BAB IV
STRUKTUR PROGRAM PROLOG

Dalam buku ini digunakan program Turbo Prolog untuk melengkapi pembahasan pemrograman logika dengan Prolog. Turbo prolog mirip dengan Turbo Pascal, Turbo C, dan sejenisnya. Tampilannya terbagi menjadi empat jendela yaitu *editor*, *dialog*, *message*, dan *trace*. Jendela editor sebagai tempat untuk menyunting program, sedangkan dialog tempat menuliskan goal dan menampilkan hasil query. Pesan-pesan error, compiler, running, dan lain-lain ditampilkan di dalam jendela message, sedangkan trace sebagai tempat untuk menelusuri jalannya program.

Secara umum, suatu program Prolog terdiri dari beberapa kelompok, yaitu domains, predicates, clauses, goal. Masing-masing bagian akan diuraikan pada bagian berikut.

DOMAINS

Domain dalam Prolog seperti type dalam Pascal, yaitu untuk menyatakan jenis variabel atau argumen, misalnya:

```
domains
    kota = symbol
    alamat = string
    list = symbol*
```

Ada lima domain baku di dalam Prolog, yaitu:

1. **char**, karakter tunggal yang diapit oleh tanda kutip tunggal: ‘a’, ‘b’, ‘\13’.
2. **integer**, bilangan bulat antara –32768 hingga 32767. Notasi $ digunakan untuk menunjukkan bilangan heksa.
3. **real**, bilangan nyata antara 1x10 –307 hingga 1 x 10 308.
4. **string**, deretan karakter yang diapit oleh tanda kutip dobel, misalnya “ipb”.
5. **symbol**, rangkaian karakter yang diawali dengan huruf kecil da tanpa tanda apa pun.

Disamping itu terdapat domain lainnya yang tidak baku, di antaranya adalah:

1. domain **file**, yang digunakan untuk memberi nama file secara simbolik seperti contoh berikut:

   ```
   file = <nama file simbolik 1> ; <nama file simbolik 2> ; …..
   ```

2. domain list, digunakan untuk menyatakan list (linked list) dimana elemen pertama mempunyai pointer ke elemen kedua dan seterusnya. Deklarasi list ini dapat dituliskan dengan bentuk:

   ```
   <nama list> = <domain>*
   list_simbol = symbol*
   ```

3. domain **majemuk**, untuk menyatakan data majemuk, seperti:

```
alamat("Jl. Pajajaran", "Bogor")
```

Pada contoh ini, alamat adalah nama obyek dan disebut sebagai fungtor, dan bagian yang ditulis dalam tanda kurung disebut argumen. Domain majemuk juga dapat digunakan untuk menyatakan beebrapa kemungkinan nilai yang masing-masing dipisahkan oleh tanda titik koma (;) seperti contoh berikut:

```
Tombol = up; down; left; right; karakter(char)
```

PREDICATES

Bagian ini untuk menuliskan setiap relasi predikat yang digunakan dalam program, kecuali predikat baku seperti *cursor*, *makewindow*, *readln*, *readchar*, dan sejenisnya tidak perlu didefinisikan. Lihat contoh-contoh program contoh pada bagian selanjutnya dari bab ini.

CLAUSES

Sekumpulan klausa dari predikat yang sama harus dikelompokkan dalam bagian ini. Dalam melakukan pemanggilan klausa, Prolog melacaknya berurutan dari atas ke bawah. Bagian ini merupakan inti dari program Prolog, dimana semua fakta dan aturan diimplementasikan di sini.

Berikut disajikan beberapa contoh lengkap program dalam Turbo Prolog 2.0.

