XMLMate: *White-Box-Tool* untuk Sistem Berbasis XML

Tugas Akhir Mata Kuliah Keamanan Perangkat Lunak EL5215

Dina Budhi Utami (23214347)

Magister Teknik Elektro
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

dinabusoft@gmail.com

ABSTRAK

XMLMATE adalah *search-based-test-generator* untuk sistem berbasis XML. XMLMATE dikembangkan oleh akademisi di Saarland University. XMLMATE merupakan *white-box-tool* untuk membangkitkan populasi masukan secara acak dengan menggunakan algoritma genetik. XMLMATE memanfaatkan struktur program, skema XML, dan XML input untuk mengembangkan berbagai masukan XML yang valid. XMLMATE mengembangkan populasi masukan untuk meningkatkan *coverage* dari pengujian sistem. Pada tulisan ini dibahas tentang XMLMATE meliputi bagaimana melakukan pengujian white-box aplikasi berbasis xml dengan menggunakan XMLMATE dan bagaimana XMLMATE dapat membangkitkan populasi masukan secara otomatis.

Kata Kunci: XMLMATE, Algoritma Genetik, *White-box-tool,* XML

1. Pendahuluan

XML digunakan untuk mendeskripsikan data. Dengan standarisasi XML, aplikasi-aplikasi yang berbeda dapat dengan mudah berkomunikasi antar satu dengan yang lain. Pada zaman sekarang, xml banyak digunakan untuk perpindahan data baik pada sistem berbasis web, Android, dan lain-lain. Banyak *tools* yang dibuat untuk menguji kualitas aplikasi. Salah satu *tools* untuk menguji aplikasi berbasis XML adalah XMLMATE.

Proses pengujian pada suatu aplikasi dilakukan dengan cara melihat modul dan kemudian menganalisa kode dari program yang dibuat, apakah terdapat kesalahan atau tidak. Pada proses pengujian, diberikan *test case* ke dalam aplikasi, dan kemudian keluaran dari aplikasi tersebut dianalisis untuk mendeteksi perbedaan antara kondisi yang ada dan kondisi yang diinginkan. Perbandingan kondisi tersebut mengindikasikan apakah terdapat kesalahan dalam kode program. Tantangan dalam pengujian adalah bagaimana menentukan dan mengembangkan *test case* yang cukup pada pengujian suatu aplikasi. Konsep *automatic test generation* merupakan perkembangan terkini dalam *tools* pengujian. Dengan konsep ini, *tools* pengujian tidak hanya dapat mengeksekusi *test case* yang diberikan secara otomatis tetapi juga dapat mengembangkan *test* case secara otomatis. XMLMATE merupakan salah satu *white-box-tool* dimana tools ini dapat membangkitkan *test case* secara acak dengan menggunakan algoritma genetik. XMLMATE memanfaatkan struktur program, skema XML, dan masukan XML untuk mengembangkan berbagai *test case* yang valid. XMLMATE mengembangkan populasi masukan untuk meningkatkan *coverage* dari pengujian sistem.

Selanjutnya tulisan ini akan membahas mengenai XMLMATE dengan tahapan sebagai berikut: Bagian II akan membahas landasan teori mengenai XML dan pengujian *white box*, bagian III akan membahas proses pembangkitan *test case* dengan algoritma genetik pada XMLMATE, bagian IV akan membahas cara melakukan pengujian *white-box* aplikasi berbasis xml dengan menggunakan XMLMATE dan terakhir akan dibahas kesimpulan.

2. Landasan Teori

2.1. XML

XML diperkenalkan oleh W3C pada bulan Februari 1998. XML merupakan turunan dari SGML [11]. XML atau singkatan dari *Extensible Markup Language* berisi informasi, bentuk, label, dan struktur Informasi. Bahasa *markup* adalah sekumpulan aturan-aturan yang mendefinisikan suatu sintaksis yang digunakan untuk menjelaskan, dan mendeskripsikan teks atau data, dan bagaimana relasi dari data tersebut [12]. Berikut ini adalah contoh sebuah XML dalam dokumen MathML.

```
<?xml version="1.0" encoding="UTF-8"?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mi mathcolor="blue">x</mi>
  <mo>+</mo>
  <mn mathcolor="#00ff33" color="purple">7</mn>
  <mo>+</mo>
  <mn color="yellow">3</mn>
  <mo>+</mo>
  <mstyle color="red">
    <mi>z</mi>
    <mo>+</mo>
    <mi mathcolor="blue">q</mi>
    <mo>+</mo>
    <mi>q</mi>
  </mstyle>
</math>
```

