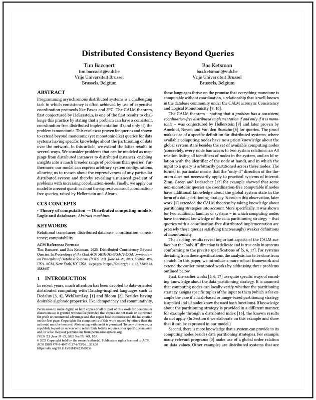
Negation, Coordination and Monotonicity

Bas Ketsman

VUB SOFTWARE LANGUAGES LAB DATABASE GROUP

Contents of this Talk



- K, Koch. *Datalog with Negation and Monotonicity*. ICDT 2020
- Baccaert, K. *Distributed Consistency Beyond Queries*. PODS 2023

Roadmap

- **Context**
- From Monotone to Negation-Free Queries
- Non-rewritable Monotone Queries
- Coordination-Freedom and System Knowledge
- Open Problems

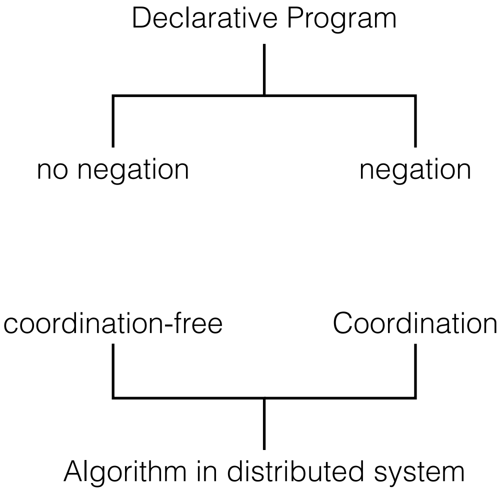
CALM

Programming asynchronous distributed systems in a declarative language

coordination-free Coordination

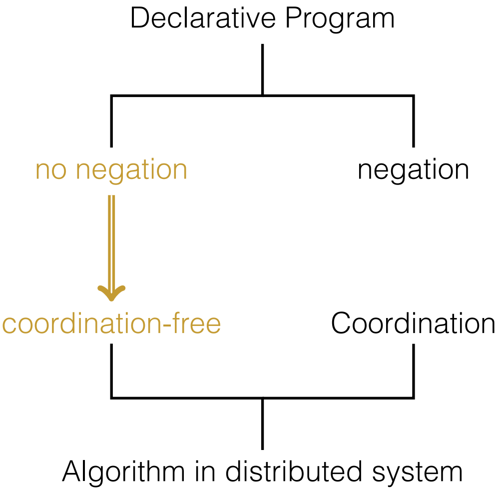
Algorithm in distributed system

CALM



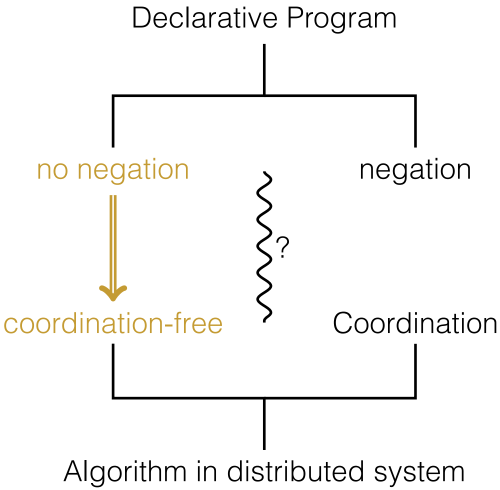
Programming asynchronous distributed systems in a declarative language

CALM



Programming asynchronous distributed systems in a declarative language

CALM



Programming asynchronous distributed systems in a declarative language

Theorem (CALM Theorem)

Monotone Programs = Coordination-Free Programs

- Conjecture: [Hellerstein 2010]
- Proof for queries: [Ameloot, Neven, Van den Bussche 2011]

Formalization

[Ameloot, Neven, Van den Bussche 2011]

- Programs computing **queries**
  - Semantic objects
  - data-, network-, partition-independent
- **Asynchronous shared-nothing** system with at-least once message arrival guarantees
  - Write-only output relations
- An algorithm computes a query if the output relations **eventually** represent the output
- An algorithm is **coordination-free** if for every network and input database there is an “ideal” data partitioning for which the algorithm generates the query answer before any **communication** is done.

