Data and Process Modelling

3. Object-Role Modeling - CSDP Step 6

Marco Montali

KRDB Research Centre for Knowledge and Data
Faculty of Computer Science
Free University of Bozen-Bolzano

Other Constraints

CSDP Step 6

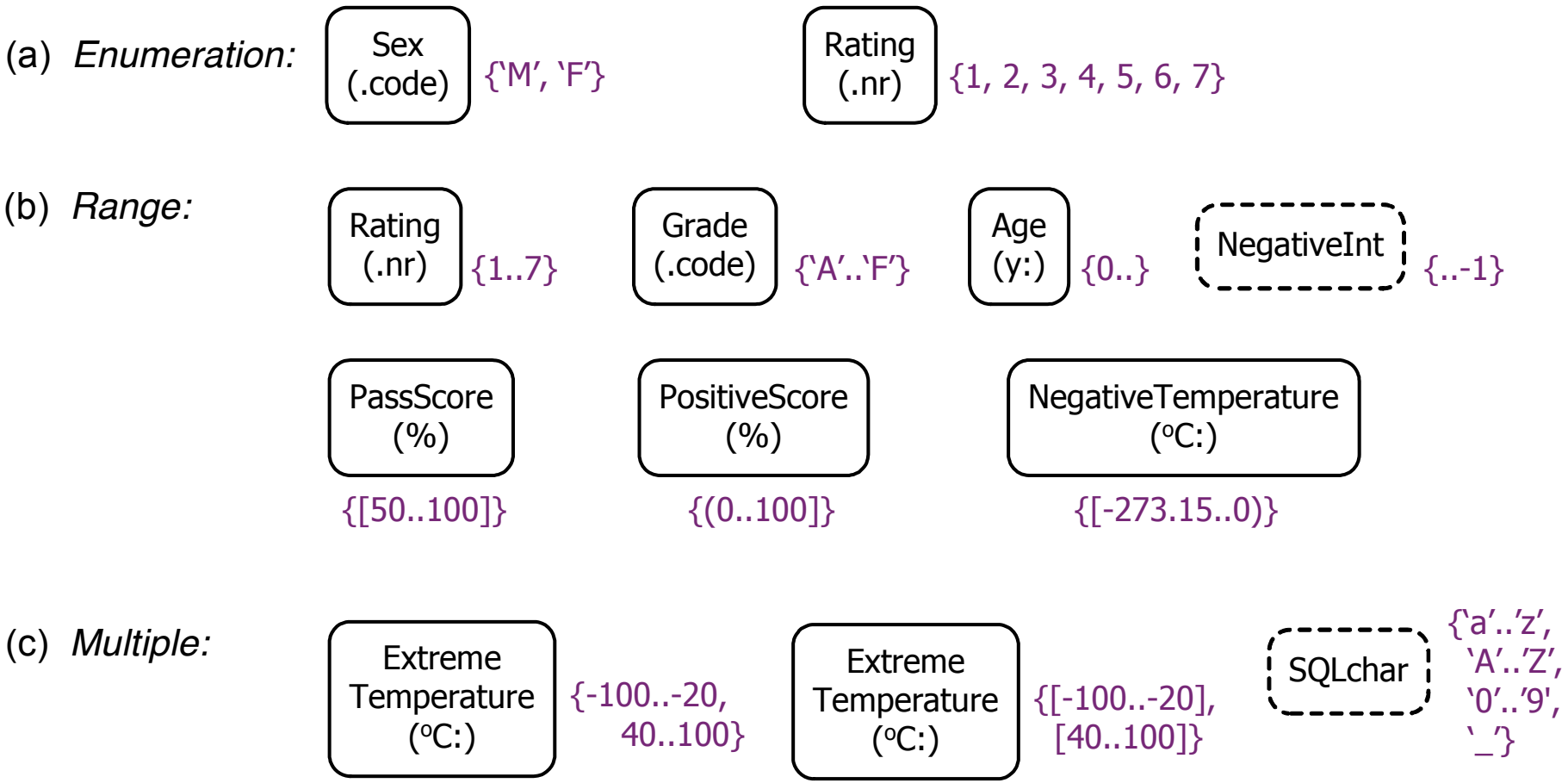
Add value, subset, equality, exclusion, and subtype constraints.

- Value (domain) constraint: which values are allowed in a value type or role.
- Set constraint: how the population of one role (sequence) relates to the population of another role (sequence).
  - Subset ($\subseteq$): $\text{pop}(r_1) \subseteq \text{pop}(r_2)$.
  - Equality ($=$): $\text{pop}(r_1) = \text{pop}(r_2)$.
  - Exclusion (X): $\text{pop}(r_1) \cap \text{pop}(r_2) = \emptyset$.
- Subtype constraint: specialization/generalization of object types to point out specific features/factorize common features.

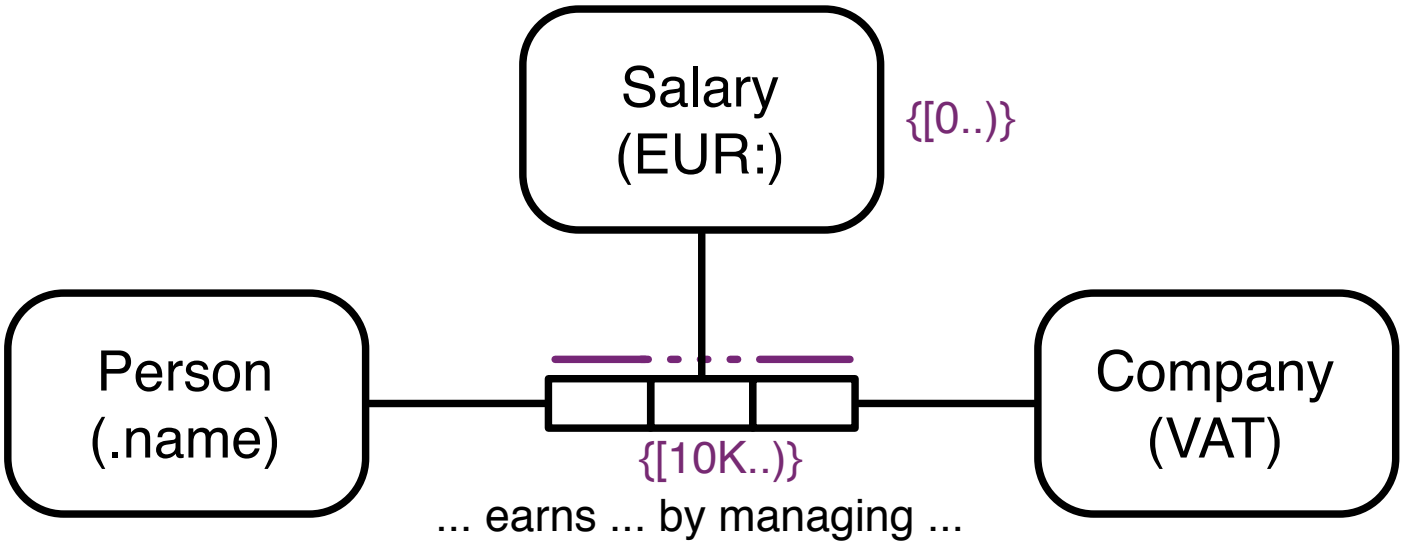
Value Constraint

- Constrains the possible values that can be assumed by a value type or a role.
- Can be used only when it is *stable*.
- Definition by specifying the extension of the allowed values: one or more enumerations or ranges enclosed in $\{\ldots\}$.
  - Enumeration: explicit list of the allowed values (e.g., {'M','F'}).
  - Range: implicit continuous list by just specifying the extreme values. In case of real number, square bracket signifies inclusion of the extreme, parenthesis exclusion.
- Object type value constraint: next to the value type, or the object type that is referenced by the value type.
- Role value constraint: next the constrained role.
- Independent object types can be used when we want to store information about an object type but the list is too large, unstable, or not known a priori.

Object Type Value Constraints

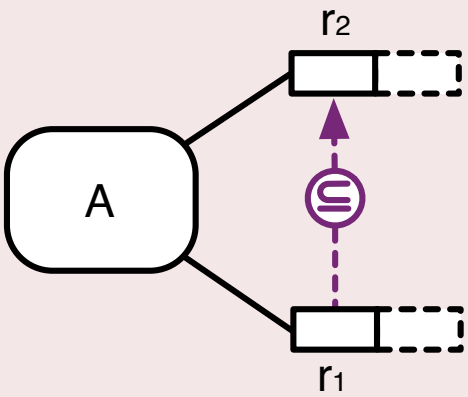


Role Value Constraints



- The salary is a positive real value.
- Managers earn at least 10.000 EUR per month.
- Value constraints on the role must be consistent with the value constraints of the attached object type.

Subset Constraint



For each information base state: $pop(r_1) \subseteq pop(r_2)$.

If some A plays r_1 then that A plays r_2.

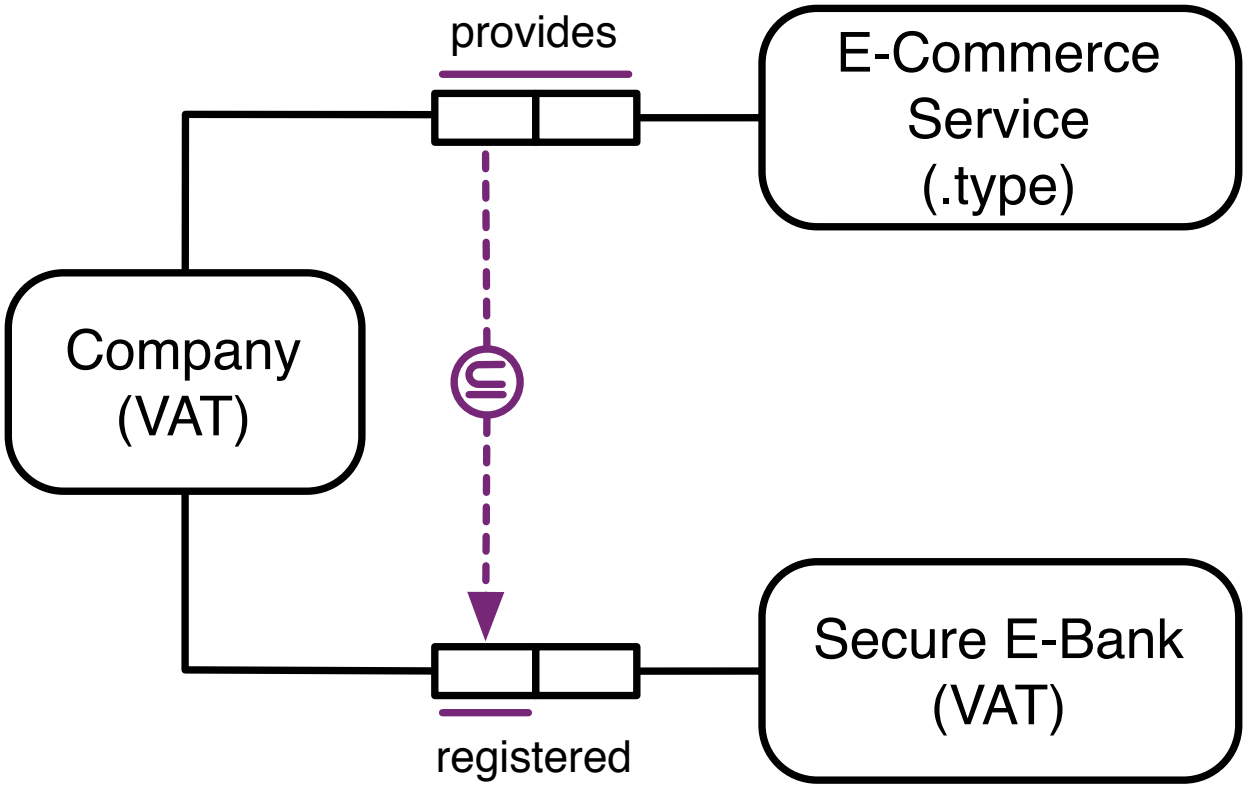
- Each object that populates r_1 must also populate r_2.
- N.B.: comparison is set-theoretic, therefore duplicates do not matter.
- For the comparison to make sense, both involved roles must be played by the same object type (or supertype, see *subtyping*).

