Remaining CALM in Declarative Networking

Frank Neven
Hasselt University
Introduction
Declarative Networking

- Application of database query-language and processing techniques to the domain of networking

- **Applications:** routing protocols, overlay networks, distributed state management

- **Datalog-based** languages: NDLog (Overlog)

- **Strengths:** reduce program size, automatic optimization, program checking and debugging

[Loo et al. 2006]
Declarative Networking

- **Datalog-based languages** for parallel and distributed computing
- Declarative
- Datalog parallellizes naturally
- Increased interest in data-centric programming languages

**Cloud computing** setting:
- asynchronous communication via messages which can be arbitrarily delayed but not lost

**CALM-conjecture**:
- suggests link between logical monotonicity and eventual consistency without the need for coordination

[Hellerstein, 2010]
CALM by Example

Query: select all triangles

\[ Q(x, y, z) : -E(x, y), E(y, z), E(z, x), x \neq y, y \neq z, z \neq x \]
Example

Query: select all triangles

- eventually consistent
- no coordination

Naive algorithm:
- Broadcast all data
- Output triangles whenever new data arrives
Query: select all open triangles

\[ Q(x, y, z) : \neg E(x, y), E(y, z), \neg E(z, x) \]
Example

Query: select all open triangles

computing node

A

computing node

B

computing node

C

requires global coordination

input graph
CALM-conjecture

“A query has a coordination-free and eventually consistent execution strategy iff the query is monotone”

[±Hellerstein, 2010]

Ameloot, N., Van den Bussche (2011): TRUE

Zinn, Green, Ludäscher (2012): FALSE when nodes have more knowledge on how data is distributed
CALM-conjecture

“A query has a coordination-free and eventually consistent execution strategy under data distribution policy $X$ iff the query is $X'$-monotone”

Ameloot, N., Van den Bussche (2011): TRUE
Zinn, Green, Ludäscher (2012): FALSE when nodes have more knowledge on how data is distributed
Ameloot, Ketsman, N., Zinn (2014): TRUE when also refining monotonicity
(Distributed) Datalog on the rise

- Distributed data management has been studied since the earliest days of databases
- Many supporters: e.g., active XML & Webdamlog

- Parallel and distributed evaluation of Datalog
- **Big data**: Massively Parallel Communication model, MapReduce

Knowledge out there on the Web
Serge Abiteboul
(Tuesday, March 25)

This talk: **Datalog & Coordination**
Overview

• Introduction

• CALM-conjecture
  • Relational Transducer Networks
  • Coordination
  • Monotonicity & Datalog
  • CALM-theorem

• Two reformulations of CALM

• Discussion
Relational Transducer Networks
Setting the stage

Infinite set of data values
\[ \text{dom} = \{a, b, c, \ldots, 1, 2, 3, \ldots\} \]

Relational schema \( \sigma \)
\[ \{E\} \]

Fact
\[ E(a, b) \]

Database instance
\[ I = \{E(a, b), E(b, c), E(c, a), E(c, d)\} \]

\[ \text{adom}(I) = \{a, b, c, d\} \]
Setting the stage

**Query**  
generic computable mapping from instances over an input schema to instances over an output schema

**Genericity**  
\( Q(\pi(I)) = \pi(Q(I)) \) for every permutation \( \pi \) of \( \text{dom} \)
Relational Transducer Networks

[Abiteboul, Vianu, Fordham, Yesha, 2000]
Relational Transducer Networks

[Ameloot, N., Van den Bussche, 2011]
Relational Transducer Networks

Purpose: compute queries

$\mathcal{N}$ network
Relational Transducer Networks

\[ \mathcal{N} \]

Network

Database instance \( \mathbf{I} \)

Horizontal distribution:
- function \( H \) mapping each node \( x \) to a database instance
- \( \mathbf{I} = \bigcup_{x \in \mathcal{N}} H(x) \)
Relational Transducer Networks

[Ameloot, N., Van den Bussche, 2011]
Relational Transducer Networks

A generic mapping from relational state + some input messages to relational state + output messages + output facts.

relational state

\[ H(x) \]

input database

aux relations

\[ \gamma_{sys} \]

id, All

input buffer

relational state

buffer

msg in

\[ \gamma_{mem} \]

input database buffer

\[ x \]

msg out

inflationary

transducer

program

node
Semantics: Transition System

- **Configuration**: input buffer and relational state for every computing node

- **Transition**:
  - at single node $x$ (active node), reads some input messages (updates own buffer)
  - updates its own relational state, and sends output messages (updating buffers of others)

