Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Perpustakaan Univeristas Multimedia Nusantara

BAB II

LANDASAN TEORI

2.1 Circuit Satisfiability Problem

Permasalahan yang disebut dengan permasalahan *NP-Complete* adalah suatu permasalahan yang masuk ke kategori permasalahan *NP* (*non-deterministic polynomial time problems*) dan *NP-Hard*. Permasalahan *NP* adalah permasalahan komputasi yang masih belum memiliki algoritma yang dirancang untuk berjalan secara polinomial, tetapi dapat melakukan pengecekkan kebenaran solusi secara polinomial. Permasalahan *NP-Hard* adalah permasalahan komputasi yang masih belum memiliki algoritma yang dirancang untuk berjalan secara polinomial, dan untuk melakukan pengecekkan kebenaran solusi juga belum dapat dilakukan secara polinomial. Permasalahan *NP-Complete* adalah permasalahan yang belum dapat diselesaikan secara polinomial, karena untuk menyelesaikannya perlu dicari algoritma yang dapat berjalan pada waktu polinomial di permasalahan yang masih hanya dapat diselesaikan pada waktu eksponensial, atau proporsional dengan ukuran permasalahan (Marques-Silva, 1995).

Contoh permasalahan NP adalah sudoku, dimana bila diberikan hasil akhir maka dapat dilakukan pengecekkan kebenaran dari hasil tersebut dengan cepat. Contoh permasalahan NP-Hard adalah mencari jalan terbaik dari permainan catur, dimana untuk melakukan pengecekkan kebenaran membutuhkan waktu dan usaha yang sangat besar. Permasalahan NP-Complete adalah permasalahan sesulit NP-Hard dan memiliki algoritma yang dirancang untuk berjalan secara polinomial, tetapi masih memiliki beberapa bagian yang masih berjalan secara eksponensial, atau proporsional dengan ukuran masalah. Bila suatu permasalahan *NP-Complete* dapat diselesaikan dalam waktu polinomial maka, secara teoritis, semua jenis dari permasalahan *NP-Complete* tersebut seharusnya dapat diselesaikan dalam waktu polinomial (Jozsa, 1999).

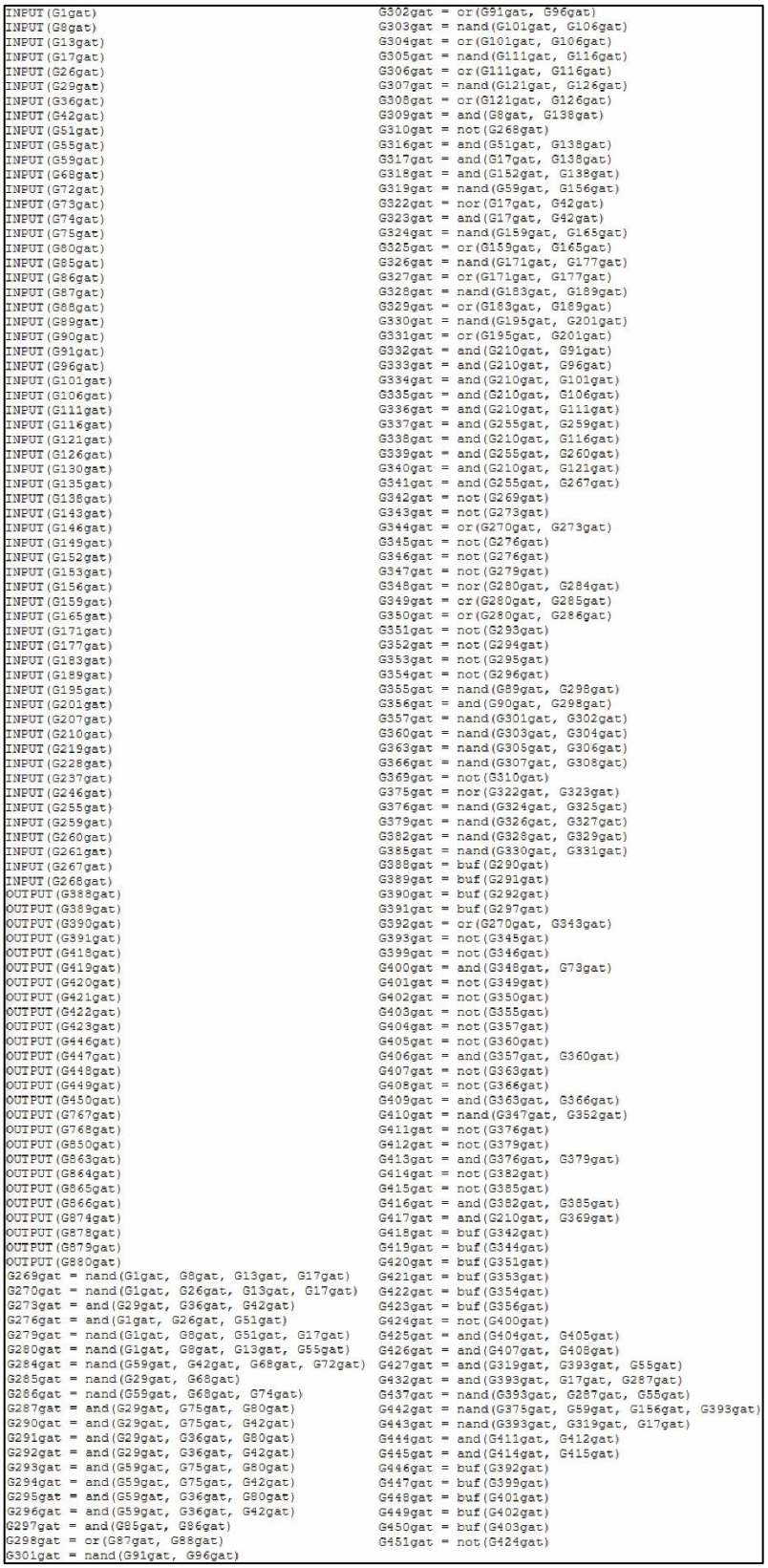
Satisfiability Problem atau yang biasa disebut dengan *SAT* adalaah permasalahan yang menghasilkan keluaran tertentu sesuai dengan suatu *constraint* tertentu. *SAT* adalah salah satu permasalahan yang disebut juga sebagai permasalahan *NP-Complete*. *SAT* adalah salah satu jenis permasalahan dalam kategori *NP-Complete*. Contoh implementasi permasalahan *SAT* dalam kehidupan sehari-hari adalah *Automation Pattern Generation*, *Timing Analysis*, dan juga *Circuit Satisfiability Problem* (e Silva dkk., 1999).

Sudah diterima secara umum, bahwa penyelesaian yang didapatkan untuk menyelesaikan permasalahan *NP-Complete* sangatlah sulit secara algoritmik dan sampai sekarang masih belum ditemukan algoritma yang dapat menyelesaikannya secara polinomial pada komputasi klasik (Marques-Silva, 1995). Walaupun belum efisien, terdapat beberapa jenis solusi yang biasa digunakan. Solusi-solusi yang didapatkan untuk penyelesaian permasalahan ini tetap berjalan dalam *worst-case*

pada waktu eksponensial dari masukan. Terdapat dua pendekatan untuk mendapatkan solusi-solusi untuk permasalahan *SAT* (e Silva dkk., 1999), yaitu dengan mencari di seluruh solusi yang mungkin dengan *exhaustive search* (Calabro dkk., 2009), atau dengan menggunakan algoritma pencarian tertentu seperti algoritma *backtracking* (Marques-Silva dan Sakallah, 2000), *path sensitization*, *decision tree,* dan *branch-and-prune* (Junttila dan Niemela, 2000), (Puget dan Van Hentenryck, 1998).

