

LAPORAN TUGAS AKHIR

**Implementasi Algoritma *Principal Type* dan  
Algoritma *Type Inhabitant* dari *Type Theory* TA-Lambda  
Menggunakan *Definite Clause Grammar* dalam PROLOG**



Oleh:

**Ario Santoso**

**1204007038**

FAKULTAS ILMU KOMPUTER

UNIVERSITAS INDONESIA

DEPOK

2008

# Halaman Persetujuan

## Judul Tugas Akhir :

Implementasi Algoritma *Principal Type* dan  
Algoritma *Type Inhabitant* dari *Type Theory* TA-Lambda  
Menggunakan *Definite Clause Grammar* dalam PROLOG

**Nama** : Ario Santoso

**NPM** : 1204007038

Laporan Tugas Akhir ini telah diperiksa dan disetujui.

Depok, Juli 2008

L.Y. Stefanus

Pembimbing Tugas Akhir

# Abstrak

Studi tentang *type theory* telah memberikan kontribusi penting dalam dunia ilmu komputer, terutama dalam rekayasa perangkat lunak, basis data, *computational linguistics*, desain bahasa pemrograman, *automated theorem proving*, *high performance compiler* dan keamanan jaringan komputer. Tugas akhir ini berfokus pada varian *type theory* yang disebut *Type Assignment* (TA). Kontribusi dari tugas akhir ini terdiri dari tiga hal pokok. Pertama, algoritma *Principal Type* (PT) dan pencarian *type inhabitant* diimplementasikan dalam PROLOG dengan menggunakan Definite Clause Grammar (DCG). Hasil implementasi ini dapat dipakai untuk mencari tipe dari sebuah  $\lambda$ -term dan juga sebaliknya, mencari *inhabitant* (berupa  $\lambda$ -term) dari sebuah tipe. Kedua, seluk-beluk TA, terutama algoritma PT dan pencarian *type inhabitant*, dipaparkan dengan bahasa yang lebih mudah dimengerti dibandingkan literatur yang sudah ada. Ketiga, sebuah antarmuka grafis dibangun untuk memudahkan *user* dalam menggunakan (mencoba) kedua algoritma tersebut. Dengan demikian, *software* ini bisa digunakan sebagai *test bed* untuk mempelajari TA, maupun untuk bereksperimen dalam *type theory*.

xv + 184 hal.; 31 gambar.; 1 tabel.; Lampiran A-P; Bibliografi: 13 (1927-2007)

# Kata Pengantar

Puji syukur penulis panjatkan ke hadirat Allah SWT karena atas rahmat dan pertolongan-Nya-lah penulis dapat menyelesaikan tugas akhir ini. Laporan ini berisi penjabaran tentang tugas akhir yang dilakukan oleh penulis.

Dalam kesempatan ini penulis juga hendak menyampaikan rasa terima kasih yang sebesar - besarnya kepada semua pihak yang telah banyak membantu penulis selama pelaksanaan tugas akhir ini, dan juga selama penulis menempuh pendidikannya di Fakultas Ilmu Komputer Universitas Indonesia, terutama kepada:

1. Keluarga penulis yang telah memberikan dukungan serta doa.
2. Bapak L.Y. Stefanus, selaku pembimbing tugas akhir penulis, yang senantiasa membimbing penulis dalam mengerjakan tugas akhir ini.
3. Bapak T.Basaruddin, selaku pembimbing akademik penulis, yang senantiasa membimbing penulis selama mengenyam pendidikan di Fakultas Ilmu Komputer Universitas Indonesia.
4. Semua dosen Fakultas Ilmu Komputer Universitas Indonesia, yang telah membimbing penulis selama menempuh pendidikan di Fakultas Ilmu Komputer Universitas Indonesia.
5. Pembuat LaTeX, yang memungkinkan saya dapat menulis dokumen laporan tugas akhir ini dengan nyaman.
6. Teman-teman lab 1237, Alidz, Arya, Pandu, dan Aji. Terima kasih atas segala dukungannya.
7. Aniz, Rama, Akin, Imam, Satrio, selaku teman seperjuangan penulis dalam mengerjakan skripsi. Walaupun kita semua berbeda bidang, tapi kita tetap punya

satu tujuan. Special thanks untuk Dewi K.S.

