Executable First-order Queries in the Logic of Information Flows

Heba Aamer[1] Bart Bogaerts[2] Dimitri Surinx[1]
Eugenia Ternovska[3] Jan Van den Bussche[1]

[1]Universiteit Hasselt, Belgium

[2]Vrije Universiteit Brussel, Belgium

[3]Simon Fraser University, Canada

ICDT, 31st September 2020

Introduction

Consider the following information source on the web, where:

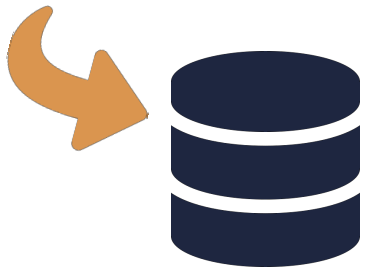
1. You give the ***username***

Introduction

Consider the following information source on the web, where:

1. You give the ***username***
2. You also give a ***password***

$$(username = x,\ password = y)$$

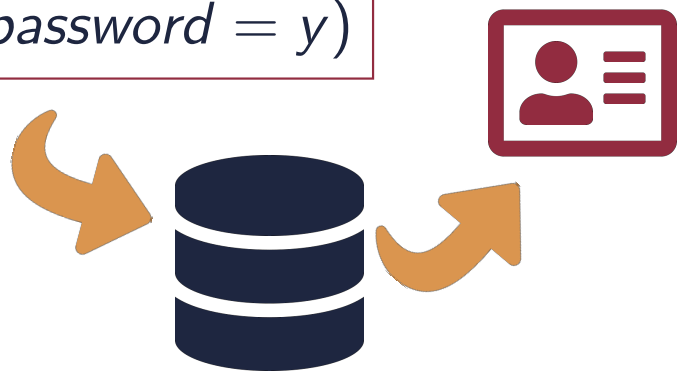


Introduction

Consider the following information source on the web, where:

1. You give the ***username***
2. You also give a ***password***
3. You get back the corresponding ***profile***

$$(username = x,\ password = y)$$

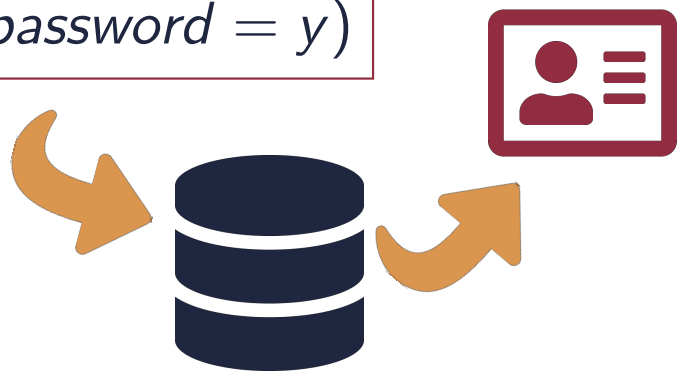


Introduction

Consider the following information source on the web, where:

1. You give the ***username***
2. You also give a ***password***
3. You get back the corresponding ***profile***

$$(username = x,\ password = y)$$



Note We do not expect the reverse access to be possible:

give the *profile* and get (*username*, *password*)

Information Sources with Limited Access Patterns

This type of information sources is said to have **limited access patterns**.

- provide values for some of the attributes
- get all the tuples that matches

Information Sources with Limited Access Patterns

This type of information sources is said to have **limited access patterns**.

- provide values for some of the attributes
- get all the tuples that matches

They are modules that:

- take inputs, and
- return (possibly multiple) outputs

Information Sources with Limited Access Patterns

This type of information sources is said to have **limited access patterns**.

- provide values for some of the attributes
- get all the tuples that matches

They are modules that:

- take inputs, and
- return (possibly multiple) outputs

Formally Every relation name has an arity, and also an **input arity**.

Information Sources with Limited Access Patterns

This type of information sources is said to have **limited access patterns**.

