

XML Functional Dependencies based on Tree Homomorphisms

Doctoral Thesis

Presented in partial fulfilment of the requirements for the degree of

DOCTOR RERUM NATURALIUM (DR. RER. NAT.)

at the

Faculty of Mathematics/Informatics and Mechanical Engineering,

Clausthal University of Technology

Diem-Thu Trinh

June 2009

Supervisor: Prof. Dr. Sven Hartmann

Co-Supervisor: Prof. Dr. Sebastian Link (Victoria University of Wellington)

Date of the Oral Examination: July 3, 2009

Chairman of the Board of Examiners: Prof. Dr. Jürgen Dix

Abstract

Functional dependencies form an important class of integrity constraints which has been well studied in the context of relational databases. Recently, the eXtensible Markup Language (XML) has emerged as a popular and flexible standard for modelling and managing data in a variety of domains. Although several attempts have been made to generalise the concept of functional dependencies to the context of XML, the search for an interesting yet practical formalisation of XML functional dependencies still poses a challenge.

In this dissertation I propose and investigate a new class of XML functional dependencies with properties (called pXFDs) defined on the basis of tree homomorphism. My first contribution is to demonstrate that one can reason about pXFDs efficiently. Through establishing a semantic equivalent between pXFD implication and logical consequence of propositional Horn clauses, I show that the problem of pXFD implication can be decided in time linear in the total number of essential v -properties in a set of pXFDs. My second contribution is to identify and prove a sound and complete axiomatisation for pXFDs. This yields an alternative approach for deciding the implication problem for pXFDs.

Based on these findings, I investigate the application of pXFDs in classifying well-designed XML schemas. The absence of redundancy is used as a suitable indicator of “good” design. For that I generalise two notions of redundancy (i.e., fact and value redundancy) from the relational data model and identify two corresponding normal forms for characterising them. I also show that the proposed normal forms can be checked efficiently: it is sufficient to examine pXFDs in a cover set rather than the set of all implied pXFDs.

Finally, I consider the problem of pXFD acquisition and discovery. To be able to benefit from pXFDs, it is imperative to have a good specification of pXFDs that describe the semantics of the data to be stored in an XML database. Discovery and acquisition are two complementary tasks which help in determining such a specification. The logical characterisation of pXFD implication allows me to develop a sample-based approach to support pXFD acquisition. For pXFD discovery, on the other hand, I adapt a method using difference sets and hypergraph transversals in the context of XML.

My investigations also demonstrate that pXFDs exhibit certain characteristics which are reminiscent of functional dependencies in the relational data model.

Contents

Abstract	iii
List of Figures	vii
List of Tables	ix
List of Algorithms	xi
Symbols and Notations	xiii
1 Introduction	1
1.1 XML Data Model	4
1.1.1 XML Schema Tree and XML Data Tree	4
1.1.2 Subgraph Terminology	6
1.2 The Role of Properties	7
1.3 XML Functional Dependencies with Properties	9
1.3.1 Examples Illustrating Expressiveness of pXFDs	10
1.3.2 Implication and Derivation of pXFDs	12
1.3.3 Observations About Property-Equality	12
1.3.4 Canonical pXFDs	15
1.4 Thesis Outline	23
2 Semantic Equivalence Theorem for pXFDs	25
2.1 Relating Canonical pXFD Implication to Horn Clauses	27
2.1.1 Horn Clause Encoding	28
2.1.2 Semantic Equivalence of Canonical pXFDs to Logic	29
2.1.3 Construction of Two- <i>v</i> -pre-image Data Tree	35
2.2 Application of the Semantic Equivalence	47
2.2.1 Deciding the Implication Problem for pXFDs	47
2.2.2 Sound and Complete Axiomatisation	50
2.2.3 Assistance with Constraints Acquisition	65
2.3 Summary	69

3	Dependency Discovery	71
3.1	Transversal Approach for pXFD Discovery	74
3.1.1	Canonical pXFD-Cover for Representing Satisfied pXFDs	74
3.1.2	Minimal Transversals for Finding Canonical pXFD-Cover	81
3.2	Transversal Approach: In Details	91
3.2.1	Finding Minimal Transversals	91
3.2.2	Finding Difference Sets from Dual Agree Sets	100
3.2.3	Finding Candidate RHSs	103
3.3	Finding Agree Sets	123
3.3.1	Finding \sqsubseteq -maximal Essential v -Ancestors in Agree Sets	124
3.3.2	Finding \sqsubseteq -maximal Essential v -Subgraphs in Agree Sets	126
3.4	Summary	149
4	Redundancy and Normal Forms	151
4.1	Fact Redundancy and FR-pXNF	152
4.2	Value Redundancy and VR-pXNF	157
4.3	Summary	163
5	Conclusion and Future Work	165
	Bibliography	169
	Acknowledgements	177
	Index	178

List of Figures

1.1	XML document about entries into photography competitions.	2
1.2	XML data tree T'_{photo} corresponding to XML document in Figure 1.1. . .	3
1.3	XML data tree T'_{dance} with dance school information.	4
1.4	XML schema tree T_{dance} and some of its subgraphs: walk boy and subgraph $\{\mathbf{boy}, \mathbf{girl}\}$	5
1.5	XML tree for the pre-image tree i_2 of v_{class} in T'_{dance} , and its projection to v_{class} -subgraph $\{\mathbf{boy}, \mathbf{girl}\}$	7
1.6	Counter-example construction when X, Y are not v -reconcilable.	15
1.7	Dance school schema tree showing its four v_{class} -units	19
2.1	Fagin's Semantic and Syntactic Proof of the semantic equivalence between FDs and propositional logic.	26
2.2	Pre-image trees p_1^W, p_2^W such that $p_1^W _W \neq p_2^W _W$ for some essential v -subgraph W but $p_1^W _X \doteq p_2^W _X$ for every v -subgraph X properly contained in W . If $ W $ is even then $ W = 2l$ and $m = l - 1$, otherwise $ W = 2m - 1$ and $m = l$	39
2.3	XML schema tree T_{abstr} containing three v -units: $U_1 = ABC$, $U_2 = DE$, $U_3 = F$	40
2.4	Pre-image trees $p_1^{W_1}, p_2^{W_1}$ such that $p_1^{W_1} _X \doteq p_2^{W_1} _X$ if and only if v -subgraph X is properly contained in $W_1 = AB$	41
2.5	Pre-images $p_2^{W_1}, p_2^{W_2}$ merged to form $p_2^{U_1}$	43
2.6	Pre-images $p_2^{U_1}, p_2^{U_2}, p_2^{U_3}$ merged to form p_2 . Then p_1, p_2 merged to form T_{abstr} -compatible data tree T'_{abstr}	46
2.7	Constraint acquisition by iterative inspection of candidates.	66
2.8	Flowchart of the propositional tableaux method for pXFDs acquisition. . .	68
3.1	$T_{purchase}$ -compatible XML data tree $T'_{purchase}$	83
3.2	XML schema tree $T_{purchase}$	83
3.3	Minimal transversals computation for $\mathfrak{D}_{T'_{purchase}}(v_{Purchase})$	99
3.4	Relationship between essential v -ancestors and candidate v -ancestor RHSs. .	106
3.5	Hasse diagram for the poset $(candRHS_{T'}(v) _U - \{\emptyset\}, \sqsubseteq)$ from Example 3.67. .	122
3.6	Relational representation of XML data tree $T'_{Purchase}$	140
3.7	Summary of pXFD Discovery Process.	148

List of Tables

2.1	System $\mathfrak{F}_{canonical}$ of inference rules for (canonical) pXFDs	50
2.2	System $\mathfrak{F}_{general}$ of inference rules for pXFDs	61
2.3	α -rule	67
2.4	β -rule	67
3.1	Summary of main transversal approach for discovering pXFDs in $T'_{purchase}$.	89

List of Algorithms

2.1	Horn_SAT(unit propagation)	48
2.2	property_closure	63
2.3	essential_property_closure	63
3.4	find_minimalTransversals	96
3.5	extend_minimalTransversals	97
3.6	find_differenceSets-fromAgreeSets	103
3.7	find_canonicalCover	104
3.8	discoverXFDs-withAncestors	109
3.9	find_subgraphRHSs	118
3.10	discoverXFDs-withSubgraphs	121
3.11	find_agreeSets-projectedToAncestors	125
3.12	find_agreeSets-projectedToSubgraphs(Pairwise)	127
3.13	find_agreeSetOnePair	129
3.14	lexical_precedes	132
3.15	value_precedes(oracle)	133
3.16	find_nonnullPartitions	139
3.17	find_agreeSets-projectedToSubgraphs(Partition)	146
3.18	find_stripPartitionDB	147
3.19	find_agreeSets	147

Symbols and Notations

XML Graph Model:

T	XML tree or XML schema tree	
T'	XML data tree	
$\phi(\cdot)$	homomorphism between two XML trees	
V_T	node set of T	
A_T	arc set of T	
r_T	root node of T	
\emptyset	empty subgraph of T	
$\check{A}_T(v)$	v -ancestor set	set of all v -ancestors in T
n, m, \dots	a node or v -ancestor	
$\check{S}_T(v)$	v -subgraph set	set of all v -subgraphs in T
W, X, \dots	a subgraph or v -property	
A, B, \dots	a walk	
$\mathcal{U}_T(v)$	v -unit set	set of all v -units in T
U	a v -unit	
$\eta(U)$	identifier for v -unit U	
$T(v)$	total v -subgraph in T	
$P_{T'}(v)$	v -pre-image tree set	set of all pre-image trees of v in data tree T'
p_1, p_2	pre-image trees	

Tree Labels:

E, A	<i>kind</i> labels on nodes	indicates element or attribute
$*, +, ?, \langle no\ label \rangle$	frequency labels on arcs	indicates occurrence restriction: arbitrarily many, at least one, at most one, exactly one

Tree Node/Walk Referencing:

v_{lbl}	node with name lbl	
i_k	node with node id k	
$1b1$	walk with leaf label lbl	

Running Example XML Trees:

$T_{purchase}$	Purchase schema tree	see Figure 3.2
$T'_{purchase}$	Purchase data tree	see Figure 3.1
T_{dance}	Dance school schema tree	see Figure 1.4
T'_{dance}	Dance school data tree	see Figure 1.3

XFD Proposals:

XFD	XML functional dependencies	
AL-XFD	XFDs from Arenas and Libkin,	see [2, 3]
HL-XFD	XFDs from Hartmann and Link (with pre-image semantics)	see [39]
pXFD	XFDs with Properties	see Definition 1.6

pXFDs and Canonical pXFDs:

v	target node	
$\mathcal{W}, \mathcal{X}, \dots$	a v -property set	
$\mathcal{X}^{\check{\Delta}}, \mathcal{Y}^{\check{\Delta}}$	a set of v -ancestors	
$\mathcal{X}^{\check{\mathfrak{S}}}, \mathcal{Y}^{\check{\mathfrak{S}}}$	a set of v -subgraphs	
$T' _X$	projection to v -property X	see Definition 1.4
\doteq	property-equality	see Definition 1.5
$=$	equality on identity	
$\models_{T'}$	satisfied in data tree T'	see Definition 1.6
$\vartheta(\mathcal{X})$	refinement of \mathcal{X}	see Definition 1.31
$\text{lca}(\mathcal{X})$	lowest contained v -ancestor of \mathcal{X}	
$\mathbf{E}_T^{\check{\Delta}}(v)$	essential v -ancestor set	see Definition 1.17
$\mathbf{E}_T^{\check{\mathfrak{S}}}(v)$	essential v -subgraph set	see Definition 1.19

pXFDs Implication and Derivation:

σ	a pXFD	
LHS_σ, RHS_σ	the LHS or RHS of pXFD σ	
Σ	a pXFD set	
Σ^*	implied pXFD set	
\models	pXFD implication	
\mathfrak{F}	system of inference rules	
$\mathfrak{F}_{canonical}$	axiomatisation for canonical pXFDs	see Theorem 2.24
$\mathfrak{F}_{prelim}, \mathfrak{F}_{general}$	axiomatisation for pXFDs	see Theorem 2.27 and 2.38
$\Sigma^+, \Sigma_{\mathfrak{F}}^+$	derived pXFD set (with system \mathfrak{F})	
$\vdash, \vdash_{\mathfrak{F}}$	pXFD derivation (with system \mathfrak{F})	
\mathcal{X}^*	property closure of \mathcal{X}	$= \bigcup \{Y \in \check{\mathfrak{S}}_T(v) \cup \check{\Delta}_T(v) \mid v : \mathcal{X} \rightarrow Y \in \Sigma^*\}$
$\mathcal{X}^{*\mathbf{E}}$	essential property closure of \mathcal{X}	$= \bigcup \{Y \in \mathbf{E}_T^{\check{\mathfrak{S}}}(v) \cup \mathbf{E}_T^{\check{\Delta}}(v) \mid v : \mathcal{X} \rightarrow Y \in \Sigma^*\}$

Relating to Semantic Equivalence Theorem for pXFDs:

H_σ, H_Σ	encoding of pXFDs as Horn Clauses	
H_T	base translation	encoding of XML schema tree as Horn Clauses
\mathcal{V}	propositional variable set	
$\varphi(\cdot)$	mapping essential v -properties to propositional variables	
$\mathbb{B}(\cdot)$	boolean truth assignment	
\mathcal{E}_T	equality set	see Definition 2.8
\mathcal{NE}_T	non-equality set	see Definition 2.9
$\mathcal{NE}_T _U$	projection of non-equality set to U	see Definition 2.11

FDs and its Semantic Equivalence:

R	relation schema	
r	relation	
t_1, t_2	tuples	
$t[X]$	restriction of tuple t to set X of attributes	
σ^r	(relational) FD	
Σ^r	(relational) FD set	
\models_r	(relational) FD implication	
$\mathfrak{F}\mathfrak{A}$	FD axiomatisation (sound and complete system e.g., Armstrong Axioms)	
$F_{\sigma^r} = \{h_\sigma\}, F_{\Sigma^r}$	Fagin's Horn Encoding	

Relating to pXFD Discovery:

\sqsubseteq, \sqsupseteq	p-subsumes	see Definition 1.36
\sqsubset, \sqsupset	strictly p-subsumes	see Definition 1.36
$\sqsubseteq\text{-min}(\mathcal{X})$	\sqsubseteq -minimal elements of set \mathcal{X}	$= \{X \in \mathcal{X} \mid \exists X' \sqsubset X, X' \in \mathcal{X}\}$
$\sqsubseteq\text{-max}(\mathcal{X})$	\sqsubseteq -maximal elements of set \mathcal{X}'	$= \{X \in \mathcal{X} \mid \exists X' \sqsupset X, X' \in \mathcal{X}\}$
$\mathfrak{C}_{T'}(v)$	canonical pXFD-cover	see Definition 3.10
$\check{\mathfrak{A}}\mathfrak{C}_{T'}(v), \check{\mathfrak{S}}\mathfrak{C}_{T'}(v)$	pXFDs in the canonical pXFD-cover with only subgraph/ v -ancestor RHS	
$\mathcal{D}, \mathcal{D}_{\{p_1, p_2\}}(v)$	v -difference set (of pre-image trees p_1, p_2)	see Definition 3.14
$\mathfrak{D}, \mathfrak{D}_{T'}(v)$	family of all v -difference sets (of T')	see Definition 3.14
$\mathfrak{D}_{T'}^Y(v)$	family of all v -difference sets of T' modulo Y	see Definition 3.14
\mathcal{T}	transversal	see Definition 3.16
$\mathfrak{T}(\mathfrak{D})$	family of all minimal transversal of \mathfrak{D}	see Definition 3.16
$\text{candRHS}_{T'}(v)$	family of candidate RHSs	see Definition 3.20

Relating to Minimal Transversal Computation:

\mathcal{H}, \mathcal{G}	hypergraph	
τ	hitting set	
$\mathfrak{H}\mathfrak{S}(\mathcal{H})$	family of all minimal hitting set of \mathcal{H}	
\vee	melding	see Definition 3.29

Relating to candidate RHSs:

c_{m_i}	m_i -partition class	see Section 3.2.3
$\text{RHS}(m_i)$	candidate v -ancestor RHS	see Definition 3.45
$\langle S \rangle$	\cap -ideal generated by S	see Definition 3.52
\check{U}	set of all v -subgraphs contained in U	
$\mathfrak{U}\mathfrak{C}_U$	family of all complete u-closed subsets for U	
$\text{max}(\mathfrak{A})$	set of all \sqsubseteq -maximal members of sets in \mathfrak{A}	$= \bigcup_{\mathcal{A} \in \mathfrak{A}} \sqsubseteq\text{-max}(\mathcal{A})$
$\mathcal{A} \sqcap \mathcal{B}$	$\sqsubseteq\text{-max}(\{a_i \cap b_j \mid a_i \in \mathcal{A}, b_j \in \mathcal{B}\})$	

Relating to v -difference sets:

$\sqsubseteq_{- \min}(\mathcal{D})$	representative subset of \sqsubseteq_{-} -upward-closed set \mathcal{D}	
$\mathbf{rep}(\mathcal{D})$	family of representative subsets of \sqsubseteq_{-} -minimal elements in v -difference sets in \mathcal{D}	$= \{\sqsubseteq_{- \min}(\mathcal{D}) \mid \mathcal{D} \in \mathcal{D}\}$
$\sqsubseteq_{- \min}(\mathcal{D})$	family of \sqsubseteq_{-} -minimal v -difference sets	$= \{\mathcal{D} \in \mathcal{D} \mid \exists \mathcal{D}' \subset \mathcal{D}, \mathcal{D}' \in \mathcal{D}\}$
$*\mathcal{D}$	family of \sqsubseteq_{-} -minimal representative subsets of v -difference sets in \mathcal{D}	$= \sqsubseteq_{- \min}(\mathbf{rep}(\mathcal{D}))$
$\mathcal{D}_{\{p_1, p_2\}}(v) _U$	projected to U	$= \{X \mid X \sqsubseteq U \text{ and } X \in \mathcal{D}_{\{p_1, p_2\}}(v)\}$

Relating to v -agree sets:

$\mathcal{A}, \mathcal{A}_{\{p_1, p_2\}}(v)$	v -agree set (of pre-image trees p_1, p_2)	see Definition 3.38
$\mathfrak{A}, \mathfrak{A}_{T'}(v)$	family of all v -agree sets (of T')	see Definition 3.38
$\sqsubseteq_{- \max}(\mathcal{A})$	representative subset of v -agree set \mathcal{A}	
$\mathbf{rep}(\mathfrak{A})$	family of representative subsets of \sqsubseteq_{-} -maximal elements of v -agree sets in \mathfrak{A}	$= \{\sqsubseteq_{- \max}(\mathcal{A}) \mid \mathcal{A} \in \mathfrak{A}\}$
$\mathcal{A}_{\{p_1, p_2\}}(v) _U$	projected to U	$= \{X \mid X \sqsubseteq U \text{ and } X \in \mathcal{A}_{\{p_1, p_2\}}(v)\}$
$\overline{\mathcal{A}_{\{p_1, p_2\}}(v) _U}$	$= \{U - Y \neq \emptyset \mid Y \in \sqsubseteq_{- \max}(\mathcal{A}_{\{p_1, p_2\}}(v) _U)\}$	
$\mathcal{A}_{\{p_1, p_2\}}(v) _{\check{\mathbb{A}}}$	project to v -ancestors	$= \{X \mid X \in \check{\mathbb{A}}_T(v) \text{ and } X \in \mathcal{A}_{\{p_1, p_2\}}(v)\}$
$\mathcal{A}_{\{p_1, p_2\}}(v) _{\check{\mathbb{S}}}$	project to v -subgraphs	$= \{X \mid X \in \check{\mathbb{S}}_T(v) \text{ and } X \in \mathcal{A}_{\{p_1, p_2\}}(v)\}$
$\check{\mathbb{S}}\mathfrak{A}_{T'}(v)$	project to v -subgraphs	$= \{\mathcal{A}_{\{p_1, p_2\}}(v) _{\check{\mathbb{S}}} \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}$
$=_v$	value-equality	
\leq_v	value-precedes	see Definition 3.72

Relating to agree-partitions:

π	partition	
$\hat{\pi}, \widehat{\mathfrak{N}}_{\{X\}}(v)$	stripped partition	no singleton partition class
$\Pi_{\mathcal{X}}(v)$	agree-partition induced by \mathcal{X}	see Definition 3.74
$\perp_{\mathcal{X}}(v)$	null class induced by \mathcal{X}	see Definition 3.76
$\mathfrak{N}_{\mathcal{X}}(v)$	non-null partition induced by \mathcal{X}	see Definition 3.76
$\Pi(v)$	partition database	$= \{\Pi_{\{X\}}(v) \mid X \in \mathbf{E}_{T'}^{\check{\mathbb{S}}}(v)\}$
$\widehat{\Pi}_{\{X\}}(v)$	stripped partition database	
$ec(p)$	$= \{(X, i) \mid p \in g_i \in \widehat{\Pi}_{\{X\}}(v) \text{ and } \Pi_{\{X\}}(v) \in \Pi(v)\}$	
$\mathbf{lmn}(B)$	lowest multiple v -ancestor in walk B	
P_w, P_v	subset of pre-images of w/v	
$\alpha_v(w'), \alpha_v(P_w)$	lifting	see Definition 3.82
$\pi(P_v, P_w)$	card-partition of P_v modulo P_w	see Definition 3.84

Chapter 1

Introduction

Integrity constraints are important because of their many potential applications such as schema design, query processing, and data processing and maintenance (including tasks like data integration and cleaning) [27, 28, 29]. Over the years, a variety of constraint formalisms has been proposed but one of the most prominent classes of integrity constraints remains functional dependencies.

In the relational data model (RDM), *functional dependencies (FDs)* are ubiquitously defined along the lines of

Given a relational schema R , a *functional dependency over relational schema R* is a statement $X \rightarrow Y$ where X and Y are subsets of attributes in R . Moreover, given a relation r over schema R , $X \rightarrow Y$ is *satisfied* in r if and only if for every pair of tuples t_1, t_2 in r whenever $t_1[X] = t_2[X]$ then $t_1[Y] = t_2[Y]$, where $t[X]$ denotes the projection of tuple t on attribute set X .

That is, whenever a pair of tuples agrees on every attribute in X they must also agree on every attribute in Y . Since its introduction by Codd in the early 1970s [16], relational FDs have been well studied and the well-founded dependency theory for FDs constitutes an important part of database theory. It is assumed that the reader is familiar with FDs and the RDM. We recommend [1, 57, 67, 89] as possible references into the area.

The *eXtensible Markup Language (XML)* is a standard recommended by the World Wide Web Consortium (W3C) that uses textual tags to annotate data [97]. The content between a pair of opening and closing tags is known as an *element* and corresponds to a basic component of XML documents. The content of an element may contain attributes, some text string and/or other elements. The basic property of being *well-formed*, which is set out in the standard, ensures that XML documents can be easily processed by available XML parsers. Furthermore, the structure of XML documents can also be restricted through *validation* against a schema specification written in a language such as Document Type Definition (DTDs), XML Schema (XSDs) or RelaxNG (see [55, 75] for a survey).

From the requirement that there must be unique root element and elements must be properly nested, well-formed XML documents have rather hierarchical, tree-like structures and are often represented as *XML trees*. Figure 1.1 presents a sample XML document

```

<?xml version="1.0">
<photography>
  <entry>
    <competition>WPPI</competition>
    <year>2009</year>
    <entrant>Binh</entrant>
    <category>
      <format>Print</format>
      <section>Nature</section>
      <noOfImg>1</noOfImg>
    </category>
    <category>
      <format>Digital</format>
      <section>Nude</section>
      <noOfImg>4</noOfImg>
    </category>
    <category>
      <format>Digital</format>
      <section>Wedding</section>
      <noOfImg>1</noOfImg>
    </category>
    <fees>75euros</fees>
  </entry>
  <entry>
    <competition>WPPI</competition>
    <year>2009</year>
    <entrant>Lin</entrant>
    <category>
      <format>Print</format>
      <section>Wedding</section>
      <noOfImg>1</noOfImg>
    </category>
    <category>
      <format>Digital</format>
      <section>Nude</section>
      <noOfImg>1</noOfImg>
    </category>
    <category>
      <format>Digital</format>
      <section>Wedding</section>
      <noOfImg>4</noOfImg>
    </category>
    <fees>55euros</fees>
  </entry>
</photography>

```

Figure 1.1: XML document about entries into photography competitions.

about entries into photography competition broken down by various categories. The XML document can be alternatively depicted as the XML tree in Figure 1.2.

There are various advantages of XML that have contributed to its popularity, including: extensible markup, self-describing data, syntactical flexibility and portability. Unlike other markup languages such as HTML, XML documents can introduce new tags as they are needed for the application at hand. Provided that tags have been appropriately chosen for the application domain, XML data are self-describing. The syntactical flexibility of XML, for example permitting elements to sometimes exist, makes it better suited for modelling heterogeneous and semi-structured data such as scientific data in the area of molecular biology. Moreover, being simply text which can be displayed by any web browser or text editor as well as more customised applications, XML data is highly more portable than their counterpart relational data.

Nowadays, XML is popular as the de facto standard for data exchange over the web, but is also emerging as a non-relational data model. With the increasing amounts of XML data being generated, there is an acute awareness that we need to consider how XML data can be stored persistently and queried. This brings into play the notion of *XML databases* and the need for a well-founded database theory for XML to support the development of such technologies.

In contrast to the singular notion of FDs in the relational context, the complex nature of XML has sparked a multitude of proposals for *XML functional dependencies (XFDs)*.

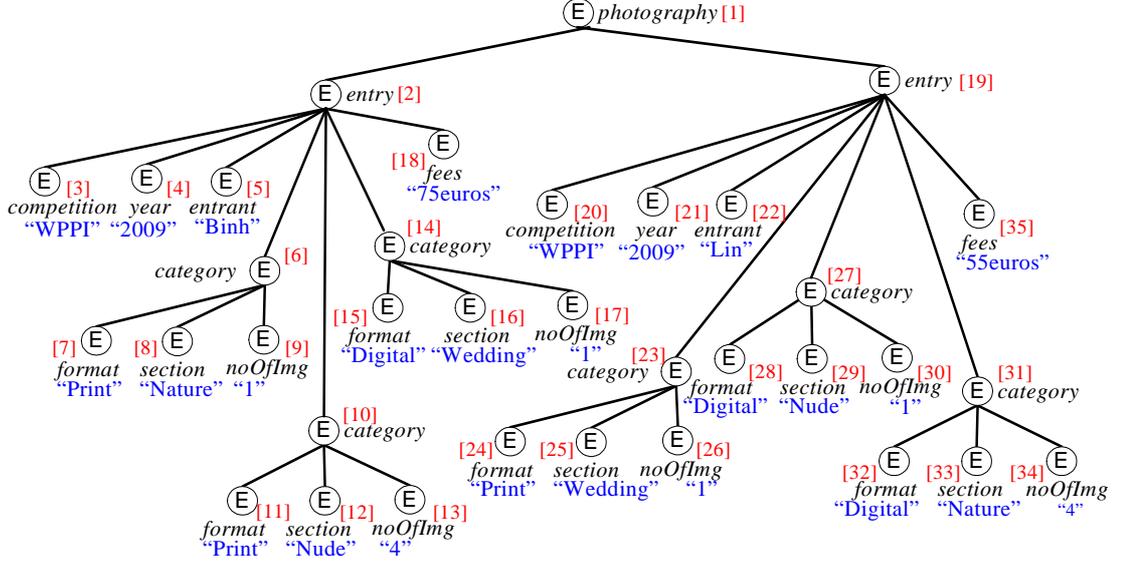


Figure 1.2: XML data tree T'_{photo} corresponding to XML document in Figure 1.1.

These XFD proposals, although deviate in the level of expressiveness, are justified by natural occurrences in XML data. Some examples of XFD proposals include [2, 39, 56, 60, 65, 93, 98, 102]. Moreover, because of the hierarchical structure of XML documents, XFDs can also be formalised by notions of functional dependencies in complex data models (e.g., [40]).

As one would expect, a significant source of differences between existing XFD proposals lies in the choice of what to assume about the underlying XML data model. For example, whether to consider:

- ordering among sibling elements;
- DTD or path-based data model;
- constraints over the number of sub-elements;
- attributes and/or ID;
- recursive elements;
- choice elements;

These assumptions give rise to the notion of *XML schemas* over which XFDs can be defined. Here we are using “XML schema” in deference to the term “relational schema”, and not as a reference to any particular XML schema specification language.

Other differences between the XFD proposals relate to how the notions of “attributes”, “tuples” and “agreement” are generalised to XML, for example:

- Instead of “tuples” one can consider tree-tuples [2], generalised tree-tuples [102], almost copies or pre-image trees [39], instance sets [60, 93], nodes [98]
- Instead of “attributes” one can consider downward paths to any nodes [2, 56, 60, 93], downward paths or subtrees to only leaves [39], downward and upward path [98, 102]

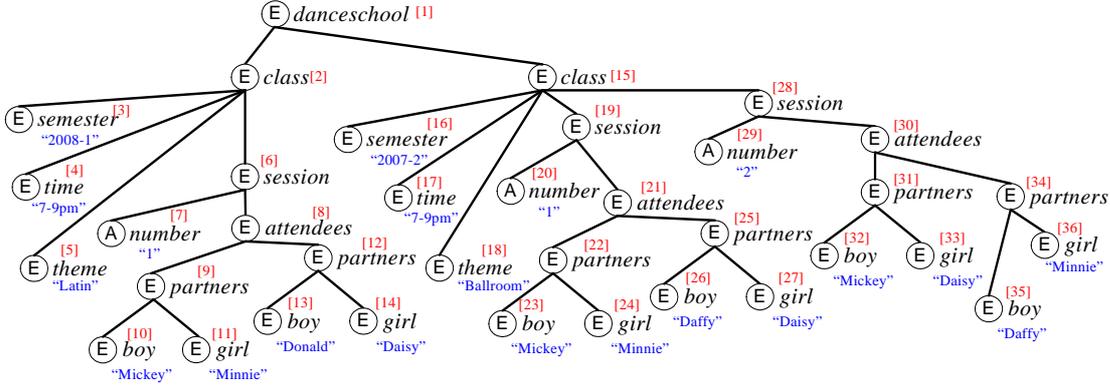


Figure 1.3: XML data tree T'_{dance} with dance school information.

- For “agreement” one can consider value-equality [39, 98], value- and node-equality [2, 102]

Despite the abundance of proposals, there are few detailed investigations into the different classes of XFDs. Consequently it remains difficult to assess and understand the trade-offs made by the different proposals, for example, between expressiveness, tractability and comprehensibility. In this dissertation we closely examine one particular class of *XML functional dependencies with properties (called pXFDs)* defined on the basis of tree homomorphism.

The remainder of the chapter sets out our proposal of pXFDs. We briefly present the graph-base XML data model from [39] together with some additional terminologies and notations which will be useful throughout the rest of the thesis. Then we highlight the differences in expressiveness between pXFDs and several other classes of XFDs. This is followed by a formal definition of pXFDs and some examples to further illustrate the sort of constraints that may be expressed by pXFDs. Finally, we identify an indispensable subclass of *canonical pXFDs* which is shown to be sufficient for capturing all constraints expressible with pXFDs. An outline of the rest of the thesis and our major contributions conclude the chapter.

1.1 XML Data Model

Our investigation uses the simple *XML graph model* proposed in [39, 43]. In this section, we provide some preliminary notations with regard to the XML graph model. Some familiarity with graph terminology is assumed.

1.1.1 XML Schema Tree and XML Data Tree

An *XML tree* is a rooted tree T with node set V_T , arc set A_T , root r_T , and mappings $name : V_T \rightarrow Names$ and $kind : V_T \rightarrow \{E, A\}$. In an XML tree, the symbols E and A

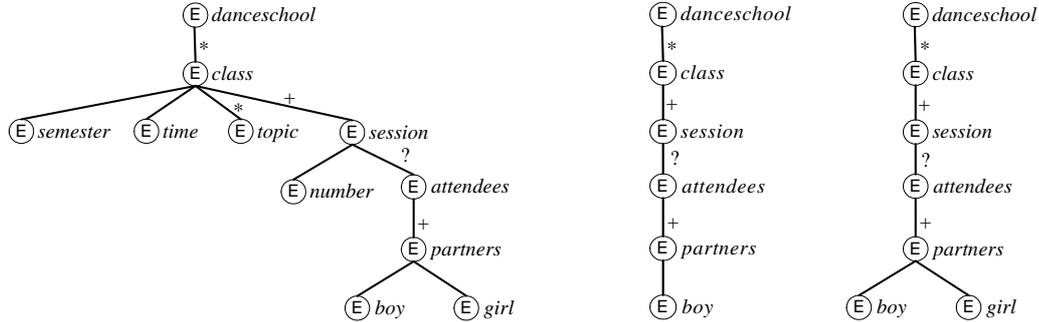


Figure 1.4: XML schema tree T_{dance} and some of its subgraphs: walk boy and subgraph $\{boy, girl\}$.

indicate elements and attributes respectively, with attributes only appearing as leaf nodes. An *XML data tree* is an XML tree T' with a mapping *valuation* : $L_{T'} \rightarrow String$ assigning string values to leaves. An *XML schema tree* is an XML tree T with frequencies $?, 1, *, +$ assigned to its arcs, where (i) arcs to attribute nodes may only have frequency $?$ or 1 and, (ii) no two siblings have the same name and kind.

We use XML schema trees to depict the structural summaries of a collection of XML data trees. Therefore, we use the notion of compatibility to convey that a particular XML data tree indeed conforms to the structural summaries depicted by some XML schema tree T . A data tree T' is *T-compatible* whenever there is a *homomorphism* $\phi : V_{T'} \rightarrow V_T$ (i.e., root-preserving, name-preserving, kind-preserving and arc-preserving mapping) such that for every node v' of T' and every arc $a = (\phi(v'), w)$ of T , the number of arcs $a' = (v', w')$ mapped to a is at most one if a has frequency label $?$, exactly one if a has frequency label 1 , at least one if a has frequency label $+$, and arbitrarily many if a has frequency label $*$.

Example 1.1. Figure 1.3 and Figure 1.4 shows some examples of XML trees. Particularly, data tree T'_{dance} is T_{dance} -compatible. \square

Path expressions are commonly used for addressing nodes within an XML documents and therefore plays an important role in many XFD proposals. The W3C recommends *XPath* as a language by which we can specify path expressions [95]. However, the standard allows numerous axis of navigation between nodes in an XML document and is therefore, often deemed to be too expressive for defining XFDs. In this thesis, we will consider only simple (downward) paths in XML trees, that is, a *path* is a sequence of symbols from alphabet L where two consecutive symbols in the sequence correspond to nodes which are parent/child of one another in the XML tree. We separate consecutive symbols in the sequence with the distinguished symbol $"/$. For the alphabet L , we may consider either the set of all element and attribute names or alternatively the set of all node ids.

A path in an XML schema tree T is *simple* if and only if it contains no arcs with frequency other than $?$ and 1 . A node v is *simple* if and only if the rooted path to v is simple. For a given node v , any node n which has a (possibly empty) path to v is called

a *v-ancestor*. We can also say v is a *descendant* of n or n is an *ancestor* of v . This of course means that, we consider every node to be its own ancestor and descendant. By $\check{A}_T(v)$ we denote the set of all v -ancestors of T . Furthermore, n is a *simple ancestor* of v (or equivalently v is a *simple descendant* of n) if and only if the path connecting n with v is simple. Clearly, every node is also its own simple ancestor and simple descendant, and every simple node is a simple descendant of the root node.

1.1.2 Subgraph Terminology

Let T be an XML tree. Given a set $L_T \subseteq V_T$ of leaves of T and a node v in T , a *walk* of T is a path from the root to a member of L_T and every walk containing v is called a *v-walk*. A *subgraph* W of T is a (possibly empty) set of walks of T and, a subgraph of T is a *v-subgraph* if and only if each of its walks contains v . Let $|W|$ denote the number of walks contained in subgraph W . By $\check{S}_T(v)$ we denote the set of all v -subgraphs of T . It is easy to see that $\check{S}_T(v)$ contains the empty set and is closed under the union, intersection and difference operators.

Notations: In the examples, we will use the following notations when referring to nodes, walks and subgraphs. In an XML tree, a node with name lbl is referred to as v_{lbl} and a node with node id k is referred to as i_k . Note that while every node in an XML tree has a unique node id, their name may not be unique.

We refer to a walk by the name of its leaf node and correspondingly we refer to a subgraph by the set of its leaves names. This is possible because the thesis purposefully contain only XML schema trees whose leaves have pairwise distinguished names. For the running example about purchases, all leaf names in an XML tree are distinct with respect to their first two letters, and so we may simply denote a walk by the first two letters of its leaf name rather than the full name. This abbreviation is useful when discussing large subgraphs and/or sets of subgraphs.

Clearly, a walk or subgraph of T describes again an XML tree. For a singleton subgraph, we may use the term “subgraph” and “walk” interchangeably. Accordingly, we will omit set parenthesis for singleton subgraphs. For example, we write B instead of $\{B\}$ where B is a walk in some XML tree T . When discussing *generic examples*, we may also simplify how subgraphs are denoted by omitting set parenthesis and delimiting commas. By generic example we mean the case where every walk is simply denoted with a single uppercase letter of the alphabet. For example, for an XML tree with walks A, B, C, D we may write simply ABC to refer to subgraph $\{A, B, C\}$. \square

Any two v -walks belong to the same *v-unit* if and only if both contain some node w , which is a proper descendant of v , and whose incoming arc is of frequency other than ? and 1. In other words, a *v-unit* is a v -subgraph U such that either (1) U consists of a single walk whereby the path from v to the leaf is simple or (2) U consists of all w -walks with w being a proper descendant of v whose incoming arc is the only arc in the path from v to w having frequency other than ? and 1. Note that v -units induce a partition on the set of all v -walks. Moreover, we can associate every v -unit U with a unique identifier

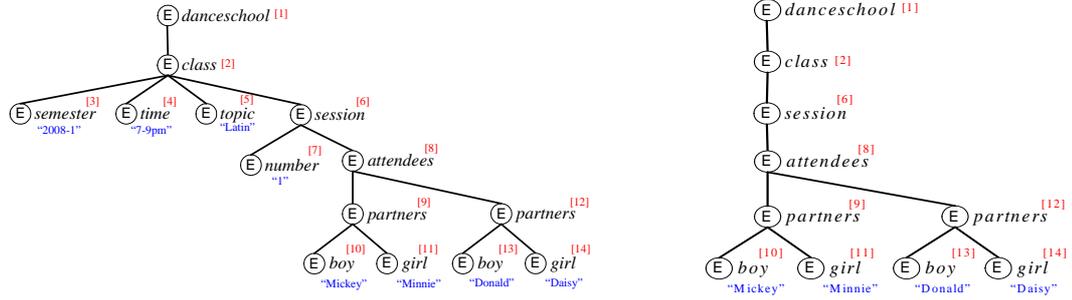


Figure 1.5: XML tree for the pre-image tree i_2 of v_{class} in T'_{dance} , and its projection to v_{class} -subgraph $\{\text{boy}, \text{girl}\}$

$\eta(U)$ as follows: If U is a singleton set then $\eta(U)$ is the leaf node, otherwise $\eta(U)$ is the node w as stated in (2) above.

The *total v -subgraph*, denoted by $T(v)$, is the set of all v -walks of XML tree T . Given a fixed node v of T , a *pre-image tree of v* in T' is just a total w -subgraph with $\phi(w) = v$. Suppose every node in a data tree T' is assigned a unique node id, then the node id for node w can be used to identify the total w -subgraph in T' . By $P_{T'}(v)$ we denote the set of all pre-image trees of v in T' . Note that a pre-image tree of v is also an XML tree.

Example 1.2. There are two pre-image trees of v_{class} in the dance school data tree T'_{dance} , one of which is depicted in Figure 1.5. There are two $v_{session}$ -units in the dance school schema tree T_{dance} : subgraph $\{\text{boy}, \text{girl}\}$ and walk number. \square

1.2 The Role of Properties

Consider again the XML data tree T'_{photo} as depicted in Figure 1.2. Note that the figure also shows the node id for every node assigned according to a depth-first traversal of the tree. Some valid paths in T'_{photo} are:

- $p_1 = \text{photography}/\text{entry}/\text{category}/\text{format}$
- $p_2 = \text{photography}/\text{entry}/\text{category}/\text{section}$
- $p_3 = \text{photography}/\text{entry}/\text{category}/\text{noOfImg}$
- $p_4 = \text{photography}/\text{entry}/\text{fees}$

The path $\text{photography}/\text{entry}/\text{noOfImg}$ is however not valid with respect to T'_{photo} because noOfImg is not a sub-element of entry .

It is regular practice for the competition organisers to fix the pricing plan every year according to the number of images entered per section. For example, suppose the entry fees charged for the WPPI competition in 2009 is as follows:

	Digital	Print
1 image in 1 section	10euros	15euros
4 images in 1 section	30euros	50euros

The pricing schedule is stored in a separate XML document which is why only the total fees paid by each entrant is recorded in T'_{photo} . Consequently we have the constraint

Constraint 1. The collection of information about the number of images entered into each section in each format determines the total fees of an entry.

With a path-based XFD (e.g., AL-XFDs proposed in [2, 3]) the closest that we come to expressing this constraint is by the XFD:

$$\{p_1, p_2, p_3\} \rightarrow \{p_4\}$$

But this is not quite what we want. In our data tree T'_{photo} , there are two entrants who have the same collection of format, the same collection of section, and the same collection of number of images entered but the fees charged are different. This means that the AL-XFD above is violated in T' . But the data tree actually satisfies Constraint 1 since it is easy to verify that the fees recorded are consistent with the pricing plan given. Although both Lin and Binh enter a total of 6 images into three sections, Lin pays less because she enters more images as digital files, while Binh pays more because he prefers to submit his images as prints so that he can have full control over image quality.

The above constraint helps to highlight two common gaps in the expressiveness of many existing classes of XFDs: comparison of singular instances (no collections), and comparing individual paths rather than tree structures. To the best of our knowledge, only XFDs based on tree homomorphism proposed by Hartmann and Link in [39] are capable of expressing constraints like above. We will refer to this class as *HL-XFDs*. In terms of the graph-based XML data model from the previous section, HL-XFDs considers pre-image trees for “tuples” and v -subgraphs for “attributes”. Thus, we can express functional relationships between tree structures and can refer to collections of subtrees (possibly containing multiple occurrences of certain sub-elements). A nice feature of HL-XFDs is that we can efficiently reason about them [44] and also HL-XFDs can be discovered with a transversal-based approach [88].

Alas, HL-XFDs only considers value-equality for deciding “agreement”. Although we are able to talk about equality of subtrees, we cannot make any statement about the *number of occurrences* of any particular subtree. For example, HL-XFDs cannot express that

Constraint 2. The entry together with the information about the format and section determines the category.

In other words, the constraint states that no entry has multiple categories with the same format and section. Such a functional dependency expresses a kind of uniqueness criteria that allows us to identify specific nodes within the data tree. This is particularly useful when developing syntactic characterisation for the absence of data redundancy - an important indicator of well-designed schema. This observation has lead us to consider a simple extension to HL-XFDs: use v -properties as “attributes”, where v -properties constitute v -subgraphs and v -ancestors:

Definition 1.3 (v -properties). Given a node v of XML schema tree T , elements of $\tilde{S}_T(v) \cup \tilde{A}_T(v)$ are called v -properties of T .

The straight-forward addition of v -ancestors yields strictly more expressiveness by allowing us to talk about the context in which certain subtrees occur. But we will also see that the extension results in rather natural notions of data redundancy for characterising “good” XML schemas, without diminishing our ability to efficiently reason about the class of XFDs. We discuss XML database design as one potential application of pXFDs, but it is by no means a major focus for the thesis. We note that numerical constraints have been proposed for specifying constraints about how frequently a structures occur in a selected context [41]. A study of the interaction between numerical constraints and HL-XFDs may also lead to alternative directions for XML database design.

1.3 XML Functional Dependencies with Properties

In this section, we formalise and exemplify the new class of pXFDs which generalises HL-XFDs. As mentioned previously we will use the pre-image trees of v as “tuples”, while v -properties will be used as “attributes”. To explain when two pre-image trees “agree” on a v -property we need to state what the projection of a pre-image tree to a v -property is. Intuitively projecting to a v -property yields either nodes or XML trees depending on whether we consider a v -ancestor or v -subgraph, and correspondingly, the ways in which we compare the projections are different depending on whether we have nodes or XML trees. In the subsequent pair of definitions, we treat v -subgraphs and v -ancestors as different cases. However, we use the same notation for both cases in order to also be able to talk about projection and agreement on v -properties in a general sense.

Our notions of projection and “agreement” are defined on the basis of tree homomorphism. Given two XML trees T and T' , we say that they are isomorphic if there is a homomorphism $\phi : V_{T'} \rightarrow V_T$ which is bijective and ϕ^{-1} is a homomorphism. In particular, we call such a mapping ϕ an *isomorphism*. A subgraph U of T_1 is a *subcopy* of T_2 if U is isomorphic to some subgraph of T_2 .

Definition 1.4 (projection). Let T be an XML schema tree and T' a T -compatible data tree. Given a v -property X , the *projection of T' to X* , denoted by $T'|_X$, is:

- the set of all pre-images of X in T' , if X is a v -ancestor of T , or
- the union of all subcopies of X in T' , if X is a v -subgraph of T . □

In the literature, two notions of agreement are popular when formalising XML integrity constraints: value-equality and node-equality. *Value-equality* (often denoted by $=_v$) compares tree structures and values at the leaves, whereas *node-equality* compares simply the node ids. We will apply a type of value-equality when comparing projection to v -subgraphs and node-equality when comparing projection to v -ancestors.

Definition 1.5 (property-equality, \doteq). *Property-equality* denoted by (\doteq) holds as follows:

- Two sets p, q of nodes in an XML data tree T' are *property-equal* if and only if $p = q$.
- Two XML data trees p, q are *property-equal* if and only if there exists a valuation-preserving isomorphism $\phi : V_p \rightarrow V_q$ between p and q .

Notations: In the previous definition, and throughout the thesis, we use “=” to refer to an equality comparison based on identity. For example, $X = Y$ if and only if $X \subseteq Y$ and $Y \supseteq X$ for sets X, Y but for nodes n, m contained in some XML tree $n = m$ if and only if n, m are the same node. \square

We now define the class of XML functional dependency with properties.

Definition 1.6 (pXFDs). Let T be an XML schema tree. Given a node v in T , an *XML functional dependency with v -properties* (pXFD) over T is an expression $v : \mathcal{X} \rightarrow \mathcal{Y}$ where both \mathcal{X} and \mathcal{Y} are sets of v -properties of T . Herein, v is referred to as the *target*, \mathcal{X} as the LHS and \mathcal{Y} as the RHS. A pXFD with a singleton RHS is said to be *singular*.

A T -compatible data tree T' *satisfies* $v : \mathcal{X} \rightarrow \mathcal{Y}$, written as $\models_{T'} v : \mathcal{X} \rightarrow \mathcal{Y}$, if and only if for any two pre-image trees $p_1, p_2 \in P_{T'}(v)$ we have $p_1|_X \doteq p_2|_X$ for all $X \in \mathcal{X}$ imply $p_1|_Y \doteq p_2|_Y$ for all $Y \in \mathcal{Y}$. Equivalently, we also say $v : \mathcal{X} \rightarrow \mathcal{Y}$ *holds* in T' . \square

It is obvious that every finite set of pXFDs is finitely satisfiable: we can choose any compatible data tree with at most one pre-image tree of v .

Notations: Let us clarify some further notations. Recall that with singleton subgraphs we sometimes simplify notations by omitting outer set parenthesis. We will *not* omit outer set parenthesis for sets of v -properties. More specifically, a (possibly empty) of v -property is *always* enclosed in set parenthesis. We will denote the empty v -subgraph of T by \emptyset , as opposed to $\{\}$ which always denotes the empty set of v -properties.

In words, we say “pre-image trees p_1, p_2 *agree on* X ” to mean $p_1|_X \doteq p_2|_X$. Likewise, we say “pre-image trees p_1, p_2 *differ on* X ” to mean $p_1|_X \not\doteq p_2|_X$. \square

Next we illustrate the expressiveness of pXFDs with some sample constraints.

1.3.1 Examples Illustrating Expressiveness of pXFDs

We consider some constraints on the dance school data that can be specified as pXFDs. All examples are based on the schema tree T_{dance} (see Figure 1.4) and satisfaction of the constraints is evaluated over the data tree T'_{dance} (see Figure 1.3). Assume a partnering always consists of a girl and a boy.

Constraint 3. In any class, a person may be assigned at most one partner.

$$\begin{aligned} v_{partners} : \{v_{class}, \mathbf{boy}\} &\rightarrow \{\mathbf{girl}\} && (pXFD1) \\ v_{partners} : \{v_{class}, \mathbf{girl}\} &\rightarrow \{\mathbf{boy}\} && (pXFD2) \end{aligned}$$

$pXFD1$ states that whenever two pre-image trees of $v_{partners}$ are grouped under the same v_{class} node (e.g. pre-image trees identified by nodes i_{22} and i_{31}) and they agree on \mathbf{boy}

then they must also agree on `girl`. And *pXFD2* similarly states that if two pre-image trees of $v_{partners}$ are grouped under the same v_{class} node and they agree on `boy` then they also agree on `girl`. When we examine the data tree, it is easy to verify that, in the Ballroom class, Mickey dances with two different girls and one of his partners Minnie and Daisy also dance with two different boys. Therefore both pXFDs do not hold in T'_{dance} .

Without the v -ancestor v_{class} we get the pXFD

$$v_{partners} : \{\mathbf{boy}\} \rightarrow \{\mathbf{girl}\} \quad (pXFD3)$$

which expresses that every boy dances with the same girl, whatever class he is attending. Consider pre-image trees of $v_{partners}$ identified by node i_9 and i_{31} . One shows Mickey and Minnie as partners in the Latin class while the other shows Mickey and Daisy as partners in the Ballroom class. Thus *pXFD3* does not hold in T'_{dance} . Note that this is not surprisingly since pXFD1 already does not hold.

Remark 1.7. When included in the LHS of a pXFD, v -ancestors serve to establish a *context* for evaluating the property-equality of projections to v -subgraphs. The default context is the root, this is implicit for pXFDs with no v -ancestors in the LHS, as we will formally see later. Obviously, pXFDs have a more general context when their LHS contains v -ancestors that are closer to the root. \square

Constraint 4. Every session has a unique number.

$$v_{session} : \{\mathbf{number}\} \rightarrow \{v_{session}\} \quad (pXFD4)$$

This pXFD states that no two $v_{session}$ nodes can agree on `number`. Nodes i_6 and i_{19} witness the violation of this pXFD (i.e., both the Latin and Ballroom class have session number “1”). However the data tree satisfies

$$v_{session} : \{v_{class}, \mathbf{number}\} \rightarrow \{v_{session}\} \quad (pXFD5)$$

That is, within each class, every session has a unique number.

Remark 1.8. With the target node on the RHS, we can express the uniqueness of nodes and therefore a kind of *key constraint*. With both v -ancestors in the LHS and the target node in the RHS we can express a *relative key constraint*. However, due to a difference in what is compared and how, this notion of keys does not coincide with existing XML key notions (e.g., XML Schema keys [96] or Buneman keys [12, 13] or Yu and Jagadish keys [102, 103] or see [38] for a survey). \square

Constraint 5. A boy and girl can partner up together for at most one class.

$$v_{partners} : \{\{\mathbf{boy}, \mathbf{girl}\}\} \rightarrow \{v_{class}\} \quad (pXFD6)$$

The satisfaction of *pXFD6* would tell us that any two pre-images of $v_{partners}$, which agree on subgraph `{boy, girl}`, are located under the same pre-image of v_{class} . In other words, only $v_{partners}$ nodes sharing the same v_{class} node in the data tree may agree on `{boy, girl}`. The pXFD does not hold in T'_{dance} because for example Mickey and Minnie partner up in both the Latin and Ballroom class.

Remark 1.9. With v -ancestors other than the target in the RHS, we can express a *grouping or classification* of pre-image trees. \square

As a final example, the two constraints about photography competitions from Section 1.2 can be expressed by the pXFDs:

$$\begin{aligned} v_{entry} &: \{\{\text{format, section, noOfImg}\}\} \rightarrow \{\text{fees}\} \\ v_{category} &: \{v_{entry, \text{format, section}}\} \rightarrow \{v_{category}\} \end{aligned}$$

1.3.2 Implication and Derivation of pXFDs

A set of pXFD Σ *implies* a pXFD σ , denoted by $\Sigma \models \sigma$, if and only if every T -compatible data tree which satisfies all pXFDs in Σ also satisfies σ . Furthermore, Σ is said to *imply* σ *in the world of two- v -pre-image data trees* if and only if every T -compatible data tree containing precisely two pre-image trees of v that satisfies all pXFDs in Σ also satisfies σ . By Σ^* we represent the set of all pXFDs which are implied by Σ .

An inference rule for pXFDs is a statement:

$$\frac{\langle \text{premises} \rangle}{\langle \text{conclusions} \rangle} \langle \text{pre-conditions} \rangle$$

where the premises and conclusions are sets of pXFDs and the pre-conditions contains properties which must hold in addition to the premises in order for the rule to be applicable in deriving the conclusions. Let \mathfrak{F} be a set of inference rules for pXFDs. We say Σ *derives* σ with \mathfrak{F} , denoted $\Sigma \vdash_{\mathfrak{F}} \sigma$, if and only if there is a sequence of pXFDs ending with σ such that every pXFDs in the sequence belongs to Σ or can be concluded from some pXFDs which occur before it in the sequence by applying some inference rule from \mathfrak{F} . By $\Sigma_{\mathfrak{F}}^+$ we denote the set of all pXFDs which can be derived from Σ with \mathfrak{F} . When \mathfrak{F} is clear from the context then we may simply writes Σ^+ .

An inference rule for pXFDs is *sound* if and only if for every XML schema tree T and every T -compatible data tree, it is the case that every valid application of the inference rule results only in pXFDs which are satisfied. In other words, we can only derive pXFDs which are implied. A set of inference rules \mathfrak{F} is *sound* if and only if all its inference rules are sound, i.e., $\Sigma^+ \subseteq \Sigma^*$. A set of inference rules is *complete* if and only if every pXFDs which can be implied is derivable by applying only inference rules in the given system, i.e., $\Sigma^* \subseteq \Sigma^+$. A sound and complete system of inference rules is called an *axiomatisation*. The existence of an axiomatisation enables us to talk about Σ^* and Σ^+ interchangeably.

Two systems $\mathfrak{F}_1, \mathfrak{F}_2$ of inference rules for pXFDs are said to be *equivalent* if and only if for every set Σ of pXFDs it is the case that $\Sigma_{\mathfrak{F}_1}^+ = \Sigma_{\mathfrak{F}_2}^+$. That is, any pXFD derivable by using inference rules from one system must also be derivable by using inference rules from the other system.

1.3.3 Observations About Property-Equality

The following observations about property-equality are foundational for our investigations into pXFD implications.

Trivially, any two pre-image trees will always agree on the empty v -subgraph and the root node, while any two distinct pre-image trees of v will always differ on at least the target node v . Moreover, if two pre-image trees of v agree on v then they are in fact the same pre-image tree and therefore will also agree on all v -properties. This last observation is shown more formally in Remark 2.39.

It is also easy to see from the definition of property-equality that agreement on a particular v -property may imply agreement on some other v -properties, and similarly, difference on a particular v -property may imply difference on some other v -properties. Understanding cases where we can make such deductions is useful for checking the satisfaction of pXFDs and allows us to identify an important equally-expressive sub-class of pXFDs in the next section.

Because XML trees are rooted, it is clear that two pre-image trees which agree on some v -ancestor n must also agree on all ancestors of n . The opposite direction, that is, whether the two pre-image trees agree on some descendant of n , is not always true. For example, consider the pre-image trees of $v_{partners}$ identified by node i_{22} and i_{31} from the dance school data tree. Both pre-image trees agree on v_{class} but differ on $v_{session}$. On the other hand, two pre-image trees sharing the same $v_{session}$ node always share their $v_{attendees}$ node. What differentiates these two scenarios? Recall that T -compatibility requires satisfaction of the occurrence restrictions imposed by frequencies. This means that, for any two v -ancestors which are connected by simple paths, property-equality on one implies property-equality on the other. These observations about v -ancestors are summarised in the following remark:

Remark 1.10. Let $n, m \in \check{\mathbb{A}}_T(v)$ be v -ancestors and $p_1, p_2 \in P_{T'}(v)$ be two pre-image trees of v , then $p_1|_n \doteq p_2|_n$ implies $p_1|_m \doteq p_2|_m$ if

- m is an ancestor of n or
- m is a simple descendant of n .

Because projection to v -subgraphs result in XML trees which are compared on the basis of tree isomorphism, it is also clear that two pre-image trees which agree on some v -subgraph X must also agree on all v -subgraphs contained in X . But as observed previously, generally two pre-image trees may agree on two v -subgraphs X, Y and still differ on the union v -subgraph $X \cup Y$. Under certain conditions, we are still able to say that two pre-image trees which agree on v -subgraphs X, Y *must* also agree on $X \cup Y$. Consider, for example, the $v_{partners}$ -walks \mathbf{Bo} and \mathbf{Gi} . It is not possible to find, in any T_{dance} -compatible data tree, two pre-image trees p_1, p_2 of $v_{partners}$ such that $p_1|_{\mathbf{Bo}} \doteq p_2|_{\mathbf{Bo}}$ and $p_1|_{\mathbf{Gi}} \doteq p_2|_{\mathbf{Gi}}$ but $p_1|_{\{\mathbf{Bo}, \mathbf{Gi}\}} \not\doteq p_2|_{\{\mathbf{Bo}, \mathbf{Gi}\}}$. As reflected in Lemma 1.14, the necessary condition turns out to be precisely the following notion of v -subgraphs being v -reconcilable.

Definition 1.11 (*v*-reconcilable). Two distinct *v*-subgraphs X, Y are called *v-reconcilable* if and only if X contains every w -walk in Y or Y contains every w -walk in X , whenever X, Y share some arc (u, w) of frequency other than ? and 1 where w is a proper descendant of v . \square

Example 1.12. Subgraphs `boy` and `girl` are $v_{partners}$ -reconcilable but, not v_{class} -reconcilable because they are distinct walks which both contain node $v_{partners}$ - a proper descendant of v_{class} whose incoming arc has frequency other than ? and 1. Also `boy` and `girl` are both v_{class} -reconcilable with $\{\text{boy, girl, semester}\}$ and $\{\text{topic}\}$. \square

Remark 1.13. If X, Y are *v*-reconcilable *v*-subgraphs then any two *v*-subgraphs X', Y' contained in X and Y , respectively, are also *v*-reconcilable. If X, Y, Z are mutually *v*-reconcilable *v*-subgraphs then so are X and $Y \cup Z$. \square

Lemma 1.14. *Let X, Y be two *v*-subgraphs. X, Y are *v*-reconcilable if and only if for every T -compatible XML data tree T' for all pre-image trees $p_1, p_2 \in P_{T'}(v)$ it is true that $p_1|_X \doteq p_2|_X$ and $p_1|_Y \doteq p_2|_Y$ imply $p_1|_{X \cup Y} \doteq p_2|_{X \cup Y}$.*

Proof. The statement obviously holds when v is simple because $P_{T'}(v)$ would contain at most one pre-image tree. Therefore suppose v is not simple.

(\Rightarrow) Consider any two pre-image trees $p_1, p_2 \in P_{T'}(v)$ from some arbitrary T -compatible data tree T' where $p_1|_X \doteq p_2|_X$ and $p_1|_Y \doteq p_2|_Y$. This means that there exists two valuation-preserving isomorphisms: $\phi_X : V_{p_1|_X} \rightarrow V_{p_2|_X}$ and $\phi_Y : V_{p_1|_Y} \rightarrow V_{p_2|_Y}$. Since X, Y are *v*-reconcilable, we infer that $p_1|_X$ shares with $p_1|_Y$ every arc which $p_2|_X$ shares with $p_2|_Y$, particularly even those arcs with frequency other than ? and 1. Therefore combining ϕ_X and ϕ_Y in the obvious way results in a valuation-preserving isomorphism between the node sets of $p_1|_{X \cup Y}$ and $p_2|_{X \cup Y}$.

(\Leftarrow) For the contrapositive proof we assume that X, Y are not *v*-reconcilable and provide a construction of a T -compatible data tree T' with two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_X \doteq p_2|_X$ and $p_1|_Y \doteq p_2|_Y$ but $p_1|_{X \cup Y} \not\doteq p_2|_{X \cup Y}$. Figure 1.6 depicts the structure of the data tree resulting from the construction that follows.

Because X, Y are not *v*-reconcilable, there is some arc (u, w) of frequency other than ? and 1 such that X contains some w -walk(s) not contained in Y or Y contains some w -walk(s) not contained in X , where w is a proper descendant of v . Let us denote by X' and Y' respectively, the set of w -walk(s) contained in X but not in Y and, the set of w -walk(s) contained in Y but not in X .

Let Z be the smallest *v*-subgraph containing $X' \cup Y'$ and any additional path with arcs of frequency 1 or + that is connected to $X' \cup Y'$. The data tree will be constructed from four copies $p_1^1, p_1^2, p_2^1, p_2^2$ of Z . Now let us specify a valuation mapping for each copy. For p_1^1 , assign the value “0” to all leaves. For p_2^1 , assign the value “1” to all leaves in $p_2^1|_{X' \cup Y'}$ and “0” to all remaining leaves. Assign values to p_2^2, p_1^2 as follows: leaves in $p_2^2|_{X'}, p_2^2|_{Y'}$ are assigned the value “0”; leaves in $p_2^2|_{Y'}, p_2^2|_{X'}$ are assigned the value “1”; and all other leaves are assigned “0”. Note that the assignment yields $p_2^1|_{X'} \doteq p_1^1|_{X'}$, $p_2^1|_{Y'} \doteq p_1^1|_{Y'}$, $p_2^2|_{X'} \doteq p_1^2|_{X'}$, $p_2^2|_{Y'} \doteq p_1^2|_{Y'}$.

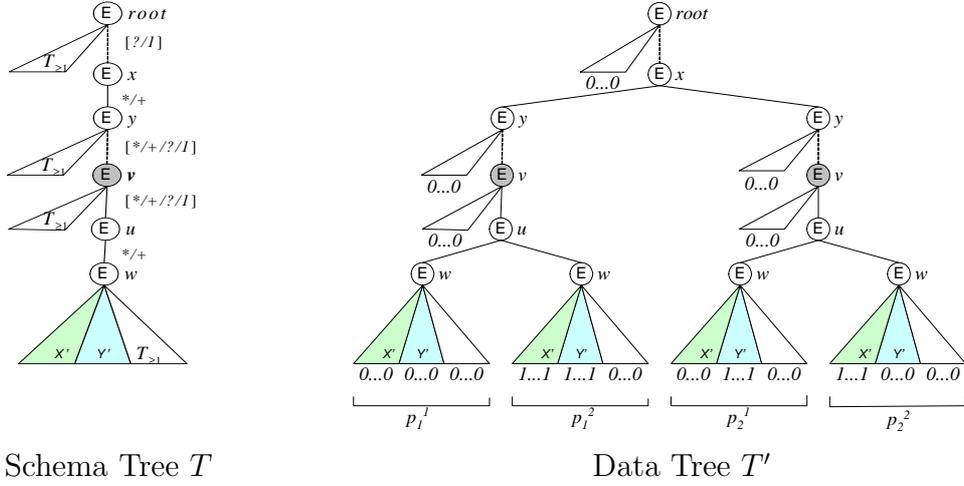


Figure 1.6: Counter-example construction when X, Y are not v -reconcilable.

We construct a pre-image tree p_i of v by merging p_i^1, p_i^2 such that they share in p_i all nodes except their pre-images of w and its descendants (for $i = 1, 2$). This is possible because (u, w) is of frequency other than $?$ and 1 , and in addition, p_i^1 and p_i^2 may only differ on v -subgraphs contained in $X' \cup Y' \subseteq T(w)$ (for $i = 1, 2$). Finally to construct T' we merge p_1, p_2 such that their respective pre-images of v remain distinct. For conciseness, let (x, y) be the first arc on the rooted path to v with frequency other than 1 and $?$. We force p_1, p_2 to share every node except their pre-images of y and its descendants. This is possible since v is not simple. It is easy to see that T' is T -compatible and we have $p_1|_X \doteq p_2|_X$ and $p_1|_Y \doteq p_2|_Y$ but $p_1|_{X \cup Y} \neq p_2|_{X \cup Y}$. \square

Remark 1.15. Let $X, Y \in \check{S}_T(v)$ be v -subgraphs and $p_1, p_2 \in P_{T'}(v)$ be two pre-image trees of v , then

- $p_1|_X \doteq p_2|_X$ implies $p_1|_Y \doteq p_2|_Y$ if Y is contained in X , or
- $p_1|_X \doteq p_2|_X$ and $p_1|_Y \doteq p_2|_Y$ implies $p_1|_{X \cup Y} \doteq p_2|_{X \cup Y}$ if X, Y are v -reconcilable.

1.3.4 Canonical pXFDs

The number of subsets of v -properties is exponential in the number of v -properties, with the number of v -subgraphs being exponential in the number of v -walks. Thus, we are looking at a rather large number of syntactically valid pXFDs as per the definition.

Example 1.16. Consider a simple XML schema tree T having v -ancestors n_1, \dots, n_k, v , where only arc (n_k, v) has frequency other than $?$ and 1 . Moreover, suppose T contains v -walks B_1, \dots, B_j , all with simple path between v and the leaf. There are 2^j possible v -subgraphs and thus $(2^{(2^j+k+1)})^2$ syntactically valid pXFDs over T . \square

Fortunately, it is sufficient to consider a subclass of pXFDs without loss of generality. We propose a refinement for transforming any given pXFD into a canonical pXFD and show that a pXFD is satisfied precisely when the corresponding canonical pXFD is satisfied. In many cases, the number of canonical pXFDs can be significantly less than the number of pXFDs in general. A pXFD from Example 1.16 can include any of $2^j + k + 1$ v -properties but canonical pXFDs can only feature at most $j + 2$ of these v -properties. In the thesis, canonical pXFDs help to reduce the number of pXFDs which must be explored for dependency discovery, as well as simplifying the correspondence with a fragment of propositional logic.

Refinement can be seen as the process of translating non-essential v -properties into essential ones. We say that some v -properties are *non-essential* because their values can always be determined from knowing the values of other v -properties. Those v -properties which we do not explicitly specify to be non-essential are deemed to be *essential*. The result of refining both the LHS and RHS of a pXFD then gives a pXFD in canonical form.

Essential v -properties

The set of essential v -properties constitutes: the set of essential v -subgraphs and the set of essential v -ancestors. We proceed to define these two sets next.

The definition of essential v -ancestors is motivated by the observation that agreement on a v -ancestor n implies agreement on all simple descendants of n .

Definition 1.17 (essential v -ancestor). Let $\mathbf{E}_T^{\check{\Delta}}(v)$ be the set consisting of the root r_T and all other v -ancestors whose incoming arc has frequency other than ? and 1. Members of $\mathbf{E}_T^{\check{\Delta}}(v)$ are called *essential v -ancestors*. \square

Remark 1.18. The root is the only essential v -ancestor which is simple. Every essential v -ancestor has only itself as a simple descendant. So if target node v is a simple node then the root is the only essential v -ancestor. \square

The definition of essential v -subgraphs is motivated by the observation that agreement on v -reconcilable v -subgraphs X and Y implies agreement on their union v -subgraph $X \cup Y$.

Definition 1.19 (essential v -subgraph). Let $\mathbf{E}_T^{\check{\Sigma}}(v)$ be a subset of $\check{\Sigma}_T(v)$ defined inductively as follows:

- The empty v -subgraph and every v -walk in $\check{\Sigma}_T(v)$ is a member of $\mathbf{E}_T^{\check{\Sigma}}(v)$.
- If two v -subgraphs $X, Y \in \mathbf{E}_T^{\check{\Sigma}}(v)$ are not v -reconcilable then $X \cup Y \in \mathbf{E}_T^{\check{\Sigma}}(v)$.
- Nothing else is a member of $\mathbf{E}_T^{\check{\Sigma}}(v)$.

The members of $\mathbf{E}_T^{\check{\Sigma}}(v)$ are called *essential v -subgraphs*. \square

It turns out that identifying and enumerating all essential v -subgraphs is rather simple since every essential v -subgraph is contained in some v -unit. The following pair of lemmas formally prove this statement.

Lemma 1.20. *Two v -subgraphs X, Y are v -reconcilable if and only if for every v -unit U we have that $X \cap U \subseteq Y \cap U$ or $Y \cap U \subseteq X \cap U$.*

Proof. (\Leftarrow) We show the contra-positive, so suppose that X, Y are not v -reconcilable. That is X, Y share some arc (u, w) of frequency other than $?$ and 1 where w is a proper descendant of v . Moreover, X does not contain every w -walk in Y , nor does Y contain every w -walk in X . That is to say, there is some w -walk contained in X which is not contained in Y and vice versa. All w -walks belong to same v -unit, say U , which then means that $X \cap U \not\subseteq Y \cap U$ and $Y \cap U \not\subseteq X \cap U$.

(\Rightarrow) Suppose for some v -unit U we have $X \cap U \not\subseteq Y \cap U$ and $Y \cap U \not\subseteq X \cap U$. It follows that U contains at least two v -walks and there exists some proper descendant w of v whose incoming arc is the only arc in the path from v to w having frequency other than $?$ and 1 . Moreover $X \cap U$ is the set of all w -walks contained in X , $Y \cap U$ is the set of all w -walks contained in Y and, neither is contained in the other. Thus X, Y are not v -reconcilable. \square

Lemma 1.21. *A v -subgraph is essential if and only if it is contained in some v -unit.*

Proof. The statement is obvious for the empty v -subgraph.

(\Leftarrow) Consider a non-empty v -subgraph X contained in some v -unit U . Every v -walk in X belongs to $\mathbf{E}_T^{\mathcal{S}}(v)$ and is contained in U . By Lemma 1.20 they are pair-wise not v -reconcilable. It follows that X , being the union of all its v -walks, is an essential v -subgraph.

(\Rightarrow) From the other direction, it is clear that v -units are essential v -subgraph. Consider then any v -unit U . From Lemma 1.20, we can infer that U is v -reconcilable with every v -walk B which belong to a v -unit other than U . This means that every v -unit is an essential v -subgraph which is maximal with respect to subgraph containment and therefore a non-empty essential v -subgraph must be contained in some v -unit of T . \square

Remark 1.22. If X is an essential v -subgraph then every v -subgraph contained in X is also an essential v -subgraph. Furthermore, v -units induce a partition of the family of all essential v -subgraphs. \square

Refinement ϑ

We can now define a refinement of pXFDs in which all non-essential v -properties are mapped to essential ones. Validity of the proposed refinement is justified by showing that a pXFD is satisfied precisely when its refinement is satisfied. Similar to projection and property-equality, refinement of sets of v -subgraphs and refinement of sets of v -ancestors constitute how pXFDs are refined.

For any v -ancestor n , its corresponding essential v -ancestor is its *highest simple ancestor*, i.e., the single simple ancestor of n which does not have an incoming arc of frequency $?$ or 1 . Furthermore, there is no need for a set to contain more than one essential v -ancestor because if two pre-image trees share the pre-image of a particular v -ancestor

n then they also share the pre-image of all ancestor of n . That is, in any given set S of v -ancestors we can just consider the *lowest contained v -ancestor*, denoted by $\text{lca}(S)$, which is the single v -ancestor in S which does not have any other v -ancestor in S as a proper descendant. These two observations brings us to the following refinement for any given set of v -ancestors.

Definition 1.23 (ϑ for v -ancestors). Let $\mathcal{X}^{\check{\Delta}}$ be a (possibly empty) set of v -ancestors. If $\mathcal{X}^{\check{\Delta}}$ is non-empty then $\vartheta(\mathcal{X}^{\check{\Delta}})$ denotes the singleton set containing the highest simple ancestor of $\text{lca}(\mathcal{X}^{\check{\Delta}})$. If instead $\mathcal{X}^{\check{\Delta}}$ is empty, then $\vartheta(\mathcal{X}^{\check{\Delta}}) = \{\}$. \square

Example 1.24. For XML schema tree T_{dance} , we have the following essential v_{partners} -ancestors: $v_{\text{danceschool}}, v_{\text{class}}, v_{\text{session}}$ and v_{partners} . We obtain the singleton set $\{v_{\text{session}}\}$ if we refine either $S' = \{v_{\text{attendees}}, v_{\text{danceschool}}\}$ or $S = \{v_{\text{attendees}}\}$. \square

The following lemma demonstrates the correctness of the refinement for v -ancestors, that if two pre-image trees agree on the single v -ancestor in $\vartheta(\mathcal{X}^{\check{\Delta}})$ then they agree on every v -ancestor in $\mathcal{X}^{\check{\Delta}}$, and vice versa.

Lemma 1.25. *Let $\mathcal{X}^{\check{\Delta}}$ be a set of v -ancestors and T' a T -compatible data tree. For any two pre-image trees $p_1, p_2 \in P_{T'}(v)$ we have $p_1|_n \doteq p_2|_n$ for all $n \in \mathcal{X}^{\check{\Delta}}$ if and only if $p_1|_{n'} \doteq p_2|_{n'}$ for $\{n'\} = \vartheta(\mathcal{X}^{\check{\Delta}})$.*

Proof. (\Rightarrow) Suppose there are two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_n \doteq p_2|_n$ for all $n \in \mathcal{X}^{\check{\Delta}}$. Because $\text{lca}(\mathcal{X}^{\check{\Delta}}) \in \mathcal{X}^{\check{\Delta}}$ we have $p_1|_{\text{lca}(\mathcal{X}^{\check{\Delta}})} \doteq p_2|_{\text{lca}(\mathcal{X}^{\check{\Delta}})}$. Then we have $p_1|_{n'} \doteq p_2|_{n'}$ for $\{n'\} = \vartheta(\mathcal{X}^{\check{\Delta}})$ because n' is an ancestor of $\text{lca}(\mathcal{X}^{\check{\Delta}})$.

