Penggunaan Logika dan Pohon dalam Proses Resolusi Query dalam Bahasa Prolog

Akhmal Iswara Adjie (13517127)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13517127@std.stei.itb.ac.id

***Abstrak*—Prolog merupakan salah satu bahasa pemrograman yang menggunakan paradigma deklaratif. Berbeda dengan kebanyakan bahasa pemrograman populer lainnya, Prolog tidak menggunakan algoritma dalam pencarian solusinya. Namun, Prolog menggunakan fakta-fakta dan aturan-aturan untuk mencari solusi dengan teknik heuristik memanfaatkan pohon logika. Dalam makalah ini akan dibahas mengenai penggunaan pohon logika tersebut untuk menyelesaikan persoalan.**

***Kata Kunci*—Prolog, logika, pemrograman logika, pohon**

I. PENDAHULUAN

Saat ini terdapat banyak sekali bahasa pemrograman. Setiap bahasa pemrograman memiliki cara kerja, fitur dan fungsinya masing-masing, yang diciptakan sesuai dengan kebutuhan programmernya. Bahasa pemrograman yang ada saat ini dikelompokkan berdasarkan cara pendekatan suatu bahasa untuk menyelesaikan permasalahan.

Terdapat banyak sekali paradigma pemrograman, namun beberapa diantaranya sering digunakan. Diantaranya paradigma prosedural (imperatif), dan deklaratif. Meskipun kebanyakan bahasa yang dipakai saat ini (misalnya C++, C, Python) menggunakan paradigma prosedural, ternyata dalam beberapa persoalan lebih mudah diselesaikan dengan paradigma lain, diantaranya paradigma deklaratif.

Paradigma deklaratif merupakan pendekatan bahasa pemrograman untuk menyelesaikan masalah hanya dengan memberi sekumpulan data dan apa yang harus dilakukan program dan tidak memberi instruksi bagaimana langkah-langkah penyelesaian permasalahan tersebut.

Pemrograman logika merupakan salah satu dari sub-paradigma deklaratif. Setiap program yang ditulis dalam bahasa yang menggunakan paradigma tersebut merupakan bentuk logika. Program tersebut menyelesaikan persoalan dengan memanfaatkan fakta-fakta yang ada dan menggunakan aturan inferensi.

Salah satu bahasa pemrograman yang menggunakan paradigma pemrograman logika adalah Prolog. Prolog sendiri memiliki beberapa versi, salah satunya adalah GNU Prolog, yang akan penulis gunakan dalam pembahasan ini.

Hal yang menarik dari pemrograman logika adalah cara kerja programnya yang tidak menggunakan perintah sekuensial seperti kebanyakan bahasa pemrograman lainnya. Pemrograman logika justru memanfaatkan kaidah logika formal dan pohon pencarian solusi.

Dengan memahami cara kerja bahasa pemrograman dengan paradigma pemrograman logika, tentunya akan membantu dalam menyelesaikan masalah, khususnya Artificial Intelligence (AI) dan expert system dengan menggunakan paradigma pemrograman tersebut.

II. DASAR TEORI

2.1. Bahasa Pemrograman Prolog

Prolog merupakan bahasa pemrograman logika yang merupakan cabang dari paradigma pemrograman deklaratif. Prolog pertama kali dikembangkan oleh Alain Colmerauer dan P.Roussel di Universitas Marseilles, Perancis, pada tahun 1972. Prolog merupakan singkatan dari "Programming in Logic". Prolog dibuat berdasarkan Kalkulus Predikat Orde Pertama.

Struktur Prolog terdiri dari fakta, aturan dan query. Fakta dan aturan merupakan klausa yang terdapat pada program Prolog tersebut. Sedangkan query terdapat di luar program.

Klausa adalah isi dari pengetahuan dasar yang dimiliki oleh prolog.

Fakta merupakan merupakan *knowledge base* (pengetahuan dasar) yang dianggap benar oleh program. Fakta seringkali ditulis dengan argumen yang merupakan konstanta. Misalnya, ibu(siti, budi).

Aturan (rules) merupakan predikat yang akan memberi tahu bagaimana cara mendeduksi fakta baru yang tidak tertulis secara eksplisit pada bagian fakta. Aturan ini juga merupakan pengetahuan bagi program untuk menurunkan fakta-fakta lain yang tidak disebutkan secara eksplisit dalam fakta. Aturan juga dapat memiliki argumen. Misalnya, ibu(X, Y) :- orangtua(X, Y), wanita(X).

