XML Normalform

- Relationale Datenbanken sind eng verbunden mit ihren Normalformen, die einen guten von einem schlechten Entwurf unterscheiden helfen.
- Existiert eine entsprechende Theorie für XML? Braucht man so etwas überhaupt?

Beispiel:
Sei $R(A, B, C)$ ein Relationsschema und sei $\mathcal{F} = \{A \rightarrow B, B \rightarrow C\}$ eine Menge funktionaler Abhängigkeiten (FDs). R ist nicht in BCNF.
Wir transformieren R in eine XML-Repräsentation gemäß der DTD:

```
<!ELEMENT db (R*)>
<!ELEMENT R EMPTY>
<!ATTLIST R A CDATA B CDATA C CDATA>
```

Offensichtlich sind die Probleme weiter präsent!

- Was bedeuten *Redundanz* und *Update-Anomalien* für XML?
- Wie können FDs definiert werden? Wann verursacht eine FD Probleme?
- Was ist eine Normalform?
- Kann ein Algorithmus gefunden werden, der eine gegebene DTD in eine Normalform transformiert?

Beispiel 1:

schlechter Entwurf:

```
<!DOCTYPE courses [
 <!ELEMENT courses (course*)>
 <!ELEMENT course (title, taken_by)>
  <!ATTLIST course cno CDATA #REQUIRED>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT taken_by (student*)>
 <!ELEMENT student (name, grade)>
  <!ATTLIST student sno CDATA #REQUIRED>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT grade (#PCDATA)>
]>
```

verbesserter Entwurf:

```
<!DOCTYPE courses [
 <!ELEMENT courses (course*, info*)>
 <!ELEMENT course (title, taken_by)>
  <!ATTLIST course cno CDATA #REQUIRED>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT taken_by (student*)>
 <!ELEMENT student (grade)>
  <!ATTLIST student sno CDATA #REQUIRED>
 <!ELEMENT grade (#PCDATA)>
 <!ELEMENT info (number*, name)>
 <!ELEMENT number EMPTY>
  <!ATTLIST number sno CDATA #REQUIRED>
 <!ELEMENT name (#PCDATA)>
]>
```

kritischer funktionaler Zusammenhang:

> Haben zwei `student`-Elemente dieselben `sno`-Werte, dann haben sie auch dieselben `name`-Werte.

Beispiel 2:

schlechter Entwurf:

```
<!DOCTYPE db [
 <!ELEMENT db (conf*)>
 <!ELEMENT conf (title, issue+)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT issue (inproceedings+)>
 <!ELEMENT inproceedings (author+, title)>
  <!ATTLIST inproceedings
            key ID #REQUIRED
            pages CDATA #REQUIRED
            year CDATA #REQUIRED>
 <!ELEMENT author (#PCDATA)>
]>
```

verbesserter Entwurf:

```
<!DOCTYPE db [
 <!ELEMENT db (conf*)>
 <!ELEMENT conf (title, issue+)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT issue (inproceedings+)>
  <!ATTLIST issue
            year CDATA #REQUIRED>
 <!ELEMENT inproceedings (author+, title)>
  <!ATTLIST inproceedings
            key ID #REQUIRED
            pages CDATA #REQUIRED>
 <!ELEMENT author (#PCDATA)>
]>
```

kritischer funktionaler Zusammenhang:

> Je zwei `inproceedings`-Elemente innerhalb desselben `issue`-Elementes haben auch dieselben `year`-Werte.

Dieser Zusammenhang ist *relativ*!

Formalisierung

Seien die folgenden zueinander disjunkten Mengen gegeben:

- El: Element-Namen,
- Att: Attribut-Namen; beginnen alle mit @,
- Str: String-Werte,
- $Vert$: Knoten-Identifikatoren.

Seien `S` (Typ `String`) und $\perp$ (Nullwert) reservierte Symbole nicht in diesen Mengen.

