Relational Calculus

Dr. Constantinos Constantinides, P.Eng.

Department of Computer Science and
Software Engineering
Concordia University

Example 1: A declaration of a Power Set type

- Consider the set Name = {John, Myriam, Mike, Suzan}.
- Consider the power set of Name:

  P Name =

  {

  ∅,

  {John}, {Myriam}, {Mike}, {Suzan},

  {John, Myriam}, {John, Mike}, {John, Suzan},

  {Myriam, Mike}, {Myriam, Suzan}, {Mike, Suzan},

  {John, Myriam, Mike}, {John, Myriam, Suzan},

  {John, Mike, Suzan}, {Myriam, Mike, Suzan},

  {John, Myriam, Mike, Suzan}

  }
- The power set of Name is a set that contains, as its elements, all subsets of Name.

Type declarations

- How do we interpret a type declaration?
- The expression v : Type is interpreted as “The variable v can assume any value supported by Type.”
- Examples:
- In $x : \mathrm{N}$, variable x can assume values 1, 2, 3, …
- In z : Boolean, variable z can assume values true, or false.

Declaring a variable to hold a set

P Name = { ∅,
{John}, {Myriam}, {Mike}, {Suzan},
{John, Myriam}, {John, Mike}, {John, Suzan},
{Myriam, Mike}, {Myriam, Suzan}, {Mike, Suzan},
{John, Myriam, Mike}, {John, Myriam, Suzan},
{John, Mike, Suzan}, {Myriam, Mike, Suzan},
{John, Myriam, Mike, Suzan} }

- The declaration known : P Name is interpreted as “The variable known can assume any value supported by P Name.”

Since the values of P Name are sets, this implies that known is declared to be a set.

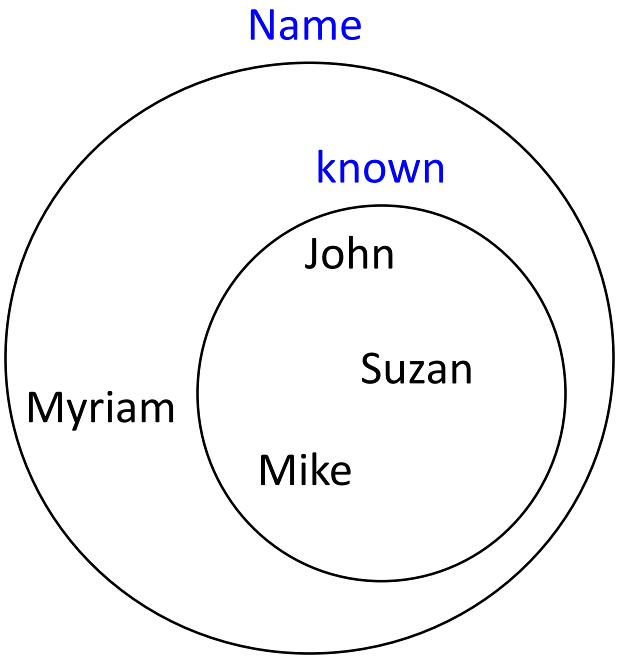
Reasoning about legitimate values for the variable

P Name = { ∅,
{John}, {Myriam}, {Mike}, {Suzan},
{John, Myriam}, {John, Mike}, {John, Suzan},
{Myriam, Mike}, {Myriam, Suzan}, {Mike, Suzan},
{John, Myriam, Mike}, {John, Myriam, Suzan},
{John, Mike, Suzan}, {Myriam, Mike, Suzan},
{John, Myriam, Mike, Suzan} }

- The declaration known : P Name is interpreted as “The variable known can assume any value supported by P Name.”

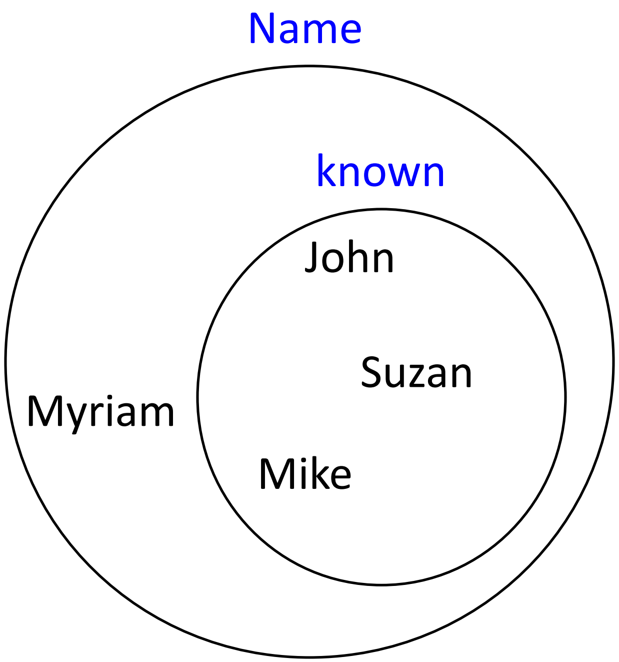
- Is {John, Mike, Suzan} a legitimate value for known? Yes.

Visualizing set relations



Is {John, Mike, Suzan} a legitimate value for known? Yes.

Variable declaration vs. set relation



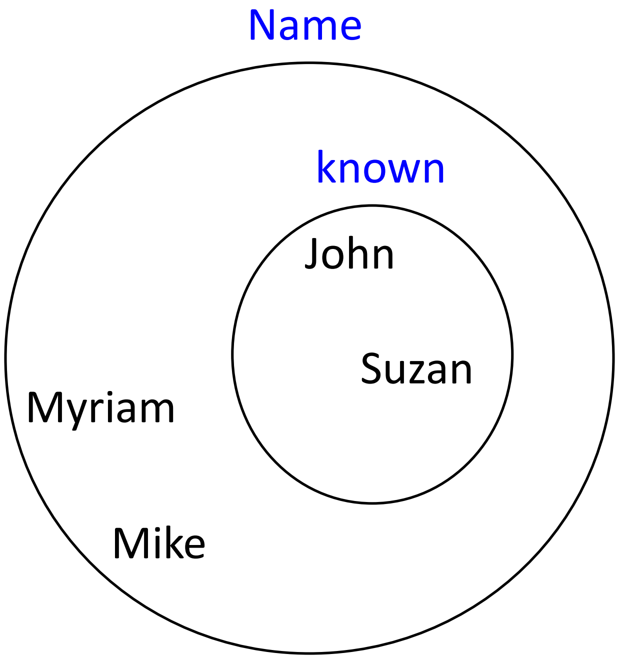
- Note that known : P Name, and known $\subseteq$ Name are both correct, but they mean a different thing.
- The expression known : P Name is a variable declaration.
- The expression known $\subseteq$ Name describes a set relation.