8. Rahmad Mahendra, selaku teman penulis yang senantiasa mengajarkan teknik-teknik penulisan kepada penulis, dan juga teman berdiskusi penulis.
9. Arfan, Desmond, Mimi, Eliza dan Franky, selaku rekan penulis yang menjadi teman berdiskusi dan memberikan masukan kepada penulis.
10. Andreas dan Rozie 2003, selaku teman penulis yang mengajarkan menggunakan LaTeX dan membantu mencari solusi ketika terjadi permasalahan berkaitan dengan LaTeX.
11. Richard, Wahyu Mirza, Daniel Cahyadi, Rado, Angky, dan Albert, selaku teman-teman penulis yang selalu memberikan dukungannya.
12. Rekan - rekan SUN Microsystem, komunitas Java User Group Indonesia (JUGI) dan komunitas Open Solaris User Group Indonesia (OSUGI), khususnya pada Pak Harry, Pak Aan, Pak Adhari, Pak Junus, Thomas W., Joshua, dan Mbak Efni.
13. Semua teman-teman fasilkom UI khususnya angkatan 2004 untuk dukungannya dalam pengerjaan tugas akhir ini.
14. Semua pihak yang telah membantu dan mendukung penulis dalam penyelesaian Tugas Akhir ini.

Penulis sadar bahwa Laporan ini masih memiliki berbagai kekurangan dan keterbatasan. Oleh karena itu penulis menerima dengan tangan terbuka semua kritik dan saran terhadap Laporan ini. Namun demikian, penulis berharap semoga Laporan ini bisa bermanfaat bagi pembaca dalam memberikan wawasan baru.

Depok, Juli 2008

Ario Santoso

# Daftar Isi

Lembar Persetujuan	ii
Abstrak	iii
Kata Pengantar	iv
Daftar Isi	vi
Daftar Gambar	xiii
Daftar Tabel	xv
<b>1 Pendahuluan</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Tujuan . . . . .	3
1.3 Ruang Lingkup . . . . .	4
1.4 Metodologi Penelitian . . . . .	4
1.5 Sistematika Penulisan Laporan . . . . .	5
<b>2 <i>Definite Clause Grammar</i> (DCG)</b>	<b>7</b>
2.1 Context Free Grammar (CFG) . . . . .	7
2.2 Pengertian DCG . . . . .	9
2.3 Kemampuan dan Fitur DCG . . . . .	10
2.3.1 Merepresentasikan CFG . . . . .	10
2.3.2 Pemberian Argumen Pada Simbol <i>Non-Terminal</i> . . . . .	16
2.3.3 Pemanggilan Predikat PROLOG . . . . .	21

<b>3</b>	<b><i>Type-Free <math>\lambda</math>-Calculus</i></b>	<b>23</b>
3.1	Penjelasan Singkat <i><math>\lambda</math>-Calculus</i> . . . . .	23
3.2	Definisi dan Struktur <i><math>\lambda</math>-Term</i> . . . . .	25
3.2.1	Definisi <i><math>\lambda</math>-Term</i> . . . . .	25
3.2.2	Notasi Dalam <i><math>\lambda</math>-Term</i> . . . . .	26
3.2.3	<i>Length</i> (Panjang) <i><math>\lambda</math>-Term</i> . . . . .	26
3.2.4	<i>Subterm</i> Dari <i><math>\lambda</math>-term</i> . . . . .	27
3.2.5	Posisi Dalam <i><math>\lambda</math>-Term</i> . . . . .	27
3.2.6	Pohon Konstruksi Dari <i><math>\lambda</math>-term</i> . . . . .	28
3.2.7	Definisi <i>Body</i> , <i>Scope</i> dan <i>Covering Abstractor</i> dalam <i><math>\lambda</math>-Term</i> . . . . .	29
3.2.8	<i>Free</i> dan <i>Bound Variable</i> dalam <i><math>\lambda</math>-Term</i> . . . . .	30
3.2.9	<i>Bound-Variable Clash</i> . . . . .	31
3.2.10	<i>Closed Term (Combinator)</i> . . . . .	31
3.3	Operasi Pada <i><math>\lambda</math>-Term</i> . . . . .	32
3.3.1	Substitusi . . . . .	32
3.3.2	<i><math>\alpha</math>-Conversion</i> (Pengubahan <i>Bound Variables</i> ) . . . . .	32
3.3.3	Reduksi $\beta$ ( <i><math>\beta</math>-reduction</i> ) . . . . .	33
3.3.3.1	Definisi <i><math>\beta</math>-redex</i> , <i><math>\beta</math>-contraction</i> , <i><math>\beta</math>-contracts</i> , <i>contractum</i> . . . . .	33
3.3.3.2	Definisi <i><math>\beta</math>-reduction</i> . . . . .	33
3.3.3.3	Definisi <i><math>\beta</math>-conversion</i> , <i><math>\beta</math>-equal</i> dan <i><math>\beta</math>-expansion</i> . . . . .	34
3.3.3.4	<i><math>\beta</math>-normal form</i> . . . . .	34
3.3.3.5	Struktur <i><math>\beta</math>-normal form</i> . . . . .	34
3.3.4	Reduksi $\eta$ ( <i><math>\eta</math>-reduction</i> ) . . . . .	35
3.3.4.1	Definisi <i><math>\eta</math>-reduction</i> . . . . .	35
3.3.4.2	Definisi <i><math>\eta</math>-family</i> . . . . .	35
3.3.4.3	<i><math>\eta</math>-normal form</i> . . . . .	35
3.3.5	Reduksi $\beta\eta$ ( <i><math>\beta\eta</math>-reduction</i> ) . . . . .	36
3.3.5.1	Definisi <i><math>\beta\eta</math>-reduction</i> . . . . .	36
3.3.5.2	<i><math>\beta\eta</math>-normal form</i> . . . . .	36
<b>4</b>	<b><i>Type-Assignment</i> pada <i><math>\lambda</math>-term</i> (<math>\mathbf{TA}_\lambda</math>)</b>	<b>37</b>
4.1	Pengertian <i>Type</i> (Tipe) . . . . .	37

