Description Logics for Conceptual Data Modeling in UML

Diego Calvanese, Giuseppe De Giacomo

Dipartimento di Informatica e Sistemistica

Università di Roma “La Sapienza”

ESSLLI 2003
Vienna, August 18–22, 2003

Part 3

Complexity of Reasoning on UML Class Diagrams

We are now ready to answer our initial questions

1. Can we develop sound, complete, and **terminating** reasoning procedures for reasoning on UML Class Diagrams?

   To answer this question we polynomially encode UML Class Diagrams in DLs

   $\leadsto$ reasoning on UML Class Diagrams can be done in EXPTIME

2. How hard is it to reason on UML Class Diagrams in general?

   To answer this question we polynomially reduce reasoning in EXPTIME-complete DLs to reasoning on UML class diagrams

   $\leadsto$ reasoning on UML Class Diagrams is in fact EXPTIME-hard

We start with point (2)

Reasoning tasks on UML class diagrams

1. Consistency of the whole class diagram
2. Class consistency
3. Class subsumption
4. Class equivalence
5. $\cdots$

Obviously:

- Consistency of the class diagram can be reduced to class consistency
- Class equivalence can be reduced to class subsumption

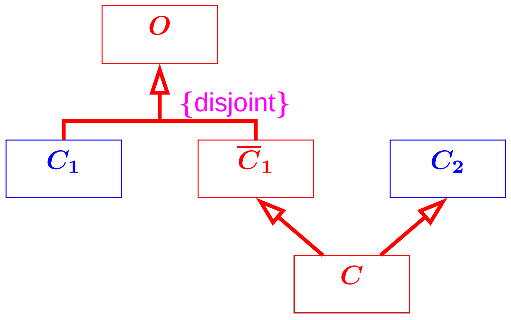
We show that also class consistency and class subsumption are mutually reducible

This allows us to concentrate on class consistency only

Reducing class subsumption to class consistency

To check whether a class C_1 subsumes a class C_2 in a class diagram $\mathcal{D}$:

1. Add to $\mathcal{D}$ the following part, with O, C, and $\overline{C}_1$ new classes

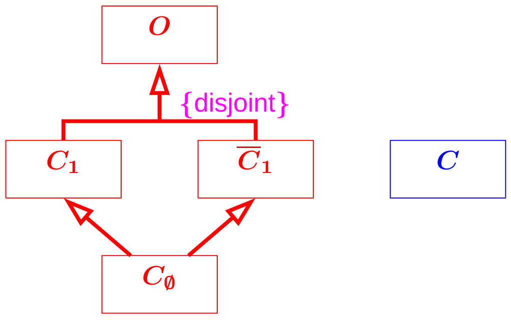


2. Check whether C is inconsistent in the resulting diagram

Reducing class consistency to class subsumption

To check whether a class C is inconsistent in a class diagram $\mathcal{D}$:

1. Add to $\mathcal{D}$ the following part, with O, C_1, $\overline{C}_1$, and $C_\emptyset$ new classes



2. Check whether $C_\emptyset$ subsumes C in the resulting diagram

Lower bound for reasoning on UML class diagrams

EXPTIME lower bound established by encoding satisfiability of a concept w.r.t. an $\mathcal{ALC}$ KBs into consistency of a class in an UML class diagram

We exploit the reductions in the hardness proof of reasoning over $\mathcal{AL}$ KBs:

- By step (1) it suffices to consider satisfiability of an atomic concept w.r.t. an $\mathcal{ALC}$ knowledge base with primitive inclusion assertions only, i.e., of the form
$$A \sqsubseteq C$$
- By step (2) it suffices to consider concepts on the right hand side that contain only a single construct, i.e., assertions of the form

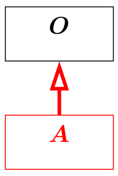
$$A \sqsubseteq B \qquad A \sqsubseteq \neg B \qquad A \sqsubseteq B_1 \sqcup B_2 \qquad A \sqsubseteq \forall P.B \qquad A \sqsubseteq \exists P.B$$

Note: by step (3) it would suffice to encode $A \sqsubseteq \exists P$ instead of $A \sqsubseteq \exists P.B$

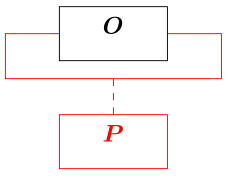
UML class diagram corresponding to an $\mathcal{ALC}$ KB

Given an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ of the simplified form above, we construct an UML class diagram $\mathcal{D}_\mathcal{K}$:

- we introduce in $\mathcal{D}_\mathcal{K}$ a class O, intended to represent the whole domain
- for each atomic concept A in $\mathcal{K}$, we introduce in $\mathcal{D}_\mathcal{K}$ a class A

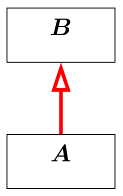


- for each atomic role P in $\mathcal{K}$, we introduce in $\mathcal{D}_\mathcal{K}$ a binary association P with related association class

