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 terdapat 3 buah contoh untuk membuat enumerate pada latex :

1. Makan
2. Minum

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

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 *TABLING* DAN *ABDUCTION* PADA *CONTEXTUAL ABDUCTION*
- Bab 4 IMPLEMENTASI
- Bab 5 EVALUASI DAN ANALISIS

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

BAB 2
LANDASAN TEORI

Pada bab ini penulis menjelaskan hal, teori, dan konsep yang berkaitan dengan implementasi *tabling* pada *contextual abduction* menggunakan *answer subsumption*. Penulis mengawali penjelasan dengan konsep dasar-dasar yang diperlukan dalam pemrograman logika, dilanjutkan dengan konsep mengenai *abduction*, *tabling*, lalu yang terakhir yaitu *answer subsumption*.

2.1 Pendahuluan

Diberikan himpunan alfabet $\mathcal{A}$ dari bahasa $\mathcal{L}$, dapat dibentuk himpunan berhingga yang saling *disjoint* yaitu himpunan konstanta, himpunan simbol fungsi, dan himpunan simbol predikat. Selain itu, diasumsikan pada himpunan alfabet juga mengandung himpunan simbol variabel. Simbol *underscore* (_) secara khusus untuk menyatakan sebuah variabel *anonymous*. Sebuah term pada $\mathcal{A}$ didefinisikan secara rekursif sebagai salah satu dari variabel, konstanta, atau ekspresi dengan bentuk $f(t_1, \ldots, t_n)$ dengan f adalah sebuah simbol fungsi pada $\mathcal{A}$ dan t_i adalah term. Sebuah *atom* pada $\mathcal{A}$ adalah ekspresi dengan bentuk $p(t_1, \ldots, t_n)$ dengan p adalah sebuah simbol predikat pada $\mathcal{A}$ dan t_i adalah term. Pada tulisan ini, notasi p/n digunakan untuk menyatakan sebuah simbol p yang memiliki arity n. Sebuah *literal* yaitu sebuah atom a atau negasinya *not* a. Literal negatif (misalnya *not* a) disebut juga sebagai *default* literal.

Sebuah term (atom maupun literal) dikatakan *ground* jika term tersebut tidak mengandung variabel. Himpunan seluruh *ground* term (yaitu *ground* atom) pada $\mathcal{A}$ disebut sebagai *Herbrand universe* (atau *Herbrand base*) dari $\mathcal{A}$.

Definisi 2.1: Program Logika. Program logika (normal) adalah himpunan berhingga dari *rule* yang memiliki bentuk:

$$H \leftarrow L_1, \ldots, L_n$$

dengan H adalah sebuah atom, $m \geq 0$, dan L_i adalah literal.

Operator koma pada sebuah *rule* dibaca sebagai konjungsi. Sebuah program logika dikatakan *definit* jika tidak terdapat *rule* yang mengandung *default* literal.

Agar menyesuaikan dengan standar yang ada, *rule* yang memiliki bentuk $H \leftarrow$ cukup ditulis sebagai H. *Rule* yang memiliki bentuk seperti ini disebut sebagai sebuah *fakta*.

Himpunan alfabet $\mathcal{A}$ yang membentuk program logika P diasumsikan mencakup seluruh konstanta, serta simbol fungsi dan simbol predikat yang muncul pada P. Herbrand *universe* (atau *base*) dari P dapat diartikan sebagai Herbrand *universe* (atau *base*) dari $\mathcal{A}$. Herbrand *base* dari P dinyatakan sebagai $\mathcal{H}_P$. Diberikan program logika yang *ground*, dapat dibentuk himpunan *ground rule* pada P dengan melakukan substitusi terhadap variabel-variabel pada P dengan setiap elemen pada Herbrand *universe*-nya.

Selanjutnya akan didefinisikan interpretasi *two-valued* dan *three-valued* serta model dari sebuah program logika. Misal F adalah sebuah himpunan atom, $F = \{a_1, \ldots, a_n\}$. Himpunan *not F* dapat didefinisikan sebagai $\{not\ a_1, \ldots, not\ a_n\}$.

Definisi 2.2: Interpretasi *Two-valued*. Interpretasi *two-valued I* terhadap suatu program logika P yaitu himpunan literal

$$I = T \cup not\ F$$

sedemikan sehingga $T \cup F = \mathcal{H}_P$ dan $T \cap F = \emptyset$.

Himpunan T merupakan himpunan atom-atom yang bernilai *true* pada I, sedangkan himpunan F merupakan himpunan atom-atom yang bernilai *false* pada I. Interpretasi I dikatakan *two-valued* karena nilai kebenaran dari sebuah atom hanya memiliki dua kemungkinan, yaitu *true* atau *false*.

Sebagai alternatif, interpretasi *three-valued* memperkenalkan nilai kebenaran yang baru selain *true* dan *false*, yaitu *undefined*, sehingga dapat digunakan untuk merepresentasikan *incomplete knowledge*.

Definisi 2.3: Interpretasi *Three-valued*. Interpretasi *three-valued I* terhadap suatu program logika P yaitu himpunan literal

$$I = T \cup not\ F$$

sedemikan sehingga $T \subseteq \mathcal{H}_P$, $F \subseteq \mathcal{H}_P$, dan $T \cap F = \emptyset$.

Pada interpretasi *three-valued*, himpunan T merupakan himpunan atom-atom yang bernilai *true* pada I, himpunan F merupakan himpunan atom-atom yang bernilai

false pada I, sedangkan atom-atom yang tidak terdapat pada T maupun F dinyatakan bernilai *undefined*.

Interpretasi I pada program logika P juga dapat dilihat sebagai sebuah fungsi $I : \mathcal{H}_P \rightarrow \mathcal{V}$, dengan $\mathcal{V} = \{0, 0.5, 1\}$, didefinisikan sebagai:

$$I(A) = \begin{cases} 0 & \text{jika } not\ A \in I \\ 1 & \text{jika } A \in I \\ 0.5 & \text{lainnya} \end{cases}$$

Dapat dilihat bahwa pada interpretasi *two-valued* tidak terdapat atom A sedemikian sehingga $I(A) = 0.5$.

Model didefinisikan seperti biasanya, yaitu dengan menggunakan fungsi *truth valuation*.

Definisi 2.4: *Truth Valuation*. Jika I adalah sebuah interpretasi, *truth valuation* $\hat{I}$ terhadap I adalah sebuah fungsi $\hat{I} : \mathcal{F} \rightarrow \mathcal{V}$, dengan $\mathcal{F}$ adalah himpunan *ground* literal, konjungsi dari literal-literal, dan *rule* yang dibentuk dari $\mathcal{L}$. $\hat{I}$ didefinisikan sebagai berikut:

- Jika L merupakan *ground* atom, maka $\hat{I}(L) = I(L)$.
- Jika L merupakan *default* literal, yaitu $L = not\ A$, maka $\hat{I}(L) = 1 - \hat{I}(A)$.
- Jika S dan T merupakan konjungsi dari literal, maka $\hat{I}(S,T) = min(\hat{I}(S), \hat{I}(T))$
- Jika $H \leftarrow B$ merupakan sebuah rule, dengan B merupakan konjungsi dari literal, maka:

$$I(H \leftarrow B) = \begin{cases} 1 & \text{jika } \hat{I}(B) \leq \hat{I}(H) \\ 0 & \text{lainnya} \end{cases}$$

Untuk setiap $F \in \mathcal{F}$, nilai 0, 0.5, dan 1 dari $\hat{I}(F)$ secara berturut-turut menyatakan nilai *false*, *undefined*, dan *true*. Dapat dituliskan $I \models F$, untuk $F \in \mathcal{F}$, jika dan hanya jika $\hat{I}(F) = 1$.

