PENYELESAIAN *BOOLEAN SATISFIABILITY PROBLEM* DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA

Disusun Oleh:

N a m a : Dicky Puja Pratama

NIM : 14 523 063

PROGRAM STUDI TEKNIK INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2018

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENYELESAIAN *BOOLEAN SATISFIABILITY PROBLEM* DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA

TUGAS AKHIR

Disusun Oleh:

N a m a : **Dicky Puja Pratama**

NIM : **14 523 063**

Yogyakarta, 18 Oktober 2018

Pembimbing,

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

HALAMAN PENGESAHAN DOSEN PENGUJI

PENYELESAIAN *BOOLEAN SATISFIABILITY PROBLEM* DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik Informatika
di Fakultas Teknologi Industri Universitas Islam Indonesia
Yogyakarta, 3 Desember 2018

Tim Penguji

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

Anggota 1

Arrie Kurniawardhani, S.Si., M.Kom. _____

Anggota 2

Almed Hamzah, S.T., M.Eng. _____

Mengetahui,
Ketua Program Studi Teknik Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dr. Raden Teduh Dirgahayu, S.T., M.Sc.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Dicky Puja Pratama

NIM : 14 523 063

Tugas akhir dengan judul:

PENYELESAIAN *BOOLEAN SATISFIABILITY PROBLEM* DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila dikemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya 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, 17 Desember 2018

(Dicky Puja Pratama)

HALAMAN PERSEMBAHAN

Alhamdulilahi Robbil 'Alamin puji syukur atas segala nikmat dan karunia yang Allah SWT berikan kepada saya sehingga tugas akhir ini dapat terselesaikan dengan baik. Atas semua dukungan dan bantuan yang telah diberikan, saya persembahkan tugas akhir ini untuk:

Kedua orangtua yang selalu melimpahkan seluruh kasih sayang, doa, nasihat, arahan, dan dukungan kepada saya sampai saat ini dan tidak pernah berhenti.

Adikku Ajeng Dwi Anggita yang selalu memberikan semangat dan juga kebahagiaan.

HALAMAN MOTO

“Maka sesungguhnya bersama kesulitan ada kemudahan. Sesungguhnya bersama kesulitan itu ada kemudahan.

(Q.S. Al Insyirah: 5-6)

"Barang siapa menginginkan soal-soal yang berhubungan dengan dunia, wajiblah ia memiliki ilmunya ; dan barang siapa yang ingin (selamat dan berbahagia) di akhirat, wajiblah ia mengetahui ilmunya pula; dan barangsiapa yang menginginkan kedua-duanya, wajiblah ia memiliki ilmu kedua-duanya pula".

(HR. Bukhari dan Muslim)

“Seorang terpelajar harus sudah berbuat adil sejak dalam pikiran apalagi dalam perbuatan”

(Pramoedya Ananta Toer)

“Perjuangan harus dilakukan dengan hati yang membara semangatnya”

(Dicky Puja Pratama)

KATA PENGANTAR

Assalamu`alaikum Warahmatullahi Wabarakatuh

Alhamdulillah, puji dan syukur penulis panjatkan kehadirat Allah SWT yang telah memberikan rahmat, hidayah, serta karunianya, sehingga Laporan Tugas Akhir **PENYELESAIAN *BOOLEAN SATISFIABILITY PROBLEM* DENGAN ALGORITMA DAVIS PUTNAM LOGEMANN LOVELAND (DPLL) MENGGUNAKAN JAVA** dapat diselesaikan. Tak lupa shalawat serta salam selalu kami curah limpahkan kepada junjungan Nabi Muhammad SAW, yang telah membawa kita dari zaman kebodohan menuju zaman yang penuh dengan ilmu pengetahuan seperti sekarang ini.

Semoga Laporan Tugas Akhir ini bisa bermanfaat. Dengan selesainya Laporan Tugas Akhir ini tidak lepas dari bantuan yang diberikan berbagai pihak kepada penulis. Oleh karena itu penulis mengucapkan terima kasih kepada:

1. Orangtua dan adik atas segala doa, usaha dan kasih sayang yang selalu diberikan kepada penulis.
2. Dekan Fakultas Teknologi Industri, Universitas Islam Indonesia.
3. Ketua Jurusan Teknik Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia.
4. Bapak Taufiq Hidayat, S.T., M.C.S. selaku Dosen Pembimbing Laporan Tugas Akhir yang telah memberikan ilmu, motivasi, serta bimbingan dalam penyusunan Tugas Akhir ini.
5. Keluarga Besar LPM PROFESI yang telah memberikan begitu banyak pengalaman organisasi dan juga pengalaman hidup yang berarti.
6. Seluruh keluarga besar Teknik Informatika 2014 yang telah menjadi teman seperjuangan dalam menjalani masa pendidikan.
7. Semua pihak yang telah banyak membantu dalam pelaksanaan Tugas Akhir yang tidak dapat disebutkan satu persatu.

Penulis menyadari bahwa penyusunan Laporan Tugas Akhir ini masih belum sempurna. Hal ini semata-mata karena keterbatasan kemampuan dan pengetahuan penulis. Oleh karena itu penulis dengan terbuka menerima kritik dan saran yang membangun.

Besar harapan penulis terhadap laporan Tugas Akhir yang telah penulis selesaikan, semoga laporan ini dapat bermanfaat dan bisa menjadi pembelajaran semua pihak.

Wassalamu`alaikum Warahmatullahi Wabarakatuh

Yogyakarta, 2 Desember 2018

(Dicky Puja Pratama)

SARI

Boolean Satisfiability Problem (SAT *Problem*) adalah konsep dasar semantik yang menentukan ada atau tidaknya interpretasi (pemberian nilai kebenaran pada setiap simbol proposisi) yang memberikan hasil yang *satisfiable* pada formula logika *boolean*. Dalam penelitian ini, SAT Problem direpresentasikan dalam bentuk *Conjunctive Normal Form* (CNF) dan diselesaikan menggunakan algoritma Davis-Putnam-Logemann-Loveland (DPLL). Algoritma DPLL tersebut diimplementasikan dalam bahasa Java untuk dibuat sebuah aplikasi.

SAT *Problem* tergolong dalam permasalahan *Non Polynomal-Complete,* yang artinya kompleksitas waktu kasus terburuknya tidak dibatasi oleh fungsi polinom. Dengan kata lain apabila formula logika *boolean* yang ingin diketahui nilai *satisfiable*-nya memiliki ukuran yang relatif besar, maka hal itu akan sulit dikerjakan dengan cara konvensional.

Dalam penerapannya, algoritma DPLL memiliki dua fungsi dasar yang dapat digunakan untuk menyelesaikan permasalahan SAT Problem. Fungsi dasar tersebut ialah *Unit Propagation, Pure Literal, Decide,* dan *Backtracking.* Dalam penelitian ini akan dilakukan penyelesaian permasalahan SAT *Problem* menggunakan fungsi-fungsi tersebut dalam bahasa Java.

Kata kunci: *Boolean Satisfiability Problem, Non Polynomal-Complete, SAT Problem, Algoritma DPLL, Conjunctive Normal Form*

GLOSARIUM

Boolean	Suatu tipe data yang hanya mempunyai dua nilai, yaitu benar atau salah. Pada beberapa bahasa pemograman nilai true bisa digantikan 1 dan nilai false digantikan 0.
Cell	Pertemuan antara baris dan kolom.
Encoding	Suatu proses untuk mengubah karakter yang ada pada suatu set karakter menjadi nilai yang dimengerti oleh komputer,
Enterprise	Perusahaan
Flowchart	Suatu bagan dengan simbol-simbol tertentu yang menggambarkan urutan proses secara mendetail dan hubungan antara suatu proses (instruksi) dengan proses lainnya dalam suatu program.
IDE	*Integrated Development Environment* adalah program komputer yang memiliki fasilitas yang diperlukan dalam pembangunan perangkat lunak.
Integer	Tipe data untuk bilangan bulat.
Java	Bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam.
Method	Sarana bagi pemrogram untuk memodularisasi atau memecah program kompleks menjadi bagian yang kecil-kecil sehingga nantinya dapat digunakan berulang-ulang, daripada menulis beberapa baris kode yang sama
Mobile	Seluler
User Interface	Bentuk tampilan grafis yang berhubungan langsung dengan pengguna

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Boolean Satisfiability Problem (SAT *Problem*) adalah konsep dasar semantik yang menentukan ada atau tidaknya interpretasi (pemberian nilai kebenaran pada setiap simbol proposisi) yang memberikan hasil yang *satisfiable* pada formula logika *Boolean* (Kalla, 2017). Formula dalam SAT *Problem* berbentuk *Conjunctive Normal Form* (CNF). CNF membuat logika proposisi hanya memiliki unsur berupa variabel proposisi (untuk selanjutnya disebut dengan variabel), negasi, operator *or*, dan *and*. Variabel tersebut diberikan nilai *true* atau *false* sedemikian rupa sehingga akan memiliki hasil akhir berupa nilai *true* ataupun *false*.

Kompleksitas untuk menyelesaikan permasalahan SAT *Problem* sendiri adalah 2^n (Monien & Speckenmeyer , 1984) dan SAT *Problem* termasuk dalam permasalahan yang tergolong *Non Polinomial Complete* (Cook, 1971). 'n' pada proposisi kompleksitas ini mewakili banyaknya variabel. Dengan kata lain untuk menentukan apakah proposisi A v B adalah *satisfiable*, maka dibutuhkan kompleksitas sebesar $2^2 = 4$. Proposisi tersebut baru memiliki dua variabel. Jika proposisi tersebut memiliki 10 variabel, maka besar kompleksitasnya adalah $2^{10}=1024$.

Penyelesaian permasalahan SAT *Problem* yang paling banyak digunakan adalah dengan menggunakan Prosedur Davis-Putnam-Logemann-Loveland (DPLL). Algoritma DPLL mampu menentukan logika proposisi yang diberikan dapat menghasilkan nilai yang *satisfiable* atau tidak. DPLL juga mampu menemukan interpretasi nilai yang harus diberikan pada masing-masing variabel, apabila logika proposisi tersebut bernilai *satisfiable*.

Karena tergolong dalam permasalahan *Non Polinomial Complete*, maka SAT P*roblem* memiliki kompleksitas yang sangat besar. Sehingga apabila permasalahan SAT *Problem* diselesaikan dengan cara manual, tidak akan mudah untuk diselesaikan dan memakan waktu yang relatif lama. Terlebih lagi, Algoritma DPLL memiliki cara kerja yang harus melakukan pengecekan terhadap masing-masing literal dan klausa yang diberikan, sehingga dengan

dibuatnya program penyelesaian SAT *Problem* ini, diharapkan mampu menyelesaikan masalah-masalah tersebut.

Berdasarkan uraian di atas, maka penelitian ini bertujuan untuk menyelesaikan permasalahan SAT *Problem,* untuk menentukan nilai *satisfiable* dari suatu logika proposisi, dengan sebuah aplikasi yang dibangun menggunakan bahasa JAVA. Permasalahan tersebut diselesaikan dengan menggunakan algoritma DPLL. Aplikasi tersebut bermanfaat untuk menentukan apakah suatu logika proposisi bernilai *satisfiable* ataupun *unsatisfiable*.

