PENGGUNAAN LIST DALAM PROLOG

Pada bagian ini akan dibahas penggunaan list dalam PROLOG untuk beberapa kasus. Kasus tersebut dipilih karena dipandang dari aspek eksekusi berbeda satu dengan lainnya. Selain itu, yang perlu diperhatikan adalah query apa saja yang dapat dibentuk (dan mewakili tujuan/arti apa) serta apakah query itu benar atau salah. Kebenaran query dikaitkan dengan spesifikasi fakta dan aturannya.

1. Kasus Predikat Murni
2. Kasus dengan Kalkulasi (hasil perlu 'dihitung')
3. Kasus Predikat dengan Output Elemen
4. Kasus Predikat dengan Output List (list perlu dibentuk)
5. Kasus Predikat dengan Output List menggunakan Akumulator

1. KASUS PREDIKAT MURNI

Apakah X anggota list **IsMember(X,Ls)**

SPESIFIKASI

{ IsMember(X,Ls) benar, jika X adalah elemen dari list Ls dg representasi [X|Xs],
Basis :

- *jika X merupakan head dari list Ls, maka X adalah anggota list*

Rekurens:

- *jika X adalah anggota dari tail(Ls), maka X juga akan menjadi anggota dari Tail(Ls) yang headnya apapun! }*

FAKTA DAN ATURAN

{ Basis }

```
{  IsMember(X,[]) is false ! }
{ Perhatikanlah bahwa IsMember (X,[X]) sudah tercakup dalam aturan sbb}
   IsMember(X,[X|Xs]). {  ←  X=X dan Xs boleh list kosong [] }
```

{ Rekurens }

```
   IsMember(X,[Y|Xs]) ←  X ≠ Y, IsMember (X,Xs).
```

FAKTA DAN ATURAN DALAM PROLOG

```
   1. isMember(X,[X|_]).                    { ← X = X }
   2. isMember(X,[Y|Xs]) :-  X <> Y,
                             isMember(X,Xs).  { ← X ≠ Y dan Xs boleh [] }
```

KEMUNGKINAN QUERY: benar / salah ?

isMember (2, [3,4]) ?
isMember (2, W) ?
isMember (V, [3]) ?
isMember (V,W) ?

2. KASUS DENGAN KALKULASI

PANJANG LIST	**Length (Ls, N)**

SPESIFIKASI

{ Length(Ls,N) benar, jika N adalah panjang dari list Ls dengan representasi [X|Xs]}
{ Basis : list kosong panjangnya 0}
{ Rekurens : jika N adalah panjang dari sebuah list Xs,
maka Succ(N) adalah panjang dari list [X|Xs] dengan X sembarang . }

FAKTA DAN ATURAN

{Basis :}

```
Length([],0).
```

{ Rekurens }

```
Length([X|Xs],Succ(N)) ← Length(Xs,N).
```

FAKTA DAN ATURAN DALAM PROLOG: dengan domain

```
1. length([],0).
2. length([_|Xs], M) :- length(Xs,N),
                        M = N + 1.
```

3. KASUS PREDIKAT DENGAN OUTPUT ELEMEN

ELEMEN TERAKHIR SEBUAH LIST	**LastElmt (Ls,X)**

SPESIFIKASI

{ LastElmt(Ls, X) benar, artinya X adalah element terakhir dari list Ls,
***X adalah parameter hasil**, dan hasil baru diperoleh setelah tercapai elemn list yang terakhir*
Basis : jika X adalah anggota dari list dengan 1 elemen yaitu X,
Rekurens : jika X adalah elemen terakhir dari Tail list asal, maka X adalah lemen dari List asal
LastElmt ([Y|Xss], X) ← LastElmt(Xs,X) }

FAKTA DAN ATURAN

{ Basis }
{ LastElmt ([],X) is false }

```
LastElmt([X],X). { ← X=X }
```

{ Rekurens }

```
LastElmt([Y|Xs],X) ←  X ≠ Y, LastELmt(Xs,X).   {Xs boleh kosong }
```

FAKTA DAN ATURAN DALAM PROLOG: dengan domain

```
1. lastElmt([X],X).                        { ← X=X }
2. lastElmt([Y|Xs],X) :-  X <> Y,
                          lastELmt(Xs,X).  {Xs boleh kosong }
```

