Prolog - Lecture 1

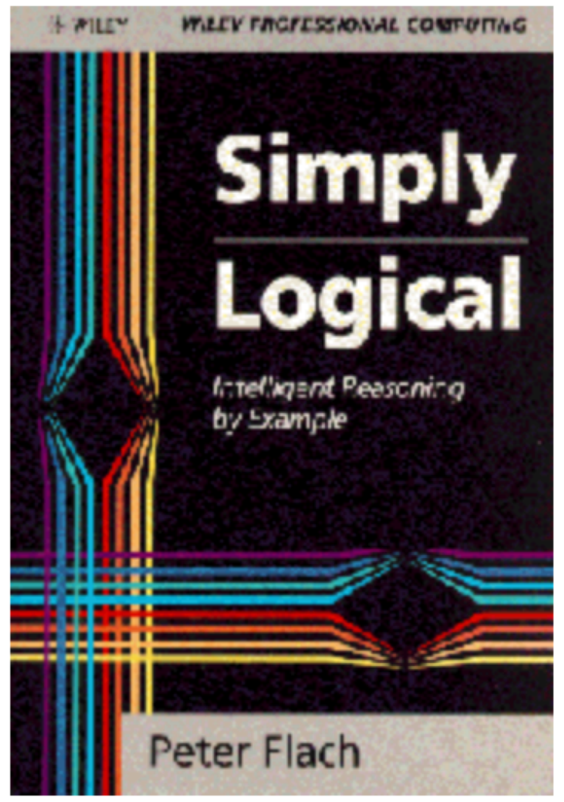
(Using slides from Peter Flach’s lectures for his book Simply Logical)

Free from Peter Flach: http://people.cs.bris.ac.uk/~flach/SimplyLogical.html

Simply Logical

Intelligent Reasoning by Example

by Peter Flach, then at Tilburg University, the Netherlands
John Wiley 1994, xvi + 240 pages, ISBN 0471 94152 2
Reprinted: December 1994, July 1998.



This book is no longer available through John Wiley publishers. You can download a free PDF copy here.
The PDF copy has a small number of discrepancies with the print version, including

- different page numbers from Part III (p.129)
- certain mathematical symbols are not displayed correctly, including
  - $\vdash$ displayed as |
  - $\nvdash$ displayed as |;/
  - $\models$ displayed as =
  - $\nvDash$ displayed as =;/
- the index is currently missing

I am working on fixing these.

- Table of Contents
- Foreword by Bob Kowalski
- Author's Preface
- On-line Prolog programs from the book:
  - compressed tar archive (Unix, 38K)
  - BinHex archive (Macintosh, 149K)
  - plain text files
- Teaching materials:
  - colour overhead transparencies (PowerPoint, HTML, PDF, PostScript)
  - lab exercises

Propositional Programs

Terminology and Setting (1)

- A **literal** is an atom or its negation (e.g., p, $\neg q$ are literals).

Terminology and Setting (1)

- A **literal** is an atom or its negation (e.g., p, $\neg q$ are literals).
- A **clause** is a disjunction of literals (e.g., $p \vee r \vee \neg q$ is a clause).

Terminology and Setting (1)

- A **literal** is an atom or its negation (e.g., $p, \neg q$ are literals).
- A **clause** is a disjunction of literals (e.g., $p \vee r \vee \neg q$ is a clause).
- We will mostly restrict our attention to formulas which are conjunctions of clauses, which we will also represent as sets of clauses.

Terminology and Setting (1)

- A **literal** is an atom or its negation (e.g., $p, \neg q$ are literals).
- A **clause** is a disjunction of literals (e.g., $p \vee r \vee \neg q$ is a clause).
- We will mostly restrict our attention to formulas which are conjunctions of clauses, which we will also represent as sets of clauses.
- A **Horn clause** is a clause with *at most one* positive literal (e.g., $p \vee \neg q \vee \neg r$, $\neg p$, $\neg p \vee \neg r$ are Horn clauses).

Terminology and Setting (1)

- A **literal** is an atom or its negation (e.g., p, $\neg q$ are literals).
- A **clause** is a disjunction of literals (e.g., $p \vee r \vee \neg q$ is a clause).
- We will mostly restrict our attention to formulas which are conjunctions of clauses, which we will also represent as sets of clauses.
- A **Horn clause** is a clause with *at most one* positive literal (e.g., $p \vee \neg q \vee \neg r$, $\neg p$, $\neg p \vee \neg r$ are Horn clauses).
- A **definite clause** is a clause with *exactly one* positive literal (e.g., $p \vee \neg q \vee \neg r$, p are definite clauses).

Terminology and Setting (2)

- A definite clause $h \vee \neg b_1 \vee \neg b_2 \vee \ldots \vee \neg b_m$ can be written also as $h \Leftarrow b_1 \wedge b_2 \wedge \ldots \wedge b_m$. Therefore we will also call definite clauses **rules.**

- A set of definite clauses will be called a **definite program** and we will also treat it, with a slight abuse of notation, as a conjunction of the clauses.

Terminology and Setting (3)

- An **interpretation** will be represented as a set of atoms which are true in it (e.g., $\{p, q\}$)

- … since **models** are interpretations, likewise for models**.** That is, for instance, when $\varphi = (a \vee \neg b) \wedge b \wedge (c \vee \neg d)$, the models of φ would be represented as $\{a, b\}, \{a, b, c\}, \{a, b, c, d\}$.

What is true in all models…

Recall that $\varphi \models \alpha$ iff the formula α is true in all models of φ.

Example:

$\varphi = (a \Leftrightarrow (b \vee c)) \wedge (\neg b \vee \neg c) \wedge a$

The models of φ are $\{a, b\}, \{a, c\}$.

Although a is true in all models of φ, the set $\{a\}$ is not a model of φ… n*ot that we wanted or needed it to be, but stay with us!*

Definite Programs Are Nice!

Example:

Consider the definite program

$\mathscr{P} = \{a \Leftarrow b, b \Leftarrow c, b\}$.

The models of this program are: $\{a, b\}, \{a, b, c\}$.

Their intersection $\{a, b\}$ is a model of $\mathscr{P}$ too (it is one of the models above after all) — **This is not a coincidence. See next!**

Least Model

- **Proposition:** Let $\mathscr{M}$ be the set of all models of a given definite program $\mathscr{P}$. Let us define $\omega_{least} = \bigcap_{\omega \in \mathscr{M}} \omega$. Then ω_{least} is a model of $\mathscr{P}$ (and hence $\omega_{least} \in \mathscr{M}$). We call ω_{least} the least model of ω .

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{ h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega \right\}$.

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}$ and $b_1, \ldots, b_m \in \omega\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.

Constructing the Least Model

- **Definition (*T_P*-operator, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
    1. $\omega_0 = \emptyset$

Constructing the Least Model

- **Definition (*T_P*-operator, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \{h \mid h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.

- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have

  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.

Constructing the Least Model

- **Definition (*T_P*-operator, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.
  4. $\omega_3 = T_P(\omega_2) = \{a, b, c\}$.

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\varnothing)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \varnothing$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.
  4. $\omega_3 = T_P(\omega_2) = \{a, b, c\}$.
  5. $\omega_4 = T_P(\omega_3) = \{a, b, c\} = \omega_3$. We have reached fix-point, ω_3 is the least model of $\mathscr{P}$.

