Description Logics

Primitive and Defined Concepts
Specialization/Generalization
Concept and Role Constructors
Reasoning with Concepts and Individuals
CLASSIC and SHIQ
Description Logics and UML, ER and CORBA

Conceptual Modeling CSC2507

Description Logics

- A ***precise*** notation for representing "noun phrases".
- An Information Management system that uses these to make assertions, and answer questions based on ***inferences*** - a logic.
- Fundamental ontology: the application domain is populated by ***individuals***, related by binary relationships, called ***roles/attributes***, grouped into ***classes*** (***concepts***).
- First Order Logic (FOL) would be fine for describing these, but it is intractable (semi-decidable, to be exact)
- Looking for a restricted subset, which allows us to reason with it while still representing useful things.

Datalog is another such useful subset of FOL

Primitive and Defined Concepts

- In addition to ***primitive*** concepts (***natural kinds***), such as PERSON, CHAIR, ... there are ***defined*** concepts.
- Defined concepts have names:
  - ✓ "person with gender=M and no object related to it by hasSpouse" ==> "BACHELOR"
  - ✓ "person with age between 13 and 17" ==> "TEENAGER"
  - ✓ "person who eats only non-meat foods" ==> "VEGETARIAN"
- They are describable by relative clauses or compound nouns:
  - ✓ "person who has at least 3 children"
  - ✓ "towns located in MA or NH or VT,.." (NE_TOWNS)

Specialization/Generalization

- Both primitive and defined concepts can have additional assertions made about them, representing necessary conditions. A standard way to make such assertions is to use is-a-subconcept-of/is-subsumed-by/is-a-kind-of (:<).
- For example,
  - ✓ PERSON :< ANIMATE ,
  - ✓ PERSON :< TEENAGER :< LIKES-FRIES
- Note (AB): Liking French fries is not part of the definition of TEENAGER, even though all teenagers have this property.

A Language for Defining Concepts

- We need a language for defining concepts, based on experiences with what has been useful in many applications:
  - ✓ Atomic/primitive concepts: PERSON, COURSE, BOOK
  - ✓ Boolean combinations thereof:
    - ✓ ANIMAL and HERBIVORE;
    - ✓ not ANIMATE;
    - ✓ PERSON or CORPORATION;
    - ✓ PERSON and (not MALE).

…But Also…

- Concepts defined by enumeration of individuals: {M,F}
- Concepts from "concrete domains" (numbers, Programming Language values)
- Sets of objects satisfying restrictions on their role fillers (for this, we need some atomic/primitive roles: graduateOf, locatedIn, likes, hasPart)
  - ✓ Objects all of whose locatedIn values are in NE_TOWNS
  - ✓ Objects some of whose graduateOf values are in UNIVERSITY
  - ✓ Objects with at least 3 graduateOf fillers
  - ✓ Objects related to the Rutgers object by graduateOf (= Objects whose graduateOf role includes Rutgers as filler).

Concept Constructors

ANIMAL and HERBIVORE	(and ANIMAL HERBIVORE)
not ANIMATE	(not ANIMATE)
PERSON or CORPORATION	(or PERSON CORPORATION)
PERSON and (not MALE)	(and PERSON (not MALE))
{M,F}	(one-of M F)
Objects with locatedIn values in NE_TOWN	(all locatedIn NE_TOWN)
Objects with some graduateOf values in UNIVERSITY	(some graduateOf UNIVERSITY)
Objects with at least 3 graduateOf fillers	(at-least 3 graduateOf)
Objects with graduateOf fillers that include Rutgers	(fills graduateOf Rutgers)
...	