Subset Constraint: Example

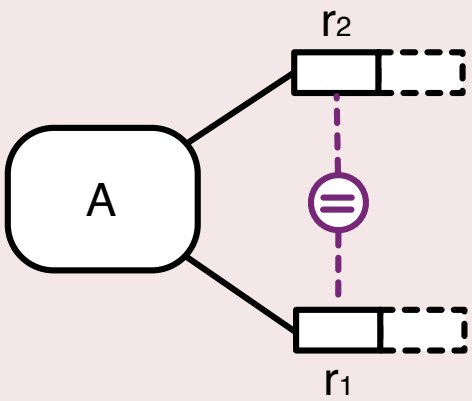
- A Company could be registered to at most one Secure E-Bank Service.
- A Company could provide E-Commerce Services.
- If some Company provides some Secure E-Commerce Service, that Company must be registered to a Secure E-Bank Service.

Subset Constraint: Example

- A Company could be registered to at most one Secure E-Bank Service.
- A Company could provide E-Commerce Services.
- If some Company provides some Secure E-Commerce Service, that Company must be registered to a Secure E-Bank Service.



Equality Constraint



For each information base state: $pop(r_1) = pop(r_2)$.

For each A, that A plays r_1 if and only if that A plays r_2.

- Each object that populates r_1 must also populate r_2, *and vice-versa*.
- N.B.: comparison is set-theoretic, therefore duplicates do not matter.
- For the comparison to make sense, both involved roles must be played by the same object type (or supertype, see *subtyping*).
- What about an equality constraint combined with:
  - one mandatory and one optional role?
  - no mandatory role, but mandatory disjunction?
  - two mandatory roles?

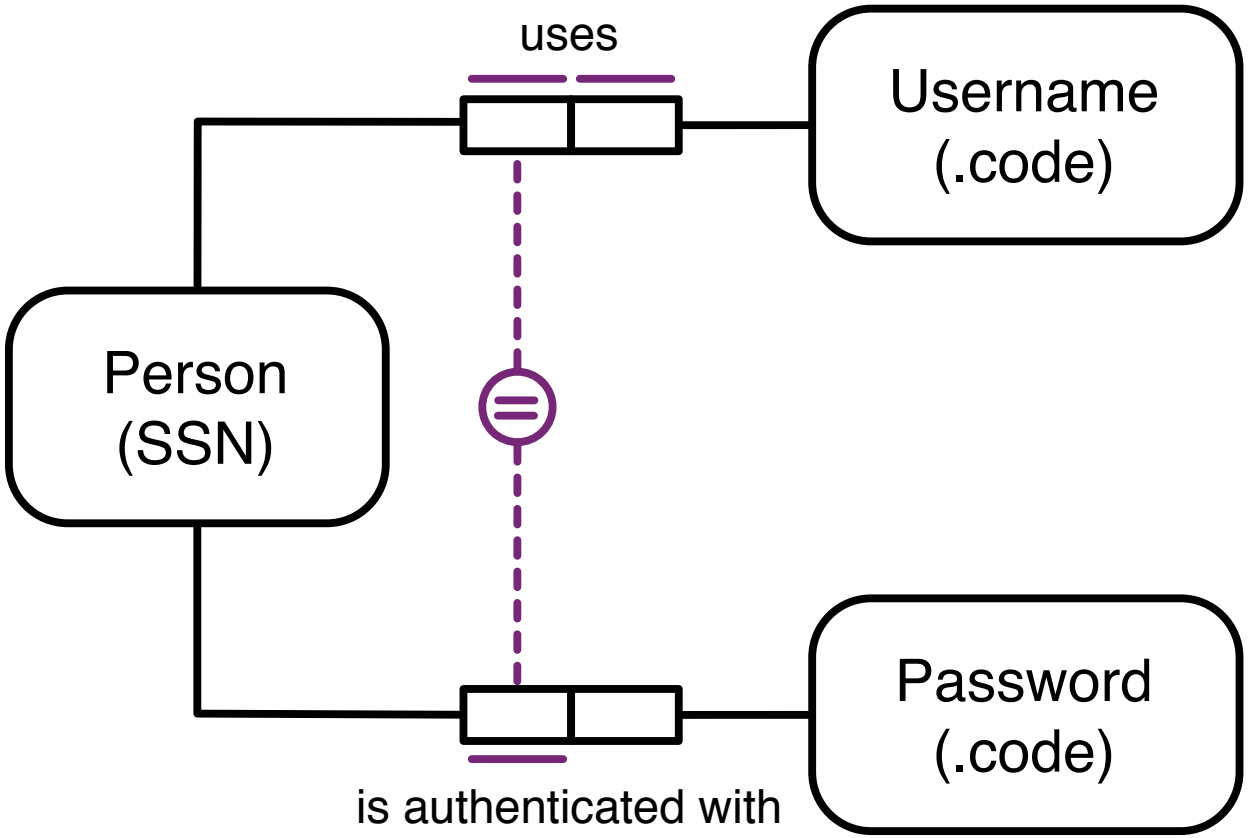
  (apply set theory/logic)

Equality Constraint: Example

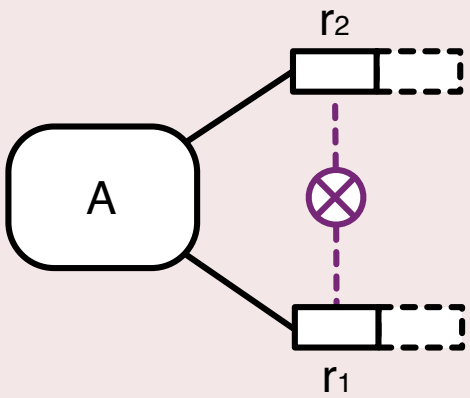
- A Person could use an Username, and be authenticated with a Password.
- Either a Person does not have a Username nor a Password, or the Person has both.

Equality Constraint: Example

- A Person could use an Username, and be authenticated with a Password.
- Either a Person does not have a Username nor a Password, or the Person has both.



Exclusion Constraint

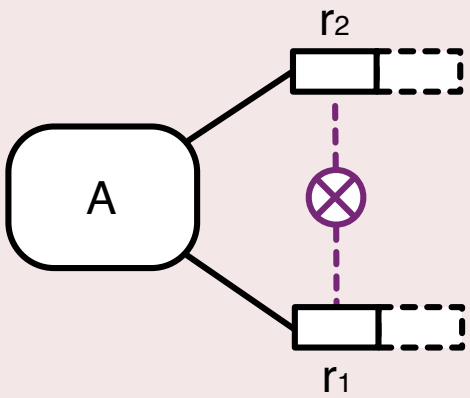


For each information base state:
$pop(r_1) \cap pop(r_2) = \emptyset$.

For each A, at most one of the following holds: A plays r_1 ; A plays r_2.

- Each object that populates r_1 cannot populate r_2, *and vice-versa*.
- N.B.: comparison is set-theoretic, therefore duplicates do not matter.
- For the comparison to make sense, both involved roles must be played by the same object type (or supertype, see *subtyping*), and *such roles must be optional*.
  - Why?

Exclusion Constraint



For each information base state:
$pop(r_1) \cap pop(r_2) = \emptyset$.

For each A, at most one of the following holds: A plays r_1 ; A plays r_2.

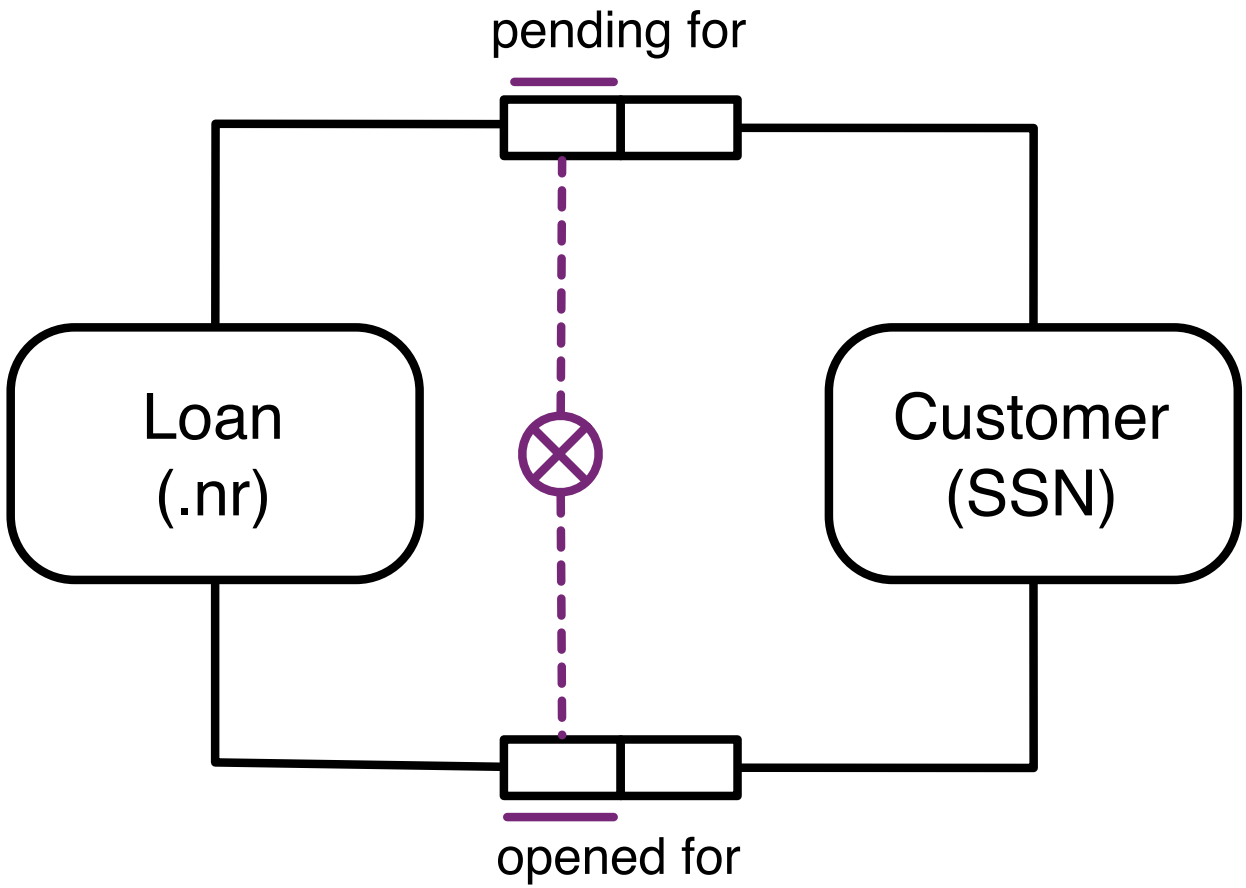
- Each object that populates r_1 cannot populate r_2, *and vice-versa*.
- N.B.: comparison is set-theoretic, therefore duplicates do not matter.
- For the comparison to make sense, both involved roles must be played by the same object type (or supertype, see *subtyping*), and *such roles must be optional*.
  - Why? If r_1 were mandatory, then r_2 could never be played by A.