Definisi 2.5: Model. Sebuah interpretasi I (*two-valued* ataupun *three-valued*) dikatakan sebagai model dari program P jika dan hanya jika untuk setiap *ground instance* $H \leftarrow B$ dari sebuah rule pada program P, $\hat{I}(H \leftarrow B) = 1$.

Selanjutnya didefinisikan pengurutan (*ordering*) terhadap interpretasi dan model seperti berikut.

Definisi 2.6: ***Classical Ordering***. Jika I dan J merupakan interpretasi maka dapat dikatakan bahwa $I \preceq J$ jika $I(A) \leq J(A)$ untuk setiap atom *ground* A. Jika $\mathcal{I}$ merupakan kumpulan interpretasi, maka suatu interpretasi $I \in \mathcal{I}$ dikatakan *minimal* di $\mathcal{I}$ jika tidak terdapat interpretasi $J \in \mathcal{I}$ sedemikian sehingga $J \preceq I$ dan $J \neq I$. Suatu interpretasi I dikatakan *least* di $\mathcal{I}$ jika $I \preceq J$ untuk setiap interpretasi lainnya $J \in \mathcal{I}$. Suatu model M disebut sebagai *minimal* model jika model tersebut *minimal* di antara seluruh model untuk P. Suatu model M disebut sebagai *least* model jika model tersebut *least* di antara seluruh model untuk P.

Definisi 2.7: ***Fitting Ordering***. Jika I dan J merupakan interpretasi maka dapat dikatakan bahwa $I \preceq_F J$ jika dan hanya jika $I \subseteq J$. Jika $\mathcal{I}$ merupakan kumpulan interpretasi, maka suatu interpretasi $I \in \mathcal{I}$ dikatakan *F-minimal* di $\mathcal{I}$ jika tidak terdapat interpretasi $J \in \mathcal{I}$ sedemikian sehingga $J \preceq_F I$ dan $J \neq I$. Suatu interpretasi I dikatakan *F-least* di $\mathcal{I}$ jika $I \preceq_F J$ untuk setiap interpretasi lainnya $J \in \mathcal{I}$. Suatu model M disebut sebagai *F-minimal* model jika model tersebut *F-minimal* di antara seluruh model untuk P. Suatu model M disebut sebagai *F-least* model jika model tersebut *F-least* di antara seluruh model untuk P.

Dapat dilihat bahwa *classical ordering* berkaitan dengan derajat kebenaran (*degree of truth*) mengenai I, sedangkan *fitting ordering* berkaitan dengan derajat informasi (degree of information). Pada *fitting ordering*, *undefined* bernilai lebih kecil dibandingkan dengan *true* ataupun dengan *false*, menyebabkan nilai *true* dan *false* tidak dapat dibandingkan.

Pada [1] telah ditunjukkan bahwa setiap program definit memiliki tepat satu buah *least model*, yang memberikan semantik pada program definit tersebut, disebut sebagai *least model semantic*. Semantik untuk program yang tidak definit, yaitu program yang memperbolehkan terdapatnya *default* literal pada *body* dari sebuah rule, juga telah diperkenalkan pada [2] sebagai *Stable Model Semantic*. Sebelumnya, akan dijelaskan terlebih dahulu operator Gelfond-Lifschitz Γ yang dapat digunakan pada interpretasi *two-values* dari suatu program P.

Definisi 2.8: Operator Gelfond-Lifschitz. Misal P merupakan sebuah program logika dan I merupakan interpretasi *two-valued* terhadap P. *GL Transformation* dari P *modulo* I yaitu program $\frac{P}{I}$ yang didapat dari P dengan melakukan operasi-operasi

berikut:

1. Hapus *rule* pada P yang mengandung *default* literal $L = not\ A$ sedemikian sehingga $\hat{I}(L) = 0$.

2. Hapus dari *rule-rule* yang tersisa *default* literal $L = not\ A$ yang memenuhi $\hat{I}(A) = 1$.

Karena program $\frac{P}{I}$ yang dihasilkan merupakan program *definit*, $\frac{P}{I}$ memiliki tepat satu *least model* J, dituliskan sebagai $\Gamma(I) = J$.

Pada [2] telah ditunjukkan bahwa *fixed point* dari operator Gelfond-Lifschitz Γ terhadap program P merupakan *minimal model* dari P.

Definisi 2.9: Stable Model Semantics. Suatu interpretasi *two-valued* I terhadap program logika P merupakan *stable model* dari P jika $\Gamma(I) = I$.

Contoh 1. Sebagai ilustrasi, program P:

$$a \leftarrow not\ b.$$
$$b \leftarrow not\ a.$$
$$c \leftarrow not\ d.$$
$$d \leftarrow not\ e.$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

memiliki dua *stable model*:

$$I_1 = \{a, d, p, not\ b, not\ c, not\ e\} \text{ dan } I_2 = \{b, d, p, not\ a, not\ c, not\ e\}$$

dengan program $\frac{P}{I_1}$:

$$a \leftarrow .$$
$$d \leftarrow .$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

dan program $\frac{P}{I_2}$:

$$b \leftarrow .$$
$$d \leftarrow .$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

Seperti yang sudah dijelaskan sebelumnya, Stable Model Semantics dapat memberikan makna pada program yang memperbolehkan terdapatnya *default* literal pada *body* dari sebuah *rule*. Namun, untuk beberapa program yang cukup "unik", Stable Model Semantics tetap tidak dapat menemukan *stable model*-nya, misalnya program $p \leftarrow not\ p$. Selain itu, beberapa program yang memiliki *stable model*, semantiknya terkadang tidak sesuai dengan semantik yang sesungguhnya diharapkan (lihat [3]).

Pada [4], *Well-Founded Semantics* diperkenalkan untuk mengatasi masalah yang ditemukan pada Stable Model Semantics. Well-Founded Semantics dapat dilihat sebagai *three-valued Stable Model Semantics* [5]. Agar dapat memberikan formalisasi terhadap ide mengenai *three-valued stable model*, pada bahasa $\mathcal{L}$ yang digunakan oleh program ditambahkan konstanta proposisi $\boldsymbol{u}$ yang menyatakan nilai *undefined* untuk interpretasi manapun. Oleh karena itu, setiap interpretasi memenuhi:

$$\hat{I}(\mathbf{u}) = \hat{I}(not\ \mathbf{u}) = 0.5$$

Program *non-negative* merupakan program yang *body* pada *rule-rule*-nya hanya dibentuk oleh atom atau $\boldsymbol{u}$. Telah dibuktikan di [5] bahwa setiap program logika *non-negative* pasti memliki tepat satu *least three-valued model*.

Selanjutnya akan didefinisikan operator Γ^*, yaitu perluasan dari operator Gelfond-Lifschitz terhadap interpretasi *three-valued*.

Definisi 2.10: Operator *Extended* Gelfond-Lifschitz. Misal P merupakan sebuah program logika dan I merupakan interpretasi *three-valued* terhadap P. *Extended GL Transformation* dari P *modulo* I yaitu program $\frac{P}{I}$ yang didapat dari P dengan melakukan operasi=operasi berikut:

1. Hapus *rule* pada P yang mengandung *default* literal $L = not\ A$ sedemikian sehingga $\hat{I}(L) = 0$.

2. Dari *rule-rule* yang tersisa, ganti *default* literal $L = not\ A$ yang memenuhi $\hat{I}(A) = 0.5$ dengan $\boldsymbol{u}$.

3. Hapus dari *rule-rule* yang tersisa *default* literal $L = not\ A$ yang memenuhi $\hat{I}(A) = 1$.

Karena program $\frac{P}{I}$ yang dihasilkan merupakan program *non-negative*, $\frac{P}{I}$ memiliki tepat satu *three-valued least model J*, dituliskan sebagai $\Gamma^*(I) = J$.

