Modes and their meaning

Manuel Hermenegildo, Jose F. Morales, Joachim Schimpf,
Theresa Swift, David Warren, Jan Wielemaker, . . .

September 12, 2025 @ ICLP/LOPSTR/PPDP, Rende, Italy

What do we mean by modes?

- Modes of a predicate: ways in which it can be called.
  E.g., `length(+,-)` and `length(-,+)`
- We concentrate on these special terms:
  `+`, `-`, `?`, `@`, etc. or `+list`, `-list`, ?list, `@list`, etc.
  which we call argument modes (or ModeSpecs)
- Predicate modes can be expressed in different ways:
  - Classical mode declarations (from Dec10/Quintus Prolog):
    :- mode length(+,-). % Computes the length of a list.
  - In comments:
    % length(+list,-int): Computes the length of a list.
  - In machine readable comments, to generate documentation:
    %! length(+list,-int): Computes the length of a list.
  - In assertions:
    :- pred length(+list,-int) # "Computes the length of a list.".

Focus for now

- Focus (at least initially) on the terms for argument modes.
- Leave open names of the types or properties (list, int, etc.)
- Eventually extend to cover, e.g., determinism and non-failure.

Is there a problem with argument modes?

- Not issue traditionally - perhaps because only used for doc?
  - But some issues, specially if objective is precise documentation, optimizations, verification...
- E.g., what is the real meaning of - in: `:- mode length(+,-).`

ISO	Variable that will be instantiated if the goal succeeds
SICStus3	Same as ISO.
SICStus4	Argument is an output argument. Usually, but not always, this implies that the argument should be uninstantiated.
SWI	Argument is an output argument. It may or may not be bound at call-time (and acknowledges the problem.)
ECLiPSe	Slightly different depending on use (& used for optimization).
Ciao	Needs precision (c.f. verification) and addresses it by making modes *user definable* and *expanding them to assertions*.

- In PIP: XSB, GNU, LogTalk, etc.

We will use the following notation (from Ciao):

```
:- modedef <argument mode symbol>(<variable>)
   [    : <call properties>     ]
   [  => <success properties> ]
   [   # "<comment>"            ] .
```

`<argument mode symbol>` +, -, @, etc.

`<call properties>` is a property or conjunction of properties that must hold for that argument at call time.

`<success properties>` is a property or conjunction of properties that must hold for that argument at success time.

`<comment>` documentation string.

- Example: `:- modedef +(A) : nonvar(A) => nonvar(A).`
- All fields within `[...]` are optional.
- Properties: `nonvar/1`, `var/1`, `ground/1`, etc.

Summary of proposal for basic modes (WIP)

- Philosophy:
  - stay as close as possible to ISO
  - clarify where it may be ambiguous
  - add new useful modes as needed
- Basic modes:

```
:- modedef +(A) : nonvar(A).
:- modedef ++(A) : ground(A).
:- modedef ?(_).
:- modedef -(A) : var(A) => nonvar(A).
:- modedef -(A) : ivar(A) => nonvar(A).
```

- Modes that are more 'documentation oriented':

```
:- modedef @(A) + not_further_inst(A).
:- modedef -+(A) + steadfast(A).  % or =
```

- Others: `:` (for meta-arguments), `!` (for mutables), etc.

Summary of proposal for parametric modes (WIP):

```
:- modedef  +(P,A)  : (nonvar(A),P(A)).
:- modedef ++(P,A)  : (ground(A),P(A)) => ground(A).
:- modedef  ?(P,A) :: P(A).
:- modedef  -(P,A)  : var(A) => (nonvar(A),P(A)).
:- modedef  @(P,A) :: P(A) + not_further_inst(A).
:- modedef -+(P,A) :: P(A) + steadfast(A). % Or =
```

Some examples of additional parametric modes:

```
:- modedef in(P,A)  : (ground(A),P(A)) => ground(A). % ++
:- modedef out(P,A) : var(A) => (ground(A),P(A)).
:- modedef go(P,A) => (ground(A),P(A)).
```

Here `::` refers to 'type compatibility'.

Thanks!

- URL: https://prolog-lang.org/ImplementersForum/0103-modes.html
- Status: draft
- Discourse (public, please participate!)