Sedangkan query adalah pertanyaan yang ingin dicari jawabannya melalui program yang telah dibuat dalam bahasa Prolog. Pertanyaan tersebut juga dalam bentuk predikat yang dapat dimengerti oleh program..

2.2. Dasar Logika

Logika merupakan ilmu tentang cara berpikir dan menalar (*reasoning*). Logika didasarkan pada hubungan antar pernyataan (*statement*).

Pernyataan yang dibahas dalam logika hanya pernyataan yang memiliki nilai benar (*true*) atau salah (*false*), tetapi tidak

keduanya. Pernyataaan tersebut dinamakan proposisi. Sedangkan kebenaran atau kesalahan suatu proposisi disebut nilai kebenaran.

Contoh proposisi adalah :

- 3 + 2 = 5 (Benar)
- Bandung adalah Ibukota Jawa Timur (Salah)

Bukan contoh proposisi :

- x + 3 = 8

2.2.1. Bentuk-Bentuk Preposisi dan Tabel Kebenaran

Preposisi dapat dinyatakan dalam 4 bentuk, yaitu

1. Preposisi atomik
   Preposisi atomik merupakan preposisi tunggal, yang tidak dapat dibagi lagi menjadi beberapa preposisi.
   Contoh : (2n + 1) selalu ganjil untuk setiap n>0
2. Preposisi majemuk
   Preposisi majemuk adalah kombinasi dari 2 atau lebih proposisi. Proposisi majemuk dibagi menjadi 3 macam, yaitu :
   a. Konjungsi (notasi : p ∧ q)
   Merupakan proposisi "p dan q". Tabel kebenarannya adalah sebagai berikut.

p	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

Tabel 1. Tabel kebenaran p ∧ q

Contoh : Hari ini hujan dan kuliah diliburkan.

b. Disjungsi (notasi : p ∨ q)
Merupakan proposisi "p atau q". Tabel kebenarannya adalah sebagai berikut.

p	q	p ∨ q
T	T	T
T	F	T
F	T	T
F	F	F

Tabel 2. Tabel kebenaran p ∨ q

Contoh : Lucky memakai payung atau hari cerah.

c. Negasi (notasi : ~p)
Merupakan proposisi "tidak p". Tabel kebenarannya adalah sebagai berikut.

p	~p
T	F
F	T

Tabel 3. Tabel kebenaran ~p

Contoh : Tidak benar bahwa hari ini hujan.

d. Disjungsi eksklusif (notasi p ⊕ q)
Merupakan proposisi "p atau q tetapi tidak keduanya". Tabel kebenarannya adalah sebagai berikut.

p	q	p ∨ q
T	T	F
T	F	T
F	T	T
F	F	F

Tabel 4. Tabel kebenaran p ⊕ q

3. Preposisi implikasi (Notasi : p → q)
   Merupakan proposisi "jika p maka q". Proposisi tersebut dapat bermakna "jika p, q", "q jika p", "p mengakibatkan q", "p syarat cukup agar q", "q syarat perlu bagi p". Tabel kebenarannya adalah sebagai berikut.

p	Q	p → q
T	T	T
T	F	F
F	T	T
F	F	T

Tabel 5. Tabel kebenaran p → q

Contoh : Jika hari ini hujan, maka Fata memakai payung.

4. Preposisi biimplikasi (Notasi : p ↔ q)
   Merupakan proposisi "p jika dan hanya jika q". Tabel kebenarannya adalah sebagai berikut.

p	Q	p ↔ q
T	T	T
T	F	F
F	T	F
F	F	T

Tabel 5. Tabel kebenaran p ↔ q

Contoh : Evan mendapat indeks 4 jika dan hanya jika Evan mendapatkan nilai A di semua mata kuliahnya.

2.3. Pohon (Tree)

Pohon merupakan bentuk khusus dari graf, yakni graf tak-berarah yang tidak memiliki sirkuit. Sirkuit merupakan lintasan graf yang berawal dan berakhir di simpul yang sama. Pohon memiliki karakteristik memiliki n simpul dan n-1 sisi.

Pohon berakar yaitu pohon yang salah satu simpulnya memiliki akar, sementara sisi-sisinya diberi arah sehingga menjadi graf berarah.