- **Run**: infinite sequence of transitions (captures nondeterminism)

- **Output** of run: union of all output facts

- **Fairness**: only fair runs

Asynchronous communication via messages which can be arbitrarily delayed but not lost
A transducer $\Pi$ computes a query $Q$ if

- for all networks $\mathcal{N}$, network independent
- for all databases $I$, consistency requirement
- for all horizontal distributions $H$ of $I$, and independent of way
- for every run $\mathcal{R}$ of $\Pi$, data is distributed

$$\text{out}(\mathcal{R}) = Q(I).$$
Correctness

Tom Ameloot
Deciding Correctness with Fairness for Simple Transducer Networks
(Today: 16-18.00)

Best Student Paper

Tom Ameloot.
Declarative Networking: Recent Theoretical Work on Coordination, Correctness, and Declarative Semantics.

Sigmod Record, 2014.
Computing all queries

Theorem
Relational transducer networks compute every query

Idea
Gather all data. Needs coordination (messages can be delayed). Locally evaluate query.

\[ \ldots, (R(\bar{d}_2), x), (R(\bar{d}_1), x) \]
Computing all queries

**Theorem**
Relational transducer networks compute every query

**Idea**
Gather all data. Needs coordination (messages can be delayed). Locally evaluate query.

Diagram:
- Two nodes labeled $x$ and $y$, connected with an arrow indicating "acknowledge every tuple".
Computing all queries

**Theorem**
Relational transducer networks compute every query.

**Idea**
Gather all data. **Needs coordination** *(messages can be delayed)*. Locally evaluate query.

global coordination
Computing all queries

**Theorem**
Relational transducer networks compute every query

**Idea**
Gather all data. Needs coordination (messages can be delayed). Locally evaluate query.

global coordination

need to know every node in the network: relation All
Coordination
Coordination: intuition

Coordination

• Global consensus
• Reduces parallelism

Coordination-free

• Embarrassingly parallel execution
• Need communication to transfer data

Goal

• Separate data-communication from coordination-communication
Coordination

Definition

$\Pi$ is \textit{coordination-free} if for all inputs $I$ there is a distribution on which $\Pi$ computes $Q(I)$ by \textit{only} heartbeat transitions.

Intuition

- On \textit{ideal} distribution: no communication
- On \textit{non-ideal} distribution: only communication for data transfer, not for coordination
Coordination

Example: output all triangles

Transducer Program:
- Broadcast local edges
- Output a triangle when one can be derived

Correctness:
- network-independent
- distribution-independent
- independent of run
- computes correct output

Coordination-free:
- Ideal distribution: whole database on one/every node
- Query is computed by only heartbeat transitions
Coordination

Definition

\[ \mathcal{F}_0 = \text{queries distributedly computed by coordination-free transducers} \]

Definition

\[ \mathcal{A}_0 = \text{queries distributedly computed by transducers without using All} \]

Theorem

\[ \mathcal{F}_0 = \mathcal{A}_0 \]

[Ameloot, N., Van den Bussche, 2011]
Monotonicity & Datalog
Monotonicity

Definition

A query $Q$ is **monotone** if $Q(I) \subseteq Q(J)$ for all database instances $I$ and $J$ with $I \subseteq J$.

Equivalent to

A query $Q$ is **monotone** if $Q(I) \subseteq Q(I \cup J)$ for all database instances $I$ and $J$. 

Denote the class of **monotone queries** by $\mathcal{M}$.
Monotonicity

Definition

A query $Q$ is monotone if $Q(I) \subseteq Q(I \cup J)$ for all database instances $I$ and $J$.

Example

select triangles in graph

$I$

$Q(I)$
Monotonicity

Definition

A query $Q$ is monotone if $Q(I) \subseteq Q(I \cup J)$ for all database instances $I$ and $J$.

Example    select triangles in graph

$I \cup J$ | $Q(I \cup J)$
Monotonicity

Definition

A query $Q$ is **monotone** if $Q(I) \subseteq Q(I \cup J)$ for all database instances $I$ and $J$.