Constructing the Least Model

- **Definition (*T_P-operator*, aka *immediate consequence operator*):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as $T_P(\omega) = \left\{h \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P} \text{ and } b_1, \ldots, b_m \in \omega\right\}$.

- **Proposition:** The least model of $\mathscr{P}$ is the least fix-point of the sequence $T_P(T_P(\ldots T_P(\emptyset)))$.
- **Example:** Let $\mathscr{P} = \{a, b \Leftarrow a, c \Leftarrow a \wedge b\}$. We have
  1. $\omega_0 = \emptyset$
  2. $\omega_1 = T_P(\omega_0) = \{a\}$.
  3. $\omega_2 = T_P(\omega_1) = \{a, b\}$.
  4. $\omega_3 = T_P(\omega_2) = \{a, b, c\}$.
  5. $\omega_4 = T_P(\omega_3) = \{a, b, c\} = \omega_3$. We have reached fix-point, ω_3 is the least model of $\mathscr{P}$.

Least Model (Recap)

- A definite program $\mathscr{P}$ always has a least model.
- The least model can be found using the immediate consequence operator. This is also sometimes called *forward-chaining.*
- Definite programs cannot entail negative literals—therefore the least model tells us everything we need to know about the program and what follows from it (do you see why?)

First-Order Programs

Setting, Notation and Terminology (1)

- Now we will upgrade definite programs from propositional to first-order.
  You have already seen first-order logic in the first part of the course.

Setting, Notation and Terminology (1)

- Now we will upgrade definite programs from propositional to first-order. *You have already seen first-order logic in the first part of the course.*

- **Convention:** Variables in Prolog start with a capital letter (e.g. V), constants with a lower-case letter (e.g. carrot).

Setting, Notation and Terminology (1)

- Now we will upgrade definite programs from propositional to first-order.
You have already seen first-order logic in the first part of the course.

- **Convention:** Variables in Prolog start with a capital letter (e.g. V), constants with a lower-case letter (e.g. carrot).

- **Convention:** A definite clause $h \Leftarrow b_1 \wedge \ldots \wedge b_m$ will be written in Prolog notation as h :- $b_1, \ldots, b_m$. All variables that appear in a definite clause are automatically assumed to be universally quantified (recall the definition of clause).

Setting, Notation and Terminology (2)

- **Definition (Term):** A *term* is a constant (e.g. carrot), a variable (e.g. *V*) or a function applied to a tuple of terms (e.g. $g(\text{carrot}, V)$).

Setting, Notation and Terminology (2)

- **Definition (Term):** A *term* is a constant (e.g. carrot), a variable (e.g. *V*) or a function applied to a tuple of terms (e.g. $g(\text{carrot}, V)$).

- **Definition (Ground Term):** A term is *ground* if it does not contain variables—e.g. *carrot* is a ground term, but V and $g(\text{carrot}, V)$ are not ground.

Setting, Notation and Terminology (3)

- **Definition (Substitution):** A *substitution* is a mapping that maps variables to terms. For instance, $\vartheta = \{X \mapsto \text{maria}\}$ is a substitution.

Setting, Notation and Terminology (3)

- **Definition (Substitution):** A *substitution* is a mapping that maps variables to terms. For instance, $\vartheta = \{X \mapsto \textsf{maria}\}$ is a substitution.

- A substitution can be applied to a clause. **For a clause C and a substitution ϑ, this is denoted as $C\vartheta$.**

Setting, Notation and Terminology (3)

- **Definition (Substitution):** A *substitution* is a mapping that maps variables to terms. For instance, $\vartheta = \{X \mapsto \text{maria}\}$ is a substitution.

- A substitution can be applied to a clause. **For a clause C and a substitution ϑ, this is denoted as $C\vartheta$.**

- For instance, let us have the definite clause $\text{isStudentOf}(X, T)$:- $\text{teaches}(T, X)$. If we apply the substitution $\{X \mapsto \text{maria}\}$ to it, we get $\text{isStudentOf}(\text{maria}, T)$:- $\text{teaches}(T, \text{maria})$.

Setting, Notation and Terminology (3)

- **Definition (Substitution):** A *substitution* is a mapping that maps variables to terms. For instance, $\vartheta = \{X \mapsto \text{maria}\}$ is a substitution.

- A substitution can be applied to a clause. **For a clause C and a substitution ϑ, this is denoted as $C\vartheta$.**

- For instance, let us have the definite clause $\text{isStudentOf}(X, T)$:- $\text{teaches}(T, X)$. If we apply the substitution $\{X \mapsto \text{maria}\}$ to it, we get $\text{isStudentOf}(\text{maria}, T)$:- $\text{teaches}(T, \text{maria})$.

- The resulting clause is said to be an *instance* of the original clause, and a *ground instance* if it does not contain variables.

Setting, Notation and Terminology (3)

- **Definition (Substitution):** A *substitution* is a mapping that maps variables to terms. For instance, $\vartheta = \{X \mapsto \text{maria}\}$ is a substitution.

- A substitution can be applied to a clause. **For a clause C and a substitution ϑ, this is denoted as $C\vartheta$.**

- For instance, let us have the definite clause $\text{isStudentOf}(X, T)$:- $\text{teaches}(T, X)$. If we apply the substitution $\{X \mapsto \text{maria}\}$ to it, we get $\text{isStudentOf}(\text{maria}, T)$:- $\text{teaches}(T, \text{maria})$.

- The resulting clause is said to be an *instance* of the original clause, and a *ground instance* if it does not contain variables.

- **Each instance of a clause is among its logical consequences.**

Setting, Notation and Terminology (4)

- **Definition (Herbrand Universe):** Given a definite program $\mathscr{P}$, its *Herbrand universe* is the set of all ground terms that are either constants appearing in $\mathscr{P}$ or can be constructed from the constants and function symbols appearing in $\mathscr{P}$.

- **Example:**

  If $\mathscr{P} = \{\text{teacherOf(peter, maria) . isStudentOf}(X, T) \text{ :- teacherOf}(T, X) \text{ . }\}$ then the Herbrand universe of $\mathscr{P}$ is $\{\text{peter, maria}\}$.

- If $\mathscr{P} = \{\text{num}(0), \text{num}(\text{suc}(X)) \text{ :- num}(X) \text{ . }\}$ then the Herbrand universe is the infinite set $\{0, \text{ suc}(0), \text{suc}(\text{suc}(0)), \text{suc}(\text{suc}(\text{suc}(0))), \ldots\}$

Setting, Notation and Terminology (5)

- **Definition (Herbrand Base):** Given a definite program $\mathscr{P}$, its *Herbrand base* is the set of all ground atoms that can be constructed using the terms from the Herbrand universe of $\mathscr{P}$.

- **Example:**

  If $\mathscr{P} = \{$teacherOf(peter, maria) . isStudentOf(X, T) :- teacherOf(T, X) . $\}$
  then the Herbrand base of $\mathscr{P}$ is $\{$teacherOf(maria, maria), teacherOf(peter, peter), teacherOf(peter, maria), teacherOf(maria, peter), studentOf(peter, peter), studentOf(maria, maria), teacherOf(peter, maria), teacherOf(maria, peter)$\}$.