Reasoning about legitimate values for the variable /cont.

P Name = { $\varnothing$,
{John}, {Myriam}, {Mike}, {Suzan},
{John, Myriam}, {John, Mike}, {John, Suzan},
{Myriam, Mike}, {Myriam, Suzan}, {Mike, Suzan},
{John, Myriam, Mike}, {John, Myriam, Suzan},
{John, Mike, Suzan}, {Myriam, Mike, Suzan},
{John, Myriam, Mike, Suzan} }

- If we now removed Mike from known, we will have {John, Suzan}.
- Is {John, Suzan} a legitimate value for known? Yes.

Visualizing set relations /cont.



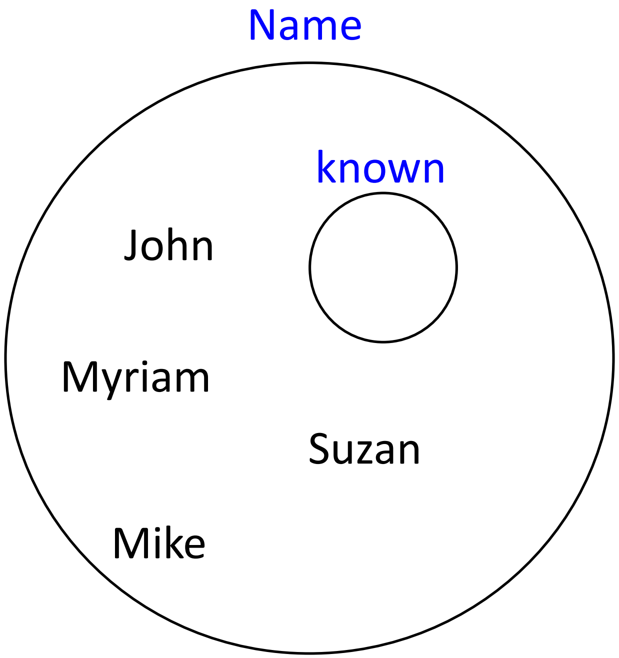
Is {John, Suzan} a legitimate value for known? Yes.

Reasoning about legitimate values for the variable /cont.

P Name = { ∅,
{John}, {Myriam}, {Mike}, {Suzan},
{John, Myriam}, {John, Mike}, {John, Suzan},
{Myriam, Mike}, {Myriam, Suzan}, {Mike, Suzan},
{John, Myriam, Mike}, {John, Myriam, Suzan},
{John, Mike, Suzan}, {Myriam, Mike, Suzan},
{John, Myriam, Mike, Suzan} }

- If we now removed all elements from known, we will have ∅.
- Is ∅ a legitimate value for known? Yes.

Visualizing set relations /cont.



Is $\varnothing$ a legitimate value for known? Yes.

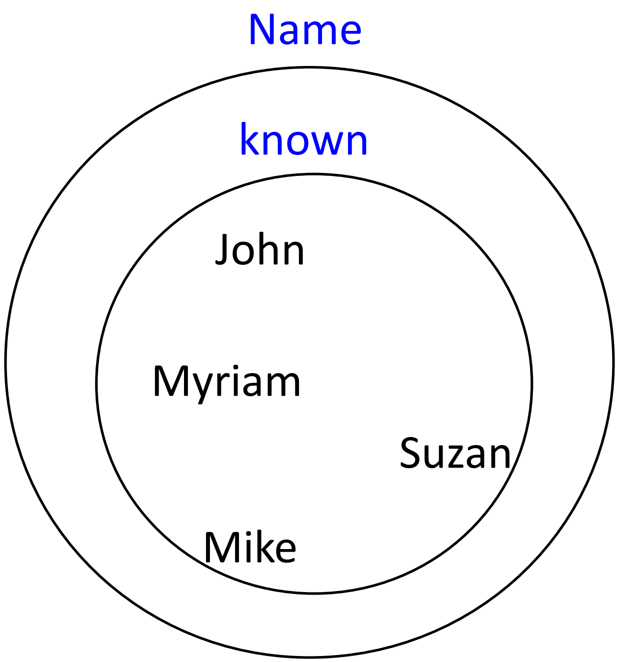
Reasoning about legitimate values for the variable /cont.

P Name = { $\varnothing$,
{John}, {Myriam}, {Mike}, {Suzan},
{John, Myriam}, {John, Mike}, {John, Suzan},
{Myriam, Mike}, {Myriam, Suzan}, {Mike, Suzan},
{John, Myriam, Mike}, {John, Myriam, Suzan},
{John, Mike, Suzan}, {Myriam, Mike, Suzan},
{John, Myriam, Mike, Suzan} }

- Let us now add all names into known.

- Is {John, Myriam, Mike, Suzan} a legitimate value for known? Yes.

Visualizing set relations /cont.



Is {John, Myriam, Mike, Suzan} a legitimate value for known? Yes.

Example 2: A database table

- Let binary relation `car : Model ↔ Make`
- The relation `car` can be used to model a database table.

car =

{
camry ↦ toyota,
avalon ↦ toyota,
highlander ↦ toyota,
wrangler ↦ jeep,
renegade ↦ jeep,
compass ↦ jeep,
altima ↦ nissan,
pathfinder ↦ nissan,
murano ↦ nissan
}

Domain and range

- Domain and range are sets of first and second elements respectively.
- `dom car` = {camry, avalon, highlander, wrangler, renegade, compass, altima, pathfinder, murano}
- `ran car` = {toyota, jeep, nissan}

car = {
camry ↦ toyota,
avalon ↦ toyota,
highlander ↦ toyota,
wrangler ↦ jeep,
renegade ↦ jeep,
compass ↦ jeep,
altima ↦ nissan,
pathfinder ↦ nissan,
murano ↦ nissan
}

Domain of car ←

→ Range of car

Model queries (1/2)

- Restriction operators can be used to model database queries.
- For example: "Select the pairs based on 'wrangler' and 'pathfinder'."

$$\{wrangler, pathfinder\} \lhd car = \{ wrangler \mapsto jeep, pathfinder \mapsto nissan \}$$

Domain restriction selects pairs based on the first element.