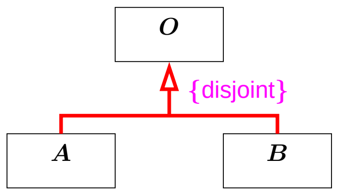


Encoding of $\mathcal{ALC}$ assertions

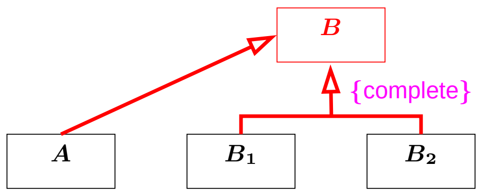
$A \sqsubseteq B$



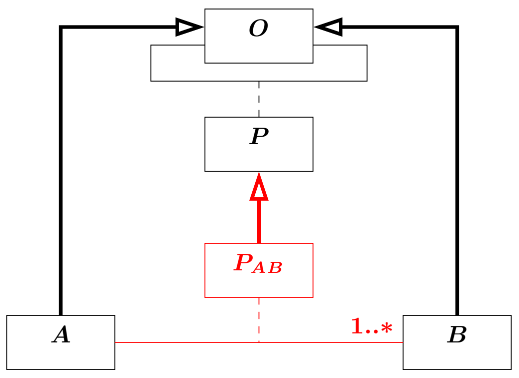
$A \sqsubseteq \neg B$



$A \sqsubseteq B_1 \sqcup B_2$

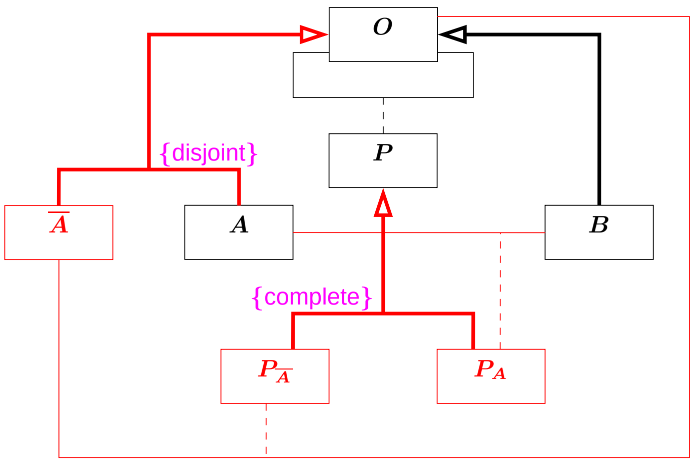


Encoding of $\mathcal{ALC}$ assertions (Cont'd)



$A \sqsubseteq \exists P.B$

Encoding of $\mathcal{ALC}$ assertions (Cont'd)



$$A \sqsubseteq \forall P.B$$

Correctness of the encoding

The encoding of an $\mathcal{ALC}$ knowledge base (of the simplified form) into an UML class diagram is correct, in the sense that it preserves concept satisfiability

Theorem:

An atomic concept A is satisfiable w.r.t. an $\mathcal{ALC}$ knowledge base $\mathcal{K}$
if and only if
the class A is consistent in the UML class diagram $\mathcal{D}_{\mathcal{K}}$ encoding $\mathcal{K}$

Proof idea: by showing a correspondence between the models of $\mathcal{K}$ and the models of (the FOL formalization of) $\mathcal{D}_{\mathcal{K}}$

Lower bound for reasoning on UML class diagrams

The UML class diagram $\mathcal{D}_{\mathcal{K}}$ constructed from an $\mathcal{ALC}$ knowledge base $\mathcal{K}$ is of polynomial size in $\mathcal{K}$

From

- EXPTIME-hardness of concept satisfiability w.r.t. an $\mathcal{ALC}$ knowledge base
- the fact that the encoding in polynomial

we obtain:

Reasoning on UML class diagrams is EXPTIME-hard

Upper bound for reasoning on UML class diagrams

EXPTIME upper bound established by encoding UML class diagrams in DLs

What we gain by such an encoding

- DLs admit decidable inference
  $\leadsto$ decision procedure for reasoning in UML
- (most) DLs are decidable in EXPTIME
  $\leadsto$ EXPTIME method for reasoning in UML (provided the encoding in polynomial)
- exploit DL-based reasoning systems for reasoning in UML
- interface case-tools with DL-based reasoners to provide support during design (see i.com demo)

Encoding of UML class diagrams in DLs

We encode an UML class diagram $\mathcal{D}$ into an $\mathcal{ALCQI}_{key}$ knowledge base $\mathcal{K}_{\mathcal{D}}$:

- classes are represented by concepts
- attributes and association roles are represented by roles
- each part of the diagram is encoded by suitable inclusion assertions
- the properties of association classes are encoded trough suitable key assertions

$\rightsquigarrow$ Consistency of a class in $\mathcal{D}$ is reduced to consistency of the corresponding concept w.r.t. $\mathcal{K}_{\mathcal{D}}$

