Consistent Query Answering

in

Relational and XML Databases

(Issues and Progress Report)

Leopoldo Bertossi

Carleton University

Ottawa, Canada

Currently: Centro de Investigación de la Web (CIW), Universidad de Chile

Joint work with Mauricio Vines and Natalia Villanueva

Introduction

Databases may become inconsistent wrt a given set of integrity constraints (ICs)

- DBMS has no mechanism to maintain certain classes of ICs
- Data of different sources are being integrated

  Even if the independent data sources are consistent, integrated data may be inconsistent
- New constraints are imposed on pre-existing, legacy data
- Soft, user, or informational constraints, to be considered at query answering, but without being enforced

Most likely most of the data in the DB is still "consistent"

We want to obtain query answers that are semantically correct

Which are those?

Considerable amount of research on consistent query answering (CQA) has been carried out in the last 6 years

In [Arenas, Bertossi, Chomicki. PODS 99]: For relational databases

- Characterization of consistent answers to queries as those that are invariant under minimal repairs of the original database; i.e. true in all minimally repaired versions of the DB
- Mechanism for computing them (for certain classes of queries and ICs)

Database repairs are consistent instances that minimize under set inclusion the set of insertions/deletions of whole database tuples

D an instance as a set of ground atoms; possibly inconsistent wrt IC:

A repair D' of D is a new instance that satisfies IC and makes

$$\Delta(D, D') = (D \smallsetminus D') \cup (D' \smallsetminus D)$$

minimal under set inclusion

Example 1: Inconsistent DB instance D wrt
FD: $Name \rightarrow Salary$, actually a Key Dependency (KD)

$Employee$	$Name$	$Salary$
	$Page$	5000
	$Page$	8000
	$Smith$	3000
	$Stowe$	7000

Repairs D_1, resp. D_2

$Employee$	$Name$	$Salary$
	$Page$	5000
	$Smith$	3000
	$Stowe$	7000

$Employee$	$Name$	$Salary$
	$Page$	8000
	$Smith$	3000
	$Stowe$	7000

Consistent answers to the queries:

- $Employee(x, y)?$: $(Smith, 3000), (Stowe, 7000)$
- $\exists y Employee(x, y)?$: $Page, Smith, Stowe$

First algorithm for CQA was based on *first-order query rewriting*

Example 2: (continued)

FD: $\forall XYZ\ (\neg Employee(X,Y)\ \vee\ \neg Employee(X,Z)\ \vee Y = Z)$

Query: $Employee(x,y)$?

Consistent answers can be obtained by means of the transformed query

$$T(Employee(x,y)) := Employee(x,y)\ \wedge$$
$$\forall z\ (\neg Employee(x,z)\ \vee\ y = z)$$

... those tuples (x,y) in the relation for which x does not have and associated z different from y ...

A *residue* has been appended to the original query; it can be obtained by resolution between the FD and the query

With FO query rewriting there is no need to compute repairs; only the original database is queried

It ensures polynomial time data complexity for CQA

However, FO query rewriting is defined (or works) for limited classes of queries and ICs

The data complexity of CQA in relational databases can have data complexity as high as Π_2^P-complete

[Arenas, Bertossi, Chomicki. ICDT 01]

[Chomicki, Marcinkowski. I&C 05]

[Cali, Lembo, Rosati. PODS 2003]

[Fuxman, Miller. ICDT 2005]

At least seven systems for CQA have been implemented

- QUECA [Celle, Bertossi]
- Hyppo [Staworko, Chomicki]
- CONQUER [Fuxman, Miller]
- INFOMIX [Leone, ...]
- Calabria [Flesca, ...]
- System for repairing numerical databases [Lopatenko, Bravo; ICDE 2007]
- ConsEX [Caniupan, Bertossi; 2006]

Inconsistencies in XML

CQA has become a subject of active research in databases

Most work has concentrated on relational databases

With some additional work in multidimensional databases and DWHs

Not much work has been done wrt to inconsistencies in XML data

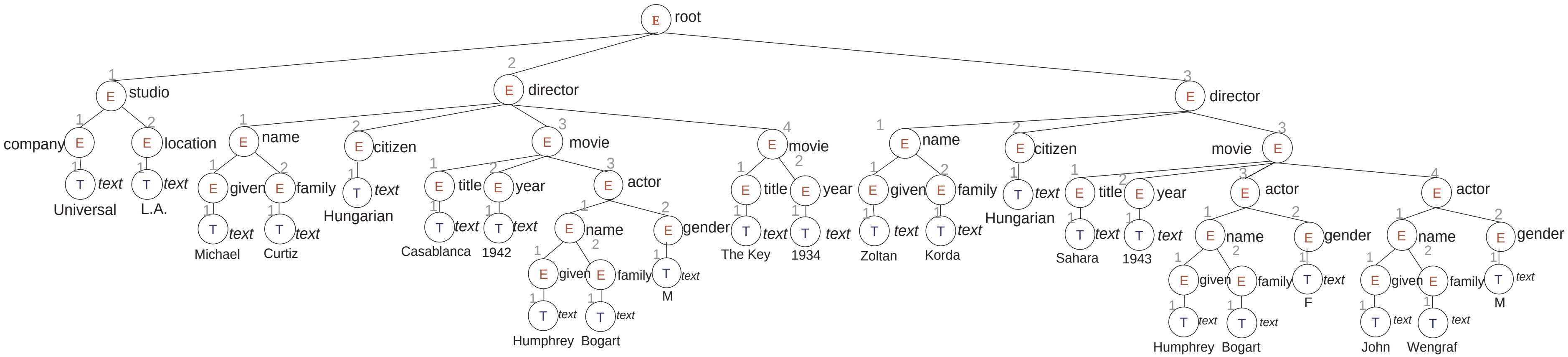
Inconsistencies wrt DTDs and ways to restore consistency has been studied in [Staworko, Chomicki. 2006]

Here we present work in progress on CQA from XML databases that may violate functional dependencies (FDs)

We need a framework to formalize and address this problem

In particular, a logical language to express FDs and queries

XML Trees



An XML tree is a tuple $T = \langle V, el, val, att, lab, root\rangle$ where:

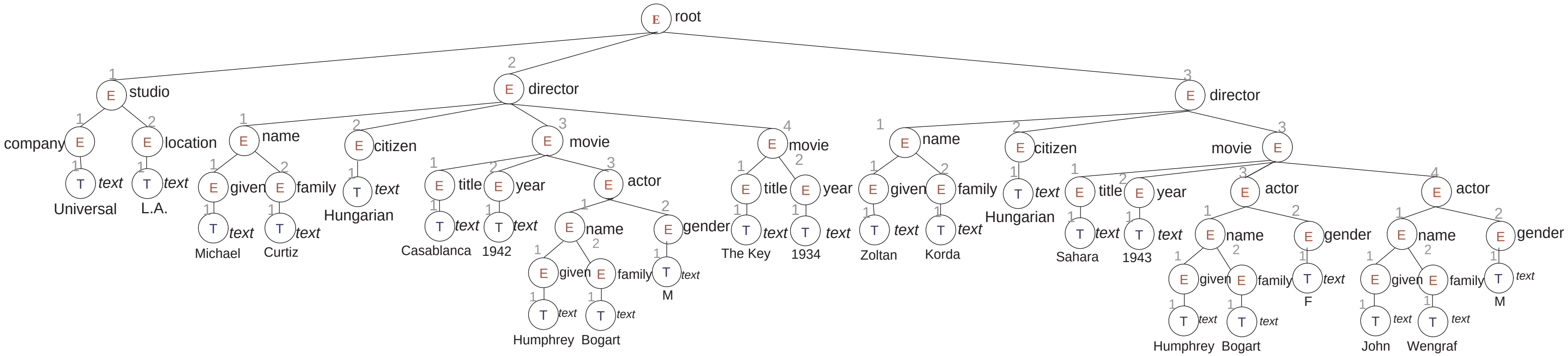
- $V \subseteq \mathbf{V}$ is a set of nodes that forms a tree with root $root$
- $lab\colon V \to \mathbf{E} \cup \mathbf{A} \cup \mathbf{S}$, gives labels to nodes in V ($E = lab^{-1}(\mathbf{E})$, etc.)
- $el\colon E \to seq(E \cup S)$, the finite sequences of elements of $E \cup S$
- $att\colon E \to P(A)$, the set of subsets of A
- $val\colon A \cup S \to \mathbf{S}$

