SAT SOLVER MENGGUNAKAN ALGORITMA DAVIS-PUTNAM-LOGEMANN-LOVELAND

TUGAS AKHIR

Diajukan Sebagai Salah Satu Syarat Untuk Memperoleh Gelar Sarjana

Jurusan Teknik Informatika

Disusun Oleh:

Nama : Yuridi Bintang Pratama

No. Mahasiswa : 12 523 117

JURUSAN TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

YOGYAKARTA

2017

LEMBAR PENGESAHAN PEMBIMBING

SAT SOLVER MENGGUNAKAN ALGORITMA DAVIS-PUTNAM-LOGEMANN-LOVELAND

TUGAS AKHIR

Disusun Oleh:

Nama : Yuridi Bintang Pratama

No. Mahasiswa : 12 523 117

Yogyakarta, Juni 2017

Menyetujui,

Dosen Pembimbing,

(Taufiq Hidayat, S.T., M.C.S.)

HALAMAN PENGESAHAN DOSEN PENGUJI

SAT SOLVER MENGGUNAKAN ALGORITMA DAVIS-PUTNAM-LOGEMANN-LOVELAND

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syaratuntuk memperoleh gelar Sarjana Teknik Informatika
di Fakultas Teknologi Industri Universitas Islam Indonesia
Yogyakarta, 14 Juni 2017

Tim Penguji,

Ketua
Taufiq Hidayat, S.T., M.C.S.

Anggota 1

Arrie Kurniawardhani, S.Si., M.Cs.

Anggota 2

Rahadian Kurniawan, S.Kom., M.Kom.

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Hendrik, S.T., M.Eng.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Saya yang bertanda tangan dibawah ini:

Nama : Yuridi Bintang Pratama

No. Mahasiswa : 12 523 117

Tugas akhir dengan judul:

SAT SOLVER MENGGUNAKAN ALGORITMA DAVIS-PUTNAM-LOGEMANN-LOVELAND

Menyatakan dengan sesungguhnya bahwa seluruh komponen dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya saya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung resiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, Juni 2017

Yuridi Bintang Pratama

HALAMAN PERSEMBAHAN

Alhamdulillahi Robbil 'Alamin puji syukur atas kehadirat Allah SWT yang telah memberikan rahmat, hidayah, petunjuk, dan kemudahan-Nya, sehingga penulis dapat menyelesaikan skripsi dengan judul **"SAT Solver Menggunakan Algoritma Davis-Putnam-Logemann-Loveland"** yang merupakan salah satu syarat untuk memperoleh gelar Sarjana Pendidikan di Universitas Islam Indonesia. Sholawat serta salam semoga selalu tercurahkan kepada Nabi Muhammad SAW sebagai pemberi syafaat kepada seluruh umat manusia.

Kepada keuda orang tua penulis, Bapak Sri Ekanto, S.H. dan Ibu Dra. Rustiyarti Mulyandari.Terimakasih atas kasih sayang, perjuangan, dan kesabarannya dalam mendidik, menjaga, mendoakan, menasehati, mendukung memotivasi, serta mengusahakan segala yang terbaik bagi anaknya. Semoga beberapa prestasi kecil penulis dapat membahagiakan Bapak dan Ibu saat ini. Semoga doa Bapak dan Ibu terus membantu penulis dalam mewujudkan keinginan untuk membahagikan Bapak dan Ibu selanjutnya. Penulis selalu berdoa semoga Allah SWT selalu memberikan kesehatan, rahmat, dan karunia-Nya kepada Bapak dan Ibu.

Terima kasih untuk Bapak Taufiq Hidayat, S.T., MCS., selaku dosen pembimbing yang selalu memotivasi dan totalitas menjalankan peran. Terima kasih atas ilmu dan saran sejak pencarian judul hingga proses pengerjaan skripsi selesai. Penulis mohon maaf atas segala tingkah laku dan perkataan penulis yang tidak berkenan dihati dan kesalahan yang lainnya. Penulis berdoa agar Allah membalas semua kebaikan Bapak, dengan pahala yang berlipat ganda dan semoga Allah juga memudahkan segala urusan Bapak. Aamiin

Terimakasih kepada teman-teman Informatika UII angkatan 2012 tercinta, khususnya Rahmad Mulya, Rizky Arif Windiarto, dan Hendri Agus Setiyanto. Terimakasih atas kebersamaan dan perjuangan dari masa-masa menjadi mahasiswa baru, kebersediaan mendengarkan keluh kesah penulis yang terkadang

tidak penting, perjuangan menggapai nilai kuliah yang memuaskan, keluangan waktu untuk sekedar bercanda gurau. Semoga selalu diberikan kemudahan dan kesusksesan dalam menggapaicita-citanya. Aamiin Ya Rabbalalamin.

Terimakasih kepada Redita Dwi Ciptasari yang telah membantu, memberi semangat, dan segalanya selama tiga tahun ini, dan akan terus lebih dari tiga tahun. Semoga Allah membalas kebaikanmu dan memperlanjar segala urusanku .Aamiin Ya Rabbalalamin.

Kepada teman-teman Unisi Robotic, terima kasih selalu memotivasi dan pengelaman selama lima tahun berurusan dengan robotika, perlombaan dari Semarang, hingga Surabaya. Selalu bersedia menjaga, membantu dan memberikan saran kepada penulis. Semoga kedepannya kalian bisa menjadi juara (lagi). Semoga semuanya berakhir indah, semoga Allah SWT meridhoi.

HALAMAN MOTO

"Hai orang-orang yang beriman, jadikanlah sabar dan shalat sebagai penolongmu, sesungguhnya Allah beserta orang-orang yang sabar."
(QS Al-Baqarah : 135)

"Sesungguhnya sesudah kesulitan itu ada kemudahan".
(Q.S. Alam Nasyrah :6)

"...Barangsiapa bertakwa kepada Allah niscaya Dia akan mengadakan baginya jalan keluar. Dan memberinya rezki dari arah yang tiada disangka-sangkanya. Dan barangsiapa yang bertawakkal kepada Allah niscaya Allah akan mencukupkan (keperluan)nya. Sesungguhnya Allah melaksanakan urusan yang (dikehendaki)Nya. Sesungguhnya Allah telah mengadakan ketentuan bagi tiap-tiap sesuatu".
(At Thalaq 2-3)

"Boleh jadi kamu membenci sesuatu padahal ia amat baik bagimu dan boleh jadi pula kamu menyukai sesuatu padahal ia amat buruk bagimu, Allah mengetahui sedang kamu tidak mengetahui"
(Qs. Al-Baqarah : 216)

"*Man Jadda Wajada – Siapa yang bersungguh-sungguh, ia akan mendapatkan kesuksesan atau keberhasilan*"
(Ulama Salaf)

"*Kebaikan itu tak akan pernah usang, dosa tak akan pernah dilupakan, dan Allah Maha Pembalas tidak akan pernah mati . Lakukanlah apa yang engkau suka . Karena sebagaimana engkau memperlakukan, seperti itulah kau akan diperlakukan*"
(Abu Darda Ra)

KATA PENGANTAR

Assalaamu'alaikum Wr.Wb.

Alhamdulillahi Robbil 'Alamin, puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "SAT Solver Menggunakan Algoritma Davis-Putnam-Logemann-Loveland*"*dengan semaksimal mungkin.

Laporan ini disusun sebagai salah satu persyaratan yang harus dipenuhi dalam rangka menyelesaikan pendidikan pada jenjang Strata 1 di Jurusan Teknik Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia. Tugas akhir ini dapat terselesaikan atas bantuan, dukungan, dan bimbingan yang diberikan dari berbagai pihak, maka dari itu penulis mengucapkan terimakasih kepada:

1. Bapak Nandang Sutrisno, S.H., LL.M., M.Hum., Ph.D., selaku rektor Universitas Islam Indonesia.
2. Bapak Dr. Drs. Imam Djati Widodo, M.Eng, Sc., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
3. Bapak Hendrik, S.T, M.Eng., selaku Ketua Jurusan Teknik Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
4. Bapak Taufiq Hidayat, S.T., MCS. selaku dosen pembimbing tugas akhir yang telah membagi ilmu dan dengan sabar memberikan waktunya membimbing penulis untuk menyelesaikan tugas akhir.
5. Dosen Jurusan Teknik Informatika yang telah membagi ilmunya kepada penulis.
6. Kedua orang tua penulis, Bapak Sri Ekanto dan Ibu Rustiyarti Mulyandari yang telah memberikan dukungan serta dorongan moril maupun materil dalam pembuatan tugas akhir ini.
7. Teman-teman Gravity, Informatika UII 2012, yang telah memberikan semangat dan mendoakan penulis.

8. Teman-teman Unisi Robotic, yang telah memberikan pengelaman yang banyak dan juga semangat.
9. Redita Dwi Ciptasari, yang telah menemani selama tiga tahun ini dan akan berlanjut terus, yang memberikan semangat juga.
10. Semua pihak yang tidak dapat penulis sebutkan satu persatu, terimakasih atas bantuan dan do'anya.

Tugas akhir ini tidak lepas dari kekurangan dan ketidaksempurnaan dikarenakan terbatasnya kemampuan dan pengalaman penulis, oleh karena itu kritik dan saran yang membangun sangat penulis harapkan agar dapat lebih baik lagi. Semoga laporan ini dapat diterima dan bermanfaat bagi para pembacanya. Aamiin.

Wassalaamu'alaikum Wr. Wb.

Yogyakarta, Juni 2017

Yuridi Bintang Pratama

SARI

Pada dasarnya kecerdasan buatan dibuat untuk menyelesaikan suatu masalah, dan salah satu contoh penerapan dari kecerdasan buatan untuk menyelesaikan suatu masalah adalah satisfiability problem. Satisfiability problem atau disingkat SAT problem adalah untuk menyelesaikan dan mencari solusi dari suatu formula logika proposional dengan hasil akhir paling tidak satu kombinasi yang bernilai true. Jika ada satu kombinasi yang bernilai true, maka formula tersebut disebut satisfiable, jika tidak ada satupun kombinasi yang menghasilkan nilai true, maka formula itu bernilai unsatisfiable

Akan mudah bagi manusia jika menyelesaikan satisfiability problem dengan literal dari logika proposional hanya sedikit, tapi jika literalnya ada banyak, maka akan susah diselesaikan menggunakan akal manusia. Oleh karena itu, untuk menyelesaikan satisfiability problem dibuatlah aplikasi satisfiability solver. Satisfiability solver adalah solusi untuk menyelesaikan SAT problem dan memberikan solusi kombinasi literal agar formula logika proposional yang diberikan bernilai paling tidak satu yang bernilai akhir true. Salah satu algoritma untuk aplikasi satisfiability solver adalah menggunakan algoritma Davis Putnam Logemann Loveland, dimana algoritma tersebut menyelesaikan SAT Problem dengan formula conjunctive normal form. Prosedurnya adalah mengeliminasi literal atau klausa yang berada di formula CNF dengan aturan-aturan yang sudah ditetapkan. Jika tidak ada sisa literal atau klausa maka formula tersebut satisfiable, tapi jika ada yang tersisa walau hanya satu maka formula tersebut adalah unsatisfiable.

Berdasarkan hasil pengujian dari sistem Sat Solver menggunakan Algoritma Davis-Putnam-Logemann-Loveland, pengguna dapat mengetahui apakah formula yang diberikan pada sistem satisfiable atau unsatisfiable, dan juga dapat membantu pengguna untuk mencari solusi yang dapat digunakan pada formula tersebut. Kedepannya sistem ini diharapkan dikembangkan agar formula tidak hanya CNF tapi normal form yang lain seperti if-else yang disederhanakan menjadi CNF.

Kata Kunci: CNF, DPLL, SAT Problem, SAT Solver

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

TAKARIR

Edit	Mengubah
Error	Eror / kesalahan
Error handling	Penanganan kesalahan
False	Salah
File	Berkas
Input	Masukan
Interface	Antarmuka
True	Benar
Output	Keluaran
Prototype	*Purwarupa*
Type	Tipe
User	Pengguna

BAB I
PENDAHULUAN

1.1. Latar Belakang

Pada dasarnya kecerdasan buatan dibuat untuk menyelesaikan suatu masalah, dan salah satu contoh penerapan dari kecerdasan buatan untuk menyelesaikan suatu masalah adalah *satisfiability problem*. *Satisfiability problem* atau disingkat SAT *problem* adalah untuk menyelesaikan dan mencari solusi dari suatu formula logika proposional dengan hasil akhir paling tidak satu kombinasi yang bernilai *true*. Jika ada satu kombinasi yang bernilai *true*, maka formula tersebut disebut *satisfiable*, jika tidak ada satupun kombinasi yang menghasilkan nilai *true*, maka formula itu bernilai *unsatisfiable*. Contoh permasalahan logika proposioal untuk mencari *satisfiability* dalam kehidupan sehari-hari adalah untuk menyelesaikan permainan *eight queen puzzle, Einstein logic*, dan yang paling populer adalah game sudoku, yang dimana game sudoku memiliki literal yang banyak.

Akan mudah bagi manusia jika menyelesaikan *satisfiability problem* dengan literal dari logika proposional hanya sedikit, tapi jika literalnya ada banyak, maka akan susah diselesaikan menggunakan akal manusia. Oleh karena ituuntuk menyelesaikan *satisfiability problem* dibuatlah aplikasi *satisfiability solver*. *Satisfiability solver* adalah solusi untuk menyelesaikan SAT *problem* dan memberikan solusi kombinasi literal agar formula logika proposional yang diberikan bernilai paling tidak satu yang bernilai akhir *true*. Salah satu algoritma untuk aplikasi *satisfiability solver* adalah menggunakan algoritma Davis Putnam Logemann Loveland, dimana algoritma tersebut menyelesaikan SAT *Problem* dengan formula *conjunctive normal form*.

Dari latar belakang diatas, maka penulis bermaksud untuk merancang dan membangun *satisfiability solver* dengan menggunakan algoritma Davis Putnam Logemann Loveland, yang diharapkan mempermudah untuk menyelesaikan *satisfiability problem* yang memiliki literal yang banyak.

