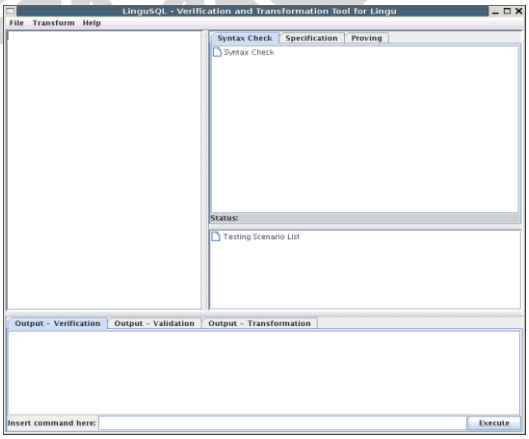
Bab III LinguSQL

Bagian ini akan mengenalkan kita kepada LinguSQL secara lebih detail. Kemudian akan dijelaskan juga langkah-langkah penggunaan dari LinguSQL.

3.1 Pengenalan LinguSQL

LinguSQL [PUR05] adalah sebuah *tool* yang berfungsi untuk melakukan verifikasi dan validasi suatu algoritma perangkat lunak. Algoritma perangkat lunak tersebut terdefinisikan dalam suatu bahasa abstrak yang disebut Lingu. Bahasa abstrak ini adalah bahasa yang menghasilkan kode tetapi tidak dapat berjalan secara langsung pada suatu sistem, sehingga harus diterjemahkan ke bahasa yang konkret seperti Java dan C [SUH07]. Lingu sendiri menekankan kepada operasi transformasi data pada aplikasi basis data. Lalu setelah proses verifikasi dan validasi berhasil dilaksanakan, kemudian LinguSQL akan melakukan proses transformasi bahasa Lingu menjadi sebuah bahasa konkrit dalam hal ini bahasa java sehingga dapat digunakan oleh para pengembang karena telah dijamin melalui proses pengujian *blackbox* dan juga *whitebox*. Kemampuan LinguSQL dalam menjalani dua macam teknik *testing* yang berbeda yakni *whitebox* dan juga *blackbox*, menjadikannya sangat dapat diandalkan oleh para pengembang perangkat lunak. Berikut ini adalah gambar tampilan utama dari LinguSQL.



Gambar 3.1 Tampilan Utama LinguSQL

Pada LinguSQL terdapat tiga buat mesin utama. Mesin-mesin itu ialah mesin verifikasi, mesin validasi, dan mesin transformasi. Tiap-tiap mesin tersebut memiliki fungsinya masing-masing dan bekerja secara berurutan.

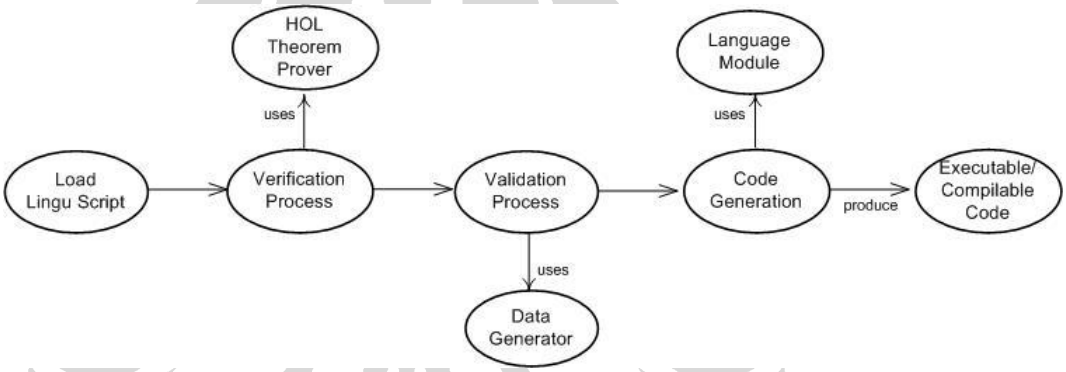
Mesin verifikasi berfungsi untuk melakukan proses verifikasi pada suatu algoritma perangkat lunak yang telah didefinisikan dengan menggunakan Lingu. Mesin ini akan mengubah spesifikasi yang didefinisikan dalam Lingu ke dalam formula metematis. Pada proses verifikasi ini, mesin ini dibantu oleh suatu *theorem prover* yaitu HOL agar bisa membuktikan setiap formula preposisi matematis yang dimasukkan ke dalam *verification conditions* (VCs). VCs adalah kumpulan formula preposisi matematis yang harus diverifikasi kebenarannya [SUH07]. Pada prakteknya, dikarenakan sifat HOL yang tidak bisa ditebak, maka dibutuhkan tenaga ahli untuk membantu penggunaan *theorem prover* HOL. Pada proses verifikasi ini digambarkan bahwa sebuah perangkat lunak sedang diuji dengan menggunakan metode *whitebox.*

Mesin selanjutnya adalah mesin validasi. Mesin ini berfungsi untuk menjalankan scenario yang sudah didefinisikan dalam Lingu dan membandingkannya hasilnya dengan spesifikasinya. Berbeda dengan terminologi validasi pada *software engineering*, validasi di sini merupakan proses untuk menunjukkan seberapa besar nilai keakuratan program terhadap kondisi-kondisi saat pemakaian sebenarnya [SUH07]. Hal ini bisa dilakukan karena LinguSQL mempunyai kemampuan untuk memberikan data *dummy* yang bisa berfungsi sebagai pengganti dari data yang sebenarnya akan digunakan sistem. Data-data ini tetap berdasarkan batasan-batasan yang ada pada algoritma program. Oleh karena itu, data tersebut bisa menjadi *data testing* untuk menguji sistem. Hasil dari mesin ini adalah catatan input yang dapat menggagalkan *testing* [SUH07].

Mesin yang terakhir yaitu mesin transformasi. Mesin ini berfungsi dalam melakukan proses tranformasi bahasa Lingu menjadi sebuah bahasa konkrit dalam hal ini bahasa java. Penerjemah yang ada merupakan hasil implementasi menggunakan JavaCC dan Atribute Grammar [JIM05]. Bahasa konkrit ini diperlukan untuk kemudian diimplementasikan untuk dijalankan di lingkungan kerja suatu organisasi.

3.2 Cara Kerja LinguSQL

Tahap-tahap kerja LinguSQL merupakan proses yeng terstruktur sehingga dari sebuah bahasa abstrak bisa menjadi sebuah bahasa konkrit. Proses-proses tersebut dijalankan dengan menggunakan mesin-mesin yang telah dijelaskan sebelumnya. Setiap mesin pada LinguSQL bekerja secara berurutan demi menghasilkan tujuan yang diinginkan. Secara umum alur kerja LinguSQL dapat dilihat pada gambar berikut ini.



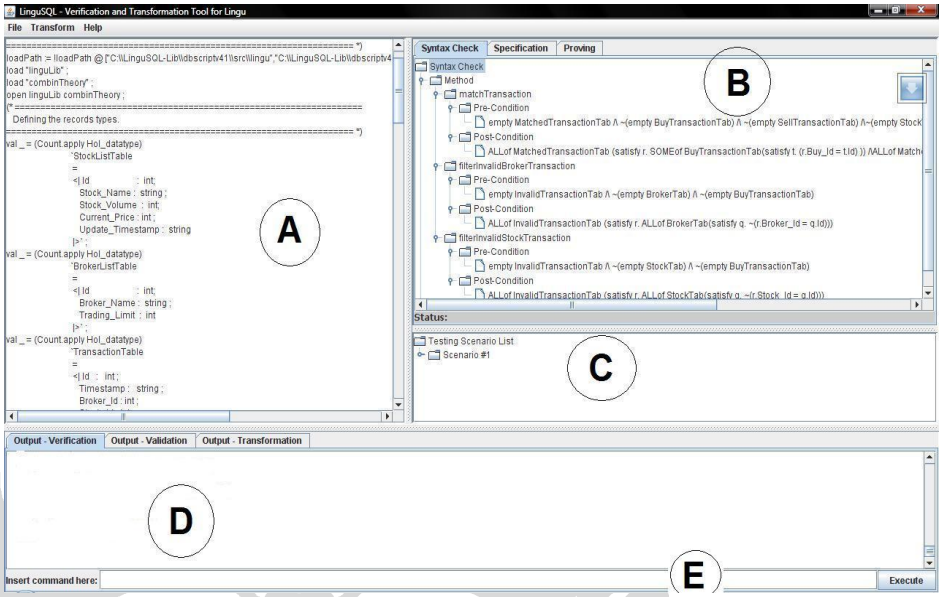
Gambar 3.2 Alur Kerja LinguSQL [PUR08]

Dalam penggunaan LinguSQL terhadap pengembangan dan pengujian suatu aplikasi, maka tahap-tahap yang harus dilakukan [PUR08] adalah sebagai berikut.