DTDs

They specify the basic structure of the XML tree (or XML document)

```
<!ELEMENT root (studio+, director+)>

<!ELEMENT studio (company,location?)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT location (#PCDATA)>

<!ELEMENT director (name,citizenship?,movie+)>

<!ELEMENT name (given,family)>

<!ELEMENT given (#PCDATA)>

<!ELEMENT family (#PCDATA)>

<!ELEMENT citizenship (#PCDATA)>

<!ELEMENT movie (title, year, actor*)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT year (#PCDATA)>

<!ELEMENT actor (name, gender)>

<!ELEMENT gender (#PCDATA)>
```

Node Address



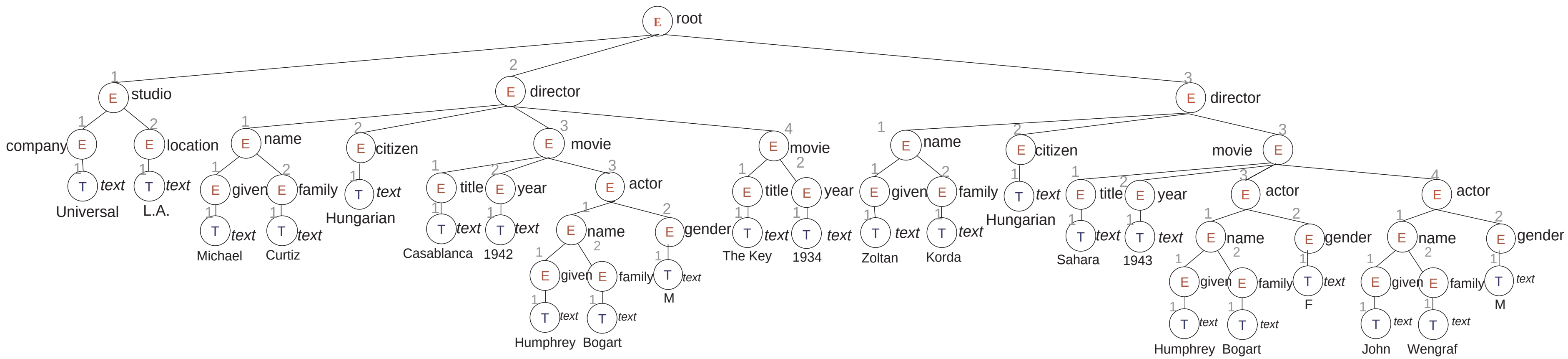
A *node address* is a persistent identifier for a node

In the example, $address(root, T_1) = [0]$ and for v the leftmost *director* node, $address(v, T_1) = [0, 2]$

Formally, given a tree $T = (V, lab, el, att, val, root)$, the node address is defined by:

- $address(root, T) = [0]$
- For $v \in E$ with $el(v) = [v_1, \ldots, v_n]$: $address(v_i, T) = address(v, T) \cdot [i]$

Substitution



We introduce a set $VAR = \{x_1, x_2, \ldots\}$ of *variables for node addresses*

A substitution on T is a function $\sigma\colon VAR \to \{address(v, T) : v \in V\}$

For example, σ may assign x_1 in T_1 to $[0, 1, 2]$, i.e. $\sigma(x_1) = [0, 1, 2]$

Address Equality and Value Equality

Given an XML tree T and a substitution σ on T:

- $T \models_\sigma (x_1 =_A x_2)$ iff $\sigma(x_1) = \sigma(x_2)$
- $T \models_\sigma (x_1 =_V x_2)$ iff there are nodes v_1 and v_2 in T such that $\sigma(x_1) = address(v_1, T)$ and $\sigma(x_2) = address(v_2, T)$ and v_1, v_2 define isomorphic XML subtrees

In the example, for $\sigma(x_1) = [0, 2, 3, 3, 1]$ and $\sigma(x_2) = [0, 3, 3, 3, 1]$:

- $T_1 \not\models_\sigma (x_1 =_A x_2)$
- $T_1 \models_\sigma (x_1 =_V x_2)$

  (two nodes defining subtrees with the same information about *Humphrey Bogart*)

Paths

A base path is defined by: $p \ ::= \ \epsilon \ | \ l \ | \ p{\cdot}p$

$l \in \mathbf{E} \setminus \{root\}$ and "$\cdot$" is concatenation

For example: $director, \quad director \cdot name$

For an XML tree T, p a base path, and $x \in VAR$, a *simple path expression* is defined by:

$$PE \ ::= \ root{\cdot}p{\cdot}B \ | \ x{\cdot}p{\cdot}B$$

$$B \ ::= \ \epsilon \ | \ text$$

For example: $x \cdot director, \quad root \cdot director \cdot name$

If a path expression contains $root$, it is absolute , otherwise is relative

Conforming Nodes

$\text{conf}_{\sigma,T}(x{\cdot}p)$ gives the set of (addresses of) nodes that can be reached via the path $x{\cdot}p$ for a given σ

Let σ be a substitution on T, $x \in \mathit{VAR} \cup \{\mathit{root}\}$, and $\sigma(\mathit{root}) := [0]$

1. For a base path p, $\text{conf}_{\sigma,T}(x{\cdot}p)$ is defined by induction on p

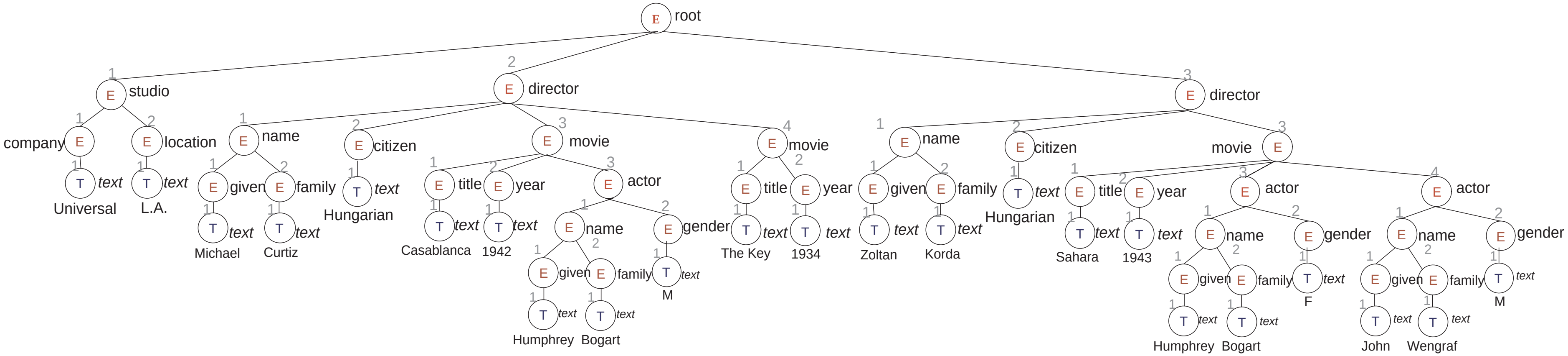
   a) $\text{conf}_{\sigma,T}(x{\cdot}\epsilon) = \{\sigma(x)\}$

   b) $\text{conf}_{\sigma,T}(x{\cdot}l) = \{\sigma(x) \cdot [i] \mid el(v) = [\ldots, v_i, \ldots],$ and $\mathit{address}(v, T) = \sigma(x)$ and $\mathit{lab}(v_i) = l\}$

   c) $\text{conf}_{\sigma,T}(x{\cdot}p{\cdot}q) = \{L_2 \mid$ exists $L_1 \in \text{conf}_{\sigma,T}(x, p)$ and $L_2 \in \text{conf}_{\sigma,T}(v{\cdot}q)$ and $\sigma(v) = L_1\}$

