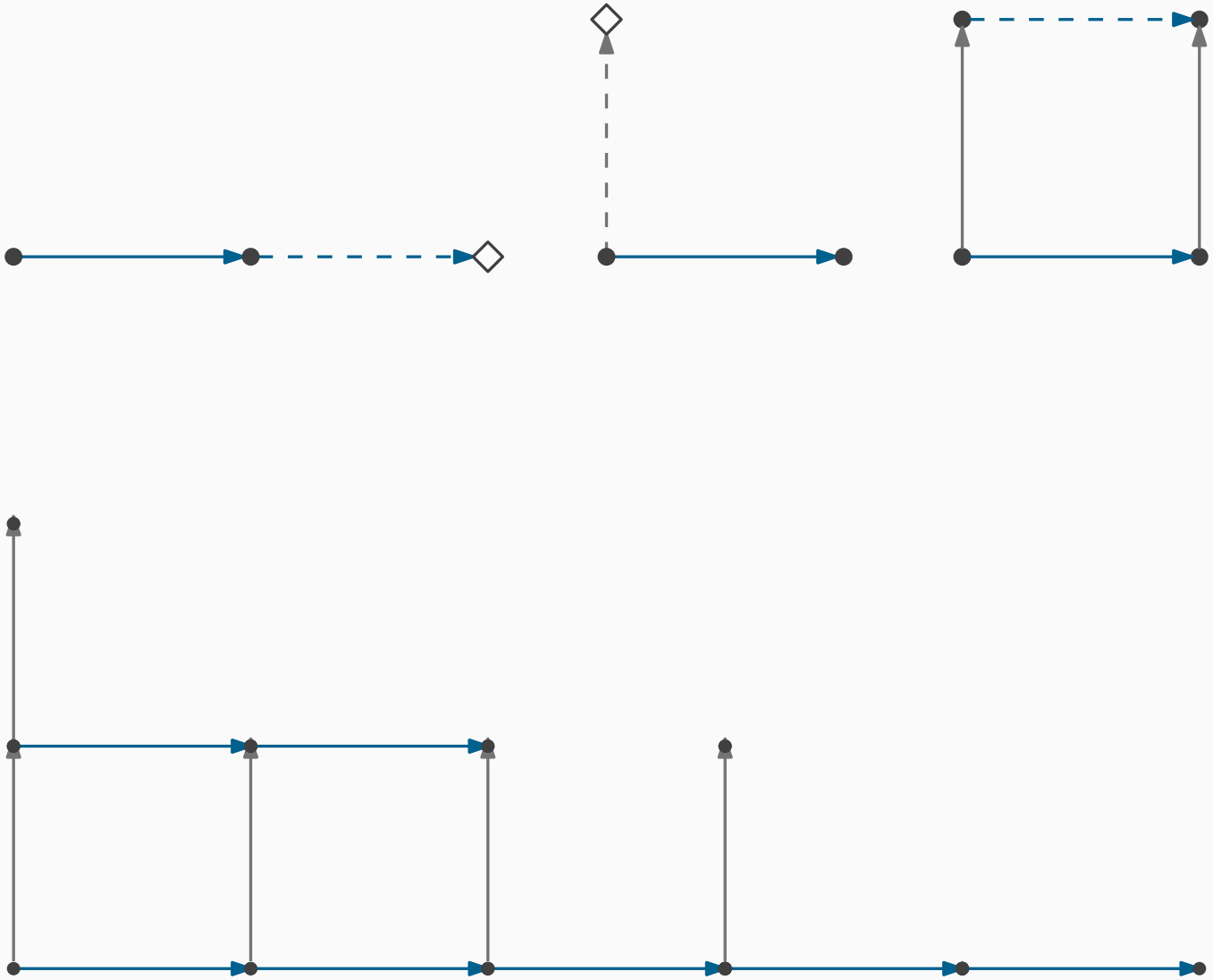


Chase example II

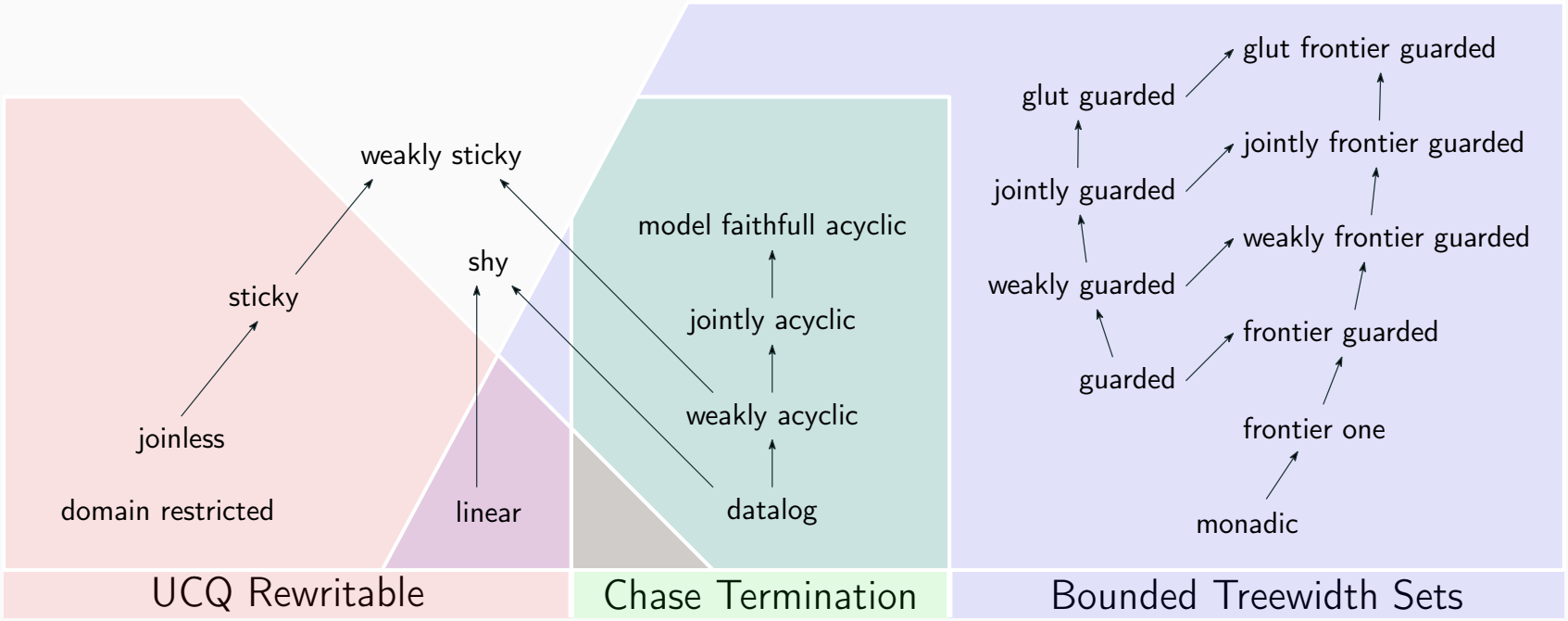




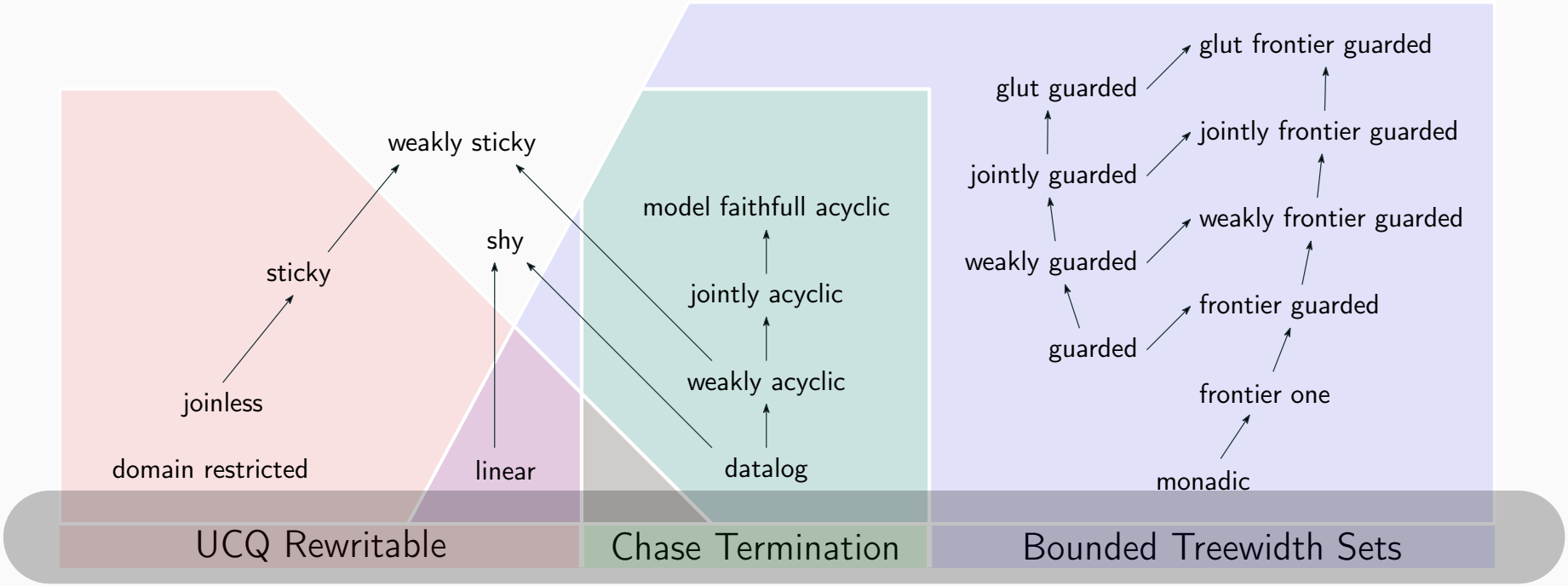
Chase example II



Classes of Existential Rules: CQ Landscape