Formalization

Queries

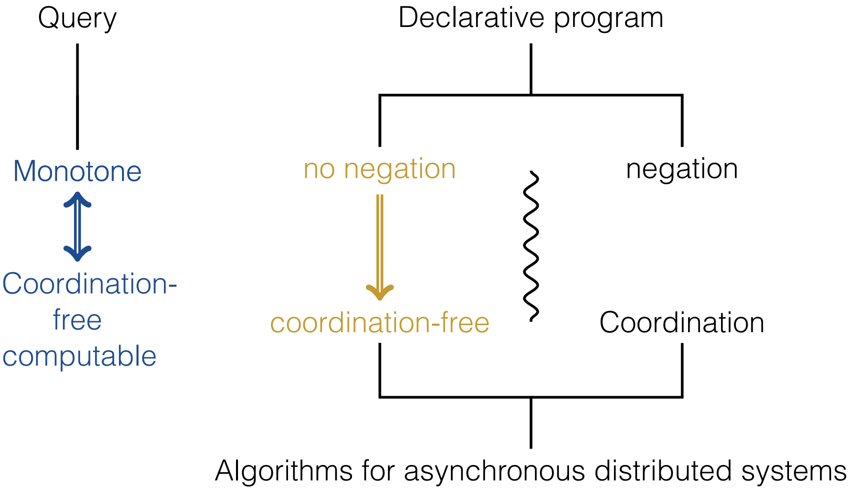
Monotone

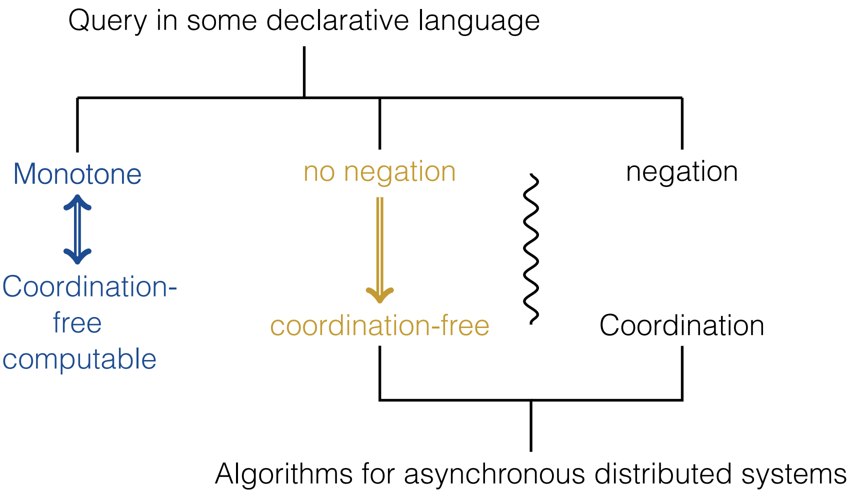
$\Updownarrow$

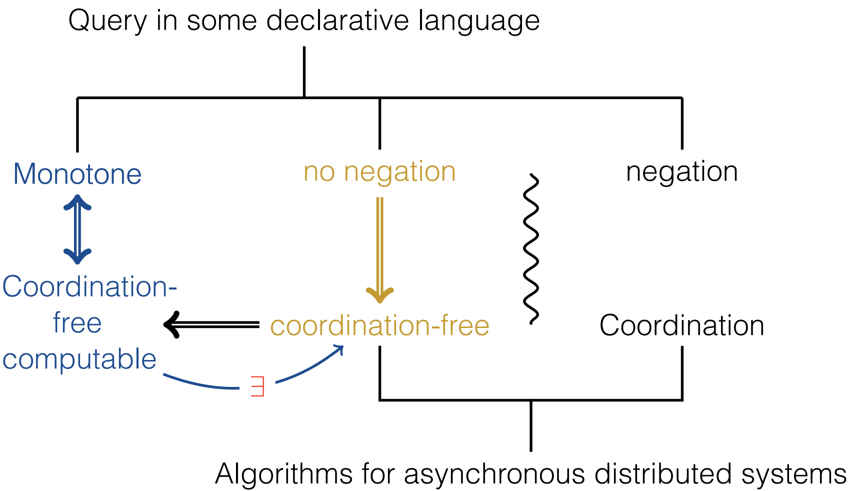
Coordination-free computable

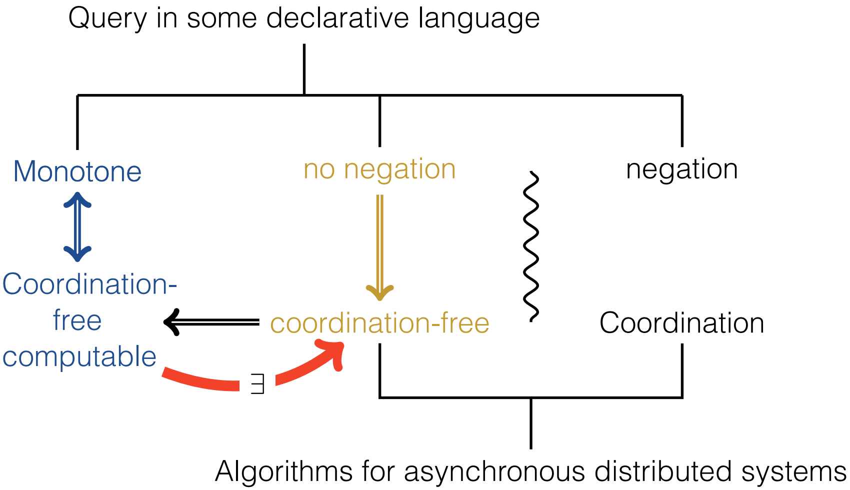
[Ameloot, Neven, Van den Bussche 2011]

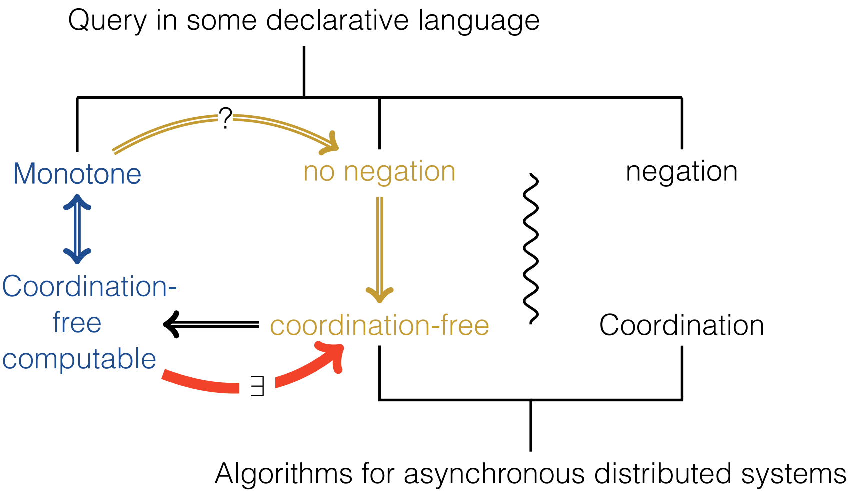
- Programs computing **queries**
  - Semantic objects
  - data-, network-, partition-independent
- **Asynchronous shared-nothing** system with at-least once message arrival guarantees
  - Write-only output relations
- An algorithm computes a query if the output relations **eventually** represent the output
- An algorithm is **coordination-free** if for every network and input database there is an “ideal” data partitioning for which the algorithm generates the query answer before any **communication** is done.