- provide values for some of the attributes
- get all the tuples that matches

They are modules that:

- take inputs, and
- return (possibly multiple) outputs

Formally Every relation name has an arity, and also an **input arity**.

Notation $R(\bar{x}; \bar{y})$ (e.g. $R(\mathit{username}, \mathit{password}; \mathit{profile})$).

Querying Databases with Limited Access Patterns

Queries are expressed with:

- low-level procedural relational algebra programs (Plans).
- high-level declarative logical language (e.g. Executable FO).

Querying Databases with Limited Access Patterns

Queries are expressed with:

- low-level procedural relational algebra programs (Plans).
- high-level declarative logical language (e.g. Executable FO).

Question

$$\text{Executable FO} \xleftarrow{\text{declarative}} \textbf{?} \xrightarrow{\text{procedural}} \text{Plans}$$

Querying Databases with Limited Access Patterns

Queries are expressed with:

- low-level procedural relational algebra programs (Plans).
- high-level declarative logical language (e.g. Executable FO).

Question

$$\text{Executable FO} \xleftarrow{\text{declarative}} \textbf{?} \xrightarrow{\text{procedural}} \text{Plans}$$

Contribution We introduce a new approach through FLIF.

Navigational (X-Path like) Graph Querying

An expression E is defined as

$$E ::= \tau \mid E \circ E \mid E^* \mid E \cup E \mid E \cap E \mid E - E$$

edge label composition transitive closure

Navigational (X-Path like) Graph Querying

An expression E is defined as

$$E ::= \tau \mid E \circ E \mid E^* \mid E \cup E \mid E \cap E \mid E - E$$

edge label — composition — transitive closure

Interpretation of E relative to a Database D ($[\![E]\!]_D$) is a binary relation defined by **relational semantics**

Navigational (X-Path like) Graph Querying

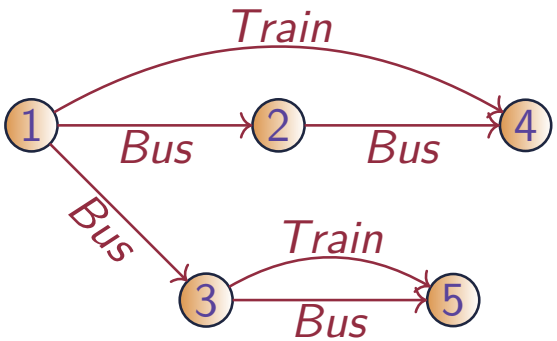
An expression E is defined as

$$E ::= \tau \mid E \circ E \mid E^* \mid E \cup E \mid E \cap E \mid E - E$$

edge label — composition — transitive closure

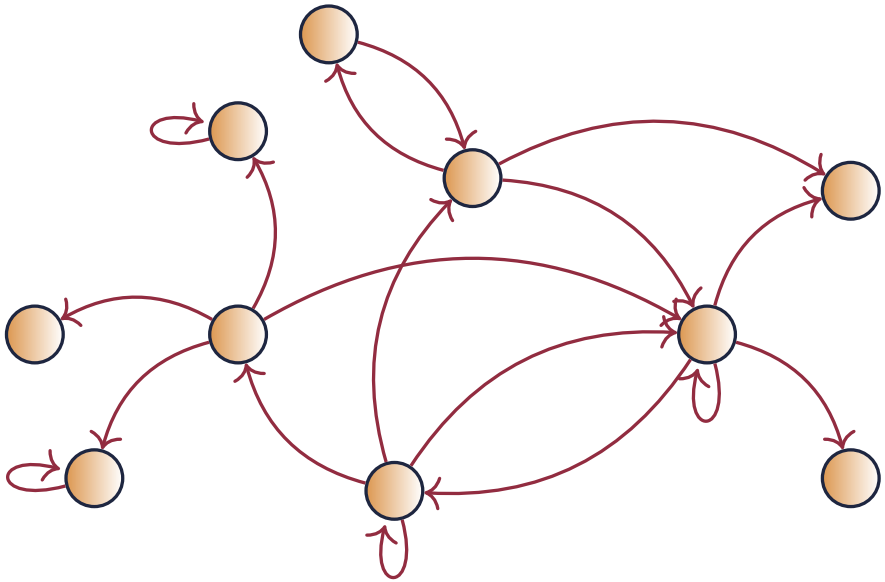
Interpretation of E relative to a Database D ($[\![E]\!]_D$) is a binary relation defined by **relational semantics**

