Aggregate functions of X-DEVICE

This page contains the details for implementing the `list` and `ord_list` aggregate functions of X-DEVICE in DEVICE. The details of how the information below is used by the DEVICE system can be found in http://www.csd.auth.gr/~lpis/publications/TKDE-12.html.

The `list` aggregate function

The object that represents the `list` aggregate function using normal Prolog lists is:

```
instance_of       aggregate_function
attributes
   name                 list
   initial_value  []
methods
   positive_next/2
      positive_next([OldList,NewElement],NextList) :-
         append(OldList,[NewElement],NextList).
   negative_next/4
      negative_next([OldList,DelElement,_,_],NextList) :-
         delete(OldList,DelElement,NextList).
   calc_result/2
      calc_result(List,List).
```

The same function can be implemented more efficiently using difference lists:

```
instance_of       aggregate_function
attributes
   name                 list
   initial_value  T-T
methods
   positive_next/2
      positive_next([OldList-T,NewElement],OldList-T1) :-
         T = [NewElement|T1].
   negative_next/4
      negative_next([OldList-T,DelElement,_,_],NextList-T) :-
         delete(OldList,DelElement,NextList).
   calc_result/2
      calc_result(List,List).
```

The `ord_list` aggregate function

The object that represents the `ord_list` aggregate function using normal Prolog lists is:

```
instance_of       aggregate_function
attributes
   name                 ord_list
   initial_value  []
methods
   positive_next/2
      positive_next([OldList,X-Vars-Order],NextList) :- !,
         insert_sorted(Order,OldList,X-Vars,NextList).
      positive_next([OldList,X-Vars],NextList) :- !,
         insert_sorted(OldList,X-Vars,NextList).
      positive_next([OldList,X],NextList) :-
         insert_sorted1(OldList,X,NextList).
   negative_next/4
      negative_next([OldList,X-Vars-Order,_,_],NextList) :- !,
         delete_sorted(Order,OldList,X-Vars,NextList).
```

```
      negative_next([OldList,X-Vars,_,_],NextList) :- !,
         delete_sorted(OldList,X-Vars,NextList).
      negative_next([OldList,X,_,_],NextList) :-
         delete_sorted1(OldList,X,NextList).
   calc_result/2
      calc_result([],[]).
      calc_result([X-_Vars|Tail1],[X|Tail2]) :- !,
         calc_results(Tail1,Tail2).
      calc_result(List,List).
```

In the above methods, the following auxiliary predicates were used:

```
% insert_sorted/4
insert_sorted(_,[],X,[X]).
insert_sorted(Order,[X1-Vars1|T],X2-Vars2,[X2-Vars2,X1-Vars1|T]) :-
   is_before(Order,Vars2,Vars1), !.
insert_sorted(Order,[X1-Vars1|T],X2-Vars2,[X1-Vars1|NT]) :-
   insert_sorted(Order,Tail,X2-Vars2,NT).

% insert_sorted/3
insert_sorted([],X,[X]).
insert_sorted([X1-Vars1|Tail],X2-Vars2,[X2-Vars2,X1-Vars1|Tail]) :-
   Vars2 @=< Vars1.
insert_sorted([X1-Vars1|Tail],X2-Vars2,[X1-Vars1|NewTail]) :-
   Vars2 @> Vars1,
   insert_sorted(Tail,X2-Vars2,NewTail).

% insert_sorted1/3
insert_sorted([],X,[X]).
insert_sorted([X1|Tail],X2,[X2,X1|Tail]) :-
   X2 @=< X1.
insert_sorted([X1|Tail],X2,[X1|NewTail]) :-
   X2 @> X1,
   insert_sorted(Tail,X2,NewTail).

is_before([],[],[]).
is_before([Op|_],[H1|_],[H2|_]) :-
   H1 Op H2, !.
is_before([_|RestOps],[H|T1],[H|T2]) :-
   is_before(RestOps,T1,T2).

% delete_sorted/4
delete_sorted(_,[],_,[]).
delete_sorted(_,[X1-Vars1|Tail],X1-Vars1,Tail) :- !.
delete_sorted(Order,[X1-Vars1|Tail],X2-Vars2,[X1-Vars1|Tail]) :-
   is_before(Order,Vars2,Vars1), !.
delete_sorted(Order,[X1-Vars1|Tail],X2-Vars2,[X1-Vars1|NewTail]) :-
   delete_sorted(Order,Tail,X2-Vars2,NewTail).

% delete_sorted/3
delete_sorted([],X,[]).
delete_sorted([X1-Vars1|Tail],X1-Vars1,Tail).
delete_sorted([X1-Vars1|Tail],X2-Vars2,[X1-Vars1|Tail]) :-
   Vars2 @< Vars1.
delete_sorted([X1-Vars1|Tail],X2-Vars2,[X1-Vars1|NewTail]) :-
   Vars2 @> Vars1,
   delete_sorted(Tail,X2-Vars2,NewTail).

% delete_sorted1/3
delete_sorted([],X,[]).
delete_sorted([X1|Tail],X1,Tail) :- !.
delete_sorted([X1|Tail],X2,[X1|Tail]) :-
   X2 @< X1.
delete_sorted([X1|Tail],X2,[X1|NewTail]) :-
   X2 @> X1,
   delete_sorted(Tail,X2,NewTail).
```

The `string` aggregate function

The object that represents the `string` aggregate function is:

```
instance_of       aggregate_function
attributes
   name                 string
```

```
   initial_value  ''
methods
   positive_next/2
      positive_next([Old,X],New) :-
         concat_atoms(Old,X,New).
   negative_next/4
      negative_next([Old,X,_,_],New) :-
         del_atom(Old,X,New).
   calc_result/2
      calc_result(X,X).
```

In the above methods, the following auxiliary predicate (in ECLiPSe Prolog v5.2) was used:

```
del_atom(Big,Small,Left) :-
   atom_string(Big,SBig), atom_string(Small,SSmall),
   substring(SBig, SSmall, Position),
   string_length(SBig,Length1), string_length(SSmall,Length2),
   P1 is Position-1, P2 is Position+Length2, P3 is Length1-P2+1,
   substring(SBig, 1, P1, SBegin),
   substring(SBig, P2, P3, SEnd),
   concat_strings(SBegin,SEnd,SLeft), atom_string(Left,SLeft).
```