PROLOG 2025

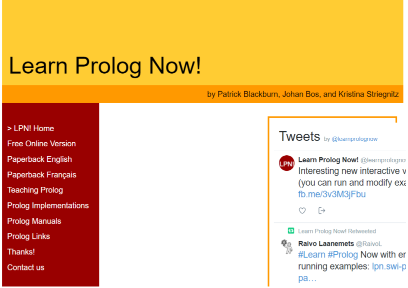
ELTE, LOGIC DEPARTMENT

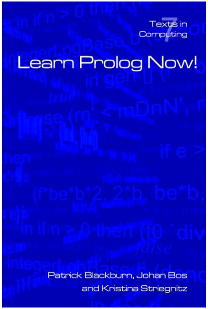
Attila Molnár
Kőbányai Szent László Gimnázium

September 9, 2025

COURSE MATERIAL: LEARN PROLOG NOW!

Aim of the course: to advertise Prolog via programming in it.





One of the best introductory books. It does not cover much, but it is an excellent introduction **written by logicians**.

(for modal logicians: one of the authors is Patrick Blackburn!)

Outline

Introduction (history, basic concepts, course material)

- history
- basic concepts
- course material

Syntax

- elements of the language
- facts, rules, knowledge base
- queries

Horn clauses

Unification

Introduction

Brief history

1972 Marseille (PROgramming in LOGic)

1978 PROLOG interpreter

1987 SWI (free; online & downloadable – has better debugger).

$\stackrel{\text{def}}{=}$ SWI PROLOG interpreter

1987 Jaffar and Lassez: Constraint logic programming. Powerful and beautiful clp packages are included since.

1990 SICStus Prolog (proprietary)

Used at…

- Watson – Q&A machine of IBM e.g. cancer
- NASA – international space station (Clarissa)
- Ericsson – Network Resource Manager (operator assistant)
- Logistics
- Data mining – Rule Discovery System (RDS)

Declarative, logic programming

Declarative programming: define **what** to solve, not **how** to solve it.

We describe the problem (instead of the way to the solution – algorithms), the program solves it using it's own strategies (e.g. backtracking)

Logic programming: syntax follows the syntax of logic (Horn logic).

Program: logical formula(s)
Running the program: evaluation of the formula(s)
No classical variables (temporal substitution vs assignment)
Recursion instead of loops

Syntax

LANGUAGE ELEMENTS

CONSTANTS

- Atoms
  - String beginning with lower case letter

    `dEPARTMENT`, `logic`, `l_o_g_i_c`, `year1988`
  - String in single quotes

    `'Mekis'`, `'_String'`, `'WHAT3V3R'`
- Numbers: usual…

  `-1`, `0`, `1`, `2`,…, integers, floats…

Language elements

Variables

- String starting with upper case letter

  `Mekis`, `YOLO`, `L_AbamBA`

- String starting with

  `_`, `_logic`, `_LoL`

The variable `_` is very special, will treat it differently!
(The string cannot contain any `(`, `)` or .)

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

```
thisiaVariable
```

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	atom
`prolog2017`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	atom
`prolog2017`	atom
`Nimbus2000`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	atom
`prolog2017`	atom
`Nimbus2000`	variable
`John loves Mary`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	atom
`prolog2017`	atom
`Nimbus2000`	variable
`John loves Mary`	neither
`_John`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	atom
`prolog2017`	atom
`Nimbus2000`	variable
`John loves Mary`	neither
`_John`	variable
`'John'`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, and which are **neither**?

`thisiaVariable`	atom
`ELTE`	variable
`this_is_a_Variable`	atom
`'_John`	neither
`'John loves Mary'`	atom
`prolog2017`	atom
`Nimbus2000`	variable
`John loves Mary`	neither
`_John`	variable
`'John'`	atom

Language elements

Terms

- Constants and variables are **terms**.
- If `f` is an atom and $\tau_1, \ldots, \tau_n$ are **terms**, then

$$\texttt{f}(\tau_1, \tau_2, \ldots, \tau_n)$$

is a (complex) **term**.

```
grandmotherof(X,motherof(fatherof(Y)))
```

The number of contained terms are called the arities of the function/functor/predicate `f`. In PROLOG, if `f` is a ternary function we refer to that in the following way: `f/3`. This will be very important, because it is not uncommon in PROLOG to use different functions such as `f`/2 , `f`/3 and `f`/5 in a parallel way. We will never do that though.