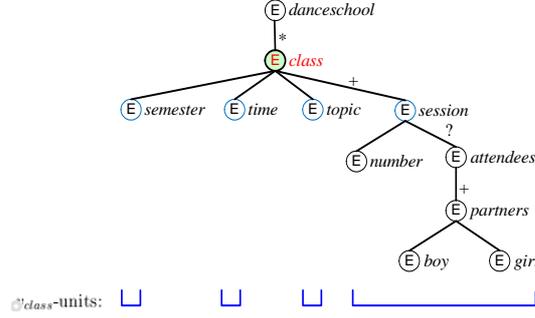
(\Leftarrow) Suppose there are two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_{n'} \doteq p_2|_{n'}$ for $\{n'\} = \vartheta(\mathcal{X}^{\check{\Delta}})$. Then $p_1|_{\text{lca}(\mathcal{X}^{\check{\Delta}})} \doteq p_2|_{\text{lca}(\mathcal{X}^{\check{\Delta}})}$ because $\text{lca}(\mathcal{X}^{\check{\Delta}})$ is a simple descendant of n' and T' is T -compatible. Moreover, every v -ancestor in $\mathcal{X}^{\check{\Delta}}$ is ancestor of $\text{lca}(\mathcal{X}^{\check{\Delta}})$ hence $p_1|_n \doteq p_2|_n$ for all $n \in \mathcal{X}^{\check{\Delta}}$. \square

Corollary 1.26. *Let $\mathcal{X}^{\check{\mathcal{S}}}, \mathcal{Y}^{\check{\mathcal{S}}}$ be sets of v -subgraphs and $\mathcal{X}^{\check{\Delta}}, \mathcal{Y}^{\check{\Delta}}$ be sets of v -ancestors. T -compatible data tree T' satisfies $v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\Delta}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\Delta}}$ if and only if T' satisfies $v : \mathcal{X}^{\check{\mathcal{S}}} \cup \vartheta(\mathcal{X}^{\check{\Delta}}) \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \vartheta(\mathcal{Y}^{\check{\Delta}})$.*

Proof. Follows from Lemma 1.25 and definition of pXFDs satisfaction. \square

Similarly, we investigate refinement for any given set of v -subgraphs. Since pre-image trees which agree on some v -subgraph will also agree on all v -subgraphs it contains, we can simply consider the maximal one with respect to subgraph containment. Again, it is the case that the set of essential v -subgraphs resulting from the refinement and the set of v -subgraphs being refined are interchangeable with respect to property-equality.

Definition 1.27 (ϑ for v -subgraphs). For a set $\mathcal{X}^{\check{\mathcal{S}}}$ of v -subgraphs consider the set of all essential v -subgraphs that are contained in some member of $\mathcal{X}^{\check{\mathcal{S}}}$. We use $\vartheta(\mathcal{X}^{\check{\mathcal{S}}})$ to denote the subset of all members of this set which are maximal with respect to subgraph containment. If $\mathcal{X}^{\check{\mathcal{S}}} = \{\}$ then let $\vartheta(\mathcal{X}^{\check{\mathcal{S}}}) = \{\}$. \square

Figure 1.7: Dance school schema tree showing its four v_{class} -units

Example 1.28. The dance school schema tree has four v_{class} -units, these are shown in Figure 1.7. For the set

$$\mathcal{X}^{\tilde{\mathcal{S}}} = \{\{\text{semester, number, boy}\}, \{\text{boy, girl}\}, \text{girl, time}\}$$

of v_{class} -subgraph we get

$$\vartheta(\mathcal{X}^{\tilde{\mathcal{S}}}) = \{\text{semester}, \{\text{number, boy}\}, \{\text{boy, girl}\}, \text{time}\}$$

Essential v_{class} -subgraphs contained in some member of $\mathcal{X}^{\tilde{\mathcal{S}}}$ but not in $\vartheta(\mathcal{X}^{\tilde{\mathcal{S}}})$ are v_{class} -walks: boy, girl and number. \square

Lemma 1.29. *Let $\mathcal{X}^{\tilde{\mathcal{S}}}$ be a set of v -subgraphs and T' a T -compatible data tree. For any two pre-image trees $p_1, p_2 \in P_{T'}(v)$ we have $p_1|_X \doteq p_2|_X$ for all $X \in \mathcal{X}^{\tilde{\mathcal{S}}}$ if and only if $p_1|_{X'} \doteq p_2|_{X'}$ for all $X' \in \vartheta(\mathcal{X}^{\tilde{\mathcal{S}}})$.*

Proof. (\Rightarrow) Observe that two pre-image trees which agree on some v -subgraph X also agree on all v -subgraphs contained in X . The statement then follows from every v -subgraph $X' \in \vartheta(\mathcal{X}^{\tilde{\mathcal{S}}})$ being contained in some v -subgraph $X \in \mathcal{X}^{\tilde{\mathcal{S}}}$.

(\Leftarrow) Suppose $p_1|_{X'} \doteq p_2|_{X'}$ for all $X' \in \vartheta(\mathcal{X}^{\tilde{\mathcal{S}}})$ but $p_1|_X \neq p_2|_X$ for some $X \in \mathcal{X}^{\tilde{\mathcal{S}}}$. Members of $\vartheta(X) = \{X_1, \dots, X_n\}$ are pair-wise v -reconcilable according to Lemma 1.20. Also, every v -subgraph belonging to $\vartheta(X)$ either belongs to $\vartheta(\mathcal{X}^{\tilde{\mathcal{S}}})$ or is contained in some v -subgraph belonging to $\vartheta(\mathcal{X}^{\tilde{\mathcal{S}}})$. It follows from the initial assumption that $p_1|_{X_i} \doteq p_2|_{X_i}$ for $i = 1, \dots, n$ and, by Lemma 1.14 also $p_1|_X \doteq p_2|_X$, a contradiction. \square

Corollary 1.30. *Let $\mathcal{X}^{\tilde{\mathcal{S}}}, \mathcal{Y}^{\tilde{\mathcal{S}}}$ be sets of v -subgraphs and $\mathcal{X}^{\tilde{\mathcal{A}}}, \mathcal{Y}^{\tilde{\mathcal{A}}}$ be sets of v -ancestors. T -compatible data tree T' satisfies $v : \mathcal{X}^{\tilde{\mathcal{S}}} \cup \mathcal{X}^{\tilde{\mathcal{A}}} \rightarrow \mathcal{Y}^{\tilde{\mathcal{S}}} \cup \mathcal{Y}^{\tilde{\mathcal{A}}}$ if and only if T' satisfies $v : \vartheta(\mathcal{X}^{\tilde{\mathcal{S}}}) \cup \mathcal{X}^{\tilde{\mathcal{A}}} \rightarrow \vartheta(\mathcal{Y}^{\tilde{\mathcal{S}}}) \cup \mathcal{Y}^{\tilde{\mathcal{A}}}$.*

Proof. Follows from Lemma 1.29 and definition of pXFD satisfaction. \square

The refinement of a set of v -properties combines the refinement of v -ancestors (i.e., Definition 1.23) with the refinement of v -subgraphs (i.e., Definition 1.27).

Definition 1.31 (ϑ for v -properties). For a set \mathcal{X} of v -properties, $\vartheta(\mathcal{X}) = \vartheta(\mathcal{X}^{\check{S}}) \cup \vartheta(\mathcal{X}^{\check{A}})$ where $\mathcal{X}^{\check{S}} = \mathcal{X} \cap \check{S}_T(v)$ and $\mathcal{X}^{\check{A}} = \mathcal{X} \cap \check{A}_T(v)$ are (possibly empty) subsets of \mathcal{X} containing all v -subgraphs and v -ancestors in \mathcal{X} , respectively. We call $\vartheta(\mathcal{X})$ the *refinement* of \mathcal{X} .

Definition 1.32 (canonical pXFD). A pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$ is a *canonical pXFD* if and only if $\mathcal{X} = \vartheta(\mathcal{X})$ and $\mathcal{Y} = \vartheta(\mathcal{Y})$.

Thus every pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$ has a corresponding canonical pXFD $v : \vartheta(\mathcal{X}) \rightarrow \vartheta(\mathcal{Y})$, whose LHS and RHS are subsets of essential v -properties. Furthermore, a pXFD is satisfied precisely when the associated canonical pXFD is satisfied.

Proposition 1.33. *Let \mathcal{X}, \mathcal{Y} be sets of v -properties. A T -compatible data tree T' satisfies the pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$ if and only if T' satisfies the canonical pXFD $v : \vartheta(\mathcal{X}) \rightarrow \vartheta(\mathcal{Y})$*

Proof. Follows from Corollary 1.26 and Corollary 1.30. □

Remark 1.34. Notice that $\vartheta(\vartheta(\mathcal{X})) = \vartheta(\mathcal{X})$. This yields an additional characterisation for essential v -properties: A v -property X is essential if and only if $\vartheta(\{X\}) = \{X\}$. □

Core Sets and P-subsumption Order

When we consider only canonical pXFDs, the potential LHSs/RHSs are core sets as defined in the following:

Definition 1.35 (core set). We call a set \mathcal{X} of v -properties a *core set* if $\vartheta(\mathcal{X}) = \mathcal{X}$ and a *non-core set* otherwise. □

In particular, remember that each core set may contain:

- at most one v -ancestor and the v -ancestor is essential, and
- only v -subgraphs which are essential and not pairwise comparable with respect to subgraph containment.

Up till now, we have encountered three orderings: set containment, ordering of v -ancestors according to their ancestor-descendant relationships, and ordering of v -subgraphs according to their containment in one another. An amalgamation of the three different orders results in a surprisingly natural ordering over the family $\mathcal{P}(\check{S}_T(v) \cup \check{A}_T(v))$ of all sets of v -properties. Since core sets are special sets of v -properties, the same ordering can be used for ordering the family of core sets.

Definition 1.36 (p-subsumption \sqsubseteq). The *p-subsumption* order (\sqsubseteq) is defined inductively as follows:

- $X \sqsubseteq Y$, if X, Y are v -subgraph and X is contained in Y
- $n \sqsubseteq m$, if n, m are v -ancestors and n is an ancestor of m

- $\mathcal{X} \sqsubseteq \mathcal{Y}$, if \mathcal{X}, \mathcal{Y} are sets of v -properties and for every $X \in \mathcal{X}$ there exists some $Y \in \mathcal{Y}$ such that $X \sqsubseteq Y$ (analogous to the Hoare ordering of sets).

For $\mathcal{X} \sqsubseteq \mathcal{Y}$ we say that \mathcal{Y} *p-subsumes* \mathcal{X} , or equivalently that \mathcal{X} is *p-subsumed* by \mathcal{Y} . Moreover, if $\mathcal{Y} \not\sqsubseteq \mathcal{X}$ also holds then we say \mathcal{Y} *strictly p-subsumes* \mathcal{X} , and denote this by $\mathcal{X} \sqsubset \mathcal{Y}$. \square

Thus, p-subsumption naturally relates to property-equality:

Remark 1.37. Consider two sets of v -properties \mathcal{X}, \mathcal{Y} . If $\mathcal{Y} \sqsubseteq \mathcal{X}$ then for each T -compatible data tree T' and any two pre-image trees $p_1, p_2 \in P_{T'}(v)$ it is the case that

$$p_1|_X \doteq p_2|_X \text{ for every } X \in \mathcal{X} \text{ implies } p_1|_Y \doteq p_2|_Y \text{ for every } Y \in \mathcal{Y}$$

This follows directly from our observations about property-equality in Section 1.3.3. \square

Because containment, ancestor-descendant relationship and Hoare ordering of sets are all reflexive and transitive, p-subsumption is a pre-order over $\mathcal{P}(\check{S}_T(v) \cup \check{A}_T(v))$. Equally conspicuous is that p-subsumption is not antisymmetric for $\mathcal{P}(\check{S}_T(v) \cup \check{A}_T(v))$. As a counter-example, assume A, B, AB are v -subgraphs, then we have $\{A, B, AB\} \sqsubseteq \{AB\}$ and $\{A, B, AB\} \sqsupseteq \{AB\}$ but the two sets are distinct. For the family of core sets, however, p-subsumption is a partial order:

Proposition 1.38. \sqsubseteq is a partial order over the family of core sets.

Proof. Proof of reflexivity and transitivity is trivial and so, we only show that \sqsubseteq is antisymmetric. Let \mathcal{X} and \mathcal{Y} be two core sets of essential v -properties and suppose $\mathcal{X} \sqsubseteq \mathcal{Y}$ and $\mathcal{Y} \sqsubseteq \mathcal{X}$. If \mathcal{X} is empty then \mathcal{Y} is empty, since only the empty set is p-subsumed by the empty set. Otherwise, both \mathcal{X}, \mathcal{Y} are non-empty and for every $X \in \mathcal{X}$ there is some $Y \in \mathcal{Y}$ such that $X \sqsubseteq Y$. Furthermore, there is some $X' \in \mathcal{X}$ such that $Y \sqsubseteq X'$.

If X is an essential v -subgraph then so are Y and X' . From the transitivity of subgraph containment, X is contained in X' . In fact $X' = X = Y$ because \mathcal{X} only consists of maximal essential v -subgraphs. This proves $X \in \mathcal{X}$ implies $X \in \mathcal{Y}$ for every essential v -subgraph X . By similar argumentation, we can prove $Y \in \mathcal{Y}$ implies $Y \in \mathcal{X}$ for every essential v -subgraph Y . This establishes that \mathcal{X} and \mathcal{Y} contains the same essential v -subgraphs.

If X is an essential v -ancestor then so are Y and X' . From the linear ordering of v -ancestors, X is an ancestor of X' and actually $X' = X$ because there is at most one v -ancestor present in \mathcal{X} . This also implies that $Y = X$. Because \mathcal{Y} also contains at most one v -ancestor, we have that \mathcal{X} and \mathcal{Y} contain the same essential v -ancestor.

The two subcases shows that $\mathcal{X} = \mathcal{Y}$, which completes the proof for antisymmetry. \square

Remark 1.39. The \sqsubseteq -minimal core set is $\{\}$ which is p-subsumed by every core set. The \sqsubseteq -maximal singleton core sets are $\vartheta(\{v\})$ and every set $\{U\}$ where U is a v -unit of T . That is, $\vartheta(\{v\})$ p-subsumes every core set containing a single essential v -ancestor, while $\{U\}$ p-subsumes every singleton core set $\{X\}$ where X is contained in U (i.e., $X \sqsubseteq U$).

The next pair of lemmas identifies some interesting and useful properties of p-subsumption when regarding sets of v -properties. We will revisit these properties when we investigate how to discovery pXFDs from a given XML data tree in Chapter 3.

Lemma 1.40. *For a sets $\mathcal{W}, \mathcal{X}, \mathcal{Y}$ of v -properties:*

1. $\vartheta(\mathcal{X}) \sqsubseteq \mathcal{X}$
2. $\vartheta(\mathcal{X} \cup \mathcal{W}) \sqsubseteq \vartheta(\mathcal{X}) \cup \vartheta(\mathcal{W})$
3. $\vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{Y})$ and $\vartheta(\mathcal{W}) \sqsubseteq \vartheta(\mathcal{Y})$ implies $\vartheta(\mathcal{X} \cup \mathcal{W}) \sqsubseteq \vartheta(\mathcal{Y})$.
4. $\vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{Y})$ implies $\vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{Y}) \cup \mathcal{W}$

Proof.

1. Immediate from Definitions 1.23 and Definitions 1.27, i.e., refinement ϑ .
2. Every essential v -subgraph $X \in \vartheta(\mathcal{X} \cup \mathcal{W})$ is contained in some element $X' \in \mathcal{X} \cup \mathcal{W}$. If $X' \in \mathcal{X}$ then X is contained in some element of $\vartheta(\{X'\}) \sqsubseteq \vartheta(\mathcal{X})$, and similarly if $X' \in \mathcal{W}$. For v -ancestors, observe $\text{lca}(\mathcal{X} \cup \mathcal{W}) \in \{\text{lca}(\mathcal{X}), \text{lca}(\mathcal{W})\}$ and so $\vartheta(\{\text{lca}(\mathcal{X} \cup \mathcal{W})\})$ is p-subsumed by $\vartheta(\mathcal{X})$ or $\vartheta(\mathcal{W})$.
3. From the definition of p-subsumption we have $\vartheta(\mathcal{X}) \cup \vartheta(\mathcal{W}) \sqsubseteq \vartheta(\mathcal{Y})$. The claim then follows from (2) and transitivity of p-subsumption.
4. $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{Y}) \cup \mathcal{W}$ and so $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{Y}) \cup \mathcal{W}$. The claim then follows from the transitivity of p-subsumption. □

Lemma 1.41. *For two sets \mathcal{X}, \mathcal{Y} of v -properties with $\mathcal{X} \sqsubseteq \mathcal{Y}$, we have $\vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{Y})$.*

Proof. The lemma follows from two claims:

- $\mathcal{X}^{\check{S}}, \mathcal{Y}^{\check{S}} \subseteq \check{S}_T(v)$ with $\mathcal{X}^{\check{S}} \sqsubseteq \mathcal{Y}^{\check{S}}$ implies $\vartheta(\mathcal{X}^{\check{S}}) \sqsubseteq \vartheta(\mathcal{Y}^{\check{S}})$
- $\mathcal{X}^{\check{A}}, \mathcal{Y}^{\check{A}} \subseteq \check{A}_T(v)$ with $\mathcal{X}^{\check{A}} \sqsubseteq \mathcal{Y}^{\check{A}}$ implies $\vartheta(\mathcal{X}^{\check{A}}) \sqsubseteq \vartheta(\mathcal{Y}^{\check{A}})$

where $\mathcal{X} = \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}}$ and $\mathcal{Y} = \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$. From Lemma 1.40 we have

$$\vartheta(\mathcal{X}^{\check{S}}) \sqsubseteq \vartheta(\mathcal{Y}^{\check{S}}) \cup \vartheta(\mathcal{Y}^{\check{A}}) \text{ and } \vartheta(\mathcal{X}^{\check{A}}) \sqsubseteq \vartheta(\mathcal{Y}^{\check{S}}) \cup \vartheta(\mathcal{Y}^{\check{A}}),$$

and so

$$\vartheta(\mathcal{X}) = \vartheta(\mathcal{X}^{\check{S}}) \cup \vartheta(\mathcal{X}^{\check{A}}) \sqsubseteq \vartheta(\mathcal{Y}^{\check{S}}) \cup \vartheta(\mathcal{Y}^{\check{A}}) = \vartheta(\mathcal{Y}).$$

It remains to prove the two claims are valid.

Case 1: $\mathcal{X}^{\check{S}}, \mathcal{Y}^{\check{S}} \subseteq \check{S}_T(v)$ with $\mathcal{X}^{\check{S}} \sqsubseteq \mathcal{Y}^{\check{S}}$

Assume to the contrary that there is some $X \in \vartheta(\mathcal{X}^{\check{S}})$ which is not contained in any essential v -subgraph belonging to $\vartheta(\mathcal{Y}^{\check{S}})$. Let Y be the smallest v -subgraph in $\mathcal{Y}^{\check{S}}$ which contains such an X . From the transitivity of p-subsumption and $\vartheta(\mathcal{X}^{\check{S}}) \sqsubseteq \mathcal{X}^{\check{S}} \sqsubseteq \mathcal{Y}^{\check{S}}$ we know that $\vartheta(\mathcal{X}^{\check{S}}) \sqsubseteq \mathcal{Y}^{\check{S}}$. The definition of p-subsumption then establishes that Y exists. Moreover $Y \notin \vartheta(\mathcal{Y})$. That is Y is not an essential v -subgraph and $\vartheta(\{Y\}) \neq \{Y\}$ by Remark 1.34. From previous assumptions, X also cannot be contained in any one essential v -subgraph belonging to $\vartheta(\{Y\})$ because these are all contained in elements of $\vartheta(\mathcal{Y}^{\check{S}})$. But this means X is the union of multiple essential v -subgraphs from $\vartheta(\{Y\})$. This implies that X is not an essential v -subgraph and thus contradicts $X \in \vartheta(\mathcal{X}^{\check{S}})$.

Case 2: $\mathcal{X}^{\check{A}}, \mathcal{Y}^{\check{A}} \subseteq \check{A}_T(v)$ with $\mathcal{X}^{\check{A}} \sqsubseteq \mathcal{Y}^{\check{A}}$

For any set $\mathcal{Y}^{\check{A}}$ of v -ancestors, the node $\text{lca}(\mathcal{Y}^{\check{A}})$ p-subsumes every other node in $\mathcal{Y}^{\check{A}}$. Since $\text{lca}(\mathcal{X}^{\check{A}}) \in \mathcal{X}^{\check{A}}$ and $\text{lca}(\mathcal{Y}^{\check{A}}) \in \mathcal{Y}^{\check{A}} \subseteq \mathcal{Y}$, we can find some v -ancestor $n \in \mathcal{Y}^{\check{A}}$ such that $\text{lca}(\mathcal{X}^{\check{A}}) \sqsubseteq n \sqsubseteq \text{lca}(\mathcal{Y}^{\check{A}})$ because $\mathcal{X}^{\check{A}} \sqsubseteq \mathcal{Y}^{\check{A}}$. Thus $\text{lca}(\mathcal{X}^{\check{A}}) \sqsubseteq \text{lca}(\mathcal{Y}^{\check{A}})$. From the definition of the refinement ϑ for v -ancestors we get $\vartheta(\mathcal{X}^{\check{A}}) = \vartheta(\{\text{lca}(\mathcal{X}^{\check{A}})\}) \sqsubseteq \vartheta(\{\text{lca}(\mathcal{Y}^{\check{A}})\}) = \vartheta(\mathcal{Y}^{\check{A}})$. □

1.4 Thesis Outline

The rest of the thesis is dedicated to the investigation of various aspects of this new class of XML functional dependencies with properties.

Firstly we show that pXFDs can be reasoned about efficiently. In Chapter 2, we present a semantic equivalence between pXFDs and a fragment of propositional logic. The chapter also explores three important applications of the semantic equivalence: a procedure for answering the implication problem for pXFDs; support for identifying and proving a sound and complete system of inference rules for pXFDs; and decision support in pXFDs acquisition. Polynomial decidability of the implication problem and finite axiomatisation of pXFDs are two particularly important results in the thesis.

Secondly, Chapter 3 presents a transversal approach for discovering pXFDs from a given XML data tree. The problem to be addressed is the discovery of all pXFDs which are satisfied in the given schema. The beginning of the chapter argues the appropriateness of the overall approach while the latter part of the chapter investigates several sub-problems which are vital for the applicability of the transversal approach to pXFDs discovery. We consider p-subsumption as an ordering over the set of all LHSs/RHSs of syntactically valid canonical pXFDs. The ordering gives rise to several additional sound inference rules for the implication of pXFDs, which in turns yields a more compact representation for the set of all satisfied pXFDs in XML tree T' in the form of the canonical pXFD-cover of T' .

Thirdly, we consider the role that pXFDs play in recognising the presence of data redundancy and thus characterising “good” schemas. This is one of the mainstream application for functional dependencies in many data models. Chapter 4 provides definition of two types of redundancy with respect to a set of pXFDs: fact redundancy and value redundancy. Furthermore, two normal forms are presented along with proofs that the normal forms correctly characterise the absence of fact and value redundancy. We further show that checking whether a schema is in one of the normal form can be done efficiently.

Finally, chapter 5 concludes the thesis with a summary of important results and some outline of possible future work.

Chapter 2

Semantic Equivalence Theorem for pXFDs

The efficiency with which we are able to reason about functional dependencies (i.e., decide implication) plays an important role in our ability to capitalise on their applications in areas such as schema design and query optimisation. In this chapter, our primary focus is on the question of whether pXFDs can be reasoned about efficiently. The answer in the affirmative is derived from the establishment of a semantic equivalence between pXFDs and Horn clauses. As it turns out, this logical characterisation of pXFD implication is not only useful for deciding the implication problem of pXFDs but lends itself to several other applications.

A semantic equivalence between FDs and a fragment of propositional logic was first demonstrated by Fagin. Since then, similar equivalences have been established for other classes of integrity constraints and data models, e.g., for relational FDs and MVDs [78], HL-XFDs [44, 45], and FDs in complex-value databases [42].

Fagin’s early work [24] established a relationship between the implication of FDs and the logical consequence of propositional formulas. Let $\Sigma^r \cup \{\sigma^r\}$ be a set of FDs over some relational schema r . We use \models_r to denote FD implication. Fagin proposed an encoding of FDs into propositional formulas. Let $F_{\Sigma^r}, F_{\sigma^r}$ be propositional formulas resulting from Fagin’s proposed encoding of σ^r and Σ^r respectively. The paper shows that statements:

- (1) implication $\Sigma^r \models_r \sigma^r$ holds, and (3) F_{σ^r} is a logical consequence of F_{Σ^r}

are equivalent. Fagin conceived two methods for proving this semantic equivalence: a semantic proof and a syntactic proof. The two approaches are summarised in Figure 2.1.

The *semantic proof* makes use of equivalences to the intermediary statement:

- (2a) implication $\Sigma^r \models_r \sigma^r$ holds in all two-tuple relations.

Specifically, Fagin first shows that it is sufficient to examine only relations consisting of two tuples when verifying FD implication. This is followed by an observation that boolean truth assignment of propositional variables (corresponding to attributes) can be related to agreement of the tuples in two-tuple relations. This special boolean assignment

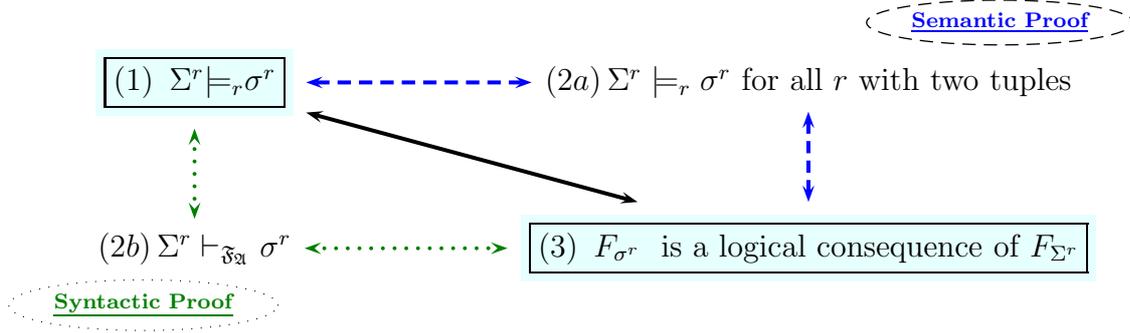


Figure 2.1: Fagin’s Semantic and Syntactic Proof of the semantic equivalence between FDs and propositional logic.

is used in the proof of equivalence between statements (2a) and (3) - guiding the inference of a counter-example boolean assignment for logical consequence from a counter-example relations for FD implication, and vice versa.

On the other hand, the *syntactic proof* establishes equivalences with the intermediary statement:

(2b) $\Sigma^r \vdash_{\mathfrak{F}_{\Sigma}} \sigma^r$ where \mathfrak{F}_{Σ} is an axiomatisation for the implication of FDs.

The equivalence between statement (1) and (2b) simply proclaims that \mathfrak{F}_{Σ} is sound and complete for the implication of FDs. Specifically, Fagin chose \mathfrak{F}_{Σ} to be equivalence to the Armstrong Axioms. To show that statement (2b) implies statement (3) Fagin argued that the translation of the Armstrong Axioms yields rules which are sound for determining the logical consequence of propositional formulas. For the opposite direction, that statement (3) implies (2b), Fagin makes use of a special “magic truth assignment” which relates the truth values of propositional variables to whether or not the corresponding attributes belong to the attribute closure $X_{\mathfrak{F}_{\Sigma}}^+$, supposing that σ^r is the FD $X \rightarrow Y$. It is finally shown that the derivation $\Sigma^r \vdash_{\mathfrak{F}_{\Sigma}} \sigma^r$ does not hold but F_{σ^r} is a logical consequence of F_{Σ^r} is a contradiction in light of the magic truth assignment.

A variation on the syntactic proof is expounded in [78], this time, for establishing a semantic equivalence between a fragment of propositional logic and implications of both FDs and multi-valued dependencies (MVDs) together. In fact, the approach taken is more mixed, proving:

- (1) and (2b) are equivalent
- if (2b) holds then so does (3)
- if (3) holds then so does (1), making use of the equivalence between (1) and (2b) and the Completeness Theorem for Formulas showing (2b) and (3) are equivalent

where we consider $\Sigma^r \cup \{\sigma^r\}$ to be a set of FDs and MVDs and \mathfrak{F}_{Σ} to be an axiomatisation for the implication of FDs and MVDs.

Next we formally prove a similar semantic equivalence between pXFDs and propositional Horn clauses: that logical implication of canonical pXFDs relates to logical consequence of the corresponding propositional Horn clauses. We follow the semantic proof approach of Fagin. Although, no syntactic proof will be provided, one can be prepared from the identified axiomatisation for implication of canonical pXFDs.

After proving the semantic equivalence, we will discuss three useful applications of a semantic equivalence to propositional logic:

- for deciding the implication problem,
- for finding and proving the correctness of an axiomatisation for pXFDs and,
- for supporting the constraint acquisition process.

The first of these applications obviously follows from the statement of the semantic equivalence itself. The second application stems from a re-think of Fagin’s syntactic proof. For the third application, we make use of the equivalence to generate sufficient sample data to support designers in deciding whether to specify a given constraint.

2.1 Relating Canonical pXFD Implication to Logical Consequence of Horn Clauses

Some preliminary work into a semantic equivalence between HL-XFDs and propositional logic, has been reported in [44]. The proof in this thesis is, for the main part, similar to that of equivalence between HL-XFDs and propositional Horn clauses. This is due to several favourable conditions. For one, structural constraints relating to v -ancestors are expressible with propositional Horn clauses. Also, the special truth value assignment which relates two- v -pre-image instances and propositional variables associated with v -subgraphs is readily extended to encompass propositional variables associated with v -ancestors. And thirdly, the two- v -pre-image counter-example construction given a suitable boolean truth assignment remains valid with minor changes.

Recall that, without loss of generalisation, we can talk about canonical pXFDs (as per Definition 1.6) rather than pXFDs. Encoding only canonical pXFDs allows us to focus on only essential v -properties which is likely to be fewer than v -properties. This leads to the obvious benefits of keeping our Horn encoding reasonably small which is an important factor in how efficiently we can decide logical consequences of Horn clauses.

We first specify a Horn encoding for canonical pXFD implication. In addition to examining pXFDs which are involved in an implication, it is also necessary to encode inter-relationships among the essential v -properties to reflect our observations about property-equality from Section 1.3.3. Note that we can ignore the observations concerning simple descendant and v -reconcillable v -subgraphs because only essential v -properties are being considered.

2.1.1 Horn Clause Encoding

Some basic familiarity with propositional logic is assumed (e.g., see [14, 46, 94]).

A *Horn clause* over some given set of literals \mathcal{V} is a *clause* (i.e., a disjunction of literals) with at most one *positive* (i.e., non-negated) literal. A *definite Horn clause* is a Horn clause having exactly one positive literal. A definite Horn clause $\neg X_1 \vee \dots \vee \neg X_j \vee Y$ can be alternatively expressed by a propositional formula of the form: $X_1 \wedge \dots \wedge X_j \Rightarrow Y$.

Let $\varphi : \mathbf{E}_T^{\mathbb{S}}(v) \cup \mathbf{E}_T^{\mathbb{A}}(v) \rightarrow \mathcal{V}$ be a mapping that assigns propositional variables to the essential v -properties of T . If σ is a canonical pXFD $v : \{X_1, \dots, X_j\} \rightarrow \{Y_1, \dots, Y_k\}$ then, let H_σ be the set of the following k Horn clauses over \mathcal{V} :

$$\begin{aligned} \varphi(X_1) \wedge \dots \wedge \varphi(X_j) &\Rightarrow \varphi(Y_1) \\ &\vdots \\ \varphi(X_1) \wedge \dots \wedge \varphi(X_j) &\Rightarrow \varphi(Y_k) \end{aligned}$$

For a set Σ of pXFDs, let H_Σ be the union of the sets H_σ for all $\sigma \in \Sigma$. Furthermore, we capture information about inherent inter-relationships among essential v -properties by the *base translation*:

$$\begin{aligned} H_T = & \{ \varphi(W) \Rightarrow \varphi(Z) \mid W, Z \in \mathbf{E}_T^{\mathbb{S}}(v) \text{ and } W \text{ covers } Z \text{ and } Z \neq \emptyset \} \\ & \cup \{ \varphi(n) \Rightarrow \varphi(m) \mid n, m \in \mathbf{E}_T^{\mathbb{A}}(v) \text{ and } m \text{ is an immediate essential ancestor} \\ & \quad \text{of } n \text{ and } n \neq r_T \} \\ & \cup \{ \varphi(n) \Rightarrow \varphi(U) \mid \{n\} = \vartheta(\{v\}) \text{ and } U \text{ is a } v\text{-unit} \} \\ & \cup \{ \varphi(\emptyset), \varphi(r_T) \} \end{aligned}$$

where

- a v -subgraph W is said to *cover* a v -subgraph Z if and only if W is the union of Z and just one additional v -walk of T
- an essential v -ancestor m is an *immediate essential ancestor* of another essential v -ancestor n (or equivalently n is an *immediate essential descendant* of m) if and only if m is a proper ancestor of n and no other essential v -ancestor is both a proper ancestor of n and a proper descendant of m (i.e., n is the next essential v -ancestor which can be reached from m).

In order to minimise the size of the base translation we have made use of the transitive nature of logical implication to consider only *covering* v -subgraphs and *immediate essential* ancestor/descendant essential v -ancestors. It is, of course, possible to use the more general notion of subgraph containment and ancestor/descendant relationship but this is likely to result in a considerably larger set of Horn clauses for the base translation.

The following example shows how we can encode pXFDs over the XML schema tree T_{dance} .

Example 2.1. Consider the node $v_{partners}$ in the schema tree T_{dance} from Figure 1.4. The essential $v_{partners}$ -properties are:

$$v_{danceschool}, v_{class}, v_{session}, v_{partners}, \mathbf{boy}, \mathbf{girl} \text{ and } \emptyset.$$

Let σ be pXFD3 and Σ be the set consisting of pXFD1, pXFD2 and

$$v_{partners} : \{\mathbf{boy}, \mathbf{girl}\} \rightarrow \{v_{class}\} \quad (pXFD6')$$

Note that canonical pXFD6' expressed the same constraint as pXFD6 from Section 1.3.4.

For some φ mapping the set of essential $v_{partners}$ -properties to propositional variables, our translation yields the following sets of Horn clauses:

$$\begin{aligned} H_\sigma &= \{ \varphi(\mathbf{boy}) \Rightarrow \varphi(\mathbf{girl}) \} \\ H_\Sigma &= \left\{ \begin{array}{l} \varphi(v_{class}) \wedge \varphi(\mathbf{boy}) \Rightarrow \varphi(\mathbf{girl}), \\ \varphi(v_{class}) \wedge \varphi(\mathbf{girl}) \Rightarrow \varphi(\mathbf{boy}), \\ \varphi(\mathbf{boy}) \wedge \varphi(\mathbf{girl}) \Rightarrow \varphi(v_{class}) \end{array} \right\} \\ H_T &= \left\{ \begin{array}{l} \varphi(v_{partners}) \Rightarrow \varphi(v_{session}), \\ \varphi(v_{session}) \Rightarrow \varphi(v_{class}), \\ \varphi(v_{partners}) \Rightarrow \varphi(\mathbf{boy}), \\ \varphi(v_{partners}) \Rightarrow \varphi(\mathbf{girl}), \\ \varphi(\emptyset), \\ \varphi(v_{danceschool}) \end{array} \right\} \end{aligned}$$

□

Next we detail a semantic equivalence in terms of logical implication of pXFDs and logical consequence of propositional Horn clauses resulting from the above encoding of pXFDs. A challenging aspect of the proof is showing the existence of a counter-example T -compatible data tree to $\Sigma \models \sigma$ if H_σ is not a logical consequence of H_Σ . The complex discussion relating to our proposed construction warrants its own subsection - this is presented after the ensuing proof of the semantic equivalence of canonical pXFDs and Horn clauses.

2.1.2 Semantic Equivalence of Canonical pXFDs to Logic

Let T be an XML schema tree in which there is a node v . Furthermore let $\Sigma \cup \{\sigma\}$ be a set of canonical pXFDs over T and let σ be $v : \mathcal{X} \rightarrow \mathcal{Y}$. We will show the following statements to be equivalent:

1. $\Sigma \models \sigma$
2. Σ implies σ in the world of two- v -pre-image data trees
3. $H_\Sigma \cup H_T$ logically implies H_σ

Briefly, recall the different notions of implication from Section 1.3.2. We use \models to denote pXFDs implication, that is, $\Sigma \models \sigma$ holds if and only if every T -compatible data tree which satisfy all pXFDs in Σ also satisfy the pXFD σ . On the other hand, to check whether an implication holds *in the world of two- v -pre-image data trees*, we need only to consider those T -compatible data trees which contain exactly two pre-image trees of v . As for “logical consequence”, we take the usual notion from logic as follows:

H_σ is a *logical consequence* of H_Σ (or equivalently H_Σ *logically implies* H_σ) if and only if for every boolean truth assignment \mathbb{B} such that every propositional formula in H_Σ evaluates to *true*, every propositional formula in H_σ also evaluates to *true* under \mathbb{B} .

We proceed to prove that statements (1) and (2) are equivalent (Proposition 2.2) and that statements (2) and (3) are equivalent (Proposition 2.6). These two proofs support that (1) and (3) are equivalent (Theorem 2.7) giving one of the main results for this chapter.

The equivalence of statement (1) and (2) reveals that T -compatible two- v -pre-image data trees are sufficient for witnessing whether a pXFD σ is implied from a set Σ of pXFDs. More specifically, we show that every T -compatible data tree which witness $\Sigma \models \sigma$ to be *false* contains a two- v -pre-image data tree that already witnesses the same fact. This fact additionally supports that finite and unrestricted implication of pXFDs coincide.

Proposition 2.2. *The following statements are equivalent:*

1. $\Sigma \models \sigma$
2. Σ *implies* σ *in the world of two- v -pre-image data trees*

Proof. (1. \Rightarrow 2.) Is trivial because all two- v -pre-image data trees are T -compatible data trees.

(1. \Leftarrow 2.) Assume $\Sigma \not\models \sigma$. From this, we infer the existence of some T -compatible data tree T' such that T' satisfies every pXFD in Σ but violates σ . Recall that σ is $v : \mathcal{X} \rightarrow \mathcal{Y}$. Particularly, T' has two pre-image trees $p_1, p_2 \in P_{T'}(v)$ with $p_1|_X \doteq p_2|_X$ for all $X \in \mathcal{X}$ and $p_1|_Y \not\doteq p_2|_Y$ for some $Y \in \mathcal{Y}$.

We trim T' to a T -compatible two- v -pre-image data tree T'' consisting of p_1, p_2 as follows:

- Remove all pre-images of v not belonging to p_1 and p_2 , as well as all its descendants.
- If a node e was removed from T' and e is the sole pre-image of some node w under the T -compatibility homomorphism, where arc (u, w) in T has frequency 1 or +, then we also remove the parent of e as well as all descendants of e .

The presence of p_1, p_2 implies v is not simple. And so there is some v -ancestor y such that its incoming arc (x, y) has frequency other than $?$ and 1 and such that x is simple. Since T' is T -compatible, there is a single pre-image of x that is shared by all pre-image trees of v . This ensures that the trimming process will terminate. Observe that the trimming process does not affect walks which are contained in the pre-image tree p_1 or p_2 , nor walks that do not belong to either p_1, p_2 but that are needed to ensure T' is T -compatible. In other words, there is a copy of these subtrees in the resulting T'' . Hence the resulting data tree T'' will be T -compatible and retains the pre-image trees p_1, p_2 . It follows that T'' satisfies every pXFD in Σ since otherwise p_1, p_2 would witness to Σ not holding in T' . Therefore T'' is a witness to the fact that Σ does not imply σ in the world of two- v -pre-image data trees.

In summary, we have reasoned that if $\Sigma \not\models \sigma$ then there is a T -compatible data tree T' which plays witness to the violation. The data tree T' can be trimmed to yield a two- v -pre-image data tree T'' which verifies that Σ does not imply σ in the world of two- v -pre-image data trees. This shows that $(1. \Leftarrow 2.)$ and thus concludes the proof. \square

The equivalence between (2) and (3) is a more complex matter. There are two pivotal questions to be answered:

- how do we relate boolean (truth) assignments to two- v -pre-image data trees in such a way that pXFDs are satisfied in T' precisely when their corresponding Horn clauses evaluate to *true* under the boolean assignment;
- how do we construct a counter-example two- v -pre-image tree for the implication given a counter-example boolean assignment for the logical consequence.

Before continuing with the semantic proof, we identify a special boolean assignments and show that it indeed relates to two- v -pre-image data trees in the way we have just described. Detailed presentation of our proposed construction for two- v -pre-image data tree counter-examples is deferred to Section 2.1.3.

Definition 2.3 (representative boolean assignment). Let T be an XML schema tree and $\varphi : \mathbf{E}_T^{\check{S}}(v) \cup \mathbf{E}_T^{\check{A}}(v) \rightarrow \mathcal{V}$ be a mapping assigning propositional variables to the essential v -properties of T . Further, let \mathbb{B} be a boolean assignment of all propositional variables in \mathcal{V} . The boolean assignment \mathbb{B} is said to be *representative* of a given T -compatible two- v -pre-image data tree T' where $P_{T'}(v) = \{p_1, p_2\}$ if and only if the following statement holds:

$$\mathbb{B}(\varphi(W)) = \textit{true} \text{ if and only if } p_1|_W \doteq p_2|_W \\ \text{for every essential } v\text{-property } W \in \mathbf{E}_T^{\check{S}}(v) \cup \mathbf{E}_T^{\check{A}}(v)$$

\square

The following pair of lemmas detail two important properties of boolean assignment \mathbb{B} which are representative of some two- v -pre-image data tree T' . The first lemma is analogous to the Semantic Lemma of Fagin [24] and relates the satisfaction of pXFDs in T' to the truth value of the corresponding Horn clauses under \mathbb{B} . The second lemma affirms

that every Horn clause belonging to H_T evaluates to *true* under \mathbb{B} . This attests to the fact that H_T captures only inherent inter-relationships among essential v -properties with respect to property-equality. For this, we make use of the observations about property-equality from Section 1.3.3.

Lemma 2.4 (Semantic Lemma). *Let \mathbb{B} be a boolean assignment which is representative of some T -compatible data tree T' . Let σ be a canonical pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$ over T . Then σ holds in T' if and only if every Horn clause in the set H_σ evaluates to *true* under \mathbb{B} .*

Proof. Let p_1, p_2 be the two distinct pre-images of v in T' .

(\Leftarrow) Assume each Horn clause in H_σ has truth value *true* under \mathbb{B} . This means $\mathbb{B}(\varphi(X)) = \textit{false}$ for some $X \in \mathcal{X}$, or $\mathbb{B}(\varphi(Y)) = \textit{true}$ for all $Y \in \mathcal{Y}$. Because \mathbb{B} is representative of T' , we obtain $p_1|_X \neq p_2|_X$ holds for some $X \in \mathcal{X}$ or $p_1|_Y \doteq p_2|_Y$ holds for all $Y \in \mathcal{Y}$. In either case, p_1, p_2 do not cause σ to be violated and so T' satisfies σ .

(\Rightarrow) Assume that T' satisfies σ . In the case that $p_1|_X \neq p_2|_X$ holds for some $X \in \mathcal{X}$ we have $\mathbb{B}(\varphi(X)) = \textit{false}$. This results in $\bigwedge_{X \in \mathcal{X}} \varphi(X)$ evaluating to *false* and therefore, each Horn clauses in H_σ evaluates to *true*. In the contrary case, where $p_1|_X \doteq p_2|_X$ for all $X \in \mathcal{X}$, we must have $p_1|_Y \doteq p_2|_Y$ for all $Y \in \mathcal{Y}$. It follows that $\mathbb{B}(\varphi(X)) = \textit{true}$ for all $X \in \mathcal{X}$ and $\mathbb{B}(\varphi(Y)) = \textit{true}$ for all $Y \in \mathcal{Y}$. This means that all Horn clauses in H_σ evaluate to *true* under \mathbb{B} . These are all the cases, hence the proof is complete. \square

Lemma 2.5 (Triviality Lemma). *Let \mathbb{B} be a boolean assignment which is representative of some T -compatible data tree T' . Then every Horn clause in H_T evaluates to *true* under \mathbb{B} .*

Proof. Recall that H_T is defined in terms of four sets as follows:

$$\begin{aligned} H_T = & \{ \varphi(W) \Rightarrow \varphi(Z) \mid W, Z \in \mathbf{E}_T^{\mathbb{S}}(v) \text{ and } W \text{ covers } Z \text{ and } Z \neq \emptyset \} \\ & \cup \{ \varphi(n) \Rightarrow \varphi(m) \mid n, m \in \mathbf{E}_T^{\mathbb{A}}(v) \text{ and } m \text{ is an immediate essential ancestor} \\ & \quad \text{of } n \text{ and } n \neq r_T \} \\ & \cup \{ \varphi(n) \Rightarrow \varphi(U) \mid \{n\} = \vartheta(\{v\}) \text{ and } U \text{ is a } v\text{-unit} \} \\ & \cup \{ \varphi(\emptyset), \varphi(r_T) \} \end{aligned}$$

Assume some Horn clause $h \in H_T$ evaluates to *false* under \mathbb{B} . We show that this leads to a contradiction. Formula h can belong to any of the constituting sets of H_T . Therefore, we have four cases to consider. Let p_1, p_2 be the two distinct pre-image trees of v in T' .

Case 1: $h \in \{ \varphi(W) \Rightarrow \varphi(Z) \mid W, Z \in \mathbf{E}_T^{\mathbb{S}}(v) \text{ and } W \text{ covers } Z \text{ and } Z \neq \emptyset \}$

From the initial assumption, we have $\mathbb{B}(\varphi(Z)) = \textit{false}$ and $\mathbb{B}(\varphi(W)) = \textit{true}$, which means that $p_1|_Z \neq p_2|_Z$ and $p_1|_W \doteq p_2|_W$ hold. Z is contained in W so $p_1|_W \doteq p_2|_W$ implies $p_1|_Z \doteq p_2|_Z$. This is a contradiction since the pre-image trees cannot both agree and differ on Z .

Case 2: $h \in \{ \varphi(n) \Rightarrow \varphi(m) \mid n, m \in \mathbf{E}_T^{\mathbb{A}}(v) \text{ and } m \text{ is an immediate essential ancestor of } n \text{ and } n \neq r_T \}$

From the initial assumption, we have $\mathbb{B}(\varphi(m)) = \text{false}$ and $\mathbb{B}(\varphi(n)) = \text{true}$, which means that $p_1|_m \neq p_2|_m$ and $p_1|_n \doteq p_2|_n$ hold. n is a descendant of m which means $p_1|_n \doteq p_2|_n$ implies $p_1|_m \doteq p_2|_m$. This is a contradiction since the pre-image trees cannot share and have distinct pre-images of m .

Case 3: $h \in \{\varphi(n) \Rightarrow \varphi(U) \mid \{n\} = \vartheta(\{v\}) \text{ and } U \text{ is a } v\text{-unit}\}$

From the initial assumption, we have $\mathbb{B}(\varphi(U)) = \text{false}$ and $\mathbb{B}(\varphi(n)) = \text{true}$, which means that $p_1|_U \neq p_2|_U$ and $p_1|_n \doteq p_2|_n$ hold. Distinct pre-image trees do not share their pre-image of v nor its simple ancestor. By definition of refinement n is a simple ancestor of v which means p_1, p_2 share the same pre-image of n and also share the same pre-image of v because T' is T -compatible. This contradicts that p_1, p_2 are distinct pre-image trees in T' .

Case 4: $h \in \{\varphi(\emptyset), \varphi(r_T)\}$

The initial assumption gives $\mathbb{B}(\varphi(\emptyset)) = \text{false}$ or $\mathbb{B}(\varphi(r_T)) = \text{false}$ which means that $p_1|_\emptyset \neq p_2|_\emptyset$ or $p_1|_{r_T} \neq p_2|_{r_T}$ hold. Projections to the empty v -subgraph yields the empty XML tree and XML trees are rooted. In other words $p_1|_X \doteq p_2|_X$ holds whenever $X \in \{\emptyset, r_T\}$, a contradiction to our initial assumption.

□

We now return to the equivalence of statements (2) and (3). Our proof is by the contrapositive. One direction follows easily from the two properties of representative boolean assignments stated above, while the other direction additionally relies on our ability to construct a two- v -pre-image data tree from some input boolean assignment \mathbb{B} such that the the resulting data tree has \mathbb{B} as a representative boolean assignment.

Proposition 2.6. *The following statements are equivalent:*

2. Σ implies σ in the world of two- v -pre-image data trees
3. $H_\Sigma \cup H_T$ logically implies H_σ

Proof. (2. \Leftarrow 3.) Assume that Σ does not imply σ in the world of two- v -pre-image data trees. There exists some T -compatible data tree T' with exactly two pre-image trees p_1, p_2 of v that satisfies Σ but not σ . Take the boolean assignment \mathbb{B} which is representative of T' . By Lemma 2.4, there is some Horn clause in H_σ that evaluates to *false* because T' violates σ . On the other hand, since T' satisfies Σ every Horn clause in H_Σ evaluate to *true* under \mathbb{B} . From Lemma 2.5 we also have every Horn clause in H_T evaluating to *true*. Therefore H_σ is not logically implied by $H_\Sigma \cup H_T$.

(2. \Rightarrow 3.) Assume that $H_\Sigma \cup H_T$ does not logically imply H_σ . Then let \mathbb{B} be a truth assignment such that every Horn clause in $H_\Sigma \cup H_T$ evaluates to *true* under \mathbb{B} but one clause in H_σ evaluates to *false* under \mathbb{B} . We will show \mathbb{B} is representative of a T -compatible two- v -pre-image data tree T' that witnesses that Σ does not imply σ . We apply the construction from Section 2.1.3 which requires, as input, an *equality set* of v -properties as per Definition 2.8 or the corresponding *non-equality set* as per Definition 2.9.

Firstly, we define the subset $\check{\mathbb{A}}\mathcal{E}_T$ inductively as follows, for every v -ancestor $u \in \check{\mathbb{A}}_T(v)$:

- $u \in \check{\mathcal{E}}_T$ if $\mathbb{B}(\varphi(u)) = \text{true}$,
- if $u \in \check{\mathcal{E}}_T$ and w is a simple descendant of u then $w \in \check{\mathcal{E}}_T$
- nothing else belongs to $\check{\mathcal{E}}_T$.

There must be at least one propositional variable which has been assigned *false* by \mathbb{B} , otherwise would result in some Horn clause in H_T evaluating to *false* under \mathbb{B} or all propositional variable being assigned *true*, both are contradictions to our initial assumption. In particular, the single node in $\vartheta(\{v\})$ must have all been assigned *false* by \mathbb{B} . It is easy to see that, $\check{\mathcal{E}}_T$ contains all essential v -ancestors whose corresponding propositional variables are assigned *true* by \mathbb{B} and obeys the four conditions relating to v -ancestors in Definition 2.8.

Secondly, we need to identify the subset $\check{\mathcal{E}}_T$, defined inductively as follows, for every v -subgraph $X, Y \in \check{\mathcal{S}}_T(v)$:

- $X \in \check{\mathcal{E}}_T$ if $\mathbb{B}(\varphi(X)) = \text{true}$,
- if $X, Y \in \check{\mathcal{E}}_T$ and X, Y are v -reconcilable then $X \cup Y \in \check{\mathcal{E}}_T$, and
- nothing else belongs to $\check{\mathcal{E}}_T$.

Note that the inductive case adds only v -subgraphs which are not essential and is the union of some set of pairwise v -reconcilable essential v -subgraphs. It also follows from the triviality lemma that: (1) the empty subgraph belongs to $\check{\mathcal{E}}_T$ and, (2) if an *essential* v -subgraph X belongs to $\check{\mathcal{E}}_T$ then so does all v -subgraphs contained in X , which then means that any subgraph belonging to $\check{\mathcal{E}}_T$ (whether essential or not) also has all the v -subgraphs it contains belonging to $\check{\mathcal{E}}_T$. In other words, $\check{\mathcal{E}}_T$ obeys all the conditions relating to v -subgraphs from Definition 2.8.

In summary, $\mathcal{E}_T = \check{\mathcal{E}}_T \cup \check{\mathcal{E}}_T$ is an equality set. By Lemma 2.19 we can create a T -compatible data tree T' possessing exactly two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that

$$p_1|_X \doteq p_2|_X \text{ if and only if } X \in \mathcal{E}_T \text{ for all } v\text{-property } X \in \check{\mathcal{S}}_T(v) \cup \check{\mathcal{A}}_T(v)$$

From our specification of \mathcal{E}_T we then have

$$p_1|_W \doteq p_2|_W \text{ if and only if } \mathbb{B}(\varphi(W)) = \text{true}, \text{ for all } W \in \mathbf{E}_T^{\check{\mathcal{S}}}(v) \cup \mathbf{E}_T^{\check{\mathcal{A}}}(v)$$

That is \mathbb{B} is representative of T' . Lemma 2.4 conveys that T' does not satisfy σ because some clause in H_σ evaluates to *false* under \mathbb{B} . Similarly we infer T' satisfies Σ . Thus, T' is a counter-example to Σ implying σ in the world of two- v -pre-image data trees. \square

All discussions thus far culminate in the following important contribution for the thesis:

Theorem 2.7 (Semantic pXFD-Equivalence Theorem). *The following statements are equivalent:*

1. $\Sigma \models \sigma$
3. $H_\Sigma \cup H_T$ logically implies H_σ

Proof. Immediate from Proposition 2.2 and Proposition 2.6. \square

2.1.3 Construction of Two- v -pre-image Data Tree

In this section, we address the problem of how to construct a T -compatible two- v -pre-image data tree which contains two distinct pre-image trees of v that *agree precisely on some given set of v -properties*. This means, the two contained pre-image trees share their pre-images of each v -ancestor in the set and are property-equal on each v -subgraph in the set, but differ on all v -properties not specified. The construction described here is applied in the proof of the Semantic Equivalence Theorem for pXFDs from the previous section, but can also be used in other context, for example, with the sample-based decision support of pXFD acquisition in Section 2.2.3.

We propose two phases to the construction:

1. construct two separate T -compatible pre-image trees of v which *agree precisely on specified v -subgraphs*,
2. merging the constructed pre-image trees to form a T -compatible data tree such that the two pre-image trees *share precisely the specified v -ancestors*.

The two-phase approach is feasible since one phase considers only the descendants of v while the other deals exclusively with nodes not descended from v .

As input, we require an XML schema tree T and a set of v -properties which respect certain conditions originating from the observations about property-equality that we made in Section 1.3.3. Without these conditions, it is clearly impossible to construct a two- v -pre-image data tree which is both T -compatible and has pre-image trees that agree precisely on a given set of v -properties.

Definition 2.8. An *equality set* \mathcal{E}_T is a subset of v -properties such that the following conditions all hold:

- The subset of v -ancestors (i.e., $\check{\mathbb{A}}\mathcal{E}_T = \mathcal{E}_T \cap \check{\mathbb{A}}_T(v)$) must:
 - contain the root node r_T ;
 - be *complete* (i.e., if a v -ancestor is included in the set then *all of its ancestors* are also included in the set);
 - if a v -ancestor is included in the set then *all of its simple descendants* are also included in the set; and
 - not contain the target node v .
- The subset of v -subgraphs (i.e., $\check{\mathbb{S}}\mathcal{E}_T = \mathcal{E}_T \cap \check{\mathbb{S}}_T(v)$) must:

- contain the empty subgraph \emptyset ;
- be *complete* (i.e., if a v -subgraph is included in the set then *all* v -subgraph contained in it are also in the set); and
- if two v -reconcilable v -subgraph X, Y are in the set then also v -subgraph $X \cup Y$ is included in the set.

□

Note that, the last two conditions for v -ancestors mean that an equality set cannot contain any descendant of $n \in \vartheta(\{v\})$, since these all have the target node v as a simple descendant.

Because equality set of v -properties are complete and there is a partial order over the set of essential v -properties (as discussed in Section 1.3.4), we can alternatively provide, as input, a (possibly empty) *non-equality set* \mathcal{NE}_T of v -properties on which the two pre-image trees must *minimally differ*. By minimally differing we mean that the two pre-image trees must agree precisely on those nodes which are *proper ancestors* of some node in \mathcal{NE}_T and on those subgraphs which are *properly contained* in some v -subgraph in \mathcal{NE}_T . It is straightforward to find a non-equality set corresponding to some given equality set and vice versa:

Definition 2.9. Given an equality set of v -properties \mathcal{E}_T , the corresponding *non-equality set* \mathcal{NE}_T is defined as follows

$$\mathcal{NE}_T = \{n \in \check{\mathbb{A}}_T(v) \mid n \notin \mathcal{E}_T \text{ and } \exists m \text{ that is a proper ancestor of } n \text{ and } m \notin \mathcal{E}_T\} \\ \cup_{\subseteq\text{-min}}(\{X \in \check{\mathbb{S}}_T(v) \mid X \notin \mathcal{E}_T\})$$

where $\subseteq\text{-min}(\mathcal{S})$ denote the subset of v -subgraphs belonging to set \mathcal{S} which are minimal with respect to subgraph containment. □

Corollary 2.10. Given a non-equality set of v -properties \mathcal{NE}_T , the corresponding equality set \mathcal{E}_T is constructed as follows

$$\mathcal{E}_T = \{n \in \check{\mathbb{A}}_T(v) \mid n \text{ is a proper ancestor of some } v\text{-ancestor } m \in \mathcal{NE}_T\} \\ \cup \{X \in \check{\mathbb{S}}_T(v) \mid X \text{ is properly contained in some } v\text{-subgraph } W \in \mathcal{NE}_T\}$$

□

More compactly, non-equality sets and its relationship to equality sets can also be stated in terms of p-subsumption as follows:

$$\mathcal{NE}_T = \subseteq\text{-min}(\{X \in \check{\mathbb{A}}_T(v) \cup \check{\mathbb{S}}_T(v) \mid X \notin \mathcal{E}_T\}) \\ \mathcal{E}_T = \{X \in \check{\mathbb{A}}_T(v) \cup \check{\mathbb{S}}_T(v) \mid X \sqsupset Y \text{ for some } Y \in \mathcal{NE}_T\}.$$

Clearly, members of every non-equality set are pairwise incomparable with respect to ancestor/descendant relationships and subgraph containment. Furthermore, a non-empty non-equality set will contain precisely one essential v -ancestor with an incoming arc of frequency other than ? and 1. Recall that essential v -subgraphs are characterised by their containment in some v -unit. It follows that we can consider a projection of a non-equality set to some v -unit thus yielding only essential v -subgraphs as follows:

Definition 2.11. The *projection of non-equality set* \mathcal{NE}_T to v -unit U is defined as

$$\mathcal{NE}_T|_U = \subseteq_{-\min}(\{X \in \check{\mathcal{S}}_T(v) \mid X \text{ is contained in } U \text{ and } X \in \mathcal{NE}_T\}).$$

□

Then, the following pair of lemmas reveals that, in fact, every non-equality set contain only essential v -subgraphs. Because of this, we can modularise the first phase of construction and proceed unit-by-unit.

Lemma 2.12. *For any v -unit U , we have $\mathcal{NE}_T|_U \subseteq \mathcal{NE}_T \cap \check{\mathcal{S}}_T(v)$.*

Proof. Let $X \in \mathcal{NE}_T|_U$ then $X \notin \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$. Moreover, every v -subgraph Y properly contained in X does not belong to $\mathcal{NE}_T|_U$. Since X is contained in U , so is every Y properly contained in X . Therefore $Y \in \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$ for every Y properly contained in X . Hence $X \in \mathcal{NE}_T \cap \check{\mathcal{S}}_T(v)$. □

Lemma 2.13. *Let $\mathcal{U}_T(v)$ be the set of all v -units in T and \mathcal{E}_T an equality set. Then $\mathcal{NE}_T \cap \check{\mathcal{S}}_T(v) = \bigcup_{U \in \mathcal{U}_T(v)} \mathcal{NE}_T|_U$.*

Proof. From Lemma 2.12 we have $\bigcup_{U \in \mathcal{U}_T(v)} \mathcal{NE}_T|_U \subseteq \mathcal{NE}_T \cap \check{\mathcal{S}}_T(v)$. It remains to show that $\mathcal{NE}_T \cap \check{\mathcal{S}}_T(v) \subseteq \bigcup_{U \in \mathcal{U}_T(v)} \mathcal{NE}_T|_U$ holds. Let $X \in \mathcal{NE}_T \cap \check{\mathcal{S}}_T(v)$. This means $X \notin \mathcal{E}_T$ and every v -subgraph contained in X belongs to \mathcal{E}_T . If X is not an essential v -subgraph then it is the union of v -reconcilable subgraphs belonging to \mathcal{E}_T . This would mean that X itself belong to \mathcal{E}_T , a contradiction. Therefore X is an essential v -subgraph and, by Lemma 1.21 X is contained in some v -unit $U \in \mathcal{U}_T(v)$. Hence $X \in \mathcal{NE}_T|_U \subseteq \bigcup_{U \in \mathcal{U}_T(v)} \mathcal{NE}_T|_U$. □

The remainder of the section details a combinatorial approach to constructing the desired data tree. We consider as input a non-equality set of v -properties \mathcal{NE}_T with corresponding equality set \mathcal{E}_T . Firstly, we consider the projection of \mathcal{NE}_T to every v -unit. Lemma 2.14 to Lemma 2.17 describe how to generate two U -compatible pre-image trees p_1^U, p_2^U of v which minimally differ on each v -subgraph in $\mathcal{NE}_T|_U$. This ensures that p_1^U, p_2^U in fact agree on precisely the v -subgraphs which are properly contained in projected non-equality set $\mathcal{NE}_T|_U$. Subsequently, Lemma 2.19 describes how to merge the generated pre-image trees for individual v -units to construct two T -compatible pre-image trees p_1, p_2 of v , and ultimately how to merge p_1, p_2 into a two- v -pre-image T -compatible data tree whose two pre-image trees agree precisely on v -properties belonging to \mathcal{E}_T .

For starter, how do we ensure that p_1, p_2 differ on a *single essential v -subgraph* $W \in \mathcal{NE}_T$ but agree on every v -subgraph properly contained in W ? For a data tree T' in which leaves are assigned the value “0” or “1”, let the *ones-subgraph* X be the largest v -subgraph for which

$$\text{val}(T'|_B) = \text{“1”} \text{ if and only if } v\text{-walk } B \text{ is contained in } X$$

We consider the following initial construction steps:

```

proc construct-pre-image-trees-one-subgraph( $W, U$ )
  Initialise  $p_1^W, p_2^W$  as empty XML trees
  for all (possibly empty)  $v$ -subgraph  $X$  which is contained in  $W$  do
    Create a copy  $c^X$  of  $U$  whose ones-subgraph is  $X$ 
    if  $|X|$  is odd then
       $p_1^W = \text{Merge } c^X$  with  $p_1^W$  on  $\eta(U)$ 
    else
       $p_2^W = \text{Merge } c^X$  with  $p_2^W$  on  $\eta(U)$ 
    end if
  end for
  return  $p_1^W, p_2^W$ 

```

What is meant by “merging” two data trees on some node w ? The merged data trees must share every node except their pre-images of w and its descendants. Observe that a data tree cannot contain multiple copies of U unless U itself contains more than one walk. If U is a singleton then there is only one copy of U to merge with the empty tree, which trivially return the copy of U . If U is not a singleton, then it has an identifier $\eta(U)$ whose incoming arc is the only arc on the path from v to $\eta(U)$ having frequency other than $?$ and 1 which is shared by all walks in U . The construction is illustrated in Figure 2.2, with Example 2.15 illustrating more concretely how the construction can be applied.

Lemma 2.14. *Let W be an essential v -subgraph contained in some v -unit U . It is possible to construct two U -compatible pre-image trees p_1^W, p_2^W of v such that*

$$p_1^W|_X \doteq p_2^W|_X \text{ if and only if } v\text{-subgraph } X \text{ is properly contained in } W$$

Proof. We show that the two pre-image trees p_1^W, p_2^W constructed by `construct-pre-image-trees-one-subgraph`, indeed differ on W but agree on every v -subgraph contained in W . It is easy to see that p_1^W, p_2^W are U -compatible and return all of its constituent copies of U . We have that W is the maximal ones-subgraph for any generated copy of U and there is exactly one copy of U whose ones-subgraph is W . Therefore p_1^W, p_2^W differ on W . What remains is to demonstrate why $p_1^W|_X \doteq p_2^W|_X$ holds for every v -subgraph X that is properly contained in W . Consider an arbitrary v -subgraph X contained in W . It is sufficient to show that there is a bijection f_X between the copies of U contained in p_1^W and the copies of U contained in p_2^W such that $c|_X \doteq f_X(c)|_X$ for every copy c of U contained in p_1^W .

Property-equality of two copies of U projected to some v -subgraph X can be determined from the ones-subgraphs of the resulting data trees:

$$c_i|_X \doteq c_j|_X \text{ if and only if } c_i|_X, c_j|_X \text{ have isomorphic ones-subgraph}$$

Furthermore, we can observe that projecting any copy of U to some v -subgraph X always result in a data tree whose ones-subgraph is contained in X . Therefore, we can partition the generated copies of U according to the ones-subgraph of their projection to X . For

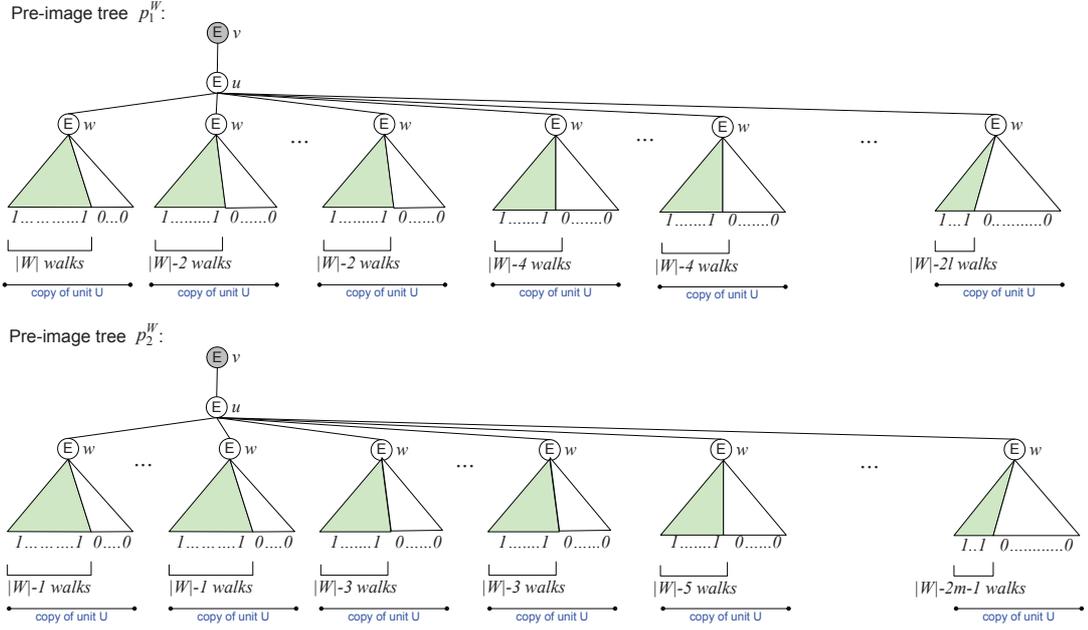


Figure 2.2: Pre-image trees p_1^W, p_2^W such that $p_1^W|_W \neq p_2^W|_W$ for some essential v -subgraph W but $p_1^W|_X \doteq p_2^W|_X$ for every v -subgraph X properly contained in W . If $|W|$ is even then $|W| = 2l$ and $m = l - 1$, otherwise $|W| = 2m - 1$ and $m = l$.

any v -subgraph Z contained in X , a copy c^Y belongs to the partition class Z if and only if Z is the ones-subgraph of $c^Y|_X$, that is:

$$Y = Z \cup Y' \text{ with } Y' \in \mathcal{P}(W - X).$$

In words, apart from those walks contained in Z , all other walks in copy c^Y which have been assigned the value “1” are contained in W but not contained in X . There are

$$\sum_{k=0}^{|W|-|X|} \binom{|W|-|X|}{k}$$

distinct copies of U which belongs to the partition class Z . Particularly, there are $\binom{|W|-|X|}{k}$ distinct copies of U whose ones-subgraph is of size $|Z| + k$ for $k = 0, \dots, |W| - |X|$.

The binomial coefficients form a complete row of Pascal’s Triangle. A well-known property of Pascal’s Triangle is that the alternate sum of every row equals 0. This means, for a row with n coefficients:

$$0 = \sum_{k=0}^n (-1)^k \binom{n}{k}$$

This is immediate upon substituting $x = 1$ and $y = -1$ into the Binomial Theorem

$$(x + y)^n = \sum_{k=0}^n x^{n-k} y^k \binom{n}{k}$$

The property states that the sum of all binomial coefficients where k is odd is equal to the sum of all binomial coefficients where k is even. As a consequence we obtain that the following two sets contain the same number of copies of U :

- the set of all copies of U belonging to the partition class Z whose ones-subgraph is of an odd size (i.e., contained in p_1^W)
- the set of all copies of U belonging to the partition class Z whose ones-subgraph is of an even size (i.e., contained in p_2^W)

Any bijection f_X^Z of the first set to the second will satisfy the condition that $c|_X \doteq f_X^Z(c)|_X$ for every copy c of U contained in the first set. Every copy of U has a unique ones-subgraph for its projection to X . Thus a bijection f_X , as described previously, simply combines the mappings from the individual bijection f_X^Z for all v -subgraph Z contained in X . The process of finding the bijection f_X is illustrated in Example 2.16. \square

The next two examples try to clarify the proof of the previous lemma. The first, Example 2.15, demonstrates how to apply the proposed construction. While the second, Example 2.16, illustrates the process of finding a bijection f_X between the copies of U contained in pre-image trees p_1^W, p_2^W , such that we can verify $p_1^W|_X \doteq p_2^W|_X$ for any v -subgraph X which is properly contained in W .

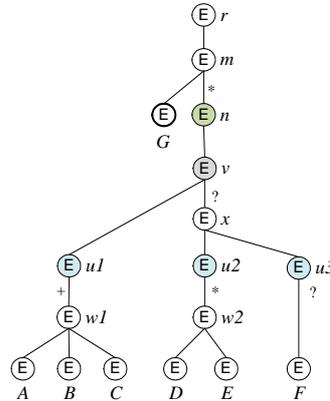


Figure 2.3: XML schema tree T_{abstr} containing three v -units: $U_1 = ABC$, $U_2 = DE$, $U_3 = F$

Example 2.15. Consider the XML schema tree T_{abstr} depicted in Figure 2.3. For the essential v -subgraph $W_1 = AB$, Figure 2.4 portrays the two U_1 -compatible pre-image trees $p_1^{W_1}, p_2^{W_1}$ resulting from applying the construction set out in the proof of Lemma 2.14, such that

$$p_1^{W_1}|_X \doteq p_2^{W_1}|_X \text{ if and only if } v\text{-subgraph } X \text{ is properly contained in } W_1$$

We first generate four copies of v -unit $U_1 = ABC$, one for each of the v -subgraph contained in AB . For emphasis, we highlight leaves which have been assigned “1” by a bold

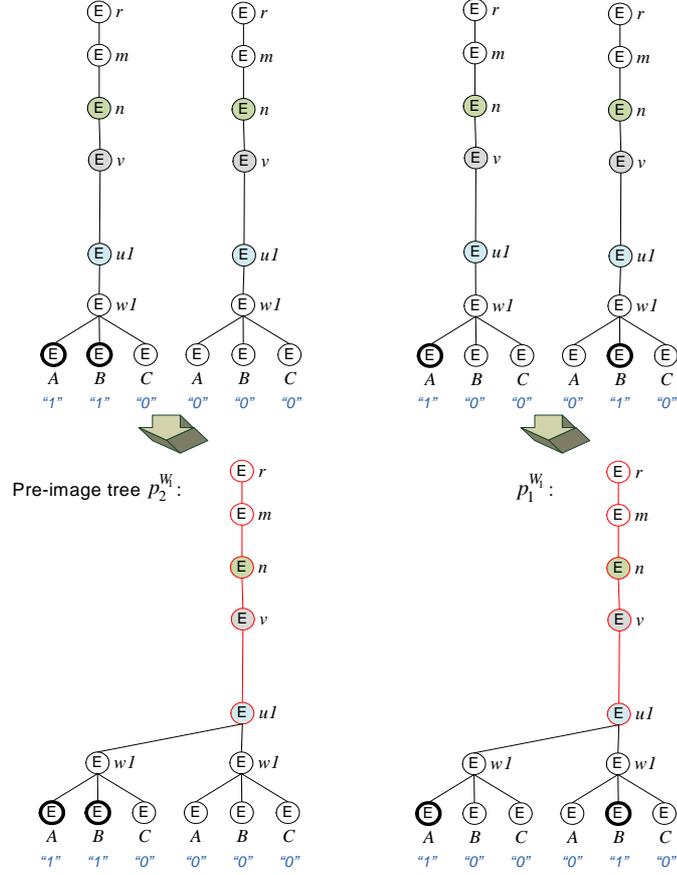


Figure 2.4: Pre-image trees $p_1^{W_1}, p_2^{W_1}$ such that $p_1^{W_1}|_X \doteq p_2^{W_1}|_X$ if and only if v -subgraph X is properly contained in $W_1 = AB$.

circle. Those copies whose ones-subgraphs have an odd number of v -walks are merged to form $p_1^{W_1}$ and, those copies whose ones-subgraphs have an even number of v -walks are merged to form $p_2^{W_1}$. \square

Example 2.16. Consider a v -subgraph $W = ABCDE$ in some schema tree T with W contained in the v -unit $U = ABCDEF$. The ones-subgraphs for the copies of U which are contained in the pre-image trees p_1^W or p_2^W are as follows:

$$\begin{aligned} p_1^W: & ABCDE, ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE, CDE, A, B, C, D, E. \\ p_2^W: & ABCD, ABCE, ABDE, ACDE, BCDE, AB, AC, AD, AE, BC, BD, BE, CD, CE, DE, \emptyset. \end{aligned}$$

Let us find the bijection f_{ABD} showing $p_1^W|_{ABD} \doteq p_2^W|_{ABD}$. All subgraphs contained in ABC are as follows: $\emptyset, A, B, D, AB, AD, BD, ABD$.

