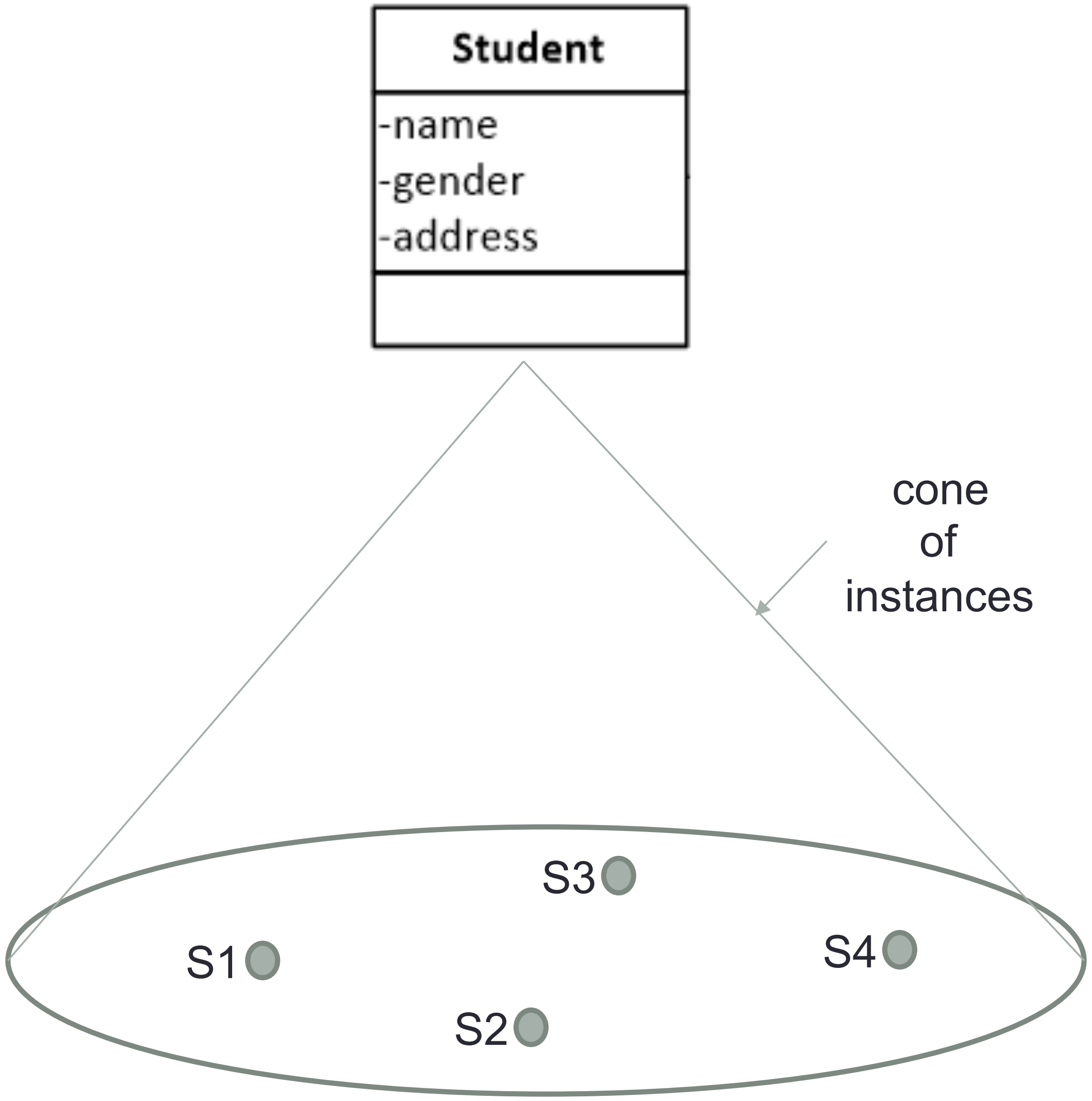
CLASS DIAGRAM EQUIVALENCE

Judy Altoyan
Don Batory

Background

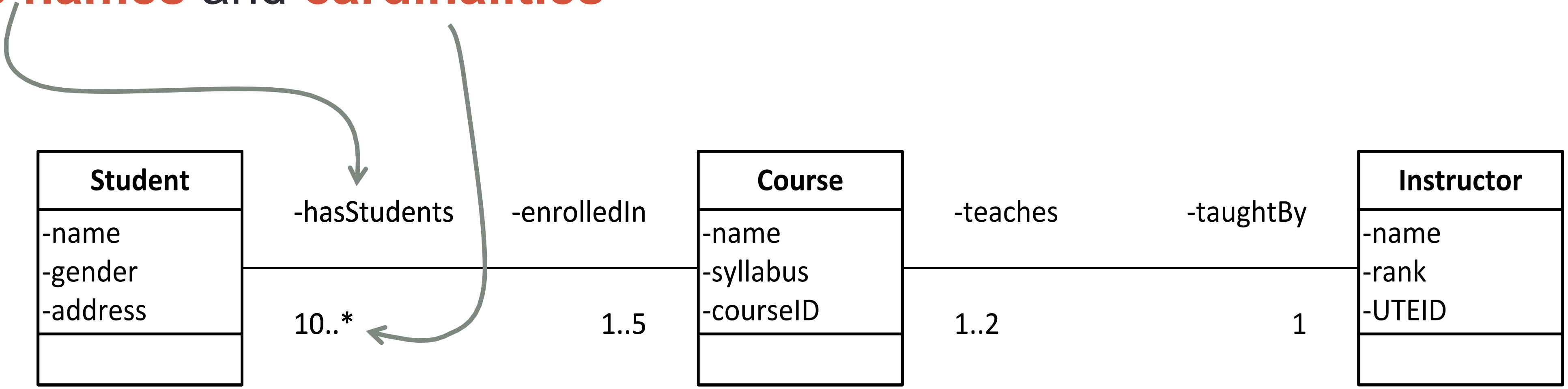
- A **class diagram (CD)** is a standard graphical notation to depict object oriented designs in terms of classes and their relationships

