Nachtrag (fortgesetzt)

Warum $\perp$ in Baumtupeln?

- Sei $D = (E, A, P, R, r)$ wobei $E = \{r, a, b\}$, $A = \emptyset$, $P(r) = (a|b)$, $P(a) = P(b) = \epsilon$.
- $paths(D) = \{r, r.a, r.b\}$,
- t mit $t(r.a), t(r.b) \neq \perp$ ist ein Baumtupel.
- Sei T ein XML-Baum mit $T \models D$.
- T kann jedoch nicht das Baumtupel t enthalten; entweder $t(r.a) = \perp$, oder $t(r.b) = \perp$.

XML Normalform, Teil 2

BCNF und XNF

Sei ein Relationsschema $G(A_1, \ldots, A_n)$ und eine Menge funktionaler Abhängigkeiten FD über G gegeben.
Schema (G, FD) wird eine XML-Repräsentation (D_G, Σ_{FD}) zugeordnet wie folgt.

Für $D_G = (E, A, P, R, db)$ gilt:

- $E = \{db, G\}$,
- $A = \{@A_1, \ldots, @A_n\}$,
- $P(db) = G*$ und $P(G) = \epsilon$,
- $R(db) = \emptyset$ und $R(G) = \{@A_1, \ldots, @A_n\}$.

Für Σ_{FD} gilt:

- Wenn $A_{l_1} \ldots A_{l_m} \rightarrow A_l \in FD$, dann $\{db.G.@A_{l_1}, \ldots, db.G.@A_{l_m}\} \rightarrow db.G.@A_l \in \Sigma_{FD}$.
- $\{db.G.@A_1, \ldots, db.G.@A_n\} \rightarrow db.G \in \Sigma_{FD}$. Warum auch diese?

(G, FD) ist in BCNF, gdw (D_G, Σ_{FD}) ist in XNF.

Algorithmus zur Normalisierung

Annahmen:

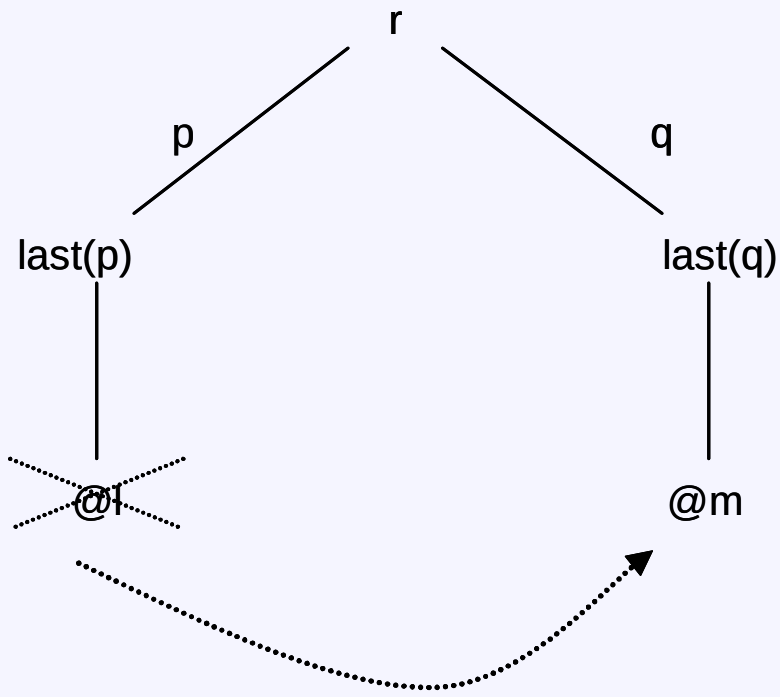
- Die DTDs sind nicht rekursiv.
- FDs sind der Form $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p$,
- S ist nicht Teil eines Pfades.
- Wenn $X \rightarrow p.@l$ die XNF verletzt, dann:
  wenn $p.@l \neq \perp$, dann auch jeder Pfad in X.