1.2 Rumusan Masalah

Dari latar belakang dan identifikasi masalah di atas, maka dapat dihasilkan rumusan masalah sebagai berikut:

Bagaimana mengimplementasikan algoritma DPLL untuk menyelesaikan SAT *Problem* menggunakan bahasa Java?

1.3 Batasan Masalah

Batasan masalah pada penelitian ini adalah sebagai berikut:

1. Formula yang diselesaikan berbentuk *Conjunctive Normal Form*.
2. Program yang dibangun berbentuk konsol.
3. Masukan dan keluaran program berbentuk teks.

1.4 Tujuan Penelitian

Adapun tujuan yang ingin dicapai dari penelitian ini adalah:

1. Membangun aplikasi yang dapat menyelesaikan SAT *Problem*.
2. Mengimplementasikan alogritma DPLL untuk menyelesaikan SAT *Problem*.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai berikut:

1. Mempermudah dan mempercepat penyelesaian SAT *Problem*.
2. Mengatasi kompleksitas penyelesaian SAT *Problem* dengan algoritma DPLL.
3. Sebagai bahan rujukan untuk pengembangan penelitian selanjtunya mengenai SAT *Problem*.

1.6 Metodologi Penelitian

Metodologi penelitian yang digunakan dalam menyusun tugas akhir ini meliputi identifikasi masalah, analisis kebutuhan, perancangan, implementasi, dan pengujian. Hal ini dilakukan agar dalam penyelesaian tugas akhir lebih mudah dan terarah. Metodologi yang digunakan antara lain:

a. Identifikasi Masalah

Mengidentifikasi masalah dan kebutuhan, cara kerja dan ruang lingkup aplikasi yang akan dibangun dengan cara:

1. Melakukan analisis terhadap permasalahan SAT *Problem* dan cara-cara penyelesaiannya.
2. Melakukan analisis terhadap algoritma DPLL dan implementasinya untuk menyelesiakan permasalahan SAT *Problem.*
3. Menganalisis penggunaan bahasa JAVA untuk menyelesaikan permasalahan SAT *Problem* menggunakan algoritma DPLL.
4. Membaca literatur baik dari jurnal, buku, ataupun penelitian sebelumnya yang berhubungan dengan penyelesaian tugas akhir ini.

b. Analisis Kebutuhan

Pada tahapan ini dilakukan analisis kebutuhan apa saja yang diperlukan untuk membangun aplikasi berbasis bahasa JAVA yang dapat menyelesaikan permasalahan SAT *Problem* menggunakan algoritma DPLL. Pada tahapan ini juga mencakup analisis kebutuhan perangkat lunak dan perangkat keras yang dibutuhkan untuk membangun aplikasi tersebut.

c. Perancangan

Pada tahapan ini dilakukan perancangan aplikasi berbasis bahasa JAVA agar dapat menerima masukan berupa proposisi logika *boolean* dalam bentuk CNF yang dapat menyelesaikan permasalahan SAT *Problem*.

d. Implementasi

Pada tahapan ini dilakukan implementasi dari perancangan yang telah dilakukan pada tahap sebelumnya. Dilakukan implementasi algoritma DPLL dalam penyelesaian permasalahan SAT *Problem* menggunakan bahasa JAVA. Dalam penelitian ini juga akan

dilakukan implementasi penerapan algoritma DPLL untuk menyelesaikan permasalah SAT *Problem*.

e. Pengujian

Pada tahapan ini merupakan tahapan akhir dari proses-proses sebelumnya yaitu aplikasi yang telah dibuat harus diuji sehingga dapat mengetahui hasil yang sesuai dengan kebutuhan. Dalam penelitian ini, akan dilakukan uji coba sistem untuk menyelesaikan permasalahan pada permainan Sudoku.

1.7 Sistematika Penulisan

Sistematika penulisan ini dibuat untuk memberikan gambaran umum mengenai penelitian yang dijalankan. Sistematika penulisan ini dibagi dalam lima bab sebagai berikut:

BAB I PENDAHULUAN

Pendahuluan membahas permasalahan umum tentang latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metodologi penelitian serta sistematika penulisan penelitian penyelesaian permasalahan SAT *Problem*.

BAB II LANDASAN TEORI

Pada bagian ini menjelaskan teori-teori yang digunakan dalam penelitian penyelesaian permasalahan SAT *Problem*. Adapun teori yang disampaikan meliputi *Boolean Satisfiability Problem*, *Non Polynomial-Complete*, *Conjunctive Normal Form*, dan Algoritma Davis-Putnam-Logemann-Loveland.

BAB III METODOLOGI

Metodologi memuat uraian tentang analisis kebutuhan dan perancangan aplikasi untuk menyelesaikan SAT *Problem*.

BAB IV IMPLEMENTASI DAN PENGUJIAN

Pada bagian ini menejelaskan mengenai hasil implementasi dan pengujian aplikasi penyelesaian SAT *Problem.*

BAB V KESIMPULAN DAN SARAN

Pada bagian ini akan dijabarkan kesimpulan yang didapatkan setelah menyelesaikan penelitian. Selain itu juga diberikan saran yang memberikan poin-poin yang dapat ditingkatkan untuk penelitian selanjutnya.

BAB II
LANDASAN TEORI

2.1 Tinjauan Pustaka

Berikut adalah beberapa penilitan sebelumnya mengenai *Boolean Satisfiability Problem* yang berhasil dirangkum.

a. Penelitian yang dilakukan oleh Cecilia E. Nugraheni dengan judul "Penyelesaian Masalah Penjadwalan Ujian dengan SAT" (Nugraheni, 2008). Makalah tersebut memaparkan penelitian yang dilakukan terhadap masalah penjadwalan ujian Seminar di Jurusan Teknik Informatika Unpar. Permasalahan yang diangkat pada penelitian tersebut adalah sejauh mana masalah penjadwalan ujian seminar ini dapat dimodelkan sebagai permasalahan SAT dan bagaimana menerapkan teknik inferensi SAT *unit propagation* dan *failed literal rule* untuk menyelesaikan permasalahan penjadwalan ujian seminar tersebut. Sedangkan tujuan dari penelitian ini adalah memodelkan masalah penjadwalan ke dalam bentuk permasalahan SAT, menerapkan teknik inferensi *unit propagation* dan *failed literal rule*, dan mengimplementasikan algoritma kedua teknik inferensi tersebut ke dalam sebuah prototipe perangkat lunak.

b. Penelitian yang dilakukan oleh Ines Lynce dan Joel Ouknine dengan judul "Sudoku *as* SAT *Problem*" (Ouaknine & Lynce , 2006). Penelitian ini dilakukan untuk mengatasi permasalahan dalam penyelesaian permainan Sudoku menggunakan penalaran, bukan pencarian. Dalam penelitian tersebut masalah diubah dalam bentuk *Conjunctive Normal Form* (CNF) dan kemudian diselesaikan menggunakan teknik kesimpulan SAT. Untuk memecahkan teka-teki Sudoku ini, pada penelitian tersebut setiap masalah dijalankan pada empat konfigurasi yang berbeda. Pada semua konfigurasi tersebut terdapat teknik penyederhanaan yang paling banyak digunakan, yaitu *unit propagation*. Keempat konfigurasi tersebut adalah sebagai berikut:
    1. *Unit propagation* (up)
    2. *Unit propagation* dan *failed literal rule* (up+flr)
    3. *Unit propagation* dan *hyper-binary resolution* (up+hypre)
    4. *Unit propagation* dan *binary failed literal rule* (up+binflr).

c. Penelitian yang dilakukan oleh Robert Nieuwenhuis et al. dengan judul "*Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to* DPLL(T)" (Nieuwenhuis, Tinelli, & Oliveras, 2006). Makalah ini membahas penelitian tentang penyelesaian SAT menggunakan DPLL *Procedure*. Aturan dasar algoritma DPLL yang diterapkan dalam penyelesaian ini adalah *unit propagate, pure literal, decide, fail,* dan *backtrack*.

2.2 Dasar Teori

Dasar teori dalam penyelesaian *SAT Problem* ini adalah teori yang menjadi acuan dalam penelitian yang dilakukan. Seperti yang disebutkan dalam penjelasan di bawah ini:

2.2.1 Boolean Satisfiabiliaty Problem (SAT Problem)

Boolean Satisfiabiliaty Problem (SAT *Problem*) merupakan permasalahan yang menanyakan apakah formula logika *boolean* (proposisi) yang diberikan, dengan operasi *boolean* seperti *AND, OR* dan *NOT*, jika masing-masing variabelnya diberikan nilai *true* atau *false* dapat menghasilkan nilai akhir *true*. SAT *Problem* telah menjadi masalah penting dalam ilmu komputer sejak Stephen Cook membuktikan bahwa peyelesaian permasalahan ini tergolong dalam *Non Polynomial Complete* (NP-*Complete*) pada tahun 1971 (Vardi, 2014).

2.2.2 Non Polynomial-Complete

Non Polynomial-Complete (NP-*Complete*) adalah salah satu algoritma yang termasuk dalam *Non Polynomial* (NP). Algoritma *Non Polynomial* adalah algoritma yang kompleksitas waktu kasus terburuknya tidak dibatasi oleh fungsi polinom dari ukuran masukannya (Darmawan, Kusumastuti, & Yundari, 2014). Sedangkan yang dimaksud dengan NP-*Complete* adalah suatu permasalahan NP yang dapat dikurangi ke dalam waktu polinomial (Karakashian & Puranda, 2008). Fungsi polinom sendiri berarti persamaan yang memiliki variabel dengan pangkat bertingkat. Pangkat tertinggi dari sebuah polinomial disebut dengan derajat. Misalnya diberikan persamaan suku banyak $x^3–x+3$, suku banyak tersebut memiliki derajat 3.

2.2.3 Conjunctive Normal Form

Ekspresi logika (proposisi) dikatakan dalam *Conjunctive Normal Form* (CNF) bila merupakan suatu konjungsi dari disjungsi literal-literal (Astuti & Teguh, 2016). Bentuk umum dari CNF ditunjukkan pada Gambar 2.1.

$$F = F1 \wedge F2 \wedge Fn,\ Fn = L1 \vee L2 \vee Ln,\ n \geq 1$$

Gambar 2.1 Bentuk umum CNF

Di mana F1, F2, F3,….,Fn merupakan disjungsi dari literal atau disebut juga sebagai klausa. Sedangkan yang dimaksud dengan literal adalah sebuah atom atau negasi dari atom. Contoh proposisi dalam bentuk CNF ditunjukkan pada Gambar 2.2.

$$F = (p \vee \sim q \vee r) \wedge (\sim p \vee q)$$

Gambar 2.2 Contoh CNF

Dari Gambar 2.2 di atas, yang dimaksud dengan literal adalah p, ~q, r, ~p, dan q. Sedangkan yang dimakusd dengan klausa adalah $(p \vee \sim q \vee r)$ dan juga $(\sim p \vee q)$.