- A **class** defines a "type" which has instances



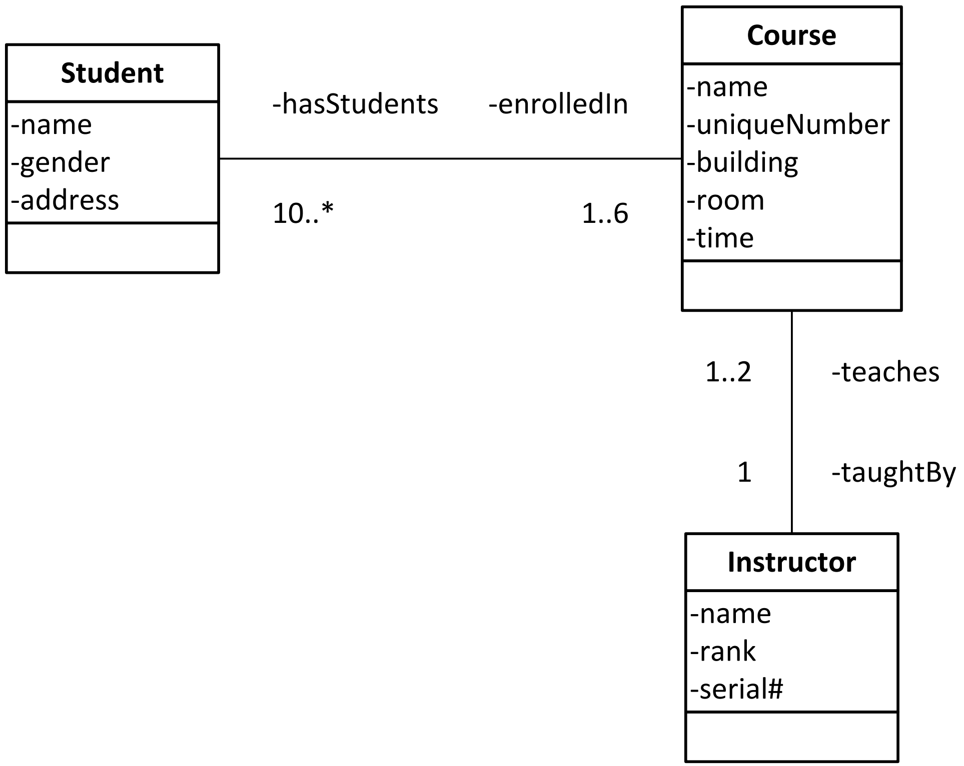
Background

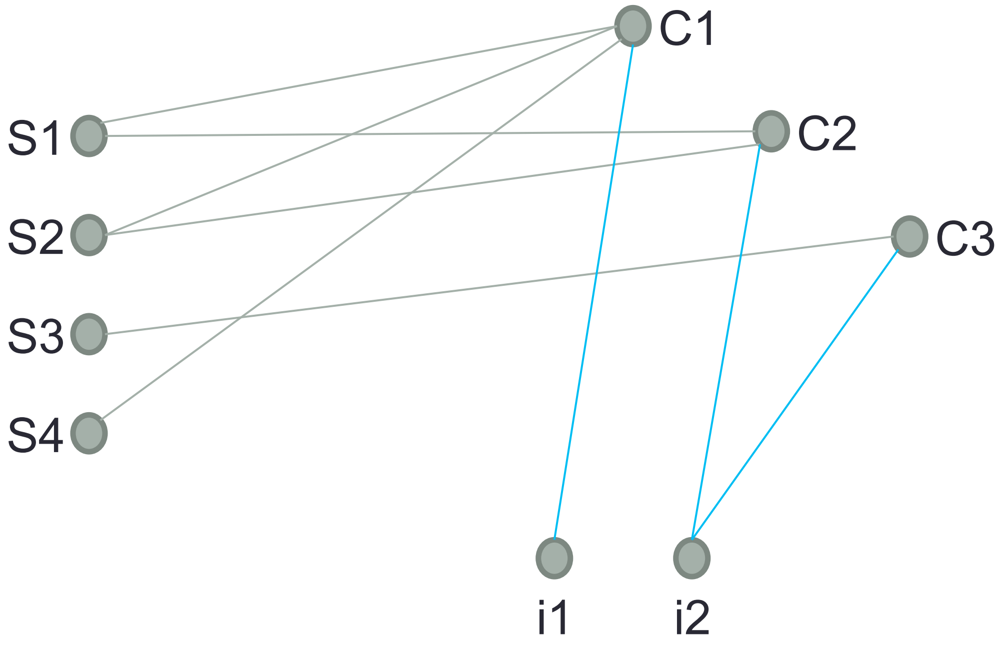
- An object oriented design usually has many classes
