SCHEMA NORMALIZATION

CS 564- Fall 2015
HOW TO BUILD A DB APPLICATION

• Pick an application
• Figure out what to model (ER model)
  – Output: ER diagram
• Transform the ER diagram to a relational schema

• Refine the relational schema (normalization)

• Now ready to implement the schema and load the data!
**Motivating Example**

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

- What is the primary key?
  - (SSN, PhoneNumber)

- What is the problem with this schema?
## Motivating Example

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

### Problems:
- **redundant storage**
- **update**: change the age of Paris?
- **insert**: what if a person has no phone number?
- **delete**: what if Arun deletes his phone number?
**Solution: Decomposition**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Age</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

**Table 1:**

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
</tr>
</tbody>
</table>

**Table 2:**

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>
GENERAL SOLUTION

• identify “bad” schemas
  – functional dependencies (FDs)
• decompose the tables (normalize) using the FDs specified
• decomposition should be used judiciously:
  – normal forms (BCNF, 3NF) guarantee against some forms of redundancy
  – does decomposition cause any problems?
    • Lossless join
    • Dependency preservation
FUNCTIONAL DEPENDENCIES
**FD: Definition**

- FDs are a form of constraint
- generalize the concept of keys

If two tuples agree on the attributes

\[ A_1, A_2, \ldots, A_n \]

then they must agree on the attributes

\[ B_1, B_2, \ldots, B_m \]

Formally:

\[ A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_m \]
### FD: Example 1

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>9347298377</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>9347298377</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>1231236455</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>3844756877</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

- $SSN \rightarrow name, age$
- $SSN, age \rightarrow name$
- $SSN \nrightarrow phoneNumber$
## FD: Example 2

<table>
<thead>
<tr>
<th>studentID</th>
<th>semester</th>
<th>courseNo</th>
<th>section</th>
<th>instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>124434</td>
<td>4</td>
<td>CS 564</td>
<td>1</td>
<td>Paris</td>
</tr>
<tr>
<td>546364</td>
<td>4</td>
<td>CS 564</td>
<td>2</td>
<td>Arun</td>
</tr>
<tr>
<td>999492</td>
<td>6</td>
<td>CS 764</td>
<td>1</td>
<td>Anhai</td>
</tr>
<tr>
<td>183349</td>
<td>6</td>
<td>CS 784</td>
<td>1</td>
<td>Jeff</td>
</tr>
</tbody>
</table>

- courseNo, section → instructor
- studentID → semester
**How Do We Infer FDs?**

- What FDs are valid for a relational schema?
  - think from an application point of view

- An FD is
  - an inherent property of an application
  - not something we can infer from a set of tuples

- Given a table with a set of tuples
  - we can confirm that a FD *seems* to be valid
  - to infer that a FD is *definitely* invalid
  - we can *never* prove that a FD is valid
### Example 3

<table>
<thead>
<tr>
<th>name</th>
<th>category</th>
<th>color</th>
<th>department</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Gadget</td>
<td>Green</td>
<td>Toys</td>
<td>49</td>
</tr>
<tr>
<td>Tweaker</td>
<td>Gadget</td>
<td>Black</td>
<td>Toys</td>
<td>99</td>
</tr>
<tr>
<td>Gizmo</td>
<td>Stationary</td>
<td>Green</td>
<td>Office-supplies</td>
<td>59</td>
</tr>
</tbody>
</table>

To confirm whether `name → department`
- erase all other columns
- check that the relationship `name-department` is many-one!
Why FDs?

- keys are special cases of FDs
- more integrity constraints for the application
- having FDs will help us detect that a table is “bad”, and how to decompose the table
More on FDs

• If the following FDs hold:
  – $A \rightarrow B$
  – $B \rightarrow C$

• then the following FD is also true:
  – $A \rightarrow C$

• This means that there are more FDs that can be found! How?
  – Armstrong’s Axioms
ARMSTRONG’S AXIOMS: 1

**Reflexivity**
For any subset $X \subseteq \{A_1, \ldots, A_n\}$:

$A_1, A_2, \ldots, A_n \rightarrow X$

- **Examples**
  - $A, B \rightarrow B$
  - $A, B, C \rightarrow A, B$
  - $A, B, C \rightarrow A, B, C$
**Armstrong’s Axioms: 2**

**Augmentation**
For any attribute sets $X, Y, Z$:
if $X \rightarrow Y$ then $X, Z \rightarrow Y, Z$

- **Examples**
  - $A \rightarrow B$ implies $A, C \rightarrow B, C$
  - $A, B \rightarrow C$ implies $A, B, C \rightarrow C$
ARMSTRONG’S AXIOMS: 3