Encoding of classes and attributes

- An UML class C is represented by an atomic concept C
- Each attribute a of type T for C is represented by an atomic role a
  - To encode the typing of a for C:

    $$C \sqsubseteq \forall a.T$$

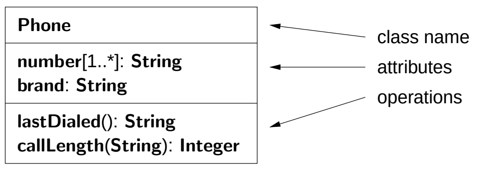
    This takes into account that other classes may also have attribute a
  - To encode the multiplicity $[i..j]$ of a:

    $$C \sqsubseteq (\geq i\, a) \sqcap (\leq j\, a)$$

    $*$ when j is $*$, we omit the second conjunct
    $*$ when the multiplicity is $[0..*]$ we omit the whole assertion
    $*$ when the multiplicity is missing (i.e., $[1..1]$), the assertion becomes:

    $$C \sqsubseteq \exists a \sqcap (\leq 1\, a)$$

Encoding of classes and attributes – Example



- To encode the class **Phone**, we introduce a concept **Phone**
- Encoding of the attributes: **number** and **brand**

$$\begin{aligned}\mathsf{Phone} &\sqsubseteq \forall \mathsf{number}.\mathsf{String} \sqcap \exists \mathsf{number}\\ \mathsf{Phone} &\sqsubseteq \forall \mathsf{brand}.\mathsf{String} \sqcap \exists \mathsf{brand} \sqcap (\leq 1\ \mathsf{brand})\end{aligned}$$

- Encoding of the operations: $\mathsf{lastDialed}()$ and $\mathsf{callLength}(\mathsf{String})$
  see later

Encoding of associations

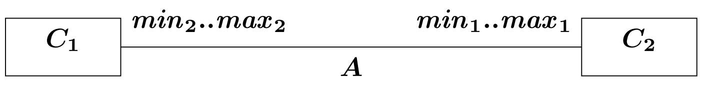
The encoding depends on:

- the presence/absence of an association class
- the arity of the association

	without association class	with association class
binary	via $\mathcal{ALCQI}$ role	via reification
non-binary	via reification	via reification

Note: an aggregation is just a particular kind of binary association without association class and is encoded via an $\mathcal{ALCQI}$ role

Encoding of binary associations without association class



- A is represented by an $\mathcal{ALCQI}$ role A, with:

$$\top \sqsubseteq \forall A.C_2 \sqcap \forall A^-.C_1$$

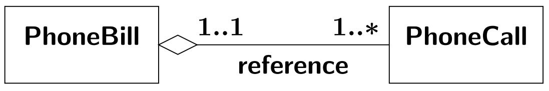
- To encode the multiplicities of A:
  - each instance of C_1 is connected through A to at least min_1 and at most max_1 instances of C_2:

$$C_1 \sqsubseteq (\geq min_1\ A) \sqcap (\leq max_1\ A)$$

  - each instance of C_2 is connected through A^- to at least min_2 and at most max_2 instances of C_1:

$$C_2 \sqsubseteq (\geq min_2\ A^-) \sqcap (\leq max_2\ A^-)$$

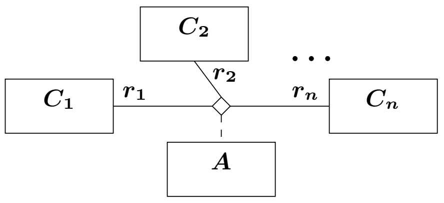
Binary associations without association class – Example

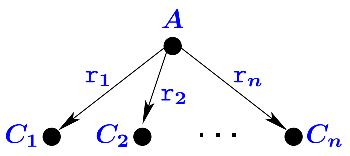


$$\begin{aligned}
\top &\sqsubseteq \forall \text{reference}.\text{PhoneCall} \sqcap \forall \text{reference}^-.\text{PhoneBill} \\
\text{PhoneBill} &\sqsubseteq (\geq 1\ \text{reference}) \\
\text{PhoneCall} &\sqsubseteq (\geq 1\ \text{reference}^-) \sqcap (\leq 1\ \text{reference}^-)
\end{aligned}$$

Note: an aggregation is just a particular kind of binary association without association class

Encoding of associations via reification





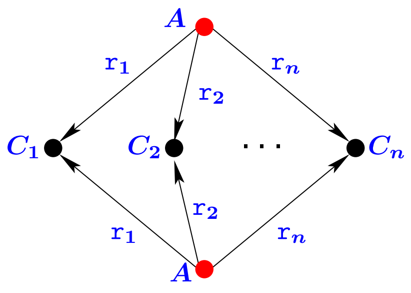
- Association A is represented by a concept A
- Each instance of the concept represents a tuple of the relation
- n (binary) roles $\mathtt{r}_1, \dots, \mathtt{r}_n$ are used to connect the object representing a tuple to the objects representing the components of the tuple
- To ensure that the instances of A correctly represent tuples:

$$A \quad \sqsubseteq \quad \exists \mathtt{r}_1.C_1 \sqcap \cdots \sqcap \exists \mathtt{r}_n.C_n \sqcap (\leq 1\, \mathtt{r}_1) \sqcap \cdots \sqcap (\leq 1\, \mathtt{r}_n)$$