2. $\text{conf}_{\sigma,T}(x{\cdot}p{\cdot}\mathit{text}) = \{L \cdot [i] \mid L \in \text{conf}_{\sigma,T}(x{\cdot}p)$ and $\mathit{lab}(v_i) = \mathit{text},$ with $\sigma(v) = L$ and $el(v) = [\ldots, v_i, \ldots]\}$

Example:



For $\sigma(x) = [0, 2]$:

- $\mathsf{conf}_{\sigma,T_1}(x \cdot movie) = \{[0, 2, 3], [0, 2, 4]\}$
- $\mathsf{conf}_{T_1}(root \cdot director \cdot movie) = \{[0, 2, 3], [0, 2, 4], [0, 3, 3]\}$

  In absence of address variables, σ is omitted

The $\mathcal{L}_{XML}$ Language

We need a language to express queries and ICs on XML trees

1. Atomic formulas
   - $true$ and $false$
   - For $x_1, x_2 \in VAR$, $x_1 =_A x_2$ and $x_1 =_V x_2$ are formulas
   - If p is a simple path expression and $x \in VAR$, $p(x)$ is a formula (a path atom)
2. Boolean combinations of formulas are formulas
3. If φ is a formula and $x \in VAR$, then $\forall x(\varphi)$ and $\exists x(\varphi)$ are formulas

For example: $x_1 \cdot title(x) \wedge x_2 \cdot title(y) \wedge x =_V y$

$$\forall x \forall y (root \cdot director(x) \wedge x \cdot movie(y))$$

Truth?

A few relevant cases:

- $T \models_\sigma p(x)$ iff $\sigma(x) = address(v, T)$ and $v \in \mathsf{conf}_{\sigma,T}(p)$

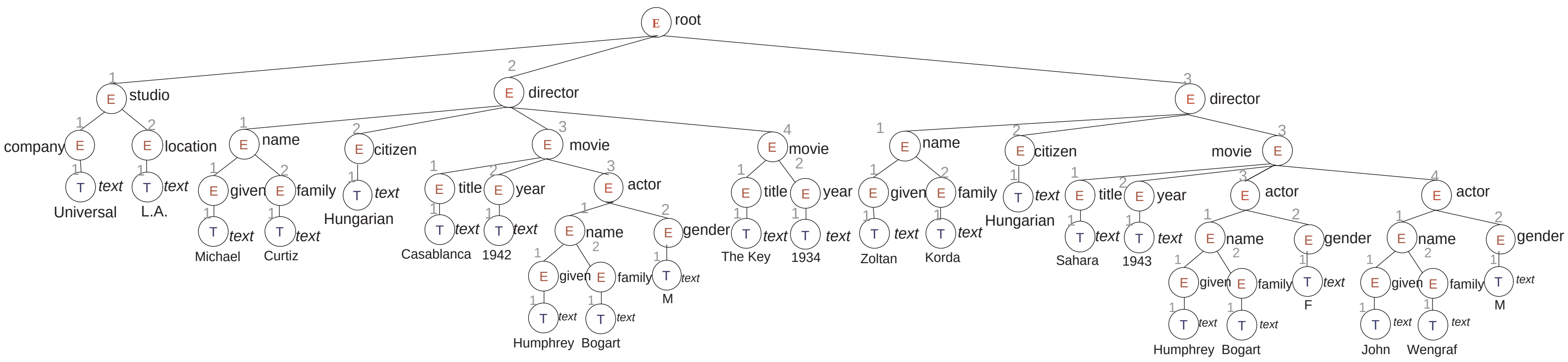
  It holds if x is assigned the address of a node v that is reached following path p

- $T \models_\sigma x \cdot p(y)$ iff for nodes v_1, v_2 it holds that $\sigma(x) = address(v_1, T)$
  and $\sigma(y) = add(v_2, T)$, and $v_2 \in \mathsf{conf}_{\sigma,T}(v_1 \cdot p)$

  It holds if x, y are assigned the addresses of nodes v_1, v_2, resp., and v_2 is reached from v_1 following path p

  In other words, if "node" y can be reached from "node" x following path p from x

Truth of equality atoms were defined, others cases are straightforward ...



For example:

- For $\sigma(x) = [0,2]$: $T_1 \models_\sigma root.director(x)$
- For $\sigma_2(x) = [0,2]$ and $\sigma_2(y) = [0,2,1]$:

  $T_1 \models_{\sigma_2} \; root{\cdot}director(x) \wedge \; x{\cdot}name(y)$
- For every σ: $T_1 \models_\sigma \; (\forall x \; \neg \; root{\cdot}director{\cdot}movie{\cdot}producer(x))$

Minimal Structural Constraints

We may have XML trees that are inconsistent wrt certain functional dependencies (FDs)

However, those tree will satisfy certain *minimal structural constraints* related to those FDs

For example, an actor must have a name, but should not have more than one name

This will allow us to compare different actors, and require, for example that if they appear with the same name in different subtrees, then they should be the same actor

The latter is the FD, but we are not imposing it by construction

Minimal structural constraints (MSCs) specify that certain elements must exists and cannot be repeated in the scope of a subtree (e.g. a movie having two titles)

The MCSs can be specified using DTDs as usual, and we can also express them using $\mathcal{L}_{XML}$ formulas

An MSC $\mathcal{M}$ is specified as a finite collection of statements of the form:

$$p[\![p_1, \ldots, p_n]\!]$$

An XML tree T satisfies this statement in $\mathcal{M}$ when:

- The paths $p \cdot p_i$ exist in T
- In T, for each node v that is reached via p, there is exactly one node reached from v via p_i

Consider the following DTD specification of elements *root* and *studio*:

```
<!ELEMENT root (studio+, director?)>

<!ELEMENT studio (ceo,location?)>
```

It can be seen also as an MSC: $root \cdot studio[\![ceo]\!]$

We can even express the requirements in $\mathcal{L}_{XML}$:

$$\exists xy(root(x) \wedge x \cdot studio(y))$$
$$\exists xy(studio(x) \wedge x \cdot ceo(y))$$
$$\forall xy_1y_2(studio(x) \wedge x \cdot ceo(y_1) \wedge x \cdot ceo(y_2) \rightarrow y_1 =_A y_2)$$

We will assume that DTDs are of the MSC form plus the possible existence of optional elements (like *director, location* here)

And not any MSCs, but those associated to functional dependencies

It is possible to extend the notation for MSCs indicating optional elements

In this sense they are "minimal and FD-oriented DTDs"

For example, in T_1 a director must not have two names

The name is a unique element for the subtree defined by director

Therefore, there will be no director called both "Michael Curtiz" and "Zoltan Korda"

On the other hand, a functional dependency may specify that there are not two different directors with the same name, which is something different

That is, an XML tree may satisfy the MSC without satisfying this last requirement and inconsistencies may arise

Integrity constraints (and FDs) for XML have been studied in the database community: Peter Buneman, Leonid Libkin, Wenfei Fan, Jerome Simeon, Marcelo Arenas, ...

Absolute Key Constraints

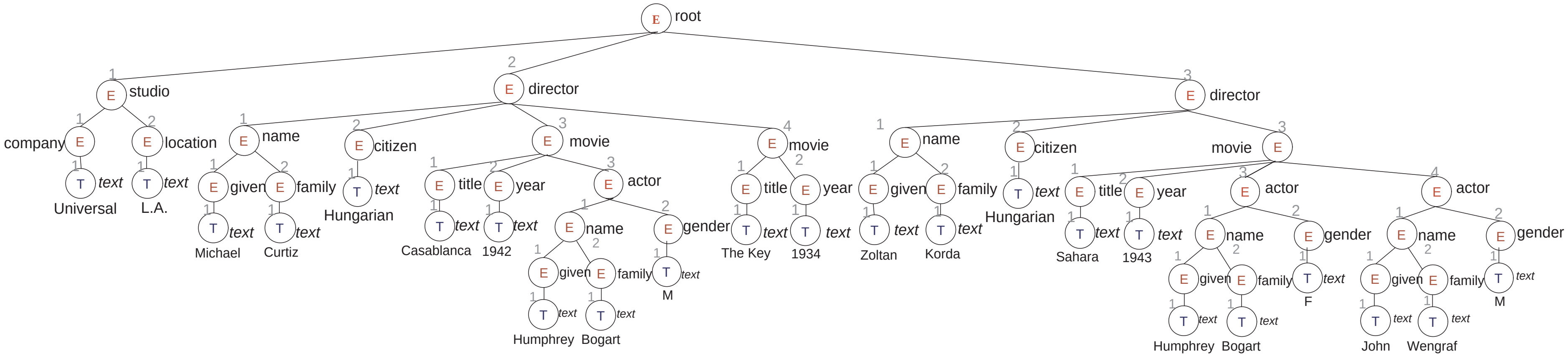
For an XML tree T that conforms to the MSC statement $B[\![C_1, \ldots, C_n]\!]$, an *absolute key constraint* (AKC) is an $\mathcal{L}_{XML}$-sentence:

$$\forall x_1 x_2 \; (B(x_1) \wedge B(x_2) \wedge \bigwedge_{i=1}^{n} x_1 \cdot C_i =_V x_2 \cdot C_i \;\; \longrightarrow \;\; x_1 =_A x_2)$$