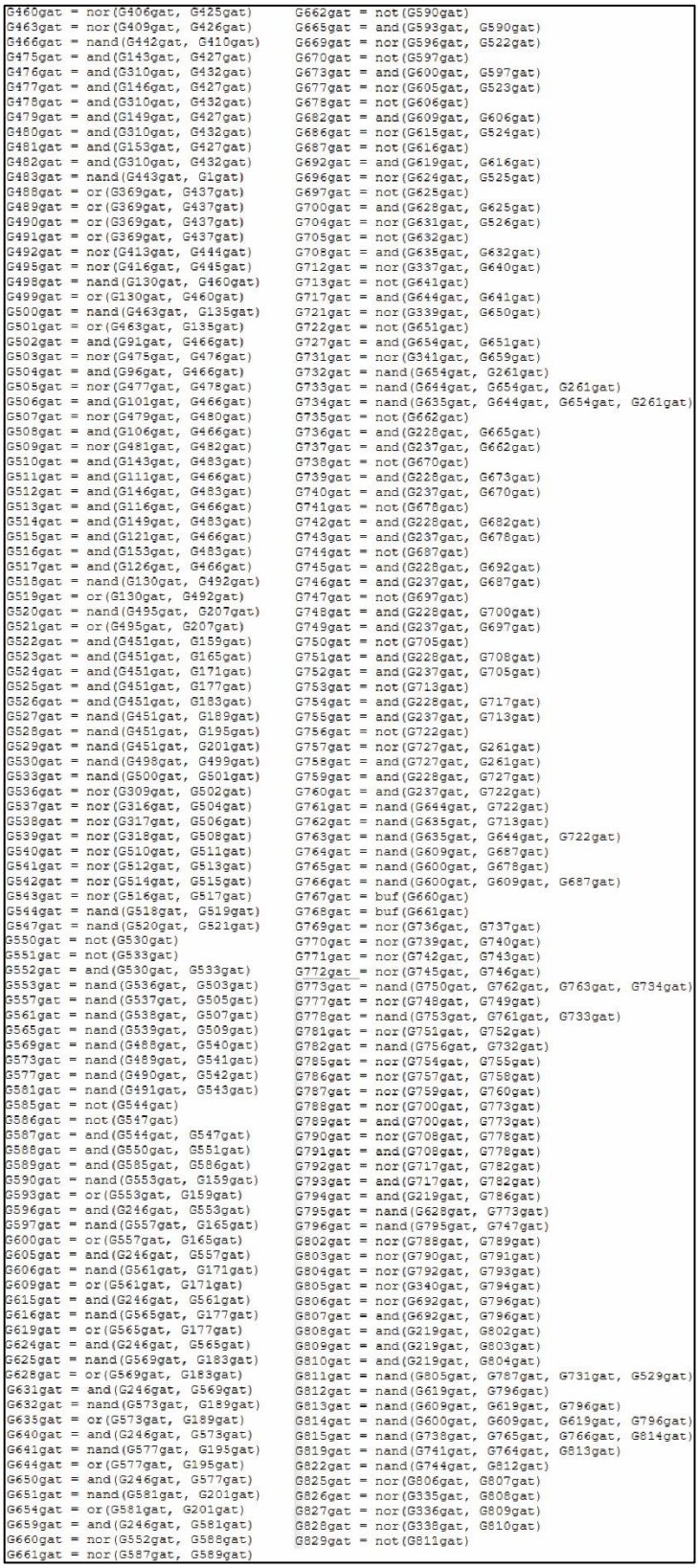
Circuit Satisfiability Problem (CSP) adalah salah satu jenis permasalahan di dalam *Satisfiability* Problem. *CSP* menggunakan sirkuit yang berisikan *logic gate* untuk menghasilkan keluaran tertentu yang diinginkan. *Keluaran* yang dicari di *CSP* ini adalah keluaran yang bernilai 1 di hasil akhir sirkuitnya. Di dalam *CSP* terdapat *n* jumlah masukan yang dimasukkan ke serangkaian sirkuit yang berisi *logic gate*, yang kemudian menghasilkan satu nilai keluaran. *Circuit Satisfiability Problem* adalah salah satu tantangan di ranah komputasi, dan dikarenakan masuk ke dalam kategori *NP-Complete,* maka belum ditemukan algoritma yang dapat berjalan dengan efektif di waktu yang polinomial di komputasi klasik (Quinn, 2003).

Sirkuit yang digunakan dalam penelitian ini adalah sirkuit C880 dari format ISCAS'85. Format ISCAS'85 memiliki sepuluh jaringan kombinatorial yang diberikan pada *International Symposium on Circuits And Systems* pada tahun 1985 (Bryan, 1985). Digunakan satu sirkuit sebagai acuan yaitu sirkuit C880 yang adalah salah satu sirkuit *benchmark* pertama. Gambar 2.1 adalah bentuk bahasa fortran dari sirkuit C880 dari jaringan kombinatorial ISCAS'85.



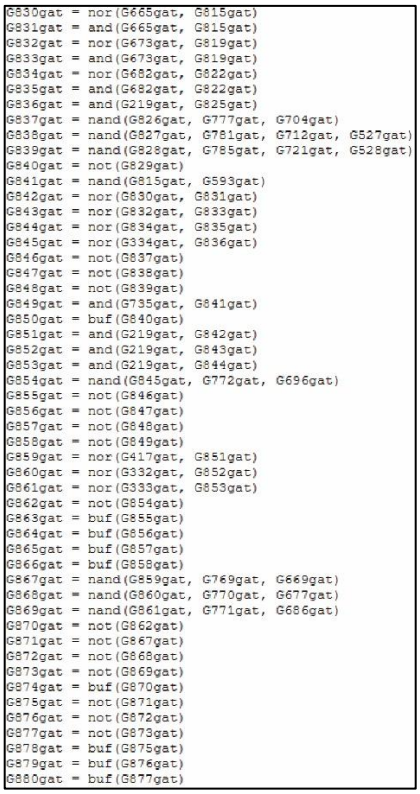
```
INPUT(G1gat)
INPUT(G8gat)
INPUT(G13gat)
INPUT(G17gat)
INPUT(G26gat)
INPUT(G29gat)
INPUT(G36gat)
INPUT(G42gat)
INPUT(G51gat)
INPUT(G55gat)
INPUT(G59gat)
INPUT(G68gat)
INPUT(G72gat)
INPUT(G73gat)
INPUT(G74gat)
INPUT(G75gat)
INPUT(G80gat)
INPUT(G85gat)
INPUT(G86gat)
INPUT(G87gat)
INPUT(G88gat)
INPUT(G89gat)
INPUT(G90gat)
INPUT(G91gat)
INPUT(G96gat)
INPUT(G101gat)
INPUT(G106gat)
INPUT(G111gat)
INPUT(G116gat)
INPUT(G121gat)
INPUT(G126gat)
INPUT(G130gat)
INPUT(G135gat)
INPUT(G138gat)
INPUT(G143gat)
INPUT(G146gat)
INPUT(G149gat)
INPUT(G152gat)
INPUT(G153gat)
INPUT(G156gat)
INPUT(G159gat)
INPUT(G165gat)
INPUT(G171gat)
INPUT(G177gat)
INPUT(G183gat)
INPUT(G189gat)
INPUT(G195gat)
INPUT(G201gat)
INPUT(G207gat)
INPUT(G210gat)
INPUT(G219gat)
INPUT(G228gat)
INPUT(G237gat)
INPUT(G246gat)
INPUT(G255gat)
INPUT(G259gat)
INPUT(G260gat)
INPUT(G261gat)
INPUT(G267gat)
INPUT(G268gat)
OUTPUT(G388gat)
OUTPUT(G389gat)
OUTPUT(G390gat)
OUTPUT(G391gat)
OUTPUT(G418gat)
OUTPUT(G419gat)
OUTPUT(G420gat)
OUTPUT(G421gat)
OUTPUT(G422gat)
OUTPUT(G423gat)
OUTPUT(G446gat)
OUTPUT(G447gat)
OUTPUT(G448gat)
OUTPUT(G449gat)
OUTPUT(G450gat)
OUTPUT(G767gat)
OUTPUT(G768gat)
OUTPUT(G850gat)
OUTPUT(G863gat)
OUTPUT(G864gat)
OUTPUT(G865gat)
OUTPUT(G866gat)
OUTPUT(G874gat)
OUTPUT(G878gat)
OUTPUT(G879gat)
OUTPUT(G880gat)
G269gat = nand(G1gat, G8gat, G13gat, G17gat)
G270gat = nand(G1gat, G26gat, G13gat, G17gat)
G273gat = and(G29gat, G36gat, G42gat)
G276gat = and(G1gat, G26gat, G51gat)
G279gat = nand(G1gat, G8gat, G51gat, G17gat)
G280gat = nand(G1gat, G8gat, G13gat, G55gat)
G284gat = nand(G59gat, G42gat, G68gat, G72gat)
G285gat = nand(G29gat, G68gat)
G286gat = nand(G59gat, G68gat, G74gat)
G287gat = and(G29gat, G75gat, G80gat)
G290gat = and(G29gat, G75gat, G42gat)
G291gat = and(G29gat, G36gat, G80gat)
G292gat = and(G29gat, G36gat, G42gat)
G293gat = and(G59gat, G75gat, G80gat)
G294gat = and(G59gat, G75gat, G42gat)
G295gat = and(G59gat, G36gat, G80gat)
G296gat = and(G59gat, G36gat, G42gat)
G297gat = and(G85gat, G86gat)
G298gat = or(G87gat, G88gat)
G301gat = nand(G91gat, G96gat)
G302gat = or(G91gat, G96gat)
G303gat = nand(G101gat, G106gat)
G304gat = or(G101gat, G106gat)
G305gat = nand(G111gat, G116gat)
G306gat = or(G111gat, G116gat)
G307gat = nand(G121gat, G126gat)
G308gat = or(G121gat, G126gat)
G309gat = and(G8gat, G138gat)
G310gat = not(G268gat)
G316gat = and(G51gat, G138gat)
G317gat = and(G17gat, G138gat)
G318gat = and(G152gat, G138gat)
G319gat = nand(G59gat, G156gat)
G322gat = nor(G17gat, G42gat)
G323gat = and(G17gat, G42gat)
G324gat = nand(G159gat, G165gat)
G325gat = or(G159gat, G165gat)
G326gat = nand(G171gat, G177gat)
G327gat = or(G171gat, G177gat)
G328gat = nand(G183gat, G189gat)
G329gat = or(G183gat, G189gat)
G330gat = nand(G195gat, G201gat)
G331gat = or(G195gat, G201gat)
G332gat = and(G210gat, G91gat)
G333gat = and(G210gat, G96gat)
G334gat = and(G210gat, G101gat)
G335gat = and(G210gat, G106gat)
G336gat = and(G210gat, G111gat)
G337gat = and(G255gat, G259gat)
G338gat = and(G210gat, G116gat)
G339gat = and(G255gat, G260gat)
G340gat = and(G210gat, G121gat)
G341gat = and(G255gat, G267gat)
G342gat = not(G269gat)
G343gat = not(G273gat)
G344gat = or(G270gat, G273gat)
G345gat = not(G276gat)
G346gat = not(G276gat)
G347gat = not(G279gat)
G348gat = nor(G280gat, G284gat)
G349gat = or(G280gat, G285gat)
G350gat = or(G280gat, G286gat)
G351gat = not(G295gat)
G352gat = not(G294gat)
G353gat = not(G295gat)
G354gat = not(G296gat)
G355gat = nand(G89gat, G298gat)
G356gat = and(G90gat, G298gat)
G357gat = nand(G301gat, G302gat)
G360gat = nand(G303gat, G304gat)
G363gat = nand(G305gat, G306gat)
G366gat = nand(G307gat, G308gat)
G369gat = not(G310gat)
G375gat = nor(G322gat, G323gat)
G376gat = nand(G324gat, G325gat)
G379gat = nand(G326gat, G327gat)
G382gat = nand(G328gat, G329gat)
G385gat = nand(G330gat, G331gat)
G388gat = buf(G290gat)
G389gat = buf(G291gat)
G390gat = buf(G292gat)
G391gat = buf(G297gat)
G392gat = or(G270gat, G343gat)
G393gat = not(G345gat)
G399gat = not(G346gat)
G400gat = and(G348gat, G73gat)
G401gat = not(G349gat)
G402gat = not(G350gat)
G403gat = not(G355gat)
G404gat = not(G357gat)
G405gat = not(G360gat)
G406gat = and(G357gat, G360gat)
G407gat = not(G363gat)
G408gat = not(G366gat)
G409gat = and(G363gat, G366gat)
G410gat = nand(G347gat, G352gat)
G411gat = not(G376gat)
G412gat = not(G379gat)
G413gat = and(G376gat, G379gat)
G414gat = not(G382gat)
G415gat = not(G385gat)
G416gat = and(G382gat, G385gat)
G417gat = and(G210gat, G369gat)
G418gat = buf(G342gat)
G419gat = buf(G344gat)
G420gat = buf(G351gat)
G421gat = buf(G353gat)
G422gat = buf(G354gat)
G423gat = buf(G356gat)
G424gat = not(G400gat)
G425gat = and(G404gat, G405gat)
G426gat = and(G407gat, G408gat)
G427gat = and(G319gat, G393gat, G55gat)
G432gat = and(G393gat, G17gat, G287gat)
G437gat = nand(G393gat, G287gat, G55gat)
G442gat = nand(G375gat, G59gat, G156gat, G393gat)
G443gat = nand(G393gat, G319gat, G17gat)
G444gat = and(G411gat, G412gat)
G445gat = and(G414gat, G415gat)
G446gat = buf(G392gat)
G447gat = buf(G399gat)
G448gat = buf(G401gat)
G449gat = buf(G402gat)
G450gat = buf(G403gat)
G451gat = not(G424gat)
```