Query
Monotone
Coordination-
free
computable
Declarative program
no negation
negation
coordination-free
Coordination
Algorithms for asynchronous distributed systems


Query in some declarative language
Monotone
Coordination-free computable
no negation
coordination-free
negation
Coordination
Algorithms for asynchronous distributed systems

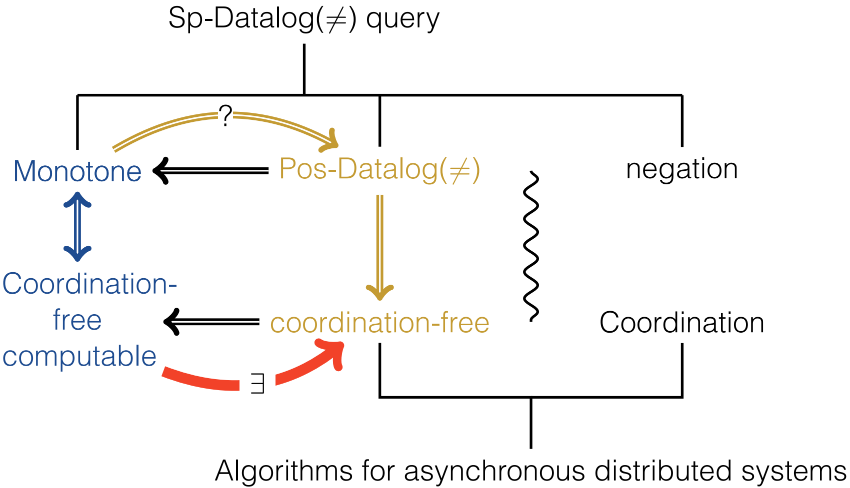

Query in some declarative language
Monotone
no negation
negation
Coordination-
free
computable
coordination-free
Coordination
∃
Algorithms for asynchronous distributed systems


Query in some declarative language
Monotone
no negation
negation
Coordination-
free
computable
coordination-free
Coordination
∃
Algorithms for asynchronous distributed systems


Query in some declarative language
?
Monotone
no negation
negation
Coordination-
free
computable
coordination-free
Coordination
∃
Algorithms for asynchronous distributed systems

Roadmap

- Context
- **From Monotone to Negation-Free Queries**
- Non-rewritable Monotone Queries
- Coordination-Freedom and System Knowledge
- Open Problems


Sp-Datalog(≠) query
?
Monotone
Pos-Datalog(≠)
negation
Coordination-free computable
coordination-free
Coordination
∃
Algorithms for asynchronous distributed systems

Textbook Datalog

[Afrati, Cosmadakis, Yannakakis 1995]

Positive Datalog Program	Monotone
Pos-Datalog	Preserved under Homomorphisms
Pos-Datalog($\neq$)	Monotone
Datalog with Negation	Not necessarily Monotone
Sp-Datalog	Semi-Monotone / Preserved under Extensions
Sp-Datalog($\neq$)	Semi-Monotone / Preserved under Extensions
Str-Datalog	

A Monotone Sp-Datalog($\neq$) Example

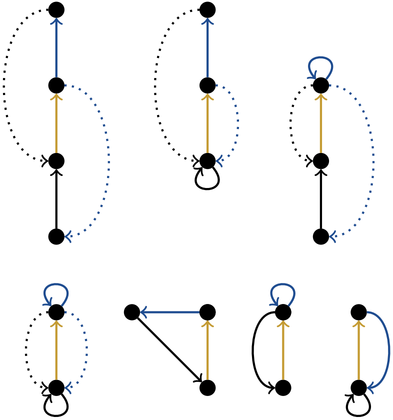
Input: Graph with green (R), blue (S), and black (T) edges

Sp-Datalog Program P**:**

$$
\begin{aligned}
\mathtt{T}_1(x,y) &\leftarrow \mathtt{R}(x,y), \neg\mathtt{S}(y,z), \mathtt{T}(z,x).\\
\mathtt{T}_2(x,y) &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \neg\mathtt{T}(z,x).\\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x,y), \mathtt{T}_2(x,y).\\
\mathtt{O}() &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \mathtt{T}(z,x), x \neq y.
\end{aligned}
$$

Output: Binary output relation O

A Monotone Sp-Datalog($\neq$) Example



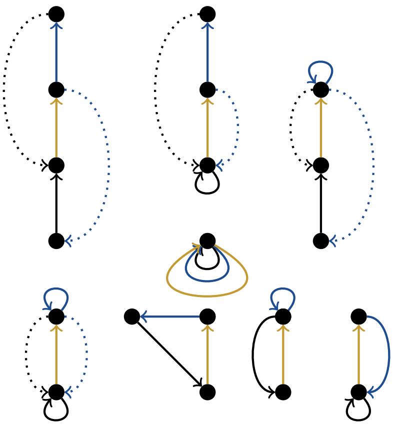
Input: Graph with green (R), blue (S), and black (T) edges

Sp-Datalog Program P:

$$\begin{aligned}
\mathtt{T}_1(x,y) &\leftarrow \mathtt{R}(x,y), \neg\mathtt{S}(y,z), \mathtt{T}(z,x).\\
\mathtt{T}_2(x,y) &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \neg\mathtt{T}(z,x).\\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x,y), \mathtt{T}_2(x,y).\\
\mathtt{O}() &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \mathtt{T}(z,x), x \neq y.
\end{aligned}$$

Output: Binary output relation O

Naive Rewriting: Remove Negated Atoms



Input: Graph with green (R), blue (S), and black (T) edges

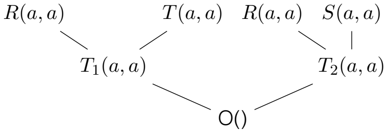
Sp-Datalog Program P:

$$\mathtt{T}_1(x,y) \leftarrow \mathtt{R}(x,y), \cancel{\neg\mathtt{S}(y,z)}, \mathtt{T}(z,x).$$
$$\mathtt{T}_2(x,y) \leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \cancel{\neg\mathtt{T}(z,x)}.$$
$$\mathtt{O}() \leftarrow \mathtt{T}_1(x,y), \mathtt{T}_2(x,y).$$
$$\mathtt{O}() \leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \mathtt{T}(z,x), x \neq y.$$