Exclusion Constraint: Example

- A Loan is either pending for a (single) Customer, or open for that Customer, but not both.

Exclusion Constraint: Example

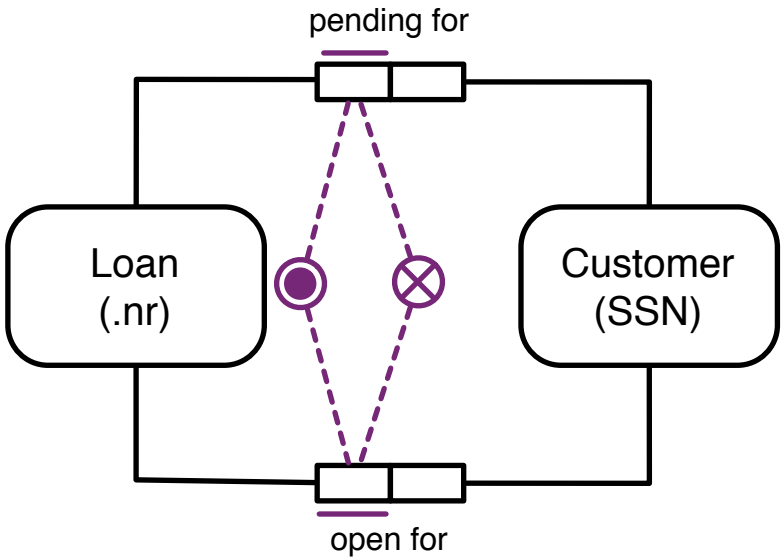
- A Loan is either pending for a (single) Customer, or open for that Customer, but not both.



- Are we fully capturing its semantics? In a stable way?

Exclusive-Or Constraint

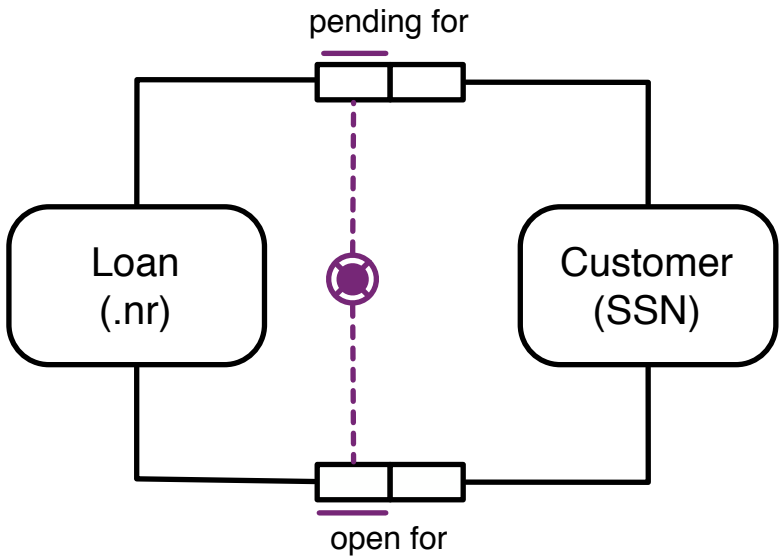
- Combination of exclusion constraint and inclusive-or constraint over the same roles → exclusive-or: each object whose type is connected to that roles plays *exactly one* of the constrained roles.
- Consider the Loan example, adding that each Loan is either pending or open.



- Resulting constraint: `For each Loan, exactly one of the following holds: that Loan is pending for some Customer; that Loan is open for some Customer.`

Exclusive-Or Constraint

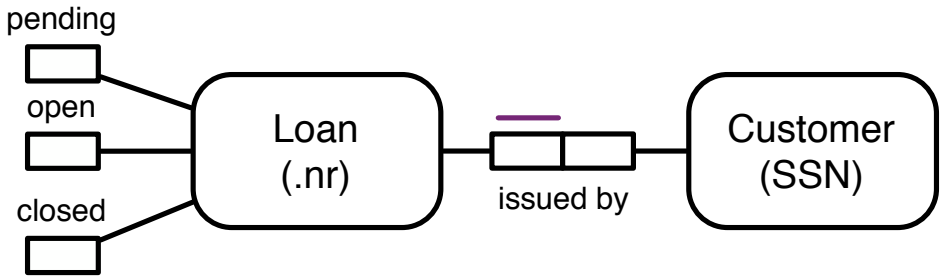
- Combination of exclusion constraint and inclusive-or constraint over the same roles → exclusive-or: each object whose type is connected to that roles plays *exactly one* of the constrained roles.
- Consider the Loan example, adding that each Loan is either pending or open.



- Resulting constraint: `For each Loan, exactly one of the following holds: that Loan is pending for some Customer; that Loan is open for some Customer.`
- Representable by superimposing the constraints (lifebuoy symbol).

Loan Example

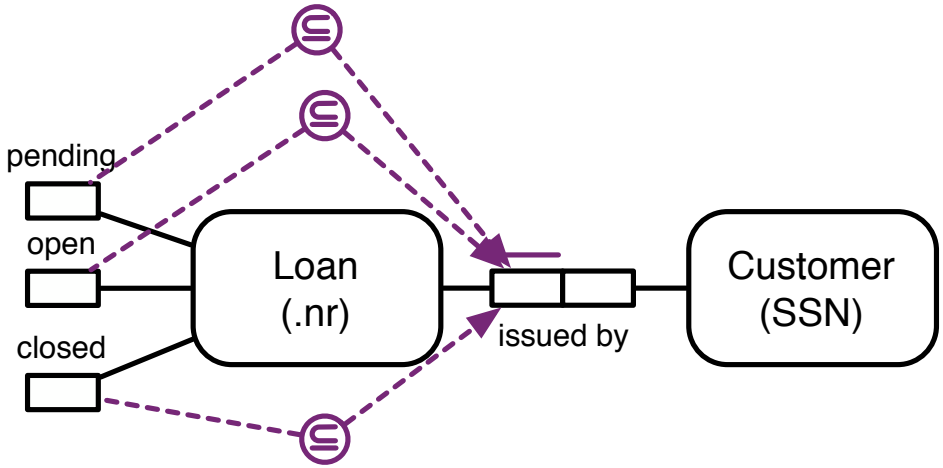
- Problem in the Loan example: there is no guarantee that the same Customer is associated to a given Loan once the Loan is moved from pending to open.
- Better modeling, considering also a further state:



- This is the initial model, let's add the constraints.

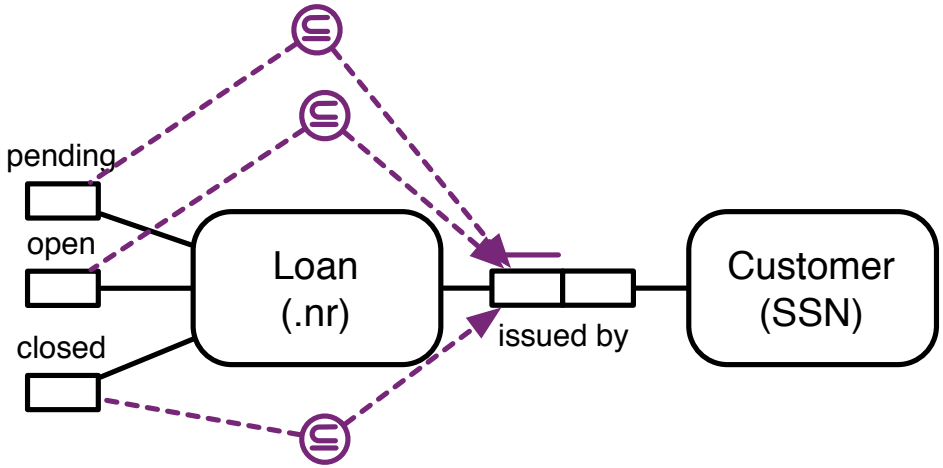
Loan Example

- A Loan in a certain state must always be associated to a customer
  $\rightarrow$ subset constraints!

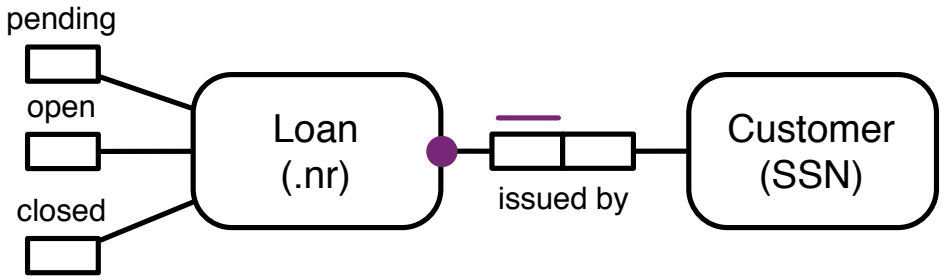


Loan Example

- A Loan in a certain state must always be associated to a customer $\rightarrow$ subset constraints!

