

Applied Databases

Lecture 20

Recap I

Sebastian Maneth

University of Edinburgh - March 24th, 2016

Recap I & II

1. XML, DTDs, XPath, deterministic regex's
2. Schemas, Normal Forms, SQL
3. TFIDF-ranking, string matching (KMP, automata, Boyer-Moore)

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

a)

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

-
- a) **not well-formed.** After “<” must follow a letter, and not ‘>’.
This is specified in the XML grammar → **context-free**

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

-
- a) **not well-formed.** After “<” must follow a letter, and not ‘>’.
This is specified in the XML grammar → **context-free**
 - b) **not well-formed.** Symbol “<” cannot appear inside a tag-name. → **context-free**

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

-
- a) **not well-formed.** After “<” must follow a letter, and not ‘>’.
This is specified in the XML grammar → **context-free**
 - b) **not well-formed.** Symbol “<” cannot appear inside a tag-name. → **context-free**
 - c) **not well-formed.** Two violations:
 (1) no end-tag for first -tag → **context-free (!)**
 (2) at="4" not allowed in an end tag → **context-free**

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

-
- a) **not well-formed.** After “<” must follow a letter, and not ‘>’.
This is specified in the XML grammar → **context-free**
 - b) **not well-formed.** Symbol “<” cannot appear inside a tag-name. → **context-free**
 - c) **not well-formed.** Two violations:
(1) no end-tag for first -tag → **context-free (!)**
(2) at="4" not allowed in an end tag → **context-free**
 - d) **well-formed.**

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML. In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

-
- a) **not well-formed.** After “<” must follow a letter, and not ‘>’.
This is specified in the XML grammar → **context-free**
 - b) **not well-formed.** Symbol “<” cannot appear inside a tag-name. → **context-free**
 - c) **not well-formed.** Two violations:
(1) no end-tag for first -tag → **context-free (!)**
(2) at="4" not allowed in an end tag → **context-free**
 - d) **well-formed.**
 - e) **not well-formed.** Missing end tag for first <a>-tag → **context-free**

1. XML

(1) well-formedness

For each of the following, explain whether or not it is well-formed XML.

In case it is not well-formed, **list all violations** that you find.

Say for each violation whether it is **context-free** or **context-dependent**.

- a) `<comment>For numbers x with $x < 5$, $x/5$ is not 1.</comment>`
- b) `<auto<node>>XF23414</auto<node>>`
- c) `<b at="7"/><b at="7"></b at="4">`
- d) ``
- e) `<a><a/><c></c>`
- f) `<a b3="a" b2="b" b1="a" b2="5"/>`

- a) **not well-formed.** After “<” must follow a letter, and not ‘>’.
This is specified in the XML grammar → **context-free**
- b) **not well-formed.** Symbol “<” cannot appear inside a tag-name. → **context-free**
- c) **not well-formed.** Two violations:
(1) no end-tag for first -tag → **context-free (!)**
(2) at="4" not allowed in an end tag → **context-free**
- d) **well-formed.**
- e) **not well-formed.** Missing end tag for first <a>-tag → **context-free**
- f) **not well-formed.** Duplicate attribute (b2) → **context-dependent**

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

- a) <bib><book></book></bib>
- b) <bib><journal isbn="xyz"><author/><title/></journal></bib>
- c) <bib><book isbn="123"><author/><title/></book><journal><author/>
<title/><cites><book isbn="123"><author/><title/><book/></cites></journal></bib>
- d) <bib book="isbn"></bib>
- e) <bib>no entries</bib>
- f) <bib></bib><bib></bib>
- g) <bib><author></author><title></title></Bib>

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

a) <bib><book></book></bib>

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

a) <bib><book></book></bib>

→ not well-formed!

(1) book must have author and title children

(2) book must have isbn attribute

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

b) `<bib><journal isbn="xyz"><author/><title/></journal></bib>`

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

b) <bib><journal isbn="xyz"><author/><title/></journal></bib>

→ not well-formed!

attribute isbn not declared for journal element

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

C) <bib><book isbn="123"><author/><title/></book><journal><author/>
<title/><cites><book isbn="123"><author/><title/><book/></cites></journal></bib>

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

C) `<bib><book isbn="123"><author/><title/></book><journal><author/>
<title/><cites><book isbn="123"><author/><title/><book/></cites></journal></bib>`
→ **not well-formed!**
Two violations: (1) end-tag of 2nd book-tag does not match (context-sensitive)
(2) ?