Output: Binary output relation O

Candidate Proof-Trees with Fringe-Conflicts

(Proof-trees decoupled from a particular database instance)



Proof-tree for P^+

Candidate Proof-Trees with Fringe-Conflicts

(Proof-trees decoupled from a particular database instance)

$R(a,a)$ $T(a,a)$ $R(a,a)$ $S(a,a)$

$T_1(a,a)$ $T_2(a,a)$

O()

$R(a,a)$ $\neg \boldsymbol{S(a,a)}$ $T(a,a)$ $R(a,a)$ $\boldsymbol{S(a,a)}$ $\neg T(a,a)$

$T_1(a,a)$ $T_2(a,a)$

O()

Proof-tree for P^+

Candidate Proof-Trees with Fringe-Conflicts

(Proof-trees decoupled from a particular database instance)

$R(a,a) \quad T(a,a) \quad R(a,a) \quad S(a,a)$

$T_1(a,a) \qquad T_2(a,a)$

O()

Proof-tree for P^+

$R(a,a)\ \boldsymbol{\neg S(a,a)}\ T(a,a) \quad R(a,a)\ \boldsymbol{S(a,a)}\ \neg T(a,a)$

$T_1(a,a) \qquad T_2(a,a)$

O()

Candidate proof-tree^a for P

with fringe conflicts

([a]In paper: candidate proof-tree without inequality conflicts)

Proof-Tree Perspective

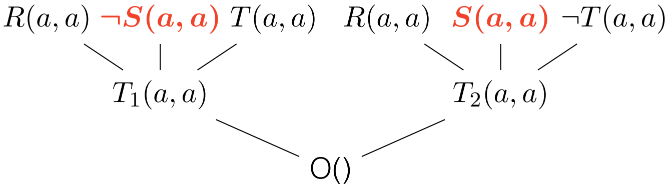
Theorem

For monotone Sp-Datalog$_{\text{conflict-free}}(\neq)$ programs whose candidate proof-trees are without fringe conflicts, negated atoms can be left away without influencing the program semantics.

Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts via disequalities

$$
\begin{aligned}
\mathtt{T}_1(x,y) &\leftarrow \mathtt{R}(x,y), \neg\mathtt{S}(y,z), \mathtt{T}(z,x).\\
\mathtt{T}_2(x,y) &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \neg\mathtt{T}(z,x).\\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x,y), \mathtt{T}_2(x,y).\\
\mathtt{O}() &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \mathtt{T}(z,x), x \neq y.
\end{aligned}
$$



Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts via disequalities

$$\begin{aligned}
\mathtt{T}_1(x, y, y, z) &\leftarrow \mathtt{R}(x, y), \neg\mathtt{S}(y, z), \mathtt{T}(z, x).\\
\mathtt{T}_2(x, y) &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \neg\mathtt{T}(z, x).\\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x, y, y', z'), \mathtt{T}_2(x, y).\\
\mathtt{O}() &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \mathtt{T}(z, x), x \neq y.
\end{aligned}$$

$R(a,a)$ $\neg \boldsymbol{S(a,a)}$ $T(a,a)$ $R(a,a)$ $\boldsymbol{S(a,a)}$ $\neg T(a,a)$

$T_1(a,a,a,a)$ $T_2(a,a)$

O()

Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts via disequalities

$$
\begin{aligned}
\mathtt{Adom}(x) &\leftarrow \text{“active domain of } R\text{, } S\text{, and } T\text{”} \\
\mathtt{T}_1(x, y, y, z) &\leftarrow \mathtt{R}(x, y), \neg\mathtt{S}(y, z), \mathtt{T}(z, x). \\
\mathtt{T}_2(x, y, y', z') &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \neg\mathtt{T}(z, x), \mathtt{Adom}(y'), \mathtt{Adom}(z'). \\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x, y, y', z'), \mathtt{T}_2(x, y, y', z'). \\
\mathtt{O}() &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \mathtt{T}(z, x), x \neq y.
\end{aligned}
$$

$R(a, a)$ $\neg \boldsymbol{S(a, a)}$ $T(a, a)$ $R(a, a)$ $\boldsymbol{S(a, a)}$ $\neg T(a, a)$ $Adom(a)$ $Adom(a)$

$T_1(a, a, a, a)$ $T_2(a, a, a, a)$

O()

Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts via disequalities

$$
\begin{aligned}
\mathtt{Adom}(x) &\leftarrow \text{“active domain of } R\text{, } S\text{, and } T\text{”}\\
\mathtt{T}_1(x, y, y, z) &\leftarrow \mathtt{R}(x, y), \neg\mathtt{S}(y, z), \mathtt{T}(z, x).\\
\mathtt{T}_2(x, y, y', z') &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \neg\mathtt{T}(z, x), \mathtt{Adom}(y'), \mathtt{Adom}(z'), y \neq y'.\\
\mathtt{T}_2(x, y, y', z') &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \neg\mathtt{T}(z, x), \mathtt{Adom}(y'), \mathtt{Adom}(z'), z \neq z'.\\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x, y, y', z'), \mathtt{T}_2(x, y, y', z').\\
\mathtt{O}() &\leftarrow \mathtt{R}(x, y), \mathtt{S}(y, z), \mathtt{T}(z, x), x \neq y.
\end{aligned}
$$

$R(a, a)$ $\neg \boldsymbol{S(a, a)}$ $T(a, a)$ $R(a, a)$ $\boldsymbol{S(a, a)}$ $\neg T(a, a)$ $Adom(a)$ $Adom(a)$