Note: when the roles of A are explicitly named in the class diagram, we can use such role names instead of $\mathtt{r}_1, \dots, \mathtt{r}_n$

Encoding of associations via reification

We have not ruled out the existence of two instances of A representing the same tuple of association A:

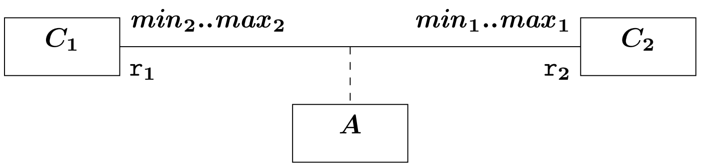


To rule out such a situation we could add a key assertion:

$$(\mathbf{key}\ A \mid \mathtt{r}_1, \dots, \mathtt{r}_n)$$

Note: in a tree-model the above situation cannot occur
$\leadsto$ Since in reasoning on an $\mathcal{ALCQI}$ KB we can restrict the attention to tree-models, we can ignore the key assertions

Multiplicities of binary associations with association class



To encode the multiplicities of A we need qualified number restrictions:

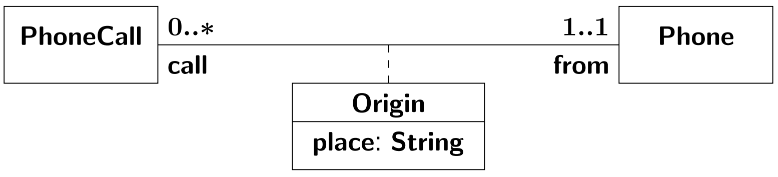
- each instance of C_1 is connected through A to at least min_1 and at most max_1 instances of C_2:

$$C_1 \sqsubseteq (\geq min_1\, r_1^-.A) \sqcap (\leq max_1\, r_1^-.A)$$

- each instance of C_2 is connected through A^- to at least min_2 and at most max_2 instances of C_1:

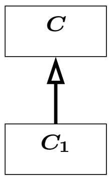
$$C_2 \sqsubseteq (\geq min_2\, r_2^-.A^-) \sqcap (\leq max_2\, r_2^-.A^-)$$

Associations with association class – Example

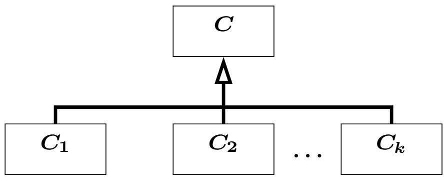


$$\begin{aligned}
\text{Origin} &\sqsubseteq \forall \text{place.String} \sqcap \exists \text{place} \sqcap (\leq 1\ \text{place})\\
\text{Origin} &\sqsubseteq \exists \text{call.PhoneCall} \sqcap (\leq 1\ \text{call}) \sqcap\\
&\quad \exists \text{from.Phone} \sqcap (\leq 1\ \text{from})\\
\text{PhoneCall} &\sqsubseteq (\geq 1\ \text{call}^-.\text{Origin}) \sqcap (\leq 1\ \text{call}^-.\text{Origin})
\end{aligned}$$

Encoding of ISA and generalization



$$C_1 \sqsubseteq C$$



$$C_1 \sqsubseteq C$$
$$\vdots$$
$$C_k \sqsubseteq C$$

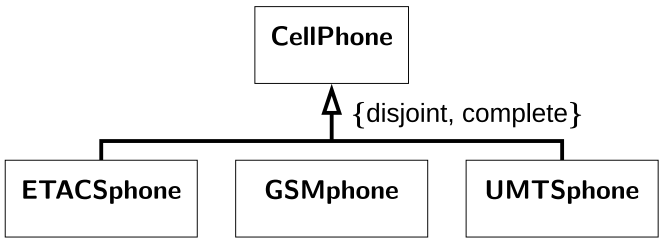
- When the generalization is disjoint

$$C_i \sqsubseteq \neg C_j \qquad \text{for } 1 \leq i < j \leq k$$

- When the generalization is complete

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_k$$

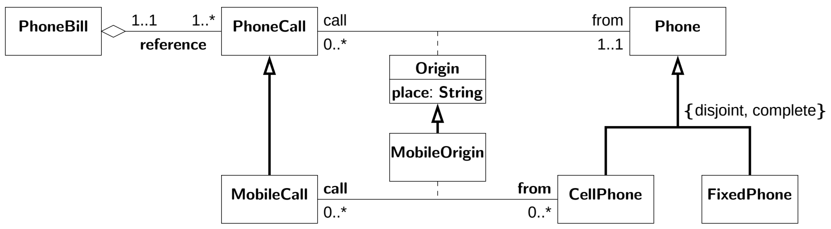
ISA and generalization – Example



ETACSphone	$\sqsubseteq$	CellPhone	ETACSphone	$\sqsubseteq$	$\neg$GSMPhone
GSMSphone	$\sqsubseteq$	CellPhone	ETACSphone	$\sqsubseteq$	$\neg$UMTSPhone
UMTSSphone	$\sqsubseteq$	CellPhone	GSMphone	$\sqsubseteq$	$\neg$UMTSPhone

