# Prolog: Interaction

CS 331 Programming Languages
Lecture Slides
Friday, April 18, 2025

Glenn G. Chappell
Department of Computer Science
University of Alaska Fairbanks
ggchappell@alaska.edu

## Unit Overview
## The Prolog Programming Language

Topics

- ✓ PL feature: execution model
- ✓ PL category: logic PLs
- ✓ Introduction to Prolog
- ✓ Prolog: simple programming
- ✓ Prolog: lists
- ✓ Prolog: flow of control
- Prolog: interaction

## Review

## Review
## Prolog: Flow of Control [1/3]

How do we do repetition in Prolog?

- Using an encapsulated list operation.
  - We wrote `map`, `filter`, and `zip` in an earlier topic.
- Repeating an operation directly, using recursion.
  - We wrote `print_squares`/2.
- Using a traditional loop construction—which we can write ourselves.
  - We wrote a for-loop predicate `myFor`/3 and used it in `print_squares2`/2. (The functionality of `myFor`/3 is already available in SWI-Prolog in the form of `between`/3.)

*For code, see* `flow.pl`*.*

## Review
## Prolog: Flow of Control [2/3]

Prolog includes one non-logical “cheat”: the **cut** (`!`).

- It takes no arguments.
- It always succeeds.
- Once it succeeds, backtracking past the *cut* is not allowed, for the current goal.

Cut can be used in a number of ways:

- To write the equivalent of a C++ `break`.
  - We wrote `print_near_sqrt`/1.
- To do selection (like if/else).
  - We wrote `test_big`/1.
- To ensure that only one fact/rule is used for any particular goal.
  - We wrote `gcd2`/3.
- To write negation.
  - We wrote `not`/1 (like the standard `\+`/1).

*For code, see* `flow.pl`*.*

## Review
## Prolog: Flow of Control [3/3]

We can repeat forever by succeeding, and then recursing.

- We wrote `myRepeat`/0 (which is already available in SWI-Prolog in the form of `repeat`/0).

But this is useless unless we have functionality available that is **nondeterministic**: it can give different results for the same input.

One way to get nondeterminism involves reading console input from the user. *Next we take a brief look at this.*

*For code, see* `flow.pl`*.*

## Prolog: Interaction

## Prolog: Interaction
Preliminaries

Some more useful things:

*For code from this topic, see* `interact.pl`*.*

`read`/1

Always succeeds. As a side effect, reads a Prolog term—*which must be followed by a period (`.`)*—from standard input. Unifies this with its argument.

This is quick-and-dirty input. As a full-featured PL, SWI-Prolog can, of course, do ordinary input of a line, with conversion to (say) a number. But that style of input is too complicated to be worth the time for us.

`flush`/0

Always succeeds. As a side effect, ensures that previous writes have completed. Do this before a `read`, if there are prior `write` calls.

## Prolog: Interaction Example [1/4]

Now we have all the pieces necessary to write simple interaction in Prolog.

TO DO

- Write a predicate `squares_interact`/0 that does interaction: input a number and print its square; then do it again, continuing until zero is entered. Use `repeat` to do a while-true-break style of loop.

*Done. See `interact.pl`.*

## Prolog: Interaction
## Example [2/4]

Suppose we wish to do a break in the *middle* of our loop, so that, after entering zero, we exit without printing its square.

We can do this using a helper predicate for the rest of the loop.

```
rest_of_loop(X) :- X = 0, !.
rest_of_loop(X) :- write( … ), …, fail.

squares_interactX :-
    repeat,
        …
        rest_of_loop(X), …
```

Because `rest_of_loop` is a separate predicate, we can give it multiple rules, allowing for more complex behavior.

TO DO

- Rewrite `squares_interact` as `squares_interact2`, which can break in the middle of its loop.

*Done. See* `interact.pl`*.*

## Prolog: Interaction Example [3/4]

We can get the effect of multiple rules with "`;`", which does OR, where "`,`" means AND. "`;`" has lower precedence than "`,`".

```
rest_of_loop(X) :- X = 0, !.
rest_of_loop(X) :- write( … ), …, fail.
```

We can rewrite the above as follows.

```
rest_of_loop(X) :-
    X = 0, !;
    write( … ), …, fail.
```

OR—this line and the next act the same as separate rules.

So maybe having a separate helper predicate is unnecessary?

## Prolog: Interaction Example [4/4]

Now we can move the body of the rule for our helper predicate into the rule for our main predicate, *placing it in parentheses*.

```
rest_of_loop(X) :-
    X = 0, !;
    write( … ), …, fail.


squares_interact2 :-
    repeat,
        …
        (      ), …
```

Because we can use parentheses, the lower precedence of the OR does not mess things up. We can say *A* AND *B* AND (*C* OR *D*) AND *E*.

TO DO

- Rewrite `squares_interact2` as `squares_interact3`, which does the same thing, but has no helper predicate.

*Done. See* `interact.pl`.

# Unit Overview
# The Prolog Programming Language

Topics

- ✓ PL feature: execution model
- ✓ PL category: logic PLs
- ✓ Introduction to Prolog
- ✓ Prolog: simple programming
- ✓ Prolog: lists
- ✓ Prolog: flow of control
- ✓ Prol[illegible] interaction

DONE