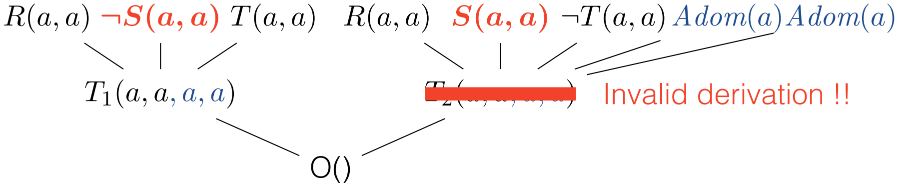
$T_1(a, a, a, a)$ $T_2(a, a, a, a)$

O()

Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts via disequalities

$$
\begin{aligned}
\texttt{Adom}(x) &\leftarrow \text{“active domain of } R\text{, } S\text{, and } T\text{”} \\
\texttt{T}_1(x, y, y, z) &\leftarrow \texttt{R}(x, y), \neg\texttt{S}(y, z), \texttt{T}(z, x). \\
\texttt{T}_2(x, y, y', z') &\leftarrow \texttt{R}(x, y), \texttt{S}(y, z), \neg\texttt{T}(z, x), \texttt{Adom}(y'), \texttt{Adom}(z'), y \neq y'. \\
\texttt{T}_2(x, y, y', z') &\leftarrow \texttt{R}(x, y), \texttt{S}(y, z), \neg\texttt{T}(z, x), \texttt{Adom}(y'), \texttt{Adom}(z'), z \neq z'. \\
\texttt{O}() &\leftarrow \texttt{T}_1(x, y, y', z'), \texttt{T}_2(x, y, y', z'). \\
\texttt{O}() &\leftarrow \texttt{R}(x, y), \texttt{S}(y, z), \texttt{T}(z, x), x \neq y.
\end{aligned}
$$



Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts through disequalities

$$\texttt{Adom}(x) \leftarrow \text{“active domain of } R,\ S, \text{ and } T\text{”}$$
$$\texttt{T}_1(x,y,y,z,z'',x'') \leftarrow \texttt{R}(x,y), \neg\texttt{S}(y,z), \texttt{T}(z,x), \texttt{Adom}(z''), \texttt{Adom}(x''), z \neq z''.$$
$$\texttt{T}_1(x,y,y,z,z'',x'') \leftarrow \texttt{R}(x,y), \neg\texttt{S}(y,z), \texttt{T}(z,x), \texttt{Adom}(z''), \texttt{Adom}(x''), x \neq x''.$$
$$\texttt{T}_2(x,y,y',z',z,x) \leftarrow \texttt{R}(x,y), \texttt{S}(y,z), \neg\texttt{T}(z,x), \texttt{Adom}(y'), \texttt{Adom}(z'), y \neq y'.$$
$$\texttt{T}_2(x,y,y',z',z,x) \leftarrow \texttt{R}(x,y), \texttt{S}(y,z), \neg\texttt{T}(z,x), \texttt{Adom}(y'), \texttt{Adom}(z'), z \neq z'.$$
$$\texttt{O}() \leftarrow \texttt{T}_1(x,y,y',z',z'',x''), \texttt{T}_2(x,y,y',z',z'',x'').$$
$$\texttt{O}() \leftarrow \texttt{R}(x,y), \texttt{S}(y,z), \texttt{T}(z,x), x \neq y.$$

Equivalent to original program and candidate proof-trees have no fringe conflicts

Making Programs Conflict-Free

Crux: Pass information about forbidden facts to other rules via IDBs and exclude conflicts with disequalities

$$\begin{aligned}
\mathtt{Adom}(x) &\leftarrow \text{“active domain of } R\text{, } S\text{, and } T\text{”} \\
\mathtt{T}_1(x,y,y,z,z'',x'') &\leftarrow \mathtt{R}(x,y), \cancel{\neg\mathtt{S}(y,z)}, \mathtt{T}(z,x), \mathtt{Adom}(z''), \mathtt{Adom}(x''), z \neq z''. \\
\mathtt{T}_1(x,y,y,z,z'',x'') &\leftarrow \mathtt{R}(x,y), \cancel{\neg\mathtt{S}(y,z)}, \mathtt{T}(z,x), \mathtt{Adom}(z''), \mathtt{Adom}(x''), x \neq x''. \\
\mathtt{T}_2(x,y,y',z',z,x) &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \cancel{\neg\mathtt{T}(z,x)}, \mathtt{Adom}(y'), \mathtt{Adom}(z'), y \neq y'. \\
\mathtt{T}_2(x,y,y',z',z,x) &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \cancel{\neg\mathtt{T}(z,x)}, \mathtt{Adom}(y'), \mathtt{Adom}(z'), z \neq z'. \\
\mathtt{O}() &\leftarrow \mathtt{T}_1(x,y,y',z',z'',x''), \mathtt{T}_2(x,y,y',z',z'',x''). \\
\mathtt{O}() &\leftarrow \mathtt{R}(x,y), \mathtt{S}(y,z), \mathtt{T}(z,x), x \neq y.
\end{aligned}$$

Equivalent to original program, even without negated atoms

Negation-Bounded Datalog

When does such a rewriting exist?

Negation-Bounded Datalog

When does such a rewriting exist?

- For all programs with a **bounded number of forbidden facts** in their proof-tree fringes;
  - [K, Koch, 2020]
- For all programs with a **bounded number of required facts** in their proof-tree fringes.
  - Symmetric case
- For all **programs without non-equalities**
  - ▶ Because Sp-Datalog $\cap$ Monotone $=$ Sp-Datalog $\cap$ Hom
  - ▶ Sp-Datalog $\cap$ Hom $=$ Pos-Datalog [Feder, Vardi, 2003]

Roadmap

- Context
- From Monotone to Negation-Free Queries
- **Non-rewritable Monotone Queries**
- Coordination-Freedom and System Knowledge
- Open Problems

Counterexample Ingredients

Heavily based on: [Rudolph, Thomazo 2015]

Definition

Given a graph, the **perfect matching problem** asks if there is a subset M of its edges such that every vertex of the graph is incident to **precisely one** edge in M.