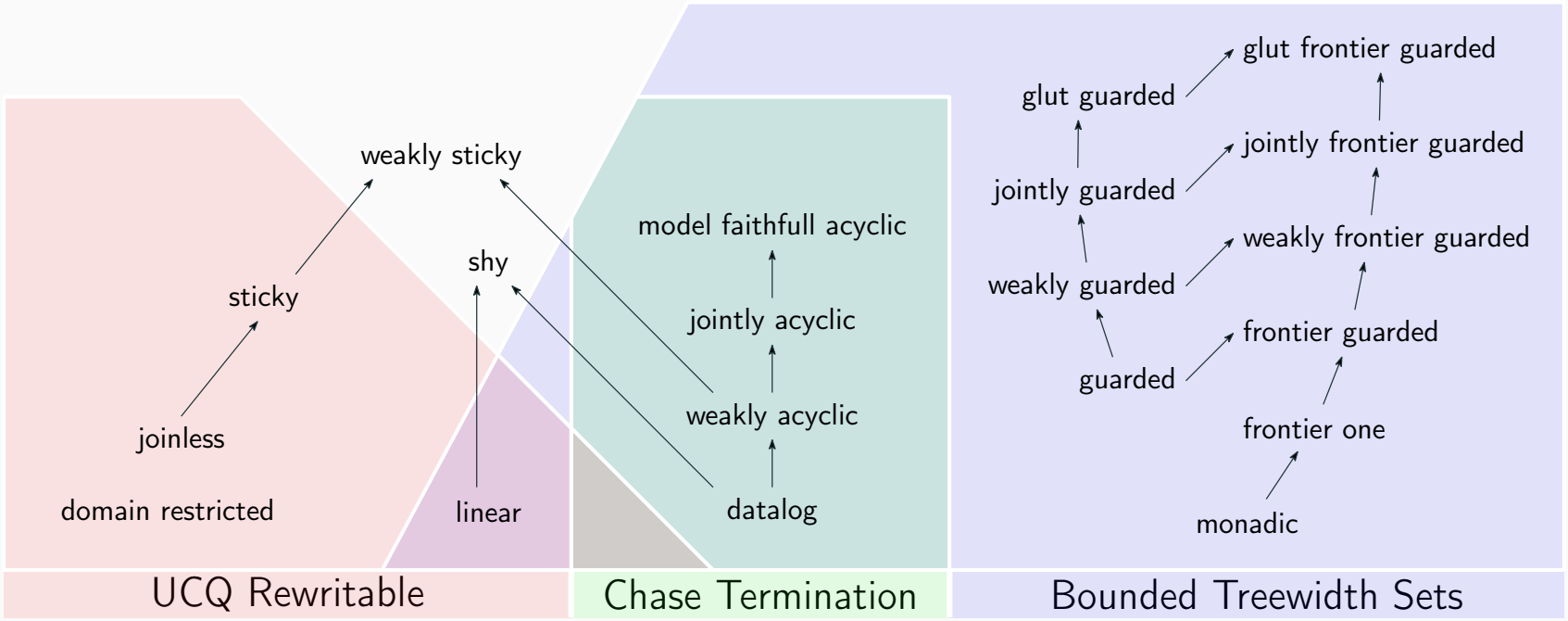


Classes of Existential Rules: CQ Landscape



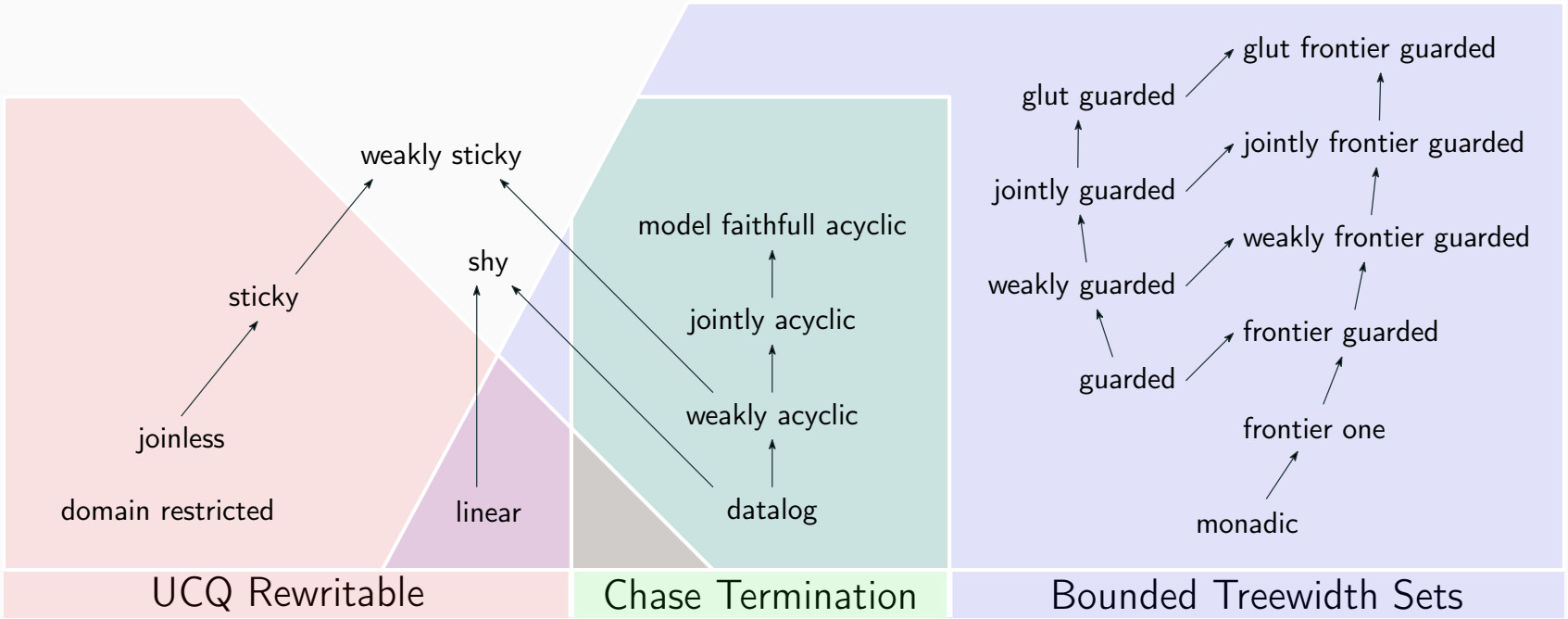
bottom row classes: decidable conjunctive query entailment

