# Introduction (1 of 2)

- Prolog was designed by Alain Colmerauer, from Aix-Marseille University, in 1972.
- Prolog is an early, and the most popular, logic PL. It is declarative, and based on first-order predicate logic: propositional logic (i.e., $\wedge$, $\vee$, and $\neg$) plus quantifiers (i.e., $\forall$ and $\exists$).
- Mercury is a similar, but more modern PL.
- 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 like 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)

- Michael Stonebraker, 2005 von Neumann Medal winner, 2014 Turing Award winner, and MIT researcher says: "Having looked at programs in Prolog and R1, I was very leery of the approach. Looking at any rule program with more than 10 statements, it is very difficult to figure out what it does."
- Prolog syntax is relatively strange. A rule seems backward. Clauses form a big disjunction (OR). Some commas in a rule form a conjunction (AND). A semicolon in a rule also forms a disjunction.
- In general, a Prolog program must be incomplete: not declaring all that is true. A result of "false" might be incorrect.
- Most Prolog programs are not purely declarative, due to practical realities.
- Prolog is strongly and dynamically typed. It has static scope, but no rule nesting.

## Program structure (1 of 3)

- 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`
- This contains punctuation, *atoms*, and *numbers*. An atom has no inherent meaning, but my intent should be clear.
- We can add rules and evaluate a query:

  `pub/etc/query1.pl`
- A rule has a LHS and RHS. A fact is the same as a rule with a RHS of `true`. A rule with no LHS is a translator directive.
- The predicate `findall` repeatedly chooses values for the variable `Person`, which make the predicate `person` true. The variable `People` accumulates those values, as a list.

## Program structure (2 of 3)

- A rule can have variables. They are like formal parameters or local variables. They *must* start with a uppercase letter. An underscore denotes an anonymous singleton variable.
- Semantics: If variable values can be chosen, which make the RHS true, then the LHS is true with those same values.
- Some built-in rules have side effects.
- Here's another example:

  `pub/etc/query2.pl`
- Our rules are beginning to define rudimentary time semantics.

## Program structure (3 of 3)

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

  `pub/etc/query3.pl`
- 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 Prolog's `member` predicate and \+ operator (logical negation), but that wouldn't be quite right. Failure to prove something true does not make it false.
- Finally, here's a sum example:

  `pub/sum/prolog/sum.pl`

  Yes, Prolog arithmetic is a bit weird.