Example

select *open* triangles in graph is **not** monotone
Monotonicity

Definition

A query $Q$ is **monotone** if $Q(I) \subseteq Q(I \cup J)$ for all database instances $I$ and $J$.

Example

select *open* triangles in graph is **not** monotone
Datalog(\neq)

Transitive closure

\[ O(x, y) \leftarrow E(x, y) \]
\[ O(x, y) \leftarrow E(x, z), O(z, y) \]

Fixpoint Semantics

\[ O^0 = \emptyset \]
\[ O^i = E \cup (E \circ O^{i-1}) \]

I

step 0
Datalog(\neq)

Transitive closure

\[ O(x, y) \leftarrow E(x, y) \]
\[ O(x, y) \leftarrow E(x, z), O(z, y) \]

Fixpoint Semantics

\[
\begin{align*}
O^0 &= \emptyset \\
O^i &= E \cup (E \circ O^{i-1})
\end{align*}
\]

I

\[ \begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \\
\end{array} \]

step 1

\[ \begin{array}{c}
\circ \quad \circ \quad \circ \quad \circ \\
\end{array} \]
Datalog(≠)

Transitive closure

\[ O(x, y) \leftarrow E(x, y) \]
\[ O(x, y) \leftarrow E(x, z), O(z, y) \]

Fixpoint Semantics

\[
\begin{align*}
O^0 &= \emptyset \\
O^i &= E \cup (E \circ O^{i-1})
\end{align*}
\]
**Datalog(\(\neq\))**

Transitive closure

\[
\begin{align*}
O(x, y) & \leftarrow E(x, y) \\
O(x, y) & \leftarrow E(x, z), O(z, y)
\end{align*}
\]

Fixpoint Semantics

\[
\begin{align*}
O^0 & = \emptyset \\
O^i & = E \cup (E \circ O^{i-1})
\end{align*}
\]

\[
\text{step 3 = step 4}
\]
Monotonicity

Fact

\[ \text{Datalog}(\neq) \subseteq \mathcal{M} \]

Theorem

\[ \text{Datalog}(\neq) \not\subseteq \mathcal{M} \cap \text{PTIME} \]

[Afrati, Cosmadakis, Yannakakis, 1994]

Theorem

\[ \text{Datalog}(\neq) \text{ with value invention } = \mathcal{M} \]

[Cabibbo, 1998]
Datalog with value invention

ILOG⁻: Datalog⁻ + Skolem functors

\[ \begin{align*}
R(\ast, x_1, x_2) & \leftarrow E(x_1, x_2) \\
R(\ast, x_1, x_2) & \leftarrow R(x_1, z, y), E(y, x_2)
\end{align*} \]

\[ \text{Skol}(P) \]

\[ \begin{align*}
R(f_R(x_1, x_2), x_1, x_2) & \leftarrow E(x_1, x_2) \\
R(f_R(x_1, x_2), x_1, x_2) & \leftarrow R(x_1, z, y), E(y, x_2)
\end{align*} \]

\[ P(I) \]

\[
\begin{array}{c|c}
R & P(I) \\
\hline
(f_R(a, b), a, b) \\
(f_R(b, c), b, c)
\end{array}
\]

[The content is about Datalog with value invention, ILOG⁻, and Skolem functors, with examples and definitions provided in the text.]
Datalog with value invention

\[ \text{ILOG}^- : \text{Datalog}^- + \text{Skolem functors} \]

\[ \text{invention position} \]

\[ R(x_1, x_2) \leftarrow E(x_1, x_2) \]

\[ R(x_1, x_2) \leftarrow R(x_1, z, y), E(y, x_2) \]

\[ \text{Skol}(P) \]

\[ R(f_R(x_1, x_2), x_1, x_2) \leftarrow E(x_1, x_2) \]

\[ R(f_R(x_1, x_2), x_1, x_2) \leftarrow R(x_1, z, y), E(y, x_2) \]

\[ \textbf{I} \]

\[ E(a, b), E(b, c) \]

\[ \text{P(I)} \]

\[ \begin{array}{c|c}
R & P(\text{I}) \\
\hline
(f_R(a, b), a, b) \\
(f_R(b, c), b, c) \\
(f_R(f_R(a, b), c), f_R(a, b), c) \\
\end{array} \]
Datalog with value invention

• Output of ILOG$^-$ can be infinite —> undefined

• An ILOG$^-$ program is safe when the output contains no invented values —> undecidable

• wILOG$^-$ = weakly safe ILOG$^-$

• syntactic restriction that ensures safety
\( \text{wILOG}^- = \text{computable queries} \)

1. Encoding.
   • A \text{wILOG}^- program \( P \) encodes all enumerations of \( I \) in parallel.
   • Skolem functors are used to represent these enumerations as lists.
   • Negation detects whether a partial enumeration is complete.