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

```
'loves(Vincent,Mia)'
```

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	complex term (butch/1)
`beats(Batman,'Superman')`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	complex term (butch/1)
`beats(Batman,'Superman')`	complex term (butch/2)
`_or(Super(M),Bat(M))`	

Exercise

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	complex term (butch/1)
`beats(Batman,'Superman')`	complex term (butch/2)
`_or(Super(M),Bat(M))`	neither
`or(super(man),bat(man))`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	complex term (butch/1)
`beats(Batman,'Superman')`	complex term (butch/2)
`_or(Super(M),Bat(M))`	neither
`or(super(man),bat(man))`	complex term (or/2, super/1, bat/1)
`(Batman beats Superman)`	

EXERCISE

Which of the following sequences of characters are **atom**s, which are **variable**s, which are **complex term**s, and which are **not term**s at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	complex term (butch/1)
`beats(Batman,'Superman')`	complex term (butch/2)
`_or(Super(M),Bat(M))`	neither
`or(super(man),bat(man))`	complex term (or/2, super/1, bat/1)
`(Batman beats Superman)`	not term
`loves(Vincent,Mia`	

EXERCISE

Which of the following sequences of characters are **atoms**, which are **variables**, which are **complex terms**, and which are **not terms** at all? Give the functor and arity of each complex term.

`'loves(Vincent,Mia)'`	atom
`loves(Vincent Mia)`	not term
`or(super(M),bat(M))`	complex term (or/2, super/1, bat/1)
`Butch(boxer)`	neither
`boxer(Butch)`	complex term (butch/1)
`beats(Batman,'Superman')`	complex term (butch/2)
`_or(Super(M),Bat(M))`	neither
`or(super(man),bat(man))`	complex term (or/2, super/1, bat/1)
`(Batman beats Superman)`	not term
`loves(Vincent,Mia`	not term

Language elements

Connectives

and	,
or	;
if	:-
not	\+

The “not” \+ is **NOT** the negation you think of! Do not use it until we learn it how to use. It modifies the searching algorithm of , and as such it is not a declarative command!

Facts, rules and knowledge base

If τ is a term *that does not contain any variable*, then τ`.` is a **fact**. (Ending dot!)

```
loves(adam, motherof('Cain')).
```

- If σ and τ are terms, then σ`:-`τ`.` is **a rule**. (if)
- If ϱ`.` is a **rule** and τ is a term, then ϱ`,`τ`.` is a **rule** as well. (and)

```
jealous(X, Y):-loves(X, Z),loves(Y, Z).
jealous(_loverNo1, _loverNo2) :-  loves(_loverNo1, LovedOne),
                                  loves(_loverNo2, LovedOne).
```

(spaces and breaks do not matter)
A knowledge base is a collection of facts and rules.

Knowledge bases

A **knowledge base** is collection of facts and rules.

```
loves(vincent, mia).
loves(marcellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
jealous(X, Y) :- loves(X, Z),loves(Y, Z).
```

QUERIES / PROOF SEARCHES

```
loves(vincent, mia).
loves(marcellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
jealous(X, Y) :- loves(X, Z),loves(Y, Z).
```

```
jealous(X, Y).
```

and will answer that...