Transitivity
For any attribute sets $X, Y, Z$:
if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

• Examples
  – $A \rightarrow B$ and $B \rightarrow C$ imply $A \rightarrow C$
  – $A \rightarrow C, D$ and $C, D \rightarrow E$ imply $A \rightarrow E$
**Applying Armstrong’s Axioms**

**Product** (name, category, color, department, price)
- \( name \rightarrow color \)
- \( category \rightarrow department \)
- \( color, category \rightarrow price \)

Inferred FDs:
- \( name, category \rightarrow price \)
  - (1) Augmentation, (2) Transitivity
- \( name, category \rightarrow color \)
  - (1) Reflexivity, (2) Transitivity
APPLYING ARMSTRONG’S AXIOMS

FD Closure
If $F$ is a set of FDs, the closure $F^+$ is the set of all FDs logically implied by $F$

Armstrong’s axioms are:

- **sound**: any FD generated by an axiom belongs in $F^+$
- **complete**: repeated application of the axioms will generate all FDs in $F^+$
Closure of Attribute Sets

Attribute Closure
If \( X \) is an attribute set, the closure \( X^+ \) is the set of all attributes \( B \) such that:
\[
X \rightarrow B
\]

In other words, \( X^+ \) includes all attributes that are functionally determined from \( X \)
**Example**

**Product** (name, category, color, department, price)

- $name \rightarrow color$
- $category \rightarrow department$
- $color, category \rightarrow price$

**Attribute Closure:**

- $\{name\}^+ = \{name, color\}$
- $\{name, category\}^+ = \{name, color, category, department, price\}$
Why is Closure Needed?

• Does \( X \rightarrow Y \) hold?
  – check if \( Y \subseteq X^+ \)

• Compute the closure \( F^+ \) of FDs
  – for each subset of attributes \( X \), compute \( X^+ \)
  – for each subset of attributes \( Y \subseteq X^+ \), output the FD \( X \rightarrow Y \)
Let $X = \{A_1, A_2, \ldots, A_n\}$

**Until** $X$ doesn’t change **repeat**:

if $B_1, B_2, \ldots, B_m \rightarrow C$ is a FD and $B_1, B_2, \ldots, B_m$ are all in $X$

then add $C$ to $X$
EXAMPLE

\[ R(A, B, C, D, E, F) \]

• \( A, B \rightarrow C \)
• \( A, D \rightarrow E \)
• \( B \rightarrow D \)
• \( A, F \rightarrow B \)

Compute:

• \( \{A, B\}^+ = \{A, B, C, D, E\} \)
• \( \{A, F\}^+ = \{A, F, B, D, E, C\} \)
Relation $\mathbf{R}$

- **superkey**: a set of attributes $A_1, A_2, \ldots, A_n$ such that for any other attribute $B$
  
  $A_1, A_2, \ldots, A_n \rightarrow B$

- **key**: a minimal superkey
  
  – none of its subsets functionally determines all attributes of $\mathbf{R}$
Computing Keys & Superkeys

- Compute $X^+$ for all sets of attributes $X$
- If $X^+ = \text{all attributes}$, then $X$ is a superkey
- If no subset of $X$ is a superkey, then $X$ is also a key
**Example**

**Product**\((name, category, price, color)\)
- \(name \rightarrow color\)
- \(color, category \rightarrow price\)

**Superkeys:**
- \{name, category\}, \{name, category, price\}
  \{name, category, color\}, \{name, category, price, color\}

**Keys:**
- \{name, category\}
**Many Keys?**

**Q:** Is it possible to have many keys in a relation $R$?

**YES!!** Take relation $R(A, B, C)$ with FDs

- $A, B \rightarrow C$
- $A, C \rightarrow B$
RECAP

• FDs and (super)keys
• Reasoning with FDs:
  – given a set of FDs, infer all implied FDs
  – given a set of attributes \( X \), infer all attributes that are functionally determined by \( X \)
• Next we will look at how to use them to detect that a table is “bad”
BCNF Decomposition
BOYCE-CODD NORMAL FORM (BCNF)

A relation \( R \) is in BCNF if whenever \( X \rightarrow B \) is a non-trivial FD, then \( X \) is a superkey in \( R \)

**Equivalent definition:** for every attribute set \( X \)
- either \( X^+ = X \)
- or \( X^+ = all \) attributes
BCNF EXAMPLE 1

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

**SSN → name, age**

- **key** = \{SSN, phoneNumber\}
- **SSN → name, age** is a “bad” dependency
- The above relation is not in BCNF!
**BCNF Example 2**

