March 1999

Inference Rules for Nested Functional Dependencies

Carmem S. Hara
*University of Pennsylvania*

Susan B. Davidson
*University of Pennsylvania*, susan@cis.upenn.edu

Follow this and additional works at: [http://repository.upenn.edu/cis_reports](http://repository.upenn.edu/cis_reports)

**Recommended Citation**
[http://repository.upenn.edu/cis_reports/53](http://repository.upenn.edu/cis_reports/53)

This paper is posted at ScholarlyCommons. [http://repository.upenn.edu/cis_reports/53](http://repository.upenn.edu/cis_reports/53)
For more information, please contact repository@pobox.upenn.edu.
Inference Rules for Nested Functional Dependencies

Abstract
Functional dependencies add semantics to a database schema, and are useful for studying various problems, such as database design, query optimization and how dependencies are carried into a view. In the context of a nested relational model, these dependencies can be extended by using path expressions instead of attribute names, resulting in a class of dependencies that we call nested functional dependencies (NFDs). NFDs define a natural class of dependencies in complex data structures; in particular they allow the specification of many useful intra- and inter-set dependencies (i.e., dependencies that are local to a set and dependencies that require consistency between sets). Such constraints cannot be captured by existing notions of functional, multi-valued, or join dependencies.

This paper presents the definition of NFDs and gives their meaning by translation to logic. It then presents a sound and complete set of eight inference rules for NFDs, and discusses approaches to handling the existence of empty sets in instances. Empty sets add complexity in reasoning since formulas such as $\forall x \in R. P(x)$ are trivially true when $R$ is empty. This axiomatization represents a first step in reasoning about constraints on data warehouse applications, where both the source and target databases support complex types.

Comments
Abstract

Functional dependencies add semantics to a database schema, and are useful for studying various problems, such as database design, query optimization and how dependencies are carried into a view. In the context of a nested relational model, these dependencies can be extended by using path expressions instead of attribute names, resulting in a class of dependencies that we call nested functional dependencies (NFDs). NFDs define a natural class of dependencies in complex data structures; in particular they allow the specification of many useful intra- and inter-set dependencies (i.e., dependencies that are local to a set and dependencies that require consistency between sets). Such constraints cannot be captured by existing notions of functional, multi-valued, or join dependencies.

This paper presents the definition of NFDs and gives their meaning by translation to logic. It then presents a sound and complete set of eight inference rules for NFDs, and discusses approaches to handling the existence of empty sets in instances. Empty sets add complexity in reasoning since formulas such as $\forall x \in R. P(x)$ are trivially true when $R$ is empty. This axiomatization represents a first step in reasoning about constraints on data warehouse applications, where both the source and target databases support complex types.

1 Introduction

Dependencies add semantics to a database schema and are useful for studying various problems such as database design, query optimization and how dependencies are carried into a view. In the context of the relational model, a wide variety of dependencies have been studied, such as functional, multivalued, join and inclusion dependencies (see [13, 2] for excellent overviews of this work). However, apart from notions of key constraints and inclusion dependencies [5, 16], dependencies in richer models than the relational model have not been as thoroughly studied.

Complex data models are, however, heavily used within biomedical and other scientific database applications. Reasoning about dependencies within these applications is becoming increasingly important as schemas get larger, queries span multiple complex databases, and new databases are created as materialized views. For example, if a new database is created as a materialized view over multiple complex databases, knowing how dependencies are carried into this complex view could eliminate expensive checking as the new database is created and later updated.
We therefore start attacking this problem by defining a notion of functional dependency for the nested relational model together with inference rules for these dependencies. We are considering the nested relational model, where set and tuple constructors are required to alternate, mainly for simplicity, but relaxing this assumption does not significantly change the inference rules. Since in this model attributes of a relation may be sets rather than atomic types, dependencies may traverse into various levels of nesting through paths. We call this new form of functional dependencies nested functional dependencies (NFDs).

As an example of what we would like to be able to express, consider a type Course defined as a set of records with attributes cnum, time, students, and books, where students is a set of records with labels sid, age, and grade, and books is a set of records with labels isbn, and title:

\[
\text{Course} : \{ \langle \text{cnum, time} \rangle, \\
\text{students} : \{ \langle \text{sid, age, grade} \rangle \}, \\
\text{books} : \{ \langle \text{isbn, title} \rangle \} \}.
\]

Some nested functional dependencies that we would like to be able to express for Course are:

1. cnum is a key.
2. Every Course instance is consistent on their assignment of title to a given isbn.
3. In a given course, each student gets a single grade.
4. Every Course instance is consistent on their assignment of age to sid.
5. A student cannot be enrolled in courses that overlap on time.

Note that there are “local” dependencies, such as dependency 3 where a student can have only one grade for a given course but may have different grades for distinct courses. There are also “global” dependencies such as dependencies 2 and 4, where the assignment of title to an isbn and age to sid must be consistent throughout the Course relation. Dependency 5 illustrates how an attribute from an outer level of nesting may be determined by attributes in a deeper level of nesting. Note that even if every level of nesting presents a “key” as suggested in [1], this type of dependency is not captured by the structure of the data.

Our definition of NFDs can also be used to express other interesting properties of sets. For example, they can be used to state that some fields in a set valued attribute are required to be disjoint, or that a set is expected to be a singleton. In AceDB [19], a database which is very popular among biologists, every attribute is defined as a set. This is useful in applications where the database is sparsely populated and evolves over time, since empty sets can model optional or undefined attributes. However, some attributes can be specified to be (maximally) singleton sets. In order to reason about constraints in this model, it is therefore important to be able to express the fact that a set must be a singleton. The importance of singleton sets is also evident in [7], which investigates when functional dependencies are maintained or destroyed when relations are nested and unnested. In most cases, this relies on knowing whether a set is a singleton or multivalued.

One of the most interesting questions involving dependencies is that of logical implication, i.e., deciding if a new dependency holds given a set of existing dependencies. For functional dependencies in the
relational model, this problem has been addressed from two different perspectives: a decision procedure called the tableau chase, and a sound and complete set of inference rules called Armstrong’s axioms.

As an example of an inference we might want to make over the complex type Course, suppose we have a database DBCourse which is known to satisfy all the dependencies listed above. We wish to know if in DBCourse, given a student ID sid, and a time, there is a unique set of books used by the student at that time. Reasoning intuitively, the answer is affirmative since a student can be enrolled in only one course cnum in a given time, and cnum, which is a key, determines a set books. However, it would be useful to have a technique and inference rules to prove this.

The development of inference rules is important for many reasons [4]: First, it helps us gain insight into the dependencies. Second, it may help in discovering efficient decision procedures for the implication problem. Third, it provides tools to operate on dependencies. For example, in the relational model, it provides the basis for testing equivalence preserving transformations, such as lossless-join decomposition, and dependency preserving decomposition, which lead to the definition of normal forms of relations, a somewhat more mechanical way to produce a database design [20].

We therefore focus in this paper on the development of a sound and complete set of inference rules for NFDs. However, the presence of empty sets in instances causes serious problems in developing such rules since formulas such as $\forall x \in R.P(x)$ are trivially true when $R$ is empty. We therefore initially restrict the inference problem to the case where empty sets cannot occur in any instance, and then suggest how this assumption can be relaxed by specifying where empty sets are known not to occur.

The remainder of the paper is organized as follows: Section 2 describes our nested relational model, the definition of nested functional dependencies in this model, and their translation into logic. We also contrast our approach to others taken in the literature. Section 3 presents the axiomatization of NFDs, illustrates their use on some examples, and discusses how empty sets in instances can cause problems. Section 4 concludes the paper and discusses some future work.

2 Functional Dependencies for the Nested Relation Model

The nested relational model has been well studied (see [2] for an overview). It extends the relational model by allowing the type of an attribute to be a set of records or a base type, rather than requiring it to be a base type (First Normal Form). For simplicity, we use the strict definition of the nested model and require that set and tuple constructors alternate, i.e. there are no sets of sets or tuples with a tuple component, although allowing nested records or sets does not substantially change the results established. For ease of presentation, we also assume that there are no repeated labels in a type, i.e., $\langle A : \text{int}, B : \{\langle A : \text{int}\rangle\}\rangle$ is not allowed.

An example of a nested relation was given by Course in the previous section.

More formally, a nested relational database $\mathcal{R}$ is a finite set of relation names, ranged over by $R_1, R_2, \ldots$. $\mathcal{A}$ is a fixed countable set of labels, ranged over by $A_1, A_2, \ldots$, and $\mathcal{B}$ is a fixed finite set of base types, ranged over by $\mathbb{b}, \ldots$.

The data types Types are as follows:

$$\tau ::= \mathbb{b} \mid \{\tau\} \mid \langle A_1 : \tau_1, \ldots, A_n : \tau_n\rangle$$

Here, $\mathbb{b}$ are base types, e.g. boolean, integer and string. The notation $\{\omega\}$ represents a set with elements
of type $\omega$, where $\omega$ must be a record type. $<a_1 : \tau_1, \ldots, a_n : \tau_n>$ represents a record type with fields $A_1, \ldots, A_n$ of types $\tau_1, \ldots, \tau_n$, respectively. Each $\tau_i$ must either be a base or a set type.

A **database schema** is a pair $(R, S)$, where $R$ is a finite set of relation names, and $S$ is a schema mapping $S : R \rightarrow \text{Types}$, such that for any $R \in R$, $R \xrightarrow{S} \tau^R$ where $\tau^R$ is a set of records in its outermost level.

**Denotations of types.** Let us denote by $D^b$ the domain of the base type $b$ for any $b$. The domain of our model $D$ is defined as the least set satisfying the equation:

$$D \equiv \bigcup_{b} D^b \cup A \xrightarrow{A} D \cup P_{fin}(D)$$

where $A \xrightarrow{A} B$ denotes the set of partial functions from $A$ to $B$.

Given a schema $(R, S)$, the interpretation of each type $\tau$ in $\text{Types}^R$, $[[\tau]]$, is defined by

$$[[b]] \equiv D^b$$

$$[[\{\tau\}]] \equiv P_{fin}([[\tau]])$$

$$[[< A_1 : \tau_1, \ldots, A_n : \tau_n >]] \equiv \{ f \in A \xrightarrow{A} D \mid \text{dom}(f) = \{A_1, \ldots, A_n\} \text{ and } f(A_i) \in [[\tau_i]], i = 1, \ldots, n \}$$

**Database instance:** A **database instance** of a database schema $(R, S)$ is a record $I$ with labels in $R$ such that $\pi_R I$ is in $[[S(R)]]$ for each $R \in R$.

We denote by $I_{SC}$ the set of all instances of schema $SC$.

As an example, if $\{\text{Course}\}, S$ is a schema where

$$S(\text{Course}) = \{ <\text{cnum} : \text{string},$$

$$\text{time} : \text{int},$$

$$\text{students} : \{ <\text{sid} : \text{int},$$

$$\text{grade} : \text{string} > \} > \}.$$

Then the following is an example of an instance of this schema:

$$<\text{Course} \mapsto \{ <\text{cnum} \mapsto "\text{cis550}" ,$$

$$\text{time} \mapsto 10, $$

$$\text{students} \mapsto \{ <\text{sid} \mapsto 1001, $$

$$\text{grade} \mapsto "A" > ,$$

$$<\text{sid} \mapsto 2002, $$

$$\text{grade} \mapsto "B" > \} > ,$$

$$<\text{cnum} \mapsto "\text{cis500}" ;$$

$$\text{time} \mapsto 12, $$

$$\text{students} \mapsto \{ <\text{sid} \mapsto 1001, $$

$$\text{grade} \mapsto "A" > \} >$$

$$<\text{cnum} \mapsto "\text{cis500}" ;$$

$$\text{time} \mapsto 12, $$

$$\text{students} \mapsto \{ <\text{sid} \mapsto 1001, $$

$$\text{grade} \mapsto "A" > \} >$$

4
2.1 Nested Functional Dependencies

The natural extension of a functional dependency \( X \rightarrow A \) for the nested relational model is to allow path expressions in \( X \) and \( A \) instead of attributes. That is, \( X \) is a set of paths and \( A \) is a single path. As an example, the requirement that a student’s age in Course be consistent throughout the database could be written as \( \text{Course} : [\text{students} : \text{sid} \rightarrow \text{students} : \text{age}] \), where “\( \rightarrow \)” indicates traversal inside a set. Note that we have enclosed the dependency in square brackets “\( [] \)” and appended the name of the nested relation, Course.

Path Expressions

We start by giving a very general definition of path expressions, and narrow them to be well-defined by a given type.

**Definition 2.1** Let \( \mathcal{A} = A_1, A_2, \ldots \) be a set of labels. A **path expression** is a string over the alphabet \( \mathcal{A} \cup \{ : \} \). \( \epsilon \) denotes the empty path. A path expression \( p \) is **well-typed** with respect to type \( \tau \) if

- \( p = \epsilon \), or
- \( p = Ap' \) and \( \tau \) is a record type \( \langle A : \tau', \ldots \rangle \) and \( p' \) is well-typed with respect to \( \tau' \), or
- \( p = [:p' \rangle \) and \( \tau \) is a set type \( \{ \tau' \} \) and \( p' \) is well-typed with respect to \( \tau' \).

As an example, \( A : B \) is well-typed with respect to \( \langle A : \{B : \text{int}, C : \text{int}\} \rangle \), but not with respect to \( \langle A : \text{int} \rangle \).

