UNIVERSITAS INDONESIA

PENGEMBANGAN LANJUT *TABLING* PADA *CONTEXTUAL ABDUCTION* DENGAN *ANSWER SUBSUMPTION*

SKRIPSI

SYUKRI MULLIA ADIL PERKASA
1306381793

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JULI 2017

UNIVERSITAS INDONESIA

PENGEMBANGAN LANJUT *TABLING* PADA *CONTEXTUAL ABDUCTION* DENGAN *ANSWER SUBSUMPTION*

SKRIPSI
Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana Ilmu Komputer

SYUKRI MULLIA ADIL PERKASA
1306381793

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI ILMU KOMPUTER
DEPOK
JULI 2017

HALAMAN PERNYATAAN ORISINALITAS

Skripsi ini adalah hasil karya saya sendiri,
dan semua sumber baik yang dikutip maupun dirujuk
telah saya nyatakan dengan benar.

Nama	:	**Syukri Mullia Adil Perkasa**
NPM	:	**1306381793**
Tanda Tangan	:	
Tanggal	:	**21 Juni 2017**

HALAMAN PENGESAHAN

Skripsi ini diajukan oleh	:	
Nama	:	Syukri Mullia Adil Perkasa
NPM	:	1306381793
Program Studi	:	Ilmu Komputer
Judul Skripsi	:	Pengembangan Lanjut *Tabling* pada *Contextual Abduction* dengan *Answer Subsumption*

Telah berhasil dipertahankan di hadapan Dewan Penguji dan diterima sebagai bagian persyaratan yang diperlukan untuk memperoleh gelar Sarjana Ilmu Komputer pada Program Studi Ilmu Komputer, Fakultas Ilmu Komputer, Universitas Indonesia.

DEWAN PENGUJI

Pembimbing	:	Ari Saptawijaya S.Kom., M.Sc., Ph.D.	(	)
Penguji	:	Penguji 1	(	)
Penguji	:	Penguji 2	(	)

Ditetapkan di	:	Depok
Tanggal	:	5 Juli 2017

KATA PENGANTAR

Alhamdulillahirabbil'alamin, segala puji dan syukur kehadirat Tuhan Yang Maha Esa, Allah Subhana Huwataala, karena hanya dengan hidayah dan rahmat-Nya, penulis dapat menyelesaikan pembuatan skripsi ini.

Allahumma sholli 'alaa sayyidina Muhammad, Sholawat serta salam tak henti-hentinya dipanjatkan kepada Rasulullah SAW, atas peranannya di muka bumi dalam memberikan tuntunan kepada seluruh umat manusia, dan sebagai inspirasi atas seluruh manusia sebagai manusia dengan akhlak terbaik.

Penulisan skripsi ini ditujukan untuk memenuhi salah satu syarat untuk menyelesaikan pendidikan pada Program Sarjana Ilmu Komputer, Universitas Indonesia. Saya sadar bahwa dalam perjalanan menempuh kegiatan penerimaan dan adaptasi, belajar-mengajar, hingga penulisan skripsi ini, penulis tidak sendirian. Penulis ingin berterima kasih kepada pihak-pihak berikut :

Depok, 21 Juni 2017

Syukri Mullia Adil Perkasa

HALAMAN PERNYATAAN PERSETUJUAN PUBLIKASI TUGAS AKHIR UNTUK KEPENTINGAN AKADEMIS

Sebagai sivitas akademik Universitas Indonesia, saya yang bertanda tangan di bawah ini:

Nama	:	Syukri Mullia Adil Perkasa
NPM	:	1306381793
Program Studi	:	Ilmu Komputer
Fakultas	:	Ilmu Komputer
Jenis Karya	:	Skripsi

demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Indonesia **Hak Bebas Royalti Noneksklusif (Non-exclusive Royalty Free Right)** atas karya ilmiah saya yang berjudul:

Pengembangan Lanjut *Tabling* pada *Contextual Abduction* dengan *Answer Subsumption*

beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Noneksklusif ini Universitas Indonesia berhak menyimpan, mengalihmedia/formatkan, mengelola dalam bentuk pangkalan data (*database*), merawat, dan memublikasikan tugas akhir saya selama tetap mencantumkan nama saya sebagai penulis/pencipta dan sebagai pemilik Hak Cipta.

Demikian pernyatan ini saya buat dengan sebenarnya.

Dibuat di : Depok
Pada tanggal : 21 Juni 2017

Yang menyatakan

(Syukri Mullia Adil Perkasa)

ABSTRAK

Nama	:	Syukri Mullia Adil Perkasa
Program Studi	:	Ilmu Komputer
Judul	:	Pengembangan Lanjut *Tabling* pada *Contextual Abduction* dengan *Answer Subsumption*

Abstrak INA

Kata Kunci:
atu, dua, *tiga*

ABSTRACT

Name	:	Syukri Mullia Adil Perkasa
Program	:	Computer Science
Title	:	Advanced Development of Tabling in Contextual Abduction with Answer Subsumption

Abstract in Eng

Keywords:
one,two,three

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR KODE

BAB 1
PENDAHULUAN

1.1 Latar Belakang

Menurut Buyya terdapat 3 buah contoh untuk membuat enumerate pada latex (Buyya, 1999):

1. Makan
2. Minum

Menurut Mozdzynski (2012), pemodelan yang sama apabila dijalankan dengan komputer *Dual Core* maka akan membutuhkan waktu 1 tahun dengan asumsi memori yang dibutuhkan cukup (Mozdzynski, 2012).

1.2 Perumusan Masalah

Pada bagian ini akan dijelaskan mengenai definisi permasalahan yang dihadapi dan ingin diselesaikan serta asumsi dan batasan yang digunakan dalam menyelesaikannya.

1.3 Tujuan dan Manfaat Penelitian

Dibawah ini adalah contoh itemize :

- Terimplementasinya .
- Menyelesaikan masalah .

1.4 Tahapan Penelitian

@todo
Tuliskan tujuan penelitian.

1.5 Ruang Lingkup Penelitian

1.6 Sistematika Penulisan

Sistematika penulisan laporan adalah sebagai berikut:

- Bab 1 PENDAHULUAN
- Bab 2 LANDASAN TEORI
- Bab 3 IMPLEMENTASI
- Bab 4 EVALUASI DAN ANALISIS
- Bab 5 PENUTUP

@todo
Tambahkan penjelasan singkat mengenai isi masing-masing bab.

BAB 2
LANDASAN TEORI

Bab ini berisi dasar-dasar pemahaman yang diperlukan dalam membuat program logika.

2.1 Pendahuluan

Diberikan himpunan alfabet $\mathcal{A}$ dari bahasa $\mathcal{L}$, akan didefinisikan himpunan berhingga yang saling *disjoint* yaitu himpunan konstanta, himpunan simbol fungsi, dan himpunan simbol predikat. Selain itu, pada alfabet juga mengandung himpunan simbol variabel. Simbol *underscore* (_) dikhususkan untuk menyatakan sebuah *anonymous* variabel. *Term* dari $\mathcal{A}$ didefinisikan secara rekursif sebagai salah satu dari: variabel, konstanta, atau ekspresi dengan bentuk $f(t_1,...,t_n)$ dengan f adalah simbol fungsi dari $\mathcal{A}$ dan t_i adalah term. *Atom* dari $\mathcal{A}$ adalah ekspresi dengan bentuk $p(t_1,...,t_n)$ dengan p adalah simbol predikat dari $\mathcal{A}$ dan t_i adalah term. Selanjutnya, notasi p/n digunakan untuk menyatakan predikat p memiliki arity n. *Literal* adalah sebuah atom a atau negasinya *not a*. Literal *not a* (seperti pada bentuk kedua) disebut *default* literal. Sebuah term (atom maupun literal) disebut *ground* jika term tersebut tidak memiliki variabel. Himpunan seluruh ground term dari $\mathcal{A}$ disebut Herbrand Universe dari $\mathcal{A}$.

2.2 Program Logika

Program logika adalah himpunan berhingga dari *rule* dengan bentuk:

$$H \leftarrow L_1, ..., L_n$$

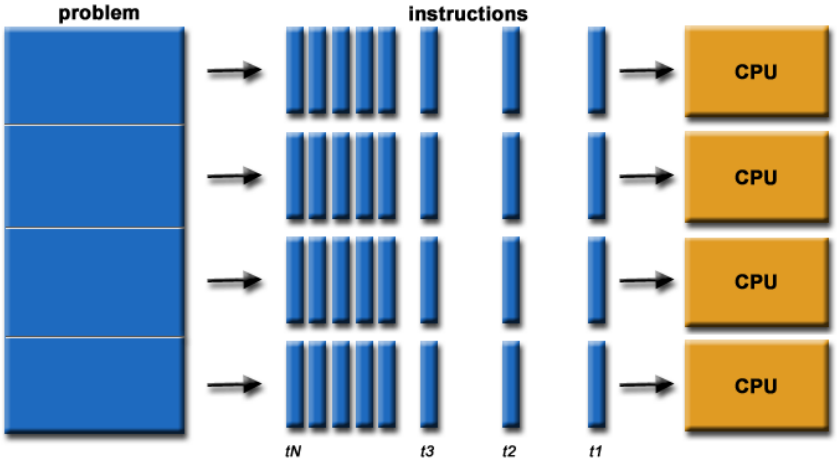
dengan H adalah sebuah *atom*, $m \geq 0$, dan L_i adalah *literal*. H dan L_i berturut-turut disebut sebagai *head* dan *body* dari sebuah rule.

