Implementasi CNF-SAT Solver Dengan Metode Algoritma DPLL (Davis Putnam Logemann Loveland) Menggunakan AHK Script

A Eganala Filadin Purwoto
Jurusan Informatika, Fakultas Teknologi Industri
Universitas Islam Indonesia
Yogyakarta
13523174@students.uii.ac.idd

***Abstract*— *Satisfiability problem* (*SAT Problem*) adalah suatu problem atau permasalahan dimana sesuatu hal bisa digunakan untuk memberikan ketentuan apakah suatu formula tersebut adalah formula yang memiliki nilai *satisfiabel* atau *unsatisfiabel*. Metode untuk menyelesaikan masalah *Satisfiablity* tersebut antara lain dengan algoritma DPLL. Namun metode DPLL masih akan memakan waktu yang cukup lama apabila *SAT Problem* yang di berikan cukup rumit dan dilakukan secara manual. Oleh karena itu dibuat lah sebuah aplikasi yang dapat menyelesaikan *SAT Problem* secara otomatis yang disebut dengan *SAT Solver*. Pada penelitian ini akan dibuat *SAT Solver* menggunakan bahasa pemrograman AHK (*Auto Hotkey*) *Script*. AHK *Script* merupakan bahasa pemrograman yang ditujukan untuk pembuatan makro. Namun kini AHK *script* juga sudah mampu digunakan untuk pembuatan berbagai jenis aplikasi lainnya. Dengan menggunakan bahasa pemrograman tersebut, penelitian ini berhasil menghasilkan sebuah *SAT Solver*. Walaupun memiliki kekurangan, *SAT Solver* yang dibuat dengan menggunakan bahasa pemrograman AHK Script ini sudah mampu menyelesaikan banyak *SAT Problem* yang diberikan.**

Keywords*— *CNF-SAT Problem; DPLL; Waktu Lama; Pencarian Heuristic

I. PENDAHULUAN

Satisfiability problem (*The Boolean Satisfiability Problem* atau SAT) merupakan suatu masalah yang menentukan kepuasan yang mana berkaitan dengan pengoptimalisasian kegiatan. *Satisfiability problem* (*SAT Problem*) adalah suatu *problem* atau permasalahan dimana sesuatu hal bisa digunakan untuk memberikan ketentuan apakah suatu formula tersebut adalah formula yang memiliki nilai *satisfiabel* atau *unsatisfiabel*. Sebuah formula dikatakan *satisfiabel*apabila terdapat kombinasi nilai - nilai kebenaran variabel yang terkandung didalamnya tersebut bernilai benar (*True*/ (T)) dengan kata lain bila mampu menggambarkan nilai benar-salah untuk suatu variabel sedemikian rupa sehingga menghasilkan pernyataan yang benar (valid), bila tidak valid maka disebut *unsatisfiable* (M. Huth dan M. Ryan,2004). Ada berbagai metode untuk menyelesaikan masalah *Satisfiablity* tersebut, beberapa diantaranya adalah dengan menggunakan algoritma Genetic dan DPLL. Untuk penelitian ini akan digunakan algoritma DPLL. Algoritma DPLL adalah algoritma yang menggunakan pencarian heuristik dilengkapi dengan sistem backtraking dalam pencarian heuristiknya sebagai cara untuk menyelesaikan SAT Problem yang diberikan. Sistem *backtracking* ini di fungsikan untuk menghindari pencarian pada bagian yang sudah pasti bukan merupakan hasil yang *Satisfiable*. Namun walaupun dengan sistem *backtracking* tersebut, masih akan cukup sulit dan memakan banyak waktu untuk menyelesaikan *SAT Problem*yang rumit dan panjang secara manual. Oleh karena itu dibuat lah sebuah aplikasi yang digunakan untuk menyelesaikan *SAT Problem* tersebut yang disebut dengan *SAT Solver*. Dalam penelitian ini dimaksudkan untuk membuat sebuah SAT Solver menggunakan bahasa pemrograman AHK (Auto Hotkey) script.

AHK Script merupakan bahasa pemrograman yang ditujukan untuk pembuatan makro, yaitu sebuah program dimana program tersebut akan menjalankan satu atau lebih perintah hanya dengan sedikit interaksi dari pengguna. Namun kini AHK script juga sudah mampu digunakan untuk pembuatan berbagai jenis program lainnya.

AHK Script memiliki beberapa fitur yang membuatnya unggul dari beberapa bahasa pemrograman yang lain. Salah satu kenunggulannya adalah kemampuannya untuk mengintegrasikan dirinya dalam aplikasi lain yang sedang berjalan, sehingga dapat digunakan untuk membantu kinerja aplikasi tersebut dan juga menggunakan aplikasi yang diintegrasikannya sebagai sarana *input output*. Bukan hanya itu saja, AHK script juga memiliki fitur *dynamic type variable*, dimana sebuah variabel tidak perlu di deklarasikan sebagai sebuah type data spesifik, tetapi variabel tersebut akan mengubah dirinya sendiri sesuai dengan isi dari variabel itu sendiri. Permisalan disini adalah apabila variabel “Var” belum di deklarasikan sebagai tipe data manapun kemudian di isi dengan angka saja, maka secara otomatis type datanya akan berubah menjadi int, double, dsbg. Namun apabila variable tersebut kemudian diisi dengan kata-kata, maka variable tersebut dengan spontan akan mengubah dirinya menjadi tipe data string. Fitur ini juga mampu di tindih dengan cara

mendeklarasikan variabel tersebut sebagai type data tertentu sebelum nya, sehingga kemungkinan terjadinya kesalahan karena fitur ini juga lumayan kecil.