1. **Tahap pendefinisian data dan fungsi**

   Tahapan ini dilakukan dengan cara mengidentifikasikan data-data yang terlibat dalam *domain* permasalahan aplikasi. Data tersebut harus didefinisikan dalam bentuk tabel di sebuah basis data. Selanjutnya adalah pendefinisian fungsi atau *method* apa saja yang akan diujikan dengan mengunakan LinguSQL. Pemilihan fungsi yang akan diujikan sepenuhnya tergantung dari tingkat kekritisan alur fungsi tersebut. Proses ini menghasilkan sebuah skrip dasar yang berisi struktur data dan *method.* Setelah skrip Lingu berhasil dibuat, kemudian skrip Lingu dimasukan ke dalam LinguSQL. Ketika

memasukan skrip Lingu pada LinguSQL maka berikut ini adalah tampilan dari LinguSQL.

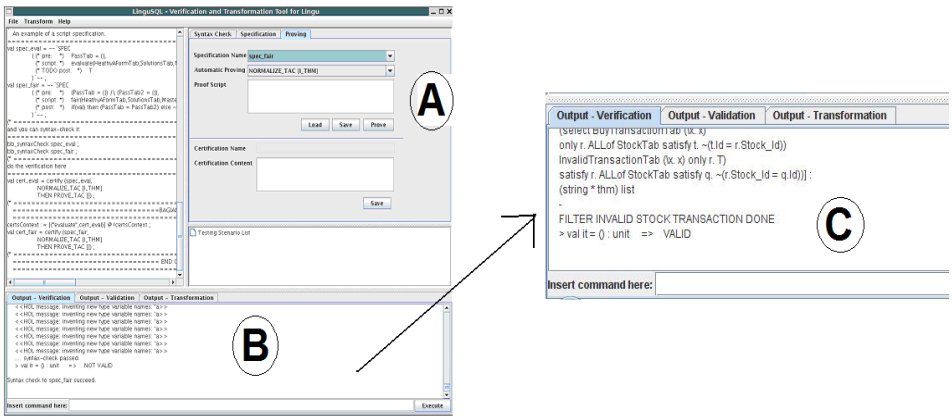


Gambar 3.3 Tampilan LinguSQL [PUR08]

Pada bagian A terdapat panel yang berfungsi sebagai penampil skrip Lingu sehingga bisa dilakukan proses *review* dan *edit.* Bagian B adalah bagian dimana akan ditampilkan daftar dari seluruh elemen *precondition* dan *postcondition* dari skrip Lingu yang ditampilkan pada bagian A. Bagian C adalah bagian dimana akan ditampilkan skenario yang akan dilaksanakan pada proses validasi. Bagian D adalah panel dimana akan ditampilkan hasil dari seluruh proses yang berlangsung. Sementara itu bagian E akan digunakan untuk membantu proses *theorem proving* yang dilakukan oleh HOL.

2. Tahap verifikasi skrip Lingu

Tahap verifikasi dilakukan untuk memastikan kebenaran alur algoritma dari prosedur yang terdapat di skrip Lingu. Verifikasi dilakukan dengan memproses skrip Lingu ke dalam HOL *theorem prover*. LinguSQL akan menangkap hasil dari *theorem prover* dan menerjemahkan apakah skrip tersebut lolos dari verifikasi atau tidak. Gambar berikut ini dapat menjelaskan ketika proses verifikasi berhasil dilaksanakan.

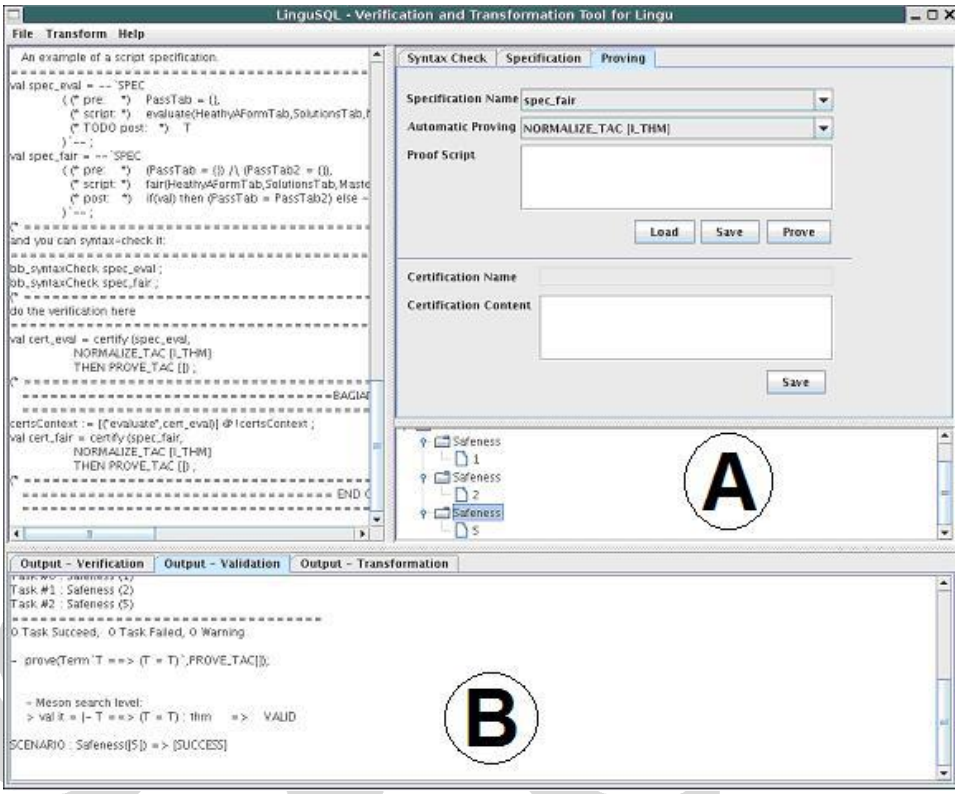


Gambar 3.4 Hasil Verifikasi oleh LinguSQL [PUR08]

Ketika akan menguji suatu skrip, pengembang dapat memilih spesifikasi yang telah terdaftar pada bagian B gambar 3.3. Setelah memilih, kemudian pengembang dapat memilih taktik dan teknik pembuktian yang akan digunakan seperti yang dapat dilihat pada bagian A gambar 3.4. Hasil dari verifikasi akan ditampilkan seperti pada bagian C gambar 3.4.