Operator koma pada sebuah rule dibaca sebagai konjungsi. Sebuah program logika disebut *definit* jika pada program logika tersebut tidak mengandung default literal. Rule yang tidak memiliki body ditulis sebagai H saja, alih-alih menuliskannya sebagai $H \leftarrow$. Rule dengan bentuk seperti ini disebut sebagai sebuah *fakta*.

2.2.1 Pengertian X

Setiap gambar dapat diberikan caption dan diberikan label. Label dapat digunakan untuk menunjuk gambar tertentu. Jika posisi gambar berubah, maka nomor gambar juga akan diubah secara otomatis. Begitu juga dengan seluruh referensi yang menunjuk pada gambar tersebut.

Contoh sederhana adalah Gambar 2.1. Silahkan lihat code LaTeX dengan nama bab2-landasan-teori.tex untuk melihat kode lengkapnya. Harap diingat bahwa caption untuk gambar selalu terletak dibawah gambar. Dibawah adda figure, jangn lupa dimention dengan 2.1.



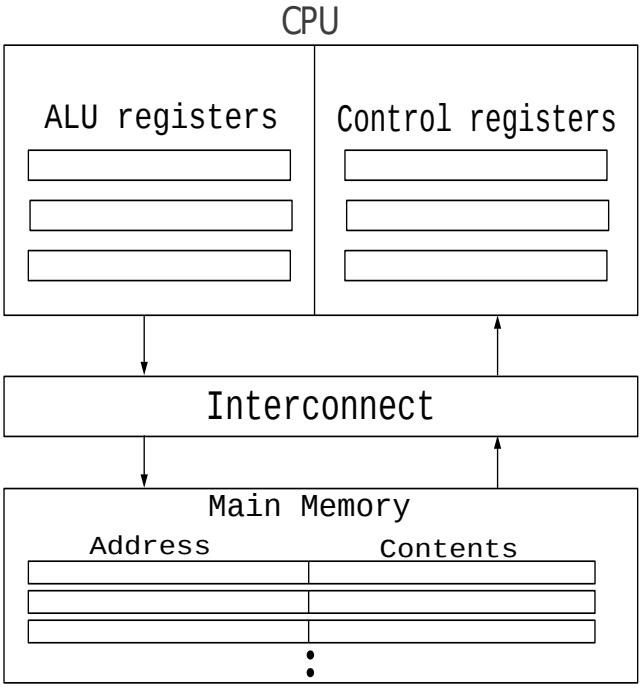
Gambar 2.1: Contoh masalah yang dikerjakan secara paralel

Sumber gambar: (Oxford Dictionaries, 2010)

2.2.2 Klasifikasi X

Figure dalam enum dan dua sitasi sekaligus (Buyya, 1999; Jones et al., 2002) :

1. ***Bold Italic***
   Penjelasan..... Untuk gambarannya dapat dilihat di Gambar 2.2.



Gambar 2.2: Arsitektur klasik *von Neumann*

Sumber gambar terinspirasi dari: (Pressman, 2010)

2. ***Sesuatu banget***
   Penjelasan.....

2.3 *Section in Eng*

Hal pertama yang mungkin ditanyakan adalah bagaimana membuat huruf tercetak tebal, miring, atau memiliki garis bawah. Pada Texmaker, Anda bisa melakukan hal ini seperti halnya saat mengubah dokumen dengan LO Writer. Namun jika tetap masih tertarik dengan cara lain, ini dia:

- **Bold**
  Gunakan perintah \textbf{} atau \bo{}.
- *Italic*
  Gunakan perintah \textit{} atau \f{}.
- Underline
  Gunakan perintah \underline{}.

- $\overline{Overline}$
  Gunakan perintah \overline.
- ^{superscript}
  Gunakan perintah \{}.
- $_{subscript}$
  Gunakan perintah _{}.

Perintah \f dan \bo hanya dapat digunakan jika package uithesis digunakan.

2.3.1 Pengertian *Section in Eng*

2.3.2 Next Subsection *Section in Eng*

2.4 *Keatas lagi*

Contoh cite yang ga ada **?**. Cite author Neal,cite tahun 2004, cite mention Guarddin (2010), dan cite di akhir kalimat (Mell dan Grance, 2009).

2.4.1 *Masuk lagi*

Footnote example nih : MPICH [1], LAM/MPI [2], dan OpenMPI [3] (McGuire, 2010). MPI-3 sedang dalam tahap perencanaan [4]. Fungsi-fungsi tersebut berada di tabel 2.1. (Contoh tabel).

Tabel 2.1: Fungsi fundamental MPI

No.	Nama Fungsi	Penjelasan
1	MPI_Init	Memulai kode MPI
2	MPI_Finalize	Mengakhiri kode MPI
3	MPI_Comm_size	Menentukan jumlah proses
4	MPI_Comm_rank	Menentukan label proses
5	MPI_Send	Mengirim pesan
6	MPI_Recv	Menerima pesan

Sumber tabel: taro sitasi disini, if i were u

[1] http://www.mpich.org/
[2] http://www.lam-mpi.org/
[3] www.open-mpi.org
[4] http://meetings.mpi-forum.org/MPI_3.0_main_page.php

BAB 3
IMPLEMENTASI

Pada bab ini penulis akan menjelaskan bagaimana implementasi *tabling* pada *contextual abduction* dengan memanfaatkan fitur *answer subsumption* yang dibuat menggunakan salah satu distribusi bahasa pemrograman Prolog, yaitu **XSB**. Pada bagian-bagian selanjutnya, program implementasi ini disebut sebagai TABDUAL.

3.1 Terminologi

Pada bagian ini penulis menjelaskan beberapa ketentuan dan istilah yang digunakan pada bagian-bagian berikutnya, yaitu:

- Pada prolog, variabel diawali dengan huruf kapital sedangkan term dan predikat diawali dengan huruf non-kapital.
- *Consult* berarti melakukan kompilasi program Prolog dan memuat hasil kompilasi program tersebut ke dalam *database* XSB sehingga program tersebut menjadi *knowledge base*.
- *Database* berarti kumpulan predikat-predikat yang disimpan pada *environment* XSB dan dijadikan sebagai *knowledge base* selama eksekusi. Predikat-predikat yang ada pada *database* dapat berasal dari program yang di-*consult* (membentuk *database* statis) ataupun ditambahkan selama eksekusi suatu program (membentuk *database* dinamis). Selama eksekusi, predikat yang sudah ditambahkan ke dalam *database* dinamis dapat dimanipulasi sesuai kebutuhan program.
- *Dual transformation by need* mengacu pada proses transformasi dual *by need* yang sudah dijelaskan pada bagian ???.

3.2 Spesifikasi TABDUAL

3.2.1 Tahapan TABDUAL

Secara garis besar, TABDUAL terbagi menjadi 2 fase:

1. **Transformasi**. Pada fase ini, TABDUAL akan melakukan transformasi program *input P* menjadi program *output* P' yang dapat mengaplikasikan *contextual abduction*. Transformasi dilakukan sesuai dengan aturan-aturan transformasi yang sudah dijelaskan pada bagian 2.

2. **Abduction**. *Contextual abduction* dapat dilakukan setelah program *input P* berhasil di-*transform* menjadi program *output* P'. Praktis, P' harus di-*consult* terlebih dahulu sebelum kita dapat memberikan *query* dan melakukan *contextual abduction*.

Penjelasan lebih detil mengenai setiap fase akan penulis jabarkan pada bagian-bagian berikutnya.

3.2.2 Berkas Implementasi TABDUAL

Agar dapat digunakan secara modular, implementasi TABDUAL dipecah ke dalam empat buah berkas yang berbeda. Keeempat berkas tersebut yaitu:

- *tabdual.p*. Berkas ini berisi implementasi utama dari TABDUAL, baik implementasi untuk fase transformasi maupun implementsai untuk fase *abduction*. Berkas ini adalah berkas yang harus di-*consult* untuk dapat menggunakan TABDUAL. Berkas-berkas lain yang diperlukan selama menggunakan TABDUAL akan di-*consult* melalui berkas ini.

- *system.p*. Berkas ini berisi predikat-predikat bantuan dan predikat-predikat yang didefinisikan secara khusus yang akan digunakan oleh TABDUAL ketika melakukan *tabling*, *contextual abduction*, dan *answer subsumption*.

- *read_clause.p* Berkas ini berisi predikat-predikat yang dikhususkan untuk membaca program *input* agar dapat diproses dan ditransformasikan menggunakan TABDUAL.

- *write_clause.p*. Berkas ini berisi predikat-predikat yang dikhususkan untuk menulis transformasi dari program *input* yang dihasilkan menggunakan TABDUAL ke program *output*.