2. Simulation.
   • The query is represented by an order independent domain Turing Machine \( M \) (Hull, Su, 1993).
   • A positive \text{wILOG} program \( P \) \( M \) simulates \( M \) on every complete enumeration of \( I \).

3. Decoding.
   • A positive \text{wILOG} program \( P \) \( \text{decode} \) transforms the output of \( M \) on each input to an instance.
   • Output is the union of all generated facts.
   • Correctness follows from genericity.
\[ \text{wILOG}(\neq) = \mathcal{M} \]  

[Cabibbo, 1998]

**Algorithm**

1. **Encoding.**
   
   A wILOG(\neq) program generates all partial enumerations of the input instance

2. **Simulation.**

3. **Decoding.**

Correctness follows from **monotonicity**:

- \( Q(I') \subseteq Q(I) \), for all \( I' \subseteq I \) and
- \( I \) is a partial enumeration of itself.
KEEP CALM AND CARRY ON

CALM-theorem
CALM-conjecture

“A query has a coordination-free and eventually consistent execution strategy iff the query is monotone”

Theorem $F_0 = A_0 = M = wILOG(\neq) \supseteq Datalog(\neq)$

[±Hellerstein, 2010]

[Ameloot, N., Van den Bussche, 2011]
\[ \mathcal{M} \subseteq \mathcal{F}_0, \quad \mathcal{M} \subseteq \mathcal{A}_0 \]

\[ Q \in \mathcal{M} \]

**Transducer program**
- Broadcast all input data
- Recompute \( Q \) and output newly derived facts

**Correct** (because of monotonicity)

**Coordination-free** \[ \rightarrow Q \in \mathcal{F}_0 \]

**No use of id or All** \[ \rightarrow Q \in \mathcal{A}_0 \]
Overview

• Introduction

• CALM-conjecture
  • 1st extension of CALM
    • Domain-distinct-monotonicity
    • Semi-positive datalog
    • Policy-based transducer networks
  • 2nd extension of CALM

• Discussion
Domain-distinct-monotonicity
Domain distinct

Definition

A fact \( f \) is **domain distinct** from instance \( I \) when

\[
\text{adom}(f) \not\subseteq \text{adom}(I).
\]

Example

\[
\begin{align*}
\text{adom}(I) &= \{a, b, c, d\} \\
\text{adom}(f) &= \{d, e\} \\
\text{adom}(f') &= \{a, d\}
\end{align*}
\]
Domain distinct

Definition

An instance $J$ is *domain distinct* from instance $I$ when every fact $f \in J$ is domain distinct from $I$.

Example

$\text{adom}(I) = \{a, b, c, d\}$
Definition

A query $Q$ is domain-distinct-monotone if

$$Q(I) \subseteq Q(I \cup J)$$

for all $I$ and $J$ for which $J$ is domain distinct from $I$.

Notation

$\mathcal{M}_{\text{distinct}}$: class of domain-distinct-monotone queries

Observation

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain distinct from $I$
Domain-distinct-monotonicity

**Definition**

A query $Q$ is **domain-distinct-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all $I$ and $J$ for which $J$ is domain distinct from $I$.

**Notation**

$\mathcal{M}_{\text{distinct}}$ : class of **domain-distinct-monotone** queries

**Remark**

$\mathcal{M}_{\text{distinct}}$ : class of queries preserved under **extension**
Domain-distinct-monotonicity

Example select open triangles in graph $\in M_{\text{distinct}}$

$I$

$Q(I)$
Domain-distinct-monotonicity

Example select open triangles in graph $\in M_{\text{distinct}}$

not domain distinct from $I$
Domain-distinct-monotonicity

Example complement of transitive-closure $\not\in \mathcal{M}_{\text{distinct}}$

stratified Datalog

\[
\begin{align*}
TC(x, y) & \leftarrow E(x, y) \\
TC(x, y) & \leftarrow E(x, z), TC(z, y) \\
O(x, y) & \leftarrow \neg TC(x, y), x \neq y
\end{align*}
\]