4. KASUS PREDIKAT DENGAN OUTPUT LIST

Menghapus sebuah elemen list **REMBER (X,L1s,L2s)**

SPESIFIKASI

{ Rember(L1s,X,L2s)benar, artinya L2s adalah hasil dari penghapusan sebuah elemen bernilai X dari L1s}
{ Konstruksi program :
Basis : tidak ada element yang dapat dihapus dari sebuah list kosong.
Hasil penghapusan elemen terhadap sebuah list kosong adalah list kosong.}
Rekurens: list tidak kosong :
jika Ys adalah hasil dari melakukan penghapusan X terhadap Xs, maka dengan menambahkan sebuah elemen X sebagai Head dari Xs dan juga menambahkan X sebagai Head dari Ys, Rember(Xs,X,Ys) tetap benar
jika Ys adalah hasil dari melakukan penghapusan X terhadap Xs, maka dengan menambahkan sebuah elemen X sebagai Head dari Xs dan Y terhadap Ys, maka
Rember ([Y|Xs], X, Concat([Y],Rember (Xs,Y), Zs) }

FAKTA DAN ATURAN

```
{ Basis }
  Rember(X,[],[]).
{Rekurens }
  Rember(X,[X|Xs],Xs).  {← X = X. dan Xs boleh list kosong}
  Rember(X,[Y|Xs],[Y|Ys]) ← X ≠ y, Rember(X,Xs,Ys).
```

FAKTA DAN ATURAN

```
1. rember(X,[],[]).
2. rember(X,[X|Xs],Xs).            {← X = X. dan Xs boleh list kosong}
3. rember(X,[Y|Xs],[Y|Ys]) ← X <> Y,
                                   rember(X,Xs,Ys).
```

CONTOH LAIN DARI KASUS 4:

Membalik List (versi-1)	**Inverse (L1s,L2s)**

SPESIFIKASI

{Inverse(L1s, L2s) benar, artinya L2s adalah hasil dari inverse L1s: List L2s urutan elemennya terbalik dibandingkan terhadap urutan elemen L1s. Dengan kata lain, L2s "dibentuk" dari L1s yang dibalik. Maka L2s adalah parameter hasil}

Basis : membalik teks kosong, hasilnya teks kosong

Rekurens : Jika Zs adalah hasil dari melakukan inverse terhadap Xs, maka dengan menambahkan sebuah elemen X sebagai Head pada Xs, Zs harus dikonkatenasi dengan [X]
Inverse([X|Xs], Concat(Inverse(Xs,Zs),[X], L2s) ← Inverse (Xs,Zs)}

FAKTA DAN ATURAN

{ Basis }

```
Inverse([], []).
```

{ Rekurens }

```
Inverse ([X|Xs], L2s ) ←  Inverse (Xs, Zs),
                          ConCat ( Zs, [X], L2s]).
```

FAKTA DAN ATURAN

```
1. inverse ([], []).

2. inverse ([X|Xs], L2s ) ←  inverse (Xs, Zs),
                             concat (Zs, [X], L2s]).
```

5. KASUS PREDIKAT DENGAN OUTPUT LIST MENGGUNAKAN AKUMULATOR

Membalik list dengan cara "akumulasi" (versi-2)

SPESIFIKASI

{ Basis : jika list Xs dibalik dan diakumulasi pada list kosong, hasilnya adalah list Ys }
{Revacc (reverse accumulate) : RevAcc(Xs,Ys,Zs): Zs adalah hasil dari
mengkonkatenasi akumulator Ys ke Xs yang sudah dibalik
Basis : hasil membalik list kosong dan kemudian mengkonkatenasi dengan Ys adalah Ys
Rekurens : jika Ys adalah hasil konkatenasi Xs yang sudah dibalik dengan X,
maka Head dari list asal sama dengan Head dari akumulator}

FAKTA DAN ATURAN

{Basis : }

```
RevAcc([],Ys,Ys).
```

{Rekurens }

```
RevAcc([X|Xs],Accs,Ys) ← RevAcc(Xs,[X|Accs],Ys).
```

{Realisasi pemanggilan }

```
Reverse(Xs,Ys) ← RevAcc(Xs,[],Ys).
```

FAKTA DAN ATURAN

```
1. revAcc([],Ys,Ys).

2. revAcc([X|Xs],Accs,Ys) ← revAcc(Xs,[X|Accs],Ys).

3. reverse(Xs,Ys) ← revAcc(Xs,[],Ys).
```