Aplikasi Fungsi dalam Komputasi sebagai Alternatif dari Mesin Turing

Moch. Sofyan Firdaus - 13521083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13521083@std.stei.itb.ac.id

***Abstraksi*—Dalaam komputasi, terdapat model abstraksi komputer yang dapat mengeksekusi berbagai instruksi (algoritme) untuk menyelesaikan masalah-masalah komputasi. Yang paling terkenal dari model tersebut adalah Mesin Turing yang diciptakan oleh matematikawan Alan Turing pada tahun 1936. Namun, pada tahun 1930 matematikawan Alonzo Church menciptakan model komputasi berbasis fungsi matematika yang setara dengan mesin Turing dan sekarang disebut Kalkulus lambda. Kalkulus lambda selanjutnya digunakan sebagai dasar untuk mengembangkan bahasa pemrograman fungsional.**

***Keywords*—mesin Turing, kalkulus lambda, fungsi, pemrograman fungsional.**

I. PENDAHULUAN

Komputasi merupakan salah satu cabang ilmu sains yang mempelajari cara-cara menyelesaikan masalah dan pengolahan informasi menggunakan komputer. Sejarah komputasi sudah dimulai sejak berabad-abad silam. Alat komputasi yang paling awal dibuat adalah abakus yang diciptakan oleh bangsa Mesopotamia sekitar 2500 tahun sebelum masehi. Abakus dapat digunakan untuk melakukan perhitungan aritmetika dasar seperti penjumalahan, pengurangan, perkalian, dan pembagian. Selanjutnya teknologi komputasi semakin berkembang hingga akhirnya matematikawan Charles Babbage memelopori teknologi komputer.

Charles Babbage adalah seorang matematikawan dan juga ahli mekanik. Ia lahir pada tahun 1791 di Teignmouth, Devon, Inggris. Charles dikenal sebagai "Bapak Komputer" atas jasanya dalam mengembangkan mesin pertama yang disebut mesin analitik. Mesin analitik merupakan mesin yang dapat melakukan perhitungan dan menyelesaikan masalah matematika dengan cepat. Meskipun mesin analitik tidak pernah selesai dibuat, mesin tersebut menjadi cikal bakal dari komputer.

Pada abad ke-20, terjadi pengembangan yang sangat penting dalam komputasi. Pengembangan tersebut dibawakan oleh seorang matematikawan sekaligus ahli ilmuwan komputer asal Inggris yang bernama Alan Turing. Pada tahun 1936, Alan Turing menulis sebuah makalah yang menjadi salah satu karya paling penting dalam komputasi. Makalah tersebut berjudul "*On Computable Numbers, with an Application to the Entscheidungsproblem.*" Dalam makalah tersebut, Alan Turing mengembangkan sebuah model abstrak yang digunakan untuk melakukan komputasi teoritis. Mesin ini merupakan sebuah abstraksi dari komputer yang mampu mengeksekusi serangkaian instruksi untuk menyelesaikan masalah komputasi.

Pada tahun 1936, Alan Turing bersama Alonzo Church mengembangkan sebuah teori yang disebut tesis Church-Turing. Tesis ini menyatakan bahwa semua algoritme yang dapat dibuat oleh manusia dapat diterjemahkan ke dalam sebuah mesin Turing, dan karena itu mesin Turing dapat melakukan semua hal yang dapat dilakukan oleh komputer modern. Tesis Church-Turing menjadi landasan bagi teori komputasi dan menunjukkan bahwa mesin Turing merupakan representasi yang akurat dari sebuah komputer.

Gambar 1. Alonzo Church (kiri) dan Alan Turing (kanan)
(Sumber: cacm.acm.com)

Dalam proses pembuktiannya, Church menciptakan sebuah model abstraksi lain yang dinamakan kalkulus lambda. Kalkulus lambda merupakan bentuk matematika yang digunakan untuk menganalisis struktur abstrak dari sebuah fungsi atau ekspresi secara lebih mendalam. Kalkulus lambda dapat digunakan untuk memodelkan permasalahan komputasi ke dalam bentuk fungsi matematika murni. Diketahui, semua mesin Turing dapat diterjemahkan ke dalam bentuk kalkulus lambda. Oleh karena itu, kalkulus lambda merupakan alternatif dari mesin Turing sebagai representasi abstrak dari komputer.