Gambar 2.1. Sirkuit C880 Dalam Bahasa Fortran (Filebox, 2015)



```
G460gat = nor(G406gat, G425gat)
G463gat = nor(G409gat, G426gat)
G466gat = nand(G442gat, G410gat)
G475gat = and(G143gat, G427gat)
G476gat = and(G310gat, G432gat)
G477gat = and(G146gat, G427gat)
G478gat = and(G310gat, G432gat)
G479gat = and(G149gat, G427gat)
G480gat = and(G310gat, G432gat)
G481gat = and(G153gat, G427gat)
G482gat = and(G310gat, G432gat)
G483gat = nand(G443gat, G1gat)
G488gat = or(G369gat, G437gat)
G489gat = or(G369gat, G437gat)
G490gat = or(G369gat, G437gat)
G491gat = or(G369gat, G437gat)
G492gat = nor(G413gat, G444gat)
G495gat = nor(G416gat, G445gat)
G498gat = nand(G130gat, G460gat)
G499gat = or(G130gat, G460gat)
G500gat = nand(G463gat, G135gat)
G501gat = or(G463gat, G135gat)
G502gat = and(G91gat, G466gat)
G503gat = nor(G475gat, G476gat)
G504gat = and(G96gat, G466gat)
G505gat = nor(G477gat, G478gat)
G506gat = and(G101gat, G466gat)
G507gat = nor(G479gat, G480gat)
G508gat = and(G106gat, G466gat)
G509gat = nor(G481gat, G482gat)
G510gat = and(G143gat, G483gat)
G511gat = and(G111gat, G466gat)
G512gat = and(G146gat, G483gat)
G513gat = and(G116gat, G466gat)
G514gat = and(G149gat, G483gat)
G515gat = and(G121gat, G466gat)
G516gat = and(G153gat, G483gat)
G517gat = and(G126gat, G466gat)
G518gat = nand(G130gat, G492gat)
G519gat = or(G130gat, G492gat)
G520gat = nand(G495gat, G207gat)
G521gat = or(G495gat, G207gat)
G522gat = and(G451gat, G159gat)
G523gat = and(G451gat, G165gat)
G524gat = and(G451gat, G171gat)
G525gat = and(G451gat, G177gat)
G526gat = and(G451gat, G183gat)
G527gat = nand(G451gat, G189gat)
G528gat = nand(G451gat, G195gat)
G529gat = nand(G451gat, G201gat)
G530gat = nand(G498gat, G499gat)
G533gat = nand(G500gat, G501gat)
G536gat = nor(G309gat, G502gat)
G537gat = nor(G316gat, G504gat)
G538gat = nor(G317gat, G506gat)
G539gat = nor(G318gat, G508gat)
G540gat = nor(G510gat, G511gat)
G541gat = nor(G512gat, G513gat)
G542gat = nor(G514gat, G515gat)
G543gat = nor(G516gat, G517gat)
G544gat = nand(G518gat, G519gat)
G547gat = nand(G520gat, G521gat)
G550gat = not(G530gat)
G551gat = not(G533gat)
G552gat = and(G530gat, G533gat)
G553gat = nand(G536gat, G503gat)
G557gat = nand(G537gat, G505gat)
G561gat = nand(G538gat, G507gat)
G565gat = nand(G539gat, G509gat)
G569gat = nand(G488gat, G540gat)
G573gat = nand(G489gat, G541gat)
G577gat = nand(G490gat, G542gat)
G581gat = nand(G491gat, G543gat)
G585gat = not(G544gat)
G586gat = not(G547gat)
G587gat = and(G544gat, G547gat)
G588gat = and(G550gat, G551gat)
G589gat = and(G585gat, G586gat)
G590gat = nand(G553gat, G159gat)
G593gat = or(G553gat, G159gat)
G596gat = and(G246gat, G553gat)
G597gat = nand(G557gat, G165gat)
G600gat = or(G557gat, G165gat)
G605gat = and(G246gat, G557gat)
G606gat = nand(G561gat, G171gat)
G609gat = or(G561gat, G171gat)
G615gat = and(G246gat, G561gat)
G616gat = nand(G565gat, G177gat)
G619gat = or(G565gat, G177gat)
G624gat = and(G246gat, G565gat)
G625gat = nand(G569gat, G183gat)
G628gat = or(G569gat, G183gat)
G631gat = and(G246gat, G569gat)
G632gat = nand(G573gat, G189gat)
G635gat = or(G573gat, G189gat)
G640gat = and(G246gat, G573gat)
G641gat = nand(G577gat, G195gat)
G644gat = or(G577gat, G195gat)
G650gat = and(G246gat, G577gat)
G651gat = nand(G581gat, G201gat)
G654gat = or(G581gat, G201gat)
G659gat = and(G246gat, G581gat)
G660gat = nor(G552gat, G588gat)
G661gat = nor(G587gat, G589gat)
G662gat = not(G590gat)
G665gat = and(G593gat, G590gat)
G669gat = nor(G596gat, G522gat)
G670gat = not(G597gat)
G673gat = and(G600gat, G597gat)
G677gat = nor(G605gat, G523gat)
G678gat = not(G606gat)
G682gat = and(G609gat, G606gat)
G686gat = nor(G615gat, G524gat)
G687gat = not(G616gat)
G692gat = and(G619gat, G616gat)
G696gat = nor(G624gat, G525gat)
G697gat = not(G625gat)
G700gat = and(G628gat, G625gat)
G704gat = nor(G631gat, G526gat)
G705gat = not(G632gat)
G708gat = and(G635gat, G632gat)
G712gat = nor(G337gat, G640gat)
G713gat = not(G641gat)
G717gat = and(G644gat, G641gat)
G721gat = nor(G339gat, G650gat)
G722gat = not(G651gat)
G727gat = and(G654gat, G651gat)
G731gat = nor(G341gat, G659gat)
G732gat = nand(G654gat, G261gat)
G733gat = nand(G644gat, G654gat, G261gat)
G734gat = nand(G635gat, G644gat, G654gat, G261gat)
G735gat = not(G662gat)
G736gat = and(G228gat, G665gat)
G737gat = and(G237gat, G662gat)
G738gat = not(G670gat)
G739gat = and(G228gat, G673gat)
G740gat = and(G237gat, G670gat)
G741gat = not(G678gat)
G742gat = and(G228gat, G682gat)
G743gat = and(G237gat, G678gat)
G744gat = not(G687gat)
G745gat = and(G228gat, G692gat)
G746gat = and(G237gat, G687gat)
G747gat = not(G697gat)
G748gat = and(G228gat, G700gat)
G749gat = and(G237gat, G697gat)
G750gat = not(G705gat)
G751gat = and(G228gat, G708gat)
G752gat = and(G237gat, G705gat)
G753gat = not(G713gat)
G754gat = and(G228gat, G717gat)
G755gat = and(G237gat, G713gat)
G756gat = not(G722gat)
G757gat = nor(G727gat, G261gat)
G758gat = and(G727gat, G261gat)
G759gat = and(G228gat, G727gat)
G760gat = and(G237gat, G722gat)
G761gat = nand(G644gat, G722gat)
G762gat = nand(G635gat, G713gat)
G763gat = nand(G635gat, G644gat, G722gat)
G764gat = nand(G609gat, G687gat)
G765gat = nand(G600gat, G678gat)
G766gat = nand(G600gat, G609gat, G687gat)
G767gat = buf(G660gat)
G768gat = buf(G661gat)
G769gat = nor(G736gat, G737gat)
G770gat = nor(G739gat, G740gat)
G771gat = nor(G742gat, G743gat)
G772gat = nor(G745gat, G746gat)
G773gat = nand(G750gat, G762gat, G763gat, G734gat)
G777gat = nor(G748gat, G749gat)
G778gat = nand(G753gat, G761gat, G733gat)
G781gat = nor(G751gat, G752gat)
G782gat = nand(G756gat, G732gat)
G785gat = nor(G754gat, G755gat)
G786gat = nor(G757gat, G758gat)
G787gat = nor(G759gat, G760gat)
G788gat = nor(G700gat, G773gat)
G789gat = and(G700gat, G773gat)
G790gat = nor(G708gat, G778gat)
G791gat = and(G708gat, G778gat)
G792gat = nor(G717gat, G782gat)
G793gat = and(G717gat, G782gat)
G794gat = and(G219gat, G786gat)
G795gat = nand(G628gat, G773gat)
G796gat = nand(G795gat, G747gat)
G802gat = nor(G788gat, G789gat)
G803gat = nor(G790gat, G791gat)
G804gat = nor(G792gat, G793gat)
G805gat = nor(G340gat, G794gat)
G806gat = nor(G692gat, G796gat)
G807gat = and(G692gat, G796gat)
G808gat = and(G219gat, G802gat)
G809gat = and(G219gat, G803gat)
G810gat = and(G219gat, G804gat)
G811gat = nand(G805gat, G787gat, G731gat, G529gat)
G812gat = nand(G619gat, G796gat)
G813gat = nand(G609gat, G619gat, G796gat)
G814gat = nand(G600gat, G609gat, G619gat, G796gat)
G815gat = nand(G738gat, G765gat, G766gat, G814gat)
G819gat = nand(G741gat, G764gat, G813gat)
G822gat = nand(G744gat, G812gat)
G825gat = nor(G806gat, G807gat)
G826gat = nor(G335gat, G808gat)
G827gat = nor(G336gat, G809gat)
G828gat = nor(G338gat, G810gat)
G829gat = not(G811gat)
```