More Concept Constructors

- (at-least 3 children DOCTOR)
  //contrast this with (and (at-least 3 children)
  (all children DOCTOR))
- (domain graduateOf)
  //objects having a filler for graduateOf role
- (range graduateOf)
  //objects which are fillers of graduateOf role
- (same-as (firstName) (lastName))
  // objects for which the firstName and lastName
  values are identical
- (subsetOf (friends) (co-workers))
  // objects whose co-workers include all their friends

Syntax

- *Can describe concepts of arbitrary complexity by nesting;*

  e.g., "Courses taken by 60 to 90 students, who are all undergrads, and taught by a CS professor"

```
(and
      COURSE
      (at-least 60 takers)
      (at-most 90 takers)
      (all takers (and STUDENT
                       (all inYear (one-of 1 2 3 4))))
      (exactly 1 taughtBy)
      (all  taughtBy (and PROFESSOR
                          (fills inDepartment "CS"))))
```


…More Syntax…

"Persons who eat only non-meat"

- *(:and PERSON (:all eats (:not MEAT)))*
- *and(PERSON, all(eats,not(MEAT)))*
- *PERSON $\sqcap \forall$ eats.$\neg$MEAT*
-
```
<concept> <and>
        <primitive name="PERSON"/>
        <all>
              <primrole name="eats"/>
              <not> <primitive name="MEAT"/>
        </not> </all> </and> </concept>
```

OWL Syntax

- <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#PERSON" />
    <owl:Restriction>
      <owl:onProperty rdf:resource="#eats"/>
      <owl:allValuesFrom>
      <owl:complementOf rdf:resource="#MEAT" />
      </owl:allValuesFrom>
    </owl:Restriction>
  </owl:intersectionOf>

Roles

- *Fundamental observation: Relationships are like concepts! Hence they can also be defined, using* ***role constructors****.*
- *childOf* is the inverse of *hasChildren*
  (inverse hasChildren)
- *descendantOf* is the transitive closure of *childOf*
  (trans childOf)
- *sonOf* is the restriction of *childOf* so that its range of values is MALE
  (restriction childOf MALE)
- *nephewOf* is the composition of *sonOf* and *siblingOf*
  (compose sonOf siblingOf)

Summary

- Descriptions are composite, variable-free **terms**, which can be built up from primitive symbols, using **constructors.**
- There are constructors for both concepts and roles (binary relationships)
- There is a collection of constructors that have been found useful over the years.
- Except for transitive closure of roles, all other constructors can be expressed in FOL -- see some examples later. (In fact, you only need 3 variables, if you can reuse them when nesting)

Standard Reasoning

- Does concept C **subsume** concept D? D :< C
  - ✓(and PERSON MALE) :< PERSON
  - ✓(at-least 3 hasChildren) :< (at-least 1 hasChildren)
  - ✓(at-most 2 parts) :< (at-most 10 parts)
- Suppose now hasSons :< hasChildren
  - ✓(all hasSons STUDENT) :< (all hasChildren PERSON)
  - ✓(fills hasSons Adam) :< (at-least 1 hasChildren)

Incoherence

- Is concept C incoherent?

  (and PERSON
  (at-least 3 hasDegree)
  (all hasDegree (one-of "BA" "BS"))
- Another way of putting it: Is C :< NOTHING?
- Problem: reasoning with the complete set of concept constructors we encountered is still as hard as for all of FOL!
- Solution: Description Logics research has been about finding subsets of constructors and characterizing the computational complexity of reasoning with them.

The CLASSIC Description Logic

primitive concept	C(.)
role	r(.,.)
attribute (role with at most 1 filler)	f(.,.)
(all p C)	$y \mid \forall z.\ p(y,z) \Rightarrow C(z)$
(at-least n p)	$y \mid \exists^{\geq n} z.\ p(y,z)$
(at-most m p)	$y \mid \exists^{\leq m} z.\ p(y,z)$
(one-of (e1 e2 ...))	$y \mid y=e1 \vee y=e2 \vee ...$
(fills p e)	$y \mid p(y,e)$
(same-as (f_1 f_2 ... f_n) (g_1 ... g_k))	$y \mid f_n(... (f_1 (y))) = g_k(... (g_1(y)))$
THING, CL-THING, NOTHING	y | true y | false
H-THING , NUMBER	
(min n)	$w \mid w \geq n$
(max m)	$w \mid w \leq m$
(inverse r)	$(x,y) \mid r(y,x)$
(test *fn* arg1 ... argk)	