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

C) <bib><book isbn="123"><author/><title/></book><journal><author/>
<title/><cites><book isbn="123"><author/><title/><book/></cites></journal></bib>
→ **not well-formed!**
Two violations: (1) end-tag of 2nd book-tag does not match (context-sensitive)
(2) isbn-attribute of type ID has repeating values

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

d) `<bib book="isbn"></bib>`

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

d) `<bib book="isbn"></bib>`

→ **not well-formed!**

attribute book **not declared for** bib **element**

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

e) <bib>no entries</bib>

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

e) <bib>no entries</bib>

→ not well-formed!

bib-content must be (book | journal)*, so cannot be #PCDATA

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

f) <bib></bib><bib></bib>

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

f) <bib></bib><bib></bib>

→ not well-formed!

no root node (must end after first </bib>) context-free

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

g) `<bib><author></author><title></title></Bib>`

(2) DTDs

Which of the following are well-formed wrt the given DTD.
List all violations that you find.

```
<!DOCTYPE bib [  
  <!ELEMENT bib (book | journal)*>  
  <!ELEMENT book (author, title)>  
  <!ELEMENT journal (author, title, cites?)>  
  <!ELEMENT cites (book | journal)*>  
  <!ELEMENT author (#PCDATA)>  
  <!ELEMENT title (#PCDATA)>  
  <!ATTLIST book isbn ID #REQUIRED>  

```

g) <bib><author></author><title></title></Bib>

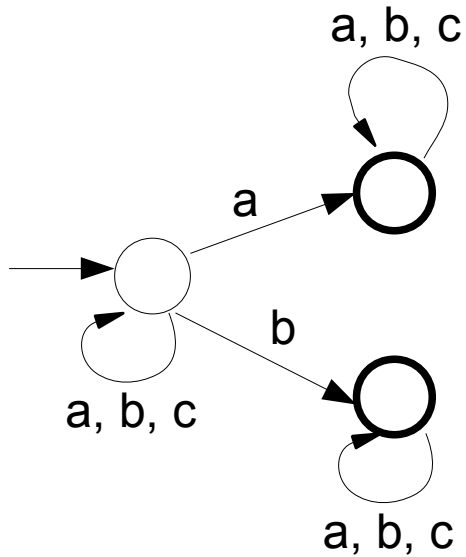
→ not well-formed!

Two violations:

(1) Bib does not match start bib-tag context-dependent

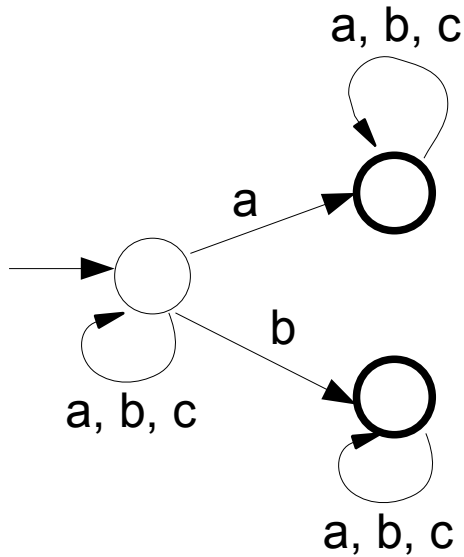
(2) bib may not have author or title children

(2) Deterministic Regular Expressions



- show a string accepted by the automaton, and one that is rejected.
Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic?
Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

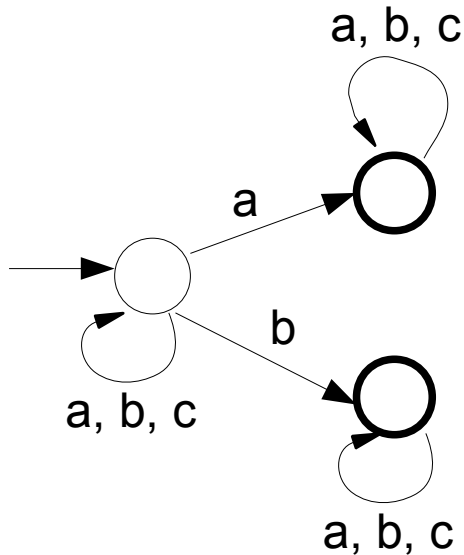
(2) Deterministic Regular Expressions



- show a string accepted by the automaton, and one that is rejected.
Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic?
Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

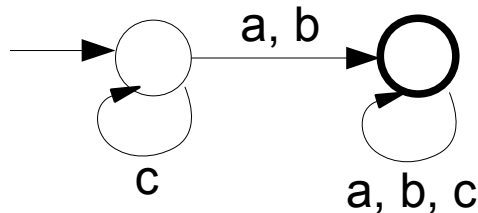
a) It accepts "a" and it rejects "c".