2.2.4 Algoritma Davis-Putnam-Logemann-Loveland

Cara penyelesaian SAT *Problem* yang cukup populer adalah menggunakan algoritma Davis-Putnam-Logemann-Loveland (DPLL). Dalam penggunaannya, masukan untuk Algoritma DPLL harus dalam bentuk CNF. Algoritma DPLL memiliki tiga fungsi dasar yang dapat digunakan untuk menyelesaikan permasalaan SAT *Problem*, yaitu *Unit Propagation*, *Pure Literal*, dan *Decide*.

a. *Unit Propagation* merupakan teknik penyederhanaan yang penting dalam suatu proposisi yang berbentuk CNF. Aturan Unit Propagation menyatakan bahwa ketika klausa hanya berisi satu literal, maka semua klausa yang mengandung literal tersebut harus dihapus, karena klausa dengan literal tunggal harus bernilai *true* agar proposisi tersebut bernilai *true*. Kemudian, setiap literal yang memiliki nilai kontradiksi dengan klausa yang berisi satu literal tersebut juga harus dihapus. (Vij, 2016). Sebagai contoh dapat dilihat pada Gambar 2.3.

$$S = (\sim P \vee Q \vee \sim R) \wedge (P \vee \sim Q) \wedge \sim P \wedge R \wedge U$$

Gambar 2.3 Contoh logika proposisi

Terapkan *Unit Propagation* dengan L = ~P, maka klausa yang memiliki literal ~P harus dihapus, seperti yang ditunjukkan pada gambar 2.4.

```
S` = (P ∨ ~Q) ∧ R ∧ U
```

Gambar 2.4 Penerapan Unit Propagation pada logika proposisi (Hapus Klausa)

Hapus ~L = P dari klausa S`, maka setiap literal ~P harus dihapus, seperti yang ditunjukkan pada Gambar 2.5.

```
S`` = ~Q ∧ R ∧ U
```

Gambar 2.5 Penerapan Unit Propagation pada logika proposisi (Hapus Literal)

b. *Pure Literal* dapat dilakukan apabila dalam suatu proposisi terdapat variabel yang hanya memiliki nilai *true* atau *false* saja. Klausa yang mengandung *Pure Literal* dapat dihapus dari suatu proposisi (Larrosa, Lynce, & Silva, 2010). Sebagai contoh diberikan logika proposisi seperti yang ditunjukkan pada Gambar 2.6.

```
φ = (~x1 ∨ x2) ∧ (x3 ∨ ~x2) ∧ (x4 ∨ ~x5) ∧ (x5 ∨ ~x4)
```

Gambar 2.6 Contoh logika proposisi

Pada proposisi di atas, `x1` dan `x3` merupakan *Pure Literal*. Maka dari itu hasil dari penerapan fungsi *Pure Literal* pada proposisi di atas ditunjukan pada Gambar 2.7

```
φ = (x4 ∨ ~x5) ∧ (x5 ∨ ~x4)
```

Gambar 2.7 Penerapan *pure literal* pada logika proposisi

c. Decide

Fungsi *decide* juga sering disebut juga dengan *splitting* dan dilakukan apabila dalam suatu proposisi sudah tidak bisa lagi diselesaikan dengan *Unit Propagation* dan *Pure Literal*. Fungsi ini memilih salah satu literal dan memberikannya nilai *true* atau *false* (Moura & Bjorner, 2011). Contoh penggunaan fungsi *decide* ditunjukkan pada Gambar 2.8.

```
⇨  (p ∨ ~q) ∧ (~p ∨ q) ∧ (~p ∨ ~q) (Decide p=false)
⇨  ~q (Unit Propagation)
⇨  ⊤ (satisfiable)
```

Gambar 2.8 Contoh penerapan fungsi *decide*

d. *Backtracking*

Fungsi *backtracking* dilakukan pada operasi decide. Apabila pemberian nilai variabel pada suatu formula tidak menghasilkan hasil yang *satisfied*, maka formula dikembalikan pada kondisi semula. Kemudian variabel yang sama diberi lagi nilai yang kontradiktif untuk diuji kembali nilai *satisfiable*-nya. Contoh penggunaan fungsi *backtracking* ditunjukkan pada Gambar 2.9.

```
(p ∨ ~q) ∧ (~p ∨ q) ∧ (~p ∨ ~q) (Decide p=true)
q ∧ ~q (Unit Propagation)
⊥ (unsatisfiable)(Backtrack)
(p ∨ ~q) ∧ (~p ∨ q) ∧ (~p ∨ ~q) (Decide p=false)
~q (Unit Propagation)
⊤ (satisfiable)
```

Gambar 2.9 Contoh penerapan fungsi *backtracking*

2.2.5 Sudoku

Sudoku (dikenal juga sebagai *Number Place* atau *Nanpure*) adalah sejenis permainan teka-teki logika. Tujuannya adalah untuk memasukkan angka dari 1 – 9 ke dalam jaring-jaring 9 x 9 yang terdiri dari 9 kotak 3 x 3 tanpa ada angka yang berulang dalam suatu kolom, baris, maupun kotak 3 x 3 tersebut. Nama Sudoku sendiri sudah menjelaskan maksud dari permainan itu sendiri, yang berasal dari Bahasa Jepang "*Suuji wa dokushin ni kagiru*" yang berarti “angka-angkanya harus tetap tunggal”. Adapun contoh soal Sudoku dan jawabannya ditunjukkan pada Gambar 2.10.

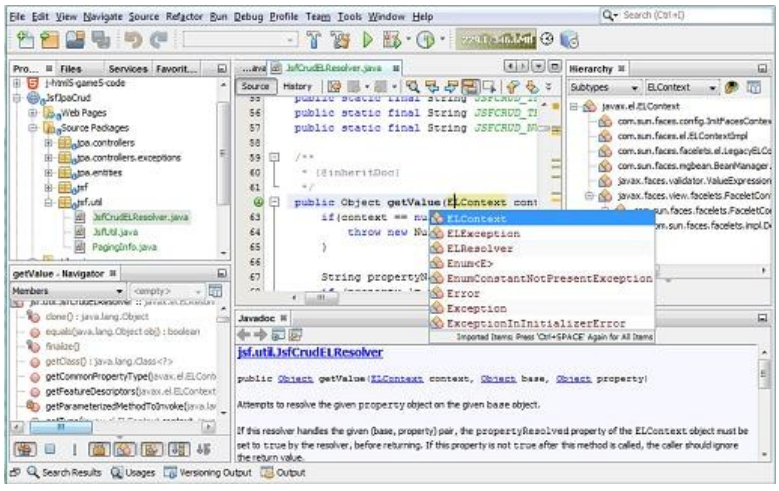
2		9				6		
	4		8	7			1	2
8				1	9		4	
	3		7			8		1
	6	5			8		3	
1				3				7
			6	5		7		9
6		4					2	
	8		3		1	4	5	

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

Gambar 2.10 Soal Sudoku (kiri) beserta jawabannya (kanan)

2.2.6 Netbeans IDE

Netbeans merupakan proyek yang disponsori oleh Sun Microsystem sejak tahun 2000. Pada proyek tersebut dihasilkan dua produk, yaitu Netbeans IDE dan Netbeans Platform. Netbeans IDE merupakan produk yang digunakan untuk melakukan pemrograman, baik menulis kode, mengompilasi, mencari kesalahan, dan mendistribusikan program. Sedangkan Netbeans Platform adalah sebuah modul yang merupakan kerangka awal dalam membangun aplikasi desktop yang besar. Netbeans merupakan salah satu IDE yang dapat digunakan dalam melakukan pemrograman Java. Selain itu, Netbeans menyediakan paket yang lengkap dalam pemrograman, dari pemrograman standar, *enterprise*, dan pemrograman perangkat *mobile* yang dapat berjalan di berbagai macam platform. Tampilan Netbeans IDE dapat dilihat pada Gambar 2.11.



Gambar 2.11 Tampilan Netbans IDE

2.2.7 Java

Java adalah bahasa pemrograman tingkat tinggi yang berorientasi objek. Program java tersusun dari bagian yang disebut kelas. Kelas terdiri atas metode-metode yang melakukan pekerjaan dan mengembalikan informasi setelah melakukan tugasnya. Dalam setiap kelas juga terdapat atribut yang merupakan sifat berupa nilai atau kondisi yang dimiliki kelas tersebut.

BAB III
METODOLOGI

3.1 Analisis Kebutuhan

Analisis kebutuhan merupakan tahapan untunk melakukan proses pengumpulan data dan informasi yang akan digunakan untuk mendukung atau menunjang pembuatan sistem aplikasi yang akan dibuat serta dapat memperoleh jawaban dari rumusan masalah yang telah dibentuk sebelumnya. Metode pengumpulan data yang dilakukan penulis menggunakan metode studi pustaka. Studi pustaka adalah sebuah metode dalam pengumpulan data dengan melakukan pencarian informasi melalui media seperti buku, jurnal atau internet. Studi pustaka dilakukan dengan membaca buka dan mencari literatur dari internet mengenai sistem yang dalam penelitian ini membahas tentang *Boolean Satisfiability Problem* dan penyelesaiannya dengan menggunakan Algoritma Davis-Putnam-Logemann-Loveland, mencari informasi mengenai perangkat keras ataupun perangkat lunak yang sesuai agar sistem dapat berjalan dan berfungsi sesuai dengan yang diharapkan.

3.1.1 Analisis Kebutuhan Fungsi

Analisis kebutuhan fungsi adalah tahapan di mana dilakukan penetapan fungsi yang dapat

dilakukan oleh sistem, sehingga dapat menjawab rumusan masalah yang ada. Sistem ini nantinya akan memiliki fungsi sebagai berikut:

a. Membaca proposisi dalam bentuk *Conjunctive Normal Form*.
b. Menyelesaikan permasalahan SAT *Problem*.
c. Menyelesaikan permasalahn Sudoku untuk pengujian.

3.1.2 Analisis Kebutuhan Masukan

Pada tahap analisis kebutuhan masukan ini dilakukan tahapan untuk menentukan masukan apa yang dibutuhkan dalam pembuatan aplikasi untuk penyelesaian SAT *Problem* ini.

a. Masukan yang diberikan untuk penyelesaian SAT *Problem* ini adalah masukan yang sudah sesuai dengan standar CNF. Adapun standar CNF yang dimaksud terdiri dari