- If $\mathscr{P} = \{$num(0), num(suc(X)) :- num(X) . $\}$ then the Herbrand base is the infinite set $\{$num(0), num(suc(0)), num(suc(suc(0))), num(suc(suc(suc(0)))), $\ldots\}$

Remember:

Herbrand universe ~ ground **terms**

Herbrand base ~ ground **atoms**

Setting, Notation and Terminology (6)

- **Definition (Herbrand Interpretation and Herbrand Model):** Given a definite program $\mathscr{P}$, let $\mathscr{B}$ be its Herbrand base. A *Herbrand interpretation* is a subset of $\mathscr{B}$. A *Herbrand model* of $\mathscr{P}$ is a Hebrand interpretation which is also a model of $\mathscr{P}$.

- **Definition (Least Herbrand Model):** Given a definite program $\mathscr{P}$, its *least Herbrand model* (*LHM*) is the intersection of all of its models.

- **Example:**

  If $\mathscr{P} = \left\{\text{teacherOf(peter, maria)}\ .\ \text{isStudentOf}(X, T)\ \text{:-}\ \text{teacherOf}(T, X)\ .\ \right\}$
  then the least Herbrand model of $\mathscr{P}$ is $\{\text{teacherOf(peter, maria), studentOf(maria, peter)}\}$.

- If $\mathscr{P} = \{\text{num}(0), \text{num}(\text{suc}(X)\ \text{:-}\ \text{num}(X)\ .\ \}$ then the Herbrand base is the infinite set $\{\text{num}(0),\ \text{num}(\text{suc}(0)), \text{num}(\text{suc}(\text{suc}(0))), \text{num}(\text{suc}(\text{suc}(\text{suc}(0)))), \ldots\}$, which turns out to be the same as the Herbrand base in this case.

Constructing the LHM

- **Definition (T_P-operator, aka *immediate consequence operator* for LHB):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as
$T_P(\omega) = \left\{ h\vartheta \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}, \vartheta \text{ is a grounding substitution and } (b_1, \ldots, b_m)\vartheta \in \omega \right\}.$

Constructing the LHM

- **Definition (T_P-operator, aka *immediate consequence operator* for LHB):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as
$T_P(\omega) = \left\{ h\vartheta \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}, \vartheta \text{ is a grounding substitution and } (b_1, \ldots, b_m)\vartheta \in \omega \right\}.$
- **Example:**
$\mathscr{P} = \left\{ \text{teacherOf(peter, maria)} \,.\, \text{isStudentOf}(X, T) \text{ :- } \text{teacherOf}(T, X) \,. \right\}$

Constructing the LHM

- **Definition (*T_P*-operator, aka *immediate consequence operator* for LHB):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as
  $T_P(\omega) = \left\{ h\vartheta \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}, \vartheta \text{ is a grounding substitution and } (b_1, \ldots, b_m)\vartheta \in \omega \right\}.$
- **Example:**
  $\mathscr{P} = \left\{ \text{teacherOf(peter, maria)}\,.\ \text{isStudentOf}(X, T) \text{ :- } \text{teacherOf}(T, X)\,. \right\}$

   1. $\omega_0 = \emptyset$

Constructing the LHM

- **Definition (T_P-operator, aka *immediate consequence operator* for LHB):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as
$T_P(\omega) = \left\{h\vartheta \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}, \vartheta \text{ is a grounding substitution and } (b_1, \ldots, b_m)\vartheta \in \omega\right\}$.
- **Example:**
$\mathscr{P} = \left\{\text{teacherOf(peter, maria)}\,.\ \text{isStudentOf}(X, T) \text{ :- } \text{teacherOf}(T, X)\,.\right\}$

1. $\omega_0 = \emptyset$
2. $\omega_1 = T_P(\omega_0) = \{\text{teacherOf(peter, maria)}\}$

Constructing the LHM

- **Definition (*T_P*-operator, aka *immediate consequence operator* for LHB):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as
$T_P(\omega) = \{h\vartheta \mid h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}, \vartheta \text{ is a grounding substitution and } (b_1, \ldots, b_m)\vartheta \in \omega\}$.
- **Example:**
$\mathscr{P} = \{\text{teacherOf(peter, maria) . isStudentOf}(X, T) \text{ :- teacherOf}(T, X) \text{ . }\}$

1. $\omega_0 = \emptyset$
2. $\omega_1 = T_P(\omega_0) = \{\text{teacherOf(peter, maria)}\}$
3. $\omega_2 = T_P(\omega_1) = \{\text{teacherOf(peter, maria), studentOf(maria, peter)}\}$

Constructing the LHM

- **Definition (T_P-operator, aka *immediate consequence operator* for LHB):** Let $\mathscr{P}$ be a definite program and ω be an interpretation. Then the T_P-operator is defined as
  $T_P(\omega) = \left\{h\vartheta \,|\, h \Leftarrow b_1 \wedge \ldots \wedge b_m \in \mathscr{P}, \vartheta \text{ is a grounding substitution and } (b_1, \ldots, b_m)\vartheta \in \omega\right\}$.
- **Example:**
  $\mathscr{P} = \left\{\text{teacherOf(peter, maria)} \,.\, \text{isStudentOf}(X, T) \text{ :- teacherOf}(T, X)\,.\right\}$

1. $\omega_0 = \emptyset$
2. $\omega_1 = T_P(\omega_0) = \{\text{teacherOf(peter, maria)}\}$
3. $\omega_2 = T_P(\omega_1) = \{\text{teacherOf(peter, maria), studentOf(maria, peter)}\}$
4. $\omega_3 = T_P(\omega_2) = \omega_2$ (fixpoint -> we have the LHM).

Resolution

Resolution

- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).

Resolution

- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).
- When we know what we want to "ask about", we can use resolution

Resolution

- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).
- When we know what we want to "ask about", we can use resolution.
- **Example:** $\mathscr{P} = \{a \Leftarrow b \wedge c, d \Leftarrow e \wedge f, b, c \Leftarrow b\}$. We want to know whether $\mathscr{P} \vDash a$. For that we negate a (with resolution, we use proof by contradiction) and add it to $\mathscr{P}$ and convert the implications to clauses: $\mathscr{P} = \{\neg a, a \vee \neg b \vee \neg c, d \vee \neg e \vee \neg f, b, c \vee \neg b\}$ and perform resolution.

