50 Years of Prolog and Beyond

Manuel Hermenegildo[1,2]

The Prolog Year
Prolog Day Symposium, November 10, 2022

[1]T. U. of Madrid (UPM)
[2]IMDEA Software Institute

*Part of the contents of this talk appear in the recent TPLP paper "**50 years of Prolog and Beyond**," by Philipp Körner, Michael Leuschel, João Barbosa, Vítor Santos Costa, Verónica Dahl, Manuel V. Hermenegildo, Jose F. Morales, Jan Wielemaker, Daniel Diaz, Salvador Abreu, and Giovanni Ciatto written for Prolog's 50th anniversary and TPLP's 20th anniversary.*

Also big thanks to Bob Kowalski for historical input, feedback, and permanent inspiration.

Prolog is 50

- So, then, Prolog is 50!
  - What, 50 years?!? Half a century?!?!
  - Is Prolog therefore now 'old'?
- Actually... continued interest:
  - Many *active implementations*, and *more appearing* continuously.
  - TIOBE index of programming languages shows Prolog:
    - In upper 10% of all languages tracked (270).
    - Stable, even somewhat upward trend since 2012.
    - One of only 13 languages that are tracked 'long term'.

But, what is Prolog?

Prolog is 50

- So, then, Prolog is 50!
  - What, 50 years?!? Half a century?!?!
  - Is Prolog therefore now 'old'?
- Actually... continued interest:
  - Many *active implementations*, and *more appearing* continuously.
  - TIOBE index of programming languages shows Prolog:
    - In upper 10% of all languages tracked (270).
    - Stable, even somewhat upward trend since 2012.
    - One of only 13 languages that are tracked 'long term'.

But, what is Prolog?

Prolog is 50

- So, then, Prolog is 50!
  - What, 50 years?!? Half a century?!?!
  - Is Prolog therefore now 'old'?
- Actually... continued interest:
  - Many *active implementations*, and *more appearing* continuously.
  - TIOBE index of programming languages shows Prolog:
    - In upper 10% of all languages tracked (270).
    - Stable, even somewhat upward trend since 2012.
    - One of only 13 languages that are tracked 'long term'.

But, what is Prolog?

Prolog is 50

- So, then, Prolog is 50!
  - What, 50 years?!? Half a century?!?!
  - Is Prolog therefore now 'old'?
- Actually... continued interest:
  - Many *active implementations*, and *more appearing* continuously.
  - TIOBE index of programming languages shows Prolog:
    - In upper 10% of all languages tracked (270).
    - Stable, even somewhat upward trend since 2012.
    - One of only 13 languages that are tracked 'long term'.

But, what is Prolog?

Prolog is 50

- So, then, Prolog is 50!
  - What, 50 years?!? Half a century?!?!
  - Is Prolog therefore now 'old'?
- Actually... continued interest:
  - Many *active implementations*, and *more appearing* continuously.
  - TIOBE index of programming languages shows Prolog:
    - In upper 10% of all languages tracked (270).
    - Stable, even somewhat upward trend since 2012.
    - One of only 13 languages that are tracked 'long term'.

But, what is Prolog?

What is Prolog? Why is it important?

Prolog is an acronym of two words:

Programming
and
Logic

- What is the best way to **program** computers?
  I.e., how do we get them to solve problems and/or do what we need?
- How can **logic** help us in this task?

What is Prolog? Why is it important?

Prolog is an acronym of two words:

Programming
and
Logic

- What is the best way to **program** computers?
  I.e., how do we get them to solve problems and/or do what we need?

- How can **logic** help us in this task?

What is Prolog? Why is it important?

Prolog is an acronym of two words:

Programming
and
Logic

- What is the best way to **program** computers?
  I.e., how do we get them to solve problems and/or do what we need?

- How can **logic** help us in this task?

A New View of Computing

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

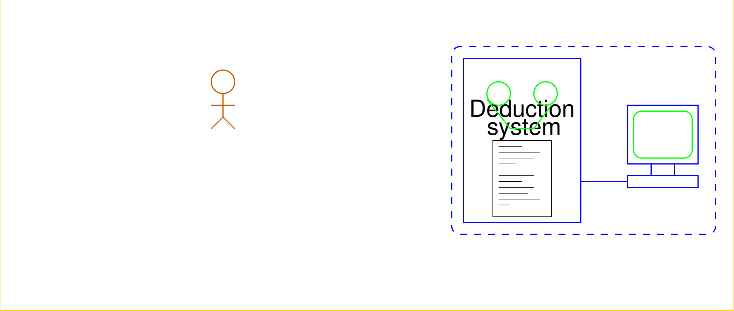
  - First: program once and for all this *deduction procedure* in the computer,
  - Then, for each problem we want to solve:
    - Find a suitable *representation* for the problem.
    - Then, to obtain solutions, ask questions and let deduction procedure do rest:

A New View of Computing

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

  - First: program once and for all this *deduction procedure* in the computer,
  - Then, for each problem we want to solve:
    - Find a suitable *representation* for the problem.
    - Then, to obtain solutions, ask questions and let deduction procedure do rest:

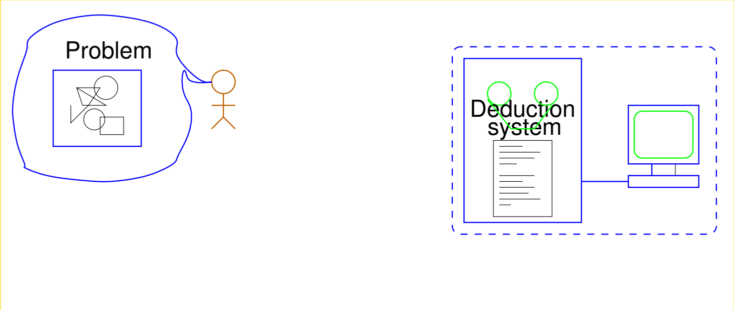


A New View of Computing

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

  - First: program once and for all this *deduction procedure* in the computer,
  - Then, for each problem we want to solve:
    - Find a suitable *representation* for the problem.
    - Then, to obtain solutions, ask questions and let deduction procedure do rest:

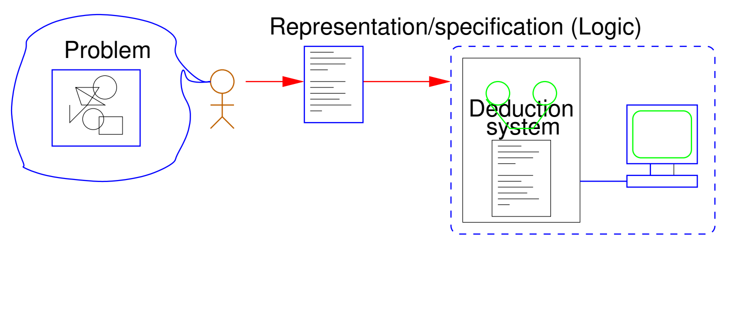


A New View of Computing

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

  - First: program once and for all this *deduction procedure* in the computer,
  - Then, for each problem we want to solve:
    - Find a suitable *representation* for the problem.
    - Then, to obtain solutions, ask questions and let deduction procedure do rest:

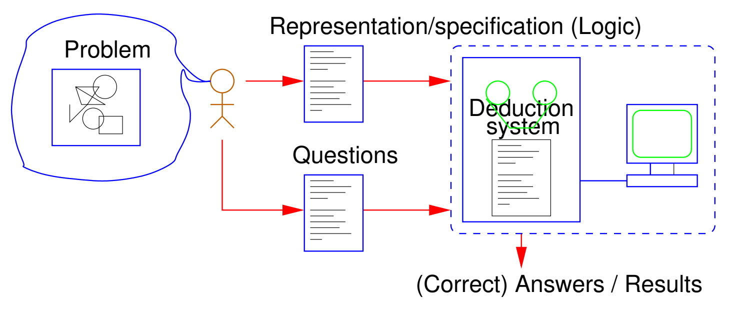


A New View of Computing

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

- First: program once and for all this *deduction procedure* in the computer,
- Then, for each problem we want to solve:
  - Find a suitable *representation* for the problem.
  - Then, to obtain solutions, ask questions and let deduction procedure do rest:

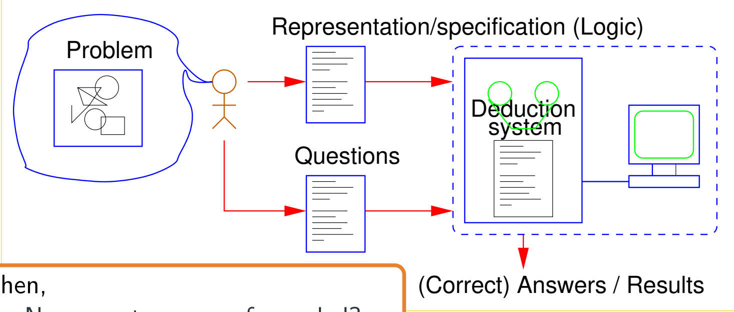


A New View of Computing

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

- ▶ First: program once and for all this *deduction procedure* in the computer,
- ▶ Then, for each problem we want to solve:
    - Find a suitable *representation* for the problem.
    - Then, to obtain solutions, ask questions and let deduction procedure do rest:



But then,

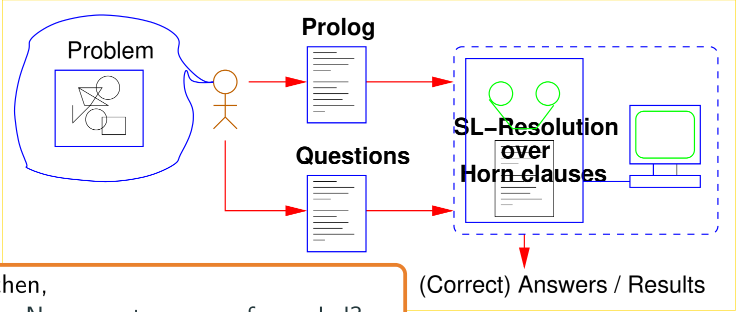
- No correctness proofs needed?
- Even no programming needed?
- Is this possible?

Prolog is the Materialization of this Dream!

- If we have an *effective mechanical proof method*.

⇝ *a new view of problem solving and computing is possible*:

  - First: program once and for all this *deduction procedure* in the computer,
  - Then, for each problem we want to solve:
    - Find a suitable *representation* for the problem.
    - Then, to obtain solutions, ask questions and let deduction procedure do rest:

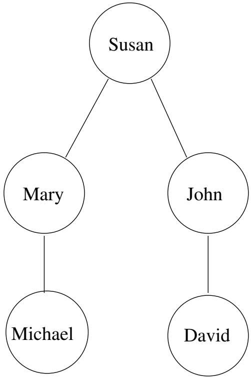


But then,

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

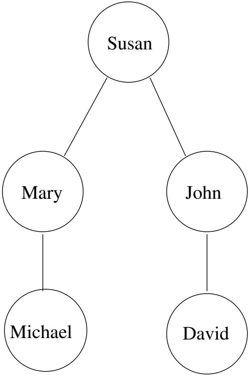
Family relations

Susan is the mother of Mary.
Susan is the mother of John.
Mary is the mother of Michael.



Family relations

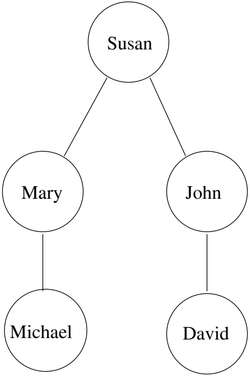
```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```



Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

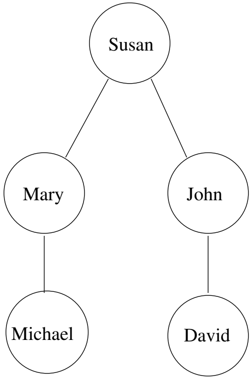
John is the father of David.



Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

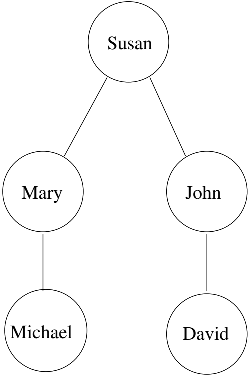


Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

One is the grandmother of someone else if one is the mother of the mother (or father) of that other person.

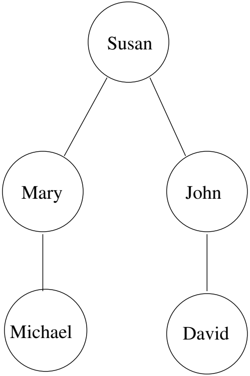


Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```



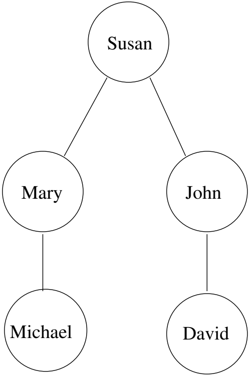
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
```



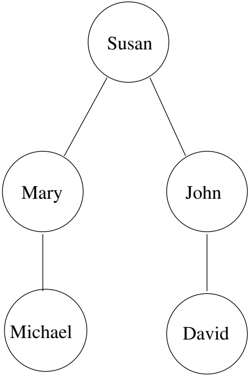
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
```



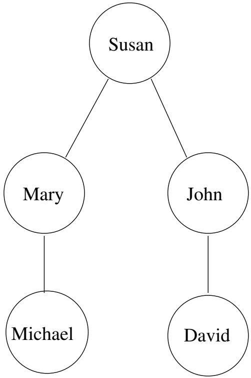
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
```



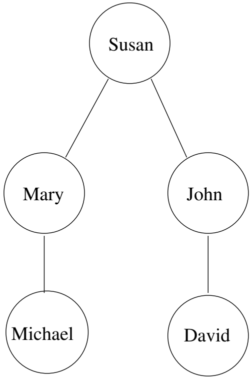
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
?- mother_of(X,mary).
```



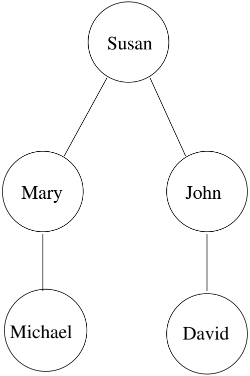
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
?- mother_of(X,mary).
X = susan ? ;
```



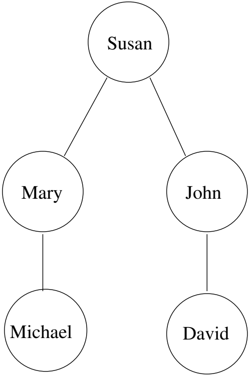
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
?- mother_of(X,mary).
X = susan ? ;
no
```



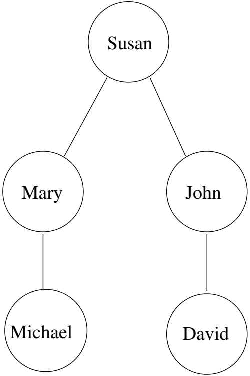
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
?- mother_of(X,mary).
X = susan ? ;
no
?- grandmother_of(X,Y).
```



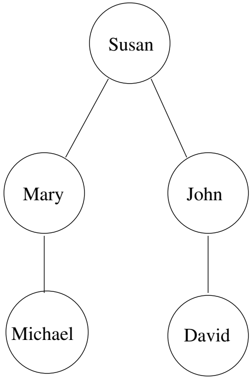
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
?- mother_of(X,mary).
X = susan ? ;
no
?- grandmother_of(X,Y).
X = susan,
Y = michael ? ;
```



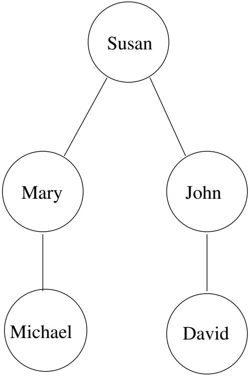
Family relations

```
mother_of(susan, mary).
mother_of(susan, john).
mother_of(mary, michael).
```

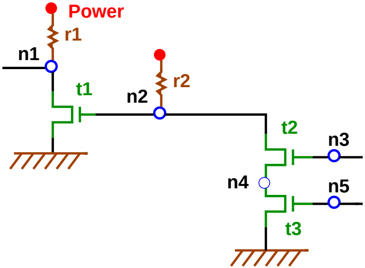
```
father_of(john, david).
```

```
grandmother_of(X,Y) :-
    mother_of(X,Z), mother_of(Z,Y).
grandmother_of(X,Y) :-
    mother_of(X,Z), father_of(Z,Y).
```

```
?- mother_of(susan,Y).
Y = mary ? ;
Y = john ? ;
no
?- mother_of(X,mary).
X = susan ? ;
no
?- grandmother_of(X,Y).
X = susan,
Y = michael ? ;
X = susan,
Y = david ? ;
no
```



Circuit topology

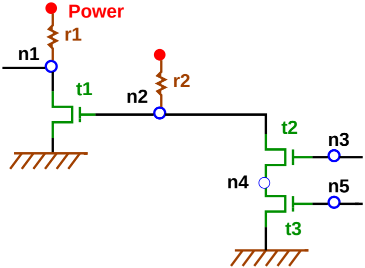



```
resistor(power,n1).
resistor(power,n2).