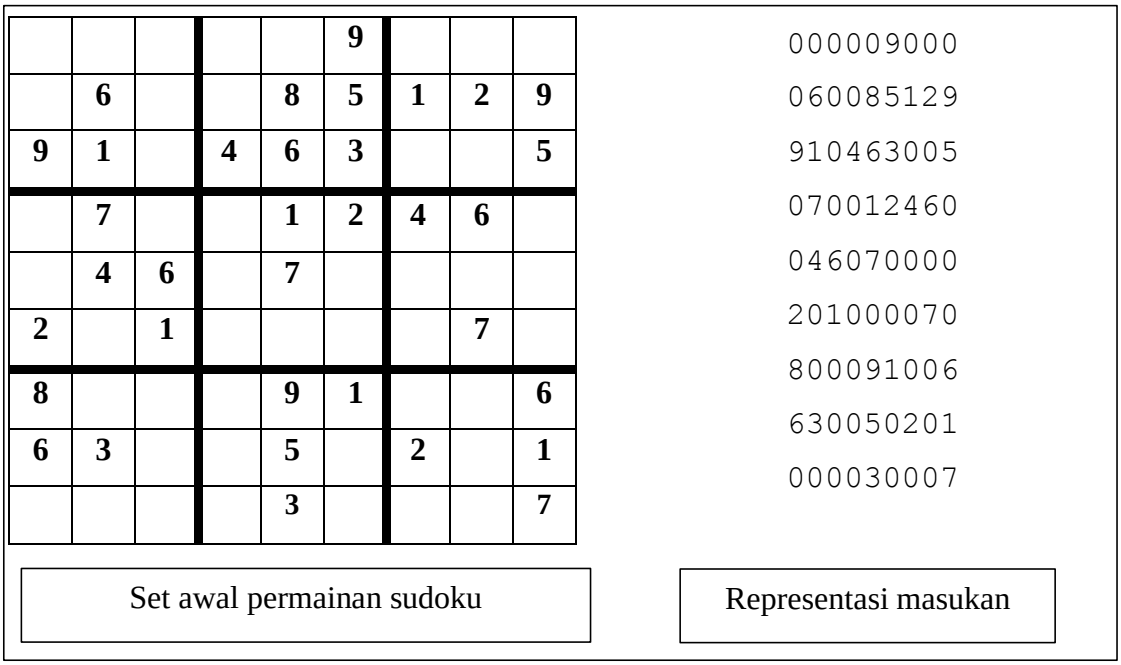
komentar, parameter, dan variabel yang diberikan dalam bentuk integer. Baris awal sebuah masukan yang diawali oleh sebuah karakter ‘c’ berarti sebuah komentar yang tidak perlu dibaca oleh program. Komentar tersebut bisa berisi identitas dari pembuat berkas masukan tersebut maupun keterangan tambahan yang diperlukan. Baris selanjutnya adalah baris yang diawali oleh sebuah karakter ‘p’ yang berarti parameter mengenai keterangan banyaknya variabel dan klausa yang terdapat pada file masukan tersebut. Variabel yang diberikan pada masukan berformat CNF direpresentasikan dalam bentuk integer. *Conjunction* antar klausa direpresentasikan dengan angka 0, sedangkan *disjunction* antar literal direpresentasikan dengan spasi. Untuk variabel yang memiliki nilai negasi ditandai dengan penambahan simbol negatif (-). Contoh konversi masukan logika proposisi menjadi format CNF bisa dilihat pada Tabel 3.1 berikut ini.

Tabel 3.1 Konversi Logika Prosposisi menjadi Format CNF

Keterangan	**Logika Proposisi**	**Konversi CNF**
Setiap Variabel diubah menjadi Integer	(p ∨ ~q) ∧ (~p ∨ q) ∧ (~p ∨ ~q)	(1 ∨ ~2) ∧ (~1 ∨ 2) ∧ (~1 ∨ ~2)
Simbol negasi diubah menjadi negatif	(1 ∨ ~2) ∧ (~1 ∨ 2) ∧ (~1 ∨ ~2)	(1 ∨ -2) ∧ (-1 ∨ 2) ∧ (-1 ∨ -2)
Simbol disjungsi diubah menjadi spasi	(1 ∨ -2) ∧ (-1 ∨ 2) ∧ (-1 ∨ -2)	(1 -2) ∧ (-1 2) ∧ (-1 -2)
Simbol konjungsi diubah menjadi angka 0	(1 -2) ∧ (-1 2) ∧ (-1 -2)	(1 -2) 0 (-1 2) 0 (-1 -2)
Tanda kurung dihilangkan	(1 -2) 0 (-1 2) 0 (-1 -2)	1 -2 0 -1 2 0 -1 -2
Setiap klausa dipisahkan denang baris baru	1 -2 0 -1 2 0 -1 -2	1 -2 0 -1 2 0 -1 -2

b. Formula masukan yang diberikan berbentuk file dengan format .txt.

c. Formula masukan Sudoku yang sesuai dengan standar *Conjunctive Normal Form* untuk pengujian. Masukan yang diberikan untuk penyelesaian permasalahan Sudoku dibuat mirip dengan penerapan Sudoku yang sebenarnya. Setiap masukan *file* terdiri dari sembilan baris. Masing-masing baris terdiri dari sembilan integer yang mewakili nilai awal sebuah *cell* pada Sudoku. *Cell* yang masih kosong direpresentasikan dengan angka nol (0). Contoh masukan Sudoku ditunjukkan pada Gambar 3.1.



Gambar 3.1 Masukan Sudoku

3.1.3 Analisis Kebutuhan Keluaran

Pada penelitian ini akan dibuat sebuah aplikasi yang akan menentukan masukan yang diberikan apakah *satisfiable* atau *unsatisfiable*. Kebutuhan keluaran dalam penelitian ini adalah sebagai berikut:

a. Sebuah keluaran string berupa "*satisfiable*" atau "*unsatisfiable*".

b. Nilai-nilai yang harus diberikan kepada setiap variabel apabila proposisi tersebut bernilai *satisfiable*.

c. Nilai-nilai yang harus diberikan kepada setiap *cell* apabila masukan Sudoku yang diberikan dapat diselesaikan.

3.1.4 Analisis Kebutuhan Perangkat Lunak

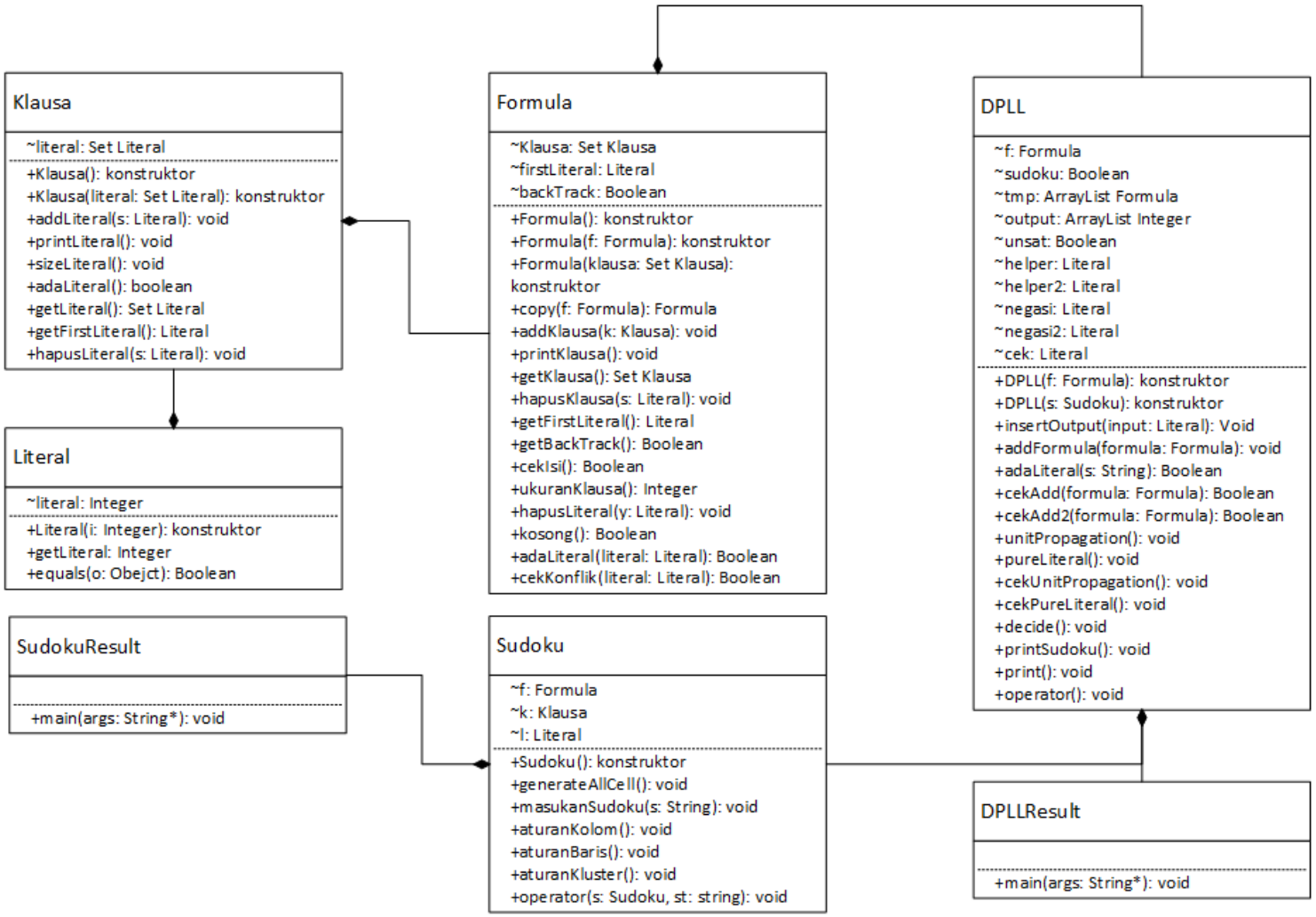
Penelitian ini menggunakan perangkat lunak dalam menuliskan kode program. Adapun perangkat lunak yang digunakan dalam penilitian ini adalah Netbeans IDE.

3.2 Metode Perancangan

Metode perancangan merupakan suatu cara untuk menjelaskan perancangan dari sebuah penelitian. Tahap ini merupakan tahapan penting sebelum lanjut ke tahap pengerjaan dalam sebuah penelitian. Pada tahap ini akan didapatkan metode yang akan digunakan dalam penelitian, sehingga bisa mengidentifikasi dan mengevaluasi kendala-kendala dalam pengembangan aplikasi.

3.2.1 Diagram Kelas

Diagram kelas adalah diagram yang digunakan untuk menampilkan beberapa kelas serta paket-paket yang ada dalam suatu sistem atau perangkat lunak yang sedang dikembangkan. Diagram kelas memberikan sebuah gambaran statis tentang sistem/perangkat lunak beserta relasi-relasi yang terdapat di dalam suatu sistem/perangkat lunak tersebut. Di dalam diagram kelas juga terdapat deskripsi dari masing-masing objek berupa properti, metode, dan relasi. Diagram kelas dalam aplikasi ini ditunjukkan pada Gambar 3.2.



Gambar 3.2 Diagram kelas

Pada Gambar 3.2 terdapat tujuh kelas, yaitu DPLL, Formula, Klausa, Literal, Sudoku, DPLLResult dan SudokuResult. Adapun fungsi dari kelas-kelas tersebut akan dijelaskan sebagai berikut:

a. Kelas Literal

Kelas Literal berfungsi untuk menyimpan nilai literal yang berasal dari masukan yang diberikan. Nilai dari literal tersebut akan disimpan dalam variabel dengan tipe data integer. Adapun atribut dan method dalam Kelas Literal ditunjukkan oleh Tabel 3.2.

Tabel 3.2 Kelas Literal

Atribut / *Method*	**Tipe Data**	**Parameter Formal**	**Keterangan**
literal	Integer	-	Menyimpan nilai literal.
Literal	Konstruktor	i: Integer	Memberikan nilai pada variabel literal
getLiteral	Integer	-	Memberikan nilai return literal.
equals	Boolean	o: Object	Memberikan nilai return Boolean apabila literal pada parameter aktual bernilai sama dengan object literal

b. Kelas Klausa

Kelas Klausa berfungsi untuk menyimpan nilai dari kelas literal. Dalam satu Kelas Klausa dapat memiliki satu atau lebih literal. Adapun atribut dan method dalam Kelas Klausa ditunjukkan oleh Tabel 3.3.