- A polynomial-time computable query.
- All polynomial-time computable queries are expressible in order-invariant SP-datalog [Papadimitriou 1985]
  - Succ$[2]$ encoding some linear order over adom;
  - Min$[1]$ and Max$[1]$ denoting minimal and maximal element.

Separating Query

Input: instance over $\{\text{Edge}[2], \text{Succ}[2], \text{Min}[1], \text{Max}[1]\}$.

Output:

- **True** if Succ, Min, Max encode an **inconsistency**
- **True** if Succ, Min, Max **contain** a complete total order and **the graph induced by the active domain elements between Min and Max in Next has a perfect matching**.
- **False** ($\emptyset$) otherwise.

Sp-Datalog($\neq$) query | monotone

Separating Query

Input: instance over $\{\text{Edge}[2], \text{Succ}[2], \text{Min}[1], \text{Max}[1]\}$.

Output:

- **True** if Succ, Min, Max encode an **inconsistency**
  - Some negation-free rules
- **True** if Succ, Min, Max **contain** a complete total order and **the graph induced by the active domain elements between Min and Max in Next has a perfect matching**.
  - Some negation-free rules + order invariant SP-Datalog program with Succ, Min, Max given as part of input
- **False** ($\emptyset$) otherwise.

Sp-Datalog($\neq$) query | monotone

Separating Query

Input: instance over $\{\text{Edge}[2], \text{Succ}[2], \text{Min}[1], \text{Max}[1]\}$.

Output:

- **True** if Succ, Min, Max encode an **inconsistency**
  - Some negation-free rules
  - We cannot fix such an inconsistency by adding more facts
- **True** if Succ, Min, Max **contain** a complete total order and **the graph induced by the active domain elements between Min and Max in Next has a perfect matching**.
  - Some negation-free rules + order invariant SP-Datalog program with Succ, Min, Max given as part of input
  - We cannot undo existence of a perfect matching by adding more facts
- **False** ($\emptyset$) otherwise.

Sp-Datalog($\neq$) query | monotone

Separating Query $\notin$ Pos-Datalog$(\neq)$

Theorem ([Razborov 85])

No family of monotone Boolean circuits exists that answer the perfect matching problem and has circuits of polynomial size in the number of input gates.

Claim: If there is a Pos-Datalog$(\neq)$ program P expressing the separating query, then there is a Boolean circuit contradicting the above result.

Proof by construction

For a graph with v vertices, consider a Boolean circuit with v^2 input gates, one per possible edge. The gate is set to 1 iff the edge is in the graph.

1. Consider all groundings of rules in P over domain $\{1, \ldots, v\}$.
2. Assume natural order over these values as interpretation for `First`, `Last`, `Next` and remove all grounded rules that contradict with it.
3. Remove all facts over interpreted relation names.
4. Make copies of rules organized in strata. First strata copies all EDBS to index 0. For next stratum all atoms in body have index i and those in heads have index $i + 1$. We need only as many strata as we need rounds to finish programs over domain $\{1 \ldots v\}$.
5. EDB atoms $=$ input gates, rules are AND gates whose inputs are the gates whose output represents body atoms; IDBs become OR gates whose inputs are the gates represented by rules that produce the ground atoms.

Roadmap

- Context
- From Monotone to Negation-Free Queries
- Non-rewritable Monotone Queries
- **Coordination-Freedom and System Knowledge**
- Open Problems

A More Fine-Grained Answer to the CALM Conjecture

Systems in which data is **arbitrarily** partitioned.

Theorem ([Ameloot, Neven, Van den Bussche, 2011])

$\mathcal{C}_q(\texttt{Id} + \texttt{All}) \cap \text{Coordination-Free} = \text{Monotone}$

A More Fine-Grained Answer to the CALM Conjecture

Systems in which data is **arbitrarily** partitioned.

Theorem ([Ameloot, Neven, Van den Bussche, 2011])

$\mathcal{C}_q(\texttt{Id} + \texttt{All}) \cap \text{Coordination-Free} = \text{Monotone}$

Systems in which data is partitioned using a **tuple-based** policy.

- For every possible tuple in the database there is a node that knows it would store that tuple if it were part of the database

Theorem ([Ameloot, K, Neven, Zinn, 2014])

$\mathcal{C}_q(\texttt{Id} + \texttt{All} + \texttt{Policy}) \cap \text{Coordination-Free} = \text{Semi-Monotone}$

A More Fine-Grained Answer to the CALM Conjecture

Systems in which data is **arbitrarily** partitioned.

Theorem ([Ameloot, Neven, Van den Bussche, 2011])

$\mathcal{C}_q(\mathtt{Id} + \mathtt{All}) \cap \text{Coordination-Free} = \text{Monotone}$

Theorem ([Ameloot, K, Neven, Zinn, 2014])

$\mathcal{C}_q(\mathtt{Id} + \mathtt{All} + \mathtt{Policy}) \cap \text{Coordination-Free} = \text{Semi-Monotone}$

Systems in which data is partitioned using a **value-based** strategy.

- For every possible data value in the database there is a node that knows it would store all tuples in the database containing that data value.

Theorem ([Ameloot, K, Neven, Zinn, 2014])

$\mathcal{C}_q(\mathtt{Id} + \mathtt{All} + \mathtt{AdomPolicy}) \cap \text{Coordination-Free} = \text{Disjoint-Monotone}$

From Queries to Behaviors

Definition

A **distributed behavior** is a **non-deterministic** mapping from distributed instances to distributed instances

From Queries to Behaviors

Definition

A **distributed behavior** is a **non-deterministic** mapping from distributed instances to distributed instances

R()@1
R()@2
R()@3

— Leader election —

L(3)@1
L(3)@2
L(3)@3

L(2)@1
L(2)@2
L(2)@3

L(1)@1
L(1)@2
L(1)@3

(Queries are specific types of distributed behaviors)

Configuration Constraints

Information made available to nodes in the form of interpreted relations or functions defined as a **behavior**.