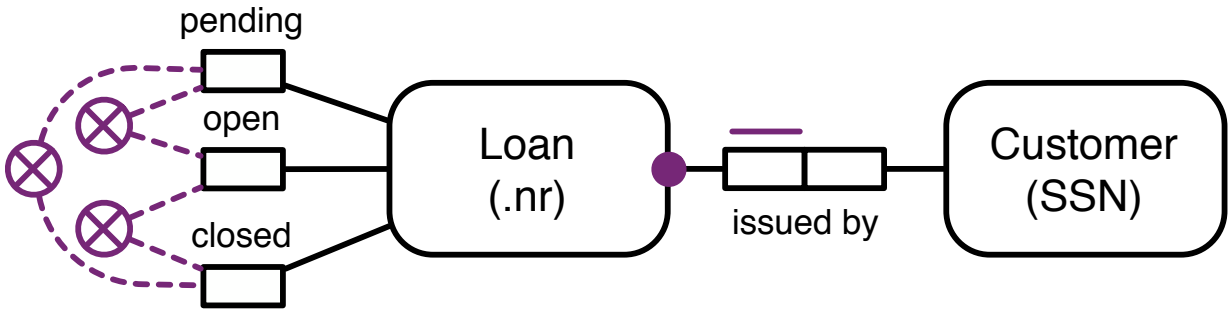


- Remember: the population of a non-isolated object type is characterized by the population of its roles.
- We can hence simplify: all the roles have a subset constraint pointing to "issued by", therefore Loan's role there is mandatory!

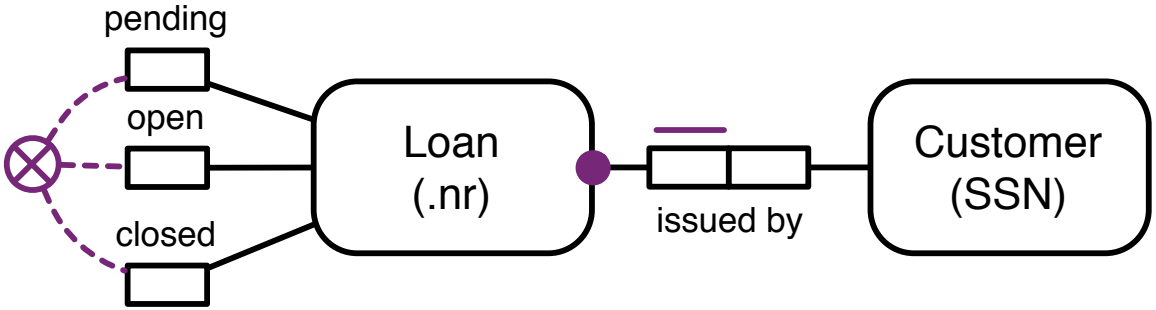


Loan Example

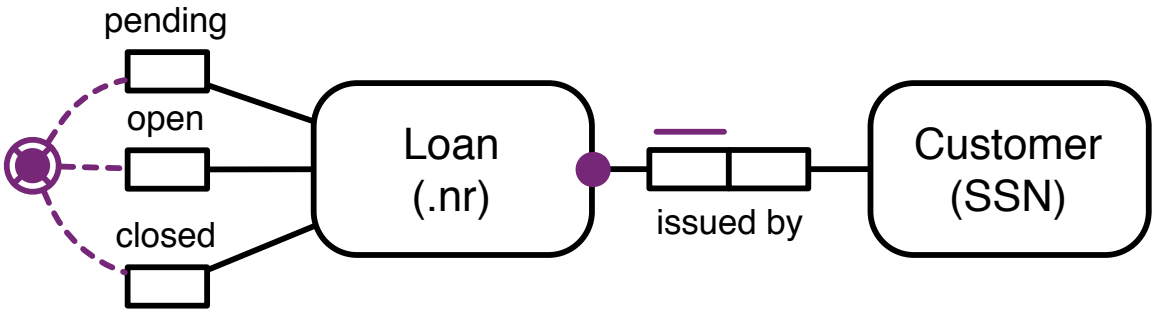
- The "state" roles are exclusive.



- This notation can be simplified as follows.

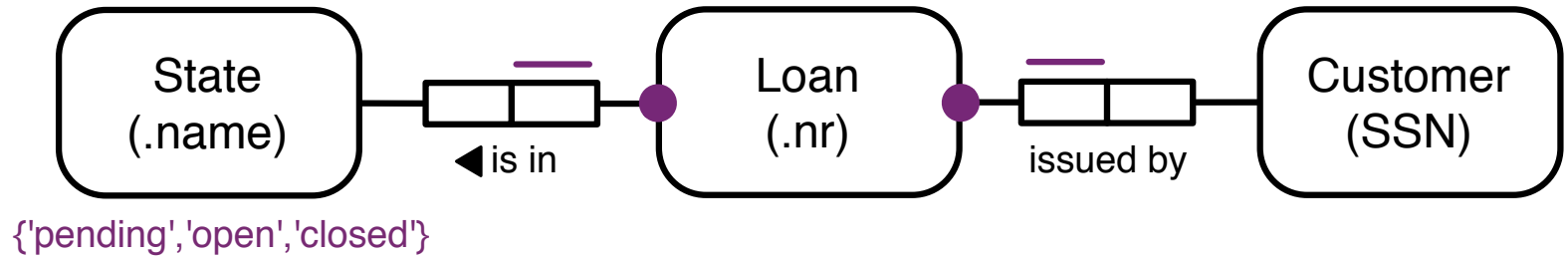


- A Loan is always in one of the states (inclusive-or, to be mixed with exclusion $\rightarrow$ ex-or).



Loan Example

- We could decide to make the schema more flexible.
- The notion of "State" could be explicitly modeled for Loans.
- Enumeration constraint to capture the possible states (easily extensible).
- The "exactly-one state" constraint modeled using the ex-or constraint now becomes simply a 1-1 participation.

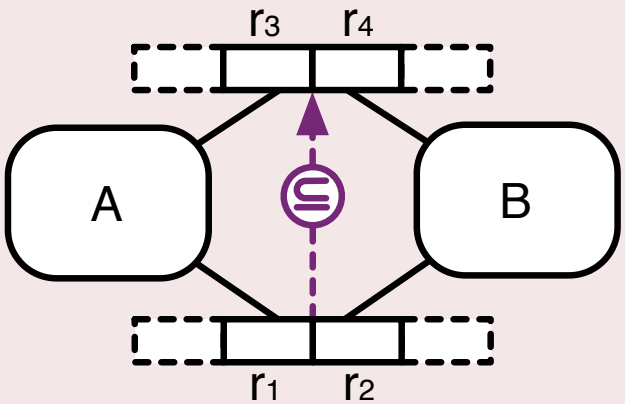


Set-Comparison with Role Sequences

Constraints seen in this step can be also applied to role sequences.

- We focus here on set-comparison.

Pair-subset constraint



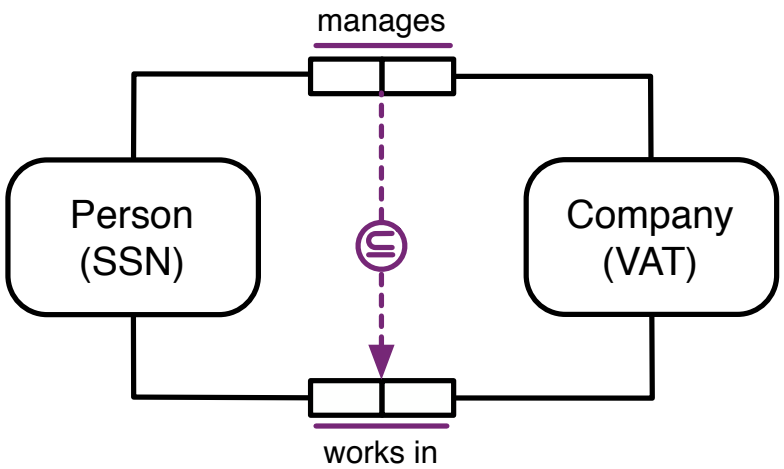
For each information base state:
$pop(r_1, r_2) \subseteq pop(r_3, r_4)$.

Each pair in $pop(r_1, r_2)$ **is also in** $pop(r_3, r_4)$.

- Can be generalized to n-ary sequences.
- The same holds for equality and exclusion constraints.

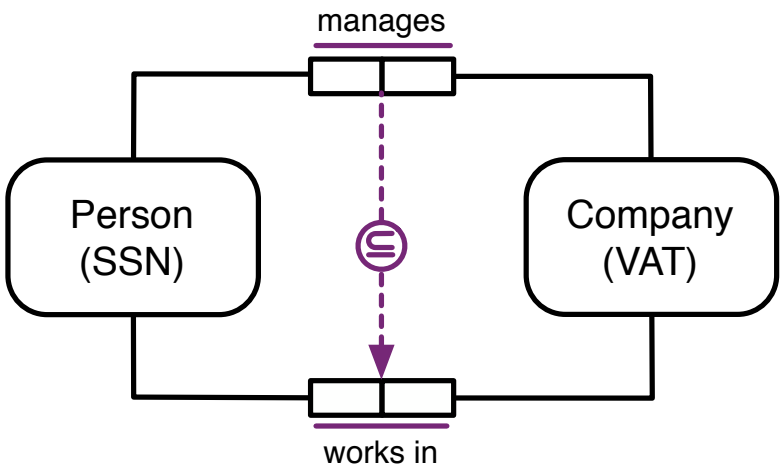
Pair-subset: Example

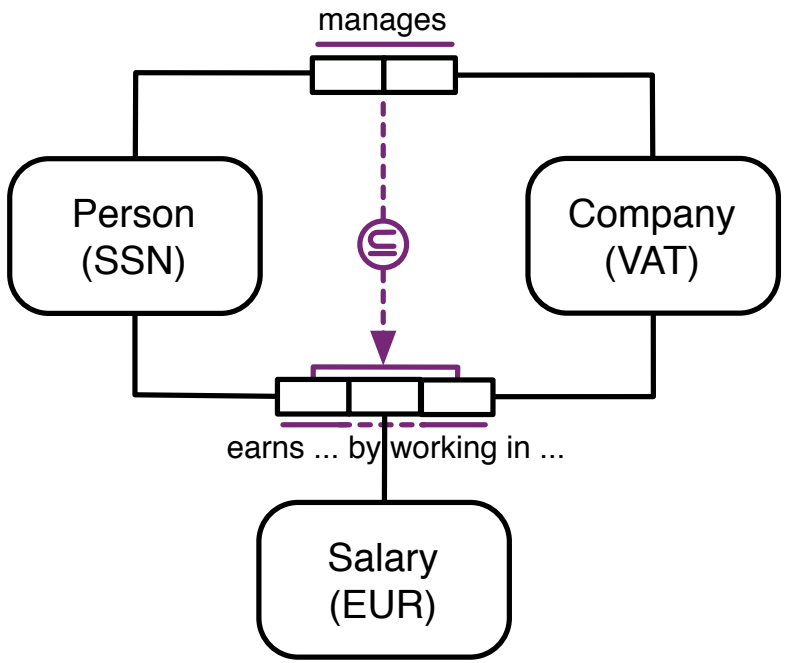
- Consider Persons who work in Companies and Persons who manage Companies.
- Clearly, **`Each`** `Person` **`who`** `manages a Company works in` **`that`** `Company`.