(2) Deterministic Regular Expressions



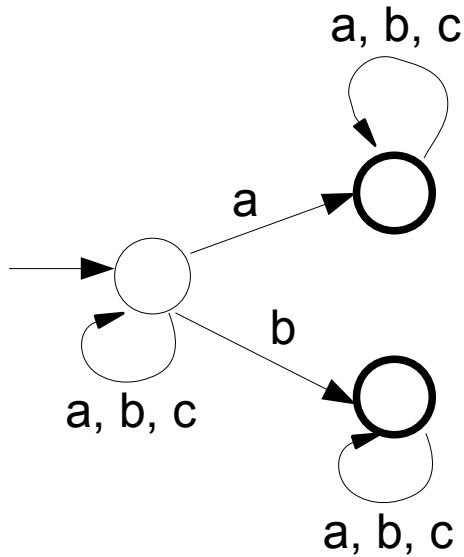
- show a string accepted by the automaton, and one that is rejected. Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic? Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

a) It accepts "a" and it rejects "c".



"strings that contain an 'a' or a 'b'"

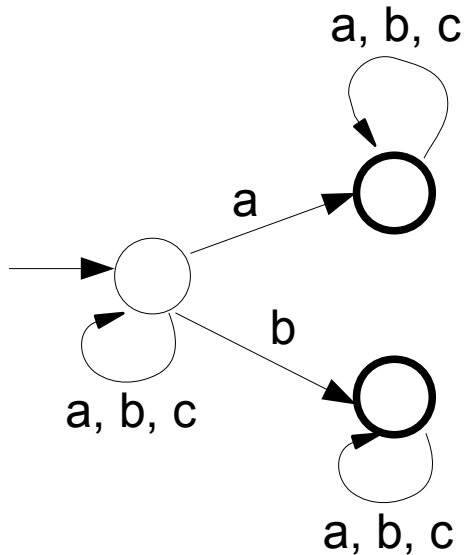
(2) Deterministic Regular Expressions



- show a string accepted by the automaton, and one that is rejected.
Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic?
Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

b) $c^*(a|b)(a|b|c)^*$

(2) Deterministic Regular Expressions

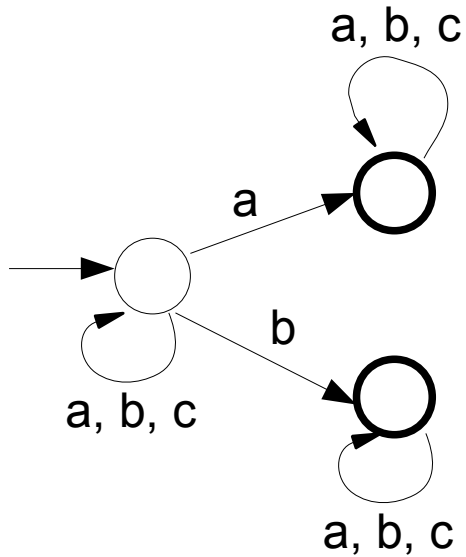


- show a string accepted by the automaton, and one that is rejected.
Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic?
Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

c) $c^*(a|b)(a|b|c)^*$

< present on blackboard >

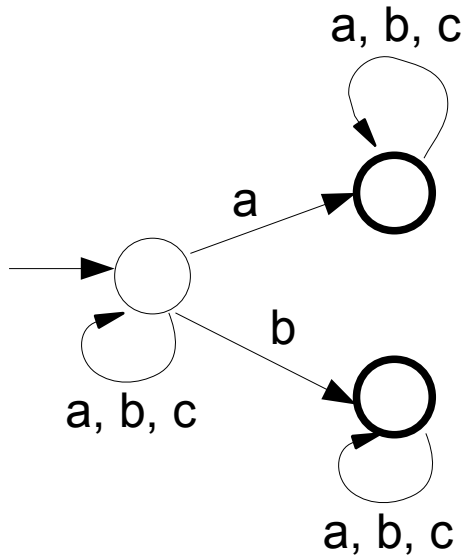
(2) Deterministic Regular Expressions



- show a string accepted by the automaton, and one that is rejected.
Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic?
Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

d)

(2) Deterministic Regular Expressions



- show a string accepted by the automaton, and one that is rejected.
Give an equivalent deterministic automaton.
- Give a regular expression for the strings accepted by the automaton.
- Is your expression from b) deterministic?
Show the Glushkov automaton.
- give a deterministic regular expression for the strings over $\{a,b,c\}$ that do not contain the substring "aa" and that end on "a".