3. Tahap validasi skrip Lingu

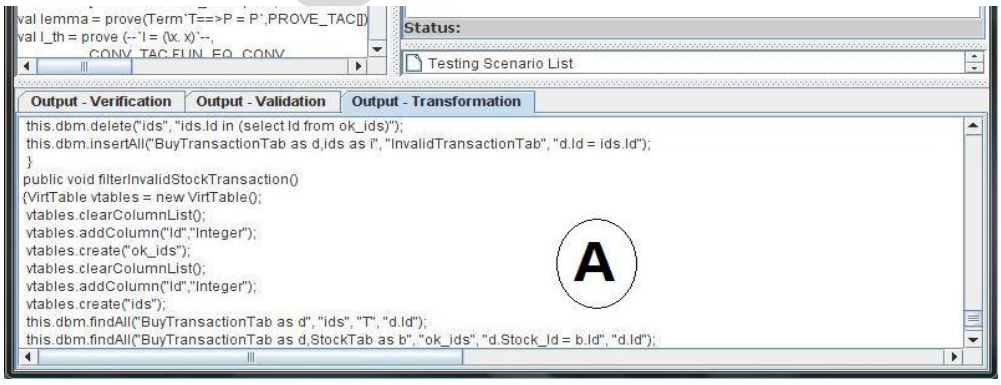
Tahap validasi digunakan untuk melakukan pengujian apakah eksekusi prosedur yang telah didefinisikan di dalam definisi Lingu akan menghasilkan *output* yang sesuai harapan. Untuk melaksanakan pengujian ini, LinguSQL dilengkapi dengan sebuah *data generator* yang dapat menghasilkan sampel data terstruktur untuk digunakan sebagai input prosedur skrip Lingu. Proses validasi dianggap berhasil jika seluruh pengujian eksekusi kode validasi mengalami keberhasilan. Proses pengujian ini dilakukan dengan menggunakan skenario yang telah dirancang sebelumnya. Pemilihan skenario dapat dilakukan dengan memilih pada panel bagian A gambar 3.5. Kemudian hasil dari proses validasi bisa dilihat pada bagian B gambar 3.5.



Gambar 3.5 Proses dan Hasil Validasi Pada LinguSQL [PUR08]

4. Tahap transformasi ke bahasa konkrit

Tahap terakhir dari pengembangan ini adalah pengubahan skrip Lingu menjadi bentuk bahasa pemrograman yang konkrit seperti java. Proses transformasi ini menggunakan *atribute grammar* dan JavaCC. Hasil dari proses transformasi yang berupa skrip java dapat dilihat pada bagian A Gambar 3.6.



Gambar 3.6 Hasil Proses Transformasi Pada LinguSQL[PUR08]

Bab IV Aplikasi Perbankan

Dalam tugas akhir ini akan digunakan suatu studi kasus yaitu aplikasi perbankan yang telah didefinisikan oleh Martin Büchi dalam bahasa spesifikasi B. Spesifikasi aplikasi perbankan dalam bahasa B dapat dilihat pada Lampiran A. Aplikasi perbankan ini meliputi sistem yang digunakan oleh *teller* dan juga sistem yang digunakan pada mesin *Auto Teller Machine* (ATM). Dalam sistem yang digunakan oleh *teller*, pengguna dapat mendaftarkan nasabah baru, membuatkan rekening untuk nasabah dan juga menerima tabungan dari nasabah. Sementara itu, dalam sistem yang berjalan pada ATM, pengguna dapat mengambil uang, melihat saldo tabungan, serta mengubah kode PIN nasabah. Secara lebih detil analisa kebutuhan dari aplikasi ini adalah sebagai berikut.

1. Nasabah didaftarkan pada sistem dengan dibuat nomor identifikasinya dengan kombinasi dari nama dan tahun lahir. Diasumsikan bahwa tidak ada nasabah yang memiliki kombinasi yang sama dari nama dan tahun lahir.
2. Nasabah dapat memiliki banyak rekening.
3. Setiap rekening memiliki nomor yang unik.
4. Setiap rekening tepat dimiliki oleh satu nasabah yang ada dalam basis data.
5. Saldo rekening tidak dapat bernilai negatif.
6. Nasabah memiliki PIN rahasia di setiap rekening yang dia punya.
7. *Teller* dapat mendaftarkan nasabah ke dalam sistem dengan memberikan data berupa nama dan tanggal lahirnya.
8. *Teller* dapat membuatkan rekening dengan saldo awal nol kepada nasabah dengan memasukan data nasabah dan juga PIN. Selanjutnya PIN dapat diubah oleh nasabah.
9. *Teller* dapat menerima tabungan dengan meminta data nomor rekening kepada nasabah. Diasumsikan bahwa *teller* tidak akan melakukan kesalahan saat memasukan data uang tabungan yang masuk ke dalam sistem.

10. Nasabah dapat mengambil uang dalam rekeningnya hingga sejumlah keseluruhan dari saldo rekening.
11. Nasabah dapat melihat saldo dari rekening tabungannya.
12. Nasabah dapat mengubah kode PIN dengan memasukan PIN lama dan PIN baru yang valid.

Aplikasi perbankan dipilih sebagai studi kasus dikarenakan aplikasi ini merupakan aplikasi basis data yang bersifat kritis. Aplikasi perbankan mengelola data dalam jumlah besar yang meliputi data diri nasabah serta data rekening nasabah. Dikarenakan data yang ditanganinya teramat besar maka aplikasi ini harus akurat dan handal. Jika aplikasi ini tidak dapat akurat dan handal maka bank dan juga nasabah akan mengalami kerugian yang amat besar.

4.1 Data Aplikasi Bank

Data yang dipakai dalam aplikasi perbankan yang didefinisikan oleh Martin Büchi adalah sebagai berikut:

4.1.1 Tipe *Records*

Terdapat tiga tipe *records* dalam aplikasi perbankan ini, yaitu:

1. **CustomerTable** merepresentasikan tipe *records* yang berisi data-data dari nasabah pada aplikasi perbankan. Data terdiri dari nomor identitas (ID), nama, dan tahun lahir. ID adalah kombinasi dari nama dan tahun lahir nasabah.
2. **AccountTable** merepresentasikan tipe *records* yang berisi data tentang rekening yang dimiliki oleh nasabah. Data terdiri dari nomor rekening, nomor PIN, nomor identitas dari nasabah yang memiliki rekening, dan data saldo dari rekening tersebut.
3. **TransactionTable** merepresentasikan tipe *records* yang berisi data tentang transaksi yang terjadi dalam suatu rekening. Data terdiri dari ID dari transaksi itu sendiri, nomor identitas dari nasabah yang melakukan transaksi, nomor rekening dimana terjadinya transaksi, jumlah transaksi, serta nomor PIN.

4.1.2 Tipe Tabel

Database **Bankdb** merepresentasikan basis data dari seluruh kegiatan pelaksanaan dari aplikasi perbankan. Database ini terdiri dari beberapa tabel yaitu:

1. **DepositTab** adalah tabel dari data transaksi pemasukan uang (*deposit*) yang terjadi dalam aplikasi perbankan. *Records* dari **DepositTab** merupakan tipe **TransactionTable**. Pada **DepositTab**, atribut nomor PIN kosong. Hal ini dikarenakan pada saat melakukan *deposit* tidak dibutuhkan data nomor PIN.
2. **WithdrawTab** adalah tabel dari data transaksi pengambilan uang (*withdraw*) yang terjadi dalam aplikasi perbankan. *Records* dari **WithdrawTab** merupakan tipe **TransactionTable**.
3. **CustomersTab** adalah tabel dari data induk dari seluruh nasabah yang terdaftar dalam aplikasi perbankan. *Records* dari **CustomersTable** merupakan **CustomerTable**.
4. **AccountsTab** adalah tabel dari data induk seluruh rekening yang terdaftar dalam aplikasi perbankan. *Records* dari **AccountsTable** merupakan **AccountTable**.
5. **InvalidTransactionTab** adalah tabel dari semua transaksi yang *invalid*. *Records* dari **InvalidTransactionTab** merupakan bagian dari tipe **TransactionTable**.
6. **SuccessfulTransactionTable** adalah tabel dari semua transaksi yang sukses. *Records* dari **SuccessfulTransactionTable** merupakan bagian dari tipe **TransactionTable**.

4.2 Modul Kritis

Definisi modul kritis adalah modul yang dipilih karena dianggap akan bermasalah. Untuk menghindari permasalahan tersebut, pada modul-modul kritis tersebut kita lakukan terlebih dahulu identifikasi masalah agar masalah tersebut mudah untuk dihindari. Pada aplikasi perbankan telah dipilih lima modul kritis yang harus diverifikasi. Modul tersebut adalah:

4.2.1 Modul Deposit

Modul ini digunakan untuk mencatat data transaksi *deposit* ke dalam log yang berupa tabel **SuccessfulTransactionTab**. Tabel yang berkaitan dengan modul ini adalah **DepositTab**, **SuccessfulTransactionTab** dan **AccountsTab**. Modul ini akan menambahkan data jumlah transaksi dalam tabel **DepositTab** dengan data saldo yang ada pada tabel **AccountsTab** dan menyimpannya pada tabel **AccountsTab**. Transaksi deposit yang ada akan dimasukkan ke dalam **SuccessfulTransactionTab** sebagai log.