Definisi 2.11: Well-Founded Semantic. Suatu interpretasi *three-valued I* terhadap program logika P merupakan *three-valued stable model* dari P jika $\Gamma^*(I) = I$. Well-Founded Semantics dari P ditentukan oleh *F-least three-valued stable model* dari P, dan bisa didapatkan dengan melakukan iterasi pada Γ^* dimulai dari interpretasi kosong.

Contoh 2. Sebagai ilustrasi, akan digunakan program pada Contoh 1. Misal I_0 adalah interpretasi kosong.

- *Least three-valued model* dari $\frac{P}{I_0}$:

$$a \leftarrow \boldsymbol{u}.$$
$$b \leftarrow \boldsymbol{u}.$$
$$c \leftarrow \boldsymbol{u}.$$
$$d \leftarrow \boldsymbol{u}.$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

  yaitu $\Gamma^*(I_0) = \{not\ e\}$.

- Misal $I_1 = \Gamma^*(I_0)$. *Least three-valued model* dari $\frac{P}{I_1}$:

$$a \leftarrow \boldsymbol{u}.$$
$$b \leftarrow \boldsymbol{u}.$$
$$c \leftarrow \boldsymbol{u}.$$
$$d \leftarrow .$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

  yaitu $\Gamma^*(I_1) = \{d, not\ e\}$.

- Misal $I_2 = \Gamma^*(I_1)$. *Least three-valued model* dari $\frac{P}{I_2}$:

$$a \leftarrow \boldsymbol{u}.$$
$$b \leftarrow \boldsymbol{u}.$$
$$d \leftarrow .$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

  yaitu $\Gamma^*(I_2) = \{d, not\ c, not\ e\}$.

- Misal $I_3 = \Gamma^*(I_2)$. *Least three-valued model* dari $\frac{P}{I_3}$:

$$a \leftarrow \boldsymbol{u}.$$
$$b \leftarrow \boldsymbol{u}.$$
$$d \leftarrow .$$
$$p \leftarrow a.$$
$$p \leftarrow b.$$

  yaitu $\Gamma^*(I_3) = \{d, not\ c, not\ e\}$.

Didapatkan *well-founded model* dari P yaitu $I_3 = \{d, not\ c, not\ e\}$, yaitu d bernilai *true*, c dan e bernilai *false*, dan a, b, dan p bernilai *undefined*. Selanjutnya, *well-founded model* dari program P dituliskan sebagai $WFM(P)$.

2.2 *Abduction*

Istilah *abduction* pertama kali diperkenalkan oleh Peirce [6], yaitu suatu bentuk penalaran logika yang berawal dengan diberikan observasi lalu mencari penjelasan yang paling baik berdasarkan hipotesis-hipotesis yang ada. Pada pemrograman logika, proses *abduction* diwujudkan dengan menambahkan *abduction hypotheses* sebagai elemen baru pada pemrogaman logika, disebut sebagai *abducible*. Sebuah *abducible* dinyatakan sebagai sebuah atom Ab, disebut sebagai *positive abducible*, atau negasinya Ab^*, disebut sebagai *negative abducible*.

Selanjutnya, akan dijelaskan mengenai *abductive framework* pada pemrograman logika [7] yang didalamnya terdapat *integrity constraint* yang digunakan untuk membatasi *abduction*. Definisi-definisi yang diberikan mengacu kepada [8].

Definisi 2.12: ***Integrity Constraint***. Sebuah *Integrity constraint* pada dasarnya merupakan sebuah *rule* yang berbentuk penyangkalan:

$$\bot \leftarrow L_1, \ldots, L_m.$$

dengan $\bot$ merupakan simbol predikat pada $\mathcal{L}$ yang menyatakan *false*, $m \geq 1$, dan $L_i (1 \leq i \leq m)$ merupakan literal.

Definisi 2.13: ***Abductive Framework***. Sebuah tripel $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$ dikatakan sebagai sebuah *abductive framework*, dengan $\mathcal{AB}$ merupakan himpunan predikat *abducible* (beserta *arity*-nya), P merupakan program logika pada bahasa $\mathcal{L} \backslash \{\bot\}$ sedemikian sehingga tidak terdapat *rule* pada P yang *head*-nya adalah predikat pada $\mathcal{AB}$, dan $\mathcal{IC}$ merupakan himpunan *integrity constraint*.

Diberikan suatu *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$, $\mathcal{AB}_{gL}$ menyatakan himpunan berhingga *abducible* yang *ground* yang dapat dibentuk dari $\mathcal{AB}$. Secara khusus, $\mathcal{AB}_{gL} = \mathcal{AB}$ jika seluruh predikat *abducible* pada $\mathcal{AB}$ adalah proposisi (predikat dengan *arity* 0).

Definisi 2.14: ***Abductive Scenario***. Misal F merupakan *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$. Sebuah *abductive scenario* pada F merupakan tupel $\langle P, \mathcal{AB}, \mathcal{S}, \mathcal{IC} \rangle$, dengan $\mathcal{S} \subseteq \mathcal{AB}_{gL}$ dan tidak terdapat $A \in \mathcal{S}$ sedemikian sehingga *not* $A \in \mathcal{S}$, dengan kata lain, $\mathcal{S}$ harus konsisten. Konsistensi dari *abductive scenario* dapat dipertahankan dengan memberikan *integrity constraint* $\bot \leftarrow Ab, Ab^*$.

Misalkan terdapat observasi O yaitu sebuah himpunan literal, analog dengan sebuah query pada pemrograman logika. *Abduction* terhadap O yaitu menemukan *abductive solution* yang konsisten terhadap tiap *goal* pada O yang tetap mempertahankan *integrity constraint*, dengan *abductive solution* yang terdapat pada semantik dari program P didapatkan dengan cara mengganti pada program P seluruh *abducible* pada $\mathcal{S}$ dengan nilai kebenaran masing-masing. Selanjutnya akan didefinisikan secara formal *abductive solution* pada Well-Founded Semantic.

Diberikan *abductive scenario* $\langle P, \mathcal{AB}, \mathcal{S}, \mathcal{IC} \rangle$ pada *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$, didefinisikan $P_{\mathcal{S}}$ sebagai himpunan terkecil dari *rule* pada P yang untuk setiap $A \in \mathcal{AB}_{gL}$ mengandung fakta A jika $A \in \mathcal{S}$, atau $A \leftarrow \boldsymbol{u}$ jika $A \notin \mathcal{S}$. Secara ekivalen, alih-alih menambahkan rule $A \leftarrow \boldsymbol{u}$ ke $P_{\mathcal{S}}$, A yang bersesuaian pada P dan $\mathcal{IC}$ dapat digantikan dengan $\boldsymbol{u}$.

Definisi 2.15: ***Abductive Solutions***. Misal F merupakan *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$ dan $\langle P, \mathcal{AB}, \mathcal{S}, \mathcal{IC} \rangle$ merupakan *abductive scenario* pada F. Him-

punan *abducible* (yang konsisten) $\mathcal{S}$ merupakan *abductive solution* dari F jika $\bot$ bernilai *false* pada $M_s = WFM(P \cup P_{\mathcal{S}} \cup \mathcal{IC})$. $\mathcal{S}$ dapat dikatakan sebagai *abductive solution* untuk *query* Q jika Q bernilai *true* pada M_s, ditulis sebagai $M_s \models Q$.

Abduction pada pemrograman logika dapat diselesaikan dengan melakukan pencarian solusi terhadap *query* yang diberikan. Pencarian dilakukan secara *top-down* melalui suatu komputasi yang melakukan eksekusi (*call*) terhadap predikat-predikat logika yang ada pada program. *Correctness* dari komputasi ini membutuhkan semantik yang relevan, sehingga dalam mencari solusi dari sebuah *query* komputasi tidak dilakukan pada seluruh model yang ada, melainkan hanya dilakukan pada *rule-rule* pada program yang relevan dengan *query* tersebut. Well-Founded Semantic memiliki properti yang relevan, yaitu dapat menemukan *abducible* mana saja yang menjadi solusi beserta nilai yang didapatkan dengan melakukan komputasi *top-down* seperti yang disebutkan sebelumnya. *Abducible* yang tidak terdapat pada solusi dapat dianggap tidak relevan dengan *query* yang diberikan.