- Classes has relationships called **associations** that have **role names** and **cardinalities**



Background

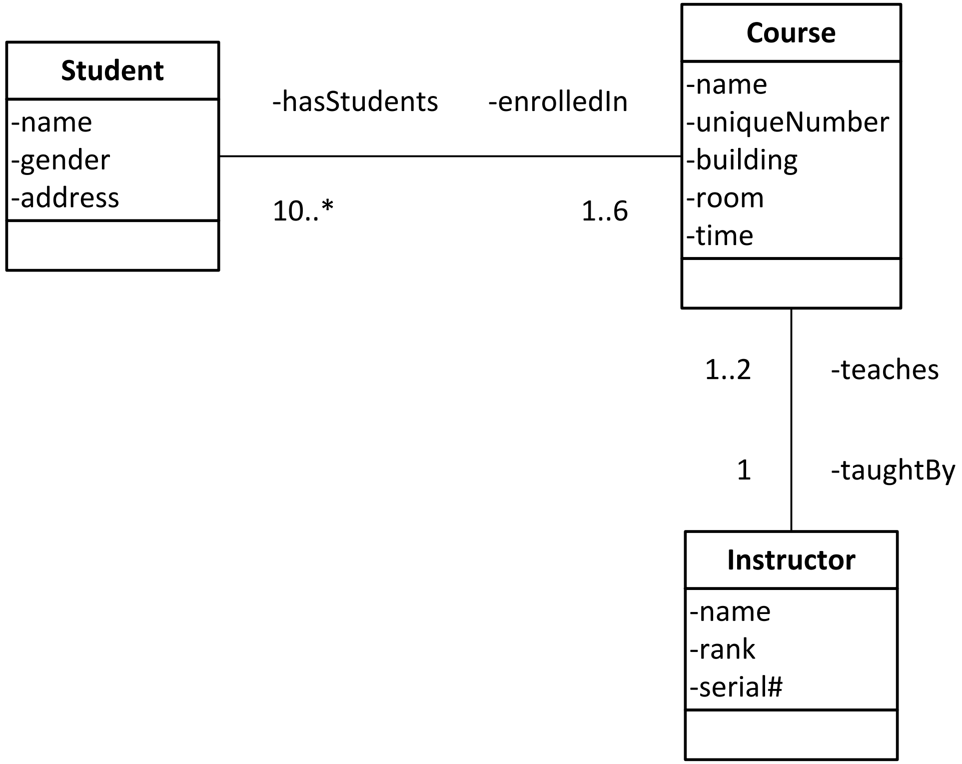
- A class diagram has instances – here is one of a colossal number of instances