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)` ⇝ `In1=n3, In2=n5, Out=n1`

Circuit topology

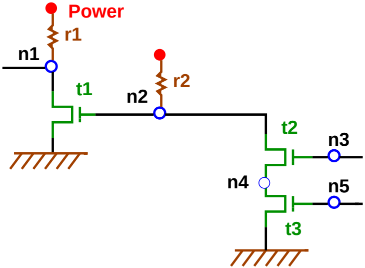



```
resistor(power,n1).
resistor(power,n2).

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)` ⇝ `In1=n3, In2=n5, Out=n1`

Circuit topology



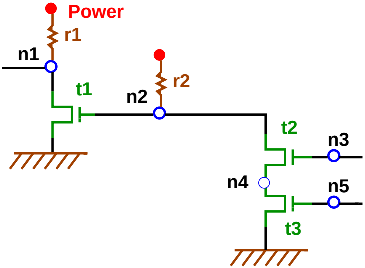

```
resistor(power,n1).
resistor(power,n2).

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)
```

⇝ 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)` ⇝ `In1=n3, In2=n5, Out=n1`

From specifications to efficient programs

"Max is the maximum element of a set if there is no element in the set that is larger than it."

$$max(L, Max) \leftarrow Max \in L \ \wedge \ \nexists E \mid E \in L \ \wedge \ E > Max$$

```
max(L,Max) :-
    member(Max,L),
    \+ (member(E,L), E>Max).
```

From specifications to efficient programs

"Max is the maximum element of a set if there is no element in the set that is larger than it."

$$max(L, Max) \leftarrow Max \in L \;\wedge\; \nexists E \mid E \in L \;\wedge\; E > Max$$

```
max(L,Max) :-
    member(Max,L),
    \+ (member(E,L), E>Max).
```

```
?- max([3,5,2,8,1],Max).
Max = 8
```

From specifications to efficient programs

"Max is the maximum element of a set if there is no element in the set that is larger than it."

$$max(L, Max) \leftarrow Max \in L \;\wedge\; \nexists E \mid E \in L \;\wedge\; E > Max$$

```
max(L,Max) :-
    member(Max,L),
    \+ (member(E,L), E>Max).
```

```
?- max([3,5,2,8,1],Max).
Max = 8
```

```
max2([H|T],Max) :-
    max_(T,H,Max).

max_([],Max,Max).
max_([H|T],TMax,Max) :-
       H > TMax,
       max_(T,H,Max).
max_([H|T],TMax,Max) :-
       H =< TMax,
       max_(T,TMax,Max).
```

From specifications to efficient programs

"Max is the maximum element of a set if there is no element in the set that is larger than it."

$$max(L, Max) \leftarrow Max \in L \ \land \ \nexists E \mid E \in L \ \land \ E > Max$$

```
max(L,Max) :-
    member(Max,L),
    \+ (member(E,L), E>Max).
```

```
?- max([3,5,2,8,1],Max).
Max = 8
```

```
max2([H|T],Max) :-
    max_(T,H,Max).

max_([],Max,Max).
max_([H|T],TMax,Max) :-
        H > TMax,
        max_(T,H,Max).
max_([H|T],TMax,Max) :-
        H =< TMax,
        max_(T,TMax,Max).
```

```
?- max2([3,5,2,8,1],Max).
Max = 8
```

So, what is Prolog?

Procedure	=	Horn clause	+	Top-down reasoning (SL-resolution)
(Algorithm	=	Logic	+	Control)

So:

- Computational procedures can be given a logical form.
- Horn clause reasoning can be performed as efficiently as computation.

The birth of Prolog (Sources: Colmerauer, Kowalski)

Colmerauer
et al 1972.
Prolog!

The birth of Prolog

(Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer
et al 1972.
Prolog!

The birth of Prolog (Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer, 1967
PhD: Precedences, analyse syntaxtique et languages de programmation

↓

Colmerauer 1970
Q–systems

Colmerauer et al 1972.
Prolog!

The birth of Prolog (Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer, 1967
PhD: Precedences, analyse syntaxtique et languages de programmation

Green, 1969
Applicatiom of Theorem Proving to Problem Solving

Colmerauer 1970
Q−systems

Colmerauer et al 1972.
Prolog!

The birth of Prolog

(Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer, 1967
PhD: Precedences, analyse syntaxtique et languages de programmation

Green, 1969
Applicatiom of Theorem Proving to Problem Solving

Colmerauer 1970
Q–systems

Kowalski and Kuener, 1971.
SL–resolution

Colmerauer et al 1972.
Prolog!

The birth of Prolog

(Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer, 1967
PhD: Precedences, analyse syntaxtique et languages de programmation

Green, 1969
Applicatiom of Theorem Proving to Problem Solving

Colmerauer 1970
Q–systems

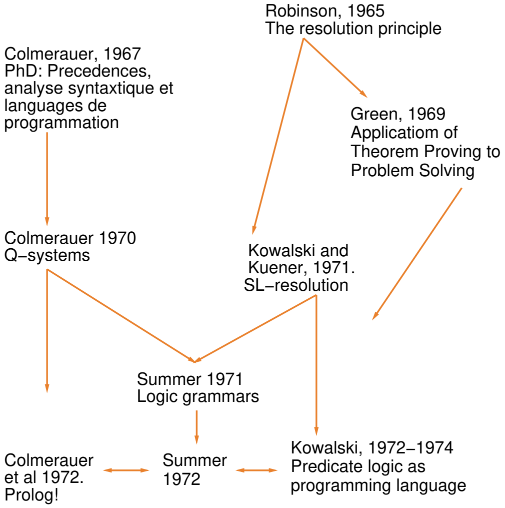
Kowalski and Kuener, 1971.
SL–resolution

Summer 1971
Logic grammars

Colmerauer et al 1972.
Prolog!

Summer 1972

The birth of Prolog (Sources: Colmerauer, Kowalski)



The birth of Prolog (Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer, 1967
PhD: Precedences, analyse syntaxtique et languages de programmation

Green, 1969
Application of Theorem Proving to Problem Solving

Hewitt, 1969
PLANNER

Foster and Elcock
1969, Absys 1

Colmerauer 1970
Q–systems

Kowalski and Kuener, 1971.
SL–resolution

Winograd, 1972
Understanding natural language

Hayes, 1971–1973
Computation and deduction

Summer 1971
Logic grammars

Colmerauer et al 1972.
Prolog!

Summer 1972

Kowalski, 1972–1974
Predicate logic as programming language

The birth of Prolog (Sources: Colmerauer, Kowalski)

Robinson, 1965
The resolution principle

Colmerauer, 1967
PhD: Precedences, analyse syntaxtique et languages de programmation

Green, 1969
Application of Theorem Proving to Problem Solving

Hewitt, 1969
PLANNER

Foster and Elcock
1969, Absys 1

Colmerauer 1970
Q–systems

Kowalski and Kuener, 1971.
SL–resolution

Hayes, 1971–1973
Computation and deduction

Winograd, 1972
Understanding natural language

Summer 1971
Logic grammars

Colmerauer et al 1972.
Prolog!

Summer 1972

Kowalski, 1972–1974
Predicate logic as programming language

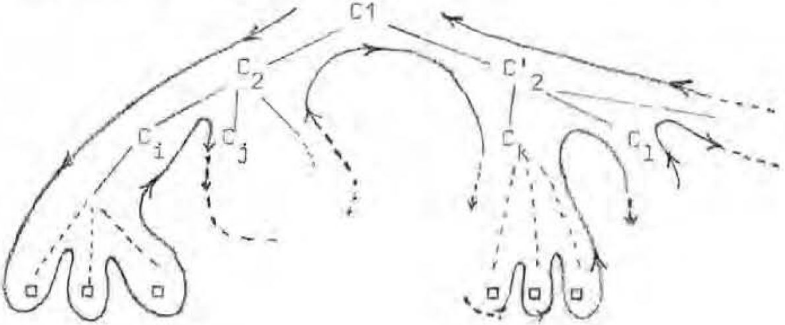
Warren et al 1975
DEC10 Prolog

The original Prolog

```
+Frere(*y,*z)-Pere(*x,*y)-Pere(*x,*z).

