How to best teach Prolog (to different audiences)

Manuel Hermenegildo$^{1,2}$ (with P. López-García$^{1,3}$ and J.F. Morales$^{1,2}$)

Based on the talk at the workshop:

$^1$U. Politécnica de Madrid (UPM)
$^2$IMDEA Software Institute
$^3$Spanish Research Council (CSIC)

Main reference: “Some Thoughts on How to Teach Prolog”, In “Prolog - The Next 50 Years”, Warren et al. (Eds.), Springer, LNCS 13900.
How to best teach Prolog

• Lots of good material and systems already exist!

• Our objective here:
  Some complementary thoughts and lessons from our experience teaching Prolog:
    ▶ Mostly to CS undergrads.
    ▶ At U.T. Austin, U. of New Mexico, and T.U.Madrid (UPM).
  (and also as developers of the Ciao prolog system, where we have added many features aimed at teaching Prolog, based on this experience).

• Students have typically been exposed to other languages (imperative/OO, sometimes functional) and possibly logic, specifications, some notions of PL implementation, etc.
  ▶ Challenge: make the material attractive, intriguing, and challenging for this audience.
  ▶ But also great audience, which can appreciate and be impressed!

Our related teaching materials (slides, examples, ALDs): https://cliplab.org/logalg
How to best teach Prolog

• Prolog / LP / CLP must be taught in CS programs,
  ► Only of few major programming paradigms,
    really interesting, different, and useful →
    A CS graduate is simply not complete without knowledge of Prolog.

  and also in other majors, and in schools, ...?

• But it has to be done right!
  ► It is a different paradigm, and needs to be taught differently.
  ► The standard 'programming paradigms' approach can be counter-productive:
    ● Not possible in a couple of weeks emulating Prolog in Scheme.
    ● But, what to do if that is the only slot available? (→ Challenge for the LP community.)

• The main message: do show the beauty!

⇒ Start by explaining “Green’s dream”...
How to best teach Prolog

• Prolog / LP / CLP must be taught in CS programs,
  ► Only of few major programming paradigms,
    really interesting, different, and useful →
    A CS graduate is simply not complete without knowledge of Prolog.

and also in other majors, and in schools, ...?

• But it has to be done right!
  ► It is a different paradigm, and needs to be taught differently.
  ► The standard 'programming paradigms' approach can be counter-productive:
    ● Not possible in a couple of weeks emulating Prolog in Scheme.
    ● But, what to do if that is the only slot available? (→ Challenge for the LP community.)

• The main message: do show the beauty!

⇒ Start by explaining “Green’s dream”...
How to best teach Prolog

- Prolog / LP / CLP must be taught in CS programs,
  - Only of few major programming paradigms,
    - really interesting, different, and useful →
      A CS graduate is simply not complete without knowledge of Prolog.

  and also in other majors, and in schools, ...?

- But it has to be done right!
  - It is a different paradigm, and needs to be taught differently.
  - The standard 'programming paradigms' approach can be counter-productive:
    - Not possible in a couple of weeks emulating Prolog in Scheme.
    - But, what to do if that is the only slot available? (→ Challenge for the LP community.)

- The main message: do show the beauty!

  ⇒ Start by explaining “Green’s dream”...
How to best teach Prolog

- Prolog / LP / CLP *must* be taught in CS programs,
  - Only of few major programming paradigms,
    - really interesting, different, and useful →
      - A CS graduate is simply not complete without knowledge of Prolog.

  and also in other majors, and in schools, ...?

- But it has to be done right!
  - It is a different paradigm, and needs to be taught differently.
  - The standard 'programming paradigms' approach can be counter-productive:
    - Not possible in a couple of weeks emulating Prolog in Scheme.
    - But, what to do if that is the only slot available? (→ Challenge for the LP community.)

- The main message: **do show the beauty!**

  ⇒ Start by explaining “Green’s dream”...
What is the best way to program a computer?
A New View of Computing
Problem

Representation/specification (Logic)

Deduction system
A New View of Computing

Problem

Representation/specification (Logic)

Questions

Deduction system

(Correct) Answers / Results
But then,

- No correctness proofs needed?
- Even no programming needed?
- Is this possible?
Prolog is the Materialization of this Dream!

But then,
- No correctness proofs needed?
- Even no programming needed?
- Is this possible?

→ Prolog (LP)!

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?

Let's develop a specification (and program):

```prolog
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X). ⇝ X=0; X=s(0); ...

?- nat_square(X,s(s(s(s(s(0)))))). ⇝ X=s(s(0))

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
Problem: calculate the squares of the naturals $< 5$. Show imperative program – is it correct?

Let’s develop a specification (and program):

```prolog
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X). \(\rightsquigarrow\) X=0; X=s(0);...

?- nat_square(X,s(s(s(s(s(0)))))). \(\rightsquigarrow\) X=s(s(0))

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?
Let’s develop a specification (and program):

```
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X).  \( \Rightarrow \)  X=0;X=s(0);...

?- nat_square(X,s(s(s(s(s(0)))))).  \( \Rightarrow \)  X=s(s(0))

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?

Let’s develop a specification (and program):

```prolog
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X).  \(\Rightarrow\)  X=0; X=s(s(0));...