Eine nicht-triviale FD heißt *anomalous*, wenn sie die XNF verletzt.
Der auf Dekomposition basierende Normalisierungsalgorithmus entfernt nacheinander anomalous FDs. Er basiert auf den folgenden Schritten:

- Verschieben eines Attributes (vergl. Beispiel 2),
- Erzeugen eines neuen Element-Typs (vergl. Beispiel 1).

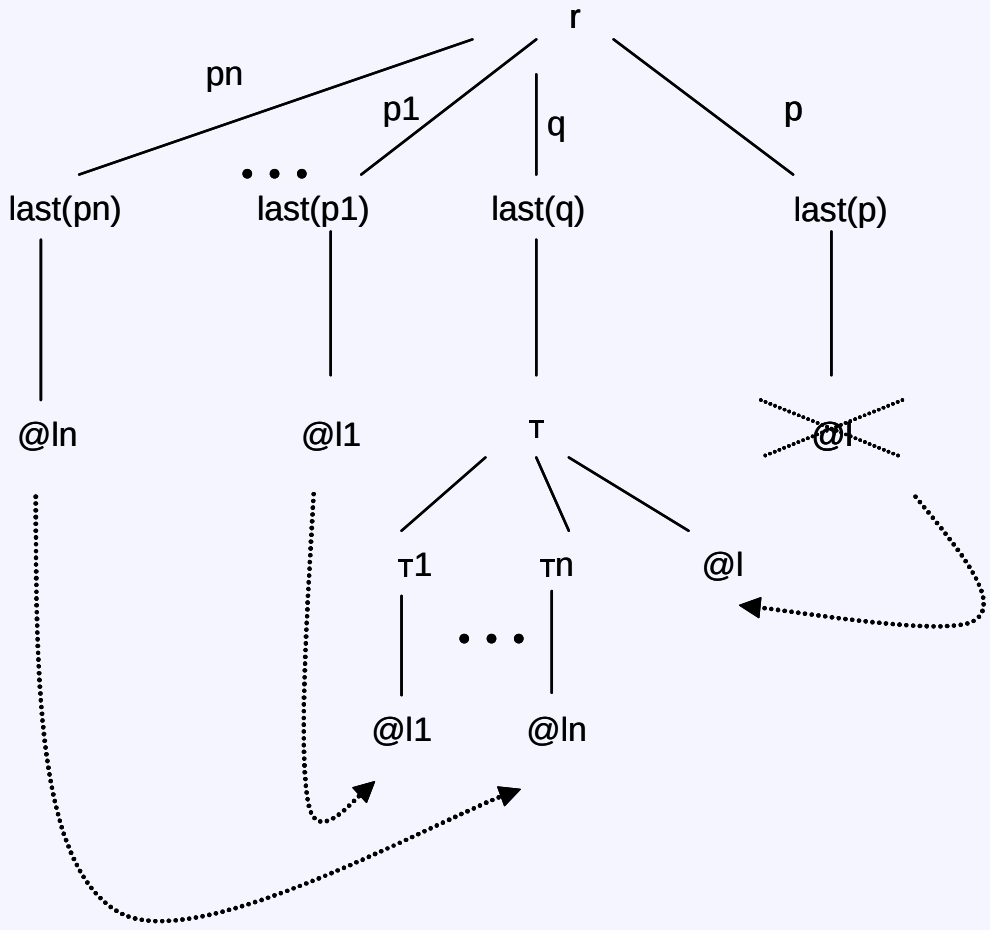
Verschieben eines Attributes

FD = $q \rightarrow p.@l$, $q \in EPaths(D)$