Contoh 1. Program untuk menyatakan hubungan kakek, ayah, dan ibu.

```
predicates
    grandfather(symbol,symbol)
    father(symbol,symbol)
    mother(symbol,symbol)

clauses
    grandfather(X,Z):-father(X,Y),father(Y,Z).
    grandfather(X,Z):-father(X,Y),motehr(Y,Z).
    father(john,bill).
    father(bill,mary).
    father(bill,tom).
    father(tom,chris).
    father(tom,bob).
    mother(mary,june).
    mother(mary,katie).

goal
    clearwindow, father(Bapak,chris), write(Bapak),
    grandfather(Kakek,chris), write(Kakek).
```

Contoh 2.

```
predicates
        ukuran(symbol, symbol)
        warna(symbol,symbol)
        gelap(symbol)

clauses
        ukuran(beruang, besar).
        ukuran(gajah, besar).
        ukuran(kucing, kecil).
        warna(beruang, coklat).
        warna(kucing, hitam).
        warna(gajah, kelabu).
        gelap(Z):-warna(Z, hitam).
        gelap(Z):-warna(Z,coklat).

goal
        clearwindow,
        gelap(Z), ukuran(Z,besar), write(Z).
```

Contoh 3.

```
domains
        list = symbol*

predicates
        append(list, list, list)

clauses
        append([],Y,Y).
        append([H|X1],Y,[H|Z1]):-append(X1,Y,Z1).

goal
        clearwindow,
        append([a,b],[c,d],Z), write(Z),
        append(X,[c,d],[a,b,c,d]), write(X),
        append([a,b],Y,[a,b,c,d]), wrtie(Y).
```

UNIFIKASI

Unifikasi adalah *instance* suatu term T yang diperoleh dengan menggantikan sub-term dari variabel-variabel T. Dengan kata lain, unifikasi merupakan proses pemadanan atau pembandingan untuk mencari jawaban seperti nilai suatu variabel. Melalui unifikasi, suatu variabel diberi nilai sehingga akan diperoleh jawaban dari suatu pertanyaan (goal). Jika Prolog mendapat pertanyaan maka Prolog akan mencari padanan goal dari bagian klausa paling atas. Bila sudah diperoleh klausa yang dicari, terjadilan pengikatan variabel bebas (jika ada) sehingga pertanyaan dan klausa menjadi identik, dan pertanyaan tersebut dikatakan menyatu dengan klausa. Perhatikan ilustrasi berikut:

goal = f(X,b) = f(a,Y).
X=a, Y=b

f(a,b) adalah *instance* dari f(a,Y),f(X,b), f(X,Y)
g(a,b) bukan *instance* dari g(X,X)

Unifikasi terjadi secara implisit yaitu pada saat suatu aturan dijalankan seperti terlihat pada contoh berikut:

fakta = identity(Z,Z)
goal = identity(f(X,b),f(a,Y))
X = a, Y = b

LACAK BALIK (BACK-TRACK)

Dalam memecahkan persoalan, sering dijumpai proses kegagalan sehingga dilakukan kembali dengan menggunakan jalan atau metode yang lain. Hal yang sama juga dilakukan oleh Prolog dalam menjawab suatu pertanyaan. Permasalahan ini akan ditunjukkan pada bagian berikut dari bab ini.

STRUKTUR DATA (LIST)

Salah satu bentuk data yang populer dalam pemrograman adalah struktur data list dimana dalam Prolog dituliskan dengan menggunakan tanda kurung [] dan setiap elemen dipisahkan oleh tanda koma(,). Sebagai contoh:

[]	list kosong
[a, b, c]	list dengan tiga elemen

Setiap list dalam Prolog dapat dituliskan sebagai **[H | T]** dimana H adalah kepala (head) yang menunjukkan elemen pertama dari suatu list dan T adalah ekor (tail), yaitu list tanpa elemen pertama. Nilai H dan T ini dapat dibandingkan dengan operasi **car** dan **cdr** pada pemrograman fungsional. Oleh karena itu, list [a, b, c] dapat dituliskan sebagai:

[a,b,c|[]]
[a,b|[c]]
[a|[b,c]]

Contoh:

goal = [H | T] = [a,b,c].
YES. H = a, T = [b,c]

Beriktu disajikan dua contoh pemrograman Prolog yang berkaitan dengan struktur data list.