Gambar 2.1. Sirkuit C880 Dalam Bahasa Fortran (Lanjutan) (Filebox, 2015)

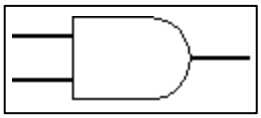


Gambar 2.1. Sirkuit C880 Dalam Bahasa Fortran (Lanjutan) (Filebox, 2015)

2.2 Gerbang Logika

Gerbang logika adalah alat untuk mengimplementasikan fungsi boolean sebagai blok dasar untuk membentuk sistem digital. Gerbang logika melakukan operasi logika tertentu dengan satu atau lebih masukan dan menghasilkan satu keluaran. Masukan dan keluaran berupa bilangan biner dengan nilai nol atau satu. Terdapat empat jenis gerbang logika dasar, yaitu gerbang *AND*, *OR*, *XOR*, dan *NOT* (Rosen, 1999).

Gerbang *AND* adalah gerbang logika yang menerima lebih dari satu masukan, menerapkan logika *AND*, dan mengeluarkan satu keluaran. Gerbang *AND* mengambil seluruh nilai masukan dan memberikan nilai satu pada keluaran hanya bila seluruh nilai masukan adalah satu. Gambar 2.2 adalah gambar gerbang *AND* dan Tabel 2.1 adalah tabel kebenaran gerbang *AND* (Rosen, 1999).

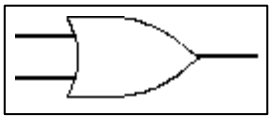


Gambar 2.2. Gerbang *AND*

Tabel 2.1. Tabel kebenaran gerbang AND

Masukan 1	Masukan 2	Keluaran
0	0	0
0	1	0
1	0	0
1	1	1

Gerbang *OR* adalah gerbang logika yang menerima lebih dari satu masukan, menerapkan logika *OR*, dan mengeluarkan satu keluaran. Gerbang *OR* mengambil seluruh nilai masukan dan memberikan nilai satu pada keluaran hanya bila salah satu dari nilai masukan adalah satu. Gambar 2.3 adalah gambar gerbang *OR* dan Tabel 2.2 adalah tabel kebenaran gerbang *OR* (Rosen, 1999).



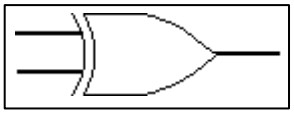
Gambar 2.3. Gerbang *OR*

Tabel 2.2. Tabel kebenaran gerbang *OR*

Masukan 1	Masukan 2	Keluaran
0	0	0
0	1	1
1	0	1
1	1	1

Gerbang *XOR* adalah gerbang logika yang menerima lebih dari satu masukan, menerapkan logika *XOR*, dan mengeluarkan satu keluaran. Gerbang *XOR* mengambil seluruh nilai masukan dan memberikan nilai satu pada keluaran hanya bila hanya terdapat satu nilai masukan yang bernilai satu. Gambar 2.4

adalah gambar gerbang *XOR* dan Tabel 2.3 adalah tabel kebenaran gerbang *XOR* (Rosen, 1999).

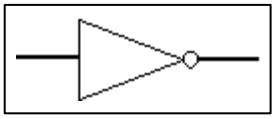


Gambar 2.4. Gerbang *XOR*

Tabel 2.3. Tabel kebenaran gerbang *XOR*

Masukan 1	Masukan 2	Keluaran
0	0	0
0	1	1
1	0	1
1	1	0

Gerbang *NOT* adalah gerbang logika yang menerima satu masukan, menerapkan logika *NOT*, dan mengeluarkan satu keluaran. Gerbang *NOT* mengambil nilai masukan dan memberikan nilai keluaran yang bukan nilai masukan tersebut. Gambar 2.5 adalah gambar gerbang *NOT* dan Tabel 2.4 adalah tabel kebenaran gerbang *NOT* (Rosen, 1999).



Gambar 2.5. Gerbang *NOT*

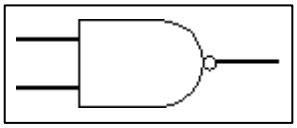
Tabel 2.4. Tabel kebenaran gerbang *NOT*

Masukan	Keluaran
0	1
1	0

Terdapat tiga gerbang logika yang menjadi turunan dari empat gerbang logika dasar tersebut. Tiga gerbang logika tersebut adalah gerbang *NAND*, *NOR*, dan *XNOR*. Ketiga gerbang ini adalah hasil gabung dari gerbang *AND*, *OR*, dan

XOR dengan gerbang *NOT*. Digunakan dua dari tiga gerbang logika tersebut untuk membangun sirkuit dalam permasalahan penelitian, yaitu gerbang *NAND*, dan *NOR* (Rosen, 1999).

Gerbang *NAND* adalah gerbang logika yang menerima lebih dari satu masukan, menerapkan logika *AND* dan kemudian logika *NOT*, dan mengeluarkan satu keluaran. Gerbang *NAND* mengambil seluruh nilai masukan dan memberikan nilai satu pada keluaran hanya bila tidak seluruh nilai masukan adalah satu. Gambar 2.6 adalah gambar gerbang *NAND* dan Tabel 2.5 adalah tabel kebenaran gerbang *NAND* (Rosen, 1999).

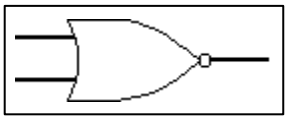


Gambar 2.6. Gerbang *NAND*

Tabel 2.5. Tabel kebenaran gerbang *NAND*

Masukan 1	Masukan 2	Keluaran
0	0	1
0	1	1
1	0	1
1	1	0

Gerbang *NOR* adalah gerbang logika yang menerima lebih dari satu masukan, menerapkan logika *OR* dan kemudian logika *NOT*, dan mengeluarkan satu keluaran. Gerbang *NOR* mengambil seluruh nilai masukan dan memberikan nilai satu pada keluaran hanya bila tidak ada nilai masukan yang adalah satu. Gambar 2.7 adalah gambar gerbang *NOR* dan Tabel 2.6 adalah tabel kebenaran gerbang *NOR* (Rosen, 1999).