Eine DTD ist gegeben zu $D = (E, A, P, R, r)$, wobei

- $E \subseteq El$ eine endliche Menge von *Element-Typen*,
- $A \subseteq Att$ eine endliche Menge von *Attributen*,
- P ist eine Abbildung von E in die Menge der *Element-Typ-Definitionen*.
  Sei $\tau \in E$. $P(\tau)$ = S, oder
  $P(\tau)$ ist ein regulärer Ausdruck α wie folgt:

$$\alpha ::= \epsilon \mid \tau' \mid \alpha \mid \alpha \mid \alpha, \alpha \mid \alpha*$$

- R ist eine Abbildung von E in die Potenzmenge von A. Gilt @$l \in R(\tau)$, dann ist @l *definiert* für τ.
- $r \in E$ ist der Element-Typ der *Wurzel*; $\forall \tau \in E : r \notin P(\tau)$.

Ein Ausdruck $w = w_1 \ldots w_n$ ist ein *Pfad* in D, wenn

- $w_1 = r$,
- w_i ist Element des Alphabets von $P(w_{i-1})$, $2 \leq i \leq n-1$, und
- w_n ist Element des Alphabets von $P(w_{n-1})$, oder $w_n = @l$ für $@l \in R(w_{n-1})$.

$length(w) = n$ und $last(w) = w_n$.
$paths(D)$ ist die Menge aller Pfade in D und
$EPaths(D)$ die Menge aller Pfade p mit $last(p) \in E$.

alle Pfade zu der DTD aus Beispiel 1:

```
courses
courses.course
courses.course.@cno
courses.course.title
courses.course.title.S
courses.course.taken_by
courses.course.taken_by.student
courses.course.taken_by.student.@sno
courses.course.taken_by.student.name
courses.course.taken_by.student.name.S
courses.course.taken_by.student.grade
courses.course.taken_by.student.grade.S
```

kein Pfad jedoch ist `courses.course.taken_by.student.student.name`

Warum Pfade?

- Pfade entsprechen Attributen im Relationenmodell.
- Im Relationenmodell ist ein Tupel eine Abbildung von einer Menge von Attributen in einen entsprechenden Wertebereich.
- Wir definieren entsprechend ein Tupel als eine Abbildung von einer Menge von Pfaden in einen entsprechenden Wertebereich.
- Wir werden solche Tupel *Baum-Tupel* t nennen.

ein Baum-Tupel t; vergl. DTD aus Beispiel 1:

```
t(courses) = v0
t(courses.course) = v1
t(courses.course.@cno) = csc200
t(courses.course.title) = v2
t(courses.course.title.S) = Automata Theory
t(courses.course.taken_by) = v3
t(courses.course.taken_by.student) = v4
t(courses.course.taken_by.student.@sno) = st1
t(courses.course.taken_by.student.name) = v5
t(courses.course.taken_by.student.name.S) = Deere
t(courses.course.taken_by.student.grade) = v6
t(courses.course.taken_by.student.grade.S) = A+
```

Ein XML-Baum ist gegeben zu $T = (V, lab, ele, att, root)$, wobei

- $V \subseteq Vert$ eine endliche Menge von Knoten,
- $lab : V \to El$,
- $ele : V \to Str \cup V^*$,
- att ist eine partielle Funktion $V \times Att \to Str$,
- $root \in V$ ist die Wurzel von T,
- die auf V definierte parent-child-Relation $\{(v_1, v_2) \mid v_2 \in ele(v_1)\}$ definiert einen Baum mit Wurzel $root$.

Wir nehmen $T \models D$ (T conforms to D, gültig bzgl. D) an, wobei D eine zugrundegelegte DTD.