Mesin Turing dan kalkulus lambda memiliki kekuatan yang sama dalam hal komputasi. Semua hal yang dapat diselesaikan dengan mesin Turing dapat diselesaikan juga dengan kalkulus

lambda. Perbedaan antara mesin Turing dan kalkulus lambda adalah mesin Turing bekerja berdasarkan *state* atau kondisi tertentu yang dicapai suatu mesin, sedangkan kalkulus lambda bekerja berdasarkan fungsi matematika murni. Namun, meski mesin Turing dan kalkulus lambda merupakan dua hal yang ekuivalen, komputer modern dibangun berdasarkan perilaku dari mesin Turing karena model kalkulus lambda tidak dapat diaplikasikan secara fisik. Walaupun demikian, kalkulus lambda menjadi dasar untuk mengembangkan bahasa-bahasa pemrograman fungsional seperti Haskell, Lisp, Scala, dan sebagainya. Di sisi lain, mesin Turing menjadi dasar bagi bahasa-bahasa pemrograman imperatif seperti C, C++, Java, Python, dan sebagainya.

II. Landasan Teori

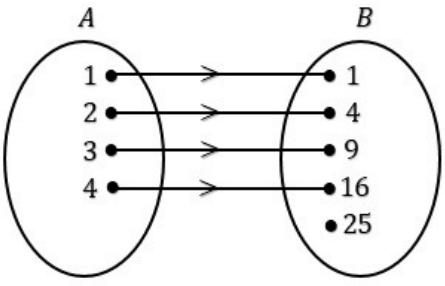
A. Relasi

Relasi merupakan hubungan antara suatu elemen di dalam suatu himpunan dengan elemen di dalam himpunan lainnya. Elemen-elemen tersebut disebut objek dan relasi menunjukkan bagaimana elemen-elemen tersebut terhubung satu sama lain. Secara formal, relasi didefinisikan sebagai berikut.

- Misalkan R adalah suatu relasi biner antara himpunan A dan himpunan B, maka R merupakan himpunan bagian dari $A \times B$ atau $R \subseteq (A \times B)$.
- $a\,R\,b$ menyatakan $(a,b) \in R$, yang artinya a dihubungkan dengan b dengan relasi R.
- $a\not R\,b$ menyatakan $(a,b) \notin R$, yang artinya a dihubungkan dengan b dengan relasi R.
- Himpunan A disebut daerah asal (domain) dari R dan himpunan B disebut daerah hasil (kodomain) dari R.
- Terdapat relasi khusus yang menghubungkan sebuah himpunan, misalnya A. Misalkan relasi tersebut adalah R, maka $R \subseteq (A \times A)$.

Relasi dapat direpresentasikan dengan berbagai cara, di antaranya:

1. *Diagram Panah*



Gambar 2. Representasi relasi dengan diagram panah
(Sumber: maretong.com)

2. *Tabel*

Dalam representasi tabel, domain disimpan pada kolom pertama dan kodomain disimpan pada kolom kedua.

A	B
Ayam	Unggas
Sapi	Mamalia
Kambing	Mamalia
Bebek	Unggas

Tabel 1. Representasi relasi dengan tabel

3. *Matriks*

Misalkan R adalah relasi dari himpunan $A=\{a_1,a_2,\ldots,a_m\}$ dan $B=\{b_1,b_2,\ldots,b_n\}$. Relasi R dapat direpresentasikan dengan matriks $M=[m_{ij}]$,

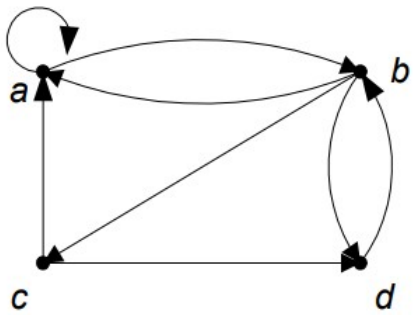
$$M = \begin{array}{c} \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{array} \begin{array}{c} \begin{array}{cccc} b_1 & b_2 & \cdots & b_n \end{array} \\ \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ m_{m1} & m_{m2} & \cdots & m_{mn} \end{bmatrix} \end{array}$$

Gambar 3. Representasi relasi dengan matriks
(Sumber: Relasi dan Fungsi Bagian 1.pdf)

dengan $m_{ij}=1$ jika $(a_i,b_j) \in R$ atau $m_{ij}=0$ jika $(a_i,b_j) \notin R$.

