

## Introduction (1 of 2)

- Prolog was designed by Alain Colmerauer, from Aix-Marseille University, in 1972.
- Prolog is one of the first, and the most popular, logic PLs. It is a declarative PL. It is based on first-order predicate logic: propositional logic (i.e.,  $\wedge$ ,  $\vee$ , and  $\neg$ ) plus quantifiers (i.e.,  $\forall$  and  $\exists$ ).
- SQL, Make, and many query languages have logic and/or declarative aspects.
- A Prolog program is a set of relations (aka, clauses): *facts* and *rules*, which declare what is true. This is analogous to a set of function definitions in a functional PL.
- A computation is the evaluation of a query, typically containing variables. The translator tries to find a set of variable values that make the query “true” according to the clauses.

## Introduction (2 of 2)

- In general, a Prolog program must be incomplete. It cannot declare everything that is true. This can lead to false failure.
- Most Prolog programs are not “purely” declarative, due to practical realities.
- Prolog syntax is relatively strange. A rule seems backward. Clauses form a big disjunction. Some commas in a rule form a conjunction. A semicolon in a rule forms a disjunction.
- Prolog is strongly and dynamically typed.
- Prolog is statically scoped, but rules cannot be nested.

## Program structure (1 of 3)

- Recall that a program is a set of facts and rules.
- Here are some facts about the free time of a set of people:  
`pub/etc/data.pl`
- Aside from punctuation, this contains *atoms* and *numbers*. They have no inherent meaning, but my intent might be clear to you.
- We can add some rules and evaluate a query:  
`pub/etc/query1.pl`
- A rule has a LHS and a RHS. A fact is the same as a rule with a RHS of `true`. A rule with no LHS is a sort of directive to the compiler.
- The `findall` predicate repeatedly finds values for the variable `Person` making `person` true. The resulting value of the variable `People` is a list of those values.

## Program structure (2 of 3)

- A rule can have variables. They are like formal parameters or local variables. They start with a uppercase letter. An underscore denotes an anonymous singleton variable.
- Semantics: If a set of variable values can be found that makes the RHS true, then the LHS is true with those values.
- Some built-in rules have side effects.
- Here's another example:  
`pub/etc/query2.pl`
- Notice that our rules are starting to define time semantics.

## Program structure (3 of 3)

- Returning to our first query, suppose we don't want duplicate results:

`pub/etc/query3.pl`

- The semantics of a list without duplicates is defined in:

`pub/etc/uniq.pl`

- Such a list must contain the original members, but only one of each.
- I could have used the built-in `member` predicate and the `\+` operator. `\+` is the *negation as failure* operator. You can think of it as logical negation, but that's not exactly right. Just because you can't prove something doesn't mean it's false.
- Finally, here's a sum example:  
`pub/sum/prolog/sum.pl`
- Yes, arithmetic is a bit weird.