Configuration Constraints

Information made available to nodes in the form of interpreted relations or functions defined as a **behavior**.
[Ameloot, Neven, Van den Bussche 2011]

- `Id`: relation containing the unique ID of the node at hand;
- `All`: relation containing all IDs of nodes in the network;

Configuration Constraints

Information made available to nodes in the form of interpreted relations or functions defined as a **behavior**.

[Ameloot, Neven, Van den Bussche 2011]

- `Id`: relation containing the unique ID of the node at hand;
- `All`: relation containing all IDs of nodes in the network;

[Ameloot, K, Neven, Zinn 2014]

- `Policy`: function mapping a fact to true iff the node is responsible for it;
- `AdomPolicy`: similar as previous but based on data values of tuples;

Configuration Constraints

Information made available to nodes in the form of interpreted relations or functions defined as a **behavior**.

[Ameloot, Neven, Van den Bussche 2011]

- `Id`: relation containing the unique ID of the node at hand;
- `All`: relation containing all IDs of nodes in the network;

[Ameloot, K, Neven, Zinn 2014]

- `Policy`: function mapping a fact to true iff the node is responsible for it;
- `AdomPolicy`: similar as previous but based on data values of tuples;

Many other options:

- `L`: relation containing the ID of some node considered leader in the network;
- `Order`: function mapping two data values a, b on true iff $a < b$ according to some coordinated order.
- …

(can be combined)

Computing Behaviors

Is a certain behavior computable in our model for a specific set of constraints?

Is leader election computable without configuration constraints?

L(3)@1
L(3)@2
L(3)@3

R()@1
R()@2
R()@3

L(2)@1
L(2)@2
L(2)@3

L(1)@1
L(1)@2
L(1)@3

Computing Behaviors

Is a certain behavior computable in our model for a specific set of constraints?

Is leader election computable with `Id`?

```
R()@1     Id(1)@1
R()@2  +  Id(2)@2  ──────────  ↗ L(3)@1, L(3)@2, L(3)@3
R()@3     Id(3)@3             → L(2)@1, L(2)@2, L(2)@3
                              ↘ L(1)@1, L(1)@2, L(1)@3
```

Computing Behaviors

Is a certain behavior computable in our model for a specific set of constraints?

Is leader election computable with `Id + All`?

```
                                                                   L(3)@1
                                                                   L(3)@2
                                                                   L(3)@3
R()@1     Id(1)@1     All(1)@1,All(2)@1,All(3)@1                 ↗              L(2)@1
R()@2  +  Id(2)@2  +  All(1)@2,All(2)@2,All(3)@2  ————————————  ————————————→  L(2)@2
R()@3     Id(3)@3     All(1)@3,All(2)@3,All(3)@3                 ↘              L(2)@3
                                                                   L(1)@1
                                                                   L(1)@2
                                                                   L(1)@3
```

Computing Behaviors

Is a certain behavior computable in our model for a specific set of constraints?

Is leader election computable with `Id + All + Order`?

$$
\begin{array}{c}\texttt{R()@1}\\ \texttt{R()@2}\\ \texttt{R()@3}\end{array}
+
\begin{array}{c}\texttt{Id(1)@1}\\ \texttt{Id(2)@2}\\ \texttt{Id(3)@3}\end{array}
+
\begin{array}{c}\texttt{All(1)@1,All(2)@1,All(3)@1}\\ \texttt{All(1)@2,All(2)@2,All(3)@2}\\ \texttt{All(1)@3,All(2)@3,All(3)@3}\end{array}
+
\begin{array}{c}\leq\texttt{@1}\\ \leq\texttt{@2}\\ \leq\texttt{@3}\end{array}
-
\begin{array}{l}
\nearrow \begin{array}{c}\texttt{L(3)@1}\\ \texttt{L(3)@2}\\ \texttt{L(3)@3}\end{array}\\
\longrightarrow \begin{array}{c}\texttt{L(2)@1}\\ \texttt{L(2)@2}\\ \texttt{L(2)@3}\end{array}\\
\searrow \begin{array}{c}\texttt{L(1)@1}\\ \texttt{L(1)@2}\\ \texttt{L(1)@3}\end{array}
\end{array}
$$

Computing Behaviors

Definition

$\mathcal{C}(\chi) =$ the set of all behaviors computable in the model with constraint χ.

- $\mathcal{C}(\texttt{Id} + \texttt{All})$ contains all (computable) **queries**;
- $\mathcal{C}(\texttt{Id} + \texttt{All} + \texttt{Order})$ equals all (computable) **behaviors**;

Coordination-Freedom for Behaviors

Definition

An algorithm is **coordination-free** if for some (ideal) partitioning of the input data, the algorithm finds the correct output before doing any **communication**.

Coordination-Freedom for Behaviors

Definition

An algorithm is **coordination-free** if for some (ideal) partitioning of the input data, the algorithm finds the correct output before doing any **communication**.

This definition only makes sense for behaviors that are data-partitioning independent**

Semantics Versus Syntactic Transducers

Coordination-Free computability = Computability in restricted variant of the model.

Theorem ([Ameloot, Neven, Van den Bussche, 2011])

- $\mathcal{C}_{\text{queries}}(\texttt{Id} + \texttt{All}) \cap \text{Coordination-Free} = \mathcal{C}_{\text{queries}}(\texttt{Id})$

Coordination Free = Ability to compute without access to `All`.

Semantics Versus Syntactic Transducers

Coordination-Free computability = Computability in restricted variant of the model.

Theorem ([Ameloot, Neven, Van den Bussche, 2011])

- $\mathcal{C}_{\text{queries}}(\texttt{Id} + \texttt{All}) \cap \text{Coordination-Free} = \mathcal{C}_{\text{queries}}(\texttt{Id})$

Theorem ([Ameloot, K, Neven, Zinn, 2014])

- $\mathcal{C}_{\text{q}}(\texttt{Id} + \texttt{All} + \texttt{Policy}) \cap \text{Coordination-Free} = \mathcal{C}_{\text{q}}(\texttt{Id} + \texttt{Policy})$
- $\mathcal{C}_{\text{q}}(\texttt{Id} + \texttt{All} + \texttt{AdomPolicy}) \cap \text{Coordination-Free} = \mathcal{C}_{\text{q}}(\texttt{Id} + \texttt{AdomPolicy})$

Coordination Free = Ability to compute without access to `All`.