Gambar 1 contoh dokumen XML

Seperti contoh pada gambar 1, XML memiliki elemen yang ditandai dengan tag pembuka yang diawali dengan < dan diakhiri dengan >, tag penutup diawali dengan </ diakhiri >. Tag pada dokumen XML bersifat *case sensitif* dimana tag pembuka dan tag penutup harus berpasangan.

Sebuah dokumen XML terdiri dari bagian-bagian yang disebut dengan *node*. Berikut adalah bagian-bagian dari XML:

- ***Root node*** yaitu *node* yang melingkupi keseluruhan dokumen. Dalam satu dokumen XML hanya ada satu *Root node*. *Node-node* yang lainnya berada di dalam *Root node*. Contoh *Root node* pada gambar 1 adalah math.

- ***Element node*** yaitu bagian dari dokumen XML yang ditandai dengan tag pembuka dan tag penutup. Contoh elemen pada gambar 1 adalah mi, mo, mn, dan mstyle.
- ***Attribute node*** ditulis pada tag awal sebuah elemen atau pada tag tunggal. Contoh attribute *node* pada gambar 1 adalah mathcolor=”*blue*”**.**
- ***Text node*** adalah text yang merupakan isi dari sebuah elemen, ditulis diantara tag pembuka dan tag penutup. Contoh text *node* pada gambar 1 adalah x.
- ***Header*** adalah baris pertama yang mendeklarasi XML. Pada contoh pada gambar 1, *header* mendefinisikan versi XML (1.0) dan pengkodean yang digunakan adalah UTF-8.

Karena XML bersifat mudah untuk dibaca dan ditulis baik oleh manusia maupun komputer, maka XML merupakan sebuah format yang dapat digunakan untuk perpindahan data (*interchange*) antar aplikasi dan platform yang berbeda (*platform independent*). XML dapat digunakan pada semua bahasa pemrograman dan datanya dapat ditransfer dengan mudah melalui protokol standar internet seperti HTTP.

2.2. Pengujian *White box*

Pengujian perangkat lunak adalah sekumpulan aktivitas untuk menemukan *error*. Pengujian dilakukan untuk memastikan apakah sistem bekerja sesuai dengan spesifikasi yang telah ditentukan. Pengujian *white box* adalah teknik verifikasi perangkat lunak dengan pengujian yang didasarkan pada pengecekan terhadap detil perancangan dari setiap modul untuk dapat meneliti dan menganalisa kode dari program yang dibuat. Pengujian *white box* disebut juga *structural testing*. Pada pengujian *white box* dibutuhkan akses ke dalam kode program. Perancangan *test case* pada pengujian ini didasarkan pada perancangan dan spesifikasi perangkat lunak. Kelebihan dari pengujian *white box* adalah sebagai berikut [14].

- Dapat mendeteksi kesalahan logika

  Dengan menggunakan sintaks ‘if’, metode pengujian *white box* ini menguji seluruh logika baki benar atau salah untuk menentukan *branch coverage*. Pengujian akan mencari dan mendeteksi segala kondisi yang dipercaya tidak sesuai.
- Dapat menguji perulangan

  Dengan mengunakan sitaks perulangan, metode pengujian *white box* ini akan menguji perulangan sesuai dengan batas perulangan dan mencari kapan suatu proses perulangan diakhiri

- Mendeteksi ketidak sesuaian asumsi

  Menampilkan dan memonitor beberapa asumsi yang diyakini tidak sesuai dengan yang diharapkan untuk selanjutnya akan dianalisa kembali dan kemudian diperbaiki oleh pengembang.
- Dapat mendeteksi kesalahan pengetikan

  Mendeteksi dan mencari bahasa-bahasa pemograman yang dianggap bersifat *case sensitif.*

Tahapan pengujian *white box* adalah:

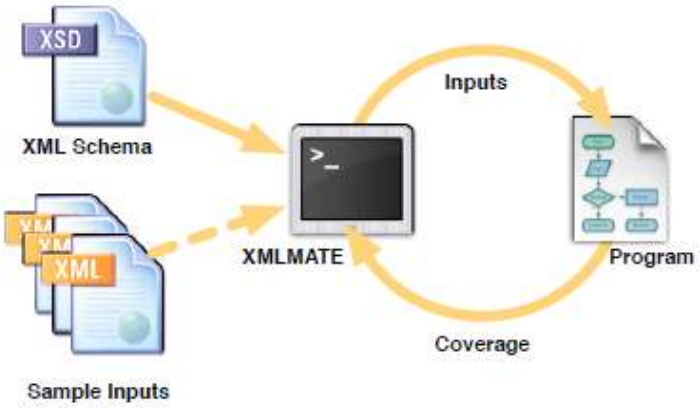
- Pertama memahami fungsi aplikasi melalui kode program. Pada pengujian *white box*, penguji harus memiliki pemahaman mendalam mengenai perangkat lunak yang akan diuji [13].
- Membuat *test case* dan mengeksekusinya [13]. Proses ini menentukan *coverage* dari pengujian. Dalam pengujian, *coverage* merupakan faktor penting. *Coverage* merupakan ukuran kehandalan pengujian yang telah dilakukan.

Pada perangkat lunak yang besar, pengujian *white box* ini dianggap mahal karena melibatkan banyak sumber daya untuk melakukannya [13]. Metode pengujian pada pengujian *white box* ini dilakukan untuk kebutuhan berikut:

- Memberikan dan membuat suatu jaminan bahwa seluruh jalur-jalur yang independen hanya menggunakan modul minimal satu kali.
- Keputusan yang sifatnya logis dapat digunakan disemua kondisi *true* (benar) atau *false* (salah).
- Mengeksekusi seluruh perulangan yang ada ke pada batas nilai dan operasional disetiap situasi dan kondisi.
- Menguji seluruh asumsi yang ada.
- Pengujian secara menyeluruh.

3. XMLMATE

XMLMATE merupakan salah satu *white-box-tool* untuk perangkat lunak berbasis XML. XMLMATE ini dapat membangkitkan populasi *test case* dengan menggunakan algoritma genetik. XMLMATE memanfaatkan struktur program, skema XML, dan sampel masukan XML untuk mengembangkan berbagai *test case* XML. XMLMATE mengembangkan populasi *test case* untuk meningkatkan *coverage* dari pengujian sistem. XMLMATE dikembangkan dalam bahasa pemrograman java dan dibangun diatas framework EvoSuite [1].



Gambar 2 Blok diagram proses pengujian dengan XMLMATE
(Sumber: XMLMate: Evolutionary XML Test Generation [1])

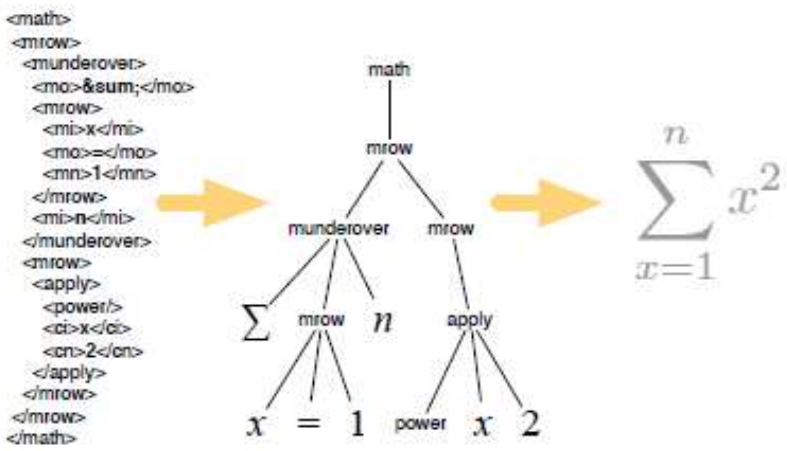
Seperti pada gambar 2, XMLMATE digambarkan dalam 3 proses yaitu

- XMLMATE adalah search-based yang secara sistematik mengembangkan populasi *test case* untuk meningkatkan *coverage*.
- XMLMATE menggunakan skema XML yang membantu menentukan validitas struktur dan sintaksis dari *test case* yang dikembangkan.
- XMLMATE dapat menggunakan sampel masukan XML yang ada, kemudian melakukan mutasi dan mengkombinasikannya kembali untuk mendapatkan *test case* baru.