4.2.2 Modul Withdraw

Modul ini digunakan untuk mencatat data transaksi *withdraw* ke dalam log yang berupa tabel **SuccessfulTransactionTab**. Tabel yang berkaitan dengan modul ini adalah **WithdrawTab**, **SuccessfulTransactionTab** dan **AccountsTab**. Modul ini akan mengurangi saldo pada tabel **AccountsTab** dengan data jumlah transaksi yang terdapat pada tabel **WithdrawTab** tersebut dan memperbaharui data saldo pada tabel **AccountsTab**. Modul ini juga memastikan kalau uang yang akan diambil dari rekening tidak boleh lebih besar dari nilai saldo yang tercatat dalam rekening. Transaksi *withdraw* yang ada dimasukkan ke dalam **SuccessfulTransactionTab** sebagai log.

4.2.3 Modul Filter Invalid Customer

Modul ini digunakan untuk memeriksa transaksi dari nasabah yang *invalid*. Definisi dari nasabah yang *invalid* adalah nasabah yang tidak tercatat dalam sistem dan nasabah yang bukan pemilik dari rekening yang akan digunakan. Tabel yang berkaitan dengan modul ini adalah **DepositTab**, **WithdrawTab**, **CustomersTab**, **AccountsTab**, dan **InvalidTransactionTab**. Sebelum modul ini dilaksanakan tabel **InvalidTransactionTab** tidak memiliki data apapun. Program ini berjalan dengan cara mencari ID nasabah pada tabel **DepositTab** atau **WithdrawTab** dan memeriksanya apakah ID tersebut ada pada tabel **CustomersTab** atau tidak. Jika tidak ditemukan, maka data pada **DepositTab** atau **WithdrawTab** yang ID nasabahnya tidak cocok atau tidak ditemukan pada C**ustomerTab** akan dipindahkan

pada tabel **InvalidTransactionTab**. Hal ini menjelaskan bahwa nasabah itu tidak ada dalam sistem. Selain itu, program juga akan mencari nomor rekening yang ada pada **DepositTab** atau **WithdrawTab** dan memeriksa apakah nomor rekening itu benar dimiliki oleh nasabah yang tercatat pada tabel itu juga. Proses ini berjalan dengan memeriksa data pada tabel **AccountsTab**. Jika rekening bukan milik nasabah yang termaksud pada tabel **DepositTab** atau **WithdrawTab** maka nasabah dinyatakan *invalid*. Setelah itu data akan dimasukkan ke dalam **InvalidTransactionTab**.

4.2.4 Modul Filter Invalid Account

Modul ini digunakan untuk memeriksa transaksi dari rekening yang *invalid*. Definisi dari rekening yang *invalid* adalah rekening yang tidak tercatat dalam sistem. Tabel yang berkaitan dengan modul ini adalah **DepositTab**, **WithdrawTab**, **AccountsTab**, dan **InvalidTransactionTab**. Sebelum modul ini dilaksanakan tabel **InvalidTransactionTab** tidak memiliki data apapun. Program ini berjalan dengan cara mencari nomor rekening pada tabel **DepositTab** atau **WithdrawTab** dan memeriksanya apakah nomor rekening tersebut ada pada tabel **AccountsTab** atau tidak. Jika tidak ditemukan, maka data pada **DepositTab** atau **WithdrawTab** akan dipindahkan pada tabel **InvalidTransactionTab**. Hal ini menjelaskan bahwa nomor rekening itu tidak ada dalam sistem.

4.2.5 Modul Check PIN

Modul ini digunakan untuk memeriksa apakah fungsi withdraw dijalankan dengan memberikan nomor PIN yang tepat. Tabel yang berkaitan dengan modul ini adalah **WithdrawTab**, **AccountsTab**, **SuccessfulTransactionTab**, dan **InvalidTransactionTab**. Program ini berjalan dengan mencari data nomor PIN yang ada pada tabel **WithdrawTab** dan mencocokkannya pada data yang ada pada tabel **AccountsTab**. Jika ditemukan transaksi yang tidak sesuai data nomor PIN-nya, maka data transaksi tersebut akan dimasukkan ke dalam tabel **InvalidTransactionTab.**

Bab V Implementasi dan Verifikasi Aplikasi Perbankan

Dalam bab ini akan dijelaskan implementasi dan verifikasi aplikasi perbankan yang permasalahannya sudah dijelaskan pada bab sebelumnya. Pada pelaksanaan implementasi dan verifikasi dibutuhkan langkah-langkah untuk memenuhi tujuan dari kegiatan tersebut. Langkah-langkah itu adalah:

1. Mendefinisikan basis data yang diperlukan.
2. Membuat definisi-definisi untuk menyederhanakan penulisan kode.
3. Membuat skrip Lingu v2 sesuai dengan spesifikasi yang telah dibuat dan setelah itu memverifikasikannya.

Skrip Lingu v2 yang dibuat disimpan dalam ekstensi **.smx**. Setelah skrip Lingu v2 dibuat kemudian akan dilakukan pembuktian dengan menggunakan *theorem prover* HOL Kananaskis 3. Hal ini seperti dijelaskan pada bab-bab awal bahwa dalam pelaksanaanya Lingu sebagai bahasa abstrak akan membutuhkan *theorem prover* untuk melakukan pembuktian program.

5.1 Basis Data Aplikasi Perbankan

Basis data aplikasi perbankan dibuat sesuai dengan penjelasan yang ada pada bab 4 tentang studi kasus aplikasi perbankan. Basis data aplikasi perbankan dalam file Lingu v2 dibuat dengan nama **bankdb** yang terdiri dari beberapa tabel yaitu **DepositTab**, **WithdrawTab**, **CustomersTab**, **AccountsTab**, **SuccessfulTransactionTab**, **InvalidTransactionTab**. Tabel- tabel dari basis data ini memiliki data *records* yang berbeda-beda sesuai dengan apa yang juga disampaikan pada bab sebelumnya. Data mengenai *records* tiap-tiap tabel basis data dapat dilihat pada Tabel 5.1. Sementara itu penulisan basis data aplikasi perbankan dalam Lingu v2 dapat dilihat pada gambar 5.1.

Tabel 5.1 Tabel-tabel dalam basis data bankdb

Nama Tabel	Tipe *records*
DepositTab	TransactionTable
WithdrawTab	TransactionTable
CustomersTab	CustomerTable
AccountsTab	AccountTable
SuccessfulTransactionTab	TransactionTable
InvalidTransactionTab	TransactionTable

```
42 (* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
43
44   Defining Database
45
46 ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
47
48 val _ =(Count.apply Hol_datatype)
49   `Bankdb
50   =
51   <| DepositTab : TransactionTable;
52      WithdrawTab : TransactionTable;
53      CustomersTab : CustomerTable;
54      AccountsTab : AccountTable;
55      SuccessfulTransactionTab : TransactionTable;
56      InvalidTransactionTab : TransactionTable
57   |>`;
```

Gambar 5.1 Kode penulisan basis data bankdb dalam Lingu v2

Data-data mengenai *records* seperti yang dijelaskan pada studi kasus aplikasi perbankan secara lebih detail dapat dilihat berikut ini:

1. Tabel CustomerTable

   Tabel ini berisi data nasabah yang terdaftar dalam aplikasi perbankan. Data yang ada pada tabel ini meliputi data nomor identitas nasabah, nama nasabah, dan tahun lahir nasabah. Penjelasan tabel ini dapat dilihat pada Tabel 5.2.