$$E = (Bus \circ Bus - Train)$$



$[\![Bus]\!]$	$[\![Train]\!]$	$[\![Bus \circ Bus]\!]$	$[\![E]\!]$
$(1,2)$	$(1,4)$	$(1,4)$	$(1,5)$
$(1,3)$	$(3,5)$	$(1,5)$	
$(2,4)$			
$(3,5)$			

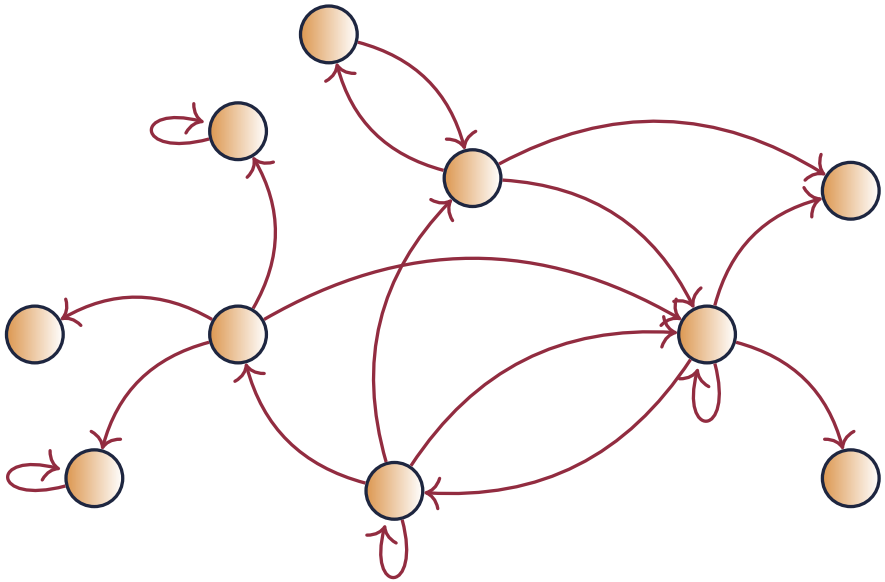
Graph?



Question 1: What are the nodes?

Question 2: What are the edges?

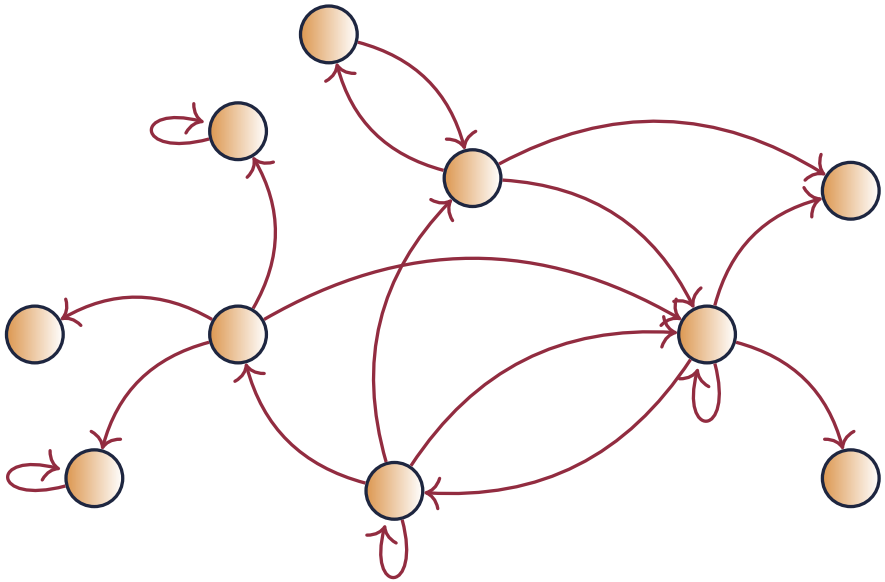
Graph?