XMLMATE menggunaan algoritma genetik untuk menghasilkan *test case* secara otomatis. Algoritma genetik ini bertanggung jawab untuk membangkitkan *test case* yang berbeda dengan tujuan meningkatkan *coverage*. Algoritma genetika adalah teknik pencarian heuristik yang didasarkan pada gagasan evolusi seleksi alam dan genetik. Penyelesaian menggunakan algoritma

genetik sangat berpengaruh dari kromosom yang dibangun, dimana kromosom ialah sebuah molekul yang berisi DNA dimana terdapat informasi genetik dalam setiap sel gen yang disimpan.

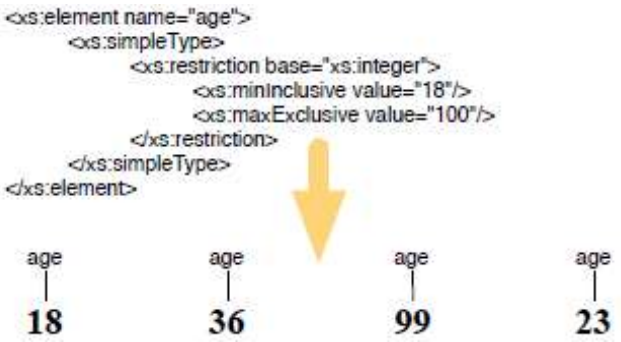
Dalam XMLMATE, struktur data utama adalah sebuah pohon XML. Pohon XML adalah representasi standard dari masukan XML seperti pada gambar 3 yang menggambarkan proses penguraian masukan XML dari bentuk teks kedalam bentuk pohon XML. Pohon XML ini disebut sebagai kromosom C = {x1,…., xn} pada algoritma genetika dimana xi adalah elemen XML yang sesuai dengan skema XML.



Gambar 3 Konversi dokumen XML ke pohon XML
(Sumber: XMLMate: Evolutionary XML Test Generation [1])

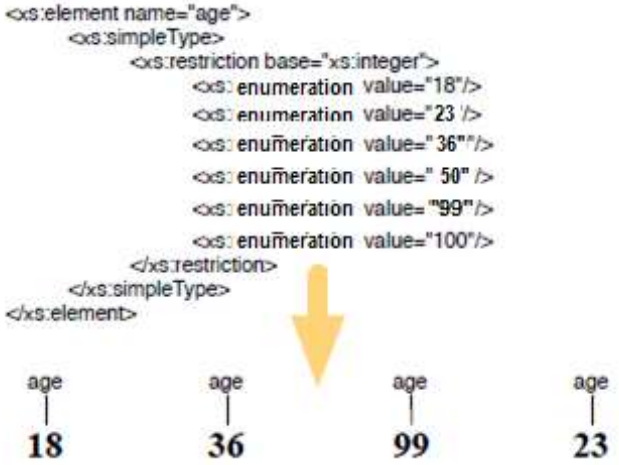
Proses pertama dalam algoritma genetik adalah menentukan populasi *test case* awal. *Test case* awal ini merupakan dasar untuk seluruh operasi mutasi dan perpindahan silang yang akan dilakukan. Jika terdapat sampel masukan XML, maka sampel masukan XML akan digunakan sebagai populasi *test case* awal. Namun jika tidak terdapat sampel masukan, XMLMATE akan membangkitkan populasi *test case* awal secara acak berdasarkan skema XML. Proses instansiasi populasi dilakukan pada elemen yang didefinisikan dalam skema XML. Pertama XMLMATE memeriksa elemen tersebut berdasarkan skema XML. Kemudian dipilih nilai acak yang sesuai dengan kriteria yang didefinisikan pada skema XML dengan aturan sebagai berikut:

- ***Range restriction*** mendefinisikan range angka untuk nilai elemen. XMLMATE memilih sebuah elemen dari range tersebut. Sebagai contoh dapat dilihat pada gambar 4 dimana elemen age didefinisikan berjenis integer dengan range nilai antara 18-100. Kemudian XMLMATE memilih nilai acak sesuai dengan range yang ditentukan.