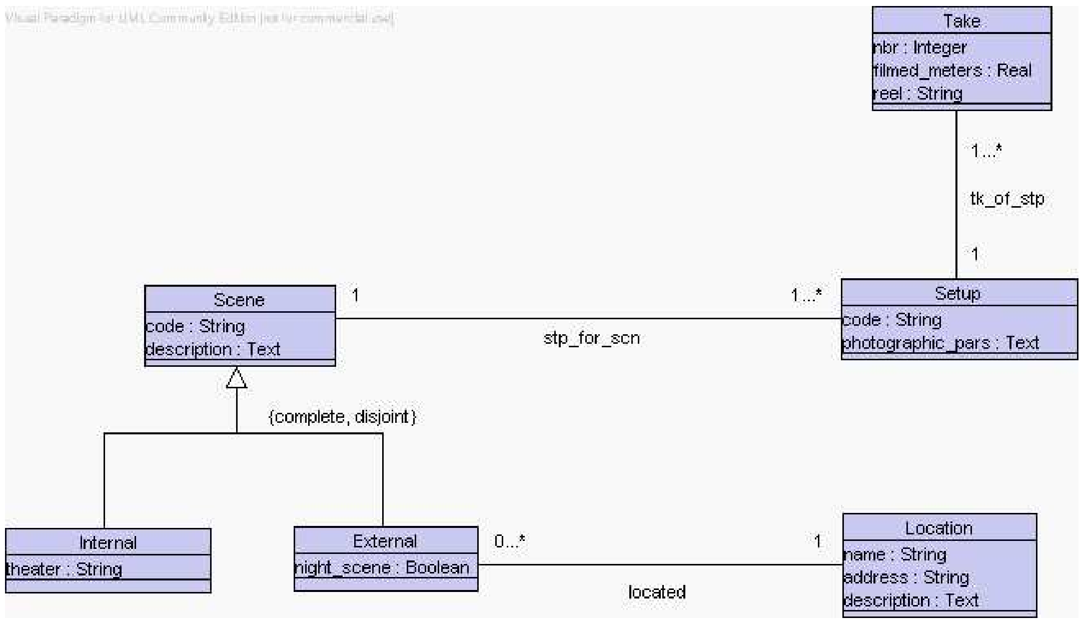
CellPhone $\sqsubseteq$ ETACSphone $\sqcup$ GSMphone $\sqcup$ UMTSPhone

Encoding of UML in DLs – Example



$$
\begin{array}{rcl}
\top & \sqsubseteq & \forall \mathsf{reference}.\mathsf{PhoneCall} \sqcap \forall \mathsf{reference}^{-}.\mathsf{PhoneBill} \\
\mathsf{PhoneBill} & \sqsubseteq & (\geq 1\, \mathsf{reference}) \\
\mathsf{PhoneCall} & \sqsubseteq & (\geq 1\, \mathsf{reference}^{-}) \sqcap (\leq 1\, \mathsf{reference}^{-}) \\
\mathsf{Origin} & \sqsubseteq & \forall \mathsf{place}.\mathsf{String} \sqcap \exists \mathsf{place} \sqcap (\leq 1\, \mathsf{place}) \\
\mathsf{Origin} & \sqsubseteq & \exists \mathsf{call}.\mathsf{PhoneCall} \sqcap (\leq 1\, \mathsf{call}) \sqcap \exists \mathsf{from}.\mathsf{Phone} \sqcap (\leq 1\, \mathsf{from}) \\
\mathsf{MobileOrigin} & \sqsubseteq & \exists \mathsf{call}.\mathsf{MobileCall} \sqcap (\leq 1\, \mathsf{call}) \sqcap \exists \mathsf{from}.\mathsf{CellPhone} \sqcap (\leq 1\, \mathsf{from}) \\
\mathsf{PhoneCall} & \sqsubseteq & (\geq 1\, \mathsf{call}^{-}.\mathsf{Origin}) \sqcap (\leq 1\, \mathsf{call}^{-}.\mathsf{Origin}) \\
\mathsf{MobileOrigin} & \sqsubseteq & \mathsf{Origin} \\
\mathsf{MobileCall} & \sqsubseteq & \mathsf{PhoneCall} \\
\mathsf{CellPhone} & \sqsubseteq & \mathsf{Phone} \\
\mathsf{FixedPhone} & \sqsubseteq & \mathsf{Phone} \sqcap \neg \mathsf{CellPhone} \\
\mathsf{Phone} & \sqsubseteq & \mathsf{CellPhone} \sqcup \mathsf{FixedPhone}
\end{array}
$$