Pair-subset: Example

- Consider Persons who work in Companies and Persons who manage Companies.
- Clearly, **`Each`** `Person` **`who`** `manages a Company works in` **`that`** `Company`.

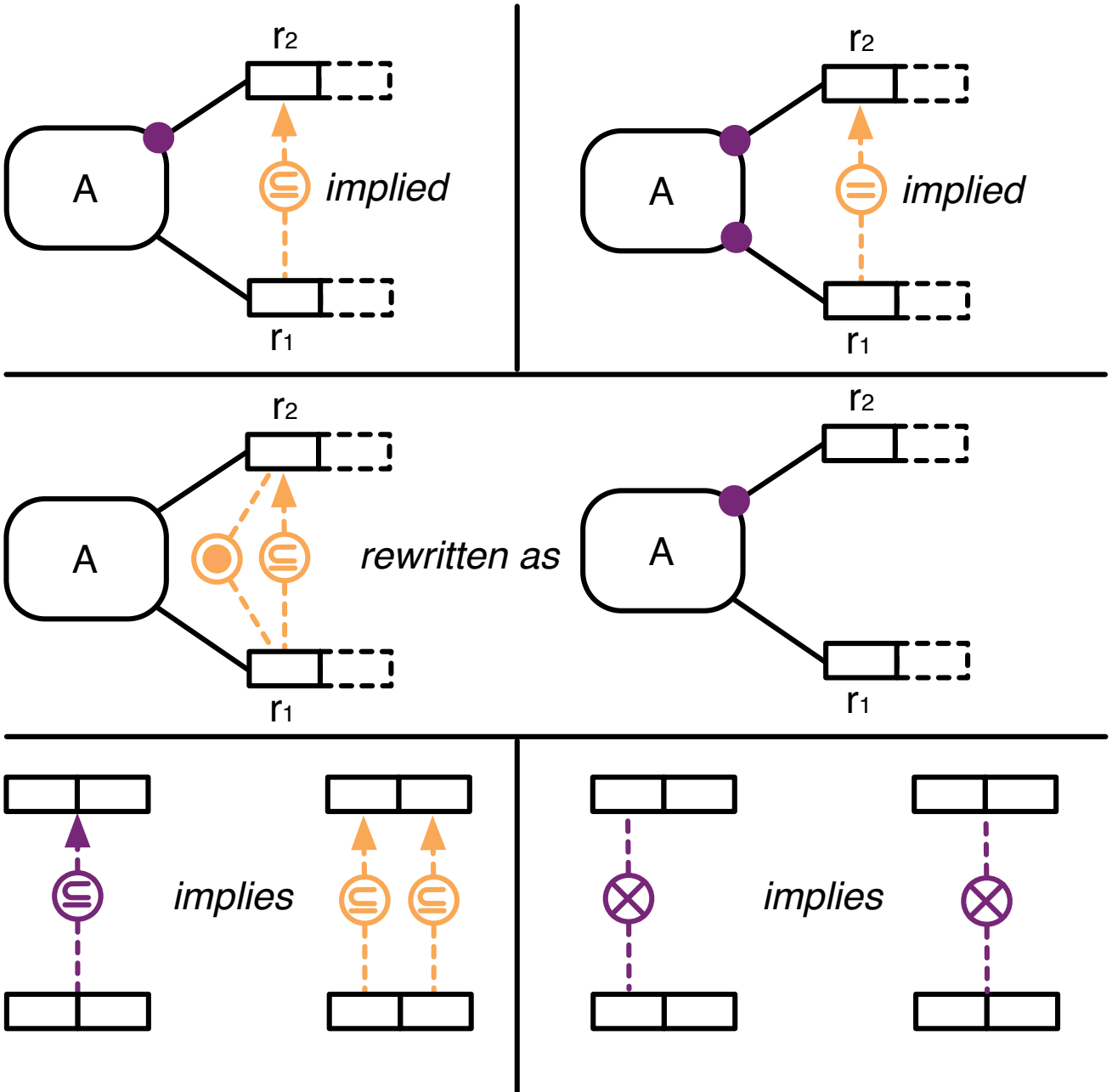




Redundancies, Implied Constraints, Inconsistencies

- Set constraints can interact with each other and with mandatory constraints.
- It could be consequently the case that they are implied, or that they could reveal other constraints.
- To avoid clutter and redundancy, implied constraints are omitted.
- It is also possible to combine constraints that lead to satisfiability only when certain roles are empty
  $\rightarrow$ such roles are not strongly satisfiable.
  - Consider the interaction between a subset and an exclusion constraint.
- This latter situation must be avoided.

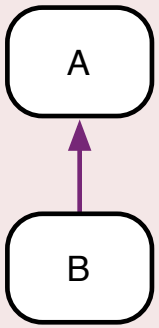
Typical Cases



Subtyping

Classification of instances of an object type into a more specific type.

Proper subtype



A $\neq$ B and for each information base state:
$pop(B) \subseteq pop(A)$.

Why subtyping?

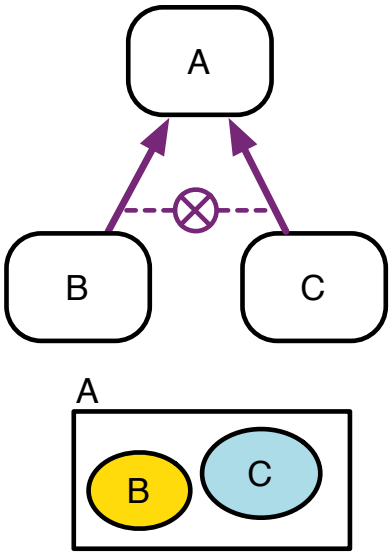
- To declare that one or more specific roles are played only by a given subtype.
- To encourage reuse of model components (extending what has been modeled so far).
- To reveal taxonomies (only if they are functional to the strategic goals, i.e., all entity types play specific roles).

Subtyping and Set-Constraints

Terminology:

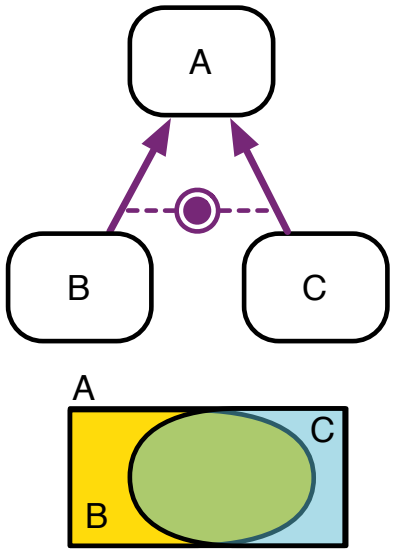
- A is a supertype, B and C are subtypes.
- Only one supertype $\rightarrow$ single inheritance.
- Multiple supertypes $\rightarrow$ multiple inheritance.
- If D subtype C, then D is an indirect subtype of A, because subtypehood relation is *transitive*.

Exclusive subtypes



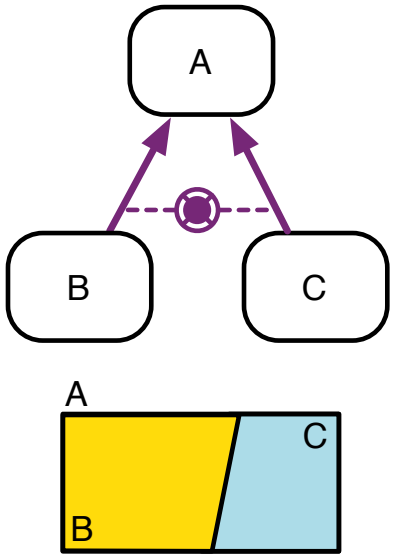
`No A is both B and C`

Exhaustive subtypes (covering)



`Each A is some B or C`

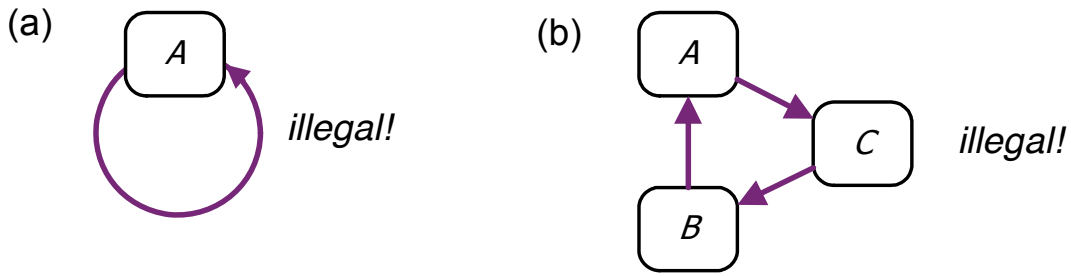
Partition



`Each A is exactly one B or C`

Subtype Graph and Acyclicity Condition

- Due to many-to-many subtype relationships, many *subtype graphs* are constructed.
- Each subtype graph has only one root constituted by a primitive entity type.
- Remember: primitive entity type are by definition mutually exclusive.
- A graph-construction step arises because of:
  - Specialization of an object type into subtypes (to add details).
  - Generalization of object types into a common supertype, retaining the subtypes for details.
- No type can be (proper) subtype of itself, hence the graph must be *acyclic*, i.e., a DAG.



Subtypes and Roles

Differently from primitive object types, no assumption is done for a subtype about mandatory roles.

Any mandatory (simple/disjunctive) constraints must be explicitly shown for a subtype.

- Each subtype inherits all roles of its supertypes, and typically extends them (if not, then it is *inactive*, just used for taxonomic purposes).

We must be able to determine membership of derived subtypes.

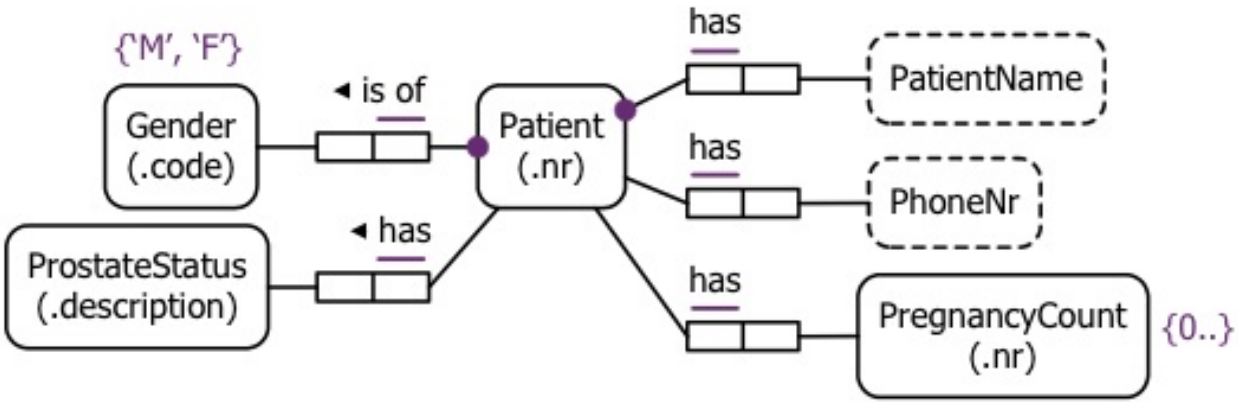
Subtype definition

Each derived subtype must be formally defined in terms of at least one role played by its supertype(s).