Model queries (2/2)

- Restriction operators can be used to model database queries.
- For example: “Select the pair(s) based on ‘jeep’.”

$$
\begin{array}{l}
\text{car} \rhd \{\text{jeep}\} = \\
\{ \\
\text{wrangler} \mapsto \text{jeep}, \\
\text{renegade} \mapsto \text{jeep}, \\
\text{compass} \mapsto \text{jeep} \\
\}
\end{array}
$$

Range restriction selects pairs based on the second element.

Model updates (1/3)

- Relational overriding can be used to model database updates.
- We have 2 types of updates: Insertions and modifications.
- For example, this update is an insertion:

car ⊕ {cherokee ↦ jeep} =
{
camry ↦ toyota,
avalon ↦ toyota,
highlander ↦ toyota,
wrangler ↦ jeep,
renegade ↦ jeep,
compass ↦ jeep,
altima ↦ nissan,
pathfinder ↦ nissan,
murano ↦ nissan,
cherokee ↦ jeep
}

All pairs of set on LHS are added to the set on RHS.

Model updates (2/3)

- Variable car’ holds the state of the set upon successful evaluation of the RHS expression.

$$
\boxed{\text{car}'} = \boxed{\text{car} \oplus \{\text{cherokee} \mapsto \text{jeep}\}} =
\begin{array}{l}
\{ \\
\begin{array}{lll}
\text{camry} & \mapsto & \text{toyota}, \\
\text{avalon} & \mapsto & \text{toyota}, \\
\text{highlander} & \mapsto & \text{toyota}, \\
\text{wrangler} & \mapsto & \text{jeep}, \\
\text{renegade} & \mapsto & \text{jeep}, \\
\text{compass} & \mapsto & \text{jeep}, \\
\text{altima} & \mapsto & \text{nissan}, \\
\text{pathfinder} & \mapsto & \text{nissan}, \\
\text{murano} & \mapsto & \text{nissan}, \\
\text{cherokee} & \mapsto & \text{jeep}
\end{array} \\
\}
\end{array}
$$

Model updates (3/3)

- Relational overriding can be used to model database updates.
- We have 2 types of updates: Insertions and modifications.
- For example, this update is a modification:

car' = car ⊕ {avalon ↦ lexus} =
{
avalon ↦ lexus,
camry ↦ toyota,
highlander ↦ toyota,
wrangler ↦ jeep,
renegade ↦ jeep,
compass ↦ jeep,
altima ↦ nissan,
pathfinder ↦ nissan,
murano ↦ nissan,
cherokee ↦ jeep
}

All pairs from set on LHS except avalon ↦ toyota, are added to the set on the set of the RHS.

Model deletions (1/2)

- “Remove all pairs where model is ‘camry’, ‘murano’, or ‘altima’.”

car’ = {camry, murano, altima} ⩤ car =

{

avalon ↦ lexus,

~~camry ↦ toyota,~~

highlander ↦ toyota,

wrangler ↦ jeep,

renegade ↦ jeep,

compass ↦ jeep,

~~altima ↦ nissan,~~

pathfinder ↦ nissan,

~~murano ↦ nissan,~~

cherokee ↦ jeep

}

Domain subtraction removes all elements from the domain of the relation.

Model deletions (2/2)

- “Remove all pairs where make is ‘jeep’.”

car’ = car $\rhd$ {jeep} =
{
avalon ↦ lexus,
highlander ↦ toyota,
~~wrangler ↦ jeep,~~
~~renegade ↦ jeep,~~
~~compass ↦ jeep,~~
pathfinder ↦ nissan,
~~cherokee ↦ jeep~~
}

Range subtraction removes all elements from the range of the relation.

Binary relations and functions revisited

- In the example, the binary relation `car` is expressed as a set of ordered pairs.

- Formally, `car` is defined as `car: Model ↔ Make`

- Alternatively we can formally define `car` as `car ⊆ Model x Make`

- The relation `car` is also a function, and it can be formally defined as `car:  Model → Make`

- Both relations and functions define relationships between sets.

- Both relations and functions can be represented as sets.

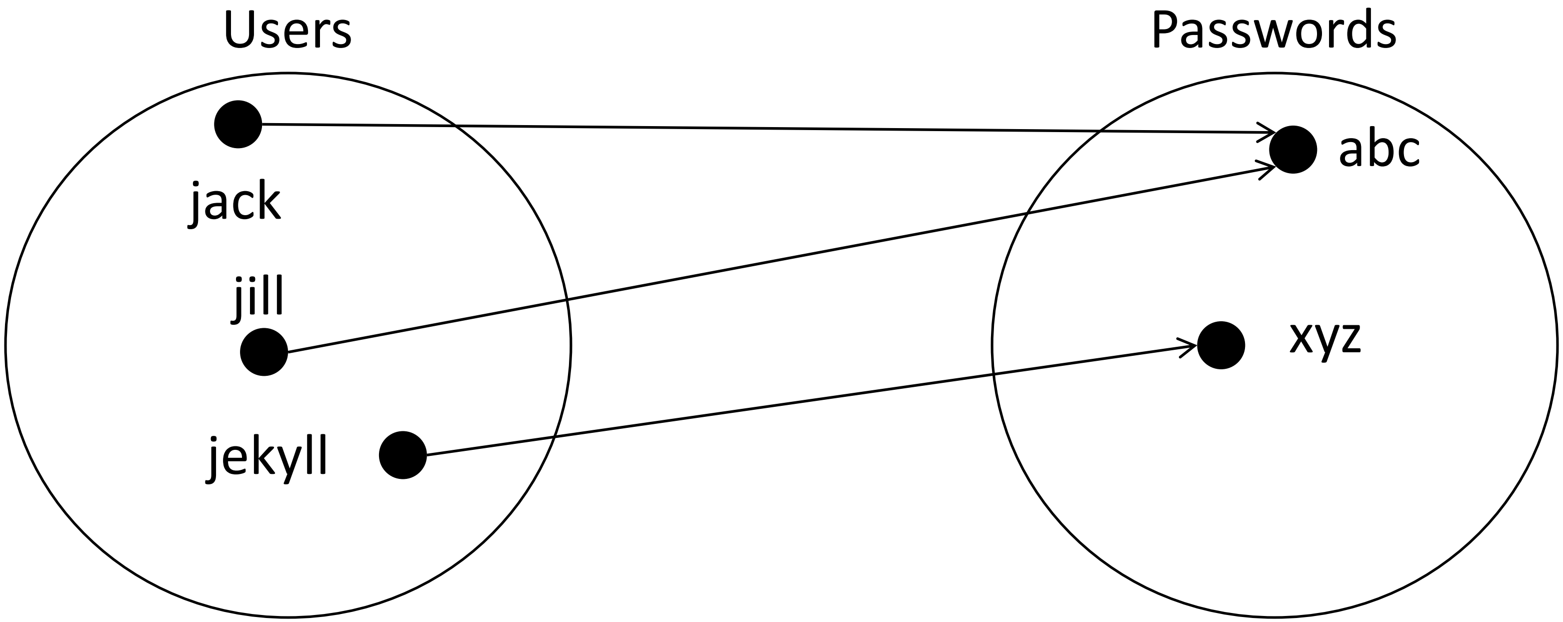
- Not all relations are functions, but all functions are relations.