3.2.3 Program *Input* TABDUAL

Program *input* yang ingin di-*transform* menggunakan TABDUAL harus memenuhi kriteria-kriteria berikut:

- *Rule* ditulis dalam bentuk $H \leftarrow B_1, \ldots, B_n.$, dengan operator $\leftarrow$ ditulis sebagai "<-" (tanda lebih kecil dari dari lalu *dash*).
- Fakta ditulis dalam bentuk $H.$ saja tanpa operator $\leftarrow$.
- *Abducible* pada program ditulis sebagai fakta menggunakan predikat $abds/1$. Argumen dari predikat ini yaitu himpunan *abducible* yang ada pada program beserta dengan *arity*-nya, direpresentasikan sebagai sebuah *list*.
- Predikat-predikat yang hanya berupa fakta dan *rule-rule* yang tidak ingin di-*transform* ditulis terpisah di bagian paling atas program *input*, di antara predikat *beginProlog* dan *endProlog*.
- Setiap *rule* dan fakta yang ditulis diakhiri dengan tanda titik (".").

Agar lebih jelas, berikut ini merupakan contoh program yang diterima sebagai program *input* untuk TABDUAL.

```
1  beginProlog.
2  q(1).
3  q(2).
4  endProlog.
5
6  abds([a/1,b/1]).
7
8  r(X) <- a(X).
9  s(X) <- b(X).
10
11 <- q(X), r(X), s(X).
```

Kode 3.1: Contoh program *input* yang diterima TABDUAL

Dan berikut ini merupakan contoh yang tidak diterima.

```
1  abds(a/1,b/1).          % argumen tidak berupa list
2
3  r(X) <- a(X).
4  s(X) <- b(X)            % rule tidak diakhiri dengan "."
5
6  beginProlog.            % diletakkan di bawah
7  q(1).
8  q(2).
9  endProlog
```

```
10
11 :- q(X), r(X), s(X).    % rule tidak menggunakan <-
```

Kode 3.2: Contoh program *input* yang tidak diterima TABDUAL

3.3 Pra Transformasi

Bagian ini menjelaskan bagian implementasi TABDUAL yang berkaitan dengan sebelum fase transformasi.

3.3.1 *Directive*

Pada Prolog, *directive* merupakan anotasi dan predikat pada program yang akan dieksekusi langsung oleh *compiler* ketika program tersebut di-*consult*. Berbeda dengan predikat biasa pada program, *directive* tidak akan disimpan sebagai *knowledge base* di *database*, melainkan langsung dieksekusi. Pada TABDUAL, *directive-directive* yang ada dapat dikelompokkan menjadi beberapa kelompok.

3.3.1.1 *Import*

Untuk mempermudah pengguna, XSB Prolog sudah menyediakan predikat-predikat *built-in* yang dapat digunakan. Predikat *built-in* tersebut dikelompokkan ke dalam modul-modul yang berbeda sesuai dengan kategori penggunaannya. *Directive* berikut ini akan meng-*import* beberapa predikat *built-in* yang diperlukan oleh TABDUAL.

```
:- import append/3, member/2, length/2 from basics.
:- import concat_atom/2 from string.
:- import trie_create/2, trie_drop/1 from intern.
```

Kode 3.3: Deklarasi *directive*: *import* modul yg diperlukan

Predikat $append/3$, $member/2$, dan $length/2$ yang sudah disediakan dalam modul *basics* berturut-turut digunakan untuk menggabungkan dua buah *list*, mengecek presensi suatu elemen pada sebuah *list*, dan menentukan panjang sebuah *list*. Predikat $concat\_atom/2$ yang sudah disediakan dalam modul *string* digunakan untuk melakukan konkatenasi atom-atom untuk membentuk suatu atom baru. Predikat $trie\_create/2$ dan $trie\_drop/1$ yang sudah disediakan dalam modul *intern* masing-

masing digunakan untuk membuat dan menghapus *trie* yang digunakan oleh *dual transformation by need*.

3.3.1.2 Deklarasi Operator

Pada Prolog, pengguna dapat mendefinisikan operator logika baru menggunakan predikat *built-in* $op/3$. Predikat $op/3$ memiliki tiga buah argumen. Argumen pertama menyatakan presedensi, argumen kedua menyatakan tipe, dan argumen ketiga menyatakan nama dari operator tersebut. Presedensi dari operator dinyatakan sebagai sebuah bilangan bulat antara 1 sampai 1200 (1 adalah presedensi dari sebuah term), semakin kecil nilainya semakin kuat presedensinya. Tipe operator menentukan apakah operator tersebut merupakan operator *prefix*, *infix*, atau *suffix*, sekaligus menyatakan sifat assosiatif yang dimilikinya, apakah assosiatif kanan, assosiatif kiri, atau tidak assosiatif. Tipe operator yang dapat digunakan untuk operator *prefix* yaitu *fx* dan *fy*, tipe operator yang dapat digunakan untuk operator *infix* yaitu *yfx*, *xfy*, dan *xfx*, dan tipe operator yang dapat digunakan untuk operator *suffix* yaitu *xf* dan *yf*. Simbol *f* pada tipe operator merepresentasikan posisi operator sedangkan simbol *x* dan *y* merepresentasikan argumen-argumennya. Simbol *x* menyatakan bahwa argumen tersebut harus memiliki presedensi kurang dari presedensi operator *f*, sedangkan simbol *y* menyatakan bahwa argumen tersebut harus memiliki presedensi kurang dari atau sama dengan presedensi operator *f*. Dengan kata lain, simbol *y* menyatakan bahwa operator tersebut bersifat assosiatif, sedangkan simbol *x* menyatakan bahwa operator tersebut bersifat tidak assosiatif.

TABDUAL mendeklarasikan dua buah operator baru yaitu *not* dan $\leftarrow$ seperti di bawah ini.

```
:- op(950, fy, not).
:- op(1110, fy, '<-').
:- op(1110, xfy, '<-').
```

Kode 3.4: Deklarasi *directive*: definisi operator baru

Operator *not* digunakan untuk menyatakan negasi dari sebuah predikat sehingga tipe operatornya adalah *fy*. Operator $\leftarrow$ digunakan untuk menyatakan implikasi yang dibalik untuk digunakan ketika mendefinisikan sebuah *rule-rule* pada program input. Operator $\leftarrow$ memiliki dua tipe operator untuk dua penggunaan yang berbeda. Tipe operator $\leftarrow$ yang pertama yaitu *fy* digunakan untuk membentuk *integrity constraint*, sedangkan tipe operator $\leftarrow$ yang kedua yaitu *xfy* digunakan untuk

membentuk *rule*.

3.3.1.3 Deklarasi Predikat Dinamis

Predikat dinamis adalah predikat yang definisi atau nilainya dapat berubah-ubah. Predikat dinamis digunakan untuk memanipulasi *database* dinamis XSB selama eksekusi. TABDUAL mendeklarasikan empat buah predikat dinamis yang digunakan selama fase transformasi dan fase *abduction*. *Directive* berikut ini mendeklarasikan keempat predikat dinamis yang digunakan.

```
:- dynamic has_rules/1, rule/2, rule/3, abds/1.
```

Kode 3.5: Deklarasi *directive*: definisi operator baru

Predikat *has_rules*/1 digunakan untuk menyimpan informasi mengenai predikat yang memiliki *rule*, dengan kata lain, predikat-predikat yang menjadi *head* pada program. Argumen dari *has_rules*/1 yaitu *R* yang menyatakan *head* yang ada pada program. *Head-head* ini disimpan pada *database* menggunakan predikat dinamis *has_rules*/1 secara *distinct*. Predikat *rule*/2 dan *rule*/3 digunakan untuk menyimpan informasi mengenai sebuah *rule* yang ada pada program. Dua buah argumen pertama dari *rule*/2 dan *rule*/3 yaitu *H* dan *B*, berturut-turut menyatakan *head* dan *body* dari *rule* tersebut. Argumen ketiga dari *rule*/3 yaitu sebuah bilangan *N* yang menyatakan bahwa $H \leftarrow B$ adalah *rule* ke-*N* mengenai *H*. Penjelasan mengapa diperlukan dua buah predikat dinamis untuk menyimpan *rule-rule* pada program input akan dijelaskan pada bagian 3.4.7. Predikat *abds*/1 digunakan untuk menyimpan informasi mengenai himpunan *abducible* yang direpresentasikan sebagai sebuah *list*. Predikat *abds*/1 memiliki satu buah argumen yaitu *list abducible* itu sendiri.

3.3.1.4 *Directive* Lainnya

Karena dibutuhkan untuk fase transformasi dan *contextual abduction*, beberapa predikat berikut ini perlu dijadikan sebagai *directive* agar dieksekusi langsung ketika TABDUAL di-*consult*.

```
:- consult_files, retractall(mode/1), assert(mode(normal)).
```

Kode 3.6: Deklarasi *directive*: lainnya

Predikat *consult_files*/0 digunakan untuk men-*consult* berkas-berkas implementasi TABDUAL lainnya, yaitu berkas *system.p*, *read_clause.p*, dan *write_clause.p*. Predikat *retractall*(*mode*/1) dan *assert*(*mode*(*normal*)) digunakan untuk me-inisialisasi ulang mode yang digunakan untuk transformasi. Penjelasan mengenai mode transformasi akan dijelaskan lebih jauh pada bagian 3.4.8.

