ALGORITMA UNTUK MATCHING PADA SISTEM PENULISAN ULANG EKSPRESI

Slamet Santosa[1], Anton Setiawan Honggowibowo [2]
[1]Staff Peneliti P3TM – BATAN, Jl. Babarsari Po Box 1008 Yogyakarta, Telp. (0274) 488435
[2]Jurusan Teknik Informatika Sekolah Tinggi Teknologi Adisutjipto (STTA), Jl. Janti Blok-R Lanud. Adisutjipto Yogyakarta, Telp. (0274) 451262 Fax. (0274) 451265

Abstract

Matching process in tree is finding subtree in a given tree which to be replaced to variables those occur in pattern tree. It is an important problem that occurs as a crucial operation in functional and equational programming such as Term Rewriting System. We present an algorithm for matching process on such term in tree based on pattern matching. We linearize both given tree and pattern tree into string representation by using Euler technique and and apply prefix-sum to computers the rank of all linearized edge. And then we do matcing on string sequential.

Kata kunci : *term Rewriting System, matching process, tree*

1. PENDAHULUAN

Di dalam teknik pemrograman komputasi logika (*computational logic*) dan pemrograman sistem persamaan (*equational system programming*) banyak dijumpai penyederhanaan ekspresi persamaan dengan berturut-turut mengganti sub-sub ekspresi, misalnya pada spesifikasi tipe data abstrak (*abstract data type specification*) dan atau pada implementasi pemrograman fungsional (*functional programming*). Pada banyak literatur tentang model komputasi Sistem Penulisan Ulang Ekspresi (*Term Rewriting Sytem, TRS* [1][2]) komputasi dengan cara penyederhanaan persamaan ekspresi tersebut adalah suatu gagasan penyederhanaan berdasar pada himpunan (*set*) tetapan (*rules*) berturut-turut hingga dicapai bentuk paling sederhana (*normal form*).

Substitusi σ adalah pemetaan dari ekspresi ke ekspresi yang memenuhi $\sigma(F(t_1,\ldots,t_n)) = F(\sigma(t_1),\ldots, \sigma(t_n))$ untuk setiap simpul fungsi F, t adalah ekspresi atau sub ekspresi dan $n \geq 0$. Contoh sederhana model komputasi TRS dapat dipelajari dengan memperhatikan himpunan tetapan (*set of rules*) sebagai berikut:

$r_1 : A(x,0) \rightarrow x$

$r_2 : A(x,S(y) \rightarrow S(A(x,y))$

$r_3 : M(x,0) \rightarrow 0$

$r_4 : M(x,S(y) \rightarrow A(M(x,y),x)$

(*A=penjumlahan(Addition),M=perkalian(Multiplication)*)

dan diberikan sebuah ekspresi M(S(S(0),S(S(0))); S(0) = 1. Pembaca dapat dengan mudah memeriksa, ekspresi tersebut dapat disederhanakan berturut-turut dengan substitusi σ_i untuk setiap langkah penyederhanaan sehingga M(S(S(0),S(S(0))) ; S(S(S(S(0))))).

Pada hakekatnya model komputasi tersebut adalah operasi penyederhanaan struktur data tree (*tree*) menggunakan sub-sub tree sebagai substitusi, dengan himpunan tetapan adalah pola-pola (*patterns*) tree.

Dalam penelitian ini kami mempelajari teknik *matching* suatu ekspresi yang mana muncul berulang-ulang pada model komputasi TRS. Diberikan dua buah ekspresi, dalam simbul

fungsi-fungsi f_1, $f_{2,...,}$ $f_{n,}$ dan variabel-variabel x_1, $x_{2,...,}$ x_n dan dalam hanya simbul fungsi-fungsi. *Matching* suatu ekspresi adalah memberikan harga-harga substitusi pada variabel-variabel sehingga kedua buah ekspresi ekivalen. Dua buah ekspresi $f_1(f_2(f_3,x_1),x_2)$ dan $f_1(f_2(f_3,a),f_4(f_5(a,b)))$ ekivalen bila dan hanya bila didapatkan substitusi-substitusi $\sigma_1 = a$ dan $\sigma_2 = f_4(f_5(a,b))$ dan masing-masing disubstitusikan sehingga $x_1 = \sigma_1$ dan $x_1 = \sigma_1$. Pada penyederhanaan struktur data tree kedua ekspresi diatas adalah masing-masing pola tree dan obyek tree.