?- nat_square(X,s(s(s(s(s(0)))))).  \(\Rightarrow\)  X=s(s(0))

(And show also a constraints version: we also have efficient arithmetic of course!)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?
Let’s develop a specification (and program):

```prolog
:- use_package(sr/bfall).  % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X).  \(\Rightarrow\) X=0; X=s(0); ...

?- nat_square(X,s(s(s(s(s(0)))))).  \(\Rightarrow\) X=s(s(0))

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?
Let’s develop a specification (and program):

:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).

?- output(X).
\[ X=0; X=s(s(0)); \ldots \]

?- nat_square(X,s(s(s(s(s(0)))))), nat_square(Y,X).
\[ X=s(s(0)) \]

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?
Let’s develop a specification (and program):

```
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X).
\[ \Rightarrow \quad X=0; X=s(0); \ldots \]

?- nat_square(X,s(s(s(s(s(0)))))).
\[ \Rightarrow \quad X=s(s(0)) \]

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?
Let’s develop a specification (and program):

```prolog
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

(And show also a constraints version: we also have efficient arithmetic of course!)
A Specification and also a Program

Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?
Let’s develop a specification (and program):

```prolog
% Use breadth-first search!
use_package(sr/bfall).
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).
```

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
**Problem: calculate the squares of the naturals** < 5. Show imperative program – is it correct?

Let’s develop a specification (and program):

```
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

(And show also a constraints version: we also have efficient arithmetic of course!)

(M. Hermenegildo, J.F. Morales, P. López-García – *How to best teach Prolog* (Prolog Education Meeting, Dec 12, 2023))
Problem: calculate the squares of the naturals $< 5$. Show imperative program – is it correct?

Let’s develop a specification (and program):

```prolog
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X). \(\Rightarrow\) X=0; X=s(0);...

?- nat_square(X,s(s(s(s(s(0)))))). \(\Rightarrow\) X=s(s(0))

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
Problem: calculate the squares of the naturals < 5. Show imperative program – is it correct?

Let’s develop a specification (and program): (click here to run)

```prolog
:- use_package(sr/bfall). % Use breadth-first search!
natural(0).
natural(s(X)) :- natural(X).

less(0,s(X)) :- natural(X).
less(s(X),s(Y)) :- less(X,Y).

add(0,Y,Y) :- natural(Y).
add(s(X),Y,s(Z)) :- add(X,Y,Z).

mult(0,Y,0) :- natural(Y).
mult(s(X),Y,Z) :- add(W,Y,Z), mult(X,Y,W).

nat_square(X,Y) :- natural(X), natural(Y), mult(X,X,Y).