Tabel 5.2 Ringkasan Data *Records* Tipe CustomerTable

Data Element	Tipe Data
ID	Int
Nama	String
TahunLahir	Int

2. Tabel AccountTable

   Tabel ini berisi data rekening pada aplikasi perbankan. Data yang ada pada tabel ini meliputi nomor rekening, nomor PIN, nomor identitas nasabah, dan saldo rekening. Penjelasan tabel ini dapat dilihat pada Tabel 5.3.

Tabel 5.3 Ringkasan Data *Records* Tipe AccountTable

Data Element	Tipe Data
ID	Int
NomorPIN	Int
CustomerID	Int
Saldo	Int

3. Tabel TransactionTable

   Tabel ini berisi data transaksi pada aplikasi perbankan. Data yang ada pada tabel ini meliputi nomor ID transaksi, nomor identitas nasabah, nomor rekening nasabah, jumlah transaksi, dan juga nomor PIN sebagai input saat melakukan transaksi penarikan uang (*withdraw*). Penjelasan tabel ini dapat dilihat pada Tabel 5.4.

Tabel 5.4 Ringkasan Data *Records* Tipe TransactionTable

Data Element	Tipe Data
ID	Int
NomorPIN	Int
CustomerID	Int
Amount	Int
AccountID	int

Kemudian ketiga tabel ini dijelaskan dalam bahasa Lingu v2 untuk keperluan spesifikasi dan juga verifikasi. Penulisannya dalam bahasa Lingu v2 dapat dilihat pada gambar 5.2.

```
(* ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
  Defining The Records Types
:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: *)
val _ = (Count.apply Hol_datatype)
  `CustomerTable
  =
  <| ID : int;
     Nama : string;
     TahunLahir : int
  |>`;
val _ = (Count.apply Hol_datatype)
  `AccountTable
  =
  <| ID : int;
     NomorPIN : int;
     CustomerID : int;
     Saldo : int
  |>`;
val _ = (Count.apply Hol_datatype)
  `TransactionTable
  =
  <| ID : int;
     CustomerID : int;
     AccountID : int;
     Amount : int;
     NomorPIN : int
  |>`;
```

Gambar 5.2 Kode Lingu v2 untuk Mendefinisikan *Records* Aplikasi Bank

5.2 Implementasi Modul Kritis

Setelah mendefinisikan basis data dari aplikasi perbankan, untuk memulai proses spesifikasi dan verifikasi harus dilakukan penjelasan dari modul-modul kritis dalam Lingu v2. Penjelasan ini sangat erat kaitannya dengan tujuan akhir proses verifikasi terhadap aplikasi perbankan. Untuk mempermudah penulisan formula pada Lingu v2, penjelasan dari modul-modul kritis perlu diringkas dengan mendefinisikan beberapa definisi tambahan yang akan meringkas skrip spesifikasi. Definisi-definisi tersebut adalah:

- empty

  Definisi tambahan ini berguna untuk menjelaskan bahwa suatu tabel kosong atau tidak mempunyai suatu *records* sama sekali [SUH07]. Definisi ini berjalan dengan menjelaskan lewat formula matematis bahwa tidak terdapat *records* pada tabel itu yang memenuhi kondisi True. Penulisan dalam Lingu v2 dapat dilihat pada gambar 5.3.
- ALL

  Definisi ini menjelaskan bahwa suatu tabel akan diambil seluruh *records* yang bernilai True [SUH07]. Definisi ini ditunjukan dengan formula matematis bahwa yang akan diambil adalah suatu *records* pada tabel yang hanya bernilai True saja. Penulisan dalam Lingu v2 juga dapat dilihat apda gambar 5.3.

```
68 val empty_def = Define `empty t = ~ SOMEof t (satisfy r. T)` ;
69
70 val ALL_def   = Define `ALL = (only r. T)` ;
```

Gambar 5.3 Definisi-definisi Tambahan

5.2.1 Pembuatan Taktik

Untuk membantu HOL bekerja dalam verifikasi skrip lingu, diperlukan suatu bantuan berupa *lemma* yang akan memudahkan HOL dalam membuktikan suatu formula. *Lemma* [SUH07] disini adalah teorema yang sederhana dan dipakai dalam

pembuktian teorema lain yang lebih kompleks. Dalam studi kasus ini, *lemma* yang digunakan adalah teorema yang menyatakan bahwa suatu pernyataan dimana premisnya bernilai True dan kesimpulannya suatu preposisi P maka kebenaran pernyataan tersebut hanya diperiksa pada preposisi P. notasi yang menggambarkan *lemma* tersebut adalah (T ==> P)P. Dalam HOL *lemma* ini akan dibuktikan dalam PROVE_TAC[]. Penulisan taktik dalam Lingu bisa dilihat pada gambar 5.4 berikut ini.

```
79 (* ========================================================================
80
81    List for verification ...
82
83 ======================================================================== *)
84
85 val lemma = prove(Term`T==>P = P`,PROVE_TAC[]);
86 val I_th = prove (--`I = (\x. x)`--,
87                     CONV_TAC FUN_EQ_CONV
88                     THEN RW_TAC std_ss [I_THM]) ;
89
90 val defs = [lemma, I_th, ALL_def, empty_def] ;
91
92 val MY_TAC = lingu_normalizer_TAC
93              THEN RW_TAC std_ss records_thms
94              THEN PROVE_TAC [lemma] ;
```

Gambar 5.4 Kode Lingu untuk Taktik

Keseluruhan dari taktik yang digunakan dalam studi kasus ini dibungkus di dalam MY_TAC yang berisikan lingu_normalizer_TAC yang merupakan taktik yang terdapat dalam *library* Lingu, setelah itu RW_TAC std_ss records_thms dan terakhir PROVE_TAC [lemma].

5.2.2 Modul Deposit

Modul ini digunakan untuk menyimpan data jumlah uang yang ditabung ke dalam saldo rekening pada aplikasi perbankan. Setelah data tersebut disimpan kemudian data saldo akan diperbarui sesuai dengan jumlah uang yang ditabungkan.

5.2.2.1 Parameter

Parameter yang digunakan pada modul ini ada yang bertipe *pass by reference* dan juga *pass by value*. Parameter-parameter yang bertipe *pass by reference* meliputi tabel DepositTab, SuccessfulTransactionTab dan AccountsTab. Paremeter yang bertipe *pass by value* adalah r. DepositTab merupakan tabel yang menyimpan data transaksi yang berupa penyimpanan uang atau *deposit*. Data-data pada tabel ini akan coba dimasukkan ke dalam tabel AccountsTab sebagai pembaruan data saldo yang ada pada tabel AccountsTab. Parameter r berfungsi sebagai *record* tambahan yang akan dimasukkan ke dalam tabel DepositTab.

5.2.2.2 Pre Post Condition

Kondisi sebelum program ini digunakan adalah kosongnya isi dari tabel DepositTab. Hal ini dimaksudkan karena jika ada transaksi baru maka tabel DepositTab akan bertambah *records*.

Kondisi akhir dari program ini adalah tidak kosongnya isi dari tabel SuccessfulTransactionTab. Penulisan dalam bahasa Lingu v2 dapat dilihat pada gambar 5.5.

```
(*===== DEPOSIT =====*)
print("\n \n *===== DEPOSIT =====*\n");
val Deposit_def = Define
  `Deposit( REF (DepositTab  : TransactionTable set),
            REF (AccountsTab : AccountTable set),
            REF (SuccessfulTransactionTab : TransactionTable set),r
          )
        =

         pre  empty DepositTab /\ ~empty AccountsTab
         post ~empty SuccessfulTransactionTab
     do /(

      ins r DepositTab;
      update AccountsTab(map c.c with Saldo := c.Saldo+r.Amount)(only c.c.ID=r.AccountID);

      let
      success = select DepositTab(only s.T)
      in
      insert success SuccessfulTransactionTab I ALL
```

Gambar 5.5 Kode Lingu untuk Deposit

5.2.2.3 Proses

Pertama kali program ini akan memasukan *records* ke dalam tabel DepositTab. Selanjutnya program akan meng-*update* data saldo yang ada pada tabel AccountsTab dengan data yang ada pada tabel DepositTab yang baru saja dimasukkan. Kemudian data pada DepositTab akan dimasukkan ke dalam SuccessfulTransactionTab.