\{ \text{stratum 1} \}

\begin{align*}
(I) \quad & a \quad \text{b} \\
Q(I) \quad & a \quad b
\end{align*}

stratum 2
Domain-distinct-monotonicity

Example complement of transitive-closure $\not\in \mathcal{M}_{\text{distinct}}$

stratified Datalog $\leftarrow$

\[
\begin{align*}
TC(x, y) & \leftarrow E(x, y) \\
TC(x, y) & \leftarrow E(x, z), TC(z, y) \\
O(x, y) & \leftarrow \neg TC(x, y), x \neq y
\end{align*}
\]

\{ \text{stratum 1} \}

\{ \text{stratum 2} \}

$I \cup J$

$Q(I \cup J)$
Semi-Positive Datalog
Semi-Positive Datalog

**Definition**

SP-Datalog is Datalog with negation restricted to input relations.

**Example**

\[
O(x, y) \leftarrow \neg E(x, y)
\]

\[
O(x, y) \leftarrow \neg E(x, z), O(z, y)
\]

**Example**

\[
Q(x, y, z) \leftarrow E(x, y), E(y, z), \neg E(z, x)
\]

**Observation**

SP-Datalog $\not\in \mathcal{M}$
Semi-Positive Datalog

Theorem 1
\[ \text{SP-Datalog} \subseteq \mathcal{M}_{\text{distinct}} \]
[Afrati, Cosmadakis, Yannakakis, 1994]

Theorem 2
\[ \text{SP-Datalog} \not\subseteq \mathcal{M}_{\text{distinct}} \cap \text{PTIME} \]
[Afrati, Cosmadakis, Yannakakis, 1994]

Theorem 3
\[ \text{SP-wILOG} = \mathcal{M}_{\text{distinct}} \]
[ Cabibbo, 1998]
Induced Subinstance

Definition

For $D \subseteq \text{dom}(I)$, the subinstance induced by $D$, is defined as $I|_D = \{ f \in I \mid \text{dom}(f) \subseteq D \}$.

Example

\[
\begin{align*}
I & \quad E(a, b), E(b, c) \\
D & \quad \{a, b\} \\
I|_D & \quad E(a, b)
\end{align*}
\]

Proposition

For $Q \in \mathcal{M}_{\text{distinct}}$, for all $I$, and $D \subseteq \text{dom}(I)$

\[Q(I|_D) \subseteq Q(I)\].
Algorithm

1. **Encoding.** An SP-wILOG program generates all enumerations of all induced subinstances $I|D$ for all $D \subseteq \text{adom}(I)$

2. **Simulation in parallel** on every $I|D$ by a wILOG($\neq$) program.

3. **Decoding** by a wILOG($\neq$) program.

Correctness follows from **domain-distinct-monotonicity**:

- $Q(I|D) \subseteq Q(I)$, for every $D \subseteq \text{adom}(I)$ and
- $I$ is an induced subinstance of itself.
Policy-aware Transducers
Policy-aware Transducers

[Zinn, Green, Ludäscher, 2012]

**Definition**

A distribution policy \( P \) for \( \sigma \) and \( \mathcal{N} \) is a total function from \( \text{facts}(\sigma) \) to the power set of \( \mathcal{N} \).

**Example**

- **distribution policy**
  
  \[
  P_1(E(a, b)) = \begin{cases} 
  \{1\} & \text{if } a \text{ is odd} \\
  \{2\} & \text{otherwise}
  \end{cases}
  \]

- **database instance**
  
  \( I = \{E(1, 3), E(3, 4), E(4, 6)\} \)

- **induced horizontal distribution**
  
  \[
  \begin{align*}
  \{1 \mapsto \{E(1, 3), E(3, 4)\}\} \\
  \{2 \mapsto \{E(4, 6)\}\}
  \end{align*}
  \]

  computing nodes
Policy-aware Transducers

[Zinn, Green, Ludäscher, 2012]

Definition

A policy-aware transducer is a transducer with the following system relations

\[ \Upsilon_{\text{sys}} = \{ \text{Id}^{(1)}, \text{All}^{(1)}, \text{MyAdom}^{(1)} \} \cup \{ \text{policy}^{(k)}_R \mid R^{(k)} \in \Upsilon_{\text{in}} \} \]