+Pere(Paul,Pierre)

+Mere(Marie,Jacques)

+Mari(Paul,Marie)

+Pere(*x* y)-Mari(*x,*z)-Mere(*z,*y)


** CONCATENATION DE LISTES ..

+CONC(*X.NIL,*X) ..
+CONC((*X.*Y).*Z,*X.*U) -CONC(*Y.*Z,*U) ..
+CONC(NIL.*X,*U) -CONC(*X,*U) ..
```

The original Prolog

On peut représenter les descendants d'une clause C_1 de <données> sous forme d'un arbre:



Some milestones (≈ up to ISO)

- First *Prolog*(s): all fundamental characteristics of the language already there!

Some milestones (≈ up to ISO)

- Dec-10 Prolog: *compilation* (+ improved syntax, etc.)

⇝ performance (≈ lisp),

⇝ much more widespread use –but portability an issue.

Some milestones (≈ up to ISO)

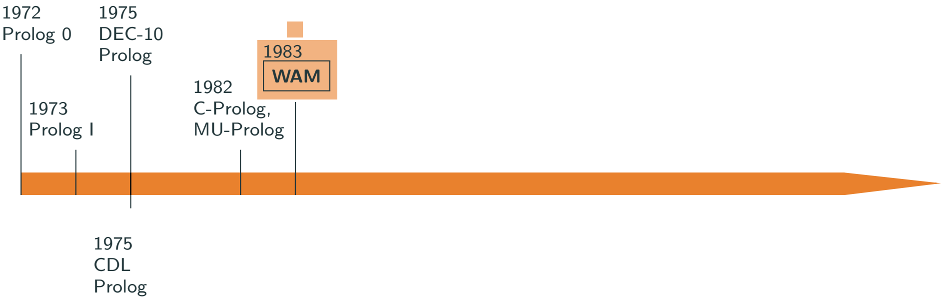
- In parallel, many further advances in the theoretical underpinnings:
  - Kowalski (1974): linear resolution for Horn clauses, no factoring rule.
  - Kowalski and vanEmden (1976): minimal model and fixed-point semantics.
  - Clark (1978): correctness of NaF w.r.t. program completion.
  - Reiter (1978): formalization of "closed world assumption."

  Others: Minker, Gallaire, Cohen, Lassez/Jaffar/Maher, DHD Warren, Sato/Tamaki, DS Warren, ...

Some milestones (≈ up to ISO)

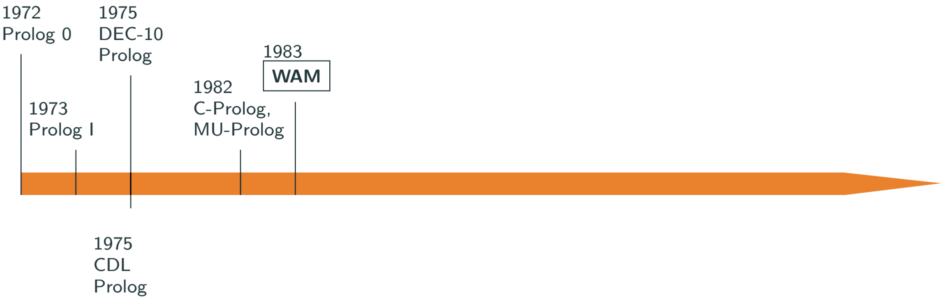
- CDL-Prolog, MU-Prolog, ...,
- C-Prolog: portability (but interpreter).

Some milestones (≈ up to ISO)



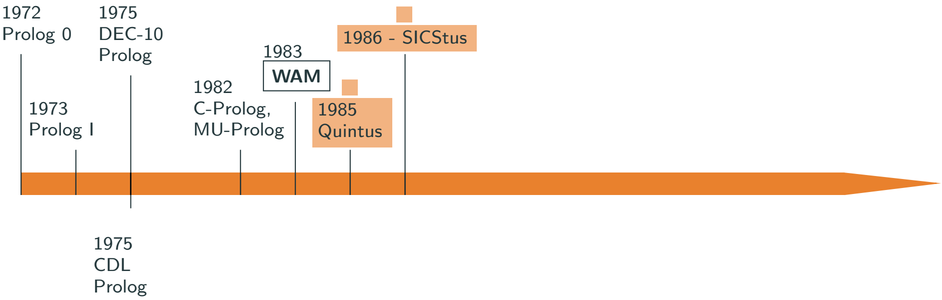
- The *WAM:* portability + speed... and implementation beauty.

Some milestones (≈ up to ISO)



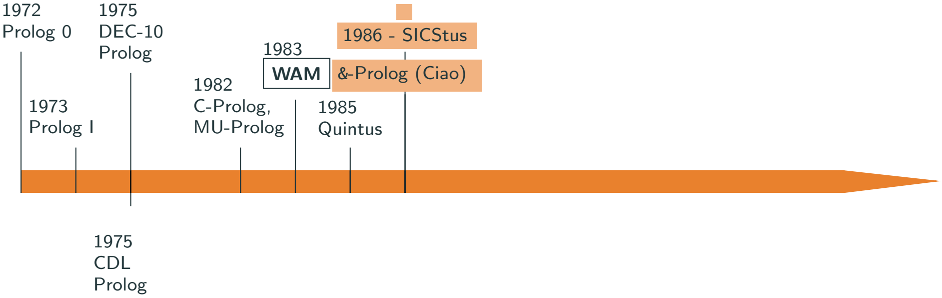
- FGCS ⇝ MCC ⇝ ECRC ⇝ ESPRIT ⇝ EU research programs, and others.

Some milestones (≈ up to ISO)



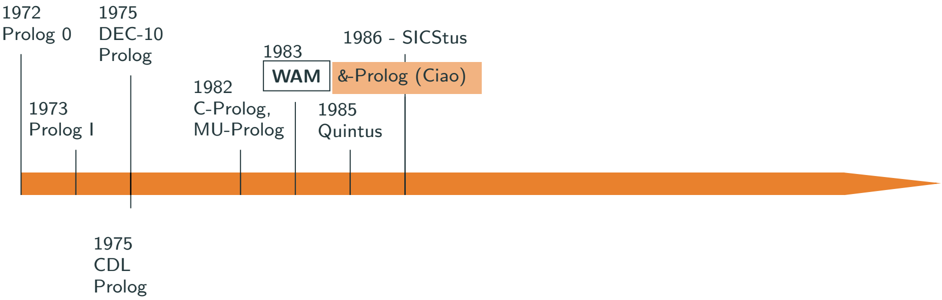
- First WAM-based systems: Quintus, SICStus, BIM, ...
  - Both commercial and public domain ⇝ more dissemination.
  - Many optimizations, GC, ... ⇝ more performance.

Some milestones (≈ up to ISO)



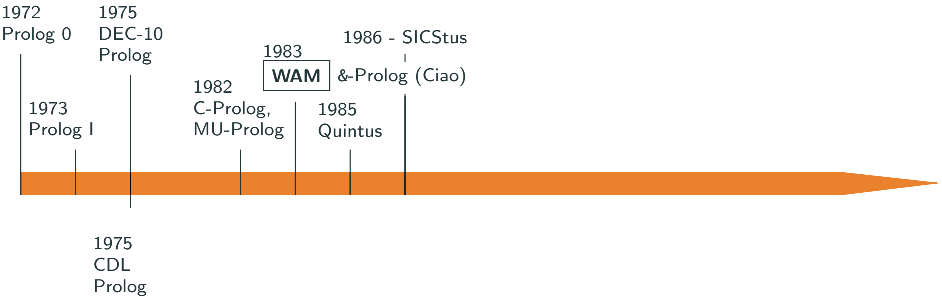
- Or- and and-parallelism: Aurora, &-Prolog/Ciao, MUSE, DASWAM, IDIOM, Andorra, EAM, ...

Some milestones (≈ up to ISO)



- Or- and and-parallelism: Aurora, &-Prolog/Ciao, MUSE, DASWAM, IDIOM, Andorra, EAM, ...
- *Global analysis* (abstract interpretation): Aquarius, &-Prolog/Ciao.
  (Independence,modes, types, determinacy, non-failure, cost, ...)
  First practical compiler(s) using abstract interpretation?

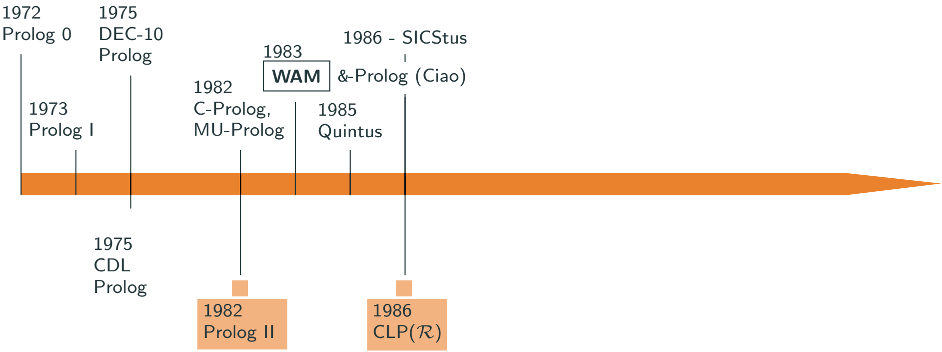
Some milestones (≈ up to ISO)



- Or- and and-parallelism: Aurora, &-Prolog/Ciao, MUSE, DASWAM, IDIOM, Andorra, EAM, ...
- *Global analysis* (abstract interpretation): Aquarius, &-Prolog/Ciao.
  (Independence,modes, types, determinacy, non-failure, cost, ...)
  First practical compiler(s) using abstract interpretation?

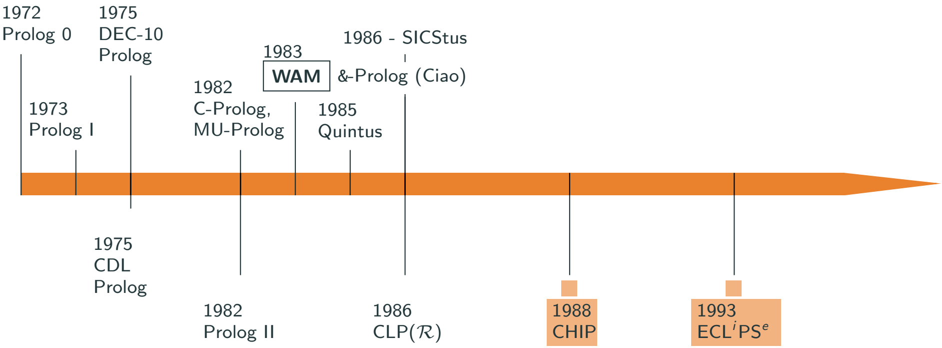
⇝ Performance (≈ imperative), auto-parallelization - real parallel speedups.

Some milestones (≈ up to ISO)



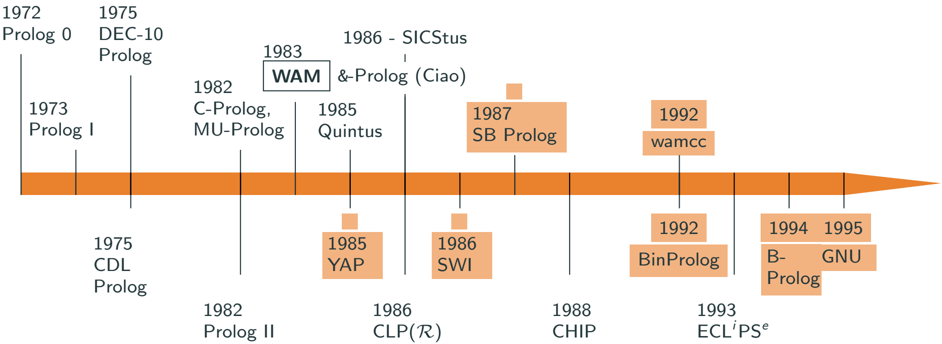
- *Constraints* (Prolog II; CLP scheme and CLP($\mathcal{R}$))
  - Allow many extensions to unification ("domains"), e.g., infinite terms.
  - Recover declarativity for Prolog arithmetic (now also reversible!).

Some milestones (≈ up to ISO)



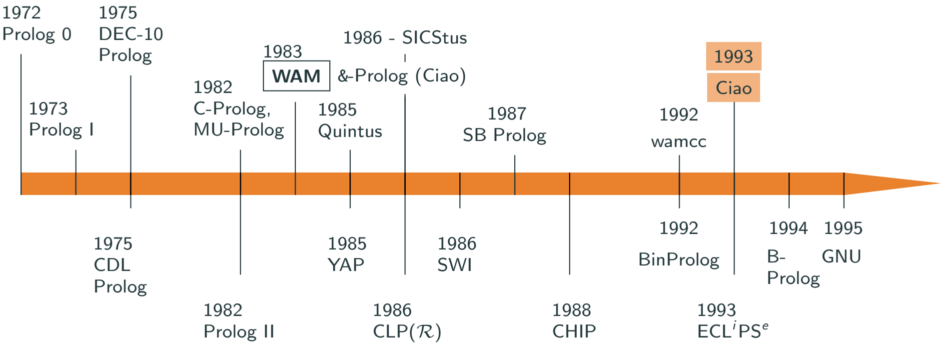
- *Constraints* (Prolog II; CLP scheme and CLP($\mathcal{R}$))
  - Allow many extensions to unification ("domains"), e.g., infinite terms.
  - Recover declarativity for Prolog arithmetic (now also reversible!).
  - Finite domains.

Some milestones (≈ up to ISO)



- *Constraints* (Prolog II; CLP scheme and CLP($\mathcal{R}$))
  - Allow many extensions to unification ("domains"), e.g., infinite terms.
  - Recover declarativity for Prolog arithmetic (now also reversible!).
  - Finite domains.
- A good number of other WAM and non-WAM-based Prologs (see later).
- Constraints in standard Prologs: "Opening the box" (attvars,CHR).

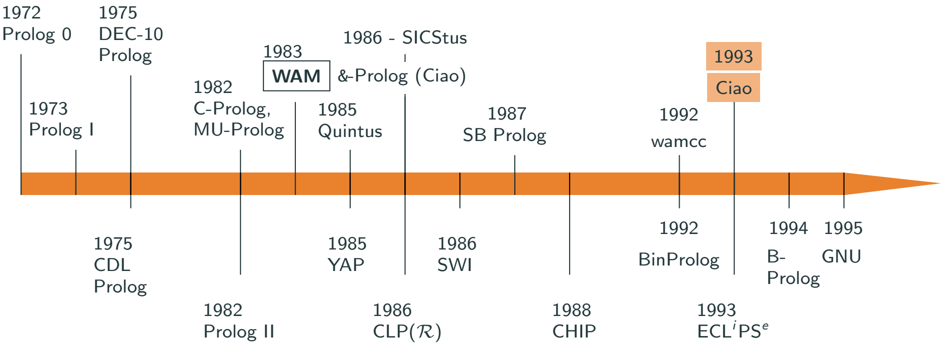
Some milestones (≈ up to ISO)



- A different form of building the language:
  - Pure kernel, all built-ins are in libraries.
  - ⇝ pure subsets of Prolog supported.
  - ⇝ Many extensions: e.g., full higher-order and functional syntax support.

(also λ-Prolog, HiLog, Hiord, ...).

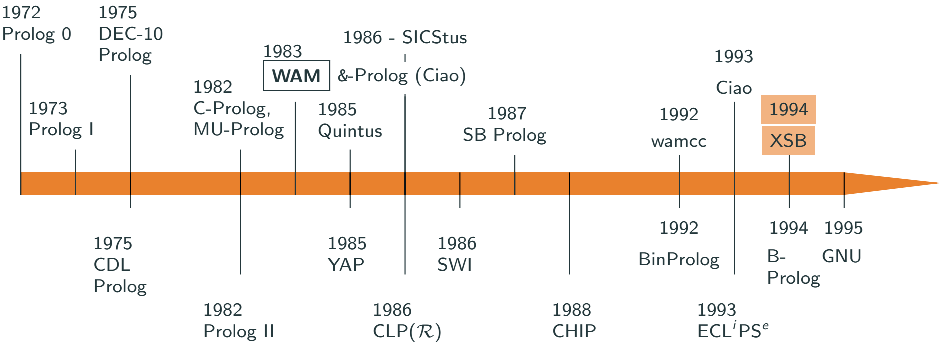
Some milestones (≈ up to ISO)



- A different form of building the language:
  - ▶ Pure kernel, all built-ins are in libraries.
  - ⇝ pure subsets of Prolog supported.
  - ⇝ Many extensions: e.g., full higher-order and functional syntax support.

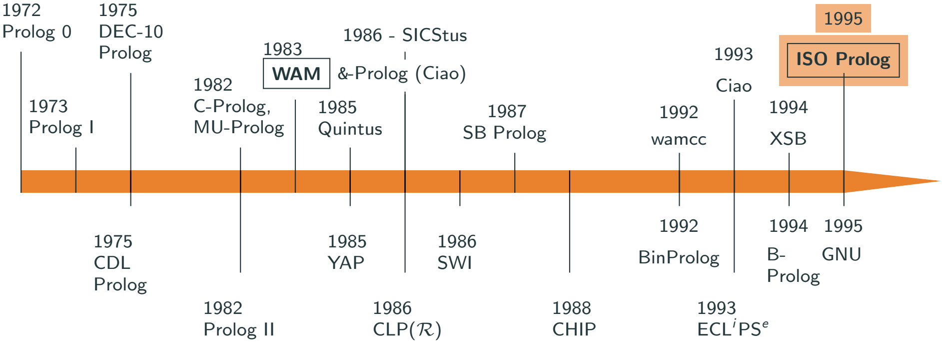
  (also λ-Prolog, HiLog, Hiord, ...).
- Assertions: Types/modes, det, cost ⇝ verification, automatic. testing.

Some milestones (≈ up to ISO)



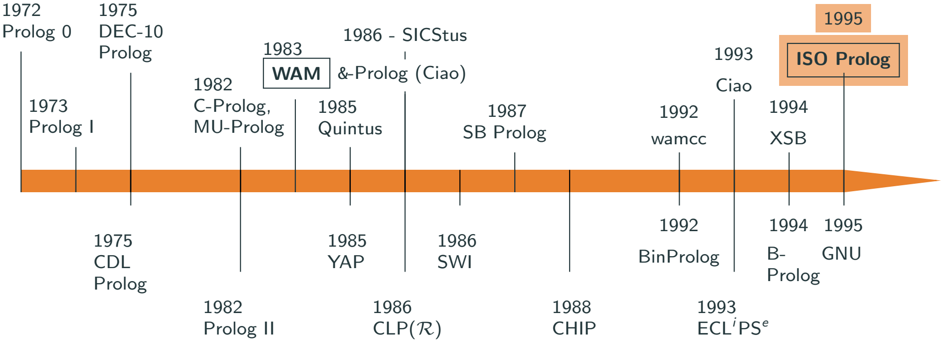
- Tabling (Early deduction, SLG-resolution, ...):
  - Much improved termination (bounded term size).
  - Some nice complexity guarantees.
  - Support for negation with well-founded semantics.

Some milestones (≈ up to ISO)



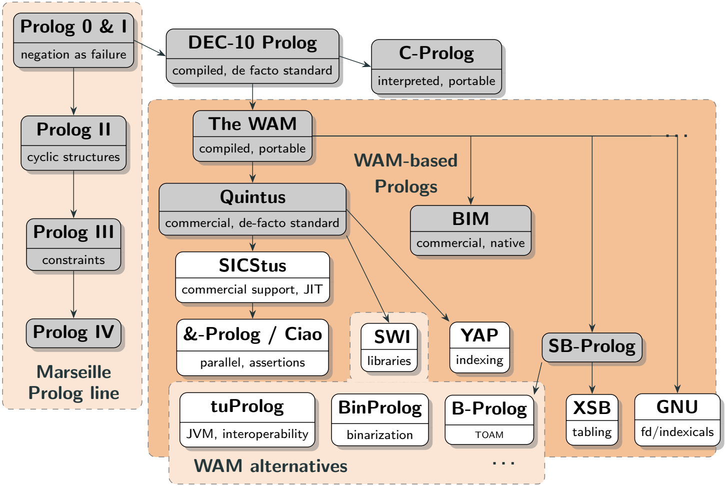
- The ISO standard brought much needed standardization; most systems followed (mostly).

Fast forward...



⇝ Let's jump forward and take a look at the current state of things!