output(X) :- natural(Y), less(Y,s(s(s(s(s(0)))))), nat_square(Y,X).
```

?- output(X).  \(\leadsto\) \(x=0; x=s(s(0));\ldots\)

?- nat_square(X,s(s(s(s(s(0)))))).  \(\leadsto\) \(x=s(s(0))\)

(And show also a constraints version: we also have efficient arithmetic of course!)

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023) 5
resistor(power,n1).
resistor(power,n2).

transistor(n2,ground,n1).
transistor(n3,n4,n2).
transistor(n5,ground,n4).

inverter(Input,Output) :-
    transistor(Input,ground,Output), resistor(power,Output).

nand_gate(Input1,Input2,Output) :-
    transistor(Input1,X,Output), transistor(Input2,ground,X),
    resistor(power,Output).

and_gate(Input1,Input2,Output) :-
    nand_gate(Input1,Input2,X), inverter(X, Output).

?- and_gate(In1,In2,Out) 连云
In1=n3, In2=n5, Out=n1
Circuit topology

resistor(power,n1).
resistor(power,n2).
transistor(n2,ground,n1).
transistor(n3,n4,n2).
transistor(n5,ground,n4).

\[ \text{inverter}(\text{Input}, \text{Output}) : - \\
\quad \text{transistor}(\text{Input}, \text{ground}, \text{Output}), \text{resistor}(\text{power}, \text{Output}). \]

\[ \text{nand gate}(\text{Input}_1, \text{Input}_2, \text{Output}) : - \\
\quad \text{transistor}(\text{Input}_1, \text{X}, \text{Output}), \text{transistor}(\text{Input}_2, \text{ground}, \text{X}), \text{resistor}(\text{power}, \text{Output}). \]

\[ \text{and gate}(\text{Input}_1, \text{Input}_2, \text{Output}) : - \\
\quad \text{nand gate}(\text{Input}_1, \text{Input}_2, \text{X}), \text{inverter}(\text{X}, \text{Output}). \]

?- and_gate(In1,In2,Out), In1=n3, In2=n5, Out=n1.
Circuit topology

resistor(power,n1).
resistor(power,n2).

transistor(n2,ground,n1).
transistor(n3,n4,n2).
transistor(n5,ground,n4).

inverter(Input,Output) :-
    transistor(Input,ground,Output), resistor(power,Output).
nand_gate(Input1,Input2,Output) :-
    transistor(Input1,X,Output), transistor(Input2,ground,X),
    resistor(power,Output).
and_gate(Input1,Input2,Output) :-
    nand_gate(Input1,Input2,X), inverter(X, Output).

?- and_gate(In1,In2,Out)
resistor(power,n1).
resistor(power,n2).

transistor(n2,ground,n1).
transistor(n3,n4,n2).
transistor(n5,ground,n4).

inverter(Input,Output) :-
transistor(Input,ground,Output), resistor(power,Output).
nand_gate(Input1,Input2,Output) :-
transistor(Input1,X,Output), transistor(Input2,ground,X),
resistor(power,Output).
and_gate(Input1,Input2,Output) :-
nand_gate(Input1,Input2,X), inverter(X, Output).

?- and_gate(In1,In2,Out)
\[ In1=n3, \ In2=n5, \ Out=n1 \]
How to best teach Prolog: Show the Beauty!

• But also explain the limits (expectation management):
  - discuss for what logics we have effective deduction procedures,
  - justify the choice of FOL, SLD-resolution, semi-decidability (see pictures later)
  → classical LP (Kowalski/Colmerauer).

• Show how logic programs are both logical theories (with declarative meaning) and procedural programs that can be debugged, followed step by step, etc.

• Show with examples (and benchmarking them) how you can go from executable specifications to efficient algorithms gradually, and as needed.
• But also explain the limits (expectation management):
  ▶ discuss for what logics we have effective deduction procedures,
  ▶ justify the choice of FOL, SLD-resolution, semi-decidability (see pictures later)
  → classical LP (Kowalski/Colmerauer).

• Show how logic programs are both logical theories (with declarative meaning) and procedural programs that can be debugged, followed step by step, etc.

• Show with examples (and benchmarking them) how you can go from executable specifications to efficient algorithms gradually, and as needed.
• But also explain the limits (expectation management):
  ▶ discuss for what logics we have effective deduction procedures,
  ▶ justify the choice of FOL, SLD-resolution, semi-decidability (see pictures later)
→ classical LP (Kowalski/Colmerauer).

• Show how logic programs are both logical theories (with declarative meaning) and procedural programs that can be debugged, followed step by step, etc.

• Show with examples (and benchmarking them) how you can go from executable specifications to efficient algorithms gradually, and as needed.
But also explain the limits (expectation management):
  - discuss for what logics we have effective deduction procedures,
  - justify the choice of FOL, SLD-resolution, semi-decidability (see pictures later)
  $\rightarrow$ classical LP (Kowalski/Colmerauer).

Show how logic programs are both logical theories (with declarative meaning) and procedural programs that can be debugged, followed step by step, etc.

Show with examples (and benchmarking them) how you can go from executable specifications to efficient algorithms gradually, and as needed.
How to best teach Prolog: Show the Beauty!

• But also explain the limits (expectation management):
  ▶ discuss for what logics we have effective deduction procedures,
  ▶ justify the choice of FOL, SLD-resolution, semi-decidability (see pictures later)
→ classical LP (Kowalski/Colmerauer).

• Show how logic programs are both logical theories (with declarative meaning) and procedural programs that can be debugged, followed step by step, etc.

• Show with examples (and benchmarking them) how you can go from executable specifications to efficient algorithms gradually, and as needed.
How to best teach Prolog: Show the Beauty!

- But also explain the limits (expectation management):
  - discuss for what logics we have effective deduction procedures,
  - justify the choice of FOL, SLD-resolution, semi-decidability (see pictures later)
  → classical LP (Kowalski/Colmerauer).

- Show how logic programs are both logical theories (with declarative meaning) and procedural programs that can be debugged, followed step by step, etc.

- Show with examples (and benchmarking them) how you can go from executable specifications to efficient algorithms gradually, and as needed.
The modulo operation, \( \text{mod}(X, Y, Z) \) where \( Z \) is the remainder from dividing \( X \) by \( Y \):

\[ \exists Q \text{ s.t. } X = Y \times Q + Z \land Z < Y \]
The modulo operation, \( \text{mod}(X, Y, Z) \) where \( Z \) is the remainder from dividing \( X \) by \( Y \):

\[
\exists Q \text{ s.t. } X = Y \times Q + Z \land Z < Y
\]

We can express this definition/specification directly in Prolog!

```prolog
mod(X, Y, Z) :-
mult(Y, Q, W), add(W, Z, X), less(Z, Y).
```

run
The modulo operation, \( \text{mod}(X, Y, Z) \) where \( Z \) is the remainder from dividing \( X \) by \( Y \):

\[
\exists Q \text{ s.t. } X = Y \times Q + Z \land Z < Y
\]

We can express this definition/specification directly in Prolog!

