Mode indicator	Description	Can be a ground term?	Can be nonground term that is not an unbound variable? (including a “nonempty” open list)	Can be an unbound variable?	Information flows into predicate?	Information flows out of predicate via further instantiation?	Example
++	At call time, the argument must be **ground**, i.e., the argument may not contain any variables that are still unbound.	must	no	no	yes	no	
+	At call time, the argument must be **instantiated** to a term satisfying some (informal) type specification. The argument thus **cannot be an unbound variable**, but need not necessarily be ground. For example, the term [_] is a list, although its only member is the anonymous variable, which is always unbound (and thus nonground).	yes	yes	no	yes	maybe	`foldl(:Goal, +List, +V0, -V)`
?	At call time, the argument must be **bound to a partial term** (a term which may or may not be ground) satisfying some (informal) type specification. Note that an **unbound variable is a partial term**. Think of the argument as either providing input or accepting output or being used for both input and output. For example, in stream_property(S, reposition(Bool)), the reposition part of the term provides input and the unbound-at-call-time Bool variable accepts output.	yes	yes	yes	maybe	maybe	`reverse(?List1, ?List2)` `length(?List, ?Int)`
-	Argument is an output argument. It may or may not be bound at call-time. If the argument is bound at call time, the goal behaves as if the argument were unbound, and then unified with that term after the goal succeeds (sometimes the this is what the procedure implementation literally does). This is what is called being **steadfast**: instantiation of output arguments at call-time does not change the semantics of the predicate, although optimizations may be performed (thus any sequence of immediately performed side-effects may be changed). For example, the goal findall(X, Goal, [T]) is good style and equivalent to findall(X, Goal, Xs), Xs = [T]. Note that any determinism specification, e.g., **det** (“succeed exactly once, leave no choicepoint”) only applies if the relevant argument is indeed unbound. For the case where the argument is bound or involved in constraints and thus more information is available and exploitable by the predicate, **det** effectively becomes **semidet** (“succeeds at most once, leaves no choicepoint”). Similarly, **multi** (“succeeds at least once, but can otherwise exhibit unconstrained behaviour”) effectively becomes **nondet** (“unconstrained behaviour”)	yes	yes	yes (and generally is)	maybe	yes	`max_member(-Max, +List)` `max_list(+List,-Max)`
- -	At call time, the argument must be **unbound**. This is typically used by predicates that create ‘something' and return a handle to the created object, such as open/3, which creates a stream.	no	no	must	no	yes	`open(+SrcDest, +Mode, --Stream)`
:	Argument is a meta-argument, for example a term that can be called as goal. The predicate is thus a meta-predicate. This flag implies +.	yes	yes	no	yes	no	`foldl(:Goal, +List, +V0, -V)`
@	Argument will not be further instantiated than it is at call-time. Typically used for type tests.	yes	yes	yes	yes	no	`var(@Term)`
!	Argument contains a mutable structure that may be modified using setarg/3 or nb_setarg/3.	yes	yes	no	yes	maybe	