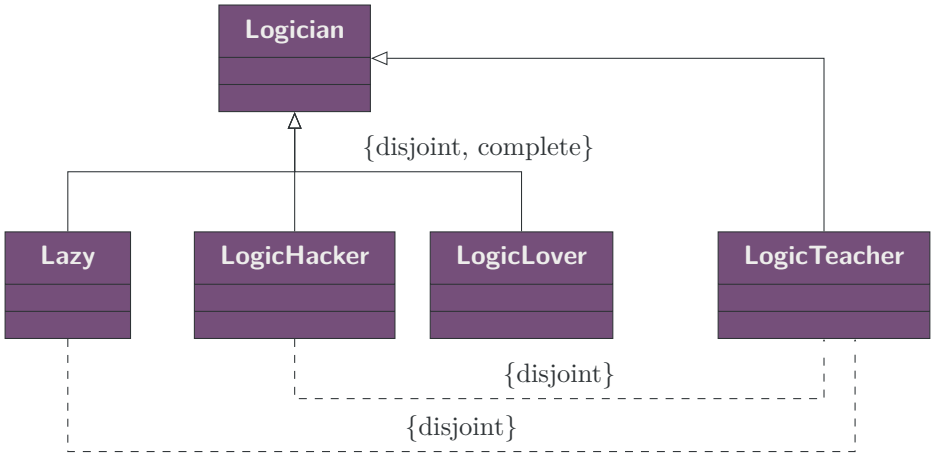
CVFP (Software Design and Formal Verification)
TD 2: UML and Propositional Logic

We've seen last week that every UML diagram can be translated in first-order logic. Let's focus on a particular subclass of UML diagrams to only fall on propositional logic.

Exercise 1 `What kind of logician are you?`

1.1. Translate the following classes diagram into propositional logic.



1.2. Why don't we need the full power of the first-order logic in this example?

Exercise 2 `It's actually difficult to know whether you shall get some points when you draw an UML classes diagram…`
We recall that the satisfiability problem (SAT) is to decide, for a given formula, if there exists a valuation of the propositional variables that makes the formula true. This problem is known to be NP-complete.

Recall. The NP class is the class of every problem decidable with a non-deterministical Turing machine in a polynomial time. Another way to perceive NP problems is to consider them as the problem solvable by "guessing" a (polynomial-sized) solution, then checking (in polynomial time) with a deterministic algorithm that this "solution" is effectively one.
A NP-hard problem is a problem at least at hard as any NP problem.

A NP-complete problem is a problem both NP and NP-hard.

Let's call $\mathcal{C}_{\text{UML}prop}$ the set of UML classes diagram only using inheritance, "disjoint" and "complete" relations.

A relevant question on an UML classes diagram is to know whether it is consistant, that is to know whether every class can contain at least one object. We're going to show that checking a consistance of a classes diagram in $\mathcal{C}_{\text{UML}prop}$ is NP-complete.

2.1. Show that the decision problem of whether a classes diagram of $\mathcal{C}_{\text{UML}prop}$ is consistant is NP.

For the SAT problem, we'll consider that the input formula is given in conjonctive normal form: it's a conjonction $C_1 \wedge \ldots \wedge C_n$ of clauses of the form $C_i = l_{i,1} \vee \ldots \vee l_{i,k}$, with $l_{i,k} = p$ or $l_{i,k} = \neg p$ for a given variable p.

2.2. Show by reducing into SAT that deciding whether a clauses diagram of $\mathcal{C}_{\text{UML}prop}$ is consistant is an NP-hard problem.

Note. Reducing a NP-hard problem $\mathcal{A}$ into another problem $\mathcal{B}$ is a way to show that $\mathcal{B}$ is NP-hard. It consists in describing[1] how to time-polynomially transform any instance of $\mathcal{A}$ into an equivalent instance of $\mathcal{B}$. Thus $\mathcal{B}$ is at least as hard as $\mathcal{A}$: if we ever find a time-polynomial algorithm solving $\mathcal{B}$, we would have one solving $\mathcal{A}$ for free.

[1] And proving that solving the reduced instance is effectively equivalent to solving the initial one.