Resolution

- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).
- When we know what we want to "ask about", we can use resolution.
- **Example:** $\mathscr{P} = \{a \Leftarrow b \wedge c, d \Leftarrow e \wedge f, b, c \Leftarrow b\}$. We want to know whether $\mathscr{P} \models a$. For that we negate a (with resolution, we use proof by contradiction) and add it to $\mathscr{P}$ and convert the implications to clauses: $\mathscr{P} = \{\neg a, a \vee \neg b \vee \neg c, d \vee \neg e \vee \neg f, b, c \vee \neg b\}$ and perform resolution.

$$a \vee \neg b \vee \neg c \qquad b \qquad c \vee \neg b$$

Resolution

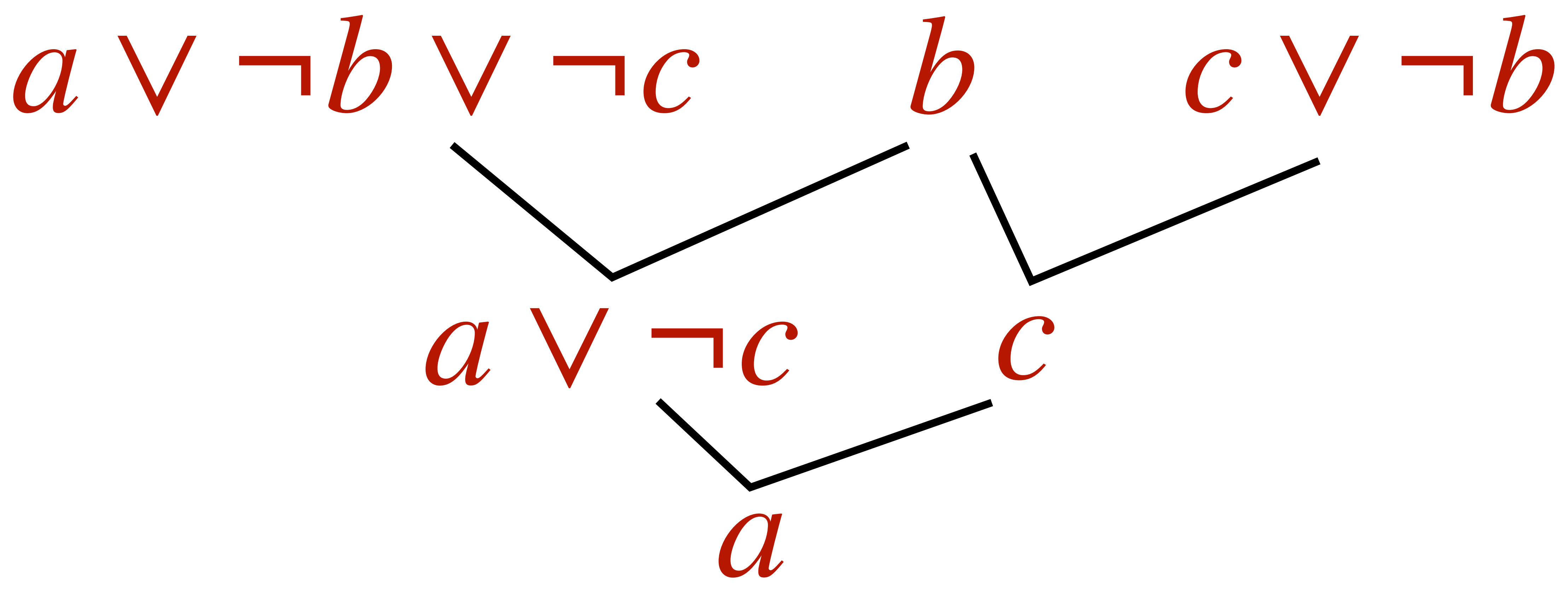
- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).
- When we know what we want to "ask about", we can use resolution.
- **Example:** $\mathscr{P} = \{a \Leftarrow b \wedge c, d \Leftarrow e \wedge f, b, c \Leftarrow b\}$. We want to know whether $\mathscr{P} \vDash a$. For that we negate a (with resolution, we use proof by contradiction) and add it to $\mathscr{P}$ and convert the implications to clauses: $\mathscr{P} = \{\neg a, a \vee \neg b \vee \neg c, d \vee \neg e \vee \neg f, b, c \vee \neg b\}$ and perform resolution.

$a \vee \neg b \vee \neg c \qquad b \qquad c \vee \neg b$

$a \vee \neg c \qquad c$

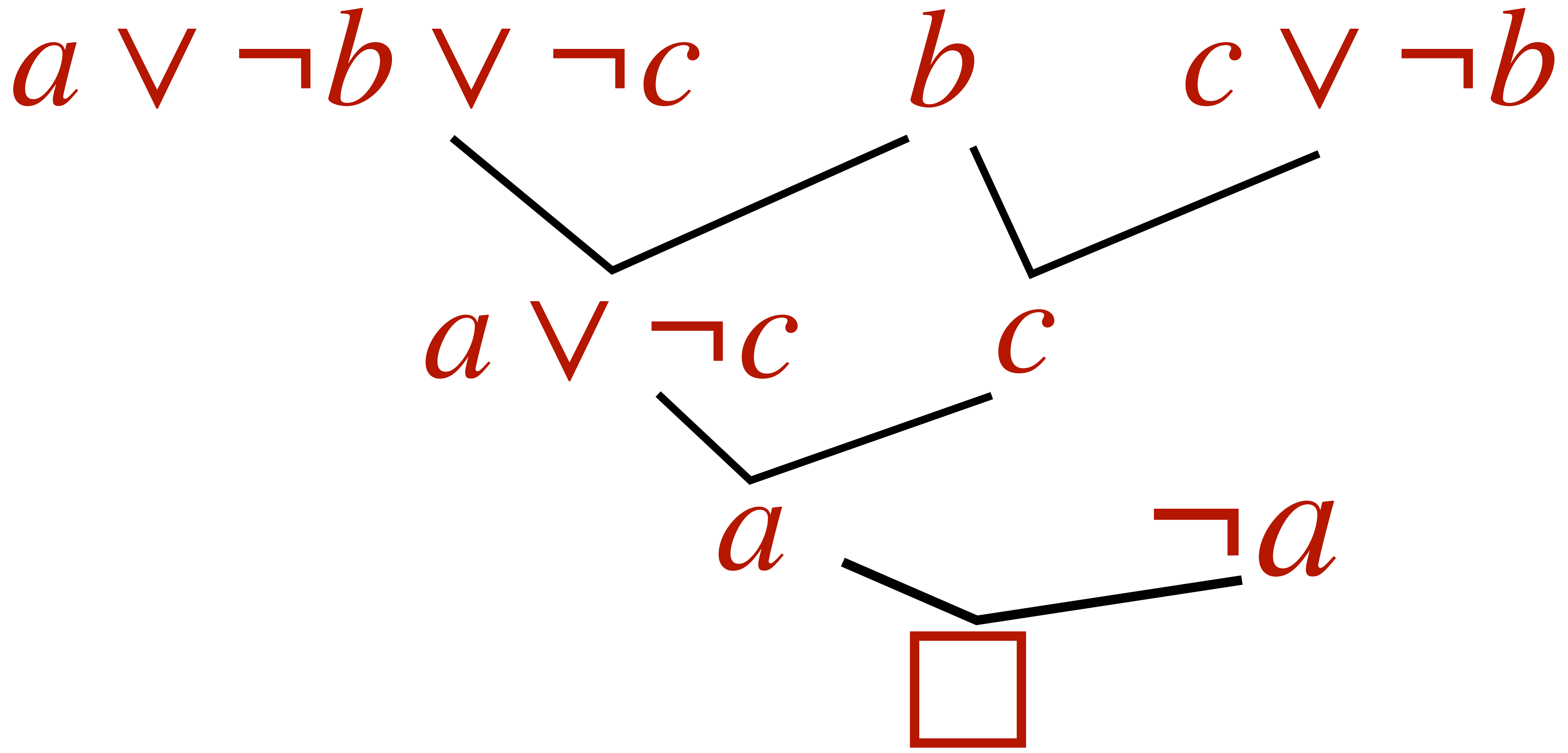
Resolution

- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).
- When we know what we want to "ask about", we can use resolution.
- **Example:** $\mathscr{P} = \{a \Leftarrow b \wedge c, d \Leftarrow e \wedge f, b, c \Leftarrow b\}$. We want to know whether $\mathscr{P} \models a$. For that we negate a (with resolution, we use proof by contradiction) and add it to $\mathscr{P}$ and convert the implications to clauses: $\mathscr{P} = \{\neg a, a \vee \neg b \vee \neg c, d \vee \neg e \vee \neg f, b, c \vee \neg b\}$ and perform resolution.