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
</tr>
</tbody>
</table>

$SSN \rightarrow name, age$

- **key** = \{SSN\}
- The above relation is in BCNF!
**BCNF Example 3**

<table>
<thead>
<tr>
<th>SSN</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

- **key** = \{SSN, phoneNumber\}
- The above relation is in BCNF!
- **Q**: can we have a binary relation that is not in BCNF?
BCNF Decomposition

• Find an FD that violates the BCNF condition
  \[ A_1, A_2, ..., A_n \rightarrow B_1, B_2, ..., B_m \]

• Decompose \( R \) to \( R_1 \) and \( R_2 \):

  \[ R_1 \quad \bigcap \quad R_2 \]

  \[ \begin{align*}
  R_1 & : \text{A’s} \\
  \text{B’s} & : \text{remaining attributes}
  \end{align*} \]

• Continue until no BCNF violations are left
The FD $SSN \rightarrow name, age$ violates BCNF.

Split into two relations $R_1, R_2$ as follows:

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

$R_1$: name, age

$R_2$: SSN, phoneNumber
**Decomposition Example**

\[ SSN \rightarrow \text{name, age} \]

<table>
<thead>
<tr>
<th>SSN</th>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>Paris</td>
<td>24</td>
</tr>
<tr>
<td>123123645</td>
<td>John</td>
<td>30</td>
</tr>
<tr>
<td>384475687</td>
<td>Arun</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>934729837</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>934729837</td>
<td>603-534-8399</td>
</tr>
<tr>
<td>123123645</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>384475687</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>
BCNF EXAMPLE

**Person** (SSN, name, age, canDrink, phoneNumber)

- **SSN** → **name, age**
- **age** → **canDrink**
DECOMPOSITION PROPERTIES
**Decomposition in General**

Let $R(A_1, \ldots, A_n)$. To decompose, create:

- $R_1(B_1, \ldots, B_m)$
- $R_2(C_1, \ldots, C_l)$
- where \( \{B_1, \ldots, B_m\} \cup \{C_1, \ldots, C_l\} = \{A_1, \ldots, A_n\} \)

Then:

- $R_1$ is the projection of $R$ onto $B_1, \ldots, B_m$
- $R_2$ is the projection of $R$ onto $C_1, \ldots, C_l$
PROPERTIES

1. minimize redundancy
2. avoid information loss
3. preserve functional dependencies
4. ensure good query performance
**INFORMATION LOSS EXAMPLE**

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>John</td>
<td>24</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>Arun</td>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

Decompose into:
- $R_1$(name, age)
- $R_2$(age, phoneNumber)

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>24</td>
</tr>
<tr>
<td>John</td>
<td>24</td>
</tr>
<tr>
<td>Arun</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>age</th>
<th>phoneNumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>608-374-8422</td>
</tr>
<tr>
<td>24</td>
<td>608-321-1163</td>
</tr>
<tr>
<td>20</td>
<td>206-473-8221</td>
</tr>
</tbody>
</table>

Can we put it back together?
The decomposition is **lossless-join** if $R'$ is the same as $R$. 

**Diagram:**

- $R(A, B, C)$
- $R_1(A, B)$
- $R_2(B, C)$
- $R'(A, B, C)$

- Decompose $R$ into $R_1$ and $R_2$.
- Recover $R'$ from $R_1$ and $R_2$. 

**Equation:**

$$R(A, B, C) \rightarrow R_1(A, B) \rightarrow R'(A, B, C) \rightarrow R_2(B, C) \rightarrow R'(A, B, C)$$
FD PRESERVING

• Given a relation $R$ and a set of FDs $F$, decompose $R$ into $R_1$ and $R_2$

• Suppose
  – $R_1$ has a set of FDs $F_1$
  – $R_2$ has a set of FDs $F_2$
  – $F_1$ and $F_2$ are computed from $F$

The decomposition is dependency preserving if by enforcing $F_1$ over $R_1$ and $F_2$ over $R_2$, we can enforce $F$ over $R$
**GOOD EXAMPLE**

**Person**\((SSN, \text{name}, \text{age}, \text{canDrink})\)

- \(SSN \rightarrow \text{name}, \text{age}\)
- \(\text{age} \rightarrow \text{canDrink}\)

Decomposes into:

- \(R_1(\text{SSN, name, age})\)
  - \(SSN \rightarrow \text{name, age}\)
- \(R_2(\text{age, canDrink})\)
  - \(\text{age} \rightarrow \text{canDrink}\)
**Bad Example**

\( R(A, B, C) \)

- \( A \rightarrow B \)
- \( B, C \rightarrow A \)

Decomposes into:

- \( R_1(A, B) \)
  - \( A \rightarrow B \)
- \( R_2(A, C) \)
  - no FDs here!!