4.1.1	Definisi <i>Type</i> (Tipe)	38
4.1.2	Notasi Penulisan Tipe	38
4.1.3	Beberapa Definisi Seputar Tipe	39
4.2	Sistem $TA_\lambda$	39
4.2.1	Definisi <i>Type-Assignment</i>	39
4.2.2	Definisi <i>Type-Context</i>	40
4.2.3	Definisi Formula $TA_\lambda$	41
4.2.3.1	Notasi Penulisan Formula $TA_\lambda$	41
4.2.4	Aksioma dan Aturan Deduksi dalam $TA_\lambda$	41
4.2.4.1	Aksioma dalam $TA_\lambda$	41
4.2.4.2	Aturan Deduksi dalam $TA_\lambda$	42
4.2.5	Deduksi $TA_\lambda$ ( $\nabla$ )	42
4.2.5.1	Contoh Deduksi $TA_\lambda$ ( $\nabla$ )	43
4.2.6	<i>Deducibility</i> dalam $TA_\lambda$	44
4.3	Teorema Konstruksi Subjek ( <i>Subject Construction Theorem</i> ) dalam $TA_\lambda$	44

**5 *Principal-Type (PT) Algorithm: Algoritma Pemberian Tipe pada  $\lambda$ -term*** **47**

5.1	Substitusi Tipe	48
5.1.1	Definisi Substitusi Tipe	48
5.1.2	Notasi Penulisan Substitusi Tipe	49
5.1.3	Istilah dan Definisi Berkaitan dengan Konsep Substitusi Tipe	49
5.1.3.1	Komponen Substitusi	49
5.1.3.2	Domain dan Range Dalam Substitusi Tipe	49
5.1.3.3	Substitusi Tunggal dan Kosong	50
5.1.3.4	Substitusi $\mathbf{s}$ yang Terbatas pada $\mathbf{V}$ ( $\mathbf{s} \upharpoonright \mathbf{V}$ )	50
5.1.3.5	Substitusi <i>Variable</i> dengan <i>Variable</i>	50
5.1.4	Substitusi Tipe Pada Barisan Tipe, <i>Context</i> , Formula $TA_\lambda$ , dan deduksi ( $\nabla$ )	50
5.1.5	Gabungan ( <i>Union</i> ) Substitusi Tipe	51
5.1.6	Komposisi Substitusi Tipe	51
5.2	Definisi <i>Principal Type</i> (PT)	52

5.3	Definisi <i>Principal Pair</i> . . . . .	53
5.4	Definisi <i>Principal Deduction</i> . . . . .	53
5.5	<i>Common Instance</i> dan <i>Most General Common Instance</i> (MGCI) . . . . .	53
5.6	Unifikasi . . . . .	54
5.6.1	<i>Unifier</i> . . . . .	55
5.6.2	<i>Most General Unifier</i> dan <i>Most General Unification</i> . . . . .	55
5.6.3	Algoritma Unifikasi . . . . .	55
5.6.3.1	<i>Comparison Procedure</i> . . . . .	57
5.6.3.2	Contoh Penerapan Algoritma Unifikasi . . . . .	58
5.7	Ide (Prinsip) Dasar untuk Bagian Inti dari Algoritma <i>Principal Type</i> (PT) . . . . .	59
5.8	Penjelasan Algoritma <i>Principal Type</i> (PT) . . . . .	62
5.9	Contoh Penerapan Algoritma PT . . . . .	67
<b>6</b>	<b>Implementasi Algoritma <i>Principal Type</i></b> . . . . .	<b>73</b>
6.1	Bentuk Representasi Data dalam Program . . . . .	73
6.1.1	Representasi $\lambda$ -Term dalam Program . . . . .	73
6.1.2	Representasi Tipe dalam Program . . . . .	75
6.1.3	Representasi substitusi Tipe ( <i>Type-Substitution</i> ) dalam Program . . . . .	76
6.1.4	Representasi <i>Type-Assignment</i> dalam Program . . . . .	77
6.1.5	Representasi <i>Type-Context</i> ( $\Gamma$ ) dalam Program . . . . .	78
6.1.6	Representasi <i>TA<math>\lambda</math>-Formula</i> dalam Program . . . . .	78
6.1.7	Representasi <i>Deduction Tree</i> dalam Program . . . . .	79
6.2	Antarmuka Program . . . . .	79
6.3	Alur Kerja Program . . . . .	80
6.3.1	Bagian <i>Input</i> . . . . .	81
6.3.2	Bagian Pemrosesan . . . . .	82
6.3.3	Bagian <i>Output</i> . . . . .	82
6.4	Implementasi <i>Parser</i> untuk $\lambda$ -Term . . . . .	87
6.5	Predikat Pendukung untuk Implementasi Algoritma Unifikasi dan <i>Principal Type Algorithm</i> . . . . .	91
6.6	Implementasi Algoritma Unifikasi ( <i>Unification Algorithm</i> ) . . . . .	99