Resolution

- Computing the complete least model using the T_P-operator is often impractical (as we will see, in the first-order case sometimes even impossible).
- When we know what we want to "ask about", we can use resolution.
- **Example:** $\mathscr{P} = \{a \Leftarrow b \wedge c, d \Leftarrow e \wedge f, b, c \Leftarrow b\}$. We want to know whether $\mathscr{P} \vDash a$. For that we negate a (with resolution, we use proof by contradiction) and add it to $\mathscr{P}$ and convert the implications to clauses: $\mathscr{P} = \{\neg a, a \vee \neg b \vee \neg c, d \vee \neg e \vee \neg f, b, c \vee \neg b\}$ and perform resolution.



Propositional Resolution

Propositional resolution is

- ✓ sound: it derives only logical consequences.
- ✓ incomplete: it cannot derive arbitrary tautologies like $a \Rightarrow a$.
- ✓ …but ***refutation-complete***: it derives the empty clause from any inconsistent set of clauses.

An Example (1): Full Program

```
likes(peter,S):-student_of(S,peter)

student_of(S,T):-follows(S,C),teaches(T,C)

follows(maria,ai_techniques)

teaches(peter,ai_techniques)
```

An Example (3)

- **Herbrand universe:** {peter, maria, ai_techniques}
- **Herbrand base:**
  {likes(peter, peter), likes(maria, maria), likes(peter, maria),
  likes(maria, peter), likes(ai_techniques, peter), . . . , student_of(peter, peter), student_of(maria, maria),
  student_of(peter, maria), student_of(maria, peter), student_of(ai_techniques, peter), …
  teaches(. . . , . . .), . . . ,

An Example (4)

```
:-likes(peter,N)
```

We want to query whether someone likes Peter (as a bonus we will also learn who that is!)

An Example (4)

```
:-likes(peter,N)
```

```
likes(peter,S):-student_of(S,peter)
```

An Example (4)

```
:-likes(peter,N)                    likes(peter,S):-student_of(S,peter)
                                  {S->N}
:-student_of(N,peter)
```

An Example (4)

```
:-likes(peter,N)                 likes(peter,S):-student_of(S,peter)

                                          {S->N}

:-student_of(N,peter)        student_of(S,T):-follows(S,C),teaches(T,C)
```

An Example (4)

```
:-likes(peter,N)                    likes(peter,S):-student_of(S,peter)
                                    {S->N}
:-student_of(N,peter)          student_of(S,T):-follows(S,C),teaches(T,C)
                                    {S->N,T->peter}
:-follows(N,C),teaches(peter,C)
```

An Example (4)

```
:-likes(peter,N)                       likes(peter,S):-student_of(S,peter)
                                       {S->N}
:-student_of(N,peter)      student_of(S,T):-follows(S,C),teaches(T,C)
                                       {S->N,T->peter}
:-follows(N,C),teaches(peter,C)     follows(maria,ai_techniques)
```

An Example (4)

:-likes(peter,N)

likes(peter,S):-student_of(S,peter)

{S->N}

:-student_of(N,peter)

student_of(S,T):-follows(S,C),teaches(T,C)

{S->N,T->peter}

:-follows(N,C),teaches(peter,C)

follows(maria,ai_techniques)

{N->maria,C->ai_techniques}

:-teaches(peter,ai_techniques)

An Example (4)

```
:-likes(peter,N)                      likes(peter,S):-student_of(S,peter)
                                      {S->N}
:-student_of(N,peter)                 student_of(S,T):-follows(S,C),teaches(T,C)
                                      {S->N,T->peter}
:-follows(N,C),teaches(peter,C)       follows(maria,ai_techniques)
                                      {N->maria,C->ai_techniques}
:-teaches(peter,ai_techniques)        teaches(peter,ai_techniques)
```

An Example (4)

```
:-likes(peter,N)                      likes(peter,S):-student_of(S,peter)
                                      {S->N}
:-student_of(N,peter)                 student_of(S,T):-follows(S,C),teaches(T,C)
                                      {S->N,T->peter}
:-follows(N,C),teaches(peter,C)       follows(maria,ai_techniques)
                                      {N->maria,C->ai_techniques}
:-teaches(peter,ai_techniques)        teaches(peter,ai_techniques)

[]
```

`N->maria` is the **answer substitution**.

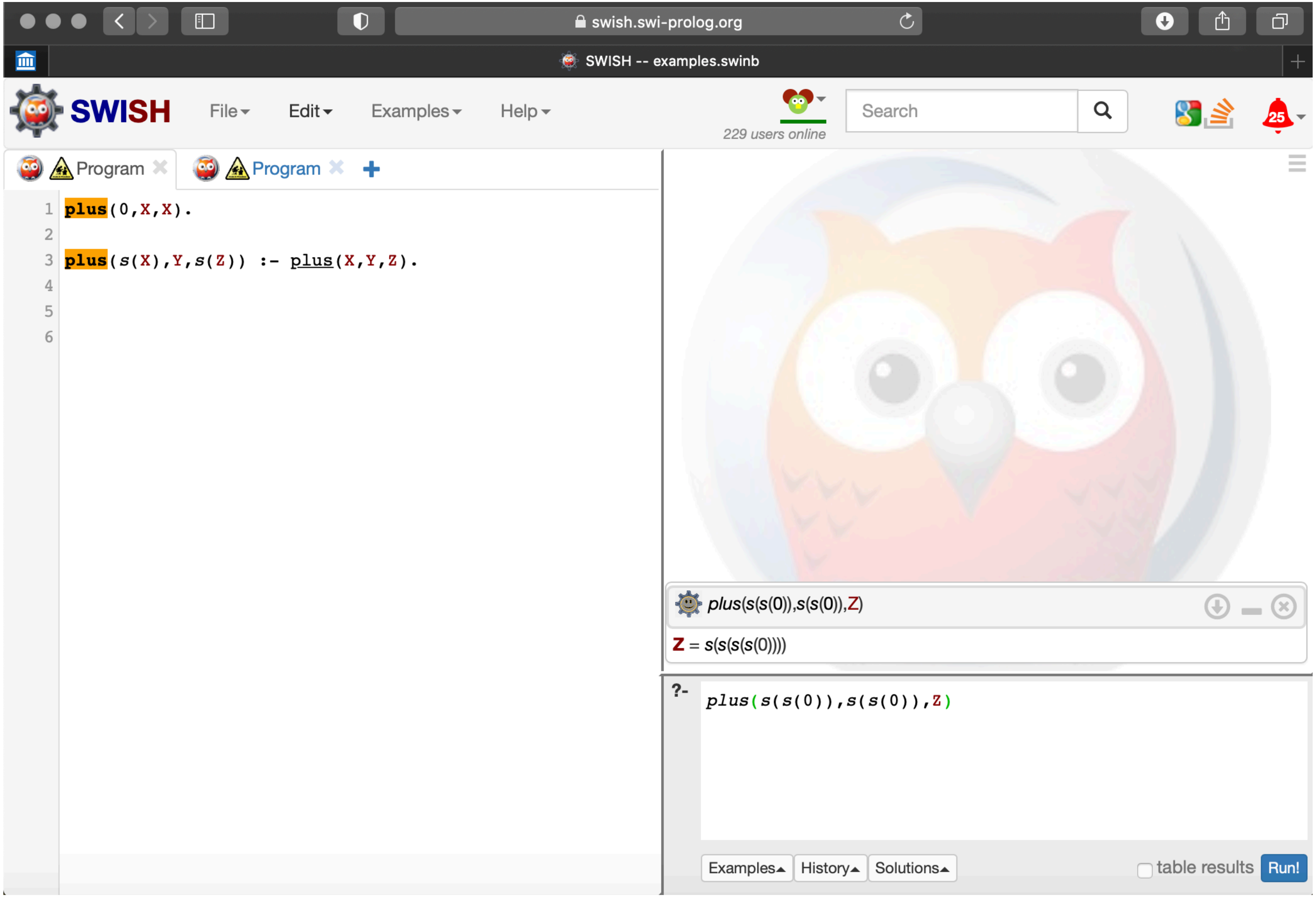
You can try to solve the previous example using the T_P-operator (it is still possible here).