2.3 *Tabling*

Pada pemrograman logika, *tabling* merupakan teknik menggunakan kembali solusi yang sudah didapat, tanpa melakukan komputasi ulang, dengan menyimpan pasangan *goal* dan *answer* dari sebuah *query*, yang didapatkan setelah melakukan ekseskusi *query* tersebut. *Tabling* sudah disediakan sebagai sebuah fitur pada beberapa distribusi Prolog, misalnya XSB Prolog, YAP Prolog, B-Prolog, Ciao, Mercury, dan ALS Prolog. Meskipun dikembangankan dari ide yang sangat sederhana, penggunaan *tabling* memiliki pengaruh yang cukup besar [9]:

- *Tabling* memiliki properti *bounded term-size* yang memastikan bahwa suatu program selalu *terminate*, yaitu dengan membatasi jumlah *subgoal* dan *answer* yang dapat dibentuk.
- *Tabling* dapat dipergunakan untuk melakukan evaluasi terhadap program yang mengandung negasi menurut Well-Founded Semantic.
- Untuk program berskala besar, penggunaan *tabling* menyebabkan evaluasi terhadap *query* menjadi jauh lebih optimal.
- *Tabling* dapat dengan mudah diintegrasikan dengan Prolog, sehingga beberapa implementasi yang sudah terdapat pada Prolog dapat dipergunakan, serta menyebabkan tidak diperlukannya penggunaan bahasa pemrograman lain untuk membentuk sistem yang utuh.

2.4 *Answer Subsumption*

Pada bagian sebelumnya telah dijelaskan bahwa *tabling* disediakan sebagai sebuah fitur pada beberapa distribusi Prolog. Namun, terdapat fitur tambahan yang dapat digunakan pada *tabling* yang tidak disediakan pada beberapa distribusi Prolog yang sudah disebutkan, yaitu fitur yang disebut sebagai *answer subsumption*.

Pada sebagian besar distribusi Prolog, *tabling* dilakukan dengan menggunakan teknik *answer variance*: sebuah *answer A* akan ditambahkan ke dalam *table T* hanya jika *A* bukan merupakan *variant* dari suatu *answer* lain yang sudah ada pada *T*. Meskipun dengan menggunakan *answer variance* sudah cukup agar *tabling* dapat melakukan komputasi terhadap Well-Founded Semantic, ataupun untuk memastikan terminasi pada program dengan memanfaatkan properti *bounded term-size*, beberapa teknik lain dalam menentukan kapan dan bagaimana suatu *answer* ditambahkan ke dalam *table* dapat dibuat. Dengan menggunakan *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 *po* yang diberikan. Selanjutnya, jika *A* ditambahkan ke dalam *T*, *answer* lainnya yang menurut relasi *po* lebih kecil akan dihapus dari *T*. Sementara itu, dengan 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*, dengan A' tetap dihapus dari *T*.

Meskipun terlihat sederhana, fitur *answer subsumption* ini dapat memberikan efek yang besar terhadap teknik *tabling* itu sendiri. *Partial order answer subsumption* memungkinkan *table* untuk menyimpan hanya *answer-answer* yang dianggap maksimal berdasarkan preferensi tertentu, sedangkan *lattice answer subsumption* dapat digunakan untuk membentuk fondasi untuk *multi-valued logics*, *quantitative logics*, dan interpretasi abstrak untuk proses dan program logika [10].

BAB 3
TABLING DAN *ABDUCTION* PADA *CONTEXTUAL ABDUCTION*

Pada *abduction*, sering kali ditemukan bahwa *abductive solution* yang didapatkan pada suatu proses *abduction* ternyata memiliki relevansi dengan proses *abduction* lainnya, menyebabkan bahwa sebenarnya *abductive solution* tersebut dapat dipergunakan kembali. Seperti yang sudah dijelaskan pada bagian sebelumnya, teknik *tabling*, terlepas dari *abduction*, dapat dipergunakan untuk mempergunakan kembali solusi yang sudah didapatkan [11]. Secara konsep, *tabling* tampaknya juga dapat diterapkan pada *abduction*, yaitu untuk mempergunakan kembali *abductive solution* yang sudah didapat. Namun praktisnya, *abductive solution* dari *goal G* tidak serta-merta dapat di-*tabling* karena solusi tersebut secara khusus terkait dengan konteks pada proses *abduction* jika diberikan *goal G*. *Abductive context* dari *goal G* dapat diartikan sebagai himpunan *abducible* yang memberikan informasi mengenai konteks dari proses *abduction* yang digunakan untuk mencari *abductive solution* dari *goal G*.

Pada bagian ini, penulis akan menjelaskan teknik untuk mempergunakan kembali *abductive solution* yang didapat dari suatu *abductive context* pada *abductive context* lainnya dengan memanfaatkan *tabling*. Teknik melakukan *tabling abductive solution* pada *contextual abduction* ini disebut sebagai TABDUAL [12]. Teknik ini didasari oleh ABDUAL [8], yaitu suatu pendekatan yang dapat digunakan untuk melakukan komputasi terhadap *abduction* pada Well-Founded Semantic. TABDUAL merealisasikan teori transformasi program yang diberikan pada ABDUAL yaitu *dual program transformation*, ditambah dengan adanya *tabling* untuk menyimpan *abductive solution*, sekaligus mengatasi permasalahan yang ditemukan ketika melakukan *abduction* pada *goal* negatif.

Penulis akan memulai menjelaskan dengan memberikan motivasi dan ide dasar diperlukannya *tabling* dan *abduction*, kemudian menunjukkan bagaimana konsep dan realisasi *tabling* dan *abduction* pada transformasi program yang dilakukan TABDUAL.

3.1 Motivasi dan Ide

Contoh 3.1. Misal terdapat sebuah *abductive framework* $\langle P_1, \{a/0, b/0\}, \emptyset \rangle$ dengan program P_1 sebagai berikut:

$$q \leftarrow a.$$
$$s \leftarrow q, b.$$
$$t \leftarrow s, q.$$

Misal akan diberikan tiga buah *query* untuk memberikan penjelasan dari q, s, dan t secara berturut-turut.

- *Query* yang pertama, q, terpenuhi hanya dengan memberikan $[a]$ sebagai *abductive solution* untuk q lalu menyimpannya ke *table*.
- *Query* berikutnya, s, harus memenuhi dua *subgoal* pada *body*-nya, yaitu dengan mengeksekusi q dan b. Karena q sudah pernah dieksekusi, solusi yang dihasilkan pada eksekusi q sebelumnya dapat digunakan kembali, menghasilkan *abductive context* $[a]$. Selanjutnya, $[a]$ yang dihasilkan dari *subgoal* q ditambahkan dengan *subgoal* berikutnya yaitu $[b]$ yang merupakan *abducible*, menghasilkan *abductive solution* $[a, b]$.
- Tidak jauh berbeda dengan *query* sebelumnya, *query* t akan mengekesekusi s dan q. Eksekusi dari *subgoal* s menghasilkan *abductive solution* yang sudah disimpan pada *table* yaitu $[a, b]$, sementara eksekusi dari q mengahasilkan *abductive solution* $[a]$ yang juga sudah disimpan pada *table*. Karena $[a] \subseteq [a, b]$, maka *query* t cukup memberikan jawaban $[a, b]$ sebagai *abductive solution*-nya.

Ilustrasi di atas menunjukkan bahwa $[a]$ sebagai *abductive solution* yang didapatkan dari *query* yang pertama q dapat digunakan pada *abductive context* dari dua *query* lainnya berikutnya (dicontohkan dengan $[b]$ pada *query* s dan $[a, b]$ pada *query* t). Dapat dibayangkan jika q memiliki *subgoal* dengan jumlah yang sangat besar, tanpa *tabling*, *cost* yang diperlukan baik dari segi ruang dan waktu akan menjadi sangat mahal.

