Description Logics

Designing Knowledge Bases

Enrico Franconi

`franconi@cs.man.ac.uk`

`http://www.cs.man.ac.uk/~franconi`

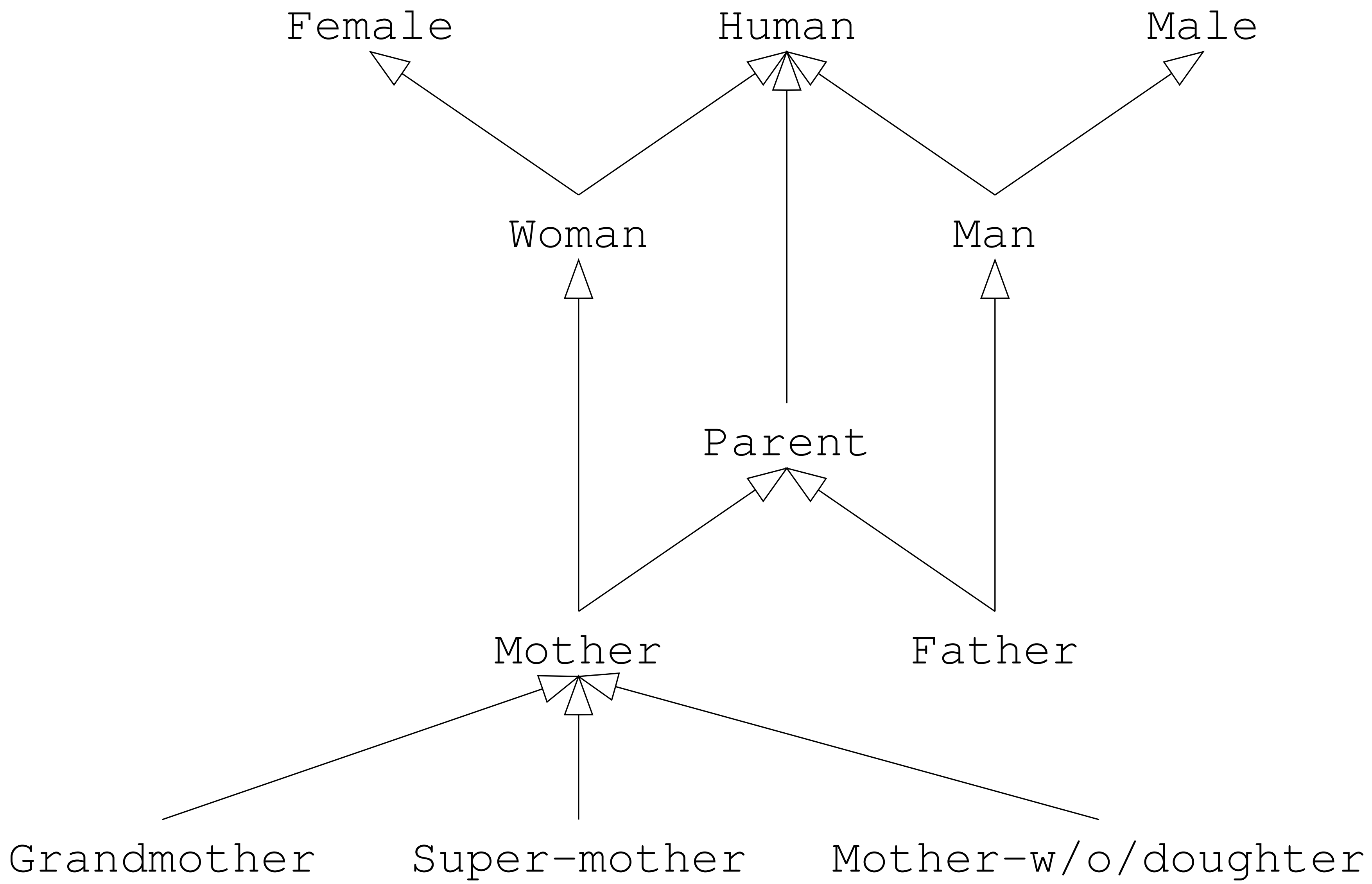
Department of Computer Science, University of Manchester