B, that starts from the root, is the objective path and $C_1, \ldots, C_n$ are the key paths

The expression $(B, \{C_1, \ldots, C_n\})$ is used as shorthand for the AKC

For example, $(root \cdot director, \{name\})$ denotes an AKC



$(root{\cdot}director, \{name\})$ requires that two directors with the same name must be the same director; in this case described in the same subtree with root in *director*

This applies to the whole tree (starting from the root), that is why it is “absolute”

Here: $T_1 \models (root.director, \{name\})$

Relative Key Constraints

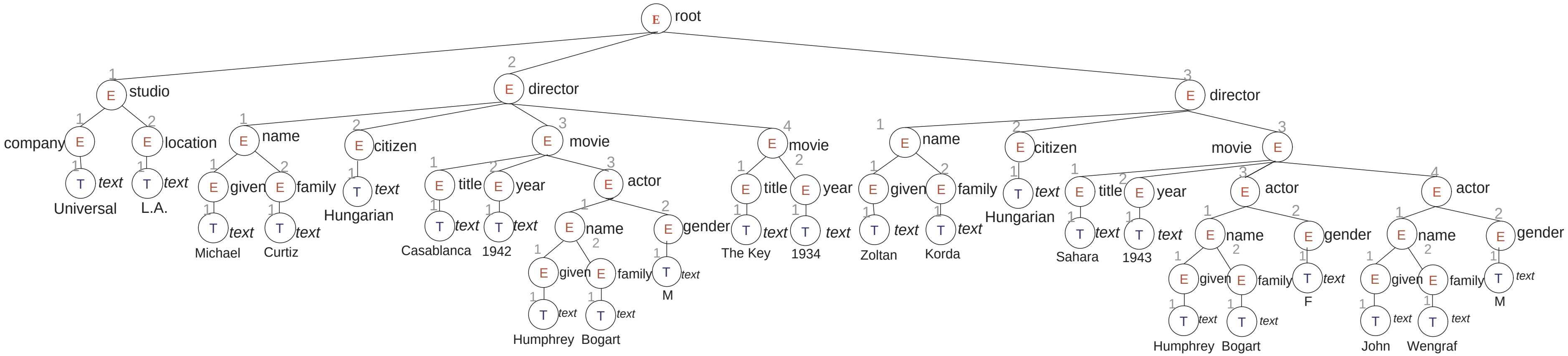
For an XML tree T that conforms to the MSC statement $B.B'[\![C_1, \ldots, C_n]\!]$, a *relative key constraint* (RKC) is an $\mathcal{L}_{XML}$ sentence:

$$\forall x x_1 x_2 \, (B(x) \wedge x{\cdot}B'(x_1) \wedge x{\cdot}B'(x_2) \wedge \bigwedge_{i=1}^{n} x_1{\cdot}C_i =_V x_2{\cdot}C_i \;\longrightarrow\; x_1 =_A x_2)$$

$B \cdot B'$, with B starting from *root*, is the *objective path* and $C_1, ..., C_n$ are the *key paths*

$(B, (B', \{C_1, \ldots, C_n\}))$ is used as a shorthand

Here, we have a key constraint relative to nodes that are reachable via path B



$T_1 \models (root.director, (movie, \{title\}))$

Intuitively, $(B, (B', \{C_1, \ldots, C_n\}))$ is an RKC when $(B', \{C_1, \ldots, C_n\})$ is an AKC for each subtree with root in a node that is reachable from the root following B

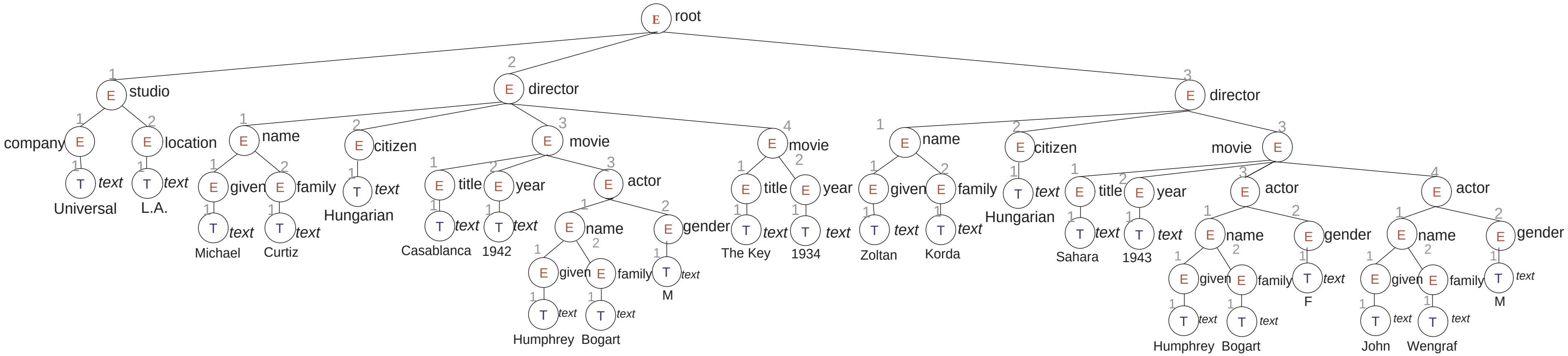
Absolute Functional Dependencies

For an XML tree T that conforms to the MSC statement $B[\![C_1, \ldots, C_n]\!]$, an *absolute functional dependency* (AFD) is an $\mathcal{L}_{XML}$-sentence:

$$\forall x_1 x_2 \; (B(x_1) \wedge B(x_2) \wedge \bigwedge_{i=1}^{n-1} x_1{\cdot}C_i =_V x_2{\cdot}C_i \rightarrow x_1{\cdot}C_n(y_1) =_V x_2{\cdot}C_n(y_2))$$

B, starting from the root, is the *objective path*, $C_1, \ldots, C_{n-1}$ are the *independent paths*, and C_n, $n \geq 1$, is the *dependent path*

$(B, \{C_1, \ldots, C_{n-1} \rightarrow C_n\})$ is used as a shorthand



$T_1 \models (root.director, \{name \to citizen\})$

Intuitively, for every pair of nodes v_1, v_2 reachable via B, it holds that if the nodes reached following C_i, $1 \leq i \leq n-1$, from v_1 and v_2, are equal in value, then the nodes reached following C_n from v_1 and v_2 must also be equal in value