AHK Script di sini juga memiliki *library* yang cukup lengkap, bahkan beberapa perintah memiliki lebih dari satu bentuk untuk mempermudah pengguna dalam melakukan penulisan. AHK script juga mampu membaca dan menggunakan *library* dari aplikasi yang di integrasikannya, walaupun prosesnya cukup rumit untuk dilakukan.

Itulah beberapa fitur-fitur *AHK script* yang lumayan menonjol. Tentu saja bukan hanya itu fitur *AHK script* yang dapat membuatnya dapat bersaing dengan bahasa pemrograman yang lain. Ukurannya yang berada di bawah 20Mb, kompilator yang tidak memakan banyak memory, memiliki fitur "*portable*" sehingga tidak perlu menginstal kembali untuk tiap kali pergantian komputer namun tetap terdapat fitur instal untuk mempermudah pengguna, dapat berjalan pada OS (*Operating System*) lama seperti *windows* XP, dan masih banyak lagi, membuat AHK script cukup popular di kalangan *programmer*, walaupun memang masih banyak yang belum mengetahui keberadaan bahasa pemrograman ini.

II. LANDASAN TEORI

Saat ini sudah ada *paper* mengenai penelitian SAT Solver dan metode DPLL *(Davis-Putnam-Logemann-Loveland)* oleh Taufiq Hidayat & Agung Bahariyanto I, pada tahun 2018 dalam makalah Seminar Nasional Aplikasi Teknologi Informasi (SNATI) mereka menulis tentang *"SAT Solver dengan DPLL dalam pemograman deklaratif"*. Penelitian ini tentang bagaimana pembuatan SAT Solver dengan pemrograman deklaratif dan bahasa Prolog serta menggunakan Algoritma *Davis-Putnam-Logemann-Loveland* (DPLL). Disini Taufik dan Agung menyatakan bahwa penelitiannya ini merupakan bagian dari penelitian tentang sistem *eksplorasi Formal Context* dengan *constraint*. Lebih lanjut Taufiq mengatakan SAT Solver ini lebih menekankan pada analisa logika, bukan matematika yang selama ini banyak berkembang. Saat ini, SAT Solver membutuhkan waktu 30 detik, sedangkan pemecahan dengan program yang berdasarkan matematika hanya satu detik, namun ke depan, kata Taufiq, SAT Solver ini akan lebih cepat karena pemecahannya dibantu komputer. Sedangkan tingkat ketepatan pemecahan masalah, SAT Solver lebih unggul, dimana setelah pengujian akhirnya diketahui bahwa SAT Solver dalam penelitian ini mampu menyelesaikan SAT Problem. Salah satu problem yang diujikan oleh Taufik dan Agung adalah problem Sudoku, dimana SAT Problem dengan 729 variabel dan lebih dari 8829 klausa. (Garuda Ristek Dikti ,2018)

Taufiq Hidayat & Muh. Nizomuddin FS (2018) juga pernah meneliti tentang pembuatan aplikasi yang dapat melakukan konversi formula proposisi ke formula lain yang ekuivalen secara logis dimana pada penelitian ini konversi dilakukan dengan menggunakan hukum-hukum ekuivalensi dan dilakukan secara bertahap. Setiap tahap, aplikasi akan menentukan hukum-hukum yang bisa diterapkan pada formula yang diperoleh pada tahapan ini serta subformula yang akan dikenai hukum tersebut. Selain itu, aplikasi ini juga dapat mengecek apakah sebuah formula sudah dalam bentuk CNF. Dalam implementasinya, aplikasi ini menggunakan tipe data abstrak pohon biner untuk merepresentasikan sebuah formula proposisi. Berdasarkan hasil penelitian tersebut telah berhasil dibangun aplikasi yang mampu melakukan konversi formula proposisi ke bentuk lain yang ekuivalen secara logis. Aplikasi ini menggunakan tipe data abstrak pohon, khususnya pohon biner untuk merepresentasikan formula. Representasi pohon ini memudahkan penentuan hukum-hukum ekuivalensi yang bisa diterapkan terhadap formuladan memudahkan dalam konversi formula dengan menggunakan sebuah hukum ekuivalensi. (Garuda Ristek Dikti ,2018)

Paul Jackson dan Daniel Sheridan pada tahun 2005 juga menulis tentang*"Clause Form Conversions For Boolean Circuits"*.Penelitian ini tentang pengontrolan transmisi arus listrik dalam sirkuit dengan menggunakan CNF dapat mempersingkat waktu yang dibutuhkan.

A. SAT Problem (Boolean Satisfiability Problem)

Dalam ilmu komputer ,Boolean Satisfiability Problem (kadang-kadang disebut satisfisability problem dan disingkat sebagai SATISFIABILITY atau SAT Problem) adalah suatu problem/permasalahan untuk menentukan apakah sebuah kalimat dapat dipenuhi (satisfiable) atau tidak sesuai formula Boolean yang diberikan. Dengan kata lain, ia menanyakan apakah variabel-rumus rumus Boolean yang diberikan dapat secara konsisten digantikan oleh nilai-nilai TRUE atau FALSE sedemikian rupa sehingga rumus tersebut bernilai TRUE .Jika ini kasusnya, rumusnya disebut *satisfiable* atau memuaskan. Di sisi lain, jika tidak seperti itu, fungsi yang dinyatakan oleh rumus adalah FALSE untuk semua variabel dan rumusnya *tidak memuaskan* . Misalnya, rumus " *a* DAN TIDAK *b* " ini dianggap satisfiable atau memuaskan karena seseorang dapat menemukan nilai *a* = TRUE dan *b* = FALSE, yang membuat (*a* DAN NOT *b*) = TRUE. Sebaliknya, " α DAN BUKAN α " dianggap tidak memuaskan(unsatisfiable).