SHIQ DL -- FaCT Reasoner

primitive concept	C(.)
role	r(.,.)
(and C D)	$y \mid C(y) \wedge D(y)$
(or C D)	$y \mid C(y) \vee D(y)$
(not C)	$y \mid \neg C(y)$
(all p C)	$y \mid \forall z.\ p(y,z) \Rightarrow C(z)$
(some p C)	$y \mid \exists z.\ p(y,z) \wedge C(z)$
(at-least n p C)	$y \mid \exists^{\geq n} z.\ p(y,z) \wedge C(z)$
(at-most m p C)	$y \mid \exists^{\leq m} z.\ p(y,z) \wedge C(z)$
(inverse r)	$(x,y) \mid r(y,x)$
r is transitive	$(y,w) \mid \forall z.\ r(y,z) \wedge r(z,w) \Rightarrow r(y,w)$

[Horrocks]

Using a DL as an Information Manager

- Descriptions can be used in *all* these languages!
- Variety of ***ASK*** operations about concepts
  - ✓ concept-subsumes?(C,D): boolean
  - ✓ incoherent?(C)): boolean
  - ✓ concept-disjoint?(C,D)): boolean

The rest of the slides use CLASSIC

Constraints

- Provide necessary conditions for primitive concepts and roles (***axioms***) of the form *name :<D* , *name :< r*
  - ✓ (*define-primitive-concept* PERSON
    (and THING (all age INTEGER)))
  - ✓ (*define-primitive-role* wifeOf
    :is-a spouseOf, *:inverse* husbandOf, *:attrib* True)
  - ✓ (*define-disjoint-primitive-concept* BIRD ANIMATE genus)

ops provided by IM;
in Classic, have prefix cl-

Definitions

- Definitions for defined concepts (name $=_{def}$ C)
  - ✓ (*define-concept* TEENAGER
    (and PERSON (all age (and (min 13) (max 17))))))
- Subsumption constraints between complex concepts ("general axioms")
  "graduate courses are taught by tenured professors"
  (and COURSE (all crsNumber (min 500))) :<
  (all taughtBy (and PROFESSOR
  (all title (one-of 'associate 'full)))
- A collection of such axioms is known as a ***T-box*** (Terminological box)

Reasoning with Axioms

- Non-recursive axioms about primitives can be expanded: if you have axiom N :< C, replace all occurrences of N by (and N C) -- CLASSIC only supports this for concepts and for roles
- General axioms are much harder to reason with (Exponential Time Complete problem for SHIQ -- seems to be ok, however, for practical KBs)
- FaCT supports this for concepts; also r1 :< r2 for roles

Recursive Axioms

- Recursive axioms, e.g., PERSON :< (all parents PERSON)
- What does this mean?

  Suppose FOO $=_{def}$ (all parents FOO)

  and BAR $=_{def}$ (all parents BAR);

  Does this mean that FOO and BAR are the same concept?
- Recursive concept definitions also make reasoning hard -- FaCT supports these too.

Managing Definitions

What kinds of services might an Information Manager offer for definition management?

- Detecting inconsistent concepts;
- Finding concepts that mean the same thing, and letting user know of the alias;
- Automatically organizing definitions into a subconcept hierarchy by finding for each concept, most specific existing subsumers and most general subsumees.

Reasoning with Concepts in Classic

- Primitive concepts: INSTRUCTOR , COURSE
- Primitive roles: takers Attributes teaches, taughtBy

  COURSE <: (and (all taughtBy INSTRUCTOR)
  (same-as (self) (taughtBy teaches)))
- "Desirable classes are ones taught by lucky instructors"

  DESIRABLE $=_{def}$ (and COURSE (all taughtBy LUCKY))
- "Lucky instructors teach classes with small enrolments"

  LUCKY $=_{def}$ (and INSTR (all teaches (atmost 8 takers)))
- "Small courses have few students in them (fewer than 8 takers)."

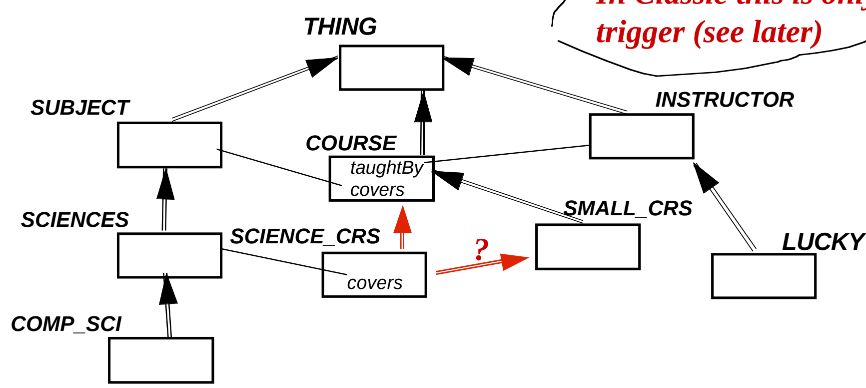
SMALL_COURSE $=_{def}$ (and COURSE (atmost 8 takers))