TABDUAL terdiri dari dua fase. Fase pertama yaitu *program tansformation*, menghasilkan program *output* yang dapat digunakan pada fase berikutnya, yaitu *abduction* itu sendiri yang dapat dilakukan dengan memberikan *query*.

3.2 Transformasi Program

Pada Contoh 3.1 telah ditunjukkan dua unsur utama pada transformasi TABDUAL:

- *Abductive context*, membawa informasi mengenai *abductive solution* dari satu *subgoal* ke *subgoal* berikutnya pada *body* dari sebuah *rule*, juga dari *head* dari sebuah *rule* ke *body*-nya, melalui *input context* dan *output context*.
- *Tabled predicate*, menyimpan *abductive solution* dari predikat yang muncul sebagai sebuah *head* pada program untuk disimpan pada *table*, sehingga *abductive solution* yang sudah disimpan dapat dipergunakan kembali pada *abductive context* yang berbeda.

Transformasi program pada TABDUAL terdiri dari beberapa bagian, yaitu transformasi untuk melakukan *tabling* terhadap *abductive solution* (bagian 3.2.1), untuk melakukan *abduction* pada *goal* negatif (bagian 3.2.2), untuk menambahkan *abducible* pada suatu *abductive context* (bagian 3.2.3), dan untuk melakukan transformasi pada *query* (bagian 3.2.4).

3.2.1 *Tabling Abductive Solution*

Untuk merealisasikan ide yang ditunjukkan pada Contoh 3.1, TABDUAL melakukan transformasi terhadap setiap *rule* yang terdapat pada program menjadi dua buah *rule*. *Rule* pertama mendefinisikan *tabled predikat* yang digunakan untuk memperoleh *abductive solution*, dengan membawa *abducible* pada *rule* tersebut sebagai *input context* untuk melakukan *abduction* mulai dari *subgoal* pertama, satu per satu hingga ke *subgoal* terakhir, lalu menyimpan *abductive solution* yang didapat ke dalam *table*. Sementara itu, *rule* kedua hasil transformasi digunakan untuk mendapatkan *abductive solution* yang sudah disimpan pada *table* sehingga dapat dipergunakan kembali pada *abductive context* yang berbeda.

Transformasi untuk membentuk kedua *rule* di atas didefinisikan secara formal sebagai pada Definisi 3.1.

Untuk selanjutnya, $\bar{t}$ digunakan untuk menyatakan $[t_1,\ldots,t_n]$, $n \geq 0$. Untuk sebuah predikat p/n, $p(\bar{t})$ digunakan untuk menyatakan $p(t_1,\ldots,t_n)$. Secara khusus, $\bar{X}$ digunakan untuk menyatakan $[X_1,\ldots,X_n]$, $p(\bar{X})$ untuk menyatakan $p(X_1,\ldots,X_n)$, dan $p(\bar{X},Y,Z)$ untuk menyatakan $p(X_1,\ldots,X_n,Y,Z)$, dengan seluruh variabel yang disebutkan merupakan variabel yang berbeda-beda.

Definisi 3.1: Transformasi untuk *tabling abductive solution*. Diberikan sebuah *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$. H_r dan $\mathcal{B}_r$ berturut-turut menunjukkan *head* dan *body* dari sebuah *rule* $r \in P$. Himpunan $\mathcal{A}_r \subseteq \mathcal{B}_r$ menunjukkan himpunan *abducible* (positif ataupun negatif) pada $r \in P$, dan r' menunjukkan *rule* sedemikian sehingga $H_{r'} = H_r$ dan $\mathcal{B}_{r'} = \mathcal{B}_r \backslash \mathcal{A}_r$.

1. Untuk setiap *rule* $r \in P$ dengan r' yaitu $l(\hat{t}) \leftarrow L_1, \ldots, L_M$, didefinisikan $\tau'(r)$:

   $$l_{ab}(\bar{t}, E_m) \leftarrow \alpha(L_1), \ldots, \alpha(L_m).$$

   dengan α didefinisikan sebagai:

   $$\alpha(L_i) = \begin{cases} l_i(\bar{t}_i, E_{i-1}, E_i) & \text{untuk } L_i = l_i(\bar{t}_i) \\ not\_l_i(\bar{t}_i, E_{i-1}, E_i) & \text{untuk } L_i = not\ l_i(\bar{t}_i) \end{cases}$$

   dengan $1 \leq i \leq m$, E_1 adalah variabel baru, dan $E_0 = \mathcal{A}_r$. Dapat dilihat bahwa E_i disediakan *abductive context*.

2. Untuk setiap predikat p/n yang menjadi *head* dari sebuah *rule* pada P, didefinisikan $\tau^+(p)$:

   $$p(\bar{X}, I, O) \leftarrow p_{ab}(\bar{X}, E), produce\_context(O, I, E).$$

   dengan $produce\_context(O, I, E)$ adalah predikat sistem TABDUAL yang digunakan untuk membentuk O dengan menambahkan setiap $e \in E$ pada I, sekaligus melakukan pengecekan apakah I konsisten dengan E, yaitu apakah terdapat dua literal yang saling berlawanan pada I dan E.

3.2.2 *Abduction* pada *Goal* Negatif

Untuk melakukan *abduction* pada *goal* negatif, transformasi pada TABDUAL menerapkan *dual program transformation* pada ABDUAL [8]. Tujuan utama dari menggunakan *dual program transformation* yaitu untuk dapat memperoleh solusi dari *goal* negatif *not G* tanpa harus menegasikan seluruh *abductive solution* dari G.

Ide dari *dual program transformation* yaitu mendefinisikan, untuk setiap atom A dan himpunan *rule* mengenai A $\mathcal{R}_A$ pada program P, himpunan *dual rule* yang *head*-nya adalah *not_A*, sedemikian sehingga *not_A* bernilai *true* jika dan hanya jika A bernilai *false* sesuai dengan $\mathcal{R}_A$ menurut semantik yang digunakan oleh program P. Alih-alih menggunakan *goal* negatif *not A*, *dual program transformation* menggunakan atom *not_A* yang bersesuaian.

Dual program transformation membentuk dua buah jenis *rule* (*atau layer*) yang berbeda. *First layer* dari *dual program transformation*, atau *first layer dual rule*, dari sebuah predikat p pada program P merupakan *rule not_p* yang didefinisikan untuk mem-*falsify* negasi dari p, yaitu *not* p, sekaligus dengan membawa *input context* yang diberikan untuk melakukan *abduction* pada setiap *subgoal-subgoal*-nya sehingga menghasilkan *abductive solution* dari p. *Subgoal-subgoal* dari *first layer dual rule* itu sendiri merupakan *second layer dual rule* p^{*i} yang didefinisikan berdasarkan *rule-rule* mengenai p yang terdapat pada program, digunakan untuk mem-*falsify* tiap *rule* pada *body* dari *rule* mengenai p, tentunya setelah dinegasikan.

Tranformasi yang membentuk kedua *layer* dari *dual rule* didefinisikan secara formal pada Definisi 3.2.

Definisi 3.2: Transformasi untuk membentuk *dual rule*. Diberikan sebuah *abductive framework* $\langle P, \mathcal{AB}, \mathcal{IC} \rangle$. Misal $P^+ = P \cup \mathcal{IC}$.