Sei eine DTD $D = (E, A, P, R, r)$ gegeben. Ein *Baumtupel t in D* ist eine Funktion von $paths(D)$ nach $Vert \cup Str \cup \{\perp\}$, so dass:

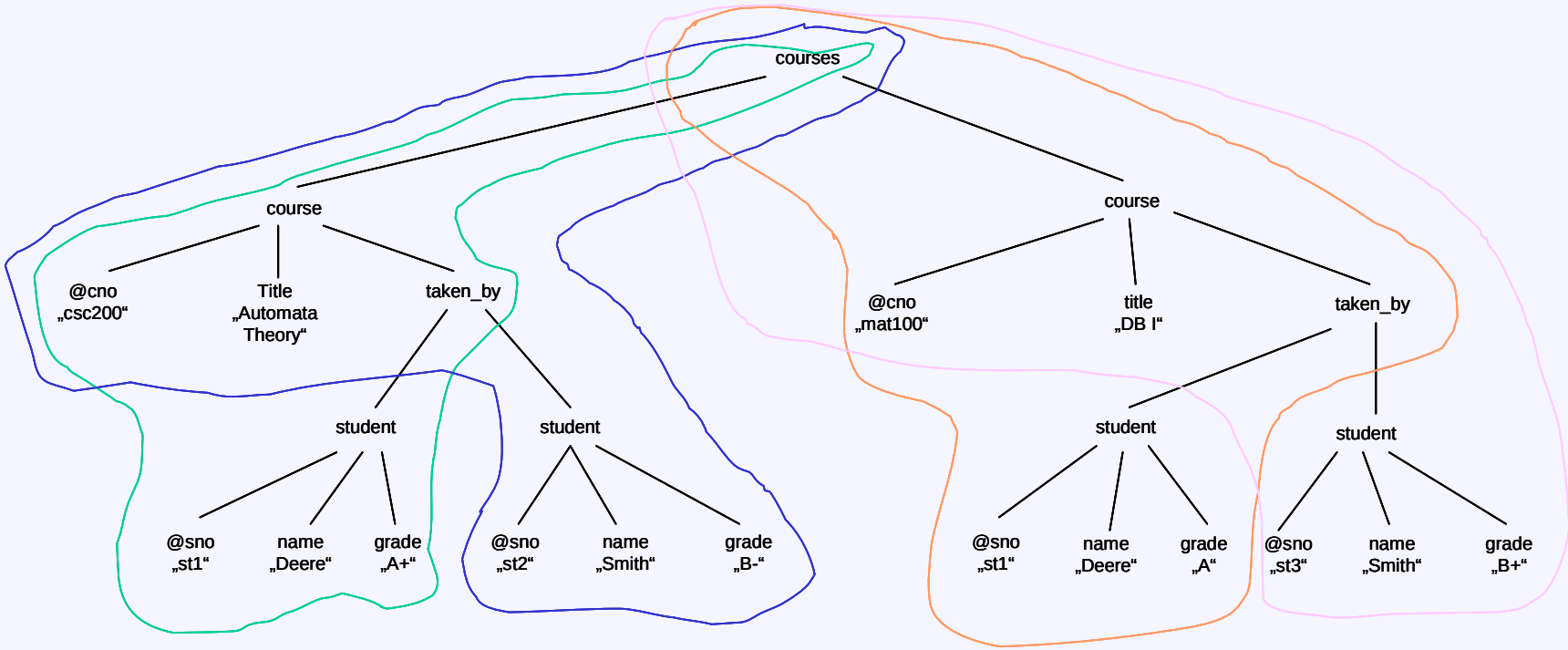
- $t(r) \neq \perp$,
- Falls $p \in EPaths(D)$, $t(p) \in Vert \cup \{\perp\}$,
- Falls $p \in paths(D) - EPaths(D)$, $t(p) \in Str \cup \{\perp\}$,
- Falls $t(p_1) = t(p_2)$ und $t(p_1) \in Vert$, dann $p_1 = p_2$.
- Falls $t(p_1) = \perp$ und p_1 ein Präfix von p_2, dann auch $t(p_2) = \perp$.
- $\{p \in paths(D) | t(p) \neq \perp\}$ ist endlich.