Classes of Existential Rules: CQ Landscape



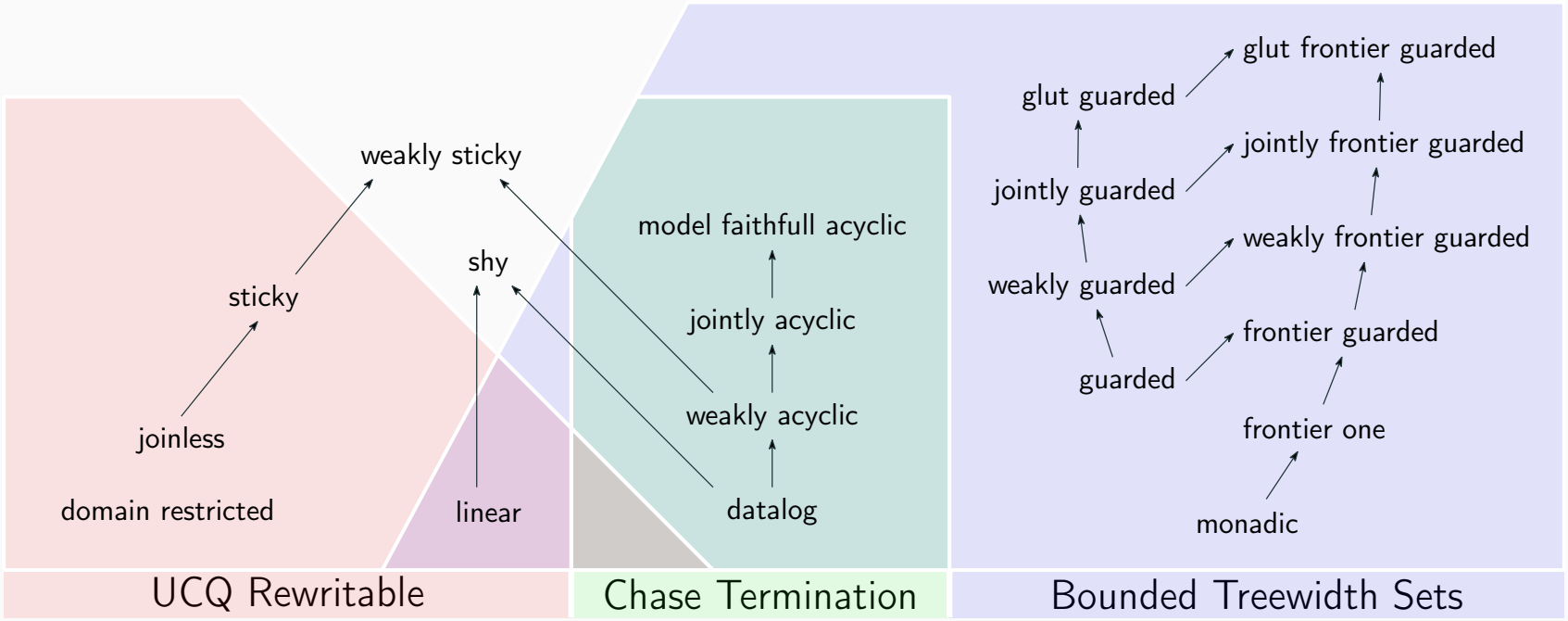
bottom row classes: decidable conjunctive query entailment

Classes of Existential Rules: CQ Landscape



bottom row classes: decidable conjunctive query entailment but membership is undecidable

Classes of Existential Rules: CQ Landscape



bottom row classes: decidable conjunctive query entailment but membership is undecidable

small classes: decidable conjunctive query entailment and membership

This paper:
RPQs and Existential Rules

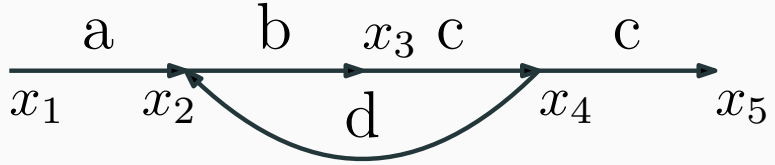
We want to focus on existential rules and Regular Path Queries with respect to the landscape.

Regular Path Queries

are defined by regular languages

$$Q = ab(c + d)^*$$

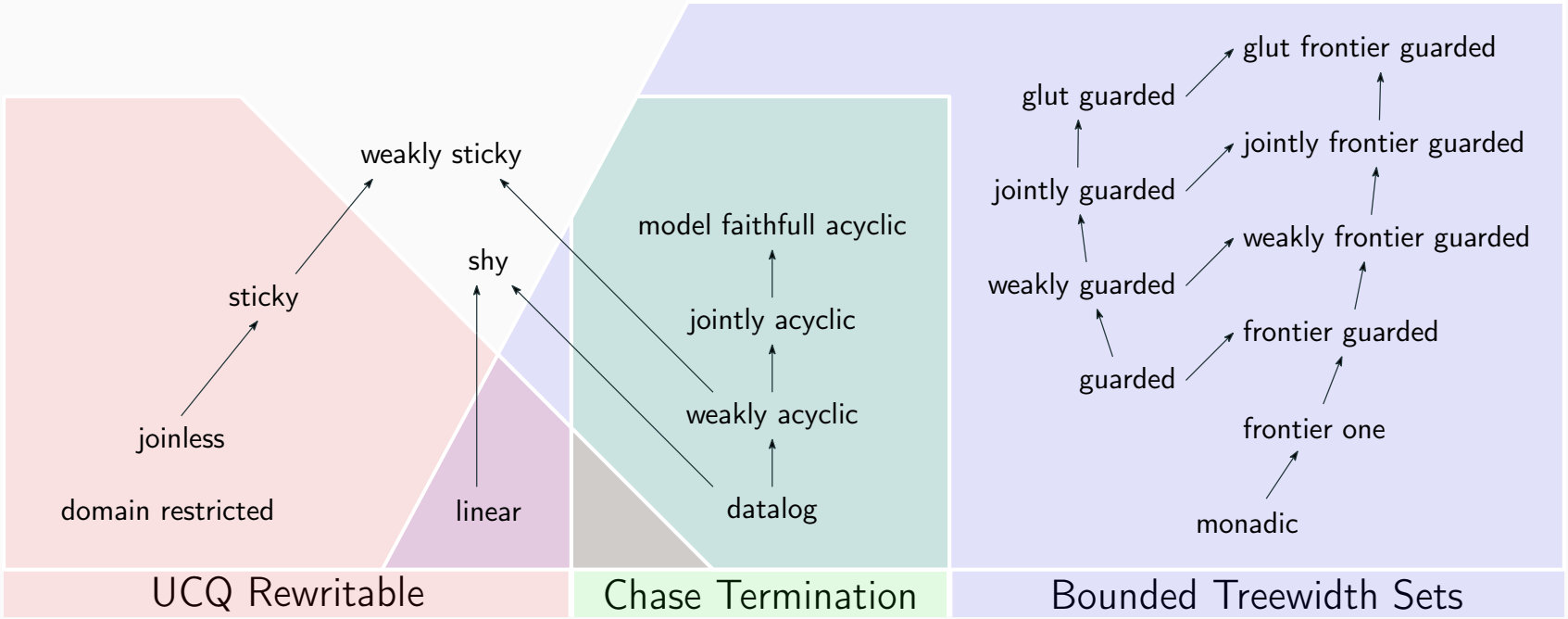
$D =$



$$Q(D) = (\langle x_1, x_3 \rangle, \langle x_1, x_4 \rangle, \langle x_1, x_5 \rangle, \langle x_1, x_2 \rangle)$$