Example 3: A password file

- Requirements:
  1. Each user has a unique user-id.
  2. Each user-id is associated with only one password.
  3. Users may have a common password.

- We define types Users and Passwords.

- A possible state of the system is shown below:



Mapping and visualization

- The two types Users and Passwords are represented as sets.
- We capture the mapping from `Users` to `Passwords` by a set of ordered pairs which we will call `password`.
- The initial state of the system can thus be expressed as
  `password = {jack ↦ abc, jill ↦ abc, jekyll ↦ xyz}`
- Note that `password` is a (binary) relation, since
  `password ⊆ Users x Passwords`
- The relation `password` is also a function, i.e.
  `password: Users → Passwords`

Properties of function ‘password’

- Function password is partial (as opposed to total) as it maps a subset of the domain set (Users). Formally this will be denoted as

  `password: Users ⇸ Passwords`

- Is not injective (one-to-one) because it is not the case that

  $$\forall \mathtt{user}_1, \mathtt{user}_2 : \mathtt{Users}\ (\mathtt{user}_1 \neq \mathtt{user}_2 \rightarrow \mathtt{password}(\mathtt{user}_1) \neq \mathtt{password}(\mathtt{user}_2))$$

- Is not surjective (onto) as it is not the case that

  $$\forall \mathtt{passwd} : \mathtt{Passwords}\ \exists \mathtt{user} : \mathtt{Users}\ (\mathtt{password}(\mathtt{user}) = \mathtt{passwd})$$

- By definition it is not bijective (one-to-one correspondence).

Case 1: Adding a new user (with a precondition)