Tabel 3.3 Kelas Klausa

Atribut / *Method*	**Tipe Data**	**Parameter Formal**	**Keterangan**
literal	Set Literal	-	Menyimpan kelas literal

Klausa	Konstruktor	-	Membuat objek literal.
Klausa	Konstruktor	literal: Set Literal	Membuat objek literal dan memberikan nilai literal
addLiteral	Void	S: Literal	Menambahkan literal
printLiteral	Void	-	Mencetak literal dalam satu klausa
sizeLiteral	Integer	-	Memberikan nilai return banyaknya literal pada objek klausa
adaLiteral	Boolean	-	Memberikan nilai return Boolean ada atau tidak adanya literal pada objek klausa
getLiteral	Set Literal	-	Memberikan nilai return objek literal
getFirstLiteral	Literal	-	Memberikan nilai return literal pertama
hapusLiteral	Void	s: Literal	Menghapus literal

c. Kelas Formula

Kelas Formula berfungsi untuk menampung nilai-nilai dari Kelas Klausa. Dalam satu Kelas Formula dapat memiliki satu atau lebih klausa. Adapun atribut dan method dalam Kelas Formula ditunjukkan oleh Tabel 3.4.

Tabel 3.4 Kelas Formula

Atribut / *Method*	**Tipe Data**	**Parameter Formal**	**Keterangan**
Klausa	Set Klausa	-	Menyimpan variabel klausa.
firstLiteral	Literal	-	Menyimpan nilai literal pertama
backTrack	Boolean	-	Memberikan nilai return kondisi backtrack
Formula	Konstruktor	-	Membuat objek klausa
Formula	Konstruktor	f: Formula	Membuat objek formula baru
Formula	Konstruktor	klausa: set Klausa	Membuat objek formula dan memberikan nilai klausa
copy	Formula	f: Formula	Menggandakan sebuah objek formula
addKlausa	Void	k: Klausa	Menambahkan klausa
printKlausa	Void	-	Mencetak nilai formula
getKlausa	Set	-	Memberikan nilai return klausa
hapusKlausa	Void	s: Literal	Menghapus klausa

			yang memiliki literal pada parameter aktual
getFirstLiteral	Literal	-	Memberikan nilai return literal pertama
getBacktrack	Boolean	-	Memberikan nilai return Boolean backtrack
cekIsi	Boolean	-	Memberikan nilai return Boolean apakah objek formula memiliki klausa ataupun literal
ukuranKlausa	Integer	-	Memberikan nilai return ukuran klausa pada objek formula
hapusLiteral	Void	y: Literal	Menghapus literal yang memiliki nilai sama dengan parameter aktual
Kosong	Boolean	-	Memberikan nilai return Boolean apakah objek Formula tidak memiliki klausa
adaLiteral	Boolean	literal: Literal	Memberikan nilai Boolean apakah pada objek formula terdapat literal yang sama dengan parameter aktual

d. Kelas DPLL

Kelas DPLL memiliki atribut dan method yang berfungsi untuk mengimplementasikan algoritma DPLL. Algortima DPLL dalam kelas DPLL ini berada dalam method *solver*. Adapaun atribut dan method dalam Kelas DPLL ditunjukkan pada Tabel 3.5.

Tabel 3.5 DPLL

Atribut / *Method*	**Tipe Data**	**Parameter Formal**	**Keterangan**
f	Formula	-	Menyimpan nilai formula ke dalam kelas formula
Sudoku	Boolean	-	Menentukan apakah masukan yang diberikan berupa formula atau Sudoku. Pembeda ini hanya digunakan untuk menentukan bentuk keluaran yang akan diberikan
tmp	Array List Formula	-	Duplikasi formula untuk *backtracking*
output	Array List Integer	-	Menyimpan nilai output yang akan digunakan untuk file output yang berisi solusi dari masukan yang diberikan
unsat	Boolean	-	Memberikan nilai Boolean apakah formula yang diberikan bernilai *satisfiable* atau

			unsatisfiable
helper	Literal	-	Menyimpan nilai literal yang memenuhi syarat untuk menjalankan operasi Unit Propagation
helper2	Literal	-	Menyimpan nilai literal yang memenuhi syarat untuk menjalankan operasi Pure Literal
negasi	Literal	-	Menyimpan nilai negasi dari literal pada variabel helper
negasi2	Literal	-	Menyimpan nilai negasi dari literal pada variabel helper2
cek	Literal	-	Menyimpan nilai literal untuk mengecek apakah terjadi konflik pada objek formula
DPLL	Konstruktor	f: Formula	Membuat objek baru kelas DPLL untuk penyelesaian SAT *Problem*
DPLL	Konstruktor	s: Sudoku	Membuat objek baru kelas DPLL untuk penyelesaian Sudoku
insertOutput	Void	input: Literal	Menyimpan nilai keluaran
addFormula	Void	formula: Formula	Menambahkan

			formula
cekKonflik	Boolean	-	Memberikan nilai return Boolean apakah objek formula terdapat klausa yang saling konflik
cekAdd	Boolean	formula: Formula	Memberikan nilai return Boolean apakah formula yang diselesaikan dengan operasi decide memberikan nilai konflik
cekAdd2	Boolean	formula: Formula	Memberikan nilai return Boolean apakah formula yang diselesaikan dengan operasi decide memberikan nilai konflik
unitPropagation	Void	-	Menjalankan operasi Unit Propagation
pureLiteral	Void	-	Menjalankan operasi Pure Literal
cekUnitPropagation	Boolean	-	Mengecek apakah formula dapat diselesaikan dengan Unit Propagation
cekPureLiteral	Boolean	-	Mengecek apakah formula dapat diselesaikan dengan Pure Literal

decide	Void	-	Menjalankan operasi Decide
printSudoku	Void	-	Mencetak keluaran untuk saran penyelesaian sudoku
print	Void	-	Mencetak keluaran untuk saran SAT*Problem*
solver	Void	-	Prosedur yang berfungsi untuk menjalankan algoritma DPLL. Method ini berfungsi untuk mengimplementasikan algoritma DPLL. Keluaran dari method ini akan disimpan di variabel output dan akan di ekspor dalam bentuk teks.

e. Kelas Sudoku

Kelas ini pada dasarnya adalah kelas yang berguna untuk melakukan fungsi *encoding* dari masukan yang diberikan. Masukan yang masih dalam bentuk teks Sudoku di-*encoding* lagi dan akan diselesaikan dengan menggunakan algoritma DPLL. Kelas ini juga berfungsi sebagai pengujian untuk menyelesaikan permasalahan Sudoku menggunakan algortima DPLL. Adapaun atribut dan method yang terdapat dalam Kelas Sudoku ditunjukkan pada Tabel 3.6.

Tabel 3.6 Sudoku

Atribut / *Method*	**Tipe Data**	**Parameter Formal**	**Keterangan**
f	Formula	-	Menyimpan objek formula
k	Klausa	-	Menyimpan objek klausa
l	Literal	-	Menyimpan objek literal
Sudoku	Konstruktor	-	Membuat objek formula
generateAllCell	Void	-	Decoding CNF aturan cell sudoku
masukanSudoku	Void	s: String	Decoding CNF input sudoku
aturanKolom	Void	-	Decoding CNF aturan kolom sudoku
aturanBaris	Void	-	Decoding CNF aturan baris sudoku
aturanKlaster	Void	-	Decoding CNF aturan klaster sudoku
solver	Void	-	Prosedur yang berfungsi untuk menjalankan algoritma DPLL untuk menyelesaikan sudoku

3.2.2 Perancangan Keluaran Permasalahan SAT *Problem*

Pada aplikasi ini, keluaran yang dihasilkan terbagi menjadi dua jenis. Keluaran yang pertama adalah untuk masukan yang berupa logika proposisi untuk permaslaahan SAT *Problem*. Keluaran yang kedua adalah untuk permasalahan Sudoku.

Keluaran untuk permasalahan SAT Problem berupa teks "*Unsatisfiable*" apabila masukan yang diberikan bersifat *unsatisfiable*. Akan tetapi apabila masukan yang diberikan bersifat *satisfiable*, maka aplikasi akan mengeluarkan teks "*Satisfiable*" diikuti dengan nilai-nilai yang harus diberikan pada variabel agar logika proposisi tersebut bernilai *satisfiable*. Adapun format mengenai rancangan tersebut ditunjukkan pada Gambar 3.3.

```
Input:
1 -1 0
2 3 0
4 0

Output:
Satisfiable
-1
2
4
```

Gambar 3.3 Keluaran SAT Problem

Seperti yang terlihat pada Gambar 3.4, apabila formula yang diberikan bernilai *satisfiable*, maka akan menampilkan nilai dari variabel-variabel yang diberikan agar dapat menghasilkan formula yang *satisfied*. Adapun maksud dari keluaran tersebut adalah variabel 1 bernilai *false*, variabel 2 & 4 bernilai *true*, sedangkan variabel 3 yang tidak muncul dalam keluaran bebas diberikan nilai *true* atau *false*.

3.2.3 *Encoding* Sudoku

Dalam penyelesaian permasalahan Sudoku ini, aplikasi menggunakan metode dan algoritma yang sama untuk penyelesaian permasalahan SAT *Problem*. Namun untuk penyelesaian Sudoku, masukan yang diberikan harus di-*encoding* terlebih dahulu sehingga berbentuk menjadi logika proposisi dan diselesaikan menggunakan algoritma DPLL.

a. *Encoding* nilai awal

   Dalam setiap permainan Sudoku, terdapat ada angka-angka awal yang sudah disematkan yang berfungsi sebagai acuan untuk melengkapi jaring-jaring yang masih kosong. *Encoding* pada setiap angka-angka awal tersebut dapat dilihat pada Gambar 3.4.

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$$

Gambar 3.4 *Encoding* pada setiap angka awal permainan Sudoku

Dalam setiap variasi permainan Sudoku akan memiliki angka awal yang berbeda, meskipun juga tidak menutup kemungkinan Sudoku tidak memiliki angka awal.

b. *Encoding* baris

Dalam setiap permainan Sudoku, tidak boleh ada angka yang sama pada setiap baris. Maka dari itu, aturan *encoding* baris dapat dilihat pada Gambar 3.5.

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$$

Gambar 3.5 *Encoding* tiap baris tidak boleh ada angka yang sama

Pada setiap baris juga tidak boleh terdapat jaring-jaring yang kosong, atau dengan kata lain pada setiap jaring-jaring di setiap baris harus diisi oleh angka. Maka dari itu, aturan *encoding* untuk hal tersebut dapat dilihat pada Gambar 3.6:

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{x=1}^{9} s_{xyz}$$

Gambar 3.6 *Encoding* tiap baris tidak boleh terdapat jaring-jaring yang kosong

c. *Encoding* kolom

Dalam setiap permainan Sudoku, tidak boleh ada angka yang sama pada setiap kolom. Maka dari itu, aturan *encoding* kolom dapat dilihat pada Gambar 3.7.

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$$

Gambar 3.7 *Encoding* tiap kolom tidak boleh ada angka yang sama

kata lain ıran *encoding* untuk hal tersebut dapat dilihat pada Gambar 3.8.

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigvee_{y=1}^{9} s_{xyz}$$

Gambar 3.8 *Encoding* setiap kolom tidak boleh terdapat jaring-jaring yang kosong

d. *Encoding* klaster

Dalam setiap permainan Sudoku, tidak boleh ada angka yang sama pada setiap klaster. Maka dari itu, aturan *encoding* klaster dapat dilihat pada Gambar 3.9.