4. *Graf Berarah*

Representasi relasi dengan graf berarah hanya didefinisikan untuk relasi yang menghubungkan satu himpunan. Setiap elemen himpunan direpresentasikan dengan sebuah simpul, sedangkan setiap pasangan terurut direpresentasikan dengan sebuah busur.



Gambar 4. Representasi relasi dengan graf berarah
(Sumber: Relasi dan Fungsi Bagian 2.pdf)

Relasi pada sebuah himpunan memiliki beberapa sifat:

1. *Refleksif*

Suatu relasi dikatakan refleksif jika dan hanya jika setiap elemen pada himpunan memiliki relasi dengan dirinya sendiri atau untuk setiap $a \in A$, $(a,a) \in R$.

2. *Transitif*

Relasi R dikatakan transitif jika untuk setiap $a,b,c \in A$, $a\,R\,b$ dan $b\,R\,c$ berakibat $a\,R\,c$

3. *Simetri*

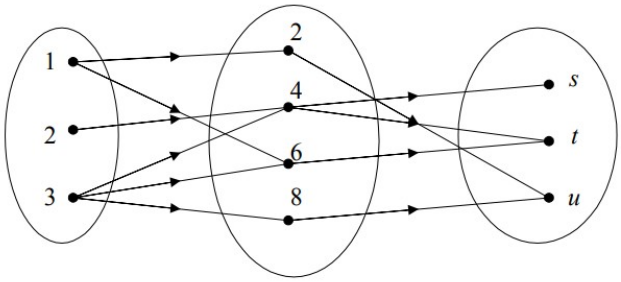
Relasi R dikatakan simetrik jika untuk setiap $a,b \in A$

, $(a,b) \in R$ berakibat $(b,a) \in R$.

4. *Antisimetri*
   Relasi yang bersifat antisimetri jika untuk setiap $a,b \in A$, $(a,b) \in R$ berakibat $(b,a) \in R$ hanya jika $a=b$.

Relasi kebalikan atau relasi inversi adalah suatu relasi yang didefinisikan pada suatu himpunan yang sama dengan relasi yang sudah ada, tetapi setiap pasangan elemen pada relasi inversi memiliki elemen pertama dan elemen kedua yang dibalik. Misalkan R adalah suatu relasi yang menghubungkan himpunan A ke himpunan B, maka relasi inversi, dinotasikan dengan R^{-1}, adalah relasi yang menghubungkan himpunan B ke himpunan A. Secara formal, relasi inversi didefinisikan oleh $R=\{(b,a) \mid (a,b)\in R\}$.

Suatu relasi dapat dikomposisikan dengan relasi lainnya. Komposisi relasi adalah penggabungan dua atau lebih relasi menjadi suatu relasi yang baru. Misalkan R adalah relasi yang menghubungkan himpunan A ke himpunan B dan S adalah relasi yang menghubungkan himpunan B ke himpunan C, komposisi dari R dan S adalah $R \circ S=\{(a,c) \mid a\in A, c\in C$ dan untuk beberapa $b\in B, (a,b)\in R$ dan $(b,c)\in S\}$. Komposisi relasi dapat direpresentasikan secara visual dengan diagram panah seperti berikut.



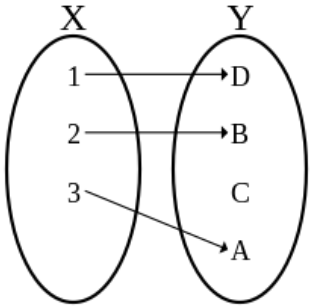
Gambar 5. Representasi dari komposisi fungsi
(Sumber: Relasi dan Fungsi Bagian2 (2020).pdf)

B. Fungsi

Fungsi, sering disebut juga pemetaan atau transformasi, merupakan suatu relasi unik antara dua himpunan, yang dinotasikan dengan $f : X \rightarrow Y$ yang artinya fungsi f memetakan X ke Y. Himpunan X disebut daerah asal (domain) atau sering disebut juga sebagai *input*, sedangkan himpunan Y disebut daerah hasil (kodomain) atau sering disebut juga *output*. Fungsi adalah relasi khusus yang mengaitkan setiap elemen di dalam daerah asal ke satu dan hanya satu elemen di dalam daerah hasil, yang dituliskan sebagai $f(x)=y$ dengan y disebut sebagai bayangan dan x disebut sebagai pra-bayagan. Himpunan yang berisi semua hasil pemetaan dari fungsi f dinamakan jelajah (*range*) dan merupakan himpunan bagian dari daerah hasil (kodomain).