The Royal Family

$$
\begin{aligned}
\mathtt{Male} &\doteq \neg\mathtt{Female}\\
\mathtt{Woman} &\doteq \mathtt{Human} \sqcap \mathtt{Female}\\
\mathtt{Man} &\doteq \mathtt{Human} \sqcap \mathtt{Male}\\
\mathtt{Mother} &\doteq \mathtt{Woman} \sqcap \exists \mathtt{CHILD.Human}\\
\mathtt{Father} &\doteq \mathtt{Man} \sqcap \exists \mathtt{CHILD.Human}\\
\mathtt{Parent} &\doteq \mathtt{Father} \sqcup \mathtt{Mother}\\
\mathtt{Grandmother} &\doteq \mathtt{Woman} \sqcap \exists \mathtt{CHILD.Parent}\\
\mathtt{Mother{-}w/o{-}doughter} &\doteq \mathtt{Mother} \sqcap \forall \mathtt{CHILD.Male}\\
\mathtt{Super{-}mother} &\doteq \mathtt{Mother} \sqcap \geqslant 3\ \mathtt{CHILD}
\end{aligned}
$$

```
Woman(elisabeth),                     CHILD(elisabeth,charles),
Woman(diana),                         CHILD(elisabeth,edward),
Man(charles), Man(edward),            CHILD(elisabeth,andrew),
Man(andrew),                          CHILD(diana,william),
Mother-w/o-doughter(diana),           CHILD(charles,william)
```

Taxonomy


Female
Human
Male
Woman
Man
Parent
Mother
Father
Grandmother
Super-mother
Mother-w/o/doughter

Questions

- What happens if we add to the knowledge base:

  `CHILD(diana,margaret), Female(margaret)`

- $\Sigma \overset{?}{\models}$ `Super-mother(elisabeth)`
- $\Sigma \overset{?}{\models} \neg$`Female(william)`
- $\Sigma \overset{?}{\models}$ `Mother-w/o-doughter(elisabeth)`

- Which are the most specific concepts of which `elisabeth` is instance (realization problem)?

- Retrieve all the instances of `Male`.

Inclusion Axioms

$$\begin{aligned}
\exists\mathtt{TEACHES}.\mathtt{Course} &\sqsubseteq (\mathtt{Student} \sqcap \exists\mathtt{DEGREE}.\mathtt{Bs}) \sqcup \mathtt{Prof} \\
\mathtt{Prof} &\sqsubseteq \exists\mathtt{DEGREE}.\mathtt{Ms} \\
\exists\mathtt{DEGREE}.\mathtt{Ms} &\sqsubseteq \exists\mathtt{DEGREE}.\mathtt{Bs} \\
\mathtt{Ms} \sqcap \mathtt{Bs} &\sqsubseteq \bot
\end{aligned}$$

TEACHES(john,cs156),
$(\leqslant 1\ \mathtt{DEGREE})$(john),
Course(cs156)

$\Sigma \overset{?}{\models}$ Student(john)

Cardinality and sub-roles

$$
\begin{array}{rcl}
\texttt{Man} & \sqsubseteq & \texttt{Human} \\
\texttt{Woman} & \sqsubseteq & \texttt{Human} \sqcap \neg\texttt{Man} \\
\texttt{Team} & \doteq & \texttt{Set} \sqcap \forall\texttt{MEMBER.Human} \sqcap \geqslant 2\ \texttt{MEMBER} \\
\texttt{LEADER} & \sqsubseteq & \texttt{MEMBER} \\
\texttt{Modern-team} & \doteq & \texttt{Team} \sqcap \leqslant 4\ \texttt{MEMBER} \sqcap \\
 & & \geqslant 1\ \texttt{LEADER} \sqcap \forall\texttt{LEADER.Woman}
\end{array}
$$

```
Modern-team(gamma),
Man(tom), Man(dick), Human(mary),
MEMBER(gamma, tom),
MEMBER(gamma, dick),
MEMBER(gamma, mary),
```
$(\leqslant 3\ \texttt{MEMBER})\ \texttt{(gamma)}$