Example

\[ I = \{ E(1, 3), E(3, 4), E(4, 6) \} \]

\[ P_1( E(a, b) ) = \begin{cases} \{ 1 \} & \text{if } a \text{ is odd} \\ \{ 2 \} & \text{otherwise} \end{cases} \]

\[ H(I) \rightarrow \{ E(1, 3), E(3, 4) \} \]

\[ \text{MyAdom} \rightarrow \{ 1, 3, 4 \} \]

\[ \text{policy}_{E(1, 4)} \checkmark \]

\[ \text{policy}_{E(1, 6)} \times \]
Policy-aware Transducers

[Zinn, Green, Ludäscher, 2012]

**Definition**

A policy-aware transducer is a transducer with the following system relations

\[
\gamma_{sys} = \{\text{Id}^{(1)}, \text{All}^{(1)}, \text{MyAdom}^{(1)}\} \cup \{\text{policy}_{R}^{(k)} | R^{(k)} \in \gamma_{in}\}
\]

**Example**

\[
I = \{E(1, 3), E(3, 4), E(4, 6)\} \quad P_{1}(E(a, b)) = \begin{cases} 
\{1\} & \text{if } a \text{ is odd} \\
\{2\} & \text{otherwise}
\end{cases}
\]

\[
H(I) \rightarrow \{E(1, 3), E(3, 4)\} \quad \{E(4, 6)\}
\]

\[
\text{MyAdom} \rightarrow \{1, 3, 4, 6\} \quad \{E(4, 6)\} \quad \{4, 6\}
\]

policy_{E(1, 6)}
Policy-aware Transducers

Definition

A policy-aware transducer $\Pi$ computes a query $Q$ if

- for all networks $\mathcal{N}$,
- for all databases $I$,
- for all distribution policies $\mathbf{P}$, and
- for every run $\mathcal{R}$ of $\Pi$,

$$\text{out}(\mathcal{R}) = Q(I).$$

Definition

$\mathcal{F}_1 =$ set of queries which are distributedly computed by policy-aware coordination-free transducers
Consistency And Logical Monotonicity

CALM-conjecture

“A query has a coordination-free and eventually consistent execution strategy under distribution policies iff the query is domain-distinct-monotone”

Theorem $F_1 = M_{\text{distinct}}$

[Ameloot, Ketsman, N, Zinn, 2014]
\[ M_{\text{distinct}} \subseteq F_1 \]

**Observation**

A subset \( C \subseteq \text{dom}(I) \) is **complete** at a node \( x \), when \( x \) knows for every fact \( f \) with \( \text{dom}(f) \subseteq C \) whether \( f \in I \) or \( f \notin I \).

\[
I_{|C} = \{ f \in I \mid \text{dom}(f) \subseteq C \} \quad \rightarrow \quad x \text{ has full knowledge on } I_{|C}
\]

By domain-distinct-monotonicity: \( Q(I_{|C}) \subseteq Q(I) \)

When \( C = \text{dom}(I) \), \( Q(I_{|C}) = Q(I) \)

**Intuition**

- Broadcast all data (pos and neg facts)
- Evaluate query on **complete** sets
Extended CALM-theorem

\[
\text{Datalog}(\neq) \subseteq \text{wILOG}(\neq) = \mathcal{M} = \mathcal{F}_0 = \mathcal{A}_0
\]

\[
\text{SP-Datalog} \not\subseteq \text{SP-wILOG} = \mathcal{M}_{\text{distinct}} = \mathcal{F}_1 = \mathcal{A}_1
\]
Overview

• Introduction
• CALM-conjecture
• 1st extension of CALM
• 2nd extension of CALM
  • Domain-disjoint-monotonicity
  • Semi-connected datalog
  • Domain-guided transducer networks
• Discussion
Domain-disjoint-monotonicity
Domain disjoint

Definition

An instance $J$ is *domain disjoint* from instance $I$ when $\text{adom}(I) \cap \text{adom}(J) = \emptyset$.

Example

$I$

$J$

$J'$
Domain-disjoint-monotonicity

Definition

A query $Q$ is **domain-disjoint-monotone** if

$$Q(I) \subseteq Q(I \cup J)$$

for all instances $I$ and $J$ for which $J$ is domain disjoint from $I$.