Question 1: What are the nodes? Valuations

Question 2: What are the edges?

Graph?



Question 1: What are the nodes? Valuations

Question 2: What are the edges? Possible Accesses

FLIF Syntax

FLIF expressions E are:

$$E ::= \tau \mid E \circ E \mid E \cup E \mid E \cap E \mid E - E$$

FLIF Syntax

FLIF expressions E are:

$$E ::= \tau \mid E \circ E \mid E \cup E \mid E \cap E \mid E - E$$

Atomic expressions τ are (where x and y are variables, and c is a constant):

$$\tau ::= \underbrace{R(\bar{x}; \bar{y})}_{\text{relation atom}} \mid \underbrace{(x = y) \mid (x = c)}_{\text{selection}} \mid \underbrace{(x := y) \mid (x := c)}_{\text{assignment}}$$

FLIF Examples

Consider a schema with two relations B and T. Each of them is a binary relation with one input.

- The expression $B(x; y) \circ T(y; z)$ takes x as an input and retrieves the possible values for y and z such that:
  1. You can go from x to z through y.
  2. You can reach y from x by a bus.
  3. You can go from y to z by a train.

FLIF Examples

Consider a schema with two relations B and T. Each of them is a binary relation with one input.

- The expression $B(x; y) \circ T(y; z)$ takes x as an input and retrieves the possible values for y and z such that:
  1. You can go from x to z through y.
  2. You can reach y from x by a bus.
  3. You can go from y to z by a train.

- The expression $(B(x; y) \circ B(y; z) \circ T(z; x)) \cap (x = x)$ takes x, y, and z as inputs and checks that:
  1. You can go from x to z by two buses through y.
  2. You can go back from z to x by a train.

Is FLIF Good?

Question: What is the expressive power of this approach? And how does it compare to Executable FO and Plans?

Executable FO

Executable FO is proposed by Nash and Ludäscher.

- **Syntax:** is a syntactic fragment of FO.
- **Semantics:** standard natural semantics.
- **Inputs:** set of variables $\mathcal{V}$.
- **Idea:** determines whether a formula is *executable* or not, relative to $\mathcal{V}$ (executable means it respects the access patterns).

Executable FO

Executable FO is proposed by Nash and Ludäscher.

- Syntax: is a syntactic fragment of FO.
- Semantics: standard natural semantics.
- Inputs: set of variables $\mathcal{V}$.
- Idea: determines whether a formula is *executable* or not, relative to $\mathcal{V}$ (executable means it respects the access patterns).

Example ($\mathcal{V}$-executable formulas)

- The formula $\exists y\, R(x; y)$ is $\{x\}$-executable.

Executable FO

Executable FO is proposed by Nash and Ludäscher.

- **Syntax:** is a syntactic fragment of FO.
- **Semantics:** standard natural semantics.
- **Inputs:** set of variables $\mathcal{V}$.
- **Idea:** determines whether a formula is *executable* or not, relative to $\mathcal{V}$ (executable means it respects the access patterns).

Example ($\mathcal{V}$-executable formulas)

- The formula $\exists y\, R(x; y)$ is $\{x\}$-executable.
- The formula $x = \mathit{john} \wedge R(x, y; z)$ is $\{y\}$-executable.

Executable FO

Executable FO is proposed by Nash and Ludäscher.

- **Syntax:** is a syntactic fragment of FO.
- **Semantics:** standard natural semantics.
- **Inputs:** set of variables $\mathcal{V}$.
- **Idea:** determines whether a formula is *executable* or not, relative to $\mathcal{V}$ (executable means it respects the access patterns).

Example ($\mathcal{V}$-executable formulas)

- The formula $\exists y\, R(x; y)$ is $\{x\}$-executable.
- The formula $x = \mathit{john} \wedge R(x, y; z)$ is $\{y\}$-executable.
- The formula $R(x, y; z) \wedge x = \mathit{john}$ is not $\{y\}$-executable.