EXERCISE

Translate the following rules into English (natural language)!

```
man('Eomund').
woman('Theodwyn').
son('Eomer', 'Eomund').
daughter('Eowyn', 'Eomund').
```

```
man('Eomer').
woman('Eowyn').
son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Theodwyn').
```

Exercise

Translate the following rules into English (natural language)!

```
man('Eomund').                    man('Eomer').
woman('Theodwyn').                woman('Eowyn').
son('Eomer', 'Eomund').           son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Eomund').      daughter('Eowyn', 'Theodwyn').
```

```
father(X,Y):- man(X), son(Y,X).
father(X,Y):- man(X), daughter(Y,X).
```

EXERCISE

Translate the following rules into English (natural language)!

```
man('Eomund').                     man('Eomer').
woman('Theodwyn').                 woman('Eowyn').
son('Eomer', 'Eomund').            son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Eomund').       daughter('Eowyn', 'Theodwyn').
```

```
father(X,Y):- man(X), son(Y,X).
father(X,Y):- man(X), daughter(Y,X).
mother(X,Y):- woman(X), son(Y,X).
mother(X,Y):- woman(X), daughter(Y,X).
```

Exercise

Translate the following rules into English (natural language)!

```
man('Eomund').
woman('Theodwyn').
son('Eomer', 'Eomund').
daughter('Eowyn', 'Eomund').
```

```
man('Eomer').
woman('Eowyn').
son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Theodwyn').
```

```
father(X,Y):- man(X), son(Y,X).
father(X,Y):- man(X), daughter(Y,X).
mother(X,Y):- woman(X), son(Y,X).
mother(X,Y):- woman(X), daughter(Y,X).
parent(X):- father(X,_); mother(X,_).
parent(X,Y):- father(X,Y); mother(X,Y).
```

EXERCISE

Translate the following rules into English (natural language)!

```
man('Eomund').                     man('Eomer').
woman('Theodwyn').                 woman('Eowyn').
son('Eomer', 'Eomund').            son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Eomund').       daughter('Eowyn', 'Theodwyn').
```

```
father(X,Y):- man(X), son(Y,X).
father(X,Y):- man(X), daughter(Y,X).
mother(X,Y):- woman(X), son(Y,X).
mother(X,Y):- woman(X), daughter(Y,X).
parent(X):- father(X,_); mother(X,_).
parent(X,Y):- father(X,Y); mother(X,Y).
sister(Y,Z):-parent(X),daughter(Y,X),daughter(Z,X),Z\=Y.
sister(Y,Z):- parent(X), daughter(Y,X), son(Z,X).
brother(Y,Z):- parent(X), son(Y,X), son(Z,X), Z\=Y.
brother(Y,Z):- parent(X), son(Y,X), daughter(Z,X).
```

EXERCISE

Translate the following rules into English (natural language)!

```
man('Eomund').                  man('Eomer').
woman('Theodwyn').              woman('Eowyn').
son('Eomer', 'Eomund').         son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Eomund').    daughter('Eowyn', 'Theodwyn').
```

```
father(X,Y):- man(X), son(Y,X).
father(X,Y):- man(X), daughter(Y,X).
mother(X,Y):- woman(X), son(Y,X).
mother(X,Y):- woman(X), daughter(Y,X).
parent(X):- father(X,_); mother(X,_).
parent(X,Y):- father(X,Y); mother(X,Y).
sister(Y,Z):-parent(X),daughter(Y,X),daughter(Z,X),Z\=Y.
sister(Y,Z):- parent(X), daughter(Y,X), son(Z,X).
brother(Y,Z):- parent(X), son(Y,X), son(Z,X), Z\=Y.
brother(Y,Z):- parent(X), son(Y,X), daughter(Z,X).
sibling(X,Y):- brother(X,Y); sister(X,Y).
```

EXERCISE

Translate the following rules into English (natural language)!

```
man('Eomund').                    man('Eomer').
woman('Theodwyn').                woman('Eowyn').
son('Eomer', 'Eomund').           son('Eomer', 'Theodwyn').
daughter('Eowyn', 'Eomund').      daughter('Eowyn', 'Theodwyn').
```

```
father(X,Y):- man(X), son(Y,X).
father(X,Y):- man(X), daughter(Y,X).
mother(X,Y):- woman(X), son(Y,X).
mother(X,Y):- woman(X), daughter(Y,X).
parent(X):- father(X,_); mother(X,_).
parent(X,Y):- father(X,Y); mother(X,Y).
sister(Y,Z):-parent(X),daughter(Y,X),daughter(Z,X),Z\=Y.
sister(Y,Z):- parent(X), daughter(Y,X), son(Z,X).
brother(Y,Z):- parent(X), son(Y,X), son(Z,X), Z\=Y.
brother(Y,Z):- parent(X), son(Y,X), daughter(Z,X).
sibling(X,Y):- brother(X,Y); sister(X,Y).
ancestor(X,Y):- parent(X,Y).
ancestor(X,Y):- parent(X,Z), ancestor(Z,Y).
```