Gambar 2.7. Gerbang *NOR*

Tabel 2.6. Tabel kebenaran gerbang *NOR*

Masukan 1	Masukan 2	Keluaran
0	0	1
0	1	0
1	0	0
1	1	0

2.3 Algoritma *Quantum Backtracking*

Komputasi kuantum adalah suatu komputasi yang dilakukan pada mesin komputer kuantum dengan menerapkan prinsip-prinsip di dalam mekanika kuantum. Dengan algoritma yang dijalankan dengan komputasi kuantum, maka terdapat banyak peningkatan efisiensi dibandingkan komputasi klasik untuk beberapa jenis permasalahan (Kwiat dkk., 2000). Di dalam komputasi kuantum, dilakukan kalkulasi pada bit kuantum (*qubit*), yang adalah unit bilangan terkecil di dalam komputasi kuantum, dengan menggunakan *reversible gate* untuk menghasilkan keluaran yang diinginkan. Tiap *qubit* memiliki kemungkinan dua nilai yaitu 0 dan 1, seperti halnya bilangan biner. Di saat *qubit* diukur (*measure*), maka *qubit* bernilai 0 atau 1 sesuai dengan state *qubit* tersebut saat *collapse*. Setiap nilai dari *qubit* memiliki suatu nilai koefisien yang merupakan suatu nilai probabilitas yang menjadi peluang *qubit* tersebut bernilai 0 ataupun 1 di saat dilakukan pengukuran (Nielsen dan Chuang, 2002).

Untuk menghasilkan hasil pengukuran yang diinginkan dengan *n qubit*, maka nilai probabilitas dari setiap nilai *qubit* tersebut harus disesuaikan sedemikian rupa agar dapat memperbesar nilai probabilitas nilai tersebut untuk

terpilih untuk setiap *qubit* di saat pengukuran. Cara untuk mengubah nilai probabilitas dari *n qubit* yang ada untuk mendapatkan hasil keluaran yang diinginkan inilah yang diterapkan di dalam algoritma dalam komputasi kuantum. Di saat suatu algoritma komputasi kuantum dijalankan, maka algoritma mengerjakan semua kemungkinan masukan secara bersamaan dan kemudian memberikan hasil yang diinginkan dengan memanfaatkan konsep komputasi kuantum dan mekanika kuantum (Nielsen dan Chuang, 2002).

Terdapat beberapa istilah yang digunakan di dalam ranah mekanika kuantum yang berkaitan erat dengan komputasi kuantum, yaitu *Quantum Superposition* dan *Quantum Entanglement*. Ada suatu keadaan di suatu waktu yang sama suatu *qubit* memiliki probabilitas untuk berada di *state* 0 dan 1 sehingga *qubit* tersebut berada di *state* 0 dan juga *state* 1 secara bersamaan. Keadaan ini dinamakan *Quantum Superposition*. Suatu *qubit* memiliki nilai 0 atau 1 setelah *qubit* tersebut telah diukur (Kwiat dkk., 2000), (Lavor dkk., 2003). Terdapat juga suatu fenomena di suatu sistem kuantum yang tiap *state* yang ada di dalamnya, walaupun memiliki nilai probabilitas yang terpisah dan berbeda-beda, bukan merupakan partikel yang tidak berpengaruh satu sama lain. Tiap *state* di dalam sistem kuantum berpengaruh ke dan dipengaruhi oleh *state* lain di dalam sistem tersebut. Hal ini dikarenakan nilai total kuadrat dari probabilitas tiap *state* haruslah bernilai 1. Fenomena ini disebut dengan *Quantum Entanglement*. Kedua keadaan di dalam mekanika kuantum ini sangat erat berpengaruh pada pelaksanaan algoritma kuantum (Rossi dkk., 2014).

Algoritma *quantum backtracking* adalah suatu algoritma dalam komputasi kuantum yang digunakan sebagai salah satu algoritma pencarian yang

dimanfaatkan pada permasalahan berbasis *tree*. Permasalahan *tree* bisa memiliki suatu permasalahan *SAT* contohnya *constraint satisfaction problem P* dengan *n* jumlah variabel, *d* jumlah value, dan *m* jumlah *constraint*. Bila suatu *state* dimana setiap variabel yang diisi dengan suatu *value* tertentu dapat memenuhi seluruh *constraint* maka *state* itu disebut sebagai *solution*. Bila semua kemungkinan *state* variabel di dalam permasalahan dapat menjadi jawaban untuk permasalahan *P* maka permasalahan disebut dengan *complete*, dan bila tidak semua kemungkinan *state* variabel di dalam permasalahan dapat menjadi jawaban untuk permasalahan *P* maka permasalahan disebut dengan *partial*. Bila suatu permasalahan *P* adalah *partial* dan terdapat *state* variabel yang menjadi solusi, maka permasalahan tersebut bernilai *valid* (Martiel dan Remaud, 2019).

Algoritma *backtracking* menempuh seluruh *path* yang mungkin ada di dalam *tree* sehingga dapat menyelesaikan permasalahan *SAT* tersebut. Algoritma *quantum backtracking* melakukan suatu *walk* yang dilakukan dengan mengaplikasikan suatu *operator* untuk dapat melintasi seluruh *path* yang ada di *tree* permasalahan. Hasil dari *walk* yang menjadi penyelesaian permasalahan berupa suatu *span* dari *eigenvector*. *Operator* yang digunakan akan mengubah nilai probabilitas dari *state* yang menjadi penyelesaian. Diaplikasikan juga *phase estimation* dari *operator* dari *walk* tersebut untuk meninggikan probabilitas dari solusi *operator* tadi. Proses meninggikan probabilitas ini disebut dengan *Amplitude Amplification*. Proses ini dilakukan agar keluaran yang diinginkan saat pengukuran memiliki probabilitas yang sangat tinggi untuk *collapse* di saat pengukuran (Martiel dan Remaud, 2019).

Tahapan-tahapan untuk mengimplementasikan algoritma *backtracking* dilakukan dengan melakukan pengulangan pengaplikasian operator R_A dan R_B dan *phase estimation*. Penggambaran umum algoritma *backtracking* yang digunakan ada di Algoritma 1 (Martiel dan Remaud, 2019).

Algoritma 1 Algoritma Backtracking (algoritma 1 Martiel dan Remaud (2019))

Require: An assignment of values x, access to $h : \mathcal{D} \rightarrow [\![1, n]\!]$ and to $P : \mathcal{D} \rightarrow \{true, indeterminate, false\}$.

1: **If** $P(x)$ is true **then** output x and **return**
2: **If** $P(x)$ is false **then return**
3: $j \leftarrow h(x)$
4: **For** $w \in [\![1, d]\!]$ **do**
5: $y \leftarrow x$ with the j -th entry replaced with w.
6: **return Algoritma 1**(y)

Berdasarkan algoritma 1, terdapat beberapa variabel yang digunakan untuk menjelaskan algoritma tersebut. Berikut adalah penjelasan mengenai variabel-variabel tersebut.

1. x = variabel pada permasalahan
2. h = hasil nilai variabel
3. $\mathcal{D}$ = nilai untuk setiap variabel pada permasalahan
4. n = jumlah variabel
5. P = hasil permasalahan
6. j = *queue* dari *tree* berdasarkan *traverse* dari *children*
7. w = variabel penyimpan sementara
8. d = nilai dari tiap variabel
9. y = nilai sementara hasil *traverse* dari *children*

Dari gambaran umum tersebut kemudian dibuat gambaran umum algoritma *quantum backtracking* yang ada di Algoritma 2 dan Algoritma 3. Algoritma *quantum backtracking* untuk mendapatkan solusi pada permasalahan

akan melalui seluruh kemungkinan dari *tree* secara *rekursif*. Setiap cabang dicek terhadap permasalahan. Bila menjadi solusi maka cabang tersebut dikeluarkan, dan bila tidak menjadi solusi maka cabang diubah dengan mengganti isi dari cabang tersebut (Martiel dan Remaud, 2019).

Algoritma 2 Mendeteksi Solusi (algoritma 2 Martiel dan Remaud (2019))

Require: Operators R_A, R_B, a failure probability δ, upper bounds on the depth n and the number of vertices T. Let $\beta, \gamma > 0$ be universal constants to be determined.

1: **Repeat** $K = \lceil \gamma \, log \, 1/\delta \rceil$ times:
2: Apply phase estimation to the operator $R_B \, R_A$ with precision $\beta \, / \sqrt{Tn}$
3: **If** the eigenvalue is 1 **then** accept
4: **if** number of acceptances $\geq 3K/8$ **then return** "Solution exists"
5: **else return** "No solution"

Berdasarkan algoritma 2, terdapat beberapa variabel yang digunakan untuk menjelaskan algoritma tersebut. Berikut adalah penjelasan mengenai variabel-variabel tersebut.