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan sebelumnya, maka permasalahan yang dapat diangkat adalah bagaimana membuat s*atisfiability solver* yang menggunakan algoritma David Putnam Logemann Loveland (DPLL).

1.3. Batasan Masalah

Terdapat beberapa batasan masalah yang dibuat agar penelitian ini dapat fokus pada masalah yang ingin diselesaikan. Batasan masalah dalam penelitian adalah sebagai berikut:

1. Formula menggunakan *conjunction normal form.*
2. Sistem fokus pada pencarian solusi dari permasalahan yang diberikan.
3. Sistem dibuat berbasis *command line.*

1.4. Tujuan Penelitian

Tujuan dari penelitian ini adalah membangun s*atisfiability solver*untuk menyelesaikan *satisfiability problem* dengan bentuk formula *conjunctive normal form* dengan menggunakan algoritma David Putnam Logemann Loveland (DPLL) dan memberikan solusi kombinasi agar formula yang diberikan bernilai akhir *true.*

1.5. Manfaat Penelitian

Penelitian ini diharapkan memberikan beberapa manfaat yaitu mempermudah yang ingin mencari solusi dari *satisfiability problem* dan menyelesaikan *satisfiability problem*, serta mengetahui apakah formula yang diberikan *satisfiable* atau *unsatisfiable.*

1.6. Metodologi Penelitian

Metodologi penelitian berfungsi sebagai penggambaran langkah-langkah atau metode yang dilakukan dalam penelitian.Berikut merupakan langkah-langkah dari penelitian yang akan dilakukan:

1.6.1. Pengumpulan Literatur

Pengumpulan literaturmerupakan pengumpulan informasi yang bersumber dari dokumen-dokumen terkait baik dokumen tertulis maupun digital, baik

dokumen yang menjelaskan tentang logika matematika maupun*conjunctive normal form, satisfiability problem,* dan *satisfiability solver*. Adapun data-data yang diperoleh melalui penelusuran literatur yang membahas mengenai penerapan algoritma DPLL dan *satisfiability solver*.

1.6.2. Analisis Kebutuhan

Pada tahap ini akan dilakukan identifikasi masalah, kemudian mendefinisikan kebutuhan yang diperlukan sistem yang akan dibuat. Analisis kebutuhan terdiri atas kebutuhan *input* (masukan), kebutuhan proses, dan kebutuhan *output* (keluaran). Selain itu didalam kebutuhan sistem juga terdapat kebutuhan perangkat lunak seperti kebutuhan data, fungsional, dan operasional.

1.6.3. Perancangan

Pembuatan model desain atau rancangan awal konsep berdasarkan informasi kebutuhan yang telah didapatkan, meliputi pembuatan alur proses dari aplikasi dan tampilan awal.

1.6.4. Implementasi

Tahap implementasi dibuat dengan menggunakan acuan rancangan yang telah dibuat sebelumnya.

1.6.5. Evaluasi

Pada tahap ini dilakukan evaluasi konsep dan fungsionalitas sistem yang telah dibangun. Evaluasi ini bertujuan untuk mengetahui kesesuaian sistem terhadap aturan dan langkah kerja dari algoritma DPLL.

1.6.6. Pengujian

Tahap pengujian perangkat lunak merupakan tahapan lanjutan dari kegiatan implementasi perangkat lunak. Pada tahap ini akan dilakukan pengujian terhadap sistem yang telah dibangun sehingga dapat diketahui sistem sudah sesuai dengan kebutuhan dan terlepas dari kesalahan *error*. Pada tahap ini menggunakan metode pengujian *black box testing*. Pada pengujian ini penulis menguji apakah *satisfiability solver* sudah benar atau belum, dan apakah solusi yang diberikan benar atau belum dengan cara mencocokan dengan tabel logika matematika dan solusi yang diberikan oleh program *satisfiability solver*.

1.7. Sistematika Penulisan

Sistematika merupakan suatu penjabaran secara deskriptif tentang hal-hal yang akan ditulis. Sistematika yang digunakan dalam penyusunan laporan penelitian ini adalah sebagai berikut:

- **Bab I Pendahuluan**, bab ini menjelaskan tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian dan sistematika penulisan.
- **Bab II Landasan Teori**, menjelaskan rujukan dan dasar teori yang digunakan dalam membangun sistem.
- **Bab III Analisis Sistem dan Perancangan**, bab ini menjelaskan tentang perancangan umum maupun uraian lebih lanjut mengenai perancangan sistem dalam pembuatan perangkat lunak. Uraian perancangan sistem ini meliputi perancangan data mengenai data *input* dan *output* sistem, perancangan proses mengenai bagaimana sistem akan bekerja dengan proses-proses tertentu, maupun perancangan antar muka dalam desain dan implementasi yang akan digunakan dalam pembuatan tugas akhir ini.
- **Bab IV Implementasi**, menjelaskan tentang kesimpulan dari penelitian yang merupakan rangkuman dari keseluruhan hasil penelitian oleh penulis dan saran dari pihak yang berkepentingan untuk perbaikan dan pengembangan sistem berikutnya.
- **Bab V Penutup**, berisi kesimpulan dari permasalahan dan saran dari beberapa pihak terkait tentang masalah yang dihadapi.

BAB II

LANDASAN TEORI

2.1. Kecerdasan Buatan

Dikutip dari buku *Artificial Intelegence* (Teknik dan Penerapannya) (Dewi, 2006), pengarang memaparkan dengan rincian sebagai berikut:

- Kecerdasan buatan memungkinkan komputer untuk berpikir dengan cara menyederhanakan program. Dengan cara ini, Kecerdasan buatan dapat menirukan proses belajar manusia sehingga informasi baru dapat diserap dan digunakan sebagai acuan di masa-masa mendatang.
- Kecerdasan atau kepandaian itu didapat berdasarkan pengetahuan dan pengalaman, untuk itu agar perangkat lunak yang dikembangkan dapat mempunyai kecerdasan maka perangkat lunak tersebut harus diberi suatu pengetahuan dan kemampuan untuk menalar dari pengetahuan yang telah didapat dalam menemukan solusi atau kesimpulan layaknya seorang pakar dalam bidang tertentu yang bersifat spesifik.
- Kecerdasan Buatan menawarkan media dan uji teori kecerdasan. Teori ini dapat dinyatakan dalam bahasa program komputer dan dibuktikan melalui eksekusinya pada komputer nyata.

2.2. Logika Matematika

Logika (*logic*) berasal dari kata Bahasa Yunani "*logos*" (Dwijono, 2004). Dalam Bahasa Inggris berarti "*word*", "*speech*", atau "*what is spoken*", lebih dekat lagi dengan istilah "*thought*" atau "*reason*". Oleh karena itu definisi logika adalah ilmu pengetahuan yang mempelajari atau berkaitan dengan prinsip-prinsip dari penalaran argunen yang valid. Logika secara umum berhubungan dengan penalaran deduktif (*deductive reasoning*) yang hanya secara umum mengambil kesimpulan dari premis-premisnya. Berbeda dengan penalaran induktif (*inductive reasoning*), yakni *study*tentang pengambilan kesimpulan umum yang diperoleh dari suatu penelitian atau observasi.

2.3. Logika Proposional

Proposisi adalah setiap pernyataan yang hanya memiliki satu nilai benar atau salah, sehingga logika yang menangani atau memproses atau memanipulasi menarik kesimpulan secara logis dari proposisi-proposisi. Logika ini disebut sebagai logika proposional. (Soesianto, 2006). Berikut adalah penjelasan lebih lanjut :

- Proposional dilihat dari bentuk struktur kalimat, suatu pernyataan akan memiliki bentuk susunan minimal terdiri dari subjek diikuti predikat, baru kemudian dapat diikuti dengan objek.
- Ada proposisi-proposisi yang disebut tautologi, yakni proposisi-proposisi yang nilainya selalu benar. Tautologi menghasilkan implikasi secara logis dan ekuivalen secara logis.
- Implikasi logis merupakan dasar dari penalaran yang kuat, sedangkan ekuivalensi logis menunjukan bagaimana proposisi-proposisi dapat dimanipulasi secara aljabar atau secara matematis sehingga disebut logika matematika.

2.4. Tabel Kebenaran

Logika tidak mempermasalahkan arti atau isi suatu pernyataan, tetapi hanya bentuk logika dari pernyataan itu (Soesianto, 2006). Logika hanya menekankan bahwa premis-premis yang benar harus menghasilkan kesimpulan yang benar (*valide*), tetapi bukan kebenaran secata aktual atau kebenaran sehari-hari. Penekanan logika pada penarikan kesimpulan tentang validitas suatu argumen untuk mendapatkan kebenaran yang bersifat abstrak, yang dibangun dengan memakai kaidah-kaidah dasar logika tentang kebenaran dan ketidakbenaran yang menggunakan perangkai logika, yakni "dan (*and*)", "atau (*or*)", "tidak (*not*)", "jika…maka…(*if…then…*)", "…jika dan hanya jika… (…*if and only if*…)". Tabel kebenaran adalah suatu tabel yang menunjukkan secara sistematis satu demi satu nilai-nilai kebenaran sebagai hasil kombinasi dari proposisi-proposisi yang sederhana. Setiap kombinasi dari proposisi-proposisi sederhana tersebut atau

variabel proposisional, nilainya tergantung dari jenis perangkai atau operator yang digunakan untuk mengkombinasikannya. Berikut adalah pemaran lebih lanjut:

2.4.1. Negasi

Negasi digunakan untuk menggantikan perangkai "tidak (*not*)". Negasi berarti hanya kebalikan dari nilai variabel proposisional yang dinegasinya. Jika F akan menjadi T dan sebaliknya, atau negasi F adalah T. Misalkan A adalah proposisi. Pernyataan "ini tidak A" adalah proposisi yang lain, disebut negasi dari A. Negasi dari A diberi simbol ¬A, dan dibaca "tidak A". Berikut Tabel 2.1 tentang tabel kebenaran negasi:

Tabel 2.1 Tabel Kebenaran Negasi

p	¬p
S	B
B	S

Saat mengubah suatu pernyataan menjadi variabel proposisional, setiap pernyataan harus memiliki subjek dan predikatnya masing-masing, dan arti (*meaning*) dari kalimat tersebut tidak dipermasalahkan.

2.4.2. Conjunction (Konjungsi)

Konjungsi adalah kata lain dari perangkai "dan (*and*)", dan konjungsi mempunyai tabel kebenaran seperti berikut:

Tabel 2.2 Tabel Kebenaran Konjungsi

p	q	p ∧ q
B	B	B
B	S	S
S	B	S
S	S	S

Pada Tabel 2.2 menegnai tabel kebenaran konjungsi, hanya ada satu nilai benar, jika pasangan tersebut keduanya bernilai benar, lainya pasti salah. Perangkai "dan" operator ∧ disebut perangkai binari (*binary logical connective*) karena dapat merangkaikan dua variabel proposisional. Misalkan A dan B adalah proposisi. Propisisi "A dan B", yang disimbolkan dengan A∧B, adalah proposisi yang bernilai Benar, jika nilai A dan B keduanya benar, lainya pasti salah. Proposisi berbentuk A∧B, disebut konjungsi A dan B.

2.4.3. Disjunctive(Disjungsi)

Tanda ∨ digunakan sama dengan perangkai "atau (*or*)". Disjungsi (*disjunction*) juga berfungsi sebagai perangkai binari. Lihat Tabel 2.3 mengenai tabel kebenaran disjungsi berikut:

Tabel 2.3 Tabel Kebenaran Disjungsi

p	q	p ∨ q
B	B	B
B	S	B
S	B	B
S	S	S

Misalkan A dan B adalah proposisi. Proposisi "A atau B", yang disimbolkan dengan A∨B, adalah proposisi yang bernilai salah, jika nilai A dan B keduanya salah, jika lainnya pasti benar. Proposisi berbentuk A∨B, disebut disjungsi A dan B.

2.5. Normal Form

Dikutip dari laman *website* (Wikipedia, 2016c) mengenai normal form berikut penjelasnnya lebih rinci:

- Normal form adalah bentuk ekspresi logika yang standar. Bentuk ekspresi logika yang standar adalah bentuk ekspresi logika yang menggunakan operator

dasar yaitu negasi, konjungsi, serta disjungsi. Karena hanya mengandung ($\wedge$, $\vee$, $\neg$) maka bentuk normal dibedakan menajdi 3 yaitu, bentuk Normal Konjungtif ($\wedge$), dan bentuk Normal Disjungtif ($\vee$), dan bentuk Normal Negasi ($\neg$).

- Bentuk Normal Konjungtif atau Conjunctive Normal Form (CNF) adalah bentuk normal yang memakai perangai konjungsi ($\wedge$) dari disjungsi ($\vee$). Bentuknya: A1$\wedge$ A2$\wedge$ … $\wedge$ An dimana Ai berbentuk p1$\vee$ p2$\vee$ … $\vee$ pn.
- Bentuk Normal Disjungtif (Disjunctive Normal Form atau DNF) adalah bentuk normal yang memakai perangkai disjungsi ($\vee$) dari konjungsi ($\wedge$). Bentuknya: A1$\vee$ A2$\vee$ … $\vee$ An di mana Ai berbentuk p1 $\wedge$ p2 $\wedge$ … $\wedge$ pn.

2.5.1. Conjungtive Normal Form (CNF)

Pada *Boolean logic,* formula disebut *conjungtive normal form* atau klausal normal form jika didalam klausa tersebut ada konjungsinya dan didalam konjungsinya ada disjungsinya, lebih mudahnya disebut "atau" didalam "dan" (Wikipedia, 2016a).

Dibawah ini adalah contoh formula dengan variabel A,B,C,D, dan E yang berbentuk CNF:

1. ¬A∧(B∨C)
2. (A∨B)∧(¬B∨C ∨¬D)∧(D∨¬E)

2.5.2. Disjungtive Normal Form (DNF)

Logika formula yang di sebut DNF jika dan hanya jika disjungsi memiliki satu atau lebih konjungsi pada satu atau lebih literal. Suatu formula DNF berisi DNF penuh jika setiap variabel terlihat satu CNF setiap klausa. (Wikipedia, 2016b).