The following copies of U belongs to the partition class $Z = \emptyset$, i.e. their projection to ABD gives the \emptyset as the ones-subgraph: \emptyset, C, E, CE . This set can be grouped into $\{\emptyset, CE\}$ and $\{C, E\}$ according to whether the size of the ones-subgraph is even or odd. One bijection f_{ABD}^\emptyset between these two sets is: $\emptyset \mapsto C, CE \mapsto E$. Alternatively, the bijection (f_{ABD}^\emptyset) can also be specified as: $\emptyset \mapsto E, CE \mapsto C$.

The following copies of U belongs to the partition class $Z = A$: A, AC, AE, ACE . From this we obtain the grouping $\{A, ACE\}$ and $\{AC, AE\}$ and a bijection f_{ABD}^A as follows: $A \mapsto AC, ACE \mapsto AE$.

Continuing in the same manner, we obtain bijections $f_{ABD}^B, f_{ABD}^D, f_{ABD}^{AB}, f_{ABD}^{AD}, f_{ABD}^{BD}, f_{ABD}^{ABD}$. Combining all the individual maplets from these bijections, we obtain the following bijection f_{ABD} between the copies of U contained in p_1^W and the copies of U contained in p_2^W :

$$\begin{array}{cccccccc}
C & E & A & ACE & B & BCE & D & CDE & ABC & ABE & ACD & ADE & BCD & BDE & ABD & ABCDE \\
\downarrow & \downarrow \\
\emptyset & CE & AC & AE & BC & BE & CD & DE & AB & ABCE & AD & ACDE & BD & BCDE & ABCD & ABDE \\
\widehat{f_{ABD}^0} & & \widehat{f_{ABD}^A} & & \widehat{f_{ABD}^B} & & \widehat{f_{ABD}^D} & & \widehat{f_{ABD}^{AB}} & & \widehat{f_{ABD}^{AD}} & & \widehat{f_{ABD}^{BD}} & & \widehat{f_{ABD}^{ABD}} &
\end{array}$$

□

By repeating the previous construction and merging the resulting data trees, we can obtain, for each v -unit U , two U -compatible pre-image trees of v which minimally differ on each v -subgraph in $\mathcal{NE}_T|_U$.

```

proc construct-pre-image-trees_one-unit( $\mathcal{NE}_T|_U, U$ )
  Initialise  $p_1^U, p_2^U$  as empty XML trees
  if  $\mathcal{NE}_T|_U = \{\}$  then
    Create a copy  $c$  of  $U$  whose ones-subgraph is  $U$ 
    return  $c, c$  //  $p_1^U = p_2^U = c$ 
  end if
  for all  $v$ -subgraph  $W \in \mathcal{NE}_T|_U$  do
     $p_1^W, p_2^W =$ construct-pre-image-trees_one-subgraph( $W, U$ )
    for  $i = 1, 2$  do
       $p_i^U =$ Merge  $p_i^W$  with  $p_i^U$  on  $\eta(U)$ 
    end for
  end for
  return  $p_1^U, p_2^U$ 

```

There can only be more than one member in $\mathcal{NE}_T|_U$ whenever U consists of more than one v -walks and we have again some proper descendant $\eta(U)$ of v with incoming arc having frequency other than ? and 1. Therefore we can merge the pre-image trees $p_i^{W_j}$ with $j = 1, \dots, |\mathcal{NE}_T|_U|$ such they share every node except their pre-images of $\eta(U)$ and its descendants.

Lemma 2.17. *Let U be a v -unit contained in some schema tree T and \mathcal{NE}_T a non-equality set of v -properties. For a projected non-equality set $\mathcal{NE}_T|_U$, there are two U -compatible pre-image trees p_1^U, p_2^U of v such that*

$$p_1^U|_X \doteq p_2^U|_X \text{ if and only if } v\text{-subgraph } X \text{ is properly contained in some } W \in \mathcal{NE}_T|_U$$

Proof. Consider the output from `construct-pre-image-trees-one-unit`. It is easy to see that the resulting pre-image trees p_1^U, p_2^U are U -compatible and $p_1^U|_{W_j} \neq p_2^U|_{W_j}$ for every $W_j \in \mathcal{NE}_T|_U$ because the merge keeps each $p_i^{W_j}$ independent. This is possible because the v -subgraphs belonging to $\mathcal{NE}_T|_U$ are pairwise incomparable with respect to subgraph containment. Finally we must show that $p_1^U|_X \doteq p_2^U|_X$ for every X which is properly contained in some $W \in \mathcal{NE}_T|_U$. Observe that there can be no v -subgraph X which is properly contained in some $W \in \mathcal{NE}_T|_U$ and yet contain some other $W' \in \mathcal{NE}_T|_U$. Therefore a bijection showing $p_1^U|_X \doteq p_2^U|_X$ is a simple union of the bijections f_X^j between the pre-image trees $p_1^{W_j}, p_2^{W_j}$ for all $W_j \in \mathcal{NE}_T|_U$ (see Lemma 2.14). \square

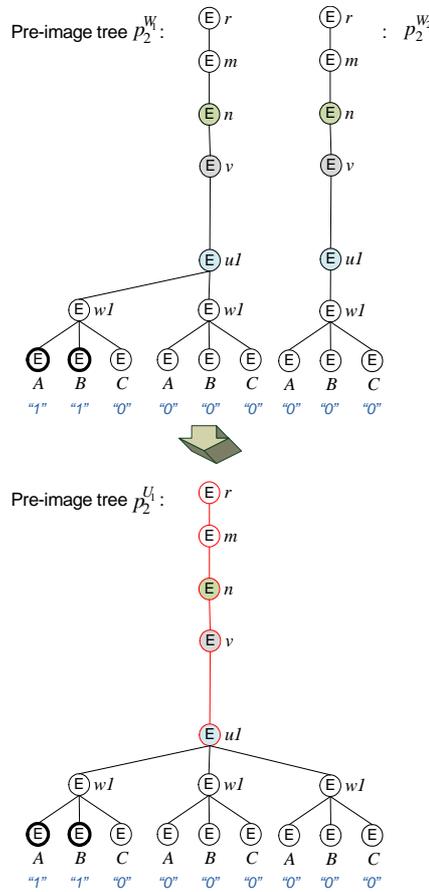


Figure 2.5: Pre-images $p_2^{W_1}, p_2^{W_2}$ merged to form $p_2^{U_1}$.

We now illustrate the construction from the previous lemma, for a non-equality set projected to some v -unit, with an example.

Example 2.18. Continuing from Example 2.15. Suppose we have non-equality set projection $\mathcal{NE}_T|_{U_1} = \{AB, C\}$. Let $W_1 = AB$ and $W_2 = C$.

To construct a pre-image tree $p_2^{U_1}$ we first generate U_1 -compatible data trees $p_2^{W_1}$ and $p_2^{W_2}$ by applying the construction from Lemma 2.14 twice. These two data trees are

shown in Figure 2.5. The figure also show the U_1 -compatible data tree $p_2^{U_1}$ resulting from the merge of $p_2^{W_1}$ and $p_2^{W_2}$. The v -unit U_1 has multiple walks and unique identifier node $\eta(U_1) = u_1$. Therefore, in $p_2^{U_1}$, we see that $p_2^{W_1}$ and $p_2^{W_2}$ share exactly those pre-images of nodes on the rooted path to u_1 . The data trees $p_1^{W_1}$ and $p_1^{W_2}$ are similarly merged so that they share only their pre-images of nodes on the rooted path to u_1 . \square

Finally we show how to combine the subtrees constructed for each v -unit to form a T -compatible data tree containing precisely two pre-image trees which agree on precisely those v -subgraphs belonging to some given equality set \mathcal{E}_T . In addition to the subtrees obtained by applications of Lemma 2.19, one further subtree is needed for ensuring T -compatibility - this additional subtree constitutes of walks that do not contain v . We construct two T -compatible pre-image trees and then merge them, with the guide of the single non-simple essential v -ancestor in $\mathcal{N}\mathcal{E}_T$. Note that in the absence of such an input v -ancestor (e.g., when considering HL-XFDs), we can use any node on the rooted path to v with incoming arc of frequency other than ? and 1.

```

proc construct-pre-image-trees_data-tree( $\mathcal{N}\mathcal{E}_T$ )
  Initialise  $T'$  as empty XML trees
  if  $\mathcal{N}\mathcal{E}_T \cap \mathbf{E}_T^{\check{S}}(v) = \{\}$  then
    Create copies  $p_1, p_2$  of  $T$  whose ones-subgraph is  $T$ 
  else
    Initialise  $p_1, p_2$  as empty XML trees
    for all  $v$ -unit  $U \in \mathcal{U}_T(v)$  do
       $p_1^U, p_2^U = \text{construct-pre-image-trees-one-unit}(\mathcal{N}\mathcal{E}_T|_U, U)$ 
      for  $i = 1, 2$  do
         $p_i = \text{Merge}^{(a)} p_i^U$  with  $p_i$ 
      end for
    end for
     $Z = \text{all walks of } T \text{ that does not contain } v$ 
    Create a copy  $p'$  of  $Z$  whose ones-subgraph is  $\emptyset$ .
    for  $i = 1, 2$  do
       $p_i = \text{Merge}^{(b)} p'$  with  $p_i$ 
    end for
  end if
   $T' = \text{Merge } p_1 \text{ with } p_2 \text{ on } n \in \mathcal{N}\mathcal{E}_T \cap \mathbf{E}_T^{\check{A}}(v)$ 
  return  $T'$ 

```

The previous algorithms make use of three variations of merge. To form pre-image tree p_i (for $i = 1, 2$) we first merge the U_k -compatible data tree $p_i^{U_k}$ for every v -unit U_k such that for any two distinct v -units U_1, U_2 , the two data trees $p_i^{U_1}, p_i^{U_2}$ share *only those nodes which are pre-images of nodes shared by the v -units U_1, U_2* . Then we merge each resulting data tree p_i (for $i = 1, 2$) with p' such that the partial data tree p_i and p' share *as many arcs as possible*. Finally, to form data tree T' , we merge pre-image trees p_1, p_2 such that p_1, p_2 share *every node except their respective pre-images of $n \in \mathcal{N}\mathcal{E}_T \cap \mathbf{E}_T^{\check{A}}(v)$ and its descendants*.

Lemma 2.19. *Let T be an XML schema tree with a non-simple node v . Given a non-equality set of v -properties \mathcal{NE}_T , with corresponding equality set \mathcal{E}_T , there is a T -compatible data tree T' consisting of precisely two pre-image trees p_1, p_2 of v such that*

$$p_1|_X \doteq p_2|_X \text{ if and only if } X \in \mathcal{E}_T, \text{ for all } v\text{-property } X \in \check{\mathcal{S}}_T(v) \cup \check{\mathcal{A}}_T(v)$$

Proof. Consider the output from `construct-pre-image-trees-data-tree`. It is clear that the resulting pre-image trees p_1, p_2 are T -compatible and

$$\begin{aligned} p_1|_X \doteq p_2|_X & \text{ if and only if } X \text{ is properly contained in some } W \in \mathcal{NE}_T|_U \\ & \text{ for some } v\text{-unit } U \in \mathcal{U}_T(v) \\ & \text{ if and only if } X \text{ is properly contained in some } W \in \mathcal{NE}_T \cap \check{\mathcal{S}}_T(v) \\ & \text{ if and only if } X \in \mathcal{E}_T, \text{ for all } v\text{-property } X \in \check{\mathcal{S}}_T(v) \end{aligned}$$

Finally we address the matter of ensuring property-equality on specified v -ancestors. For conciseness, let n be the single essential v -ancestor in \mathcal{NE}_T . To form data tree T' , we merge pre-image trees p_1, p_2 such that p_1, p_2 share every node except their respective pre-images of n and its descendants. With n being a non-simple essential v -ancestor, this final merging results in a T -compatible data tree which contains precisely the two pre-image trees p_1, p_2 and moreover

$$\begin{aligned} p_1|_X \doteq p_2|_X & \text{ if and only if } X \text{ is a proper ancestor of } n \in \mathcal{NE}_T \cap \check{\mathcal{A}}_T(v) \\ & \text{ if and only if } X \in \mathcal{E}_T, \text{ for all } v\text{-ancestor } X \in \check{\mathcal{A}}_T(v) \end{aligned}$$

Thus concludes the proof. \square

Another example demonstrates the final steps in the construction as outlined in Lemma 2.19.

Example 2.20. Recall the schema tree T_{abstr} from Figure 2.3. For the sake of clarity, in the rest of this example, we denote T_{abstr} simply by T . Suppose that for the input we are given non-equality set $\mathcal{NE}_T = \{AB, C, DE, n\}$.

From this we get: $\mathcal{NE}_T|_{U_1} = \{AB, C\}$, $\mathcal{NE}_T|_{U_2} = \{DE\}$, and $\mathcal{NE}_T|_{U_3} = \emptyset$. In Example 2.18, we have already considered the construction step involving v -unit U_1 . Similarly, we can consider the remaining two v -units and generate U_k -compatible data trees $p_1^{U_k}, p_2^{U_k}$ with $k = 2, 3$. In addition, there is one walk G which does not contain v , and so, we also generate data trees p'_1, p'_2 which constitute a copy of the walk G whose leaf is assigned the value “0”.

Now we can start assembling pre-image trees p_i where $i = 1, 2$. Since U_1 and U_2 shares the rooted path up to v , we merge the pre-image $p_i^{U_1}$ and $p_i^{U_2}$ in such a way that they share only the pre-images of those nodes on the rooted path to v . Let the resulting subtree be $p_i^{U_{12}}$. Next U_3 shares the rooted path to x with $U_1 \sqcup U_2$, hence we then merge $p_i^{U_3}$ so that $p_i^{U_3}$ and $p_i^{U_{12}}$ only share the pre-image of those nodes on the rooted path to x . Let the resulting subtree be $p_i^{U_{123}}$. Then we merge p'_i with $p_i^{U_{123}}$. The walk G shares the rooted path to m with the rooted path to v . So after merging, p'_i and $p_i^{U_{123}}$ only share the pre-images of m and the root. This gives T_{abstr} -compatible pre-images p_1, p_2 .

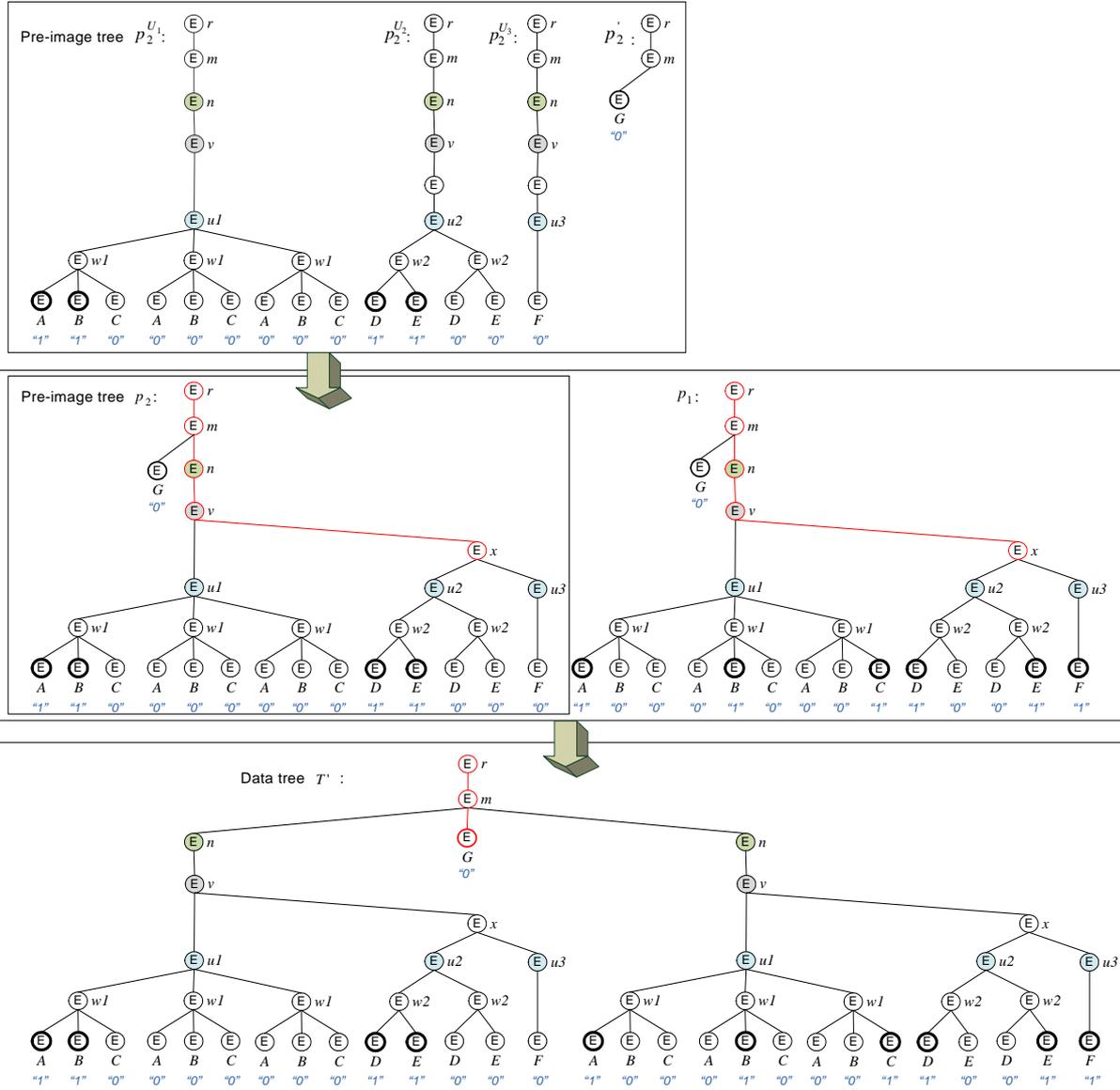


Figure 2.6: Pre-images $p_2^{U_1}, p_2^{U_2}, p_2^{U_3}$ merged to form p_2 . Then p_1, p_2 merged to form T_{abstr} -compatible data tree T'_{abstr} .

To obtain the two- v -pre-image data tree we merge p_1, p_2 such that they share every node except their respective pre-images of n and its descendants. The pre-images p_1, p_2 and the resulting data tree are shown in Figure 2.6. \square

The main focus of this subsection has been to present a construction for a T -compatible two- v -pre-image data tree such that its two pre-image trees of v agree on precisely those v -properties given in some equality set \mathcal{E}_T . The construction is an essential part of a semantic proof for establishing the Semantic Equivalence Theorem for pXFDs. Particularly, we relied on the construction in the proof of Proposition 2.6.

2.2 Application of the Semantic Equivalence

For the remainder of the chapter, we demonstrate some useful applications of the Semantic Equivalence Theorem for pXFDs (Theorem 2.7). Particularly we apply the equivalence in:

1. Deciding the implication problem for pXFDs;
2. Finding and proving a sound and complete system of inference rules for pXFDs;
3. Supporting pXFDs acquisition.

2.2.1 Deciding the Implication Problem for pXFDs

Let $\Sigma \cup \{\sigma\}$ be a set of pXFDs over some XML schema tree T . The *implication problem* poses the question of whether $\Sigma \models \sigma$ holds for an arbitrary set $\Sigma \cup \{\sigma\}$ of pXFDs over T .

Through the Semantic Equivalence Theorem for pXFDs, the implication problem can be answered by considering whether H_σ is a logical consequence of $H_\Sigma \cup H_T$. Specifically, $\Sigma \models \sigma$ holds unless H_σ is *not* a logical consequence of $H_\Sigma \cup H_T$. In other words, we raise the question:

For some $h \in H_\sigma$, is there a boolean assignment \mathbb{B} of the propositional variables such that every propositional formula in the set $H_\Sigma \cup H_T \cup \{\neg h\}$ evaluates to *true*.

This is in fact the problem of *Horn-satisfiability* (*Horn-SAT*) which is known to be decidable in linear time. It follows that:

Corollary 2.21. *The problem of whether Σ implies σ can be decided in time linear in the total number of essential v -properties of T .*

For completeness, we outline an algorithm for solving Horn-SAT based on the rule of unit propagation [21]. Recall that a clause is a disjunction of literals. A *unit clause* is a clause which consists of a single literal. The rules of *unit propagation* are as follows:

- (i) if the formula contains a unit clause l , then all clauses containing l are removed, and

- (ii) all clauses containing $\neg l$ have the literal $\neg l$ removed.

The basic idea is to exploit these rules and simplify a Horn formula Φ until we find either a contradictory pair of literals l and $\neg l$, or there is no further unit clauses to propagate.

Algorithm 2.1 Horn_SAT(unit propagation)

Input: Horn formula Φ

Output: *true* if Φ is satisfiable, *false* otherwise

1. **while** Φ contains a unit clause l **do**
 2. **if** Φ contains some unit clauses k and $\neg k$ **then**
 3. **return** *false*
 4. **end if**
 5. Remove from Φ all clauses containing l
 6. Remove from Φ literal $\neg l$ from all remaining clauses
 7. **end while**
 8. **return** *true*
-

Next, we exemplify how to decide the implication problem by considering Horn-SAT.

Example 2.22. Recall the following pXFDs over the dance school schema tree T_{dance} :

$$\begin{array}{ll}
 v_{partners} : \{v_{class}, \mathbf{boy}\} \rightarrow \{\mathbf{girl}\} & (pXFD1) \\
 v_{partners} : \{v_{class}, \mathbf{girl}\} \rightarrow \{\mathbf{boy}\} & (pXFD2) \\
 v_{partners} : \{\{\mathbf{boy}, \mathbf{girl}\}\} \rightarrow \{v_{class}\} & (pXFD6) \\
 v_{partners} : \{\mathbf{boy}\} \rightarrow \{\mathbf{girl}\} & (pXFD3)
 \end{array}$$

Suppose we want to know whether pXFD3 follows from pXFD1, pXFD2 and pXFD6. In Example 2.1, we have presented the base translation $H_{T_{dance}}$ together with the Horn clauses corresponding to these four pXFDs. In particular, H_σ results from the translation of pXFD3 and H_Σ results from the translation of the set containing pXFD1, pXFD2 and pXFD6' (i.e., the canonical pXFD for pXFD6).

Next we check whether there is a boolean assignment which makes all formulas in $H_\Sigma \cup H_{T_{dance}}$ *true* and the single formula $h \in H_\sigma$ *false*. This is equivalent to checking the satisfiability of the following Horn formula

$$\begin{array}{l}
 H_\Sigma \left\{ \begin{array}{l}
 (\neg\varphi(v_{class}) \vee \neg\varphi(\mathbf{boy}) \vee \varphi(\mathbf{girl})) \\
 \wedge (\neg\varphi(v_{class}) \vee \neg\varphi(\mathbf{girl}) \vee \varphi(\mathbf{boy})) \\
 \wedge (\neg\varphi(\mathbf{boy}) \vee \neg\varphi(\mathbf{girl}) \vee \varphi(v_{class}))
 \end{array} \right. \\
 \hline
 H_{T_{dance}} \left\{ \begin{array}{l}
 \wedge (\neg\varphi(v_{partners}) \vee \varphi(v_{session})) \\
 \wedge (\neg\varphi(v_{session}) \vee \varphi(v_{class})) \\
 \wedge (\neg\varphi(v_{partners}) \vee \varphi(\mathbf{boy})) \\
 \wedge (\neg\varphi(v_{partners}) \vee \varphi(\mathbf{girl})) \\
 \wedge \varphi(\emptyset) \\
 \wedge \varphi(v_{danceschool})
 \end{array} \right. \\
 \hline
 \neg h \left\{ \begin{array}{l}
 \wedge \varphi(\mathbf{boy}) \\
 \wedge \neg\varphi(\mathbf{girl})
 \end{array} \right.
 \end{array}$$

The process of unit propagation yields:

- after propagation of $\varphi(\mathbf{boy})$

$$\begin{aligned} & (\neg\varphi(v_{class}) \vee \varphi(\mathbf{girl})) \wedge (\neg\varphi(\mathbf{girl}) \vee \varphi(v_{class})) \\ & \wedge (\neg\varphi(v_{partners}) \vee \varphi(v_{session})) \wedge (\neg\varphi(v_{session}) \vee \varphi(v_{class})) \\ & \quad \wedge (\neg\varphi(v_{partners}) \vee \varphi(\mathbf{girl})) \\ & \quad \wedge \varphi(\emptyset) \wedge \varphi(v_{danceschool}) \\ & \wedge \neg\varphi(\mathbf{girl}) \end{aligned}$$

- after propagation of $\neg\varphi(\mathbf{girl})$

$$\begin{aligned} & \neg\varphi(v_{class}) \\ & \wedge (\neg\varphi(v_{partners}) \vee \varphi(v_{session})) \wedge (\neg\varphi(v_{session}) \vee \varphi(v_{class})) \\ & \quad \wedge \neg\varphi(v_{partners}) \\ & \quad \wedge \varphi(\emptyset) \wedge \varphi(v_{danceschool}) \end{aligned}$$

- Continuing in the same manner we end up removing all literals from the original Horn formula.

This shows that the original Horn formula is satisfied. Moreover, unit propagation allows us to determine the following counter-example boolean assignment:

$$\begin{array}{ll} \mathbb{B}(\varphi(\emptyset)) = true & \mathbb{B}(\varphi(v_{partners})) = false \\ \mathbb{B}(\varphi(v_{danceschool})) = true & \mathbb{B}(\varphi(v_{class})) = false \\ \mathbb{B}(\varphi(\mathbf{boy})) = true & \mathbb{B}(\varphi(v_{session})) = false \\ \mathbb{B}(\varphi(\mathbf{girl})) = false & \end{array}$$

It follows that H_σ is not a logical consequence of $H_\Sigma \cup H_{T_{dance}}$ and we can correspondingly conclude that Σ does not imply σ . \square

Alternatively, the implication problem can also be answered by identifying a sound and complete system \mathfrak{F} of inference rules and checking whether $\Sigma \vdash_{\mathfrak{F}} \sigma$. In comparison with the Semantic Equivalence Theorem for pXFDs, an axiomatisation for pXFD implication also allows us to develop algorithms for enumerating all implied pXFDs. Next we explore several axiomatisations for pXFDs. Our starting point is to exploit the Semantic Equivalence Theorem for pXFDs in order to identify a sound and complete set of inference rules for the implication of canonical pXFDs.

$\frac{}{v : \mathcal{X} \rightarrow \mathcal{Y}} \mathcal{Y} \subseteq \mathcal{X}$ <p>(reflexivity)</p>	$\frac{v : \mathcal{X} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{X} \cup \mathcal{Y}}$ <p>(extension)</p>	$\frac{v : \mathcal{X} \rightarrow \mathcal{Y}, v : \mathcal{Y} \rightarrow \mathcal{Z}}{v : \mathcal{X} \rightarrow \mathcal{Z}}$ <p>(transitivity)</p>
$\frac{}{v : \{X\} \rightarrow \{Y\}} Y \text{ is contained in } X$ <p>(subgraph)</p>	$\frac{}{v : \{\} \rightarrow \{r_T\}}$ <p>(root)</p>	
$\frac{}{v : \{n\} \rightarrow \{m\}} m \text{ is an ancestor of } n$ <p>(ancestor)</p>	$\frac{}{v : \{\} \rightarrow \{\emptyset\}}$ <p>(empty subgraph)</p>	
$\frac{}{v : \vartheta(\{v\}) \rightarrow \{U\}} U \text{ is a } v\text{-unit}$ <p>(target)</p>		

Table 2.1: System $\mathfrak{F}_{canonical}$ of inference rules for (canonical) pXFDS

2.2.2 Sound and Complete Axiomatisation

Axiomatisation of Canonical pXFDS

Recall Fagin's syntactic proof illustrated in Figure 2.1. In a syntactic proof for a Semantic Equivalence Theorem, we would identify an axiomatisation for implication from which to prove a Completeness Theorem for Formulas. From a slightly different perspective we obtain an alternative approach to proving the correctness of an axiomatisation for the class of constraints involved in the semantic equivalence - this has been remarked upon in [78]. We can apply the Semantic Equivalence Theorem and Completeness Theorem for Formulas to support the soundness and completeness of a system of inference rules.

We also get a clue as to which inference rules are sound from the Semantic Equivalence Theorem for pXFDS and proposed Horn encoding. In [24], Fagin has effectively shown that any set of inference rules for the implication of FDs which is equivalent to the Armstrong Axioms translates to a set of rules which is sound and complete for the logical implication of propositional Horn clauses. This is the foundation for our first system of inference rules. Additional axioms arise from propositional Horn clauses in the base translation H_T . Next we present some inference rules for pXFDS and prove a Completeness Theorem for Formulas with respect to these rules.

Theorem 2.23 (Completeness Theorem for Formulas (and pXFDS)). *Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be sets of essential v -properties, X, Y, U be essential v -subgraphs, and n, m be essential v -ancestors. Let $\Sigma \cup \{\sigma\}$ be a set of canonical pXFDS where σ has the form $v : LHS_\sigma \rightarrow RHS_\sigma$. Then H_σ is a logical consequence of $H_\Sigma \cup H_T$ if and only if $\Sigma \vdash_{\mathfrak{F}_{canonical}} \sigma$, with $\mathfrak{F}_{canonical}$ being the system of inference rules from Table 2.1*

Proof. (\Rightarrow) Suppose H_σ is logically implied by $H_\Sigma \cup H_T$. Finding a derivation sequence from the logical implication is straightforward:

- If the logical implication involve one of the translated Armstrong Axioms then we apply the corresponding Armstrong inference rule from our system (i.e., reflexivity, extension, transitivity).
- If the logical implication involve some Horn clause $h \in H_T$ then we apply the inference rule corresponding to the case which gave rise to the formula as follows:

Case 1: $h \in \{\varphi(W) \Rightarrow \varphi(Z) \mid W, Z \in \mathbf{E}_T^{\check{S}}(v) \text{ and } W \text{ covers } Z\}$

Apply the subgraph axiom.

Case 2: $h \in \{\varphi(n) \Rightarrow \varphi(m) \mid n, m \in \mathbf{E}_T^{\check{A}}(v) \text{ and } m \text{ is an immediate essential ancestor of } n\}$

Apply the ancestor axiom

Case 3: $h \in \{\varphi(n) \Rightarrow \varphi(U) \mid \{n\} = \vartheta(\{v\}) \text{ and } U \text{ is a } v\text{-unit}\}$

Apply the target axiom.

Case 4: $h \in \{\varphi(\emptyset), \varphi(r_T)\}$

Apply the empty subgraph axiom or root axiom respectively.

It is easy to see that in each case, the pre-condition for the corresponding inference rule is satisfied and the inference rule can in fact be applied.

(\Leftarrow) The proof involves showing that if $\Sigma \vdash_{\mathfrak{F}_{canonical}} \sigma$ then there is a logical implication of H_σ from $H_\Sigma \cup H_T$. Given a derivation of σ from Σ using $\mathfrak{F}_{canonical}$, we can find a logical implication of H_σ from $H_\Sigma \cup H_T$ as follows:

- an application of an Armstrong inference rule from our system (reflexivity, extension or transitivity) yields an application of the corresponding translated Armstrong Axiom for propositional Horn clauses;
- an application of the subgraph axiom where $RHS_\sigma = LHS_\sigma$ can be transformed into one which applies the reflexivity rule. On the other hand, an application of the subgraph axiom where $RHS_\sigma = \{\emptyset\}$ can be transformed into a derivation involving the reflexivity rule and empty subgraph axiom followed by the transitivity rule. In the remaining cases, we get a corresponding logical implication consisting of repeated applications of the translated transitivity rule over a sequence

$$\varphi(X_{\sigma_1}) \Rightarrow \varphi(Y_{\sigma_1}), \varphi(X_{\sigma_2}) \Rightarrow \varphi(Y_{\sigma_2}), \dots, \varphi(X_{\sigma_n}) \Rightarrow \varphi(Y_{\sigma_n})$$

of Horn clauses in H_T whereby

- $X_{\sigma_1} = LHS_\sigma$ and $Y_{\sigma_n} = RHS_\sigma$, and
- $Y_{\sigma_i} = X_{\sigma_{i+1}}$ for $i = 1, \dots, n - 1$, and

- X_{σ_j} covers Y_{σ_j} for $j = 1, \dots, n$

The pre-condition of the subgraph axiom states RHS_σ is contained in LHS_σ . Successively removing from RHS_σ a walk belonging to LHS_σ but not belonging to RHS_σ gives a sequence $X_{\sigma_1}, Y_{\sigma_1}, Y_{\sigma_2}, \dots, Y_{\sigma_n}$ of v -subgraphs as required in the Horn clauses above.

- an application of the ancestor axiom where $RHS_\sigma = LHS_\sigma$ is equivalent to a derivation comprising of an application of the reflexivity rule. Likewise, an application of the ancestor axiom where $RHS_\sigma = \{r_T\}$ is equivalent to one involving an application of the reflexivity rule, root axiom and transitivity rule. In the remaining cases, we get a corresponding logical implication consisting of repeated applications of the translated transitivity rule over a sequence

$$\varphi(X_{\sigma_1}) \Rightarrow \varphi(Y_{\sigma_1}), \varphi(X_{\sigma_2}) \Rightarrow \varphi(Y_{\sigma_2}), \dots, \varphi(X_{\sigma_n}) \Rightarrow \varphi(Y_{\sigma_n})$$

of Horn clauses in H_T whereby

- $X_{\sigma_1} = LHS_\sigma$ and $Y_{\sigma_n} = RHS_\sigma$, and
- $Y_{\sigma_i} = X_{\sigma_{i+1}}$ for $i = 1, \dots, n - 1$, and
- X_{σ_j} is an immediate essential descendant of Y_{σ_j} for $j = 1, \dots, n$

The pre-condition of the ancestor axiom requires that RHS_σ is an ancestor of LHS_σ . The sequence $X_{\sigma_1}, Y_{\sigma_1}, Y_{\sigma_2}, \dots, Y_{\sigma_n}$ in the Horn clauses above corresponds to the path from LHS_σ to RHS_σ .

- an application of the target axiom corresponds to a Horn clause $\varphi(n) \Rightarrow \varphi(U) \in H_T$ where $\{n\} = \vartheta(\{v\})$ and U is a v -unit of T .
- an application of the empty subgraph axiom or root axiom corresponds to the Horn clause $\varphi(\emptyset)$ or $\varphi(r_T)$ in H_T respectively.

□

The previous theorem together with the Semantic Equivalence Theorem for pXFDs gives us the proof for soundness and completeness of $\mathfrak{F}_{canonical}$.

Theorem 2.24 (Canonical pXFDs Axiomatisation, $\mathfrak{F}_{canonical}$). *Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be sets of essential v -properties, X, Y, U be essential v -subgraphs, and n, m be essential v -ancestors. The system $\mathfrak{F}_{canonical}$ of inference rules from Table 2.1 is sound and complete for the implication of canonical pXFDs.*

Proof. Follows from the Semantic Equivalence Theorem for pXFDs (Theorem 2.7) and the Completeness Theorem for Formulas and pXFDs (Theorem 2.23). □

Note that we have not considered the original set of Armstrong Axioms (i.e., reflexivity, augmentation and transitivity rule [67]) nor the equivalent set of inference rules used

by Fagin (reflexivity, union, decomposition and transitivity [24]), But in fact, any axiomatisation which is equivalent to the Armstrong Axioms will suffice. The following are some useful additional inference rules which can be derived from our Armstrong inference rules.

Lemma 2.25. *Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be sets of essential v -properties. The following inference rules are sound for the implication of canonical pXFDs:*

$$\begin{array}{ccc} \frac{v : \mathcal{X} \rightarrow \mathcal{Y}}{v : \mathcal{X} \cup \mathcal{Z} \rightarrow \mathcal{Y} \cup \mathcal{Z}} & \frac{v : \mathcal{X} \rightarrow \mathcal{Y}, v : \mathcal{X} \rightarrow \mathcal{Z}}{v : \mathcal{X} \rightarrow \mathcal{Y} \cup \mathcal{Z}} & \frac{v : \mathcal{X} \rightarrow \mathcal{Y} \cup \mathcal{Z}}{v : \mathcal{X} \rightarrow \mathcal{Y}, v : \mathcal{X} \rightarrow \mathcal{Z}} \\ \text{(augmentation)} & \text{(union)} & \text{(decomposition)} \end{array}$$

Proof. Similar to the proofs of analogous inference rules for FDs in the relational data model. For example, see [77, 89]. \square

Adding/removing the empty v -subgraph and root to the LHS and/or RHS of a pXFD σ results in a pXFD which is implied by σ . This is intuitively true because any two pre-image trees agree on the empty v -subgraph and the root. This gives us four simple inference rules.

Lemma 2.26. *Let \mathcal{X}, \mathcal{Y} be sets of essential v -properties. The following inference rules are sound for the implication of canonical pXFDs:*

$$\begin{array}{cc} \frac{v : \mathcal{X} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\emptyset\}} & \frac{v : \mathcal{X} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y} \cup \{r_T\}} \\ \text{(supplement empty subgraph)} & \text{(supplement root)} \\ \\ \frac{v : \mathcal{X} \cup \{\emptyset\} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y}} & \frac{v : \mathcal{X} \cup \{r_T\} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y}} \\ \text{(trim empty subgraph)} & \text{(trim root)} \end{array}$$

Proof. Let us first consider the supplement empty subgraph rule. A simple derivation tree proof is as follows:

$$\frac{v : \mathcal{X} \rightarrow \mathcal{Y} \quad \frac{\frac{}{v : \mathcal{X} \rightarrow \{\}} \text{(1)} \quad \frac{}{v : \{\} \rightarrow \{\emptyset\}} \text{(2)}}{v : \mathcal{X} \rightarrow \{\emptyset\}} \text{(3)}}{v : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\emptyset\}} \text{(4)}$$

(1):	Reflexivity
(2):	Empty subgraph
(3):	Transitivity
(4):	Union

A derivation tree proof for the supplement root rule is analogous, replacing in (2) the empty subgraph axiom with the root axiom.

The trim empty subgraph rule and trim root rule follows from the supplement empty subgraph rule and supplement root rule respectively. The derivation tree for the trim empty subgraph rule is as follows:

$$\frac{\frac{\overline{v : \mathcal{X} \rightarrow \mathcal{X}} \quad (1)}{v : \mathcal{X} \rightarrow \mathcal{X} \cup \{\emptyset\}} \quad (2) \quad v : \mathcal{X} \cup \{\emptyset\} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y}} \quad (3)$$

- (1): Reflexivity
 (2): Supplement empty subgraph
 (3): Transitivity

The derivation tree for the trim root rule is analogous, replacing in (2) the supplement empty subgraph rule with the supplement root rule. \square

Axiomatisation of pXFDs

It is easy to see that the inference rules from $\mathfrak{F}_{canonical}$, and thus all inference rules which can be derived from them, are also sound for the implication of pXFDs in general. However, to obtain a complete set of inference rules we must also consider the transformation between canonical pXFDs and those pXFDs which are not canonical.

Corollary 1.26 and Corollary 1.30, which provided justification for refinement ϑ , can be directly transformed into inference rules. This gives us our first axiomatisation for the implication of all pXFDs. The following theorem reconsiders the inference rules from $\mathfrak{F}_{canonical}$, but explicitly, we relax the assumption that the v -properties are essential.

Theorem 2.27 (pXFDs Axiomatisation, \mathfrak{F}_{prelim}). *For the inference rules from $\mathfrak{F}_{canonical}$ assume $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ are sets of v -properties, X, Y, U are v -subgraphs and n, m are v -ancestors. The inference rules from $\mathfrak{F}_{canonical}$ together with the following inference rules are sound and complete for the implication of pXFDs:*

$$\begin{array}{ll} \frac{v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}}{v : \vartheta(\mathcal{X}^{\check{S}}) \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}} & \frac{v : \vartheta(\mathcal{X}^{\check{S}}) \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}}{v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}} \\ \text{(subgraph LHS refinement)} & \text{(subgraph LHS reinstatement)} \\ \\ \frac{v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}}{v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \vartheta(\mathcal{Y}^{\check{S}}) \cup \mathcal{Y}^{\check{A}}} & \frac{v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \vartheta(\mathcal{Y}^{\check{S}}) \cup \mathcal{Y}^{\check{A}}}{v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}} \\ \text{(subgraph RHS refinement)} & \text{(subgraph RHS reinstatement)} \end{array}$$

$$\begin{array}{c}
\frac{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\mathcal{A}}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\mathcal{A}}}}{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \vartheta(\mathcal{X}^{\check{\mathcal{A}}}) \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\mathcal{A}}}} \\
\text{(ancestor LHS refinement)}
\end{array}
\qquad
\begin{array}{c}
\frac{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \vartheta(\mathcal{X}^{\check{\mathcal{A}}}) \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\mathcal{A}}}}{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\mathcal{A}}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\mathcal{A}}}} \\
\text{(ancestor LHS reinstatement)}
\end{array}$$

$$\begin{array}{c}
\frac{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\mathcal{A}}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\mathcal{A}}}}{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\mathcal{A}}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \vartheta(\mathcal{Y}^{\check{\mathcal{A}}})} \\
\text{(ancestor RHS refinement)}
\end{array}
\qquad
\begin{array}{c}
\frac{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\mathcal{A}}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \vartheta(\mathcal{Y}^{\check{\mathcal{A}}})}{v : \mathcal{X}^{\check{\mathcal{S}}} \cup \mathcal{X}^{\check{\mathcal{A}}} \rightarrow \mathcal{Y}^{\check{\mathcal{S}}} \cup \mathcal{Y}^{\check{\mathcal{A}}}} \\
\text{(ancestor RHS reinstatement)}
\end{array}$$

where $\mathcal{X}^{\check{\mathcal{S}}}, \mathcal{Y}^{\check{\mathcal{S}}}$ are sets of v -subgraphs, $\mathcal{X}^{\check{\mathcal{A}}}, \mathcal{Y}^{\check{\mathcal{A}}}$ are sets of v -ancestors and ϑ is refinement as per Definition 1.27 and Definition 1.23.

Proof. (Soundness) We omit the straightforward proofs that the inference rules from $\mathfrak{F}_{\text{canonical}}$ are also sound for the implication of pXFDs. The subgraph LHS/RHS refinement rules and the subgraph LHS/RHS reinstatement rules are sound from Corollary 1.26. Likewise, the ancestor LHS/RHS refinement rules and ancestor LHS/RHS reinstatement rules are sound from Corollary 1.30.

(Completeness) Assume there is some pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$ which cannot be derived from a set Σ of pXFDs using inference rules from $\mathfrak{F}_{\text{prelim}}$. More specifically, there must exist some canonical pXFD $v : \vartheta(\mathcal{X}) \rightarrow B$ with $B \in \vartheta(\mathcal{Y})$ which cannot be derived from Σ using $\mathfrak{F}_{\text{prelim}}$ because of the soundness of the union rule and the four reinstatement rules above. We need to show that there exists a T -compatible data tree T' such that all pXFDs in Σ are satisfied but $v : \vartheta(\mathcal{X}) \rightarrow B$ is violated.

Let $\mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+ = \bigcup \{Y \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma_{\mathfrak{F}_{\text{prelim}}}^+\}$ be the *closure of \mathcal{X}* under derivation using the system $\mathfrak{F}_{\text{prelim}}$. Observe that $\mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+$ is an equality set because of the subgraph axiom, ancestor axiom, empty subgraph axiom, root axiom, ancestor/subgraph RHS reinstatement rule, and transitivity rule.

Reusing the counter-example construction from Lemma 2.19 we obtain a T -compatible two- v -pre-image data tree T' containing two pre-image trees p_1, p_2 such that

$$p_1|_X \doteq p_2|_X \text{ if and only if } X \in \mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+ \text{ for every } v\text{-property } X.$$

The reflexivity axiom gives $v : \mathcal{X} \rightarrow \mathcal{X}$ which means $\mathcal{X} \subseteq \mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+$. Then by the two RHS refinement rules, we also infer that $v : \mathcal{X} \rightarrow \vartheta(\mathcal{X})$. Thus $p_1|_X \doteq p_2|_X$ for every $X \in \vartheta(\mathcal{X})$. Our assumption that $v : \vartheta(\mathcal{X}) \rightarrow B$ cannot be derived from Σ by $\mathfrak{F}_{\text{prelim}}$ means that $B \notin \mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+$. This then means, T' does not satisfy the pXFD $v : \vartheta(\mathcal{X}) \rightarrow B \notin \Sigma^+$.

Next we show that T' satisfies every pXFD in Σ . Let $v : \mathcal{W} \rightarrow \mathcal{Z}$ be an arbitrary pXFD belonging to Σ . If $\mathcal{W} \subseteq \mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+$ then $\mathcal{Z} \subseteq \mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+$ due to the transitivity rule. This yields T' satisfies $v : \mathcal{W} \rightarrow \mathcal{Z}$. Alternatively, if $\mathcal{W} \not\subseteq \mathcal{X}_{\mathfrak{F}_{\text{prelim}}}^+$ then $p_1|_W \neq p_2|_W$ for some $W \in \mathcal{W}$ and the pXFD $v : \mathcal{W} \rightarrow \mathcal{Z}$ is trivially satisfied in T' . Thus conclude our proof. \square

But actually, half of the new inference rules in the previous theorem are redundant with respect to the inference rules in $\mathfrak{F}_{canonical}$ and the other half can be replaced by more basic inference rules as we will show in the sequel. Thus, replacing the eight new inference rules in \mathfrak{F}_{prelim} by two other inference rules gives us an alternate, more elegant axiomatisation for pXFDs.

Firstly, we show how to derive four of the inference rules introduced in the previous theorem using $\mathfrak{F}_{canonical}$, these are: ancestor RHS refinement rule, ancestor LHS reinstatement rule, subgraph RHS refinement rule, subgraph LHS reinstatement rule. The next four lemmas present possible derivations of these inference rules.

Lemma 2.28. *The ancestor RHS refinement rule is derivable from the ancestor axiom and the Armstrong Axioms.*

Proof. A derivation for the ancestor RHS refinement rule is as follows:

	Inference rule applied:	
$v : \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}} \rightarrow \{\text{lca}(\mathcal{Y}^{\check{A}})\}$	<i>Reflexivity</i>	(1)
$v : \{\text{lca}(\mathcal{Y}^{\check{A}})\} \rightarrow \vartheta(\mathcal{Y}^{\check{A}})$	<i>Ancestor</i>	(2)
$v : \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}} \rightarrow \vartheta(\mathcal{Y}^{\check{A}})$	<i>Transitivity of (1) with (2)</i>	(3)
$v : \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}}$	<i>Reflexivity</i>	(4)
$v : \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}})$	<i>Union of (4) with (3)</i>	(5)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}})$	<i>Transitivity of (5) with $v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$</i>	(6)

From Definition 1.23 we observe that, for every set of v -ancestors $\mathcal{X}^{\check{A}}$, $n \in \vartheta(\mathcal{X}^{\check{A}})$ is an ancestor of $\text{lca}(\mathcal{X}^{\check{A}})$. Therefore, the pre-condition of the ancestor axiom holds. \square

Lemma 2.29. *The ancestor LHS reinstatement rule is derivable from the ancestor axiom and Armstrong Axioms.*

Proof. A derivation is as follows:

	Inference rule applied:	
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \{\text{lca}(\mathcal{X}^{\check{A}})\}$	<i>Reflexivity</i>	(1)
$v : \{\text{lca}(\mathcal{X}^{\check{A}})\} \rightarrow \vartheta(\mathcal{X}^{\check{A}})$	<i>Ancestor</i>	(2)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \vartheta(\mathcal{X}^{\check{A}})$	<i>Transitivity of (1) with (2)</i>	(3)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{X}^{\check{S}}$	<i>Reflexivity</i>	(4)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}})$	<i>Union of (3) with (4)</i>	(5)
$v : v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of (5) with</i>	
	$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	(6)

□

Lemma 2.30. *The subgraph RHS refinement rule is derivable from the subgraph axiom and the Armstrong inference rules.*

Proof. A derivation is as follows:

	Inference rule applied:
$\forall Y \in \mathcal{Y}^{\check{S}}. v : \mathcal{Y}^{\check{S}} \rightarrow \{Y\}$	<i>Reflexivity</i> (1)
$\forall Y' \in \vartheta(\mathcal{Y}^{\check{S}}). \exists Y \in \mathcal{Y}^{\check{S}}. v : \{Y\} \rightarrow \{Y'\}$	<i>Subgraph</i> (2)
$\forall Y' \in \vartheta(\mathcal{Y}^{\check{S}}). v : \mathcal{Y}^{\check{S}} \rightarrow \vartheta(\{Y'\})$	<i>Transitivity of (1) with (2)</i> (3)
$v : \mathcal{Y}^{\check{S}} \rightarrow \vartheta(\mathcal{Y}^{\check{S}})$	<i>Union of (3)</i> (4)
$v : \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}} \rightarrow \vartheta(\mathcal{Y}^{\check{S}}) \cup \mathcal{Y}^{\check{A}}$	<i>Augmentation of (4) with $\mathcal{Y}^{\check{A}}$</i> (5)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \vartheta(\mathcal{Y}^{\check{S}}) \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of $v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$ with (5)</i> (6)

From Definition 1.27 we observe that every v -subgraph in $\vartheta(\mathcal{Y}^{\check{S}})$ is contained in some v -subgraph in $\mathcal{Y}^{\check{S}}$. Therefore the pre-condition of the subgraph axiom hold in the derivation above. □

Lemma 2.31. *The subgraph LHS reinstatement rule is derivable from the subgraph axiom and the Armstrong inference rules.*

Proof. A derivation is as follows:

	Inference rule applied:
$\forall X \in \mathcal{X}^{\check{S}}. v : \mathcal{X}^{\check{S}} \rightarrow \{X\}$	<i>Reflexivity</i> (1)
$\forall X' \in \vartheta(\mathcal{X}^{\check{S}}). \exists X \in \mathcal{X}^{\check{S}}. v : \{X\} \rightarrow \{X'\}$	<i>Subgraph</i> (2)
$\forall X' \in \vartheta(\mathcal{X}^{\check{S}}). v : \mathcal{X}^{\check{S}} \rightarrow \{X'\}$	<i>Transitivity of (1) with (2)</i> (3)
$v : \mathcal{X}^{\check{S}} \rightarrow \vartheta(\mathcal{X}^{\check{S}})$	<i>Union of (3)</i> (4)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \vartheta(\mathcal{X}^{\check{S}}) \cup \mathcal{X}^{\check{A}}$	<i>Augmentation of (4) with $\mathcal{X}^{\check{A}}$</i> (5)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of (5) with</i> $v : \vartheta(\mathcal{X}^{\check{S}}) \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$ (6)

□

The next set of lemmas shows that the ancestor LHS refinement rule and the ancestor RHS reinstatement can be replace by the addition of the simple descendant axiom defined as follows:

Lemma 2.32 (simple descendant axiom). *Let n, m be v -ancestors. The following axiom is sound for the implication of pXFDs*

$$\frac{}{v : \{m\} \rightarrow \{n\}} \quad n \text{ is a simple descendant of } m$$

(simple descendant)

Proof. True by definition of T -compatibility - any pre-image of m in a T -compatible data tree T' has at most one descendant which is a pre-image of n . \square

Lemma 2.33. *The ancestor LHS refinement rule is derivable from the ancestor axiom, simple descendant axiom and the Armstrong Axioms.*

Proof. A derivation for the ancestor LHS refinement rule is as follows:

	Inference rule applied:	
$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \vartheta(\mathcal{X}^{\check{A}})$	<i>Reflexivity</i>	(1)
$v : \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \{\text{lca}(\mathcal{X}^{\check{A}})\}$	<i>Simple descendant</i>	(2)
$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \{\text{lca}(\mathcal{X}^{\check{A}})\}$	<i>Transitivity of (1) with (2)</i>	(3)
$\forall n' \in \mathcal{X}^{\check{A}}. v : \{\text{lca}(\mathcal{X}^{\check{A}})\} \rightarrow \{n'\}$	<i>Ancestor</i>	(4)
$\forall n' \in \mathcal{X}^{\check{A}}. v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \{n'\}$	<i>Transitivity of (3) with (4)</i>	(5)
$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \mathcal{X}^{\check{A}}$	<i>Union of (5)</i>	(6)
$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \mathcal{X}^{\check{S}}$	<i>Reflexivity</i>	(7)
$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}}$	<i>Union of (6) with (7)</i>	(8)
$v : \mathcal{X}^{\check{S}} \cup \vartheta(\mathcal{X}^{\check{A}}) \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of (8) with</i>	
	$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	(9)

From Definition 1.23 we can observe that, for every set of v -ancestors $\mathcal{X}^{\check{A}}$, the set $\vartheta(\mathcal{X}^{\check{A}})$ contains a single node n which is a simple ancestor of $\text{lca}(\mathcal{X}^{\check{A}})$. Moreover every v -ancestor $n \in \mathcal{X}^{\check{A}}$ is an ancestor of $\text{lca}(\mathcal{X}^{\check{A}})$. Therefore, applications of the simple descendant axiom and the ancestor axiom in the derivation above are clearly valid. \square

Lemma 2.34. *The ancestor RHS reinstatement rule is derivable from the ancestor axiom, simple descendant axiom and Armstrong Axioms.*

Proof. A derivation is as follows:

	Inference rule applied:	
$v : \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \vartheta(\mathcal{Y}^{\check{A}})$	<i>Reflexivity</i>	(1)
$v : \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \{\text{lca}(\mathcal{Y}^{\check{A}})\}$	<i>Simple descendant</i>	(2)
$v : \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \{\text{lca}(\mathcal{Y}^{\check{A}})\}$	<i>Transitivity of (1) with (2)</i>	(3)
$\forall n' \in \mathcal{Y}^{\check{A}}. v : \{\text{lca}(\mathcal{Y}^{\check{A}})\} \rightarrow \{n'\}$	<i>Ancestor</i>	(4)
$\forall n' \in \mathcal{Y}^{\check{A}}. v : \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \{n'\}$	<i>Transitivity of (3) with (4)</i>	(5)
$v : \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \mathcal{Y}^{\check{A}}$	<i>Union of (5)</i>	(6)
$v : \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \mathcal{Y}^{\check{S}}$	<i>Reflexivity</i>	(7)
$v : \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}}) \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Union of (6) with (7)</i>	(8)
$v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of $v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \vartheta(\mathcal{Y}^{\check{A}})$ with (8)</i>	(9)

□

Similarly, the subgraph LHS refinement rule and the subgraph RHS reinstatement can be replaced by the addition of the following inference rule:

Lemma 2.35 (join axiom). *Let X, Y be v -subgraphs. The following axiom is sound for the implication of $pXFDs$*

$$\frac{}{v : \{X, Y\} \rightarrow \{X \sqcup Y\}} \quad X, Y \text{ are } v\text{-reconcilable} \\ \text{(join)}$$

Proof. Follows from Lemma 1.14. □

Lemma 2.36. *The subgraph LHS refinement rule is derivable from the subgraph axiom, join axiom and the Armstrong inference rules.*

Proof. A derivation is as follows:

	Inference rule applied:
$\forall X \in \vartheta(\mathcal{X}^{\check{S}}). v : \vartheta(\mathcal{X}^{\check{S}}) \rightarrow \{X\}$	<i>Reflexivity</i> (1)
$\forall X \in \mathcal{X}^{\check{S}}. \forall X' \in \vartheta(\{X\}). v : \{X\} \rightarrow \{X'\}$	<i>Subgraph</i> (2)
$\forall X \in \mathcal{X}^{\check{S}}. \forall X' \in \vartheta(\{X\}). v : \vartheta(\mathcal{X}^{\check{S}}) \rightarrow \{X'\}$	<i>Transitivity of (1) with (2)</i> (3)
$\forall X \in \mathcal{X}^{\check{S}}. v : \vartheta(\mathcal{X}^{\check{S}}) \rightarrow \vartheta(\{X\})$	<i>Union of (3)</i> (4)
$\forall X \in \mathcal{X}^{\check{S}}. v : \vartheta(\{X\}) \rightarrow \{X\}$	<i>Join</i> (5)
$\forall X \in \mathcal{X}^{\check{S}}. v : \vartheta(\mathcal{X}^{\check{S}}) \rightarrow \{X\}$	<i>Transitivity of (4) with (5)</i> (6)
$v : \vartheta(\mathcal{X}^{\check{S}}) \rightarrow \mathcal{X}^{\check{S}}$	<i>Union of (6)</i> (7)
$v : \vartheta(\mathcal{X}^{\check{S}}) \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}}$	<i>Augmentation of (7) with $\mathcal{X}^{\check{A}}$</i> (8)
$v : \vartheta(\mathcal{X}^{\check{S}}) \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of (8) with</i> $v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$ (9)

From Definition 1.27 we can observe that every v -subgraph belonging to $\vartheta(\{X\})$ is contained X for every $X \in \mathcal{X}^{\check{S}}$. Therefore the pre-condition of the subgraph axiom hold in the derivation above. As remarked, members of $\vartheta(\{X\})$ are pairwise v -reconcilable making applications of the join axiom in (5) valid. \square

Lemma 2.37. *The subgraph RHS reinstatement rule is derivable from the subgraph axiom, join axiom and the Armstrong inference rules.*

Proof. A derivation is as follows:

	Inference rule applied:
$\forall Y \in \vartheta(\mathcal{Y}^{\check{S}}). v : \vartheta(\mathcal{Y}^{\check{S}}) \rightarrow \{Y\}$	<i>Reflexivity</i> (1)
$\forall Y \in \mathcal{Y}^{\check{S}}. \forall Y' \in \vartheta(\{Y\}). v : \{Y\} \rightarrow \{Y'\}$	<i>Subgraph</i> (2)
$\forall Y \in \mathcal{Y}^{\check{S}}. \forall Y' \in \vartheta(\{Y\}). v : \vartheta(\mathcal{Y}^{\check{S}}) \rightarrow \{Y'\}$	<i>Transitivity of (1) with (2)</i> (3)
$\forall Y \in \mathcal{Y}^{\check{S}}. v : \vartheta(\mathcal{Y}^{\check{S}}) \rightarrow \vartheta(\{Y\})$	<i>Union of (3)</i> (4)
$\forall Y \in \mathcal{Y}^{\check{S}}. v : \vartheta(\{Y\}) \rightarrow \{Y\}$	<i>Join</i> (5)
$\forall Y \in \mathcal{Y}^{\check{S}}. v : \vartheta(\mathcal{Y}^{\check{S}}) \rightarrow \{Y\}$	<i>Transitivity of (4) with (5)</i> (6)
$v : \vartheta(\mathcal{Y}^{\check{S}}) \rightarrow \mathcal{Y}^{\check{S}}$	<i>Union of (6)</i> (7)
$v : \vartheta(\mathcal{Y}^{\check{S}}) \cup \mathcal{Y}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Augmentation of (7) with $\mathcal{Y}^{\check{A}}$</i> (8)
$\mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \mathcal{Y}^{\check{S}} \cup \mathcal{Y}^{\check{A}}$	<i>Transitivity of $v : \mathcal{X}^{\check{S}} \cup \mathcal{X}^{\check{A}} \rightarrow \vartheta(\mathcal{Y}^{\check{S}}) \cup \mathcal{Y}^{\check{A}}$</i> <i>with (6)</i> (9)

$\frac{}{v : \mathcal{X} \rightarrow \mathcal{Y}} \mathcal{Y} \subseteq \mathcal{X}$ <p>(reflexivity)</p>	$\frac{v : \mathcal{X} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{X} \cup \mathcal{Y}}$ <p>(extension)</p>	$\frac{v : \mathcal{X} \rightarrow \mathcal{Y}, v : \mathcal{Y} \rightarrow \mathcal{Z}}{v : \mathcal{X} \rightarrow \mathcal{Z}}$ <p>(transitivity)</p>
$\frac{}{v : \{X, Y\} \rightarrow \{X \sqcup Y\}} X, Y \text{ are } v\text{-reconcilable}$ <p>(join)</p>		
$\frac{}{v : \{m\} \rightarrow \{n\}} n \text{ is a simple descendant of } m$ <p>(simple descendant)</p>		
$\frac{}{v : \{X\} \rightarrow \{Y\}} Y \text{ is contained in } X$ <p>(subgraph)</p>	$\frac{}{v : \{\} \rightarrow \{r_T\}}$ <p>(root)</p>	
$\frac{}{v : \{n\} \rightarrow \{m\}} m \text{ is an ancestor of } n$ <p>(ancestor)</p>	$\frac{}{v : \{\} \rightarrow \{\emptyset\}}$ <p>(empty subgraph)</p>	
$\frac{}{v : \vartheta(\{v\}) \rightarrow \{U\}} U \text{ is a } v\text{-unit}$ <p>(target)</p>		

Table 2.2: System $\mathfrak{F}_{general}$ of inference rules for pXFDs

□

The second important result for the thesis is the following axiomatisation for the class of pXFDs in general. Note, however, that we are only talking about reasoning among the class of all pXFDs having the *same target node*. Reasoning about pXFDs which may have different nodes for targets is also interesting but is left as possible future work.

Theorem 2.38 (pXFDs Axiomatisation, $\mathfrak{F}_{general}$). *Let $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$ be sets of v -subgraphs, X, Y, U be v -subgraphs, and n, m be v -ancestors. The system $\mathfrak{F}_{general}$ of inference rules from Table 2.2 is sound and complete for the implication of pXFDs.*

Proof. Follows from Theorem 2.27, Lemma 2.36, Lemma 2.30, Lemma 2.31, Lemma 2.37, Lemma 2.33, Lemma 2.28, Lemma 2.29 and Lemma 2.34. □

In the next remark we formally prove our observation that property-equality on the target v determines property-equality on every v -property. We go one step further to note that if the target node is simple, then every pXFD is satisfied by every T -compatible

data tree. These observations will be revisited later when we consider the notion of trivial pXFDs (to be defined in Definition 3.8).

Remark 2.39. Let T' be some arbitrary T -compatible data tree. Then

$$T' \text{ satisfies } v : \{v\} \rightarrow \{Y\} \text{ for every } v\text{-property } Y \in \check{\mathfrak{S}}_T(v) \cup \check{\mathfrak{A}}_T(v).$$

We know T' satisfies $v : \vartheta(\{v\}) \rightarrow \{Y\}$ for every v -subgraph $Y \in \check{\mathfrak{S}}_T(v)$ from the soundness of the target axiom, join axiom, subgraph axiom, and transitivity rule. The ancestor axiom, simple descendant axiom and transitivity rule yield that T' satisfies the pXFD $v : \vartheta(\{v\}) \rightarrow \{n\}$ for every v -ancestor $n \in \check{\mathfrak{A}}_T(v)$. Altogether T' satisfies $v : \vartheta(\{v\}) \rightarrow \{Y\}$ for every $Y \in \check{\mathfrak{S}}_T(v) \cup \check{\mathfrak{A}}_T(v)$. Finally, we get the statement above since the ancestor axiom gives T' satisfies $v : \{v\} \rightarrow \vartheta(\{v\})$.

When v is simple then the simple descendant axiom states that T' satisfies $v : \{r_T\} \rightarrow \{v\}$ for any T -compatible data tree T' . The previous observation and the root axiom and transitivity rule further yield that

$$T' \text{ satisfies } v : \{\} \rightarrow \{Y\} \text{ for every } v\text{-property } Y \in \check{\mathfrak{S}}_T(v) \cup \check{\mathfrak{A}}_T(v).$$

In other words, if v is simple then all pXFDs can be determined from the empty set of pXFDs, thus every pXFD is satisfied by every T -compatible data tree. \square

Our axiomatisations reveal that implication of pXFDs (canonical or in general) is syntactically captured by Armstrong's well-known inference system for relational functional dependencies plus additional axioms that express the structural properties of XML schema trees. Unlike the Semantic Equivalence Theorem for pXFDs which allows us to check one specific pXFD implication, the axiomatisations allow us to enumerate all derivable (canonical) pXFDs. We can adopt the same idea as in the relational data model - we compute property closures which are proposed as analogies to the notion of attribute closures. For a set of v -properties \mathcal{X} , let

$$\begin{aligned} \mathcal{X}^* &= \{Y \in \check{\mathfrak{S}}_T(v) \cup \check{\mathfrak{A}}_T(v) \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma^*\} \\ &= \{Y \in \check{\mathfrak{S}}_T(v) \cup \check{\mathfrak{A}}_T(v) \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma_{\check{\mathfrak{S}}_{general}}^+\} \end{aligned}$$

be the set of all v -properties which are functionally determined by \mathcal{X} . We call \mathcal{X}^* the *property closure* of \mathcal{X} with respect to Σ . To enumerate all implied pXFDs, we need to find the property closure for every subset \mathcal{X} of v -properties, i.e., for all $\mathcal{X} \subseteq \check{\mathfrak{S}}_T(v) \cup \check{\mathfrak{A}}_T(v)$. To check whether an implication $\Sigma \models v : \mathcal{X} \rightarrow \mathcal{Y}$ holds, we can check whether $\mathcal{Y} \subseteq \mathcal{X}^*$. Indeed, it is sufficient to consider for every set \mathcal{X} its *essential property closure* as follows

$$\begin{aligned} \mathcal{X}^{*\mathbf{E}} &= \{Y \in \mathbf{E}_{\check{\mathfrak{S}}_T}^{\check{\mathfrak{S}}_T}(v) \cup \mathbf{E}_{\check{\mathfrak{A}}_T}^{\check{\mathfrak{A}}_T}(v) \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma^*\} \\ &= \{Y \in \mathbf{E}_{\check{\mathfrak{S}}_T}^{\check{\mathfrak{S}}_T}(v) \cup \mathbf{E}_{\check{\mathfrak{A}}_T}^{\check{\mathfrak{A}}_T}(v) \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma_{\check{\mathfrak{S}}_{canonical}}^+\} \end{aligned}$$

We can determine whether $\Sigma \models v : \mathcal{X} \rightarrow \mathcal{Y}$ holds by checking whether $\vartheta(\mathcal{Y}) \subseteq (\vartheta(\mathcal{X}))^{*\mathbf{E}}$.

Algorithmically, there are two obvious approaches to adapt the attribute closure algorithm from the relational data model for computing property closures with respect to Σ .

Algorithm 2.2 property_closure

Input: Σ, \mathcal{X} /* where Σ is a set of singular pXFDs over T and $\mathcal{X} \subseteq \check{\mathbb{S}}_T(v) \cup \check{\mathbb{A}}_T(v)$ */**Output:** \mathcal{X}^* /* where $X^* = \{Y \in \check{\mathbb{S}}_T(v) \cup \check{\mathbb{A}}_T(v) \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma^*\}$ */

1. $\Sigma' := \text{Extend } \Sigma$ by adding pXFDs corresponding to the join, simple descendant, subgraph, ancestor, and target axioms (i.e., analogous to base translation).
 2. $\mathcal{X}^* := \mathcal{X} \cup \{\emptyset, r_T\}$
 3. **while** there exists $Y \in \check{\mathbb{S}}_T(v) \cup \check{\mathbb{A}}_T(v)$ such that $Y \notin \mathcal{X}^*$
and there exists $v : \mathcal{W} \rightarrow \{Y\} \in \Sigma'$ with $\mathcal{W} \subseteq \mathcal{X}^*$ **do**
 4. $\mathcal{X}^* := \mathcal{X}^* \cup \{Y\}$
 5. **end while**
 6. **return** \mathcal{X}^*
-

They are shown as Algorithm 2.2 and Algorithm 2.3. In both cases, we must consider the axioms which encode structural properties. We can initialise the result set to $\mathcal{X} \cup \{\emptyset, r_T\}$ in order to reflect the empty subgraph axiom and the root axiom. The remaining structural properties (i.e., relating to the join, simple descendant, subgraph, ancestor, and target axioms) can be considered in two ways. For one, we can express the structural properties as pXFDs for augmenting the given set Σ (cf., base translation in the Horn encoding). This approach has been used in Algorithm 2.2.

Algorithm 2.3 essential_property_closure

Input: Σ, \mathcal{X} /* where Σ is a set of singular pXFDs over T and $\mathcal{X} \subseteq \check{\mathbb{S}}_T(v) \cup \check{\mathbb{A}}_T(v)$ */**Output:** \mathcal{X}^{*E} /* where $X^{*E} = \{Y \in \mathbf{E}_{\check{\mathbb{S}}_T(v)} \cup \mathbf{E}_{\check{\mathbb{A}}_T(v)} \mid v : \mathcal{X} \rightarrow \{Y\} \in \Sigma^*\}$ */

1. $\mathcal{X}^{*E} := \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$ // empty subgraph and root axioms
 2. **while** there exists $Y \in \check{\mathbb{S}}_T(v) \cup \check{\mathbb{A}}_T(v)$ such that $\vartheta(\{Y\}) \not\sqsubseteq \mathcal{X}^{*E}$
and there exists $v : \mathcal{W} \rightarrow \{Y\} \in \Sigma$ with $\vartheta(\mathcal{W}) \sqsubseteq \mathcal{X}^{*E}$ **do**
 3. $\mathcal{X}^{*E} := \mathcal{X}^{*E} \cup \vartheta(\{Y\})$
 4. **end while**
 5. **if** $\vartheta(\{v\}) \in \mathcal{X}^{*E}$ **then** // target axiom
 6. **return** $\mathbf{E}_{\check{\mathbb{S}}_T(v)} \cup \mathbf{E}_{\check{\mathbb{A}}_T(v)}$
 7. **else** // subgraph and ancestor axioms
 8. $\mathcal{X}^{*E} := \mathcal{X}^{*E} \cup \{W \in \mathbf{E}_{\check{\mathbb{S}}_T(v)} \cup \mathbf{E}_{\check{\mathbb{A}}_T(v)} \mid W \sqsubseteq X \text{ where } X \in \mathcal{X}^{*E}\}$
 9. **end if**
 10. **return** \mathcal{X}^{*E}
-

Alternatively, we can use a second approach which exploits the main idea from the attribute closure algorithm - considering pXFDs derivable by the Armstrong inference rules. When doing so, we

- replace \subseteq in the attribute closure algorithm by our more general partial-order p-subsumption (as defined in Definition 1.36), and
- examine the LHS/RHS of pXFDs in terms of their refinement, e.g., $\vartheta(\{Y\})$ rather than $\{Y\}$.

Afterwards, we augment the result set by considering the structural properties. The target axiom yields the special case that the (essential) property closure is the family of all (essential) v -properties if $\vartheta(\{v\}) \in \mathcal{X}^*$ (or \mathcal{X}^{*E} , respectively). In contrast, the other structural properties help to ensure that the property closure set is an equality set. The subgraph and ancestor axioms relate to completeness, that is, by enforcing the addition of all (essential) v -properties which are p-subsumed by other (essential) v -properties which already belong to the intermediary (essential) property closure set.

When we use the second approach to generate essential property closures, the join and simple descendant axioms can be disregarded, as done in Algorithm 2.3. When we use the second approach to compute full property closures, we take the output of Algorithm 2.3 and further augment it as follows:

```

 $\mathcal{X}^* := \mathcal{X}^{*E}$ 
while there exists  $Y \in \check{\mathcal{S}}_T(v) \cup \check{\mathcal{A}}_T(v)$  such that  $Y \notin \mathcal{X}^*$  and  $\vartheta(\{Y\}) \subseteq \mathcal{X}^*$ 
do
     $\mathcal{X}^* := \mathcal{X}^* \cup \{Y\}$ 
end while

```

Example 2.40. Consider again the pXFD implication problem from Example 2.22 in which we check whether pXFD3 follows from $\Sigma = \{\text{pXFD1, pXFD2 and pXFD6}'\}$, where

$$\begin{array}{ll}
 v_{\text{partners}} : \{v_{\text{class}}, \text{boy}\} \rightarrow \{\text{girl}\} & (\text{pXFD1}) \\
 v_{\text{partners}} : \{v_{\text{class}}, \text{girl}\} \rightarrow \{\text{boy}\} & (\text{pXFD2}) \\
 v_{\text{partners}} : \{\text{boy}, \text{girl}\} \rightarrow \{v_{\text{class}}\} & (\text{pXFD6}') \\
 v_{\text{partners}} : \{\text{boy}\} \rightarrow \{\text{girl}\} & (\text{pXFD3})
 \end{array}$$

To answer, we compute the property closure $\{\text{boy}\}^*$ with respect to Σ' , where Σ' is the extension of Σ by the following pXFDs:

$$\begin{array}{ll}
 v_{\text{partners}} : \{v_{\text{partners}}\} \rightarrow \{v_{\text{attendees}}\} & v_{\text{partners}} : \{\{\text{boy}, \text{girl}\}\} \rightarrow \{\text{girl}\} \\
 v_{\text{partners}} : \{v_{\text{attendees}}\} \rightarrow \{v_{\text{session}}\} & v_{\text{partners}} : \{\{\text{boy}, \text{girl}\}\} \rightarrow \{\text{boy}\} \\
 v_{\text{partners}} : \{v_{\text{session}}\} \rightarrow \{v_{\text{class}}\} & v_{\text{partners}} : \{\text{boy}, \text{girl}\} \rightarrow \{\{\text{girl}, \text{boy}\}\} \\
 v_{\text{partners}} : \{v_{\text{session}}\} \rightarrow \{v_{\text{attendees}}\} & v_{\text{partners}} : \{\} \rightarrow \{\emptyset\} \\
 v_{\text{partners}} : \{\} \rightarrow \{v_{\text{danceschool}}\} & \\
 & v_{\text{partners}} : \{v_{\text{partners}}\} \rightarrow \{\text{boy}\} \\
 & v_{\text{partners}} : \{v_{\text{partners}}\} \rightarrow \{\text{girl}\}
 \end{array}$$

Since

$$\{\text{girl}\} \not\subseteq \{\text{boy}\}^* = \{\text{boy}, v_{\text{danceschool}}, \emptyset, \}$$

we conclude that pXFD3 is not implied by Σ . □

2.2.3 Assistance with Constraints Acquisition

Constraint acquisition is the process of determining a (hopefully complete) constraint specification that captures meaningful semantics about the underlying application domain to be modelled. In addition to restricting data instances to those which are meaningful, there are various other applications which can be leveraged through identification of a correct and complete set of constraints, such as normalisation and query optimisation. In practice, several participants take part in constraint acquisition. Candidate constraints are suggested and inspected by the committee in views of knowledge the participants have about the underlying application domain. However, some people may interpret the same constraint quite differently, and coming to a mutual resolve can be challenging.