Erzeugen eines neuen Element-Typs

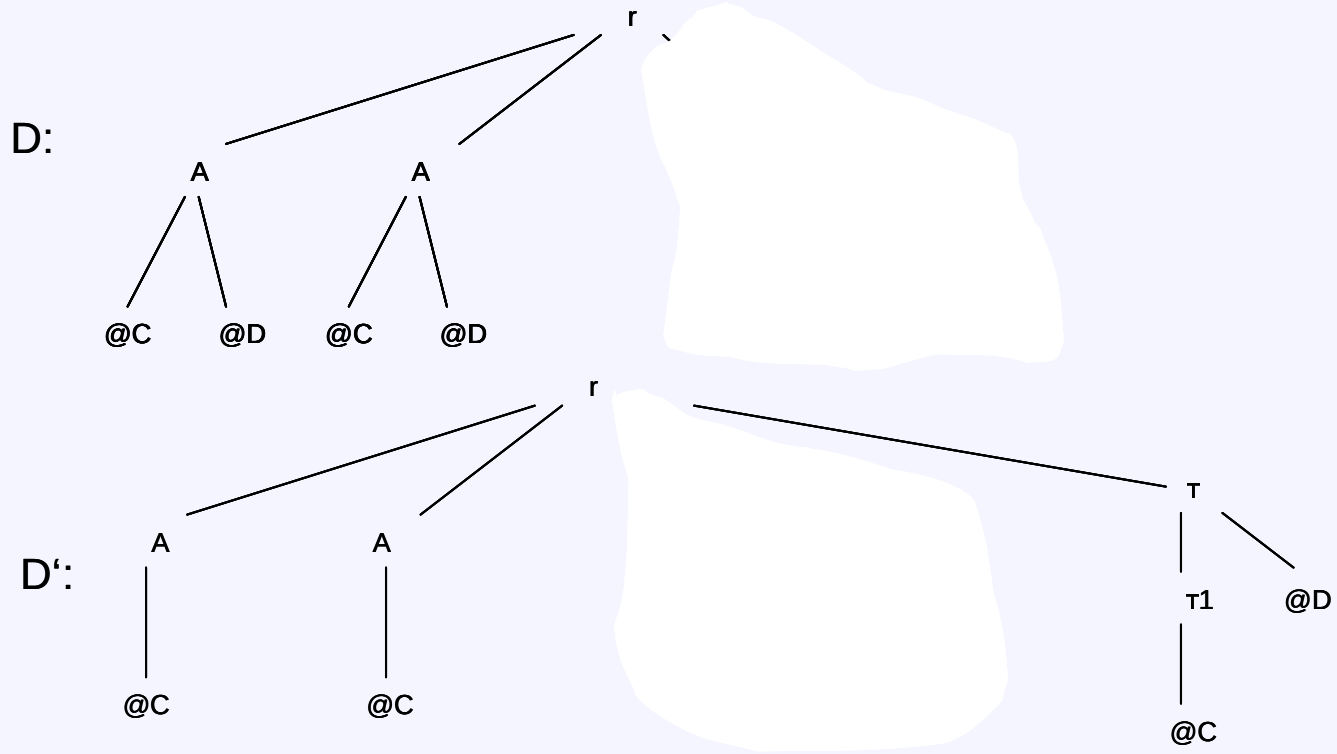
FD = $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p.@l,\ q \in EPaths(D)$



Verlustfreiheit der Dekomposition

Im relationalen Modell wird die Verlustfreiheit mittels Algebra definiert. Diese Basis existiert für XML (noch) nicht. Andere Arten der Formalisierung werden somit notwendig und möglich.

Ist D' eine verlustfreie Zerlegung von D? Betrachte zugehörige XML-Bäume T, T'.