Berikut contoh dari DNF:

1. (A∧¬B∧¬C)∨(¬D∧E ∧F)
2. (A∧B)∨C

2.6. Satisfiability Problem

Boolean satisfiability problem adalah salah satu dari *problem* yang ada di ilmu komputer, yang berasal dari logika matematika. Pada logika proposional, formula akan *satisfiable* jika *variable* yang ada berakir dengannilai *true*. Penting

untuk diketahui bahwa formula tidak selamanya hasil akirnya *true*, atau dengan kata lain formula akan bernilai *false* tidak peduli nilai dari *variable* tersebut. Ini disebut dengan *unsatisfiable*. Contoh *satisfiability problem* di dalam kehidupan sehari-hari adalah *satisfiabilty problem* adalah untuk menyelesaikan permasalahan *game* sudoku yang sudah direpresentasikan dengan bentuk *conjunctive normal form*. (Holldobler, 2009b)

Contoh lain adalah saat kasus untuk mencari hari yang bisa digunakan untuk rapat (Gu, Franco, Wah, & Purdom, 2002). A hanya bisa kumpul pada hari Senin, Rabu, dan Kamis. Si B tidak bisa kumpul pada hari Rabu. Si C tidak bisa kumpul pada hari Jum'at. Dan si D tidak dapat kumpul pada hari Selasa atau Kamis. Jika di representasikan sebagai literal maka:

1 = Senin

2 = Selasa

3 = Rabu

4 = Kamis

5 = Jum’at

Jika direperesentasikan sebagai *conjunctive normal form* adalah:

(1 or 3 or 4) and (-3) and (-5) and (-2 or -4)

Jika dilihat secara biasa maka akan mudah ditentukan yaitu hari Senin yang direpresentasikan dengan *variable* 1. Ini adalah salah satu contoh penerapan *satisfiability problem* pada kehidupan sehari-hari. Dari contoh diatas literal yang diterapkan ada 5 dan *conjunctive normal form*-nya ada empat, maka masih terlihat mudah, jika sudah banyak akan susah untuk dikerjakan oleh manusia.

2.7. Algoritma Davis Putnam Logemann Loveland

Algoritma Davis Putnam Logemann Loveland atau DPLL adalah algoritma untuk menyesuaikan *satisfiability problem*. Algoritma inimerupakan prosedur paling efisien untuk memeriksa *satisfiability* dari klausa. Dengan *input conjunctive normal form* akan diproses untuk memutuskan apakah klausa tersebut *satisfiability* atau tidak. Jika formula yang diberikan *satisfiable* maka akan memunculkan solusi yang dapat digunakan untuk formula yang di*input* tadi.

Prosedur DPLL ini digagas oleh Davis dan Putnam pada tahun 1960, lalu dilanjutkan tahun 1962 oleh Davis, Logemann dan Loveland. (Nieuwenhuis, Oliveras, & Tinelli, 2006)

Ada beberapa prosedur yang harus dilakukan pada algoritma DPLL (Holldobler, 2009a), berikut penjelasan lebih lebih lanjut mengenai prosedur algoritma DPLL:

- Prosedur yang digunakan oleh algoritma ini, pertama adalah memberikan formula dengan bentuk *conjunctive normal form*, lalu dicek apakah *unsatisfiable*atau tidak. *Unsatisfiable* disini adalah apakah formula berisi klausa kosong atau tidak. Jika *unsatisfiable*, maka dilakukan prosedur eliminasi dengan aturan yang sudah dibuat.
- Prosedur eliminasi yang pertama adalah *tautologi*, yaitu dimana formula memiliki literal dan komplemennya pada satu klausa tanpa ada literal yang sama di klausa lain. Contohnya : (A, B, C, D, -A) dan (G, B, C, D). Maka aturan tautologi akan dipakai disini karena A ada komplemennya di satu klausa yang sama, dan tidak ada A lagi di klausa yang lain. Jika aturan ini tidak bisa diterapkan, maka dilanjutkan ke aturan berikutnya.
- Yang kedua adalah prosedur eliminasi pure literal, yaitu dimana formula memiliki satu literal baik dalam klausa utuh atau *single* literal yang tidak ada komplemennya di satu klausa, maupun diklausa yang lain. Contohnya: (A, B, C) dan (-B, -C). Maka aturan *pure* literal bisa digunakan karena A tidak ada komplemennya pada satu klausa dan klausa yang lain. Jika aturan ini tidak bisa diterapkan, maka dilanjutkan ke aturan berikutnya.
- Yang ketiga adalah prosedur eliminasi unit klausa, yaitu dimana formula memiliki klausa yang berbentuk *single* literal, yang memiliki komplemennya di klausa yang lain tapi bukan single literal. Seperti contohnya: (A, B, -C) or (A, -B) or (-A) or (-C) or (D). Maka aturan unit klausa akan digunakan karena -A sebagai klausa tunggal memiliki komplemennya di klausa yang lain. Jika aturan ini tidak tebisa diterapkan, maka dilanjutkan ke aturan berikutnya.
- Yang keempat dan yang terakhir adalah eliminasi split, yaitu dimana formula tidak bisa diterapkan aturan sebelumnya, yang memiliki literal, dan

komplemennya di klausa lain. Lalu memberikan nilai *true* pada salah satu literal tersebut. Lalu jika bernilai *true*, maka hilang satu klausa, dan kalau *false* akan hilang satu literal.

- Jika sudah diterapkan lalu tidak menyisakan klausa atau literal yang tertinggal maka formula yang dimasukan bernilai *satisfiable*, dengan solusi setiap eliminasi yang ada. Tapi jika semua aturan sudah diterapkan tetapi masih tersisa klausa atau literal, maka formula itu bernilai *unsatisfiable* atau tidak ada nilai *true* pada hasil akhirnya.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1. Gambaran Umum Sistem

Aplikasi dapat mencari solusi di setiap *satisfiability problem* dengan menggunakan algoritma Davis Putnam Logemann Loveland, agar pada akhirnya mendapat solusi yang tepat untuk permasalahan tersebut. Aplikasi ini juga diharapkan mengurangi beban kerja pemakai untuk mencari *satisfiability problem*.

3.2. Analisis Kebutuhan

Analisis kebutuhan adalah proses untuk menentukan kebutuhan–kebutuhan yang harus dipenuhi dalam perangkat lunak agar dapat berjalan dengan baik. Analisis kebutuhan meliputi analisis kebutuhan masukan (*input*), analisis kebutuhan proses, analisis kebutuhan keluaran (*output*), dan kebutuhan antarmuka (*interface*).

3.2.1. Kebutuhan Masukan

Analisis kebutuhan *input* merupakan kebutuhan masukan data yang digunakan pada sistem. Berikut kebutuhan *input* yang dibutuhkan oleh sistem:

1. *Input* yang diperlukan untuk kebutuhan ini, yaitu data dari *file text* yang dimana berisi formula dengan beberapa klausa, yang didalam klausa itu terdapat literal.
2. Literal direpresentasikan sebagai angka dan spasi sebagai konjungsi.
3. Diakhiri dengan enter atau ganti baris sebagai tanda disjungsi dan habis satu klausa.
4. Tanda min untuk menunjukkan bahwa literal bernilai ingkaran.

3.2.2. Kebutuhan Proses

Analisis kebutuhan proses merupakan kebutuhan seluruh proses yang dibutuhkan pada sistem. Berikut kebutuhan proses yang akan digunakan pada sistem:

1. *Input* file

   Proses *input* file digunakan untuk memasukan file dan isi file tersebut jika sudah memenuhi syarat yaitu berupa angka, tidak boleh ada nilai nol (0) dan tidak ada huruf di dalam formula yang akan dimasukan nanti, setelah terpenuhi maka diubah menjadi formula.

2. *Unsatisfiable*

   Proses unsat digunakan untuk melihat apakah formula unsat atau tidak.

3. *Satisfiable*

   Proses sat digunakan untuk melihat apakah formula sat atau tidak.

4. Tautologi

   Proses tautologi digunakan untuk melihat apakah formula bisa diterapkan eliminasi tautologi atau tidak.

5. Pure Literal

   Proses cek pure literal digunakan untuk melihat apakah formula bisa diterapkan eliminasi pure literal atau tidak.

6. Unit Klausa

   Proses Proses cek unit klausa digunakan untuk melihat apakah formula bisa diterapkan eliminasi unit klausa atau tidak.

7. Split

   Proses split digunakan setelah semua aturan tidak bisa digunakan, lalu melihat apakah aturan split bisa digunakan atau tidak.

8. Proses pemberian solusi

   Proses pemberian nilai pada literal yang telah dieliminasi, proses ini digunakan setelah formula bernilai *satisfiable*.

3.2.3. Kebutuhan Keluaran Sistem (Output)

1. Informasi aturan mana yang diterapkan pada formula.
2. Menampilkan hasil akhir apakah sat atau unsat.
3. Menampilkan solusi berupa nilai dari literal yang dapat menghasilkan persamaan CNF bernilai akhir *true*saat persamaan tersebut bernilai SAT.

3.3. Perancangan Aturan Algoritma Davis-Putnam-Logemann-Loveland

Aturan algoritma Davis-Putnam-Logemann-Loveland ada enam dan satu aturan tambahan, antara lain:

1. Unsat
2. Sat
3. Tautologi
4. *Pure* Literal
5. Unit Klausa
6. *Split*
7. *Backtracking*

Aturan yang sudah dibuat selanjutnya dikonversi ke notasi matematika sebagai himpunan atau *set,* dan untuk implementasi nanti akan menggunakan tipe data *list*. Berikut ini notasi per-aturan algoritma Davis Putnam Logemann Loveland:

3.3.1. Unsat

Fungsi ini berguna untuk melihat apakah masih ada sisa literal atau klausa dalam formula yang telah di eliminasi atau tidak, atau klausa berhimpunan kosong atau tidak. Notasinya adalah:$\exists$ C .$\exists$ F (C $\in$ F & C = Ø), dibaca ada C, ada F, dimana C adalah elemen dari F, dan C himpunan kosong. Jika kondisi ini terpenuhi maka formula *unsatisfiable*.

3.3.2. Sat

Fungsi ini berguna untuk melihat apakah formula sudah habis tereliminasi atau tidak. Notasinya adalah F = Ø, dibaca, F himpunan kosong. Jika kondisi ini terpenuhi maka formula *Satisfiable*.

3.3.3. Tautology

Aturantautologi yaitu dimana formula memiliki literal dan komplemennya pada satu klausa. Tanpa ada literal yang sama di klausa lain. Contohnya : (A, B, C, D, -A) and (G, B, C, D). Maka aturan tautologi akan dipakai disini karena A ada komplemennya di satu klausa, dan tidak ada A lagi di klausa yang lain, setelah itu A di*copy* pada *variable value* dan klausa yang mengandung A dihapus.

Fungsi ini berguna untuk melihat apakah aturan tautologi bisa digunakan atau tidak, dengan aturan dimana satu klausa ada literal dan komplennya, tanpa ada literal lain di klausa yang lain. Notasinya adalah:∃ C1. (C1 ∈ F & L1, L2 ∈ C1 & L1 = ¬ L2) dibaca ada C1, C1 elemen dari F dan L1 juga L2 adalah elemen dari C1, dan L2 adalah komplemen atau ingkarn dari L1. Jika kondisi ini terpenuhi maka berlaku eliminasi tautologi. Sebagai gambaran: {[L1, . . . , Lm, L, L], C1, . . . , Cm} ≡ {C1, . . . , Cm}. Hal ini dapat dibuktikan dengan aturan tautologi pada hukum logika proposional, dimana ada literal dikonjungsikan, atau diberi operator "atau" dengan komplemennya pasti akan bernilai *true*. Dan aturan *identity of v* (*dominition laws*) jika suatu literal berkonjungsi dengan *true* pasti akan *true*, sebagai gambaran (P v -P v Q v R), dimana P dan -P adalah komplemen. Jika P v *true*≡*true*, sehingga kita tidak usah memikirkan literal yang lain karena klausa sudah pasti bernilai *true*.

3.3.4. Pure Literal

Aturan*pure* literal, yaitu dimana formula memiliki satu literal baik dalam klausa utuh atau *single* literal yang tidak ada komplemennya di satu klausa, maupun diklausa yang lain. Contohnya: (A, B, C) dan (-B, -C). Maka aturan *pure* literal bisa digunakan karena A tidak ada komplemennya pada satu klausa dan klausa yang lain, setelah itu A di*copy* pada *variable value* dan klausa yang mengandung A dihapus.

Fungsi ini berguna untuk melihat apakah aturan pure literal bisa digunakan atau tidak, dengan aturan dimana satu literal tanpa ada komplemennya pada satu klausa maupun klausa yang lainnya. Notasinya adalah ∃ C.∃ C(C ∈ F & L1 ∈ C &∃! -L1 ∈ C), dibaca ada C, C elemen dari F dan L1 adalah elemen dari C, dan tidak ada komplemen atau ingkarn L1 di klausa yang lain. jika kondisi ini terpenuhi maka berlaku eliminasi pure literal. Aturan *pure*literal adalah menyederhakanan klausa tunggal yang tidak ada ingkarannya pada formula. Aturan ini idak memandang apakah itu klausa bernilai *true* atau *false*. Sebagaigambaran (P v Q) dan (Q v R) dan P ≡ (Q v R) karena P diberi nilai *true*, suatu literal dikonjungsikan dengan *true* maka akan *true*, sehingga akan menyisakan klausa yang tidak mengandung literal yang tidak ada komplemennya.

3.3.5. Unit Klausa

Yang ketiga adalah unit klausa, yaitu dimana formula memiliki klausa yang berbentuk *single* literal, yang memiliki komplemennya di klausa yang lain tapi bukan *single* literal. Seperti contohnya: (A, B, -C) dan (A, -B) dan (-A) dan (-C) or (D). Maka aturan unit klausa akan digunakan karena -A sebagai klausa tunggal memiliki komplemennya di klausa yang lain, setelah itu A yang di *single* klausa di*copy* ke *variable value* dan dihapus klausanya, dan A yang komplemennya, dihapus literalnya.