Some Programs Have Infinite LHMs

- …for such programs we cannot construct the LHM using the T_P-operator in practice (it still works well as a theoretical construct, though) and backward chaining (using resolution) is our only hope.
- **Example:**

```
plus(0,X,X).
plus(s(X),Y,s(Z)):-plus(X,Y,Z).
```

- **Herbrand universe:** set of ground terms $\{0, \text{s}(0), \text{s}(\text{s}(0)), \text{s}(\text{s}(\text{s}(0))), \ldots\}$
- **Herbrand base:** $\{\text{plus}(0,0,0), \text{plus}(s(0),0,0), \ldots, \ldots\}$
- **LHM:** … try yourself.


swish.swi-prolog.org
SWISH -- examples.swinb
SWISH
File
Edit
Examples
Help
229 users online
Search
Program
Program
plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
plus(s(s(0)),s(s(0)),Z)
Z = s(s(s(s(0))))
?-
plus(s(s(0)),s(s(0)),Z)
Examples
History
Solutions
table results
Run!

Now we can also get subtraction from addition (using X-Y=Z iff X=Y+Z):

```
minus(X,Y,Z):-plus(Y,Z,X).
```

SWISH -- examples.swinb

SWISH

Program | Program

```
plus(0,X,X).

plus(s(X),Y,s(Z)) :- plus(X,Y,Z).

minus(X,Y,Z) :- plus(Y,Z,X).
```

minus(s(s(s(s(s(0))))),s(s(0)),Z)

Z = s(s(s(0)))

```
?- minus(s(s(s(s(s(0))))),s(s(0)),Z)
```

Examples History Solutions table results Run!

Another Example

A Prolog DB (1)

```
connected(nemocnice_motol,petriny,green).
connected(petriny,nadrazi_veleslavin,green).
connected(nadrazi_veleslavin,borislavka,green).
connected(borislavka,dejvicka,green).
connected(dejvicka,hradcanska,green).
connected(hradcanska,malostranska,green).
connected(malostranska,staromestska,green).
connected(staromestska,mustek,green).
connected(mustek,muzeum,green).
connected(muzeum,namesti_miru,green).
connected(namesti_miru,jiriho_z_podebrad,green).
connected(jiriho_z_podebrad,flora,green).
connected(flora,zelivskeho,green).
connected(zelivskeho,strasnicka,green).
connected(strasnicka,skalka,green).
connected(skalka,depo_hostivar,green).
```

A Prolog DB (2)

```
connected(letnany,prosek,red).
connected(prosek,strizkov,red).
connected(strizkov,ladvi,red).
connected(ladvi,kobylisy,red).
connected(kobylisy,nadrazi_holesovice,red).
connected(nadrazi_holesovice,vltavska,red).
connected(vltavska,florenc,red).
connected(florenc,hlavni_nadrazi,red).
connected(hlavni_nadrazi,muzeum,red).
connected(muzeum,i_p_pavlova,red).
connected(i_p_pavlova,vysehrad,red).
connected(vysehrad,prazskeho_povstani,red).
connected(prazskeho_povstani,pankrac,red).
connected(pankrac,budejovicka,red).
connected(budejovicka,kacerov,red).
connected(kacerov,roztyly,red).
connected(roztyly,chodov,red).
connected(chodov,opatov,red).
connected(opatov,haje,red).
```

A Prolog DB (3)

```
connected(zlicin,stodulky,yellow).
connected(stodulky,luka,yellow).
connected(luka,luziny,yellow).
connected(luziny,hurka,yellow).
connected(hurka,nove_butovice,yellow).
connected(nove_butovice,jinonice,yellow).
connected(jinonice,radlicka,yellow).
connected(radlicka,smichov,yellow).
connected(smichov,andel,yellow).
connected(andel,karlovo_namesti,yellow).
connected(karlovo_namesti,narodni_trida,yellow).
connected(narodni_trida,mustek,yellow).
connected(mustek,namesti_republiky,yellow).
connected(namesti_republiky,florenc,yellow).
connected(florenc,krizikova,yellow).
connected(krizikova,invalidovna,yellow).
connected(invalidovna,palmovka,yellow).
connected(palmovka,ceskomoravska,yellow).
connected(ceskomoravska,vysocanska,yellow).
connected(vysocanska,kolbenova,yellow).
connected(kolbenova,hloubetin,yellow).
connected(hloubetin,rajska_zahrada,yellow).
connected(rajska_zahrada,cerny_most,yellow).
```

“Nearby”

Two stations are nearby if they are on the same line with at most one other station in between:

“Nearby”

Two stations are nearby if they are on the same line with at most one other station in between:

```
nearby(zlicin,luka).
nearby(luka,zlicin).
nearby(zlicin,stodulky).
nearby(stodulky,zlicin).
nearby(luka,luziny).
nearby(luziny,luka).
nearby(luka,hurka).
```

“Nearby”

Two stations are nearby if they are on the same line with at most one other station in between:

```
nearby(zlicin,luka).
nearby(luka,zlicin).
nearby(zlicin,stodulky).
nearby(stodulky,zlicin).
nearby(luka,luziny).
nearby(luziny,luka).
nearby(luka,hurka).
…
```

or better

```
nearby(X,Y):-connectedS(X,Y,L).
nearby(X,Y):-connectedS(X,Z,L),connectedS(Z,Y,L).
connectedS(X,Y,W) :- connected(X,Y,W).
connectedS(X,Y,W) :- connected(Y,X,W).
```

“Not too far”

Compare

```
nearby(X,Y):-connectedS(X,Y,L).
nearby(X,Y):-connectedS(X,Z,L),connectedS(Z,Y,L).
```

with

```
not_too_far(X,Y):-connectedS(X,Y,L).
not_too_far(X,Y):-connectedS(X,Z,L1),connectedS(Z,Y,L2).
```

“Not too far”