d) $(b|c)^*a((b|c)+a)^*$

(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

a) `//a`

b) `/*/*/*//a[preceding::a]`

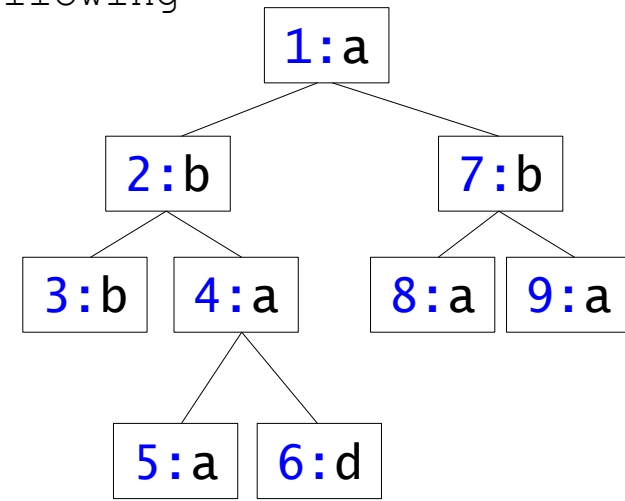
c) `//*[.//d]`

d) `/*[not(a and b)]`

e) `//*[count(.//*)= count(ancestor::*)]`

f) `/descendant:*[position() mod 2 = count(.//*)]`

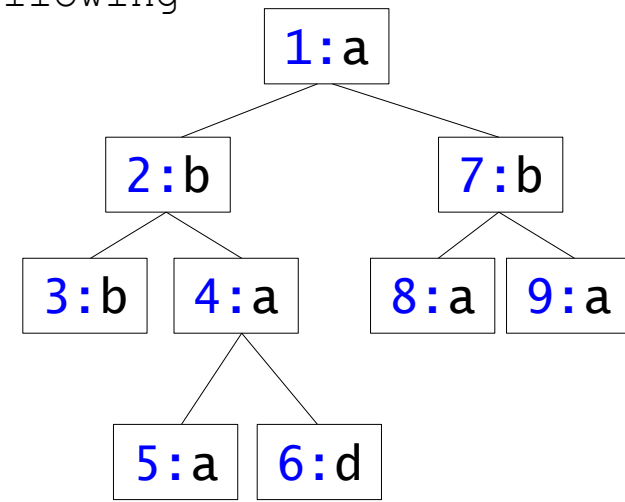
g) `//*[preceding-sibling::b]`



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

a) `//a`

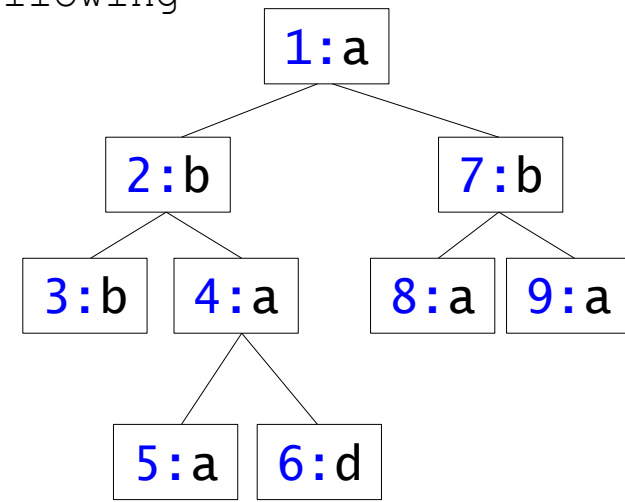


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

a) `//a`

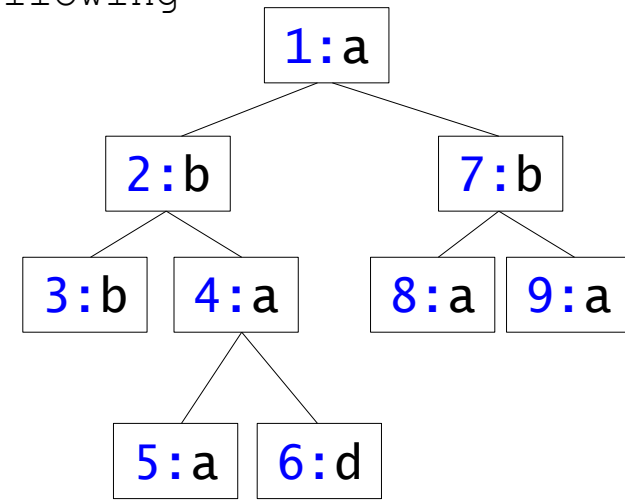
Answer: 1,4,5,8,9



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

b) `/*/*/*//a[preceding::a]`

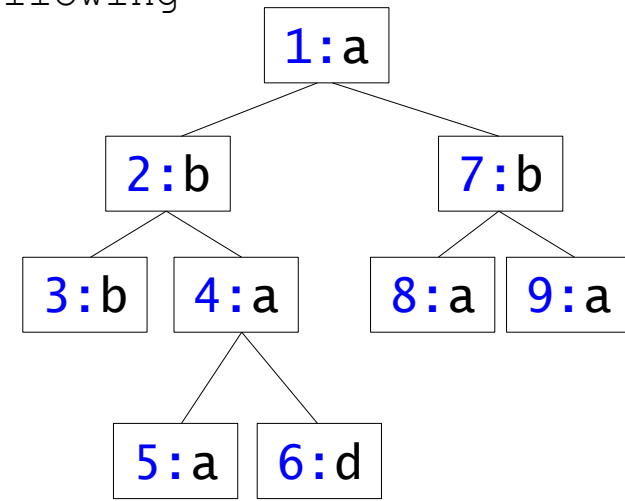


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

b) `/*/*/*//a[preceding::a]`

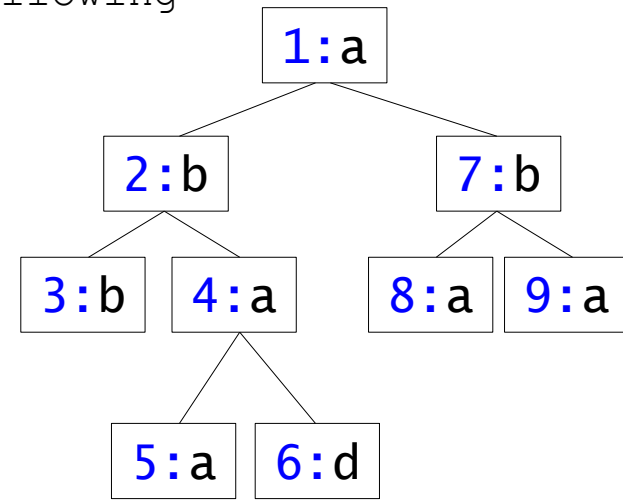
Answer: 8,9



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

c) `//*[.//d]`

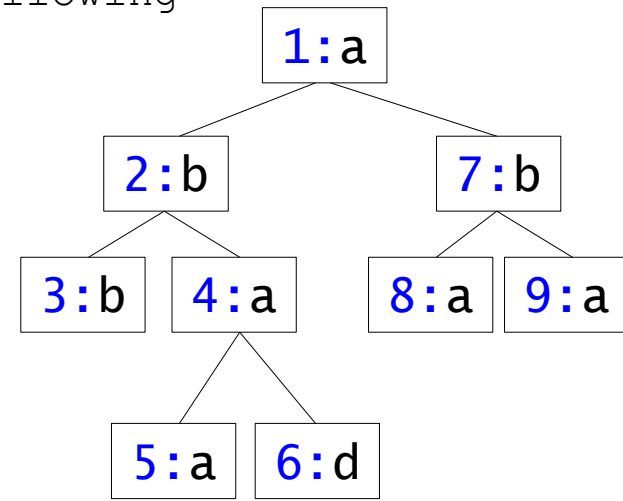


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