$\Sigma \overset{?}{\models}$ `Woman(mary)`

Cardinality and sub-roles - II

$$\begin{aligned} \mathtt{DAUGHTER} &\sqsubseteq \mathtt{CHILD} \\ \mathtt{SON} &\sqsubseteq \mathtt{CHILD} \\ \mathtt{Female} &\sqsubseteq \neg\mathtt{Male} \end{aligned}$$

$$\geqslant 2\ \mathtt{SON} \sqcap \geqslant 2\ \mathtt{DOUGHTER} \sqcap$$
$$\forall\mathtt{SON}.\mathtt{Male} \sqcap \forall\mathtt{DAUGHTER}.\mathtt{Female}$$

$$\overset{?}{\sqsubseteq}$$

$$\geqslant 4\ \mathtt{CHILD}$$

Cardinality, sub-roles, and functional roles

$$\mathtt{DAUGHTER} \sqsubseteq \mathtt{CHILD}$$

$$\Sigma \overset{?}{\models} \exists\mathtt{CHILD.Rich} \sqsubseteq \exists\mathtt{DAUGHTER.Rich}$$

$$\Sigma \overset{?}{\models} \exists\mathtt{DAUGHTER.Rich} \sqsubseteq \exists\mathtt{CHILD.Rich}$$

$$\Sigma \overset{?}{\models} \forall\mathtt{CHILD.Rich} \sqsubseteq \forall\mathtt{DAUGHTER.Rich}$$

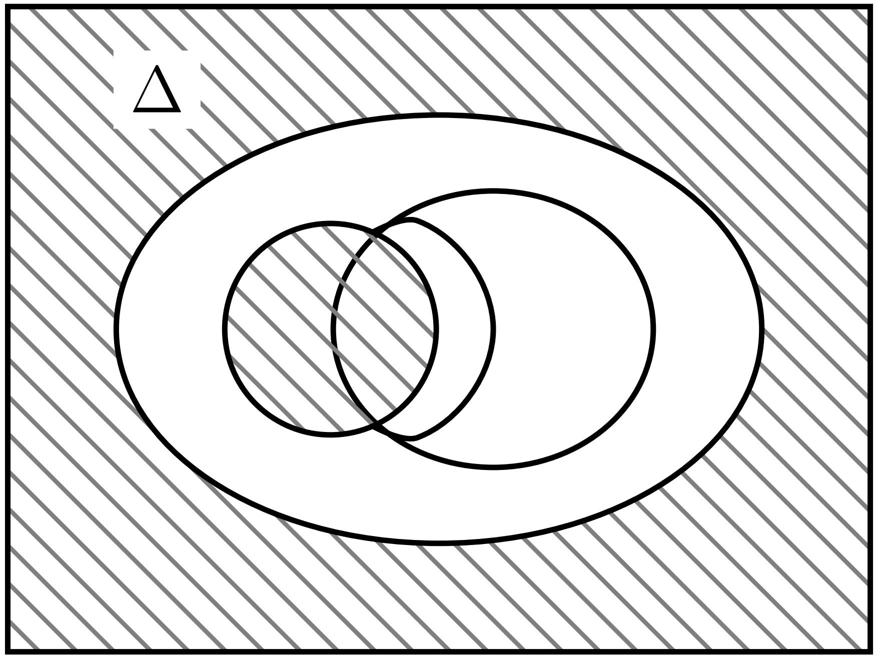
$$\Sigma \overset{?}{\models} \forall\mathtt{DAUGHTER.Rich} \sqsubseteq \forall\mathtt{CHILD.Rich}$$

$$\Sigma \overset{?}{\models} \forall\mathtt{CHILD.Rich} \sqcap \exists\mathtt{CHILD} \sqsubseteq \forall\mathtt{DAUGHTER.Rich} \sqcap \exists\mathtt{DAUGHTER}$$