5.2.2.4 Verifikasi

Pada awal proses verifikasi, semua definisi yang dibuat akan direduksi pada modul ini. Setelah proses reduksi, kemudian akan dilakukan pemanggilan fungsi auto verifikasi yang terdapat pada *library* LinguLib dengan menggunakan taktik MY_TAC yang telah didefinisikan sebelumnya. Hasil verifikasi modul ini dapat dilihat pada Lampiran. Hasil verifikasi dapat dilihat pada Lampiran B.

5.2.3 Modul Withdraw

Modul ini digunakan untuk menyimpan data jumlah uang yang ditarik oleh nasabah dari saldo rekening nasabah tersebut. Setelah data tersebut disimpan kemudian data saldo akan diperbarui sesuai dengan jumlah uang yang ditarik dari rekening tersebut.

5.2.3.1 Parameter

Parameter yang digunakan pada modul ini ada yang bertipe *pass by reference* dan juga *pass by value*. Parameter-parameter yang bertipe *pass by reference* meliputi tabel WithdrawTab, SuccessfulTransactionTab dan AccountsTab. Paremeter yang bertipe *pass by value* adalah r. WithdrawTab merupakan tabel yang menyimpan data transaksi yang berupa pengambilan uang atau *withdraw*. Data Saldo pada tabel AccountsTab akan diperbarui sesuai dengan data *Amount* yang terdapat pada tabel WithdrawTab. Parameter r berfungsi sebagai *record* tambahan yang akan dimasukkan ke dalam tabel DepositTab.

5.2.3.2 Pre Post Condition

Kondisi sebelum program ini digunakan adalah kosongnya tabel WithdrawTab. Hal ini dikarenakan data akan dimasukkan ke dalam tabel jika benar-benar terjadi transaksi.

Kondisi akhir dari program ini adalah tidak kosongnya tabel SuccessfulTransactionTab. Penulisan dalam bahasa Lingu v2 dapat dilihat pada Gambar 5.6.

5.2.3.3 Proses

Pertama kali program ini akan memasukkan *records* pada tabel WithdrawTab. Setelah itu program akan meng-*update* data saldo dengan menguranginya dengan data Amount yang terdapat pada tabel WithdrawTab. *Records* yang dipilih harus *records* yang tepat dengan mensyaratkan nomor identitas nasabah pada WithdrawTab harus sama dengan yang ada pada tabel AccountsTab. Selain itu, jumlah uang yang diambil juga harus lebih kecil atau sama dengan jumlah saldo yang ada pada rekening. Setelah itu data pada WithdrawTab akan dimasukkan ke dalam SuccessfulTransactionTab.

5.2.3.4 Verifikasi

Pada awal proses verifikasi, semua definisi yang dibuat akan direduksi pada modul ini. Setelah proses reduksi, kemudian akan dilakukan pemanggilan fungsi auto verifikasi yang terdapat pada *library* LinguLib dengan menggunakan taktik MY_TAC yang telah didefinisikan sebelumnya. Hasil verifikasi modul ini dapat dilihat pada Lampiran. Hasil verifikasi dapat dilihat pada Lampiran B.

```
(*===== WITHDRAW =====*)
print("\n \n *===== WITHDRAW =====*\n");
val Withdraw_def = Define
  `Withdraw( REF (WithdrawTab : TransactionTable set),
             REF (AccountsTab : AccountTable set),r
           )
         =

          pre  empty WithdrawTab /\ ~empty AccountsTab
          post ~empty SuccessfulTransactionTab
   do /{

      ins r WithdrawTab;
      update AccountsTab(map c.c with Saldo := c.Saldo-r.Amount)(only c.(c.ID=r.AccountID)/\(c.Saldo>=r.Amount));

      let
      success = select WithdrawTab(only s.T)
      in
      insert success SuccessfulTransactionTab I ALL

            /} return void`;
```

Gambar 5.6 Kode Lingu untuk Withdraw

5.2.4 Modul Filter Invalid Customer

Modul ini digunakan untuk menyaring nasabah yang tidak terdata dalam sistem aplikasi perbankan. Selain itu, modul ini juga akan menyaring nasabah yang bukan pemilik dari suatu rekening.

5.2.4.1 Parameter

Parameter yang digunakan pada modul ini keseluruhannya bertipe *pass by reference* dari tabel DepositTab, WithdrawTab, CustomersTab, AccountsTab, dan InvalidTransactionTab. DepositTab dan WithdrawTab adalah tabel yang berisi data-data transaksi menurut namanya masing-masing yaitu *deposit* atau *withdraw*. CustomersTab adalah tabel yang berisi data-data nasabah yang ada dalam aplikasi perbankan. AccountsTab adalah tabel yang menyimpan data rekening yang dimiliki oleh nasabah yang ada di dalam aplikasi perbankan. Seluruh data transaksi dengan nasabah *invalid* atau rekening yang *invalid* akan dimasukkan ke dalam tabel InvalidTransactionTable.

5.2.4.2 Pre Post Condition

Kondisi sebelum program dijalankan adalah tabel InvalidTransactionTable merupakan tabel yang kosong. Kondisi ini dapat dilihat selengkapnya dalam bahasa Lingu v2 pada gambar 5.7.

Kondisi akhir sesudah program dijalankan adalah seluruh data dalam tabel InvalidTransactionTable memenuhi *records* yang ID-nya tidak ada yang sama dengan *records* yang ada pada tabel CustomersTab dan ID yang ada pada CustomersTable harus sama dengan yang ada pada tabel AccountsTable.

5.2.4.3 Proses

Pada implementasi ini, proses yang berjalan adalah menghapus data dalam tabel DepositTab dan WithdrawTab yang terdaftar dalam tabel CustomersTab dengan data pada AccountsTab harus bersesuaian dengan data yang ada pada CustomersTab. Selanjutnya data yang tersisa adalah data yang *invalid* dan akan dimasukkan dalam tabel InvalidTransactionTable.

5.2.4.4 Verifikasi

Verifikasi pada modul ini tidak berbeda pada modul-modul sebelumnya yaitu pada awal proses verifikasi, semua definisi yang dibuat akan direduksi pada modul ini. Setelah proses reduksi, kemudian akan dilakukan pemanggilan fungsi auto verifikasi yang terdapat pada *library* LinguLib dengan menggunakan taktik MY_TAC yang telah didefinisikan sebelumnya. Hasil verifikasi modul ini dapat dilihat pada Lampiran. Hasil verifikasi dapat dilihat pada Lampiran B.

```
(*===== FILTER INVALID CUSTOMER =====*)
print("\n \n *===== FILTER INVALID CUSTOMER =====*\n");
val FilterInvalidCustomer_def = Define
  ` FilterInvalidCustomer(
    REF (DepositTab : TransactionTable set),
    REF (WithdrawTab : TransactionTable set),
    REF (CustomersTab : CustomerTable set),
    REF (AccountsTab : AccountTable set),
    REF (InvalidTransactionTab : TransactionTable set)
  )
   =
   pre  (empty InvalidTransactionTab)
   post (
    ALLof InvalidTransactionTab ( satisfy r.~SOMEof CustomersTab ( satisfy
t.SOMEof AccountsTab(satisfy s.(t.ID=r.CustomerID)/\(s.CustomerID=t.ID))))
         )
   do /{
      delete DepositTab (drop r.(SOMEof CustomersTab(satisfy t.(SOMEof
AccountsTab(satisfy s.(r.CustomerID = t.ID)/\(s.CustomerID = t.ID))))));
      insert DepositTab InvalidTransactionTab I ALL;
      delete WithdrawTab (drop r.(SOMEof CustomersTab(satisfy t.(SOMEof
AccountsTab(satisfy s.(r.CustomerID = t.ID)/\(s.CustomerID = t.ID))))));
      insert WithdrawTab InvalidTransactionTab I ALL

   /} return void` ;

(*----- the verification -----*)
reduce defs FilterInvalidCustomer_def;
L0min_vcg.autoverify MY_TAC ;
L0min_vcg.VCs;
L0min_vcg.conclude();
print("\n \n FILTER INVALID CUSTOMER DONE \n");
```

Gambar 5.7 Kode Lingu untuk Filter Invalid Customer

5.2.5 Modul Filter Invalid Account

Modul ini digunakan untuk menyaring data rekening yang tidak terdata dalam sistem aplikasi perbankan.