Inputs and Outputs of an FLIF Expression

The variables of an expression E are classified into:

- **Inputs** $I(E)$: are needed from the beginning
- **Outputs** $O(E)$: can change during the evaluation

Inputs and Outputs of an FLIF Expression

The variables of an expression E are classified into:

- **Inputs** $I(E)$: are needed from the beginning
- **Outputs** $O(E)$: can change during the evaluation

E	$I(E)$	$O(E)$
$R(\bar{x};\bar{y})$	$\bar{x}$	$\bar{y}$

Inputs and Outputs of an FLIF Expression

The variables of an expression E are classified into:

- **Inputs** $I(E)$: are needed from the beginning
- **Outputs** $O(E)$: can change during the evaluation

E	$I(E)$	$O(E)$
$R(\bar{x};\bar{y})$	$\bar{x}$	$\bar{y}$
$E_1 \circ E_2$	$I(E_1) \cup (I(E_2) - O(E_1))$	$O(E_1) \cup O(E_2)$

Inputs and Outputs of an FLIF Expression

The variables of an expression E are classified into:

- **Inputs** $I(E)$: are needed from the beginning
- **Outputs** $O(E)$: can change during the evaluation

E	$I(E)$	$O(E)$
$R(\bar{x};\bar{y})$	$\bar{x}$	$\bar{y}$
$E_1 \circ E_2$	$I(E_1) \cup (I(E_2) - O(E_1))$	$O(E_1) \cup O(E_2)$
$E_1 \cup E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1) \cup O(E_2)$

Inputs and Outputs of an FLIF Expression

The variables of an expression E are classified into:

- **Inputs** $I(E)$: are needed from the beginning
- **Outputs** $O(E)$: can change during the evaluation

E	$I(E)$	$O(E)$
$R(\bar{x};\bar{y})$	$\bar{x}$	$\bar{y}$
$E_1 \circ E_2$	$I(E_1) \cup (I(E_2) - O(E_1))$	$O(E_1) \cup O(E_2)$
$E_1 \cup E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1) \cup O(E_2)$
$E_1 \cap E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1) \cap O(E_2)$
$E_1 - E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1)$

Inputs and Outputs of an FLIF Expression

The variables of an expression E are classified into:

- **Inputs** $I(E)$: are needed from the beginning
- **Outputs** $O(E)$: can change during the evaluation

E	$I(E)$	$O(E)$
$R(\bar{x};\bar{y})$	$\bar{x}$	$\bar{y}$
$E_1 \circ E_2$	$I(E_1) \cup (I(E_2) - O(E_1))$	$O(E_1) \cup O(E_2)$
$E_1 \cup E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1) \cup O(E_2)$
$E_1 \cap E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1) \cap O(E_2)$
$E_1 - E_2$	$I(E_1) \cup I(E_2) \cup (O(E_1) \triangle O(E_2))$	$O(E_1)$
$(x = y)$	$\{x, y\}$	$\emptyset$
$(x := y)$	$\{y\}$	$\{x\}$
$(x = c)$	$\{x\}$	$\emptyset$
$(x := c)$	$\emptyset$	$\{x\}$

IO-disjoint Expressions

What happens if a variable is an input and an output?

- Relation a binary relation $Inc = \{(0,1),(1,2),(2,3),\ldots\}$.
- Expression $Inc(x;x)$ increments the value of x
- Discrepancy $Inc(x;x)$ is not satisfied in Executable FO

IO-disjoint Expressions

What happens if a variable is an input and an output?

- Relation a binary relation $Inc = \{(0,1),(1,2),(2,3),\ldots\}$.
- Expression $Inc(x;x)$ increments the value of x
- Discrepancy $Inc(x;x)$ is not satisfied in Executable FO

Solution work with ($\mathrm{FLIF}^{\mathrm{io}}$): the input-output disjoint fragment of FLIF.