Given an object \( e \), the semantics of path expressions is given by:

\[
\begin{align*}
\llbracket \epsilon \rrbracket & \equiv \llbracket e \rrbracket \\
\llbracket A \ e \rrbracket & \equiv \llbracket e \rrbracket (A) \\
\llbracket [: \ e \rrbracket & \equiv \begin{cases} 
\text{undefined}, & \text{if } \llbracket e \rrbracket = \{ \} \\
\llbracket e_1 \rrbracket, & \text{otherwise, where } \llbracket e_1 \rrbracket \\
& \text{is an element of } \llbracket e \rrbracket
\end{cases}
\end{align*}
\]

Note that the value of a path expression that traverses into an empty set is undefined, i.e., it does not yield a value in the database domain. We say that a path expression \( p \) is **well defined** on \( v \) if it always yields a value in the database domain.

As an example, if

\[
v = \langle A \mapsto \{ \langle B \mapsto 10, C \mapsto 20 \rangle, \\
\langle B \mapsto 15, C \mapsto 21 \rangle \} \rangle
\]

then

- \( A(v) = \{ \langle B \mapsto 10, C \mapsto 20 \rangle, \\
\langle B \mapsto 15, C \mapsto 21 \rangle \} \)
\* \* A : B(v) = 10 or A : B(v) = 15 \*

To help define nested functional dependencies, we introduce the notions of path prefix and size of a path expression.

**Definition 2.2** Path expression \( p_1 \) is a prefix of \( p_2 \) if \( p_2 = p_1 p_2' \). Path \( p_1 \) is a proper prefix of \( p_2 \) if \( p_1 \) is a prefix of \( p_2 \) and \( p_1 \neq p_2 \).

**Definition 2.3** The size of a path expression of the form \( p = A_1 : \ldots : A_k \), denoted as \( |p| \), is \( k \), the number of labels in \( p \).

With these notions, we are now in a position to define nested functional dependencies (NFDs), and how an instance is said to satisfy an NFD.

**Definition 2.4** Let \( SC = (R, S) \) be a schema. A nested functional dependency (NFD) over \( SC \) is an expression of the form \( x_0 : [x_1, \ldots, x_{m-1} \rightarrow x_m], m \geq 1 \), such that all \( x_i, 0 \leq i \leq m \), are \( \tau \)-well-typed path expressions with respect to \( \tau R \).

In general, the base path \( x_0 \) can be an arbitrary path rather than just a relation name. For the degenerate case where \( m = 1 \), i.e. the NFD is of form \( x_0 : [\emptyset \rightarrow x_m] \), then in any value of \( x_0 : x_m \) must be a constant.

**Definition 2.5** Let \( f = x_0 : [x_1, \ldots, x_{m-1} \rightarrow x_m] \) be an NFD over schema \( SC \), \( I \) an instance of \( SC \), and \( v_1, v_2 \) two values of \( x_0 : (I) \) in the database domain. \( I \) satisfies \( f \), denoted \( I \models f \), if for all \( v_1, v_2 \), whenever

1. \( x_i(v_1) = x_i(v_2) \) for all \( 1 \leq i < m \), and
2. for every path \( x \) which is a common prefix of \( x_i, x_j, 1 \leq i, j \leq m \), \( x(v_1) \) coincide in \( x_i(v_1) \) and \( x_j(v_1) \) and \( x(v_2) \) coincide in \( x_i(v_2) \) and \( x_j(v_2) \) (i.e. \( x_i \) and \( x_j \) follow the same path up to \( x \) in \( v_1 \) and in \( v_2 \))

then

\[ x_m(v_1) = x_m(v_2) \]

If for some \( x_i, 1 \leq i \leq m \), \( x_i(v_1) \), or \( x_i(v_2) \) is undefined, we say \( f \) is trivially true.

In the next section, we give a translation of NFD to logic to precisely define its semantics.

Our definition of NFDs is very broad, and captures many natural constraints. As an example, we can precisely state the constraints on Course described in the introduction.

**Example 2.1** In Course, \( cnum \) is a key.

\* Course : [\( cnum \rightarrow time \)]
\* Course : [\( cnum \rightarrow students \)]
\* Course : [\( cnum \rightarrow books \)]
Example 2.2 For any two instances in Course, if they agree on isbn for some element of books then they must also agree on title for that element of books.
\[ \text{Course} : [\text{books} : \text{isbn} \rightarrow \text{books} : \text{title}] \]

Example 2.3 In a given course, each student gets a single grade.
\[ \text{Course} : \text{students} [\text{sid} \rightarrow \text{grade}] \]

Note that in this example, sid is a “local” key to grade; this illustrates the use of a path rather than just a relation name outside the “[]”. Contrast this to the previous example, where the NFD requires that isbn and title be consistent throughout the database.

Example 2.4 Every Course instance is consistent on their assignment of age to sid.
\[ \text{Course} : [\text{students} : \text{sid} \rightarrow \text{students} : \text{age}] \]

Example 2.5 A student cannot be enrolled in courses that overlap on time.
\[ \text{Course} : [\text{time}, \text{students} : \text{sid} \rightarrow \text{cnum}] \]

Some interesting properties of sets can also be expressed by NFDs. For example, if an instance \( I \) satisfies an NFD of the form \( x_0 : [x_1 : x_2 \rightarrow x_1] \), then given two values \( v_1, v_2 \) of \( x_0 : x_1(I) \), either \( v_1 = v_2 \), or \( v_1 \cap v_2 = \emptyset \).

As an example, suppose that a university’s courses database is defined as \( \text{Courses} : \{<\text{school}, s\text{courses} : \{<\text{cnum}, \text{time}>\}>\} \), and it satisfies the NFD \( \text{Courses} : [s\text{courses} : \text{cnum} \rightarrow \text{school}] \). We can conclude that schools in the university do not share course numbers, because the existence of the same cnum in different schools would violate the NFD.

NFDs can also express that if a set is not empty then it must be a singleton. I.e., if an instance \( I \) satisfies an NFD of the form \( x_0 : [x_1, \ldots, x_m \rightarrow x_n : A] \), where \( x_n \) is not a proper prefix of any \( x_i \), \( 1 \leq i \leq m \), then for any value \( v \) of \( x_0 : (I) \) in which paths \( x_1 \ldots x_m \) are well-defined, all elements \( e \) of \( x_n(v) \) have the same value for \( A(e) \).

For example, let \( R \) be a relation with schema \( \{<A : \{<B : \text{int}, C : \text{int}>\}, D : \text{int}>\} \). If \( R : [D \rightarrow A : B] \), and \( R : [D \rightarrow A : C] \), then it must be the case that \( A \) is either empty, or a singleton set, since for every value of \( A \) all elements agree on the values of \( B \) and \( C \). Since these are the only attributes in \( A \), then \( A \) has a single element.

It should be noted that our definition also allows some uninteresting NFDs. For example, assume \( R : \{<A, B : \{<C, D>\}, E : \{<F, G>\}>\} \). Then the NFD \( R : [B : C \rightarrow E : F] \) implies that:

- all tuples \( <F, G> \) in \( E \) have the same value for \( F \) when \( B \) is not empty, and
- if any tuple \( <C, D> \) in \( B \) agrees on the value of \( C \), then the elements \( <F, G> \) in \( E \) must have the same value for \( F \).

Figure 1 shows an instance of \( R \) that does not satisfy \( R : [B : C \rightarrow E : F] \). If we only consider the first line in the table, the NFD is satisfied since all values of attribute \( F \) coincide, i.e. \( B : C = 1 \) determines

1 Note that values of \( x_0 : x_1(I) \) must be of set type.
Figure 1: An instance that violates $R: [B : C \rightarrow E : F]$.

$E : F = 5$. The existence of more than one value for $F$ automatically invalidates the constraint because a single value in $C$ would be related to distinct values in $F$ as in the second line. The second line also violates the dependency because it has a value in $B : C$ that also appears in the first line, but has a different value for $E : F$.

2.2 NFDs expressed in logic

In the relational model, a functional dependency $Course : [cnum \rightarrow time, students]$ can be understood as the following formula:

$$\forall c_1 \in Course \forall c_2 \in Course$$
$$\quad (c_1.cnum = c_2.cnum) \rightarrow$$
$$\quad (c_1.time = c_2.time \land c_1.students = c_2.students)$$

There is also a precise translation of NFDs to logic. Intuitively, given an NFD $R : [x_1 \ldots x_{m-1} \rightarrow x_m]$, we introduce two universally quantified variables for $R$ and for each set-valued attribute in $x_1 \ldots x_m^2$. The body of the formula is an implication where the antecedent is the conjunction of equalities of the last attributes in $x_1 \ldots x_{m-1}$ and the consequence is an equality of the last attribute in $x_m$.

As an example, $Course : [students : sid \rightarrow students : age]$ can be translated to the following formula:

$$\forall c_1 \in Course \forall c_2 \in Course$$
$$\forall s_1 \in c_1.students \forall s_2 \in c_2.students.$$ 
$$\quad (s_1.sid = s_2.sid \rightarrow s_1.age = s_2.age)$$

To formalize this translation, we define functions $var$, and $parent$. Let $SC = (R, S)$ be a schema, $I$ an instance of $SC$, and $f = x_0 : [x_1 \ldots x_{m-1} \rightarrow x_m]$ be an NFD defined over $SC$, where $x_i = A_i^1 : \ldots : A_i^{k_i}$, $0 \leq i \leq m$, and $A_i^0 = R$, $R \in R$.

Define $var$ as a function that maps labels to variable names as follows:

- for each label $A$ in $\tau^R$ that appears in some path $x_i$, $0 \leq i \leq m$, $var(A) = v_A$. Recall that we assume labels cannot be repeated.

\[\text{\text{\footnotesize{\textsuperscript{2}}}It is a little more complicated for the general case where the base path can be an arbitrary path rather than $R$.}\]
The function \( parent \) maps a label to the variable defined for its parent as follows:

- for all \( A_i^1, 1 \leq i \leq m \), \( parent(A_i^1) = \text{var}(A_{k_0}^0) \), i.e., the parent of the first labels in paths \( x_1 \ldots x_m \) is the variable associated with the last label in path \( x_0 \).

- \( parent(A_{j+1}^j) = \text{var}(A_j^1) \). Let \( \{A_1^1 \ldots A_k^1\} \) be the set of such \( A_j \) labels, i.e., the set of labels that have some descendent in a path expression.

Also, let \( parent(A_1^0)A_0^0 = R \). Then \( f \) is equivalent to the following logic formula:

\[
\forall v_{A^0} \in parent(A_0^0).A_1^0 \ldots
\forall v_{A^0_{k_0-1}} \in parent(A_{k_0-1}^0).A_{k_0-1}^0
\forall v_{A^0_{k_0}} \in parent(A_{k_0}^0).A_{k_0}^0
\forall v_{A^1_1} \in parent(A_1^1).A_1^1 \forall v_{A^1_2} \in parent(A_1^1).A_1^2 \ldots
\forall v_{A^m_1} \in parent(A_m^1).A_m^1 \forall v_{A^m_2} \in parent(A_m^1).A_m^2
\left( \text{true} \land
parent(A_{k_0}^1).A_{k_0}^1 = parent(A_{k_0}^1).A_{k_0}^1 \land \ldots \land
parent(A_{k_{m-1}}^m).A_{k_{m-1}}^m = parent(A_{k_{m-1}}^m.A_{k_{m-1}}^m)\right)
\rightarrow
\left( parent(A_m^m).A_m^m = parent(A_m^m.A_m^m) \right)
\]

Note that only one variable is mapped to each label in \( A_1^1, \ldots, A_{k_0-1}^0 \), whereas two variables are used elsewhere.

Using this translation, examples 2.2 and 2.3 can be expressed as:

- \( \text{Course} : [\text{books} : \text{isbn} \rightarrow \text{books} : \text{title}] \)
  \( \forall c_1 \in \text{Course} \forall c_2 \in \text{Course} \forall b_1 \in c_1.\text{books} \forall b_2 \in c_2.\text{books} .
  (b_1.\text{isbn} = b_2.\text{isbn} \rightarrow b_1.\text{title} = b_2.\text{title}) \)
  Note that \( \text{books} \) is referred to twice in the dependency, but that only two variables for \( \text{books} \) are introduced in the logical form.

- \( \text{Course} : \text{students} : [\text{sid} \rightarrow \text{grade}] \)
  \( \forall c \in \text{Course} \forall s_1 \in c.\text{students} \forall s_2 \in c.\text{students} .
  (s_1.\text{sid} = s_2.\text{sid} \rightarrow s_1.\text{grade} = s_2.\text{grade}) \)
  Note that only one variable is introduced for labels in \( x_0 \) (except for the last label), and that two variables are introduced for all other labels.

### 2.3 Classification of NFDs

When we discuss an axiomatization for NFDs, it will be useful to refer to three different forms of NFDs: upwards, sideways, and downwards. Each of them behave differently in terms of inferences that can be made. In what follows, let \( f = x_0 : [x_1, \ldots, x_{m-1} \rightarrow x_m] \) be an NFD, \( A, A_1, \ldots \) be labels, and \( y, z \) be path expressions.
Definition 2.6 (upward) \( f \) is upward if \( x_m = yA \) and there exists an \( x_i, 1 \leq i < m \) such that \( x_i = yz \), where \( |z| > 1 \).

The following are examples of upward NFDs: \( R : [A : B \rightarrow A] \), \( R : [C : D \rightarrow E] \), \( R : [F : G : H \rightarrow F : I] \). Note that in the first two NFDs \( y = e \).

Definition 2.7 (sideways) \( f \) is sideways if \( f \) is not upward, \( x_m = yA_m \), and there exists an \( x_i, 1 \leq i < m \) such that \( x_i = yA_i \).