Adapun Rumus logika proposisi disebut ekspresi Boolean yang dibangun dari variabel , operator DAN (konjungsi , juga dilambangkan dengan ∧), ATAU (disjungsi , ∨), BUKAN (negasi , ¬), dan tanda kurung. Rumus dikatakan *satisfiable* jika dapat dibuat BENAR dengan menetapkan nilai logis yang sesuai (yaitu BENAR, SALAH) untuk variabel-variabelnya.*satisfiability* problem Boolean (SAT), diberikan rumus, untuk memeriksa apakah itu *satisfiable*. Masalah keputusan ini sangat penting di berbagai bidang ilmu komputer , termasuk teori ilmu komputer , teori kompleksitas , algoritmik , kriptografi dan kecerdasan buatan.

B. SAT Solver

SAT Solver merupakan suatu perangkat lunak yang digunakan untuk menyelesaikan SAT Problem *(satisfiability problem),* dimana untuk menentukan dari sebuah formula logika proposisi itu *satisfiable* atau *unsatisfiable*. Formula logika proposisi yang diselesaikan dengan SAT Solver harus dalam bentuk CNF *(conjunctive normal form)*. Dalam penelitian yang terdahulu, *SAT Solver* digunakan untuk menyelesaikan eksplorasi atribut *Formal Context* dengan batasan *(constraint)*. (Taufik H; Agung B ,2018).

Langkah penyelesaian problem entailment dengan SAT Solver adalah sebagai berikut:

Problem entailment dinyatakan sebagai SAT *Problem.*

SAT *Problem* disimpan pada sebuah file teks.

SAT Solver dijalankan dengan *input* berupa file teks untuk *entailment problem*.(Jackson, 2005).

C. Davis-Putnam-Logemann-Loveland (DPLL)

Algoritma Davis-Putnam-Logemann-Loveland (DPLL) adalah suatu algoritma yang dikembangkan oleh Martin Davis, Hilary Putnam, George Logemann, dan Donald Loveland, pada tahun 1960-an untuk dapat menyelesaikan masalah SAT Problem dalam bentuk *Conjunctive Normal Form* (CNF). Meskipun algoritma ini sudah sangat lama dikembangkan, namun sampai saat ini banyak pemecah SAT (*SAT Solver*) yang dibuat berdasarkan algoritma ini. Beberapa SAT Solver adalah SAT Solver yang dibuat berdasarkan pengembangan dari Algoritma DPLL. Algoritma DPLL dibuat berbasis aturan. Terdapat 5 aturan pada algoritma DPLL, yaitu *unit propagation, pure literal, decide, fail (unsatisfiable),* dan *backtrack*. Aturan-aturan ini akan diterapkan terhadap formula CNF sampai bisa diputuskan apakah formula tersebut *satisfiabel* atau *unsatisfiabel*. (Taufik.H dan Agung.B, 2018).

D. AHK Script

AHK adalah perangkat lunak yang awalnya dimaksudkan untuk mengubah hotkey kustom ke tindakan yang berbeda tetapi sekarang merupakan suite otomatisasi Windows lengkap.Script adalah teks berisi perintah yang akan di jalankan oleh program dalam hal ini adalah AutoHotkey (autohotkey.exe). Isi script bisa berupa *hotkeys atau hotstrings* bisa juga kombinasi dari keduanya. Script akan menjalankan perintah yang ditulis secara ber-urutan dari atas ke bawah pada saat diaktifkan. **AHK** memiliki konsep cukup sederhana, tetapi merupakan bahasa pemrograman Turing-complete yang lengkap.

Code AHK dapat dibuat pendek atau panjang sesuai kebutuhan,tak ada batasan khusus dalam pembuatan baris-baris code,didalam code juga dapat ditambahkan sebuah komentar dengan terlebih dahulu menambahkan simbol ; (titik koma) di ikuti komentar. Sebuah komentar tak akan ikut tereksekusi bersama baris perintah code lain.

Proses instalasi AutoHotkey sangat mudah.Dengan mengunduh penginstal dari situs web resmi dan pilih "Instalasi Ekspres." Setelah menginstal perangkat lunak tersebut, kemudian klik kanan di mana saja dan pilih New> AutoHotkey Script untuk membuat skrip baru.

III. METODE PENELITIAN

A. Identifikasi Masalah

- Melakukan analisis terhadap permasalahan *CNF-SAT* dan cara penyelesaiannya.
- Melakukan analisis terhadap algoritma DPLL dan implementasinya untuk menyelesaikan permasalahan *CNF-SAT.*
- Menganalisis penggunaan AHK Script untuk menyelesaikan permasalahan *CNF-SAT*menggunakan algoritma DPLL.
- Membaca literatur baik dari jurnal, buku, ataupun penelitian sebelumnya yang berhubungan dengan penyelesaian tugas akhir ini.

B. Analisis Kebutuhan

Analisis kebutuhan apa saja yang diperlukan untuk membuat program dengan AHK Script yang dapat menyelesaikan permasalahan *CNF-SAT* menggunakan algoritma DPLL dilakukan pada tahapan ini.

C. Perancangan

Dilakukan perancangan program pada tahapan ini dengan metode Algoritma DPLL menggunakan AHK Script agar dapat menerima masukan berupa proposisi logika *boolean* dalam bentuk CNF yang dapat menyelesaikan permasalahan *CNF-SAT.*

D. Implementasi

Implementasi dari perancangan yang telah dilakukan pada tahap sebelumnya pada tahapan ini dilakukan. Dengan implementasi algoritma DPLL dalam penyelesaian permasalahan *CNF-SAT* menggunakan AHK Script. Dalam penelitian ini juga akandilakukan implementasi penerapan algoritma DPLL untuk menyelesaikan permasalah *CNF-SAT.*

E. Pengujian

Akhirnya tahapan akhir dari tahapan ini adalah melakukan pengujian dari proses-proses sebelumnya yaitu program yang telah dibuat harus diuji sehingga dapat mengetahui hasil yang sesuai dengan kebutuhan. Dalam penelitian ini, pengujian dilakukan untuk mengetahui apakah aplikasi yang dibuat dapat berjalan dengan lancar. Selain itu tahap ini juga berfungsi untuk melihat kelebihan dan kekurangan aplikasi dalam penerapan penyelesaian permasalahan SAT Problem. Adapun pengujian dilakukan dengan cara penginputan data dummy yang digunakan untuk melihat kemampuan aplikasi.