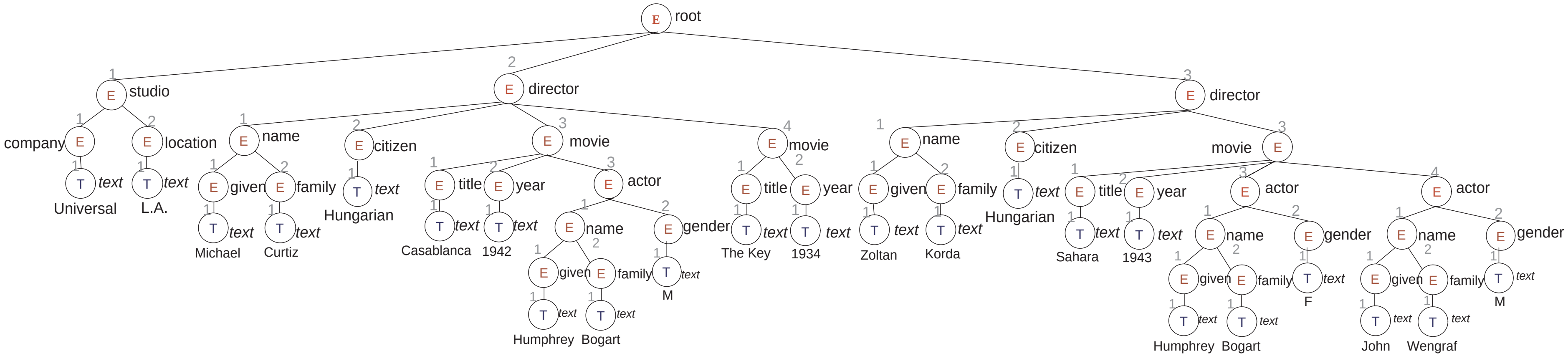
Relative Functional Dependency

For an XML tree T that conforms to the MSC statement $B \cdot B' [\![C_1, \ldots, C_n]\!]$, a *relative functional dependency* (RFD) is an $\mathcal{L}_{XML}$-sentence:

$$\forall x x_1 x_2 (B(x) \wedge \; x \cdot B'(x_1) \wedge \; x \cdot B'(x_2) \wedge \bigwedge_{i=1}^{n-1} x_1 \cdot C_i =_V x_2 \cdot C_i \; \longrightarrow x_1 \cdot C_n =_V x_2 \cdot C_n)$$

$B \cdot B'$, B starting from root, is the *objective path*, $C_1, ..., C_{n-1}$ are the *independent paths* and C_n, $n \geq 1$, is the *dependent path*

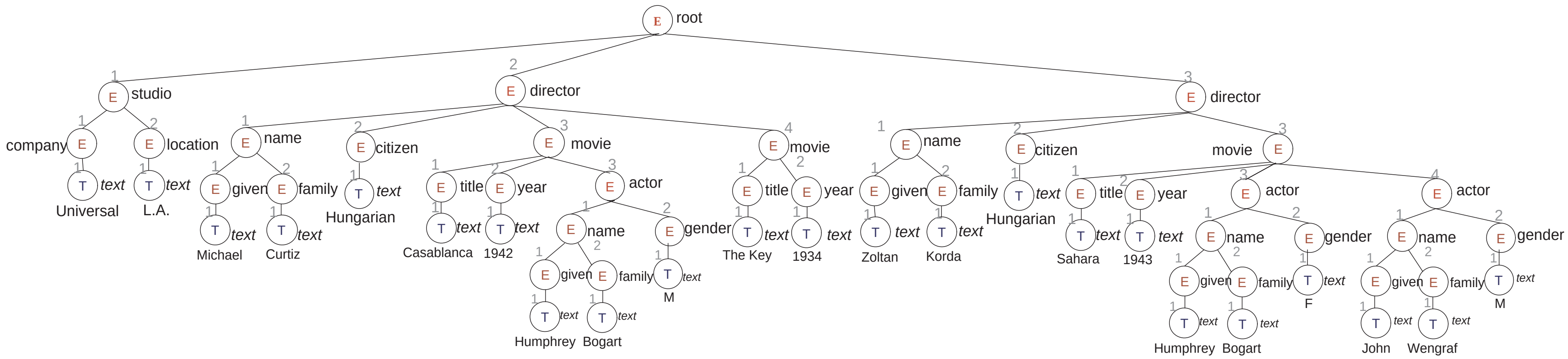
$(B, (B', \{C_1, \ldots, C_{n-1} \rightarrow C_n\}))$ is a shorthand for the RFD



$$T_1 \models (root.director, (movie, \{title \rightarrow year\}))$$

Intuitively, $(B, (B', \{C_1, \ldots, C_{n-1} \rightarrow C_n\}))$ is an RFD when $(B', \{C_1, \ldots, C_{n-1} \rightarrow C_n\})$ is an AFD for all subtrees with root in the nodes reached following B from *root*

Consistency wrt FDs



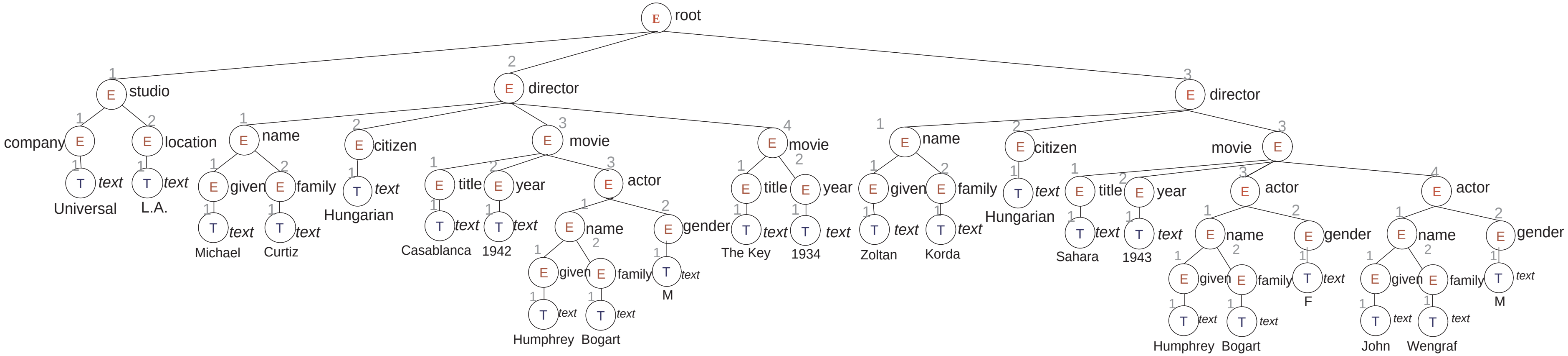
$T_1 \models FD$ for $FD = \{root.director\{name\}, root.director\{name \rightarrow citizen\}\}$

Given a set of FDs FD and an XML tree T that conforms to the MSC statements associated to FD:

T is *consistent wrt* FD, denoted $T \models FD$, iff T satisfies all the FDs in FD

Otherwise, T is *inconsistent*

Queries



We can use $\mathcal{L}_{XML}$-formulas to express queries to XML trees

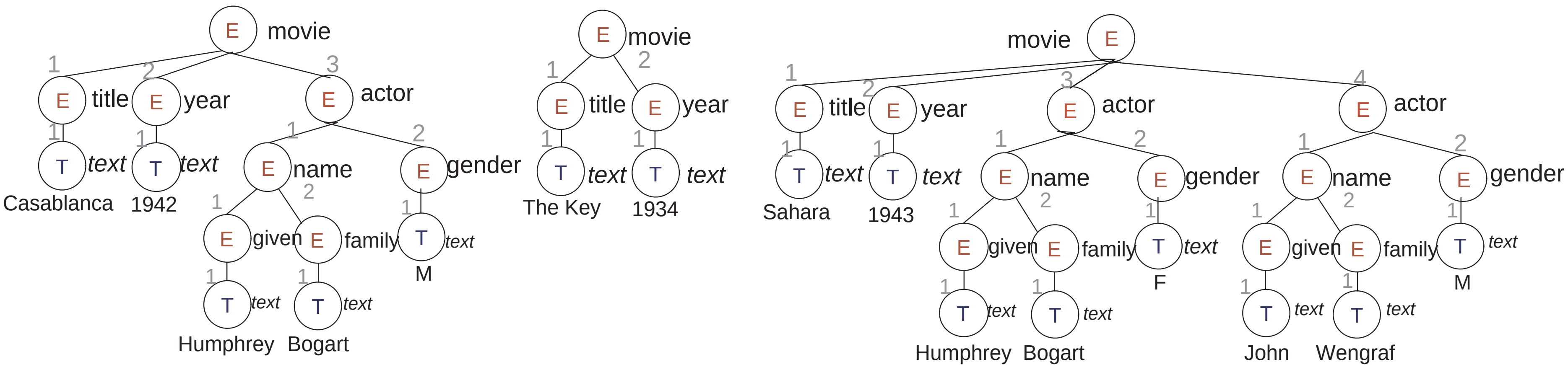
- $Q_1(x): \quad root \cdot director(x)$ posed to T_1 asks for all directors
- $Q_2(x): \quad root \cdot director \cdot name(x)$ asks for all director names
- $Q_3(x): \quad root \cdot director \cdot movie(x) \wedge$
  $\forall y(\neg root \cdot director \cdot movie(y) \vee x \cdot title \neq_V y \cdot title \vee x =_A y))$
  asks for movies with a unique title