3.3.2 Predikat *wrapper transform*/1

Fase transformasi yang dilakukan TABDUAL di-*wrap* ke dalam satu buah predikat yaitu *transform*/1. Predikat *transform*/1 memiliki sebuah argumen yang menyatakan nama program *input* yang ingin di-*transform*. Berikut definisi dari predikat *transform*/1.

```
transform(Filename) :-
    see_input_file(Filename),
    tell_output_file(Filename),
    pre_transform,
    transform,
    seen,
    told.
```

Kode 3.7: Definisi predikat *transform*/1

Terdapat enam buah *goal* yang harus dieksekusi pada predikat *transform*/1. Predikat *see_input_file*/1 menentukan *input stream* untuk fase transformasi TABDUAL yaitu program *input* yang ingin di-*transform*. Predikat *tell_output_file*/1 menentukan *output stream* untuk fase transformasi TABDUAL, yaitu program *output* yang akan dihasilkan. Program *input* harus memiliki ekstensi *.ab* dan program *output* yang dihasilkan akan memiliki nama yang sama namun dengan ekstensi *.p*. Predikat *pre_transform*/0 melakukan beberapa hal yang harus dilakukan sebelum memulai transformasi (akan dijelaskan pada bagian 3.4.3) dan predikat *transform*/0 adalah predikat yang akan melakukan transformasi (akan dijelaskan pada bagian 3.4). Predikat *seen*/0 dan *told*/0 berturut-turut mengembalikan input *stream* dan output *stream* menjadi seperti semula yaitu *prompt* Prolog.

3.3.3 Predikat *pre_transform*/0

Terdapat beberapa hal perlu dilakukan sebelum memulai fase transformasi. Hal-hal tersebut di-*wrap* ke dalam predikat *pre_transform*/0 yang pada TABDUAL didefinisikan seperti di bawah ini.

```
pre_transform :-
    clear,
    load_rules,
    add_indices.
```

Kode 3.8: Definisi predikat *pre_transform*/0

Terdapat tiga buah *goal* yang harus dieksekusi pada predikat *pre_transform*/0. Predikat *clear*/0 mengosongkan *database* dengan menghapus semua predikat dinamis yang sudah tersimpan. Predikat *load_rules*/0 membaca program input dan menyimpan program yang didapat ke dalam *database* menggunakan predikat dinamis. Predikat *add_indices*/0 menambahkan indeks pada setiap *rule* yang disimpan menggunakan predikat dinamis *rule*/2. Penjelasan lebih lanjut mengenai ketiga predikat ini akan dijelaskan pada bagian-bagian berikutnya.

3.3.4 Predikat *clear*/0

Predikat *clear*/0 digunakan untuk mengosongkan *database* dengan menghapus semua predikat dinamis yang sudah tersimpan, sekaligus menghapus dan membuat ulang *trie* yang digunakan oleh *dual transformation by need*. Pada TABDUAL predikat *clear*/0 didefinisikan sebagai berikut.

```
clear :-
    retractall(has_rules/1),
    retractall(rule/2),
    retractall(rule/3),
    retractall(abds/1),
    trie_drop(dual),
    trie_create(dual).
clear :-
    trie_create(dual).
```

Kode 3.9: Definisi predikat *clear*/0

Empat *goal* pertama pada definisi predikat *clear*/0 menghapus seluruh predikat dinamis yang sudah disimpan menggunakan predikat *built-in retractall*/1. Pada *goal* selanjutnya, predikat *trie_drop*/1 menghapus dan membuat ulang *trie* dengan alias *dual* yang akan digunakan oleh *dual transformation by need*. Jika penghapusan gagal, maka *trie_create*/2 pada definisi *clear*/0 akan dieksekusi untuk membuat

trie yang baru, juga dengan alias *dual*, agar dapat digunakan oleh *dual transformation by need*.

3.3.5 Predikat *load_rules*/0

Predikat *load_rules*/0 membaca program input *rule* demi *rule* dan menyimpan *rule* yang dibaca ke dalam *database* menggunakan predikat dinamis. Berikut ini definisi dari predikat *load_rules*/0 pada TABDUAL yang didefinisikan secara rekursif.

```
1  load_rules :-
2      read(C),
3      (
4      C = end_of_file
5      ->
6      true
7      ;
8      C = beginProlog
9      ->
10     load_just_facts
11     ;
12     load_rule(C),
13     load_rules
14     ).
```

Kode 3.10: Definisi predikat *load_rules*/0

Goal pertama yang dieksekusi pada predikat *load_rules*/0 yaitu predikat *built-in read*/1 yang digunakan untuk membaca satu term pada input *stream* yang diberikan. Argumen dari predikat *read*/1 yaitu term yang berhasil dibaca. Selanjutnya, potongan kode dari baris 3 hingga 14 merupakan *statement* kondisional yang terdiri dari tiga buah kondisi yang saling *mutually exclusive*, atau dengan kata lain, dapat dibaca sebagai kondisional *if–else if–else*. Kondisi pertama merupakan *base case*, yaitu jika term yang dibaca adalah term *built-in end_of_file*/0, maka program output selesai dibaca dan *load_rules*/0 sukses. Kondisi kedua yaitu jika term yang dibaca adalah term *beginProlog*/0, maka predikat *load_just_facts*/0 akan dieksekusi sebagai sebuah *goal*. Penjelasan lebih lanjut mengenai predikat *load_just_facts*/0 akan dijelaskan pada bagian selanjutnya. Kondisi ketiga merupakan *recursive case*, yaitu jika kedua kondisi sebelumnya tidak terpenuhi, maka predikat *load_rule*/1 akan dieksekusi sebagai sebuah *goal*. Predikat *load_rule*/1 menyimpan term yang dibaca menggunakan predikat-predikat dinamis yang sesuai dengan bentuk term tersebut, apakah merupakan *abducible*, rule, atau fakta. Sete-

lah eksekusi predikat *load_rule*/1, terjadi pemanggilan rekursif terhadap predikat *load_rules*/0 yang terus diulang hingga seluruh program input selesai dibaca.

3.3.6 Predikat *load_just_facts*/0

Predikat *load_just_facts*/0 membaca term-term pada program input yang ditulis di antara predikat *beginProlog* dan *endProlog* kemudian langsung melakukan transformasi terhadap term-term tersebut. Pada TABDUAL, predikat *load_just_facts*/0 didefinisikan secara rekursif seperti berikut.

```
1  load_just_facts :-
2      read(C),
3      (
4      C = endProlog
5      ->
6      transform_just_fact,
7      load_rules
8      ;
9      load_rule(C),
10     load_just_facts
11     ).
```

Kode 3.11: Definisi predikat *load_just_facts*/0

Sama seperti predikat *load_rules*/0, *goal* pertama yang dieksekusi pada predikat *load_just_facts*/0 yaitu predikat *built-in read*/1 yang digunakan untuk membaca satu term pada input *stream* yang diberikan. Selanjutnya, potongan kode dari baris 3 hingga 14 merupakan *statement* kondisional yang terdiri dari dua buah kondisi yang saling *mutually exclusive*, atau dengan kata lain, dapat dibaca sebagai kondisional *if–else*. Kondisi pertama merupakan *base case*, yaitu ketika term yang dibaca adalah term *endProlog*/0. Artinya, seluruh term yang terdapat di antara predikat *beginProlog*/0 dan *endProlog*/0 sudah dibaca dan disimpan ke dalam *database* sehingga dapat digunakan oleh predikat *transform_just_facts*/0 untuk ditransformasikan sesuai dengan aturan transformasi pada bagian ???. Penjelasan lebih lanjut mengenai predikat *transform_just_facts*/0 akan dijelaskan pada bagian 3.4.4. Selanjutnya, kondisi kedua merupakan *recursive case*. Sama seperti pada predikat *load_rules*/0, predikat *load_rule*/1 akan dieksekusi sebagai sebuah *goal*. Setelah eksekusi predikat *load_rule*/1, terjadi pemanggilan rekursif terhadap predikat *load_just_facts*/0 yang terus diulang hingga bertemu dengan term *endProlog*/0.

3.3.7 Predikat *add_indices*/0

Pada bagian sebelumnya telah dijelaskan bahwa diperlukan dua buah predikat dinamis untuk menyimpan *rule-rule* pada program input, yaitu predikat dinamis *rule*/2 dan *rule*/3. Predikat dinamis *rule*/3 merupakan ekstensi dari predikat dinamis *rule*/2 dengan penambahan satu buah argumen yang menyatakan urutan definisi mengenai *rule* tersebut. Informasi mengenai urutan ini diperlukan untuk mengimplementasikan *dual transformation by need*. Predikat *add_indices*/0 memanfaatkan *rule*/2 yang sudah disimpan pada *database* untuk membentuk *rule*/3 yang sesuai. Berikut ini definisi dari predikat *add_indices*/0 pada TABDUAL.

```
add_indices :-
    retract(has_rules(H)),
    find_rules(H, R),
    add_indices_to_rule(R),
    add_indices,
    assert(has_rules(H)).
```

Kode 3.12: Definisi predikat *add_indices*/0

Terdapat lima buah *goal* yang harus dieksekusi pada predikat *add_indices*/0. Predikat *retract*(*has_rules*/1) menghapus informasi mengenai adanya *rule H* dari *database*. Dengan memanfaatkan *rule*/2 yang sudah disimpan di *database*, predikat *find_rules*/2 mengoleksikan seluruh *rule* mengenai *H* ke dalam sebuah *list R*. Predikat *add_indices_to_rule*/1 menggunakan *R* untuk membentuk sekaligus menyimpan *rule*/3 yang sesuai. Selanjutnya terjadi pemanggilan rekursif terhadap predikat *add_indices*/0. Pemanggilan rekursif ini akan terus dilakukan hingga tidak ada lagi *has_rules*/1 pada *database*. Setelah pemanggilan rekursif selesai dilakukan, setiap *has_rules*/1 yang baru saja dihapus ditambahkan kembali ke dalam *database* untuk dapat dipergunakan lagi.

