PREDIKAT SISTEM PADA PROLOG
CUT & FAIL

Definisi Cut

Cut adalah salah satu predikat sistem yang disediakan Prolog.
Secara konsep, cut digunakan untuk merealisasikan analisa kasus pada Prolog.

Jenis Cut

Terdapat 2 (dua) tujuan penggunaan cut yang menyebabkannya menjadi jenis yang berbeda:

1. Green Cut : cut digunakan untuk efisiensi eksekusi, karena jika cut dihilangkan arti program tidak berubah.
2. Red Cut : cut digunakan untuk kebenaran eksekusi, karena jika cut dihilangkan arti program berubah.

Penggunaan Cut

Kenapa program Prolog memanfaatkan Cut?
Tujuannya agar eksekusi program lebih efisien. Sehingga, pada prinsipnya, tanpa cut pun kita dapat membuat program Prolog yang dapat dieksekusi dengan benar.

Lalu, kapan kita harus memakai cut?
Jika dengan penggunaan cut proses pencarian solusi dapat dibatasi, sehingga mempercepat pencapaian solusi atau hasil dari program.
Hal ini dimungkinkan, karena cut memotong pohon eksekusi sehingga cabang yang tidak menghasilkan solusi tidak perlu dijalani/dicoba, dengan cara menghilangkan proses backtrack ke cabang tersebut.

Tahapan implementasi Cut

Untuk mengimplementasikan predikat cut dengan benar, maka tahapan implementasinya adalah sbb:

1. Tuliskan terlebih dulu semua kondisi eksplisit
2. Tambahkan predikat cut sesuai tujuannya, sebagai green cut atau red cut.

Berikut ini diberikan dua contoh realisasi analisa kasus, dan dua contoh kasus nyata.

Efek Penggunaan Cut

Efek penggunaan cut dijelaskan dengan contoh berikut:

```
A :- B, C, D, E, F.
Z :- B, C, !, D, E, F.
```

Pada predikat A:
untuk mencari solusinya, Prolog dapat menelusuri seluruh kemungkinan solusi dengan cara melakukan backtrack ke setiap nilai predikat B,C,D,E,F.

Pada predikat Z:
untuk mencari solusinya, setelah predikat cut dilewati, Prolog tidak dapat melakukan backtrack ke predikat di sebelah kiri predikat cut. Sehingga Prolog hanya dapat melakukan backtrack ke setiap nilai predikat D,E,F.

Cut digunakan untuk implementasi ekspresi **IF-THEN-ELSE**

IF_THEN_ELSE **versi tanpa cut**

FAKTA DAN ATURAN

{ if_then_else (P,Q,R) akan menghasilkan Q jika P benar, atau menghasilkan R jika P bernilai false. Dengan aturan sbb: dilakukan evaluasi kondisi pada setiap aturan. Maka aturan berikut mewakili if_then_else, tetapi tanpa menggunakan predikat cut }

```
if_then_else(P,Q,R) ←  P, Q.
if_then_else(P,Q,R) ←  not P, R.
```

IF_THEN_ELSE **versi dengan green cut**

FAKTA DAN ATURAN

{ if_then_else (P,Q,R) akan menghasilkan Q jika P benar, atau menghasilkan R jika P bernilai false. Dengan aturan sbb: dilakukan evaluasi kondisi pada setiap aturan. Maka aturan berikut mewakili if_then_else, tetapi menggunakan predikat cut }

```
if_then_else(P,Q,R) ←  P, !, Q.
if_then_else(P,Q,R) ←  not P, R.
```

IF_THEN_ELSE **versi dengan red cut**

FAKTA DAN ATURAN

{ if_then_else (P,Q,R) akan menghasilkan Q jika P benar, atau menghasilkan R jika P bernilai false. Dengan aturan sbb: dilakukan evaluasi kondisi pada setiap aturan. Maka aturan berikut mewakili if_then_else, tetapi menggunakan predikat cut }

```
if_then_else(P,Q,R) ←  P, !, Q.
if_then_else(P,Q,R) ←  R.
```

Cut digunakan untuk implementasi ekspresi **CASE**

CASE - 4 kondisi **versi tanpa cut**

FAKTA DAN ATURAN

{ case (P,Q,R,S,T) benar, artinya merealisasikan case sbb:

- *jika P = <kondisi-1> , maka menghasilkan Q.*
- *jika P = <kondisi-2> , maka menghasilkan R.*
- *jika P = <kondisi-3> , maka menghasilkan S.*
- *jika P = <kondisi-4> , maka menghasilkan T.}*

```
case(P,Q,R,S,T)      ← P = <kondisi-1>, Q.

case(P,Q,R,S,T)      ← P = <kondisi-2>, R.

case(P,Q,R,S,T)      ← P = <kondisi-3>, S.

case(P,Q,R,S,T)      ← P = <kondisi-4>, T.
```