- Typical case: qualify optional role or set-comparison constraint.
  - `Each MaleEmployee is an Employee who is of Gender 'M'.`
- Another typical case is to substitute unqualified, set-comparison constraints.

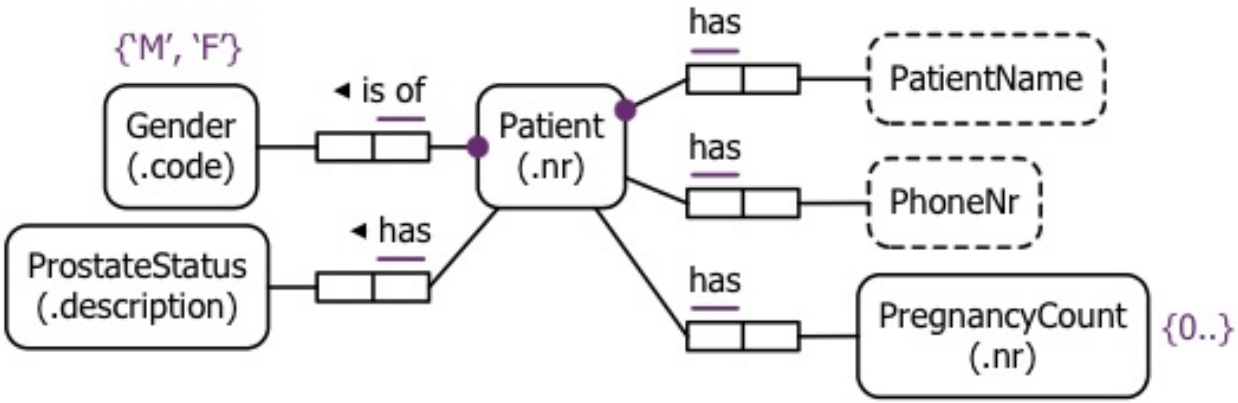
Subtypes and Derivation Rules

Incomplete diagram: some exams can only be associated to female/male Patients (optional roles are in fact used).

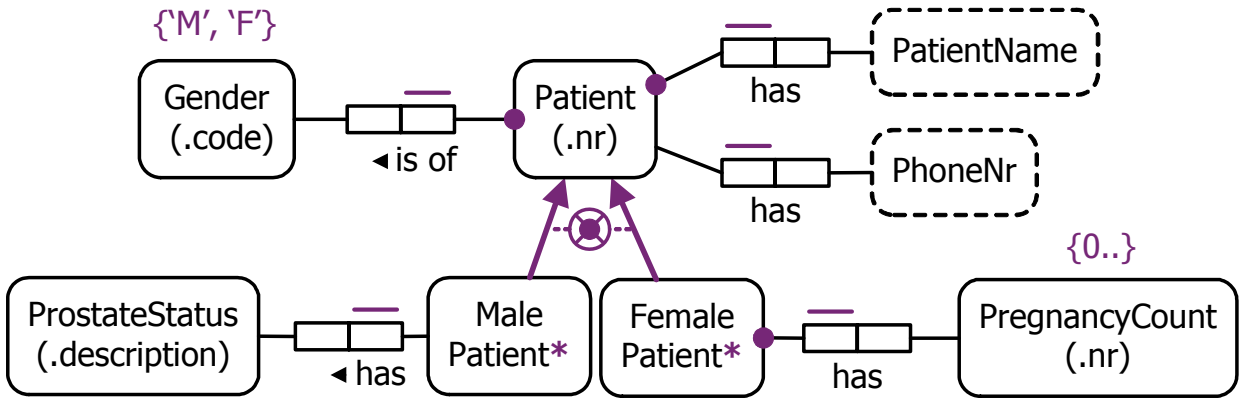


Subtypes and Derivation Rules

Incomplete diagram: some exams can only be associated to female/male Patients (optional roles are in fact used).



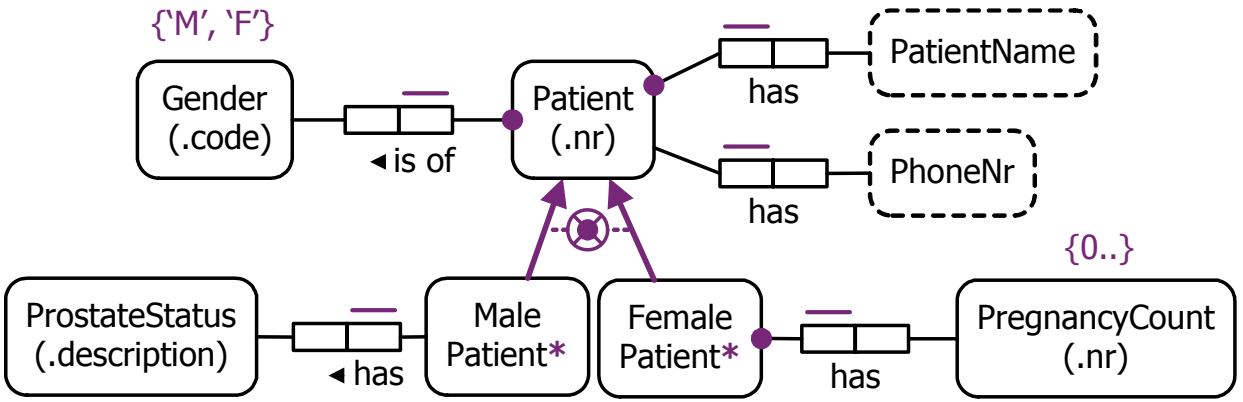
Solution: introduce subtyping, defining each subtype in terms of the Gender object type.



***Each** MalePatient **is a** Patient **who** is of Gender 'M'.

***Each** FemalePatient **is a** Patient **who** is of Gender 'F'.

Subtypes Constraints and Redundancy



***Each** MalePatient **is a** Patient **who** is of Gender 'M'.
***Each** FemalePatient **is a** Patient **who** is of Gender 'F'.

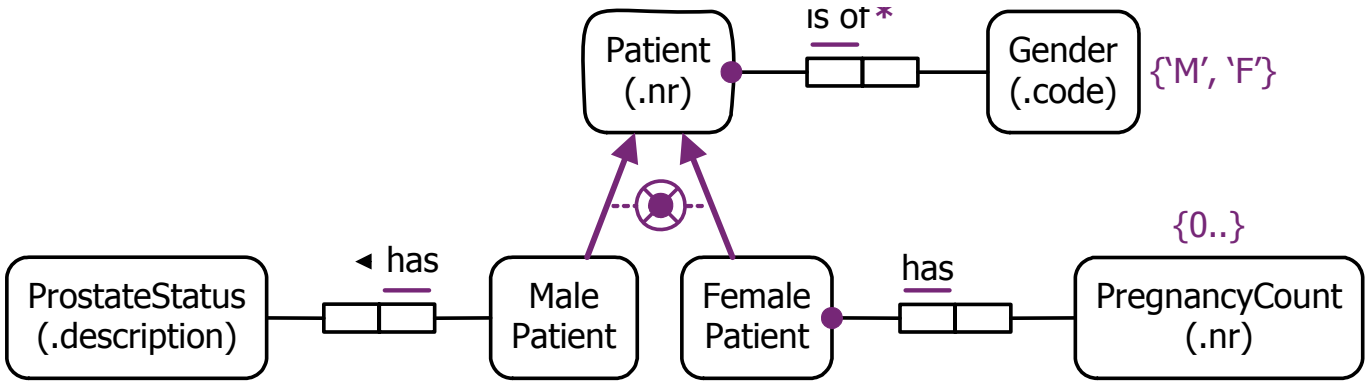
- Any exclusion/covering constraint in subtypes is always implied in a well-designed diagram if a definition for that subtypes is provided.
- In the example, definitions + value constraint on Gender + mandatory role on Patient for its relationship with Gender $\rightarrow$ exhaustive subtypes.
- The graphical symbol is maintained for clarity.

Asserted Subtype

- Subtype without an explicit definition.
- Corresponds to a unary predicate on the supertype.
- All exclusion/covering constraints must be obviously explicitly listed.
- Derivations can in this case be found using the taxonomy.

Asserted Subtype

- Subtype without an explicit definition.
- Corresponds to a unary predicate on the supertype.
- All exclusion/covering constraints must be obviously explicitly listed.
- Derivations can in this case be found using the taxonomy.



*Patient is of Gender **iff**
Patient **is a** MalePatient **and** Gender = 'M'
or Patient **is a** FemalePatient **and** Gender = 'F'.

- This is more similar to OO programming...

If a taxonomy is captured both via subtyping and classifying fact type, we need a definition to connect the two (we can chose where to put it).

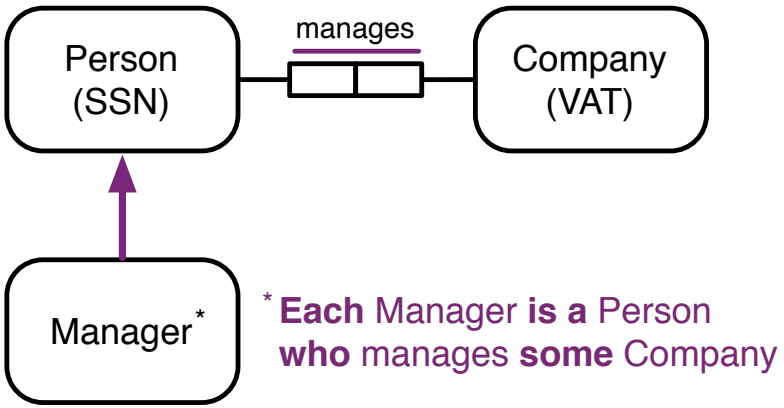
Semiderived Subtype

- Typically used to denote incomplete knowledge.
- Incomplete knowledge is related to the fact type used in the derivation rule that characterizes the subtype.

Semiderived Subtype

- Typically used to denote incomplete knowledge.
- Incomplete knowledge is related to the fact type used in the derivation rule that characterizes the subtype.

Complete derivation of Manager from "manages".