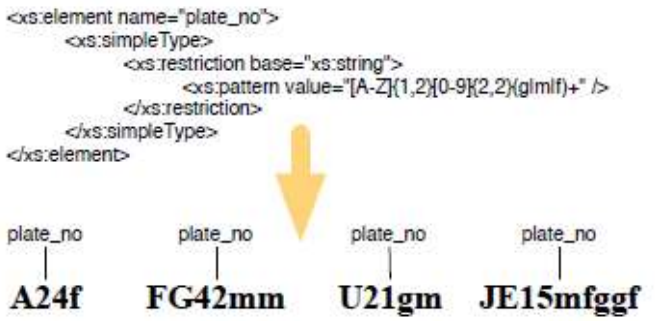
Gambar 4 Range restriction dan contoh nilai elemen yang dipilih
(Sumber: XMLMate: Evolutionary XML Test Generation [1])

- ***Enumeration restriction*** menentukan daftar dari masukan yang valid untuk sebuah elemen. XMLMATE memilih nilai elemen dari daftar seperti pada gambar 5.



Gambar 5 Range restriction dan contoh nilai elemen yang dipilih

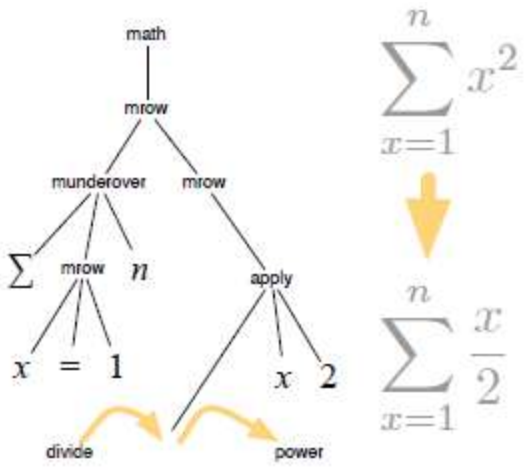
- ***Pattern restriction*** berisi ekspresi reguler yang mendefinisian seluruh masukan yang valid. XMLMATE menggunakan library Automaton untuk menterjemahkan ekspresi reguler tersebut [1]. XMLMATE menerjemahkan ekspresi reguler kedalam sebuah otomata berhingga, kemudian secara acak membangkitkan jalur pada otomata untuk mendapatkan masukan yang valid. Contoh nilai elemen hasil pembangkitan jalur pada otomata berdasarkan ekspresi regular pada skema XML dapat dilihat pada gambar 6.



Gambar 6 Pattern restriction dan contoh nilai elemen yang dipilih
(Sumber: XMLMate: Evolutionary XML Test Generation [1])

- ***Constrain restriction*** dimana nilai elemen adalah sebuah konstanta yang didefinisikan dalam skema.

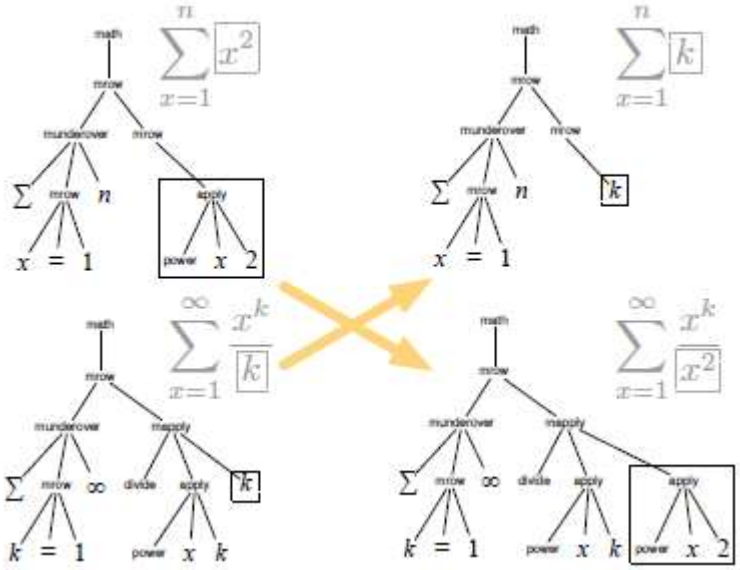
Selanjutnya, operator mutasi melakukan mutasi elemen secara acak sesuai dengan jenis yang didefinisikan pada skema XML. Sebagai contoh pada gambar 7, XMLMATE melakukan mutasi operator pangkat menjadi pembagian pada elemen operator matematik. Baik pembagian atau pangkat merupakan operator matematik.