DESIRABLE == SMALL_COURSE

Classification

- SCIENCE_CRS $=_{def}$ (and COURSE (all covers SCIENCES))
- SCIENCE_CRS :< (and (atmost 7 takers) (all takers (fills year 4)))

In Classic this is only a trigger (see later)



CLASSIC Rules

- *Classic **rules** express general axioms like C :< D, for named concepts C, even if C is a defined concept. But these are not used in subsumption reasoning. So the inference SCIENCE_CRS :< SMALL_CRS would not be made in CLASSIC.*
- *Rules act like triggers: when an **individual** becomes an instance of C, it is also added to D. This means that even for individuals, reasoning does not work backwards: from not D one cannot infer not C.*

Test Concepts

- You can think of test-concepts as placeholders where one can put constructors not available in Classic.
- So for example, you can say
  (test-c some child Doctor)
  where presumably you will eventually define a 3 place function called *some*
  (*some* <role> <Concept> <individual>)
  that checks if the third argument satisfies what we know (some r C) means.
- But this definition of some cannot be used in subsumption reasoning -- it will be treated as a black box.

The Assertional Box (A-Box)

- The A-Box provides operations for manipulating individuals, and relationships between them:
  - ✓ Create individuals
    (*ind-create* cs430)
  - ✓ Inter-relate them
    (*ind-add-fillers* cs430 covers Databases)
    (*ind-add-fillers* cs430 taughtBy Gabrielle)
  - ✓ Assert them to be instances of concepts
    (*ind-add* cs430 COURSE)

Capturing Incomplete Information

- A full DL can be used as part of the A-Box Tell language:
  - ✓ (ind-add cs323 (all taughtBy (fills dept "CS")))

    "We do not know who teaches cs323, but it will be from CS dept"
  - ✓ (ind-add mahdi (all hasDegree (one-of BA BS)))

    "Mahdi's degree is either BA or BS"
- This feature can be used to captur ***incomplete*** information.

Reasoning with Individuals

- Individuals can be asserted to satisfy descriptions, e.g.,

  Calvin : PERSON

  Calvin : (all friendOf (the age (and (min 5) (max 7))))
- Consistency checking: From friendOf(Calvin, Susie) verify that Susie's age is not known to be under 5 or over 7
- Propagation: If Susie's age is not known, then infer partial information

  Susie : (the age (and (min 5) (max 7)))
- Individual Classification: In either case, if

  CHILD $=_{def}$ (the age (and (min 0) (max 12)))

  then Susie is inferred to be a child

  Susie : CHILD

Open World Reasoning

- Suppose you have been told the following
  (ind-creat Calvin), (ind-create Susie)
  (ind-add-filler Calvin friendOf Susie)
- From {friendOf(Calvin,Susie), Susie:FEMALE} one cannot conclude Calvin : (all friendOf FEMALE) because not all friends might be known at this time -- one might find more in the future.
- The way to say that all friends are known is to add an atmost bound equal to the current number of fillers:
  (*ind-add* Calvin (atmost 1 friendOf)).

This "closes" the role friendOf on Calvin, and allows all restrictions to be checked on role friendOf for Clavin.