```prolog
mod(X, Y, Z) :-
    mult(Y, Q, W),
    add(W, Z, X),
    less(Z, Y).
```

?- op(500, fy, s).
yes
?- mod(X, Y, s 0).
X = s 0,
Y = s s 0 ? ;
X = s 0,
Y = s s s 0 ? ;
X = s s s 0,
Y = s s 0 ? ;
...
The modulo operation, \( \text{mod}(X,Y,Z) \) where \( Z \) is the remainder from dividing \( X \) by \( Y \):

\[ \exists Q \text{ s.t. } X = Y \times Q + Z \land Z < Y \]

We can express this definition/specification directly in Prolog!

```prolog
mod(X,Y,Z) :-
mult(Y,Q,W), add(W,Z,X), less(Z, Y).
```

?- op(500, fy, s).
yes
?- mod(X,Y, s 0).
  X = s 0,
  Y = s s 0 ? ;
  X = s 0,
  Y = s s s 0 ? ;
  X = s s 0,
  Y = s s 0 ? ;
  ...
```

Or write a more efficient version, also within (pure) Prolog:

```prolog
mod(X,Y,X) :- less(X, Y).
mod(X,Y,Z) :- add(X1,Y,X), mod(X1,Y,Z).
```

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
The modulo operation, \( \text{mod}(X,Y,Z) \) where \( Z \) is the remainder from dividing \( X \) by \( Y \):

\[
\exists Q \text{ s.t. } X = Y \times Q + Z \land Z < Y
\]

We can express this definition/specification directly in Prolog:

```prolog
mod(X,Y,Z) :-
    mult(Y,Q,W), add(W,Z,X), less(Z, Y).
```

?- op(500,fy,s).
yes
?- mod(X,Y, s 0).
X = s 0,
Y = s s 0 ? ;
X = s 0,
Y = s s s 0 ? ;
X = s s s 0,
Y = s s 0 ? ;
...
```

Or write a more efficient version, also within (pure) Prolog:

```prolog
mod(X,Y,X) :- less(X, Y).
mod(X,Y,Z) :- add(X1,Y,X), mod(X1,Y,Z).
```

?- mod(s(s(s(s(s(0))))), s(s(0)), R).
R = s(0) ?
The modulo operation, \( \text{mod}(X, Y, Z) \) where \( Z \) is the remainder from dividing \( X \) by \( Y \):

\[
\exists Q \text{ s.t. } X = Y \times Q + Z \land Z < Y
\]

We can express this definition/specification directly in Prolog!

```prolog
mod(X, Y, Z) :-
    mult(Y, Q, W),
    add(W, Z, X),
    less(Z, Y).
```

```
?- op(500, fy, s).
yes
?- mod(X, Y, s 0).
X = s 0,
Y = s s 0 ? ;
X = s 0,
Y = s s s 0 ? ;
X = s s s 0,
Y = s s s 0 ? ;
...
```

Or write a more efficient version, also within (pure) Prolog:

```prolog
mod(X, Y, X) :-
    less(X, Y).
mod(X, Y, Z) :-
    add(X1, Y, X),
    mod(X1, Y, Z).
```

```
?- mod(s(s(s(s(s(0))))), s(s(0)), R).
R = s(0) ?
```

Again, we can also show the constraints version.

And we can discuss modes and how they affect determinacy, cost, termination, etc.
• Show how unification is also a device for constructing and matching complex data structures with (declarative) pointers. Show it in the top level, giving “the data structures class.”

?- X=f(K, g(K)),
   Y=a,
   Z=g(L),
   W=h(b, L),
   % Heap memory at this point  →
   p(X, Y, Z, W).

• Do use types (and properties in general): define them as predicates, show them used to check if something is in the type (dynamic checking), or “run backwards” to generate the “inhabitants”; LP has property-based testing for free!

natlist([]).
natlist([H|T]) :- natural(H), natlist(T).
Show how unification is also a device for constructing and matching complex data structures with (declarative) pointers. Show it in the top level, giving “the data structures class.”

?- X = f(K, g(K)),
   Y = a,
   Z = g(L),
   W = h(b, L),
   % Heap memory at this point →
   p(X, Y, Z, W).

Do use types (and properties in general): define them as predicates, show them used to check if something is in the type (dynamic checking), or “run backwards” to generate the “inhabitants”; LP has property-based testing for free!