1. R_A = operator untuk *traverse* ke *leaf* genap
2. R_B = operator untuk *traverse* ke *leaf* ganjil
3. δ = probabilitas kegagalan (sesuai nilai variabel)
4. T = jumlah verteks dari *tree*
5. β = konstanta presisi
6. γ = konstanta *phase estimation*
7. K = jumlah pengulangan *phase estimation*

Algoritma 3 Menemukan Solusi (algoritma 3 Martiel dan Remaud (2019))

1: Apply algoritma 2 to the entire tree.
2: **If** it outputs "No solution" **then return**
3: Apply algoritma 2 to each child of the root until one outputs "Solution exists"
4: **If** this child is a solution **then** output its label and **return**
5: **else** go back to step 3 with this child as the root

Algoritma 2 dan 3 berjalan di ranah komputasi kuantum. Algoritma 2 berfungsi untuk mengetahui apakah ada solusi di permasalahan, dan algoritma 3 berfungsi untuk mencari solusi bila permasalahan dapat diselesaikan. Algoritma 2 menjalankan *quantum phase estimation* ke hasil dari operator R_A dan R_B ke tree permasalahan. Algoritma 3 mencari keluaran dengan mencari ke seluruh cabang (Martiel dan Remaud, 2019).

Quantum phase estimation adalah suatu algoritma yang mencari *eigenvalue* dari suatu *eigenvector* berdasarkan *unitary operator* tertentu. Bila hasil *eigenvalue* adalah satu, *unitary operator* tersebut adalah matriks yang digunakan untuk mencari eigenvalue dari *eigenvector*. Pada algoritma 2, operator R_A dan R_B adalah *unitary operator* dan *eigenvector* adalah masukan permasalahan (Martiel dan Remaud, 2019).

Tree untuk melakukan *backtracking* dimulai dari *root* dengan penomoran 0. Saat dibuat tree untuk melakukan *backtracking*, dari satu *parent* terbentuk dua *child*. Pada dua *child* ini diaplikasikan *operator* R_A dan R_B. *Operator* R_A diaplikasikan pada *child* genap dan R_B diaplikasikan pada *child* ganjil. Operator R_A dan R_B memiliki rumus sebagai berikut (Martiel dan Remaud, 2019).

$$R_A = \bigoplus_{x \in A} D_X \text{ dan } R_B = |r\rangle\langle r| + \bigoplus_{x \in B} D_X \qquad ...(2.1)$$

R_A mencari hasil *diffusion operator* D_x pada seluruh *x* pada cabang genap dari *tree* (sisi kanan). R_B melakukan proyeksi dari root dan menambahkan hasil *diffusion operator* D_x pada seluruh *x* pada cabang ganjil dari *tree* (sisi kiri). Hasil Operator R_A dan R_B memiliki gambaran umum algoritma seperti pada Algorimta 4. Untuk gambaran implementasi tiap tahapan dari algoritma tersebut dapat dilihat di Gambar 2.8 (Martiel dan Remaud, 2019).

Algoritma 4 Implementasi operator R_A (algoritma 4 Martiel dan Remaud (2019))

Require: A basis state $|l\rangle|v_1\rangle \cdots |v_n\rangle \in C^{n+1} \bigotimes (C^{d+1})^{\otimes n}$ corresponding to a partial assignment $x_1 = v_1, \cdots, x_l = v_l$. Ancilla Registers : $\mathcal{H}_{anc}$, $\mathcal{H}_{children}$, storing a tuple (a, S), where a $\in \{*\} \cup [d], S \subseteq [d]$, initialized to $a = *$, S = $\emptyset$.

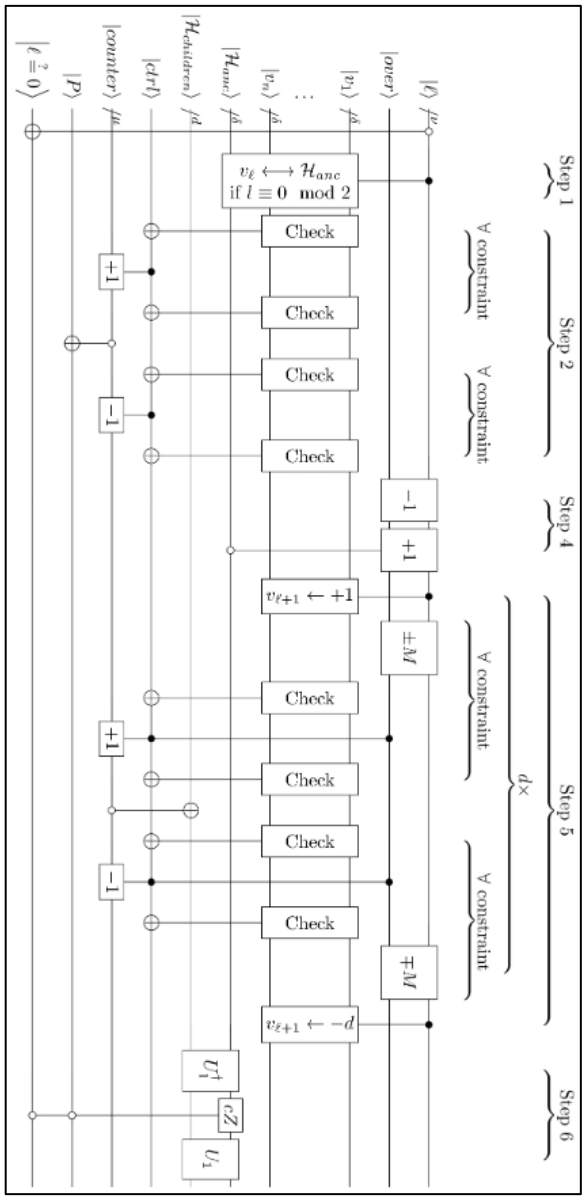
1: If *l* is odd, swap *a* with v_l.
2: Compute $P(x)$.
3: If $P(x)$ is true, go to step 8.
4: If $a \neq *$, subtract 1 from l.
5: For each w $\in [d]$, if $P(v_1, \cdots, v_l, w)$ is not false, set S = S $\cup \{w\}$.
6: If l = 0, i = n, else, i = 1. Perform $I - 2\,|\phi_{i,S}\rangle\langle\phi_{i,S}|$ on $\mathcal{H}_{anc}$.
7: Revert steps 5 and 4.
8: Revert steps 2 and 1.

Berdasarkan algoritma 4, terdapat beberapa variabel yang digunakan untuk menjelaskan algoritma tersebut. Berikut adalah penjelasan mengenai variabel-variabel tersebut.

1. l = vektor dari *state* dalam ranah *quantum*
2. v = state dari tiap nilai yang ada di variabel
3. C = *constraint* pada permasalahan
4. $\mathcal{H}_{anc}$ = *ancilla qubit* penyimpan hasil akhir
5. $\mathcal{H}_{children}$ = *ancilla qubit* penyimpan hasil *traverse* dari *children*
6. *a* = nilai *state* dalam desimal
7. *S* = *state* yang disimpan pada *ancilla qubit*
8. $I - 2\,|\phi_{i,S}\rangle\langle\phi_{i,S}|$ = *measurement* dilakukan pada *state* S yang ada di *ancilla qubit* $\mathcal{H}_{anc}$

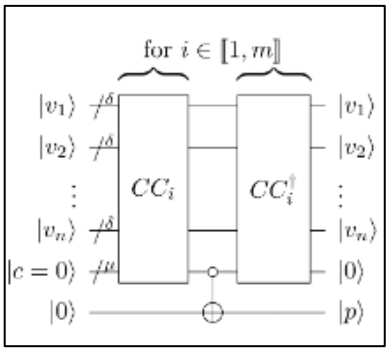
Pada Algoritma 4, *compute P(x)* adalah fungsi untuk mengetahui hasil keluaran dari masukan *x* pada permasalahan *P*. Bila hasil *P(x)* adalah benar berarti *x* adalah solusi *complete*, dan bila tidak akan terdapat dua kemungkinan yaitu bernilai salah dan *indeterminate* yang masih dapat berupa solusi. Pada tahap 5 dilakukan pengecekkan nilai salah dan *indeterminate*, bila bernilai *indeterminate*

maka *x* merupakan solusi. Variabel *S* adalah kumpulan *state vector* yang merupakan solusi dari *P(x)*. Pada tahap 6 diterapkan pengukuran ke sirkuit kuantum yang dibuat untuk menghasilkan keluaran akhir dari sirkuit kuantum yang dijalankan. Operator R_B melakukan hal yang mirip dengan R_A, hanya memiliki beberapa nilai yang berbeda. Pada tahap satu, *odd* diubah menjadi *even* karena R_A untuk cabang kanan (*odd*) dan R_B untuk cabang kiri (*even*). Pada tahap enam, "if l = 0" dihilangkan karena cabang ganjil tidal memiliki nilai *root* (0). Gambar 2.8 menerapkan Algoritma 4 dalam komputasi kuantum (Martiel dan Remaud, 2019).



Gambar 2.8. Implementasi Operator R_A (Martiel dan Remaud, 2019)

Tahap ke-2 dari algoritma pada Algoritma 4 melakukan komputasi kepada permasalahan *SAT*. Gambaran implementasi tahap ke-2 dapat dilihat di Gambar 2.9. Tahap ke-5 dari Algoritma 4 melakukan pengecekkan *constraint* yang ada pada permasalahan *SAT*. Untuk *CSP* tidak terdapat *constraint* tertentu pada nilai masukan, sehingga step ini dapat dikatakan selalu terpenuhi (Martiel dan Remaud, 2019).