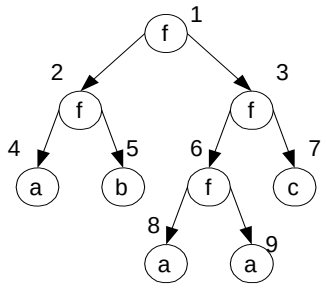
Pada penelitian ini kami berhasil merancang algoritma yang optimal untuk *matching* pola pada obyek tree, yang merupakan bagian penting pada teknik komputasi berbasis tetapan dengan model komputasi TRS. Algoritma tersebut mengadopsi teknik Euler (*Eulerian circuit*) dan menggunakan urutan *string* yang terbentuk untuk melakukan proses *matching*.

2. REPRESENTASI TEKNIK EULER

Eulerian graph adalah merupakan rangkaian terarah (*directed circuit*) yang merupakan hasil jejakan (*traversal*) pada struktur data tree. Diberikan sebuah tree $T = (V, E)$, dengan $v \in V$ adalah himpunan node-node (*nodes*) dan $e \in E$ adalah himpunan edge-edge (*edges*). Rangkaian terarah $T' = (V, E')$ didapat dengan mengganti masing-masing edge (u, v) dengan dua buah busur (*arc*) $<u, v>$ dan $<v, u>$. Secara sekuensial telah diperiksa oleh Jaja [3] bahwa, *Eulerian graph* dapat didefinisikan dengan menyatakan fungsi pengganti (*succesor function*) s dan memetakan tiap-tiap busur $e \in E$ ke $s(e) \in E'$ sehingga e kontinyu pada $T' = (V, E')$.

Untuk mendapatkan fungsi pengganti, pada setiap node $v \in V$ dari tree, adalah dengan menentukan node-node yang bersebelahan dengan v dengan membuat urutan (*ordering*) pada himpunan node-node, menggunakan fungsi $adj(v) = <u_0, u_1,\ldots,u_{d-1}>$, dengan d adalah derajat dari node v. Pengganti dari masing-masing busur $e = (u_i,v)$ adalah $s(<u_i,v>)=<v,u_{(j+1)} \bmod d>$, untuk tiap-tiap $0 \leq i \leq d$ -1. Untuk lebih jelasnya, perhatikan contoh sebagai berikut.

Contoh:
Diberikan sebuah tree $T = (V, E)$ seperti gambar 1(a). Urutan node-node yang bersebelahan dengan masing-masing node v dan fungsi pengganti yang dihasilkan ada pada gambar 1(b) dan 1(c). dan rangkaian terarah $T' = (V, E')$ ada pada gambar 1(d). Catatan: tree pada gambar 1(a) menyatakan ekspresi $f(f(a,b),f(f(a,a),c))$.



1(a)

V	adj(v)
1	2,3
2	4, 5, 1
3	6, 7, 1
4	2
5	2
6	8, 9, 3
7	3
8	6
9	6

1(b)

arc	2,1	3,1	4,2	5,2	1,2	6,3	7,3	1,3	2,4	2,5	8,6	9,6	3,6	3,7	6,8	6,9
succ	1,3	1,2	2,5	2,1	2,4	3,7	3,1	3,6	4,2	5,2	6,9	6,3	6,8	7,3	8,6	9,6

1(c)

1,2 → 2,4 → 4,2 → 2,5 → 5,2 → 2,1 → 1,3 → 3,6 →
6,8 → 8,6 → 6,9 → 6,3 → 3,7 → 7,3 → 3,1

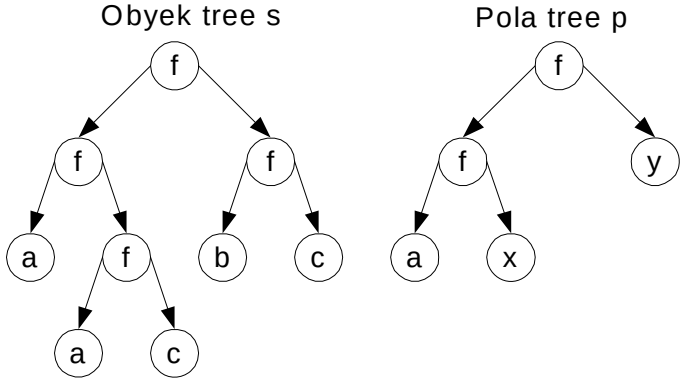
1(d)

Gambar 1. Rangkaian Euler dari sebuah *tree*.