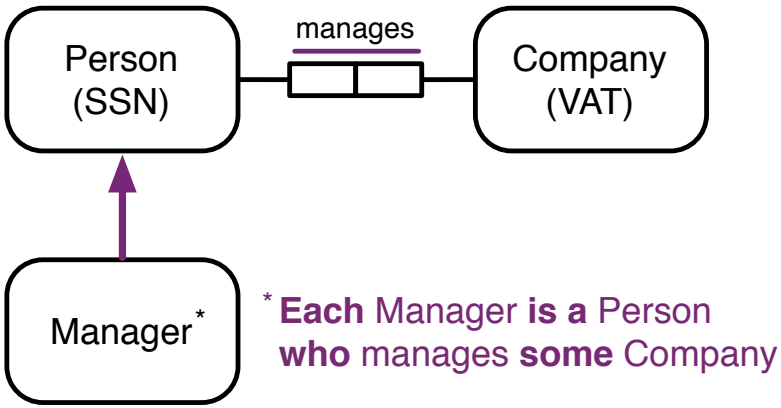


A Manager can only be added through the "manages" fact type, specifying the managed Company.

Semiderived Subtype

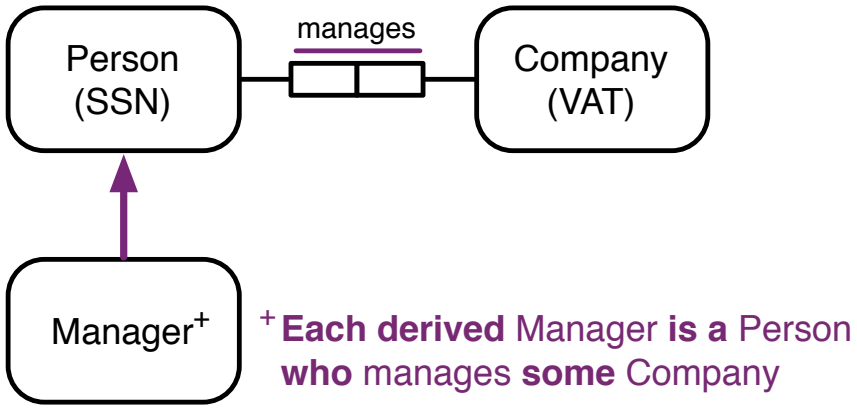
- Typically used to denote incomplete knowledge.
- Incomplete knowledge is related to the fact type used in the derivation rule that characterizes the subtype.

Complete derivation of Manager from "manages".



A Manager can only be added through the "manages" fact type, specifying the managed Company.

Semiderivation due to incomplete knowledge about "manages".



A Manager can also be added directly, without specifying the managed Company.

Subtyping and Identification Schemes

How do we determine the preferred identification scheme of a subtype?

- Inherited from the corresponding supertype: solid arrow.
- Inherited from another supertype or autonomously determined: dashed arrow.

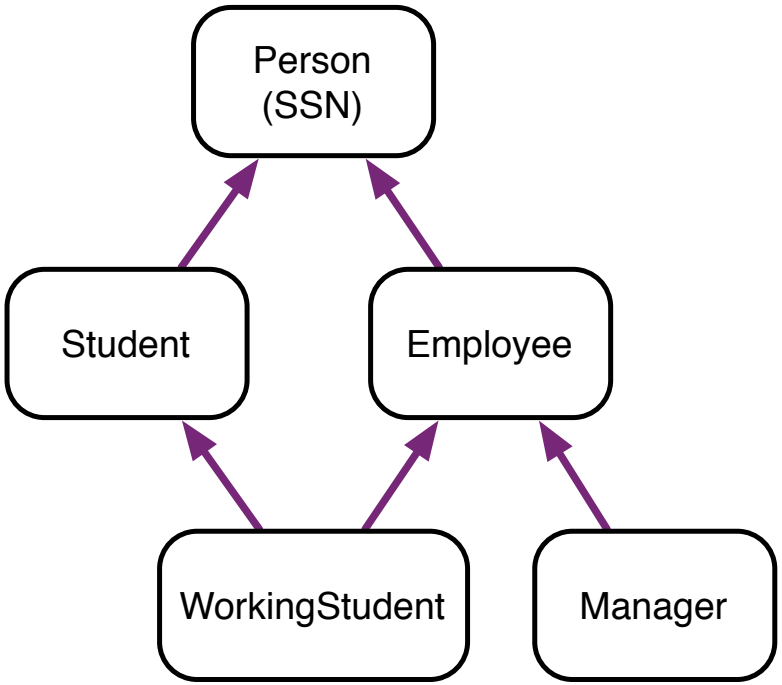
The second case reflects a *context-dependent reference scheme*.

Subtyping and Identification Schemes

How do we determine the preferred identification scheme of a subtype?

- Inherited from the corresponding supertype: solid arrow.
- Inherited from another supertype or autonomously determined: dashed arrow.

The second case reflects a *context-dependent reference scheme*.



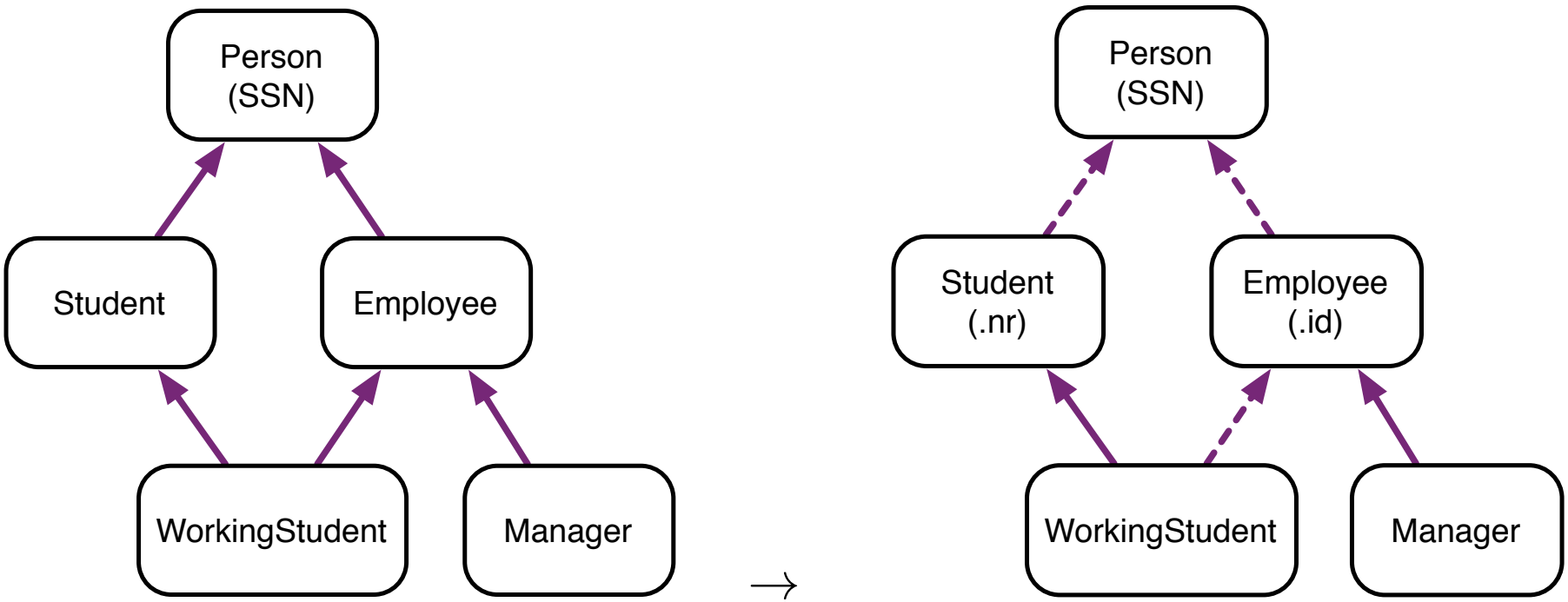
$\rightarrow$

Subtyping and Identification Schemes

How do we determine the preferred identification scheme of a subtype?

- Inherited from the corresponding supertype: solid arrow.
- Inherited from another supertype or autonomously determined: dashed arrow.

The second case reflects a *context-dependent reference scheme*.



Specialization Procedure

Given an object type. . .

1. Specify all mandatory role constraints.
2. For each optional role: **if**
   - it is recorded only for a known subtype **and**
   - there is a subtype definition stronger than a set-comparison constraint **or** another role is recorded only for that subtype

   then
   - 2.1 Introduce the subtype.
   - 2.2 Attach its specific roles.
   - 2.3 Declare the subtype derived (*), asserted, or semiderived ($^+$).
   - 2.4 If * or $^+$, add a derivation rule.
   - 2.5 Goto step (1) considering the subtype.

Spotting Subtypes

FormNr: 5001

1. Age (years): .....

2. Nr hours spent per week watching TV: .....

   If you answered 0 then go to Question 4

3. What is your favorite TV channel? .....

---

4. Nr hours spent per week reading newspapers: .....

   If you answered 0 then Stop (no more answers are required).

5. What is your favorite newspaper? .....

   If you are younger than 18, or answered 0 to question 2 or 4 then Stop (no more answers are required)