The use of sample databases in the database design process is not new. Sample instances has strong connection to the application domain and can often be better suited in the communication of constraints with domain experts than formal constraint specification. *Armstrong databases* are sample instances which *satisfy precisely* a given set of specified constraints and its implications. As such they offer an alternative user-friendly representation for constraint specifications and is considered to be an important tool for design-by-example [68, 83]. The notion of Armstrong instances is accredited to the seminal work by Armstrong [5], who showed that the class of relational FDs enjoys Armstrong relation, that is, for every set of FDs Σ^r there is a relation which satisfies precisely the FDs in Σ^{r*} . We recommend the paper [25] by Fagin, for an early survey of Armstrong databases in the context of the relational data model.

However, Armstrong databases may not provide adequate indication of whether a particular constraint should be specified. On the one hand, Armstrong databases must simultaneously violate all constraints that are not implied by a given set and therefore can be quite large [9], and on the other, an Armstrong databases only exhibit a single violation of a candidate constraint. In addition, it may even be that Armstrong databases do not exist for a certain class of integrity constraints (e.g., [51, 87]). To date, it is not known whether the class of pXFDs enjoys Armstrong databases.

Our last application of the Semantic Equivalence Theorem for pXFDs is to facilitate a sample-based approach for supporting pXFDs acquisition. The sample-based approach that we present here was originally developed for providing decision support to the acquisition of FDs and MVDs [45]. Unlike Armstrong instances, the sample-based method identifies small two-member counter-examples which allow users to more readily focus on the task at hand - deciding whether or not a single candidate constraint should be explicitly specified.

The basic idea is to partition the set of all syntactically valid pXFDs into two parts:

- Σ containing all pXFDs which must be specified explicitly, and
- Ψ containing all pXFDs which need not be specified.

To do so, we iteratively inspect through the set of all possible pXFDs. A candidate pXFD need not be specified explicitly if it is implied by Σ or does not represent any meaningful semantics with respect to the application domain.

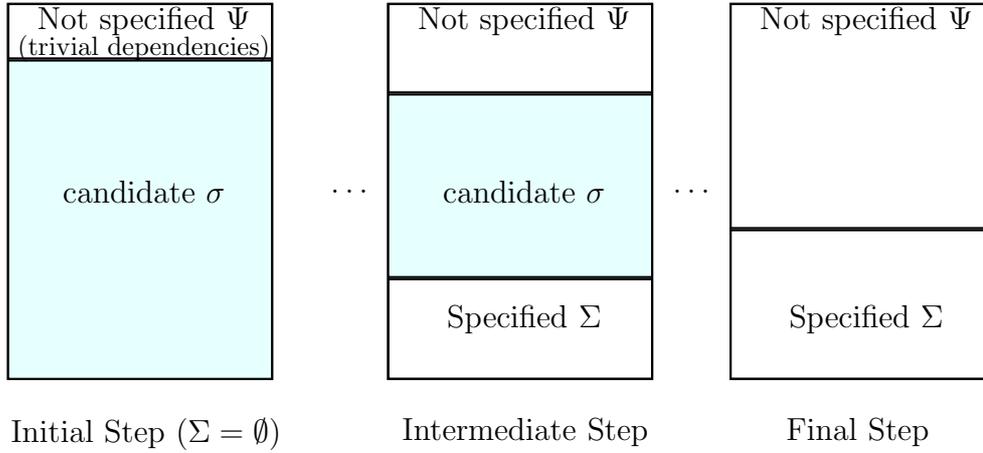


Figure 2.7: Constraint acquisition by iterative inspection of candidates.

In the beginning there may be some or no or limited knowledge about Σ and Ψ . A suggestion to specify some candidate pXFD σ is made. We verify that σ is not implied by the current pXFD specification Σ . If this is true then we proceed to check whether it makes sense to violate σ . If we can answer in the affirmative then σ should not be specified. To be able to reason thus, we must identify sufficient counter-examples showing the simultaneous violation of σ and satisfaction of Σ . Propositional tableaux and the construction from Section 2.1.3 provide us with a semi-automatic generation of the sample two-*v*-pre-image data trees which are necessary for justifying the decision about whether to specify σ .

Propositional Tableaux

We will briefly outline the concept of propositional tableaux before continuing to detail a sample-based approach for pXFDs acquisition. A *propositional tableau* is an unordered, finite, rooted tree in which nodes are labelled with sets of propositional formulas defined inductively as follows:

- A single node containing a set P of propositional formulas is a tableau for P .
- If T is a tableau for P and T' results from T by applying an expansion rule from Table 2.3 and Table 2.4, then T' is also a tableau for P .

We only briefly summarise how to construct a propositional tableau for a given set of propositional formulas. [84] is recommended for a comprehensive look into first-order logic from a propositional tableaux point of view.

An expansion rule consists of a premise and a conclusion. Both consist of a set of propositional formulas and can be categorised into either α -rules or β -rules. The classification reflects whether the formulas in the conclusion can be appended together in a single node or split into multiple nodes. Note that under any truth assignment, the

<i>premise</i>	<i>conclusion</i>	
	α_1	α_2
α		
$\varphi \vee \psi$	φ	ψ
$\varphi \wedge \psi$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \Rightarrow \psi)$	φ	$\neg\psi$
$\neg\neg(\varphi)$	φ	

Table 2.3: α -rule

<i>premise</i>	<i>conclusion</i>	
	β_1	β_2
β		
$\varphi \vee \psi$	φ	ψ
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \Rightarrow \psi$	$\neg(\varphi)$	ψ

Table 2.4: β -rule

premise of an α -rule is *true* precisely if both formulas in the conclusion are *true*. On the other hand, the premise of a β -rule is *true* precisely if at least one of the formulas in the conclusion is *true*. A rule can be applied to expand a leaf whenever the premise of expansion rule matches a formula that appears somewhere on the path from the root to that leaf. The rules are summarised in Table 2.3 and 2.4.

A *branch* of a tableau, i.e., a path from the root to a leaf, is said to be *closed* if it contains a formula and its negation. A tableau is said to be *closed* if all its branches are closed. A branch \mathfrak{b} of a tableau is *complete* if for every formula α which occurs in \mathfrak{b} also α_1, α_2 occur in \mathfrak{b} and, for every formula β which occurs in \mathfrak{b} at least one of β_1, β_2 occurs in \mathfrak{b} . A tableau is *completed* if every branch is either close or complete.

It is well known that every complete open branch of a tableau for P gives a boolean truth assignment that makes all formulas in P *true*. That is, a completed tableau for P allow us to read off all models for which P is *true*.

Decision Support for pXFDs Acquisition

How do propositional tableaux help in deciding whether or not to specify σ ?

Since we need not specify those pXFDs which can be implied, we can disregard all non-singular, non-canonical pXFD as candidates (recall Section 1.3.4 and the union rule).

Let σ then be a singular canonical pXFD with $\{h_\sigma\} = H_\sigma$. By constructing a propositional tableau for the formula $\bigwedge H_\Sigma \wedge \bigwedge H_T \wedge \neg h_\sigma$ we can firstly check whether Σ implies σ , in which case the tableau is closed. The alternative is that the tableau is complete and open. Such a tableau gives us an insight into *all* boolean assignments such that $\bigwedge H_\Sigma \wedge \bigwedge H_T \wedge \neg h_\sigma$ evaluates to *true*. Instead of propositional tableaux, it is of course possible to use any SAT-solver which is capable of enumerating all models. Some examples of other approaches for enumerating models are presented in [34, 49].

From the Semantic Equivalence Theorem for pXFDs, we can construct a counter-example data tree for each model. The counter-examples are to be validated by some panel of domain experts. By negotiation or some other conflict resolution mechanism, the panel is required to conclude whether each counter-example data tree presented to them is “acceptable” or “unacceptable”. If at least one of the counter-example data trees is acceptable, that is, the data instance is meaningful to store in practice, then σ can be reasonably violated and therefore should be included in Ψ , i.e., not specified. In the case that all generated counter-examples are rejected, we can conclude that σ is not implied

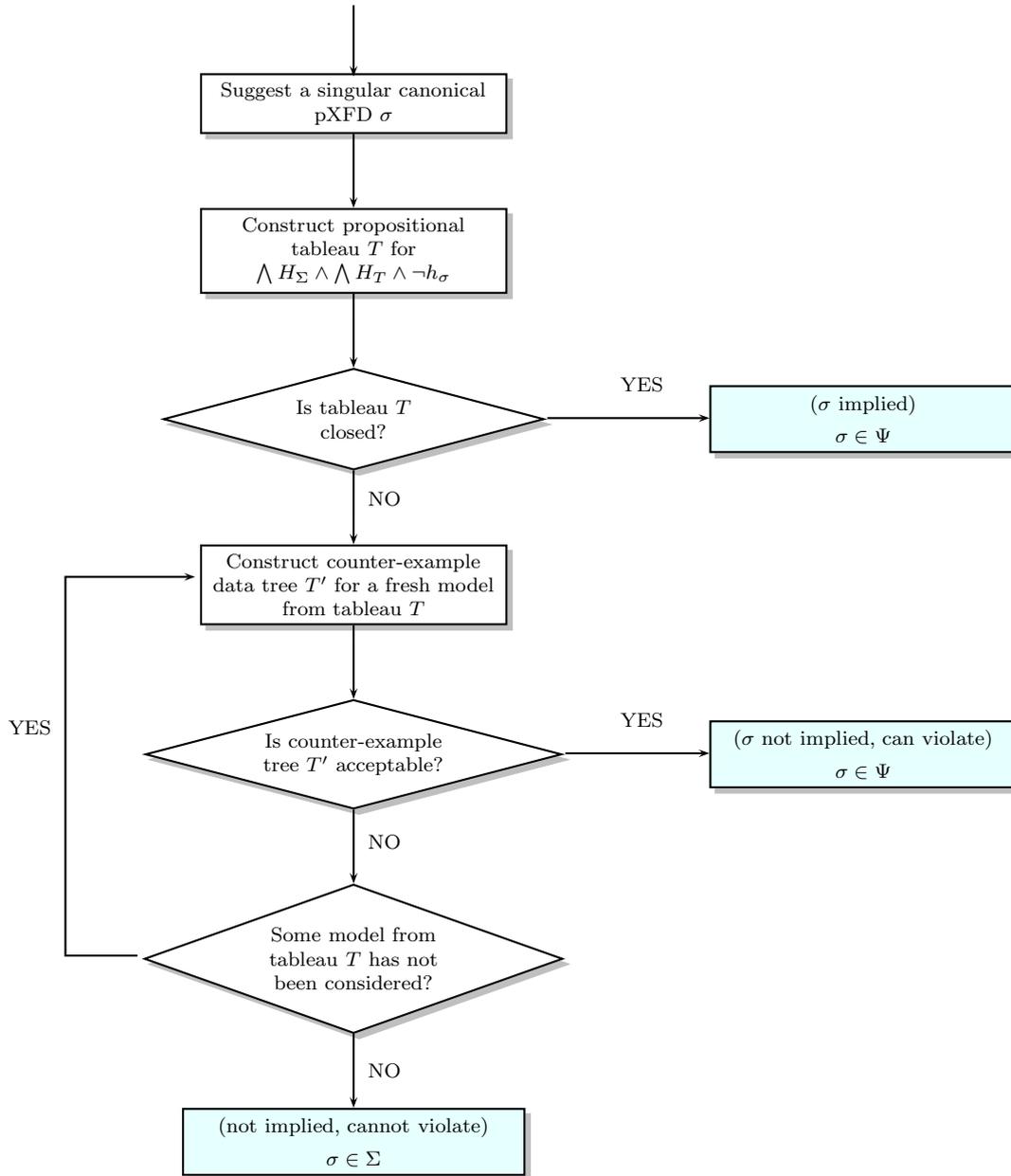


Figure 2.8: Flowchart of the propositional tableaux method for pXFDs acquisition.

by Σ but cannot be reasonably violated. Obviously σ must be specified explicitly and we add it to Σ . The flowchart in Figure 2.8 summarises the sample-based approach to deciding whether to specify a given candidate pXFD.

The effectiveness of this sample-based approach hinges largely on our ability to generate counter-examples which are user-friendly and contain helpful sample data. The construction in Section 2.1.3 ensures that the counter-examples are compact with respect to the number of pre-image trees, although other quality criteria may prove to be equally important.

The performance of the sample-based approach may also improve with more stringent selection of each candidate pXFD. We can try to make use of the fact that both Σ and Ψ can be used to imply pXFDs which are known to be satisfied/violated in any T -compatible data tree. A complete yet efficient exploration through all syntactically valid pXFDs is helpful in this endeavour and can also be useful for developing a level-wised approach for pXFD discovery (a counter-part of the transversal approach discussed in the next chapter). Clearly, we need to consider the trade-off between time saved from checking fewer candidate pXFDs versus time spent to select the pXFD candidates. Further investigation of pXFDs implication from the next chapter (see Section 3.1) may help in this direction, but this is, for the main part, left as future work.

2.3 Summary

In this chapter, we have established the Semantic Equivalence Theorem for pXFDs which relates canonical pXFDs and propositional Horn clauses. Specifically we proposed a translation of canonical pXFDs to sets of propositional Horn clauses such that the implication of canonical pXFDs is equivalent to the logical consequence of corresponding propositional logic statements under the provided translation. A semantic proof, borrowing ideas from the work of Fagin, was provided. One important part of the proof was the construction of two- v -pre-image data trees guided by a set of v -properties on which the two constructed pre-image trees must agree. The construction was discussed at length and formally shown to be appropriate.

The rest of the chapter discussed three interesting applications of the Semantic Equivalence Theorem for pXFDs.

The first, and most obvious, application was in deciding the implication problem for pXFDs. The problem in question is whether $\Sigma \models \sigma$ holds, where $\Sigma \cup \{\sigma\}$ is a set of pXFDs over some XML schema tree T . The pXFD implication problem can be easily translated into that of Horn-satisfiability which is known to be decidable in linear time.

The second application was in finding and proving a sound and complete system of inference rules for pXFDs. From the semantic equivalence, we first established a sound and complete system $\mathfrak{F}_{canonical}$ for the implication of canonical pXFDs. This system was then easily generalised into two systems of inference rules which are sound and complete for the implication of pXFDs in general. The most significant result of this subsection is the system $\mathfrak{F}_{general}$ of straightforward inference rules for the implication of pXFDs.

The third and final application was a sample-based method for supporting pXFDs acquisition. The constraint acquisition process is one in which we try to find a set of pXFDs which should be specify over a given schema tree. Through the Semantic Equivalence Theorem for pXFDs we can verify whether a candidate pXFD σ is implied by the current pXFD specification. If so then there is no need to specify σ . With the help of propositional tableaux, we can also explore whether there is an acceptable way to violate σ while satisfying the current pXFD specification. This allows us to conclude whether or not to specify σ when $\Sigma \not\models \sigma$.

Chapter 3

Dependency Discovery

Dependency discovery, otherwise known as *dependency inference*, is the task of finding all constraints that hold in a given data instance. This is useful as a supplementary process to constraint acquisition mentioned in the previous chapter - we can find candidate constraints to specify by discovering them from good sample data instances. Furthermore, dependency discovery can be used to re-verify and update obsolete constraint specifications, generate keys in the relational data model [32], as well as to discover constraint specifications for areas such as data cleaning and data integration.

There has been many investigation into dependency discovery in the context of the relational data model. Several researchers have tackled how to discover inclusion dependencies (e.g., [8, 52, 17]) and minimal keys (e.g., [18, 32]). But this is eclipsed by the countless works which ponder the problem of FD discovery, such as, [10, 30, 35, 48, 59, 58, 61, 62, 72, 76, 81, 100, 101, 103]. The majority of the literature follows one of two mainstream approaches for FD discovery: transversal approach or level-wise approach. There are, however, a few works into other directions, for example, by considering FD in the context of formal concept analysis (e.g., [7, 62]), incrementally discovering FDs (e.g., [99]) and in terms of approximate FDs and association rules (e.g., [26, 54, 79]).

One of the earliest scrutiny of FD discovery appeared in [69]. In the paper, Mannila and R  ih   present a *hypergraph-based method* for discovering FDs in relational databases (see also [68, 71, 72]). The FD discovery problem is reduced to the well-known problem of finding minimal hypergraph transversals [22, 23]. To avoid confusion with the notion of “transversal” that will be introduced later in the chapter, from now on we will refer to “hypergraph transversals” by its alternate name of “*hitting sets*”.

The hypergraph-based method re-characterises the notion of FD satisfaction:

An FD $X \rightarrow Y$ is satisfied in a relation r if and only if any two tuples t_1, t_2 in r that *do not agree* on their projections Y (i.e., $t_1[Y] \neq t_2[Y]$) also *do not agree* on their projections to X (i.e., $t_1[X] \neq t_2[X]$).

Disagree sets of r , sometimes also known as *difference sets* of r , are proposed as auxiliary source of information about whether a pair of tuples do not agree on some set of attributes:

A disagree set of r is a minimal set of attributes on which some pair of distinct tuples in r do not agree. The dual notion of *agree sets*, that is, the maximal sets of attributes on which pairs of distinct tuples in r do agree, is commonly used as a starting point for computing disagree sets. Given an agree set X , the dual disagree set is just the set difference $R - X$.

The desired outcome of the hypergraph-based method is the *canonical cover* of satisfied FDs consisting of only singular, non-trivial, and left-reduced FDs which are satisfied in a given relation r . Discovering the canonical cover is sufficient because it implies all FDs which are satisfied in r . Every attribute $A \in R$ may yield an FD $X \rightarrow A$ that belongs to the canonical cover. Hence we find the *necessary set* with respect to A , denoted by $nec(r, A)$, which consists of all disagree sets of r that contain A . Note that $nec(r, A)$ contains the disagree sets for every pair of tuples which do not agree on the RHS A . It follows that an FD $X \rightarrow A$ is satisfied in r if and only if X is a hitting set of the hypergraph $\mathcal{H} = (R, nec(r, A))$. By definition, an FD $X \rightarrow Y$ is trivial if and only if $Y \subset X$. Moreover left-reduced FDs are those with \subseteq -minimal LHSs. Hence, we can find satisfied FD $X \rightarrow A$ which are both left-reduced and non-trivial by removing A from each disagree sets in $nec(r, A)$ before computing the minimal hitting sets.

A popular alternative to the hypergraph-based method is a *level-wise* examination of syntactically valid FDs (e.g. [30, 48, 58, 76, 101]). For such approaches, it is important to identify easy means for checking the satisfaction of any given FD and a suitable enumeration sequence and efficient pruning criteria for traversing the search space of all valid FDs. Of particular interest among the level-wise approaches is TANE [47, 48] through which Huhtala et. al. advocate the use of partitions in the realm of FD discovery. Partitions of tuples are used for capturing information about which tuples agree on each set of attributes. The partitions induced by singleton sets of attributes are computed from the relation itself and from such partitions we can compute all the partitions induced by other sets of attributes. TANE exploits the partitions for checking FD satisfactions. This is motivated by a desire to reduce the frequency of potentially expensive access to the given relation.

Many recent approaches for FD discovery have followed suit and also adopt partitions as an auxiliary source of information about value-equality. In particular, Dep-Miner [61] and FastFDs [100] augment the hypergraph-based method with partitions. Each work focuses on improving different aspects of the hypergraph-based method: Dep-Miner addresses the problem of computing agree sets more efficiently, while FastFDs targets the problem of computing minimal hitting sets more efficiently.

In [61], Lopes et. al. offers an alternative characterisation of agree sets. The authors consider two approaches for determining agree sets from the partitions. The naive approach is to iterate through the partitions, identify all pair of tuples belonging to the same partition class and use the information to update the agree set for this pair. Alternatively, the authors propose a characterisation of agree sets in terms of $ec(t)$ - which collates information about which partition classes each tuple t belongs to. The paper shows that an agree set for two distinct tuples t_1, t_2 is found by examining the intersection $ec(t_1) \cap ec(t_2)$. The benefit of this approach over the naive one is that we can

determine the agree set for any two distinct tuples easily, thus shortening the time it takes to output the first agree set and reducing the space requirement for the computation of all agree sets. Still, a potential source of inefficiency is that we need to determine the agree set for *every* pair of distinct tuples whose agree sets is non-empty.

FastFDs proposes a “depth-first, heuristic driven” strategy for computing hitting sets. In contrast to standard level-wise strategies, the depth-first algorithm exhibits several desirable features: the time it takes to discover the first FD is shortened and it is more space efficient to compute all hitting sets. In a greedy approach, we compute each hitting set by incrementally adding new attributes which belongs to the most number of disagree sets for which the old set is not a transversal. Every hitting set of the hypergraph contains some hitting set which is found by the depth-first algorithm, but we are *not* guaranteed to find *only* minimal hitting sets. This means that we can find all FDs which *potentially* belong to the canonical cover from the hitting sets returned by the algorithm but the minimality of each generated LHS needs to be verified before we can confirm an FD *actually* belongs to the canonical cover. The paper also shows some experimental results comparing FastFDs, Dep-Miner and TANE.

In the context of XML, relatively few works have addressed dependency discovery. Discovery of XML keys have been discussed by Grahne and Zhu in [33]. Zhou has investigated approaches for discovering the class of XFDs proposed by Arenas and Libkin [104]. While Yu and Jagadish have presented a level-wised procedure for discovering a class of path-based XFDs evaluated over generalised tree tuples [102, 103]. Both works, by Zhou and by Yu and Jagadish, first transform XML data into relational data and then search for satisfied dependencies. Zhou adapts algorithms originally proposed for discovering FDs while Yu and Jagadish propose their own partition-based approach for discovery that partially applies ideas from the relational context. In both works, partitions of tuples are used for determining satisfiability of functional dependencies.

In this chapter, we investigate the pXFD discovery problem as follows:

Given a T -compatible data tree T' , find all pXFDs which are satisfied in T' .

The approach we proposed in this chapter differs from the works of Zhou, Yu and Jagadish in several ways. Most conspicuously, the classes of XFD differ in terms of expressibility. But also, by considering v -ancestors and v -subgraphs as well as walks, we face a larger and more intricate search space of valid dependencies. cursory investigation indicates that a systematic examination of syntactically valid pXFDs is quite challenging. Fortunately, our transversal approach for discovering pXFDs is natural and preserves the basic ideas of the hypergraph transversal method of Mannila and Rähkä.

Some preliminary work relating to the discovery of HL-XFDs has been presented in [88]. Significant differences between the transversal approach for discovering HL-XFDs and for discovering pXFDs include: the ordering of potential LHSs and RHSs, and the characterisation for a dependency being trivial. We will also delve in more details into the issue of computing minimal transversals, the determination of all candidate RHSs, and computation of v -agree sets. In particular, several algorithms are suggested for solving these important sub-problems.

3.1 Transversal Approach for pXFD Discovery

In this section, we will sketch our proposed transversal approach for pXFDs discovery. We formally show the correctness of the approach by reducing the pXFD discovery problem to the problem of finding minimal transversals of certain families of v -difference sets. This reduction is carried out in two steps:

1. Reduce the pXFD discovery problem to that of computing the canonical pXFD-cover
2. Reduce the problem of computing the canonical pXFD-cover to that of finding all minimal transversals of certain families of v -difference sets.

The first part of the reduction establishes that the canonical pXFD-cover of any data tree T' is a suitable representation of the set of all pXFDs which are satisfied in T' . The second part of the reduction presents the notions of v -difference sets of T' , transversals of a family of v -difference sets and discuss the relationship between transversal of particular families of v -difference sets of T' and the canonical pXFD-cover of T' .

3.1.1 Canonical pXFD-Cover for Representing Satisfied pXFDs

As remarked upon earlier, there is potentially a very large number of syntactically valid pXFDs according to Definition 1.6, all of which may hold in T' . This means that explicitly enumerating *all satisfied pXFDs* is often inefficient and impractical, not to mention the issue of clarity and usefulness of such information. Fortunately, implications of pXFDs give rise to the notion of pXFD-covers. For two sets Σ_1, Σ_2 of pXFDs, we say Σ_2 is a *pXFD-cover* for Σ_1 if and only if Σ_2 has the same set of all implied pXFDs as Σ_1 , i.e., $(\Sigma_2)^* = (\Sigma_1)^*$.

For pXFD discovery it is not necessary that we find the smallest pXFD-cover for the set of all satisfied pXFDs, but rather a suitably smaller pXFD-cover that can be computed efficiently. This motivates us to define the notion of *canonical pXFD-cover* as per Definition 3.10. Intuitively, we arrive at our choice of what constitute a canonical pXFD-cover by determining which kind of satisfied pXFDs can be excluded because they are implied from other satisfied pXFDs.

Most obviously, the union rule and our discussion from Section 1.3.4 support the exclusion of any pXFD which is not both singular and canonical. The next part shows implication of singular canonical pXFDs that have comparable LHSs/RHSs with respect to p-subsumption. Before continuing, it is useful to note that Remark 1.37 relating p-subsumption and property-equality can be alternatively expressed as an inference rule which generalises the reflexivity, subgraph and ancestor axioms:

Lemma 3.1. *Let \mathcal{X}, \mathcal{Y} be sets of v -properties. The following rule is sound for the implication of pXFDs*

$$\frac{}{v : \mathcal{X} \rightarrow \mathcal{Y} \sqsubseteq \mathcal{X}} \quad (\sqsubseteq \text{-reflexivity})$$

Proof. The derivation tree for the \sqsubseteq -reflexivity rule using the axiomatisation $\mathfrak{F}_{general}$ (see Theorem 2.38) is as follows:

$$\frac{\frac{\frac{}{\forall X \in \mathcal{X}. v : \mathcal{X} \rightarrow \{X\}} \quad (1) \quad \frac{}{\forall Y \in \mathcal{Y}. \exists X \in \mathcal{X}. v : \{X\} \rightarrow \{Y\}} \quad (2)}{\forall Y_i \in \mathcal{Y}. v : \mathcal{X} \rightarrow \{Y\}} \quad (3)}{v : \mathcal{X} \rightarrow \mathcal{Y}} \quad (4)$$

(1):	Reflexivity
(2):	Subgraph or Ancestor
(3):	Transitivity
(4):	Union

□

Left-reduced and Right-maximal pXFDs

The soundness of the \sqsubseteq -reflexivity rule means that we can exclude pXFDs which do not have the \sqsubseteq -smallest LHS and the \sqsubseteq -largest RHS possible among those singular and canonical pXFDs which are satisfied. This is expressed by the notion of a pXFD being left-reduced or right-maximal with respect to some data tree T' .

Definition 3.2 (left-reduced pXFD). Let T' be a T -compatible data tree. A canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ over T is *left-reduced with respect to T'* if and only if

- T' satisfies $v : \mathcal{X} \rightarrow \{Y\}$, and
- set of v -properties \mathcal{W} with $\mathcal{W} \sqsubset \mathcal{X}$ implies that $v : \mathcal{W} \rightarrow \{Y\}$ does not hold in T' .

□

Definition 3.3 (right-maximal pXFD). let T' be a T -compatible data tree. A canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ over T is *right-maximal with respect to T'* if and only if

- T' satisfies $v : \mathcal{X} \rightarrow \{Y\}$, and
- essential v -property Z with $Y \sqsubset Z$ implies that $v : \mathcal{X} \rightarrow \{Z\}$ does not hold in T' .

□

That is, a pXFD which is left-reduced w.r.t. T' has the \sqsubseteq -minimal LHS among those singular, canonical pXFDs with the same RHS which are satisfied in T' , and likewise, a pXFD which is right-maximal w.r.t. T' has the \sqsubseteq -maximal RHS among those singular, canonical pXFDs with the same LHS which are satisfied in T' . Next we assert that a pXFD which is not both left-reduced and right-maximal w.r.t. T' can be implied by one which is. The next inference rule is useful for this purpose.

Lemma 3.4. *The p-subsumption rule defined as follows is sound*

$$\frac{v : \mathcal{W} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y}} \mathcal{W} \sqsubseteq \mathcal{X}$$

(*p* - subsumption)

Proof.

$$\frac{\frac{}{v : \mathcal{X} \rightarrow \mathcal{W}} \text{ (}\sqsubseteq\text{-reflexivity)} \quad v : \mathcal{W} \rightarrow \mathcal{Y}}{v : \mathcal{X} \rightarrow \mathcal{Y}} \text{ (transitivity)}$$

□

It is sufficient to examine singular canonical pXFDs which are left-reduced because every satisfied unary canonical pXFDs which is not left-reduced can be implied from one which is.

Lemma 3.5. *Let T' be a T -compatible data tree and $v : \mathcal{X} \rightarrow \{Y\}$ be a canonical pXFD which holds in T' . If $v : \mathcal{X} \rightarrow \{Y\}$ is not left-reduced w.r.t. T' then it is implied by some canonical pXFD $v : \mathcal{W}^k \rightarrow \{Y\}$ with $\mathcal{W}^k \sqsubseteq \mathcal{X}$ which is left-reduced w.r.t. T' .*

Proof. If $v : \mathcal{X} \rightarrow \mathcal{Y}$ is not left-reduced w.r.t. T' then, by definition, there is some $\mathcal{W} \sqsubseteq \mathcal{X}$ such that T' satisfies $v : \mathcal{W} \rightarrow \mathcal{Y}$. Similarly, if $v : \mathcal{W} \rightarrow \mathcal{Y}$ is not left-reduced w.r.t. T' , then there is some $\mathcal{W}^2 \sqsubseteq \mathcal{W}$ such that T' satisfies $v : \mathcal{W}^2 \rightarrow \mathcal{Y}$. Continuing in this way, we obtain a sequence $\mathcal{W}^k \sqsubseteq \mathcal{W}^{k-1} \sqsubseteq \dots \sqsubseteq \mathcal{W}^2 \sqsubseteq \mathcal{W} \sqsubseteq \mathcal{X}$ whereby $v : \mathcal{W}^k \rightarrow \mathcal{Y}$ is left-reduced w.r.t. T' . The sequence is always finite because the empty subgraph is p-subsumed by all core sets but strictly p-subsumes none of the core sets (Remark 1.39). That \mathcal{W}^k must be a core set follows from Lemma 1.40. Thus $v : \mathcal{W}^k \rightarrow \{Y\}$ is a canonical pXFD. From transitivity of \sqsubseteq we have $\mathcal{W}^k \sqsubseteq \mathcal{X}$. By application of the *p*-subsumption rule (see Lemma 3.4) it follows that $v : \mathcal{W}^k \rightarrow \mathcal{Y}$ implies $v : \mathcal{X} \rightarrow \mathcal{Y}$. □

Similarly, it is sufficient to examine singular canonical pXFDs which are right-maximal because every satisfied unary canonical pXFDs which is not right-maximal can be implied from one which is.

Lemma 3.6. *Let T' be a T -compatible data tree and $v : \mathcal{X} \rightarrow \{Y\}$ be a canonical pXFD which holds in T' . If $v : \mathcal{X} \rightarrow \{Y\}$ is not right-maximal w.r.t. T' then it is implied by some canonical pXFD $v : \mathcal{X} \rightarrow \{Z^k\}$ with $Y \sqsubseteq Z^k$ which is right-maximal w.r.t. T' .*

Proof. The proof is similar to that of Lemma 3.5. By consulting the definition of right-maximality, we find the sequence of canonical pXFDs which are all satisfied in T' :

$$v : \mathcal{X} \rightarrow \{Z\}, v : \mathcal{X} \rightarrow \{Z^2\}, \dots, v : \mathcal{X} \rightarrow \{Z^{k-1}\}, v : \mathcal{X} \rightarrow \{Z^k\}$$

where $Y \sqsubseteq Z \sqsubseteq Z^2 \sqsubseteq \dots \sqsubseteq Z^{k-1} \sqsubseteq Z^k$ and $v : \mathcal{X} \rightarrow \{Z^k\}$ is a canonical pXFD which is right-maximal w.r.t. T' . The sequence is finite because every essential v -subgraph is contained in some v -unit which is \sqsubseteq -maximal and every v -ancestor has \sqsubseteq -maximal

$n \in \vartheta(\{v\})$ as a descendant (Remark 1.39). Note that $v : \mathcal{X} \rightarrow \{Z^k\}$ is canonical because $v : \mathcal{X} \rightarrow \{Y\}$ is canonical and Z^k is an essential v -property. By transitivity of p -subsumption we have $Y \sqsubset Z^k$ and $v : \{Z^k\} \rightarrow \{Y\}$ is derivable by applying \sqsubseteq -reflexivity rule. Together with $v : \mathcal{X} \rightarrow \{Z^k\}$ and application of the transitivity rule, we derive $v : \mathcal{X} \rightarrow \{Y\}$. Hence, $v : \mathcal{X} \rightarrow \{Z^k\}$ implies $v : \mathcal{X} \rightarrow \{Y\}$ which completes the proof. \square

By applying the two previous lemmas together, we can make the stronger observation that any singular canonical pXFD which is satisfied in T' but not both left-reduced and right-maximal w.r.t. T' can be implied by one which is.

Theorem 3.7. *Let T' be a T -compatible data tree and $v : \mathcal{X} \rightarrow \{Y\}$ be a canonical pXFD over T . If T' satisfies $v : \mathcal{X} \rightarrow \{Y\}$ then $v : \mathcal{X} \rightarrow \{Y\}$ is implied by some canonical pXFD $v : \mathcal{W} \rightarrow \{Z\}$ with $\mathcal{W} \sqsubseteq \mathcal{X}$ and $Y \sqsubseteq Z$ which is left-reduced and right-maximal w.r.t. T' .*

Proof. The statement is true if $v : \mathcal{X} \rightarrow \{Y\}$ is both left-reduced and right-maximal w.r.t. T' . So assume it is not left-reduced and/or not right-maximal w.r.t. T' .

If $v : \mathcal{X} \rightarrow \{Y\}$ is not left-reduced w.r.t. T' then Lemma 3.5 states $v : \mathcal{X} \rightarrow \{Y\}$ is implied by some canonical pXFD $v : \mathcal{W} \rightarrow \{Y\}$ with $\mathcal{W} \sqsubset \mathcal{X}$ which is left-reduced w.r.t. T' . Moreover, if $v : \mathcal{W} \rightarrow \{Y\}$ is not right-maximal w.r.t. T' , it is implied by some canonical pXFD $v : \mathcal{W} \rightarrow \{Z\}$ with $Y \sqsubset Z$ which is right-maximal w.r.t. T' (Lemma 3.6). Now suppose $v : \mathcal{W} \rightarrow \{Z\}$ is not left-reduced w.r.t. T' . Like above, we deduce that $v : \mathcal{W} \rightarrow \{Z\}$ is implied by some canonical pXFD $v : \mathcal{W}' \rightarrow \{Z\}$ with $\mathcal{W}' \sqsubset \mathcal{W}$ which is left-reduced w.r.t. T' . But from $v : \mathcal{W}' \rightarrow \{Z\}$ we can derive $v : \mathcal{W}' \rightarrow \{Y\}$ by application of the \sqsubseteq -reflexivity and transitivity rule. This would contradict $v : \mathcal{W} \rightarrow \{Y\}$ being left-reduced w.r.t. T' . Hence $v : \mathcal{W} \rightarrow \{Z\}$ is also right-maximal w.r.t. T' . Implication of $v : \mathcal{X} \rightarrow \{Y\}$ from canonical pXFD $v : \mathcal{W} \rightarrow \{Z\}$, which is both left-reduced right-maximal w.r.t. T' , is implicit in the proof Lemma 3.5 and Lemma 3.6.

The case where $v : \mathcal{X} \rightarrow \{Y\}$ is not left-reduced nor right-maximal w.r.t. T' can be handled as above. This leaves the case of $v : \mathcal{X} \rightarrow \{Y\}$ being left-reduced but not right-maximal w.r.t. T' . Then it is implied by some canonical pXFD $v : \mathcal{X} \rightarrow \{Z\}$ with $Y \sqsubset Z$ which is right-maximal w.r.t. T' . If $v : \mathcal{X} \rightarrow \{Z\}$ is not left-reduced w.r.t. T' then it is implied by some canonical pXFD $v : \mathcal{W}' \rightarrow \{Z\}$ with $\mathcal{W}' \sqsubset \mathcal{X}$ which is left-reduced w.r.t. T' . Since $Y \sqsubset Z$ we can imply the canonical pXFD $v : \mathcal{W}' \rightarrow \{Y\}$ which contradicts $v : \mathcal{X} \rightarrow \{Y\}$ being left-reduced w.r.t. T' . Thus canonical pXFD $v : \mathcal{W} \rightarrow \{Z\}$ with $\mathcal{W} = \mathcal{X}$ is both left-reduced and right-maximal w.r.t. T' and implies $v : \mathcal{X} \rightarrow \{Y\}$. \square

Trivial pXFDs

There are certain pXFDs which are always satisfied, regardless of which T -compatible data tree we consider.

Definition 3.8 (trivial pXFD). A pXFD over T is *trivial* if it is satisfied in every T -compatible data tree, and *non-trivial* otherwise. \square

That is not to say that trivial pXFDs, expressing inherent structural properties of XML schema trees, are uninteresting. Having said that, trivial pXFDs are not interesting candidates in the context of pXFD discovery because it is sufficient to discover all trivial pXFDs once rather than for each given data tree.

The above definition is however difficult to apply for identifying trivial pXFDs. An alternative way to look at trivial pXFDs is that they are implied by the empty set of pXFDs. As such, just knowing that two pre-image trees agree on v -properties in the LHS should allow us to deduce that they also agree on v -properties in the RHS. This is true in the case of pXFDs which are derivable by application of an inference rule with no premise. Such inference rules in the axiomatisation $\mathfrak{F}_{canonical}$ (see Theorem 2.24) yield the following simple syntactic characterisation for trivial pXFDs featuring p-subsumption. The first condition verifies whether the LHS already determines the target trivially because as a direct result the LHS can also determine all potential RHSs trivially (recall Remark 2.39). In the second, we are verifying whether a pXFD is derivable by application of the \sqsubseteq -reflexivity rule.

Theorem 3.9. *Let σ be a pXFD $v : LHS_\sigma \rightarrow RHS_\sigma$ over T . Then σ is trivial if and only if at least one of the following statements hold true:*

1. $\vartheta(\{v\}) \sqsubseteq \vartheta(LHS_\sigma) \cup \{r_T\}$ or,
2. $\vartheta(RHS_\sigma) \sqsubseteq \vartheta(LHS_\sigma) \cup \{\emptyset, r_T\}$

Proof. (\Leftarrow) Assume neither of the statements hold. We show that there must exist some T -compatible data tree T' such that $\not\models_{T'} \sigma$. This means, T' contains two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_X \doteq p_2|_X$ for all $X \in LHS_\sigma$ but $p_1|_Y \neq p_2|_Y$ for some $Y \in RHS_\sigma$.

Firstly, v cannot be simple, since otherwise all v -ancestors refine to r_T , and thus $\vartheta(\{v\}) = \{r_T\} \sqsubseteq \vartheta(LHS_\sigma) \cup \{r_T\}$ contradicting the first statement above. The violation of the second statement gives rise to two cases:

Case 1: there is some v -ancestor $n \in \vartheta(RHS_\sigma)$ such that there is no v -ancestor $m \in \vartheta(LHS_\sigma) \cup \{\emptyset, r_T\}$ which p-subsumes n

Let m be the \sqsubseteq -maximal v -ancestor in $\vartheta(LHS_\sigma) \cup \{r_T\}$. Then n must be a proper non-simple descendant of m , and there exists some arc (u, w) on the path from m to n with frequency other than $?$ and 1 . In this case, T' can be constructed by merging two property-equal copies of T such that they share every node except their pre-image of w and all its descendants. It is easy to see that T' contains p_1, p_2 as described above.

Case 2: there is some v -subgraph $Y \in \vartheta(RHS_\sigma)$ such that there is no v -subgraph $X \in \vartheta(LHS_\sigma) \cup \{\emptyset, r_T\}$ which p-subsumes Y

For this case we will apply the construction in Lemma 2.19 to generate a counter-example two- v -pre-image data tree T' in which σ (i.e., $v : LHS_\sigma \rightarrow \{Y\}$) is violated. Recall that the input for the construction is an equality set \mathcal{E}_T .

For the subset $\mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$, we take:

- $\vartheta(LHS_\sigma) \cup \{\emptyset\} - \check{\mathbb{A}}_T(v) \subseteq \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$
- if $W \in \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$ and Z is contained in W then $Z \in \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$
- if $W, Z \in \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$ and W, Z are v -reconcilable subgraph then $W \cup Z \in \mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$
- nothing else belongs to $\mathcal{E}_T \cap \check{\mathcal{S}}_T(v)$.

Let $m = \text{lca}(\vartheta(LHS_\sigma) \cup \{r_T\})$. Then m is the \sqsubseteq -maximal v -ancestor belonging to $\vartheta(LHS_\sigma) \cup \{r_T\}$. There must exist some arc (u, w) on the path from m to v with frequency other than ? and 1, otherwise $\vartheta(\{v\}) = \{m\} \sqsubseteq \vartheta(LHS_\sigma) \cup \{r_T\}$ which would contradict the initial assumption that $\vartheta(\{v\}) \not\sqsubseteq \vartheta(LHS_\sigma) \cup \{r_T\}$. Therefore, let $\mathcal{E}_T \cap \check{\mathbb{A}}_T(v) = \{\text{all ancestors of } u\}$. The set $\mathcal{E}_T = (\mathcal{E}_T \cap \check{\mathcal{S}}_T(v)) \cup (\mathcal{E}_T \cap \check{\mathbb{A}}_T(v))$ clearly possesses all properties of an equality set as per Definition 2.8.

After applying the construction in Lemma 2.19 with equality set \mathcal{E}_T we obtain a two- v -pre-image data tree T' with pre-image trees p_1, p_2 such that

$$p_1|_W \doteq p_2|_W \text{ if and only if } W \in \mathcal{E}_T \text{ for all } v\text{-property } W \in \check{\mathcal{S}}_T(v) \cup \check{\mathbb{A}}_T(v).$$

From above, we infer that $LHS_\sigma \cup \{\emptyset, r_T\} \subseteq \mathcal{E}_T$. Moreover $Y \notin \mathcal{E}_T$ because Y cannot be contained in any $Z \in \vartheta(LHS_\sigma) \cup \{\emptyset, r_T\}$ and all other v -subgraphs in \mathcal{E}_T are not essential. That is T' which violates $v : LHS_\sigma \rightarrow \{Y\}$.

(\Rightarrow) We show that $\{\} \vdash \sigma$ whenever one of the statement from the proposition holds.

Case 1: $\vartheta(\{v\}) \sqsubseteq \vartheta(LHS_\sigma) \cup \{r_T\}$

$$\frac{\frac{\frac{\overline{v : \vartheta(LHS_\sigma) \cup \{r_T\} \rightarrow \vartheta(\{v\})} \quad (1)}{v : \vartheta(LHS_\sigma) \cup \{r_T\} \rightarrow \{v\}} \quad (2)}{\forall Y \in \check{\mathcal{S}}_T(v) \cup \check{\mathbb{A}}_T(v). v : \{v\} \rightarrow \{Y\}} \quad (3)}{\forall Y \in \check{\mathcal{S}}_T(v) \cup \check{\mathbb{A}}_T(v). v : \vartheta(LHS_\sigma) \cup \{r_T\} \rightarrow \{Y\}} \quad (4)}{\frac{\forall Y \in \check{\mathcal{S}}_T(v) \cup \check{\mathbb{A}}_T(v). v : LHS_\sigma \rightarrow \{Y\}}{v : LHS_\sigma \rightarrow RHS_\sigma} \quad (5)}{\quad (6)}$$

- | | |
|------|--|
| (1): | \sqsubseteq -reflexivity |
| (2): | Simple descendant |
| (3): | Remark 2.39: target, subgraph, join and ancestor etc |
| (4): | Transitivity |
| (5): | Subgraph reinstatement, ancestor reinstatement and trim root |
| (6): | Union |

Case 2: $\vartheta(RHS_\sigma) \sqsubseteq \vartheta(LHS_\sigma) \cup \{\emptyset, r_T\}$

$$\frac{}{v : \vartheta(LHS_\sigma) \cup \{\emptyset, r_T\} \rightarrow \vartheta(RHS_\sigma)} \quad (1)$$

$$\frac{}{LHS_\sigma \cup \{\emptyset, r_T\} \rightarrow RHS_\sigma} \quad (2)$$

$$\frac{}{v : LHS_\sigma \rightarrow RHS_\sigma} \quad (3)$$

- (1): \sqsubseteq -reflexivity
(2): Subgraph reinstatement and ancestor reinstatement
(3): Trim empty subgraph and trim root

□

The previous characterisation of trivial pXFDs can be applied to discover all trivial pXFDs and to verify whether a given pXFD is trivial, and moreover we need only to consider the XML schema tree.

Canonical pXFD-Cover

All discussion in the section thus far culminates in the following definition.

Definition 3.10 (canonical pXFD-cover). Let T' be a T -compatible data tree. The *canonical pXFD-cover* of T' is defined by

$$\mathfrak{C}_{T'}(v) = \{v : \mathcal{X} \rightarrow \{Y\} \mid v : \mathcal{X} \rightarrow \{Y\} \text{ is a canonical non-trivial pXFD which is left-reduced and right-maximal with respect to } T'\}$$

□

This takes us to the first important result for the chapter: that the canonical pXFD-cover of T' accurately represents the set of all satisfied pXFDs in T' . We show this formally in the next pair of theorems. The first establishes that the canonical pXFD-cover of T' can be used to represent the set of *all canonical* pXFDs which are satisfied in T' . Then in the second theorem, we recognise that, in fact, the canonical pXFD-cover of T' can be used to represent the set of *all* pXFDs which hold in T' , irrespective of whether they are canonical.

Theorem 3.11. *Let $v : \mathcal{X} \rightarrow \{Y\}$ be a pXFD over T and let T' be a T -compatible data tree. If $v : \mathcal{X} \rightarrow \{Y\}$ is canonical and $\models_{T'} v : \mathcal{X} \rightarrow \{Y\}$ then $v : \mathcal{X} \rightarrow \{Y\}$ is implied by $\mathfrak{C}_{T'}(v)$.*

Proof. The proof is by contradiction and assumes canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ is satisfied in T' but is not implied by $\mathfrak{C}_{T'}(v)$. This means, firstly, that $v : \mathcal{X} \rightarrow \{Y\}$ is non-trivial. If, in addition, $v : \mathcal{X} \rightarrow \{Y\}$ is both left-reduced and right-maximal w.r.t. T' then it belongs to $\mathfrak{C}_{T'}(v)$ which contradicts that it is not implied by $\mathfrak{C}_{T'}(v)$. On other hand, Theorem 3.7 states that $v : \mathcal{X} \rightarrow \{Y\}$ is implied by some canonical pXFD $v : \mathcal{W} \rightarrow \{Z\}$ where $\mathcal{W} \sqsubseteq \mathcal{X}$ and $Y \sqsubseteq Z$ which is left-reduced and right-maximal w.r.t. T' . It must be that $v : \mathcal{W} \rightarrow \{Z\}$ is trivial, since otherwise $v : \mathcal{W} \rightarrow \{Z\}$ would belong to $\mathfrak{C}_{T'}(v)$ giving another contradiction to $v : \mathcal{X} \rightarrow \{Y\}$ not being implied by $\mathfrak{C}_{T'}(v)$.

Recall Lemma 1.41. There are two conditions under which $v : \mathcal{W} \rightarrow \{Z\}$ is trivial.

Case 1: $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{W}) \cup \{r_T\}$

From $\mathcal{W} \sqsubseteq \mathcal{X}$ we get $\vartheta(\mathcal{W}) \sqsubseteq \vartheta(\mathcal{X})$ and so

$$\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{W}) \cup \{r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$$

Case 2: $\vartheta(\{Z\}) \sqsubseteq \vartheta(\mathcal{W}) \cup \{\emptyset, r_T\}$

From $\mathcal{W} \sqsubseteq \mathcal{X}$ and $Y \sqsubseteq Z$ we get $\vartheta(\mathcal{W}) \sqsubseteq \vartheta(\mathcal{X})$ and $\vartheta(\{Y\}) \sqsubseteq \vartheta(\{Z\})$ respectively. And so

$$\vartheta(\{Y\}) \sqsubseteq \vartheta(\{Z\}) \sqsubseteq \vartheta(\mathcal{W}) \cup \{\emptyset, r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$$

What we have just shown is that under all possible conditions where $v : \mathcal{W} \rightarrow \{Z\}$ is trivial then $v : \mathcal{X} \rightarrow \{Y\}$ is also trivial, a contradiction. Thus complete the proof. \square

Theorem 3.12. *If $\models_{T'} v : \mathcal{X} \rightarrow \{Y\}$ then $v : \mathcal{X} \rightarrow \{Y\}$ is implied by $\mathfrak{C}_{T'}(v)$.*

Proof. Follows from Theorem 2.38 and Theorem 3.11. \square

3.1.2 Minimal Transversals for Finding Canonical pXFD-Cover

Next we consider how to compute the canonical pXFD-cover of some given T -compatible data tree T' . We propose a transversal approach which is motivated by the works of Mannila and R aih a [68, 69, 70, 71, 72].

Remark 3.13. It follows from Remark 2.39 that when v is simple then all pXFDs are trivial and $\mathfrak{C}_{T'}(v) = \{\}$ for all T -compatible data tree T' . The immediate consequence is, no discovery action is actually needed in the case that target v is simple. Therefore, in our proposed transversal approach we assume that target v is not simple.

Similar to Mannila and R aih a, we think about pXFD satisfaction in a different way than the formal definition. A pXFD is satisfied if there exist no pair of pre-image trees witnessing its violation, which means:

T' satisfies some pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$ if and only if any two pre-image trees in T' which differ on some v -properties in \mathcal{Y} also differ on all v -properties in \mathcal{X} .

It is thus useful to gather together, for every pair of pre-image trees the essential v -properties on which they differ, and to be able to group together all such difference sets which contain a given essential v -property.

Definition 3.14 (difference set). Let T' be a T -compatible data tree and $p_1, p_2 \in P_{T'}(v)$ be two pre-image trees of v .

- $\mathcal{D}_{\{p_1, p_2\}}(v) = \{X \mid X \in \mathbf{E}_T^{\mathfrak{S}}(v) \cup \mathbf{E}_T^{\mathfrak{A}}(v) \text{ and } p_1|_X \neq p_2|_X\}$
- $\mathfrak{D}_{T'}(v) = \{\mathcal{D}_{\{p_1, p_2\}}(v) \mid p_1, p_2 \in P_{T'}(v) \text{ and } p_1 \neq p_2\}$

- $\mathfrak{D}_{T'}^Y(v) = \{\mathcal{D} \in \mathfrak{D}_{T'}(v) \mid Y \in \mathcal{D}\}$

We call $\mathcal{D}_{\{p_1, p_2\}}(v)$ the v -difference set of p_1 and p_2 , $\mathfrak{D}_{T'}(v)$ the family of all v -difference sets of T' and, $\mathfrak{D}_{T'}^Y(v)$ the family of v -difference sets of T' modulo Y . \square

Remark 3.15. The notion of v -difference sets is analogous to the idea of non-equality set from Section 2.1.3. Provided the target node v is *not simple* and T' contains *more than one* pre-image tree of v , every v -difference set of T' is non-empty, and more specifically contain some essential v -ancestor. This is because two distinct pre-image trees of v must differ on at least v and consequently will differ on $n \in \vartheta(\{v\})$ which is essential. \square

Notations: Whenever two pre-image trees p_1, p_2 differ on some essential v -property X then p_1, p_2 also differ on any essential v -property $X \sqsubseteq X'$. That is, if $X \in \mathcal{D}$ then $X \sqsubseteq X'$ implies $X' \in \mathcal{D}$. Thus every v -difference set is upward-closed with respect to p-subsumption and can be represented by its \sqsubseteq -minimal elements. For conciseness and clarity, examples will denote a v -difference set \mathcal{D} by its representative subset $\sqsubseteq\text{-min}(\mathcal{D})$ of \sqsubseteq -minimal elements, unless stated otherwise. \square

Recall that we are only interested in discovering *singular canonical* pXFDs which are satisfied in T' . In particular, the RHS of such a pXFD contains exactly one essential v -property. The family $\mathfrak{D}_{T'}^Y(v)$ identifies all pairs of pre-image trees in T' which differ on Y together with all other essential v -properties on which these pairs of pre-image trees differ. Therefore our alternate view of pXFD satisfaction above states that a pXFD $v : LHS_\sigma \rightarrow \{Y\}$ is satisfied in T' if and only if LHS_σ is a set of essential v -properties which contains at least one element from every v -difference set in $\mathfrak{D}_{T'}^Y(v)$. Such sets of essential v -properties are transversals of $\mathfrak{D}_{T'}^Y(v)$ according to the subsequent definition. We investigate possible ways to compute minimal transversals in Section 3.2.

Definition 3.16 (transversal). For a family of sets of essential v -properties \mathfrak{D} , a *transversal* \mathcal{T} of \mathfrak{D} is a (possibly empty) set of essential v -properties such that for every $\mathcal{D} \in \mathfrak{D}$ we have $\mathcal{T} \cap \mathcal{D} \neq \{\}$. Moreover \mathcal{T} is a *minimal transversal* of \mathfrak{D} if no $\mathcal{T}' \sqsubset \mathcal{T}$ is a transversal of \mathfrak{D} . Specifically $\{\}$ is the only minimal transversal of $\mathfrak{D} = \{\}$. By $\mathfrak{Tr}(\mathfrak{D})$ we denote the family of all minimal transversals of \mathfrak{D} . \square

Example 3.17. Consider the purchase data tree $T'_{purchase}$ from Figure 3.1 and the corresponding schema tree $T_{purchase}$ as shown in Figure 3.2. Note that the leaf labels of $T_{purchase}$ are distinguishable by the first two letters. And so, every walk will be denoted by the first two letters of its leaf's label.

Let $v_{Purchase}$ be the target node. There are four pre-images of $v_{Purchase}$ in $T'_{purchase}$: i_5, i_{18}, i_{31} and i_{44} . These node ids identify four corresponding pre-image trees in $T'_{purchase}$. The family of $v_{Purchase}$ -difference sets of $T'_{purchase}$ is as follows:

$$\begin{aligned} \mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase}) &= \{\text{Pn}, \{\text{De}, \text{Pr}\}, \text{Di}, \text{Sa}, v_{Purchase}\} \\ \mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}) &= \{\text{Ti}, \text{Pn}, \text{De}, \text{Pr}, v_{Outlet}\} \\ \mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase}) &= \{\text{Ti}, \text{Pn}, \text{Pr}, \text{Di}, \text{Sa}, v_{Outlet}\} = \mathcal{D}_{\{i_{18}, i_{44}\}}(v_{Purchase}) \\ \mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase}) &= \{\text{Ti}, \text{Pn}, \text{De}, \text{Pr}, \text{Di}, \text{Sa}, v_{Outlet}\} \\ \mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase}) &= \{\text{Pn}, \text{De}, \text{Di}, \text{Sa}, v_{Purchase}\} \end{aligned}$$

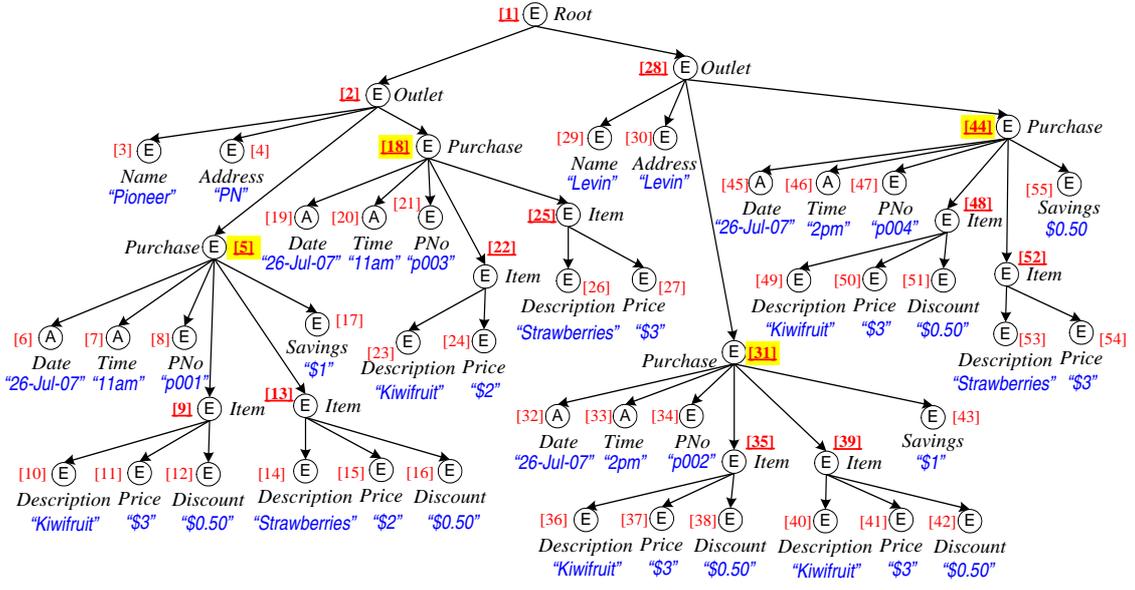


Figure 3.1: $T_{purchase}$ -compatible XML data tree $T'_{purchase}$.

All five $v_{Purchase}$ -difference sets of $T'_{purchase}$ constitute the family $\mathcal{D}_{T'_{purchase}}^{\{De,Pr,Di\}}(v_{Purchase})$ which has the following minimal transversals: $\{Pn\}$, $\{\{De,Pr\}\}$, $\{De,Di\}$, $\{De,Sa\}$, $\{Di,Ti\}$, $\{Di,Pr\}$, $\{Sa,Ti\}$, $\{Sa,Pr\}$, $\{v_{Outlet},Di\}$, $\{v_{Outlet},Sa\}$, and $\{v_{Purchase}\}$. Two non-minimal transversals of $\mathcal{D}_{T'_{purchase}}^{\{De,Pr,Di\}}(v_{Purchase})$ are: $\{Pn,Di\}$ and $\{\{De,Pr\},Pr\}$. For a demonstration of how to compute these minimal transversals see Example 3.37. \square

The following proposition shows that transversals enable us to find all singular canonical pXFDs which are satisfied in T' .

Proposition 3.18. *A canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ holds in T' if and only if \mathcal{X} is a transversal of $\mathcal{D}_{T'}^Y(v)$.*

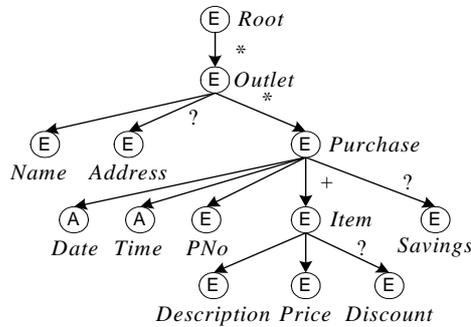


Figure 3.2: XML schema tree $T_{purchase}$

Proof. (\Leftarrow) Suppose \mathcal{X} is not a transversal for $\mathfrak{D}_{T'}^Y(v)$. Particularly, there are p_1, p_2 such that $\mathcal{D}_{\{p_1, p_2\}}(v) \in \mathfrak{D}_{T'}^Y(v)$ and $\mathcal{X} \cap \mathcal{D}_{\{p_1, p_2\}}(v) = \{\}$. But by Definition 3.14 $Y \in \mathcal{D}_{\{p_1, p_2\}}(v)$. It follows that p_1, p_2 attest to the violation of $v : \mathcal{X} \rightarrow \{Y\}$ in T' .

(\Rightarrow) Suppose T' violates $v : \mathcal{X} \rightarrow \{Y\}$. Then there are two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_Y \neq p_2|_Y$ but $p_1|_X = p_2|_X$ for all $X \in \mathcal{X}$. It follows that $Y \in \mathcal{D}_{\{p_1, p_2\}}(v)$ and no $X \in \mathcal{X}$ belongs to $\mathcal{D}_{\{p_1, p_2\}}(v)$. Therefore $\mathcal{D}_{\{p_1, p_2\}}(v) \in \mathfrak{D}_{T'}^Y(v)$ but $\mathcal{X} \cap \mathcal{D}_{\{p_1, p_2\}}(v) = \{\}$. Hence \mathcal{X} is not a transversal of $\mathfrak{D}_{T'}^Y(v)$. \square

Corollary 3.19. *A canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ is left-reduced w.r.t. T' if and only if \mathcal{X} is a minimal transversal of $\mathfrak{D}_{T'}^Y(v)$.* \square

A basic transversal approach for computing $\mathfrak{C}_{T'}(v)$ proceeds as follows:

```

proc transversal_discovery(basic)
for all essential  $v$ -property  $Y \in \mathbf{E}_T^{\check{S}}(v) \cup \mathbf{E}_T^{\check{A}}(v)$  do
  Compute  $\mathfrak{D}_{T'}^Y(v)$ 
  Compute  $\mathfrak{I}\mathfrak{r}(\mathfrak{D}_{T'}^Y(v))$ 
  /* Extract pXFD with  $\{Y\}$  as RHS */
  for all  $\mathcal{X} \in \mathfrak{I}\mathfrak{r}(\mathfrak{D}_{T'}^Y(v))$  do
    if  $v : \mathcal{X} \rightarrow \{Y\}$  is non-trivial and right-maximal w.r.t.  $T'$  then
       $\mathfrak{C}_{T'}(v) := \mathfrak{C}_{T'}(v) \cup \{v : \mathcal{X} \rightarrow \{Y\}\}$ 
    end if
  end for
end for

```

Applying Corollary 3.19 we find all canonical pXFDs with singleton RHS $\{Y\}$ which are left-reduced w.r.t. T' by computing the minimal transversals of $\mathfrak{D}_{T'}^Y(v)$. And then from these, we select only those pXFDs which are also both non-trivial and right-maximal w.r.t. T' . Naively, we can consider all essential v -properties to be *potential RHSs* and verify triviality and right-maximality of a pXFD by applying Definition 3.3 and Theorem 3.9. But in fact, it is possible to identify a smaller set of potential RHSs which leads to simpler conditions for verifying triviality and right-maximality. We will look briefly at these two issues before proposing our main transversal approach and formally proving its correctness.

Why should we not simply consider all essential v -properties as potential RHS? Let first look at an example. For the purchase data tree $T'_{purchase}$ we have

$$\mathfrak{D}_{T'_{purchase}}^{\{\text{De}, \text{Pr}\}}(v_{Purchase}) = \mathfrak{D}_{T'_{purchase}}^{\{\text{De}, \text{Di}\}}(v_{Purchase}) = \mathfrak{D}_{T'_{purchase}}^{\{\text{Pr}, \text{Di}\}}(v_{Purchase}) = \mathfrak{D}_{T'_{purchase}}^{\{\text{De}, \text{Pr}, \text{Di}\}}(v_{Purchase})$$

which means they all share exactly the same set of minimal transversals. This then means that *no* canonical pXFD with $\{\{\text{De}, \text{Pr}\}\}$, $\{\{\text{De}, \text{Di}\}\}$, or $\{\{\text{Pr}, \text{Di}\}\}$ as the RHS can be right-maximal w.r.t. $T'_{purchase}$. Most significantly, we know all this without having to find the minimal transversals for these families of $v_{Purchase}$ -difference sets. This example

illustrates that some essential v -property may not occur as the RHS for any singular canonical pXFD which is right-maximal w.r.t. T' . This motivates our subsequent definition of candidate RHSs. Obviously, pre-computing candidate RHSs before applying the transversal approach will improve efficiency over the basic transversal approach, provided we can identify the complete set of candidate RHS efficiently. One possible process to identify all candidate RHSs is discussed in Section 3.2.

Definition 3.20. Let T' be a T -compatible data tree.

$$\mathit{candRHS}_{T'}(v) = \{Y \in \mathbf{E}_T^{\check{S}}(v) \cup \mathbf{E}_T^{\check{A}}(v) \mid v : \{Y\} \rightarrow \{Y\} \text{ is right-maximal w.r.t. } T'\}$$

Members are $\mathit{candRHS}_{T'}(v)$ are called *candidate RHSs* (of T'). \square

The next lemma verifies that $\mathit{candRHS}_{T'}(v)$ include exactly those RHSs of singular canonical pXFDs which are right-maximal w.r.t. T' .

Lemma 3.21. *Let Y be an essential v -property of T . The following statements are equivalent:*

(i) $Y \in \mathit{candRHS}_{T'}(v)$

(ii) *There exists a canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ which is right-maximal w.r.t. T'*

Proof. (i) \Rightarrow (ii): Choose $\mathcal{X} = \{Y\}$.

$\neg(i) \Rightarrow \neg(ii)$: Let $v : \mathcal{X} \rightarrow \{Y\}$ be any canonical pXFD holding in T' . Since $Y \notin \mathit{candRHS}_{T'}(v)$ the definition of right-maximality tells us that there exists an essential v -property $Z \sqsupset Y$ such that $v : \{Y\} \rightarrow \{Z\}$ holds in T' . But this implies T' satisfies $v : \mathcal{X} \rightarrow \{Z\}$, yielding $v : \mathcal{X} \rightarrow \{Y\}$ is not right-maximal with respect to T' . \square

The set $\mathit{candRHS}_{T'}(v)$ is likely to be strictly smaller than the set of all essential v -properties. The above characterisation lamentably does not guarantee that there is a *non-trivial* canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ for every $Y \in \mathit{candRHS}_{T'}(v)$, nor that *every* singular canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ with $Y \in \mathit{candRHS}_{T'}(v)$ which is left-reduced w.r.t. T' is also right-maximal w.r.t. T' . That is, we still have to check whether a pXFD extracted from a minimal transversal computation is non-trivial and right-maximal w.r.t. T' .

Example 3.22. For example the pXFD $v_{Purchase} : \{\text{Di}\} \rightarrow \{\text{Di}\}$ is satisfied in data tree $T'_{purchase}$ but

$$\begin{aligned} v_{Purchase} &: \{\text{Di}\} \rightarrow \{\{\text{De}, \text{Di}\}\} \\ v_{Purchase} &: \{\{\text{Pr}, \text{Di}\}\} \\ v_{Purchase} &: \{\{\text{De}, \text{Pr}, \text{Di}\}\} \end{aligned}$$

are all violated. Therefore $v_{Purchase} : \{\text{Di}\} \rightarrow \{\text{Di}\}$ is right-maximal since these are all the possible counter-examples. The pXFD $v_{Purchase} : \{\{\text{De}, \text{Pr}, \text{Di}\}\} \rightarrow \{\{\text{De}, \text{Pr}, \text{Di}\}\}$ is right-maximal because there is no essential $v_{Purchase}$ -subgraph which strictly p-subsumes the essential $v_{Purchase}$ -subgraph $\{\text{De}, \text{Pr}, \text{Di}\}$. However both pXFDs are trivial.

The pXFDs $v_{Purchase} : \{\text{Pn}\} \rightarrow \{\text{Di}\}$ and $v_{Purchase} : \{\text{Pn}\} \rightarrow \{\{\text{De}, \text{Pr}, \text{Di}\}\}$ are also satisfied by $T'_{purchase}$. Since there are pairs of pre-image trees of v with distinct projections to Di we know $v_{Purchase} : \{\} \rightarrow \{\text{Di}\}$ is violated. This means that

$$\begin{aligned} v_{Purchase} &: \{\text{Pn}\} \rightarrow \{\text{Di}\}, \text{ and} \\ v_{Purchase} &: \{\text{Pn}\} \rightarrow \{\{\text{De}, \text{Pr}, \text{Di}\}\} \end{aligned}$$

are left-reduced with a candidate RHS as the RHS. However, $v_{Purchase} : \{\text{Pn}\} \rightarrow \{\text{Di}\}$ is not right-maximal because $v_{Purchase} : \{\text{Pn}\} \rightarrow \{\{\text{De}, \text{Pr}, \text{Di}\}\}$ also holds in $T'_{purchase}$ with $\text{Di} \sqsubset \{\text{De}, \text{Pr}, \text{Di}\}$. \square

Unlike the approach of Mannila and R aih a, our job is not finished once the minimal transversals are computed. On a brighter note, the check for whether a pXFD $v : \mathcal{X} \rightarrow \{Y\}$ with $Y \in \text{candRHS}_{T'}(v)$ and $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v))$ is trivial and right-maximal w.r.t. T' is more straightforward than stated in Definition 3.3 and Theorem 3.9. For right-maximality, we check only essential v -properties which are *candidate RHSs* (as opposed to all essential v -properties), while for triviality, we check the simple conditions in the next lemma.

Lemma 3.23. *Consider a pXFD $v : \mathcal{X} \rightarrow \{Y\}$ where $Y \in \mathbf{E}_T^{\check{S}}(v) \cup \mathbf{E}_T^{\check{A}}(v)$ and \mathcal{X} is a minimal transversal of $\mathfrak{D}_v^Y(T')$. Then $v : \mathcal{X} \rightarrow \{Y\}$ is trivial if and only if one of the following statements holds*

1. v is simple
2. $Y \in \{\emptyset, r_T\}$
3. $\mathcal{X} = \{Y\}$ or $\mathcal{X} = \vartheta(\{v\})$

Proof. (\Rightarrow) Suppose $v : \mathcal{X} \rightarrow \{Y\}$ is trivial. For each of the two statements in Theorem 3.9 we show a reduction to one of the statements above.

Case 1: $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$

Because $\vartheta(\{v\})$ is a singleton set, either $\vartheta(\{v\}) \sqsubseteq \{r_T\}$ or $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X})$. In the first instance, v is simple which is the first statement in the lemma. In the second instance, $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X})$. Any two distinct pre-image trees differ on v and thus cannot agree on the single v -ancestor belonging to $\vartheta(\{v\})$. This means $\vartheta(\{v\})$ is a subset of every v -difference set and so $\vartheta(\{v\})$ is a transversal of $\mathfrak{D}_{T'}^Y(v)$. From $\vartheta(\mathcal{X}) \sqsubseteq \mathcal{X}$ (Lemma 1.40) we get $\vartheta(\{v\}) \sqsubseteq \mathcal{X}$ which implies $\vartheta(\{v\}) = \mathcal{X}$ otherwise we violate \mathcal{X} being a minimal transversal. This is the third statement in the lemma.

Case 2: $\vartheta(\{Y\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

Because Y is an essential v -property, it is the case that $\vartheta(\{Y\}) = \{Y\}$. This means either $\vartheta(\{Y\}) \sqsubseteq \{\emptyset, r_T\}$ or $\vartheta(\{Y\}) \sqsubseteq \vartheta(\mathcal{X})$. In the first instance, $Y \in \{\emptyset, r_T\}$. This is the second statement in the lemma. In the second instance, $\vartheta(\{Y\}) \sqsubseteq \vartheta(\mathcal{X})$. By definition, every v -difference set in $\mathfrak{D}_{T'}^Y(v)$ contains Y and thus $\{Y\}$ is a transversal of $\mathfrak{D}_{T'}^Y(v)$. Because $\vartheta(\mathcal{X}) \sqsubseteq \mathcal{X}$ we get $\{Y\} \sqsubseteq \mathcal{X}$. Again, to not contradict \mathcal{X} being a minimal transversal $\{Y\} = \mathcal{X}$. This is again the third statement in the lemma.

(\Leftarrow) The opposite direction is even more straightforward.