Fungsi ini berguna untuk melihat apakah aturan unit klausa bisa digunakan atau tidak, dengan aturan dimana ada satu literal yang literal itu adalah *single* literal, yang artinya klausa hanya ada satu literal, dengan komplemen atau ingkaran dari literal ada di klausa yang lain. Notasinya adalah ∃ C1 .∃C2(C1,C2 ∈ F & [L1] ∈ C1 & L2 ∈ C2 & L1 = ¬L2),dibaca ada C1, ada C2, C1 C2 adalah elemen dari F dan L1 adalah *single* literal yang elemen dari L1, dan L2 elemen dari C2 dan L2 adalah ingkaran L1. Unit klausa adalah aturan ketiga yang harus dicek saat penyederhanaan formula di algoritma DPLL. Dalam aturan ini dilakukanpenghapusan ingkaran dari klausa yang *single* literal yang terdapat pada klausa yang lain. Sebagai gambaran {[L1, . . . , Lm, L], [L] ≠ [L1, . . . , Lm], [L]}. Pada proses unit klausa, penulis memberikan perbedaan yaitu jika unit dilakukan diawal, maka literal yang *single* literal, akan di*copy* ke *value,* tanpa mengeliminasi klausa tersebut, karena untuk menunjukan bahwa literal tereliminasi itu bukan hasil dari *split*melainkan hasil dari eliminasi unit klausa.

3.3.6. Split

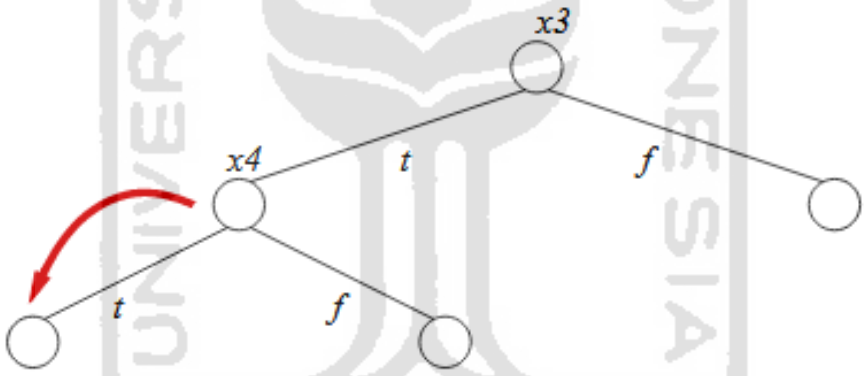
Jika aturan semua tidak bisa dijalankan, maka lakukan aturan *split*, yaitu dimana formula tidak bisa diterapkan aturan sebelumnya yang memiliki literal dan komplemennya di klausa lain. Lalu memberikan nilai *true* pada salah satu literal tersebut. Lalu jika bernilai *true*, maka hilang satu klausa dan jika*false* akan hilang satu literal.

Fungsi ini berguna untuk melihat apakah aturan *split* bisa digunakan atau tidak, dengan aturan dimana ada literal pada suatu klausa dengan komplemen atau ingkaran dari literal ada di klausa yang lain. Notasinya adalah:∃ C1 . ∃ C2 (C1,C2

∈ F & L1 ∈ C1 & L2 ∈ C2 & L1 = ¬L2), dibaca ada C1, ada C2, C1 C2 adalah elemen dari F dan L1 elemen dari C1, dan L2 elemen dari C2 dan L2 adalah komplemen atau ingkaran L1. Khusus untuk *split* akan menaruh salah satu literalnya bernilai *true*, dan komplemennya *false*. Lalu merekursif atau memanggil lagi aturan Tautologi, *Pure* Literal, Unit Klausa, dan *Split* itu sendiri. Jika bernilai unsat maka melakukan fungsi *backtrack*.

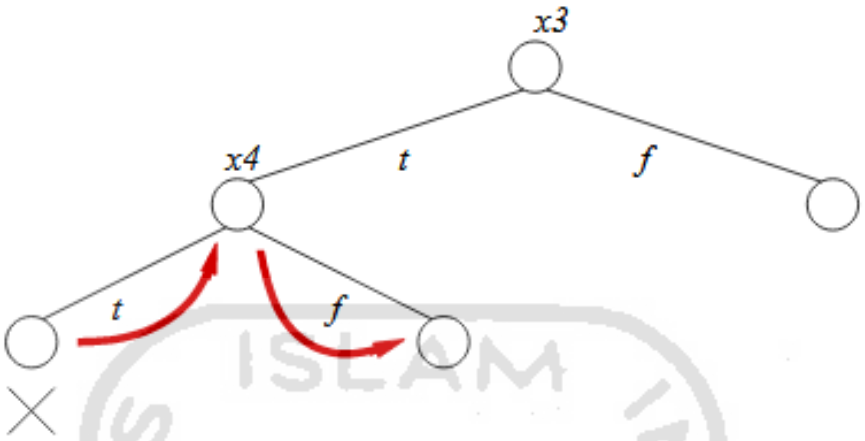
3.3.7. Backtrack

Fungsi ini digunakan saat split yang bernilai unsat, yang jika L1 yang diberikan nilai *true* dan L2 *false*, maka jika di*backtrack* nilai dari L1 dan L2 dirubah menjadi L1 *false* L2 *true*. Lalu merekursif lagi aturan sebelumnya, yaitu tautologi, *pure* literal, unit klausa, dan *split* itu sendiri.



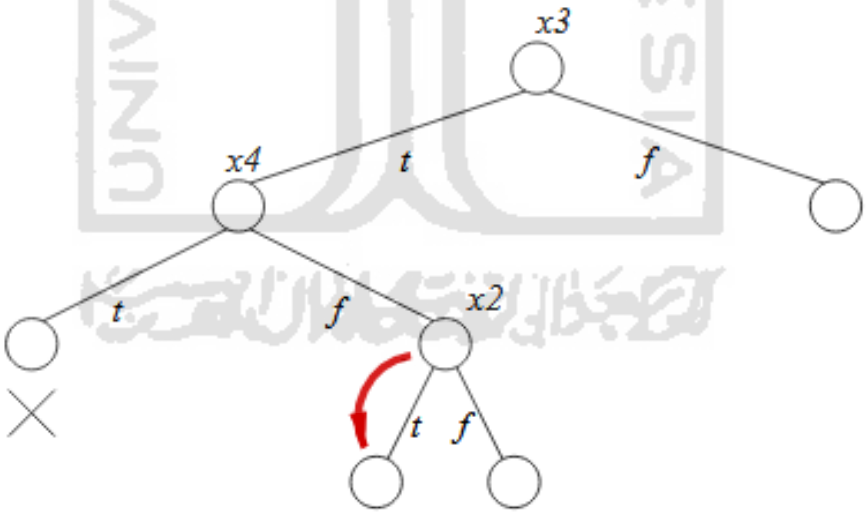
Gambar 3.1 Pemilihan *Split*

Gambar 3.1 menunjukan cara *split*, yaitu memilih salah satu literal lalu diberi nilai *true*. Jika nanti *unsatisfiable*, maka akan *backtrack* dan memberikan nilai *false* pada literal yang sebelumnya, seperti pada Gambar 3.2 dibawah ini:



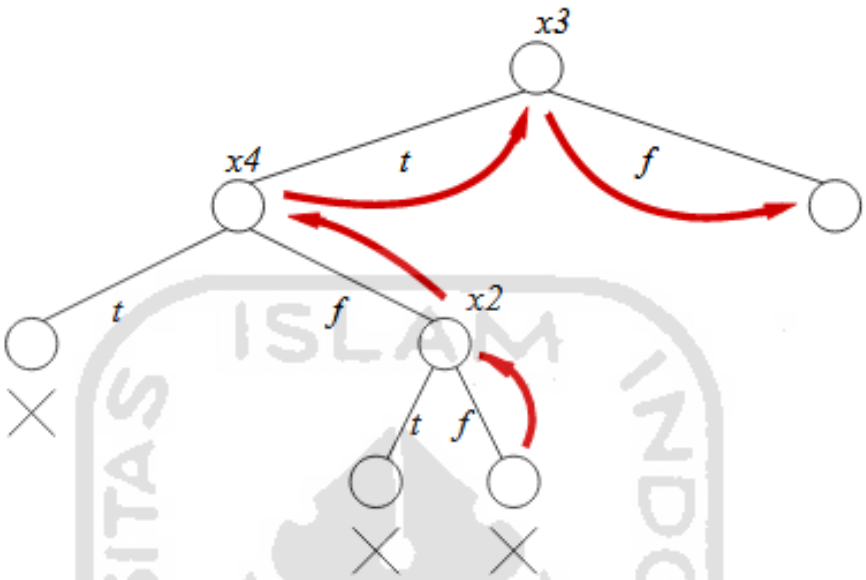
Gambar 3.2 *Backtrack*yang Pertama

Jika diteruskan ada aturan split dapat dilakukan maka lakukan split lagi seperti Gambar 3.3. Seperti langkah yang sebelumnya, literal diberikan nilai true, jika unsatisfiable, maka backtrack dan mengganti nilainya.



Gambar 3.3 *Split* yang Kedua

Seperti Gambar 3.4 dibawah , jika *unsatisfiable*, maka *backtrack* kembali ke atas, jika literal sudah diberikan nilai *false* maka kembali lagi ke atas jika memungkinkan.

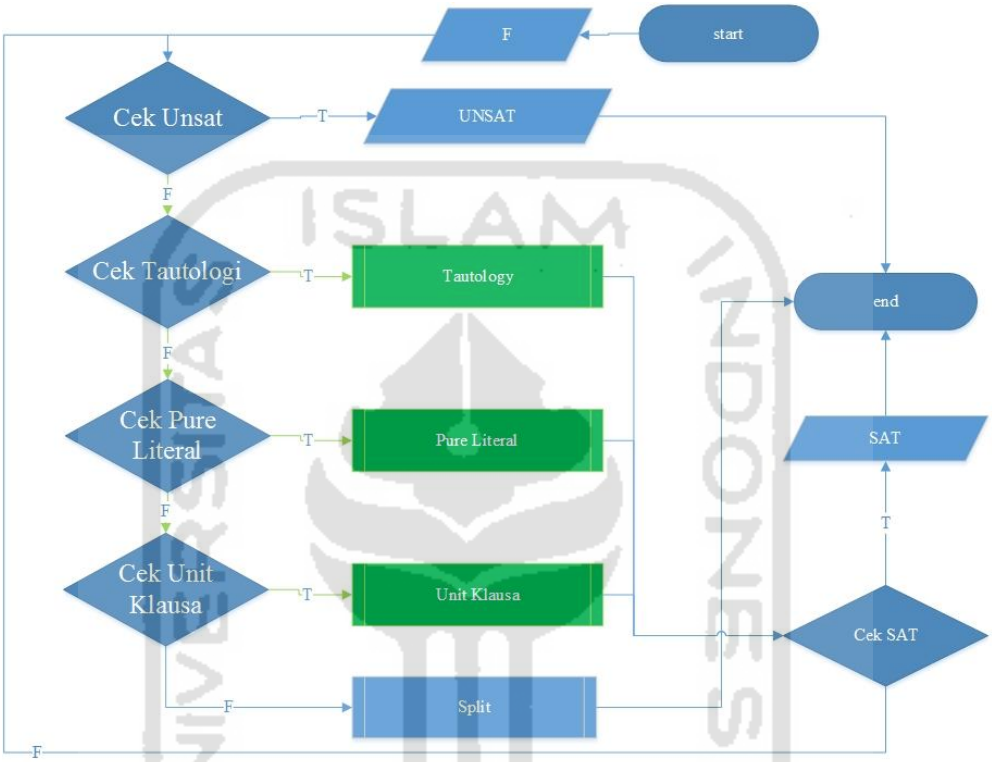


Gambar 3.4 *Backtrack* yang Kedua

3.4. Perancangan Representasi Algoritma DPLL

Algoritma DPLL berjalan dengan memilih literal, menempatkan nilai kebenaran. Untuk itu, menyederhanakan rumus dengan beberapa rumus aturan kemudian secara rekursif memeriksa apakah rumus disederhanakan *satisfiable*, jika hal ini terjadi maka formula yang diberikan tadi *satisfiability* dan tidak menutup kemungkinan untuk kembali menggunakan rumus yang ada kembali. jika tidak *satisfiability*, cek rekursif yang sama yang dilakukan tadi dengan asumsi nilai kebenaran yang berlawanan. Hal ini dikenal sebagai aturan *Spliting*, karena membagi klausa menjadi dua sederhana sub-klausa. Langkah penyederhanaan dasarnya menghapus semua klausa dan literal yang sudah pasti benar pada formula yang diberikan. *Unsatisfiability* dari formula yang diberikan terdeteksi jika salah satu klausa menjadi kosong, yaitu jika semua variabel yang telah ditetapkan dengan cara yang membuat literal sesuai palsu. *Satisfiability* terdeteksi baik ketika semua variabel ditugaskan tanpa menghasilkan klausa kosong atau dalam implementasinya hasil akhir formulanya adalah *true*. *Unsatisfiability* dari rumus formula hanya dapat dideteksi setelah pencarian yang panjang.

Pada Gambar 3.5 merupakan gambar dari representasi aturan yang harus dijalankan oleh sistem untuk mengeliminasi literal dan klausa, sehingga mendapatkan nilai sat atau unsat. Jika sat, akan memunculkan nilai solusi untuk formula tersebut.



Gambar 3.5 Algoritma DPLL

Sistem menerima masukan formula dengan komposisi klausa-klausa dan dimana klausa memiliki literal, jika dibuat notasinya F = {C1, ... Ck} dimana Ci = L1 v L2 v .... Ln.

Prosedur algoritma ini meliputi yang pertama adalah memberikan formula dengan *inputconjunctive normal form* akan diproses untuk memutuskan apakah klausa tersebut *satisfiability* atau tidak, dengan ketentuan tidak ada angka nol (0) dan tidak ada huruf di formula yang telah diberikan tadi.

Setelah mendapatkan formula yang benar, lalu dicek apakah ada klausa yang kosong tidak ada isinya. Jika ada maka bisa dipastikan unsat. Apabila dicek tidak ada klausa kosong, lalu cek apakah bisa menggunakan aturan *pure* literal,

jika bisa dilakukan maka literal akan dieliminasi, lalu dicek *satisfiability* atau tidak.