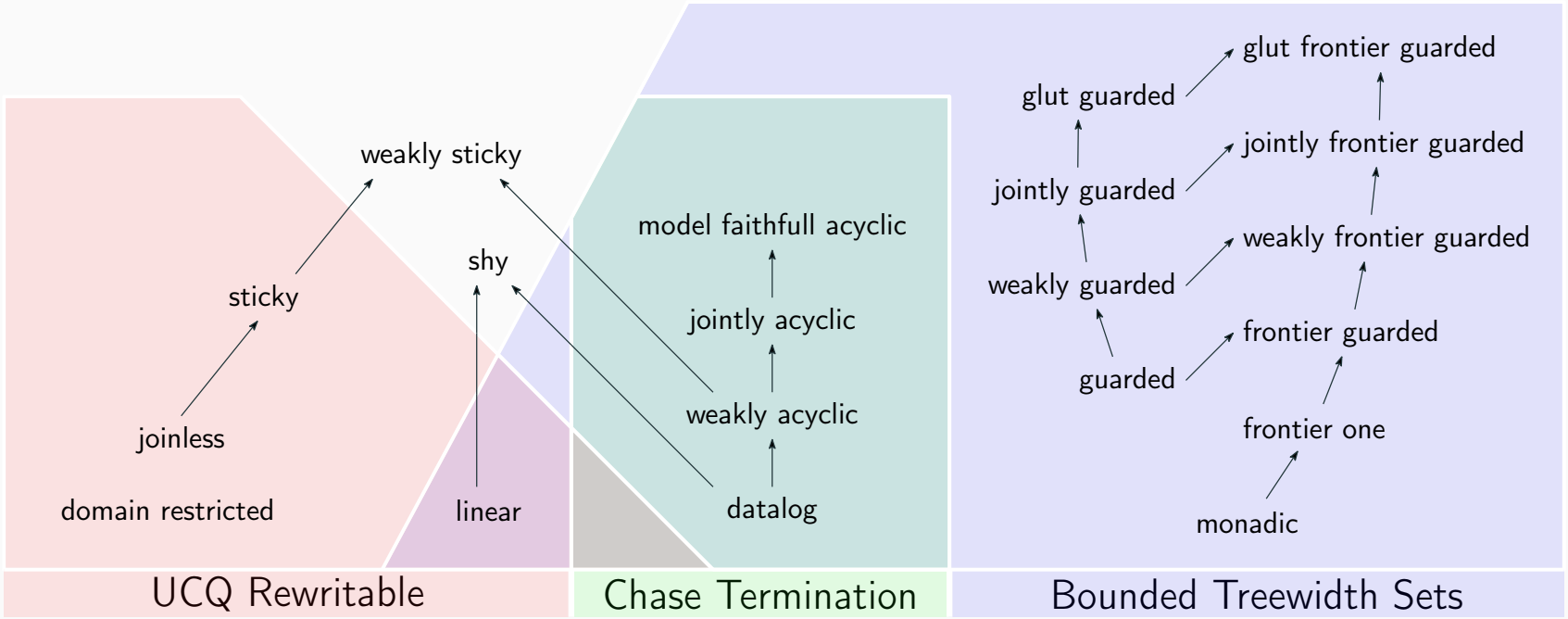
Question?

How does the landscape translate to the RPQ case?



Question?

How does the landscape translate to the RPQ case?

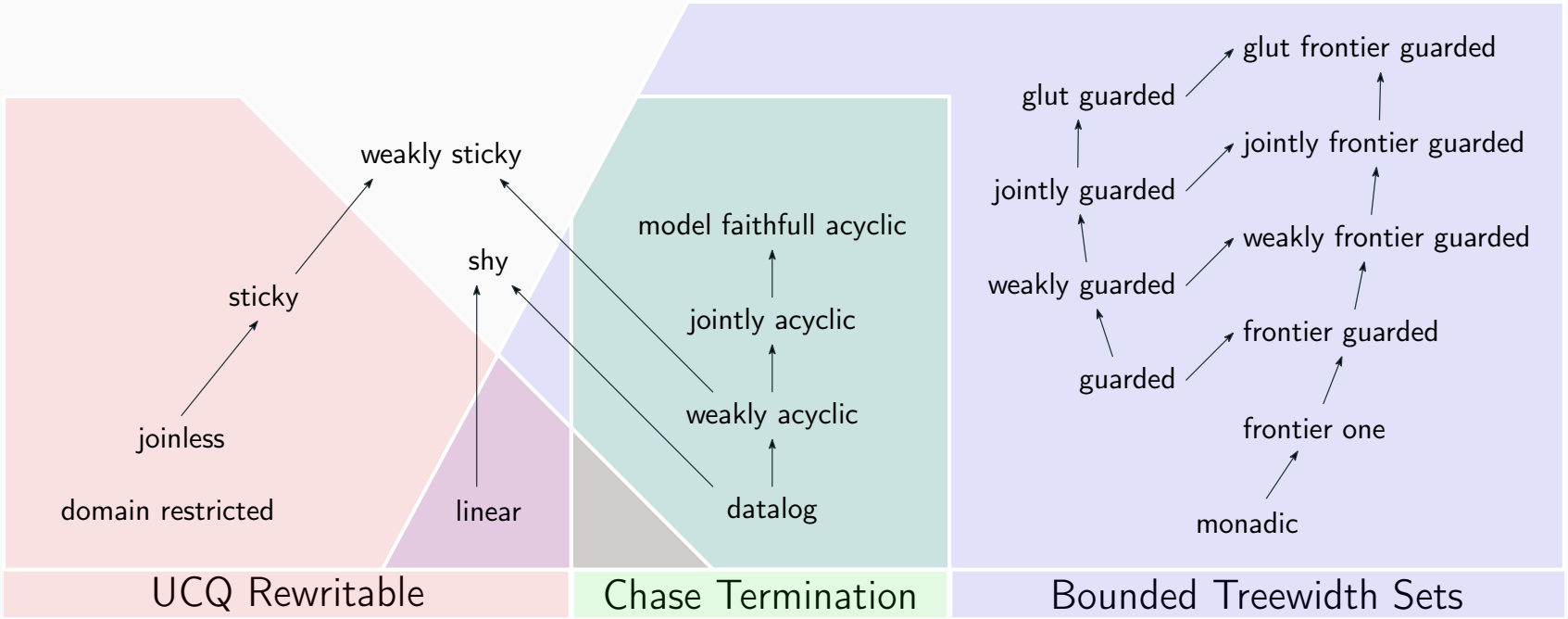


Feller et al. 2023:

Finite Clicquewidth Sets (`fcs`) class of existential rules admits decidable C2RPQ entailment.

Question?

How does the landscape translate to the RPQ case?