Die Menge aller Baumtupel zu D ist $\mathcal{T}(D)$.

Beispiel eines Baumtupels:

```
t(courses) = v0
t(courses.course) = v1
t(courses.course.@cno) = csc200
t(courses.course.title) = v2
t(courses.course.title.S) = Automata Theory
t(courses.course.taken_by) = v3
t(courses.course.taken_by.student) = v4
t(courses.course.taken_by.student.@sno) = st1
t(courses.course.taken_by.student.name) = v5
...
```

- Jedem Baumtupel t wird ein XML-Baum $tree_D(t)$ zugeordnet.
- Jedem XML-Baum T wird eine Menge von Baumtupeln $tuples_D(T)$ zugeordnet.
- Wir setzen im folgenden bei Bedarf Baumtupel t und ihre Baumdarstellung $tree_D(t)$ gleich.



Funktionale Abhängigkeiten (FA)

- Eine FA über einer DTD D ist ein Ausdruck $S_1 \rightarrow S_2$, wobei $S_1, S_2 \subseteq paths(D)$ nicht-leere, endliche Mengen.
- Sei $S \subseteq paths(D)$ und $t, t' \in \mathcal{T}(D)$.
  $t.S = t'.S$, wenn $t.p = t'.p$ für alle $p \in S$.
- $t.S \neq \perp$ heißt $t.p \neq \perp$ für alle p in S.
- Ein XML-Baum T *erfüllt* eine FA $S_1 \rightarrow S_2$, ($T \models S_1 \rightarrow S_2$), wenn für $t_1, t_2 \in tuples_D(T)$ gilt:

$$t_1.S_1 = t_2.S_1 \text{ und } t_1.S_1 \neq \perp \Longrightarrow t_1.S_2 = t_2.S_2.$$

zu Beispiel 1 exisiteren die FAs:

```
courses.course.@cno → courses.course

{courses.course, courses.course.taken_by.student.@sno} →

        courses.course.taken_by.student

courses.course.taken_by.student.@sno → courses.course.taken_by.student.name.S
```

XML Normalform XNF

Sei eine DTD D gegeben und Σ eine Menge FAs über D.
(D, Σ) ist in XNF, wenn für jede nicht-triviale FD $\psi \in \Sigma^+$ der Form

- $S \longrightarrow p.@l$, oder
- $S \longrightarrow p.\mathsf{S}$

auch $S \longrightarrow p \in \Sigma^+$.

zu Beispiel 1:

`courses.course.taken_by.student.@sno` $\longrightarrow$ `courses.course.taken_by.student.name.S`

verletzt die XNF.

zu Beispiel 2:

`db.conf.issue` $\longrightarrow$ `db.conf.issue.inproceedings.@year`

verletzt die XNF.

Nachtrag

Sei D eine DTD. Die Menge aller FDs über D ist $\mathcal{FD}(D)$.

- Sei $\Sigma \subseteq \mathcal{FD}(D)$ und $\phi \in \mathcal{FD}(D)$.
  (D, Σ) impliziert ϕ, $(D, \Sigma) \vdash \phi$, wenn für jeden XML-Baum T mit $T \models D$ und $T \models \Sigma$ auch $T \models \phi$ gilt.
  Die Menge aller durch (D, Σ) implizierten FDs ist $(D, \Sigma)^+$.
- Eine FD ϕ heißt *trivial*, wenn $(D, \emptyset) \vdash \phi$. Beispiele:
  - Sei $p \in EPaths(D)$ und p' Präfix von p. $p \rightarrow p'$ ist trivial.
  - Seien $p, p.@l \in paths(D)$. $p \rightarrow p.@l$ ist trivial.
  - Sei r Wurzel einer DTD und $P(r) = (a|b)$. Dann ist für jeden Pfad $p \in paths(D)$ die FD $\{r.a, r.b\} \rightarrow p$ trivial. Warum?