IV. IMPLEMENTASI DAN PENGUJIAN

A. Impelentasi

Program mengimplementasikan algoritma DPLL menggunakan beberapa fungsi yang bukan hanya saling bergantung pada satu sama lain namun juga di gunakan untuk menyelesaikan SAT problem yang diberikan. Adapun fungsi-fungsi tersebut adalah :

1) Metode Unit Propagnation

Metode ini adalah metode yang berfungsi memberikan nilai *True* terhadap klausa yang memiliki 1 literal dan juga klausa lain yang memiliki literal yang sama dalam klausa tersebut.

```
unitPropagnation(byref Claw){
tempVar := ""
tempArray := object()
loop % Claw.MaxIndex()
{
        tempArray := StrSplit(Claw[A_Index],A_Space)
        i:=0
        loop
        {
                i++
                if (tempArray.MaxIndex() = "" or i > tempArray.MaxIndex())
                        break
                if tempArray[i] = "0"
                {
                        tempArray.removeat(i)
                        i:=0
                }
        }
        if tempArray.MaxIndex() = 1
                if tempArray[1] is integer
                {
                        tempVar := tempArray[1]
                        break
                }
}
if tempVar is integer
{
        if tempVar is integer
                NtempVar := tempVar * -1
        else
                if InStr(tempVar,"-")
                        NtempVar := StrReplace(tempVar,"-")
                else
                        NtempVar := "-" + tempVar

        i:=0
        loop
        {
                i++
                if (Claw.MaxIndex() = "" or i > Claw.MaxIndex())
                        Break

                tempArray := StrSplit(Claw[i],A_Space)
                Claw[i] := StrReplace(Claw[i],NtempVar " ")

                loop % tempArray.MaxIndex()
                        if tempArray[A_Index] = tempVar
                        {
                                Claw.RemoveAt(i)
                                i:=0
                        }
        }

}
return
}
```

2) Metode Pure Literals

Metode ini adalah metode yang berfungsi memberikan nilai *True* terhadap klausa yang memiliki literal tanpa adanya bentuk negasi dari literal tersebut dalam klausa manapun

```
PureLiterals(byref Claw){
tempVar:=object()
NtempVar:=object()
Claw2 :=Claw.Clone()
ExLiter( Claw2 ,tempVar)
loop % tempVar.MaxIndex()
        NtempVar[A_Index]:= tempVar[A_Index] * -1
loop % tempVar.MaxIndex()
{
        tempArray:=object()
        B_Index:= A_Index
        point:=0
        loop % Claw.MaxIndex()
        {
                tempArray := StrSplit(Claw[A_Index],A_Space)
                if existIn(tempArray,tempVar[B_Index]) = 1
                {
                        point:=point + 1
                        break
                }
        }
        loop % Claw.MaxIndex()
        {
                tempArray := StrSplit(Claw[A_Index],A_Space)
                if existIn(tempArray,NtempVar[B_Index]) = 1
                {
                        point:=point + 2
                        break
                }
        }
        i:=0
        if point = 1
                loop
                {
                        i++
                        if (Claw.MaxIndex() = "" or i > Claw.MaxIndex())
                                Break
                        if InStr(Claw[i],tempVar[B_Index]) > 0
                        {
                                Claw.RemoveAt(i)
                                i:=0
                        }
                }
        if point = 2
                loop
                {
                        i++
                        if (Claw.MaxIndex() = "" or i > Claw.MaxIndex())
                                Break
                        if InStr(Claw[i],NtempVar[B_Index]) > 0
                        {
                                Claw.RemoveAt(i)
                                i:=0
                        }
                }
}
return
}
```

3) Metode Decide

Metode ini adalah metode yang berfungsi memberikan nilai *True* terhadap literal secara random.

```
DecidePrep(byref dClaw,byref backside,byref choice){
Claw:=dClaw.Clone()
tempLiter:=object()
choiceVar:=0
ExLiter(Claw,tempLiter)

Random,choiceVar,1,tempLiter.MaxIndex()

backside.push(Claw.Clone())
choice.push(tempLiter[choiceVar])

Decide(Claw,tempLiter[choiceVar])
dClaw:=Claw.Clone()
return
}


Decide(byref Claw,choice,negate:=0){
tempArray:=object()
if negate=1
        choice := choice * -1
Nchoice := choice * -1

i:=0
loop
{
        i++
        if (Claw.MaxIndex() = "" or i > Claw.MaxIndex())
                Break

        tempArray:= StrSplit(Claw[i],A_Space)
        Claw[i] := StrReplace(Claw[i],Nchoice " ")

        loop % tempArray.MaxIndex()
        {

                if tempArray[A_Index] = choice
                {
                        Claw.removeat(i)
                        i:=0
                }
        }
}
return
}
```