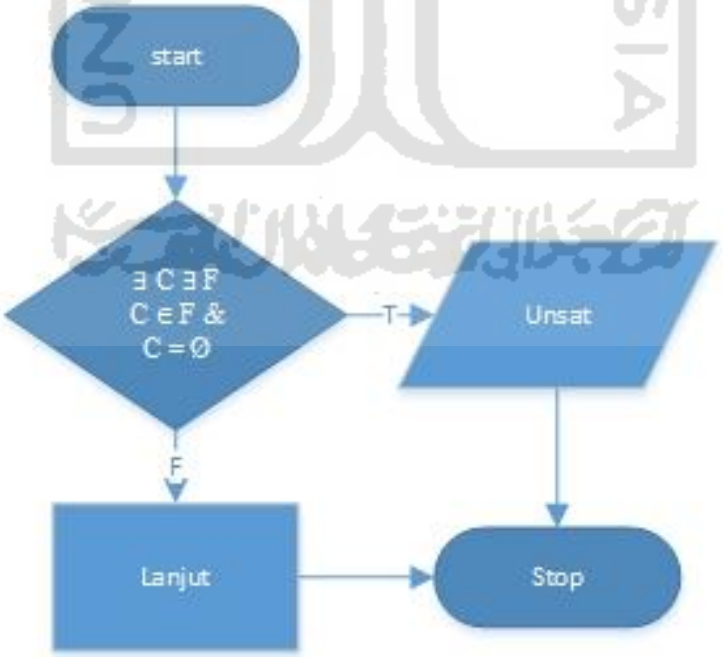
Jika tidak bisa menggunakan aturan *pure* literal, dicek apakah bisa menggunakan aturan unit klausa,jika bisa dilakukan maka literal yang akan dieliminasi, lalu dicek *satisfiability* atau tidak. Jika tidak bisa menggunakan aturan unit klausa maka lakukan aturan *split*.

Dicek apakah bisa menggunakan aturan *split*, jika bisa maka ditambahkan literal yang akan dieliminasi kedalam *value*, lalu eliminasi. Jika tidak bisa lakukan *backtracking*, yang tadinya bernilai literal bernilai *true* dan komplemennya bernilai *false* lalu dirubah nilainya.

Aturan akan berhenti jika formula tidak bisa lagi dilakukan. Jika kondisinya F={} atau [] di mana [] adalah klausa kosong.Jika F={} maka formula yang diberikan *satisfiability*, dan kita bisa berhenti. Jika C={} maka formula yang diberikan *unsatisfiability*.

3.4.1. Proses Pengecekan Unsat

Pada Gambar 3.6 dibawah ini menunjukan pengecekan unsat, dimana jika klausa berhimpunan kosong di formula:

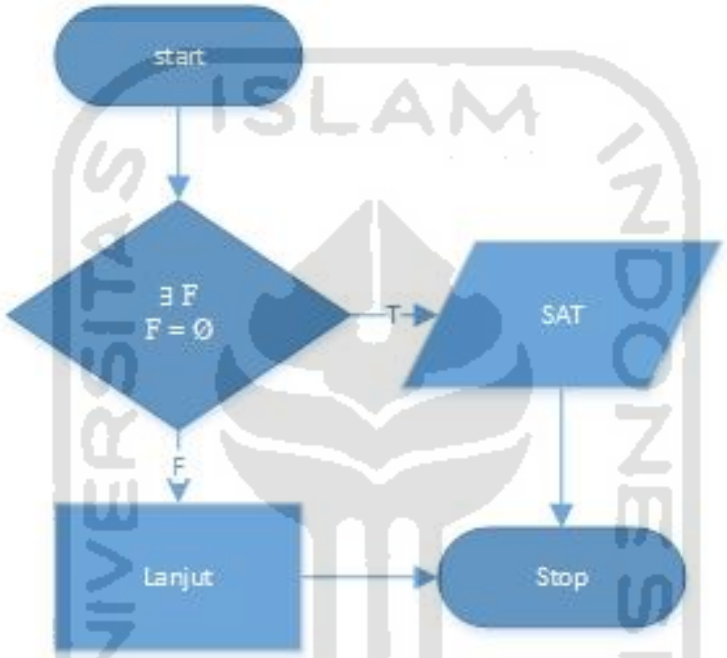


Gambar 3.6 Pengcekan Unsat

Fungsi ini berguna untuk melihat apakah masih ada sisa literal atau klausa dalam formula yang telah dieliminasi atau tidak, atau klausa berhimpunan kosong atau tidak. Fungsi ini juga dijalankan saat diberi formula pertama kali.

3.4.2. Proses Penegcekan Sat

Pada Gambar 3.7 dibawah menunjukan pengecekan sat, dimana F adalah himpunan kosong.

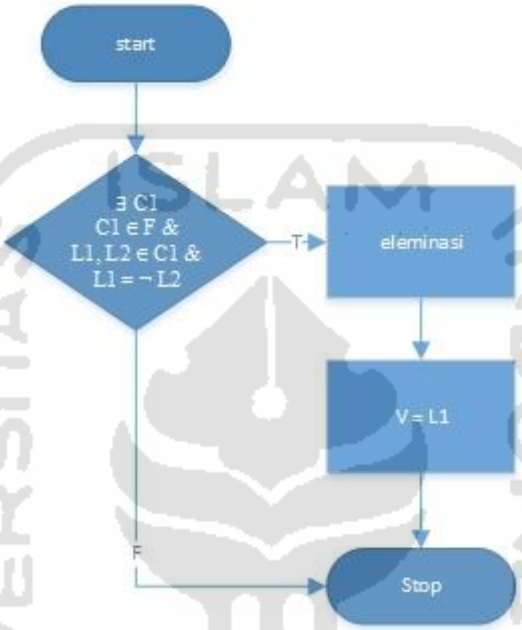


Gambar 3.7 Pengecekan SAT

Jika bukan himpunan kosong maka lanjut eliminasi.Fungsi ini berguna untuk melihat apakah formula sudah habis tereliminasi atau tidak. Notasinya adalah F = Ø, dibaca, F himpunan kosong. Jika kondisi ini terpenuhi maka formula *satisfiable*. Jika belum makan dilanjutkan ke eliminasi selanjutnya, jika sudah diterapkan lalu lalu dicek kembali apakah masih menyisakan klausa atau literal tidaknya. Jika tidak ada klausa atau literal yang tertinggal maka formula yang dimasukan bernilai *satisfiable*, dengan solusi setiap eliminasi yang ada.

3.4.3. Proses Pengecekan Tautologi

Pada Gambar 3.8 menunjukan pengecekan tautologi, dimana suatu klausa memiliki literal dan komplemennya dalam satu klausa Jika memenuhi maka eliminasi klausa.

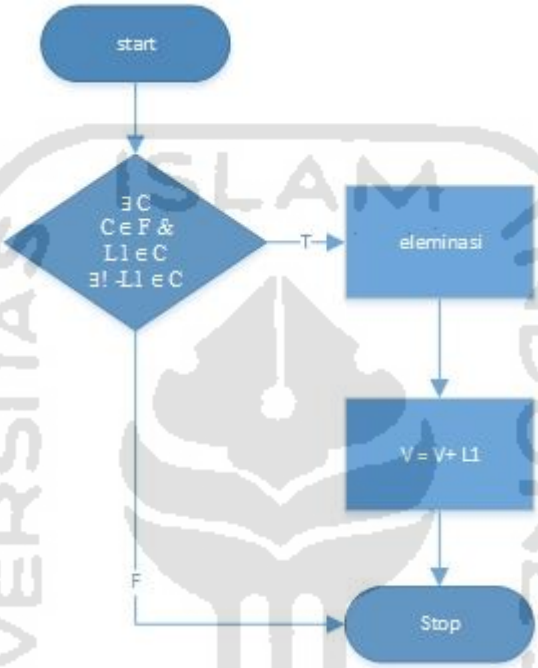


Gambar 3.8 Pengecekan Tautologi

Fungsi ini berguna untuk melihat apakah aturan tautologi bisa digunakan atau tidak, dengan aturan dimana satu klausa ada literal dan komplennyatanpa ada literal lain di klausa yang lain. Notasinya adalah:∃ C1. (C1 ∈ F & L1, L2 ∈ C1 & L1 = ¬ L2) dibaca ada C1, C1 elemen dari F dan L1 juga L2 adalah elemen dari C1, dan L2 adalah komplemen atau ingkaran dari L1. Jika kondisi ini terpenuhi maka berlaku eliminasi tautologi dan menambahkan literal ke *variablevalue.*

3.4.4. Proses Pengecekan Pure Literal

Pada Gambar 3.9 dibawah menunjukan pengecekan pure literal, dimana ada literal di satu klausa tanpa ada komplmennya baik di satu klausa maupun di klausa yang lain.

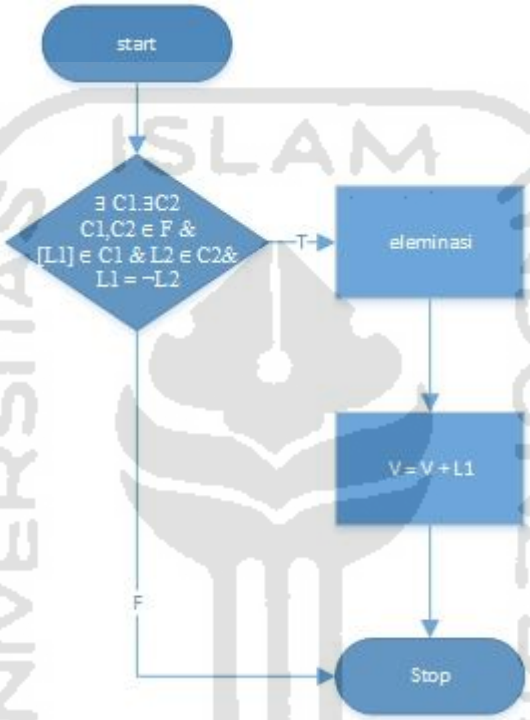


Gambar 3.9 Pengecekan Pure Literal

Fungsi ini berguna untuk melihat apakah aturan *pure* literal bisa digunakan atau tidak, dengan aturan dimana satu literal tanpa ada komplemennya pada satu klausa maupun klausa yang lainnya. Notasinya adalah ∃ C.∃ C (C ∈ F & L1 ∈ C &∃! -L1 ∈ C), dibaca ada C, C elemen dari F dan L1 adalah elemen dari C, dan tidak ada komplemen atau ingkaran L1 di klausa yang lain Jika kondisi ini terpenuhi maka berlaku eliminasi *pure* literal.

3.4.5. Proses Pengecekan Unit Klausa

Pada Gambar 3.10 dibawah menunjukan pengecekan unit klausa, dimana suatu klausa memiliki yang isinya *single* literal dan memiliki komplemennya di klausa yang lain klausa

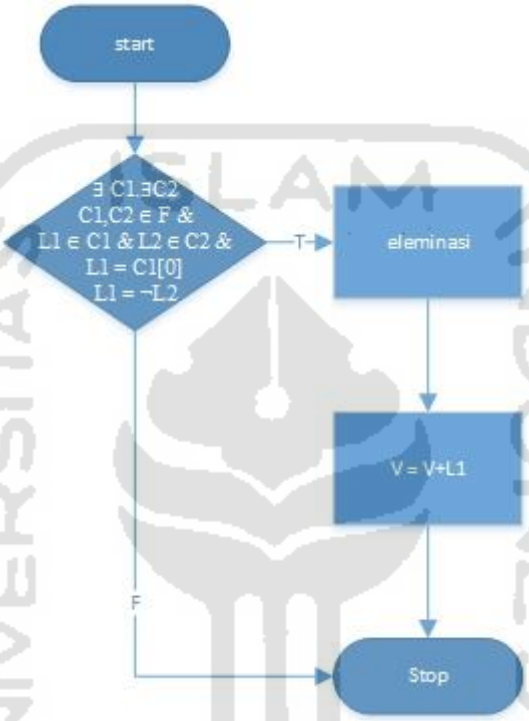


Gambar 3.10 Pengecekan Unit Klausa

Jika memenuhi maka eliminasi. Fungsi ini berguna untuk melihat apakah aturan unit klausa bisa digunakan atau tidak, dengan aturan dimana ada satu literal yang literal itu adalah *single* literal, yang artinya klausa hanya ada satu literal, dengan komplemen atau ingkaran dari literal ada di klausa yang lain. Notasinya adalah ∃ C1 .∃C2 (C1,C2 ∈ F & [L1] ∈ C1 & L2 ∈ C2 & L1 = ¬L2), dibaca ada C1, ada C2, C1 C2 adalah elemen dari F dan L1 adalah *single* literal yang elemen dari L1, dan L2 elemen dari C2 dan L2 adalah ingkaran L1. Jika terpenuhi maka lakukan eliminasi dan literal ditambahkan ke *variable value.*

3.4.6. Proses Split

Pada Gambar 3.11 dibawah menunjukan pengecekan unit klausa, dimana memilih literal pertama dalam klausa pertama dan memiliki komplemennya di klausa yang lain klausa.

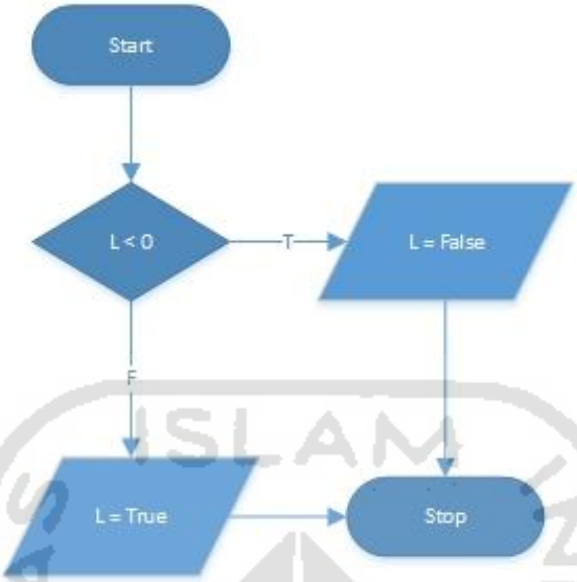


Gambar 3.11 Pengecekan Split

Jika memenuhi maka eliminasi.Fungsi ini berguna untuk melihat apakah aturan *split* bisa digunakan atau tidak, dengan aturan dimana ada literal pada suatu klausa dengan komplemen atau ingkaran dari literal ada di klausa yang lain. Notasinya adalah:∃ C1 . ∃ C2 (C1,C2 ∈ F & L1 ∈ C1 & L2 ∈ C2 & L1 = ¬L2), dibaca ada C1, ada C2, C1 C2 adalah elemen dari F dan L1 elemen dari C1, dan L2 elemen dari C2 dan L2 adalah komplemen atau ingkaran L1.