Encoding of UML in DLs – Exercise 1



Translate the above UML class diagram into an $\mathcal{ALCQI}$ knowledge base

Encoding of UML in DLs – Solution of Exercise 1

Encoding of classes and attributes

$$\begin{array}{rcl}
\mathsf{Scene} & \sqsubseteq & \forall \mathsf{code}.\mathsf{String} \sqcap \exists \mathsf{code} \sqcap (\leq 1\, \mathsf{code}) \\
\mathsf{Scene} & \sqsubseteq & \forall \mathsf{description}.\mathsf{Text} \sqcap \exists \mathsf{description} \sqcap (\leq 1\, \mathsf{description}) \\
\mathsf{Internal} & \sqsubseteq & \forall \mathsf{theater}.\mathsf{String} \sqcap \exists \mathsf{theater} \sqcap (\leq 1\, \mathsf{theater}) \\
\mathsf{External} & \sqsubseteq & \forall \mathsf{night\_scene}.\mathsf{Boolean} \sqcap \exists \mathsf{night\_scene} \sqcap (\leq 1\, \mathsf{night\_scene}) \\
\mathsf{Take} & \sqsubseteq & \forall \mathsf{nbr}.\mathsf{Integer} \sqcap \exists \mathsf{nbr} \sqcap (\leq 1\, \mathsf{nbr}) \\
\mathsf{Take} & \sqsubseteq & \forall \mathsf{filmed\_meters}.\mathsf{Real} \sqcap \exists \mathsf{filmed\_meters} \sqcap (\leq 1\, \mathsf{filmed\_meters}) \\
\mathsf{Take} & \sqsubseteq & \forall \mathsf{reel}.\mathsf{String} \sqcap \exists \mathsf{reel} \sqcap (\leq 1\, \mathsf{reel}) \\
\mathsf{Setup} & \sqsubseteq & \forall \mathsf{code}.\mathsf{String} \sqcap \exists \mathsf{code} \sqcap (\leq 1\, \mathsf{code}) \\
\mathsf{Setup} & \sqsubseteq & \forall \mathsf{photographic\_pars}.\mathsf{Text} \sqcap \exists \mathsf{photographic\_pars} \sqcap (\leq 1\, \mathsf{photographic\_pars}) \\
\mathsf{Location} & \sqsubseteq & \forall \mathsf{name}.\mathsf{String} \sqcap \exists \mathsf{name} \sqcap (\leq 1\, \mathsf{name}) \\
\mathsf{Location} & \sqsubseteq & \forall \mathsf{address}.\mathsf{String} \sqcap \exists \mathsf{address} \sqcap (\leq 1\, \mathsf{address}) \\
\mathsf{Location} & \sqsubseteq & \forall \mathsf{description}.\mathsf{Text} \sqcap \exists \mathsf{description} \sqcap (\leq 1\, \mathsf{description})
\end{array}$$

Encoding of UML in DLs – Solution of Exercise 1 (Cont'd)

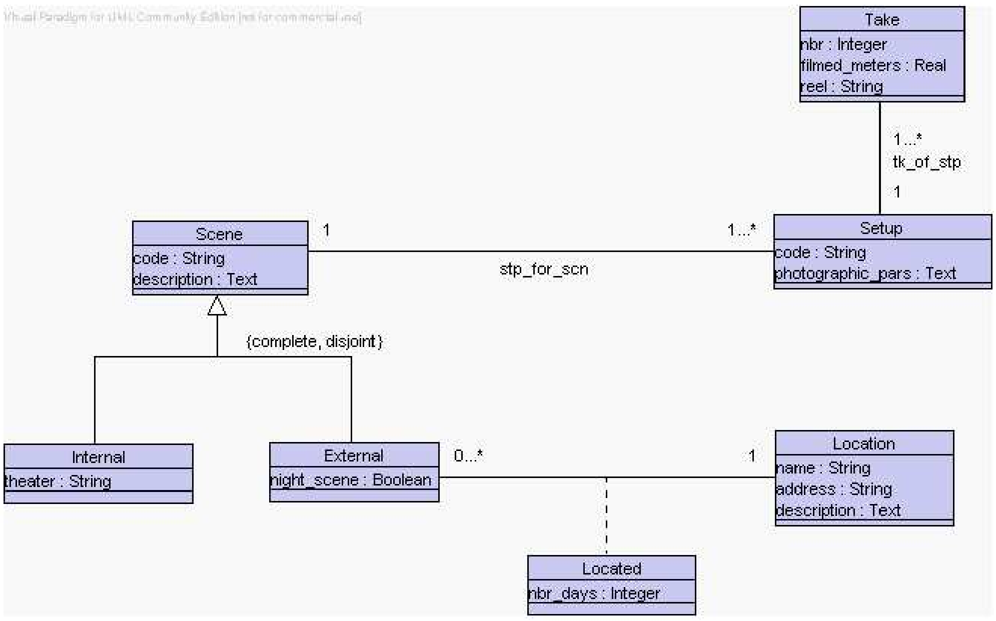
Encoding of hierarchies

$$\begin{array}{rcl}
\mathsf{Internal} & \sqsubseteq & \mathsf{Scene} \\
\mathsf{External} & \sqsubseteq & \mathsf{Scene} \\
\mathsf{Scene} & \sqsubseteq & \mathsf{Internal} \sqcup \mathsf{External} \\
\mathsf{Internal} & \sqsubseteq & \neg \mathsf{External}
\end{array}$$