```
natlist([]).   
natlist([H|T]) :- natural(H), natlist(T).
```
How to best teach Prolog: Show the Beauty!

- Show the (3-line) meta-interpreter + an adorned one.
  - It is a thing of beauty.
  - An excellent demonstrator of the unique powers of Prolog.

- Use motivational examples that involve search (puzzles, etc.).
  - it is a unique characteristic of the language
  and give advice on how to control it.

- Incomplete data structures, automata, DCGs ... (and run them backwards as generators of course!)

- Show that there are plenty of interfaces to other languages, data representations, etc.
**Dispel unfounded myths** about the language, and show that many of the shortcomings of early Prologs have been *addressed over the years*.

- **Explaining termination:**
  
  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  
  - Use/build system to run *alternatively and selectively* in breadth-first, iterative deepening, tabling, etc.
  - Start by running all predicates, e.g., breadth-first – everything works!
  - Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.
Dispel unfounded myths about the language, and show that many of the shortcomings of early Prologs have been addressed over the years.

- Explaining termination:

  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  
  - Use/build system to run *alternatively and selectively* in breadth-first, iterative deepening, tabling, etc.
  - Start by running all predicates, e.g., breadth-first — everything works!
  - Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.
**Dispel unfounded myths** about the language, and show that many of the shortcomings of early Prologs have been *addressed over the years.*

- **Explaining termination:**

  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  - Use/build system to run alternatively and selectively in breadth-first, iterative deepening, tabling, etc.
  - Start by running all predicates, e.g., breadth-first — everything works!
  - Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.
How to best teach Prolog: Dispelling Myths and Misconceptions

Dispel unfounded myths about the language, and show that many of the shortcomings of early Prologs have been addressed over the years.

- Explaining termination:

  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  - Use/build system to run *alternatively and selectively* in breadth-first, iterative deepening, tabling, etc.
  - Start by running all predicates, e.g., breadth-first – everything works!
  - Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.
How to best teach Prolog: Dispelling Myths and Misconceptions

Dispel unfounded myths about the language, and show that many of the shortcomings of early Prologs have been *addressed over the years*.

- Explaining termination:
  
  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  
  ▶️ Use/build system to run *alternatively and selectively* in breadth-first, iterative deepening, tabling, etc.
  ▶️ Start by running all predicates, e.g., breadth-first – everything works!
  ▶️ Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.
How to best teach Prolog: Dispelling Myths and Misconceptions

Dispel unfounded myths about the language, and show that many of the shortcomings of early Prologs have been *addressed over the years*.

- Explaining termination:

  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  
  ▶ Use/build system to run *alternatively and selectively* in breadth-first, iterative deepening, tabling, etc.
  ▶ Start by running all predicates, e.g., breadth-first – everything works!
  ▶ Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.
Characterization of the search tree