1. Untuk setiap predikat p/n yang memiliki *rule* pada P^+ seperti berikut:

$$p(\bar{t}_1) \leftarrow L_{11}, \ldots, L_{1n_1}.$$
$$\vdots$$
$$p(\bar{t}_m) \leftarrow L_{m1}, \ldots, L_{mn_m}.$$

dengan $n_i \geq 0, 1 \leq i \leq m$:

(a) *First layer dual rule* didefinisikan sebagai $\tau^-(p)$:

$$not\_p(\bar{X}, T_0, T_m) \leftarrow p^{*1}(\bar{X}, T_0, T_1), \ldots, p^{*1}(\bar{X}, T_{m-1}, T_m).$$

dengan $T_i, i \leq i \leq m$ adalah variabel baru yang disediakan sebagai *abductive context*.

(b) *Second layer dual rule* didefinisikan sebagai $\tau^*(p) = \bigcup_{i=1}^{m} \tau^{*i}(p)$, dengan $\tau^{*i}(p)$ merupakan himpunan terkecil yang mengandung *rule-rule* seba-

gai berikut:

$$\begin{aligned} p^{*i}(\bar{X},I,I) &\leftarrow \bar{X} \neq \bar{t}_i. \\ p^{*i}(\bar{X},I,O) &\leftarrow \sigma(L_{i1},I,O). \\ &\vdots \\ p^{*i}(\bar{X},I,O) &\leftarrow \sigma(L_{in_i},I,O). \end{aligned}$$

dengan σ didefinisikan sebagai berikut:

$$\sigma(L_{ij},I,O) = \begin{cases} l_{ij}(\bar{t}_{ij},I,O) & \text{jika } L_i \text{ adalah } \textit{default} \text{ literal } \textit{not } l_{ij}(\bar{t}_{ij}) \\ & \text{atau } \textit{abducible} \text{ negatif } l^*_{ij}(\bar{t}_{ij}) \\ \textit{not\_}l_{ij}(\bar{t}_{ij},I,O) & \text{jika } L_i \text{ adalah atom } l_{ij}(\bar{t}_{ij}) \\ l^*_{ij}(\bar{t}_{ij},I,O) & \text{jika } L_i \text{ adalah } \textit{abducible} \text{ positif } l_{ij}(\bar{t}_{ij}) \end{cases}$$

Untuk kasus $p/0$, *rule* $p^{*i}(\bar{X},I,I) \leftarrow \bar{X} \neq \bar{t}_i$ dihilangkan karena $\bar{X}$ dan $\bar{t}_i$ merupakan $[]$.

2. Untuk setiap predikat r/n pada $P^+ (n \geq 0)$ yang tidak memiliki *rule*, didefinisikan $\tau^-(r)$:

$$\textit{not\_}r(\bar{X},I,I).$$

Secara khusus, jika $\mathcal{IC} = \emptyset$, didefinisikan $\tau^-(r) : \textit{not\_}\bot(I,I)$.

3.2.3 *Transformasi Abducible*

Abduction pada TABDUAL dilakukan dengan melakukan transformasi pada setiap *abducible* menjadi *rule* yang dapat menambahkan *abducible* pada *abductive context* yang sudah ada. Spesifikasi untuk melakukan transformasi terhadap *abducible* akan diberikan secara formal pada Definisi 3.3.

Definisi 3.3: Transformasi *abducible*. Diberikan sebuah *abductive framework* $\langle P,\mathcal{AB},\mathcal{IC}\rangle$. Untuk setiap $a/n \in \mathcal{AB}$, didefinisikan $\tau^\circ(a)$ yaitu himpunan terkecil yang mengandung *rule*:

$$\begin{aligned} a(X_1,\ldots,X_n,I,O) &\leftarrow \textit{insert\_abducible}(a(X_1,\ldots,X_n),I,O). \\ a^*(X_1,\ldots,X_n,I,O) &\leftarrow \textit{insert\_abducible}(a^*(X_1,\ldots,X_n),I,O). \end{aligned}$$

dengan $\textit{insert\_abducible}(A,I,O)$ merupakan predikat sistem TABDUAL yang

menambahkan *abducible A* ke *input context I* dan menghasilkan *output context O*. Predikat ini juga mempertahankan konsistensi dari *abductive context I* jika ditambahkan oleh *abducible A*.

Spesifikasi program transformation TABDUAL secara utuh akan diberikan pada Definisi 3.4.

Definisi 3.4: Transformasi TABDUAL. Diberikan *abductive framework* $\mathcal{F} = \langle P, \mathcal{AB}, \mathcal{IC} \rangle$, $\mathcal{P}$ adalah himpunan predikat yang terdapat pada P, dan $P^+ = P \cup \mathcal{IC}$. Menggunakan definisi-definisi transformasi sebelumnya didapatkan:

- $\tau'(\mathcal{F}) = \{\tau'(r) \mid r \in P\}$
- $\tau^+(\mathcal{F}) = \{\tau^+(p) \mid p \in \mathcal{P}$ yang memiliki *rule* pada $P\}$
- $\tau^-(\mathcal{F}) = \{\tau^-(p) \mid p \in \mathcal{P} \cup \{\bot\}\}$
- $\tau^*(\mathcal{F}) = \{\tau^*(p) \mid p \in \mathcal{P} \cup \{\bot\}$ yang memiliki *rule* pada $P\}$
- $\tau^\circ(\mathcal{F}) = \{\tau^\circ(a) \mid a \in \mathcal{AB}\}$

maka transformasi TABDUAL didefinisikan sebagai:

$$\tau(\mathcal{F}) = \tau'(\mathcal{F}) \cup \tau^+(\mathcal{F}) \cup \tau^-(\mathcal{F}) \cup \tau^*(\mathcal{F}) \cup \tau^\circ(\mathcal{F})$$

3.2.4 *Transformasi Query*

Sebagai konsekuensi dari transformasi TABDUAL, *query* terhadap program juga harus di-*transform*:

- *Goal* positif *G* cukup di-*transform* dengan menambahkan dua buah argumen untuk menyatakan *input context* dan *putput context*.
- *Goal* negatif *not G* di-*transform* dengan mengubah namanya menjadi *not_G* serta ditambahkan dua buah argumen untuk menyatakan *input context* dan *putput context*.

Selanjutnya, *query* yang diberikan juga harus memenuhi seluruh *integrity constraint* yang ada. Hal ini dapat dilakukan dengan menambahkan *goal not*$\_\bot/2$ yang menyatakan *dual rule* dari *integrity constraint*. Definisi 3.5 memberikan definisi formal untuk melakukan transformasi pada *query*.

Definisi 3.5: Transformasi *query*. Diberikan *abductive framework* $\mathcal{F} = \langle P, \mathcal{AB}, \mathcal{IC} \rangle$ dan query Q:

$$? - \ G_1, \ldots, G_m.$$

TABDUAL melakukan transformasi *query* Q menjadi $\Delta(Q)$:

$$? - \ \delta(G_1), \ldots, \delta(G_m), not\_\bot(T_m, O).$$

dengan δ didefinisikan sebagai:

$$\alpha(L_i) = \begin{cases} g_i(\bar{t}_i, T_{i-1}, T_i) & \text{jika } G_i = g_i(\bar{t}_i) \\ not\_g_i(\bar{t}_i, E_{i-1}, T_i) & \text{jika } G_i = not\ g_i(\bar{t}_i) \end{cases}$$

3.3 Aspek Implementasi

TABDUAL diimplementasikan menggunakan XSB Prolog [9], sehingga dapat memanfaatkan fitur-fitur yang sudah ada. Namun di sisi lain, sebagai konsekuensinya implementasi yang dibuat juga harus disesuaikan dengan *behavior* yang dimiliki XSB Prolog.