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$$
$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z}$$

Gambar 3.9 *Encoding* tiap klaster tidak boleh ada angka yang sama

Pada setiap klaster juga tidak boleh terdapat jaring-jaring yang kosong, atau dengan kata lain pada setiap jaring-jaring di setiap klaster harus diisi oleh angka. Maka dari itu, aturan *encoding* untuk hal tersebut dapat dilihat pada Gambar 3.10.

$$\bigvee_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} s_{(3i+x)(3j+y)z}.$$

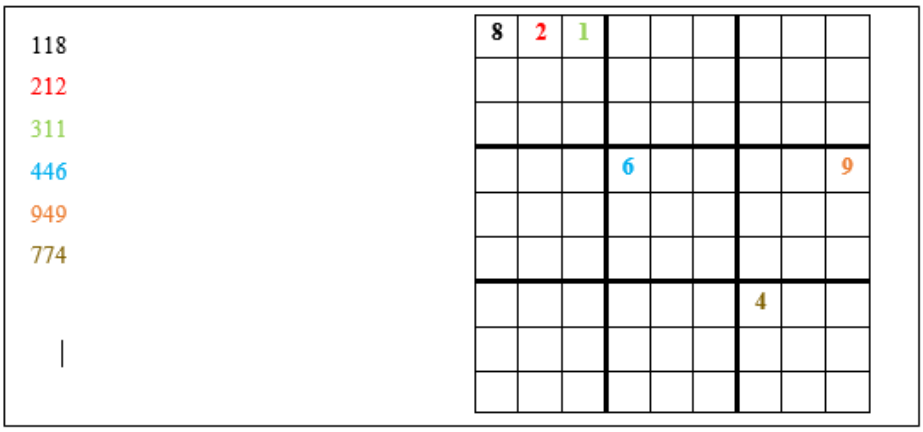
Gambar 3.10 *Encoding* setiap klaster tidak boleh terdapat jaring-jaring yang kosong

3.2.4 Keluaran Sudoku

Keluaran untuk permasalahan Sudoku sama seperti keluaran SAT *Problem*. Akan tetapi keluaran tersebut direpresentasikan dalam bentuk baris dan kolom yang berukuran 9 x 9 agar menggambarkan papan pada permainan Sudoku. Adapun prerepresentasian tersebut menggunakan nilai-nilai yang terdapat pada literal yang dihasilkan dari masukan yang telah diberikan.

Hasil dari *encoding* berupa literal-literal yang bernilai integer, yang akan diselesakaian dengan algoritma DPLL untuk dicari solusinya agar bernilai *satisfied*. Adapaun literal-literal hasil *encoding* tersebut memiliki tiga digit. Digit pertama mewakili kolom, digit kedua mewakili baris, digit ketiga mewakili nilai yang terdapat pada kolom dan baris di papan Sudoku tersebut. Literal yang merupakan jawaban dari permainan Sudoku adalah literal yang

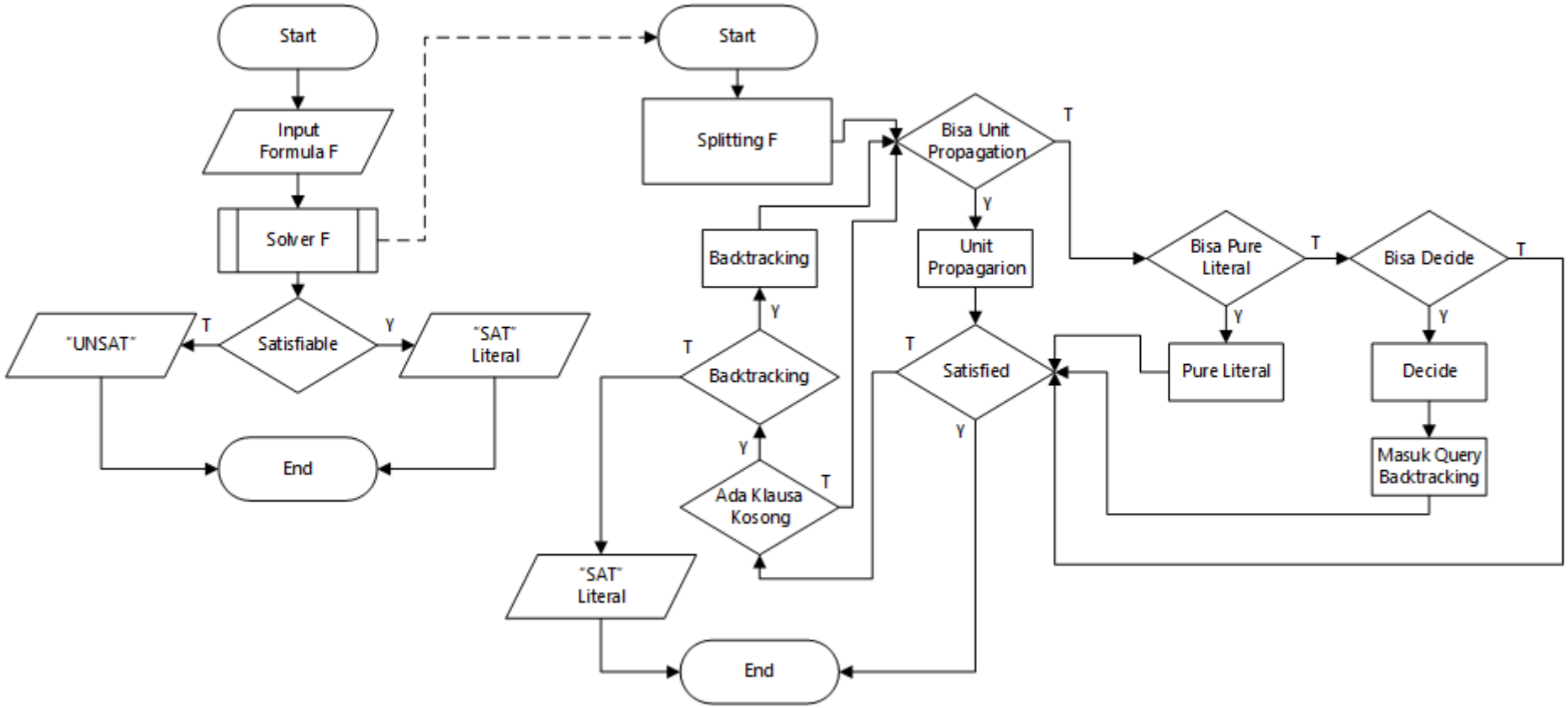
memiliki nilai *true*. Adapaun representasi literal ke papan Sudoku dapat dilihat pada Gambar 3.11 berikut.



Gambar 3.11 Representasi literal terhadap nilai pada papan Sudoku

3.2.5 Flowchart

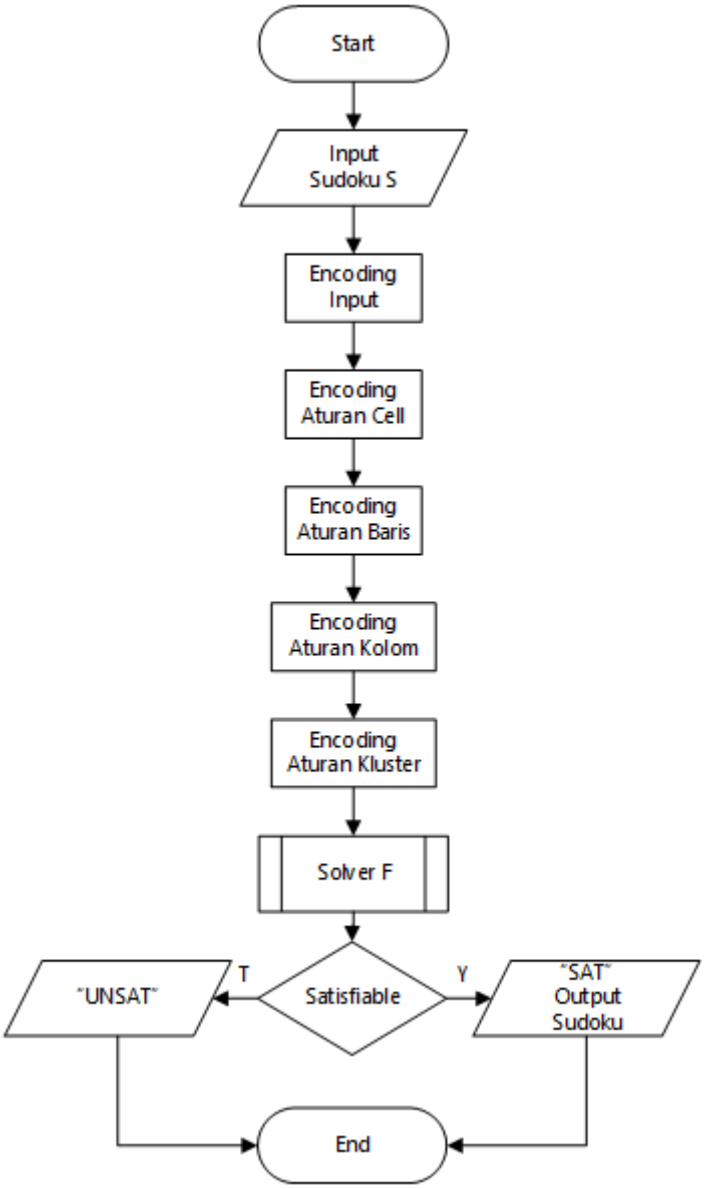
Flowchart adalah gambaran dalam bentuk diagram alir dari algoritma dalam suatu program yang menyatakan arah alur program dalam menyelesaikan suatu masalah. Dalam penelitian ini, *flowchart* dibuat untuk menjelaskan implementasi algoritma Davis-Putnam-Logemann-Loveland (DPLL) agar lebih mudah mengubahnya ke dalam kode program. Dalam penelitian ini *flowchart* dibagi menjadi dua, yaitu *flowchart method solver* dalam Kelas DPLL dan *flowchart method solver* dalam Kelas Sudoku. Masing-masing method tersebut mengimplementasikan algoritma DPLL. Adapun *flowchart* method solver dalam Kelas DPLL ditunjukkan pada Gambar 3.12.



Gambar 3.12 *Flowchart method solver* Kelas DPLL

Pada Gambar 3.12 setelah formula dimasukkan, algoritma ini akan melakukan proses *splitting* atau pemisahan terhadap formula yang diberikan. *Splitting* berfungsi untuk membagi formula yang dimasukkan ke dalam klausa dan literal. Seletah dilakukan proses splitting, terlihat tiga operasi algoritma DPLL digunakan secara berurutan, mulai dari *Unit Propagation*, *Pure Literal*, dan *Decide*. Pada saat akan menggunakan operasi *Unit Propagation* ataupun *Pure Literal*, terlebih dahulu dilakukan pengecekan apakah operasi itu mungkin dilakukan pada formula yang diberikan. Jika memungkinkan, maka operasi tersebut dilakukan, jika tidak maka akan berlanjut ke operasi berikutnya. Proses tersebut dilakukan terus menerus sampai dapat membuktikan suatu formula yang diberikan *satisfiable* atau tidak.

Untuk penyelesaian permasalahan Sudoku sebagai uji coba, alur program yang digunakan hampir sama dengan alur program yang digunakan untuk menyelesaikan permasalah SAT *Problem*. Hanya saja ada tambahan proses *encoding* dari masukan Sudoku yang diberikan menjadi format CNF. *Flowchart* method solver dalam Kelas Sudoku oleh Gambar 3.13.