c) `//*[.//d]`

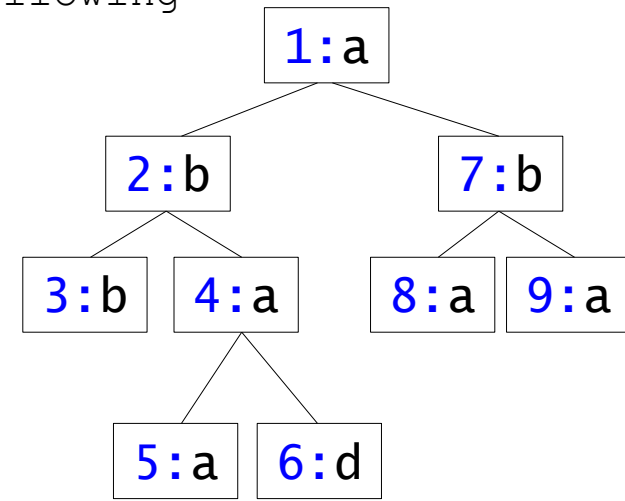
Answer: 1,2,4



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

d) `/*[not(a and b)]`

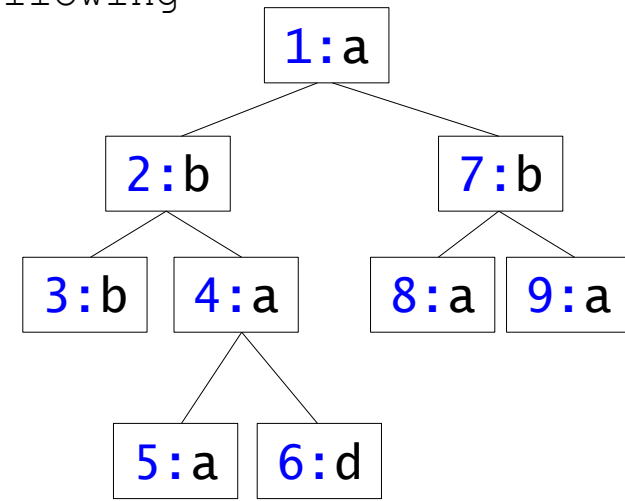


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

d) `/*[not(a and b)]`

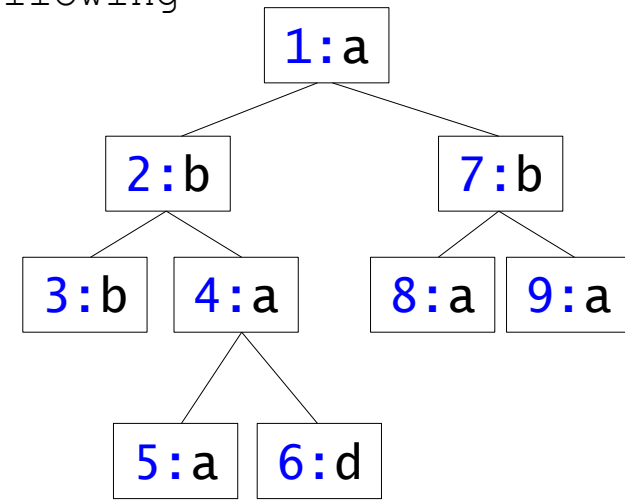
Answer: 1



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

e) `//*[count(.//*)= count(ancestor::*)]`

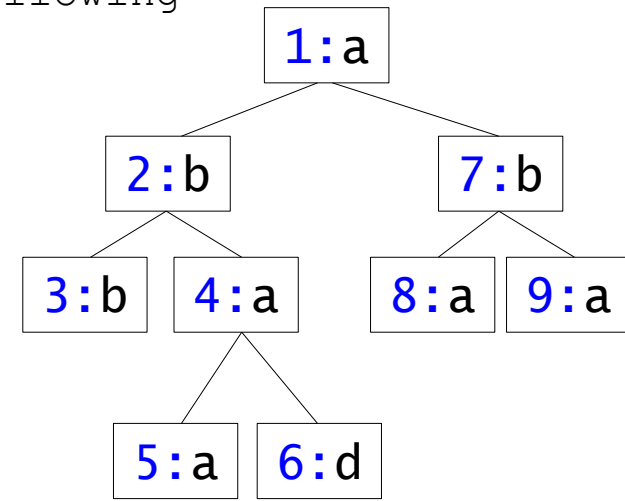


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

e) `//*[count(.//*)= count(ancestor::*)]`

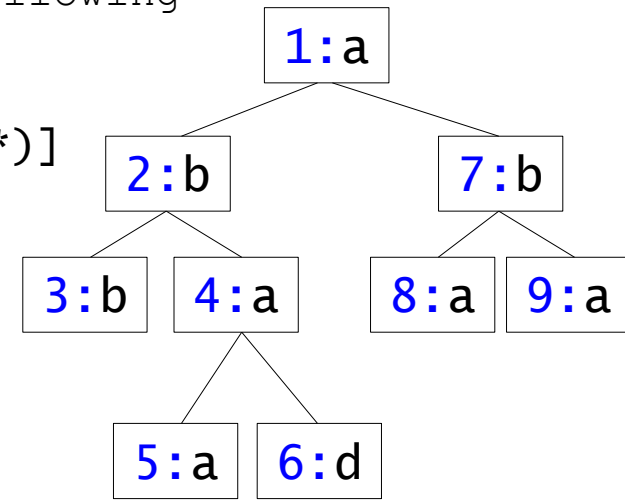
Answer: 4



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

f) `/descendant:*[position() mod 2 = count(../*)]`

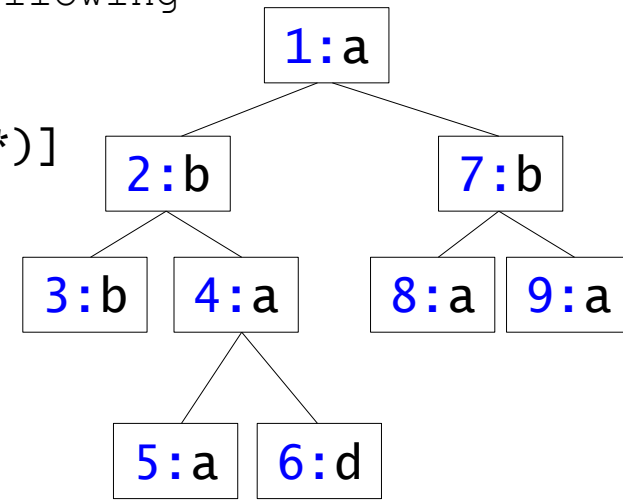


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

f) `/descendant:*[position() mod 2 = count(../*)]`

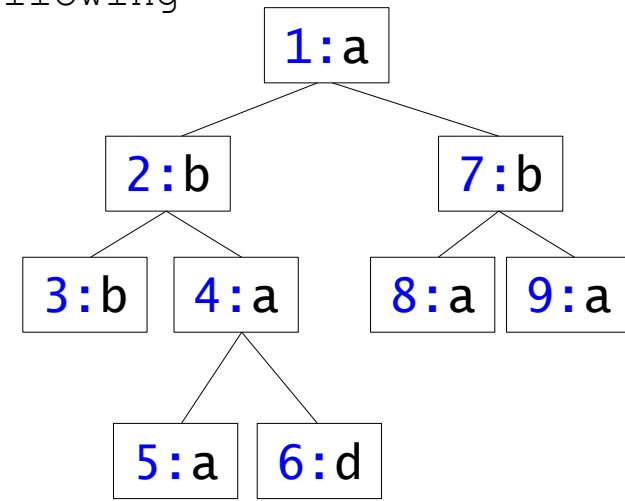
Answer: 6,8



(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

g) `//*[preceding-sibling::b]`