Notation

$\mathcal{M}_{\text{disjoint}}$ : class of **domain-disjoint-monotone** queries
Domain-disjoint-monotonicity

Example complement of transitive-closure $\in M_{\text{disjoint}}$

$I$

$TC(x, y) \leftarrow E(x, y)$
$TC(x, y) \leftarrow E(x, z), TC(z, y)$
$O(x, y) \leftarrow \neg TC(x, y), x \neq y$

$Q(I)$
Example no closed triangle \( \not\in \mathcal{M}_{\text{disjoint}} \)

\[
\begin{align*}
T(x, y, z) & \iff E(x, y), E(y, z), E(z, x) \\
O(u, v) & \iff E(u, v), \neg T(x, y, z)
\end{align*}
\]
Domain-disjoint-monotonicity

Example no closed triangle $\not\in \mathcal{M}_{\text{disjoint}}$

$I \cup J$

$T(x, y, z) \leftarrow E(x, y), E(y, z), E(z, x)$

$O(u, v) \leftarrow E(u, v), \neg T(x, y, z)$

$Q(I \cup J)$
Connected Components

Notation

\[ \text{co}(\mathbf{I}) = \text{connected components of } \mathbf{I} \]

Proposition

For \( Q \in \mathcal{M}_{\text{disjoint}} \), for all \( \mathbf{I} \), for \( C \subseteq \text{co}(\mathbf{I}) \),

\[ Q(C) \subseteq Q(I). \]

\[ \mathbf{I} = C \cup (I \setminus C) \]

domain disjoint
Semi-connected Datalog
Connected Datalog Rule

Example

\[ \varphi \equiv O(x, y, z) \leftarrow E(x, y), E(y, z), E(z, x) \] is connected

\[ \text{graph}^+(\varphi) \]

\[ O(x, y, z) \leftarrow E(x, y), \neg E(y, z) \] is not connected
Semi-Connected Datalog

Definition

A Datalog$^-$ program is semi-connected if all rules are connected except (possibly) those of the last stratum.

Example

\[
\begin{align*}
TC(x, y) & \leftarrow E(x, y) \\
TC(x, y) & \leftarrow E(x, z), TC(z, y) \\
O(x, y) & \leftarrow \neg TC(x, y), x \neq y
\end{align*}
\]

Observation

SP-Datalog $\subseteq$ semi-connected Datalog$^-$
Semi-Connected Datalog

Theorem
Semi-connected Datalog$^\neg$ $\subseteq \mathcal{M}_{\text{disjoint}}$

[Ameloot, Ketsman, N, Zinn, 2014]

Theorem
Semi-connected wILOG$^\neg$ $= \mathcal{M}_{\text{disjoint}}$

[Ameloot, Ketsman, N, Zinn, 2014]

enumerate unions of connected components of input instance
Domain-guided transducers
Domain-guided policies

**Definition**

A domain assignment $\alpha$ for $\mathcal{N}$ is a total function from $\text{dom}$ to the power set of $\mathcal{N}$.

**Definition**

$\alpha(a) =$ all nodes responsible for $a$

A domain assignment $\alpha$ induces the domain-guided distribution policy $P_{\alpha}$ where

$$P_{\alpha}(R(a_1, \ldots, a_k)) := \bigcup_{i=1}^{k} \alpha(a_i).$$

every node in $\alpha(a)$ is responsible for all facts containing an occurrence of $a$

**Example**

$I = \{E(1, 3), E(3, 4), E(4, 6)\}$

$\alpha(a) = \begin{cases} 
\{1\} & \text{if } a \text{ is odd} \\
\{2\} & \text{otherwise}
\end{cases}$

$P_\alpha(E(1, 3)) = \{1\}$

$P_\alpha(E(3, 4)) = \{1, 2\}$

$P_\alpha(E(4, 6)) = \{2\}$
Domain-guided policies

Definition

A policy-aware transducer $\Pi$ computes a query $Q$ under domain-guidance if

- for all networks $N$,
- for all databases $I$,
- for all domain-guided policies $P$, and
- for every run $\mathcal{R}$ of $\Pi$,

$$\text{out}(\mathcal{R}) = Q(I).$$

Definition

$\mathcal{F}_2 = \text{queries distributedly computed under domain-guidance by policy-aware coordination-free transducers}$
Consistency And Logical Monotonicity