3.4.7. Proses Pemberian Value

Untuk lebih jelasnya tentang pemberian value berikut Gambar 3.12 diagram alir dari pemberian value:



Gambar 3.12 Pemberian Value

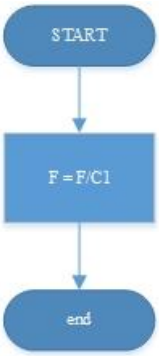
Pada Gambar 3.12 menunjukan proses pemberian value, jika literal lebih dari 0 maka literal bersolusi nilai true, jika literal kurang dari 0 maka literal bersolusi nilai false.

3.5. Perancangan Eliminasi Aturan Davis-Putnam-Logemann-Loveland

Perancangan eliminasi ini dibuat berdasarkan atauran yang dibuat dan yang telah dianalisis sebelumnya, yaitu mengeliminasi klausa maupun literal yang dapat dieliminasi sesuai aturan yang telah dipaparkan sebelumnya. Berikut merupakan rancangan eliminasi aturan dari algoritma DPLL:

3.5.1. Eliminasi Tautologi

Gambar 3.13 dibawah menunjukan formula yang direpresentasikan sebagai F dan klausa di representasikan sebagai C1 .

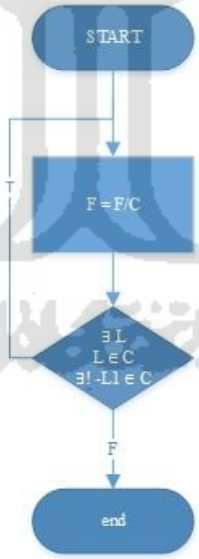


Gambar 3.13 Eliminasi Tautologi

Dimana F dengan menghapus elemenya yang berupa C1, yang dimana C1 adalah klausa yang dapat diterapkan aturan tautologi.

3.5.2. Eliminasi Pure Literal

Untuk lebih jelasnya tentang eliminasi pure literal, berikut Gambar 3.14 diagram alir dari elimnasi pure literal:



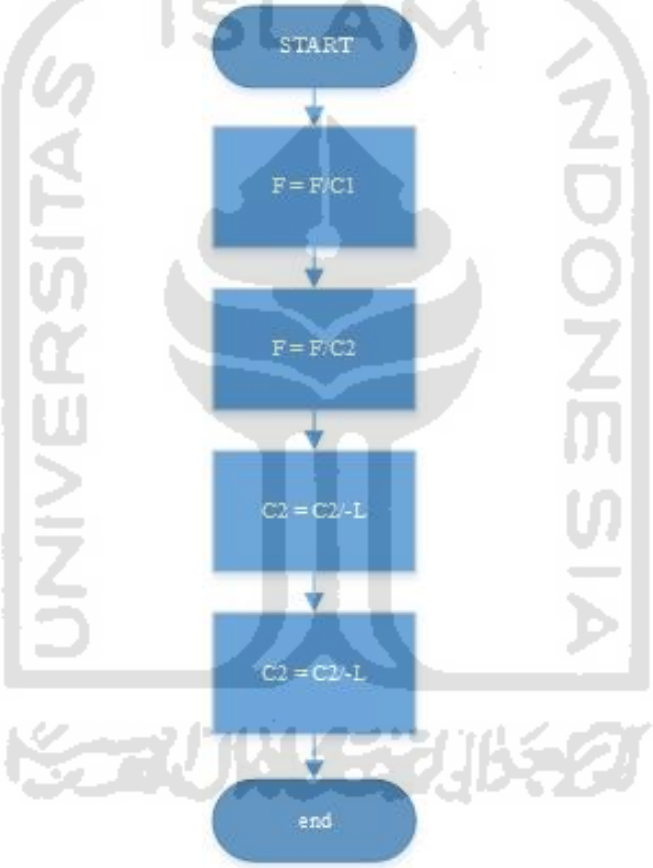
Gambar 3.14 Pengecekan Pure Literal

Gambar 3.14 menunjukan emilinasi klausa dari pure literal dengan mengeliminasi klausa yang dapat diterapkan aturan pure literal. Setelah dieliminasi dicek kembali apakah ada klausa yang dapat dieliminasi kembaliyang

sama dengan literalnya. Jika ada maka dieliminasi lagi, jika tidak maka proses eliminasi selesai.

3.5.3. Eliminasi Unit Klausa

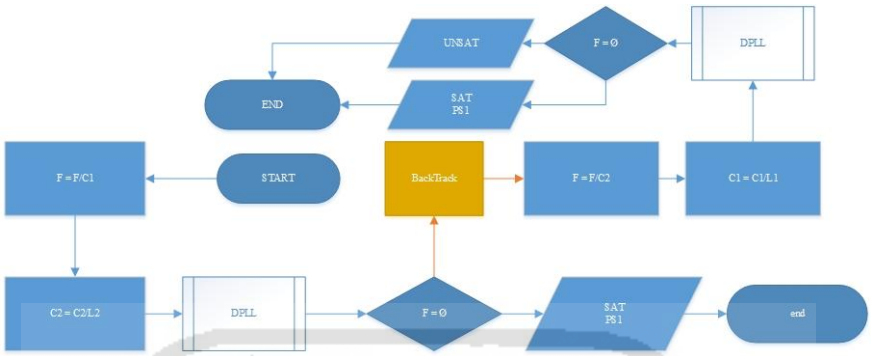
Untuk lebih jelasnya tentang eliminasi unit klausa, berikut Gambar 3.15 diagram alir dari elimnasi unit klausa:



Gambar 3.15 Eliminasi Unit Klausa

Gambar 3.15 menunjukan eliminasi klausa yang ada di F dengan menghapus klausa yang dapat diterapkan aturan unit klausa. Mengeliminasi C1 yang dimana *single* literal yang berisi dan mengliminasi L2 dimana L2 adalah komplemen L1.

3.5.4. Split dan Backtrack



Gambar 3.16 Eliminasi Split dan Backtrack

PadaGambar 3.16 menunjukan eliminasi *split*, yaitu memberikan nilai *true* pada salah satu literal dan *false* pada literal komplemennya. Lalu mengeliminasi literal yang bernilai *false* dari klausanya dan mengeliminasi klausa yang literalnya diberikan *true*. Lalu rekursif atau memanggil ulang fungsi DPLL yaitu tautologi, pure literal, unit klausa dan juga *split*. Jika formula bernilai himpunan kosong, maka formula *satisfiable*. Jika tidak bernilai himpunan kosong, maka unsat. Berbeda dengan proses lain yang jika dicek formula unsat atau tidak dan ternyata unsat, maka tidak bisa dirubah kembali literalnya. Maka *split* mempunyai kesempatan untuk membalikkan nilai literal.Sehingga yang telah diberikan nilai *true*dan dihapus klausanya, dirubah menjadi *false* dan literalnya yang dihapus. Sedangkan untuk yang telah dihasilkan*false* maka dirubah menjadi *true* dan yang tadi literalnya dihapus, untuk sekarang klausanya yang dihapus. Lalu *rekursif* atau memanggil ulang fungsi DPLLyaitu tautologi, pure literal, unit klausa dan juga *split*. Jika formula bernilai himpunan kosong, maka formula *satisfiable* dan jika tidak bernilai himpunan kosong, maka unsat dan kali ini tidak bisa dirubah nilainya.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi Sistem

Tahap implementasi sistem merupakan tahap dimana sistem telah selesai dibuat dan akan diimplementasikan.

4.1.1 Implementasi *input* file

Implementasi *input* file merupakan prosedur pertama untuk menjalankan programdengan *input* berupa *file text* dengan ekstensi txt. Implementasi dari proses *input* dapat dilihat pada Gambar 4.1.

```
if argv is None: argv = sys.argv
       if len(argv) !=2:
              print "filenya belom di input -_-"
              return 1
       try:
              filename = argv[1]
              solve_file(filename,pp,trace)
       except IOError:
              print "yang bener inputnya -_-"
       return 0

def solve_file(filename, pp=False, trace=False):
       f=baca_file(filename)
       print "formulanya : ", f
       print " "
       filenya.write("Formulanya : ")
       filenya.write(str(f))
       filenya.write("\n")
       return solve_formula(f, pp, trace)

def baca_file(filename):
              formula = []
              f = open(filename, 'r')
              for line in f:
                     klausa = [int(x) for x in line.split(" ")]
                     klausa = list(set(klausa))
                     formula.append(klausa[:1000000])
              print " "
              return formula
```

Gambar 4.1 Kode Program Implementasi *Input* File

Pada Gambar 4.1 file dimasukan lalu dibaca pada fungsi solve_file, yang dimana fungsi *solve file* menerima nilai dari fungsi baca_file. Fungsi baca file sendiri memparsing menjadi satu klausa pada satu baris sebelum ganti baris.

4.1.2 Implementasi unsatisfiable

Implementasi cek unsat merupakan prosesdur untuk mencek apakah file tersebut *unsatifiable* atau tidak. Implementasi proses cek unsat dapat dilihat pada Gambar 4.2.

```
def unsatisfy(f,v):
       filenya.write("solusinya : ")
       filenya.write(str(v))
       filenya.write("\n")
       print "solusinya ", v
       print " "
       return adanya(f, lambda c: semua(c, lambda l: -l in v))
```

Gambar 4.2 Kode Program Implementasi*Unsatisfiable*

Pada Gambar 4.2 adalah fungsi untuk memeriksa apakah masih ada klausa di C atau tidak.

4.1.3 Implementasi satisfiable

Implementasi cek *satisfiable* merupakan prosesdur untuk mengecek apakah file tersebut *satisfiable* atau tidak. Implementasi proses cek *satisfiable* dapat dilihat pada Gambar 4.3.

```
def klausa_satisfy(c,v):
       return adanya(c, lambda l: l in v)

def satsify(f,v):
       #print f
       return semua(f, lambda c: klausa_satisfy(c,v))
```

Gambar 4.3 Kode Program Implementasi *Satisfiable*

Pada Gambar 4.3 fungsi klausa_satisfy memerika apakah ada literal dalam satu klausa atau tidak. Dan fungsi *satify* adalah mengecek adanya klausa tersisa atau tidak pada formula.

4.1.4 Implementasi tautologi

Implementasi *satisfiable* merupakan prosesdur untuk mengecek apakah formula tersebut dapat diterapkan aturan tautologi atau tidak. Implementasi proses cek tautologi dapat dilihat pada Gambar 4.4.

```
def tautologi(f):
	for c in f:
		for l in c:
			if l in c and -l in c:
				return l, [c for c in f if -l not in c
and l not in c]
	return None, None
```

Gambar 4.4 Kode Program Implementasi tautologi

Pada Gambar 4.4 fungsi tautologi memeriksa apakah dalam suatu klausa ada literal dan komplemennya. Jika ada, maka *return*klausa pada f dimana literal dan komplemennya tidak ada pada C.

4.1.5 Implementasi pure literal

Implementasi cek pure literal merupakan prosesdur untuk mengecek apakah formula tersebut dapat diterapkan aturan *pure* literal atau tidak. Implementasi proses cek *pure* literal dapat dilihat pada Gambar 4.5.

```
def pure_literal(f):
	literal = rata(f)
	for l in literal:
		if -l not in literal:
			return l, [c for c in f if l not in c]
	return None, None
```

Gambar 4.5 Kode Program Implementasi pure literal

Pada Gambar 4.5 fungsi pure_literal memeriksa adakah literal yang tidak ada komplemennya pada klausa satu dan yang lainya dalam satu formula. Jika tidak ada makak *return*klausa yang literal tidak ada dalam klausa.

4.1.6 Implementasi unit klausa

Implementasi cek unit klausa merupakan prosedur untuk mengecek apakah formula tersebut dapat diterapkan aturan unit klausa atau tidak. Implementasi proses cek tautologi dapat dilihat pada Gambar 4.6.

```
def unit_klausa(f,v):
	def literalss(c):
		return [ l for l in c if l not in v and -l not in v ]
	for c in f:
		unassigned = literalss(c)
	if len(unassigned) == 1:
		return unassigned[0]
	return None
```

Gambar 4.6 Kode Program Implementasi cek unit klausa

Pada Gambar 4.6 fungsi unit_klausa memeriksa adakah literal dengan panjang *list*hanya satu dan ada komplemennya di klausa yang lain atau tidak. Jika ada maka *return*memindahkan klausa yang panjangnya hanya satu ke dalam *variable value*.

4.1.7 Implementasi split

Implementasi cek *satisfiable* merupakan prosesdur untuk mengecek apakah formula tersebut dapat diterapkan aturan *split* atau tidak. Implementasi proses cek *split* dapat dilihat pada Gambar 4.7.

```
def split(f, sym, v):
	f1 = removeKlausa(f, v[-1])
		#print f
		#print f1
	v1 = v
	sym1 = sym
		#l, s = sym1[0], sym1[1:]
		#print "xxxxx"
	return dfs(f1, sym1, v1) #or dfs(f, sym, v)

def removeKlausa(f, l):
	literal = rata(f)
	for i in literal:
		if l or -l in literal:
			s = [c for c in f if l not in c]
			return [remove(k, -l) for k in s]
	return None, None
```

Gambar 4.7 Kode Program Implementasi Split

Pada Gambar 4.7 fungsi *split* adalah untuk mencari literal pertama pada klausa pertama yang ada dan mencari ingkarannya di klausa yang lain. Lalu di formula dihapus pada fungsi removeKlausa dimana removeKlausa membalikan nilai formula yang sudah dieliminasi. Setelah proses removeKlausa maka fungsi *split* memberi nilai *return* ke fungsi dfs lagi.

