No projecto podem usar qualquer predicado existente em Prolog. Os seguintes predicados podem ser importantes para o projecto

maplist(:Goal, ?List)
True if Goal can successfully be applied on all elements of List. Arguments are reordered to gain performance as well as to make the predicate deterministic under normal circumstances.

maplist(:Goal, ?List1, ?List2)
As maplist/2, operating on pairs of elements from two lists.

maplist(:Goal, ?List1, ?List2, ?List3)
As maplist/2, operating on triples of elements from three lists.

maplist(:Goal, ?List1, ?List2, ?List3, ?List4)
As maplist/2, operating on quadruples of elements from four lists.

include(+File) [ISO]
Textually include the content of File in the file in which the directive :- include(File). appears. The include construct is only honoured if it appears as a directive in a source file. Textual include (similar to C/C++ #include) is obviously useful for sharing declarations such as dynamic/1 or multifile/1 by including a file with directives from multiple files that use these predicates.

Textual including files that contain clauses is less obvious. Normally, in SWI-Prolog, clauses are owned by the file in which they are defined. This information is used to replace the old definition after the file has beeen modified and is reloaded by, e.g., make/0. As we understand it, include/1 is intended to include the same file multiple times. Including a file holding clauses multiple times into the same module is rather meaningless as it just duplicates the same clauses. Including a file holding clauses in multiple modules does not suffer from this problem, but leads to multiple equivalent copies of predicates. Using use_module/1 can achieve the same result while sharing the predicates.

Despite these observations, various projects seem to be using include/1 to load files holding clauses, typically loading them
only once. Such usage would allow replacement by, e.g., consult/1. Unfortunately, the same project might use include/1 to share directives. Another example of a limitation of mapping to consult/1 is that if the clauses of a predicate are distributed over two included files, discontiguous/1 is appropriate, while if they are distributed over two consulted files, one must use multifile/1.

To accommodate included files holding clauses, SWI-Prolog distinguishes between the source location of a clause (in this case the included file) and the owner of a clause (the file that includes the file holding the clause). The source location is used by, e.g., edit/1, the graphical tracer, etc., while the owner is used to determine which clauses are removed if the file is modified. Relevant information is found with the following predicates:

- source_file/2 describes the owner relation.
- predicate_property/2 describes the source location (of the first clause).
- clause_property/2 provides access to both source and ownership.
- source_file_property/2 can be used to query include relation-

append(+File)
Similar to tell/1, but positions the file pointer at the end of File rather than truncating an existing file. The pipe construct is not accepted by this predicate.

append(?List1, ?List2, ?List1AndList2)
List1AndList2 is the concatenation of List1 and List2

append(+ListOfLists, ?List)
Concatenate a list of lists. Is true if ListOfLists is a list of lists, and List is the concatenation of these lists.

Arguments_
ListOfLists must be a list of possibly partial lists

nth1(?Index, ?List, ?Elem)
Is true when Elem is the Index'th element of List. Counting starts at 1.
See also nth0/3.

nth1(?N, ?List, ?Elem, ?Rest) [det]
As nth0/4, but counting starts at 1.

length(?List, ?Int) [ISO]
True if Int represents the number of elements in List. This predicate is a true relation and can be used to find the length of a list or produce a list (holding variables) of length Int. The predicate is non-deterministic, producing lists of increasing length if List is a partial list and Int is unbound. It raises errors if
- Int is bound to a non-integer.
- Int is a negative integer.
- List is neither a list nor a partial list. This error condition includes cyclic lists.

This predicate fails if the tail of List is equivalent to Int (e.g., length(L,L)).

subtract(+Set, +Delete, -Result) **[det]**
Delete all elements in Delete from Set. Deletion is based on unification using memberchk/2. The complexity is |Delete|*|Set|.

See also ord_subtract/3.

member(?Elem, ?List)
True if Elem is a member of List. The SWI-Prolog definition differs from the classical one. Our definition avoids unpacking each list element twice and provides determinism on the last element. E.g. this is deterministic:

||____member(X,_[One])._____ ||

author Gertjan van Noord

sort(+List, -Sorted) **[ISO]**
True if Sorted can be unified with a list holding the elements of List, sorted to the standard order of terms (see section 4.7). Duplicates are removed. The implementation is in C, using natural merge sort. The sort/2 predicate can sort a cyclic list, returning a non-cyclic version with the same elements.
msort(+List, -Sorted)
Equivalent to sort/2, but does not remove duplicates. Raises a type_error if List is a cyclic list or not a list.