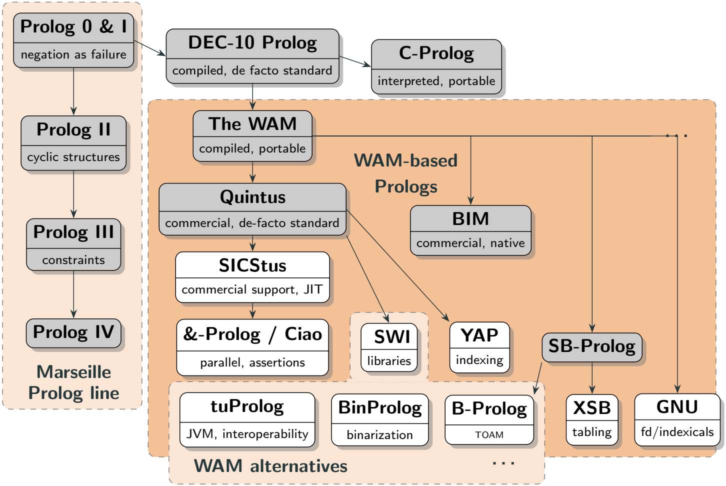
Prolog system heritage



White background: currently active/supported systems.
Lower legends: just some highlight(s) (see later).
Arrows: influences and inspiration.

Again, more missing!: microProlog, LPA, ECL^iPS^e, IBM, LIFE, Andorra-I, Scryer, Tau, ...

Prolog system heritage



White background: currently active/supported systems.
Lower legends: just some highlight(s) (see later).
Arrows: influences and inspiration.

Again, more missing!: microProlog, LPA, ECL^{i}PS^e, IBM, LIFE, Andorra-I, Scryer, Tau, ...

Some features of current systems - I

System	Open Src.	Modules	Non-Std. Data Types	Foreign Language Interfaces
B-Prolog			arrays, sets, hashtables	C, Java
Ciao	✓	✓		C, Java, Python, JScrpt
ECLiPSe	✓	✓	arrays, strings	C, Java, Python, PHP
GNU Prolog	✓		arrays	C, Java, PHP
JIProlog	✓	✓		Java
SICStus		✓		C, Java, .NET, Tcl/Tk
SWI	✓	✓	dicts, strings	C, C++, Java
τProlog	✓	✓		JavaScript
tuProlog	✓		arrays	Java, .NET, Android, iOS
XSB	✓	✓		C, Java, PERL, Python
YAP	✓	✓		C, Python, R

Some features of current systems - I

System	Open Src.	Modules	Non-Std. Data Types	Foreign Language Interfaces
B-Prolog			arrays, sets, hashtables	C, Java
Ciao	✓	✓		C, Java, Python, JScrpt
ECLiPSe	✓	✓	arrays, strings	C, Java, Python, PHP
GNU Prolog	✓		arrays	C, Java, PHP
JIProlog	✓	✓		Java
SICStus		✓		C, Java, .NET, Tcl/Tk
SWI	✓	✓	dicts, strings	C, C++, Java
τProlog	✓	✓		JavaScript
tuProlog	✓		arrays	Java, .NET, Android, iOS
XSB	✓	✓		C, Java, PERL, Python
YAP	✓	✓		C, Python, R

Some features of current systems - I

System	Open Src.	Modules	Non-Std. Data Types	Foreign Language Interfaces
B-Prolog			arrays, sets, hashtables	C, Java
Ciao	✓	✓		C, Java, Python, JScrpt
ECLiPSe	✓	✓	arrays, strings	C, Java, Python, PHP
GNU Prolog	✓		arrays	C, Java, PHP
JIProlog	✓	✓		Java
SICStus		✓		C, Java, .NET, Tcl/Tk
SWI	✓	✓	dicts, strings	C, C++, Java
τProlog	✓	✓		JavaScript
tuProlog	✓		arrays	Java, .NET, Android, iOS
XSB	✓	✓		C, Java, PERL, Python
YAP	✓	✓		C, Python, R

Some features of current systems - I

System	Open Src.	Modules	Non-Std. Data Types	Foreign Language Interfaces
B-Prolog			arrays, sets, hashtables	C, Java
Ciao	✓	✓		C, Java, Python, JScrpt
ECLiPSe	✓	✓	arrays, strings	C, Java, Python, PHP
GNU Prolog	✓		arrays	C, Java, PHP
JIProlog	✓	✓		Java
SICStus		✓		C, Java, .NET, Tcl/Tk
SWI	✓	✓	dicts, strings	C, C++, Java
τProlog	✓	✓		JavaScript
tuProlog	✓		arrays	Java, .NET, Android, iOS
XSB	✓	✓		C, Java, PERL, Python
YAP	✓	✓		C, Python, R

Some features of current systems - I

System	Open Src.	Modules	Non-Std. Data Types	Foreign Language Interfaces
B-Prolog			arrays, sets, hashtables	C, Java
Ciao	✓	✓		C, Java, Python, JScrpt
ECLiPSe	✓	✓	arrays, strings	C, Java, Python, PHP
GNU Prolog	✓		arrays	C, Java, PHP
JIProlog	✓	✓		Java
SICStus		✓		C, Java, .NET, Tcl/Tk
SWI	✓	✓	dicts, strings	C, C++, Java
τProlog	✓	✓		JavaScript
tuProlog	✓		arrays	Java, .NET, Android, iOS
XSB	✓	✓		C, Java, PERL, Python
YAP	✓	✓		C, Python, R

Some features of current systems - II

System	CLP	CHR	Tabling	Parallelism	Indexing	Coroutines
B-Prolog	$\mathcal{FD}$, $\mathcal{B}$, *Set*	✓	✓		N-FA	✓
Ciao	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	FA, MA	✓
ECLiPSe	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$, *Set*	✓		✓	most suitable	✓
GNU Prolog	$\mathcal{FD}$, $\mathcal{B}$				FA	
JIProlog					undocumented	
SICStus	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓			FA	✓
SWI	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	MA, deep, JIT	✓
τProlog					undocumented	
tuProlog				✓	FA	
XSB	$\mathcal{R}$	✓	✓	✓	all, trie	✓
YAP	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓		FA, MA, JIT	

Some features of current systems - II

System	CLP	CHR	Tabling	Parallelism	Indexing	Coroutines
B-Prolog	$\mathcal{FD}$, $\mathcal{B}$, *Set*	✓	✓		N-FA	✓
Ciao	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	FA, MA	✓
ECLiPSe	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$, *Set*	✓		✓	most suitable	✓
GNU Prolog	$\mathcal{FD}$, $\mathcal{B}$				FA	
JIProlog					undocumented	
SICStus	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓			FA	✓
SWI	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	MA, deep, JIT	✓
τProlog					undocumented	
tuProlog				✓	FA	
XSB	$\mathcal{R}$	✓	✓	✓	all, trie	✓
YAP	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓		FA, MA, JIT	

Some features of current systems - II

System	CLP	CHR	Tabling	Parallelism	Indexing	Coroutines
B-Prolog	$\mathcal{FD}$, $\mathcal{B}$, *Set*	✓	✓		N-FA	✓
Ciao	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	FA, MA	✓
ECLiPSe	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$, *Set*	✓		✓	most suitable	✓
GNU Prolog	$\mathcal{FD}$, $\mathcal{B}$				FA	
JIProlog					undocumented	
SICStus	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓			FA	✓
SWI	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	MA, deep, JIT	✓
τProlog					undocumented	
tuProlog				✓	FA	
XSB	$\mathcal{R}$	✓	✓	✓	all, trie	✓
YAP	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓		FA, MA, JIT	