[Classic will do the counting and add the at-most automatically, if you just say (*ind-close-role* Calvin friendOf)]

Reasoning with Individuals

Remember:

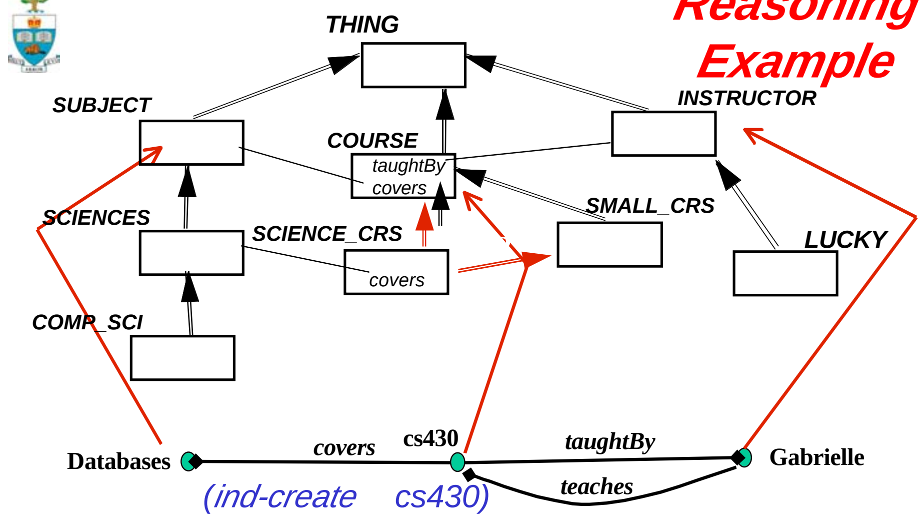
COURSE <: (and (all taughtBy INSTRUCTOR)
(all covers SUBJECT)
(same-as self (teaches taughtBy)))

LUCKY $=_{def}$ (and INSTR (all teaches (atmost 8 takers)))

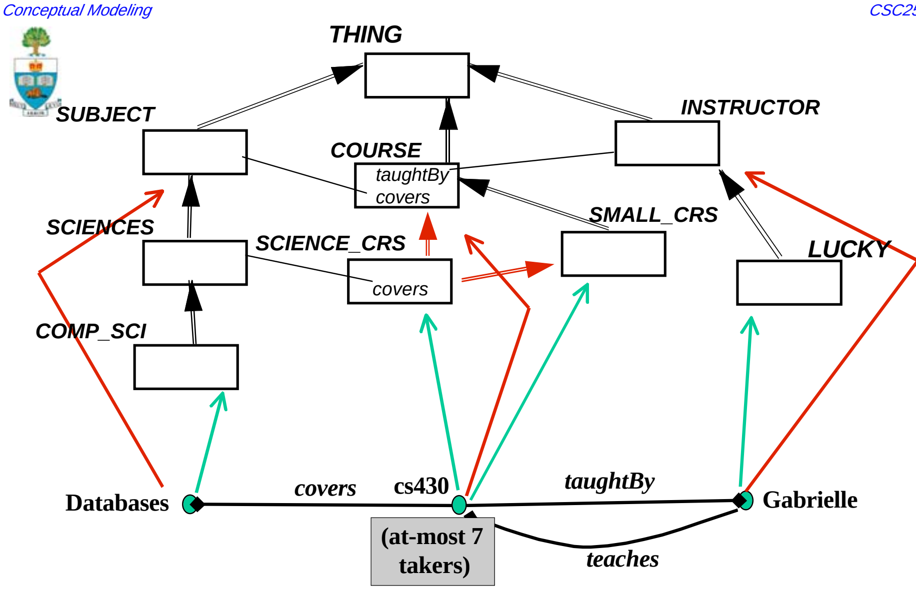
SCIENCE_CRS $=_{def}$ (and COURSE (all covers
SCIENCES))

SCIENCE_CRS :< (and (at-most 7 takers)
(all takers (fills year 4)))

Reasoning Example



(ind-create cs430)
(ind-add-fillers cs430 covers Database)
(ind-add-fillers cs430 taughtBy Gabrielle)
(ind-add cs430 COURSE)



Also add that Database is a COMP_SCI instance:
(ind-add Database COMP_SCI)

Questions about Individuals

- *(instance? <individual e> <concept C>) -- test for membership.*
- *(instances <concept Q>): -- what are instances classified under Q; i.e., any concept can act as query.*
- *(ind-parents/ancestors <individual e>) -- what named concepts is e an instance of.*
- *(fillers <individual e> <role r>) -- fillers of role r for e.*
- *(ind-expr <individual e>) -- description of e.*
- *(cl-all <individuals e>)*
  - ✓ *What is Susie's age?": (cl-all Susie age)*
    would return (and (min 5) (max 7))
  - ✓ *"What is common to all takers of a SCIENCE_CRS?":*
    (cl-all cs430 takers) ==> (fills year 4)