The answer to a query $Q(x)$ from T:

- Is the set of subtrees of T
- with root in $v \in T$ such that $address(v, T) = \sigma(x)$ and $T \models_\sigma Q(x)$

For example, for T_1 and $Q(x)$: $root \cdot director \cdot movie(x)$

The answer contains the following trees:



Repairs

An XML tree T may not satisfy a given set FD of FDs

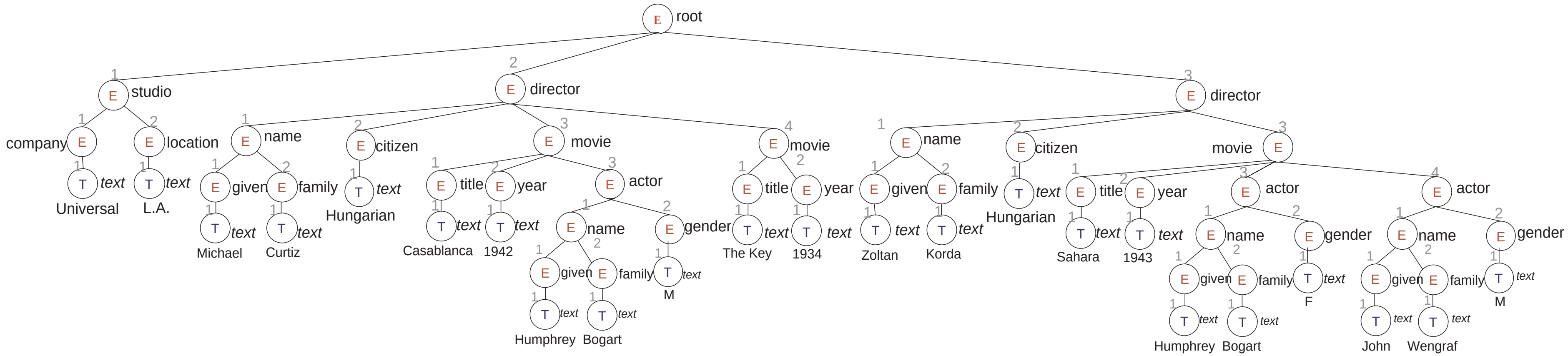
Inconsistencies wrt FDs can be solved by eliminating subtrees

To restore consistency wrt FD, we use the operation $eliminate(a)$

To eliminate subtrees whose root has address a in T corresponds to a conflicting node wrt one or more of the FDs

We eliminate *minimal* subtrees, then a *repair* of T is a maximal subtree T' that satisfies FD

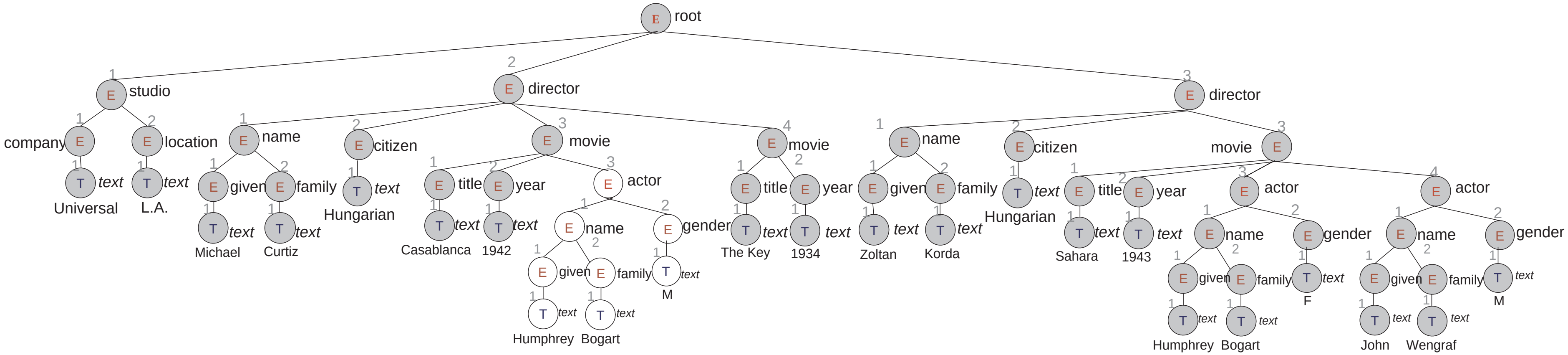
The subtrees will have *absolute addresses*, i.e. their nodes will retain the addresses they had in T (in order to compare nodes in different repairs)

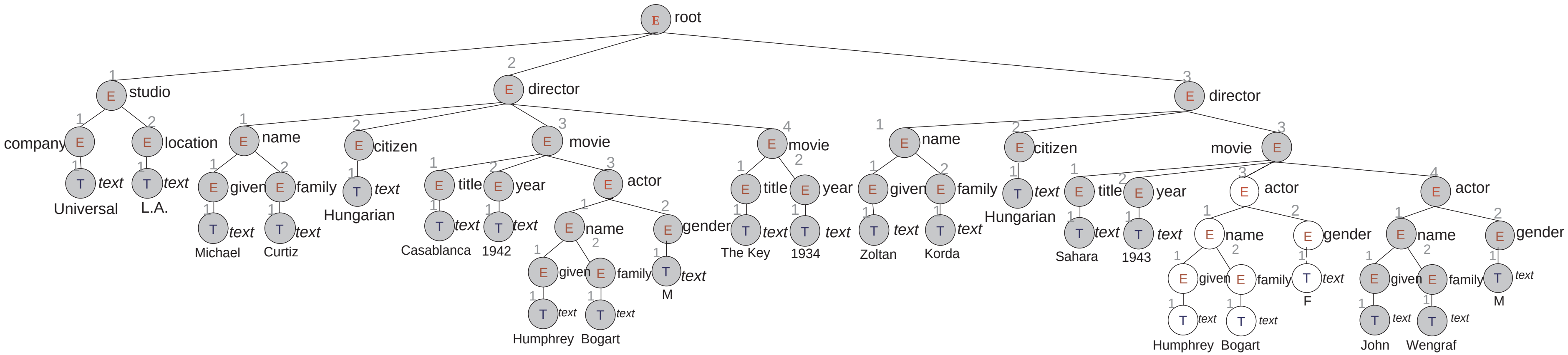


For the AFD $(root \cdot director \cdot movie(actor\{name \rightarrow gender\}))$

Repairs are obtained by applying the operations $eliminate([0, 2, 3, 3])$ and $eliminate([0, 3, 3, 3])$, resp.