Case 1: v is simple

$$\vartheta(\{v\}) = \{r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$$

Case 2: $Y \in \{\emptyset, r_T\}$

$$\vartheta(\{Y\}) = \{Y\} \sqsubseteq \{\emptyset, r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$$

Case 3: $\mathcal{X} = \{Y\}$ or $\mathcal{X} = \vartheta(\{v\})$

$$\vartheta(\{Y\}) = \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$$

$$\vartheta(\{v\}) = \vartheta(\vartheta(\{v\})) = \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$$

□

Taking everything into account, we can enhance the basic transversal approach from previously. This gives our main transversal approach for pXFD discovery as follows:

```

proc transversal_discovery(main)
  Compute  $candRHS_{T'}(v)$ 
  while  $candRHS_{T'}(v) - \{\emptyset, r_T\} \neq \{\}$  do
    Choose  $Y \in \sqsubseteq\text{-max}(candRHS_{T'}(v))$ 
    Compute  $\mathfrak{D}_{T'}^Y(v)$ 
    Compute  $\mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v))$ 
    /* Extract pXFD with  $\{Y\}$  as RHS */
    for all  $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v)) - \{\{Y\}, \vartheta(\{v\})\}$  do
      if  $\mathcal{X} \notin \mathfrak{Tr}(\mathfrak{D}_{T'}^Z(v))$  for any  $Z \in candRHS_{T'}(v)$  with  $Z \sqsupset Y$  then
         $\mathfrak{C}_{T'}(v) := \mathfrak{C}_{T'}(v) \cup \{v : \mathcal{X} \rightarrow \{Y\}\}$ 
      end if
    end for
     $candRHS_{T'}(v) := candRHS_{T'}(v) - \{Y\}$ 
  end while

```

The set $candRHS_{T'}(v)$ of all candidate RHSs is pre-computed. We consider \sqsubseteq -maximal candidate RHS first because checking whether a canonical pXFD $v : \mathcal{X} \rightarrow \{Y\}$ is right-maximal w.r.t. T' requires knowledge about all satisfied canonical pXFD $v : \mathcal{X} \rightarrow \{Z\}$ with $Z \sqsupset Y$ (where Y, Z are both a candidate RHS). Furthermore, we integrate the removal of all trivial pXFDs with a simple set difference operator. Like Mannila and Rähilä it is possible to prevent trivial pXFDs from being generated by removing Y and $n \in \vartheta(\{v\})$ from each v -difference set in $\mathfrak{D}_{T'}^Y(v)$ before computing the transversals. However, with a vision for re-using transversal computations, it is better to remove only $n \in \vartheta(\{v\})$ in the beginning and deal with the other cases at the time of pXFD extraction.

The appropriateness of the transversal approach for computing canonical pXFD-covers is formally proven in the ensuing theorem.

Theorem 3.24. *Let v be a node which is not simple and let Y be an essential v -property of T . Further let*

$$LHS(Y) = \mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v) - \{\{Y\}, \vartheta(\{v\})\}) - \bigcup_{Z \in \text{candRHS}_{T'}(v) \text{ and } Z \sqsupset Y} \mathfrak{Tr}(\mathfrak{D}_{T'}^Z(v)).$$

The following statements are equivalent

- (i) $v : \mathcal{X} \rightarrow \{Y\} \in \mathfrak{C}_{T'}(v)$
- (ii) $Y \in \text{candRHS}_{T'}(v) - \{\emptyset, r_T\}$ and $\mathcal{X} \in LHS(Y)$

Proof. (i) \Leftrightarrow (ii) : Assume (ii) holds. From the definition of $LHS(Y)$ we know \mathcal{X} is a minimal transversal of $\mathfrak{D}_{T'}^Y(v)$. Moreover we know $\mathcal{X} \cup \{Y\}$ is a set of essential v -properties since $\vartheta(\mathcal{X}) \sqsubseteq \mathcal{X}$ by Lemma 1.40 means that $\vartheta(\mathcal{X}) = \mathcal{X}$. In other words $v : \mathcal{X} \rightarrow \{Y\}$ is a canonical pXFD which, by Corollary 3.19, is left-reduced w.r.t. T' . Suppose $v : \mathcal{X} \rightarrow \{Y\}$ is not right-maximal w.r.t. T' . Then it is implied by some canonical pXFD $v : \mathcal{X} \rightarrow \{Z\}$ with $Z \sqsupset Y$ which is left-reduced and right-maximal w.r.t. T' (Theorem 3.7). It follows that \mathcal{X} is a minimal transversal of $\mathfrak{D}_{T'}^Z(v)$, i.e., $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Z(v))$ (Corollary 3.19). Also $Z \in \text{candRHS}_{T'}(v)$ by Lemma 3.21 which contradicts $\mathcal{X} \in LHS(Y)$ (see definition of $LHS(Y)$ above). Therefore $v : \mathcal{X} \rightarrow \{Y\}$ is right-maximal. It remains to show that $v : \mathcal{X} \rightarrow \{Y\}$ is non-trivial. This follows directly from Lemma 3.23.

(i) \Rightarrow (ii) : Assume $v : \mathcal{X} \rightarrow \{Y\} \in \mathfrak{C}_{T'}(v)$. Then the pXFD is canonical, non-trivial, left-reduced and right-maximal w.r.t. T' according to Definition 3.10. In particular, $Y \in \text{candRHS}_{T'}(v) - \{\emptyset, r_T\}$ and $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v) - \{\{Y\}, \vartheta(\{v\})\})$ by Corollary 3.19, Lemma 3.23 and Lemma 3.21. If $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Z(v))$ for some $Z \in \text{candRHS}_{T'}(v)$ where $Z \sqsupset Y$ then $\models_{T'} v : \mathcal{X} \rightarrow \{Z\}$ (Corollary 3.19). But this would mean $v : \mathcal{X} \rightarrow \{Y\}$ is not right-maximal w.r.t. T' , contradicting our assumption. Therefore, $\mathcal{X} \in LHS(Y)$. We have shown that assuming statement (i) is true results in statement (ii) also being true. Thus complete the second part of the proof. \square

The main transversal approach for pXFD discovery is demonstrated for the purchase data tree T'_{purchase} .

Example 3.25. Let us compute the canonical pXFD-cover of the purchase data tree T'_{purchase} . We have the following ten v_{Purchase} -properties as candidate RHSs:

$$\{\text{De, Pr, Di}\}, \text{De, Pr, Di, Da, Ti, Pn, Sa}, v_{\text{Purchase}} \text{ and } v_{\text{Outlet}}$$

with $\{\text{De, Pr, Di}\}, \text{Da, Ti, Pn, Sa}$ and v_{Purchase} being \sqsubseteq -maximal among the candidate RHSs. Example 3.47 and Example 3.66 demonstrate possible approaches for finding respectively the v_{Purchase} -ancestors and v_{Purchase} -subgraphs which are candidate RHSs. Note that we have ruled-out four essential v_{Purchase} -subgraphs and one essential v_{Purchase} -ancestor as potential RHSs.

A summary of the computation of the canonical pXFD-cover of T'_{purchase} using the main transversal approach is provided in Table 3.1. For each candidate RHS Y the table shows:

<i>and. RHS</i>	difference sets modulo <i>and.RHS</i>	minimal transversals	extracted XFDs
{De, Pr, Di}	$\mathfrak{D}_{T'_{purchase}}^{\{De, Pr, Di\}}(v_{Purchase})$ $= \mathfrak{D}_{T'_{purchase}}^{v_{Purchase}}(v_{Purchase})$	{Pn}, {{De, Pr}}, {De, Di}, {De, Sa}, {Di, Ti}, {Di, Pr}, {Sa, Ti}, {Sa, Pr}, {vOutlet, Di}, {vOutlet, Sa}, {vPurchase}	$v_{Purchase} : \{Pn\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{\{De, Pr\}\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{De, Di\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{De, Sa\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{Di, Ti\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{Di, Pr\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{Sa, Ti\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{Sa, Pr\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{vOutlet, Di\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{vOutlet, Sa\} \rightarrow \{\{De, Pr, Di\}\}$ $v_{Purchase} : \{vOutlet, Sa\} \rightarrow \{\{De, Pr, Di\}\}$
De	{Ti, Pn, De, Pr, vOutlet}, {Ti, Pn, De, Pr, Di, Sa, vOutlet}, and {Pn, De, Di, Sa, vPurchase}	{De}, {Pn}, {Di, Pr}, {Di, Ti}, {Sa, Pr}, {Sa, Ti}, {vOutlet, Di}, {vOutlet, Sa}, {vPurchase}	No pXFD extracted.
Pr	{Ti, Pn, De, Pr, vOutlet}, {Ti, Pn, Pr, Di, Sa, vOutlet}, and {Ti, Pn, De, Pr, Di, Sa, vOutlet}	{Pr}, {Ti}, {Pn}, {De, Di}, {De, Sa}, and {vOutlet}	$v_{Purchase} : \{Ti\} \rightarrow \{Pr\}$ $v_{Purchase} : \{vOutlet\} \rightarrow \{Pr\}$
Di	{Pn, Di, {De, Pr}, Sa, vPurchase}, {Ti, Pn, Pr, Di, Sa, vOutlet}, {Ti, Pn, De, Pr, Di, Sa, vOutlet}, and {Pn, De, Di, Sa, vPurchase}	{Pn}, {{De, Pr}}, {Sa}, {Di}, and {vPurchase}	$v_{Purchase} : \{Sa\} \rightarrow \{Di\}$
Da	$\mathfrak{D}_{T'_{purchase}}^{Da}(v_{Purchase}) = \{\}$	{}	$v_{Purchase} : \{\} \rightarrow \{Da\}$
Ti	$\mathfrak{D}_{T'_{purchase}}^{Ti}(v_{Purchase})$ $= \mathfrak{D}_{T'_{purchase}}^{Pr}(v_{Purchase})$	{Pr}, {Ti}, {Pn}, {De, Di}, {De, Sa}, and {vOutlet}	$v_{Purchase} : \{Pn\} \rightarrow \{Ti\}$ $v_{Purchase} : \{Pr\} \rightarrow \{Ti\}$ $v_{Purchase} : \{De, Di\} \rightarrow \{Ti\}$ $v_{Purchase} : \{De, Sa\} \rightarrow \{Ti\}$ $v_{Purchase} : \{vOutlet\} \rightarrow \{Ti\}$
Pn	$\mathfrak{D}_{T'_{purchase}}^{Da}(v_{Purchase})$ $= \mathfrak{D}_{T'_{purchase}}^{\{De, Pr, Di\}}(v_{Purchase})$	{Pn}, {{De, Pr}}, {De, Di}, {De, Sa}, {Di, Ti}, {Di, Pr}, {Sa, Ti}, {Sa, Pr}, {vOutlet, Di}, {vOutlet, Sa}, {vPurchase}	$v_{Purchase} : \{\{De, Pr\}\} \rightarrow \{Pn\}$ $v_{Purchase} : \{De, Di\} \rightarrow \{Pn\}$ $v_{Purchase} : \{De, Sa\} \rightarrow \{Pn\}$ $v_{Purchase} : \{Di, Ti\} \rightarrow \{Pn\}$ $v_{Purchase} : \{Di, Pr\} \rightarrow \{Pn\}$ $v_{Purchase} : \{Sa, Ti\} \rightarrow \{Pn\}$ $v_{Purchase} : \{Sa, Pr\} \rightarrow \{Pn\}$ $v_{Purchase} : \{vOutlet, Di\} \rightarrow \{Pn\}$ $v_{Purchase} : \{vOutlet, Sa\} \rightarrow \{Pn\}$ $v_{Purchase} : \{vOutlet, Sa\} \rightarrow \{Pn\}$
Sa	{Pn, Di, {De, Pr}, Sa, vPurchase} {Ti, Pn, Pr, Di, Sa, vOutlet}, and {Ti, Pn, De, Pr, Di, Sa, vOutlet}	{Sa}, {Pn}, {Di}, {{De, Pr}}, and {vPurchase}	$v_{Purchase} : \{Pn\} \rightarrow \{Sa\}$ $v_{Purchase} : \{Di\} \rightarrow \{Sa\}$ $v_{Purchase} : \{\{De, Pr\}\} \rightarrow \{Sa\}$
vPurchase	$\mathfrak{D}_{T'_{purchase}}^{v_{Purchase}}(v_{Purchase})$ $= \mathfrak{D}_{T'_{purchase}}^{\{De, Pr, Di\}}(v_{Purchase})$	{Pn}, {{De, Pr}}, {De, Di}, {De, Sa}, {Di, Ti}, {Di, Pr}, {Sa, Ti}, {Sa, Pr}, {vOutlet, Di}, {vOutlet, Sa}, {vPurchase}	$v_{Purchase} : \{Pn\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{\{De, Pr\}\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{De, Di\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{De, Sa\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{Di, Ti\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{Di, Pr\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{Sa, Ti\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{Sa, Pr\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{vOutlet, Di\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{vOutlet, Sa\} \rightarrow \{v_{Purchase}\}$ $v_{Purchase} : \{vOutlet, Sa\} \rightarrow \{v_{Purchase}\}$
vOutlet	$\mathfrak{D}_{T'_{purchase}}^{v_{Outlet}}(v_{Purchase})$ $= \mathfrak{D}_{T'_{purchase}}^{Pr}(v_{Purchase})$	{Pr}, {Ti}, {Pn}, {De, Di}, {De, Sa}, and {vOutlet}	$v_{Purchase} : \{Ti\} \rightarrow \{v_{Outlet}\}$ $v_{Purchase} : \{Pr\} \rightarrow \{v_{Outlet}\}$

Table 3.1: Summary of main transversal approach for discovering pXFDs in $T'_{purchase}$.

- the family $\mathfrak{D}_{T'_{purchase}}^Y(v_{Purchase})$ of all $v_{Purchase}$ -difference sets modulo Y , and
- all minimal transversals of $\mathfrak{D}_{T'_{purchase}}^Y(v_{Purchase})$, and
- the pXFDs which can be extracted from the minimal transversals as per Theorem 3.24.

The candidate RHSs which are \sqsubseteq -comparable are grouped together.

Let us consider the candidate RHS $\{\text{De, Pr, Di}\}$. There are eleven minimal transversals of $\mathfrak{D}_{T'_{purchase}}^{\{\text{De, Pr, Di}\}}(v_{Purchase})$ (see Example 3.17). Since no essential v -subgraph can properly contain a $v_{Purchase}$ -unit, every minimal transversal except for $\{\{\text{De, Pr, Di}\}\}$ and $\vartheta(\{v_{Purchase}\}) = \{v_{Purchase}\}$ gives us a pXFD in the canonical pXFD-cover of $T'_{purchase}$:

$$\begin{array}{ll}
v_{Purchase} : \{\text{Pn}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} & v_{Purchase} : \{\text{Di, Pr}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} \\
v_{Purchase} : \{\{\text{De, Pr}\}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} & v_{Purchase} : \{\text{Sa, Ti}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} \\
v_{Purchase} : \{\text{De, Di}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} & v_{Purchase} : \{\text{Sa, Pr}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} \\
v_{Purchase} : \{\text{De, Sa}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} & v_{Purchase} : \{\text{vOutlet, Di}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} \\
v_{Purchase} : \{\text{Di, Ti}\} & \rightarrow \{\{\text{De, Pr, Di}\}\} & v_{Purchase} : \{\text{vOutlet, Sa}\} & \rightarrow \{\{\text{De, Pr, Di}\}\}
\end{array}$$

It is easy to see that

$$\mathfrak{D}_{T'_{purchase}}^{v_{Purchase}}(v_{Purchase}) = \mathfrak{D}_{T'_{purchase}}(v_{Purchase}) = \mathfrak{D}_{T'_{purchase}}^{\{\text{De, Pr, Di}\}}(v_{Purchase}).$$

and $v_{Purchase}$ is also \sqsubseteq -maximal among all essential v -properties. Particularly, node $v_{Purchase}$ is not \sqsubseteq -comparable with subgraph $\{\text{De, Pr, Di}\}$. Therefore, every minimal transversal of $\mathfrak{D}_{T'_{purchase}}^{v_{Purchase}}(v_{Purchase})$ except for $\{v_{Purchase}\}$ gives us a pXFD for the canonical pXFD-cover of $T'_{purchase}$:

$$\begin{array}{ll}
v_{Purchase} : \{\text{Pn}\} & \rightarrow \{v_{Purchase}\} & v_{Purchase} : \{\text{Di, Pr}\} & \rightarrow \{v_{Purchase}\} \\
v_{Purchase} : \{\{\text{De, Pr}\}\} & \rightarrow \{v_{Purchase}\} & v_{Purchase} : \{\text{Sa, Ti}\} & \rightarrow \{v_{Purchase}\} \\
v_{Purchase} : \{\text{De, Di}\} & \rightarrow \{v_{Purchase}\} & v_{Purchase} : \{\text{Sa, Pr}\} & \rightarrow \{v_{Purchase}\} \\
v_{Purchase} : \{\text{De, Sa}\} & \rightarrow \{v_{Purchase}\} & v_{Purchase} : \{\text{vOutlet, Di}\} & \rightarrow \{v_{Purchase}\} \\
v_{Purchase} : \{\text{Di, Ti}\} & \rightarrow \{v_{Purchase}\} & v_{Purchase} : \{\text{vOutlet, Sa}\} & \rightarrow \{v_{Purchase}\}
\end{array}$$

Now all remaining candidate RHSs are \sqsubseteq -maximal and can be considered in any order. Suppose next we consider the walk Di . The family of $v_{Purchase}$ -difference sets modulo Di consists of:

$$\begin{array}{ll}
\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase}) & = \{\text{Pn, Di, \{De, Pr\}, Sa, v_{Purchase}\} \\
\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase}) & = \{\text{Ti, Pn, Pr, Di, Sa, v_{Outlet}\} \\
\mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase}) & = \{\text{Ti, Pn, De, Pr, Di, Sa, v_{Outlet}\} \\
\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase}) & = \{\text{Pn, De, Di, Sa, v_{Purchase}\}.
\end{array}$$

which has minimal transversals:

$$\{\text{Pn}\}, \{\{\text{De, Pr}\}\}, \{\text{Sa}\}, \{\text{Di}\} \text{ and } \{v_{Purchase}\}.$$

That is to say we can extract up to five pXFD. However, minimal transversals $\{\text{Di}\}$ and $\{v_{Purchase}\}$ yield trivial pXFDs while $\{\text{Pn}\}$ and $\{\{\text{De, Pr}\}\}$ are minimal transversals of the $v_{Purchase}$ -difference sets modulo $\{\text{De, Pr, Di}\}$ where $\{\text{De, Pr, Di}\} \sqsupset \text{Di}$. Thus, we only one new pXFD $v_{Purchase} : \{\text{Sa}\} \rightarrow \{\text{Di}\}$ can be added to the canonical pXFD-cover of $T'_{purchase}$. \square

3.2 Transversal Approach: In Details

The applicability of the transversal approach hinges on our ability to efficiently solve several sub-problems:

- How to find all minimal transversals for a given family of v -difference sets?
- How to find the input v -difference sets?
- How to find all candidate RHSs?

This section aims to suggest possible answers to these related problems.

3.2.1 Finding Minimal Transversals

The problem of finding all minimal transversals is similar to the famous graph theoretic problem of *Transversal Hypergraph Generation* which aims to enumerate all minimal hitting sets of a given hypergraph. Transversal Hypergraph Generation relates to many other interesting problems [22, 23] and its importance earns it much attention from the research community.

A *hypergraph* $\mathcal{H} = (V, E)$ is a finite collection E of sets over a finite set V . Members of E are called *hyperedges*. A set $\tau \subseteq V$ which intersects with every hyperedge of \mathcal{H} is called a *hitting set* of \mathcal{H} , i.e., $\tau \cap e_i \neq \{\}$ for all $e_i \in E$. A *minimal hitting set* τ of \mathcal{H} is a hitting set of \mathcal{H} such that no proper subset τ' of τ is a hitting set of \mathcal{H} . A *transversal hypergraph* \mathcal{G} of \mathcal{H} is the hypergraph $\mathcal{G} = (V, E')$ where E' is the collection of all minimal hitting set of \mathcal{H} . For more details we refer to [11, 31, 50].

For the theoretic worst-case analysis, the Transversal Hypergraph Generation problem is said to be *efficiently solvable* if there is an *output-polynomial* algorithm (i.e., one that run in time bounded polynomially in the size of both the input and output). After all, the number of minimal hitting sets for a given hypergraph may already be exponential in the size of the hypergraph. It is still unknown whether an output-polynomial algorithm for Transversal Hypergraph Generation exists [36, 37]. So far, the only proven non-trivial lower bound of any algorithm is that of *not being output-polynomial* - for the sequential method of Berge by Takata [85] and for three other related algorithms by Hagen [37].

A collection $\mathfrak{D} \subseteq \mathfrak{D}_{T'}(v)$ of v -difference sets over T -compatible data tree T' induces the hypergraph $\mathcal{H}_{\mathfrak{D}} = (\mathbf{E}_T^{\mathfrak{S}}(v) \cup \mathbf{E}_T^{\mathfrak{A}}(v), \mathfrak{D})$. Although our notion of transversals coincides with that of hitting sets, the idea of a minimal transversal differs from that of minimal hitting set. This is because the nodes in $\mathcal{H}_{\mathfrak{D}}$ (i.e., essential v -properties) are not necessarily independent from one another. In order to determine all left-reduced pXFDs, we need to consider minimality of transversals with respect to p-subsumption rather than just set containment. The set of minimal transversals is precisely the set of minimal hitting sets whenever all essential v -properties are pairwise incomparable with respect to p-subsumption (i.e., discovering HL-XFD for a schema tree with only singleton v -units). More generally, the relationship between minimal transversals and minimal hitting sets is as follows:

Minimal transversals of \mathfrak{D} of v -difference sets are \sqsubseteq -minimal minimal hitting sets of hypergraph $\mathcal{H}_{\mathfrak{D}}$.

As a result, any algorithm for enumerating minimal hitting sets (e.g., [6, 11, 20, 32, 50, 100]) can also be applied for finding minimal transversals. But sometimes computing minimal hitting sets in order to compute minimal transversals can be expensive. As the following example demonstrates, the number of minimal hitting sets may be much larger than that of minimal transversals.

Example 3.26. Consider an XML schema tree with a single v -unit consisting of three v -walks A, B, C and, a T -compatible data tree T' such that

$$\mathfrak{D}_{T'}^{ABC}(v) = \{\{A, C, AB, AC, BC, ABC\}, \{B, AB, BC, ABC\}\}$$

There are six minimal hitting sets: $\{AB\}, \{BC\}, \{ABC\}, \{A, B\}, \{B, C\}, \{AC, B\}$ but only two minimal transversals: $\{A, B\}, \{B, C\}$. We observe $\{AB\}$ and $\{ABC\}$ p-subsume $\{A, B\}$ while both $\{BC\}$ and $\{AC, B\}$ p-subsume $\{B, C\}$. \square

Instead, we present an algorithm for computing all minimal transversals directly - our algorithm is especially similar to the sequential method proposed by Berge [11].

The sequential method takes as input a *simple hypergraph*, that is, a hypergraph in which there are no two hyperedges e, e' with $e \subseteq e'$. This is without loss of generality because every hypergraph can be transformed into a simple hypergraph having the same transversal hypergraph in time polynomial in the number of hyperedges. The sequential method is grounded in two observations:

- the minimal hitting sets for a hypergraph with exactly one hyperedge are simply the singleton subsets of the hyperedge
- the minimal hitting sets for the union $\mathcal{H} \cup \mathcal{G} = (V, E \cup E')$ of two hypergraphs $\mathcal{H} = (V, E)$ and $\mathcal{G} = (V, E')$, is the \sqsubseteq -minimal set in the union of every minimal hitting sets of \mathcal{H} with every minimal hitting sets of \mathcal{G} .

And so, to find all minimal hitting sets for a given hypergraph $\mathcal{H} = (V, E)$, then we start by determining the minimal hitting sets for hypergraph $\mathcal{H}_1 = (V, E_1)$ where $E_1 = \{e_1\}$ consists of one of the hyperedges of \mathcal{H} . This is straightforward from the first observation. For each of the remaining hyperedge e_i we determine, as per the second observation, the minimal hitting sets for hypergraph $\mathcal{H}_i = (V, E_{i-1}) \cup (V, \{e_i\})$ where $E_{i-1} \subset E$ contains all hyperedges considered thus far.

Likewise, we propose to compute the minimal transversals for a family \mathfrak{D} of v -difference sets sequentially by considering the v -difference sets one-by-one. Such a sequential approach facilitates re-use of minimal transversal computations, as we will see later on. Particularly, if we have two essential v -properties Y, Z such that $Y \sqsubseteq Z$ then $\mathfrak{D}_{T'}^Z(v) \supseteq \mathfrak{D}_{T'}^Y(v)$. In this case we can find all minimal transversals for $\mathfrak{D}_{T'}^Y(v)$ in the process of determining all minimal transversals for $\mathfrak{D}_{T'}^Z(v)$. More about this a little later, when we consider generation of all candidate RHSs.

Consider two sets $\mathcal{D}, \mathcal{D}'$ of v -properties such that $\mathcal{D} \subseteq \mathcal{D}'$. Any transversal for $\{\mathcal{D}\}$ is of course a transversal for $\{\mathcal{D}'\}$. On the other hand, no transversal of $\{\mathcal{D}'\}$ which is not a transversal of $\{\mathcal{D}\}$ can be a minimal transversal for $\{\mathcal{D}, \mathcal{D}'\}$. Let \mathfrak{D} be a family of v -difference sets, then

$$\subseteq\text{-min}(\mathfrak{D}) = \{\mathcal{D} \mid \text{there exists no } \mathcal{D}' \subset \mathcal{D} \text{ such that } \mathcal{D}' \in \mathfrak{D}\}$$

is the family of all \subseteq -minimal members of \mathfrak{D} which can be computed in time polynomial in the number of sets belonging to \mathfrak{D} .

Lemma 3.27. *Let \mathfrak{D} be a family of v -difference sets. Then $\mathfrak{Tr}(\mathfrak{D}) = \mathfrak{Tr}(\subseteq\text{-min}(\mathfrak{D}))$.*

Proof. Follows from $\subseteq\text{-min}(\mathfrak{D}) \subseteq \mathfrak{D}$ and every set $\mathcal{D} \in \mathfrak{D} - \subseteq\text{-min}(\mathfrak{D})$ is superset of some $\mathcal{D}' \in \subseteq\text{-min}(\mathfrak{D})$. \square

Actually, it is possible to show that $\mathfrak{Tr}(\mathfrak{D}) = \mathfrak{Tr}(\subseteq\text{-min}(\mathfrak{D}))$. However comparing two v -difference sets on the basis of set containment is simpler than comparing them on the basis of p -subsumption. Instead we can observe that every minimal transversal of \mathfrak{D} contain only \subseteq -minimal elements from each v -difference set in \mathfrak{D} . That is, $\mathfrak{Tr}(\mathfrak{D}) = \mathfrak{Tr}(\text{rep}(\mathfrak{D}))$ where $\text{rep}(\mathfrak{D}) = \{\subseteq\text{-min}(\mathcal{D}) \mid \mathcal{D} \in \mathfrak{D}\}$ is the family of representative subsets of v -difference sets in \mathfrak{D} . And so, and for computing the minimal transversals for \mathfrak{D} , we take as input the family $*\mathfrak{D} = \subseteq\text{-min}(\text{rep}(\mathfrak{D}))$.

Corollary 3.28. *Let \mathfrak{D} be a family of v -difference sets. Then $\mathfrak{Tr}(\mathfrak{D}) = \mathfrak{Tr}(*\mathfrak{D})$.*

Next we will show how minimal transversals for a family $\mathfrak{D} \cup \{\mathcal{D}\}$ can be deduced from the minimal transversals for \mathfrak{D} and the minimal transversals for $\{\mathcal{D}\}$. The following melding operator is used for the deduction.

Definition 3.29. Let $\mathfrak{D}, \mathfrak{D}'$ be two families of sets of v -properties. We call

$$\mathfrak{D} \vee \mathfrak{D}' = \{\mathcal{D} \cup \mathcal{D}' \mid \mathcal{D} \in \mathfrak{D} \text{ and } \mathcal{D}' \in \mathfrak{D}'\}$$

the *melding* of \mathfrak{D} and \mathfrak{D}' . \square

The melding of two core sets may not result in a core set. However, we have previously observed that $\vartheta(\mathcal{X}) \sqsubseteq \mathcal{X}$ for every set \mathcal{X} of v -properties. Moreover, the refinement of sets resulting from the melding may still be comparable with respect to p -subsumption. Therefore to find minimal transversals, we should first refine members of the melding and then verify whether or not the transversal is indeed minimal with respect to p -subsumption.

Proposition 3.30. *Let $\mathfrak{D} \cup \{\mathcal{D}\}$ be a family of v -difference sets with $\mathcal{D} \notin \mathfrak{D}$. Then*

$$\mathfrak{Tr}(\mathfrak{D} \cup \{\mathcal{D}\}) = \subseteq\text{-min}(\{\vartheta(\mathcal{X}) \mid \mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}) \vee \mathfrak{Tr}(\{\mathcal{D}\})\})$$

Proof. (\supseteq) Suppose $\mathcal{T} \in \sqsubseteq\text{-min}(\{\vartheta(\mathcal{X}) \mid \mathcal{X} \in \mathfrak{Tr}(\mathcal{D}) \vee \mathfrak{Tr}(\{\mathcal{D}\})\})$. In particular there exist $\mathcal{S}_i \in \mathfrak{Tr}(\mathcal{D})$ and $\mathcal{S}'_i \in \mathfrak{Tr}(\{\mathcal{D}\})$ such that

$$\mathcal{T} = \vartheta(\mathcal{S}_i \cup \mathcal{S}'_i)$$

Note that $\mathcal{S}_i \cup \mathcal{S}'_i$ is a transversal for $\mathcal{D} \cup \{\mathcal{D}\}$. The refinement ϑ removes only non-maximal elements from $\mathcal{S}_i \cup \mathcal{S}'_i$. But v -difference sets are upward-closed and so \mathcal{T} is a transversal for $\mathcal{D} \cup \{\mathcal{D}\}$. For any $\mathcal{T}' \sqsubset \mathcal{T}$, then $\mathcal{T} \not\sqsubseteq \mathcal{T}'$. Specifically, there exist some $X \in \mathcal{T}$ which is not p -subsumed by any members of \mathcal{T}' and so, $\mathcal{T}' = \mathcal{T} - \{X\} \cup \mathcal{X}'$ where $\mathcal{X}' \sqsubset \{X\}$. Because \mathcal{S}'_i is a minimal transversal for $\{\mathcal{D}\}$, if $X \in \mathcal{S}'_i$ then \mathcal{T}' is not a transversal for $\{\mathcal{D}\}$. Likewise, if $X \in \mathcal{S}_i$ then \mathcal{T}' is not a transversal for \mathcal{D} . In either case \mathcal{T}' is not transversal for $\mathcal{D} \cup \{\mathcal{D}\}$ and so \mathcal{T} is a minimal transversal of $\mathcal{D} \cup \{\mathcal{D}\}$, i.e., $\mathcal{T} \in \mathfrak{Tr}(\mathcal{D} \cup \{\mathcal{D}\})$.

(\subseteq) Let $\mathcal{T} \in \mathfrak{Tr}(\mathcal{D} \cup \{\mathcal{D}\})$. Then \mathcal{T} must be a transversal for \mathcal{D} and $\{\mathcal{D}\}$. This means that there are $\mathcal{S}_i \in \mathfrak{Tr}(\mathcal{D}), \mathcal{S}'_i \in \mathfrak{Tr}(\{\mathcal{D}\})$ such that $\vartheta(\mathcal{S}_i \cup \mathcal{S}'_i) \sqsubseteq \mathcal{S}_i \cup \mathcal{S}'_i \sqsubseteq \mathcal{T}$. If $\vartheta(\mathcal{S}_i \cup \mathcal{S}'_i) \sqsubset \mathcal{T}$ then there is a contradiction with \mathcal{T} being a minimal transversal of $\mathcal{D} \cup \{\mathcal{D}\}$. Therefore suppose $\vartheta(\mathcal{S}_i \cup \mathcal{S}'_i) \sqsupseteq \mathcal{T}$ which gives $\vartheta(\mathcal{S}_i \cup \mathcal{S}'_i) = \mathcal{T}$. If $\mathcal{T}' \sqsubset \mathcal{T}$ then $\mathcal{T}' \notin \{\vartheta(\mathcal{X}) \mid \mathcal{X} \in \mathfrak{Tr}(\mathcal{D}) \vee \mathfrak{Tr}(\{\mathcal{D}\})\}$, otherwise we again contradict \mathcal{T} being a minimal transversal of $\mathcal{D} \cup \{\mathcal{D}\}$. Hence $\mathcal{T} \in \sqsubseteq\text{-min}(\{\vartheta(\mathcal{X}) \mid \mathcal{X} \in \mathfrak{Tr}(\mathcal{D}) \vee \mathfrak{Tr}(\{\mathcal{D}\})\})$. \square

The next example demonstrates an application of the previous proposition.

Example 3.31. Consider an XML schema tree with three v -units:

$$U_1 = AB, \quad U_2 = CD, \quad U_3 = E$$

Suppose there are two v -difference sets $\mathcal{D}_1 = \{A, B, C, AB, CD\}$ and $\mathcal{D}_2 = \{AB, CD, E\}$. The corresponding families of minimal transversals are $\mathfrak{Tr}(\{\mathcal{D}_1\}) = \{\{A\}, \{B\}, \{C\}\}$ and $\mathfrak{Tr}(\{\mathcal{D}_2\}) = \{\{AB\}, \{CD\}, \{E\}\}$. The melding $\mathfrak{Tr}(\{\mathcal{D}_1\}) \vee \mathfrak{Tr}(\{\mathcal{D}_2\})$ can be computed as follows:

- $\{A\} \cup \{AB\} = \{A, AB\}$
- $\{B\} \cup \{AB\} = \{B, AB\}$
- $\{C\} \cup \{AB\} = \{C, AB\}$
- $\{A\} \cup \{AB\} = \{A, CD\}$
- $\{B\} \cup \{AB\} = \{B, CD\}$
- $\{C\} \cup \{AB\} = \{C, CD\}$
- $\{A\} \cup \{AB\} = \{A, E\}$
- $\{B\} \cup \{AB\} = \{B, E\}$
- $\{C\} \cup \{AB\} = \{C, E\}$

To compute the minimal transversals for the family $\{\mathcal{D}_1, \mathcal{D}_2\}$ we then refine the members of the melding above, and eliminate those elements which are not \sqsubseteq -minimal. Refinement results in:

- both $\{A, AB\}$ and $\{B, AB\}$ refine to $\{AB\}$,
- $\{C, CD\}$ refines to $\{CD\}$, and
- the remaining sets $\{C, AB\}, \{A, CD\}, \{B, CD\}, \{A, E\}, \{B, E\}, \{C, E\}$ are unchanged after refinement.

Moreover, we have $\{AB\} \sqsubseteq \{C, AB\}$, $\{CD\} \sqsubseteq \{A, CD\}$ and $\{CD\} \sqsubseteq \{B, CD\}$. Therefore, the minimal transversals for $\{\mathcal{D}_1, \mathcal{D}_2\}$ are

$$\{AB\}, \{CD\}, \{A, E\}, \{B, E\}, \{C, E\}$$

□

The next lemma and subsequent corollary establish important foundations for our proposed inductive computation of minimal transversals.

Lemma 3.32. *Let \mathcal{D} be a v -difference set. Then $\mathfrak{Tr}(\{\mathcal{D}\}) = \{\{X\} \mid X \in \sqsubseteq_{-\min}(\mathcal{D})\}$.*

Proof. Immediate from definition of minimal transversals. □

Corollary 3.33. *Let $\mathfrak{D} \cup \{\mathcal{D}\}$ be a family of v -difference sets with $\mathcal{D} \notin \mathfrak{D}$. Then*

$$\mathfrak{Tr}(\mathfrak{D} \cup \{\mathcal{D}\}) = \sqsubseteq_{-\min}(\mathfrak{Tr}(\mathfrak{D}) \vee \{\{X\} \mid X \in \sqsubseteq_{-\min}(\mathcal{D})\})$$

□

Suppose we want to generate $\mathfrak{Tr}(\mathfrak{D})$ with $\mathfrak{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_m\}$. After each $i = 1, \dots, m$ we obtain the minimal transversals for a partial family \mathfrak{D}_i which contains v -difference sets $\mathcal{D}_1, \dots, \mathcal{D}_i$. In the beginning, the minimal transversals of $\{\mathcal{D}_1\}$ are exactly the $\sqsubseteq_{-\min}$ -minimal members of \mathcal{D}_1 . For each subsequent $2 \leq i \leq m$ we compute the refinement of every set of v -properties belonging to the melding $\mathfrak{Tr}(\mathfrak{D}_{i-1}) \vee \mathfrak{Tr}(\{\mathcal{D}_i\})$ and discard any transversal which is not $\sqsubseteq_{-\min}$ -minimal.

Similar to the case with hitting sets, we face the problem of having to compute all minimal transversals of the partial family \mathfrak{D}_i (for $i = 1, \dots, m - 1$) and from those only selecting the $\sqsubseteq_{-\min}$ -minimal ones. There are possibly exponentially many such intermediate transversals since the melding results in a sort of Cartesian product. With the following pair of lemmas, we try to reduce the number of essential v -properties in $\mathfrak{Tr}(\mathfrak{D}_{i-1})$ and in $\sqsubseteq_{-\min}(\{\mathcal{D}_i\})$ which participate in the melding, in particular:

- Lemma 3.34 identifies minimal transversals of \mathfrak{D} which are minimal transversals of $\mathfrak{D} \cup \{\mathcal{D}\}$.
- Lemma 3.35 identifies minimal transversals of $\{\mathcal{D}\}$ which are minimal transversals of $\mathfrak{D} \cup \{\mathcal{D}\}$.

The melding operation as per Corollary 3.33 will only be applied for the smallest subsets of $\mathfrak{Tr}(\mathfrak{D})$ and $\mathfrak{Tr}(\{\mathcal{D}\})$ whose members are not already identified as being minimal transversals of $\mathfrak{D} \cup \{\mathcal{D}\}$ by Lemma 3.34 and Lemma 3.35. A similar heuristic has also been applied for FD discovery [19].

Lemma 3.34. *Let $\mathfrak{D} \cup \{\mathcal{D}\}$ be a family of v -difference sets with $\mathcal{D} \notin \mathfrak{D}$. Further Let \mathcal{X} be a transversal of \mathfrak{D} . The existence of some $X \in \sqsubseteq_{-\min}(\mathcal{D})$ such that $\{X\} \sqsubseteq \mathcal{X}$ implies that \mathcal{X} is a minimal transversal for $\mathfrak{D} \cup \{\mathcal{D}\}$.*

Algorithm 3.4 find_minimalTransversals**Input:** $*\mathcal{D}$ /* where $*\mathcal{D} := \sqsubseteq_{-\min}(\{\sqsubseteq_{-\min}(\mathcal{D}) \mid \mathcal{D} \in \mathcal{D}\})$ with $\mathcal{D} \subseteq \mathcal{D}_{T'}(v)$ */**Output:** $\mathfrak{Tr}(\mathcal{D})$

```

1. currentTrans := {}
2. if  $*\mathcal{D} = \{\}$  then
3.   return {}
4. end if
5.  $\sqsubseteq_{-\min}(\mathcal{D}) :=$  Remove a set from  $*\mathcal{D}$ 
6. currentTrans :=  $\{\{X\} \mid X \in \sqsubseteq_{-\min}(\mathcal{D})\}$  // Lemma 3.32
7. if  $*\mathcal{D} \neq \{\}$  then
8.   currentTrans := extend_minimalTransversals(currentTrans,  $*\mathcal{D}$ )
9. end if
10. return currentTrans

```

Proof. $\{X\} \sqsubseteq \mathcal{X}$ means that there is some $X' \in \mathcal{X}$ such that $X \sqsubseteq X'$. Since \mathcal{D} is \sqsubseteq -upward-closed and $X \in \sqsubseteq_{-\min}(\mathcal{D})$, it follows that $X' \in \mathcal{D}$. Therefore $\mathcal{X} \cap \mathcal{D} \neq \{\}$ and \mathcal{X} is a transversal for $\{\mathcal{D}\}$. Since \mathcal{X} is a minimal transversal of \mathcal{D} , \mathcal{X} is a transversal for $\mathcal{D} \cup \{\mathcal{D}\}$ and no set $\mathcal{X}' \sqsubset \mathcal{X}$ is a transversal of $\mathcal{D} \cup \{\mathcal{D}\}$. Hence \mathcal{X} is a minimal transversal for $\mathcal{D} \cup \{\mathcal{D}\}$. \square

Lemma 3.35. *Let $\mathcal{D} \cup \{\mathcal{D}\}$ be a family of v -difference sets with $\mathcal{D} \notin \mathcal{D}$. Further let \mathcal{X} be a minimal transversal of \mathcal{D} . The existence of some $X \in \sqsubseteq_{-\min}(\mathcal{D})$ such that $\mathcal{X} \sqsubseteq \{X\}$ implies that $\{X\}$ is a minimal transversal of $\mathcal{D} \cup \{\mathcal{D}\}$.*

Proof. $X \in \sqsubseteq_{-\min}(\mathcal{D})$ means that X is an essential v -property and a minimal transversal of $\{\mathcal{D}\}$. Moreover with $\mathcal{X} \sqsubseteq \{X\}$ being a minimal transversal of \mathcal{D} we find that $\vartheta(\{\mathcal{X}\} \vee \{X\}) = \{X\}$ is a transversal of $\mathcal{D} \cup \{\mathcal{D}\}$. Since $X \in \sqsubseteq_{-\min}(\mathcal{D})$ no element $X' \sqsubset X$ belongs to \mathcal{D} . Every $\mathcal{X}' \sqsubset \{X\}$ contains only elements which are strictly p -subsumed by X . Therefore no $\mathcal{X}' \sqsubset \{X\}$ is a transversal of $\{\mathcal{D}\}$. That is, no $\mathcal{X}' \sqsubset \{X\}$ is a transversal of $\mathcal{D} \cup \{\mathcal{D}\}$. By definition, $\{X\}$ is then a minimal transversal of $\mathcal{D} \cup \{\mathcal{D}\}$. \square

Algorithm 3.4 summarises our sequential approach for computing all minimal transversals of some given family \mathcal{D} of v -difference sets.

Theorem 3.36. *Algorithm 3.4 always terminates and accurately computes all minimal transversals of \mathcal{D} .*

Proof. The number of v -difference sets in a given data tree T' is finite which means $*\mathcal{D}$ is finite. The algorithm clearly terminates if $*\mathcal{D} = \{\}$ (line 3) or given that $*\mathcal{D}$ contains exactly one core set. The last case is that $*\mathcal{D}$ contains two or more core sets and we make a call to Algorithm 3.5.

The **while** loop in Algorithm 3.5 applies until $*\mathcal{D}$ is empty, and after each iteration through the **while** loop we reduce the size of $*\mathcal{D}$ by one element. Thus Algorithm 3.5 terminates and consequently so does Algorithm 3.4.

Algorithm 3.5 `extend_minimalTransversals`**Input:** $\mathfrak{T}(\mathcal{D}')$ and $*\mathcal{D} \neq \{\}$ /* where $*\mathcal{D} := \sqsubseteq_{-\min}(\{\sqsubseteq_{-\min}(\mathcal{D}) \mid \mathcal{D} \in \mathcal{D}\})$ and $\mathcal{D}' \cap \mathcal{D} = \{\}$ */**Output:** $\mathfrak{T}(\mathcal{D}' \cup \mathcal{D})$

1. $currentTrans := \mathfrak{T}(\mathcal{D}')$
2. **while** $*\mathcal{D} \neq \{\}$ **do**
3. $\sqsubseteq_{-\min}(\mathcal{D}) :=$ Remove a set from $*\mathcal{D}$
4. $extn := \{\{X\} \mid X \in \sqsubseteq_{-\min}(\mathcal{D})\}$ // Lemma 3.32
5. $newTrans := \{\mathcal{T} \in currentTrans \mid \exists X \in \sqsubseteq_{-\min}(\mathcal{D}). \{X\} \sqsubseteq \mathcal{T}\}$ // Lemma 3.34
6. $baseTrans := currentTrans - newTrans$
7. $not_extn := \{\mathcal{X} \in extn \mid \exists \mathcal{T} \in currentTrans. \mathcal{T} \sqsubseteq \mathcal{X}\}$ // Lemma 3.35
8. $newTrans := newTrans \cup not_extn$
9. $extn := extn - not_extn$
10. /* Proposition 3.30 */
11. **if** $baseTrans \neq \{\}$ and $extn \neq \{\}$ **then**
12. $currentTrans := \sqsubseteq_{-\min}(newTrans \cup \{\vartheta(\mathcal{T}) \mid \mathcal{T} \in baseTrans \vee extn\})$
13. **end if**
14. **end while**
15. **return** $currentTrans$

That we obtain all minimal transversals follows from Corollary 3.33 together with, Lemma 3.34 and Lemma 3.35. Recall from the definition of transversals that only the empty set is a minimal transversal of $\mathcal{D} = \{\}$. This appears as a special case in the algorithm. \square

Next, we provide an example to illustrate how an inductive minimal transversal computation proceeds.

Example 3.37. Recall the five $v_{Purchase}$ -difference sets of $T'_{purchase}$:

$$\begin{aligned}
\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}) &= \{\text{Ti}, \text{Pn}, \text{De}, \text{Pr}, v_{Outlet}\} \\
\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase}) &= \{\text{Ti}, \text{Pn}, \text{Pr}, \text{Di}, \text{Sa}, v_{Outlet}\} \\
\mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase}) &= \{\text{Ti}, \text{Pn}, \text{De}, \text{Pr}, \text{Di}, \text{Sa}, v_{Outlet}\} \\
\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase}) &= \{\text{Pn}, \text{Di}, \{\text{De}, \text{Pr}\}, \text{Sa}, v_{Purchase}\} \\
\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase}) &= \{\text{Pn}, \text{De}, \text{Di}, \text{Sa}, v_{Purchase}\}
\end{aligned}$$

Let us inductively compute the minimal transversals for the family of all $v_{Purchase}$ -difference sets of $T'_{purchase}$. The computation is summarised in Figure 3.3. We will consider the $v_{Purchase}$ -difference sets in the order they are listed above.

To start with,

$$\begin{aligned}
\mathfrak{T}(\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase})\}) &= \{\{X\} \mid X \in \sqsubseteq_{-\min}(\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}))\} \\
&= \{\{\text{Ti}\}, \{\text{Pn}\}, \{\text{De}\}, \{\text{Pr}\}, \{v_{Outlet}\}\}
\end{aligned}$$

Next consider $\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})$. Firstly, we look for those minimal transversals for $\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase})\}$ which p-subsumes some singleton set of element of $\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})$. We find four: $\{\text{Ti}\}$, $\{\text{Pn}\}$, $\{\text{Pr}\}$ and $\{v_{Outlet}\}$. And so, $extn = \{\{\text{Di}\}, \{\text{Sa}\}\}$. Secondly, we try to find those singleton set of element of $\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})$ which p-subsumes some minimal transversal of $\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase})\}$. Because all elements of $\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})$ are v -walks or v -ancestor, we find exactly those from the first case. This gives $baseTrans = \{\{\text{De}\}\}$. It remains to perform the melding,

$$\{\{\text{De}\}\} \vee \{\{\text{Di}\}, \{\text{Sa}\}\} = \{\{\text{De, Di}\}, \{\text{De, Sa}\}\}$$

All transversals that we found are pairwise not \sqsubseteq -comparable, and so

$$\mathfrak{Tr}(\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})\}) = \left\{ \begin{array}{l} \{\text{Ti}\}, \{\text{Pn}\}, \{\text{Pr}\}, \{v_{Outlet}\}, \\ \{\text{De, Di}\}, \{\text{De, Sa}\} \end{array} \right\}$$

Next we consider $\mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase})$. We find that all minimal transversals from the previous step p-subsumes some singleton set of element of $\mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase})$ and so,

$$\begin{aligned} \mathfrak{Tr}(\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase}), \mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase})\}) \\ = \mathfrak{Tr}(\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})\}) \end{aligned}$$

Next we consider $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})$. There are three minimal transversals from the previous step which p-subsumes some singleton set of element of $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})$, these are: $\{\text{Pn}\}$, $\{\text{De, Di}\}$, and $\{\text{De, Sa}\}$. Therefore $extn = \{\{\text{Ti}\}, \{\text{Pr}\}, \{v_{Outlet}\}\}$. We find that three singleton sets of element of $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})$ p-subsumes some minimal transversals from the previous step: $\{\text{Pn}\}$, $\{\{\text{De, Pr}\}\}$ and $\{v_{Purchase}\}$. Therefore $baseTrans = \{\{\text{Di}\}, \{\text{Sa}\}\}$. It remains to perform the melding:

$$\{\{\text{Ti}\}, \{\text{Pr}\}, \{v_{Outlet}\}\} \vee \{\{\text{Di}\}, \{\text{Sa}\}\} = \left\{ \begin{array}{l} \{\text{Ti, Di}\}, \{\text{Ti, Sa}\}, \\ \{\text{Pr, Di}\}, \{\text{Pr, Sa}\}, \\ \{v_{Outlet, Di}\}, \{v_{Outlet, Sa}\} \end{array} \right\}$$

Again all transversals that we found are not pairwise \sqsubseteq -comparable and so

$$\begin{aligned} \mathfrak{Tr}(\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase}), \mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})\}) \\ = \left\{ \begin{array}{l} \{\text{Pn}\}, \{\text{De, Di}\}, \{\text{De, Sa}\}, \{\{\text{De, Pr}\}\}, \{v_{Purchase}\}, \{\text{Ti, Di}\}, \{\text{Ti, Sa}\}, \\ \{\text{Pr, Di}\}, \{\text{Pr, Sa}\}, \{v_{Outlet, Di}\}, \{v_{Outlet, Sa}\} \end{array} \right\} \\ = \mathfrak{Tr}(\mathfrak{D}_{T'_{purchase}}(v_{Purchase}) - \{\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase})\}) \end{aligned}$$

Finally when we consider $\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase})$ we find that all minimal transversals for $\mathfrak{D}_{T'_{purchase}}(v_{Purchase}) - \{\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase})\}$ p-subsumes some singleton set of element of $\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase})$ and so,

$\mathfrak{D}_{\mathcal{P}}(v_{Purchase})$				
$C_{v_{Outlet}}$			$C_{v_{Purchase}}$	
$\mathcal{P} = \{i_5, i_{31}\}$	$\mathcal{P} = \{i_5, i_{44}\}$	$\mathcal{P} = \{i_{18}, i_{31}\}$	$\mathcal{P} = \{i_5, i_{18}\}$	$\mathcal{P} = \{i_{31}, i_{44}\}$
Ti	.	.	Ti, Di Ti, Sa	.
Pn
De	De, Di De, Sa	.	.	.
Pr	.	.	{De, Pr} Pr, Di Pr, Sa	.
v_{Outlet}	.	.	v_{Outlet} , Di v_{Outlet} , Sa $v_{Purchase}$.

 Figure 3.3: Minimal transversals computation for $\mathfrak{D}_{T'_{purchase}}(v_{Purchase})$

$$\begin{aligned}
 & \mathfrak{Tr}(\mathfrak{D}_{T'_{purchase}}(v_{Purchase})) \\
 &= \mathfrak{Tr}(\{\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase}), \mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase}), \mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})\}) \\
 &= \left\{ \begin{array}{l} \{\text{Pn}\}, \{\text{De, Di}\}, \{\text{De, Sa}\}, \{\{\text{De, Pr}\}\}, \{v_{Purchase}\}, \{\text{Ti, Di}\}, \{\text{Ti, Sa}\}, \\ \{\text{Pr, Di}\}, \{\text{Pr, Sa}\}, \{v_{Outlet}, \text{Di}\}, \{v_{Outlet}, \text{Sa}\} \end{array} \right\}
 \end{aligned}$$

□

We have split the sequential minimal transversal computation into two parts such that:

- Algorithm 3.4 finds all minimal transversals of a given family \mathfrak{D} from scratch, and
- Algorithm 3.5 finds the minimal transversals for \mathfrak{D} starting from the minimal transversals for some subfamily $\mathfrak{D}' \subset \mathfrak{D}$.

As we will see, in the discussion of finding candidate RHSs, this separation is geared towards re-use of minimal transversals computation when considering two or more related families of v -difference sets.

Our heuristic requires that the families of v -difference sets are comparable with respect to set containment. In particular, for two essential v -properties Y, Z , if

$Y \sqsubset Z$ then $\mathcal{D}_{T'}^Z(v) \subset \mathcal{D}_{T'}^Y(v)$. If we want to find the minimal transversals for both families of v -difference sets, then it is worthwhile to apply the sequential minimal transversal computation: we can first compute the minimal transversals of $\mathcal{D}_{T'}^Z(v)$ and then use this as a basis for finding the minimal transversals of $\mathcal{D}_{T'}^Y(v)$. Clearly, $\mathfrak{Tr}(\mathcal{D}_{T'}^Y(v)) = \mathfrak{Tr}(\mathcal{D}_{T'}^Z(v)) \vee \mathfrak{Tr}(\mathcal{D}_{T'}^Y(v) - \mathcal{D}_{T'}^Z(v))$. In other words, we compute the minimal transversals for $\mathcal{D}_{T'}^Y(v)$ starting from the minimal transversals for $\mathcal{D}_{T'}^Z(v)$ and consider additionally those v -difference sets in $\mathcal{D}_{T'}^Y(v) - \mathcal{D}_{T'}^Z(v)$. Thus, the effort of computing minimal transversals for both families is reduced. Here we are saved from having to consider the v -difference sets in $\mathcal{D}_{T'}^Z(v)$ twice.

More generally speaking, we benefit from this heuristic when we have a long linear chain of candidate RHSs, and even more so when many of the v -difference sets contain \sqsubseteq -smaller candidate RHSs from the chain. Therefore when computing the canonical pXFD-cover, we should consider the candidate RHSs chain-by-chain and consider each chain starting from the \sqsubseteq -minimal candidate RHSs. We discuss this heuristics in more details in Section 3.2.3 where we examine the problem of identifying all candidate RHSs.

3.2.2 Finding Difference Sets from Dual Agree Sets

For computing minimal transversals using the sequential method, we are only interested in the \sqsubseteq -minimal elements of each v -difference set. A naive approach for generating all v -difference sets is then to compare every pair of distinct pre-image trees in $P_{T'}(v)$ and their projections to each essential v -property, starting from \sqsubseteq -minimal to \sqsubseteq -maximal essential v -properties. Once we find that the two pre-image trees differ on some v -property, we do not need to check the pre-image trees projections to any \sqsubseteq -larger essential v -property.

Alternatively, we can follow relational approaches and first find the corresponding v -agree sets. As opposed to how a v -difference set $\mathcal{D}_{\{p_1, p_2\}}(v)$ contains all essential v -properties on which two pre-image trees $p_1, p_2 \in P_{T'}(v)$ differ, a v -agree set $\mathcal{A}_{\{p_1, p_2\}}(v)$ aggregates all essential v -properties on which p_1, p_2 agree.

Definition 3.38 (agree set). Let T' be a T -compatible data tree with node v and pre-image trees $p_1, p_2 \in P_{T'}(v)$:

- $\mathcal{A}_{\{p_1, p_2\}}(v) := \{X \mid X \in \mathbf{E}_T^{\mathbb{S}}(v) \cup \mathbf{E}_T^{\mathbb{A}}(v) \text{ and } p_1|_X \doteq p_2|_X\}$
- $\mathfrak{A}_{T'}(v) := \{\mathcal{A}_{\{p_1, p_2\}}(v) \mid p_1, p_2 \in P_{T'}(v) \text{ and } p_1 \neq p_2\}$

We call $\mathcal{A}_{\{p_1, p_2\}}(v)$ the v -agree set of p_1 and p_2 and we call $\mathfrak{A}_{T'}(v)$ the family of all v -agree sets of T' . \square

Remark 3.39. Every v -agree set contains the root and empty subgraph.

Notations: From the \sqsubseteq -reflexivity axiom, every v -agree set is downward-closed with respect to p-subsumption and can be represented by its \sqsubseteq -maximal elements. For conciseness and clarity, examples will denote a v -agree set \mathcal{A} by its representative subset $\sqsubseteq\text{-max}(\mathcal{A})$ of \sqsubseteq -maximal v -properties, unless stated otherwise. Moreover, we use

$\text{rep}(\mathfrak{A}) = \{\sqsubseteq_{-\max}(\mathcal{A}) \mid \mathcal{A} \in \mathfrak{A}\}$ to explicitly refer to the family of representative subsets of v -agree sets in \mathfrak{A} . \square

Since every pair of pre-image trees $p_1, p_2 \in P_{T'}(v)$ must either agree or differ on each essential v -property,

$$\mathcal{D}_{\{p_1, p_2\}}(v) = \left(\mathbf{E}_T^{\check{S}}(v) \cup \mathbf{E}_T^{\check{A}}(v) \right) - \mathcal{A}_{\{p_1, p_2\}}(v).$$

Similarly, we find a relationship between the \sqsubseteq -minimal elements of v -difference sets (i.e., $\sqsubseteq_{-\min}(\mathcal{D}_{\{p_1, p_2\}}(v))$) and the \sqsubseteq -maximal elements of v -agree sets (i.e., $\sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$). This relationship is stated here. For a discussion of how to find all v -agree sets, or rather their representative subsets of \sqsubseteq -maximal v -properties, see Section 3.3.

For every essential v -property X we have

$$X \in \mathcal{D}_{\{p_1, p_2\}}(v) \text{ if and only if } X \not\sqsubseteq Y \text{ for every } Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$$

That is, we must find those essential v -properties which are not p -subsumed by any essential v -properties in $\sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$.

For v -ancestors this can be done with a simple lookup: $\sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$ contains at most one v -ancestor whose immediate essential descendant is the single v -ancestor belonging to $\mathcal{D}_{\{p_1, p_2\}}(v)$.

Finding v -subgraphs belonging to $\mathcal{D}_{\{p_1, p_2\}}(v)$ requires more effort. An essential v -subgraph X is not contained in any member of $\sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$ if and only if for every $Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$ we have that X contains at least one v -walk not contained in Y . Clearly, X is \sqsubseteq -minimal with this property if for every $Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v))$ we have X contains exactly one v -walk which is not contained in Y .

Recall that every essential v -subgraph is contained in exactly one v -unit. This makes it possible to compute $\mathcal{D}_{\{p_1, p_2\}}(v)$ (and $\mathcal{A}_{\{p_1, p_2\}}(v)$) by examining each v -unit individually. We next define the projection of v -difference sets and v -agree sets to some v -unit.

Definition 3.40. Given a v -unit U , the *projection of a v -difference set $\mathcal{D}_{\{p_1, p_2\}}(v)$ and a v -agree set $\mathcal{A}_{\{p_1, p_2\}}(v)$ to U* is as follows:

- $\mathcal{D}_{\{p_1, p_2\}}(v)|_U := \{X \in \mathcal{D}_{\{p_1, p_2\}}(v) \mid X \sqsubseteq U\}$
- $\mathcal{A}_{\{p_1, p_2\}}(v)|_U := \{X \in \mathcal{A}_{\{p_1, p_2\}}(v) \mid X \sqsubseteq U\}$

\square

Note that, since v -ancestors are not comparable with v -subgraphs, the projections of v -difference sets and v -agree sets to v -units always yield collections of essential v -subgraphs.

Every essential v -subgraph X , which is contained in U , either belongs to $\mathcal{D}_{\{p_1, p_2\}}(v)|_U$ or $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$. Thus, for every essential v -subgraph X contained in U we have

$$X \in \mathcal{D}_{\{p_1, p_2\}}(v)|_U \text{ if and only if } X \not\sqsubseteq Y \text{ for every } Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$$

Consider the set

$$\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{U - Y \neq \emptyset \mid Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)\}.$$

It is easy to make the connection that members of $\mathcal{D}_{\{p_1, p_2\}}(v)|_U$ are hitting set of the hypergraph $\mathcal{H} = (U, \overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U})$. With a slight abuse of notation, we say that members of $\mathcal{D}_{\{p_1, p_2\}}(v)|_U$ are hitting sets of $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$.

Lemma 3.41. *If $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} \neq \{\}$ then*

X is a hitting set of $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$ if and only if $X \in \mathcal{D}_{\{p_1, p_2\}}(v)|_U$

Proof. (\Rightarrow) Assume X is a hitting set of $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$. Every member of $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$ contains some v -walk which also belongs to X . According to how $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$ is defined, X is contained in U but is not a v -subgraph of any maximal element of $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$. Therefore $p_1|_X \neq p_2|_X$ and $X \in \mathcal{D}_{\{p_1, p_2\}}(v)|_U$.

(\Leftarrow) Suppose $X \in \mathcal{D}_{\{p_1, p_2\}}(v)|_U$ but X is not a hitting set of $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$. Then there is some non-empty v -subgraph $U - Y \in \overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$ such that X does not contain any walk in $U - Y$. From our initial assumption X is contained in U . It follows that X is contained in Y since it is a subgraph of U but does not contain any walk in $U - Y$. This leads to $X \in \mathcal{A}_{\{p_1, p_2\}}(v)|_U$ and $p_1|_X \doteq p_2|_X$. Hence $X \notin \mathcal{D}_{\{p_1, p_2\}}(v)|_U$, a contradiction. \square

We treat $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{\}$ as a special case because the minimal hitting set of $\{\}$ is the empty set of v -walk which corresponds to the empty v -subgraph, but this is hardly what we want since two pre-image trees *always* agree on the empty v -subgraph. More accurately, whenever we find $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{\}$ then the two pre-image trees agree on the v -unit itself and therefore $\mathcal{D}_{\{p_1, p_2\}}(v)|_U = \{\}$.

Lemma 3.42. *If $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{\}$ then $\mathcal{D}_{\{p_1, p_2\}}(v)|_U = \{\}$*

Proof. If $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{\}$ then U is the single \sqsubseteq -maximal v -subgraph of U in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$. In particular, $p_1|_X \doteq p_2|_X$ for every v -subgraph X contained in U . Hence $\mathcal{D}_{\{p_1, p_2\}}(v)|_U = \{\}$. \square

Corollary 3.43. *If $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{\}$ then $\sqsubseteq_{-\min}(\mathcal{D}_{\{p_1, p_2\}}(v)|_U) = \{\}$. Otherwise,*

X is a minimal hitting set of $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$ if and only if $X \in \sqsubseteq_{-\min}(\mathcal{D}_{\{p_1, p_2\}}(v)|_U)$

\square

Algorithm 3.6 outlines the steps for finding v -difference sets from v -agree sets as per Corollary 3.43 and the following example illustrates the process. Note that, apart from Transversal Hypergraph Generation algorithms, we can also use Algorithm 3.4 (`find_minimalTransversals`) for computing minimal hitting sets - this requires that we first transform each singleton set of v -subgraphs in $\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U}$ into a set of v -walks contained in the v -subgraph.

Algorithm 3.6 find_differenceSets-fromAgreeSets**Input:** $\text{rep}(\mathfrak{A}_{T'}(v))$ /* where $\text{rep}(\mathfrak{A}_{T'}(v)) := \{\sqsubseteq\text{-max}(\mathcal{A}) \mid \mathcal{A} \in \mathfrak{A}_{T'}(v)\}$ */**Output:** $\text{rep}(\mathfrak{D}_{T'}(v))$ /* where $\text{rep}(\mathfrak{D}_{T'}(v)) := \{\sqsubseteq\text{-min}(\mathcal{D}) \mid \mathcal{D} \in \mathfrak{D}_{T'}(v)\}$ */

1. $\mathfrak{D} := \{\}$
2. **for all** $\mathcal{A} \in \text{rep}(\mathfrak{A}_{T'}(v))$ **do**
3. $\mathcal{D} := \{\}$
4. **for all** v -unit U in T **do**
5. $\overline{\mathcal{A}}|_U := \{U - Y \neq \emptyset \mid Y \in \mathcal{A}|_U\}$
6. **if** $\overline{\mathcal{A}}|_U \neq \{\}$ **then**
7. $\text{hittingSet} := \{X \mid X \text{ is a minimal hitting set of } \overline{\mathcal{A}}|_U\}$
8. $\mathcal{D} := \mathcal{D} \cup \text{hittingSet}$
9. **end if**
10. **end for**
11. $\mathfrak{D} = \mathfrak{D} \cup \{\mathcal{D}\}$
12. **end for**
13. **return** \mathfrak{D}

Example 3.44. Let us find the dual $v_{Purchase}$ -difference set $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})$ for $v_{Purchase}$ -agree set $\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase}) = \{\text{Da}, \text{Ti}, \text{De}, \text{Pr}, v_{Outlet}\}$ in data tree $T'_{purchase}$. Note that all $v_{Purchase}$ -subgraphs in this agree set are \sqsubseteq -maximal. Recall that there are five $v_{Purchase}$ -units: Da, Ti, Pn, $\{\text{De}, \text{Pr}, \text{Di}\}$ and Sa.

The first two $v_{Purchase}$ -units are members of the agree set and therefore

$$\begin{aligned} \{\} &= \overline{\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\text{Da}}} = \overline{\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\text{Ti}}} \\ &= \mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\text{Da}} = \mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\text{Ti}} \end{aligned}$$

Since no element of the agree set is a subgraph of the singleton $v_{Purchase}$ -units Pn and Sa, it follows that $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\text{Pn}} = \{\text{Pn}\}$ and $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\text{Sa}} = \{\text{Sa}\}$.

For the last unit we have $\overline{\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\{\text{De}, \text{Pr}, \text{Di}\}}} = \{\{\text{Di}, \text{Pr}\}, \{\text{De}, \text{Di}\}\}$. The minimal hitting sets here yield: $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\{\text{De}, \text{Pr}, \text{Di}\}} = \{\text{Di}, \{\text{De}, \text{Pr}\}\}$.

Altogether, they give $\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase}) = \{\text{Pn}, \text{Sa}, \text{Di}, \{\text{De}, \text{Pr}\}\}$. \square

3.2.3 Finding Candidate RHSs

Another important question in the transversal approach is how to compute $\text{candRHS}_{T'}(v)$ efficiently (recall Definition 3.20 and Lemma 3.21). In addition to the generation of candidate RHSs, we also discuss one heuristic for re-using minimal transversal computations and the resulting simplified application of Theorem 3.24 for extracting pXFDs belonging to the canonical pXFD-cover.

The notion of right-maximality is based on the p-subsumption ordering which does not relate v -subgraphs to v -ancestors. Unsurprisingly, the generation of candidate v -subgraph and v -ancestor RHSs are handled separately and we discover all pXFDs in the canonical pXFD-cover with a v -ancestor RHS and those with a v -subgraph RHS independently as well. As shown in Algorithm 3.7, we proceed to compute the canonical pXFD-cover in two phases.

Algorithm 3.7 `find_canonicalCover`

Input: $*\mathcal{D}_{T'}(v)$, $\text{rep}(\mathcal{A}_{T'}(v))$

$/*$ where $*\mathcal{D}_{T'}(v) = \sqsubseteq\text{-min}(\{\sqsubseteq\text{-min}(\{\mathcal{D}\}) \mid \mathcal{D} \in \mathcal{D}_{T'}(v)\})$ $*/$

$/*$ where $\text{rep}(\mathcal{A}_{T'}(v)) = \{\sqsubseteq\text{-max}(\{\mathcal{A}\}) \mid \mathcal{A} \in \mathcal{A}_{T'}(v)\}$ $*/$

Output: $\mathcal{C}_{T'}(v)$

1. $/*$ Discover pXFDs having RHSs which are v -ancestors $*/$
 2. $\overset{\mathbb{A}}{\mathcal{C}}_{T'}(v) := \text{discoverXFDs-withAncestors}(*\mathcal{D}_{T'}(v))$
 3. $/*$ Discover pXFDs having RHSs which are v -subgraphs $*/$
 4. $\overset{\mathbb{S}}{\mathcal{C}}_{T'}(v) := \text{discoverXFDs-withSubgraphs}(*\mathcal{D}_{T'}(v), \text{rep}(\mathcal{A}_{T'}(v)))$
 5. **return** $\overset{\mathbb{S}}{\mathcal{C}}_{T'}(v) \cup \overset{\mathbb{A}}{\mathcal{C}}_{T'}(v)$
-

Even though we consider v -ancestors and v -subgraphs separately, there are a few observations which are useful for both cases. Irrespective of whether we are considering v -ancestors or v -subgraphs, every \sqsubseteq -maximal essential v -property Y is a candidate RHS, for the simple reason that we cannot find any essential v -properties which strictly p-subsumes Y , that is, there is no counter-example to prove that $v : \{Y\} \rightarrow \{Y\}$ is not right-maximal with respect to T' . Recall, the \sqsubseteq -maximal essential v -properties are: $n \in \mathcal{V}(\{v\})$ and every v -units of T' . Furthermore, in order to violate a pXFD we require the existence of at least two distinct pre-image trees of v in T' . Any data tree with only one pre-image tree of v can always be treated as a special case because, without performing a single discovery action, we know that all pXFDs are satisfied. Thus, we do not apply the transversal approach for pXFDs discovery at all in the case that the data tree has only one pre-image tree of v . In particular, our approach to finding all candidate RHSs $\text{candRHS}_{T'}(v)$ implicitly assumes the given data tree T' has two or more distinct pre-image trees of v .

We will start with identification of all candidate v -ancestor RHSs and application of the transversal approach for extracting pXFDs with a singleton set of essential v -ancestors as the RHS.

Essential v -ancestors as candidate RHSs

Whenever we have two essential v -properties X, Y and $\mathcal{D}_{T'}^X(v) = \mathcal{D}_{T'}^Y(v)$ then every minimal transversal for $\mathcal{D}_{T'}^X(v)$ is a minimal transversal for $\mathcal{D}_{T'}^Y(v)$. If, in addition, $X \sqsubseteq Y$ then, it is clear that no canonical left-reduced pXFDs with $\{X\}$ as the RHS can be right-maximal (see Corollary 3.19). Consider a partition of the set $\mathbf{E}_T^{\mathbb{A}}(v)$ as follows: for any

two essential v -ancestors $u, w \in \mathbf{E}_T^{\check{\Delta}}(v)$ we have

$$u, w \text{ belong to the same partition class if and only if } \mathfrak{D}_{T'}^u(v) = \mathfrak{D}_{T'}^w(v).$$

Then, the candidate v -ancestor RHSs are simply the \sqsubseteq -maximal essential v -ancestors belonging to each partition class above. Since p-subsumption linearly orders essential v -ancestor, we find exactly one candidate v -ancestor RHSs from each of the partition class and a partition of $*\mathfrak{D}_{T'}(v)$ as follows, for any two v -difference sets $\mathcal{D}_1, \mathcal{D}_2 \in *\mathfrak{D}_{T'}(v)$ we have

$$\mathcal{D}_1, \mathcal{D}_2 \in c_{m_i} \text{ if and only if } m_i \in \sqsubseteq\text{-min}(\mathcal{D}_1) \cap \sqsubseteq\text{-min}(\mathcal{D}_2).$$

Except for the \sqsubseteq -maximal essential v -ancestor $n \in \vartheta(\{v\})$, we can find the remaining candidate v -ancestor RHSs by the following definition.

Definition 3.45. Let m_1, m_2, \dots, m_k be the sequence of all essential v -ancestors which are \sqsubseteq -minimal in some v -difference set of T' such that $m_i \sqsubseteq m_j$ whenever $i \leq j$, for $i, j = 1, \dots, k$. Then

$$RHS(m_1) = \sqsubseteq\text{-max}(\{n \in \mathbf{E}_T^{\check{\Delta}}(v) \mid n \sqsubset m_1\})$$

and for $i = 2, \dots, k$

$$RHS(m_i) = \sqsubseteq\text{-max}(\{n \in \mathbf{E}_T^{\check{\Delta}}(v) \mid m_{i-1} \sqsubseteq n \sqsubset m_i\}).$$

We call $RHS(m_i)$ for $i = 1, \dots, k$ and $n \in \vartheta(\{v\})$ candidate v -ancestor RHSs of T' . \square

Figure 3.4 visually depicts the relationship between the \sqsubseteq -minimal essential v -ancestors of the v -difference sets and the candidate v -ancestors RHSs. Each $RHS(m_i)$ is the immediate essential descendant of m_i (for $i = 1, \dots, k$). Moreover, because of the duality between difference sets and agree sets, it follows that

$$\mathcal{D}_1, \mathcal{D}_2 \in c_{m_i} \text{ if and only if } RHS(m_i) \in \sqsubseteq\text{-max}(\mathcal{A}_1) \cap \sqsubseteq\text{-max}(\mathcal{A}_2)$$

where \mathcal{A}_j and \mathcal{D}_j are dual of each other (for $i = 1, 2$). In words, each $RHS(m_i)$ is the \sqsubseteq -maximal v -ancestor in the dual v -agree sets of the v -difference sets in m_i -partition class c_{m_i} . This makes finding all candidate v -ancestor RHSs simple: we just need to find the set

$$\mathbf{max}^{\check{\Delta}}(\mathfrak{A}) = \bigcup_{\mathcal{A} \in \mathfrak{A}_{T'}(v)} \sqsubseteq\text{-max}(\mathcal{A} \cap \mathbf{E}_T^{\check{\Delta}}(v))$$

and replace the \sqsubseteq -maximal v -ancestor in $\mathbf{max}^{\check{\Delta}}(\mathfrak{A})$ by $n \in \vartheta(\{v\})$.

Intuitively, the following observations show that each candidate v -ancestor RHS is a representative for a partition class in the partition of $\mathbf{E}_T^{\check{\Delta}}(v)$ as described earlier. No essential v -ancestors which is strictly p-subsumed by m_1 belong to any v -difference set of T' and so

$$\mathfrak{D}_{T'}^{RHS(m_1)}(v) = \{v\}.$$

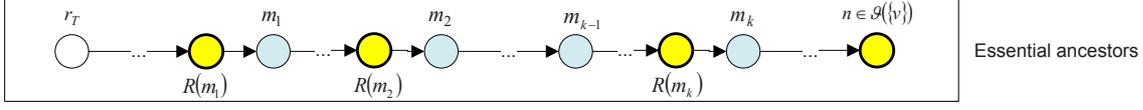


Figure 3.4: Relationship between essential v -ancestors and candidate v -ancestor RHSs.

Since $RHS(m_{i-1}) \sqsubset m_{i-1} \sqsubseteq RHS(m_i)$, and v -difference sets are \sqsubseteq -upward-closed, it is clear that

$$\mathfrak{D}_{T'}^{RHS(m_i)}(v) = \mathfrak{D}_{T'}^{m_{i-1}}(v) = \mathfrak{D}_{T'}^{RHS(m_{i-1})}(v) \cup c_{m_{i-1}} = c_{m_1} \cup \dots \cup c_{m_{i-1}} \text{ for } i = 2, \dots, k.$$

Both $n \in \vartheta(\{v\})$ and m_k belong to every v -difference set of T' (although only m_k is \sqsubseteq -minimal). This gives

$$\mathfrak{D}_{T'}^n(v) = \mathfrak{D}_{T'}^{m_k}(v) = \mathfrak{D}_{T'}(v) \text{ where } \{n\} = \vartheta(\{v\}).$$

More formally, the next theorem show that candidate v -ancestor RHSs are precisely the candidate RHSs which are v -ancestors which we are interested in.