Description Logics for Information Management

- ***Define the schema:*** *define-concept* ***L - Define***
- ***Define views:*** *define-concept* ***L - Define***
- ***Describe individuals partially :*** *assert-ind* ***L - Tell***
- ***State queries:*** *ask-instances* ***L - Ask***
- ***Intensional answers:*** *concept-aspect* ***L - Answer***
- ***Set up simple triggers:*** *assert-rule* ***L - Tell***

The Meaning of Links Revisited

$FOO \xrightarrow{hasColor} GREEN$

How does this help disambiguate:

- FOO :< (all hasColor GREEN) "GREEN is a set of values
- FOO :< (some hasColor GREEN)
- FOO :< (fills hasColor GREEN) "GREEN is a value"
- (some hasColor) :< FOO

Clarifying Is-A

- PERSON :< (the age INT) (the gender (oneof M F)) (the wt INT)
- 49'ER :< (fills age 49) (fills gender M) (the wt INT)

  49'ER is not a subconcept of PERSON!

  Did you mean (and PERSON (atmost 1 age) ...)?
- MAN = PERSON (the gender (oneof M)) (the wt INT)
- Mahdi : PERSON (fills age 49)(fills gender M)(fills wt 145)

  (the wife PERSON)

The Arch Example

roles: subpart , lintel, upright

lintel :< subpart, upright :<subpart

PHYSICAL-OBJECT :< THING

BLOCK :< PHYSICAL-OBJECT

COMPOSITE-OBJECT $=_{def}$
  (and PHYSICAL-OBJECT (at-least 1 subpart)
? :< ? =def ?   (all subpart PHYSICAL-OBJECT)

ARCH $=_{def}$
  (and COMPOSITE-OBJECT (the lintel BLOCK)
    (all upright BLOCK) (at-least 2 upright)
    (all materials (one-of Marble Brick Steel)))

(domain lintel) :< ARCH

UML in SHIQ

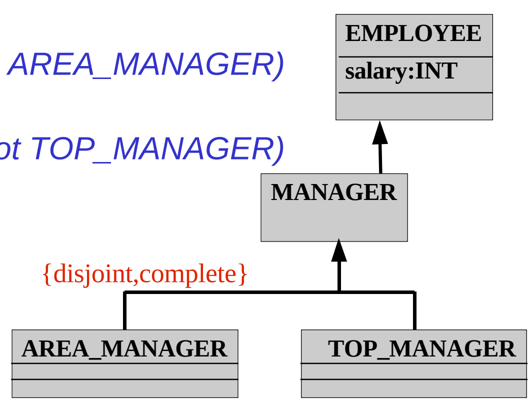
MANAGER :< EMPLOYEE, TOP_MANAGER :< MANAGER

AREA_MANAGER :< MANAGER

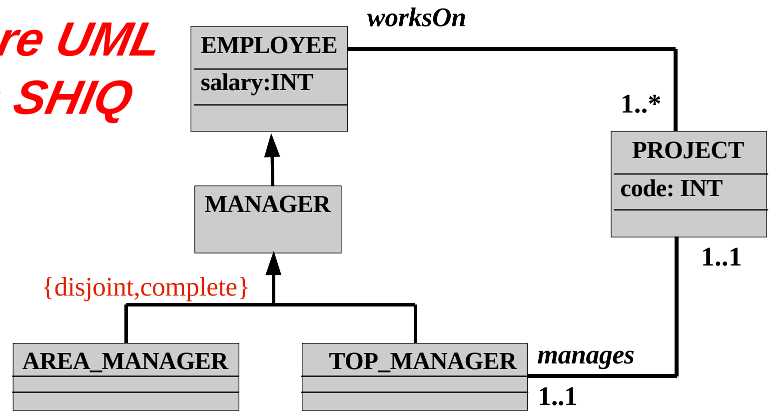
MANAGER :< (or TOP_MANAGER AREA_MANAGER)
  //complete