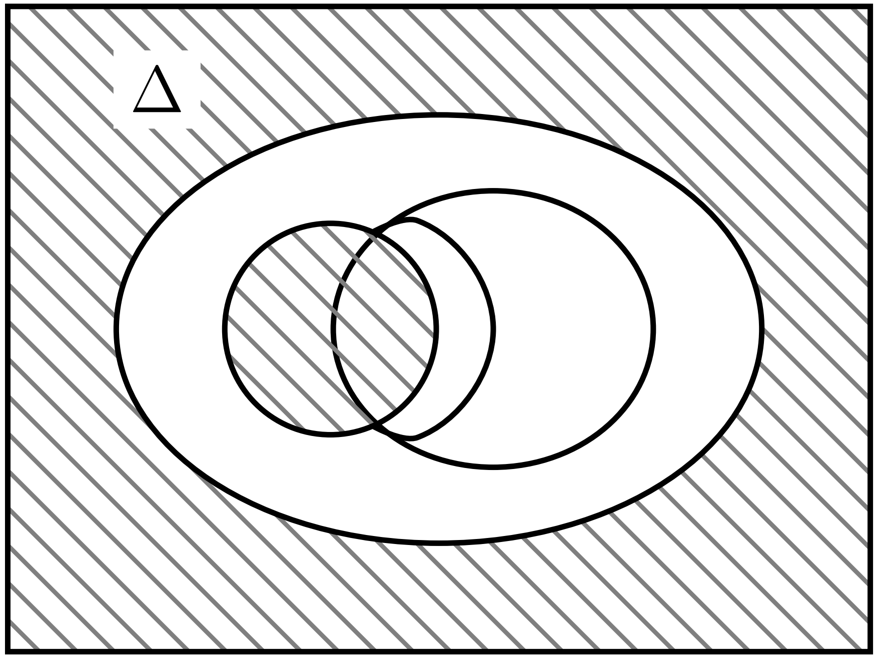
$$\Sigma \overset{?}{\models} \forall\mathtt{DAUGHTER.Rich} \sqcap \exists\mathtt{DAUGHTER} \sqsubseteq \forall\mathtt{CHILD.Rich} \sqcap \exists\mathtt{CHILD}$$

What if the roles are functional?

$$\Sigma \models \forall \mathtt{CHILD.Rich} \sqsubseteq \forall \mathtt{DAUGHTER.Rich}$$



$\forall\mathtt{CHILD.Rich} \sqcap \exists\mathtt{CHILD} \not\sqsubseteq \forall\mathtt{DAUGHTER.Rich} \sqcap \exists\mathtt{DAUGHTER}$



Tests

- In order to allow *procedures* to be used in specifying concepts, the $\mathtt{TEST{-}C}$ operator is used.
- A test restriction requires that an individual must pass the test to satisfy the restriction.

Example:

$$\mathtt{Even{-}integer} \doteq \mathtt{Integer} \sqcap (\text{:TEST-C } \mathtt{evenp})$$

where EVENP is a function in the host language.

The reasoning procedures are changed in a way that individuals *passing* some test function “f” will be in the extension of the concept $(\mathtt{:TEST{-}C\ \ f})$. The individual to be tested is passed as an argument to the function.

Test functions

Test functions return one of three values when applied to an individual:

- FALSE: the individual is inconsistent with the restriction.
- UNKNOWN: the individual is consistent with the restriction, but if information is added to the individual, the individual may become either inconsistent with or provably described by the restriction.
- TRUE: the individual definitely passes the test, i.e., it provably satisfies it.

Test functions must be monotonic; that is, it should not be possible for the same test function to return TRUE (or FALSE) for an individual at one time, and FALSE (or TRUE) at a later time, when the information regarding the individual has been refined.

Forward-chaining Rules

- A rule consists of an antecedent and a consequent.
- An antecedent is always a concept name.
- A consequent is a generic concept description.

As soon as an individual is known to be in the extension of the antecedent concept, the rule is triggered, and the individual is also known to satisfy the consequent concept.

Forward-chaining Rules

- Intuitively, a rule $A \Longrightarrow C$ seems to have the same semantics as $A \sqsubseteq C$, in the sense that every individual in A should be also in C.
- However, a careful formal analysis shows that a rule is an autoepistemic statement, with a nonmonotonic behaviour.
- Rules play a role only for individuals, and not for concept definitions.
- They are mostly used for expressing contingent properties.

Integrity Checking

- The aim of integrity checking is to support the activity of populating the KB with additional checks on the structures of the individuals w.r.t. to the concepts they are instances of.
- This can not be obtained by adding extra restrictions (tests) to the definition of the concepts, because this would modify it (with problems for the classification).
- The solution: Rule + Test.