Theorem 3.46. *Let u be an essential v -ancestor. Then $u \in candRHS_{T'}(v)$ if and only if $\{u\} = \vartheta(\{v\})$ or $u = RHS(m_i)$ for some $i = 1, \dots, k$.*

Proof. (\Leftarrow) If $\{u\} = \vartheta(\{v\})$ then u is the \sqsubseteq -maximal essential v -ancestor. As such there is no counter-witness to $v : \vartheta(\{v\}) \rightarrow \vartheta(\{v\})$ being right-maximal. Next consider $u = RHS(m_i)$. Since $u \sqsubset m_i$, then $u \notin \mathcal{D}$ for any v -difference set $\mathcal{D} \in c_{m_j}$ where $j \geq i$. In other words, $\{u\}$ is not be a minimal transversal for any $\mathfrak{D}_{T'}^w(v)$ where $w \sqsupset u$. Correspondingly $v : \{u\} \rightarrow \{w\}$ does not hold in T' for any $w \sqsupset u$ which means $v : \{u\} \rightarrow \{u\}$ is right-maximal w.r.t. T' .

(\Rightarrow) We present a contra-positive proof. Assume that $\{u\} \neq \vartheta(\{v\})$ and $u \neq RHS(m_i)$ for all $i = 1, \dots, k$. Let $\{n\} = \vartheta(\{v\})$.

If $m_k \sqsubseteq u$ then $u \sqsubset n$ and also $\mathfrak{D}_{T'}^{m_k}(v) = \mathfrak{D}_{T'}(v) = \mathfrak{D}_{T'}^n(v) = \mathfrak{D}_{T'}^u(v)$. Clearly $\{u\}$ is a transversal for $\mathfrak{D}_{T'}^u(v)$ which then implies $\{u\}$ is a transversal for $\mathfrak{D}_{T'}^n(v)$. This means T' satisfies both $v : \{u\} \rightarrow \{u\}$ and $v : \{u\} \rightarrow \{n\}$, which means that $v : \{u\} \rightarrow \{u\}$ is not right-maximal w.r.t. T' and therefore $u \notin candRHS_{T'}(v)$.

Alternatively, there is some essential v -ancestor m_j such that $u \sqsubset m_j$ and $1 \leq j \leq k$. Should $j = 1$, no v -difference set contains u and we have $\mathfrak{D}_{T'}^u(v) = \{\} = \mathfrak{D}_{T'}^{RHS(m_1)}(v)$. As a result, T' satisfies $v : \{\} \rightarrow \{RHS(m_1)\}$ and $v : \{\} \rightarrow \{u\}$. This then means that T' also satisfies $v : \{u\} \rightarrow \{RHS(m_1)\}$ and $v : \{u\} \rightarrow \{u\}$. It follows that $v : \{u\} \rightarrow \{u\}$ is not right-maximal w.r.t. T' and again $u \notin candRHS_{T'}(v)$. For $2 \leq j \leq k$ we have $m_{j-1} \sqsubseteq u \sqsubset RHS(m_j)$ and consequently $\mathfrak{D}_{T'}^{m_{j-1}}(v) = \mathfrak{D}_{T'}^u(v) = \mathfrak{D}_{T'}^{RHS(m_j)}(v)$. Thus $\{u\}$ is a transversal for both $\mathfrak{D}_{T'}^u(v)$ and $\mathfrak{D}_{T'}^{RHS(m_j)}(v)$. That is, $v : \{u\} \rightarrow \{u\}$ is not right-maximal w.r.t. T' because T' also satisfies $v : \{u\} \rightarrow \{RHS(m_j)\}$. Thus completes the proof. \square

Example 3.47. The $v_{Purchase}$ -difference sets of $T'_{purchase}$ (from Example 3.17) can be partitioned into two classes: one identified by v_{Outlet} and the other identified by $v_{Purchase}$. Since $RHS(v_{Outlet})$ is the root, we only get two noteworthy candidate $v_{Purchase}$ -ancestor RHSs:

$$\vartheta(\{v_{Purchase}\}) = v_{Purchase} \text{ and } RHS(v_{Purchase}) = v_{Outlet}.$$

□

Now we can address how to apply the transversal approach for extracting pXFDs belonging to $\mathring{\mathfrak{C}}_{T'}(v)$. Our approach requires one minimal transversal computation - for the family $\mathfrak{D}_{T'}(v)$ - and interleaves extraction of pXFDs within the minimal transversal computation.

With *exactly one* application of the minimal transversal computation from Section 3.2.1, it is possible to determine *all* pXFDs in the canonical pXFD-cover of T' having only essential v -ancestors in the RHS. There is a linear p-subsumption ordering of all candidate v -ancestor RHSs:

$$RHS(m_1) \sqsubset RHS(m_2) \sqsubset \dots \sqsubset RHS(m_k) \sqsubset n \text{ where } \{n\} = \vartheta(\{v\})$$

which means that:

$$\{\} = \mathfrak{D}_{T'}^{RHS(m_1)}(v) \subset \mathfrak{D}_{T'}^{RHS(m_2)}(v) \subset \dots \subset \mathfrak{D}_{T'}^{RHS(m_k)}(v) \subset \mathfrak{D}_{T'}^n(v) = \mathfrak{D}_{T'}(v).$$

Recall that $\mathfrak{D}_{T'}^{m_k}(v) = \mathfrak{D}_{T'}^n(v)$ and $\mathfrak{D}_{T'}^{m_i}(v) = \mathfrak{D}_{T'}^{RHS(m_{i+1})}(v)$ for $i = 1, \dots, k-1$. We can consider all v -difference sets in the partition class c_{m_1} , then all the v -difference sets in c_{m_2} and so on until we have considered all v -difference sets of T' . Doing so will enable us to find the minimal transversals for all families $\mathfrak{D}_{T'}^{RHS(m_i)}(v)$ with $i = 1, \dots, k$ in the process of finding the minimal transversals for $\mathfrak{D}_{T'}(v)$. Moreover, we can extract pXFDs during the minimal transversal computation, rather than when the computation completes. For an essential v -ancestor Y , we add pXFD $v : \mathcal{X} \rightarrow \{Y\}$ to the canonical pXFD-cover $\mathfrak{C}_{T'}(v)$ in each of the following cases:

- before starting to compute $\mathfrak{Tr}(\mathfrak{D}_{T'}(v))$
 - $Y = RHS(m_1)$ and $\mathcal{X} = \{\}$, provided $RHS(m_1) \neq r_T$
- after computing minimal transversals for $\mathfrak{D}_{T'}^{RHS(m_{i+1})}(v)$ for $i = 2, \dots, k$
 - $Y = RHS(m_i)$ and $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^{RHS(m_i)}(v)) - \{\{Y\}\} - \mathfrak{Tr}(\mathfrak{D}_{T'}^{RHS(m_{i+1})}(v))$
- after finishing to compute $\mathfrak{Tr}(\mathfrak{D}_{T'}(v))$
 - $\{Y\} = \vartheta(\{v\})$ and $\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}(v)) - \{\{Y\}\}$

Applications of Theorem 3.24 is simplified into the three cases above. Observe that $\mathfrak{D}_{T'}^{RHS(m_1)}(v) = \{\}$ can only have the empty set as a minimal transversal and is the only

family of v -difference sets modulo some candidate v -ancestor RHS to be empty. Because we are not considering v -subgraph the non-triviality test in Theorem 3.24 simplifies to

$$\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v)) - \{\{Y\}\} \text{ and } Y \in \left(\text{candRHS}_{T'}(v) \cap \mathbf{E}_T^{\mathbb{A}}(v) \right) - \{r_T\}.$$

But since we assume v to be not simple, only $RHS(m_1)$ may be the root node and $n \in \vartheta(\{v\})$ cannot be a minimal transversal for any family $\mathfrak{D}_{T'}^{RHS(m_i)}(v)$ where $i = 1, \dots, k$. Right-maximality check requires only a pair $\mathfrak{D}_{T'}^{RHS(m_j)}(v), \mathfrak{D}_{T'}^{RHS(m_{j+1})}(v)$ of consecutive candidate v -ancestor RHSs. This is because the minimal transversals of $\mathfrak{D}_{T'}^{RHS(m_l)}(v)$ for $m < l \leq k$ are extended from the minimal transversals of $\mathfrak{D}_{T'}^{RHS(m_j)}(v)$ by melding, and so, $\mathfrak{Tr}(\mathfrak{D}_{T'}^{RHS(m_j)}(v)) \sqsubseteq \mathfrak{Tr}(\mathfrak{D}_{T'}^{RHS(m_{j+1})}(v)) \sqsubseteq \mathfrak{Tr}(\mathfrak{D}_{T'}^{RHS(m_l)}(v))$ for $m < l \leq k$.

There are two noteworthy benefits of interleaving the extraction of pXFDs with the minimal transversal computation. Firstly, we shorten the time that it takes to discover the first pXFD belonging to the canonical pXFD-cover. The second benefit is that, at any point in the computation, we need only to keep track of *at most two* families of minimal transversals, as opposed to for all candidate v -ancestor RHSs.

The steps that we have just described for extracting all pXFDs with v -ancestors in the RHS is summarised in Algorithm 3.8 and illustrated in the next example.

Example 3.48. Continuing on from Example 3.47, let us find all pXFDs belonging to the canonical pXFD-cover $\mathfrak{C}_{T'_{purchase}}(v_{Purchase})$ with only essential v -ancestors in the RHS.

Three of the five $v_{Purchase}$ -difference sets of $T'_{purchase}$

$$\mathcal{D}_{(i_5, i_{31})}(v_{Purchase}), \mathcal{D}_{(i_5, i_{44})}(v_{Purchase}), \text{ and } \mathcal{D}_{(i_{18}, i_{31})}(v_{Purchase})$$

form the v_{Outlet} -partition class and the family $\mathfrak{D}_{T'_{purchase}}^{v_{Outlet}}(v_{Purchase})$. The remaining two $v_{Purchase}$ -difference sets

$$\mathcal{D}_{(i_5, i_{18})}(v_{Purchase}) \text{ and } \mathcal{D}_{(i_{31}, i_{44})}(v_{Purchase})$$

form the $v_{Purchase}$ -partition class.

We have $v_{Outlet} \sqsubset v_{Purchase}$. Therefore, when computing the minimal transversals for $\mathfrak{D}_{T'_{purchase}}(v_{Purchase})$ we should consider the three $v_{Purchase}$ -difference sets in the v_{Outlet} -partition class first, and then the two remaining $v_{Purchase}$ -difference sets from the $v_{Purchase}$ -partition class. One possible minimal transversals computation is the one from Example 3.37.

Before we start the minimal transversal computation, we consider $RHS(m_1)$. Since $RHS(m_1) = r_{T'_{purchase}}$ we extract no pXFD with $\{RHS(m_1)\}$ as the RHS. After completing the minimal transversal computation for $\mathfrak{D}_{T'_{purchase}}(v_{Purchase})$ we find

$$\begin{aligned} \mathfrak{Tr}(\mathfrak{D}_{T'_{purchase}}^{v_{Outlet}}(v_{Purchase})) &= \{ \{Ti\}, \{Pn\}, \{De, Di\}, \{De, Sa\}, \{Pr\}, \{v_{Outlet}\} \} \\ \mathfrak{Tr}(\mathfrak{D}_{T'_{purchase}}^{v_{Purchase}}(v_{Purchase})) &= \left\{ \begin{array}{l} \{Ti, Di\}, \{Ti, Sa\}, \{Pn\}, \{De, Di\}, \{De, Sa\}, \\ \{\{De, Pr\}\}, \{Pr, Di\}, \{Pr, Sa\}, \{v_{Outlet}, Di\}, \\ \{v_{Outlet}, Sa\}, \{v_{Purchase}\} \end{array} \right\} \end{aligned}$$

Algorithm 3.8 discoverXFDs-withAncestors**Input:** $*\mathcal{D}_{T'}(v)$ /* where $*\mathcal{D}_{T'}(v) = \sqsubseteq_{-\min}(\{\sqsubseteq_{-\min}(\{\mathcal{D}\}) \mid \mathcal{D} \in \mathcal{D}_{T'}(v)\})$ */**Output:** $\check{\mathcal{C}}_{T'}(v)$ /* where $\check{\mathcal{C}}_{T'}(v) = \{v : \mathcal{X} \rightarrow \{Y\} \mid v : \mathcal{X} \rightarrow \{Y\} \in \mathcal{C}_{T'}(v) \text{ and } Y \in \mathbf{E}_T^{\check{\mathcal{A}}}(v)\}$ */

1. $\check{\mathcal{C}}_{T'}(v) := \{\}$
2. $partition := \text{find_ancestorPartition}(*\mathcal{D}_{T'}(v))$
3. $c_{m_i} := \text{Remove first partition class from } partition$
4. $Y := \text{RHS}(m_i)$
5. /* All interesting pXFDs with $\{\text{RHS}(m_1)\}$ as the RHS */
6. **if** $Y \neq r_T$ **then**
7. $\check{\mathcal{C}}_{T'}(v) := \check{\mathcal{C}}_{T'}(v) \cup \{v : \{\} \rightarrow \{Y\}\}$
8. **end if**
9. /* All interesting pXFDs with $\{\text{RHS}(m_i)\}$ as the RHS for $i = 2, \dots, k$ */
10. $oldTrans := \text{find_minimalTransversals}(c_{m_i})$
11. **while** $partition$ is not empty **do**
12. $c_{m_{i+1}} := \text{Remove next partition class from } partition$
13. $Y := \text{RHS}(m_{i+1})$ // $= \sqsubseteq_{-\max}(\{n \in \mathbf{E}_T^{\check{\mathcal{A}}}(v) \mid m_{i-1} \sqsubseteq n \sqsubseteq m_i\})$
14. $nextTrans := \text{extend_minimalTransversals}(oldTrans, c_{m_{i+1}})$
15. /* Extract pXFDs */
16. **for all** $\mathcal{X} \in oldTrans - \{\{Y\}\} - nextTrans$ **do**
17. $\check{\mathcal{C}}_{T'}(v) := \check{\mathcal{C}}_{T'}(v) \cup \{v : \mathcal{X} \rightarrow \{Y\}\}$
18. **end for**
19. $oldTrans := nextTrans$
20. **end while**
21. /* All interesting pXFDs with $\{\vartheta(\{v\})\}$ as the RHS */
22. **for all** $\mathcal{X} \in oldTrans - \{\vartheta(\{v\})\}$ **do**
23. $\check{\mathcal{C}}_{T'}(v) := \check{\mathcal{C}}_{T'}(v) \cup \{v : \mathcal{X} \rightarrow \vartheta(\{v\})\}$
24. **end for**
25. **return** $\check{\mathcal{C}}_{T'}(v)$
26. **proc** $\text{find_ancestorPartition}(*\mathcal{D}_{T'}(v))$
27. Partition $*\mathcal{D}_{T'}(v)$ into $c_{m_i} = \{\sqsubseteq_{-\min}(\mathcal{D}) \in *\mathcal{D}_{T'}(v) \mid m_i \in \sqsubseteq_{-\min}(\mathcal{D})\}$ for $i = 1, \dots, k$
28. **return** c_{m_1}, \dots, c_{m_k} where $m_i \sqsubseteq m_j$ whenever $i \leq j$ for $i, j = 1, \dots, k$

The two families have three minimal transversals in common, these are: $\{\text{Pn}\}$, $\{\text{De}, \text{Di}\}$ and $\{\text{De}, \text{Sa}\}$. Also $\{v_{\text{Outlet}}\}$ would only yield a trivial pXFD with $\{v_{\text{Outlet}}\}$ as the RHS. Hence, the canonical pXFD-cover contains only two pXFDs with $\{v_{\text{Outlet}}\}$ as the RHS:

$$\begin{aligned} v_{\text{Purchase}} : \{\text{Ti}\} &\rightarrow \{v_{\text{Outlet}}\} \\ v_{\text{Purchase}} : \{\text{Pr}\} &\rightarrow \{v_{\text{Outlet}}\} \end{aligned}$$

Finally, we extract pXFDs with $\{v_{\text{Purchase}}\}$ as the RHS. Apart from $\{v_{\text{Purchase}}\}$, which would only yield a trivial pXFD, the other ten minimal transversals of $\mathfrak{D}_{T'_{\text{purchase}}}^{v_{\text{Purchase}}}(\{v_{\text{Purchase}}\})$ each yields a pXFD with $\{v_{\text{Purchase}}\}$ as the RHS that belong to the canonical pXFD-cover of T'_{purchase} :

$$\begin{array}{ll} v_{\text{Purchase}} : \{\text{Ti}, \text{Di}\} &\rightarrow \{v_{\text{Purchase}}\} & v_{\text{Purchase}} : \{\{\text{De}, \text{Pr}\}\} &\rightarrow \{v_{\text{Purchase}}\} \\ v_{\text{Purchase}} : \{\text{Ti}, \text{Sa}\} &\rightarrow \{v_{\text{Purchase}}\} & v_{\text{Purchase}} : \{\text{Pr}, \text{Di}\} &\rightarrow \{v_{\text{Purchase}}\} \\ v_{\text{Purchase}} : \{\text{Pn}\} &\rightarrow \{v_{\text{Purchase}}\} & v_{\text{Purchase}} : \{\text{Pr}, \text{Sa}\} &\rightarrow \{v_{\text{Purchase}}\} \\ v_{\text{Purchase}} : \{\text{De}, \text{Di}\} &\rightarrow \{v_{\text{Purchase}}\} & v_{\text{Purchase}} : \{v_{\text{Outlet}}, \text{Di}\} &\rightarrow \{v_{\text{Purchase}}\} \\ v_{\text{Purchase}} : \{\text{De}, \text{Sa}\} &\rightarrow \{v_{\text{Purchase}}\} & v_{\text{Purchase}} : \{v_{\text{Outlet}}, \text{Sa}\} &\rightarrow \{v_{\text{Purchase}}\} \end{array}$$

□

Essential v -subgraphs as candidate RHS.

To determine all candidate subgraph RHSs, we can consider each v -unit U of T independently. That is, we seek to compute

$$\text{candRHS}_{T'}(v)|_U = \{Y \in \text{candRHS}_{T'}(v) \mid Y \sqsubseteq U\}$$

We call every essential v -subgraph which is a member of $\text{candRHS}_{T'}(v)$ *e-closed*. We first extend the notion of being “closed” to sets of essential v -subgraphs $\mathcal{Y} \sqsubseteq \{U\}$.

Definition 3.49. Let U be a v -unit of T and $\mathcal{Y} \subseteq \mathbf{E}_T^{\check{S}}(v)$ be a set of essential v -subgraphs with $\mathcal{Y} \sqsubseteq \{U\}$. We call \mathcal{Y} *u-closed* (w.r.t. a T -compatible data tree T') if and only if for every $\mathcal{Y}' \subseteq \mathbf{E}_T^{\check{S}}(v)$ with $\mathcal{Y} \sqsubset \mathcal{Y}' \sqsubseteq \{U\}$ the pXFD $v : \mathcal{Y} \rightarrow \mathcal{Y}'$ does not hold in T' . □

Note that the notions of being e-closed and u-closed are different. While every u-closed singleton set $\{Y\}$ is also e-closed (we will show this later), the opposite does not hold in general.

Example 3.50. Let $U = AB$ be a v -unit of T with two v -walks A and B . Further, suppose there is a T -compatible data tree T' such that $\models_{T'} v : \{A\} \rightarrow \{B\}$, but $\not\models_{T'} v : \{A\} \rightarrow \{AB\}$. Then $\{A\}$ is e-closed but not u-closed w.r.t. T' because $\models_{T'} v : \{A\} \rightarrow \{A, B\}$ with $\{A\} \sqsubseteq \{A, B\}$. □

Even though not every e-closed singleton set $\{Y\}$ is also u-closed, it is possible to characterise e-closed Y in terms of u-closed sets.

Lemma 3.51. *Let U be a v -unit and Y an essential v -subgraph contained in U . Then Y is e-closed if and only if there exists a u-closed set $\mathcal{Y} \sqsubseteq \{U\}$ with $Y \in \mathcal{Y}$ being \sqsubseteq -maximal.*

Proof. (\Rightarrow): Let \mathcal{Y} be the closure of Y projected to U as follows

$$\mathcal{Y} = \{Y' \in \mathbf{E}_T^{\check{S}}(v) \mid Y' \sqsubseteq U \text{ and } v : \{Y\} \rightarrow \{Y'\} \text{ holds in } T'\}$$

Since Y is e-closed, Y is \sqsubseteq -maximal in \mathcal{Y} . It remains to show that \mathcal{Y} is u-closed. For this let $\mathcal{Y}' \sqsubseteq \{U\}$ with $\models_{T'} v : \mathcal{Y} \rightarrow \mathcal{Y}'$. Since $v : \{Y\} \rightarrow \mathcal{Y}$ holds in T' , we get $\models_{T'} v : \{Y\} \rightarrow \mathcal{Y}'$, and thus $\mathcal{Y}' \subseteq \mathcal{Y}$ by definition of \mathcal{Y} .

(\Leftarrow): Assume to the contrary that Y is not e-closed, so that $v : \{Y\} \rightarrow \{Y'\}$ holds in T' for some essential v -subgraph $Y' \sqsupset Y$. Since $Y \in \mathcal{Y}$ we get $\models_{T'} v : \mathcal{Y} \rightarrow \mathcal{Y}'$ with $\mathcal{Y}' = \mathcal{Y} \cup \{Y'\}$. Since Y is maximal in \mathcal{Y} we have $\mathcal{Y} \sqsubset \mathcal{Y}'$, which contradicts \mathcal{Y} being u-closed. \square

Note that Lemma 3.51 shows in particular that u-closed singleton sets are also e-closed, as claimed earlier.

We characterise the u-closed sets of T' via the v -agree sets of T' next. For ease of presentation, whilst discussing the generation of candidate subgraph RHSs, we denote the set $\mathfrak{A}_{T'}(v)$ of all v -agree sets of T' simply by \mathfrak{A} , and the projection of its elements to a v -unit U by $\mathfrak{A}|_U$.

Definition 3.52. Let M be a set and let $S \subseteq \mathcal{P}(M)$ be a family of subsets of M . Then the \cap -ideal generated by S , denoted by $\langle S \rangle$, is the set of all intersections of subsets of S :

$$\langle S \rangle = \left\{ \bigcap S' \mid S' \subseteq S \right\}$$

with $\bigcap \{\} = M$. Clearly $S \cup \{M\} \subseteq \langle S \rangle \subseteq \mathcal{P}(M)$, and $\langle \cdot \rangle$ is a closure operation. \square

Next we illustrate the notion of a \cap -ideal generated by a given family $\mathfrak{A}|_U$.

Example 3.53. Let $U = ABCDE$ be a v -unit and

$$\mathfrak{A}|_U = \left\{ \begin{array}{l} \{AB, A, B, ACD, AC, AD, CD, C, D\} \\ \{ACE, AC, AE, CE, A, C, E, BE, B\} \\ \{AC, A, C, BC, B\} \end{array} \right\}$$

Let $\check{U} = \{ABCDE, ABCD, ABCE, ABDE, ACDE, \dots, A, B, C, D, E\}$ be the set of all v -subgraphs contained in U . Then $\mathfrak{A}|_U \subseteq \mathcal{P}(\check{U})$

There are $2^3 = 8$ subsets of the given $\mathfrak{A}|_U$. From the intersections for singleton subsets of $\mathfrak{A}|_U$ we obtain $\mathfrak{A}|_U \subseteq \langle \mathfrak{A}|_U \rangle$. Moreover $\bigcap \{\} = \check{U} \in \langle \mathfrak{A}|_U \rangle$.

Next we find the intersections for subfamilies of $\mathfrak{A}|_U$ containing two or three sets:

- $\{AB, A, B, ACD, AC, AD, CD, C, D\} \cap \{ACE, AC, AE, CE, A, C, E, BE, B\} = \{AC, A, C, B\}$
- $\{AB, A, B, ACD, AC, AD, CD, C, D\} \cap \{AC, A, C, BC, B\} = \{AC, A, C, B\}$
- $\{ACE, AC, AE, CE, A, C, E, BE, B\} \cap \{AC, A, C, BC, B\} = \{AC, A, C, B\}$

- $\{AB, A, B, ACD, AC, AD, CD, C, D\}$
 $\cap \{ACE, AC, AE, CE, A, C, E, BE, B\} \cap \{AC, A, C, BC, B\}$
 $= \{AC, A, C, B\}$

In summary

$$\langle \mathfrak{A}|_U \rangle = \left\{ \begin{array}{l} \{AB, A, B, ACD, AC, AD, CD, C, D\}, \\ \{ACE, AC, AE, CE, A, C, E, BE, B\}, \\ \{AC, A, C, BC, B\}, \\ \{ABCDE, ABCD, ABCE, ABDE, ACDE, \dots, A, B, C, D, E\}, \\ \{AC, A, C, B\} \end{array} \right\}$$

□

It turns out that for every v -unit U , the family of all u -closed sets is exactly the \cap -ideal generated by $\mathfrak{A}|_U$. We call sets of v -subgraphs which are downward-closed with respect to p -subsumption *complete sets*. Remember v -agree sets are complete sets. Note also that the \cap -ideal generated by a family of complete sets contains only complete sets. It is sufficient to consider only complete u -closed sets when determining all candidate subgraph RHSs, since we are only interested in their maximal elements.

Lemma 3.54. *Let U be a v -unit of T , and $\mathcal{Y} \subseteq \mathbf{E}_T^{\mathfrak{S}}(v)$ be complete set of essential v -subgraphs with $\mathcal{Y} \sqsubseteq \{U\}$. Further, let \mathfrak{A} be the family of all v -agree sets of T -compatible data tree T' . Then \mathcal{Y} is u -closed if and only if*

$$\mathcal{Y} = \bigcap \{ \mathcal{A} \in \mathfrak{A}|_U \mid \mathcal{Y} \subseteq \mathcal{A} \} \quad (1)$$

Proof. (\Rightarrow): Since \mathcal{Y} is complete and u -closed, the pXFD $v : \mathcal{Y} \rightarrow \{Z\}$ does not hold for any $Z \in \mathbf{E}_T^{\mathfrak{S}}(v) - \mathcal{Y}$ where $Z \sqsubseteq U$, as this would imply $v : \mathcal{Y} \rightarrow \mathcal{Y} \cup \{Z\}$ with $\mathcal{Y} \subset \mathcal{Y} \cup \{Z\} \subseteq \mathbf{E}_T^{\mathfrak{S}}(v)$ and $\mathcal{Y} \cup \{Z\} \sqsubseteq \{U\}$ which contradicts \mathcal{Y} being u -closed. And so there must exist some v -agree set $\mathcal{A} \in \mathfrak{A}|_U$ with $\mathcal{Y} \subseteq \mathcal{A}$ but $Z \notin \mathcal{A}$. This gives us (1).

(\Leftarrow): Assume (1) holds. Suppose there is some $\mathcal{Y}' \supset \mathcal{Y}$ with $\mathcal{Y}' \sqsubseteq \{U\}$. Then there exists some v -agree set $\mathcal{A} \in \mathfrak{A}|_U$ with $\mathcal{Y} \subseteq \mathcal{A}$ but $\mathcal{Y}' \not\subseteq \mathcal{A}$. Thus $v : \mathcal{Y} \rightarrow \mathcal{Y}'$ does not hold in T' . This shows \mathcal{Y} is u -closed, as required. □

Lemma 3.55. *Let U be a v -unit of T , and \mathfrak{AC}_U the family of all complete u -closed subsets of $\mathbf{E}_T^{\mathfrak{S}}(v)$ which are p -subsumed by $\{U\}$. Further, let \mathfrak{A} be the family of all v -agree sets of T -compatible data tree T' . Then*

$$\mathfrak{AC}_U = \langle \mathfrak{A}|_U \rangle$$

Proof. (\subseteq): Clear by Lemma 3.54.

(\supseteq): Let $\mathcal{Y} \in \langle \mathfrak{A}|_U \rangle$, so that $\mathcal{Y} = \bigcap \mathfrak{A}'$ for some $\mathfrak{A}' \subseteq \mathfrak{A}|_U$. This gives us:

$$\mathcal{Y} = \bigcap \{ \mathcal{A} \in \mathfrak{A}|_U \mid \mathcal{Y} \subseteq \mathcal{A} \}$$

Thus \mathcal{Y} is u -closed by Lemma 3.54. □

We can therefore compute the candidate subgraph RHSs by finding the maximal elements in the sets $\langle \mathfrak{A}|_U \rangle$. We denote the \sqsubseteq -maximal elements of sets in \mathfrak{A} by

$$\mathbf{max}(\mathfrak{A}) = \bigcup_{\mathcal{A} \in \mathfrak{A}} \sqsubseteq\text{-max}(\mathcal{A})$$

With this notation, Lemma 3.51 and Lemma 3.55 together give us

Theorem 3.56. $\mathit{candRHS}_{T'}(v)|_U = \mathbf{max}(\langle \mathfrak{A}|_U \rangle)$

Note that $\mathbf{max}(\mathfrak{A})$ is not a set of v -subgraphs in general, but $\mathbf{max}(\langle \mathfrak{A}|_U \rangle)$ yields only v -subgraphs because $\mathfrak{A}|_U$ itself contains only v -subgraphs.

Example 3.57. Reconsider Example 3.53 where $U = ABCDE$ is a v -unit and

$$\mathfrak{A}|_U = \left\{ \begin{array}{l} \{AB, A, B, ACD, AC, AD, CD, C, D\} \\ \{ACE, AC, AE, CE, A, C, E, BE, B\} \\ \{AC, A, C, BC, B\} \end{array} \right\}$$

be the family of complete v -agree sets projected to U . We found

$$\langle \mathfrak{A}|_U \rangle = \left\{ \begin{array}{l} \{AB, A, B, ACD, AC, AD, CD, C, D\}, \\ \{ACE, AC, AE, CE, A, C, E, BE, B\}, \\ \{AC, A, C, BC, B\}, \\ \{ABCDE, ABCD, ABCE, ABDE, ACDE, \dots, A, B, C, D, E\}, \\ \{AC, A, C, B\} \end{array} \right\}$$

The \sqsubseteq -maximal elements for each set belonging to $\langle \mathfrak{A}|_U \rangle$ are highlighted by *bold* font. Simply collecting together all these \sqsubseteq -maximal elements gives us

$$\mathit{candRHS}_{T'}(v)|_U = \mathbf{max}(\langle \mathfrak{A}|_U \rangle) = \{AB, ACD, ACE, BE, AC, BC, ABCDE, B\}$$

□

Computationally, computing $\langle \mathfrak{A}|_U \rangle$ to extract the candidate v -subgraph RHSs from it can be expensive though, since the number of u-closed sets can be much larger than the number of its \sqsubseteq -maximal elements.

Example 3.58. Let $U = a_1 \dots a_n$ be a v -unit, and

$$\mathfrak{A}|_U = \left\{ \begin{array}{l} \{\{a_1\}, \{a_2\}, \{a_3\}, \dots, \{a_{n-1}\}, \{a_n\}\} \\ \{\{a_2\}, \{a_3\}, \dots, \{a_{n-1}\}, \{a_n\}\} \\ \{\{a_1\}, \{a_3\}, \dots, \{a_{n-1}\}, \{a_n\}\} \\ \vdots \\ \{\{a_1\}, \{a_2\}, \{a_3\}, \dots, \{a_{n-1}\}\} \end{array} \right\}$$

Then the u-closed subsets of $\mathcal{P}(\check{U})$, apart from $\mathcal{P}(U)$ itself, are exactly those containing only singletons. Thus there are $2^n + 1$ u-closed sets, but they contain only $n + 1$ different maximal elements. □

Instead, we aim to compute the candidate v -subgraph RHSs without computing $\langle \mathfrak{A}|_U \rangle$ first. Once again, let us denote complete sets of v -subgraphs by their \sqsubseteq -maximal elements. Then the \sqsubseteq -maximal elements in the intersection $\mathcal{A} \cap \mathcal{B}$ of two complete sets of v -subgraphs \mathcal{A}, \mathcal{B} where

$$\begin{aligned}\sqsubseteq\text{-max}(\mathcal{A}) &= \{a_1, \dots, a_n\} \\ \sqsubseteq\text{-max}(\mathcal{B}) &= \{b_1, \dots, b_m\}\end{aligned}$$

can be computed as

$$\mathcal{A} \cap \mathcal{B} = \sqsubseteq\text{-max}(\{a_i \cap b_j \mid a_i \in \mathcal{A}, b_j \in \mathcal{B}\})$$

Thus the \sqsubseteq -maximal elements of the sets in $\langle \mathfrak{A}|_U \rangle$ can all be obtained as the intersection of \sqsubseteq -maximal elements of sets in $\mathfrak{A}|_U$:

$$\mathbf{max}(\langle \mathfrak{A}|_U \rangle) \subseteq \langle \mathbf{max}(\mathfrak{A}|_U) \rangle$$

However, inclusion in the opposite direction does not hold. As shown in Example 3.53, $A = AB \cap ACE$ lies in $\langle \mathbf{max}(\mathfrak{A}|_U) \rangle$ but not in $\mathbf{max}(\langle \mathfrak{A}|_U \rangle)$.

Elements of $\mathit{candRHS}_{T'}(v)|_U$ are e-closed, therefore

$$\mathit{candRHS}_{T'}(v)|_U = \{Y \sqsubseteq U \mid Y \in \langle \mathbf{max}(\mathfrak{A}|_U) \rangle \text{ and } Y \text{ is e-closed}\}$$

In other words, the property of being e-closed can be used to distinguish those elements in $\langle \mathbf{max}(\mathfrak{A}|_U) \rangle - \mathbf{max}(\langle \mathfrak{A}|_U \rangle)$. During the computation of $\mathbf{max}(\langle \mathfrak{A}|_U \rangle)$, the test of whether newly generated elements are e-closed can be done as follows:

Lemma 3.59. *An essential v -subgraph $Y \sqsubseteq U$ is e-closed if and only if Y is \sqsubseteq -maximal in*

$$\bigcap \{\mathcal{A} \in \mathfrak{A}|_U \mid Y \in \mathcal{A}\}$$

Proof. Follows immediately from Lemmas 3.51 and 3.54. □

Elements which are not e-closed can be safely discarded at the end, once $\langle \mathbf{max}(\mathfrak{A}|_U) \rangle$ have been computed. However, it would be better to be able to discard non-e-closed sets immediately, so that we do not need to intersect them with further elements of $\mathbf{max}(\mathfrak{A}|_U)$. The benefit of this can be substantial, as $\langle \mathbf{max}(\mathfrak{A}|_U) \rangle$ can be much bigger than $\mathbf{max}(\langle \mathfrak{A}|_U \rangle)$.

Example 3.60. Let $U = a_1 \dots a_n$ and

$$\mathfrak{A}|_U = \{\{U - a_1, U - a_2, \dots, U - a_n\}\}$$

consist of a single agree set containing all subgraphs of size $n - 1$ which are contained in U . Then

$$\langle \mathbf{max}(\mathfrak{A}|_U) \rangle = \mathcal{P}(U)$$

contains 2^n elements, but only $n + 1$ of them are e-closed (the n \sqsubseteq -maximal elements of the agree set plus U itself). □

Note that in the previous example we obtained the superfluous non-e-closed sets only by intersecting elements from the same agree set. Computing such intersections is unnecessary, and could easily be avoided with some extra effort. However, even if we consider only the intersections of \sqsubseteq -maximal elements from different agree sets, we can obtain an exponential number of such intersections, while the number of e-closed sets is only polynomial.

Example 3.61. Let $U = a_1 \dots a_n b$, and let

$$U_2 := \{X \subset U \mid |X| = |U| - 2\}$$

contain all subsets of U missing exactly two walks contained in U . We split U_2 into those sets containing b , and those not containing b :

$$\begin{aligned} \mathcal{B} &= \{X \in U_2 \mid b \in X\} \\ \mathcal{B}' &= \{X \in U_2 \mid b \notin X\} \end{aligned}$$

Note that \mathcal{B}' is exactly the agree set from our last example. We now assume our agree sets

$$\mathfrak{A}|_U = \{\{X\} \cup \mathcal{B} \mid X \in \mathcal{B}'\}$$

Then all n elements in \mathcal{B}' lie in different agree sets, and we can obtain 2^n different intersections from them. On the other hand, the only e-closed sets w.r.t. $\mathfrak{A}|_U$ are those in U_2 plus U itself, which are only $O(n^2)$ many. \square

Discarding non-e-closed sets immediately raises the question of whether we might “lose” some e-closed sets, which we could otherwise have obtained by intersecting non-e-closed sets with elements of $\mathbf{max}(\mathfrak{A}|_U)$. In the following we will show that no matter which order the elements of $\mathbf{max}(\mathfrak{A}|_U)$ are arranged, every element in $\mathbf{max}(\langle \mathfrak{A}|_U \rangle)$ is generated by taking the intersection of two e-closed elements. As a consequence, we can discard elements in $\langle \mathbf{max}(\mathfrak{A}|_U) \rangle$ immediately if they turn out not to be e-closed, since they are not needed for generating elements in $\mathbf{max}(\langle \mathfrak{A}|_U \rangle)$.

Lemma 3.62. *For a family \mathfrak{A} of complete sets with $X \in \mathbf{max}(\langle \mathfrak{A} \rangle)$. Let \mathfrak{A}_X be defined as*

$$\mathfrak{A}_X = \{\mathcal{B}_1, \dots, \mathcal{B}_n\} = \{\{Y \in \mathcal{A} \mid X \subseteq Y\} \mid \mathcal{A} \in \mathfrak{A}\} - \{\{\}\}.$$

Then $X \in \mathbf{max}(\langle \mathfrak{A}_X \rangle) \subseteq \mathbf{max}(\langle \mathfrak{A} \rangle)$, and for any Z_1, \dots, Z_n with $Z_i \in \mathcal{B}_i$ we have

$$X = Z_1 \cap \dots \cap Z_n$$

Proof. (1) Since $X \in \mathbf{max}(\langle \mathfrak{A} \rangle)$ we have

$$X = Y_1 \cap \dots \cap Y_k$$

for some \sqsubseteq -maximal $Y_i \in \mathcal{A}_i \in \mathfrak{A}$ for all $i = 1, \dots, k$. Thus X is \sqsubseteq -maximal in $\mathcal{A}_1 \sqcap \dots \sqcap \mathcal{A}_k$. Since $X \subseteq Y_i$ for all $1 \leq i \leq k$, the Y_i do not get removed when constructing \mathfrak{A}_X . That is, there exist $\mathcal{A}'_1, \dots, \mathcal{A}'_k \in \mathfrak{A}_X$ with

$$Y_i \in \mathcal{A}'_i \subseteq \mathcal{A}_i \text{ for all } 1 \leq i \leq k$$

Thus X is a \sqsubseteq -maximal element in $\mathcal{A}'_1 \sqcap \dots \sqcap \mathcal{A}'_k$ which shows $X \in \mathbf{max}(\langle \mathcal{A}_X \rangle)$.

(2) To prove $\mathbf{max}(\langle \mathcal{A}_X \rangle) \subseteq \mathbf{max}(\langle \mathcal{A} \rangle)$ let $X' \in \mathbf{max}(\langle \mathcal{A}_X \rangle)$. So X' is \sqsubseteq -maximal in

$$\mathcal{A}'_1 \sqcap \dots \sqcap \mathcal{A}'_k$$

for some $\mathcal{A}'_1, \dots, \mathcal{A}'_k \in \mathcal{A}_X$. Let $\mathcal{A}_1, \dots, \mathcal{A}_k \in \mathcal{A}$ be the corresponding sets in \mathcal{A} , that is $\mathcal{A}'_i = \{Y \in \mathcal{A}_i \mid X \subseteq Y\}$ for all $i = 1, \dots, k$. Then X' is in

$$\mathcal{A}_1 \sqcap \dots \sqcap \mathcal{A}_k$$

Since every subgraphs in \mathcal{A}_X contains X , every intersection of some subset of \mathcal{A}_X contains X , i.e., X' contains X . Since \mathcal{A}'_i and \mathcal{A}_i differ only by elements which do not contain X , which thus also do not contain the superset X' either, X' is \sqsubseteq -maximal in

$$\mathcal{A}_1 \sqcap \dots \sqcap \mathcal{A}_k$$

This shows $X' \in \mathbf{max}(\langle \mathcal{A} \rangle)$.

(3) Now consider Z_1, \dots, Z_n as above. Since $X \subseteq Z_i$ for all Z_i , we have

$$X \subseteq Z_1 \sqcap \dots \sqcap Z_n$$

Furthermore, there exists some $X' \in \mathcal{B}_1 \sqcap \dots \sqcap \mathcal{B}_n$ with

$$Z_1 \sqcap \dots \sqcap Z_n \subseteq X'$$

Since X is \sqsubseteq -maximal in the intersection of some of the \mathcal{B}_i due to $X \in \mathbf{max}(\langle \mathcal{A}_X \rangle)$, it is also maximal in the intersection of all of them (provided it lies in this intersection). Thus $X = X'$ which completes the proof. \square

Lemma 3.63. *Let $\mathcal{A}' \subseteq \mathcal{A}$ be two families of complete sets. Then $\mathbf{max}(\langle \mathcal{A}' \rangle) \subseteq \mathbf{max}(\langle \mathcal{A} \rangle)$.*

Proof. Clear by definition. \square

The next lemma implies that Algorithm 3.9 works correctly, that when computing $\langle \mathbf{max}(\mathcal{A}|_U) \rangle$ we can consider the elements of $\mathbf{max}(\mathcal{A}|_U)$ in any order and elements which are not e-closed can be discarded immediately.

Lemma 3.64. *Let $\mathbf{max}(\mathcal{A}|_U) = \{Y_1, \dots, Y_n\}$ be ordered in any manner, with the indices describing this order. Let $X \in \mathbf{max}(\langle \mathcal{A}|_U \rangle)$. Then there exist a subsequence Y_{i_1}, \dots, Y_{i_k} with $1 \leq i_1 < \dots < i_k \leq n$ such that*

$$X = Y_{i_1} \sqcap \dots \sqcap Y_{i_k}$$

and for every $j \in 1, \dots, k$ we have

$$Y_{i_1} \sqcap \dots \sqcap Y_{i_j} \in \mathbf{max}(\langle \mathcal{A}|_U \rangle)$$

Proof. We will show this by induction on the number of agree sets $|(\mathfrak{A}|_U)|$. The lemma is trivial for $|(\mathfrak{A}|_U)| \leq 2$.

For $X = U$ we can simply take the empty subsequence. For every other $X \in \mathbf{max}(\langle \mathfrak{A}|_U \rangle)$ there exists at least some index i_j with $X \subseteq Y_{i_j}$. So let $i_k \leq n$ be the highest index such that $X \subseteq Y_{i_k}$. Now define $\mathfrak{A}' \subseteq \mathfrak{A}|_U$ as the family of agree sets which do not include Y_{i_k} :

$$\mathfrak{A}' = \{\mathcal{A} \in \mathfrak{A}|_U \mid \exists Y \in \mathcal{A}. Y_{i_k} \subseteq Y\}$$

Furthermore, let $(\mathfrak{A}')_X$ be defined as in Lemma 3.62:

$$(\mathfrak{A}')_X = \{\mathcal{B}_1, \dots, \mathcal{B}_n\} = \{\{Y \in \mathcal{A} \mid X \subseteq Y\} \mid \mathcal{A} \in \mathfrak{A}'\} - \{\{\}\}$$

By definition of \mathfrak{A}' , all agree sets which lie in $(\mathfrak{A}|_U)_X$ but not in $(\mathfrak{A}')_X$ include Y_{i_k} . Using Lemma 3.62 ($X = Z_1 \cap \dots \cap Z_n$), it follows that there exists a set $X' \in \mathbf{max}(\langle (\mathfrak{A}')_X \rangle)$ with $(X' = Z_1 \cap \dots \cap Z_k, Y_{i_k} = Z_{k+1} \cap \dots \cap Z_n)$

$$X = X' \cap Y_{i_k} \tag{2}$$

Again by Lemma 3.62, we have $\mathbf{max}(\langle (\mathfrak{A}')_X \rangle) \subseteq \mathbf{max}(\langle \mathfrak{A}' \rangle)$ and thus $X' \in \mathbf{max}(\langle \mathfrak{A}' \rangle)$.

Since $|\mathfrak{A}'| < |(\mathfrak{A}|_U)|$, we can now use the induction hypothesis for \mathfrak{A}' . Using the same ordering for $\mathbf{max}(\mathfrak{A}')$ as for $\mathbf{max}(\mathfrak{A}|_U)$, we obtain a subsequence $Y_{i_1}, \dots, Y_{i_{k-1}}$ with $i_1 < \dots < i_{k-1}$ such that

$$X' = Y_{i_1} \cap \dots \cap Y_{i_{k-1}}$$

and for every $j \in 1 \dots k-1$ we have

$$Y_{i_1} \cap \dots \cap Y_{i_j} \in \mathbf{max}(\langle \mathfrak{A}' \rangle)$$

Since all the Y_{i_j} contain X' , and by definition i_k is the largest index containing $X \subseteq X'$, we have $i_{k-1} < i_k$. Equation 2 now gives us

$$X = Y_{i_1} \cap \dots \cap Y_{i_{k-1}} \cap Y_{i_k}$$

Finally, we have $\mathbf{max}(\langle \mathfrak{A}' \rangle) \subseteq \mathbf{max}(\langle \mathfrak{A}|_U \rangle)$ by Lemma 3.63, which shows that for every $j \in 1 \dots k-1$ we have

$$Y_{i_1} \cap \dots \cap Y_{i_j} \in \mathbf{max}(\langle \mathfrak{A}|_U \rangle)$$

and completes the proof. \square

Algorithm 3.9 encapsulates our discussion above, and outlines the process for determining all candidate subgraph RHSs (except the empty v -subgraph) from the family of all v -agree sets of T' . The correctness of the algorithm is supported by the previous lemma. The idea is to consider the \sqsubseteq -maximal elements of the v -agree sets, one at a time in any order. Each element is a candidate subgraph RHSs and any non-empty e-closed intersection with other candidate subgraph RHSs that we have already found is also a candidate subgraph RHS. Naturally, the intersection of two \sqsubseteq -maximal subgraphs can result in the

Algorithm 3.9 find_subgraphRHSs**Input:** $\text{rep}(\mathfrak{A}|_U)$ /* where $\text{rep}(\mathfrak{A}|_U) := \{\sqsubseteq\text{-max}(\mathcal{A}) \mid \mathcal{A} \in \mathfrak{A}|_U\}$ */**Output:** $\text{candRHS}_{T'}(v)|_U - \{\emptyset\}$ /* where $\text{candRHS}_{T'}(v)|_U = \text{max}(\langle \mathfrak{A}|_U \rangle)$ */

1. $\text{max}(\mathfrak{A}|_U) := \bigcup_{\mathcal{A} \in \mathfrak{A}|_U} \sqsubseteq\text{-max}(\mathcal{A})$
2. $\text{baseRHS} := \text{max}(\mathfrak{A}|_U)$
3. $\text{candRHS} := \{\}$
4. **for all** $X \in \text{baseRHS}$ **do**
5. $\text{newRHS} := \{X\}$
6. **for all** $Y \in \text{candRHS} - \{X\}$ **do**
7. $Z := X \cap Y$
8. **if** $Z \neq \emptyset$ and $Z \notin \text{candRHS} \cup \text{newRHS}$ **then**
9. **if** $\text{is_eclosed}(Z, \text{rep}(\mathfrak{A}|_U))$ **then**
10. $\text{newRHS} := \text{newRHS} \cup \{Z\}$
11. **end if**
12. **end if**
13. **end for**
14. $\text{candRHS} := \text{candRHS} \cup \text{newRHS}$
15. **end for**
16. **return** $\text{candRHS} \cup \{U\}$
17. **proc** $\text{is_eclosed}(Z, \text{rep}(\mathfrak{A}|_U))$
18. **return** Z is \sqsubseteq -maximal in $\bigcap \{\mathcal{A} \in \mathfrak{A}|_U \mid Z \in \mathcal{A}\}$

empty v -subgraph. However, Algorithm 3.9 safely disregard the empty v -subgraph as element of $\text{candRHS}_{T'}(v)$, without checking whether it is e-closed because every pXFD with $\{\emptyset\}$ as the RHS is trivial. Also of note, is that we do not consider the special case $\sqsubseteq\text{-max}(\bigcap\{\}) = \{U\}$ (i.e., $U \in \text{candRHS}_{T'}(v)|_U$) until all other candidate subgraph RHSs have been identified. This is because U intersects with all v -subgraphs contained in U but do not generate any additional \sqsubseteq -maximal elements. Recognising U as a candidate subgraph RHS earlier would only necessitate superfluous comparisons. The next pair of examples, illustrate our proposed method for finding candidate v -subgraph RHSs.

Example 3.65. The candidate RHSs in Example 3.57 can alternatively be found through Algorithm 3.9. Firstly, we have

$$\begin{aligned} \text{max}(\mathfrak{A}|_U) &= \{AB, ACD\} \cup \{ACE, BE\} \cup \{AC, BC\} \\ &= \{AB, ACD, ACE, BE, AC, BC\} \end{aligned}$$

Let us consider the members of $\text{baseRHS} = \text{max}(\mathfrak{A}|_U)$ in the order they are listed above. The first v -subgraph AB simply gives $\text{candRHS} = \{AB\}$.

Next consider ACD . The intersection $ACD \cap AB = A$ is a possible candidate v -subgraph RHS. To check whether A is e-closed, we need to intersect all v -agree sets in $\mathfrak{A}|_U$ which contains A . In this case, we intersect all members of $\mathfrak{A}|_U$. This yields the set $\{AC, A, C, B\}$ of which A is not \sqsubseteq -maximal. Therefore A is not e-closed and the v -subgraph ACD only gives us itself as a new candidate v -subgraph RHS, i.e., currently $candRHS = \{AB, ACD\}$.

Next consider BE . We can generate at most two new candidate RHS by performing intersections. Firstly we check whether the intersection $BE \cap AB = B$ is e-closed. Like before this involve looking at the intersection of all members of $\mathfrak{A}|_U$. This time, we find B is \sqsubseteq -maximal and therefore B is added as a new candidate v -subgraph RHS, along with BE . Secondly we have the intersection $BE \cap ACD$. But this gives the empty set and no further candidate v -subgraph RHS is generated. So now, $candRHS = \{AB, ACD, BE, B\}$.

After considering the last two base RHS AC and BC , we get the set

$$candRHS = \{AB, ACD, BE, B, AC, BC\}.$$

Note that C is also not \sqsubseteq -maximal in the intersection of all members of $\mathfrak{A}|_U$. The final candidate v -subgraph RHS to be added is the unit $U = ABCDE$ itself. Altogether, we obtain the same result set as Example 3.57. \square

Example 3.66. We find the following $v_{Purchase}$ -agree sets of $T'_{purchase}$:

$$\begin{aligned} \mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase}) &= \{\text{Da, Ti, De, Pr, } v_{Outlet}\} \\ \mathcal{A}_{\{i_5, i_{31}\}}(v_{Purchase}) &= \{\text{Da, Di, Sa, } v_{Root}\} \\ \mathcal{A}_{\{i_5, i_{44}\}}(v_{Purchase}) &= \{\text{Da, De, } v_{Root}\} = \mathcal{A}_{\{i_{18}, i_{44}\}}(v_{Purchase}) \\ \mathcal{A}_{\{i_{18}, i_{31}\}}(v_{Purchase}) &= \{\text{Da, } v_{Root}\} \\ \mathcal{A}_{\{i_{31}, i_{44}\}}(v_{Purchase}) &= \{\text{Da, Ti, Pr, } v_{Outlet}\} \end{aligned}$$

See Example 3.75 for one possible computation of these $v_{Purchase}$ -agree sets.

There are five $v_{Purchase}$ -units, these are: $\text{Da, Ti, Pn, } \{\text{De, Pr, Di}\}$ and Sa . Without any computation, we know the four singleton $v_{Purchase}$ -units gives only themselves as candidate subgraph RHSs. Projecting the $v_{Purchase}$ -agree sets to the final $v_{Purchase}$ -unit $\{\text{De, Pr, Di}\}$ yield the following \sqsubseteq -maximal subgraphs for $baseRHS$: De, Pr, Di . Since the intersection of a walk with any subgraph is simply a walk, and all the $v_{Purchase}$ -subgraphs $baseRHS$ are in fact walks, it easily follows that we obtain four further candidate subgraph RHSs for the $v_{Purchase}$ -unit $\{\text{De, Pr, Di}\}$, these are the three walks in $baseRHS$ and the $v_{Purchase}$ -unit itself.

In summary, the candidate subgraph RHSs for $T'_{purchase}$ are:

$$\text{Da, Ti, Pn, Sa, } \{\text{De, Pr, Di}\}, \text{De, Pr, and Di.}$$

\square

This brings us to the question of how to apply the transversal approach for extracting pXFDs belonging to $\mathfrak{S}\mathfrak{C}_{T'}(v)$. Analogous to the case of candidate v -ancestor RHSs, candidate subgraph RHSs should be examined in a particular order so that we can exploit

re-use of minimal transversal computations. Once again, by interleaving the application of the transversal approach within minimal transversal computations, we can also reduce the time it takes to discover the first pXFD. However, the situation is not identical as for candidate v -ancestor RHSs since candidate subgraph RHSs are unlikely to be linearly ordered.

For each v -unit U , we have a poset $(candRHS_{T'}(v)|_U - \{\emptyset\}, \sqsubseteq)$ in which the \sqsubseteq -maximal v -subgraph is U and there are potentially many \sqsubseteq -minimal v -subgraphs. To discover all pXFDs whose RHS is p-subsumed by $\{U\}$, we require as many minimal transversal computations for the family $\mathfrak{D}_{T'}^U(v)$ as the number of complete chains in the poset $(candRHS_{T'}(v)|_U - \{\emptyset\}, \sqsubseteq)$. This is in contrast to the worst-case where we require as many minimal transversal computations as the number of candidate subgraph RHSs contained in U . For example, in a simple approach where we consider \sqsubseteq -maximal candidate subgraph RHSs before \sqsubseteq -smaller ones - we can check whether to include extracted pXFDs as soon as the minimal transversals for a single family of v -difference sets is computed, but each computation is started from scratch.

Since we are only considering candidate subgraph RHSs, the check for triviality reduces to

$$\mathcal{X} \in \mathfrak{Tr}(\mathfrak{D}_{T'}^Y(v)) - \{\{Y\}\} \text{ and } Y \in candRHS_{T'}(v)|_U - \{\emptyset\} \text{ for some } v\text{-unit } U.$$

There is also no need to check for right-maximality of extracted pXFDs whose RHS contains a single \sqsubseteq -maximal candidate subgraph RHS (i.e., the v -units of T'). When we do check for right-maximality, then we need not consider all essential v -subgraphs which strictly p-subsumes Y , but rather only those candidate subgraph RHS $Z \sqsupset Y$ such that there are no other candidate subgraph RHS Z' with $Z \sqsupset Z' \sqsupset Y$. In contrast to the case of candidate v -ancestor RHSs, there are potentially more than one candidate subgraph RHS Z which minimally strictly p-subsumes Y . For each candidate subgraph RHS Y

$$\mathcal{Y}' = \{Z \in candRHS_{T'}(v)|_U \mid Z \sqsupset Y \text{ and} \\ \nexists Z' \in candRHS_{T'}(v)|_U \text{ such that } Z \sqsupset Z' \sqsupset Y\}$$

is the set of candidate subgraph RHSs which minimally strictly p-subsumes Y . After finding the minimal transversals of $\mathfrak{D}_{T'}^Z(v)$ for all v -subgraph $Z \in \mathcal{Y}' \cup \{Y\}$ we can extract all interesting pXFDs with $\{Y\}$ as the RHS. This speeds up the time to discover pXFDs in comparison with the alternative approach where we first complete a minimal transversal computation guided by some chain in the poset. Also note that, the minimal transversals for each $\mathfrak{D}_{T'}^Z(v)$ with $Z \in \mathcal{Y}'$ are extensions of the minimal transversal computation of $\mathfrak{D}_{T'}^Y(v)$ since $\mathfrak{D}_{T'}^Z(v) \supset \mathfrak{D}_{T'}^Y(v)$.

In summary, we propose the following process for extracting the subset $\mathfrak{S}\mathfrak{C}_{T'}(v)$ of the canonical pXFD-cover. We consider the candidate subgraph RHSs in every poset $(candRHS_{T'}(v)|_U - \{\emptyset\}, \sqsubseteq)$ in a *bottom-up depth-first* manner. For each candidate subgraph RHS Y under examination, we compute the minimal transversals of $\mathfrak{D}_{T'}^Y(v)$, find the related set \mathcal{Y}' as described above and compute the minimal transversals of $\mathfrak{D}_{T'}^Z(v)$ for all v -subgraph in $Z \in \mathcal{Y}'$. We extract all those pXFD with $\{Y\}$ as the RHS and

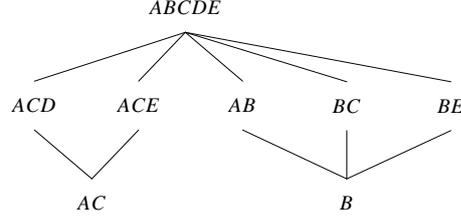


Figure 3.5: Hasse diagram for the poset $(candRHS_{T'}(v)|_U - \{\emptyset\}, \subseteq)$ from Example 3.67.

check whether they can be added to $\check{\mathcal{C}}_{T'}(v)$. In a similar manner, we then examine each candidate subgraph RHSs in \mathcal{Y}' . Chains sharing the same \subseteq -minimal v -subgraph are considered together but every candidate subgraph RHSs in each chain is examined before considering the next chain. The process is summarised in Algorithm 3.10. Next we present an example to highlight the depth-first nature of the procedure.

Example 3.67. Reconsider

$$candRHS_{T'}(v)|_U = \{AB, ACD, ACE, BE, AC, BC, B, ABCDE\}$$

from Example 3.53. The elements of $candRHS_{T'}(v)|_U - \{\emptyset\}$ form the poset depicted in Figure 3.5. Algorithm 3.10 starts with either candidate AC or B . Suppose we choose AC .

- From candidate RHS $Y = AC$ we obtain $\mathcal{Y}' = \{ACD, ACE\}$. We extract all appropriate pXFDs with $\{AC\}$ as the RHS by computing the minimal transversals for $\mathfrak{D}_{T'}^{AC}(v)$ and then the minimal transversals for $\mathfrak{D}_{T'}^{ACD}(v)$ and for $\mathfrak{D}_{T'}^{ACE}(v)$. The minimal transversal computation for $\mathfrak{D}_{T'}^{ACD}(v)$ and for $\mathfrak{D}_{T'}^{ACE}(v)$ make use of the knowledge about the minimal transversals for $\mathfrak{D}_{T'}^{AC}(v)$.
- Next, the algorithm will consider, in any order ACD and ACE as candidate. The minimal transversals for $\mathcal{D}_{ACD}(v)$ and $\mathcal{D}_{ACE}(v)$ computed previously are carried forward in the next steps.
 - From $Y = ACD$ we obtain $\mathcal{Y}' = \{ABCDE\}$. Therefore to find all appropriate pXFDs with $\{ACD\}$ as the RHS, we find the minimal transversals for $\mathfrak{D}_{T'}^{ABCDE}(v)$ using the minimal transversals for $\mathfrak{D}_{T'}^{ACD}(v)$ as the starting point.
 - * From $Y = ABCDE$ we obtain $\mathcal{Y}' = \{\}$. Thus, we add the pXFD $v : \mathcal{X} \rightarrow \{ABCDE\}$ to the canonical pXFD-cover, where \mathcal{X} is a minimal transversal for $\mathfrak{D}_{T'}^{ABCDE}(v)$ other than $\{ABCDE\}$.
 - Then, we back-track to consider $Y = ACE$ with $\mathcal{Y}' = \{ABCDE\}$. We find all appropriate pXFDs with $\{ACE\}$ as the RHS by re-computing the minimal transversals for $\mathfrak{D}_{T'}^{ABCDE}(v)$ using knowledge about the minimal transversals for $\mathfrak{D}_{T'}^{ACE}(v)$.
- Since $ABCDE$ has already been considered as the RHS, we next consider $Y = B$ and proceeds in a similar manner.

Hence, one possible order in which $\text{candRHS}_{T'}(v)|_U$ is processed by Algorithm 3.10 is: $AC, ACD, ABCDE, ACE, B, AB, BC, BE$. \square

In the previous example, although $ABCDE$ is considered as the RHS only once, Algorithm 3.10 re-computes the minimal transversals for $\mathfrak{D}_{T'}^{ABCDE}(v)$ when processing ACE, AB, BC, BE as the RHS. This is because $ABCDE$ belongs to more than one chain in the poset and therefore the minimal transversals of $\mathfrak{D}_{T'}^{ABCDE}(v)$ are needed to determine the right-maximality of pXFDs with $\{ACE\}, \{AB\}, \{BC\}$ or $\{BE\}$ as the RHS. The main heuristic behind Algorithm 3.10 support re-use of minimal transversal computation within one chain whilst keeping track of as few minimal transversal computations as possible. So, despite $\mathfrak{D}_{T'}^{ABCDE}(v)$ being needed multiple times, the algorithm reduces the effort required for computing $\mathfrak{D}_{T'}^{ABCDE}(v)$ each time, rather than recognising that the entire minimal transversal computation for $\mathfrak{D}_{T'}^{ABCDE}(v)$ can be re-used.

It is simple to keep track of the minimal transversal for $\mathfrak{D}_{T'}^U(v)$ whenever we detect multiple chains in the poset. We can even instate condition - whether the candidate subgraph RHS p-subsumes some other candidate subgraph RHS still to be considered - for determining other families of minimal transversals for which we can re-use entire minimal transversal computations. The trade-off faced lies, of course, between the effort required to keep track of the minimal transversal computations and the effort required to re-compute minimal transversals partially.

3.3 Finding Agree Sets

In this section we focus on the problem of computing the family $\mathfrak{A}_{T'}(v)$ of v -agree sets for a given T -compatible data tree T' . Actually, from the discussion so far, we are mainly interested in finding the \sqsubseteq -maximal v -properties of each v -agree set of T' . Thus the problem we really address here is how to compute

$$\text{rep}(\mathfrak{A}_{T'}(v)) = \{\sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)) \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}.$$

Since essential v -ancestors are incomparable with essential v -subgraphs, we find the \sqsubseteq -maximal essential v -ancestors and \sqsubseteq -maximal essential v -properties belonging some v -agree set $\mathcal{A}_{\{p_1, p_2\}}(v)$ independently. Let

- $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}} = \{X \in \check{\mathbb{A}}_T(v) \mid X \in \mathcal{A}_{\{p_1, p_2\}}(v)\}$
be the *projection of v -agree set $\mathcal{A}_{\{p_1, p_2\}}(v)$ to v -ancestors* containing all essential v -ancestors on which p_1, p_2 agree, and
- $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{S}}} = \{X \in \check{\mathbb{S}}_T(v) \mid X \in \mathcal{A}_{\{p_1, p_2\}}(v)\}$
be the *projection of v -agree set $\mathcal{A}_{\{p_1, p_2\}}(v)$ to v -subgraphs* containing all essential v -subgraphs on which p_1, p_2 agree.

Then $\sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)) = \sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}}) \cup \sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{S}}})$. And similarly, we can project the difference sets to v -ancestors and to v -subgraphs

3.3.1 Finding \sqsubseteq -maximal Essential v -Ancestors in Agree Sets

Finding the \sqsubseteq -maximal essential v -ancestors of all v -agree sets is straightforward.

We are looking for a singleton set of v -ancestors which always exists. Any two pre-image trees of v must agree on the root and therefore every v -agree set of T' must contain some essential v -ancestor. Furthermore, essential v -ancestors are linearly ordered with respect to p-subsumption, and so every v -agree set of T' has exactly one \sqsubseteq -maximal essential v -ancestor.

Our approach considers the essential v -ancestors n from the \sqsubseteq -maximal to \sqsubseteq -minimal. For each essential v -ancestor n , we identify all pairs of pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $\{n\} = \sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}})$. This determination is aided by the observation that n is \sqsubseteq -maximal in $\mathcal{A}_{\{p_1, p_2\}}(v)$ if and only if $p_1|_n \doteq p_2|_n$ and $p_1|_m \neq p_2|_m$ where m is the immediate essential descendant of n . This process is applied in Algorithm 3.11 and results in

$$\check{\mathfrak{A}}_{T'}(v) = \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}} \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}$$

and without extra effort we also gain

$$\check{\mathfrak{D}}_{T'}((v)) = \{\mathcal{D}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}} \mid \mathcal{D}_{\{p_1, p_2\}}(v) \in \mathfrak{D}_{T'}(v)\}.$$

We use a simple partition of $P_{T'}(v)$ for verifying whether two pre-image trees agree/differ on a given essential v -ancestor. Furthermore, this requires only a single depth-first traversal of the given T -compatible data tree T' . During our traversal of the data tree, we assign to each pre-image of some essential v -ancestor $\phi(e^i)$ a unique identifier e^i for $i = 1, \dots, k$ such that we can compose for each pre-image tree $p \in P_{T'}(v)$ the *identifier path*

$$s_k(p) = e_p^1 / \dots / e_p^k.$$

where e_p^1 is the unique identifier of the root node, e_p^k is the unique identifier of the pre-image of v in p and every $\phi(e_p^i)$ (for $i = 1, \dots, k-1$) is the immediate essential ancestor of $\phi(e_p^{i+1})$.

Let $\mathcal{S}_k = \{(p, s_k(p)) \mid p \in P_{T'}(v)\}$ be the set of all pre-image trees of v in T' together with their associated identifier path. Of course, the identifier path of each element in \mathcal{S}_k is distinct and all of them have the same length (i.e., the same number of nodes $k = |\mathbf{E}_{T'}^{\check{\mathbb{A}}}(v)|$). The identifier paths induces a partition on $P_{T'}(v)$ and provides information on whether or not two pre-image trees agree/diff on their pre-image of $n \in \vartheta(\{v\})$. Similarly, if we consider the root subpath of these identifier paths, we induce partitions of $P_{T'}(v)$ relating to whether two pre-images trees agree on every essential v -ancestors.

An SQL database can be used to implement the partitioning. We store each \mathcal{S}_i as a relation r_{S_i} with two fields: “@pre-image” for recording the pre-image tree identifier of p ; and “@path” for recording the rooted path $s_i(p)$. Then each pre-image tree in $P_{T'}(v)$ is represented by a single row in r_{S_i} and the partitioning of \mathcal{S}_i can be achieved by a simple GROUP BY query over r_{S_i} :

```
SELECT @pre-image FROM r_{S_i} GROUP BY @path;
```

Algorithm 3.11 find_agreeSets-projectedToAncestors

Input: \mathcal{S}_k /* where $\mathcal{S}_k = \{(p, s_k(p)) \mid s_k(p) \text{ is the identifier path for } p \in P_{T'}(v)\}$ */**Output:** $\overset{\check{\Delta}}{\mathcal{A}}_{T'}(v), \overset{\check{\Delta}}{\mathcal{D}}_{T'}(v)$ /* where $\overset{\check{\Delta}}{\mathcal{A}}_{T'}(v) := \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\Delta}} \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathcal{A}_{T'}(v)\}$
 $\overset{\check{\Delta}}{\mathcal{D}}_{T'}(v) := \{\mathcal{D}_{\{p_1, p_2\}}(v)|_{\check{\Delta}} \mid \mathcal{D}_{\{p_1, p_2\}}(v) \in \mathcal{D}_{T'}(v)\}$ */

1. $\mathcal{A} := \{\}$
 2. $\mathcal{D} := \{\}$
 3. $i := k$
 4. $\pi_i := \{\{(p, s_i(p)) \mid (p, s_i(p)) \in \mathcal{S}_i\}$ // Partition \mathcal{S}_i
 5. **while** $i > 0$ **do**
 6. $\mathcal{S}_{i-1} := \{(p, s_{i-1}(p)) \mid (p, s_i(p)) \in \mathcal{S}_k \text{ and } s_{i-1}(p) \text{ is the rooted prefix path of } s_i(p) \text{ of length } i - 1\}$
 7. $\pi_{i-1} :=$ Partition \mathcal{S}_{i-1} such that two elements belong to the same partition class if and only if they have the same path
 8. **for all** $c \in \pi_{i-1}$ **do**
 9. **for all** $\{(p_1, s_{i-1}(p_1)), (p_2, s_{i-1}(p_2))\} \subseteq c$ such that $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\Delta}} \notin \mathcal{A}$ **do**
 10. **if** $\exists c' \in \pi_i$ such that $\{(p_1, s_i(p_1)), (p_2, s_i(p_2))\} \subseteq c'$ **then**
 11. $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\Delta}} := \{name(e_{p_1}^{i-1})\}$
 12. $\mathcal{A} := \mathcal{A} \cup \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\Delta}}\}$
 13. $\mathcal{D}_{\{p_1, p_2\}}(v)|_{\check{\Delta}} := \{name(e_{p_1}^i)\}$
 14. $\mathcal{D} := \mathcal{D} \cup \{\mathcal{D}_{\{p_1, p_2\}}(v)|_{\check{\Delta}}\}$
 15. **end if**
 16. **end for**
 17. **end for**
 18. $\mathcal{S}_k := \mathcal{S}_{i-1}$
 19. $i = i - 1$
 20. **end while**
 21. **return** \mathcal{A}, \mathcal{D}
-

The relation $r_{\mathcal{S}_{i-1}}$ can be constructed from $r_{\mathcal{S}_i}$ by trimming the $@path$ value from the right.

Example 3.68. Consider the purchase schema tree $T_{Purchase}$ (Figure 3.2) and its compatible data tree $T'_{purchase}$ (Figure 3.1). The essential $v_{Purchase}$ -ancestors of $T'_{purchase}$ are:

$$e^1 = v_{Root}, e^2 = v_{Outlet}, e^3 = v_{Purchase}.$$

There are four pre-image trees of $v_{Purchase}$ with

$$\mathcal{S}_3 = \{ (i_5, i_1/i_2/i_5), (i_{18}, i_1/i_2/i_{18}), (i_{31}, i_1/i_{28}/i_{31}), (i_{44}, i_1/i_{28}/i_{44}) \}$$

Recall that the partition classes of \mathcal{S}_3 are the singleton subsets of \mathcal{S}_3 . Then

$$\mathcal{S}_2 = \{ (i_5, i_1/i_2), (i_{18}, i_1/i_2), (i_{31}, i_1/i_{28}), (i_{44}, i_1/i_{28}) \}$$

which is partitioned into two:

$$\begin{aligned} c_1^2 &= \{ (i_5, i_1/i_2), (i_{18}, i_1/i_2) \} \\ c_2^2 &= \{ (i_{31}, i_1/i_{28}), (i_{44}, i_1/i_{28}) \} \end{aligned}$$

The $v_{Purchase}$ -ancestor with rooted path of length 2 is v_{Outlet} . Therefore, from c_1^2 and c_2^2 , we find

$$\begin{aligned} v_{Outlet} &\in \sqsubseteq\text{-max}(\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase})) & v_{Purchase} &\in \sqsubseteq\text{-min}(\mathcal{D}_{\{i_5, i_{18}\}}(v_{Purchase})) \\ v_{Outlet} &\in \sqsubseteq\text{-max}(\mathcal{A}_{\{i_{31}, i_{44}\}}(v_{Purchase})) & v_{Purchase} &\in \sqsubseteq\text{-min}(\mathcal{D}_{\{i_{31}, i_{44}\}}(v_{Purchase})) \end{aligned}$$

Next, we have

$$\mathcal{S}_1 = \{ (i_5, i_1), (i_{18}, i_1), (i_{31}, i_1), (i_{44}, i_1) \}$$

where every element belongs to the same partition class. This gives us the \sqsubseteq -maximal essential v -ancestor for four further agree sets of $T'_{purchase}$:

$$\begin{aligned} v_{Root} &\in \sqsubseteq\text{-max}(\mathcal{A}_{\{i_5, i_{31}\}}(v_{Purchase})) & v_{Outlet} &\in \sqsubseteq\text{-max}(\mathcal{D}_{\{i_5, i_{31}\}}(v_{Purchase})) \\ v_{Root} &\in \sqsubseteq\text{-max}(\mathcal{A}_{\{i_5, i_{44}\}}(v_{Purchase})) & v_{Outlet} &\in \sqsubseteq\text{-max}(\mathcal{D}_{\{i_5, i_{44}\}}(v_{Purchase})) \\ v_{Root} &\in \sqsubseteq\text{-max}(\mathcal{A}_{\{i_{18}, i_{31}\}}(v_{Purchase})) & v_{Outlet} &\in \sqsubseteq\text{-max}(\mathcal{D}_{\{i_{18}, i_{31}\}}(v_{Purchase})) \\ v_{Root} &\in \sqsubseteq\text{-max}(\mathcal{A}_{\{i_{18}, i_{44}\}}(v_{Purchase})) & v_{Outlet} &\in \sqsubseteq\text{-max}(\mathcal{D}_{\{i_{18}, i_{44}\}}(v_{Purchase})) \end{aligned}$$

We have finished since the path length becomes 0. \square

3.3.2 Finding \sqsubseteq -maximal Essential v -Subgraphs in Agree Sets

The task of finding all \sqsubseteq -maximal essential v -subgraphs belonging to each v -agree set of T' is more challenging.