Gambar 7 Mutasi operasi pada MathML dari pangkat ke pembagi
(Sumber: XMLMate: Evolutionary XML Test Generation [1])

Proses reproduksi pada algoritma genetik tidak hanya melibatkan proses mutasi, tetapi juga melakukan perpindahan silang. Pada XMLMATE, operasi persilangan dimplementasikan dalam 2 level. Pada level kromosom, operator pertuaran silang dapat menukar *test case* XML dalam sebuah populasi. Hal ini tidak mengubah individu *test case* XML. Operasi perpindahan silang lain terletak pada level pohon XML. Operasi perpindahan silang ini menghasilan individu *test case* XML baru yang memiliki struktur yang berbeda dengan individu induknya. Seperti pada gambar 8,

perpindahan silang antara sub pohon (x^2 dan k) dari dua buah pohon XML menghasilkan individu baru yang memiliki struktur pohon yang berbeda. Operator perpindahan silang mempertahankan validitas *test case* XML sesuai dengan skema XML.



Gambar 8 Perpindahan silang x^2 dan k pada MathML
(Sumber: XMLMate: Evolutionary XML Test Generation [1])

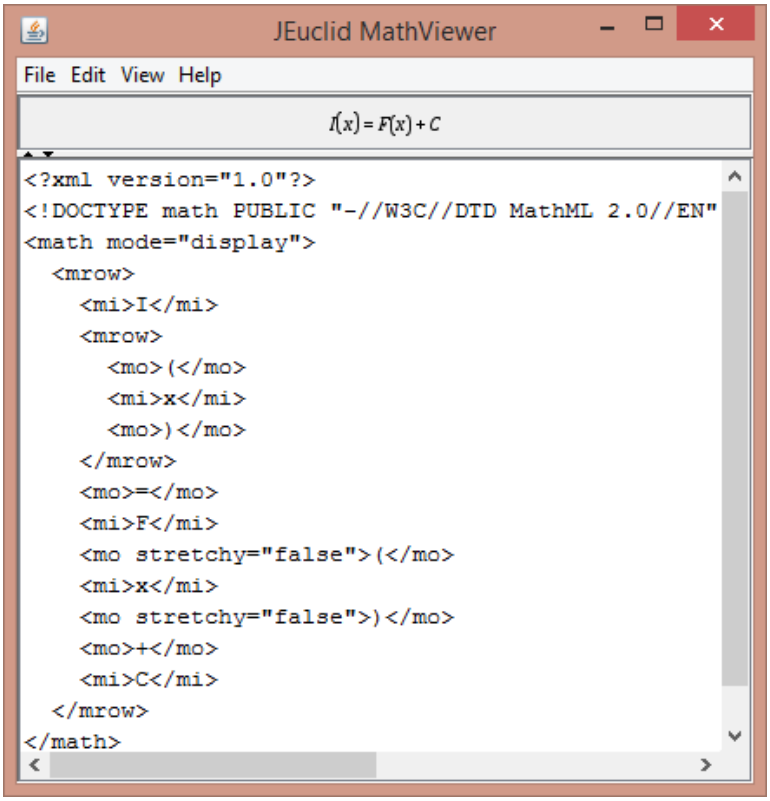
XMLMATE menghitung *branch coverage* yang diperoleh setiap individu *test case* XML sebagai fungsi fitness. Nilai fitness menyatakan seberapa baik nilai dari suatu individu. Nilai fitness digunakan untuk menentukan elemen yang dipilih dari populasi. *Branch coverage* adalah nilai pengukuran seberapa banyak *branch* dalam kode program yang telah dieksekusi (*conditional jump*) [2]. *Branch coverage* dihitung berdasarkan:

- ***Branch distance,*** yaitu menghitung jumlah *branch* yang telah dilalui sampai sebuah *branch* baru ditemukan.
- ***Object distance,*** menghitung seberapa dekat suatu objek yang memenuhi syarat pada sebuah *branch*. Sebagai contoh pada kondisi x>0, maka nilai x=5 lebih dekat dari pada x=100.
- ***Exceptions,*** menghitung jumlah eksepsi.

Setelah dipilih individu *test case* baru yang memiliki nilai fitness terbaik, selanjutnya individu baru dilakukan mutasi dan perpindahan silang pada iterasi berikutnya.

4. Pengujian *White Box* dengan XMLMATE

Selanjutnya akan dilakukan pengujian *white box* sebuah aplikasi berbasis XML dengan menggunakan XMLMATE. Sebagai contoh, dilakukan pengujian *white box* aplikasi JEuclid dengan menggunakan XMLMATE. JEuclid adalah aplikasi untuk pembacaan dokumen MathML. MathML atau Mathematical Markup Language adalah aplikasi dari XML untuk menggambarkan notasi matematis. Antarmuka JEuclid dapat dilihat pada gambar 9 dimana menampilkan notasi matematis dan MathML dari suatu persamaan matematis.



Gambar 9 Tampilan notasi matematis dan MathML pada JEuclid

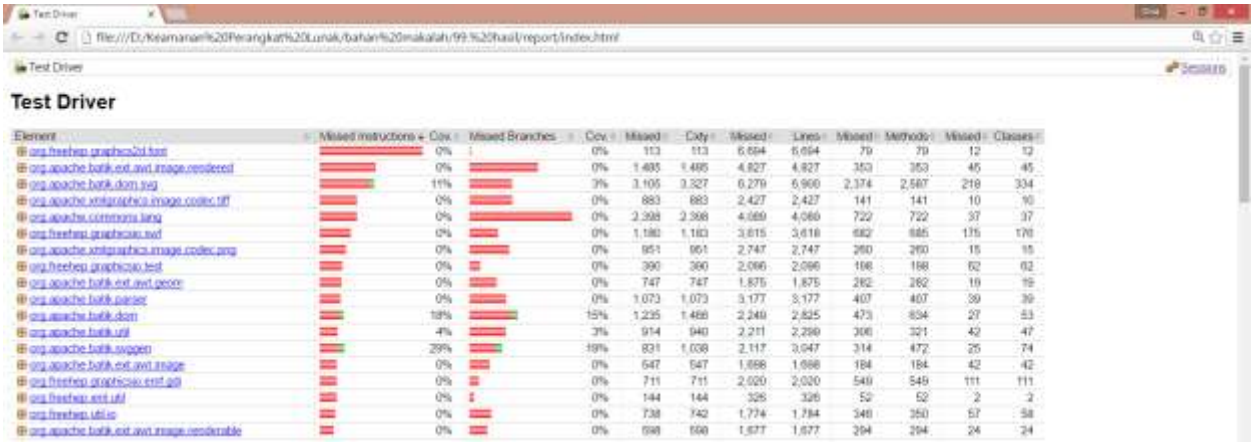
Selanjutnya dilakukan simulasi pengujian *white-box* aplikasi JEuclid dengan menggunakan XMLMATE pada sitem operasi Ubuntu seperti pada gambar 10 dan 11. Selama pengujian XMLMATE membangkitkan *test case* dengan algoritma genetic dan dihasikan 5 buah file *test case*. Hasil pengujian menghasilkan *coverage* 0.042 dengan waktu evolusi selama 89472 milidetik.

Gambar 10 Tampilan awal pengujian JEuclid dengan XMLMATE

Gambar 11 Tampilan akhir pengujian JEuclid dengan XMLMATE

Dalam XMLMATE, tersedia fitur untuk membuat laporan pengujian *white box*. Laporan tersebut dibuat dalam bentuk file html. Selanjutnya dibuat laporan untuk hasil pengujian *white box* aplikasi JEuclid seperti pada gambar 12.