5.2.5.1 Parameter

Parameter yang digunakan pada modul ini keseluruhannya bertipe *pass by reference* dari tabel DepositTab, WithdrawTab, AccountsTab, dan InvalidTransactionTab. DepositTab dan WithdrawTab adalah tabel yang berisi data-data transaksi menurut namanya masing-masing yaitu *deposit* atau *withdraw*. AccountsTab adalah tabel yang menyimpan data rekening yang dimiliki oleh nasabah yang ada di dalam aplikasi

perbankan. Seluruh data transaksi dengan rekening yang *invalid* akan dimasukkan ke dalam tabel InvalidTransactionTable.

5.2.5.2 Pre Post Condition

Kondisi sebelum program dijalankan adalah tabel InvalidTransactionTable sebagai tabel yang kosong. Kondisi ini dapat dilihat selengkapnya dalam bahas Lingu v2 pada gambar 5.8.

Kondisi akhir sesudah program dijalankan adalah seluruh data dalam tabel InvalidTransactionTable memenuhi *records* yang ID-nya tidak ada yang sama dengan *records* yang ada pada tabel AccountssTab.

5.2.5.3 Proses

Pada implementasi ini, proses yang berjalan adalah menghapus data dalam tabel DepositTab dan WithdrawTab yang terdaftar dalam tabel AccountsTab. Selanjutnya data yang tersisa adalah data yang *invalid* dan akan dimasukkan dalam tabel InvalidTransactionTable.

5.2.5.4 Verifikasi

Verifikasi pada modul ini tidak berbeda pada modul-modul sebelumnya yaitu pada awal proses verifikasi, semua definisi yang dibuat akan direduksi pada modul ini. Setelah proses reduksi, kemudian akan dilakukan pemanggilan fungsi auto verifikasi yang terdapat pada *library* LinguLib dengan menggunakan taktik MY_TAC yang telah didefinisikan sebelumnya. Hasil verifikasi modul ini dapat dilihat pada Lampiran B.

```
(*===== FILTER INVALID ACCOUNT =====*)
print("\n \n *===== FILTER INVALID ACCOUNT =====*\n");
val FilterInvalidAccount_def = Define
  ` FilterInvalidAccount(
    REF (DepositTab : TransactionTable set),
    REF (WithdrawTab : TransactionTable set),
    REF (AccountsTab : AccountTable set),
    REF (InvalidTransactionTab : TransactionTable set)
  )
   =
   pre  (empty InvalidTransactionTab)
   post (
    ALLof InvalidTransactionTab ( satisfy r.~SOMEof AccountsTab ( satisfy t.r.AccountID = t.ID))
        )
   do /(
      delete DepositTab (drop r.(SOMEof AccountsTab(satisfy t.(r.AccountID = t.ID))));
      insert DepositTab  InvalidTransactionTab I ALL;
      delete WithdrawTab (drop r.(SOMEof AccountsTab(satisfy t.(r.AccountID = t.ID))));
      insert WithdrawTab InvalidTransactionTab I ALL

   /) return void` ;

(*----- the verification -----*)
reduce defs FilterInvalidAccount_def;
LOmin_vcg.autoverify MY_TAC ;
LOmin_vcg.VCs;
LOmin_vcg.conclude();
print("\n \n FILTER INVALID ACCOUNT DONE \n");
```

Gambar 5.8 Kode Lingu untuk Filter Invalid Account

5.2.6 Modul Check PIN

Modul ini digunakan untuk memeriksa PIN yang dimasukan saat modul Withdraw berjalan. Modul ini akan memeriksa nomor PIN dari data yang ada dalam sistem dengan nomor PIN yang dimasukan oleh pengguna saat akan menggunakan aplikasi perbankan ini. Penulisan dalam Lingu dapat dilihat pada gambar 5.9.

5.2.6.1 Parameter

Parameter yang digunakan pada modul ini keseluruhannya bertipe *pass by reference* dari tabel WithdrawTab, AccountsTab, dan InvalidTransactionTab. WithdrawTab adalah tabel yang berisi data-data transaksi *withdraw*. AccountsTab adalah tabel yang menyimpan data rekening yang dimiliki oleh nasabah yang ada di dalam aplikasi perbankan. Seluruh data transaksi dengan nomor PIN yang *invalid* akan dimasukkan ke dalam tabel InvalidTransactionTable.

5.2.6.2 Pre Post Condition

Kondisi sebelum program dijalankan adalah tabel InvalidTransactionTable sebagai tabel yang kosong.

Kondisi akhir sesudah program dijalankan adalah seluruh data dalam tabel InvalidTransactionTable memenuhi *records* yang ID-nya tidak ada yang sama dengan *records* yang ada pada tabel AccountssTab.

5.2.6.3 Proses

Pada implementasi ini, proses yang berjalan adalah menghapus data dalam tabel WithdrawTab yang terdaftar dalam tabel AccountsTab. Selanjutnya data yang tersisa adalah data yang *invalid* dan akan dimasukkan dalam tabel InvalidTransactionTable.

5.2.6.4 Verifikasi

Verifikasi pada modul ini tidak berbeda pada modul-modul sebelumnya yaitu pada awal proses verifikasi, semua definisi yang dibuat akan direduksi pada modul ini. Setelah proses reduksi, kemudian akan dilakukan pemanggilan fungsi auto verifikasi yang terdapat pada *library* LinguLib dengan menggunakan taktik MY_TAC yang telah didefinisikan sebelumnya. Hasil verifikasi modul ini dapat dilihat pada Lampiran B.

```
(*===== CHECK PIN =====*)
print("\n \n *===== CHECK PIN =====*\n");
val CheckPIN_def = Define
  `CheckPIN(
            REF (WithdrawTab  : TransactionTable set),
            REF (AccountsTab : AccountTable set),
            REF (InvalidTransactionTab : TransactionTable set)
           )
         =

          pre  (empty InvalidTransactionTab)
          post (
           ALLof InvalidTransactionTab( satisfy r.~SOMEof AccountsTab (satisfy t. r.NomorPIN = t.NomorPIN))
               )
     do /(

           delete WithdrawTab (drop r.(SOMEof AccountsTab (satisfy t.(r.NomorPIN=t.NomorPIN))));
           insert WithdrawTab InvalidTransactionTab I ALL

     /) return void`;
(*----- the verification -----*)
reduce defs CheckPIN_def;
L0min_vcg.autoverify MY_TAC ;
L0min_vcg.VCs;
L0min_vcg.conclude();
print("\n \n CHECK PIN DONE \n ");
```

Gambar 5.9 Kode Lingu untuk Check PIN

5.3 Transformasi

Setelah kegiatan verifikasi selesai dilakukan, kegiatan yang selanjutnya dilakukan adalah melakukan transformasi bahasa Lingu ke dalam bahasa Java. Tentunya skrip yang akan ditransformasi adalah skrip yang sudah berhasil di-verifikasi. Proses transformasi ini didukung oleh *Attribut Grammar* sehingga mampu untuk mengubah bahasa lingu menjadi bahasa Java [JIM05]. Pada tahap ini penulis tidak bisa menyelesaikan proses transformasi dari skrip Lingu yang telah diverifikasi. Hal ini disebabkan adanya masalah pada perangkat lunak Hugs dan Java yang tidak penulis mengerti. Masalah disebabkan karena kekurangan resource momory pada Java dan juga permasalahan pada perangkat lunak Hugs 98 yang tidak dimengerti.