We will discuss two alternate approaches. On the one hand, we can consider every pair of pre-image trees and test their property-equality on each essential v -subgraph. On the other, we can generate partitions as auxiliary sources of information about property-equality of pre-image trees, and use partitions to assemble the agree sets. In the former (pair by pair) approach we can reduce the number of essential v -subgraphs which need to be considered, while in the latter (partition-based) approach we aim to reduce the frequency of potentially costly access to the data tree when finding agree sets.

Finding \sqsubseteq -maximal Essential v -Subgraphs: Pair by Pair

For each pair of distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ we must decide whether p_1, p_2 agree on each essential v -subgraph. This can be done in polynomial time in the size of T' (e.g., see [15, 90]).

Algorithm 3.12 `find_agreeSets-projectedToSubgraphs(Pairwise)`**Input:** Data tree T' **Output:** \mathfrak{A} /* where $\mathfrak{A} = \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\S} \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}$ */

1. $\mathfrak{A} := \{\}$
2. **for all** $p_1, p_2 \in P_{T'}(v)$ **do**
3. $\mathcal{D}_{\{p_1, p_2\}}(v)|_{\S}, \mathcal{A}_{\{p_1, p_2\}}(v)|_{\S} := \text{find_agreeSetOnePair}(p_1, p_2, \text{oracle}(\mathcal{A}_{\{p_1, p_2\}}(v)))$
4. $\mathfrak{A} := \mathfrak{A} \cup \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\S}\}$
5. **end for**
6. **return** \mathfrak{A}

Once again, it is natural to consider each v -unit U independently. All v -subgraphs contained in v -unit U are essential, and their number is exponential in the size of U . On the positive side, it is not always necessary to test every essential v -subgraph X . Recall our observations that v -difference sets are \sqsubseteq -upward-closed and v -agree sets are \sqsubseteq -downward-closed. If we know two pre-image trees $p_1, p_2 \in P_{T'}(v)$ differ on X , then there is no need to test whether they agree on an essential v -subgraph containing X . Likewise, if p_1, p_2 agree on X , we know they agree on all subgraphs contained in X .

This motivates a “bottom up” approach testing \sqsubseteq -smaller v -subgraphs contained in U first, and testing \sqsubseteq -larger v -subgraphs only if all v -subgraphs that it contains are known to be in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$. However, this is still inefficient if $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ contains large essential v -subgraphs, for example $\mathcal{A}_{\{p_1, p_2\}}(v)|_U = \{U\}$. Instead we can look at an approach outlined as follows:

- keep track of a set \mathcal{A}_{max} containing all maximal elements in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ found so far
- keep track of a set \mathcal{D}_{min} containing all minimal sets not included in some element of \mathcal{A}_{max}
- while some $X \in \mathcal{D}_{min}$ lies in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$, we find the \sqsubseteq -maximal $X' \sqsupseteq X$ with $X' \in \mathcal{A}_{\{p_1, p_2\}}(v)|_U$, add it to \mathcal{A}_{max} and update \mathcal{D}_{min}

The objective is that, when the algorithm terminates, \mathcal{D}_{min} contains all \sqsubseteq -minimal elements not in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ and \mathcal{A}_{max} contains all \sqsubseteq -maximal subgraph elements of $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$. For us, this means $\mathcal{D}_{min} = \sqsubseteq\text{-min}(\mathcal{D}_{\{p_1, p_2\}}(v)|_U)$ is given “for free”.

In Corollary 3.43 of Section 3.2.2 we established that

$$\sqsubseteq\text{-min}(\mathcal{D}_{\{p_1, p_2\}}(v)|_U) = \mathfrak{S}\mathfrak{S}(\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U})$$

where

$$\overline{\mathcal{A}_{\{p_1, p_2\}}(v)|_U} = \{U - Y \neq \emptyset \mid Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)\} \neq \{\}$$

and $\mathfrak{Hs}(E)$ denotes the family of all minimal hitting sets for the hypergraph $\mathcal{H} = (U, E)$. Thus

$$\mathcal{D}_{min} = \mathfrak{Hs}(\overline{\mathcal{A}_{max}})$$

where $\overline{\mathcal{A}_{max}} = \{U - Y \mid Y \in \mathcal{A}_{max}\}$, provided $U \notin \mathcal{A}_{max}$. Updating \mathcal{D}_{min} to \mathcal{D}'_{min} when a single new set X is added to \mathcal{A}_{max} is very simple:

$$\mathcal{D}'_{min} = \sqsubseteq_{-\min}(\{Y \cup \{B\} \mid Y \in \mathcal{D}_{min} \text{ and } B \in U - X\})$$

Before the algorithm terminates, \mathcal{D}_{min} may contain elements which belong to $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ and so we divide \mathcal{D}_{min} into \mathcal{D}_{final} and \mathcal{D}_{maybe} , where \mathcal{D}_{final} contains exactly those sets in \mathcal{D}_{min} which are known to *not* lie in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$. This gives us Algorithm 3.13.

For the time being, we pre-suppose that an oracle function is given which tells us whether two pre-image trees agree on a given v -subgraph or not. The boxes in Algorithm 3.13 indicate calls to the oracle. Potential improvement may be possible by testing each v -walk B contained in U first. This would allow us to remove B with $\{B\} \notin \mathcal{A}_{\{p_1, p_2\}}(v)|_U$ from U and only add them to \mathcal{D}_{final} afterwards provided $\mathcal{D}_{final} \neq \{\}$.

Proposition 3.69. *Algorithm 3.13 computes the \sqsubseteq -maximal essential v -subgraphs in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ and the \sqsubseteq -minimal essential v -subgraphs not in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ correctly, for every v -unit U .*

Proof. It is easy to see that the following invariants hold:

- $\mathcal{A}_{max} \subseteq \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$
- $\mathcal{D}_{final} \subseteq \mathcal{P}(U) - \mathcal{A}_{\{p_1, p_2\}}(v)|_U$
- $\mathfrak{Hs}(\overline{\mathcal{A}_{max}}) = \mathcal{D}_{final} \cup \mathcal{D}_{maybe}$

Additionally, when the algorithm terminates, we have $\mathcal{D}_{maybe} = \{\}$, and thus

$$\mathfrak{Hs}(\overline{\mathcal{A}_{max}}) = \mathcal{D}_{final}$$

Now assume that $Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$ does not lie in \mathcal{A}_{max} . Since Y is \sqsubseteq -maximal in $\mathcal{A}_{\{p_1, p_2\}}(v)|_U$ and $\mathcal{A}_{max} \subseteq \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$, Y is not a subset of any $Y' \in \mathcal{A}_{max}$. Thus Y is a transversal of \mathcal{A}_{max} and there exists a minimal hitting set $T \subseteq Y$ with

$$T \in \mathfrak{Hs}(\overline{\mathcal{A}_{max}}) = \mathcal{D}_{final} \subseteq \mathcal{P}(U) - \mathcal{A}_{\{p_1, p_2\}}(v)|_U$$

This leads to a contradiction of $Y \in \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$, since it shows $T \notin \mathcal{A}_{\{p_1, p_2\}}(v)|_U$ and therefore $Y \notin \mathcal{A}_{\{p_1, p_2\}}(v)|_U$. This gives us $\mathcal{A}_{max} \supseteq \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$, and thus

$$\mathcal{A}_{max} = \sqsubseteq_{-\max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$$

The second part of the theorem follows directly from $\mathfrak{Hs}(\overline{\mathcal{A}_{max}}) = \mathcal{D}_{final}$. □

Algorithm 3.13 find_agreeSetOnePair

Input: Pre-image trees $p_1, p_2 \in P_{T'}(v)$ and $\text{oracle}(\mathcal{A}_{\{p_1, p_2\}}(v))$ **Output:** $\sqsubseteq_{- \min}(\mathcal{D}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}})$ and $\sqsubseteq_{- \max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}})$

1. $\mathcal{D} := \{\}, \mathcal{A} := \{\}$
2. **for all** v -unit U in T' **do**
3. $\mathcal{D}_{final}, \mathcal{A}_{max} := \text{find_agreeSets-inUnit}(U, \text{oracle}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U))$
4. $\mathcal{D} := \mathcal{D} \cup \mathcal{D}_{final}, \mathcal{A} := \mathcal{A} \cup \mathcal{A}_{max}$
5. **end for**
6. **return** \mathcal{D}, \mathcal{A}

7. **proc** find_agreeSets-inUnit($U, \text{oracle}(\mathcal{A}_{\{p_1, p_2\}}(v)|_U)$)
8. $\mathcal{A}_{max} := \{\}, \mathcal{D}_{final} := \{\}, \mathcal{D}_{maybe} = \{\emptyset\}$
9. **while** $\mathcal{D}_{maybe} \neq \{\}$ **do**
10. $X := \text{Remove arbitrary element from } \mathcal{D}_{maybe}$
11. **if** $X \notin \mathcal{A}_{\{p_1, p_2\}}(v)|_U$ **then** // oracle call
12. Move X from \mathcal{D}_{maybe} to \mathcal{D}_{final}
13. **else**
14. /* Add \sqsubseteq -maximal $X' \supseteq X$ with $X' \in \mathcal{A}_{\{p_1, p_2\}}(v)|_U$ to \mathcal{A}_{max} */
15. **for all** $B \in U - X$ **do**
16. **if** $X \cup \{B\} \in \mathcal{A}_{\{p_1, p_2\}}(v)|_U$ **then** // oracle call
17. $X := X \cup \{B\}$
18. **end if**
19. **end for**
20. $\mathcal{A}_{max} := \mathcal{A}_{max} \cup \{X\}$
21. /* Update \mathcal{D}_{min} to $\mathfrak{S}(\overline{\mathcal{A}_{max}})$ */
22. **if** $U \in \mathcal{A}_{max}$ **then**
23. **return** $\{\}, \{U\}$
24. **end if**
25. $\mathcal{D}_\delta := \{\}$
26. **for all** $Y \in \mathcal{D}_{maybe}$ with $Y \subseteq X$ **do**
27. $\mathcal{D}_{maybe} := \mathcal{D}_{maybe} - \{Y\}$
28. $\mathcal{D}_\delta := \mathcal{D}_\delta \cup \{Y \cup \{B\} \mid B \in U - X\}$
29. **end for**
30. **for all** $Y \in \mathcal{D}_\delta$ **do**
31. **if** Y is not \sqsubseteq -minimal in $\mathcal{D}_{maybe} \cup \mathcal{D}_\delta \cup \mathcal{D}_{final}$ **then**
32. $\mathcal{D}_\delta := \mathcal{D}_\delta - \{Y\}$
33. **end if**
34. **end for**
35. $\mathcal{D}_{maybe} := \mathcal{D}_{maybe} \cup \mathcal{D}_\delta$
36. **end if**
37. **end while**
38. **return** $\mathcal{D}_{final}, \mathcal{A}_{max}$

Lemma 3.70. *The number of oracle calls in Algorithm 3.13 is bounded by*

$$|\mathcal{A}_{max}| \times |U| + |\mathcal{D}_{final}|$$

The number of iterations of the “while” loop is exactly

$$|\mathcal{A}_{max}| + |\mathcal{D}_{final}|$$

Proof. \mathcal{A}_{max} and \mathcal{D}_{final} are initially empty, and each iteration adds exactly one new element to \mathcal{A}_{max} or \mathcal{D}_{final} . For iterations where $|\mathcal{D}_{final}|$ is increased, only one oracle call is made, and for those increasing $|\mathcal{A}_{max}|$ the number of oracle calls is $|U - X| \leq |U|$. \square

While the number of oracle calls is polynomial in the size of the output, the overall runtime need not be, since the set \mathcal{D}_{maybe} of intermediate transversals can be much larger than \mathcal{D}_{final} . An output-polynomial algorithm cannot be expected though, since no output-polynomial algorithm for computing the hitting sets of a hypergraph is known either [36, 37]. This raises the question of whether this issue with the complexity is caused by the computation of \mathcal{D}_{final} , which we claimed was “for free” (i.e., at no additional computational effort). But even if we did not want to obtain \mathcal{D}_{final} , our problem is still as hard as the transversal problem.

The Oracle: For completeness, we suggest one possible way to define an oracle for telling us whether two pre-image trees agree on a given essential v -subgraph. It is also possible to consider any unordered tree isomorphism algorithm or the notions of partitions as defined in the next section. The existence of a more efficient oracle is left open.

Given an essential v -subgraph X and pre-image trees $p_1, p_2 \in P_{T'}(v)$ we can apply a recursive approach for deciding whether $p_1|_X \doteq p_2|_X$ holds.

Lemma 3.71. *Let T_A, T_B be two data trees with roots r_A, r_B having direct children nodes $\{a_1, \dots, a_m\}$ and $\{b_1, \dots, b_n\}$ respectively. Let us further denote the total subtree of T_A with root a_i by $T_A(a_i)$, and similarly $T_B(b_i)$ denotes the total subgraph of T_B with root b_i . Then $T_A \doteq T_B$ if and only if $name(r_A) = name(r_B)$, $m = n$ and there exists a bijective mapping*

$$\pi : \{T_A(a_1), \dots, T_A(a_m)\} \rightarrow \{T_B(b_1), \dots, T_B(b_n)\}$$

such that $T_A(a_i) \doteq \pi(T_A(a_i))$, for all $1 \leq i \leq m$.

Proof. Follows directly from the definition of property-equality. \square

Thus we can reduce the problem of checking whether two data trees T_A, T_B are property-equal to checking pairwise property-equality of its child-subtrees with root r_A, r_B (assuming that $name(r_A) = name(r_B)$ and $m = n$, otherwise no further check is necessary). However, it is not immediately clear how the mapping π is to be chosen (because we consider un-ordered trees), and testing all pairs of child-subtrees $(T_A(a_i), T_B(b_i))$ for property-equality to decide whether such a mapping π exists is expensive.

The typical answer is to introduce a canonisation of all data trees using some pre-ordering \leq_v such that T_A, T_B has identical canonical form if and only if they are property-equal. The canonical form of T_A results from sorting the child-subtrees $T_A(a_i)$ of T_A according to \leq_v so that (after re-indexing) we have

$$T_A(a_1) \leq_v T_A(a_2) \leq_v \dots \leq_v T_A(a_m)$$

and similarly for $T_B(b_i)$. Then by Lemma 3.71, $T_A \doteq T_B$ if and only if

$$T_A(a_i) \doteq T_B(b_i) \text{ for every } i = 1, \dots, m$$

This reduces the number of recursive tests for property-equality of child-subtrees to at most m .

The pre-order \leq_v can be defined recursively. For that we assume that some linear pre-ordering for individual nodes are given (based on name, kind and leaf values). For concreteness, we suppose $n_A \leq_v n_B$ where n_A, n_B are two nodes in a data tree to be given by the lexicographical order as follows, $n_A \leq_v n_B$ if and only if one of the following statements holds:

- $kind(n_A) < kind(n_B)$, or
- $kind(n_A) = kind(n_B)$ and $name(n_A) < name(n_B)$, or
- $kind(n_A) = kind(n_B)$ and $name(n_A) = name(n_B)$ and either n_A, n_B are internal nodes or $valuation(n_A) \leq valuation(n_B)$.

Furthermore let $n_A =_v n_B$ denote $n_A \leq_v n_B$ and $n_B \leq_v n_A$. And we also denote by $n_A <_v n_B$ the fact that $n_A \leq_v n_B$ and $n_B \not\leq_v n_A$. This is summarised in Algorithm 3.14.

Definition 3.72. We say T_A *value-precedes* T_B , written $T_A \leq_v T_B$, if and only if

- $r_A <_v r_B$, or
- $r_A =_v r_B$ and $m < n$, or
- $r_A =_v r_B$ and $m = n$ and there is some $1 \leq i \leq m$ such that $T_A(a_j) =_v T_B(b_j)$ for all $j = 1, \dots, i$ and $T_A(a_{i+1}) <_v T_B(b_{i+1})$,
assuming that $T_A(a_1), \dots, T_A(a_m)$ is ordered according to \leq_v , i.e., we have

$$T_A(a_1) \leq_v T_A(a_2) \leq_v \dots \leq_v T_A(a_m)$$

and similarly for $T_B(b_1), \dots, T_B(b_m)$,

Computing whether $T_A \doteq T_B$ holds can be done recursively, by applying the definition. At each recursion level, this requires us to sort the child-subtrees A_i and B_i (using recursion) and compare them, again recursively. With this approach, checking property-equality for trees becomes a special case of checking value-precedence. In particular, $T_A \doteq T_B$ if and only if $T_A =_v T_B$. Thus we only need one simple recursive function: Algorithm 3.15.

Algorithm 3.14 lexical_precedes**Input:** Nodes n_A, n_B **Output:** $\circ \in \{<_v, >_v, =_v\}$ such that $n_A \circ_v n_B$ holds

1. **if** $kind(n_A) < kind(n_B)$ **then**
2. **return** “ $<_v$ ”
3. **else if** $kind(n_A) > kind(n_B)$ **then**
4. **return** “ $>_v$ ”
5. **else if** $name(n_A) < name(n_B)$ **then**
6. **return** “ $<_v$ ”
7. **else if** $name(n_A) > name(n_B)$ **then**
8. **return** “ $>_v$ ”
9. **else if** n_A, n_B are internal nodes **then**
10. **return** “ $=_v$ ”
11. **else if** $valuation(n_A) < valuation(n_B)$ **then**
12. **return** “ $<_v$ ”
13. **else if** $valuation(n_A) > valuation(n_B)$ **then**
14. **return** “ $>_v$ ”
15. **else**
16. **return** “ $=_v$ ”
17. **end if**

Example 3.73. Consider the pair of pre-image trees of $v_{Purchase}$: $\{i_5, i_{18}\}$. To determine whether pre-image trees i_5 and i_{18} agree on $\{De, Pr, Di\}$ we apply Algorithm 3.15 to compare the data tree $T_{i_5} = i_5|_{\{De, Pr, Di\}}$ with $T_{i_{18}} = i_{18}|_{\{De, Pr, Di\}}$.

The data trees $T_{i_5}, T_{i_{18}}$ have roots i_5 and i_{18} respectively. Both of these internal nodes are elements (i.e., $kind = E$) with $name = \text{“Purchase”}$ which means $i_5 =_v i_{18}$. Moreover they both have the same number of children nodes, and so we sort the children nodes of T_{i_5} and $T_{i_{18}}$.

The data tree T_{i_5} has children nodes i_9 and i_{13} which are sorted by asking whether the data tree T_{i_9} with root i_9 value-precedes the data tree $T_{i_{13}}$ with root i_{13} . Both i_9 and i_{13} are internal node with $kind = E$ and $name = \text{“Item”}$ and have three children nodes. Therefore, we next sort the children nodes of T_{i_9} and $T_{i_{13}}$. This yield

$$T_{i_{10}} \leq_v T_{i_{12}} \leq_v T_{i_{11}} \quad \text{and} \quad T_{i_{14}} \leq_v T_{i_{16}} \leq_v T_{i_{15}}$$

for the subtrees rooted at children nodes of T_{i_9} and $T_{i_{13}}$ respectively. We find $T_{i_{10}} <_v T_{i_{14}}$ and so $T_{i_9} <_v T_{i_{13}}$. That is, the subtrees rooted at children nodes of T_{i_5} are sorted as

$$T_{i_9} \leq_v T_{i_{13}}.$$

Similarly, we sort the subtrees rooted at children nodes of $T_{i_{18}}$ and find

$$T_{i_{22}} <_v T_{i_{25}}.$$

Then, comparing the sorted children nodes of T_{i_5} against the sorted children nodes of $T_{i_{18}}$ we find $T_{i_9} <_v T_{i_{22}}$ and so $T_{i_5} <_v T_{i_{18}}$. In particular, this means $T_{i_5} \neq T_{i_{18}}$.

Algorithm 3.15 `value_precedes(oracle)`

Input: Data trees $T_A(r_A), T_B(r_B)$ **Output:** $\circ \in \{<_v, >_v, =_v\}$ such that $T_A \circ_v T_B$ holds

1. $\circ := \text{lexical_precedes}(r_A, r_B)$
 2. **if** $\circ \neq =_v$ **then**
 3. **return** \circ
 4. **else if** $m < n$ **then**
 5. **return** $<_v$
 6. **else if** $m > n$ **then**
 7. **return** $>_v$
 8. **end if**
 9. `sort_children`($T_A(r_A)$)
 10. `sort_children`($T_B(r_B)$)
 11. **for** i from 1 to m **do**
 12. $\circ := \text{value_precedes}(T_A(a_i), T_B(b_i))$
 13. **if** $\circ \neq =_v$ **then**
 14. **return** \circ
 15. **end if**
 16. **end for**
 17. **return** $=_v$
-
18. **proc** `sort_children`($T_X(r_X)$)
 19. $\mathcal{X} := \{T_X(x_i) \mid x_i \text{ is a direct child node of } r_X\}$
 20. **return** $T_X(x_1), \dots, T_X(x_k)$
 where $T_X(x_1) \leq_v T_X(x_2) \leq_v \dots \leq_v T_X(x_k)$
 and $\mathcal{X} = \{T_X(x_1), \dots, T_X(x_k)\}$
-

Finding \sqsubseteq -maximal Essential v -Subgraphs: From Agree-Partitions

In the pair-by-pair approach from the previous section, we frequently need to access the given data tree T' . Instead, property-equality information from T' can be represented by a partition database, which can then be used to derive the \sqsubseteq -maximal essential v -subgraphs which belongs to each agree set of T' .

Given a set M , a *partition* π on M is a family of mutually disjoint, non-empty subsets of M . The members of π are the *partition classes* and their union is the *support* of π . We call M the *context* of π , and the *exterior* of π is the difference between its context and its support. We call a partition *total* if its exterior is empty. We call a partition *stripped* if all its partition classes are non-empty non-singleton. We obtain the *stripped partition* $\hat{\pi}$ of π by considering only the non-empty non-singleton partition classes of π .

Definition 3.74 (agree-partition). A set \mathcal{X} of essential v -subgraphs induces a (total) *agree-partition* $\Pi_{\mathcal{X}}(v)$ on $P_{T'}(v)$ where two pre-images $p_1, p_2 \in P_{T'}(v)$ belong to the same partition class if and only if for all $X \in \mathcal{X}$ we have $p_1|_X \doteq p_2|_X$.

We can reduce the *partition database* $\Pi(v) = \{\Pi_{\{X\}}(v) \mid X \in \mathbf{E}_T^{\tilde{\S}}(v)\}$ to a *stripped partition database*

$$\widehat{\Pi}(v) = \{\widehat{\Pi}_{\{X\}}(v) \mid \Pi_{\{X\}}(v) \in \Pi(v)\}$$

from which we can find the family

$$\tilde{\S}\mathfrak{A}_{T'}(v) = \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}} \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}$$

of v -agree sets projected to subgraphs by following the simple procedure:

```

proc find_agreeSets-fromPartition
 $\widehat{\Pi}(v) := \text{find\_stripPartitionDB}$  // Algorithm 3.18
 $\tilde{\S}\mathfrak{A}_{T'}(v) := \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}} \mid p_1, p_2 \in P_{T'}(v) \text{ and } p_1 \neq p_2\}$ 
Initialise  $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}} := \{\}$  for all  $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}} \in \tilde{\S}\mathfrak{A}_{T'}(v)$ 

for all  $\Pi_{\{X\}}(v) \in \widehat{\Pi}(v)$  do
  for all  $g \in \Pi_{\{X\}}(v)$  do
    for all  $p_1, p_2 \in g$  such that  $p_1 \neq p_2$  do
       $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}} := \mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}} \cup \{X\}$ 
    end for
  end for
end for

```

This simple procedure takes a long time to output the first projected v -agree set and need to keep track of $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\tilde{\S}}$ for all pairs of distinct pre-image trees even though $\mathfrak{A}_{T'}(v)|_{\tilde{\S}}$ can potentially be much smaller. Instead, [61] proposes an alternate characterisation of agree sets in relational databases in terms of partitions. Their characterisation can be extended to our situation here. We briefly sketch the approach here. For every

stripped partition $\widehat{\Pi_{\{X\}}}(v)$ we fix an enumeration of the partition classes, say g_i with $i = 1, \dots, n_X$. We call pre-image $p \in P_{T'}(v)$ an (X, i) -node if $p \in g_i \in \widehat{\Pi_{\{X\}}}(v)$.

For each pre-image tree p we can identify all partition classes it belongs to:

$$ec(p) = \{(X, i) \mid p \text{ is an } (X, i)\text{-node}\}$$

Every pair of distinct (X, i) -nodes is called an *MC-pair* and corresponds to exactly one v -agree set $\mathcal{A} \in \mathfrak{A}_{T'}(v)$ such that $\mathfrak{A}_{T'}(v) \neq \{\}$. In fact, we can easily show that for any pair of distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ we have $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}} \neq \{\}$ if and only if $\{p_1, p_2\}$ is an *MC-pair*. The terms $ec(\cdot)$ and *MC-pair* are borrowed from [61] where ec probably corresponds to “equivalence classes” and *MC* for “maximal class” (since it is sufficient to examine the maximal partition classes to find all *MC-pairs*).

The v -agree set for an *MC-pair* p_1, p_2 projected to essential v -subgraphs is given by

$$\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}} = \{X \mid \exists k. (X, k) \in ec(p_1) \cap ec(p_2)\}$$

and so

$$\sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}}) = \vartheta(\{X \mid \exists k. (X, k) \in ec(p_1) \cap ec(p_2)\}).$$

In summary, the following is an alternative procedure for finding $\mathfrak{A}_{T'}(v)|_{\mathfrak{S}}$ from the agree-partitions:

```

proc find_agreeSets-fromPartition(Alternative)
 $\widehat{\Pi}(v) := \text{find\_stripPartitionDB}$  // Algorithm 3.18
Enumerate partition classes for each agree partition from  $\widehat{\Pi}(v)$ 
for all  $p \in P_{T'}(v)$  do
     $ec(p) := \{(X, i) \mid p \text{ is an } (X, i)\text{-node}\}$ 
end for
 $MC\text{-pair} := \{\{p_1, p_2\} \mid p_1 \neq p_2 \text{ and } \exists X \in \mathbf{E}_T^{\mathfrak{S}}(v). p_1, p_2 \text{ are } (X, i)\text{-node}\}$ 
 $\mathfrak{A}_{T'}(v) := \{\}$ 
for all  $\{p_1, p_2\} \in MC\text{-pair}$  do
     $\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}} := \vartheta(\{X \mid \exists k. (X, k) \in ec(p_1) \cap ec(p_2)\})$ 
     $\mathfrak{A}_{T'}(v) := \mathfrak{A}_{T'}(v) \cup \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}}\}$ 
end for

```

Example 3.75. From the purchase data tree $T'_{purchase}$ we find the following agree-partitions

induced by the singleton sets of essential $v_{Purchase}$ -subgraphs:

$$\begin{aligned}
\Pi_{\{Da\}}(v_{Purchase}) &= \{ \{i_5, i_{18}, i_{31}, i_{44}\} \} \\
\Pi_{\{Ti\}}(v_{Purchase}) &= \{ \{i_5, i_{18}\}, \{i_{31}, i_{44}\} \} \\
\Pi_{\{Sa\}}(v_{Purchase}) &= \{ \{i_5, i_{31}\}, \{i_{44}\}, \{i_{18}\} \} \\
\Pi_{\{De\}}(v_{Purchase}) &= \{ \{i_5, i_{18}, i_{44}\}, \{i_{31}\} \} \\
\Pi_{\{Pr\}}(v_{Purchase}) &= \{ \{i_5, i_{18}\}, \{i_{31}, i_{44}\} \} \\
\Pi_{\{Di\}}(v_{Purchase}) &= \{ \{i_5, i_{31}\}, \{i_{44}\}, \{i_{18}\} \} \\
\Pi_{\{Pn\}}(v_{Purchase}) &= \{ \{i_5\}, \{i_{18}\}, \{i_{31}\}, \{i_{44}\} \} \\
&= \Pi_{\{De, Pr\}}(v_{Purchase}) \\
&= \Pi_{\{De, Di\}}(v_{Purchase}) \\
&= \Pi_{\{Pr, Di\}}(v_{Purchase}) \\
&= \Pi_{\{De, Pr, Di\}}(v_{Purchase})
\end{aligned}$$

which gives us the stripped partition database

$$\widehat{\Pi}(v) = \left\{ \begin{array}{l} \widehat{\Pi}_{\{Da\}}(v_{Purchase}) = \{ \{i_5, i_{18}, i_{31}, i_{44}\} \}, \\ \widehat{\Pi}_{\{Ti\}}(v_{Purchase}) = \{ \{i_5, i_{18}\}, \{i_{31}, i_{44}\} \}, \\ \widehat{\Pi}_{\{Sa\}}(v_{Purchase}) = \{ \{i_5, i_{31}\} \}, \\ \widehat{\Pi}_{\{De\}}(v_{Purchase}) = \{ \{i_5, i_{18}, i_{44}\} \}, \\ \widehat{\Pi}_{\{Pr\}}(v_{Purchase}) = \{ \{i_5, i_{18}\}, \{i_{31}, i_{44}\} \}, \\ \widehat{\Pi}_{\{Di\}}(v_{Purchase}) = \{ \{i_5, i_{31}\} \} \end{array} \right\}$$

In this case, each pair of distinct pre-images of $v_{Purchase}$ form an MC -pair, that is, there are six MC -pairs. Enumeration of the stripped partition classes in the order that they appear above yields:

$$\begin{aligned}
ec(i_5) &= \{(Da, 1), (Ti, 1), (Sa, 1), (De, 1), (Pr, 1), (Di, 1)\} \\
ec(i_{18}) &= \{(Da, 1), (Ti, 1), (De, 1), (Pr, 1)\} \\
ec(i_{31}) &= \{(Da, 1), (Ti, 2), (Sa, 1), (Pr, 2), (Di, 1)\} \\
ec(i_{44}) &= \{(Da, 1), (Ti, 2), (De, 1), (Pr, 2)\}
\end{aligned}$$

Then

$$ec(i_5) \cap ec(i_{18}) = \{(Da, 1), (Ti, 1), (De, 1), (Pr, 1)\}$$

and so

$$\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\S} = \{Da, Ti, De, Pr\} = \sqsubseteq_{-max} (\mathcal{A}_{\{i_5, i_{18}\}}(v_{Purchase})|_{\S})$$

By a similar process we can find all $v_{Purchase}$ -agree sets in $\mathfrak{A}_{T'_{purchase}}(v_{Purchase}|_{\S})$. All complete $v_{Purchase}$ -agree sets for $T'_{purchase}$ are listed in Example 3.66. \square

Thus the problem of computing $\mathfrak{S}\mathfrak{A}_{T'}(v)$ reduces to the problem of generating the stripped partition database of T' .

Computing agree-Partitions: To compute each agree-partition we can access the data tree. However, it is the frequency of these potentially costly accesses that we want to keep as small as possible. The question is then which agree-partitions can be derived (and how) from other partitions that we have already found?

When considering how to compute one partition from another, it is useful to distinguish those projections which are empty from those which are non-empty. The following observations provide some motivations. Firstly, for any distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ projection to X gives the empty subtree if and only if projection to every v -walk contained in X gives the empty subtree. But this does not hold for non-empty projections. Recall that property-equality on some v -walks does not necessarily imply property-equality on the v -subgraph consisting of those v -walks. Secondly, knowing that pre-image trees of w have property-equal non-empty projection on some property allows us to deduce certain information about property-equality of the pre-image trees of v containing those pre-image trees of w . This is not the case for property-equal empty projections. For example, the data trees with the parents of nodes $i_9, i_{13}, i_{35}, i_{39}, i_{48}$ as the roots must also have non-empty projections to Di but, knowing pre-images i_{22}, i_{25}, i_{52} of v_{Item} have empty projections to Di does not tell us anything about whether the pre-images of any ancestor of v_{Item} have empty projections to Di . Thirdly, a pre-image tree of v having non-empty projections implies the existence of descendants of v ; but the same cannot be said about pre-images with empty projections.

Definition 3.76. For an agree-partition $\Pi_{\mathcal{X}}(v)$ we have

- the *null class* $\perp_{\mathcal{X}}(v)$ is the set of all pre-images $p \in P_{T'}(v)$ whose projection $p|_X$ is empty for all $X \in \mathcal{X}$.
- the *non-null partition* $\mathfrak{N}_{\mathcal{X}}(v)$ is obtained by removing the null class $\perp_{\mathcal{X}}(v)$ from the total partition $\Pi_{\mathcal{X}}(v)$

□

The following example provides an example of these concepts over the purchase data tree T'_{purchase} .

Example 3.77. For the purchase data tree in Figure 3.1, an example of a total partition induced by a set of subgraph, its null class and the corresponding non-null partition are respectively $\Pi_{\{\text{Di}\}}(v_{\text{Item}}) = \{\{i_9, i_{13}, i_{35}, i_{39}, i_{48}\}, \{i_{22}, i_{25}, i_{52}\}\}$, $\perp_{\{\text{Di}\}}(v_{\text{Item}}) = \{i_{22}, i_{25}, i_{52}\}$ and $\mathfrak{N}_{\{\text{Di}\}}(v_{\text{Item}}) = \{\{i_9, i_{13}, i_{35}, i_{39}, i_{48}\}\}$. □

Firstly, we address the question of how to find null classes. Provided $\perp_{\{X\}}(v) \neq \{\}$, we have $\Pi_{\{X\}}(v) = \mathfrak{N}_{\{X\}}(v) \cup \{\perp_{\{X\}}(v)\}$, which suggests

Lemma 3.78. $\perp_{\mathcal{X}}(v)$ equals $P_{T'}(v)$ minus the support of $\mathfrak{N}_{\mathcal{X}}(v)$.

Proof. Follows from $\Pi_{\mathcal{X}}(v)$ being a total partition of $P_{T'}(v)$ and $\Pi_{\mathcal{X}}(v)$ is composed of partition classes in $\mathfrak{N}_{\mathcal{X}}(v)$ together with non-empty null class $\perp_{\mathcal{X}}(v)$. □

This tells us how to compute the null class $\perp_{\mathcal{X}}(v)$ from the non-null partition $\mathfrak{N}_{\mathcal{X}}(v)$. But in fact, we can compute all null classes from the non-null partitions induced by singleton sets of v -walks because of our observation that

$$p_1|_X \doteq p_2|_X = \emptyset \text{ if and only if } p_1|_B \doteq p_2|_B = \emptyset \text{ for every } v\text{-walk } B \in X$$

And so, from the non-null partitions induced by v -walks we can compute the corresponding null classes for v -walks. Then for a v -subgraph $X = \{B_1, \dots, B_n\}$ we have

$$\perp_{\{X\}}(v) = \perp_{\{B_1\}}(v) \cap \dots \cap \perp_{\{B_n\}}(v).$$

To determine the strip partition database, we first compute the non-null partitions for all essential v -properties and from those compute the null classes. This brings us to the question of how to compute non-null partitions for essential v -subgraphs. Recall that for a v -unit U containing more than one v -walk, the unique identifier $\eta(U)$ is the proper descendant w of v included in U such that the incoming arc of w is the only arc on the path from v to w which has frequency other than ? and 1. We suggest a recursive approach for computing non-null partitions whereby:

- for every singleton v -unit U , the non-null partition $\mathfrak{N}_{\{U\}}(v)$ is computed by accessing the data, and
- for every non-singleton v -unit U and every essential v -subgraph X contained in U , the non-null partition $\mathfrak{N}_{\{X\}}(v)$ is computed from $\mathfrak{N}_{\{X\}}(\eta(U))$.

Algorithm 3.16 summarises our recursive approach for computing non-null partitions. We consider each v -unit U independently. If U is a singleton then it consists of single v -walk where the path from v to the leaf is simple. We handle singleton v -units in the base case and we compute the non-null partitions from a relational representation of the original XML data (Step 4). If U is not a singleton then it has an identifier node $\eta(U) = w$. We then first compute the non-null partitions of $P_{T'}(w)$ induced by essential w -subgraphs. This is done in a similar manner, by considering every w -units. In the case that an essential v -subgraph X is not an essential w -subgraph then we can observe that $\vartheta_w(\{X\})$ is a set of pair-wise reconcilable w -subgraphs whose union is X . Thus $\mathfrak{N}_{\{X\}}(w)$ is the composition of the non-null partitions induced by each w -subgraph in $\vartheta_w(\{X\})$ (Step 12). This step is supported by Proposition 3.93. Then for every essential v -subgraph X we have $\mathfrak{N}_{\{X\}}(w)$ from which we can compute $\mathfrak{N}_{\{X\}}(v)$ by applying Proposition 3.87 (Step 16).

Computation of non-null partitions for v -walks from the given data tree can be further simplified by a relational representation of the data tree. The issues of transforming XML data into relational data for the purpose of leveraging existing technologies have been addressed by many works in the literature (e.g., [63, 82, 86]). However, we do not recommend our relational representation as a general XML to relational mapping.

For each v -walk B , the *lowest multiple node*, denoted by $\mathbf{lmn}(B)$, is the lowest node contained in B such that the incoming arc of $\mathbf{lmn}(B)$ has frequency other than ? and 1

Algorithm 3.16 find_nonnullPartitions

Input: Node v , database db /* where db is the relational representation of data tree T' */**Output:** $\mathfrak{N}(v)$ /* where $\mathfrak{N}(v) = \{\mathfrak{N}_{\{X\}}(v) \mid X \in \mathbf{E}_T^{\S}(v)\}$ */

1. $\mathfrak{N}(v) := \{\}$;
2. **for all** v -unit U in T **do**
3. **if** $|U| = 1$ **then**
4. $\mathfrak{N}_{\{U\}}(v) :=$ find partition class from GROUPBY query on $r_v \in db$;
5. $\mathfrak{N}(v) := \mathfrak{N}(v) \cup \{\mathfrak{N}_{\{U\}}(v)\}$;
6. **else**
7. $w := \eta(U)$;
8. $\mathfrak{N}(w) :=$ find_nonnullPartition(w, db);
9. **for all** v -subgraph X which is contained in U **do**
10. /* Proposition 3.93 */
11. **if** $\mathfrak{N}_{\{X\}}(w) \notin \mathfrak{N}(w)$ **then** // X not essential w -subgraph
12. $\mathfrak{N}_{\{X\}}(w) :=$ compose($\{\mathfrak{N}_{\{Y\}}(w) \in \mathfrak{N}(w) \mid Y \in \vartheta_w(\{X\})\}$)
13. **end if**
14. /* Proposition 3.87 */
15. $card_partition := \{\pi(\alpha_v(g), g) \mid g \in \mathfrak{N}_{\{X\}}(w)\}$
16. $\mathfrak{N}_{\{X\}}(v) :=$ compose($card_partition$)
17. $\mathfrak{N}(v) := \mathfrak{N}(v) \cup \{\mathfrak{N}_{\{X\}}(v)\}$;
18. **end for**
19. **end if**
20. **end for**
21. **return** $\mathfrak{N}(v)$;

while the path from $\mathbf{lmn}(B)$ to the leaf of B is simple. Since we are only interested in discovering pXFDs with non-simple target v , this node is guaranteed to exist for every v -walk. We generate a relation r_n if there is some v -walk B with $\mathbf{lmn}(B) = n$ and we give r_n the fields:

- “@pre-image” for storing the identifier path $s_k(p)$ for pre-image tree $p \in P_{T'}(n)$ (see the beginning of Section 3.3), and
- a field “ B ” for every v -walk B with $\mathbf{lmn}(B) = n$, which stores the string value assigned to the leaf $p|_B$. We say that such a B is a walk *represented in* r_n .

We populate an instance of r_n with a tuple for every pre-image tree $p \in P_{T'}(n)$ which has a non-empty projection $p|_X$ to some n -walk X represented in r_n . Of course such a pre-image tree *may not* have non-empty projections to *every* walk represented by r_n ; when a projection $p|_B = \emptyset$ then we write “NULL” as the string value for field B of the tuple associated with p .

$r_{v_{Outlet}}$		
@pre-image	Na	Ad
i_1/i_2	Pioneer	PN
i_1/i_{28}	Levin	Levin

$r_{v_{Purchase}}$				
@pre-image	Da	Ti	Pn	Sa
$i_1/i_2/i_5$	26-Jul-07	11am	p001	\$1
$i_1/i_2/i_{18}$	26-Jul-07	11am	p003	NULL
$i_1/i_{28}/i_{31}$	26-Jul-07	2pm	p002	\$1
$i_1/i_{28}/i_{44}$	26-Jul-07	2pm	p004	\$0.50

$r_{v_{Item}}$			
@pre-image	De	Pr	Di
$i_1/i_2/i_5/i_9$	Kiwifruit	\$3	\$0.50
$i_1/i_2/i_5/i_{13}$	Strawberries	\$2	\$0.50
$i_1/i_2/i_{18}/i_{22}$	Kiwifruit	\$2	NULL
$i_1/i_2/i_{18}/i_{25}$	Strawberries	\$3	NULL
$i_1/i_{28}/i_{31}/i_{35}$	Kiwifruit	\$3	\$0.50
$i_1/i_{28}/i_{31}/i_{39}$	Kiwifruit	\$3	\$0.50
$i_1/i_{28}/i_{44}/i_{48}$	Kiwifruit	\$3	\$0.50
$i_1/i_{28}/i_{44}/i_{52}$	Strawberries	\$3	NULL

Figure 3.6: Relational representation of XML data tree $T'_{Purchase}$.

For every walk B represented in r_n , we can compute the non-null partition $\mathfrak{N}_{\{B\}}(n)$ from relation r_n with the query

SELECT @pre-image FROM r_n WHERE $B \neq$ “NULL” GROUP BY B ;

Example 3.79. From the purchase schema tree $T_{purchase}$ in Figure 3.2, we can identify the lowest multiple node for each walk in $T_{purchase}$, these are:

$$\begin{array}{lll}
 \text{lmn}(\text{Na}) = v_{Outlet} & \text{lmn}(\text{Da}) = v_{Purchase} & \text{lmn}(\text{De}) = v_{Item} \\
 \text{lmn}(\text{Ad}) = v_{Outlet} & \text{lmn}(\text{Ti}) = v_{Purchase} & \text{lmn}(\text{Pr}) = v_{Item} \\
 & \text{lmn}(\text{Pn}) = v_{Purchase} & \text{lmn}(\text{Di}) = v_{Item} \\
 & \text{lmn}(\text{Sa}) = v_{Purchase} &
 \end{array}$$

Accordingly, we identify three relation schemas:

$$\begin{array}{l}
 r_{v_{Outlet}} = \{ @pre-image, \text{Na}, \text{Ad} \} \\
 r_{v_{Purchase}} = \{ @pre-image, \text{Da}, \text{Ti}, \text{Pn}, \text{Sa} \} \\
 r_{v_{Item}} = \{ @pre-image, \text{De}, \text{Pr}, \text{Di} \}
 \end{array}$$

The purchase data tree $T'_{purchase}$ from Figure 3.1, can therefore be represented by the set of relations shown in Figure 3.6.

There is a tuple in $r_{v_{Item}}$ for each pre-image tree

$$p \in P_{T'_{purchase}}(v_{Item}) = \{i_9, i_{13}, i_{22}, i_{25}, i_{35}, i_{39}, i_{48}, i_{52}\}.$$

For example the third row of the relation $r_{v_{Item}}$ in Figure 3.6 corresponds to pre-image tree i_{22} whose projection to walk De contains a single leaf node with string value “Kiwifruit” and whose projection to walk Pr contains a single leaf node with string value “\$2”. The projection of i_{22} to walk Di results in the empty subgraph, and so we use the value “NULL” for the field Di in the tuple with @pre-image value “ $i_1/i_2/i_{18}/i_{22}$ ”.

Using the relation $r_{v_{Purchase}}$ and simple GROUP BY queries, we can compute the non-null partitions $\mathfrak{N}_{\{Da\}}(v_{Purchase})$, $\mathfrak{N}_{\{Ti\}}(v_{Purchase})$, $\mathfrak{N}_{\{Pn\}}(v_{Purchase})$ and $\mathfrak{N}_{\{Sa\}}(v_{Purchase})$ of $P_{T'_{purchase}}(v_{Item})$. Note that these are the non-null partitions induced by singleton $v_{Purchase}$ -units of $T_{purchase}$. For example, to find the non-null partition $\mathfrak{N}_{\{Da\}}(v_{Purchase})$, we execute the GROUP BY query

```
SELECT @pre-image FROM r_{v_{Purchase}} WHERE Da ≠ "NULL" GROUP BY Da;
```

which returns a relation with one group

$$\left\{ \begin{array}{l} i_1/i_2/i_5, \\ i_1/i_2/i_{18}, \\ i_1/i_{28}/i_{31}, \\ i_1/i_{28}/i_{44} \end{array} \right\}$$

This gives the non-null partition $\mathfrak{N}_{\{Da\}}(v_{Purchase}) = \{\{i_5, i_{18}, i_{31}, i_{44}\}\}$.

The GROUP BY query

```
SELECT @pre-image FROM r_{v_{Purchase}} WHERE Ti ≠ "NULL" GROUP BY Ti;
```

which returns a relation with two groups, i.e., $\mathfrak{N}_{\{Ti\}}(v_{Purchase}) = \{\{i_5, i_{18}\}, \{i_{31}, i_{44}\}\}$. \square

Finally, let us investigate how to compute the non-null partitions induced by essential v -subgraphs from other non-null partitions. Consider a v -unit U identifiable by $\eta(U) = w$. For every essential v -subgraph X contained in U which is an essential w -subgraph, we have $p_1|_X \doteq p_2|_X$ for $p_1, p_2 \in P_{T'}(v)$ if and only if p_1 and p_2 contain the same number of pre-image trees of w which agree on X . This obviously ensures that a valuation-preserving isomorphism between $p_1|_X$ and $p_2|_X$ exists. On the other hand, if the essential v -subgraph X is not itself an essential w -subgraph then we can observe that X is the union of essential w -subgraphs from $\vartheta_w(\{X\})$ which are pair-wise w -reconcilable. Thus we need that every pair of pre-image trees of w which agree on every w -subgraph in $\vartheta_w(\{X\})$ will also agree on X . In particular, the projection of these pre-image trees of w to X will be non-empty if their projections to at least one w -subgraph in $\vartheta_w(\{X\})$ is non-empty.

Before detailing and proving the correctness of these steps, we need to define some operations on the partitions. For two partitions π_1, π_2 on some set M we say

- π_1 *refines* π_2 if every partition class in π_1 is a subset of some partition class in π_2 .
- *product* of π_1, π_2 gives the greatest partition on M which refines both π_1 and π_2 .
- a *residual* of π_1 modulo π_2 is a set obtained by removing the support of π_2 from some partition class $g \in \pi_1$.

The product of π_1, π_2 can be thought as the set of all non-empty intersections between partition classes in π_1 and partition classes in π_2 . Furthermore, the product of π_1, π_2 is again a partition on M . But a pre-image tree with non-empty projection to some v -subgraph X may not have non-empty projection to *every* v -walk contained in X , just at least for one of them. This leads us to defining a new operation for partitions as follows:

Definition 3.80. (composition of partitions) Given two partitions π_1, π_2 on some set M , the *composition* of π_1, π_2 gives the partition consisting of

- all partition classes in the product of π_1, π_2 and
- all non-empty residuals of each π_1 and π_2 modulo the product of π_1, π_2 . □

The composition of a total partition on M with other (possibly partial) partitions results in a total partition on M . If both π_1, π_2 are total partitions on M then the composition and product of π_1, π_2 are equivalent. It is also easy to see that if the stripped partition of one of π_1, π_2 is empty then their composition also yield an empty stripped partition.

Example 3.81. In addition to the total partition

- $\mathfrak{N}_{\{\text{Di}\}}(v_{Item}) = \{\{i_9, i_{13}, i_{35}, i_{39}, i_{48}\}\}$

from Example 3.77, we also have the following total partitions:

- $\Pi_{\{\text{De}\}}(v_{Item}) = \mathfrak{N}_{\{\text{De}\}}(v_{Item}) = \{\{i_9, i_{22}, i_{35}, i_{39}, i_{48}\}, \{i_{13}, i_{25}, i_{52}\}\}$
- $\Pi_{\{\text{Pr}\}}(v_{Item}) = \mathfrak{N}_{\{\text{Pr}\}}(v_{Item}) = \{\{i_9, i_{25}, i_{35}, i_{39}, i_{48}, i_{52}\}, \{i_{13}, i_{22}\}\}$

Note that all three v_{Item} -walks are pairwise reconcilable v_{Item} -subgraphs. The composition (and product) of $\mathfrak{N}_{\{\text{De}\}}(v_{Item})$ and $\mathfrak{N}_{\{\text{Pr}\}}(v_{Item})$ is the total partition

$$\mathfrak{N}_{\{\text{De,Pr}\}}(v_{Item}) = \mathfrak{N}_{\{\{\text{De,Pr}\}\}}(v_{Item}) = \{\{i_9, i_{35}, i_{39}, i_{48}\}, \{i_{25}, i_{52}\}, \{i_{22}\}, \{i_{13}\}\}.$$

For $\mathfrak{N}_{\{\text{De}\}}(v_{Item})$ and $\mathfrak{N}_{\{\text{Di}\}}(v_{Item})$, their product is

$$\{\{i_9, i_{35}, i_{39}, i_{48}\}, \{i_{13}\}\}$$

with a support of

$$\{i_9, i_{13}, i_{35}, i_{39}, i_{48}\}.$$

The residuals of $\mathfrak{N}_{\{\text{De}\}}(v_{Item})$ modulo the product are $\{i_{22}\}$ and $\{i_{25}, i_{52}\}$. On the other hand, there is no residuals of $\mathfrak{N}_{\{\text{Di}\}}(v_{Item})$ modulo the product because the single partition class of $\mathfrak{N}_{\{\text{Di}\}}(v_{Item})$ is exactly the support of product. The composition of $\mathfrak{N}_{\{\text{De}\}}(v_{Item})$ and $\mathfrak{N}_{\{\text{Di}\}}(v_{Item})$ therefore results in the partition

$$\mathfrak{N}_{\{\text{De,Di}\}}(v_{Item}) = \mathfrak{N}_{\{\{\text{De,Di}\}\}}(v_{Item}) = \{\{i_9, i_{35}, i_{39}, i_{48}\}, \{i_{13}\}, \{i_{22}\}, \{i_{25}, i_{52}\}\}$$

The composition of $\mathfrak{N}_{\{\text{De}\}}(v_{Item})$ and $\mathfrak{N}_{\{\text{Di}\}}(v_{Item})$ gives the partition $\mathfrak{N}_{\{\{\text{De,Di}\}\}}(v_{Item})$ due to Proposition 3.93, which we will come to later on. □

Definition 3.82. Let v be a node of schema tree T and w a descendant of v . For a pre-image w' of w in T -compatible data tree T' , we use $\alpha_v(w')$ to denote the unique pre-image of v in T' that is an ancestor of w' . Further, for a set $P_w \subseteq P_{T'}(w)$ of pre-images we put $\alpha_v(P_w) = \{\alpha_v(w') \mid w' \in P_w\}$. We call $\alpha_v(P_w)$ the *lifting* of P_w to v . □

Remark 3.83. In the relational representation, we have opted to store the identifier path (instead of simply the node id) of essential v -ancestors for each pre-image tree. For a pre-image w' of some essential v -ancestor w , we can identify the unique pre-image of essential v -ancestors u that is an ancestor of w by trimming the path expression from the right. For example, the pre-image i_9 of v_{Item} in the the purchase data tree has the identifier path $i_1/i_2/i_5/i_9$. To lift i_9 to node $v_{Purchase}$, an immediate essential ancestor of v_{Item} , we trim the identifier path to get rid of the last node id and separator on the right. This gives $\alpha_{v_{Purchase}}(i_9) = i_5$. \square

Definition 3.84 (card-partition). Consider two sets $P_v \subseteq P_{T'}(v)$ and $P_w \subseteq P_{T'}(w)$ of pre-images. We define a partition $\pi(P_v, P_w)$ on P_v such that two members of P_v belong to the same partition class if and only if they have the same number of descendants in P_w , and call it the *card-partition* of P_v modulo P_w . \square

The following collection of statements establishes how we can identify $\mathfrak{N}_{\{X\}}(v)$ from $\mathfrak{N}_{\{X\}}(w)$ where $w = \eta(U)$ and U is not a singleton. Firstly, supposing w is a child of v , we reflect upon the fact that the composition of card-partition allows us to determine whether the pre-image trees of v contains the same number of each distinct (w.r.t. property-equality) pre-image subtrees of w , and therefore grouped in the same partition class in the corresponding non-null partition on $P_{T'}(v)$.

Lemma 3.85. *Let X be a w -subgraph, and w a child of v . Then $\mathfrak{N}_{\{X\}}(v)$ is the composition of the card-partitions $\pi(\alpha_v(g), g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$.*

Proof. We show that two pre-image trees $p_1, p_2 \in P_{T'}(v)$ belong to the same partition class of $\mathfrak{N}_{\{X\}}(v)$ if they belong to the same partition class of the composition of the card-partitions, and vice versa. This implies a partition class in $\mathfrak{N}_{\{X\}}(v)$ is a partition class in the composition of the card-partitions $\pi(\alpha_v(g), g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$, and vice versa.

(\subseteq) Consider two pre-images $p_1, p_2 \in P_{T'}(v)$ that belong to the same partition class of $\mathfrak{N}_{\{X\}}(v)$. Their projections $p_1|_X, p_2|_X$ are non-empty and therefore, p_1, p_2 must have children which are pre-images of w whose projection to X is non-empty. Let $p_1(w), p_2(w)$ be the set of such children belonging to p_1, p_2 respectively. Note that other pre-images of w not in $p_1(w), p_2(w)$ are not contained in the projection $p_1|_X, p_2|_X$ as they are either not descendant of p_1, p_2 or having empty projection to X . The valuation-preserving isomorphism between $p_1|_X, p_2|_X$ induces a bijective function $f : p_1(w) \rightarrow p_2(w)$ where for every $q_1 \in p_1(w)$ we have projections $q_1|_X, f(q_1)|_X$ are property-equal and non-empty. This means $q_1, f(q_1)$ belong to the same partition class in $\mathfrak{N}_X(w)$. Because the function is bijective, the number of children of p_1, p_2 which belongs to each partition class of $\mathfrak{N}_X(w)$ must be the same (including none). That is, for each card-partition $\pi(\alpha_v(g), g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$, we have either both p_1, p_2 or neither belongs to a partition class of $\pi(\alpha_v(g), g)$. It is easy to see that p_1, p_2 will therefore belong to the same partition class resulting from the composition of these card-partitions.

(\supseteq) Consider two pre-images $p_1, p_2 \in P_{T'}(v)$ belonging to the same partition class resulting from the composition of the card-partitions $\pi(\alpha_v(g), g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$. This means for each partition class $g \in \mathfrak{N}_{\{X\}}(w)$ both p_1, p_2 must have the same number of

children which belongs to g (including none). As above, let $p_1(w), p_2(w)$ be the set of children of p_1, p_2 respectively which are pre-images of w and contained in some partition class of $\mathfrak{N}_{\{X\}}(w)$. There exists a bijective function $f : p_1(w) \rightarrow p_2(w)$ such that for every $q_1 \in p_1(w)$ we have that the projections $q_1|_X \doteq f(q_1)|_X$ are non-empty. A valuation-preserving isomorphism between $p_1|_X, p_2|_X$ results from extending f by the valuation-preserving isomorphisms between elements of $p_1(w)$ and its image under f . Note again that pre-images of w not in $p_1(w), p_2(w)$ are not contained in the projections $p_1|_X, p_2|_X$ for the same reasons and therefore do not affect property-equality of $p_1|_X, p_2|_X$. \square

Of course if the path between v and a descendant w is simple then we do not even need to perform a composition at all, since the frequency restriction guarantees a one-to-one correspondence between the pre-image trees of w and that of v . This then means that, because the path from w to the parent of $w = \eta(U)$ is simple, we can apply the previous lemma to first find $\mathfrak{N}_{\{X\}}(u)$ where u is the parent of w and then the next corollary to transform the result into $\mathfrak{N}_{\{X\}}(v)$. It is easy to see that, alternatively, we can find $\mathfrak{N}_{\{X\}}(v)$ by first performing a lifting of pre-image trees in $\mathfrak{N}_{\{X\}}(w)$, then composing the subsequent card-partitions.

Corollary 3.86. *Let w be a simple descendant of v and X be w -subgraph. Then $\mathfrak{N}_{\{X\}}(v)$ consists of the sets $\alpha_v(g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$.*

Proof. The mapping $\alpha_v : P_{T'}(w) \rightarrow P_{T'}(v)$ is injective. Hence any two of the sets $\alpha_v(g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$ are disjoint, and each card-partition $\pi(\alpha_v(g), g)$ has $\alpha_v(g)$ as its only partition class. This implies the statement. \square

Proposition 3.87. *Let U be a non-singleton v -unit identified by $\eta(U)$. Further, let X be a v -subgraph contained in U . Then $\mathfrak{N}_{\{X\}}(v)$ is the composition of the card-partitions $\pi(\alpha_v(g), g)$ with $g \in \mathfrak{N}_{\{X\}}((\eta(U)))$.*

Proof. Let u be the parent of $\eta(U)$. By Lemma 3.85, $\mathfrak{N}_{\{X\}}((u))$ is the composition of the card-partitions $\pi(\alpha_u^{\eta(U)}(g), g)$ with $g \in \mathfrak{N}_{\{X\}}((\eta(U)))$ and by Corollary 3.86 $\mathfrak{N}_{\{X\}}(v)$ is the sets $\alpha_v^u(h)$ with $h \in \mathfrak{N}_{\{X\}}((u))$.

Note that $\alpha_v : P_{T'}(\eta(U)) \rightarrow P_{T'}(v)$ is injective. Moreover $\alpha_v(g) = \alpha_v^u(\alpha_u^{\eta(U)}(g))$ with $g \in \mathfrak{N}_X(\eta(U))$ because $\alpha_v^u : P_{T'}(u) \rightarrow P_{T'}(v)$ and $\alpha_u^{\eta(U)} : P_{T'}(\eta(U)) \rightarrow P_{T'}(u)$ are both injective. Actually $\pi(\alpha_v(g), g)$ is the same as the set consisting of $\alpha_v^u(f)$ with $f \in \pi(\alpha_u^{\eta(U)}(g), g)$. Because α_v^u is injective, it is easy to see that composing the card-partitions before applying α_v^u results in the same partition as the composition of the card-partitions after applying α_v^u . \square

Next we exemplify an application of Proposition 3.87.

Example 3.88. We can find the non-null partition $\mathfrak{N}_{\{\text{Pr}\}}(v_{\text{Purchase}})$ using non-null partition

$$\mathfrak{N}_{\{\text{Pr}\}}(v_{\text{Item}}) = \{\{i_9, i_{25}, i_{35}, i_{39}, i_{48}, i_{52}\}, \{i_{13}, i_{22}\}\}.$$

According to the purchase data tree, the mapping $\alpha_{v_{\text{Purchase}}}(P_{T'_{\text{purchase}}}(v_{\text{Item}}))$ is defined as follows:

$$\begin{array}{cccc} i_9 \mapsto i_5, & i_{13} \mapsto i_5, & i_{22} \mapsto i_{18}, & i_{25} \mapsto i_{18}, \\ i_{35} \mapsto i_{31}, & i_{39} \mapsto i_{31}, & i_{48} \mapsto i_{44}, & i_{52} \mapsto i_{44}. \end{array}$$

If we replace the pre-images in each partition class of $\mathfrak{N}_{\{\text{Pr}\}}(v_{\text{Item}})$ by its ancestor according to $\alpha_{v_{\text{Purchase}}}(P_{T'_{\text{Purchase}}}(v_{\text{Item}}))$ then we get the sets $\{i_5, i_{18}, i_{31}, i_{31}, i_{44}, i_{44}\}$ and $\{i_5, i_{18}\}$. The first set indicates that i_5 and i_{18} each has one descendant which is a pre-image of v_{Item} belonging to $\{\{i_9, i_{25}, i_{35}, i_{39}, i_{48}, i_{52}\}$ while i_{31} and i_{44} each has two such descendants. Therefore the set $\{i_5, i_{18}, i_{31}, i_{31}, i_{44}, i_{44}\}$ gives rise to the card-partition $\{\{i_5, i_{18}\}, \{i_{31}, i_{44}\}\}$. The second set yields the card-partition $\{\{i_5, i_{18}\}\}$. Composition of the two card-partition then results in

$$\mathfrak{N}_{\{\text{Pr}\}}(v_{\text{Purchase}}) = \{\{i_5, i_{18}\}, \{i_{31}, i_{44}\}\}.$$

□

Remark 3.89. To compute $\pi(\alpha_v(g), g)$ with $g \in \mathfrak{N}_{\{X\}}(w)$, we can use α_v to translate the set g into a bag h . Then we can gather those pre-images of v which occur with the same multiplicity in h to form the partition classes of $\pi(\alpha_v(g), g)$. □

We have just established the recursive step to compute $\mathfrak{N}_{\{X\}}(v)$ from $\mathfrak{N}_{\{X\}}(\eta(U))$. Note that thus far, we have only discussed how to compute non-null partitions for essential subgraphs. But an essential v -subgraph X may not be an essential $\eta(U)$ -subgraph for any v -unit U . For conciseness, assume X is contained in some specific v -unit U . Then, after computing the non-null partition induced by every essential $\eta(U)$ -subgraph we still do not obtain $\mathfrak{N}_{\{X\}}(\eta(U))$. Fortunately X must be the union of certain essential $\eta(U)$ -subgraphs and we can compute $\mathfrak{N}_{\{X\}}(\eta(U))$ from their non-null partitions. This is possible by applying Proposition 3.93, which we will prove in the following.

Lemma 3.90. $\mathfrak{N}_{\mathcal{X}}(v)$ is the composition of the non-null partitions $\mathfrak{N}_{\{X\}}(v)$ with $X \in \mathcal{X}$.

Proof. By definition, $\Pi_{\mathcal{X}}(v)$ is the product of the total partitions $\Pi_{\{X\}}(v)$ with $X \in \mathcal{X}$, and $\perp_{\mathcal{X}}(v)$ is the intersection of the null classes $\perp_{\{X\}}(v)$ with $X \in \mathcal{X}$. □

Corollary 3.91. $\Pi_{\mathcal{X}}(v)$ is the composition of the total partitions $\Pi_{\{X\}}(v)$ with $X \in \mathcal{X}$.

Remark 3.92. Corollary 3.91 provides yet another alternate approach to deriving v -agree sets projected to essential v -subgraphs from the partition database of some data tree. Every subset of partitions in the stripped partition database yield a potential v -agree set but it is easy to see that we only need to consider core sets. For each subset \mathcal{X} of essential v -subgraphs we compute the agree-partition by composing the agree-partitions for v -subgraphs in \mathcal{X} as stated in the previous corollary. After computing the partitions induces by every potential v -agree sets we can identify the representative subset of \sqsubseteq -maximal v -subgraphs of each agree set of T' . □

Proposition 3.93. Let \mathcal{X} consist of mutually v -reconcilable v -subgraphs. Then $\mathfrak{N}_{\{\cup \mathcal{X}\}}(v)$ is the composition of the non-null partitions $\mathfrak{N}_{\{X\}}(v)$ with $X \in \mathcal{X}$.

Proof. Observe that if X, Y, Z are mutually v -reconcilable v -subgraphs then so are X and $Y \cup Z$. Hence it remains to verify the claim holds for two-element sets \mathcal{X} . By Lemma 3.90, the composition of the non-null partitions $\mathfrak{N}_{\{X\}}(v)$ with $X \in \mathcal{X}$ results in $\mathfrak{N}_{\mathcal{X}}(v)$. The last part of the proof is to show that $\mathfrak{N}_{\mathcal{X}}(v) = \mathfrak{N}_{\{\cup \mathcal{X}\}}(v)$.

As $\mathfrak{N}_{\mathcal{X}}(v)$ is obtained by removing $\perp_{\mathcal{X}}(v)$ from $\Pi_{\mathcal{X}}(v)$, two pre-images $p_1, p_2 \in P_{T'}(v)$ belongs to the same partition class if and only if for all $X \in \mathcal{X}$ their projections $p_1|_X, p_2|_X$ are property-equal. Then from Lemma 1.14 and the subgraph axiom, we infer that $p_1, p_2 \in P_{T'}(v)$ belong to the same partition class in $\Pi_{\mathcal{X}}(v)$ if and only if they belong to the same partition class in $\Pi_{\{\cup \mathcal{X}\}}(v)$. Observe that a pre-image $p \in P_{T'}(v)$ has empty projection $p|_X$ for all $X \in \mathcal{X}$ if and only if it has empty projection $p|_{\cup \mathcal{X}}$. This yields $\perp_{\mathcal{X}}(v) = \perp_{\{\cup \mathcal{X}\}}(v)$. Therefore,

$$\mathfrak{N}_{\mathcal{X}}(v) = \Pi_{\mathcal{X}}(v) - \{\perp_{\mathcal{X}}(v)\} = \Pi_{\{\cup \mathcal{X}\}}(v) - \{\perp_{\{\cup \mathcal{X}\}}(v)\} = \mathfrak{N}_{\{\cup \mathcal{X}\}}(v).$$

□

Example 3.94. To find $\mathfrak{N}_{\{\{\text{De}, \text{Pr}\}\}}(v_{Purchase})$ we first compute $\mathfrak{N}_{\{\{\text{De}, \text{Pr}\}\}}(v_{Item})$ by composing $\mathfrak{N}_{\{\text{De}\}}(v_{Item})$ with $\mathfrak{N}_{\{\text{Pr}\}}(v_{Item})$ since $\{\text{De}, \text{Pr}\}$ is not an essential v_{item} -subgraph. From Example 3.81, this results in $\mathfrak{N}_{\{\{\text{De}, \text{Pr}\}\}}(v_{Item}) = \{\{i_9, i_{35}, i_{39}, i_{48}\}, \{i_{25}, i_{52}\}, \{i_{22}\}, \{i_{13}\}\}$. Lifting the pre-image of v_{Item} to those to $v_{Purchase}$ yields the bags: $\langle i_5, i_{31}, i_{31}, i_{44} \rangle, \langle i_{18}, i_{44} \rangle, \langle i_{18} \rangle, \langle i_5 \rangle$ and corresponding card-partitions: $\{\{i_5, i_{44}\}, \{i_{31}\}\}, \{\{i_{18}, i_{44}\}\}, \{\{i_{18}\}\}$ and $\{\{i_5\}\}$. Composition of these card-partitions then gives, $\mathfrak{N}_{\{\{\text{De}, \text{Pr}\}\}}(v_{Purchase}) = \{\{i_5\}, \{i_{18}\}, \{i_{31}\}, \{i_{44}\}\}$ □

The steps towards computing the \sqsubseteq -maximal v -subgraph in agree sets from partitions are as outlined in Algorithm 3.18. We first find the set of non-null partitions induced by each essential v -subgraph with the recursive approach in Algorithm 3.16. From here we can find the null classes for each essential v -subgraph and thus the stripped partition database containing the total partitions induced by each essential v -subgraph.

Algorithm 3.17 find_agreeSets-projectedToSubgraphs(Partition)

Input: db

/* where db is the relational database representing data tree T' */

Output: $\mathfrak{S}\mathfrak{A}_{T'}(v)$

/* where $\mathfrak{S}\mathfrak{A}_{T'}(v) = \{\mathcal{A}_{\{p_1, p_2\}}(v)|_{\mathfrak{S}} \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}$ */

1. $\widehat{\Pi}(v) := \text{find_stripPartitionDB}(db)$
 2. $\mathfrak{S}\mathfrak{A}_{T'}(v) := \text{find_agreeSets-fromPartition}(\widehat{\Pi}(v))$
-

Finally, we summarise in Figure 3.7 the different steps of the pXFD discovery process. There are two phases: finding the family of agree sets and difference sets for the data tree, and applying the transversal approach to compute the canonical pXFD-cover. Each of the phases constitutes two sub-cases: one in which we deal solely with v -ancestors, and the other in which we deal solely with v -subgraphs.

Algorithm 3.18 find_stripPartitionDB

Input: db /* where db is the relational database representing data tree T' */**Output:** $\widehat{\Pi}(v)$ /* where $\widehat{\Pi}(v) := \{\widehat{\Pi}_{\{X\}}(v) \mid \Pi_{\{X\}}(v) \in \Pi(v)\}$ */

1. $\mathfrak{N}(v) := \text{find_nonnullPartitions}(v, db)$
 2. $\perp(v) := \{\perp_B(v) \mid B \text{ is a } v\text{-walk and } \mathfrak{N}_B(v) \in \mathfrak{N}(v) \text{ and } \perp_B(v) = P_{T'}(v) - \bigcup \mathfrak{N}_B(v)\}$
 3. $\widehat{\perp}(v) := \left\{ \bigcap S \mid S \subseteq \perp(v) \right\}$ // \cap -ideal from Definition 3.52
 4. $\widehat{\Pi}(v) := \{\widehat{\Pi}_{\{X\}}(v) \neq \{\} \mid \mathfrak{N}_{\{X\}}(v) \in \mathfrak{N}(v) \text{ and } \perp_{\{X\}}(v) \in \widehat{\perp}(v) \text{ and } \Pi_{\{X\}}(v) = \mathfrak{N}_{\{X\}}(v) \cup \{\perp_{\{X\}}(v)\}\}$
 5. **return** $\widehat{\Pi}(v)$
-

Algorithm 3.19 find_agreeSets

Input: Data tree T' **Output:** $\text{rep}(\mathfrak{A}_{T'}(v))$ /* where $\text{rep}(\mathfrak{A}_{T'}(v)) = \{\sqsubseteq\text{-max}(\mathcal{A}_{\{p_1, p_2\}}(v)) \mid \mathcal{A}_{\{p_1, p_2\}}(v) \in \mathfrak{A}_{T'}(v)\}$ */

1. $\mathcal{S}_k := \{(p, s_k(p)) \mid p \in P_{T'}(v) \text{ and } s_k(p) \text{ is the identifier path of } p\}$
 2. $\mathfrak{A}|_{\check{\mathbb{A}}} := \text{find_agreeSets-projectedToAncestors}(\mathcal{S}_k)$
 3. $db := \text{Create relational database representing } T'$
 4. $\mathfrak{A}|_{\check{\mathbb{S}}} := \text{find_agreeSets-projectedToSubgraphs}(db)$
 5. $\mathfrak{A} = \{\}$
 6. **for all** $p_1, p_2 \in P_{T'}$ such that $\{\} \neq \mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}} \in \mathfrak{A}|_{\check{\mathbb{A}}}$ or $\{\} \neq \mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{S}}} \in \mathfrak{A}|_{\check{\mathbb{S}}}$ **do**
 7. $\mathfrak{A} := \mathfrak{A} \cup \{\mathcal{A}_{\{p_1, p_2\}}(v) \mid \mathcal{A}_{\{p_1, p_2\}}(v) = \mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{A}}} \cup \mathcal{A}_{\{p_1, p_2\}}(v)|_{\check{\mathbb{S}}}\}$
 8. **end for**
 9. **return** \mathfrak{A}
-

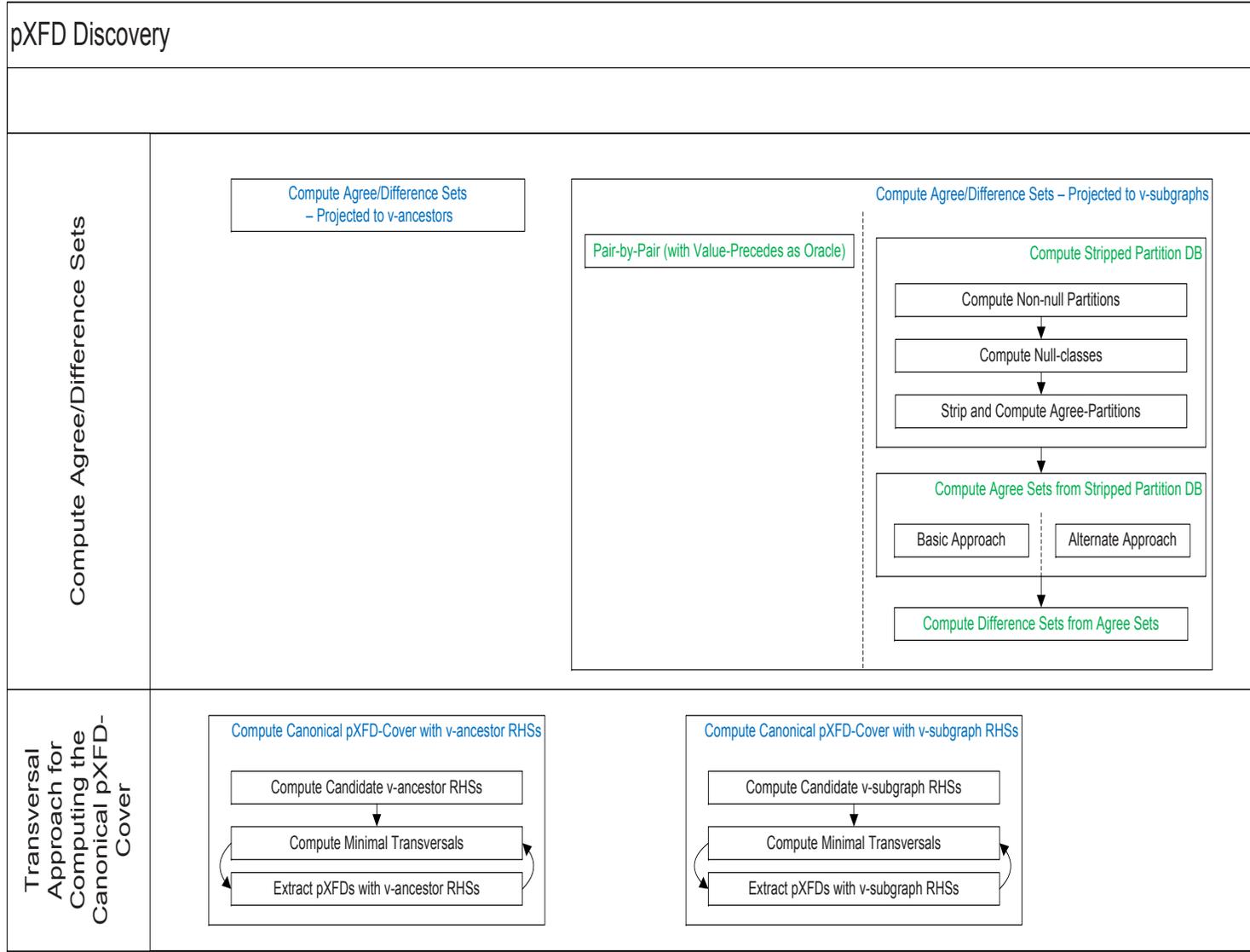


Figure 3.7: Summary of pXFD Discovery Process.