Some features of current systems - II

System	CLP	CHR	Tabling	Parallelism	Indexing	Coroutines
B-Prolog	$\mathcal{FD}$, $\mathcal{B}$, *Set*	✓	✓		N-FA	✓
Ciao	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	FA, MA	✓
ECLiPSe	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$, *Set*	✓		✓	most suitable	✓
GNU Prolog	$\mathcal{FD}$, $\mathcal{B}$				FA	
JIProlog					undocumented	
SICStus	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓			FA	✓
SWI	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	MA, deep, JIT	✓
τProlog					undocumented	
tuProlog				✓	FA	
XSB	$\mathcal{R}$	✓	✓	✓	all, trie	✓
YAP	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓		FA, MA, JIT	

Some features of current systems - II

System	CLP	CHR	Tabling	Parallelism	Indexing	Coroutines
B-Prolog	$\mathcal{FD}$, $\mathcal{B}$, *Set*	✓	✓		N-FA	✓
Ciao	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	FA, MA	✓
ECLiPSe	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$, *Set*	✓		✓	most suitable	✓
GNU Prolog	$\mathcal{FD}$, $\mathcal{B}$				FA	
JIProlog					undocumented	
SICStus	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓			FA	✓
SWI	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	MA, deep, JIT	✓
τProlog					undocumented	
tuProlog				✓	FA	
XSB	$\mathcal{R}$	✓	✓	✓	all, trie	✓
YAP	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓		FA, MA, JIT	

Some features of current systems - II

System	CLP	CHR	Tabling	Parallelism	Indexing	Coroutines
B-Prolog	$\mathcal{FD}$, $\mathcal{B}$, *Set*	✓	✓		N-FA	✓
Ciao	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	FA, MA	✓
ECLiPSe	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$, *Set*	✓		✓	most suitable	✓
GNU Prolog	$\mathcal{FD}$, $\mathcal{B}$				FA	
JIProlog					undocumented	
SICStus	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓			FA	✓
SWI	$\mathcal{FD}$, $\mathcal{B}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓	✓	MA, deep, JIT	✓
τProlog					undocumented	
tuProlog				✓	FA	
XSB	$\mathcal{R}$	✓	✓	✓	all, trie	✓
YAP	$\mathcal{FD}$, $\mathcal{Q}$, $\mathcal{R}$	✓	✓		FA, MA, JIT	

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

- Many other features and extensions:
  - Other types of negation, other combinations with ASP.
  - Attributed variables, enhanced expansions.
  - Functional syntax, lazy execution, higher-order, objects, ...
  - Learning (ILP), probabilistic rules, combination with deep learning.

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

- Many other features and extensions:
  - Other types of negation, other combinations with ASP.
  - Attributed variables, enhanced expansions.
  - Functional syntax, lazy execution, higher-order, objects, ...
  - Learning (ILP), probabilistic rules, combination with deep learning.
  - Auto-documentation, (integration with) program development environments.
  - Playgrounds, in-browser execution, notebooks, embeddable engines, ...

Some features of current systems - III

System	Debugger	Global Vars.	Mutables	Testing	Types/Modes	s(CASP)
B-Prolog	trace	✓				
Ciao	trace / source	✓	✓	✓	✓	✓
ECLiPSe	trace	✓		✓		
GNU Prolog	trace	✓	✓			
JIProlog	trace					
SICStus	trace / source		✓	✓		
SWI	trace / graphical	✓	✓	✓		✓
τProlog						
tuProlog	spy					
XSB	trace		✓			
YAP	trace	✓				

- Many other features and extensions:
  - Other types of negation, other combinations with ASP.
  - Attributed variables, enhanced expansions.
  - Functional syntax, lazy execution, higher-order, objects, ...
  - Learning (ILP), probabilistic rules, combination with deep learning.
  - Auto-documentation, (integration with) program development environments.
  - Playgrounds, in-browser execution, notebooks, embeddable engines, ...
  - Applications of Prolog technology to other languages (analyzers, provers, ...).

Summary of current state

- Prolog systems have come a very long way!
  - As seen, a good number of features available on several systems:
    - Indexing, constraints/CHR, multi-threading, tabling, foreign interfaces, coroutining, global vars, mutables, testing, ...
- An issue is portability:
  - ISO standard generally supported (with only minor differences).
  - *Basic* module system pretty compatible.

  However,
  - Interfaces and details of extensions often differ.
    Can mostly be bridged (c.f., Paolo Moura's work), but a real nuisance.
  - Some useful features still present in only a few systems:
    e.g., types/modes/verification, functional syntax, s(CASP), ...

→ Work needed to improve portability.

- Also, better community infrastructure would be useful (see at the end).

Summary of current state

- Prolog systems have come a very long way!
  - As seen, a good number of features available on several systems:
    - Indexing, constraints/CHR, multi-threading, tabling, foreign interfaces, coroutining, global vars, mutables, testing, ...
- An issue is portability:
  - ISO standard generally supported (with only minor differences).
  - *Basic* module system pretty compatible.

  However,