Fungsi transformasi untuk Lingu ini sendiri masih terus dikembangkan. Kemampuan transformasi bahasa Lingu ke bahasa Java saat ini masih berupa deklarasi tipe data, sedangkan untuk *method* sampai saat ini belum diaplikasikan. Kemudian fungsi transformasi ini juga harus diverifikasi agar bisa berfungsi sesuai dengan keinginan.

BAB VI Studi Banding Lingu dengan Metode-B

Dalam bab ini akan dijelaskan perbandingan proses spesifikasi dan verifikasi yang dilakukan dengan Lingu terhadap proses yang dijalankan dengan Metode-B. Pekerjaan dengan menggunakan Metode-B sendiri sudah dilakukan oleh Theresia [BUD06]. Perbandingan yang dilakukan bertujuan untuk menjelaskan tentang kelebihan dan kekurangan masing-masing bahasa spesifikasi. Terkait dengan pengembangan Lingu dan LinguSQL, diharapkan dari perbandingan ini Lingu dan LinguSQL dapat dikembangkan untuk menjadi lebih baik.

6.1 Pelaksanaan Studi Kasus Aplikasi Perbankan

Studi kasus aplikasi perbankan seperti sudah dijelaskan pada Bab IV merupakan studi kasus kritis yang sangat erat dengan transaksi basis data. Dalam tugas akhir ini akan ditelaah langkah demi langkah dari tiap-tiap alat verifikasi yaitu LinguSQL dan Atelier-B dalam pelaksanaannya terhadap studi kasus aplikasi perbankan.

Secara umum menurut Theresia [BUD06], langkah-langkah implementasi studi kasus aplikasi perbankan dengan menggunakan Metode-B dan Atelier-B bisa dijelaskan sebagai berikut ini:

1. Membuat suatu proyek baru dengan menggunakan Atelier-B, kemudian semua mesin-mesin yang digunakan dimasukkan ke dalam proyek tersebut. Dalam Atelier-B mesin-mesin yang digunakan dalam suatu poryek disebut komponen proyek. Pengertian dari mesin di sini ialah bagian dari studi kasus yang akan kita implementasikan ke dalam Metode-B.
2. Melakukan pengecekan tipe terhadap semua komponen proyek.
3. Membangkitkan *proof obligation* terhadap masing-masing komponen proyek.
4. Membuktikan *proof obligation* dengan menggunakan *semi-automatic prover* yang disediakan oleh Atelier-B.

5. Melakukan pembuktian secara interaktif terhadap *proof obligation* yang tidak dapat dibuktikan secara otomatis.
6. Melakukan pengecekan B0. B0 adalah bagian dari bahasa B yang mendeskripsikan operasi-operasi dan data dari suatu mesin implementasi [CLE03].
7. Menerjemahkan komponen-komponen tersebut ke bahasa pemograman C.

Kemudian langkah-langkah yang ditempuh jika menggunakan Lingu dan LinguSQL yang dilakukan pada tugas akhir ini dapat dijelaskan sebagai berikut:

1. Menentukan modul kritis dari studi kasus yang akan diperiksa.
2. Mendefinisikan basis data yang diperlukan oleh modul kritis.
3. Membuat skrip Lingu v2 sesuai dengan spesifikasi dan memverifikasikannya.
4. Membuat taktik dalam membantu HOL sebagai *theorem prover* pada LinguSQL
5. Melakukan verifikasi modul kritis dengan pembuktian secara automatis.
6. Melakukan pembuktian secara interaktif untuk membantu HOL dalam membuktikan suatu skrip Lingu.
7. Melakukan validasi terhadap skrip lingu yang berhasil diverifikasi.
8. Melakukan transformasi skrip Lingu yang berhasil diverifikasi ke bahasa Java.

6.2 Dukungan Terhadap Basis Data

Seperti diinformasikan dalam Bab II tentang proses spesifikasi dan verifikasi, Lingu diciptakan khusus untuk mengatasi aplikasi yang berhubungan dengan transaksi basis data. Dalam sub-bab ini perbandingan Lingu dan Metode-B akan difokuskan terhadap dukungan tiap-tiap alat verifikasi kepada aplikasi transaksi basis data.

Dalam Lingu, pembangunan aplikasi basis data sangat didukung. Hal ini dapat dilihat pada langkah-langkah dalam spesifikasi dan verifikasi modul kritis dimana terdapat langkah untuk menentukan basis data suatu modul kritis. Dalam syntax Lingu yang dijelaskan pada Bab II, dapat dilihat pula bahwa Lingu mendukung operasi-operasi yang berkaitan tentang basis data. Operasi-operasi yang dilakukan mencakup kepada

operasi pada *records* sebagai operasi dalam basis data. Syntax Lingu pada operasi-operasi yang berhubungan dengan basis data dapat dilihat sebagi berikut.

```
TabelExpr )
                    | TableExpr union TableExpr
                    | SimpleExpr IN Expr
                    | ALLof Expr ( satisfy Var . Expr )
                    | SOMEof Expr ( satisfy Var . Expr )

Instr          → | ins SimpleExpr Expr
                 | insert Expr Expr (map Var . SimpleExpr)(only Var . Expr)
                 | del SimpleExpr Expr
                 | delete Expr (drop Var , Expr)
                 | update Expr (map Var . SimpleExpr)(only Var . Expr)
```

Gambar 6.1 Operasi yang Mendukung Basis Data pada Lingu v2

Sementara itu dalam Metode-B, proses pembuatan tabel-tabel data dan operasi-operasi terhadapnya harus dibuat dari awal. Hal ini dikarenakan dukungan Metode-B untuk *records* masih terbatas. Menurut Siti Aminah [AMI05], pengembangan aplikasi basis data tidak secara khusus didukung oleh Metode-B. Tahap awal yang harus dilakukan adalah membuat bentuk abstraksi tipe *records*. Tahap awal ini dilakukan karena dalam aplikasi basis data sangat dibutuhkan tabel yang merupakan kumpulan dari banyak *records*. Contoh dari abstraksi tipe *records* pada Metode-B adalah sebagai berikut:

```
MACHINE Database
SETS
   REC; SETA; SETB
CONSTANTS
  afield, bfield
PROPERTIES
  afield ∈ REC→SETA  ∧ bfield ∈ REC→SETB ∧
  ∀ (aa, bb•((aa ∈ SETA ∧ bb ∈ SETB)⇒
  ∃ rr. (rr ∈ REC ∧ afield(rr) = aa ∧ bfield(rr) = bb))
VARIABLES
  db
INVARIANT
  db ∈ P(REC)
INITIALISATION
   db:=∅
OPERATIONS
 Add_Database ≙
  ANY af,bf,rn WHERE af ∈ SETA ∧ bf ∈ SETB ∧ rn ∈ REC ∧
                     afield(rn)=af ∧ bfield(rn)= bf
   THEN
     db:= db ∪ {rn}
   END
END
```

Gambar 6.2 Contoh Mesin Abstrak Basis Data pada Metode-B [REZ03]

6.3 Proses Spesifikasi dan Verifikasi

Secara umum yang dilakukan dalam LinguSQL adalah proses pembuatan spesifikasi dan verifikasi. Sementara itu, Atelier-B melakukan proses *refinement* pada perangkat lunak yang akan dibuat. Kedua macam proses ini tidak jauh berbeda, tetapi tahapan pelaksanaannya yang berbeda.

Pada linguSQL, proses pembuatan spesifikasi dan verifikasi dilaksanakan setelah *prototype* dari suatu perangkat lunak diciptakan. Dari *prototype* itulah ditemukan modul-modul yang kritis yang kemudian akan diperiksa. Proses pembuatan spesifikasi dan verifikasi yang akan menjamin perangkat lunak tersebut.

Berbeda denga LinguSQl, proses *refinement* pada Atelier-B dilaksanakan sebelum suatu perangkat lunak diimplementasikan. Proses ini akan mendefinisikan komponen-komponen proyek yang akan ada pada perangkat lunak lalu mentransformasikannya ke dalam bahasa konkret yaitu C. Pengembangan perangkat lunak ini diteruskan dengan menggunakan bahasa konkret tersebut.