3.3.1 *Grounding Dualized Negated Subgoals*

Contoh 3.2. Misal terdapat *abductive framework* $\langle P_4, \{a/1\}, \mathcal{IC}_4 \rangle$ dengan program P_4 sebagai berikut:

$$q(1).$$
$$r(X) \leftarrow a(X).$$

dan $\mathcal{IC}_4$:

$$\bot \leftarrow q(X), r(X).$$

Transformasi TABDUAL akan menghasilkan program:

1. $q_{ab}(1,[\,]).$
2. $q(X,I,O) \leftarrow q_{ab}(X,E), produce\_context(O,I,E).$
3. $not\_q(X,I,O) \leftarrow q^{*1}(X,I,O).$
4. $q^{*1}(X,I,I) \leftarrow X \setminus = 1.$

5. $r_{ab}(X,[a(X)]).$
6. $r(X,I,O) \leftarrow r_{ab}(X,E), produce\_context(O,I,E).$
7. $not\_r(X,I,O) \leftarrow r^{*1}(X,I,O).$
8. $r^{*1}(X,I,O) \leftarrow X \setminus = \_.$
9. $r^{*1}(X,I,O) \leftarrow a^{*}(X,I,O).$

10. $not\_\bot(I,O) \leftarrow \bot^{*1}(I,O).$
11. $\bot^{*1}(I,O) \leftarrow not\_q(X,I,O).$
12. $\bot^{*1}(I,O) \leftarrow not\_r(X,I,O).$

Lalu diberikan *query* $q(1)$, yang di-*transform* menjadi:

$$?-q(1,[\,],T), not\_\bot(T,O).$$

progam akan memberikan *abductive solution* $[a^{*}(X)]$ (tidak *ground*), tidak sesuai dengan *abductive solution* yang seharusnya diberikan yaitu $[a^{*}(1)]$. Kesalahan ini disebabkan karena ketika program mencari solusi, *fail*-nya *subgoal* dari *rule* 11 $not\_q(X,I,O)$ menyebabkan program mengekesekusi $not\_r(X,I,O)$ pada *rule* 12 sebagai alternatif lain dari *goal* $\bot^{*1}(I,O)$, namun dengan X yang tidak *ground* karena memang belum diinstansiasi.

Untuk memperbaiki kesalahan ini, *dualized negated subgoal* (pada contoh di atas yaitu *subgoal* dari *rule* 4, 8, 9, 11, 12) perlu untuk di-*ground*-kan terlebih dahulu. *Grounding subgoal* ini dapat dilakukan dengan tetap mengikutsertakan pada *second layer dual rule* literal-literal positif yang pada *rule* aslinya terdapat sebelum predikat yang sedang di-*dual*-kan. Dengan begitu, *rule* 12 pada program di atas akan menjadi:

$$\bot^{*1}(I,O) \leftarrow q(X,I,T), not\_r(X,T,O).$$

Adanya $q(X,I,T)$ sebagai *subgoal* sebelum $not\_r(X,T,O)$ pada *rule* 12 menyebabkan X akan diinstansiasi terlebih dahulu menggunakan *rule* 4 sehingga *query* $q(1)$ menghasilkan *abductive solution* yang sesuai yaitu $[a^*(1)]$

3.3.2 *Non-Ground Negative Goal*

Contoh 3.3. Misal terdapat *abductive framework* $\langle P_5, \{a/1\}, \emptyset \rangle$ dengan program P_5 sebagai berikut:

$$p(1) \leftarrow a(1).$$
$$p(2) \leftarrow a(2).$$

Query $p(X)$ pada program diatas akan memberikan *abductive solution* yang sesuai yaitu $[a(1)]$ untuk $X = 1$ dan $[a(2)]$ untuk $X = 2$. Sementara itu, *query* *not* $p(X)$ memberikan hasil yang tidak sesuai. *Abuctive solution* seharusnya yang dihasilkan yaitu $[a^*(1), a^*(2)]$ untuk apa pun instansiasi X, namun TABDUAL memberikan *abductive solution* $[a*(1)]$ untuk $X = 1$.

Berikut ini merupakan definisi predikat $not\_p/3$:

$$\begin{aligned}
&1.\ not_p(X,I,O) && \leftarrow p^{*1}(X,I,T), p^{*2}(X,T,O).\\
&2.\ p^{*1}(X,I,O) && \leftarrow X \setminus = 1.\\
&3.\ p^{*1}(X,I,O) && \leftarrow a^*(1,I,O).\\
&4.\ p^{*2}(X,I,O) && \leftarrow X \setminus = 2.\\
&5.\ p^{*2}(X,I,O) && \leftarrow a^*(2,I,O).
\end{aligned}$$

dan query *not* $p(X)$ yang dihasilkan oleh transformasi TABDUAL:

$$?- not\_p(X,[\,],N), not\_\bot(N,O).$$

Goal $not\_p(X,[\,],N)$ pertama-tama mengeksekusi $p^{*1}(X,I,O)$ dengan $I = [\,]$, lalu dengan menggunakan *rule* 3 berhasil memberikan $T = [a^*(1)]$ dengan X diinstansiasi dengan 1. Selanjutnya, *subgoal* kedua yaitu $p^{*2}(X,I,O)$, dieksekusi dengan instansiasi X dan T yang sama ($X = 1$ dan $T = [a^*(1)]$), dan dengan menggunakan *rule* 4, $p^{*2}(X,I,O)$ berhasil memberikan $O = [a^*(1)]$, tetap dengan X yang diinstansiasi dengan 1. $[a^*(1)]$ yang merupakan *abductive solution* dari *goal* pertama pada *query* dibawa sebagai N ke *goal* berikutnya yaitu $not\_\bot(N,O)$. Dikarenakan tidak terdapat *integrity constraint*, *subgoal* $not\_\bot(N,O)$ menghasilkan O yang sama dengan N, menghasilkan $[a^*(1)]$ sebagai *abductve solution* dari *query* *not* $p(X)$.

Dapat dilihat bahwa ketidaksesuaian *abductive solution* yang dihasilkan terjadi karena terdapat *shared variable* pada kedua *subgoal* dari $not\_p(X,I,O)$, yaitu X, menyebabkan $p^{*1}(X,I,O)$ dan $p^{*2}(X,I,O)$ dieksekusi dengan X yang sama. Seharusnya, kedua *subgoal* ini dieksekusi dengan X yang saling independen satu sama lain. Ketidaksesuaian ini dapat diatasi dengan mengubah definisi dari predikat $not\_p/3$ menjadi:

$$not\_p(X,I,O) \leftarrow copy\_term([X],[X_1]), p^{*1}(X_1,I,T), \\ copy\_term([X],[X_2])p^{*2}(X_2,T,O).$$

dengan predikat $copy\_term/2$ merupakan predikat *built-in* yang disediakan oleh Prolog. $copy\_term/2$ ditambahkan sebelum masing-masing *subgoal* dari *first layer dual rule* (dalam kasus ini sebelum $p^{*1}(X,I,O)$ dan $p^{*2}(X,I,O)$), digunakan untuk memberikan *varian* baru terhadap X agar dapat digunakan secara independen.

3.3.3 Transformasi Fakta

TABDUAL melakukan transformasi pada seluruh predikat yang ada pada program, termasuk predikat yang sebenarnya hanya berupa fakta yang bahkan tidak menginduksikan *abduction* sama sekali. Oleh karena itu, transformasi yang lebih sederhana dapat diterapkan pada predikat-predikat yang hanya berupa fakta.

Misalkan terdapat predikat $q/1$ yang hanya berupa fakta seperti berikut:

$$q(1). \qquad q(2). \qquad q(3).$$

Rule transformasi untuk $q/1$ yang telah didefinisikan pada Definisi 3.1, yaitu $q_{ab}/1$ dan $q/3$ dapat digantikan dengan satu buah *rule* berikut:

$$q(X,I,I) \leftarrow q(X).$$

dan *dual rule* untuk $q/1$ yang telah didefinisikan pada Definisi 3.2 juga cukup digantikan dengan satu buah *rule* berikut:

$$not\_q(X,I,I) \leftarrow not\ q(X).$$

Transformasi untuk fakta seperti predikat $q/1$ bersifat independen terhadap jumlah fakta yang ada, sehingga dapat dipisahkan dari keseluruhan program dan dikelompokkan menjadi bagian yang bersifat *non-abductive* untuk diproses secara khusus. Pada TABDUAL, bagian program yang bersifat *non-abductive* dapat dip-

isahkan dengan *indentifier beginProlog* dan *endProlog*. Program yang terdapat di-antara kedua identifier ini tidak akan di-*transform* oleh TABDUAL, melainkan hanya akan dibaca dan ditulis ulang ke program *output*.