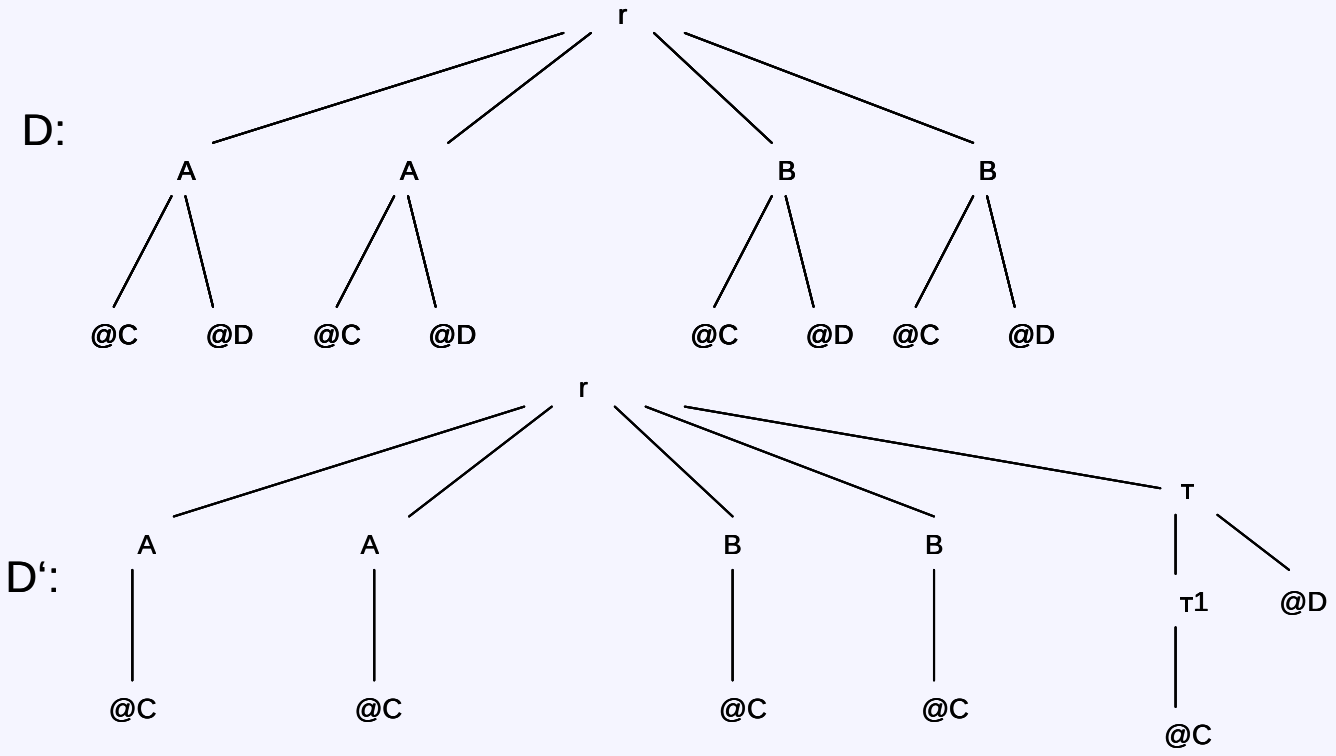
Seien $(D, \Sigma), (D', \Sigma')$ gegeben.
(D', Σ') ist eine *verlustfreie Zerlegung* von (D, Σ), $(D, \Sigma) \leq_{lossless} (D', \Sigma')$, wenn eine Abbildung f von D' nach D existiert, so dass zu jedem $T \models (D, \Sigma)$ ein $T' \models (D', \Sigma')$ existiert, so dass $T \equiv_f T'$.

- Sei $f : paths(D') \rightarrow paths(D)$ surjektiv, wobei weiter gilt: $p \in EPaths(D')$ gdw. $f(p) \in EPaths(D)$.
- Seien $t \in \mathcal{T}(D)$ und $t' \in \mathcal{T}(D')$. Wir schreiben $t \equiv_f t'$, wenn für alle $p \in paths(D') \setminus EPaths(D')$ gilt, $t'.p = t.f(p)$.
- Seien $X \subseteq \mathcal{T}(D)$ und $X' \subseteq \mathcal{T}(D')$ nicht-leere Mengen. Wir schreiben $X \equiv_f X'$, wenn zu jedem $t \in X$ ein $t' \in X'$ existiert, so dass $t \equiv_f t'$, und umgekehrt.
- Seien T, T' XML-Bäume mit $T \models D$ und $T' \models D'$. Wir schreiben $T \equiv_f T'$, wenn $tuples_D(T) \equiv_f tuples_D(T)$.