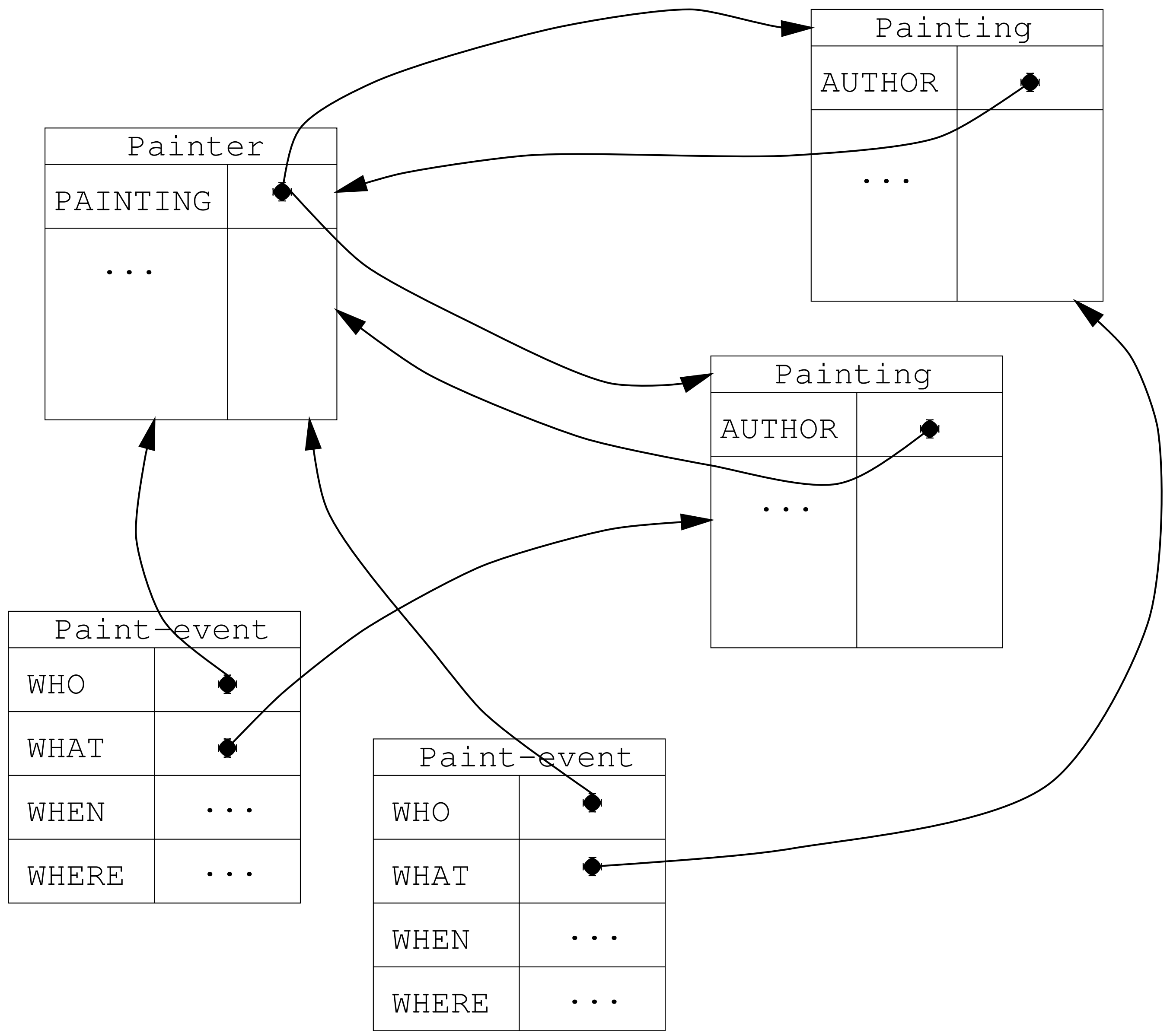
Example:

$\texttt{Late}-\texttt{harvest}-\texttt{grape} \Longrightarrow \big(\text{:TEST-C } \texttt{sugar}>30\big)$

Where the test function SUGAR¿30 provides (if it is the case) to remove the candidate instance from the `Late-harvest-grape` concept, too.

Rule + Test can also emulate *methods*.

Modelling a Museum



Concept as Role

- The painting *The Announcement* of Giotto's is in Florence.
- The painting *The Announcement*, painted by Giotto in 1285 in Venice, is in Florence.

$$\texttt{Painter} \sqsubseteq \forall \texttt{PAINTING}.\texttt{Painting}$$

$$\texttt{Painting} \sqsubseteq \forall \texttt{AUTHOR}.\texttt{Painter}$$

$$\texttt{PaintEvent} \sqsubseteq \exists \texttt{WHO}.\texttt{Painter} \sqcap \exists \texttt{WHAT}.\texttt{Painting}$$

- This TBox forces redundancy.
- This TBox does not reveal inconsistencies.
- This TBox is cyclic.

