# First Order Logic

1. A mechanic likes Bob.

## 1. A mechanic likes Bob.

Predicates:

## 1. A mechanic likes Bob.

Predicates:

- mechanic(X)

## 1. A mechanic likes Bob.

Predicates:

- mechanic(X)
- likes(X, Y)

# 1. A mechanic likes Bob.

Predicates:

- mechanic(X)
- likes(X, Y)

Solution:

$\exists X(\text{mechanic}(X) \land \text{likes}(X, \text{Bob}))$

## 2. A mechanic likes herself.

## 2. A mechanic likes herself.

Predicates:

- mechanic(X)
- likes(X, Y)

## 2. A mechanic likes herself.

Predicates:

- mechanic(X)
- likes(X, Y)

Solution:

$\exists X(\text{mechanic}(X) \land \text{likes}(X, X))$

## 3. Every mechanic likes Bob.

## 3. Every mechanic likes Bob.

Predicates:

- mechanic(X)
- likes(X, Y)

## 3. Every mechanic likes Bob.

Predicates:

- mechanic(X)
- likes(X, Y)

Solution:

$\forall X(\text{mechanic}(X) \rightarrow \text{likes}(X, \text{Bob}))$

## 4. Some mechanic likes every nurse.

## 4. Some mechanic likes every nurse.

Predicates:

- mechanic(X)
- likes(X, Y)
- nurse(X)

## 4. Some mechanic likes every nurse.

Predicates:

- mechanic(X)
- likes(X, Y)
- nurse(X)

Solution:

$\exists X(\text{mechanic}(X) \land \forall Y(\text{nurse}(Y) \rightarrow \text{likes}(X, Y)))$

## 5. There is a mechanic who is liked by every nurse

## 5. There is a mechanic who is liked by every nurse

Predicates:

- mechanic(X)
- likes(X, Y)
- nurse(X)

## 5. There is a mechanic who is liked by every nurse

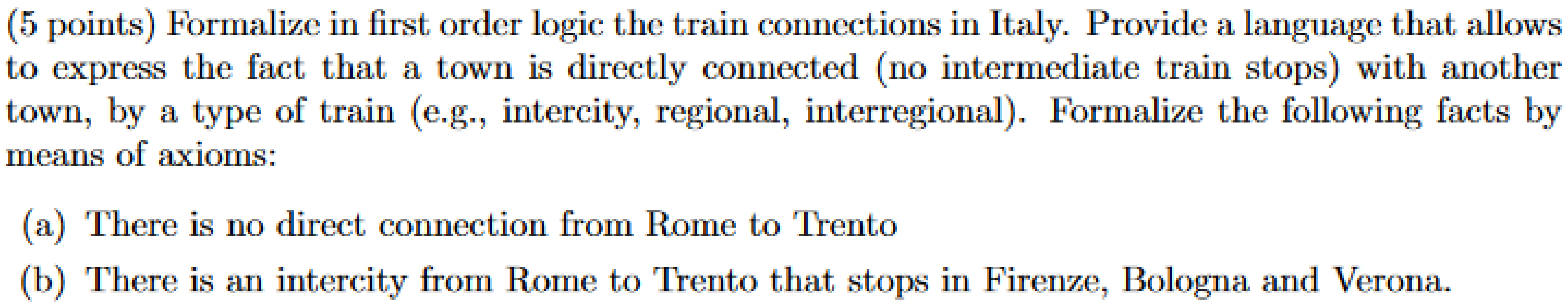
Predicates:

- mechanic(X)
- likes(X, Y)
- nurse(X)

Solution:

$\exists X(\text{mechanic}(X) \land \forall Y(\text{nurse}(Y) \rightarrow \text{likes}(Y, X)))$

## From an old exam…



(5 points) Formalize in first order logic the train connections in Italy. Provide a language that allows to express the fact that a town is directly connected (no intermediate train stops) with another town, by a type of train (e.g., intercity, regional, interregional). Formalize the following facts by means of axioms:

(a) There is no direct connection from Rome to Trento

(b) There is an intercity from Rome to Trento that stops in Firenze, Bologna and Verona.

## a) There is no direct connection form Rome to Trento

Predicates:

## a) There is no direct connection form Rome to Trento

Predicates:

- DirectConn(X, Y, Z)

*There exists a train X which goes directly from Y to Z.*

## a) There is no direct connection form Rome to Trento

Predicates:

- DirectConn(X, Y, Z)

*There exists a train X which goes directly from Y to Z.*

Solution:

¬∃DirectConn(X, Rome, Trento)

## b) There is an intercity from Rome to Trento that stops in Firenze, Bologna and Verona.

Predicates:

## b) There is an intercity from Rome to Trento that stops in Firenze, Bologna and Verona.

Predicates:

- DirectConn(X, Y, Z)
- TrainType(X, Y)

## b) There is an intercity from Rome to Trento that stops in Firenze, Bologna and Verona.

Predicates:

- DirectConn(X, Y, Z)
- TrainType(X, Y)

Solution:

∃X.(DirectConn(X, Rome, Firenze) ∧ DirectConn(x, Firenze, Bologna) ∧ DirectConn(x, Bologna, Verona) ∧ DirectConn(X, Verona, Trento) ∧ TrainType(X, interCity))

# Logic Programming

A brief introduction

## Logic Programming

- A program is a set of formulas (more precisely, *clauses*).

## Logic Programming

- A program is a set of formulas (more precisely, *clauses*).
- A query for a program (or *goal*) is also a clause.

## Logic Programming

- A program is a set of formulas (more precisely, *clauses*).
- A query for a program (or *goal*) is also a clause.
- Computation is the same as logic deduction (resolution).

## Logic Programming

- A program is a set of formulas (more precisely, *clauses*).
- A query for a program (or *goal*) is also a clause.
- Computation is the same as logic deduction (resolution).
- Logic programming languages are *declarative*: the programmer specifies the *what*, while the machine takes care of the *how*.

# Prolog

The basics

## (A very basic) Prolog crash course

- A Prolog program is a set of *rules* expressed in the following notation

  *head :- body.*

# (A very basic) Prolog crash course

- A Prolog program is a set of *rules* expressed in the following notation

*head :- body.*

- If the body is true, then the head is also true

# (A very basic) Prolog crash course

- A Prolog program is a set of *rules* expressed in the following notation

```
head :- body.
```

- If the body is true, then the head is also true

- If the body is not present, it is considered true. In this case the head is also true, and it is called a fact.

```
cat(garfield).
```

# (A very basic) Prolog crash course

- The body is a conjunction of terms separated by commas

```
p(a, 12, X) :- q(a), r(13), s(X, 1).
```

# (A very basic) Prolog crash course

- The body is a conjunction of terms separated by commas

```
p(a, 12, X) :- q(a), r(13), s(X, 1).
```

- Both the head and the body can contain *variables*, which start with a capital letter:

```
p(a, 12, X) :- q(a), r(13), s(X, 1).
s(Y, X) :- q(X), r(Y).
```

# (A very basic) Prolog crash course

- The body is a conjunction of terms separated by commas

```
p(a, 12, X) :- q(a), r(13), s(X, 1).
```

- Both the head and the body can contain *variables*, which start with a capital letter:

```
p(a, 12, X) :- q(a), r(13), s(X, 1).
s(Y, X) :- q(X), r(Y).
```

- The scope of the variables is the single rule. The X in the first rule is not related to the X in the second one.

## SWI Prolog

A comprehensive free Prolog environment:

https://www.swi-prolog.org/

It also offers an online platform where you can create and test your programs:

https://swish.swi-prolog.org/