Gambar 3.13 *Flowchart* method solver Kelas Sudoku

Pada Gambar 3.13 terlihat bahwa pada awal alur program dilakukan proses *encoding*. *Encoding* yang dilakukan meliputi *encoding* masukan Sudoku, aturan cell, aturan baris, aturan kolom, dan aturan Klaster. Setelah semua aturan di *encoding* menjadi format CNF, kemudian CNF tersebut diselesaikan dengan menggunakan algoritma DPLL. Pada *flowchart* tersebut juga terdapat proses "Solver F" yang isinya sama dengan proses "Solver F" pada Gambar 3.12.

BAB IV
IMPLEMENTASI DAN PENGUJIAN

4.1 Implementasi

Implementasi adalah tahap selanjutnya untuk melakukan apa yang telah dirancang pada tahapan di bab sebelumnya. Implementasi akan menunjukkan apakah perancangan yang telah dilakukan sebelumnya dapat berjalan dan dapat digunakan dengan baik. Pada tahapan implementasi ini dibagi menjadi dua, yaitu implementasi penyelesaian SAT *Problem* dengan menggunakan algoritma DPLL dan penyelesaian Sudoku sebagai SAT *Problem* dengan menggunakan algoritma DPLL.

4.1.1 Implementasi Algoritma DPLL

Algoritma DPLL diimplementasikan dalam sebuah kelas yang bernama kelas DPLL. Di dalam kelas tersebut terdapa *method* yang berfungsi untuk menjalankan algoritma DPLL. Adapun *method-method* tersebut adalah sebagai berikut ini:

a. *Method* Unit Propagation

Method Unit Propagation dapat dilakukan apabila dalam suatu formula, terdapat klausa yang hanya memiliki satu literal. Untuk mengetahui hal tersebut, maka dibuatlah *method* cekUnitPropagation yang akan memberikan nilai *true* apabila Unit Propagation dapat dioperasikan pada formula tersebut. Adapun *method* cekUnitPropagation dapat dilihat pada Gambar 4.1.

```
public boolean cekUnitPropagation(){
        for(Klausa k:f.getKlausa()){
            if(k.literal.size()==1){
                helper = k.getFirstLiteral();
                return true;
            }
        }return false;
    }
```

Gambar 4.1 *Method* cekUnitPropagation

Ketika *method* cekUnitPropagation memberikan nilai *true*, maka selanjutnya program akan menjalankan operasi unitPropagation. Adapun *method* unitPropagation dapat dilihat pada Gambar 4.2.

```
   void unitPropagation(){
           negasi = new Literal(helper.literal*(-1));
           f.hapusKlausa(helper);
           f.hapusLiteral(negasi);
           insertOutput(helper);
           System.out.println("UP: "+helper.literal);
       }
```

Gambar 4.2 *Method* unitPropagation

b. *Method* Pure Literal

Method Pure Literal dapat dilakukan apabila dalam suatu formula, terdapat satu variabel yang hanya memiliki nilai *true* saja atau *false* saja. Untuk mengetahui hal tersebut, maka dibuatlah *method* cekPureLiteral yang akan memberikan nilai *true* apabila Pure Literal dapat dioperasikan pada formula tersebut. Adapun *method* cekPureLiteral dapat dilihat pada Gambar 4.3.

```
public boolean cekPureLiteral(){
   for(Klausa k : f.getKlausa()){
      for(Literal literal : k.getLiteral()){
         negasi2 = new Literal(literal.literal*(-1));
         if(!f.adaLiteral(negasi2) && f.ukuranKlausa()>1 ){
            helper2 = literal;
            return true;
         }
```

Gambar 4.3 *Method* cekPureLiteral

Ketika *method* cekPureLiteral memberikan nilai *true*, maka selanjutnya program akan menjalankan operasi pureLiteral. Adapun *method* pureLiteral dapat dilihat pada Gambar 4.4.

```
   void pureLiteral(){
      f.hapusKlausa(helper2);
      f.hapusLiteral(negasi2);
      insertOutput(helper2);
      System.out.println("PL: "+helper2.literal);
   }
```

Gambar 4.4 *Method* pureLiteral

c. *Method* Decide

Method decide dilakukan apabila operasi unitPropagation dan pureLiteral tidak dapat digunakan pada formula yang diberikan. Dalam operasi decide juga terdapat operasi *backtracking*. Adapaun *method* decide ditunjukkan pada Gambar 4.5.

```
public void decide(){
   if(cekAdd(f) && !f.backTrack){
      addFormula(f);
      System.out.println("Decide: "+f.firsLiteral.literal);
      negasi = new Literal( f.firsLiteral.literal*(-1));
      f.hapusKlausa(f.firsLiteral);
      f.hapusLiteral(negasi);
      insertOutput(f.firsLiteral);
      if(f.cekIsi()){
         f.firsLiteral=f.getFirstLiteral();
      }
   }else if(cekAdd2(f)){
      if(!f.backTrack){
         f.firsLiteral = new Literal(f.firsLiteral.literal*(-
         1));
```

```
if(tmp.size()==1){
f = new Formula (tmp.get(tmp.size()-1));
f.firsLiteral = new Literal(tmp.get(tmp.size()-
1).firsLiteral.literal*(-1));
tmp.get(tmp.size()-1).backTrack = true;
f.backTrack = tmp.get(tmp.size()-1).backTrack;
tmp.remove(tmp.size()-1);
}
else {
tmp.remove(tmp.size()-1);
f = new Formula (tmp.get(tmp.size()-1));
f.firsLiteral = new Literal(tmp.get(tmp.size()-
1).firsLiteral.literal*(-1));
tmp.get(tmp.size()-1).backTrack = true;
f.backTrack = tmp.get(tmp.size()-1).backTrack;
}
}else{
unsat = true;
}
}
```

Gambar 4.5 *Method* decide

4.1.2 Implementasi Penyelesaian Sudoku dengan Algoritma DPLL

Dalam penyelesaian Sudoku sebagai SAT *Problem* menggunakan Algoritma DPLL, harus dilakukan *encoding* terlebih dahulu. *Encoding* yang dimaksud adalah mengubah nilai awal yang merupakan masukan dari Sudoku, aturan baris, aturan kolom, dan aturan klaster ke dalam bentuk *Conjunctive Normal Form*. Adapun penjelasan mengenai *encoding* tersebut akan dijelaskan seperti berikut ini:

a. Implementesai *encoding* nilai awal Sudoku

   Implementasi pada program Java untuk *encoding* nilai awal Sudoku dapat dilihat pada Gambar 4.6.

```
public void masukanSudoku(String s){
        int i = 0;
        for(int x = 110;x<=990;x+=100){
            for(int y = 1;y<=9;y++){
                if(x%100!=0 && s.charAt(i)!='0'){
                    k = new Klausa();
                    l = new
Literal(x+(y*10)+Integer.parseInt(String.valueOf(s.cha
rAt(i)))-10);
                    k.addLiteral(l);
                }
                if(k!=null && !k.isEmpty()){
                    f.addKlausa(k);
                }
                i++;
            }
        }
    }
```

Gambar 4.6 *Method encoding* nilai awal Sudoku

b. Implementasi *encoding* baris

Implementasi pada program Java untuk *encoding* baris ditunjukkan pada Gambar 4.7.

```
public void aturanBaris(){
        for(int b = 0;b<=8;b++){
            for(int a = 0;a<=80;a+=10){
                k = new Klausa();
                for(int x = 111;x<=911;x+=100){
                    l = new Literal((x+a+b));
                    k.addLiteral(l);
                }f.addKlausa(k);
            }
        }
```

```
        for(int b = 0;b<=8;b++){
            for(int a = 0;a<=80;a+=10){
                for(int x = 111;x<=811;x+=100){
                    for(int y = x+100;y<=911;y+=100){
                        k = new Klausa();
                        l = new Literal((x+a+b)*(-1));
                        k.addLiteral(l);
                        l = new Literal((y+a+b)*(-1));
                        k.addLiteral(l);
                        f.addKlausa(k);
                    }
                }
            }
        }
    }
```

Gambar 4.7 *Method encoding* aturan baris

e. *Encoding* kolom

Implementasi terhadap program Java untuk *encoding* kolom dapat dilihat pada Gambar 4.8 berikut ini.

```
public void aturanKolom(){
        for(int b = 0;b<=800;b+=100){
            for(int a = 0;a<9;a++){
                k = new Klausa();
                for(int x = 111;x<=191;x+=10){
                    l = new Literal((x+a+b));
                    k.addLiteral(l);
                }f.addKlausa(k);
            }
        }
        for(int b = 111;b<=911;b+=100){
            for(int a = 0;a<9;a++){
                for(int x = b;x<=b+99;x+=10){
                    for(int y = x+10;y<=b+99;y+=10){
                        if((y/10)%10!=0){
                            k = new Klausa();
                            l = new Literal((x+a)*(-
1));
                            k.addLiteral(l);
                            l = new Literal((y+a)*(-
1));
                            k.addLiteral(l);
                            f.addKlausa(k);
                        }
                    }
                }
            }
        }
    }
```

Gambar 4.8 *Method encoding* aturan kolom

f. *Encoding* klaster

Implementasi *encoding* klaster terhadap program Java dapat dilihat pada Gambar 4.9 berikut ini.

```
public void aturanKlaster(){
   for(int e = 0;e<=60;e+=30){
      for(int d = 0;d<=600;d+=300){
         for(int c = 0;c<=8;c++){
            k = new Klausa();
            for(int a=0;a<=200;a+=100){
               for(int x = 111;x<=131;x+=10){
                   l = new Literal((x+a+c+d+e));
                   k.addLiteral(l);
               }
            }f.addKlausa(k);
         }
      }
   }
   for(int e = 0;e<=60;e+=30){
      for(int d = 0;d<=600;d+=300){
         for(int c = 0;c<=8;c++){
            for(int a=0;a<=200;a+=100){
               for(int x = 111;x<=131;x+=10){
                  for(int b=0;b<=200;b+=100){
                     for(int y = 111;y<=131;y+=10){
                     if((x+a)!=(y+b) && (x+a)<(y+b)){
                     k = new Klausa();
                     l = new Literal((x+a+c+d+e)*(-1));
                     k.addLiteral(l);
                     l = new Literal((y+b+c+d+e)*(-1));
                     k.addLiteral(l);f.addKlausa(k);
                     }}}}}}}}}
```

Gambar 4.9 *Method encoding* aturan klaster

4.2 Pengujian Penelitian

Tahap pengujian penelitian dilakukan untuk mengetahui apakah aplikasi yang dibuat dapat berjalan dengan baik dan lancar. Dengan dilakukannya pengujian ini, dapat diketahui kelebihan dan kekurangan penerapan algoritma DPLL untuk menyelesaikan permasalahan SAT PROBLEM. Adapun pengujian dilakukan dengan cara sebagai berikut:

a. Melakukan uji penyelesaian permasalahan SAT *Problem* dengan berbagai macam ukuran.
b. Melakukan uji penyelesaian permasalahan SUDOKU.