Coordination / Computability

Definition

We call a behavior **coordinating-free** if it is in the set $\mathcal{C}(\mathtt{Id} + \mathtt{Ord})$

Definition

We call a behavior **weakly coordinating** if it is in a set $\mathcal{C}(\mathtt{Id} + \mathtt{Something})$ with property that there is no termination-aware algorithm to compute $\mathtt{All}$.

$$\mathcal{C}(\mathtt{Id} + \mathtt{All} + \mathtt{Order}) = \mathtt{All\ \ behaviors}$$

Monotone Behaviors?

Theorem

$\mathcal{C}(\mathtt{Id} + \mathtt{Ord} + \chi) =$ all χ-monotone behaviors.

Details in [Baccaert, K, 2023]

Monotone Behaviors?

Example:

- Consider the query that takes a graph as input and asks if the graph as an isolated edge.
- Consider $\chi = \mathtt{Id} + \mathtt{Policy} + \mathtt{Min} + \mathtt{Max} + \mathtt{Ord}$ as configuration constraints

Monotone Behaviors?

Example:

- Consider the query that takes a graph as input and asks if the graph as an isolated edge.
- Consider $\chi = \texttt{Id} + \texttt{Policy} + \texttt{Min} + \texttt{Max} + \texttt{Ord}$ as configuration constraints

$E(1,2)$ $E(4,5)$

Output: **True**

Monotone Behaviors?

Example:

- Consider the query that takes a graph as input and asks if the graph as an isolated edge.
- Consider $\chi = \texttt{Id} + \texttt{Policy} + \texttt{Min} + \texttt{Max} + \texttt{Ord}$ as configuration constraints

$$E(1,2)$$
`id(1)`
$$\{(1,2),(2,1),(1,3),(3,1),(1,4),(4,1),(1,5),(5,1),\ldots\}$$
$$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$$

$$E(4,5)$$
`id(3)`
$$\{(2,4),(4,2),(2,5),(5,2),(4,5),(5,4),\ldots\}$$
$$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$$

Output: **True**

Monotone Behaviors?

Example:

- Consider the query that takes a graph as input and asks if the graph as an isolated edge.
- Consider $\chi = \texttt{Id} + \texttt{Policy} + \texttt{Min} + \texttt{Max} + \texttt{Ord}$ as configuration constraints

$E(1,2)$	$E(2,3)E(3,4)$	$E(4,5)$
`id(1)`	`id(2)`	`id(3)`
$\{(1,2),(2,1),(1,3),(3,1),$ $(1,4),(4,1),(1,5),(5,1),\ldots\}$	$\{(2,3),(3,2),(3,4),(4,3)$ $(3,5),(5,3),\ldots\}$	$\{(2,4),(4,2),(2,5),(5,2)$ $(4,5),(5,4),\ldots\}$
$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$	$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$	$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$

Output: **False**

Monotone Behaviors?

Example:

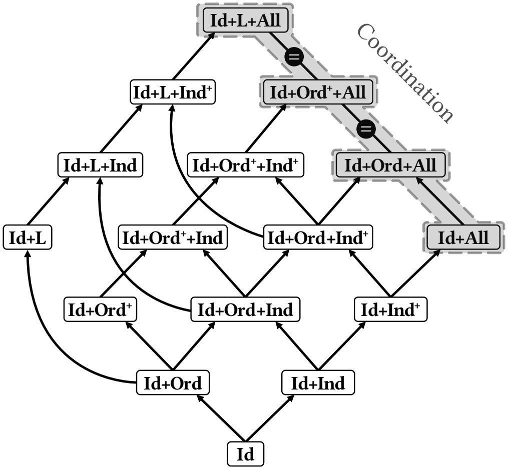
- Consider the query that takes a graph as input and asks if the graph as an isolated edge.
- Consider $\chi = \texttt{Id} + \texttt{Policy} + \texttt{Min} + \texttt{Max} + \texttt{Ord}$ as configuration constraints

$E(1,2)$	$E(2,3)E(3,4)$	$E(4,5)$
`id(1)`	`id(2)`	`id(3)`
$\{(1,2),(2,1),(1,3),(3,1),$ $(1,4),(4,1),(1,5),(5,1),\ldots\}$	$\{(2,3),(3,2),(3,4),(4,3)$ $(3,5),(5,3),\ldots\}$	$\{(2,4),(4,2),(2,5),(5,2)$ $(4,5),(5,4),\ldots\}$
$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$	$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$	$\mathbf{1} \leq 2 \leq 3 \leq 4 \leq \mathbf{5}$

Output: **False**

$\Rightarrow$ The isolated edge query is not χ-monotone $\Rightarrow$ not in $\mathcal{C}(\chi)$

An Overview


Id+L+All
Coordination
Id+L+Ind⁺
Id+Ord⁺+All
Id+L+Ind
Id+Ord⁺+Ind⁺
Id+Ord+All
Id+L
Id+Ord⁺+Ind
Id+Ord+Ind⁺
Id+All
Id+Ord⁺
Id+Ord+Ind
Id+Ind⁺
Id+Ord
Id+Ind
Id

Roadmap

- Context
- From Monotone to Negation-Free Queries
- Non-rewritable Monotone Queries
- Coordination-Freedom and System Knowledge
- **Open Problems**

Open Questions

- Are there other techniques to eliminate negation in monotone Datalog programs?
- Is there an elegant semantic definition of coordination-freedom for behaviors that matches our definition based on the `All` relation?
- The definitions of behavior and constraint are all semantic, are there suitable languages that can be used to define them in a more elegant way?

More information: bas.ketsman@vub.be