  - Interfaces and details of extensions often differ.
    Can mostly be bridged (c.f., Paolo Moura's work), but a real nuisance.
  - Some useful features still present in only a few systems:
    e.g., types/modes/verification, functional syntax, s(CASP), ...

$\rightarrow$ Work needed to improve portability.

- Also, better community infrastructure would be useful (see at the end).

Prolog influences

- In other languages within LP and its extensions:
  - Goedel, Mercury, Turbo-Prolog (static typing)
  - λ-Prolog, Curry, Babel, HiLog (FP/HO)
  - CP, GHC, Parlog, Erlang (committed choice)
  - Datalog, ASP – Co-inductive LP, s(ASP) and s(CASP) (Prolog extensions)
  - HyProlog (assumptions and abduction), Flora-2/ErgoAI, ...
  - Probabilistic LP, ProbLog, ...
  - ProGol, ILP (learning)
  - LogTalk (objects), Picat (imperative syntax)
  - CHR, CHRG, ...
- Beyond LP:
  - Theorem proving technology
  - Erlang
  - Java (abstract machine, specification, ...)
  - Many embeddings in other languages
  - Many others: C++, many compilers, ...
  - Many analyzers and verifiers for other languages
  - ...

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with “declarative” pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Some Prolog strong points

- Powerful programming paradigm, includes most others (e.g. functions are relations).
- Allows going smoothly from executable specifications to efficient implementation.
- Clean, simple syntax and semantics. Easy meta-programming.
- Immutable persistent data structures, with "declarative" pointers (logic variables).
- Safety: garbage collection, no NullPointer exceptions, ...
- Efficiency: very efficient inference, pattern matching, and unification; tail-recursion and last-call optimization; indexing, efficient tabling.
- Many features (as we saw, but also DCGs, arbitrary precision arithmetic, ...).
- Fast development: interactive top-level, debugging, ...
- Sophisticated tools: analyzers, verifiers, partial evaluators, parallelizers, ...
- Community:
  - Both commercial and open-source systems (some very substantive and mature!).
  - Active developer community with constant new implementations, features, etc.
  - Many books, courses, and learning materials.
- Successful applications, including:
  - Analyzers (Abstr. Interp., Set-Based Anal., Datalog, energy, gas, ...), compilers, ...
  - Compilers, interpreters, domain-specific languages, ...
  - Heterogeneous data integration.
  - Computational law.
  - Configuration, scheduling, ...
  - Natural language processing.
  - Efficient inference (expert systems, theorem provers), symbolic AI in general, ...

  See the applications sessions today!

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!
- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!
- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!
- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!
- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!
- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Prolog perceived weaknesses → and how to address them

- Learning curve, beginners can easily write programs that loop or consume a huge amount of resources → teach it well, use the right tools (see later)
- Lack of (static) typing / data hiding / object orientation. → but notable exceptions!
- Limited support for embedded or app development → but notable exceptions!

- Syntactically different from "traditional" programming languages, not a mainstream language → offer alternative syntax?
- IDEs and development tools: much progress but still limitations in some areas (e.g., refactoring) → future work?
- Other limitations in portability across systems (e.g., packages) → need to improve.
- UI development (usually conducted in a foreign language via FLI) → exceptions / need to improve? / actually OK?

Summary: much can be taken from other Prolog systems; also work still needed.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Possible threats to Prolog's future → and how to address them

- Comparatively small user base.
- Somewhat fragmented community.
- Active developer community with constant new implementations, features. (Good but possible further fragmentation of Prolog implementations.)
- New programming languages.
- The perception that it is an "old" language.
- Wrong image due to "shallow" teaching of the language.

→

- Many weaknesses already addressed by different systems. → cooperative/competitive evolution (vs. unified system and/or libraries).
- In any case, good forum needed for discussion and bringing together community across systems.

Opportunities for Prolog

- New application areas, addressing societal challenges:
  - Neuro-Symbolic AI.
  - Explainable AI, verifiable AI.
  - Big Data.
- New features and developments:
  - Probabilistic reasoning.
  - Embedding ASP and SAT or SMT solving, s(CASP) applications.
  - Opportunity still for performance gains (and we have the technology):
    - Full-fledged JIT compiler.
    - Global optimization, partial evaluation ('provably correct refactoring').