Gambar 2.9. Implementasi Komputasi Permasalahan SAT

Gambar 2.9 menampilkan bagaimana permasalahan diselesaikan. Masukan *x* tiap cabang adalah nilai dari tiap masukan dari v_1 sampai dengan v_n. Blok pertama berisikan sirkuit untuk mengetahui apakah masukan *x* merupakan penyelesaian permasalahan *P(x)* dengan melihat setiap *constraint* (C_i) yang ada. Bila adalah penyelesaian, maka ada register sebagai penanda nilai *x*. Setelah menyimpan di *register*, masukan harus dikembalikan seperti semula sehingga diterapkan matriks *inverse* ($C_i^\dagger$) agar masukan tidak berubah (Martiel dan Remaud, 2019).

6.3.1. Gerbang Kuantum

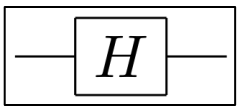
Gerbang kuantum adalah gerbang logika yang digunakan dalam komputasi kuantum untuk melakukan operasi kuantum pada *qubit*. Gerbang kuantum pada komputasi kuantum adalah seperti gerbang logika pada komputasi klasik. Gerbang kuantum bersifat *reversible*, dimana hasil komputasi yang dilakukan dapat dikembalikan ke kondisi awal sebelum komputasi. Komputasi klasik yang *irreversible* dapat diterapkan di komputasi kuantum dengan menggunakan kombinasi satu atau lebih gerbang kuantum. Terdapat empat jenis gerbang kuantum yang digunakan pada penelitian ini untuk membentuk gerbang logika pada komputasi klasik, yaitu Gerbang Hadamard, Gerbang Pauli-X, Gerbang

Contolled Not, dan Gerbang Toffoli. Fungsi dari tiap gerbang tersebut akan dipaparkan di empat paragraf setelah ini (Lavor dkk., 2003).

Gerbang Hadamard (*H*) adalah gerbang kuantum yang bekerja pada satu masukan *qubit*, dan memetakan *state* dari *qubit* tersebut menjadi *superposition*. Gambar 2.10 adalah representasi matriks dari Gerbang Hadamard, dan Gambar 2.11 adalah gambar simbol Gerbang Hadamard (Lavor dkk., 2003).

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Gambar 2.10. Representasi Matriks Gerbang Hadamard

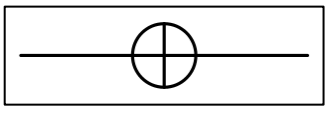


Gambar 2.11. Simbol Gerbang Hadamard

Gerbang Pauli-X (*X*) adalah gerbang kuantum yang bekerja pada satu masukan *qubit*, dan membalik probabilitas dari dua *state* di *qubit* tersebut. Gerbang ini bekerja seperti gerbang *NOT* pada komputasi klasik. Gambar 2.12 adalah representasi matriks dari Gerbang Pauli-X, dan Gambar 2.13 adalah gambar simbol Gerbang Pauli-X (Lavor dkk., 2003).

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Gambar 2.12. Representasi Matriks Gerbang Pauli-X

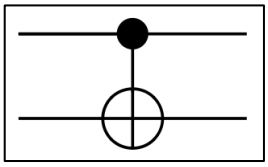


Gambar 2.13. Simbol Gerbang Pauli-X

Gerbang Controlled Not (*CNOT*) adalah gerbang kuantum yang bekerja pada dua masukan *qubit*, dan melakukan operasi *NOT* pada *qubit* kedua bila *state* pada *qubit* pertama bernilai 1. Gambar 2.14 adalah representasi matriks dari Gerbang Controlled Not, dan Gambar 2.15 adalah gambar simbol Gerbang Controlled Not (Lavor dkk., 2003).

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Gambar 2.14. Representasi Matriks Gerbang Controlled Not

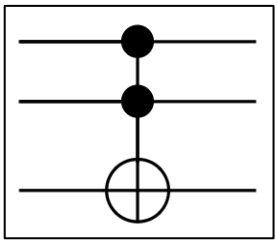


Gambar 2.15. Simbol Gerbang Controlled Not

Gerbang Toffoli (*CCNOT*) adalah gerbang kuantum yang bekerja pada tiga masukan *qubit*, dan melakukan operasi *NOT* pada *qubit* ketiga bila *state* pada *qubit* pertama dan kedua bernilai 1. Gambar 2.16 adalah representasi matriks dari Gerbang Toffoli, dan Gambar 2.17 adalah gambar simbol Gerbang Toffoli (Lavor dkk., 2003).

$$CCNOT = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Gambar 2.16. Representasi Matriks Gerbang Toffoli



Gambar 2.17. Simbol Gerbang Toffoli

2.4 Rigetti Computing

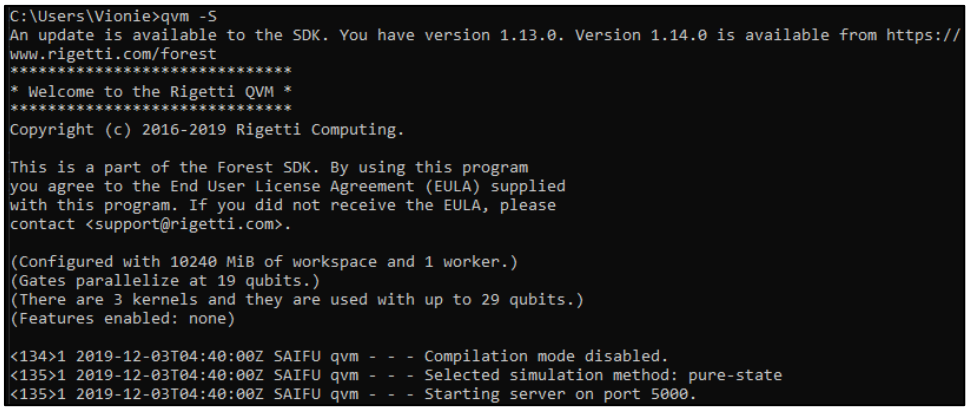
Rigetti Computing adalah *startup* teknologi yang didirikan pada tahun 2013 oleh Chad Rigetti. *Rigetti Computing* ingin membangun komputer terkuat di dunia untuk komputasi kuantum untuk membantu menyelesaikan permasalahan sulit umat manusia. Pada acara TechCrunch Disrupt SF 2018, *Rigetti Computing* mengumumkan peluncuran *platform* komputasi kuantum yang diimplementasikan secara *hybrid*. Platform ini menyediakan *API* untuk mengakses platform komputasi kuantum dengan nama Quantum Cloud Services (QCS) untuk menjalankan komputasi kuantum melewati servis *cloud* dan *platform development* Forest untuk dapat mengakses *quantum backend* yang dimiliki oleh Rigetti yang dapat dijalankan secara lokal (Rigetti Computing, 2019), (Lardinois, 2018).

Rigetti Computing memiliki *software development platform* bernama Forest. Forest menjalankan komputasi kuantum dengan menggunakan bahasa instruksi kuantum bernama quil yang dibentuk dengan *library* python pyQuil. Program Quil yang dibentuk di-*compile* dengan *compiler* dan dijalankan pada simulator kuantum. Simulasi pada *Rigetti Computing* dapat dijalankan dengan menggunakan Rigetti Quantum Virtual Machine (QVM), dan Rigetti Quil Compiler (quilc). Quantum Virtual Machine digunakan untuk menjalankan simulasi program quil. Quil Compiler digunakan untuk melakukan kompilasi dan optimisasi dari program Quil yang dijalankan. *Platform* Forest Rigetti berisikan

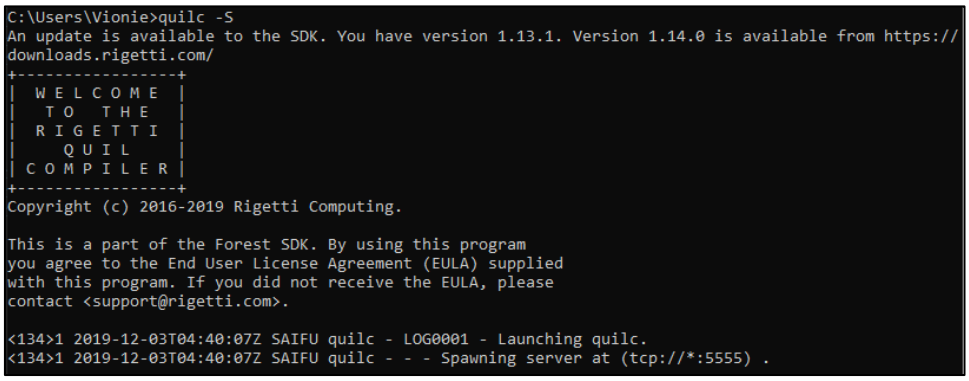
QVM dan quilc sehingga dapat menjalankan simulasi komputasi kuantum. *Library* pyQuil juga digunakan untuk menghasilkan dan menjalankan program Quil pada *platform* Forest Rigetti tersebut (PyQuil 2.14.0 Documentation, 2019).