$\leq_{lossless}$ ist transitiv.

Grenzen der Formalisierung

Ist D' eine verlustfreie Zerlegung von D? Betrachte zugehörige XML-Bäume T, T'.



Ist dies ein Fehler?

Sei (D', Σ') Resultat der Normalisierung von (D, Σ) mittels Dekomposition. Es gilt $(D, \Sigma) \leq_{lossless} (D', \Sigma')$.

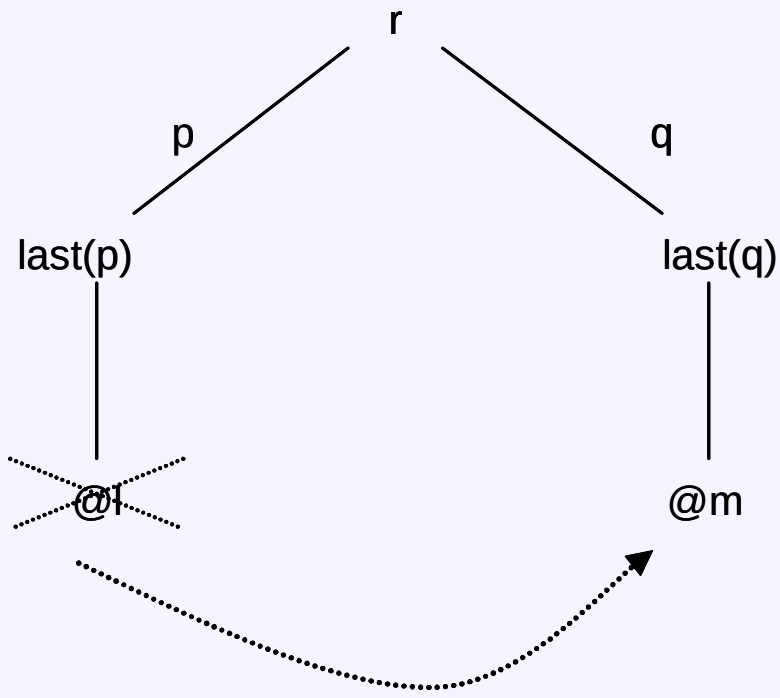
Attribut Verschiebung: FD = $q \longrightarrow p.@l$.

- $f(q.@m) = p.@l$,
- $f(p') = p$ sonst.

Element-Typ Definition: FD = $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \longrightarrow p.@l$.

