Prédicats prédéfinis sur les listes

Prédicat		Format	Définition
name/2		name(?Atome, ?ListeCodesAscii)	Unifie un atome avec la liste des codes ASCII des caractères qui le composent
atom_chars/2		atom_chars(?Atome, ?ListeCar)	Unifie un atome avec la liste des atomes caractères qui le composent
is_list/1	*	is_list(+Terme).	Réussit si Terme est une liste.
length/2	*	length(+Liste, +Int).	Unifie Int avec la longueur de la liste.
nth0/3		nth0(?Ind, ?List, ?Elem).	Réussit si l'élément d'indice Ind de la liste List s'unifie avec Elem. Les indices commencent à 0. nth0 sert généralement à récupérer l'élément d'indice Ind dans la liste List.
nth1/3		nth1(?Ind, ?List, ?Elem).	Pareil que nth0, mais avec les indices commençant à 1. Les prédicats nth0/3 et nth1/3 permettent en plus de connaître l'indice où se trouve l'élément de la liste.
last/2		last(?List, ?Elem).	Unifie Elem avec le dernier élément de la liste List.
numlist/3		numlist(+Min, +Max, -List).	Unifie List avec la liste des entiers compris entre Min et Max. Ce prédicat est très utile pour créer des intervalles de valeurs entières, pour symboliser le domaine d'une variable, par exemple. Si l'usage d'une liste n'est pas justifié, préférer l'usage de between/3
reverse/2		reverse(+List1, -List2).	Inverse l'ordre des éléments de List1 et unifie le résultat dans List2.
append/3		append(?List1, ?List2, ?List3).	Réussit si List3 est la concaténation de List1 et List2. append/3 sert principalement pour ajouter le contenu de List2 à la suite de List1 et unifier le résultat dans List3. Cependant, append/3 ne se limite pas à cet usage et peut unifier indifféremment les variables List1, List2 et List3
flatten/2		flatten(+List1, -List2).	Applanit List1 et unifie le résultat dans List2. List1 peut contenir de nombreuses listes imbriquées récursivement. flatten/2 extrait tous les éléments contenus dans List1 et stoque le résultat dans une liste "plane" (à une seule dimension).
sumlist/2		sumlist(+List, -Sum).	Unifie Sum avec la somme de tous les éléments de List.
sublist/3		sublist(+Pred, +List1, ?List2).	Extrait tous les éléments de List1 pour lesquels Pred réussit, unifie le résultat dans List2. Permet d'extraire des éléments vérifiant une propriété donnée.
permutation/2		permutation(?List1, ?List2).	Permet de vérifier si List1 est une permutation de List2 ou inversement. Permet également d'énumérer toutes les permutations possibles de List1 ou List2.
memberchk/2	*	memberchk(?Elem, ?List)	Permet de vérifier si Elem appartient à List. memberchk/2 parcours List et s'arrête au premier élément qui s'unifie avec Elem. memberchk/2 retournera au maximum 1 solution (la première), alors que member/2 retournera autant de solutions qu'il y a d'éléments qui s'unifient avec Elem.
member/2		member(?Elem, ?List).	Réussit si Elem avec un élément de la liste List. Utilisé principalement pour énumérer les membres de la liste.
nextto/3		nextto(?X, ?Y, ?List).	Réussit si X et Y se suivent dans List. Permet : de vérifier si X et Y sont cote-à-cote dans List, de déterminer quel élément suit X (ou précède Y), d'énumérer les éléments X et Y qui se suivent dans List.
select/3		select(?Elem, ?List, ?Rest).	Se comporte comme member/2, sauf qu'en plus il unifie Rest avec la liste List privée de l'élément Elem.
name/2		name(?Atom, ?Number).	Unifie un atome avec la liste des codes ASCII des caractères le composant.
atom_chars/2		atom_chars(?Atom, ?CharList).	Unifie un atome avec la liste de caractères le composant.
msort/2	*	msort(+List, -Triee).	Trie la liste List et unifie le résultat dans Triee. Le tri est très rapide car implémenté en interne en C. L'algorithme utilisé est le tri fusion naturel (rapide et stable). Pour ordonner les éléments, msort/2 se base sur l'ordre standard des termes défini dans Prolog.
sort/2	*	sort(+List, -Triee).	Se comporte comme msort/2, mais supprime les doublons.
merge/3	*	merge(+List1, +List2, -List3).	List1 et List2 sont des listes triées. merge/3 fusionne List1 et List2 pour en faire en une liste triée et unifie le résultat dans List3. merge/3 ne supprime pas les doublons. List1 et List2 doivent être triées (sinon, comportement