The following are examples of sideways NFDs: \( R : [A \rightarrow B] \), \( R : [B \rightarrow C] \)

Definition 2.8 (downward) \( f \) is downward in all other cases, i.e., if \( x_m = yA, |y| \geq 1 \), and for all \( x_i, 1 \leq i < m \), \( y \) is not a proper prefix of \( x_i \).

\( R : [A \rightarrow B : C] \), \( R : [B : C \rightarrow D : E] \), \( R : [F : G : H \rightarrow F : I : J] \) are examples of downward NFDs.

Intuitively, \( f \) is upward if the value of \( x_m \) is determined by some attribute nested in the same set as \( x_m \). \( f \) is sideways if it is determined by attributes in the same level of nesting; and \( f \) is downward if it is determined by some paths that do not traverse the set that \( x_m \) is nested in.

Some observations follow from these definitions:

Observation 2.1 If an instance \( I \) satisfies an upward NFD \( x_0 : [x_1 : x_2 \rightarrow x_1] \), then given two values \( v_1, v_2 \) of \( x_0 : x_1(I) \), either \( v_1 = v_2 \), or \( v_1 \cap v_2 = \emptyset \).

Observation 2.2 If an instance \( I \) satisfies a downward NFD \( x_0 : [x_1, \ldots, x_m \rightarrow x_n : A] \), then for any value \( v \) of \( x_0(I) \) in which paths \( x_1 \ldots x_m \) are well-defined, all elements \( e \) of \( x_n(v) \) have the same value for \( A(e) \).

2.4 Discussion

In the definition of NFDs, the base path can be an arbitrary path rather than just a relation name. The motivation for allowing this is to syntactically differentiate between local and global functional dependencies: \( R : A : [B \rightarrow C] \) is a local functional dependency in \( A \), while \( R : [A : B \rightarrow A : C] \) defines a global dependency between \( B \) and \( C \). However, the local dependency is provably equivalent\(^4\) to the dependency \( R : [A, A : B \rightarrow A : C] \). Intuitively, by requiring equality on \( A \) (as a set), the dependency between \( B \) and \( C \) becomes local to the set. Therefore, the expressive power of NFDs with arbitrary paths and relation names as base paths are the same. However, we believe that the first form is more intuitive.

Most of the early work on functional dependencies for the nested relational model either used the definition of functional dependencies given for the relational model \([15]\), or proposed a simple extension to allow equality on sets \([11]\). Our definition clearly subsumes these definitions.

\(^3\)Note that values of \( x_0 : x_1(I) \) must be of set type.

\(^4\)The equivalence of these two forms is proved in the next section.
The idea of extending functional dependencies to allow path expressions instead of simple attribute names has been investigated by Weddell [22] and also by Tari et al. [18] in the context of an object-oriented data model. While Weddell’s work supports a data model of classes, where each class is associated with a simple type (a flat record type), our model supports a nested relational model with arbitrary levels of nesting. In [22], following a path entails an implicit “dereference” operation, while in NFDs following a path means traversal into an element of a nested set. They present a set of inference rules and prove they are complete. We believe this work and ours are complementary and that it would be interesting to investigate how the two approaches could be combined into a single framework.

In [18], more general forms of functional dependencies for the object-oriented model are proposed. Their model supports nested sets, and classes of objects, and the dependencies allow inter- and intra-set dependencies, and also dependencies between objects without specifying a specific path. For example, it is possible to express that any path between two objects should lead to the same value. But, as opposed to our model, they assume that every level of nesting presents a key or an object ID. Inference rules for the proposed forms of functional dependencies are presented, but they do not claim or prove their completeness.

3 Inference Rules for NFDs Without Empty Sets

One of the most interesting questions involving NFDs is that of logical implication, i.e., deciding if a new dependency holds given a set of existing dependencies. This problem can be addressed from two perspectives: One is to develop algorithms to decide logical implication, for example, tableau chase techniques (see [12] for the relational model, and more recently [16, 17] for a complex object model). The other is to develop inference rules that allow us to derive new dependencies from the given ones.

In the relational model, a simple set of three rules—called Armstrong’s Axioms—are sound and complete for functional dependencies (FDs). Presented using our notation, where “paths” are single attributes, they are:

- **reflexivity:**
  \[ \text{if } A \in X \text{ then } R : [X \rightarrow A]. \]

- **augmentation:**
  \[ \text{if } R : [X \rightarrow Z] \text{ then } R : [XY \rightarrow Z]. \]

- **transitivity:**
  \[ \text{if } R : [X \rightarrow Y], R : [Y \rightarrow Z] \text{ then } R : [X \rightarrow Z]. \]

The logical implication problem for these rules is formally defined as:

**Definition 3.1** Let \( SC \) be a schema, \( \Sigma \) be a set of FDs over \( SC \), and \( \sigma \) an FD over \( SC \). \( \Sigma \) logically implies \( \sigma \) under \( SC \), denoted \( \Sigma \vdash_{SC} \sigma \) if for all instances \( I \) of \( SC \), \( I \models \Sigma \text{ implies } I \models \sigma \).

The implication problem for NFDs that we will consider is slightly changed from that for FDs: no instances are allowed to contain empty sets. Empty sets cause tremendous difficulties in reasoning since formulas such as

\[ \forall x \in R.P(x) \]
are trivially true when \( R \) is empty. These problems are discussed in detail in Section 3.3. For completeness, we state below the implication problem that we are considering for NFDs.

The implication problem for NFDs that we are considering is therefore defined as:

**Definition 3.2** Let \( SC \) be a schema, \( \Sigma \) be a set of NFDs over \( SC \), and \( \sigma \) an NFD over \( SC \). \( \Sigma \) logically implies \( \sigma \) under \( SC \), denoted \( \Sigma \models_{SC} \sigma \) if for all instances \( I \) of \( SC \) with no empty sets, \( I \models \Sigma \) implies \( I \models \sigma \).

In this section, we present a sound and complete set of inference rules for NFDs in the restricted case in which no empty sets are present in any instance. The extension to allow empty sets in instances is discussed in detail in Section 3.3.

### 3.1 NFD Rules

Conceptually, the NFD rules can be broken up into three categories: The first three mirror Armstrong’s axioms – reflexivity, augmentation and transitivity. The next two – push-in and pull-out – transform between the alternate forms of NFDs discussed at the end of the last section. The last three rules allow inferences based solely on the nested form of the data – locality, singleton, and prefix.

In the following, \( x, y, z, x_0, x_1, \ldots \) are path expressions, and \( A_1, A_2, \ldots, B_1, B_2, \ldots \) are attribute labels, \( XY \) denotes \( X \cup Y \), where \( X, Y \) are sets of path expressions, and \( x : X \) denotes the set \( \{ x : x_1, \ldots, x : x_k \} \), where \( X = \{ x_1, \ldots, x_k \} \).

The NFD-rules are:

- **reflexivity:**
  if \( x \in X \) then \( x_0 : [X \rightarrow x] \).

- **augmentation:**
  if \( x_0 : [X \rightarrow z] \) then \( x_0 : [XY \rightarrow z] \).

- **transitivity:**
  if \( x_0 : [X \rightarrow x_1], \ldots, x_0 : [X \rightarrow x_n] \),
  \( x_0 : [x_1, \ldots, x_n \rightarrow y] \)
  then \( x_0 : [X \rightarrow y] \).

- **push-in:**
  if \( x_0 : y \rightarrow [X \rightarrow z] \) then \( x_0 : [y, y : X \rightarrow y \rightarrow z] \).

- **pull-out:**
  if \( x_0 : [y, y : X \rightarrow y \rightarrow z] \) then \( x_0 : y : [X \rightarrow z] \).

- **locality:**
  if \( x_0 : [A : X, B_1, \ldots, B_k \rightarrow A : z] \)
  then \( x_0 : A : [X \rightarrow z] \).

- **singleton:** if

---

5 A discussion of why we don’t adopt a simpler form of NFDs which would eliminate these two rules is deferred to Section 3.3.
1. \( x_0 : [x \to x : A_1], \ldots, x_0 : [x \to x : A_n] \)
2. type of \( x \) is \( \{A_1, \ldots, A_n\} \)
then \( x_0 : [x : A_1, \ldots, x : A_n \to x] \)

- prefix: if
  1. \( x_0 : [x_1 : A, x_2, \ldots, x_k \to y] \)
  2. \( x_1 \) has one or more labels
  3. \( x_1 \) is not prefix of \( y \)
then \( x_0 : [x_1, x_2, \ldots, x_k \to y] \)

As an example of the use of the NFD-rules, let \( R \) be a relation with schema \( \{A : \{B : \{C\}, E : \{F, G\}\}, D\} \), on which the following NFDs are defined:

(\text{nf1d}) \( R : [A : B : C, D \to A : E : F] \)
(\text{nf1d}) \( R : [A \to B : E : G] \)

We claim that \( R : A : [B \to E] \). The proof is as follows:

1. \( R : A : [B : C \to E : F] \) by locality of \text{nf1d}.

   The locality rule allows us to derive a local NFD from a global one by dismissing the attributes outside the level of nesting of the local NFD. In the example above, note that for any element in \( R \), given a value of \( A \) there exists a unique value of \( D \), since they are labels in a record type. Therefore, locally for any value of \( A, B : C \to E : F \).

2. \( R : A : [B \to E : F] \) by prefix rule on (1).

   (1) states that whenever two tuples in \( R \) have a common value for \( C \) in the set \( B \), then the value of \( E : F \) must also agree. In particular, if two tuples agree on the value of \( B \) then they present a common element, since we assumed that there are no empty sets in instances of \( R \).

3. \( R : A : E : [\emptyset \to F] \) by locality of (2).

   If in any tuple in \( R : A \) the value of \( B \) determines the value of \( E : F \), then all elements in \( E \) must agree on the value of \( F \), otherwise (2) would be violated. Therefore, locally in any \( A : E \) the value of \( F \) is constant.

4. \( R : A : [E \to E : F] \) by push-in.

   If the value of \( F \) is constant inside any value of \( A : E \), then for any given value of \( A : E \) there exists a unique value of \( F \). Therefore, the whole set determines the value of \( F \).

5. \( R : A : E : [\emptyset \to G] \) by locality of \text{nf1d}.

6. \( R : A : [E \to E : G] \) by push-in.

7. \( R : A : [E : F, E : G \to E] \) by singleton with (4) and (6).

   Since the value of the set \( E \) determines the value of each of its attributes, then \( E \) must be a singleton. Therefore, the values of its unique element determines the value of the set.

8. \( R : A : [B \to E] \) by transitivity with (7), (2), and \text{nf1d}.  

13
Lemma 3.1 (Soundness of NFD-rules) Let SC be a schema. The NFD-rules are sound for logical implication of NFDs under SC for the case when no empty sets are present in an instance.

Proof.

1. **reflexivity:** Suppose \( f \equiv x_0 : [X \to x] \) is not satisfied for some \( x \in X \). Let \( v_1, v_2 \) be two arbitrary values of \( x_0 : (I) \). If for some \( y \in X \), \( y(v_1) \neq y(v_2) \), then \( v_1, v_2 \) can not violate \( f \). If for all \( y \in X \) \( y(v_1) = y(v_2) \), and \( x(v_1) \neq x(v_2) \), \( v_1, v_2 \) violates \( f \). But \( x \in X \), therefore \( x(v_1) = x(v_2) \).

2. **augmentation:** Suppose \( I \) satisfies \( f_1 \equiv x_0 : [X \to z] \), but not \( f_2 \equiv x_0 : [XY \to z] \). Let \( v_1, v_2 \) be two arbitrary values of \( x_0 : (I) \). Suppose for all \( y \in Y \), \( y(v_1) = y(v_2) \), and for all \( x \in X \), \( x(v_1) = x(v_2) \), yet \( z(v_1) \neq z(v_2) \). But since \( f_1 \) is satisfied and for all \( x \in X \), \( x(v_1) = x(v_2) \), \( z(v_1) = z(v_2) \), a contradiction.

3. **transitivity:** Suppose \( I \) satisfies \( f_1 \equiv x_0 : [X \to x_1, \ldots, x_n \to y] \). Yet, \( I \) does not satisfy \( f \equiv x_0 : [X \to y] \). Let \( v_1, v_2 \) be two arbitrary values of \( x_0 : (I) \). Suppose \( p(v_1) = p(v_2) \) for all \( p \in X \). Since \( I \) satisfies \( f_i x_i(v_1) = x_i(v_2) \) for all \( i, 1 \leq i \leq n \). But \( I \) also satisfies \( f_y \), therefore \( y(v_1) = y(v_2) \), and \( I \) satisfies \( f \).

4. **push-in:** Suppose \( I \) satisfies \( f_1 \equiv x_0 : [y, X \to y : z] \), but not \( f_2 \equiv x_0 : [y, Y \times X \to y : z] \). Let \( v_1, v_2 \) be two arbitrary values of \( x_0 : (I) \), such that \( y(v_1) = y(v_2) \), and \( e_1 \) an element in \( y(v_1) \), and \( e_2 \) an element in \( y(v_2) \) such that for all \( x \in X \), \( x(e_1) = x(e_2) \), and \( z(e_1) \neq z(e_2) \). But \( \{ e_1, e_2 \} \subseteq y(v_1) \) and since \( I \) satisfies \( f_1, z(e_1) = z(e_2) \), a contradiction.

5. **pull-out:** Suppose \( I \) satisfies \( f_1 \equiv x_0 : [y, X \to y : z] \), but not \( f_2 \equiv x_0 : [y, X \to y : z] \). Let \( v_1 \) be an arbitrary value of \( x_0 : (I) \), and \( e_1, e_2 \) two elements in \( y(v_1) \) such that for all \( x \in X \), \( x(e_1) = x(e_2) \), and \( z(e_1) \neq z(e_2) \). But from \( f_1 \) if \( y(v_1) = y(v_1) \) and \( x(e_1) = x(e_2) \) for all \( x \in X \) then \( z(e_1) = z(e_2) \), a contradiction.