Redundant KB

$\mathtt{Painter} \sqsubseteq \forall \mathtt{PAINTING.Painting}$

$\mathtt{Painting} \sqsubseteq \forall \mathtt{AUTHOR.Painter}$

$\mathtt{PaintEvent} \sqsubseteq \exists \mathtt{WHO.Painter} \sqcap \exists \mathtt{WHAT.Painting}$

```
Painter(giotto),

PAINTING(giotto, announcement),   PaintEvent(e1),
Painting(announcement),           WHO(e1, giotto),
AUTHOR(announcement, giotto),     WHAT(e1, announcement),

PAINTING(giotto, escape),         PaintEvent(e2),
Painting(escape),                 WHO(e2, giotto),
AUTHOR(escape, giotto),           WHAT(e2, escape)
```

Reification

$$\forall xy.\, \mathtt{PAINTING}(x, y) \leftrightarrow \mathtt{AUTHOR}(y, x) \leftrightarrow$$
$$\exists z.\, \mathtt{PaintEvent}(z) \wedge \mathtt{WHO}(z, x) \wedge \mathtt{WHAT}(z, y)$$

$$\mathtt{PaintEvent} \sqsubseteq \exists \mathtt{WHO}. \sqcap \leqslant 1\, \mathtt{WHO} \sqcap \exists \mathtt{WHAT}. \sqcap \leqslant 1\, \mathtt{WHAT}$$

$$\mathtt{Painter} \doteq \exists \mathtt{WHO}^{-}.\mathtt{PaintEvent}$$

$$\mathtt{Painting} \doteq \exists \mathtt{WHAT}^{-}.\mathtt{PaintEvent}$$

$$\mathtt{PAINTING} \doteq \mathtt{WHO}^{-}\,|_{\mathtt{PaintEvent}} \circ \mathtt{WHAT}$$

$$\mathtt{AUTHOR} \doteq \mathtt{WHAT}^{-}\,|_{\mathtt{PaintEvent}} \circ \mathtt{WHO}$$

```
PaintEvent(e1), WHO(e1, giotto), WHAT(e1, announcement)

    Painter(giotto),
    PAINTING(giotto, announcement),
⊨   Painting(announcement),
    AUTHOR(announcement, giotto)
```

```
PAINTING(giotto, announcement)

     Painter(giotto),
⊨    Painting(announcement),
     AUTHOR(announcement, giotto)
```

Building Knowledge Bases

In order to build good KBs some choices must be done during its design. It is important to well understand some subtle distinctions:

- Primitive vs. Defined.
- Definitional vs. Incidental.
- Concept vs. Individual.
- Concept vs. Role.

When to Use Primitive Concepts?

- some concepts can not be completely defined (e.g. natural kinds);
- it can be not convenient/useful to completely define a concept;
- sooner or later we must end up with something not completely defined (*encyclopedic knowledge* cannot be given).

Thus, primitive concepts must be used when:

- there is no other way;
- even if it were defined, no (automatic) classification below it will be never required by the application.

Typically primitive concepts lie in the top region of the taxonomy.

When to Use Defined Concepts?

- ontological reason: it is easy and natural (in the context of the application) to give a complete definition of the concept;
- organisation of the antecedents of rules;
- capturing complete descriptions used by rules for populating primitive concepts.

Definitional vs. Incidental

Are *incidental* all the properties that are contingent features for a concept, and thus must not be part of its definition.

Examples:

$(\forall \mathtt{SUGAR}.\mathtt{Dry})$

is incidental for the concept RED−BORDEAUX−WINE, while not

$(\forall \mathtt{COLOR}.\mathtt{Red})$ and $(\forall \mathtt{REGION}.\mathtt{Bordeaux})$

$(\forall \mathtt{INTELLIGENCE}.\mathtt{Stupid})$

is incidental for CHICKEN, while not

$(\forall \mathtt{REPRODUCES{-}WITH}.\mathtt{Egg})$

Concept vs. Individual

- the set of individuals is a countable, discrete set;
- the concept space is ideally continuous and infinite;
- each individual has a clear identity: even if two individual have the same properties, they are distinct;
- if two concepts have equivalent descriptions, they **denote the same** concept;
- individual descriptions can be modified;
- concept descriptions can not be modified;
- individual update does not (usually) change the concept hierarchy;
- rules applies only to individuals.