Contoh 1. Menggabungkan dua buah list.

```
clauses
    append([],Y,Y).
    append([H|X],Y,[H|Z]):-append(X,Y,Z).

goal = append ([a,b],[c,d],Z).
YES.  Z = [a,b,c,d]

goal = append ([a,b],Y,[a,b,c,d]).
YES.  Y = [c,d]
```

Contoh 2. Memeriksa keanggotaan dalam list.

```
clauses
    member(X,[X|_]).
    member(X,[_|Y]):-member(X,Y).

goal = member(a,[p,q,a]).
YES.
```

Open List

Open List adalah list yang mempunyai elemen berupa variabel, misalnya [a,b|X]. Bila X diunifikasi dengan [] menjadi [a,b], dan bila diunifikasi dengan [c] menjadi [a,b,c].

Place Holder

Place holders adalah nama variabel sementara yang digunakan kompiler, biasanya sebagai pengganti open list. Perhatikan contoh berikut:

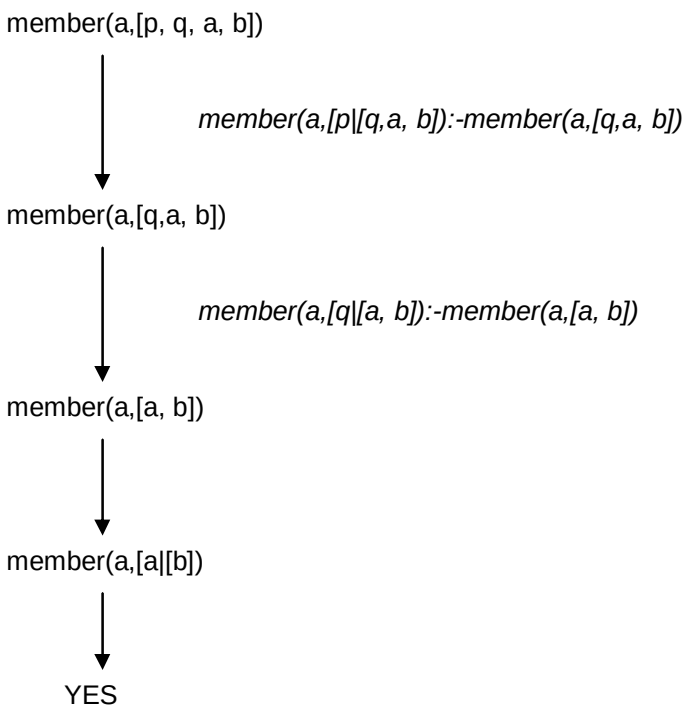
```
goal = append (X,[c],Z).
X=[_1],
Z=[_1,c]
X=[_1,_2],Z=[_1,_2,c]
dan seterusnya
```

SEARCH TREE

Search tree (sering juga disebut sebagai ***Prolog Search Tree***) adalah tree yang tersusun pada saat aturan diterapkan berdasarkan pertanyaan (goal). Contoh search tree berdasarkan relasi member untuk pertanyaan:

```
goal = member(a,[p, q,a])
```

adalah sebagai berikut:



Dalam mengevaluasi pertanyaan, kontrol di dalam Prolog ditentukan oleh dua hal, yaitu urutan pertanyaan dan urutan fakta maupun aturan. Untuk urutan pertanyaan, Prolog menggunakan algoritme evaluasi sebagai berikut:

start with a query as the current goal;
while the current goal is nonempty do
    let the current goal be G_1, G_2, …, G_k, where $k \geq 1$;
    choose the leftmost subgoal G_1;
    if a rule applies to G_1 then
        select the first such rule A :- B_1, B_2, …, B_j, where $j \geq 0$;
        let σ be the most general unifier of G_1 and A;
        the current goal becomes $B_1\sigma$, $B_2\sigma$, …, $B_j\sigma$, $G_2\sigma$, $G_3\sigma$, …, $G_k\sigma$
        else
        backtrack
    end if
end while;
succeed