4) Metode Fail

Metode Fail merupakan metode yang digunakan untuk melihat apakah pada hasil iterasi penyelesaian permasalahan terdapat klausa kosong/*Fail* atau tidak, dimana nantinya digunakan sebagai basis dibutuhkannya backtracking atau tidak. Metode Fail dalam aplikasi penulis bukan merupakan metode yang berdiri sendiri, melainkan berupa kondisi pengecekan pada metode methodCheck().

```
if (Claw.MaxIndex() != "" and BackLog.MaxIndex() != "")  ;Fail & BackTrack Check
{
        tempArray := object()
        loop % Claw.MaxIndex()
        {
                tempArray := StrSplit(Claw[A_Index],A_Space)
                if tempArray[1] = "0"
                {
                        method := 4
                        break
                }
        }
}
```

5) Metode Backtracking

Metode Backtracking merupakan metode untuk melakukan *Backtrack* dengan memanfaatkan metode Decide untuk membuat *Save Point* yang digunakan untuk melakukan *Backtrack* (penelusuran ulang) apabila terjadi nya kondisi tertentu yang memerlukan *Backtrack*.

```
BackTrack(byref dClaw,byref backside,byref choice){
Claw:=backside.Pop()
Decide(Claw,choice.Pop(),1)
dClaw:=Claw.Clone()
return
}
```

B. Pengujian

Tahap pengujian ini dilakukan untuk mengetahui apakah *SAT Solver* yang dibuat dapat berjalan dengan lancar.Selain itu tahap ini juga berfungsi untuk melihat kelebihan dan kekurangan *SAT Solver* dalam penerapan penyelesaian permasalahan *SAT Problem*. Adapun pengujian dilakukan

dengan cara penginputan data nyata yang digunakan untuk melihat kemampuan *SAT Solver*. Hasil dari pengujian tersebut adalah sebagai berikut :

Tabel 4.1Pengujian Penyelesaian *SAT Problem*

No	Banyak Variabel	Banyak Klausa	Waktu	Benchmark	Hasil
1	1	2	0 detik	Satisfied	Satisfied
2	2	3	0 detik	Satisfied	Satisfied
3	6	6	0 detik	Satisfied	Satisfied
4	3	8	0 detik	Not Satisfied	Not Satisfied
5	3	11	0 detik	Satisfied	Satisfied
6	20	70	2 detik	Satisfied	Satisfied
7	20	90	Unlimited	Satisfied	???

Menurut hasil pengujian pada Tabel 4.1, *SAT Solver* sudah mampu menyelesaikan beberapa masalah yang termasuk golongan mudah (dapat diselesaikan secara manual dengan cepat dan tidak rumit), namun dikarenakan alasan yang belum diketahui *SAT Solver* tidak dapat menyelesaikan soal nomer 7.

Ada beberapa kemungkinan mengapa *SAT Solver* tidak mampu menyelesaikan soal nomer 7. Beberapa dari kemungkinan tersebut termasuk kesalahan logika pada metode dan perintah yang digunakan. Kesalahan logika dapat terjadi akibat cara pengerjaan *SAT Solver* yang berbasis pada iterasi dan bukan pada rekursif, sedangkan metode DPLL ini seharusnya besbasiskan pada rekursif. Adapun kemungkinan lainnya adalah memori yang tidak mencukupi akibat banyak nya data yang digunakan sebagai *backup* yang dimanfaatkan dalam melakukan *backtrack*. Oleh karena itu pengujian pun berlanjut menggunakan data *dummy* yang memiliki beberapa spesifikasi khusus. Hasil dari pengujian lebih lanjut tersebut dapat dilihat pada Tabel 4.2 berikut :

Tabel 4.2Pengujian Lanjutan

No	Banyak Variabel	Banyak Klausa	Spesifikasi Khusus	Waktu	Benchmark	Hasil
	10	20	Berisi formula dengan banyak *backtrack*	2 detik	Not Satisfied	Not Satisfied
2	10	90	Modifikasi soal nomer 7 pada Tabel 4.1 menggunakan lebih sedikit variabel	2 detik	Satisfied	Satisfied
3	40	40	Soal dengan menggunakan banyak variabel dan *decide*, namun sedikit klausa	3 detik	Satisfied	Satisfied
4	12	24	Modifikasi soal nomer 1 dengan lebih banyak variabel, klausa dan *backtrack*	Unlimited/ 2 detik	Not Satisfied	Tidak selesai /Not satisfied

Tabel 4.2 menunjukkan bahwa tidak terjadi kesalahan sintaks maupun logika pada *SAT Solver*. Hal ini dibuktikan dengan kemampuan *SAT Solver* dalam menyelesaikan soal no 1 sampai nomer 4 pada Tabel 4.2.Namun terdapat keganjalan pada *SAT Solver* dimana *SAT Solver* terkadang tidak mampu menyelesaikan soal nomor 4 walaupun ketika *SAT Solver* mampu menyelesaikan soal tersebut, *SAT Solver* sering menggunakan lebih banyak iterasi dan *backtrack* dibandingkan ketika *SAT Solver* tidak sanggup menyelesaikan soal tersebut.