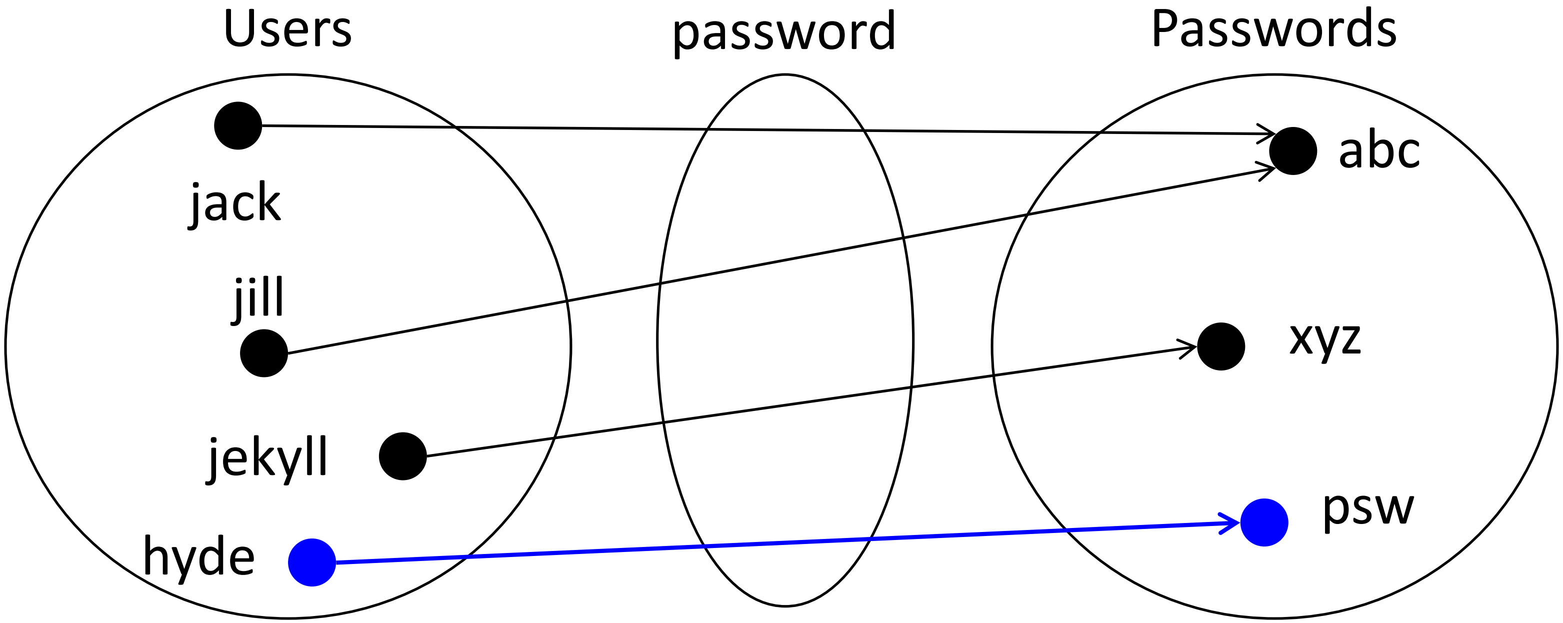
password =

{
jack ↦ abc,
jill ↦ abc,
jekyll ↦ xyz,
hyde ↦ psw
}

pre: user ∉ dom password

Precondition successful

password' =
password U {hyde ↦ psw}
or
password ⊕ {hyde ↦ psw}



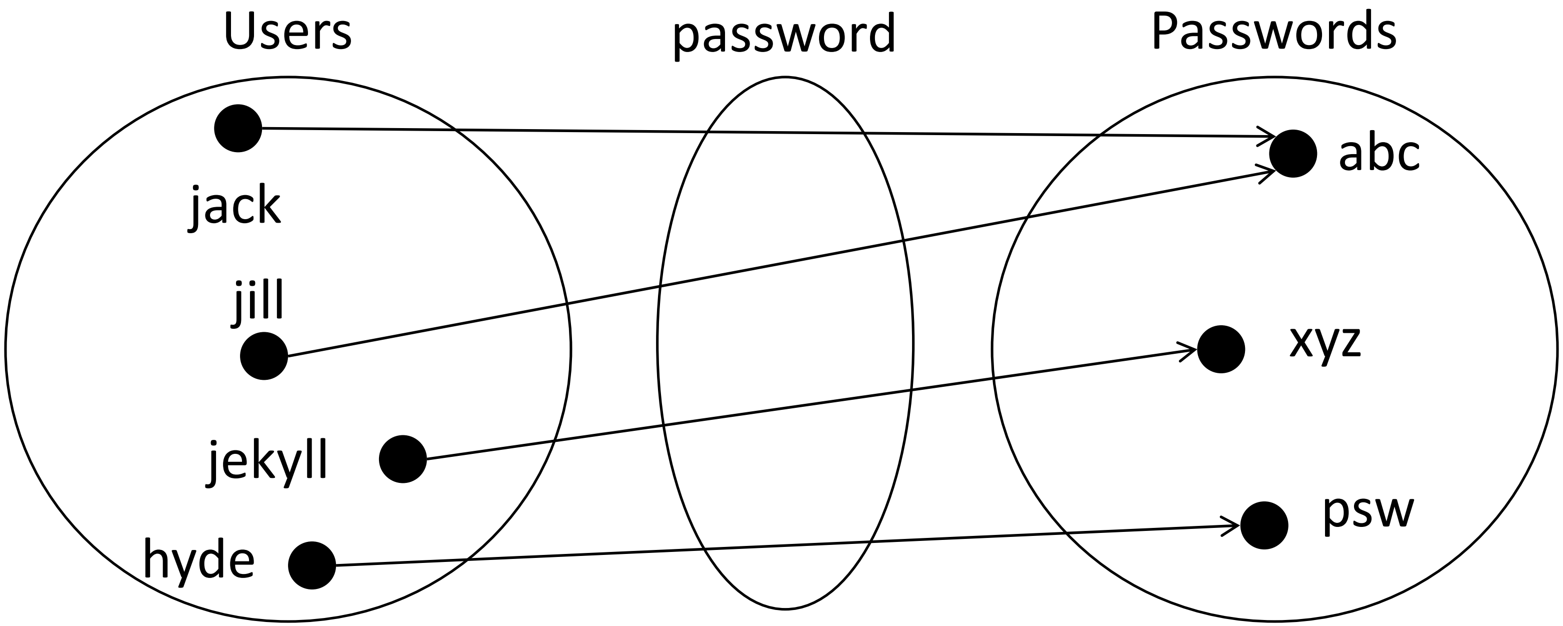
Case 1: Adding a new user (with a precondition) /cont.

password =
{
jack ↦ abc,
jill ↦ abc,
jekyll ↦ xyz,
hyde ↦ psw
}

pre: user ∉ dom password

Wish to add: jekyll ↦ psw

Precondition failure



Case 2: Adding a new user with set union (without a precondition)