6. **locality:** Suppose \( I \) satisfies \( f_1 \equiv x_0 : [A \times x_1, \ldots, A \times x_m, B_1, \ldots, B_k \to A : x_n] \), but not \( f_2 \equiv x_0 : A : x_1, \ldots, m, x : x_n \). Let \( r \) be an arbitrary value of \( x_0 : (I) \), and \( v_1, v_2 \) arbitrary values of \( A : (r) \). Suppose \( x_i(v_1) = x_i(v_2) \) for all \( i, 1 \leq i \leq m \), yet \( x_n(v_1) \neq x_n(v_2) \). But \( x_i(v_1), x_i(v_2) \) are values of \( A : x_i(r) \), and since \( r \) is a record with labels \( A, B_1, \ldots, B_k \) there is only one value for all \( B_i, 1 \leq i \leq k \). Since \( I \) satisfies \( f_1, x_i(v_1) = x_n(v_2) \).

7. **singleton:** Note first that if the value of a set \( x \) is proven to be a singleton, then the unique element of the set determines the value of the set. In particular, if the element of the set is a record then the set of attributes of the record, \( \{ A_1, \ldots, A_n \} \), determines the value of the set, i.e., \( x_0 : [x : A_1, \ldots, A_n \to x] \). Suppose \( I \) satisfies \( f_1 \equiv x_0 : [x \to x : A_1, \ldots, f_n : x \to x : A_n] \). Yet, \( I \) does not satisfy \( f \equiv x_0 : [x \to x : A_1, \ldots, x : A_n \to x] \). We’ll show that under the assumptions \( x \) is a singleton. Suppose not. Let \( v_1 \) be arbitrary value of \( x_0 : (I) \), and let \( e_1, e_2 \) be two elements in \( v_1 \). There must exist some \( A_i \) such that \( A_i(e_1) \neq A_i(e_2) \). But \( I \) satisfies \( x_0 : [x \to x : A_i] \), and therefore for all \( A_i, A_i(e_1) = A_i(e_2) \), and as a consequence \( x \) is a singleton.

8. **prefix:** Suppose \( I \) satisfies \( f_1 \equiv x_0 : [x_1 : A, x_2, \ldots, x_k \to y], |x_1| \geq 1 \), but \( I \) does not satisfy \( f_2 \equiv x_0 : [x_1, x_2, \ldots, x_k \to y] \). Let \( v_1, v_2 \) be two arbitrary value of \( x_0 : (I) \). Suppose for all \( x_i, x_i(v_1) = x_i(v_2) \), but \( y(v_1) \neq y(v_2) \). \( x_1(v_1) = x_1(v_2) \) by assumption. Then for every element \( e_1 \in x_1(v_1) \) there exists an element \( e_2 \in x_1(v_2) \) such that \( x_1 : A(v_1) = x_1 : A(v_2) \). The value of \( y(v_1), y(v_2) \) does not depend on the elements \( e_1, e_2 \) chosen because \( x_1 \) is not prefix of \( y \) by assumption. Therefore, \( y(v_1) = y(v_2) \), which contradicts our initial assumption. Hence, \( x_0 : [x_1 \ldots x_k \to y] \).
There are several rules that are consequences of the rules defined above. Here we give just one that will be useful in later discussions.

- **full-locality:**
  - if
    1. $x_0 : [x : X, Y \rightarrow x : z]$
    2. $x$ is not a proper prefix of any $y \in Y$
  then $x_0 : [x, x : X \rightarrow x : z].$

**Proof:** Let $x = A_1 : \ldots : A_k$, i.e., we can rewrite the NFD as $x_0 : [Y, A_1 : \ldots : A_k : X \rightarrow A_1 : \ldots : A_k : z]$. Apply the prefix rule multiple times on paths in $Y$. We get $x_0 : [B_1, \ldots, B_m, A_1 : Y_1, \ldots, A_1 : \ldots : A_{k-1} : Y_{k-1}, A_1 : \ldots : A_k : X \rightarrow A_1 : \ldots : A_k : z]$, where for all $y \in Y_i$, $1 \leq i < k$, $|y| = 1$, and for all $p \in \{B_1, \ldots, B_m\} \cup A_1 : Y_1 \cup \ldots \cup A_1 : \ldots : A_{k-1} : Y_{k-1}$ there exists a $q \in Y$ such that $q = pq'$. We can then apply the locality rule and get $x_0 : A_1 : [Y_1, \ldots, A_2 : \ldots : A_{k-1} : Y_{k-1}, A_2 : \ldots : A_k : X \rightarrow A_1 : \ldots : A_k : z]$. Applying locality rule $k - 1$ more times we get $x_0 : A_1 : \ldots : A_k : [X \rightarrow z]$. Then by push-in $x_0 : [x, x : X \rightarrow x : z].$ □

3.2 **Completeness of the NFD-rules**

In order to prove completeness, we need to define the set of paths in a schema, and the closure of a set of paths.

**Definition 3.3** Let $SC = (R, S)$ be a schema. Then the paths of $SC$, denoted as Paths($SC$), is the set of all path expressions $p \equiv Rp'$, such that $R \in R$, and $p'$ is well-typed with respect to $\tau^R$. Similarly, the paths of $R$, $R \in R$, denoted as Paths$_{SC}(R)$, is the set of paths $p$ such that $p \in$ Paths($SC$), and $p \equiv Rp'$.

**Definition 3.4** Let $SC$ be a schema, $\Sigma$ a set of NFDs over $SC$, $x_0$ a path expression, and $X = \{x_1, \ldots, x_n\}$, such that $\{x_0, x_0 : x_1, \ldots, x_0 : x_n\} \subseteq$ Paths($SC$). The **closure** of $X$ under $x_0$, and $\Sigma$, denoted $(x_0, X, \Sigma)^{*,SC}$ (or $(x_0, X)^*$ when $\Sigma, SC$ are understood) is the set of paths $x_0 : q \in$ Paths($SC$), and $x_0 : [X \rightarrow q]$ can be derived from the NFD-rules.

Let $SC = (R, S)$ be a schema, $\Sigma$ a set of NFDs over $SC$, $X$ a set of paths such that $X \subseteq$ Paths($R$), $R \in R$, and $x_0$ a path in Paths($R$). The completeness proof is based on the construction of an instance $I$ of $R$ such that $I = \Sigma$, but $I \not\models x_0 : [X \rightarrow x]$ if $x \not\in (x_0, X, \Sigma)^{*,SC}$. In the following we describe the construction of $I$.

We assume that the domain of all base types are infinite, and to make the exposition simpler, we consider a unique base type $b$ in our data model.

**Construction of $I$:** Let closure be $(x_0, X, \Sigma)^{*,SC}$, where $x_0 \equivRx_0$. value($p$) are global variables. If $p$ is a set of records and in its construction value($p'$) is used (this happens when $p$ is prefix of $p'$) then value($p'$) should be thought as a placeholder until its value is evaluated.
val := newValue();
for all \( p \in \text{closure} \)
    value(\( p \)) := assignVal(val, \( p \));
I := assignX_0(R);

The auxiliary functions are defined as:

new Value(): returns a fresh new value in the domain of \( b \).

assignX_0(\( p \)): it is a function that starts the construction of instance \( I \) by assigning new fresh values to every path that is not a prefix of \( x_0 \). \( r \) is a local variable of type \(<A_1, \ldots, A_n>\), where type of \( p \) is \(<A_1, \ldots, A_n>\).

If \( p = x_0 \) then return assignVal(0, \( x_0 \));
for each \( A_i, 1 \leq i \leq n \)
    if \( p : A_i \) is prefix of \( x_0 \) then
        \( r.A_i := \text{assignX}_0(p : A_i) \);
    else
        \( r.A_i := \text{assignNew}(p : A_i) \);
return \{r\};

assignVal (val, \( p \)): it is a function that gives a value \( val \) for a path \( p \) depending on the type of \( p \) in a schema \( SC \). \( r_1 \), and \( r_2 \) are local variables of type \( t \), where the type of \( p \) is \{\( t \}\).

if \text{type}_{SC}(p) = b \) then return val;
if \text{type}_{SC}(p) = \{b\} \) then return \{val\};
if \text{type}_{SC}(p) = \{<A_1, \ldots, A_n>\} \) then
    for all \( A_i, 1 \leq i \leq n \)
        if \( p : A_i \in \text{closure} \) then
            \( r_1.A_i := \text{value}(p : A_i) \);
            \( r_2.A_i := \text{value}(p : A_i) \);
        else
            \( r_1.A_i := \text{assignNew}(p : A_i) \);
            \( r_2.A_i := \text{assignNew}(p : A_i) \);
    return \{r_1, r_2\};

assignNew (\( p \)): it is a function that gives a new fresh value for a path \( p \), \( p \notin \text{closure} \), depending on the type of \( p \) in a schema \( SC \). If type of \( p \) is \{\( t \}\), then \( r \) is a local variable of type \( t \).

if \text{type}_{SC}(p) = b \) then return newValue();
if \text{type}_{SC}(p) = \{b\} \) then return \{newValue()\};
if \text{type}_{SC}(p) = \{<A_1, \ldots, A_n>\} \) then
    for all \( A_i, 1 \leq i \leq n \)
        if \( p : A_i \in \text{closure} \) then
            \( r.A_i := \text{value}(p : A_i) \)
        else \( r.A_i := \text{assignNew}(p : A_i) \);
    if \{p : A_1, \ldots, p : A_n\} \subseteq \text{closure} \) then
        return \{r, newRow(p, (p, \emptyset)^* )\}
    else
        return \{r\}
**newRow(p, sameVal):** The type of \( p \) is \( \{A_1, \ldots, A_n\} \), where \( p \notin \text{closure} \), and for all \( A_i, 1 \leq i \leq n \), \( p : A_i \in \text{closure} \). This function returns a record, where the value of \( A_i, 1 \leq i \leq n \) is set to \( \text{value}(p : A_i) \) if \( p : A_i \in \text{sameVal} \), otherwise \( A_i \) is given a new fresh value. \( r \) is a local variable of type \( \{A_1, \ldots, A_n\} \) for all label \( A_i, 1 \leq i \leq n \).

- \( \text{if } p : A_i \in \text{sameVal} \text{ then} \)
  \( r.A_i := \text{value}(p : A_i) \)
- \( \text{else} \)
  \( \text{if } \text{type}_{SC}(p : A_i) = b \text{ then} \)
  \( r.A_i := \text{newValue}() \)
  \( \text{if } \text{type}_{SC}(p : A_i) = \{b\} \text{ then} \)
  \( r.A_i := \{\text{newValue}()\} \)
  \( \text{if } \text{type}_{SC}(p : A_i) = \{B_1, \ldots, B_k\} \text{ then} \)
  \( r.A_i := \{\text{newRow}(p : A_i, \text{sameVal})\} \)

return \( r \);

To illustrate the algorithm described consider the following examples.

**Example 3.1** Let \( R \) be a relation with schema \( \{A, B : \{C\}, D, E : \{F, G\}, H : \{J, L\}, I, M : \{N, O\}\} \). The set \( \Sigma \) of NFDs defined for \( R \) are:

\[
\begin{align*}
R &\colon [A \rightarrow B : C] \\
R &\colon [B : C \rightarrow D] \\
R &\colon [D \rightarrow E : F] \\
R &\colon [A \rightarrow E : G] \\
R &\colon [B : C \rightarrow H] \\
R &\colon [I \rightarrow H : J]
\end{align*}
\]

Then, \( (R, \{B\}, \Sigma)^* = \{R : B, R : B : C, R : D, R : E : F, R : H, R : H : J\} \). The following instance is constructed using the algorithm presented.

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>E</td>
<td>H</td>
<td>I</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Example 3.2** Let \( R \) be a relation with schema \( \{A : \{B : \{C, D, E : \{F, G\}\}\}, H\} \). The set \( \Sigma \) of NFDs defined for \( R \) are:

\[
\begin{align*}
R &\colon [A : B : C \rightarrow A : B] \\
R &\colon [A : B : C \rightarrow A : B : E : F] \\
R &\colon [H \rightarrow A : B : D]
\end{align*}
\]

Then, \( (R, \{A : B : C\}, \Sigma)^* = \{R : A : B : C, R : A : B, R : A : B : D, R : A : B : E : F\} \). The following instance is constructed using the algorithm presented.
In order to simplify the completeness proof, we first make a number of observations about the instance constructed as a result of the algorithm described above, as well as consequences of the NFD-rules.

**Observation 3.1** Let \( p, p : q \) be paths such that \( p \not\in \text{closure} \), type of \( p \) is \( \langle A_1, \ldots, A_k \rangle \), and for all \( A_i, 1 \leq i \leq k, p : A_i \in \text{closure} \). If \( p : q \in (p, q)^* \) then \( p : q \in \text{closure} \).

**Proof:** Let \( p \equiv x_0 : p' \). If \( |q| = 1 \) then it is direct because by assumption for all \( A_i, x_0 : [X \to p' : A_i] \). Suppose \( |q| > 1 \), i.e., \( q = A_1^q : q' \), where, \( |q'| \geq 1 \). By assumption, \( x_0 : p'[\emptyset \to q] \). Then by push-in \( x_0[p' \to p' : A_1^q : q'] \). By full-locality, \( x_0[p' : A_1 \to p' : A_1^q : q'] \), and then by transitivity, \( x_0[X \to p' : A_1^q : q'] \). Therefore, \( p : q \in \text{closure} \).