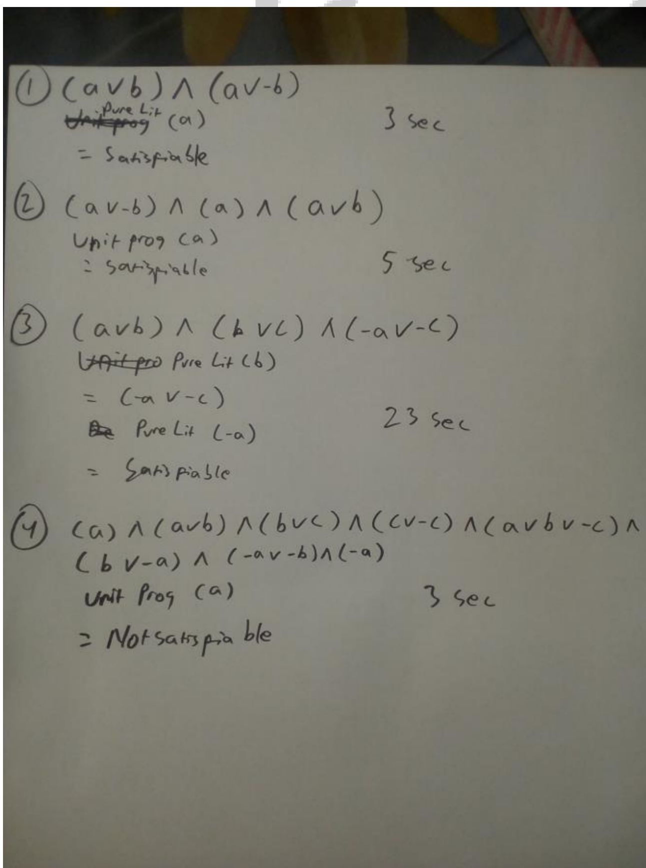
Dari uji coba diatas, dapat disimpulkan bahwa *SAT Solver* tidak mampu menyelesaikan soal nomer 7 pada tabel Tabel 4.1 bukan akibat dari kesalahan logika pada metode yang digunakan, maupun kurangnya memori. Hal ini dibuktikan dengan kemampuan *SAT Solver* dalam menyelesaikan soal nomer 1, 2 dan 3 dengan jawaban yang akurat sebagai bukti tidak adanya kesalahan logika maupun sintaks pada *SAT Solver*, dan soal nomer 3 juga sebagai bukti bahwa *SAT Solver* tidak banyak mengguakan memori ketika menyimpan *backup* yang digunakan untuk *backtrack*.

Adapun hasil komparasi pengujian antara pengerjaan manual dengan menggunakan *SAT Solver* dapat dilihat pada Tabel 4.3 :

Tabel 4.3Pengujian Komparasi *SAT Problem*

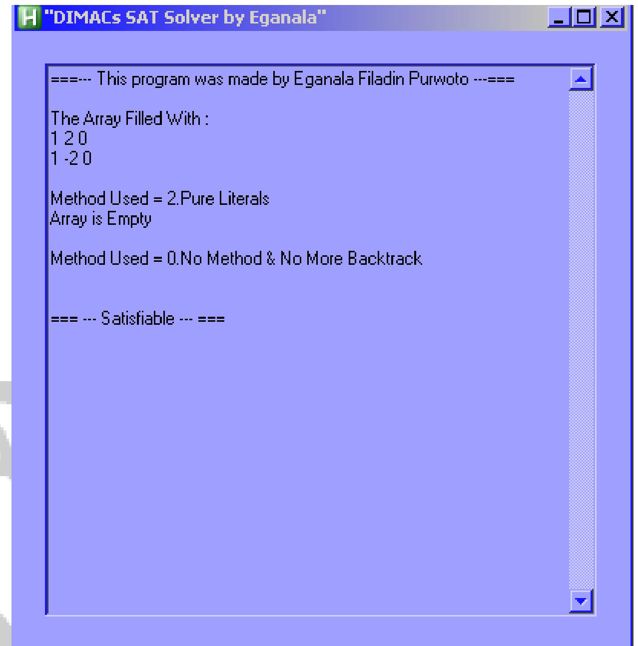
No	Banyak Variabel	Banyak Klausa	Manual		*SAT Solver*		Benchmark
			Hasil	Waktu	Hasil	Waktu	
1	2	2	Satisfied	3 sec	Satisfied	<1 sec	Satisfied
2	2	3	Satisfied	5 sec	Satisfied	<1 sec	Satisfied
3	3	3	Satisfied	23 sec	Satisfied	<1 sec	Satisfied
4	3	8	Not Satisfied	3 sec	Not Satisfied	<1 sec	Not Satisfied

Dan adapun *screenshot* hasil pengerjaan untuk tabel komparasi adalah sebagai berikut :



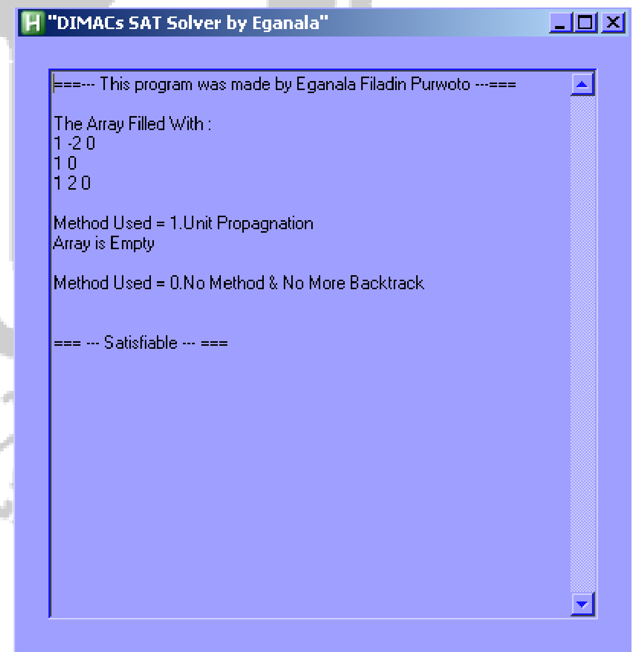
Gambar 4.6*Screenshot* Komparasi Manual

Gambar 4.6 diatas merupakan foto dari soal dan pengerjaan manual yang digunakan dalam tes komparasi. Permasalahan yang sama digunakan dalam penghitungan menggunakan *SAT Solver*.