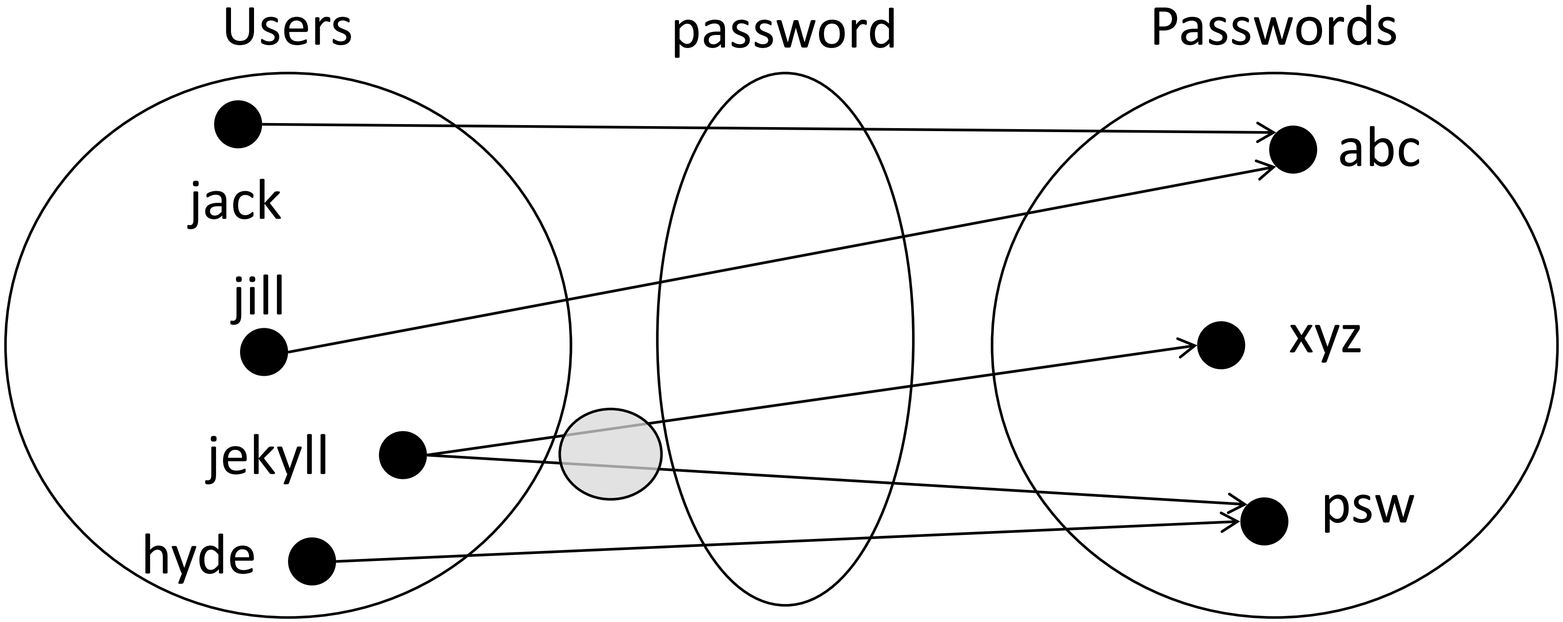
password =

{
jack ↦ abc,
jill ↦ abc,
jekyll ↦ xyz,
jekyll ↦ psw,
hyde ↦ psw
}

password' =

password U {jekyll ↦ psw}

Violation of requirements



Case 2: Adding a new user with set union (without a precondition) /cont.

password =
{
jack ↦ abc,
jill ↦ abc,
jekyll ↦ xyz,
hyde ↦ psw
}

password’ =
password U {jekyll ↦ psw}

- In the absence of a precondition, will there always be a violation of requirements with set union?
- If there exists no such user, then the user-password pair will be added to the file. No violation of requirements.
- If there already exists such user, then the existing user-password pair will also be added to the file. Violation of requirements (duplicate user).

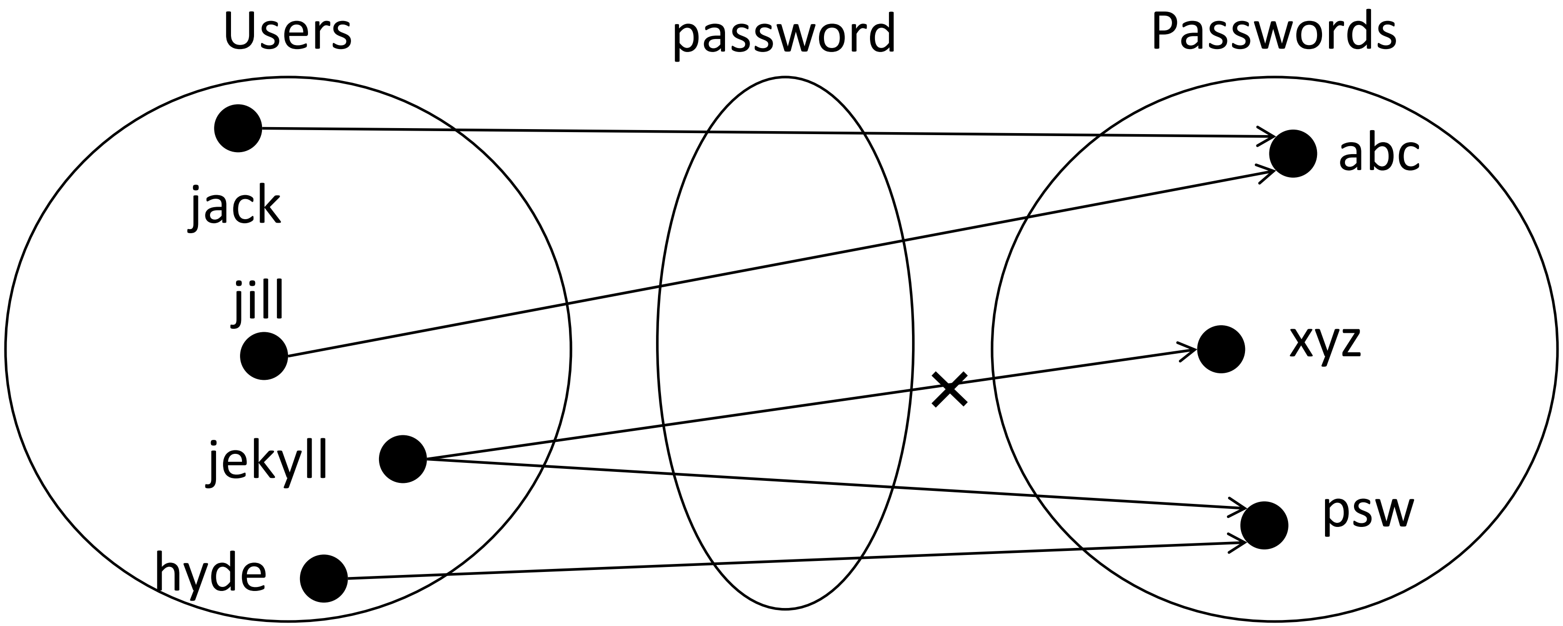
Case 2: Adding a new user with relational override (without a precondition)

password =

{
jack ↦ abc,
jill ↦ abc,
~~jekyll ↦ xyz,~~
jekyll ↦ psw,
hyde ↦ psw
}

password’ =
password ⊕ {jekyll ↦ psw}

Violation of requirements



Case 2: Adding a new user with relational override (without a precondition) /cont.

password =

$$\{ jack \mapsto abc, jill \mapsto abc, jekyll \mapsto xyz, hyde \mapsto psw \}$$

password' =

$$password \oplus \{jekyll \mapsto psw\}$$

- In the absence of a precondition, will there always be a violation of requirements with relational override?

- If there exists no such user, then the user-password pair will be added to the file. No violation of requirements.

- However, if there already exists such user, then the existing user-password pair will be replaced by a new such pair. Violation of requirements.