```
M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog  (Prolog Education Meeting, Dec 12, 2023) 12
```
Depth-First Search

[Diagram showing a tree structure with nodes labeled as "solution", "fail", and "infinite failure".]

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
Breadth-First Search

solution

fail

solution

fail

solution

fail

infinite failure

solution

fail

solution

fail

solution

fail

infinite failure

solution

fail

solution

fail

solution

fail

infinite failure

solution

fail

solution

fail

solution

fail

infinite failure

solution

fail

solution

fail

solution

fail

infinite failure

solution

fail

solution

fail

solution

fail

infinite failure
**Dispel unfounded myths** about the language, and show that many of the shortcomings of early Prologs have been *addressed over the years.*

- **Explaining termination:**

  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:

  - Use/build system to run *alternatively and selectively* in breadth-first, iterative deepening, tabling, etc.
  - Start by running all predicates, e.g., breadth-first – everything works!
  - Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.

  - Do relate semi-decidability to the *halting problem:* no-one (Prolog, logic, nor other Turing-complete prog. language) can solve that (but tabling helps: good time to introduce it!).

  - Discuss advantages and disadvantages of search rules (time, memory). Motivate the choices made for Prolog benchmarking actual executions.
Dispel unfounded myths about the language, and show that many of the shortcomings of early Prologs have been addressed over the years.

Explaining termination:

Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:

- Use/build system to run alternatively and selectively in breadth-first, iterative deepening, tabling, etc.
- Start by running all predicates, e.g., breadth-first – everything works!
- Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.

- Do relate semi-decidability to the halting problem: no-one (Prolog, logic, nor other Turing-complete prog. language) can solve that (but tabling helps: good time to introduce it!).

- Discuss advantages and disadvantages of search rules (time, memory).
  Motivate the choices made for Prolog benchmarking actual executions.
How to best teach Prolog: Dispelling Myths and Misconceptions

Dispel unfounded myths about the language, and show that many of the shortcomings of early Prologs have been addressed over the years.

- Explaining termination:
  
  Non-termination is a fact of life for any powerful programming language or proof system. However, it is likely to discourage beginners if not explained well:
  
  ▶ Use/build system to run alternatively and selectively in breadth-first, iterative deepening, tabling, etc.
  
  ▶ Start by running all predicates, e.g., breadth-first – everything works!
  
  ▶ Then, explain the shape of the tree (solutions at finite depth, possible infinite failures, etc.), and thus why breadth-first works, and why depth-first sometimes may not.

  ▶ Do relate semi-decidability to the halting problem: no-one (Prolog, logic, nor other Turing-complete prog. language) can solve that (but tabling helps: good time to introduce it!).

  ▶ Discuss advantages and disadvantages of search rules (time, memory). Motivate the choices made for Prolog benchmarking actual executions.
• Showing that Prolog arithmetic can also be reversible:
  ▶ We show first Peano arithmetic: beautiful and only needs pure LP, but slow.
  ▶ We also show (arithmetic) constraint domains: beautiful and efficient!
  ▶ We justify uses of ISO arithmetic for efficiency.

• The occur check is available (if needed):
  ▶ Explain why, and that there is a built-in for it.
  ▶ Have a package (expansion) that calls it by default for all unifications.
  ▶ Explain the existence of infinite tree unification (as a constraint domain).

• Prolog can be pure (despite cut, assert, etc.):
  ▶ Have a pure mode in the implementation so that impure built-ins are simply not present.
  ▶ Develop pure libraries.
  ▶ Develop purer built-ins that can be loaded alternatively.

and also accept that impurity is necessary sometimes, but we keep it as encapsulated as possible.
How to best teach Prolog: Dispelling Myths and Misconceptions

• Showing that Prolog arithmetic can also be reversible:
  ▶ We show first Peano arithmetic: beautiful and only needs pure LP, but slow.
  ▶ We also show (arithmetic) constraint domains: beautiful and efficient!
  ▶ We justify uses of ISO arithmetic for efficiency.

• The occur check is available (if needed):
  ▶ Explain why, and that there is a built-in for it.
  ▶ Have a package (expansion) that calls it by default for all unifications.
  ▶ Explain the existence of infinite tree unification (as a constraint domain).

• Prolog can be pure (despite cut, assert, etc.):
  ▶ Have a pure mode in the implementation so that impure built-ins are simply not present.
  ▶ Develop pure libraries.
  ▶ Develop purer built-ins that can be loaded alternatively.
  and also accept that impurity is necessary sometimes, but we keep it as encapsulated as possible.
• Showing that Prolog arithmetic can also be reversible:
  ▶ We show first Peano arithmetic: beautiful and only needs pure LP, but slow.
  ▶ We also show (arithmetic) constraint domains: beautiful and efficient!
  ▶ We justify uses of ISO arithmetic for efficiency.

• The occur check is available (if needed):
  ▶ Explain why, and that there is a built-in for it.
  ▶ Have a package (expansion) that calls it by default for all unifications.
  ▶ Explain the existence of infinite tree unification (as a constraint domain).

• Prolog can be pure (despite cut, assert, etc.):
  ▶ Have a pure mode in the implementation so that impure built-ins are simply not present.
  ▶ Develop pure libraries.
  ▶ Develop purer built-ins that can be loaded alternatively.

  and also accept that impurity is necessary sometimes, but we keep it as encapsulated as possible.
• Showing that Prolog arithmetic can also be reversible:
  ► We show first Peano arithmetic: beautiful and only needs pure LP, but slow.
  ► We also show (arithmetic) constraint domains: beautiful and efficient!
  ► We justify uses of ISO arithmetic for efficiency.

• The occur check is available (if needed):
  ► Explain why, and that there is a built-in for it.
  ► Have a package (expansion) that calls it by default for all unifications.
  ► Explain the existence of infinite tree unification (as a constraint domain).

• Prolog can be pure (despite cut, assert, etc.):
  ► Have a pure mode in the implementation so that impure built-ins are simply not present.
  ► Develop pure libraries.
  ► Develop purer built-ins that can be loaded alternatively.
  and also accept that impurity is necessary sometimes, but we keep it as encapsulated as possible.
How to best teach Prolog: Dispelling Myths and Misconceptions

- **Negation:**
  - Explain negation as failure devoting time to discuss limitations.
  - Can also go into other types of negation, s(CASP), etc.

- **Prolog has many applications and uses.**
  - Show the many examples of impressive applications (cf. Prolog Year/Book).