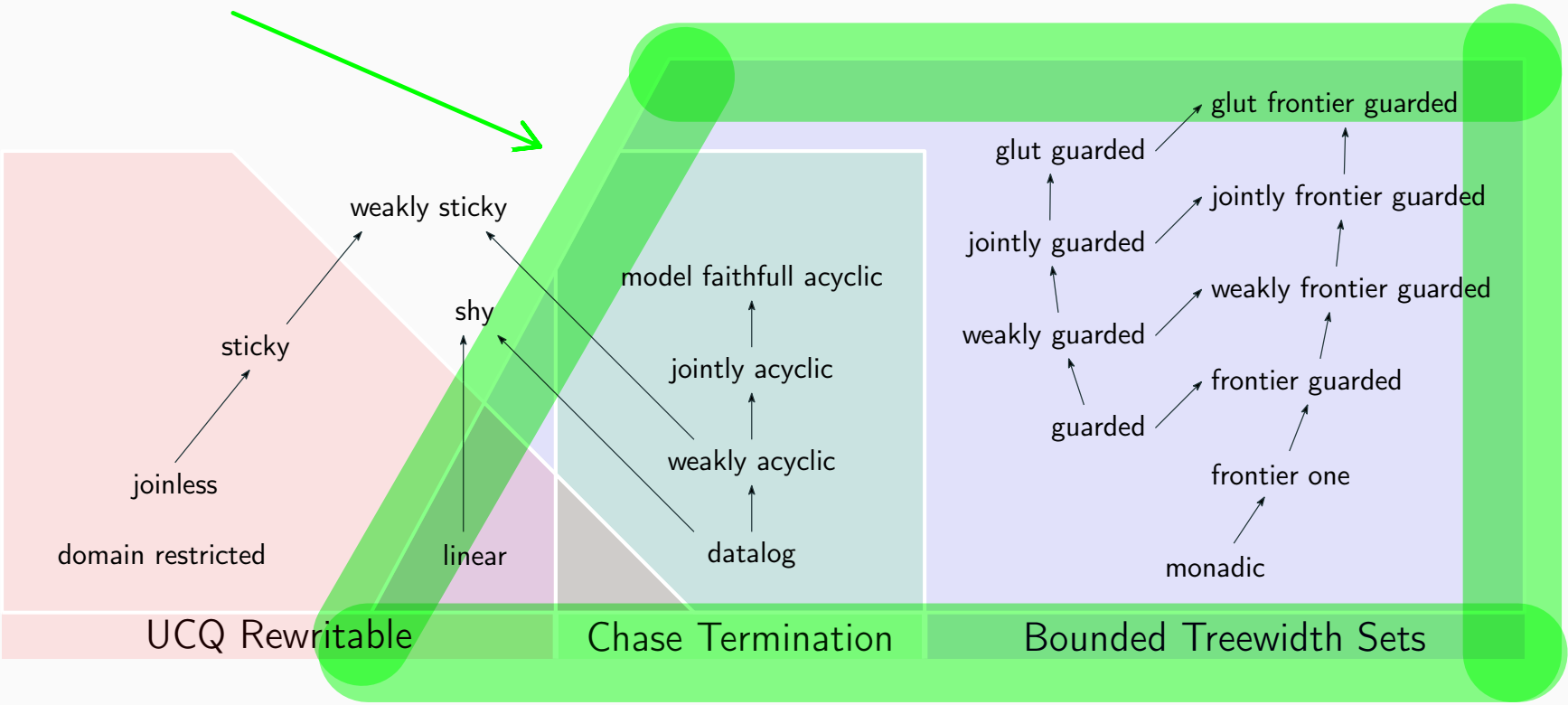
Feller et al. 2023:

subsumes

Finite Clicquewidth Sets (`fcs`) class of existential rules admits decidable C2RPQ entailment.

Question?

How does the landscape translate to the RPQ case?

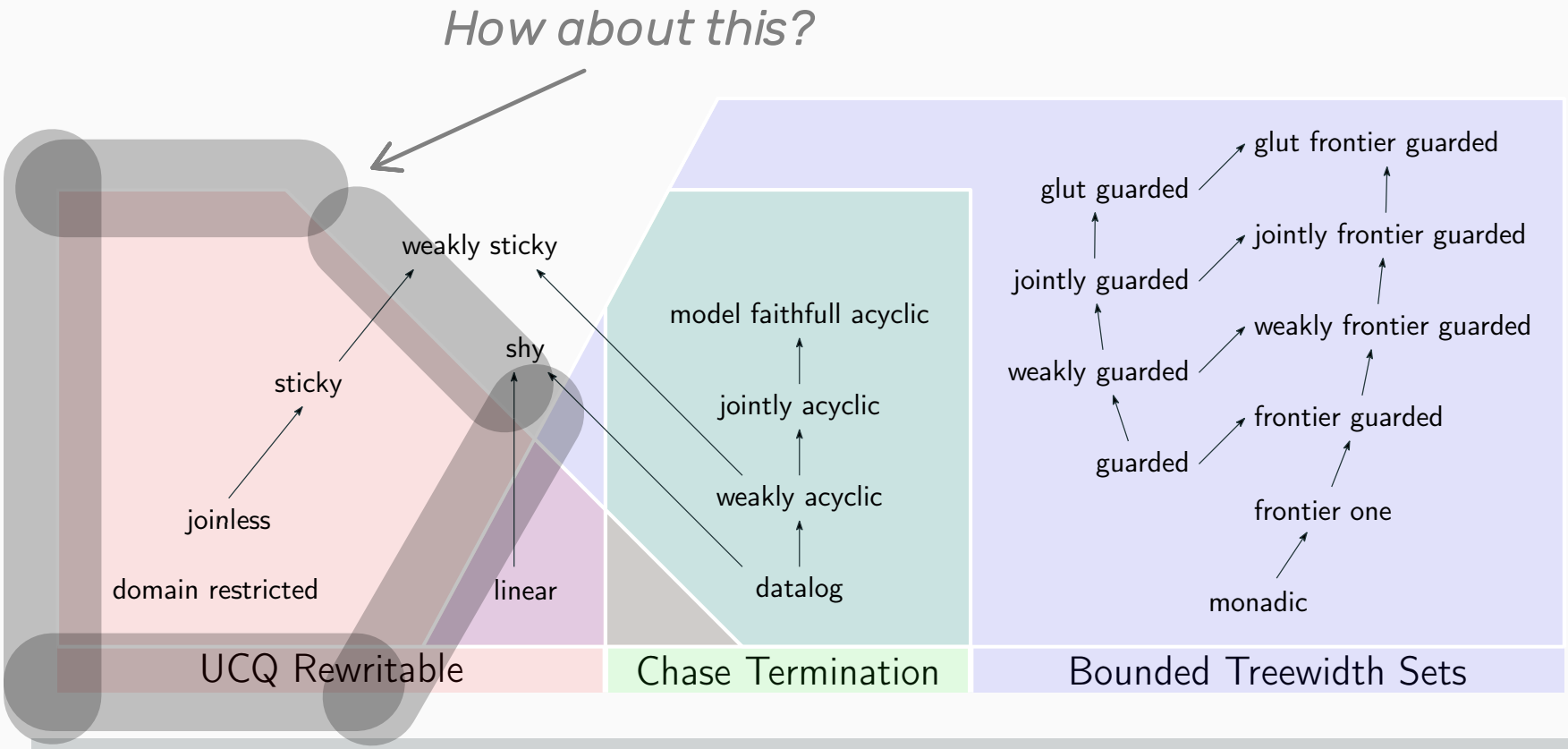


Feller et al. 2023:

Finite Clicquewidth Sets (`fcs`) class of existential rules admits decidable C2RPQ entailment.

Question?

How does the landscape translate to the RPQ case?



Feller et al. 2023:

Finite Clicquewidth Sets (`fcs`) class of existential rules admits decidable C2RPQ entailment.

The main character: UCQ-rewritable class of $\exists$-rules

The main character: UCQ-rewritable class of ∃-rules

UCQ-rewritable class definition

The main character: UCQ-rewritable class of $\exists$-rules

UCQ-rewritable class definition

A ruleset $\mathcal{R}$ is *UCQ-rewritable* if for every CQ Q there exists a UCQ Q' such that for every database $\mathcal{D}$ the following holds:

The main character: UCQ-rewritable class of $\exists$-rules

UCQ-rewritable class definition

A ruleset $\mathcal{R}$ is *UCQ-rewritable* if for every CQ Q there exists a UCQ Q' such that for every database $\mathcal{D}$ the following holds:

$$\mathcal{D}, \mathcal{R} \models Q \iff \mathcal{D} \models Q'$$

The main character: UCQ-rewritable class of $\exists$-rules

UCQ-rewritable class definition

A ruleset $\mathcal{R}$ is *UCQ-rewritable* if for every CQ Q there exists a UCQ Q' such that for every database $\mathcal{D}$ the following holds:

$$\mathcal{D}, \mathcal{R} \models Q \iff \mathcal{D} \models Q'$$

If a ruleset is UCQ-rewritable then the entailment problem for CQs is decidable.