Prédicats prédéfinis sur les listes

Prédicat	Format	Définition
maplist/2	maplist(+Pred, +List).	Applique le prédicat Pred à tous les membres d'une liste ou jusqu'à ce que Pred échoue, auquel cas le prédicat maplist/2 échoue. Ce prédicat permet de vérifier que tous les éléments d'une liste vérifient certaines propriétés (grâce au prédicat Pred).
maplist/3	maplist(+Pred, ?List1, ?List2).	Appelle le prédicat Pred (d'arité 2 au minimum) avec comme arguments les membres de List1 et List2. Ce prédicat sert à appliquer le prédicat Pred sur les membres d'une liste et à mettre le résultat dans l'autre.
maplist/4	maplist(+Pred, ?List1, ?List2, ?List3).	Appelle le prédicat Pred (d'arité 3 au minimum) avec comme arguments les membres de List1, List2 et List2. Ce prédicat sert à appliquer le prédicat Pred sur les membres de 2 listes et à mettre le résultat dans une troisième.
is_set/1	is_set(+Terme).	Réussit si Terme est un ensemble c'est-à-dire une liste ne comportant pas de doublons.
list_to_set/2	list_to_set(+List, -Set).	Transforme une liste en set. En d'autres termes, supprime tous les doublons de List et unifie le résultat dans Set. list_to_set/2 conserve l'ordre des éléments (si la liste contient des doublons, seul le premier est gardé)
subset/2	subset(+Subset, +Set).	Réussit si tous les éléments de Subset appartiennent à Set. subset/2 ne vérifie pas que Subset et Set ne contiennent pas de doublons.
intersection/3	intersection(+Set1, +Set2, -Set3).	Unifie Set3 avec l'intersection de Set1 et Set2. Les listes n'ont pas besoin d'être ordonnées. Si Set1 et Set2 contiennent des doublons, Set3 est susceptible de contenir des doublons. Utilisez list_to_set/2 pour supprimer les doublons.
union/3	union(+Set1, +Set2, -Set3).	Unifie Set3 avec l'union de Set1 et Set2. Les listes n'ont pas besoin d'être ordonnées. Aucune vérification n'est faite pour garantir que Set1 et Set2 ne contiennent pas de doublons. Si Set1 et Set2 ne contiennent pas de doublons, Set3 ne contiendra pas de doublon.
merge_set/3	merge_set(+Set1, +Set2, -Set3).	Set1 et Set2 sont des listes triées, supposées sans doublons. merge_set/3 fusionne Set1 et Set2 pour en faire en une liste triée sans doublons et unifie le résultat dans Set3. Set1 et Set2 doivent être triés (sinon, comportement aléatoire), Set1 et Set2 ne doivent pas contenir de doublons.
subtract/3	subtract(+Set, +Delete, -Rest).	Supprime de Set tous les éléments contenus dans Delete et unifie le résultat dans Rest. Pour des raisons de performance, aucune vérification n'est faite pour garantir que Set et Set2 ne contient pas de doublons.
findall/3	findall(+Motif, +But, -Liste).	Le prédicat findall/3 permet de faire la liste des solutions d'un prédicat : findall/3 créé une liste des instanciations de Motif par retours arrières sur But et unifie le résultat dans Liste. Si le But n'a pas de solution, findall retournera la liste vide: [].
bagof/3	bagof(+Motif, +But, -Liste).	Se comporte comme findall/3, à la différence que bagof/3 échoue si But n'a pas de solution
setof/3	setof(+Motif, +But, -Liste).	Se comporte comme bagof/3, à la différence qu'il y a suppression des doublons et les résultats sont triés
Exemple :	foo(a, b, c).	?- findall(C, foo(A,B,C), L). ⟶ L = [c, f, d, f, f, e, g]
	foo(a, b, f).	?- bagof(C, foo(A,B,C), L). ⟶ A = a, B = b, L = [c, f, d, f] ;
	foo(a, b, d).	⟶ A = b, B = c, L = [f, e] ;
	foo(a, b, f).	⟶ A = c, B = c, L = [g]
	foo(b, c, f).	?- setof(C, foo(A,B,C), L). ⟶ A = a, B = b, L = [c, d, f] ;
	foo(b, c, e).	⟶ A = b, B = c, L = [e, f] ;
	foo(c, c, g).	⟶ A = c, B = c, L = [g]