CALM-conjecture

“A query has a coordination-free and eventually consistent execution strategy under domain-guided distribution policies iff the query is domain-disjoint-monotone”

Theorem $\mathcal{F}_2 = \mathcal{M}_{\text{disjoint}}$

[Ameloot, Ketsman, N., Zinn, 2014]
\[ \mathcal{M}_{\text{disjoint}} \subseteq \mathcal{F}_2 \]

**Observation**

A subset \( C \subseteq \text{adom}(I) \) is complete at a node \( x \), when \( x \) knows for every fact \( f \) with \( \text{adom}(f) \cap C \neq \emptyset \) whether \( f \in I \) or \( f \not\in I \).

\[
I(C) = \{ f \in I \mid \text{adom}(f) \cap C \neq \emptyset \}
\]

\[ \rightarrow \text{union of components of } I \]

\[ \rightarrow x \text{ has full knowledge on } I(C) \]

By domain-disjoint-monotonicity: \( Q(I(C)) \subseteq Q(I) \)

When \( C = \text{adom}(I) \), \( Q(I(C)) = Q(I) \)

**Intuition**

Evaluate \( Q \) on increasingly larger complete subsets
\[ M_{\text{disjoint}} \subseteq F_2 \]

Program for node \( x \):

- **Broadcast active domain**
- **When a new domain element \( a \) is received, coordinate with another node to transfer all tuples related to \( a \)**
  
  - send out a request \((x, a)\) which asks to send to \( x \) all tuples in which \( a \) occurs
  
  - there always is (at least) one node \( y \) responsible for \( a \)
  
  - local synchronisation between \( x \) and \( y \) is set up to transfer tuples

- **Evaluate query for every complete subset**
Extended CALM-theorem

$$\text{Datalog}(\neq) \subsetneq \text{wILOG}(\neq) = M = \mathcal{F}_0 = A_0$$

$$\text{SP-Datalog} \subsetneq \text{SP-wILOG} = M_{\text{distinct}} = \mathcal{F}_1 = A_1$$

$$\text{semicon-Datalog}^- \subsetneq \text{semicon-wILOG}^- = M_{\text{disjoint}} = \mathcal{F}_2 = A_2$$
Discussion
Can we put the CALM-conjecture to rest?

Theorem \[ F_0 \subseteq F_1 \subseteq F_2 \subseteq F_3 = \mathcal{C} \]

[Zinn, Green, Ludäscher, 2012]

computable queries

coordination-free policy-aware transducers that have access to the global active domain
Coordination-freeness ≠ efficient evaluation

• Classes where no communication is required
  • Connected Datalog over connected components
• Pair coordination with communication efficiency
  • Take structure of query into account
• MPC by Beame, Koutris, Suciu
Connected Datalog

**Theorem** The query computing the won positions of the win-move game is in $F_2$.

[Zinn, Green, Ludäscher, 2012]

**Proof** simulation in semi-monotone fragment of Datalog$\forall\forall$ with a novel disorderly semantics

[Abiteboul, Vianu, 1991]

**Theorem** Queries expressed in connected Datalog under the well-founded semantics are in $F_2$

$\text{win}(X) \leftarrow \text{move}(X,Y), \neg\text{win}(Y)$

[Ameloot, Ketsman, N., Zinn, 2014]
Thank You

Bas  Jan  Tom  Daniel

Questions?

Keep Calm and Carry On

frankneven.com
Semi-Positive Datalog

**Theorem** \( \text{SP-Datalog} \cap \mathcal{H} = \text{Datalog} \)  
[Feder, Vardi, 2003]

**Theorem** \( \text{LFP} \cap \mathcal{H} \not\subseteq \text{Datalog} \)  
[Dawar, Kreutzer, 2008]

**Open** \( \text{SP-Datalog} \not= \mathcal{E} \cap \text{Stratified Datalog} \)

**Theorem** \( \text{FO}[\exists^* \forall \exists] \cap \mathcal{E} \subseteq \text{SP-Datalog} \)  
[Rosen, 1995]

Tait’s example separating \( \text{FO}[\exists] \) from \( \mathcal{E} \cap \text{FO} \) is definable in \( \text{SP-Datalog} \)