---

6. Which do you prefer as a news source?
   (Check the box of your choice)

   ❑ Television

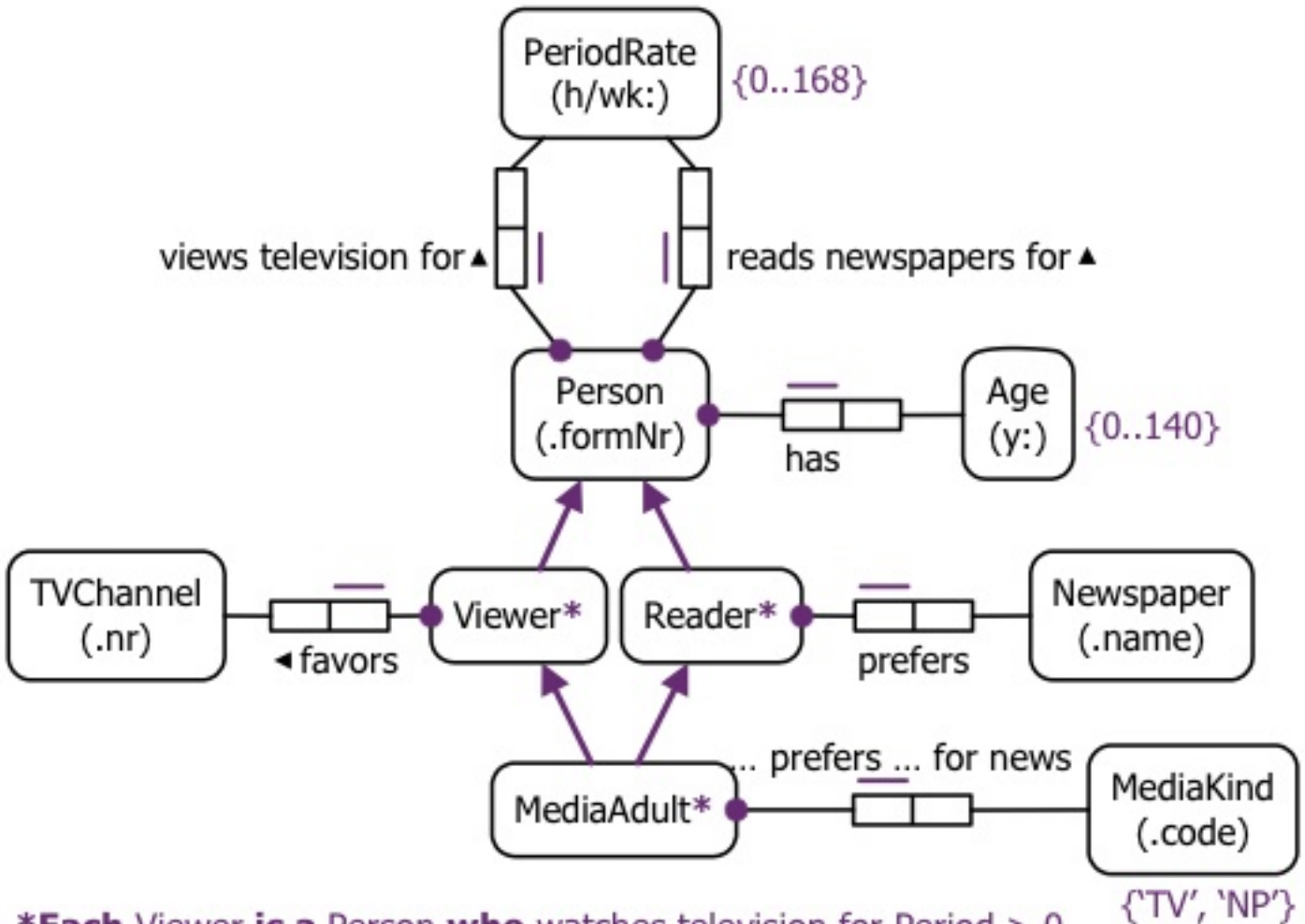
   ❑ Newspaper

Spotting Subtypes

Person	*Age (y)*	*Television (h/week)*	*Newspaper (h/week)*	*Favorite channel*	*Favorite paper*	*Preferred news*
5001	41	0	10	–	The Times	–
5002	60	0	25	–	The Times	–
5003	16	20	2	9	The Times	–
5004	18	20	5	2	Daily Mail	TV
5005	13	35	0	7	–	–
5006	17	14	4	9	Daily Sun	–
5007	50	8	10	2	Daily Sun	NP
5008	33	0	0	–	–	–
5009	13	50	0	10	–	–

- There are different kinds of missing values
  - ? denotes unknown.
  - – denotes not applicable.
- Not applicable fields suggest here that we have different subtypes associated to different information sources.

Spotting Subtypes



Generalization

Introduction of a supertype that factorize common roles and features of several object types.

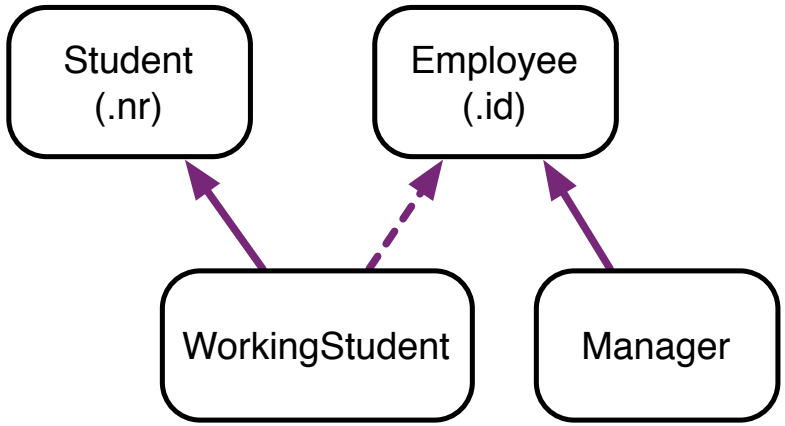
- Having common roles to be factorized is not sufficient to motivate the introduction of a supertype.
- Connection with Step 3: combination of different object types into a unique object type, but now we can also retain the subtypes.

Generalization

Introduction of a supertype that factorize common roles and features of several object types.

- Having common roles to be factorized is not sufficient to motivate the introduction of a supertype.
- Connection with Step 3: combination of different object types into a unique object type, but now we can also retain the subtypes.

How to query all the persons we have?

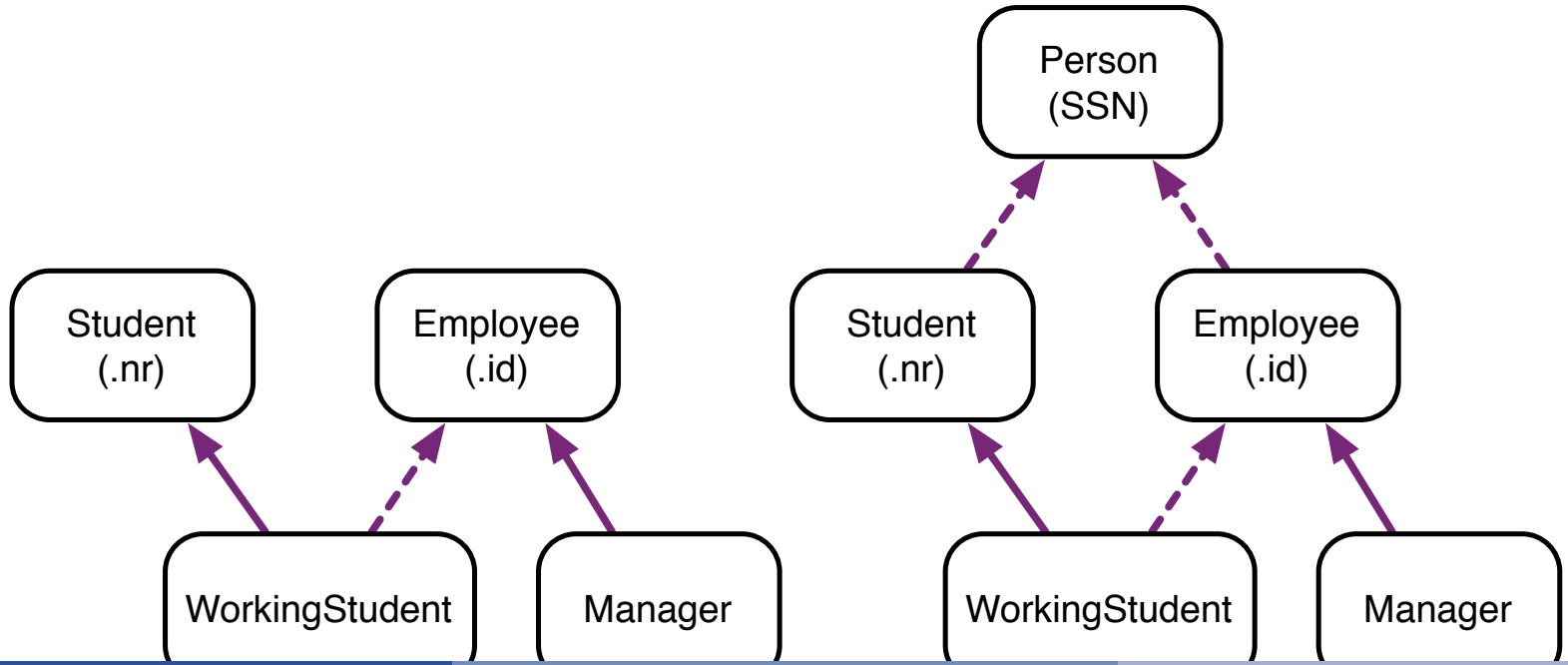


Generalization

Introduction of a supertype that factorize common roles and features of several object types.

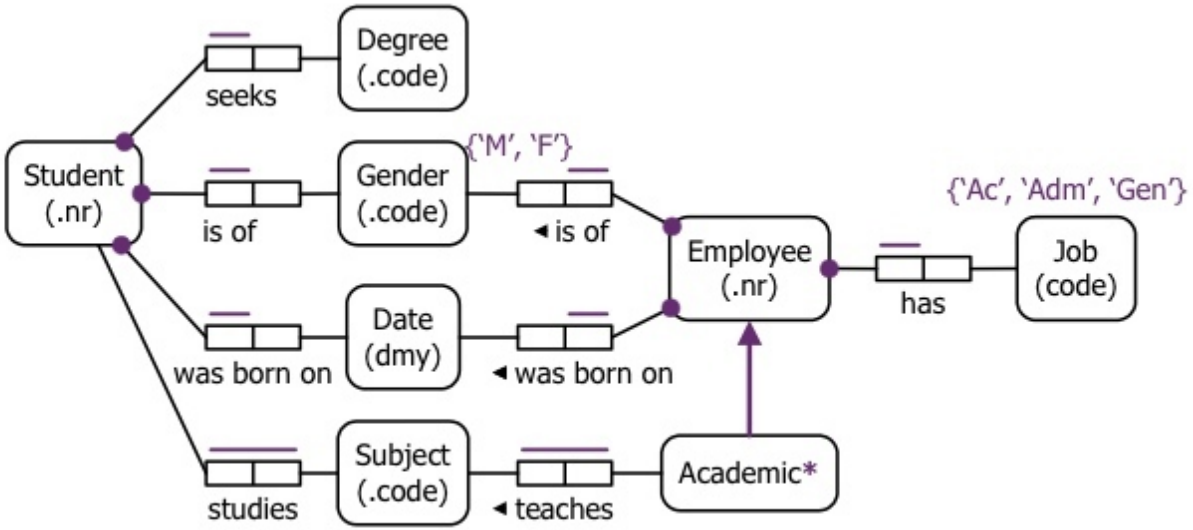
- Having common roles to be factorized is not sufficient to motivate the introduction of a supertype.
- Connection with Step 3: combination of different object types into a unique object type, but now we can also retain the subtypes.

How to query all the persons we have?



Generalization Guidelines

- For exclusive object types, generalization is needed only when common details for them must be listed in the same query.
- Remember that, by default, top-level (primitive) object types are mutually exclusive.
- Therefore, if we want primitive object types to overlap, we must introduce a common supertype for them.
- As usual, the model could differ from reality. This must be motivated and documented.



***Each** Academic **is an** Employee **who** has Job 'Ac'.

Generalization Procedure

Given two completely separated object types A and B...
If

- A and B overlap, or can be compared
  and we want to model this
- **or** A and B are mutually exclusive **and** common information is listed for both
  and we want to list A and B together for this information

then

1. Introduce their supertype A $\cup$ B with its own identification scheme.
2. Add classification predicates on the supertype, to identify A and B.
3. Attach common roles to the supertype.
4. **If** A (or B) plays specific roles
   then define A (B) as a subtype and attach such roles.