**Observation 3.2** Let \( x_0 : p \) be a path. If \( x_0 : p \in \text{closure} \) then any \( x_0 : p(I) \) is constructed either by assignVal or newRow. If \( x_0 : p \not\in \text{closure} \) then any \( x_0 : p(I) \) is constructed either by assignNew or newRow.

**Proof:** By construction.

**Observation 3.3** Let \( p \) be a set-valued path, and \( v \) a value of \( p(I) \). If \( v \) was built by newRow\((p, (p', \emptyset)^*)\) then \( v \) is a singleton.

**Proof:** By construction.

**Observation 3.4** Let \( p \) be a path of type \( \langle A_1, \ldots, A_k \rangle \), and \( v \) a value of \( p(I) \). If \( v \) has more than one element then either
1. $p = x_0$ or $p \in \text{closure}$ and there exists an $A_i$, $1 \leq i \leq k$, such that $p : A_i \notin \text{closure}$, or

2. $p \notin \text{closure}$ and for all $A_i$, $1 \leq i \leq k$, $p : A_i \in \text{closure}$, and there exists an $A_j$, $1 \leq i \leq k$, such that $p : A_j \notin (p, \emptyset)^*$. 

**Proof:** The function $\text{newRow}$ always builds singletons. Therefore, if $v$ has more than one element, $v$ was built by $\text{assignVal}$ or $\text{assignNew}$. $\text{assignVal}$ builds the value of $x_0$, and every path $q \notin \text{closure}$. Suppose for all $A_i$, $1 \leq i \leq k$, $p : A_i \in \text{closure}$. Then, by construction, rows $r_1, r_2$ are identical, and the value resulting from the function is a singleton. Therefore, there exists an $A_i$, such that $p : A_i \notin \text{closure}$. $\text{assignNew}$ builds the value of path $q \notin \text{closure}$, and it only results in a set with more than one element if $\{p : A_1 \ldots p : A_k\} \subseteq \text{closure}$. Let $p \equiv x_0 : p'$. Suppose for all $A_i$, $1 \leq i \leq k$, $A_i \in (p, \emptyset)^*$. Then for all $A_i$, by push-in rule $x_0 : [p \rightarrow p : A_i]$, and then by singleton rule $x_0[p : A_1, \ldots, p : A_k \rightarrow p]$, and $p \in \text{closure}$ by transitivity. This contradicts our assumption that $p \notin \text{closure}$, and therefore, there exists at least an $A_i$, such that $p : A_i \notin (p, \emptyset)^*$.

**Observation 3.5** Let $f \equiv x_0 : [X \rightarrow y]$ be an NFD and $w$ the largest common prefix between $y$, and any path $x \in X$, i.e., $y \equiv wy'$, and $x \equiv wx'$. If $f$ is an upwards or sideways NFD, given a value $v$ of $x_0 : w(I)$, there exists a unique value of $y(f(v))$.

**Proof:** If $y' \equiv \epsilon$, it is trivial. By definition of upwards and sideways NFDs, $y \equiv A_1 : \ldots : A_{k-1} : A_k$, where $w \equiv A_1 : \ldots : A_{k-1} : A_i : A_k$. Therefore the type of $w$ is a record, and given a value of a record, there is only one value for a given attribute $A_k$.

**Observation 3.6** Let $p, p : q$ be paths such that type of $p : q$ is $\{<B_1, \ldots, B_n>\}$. If $\{p : q : B_1, \ldots, p : q : B_n\} \subseteq (p, \emptyset)^*$ then $p : q \in (p, \emptyset)^*$.

**Proof:** Let $p \equiv x_0 : p'$. By assumption, for all $B_i$, $1 \leq i \leq n$, $x_0 : [p' \rightarrow p' : q : B_i]$ then by full-locality $x_0 : [p' \rightarrow p' : q : B_i]$. Then by singleton, $x_0 : [p' : q : B_1, \ldots, p' : q : B_n \rightarrow p' : q]$. By transitivity, $x_0 : [p' \rightarrow p' : q]$, and by pull-out $x_0 : p'[0] \rightarrow q$.

**Observation 3.7** Let $p, p'$ be paths. If there exists a value $v$ of $p(I)$ built by $\text{newRow}(p, (p', \emptyset)^*)$ then $p'$ is a prefix of $p$ ($p \equiv p' : q$), for all $p'$ prefix of $q$, $p' : q' \notin (p', \emptyset)^*$, $v$ is part of an element resulting from $\text{assignNew}(p')$, $p' \notin \text{closure}$, type of $p'$ is $\{<A_1, \ldots, A_k>\}$, and for all $A_i$, $1 \leq i \leq k$, $p' : A_i \in \text{closure}$.

**Proof:** By construction, if $v$ was built by $\text{newRow}(p, (p', \emptyset)^*)$, this value is inside a value built by $\text{newRow}(p', (p', \emptyset)^*)$, which is an element of a value built by $\text{assignNew}(p')$, where $p' \notin \text{closure}$, type of $p'$ is $\{<A_1, \ldots, A_k>\}$, and for all $A_i$, $p' : A_i \in \text{closure}$. If in the construction of $\text{newRow}(p', (p', \emptyset)^*)$, there exists a prefix $q'$ of $q$ such that $p' : q' \in (p', \emptyset)^*$, then the value of $p' : q'$ is set to $\text{value}(p' : q')$, and therefore the value of $p$ in $v$ could not be constructed by $\text{newRow}(p, (p', \emptyset)^*)$.

**Observation 3.8** Let $p, pq$ be paths. If $v$ is a value of $pq(I)$ built by $\text{newRow}(pq, (p, \emptyset)^*)$ then $v$ is distinct from any other value in $I$.

**Proof:** By Observation 3.7 if $v$ was built by $\text{newRow}(pq, (p, \emptyset)^*)$ then for all prefix $q'$ of $q$, $pq' \notin (p, \emptyset)^*$. We’ll show that $v$ is distinct by induction on the structure of $pq$. 

19
Base Case: If type of $pq$ is a base type or a set of base types and $pq \not\in (p, \emptyset)^*$, then the value returned by $\text{newRow}$ is given by $\text{newValue}(\cdot)$, which is a value distinct from any other in $I$.

Inductive Step: Let type of $p : q$ be $\{B_1, \ldots, B_n\}$. By Observation 3.6, if $p : q \not\in (p, \emptyset)^*$ then there exists at least one $B_i$, $1 \leq i \leq n$, such that $p : q : B_i \not\in (p, \emptyset)^*$. By inductive hypothesis the values of $p : q : B_i$ are distinct and therefore the value of $p : q$ is distinct. □

**Observation 3.9** Let $p, pq$ be paths, and $v$ a value returned by $\text{newRow}(p, (q', \emptyset)^*)$. If $q(v)$ is not a value distinct from any other value in $I$ then there exists a prefix $q'$ of $q$ such that $p : q' \in (q', \emptyset)^*$.

**Proof:** It is a direct consequence of Observations 3.8 and 3.7. □

**Observation 3.10** Let $p, pq$ be paths, and $v$ a value resulting from function $\text{newRow}(p, (q', \emptyset)^*)$, where $q'$ is a prefix of $p$, and $q \equiv B_1 : \ldots : B_k$. If for all $B_i$, $1 \leq i \leq k$, $p : B_1 : \ldots : B_i \not\in (p, \emptyset)^*$ then there exists a single value of $q(v)$.

**Proof:** By construction. □

**Observation 3.11** Let $p, pq$ be paths, and $v$ a value resulting from function $\text{assignNew}(p)$, where $q \equiv B_1 : \ldots : B_k$. If for all $B_i$, $1 \leq i \leq k$, $p : B_1 : \ldots : B_i \not\in \text{closure}$ then there exists a single value of $q(v)$.

**Proof:** By construction. □

**Observation 3.12** Let $p$ be a path of type $\{A_1, \ldots, A_k\}$, $pq$ a path not in $\text{closure}$, and $v$ a value of $p : (I)$. If for all prefix $q'$ of $q$, $p : q' \not\in \text{closure}$ then for all $q'$ there exists a unique value of $q'(v)$.

**Proof:** By Observation 3.2, the value of any path not in $\text{closure}$ is given either by function $\text{assignNew}$, or $\text{newRow}$. Suppose $q \equiv A_1 : z$. If $A_i(v)$ was built by $\text{assignNew}(p)$, then by Observation 3.11, for every prefix $q'$ of $q$ there exists a single value of $q'(v)$. If $A_i(v)$ was built by $\text{newRow}(p, (q', \emptyset)^*)$, then by Observation 3.7, $q' \not\in \text{closure}$, type of $q'$ is $\{A_1, \ldots, A_k\}$, and for all $A_i$, $1 \leq i \leq k$, $p' : A_i \not\in \text{closure}$. If for all prefix $q'$ of $p : q' \not\in \text{closure}$ then by Observation 3.1, $p : q' \not\in (q', \emptyset)^*$. Then, by Observation 3.10 there exists a single value of $q'(v)$. □

**Observation 3.13** Let $p, pq$ be paths, and $v$ a value resulting from function $\text{assignVal}(l, p)$, for some value $l$, where $q \equiv B_1 : \ldots : B_k$. If for all $B_i$, $1 \leq i \leq k$, $p : B_1 : \ldots : B_i \in \text{closure}$ then there exists a single value of $q(v)$ which is $\text{value}(pq)$.

**Proof:** By construction. □

**Observation 3.14** If $p$ is a path and $x_0$ is not a proper prefix of $p$ then there is a unique value of $p(I)$.

**Proof:** If $p = x_0$ there is a single value of $p(I)$ given by $\text{assignVal}(0, p)$ (note that there is a single value of $x_0(I)$, but $x_0 : (I)$ can have one or two different values). If $p \neq x_0$ then $p$ is either a prefix of $x_0$, or $p \equiv x_0' A_1 : \ldots : A_k$, where $x_0'$ is a proper prefix of $x_0$. The value of any prefix of $x_0$ is given by $\text{assignX}_0$ which always returns a singleton, and therefore have a unique value. The value of $p' A_1$ is given by $\text{assignNew}$. Since for all $p' A_1 : \ldots : A_i$, $1 \leq i \leq k$, $p' A_1 : \ldots : A_i \not\in \text{closure}$, since $x_0$ is not a prefix, they have a unique value by Observation 3.11. □
Observation 3.15 Let \( p \) be a path in closure, and \( v \) a value of \( p(I) \). If \( v \neq value(p) \) then its value is given by function \( newRow(p, (p', \emptyset)^* \)\), where \( p \equiv p' : q \) and for all prefix \( q' \) of \( q \), \( p' : q' \not\in (p', \emptyset)^* \).

Proof: By construction. Both functions assignVal, and assignNew assign value(p) when \( p \in closure \). Therefore, if \( v \neq value(p) \) it had to be assigned by newRow. The other consequences follow from Observation 3.7.

Observation 3.16 Let \( p, pq \) be paths, and \( v \) a value constructed by assignNew(p), such that \( p \not\in closure \), type of \( p \) is \( \{ A_1, \ldots, A_k \} \), and for all \( A_i, 1 \leq i \leq k; p : A_i \in closure \). If for all prefix \( q' \) of \( q \), \( pq' \not\in (p, \emptyset)^* \), then there exists at least two values of \( q(v) \).

Proof: By construction. If for all \( A_i, p : A_i \in closure \), \( v = \{ e_1, e_2 \} \), where for all \( A_i, A_i(e_1) = value(p : A_i) \), and \( e_2 \) is constructed by newRow(p, (p, \emptyset)^*). Since for all prefix \( q' \) of \( q \), \( pq' \not\in (p, \emptyset)^* \), the value of \( pq \) is given by newRow(pq, (p, \emptyset)^*), and by Observation 3.8 this is a value distinct from any other value in \( I \). Therefore, \( q(v) \) has at least two values, one inside value(p : A_i), and the other built by newRow.

Observation 3.17 Let \( p, pq \) be paths, such that \( pq \in closure \), and \( v \) a value of \( p(I) \). There exists at least two distinct values of \( q(v) \) if and only if there exists a prefix \( q' \) of \( q \) such that a value of \( q(v) \) is built by newRow(pq, (p, \emptyset)^*).

Proof:

(\(-\)) Let \( q \equiv q'q'' \). If there exists a value built by newRow(pq, (p, \emptyset)^*), then by Observation 3.7, \( p(\emptyset)^* \) was built by assignNew(pq), \( pq' \not\in closure \), type of \( pq' \) is \( \{ A_1, \ldots, A_k \} \), for all \( A_i, q' : A_i \in closure \), and for all prefix \( r \) of \( q'' \) \( pq' \not\in (p, \emptyset)^* \). Then by Observation 3.16 there exists at least two values of \( q'' \) in \( q'(v) \), and therefore, in \( q(v) \).