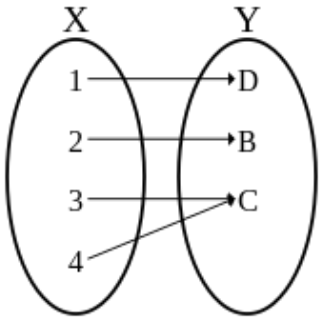
Seperti relasi, fungsi memiliki beberapa sifat, antara lain:

1. *Injektif*
   Fungsi injektif adalah suatu fungsi yang setiap elemen dari daerah asal dipetakan ke elemen yang berbeda-beda di dalam daerah hasil. Dengan kata lain, tidak ada elemen di dalam himpunan jelajah yang merupakan hasil pemetaan dari dua atau lebih elemen yang berbeda. Fungsi injektif disebut juga fungsi satu-ke-satu



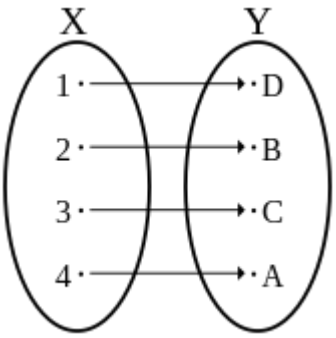
Gambar 6. Fungsi injektif
(Sumber: wikipedia.com)

2. *Surjektif*
   Fungsi surjektif adalah fungsi yang himpunan jelajahnya (*range*-nya) sama dengan daerah hasilnya. Dengan kata lain, tidak ada elemen di dalam daerah hasil yang bukan hasil pemetaan (bayangan) dari elemen di dalam daerah asal. Fungsi surjektif disebut juga fungsi pada (*onto*).



Gambar 7. Fungsi surjektif
(Sumber: wikipedia.com)

3. *Bijektif*
   Fungsi bijektif adalah fungsi yang bersifat injektif sekaligus bersifat surjektif. Dengan kata lain, setiap elemen di dalam daerah asal dipetakan ke daerah hasil dan setiap elemen di dalam daerah hasil merupakan hasil hasil pemetaan (bayangan) dari elemen daerah asal. Fungsi bijektif disebut juga korespondensi satu-satu.



Gambar 8. Fungsi bijektif
(Sumber: wikipedia.com)

Sama halnya seperti relasi, fungsi memilki fungsi balikan. Namun, suatu fungsi dapat memiliki balikan jika dan hanya jika fungsi tersebut merupakan fungsi bijektif. Jika didefiniskan $f : X \rightarrow Y$ dengan $f(x) = y$, maka fungsi inversinya didefinisikan $f^{-1} : Y \rightarrow X$ dengan $f^{-1}(y) = x$.

Suatu fungsi juga dapat dikomposisikan dengan fungsi lainnya. Misalkan $f : X \rightarrow Y$, dan $g : Y \rightarrow Z$, maka komposisi fungsi *f* dan *g* didefinisikan oleh $(f \circ g)(x) = f(g(x))$.

Suatu fungsi dapat didefinisikan dengan menggunakan dirinya sendiri. Fungsi seperti ini disebut dengan fungsi rekursif. Contoh fungsi rekursif yaitu fungsi yang menghasilkan bilangan fibonacci ke-n. Fungsi fibonacci didefinisikan secara rekursif oleh $fib(n) = fib(n-2) + fib(n-1)$, dengan basis $fib(1) = 1$, dan $fib(2) = 1$.

III. PEMBAHASAN

A. Kalkulus Lambda

1. Pengertian Kalkulus Lambda

Kalkulus lambda adalah teknik matematika yang berguna untuk menganalisis proses pemrograman komputer. Kalkulus lambda merupakan suatu cara untuk mengkonstruksi dan mengevaluasi ekspresi pemrograman menggunakan konsep dasar seperti fungsi, variabel, dan parameter. Kalkulus lambda juga dapat digunakan untuk menganalisis kemungkinan dan batasan dari berbagai algoritma dan struktur data yang digunakan dalam permrograman.