Salah satu cara efisien untuk menyelesaikan masalah pengorderan node-node dari tree adalah dengan mengoperasikan *linked-list*. Diberikan $T = (V, E)$, untuk setiap node $v \in V$ node-node yang bersebelahan dengan v adalah elemen dari *linked-list* L[v]= $<u_0, u_1,\ldots,u_{d-1}>$. Sehingga T '= (V, E'), L[v] dapat diakses untuk menyatakan semua busur dari v, yaitu $<v,u_0>$, $<v,u_1>,\ldots,<v,u_{d-1}>$. Dengan demikian L[$v$] menyatakan secara unik semua busur T '= (V, E'). untuk mendapatkan busur $<u_i,v>$ dapat dilakukan dengan menambah *pointer* pada tiap-tiap node u. Kemudian dengan proses *rooting* didapatkan keluaran string berurutan mulai dari root (*root*) dan kembali ke root.

3. PROSES *MATCHING* PADA EKSPRESI

Untuk memudahkan pembahasan kami akan mempergunakan sebuah obyek tree *s* dan pola tree *p* sederhana seperti pada gambar 2. Pada aplikasi lanjut memungkinkan menggunakan pola-pola tree sebanyak jumlah ekspresi tetapan yang dipakai untuk menyelesaikan *matching* pada obyek tree yang besar dan rumit. Pada bab ini kami menyajikan beberapa definisi yang berkaitan dengan proses *matching* pada suatu ekspresi yang bermanfaat untuk pembahasan pada aspek perancangan algoritma.



Gambar 2. Obyek tree dan Pola *tree*.

Pada set variabel V dan set simbul fungsi F kami mengambil asumsi $V \cap F = \phi$. Masing-masing simbul fungsi f merupakan ariti a_f, intejer positif dan unik. Ariti nol adalah konstanta. Sehingga suatu ekspresi dapat didefinisikan sebagai berikut.

Definisi 1:

1. Sebuah variabel dan atau sebuah konstata adalah ekspresi, dan
2. Jika $f \in F$ dan $t_1, t_2,\ldots, t_{af}$ adalah ekspresi demikian juga $f(t_1, t_2,\ldots, t_{af})$
3. Ekspresi $f_1(f_2(a,b),b)$ dan $f_1(b,f_2(a,b))$ adalah dua buah ekspresi yang berbeda.

Matching pola tree pada obyek *tree* didefinisikan sebagai berikut. Diberikan sebuah obyek tree *s* berlabel dan tanpa variabel dan pola tree *p* berlabel dan mempunyai *k* variabel.

Definisi 2:
p matching s pada node x_i adalah bila dan hanya bila terdapat sub-sub tree menyatakan ekspresi t_1, t_2,…, t_n dan dengan mengganti t_i untuk variabel-variabel yang muncul pada *p* akan didapatkan tree baru yang identik dengan subtree dari s dengan root x_i.

Pada gambar 2, *p matching s* adalah pada saat didapatkan $t_1 = f(a,c)$ dan $t_2 = f(b,c)$ yang masing-masing adalah substitusi untuk variabel-variabel *x* dan *y*.

Dalam mendesain algoritma untuk *matching* kami mempergunakan pendekatan proses *matching* string yang mana adalah hasil dari [4], yang kami definisikan ulang sebagai berikut.

Definisi 3:
String *p matching* string *s* pada posisi string ke *i* adalah bila dan hanya bila string s_1, s_2,…, s_k dan dengan mensubstitusi s_i untuk variabel-variabel pada string *p* akan didapatkan string baru *x* yang identik dengan substring *s* pada posisi ke *i* dan *i* + |*x*| -1.

Pada gambar 2, didapat string rangkaian Euler *Ep* = *f*,*f*,*a*,*f*,*x*,*f*,*f*,*y*,*f* dan *E*s = *f*,*f*,*a*,*f*,*a*,*f*,*c*,*f*,*f*,*f*,*f b*,*f*,*c*,*f*,*f*. dapat diperiksa bahwa *Ep matches Es* untuk x = *f*,*a*,*f*,*c*,*f* dan *y* = *f*,*b*,*f*,*c*,*f*.

Pada algoritma *matching* string diperlukan masukan obyek string dan pola string masing-masing pada array |*Es*| dan |*Ep*| sebagai keluaran proses linearisasi menggunakan teknik Euler. Kemudian untuk verifikasi pada algoritma kami memanfaatkan sifat-sifat *Es*| dan |*Ep*| sebagai berikut.

1. Tiap-tiap daun (*lead*) dari tree hanya muncul satu kali pada rangkaian Euler.
2. Node yang mempunyai ariti *af* muncul sebanyak *af* + 1 kali.
3. Substring pada rangkaian Euler pada antara munculnya node *x* pertama dan terakhir adalah juga rangkaian Euler dengan root *x*.