Compare

```
nearby(X,Y):-connectedS(X,Y,L).
nearby(X,Y):-connectedS(X,Z,L),connectedS(Z,Y,L).
```

with

```
not_too_far(X,Y):-connectedS(X,Y,L).
not_too_far(X,Y):-connectedS(X,Z,L1),connectedS(Z,Y,L2).
```

This can be rewritten with don’t cares:

```
not_too_far(X,Y):-connectedS(X,Y,_).
not_too_far(X,Y):-connectedS(X,Z,_),connectedS(Z,Y,_).
```

```
?-nearby(mustek,W)
```

```
?-nearby(mustek,W)
```

```
nearby(X1,Y1):-connected(X1,Y1,L1)
```

```
?-nearby(mustek,W)

                        nearby(X1,Y1):-connected(X1,Y1,L1)

                        {X1->mustek, Y1->W}

?-connected(mustek,W,L1)
```

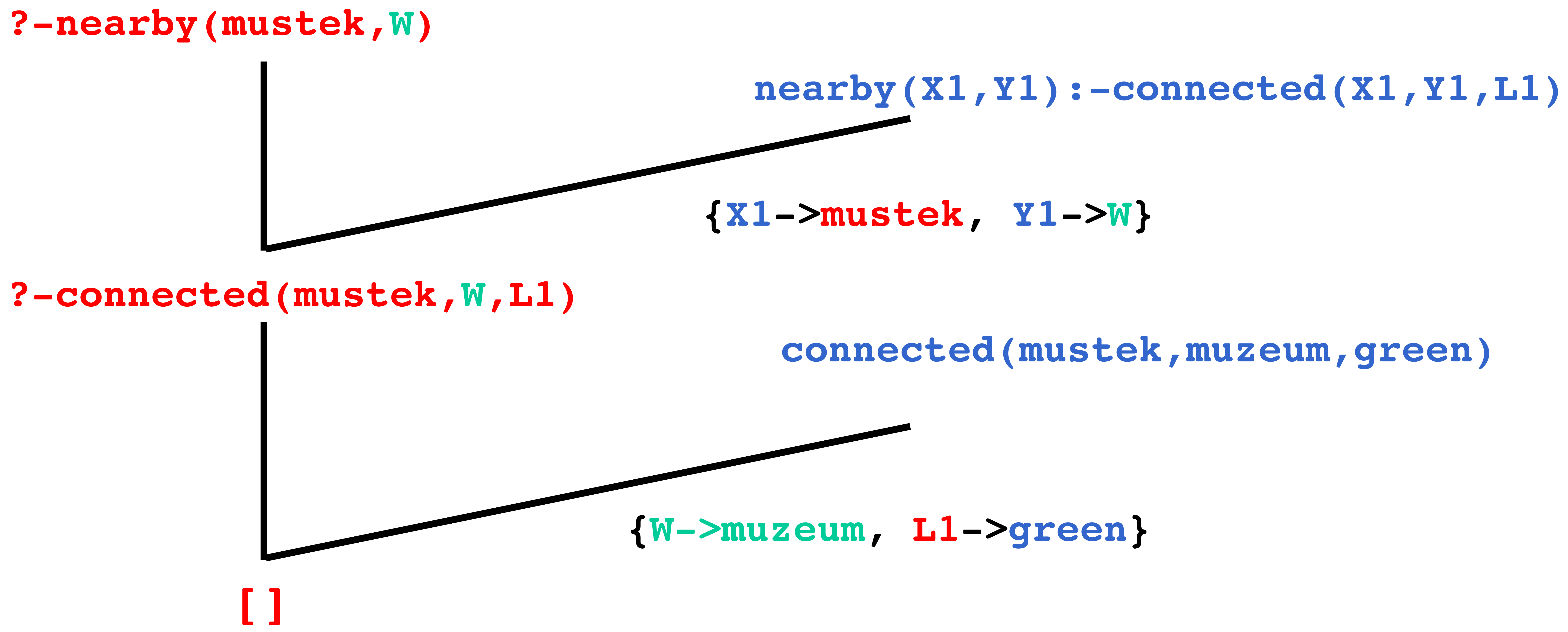
```
?-nearby(mustek,W)

                                nearby(X1,Y1):-connected(X1,Y1,L1)

                              {X1->mustek, Y1->W}

?-connected(mustek,W,L1)

                                connected(mustek,muzeum,green)
```


?-nearby(mustek,W)
nearby(X1,Y1):-connected(X1,Y1,L1)
{X1->mustek, Y1->W}
?-connected(mustek,W,L1)
connected(mustek,muzeum,green)
{W->muzeum, L1->green}
[]

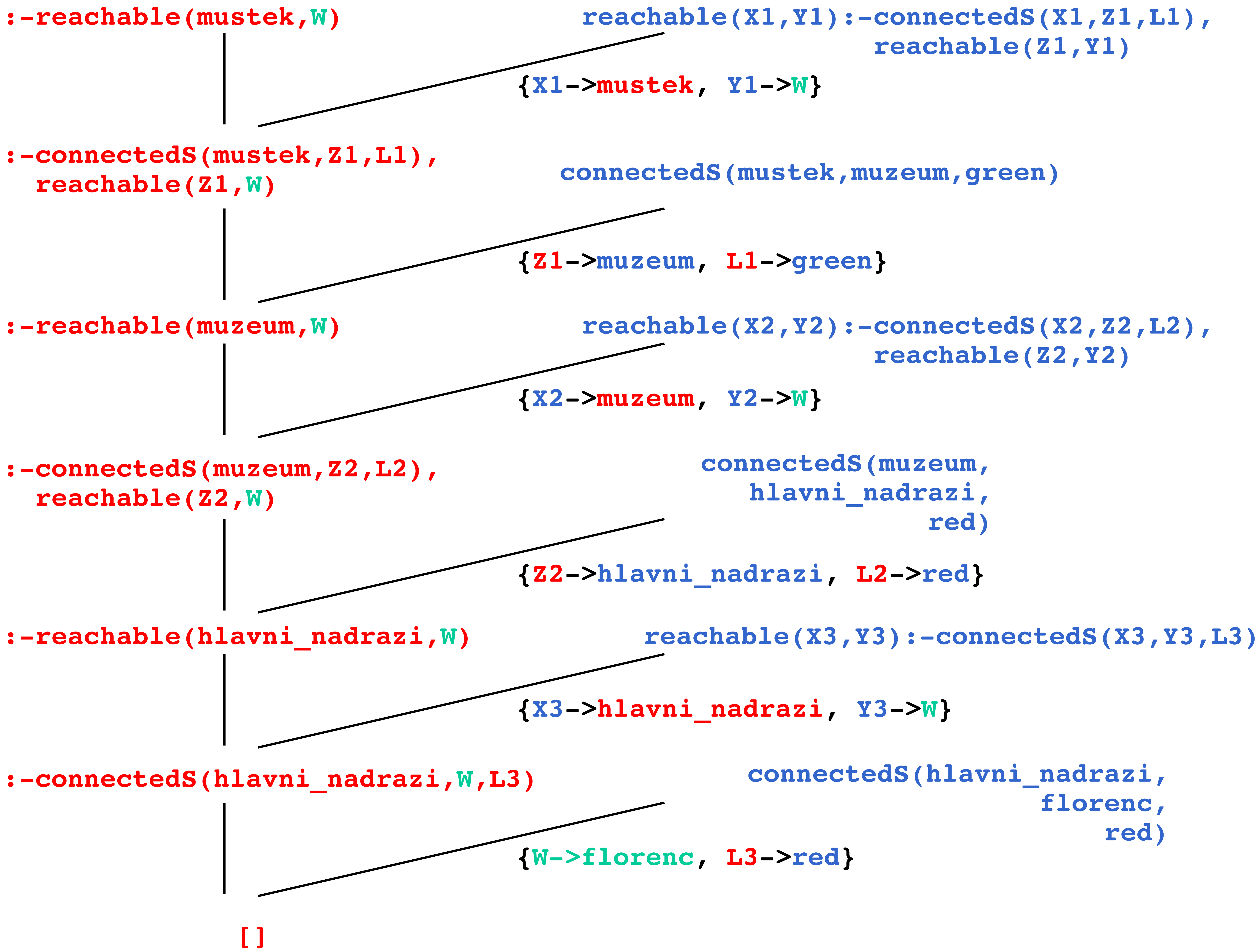
“Reachable”