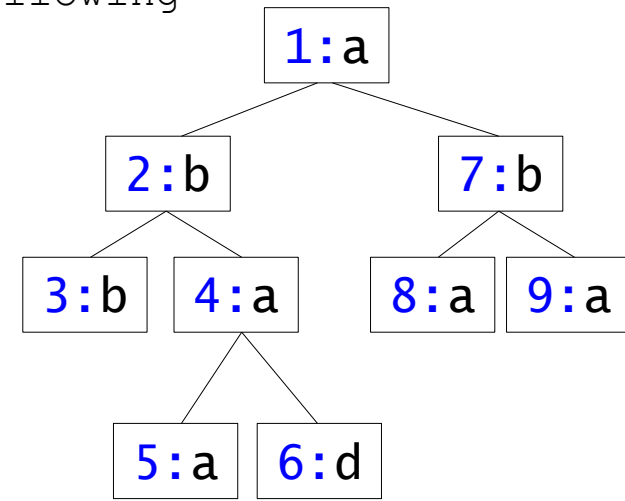


(3) XPath

Write `node-numbers` of nodes selected by the following XPath expressions:

g) `//*[preceding-sibling::b]`

Answer: 4, 7



2. Relational DBs

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
- 2) explain **BCNF** and how it removes **fd-redundancies**.
- 3) are there any “harmful” side-effects when transforming a table to **BCNF**?

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2

Functional dependencies? (“closed world assumption”)

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2

Functional dependencies? (“closed world assumption”)

- 1) **$X \rightarrow A$**
- 2) **$A \rightarrow X$**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2

Functional dependencies? (“closed world assumption”)

- 1) **X** → **A**
- 2) **A** → **X**

→ what are the **superkeys**?

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2

Functional dependencies? (“closed world assumption”)

- 1) **X** → **A**
- 2) **A** → **X**

→ what are the **superkeys**?

- 1) **X**
- 2) **A**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2

Functional dependencies? (“closed world assumption”)

- 1) **X** → **A**
- 2) **A** → **X**

→ what are the **superkeys**?

S is **superkey** if $S \rightarrow T$ and
 $S \cup T = \text{all attributes}$.

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2
2	2

Functional dependencies? (“closed world assumption”)
→ **and now?**

- 1) **X → A**
- 2) **A → X**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).

A table R has a **functional dependency from S to T**,
If R's projection to S union T gives a function from S to T.

Such a function implies that for every S-tuple, there is at most one T-tuple in R.

X	A
1	2
2	2

Functional dependencies? (“closed world assumption”)
→ **and now?**

- 1) **X → A**
- 2) ~~**A → X**~~

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

X → **A**

A → **X**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

X → **A**
A → **X**
Z → **X**
Z → **A**
Z → { **X**, **A** }

→ any more?