2.3.1 Terminologi Pohon Berakar

Terdapat beberapa terminologi dalam pohon berakar, diantaranya adalah:

1. Anak (*child*) dan orang tua (*parent*)
   Suatu simpul yang menghasilkan simpul lainnya disebut orang tua, sementara simpul yang dihasilkannya disebut anak.
2. Lintasan
   Lintasan adalah urutan simpul-simpul yang dilewati dari simpul awal ke lintasan akhir. Panjang lintasan adalah jumlah sisi yang dilewati dalam satu lintasan.
3. Saudara kandung (*sibling*)
   Saudara kandung adalah simpul yang memiliki orang tua yang sama.
4. Upapohon (*subtree*)
   Upapohon adalah pohon yang dibentuk dari pohon utama.
5. Derajat
   Derajat adalah jumlah upapohon atau jumlah anak pada simpul tersebut. Yang dimaksud dengan derajat dalam

tree adalah derajat-keluar pada terminologi graf.

6. Daun
   Daun adalah simpul yang tidak memiliki anak.
7. Simpul dalam
   Simpul dalam merupakan simpul yang memiliki anak.
8. Aras (*level*)
   Aras adalah panjang lintasan dari akar menuju suatu simpul.
9. Tinggi atau kedalaman
   Tinggi atau kedalaman adalah aras maksimum dari suatu pohon.

III. TRANSLASI SINTAKS PROLOG MENJADI BENTUK LOGIKA

Sebelum memahami cara kerja Prolog sebagai bahasa pemrograman paradigma deklaratif, diperlukan pemahaman mengenai sintaks dari Prolog itu sendiri. Berikut ini adalah contoh program yang ditulis menggunakan bahasa Prolog.

```
makhluk(kucing).
makhluk(ikan).
makhluk(cacing).
makhluk(mawar).

kingdom(vertebrata, animalia).
kingdom(invertebrata, animalia).

adalah(kucing, vertebrata).
adalah(ikan, vertebrata).
adalah(cacing, invertebrata).
adalah(mawar, spermatopytha).

besar(kucing).
besar(ikan).
besar(mawar).

hewan(X) :- makhluk(X), adalah(X, Y), kingdom(Y,
            animalia).
hewanBesar(X) :- besar(X), hewan(X).
```

Contoh kode prolog 1. Sumber : penulis

Predikat makhluk(kucing) bermakna bahwa kucing adalah makhluk. Predikat `kingdom(vertebrata, hewan).` bermakna vertebrata memiliki kingdom animalia. Predikat `kingdom(invertebrata, animalia).` bermakna bahwa invertebrata memiliki kingdom animalia. Begitu pula dengan `adalah(X, Y).` yang bermakna bahwa X adalah Y.

Sementara predikat `hewan(X) :- makhluk(X), adalah(X, Y), kingdom(Y, animalia).` memiliki makna "X adalah hewan jika ada Y, sehingga X adalah Y dan Y memiliki kingdom animalia".

Sehingga dapat disimpulkan bahwa fakta merupakan proposisi tunggal yang dianggap benar oleh program. Tanda :- diartikan sebagai "jika". Tanda koma diartikan sebagai "dan". Apabila ingin menulis "atau" dapat dilakukan dengan menulis ulang proposisi syarat perlu, diikuti syarat cukup yang berbeda. Misalnya, apabila ingin menuliskan predikat "X adalah hewan jika X adalah vertebrata atau X adalah invertebrata", dapat dituliskan sebagai berikut.

```
hewan(X) :- adalah(X, vertebrata).
hewan(X) :- adalah(X, invertebrata).
```

Suatu predikat dapat dinegasikan dengan cara memberi tanda "\+" didepan predikatnya.

Prolog mendukung penggunaan cut (!) dan fail untuk mencegah atau justru memaksa untuk melakukan backtracking. Penjelasan mengenai backtraking akan dijelaskan pada bagian keempat.

IV. PROSES RESOLUSI QUERY DALAM PROLOG

Telah dijelaskan sebelumnya bahwa query merupakan pertanyaan yang ingin dicari kebenarannya. Query tidak hanya berupa predikat yang argumennya konstanta, tetapi boloh juga predikat yang memiliki argumen campuran antara konstanta dengan variabel, atau keseluruhannya merupakan variabel. Query juga boleh merupakan proposisi majemuk.