A station is reachable from another if they are on the same line, or with one, two, … changes:

```
reachable(X,Y):-connectedS(X,Y,L).
reachable(X,Y):-connectedS(X,Z,L1),connectedS(Z,Y,L2).
reachable(X,Y):-connectedS(X,Z1,L1),connectedS(Z1,Z2,L2),
                connectedS(Z2,Y,L3).
...
```

or better

```
reachable(X,Y):-connectedS(X,Y,L).
reachable(X,Y):-connectedS(X,Z,L),reachable(Z,Y).
```


:-reachable(mustek,W)
reachable(X1,Y1):-connectedS(X1,Z1,L1),
reachable(Z1,Y1)
{X1->mustek, Y1->W}
:-connectedS(mustek,Z1,L1),
reachable(Z1,W)
connectedS(mustek,muzeum,green)
{Z1->muzeum, L1->green}
:-reachable(muzeum,W)
reachable(X2,Y2):-connectedS(X2,Z2,L2),
reachable(Z2,Y2)
{X2->muzeum, Y2->W}
:-connectedS(muzeum,Z2,L2),
reachable(Z2,W)
connectedS(muzeum,
hlavni_nadrazi,
red)
{Z2->hlavni_nadrazi, L2->red}
:-reachable(hlavni_nadrazi,W)
reachable(X3,Y3):-connectedS(X3,Y3,L3)
{X3->hlavni_nadrazi, Y3->W}
:-connectedS(hlavni_nadrazi,W,L3)
connectedS(hlavni_nadrazi,
florenc,
red)
{W->florenc, L3->red}
[]

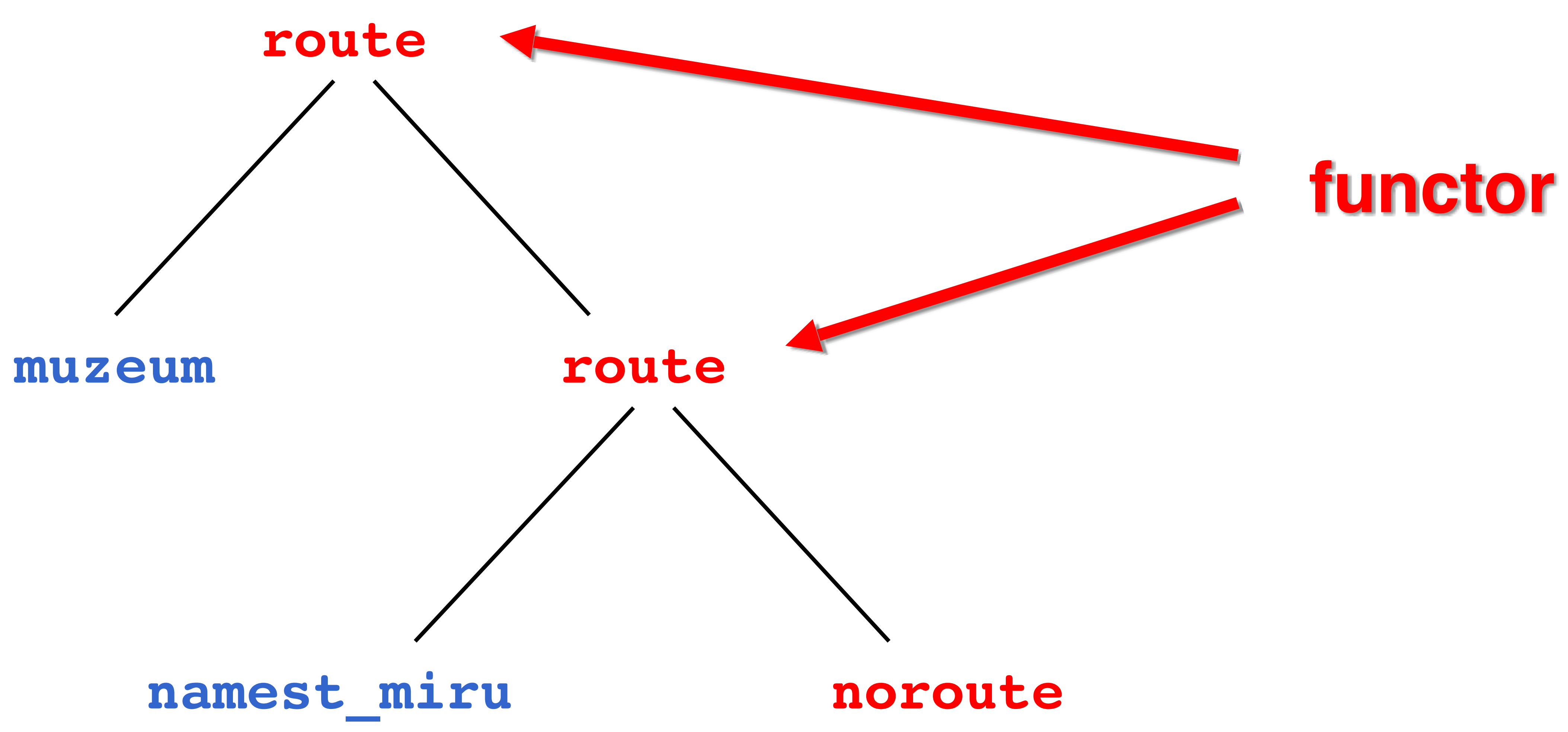
There is a catch!

- The answers that we get depend on the exact way Prolog works inside. We will talk about that next time.

"Recording the Path"

```
reachable(X,Y,noroute):-connected(X,Y,L).
reachable(X,Y,route(Z,R)):-connected(X,Z,L),
                           reachable(Z,Y,R).

?-reachable(mustek,jiriho_z_podebrad,R).
R = route(muzeum,route(namesti_miru,noroute));
…
```



A Digression: Skolemization

"Everybody knows somebody."

A Digression: Skolemization

"Everybody knows somebody."

Skolemization to avoid an existential quantifier

```
knows(X,person_known_by(X)).
```

functor term

complex term

```
knows(peter,person_known_by(peter)).
knows(anna,person_known_by(anna)).
knows(paul,person_known_by(paul)).
…
```

To be continued…