(The *actor* nodes that identify *Humphrey Bogart* as a *male* and *female* resp.)


root
studio
director
director
company
location
name
citizen
movie
movie
name
citizen
movie
text
Universal
L.A.
given
family
Michael
Curtiz
Hungarian
title
year
actor
Casablanca
1942
name
gender
given
family
M
Humphrey
Bogart
The Key
1934
Zoltan
Korda
Hungarian
Sahara
1943
actor
actor
name
gender
F
Humphrey
Bogart
John
Wengraf
M


root
studio
director
director
company
location
name
citizen
movie
movie
name
citizen
movie
text
Universal
L.A.
given
family
Michael
Curtiz
Hungarian
title
year
actor
Casablanca
1942
name
gender
given
family
M
Humphrey
Bogart
The Key
1934
Zoltan
Korda
Hungarian
Sahara
1943
actor
actor
name
gender
F
Humphrey
Bogart
John
Wengraf
M

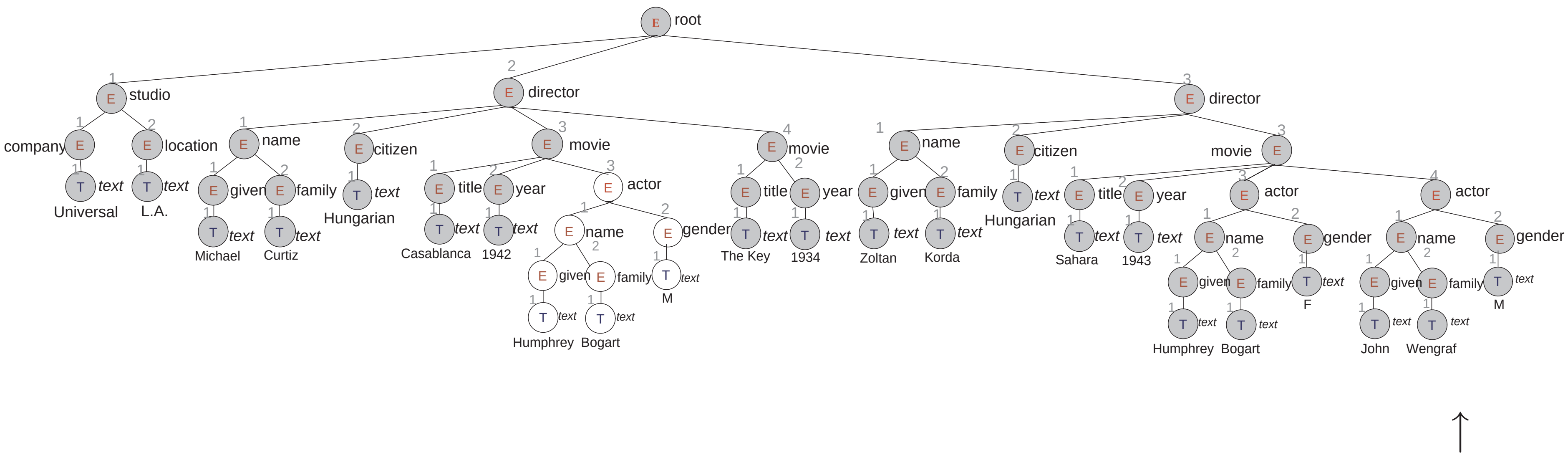
Consistent Answers

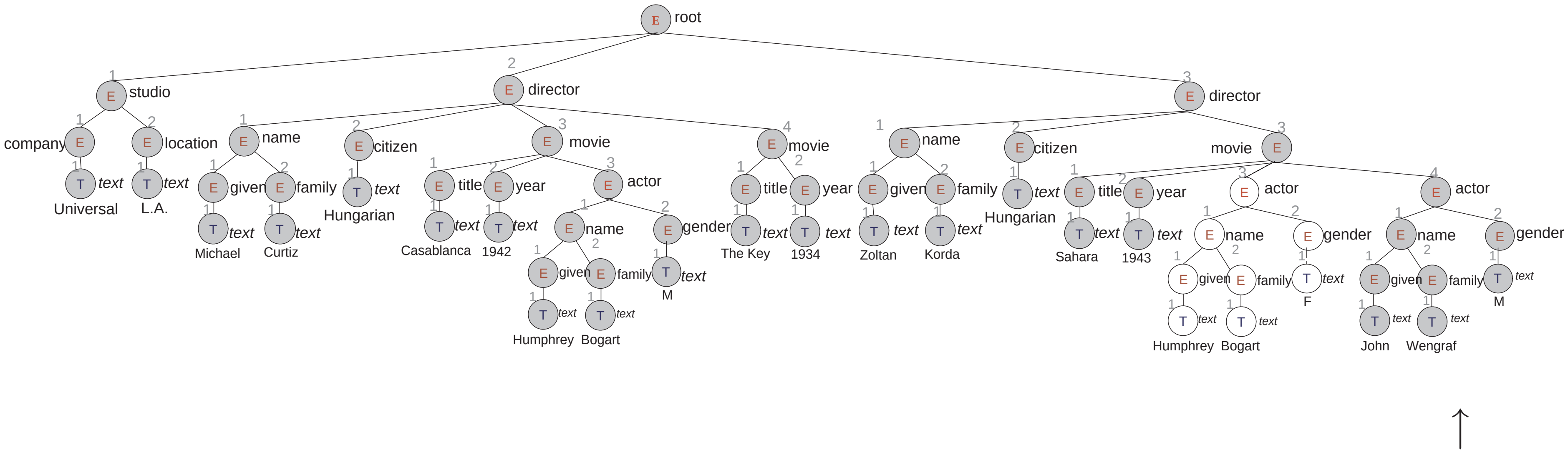
An an answer to a query $Q(x)$ in T is a *consistent answer* wrt FD if it is an answer to Q in every repair of T wrt FD

For the query $Q_4(x)\colon\ root \cdot director \cdot movie \cdot actor(x)$

AFD $(root \cdot director \cdot movie(actor\{name \rightarrow gender\}))$:

The answer is the subtree rooted at $[0, 3, 3, 4]$, the only answer in common to all repairs


root
studio
director
director
company
location
name
citizen
movie
movie
name
citizen
movie
text
text
given
family
text
title
year
actor
title
year
given
family
text
title
year
actor
actor
Universal
L.A.
Hungarian
Hungarian
text
text
text
text
name
gender
text
text
text
text
text
text
name
gender
name
gender
Michael
Curtiz
Casablanca
1942
The Key
1934
Zoltan
Korda
Sahara
1943
given
family
text
given
family
text
given
family
text
M
F
M
text
text
text
text
text
text
Humphrey
Bogart
Humphrey
Bogart
John
Wengraf


root
studio
director
director
company
location
name
citizen
movie
movie
name
citizen
movie
text
text
given
family
text
title
year
actor
title
year
given
family
text
title
year
actor
actor
Universal
L.A.
Hungarian
Hungarian
text
text
text
text
name
gender
text
text
text
text
text
text
name
gender
name
gender
Michael
Curtiz
Casablanca
1942
The Key
1934
Zoltan
Korda
Sahara
1943
given
family
text
given
family
text
given
family
text
M
F
M
text
text
text
text
text
text
Humphrey
Bogart
Humphrey
Bogart
John
Wengraf

Computing Consistent Answers (Sometimes)

We want to compute consistent answers to simple queries by applying simple query rewriting algorithms

As in the case of relational selection-free conjunctive queries and FDs, the mechanism will be based on the syntactic interaction between queries and FDs

From this interaction, *residues* to be appended to the original query are obtained

In some cases, query rewriting alone will not suffice, and a *query reconstruction process* will also be applied

The need for the extra step will depend upon the relationship between the *objective paths of the FDs* and the *path of the query*

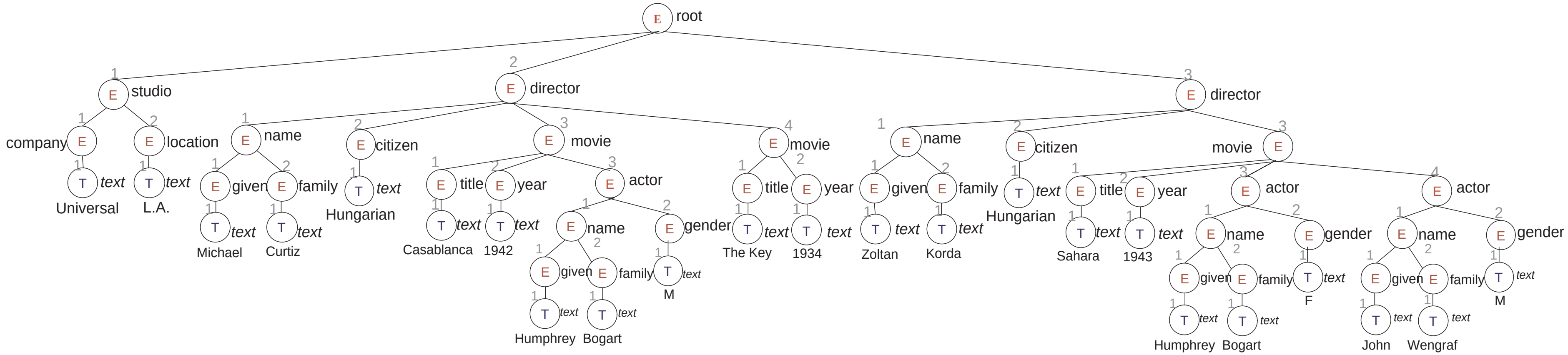
We will basically consider queries that ask for nodes (subtrees) that are reachable through a given path