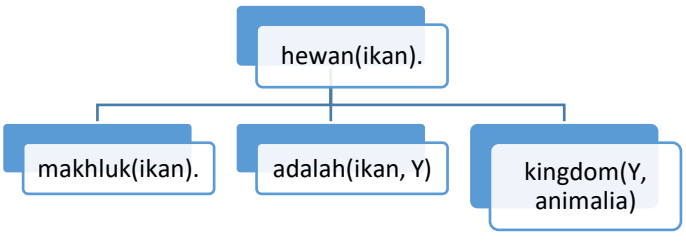
Pada contoh 1 diatas, misalkan program diberikan query `?- hewan(ikan).` program akan menampilkan jawaban "yes". Namun, ketika program diberikan query ?- hewan(mawar). atau ?- Hewan(zebra) program akan menampilkan jawaban "no".

Bila kita *trace* pada saat pencarian jawaban dari query `?- hewan(ikan).` akan muncul pencarian sebagai berikut.

```
| ?- hewan(ikan).
      1    1  Call: hewan(ikan) ?
      2    2  Call: makhluk(ikan) ?
      2    2  Exit: makhluk(ikan) ?
      3    2  Call: adalah(ikan,_128) ?
      3    2  Exit: adalah(ikan,vertebrata) ?
      4    2  Call: kingdom(vertebrata,animalia) ?
      4    2  Exit: kingdom(vertebrata,animalia) ?
      1    1  Exit: hewan(ikan) ?
```

Gambar 1. Hasil *trace* query hewan(ikan).
Sumber : penulis

Pertama, program akan mengecek rules dari hewan(X), dengan mengganti X dengan ikan. Program harus mengecek makhluk(ikan), adalah(ikan, Y), lalu mengecek kingdom(Y, animalia). Untuk menghasilkan jawaban yes, ketiga predikat harus bernilai true. Setelah mengecek makhluk(ikan) di baris 2 dan 3, program mengecek adalah(ikan, Y). Pada gambar diatas, program mengganti Y dengan _128 yang juga bermakna variabel juga. Kemudian program menemukan _128 cocok apabila diganti dengan vertebrata berdasarkan fakta pada baris ke 10 dari kode tersebut (Y di-*assign* dengan vertebrata). Kemudian barulah program mengecek kingdom (vertebrata, animalia) dan akhirnya menghasilkan true. Proses pencarian ini dapat digambarkan dengan pohon berakar dengan akar dari pohon tersebut adalah hewan(ikan).



Gambar 2. Pohon Pencarian hewan(ikan).
Sumber : Penulis

Pohon tersebut memiliki kedalaman 1, dan memiliki akar hewan(ikan) yang merupakan query. Sementara daun-daunnya adalah predikat yang berada di sebelah kanan tanda :- pada bagian aturan hewan(X).

Apabila dilakukan *trace* pada saat pencarian jawaban dari query `?- hewan(mawar).` hasilnya akan menjadi sebagai berikut.

```
| ?- hewan(mawar).
      1    1  Call: hewan(mawar) ?
      2    2  Call: makhluk(mawar) ?
      2    2  Exit: makhluk(mawar) ?
      3    2  Call: adalah(mawar,_128) ?
      3    2  Exit: adalah(mawar,spermatopytha) ?
      4    2  Call: kingdom(spermatopytha,animalia) ?
      4    2  Fail: kingdom(spermatopytha,animalia) ?
      1    1  Fail: hewan(mawar) ?

(15 ms) no
```

Gambar 3. Hasil *trace* query hewan(mawar)
Sumber : penulis

Sama seperti pencarian query sebelumnya, program akan mengecek kebenaran dari query `?- hewan(mawar).` menggunakan pohon di gambar 2. Setelah mengecek predikat makhluk(mawar) dan adalah(mawar, Y), dan keduanya bernilai true, Y bernilai spermatopytha. Selanjutnya program akan mengecek apakah kingdom(spermatopyta, animalia) merupakan fakta. Karena dalam program tidak didefinisikan fakta seperti itu, maka query tersebut false dan mengembalikan "no".

Apabila kita memanggil query hewan(meja). atau hewan(zebra), program akan mengembalikan no, karena makhluk(meja) dan makhluk(zebra) tidak didefinisikan dalam fakta.