    - Parallelism.
  - ...

Opportunities for Prolog

- New application areas, addressing societal challenges:
  - Neuro-Symbolic AI.
  - Explainable AI, verifiable AI.
  - Big Data.
- New features and developments:
  - Probabilistic reasoning.
  - Embedding ASP and SAT or SMT solving, s(CASP) applications.
  - Opportunity still for performance gains (and we have the technology):
    - Full-fledged JIT compiler.
    - Global optimization, partial evaluation ('provably correct refactoring').
    - Parallelism.
  - ...

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures $\rightsquigarrow$ Horn clauses & SLD-resolution (effectiveness/semi-decidability) $\rightsquigarrow$ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

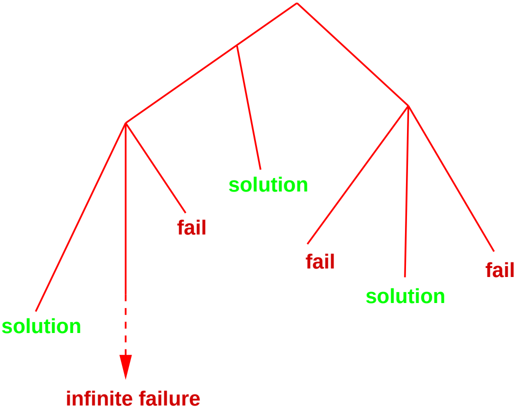
- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures $\rightsquigarrow$ Horn clauses & SLD-resolution (effectiveness/semi-decidability) $\rightsquigarrow$ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

On teaching Prolog (specially for CS students and programmers)

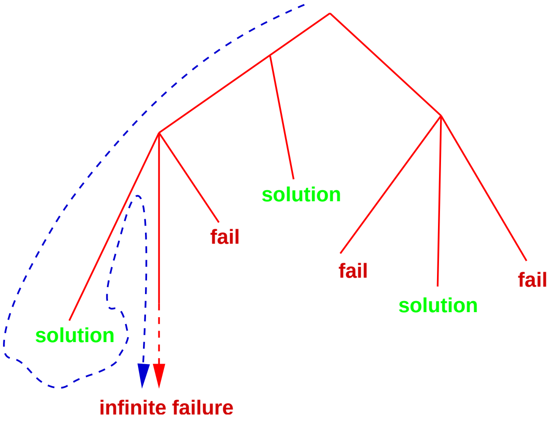
- Should Prolog/LP/CLP be taught in CS programs?
  Yes!: CS grads simply not complete without. But... has to be done right.
  - Typical 'programming paradigms' course can be counter-productive.
  - But what to do if, as frequently, that is the only slot available?

- Modern Prologs address well most of the shortcomings of early systems via tabling, constraints, multiple search rules, etc. — take advantage of this!
- Discuss the "dream" (Green), logics and deduction procedures ⇝ Horn clauses & SLD-resolution (effectiveness/semi-decidability) ⇝ classical LP (Kowalski/Colmerauer).
- Important to show the beauty and multiple facets of the language:
  - Show with examples how you can go from problem representations and executable specifications to efficient algorithms, gradually, as needed.
- Students will find non-termination early on: help them understand it.
  - A fact of life in programming languages (halting problem) or proof systems.
  - Start running programs breadth-first (or iterative deepening, tabling, etc.): all solutions in finite time (explain with picture of tree); discus the trade-offs.
- Arithmetic: Peano (beautiful/slow), constraints, is/2. Discuss trade-offs.
- Occur check: `unify_with_occurs_check/2`, cyclic terms. Discuss trade-offs.

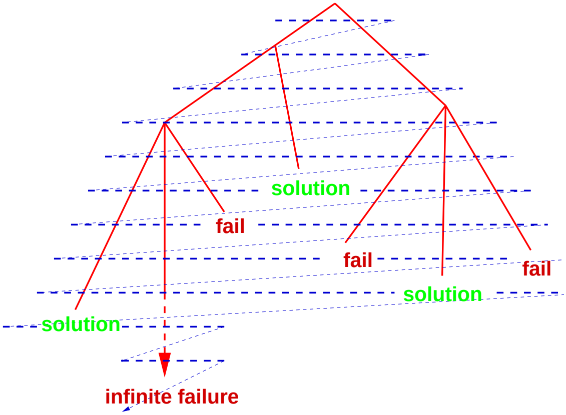
Characterization of the search tree



Depth-First Search



Breadth-First Search



On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

On teaching Prolog (specially for CS students and programmers)

Specially relevant to teaching students that have already been exposed to other programming languages (imperative/OO, sometimes functional) and have some notions of PL implementation:

- Discuss Prolog as a traditional programming language but with "much more"
  - "Normal" if used in one mode and there is only one definition per procedure.
  - But it can also have several definitions, search, run "backwards," etc.
  - As any language, Prolog has a stack of forward continuations, to know where to return when a procedure ends (succeeds)... but also a stack of *backwards continuations* to go if there is a failure (previous *choice point*).
- Use predicates to define types and properties; to do dynamic checking or "run backwards" to generate the "inhabitants"; property-based testing for free!
- Show examples of great applications (e.g., from 2022 census). TIOBE index?

- Types of system for teaching:
  - Advanced students: classical installation.
    Show serious, competitive language; ready for "real" use.
  - Beginners/tutorials: playgrounds, notebooks.
    (E.g., Ciao Playgrounds/Active Documents, SWISH, τ-Prolog).
- An ideal system should allow covering: pure LP (w/several search rules, tabling), ISO-Prolog, constraints, higher-order (and functional prog.), ASP/s(CASP), etc.

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!

Final thoughts

- The classical characteristics of Prolog are still unique and demanded.
- It is still one of the most interesting computing paradigms.
- Plus, it is also not 'your grandparents's Prolog' any more.
- Many (most?) of the initial shortcomings of the language have been addressed, even if sometimes by different systems.
- More relevant than ever at a time in need for explainable AI.
- Needs to be taught, and to be taught well. (ACM curriculum!)

- Regarding system coordination: despite the intense evolution, differences between systems are not fundamental. To progress:
  - Forum (e.g., a web platform) to discuss proposals and solutions, in order to reach consensus on the most important extensions of current implementations.
  - A structured workflow for tracking proposals.
  - Taking advantage of / build on existing mechanisms such as the ISO standard or (an updated version of) the Prolog Commons.
  - Involving implementors and users.
  - Under the wings of ALP.

⇝ See the "Online Prolog Community" presentation at the end of the day!