For computing the projection $\mathring{\mathcal{A}}_{T'}(v)$ of agree sets to v -ancestors we apply Algorithm 3.11(`find_agreeSets-projectedToAncestors`) which also gives us the projection of difference sets to v -ancestors for free.

For computing the projection $\mathring{\mathcal{A}}_{T'}(v)$ of agree sets to v -subgraphs we have considered two alternate approaches: pairwise comparison of pre-image trees or through computation of a stripped partition database. For the former approach, we apply Algorithm 3.12(`find_agreeSets-projectedToSubgraphs(Pairwise)`) which incidentally also gives us the projection of difference sets to v -subgraphs for free. For the latter, we apply Algorithm 3.17(`find_agreeSets-projectedToSubgraphs(Partition)`).

The partition-based approach consists of three principle steps:

1. Determine the stripped partition database,
as per Algorithm 3.18(`find_stripPartitionDB`);
2. Determine the agree sets from the partition database,
e.g., as per procedures `find_agreeSets-fromPartition` or
`find_agreeSets-fromPartition(Alternative)`;
3. Find difference sets from the agree sets,
as per Algorithm 3.6(`find_differenceSets-fromAgreeSets`).

To identify pXFDs belonging to the canonical pXFD-cover with v -ancestors and with v -subgraphs as the RHSs we apply Algorithm 3.8(`discoverXFDs-withAncestors`) and Algorithm 3.10(`discoverXFDs-withSubgraphs`) respectively. In either cases, the general idea is to interleave the extraction of pXFDs (i.e., by applying the transversal approach) and the process of determining all candidate v -ancestor/subgraph RHSs.

3.4 Summary

This chapter tackled the issue of how to discover all satisfied pXFDs in a given XML schema tree. We contended that the set of all satisfied pXFDs can be appropriately represented by the canonical pXFD-cover which is likely to be much smaller in size. A transversal approach is then suggested for the computation of the canonical pXFD-cover for any given XML schema tree: To find all pXFDs in the canonical pXFD-cover with a particular v -property Y as the RHS, we find all minimal transversals for the family of v -difference sets modulo Y . Our transversal approach bears similarity to the hypergraph transversal approach for the discovery of FDs introduced by Mannila and R  ih  .

We then discussed three important sub-problems relating to the transversal approach:

- how to find all minimal transversals for a given family of v -difference sets,
- how to find all v -difference sets of a given data tree, and
- how to find all candidate RHSs, i.e., v -properties which are the RHS of some right-maximal pXFD,

Having related the problem of finding v -difference sets to that of v -agree sets, we focused the remaining part of the chapter on the computation of all v -agree sets. We examined two possible approaches. The first approach finds the v -agree sets for each pair of pre-image trees with the help of an oracle for determining whether a pair of pre-image trees agree on a given v -subgraph. The second is a partition-based approach. Agree-partitions are proposed for representing the information about which pre-image trees agree on which v -properties. We briefly discussed two ways that v -agree sets can be assembled from the partitions. The final question addressed was how to compute the partitions themselves. For this we proposed to consider property-equal projections which are empty and those which are non-empty independently, giving rise to the notion of null classes and non-null partitions. We showed how null classes can be computed from non-null partitions and then provided a recursive approach for finding non-null partitions.

Chapter 4

Redundancy and Normal Forms

For various data models the existence of data redundancy is known to be related to problems such as insertion anomalies, deletion anomalies and modification anomalies [91] which may hinder processing. Hence being *redundancy-free* is often deemed to be an important characteristic of well-designed schema. *Normal forms* specify conditions for identifying “good” schemas at design time. The conditions are expressed over the schemas themselves, rather than looking to the data instances. In the relational data model, Boyce-Codd Normal Form (BCNF) guarantees the absence of redundancy with respect to FDs, while Fourth Normal Form (4NF) guarantees freeness from redundancy with respect to both FDs and multi-valued dependencies (MVDs). *Normalisation* is the process of decomposing a relation which is not in the desired normal into a set of relations which are. The principle problem with applying normalisation is that we may not be able to preserve all FDs, in which case, enforcing constraints becomes more resource intensive. Third Normal Form (3NF) has been proposed as a compromise: tolerate some redundancy in order to preserve FDs. For any set Σ^r of FDs we are guaranteed to find - by the process of *synthesis* - a set of relations which are in 3NF and preserve all FDs implied by Σ^r .

There are two popular notions of redundancy with respect to functional dependencies: fact redundancy and value redundancy. *Fact redundancy* treats each functional dependency as representing some fact. A given data instance is said to contain fact redundancy if it contains multiple occurrences of the same fact. Specifically, in the relational data model, there are more than one occurrences of some fact if we can find two or more tuples whose projections to the LHS and RHS of the corresponding FD are the same. With *value redundancy*, we check whether some individual value in a data instance is redundant. A value is said to be redundant if there is a functional dependency such that every modification of the value into any other value in the domain leads to a violation of the functional dependency. A schema is said to contain fact/value redundancy if any of its compatible data instance contain fact/value redundancy. In the relational data model, an example of fact and value redundancy can be found in [91] and [74] respectively.

Redundancy has also been an important motivation for many XFD proposals. There are several investigations into normal forms and normalisation of XML, for example, [3, 53, 64, 66, 73, 80, 92, 98, 103]. The most ubiquitous XML normal form (XNF) is that

of Arenas and Libkin [2, 3]:

A Document Type Definition (DTD) D is said to be in XNF with respect to a set Σ of AL-XFDs if and only if $S \rightarrow p$ is implied by Σ for every non-trivial $S \rightarrow p.@l$ in Σ .

With element nodes being compared on the basis of node identity, we can see that the condition is paramount to S being a kind of key for the element nodes reachable by following path p . Two algorithms for normalising DTDs which are not in XNF are also provided. Although the article motivates the idea of redundancy with several examples, no formal definition of redundancy is provided. Instead, the appropriateness of the proposed XNF is justified by showing that it generalises two popular normal forms from the relational data model: BCNF and Nested Normal Form. Follow up work provides an alternate justification for the correctness of XNF from an information-theoretic point of view [4]. The information-theoretic measure has also been applied in the investigation of X3NF [53]. However, few results have been published about our ability to reason about AL-XFDs and, to the best of our knowledge, there is no syntactic characterisation of trivial AL-XFDs.

More recently, Yu and Jagadish have proposed a generalisation of AL-XFDs with set semantics together with GTT-XNF as a generalisation of Arenas and Libkin's XNF [102, 103]. Instead of checking non-trivial XFDs, GTT-XNF requires the LHSs of all 'interesting XFDs' to be keys. The notion of keys however deviates from keys which are generalised by AL-XFDs. Also of note is that redundancy is defined on the basis of key satisfaction.

In this chapter, we generalise both fact and value redundancy with respect to pXFDs. Unlike for FDs, the two notions no longer coincide - more specifically, fact redundancy with respect to pXFDs implies value redundancy with respect to pXFDs, but not vice versa.

4.1 Fact Redundancy and FR-pXNF

We consider each pXFD to represent some fact and projection of every pre-image tree to the union of all v -properties in the pXFD is an instantiation of such a fact. When two or more distinct pre-image trees can contain the same instantiation of a fact then this indicates redundancy because only one of the pre-image tree is sufficient. Trivial pXFDs can naturally be disregarded because they correspond to inherent structural constraints imposed by the schema. Recall the characterisation of trivial pXFDs from Theorem 3.9.

For the rest of the chapter, let σ denote the pXFD $v : \mathcal{X} \rightarrow \mathcal{Y}$, unless stated otherwise.

Definition 4.1 (fact redundancy). Let T be an XML schema tree with set Σ of pXFDs over T . Then T is *fact-redundant* with respect to Σ if and only if there exists a T -compatible data tree T' that satisfies Σ and there are distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$ for some non-trivial pXFD $\sigma \in \Sigma$. \square

In fact, an XML schema tree is fact-redundant with respect to a set Σ of pXFDs if and only if it is fact-redundant with respect to the set Σ^* of all implied pXFDs. As a result, checking whether an XML schema tree is fact-redundant w.r.t. Σ^* can be achieved more efficiently by considering an appropriate cover set of pXFDs for Σ^* - evidently, we should choose a minimal cover of Σ^* provided one can be computed efficiently.

Theorem 4.2. *Let T be an XML schema tree with set Σ of pXFDs over T . Then T is fact-redundant with respect to Σ if and only if T is fact-redundant with respect to Σ^**

Proof. (\Rightarrow) We have $\Sigma \subseteq \Sigma^*$, and every T -compatible data tree satisfying Σ also satisfies Σ^* .

(\Leftarrow) Conversely, we need to show that if T is fact-redundant with respect to Σ^* then it is already fact-redundant with respect to Σ . For this consider a chain as follows

$$\Sigma = \Sigma_0 \subset \Sigma_1 \subset \dots \subset \Sigma_k = \Sigma^+$$

where each Σ_j results from Σ_{j-1} by one application of some inference rule from the axiomatisation $\mathfrak{F}_{general}$ given in Theorem 2.38. Recall that soundness and completeness of the inference rules means $\Sigma^* = \Sigma^+$. We proceed by induction on $k \geq 0$. When $k = 0$ then there is nothing to prove. For $k \geq 1$, we will show that for any $0 \leq j \leq k$, if there is no fact redundancy with respect to Σ_{j-1} then there is no fact redundancy with respect to Σ_j . Suppose to the contrary that there is no fact redundancy with respect to Σ_{j-1} but some fact redundancy with respect to Σ_j . Then there is some non-trivial $\sigma \in \Sigma_j - \Sigma_{j-1}$ such that we can find a T -compatible data tree T' satisfying Σ_{j-1} , having two distinct pre-image trees p_1, p_2 such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$.

Lemma 4.3 implies that σ results from application of either the extension rule or transitivity rule, since an application of any other inference rule in $\mathfrak{F}_{general}$ results in a trivial pXFD. Suppose σ results from an application of the:

extension rule, where $v : \mathcal{X} \rightarrow \mathcal{Z} \in \Sigma_{j-1}$ and $\mathcal{Y} = \mathcal{X} \cup \mathcal{Z}$.

Then T' satisfying $v : \mathcal{X} \rightarrow \mathcal{Z}$ means that $p_1|_Z \doteq p_2|_Z$ for every $Z \in \mathcal{Z}$ and so $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Z}$. Furthermore $v : \mathcal{X} \rightarrow \mathcal{Z}$ is non-trivial by Lemma 4.4 which contradicts T not being fact-redundant with respect to Σ_{j-1} .

transitivity rule, where $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are both in Σ_{j-1} .

From T' satisfying Σ_{j-1} we get $p_1|_Z \doteq p_2|_Z$ for all $Z \in \mathcal{Z}$ and so $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Z}$ and $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{Z} \cup \mathcal{Y}$. If both $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are trivial then σ is trivial by Lemma 4.5, therefore at least one must be non-trivial. Again this leads to a contradiction of T not being fact-redundant with respect to Σ_{j-1} .

□

Lemma 4.3. *Let $\Sigma \cup \{\sigma\}$ be a set of pXFDs such that $\sigma \notin \Sigma$. Then σ is trivial if $\Sigma \vdash \sigma$ by application of any of the following inference rules: reflexivity, join, simple descendant, subgraph, ancestor, target, root or empty subgraph axiom.*

Proof. Recall Theorem 3.9 which states that σ is trivial if and only if one of the following statements hold:

1. $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$ or,
2. $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

Furthermore, recall the properties of p-subsumption from Lemma 1.41 and Lemma 3.1. Suppose σ results from an application of the:

reflexivity axiom, $\left(\frac{}{v : \mathcal{X} \rightarrow \mathcal{Y}} \mathcal{Y} \subseteq \mathcal{X} \right)$

Then $\mathcal{Y} \sqsubseteq \mathcal{X}$ and $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

join axiom, $\left(\frac{}{v : \{X, Y\} \rightarrow \{X \sqcup Y\}} X, Y \text{ are } v\text{-reconcilable} \right)$

Then $\vartheta(\{X \sqcup Y\}) = \vartheta(\{X, Y\})$ and $\vartheta(\mathcal{Y}) = \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

simple descendant axiom, $\left(\frac{}{v : \{m\} \rightarrow \{n\}} n \text{ is a simple descendant of } m \right)$

Then $\vartheta(\{n\}) = \vartheta(\{m\})$ and $\vartheta(\mathcal{Y}) = \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

subgraph axiom, $\left(\frac{}{v : \{X\} \rightarrow \{Y\}} Y \text{ is contained in } X \right)$

Then $Y \sqsubseteq X$ and $\mathcal{Y} \sqsubseteq \mathcal{X}$ and $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

ancestor axiom, $\left(\frac{}{v : \{n\} \rightarrow \{m\}} m \text{ is an ancestor of } n \right)$

Then $m \sqsubseteq n$ and $\mathcal{Y} \sqsubseteq \mathcal{X}$ and $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

target axiom, $\left(\frac{}{v : \vartheta(\{v\}) \rightarrow \{U\}} U \text{ is a } v\text{-unit} \right)$

Then $\vartheta(\{v\}) = \vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$

root axiom, $\left(\frac{}{v : \{\} \rightarrow \{r_T\}} \right)$

Then $\vartheta(\mathcal{Y}) = \{r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

empty subgraph axiom, $\left(\frac{}{v : \{\} \rightarrow \{\emptyset\}} \right)$

$\vartheta(\mathcal{Y}) = \{\emptyset\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$

□

Lemma 4.4. *If $v : \mathcal{X} \rightarrow \mathcal{Y}$ is trivial then $v : \mathcal{X} \rightarrow \mathcal{X} \cup \mathcal{Y}$ is also trivial.*

Proof. Suppose $v : \mathcal{X} \rightarrow \mathcal{Y}$ is trivial. There is nothing to show for $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X})$. Suppose instead that $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$. Obviously $\vartheta(\mathcal{X}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$. Hence, by Lemma 3.1, $\vartheta(\mathcal{X} \cup \mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$ which completes the proof. □

Lemma 4.5. *If $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are both trivial then $v : \mathcal{X} \rightarrow \mathcal{Y}$ is also trivial.*

Proof. Assume $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are trivial. First, let us consider the triviality of $v : \mathcal{X} \rightarrow \mathcal{Z}$. If $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$ then we also signal the triviality of $v : \mathcal{X} \rightarrow \mathcal{Y}$. So, suppose $\vartheta(\{v\}) \not\sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$ and $\vartheta(\mathcal{Z}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$. It follows that $\vartheta(\mathcal{Z}) \cup \{r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$.

In addition, we consider the triviality of $v : \mathcal{Z} \rightarrow \mathcal{Y}$. If $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{Z}) \cup \{r_T\}$ then by transitivity of p-subsumption $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$. If instead $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{Z}) \cup \{\emptyset, r_T\}$ then also $\vartheta(\mathcal{Y}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\} \cup \{\emptyset, r_T\} = \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$. In all cases, $v : \mathcal{X} \rightarrow \mathcal{Y}$ is trivial by Theorem 3.9. \square

Definition 4.1 is a semantic notion which requires us to examine *every* T -compatible data trees when checking for fact-redundancy. This is impractical, especially since there is an infinite number of T -compatible data trees when we consider infinite domains of values. A syntactic characterisation for fact redundancy in the following is thus more useful.

Since every pre-image tree is uniquely identified by its pre-image of v , intuitively we can verify the absence/existence of fact redundancy by checking whether the LHS of each implied non-trivial pXFD also determine the target v . This motivates the following definition of normal form. That the proposed normal form correctly characterise fact redundancy is formally shown in Theorem 4.7.

Definition 4.6 (FR-pXNF). A schema tree T is said to be in *FR-pXNF* with respect to Σ if and only if for every non-trivial $\sigma \in \Sigma^*$ we have $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$ \square

Theorem 4.7. *Let T be an XML schema tree with set Σ of pXFDs over T . Then T is not fact-redundant with respect to Σ if and only if T is in FR-pXNF with respect to Σ .*

Proof. (\Leftarrow) Assume T is fact-redundant with respect to Σ . Then there exists a T -compatible data tree T' satisfying all pXFDs in Σ^* with two distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$ for some non-trivial $\sigma \in \Sigma^*$. It follows that $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$ because T' can testify to its violation. Therefore T is not in FR-pXNF.

(\Rightarrow) Assume T is not in FR-pXNF with respect to Σ . Then there is some non-trivial $\sigma \in \Sigma^*$ such that $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. Then we can construct a two- v -pre-image T -compatible data tree T' satisfying Σ but violating $v : \mathcal{X} \rightarrow \{v\}$ (see Theorem 2.7[Semantic Equivalence Theorem for pXFDs]). Since $\sigma \in \Sigma^*$ and the two distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ must obey $p_1|_X \doteq p_2|_X$ for all $X \in \mathcal{X}$ in order to violate $v : \mathcal{X} \rightarrow \{v\}$, we see that T' has fact redundancy with respect to Σ^* . Hence T is fact-redundant with respect to Σ^* . \square

Example 4.8. Let

$$\Sigma = \left\{ \begin{array}{l} v_{partners} : \{v_{class}, \mathbf{Bo}\} \rightarrow \{\mathbf{Gi}\} \\ v_{partners} : \{v_{class}, \mathbf{Gi}\} \rightarrow \{\mathbf{Bo}\} \\ v_{partners} : \{\mathbf{Bo}, \mathbf{Gi}\} \rightarrow \{v_{class}\} \end{array} \right\}$$

be a set of pXFDs over the dance school schema tree. In fact, this is the set Σ from Example 2.1.

All three pXFDs are non-trivial. However $v_{partners} : \{v_{class}, \mathbf{Bo}\} \rightarrow \{v_{partners}\} \notin \Sigma^*$. Therefore the dance school schema is not in FR-pXNF with respect to Σ . Actually, we do not have $v_{partners} : \{v_{class}, \mathbf{Gi}\} \rightarrow \{v_{partners}\} \in \Sigma^*$ or $v_{partners} : \{\mathbf{Bo}, \mathbf{Gi}\} \rightarrow \{v_{partners}\} \in \Sigma^*$ either. \square

Analogous to Theorem 4.2, we want to be able to check whether an XML schema tree is in FR-pXNF by simply examining all non-trivial pXFDs in Σ rather than all implied non-trivial pXFDs. This is shown to be possible by the next theorem.

Theorem 4.9. *Let T be an XML schema tree with set Σ of pXFDs over T . Then T is in FR-pXNF with respect to Σ if and only if for every non-trivial pXFD $\sigma \in \Sigma$ we have $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$.*

Proof. (\Rightarrow) Since $\Sigma \subseteq \Sigma^*$, if there is some non-trivial pXFD $\sigma \in \Sigma$ such that $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$ then $\sigma \in \Sigma^*$ and T is not in FR-pXNF with respect to Σ .

(\Leftarrow) For the converse, we show that if T is not in FR-pXNF with respect to Σ then there is already some non-trivial $\sigma \in \Sigma$ such that $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. Consider a chain

$$\Sigma = \Sigma_0 \subset \Sigma_1 \subset \dots \subset \Sigma_k = \Sigma^+$$

where each Σ_j results from Σ_{j-1} by one application of some inference rule from the axiomatisation $\mathfrak{F}_{general}$ in Theorem 2.38. We perform induction over $k \geq 0$. If $k = 0$ then $\Sigma = \Sigma^*$ and the statement is immediate. For $k > 0$, we show that for any $0 \leq j \leq k$ if there is some non-trivial $\sigma \in \Sigma_j$ with $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$ then there already exists some non-trivial $v : \mathcal{W} \rightarrow \{v\} \in \Sigma_{j-1}$ such that $v : \mathcal{W} \rightarrow \{v\} \notin \Sigma^*$. Assume there exists $\sigma \in \Sigma_j - \Sigma_{j-1}$ which is non-trivial with $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. By Lemma 4.3 we know σ is derived by application of either extension or transitivity rule. Suppose σ results from an application of the

extension rule, where $v : \mathcal{X} \rightarrow \mathcal{Z} \in \Sigma_{j-1}$ and $\mathcal{Y} = \mathcal{X} \cup \mathcal{Z}$.

Then $v : \mathcal{X} \rightarrow \mathcal{Z}$ is non-trivial, otherwise σ is trivial by Lemma 4.4. The initial assumption gives $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$.

transitivity rule, where $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are both in Σ_{j-1} .

At least one of $v : \mathcal{X} \rightarrow \mathcal{Z}, v : \mathcal{Z} \rightarrow \mathcal{Y}$ is non-trivial, otherwise σ is trivial by Lemma 4.5. If $v : \mathcal{X} \rightarrow \mathcal{Z}$ is non-trivial then the initial assumption gives $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. If instead $v : \mathcal{Z} \rightarrow \mathcal{Y}$ is non-trivial and $v : \mathcal{Z} \rightarrow \{v\} \in \Sigma^*$ then $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$ by application of the transitivity rule which contradicts our initial assumption. This leaves the case where $v : \mathcal{Z} \rightarrow \mathcal{Y}$ is non-trivial and $v : \mathcal{Z} \rightarrow \{v\} \notin \Sigma^*$, for which nothing further needs to be shown. \square

In some cases, FR-pXNF appears to be too strong. An XML schema tree may not be in FR-pXNF but at the same time may not suffer from processing difficulties such as update anomalies.

Example 4.10. For the schema tree T_{dance} consider a set Σ' consisting of a single pXFD

$$v_{partners} : \{\mathbf{Bo}, \mathbf{Gi}\} \rightarrow \{v_{class}\} \quad (pXFD6').$$

We cannot derive $v_{partners} : \{\mathbf{Bo}, \mathbf{Gi}\} \rightarrow \{v_{partners}\}$ from Σ' and so T_{dance} is not in FR-pXNF with respect to Σ' . This means there is some T -compatible data tree T' which has fact redundancy with respect Σ' . In particular, T' contains two pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \{\mathbf{Bo}, \mathbf{Gi}, v_{class}\}$. The pre-image trees p_1, p_2 do not share the same pre-image of $v_{partners}$ but do share the same pre-image of v_{class} . There is no value which is really redundant in relation to $pXFD6'$. If we do not know the pre-image tree of v_{class} which p_1 belongs to then we also do not know the pre-image tree of v_{class} which p_2 belongs to. If we change some information for the v_{class} node which p_1 belongs to, such as the semester or time, then the changed information is immediately related to p_2 and all other pre-image tree of $v_{partners}$ nested under the same v_{class} node. \square

4.2 Value Redundancy and VR-pXNF

To be able to define value redundancy, we need to ask what a “*redundant value*” is in a given data instance. Two common notions are: (1) a value which can be removed from the data instance and still be determined from the remaining data; (2) a value in the data instance such that its modification to any other value results in a violation of some previously satisfied dependency. We will adopt the second notion, but with a focus on the pXFD which becomes violated. Mainly, we seek conditions for determining whether a pXFD witnesses some value being redundant in the given XML data tree. As Example 4.10 suggests, pXFDs with only a single v -ancestor on the RHS do not witness the redundancy of any value. Clearly, nor does pXFDs which are implied by only them - this will in fact include trivial pXFDs.

Definition 4.11 (harmless pXFDs). Let T be an XML schema tree with set Σ of pXFDs over T . We called a singular pXFD $v : \mathcal{X} \rightarrow \{n\}$, where n is a v -ancestor, *nodal*. Let Σ_{nodal} be the set of all nodal pXFDs in Σ^* . We call a pXFD σ *harmless* (with respect to Σ) if and only if $\sigma \in \Sigma_{nodal}^*$. \square

From the pXFD inference rules in $\mathfrak{F}_{general}$, it is clear that all trivial pXFDs and those with only v -ancestors on the RHS are harmless. But, we can also say something about v -subgraphs that may occur on the RHS of harmless pXFDs. Recall that when we can determine $\vartheta(\{v\})$ from the LHS then we can in fact determine all v -properties. On the other hand, if we have a harmless pXFD whose LHS does not determine $\vartheta(\{v\})$ then any v -subgraph on the RHS can only be determined trivially. This gives us the following syntactic characterisation for harmless pXFDs:

Lemma 4.12. *Let σ be a pXFD over T . Then σ is harmless (i.e., $\sigma \in \Sigma_{nodal}^*$) if and only if at least one of the following statements hold true:*

- $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}$, or
- $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v)$ is trivial

Proof. (\Rightarrow) Our proof is by induction over the set Σ_{nodal}^* . Consider the chain

$$\Sigma_{nodal} = \Sigma_0 \subset \Sigma_1 \subset \dots \subset \Sigma_k = \Sigma_{nodal}^+$$

where each Σ_j results from Σ_{j-1} by one application of some inference rule from the axiomatisation $\mathfrak{F}_{general}$. We proceed by induction on $k \geq 0$. At $k = 0$, $\Sigma_{nodal} = \Sigma_{nodal}^+$. Thus for every $\sigma \in \Sigma_{nodal}^+$ we have $\mathcal{Y} - \check{\mathbb{A}}_T(v) = \{\}$ and

$$\vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) = \vartheta(\{\}) = \{\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$$

That is, every $\sigma \in \Sigma_{nodal}$ satisfies the second condition of this lemma. For $k > 0$, we show that for any $0 < j \leq k$ if all pXFDs in Σ_{j-1} satisfies one of the two statements in the lemma then so does $\sigma \in \Sigma_j - \Sigma_{j-1}$. If σ was derived by application of the

reflexivity axiom, where $v : \mathcal{X} \rightarrow \mathcal{Y} \in \Sigma_{j-1}$ and $\mathcal{Y} \subseteq \mathcal{X}$.

Then $\mathcal{Y} - \check{\mathbb{A}}_T(v) \subseteq \mathcal{X}$ and $\vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$.

join axiom, subgraph axiom, target axiom or empty subgraph axiom, then

$\mathcal{Y} - \check{\mathbb{A}}_T(v) = \mathcal{Y}$ and by Lemma 4.3 σ is trivial.

simple descendant axiom, ancestor axiom or root axiom, then

$\mathcal{Y} - \check{\mathbb{A}}_T(v) = \{\}$ which is analogous to the case when $k = 0$.

extension rule, where $v : \mathcal{X} \rightarrow \mathcal{Z} \in \Sigma_{j-1}$ and $\mathcal{Y} = \mathcal{X} \cup \mathcal{Z}$.

Then

$$\begin{aligned} v : \mathcal{X} \rightarrow \vartheta(\{v\}) &\in \Sigma_{nodal}, \text{ or} \\ v : \mathcal{X} \rightarrow \mathcal{Z} - \check{\mathbb{A}}_T(v) &\text{ is trivial.} \end{aligned}$$

For the former, there is nothing to prove. From the latter, we infer

$$v : \mathcal{X} \rightarrow \mathcal{X} \cup (\mathcal{Z} - \check{\mathbb{A}}_T(v)) \text{ is trivial}$$

by Lemma 4.4. This means

$$\begin{aligned} \vartheta(\{v\}) &\sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}, \text{ or} \\ \vartheta(\mathcal{X} \cup (\mathcal{Z} - \check{\mathbb{A}}_T(v))) &\sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}. \end{aligned}$$

The first case immediately yields $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v)$ is trivial. For the second case, observe

$$\vartheta(\underbrace{(\mathcal{X} \cup \mathcal{Z})}_{\mathcal{Y}} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{X} \cup (\mathcal{Z} - \check{\mathbb{A}}_T(v))) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$$

that is to say, $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v)$ is trivial.

transitivity rule, where $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are both in Σ_{j-1}

Then

$$(a1) \quad v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}, \text{ or}$$

$$(a2) \quad v : \mathcal{X} \rightarrow \mathcal{Z} - \check{\mathbb{A}}_T(v) \text{ is trivial,}$$

and

$$(b1) \quad v : \mathcal{Z} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}, \text{ or}$$

$$(b2) \quad v : \mathcal{Z} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v) \text{ is trivial}$$

There is nothing further to prove for (a1). Equally obvious, in case (b1) we have $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}$ because of the transitivity rule. Thus the subcase remains where we have (a2) and (b2) in which

$$(a2.1) \quad \vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}, \text{ or}$$

$$(a2.2) \quad \vartheta(\mathcal{Z} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\},$$

and

$$(b2.1) \quad \vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{Z}) \cup \{r_T\}, \text{ or}$$

$$(b2.2) \quad \vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{Z}) \cup \{\emptyset, r_T\}$$

Case (a2.1) immediately yield $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v)$ is trivial. For case (b2.1), if $\vartheta(\{v\}) \sqsubseteq \{r_T\}$ then $\vartheta(\{v\}) = \{r_T\}$ and we easily infer $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}$. Otherwise we have $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{Z}) \sqsubseteq \mathcal{Z}$, which results in $v : \mathcal{Z} \rightarrow \vartheta(\{v\}) \in \Sigma^+$ and by transitivity rule $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}$. This leaves the case where (a2, 2) and (b, 2.2) both holds. Clearly

- (a2.2) implies $\vartheta(\mathcal{Z} - \check{\mathbb{A}}_T(v)) \cup \{\emptyset, r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$, and
- (b2.2) implies $\vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{Z} - \check{\mathbb{A}}_T(v)) \cup \{\emptyset, r_T\}$.

Thus

$$\vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{Z} - \check{\mathbb{A}}_T(v)) \cup \{\emptyset, r_T\} \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}.$$

(\Leftarrow) For the other direction, we assume each of the two statements in the lemma holds, and show that $\sigma \in \Sigma_{nodal}^*$. Suppose $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}$. Then $v : \mathcal{X} \rightarrow \{v\} \in \Sigma_{nodal}$. Remark 2.39 implies that $v : \{v\} \rightarrow \mathcal{Z} \in \Sigma_{nodal}^*$ for every set \mathcal{Z} of v -properties, including \mathcal{Y} . It then follows that $\sigma \in \Sigma_{nodal}^*$. Next suppose that the second statement in the lemma is true, that $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v)$ is trivial. This means $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$ or $\vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$. The case of $\vartheta(\{v\}) \sqsubseteq \vartheta(\mathcal{X}) \cup \{r_T\}$ leads us back to the case of $v : \mathcal{X} \rightarrow \vartheta(\{v\})$ above. On the other hand $\vartheta(\mathcal{Y} - \check{\mathbb{A}}_T(v)) \sqsubseteq \vartheta(\mathcal{X}) \cup \{\emptyset, r_T\}$ implies that $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v) \in \{\}^* \subseteq \Sigma_{nodal}^*$. Since $v : \mathcal{X} \rightarrow \{n\} \in \Sigma_{nodal} \subseteq \Sigma_{nodal}^*$ where $n \in \mathcal{Y} \cap \check{\mathbb{A}}_T(v)$ we obtain $\sigma \in \Sigma_{nodal}^*$ by application of the union rule. \square

Value redundancy is defined in much the same way as fact redundancy, but the condition ignores all harmless pXFDs rather than just the trivial ones.

Definition 4.13 (value redundancy). Let T be an XML schema tree with set Σ of pXFDs over T . Then T is *value-redundant* with respect to Σ if and only if there exists a T -compatible data tree T' that satisfies Σ and there are distinct $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$ for some $\sigma \in \Sigma$ which is not harmless. \square

Again, we can show that value redundancy with respect to a set of pXFDs can be checked with any cover, rather than the set of all implied pXFDs. The next theorem is analogous to Theorem 4.2

Theorem 4.14. *Let T be an XML schema tree with set Σ of pXFDs over T . Then T is value-redundant with respect to Σ if and only if T is value-redundant with respect to Σ^* .*

Proof. (\Rightarrow) Follows from $\Sigma \subseteq \Sigma^*$, and every T -compatible data tree satisfying pXFDs in Σ also satisfies pXFDs in Σ^* .

(\Leftarrow) Conversely, we need to show that if T is value-redundant with respect to Σ^* then it is already value-redundant with respect to Σ . For this consider a chain as follows

$$\Sigma = \Sigma_0 \subset \Sigma_1 \subset \dots \subset \Sigma_k = \Sigma^+$$

where each Σ_j results from Σ_{j-1} by one application of some inference rule from the axiomatisation $\mathfrak{F}_{general}$. We perform induction over $k \geq 0$. If $k = 0$ then $\Sigma = \Sigma^+$ and nothing further needs to be proven. For $k > 0$, we show that if T is value-redundant with respect to Σ_j for any $0 \leq j \leq k$ then T is already value-redundant with respect to Σ_{j-1} .

Assume to the contrary that T is value-redundant with respect to Σ_j but not value-redundant with respect to Σ_{j-1} . Then $\sigma = \Sigma_j - \Sigma_{j-1}$ is not harmless and there is a T -compatible data tree T' satisfying Σ_j with two distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$. From T' satisfying all pXFDs in Σ_j and $\Sigma_{j-1} \subset \Sigma_j$ we have that T' also satisfies all pXFDs in Σ_{j-1} .

All trivial pXFDs are harmless, therefore σ was derived by application of either extension or transitivity rule (Lemma 4.3). If σ results from an application of the

extension rule, where $v : \mathcal{X} \rightarrow \mathcal{Z} \in \Sigma_{j-1}$ and $\mathcal{Y} = \mathcal{X} \cup \mathcal{Z}$.

Then $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Z}$. Moreover $v : \mathcal{X} \rightarrow \mathcal{Z}$ is not harmless, otherwise σ is harmless. Therefore T is value-redundant with respect to Σ_{j-1} , a contradiction.

transitivity rule, where $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are both in Σ_{j-1} .

So $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Z}$ and $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{Z} \cup \mathcal{Y}$. If both $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are harmless then σ is harmless. Therefore one of them is not harmless. This contradicts T not being value-redundant with respect to Σ_{j-1} .

\square

We next proffer a normal form which characterises value redundancy.

Definition 4.15 (VR-pXNF). Let T be an XML schema tree with set Σ of pXFDs over T . Then T is in VR-pXNF with respect to Σ if and only if for every $\sigma \in \Sigma^*$ which is not harmless we have $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$. \square

Theorem 4.16. Let T be an XML schema tree with set Σ of pXFDs over T . Then T is not value-redundant with respect to Σ if and only if T is in VR-pXNF with respect to Σ .

Proof. (\Leftarrow) Assume T is not in VR-pXNF and there is some $\sigma \in \Sigma^* - \Sigma_{nodal}^*$ such that $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. Then it is possible to construct a two- v -pre-image T -compatible data tree T' such that T' satisfies Σ but violates $v : \mathcal{X} \rightarrow \{v\}$ (see Theorem 2.7). In particular, we have two distinct pre-image trees $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X}$. From T' satisfying Σ we get $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$. Hence T is value-redundant with respect to Σ^* .

(\Rightarrow) Assume T is value-redundant with respect to Σ . There exists some T -compatible data tree T' which satisfies Σ and has distinct $p_1, p_2 \in P_{T'}(v)$ such that $p_1|_W \doteq p_2|_W$ for all $W \in \mathcal{X} \cup \mathcal{Y}$ and some $\sigma \in \Sigma^* - \Sigma_{nodal}^*$. Since T' satisfies Σ^* and p_1, p_2 are distinct we find $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. This means T is not in VR-pXNF. \square

Next, we show that VR-pXNF can be checked efficiently by examining only pXFDs in Σ rather than all implied pXFDs in Σ^* . This is analogous with what we have shown for FR-pXNF in Theorem 4.9.

Theorem 4.17. Let T be an XML schema tree with set Σ of pXFDs over T . Then T is in VR-pXNF with respect to Σ if and only if for every pXFD $\sigma \in \Sigma$ which is not harmless we have $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$.

Proof. (\Rightarrow) Follows from $\Sigma \subseteq \Sigma^*$, whereby $\sigma \in \Sigma - \Sigma_{nodal}^*$ implies $\sigma \in \Sigma^* - \Sigma_{nodal}^*$.

(\Leftarrow) Consider a chain

$$\Sigma = \Sigma_0 \subset \Sigma_1 \subset \dots \subset \Sigma_k = \Sigma^+$$

where each Σ_j results from Σ_{j-1} by one application of some inference rule from the axiomatisation $\mathfrak{F}_{general}$. We perform induction over $k \geq 0$. If $k = 0$ then $\Sigma = \Sigma^+$ and there is nothing further to prove. For $k > 0$, we show that if there is some $\sigma \in \Sigma_j - \Sigma_{nodal}^*$ with $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$ for any $0 \leq j \leq k$ then there already exists some non-trivial $\sigma' \in \Sigma_{j-1} - \Sigma_{nodal}^*$ with $v : LHS_{\sigma'} \rightarrow \{v\} \notin \Sigma^*$.

Assume to the contrary, then there exists $\sigma \in (\Sigma_j - \Sigma_{j-1}) - \Sigma_{nodal}^*$ such that $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. Trivial pXFDs are harmless, therefore σ is non-trivial and was derived by application of either extension or transitivity rule (Lemma 4.3). Suppose σ results from an application of the

extension rule, where $v : \mathcal{X} \rightarrow \mathcal{Z} \in \Sigma_{j-1}$ and $\mathcal{Y} = \mathcal{X} \cup \mathcal{Z}$.

Then $v : \mathcal{X} \rightarrow \mathcal{Z} \notin \Sigma_{nodal}^*$ because $\sigma \notin \Sigma_{nodal}^*$. The initial assumption already gives $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$.

transitivity rule, where $v : \mathcal{X} \rightarrow \mathcal{Z}$ and $v : \mathcal{Z} \rightarrow \mathcal{Y}$ are both in Σ_{j-1} .

At least one of $v : \mathcal{X} \rightarrow \mathcal{Z}$, $v : \mathcal{Z} \rightarrow \mathcal{Y}$ does not belong to Σ_{nodal}^* . If $v : \mathcal{X} \rightarrow \mathcal{Z} \notin \Sigma_{nodal}^*$ then the initial assumption gives $v : \mathcal{X} \rightarrow \{v\} \notin \Sigma^*$. If instead $v : \mathcal{Z} \rightarrow \mathcal{Y} \notin \Sigma_{nodal}^*$ and $v : \mathcal{Z} \rightarrow \{v\} \in \Sigma^*$ then $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$ by application of the transitivity rule which contradicts our initial assumption. The only remaining case is $v : \mathcal{Z} \rightarrow \mathcal{Y} \notin \Sigma_{nodal}^*$ and $v : \mathcal{Z} \rightarrow \{v\} \notin \Sigma^*$, for which nothing further needs to be proven. □

In fact, the check for VR-pXNF can be simplified even further. The following theorem shows that an XML schema tree is in VR-pXNF with respect to Σ if and only if Σ contains only harmless pXFDs.

Theorem 4.18. *Let T be an XML schema tree with set Σ of pXFDs over T . Then T is in VR-pXNF with respect to Σ if and only if $\Sigma - \Sigma_{nodal}^* = \{\}$.*

Proof. (\Leftarrow) Direct consequence of Theorem 4.17.

(\Rightarrow) Suppose that $\Sigma - \Sigma_{nodal}^* \neq \{\}$ but T is in VR-pXNF with respect to Σ . Then there is some $\sigma \in \Sigma - \Sigma_{nodal}^*$ such that $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$. It follows that $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma^*$ and $v : \mathcal{X} \rightarrow \vartheta(\{v\}) \in \Sigma_{nodal}$. Lemma 4.12 states that $\sigma \in \Sigma_{nodal}^*$. This gives a contradiction. □

Corollary 4.19. *Let T be an XML schema tree with set Σ of pXFDs over T . Then the following statements are equivalent*

- T is in VR-pXNF with respect to Σ
 - For every non-trivial $v : \mathcal{X} \rightarrow \mathcal{Y} - \check{\mathbb{A}}_T(v) \in \Sigma$ we have $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$
 - For every $\sigma \in \Sigma$ we have $\mathcal{Y} - \check{\mathbb{A}}_T(v) \sqsubseteq \mathcal{X} \cup \{\emptyset, r_T\}$ or $v : \mathcal{X} \rightarrow \{v\} \in \Sigma^*$,
-

In other words, the characterisation of harmless pXFDs in Lemma 4.12 says that if we specify a pXFD which is non-trivially determined subgraphs on the RHS then we must also specify the LHS of such a pXFD to determine the target node v . This fits in with our earlier idea, that RHSs which are v -ancestors do not witness the redundancy of any value.

Example 4.20.] Recall Example 4.10. We have the dance school schema tree T_{dance} and singleton set $\Sigma = \{v_{partners} : \{\mathbf{Bo}, \mathbf{Gi}\} \rightarrow \{v_{class}\}\}$. $\Sigma_{nodal} = \Sigma$ and so $\Sigma - \Sigma_{nodal}^* = \{\}$. This gives us that T_{dance} is in VR-pXNF with respect to Σ . □

In general, all trivial pXFDs are harmless but not all harmless pXFDs may be trivial. Therefore, an XML schema tree which is in FR-pXNF is also in VR-pXNF, but not necessary the reverse. Example 4.10 is a simple illustration.

4.3 Summary

In this chapter we have considered the use of pXFDs for identifying well-designed XML schema trees. We adopted the common idea that well-designed schemas should be free of redundancy. Two notions of redundancy were defined with respect to pXFDs: fact redundancy and value redundancy. While fact redundancy considers every *non-trivial* pXFDs to be representing some fact, value redundancy argues that only pXFDs which are *not harmless* can witness the existence of redundant values.

We showed that when checking each type of redundancy, it is sufficient to consider a covering set of pXFDs rather than the set of all implied pXFDs. Furthermore, we offered two normal forms: FR-pXNF and VR-pXNF for checking at design time whether an XML schema tree is free of fact and value redundancy respectively. We gave a characterisation of harmless pXFDs which makes it easy to see that trivial pXFDs are a special case of harmless pXFDs. This leads to the unsurprising finding that, although fact redundancy and value redundancy are quite similar, they do not coincide like for FDs in the relational data model. Generally speaking, FR-pXNF implies VR-pXNF but not vice versa.

Chapter 5

Conclusion and Future Work

In this thesis we have proposed pXFDs, a new class of XML functional dependency (XFDs) which generalises the class of HL-pXFDs defined on the basis of tree-homomorphism from [39]. By comparing instances of subtrees, HL-pXFDs can already express several kinds of constraints that other classes of XFDs based on paths cannot. However HL-pXFDs lack a mechanism to enforce at most one occurrence of particular subtrees which makes it difficult to express the notion of identifier. This significantly hinders progress towards normal forms for characterising data redundancy, widely considered as an important property of well-designed schema.

Our generalisation of HL-pXFDs is rather simple but powerful: in addition to subtrees, we also consider the ancestors of the target node as properties that can be used for specifying functional dependencies. The presence of these v -ancestors allows us to talk about the context in which subtrees occur (i.e., categorisations of sort), and about identifiers for nodes. Some examples were used to highlight the different sorts of constraints that we can express with pXFDs. We then showed that, without loss of generality, it is possible to consider a subclass of *canonical pXFDs* rather than the full class of pXFDs. More specifically, we proposed a refinement operator for transforming every syntactically valid pXFD into a canonical pXFD and showed that the satisfaction of one implies the other. The rest of the thesis investigated various properties and applications of pXFDs.

Firstly, the issue of reasoning is investigated. It was shown that the problem of pXFD implication, where we are to decide whether the implication $\Sigma \models \sigma$ holds, can be decided in linear time. This result was a consequence of our ability to show a semantic equivalence between pXFDs and a fragment of propositional logic. In particular, we propose a translation H of pXFDs into Horn clauses and were able to establish that

$$\Sigma \models \sigma \text{ holds if and only if } H_\sigma \text{ is a logical consequence of } H_\Sigma \cup H_T$$

We presented a semantic proof, analogous to the work of Fagin, who proved in [24] a similar semantic equivalence relating relational functional dependency (FD) implication and logical consequence of propositional logic formulas. Our preliminary study had already revealed that a similar semantic equivalence holds for HL-pXFDs [44]. With discussion of two further applications, we showed that the usefulness of the semantic equivalence

between pXFDs and propositional logic formulas extends beyond the problem of pXFD implication.

The semantic equivalence supports the processes of identifying and proving an axiomatisation for pXFDs. Starting from the translation H we acquired a sound and complete set $\mathfrak{F}_{canonical}$ of inference rules for the implication of canonical pXFDs. Then, by rendering the relationship between pXFDs and canonical pXFDs into further inference rules we were able to obtain a sound and complete set \mathfrak{F}_{prelim} of inference rules for the implication of pXFDs in general. The inference rules introduced in this preliminary axiomatisation were diffused further into simpler and more natural rules to give the elegant axiomatisation $\mathfrak{F}_{general}$ for pXFD implication. Interestingly, the axiomatisation $\mathfrak{F}_{general}$ is essentially composed of the Armstrong Axioms counterparts and additional rules for capturing inherent structural constraints enforced by the XML schema tree.

The other application for the semantic equivalence lies in providing decision support for the process of constraint acquisition. This application was initially proposed for FD acquisition in [45]. The task at hand is to develop a constraint specification which (comprehensively) represent meaningful information about the application domain. The connection with propositional logic provides us with tools for finding all necessary counter-examples for arguing that a pXFD under examination should not be specified. If we are able to say that one of the counter-example may reasonably occur as a data instance then the pXFD is not specified. If, however, all counter-examples are dismissed as being infeasible then we specify the pXFD, provided it is not implied by the other pXFDs that we have also chosen to specify.

We then turned our attention to the related task of dependency discovery, particularly, that of pXFD discovery where we try to find all pXFDs which are satisfied in a given data tree. The importance of the dependency discovery and constraint acquisition tasks is clear; we need a reasonable good set of pXFDs specification in order to make use of applications of pXFDs for example in schema design. We propose a transversal approach for pXFD discovery with the aim of computing the canonical pXFD-cover for some given data tree. The canonical pXFD-cover implies the set of all satisfied pXFDs, and therefore serves as an appropriate output. The basic ideas for our transversal approach is motivated by the works of Mannila and R  ih   [69, 71, 72]. We introduced v -difference sets as an auxiliary source of information about properties on which pairs of pre-image trees differ. The pXFDs which we are interested in are shown to be minimal transversals covering the families of v -difference sets modulo the candidate RHSs.

We also explored three problems which are closely related to the transversal approach: computation of all minimal transversals, computation of difference sets and, computation of candidate RHSs. The idea of identifying candidate RHSs rather than consider all singleton sets of v -properties as potential RHSs is new since, unlike in the relational data model, we have potential RHSs which are comparable. It is this inter-relationship among potential RHSs which enables us to further eliminate unnecessary pXFDs from the canonical pXFD-cover. We relate the problem of finding v -difference sets to that of finding the dual v -agree sets and also explore a few approaches for finding v -agree sets. We proposed to find nodes and subtrees which belongs to v -agree sets independently.

The problem of identifying subtrees which belongs to a given v -agree set proved to be the more challenging problem. We investigated two possible approaches for finding subtrees in v -agree sets, one based on comparison of every pair of pre-image trees and the other based on first computing partitions of the pre-image trees base on whether they agree on a particular property/set of properties.

Finally, we looked at one possible application for pXFDs. We identified two notions of redundancy relating to the presence of pXFDs and proposed two normal forms for characterising the absence of such redundancy. Our two notions of redundancy are based on the common notions of fact and value redundancy from the relational data model. FR-pXNF was proposed for detecting fact redundancy and VR-pXNF for detecting value redundancy. It was shown that although FR-pXNF implies VR-pXNF, the converse does not hold in general. However, both normal forms and both notions of redundancy has the nice property that we can verify them efficiently, by examining pXFDs from a cover set as opposed to all pXFDs which are implied.

Unlike other classes of XFDs in the literature, pXFDs demonstrates promising looks and feels of FDs. This may allow us to apply certain knowledge from the dependency theory for the relational data model and is likely to facilitate the adoption of pXFDs in practice.

To conclude the thesis, we identify some possible future work:

- Investigate how our proposed normal form compares with other normal forms. Do FR-pXNF and/or VR-pXNF generalise BCNF, NNF, XNF, GTT-XNF? On the one hand, we can consider the equivalence of the normal forms given specific transformations between relational schemas and XML schema trees. On the other, we can compare the information theoretic measures [4].
- Developing a normalisation algorithm and/or synthesis algorithm for transforming XML schema trees which are not in the normal forms into that of FR-pXNF and/or VR-pXNF. Further investigate whether we can do so losslessly (w.r.t. information content) and faithfully (w.r.t. dependency preservation). Is there a need for a 3NF? What might a definition of 3NF be and what is it good for?
- Define a notion of value data redundancy which is independent from the type of constraint that is provided together with the XML schema tree specification.
- Define notions of insertion and modification anomalies in the context of XML schemata and sets of XML constraints, and propose normal forms that characterize the absence of such anomalies.
- Investigate other applications of pXFD, for example, in query optimisation, data cleaning, etc.
- Investigate the existence and possible construction of Armstrong instances, one which satisfies precisely a set of pXFDs and its implications but no other pXFDs. Can we follow the approaches in the relational data models, e.g., [61]?

- What is a “good” two- v -pre-image instance and how do we construct one? This is useful, for example, as a counter-witness to pXFD implication, for supporting constraint acquisition and possibly for construction of realistic Armstrong instances.
- Implement the proposed algorithms for dependency discovery, and perform a thorough performance analysis. Using such implementations, investigate which classes of XFDs are most relevant for which application domains.
- Investigate alternative pXFD discovery solutions. For example, can we systematically explore the set of all syntactically valid pXFD in a level-wised manner - what is an adequate enumeration of the search space, what are good pruning rules, how do we efficiently check pXFD satisfaction? Implement and test the efficiency of our algorithms for comparison.
- Investigate approximate pXFDs, their discovery and possible role, for example, in data cleaning.
- Investigates the implication and possible axiomatisation of pXFDs with different target nodes.
- Are there other uses for v -agree sets? For example in constructing Armstrong instances, in verifying normal forms, in inferring keys?
- Apply the concept of ancestor properties to data dependencies in complex-value databases.
- Extend the class of pXFDs to Boolean dependencies, or also classes of join dependencies, and investigate the associated axiomatisability and implication problem.

Bibliography

- [1] ABITEBOUL, S., HULL, R., AND VIANU, V. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1995.
- [2] ARENAS, M., AND LIBKIN, L. A normal form for XML documents. In *PODS (2002)*, ACM, pp. 85–96.
- [3] ARENAS, M., AND LIBKIN, L. A normal form for XML documents. *ACM Trans. Database Syst.* 29, 1 (2004), 195–232.
- [4] ARENAS, M., AND LIBKIN, L. An information-theoretic approach to normal forms for relational and XML data. *J. ACM* 52, 2 (2005), 246–283.
- [5] ARMSTRONG, W. W. Dependency structures of data base relationships. In *IFIP Congress (1974)*, pp. 580–583.
- [6] BAILEY, J., MANOUKIAN, T., AND RAMAMOHANARAO, K. A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns. In *ICDM (2003)*, pp. 485–488.
- [7] BAIXERIES, J. A formal concept analysis framework to model functional dependencies. In *Workshop on Mathematical Methods for Learning (2004)*.
- [8] BAUCKMANN, J., LESER, U., NAUMANN, F., AND TIETZ, V. Efficiently detecting inclusion dependencies. In *ICDE (2007)*, IEEE, pp. 1448–1450.
- [9] BEERI, C., DOWD, M., FAGIN, R., AND STATMAN, R. On the structure of armstrong relations for functional dependencies. *J. ACM* 31, 1 (1984), 30–46.
- [10] BELL, S., AND BROCKHAUSEN, P. Discovery of constraints and data dependencies in databases. In *European Conference on Machine Learning (1995)*, pp. 267–270.
- [11] BERGE, C. *Hypergraphs*. Elsevier, 1989.
- [12] BUNEMAN, P., DAVIDSON, S. B., FAN, W., HARA, C. S., AND TAN, W. C. Keys for XML. In *WWW (2001)*, pp. 201–210.
- [13] BUNEMAN, P., DAVIDSON, S. B., FAN, W., HARA, C. S., AND TAN, W. C. Reasoning about keys for XML. *Inf. Syst.* 28, 8 (2003), 1037–1063.

- [14] BÜNING, H. K., AND LETTMANN, T. *Propositional Logic: Deduction and Algorithms*. Cambridge University Press, 1999.
- [15] CAMPBELL, D. M., AND RADFORD, D. Tree isomorphism algorithms: Speed vs. clarity. *Mathematics Magazine* 64, 4 (1991), 252–261.
- [16] CODD, E. F. Further normalization of the data base relational model. *IBM Research Report RJ909* (1971).
- [17] DE MARCHI, F., LOPES, S., AND PETIT, J.-M. Unary and n -ary inclusion dependency discovery in relational databases. *J. Intell. Inf. Syst.* 32, 1 (2009), 53–73.
- [18] DEMETROVICS, J., AND THI, V. D. Relations and minimal keys. *Acta Cybern.* 8 (1988), 279–285.
- [19] DEMETROVICS, J., AND THI, V. D. Some remarks on generating Armstrong and inferring functional dependencies relation. *Acta Cybern.* 12, 2 (1995), 167–180.
- [20] DONG, G., AND LI, J. Mining border descriptions of emerging patterns from dataset pairs. *Knowl. Inf. Syst.* 8, 2 (2005), 178–202.
- [21] DOWLING, W. F., AND GALLIER, J. H. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Log. Program.* 1, 3 (1984), 267–284.
- [22] EITER, T., AND GOTTLOB, G. Identifying the minimal transversals of a hypergraph and related problems. *SIAM J. Comput.* 24, 6 (1995), 1278–1304.
- [23] EITER, T., AND GOTTLOB, G. Hypergraph transversal computation and related problems in logic and AI. In *European Conference on Logics in Artificial Intelligence (JELIA)* (2002), vol. 2424 of *Lecture Notes in Computer Science*, Springer, pp. 549–564.
- [24] FAGIN, R. Functional dependencies in a relational data base and propositional logic. *IBM Journal of Research and Development* 21, 6 (1977), 543–544.
- [25] FAGIN, R. Armstrong databases. In *Symposium on Mathematical Foundations of Computer Science* (1982), IBM.
- [26] FAKHRAHMAD, S. M., SADREDDINI, M. H., AND JAHROMI, M. Z. AD-Miner: A new incremental method for discovery of minimal approximate dependencies using logical operations. *Intell. Data Analys.* 12, 6 (2008), 607–619.
- [27] FAN, W. XML constraints: Specification, analysis, and applications. In *DEXA Workshops* (2005), IEEE, pp. 805–809.
- [28] FAN, W. Dependencies revisited for improving data quality. In *PODS* (2008), ACM, pp. 159–170.

- [29] FAN, W., GEERTS, F., AND JIA, X. A revival of integrity constraints for data cleaning. *PVLDB* 1, 2 (2008), 1522–1523.
- [30] FLACH, P. A., AND SAVNIK, I. Database dependency discovery: A machine learning approach. *AI Communications* 12, 3 (1999), 139–160.
- [31] GOTTLOB, G. Hypergraph transversals. In *FoIKS* (2004), vol. 2942 of *Lecture Notes in Computer Science*, Springer, pp. 1–5.
- [32] GOTTLOB, G., AND LIBKIN, L. Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. *Acta Cybernetica* 9, 4 (1990), 385–402.
- [33] GRAHNE, G., AND ZHU, J. Discovering approximate keys in XML data. In *CIKM* (2002), pp. 453–460.
- [34] GRUMBERG, O., SCHUSTER, A., AND YADGAR, A. Memory efficient all-solutions SAT solver and its application for reachability analysis. In *International Conference on Formal Methods in Computer-Aided Design (FMCAD)* (2004), Springer, pp. 275–289.
- [35] GUNOPULOS, D., KHARDON, R., MANNILA, H., SALUJA, S., TOIVONEN, H., AND SHARM, R. S. Discovering all most specific sentences. *ACM Trans. Database Syst.* 28, 2 (2003), 140–174.
- [36] HAGEN, M. Lower bounds for three algorithms for the transversal hypergraph generation. In *WG* (2007), vol. 4769 of *Lecture Notes in Computer Science*, Springer, pp. 316–327.
- [37] HAGEN, M. Lower bounds for three algorithms for transversal hypergraph generation. *Discrete Appl. Math.* 157, 7 (2009), 1460–1469.
- [38] HARTMANN, S., KÖHLER, H., LINK, S., TRINH, T., AND WANG, J. On the notion of an XML key. In *SDKB* (2008), vol. 4925 of *Lecture Notes in Computer Science*, Springer, pp. 103–112.
- [39] HARTMANN, S., AND LINK, S. More functional dependencies for XML. In *ADBIS* (2003), vol. 2798 of *Lecture Notes in Computer Science*, Springer, pp. 355–369.
- [40] HARTMANN, S., AND LINK, S. Deciding implication for functional dependencies in complex-value databases. *Theor. Comput. Sci.* 364, 2 (2006), 212–240.
- [41] HARTMANN, S., AND LINK, S. Numerical constraints for XML. In *WoLLIC* (2007), vol. 4576 of *Lecture Notes in Computer Science*, Springer, pp. 203–217.
- [42] HARTMANN, S., AND LINK, S. Characterising nested database dependencies by fragments of propositional logic. *Ann. Pure Appl. Logic* 152, 1-3 (2008), 84–106.

- [43] HARTMANN, S., LINK, S., AND KIRCHBERG, M. A subgraph-based approach towards functional dependencies for XML. In *World Multiconference on Systemics, Cybernetics and Informatics (SCI)* (2003), IIS, pp. 200–211.
- [44] HARTMANN, S., LINK, S., AND TRINH, T. Efficient reasoning about XFDs with pre-image semantics. In *DASFAA* (2007), vol. 4443 of *Lecture Notes in Computer Science*, Springer, pp. 1070–1074.
- [45] HARTMANN, S., LINK, S., AND TRINH, T. Constraint acquisition - you can chase but you cannot find. In *APCCM* (2008), vol. 79 of *CRPIT*, ACS, pp. 59–68.
- [46] HEDMAN, S. *A First Course in Logic: An Introduction to Model Theory, Proof Theory, Computability, and Complexity*. Oxford University Press, 2004.
- [47] HUHTALA, Y., KÄRKKÄINEN, J., PORKKA, P., AND TOIVONEN, H. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE* (1998), pp. 392–401.
- [48] HUHTALA, Y., KÄRKKÄINEN, J., PORKKA, P., AND TOIVONEN, H. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.* 42, 2 (1999), 100–111.
- [49] JIN, H., AND SOMENZI, F. Prime clauses for fast enumeration of satisfying assignments to Boolean circuits. In *Annual Conference on Design Automation (DAC)* (2005), ACM, pp. 750–753.
- [50] KAVVADIAS, D. J., AND STAVROPOULOS, E. C. An efficient algorithm for the transversal hypergraph generation. *J. Graph Algorithms Appl.* 9, 2 (2005), 239–264.
- [51] KHARDON, R., MANNILA, H., AND ROTH, D. Reasoning with examples: Propositional formulae and database dependencies. *Acta Inf.* 36, 4 (1999), 267–286.
- [52] KOELLER, A., AND RUNDENSTEINER, E. A. Heuristic strategies for the discovery of inclusion dependencies and other patterns. *J. Data Semantics* 5 (2006), 185–210.
- [53] KOLAH, S. Dependency-preserving normalization of relational and XML data. *J. Computer System Sciences* 73 (2007), 636–647.
- [54] KRYSZKIEWICZ, M., AND LASEK, P. FUN: Fast discovery of minimal sets of attributes functionally determining a decision attribute. *Trans. Rough Sets* 9 (2008), 76–95.
- [55] LEE, D., AND CHU, W. W. Comparative analysis of six XML schema languages. *SIGMOD Record* 29, 3 (2000), 76–87.
- [56] LEE, M.-L., LING, T. W., AND LOW, W. L. Designing functional dependencies for XML. In *EDBT* (2002), vol. 2287 of *Lecture Notes in Computer Science*, Springer, pp. 124–141.

- [57] LEVENE, M., AND LOIZOU, G. *A Guided Tour of Relational Databases and Beyond*. Springer, 1999.
- [58] LIM, W. M., AND HARRISON, J. Discovery of constraints from data for information system reverse engineering. In *Software Engineering Conference (1997)*, pp. 39–48.
- [59] LIM, W. M., AND HARRISON, J. Parallel approaches for discovering functional dependencies from data for information system design recovery. In *International Symposium on Parallel Architectures, Algorithms and Networks (I-SPAN) (1997)*, pp. 254–260.
- [60] LIU, J., VINCENT, M., AND LIU, C. Local XML functional dependencies. In *WIDM (2003)*, ACM, pp. 23–28.
- [61] LOPES, S., PETIT, J.-M., AND LAKHAL, L. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT (2000)*, Springer, pp. 350–364.
- [62] LOPES, S., PETIT, J.-M., AND LAKHAL, L. Functional and approximate dependency mining: database and FCA points of view. *J. Experim. & Theor. Artif. Intell.* 14, 2-3 (2002), 93–114.
- [63] LU, S., SUN, Y., ATAY, M., AND FOTOUHI, F. A new inlining algorithm for mapping XML DTDs to relational schemas. In *ER Workshops (2003)*, vol. 2813 of *Lecture Notes in Computer Science*, Springer, pp. 366–377.
- [64] LV, T., GU, N., AND YAN, P. Normal forms for XML documents. *Information & Software Technology* 46, 12 (2004), 839–846.
- [65] LV, T., AND YAN, P. XML constraint-tree-based functional dependencies. In *ICEBE (2006)*, IEEE, pp. 224–228.
- [66] LV, T., AND YAN, P. XML normal forms based on constraint-tree-based functional dependencies. In *APWeb/WAIM Workshops (2007)*, vol. 4537 of *Lecture Notes in Computer Science*, Springer, pp. 348–357.
- [67] MAIER, D. *The Theory of Relational Databases*. Computer Science Press, 1983.
- [68] MANNILA, H., AND RÄIHÄ, K.-J. Design by example: An application of Armstrong relations. Tech. rep., Cornell University, 1985.
- [69] MANNILA, H., AND RÄIHÄ, K.-J. Design by example: An application of Armstrong relations. *J. Computer System Sciences* 33, 2 (1986), 126–141.
- [70] MANNILA, H., AND RÄIHÄ, K.-J. Dependency inference. In *VLDB (1987)*, pp. 155–158.

- [71] MANNILA, H., AND RÄIHÄ, K.-J. *The design of relational databases*. Addison-Wesley, 1992.
- [72] MANNILA, H., AND RÄIHÄ, K.-J. Algorithms for inferring functional dependencies from relations. *Data Knowl. Eng.* 12, 1 (1994), 83–99.
- [73] MOK, W., AND EMBLEY, D. Generating compact redundancy-free XML documents from conceptual-model hypergraphs. *IEEE Trans. Knowl. Data Engin.* 18, 8 (2006), 1082–1096.
- [74] MOK, W. Y., NG, Y.-K., AND EMBLEY, D. W. A normal form for precisely characterizing redundancy in nested relations. *ACM Trans. Database Syst.* 21, 1 (1996), 77–106.
- [75] MURATA, M., LEE, D., MANI, M., AND KAWAGUCHI, K. Taxonomy of XML schema languages using formal language theory. *ACM Trans. Internet Techn.* 5, 4 (2005), 660–704.
- [76] NOVELLI, N., AND CICCETTI, R. FUN: An efficient algorithm for mining functional and embedded dependencies. In *ICDT (2001)*, vol. 1973 of *Lecture Notes in Computer Science*, Springer, pp. 189–203.
- [77] RAMAKRISHNAN, R., AND GEHRKE, J. *Database Management Systems*. McGraw-Hill, 2002.
- [78] SAGIV, Y., DELOBEL, C., D. SCOTT PARKER, J., AND FAGIN, R. An equivalence between relational database dependencies and a fragment of propositional logic. *J. ACM* 28, 3 (1981), 435–453.
- [79] SÁNCHEZ, D., SERRANO, J.-M., BLANCO, I., MARTÍN-BAUTISTA, M. J., AND MIRANDA, M. A. V. Using association rules to mine for strong approximate dependencies. *Data Min. Knowl. Discov.* 16, 3 (2008), 313–348.
- [80] SCHEWE, K.-D. Redundancy, dependencies and normal forms for XML databases. In *ADC (2005)*, vol. 39 of *CRPIT*, ACS, pp. 7–16.
- [81] SCHLIMMER, J. C. Efficiently inducing determinations: A complete and systematic search algorithm that uses optimal pruning. In *International Conference on Machine Learning (1993)*, pp. 284–290.
- [82] SHANMUGASUNDARAM, J., TUFTE, K., ZHANG, C., HE, G., DEWITT, D. J., AND NAUGHTON, J. F. Relational databases for querying XML documents: Limitations and opportunities. In *VLDB (1999)*, pp. 302–314.
- [83] SILVA, A. M., AND MELKANOFF, M. A. A method for helping discover the dependencies of a relation. In *Advances in Data Base Theory (1979)*, pp. 115–133.

- [84] SMULLYAN, R. M. *First-order logic*. Dover Publications, 1995.
- [85] TAKATA, K. A worst-case analysis of the sequential method to list the minimal hitting sets of a hypergraph. *SIAM J. Discrete Math.* 21, 4 (2007), 936–946.
- [86] TATARINOV, I., VIGLAS, S. D., BEYER, K., SHANMUGASUNDARAM, J., SHEKITA, E., AND ZHANG, C. Storing and querying ordered XML using a relational database system. In *SIGMOD* (2002), ACM, pp. 204–215.
- [87] TRINH, T. Axiomatising functional dependencies for XML with frequencies. Master’s thesis, Massey University, 2004.
- [88] TRINH, T. Using transversals for discovering XML functional dependencies. In *FoIKS* (2008), vol. 4932 of *Lecture Notes in Computer Science*, Springer, pp. 199–218.
- [89] ULLMAN, J. D. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.
- [90] VALIENTE, G. *Algorithms on Trees and Graphs*. Springer, 2002.
- [91] VINCENT, M. W. *Semantic Justification for Normal Forms in Relational Database Design*. PhD thesis, Monash University, 1994.
- [92] VINCENT, M. W., LIU, J., AND LIU, C. Redundancy free mappings from relations to XML. In *WAIM* (2004), vol. 3129 of *Lecture Notes in Computer Science*, Springer, pp. 346–356.
- [93] VINCENT, M. W., LIU, J., AND LIU, C. Strong functional dependencies and their application to normal forms in XML. *ACM Trans. Database Syst.* 29, 3 (2004), 445–462.
- [94] VON RAUTENBERG, W. *A Concise Introduction to Mathematical Logic*. Springer, 2006.
- [95] W3C. XML Path Language (XPath) version 1.0. Recommendation, W3C, 1999.
- [96] W3C. XML Schema part 0: Primer second edition. Recommendation, W3C, 2004.
- [97] W3C. Extensible Markup Language (XML) 1.0 (fourth edition). Recommendation, W3C, 2006.
- [98] WANG, J., AND TOPOR, R. W. Removing XML data redundancies using functional and equality-generating dependencies. In *ADC* (2005), pp. 65–74.
- [99] WANG, S.-L., SHEN, J.-W., AND HONG, T.-P. Incremental discovery of functional dependencies using partitions. In *IFSA World Congress and NAFIPS International Conference* (2001), vol. 3, pp. 1322–1326.

- [100] WYSS, C., GIANNELLA, C., AND ROBERTSON, E. L. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK)* (2001), Springer, pp. 101–110.
- [101] YAO, H., AND HAMILTON, H. J. Mining functional dependencies from data. *Data Min. Knowl. Discov.* 16, 2 (2008), 197–219.
- [102] YU, C., AND JAGADISH, H. V. Efficient discovery of XML data redundancies. In *VLDB* (2006), pp. 103–114.
- [103] YU, C., AND JAGADISH, H. V. XML schema refinement through redundancy detection and normalization. *VLDB J.* 17, 2 (2008), 203–223.
- [104] ZHOU, Z. Algorithms and implementation of functional dependency discovery for XML. Master thesis, Massey University, 2006.

Acknowledgements

I am grateful to my supervisor Sven Hartmann for his guidance, insights, patience and eternal optimism which has enabled me to reach the finish line. In addition to discussions about my research, he has always been a great help in many other ways including but not limited to applying for scholarships, my attendance at several memorable conferences and summer schools, and an unexpected though much need chance to experience a more independent lifestyle.

My thanks go out to Sebastian Link who was always on hand to lend motivation and support, especially for things like writing up publications and preparing for conferences.

I have also been lucky to be able to work and socialise with a great bunch of people from my former IS department. It is a shame how things ended but I very much appreciate all the great times we shared and the direct support that many of them have given me in various endeavours. In particular, I would like to thank Henning for discussions on some of my ideas about HL-XFD discovery, and Qing for her steadfast confidence and friendship.

Also, big thanks to all my other friends. Always there to make sure I live a little.

Thanks to the Tertiary Education Commission of New Zealand for the financial support that has, in part, made it possible for me to attend such interesting conferences and summer schools and, in many ways freed me from the financial worries that plagued colleagues and friends.

Finally I would like to dedicate this thesis to my parents, Khanh Tuoc Trinh and Huong Tran, and the rest of my family. I'm grateful to my dad for his calming and easy-going attitude, and to my mum for her persistence attention to details and care that have helped me through many journeys and stressful periods. My brothers and sister have always been there for me and my childhood memory is all the richer for having grown up with them.

Index

- agree-partition, 134
- ancestor, 6
 - highest simple ancestor, 17
 - immediate essential ancestor, 28
 - lowest contained v -ancestor, 18
- Armstrong databases, 65
- axiomatisation, 50–64
 - $\mathfrak{F}_{canonical}$, 52
 - $\mathfrak{F}_{general}$, 61
 - \mathfrak{F}_{prelim} , 54
- base translation, 28
- candidate RHS, 85, 103–123
- canonical pXFD, 15–21
- canonical pXFD-cover, 80, 107–110, 119–123
- complete sets, 35, 36, 112
- composition, 142
- core set, 20
- descendant, *see* ancestor
- e-closed, 110, 114, 117
- equality set, 33–35, 44
- essential
 - v -ancestor, 16
 - v -properties, 16
 - v -subgraph, 16
 - property closure, 62
- fact-redundant, *see* redundancy
- FR-pXNF, *see* normal form
- frequencies, 5
- functional dependencies, 1
 - discovery, 71–73
 - logical characterisation, 25–26
 - normal forms, 151
- harmless pXFD, 157–159, 162
- hitting set, 91, 101
- HL-XFDs, 8
- homomorphism, 5
- Horn clause, 28
 - encoding, 28–29
 - satisfiability, 47–49
- hypergraph, 72, 91, 102
- hypergraph transversal, *see* hitting set
- inference rule
 - \sqsubseteq -reflexivity, 74
 - ancestor, 50, 52, 61
 - Armstrong, 50, 52, 53, 61
 - empty subgraph, 50, 52, 53, 61
 - join, 59, 61
 - p-subsumption, 76
 - refinement/reinstatement, 54–61
 - root, 50, 52, 53, 61
 - simple descendant, 58, 61
 - subgraph, 50, 52, 61
 - target, 50, 52, 61
- isomorphism, 9, 10
- left-reduced pXFD, 75–77, 80
- logical consequence, 30
- minimal transversal, 82, 100, 104
- non-null partition, 137
- normal form
 - FR-pXNF, 155–157, 162
 - VR-pXNF, 160–162
- null class, 137

- p-subsumption, 20–75
- partition, 134
 - refine, product, residual, 141
 - support, context, exterior, 134
 - agree-partition, 134, 137–149
 - card-partition, 143
 - database, 134, 146
 - non-null partition, 137–146
 - null class, 137
 - stripped, 134
- path, 5
- pre-image tree of v , 7
- projection, 9
- property closure, 62
- property-equal, 10, 12–15, 18, 19, 21, 130–132
- propositional tableau, 66
- pXFD, 10
 - implication, derivation, 12, *see* axiomatisation, *see* semantic equivalence theorem
 - acquisition, 65–69
 - canonical, 20, *see* canonical pXFD
 - canonical cover, 80, *see* canonical pXFD-cover
 - discovery, 73–90
 - harmless, 157, *see* harmless pXFD
 - implication problem, 47–49, *see* axiomatisation
 - left-reduced, 75, *see* left-reduced pXFD
 - LHS, 10
 - nodal, 157
 - RHS, 10
 - right-maximal, 75, *see* right-maximal pXFD
 - satisfaction, 10, 80–81
 - singular, 10
 - target, 10
 - trivial, 77, *see* trivial pXFD
 - value redundancy, 159–161
- refinement, 17–20, 22–23
- representative boolean assignment, 31–35
- right-maximal pXFD, 75–77, 80, *see* candidate RHS
- semantic equivalence theorem, 29
- Semantic Equivalence Theorem for pXFDs, 35
- semantic proof, 25
- simple
 - ancestor, descendant, 6
 - node, 5
 - path, 5
 - target, 61
- strictly p-subsumes, 21
- stripped partition, 134
- syntactic proof, 26
- T -compatible, 5
- target, *see* simple
- total v -subgraph, 7
- transversal, 82
- trivial pXFD, 62, 77–80, 86–87, 107–108, 120, 153–155, 162
- two- v -pre-image data tree, 12, 29–31
 - construction, 35–47
- u-closed, 110, 112–113, 117
- v -agree set, 100, 123–136
- v -ancestor, 6
- v -difference set, 82, 105–106
 - dual, 100–103, 127–128
- v -property, 7–9
- v -reconcilable, 14
- v -subgraph, 6
- v -unit, 6, 17, 36–45, 101–103, 110–119, 138, 144
- v -walk, 6
- value-redundant, *see* redundancy
- XML tree, 4–5
 - relational representation, 138–141
- redundancy
 - fact redundancy, 152