Sebagai contoh, perhatikan aturan berikut:

```
append ([],Y,Y).
append ([H|X],Y,[H|Z]):-append(X,Y,Z).
prefix(X,Z):-append(X,Y,Z).
suffix(Y,Z):-append(X,Y,Z).
appen2([h|k],Y,[H|Z]):-appen2(X,Y,Z).
appen2([],Y,Y).
```

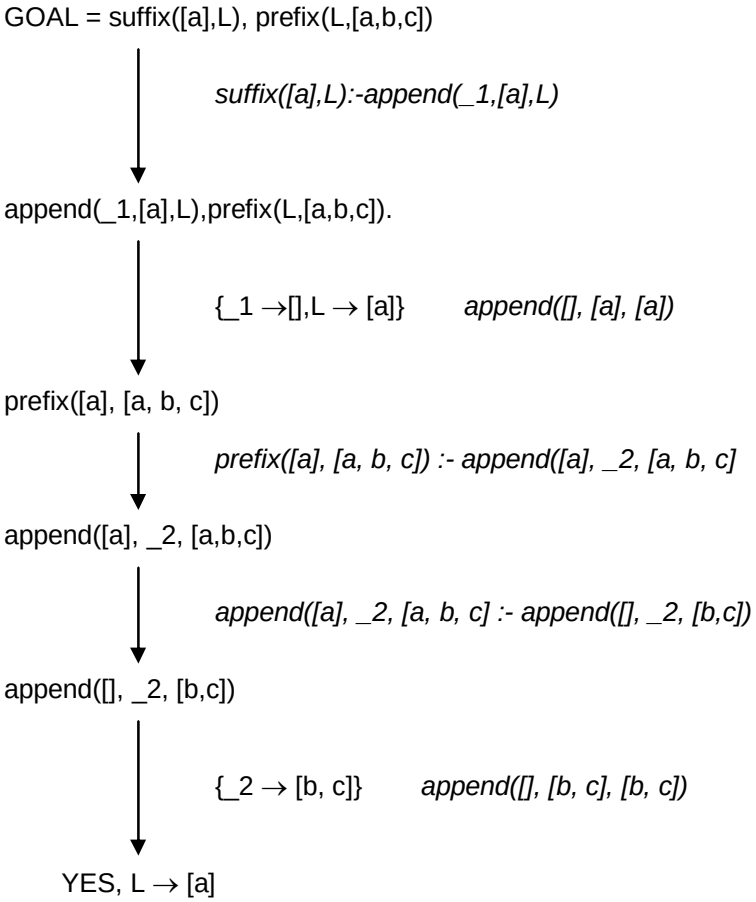
1. Letak subgoal mempengaruhi hasil seperti pada contoh berikut:

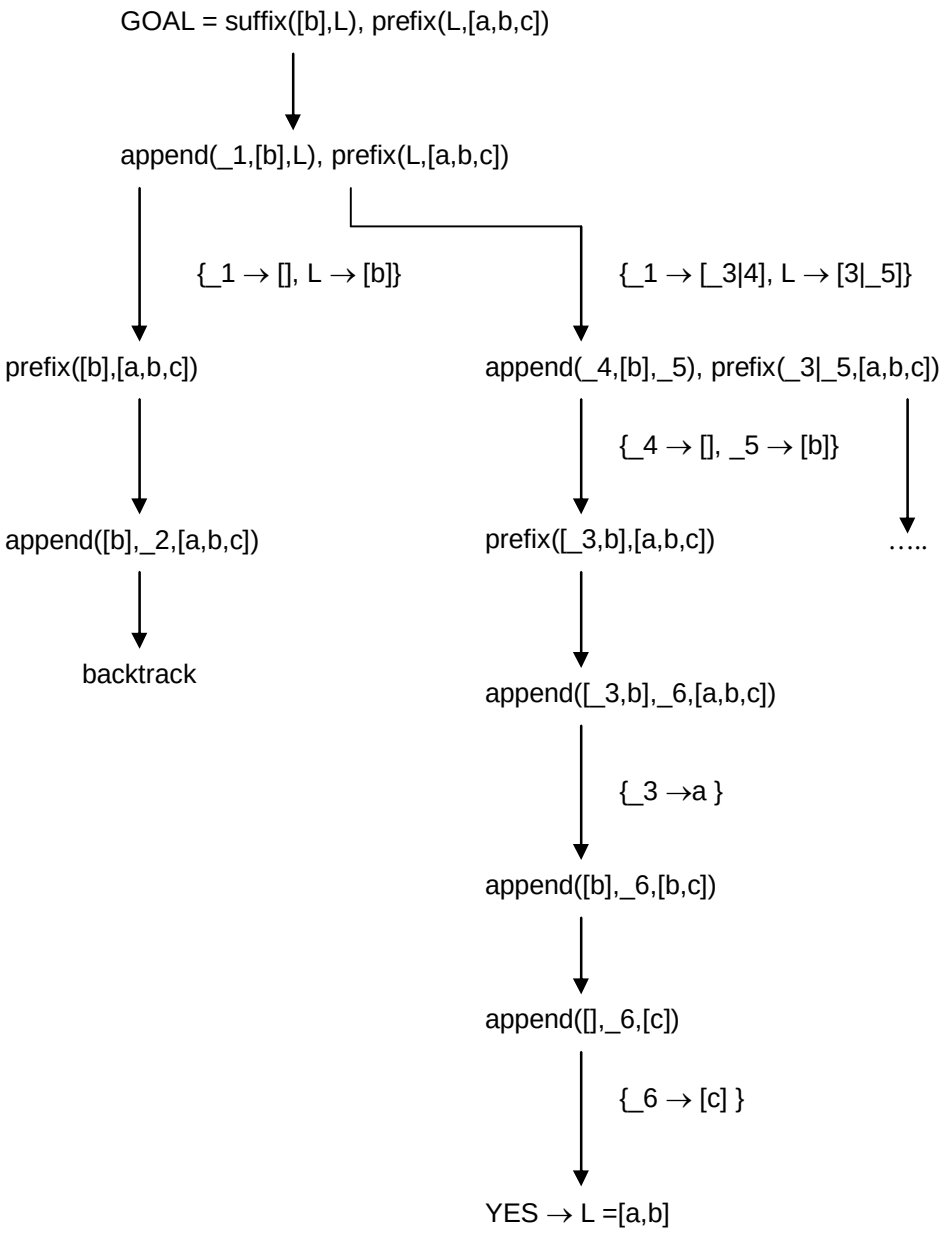
```
goal = prefix(X,[a,b,c]), suffix([e],X).
no solutions
goal = suffix([e],X), prefix(X,[a,b,c]).
trail overflow ---> infinite computation
```