4.1.8 Implementasi backtrack

Implementasi *backtrack* merupakan proses setelah *split* dan mendapatkan hasil *unsatifiable* pada percobaan pemberian nilai literal pada perocbaan pertamalalu memberikan nilai *true* yang sebelumnya*false* dan memberikan *false* yang sebelumnya *true*. Implementasi *backtrack* dapat dilihat pada Gambar 4.8.

```
print "formulanya f sebelum di eliminasi : ",f
                        print "split ", -l
                        filenya.write("split ")
                        filenya.write(str(-l))
                        filenya.write("\n")
                        #time.sleep(.500)
                        #print l, s
                        a = split(f, remove(sym,abs(l)), v+[-l])
                        if (a):
                                return a
                        else:
                                filenya.write("formulanya f
sebelum di eliminasi : ")
                                filenya.write(str(f))
                                filenya.write("\n")
                                print "formulanya f sebelum di
eliminasi : ",f
                                print "split ", l
                                filenya.write("split ")
                                filenya.write(str(l))
                                filenya.write("\n")
                                return split(f,
remove(sym,abs(l)), v+[l])
```

Gambar 4.8 Kode Program Implementasi *Backtrack*

Pada Gambar 4.8*backtrack* dilakukan karena terjadi *unsatisfiable* setelah *split* yang pertama lalu merubah yang tsebelumnya nilai literal *false* menjadi *true*, dan *true* menjadi *false* yang kemudian*return*ke fungsi split.

4.1.9 Keluaran solusi

Implementasi informasi solusi adalah *output* solusi literal mana saja yang harus bernilai *true* dan mana saja yang harus bernilai *false*. Contoh solusi dapat dilihat pada Gambar 4.9.

```
def nilai(v):
	sym = [ abs(l) for l in v ]
	true =  [ l for l in v if l > 0 ]
	false = [ l for l in v if l < 0 ]
	result = []
	if len(true) >0:
		for l in true:
			print str(l)+" true"
			filenya.write(str(l))
			filenya.write(" true")
	if len(false)>0:
		for l in false:
			print str(abs(l))+" false"
			filenya.write(str(l))
			filenya.write(" false")
	return "\n".join(result)
```

Gambar 4.9 Kode Program Implementasi Keluaran Solusi

Pada Gambar 4.12 fungsi nilai memproses literal yang telah didapat setelah diproses sebelumnya. Jika literal itu kurang dari nol, maka *variable* bernilai *false* dan jika literal lebih dari nol maka *variable* bernilai *true.*

4.1.10 Isi file *input*

Isi dari file masukan adalah berupa angkaatau bilangan integer dan tidak boleh terdapat unsure huruf. Contoh file masukan dapat dilihat pada Gambar 4.10

Gambar 4.10 Isi File Masukan

4.1.11 Eksport hasil ke file txt

Isi dari file hasil.txt adalah langkah eliminasi dapat dilihat pada Gambar 4.13.

```
filenya = open("hasil.txt","w")
                                    filenya.write("solusinya : ")
                                    filenya.write(str(v))
                filenya.write("formulanya f sebelum di eliminasi : ")
                                    filenya.write(str(f))
                                    filenya.write("\n")
                                    filenya.write("tautologi ")
                                    filenya.write(str(l))
                                    filenya.write("\n")
                                    filenya.write("pure literal ")
                                    filenya.write(str(l))
                                    filenya.write("\n")
                                    filenya.write("unit klausa ")
                                    filenya.write(str(l))
                                    filenya.write("\n")
                                    filenya.write("split ")
                                    filenya.write(str(-l))
                                    filenya.write("\n")
                                    filenya.write("split ")
                                    filenya.write(str(l))
                                    filenya.write("\n")
```

Gambar 4.11 Kode Program Implementasi Eksport Txt

Pada Gambar 4.11 file dibuat dengan nama hasil.txtlalu dituliskan apa yang kita inginkan.

4.2 Evaluasi Sistem

Pengujian sistem dilakukan secara pengujian tertutup yaitu mempraktekkan sistem (demo program) dan melihat apakah ada yang *error* atau tidak. Berikut merupakan daftar *error*:

1. Jika ada angka 0 pada *input*

Sistem SAT *Solver* menggunakan algoritma Davis-Putnam-Logemann-Loveland memasukan angka yang sebagai literal dan 0 juga merupakan angka tetapi tidak memiliki nilai minus untuk komplemennya. Maka penulis membarikan *error handling* dengan langsung keluar dari sistem dan mengarahkan kepada pengguna untuk mengganti nilai 0.

2. *Input* file salah format

Sistem SAT *Solver* menggunakan algoritma Davis-Putnam-Logemann-Loveland memasukan file berupa file extensi txt atau word, tapi jika file bukan yang diminta maka akan *error*. Oleh karena itu penulis memberikan *error handling* dengan langsung keluar dari sistemdan mengarahkankepada pengguna untuk memberikan *input* file dengan format yang benar.

3. Belum *input* file

Sistem SAT *Solver* menggunakan algoritma Davis-Putnam-Logemann-Loveland butuh *input* berupa file. Jika pengguna belum memberikan masukan maka penulis memberikan *error handling* dengan keluar dari sistem dan mengarahkan kepada pengguna untuk memasukan file.

4. Mengeksport file berbentuk txt untuk hasil

Pada saat progres tugas akhir, dosen memberikan saran untuk meng-*export* file dalam bentuk txt agar lebih menambah fitur yang dirasa dosen belum begitu lengkap.

4.3 Pengujian sistem

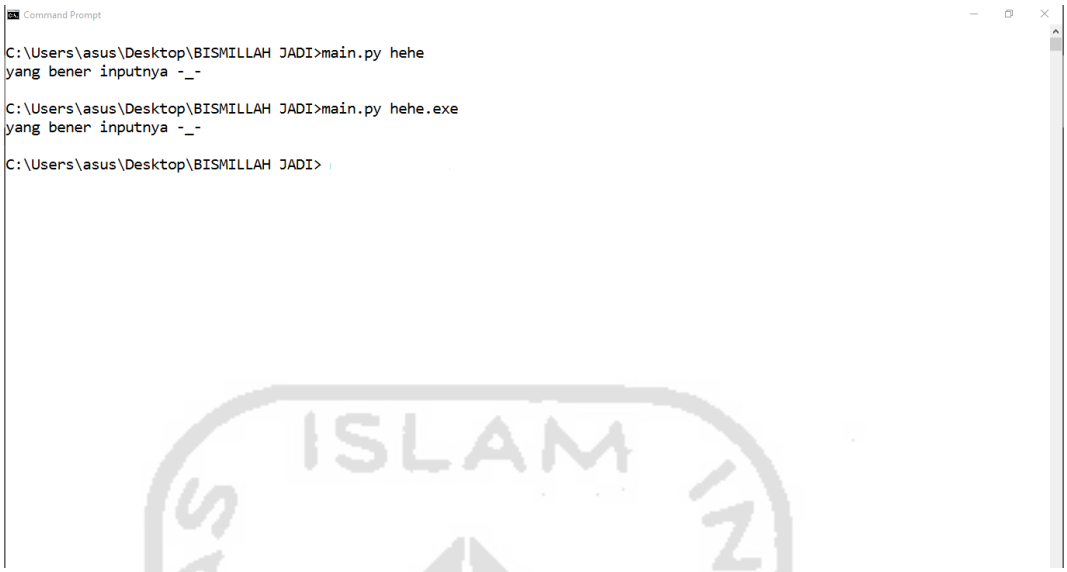
Pengujian sistem merupakan tahapan terakhir dalam pembangunan sistem (perangkat lunak), di mana pada tahap ini dapat diketahui apakah sistem dapat berjalan dengan baik atau tidak.

4.3.1 Penanganan kesalahan

Sistem yang baik adalah sistem yang mampu berinteraksi dengan ramah terhadap penggunanya. Hal itu dapat dilihat ketika pengguna berhasil memasukkan data dan melakukan suatu proses maka sistem akan menampilkan informasi berupa pesan konfirmasi. Begitu pula ketika terjadi kesalahan dalam memasukkan data dan melakukan proses, maka sistem akan menampilkan informasi berupa pesan kesalahan. Berikut ini merupakan implementasi penanganan kesalahan dan konfirmasi dari sistem yang dibuat.

1. Penanganan kesalahan format ekstensi pada masukan

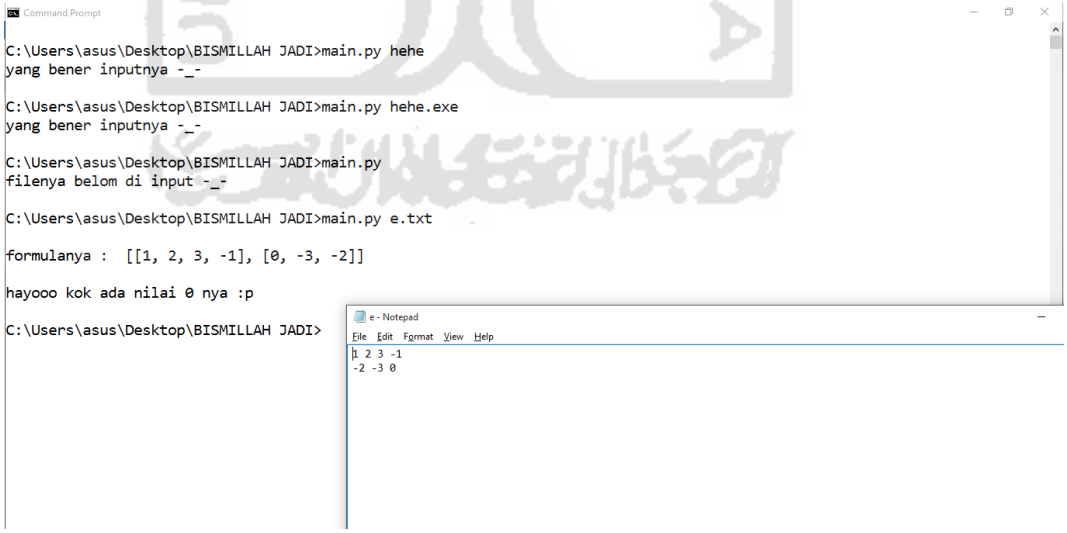
Kesalahan ini dilakukan ketika ekstensi file masukan tidak benar, maka sistem dapat mengatasinya dengan menampilkan informasi berupa pesan kesalahan. Informasi pesan kesalahan yang akan ditampilkan dapat dilihat pada Gambar 4.12 berikut.



Gambar 4.12 Implementasi *Error Handling*

2. Penanganan ada nilai 0 pada masukan

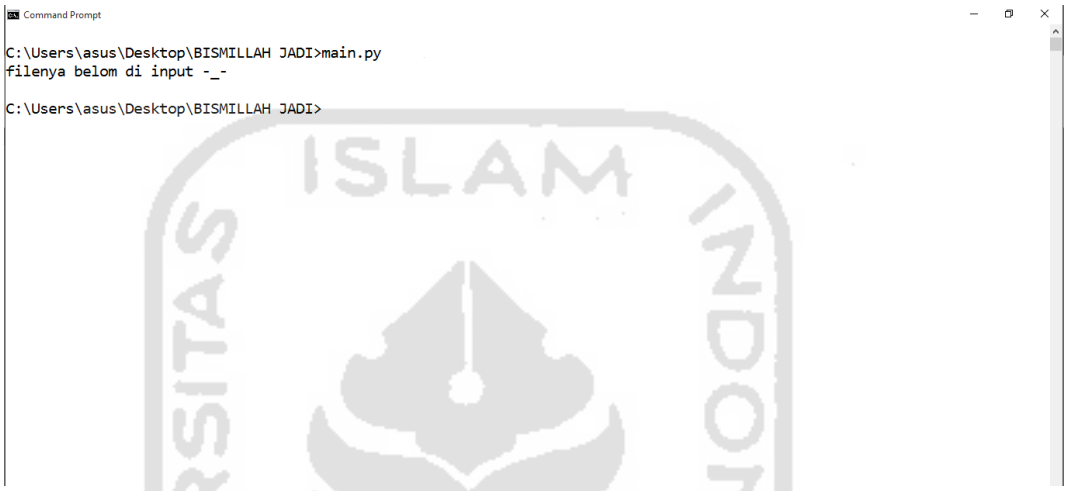
Kesalahan ini dilakukan ketika file masukan sudah benar tapi masih ada angka 0, maka sistem dapat mengatasinya dengan menampilkan informasi berupa pesan kesalahan. Informasi pesan kesalahan yang akan ditampilkan dapat dilihat pada Gambar 4.13 berikut.



Gambar 4.13 Implementasi *Error*niali 0 (nol)

3. Penanganan belum *input* file

Kesalahan ini dilakukan ketika file belum di *input*, maka sistem dapat mengatasinya dengan menampilkan informasi berupa pesan kesalahan. Informasi pesan kesalahan yang akan ditampilkan dapat dilihat pada Gambar 4.14berikut.