Horn clauses

Horn clauses in propositional logic

Definition

A literal is an atomic sentence or a negation of an atomic sentence.

$$p, q, \dots, \neg p, \neg q, \dots$$

Definition

A clause is a disjunction of literals

$$p \vee q \vee \neg p \vee \neg r \vee q$$

Definition

A Horn clause is a clause with **at most one** positive (unnegated) literals in it.

$$\neg p \vee \neg q \vee \neg p \vee \neg r \vee q$$

Remember that $\neg A \vee B \iff A \rightarrow B$, therefore Horn clauses are sentences of form $(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \rightarrow q$ or $p_1 \rightarrow (p_2 \rightarrow \dots \rightarrow (p_n \rightarrow q) \dots)$ And these sentences are very convenient to apply modus ponens...

Horn clauses in propositional logic

Theorem

The satisfiability problem of a conjunction of Horn clauses is **P-complete** and solvable in **linear time**.

i.e., Horn-logic is super fast.
The cost is of course the expressive power. To use Horn-logic to solve problems the working logician sometimes has to be very tricky in choosing the primitives…

Horn clauses in predicate logic

In predicate logic, atomic sentences are predicates that may contain variables.

$$\text{human}(x) \rightarrow \text{mortal}(x)$$

All unquantified sentences are meant to be universally quantified (check the definition of $\mathfrak{M} \models \varphi$ compared to $\mathfrak{M}, \sigma \models \varphi$!)

$$\forall x(\text{human}(x) \rightarrow \text{mortal}(x))$$

The only way to play with the variables is to sometimes use the same variable again. The key to thinking in that logic is the so-called unification. Within a first-order environment, Horn logic gains so much expressive power that it torpedos P-completeness; the satisfiabilty problem of conjunction of Horn clauses is undecidable.

Unification

Unification: =/2

This is not equality. Equality is `==/2`. Roughly:

„Two terms unify if they are the same term or if they contain variables that ***can be*** *uniformly instantiated with terms in such a way that the resulting terms are equal."*

Semantics of =/2.

- Constants: `a=b` is true iff they are the same atom or the same number, i.e., `a=b`.
- Variable + Term: `X=`τ is true, and $X \mapsto \tau$. ($\mapsto$: variable instantiation)
- Term + Variable: Similarly.
- Complex terms: $\sigma(\tau_1, \ldots, \tau_n) = \sigma'(\tau'_1, \ldots, \tau'_k)$ is true iff
  - $\sigma = \sigma', n = k$,
  - $(\forall i \leq n)\tau_i = \tau'_i$, and
  - the **variable instantiations are compatible**. $X \mapsto a, X \mapsto b \implies a = b$ („For example, it is not possible to instantiate variable X to mia when unifying one pair of arguments, and to instantiate X to vincent when unifying another pair of arguments.")

Unification is intensional!

Excercises (see LPN 2.1)

The symbol `?-`φ is true iff 🦉 finds a way to make φ true. Later we will see what is the backtracking algorithm that 🦉 uses.

```
?- mia = mia.
```

Excercises (see LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
?- mia = mia.                                    yes
?- mia = vincent.
```

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
?- mia = mia.                                   yes
?- mia = vincent.                               yes
?- 2 = 2.
```

Excercises (see LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
1 ?- mia = mia.                                   yes
2 ?- mia = vincent.                               yes
3 ?- 2 = 2.                                       yes
4 ?- 'mia' = mia.
```

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
?- mia = mia.                yes
?- mia = vincent.            yes
?- 2 = 2.                    yes
?- 'mia' = mia.              yes
?- '2' = 2.
```

Excercises (see LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
?- mia = mia.                                   yes
?- mia = vincent.                               yes
?- 2 = 2.                                       yes
?- 'mia' = mia.                                 yes
?- '2' = 2.                                      no
?- mia = X.
```

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff  finds a way to make φ true. Later we will see what is the backtracking algorithm that  uses.