Encoding of associations

$$\begin{array}{rcl}
\top & \sqsubseteq & \forall \mathsf{stp\_for\_scn}.\mathsf{Setup} \sqcap \forall \mathsf{stp\_for\_scn}^-.\mathsf{Scene} \\
\mathsf{Scene} & \sqsubseteq & (\geq 1\, \mathsf{stp\_for\_scn}) \\
\mathsf{Setup} & \sqsubseteq & (\geq 1\, \mathsf{stp\_for\_scn}^-) \sqcap (\leq 1\, \mathsf{stp\_for\_scn}^-) \\
\top & \sqsubseteq & \forall \mathsf{tk\_of\_stp}.\mathsf{Take} \sqcap \forall \mathsf{tk\_of\_stp}^-.\mathsf{Setup} \\
\mathsf{Setup} & \sqsubseteq & (\geq 1\, \mathsf{tk\_of\_stp}) \\
\mathsf{Take} & \sqsubseteq & (\geq 1\, \mathsf{tk\_of\_stp}^-) \sqcap (\leq 1\, \mathsf{tk\_of\_stp}^-) \\
\top & \sqsubseteq & \forall \mathsf{located}.\mathsf{Location} \sqcap \forall \mathsf{located}^-.\mathsf{External} \\
\mathsf{External} & \sqsubseteq & (\geq 1\, \mathsf{located}) \sqcap (\leq 1\, \mathsf{located})
\end{array}$$

Encoding of UML in DLs – Exercise 2



How does the translation change w.r.t. the one for Exercise 1?

Encoding of UML in DLs – Solution of Exercise 2

The change is in the encoding of the association **located**, which now must be reified into a concept **Located**, i.e.,

replace

$$\begin{array}{rcl} \top & \sqsubseteq & \forall \mathsf{located}.\mathsf{Location} \sqcap \forall \mathsf{located}^-.\mathsf{External} \\ \mathsf{External} & \sqsubseteq & (\geq 1\, \mathsf{located}) \sqcap (\leq 1\, \mathsf{located}) \end{array}$$

with

$$\begin{array}{rcl} \mathsf{Located} & \sqsubseteq & \exists \mathsf{r}_1.\mathsf{External} \sqcap (\leq 1\, \mathsf{r}_1) \sqcap \exists \mathsf{r}_2.\mathsf{Location} \sqcap (\leq 1\, \mathsf{r}_2) \\ \mathsf{External} & \sqsubseteq & (\geq 1\, \mathsf{r}_1.\mathsf{Located}) \sqcap (\leq 1\, \mathsf{r}_1.\mathsf{Located}) \\ \mathsf{Located} & \sqsubseteq & \forall \mathsf{nbr\_days}.\mathsf{Integer} \sqcap \exists \mathsf{nbr\_days} \sqcap (\leq 1\, \mathsf{nbr\_days}) \end{array}$$

Encoding of operations

Operation $f(P_1, \ldots, P_m) : R$ for class C corresponds to an $(m+2)$-ary relation that is functional on the last component

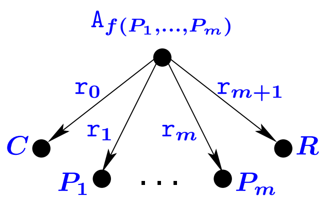
- Operation $f() : R$ without parameters directly represented by an atomic role $\mathtt{P}_{f()}$, with:

$$C \sqsubseteq \forall \mathtt{P}_{f()}.R \sqcap (\leq 1\, \mathtt{P}_{f()})$$

- Operation $f(P_1, \ldots, P_m) : R$ with one or more parameters cannot be expressed directly in $\mathcal{ALCQI}_{key}$ $\leadsto$ we make use of reification:
  - relation is reified by using a concept $\mathtt{A}_{f(P_1,\ldots,P_m)}$
  - each instance of the concept represents a tuple of the relation
  - (binary) roles $\mathtt{r}_0, \ldots, \mathtt{r}_{m+1}$ connect the object representing a tuple to the objects representing the components of the tuple

Reification of operations

To represent operation $f(P_1, \ldots, P_m) : R$ for class C:



$$\mathtt{A}_{f(P_1,\ldots,P_m)} \sqsubseteq \exists \mathtt{r}_0 \sqcap \cdots \sqcap \exists \mathtt{r}_{m+1} \sqcap (\leq 1\, \mathtt{r}_0) \sqcap \cdots \sqcap (\leq 1\, \mathtt{r}_{m+1}) \quad (1)$$

$$\mathtt{A}_{f(P_1,\ldots,P_m)} \sqsubseteq \forall \mathtt{r}_1.P_1 \sqcap \cdots \sqcap \forall \mathtt{r}_m.P_m \quad (2)$$

$$C \sqsubseteq \forall \mathtt{r}_0^-.(\mathtt{A}_{f(P_1,\ldots,P_m)} \Rightarrow \forall \mathtt{r}_{m+1}.R) \quad (3)$$