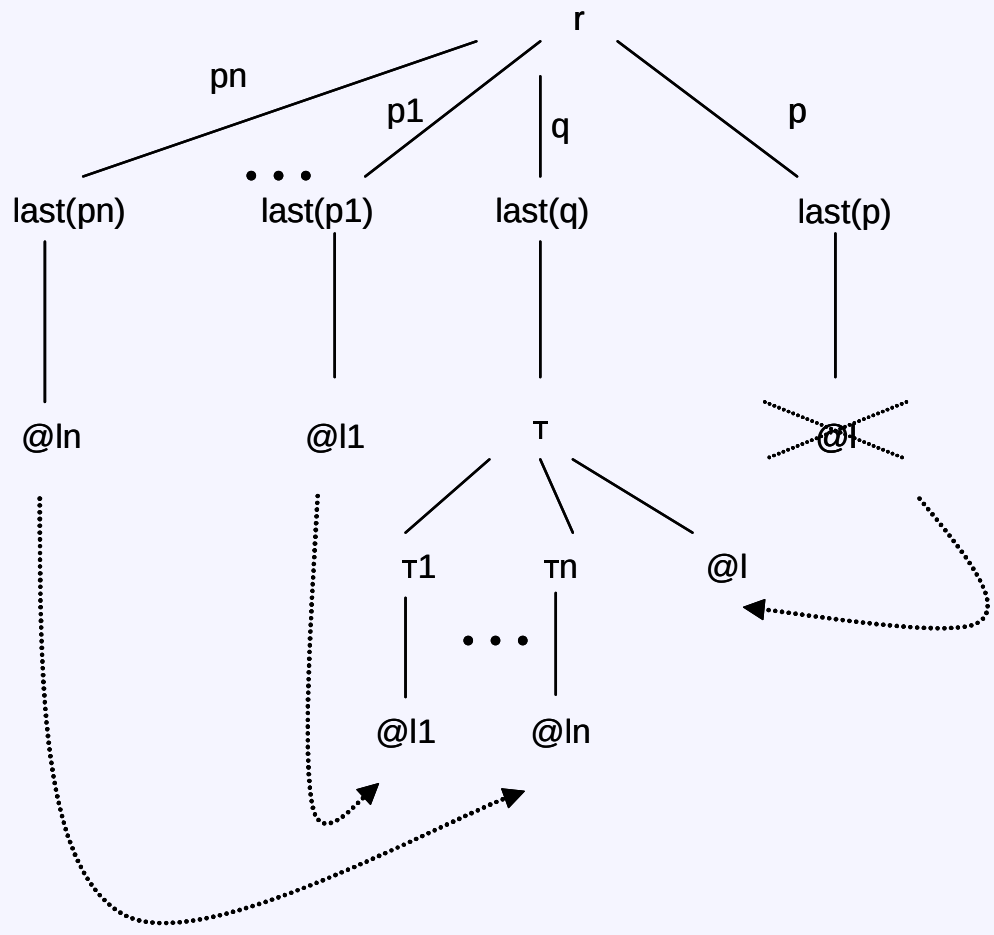
- $f(q.\tau) = p$,
- $f(q.\tau.@l) = p.@l$,
- $f(q.\tau.\tau_i) = p_i$,
- $f(q.\tau.\tau_i.@l_i) = p_i.@l_i$,
- $f(p') = p'$ sonst.

FD = $q \rightarrow p.@l$, $q \in EPaths(D)$



$f(q.@m) = p.@l$, $f(p') = p$ sonst.

FD = $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p.@l$, $q \in EPaths(D)$



$f(q.\tau) = p$, $f(q.\tau.@l) = p.@l$, $f(q.\tau.\tau_i) = p_i$, $f(q.\tau.\tau_i.@l_i) = p_i.@l_i$, $f(p') = p'$ sonst.

Kann die Attribut-Verschiebung und Elementtyp-Erzeugung mittels XQuery analog zur Projektion in der Relationenalgebra implementiert werden? Auch die inverse Richtung vergleichbar zum natürlichen Verbund? . . . im Prinzip schon.

zu Beispiel 2 (”Projektion”):

```
let $root := document("dbpl.xml")/db
return <db>
{ for $cno in $root/conf
  return <conf>
  <title> { $cno/title/text() } </title>
  { for $is in $co/issue
    let $value := $is/inproceedings[position()=1]/@year
    return <issue year="{ $value }">
    { for $in in $is/inproceedings
      return <inproceedings key="{ $in/@key }" pages="{ $in/@pages }">
      { for $au in $in/author
        return <author> { $au/text() } </author>
        <title> { $in/title/text() } </title>
      }
      </inproceedings>
    }
  </conf>
}
</db>
```

zu Beispiel 2 (”Verbund”):

```
let $root := document("dbpl.xml")/db
return <db>
{ for $cno in $root/conf
  return <conf>
  <title> { $cno/title/text() } </title>
  { for $is in $co/issue
    let $value := @year
    return <issue>
    { for $in in $is/inproceedings
      return <inproceedings key="{ $in/@key }"
                            pages="{ $in/@pages }"
                            year="{ $value }">
      { for $au in $in/author
        return <author> { $au/text() } </author>
        <title> { $in/title/text() } </title>
      }
      </inproceedings>
    }
  </conf>
}
</db>
```

Literatur

M. Arenas, L. Libkin. *A Normal Form for XML Documents.* ACM ToDS, Vol. 29, No.1, 2004