Case 3: Modifying the password of an existing user (with a precondition)

password =

{

jack ↦ abc,

~~jill ↦ abc,~~

jill ↦ xyz,

jekyll ↦ xyz,

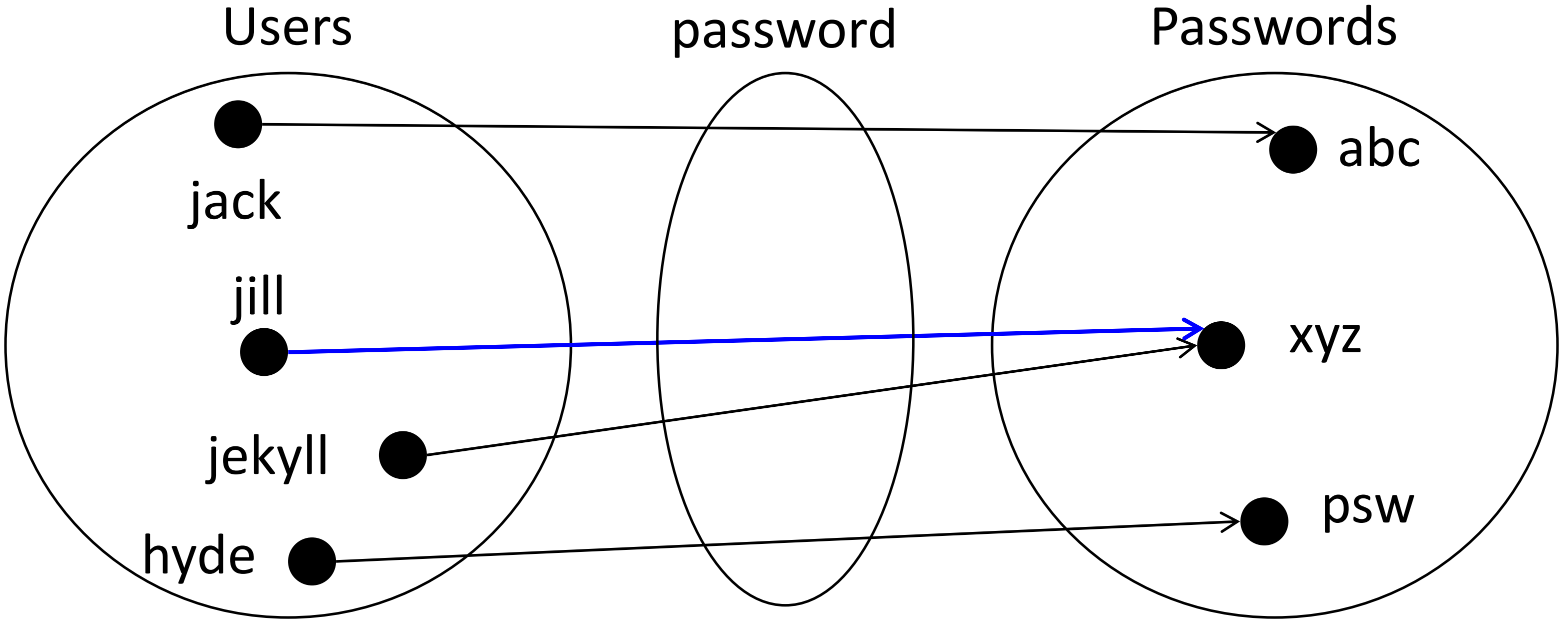
hyde ↦ psw

}

pre: user ∈ dom password

Wish to modify jill's password to xyz

password' = password ⊕ {jill ↦ xyz}



Case 3: Modifying the password of an existing user (with a precondition) /cont.

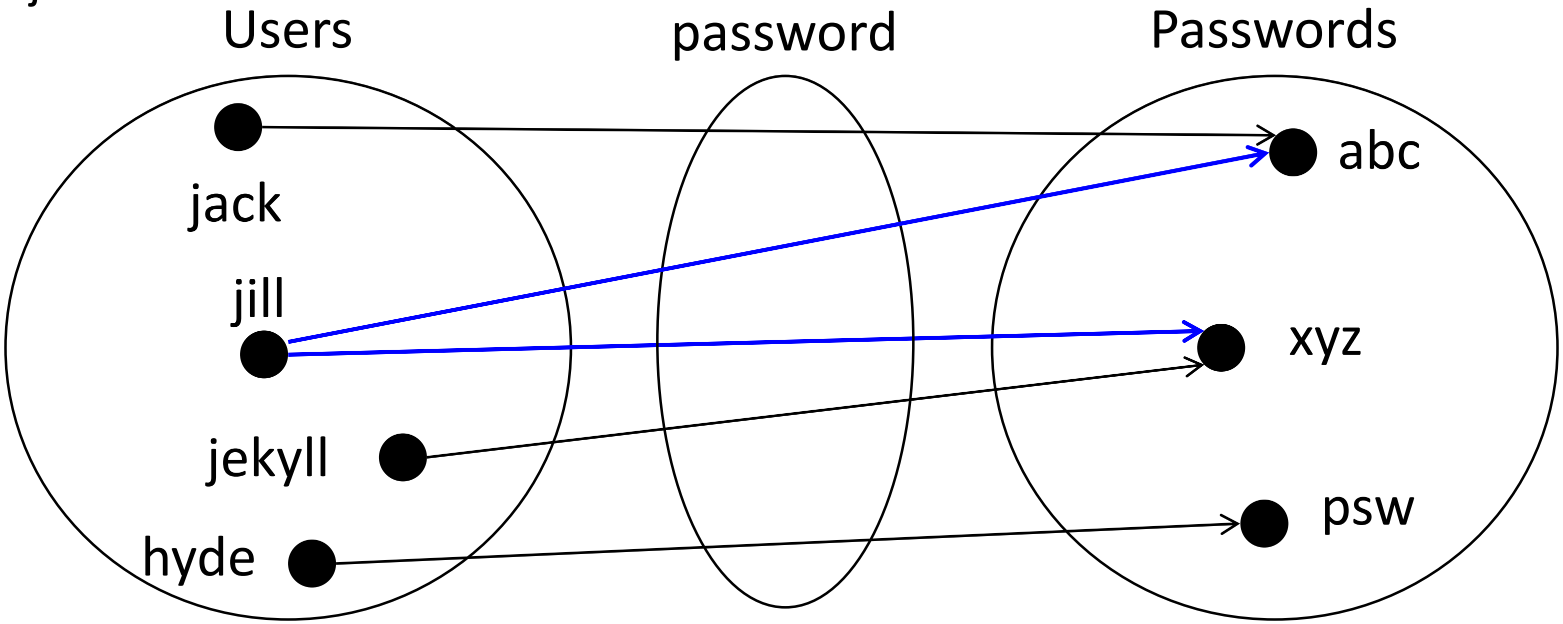
password =
{
jack $\mapsto$ abc,
jill $\mapsto$ abc,
jill $\mapsto$ xyz,
jekyll $\mapsto$ xyz,
hyde $\mapsto$ psw
}

Should be excluded !

pre: user $\in$ dom password
Wish to modify jill's password to xyz

How about
password' = password U {jill $\mapsto$ xyz} ?