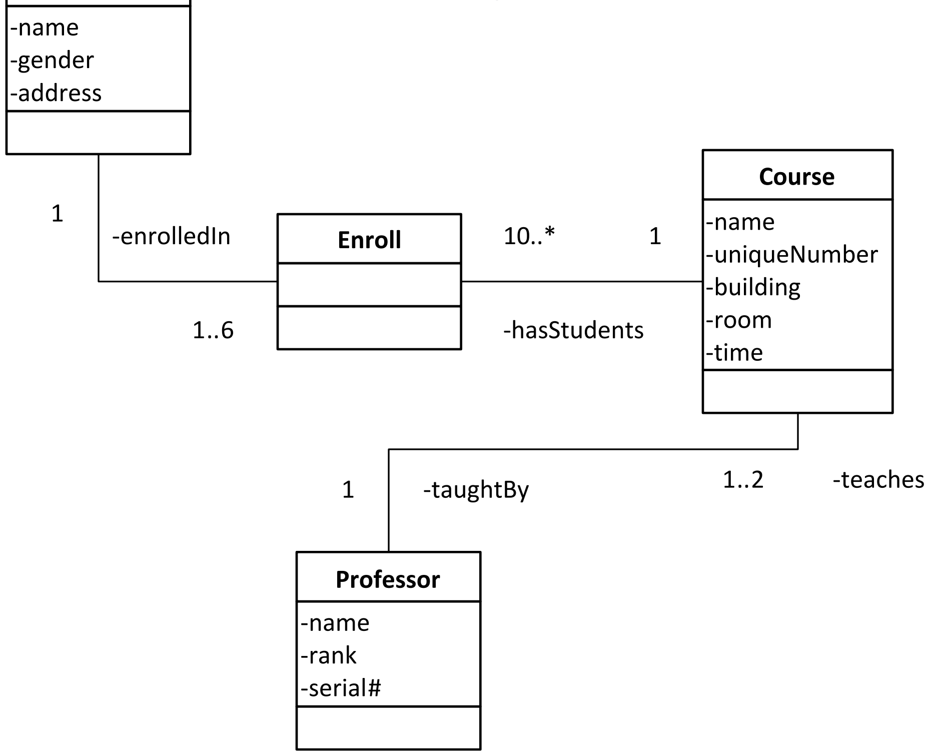




Basic Question: Equivalence

- Do two class diagrams encode the same information?
- If so, we say they **refactorings** of each other – show by applying a series of equality rewrites