(\(\rightarrow\)) By Observation 3.15, if there exists two different values of \( pq(1) \), and \( pq \in closure \), then at least one of them was built by function newRow. Suppose \( pq \equiv p'w \), where \( p' \) is the largest prefix of \( pq \) such that \( q(v) \) was built by newRow(pq, (p', \emptyset)^*). By Observation 3.7, the value \( v' \) of \( p'(I) \) was built by assignNew(p'). Then, by 3.16 there exists at least two values of \( w(v') \). So, we only have to show that \( |p'| \geq |p| \). Suppose not. By construction, \( v' = \{ e_1, e_2 \} \), where for all \( A_i, A_i(e_1) = value(p' : A_i) \), and \( e_2 \) is the value returned by newRow(p', (p', \emptyset)^*). If \( |p'| < |p| \), then the value of \( v \) is either part of value(p' : A_i), or a value built by newRow(p', (p', \emptyset)^*). Suppose \( v \) is in value(p' : A_i). By construction, value(p' : A_i) cannot contain a value built by newRow(pq, (p', \emptyset)^*), where \( |p'| < |p' : A_i| \), which contradicts our assumption that \( p' \) was the largest prefix of \( pq \), such that \( q(v) \) was built by newRow(pq, (p', \emptyset)^*). Now suppose that \( v \) was built by newRow(pq, (p', \emptyset)^*). If \( q(v) \) was built by newRow(pq, (p', \emptyset)^*), then by Observation 3.7 for all prefix \( u' \) of \( u \), \( p'u' \not\in (p', \emptyset)^* \). By Observation 3.10 there exists a unique value of \( q(v) \), which contradicts our assumption that there exists at least two distinct values of \( q(v) \). Therefore, \( |p'| \geq |p| \).

Observation 3.18 Let \( p \) be a path such that \( p \not\in closure \). Then function assignNew(p) never returns the same value for \( p \) in the construction of \( I \).

Proof: By induction on the structure of \( p \).

Base Case: If \( p \) is a base type of a set of base types then the value is given by new Val u e(), which always
returns a fresh new value. Since by assumption the domain of the base types is infinite, we're done.

**Inductive Step:** Let type of $p$ be $\langle A_1, \ldots, A_k \rangle$. If there exists an $A_i$, $1 \leq i \leq k$, such that $p : A_i \notin \text{closure}$, then the value returned by `assignNew` is a singleton in which at least one attribute is built by `assignNew`. By inductive hypothesis, these are all distinct values. Therefore, the value of $p$ is also distinct as a set. If $\{ p : A_1, \ldots, p : A_k \} \subseteq \text{closure}$ then `assignNew` returns a set of two elements, where one of the them is built by `newRow(p, (p, \emptyset)*)`. By Observation 3.8 these values are always distinct, and therefore the value of $p$ is also distinct as a set.

**Observation 3.19** Let $p, p : q$ be paths, $v$ a value of $p(I)$, and $v_1, v_2$ two elements in $v$ such that $v_1 \neq v_2$. If $q(v_1) = q(v_2)$ then there exists a prefix $q'$ of $q$ such that $p : q' \in \text{closure}$.

**Proof:** Suppose, on the contrary, that for all prefix $q''$ of $q$, $p : q'' \notin \text{closure}$. By Observation 3.4, if $v$ has at least two distinct elements and for all prefix $q''$ of $q$, $q'' \notin \text{closure}$, either $p \equiv v_0$, or $p \notin \text{closure}$. $x_0$ is always built by `assignVal` by construction, and by Observation 3.2 $v$ was built either by `newRow`, or `assignVal`. If $v$ was built by `newRow`, then by Observation 3.10 for all prefix $q'$ of $q$ there exists a unique value of $q'(v)$, a contradiction. Therefore, $p$ was built by `assignVal`, and by construction for all prefix $q''$ of $q$, $q''(v)$ was built by `assignNew`. But by Observation 3.18 the values returned by these functions are always distinct, which contradicts our assumption that $q(v_1) = q(v_2)$. Therefore, there exists a prefix $q'$ of $q$ such that $p : q' \in \text{closure}$.

**Observation 3.20** Let $p$ be a path, and $v$ a value of $p(I)$. If $v$ was built by `assignVal`, then there exists no path $pq$ such that $pq$ was built by `newRow(pq, (z, \emptyset)*)$, where $|z| < |p|$.

**Proof:** Suppose there exists a path $pq$ built by `newRow(pq, (z, \emptyset)*)$, where $|z| < |p|$. By Observation 3.7, $z \notin \text{closure}$, $z$ is a prefix of $pq$ ($pq \equiv zq'$), and for all prefix $z''$ of $q'$, $z'' \notin (z, \emptyset)^*$. But since $|z| < |p|$, $p \equiv zp'$ where $p'$ is a prefix of $z'$. If $v$ was built by `assignVal` then by construction $p \in (z, \emptyset)^*$, a contradiction.

**Observation 3.21** Let $p, pq$ be paths. If $v$ is a value of $p(I)$ built by `assignVal` and $pq \in \text{closure}$ then for every element $e \in v$ there exists a value of $q(e) = \text{value}(pq)$.

**Proof:** We will show that if $v$ is a value of $p(I)$ given by `assignVal` and there exists a value of a path $pp' \in \text{closure}$ given by `newRow` then there exists also a value of $q'(v)$ given by `assignVal = value(pp')`. Let $q'$ be the result of function `newRow(pp', (z, \emptyset)*)`. By Observation 3.7, $q'$ is part of an element resulting from `assignNew(z)`, type of $z$ is $\langle A_1, \ldots, A_k \rangle$, and for all $A_i$, $1 \leq i \leq k$, $z : A_i \in \text{closure}$. By construction, in the other element resulting from `assignNew(z)`, the value of $z : A_i$ is `value(z : A_i)`, for all $A_i$, which are built by `assignVal`. By Observation 3.20, $|z| > |p|$ ($z \equiv pz'$). So, there exists a value of $z' : A_i(v) = \text{value}(z' : A_i)$. Since for some $A_i$, $z' : A_i$ is a prefix of $p'$, and functions `assignVal`, and `assignNew` always assigns `value(pq)` when $pq \in \text{closure}$, there exists also a value of $q'(v)$ given by `assignVal = value(pp')`. By construction, both elements in $p(I)$ are built in the same way by `assignVal`, and therefore this is true for every element in $v$.

**Observation 3.22** Let $p, pq$ be paths and $v$ a value of $p(I)$. If $p$ is the largest prefix of $pq$ built by `assignVal` and $pq \notin \text{closure}$, and there exists a value of $q'(v)$ built by `assignNew`. 22
Proof: First, we will show that for all prefix \( q' \) of \( q \), \( pq' \not\in \text{closure} \). Suppose there exists a prefix \( q' \) of \( q \) in \text{closure}. By Observation 3.2, if \( pq' \in \text{closure} \), \( pq'(v) \) was built by either \text{assignVal} or \text{newRow}. \( pq' \) cannot be built by \text{assignVal} because this contradicts our assumption that \( p \) is the largest prefix of \( pq \) built by this function. Therefore, it was built by \text{newRow}(pq', (z, \emptyset)^*) \). By Observation 3.20, \( |z| > |p| \), and by Observation 3.7 \( z \not\in \text{closure} \), \( z \) is the result of \text{assignNew}, type of \( z \) is \( \langle A_1, \ldots, A_k \rangle \), and for all \( A_i \), \( 1 \leq i \leq k \), \( z : A_i \in \text{closure} \). If the value of \( z \) is the result of \text{assignNew}, then by construction there exists a value of \( z : A_i \) prefix of \( pq \) built by \text{assignVal}, which also contradicts that \( p \) is the largest prefix of \( pq \) built by \text{assignVal}. Therefore, for all \( q' \) prefix of \( q \), \( pq' \in \text{closure} \).

Now, we will show that for all \( q' \) prefix of \( q \), there exists a value of \( pq'(v) \) built by \text{assignNew}. Since for all \( q' \), \( pq' \not\in \text{closure} \), by Observation 3.2, \( pq' \) was built either by \text{assignNew} or \text{newRow}. So we have to show that there exists no \( pq' \) built by \text{newRow}(pq', (z, \emptyset)^*) \). Suppose there exists one. By Observation 3.20, since \( v \) was built by \text{assignVal}, \( |z| > |p| \). By Observation 3.7, \( z \) is the result of \text{assignNew}, type of \( z \) is \( \langle A_1, \ldots, A_k \rangle \), and for all \( A_i \), \( 1 \leq i \leq k \), \( z : A_i \in \text{closure} \). By construction of \text{assignNew}, there exists a value of \( z : A_i \) built by \text{assignVal}. Since for some \( A_i \), \( z : A_i \equiv px' \) and \( px' \) is a prefix of \( pq' \), it contradicts our assumption that \( p \) is the largest prefix of \( pq \) built by \text{assignVal}. Therefore, for all \( q' \), \( pq' \) was built by \text{assignNew}.

Now, we're ready to prove the completeness of the inference rules.

**Lemma 3.2 (Completeness of the NFD-rules)** The NFD-rules are complete for all instances that contain no empty sets.

**Proof:** From the definition of closure, \( x_0 : [X \rightarrow y] \) follows from a given set of NFDs \( \Sigma \) using the NFD-rules if and only if \( x_0 : y \in (x_0, X, \Sigma)^{\ast,SC} \).

We have to show that considering the instance \( I \) constructed as described:

1. \( I \models \Sigma \)
2. \( I \not\models x_0 : [X \rightarrow y] \) if \( x_0 : y \not\in (x_0, X, \Sigma)^{\ast,SC} \).

1) \( I \models \Sigma \)

We will show that for any \( f \equiv u_0 : [U \rightarrow z] \in \Sigma, I \models f \). Suppose on the contrary, that \( I \not\models f \).

If \( x_0 \) is not prefix of \( u_0 : z \) then by Observation 3.14, there exists a single value for \( u_0 : z(I) \) and therefore \( I \) cannot violate \( f \). Therefore, \( x_0 \) is a prefix of \( u_0 : z \).

Suppose \( |u_0| < |x_0| \), i.e., \( x_0 = u_0 : u'_0 \). Let \( B_1u_1, \ldots, B_au_i \) be the paths in \( U \) that do not have \( u'_0 \) as prefix, and \( u_{i+1}, \ldots, u_k \) be the paths in \( U \) that have \( u'_0 \) as prefix, i.e., for all \( u_i \), \( 1 \leq i \leq k \), \( u_i = u'_0 : u'_i \). Applying the prefix rule multiple times we have \( u_0 : [u'_0 : u'_{i+1}, \ldots, u'_k B_1 \ldots B_l \rightarrow u'_0 : z'] \). Applying locality, and pull-out, we get \( f' \equiv u_0 : u'_0 : [u'_{i+1}, \ldots, u'_k \rightarrow z'] \).

For every \( B_j u_j \), \( 1 \leq j \leq l \), there exists a unique value in \( I \) by Observation 3.14. Therefore, if \( I \models f' \), then \( I \models f \).

So, we can assume that \( |u_0| \geq |x_0| \), i.e. \( u_0 \equiv x_0 : u'_0 \). Let \( w \) be the largest common prefix between \( z \) and any \( u \in U \). We will use induction on \( |w| \).

**Base Case:** \( |w| = 0 \)

**Case 1:** \( |z| = 1 \)
By definition, \( f \) is either an upward or sideways NFD. Let \( U = \{ u_1, \ldots, u_n \} \), \( v \) and arbitrary value of \( u_0(I) \), and \( v_1, v_2 \) two arbitrary elements in \( v \) such that for all \( u \in U \), \( u(v_1) = u(v_2) \). By Observation 3.5, if \( v_1 \), and \( v_2 \) traverse exactly the same path, there exists a unique value of \( z(v_1) = z(v_2) \) if \( f \) is a upwards or sideways NFD. Therefore, \( I \) cannot violate \( f \).

So, there exists a prefix \( p \) of \( u \in U \) such that \( p(v_1) \neq p(v_2) \), i.e., \( v_1 \) and \( v_2 \) do not follow identical paths. By Observation 3.19, for all \( u_i \in U \) there exists a prefix \( u_i' \) of \( u_i \) such that \( u_i' \in \text{closure} \). Since there is no common prefix between any \( U \) and \( z \), we can apply the prefix rule multiple times and get \( u_0 : [u_1', \ldots, u_n' \rightarrow z] \).

If \( z(v_1) \neq z(v_2) \) (and \( |z| = 1 \)), then \( u_0 \) must have at least two elements. By Observation 3.2 and 3.4, either \( u_0 \) was built by \( \text{assignVal} \), and \( u_0 \in \text{closure} \) or \( u_0 \equiv x_0 \), or it was built by \( \text{assignNew} \), and \( u_0 \notin \text{closure} \). Consider the first case, and let \( u_0 \equiv x_0 : u_0' \). Then by push-in rule \( x_0 : [u_0', u_0'^1 : \ldots, u_0'^n : u_n' \rightarrow u_n : z] \), and by transitivity \( x_0 : z \in \text{closure} \). But if \( x_0 : z \in \text{closure} \), by the construction of \( v \) by \( \text{assignVal} \), \( z(v_1) = z(v_2) = \text{value}(x_0 : z) \).

Now suppose \( v \) was built by \( \text{assignNew} \), type of \( u_0 \) is \( \langle A_1, \ldots, A_k \rangle \), for all \( A_i \), \( 1 \leq i \leq k \), \( u_0 : A_i \in \text{closure} \), and \( v = \{e_1, e_2\} \), where \( A_i(e_i) = \text{value}(u_0 : A_i) \), and the value of \( e_2 \) is given by \( \text{newRow}(p, (p, \emptyset) \ast) \). If \( z(e_1) \neq z(e_2) \), and \( |z| = 1 \), then \( z \notin (u_0, \emptyset) \ast \). By Observation 3.8, if for all \( u_i \), \( 1 \leq i \leq n \), \( u_i(e_1) = u_i(e_2) \) then \( u_i(e_2) \) cannot have been built by \( \text{newRow}(u_0 : u_i, (u_0, \emptyset) \ast) \). Therefore, for all \( u_i \) there exists a prefix \( u_i' \) of \( u_i \), such that \( u_0 : u_i' \in (u_0, \emptyset) \ast \). By transitivity, \( u_0 : z \in (u_0, \emptyset) \ast \), a contradiction.

Case 2: \( |z| > 1 \).