3.3.8 Predikat *switch_mode*/1

TABDUAL memiliki dua mode transformasi yang dapat dipilih oleh pengguna, yaitu transformasi *normal* dan transformasi *subsumed*. Mode transformasi *normal* akan menghasilkan program output yang akan menggunakan teknik *tabling* standar yang disediakan oleh XSB Prolog, sedangkan mode transformasi *subsumed* akan menghasilkan program output yang akan menggunakan teknik *tabling* dengan memanfaaatkan fitur *answer subsumption*. Mode ransformasi *normal* dapat digunakan ketika pengguna ingin melakukan *abduction* untuk menememukan

seluruh penjelasan terkait observasi yang diberikan. Mode transformasi *subsumed* dapat digunakan ketika pengguna hanya tertarik untuk menemukan penjelasan-penjelasan minimal terkait observasi yang diberikan. TABDUAL menyediakan predikat *switch_mode*/1 yang dapat digunakan untuk beralih dari satu mode transformasi ke mode lainnya. Hanya ada dua nilai yang dapat digunakan sebagai argumen dari predikat *switch_mode*/1, yaitu *normal* atau *subsumed*. Secara *default*, mode transformasi yang digunakan yaitu mode transformasi *normal*.

3.4 Transformasi

Bagian ini menjelaskan implementasi TABDUAL yang berkaitan dengan fase transformasi program input menjadi program output. Pada TABDUAL fase transformasi dilakukan oleh predikat *transform*/0 dan predikat *transform_just_facts*/0. Pada TABDUAL predikat *transform*/0 didefinisikan sebagai berikut.

```
transform :-
    transform_per_rule,
    transform_if_no_ic,
    transform_abducibles.
```

Kode 3.13: Definisi predikat *transform*/0

Terdapat tiga buah *goal* yang harus dieksekusi oleh predikat *transform*/0. Predikat *transform_per_rule*/0 membentuk transformasi τ', τ^+, dan τ^- untuk program input P (transformasi τ^* dibentuk secara *on-the-fly* saat fase *abduction* menggunakan *dual transformation by need*). Predikat *transform_if_no_ic*/0 membentuk transformasi $\tau^- = not\_\bot(I,I)$ jika pada program input P tidak terdapat *integrity constraint*. Predikat *transform_abducibles*/0 membentuk transformasi τ° untuk program input P. Bagian berikutnya akan menjelaskan lebih lanjut mengenai ketiga predikat di atas serta predikat *transform_just_facts*/0.

3.4.1 Predikat *transform_per_rule*/0

Predikat *transform_per_rule*/0 digunakan untuk membentuk transformasi τ', τ^+, dan τ^-. Transformasi ini dilakukan setelah seluruh program input dibaca dan sudah disimpan di dalam *database* menggunakan predikat-predikat dinamis yang sesuai. Berikut ini definisi dari predikat *transform_per_rule*/0 yang diberikan oleh TABDUAL.

```
transform_per_rule :-
    retract(has_rules(H)),
    find_rules(H, R),
    generate_apostrophe_rules(R),
    generate_positive_rules(H),
    generate_dual_rules(H, R),
    transform_per_rule.
```

Kode 3.14: Definisi predikat *transform_per_rule*/0

Predikat *retract*/1 menghapus informasi mengenai *rule H* dari *database*. Predikat *find_rules*/2 menggunakan H untuk mengoleksikan semua rule mengenai H yang terdapat di *database*. Koleksi tersebut dikumpulkan ke dalam list R yang kemudian digunakan oleh predikat *generate_apostrophe_rules*/1, *generate_positive_rules*/1, dan (disertai dengan H juga digunakan oleh) *generate_dual_rules*/2. Predikat *generate_apostrophe_rules*/1 digunakan untuk membentuk transformasi τ'. Predikat *generate_positive_rules*/1 digunakan untuk membentuk τ^+ dan membentuk *directive* untuk mendefinisikan *tabled predicate*, predikat yang akan di-*table* pada fase *abduction*. Predikat *generate_dual_rules*/1 digunakan untuk membentuk τ^- yang sudah disesuaikan agar dapat menerapkan *dual transformation by need*.

3.4.2 Predikat *transform_if_no_ic*/0

Predikat *transform_if_no_ic*/0 digunakan untuk membentuk $not\_\bot(I,I)$ sebagai hasil transformasi τ^- ketika tidak terdapat *integrity constraint* pada program input. Predikat *transform_if_no_ic*/0 didefinisikan oleh TABDUAL seperti berikut.

```
transform_if_no_ic :-
    find_rules(false, R),
    length(R, 0),
    generate_dual_rules_no_ic.
```

Kode 3.15: Definisi predikat *transform_if_no_ic*/0

Predikat *find_rules*/2 mengoleksikan seluruh *rule* yang merupakan *integrity constraint*, yaitu *rule* yang *head*-nya adalah predikat *false*, dan mengumpulkan hasil koleksi ke dalam *list R*. Untuk mengecek terdapat atau tidaknya *integrity constraint*, predikat *built-in length*/2 digunakan untuk melakukan pengecekan apakah panjang dari R sama dengan nol. Jika ya, maka hasil transformasi $\tau^- = not\_\bot(I,I)$ akan dibentuk oleh predikat *generate_dual_rules_no_ic*/0.

3.4.3 Predikat *transform_abducibles*/0

Predikat *transform_abducibles*/0 digunakan untuk membentuk transformasi τ°. TABDUAL memberikan definisi untuk predikat *transform_abducibles*/0 seperti di bawah ini.

```
transform_abducibles :-
    get_abducibles(A),
    generate_abd_rules(A).
```

Kode 3.16: Definisi predikat *transform_abducibles*/0

Predikat *get_abducibles*/1 mengoleksikan seluruh *abducible* yang terdapat pada program input dan mengumpulkan hasil koleksinya ke dalam *list A*. *Abducible* yang telah dikumpulkan pada *A* digunakan oleh predikat *generate_abd_rules*/1 untuk membentuk transformasi τ°, yaitu transformasi dari masing-masing *abducible* yang ada pada *A*.

3.4.4 Predikat *transform_just_facts*/0

Pada bagian sebelumnya telah dijelaskan bahwa predikat *transform_just_facts*/0 melakukan transformasi terhadap term-term yang terdapat di antara predikat *beginProlog*/0 dan *endProlog*/0 sesuai dengan aturan transformasi pada bagian ???. TABDUAL melakukan transformasi terhadap term-term tersebut tepat setelah membaca predikat *endProlog*/0 pada program input. Berikut ini definisi dari predikat *transform_just_facts*/0 yang pada TABDUAL.

```
transform_just_facts :-
    retract(has_rules(F)),
    generate_pos_fact(F),
    generate_neg_fact(F),
    transform_just_facts.
```

Kode 3.17: Definisi predikat *transform_just_facts*/0

Predikat *retract*/1 menghapus informasi mengenai *rule F* dari *database*. Predikat *generate_pos_fact*/1 dan *generate_neg_fact*/1 menggunakan *F* yang didapat untuk melakukan transformasi terhadap *F*, berturut-turut untuk membentuk *rule* hasil transformasi F' positif dan negatif sesuai dengan aturan transformasi pada bagian ???. Selanjutnya terjadi pemanggilan rekursif terhadap predikat

transform_just_facts/0 yang terus dilakukan hingga seluruh term yang terdapat di antara *beginProlog*/0 dan *endProlog*/0 ditransformasikan.

3.5 *Abduction*

Bagian ini menjelaskan implementasi TABDUAL yang berkaitan dengan fase *abduction*. Pada fase ini, konsep *abduction* digunakan untuk memberikan jawaban terhadap suatu *query* yang diberikan. Seperti yang sudah dijelaskan pada bagian ???, TABDUAL juga melakukan transformasi terhadap *query* yang diberikan sehingga *contextual abduction* dapat diterapkan pada *query* tersebut. Selain itu, sebelum dapat memberikan *query*, program output yang dihasilkan oleh fase transformasi perlu di-*consult* terlebih dahulu.

3.5.1 Men-*consult* Program Output

Agar dimuat ke dalam *database*, program output yang dihasilkan dari transformasi perlu untuk di-*consult* terlebih dahulu. TABDUAL mendefinisikan predikat *load*/1 untuk men-*consult* program output yang dihasilkan. Argumen dari predikat *load*/1 yaitu nama program input yang ditransformasikan. Predikat *load*/1 dapat menggunakan nama program input sebagai argumennya karena TABDUAL menyimpan hasil transformasi ke program output dengan nama berkas yang sama dengan program input, hanya berbeda pada ekstensi berkasnya saja. Sebagai contoh, jika ingin men-*consult* program output hasil dari transformasi program input yang nama berkasnya adalah *in.ab*, maka cukup gunakan *load*(*in*).