3.3.4 *Dual Transformation by Need*

Transformasi yang dilakukan oleh TABDUAL bersifat *once for all*, yaitu men-*transform* seluruh predikat, atom, dan *abducible* yang terdapat pada program, tanpa pertimbangan apakah predikat/atom/*abducible* yang di-*transform* akan digunakan pada fase *abduction* atau tidak. Praktis, transformasi seperti ini harus dihindari dikarenakan *cost* yang diperlukan terlalu besar untuk melakukan transformasi yang belum tentu digunakan. Salah satu solusi untuk mengatasi permasalahan ini yaitu dengan melakukan transformasi *dual rule* secara *by need*, yaitu dengan tidak membentuk *dual rule* pada fase transformasi, melainkan pada saat fase *abduction* ketika *dual rule* tersebut dibutuhkan untuk dieksekusi. Program *output* yang dihasilkan tetap mengandung *first layer dual rule*, namun tidak *second layer dual rule*. *Second layer dual rule* akan dibentuk pada fase *abduction* dengan bantuan predikat sistem TABDUAL yang dikhususkan untuk membentuk definisi konkrit dari *second layer dual rule* ini.

BAB 4
IMPLEMENTASI

Pada bab ini penulis akan menjelaskan implementasi TABDUAL yang dibuat oleh penulis.

4.1 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 ???.

4.2 Spesifikasi TABDUAL

4.2.1 Tahapan TABDUAL

Seperti yang sudah dijelaskan pada bagian 3.1, 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.

4.2.2 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*.

4.2.3 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 4.1: 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 4.2: Contoh program *input* yang tidak diterima TABDUAL

4.3 Pra Transformasi

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

4.3.1 *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.

4.3.1.1 *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 4.3: 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*.

4.3.1.2 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 4.4: 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*.

4.3.1.3 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 4.5: 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.

4.3.1.4 *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 4.6: 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.

4.3.2 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 4.7: 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.

4.3.3 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 4.8: 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.

4.3.4 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 4.9: 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*.

4.3.5 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 4.10: 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.

4.3.6 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 4.11: 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.

4.3.7 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 4.12: 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.

4.3.8 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*.

4.4 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 4.13: 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.

4.4.1 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 4.14: 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*.

4.4.2 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 4.15: 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.

4.4.3 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 4.16: 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*.

4.4.4 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 4.17: 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.

4.5 *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.

4.5.1 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*).

4.5.2 Transformasi *Query*

Pada bagian ??? telah dijelaskan bahwa *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 4.18: 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.

4.6 *Answer Subsumption*

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

4.6.1 *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 4.19: *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

4.7 Predikat Sistem

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

4.7.1 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 4.20: 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*.

4.7.2 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 4.21: 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*.

4.7.3 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 4.22: 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*.

4.7.4 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 4.23: 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 4.24: 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 4.25: 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 4.26: 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.

4.8 Pengujian

4.8.1 INI BELOM

Berwarna!

Kode 4.27: 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
...
```

4.8.2 INI JUG BELOM

Contoh skrip yang dimasukkan pada *form* yang disediakan dapat dilihat pada kode 4.28.

Kode 4.28: 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 5
EVALUASI DAN ANALISIS

5.1 Hasil Pengujian

5.1.1 Hasil Pengujian Kasus Uji 1

Tabel lain. Hasil tersebut dapat dilihat pada tabel 5.1.

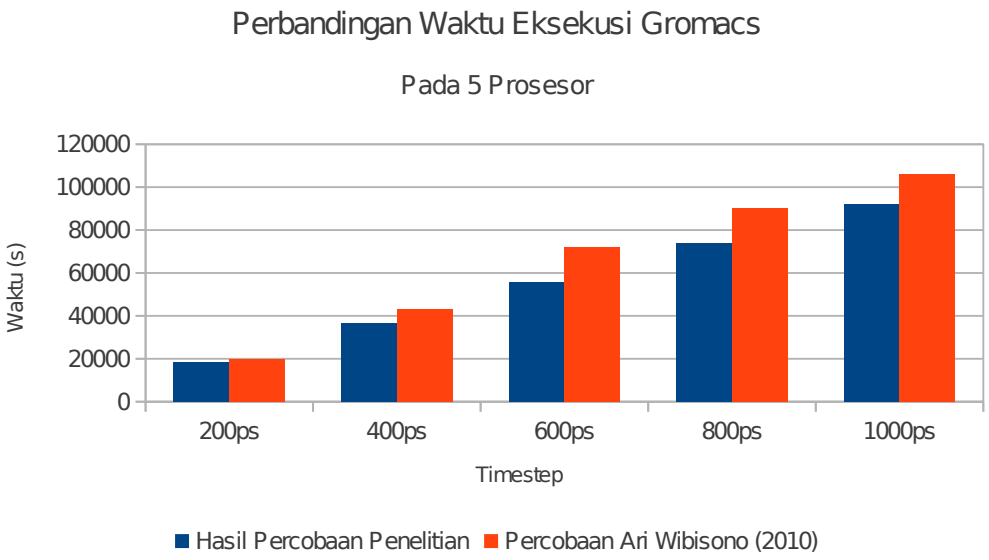
Tabel 5.1: Hasil pengujian menggunakan gromacs

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

5.2 Evaluasi Hasil Kasus Uji

5.2.1 Evaluasi Kasus Uji 1

Tabel 5.1 menunjukkan hasil uji coba pada penelitian ini. Gambar 5.1 menunjukkan perbandingan waktu eksekusi pada aplikasi x dengan jumlah prosesor sebanyak 5 buah.



Gambar 5.1: Perbandingan waktu eksekusi x untuk 5 prosesor

BAB 6
PENUTUP

Pada bab terakhir ini,

6.1 Kesimpulan

6.2 Saran

DAFTAR REFERENSI

[1] Maarten H Van Emden and Robert A Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM (JACM)*, 23(4):733–742, 1976.

[2] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

[3] Jose Julio Alferes, Luis Moniz Pereira, J Siekmann, and JG Carbonell. *Reasoning with logic programming*, volume 1111. Springer Heidelberg, 1996.

[4] Allen Van Gelder, Kenneth A Ross, and John S Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3): 619–649, 1991.

[5] Halina Przymusinska and Teodor Przymusinski. Semantic issues in deductive databases and logic programs. In *Formal Techniques in Artificial Intelligence*. Citeseer, 1990.

[6] WV Quine. Collected papers of charles sanders peirce.–volume ii: Elements of logic charles hartshorne paul weiss. *History of Science*, 19(1), 1933.

[7] Antonis C Kakas, Robert A Kowalski, and Francesca Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[8] José Júlio Alferes, Luís Moniz Pereira, and Terrance Swift. Abduction in well-founded semantics and generalized stable models via tabled dual programs. *Theory and Practice of Logic Programming*, 4(04):383–428, 2004.

[9] Terrance Swift and David S Warren. Xsb: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming*, 12(1-2):157–187, 2012.

[10] Terrance Swift and David S Warren. Tabling with answer subsumption: Implementation, applications and performance. In *European Workshop on Logics in Artificial Intelligence*, pages 300–312. Springer, 2010.

[11] Terrance Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3):201–240, 1999.

[12] A Saptawijaya and LUÍS MONIZ Pereira. Tabdual: a tabled abduction system for logic programs. *IfCoLog Journal of Logics and their Applications*, 2(1): 69–123, 2015.

LAMPIRAN

LAMPIRAN 1 : KODE SUMBER