Step-step dari algoritma untuk *matching* ekspresi adalah sebagai berikut:

1. Linearisasi pada obyek dan pola tree yang diberikan menggunakan teknik Euler.
2. Proses *rooting* menggunakan *prefix-sum* pada tiap edge yang terbentuk.
3. Partisi pola string untuk membentuk struktur pola string: $\sigma_1 v_1 \sigma_2 v_2 \ldots \sigma_k v_k \sigma_k v_{k-1}$, *v* adalah variabel string.
4. Akses tabel untuk mendata komparasi string pada tiap-tiap substring.

4. ALGORITMA *MATCHING* EKSPRESI

Linearisasi tree:
Input: Sebuah tree berlabel, tiap node mempunyai 2 buah pointer
Output: List struktur node, pointer edge menyatakan rangkaian terarah Euler

Begin

1. Lakukan jejakan (*traversal*) dengan *Dept-First-Search* oder
2. Pada tiap node, cek struktur pointer dan tentukan adjacent nodes,
   Aktifkan *linked-list*. *Update* struktur pointer berdasar $adj(v) = < u_0, u_1,\ldots,u_{d-1} >$
3. Dengan fungsi pengganti $s(< u_i,v >) = < v,u_{(i+1)} \bmod d >$ set edge variabel,
   Update struktur node pointer. Simpan variabel adjacent edge pada array.

End.

Proses rooting:
Input: List struktur node, pointer edge untuk rangkaian terarah Euler
Output: List struktur node. Elemen pertama adalah root.

Begin

1. Berikan harga 1 pada tiap-tiap edge <*x*,*y*> untuk x > y dan harga 0 untuk x < y
2. Aktifkan prosedur prefix-sum untuk menghilangkan harga prefik tiap edge
3. For *j = 1 to all edge*, simpan struktur node ke array mulai dari yang harga Prefik-sum=0.
4. Verifikasi struktur node.

End.
Partisi pola string:
Input: Array string |*Ep*|
Output: Array hasil partisi pola string *p*

Begin
1. Set *k* = jumlah variabel pada |*Ep*|. Potong input array string |*Ep*| menjadi *k*+1.
2. *For j = 1 to length* (|*Ep*|), set string pada posisi 1..[v2 – 1] ke σ_1, posisi [v1+1]…[v2-1] ke σ_2 dan seterusnya, sehingga |*Ep*| = $\sigma_1 v_1 \sigma_2 v_2 \ldots \sigma_k v_k \sigma_{k-1} \ldots$
3. Simpan hasil partisi pada array. Untuk pola tree pada gambar2, σ_1= *f*,*f*,*a*,*f* σ_2= *f*,*f* dan σ_1= *f*.

End.

Untuk algoritma *matching* secara keseluruhan adalah mempergunakan ketiga buah algoritma di atas dengan menambah prosedur komparasi string untuk tiap-tiap partisi pola string dan membuat tabel untuk tiap-tiap komparasi dengan substring. Selanjutnya adalah melakukan *matching* untuk tiap-tiap variabel yang muncul pada pola string. Proses ini dengan mudah dapat dilaksanakan dengan menjejaki obyek string |*Es*|, untuk tiap-tiap partisi string dari pola string, baca posisi string |Ep| pada tabel kemudian melakukan komparasi pada posisi variabel-variabel pada |Es|.

5. KESIMPULAN

Untuk keperluan aplikasi, algoritma yang kami rancang adalah sangat bermanfaat, karena dapat dimanfaatkan untuk mengkomputasi hampir segala masalah. Akan tetapi pemakai dituntut untuk dapat menyusun tetapan (*rules*), sedetail mungkin, bahkan kalau mungkin sampai dengan level aksion dan merumuskan masalah yang akan diselesaikan kedalam bentuk fungsi-fungsi untuk pemrograman struktur data tree.

DAFTAR PUSTAKA

[1] N.Dershowitz, JP. Jouannaud, JW. Klop, **"*Mode Problems in Rewriting Technique*"**,, Research report of Computer Science, Report CS-R0332, 1993.
[2] N.Dershowitz, JP. Jouannaud, **"*Rewriting System*"**, Handbook of Theoretical Computer Science, North-Holland, chapter 6, pages 243-320, 1990.
[3] Joseph Jaja, **"*An Introduction to Parallel Algorithms*"**, Addison-Wesley Publicing Company, USA, 1992.
[4] Herbert Schildt, **"*The Complete C Reference*"**, Osborne Mc. Graw-Hill 3d Edition, California, USA, 1995.
[5] Z.Galil, **"*Optimal Parallel Algorithms for String Matching*"**, Jurnal Information and Control, pp. 144-157, 1985.