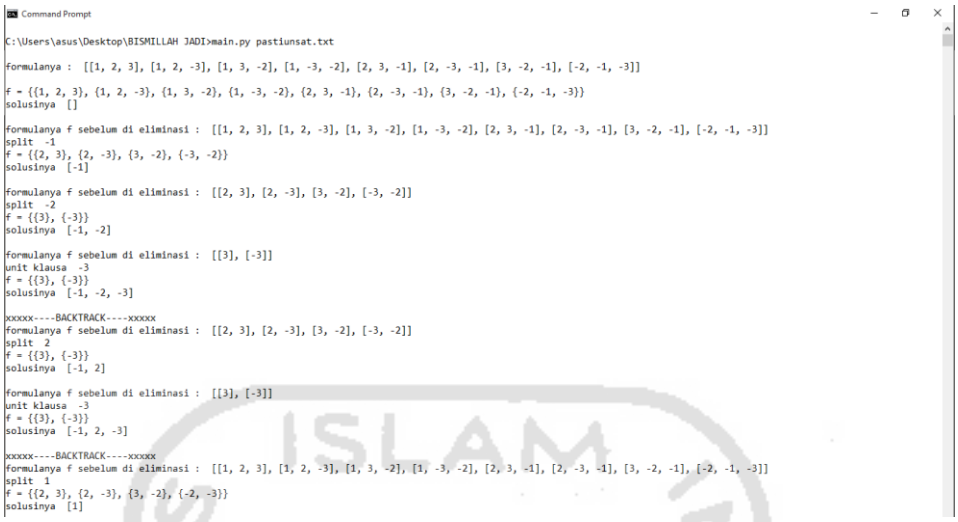
Gambar 4.14 *Error Handling* Belum *Input*

4.3.2 Black Box Testing

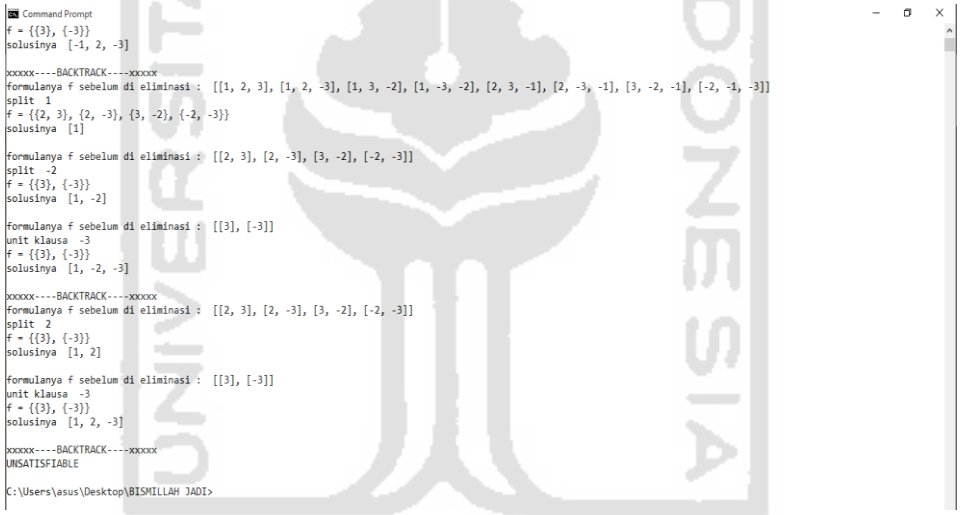
Sistem yang baik adalah sistem yang mampu berinteraksi dengan ramah terhadap penggunanya dan benar juga implementasinya, sehingga harus ditest apakah sistem sudah berjalan sesuai harapan atau belum. Berikut hasil test dari beberapa masukan secara acak isinya.

1. Percobaan pertama

   Diberikan formula {{1,2,3},{1,2,-3},{1,-2,3},{1,-2,-3},{-1,2,3},{-1,2,-3},{-1,-2,3},{-1,-2,-3}}. Dari formula tersebut kita mendapatkan bahwa formula tersebut bernilai *unsatifiable*, karena nilai akhirnya tidak ada yang bernilai benar walau hanya satu.



Gambar 4.15 Pengujian 1 (i)

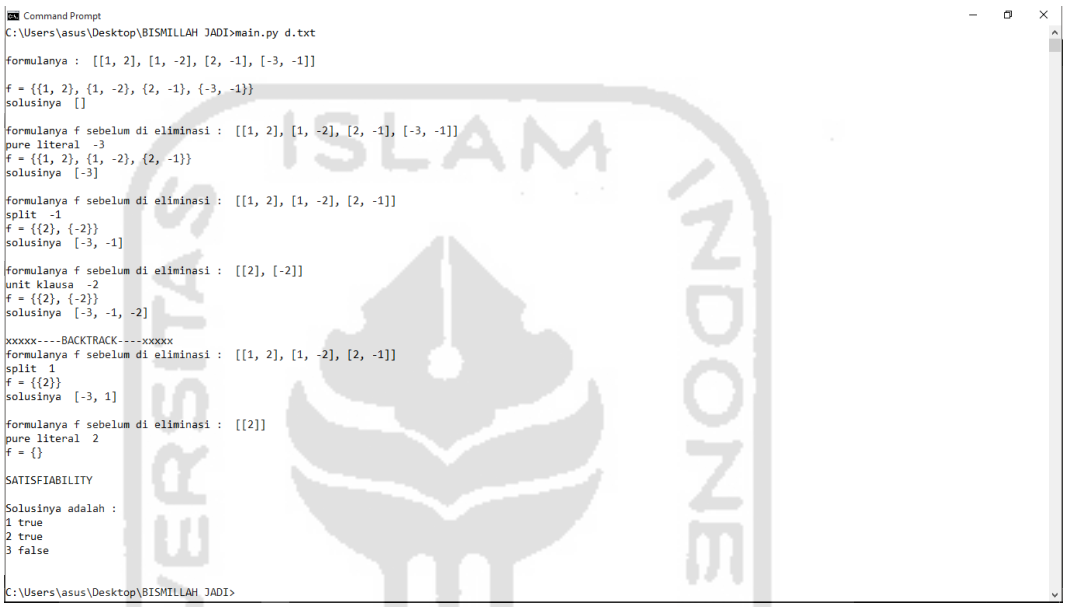


Gambar 4.16 Pengujian 1 (ii)

Dari Gambar 4.15 dan Gambar 4.16 dapat diamati pembuktian menggunakan *satisfiability solver* dengan algoritma Davis Putnam Logemann Loveland, dan terbukti bahwa tabel formula tersebut adalah *unsatisfiable* dengan waktu satu menit. Sehingga sistem terbukti berjalan dengan benar dan effisien.

2. Percobaan Kedua

Diberikan formula {{1,2},{1,-2},{-1,2},{-1,-3}}. Jika dikerjakan dengan sat *solver* menggunakan algoritma DPLL, maka dihasilkan seperti Gambar 4.17



Gambar 4.17 Pengujian 2

Dari Gambar 4.17 dapat diamati pembuktian menggunakan *satisfiability solver* dengan algoritma David Putnam Logemann Loveland, terbukti bahwa tabel formula tersebut adalah *satisfiable*, dengan solusi 1 *true* 2 *true* 3 *false*. Sehingga sistem terbukti berjalan dengan benardan waktu pengerjaan tidak lebih dari satu detik.Hal ini masih dirasa wajar karena *variable* yang digunakan masih sedikit. Jika diterapkan pada rumus tersebut maka:

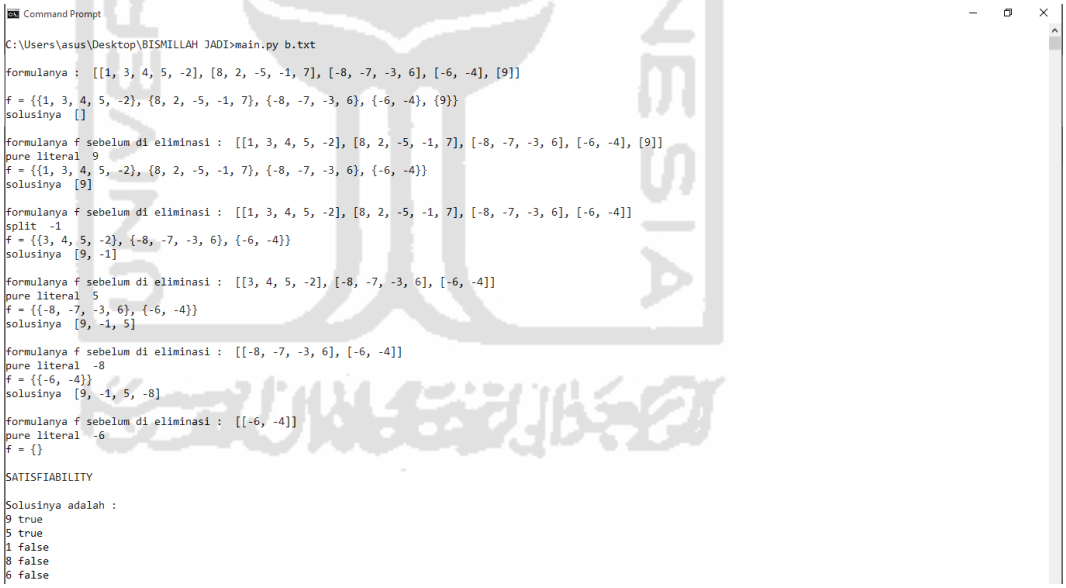
Tabel 4.1 Tabel Kebaran Percobaan 2

1	2	3	-1	-2	-3	A 1 v 2	B -1 v 2	C 1 v -2	D -1 v -3	A, B, C, D
T	T	F	F	F	T	T	T	T	T	T

Setelah diperiksa menggunakan cara manual seperti pada Tabel 4.1 diatas, ternyata hasilnya *satisfiable* dan solusi yang diberikan juga benar. Maka sat *solver* yang digunakan benar.

3. Percobaan ketiga

Diberikan formula {{1,3,4,5,-2},{8,2,-5,-1,7},{-8,-7,-3,6},{-6,-4},{9}} Jika dikerjakan dengan sat *solver* menggunakan algoritma DPLL, maka hasilnya akan menjadi seperti Gambar 4.18



Gambar 4.18 Pengujian 3

Dari Gambar 4.18 dapat diamati pembuktian menggunakan *satisfiability solver* dengan algoritma David Putnam Logemann Loveland, membuktikan bahwa formula tersebut adalah*satisfiable*, dengan solusi 9 *true* 5 *true* 1 *false* 8 *false* 6 *false*. Sehingga sistem

terbukti berjalan dengan benar dan waktu pengerjaan tidak lebih dari tiga menit. Hal ini masih wajar karena *variable*dan klausa yang digunakan masih sedikit. Jika diterapkan pada rumus tersebut maka dapat dilihat seperti Tabel 4.1 dibawah ini:

Tabel 4.2 Tabel Kebenaran Percobaan3

1	5	6	8	9	-	-	-	A	B	C	D	E	A,B,C,D,E
					1	6	8	1 v 3 v 4 v 5 v -2	8 v 2 v -5 v -1 v 7	-8 v -7 v -3 v 6	-6 v -4	9	
1	1	1	1	1	0	0	0	1	1	1	1	1	1

Setelah diperiksa menggunakan cara manual, ternyata hasilnya *satisfiable*dan solusi yang diberikan juga benar. Maka sat *solver* yang digunakan benar.

Lalu padawebsite http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html menyediakan kumpulan dari *satisfiability problem* yang dapat didownload. Setelah dijalankan menggunakan program SAT solver dapat diselesaikan dengan hasil yang baik dan benar.

4.4 Kelebihan dan Kekurangan Sistem

Satisfiability solver menggunakan algoritma David Putnam Logemann Loveland memiliki beberapa kelebihan dan kekurangan. Berikut adalah kelebihan dari sistem tersebut:

1) Sistem memberikan solusi nilai dari literal yang dieliminasi.
2) Sistem diberikan *error handling* yang cukup lengkap.

Berikut adalah kelebihan dari sistem tersebut:

1) Tidak adanya*graphic user interface*, sehingga kurang menarik walaupun sudah cukup informatif dan cukup untuk sebuah sistem.

BAB V
KESIMPULAN

5.1 Kesimpulan

Berdasarakan analisis, perancangan, dan pembuatan aplikasi SAT *Solver* ini hingga tahap penyelesaian dan pengujian, dapat diambil beberapa kesimpulan sebagai berikut:

1. SAT *Solver* dapat digunakan untuk yang mencari *satisfiability problem*.
2. SAT *Solver* cukup mumpuni dalam menggantikan pencarian *satisfiability problem* yang mencarinya dengan cara manual. SAT *Solver* dapat dikatakan sangat praktis bagi pengguna karena menampilkan proses dan solusi yang ada dapat digunakan.

5.2 Saran

Berdasarkan kekurangan–kekurangan yang didapati selama proses pembuatan dan pengujian, dengan ini saran yang diperoleh yaitu:

1. *User Interface* hanya dari *command line*, sehingga kurang menarik walau sudah cukup memberikan informasi dan solusi yang tepat.
2. Hanya menggunakan *conjuntive normal form*, sehingga tidak menggunakan *normal form* yang lain. Disarankan menjadi judul tugas akhir adik-adik mahasiswa untuk membuat aplikasi yang dapat merubah *normal form* menjadi *conjuntive normal form.*
3. *Satisfiability problem* dalam kehidupan sehari-hari memiliki contoh problem yaitu pada *game* sudoku, sehingga dapat menjadi topik tugas akhir dengan judul “sudoku *solver* menggunakan algoritma DPLL”.

DAFTAR PUSTAKA

Dewi, S. K. (2006). *Artificial Intelegence (Teori dan Penerapannya)*. Yogyakarta: Andi.

Dwijono, D. (2004). *Logika Matematika Untuk Ilmu Komputer*. Yogyakarta: Andi.

Gu, J., Franco, J., Wah, B. W., & Purdom, P. W. (2002). Algorithm For the Satisfiability (SAT) Problem. In *Combinatorial Optimization* (pp. 379–572). Springer US.

Holldobler, S. (2009a). Davis-Putnam-Logemann-Loveland Method. *Journal of Applied Logic*, *2*(3), 245–272.

Holldobler, S. (2009b). Satisfiability Testing or How to Solve Sudoku Puzzles. *Propositional Logic and the Satisfiability Problem*, *4*(22), 304–315.

Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2006). Solving SAT and SAT Modulo Theories: From and Abstract Davis-Putnam-Logemann-Loveland. *Journal of the ACM (JACM)*, *53*(6), 937–977.

Soesianto, F. (2006). *Logika Matematika Dalam Komputer*. Yogyakarta: Andi.

Wikipedia. (2016a). Conjunctive Normal Form. Retrieved February 14, 2017, from https://en.wikipedia.org/wiki/Conjunctive_normal_form

Wikipedia. (2016b). Disjunctive Normal Form. Retrieved from https://en.wikipedia.org/wiki/Disjunctive_normal_form

Wikipedia. (2016c). Normal Form. Retrieved February 14, 2017, from https://en.wikipedia.org/wiki/Algebraic_normal_form