This paper:
Main Results

Main Results

Undecidability

In general, the entailment problem for UCQ-rewritable rulesets and RPQs is undecidable.

Main Results

Undecidability

In general, the entailment problem for UCQ-rewritable rulesets and RPQs is undecidable.

Decidability

The entailment problem for RPQs and the *sticky* subclass of UCQ-rewritable rulesets is decidable.

Main Results

Undecidability

In general, the entailment problem for UCQ-rewritable rulesets and RPQs is undecidable.

Decidability

The entailment problem for RPQs and the *sticky* subclass of UCQ-rewritable rulesets is decidable.

Decidability Gap

By tweaking the undecidability proof we can almost close the gap between decidable and undecidable!

Undecidability Proof

Source of Undecidability

Two-counter automaton. A TCA has a finite set of states $\mathbb{Q}$,

Source of Undecidability

Two-counter automaton. A TCA has a finite set of states $\mathbb{Q}$, two positive integer counters $\mathtt{C}_{\mathtt{x}}$ and $\mathtt{C}_{\mathtt{y}}$,

Source of Undecidability

Two-counter automaton. A TCA has a finite set of states $\mathbb{Q}$, two positive integer counters $\mathtt{C_x}$ and $\mathtt{C_y}$, and an instruction for each state:

Instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

Source of Undecidability

Two-counter automaton. A TCA has a finite set of states $\mathbb{Q}$, two positive integer counters $\mathtt{C_x}$ and $\mathtt{C_y}$, and an instruction for each state:

Instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

Determining whether a TCA $\mathcal{M}$ reaches the halting state q_{fin} from the starting state q_0 with both counters empty is **undecidable**.

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

Ruleset $\mathcal{R}$:

- Independent from $\mathcal{M}$

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

Ruleset $\mathcal{R}$:

- Independent from $\mathcal{M}$
- Builds a grid

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

Ruleset $\mathcal{R}$:

- Independent from $\mathcal{M}$
- Builds a grid
- Tricky with UCQ-rewritable

The Reduction

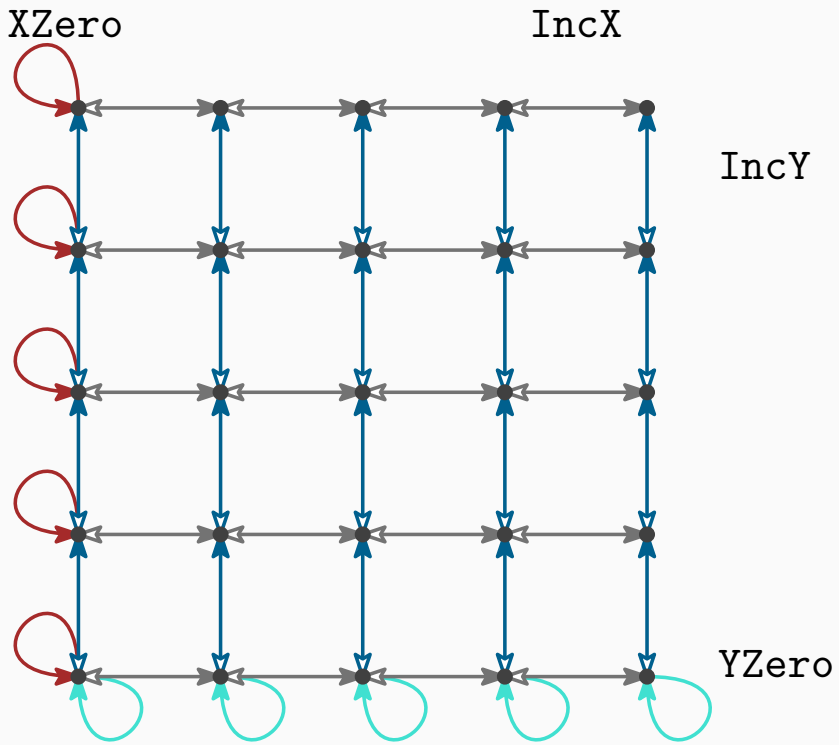
Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

Ruleset $\mathcal{R}$:

- Independent from $\mathcal{M}$
- Builds a grid
- Tricky with UCQ-rewritable



Chase of $\mathcal{D}$ and $\mathcal{R}$

The Reduction

Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

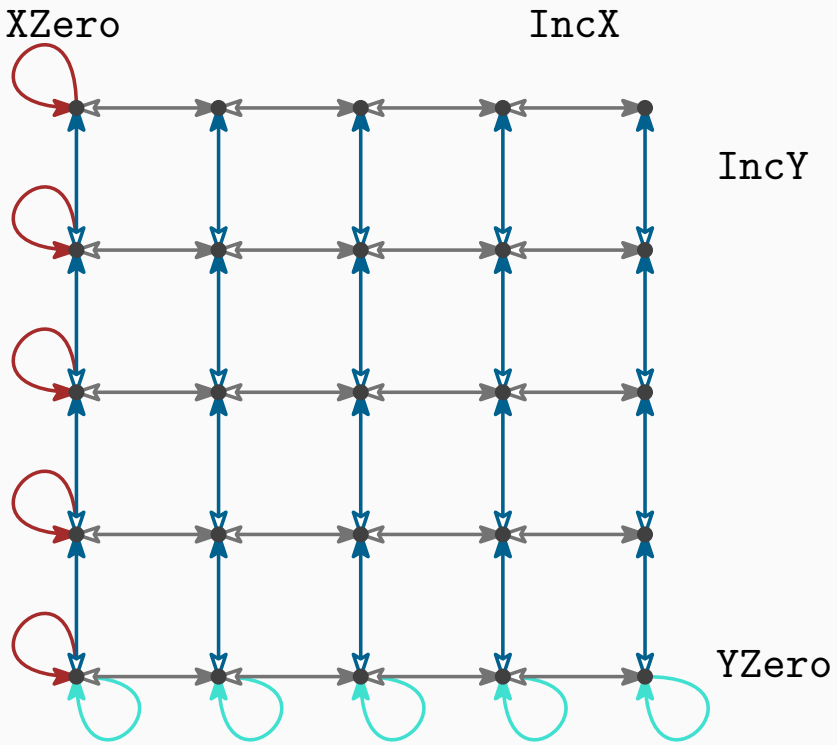
- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

Ruleset $\mathcal{R}$:

- Independent from $\mathcal{M}$
- Builds a grid
- Tricky with UCQ-rewritable

Regular Path Query $Q_{\mathcal{M}}$:

- Depends on $\mathcal{M}$!



Chase of $\mathcal{D}$ and $\mathcal{R}$

The Reduction

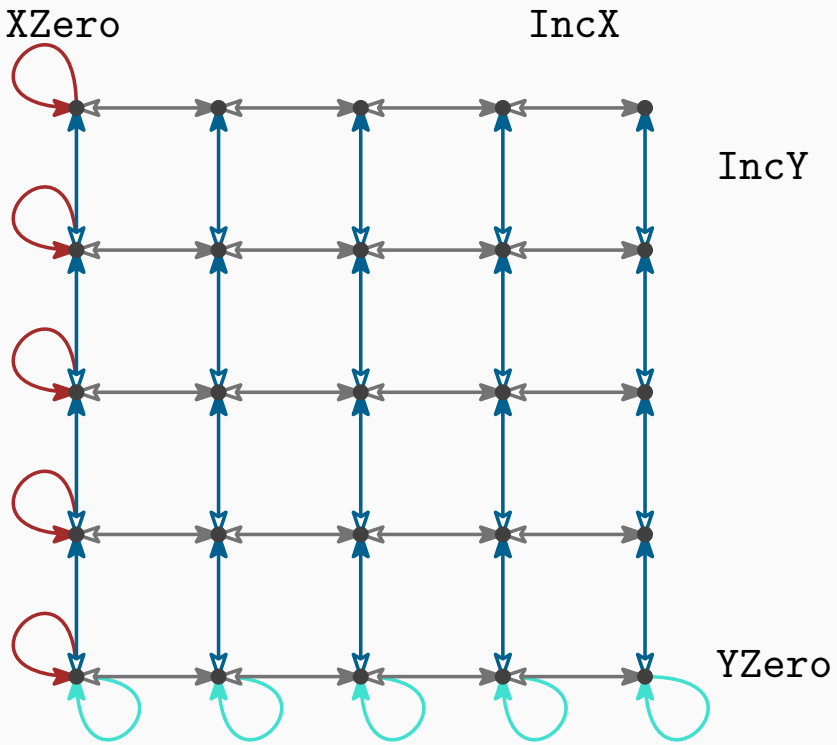
Take a TCA $\mathcal{M}$ and produce $\mathcal{D}, \mathcal{R}$, and $Q_{\mathcal{M}}$