- **Prolog is in many ways as other languages, but adds unique, useful features.**
  - Show that Prolog *subsumes* functional and imperative programming (after SSA). It is *simply more*.
    - (Useful for analysis of other languages!)
  - It is “standard” if used in one direction and there is only one definition per procedure.
  - But it can also have several definitions, search, run backwards, etc.
  - In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of *backwards continuations* to go if there is a failure (previous choice point).
Negation:

- Explain negation as failure devoting time to discuss limitations.
- Can also go into other types of negation, s(CASP), etc.

Prolog has many applications and uses.

- Show the many examples of impressive applications (cf. Prolog Year/Book).

Prolog is in many ways as other languages, but adds unique, useful features.

- Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more. (Useful for analysis of other languages!)
- It is “standard” if used in one direction and there is only one definition per procedure.
- But it can also have several definitions, search, run backwards, etc.
- In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
Negation:
- Explain negation as failure devoting time to discuss limitations.
- Can also go into other types of negation, s(CASP), etc.

Prolog has many applications and uses.
- Show the many examples of impressive applications (cf. Prolog Year/Book).

Prolog is in many ways as other languages, but adds unique, useful features.
- Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more.
  (Useful for analysis of other languages!)
- It is “standard” if used in one direction and there is only one definition per procedure.
- But it can also have several definitions, search, run backwards, etc.
- In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
• Negation:
  ▶ Explain negation as failure devoting time to discuss limitations.
  ▶ Can also go into other types of negation, s(CASP), etc.

• Prolog has many applications and uses.
  ▶ Show the many examples of impressive applications (cf. Prolog Year/Book).

• Prolog is in many ways as other languages, but adds unique, useful features.
  ▶ Show that Prolog *subsumes* functional and imperative programming (after SSA). It is *simply more.* (Useful for analysis of other languages!)
  ▶ It is “standard” if used in one direction and there is only one definition per procedure.
  ▶ But it can also have several definitions, search, run backwards, etc.
  ▶ In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
• Negation:
  ▶ Explain negation as failure devoting time to discuss limitations.
  ▶ Can also go into other types of negation, s(CASP), etc.

• Prolog has many applications and uses.
  ▶ Show the many examples of impressive applications (cf. Prolog Year/Book).

• Prolog is in many ways as other languages, but adds unique, useful features.
  ▶ Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more.
    (Useful for analysis of other languages!)
  ▶ It is “standard” if used in one direction and there is only one definition per procedure.
  ▶ But it can also have several definitions, search, run backwards, etc.
  ▶ In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
How to best teach Prolog: Dispelling Myths and Misconceptions

• Negation:
  ▶ Explain negation as failure devoting time to discuss limitations.
  ▶ Can also go into other types of negation, s(CASP), etc.

• Prolog has many applications and uses.
  ▶ Show the many examples of impressive applications (cf. Prolog Year/Book).

• Prolog is in many ways as other languages, but adds unique, useful features.
  ▶ Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more.
    (Useful for analysis of other languages!)
  ▶ It is “standard” if used in one direction and there is only one definition per procedure.
  ▶ But it can also have several definitions, search, run backwards, etc.
  ▶ In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
How to best teach Prolog: Dispelling Myths and Misconceptions

• Negation:
  ▶ Explain negation as failure devoting time to discuss limitations.
  ▶ Can also go into other types of negation, s(CASP), etc.

• Prolog has many applications and uses.
  ▶ Show the many examples of impressive applications (cf. Prolog Year/Book).

• Prolog is in many ways as other languages, but adds unique, useful features.
  ▶ Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more. (Useful for analysis of other languages!)
  ▶ It is “standard” if used in one direction and there is only one definition per procedure.
  ▶ But it can also have several definitions, search, run backwards, etc.
  ▶ In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
How to best teach Prolog: Dispelling Myths and Misconceptions

• Negation:
  ▶ Explain negation as failure devoting time to discuss limitations.
  ▶ Can also go into other types of negation, s(CASP), etc.

• Prolog has many applications and uses.
  ▶ Show the many examples of impressive applications (cf. Prolog Year/Book).

• Prolog is in many ways as other languages, but adds unique, useful features.
  ▶ Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more. (Useful for analysis of other languages!)
  ▶ It is “standard” if used in one direction and there is only one definition per procedure.
  ▶ But it can also have several definitions, search, run backwards, etc.
  ▶ In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
How to best teach Prolog: Dispelling Myths and Misconceptions

- **Negation:**
  - Explain negation as failure devoting time to discuss limitations.
  - Can also go into other types of negation, s(CASP), etc.

- **Prolog has many applications and uses.**
  - Show the many examples of impressive applications (cf. Prolog Year/Book).

- **Prolog is in many ways as other languages, but adds unique, useful features.**
  - Show that Prolog *subsumes* functional and imperative programming (after SSA). It is *simply more*. (Useful for analysis of other languages!)
  - It is “standard” if used in one direction and there is only one definition per procedure.
  - But it can also have several definitions, search, run backwards, etc.
  - In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
How to best teach Prolog: Dispelling Myths and Misconceptions

- Negation:
  - Explain negation as failure devoting time to discuss limitations.
  - Can also go into other types of negation, s(CASP), etc.

- Prolog has many applications and uses.
  - Show the many examples of impressive applications (cf. Prolog Year/Book).

- Prolog is in many ways as other languages, but adds unique, useful features.
  - Show that Prolog subsumes functional and imperative programming (after SSA). It is simply more. (Useful for analysis of other languages!)
  - It is “standard” if used in one direction and there is only one definition per procedure.
  - But it can also have several definitions, search, run backwards, etc.
  - In addition to a stack of forward continuations (as in any language, for procedure return), also a stack of backwards continuations to go if there is a failure (previous choice point).
Show that Prolog can support functional **syntax** (sometimes more compact):