Kalkulus lambda merupakan formalisasi dari konsep efektifitas komputabilitas, seperti halnya mesin Turing. Faktanya semua turing dapat diterjemahkan menjadi kallkulus lambda dan sebaliknya kalkulus lambda juga bisa diterjemahkan menjadi mesin Turing sehingga mesin Turing dan kalkulus Lambda memilki kemampuan yang sama dalam komputasi dan penyelesaian masalah komputasi. Perbedaan besar antara antara mesin Turing dan kalkulus lambda adalah mesin Turing memanfaatkan *state* internal sedangkan kalkulus lambda murni menggunakan fungsi matematika.

2. Struktur Kalkulus Lambda

Lambda kalculus terdiri dari tiga elemen: ekspresi, variabel, dan abstraksi. Ekspresi adalah kategori yang paling luas dan dapat mencakup variabel, abstraksi, atau kombinasi keduanya. Variabel adalah elemen sederhana yang tidak memiliki makna atau nilai yang melekat dan hanya merupakan penanda untuk masukan ke sebuah fungsi. Abstraksi adalah sebuah fungsi yang nantinya diaplikasikan ke sebuah argumen. Argumen adalah nilai yang diberikan kepada fungsi tersebut. Abstraksi terdiri dari dua bagian: kepala dan tubuh. Kepala dari fungsi ini adalah simbol lambda (λ) yang diikuti oleh nama variabel. Tubuh dari fungsi tersebut merupakan sebuah ekspresi. Contoh fungsi sederhana dalam kalkulus lambda terlihat seperti berikut:

$$\lambda x.x$$

Variabel yang terletak pada kepala adalah parameter. Parameter mengikat semua variabel yang sama yang berada di dalam tubuh fungsi. Ketika fungsi tersebut diaplikasikan ke sebuah argumen, setiap variabel yang sama dengan parameter, dalam hal ini variabel *x*, yang berada di dalam tubuh fungsi akan digantikan dengan nilai yang sama dengan argumen. Simbol titik memisahkan antara kepala fungsi dengan tubuhnya.

Sebuah abstraksi secara keseluruhan tidak memilki nama. Hal itu disebut abstraksi karena merupakan sebuah konsep yang luas yang berasal dari contoh spesifik dari suatu masalah. Sebuah nama dapat digunakan untuk mewakili nilai-nilai spesifik dalam abstraksi sehingga memungkinkan untuk menerapkan konsep umum yang sama untuk nilai atau tipe nilai yang berbeda.

Nama dari suatu parameter tidak memiliki arti semantik kecuali dalam ekspresinya sendiri. Oleh sebab itu,

$$\lambda x.x$$
$$\lambda y.y$$
$$\lambda z.z$$

merupakan fungsi yang sama.

3. Reduksi Beta

Ketika memasukkan argumen ke dalam sebuah fungsi dengan nilai tertentu, semua variabel yang mengikat parameter dalam tubuh fungsi digantikan dengan nilai argumen tersebut dan kepala dari abstraksi juga dihilangkan karena itu sudah tidak memiliki arti lagi. Proses ini disebut dengan reduksi beta.

Misalkan sebuah fungsi didefinisikan sebagai berikut:

$$\lambda x.x+1$$

Jika dilakukan reduksi beta dengan mengaplikasikan fungsi tersebut ke sebuah nilai, misalnya 1, maka setiap parameter, dalam hal ini *x*, harus digantikan dengan 1, lalu kepala dari abstraksi dihilangkan.

$$\begin{aligned}(\lambda x.x+1)\,1\\ [x:=1]\\ 1+1\\ 2\end{aligned}$$

Setelah dilakukan reduksi beta, fungsi di atas menghasilkan nilai 2.

Pada dasarnya, setiap ekspresi dapat menjadi argumen untuk fungsi, termasuk fungsi yang lain.

$$\begin{aligned}(\lambda x.x)(\lambda y.y)\\ [x:=(\lambda y.y)]\\ \lambda y.y\end{aligned}$$

Proses reduksi beta akan terus dilakukan sampai terbentuk ekspresi yang paling sederhana sehingga proses beta reduksi tidak bisa dilakukan lagi. Bentuk ekspresi paling sederhana ini disebut dengan bentuk beta normal.

$$\begin{aligned}(\lambda x.x)(\lambda y.y)z\\ [x:=(\lambda y.y)]\\ (\lambda y.y)z\\ [y:=z]\\ z\end{aligned}$$

Dalam sebuah ekspresi, dimungkinkan terdapat variabel bebas yang tidak diikat oleh kepala abstraksi. Variabel bebas tersebut tidak bisa digantikan dengan apapun.

$$\begin{aligned}(\lambda u.uv)w\\ [u:=w]\\ wv\end{aligned}$$

Bentuk tersebut sudah merupakan bentuk beta normal sehingga tidak bisa direduksi lagi.