4.2.1 Pengujian SAT Problem

Dalam pengujian SAT *Problem*, masukan diambil dari *benchmark* yang didapatkan di situs web University Britsih Columbia dengan alamat tautan: https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html. Adapun hasil dari pengujian dapat dilihat pada Tabel 4.1 berikut ini:

Tabel 4.1 Pengujian Penyelesaian SAT PROBLEM

No	Banyak Literal	Banyak Klausa	Waktu	Benchmark	Hasil
1	1	2	0 *seconds*	*Unsatisfied*	*Unsatisfied*
2	2	1	0 *seconds*	*Satisfied*	*Satisfied*
3	2	2	0 *seconds*	*Satisfied*	*Saitsfied*
4	20	91	0 *seconds*	*Satisfied*	*Satisfied*
5	22	25	0 *seconds*	*Satisfied*	*Satisfied*
6	42	133	7 *seconds*	*Unsatisfied*	*Unsatisfied*
7	61	581	2 *seconds*	*Satisfied*	*Satisfied*
8	61	581	2 *seconds*	*Satisfied*	*Satisfied*
9	155	1135	0 *seconds*	*Satisfied*	*Satisfied*
10	265	5666	*Unlimited*	*Satisfied*	-

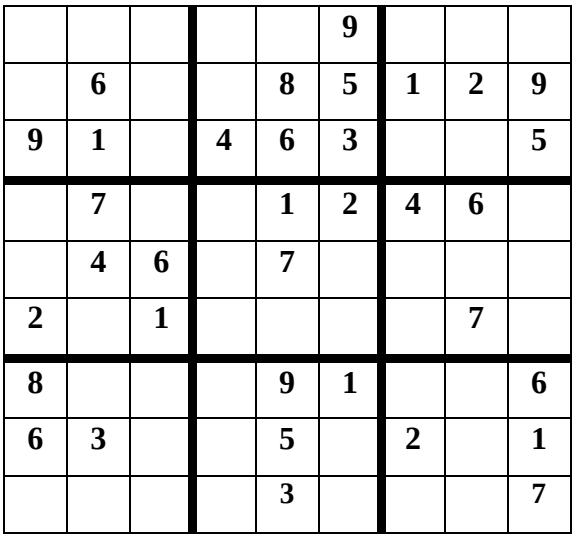
Pada hasil dari tabel pengujian tersebut, dapat dilihat bahwa penyelesaian SAT *Problem* yang memiliki lebih banyak literal dan klausa belum tentu membutuhkan waktu yang lebih lama jika dibandingkan dengan formula yang memiliki literal dan klausa yang lebih sedikit. Hal tersebut dapat terjadi karena perbedaan operator yang dapat dilakukan dalam penyelesaian SAT *Problem*. Operator yang dimaksud dalam hal ini adalah pada operasi *backtracking*.

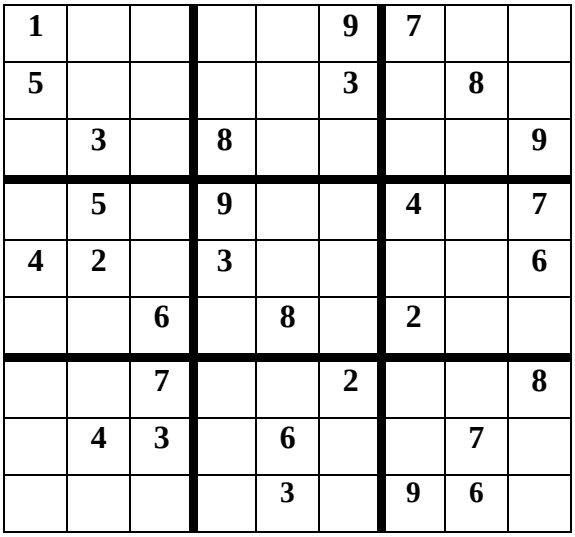
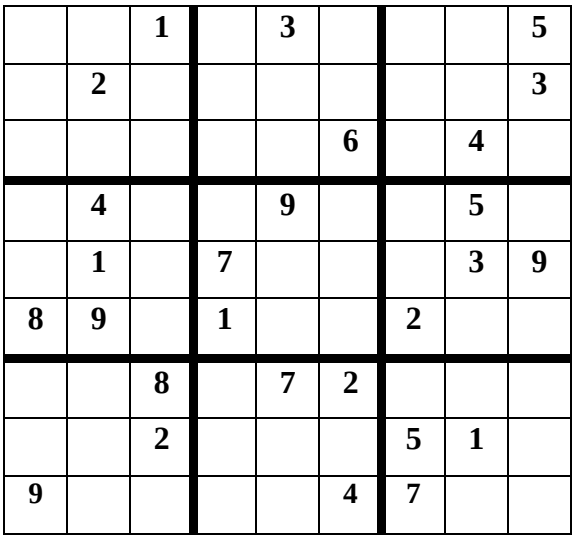
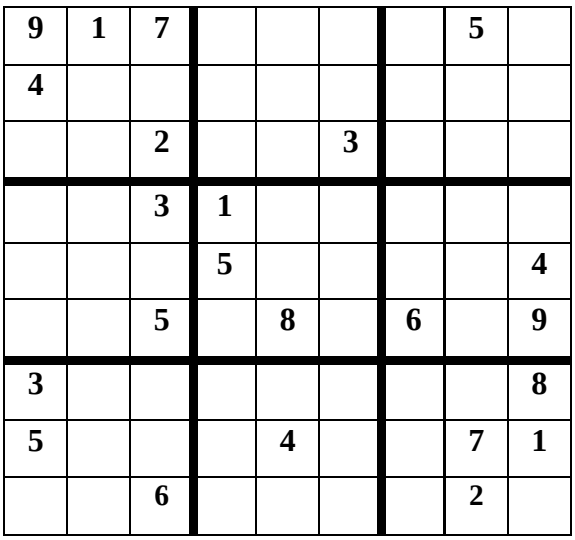
Pada pengujian nomor 10, dapat dilihat bahwa program yang dibuat belum mampu untuk menyelsaikan SAT *Problem* pada formula yang memiliki 265 klausa dan 5666 literal. Hal ini dapat terjadi karena algoritma DPLL memiliki kelemahan, yaitu membutuhkan waktu yang lama dalam menyelesaikan SAT *Problem*, terlebih apabila formula yang diberikan mengharuskan program untuk melakukan *backtracking* yang banyak dan bercabang.

4.2.2 Pengujian Sudoku

Algoritma DPLL dan SAT *Problem* dapat juga digunakan untuk menyelesaikan permainan Sudoku. Untuk menguji program yang dibuat, maka dilakukan uji coba penyelesaian permainan Sudoku tersebut. Adapun hasil dari pengujian dapat dilihat pada Tabel 4.3 berikut ini.

Tabel 4.2 Pengujian Sudoku

No	Set Awal Permainan Sudoku	Level	Format Masukan	Nilai Keluaran
1		Mudah	000009000 060085129 910463005 070012460 046070000 201000070 800091006 630050201 000030007	785|129|634 463|785|129 912|463|785 ---------- 578|912|463 346|578|912 291|346|578 ---------- 857|291|346 634|857|291 129|634|857

2		Menengah	100009700 500003080 030800009 050900407 420300006 006080200 007002008 043060070 000030960	168|249|753 592|673|184 734|815|629 --- 851|926|437 429|357|816 376|481|295 --- 617|592|348 943|168|572 285|734|961
3		Sulit	001030005 020000003 000006040 040090050 010700039 890100200 008072000 002000510 900004700	681|437|925 724|951|683 359|286|147 --- 247|693|851 516|728|439 893|145|276 --- 168|572|394 472|369|518 35|814|762
4		Sangat Sulit	917000050 400000000 002003000 003100000 000500004 005080609 300000008 500040071 006000020	917|468|253 438|275|196 652|913|847 --- 763|194|582 891|526|734 245|387|619 --- 324|751|968 589|642|371 176|839|425

Dalam pengujian tersebut, dapat diketahui bahwa SAT *Problem*, dengan algoritma DPLL dapat digunakan untuk menyelesaikan permainan Sudoku dengan berbagai macam level kesulitan dengan benar. Masukan yang digunakan untuk pengujian terdiri dari beberapa level, yaitu mudah, menengah, sulit, dan sangat sulit. Perbedaan level tersebut ditentukan dari nilai set awal permainan Sudoku. Level mudah memiliki 35 nilai set awal, menengah memiliki 30 nilai set awal, sulit memiliki 27 nilai set awal, dan sangat sulit memiliki 23 nilai set awal. Adapun soal tersebut didapatkan dari situs web dengan alamat *www.sudoku.game*.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan perancangan dan pengujian yang sudah dilakukan, kesimpulan yang dapat diambil dalam penelitian ini adalah sebagai berikut:

a. Program yang dibangun mampu mengimplementasikan Algoritma DPLL untuk menyelesaikan SAT *Problem*.
b. Waktu yang dibutuhkan dalam penyelesaian SAT *Problem* tidak bergantung terhadap banyaknya klausa ataupun literal yang diberikan, tetapi terhadap operasi DPLL yang dapat dilakukan.
c. Semakin banyak operasi *backtracking* yang harus dilakukan, membuat eksekusi program dalam menyelesaikan SAT *Problem* semakin lama.
d. Program yang dibangun dapat menyelesaikan permainan Sudoku menggunakan konsep SAT *Problem* dengan Algoritma DPLL.
e. Bahasa pemrograman Java dengan konsep Pemrograman Berorientasi Objek dapat dengan baik digunakan untuk menyelesaikan permasalahan SAT *Problem*.

5.2 Saran

Saran untuk perbaikan dan pengembangan dari penelitian ini adalah sebagai berikut:

a. Penerapan algoritma yang lebih baik dalam pengimplementasian algoritma DPLL yang digunakan untuk menyelesaikan SAT *Problem*.
b. Penambahan *user interface* sehingga aplikasi dapat digunakan secara lebih luas lagi.
c. Menggunakan konsep yang serupa (penyelesaian SAT *Problem* dengan Algoritma DPLL) untuk menyelesaikan permasalahan yang lain.

DAFTAR PUSTAKA

Astuti, W., & Teguh, R. (2016). *Diktat Logika Informatika.* Palembang: STMIK MDP.

Cook, S. A. (1971). The Comlexity Of Theorem-Proving Procedure. 8.

Darmawan, O., Kusumastuti, N., & Yundari. (2014). KONSTRUKSI PELABELAN SISI AJAIB SUPER PADA GRAF ULAT. 8.

Kalla, P. (2017). The Boolean Satisfiability (SAT) Problem, SAT Solver Technology, and Equivalence Verification. 28.

Karakashian, S., & Puranda, R. (2008). Introduction to NP-Complete Problems. 15.

Larrosa, J., Lynce, I., & Silva, J. M. (2010). Satisfiability: Algorthms, Applications and Extensions. *SAC*, 95.

Monien, B., & Speckenmeyer , E. (1984). Solving Satisfiability In Less Than 2n Steps. *Discrete Applied Mathematics*, 8.

Moura, L. D., & Bjorner, N. (2011). Satisfiablity Modulo Theories: Introduction and Applications. 9.

Nieuwenhuis, R., Tinelli, C., & Oliveras, A. (2006). Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL (T). *ACM*, 43.

Nugraheni, C. E. (2008). Penyelesaian Masalah Penjadwalan Ujian Dengan SAT. *Konferensi Nasional Sisten dan Informatika*, 6.

Ouaknine, J., & Lynce , I. (2006). Sudoku as a SAT Problem. 9.

Vardi, M. Y. (2014). Boolean Satisfiability: Theory and Engineering. *ACM*, 5.

Vij, S. (2016). Sudoku Solver (Using Proportional Logic). 38.

LAMPIRAN