Violation of requirements



Example 4: Revisiting Sets, Binary Relations and Functions

- Recall that:

1. **Binary relations are sets**.
   (though not all sets are binary relations)

2. **Functions are binary relations**.
   (though not all binary relations are functions)

Relational overriding

- Consider the following relations:

R = { (Mike, 20), (Roger, 18), (Anne, 23) }

dom **R** = { Mike, Roger, Anne }, **ran** **R** = { 20, 18, 23 }

S = { (Anne, 20), (Yang, 19) }

dom **S** = { Anne, Yang }, and **ran** **S** = { 20, 19 }

- Relational override is a **binary operation** on relations.

R ⊕ **S** is defined as a set that contains all pairs of **S** plus those pairs from **R** whose first coordinates do not belong to the domain of **S**.

R $\oplus$ **S** is defined as a set that contains all pairs of **S** plus those pairs from **R** whose first coordinates do not belong to the domain of **S**.

R = { (**Mike**, **20**), (Roger, 18), (Anne, 23) }

S = { (Anne, 20), (Yang, 19) }

dom S = { Anne, Yang }

- Will (**Mike**, **20**) be added to the resulting set?

  Yes, because **Mike** is not in **dom S**.

  Result (so far):

  { (Anne, 20), (Yang, 19), (Mike, 20) }

R $\oplus$ **S** is defined as a set that contains all pairs of **S** plus those pairs from **R** whose first coordinates do not belong to the domain of **S**.

R = { (Mike, 20), (**Roger**, **18**), (Anne, 23) }

S = { (Anne, 20), (Yang, 19) }

dom **S** = { Anne, Yang }

- Will (**Roger**, **18**) be added to the resulting set?

  Yes, because **Roger** is not in **dom** **S**.

  Result (so far):

  { (Anne, 20), (Yang, 19), (Mike, 20), (Roger, 18) }

R $\oplus$ **S** is defined as a set that contains all pairs of **S** plus those pairs from **R** whose first coordinates do not belong to the domain of **S**.

R = { (Mike, 20), (Roger, 18), (**Anne**, **23**) }

S = { (Anne, 20), (Yang, 19) }

dom **S** = { Anne, Yang }

- Will (**Anne**, **23**) be added to the resulting set?

  No, because **Anne** is in **dom** **S**.

  Final result:

  { (Anne, 20), (Yang, 19), (Mike, 20), (Roger, 18) }

- In this example, the two binary relations are both functions:

  R = { (Mike, 20), (Roger, 18), (Anne, 23) }, and

  S = { (Anne, 20), (Yang, 19) }

- Is **R** U **S** a function? No.
- Is **R** $\oplus$ **S** or **S** $\oplus$ **R** a function? Yes.

Example 5: Set Union vs. Relational Override

- Consider the following relations:

S = { (**Ali**, **555**), (Ellie, 100), (Bruce, 430) }

dom **S** = { Ali, Ellie, Bruce }, **ran** **S** = { 555, 100, 430 }

T = { (Bruce, 400), (Bruce, 300), (Ellie, 999) }

dom **T** = { Bruce, Ellie }, and **ran** **T** = { 400, 300, 999 }

- Need to calculate **S** $\oplus$ **T**

- Will (**Ali**, **555**) be added to the resulting set?

Yes, because **Ali** is <u>not</u> in **dom** **T**.

Result (so far):

{ (Bruce, 400), (Bruce, 300), (Ellie, 999) , (Ali, 555) }

- Consider the following relations:

  S = { (Ali, 555), (**Ellie**, **100**), (Bruce, 430) }
  dom S = { Ali, Ellie, Bruce }, **ran S** = { 555, 100, 430 }

  T = { (Bruce, 400), (Bruce, 300), (Ellie, 999) }
  dom T = { Bruce, Ellie }, and **ran T** = { 400, 300, 999 }

- Need to calculate **S** $\oplus$ **T**

- Will (**Ellie**, **100**) be added to the resulting set?
  No, because **Ellie** is in **dom T**.

  Result (so far):
  { (Bruce, 400), (Bruce, 300), (Ellie, 999) , (Ali, 555) }

- Consider the following relations:

↘

S = { (Ali, 555), (Ellie, 100), (**Bruce**, **430**) }

dom **S** = { Ali, Ellie, Bruce }, **ran** **S** = { 555, 100, 430 }

T = { (Bruce, 400), (Bruce, 300), (Ellie, 999) }

dom **T** = { Bruce, Ellie }, and **ran** **T** = { 400, 300, 999 }

- Need to calculate **S** ⊕ **T**

- Will (**Bruce**, **430**) be added to the resulting set?

No, because **Bruce** is in **dom** **T**.

Final result:

{ (Bruce, 400), (Bruce, 300), (Ellie, 999) , (Ali, 555) }

- In this example, it is not the case that both binary relations are functions:

S = { (Ali, 555), (Ellie, 100), (Bruce, 430) }

T = { (Bruce, 400), (Bruce, 300), (Ellie, 999) }

Example 6: A final example on Relational Override

- Consider the following relations:

S = { (Ali, 555), (Ellie, 100), (Ellie, 88), (Sanjay, 100), (Cho, 300),
(Sanjay, 999), (Bruce, 001)
}

T = { (Joseph, 530), (Bruce, 300) }

- **S** ⊕ **T** = {
(Joseph, 530), (Bruce, 300),
(Ali, 555), (Ellie, 100), (Ellie, 88), (Sanjay, 100),
(Cho, 300), (Sanjay, 999)
}

All the pairs of **T**, …

…plus those pairs from **S** whose first coordinate is not in the domain of **T**.