We need the notion of *objective path formula* associated to the objective path of an FD:

- For an AKC $(B, \{C_1, \ldots, C_n\})$: $B(x)$
- For an RKC $(B, (B', \{C_1, \ldots, C_n\}))$: $B(x) \wedge\ x \cdot B'(x_1)$
- For an AFD $(B, \{C_1, \ldots, C_{n-1} \rightarrow C_n\})$: $B(x)$
- For an RFD $(B, (B', \{C_1, \ldots, C_{n-1} \rightarrow C_n\}))$: $B(x) \wedge\ x \cdot B'(x_1)$

Here $x, x_1 \in VAR$

Query Rewriting



FD:

- $\varphi_1: \ (root, (director\{movie{\cdot}title\}))$ (RKC)
- $\varphi_2: \ (root{\cdot}director{\cdot}movie{\cdot}actor\{name \rightarrow gender\})$ (AFD)

Want consistent answers to $Q_2(x): \ root{\cdot}director{\cdot}movie{\cdot}actor(x)$

1. Take each of the FDs and write it as a denial constrain

$$\varphi_1 \mapsto \neg\bar{\exists}[root(x) \ \wedge \ x\cdot director(x_1) \wedge x\cdot director(x_2) \ \wedge \ x_1\cdot movie\cdot title =_V x_2\cdot movie\cdot title \ \wedge \ x_1 \neq_A x_2]$$

$$\varphi_2 \mapsto \neg\bar{\exists}[root\cdot director\cdot movie\cdot actor(x_1) \ \wedge \ root\cdot director\cdot movie\cdot actor(x_2) \ \wedge \ x_1\cdot name =_V x_2\cdot name \ \wedge \ x_1\cdot gender \neq_V x_2\cdot gender]$$

2. Create for each FD its objective path formula:

FD	objective path	objective path formula
φ_1	$root.director$	$root(x) \wedge x\cdot director(x_1)$
φ_2	$root.director.movie.actor$	$root.director.movie.actor(x)$

3. Rewrite Q_2 to make the variables in the objective path formulas appear in the query

   $Q_2'(x_2)$: $root(x) \wedge x.director(x_1) \wedge x_1.movie.actor(x_2)$

4. Compute the *residues* by cancelling the objective path formulas from the FDs when they appear in the query and leaving the rest (a resolution step)

   The residues are appended to the original query

The rewritten query is:

$$\begin{aligned}
Q_2''(x_2):\ & root.(x) \wedge x.director(x_1) \ \wedge x_1.movie.actor(x_2) \wedge \\
& \neg\exists x_3(x.director(x_3) \wedge x_1.movie.title =_V x_3.movie.title \ \wedge \ x_1 \neq_A x_3) \\
& \ \wedge \\
& \neg\exists x_4(root.director.movie.actor(x_4) \wedge x_2.name =_V x_4.name \wedge \\
& \ x_2.gender \neq_V x_4.gender)
\end{aligned}$$

This query is posed to the original tree T_1 and its only answer is the consistent answer to the original query: The subtree rooted at the node with address $[0, 3, 3, 4]$

A query like this can be translated into a query written in XQuery and posed to the corresponding XML document

No need for a computation of repairs ...

A Little Problem

In the previous example the objective paths of the FDs were subpaths of the query path

In those cases, a rewriting like the one above is good enough

However, when this condition is not satisfied, an additional step is necessary

$Q_3(x):\ root \cdot director(x)$

$\varphi:\ (root \cdot director \cdot movie \cdot actor\{name \to gender\})$ (AFD)

There is no way to append a residue in a sensible way

This form of interaction between FDs and queries must be treated differently

1. First, specialize the query to
$$Q_{3*}(z):\ \ root\cdot director\cdot movie\cdot actor(z)$$
whose query path is equal to the objective path of the FD

2. Append the residues to Q_{3*} as before

3. Evaluate the rewritten query Q'_{3*}

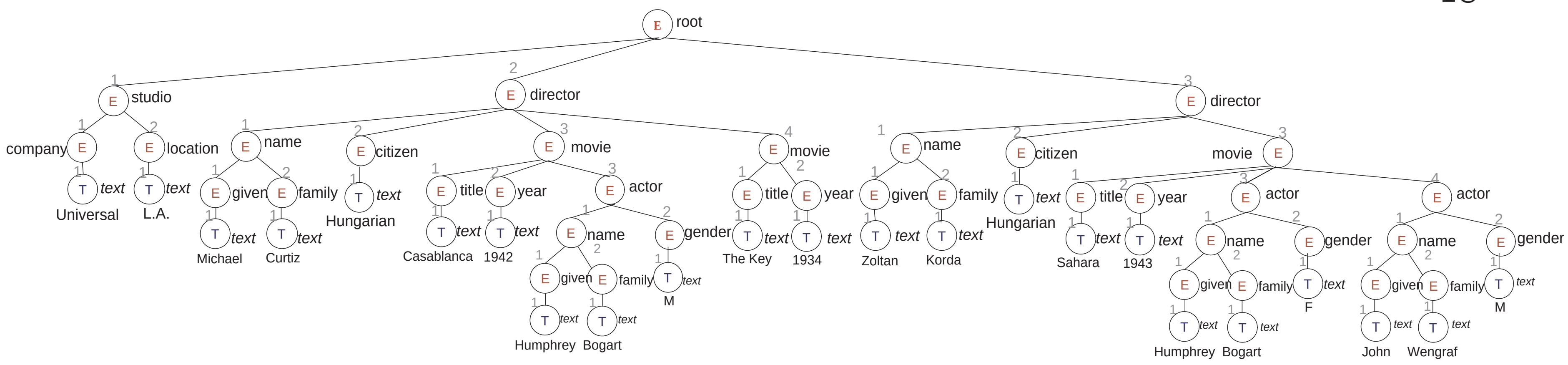
   We obtain the subtree with root at the node with address $[0, 3, 3, 4]$

   The consistent answer to Q_{3*} represents a portion of the consistent answer to Q_3

4. The consistent answer to Q_{3*} is "reconstructed upwards" the XML tree to obtain the consistent answer to Q_3

To obtain this answer, all the FDs must be used, which ensures that the answer reconstructed from the consistent answer to Q_{3*} is a consistent answer to Q_3

Upwards the XML tree, we have to include nodes that are part of the consistent answer to the original query



In order to obtain a consistent answer to $Q_3(x)\colon\ root.director(x)$ from the consistent answer to $Q_{3*}(z)\colon\ root.director.movie.actor(z)$

- We have to restore nodes, like *movie* to the answer to Q_{3*}
- But also child nodes of those recovered "upwards"

  1. Not every child can be added

     In the example, by adding node *movie*, and then all *actors*, we may be adding actors that violate some FDs on *actors*
  2. We also have to check if the reconstruction process satisfies the MSC associated to the FDs

     For example, we cannot add a *director* without a *name*

We have developed a “consistent answer reconstruction” algorithm

It is based on logic programs with annotations

Annotations that are used to detect and react to possible semantic conflicts

Which determines the nodes and the children to add to the answer obtained after the specialization

Many Open Issues

- Use of other logical languages for XML to express FDs and queries

  Core XPath? [M. Marx]

- Complexity of the reconstruction process

- Precise application scope of the algorithm?

  It works for some path queries with projection and selection, e.g.

  $Q_8(x):\ root.director(y) \wedge y.name(x) \wedge y.movie.year.text =_V \text{“1942”}$

- More expressive queries?

- More generally, repairs wrt to arbitrary DTDs that go beyond the MSC associated to the FDs

- Etc.