Database $\mathcal{D}$:

- Independent from $\mathcal{M}$
- "Bottom-left grid corner"

Ruleset $\mathcal{R}$:

- Independent from $\mathcal{M}$
- Builds a grid
- Tricky with UCQ-rewritable



Chase of $\mathcal{D}$ and $\mathcal{R}$

Regular Path Query $Q_{\mathcal{M}}$:

- Depends on $\mathcal{M}$!
- Explained on the next slide

Simulating TCAs with RPQs

Simulating TCAs with RPQs

Take TCA $\mathcal{M}$ and the instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

Simulating TCAs with RPQs

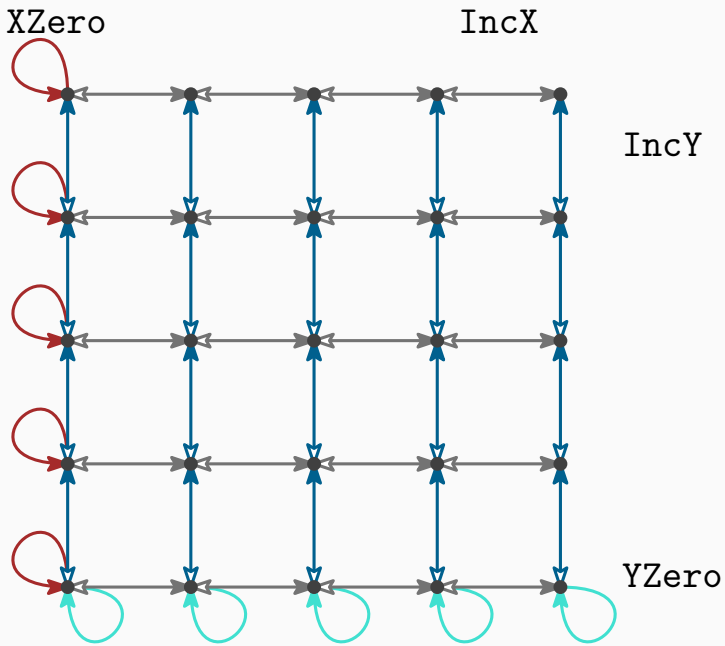
Take TCA $\mathcal{M}$ and the instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

How RPQ reflects TCA

TCA in q with $\langle x, y \rangle$-counters $\cong$
RPQ in q at $\langle x, y \rangle$ in the grid



Simulating TCAs with RPQs

Take TCA $\mathcal{M}$ and the instruction assigned to state q

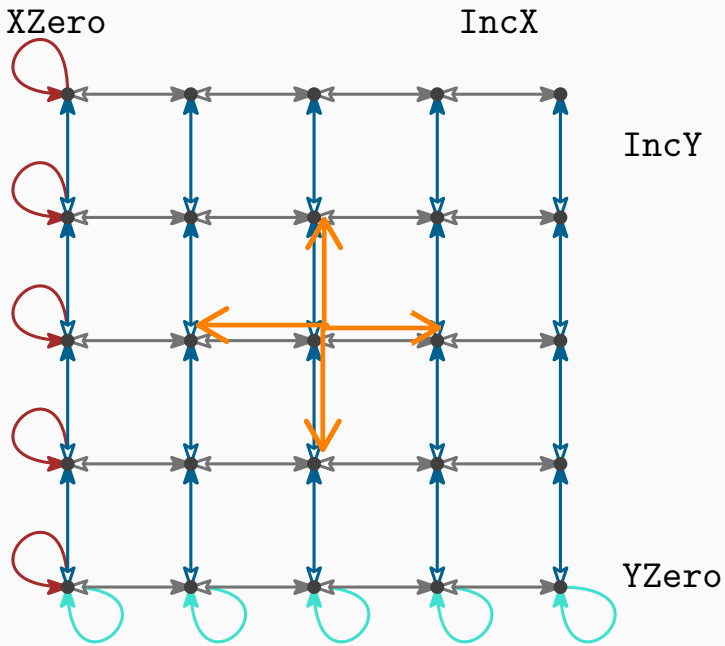
1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,\ \mathtt{move\ from\ q\ to\ q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},\ \mathtt{move\ from\ q\ to\ q_f}$

where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

Easy

How RPQ reflects TCA

TCA in q with $\langle x, y \rangle$-counters $\cong$
RPQ in q at $\langle x, y \rangle$ in the grid



Simulating TCAs with RPQs

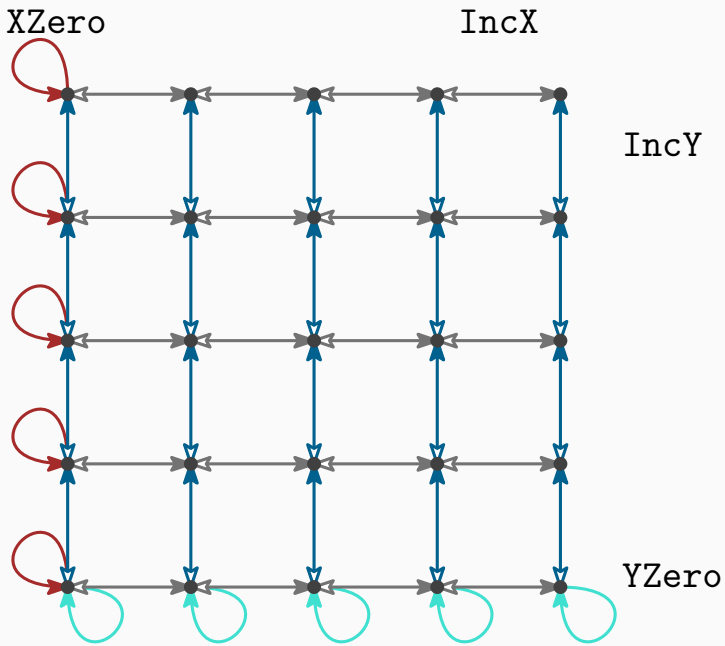
Take TCA $\mathcal{M}$ and the instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then** *Not that easy*
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

How RPQ reflects TCA

TCA in q with $\langle x, y \rangle$-counters $\cong$
RPQ in q at $\langle x, y \rangle$ in the grid



Simulating TCAs with RPQs

Take TCA $\mathcal{M}$ and the instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