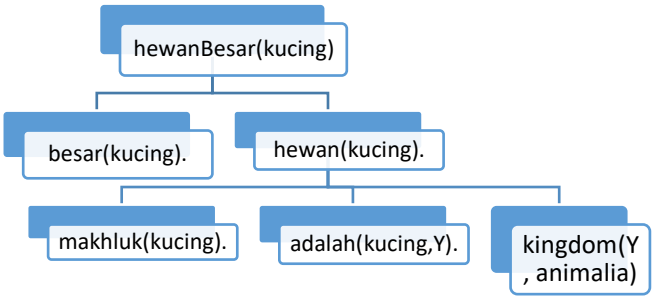
Pohon pencarian akan lebih dalam apabila nilai kebenaran anak dari simpul ternyata didapat dari hasil deduksi aturan lainnya maupun memanggil aturan itu sendiri. Jumlah anak akan semakin banyak apabila banyak predikat setelah tanda ":- " , yang dipisahkan oleh tanda koma.

Misalnya, apabila dilakukan *trace* pada resolusi query hewanBesar(kucing), hasilnya akan sebagai berikut.

```
| ?- hewanBesar(kucing).
      1    1  Call: hewanBesar(kucing) ?
      2    2  Call: besar(kucing) ?
      2    2  Exit: besar(kucing) ?
      3    2  Call: hewan(kucing) ?
      4    3  Call: makhluk(kucing) ?
      4    3  Exit: makhluk(kucing) ?
      5    3  Call: adalah(kucing,_175) ?
      5    3  Exit: adalah(kucing,vertebrata) ?
      6    3  Call: kingdom(vertebrata,animalia) ?
      6    3  Exit: kingdom(vertebrata,animalia) ?
      3    2  Exit: hewan(kucing) ?
      1    1  Exit: hewanBesar(kucing) ?

(31 ms) yes
```

Gambar 4. Hasil *trace* query hewanBesar(kucing)
Sumber : penulis

Proses pencarian tersebut dapat digambarkan dengan pohon sebagai berikut.



Gambar 5. Pohon Pencarian hewanBesar(kucing).
Sumber : Penulis

Pohon resolusi query pada gambar 5 memiliki kedalaman 2, dengan akar sama dengan querynya, yaitu hewanBesar(kucing). Pohon ini memiliki kedalaman yang lebih dalam daripada pohon di gambar 2, karena hewan(kucing) yang merupakan anak dari akar bukanlah fakta yang eksplisit, namun dibutuhkan deduksi lagi untuk menemukan kebenaran dari hewan(kucing).

Seringkali dalam proses pencarian diperlukan *backtracking* (runut-balik). Algoritma runut-balik dalam prolog digunakan, sehingga apabila sebuah predikat tertentu bernilai salah, maka program akan mencari alternatif lain yang bernilai benar. Sehingga dengan algoritma ini proses pencarian akan lebih mangkus, karena tidak perlu mencari seluruh jawaban, hanya pencarian yang mengarah pada satu jawaban. Apabila ingin mencari solusi lain, pengguna dapat memberi perintah agar program melakukan backtracking.

Sebagai contoh, apabila program diminta untuk melakukan resolusi query hewanBesar(X), hasil potongan *trace*-nya adalah sebagai berikut.

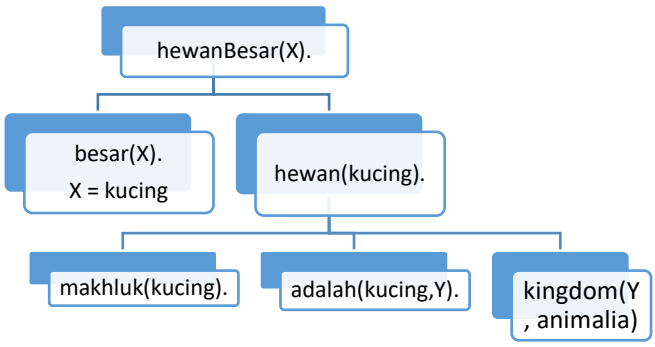
```
| ?- hewanBesar(X).
      1    1  Call: hewanBesar(_42) ?
      2    2  Call: besar(_42) ?
      2    2  Exit: besar(kucing) ?
      3    2  Call: hewan(kucing) ?
      4    3  Call: makhluk(kucing) ?
      4    3  Exit: makhluk(kucing) ?
      5    3  Call: adalah(kucing,_181) ?
      5    3  Exit: adalah(kucing,vertebrata) ?
      6    3  Call: kingdom(vertebrata,animalia) ?
      6    3  Exit: kingdom(vertebrata,animalia) ?
      3    2  Exit: hewan(kucing) ?
      1    1  Exit: hewanBesar(kucing) ?

X = kucing ? ;
      1    1  Redo: hewanBesar(kucing) ?
      2    2  Redo: besar(kucing) ?
      2    2  Exit: besar(ikan) ?
      3    2  Call: hewan(ikan) ?
      4    3  Call: makhluk(ikan) ?
      4    3  Exit: makhluk(ikan) ?
      5    3  Call: adalah(ikan,_181) ?
      5    3  Exit: adalah(ikan,vertebrata) ?
      6    3  Call: kingdom(vertebrata,animalia) ?
      6    3  Exit: kingdom(vertebrata,animalia) ?
      3    2  Exit: hewan(ikan) ?
      1    1  Exit: hewanBesar(ikan) ?

X = ikan ? ;
```