```
grandparent(X, ~parent(~parent(X))).  \rightarrow
grandparent(X, ~parent(Z)) :- parent(X,Z).  \rightarrow
grandparent(X,Y) :- parent(X,Z), parent(Z,Y).
```

I.e., read ~ as “last argument of”; \rightarrow as “is expanded to.”

```
?− E = ~append(~append(A,B),D).  \rightarrow
?− append(A,B,C), E = ~append(C,D).  \rightarrow
?− append(A,B,C), append(C,D,E).
```

```
list := [] | [_|~list].  \rightarrow
list([]).
list([_|X]) :- list(X).
```

Same with loops, mutable variables/assignment, etc.

Show that Prolog can also have types (and modes, assertions, etc.) *if needed.*

And of course show that Prolog is fast, can be compiled and generate standard executables, has tests, auto-documenters, linters, and great environments in general.
• Show that Prolog can support functional **syntax** (sometimes more compact):

  grandparent(X, ▼parent(▼parent(X))). ▼▼▼
  grandparent(X, ▼parent(Z)) :- parent(X, Z). ▼▼▼
  grandparent(X, Y) :- parent(X, Z), parent(Z, Y).

  i.e., read ▼ as “last argument of”; ▼▼▼ as “is expanded to.”

• Same with loops, mutable variables/assignment, etc.

• Show that Prolog can also have types (and modes, assertions, etc.) *if needed.*

• And of course show that Prolog is fast, can be compiled and generate standard executables, has tests, auto-documenters, linters, and great environments in general.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    - Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    - (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
      - Can be attractive for beginners, young students.
      - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
      - Server-based vs. browser-based.
      - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, \(\tau\)-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, $\tau$-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, \(\tau\)-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)

- Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
  - Server-based vs. browser-based.
  - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    - Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    - (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
      - Can be attractive for beginners, young students.
      - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
      - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use. Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

● System types:
  ▶ Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  ▶ Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
    ● Can be attractive for beginners, young students.
    ● Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    ● Server-based vs. browser-based.
    ● Very useful for executable examples in manuals and tutorials.

Offer both types to students!

● Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  ▶ Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
● Ideally the system should allow covering:
  ▶ pure LP (with several search rules, tabling),
  ▶ ISO-Prolog,
  ▶ higher-order support and functional syntax,
  ▶ constraints,
  ▶ ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    - Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

• System types:
  ► Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  ► Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
    ● Can be attractive for beginners, young students.
    ● Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    ● Server-based vs. browser-based.
    ● Very useful for executable examples in manuals and tutorials.

Offer both types to students!
• Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  ► Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
• Ideally the system should allow covering:
  ► pure LP (with several search rules, tabling),
  ► ISO-Prolog,
  ► higher-order support and functional syntax,
  ► constraints,
  ► ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, τ-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, $\tau$-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

• System types:
  ▶ Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  ▶ Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, $\tau$-Prolog, SICStus+Jupyter).
    ● Can be attractive for beginners, young students.
    ● Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    ● Server-based vs. browser-based.
    ● Very useful for executable examples in manuals and tutorials.

Offer both types to students!

• Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  ▶ Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).

• Ideally the system should allow covering:
  ▶ pure LP (with several search rules, tabling),
  ▶ ISO-Prolog,
  ▶ higher-order support and functional syntax,
  ▶ constraints,
  ▶ ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    - Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    - (e.g., Ciao Playground/Active Logic Documents, SWISH, $\tau$-Prolog, SICStus+Jupyter).
      - Can be attractive for beginners, young students.
      - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
      - Server-based vs. browser-based.
      - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

• System types:
  ► Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  ► Playgrounds and notebooks –see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, \(\tau\)-Prolog, SICStus+Jupyter).
    ◦ Can be attractive for beginners, young students.
    ◦ Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    ◦ Server-based vs. browser-based.
    ◦ Very useful for executable examples in manuals and tutorials.

Offer both types to students!
• Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  ► Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
• Ideally the system should allow covering:
  ► pure LP (with several search rules, tabling),
  ► ISO-Prolog,
  ► higher-order support and functional syntax,
  ► constraints,
  ► ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.
How to best teach Prolog

- System types:
  - Classical installation: most appropriate for more advanced students / “real” use.
    - Show serious, competitive language.
  - Playgrounds and notebooks –see PL50 Book papers!
    - (e.g., Ciao Playground/Active Logic Documents, SWISH, $\tau$-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F.Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in https://cliplab.org/logalg.

M. Hermenegildo, J.F. Morales, P. López-García – How to best teach Prolog (Prolog Education Meeting, Dec 12, 2023)
How to best teach Prolog

- **System types:**
  - Classical installation: most appropriate for more advanced students / “real” use.
    Show serious, competitive language.
  - Playgrounds and notebooks – see PL50 Book papers!
    (e.g., Ciao Playground/Active Logic Documents, SWISH, $\tau$-Prolog, SICStus+Jupyter).
    - Can be attractive for beginners, young students.
    - Some places (e.g., schools) may not have personnel/machines for installation, but will have a tablet.
    - Server-based vs. browser-based.
    - Very useful for executable examples in manuals and tutorials.

Offer both types to students!
- Block-based versions can be useful starters for youngest (cf. Laura Cecchi’s paper in PL50 Book)
  - Also, nice tool for kids developed by J.F. Morales and S. Abreu for the Prolog Year (see PL50 Book).
- Ideally the system should allow covering:
  - pure LP (with several search rules, tabling),
  - ISO-Prolog,
  - higher-order support and functional syntax,
  - constraints,
  - ASP/s(CASP), etc.

Much more in the PL50 book papers and teaching materials in [https://cliplab.org/logalg](https://cliplab.org/logalg).