Concept vs. Individual, cont.

Nevertheless, it is not always easy to decide whether an object should be a concept or an individual. The main issue to deal with is the “granularity” level.

Example:

Consider the KB describing courses in a Computer Science department : is “Introduction To Data Structures And Algorithms (503)” course a concept or an individual?

Individual vs. Concept

Another example: if we have `Wine` and `White-wine`, what about:

- chardonay-wine
- forman-chardonay
- 1981-forman-chardonay
- 1981-forman-chardonay-from-vineyard32
- 1981-forman-chardonay-from-in-cask18
- 1981-forman-chardonay-bottle#1576

A key to solve the problem could be asking the domain/application expert: “how many wines do you have?”, in order to understand the needed granularity.

Concept vs. Role

Is not always easy to decide what must be a concept and what a role.

E.g.:

- PERSON: it is a concept.
- MOTHER:
  - consider “Sue is a new mother” and “Sue is the mother of Tom”
  - `Mother` as a concept does not exist if we don’t consider the “role she plays” in a parental relation, i.e., if “Sue is a new mother” she must be the `MOTHER` of somebody!

Thus:

$$\mathtt{Mother} \doteq (\mathtt{Woman} \sqcap (\exists \mathtt{MOTHER}.\mathtt{Person}))$$

Concept vs. Role

- But when the role is not the only important component of the definition, this dual use is not so neat (consider `VINTAGE`, `GRAPE`).
- Another problem is the *reading direction*: in the above example the role could be, `MOTHER` or `CHILD`.
- A clear convention must be stated, possibly creating long, non ambiguous names for roles, as, e.g.: `HAS-CHILD, IS-THE-PARENT-OF, HAS-VINTAGE, HAS-GRAPE`.
- As an alternative, the adoption of long names for concepts can be also suggested: e.g., `WINE-GRAPE`.

How to design a KB in steps (1-3)

- **Enumerate Objects**. As a bare list of elements of the KB; they will became individuals, concepts, or role.
- **Distinguish Concepts from Roles**. Make a first decision about what object must be considered role; remember that some could have a “natural” concept associated. The remaining objects will be concepts (or maybe individuals). Also, try to distinguish roles from attributes.
- **Develop Concept Taxonomy**. Try to decide a classification of all the concepts, imagining their extensions. This taxonomy will be used as a first reference, and could be revised when definition will be given. It will be used also to check if definition meet our expectations (sometime, interesting, unforeseen (re)classifications are found).

How to design a KB in steps (4-5)

- **Devise partitions**. Try to make explicit all the disjointness and covering constraints among classes, and reclassify the concepts.
- **Individuals**. Try to list as many as possible *generally* useful individuals . Some could have been already listed in step 1. Try to describe them (classify).

How to design a KB in steps (6)

- **Properties and Parts.** Begin to define the internal structure of concepts (this process will continue in the next steps). For each concept list:
  - *intrinsic* properties, that are part of the very nature of the concept;
  - *extrinsic* properties, that are contingent or external properties of the object; they can sometime change during the time;
  - *parts*, in the case of structured or collective objects. They can be physical (e.g., “the components of a car”, “the casks of a winery”, “the students of a class”, “the members of a group”, *“the grape of a wine”*) or abstract (e.g., *“the courses of a meal”*, “the lessons of a course”, *“the topics of a lesson”*).

How to design a KB in steps (6)

- In some cases some relationships between individuals of classes can be considered too accidental to be listed above (e.g., “the employees of a winery”; but the matter could change if we consider `Winery` as a subconcept of `Firm`).
- In general, the above distinctions depend on the level of detail adopted.
- Some of the listed roles will be later considered definitional, and some incidental.
- After this and the next steps check/revision of the taxonomy could be necessary.

How to design a KB in steps (7-12)

- **Cardinality Restrictions.** For the relevant roles for each concept.
- **Value Restriction.** As above. Also, chose the right restriction.
- **Propagate Value Restrictions.** If some value restrictions stated in the previous step does not correspond to already existing concepts, they must be defined.
- **Inter-role Relationship.** Even if hardly definable in DL, they can be useful during the populating and debugging phases.
- **Definitional and Incidental.** It is important distinguish between definitional and incidental properties, w.r.t. to the particular application.
- **Primitive and Defined.** As above.