(1) ensures that the instances of $\mathtt{A}_{f(P_1,\ldots,P_m)}$ represent tuples

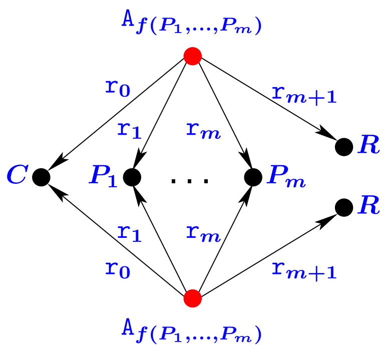
(2) ensures that the parameters of the operation have the correct types

(3) ensures that, when the operation is applied to an instance of C, then the result is an instance of R

Note: the name of the concept representing the operation includes the types of the parameters, but not the invocation class or the type of the return value
$\leadsto$ allows for correct encoding of overloading of operations

Reification of operations (Cont'd)

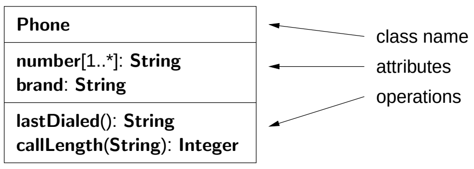
Again, we have not ruled out two instances of $\mathtt{A}_{f(P_1,\ldots,P_m)}$ representing two applications of the operation with identical parameters but different result:



To rule out such a situation we could add a key assertion:
$(\mathbf{key}\ \mathtt{A}_{f(P_1,\ldots,P_m)} \mid \mathtt{r}_0, \mathtt{r}_1, \ldots, \mathtt{r}_m)$

Again, by the tree-model property of $\mathcal{ALCQI}$, we can ignore the key assertion for reasoning

Encoding of operations – Example



- Encoding of the attributes: **number** and **brand**

$$\begin{array}{lcl} \mathsf{Phone} & \sqsubseteq & \forall \mathsf{number}.\mathsf{String} \sqcap \exists \mathsf{number} \\ \mathsf{Phone} & \sqsubseteq & \forall \mathsf{brand}.\mathsf{String} \sqcap \exists \mathsf{brand} \sqcap (\leq 1\, \mathsf{brand}) \end{array}$$

- Encoding of the operations: **lastDialed()** and **callLength(String)**

$$\begin{array}{lcl} \mathsf{Phone} & \sqsubseteq & \forall \mathtt{P}_{\mathsf{lastDialed}()}.\mathsf{String} \sqcap (\leq 1\, \mathtt{P}_{\mathsf{lastDialed}()}) \\ \mathtt{P}_{\mathsf{callLength}(\mathsf{String})} & \sqsubseteq & \exists \mathtt{r}_0 \sqcap (\leq 1\, \mathtt{r}_0) \sqcap \exists \mathtt{r}_1 \sqcap (\leq 1\, \mathtt{r}_1) \sqcap \exists \mathtt{r}_2 \sqcap (\leq 1\, \mathtt{r}_2) \\ \mathtt{P}_{\mathsf{callLength}(\mathsf{String})} & \sqsubseteq & \forall \mathtt{r}_1.\mathsf{String} \\ \mathsf{Phone} & \sqsubseteq & \forall \mathtt{r}_0^-.(\mathtt{P}_{\mathsf{callLength}(\mathsf{String})} \Rightarrow \forall \mathtt{r}_2.\mathsf{Integer}) \end{array}$$

Correctness of the encoding

The encoding of an UML class diagram into an $\mathcal{ALCQI}$ knowledge base is correct, in the sense that it preserves the reasoning services over UML class diagrams

Theorem:

A class C is consistent in an UML class diagram $\mathcal{D}$
if and only if
the concept C is satisfiable in the $\mathcal{ALCQI}$ knowledge base $\mathcal{K}_{\mathcal{D}}$ encoding $\mathcal{D}$

Proof idea: by showing a correspondence between the models of (the FOL formalization of) $\mathcal{D}$ and the models of $\mathcal{K}_{\mathcal{D}}$

Complexity of reasoning on UML class diagrams

All reasoning tasks on UML class diagrams can be reduced to reasoning tasks on $\mathcal{ALCQI}$ knowledge bases

From

- EXPTIME-completeness of reasoning on $\mathcal{ALCQI}$ knowledge bases
- the fact that the encoding in polynomial

we obtain:

Reasoning on UML class diagrams can be done in EXPTIME

Conclusions

- We have formalized UML class diagrams in logics, which gives us the ability to reason on them so as to detect and deduce relevant properties
- We have provided an encoding in the DL $\mathcal{ALCQI}$ thus showing that:
  1. Reasoning on UML class diagrams is decidable, and in fact EXPTIME-complete, and thus can be automatized
  2. We can perform such automated reasoning using state-of-the-art DL reasoning systems

The above results lay the foundation for advanced CASE tools with integrated automated reasoning support

Such a prototype tool is i.com, developed at the Univ. of Bolzano