By definition, \( f \) is a downward NFD. We will consider two cases:

1. \( u_0 : z \in \text{closure} \):

   Let \( v \) be an arbitrary value of \( u_0(I) \). We will show that there exists a single value of \( v(z) \). From Observations 3.7 and 3.17, if there exists two distinct values of \( u_0 : q(v) \) then there must exist a prefix \( z' \) of \( z \) such that \( u_0 : z' \notin \text{closure} \), and \( u_0 : z \notin (u_0 : z', \emptyset) \ast \). But since \( |w| = 0 \) by full-locality rule, for all prefix \( z' \) of \( z \). Let \( z \equiv z'' \). By pull-out rule, \( u_0 : z''[\emptyset \rightarrow z'' \ast] \ast \). By transitivity, \( u_0 : z \in \text{closure} \), which contradicts our assumption. Therefore, if \( u_0 : z \notin \text{closure} \) then there exists no prefix \( z' \) of \( z \) such that \( u_0 : z' \in \text{closure} \) and for any element \( e \in u_0(I) \) there exists a single value of \( z(e) \).

2. \( u_0 : z \notin \text{closure} \):

   Let \( v \) be an arbitrary value of \( u_0(I) \). We will first show that if \( u_0 : z \notin \text{closure} \) then for any element \( e \in v \) there exists a single value of \( z(e) \). Suppose the contrary. By Observation 3.12, if for all prefix \( p \) of \( z \) \( u_0 : p \notin \text{closure} \) then there exists a unique value of \( z(e) \). Therefore, if there exists two different values of \( z(e) \) then there exists a prefix \( z' \) of \( z \) such that \( u_0 : z' \in \text{closure} \). Let \( z \equiv z' \equiv z'' \), and \( u_0 \equiv x_0 : u_0' \). Since there is no common prefix between \( z \) and any \( u \in U \), by the full-locality rule \( x_0 : u_0'[z' \rightarrow z' : z''] \ast \), by pull-out, \( x_0 : u_0' : z' \rightarrow z'' \ast \), and by push-in \( x_0 : [u_0' : z' \rightarrow u_0' : z' : z''] \ast \). Then, by transitivity, \( u_0 : z \in \text{closure} \), which contradicts our assumption. Therefore, if \( u_0 : z \notin \text{closure} \) then there exists no prefix \( z' \) of \( z \) such that \( u_0 : z' \in \text{closure} \) and for any element \( e \in u_0(I) \) there exists a single value of \( z(e) \).

Now we have to show that if \( u_0 : z \notin \text{closure} \) there are no two distinct elements \( v_1, v_2 \) in \( u_0(I) \) such that for all \( u \in U \) \( u_i(v_1) = u_i(v_2) \), and \( z(v_1) \neq z(v_2) \). Suppose there exists such elements \( v_1, v_2 \). By Observation 3.19, if for all \( u_i \in U \), \( u_i(v_1) = u_i(v_2) \) then for all \( u_i \) there exists a prefix \( u_i' \) such that \( u_i' \in \text{closure} \). Since there is no common prefix between any \( u_i \in U \) and \( z \), applying the prefix rule we get \( x_0 : u_0 : [u_1', \ldots, u_n' \rightarrow z] \ast \), and by push-in rule \( x_0 : [u_0', u_0'^1 : \ldots, u_0'^n : u_n' \rightarrow u_0 : z] \ast \). We've shown that for all prefix \( z' \) of \( z : z' \notin \text{closure} \). Then by Observation 3.4 since \( u_0(I) \) has
two elements then either \( u_0 \in \text{closure} \) or \( u_0 \equiv x_0 \). But then, by transitivity, \( u_0 : z \in \text{closure} \), a contradiction.

**Inductive Step:** \( |w| > 0 \).

Let \( w = A : w' \), and \( f = u_0 : [A : u_1, \ldots, A : u_k, u_{k+1}, \ldots, u_m \to A : z] \), where \( A \) is not prefix of any \( u_i, k < i \leq m \). By locality, \( u_0 : A[u_1, \ldots, u_k \to z] \). By inductive hypothesis, this NFD is satisfied. Therefore, if for every value \( v \) of \( u_0(I) \) all elements agree on the value of \( A \), then \( I \) cannot violate \( f \). So, there exists a value \( v \) of \( u_0(I) \) such that \( v \) has at least two elements, \( v_1, v_2 \), and \( A(v_1) \neq A(v_2) \). By Observation 3.4, either \( u_0 \in \text{closure} \) and \( u_0 : A \notin \text{closure} \), or \( u_0 \notin \text{closure} \) and \( u_0 : A \in \text{closure} \).

We will first show that if there exists two elements \( v_1, v_2 \in u_0(I) \) such that \( A(v_1) \neq A(v_2) \) and for all \( u_i, 1 \leq i \leq m, u_i(v_1) = u_i(v_2) \) then \( u_0 : z \in \text{closure} \). Let \( w = A : w', u_0 \equiv x_0 : u'_0 \), and \( f = u_0[u_1, \ldots, u_m \to w : z] \). By Observation 3.19, for all \( u_i, 1 \leq i \leq m \), there exists a prefix \( u'_i \) of \( u_i \), such that \( u_0 : u'_i \in \text{closure} \). Let \( u'_i \) be the longest prefix of \( u_i \) such that \( u_0 : u'_i \notin \text{closure} \), for all \( i, 1 \leq i \leq m \). We will consider two cases:

**Case 1:** For all \( u_i, u'_i \) is not a prefix of \( w : z \).

In this case, we can apply the prefix rule multiple times and get \( u_0 : [u'_0, u'_1, \ldots, u'_n \to w : z] \). By push-in rule \( x_0 : [u'_0, u'_1, \ldots, u'_n \to u'_0 \in \text{closure} \).

If \( x_0 : u'_0 \in \text{closure} \) or \( u_0 \equiv x_0 \), then by transitivity, \( u_0 : z \in \text{closure} \). If \( x_0 : u'_0 \notin \text{closure} \), then \( u_0 : z \in \text{closure} \) by Observation 3.4. By Observations 3.2 and 3.3 \( w \) was built by \( \text{assignNew}(u_0) \), and \( u_0 : A \notin (u_0, \emptyset)^* \). By construction, either \( v_1 \) or \( v_2 \) was built by \( \text{newRow}(u_0, (u_0, \emptyset)^*) \). Since for all \( u_i, 1 \leq i \leq m, u_i(v_1) = u_i(v_2) \), by Observation 3.9 for all \( u_i \) there exists a prefix \( u'_i \) such that \( u_0 : u'_i \in (u_0, \emptyset)^* \). Then by transitivity, \( u_0 : z \in (u_0, \emptyset)^* \). I.e., \( x_0 : u'_0[\emptyset \to z] \). By push-in rule \( x_0 : [u'_0 \to u'_0 : z] \). Let \( z = A : z' \), then by full-locality rule \( x_0 : [u'_0 : A \to u'_0 : A : z'] \). Since \( u_0 : A \in \text{closure} \), by transitivity, \( u_0 : z \in \text{closure} \).

**Case 2:** There exists a \( u_i \) such that \( u'_i \) is a prefix of \( w : z \).

Let \( P \) be the set of paths \( p \) such that \( p \) is the largest prefix in \( \text{closure} \) of some \( u_i \in U \), and \( p \) is also a prefix of \( w : z \). If for any \( p \in P, p \equiv w : z \), then by the reflexivity rule \( u_0 : w : z \in \text{closure} \). Let \( p_i \) be the element in \( P \) that corresponds to some \( u_i \in U \). Consider each \( p_i \), where \( |p_i| < |w : z| \). By Observation 3.2, \( p_i(v_1) \), \( p_i(v_2) \) were built either by \( \text{assignVal} \), or \( \text{newRow} \).

First, consider the case when \( p_i(v_1) \) was built by \( \text{newRow}(p, (p'_i, \emptyset)^*) \), where \( p'_i \) is a prefix of \( p \). Let \( u_i \equiv p_i : u'_i \), and \( \nu \) the value of \( p_i(v_1) \). If \( u(v_2) = \nu \), then by Observation 3.9 there exists a prefix \( u''_i \) of \( u'_i \) such that \( p_i : u''_i \in (p'_i, \emptyset)^* \), and by Observation 3.1 \( p_i : u''_i \in \text{closure} \).

If for all \( u_i \in U, p_i \) was built by function \( \text{newRow} \), then by Observation 3.9 and 3.1 there exists a prefix \( u''_i \) such that \( p_i : u''_i \in \text{closure} \). Using the prefix rule we get \( u_0 : [p_1 : u''_1, \ldots, p_m : u''_m \to w : z] \). Let \( p_k \) be the largest prefix common to some \( p_i, 1 \leq i \leq m \), and \( w : z \equiv p_k : z' \), \( u_0 \equiv x_0 : u'_0 \). By pull-out \( x_0 : [u'_0 : p_k, u'_0 : u'_0 \to u'_0 : p_k : z'] \). By push-in \( x_0 : [u'_0 : p_k, u'_0 : u'_0 \to u'_0 : p_k : z'] \). Therefore, \( x_0 : u'_0 : p_k : z' \equiv u_0 : w : z \in \text{closure} \).

Now suppose there exists a \( p_i(v_1) \) built by function \( \text{assignVal} \). If for all prefix \( u''_i \), \( p_i : u''_i \notin \text{closure} \) then by Observation 3.19 it can not be the case that \( u_i(v_1) = u_i(v_2) \). Therefore, for every \( u'_i \) there exists a prefix \( u''_i \) such that \( u''_i \in \text{closure} \). Therefore, we can use the same argument as used in the previous case to show that \( u_0 : w : z \in \text{closure} \).

Now, we’ll show that if \( u_0 : z \in \text{closure} \) then \( z(v_1) = z(v_2) \). Suppose not. Let \( w = A : w' \), where \( |w| \geq 1, v \) a value of \( u_0(I) \), and \( v_1, v_2 \) two elements in \( v \) such that \( v_1(A) \neq v_2(A) \), and for all \( u_i \),
1 ≤ i ≤ m, u_i(v_1) = u_i(v_2), but z(v_1) ̸= z(v_2). By Observation 3.15, either z(v_1), or z(v_2), or both were built by newRow(u_0 : z, (p, ∅)*), where u_0 : z ̸≡ p : z', and u_0 : z ̸∈ (p, ∅)*. By Observation 3.17, since there exists two distinct values of : z(v), |p| ≥ |u_0|. If |p| > |v| then by full-locality and pull-out rules, u_0 : p[[0] → z'], and u_0 : z ∈ (p, ∅)*, and therefore z(v_1) = z(v_2) = value(z). Therefore, |u_0| ≤ |p| ≤ |v|.

Let p ̸≡ u_0 : p', and f ̸≡ u_0 : [p' : u_1, u_2, ..., p' : u_k, u_{k+1}, ..., u_m → p' : z]. By full-locality and pull-out rules, p : [u_1, u_2, ..., u_k → z].

Let w ̸≡ p : w'. For all u_i, 1 ≤ i ≤ k, if p : u_i(v_1) = p : u_i(v_2), then by Observation 3.9, there exists a prefix u'_i of u_i such that p : u'_i ∈ (p, ∅)*. By Observation 3.7, if p : z'(v_1) was built by newRow(p : z', (p, ∅)*) then for all prefixes z'' of z', p : z'' ̸∈ (p, ∅)*. Therefore, for all u'_i, if p : u'_i ∈ (p, ∅)*, then p : u'_i is not a prefix of p : z', and we can apply the prefix rule multiple times to get p : [u'_1, u'_2, ..., u'_k → z']. But then p : z' ̸∈ (p, ∅)*, a contradiction. Therefore, z(v_1) (and z(v_2)) could not be built by newRow and z(v_1) = z(v_2) = value(u_0 : z).

2) I ̸|= x_0 : [X → y] if x_0 : y ∉ (x_0, X, Σ)*SC

We will first show that either x_0(I) has two elements or there exists a prefix y' of y such that x_0 : y'(I) was built by assignVal and x_0 : y'(I) has two elements. Suppose on the contrary, that there exists no path p, |p| ≥ |x_0| such that p is a prefix of x_0 : y, p(I) has two elements, and p(I) was built by assignVal. By construction, x_0(I) is built by assignVal, and by Observation 3.4 if it has only one element, for all labels A in x_0, x_0 : A ∈ closure. By construction, in this case, the value of all x_0 : A as are given by assignVal(x_0 : A). But by Observation 3.4 if assignVal(x_0 : A) has only one element then all labels in x_0 : A must also be in closure. But, by assumption, x_0 : y ∉ closure. Therefore, either there exists a prefix y'' of y such that x_0 : y''(I) was built by assignVal and it has two elements, or x_0(I) has two elements.

Let p be the largest prefix of x_0 : y built by assignVal with two elements. We’ve shown |p| ≥ |x_0|.

Let f = x_0 : [p : u_1, ..., u_k, u_{k+1}, ..., u_m → p : y']. We claim that if I ̸|= x_0 : [u_1, ..., u_k → y'], then I ̸|= f. If I ̸|= x_0 : [u_1, ..., u_k → y'] then there exists two elements v_1, v_2 in p(I) such that u_i(v_1) = u_i(v_2) for all i, 1 ≤ i ≤ k, and y'(v_1) ̸= y'(v_2). Let v_0 be the value of x_0 : (I) where p(v_0) = {v_1, v_2}. Take one value of u_i(w_0), for all i, k < i < m, and the values of u_i(v_1) = u_i(v_2) for all i, 1 ≤ i ≤ k. Then by assumption y'(v_1) ̸= y'(v_2) and therefore I ̸|= f.