```
1 ?- mia = mia.                                  yes
2 ?- mia = vincent.                              yes
3 ?- 2 = 2.                                      yes
4 ?- 'mia' = mia.                                yes
5 ?- '2' = 2.                                     no
6 ?- mia = X.                          X = mia   yes
7 ?- X = Y.
```

Excercises (see LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
1 ?- mia = mia.                                              yes
2 ?- mia = vincent.                                          yes
3 ?- 2 = 2.                                                  yes
4 ?- 'mia' = mia.                                            yes
5 ?- '2' = 2.                                                 no
6 ?- mia = X.                                    X = mia     yes
7 ?- X = Y.                    X = _5071    Y = _5071        yes
8 ?- X = mia, X = vincent.
```

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

```
1 ?- mia = mia.                                              yes
2 ?- mia = vincent.                                          yes
3 ?- 2 = 2.                                                  yes
4 ?- 'mia' = mia.                                            yes
5 ?- '2' = 2.                                                 no
6 ?- mia = X.                                      X = mia   yes
7 ?- X = Y.                     X = _5071    Y = _5071       yes
8 ?- X = mia, X = vincent.                                    no
```

„An instantiated variable isn't really a variable anymore: it has become what it was instantiated with."

```
9 ?- k(s(g),Y) = k(X,t(k)).
```

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

1 `?- mia = mia.` `yes`

2 `?- mia = vincent.` `yes`

3 `?- 2 = 2.` `yes`

4 `?- 'mia' = mia.` `yes`

5 `?- '2' = 2.` `no`

6 `?- mia = X.` `X = mia` `yes`

7 `?- X = Y.` `X = _5071` `Y = _5071` `yes`

8 `?- X = mia, X = vincent.` `no`
„An instantiated variable isn't really a variable anymore: it has become what it was instantiated with."

9 `?- k(s(g),Y) = k(X,t(k)).` `X = s(g)` `Y = t(k)` `yes`

10 `?- k(s(g), t(k)) = k(X,t(Y)).`

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

1	`?- mia = mia.`		`yes`
2	`?- mia = vincent.`		`yes`
3	`?- 2 = 2.`		`yes`
4	`?- 'mia' = mia.`		`yes`
5	`?- '2' = 2.`		`no`
6	`?- mia = X.`	`X = mia`	`yes`
7	`?- X = Y.`	`X = _5071    Y = _5071`	`yes`
8	`?- X = mia, X = vincent.`		`no`

„An instantiated variable isn't really a variable anymore: it has become what it was instantiated with."

9	`?- k(s(g),Y) = k(X,t(k)).`	`X = s(g)   Y = t(k)`	`yes`
10	`?- k(s(g), t(k)) = k(X,t(Y)).`	`X = s(g)   Y = k`	`yes`
11	`?- loves(X,X) = loves(marcellus,mia).`		

EXCERCISES (SEE LPN 2.1)

The symbol `?-`φ is true iff finds a way to make φ true. Later we will see what is the backtracking algorithm that uses.

1 `?- mia = mia.` — yes

2 `?- mia = vincent.` — yes

3 `?- 2 = 2.` — yes

4 `?- 'mia' = mia.` — yes

5 `?- '2' = 2.` — no

6 `?- mia = X.` — `X = mia` — yes

7 `?- X = Y.` — `X = _5071` `Y = _5071` — yes

8 `?- X = mia, X = vincent.` — no

„An instantiated variable isn't really a variable anymore: it has become what it was instantiated with."

9 `?- k(s(g),Y) = k(X,t(k)).` — `X = s(g)` `Y = t(k)` — yes

10 `?- k(s(g), t(k)) = k(X,t(Y)).` — `X = s(g)` `Y = k` — yes

11 `?- loves(X,X) = loves(marcellus,mia).` — no