3.5.2 Transformasi *Query*

Pada bagian ??? telah dijelaskan bahwa agar dapat mengaplikasikan *contextual abduction*, *query* yang diberikan juga perlu ditransformasikan. TABDUAL mendefinisikan predikat *ask*/2 yang dapat digunakan untuk memberikan *query*. Argumen pertama dari predikat *ask*/2 adalah *query* yang ingin dieksekusi dan argumen keduanya adalah jawaban yang diberikan TABDUAL atas *query* tersebut. Sebelum *query* yang diberikan dieksekusi, predikat *ask*/2 melakukan transformasi terhadap *query* tersebut sesuai dengan aturan transformasi yang sudah dijelaskan pada bagian ???. Selain predikat *ask*/2, TABDUAL juga mendefinisikan predikat *ask*/3 yang dapat digunakan untuk memberikan *query* dengan *input context* tertentu. Argumen pertama dari *ask*/3 yaitu *query* yang ingin dieksekusi, argumen keduanya yaitu *input context* yang ingin diberikan dan direpresentasikan sebagai sebuah

list, dan argumen ketiganya yaitu jawaban yang diberikan TABDUAL atas *query* tersebut. Berikut ini merupakan definisi dari predikat *ask*/2 dan *ask*/3 yang didefinisikan oleh TABDUAL.

```
ask(Q, O) :-
    ask(Q, [], O).
ask(Q, I, O) :-
    transform_and_call_query(Q, I, O).
```

Kode 3.18: Definisi predikat *ask*/2 dan *ask*/3

Terlihat bahwa predikat *ask*/2 akan mengeksekusi predikat *ask*/3 tanpa *input context* apapun. Selanjutnya, predikat *transform_and_call_query*/3 melakukan transformasi sekaligus melakukan eksekusi terhadap *query Q* yang diberikan.

3.6 *Answer Subsumption*

Bagian ini menjelaskan bagaimana TABDUAL mengimplementasikan fitur *answer subsumption* yang disediakan oleh XSB.

3.6.1 *Answer Subsumption* pada XSB

Secara *default*, XSB melakukan *tabling* dengan cara seperti berikut: XSB menambahkan sebuah *answer A* ke dalam *table T* hanya jika *A* bukan merupakan *variant* dari suatu *answer* lain yang sudah ada pada *T*. Walaupun begitu, XSB memberikan pilihan untuk dapat melakukan *tabling* dengan cara yang berbeda, salah satunya yaitu dengan memanfaatkan fitur *answer subsumption*. Pada XSB, terdapat dua jenis penggunaan fitur *answer subsumption*, yaitu *partial order answer subsumption* dan *lattice answer subsumption*. Pada *partial order answer subsumption*, *A* ditambahkan ke dalam *T* hanya jika *A* adalah maksimal dibandingkan dengan *answer* lainnya yang sudah terdapat pada *T* berdasarkan sebuah relasi terurut parsial *P* yang diberikan. Selanjutnya, jika *A* ditambahkan ke dalam *T*, *answer* lainnya yang menurut relasi *P* lebih kecil akan dihapus dari *T*. Sementara, jika menggunakan *lattice answer subsumption*, yang akan ditambahkan ke *T* mungkin saja bukan *A* melainkan gabungan yang diambil dari *A* dan suatu *answer* lainnya A' pada *T* (A' tetap dihapus dari *T*). Meskipun terlihat sederhana, fitur *answer subsumption* ini dapat memberikan efek yang besar terhadap teknik *tabling* itu sendiri.

3.6.2 *Answer Subsumption* pada TABDUAL

Untuk mendapat penjelasan yang minimal saat melakukan *abduction*, TABDUAL mengimplementasikan *tabling* menggunakan *partial order answer subsumption* dengan relasi *subset* pada himpunan sebagai relasi terurut parsial yang digunakan. Untuk menggunakan *partial order answer subsumption*, *directive* yang digunakan untuk mendefinisikan *tabled predicate* harus diubah. Sebagai contoh, untuk *tabled predicate* $t\_ab/3$, *directive* yang digunakan diubah menjadi seperti berikut ini.

```
:- table t_ab(_,_,po(subset/2)).
```

Kode 3.19: *Directive* untuk $t\_ab/3$ menggunakan *answer subsumption*

Predikat $po(subset/2)$ ditambahkan sebagai argumen pada *directive* yang mendefinisikan *tabled predicate*, bersesuaian dengan argumen yang menyatakan *output context* dari *tabled predikat* tersebut. Predikat $po(subset/2)$ menyatakan bahwa *tabled predicate* tersebut akan di-*tabling* menggunakan *partial order answer subsumption* dengan predikat $subset/2$ sebagai relasi terurut parsial yang digunakan. Definisi predikat $subset/2$ akan dijelaskan pada bagian 3.7.4

3.7 Predikat Sistem

Bagian ini menjelaskan predikat-predikat yang didefinisikan secara khusus untuk digunakan oleh TABDUAL dalam melakukan transformasi ataupun dalam melakukan *contextual abduction*.

3.7.1 Predikat *produce_context*//3

Predikat $produce\_context/3$ digunakan untuk menggabungkan himpunan *input context* dan *tabled context* (*context* yang didapatkan dari *table*) untuk menghasilkan *output context*. Argumen-argumen dari predikat $produce\_context/3$ secara berturut-turut yaitu *output context* O, *input context* I, dan *tabled context* E, ketiganya direpresentasikan sebagai *list*. Selain menggabungkan, predikat $produce\_context/3$ juga melakukan pengecekan apakah I konsisten dengan E, yaitu apakah terdapat dua literal yang saling berlawanan pada I dan E. Berikut ini definisi predikat $produce\_context/3$ pada TABDUAL yang didefinisikan secara rekursif untuk menambahkan E satu per satu ke dalam I dengan memperhatikan konsistensinya.

```
produce_context(I, I, []).
produce_context(E, [], E).
produce_context(O, I, [E|EE]) :-
    member(E, I), !,
    produce_context(O, I, EE).
produce_context(O, I, [E|EE]) :-
    negate(E, NE),
    \+ member(NE, I),
    append(I, [E], IE),
    produce_context(O, IE, EE).
```

Kode 3.20: Definisi predikat *produce_context*/3

Terdapat empat definisi untuk predikat *produce_context*/3. Definisi pertama dan kedua digunakan untuk mengatasi berturut-turut jika tidak ada *input context* yang diberikan dan tidak ada *tabled context* yang didapatkan. Definisi ketiga digunakan untuk mengatasi kasus ketika sebuah *abducible E* pada *tabled context* sudah terdapat pada *input context I*. Definisi keempat digunakan untuk menambahkan suatu *abducible E* pada *tabled context* yang tidak terdapat pada *input context I*, tentu dengan memperhatikan konsistensinya. Predikat *produce_context*/3 akan gagal ketika ditemukan inkonsistensi antara *input context* dengan *tabled context*.

3.7.2 Predikat *insert_abducible*//3

Predikat *insert_abducible*/3 digunakan untuk menambahkan sebuah *abducible* pada suatu *input context*. Argumen-argumen dari predikat *insert_abducible*/3 secara berturut-turut yaitu *abducible* yang ingin ditambahkan, *input context* yang ingin ditambahkan dengan *abducible* pada argumen pertama, dan *context* yang dihasilkan dari penambahan tersebut. Sama halnya dengan predikat *produce_context*/3, predikat *insert_abducbile*/3 juga memperhatikan konsistensi saat melakukan penambahan. Predikat *insert_abducible*/3 didefinisikan pada TABDUAL seperti di bawah ini.

```
insert_abducible(A, I, I) :-
    member(A, I), !.
insert_abducible(A, I, O) :-
    negate(A, NA),
    \+ member(NA, I),
    append(I, [A], O).
```

Kode 3.21: Definisi predikat *insert_abducible*/3

Terdapat dua buah definisi untuk predikat *insert_abducible*/3. Definisi pertama digunakan untuk mengatasi kasus ketika *abducible* yang ingin ditambahkan, *A*, sudah terdapat pada *input context I*. Definisi kedua digunakan untuk mengatasi kasus ketika *abducible* yang ingin ditambahkan, *A*, belum terdapat pada *input context I*. Predikat *insert_abducible*/3 akan gagal ketika ditemukan inkonsistensi pada *output context O* setelah menambahkan *abducible A* pada *input context I*.