4. Currying

Suatu lambda pada abstraksi hanya dapat mengikat satu variabel. Supaya dapat mengaplikasikan fugsi ke beberapa

argumen, diperlukan suatu abstraksi bersarang yang setiap abstraksinya menerima masing-masing satu argumen. Proses ini disebut dengan *currying*, dinamakan berdasarkan matematikawan Haskell Curry.

$$(\lambda xyz.xz(yz))(\lambda mn.m)(\lambda p.p)$$
$$(\lambda x\lambda y\,\lambda z.xz(yz))(\lambda m\lambda n.m)(\lambda p.p)$$
$$[x:=(\lambda m\lambda n.m)]$$
$$(\lambda y\lambda z(\lambda m\lambda n.m)z(yz))(\lambda p.p)$$
$$[y:=(\lambda p.p)]$$
$$\lambda z(\lambda m\lambda n.m)(z)((\lambda p.p)z)$$
$$[m:=z]$$
$$\lambda z(\lambda n.z)((\lambda p.p)z)$$
$$[n:=((\lambda p.p)z)]$$
$$\lambda z.z$$

B. Aplikasi Kalkulus Lambda

Kalkulus lambda menjadi salah satu konsep penting dalam bidang komputasi. Kalkulus lambda saat ini dijadikan dasar untuk mengembangkan berbagai macam bahasa pemrograman dengan paradigma fungsional. Bahasa pemrograman fungsional seperti Lisp, Haskell, Scala, OCaml, dan merupakan implementasi dari kalkulus lambda. Bahasa pemrograman fungsional menekankan pada abstraksi sehingga semua hal dinyatakan sebagai fungsi. Hal ini menyebabkan membuat aplikasi yang *stateful* menjadi sulit karena aplikasi seperti itu membutuhkan pendekatan prosedural.

Meskipun bahasa pemrograman fungsional cukup sulit digunakan untuk membuat aplikasi yang *stateful*, aplikasi tersebut masih bisa dibuat karena pada dasarnya kemampuan bahasa fungsional yang berdasarkan kalkulus lambda dengan bahasa prosedural yang berdasarkan mesin Turing adalah sama sehingga bahasa pemrograman fungsional dapat menyelesaikan setiap persoalan yang bisa diselesaikan oleh bahasa pemrograman prosedural. Hal ini bisa dibuktikan dengan *turing completeness*.

Turing completeness adalah sebuah konsep yang digunakan untuk mengukur kemampuan sebuah mesin atau bahasa pemrograman dalam menyelesaikan persoalan komputasi. Bahasa pemrograman yang *turing complete* rmerupakan bahasa yang dapat menyelesaikan segaka permasalahan yang secara teoritis dapat diselesaikan oleh sebuah komputer. Ada beberapa cara untuk membuktikan bahwa sebuah bahasa pemrograman merupakan bahasa yan *turing complete*. Salah satu caranya adalah dengan mengimplementasikan *rule 110*.

R*ule 110* merupakan salah satu aturan dalam automata selular yang menentukan bagaimana satu sel berubah dari satu generasi ke generasi yang lain. Automata selular adalah suatu sistem komputasi yang terdiri dari sekumpulan sel yang saling terhubung dan mampu menjalankan suatu aturan kompleks untuk mengahasilkan pola yang kompleks pula.

Rule 110 adalah salah satu aturan yang paling dikenal dalam automata selular karena dianggap sebagai salah satu contoh yang paling kompleks dari aturan yang dapat dijalankan oleh sebuah automata selular Aturan ini ditulis dalam bentuk biner, yaitu 01101110, yang merupakan bilangan desimal 110 dalam sistem biner.