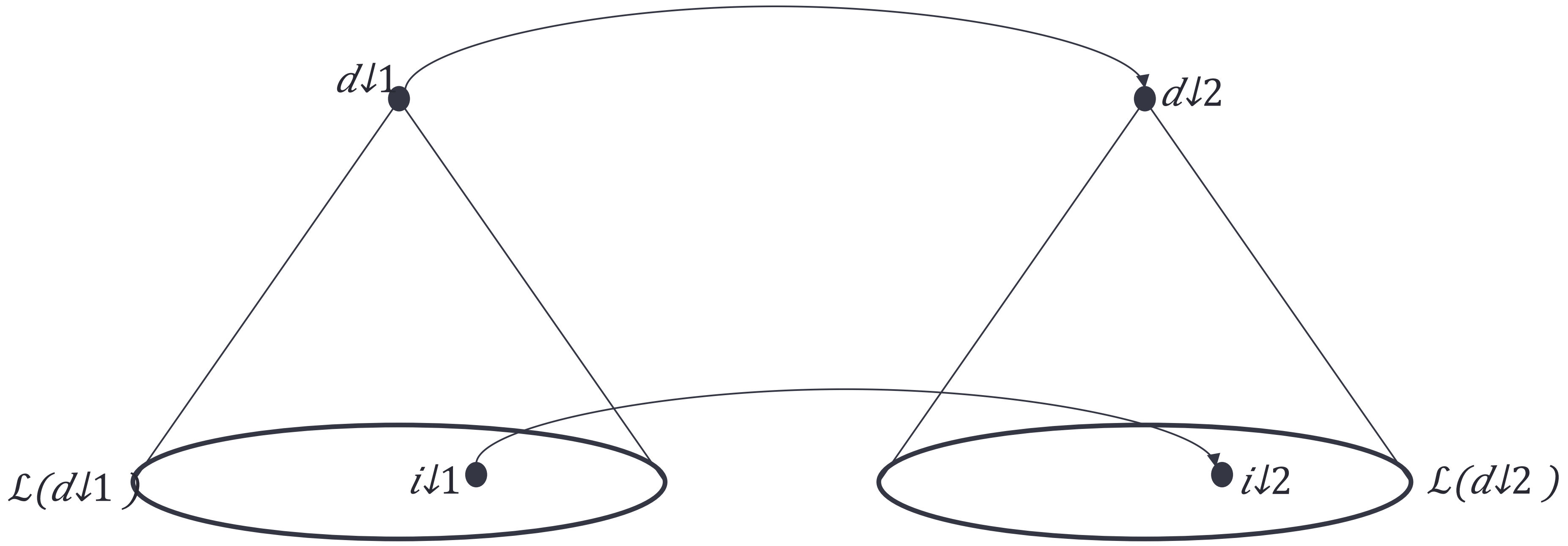




Foundation for Proving Equivalence uses CD Transformations / Rewrites

- A class diagram d_1 is a mapping or embedding into another class diagram d_2 , $\mathcal{T}(d_1)=d_2$ such that:

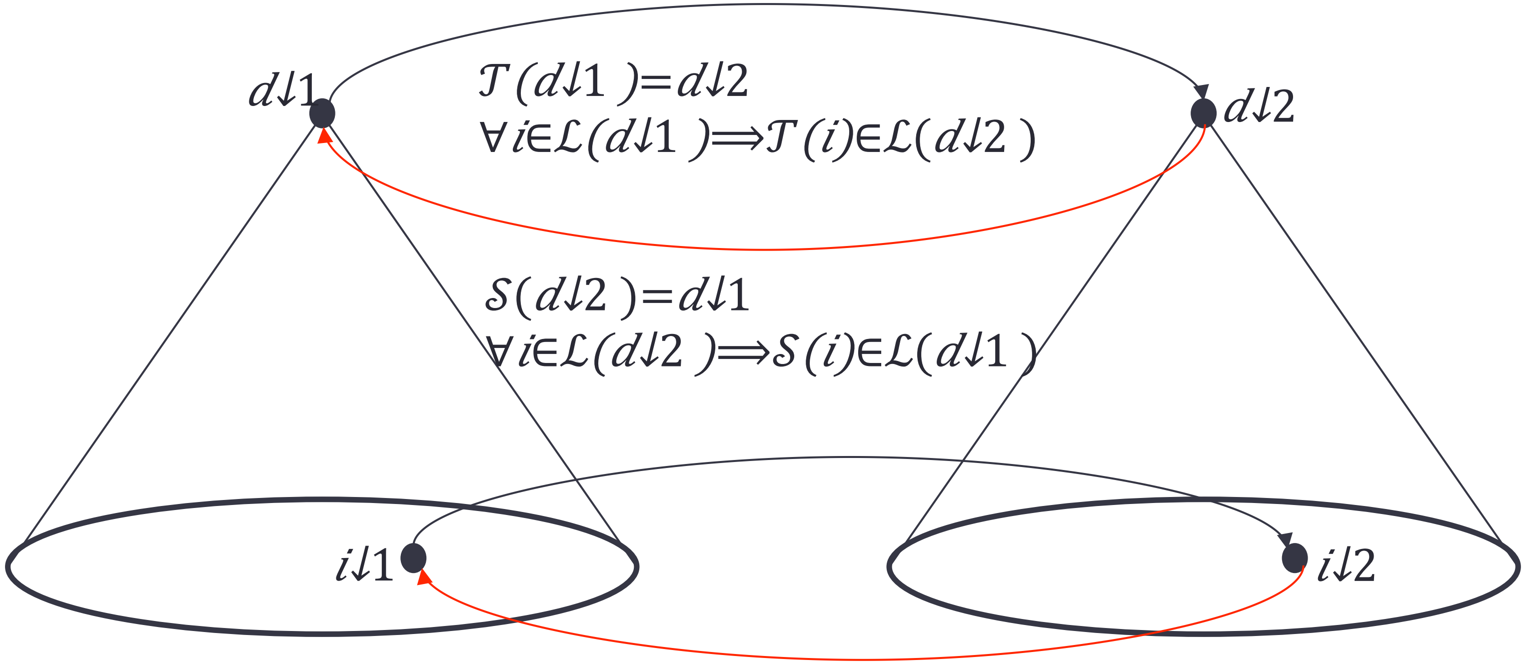
$$\forall i \in \mathcal{L}(d_1) \Longrightarrow \mathcal{T}(d_1) \in \mathcal{L}(d_2)$$