Gambar 12 Tampilan pembuatan laporan hasil pengujian *white box* aplikasi JEuclid dengan XMLMATE



Gambar 13 Tampilan awal laporan pengujian JEuclid dengan XMLMATE

Gambar 14 Tampilan akhir laporan pengujian JEuclid dengan XMLMATE

Dalam laporan hasil pengujian, terdapat informasi modul-modul pada aplikasi JEuclid dan detil pengujiannya meliputi jumlah instruksi, *branch*, baris kode, *method*, dan kelas pada aplikasi yang diuji seperti pada gambar 13 dan 14. Laporan tersebut dibuat dalam bentuk file html. Pada simulasi ini hanya dihasilkan coverage 4%. Namun pada referensi, pengujian *white box* JEuclid dengan beberapa skema menghasilkan coverage 48% [1]. Oleh karena itu dibutuhkan analisis lebih lanjut mengenai penggunaan aplikasi XMLMATE ini.

5. Kesimpulan

XMLMATE merupakan salah satu *white-box-tool* untuk perangkat lunak berbasis XML. XMLMATE ini dapat membangkitkan populasi *test case* dengan menggunakan algoritma genetik. XMLMATE memanfaatkan struktur program, skema XML, dan sampel masukan XML. Dengan menerapkan algoritma genetik, XMLMATE menggunakan sampel masukan XML yang ada atau membangkitkan masukan XML sesuai dengan skema XML, kemudian melakukan mutasi dan mengkombinasikannya kembali untuk mendapatkan *test case* baru. Skema XML yang didefinisikan sangat berpengaruh dalam pembangkitan *test case*. Sehingga dibutuhkan analisis lebih lanjut mengenai penggunaan XMLMATE dan bentuk skema XML dan sampel masukan XML yang sesuai untuk meningkatkan *coverage*.

6. Referensi

[1.] Havrikov Nikolas, Matthias Höschele, Juan Pablo Galeotti, Andreas Zeller. XMLMate: Evolutionary XML Test Generation. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*:719-722, 2014

[2.] Rahardjo, Budi. 2014. Keamanan Perangkat Lunak. Bandung: PT Insan Indonesia

[3.] G. Fraser and A. Arcuri. Whole test suite generation. *IEEE Trans. Softw. Eng*., 39(2):276–291, Feb. 2013.

[4.] XMLMate. https://www.st.cs.uni-saarland.de/testing/xmlmate/

[5.] P. Godefroid, A. Kiezun, and M. Y. Levin. Grammar-based white box fuzzing. *SIGPLAN Not, 43(6):206–215, June 2008*.

[6.] P. Godefroid, M. Y. Levin, D. A. Molnar, et al. Automated white box fuzz testing. *In Network Distributed Security Symposium (NDSS*). Internet Society, 2008.

[7.] S. C. Lee and J. Offutt. Generating *test case*s for XML-based web component interactions using mutation analysis. *In Software Reliability Engineering, 2001. ISSRE 2001. Proceedings. 12th International Symposium on*, pages 200–209. IEEE, 2001.

[8.] Møller. dk.brics. automaton – finite-state automata and regular expressions for Java, 2010. http://www.brics.dk/automaton/.

[9.] J. Offutt and W. Xu. Generating *test case*s for web services using data perturbation. *ACM SIGSOFT Software Engineering Notes*, 29(5):1–10, 2004.

[10.] W. Xu, J. Offutt, and J. Luo. Testing web services by XML perturbation. *In Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium* on, pages 10–pp. IEEE, 2005.

[11.] Gulbransen, David. 2002. Using XML. United States: Que Publishing.

[12.] Erik T. Ray. 2003. Learning XML. United States: O'Reily Media, Inc.

[13.] Mohd. Ehmer Khan - Different Approaches to White Box Testing Technique for Finding error. *International Journal of Software Engineering and Its Applications* Vol.5 No.3, 2011.

[14.] Srinivas Nidhra and Jagruthi Dondeti. Black Box and White Box Testing Techniques – A Literature Review. *International Journal of Embedded Systems and Applications (IJESA)* Vol. 2, No. 2 pp 29-50, June 2012.