Executable FO to $\mathrm{FLIF}^{\mathrm{io}}$ Theorem

Let φ be a $\mathcal{V}$-executable formula over schema $\mathcal{S}$. There exists an $\mathrm{FLIF}^{\mathrm{io}}$ expression E over $\mathcal{S}$ with the following properties:

1. $I(E) = \mathcal{V}$.
2. $O(E) \supseteq \mathsf{FreeVar}(\varphi) - \mathcal{V}$.
3. For every D, ν_{in} on $\mathcal{V}$ and $\nu_{\mathrm{in}} \subseteq \nu$:
   - If $(\nu_{\mathrm{in}}, \nu) \in [\![E]\!]_D$, then $D, \nu \models \varphi$.
   - If $D, \nu \models \varphi$, then there exists ν' such that $(\nu_{\mathrm{in}}, \nu') \in [\![E]\!]_D$ and $\nu = \nu'$ on $\mathsf{FreeVar}(\varphi) \cup \mathcal{V}$.

Executable FO to $\mathrm{FLIF}^{\mathrm{io}}$ Theorem

Let φ be a $\mathcal{V}$-executable formula over schema $\mathcal{S}$. There exists an $\mathrm{FLIF}^{\mathrm{io}}$ expression E over $\mathcal{S}$ with the following properties:

1. $I(E) = \mathcal{V}$.
2. $O(E) \supseteq \mathsf{FreeVar}(\varphi) - \mathcal{V}$.
3. For every D, ν_{in} on $\mathcal{V}$ and $\nu_{\mathrm{in}} \subseteq \nu$:
   - If $(\nu_{\mathrm{in}}, \nu) \in [\![E]\!]_D$, then $D, \nu \models \varphi$.
   - If $D, \nu \models \varphi$, then there exists ν' such that $(\nu_{\mathrm{in}}, \nu') \in [\![E]\!]_D$ and $\nu = \nu'$ on $\mathsf{FreeVar}(\varphi) \cup \mathcal{V}$.

Example

Consider an $\{x\}$-executable FO formula φ is $R(x; x, y) \wedge M(y; z)$.

- Conjunction is composition.
- $R(x; x, y)$ is equivalent to $R(x; u, y) \wedge u = x$.

We can take E to be the expression $(R(x; u, y) \circ (u = x)) \circ M(y; z)$.

$\mathrm{FLIF}^{\mathrm{io}}$ to Executable FO Theorem

Let E be an $\mathrm{FLIF}^{\mathrm{io}}$ expression over schema $\mathcal{S}$. There exists an $I(E)$-executable FO formula φ over $\mathcal{S}$, such that for every D, ν_{in} on $I(E)$ and $\nu_{\mathrm{in}} \subseteq \nu$:

$$(\nu_{\mathrm{in}}, \nu) \in [\![E]\!]_D \text{ iff } D, \nu \models \varphi$$

$\mathrm{FLIF}^{\mathrm{io}}$ to Executable FO Theorem

Let E be an $\mathrm{FLIF}^{\mathrm{io}}$ expression over schema $\mathcal{S}$. There exists an $I(E)$-executable FO formula φ over $\mathcal{S}$, such that for every D, ν_{in} on $I(E)$ and $\nu_{\mathrm{in}} \subseteq \nu$:

$$(\nu_{\mathrm{in}}, \nu) \in [\![E]\!]_D \text{ iff } D, \nu \models \varphi$$

Example

Consider the expression $R(x; y, u) \circ M(x; z, u)$.

1. R gives (y, u)-bindings for the given x.
2. Then, M gives (z, u)-bindings for the same x.

A correct translation into Executable FO is $(\exists u\, R(x; y, u)) \wedge M(x; z, u)$.

Conclusion

- Viewing a DB with limited access patterns as a graph leads to interesting new query languages.

Conclusion

- Viewing a DB with limited access patterns as a graph leads to interesting new query languages.
- Every $\mathrm{FLIF}^{\mathrm{io}}$ expression has a simple plan:
  - similar structure
  - no renaming
  - only natural joins