Gambar 6. Hasil *trace* query hewanBesar(X).
Sumber : Penulis

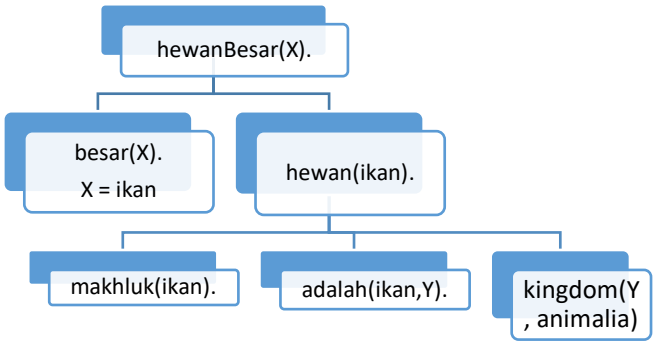
Pada gambar 6 diatas, terlihat bahwa program tidak memberikan keseluruhan jawaban, namun hanya memberikan salah satu jawaban saya, yaitu X = kucing. Tanda semicolon merupakan masukan pengguna untuk mencari jawaban lain, sehingga program melakukan backtracking. Sehingga dicari alternatif jawaban yang lain, yaitu X = ikan. Pengguna juga dapat mencari semua solusi dengan memasukkan huruf a.

Bila melakukan backtracking, program akan mengambil fakta besar(mawar). Namun hewan(mawar) akan mengembalikan false, sehingga hasil backtrackingnya adalah "no".

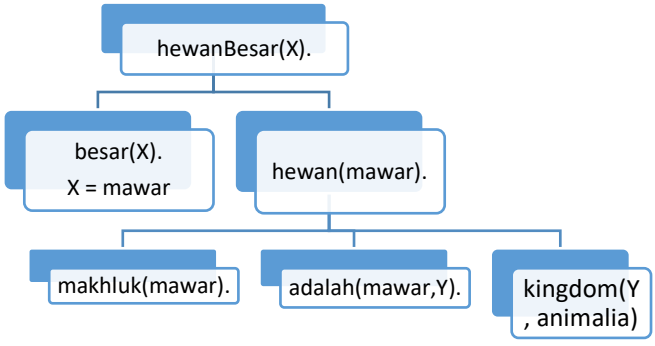
Berikut ini adalah pohon pencarian untuk melakukan proses resolusi query hewanBesar(X).



Gambar 7. Pohon Pencarian hewanBesar(X) sebelum backtracking pertama.
Sumber : Penulis



Gambar 8. Pohon Pencarian hewanBesar(X) setelah backtracking pertama.
Sumber : Penulis



Gambar 9. Pohon Pencarian hewanBesar(X) setelah backtracking kedua (mengembalikan "no").
Sumber : Penulis

Dengan backtracking, pencarian tidak perlu dilakukan secara keseluruhan sehingga dapat menghemat waktu. Namun cukup menemukan satu solusi. Apabila solusi dirasa belum cukup, pengguna dapat membuat program untuk melakukan backtracking untuk mencari satu solusi yang lain. Atau dapat membuat program mencari semua kemungkinan solusi. Hal ini akan bermanfaat, terutama apabila solusi dari query memiliki banyak jawaban, bahkan tak terhingga banyaknya dan pengguna hanya mencari salah satu solusi saja.

Seringkali solusi yang diinginkan hanya satu saja. Oleh karena itu, Prolog menyediakan fitur cut (!). Cut berfungsi untuk mencegah program melakukan backtracking, baik ketika menemukan solusi maupun tidak. Cut dapat dimanfaatkan agar waktu eksekusi lebih mangkus, atau dapat digunakan mengubah makna dari suatu program.