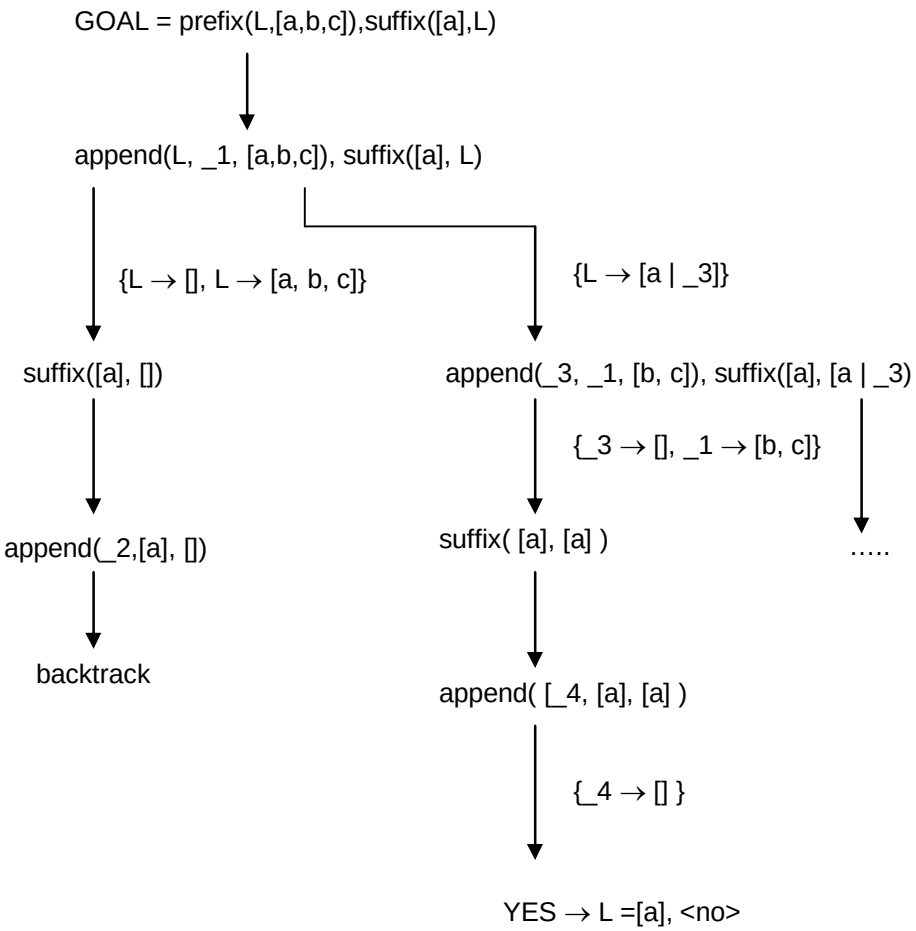
2. Letak rule mempengaruhi hasil

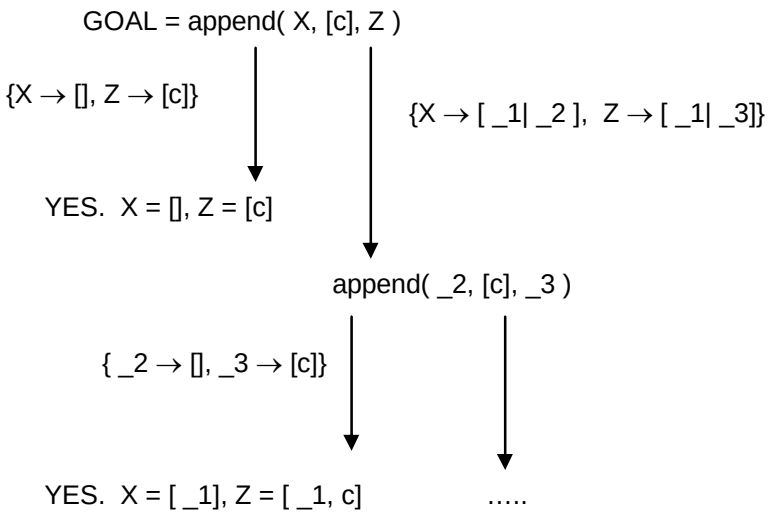
```
goal = append(X,[c],Z).
X=[], Z=[c]
X=[_1], Z=[_1,c]
dst.

goal = appen2(X,[c],Z).
<infinite computation>
```

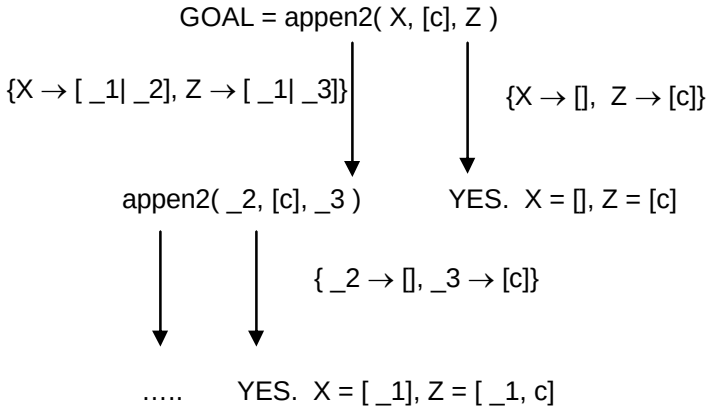
Contoh Prolog search tree untuk kasus ini adalah:




GOAL = suffix([b],L), prefix(L,[a,b,c])
append(_1,[b],L), prefix(L,[a,b,c])
{_1 → [], L → [b]}
{_1 → [_3|4], L → [3|_5]}
prefix([b],[a,b,c])
append(_4,[b],_5), prefix(_3|_5,[a,b,c])
{_4 → [], _5 → [b]}
append([b],_2,[a,b,c])
prefix([_3,b],[a,b,c])
…..
backtrack
append([_3,b],_6,[a,b,c])
{_3 →a }
append([b],_6,[b,c])
append([],_6,[c])
{_6 → [c] }
YES → L =[a,b]


GOAL = prefix(L,[a,b,c]),suffix([a],L)
append(L, _1, [a,b,c]), suffix([a], L)
{L → [], L → [a, b, c]}
{L → [a | _3]}
suffix([a], [])
append(_3, _1, [b, c]), suffix([a], [a | _3)
{_3 → [], _1 → [b, c]}
append(_2,[a], [])
suffix([a], [a])
.....
backtrack
append([_4, [a], [a])
{_4 → [] }
YES → L =[a], <no>





CUT

Predikat CUT digunakan untuk memotong jejak query untuk melakukan lacak balik (backtrack), dan dilambangkan dengan tanda seru (!). Bentuk umum adalah:

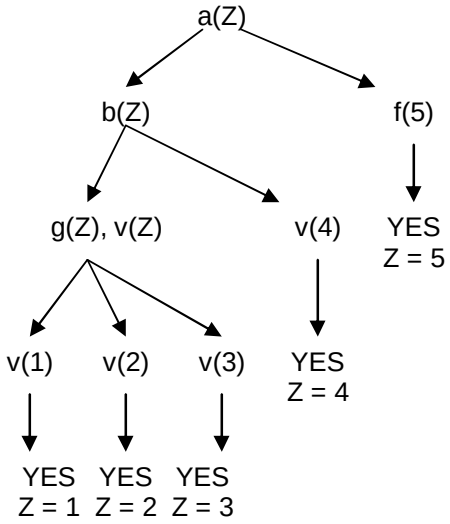
B :- C_1, C_2, ...,C_{j-1}, !, C_j, C_{j+1}, ..., C_n

Proses query karena cut adalah jawaban yang memenuhi sampai dengan Cj-1 di-*freeze* (tak ada backtrack lagi). Backtrack hanya boleh pada C_j,C_{j+1},...,C_n aturan lain dari B tidak diperhatikan lagi.

Ada dua jenis CUT, yaitu ***green* CUT** dimana predikat ini tidak mempengaruhi logika program melainkan hanya untuk peningkatan efisiensi atau kecepatan proses, dan ***red* CUT** yang secara logika memang diperlukan opeh aturan yang dibuat agar dapat digunakan untuk mencegah perhatian Prolog terhadap alternatif sub-goal yang ada.

Berikut disajikan contoh pengaruh predikat CUT terhadap proses evaluasi dari suatu pertanyaan yang diberikan.

```
clauses
    a(X) :- b(X).
    a(X) :- f(X).
    b(X) :- g(X), v(X).
    b(X) :- X=4, v(X).
    g(1).
    g(2).
    g(3).
    v(X).
    f(5).
goals
    a(Z).
```




```
clauses
    a(X) :- b(X).
    a(X) :- f(X).
    b(X) :- g(X), !, v(X).
    b(X) :- X=4, v(X).
    g(1).
    g(2).
    g(3).
    v(X).
    f(5).
goals
    a(Z).
```