Untuk memulai simulasi secara lokal, perlu dibuat server QVM dan quilc secara lokal pada dua terminal yang berjalan pada *background*. Program dibuat dengan menggunakan bahasa python dan menggunakan *library* pyQuil. Program dapat dijalankan pada QVM untuk mendapatkan simulasi komputasi kuantum. *Library* pyQuil memiliki 27 fungsi yang dapat digunakan untuk merepresentasikan gerbang kuantum, dan 9 fungsi yang dapat digunakan untuk menampilkan objek hasil QVM. Fungsi untuk merepresentasikan gerbang kuantum digunakan dengan memanggil pyQuil.gates dan fungsi untuk menampilkan objek hasil QVM digunakan dengan menggunakan *Wavefunction Simulator* (PyQuil 2.14.0 Documentation, 2019).

Untuk menggunakan Forest untuk membangun dan menjalankan simulasi komputasi kuantum, perlu di-*install library* pyQuil pada python versi 3. Saat melakukan instalasi Forest, maka QVM dan quilc juga sekaligus di-*install* sehingga dapat digunakan saja. Untuk memulai simulasi kuantum, perlu dijalankan server QVM dan quilc secara lokal. Kode untuk program quil dapat dibuat dengan python sehingga dapat langsung dijalankan pada QVM dan quilc. Bila kode program dijalankan maka program dikirimkan ke server quilc untuk di-*compile* dan dikirimkan ke QVM untuk dijalankan. Hasil dari QVM dikirim kembali dan memberikan hasil yang diinginkan (PyQuil 2.14.0 Documentation, 2019).



Gambar 2.18. Server QVM Dimulai



Gambar 2.19. Server quilc Dimulai

```
1  from pyquil import Program, get_qc
2  from pyquil.gates import *
```

```
1  # construct a Bell State program
2  p = Program(H(0), CNOT(0, 1))
```

```
1  # run the program on a QVM
2  qc = get_qc('9q-square-qvm')
3  result = qc.run_and_measure(p, trials=100)
4  print(result[0])
5  print(result[1])
```

Gambar 2.20. Contoh Kode Komputasi dengan *Rigetti Computing*

```
<134>1 2019-12-03T04:40:07Z SAIFU quilc - LOG0001 - Launching quilc.
<134>1 2019-12-03T04:40:07Z SAIFU quilc - - - Spawning server at (tcp://*:5555) .

<134>1 2019-12-03T04:40:58Z SAIFU quilc - - - Request efb42c37-5ffb-43ea-9383-3e9e5eba6b0c received f
or get_version_info
<134>1 2019-12-03T04:40:58Z SAIFU quilc - LOG0002 [rigetti@0000 methodName="get_version_info" request
ID="efb42c37-5ffb-43ea-9383-3e9e5eba6b0c" wallTime="0.133" error="false"] Requested get_version_info
completed
<134>1 2019-12-03T04:40:58Z SAIFU quilc - - - Request 7492b867-bf31-459e-8bb2-cf7cabf358f7 received f
or get_version_info
<134>1 2019-12-03T04:40:58Z SAIFU quilc - LOG0002 [rigetti@0000 methodName="get_version_info" request
ID="7492b867-bf31-459e-8bb2-cf7cabf358f7" wallTime="0.002" error="false"] Requested get_version_info
completed
<134>1 2019-12-03T04:40:58Z SAIFU quilc - - - Request 90166593-2ccc-46af-9b56-0519a6443188 received f
or quil_to_native_quil
<134>1 2019-12-03T04:40:58Z SAIFU quilc - LOG0002 [rigetti@0000 methodName="quil_to_native_quil" requ
estID="90166593-2ccc-46af-9b56-0519a6443188" wallTime="0.172" error="false"] Requested quil_to_native
_quil completed
```

Gambar 2.21. Server quilc Saat Kode Dijalankan

```
<135>1 2019-12-03T04:40:00Z SAIFU qvm - - - Starting server on port 5000.
<135>1 2019-12-03T04:40:58Z SAIFU qvm - - - [127.0.0.1 Session:1] Got "version" request from API key/
User ID: NIL / NIL
<134>1 2019-12-03T04:40:58Z SAIFU qvm - - - 127.0.0.1 - [2019-12-03 11:40:58] "POST / HTTP/1.1" 200 1
6 "-" "python-requests/2.21.0"

<135>1 2019-12-03T04:40:58Z SAIFU qvm - - - [127.0.0.1 Session:1] Got "multishot" request from API ke
y/User ID: NIL / NIL
<135>1 2019-12-03T04:40:58Z SAIFU qvm - - - [127.0.0.1 Session:1] Making qvm of 9 qubits
<135>1 2019-12-03T04:40:58Z SAIFU qvm - - - [127.0.0.1 Session:1] Running experiment with 100 trials
on PURE-STATE-QVM
<135>1 2019-12-03T04:40:58Z SAIFU qvm - - - [127.0.0.1 Session:1] Finished in 15 ms
<134>1 2019-12-03T04:40:58Z SAIFU qvm - - - 127.0.0.1 - [2019-12-03 11:40:58] "POST /qvm HTTP/1.1" 20
0 2008 "-" "python-requests/2.21.0"
```

Gambar 2.22. Server QVM Saat Kode Dijalankan

```
1  # run the program on a QVM
2  qc = get_qc('9q-square-qvm')
3  result = qc.run_and_measure(p, trials=100)
4  print(result[0])
5  print(result[1])
```

```
[0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 0 1 0
 0 1 1 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1
 1 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1]
[0 0 0 0 1 1 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 1 0 0 0 0 0 1 0 1 0 1 0
 0 1 1 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 1 0 0 0 1
 1 0 0 1 1 1 1 0 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1]
```

Gambar 2.23. Hasil Collapse

Gambar 2.18 dan Gambar 2.19 adalah keadaan server quilc dan server QVM yang digunakan untuk menjalankan simulasi dapat berjalan dengan baik. Kedua server ini menunggu perintah untuk mengerjakan simulasi dari eksekusi kode dengan *library* pyQuil. Bila kedua server tersebut telah berjalan dengan baik, maka kode yang dibuat baru dapat dijalankan dalam simulator QVM. Kode yang dijalankan di-*compile* pada quilc dan dijalankan pada QVM. Hasil komputasi

kuantum di server QVM dapat ditampilkan sebagai hasil pengukuran diinginkan (PyQuil 2.14.0 Documentation, 2019).

Jumlah *qubit* yang dapat digunakan untuk simulasi yang dijalankan secara lokal akan bergantung pada spesifikasi komputer yang digunakan. Untuk komputer dengan RAM sebesar 8GiB dapat digunakan *qubit* sebanyak 26 *qubit*, dan untuk komputer dengan RAM sebesar 64GiB dapat menggunakan *qubit* sebanyak 29 *qubit*. Simulator lokal QVM dari Rigetti memanfaatkan penggunaan memori secara dinamis, sehingga memungkinkan simulasi dengan 29 *qubit* di komputer dengan RAM sebesar 16GiB (PyQuil 2.14.0 Documentation, 2019).

2.5 Metode Evaluasi F-Score

Metode evaluasi yang digunakan adalah metode akurasi dan *F-score*. Metode akurasi dan *F-score* yang digunakan memanfaatkan jumlah percobaan yang memiliki nilai *true positive*, *true negative*, *true positive*, *false positive*, dan *false negative*. Nilai *true positive* dari percobaan adalah hasil pengukuran yang menghasilkan keluaran bernilai 1 untuk *state* yang memiliki nilai 1, nilai *true negative* dari percobaan adalah hasil pengukuran yang menghasilkan keluaran bernilai 0 untuk *state* yang memiliki nilai 0, nilai *false positive* dari percobaan adalah hasil pengukuran yang menghasilkan keluaran bernilai 1 untuk *state* yang memiliki nilai 0, dan nilai *false negative* dari percobaan adalah hasil pengukuran yang menghasilkan keluaran bernilai 0 untuk *state* yang memiliki nilai 1.

Metode akurasi adalah metode evaluasi akurasi yang mempertimbangkan seluruh hasil benar yang dihasilkan. Metode akurasi digunakan untuk melihat kedekatan hasil implementasi dengan hasil kebenaran yang diinginkan. Berikut ini adalah rumus dari akurasi (Alvarez, 2002).

$$\text{akurasi} = \frac{true\ positive + true\ negative}{true\ positive + false\ positive + true\ negative + false\ negative} \quad ...(2.2)$$

Metode *F-score* adalah metode evaluasi akurasi yang mempertimbangkan presisi dan *recall* dari tes. Presisi adalah jumlah hasil *true positive* dibandingkan seluruh jumlah hasil *positive* dari hasil tes (*true positive* + *false positive*). *Recall* adalah jumlah hasil *true positive* dibandingkan jumlah seluruh elemen *positive* dari hasil sebenarnya (*true positive* + *false negative*). *F-score* sendiri adalah rata-rata perbandingan nilai presisi dan *recall* dari hasil. Berikut ini adalah rumus dari presisi, *recall*, dan *F-score* (Sokolova, Japkowicz, dan Szpakowicz, 2006).

$$\text{precision} = \frac{true\ positive}{true\ positive + false\ positive} \quad ...(2.3)$$

$$\text{recall} = \frac{true\ positive}{true\ positive + false\ negative} \quad ...(2.4)$$

$$\text{F} - \text{score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \quad ...(2.5)$$