Terkadang, program dapat melakukan pencarian di cabang yang sudah pasti menghasilkan false. Agar program menjadi mangkus, digunakan fail untuk menggagalkan proses pencarian di simpul tersebut.

Sementara kombinasi dari cut dan fail dapat berefek menghentikan proses pencarian (stop).

Berikut ini adalah contoh program Prolog yang menggunakan cut dan fail dalam implementasinya.

```
b(1).
b(2).
c(1).
c(3).
d(2).

a(X) :- b(X), c(X), !.
a(X) :- d(X).
```

Contoh kode prolog 2.
Sumber : diadaptasi dari
https://stackoverflow.com/15138706/cut-and-fail-in-prolog

Dengan menggunakan kode Prolog tersebut, apabila program menerima query a(X), program hanya menuliskan X = 1 dan tidak akan backtrack menuju rules yg berada di paling bawah. Hal ini terjadi karena tanda cut (!) mencegah terjadinya backtracking. Berbeda dengan tanpa cut ataupun fail yang dapat melakukan backtrack setelah menemukan solusi X = 1, yaitu X = 2.

Apabila tanda cut diganti fail, maka saat memproses resolusi query a(X), program akan langsung mengirimkan fail, karena saat b(X) dan c(X) bernilai true, akan digagalkan oleh fail. Namun saat memanggil a(2) akan mengembalikan true.

Dapat disimpulkan, penggunaan cut dan fail dapat berefek pada proses backtracking. Sehingga dapat mengubah eksekusi program, bahkan dapat mengubah makna dari program tersebut.

V. KESIMPULAN DAN SARAN

Teori logika dan graf sangat sering dipakai dalam keilmuan informatika. Diantaranya dalam bahasa pemrograman dengan paradigma logika, salah satunya Prolog.

Dalam proses resolusi query dalam prolog memanfaatkan

logika sebagai paradigmanya dan menggunakan pohon untuk melakukan resolusi terhadap query tersebut. Pohon tersebut dapat memiliki kedalaman yang berbeda, jumlah daun yang berbeda tergantung pada fakta dan aturan yang ada dalam kode Prolog tersebut. Pohon pencarian dapat berbeda-beda untuk setiap solusi, namun memiliki akar yang sama, apabila query memiliki lebih dari satu alternatif pencarian solusi.

Pemahaman terhadap pemrograman logika diperlukan dalam memahami expert system, machine learning, natural language processing (NLP), juga artificial intelligence (AI). Oleh karena itu, programmer diharapkan untuk memahami pemrograman logika terlebih dahulu, terutama bagi programmer di bidang tersebut.

VI. UCAPAN TERIMA KASIH

Ucapan terima kasih penulis haturkan kepada Allah SWT atas berkat rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan makalah ini tepat waktu. Penulis juga berterima kasih kepada orang tua yang telah memberi dukungan penuh untuk perkuliahan penulis. Tidak lupa penulis juga berterima kasih kepada Bapak Rinaldi Munir selaku dosen penulis yang telah mencurahkan ilmu, waktu dan tenaganya untuk membimbing penulis dalam mata kuliah Matematika Diskrit sehingga penulis memahami ilmu Matematika Diskrit. Penulis juga berterima kasih kepada teman-teman yang memberikan semangat untuk penulis.

REFERENSI

[1] Maulidevi, Nur Ulfa. 2016. *Introduction to Prolog.* Bandung : Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung

[2] Munir, Rinaldi. 2006. *Matematika Diskrit*. Bandung : Institut Teknologi Bandung

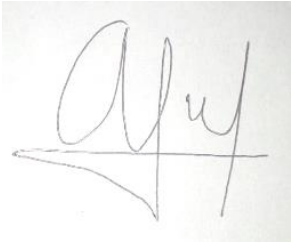
[3] Rahman, Edwin. 2017. *Penggunaan Algoritma Runut Balik dalam Proses Resolusi Query dalam Bahasa Prolog*. Makalah IF2120 Matematika Diskrit.

[4] Striegnitz, Kristina. 2003. *Learn Prolog Now!* . http://cs.union.edu/~striegnk/learn-prolog-now/html/. Diakses pada tanggal 9 Desember 2018.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 10 Desember 2018



Akhmal Iswara Adjie
NIM 13517127