To Prove a CD Refactoring

Of a class diagram d_1 to class diagram d_2 requires transformation $\mathcal{T}$ to be invertible:

$$\mathcal{T}^{-1} \cdot \mathcal{T}(x) = x \textbf{ and } \mathcal{T} \cdot \mathcal{T}^{-1}(y) = y$$



Where $\mathcal{S} = \mathcal{T}^{-1}$

In General the Information in a Class Diagram is

- **Classes** – with their scalar-valued fields, domains of objects
- **Associations** among pairs of classes + role names (turn in to set-valued fields)
- **Cardinalities** – how many objects of class T are connect to objects of class R
- Can be **additional constraints**

    all courses have unique course numbers
    no two students have the same name and postal address
    …

Our Immediate Goal

- Given two class diagrams (CDs), how do we prove they are equivalent?
- This much we know & need:
  1. We need a formal representation of a CD
  2. And a mapping/correspondence between CDs must be defined
  - Should be able to prove disprove equivalence
- Mechanize what we are doing by hand now…

So What? (Always a Good Question To Ask)

- Not possible to verify refactorings in commercial languages – Java
  - no formal model of Java exists, only tiny versions (Featherweight Java)
- Class diagrams are as close as likely anyone can get now
  - is still a fundamental open problem in MDE ~15+ years old, UML $\geq$ 20 years
- Fundamental problem:
  - CD transformations (that's what MDE is all about)
  - database to database transformations (that's what database migration is all about)
- It is high time to make progress

Formal Notations That Have Been Used

- First-order-logic: predicate logic with quantifiers over variables.
- Description logic (DL): decidable fragments of first-order logic
  - define sets, subset relationships, cross-products, cardinality constraints
- Relational Algebra: includes projection, join, etc. on database tables
  - CDs represent database schemas
  - Mappings represent database translations
- Formal specification languages: Alloy, Z notation, Object-Z, Coq

Example in FOL notation

- Classes are unary predicates:

  $Course(x)$

  $Instructor(x)$

- Associations are binary predicates:

  $\forall x,y. teaches(x,y) \rightarrow Instructor(x) \wedge Course(y)$

- Attributes are binary predicates:

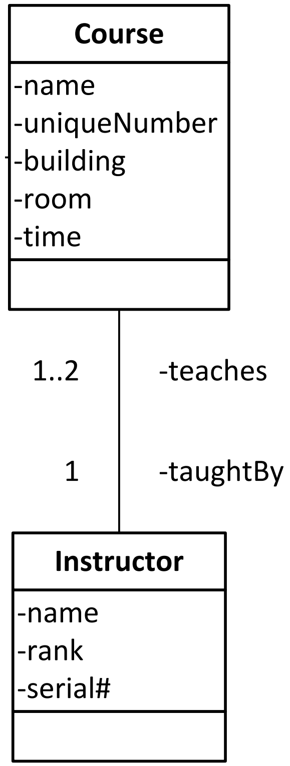
  $\forall x,y. rank(x,y) \rightarrow Instructor(x) \wedge String(y)$

- Cardinalities are constraints:

  $(\forall x. Course(x) \rightarrow \exists y. taughtBy(x,y)) \wedge$

  $(\forall x,y,y' . taughtBy(x,y) \wedge taughtBy(x,y') \rightarrow y=y')$

need set cardinality primitives



Proof Tools

Problem: maturity and dependability of tools. Most student-produced tools aren't very good.

- DL reasoners: different reasoner for each DL notation. Seem flakey….
  - E.g. FaCT++, Pellet, Racer, etc.
- Proof assistants: require user interaction.
  - E.g. PVS, Isabell, **Coq**, etc.
- Theorem provers: fully automated.
  - E.g. **ACL2**, **Prover9**, Vampire, SPASS, etc.
- SAT solvers
  - E.g. **Alloy**

Prior Work

- Most work on class diagram analysis is to prove that it is **satisfiable** – it has at least one instance
  - Description logic reasoners were used to detect unsatisfiable concepts (i.e. classes that cannot be instantiated).
- Alloy was used to formally represent CDs and analyze them for inconsistences. However, since Alloy only permits bounded analysis scope, it cannot be used as a theorem prover.
- One work on Alloy Model equivalence used PVS where an equivalence notion was defined. CD equivalence can then be derived by translating to corresponding Alloy models.

We Need Advice…

- What tool is most suited for proving CD equivalence?
  - ideally it directly supports the concepts that we need to express class diagrams
- What is the ramp time to learn a tool?
- Anyone here (or that you know of) have interest in this problem?

Thank Ewe!

شكر
ا