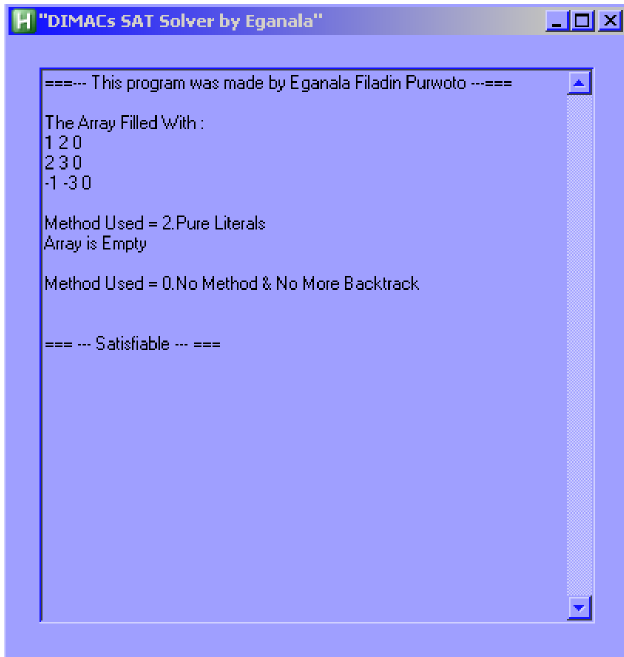
Gambar 4.7*Screenshot* Komparasi *SAT Solver* Soal 1

Gambar 4.7 diatas adalah hasil screenshot dari pengerjaan soal 1 yang sebelum nya dikerjakan manual. Soal di atas dapat diselesaikan hanya menggunakan 1 iterasi saja menggunakan metode nomer 2, yaitu pure literals dengan hasil akhir satisfiable.



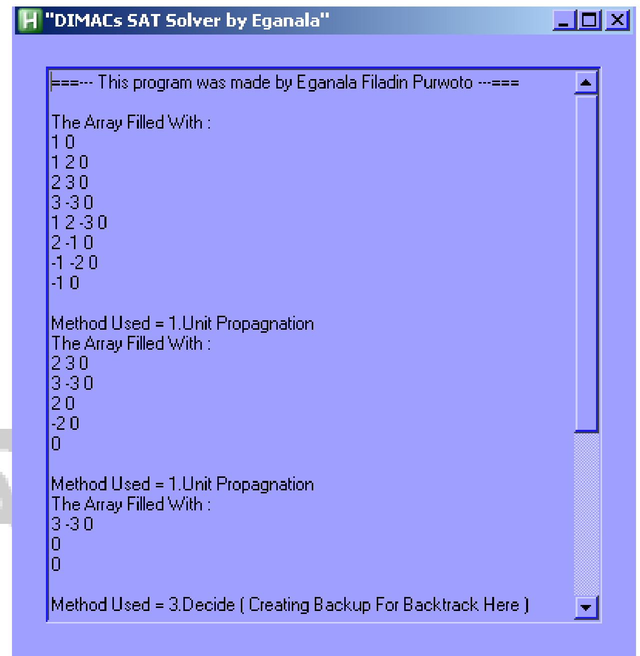
Gambar 4.8*Screenshot* Komparasi *SAT Solver* Soal 2

Gambar 4.8 juga merupakan screenshot hasil pengerjaan *SAT Solver*, namun untuk permasalahan nomer 2, dimana hanya diperlukan 1 iterasi dan 1 metode saja, yaitu metode unit propagation, untuk menyelesaikan permasalahan tersebut dengan hasil akhir *satisfiable*.



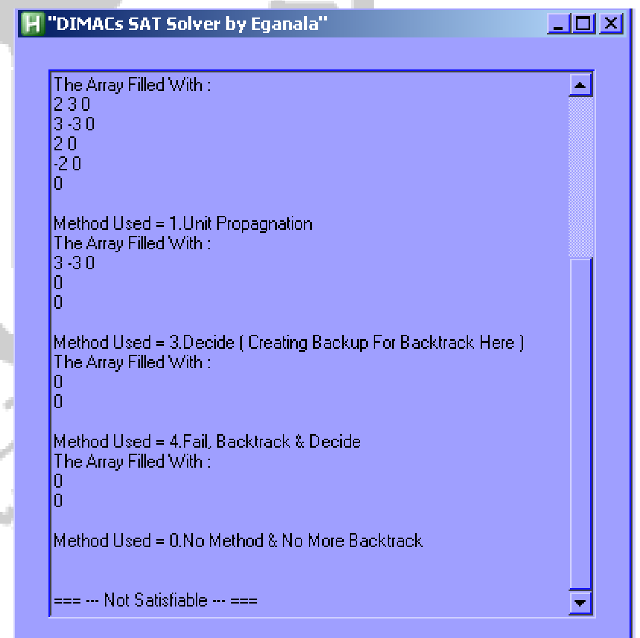
Gambar 4.9 *Screenshot* Komparasi *SAT Solver* Soal 3

Sama seperti gambar Gambar 4.8, Gambar 4.9 juga merupakan screenshot hasil pengerjaan *SAT Solver*, menggunakan permasalahan nomer 3. Sama seperti soal 1 dan 2, di sini permasalahan dapat terselesaikan dengan menggunakan 1 iterasi dan 1 metode yaitu *pure literal* saja dengan hasil akhir *satisfiable*.



Gambar 4.10*Screenshot* Komparasi *SAT Solver* Soal 4-1

Gambar 4.10 juga merupakan screenshot hasil pengerjaan *SAT Solver* untuk nomer 4, namun kali ini sedikit berbeda, dikarenakan soal kali ini membutuhkan lebih dari 1 iterasi dan juga lebih dari 1 jenis metode yang digunakan dalam penyelesaiaanya, yaitu unit propagation dan decide, dimana hasil akhirnya dapat dilihat pada Gambar 4.11.