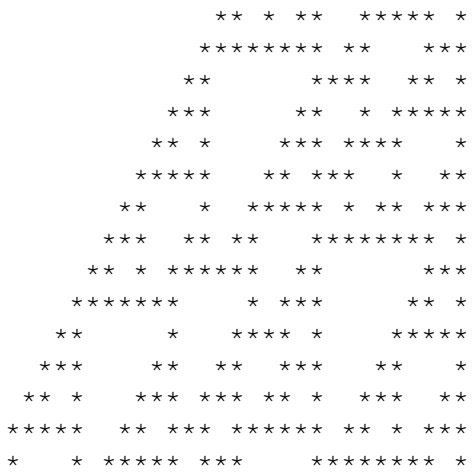
Untuk menentukan bagaimana sel akan berubah dari satu generasi ke generasi berikutnya, rule 110 memperhatikan tiga sel yang ada di sebelah kiri, sel saat ini, dan tiga sel di sebelah kanan. Jika sel saat ini bernilai 1 dan sel-sel di sebelahnya bernilai 1 1 0, maka sel saat ini akan berubah menjadi 0 pada generasi berikutnya. Jika kondisi ini tidak terpenuhi, maka sel saat ini akan tetap bernilai 1.

Berikut adalah implementasi *rule 110* dalam bahasa Haskell yang merupakan salah satu bahasa pemrograman fungsional yang didasari dengan kalkulus lambda.

```
module Rule110 where

import Data.Bits

initialState :: Int -> [Int]
initialState n = take (n - 2) (repeat 0) +
+ [1, 0]

nextState :: [Int] -> [Int]
nextState state = updateState 1 pattern
state
  where
   pattern = ((state !! 0) `shiftL` 1) .|.
(state !! 1)
   n = length state
   updateState :: Int -> Int -> [Int] ->
[Int]
   updateState x pattern' state'
      | x == n - 1 = state'
      | otherwise = updateState (x + 1)
pattern'' state''
      where
        pattern'' = ((pattern' `shiftL` 1)
.&. 7) .|. (state' !! (x + 1))
        (a, c) = splitAt (x - 1) state'
        b = ((110 :: Int) `shiftR`
pattern') .&. 1
        state'' = a ++ (b : (tail c))
```

Berikut adalah output dari program di atas dengan melakukan iterasi sebanyak 28 kali dan ukuran banyak sel setiap generasinya adalah 30

```
λ> putStrLn $ unlines $ map (map (\x -> if
x == 0 then ' ' else '*')) $ take 30 $
iterate nextState $ initialState 30
                              *
                             **
                            ***
                           ** *
                          *****
                         **   *
                        ***  **
                       ** * ***
                      ******* *
                     **     ***
                    ***    ** *
                   ** *   *****
                  *****  **   *
                 **   * ***  **
                ***  **** * ***
```



Pola tersebut merupakan pola dari *rule 110*. Dengan demikian, terbukti bahwa bahasa Haskell merupakan bahasa yang *turing complete* meskipun berdasarkan kalkulus lambda.

IV. Simpulan

Berdasarkan bab III, terbukti bahwa kalkulus lambda memiliki kemampuan komputasi yang sama dengan mesin Turing sehingga kalkulus lambda memang dapat dijadikan alternatif untuk mesin turing dalam menyelesaikan persoalan komputasi.

VI. Ucapan Terima Kasih

Pertama-tama, penulis mengucapkan puji dan syukur kepada Allah swt. atas nikmat dan ksehatan yang telah dikaruniakan-Nya sehingga penulis dapat menyelesaikan tugas makalah mata kuliah IF2120 Matematika Diskrit. Saya mengucapkan terima kasih yang sebesar-besarnya kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen dan pengajar yang telah memberikan ilmu yang bermanfaat bagi penulis sehingga penulis dapat menyelesaikan pembuatan makalah ini. Semoga Allah swt. membalas semua kebaikan dengan kebaikan yang berliapat ganda. Semoga pembahasan pada makalah ini dapat bermanfaat bagi siapa pun yang membaca dan dapat terus dikembangkan lebih lanjut lagi. Terakhir, penulis mohon maaf atas segala kekurangan dan kesalahan yang penulis lakukan dalam makalah ini.

References

[1] https://www.youtube.com/watch?v=eis11j_iGMs&t=154s. Diakses pada tanggal 9 Desember 2022

[2] Allen, C. & Moronuki, J. (n.d.). *Haskell Programming from First Principles*.

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, 12 Desember 2022

Moch. Sofyan Firdaus 13521083