3.7.3 Predikat *dual*/4

Predikat *dual*/4 digunakan untuk melakukan *dual transformation by need* yang telah dijelaskan pada bagian ???, yaitu dengan membuat transformasi τ^* secara *on-the-fly* ketika memang *rule* spesifik dari τ^* diperlukan saat fase *abduction* dan menyimpan τ^* yang sudah ditransformasikan ke dalam *trie* agar dapat dipergunakan kembali. *Dual rule* yang disimpan pada *trie* direpresentasikan secara *generic* menggunakan predikat *d*(*N*, *P*, *Dual*, *Pos*), menyimpan informasi bahwa *Dual* adalah *dual rule* ke-*N* dari *rule P* disertai dengan *Pos* yang menyimpan informasi mengenai posisi *goal* mana pada *P* yang sedang dan belum di-*dual*-kan. TABDUAL memberikan definisi untuk predikat *dual*/4 sebagai berikut.

```
dual(N, P, I, O) :-
    trie_property(T, alias(dual)),
    dual(T, N, P, I, O).

dual(T, N, P, I, O) :-
    trie_interned(d(N, P, Dual, _), T),
    call_dual(P, I, O, Dual).
dual(T, N, P, I, O) :-
    current_pos(T, N, P, Pos),
    dualize(Pos, Dual, NextPos),
    store_dual(T, N, P, Dual, NextPos),
    call_dual(P, I, O, Dual).
```

Kode 3.22: Definisi predikat *dual*/4

Dengan asumsi bahwa sudah dibuat *trie T* dengan alias *dual*, predikat *dual*/4 menggunakan predikat bantu *dual*/5 yang mendapatkan akses ke *trie T* dari penggunaan predikat *trie_property*/2. Selanjutnya, terdapat dua definisi untuk predikat *dual*/5. Definisi pertama digunakan ketika *dual rule* yang ingin dieksekusi sudah ada tersimpan di dalam *trie* sehingga dapat langsung digunakan tanpa harus membentuk ulang *dual rule* tersebut. *Dual rule* yang tersimpan di dalam *trie* diambil

menggunakan predikat *trie_interned*/2. Setelah berhasil didapatkan, maka predikat *call_dual*/4 melakukan instansiasi *Dual* dengan argumen-argumen yang terdapat pada *P* beserta *input context I*, kemudian melakukan eksekusi *Dual* yang sudah terinstansiasi dan memberikan jawabannya pada *output context* O. Sementara itu, definisi kedua dari *dual*/5 digunakan untuk terlebih dahulu membentuk *dual rule* yang ingin dieksekusi, baru setelah itu *dual rule* tersebut disimpan ke dalam *trie* dan dieksekusi. Predikat *current_pos*/4 digunakan untuk menentukan *Pos*, yaitu posisi *goal* pada *rule* ke-*N* dari *P* yang ingin di-*dual*-kan, yang dapat ditentukan berdasarkan argumen keempat dari predikat *d*/4 yang sudah tersimpan di dalam *trie*. Predikat *dualize*/3 memanfaatkan informasi yang terdapat pada *Pos* untuk membentuk *dual rule Dual* serta membentuk *NextPos* yang memperbarui informasi pada *Pos* sehingga dapat digunakan kembali untuk proses pembentukan *dual rule* berikutnya. Predikat *store_dual*/4 menyimpan *Dual* yang baru saja dibentuk beserta informasi mengenai *N*, *P*, dan *NextPos* ke dalam *trie T* agar dapat dipergunakan kembali. Dengan cara yang sama, *call_dual*/4 melakukan instansiasi dan eksekusi dari *dual rule Dual*.

3.7.4 Predikat Sistem Lainnya

Selain *produce_context*/3, *insert_abducible*/3, dan *dual*/4, TABDUAL mendefiniskan beberapa predikat bantu lainnya, beberapa diantaranya yaitu:

- *find_rules*/2 yang digunakan untuk mengoleksikan seluruh *rule* mengenai suatu predikat yang tersimpan pada *database*, didefinisikan sebagai berikut.

```
find_rules(H, R) :-
    findall(rule(H, B), clause(rule(H, B), true), R).
```

Kode 3.23: Definisi predikat *find_rules*/2

Predikat *find_rules*/ menggunakan predikat *built-in findall*/3 yang dapat mengoleksikan sebuah predikat yang terdapat pada *database*. Predikat *findall*/3 memiliki tiga argumen yaitu *Template*, *Goal*, dan *List*. *Template* menyatakan template yang digunakan untuk menyimpan hasil koleksi, *Goal* menyatakan predikat yang ingin dikoleksikan dari *database*, dan *List* menyatakan himpunan hasil koleksi yang didapat yang direpresentasikan sebagai sebuah *list*. Predikat *clause*/2 yang digunakan sebagai *Goal* menyatakan bahwa *find_rules*/2 hanya mengoleksikan dari *database* dinamis.

- *negate*/2 yang digunakan untuk membentuk negasi dari suatu predikat, didefinisikan sebagai berikut.

```
negate((not A),A).
negate(A,(not A)).
```

Kode 3.24: Definisi predikat *negate*/2

Predikat *negate*/2 cukup menambahkan operator *not* untuk membentuk negasi dari literal positif, atau menghilangkan operator *not* yang sudah ada untuk membentuk negasi dari literal negatif.

- *get_abducibles*/1 yang digunakan untuk mengoleksikan *abducible* yang sudah disimpan pada predikat dinamis, didefinisikan sebagai berikut.

```
get_abducibles(A) :-
    abds(A).
get_abducibles([]).
```

Kode 3.25: Definisi predikat *get_abducibles*/1

Predikat *get_abducibles*/1 cukup melakukan unifikasi argumennya, *A*, dengan *list abducible* yang sudah tersimpan pada *database*. Jika *abducible* tidak ditemukan, maka *get_abducibles*/1 memberikan *list* kosong.

- *subset*/2 yang digunakan untuk melakukan pengecekan apakah suatu *list* merupakan *subset* dari *list* yang lain, didefinisikan sebagai berikut.

```
subset([], _).
subset([L|L1], L2):-
    member(L, L2),
    subset(L1, L2).
```

Kode 3.26: Definisi predikat *subset*/2

Predikat *subset*/2 melakukan pengecekan apakah *list* pada argumen pertama *L1* merupakan *subset* dari *list* pada argumen kedua *L2* dengan cara melakukan pengecekan apakah setiap elemen pada *L1* merupakan elemen dari *L2*. Predikat *subset*/2 digunakan sebagai relasi terurut parsial pada *partial order answer subsumption* yang diimplementasikan oleh TABDUAL.

3.8 Pengujian

3.8.1 Kasus Uji

Berwarna!

Kode 3.27: Potongan skrip submisi *job* melalui torqace

```
# Go To working directory
cd $PBS_O_WORKDIR

#openMPI prerequisite
. /opt/torque/etc/openmpi-setup.sh

mpirun -np 5 -machinefile $PBS_NODEFILE mdrun -v -s \
  curcum400ps.tpr -o md_prod_curcum400_5np.trr -c lox_pr.gro
...
```

3.8.2 Kasus Uji

Contoh skrip yang dimasukkan pada *form* yang disediakan dapat dilihat pada kode 3.28.

Kode 3.28: Potongan `Makefile` *project*

```
# Make file for MPI
SHELL=/bin/sh

# Compiler to use
# You may need to change CC to something like CC=mpiCC
# openmpi : mpiCC
# mpich2  : /opt/mpich2/gnu/bin/mpicxx
CC=mpiCC
...
...
```

BAB 4
EVALUASI DAN ANALISIS

4.1 Hasil Pengujian

4.1.1 Hasil Pengujian Kasus Uji 1

Tabel lain. Hasil tersebut dapat dilihat pada tabel 4.1.

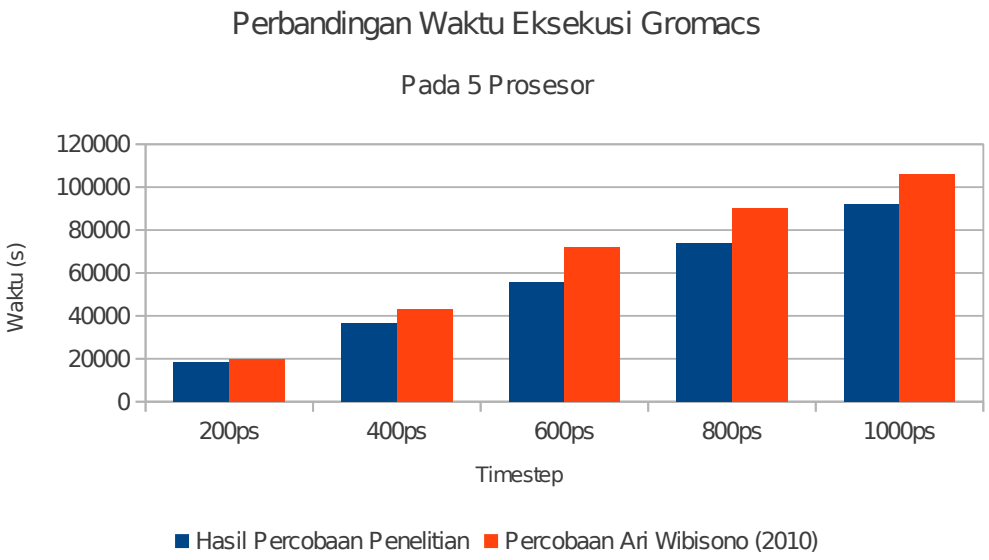
Tabel 4.1: Hasil pengujian menggunakan gromacs

No	*Timestep*	Waktu eksekusi berdasar jumlah prosesor		
		1	2	5
1	200ps	20h:27m:16s	12h:59m:04s	5h:07m:03s
2	400ps	1d:22h:40m:03s	1d:02h:08m:47s	10h:09m:39s
3	600ps	2d:23h:29m:21s	1d:14h:52m:52s	15h:25m:22s
4	800ps	4d:02h:05m:57s	2d:03h:30m:07s	20h:29m:38s
5	1000ps	5d:03h:29m:12s	2d:16h:32m:22s	1d:01h:34m:38s

4.2 Evaluasi Hasil Kasus Uji

4.2.1 Evaluasi Kasus Uji 1

Tabel 4.1 menunjukkan hasil uji coba pada penelitian ini. Gambar 4.1 menunjukkan perbandingan waktu eksekusi pada aplikasi x dengan jumlah prosesor sebanyak 5 buah.