Therefore, we have to show that if p is the largest prefix of x_0 : y built by assignVal, and p(I) = {v_1, v_2}, v_1 ̸= v_2, then for all u_i, 1 ≤ i ≤ k, there exists a value of u_i(v_1), u_i(v_2) such that u_i(v_1) = u_i(v_2), and y'(v_1) ̸= y'(v_2). By assumption, v_1 ̸= v_2. Since for all u_i, 1 ≤ i ≤ k, x_0 : p : u_i ∈ closure, by Observation 3.21 for all u_i there exists values of u_i(v_1), u_i(v_2), such that u_i(v_1) = u_i(v_2) = value(x_0 : p : u_i).

Let y' ≡ A_1 : ... : A_n. Since p is the largest prefix of x_0 : y built by assignVal, by Observation 3.22, for all A_i, 1 ≤ i ≤ n, A_1 : ... : A_i ∉ closure, and A_1 : ... : A_i(v_1), A_1 : ... : A_i(v_2) were built by assignNew. By Observation 3.12 there exists a single value of y'(v_1), and y'(v_2), and by Observation 3.18 they are distinct.

3.3 Discussion

Simple NFDs. Note that push-in and pull-out simply change between equivalent forms of NFDs. I.e., an NFD of form R : y : [x_1, ..., x_k → z] is equivalent to R : [y, y : x_1, ..., y : x_k → y : z]. Therefore, we could change the definition of an NFD to allow only relation names as the base path (x_0) of an NFD, without changing its expressive power.
In this simpler form of NFDs, it can be shown that there are only six inference rules: push-in and pull-out are unnecessary. Of the remaining rules, only locality must be modified to what we call full-locality: if

1. \( x_0 : [x : X, Y \rightarrow x : z] \)
2. \( x \) is not a proper prefix of any \( y \in Y \)

then \( x_0 : [x, x : X \rightarrow x : z] \).

Note that full-locality combines the pull-out and locality rules. As an example of the need to use full-locality rather than locality, consider the following:

**Example 3.3** Let \( f_1 \) be the NFD \( R : [A : B : C, A : D \rightarrow A : B : E] \). Applying the locality rule, we can get \( R : [A, A : B : C, A : D \rightarrow A : B : E] \), but not \( R : [A : B, A : B : C \rightarrow A : B : E] \). The latter is derivable using full-locality.

Although the simpler form of NFDs yields a smaller set of axioms, we believe that the first form, which allows an arbitrary base path, is more intuitive since it makes a syntactic distinction between inter- and intra-set dependencies.

**Comparison with inference rules for the relational model.** Since the simple form of NFDs appears to closely resemble the definition of functional dependencies for the relational model, the natural question that arises is: Can we infer all the simple NFDs using the derivation rules for functional dependencies (FDs) and multivalued dependencies (MVDs) for the relational model?

The answer to this question is no. The “extra” rules that are not part of Armstrong axioms, locality, singleton, and prefix, are the rules that allow us to infer NFDs based solely on the nested form of the data.

We will first informally define multivalued dependencies and present their inference rules. Then, using an example, show that some NFDs cannot be inferred using only Armstrong axioms and the MVD-rules.

Let \( R \) be a relation, \( U \) the set of attributes in \( R \), and \( X, Y \) subsets of \( U \). We say that “\( X \) multidetermines \( Y \)” (written \( X \rightarrow Y \)), or “there is a multivalued dependency of \( Y \) on \( X \)”, if given values for the attributes of \( X \) there is a set of zero or more associated values for the attributes of \( Y \), and this set of \( Y \)-values is not connected in any way to values of the attributes in \( U - X - Y \) [20]. In the following, we will consider the following inference rules for MVDs [20]:

- complementation: if \( X \rightarrow Y \), then \( X \rightarrow (U - X - Y) \).
- MVD-reflexivity: if \( Y \subseteq X \), then \( X \rightarrow Y \).
- MVD-augmentation: if \( X \rightarrow Y \), and \( V \subseteq W \), then \( XW \rightarrow YV \).
- MVD-transitivity: if \( X \rightarrow Y \), and \( Y \rightarrow Z \), then \( X \rightarrow (Z - Y) \).

Also, we will consider the following rules that relate FDs and MVDs:
• conversion: if $X \rightarrow Y$, then $X \rightarrow Y$.

• interaction: if $X \rightarrow Y$, $Z \subseteq Y$, and for some $W$ disjoint from $Y$, $W \rightarrow Z$, then $X \rightarrow Z$.

It’s been shown that for the relational model, Armstrong’s axioms plus the rules above are sound and complete for logical implication of FDs and MVDs considered together. So, the question we posed above can be rephrased as: are these rules complete for logical implication of NFDs? We will show that this is not the case. In the following example, we present a possible flatten representation of an instance from our model and how NFDs could be translated as a set of FDs and MVDs in this model.

**Example 3.4** Let $R$ be a relation with schema \(\{A : \{B, C\}, D : \{E, F\}\}\) with the following constraints:

$R : [D : E \rightarrow A : B]$

$R : [A : B \rightarrow A : C]$

$R : D : [E \rightarrow F]$.

Let $I$ be an instance of $R$:

<table>
<thead>
<tr>
<th>A</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Now suppose we build $I'$, a “flatten” representation of $I$ with the relational schema $R' = \{A, A : B, A : C, D, D : E, D : F\}$ as:

<table>
<thead>
<tr>
<th>A</th>
<th>A : B</th>
<th>A : C</th>
<th>D</th>
<th>D : E</th>
<th>D : F</th>
</tr>
</thead>
<tbody>
<tr>
<td>{B : 1, C : 2}</td>
<td>1</td>
<td>2</td>
<td>{E : 3, F : 5}</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>{B : 1, C : 2}</td>
<td>1</td>
<td>2</td>
<td>{E : 3, F : 6, E : 5, F : 7}</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>{B : 1, C : 2}</td>
<td>1</td>
<td>2</td>
<td>{E : 3, F : 6, E : 5, F : 7}</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>{B : 3, C : 4}</td>
<td>3</td>
<td>4</td>
<td>{E : 4, F : 5}</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

The NFDs defined on $R$ could be expressed on $R'$ as:

$R'[D : E \rightarrow A : B]$

$R'[A : B \rightarrow A : C]$


Notice that the last constraint needs to enforce equality on attribute $D$, since it is a local dependency in $D$. For example, in $I'$, it is not the case that $R'[D : E \rightarrow D : F]$.

Also there are two multivalued dependencies resulting from the unnesting of $R$:

$R'[A \rightarrow A : B, A : C]$

$R'[D \rightarrow D : E, D : F]$.

Using only Armstrong axioms on $R'$ we are able to derive dependencies as:
Other dependencies can be derived using the MVD-rules. For example:

- \( R' : [D \rightarrow A : B] \) can be derived using the complementation and interaction rules. Similarly, using the NFD-rules \( R : [D \rightarrow A : B] \) can be directly derived from the prefix rule.

- \( R' : [A \rightarrow A : B] \) can be derived using the interaction rule. Also, using the NFD-rules \( R : [A \rightarrow A : B] \) can be derived from locality, and push-in rules.

But, the dependency \( R' : [D \rightarrow A] \) cannot be derived from the rules for the relational model, although \( R : [D \rightarrow A] \) can be derived using the NFD rules singleton, transitivity, locality, and push-in. The reason is that although in \( R' \) we express the relation between a set-valued attribute and its elements by a multi-valued dependency, as in \( R' : [A \rightarrow A : B, A : C] \), we don’t actually express that \( B, \) and \( C \) are the (only) attributes in \( A \). Therefore, using the relational rules, we can derive \( R' : [D \rightarrow A : B], R' : [D \rightarrow A : C], R' : [A \rightarrow A : B], R' : [A \rightarrow A : C], \) but we cannot derive \( R' : [A : B, A : C \rightarrow A]\), which by transitivity would result \( R' : [D \rightarrow A]. \) The derivation step missing in this case is expressed by the singleton rule.

Some previous work on normal forms for the nested relational model, as of Ozsoyoglu and Yuan [15], and Mok, Ng, and Embley [14], use only the inference rules for FDs and MVDs of the relational model. But their data model is different because there are no labels for set-valued attributes, and therefore they cannot be referenced by any dependency. For example, the relation \( R \) in the example above would be defined in this model as \(<B, C^*, E, F^*>\), where \(<B, C^*>\) represents a set of records with attributes \( B, \) and \( C \). The first two dependencies defined on \( R \) would be defined as:

\[
R : [D : E \rightarrow A : B] \\
R : [A : B \rightarrow A : C]
\]

But the third dependency cannot be expressed in this model, since the absence of labels for set-valued attributes prevents equality on sets. In this simpler nested relational model, the derivation rules for FDs and MVDs are sound and complete [14]. One interesting point, though, is that in [14] one of the concerns in the normalization process is to avoid singleton sets. In this work, schemas are represented as scheme trees. Since the existence of singleton sets cannot be expressed or derived from the inference rules for the relational model, they define a condition on scheme trees in order to identify potential singleton sets. Our definition of NFDs and the NFD-rules allow singleton sets to be expressed and identified and it would be, therefore, a more general basis for the development of a normal form for the nested relational model.

**The Problem of Empty Sets.** As mentioned earlier, the presence of empty sets causes difficulties in reasoning since formulas such as \( \forall x \in R. P(x) \) are trivially true when \( R \) is empty. In particular, the transitivity rule is no longer sound in the presence of empty sets, as illustrated below.

**Example 3.5** The instance of \( R \) below satisfies \( R : [A \rightarrow B : C], R : [B : C \rightarrow D], \) but not \( R : [A \rightarrow D]. \)
One reasonable solution to this problem is to disallow empty sets only in certain portions of the schema; this is analogous to specifying NON-NULL for certain attributes in a relational schema. The transitivity rule can then be modified to reason about where empty sets are known not to occur. We do this by introducing a new relation follow between paths.

**Definition 3.5** Path expression $p_1$ follows $p_2$ if $p_1 = p'_1 A$, and $p'_1$ is a proper prefix of $p_2$.

Intuitively, $p_1$ follows $p_2$ if it only traverses the set-valued attributes traversed by $p_2$. For example, a path $A$ follows any path $p$, $|p| \geq 1$, since $A \equiv eA$, and $\epsilon$ is a proper prefix of any path. A path $A : B$ follows $A : B$, $A : C : D$, but not $A$, $E$, and $F : G$.

The new transitivity rule is then defined as: if

1. $x_0 : [X \to x_1], \ldots, x_0 : [X \to x_n]$
   $x_0 : [x_1, \ldots, x_n \to y]$
2. for all $p$ in $\{x_1, \ldots, x_n\} - X$, if $p$ does not follow $y$, then $p$ is known not to be an empty set

then $x_0 : [X \to y]$.

The fact that transitivity does not generally hold in the presence of empty sets has also influenced our definition of NFDs to allow only single paths on the right-hand side of functional dependencies rather than sets of paths.

Recall that in the relational model, a functional dependency (FD) $X \to Y$, where $X,Y$ are sets of attributes, can be decomposed into a set of FDs with single attributes on the right-hand side of the implication. Unfortunately, the decomposition rule follows from reflexivity and transitivity and cannot therefore be uniformly applied with NFDs in the presence of empty sets.

The presence of empty sets also affects the prefix rule. Consider the instance $I$ presented in Example 3.5. Notice that $I$ satisfies $R : [B : C \to E]$, but not $R : [B \to E]$. A modified prefix rule to take this into account is: if

1. $x_0 : [x_1 : A, x_2, \ldots, x_k \to y]$
2. $x_1$ has one or more labels, and $x_1$ is not prefix of $y$
3. $x_1$ is not an empty set

then $x_0 : [x_1, x_2, \ldots, x_k \to y]$
4 Conclusion

We have presented a definition of functional dependencies (NFD) for the nested relation model. NFDs naturally extend the definition of functional dependencies for the relational model by using path expressions instead of attribute names. The meaning of NFDs was given by defining their translation to logic.

NFDs provide a framework for expressing a natural class of dependencies in complex data structures. Moreover, they can be used to reason about constraints on data integration applications, where both sources and target databases support complex types.

We presented a set of inference rules for NFDs that are sound and complete for the case when no empty sets are present. Although for simplicity we have adopted the nested relational model, and the syntax of NFDs is closely related to this model, allowing nested records or sets would not change the inference rules presented significantly. However, new rules would have to be added to consider path expressions of record types as the current syntax only allows path expressions of set and base types. As an example, we would need a rule that states that if in $R \times x$ is a path of type $<A_1, \ldots, A_n>$, then $R : [x.A_1 \ldots x.A_n \rightarrow x]$, where "." indicates record projection.

In [7], Fischer, Saxton, Thomas and Van Gucht investigate how nesting defined on a normalized relation destroys or preserves functional and multivalued dependencies; they also present results on the interaction of inter- and intra-set dependencies. Their results are based on case studies of the cardinality of relations, and of the containment relation between the set of attributes over which the nesting is defined and the set of attributes involved in the dependency. Many results depend on the fact that a nested relation is a singleton set. In our definition of NFDs, both inter- and intra-set dependencies can be expressed. NFDs can also express that a given set is expected to be a singleton. As a result, our work generalizes their results by providing a general framework to reason about interactions between nesting and functional dependencies.

In future work, we intend to investigate a relaxation of the assumption that no empty sets are present in any instance, by requiring the user to define which set-valued paths are known to have at least one element. We believe this is a natural requirement to make, since definition of cardinality has long been recognized as integral part of schema design [6] and is part of the DDL syntax for SQL (NON-NULL). Generalizing the inference rules to this case would allow us to reason about constraints for a larger family of instances.

References


31


