SWI Prolog
Predykaty wbudowane służące do przetwarzania list

is_list(+Term)
Spełniony, jeżeli Term jest związany z listą pustą ([]) lub z termem o dwuargumentowym funktorze "." i drugi argument jest listą.
Definicja:

```
is_list(X):- var(X),!,fail.
is_list([]).
is_list([_|T]):- is_list(T).
```

---

append(?*List1*, ?*List2*, ?*List3*)
Spełniony, jeżeli *List3* daje się uzgodnić z konkatenacją list *List1* i *List2*. Predykat append może być użyty z dowolnymi podtawieniami (nawet wszystkie trzy zmienne).

---

member(?*Elem*, ?*List*)
Spełniony, jeżeli *Elem* daje się uzgodnić z jakimś elementem listy *List*. Może być użyty przy dowolnych podstawieniach.

---

memberchk(?*Elem*, +*List*)
Równoważny predykatowi member/2, ale podaje tylko jedno rozwiązanie.

---

nextto(*?X,?Y,?List*)
Spełniony, gdy Y następuje bezpośrednio po *X* na liście *List*.

---

delete(+*List1*, ?*Elem*, ?*List2*)
Z listy *List1* usuwa wszystkie wystąpienia elementu *Elem* i listę wynikową uzgadnia z listą *List2*.

---

select(?*Elem*, ?*List* , ?*Rest*)
Z listy *List* wybiera element, który daje się uzgodnić z *Elem*. Lista *Rest* jest następnie uzgadniana z listą, która powstaje z listy *List* po usunięciu wybranego elementu. Jeżeli *Elem* występuje na liście *List* więcej niż jeden raz, otrzymujemy rozwiązania alternatywne. Z reguły predykat select/3 jest używany przy podstawiniach: -*Elem*, +*List*, -*Rest*, ale może być również użyty do wstawienia elementu do listy: +*Elem*,-*List*, +*Rest*.

---

nth0(?*Index*, ?*List*, ?*Elem*)
Spełniony, jeżeli element listy *List* o numerze *Index* daje się uzgodnić z elemntem *Elem*. Elementy listy są numerowane poczynając od 0.

---

nth1(?*Index*, ?*List*, ?*Elem*)
Spełniony, jeżeli element listy *List* o numerze *Index* daje się uzgodnić z elemntem *Elem*. Elementy listy są numerowane poczynając od 1.

---

last(*?List, ?Elem*)
Spełniony, jeżeli element *Elem* daje się uzgodnić z ostatnim elementem listy *List*. Jeżeli lista *List* jest listą właściwą predykat last/2 jest detrministyczny. Jeżeli lista *List* posiada nieograniczony ogon, mechanizm nawracania będzie powodował zwiększanie długości listy *List*.

---

reverse(*+List1,-List2*)
Odwraca porządek elementów listy *List1* i unifikuje rezultat z listą *List2*.

---

permutation(*?List1,?List2*)
Spełniony, gdy lista *List1* jest permutacją listy *List2*.

---

flatten(*+List1,-List2*)
Przekształca listę *List1*, której elementy mogą być również listami w listę *List2*, w której każda lista składowa zostaje zastąpiona przez swoje elementy (rekurencyjnie).
Przykład:
```
?- flatten([a,[b,[c,d],e,f]],X).
X = [a,b,c,d,e,f]
```

---

sumlist(*+List,-Sum*)
Unifikuje *Sum* z sumą elementów listy liczbowej *List*.

---

numlist(*+Low,+High,-List*)
Jeżeli *Low* i *High* są liczbami całkowitymi (integers) takimi, że *Low* =< *High*, to lista *List* zostanie zunifikowana z listą [*Low,Low+1,...,High*].

---

length(*?List,?Int*)
Spełniony, jeżeli liczba naturalna *Int* repezentuje liczbę elementów listy *List*. Predykat ten może być użyty do tworzenia list zawierających tylko zmienne.

---

merge(*+List1,+List2,-List3*)
Listy *List1* i *List2* są listami uporządkowanymi zgodnie ze standardowym porządkiem termów. *Lista3* będzie uporządkowaną listą zawierającą elemnty list *List1* i *List2*. Elementy powtarzające się **nie są** usuwane.

---

sort(*+List,-Sorted*)
Spełniony, jeżeli lista *Sorted* daje się zunifikować z listą zawierającą elementy listy *List* uporządkowane według standardowego porządku termów. Elementy powtarzające się są usuwane.

---

msort(*+List,-Sorted*)
Równoważny predykatowi sort/2, ale elementy powtarzające się nie są usuwane.

SWI Prolog
Wywoływanie predykatów dla wszystkich elementów listy.

maplist(*+Pred,+List*)
Spełniony, jeżeli predykat *Pred* jest spełniony dla wszystkich elementów listy *List*. Predykat *Pred* powinien być tak określony, aby argument za który mają być podstawiane elementy listy *List* był ostatnim argumentem tego predykatu. Argument ten pomijamy przy wywołaniu.

---

maplist(*+Pred,?List1,?List2*)
Spełniony, jeżeli predykat Pred jest spełniony dla każdej pary odpowiadających sobie elementów listy *List1* i *List2*.

---

maplist(*+Pred,?List1,?List2,?List3*)
Spełniony, jeżeli predykat Pred jest spełniony dla każdej trójki odpowiadających sobie elementów list *List1*, *List2* i *List3*

---

sublist(*+Pred,+List1,?List2*)
Unifikuje listę *List2* z listą zawierającą elementy listy *List1*, dla których spełniony jest predykat *Pred*.

---

include(*+Pred,+List1,?List2*)
Unifikuje listę *List2* z listą zawierającą elementy listy *List1*, dla których spełniony jest predykat *Pred*.

---

exclude(*+Pred,+List1,?List2*)
Unifikuje listę *List2* z listą zawierającą elementy listy *List1*, dla których nie jest spełniony predykat *Pred*.

---

partition(*+Pred, +List, ?Included, ?Excluded*)
Unifikuje listę *Included* z listą zawierającą elementy listy *List*, dla których spełniony jest predykat *Pred*, a listę *Excluded* z listą zawierającą elementy listy *List*, dla których nie jest spełniony predykat *Pred*.

SWI Prolog
Predykaty do przetwarzania zbiorów.

is_set(*+Set*)
Spełniony, gdy *Set* jest listą właściwą bez elementów powtarzających się.

---

list_to_set(*+List,-Set*)
Unifikuje zbiór *Set* z listą zawierającą elementy listy *List* w tej samej kolejności. Jeżeli lista *List* zawiera elementy powtarzające się, do zbioru *Set* włączany jest pierwszy z nich.

---

intersection(*+Set1,+Set2,-Set3*)
Spełniony, jeżeli zbiór *Set3* daje się zunifikować z częścią wspólną zbiorów *Set1* i *Set2*.

---

subtract(*+Set,+Delete,-Result*)
Usuwa wszystkie elementy zbioru *Delete* ze zbioru *Set* i rezultat unifikuje ze zbiorem *Result*.

---

union(*+Set1,+Set2,-Set3*)
Spełniony, jeżeli *Set3* daje się zunifikowac z sumą zbiorów *Set1* i *Set2*.

---

subset(*+Subset,+Set*)
Spełniony, jeżeli wszystkie elementy zbioru *Subset* są elementami zbioru *Set*.

---

merge_set(*+Set1,+Set2,-Set3*)
Zbiory (listy bez powtórzeń) *Set1* i *Set2* są uporządkowane według standardowego uporządkowania termów. Zbiór *Set3* jest uporządkowaną sumą zbiorów *Set1* i *Set2*.