The recovered table violates \( B, C \rightarrow A \)
DECOMPOSITION

• When decomposing a relation $R$, we want to achieve good properties
• These properties can be conflicting
• BCNF decomposition achieves some of these:
  – removes certain types of redundancy
  – is lossless-join
  – is not always dependency preserving
Why is BCNF Lossless-Join?

Example:

\[ \mathbf{R}(A, B, C) \text{ with } A \rightarrow B \text{ decomposes into:} \]
\[ \mathbf{R}_1(A, B) \text{ and } \mathbf{R}_2(A, C) \]

- Suppose tuple \((a, b, c)\) is in the recovered \(\mathbf{R}'\)
- Then, \((a, b)\) in \(\mathbf{R}_1\) and \((a, c)\) in \(\mathbf{R}_2\)
- But then \((a, b', c)\) is in \(\mathbf{R}\)
- Since \(A \rightarrow B\) it must be that \(b' = b\)
- So \((a, b, c)\) is also in \(\mathbf{R}\)!
Why is BCNF Not FD Preserving?

\( R(A, B, C) \)
- \( A \rightarrow B \)
- \( B, C \rightarrow A \)

The BCNF decomposition is:
- \( R_1(A, B) \) with FD \( A \rightarrow B \)
- \( R_2(A, C) \) with no FDs

There may not exist any BCNF decomposition that is FD preserving!
NORMAL FORMS

BCNF is what we call a **normal form**

Other normal forms exist:

- **1NF**: flat tables (atomic values)
- **2NF**
- **3NF**
- **BCNF**
- **4NF**
- ...
THIRD NORMAL FORM (3NF)
3NF Definition

A relation $R$ is in **3NF** if whenever $X \rightarrow A$, one of the following is true:

- $A \in X$ (trivial FD)
- $X$ is a superkey
- $A$ is part of some key of $R$ (prime attribute)

BCNF implies 3NF
• **Example**: \( R(A, B, C) \) with \( A, B \rightarrow C \) and \( C \rightarrow A \)
  – is in 3NF. Why?
  – is not in BCNF. Why?

• Compromise used when BCNF not achievable: *aim for BCNF and settle for 3NF*

• Lossless-join and dependency-preserving decomposition of \( R \) into a collection of 3NF relations is always possible!
3NF Decomposition

- The algorithm for BCNF decomposition can be used to get a lossless-join decomposition into 3NF.
- We can typically stop earlier!
- To ensure dependency preservation as well, instead of the given set of FDs $F$, we use a minimal (or canonical) cover for $F$. 

CS 564 [Fall 2015] - Paris Koutris
MINIMAL COVER FOR FDs

• minimal cover $G$ for a set of FDs $F$:
  – $G^+ = F^+$
  – If we modify $G$ by deleting an FD or by deleting attributes from an FD in $G$, the closure changes
  – The LHS of each FD is unique

• The minimal cover gives a lossless-join and dependency-preserving decomposition algorithm!
Example:

Example:
- $A \rightarrow B$
- $A, B, C, D \rightarrow E$
- $E, F \rightarrow G, H$
- $A, C, D, F \rightarrow E, G$

The minimal cover is the following:
- $A \rightarrow B$
- $A, C, D \rightarrow E$
- $E, F \rightarrow G, H$
3NF Algorithm

1. Find a lossless-join 3NF decomposition (that might violate some FDs)
2. Compute a minimal cover $F$
3. Find the FDs in $F$ that are not preserved
4. For each non-preserved FD $X \rightarrow A$ add a new relation $R(X, A)$
**IS NORMALIZATION ALWAYS GOOD?**

- **Example:** suppose A and B are always used together, but normalization says they should be in different tables
  - decomposition might produce unacceptable performance loss
- **Example:** data warehouses
  - huge historical DBs, rarely updated after creation
  - joins expensive or impractical
- **Everyday DBs:** aim for BCNF, settle for 3NF!
RECAP

- Bad schemas lead to redundancy
  - redundant storage, update, insert, and delete anomaly
- To “correct” bad schemas: decompose relations
  - must be a lossless-join decomposition
  - would like dependency-preserving decompositions
- Desired normal forms
  - **BCNF**: only superkey FDs
  - **3NF**: superkey FDs + dependencies with prime attributes on the RHS