CASE - 4 kondisi **versi dengan green cut**

FAKTA DAN ATURAN

{ case (P,Q,R,S,T) benar, artinya merealisasikan case sbb:

- *jika P = <kondisi-1> , maka menghasilkan Q.*
- *jika P = <kondisi-2> , maka menghasilkan R.*
- *jika P = <kondisi-3> , maka menghasilkan S.*
- *jika P = <kondisi-4> , maka menghasilkan T.}*

```
case(P,Q,R,S,T)      ← P = <kondisi-1>, !, Q.

case(P,Q,R,S,T)      ← P = <kondisi-2>, !, R.

case(P,Q,R,S,T)      ← P = <kondisi-3>, !, S.

case(P,Q,R,S,T)      ← P = <kondisi-4>, !, T.
```

CASE - 4 kondisi **versi dengan red cut**

FAKTA DAN ATURAN

{ case (P,Q,R,S,T) benar, artinya merealisasikan case sbb:

- *jika P = <kondisi-1> , maka menghasilkan Q.*
- *jika P = <kondisi-2> , maka menghasilkan R.*
- *jika P = <kondisi-3> , maka menghasilkan S.*
- *jika P = <kondisi-4> , maka menghasilkan T.}*

```
case(P,Q,R,S,T)      ← P = <kondisi-1>, !, Q.

case(P,Q,R,S,T)      ← P = <kondisi-2>, !, R.

case(P,Q,R,S,T)      ← P = <kondisi-3>, !, S.

case(P,Q,R,S,T)      ← T.
```

Contoh 1 : Dengan penambahan cut, efisiensi program hanya meningkat sedikit.

Maksimum dua harga **versi tanpa cut**

FAKTA DAN ATURAN

{ Max2(X,Y, Z) benar, artinya Z adalah maksimum dari X dan Y.
Max2(X,Y,X) benar, jika X >= Y
Max2(X,Y,Y) benar, jika X <= Y }

```
Max2(X,Y,X) ← X >= Y.
Max2(X,Y,Y) ← X < Y.
```

Contoh 1a : GREEN CUT

Maksimum dua harga **versi dengan green cut**

FAKTA DAN ATURAN

{ Max2(X,Y, Z) benar, artinya Z adalah maksimum dari X dan Y.
Max2(X,Y,X) benar, jika X >= Y
Max2(X,Y,Y) benar, jika X <= Y }

```
Max2(X,Y,X) ← X >= Y, !.
Max2(X,Y,Y) ← X < Y.
```

Contoh 1b : RED CUT

Maksimum dua harga **versi dengan red cut**

FAKTA DAN ATURAN

{ Max2(X,Y, Z) benar, artinya Z adalah maksimum dari X dan Y.
Max2(X,Y,X) benar, jika X >= Y
Max2(X,Y,Y) benar, jika X <= Y }

```
Max2(X,Y,X) ← X >= Y, !.
Max2(X,Y,Y).
```

Contoh 2 : Dengan penambahan cut, efisiensi program meningkat dengan besar.

Penentuan Jenis Character **versi tanpa cut**

FAKTA DAN ATURAN

{ WriteOpr(Ch) benar, artinya jika :

- *Ch adalah operator aritmatika maka akan dituliskan: "operator aritmatika"*
- *Ch bukan operator aritmatika maka akan dituliskan: "bukan operator aritmatika"*

*dengan operator aritmatika: * / + - }*

```
WriteOpr(Ch) :- OprArit(Ch),
                writeln("Operator Aritmatika").
WriteOpr(Ch) :- not(OprArit(Ch)),
                writeln("Bukan Operator Aritmatika").
```

*{ OprArit(Ch) benar, artinya Ch adalah operator aritmatika: * / + - }*

```
OprArit("*").
OprArit("/").
OprArit("+").
OprArit("-").
```

Contoh 2a : GREEN CUT

Penentuan Jenis Character **versi dengan green cut**

FAKTA DAN ATURAN

{ WriteOpr(Ch) benar, artinya jika :

- *Ch adalah operator aritmatika maka akan dituliskan: "operator aritmatika"*
- *Ch bukan operator aritmatika maka akan dituliskan: "bukan operator aritmatika"*