Gambar 4.1: Perbandingan waktu eksekusi x untuk 5 prosesor

BAB 5
PENUTUP

Pada bab terakhir ini,

5.1 Kesimpulan

5.2 Saran

DAFTAR REFERENSI

Buyya, R. (1999). *High Performance Cluster Computing: Architectures and systems*. High Performance Cluster Computing. Prentice Hall PTR.

Guarddin, G. (2010). Percobaan Kompresi Menggunakan MPIBZIP2 pada Cluster Hastinapura. Peronal Communication.

Jackson, D. B., Snell, Q., dan Clement, M. J. (2001). Core algorithms of the maui scheduler. In *Revised Papers from the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, JSSPP '01, pages 87–102, London, UK, UK. Springer-Verlag.

Jones, J. P., Lifka, D., Nitzberg, B., dan Tannenbaum, T. (2002). Cluster workload management. In Sterling, T., editor, *Beowulf cluster computing with Linux*, pages 301–306. MIT Press, Cambridge, MA, USA.

McGuire, T. J. (2010). A gentle way of introducing multi-core programming into the curriculum: tutorial presentation. *J. Comput. Sci. Coll.*, 26(1):124–125.

Mell, P. dan Grance, T. (2009). The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, Information Technology Laboratory. [Diakses 17 April 2013].

Mozdzynski, G. (2012). Concepts of Parallel Computing. http://www.ecmwf.int/services/computing/training/material/com_hpcf.html. [Diakses 20 Maret 2013].

Neal, R. (2010). Cloud computing brings savings in energy, maintenance costs. http://www.federaltimes.com/article/20100323/IT03/3230304/Cloud-computing-brings-savings-energy-maintenance-costs. [Diakses 1 April 2013].

Oxford Dictionaries (2010). http://oxforddictionaries.com/definition/english/parallel. [Diakses 1 April 2013].

Pressman, R. (2010). *Software engineering: a practitioner's approach*. McGraw-Hill higher education. McGraw-Hill Higher Education.

Treese, W. (2004). How to build a supercomputer. *netWorker*, 8(4):15–18.

LAMPIRAN

LAMPIRAN 1 : KODE SUMBER

admin_useraddmaster

Skrip ini diletakkan pada direktori `/usr/sesuatu` dan hanya dapat dieksekusi oleh *root*. Skrip ini berguna untuk menambahkan pengguna baru sesuai dengan konfigurasi baru yang telah ditetapkan.

Kode 1: Skrip menambahkan pengguna baru

```
#!/bin/csh -f
blah blah blah
blah blah blah
blah blah blah
blah blah blah
blah blah blah
```

getuser.cron

Penjelasan skrip disini

Kode 2: *Cronjob* menambahkan pengguna baru

```
#!/bin/bash
# Change these two lines to localize to your system:
# Adapted from /usr/local/sbin/admin_useradd

cat /dev/null > $userlist
for (( i=0; i<${#listemailto[@]}; i++ ))
do
        uname=${listusername[$i]}
        mailto=${listemailto[$i]}

        echo "User $uname created, please use torqace wisely." | mail -s "Torqace
             user registration" $mailto
done
```

LAMPIRAN 2 : BERKAS KONFIGURASI

compute.xml

Kode 3: Berkas `compute.xml`

```
<?xml version="1.0" standalone="no"?>
<kickstart>
<description>
  Compute node XML file
</description>
</kickstart>
```

LAMPIRAN 8 : UAT DAN KUESIONER

Tabel 1: Tabel UAT dan Kuesioner

No.	Langkah Penggunaan	Fitur Berjalan	Tingkat Kemudahan (1-5)	Tingkat Kepuasan (1-5)	Saran / Komentar
		Berhasil /Tidak	1:Sangat sulit ; 5:sangat mudah	1 : Sangat kecewa ; 5 : sangat puas	
Use Case : Login					
1.1	Pengguna berada pada halaman depan torqace				
1.2	Pengguna memasukkan username dan password pada field yang telah disedi-akan.Kemudian menekan tombol 'login'				
1.3	Apabila Sukses, maka pengguna masuk ke dalam sistem dan dihadapkan pada menu utama				
Use Case : Register					
2.1	Pengguna berada pada halaman registrasi pengguna torqace				

2.2	Pengguna memasukkan user-name,password, dan email pada field yang telah disediakan. Kemudian menekan tombol 'submit'				
2.3	Sistem akan mengonfirmasi masukan, dan akan mengirimkan email untuk mem-beritahu pengguna apabila proses pendaf-taran telah selesai				
Use Case : Logout					
3.1	Pengguna memilih menu untuk melakukan logout				
3.2	Sistem akan mengeluarkan pengguna, dan pengguna tidak dapat menggunakan fitur-fitur utama aplikasi				
Use Case : Upload Job Sederhana					
4.1	Pengguna memilih menu upload file/pro-ject pada menu utama				
4.2	Pengguna memilih pilihan 'single file' pada tipe project				

4.3	Pengguna memilih berkas yang akan diunggah, mengisi label, dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
4.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
4.5	Sistem akan menampilkan informasi terkait berkas yang diupload				
Use Case : Upload Job Compressed					
5.1	Pengguna memilih menu upload file/project pada menu utama				
5.2	Pengguna memilih pilihan 'compressed files' pada tipe project				
5.3	Pengguna memilih arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
5.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				

5.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				
Use Case : Upload Array Job					
6.1	Pengguna memilih menu upload file/project pada menu utama				
6.2	Pengguna memilih pilihan 'array' pada tipe project				
6.3	Pengguna memilih arsip-arsip yang akan diunggah, mengisi label, menentukan akan melakukan make atau tidak dan menentukan apakah akan menimpa project sebelumnya dengan nama yang sama atau tidak				
6.4	Pengguna menekan tombol 'submit' dan mengonfirmasi				
6.5	Sistem akan menampilkan informasi terkait berkas yang diupload dan diekstrak. Keluaran make juga akan ditampilkan bila dipilih				

Use Case : Melihat antrian pada queue					
7.1	Pengguna memilih menu queue status pada menu utama				
7.2	Pengguna berada pada halaman yang berisi informasi queue				
Use Case : Melihat detil antrian					
8.1	Dari halaman status queue, pengguna memilih job tertentu				
8.2	Informasi mengenai detil job tersebut ditampilkan dalam bentuk tabel				
8.2.1	Apabila job tersebut bukan milik pengguna, maka sistem akan melarang pengguna melihat informasi detil suatu job				
Use Case : Membuat script job					
9.1	Pengguna memilih untuk melakukan 'generate script' baik dari laporan upload berkas, atau dari penjelajahan direktori				
9.2	Pengguna mengisi nama job, parameter job, dan script yang akan dijalankan.				
9.3	Pengguna mengonfirmasi konfirmasi submit job				

9.4	Pengguna dapat melihat informasi script secara keseluruhan dan pesan apakah terjadi kegagalan atau tidak, serta id job yang diberikan				
Use Case : Load spesifikasi job lain					
10.1	Pengguna berada pada halaman untuk membuat script				
10.2	Pengguna memilih 'Load a Previous Job'				
10.3	Pengguna memilih job mana yang akan dimuat dan menekan tombol 'Load'				
10.4	Pengguna kembali ke halaman pembuatan script dengan spesifikasi job sebelumnya				
Use Case : Menjelajah Direktori					
11.1	Pengguna memilih menu 'View File/Project' pada menu utama				
11.2	Pengguna dapat melakukan navigasi untuk masuk ke dalam direktori tertentu, atau kembali ke direktori diatasnya, dan dapat melihat terdapat berkas apa saja dalam direktori				

Use Case : Menghapus Berkas/Direktori					
12.1	Pengguna berada pada halaman penjelajahan direktori				
12.2	Pengguna memilih pilihan untuk menghapus berkas/direktori di samping item yang akan dihapus				
12.3	Pengguna mengonfirmasi konfirmasi penghapusan				
Use Case : Mengunduh Berkas/Direktori					
13.1	Pengguna berada pada halaman penjelajahan direktori				
13.2	Pengguna memilih pilihan untuk mengunduh berkas/direktori di samping item yang akan dihapus				
Use Case : Melihat Berkas					
14.1	Pengguna berada pada halaman penjelajahan direktori				
14.2	Pengguna memilih berkas yang berupa berkas teks				
14.3	Sistem akan menampilkan konten dari berkas tersebut				