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

X → **A**
A → **X**
Z → **X**
Z → **A**
Z → { **X**, **A** }
{ **Z**, **A** } → **X**
{ **Z**, **X** } → **A**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

$X \rightarrow A$
 $A \rightarrow X$
 $Z \rightarrow X$
 $Z \rightarrow A$
 $Z \rightarrow \{X, A\}$
 $\{Z, A\} \rightarrow X$
 $\{Z, X\} \rightarrow A$

→ how many **superkeys**?

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

X	A	Z
1	2	5
1	2	6
1	2	7

← how many functional dependencies?

$X \rightarrow A$
 $A \rightarrow X$
 $Z \rightarrow X$
 $Z \rightarrow A$
 $Z \rightarrow \{X, A\}$
 $\{Z, A\} \rightarrow X$
 $\{Z, X\} \rightarrow A$

→ how many **superkeys**?

three

$Z, \{Z, A\}, \{Z, X\}$

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).

Functional dependency from S to T:

for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,

if R contains **two distinct tuples** with equal (S,T)-values.

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6

← are there any **fd-redundancies**?

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6

← are there any fd-redundancies?

- Yes: 1) **fd-redundancy wrt $X \rightarrow A$**
2) **fd-redundancy wrt $A \rightarrow X$**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.
-

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6
2	2	6

← list **all fd-redundancies!**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6
2	2	6

← list **all fd-redundancies!**

- 1) **fd-redundancy wrt $X \rightarrow A$**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6
2	2	6

← list **all fd-redundancies!**

1) **fd-redundancy wrt $X \rightarrow A$**

→ **A to X** is not a functional dependency anymore!

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6
2	2	6

← list **all fd-redundancies!**

- 1) **fd-redundancy wrt $X \rightarrow A$**
→ **A to X** is not a functional dependency anymore!
- 2) **fd-redundancy wrt $Z \rightarrow A$**

- 1) explain, using examples, what a **functional dependency (fd)** is, and what a **fd-redundancy** is.

Let S and T be **non-empty disjoint sets** of attributes (column names).
Functional dependency from S to T:
for every S-tuple, there is at most one T-tuple in R.

A table R has **fd-redundancy w.r.t. $S \rightarrow T$** ,
if R contains **two distinct tuples** with equal (S,T)-values.

X	A	Z
1	2	5
1	2	6
2	2	6

← list **all fd-redundancies!**

- 1) **fd-redundancy wrt $X \rightarrow A$**
→ **A to X** is not a functional dependency anymore!
- 2) **fd-redundancy wrt $Z \rightarrow A$**

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A
1	2
2	2

← in BCNF?

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A
1	2
2	2

← in BCNF?

Yes: X is superkey, and

$X \rightarrow A$ is the only functional dependency.

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

← in BCNF?

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6

← in BCNF?

No: $X \rightarrow A$ is fd, but X is not a superkey

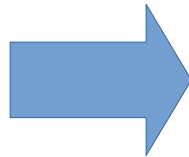
$A \rightarrow X$ is fd, but A is not a superkey

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6



X	A
1	2
1	2

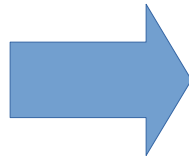
X	Z
1	5
1	6

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6



X	A
1	2
1	2

X	Z
1	5
1	6

In BCNF, there can be no fd-redundancies.

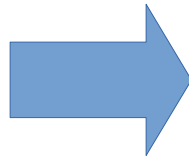
Why?

2) explain BCNF and how it removes fd-redundancies.

BCNF = if $S \rightarrow T$ is a functional dependency of R, then S is a superkey.

(assuming S disjoint T)

X	A	Z
1	2	5
1	2	6



X	A
1	2
1	2

X	Z
1	5
1	6

In BCNF, there can be no fd-redundancies.

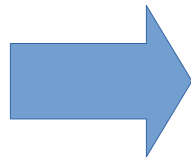
Why?

Would imply that a tuple exists **twice** in R



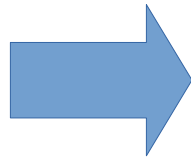
3) are there any “harmful” side-effects when transforming a table to **BCNF**?

X	A	Z
1	2	5
1	2	6
2	2	6



3) are there any “harmful” side-effects when transforming a table to BCNF?

X	A	Z
1	2	5
1	2	6
2	2	6

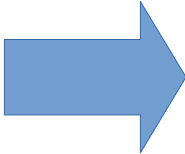


X	A
1	2
2	2

X	Z
1	5
1	6
2	6

3) are there any “harmful” side-effects when transforming a table to BCNF?

X	A	Z
1	2	5
1	2	6
2	2	6



X	A
1	2
2	2

X	Z
1	5
1	6
2	6

We lost the dependency $\{ \mathbf{X}, \mathbf{Z} \} \rightarrow \mathbf{A}$

END

Lecture 20