*dengan operator aritmatika: * / + - }*

```
WriteOpr(Ch) :- OprArit(Ch), !,
                writeln("Operator Aritmatika").
WriteOpr(Ch) :- not(OprArit(Ch)),
                writeln("Bukan Operator Aritmatika").
```

*{ OprArit(Ch) benar, artinya Ch adalah operator aritmatika: * / + - }*

```
OprArit("*").
OprArit("/").
OprArit("+").
OprArit("-").
```

Contoh 2b : RED CUT

Penentuan Jenis Character **versi dengan red cut**

FAKTA DAN ATURAN

{ WriteOpr(Ch) benar, artinya jika :

- *Ch adalah operator aritmatika maka akan dituliskan: "operator aritmatika"*
- *Ch bukan operator aritmatika maka akan dituliskan: "bukan operator aritmatika"*

*dengan operator aritmatika: * / + - }*

```
WriteOpr(Ch) :- OprArit(Ch), !,
                writeln("Operator Aritmatika").
WriteOpr(Ch) :- writeln("Bukan Operator Aritmatika").
```

*{ OprArit(Ch) benar, artinya Ch adalah operator aritmatika: * / + - }*

```
OprArit("*").
OprArit("/").
OprArit("+").
OprArit("-").
```

Selain itu, berikut ini diberikan contoh penggunaan macam-macam predikat cut.

Contoh: GREEN CUT

Member

FAKTA DAN ATURAN

{Keanggotaan X sebagai elemen list: Member(X,Ls) benar, jika ada elemen list L yang bernilai X }

{Basis }

```
Member (X, [X|Xs])    :- !.
```

{Recc }

```
Member (X, [Y|Xs])    :- X <> Y, member (X,Ys).
```

Contoh: RED CUT

Delete

FAKTA DAN ATURAN

{ Menghapus semua kemunculan X dalam sebuah list : Delete (X,Ls, Lhs) benar, jika Lhs adalah hasil menghapus semua kemunculan X pada Ls }

{Basis }

```
Delete ([], X, [])       :- !.
```

{Recc }

```
Delete ([X|Ys], X, Zs)      :- !, delete (Ys, X, Zs).
Delete ([Y|Ys], X, [Y|Zs]) :- delete (Ys, X, Zs).
```

Definisi Predikat FAIL

Predikat FAIL juga salah satu predikat sistem yang disediakan Prolog.
Secara konsep, FAIL digunakan untuk memanipulasi program sehingga suatu proses tertentu harus dilakukan. Tetapi, jika dikombinasikan cut dan fail, secara konsep, digunakan untuk merealisasikan ***STOP Statement*** pada Prolog, artinya untuk memaksa program Prolog berhenti pada satu kasus tertentu.

Penggunaan Predikat FAIL

Kenapa program Prolog memanfaatkan FAIL?
Umumnya digunakan untuk merealisasikan *Error Message*.

Jika dengan penggunaan predikat CUT, Prolog memotong pohon eksekusi sehingga cabang yang tidak menghasilkan solusi tidak perlu dijalani/dicoba, dengan cara menghilangkan proses backtrack ke cabang tersebut. Sebaliknya dengan FAIL, Prolog justru memaksa proses backtrack ke suatu cabang tertentu.
Tetapi, dengan kombinasi CUT yang dilanjutkan dengan FAIL, Prolog dapat memaksa proses backtrack ke suatu cabang tertentu untuk selanjutnya berhenti di cabang tersebut (tidak melanjutkan proses backtrack ke cabang lain).

Efek Penggunaan Predikat FAIL

Efek penggunaan predikat FAIL dijelaskan dengan contoh sbb:

0. Z :- A.
1. A :- B, fail.
2. A :- C.

Jika predikat Z dieksekusi, maka selanjutnya akan memproses predikat A.
Pada saat A dieksekusi dengan menelusuri kalimat (1), jika B bernilai true maka (1) akan digagalkan (bernilai false). Selanjutnya Prolog akan menelusuri kalimat (2), dan hasilnya tergantung pada C.

Efek Penggunaan Predikat CUT-FAIL

Efek penggunaan predikat CUT dilanjutkan dengan FAIL dijelaskan dengan contoh sbb:

0. Z :- A.
1. A :- B, !, fail, ErrorMsg.
2. A :- C.
3. ErrorMsg :- writeln(“Error Message”).

Jika predikat Z dieksekusi, maka selanjutnya akan memproses predikat A.
Pada saat A dieksekusi dengan menelusuri kalimat (1), jika B bernilai true maka (1) akan digagalkan (bernilai false). Selanjutnya Prolog akan menelusuri (3), tetapi tanpa menelusuri kalimat (2).