Gambar 4.11*Screenshot* Komparasi *SAT Solver* Soal 4-2

Gambar 4.11 diatas merupakan kelanjutan dari Gambar 4.10. Berdasarkan Gambar 4.11 dapat disimpulkan bahwa no 4 bukan merupakan formula/permasalahan yang *satisfiable*.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Dari setiap seluruh tahapan yang dilakukan di atas, dapat di simpulkan bahwa dengan menggunakan bahasa pemrograman *AHK Script*, penelitian ini berhasil menghasilkan sebuah SAT Solver yang mampu menyelesaikan permasalahan *SAT Problem* yang diberikan dan juga mempercepat sekaligus mempermudah penyelesaian permasalahan *SAT Problem* tersebut. Walaupun memiliki kekurangan, SAT Solver yang dibuat dengan menggunakan bahasa pemrograman AHK Script ini sudah mampu menyelesaikan banyak SAT Problem yang diberikan. Namun SAT Solver ini hanya dapat digunakan apabila SAT Problem yang diberikan tidak melebihi dari batas kemampuat SAT Solver, yaitu terdiri dari tidak lebih dari 20 variabel dan 70 klausa.

B. Saran

Untuk penelitian berikutnya, ada baiknya menyempurnakan kemampuan *backtrack* program agar dapat menyelesaikan permasalahan dengan lebih banyak lagi variabel dan atau klausa.

DAFTAR PUSTAKA

[1] Boolos,Georges; Burgess,John P; Jeffrey, Richard C (2007). *Computability and Logic* , Fifth Edition , Cambridge,U.K.: CambridgeUniversity Press.

[2] Chiswell,Ian; Hodges, Wilfrid; (2007). *Mathematical Logic,* Oxford University Press.Schoning, Uwe.

[3] Davis, M. dan Putnam, H. (1960). A computing procedure for quantification theory.*Journal of the Association for Computing Machinery*, 7, 201-215.

[4] Davis, Martin; Logemann, George; Loveland, Donald (1962). *A Machine Program for Theorem Proving*. Retrievedfrom Archive. URL:https://archive.org/details/machineprogramfo00davi.

[5] Fröhlich, Andreas & Kovásznai, Gergely & Biere, Armin. (2018). *A DPLL Algorithm for Solving DQBF*. Retrieved from Research Gate, URL:https://www.researchgate.net/publication/266487396_A_DPLL_Algorithm_for_Solving

[6] Garuda Ristek Dikti , (2018) , *SAT SOLVER DENGAN DPLL,* URL : http://garuda.ristekdikti.go.id/journal/view/576?q=SAT Solver denganDPLL.

[7] Garuda Ristek Dikti , (2018) , *Aplikasi Konversi Ekuivalensi Logis Formula Proposisi dengan Pohon Biner*, URL : http://garuda.ristekdikti.go.id/documents/detail.

[8] Jackson,P; Sheridan, D, (2005),*Clause Form Conversions For Boolean Circuits,* Theory and Applications of Satisfiability Testing, H.H. Hoos and D.G. Mitchell (Eds.): 7th International Conference,SAT 2004, LNCS 3542,183-198, Springer-Verlag Berlin Heidelberg

[9] M. Huth dan M. Ryan, (2004),*Logic in Computer Science: Modelling and Reasoning about Systems"*, New York: Cambridge University Press

[10] Nugraheni, C. E. (2008),*Penyelesaian Masalah Penjadwalan Ujian Dengan SAT.* KonferensiNasional Sistem danInformatika

[11]

[12] Purwanto & Sulistyastuti, (2012),*Implementasi Kebijakan Publik Konsep dan Aplikasinya di Indonesia*, Yogyakarta, GAVA MEDIA, 17

[13] Rachmat C,Antonius; (2011).*AlgoritmaPemrogramandenganBahasa C*; Penerbit Andi Yogyakarta

[14] Reddy Septiando, (2011),*Integrated Framework For Business Process Complexity Analysis*. RetrievedfromURL:http://reddyseptiando.blogspot.com/2011/10/aljabar-boolean.htm.

[15] Sedgewick,Robert; Wayne, Kevin; (2011), *Algorithms,*4thed, Princeton UniversityUniversity

[16] Siti Jana Ena, (2010),*Kesimpulan Implementasi Sistem Informasi*, RetrievedfromURL:http://zayna-zayna-cerb.blogspot.com/2010/05/kesimpulan-implementasi sistem.

[17] Taufiq Hidayat ; Irhasni, Agung B (2018), *SAT Solver dengan DPLL dalam Pemrograman Deklaratif*, Jurnal : Seminar Nasional Aplikasi Teknologi Informasi(SNATI)

[18] Taufiq Hidayat; Fauza Sidiq,M.Nizomuddin (2018), *Aplikasi Konversi Ekuivalensi Logis Formula Proposisi dengan Pohon Biner,*Jurnal : Seminar Nasional Aplikasi Teknologi Informasi(SNATI).

[19] Vizel, Yakir; Weissenbacher, Georg; Malik, Sharad (2015). Boolean Satisfiability Solvers and Their Applications in Model Checking, Proceedings of the IEEE,103(SNATI).

[20] Waridah,Ernawati S.S,(2017).*Kamus Bahasa Indonesia*, BMedia,Jakarta.

[21] Whitten ,Jeffrey L;Bentley ,Lonnie D,(2007), *Systems Analysis And Design Methods*, 7th Ed, 689-694,McGraw-Hill Companies. Inc.Americas. New York.

[22] Wink, (2015), *Penemu Algoritma*, RetrievedfromURL:https://www.penemu.co/penemualgoritma-al-khawarizmi

[23] Yodi Arya Ndaru , Pseudocode, (3 Oktober 2011), URL:http://aryandaru.blog.unsoed.ac.id