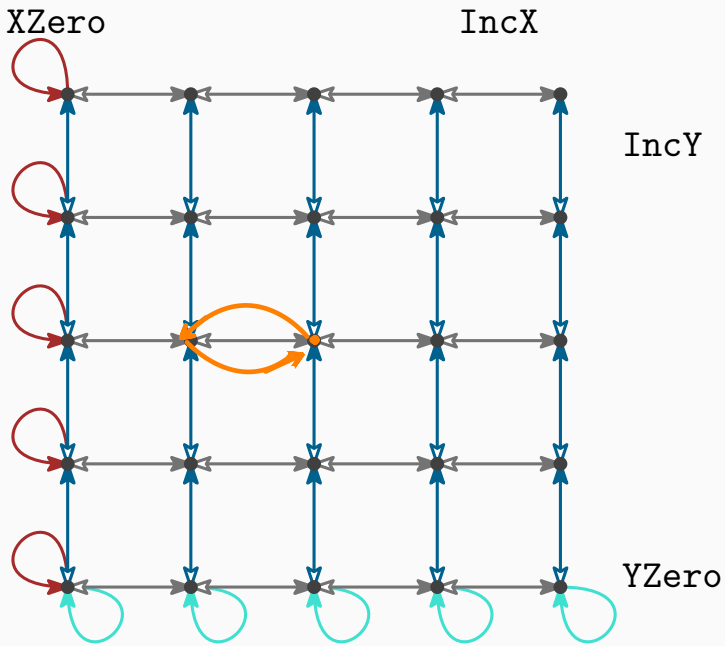
where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

How RPQ reflects TCA

TCA in q with $\langle x, y \rangle$-counters $\cong$ RPQ in q at $\langle x, y \rangle$ in the grid

Conditionals with RPQs

- $\mathtt{C_x} > 0$, go left and then right
- $\mathtt{C_y} > 0$, go down and then up
- $\mathtt{C} = 0$, use X/YZero



Simulating TCAs with RPQs

Take TCA $\mathcal{M}$ and the instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

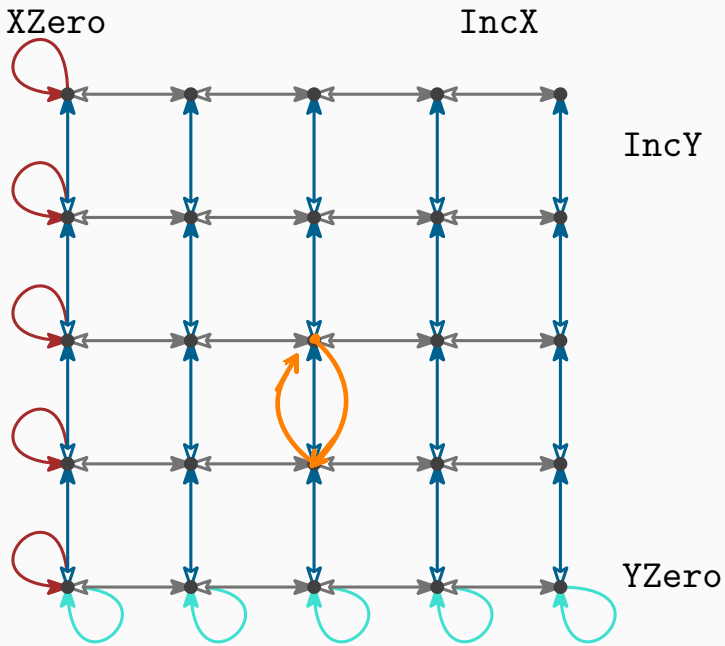
where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

How RPQ reflects TCA

TCA in q with $\langle x, y \rangle$-counters $\cong$
RPQ in q at $\langle x, y \rangle$ in the grid

Conditionals with RPQs

- $\mathtt{C_x} > 0$, go left and then right
- $\mathtt{C_y} > 0$, go down and then up
- $\mathtt{C} = 0$, use X/YZero



Simulating TCAs with RPQs

Take TCA $\mathcal{M}$ and the instruction assigned to state q

1: **if** $\mathtt{C} == 0$ **then**
2: $\quad \mathtt{C} := \mathtt{C} + 1,$ move from q to $\mathtt{q_t}$
3: **else**
4: $\quad \mathtt{C} := \mathtt{C} + \mathtt{d},$ move from q to $\mathtt{q_f}$

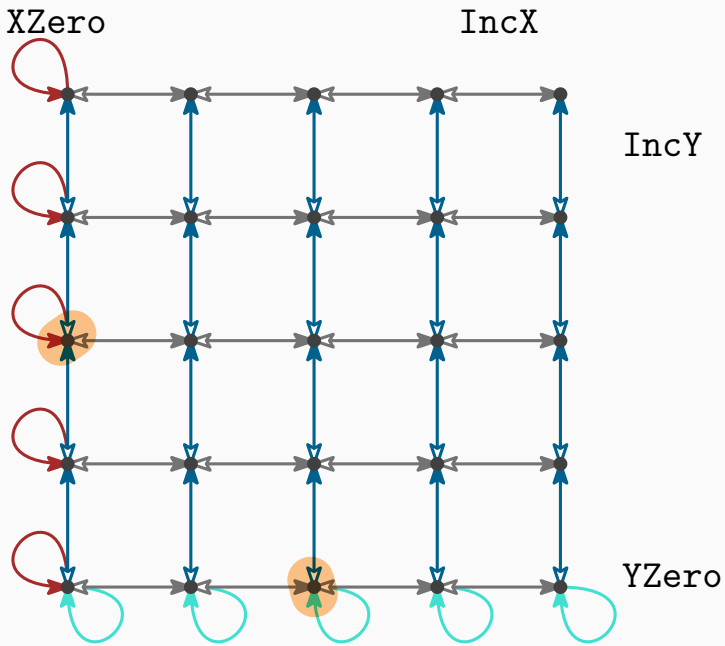
where $\mathtt{C} \in \{\mathtt{C_x}, \mathtt{C_y}\}$, $\mathtt{d} \in \{-1, 1\}$

How RPQ reflects TCA

TCA in q with $\langle x, y \rangle$-counters $\cong$
RPQ in q at $\langle x, y \rangle$ in the grid

Conditionals with RPQs

- $\mathtt{C_x} > 0$, go left and then right
- $\mathtt{C_y} > 0$, go down and then up
- $\mathtt{C} = 0$, use X/YZero



Decidability Proof

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

Co-Recursive Enumerability

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.

Co-Recursive Enumerability

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

1. Define new *stellar* class of rulesets.

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

1. Define new *stellar* class of rulesets.
2. Stellar class is RPQ-finitely controllable.

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

1. Define new *stellar* class of rulesets.
2. Stellar class is RPQ-finitely controllable.
3. Reduce sticky to stellar.

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

1. Define new *stellar* class of rulesets.
2. Stellar class is RPQ-finitely controllable.
3. Reduce sticky to stellar.
    - Extensive rewritability techniques,

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

1. Define new *stellar* class of rulesets.
2. Stellar class is RPQ-finitely controllable.
3. Reduce sticky to stellar.
   - Extensive rewritability techniques,
   - Lots of ruleset surgeries,

Overview of the Decidability Proof

Theorem

The entailment problem for RPQs and the sticky subclass of UCQ-rewritable rulesets is decidable.

Given a database $\mathcal{D}$, a sticky ruleset $\mathcal{R}$ and an RPQ Q we do:

Recursive Enumerability

1. Encode Q in Datalog as Π_Q.
2. Enumerate FO consequences of $\mathcal{D} \wedge \mathcal{R} \wedge \Pi_Q$

Co-Recursive Enumerability

1. Define new *stellar* class of rulesets.
2. Stellar class is RPQ-finitely controllable.
3. Reduce sticky to stellar.
   - Extensive rewritability techniques,
   - Lots of ruleset surgeries,
   - *Heavily* relies on stickiness.

Main Results

Undecidability

In general, the entailment problem for UCQ-rewritable rulesets and RPQs is undecidable.

Main Results

Undecidability

In general, the entailment problem for UCQ-rewritable rulesets and RPQs is undecidable.

Decidability

The entailment problem for RPQs and the *sticky* subclass of UCQ-rewritable rulesets is decidable.

Main Results

Undecidability

In general, the entailment problem for UCQ-rewritable rulesets and RPQs is undecidable.

Decidability

The entailment problem for RPQs and the *sticky* subclass of UCQ-rewritable rulesets is decidable.

Decidability Gap

By tweaking the undecidability proof we can almost close the gap between decidable and undecidable!

Thank you!

Questions?