TOP_MANAGER :< (not AREA_MANAGER)
  //disjoint

AREA_MANAGER :< (not TOP_MANAGER)
  //disjoint



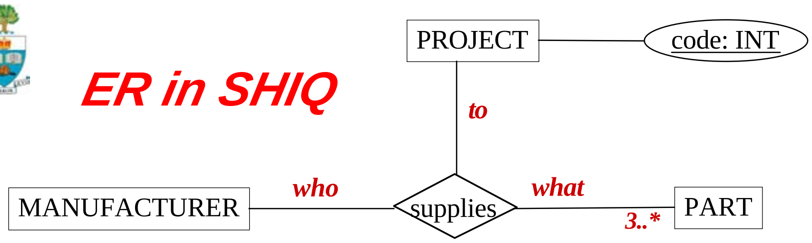
More UML in SHIQ



TOP_MANAGER :< (all manages PROJECT) (exactly 1 manages)

PROJECT :< (all (inverse manages) TOP_MANAGER) (exactly 1 (inverse manages))

ER in SHIQ



Relationships are "reified" into concepts:

- ✓ *SUPPLIES :< (the who MANUF) (the what PART) (the to PROJECT)*
- ✓ *MANUF :< (all (inverse who) SUPPLIES)*
- ✓ *PROJECT :< (all (inverse to) SUPPLIES)*
- ✓ *PART :< (and (all (inverse what) PART) (at-least 5 (inverse what)))*
- ✓ *PROJECT :< (and (the code INT) (all code (at-most 1 (inverse code))) // key constraint*

Modeling CORBA Services

```
interface CAR{
 attrib CAR-MODEL model;
 attrib OWNER ownedBy;
 attrib MANUFACT madeBy;
 ...
 deliver( in MANUFACT src,
     in DEALER dest,
     in DATE time) signals
  (BadDealer);
 sell(...);
 destroy(...);
```

```
DELIVER :< (and ACTION
     (the this CAR)
     (the src  MANUFACT)
     (the dest DEALER)
     (the time DATE)
```


Modeling a CORBA Interface

- CAR :<
- (**the** model CAR_MODELS)
- (**the** ownedBY OWNER)
- (**the** madeBy MANUFACT)
  - (**the** deliver DELIVER)
  - *//preconds include*
  - (**same-as** madeBy (deliver src))
  - *//postconds include*
  - (**same-as** ownedBy (deliver dest))
  - *//exception BadDealer signalled when*
  - (**not** (**overlaps** src (dest represents)))

Medical Ontologies

- Check out

  saussure.irmkant.rm.cnr.it/onto/

for a philosophically well-thought out and detailed medical ontology

Some Complexity Results

Constructors	T box			Subsumes?	Member?
	(prim :< D)	(D :< C)	cyclic		
AL (and, `all`)	-	-	-	O(n^2)	
AL	+	-	-	co-NP-complete	
CLASSIC with host individuals	+	-	-	O(n^3)	
ALE(and,all,some)				NP-compl.	PSPACE
ALC (and,all,not)	-			PSPACE-complete	
ALC (and,all,not)	+	+		EXPTIME-complete	
ALCNR(r-and,nrs)	-			PSPACE	PSPACE
ALCNR,SHIQ	+	+	+	NEXPTIME	NEXPTIME
muALCQ, ALCN+complex roles but not `r-and`				EXPTIME-complete	
AL & role `same-as`	-	-	-	undecidable	
AL & attrib. `same-as`			+	undecidable	

Implementation Strategies

- Translate to other logics, use existing theorem provers.
- Normalize and compare approach: Find normal form which makes *explicit* facts implied by a description

  e.g., atmost(0,p) ==> atmost(0,p) & all(p,NOTHING)

  - ✓ usually relatively fast; used in most widely distributed systems (Classic, Loom, Back)
  - ✓ often incomplete; has problems with disjunction, case-by-case reasoning, reasoning by contradiction

Implementation Strategies

- Tableaux-like calculus: show C :< D by showing (and C (not D)) inconsistent;
  - ✓ Prove this by *contradiction*: try to construct an individual object that can be an instance of (and C (not D))
  - ✓ Uses "completion rules", e.g., if you have { y:all(P,C), P(y,w)} then add {w:C};
  - ✓ Usually complete, so termination of the algorithm is the big issue.