6.6.1	<i>Comparison Procedure</i> . . . . .	101
6.7	Optimisasi <i>Principal Type Algorithm</i> . . . . .	105
6.8	Implementasi <i>Principal Type Algorithm</i> . . . . .	106
6.9	Uji Coba Hasil Implementasi Algoritma PT . . . . .	121
<b>7</b>	<b><i>Typed Term</i></b> . . . . .	<b>123</b>
7.1	Definisi <i>Typed Term</i> . . . . .	124
7.2	Definisi <i>Type-Erasure</i> ( $M^r$ ) . . . . .	125
<b>8</b>	<b>Algoritma Pencarian <i>Type Inhabitant</i></b> . . . . .	<b>127</b>
8.1	<i>Inhabitant</i> . . . . .	128
8.1.1	<i>Typed</i> dan <i>Untyped Inhabitant</i> . . . . .	128
8.1.2	$\beta$ - <i>Normal Inhabitant</i> . . . . .	128
8.1.3	$\beta\eta$ - <i>Normal Inhabitant</i> . . . . .	128
8.1.4	<i>Lemma</i> tentang <i>Typed</i> dan <i>Untyped Inhabitant</i> . . . . .	128
8.1.5	<i>Principal Inhabitant</i> . . . . .	129
8.1.6	Kardinalitas . . . . .	129
8.2	Struktur dari <i>typed <math>\beta</math>-nf</i> . . . . .	130
8.3	<i>Depth</i> dari <i>Typed</i> atau <i>Untyped <math>\beta</math>-nf</i> . . . . .	130
8.4	<i>Long Typed <math>\beta</math>-nf</i> . . . . .	131
8.5	Ide Dasar dan Contoh dari Pencarian <i>Inhabitant</i> . . . . .	132
8.5.1	Ulasan (Uraian) struktur <i>long typed <math>\beta</math>-nf</i> [Hin97] . . . . .	132
8.5.2	Contoh pada Tipe $\tau$ dengan $\#(\tau) = 1$ [Hin97] . . . . .	133
8.5.3	Contoh pada Tipe $\tau$ dengan $\#(\tau) = m$ [Hin97] . . . . .	135
8.5.4	Contoh pada Tipe $\tau$ dengan $\#(\tau) = 0$ [Hin97] . . . . .	136
8.5.5	Contoh pada Tipe $((a \rightarrow b) \rightarrow a) \rightarrow a$ ( <i>Pierce law</i> ) [Hin97] . . . . .	137
8.6	<i>NF-scheme</i> . . . . .	137
8.6.1	Definisi <i>NF-Scheme</i> . . . . .	137
8.6.2	<i>Proper NF-Scheme</i> dan <i>Closed NF-Scheme</i> . . . . .	138
8.6.3	<i>Typed NF-Scheme</i> . . . . .	139
8.6.4	<i>Long Typed NF-Scheme</i> . . . . .	139
8.7	Algoritma Pencarian <i>Type Inhabitant</i> . . . . .	139

8.7.1	Contoh Penggunaan Algoritma Pencarian <i>Type Inhabitant</i> dan Penjelasannya . . . . .	143
8.7.2	Contoh lain Penggunaan Algoritma Pencarian <i>Type Inhabitant</i> .	144
<b>9</b>	<b>Implementasi Algoritma Pencarian <i>Type Inhabitants</i></b>	<b>147</b>
9.1	Bentuk Representasi Data dalam Program . . . . .	147
9.1.1	Representasi $\lambda$ - <i>Term</i> dalam Program . . . . .	147
9.1.2	Representasi Tipe dalam Program . . . . .	148
9.1.3	Representasi Himpunan $\mathcal{A}(\tau, d)$ dalam Program . . . . .	148
9.1.4	Representasi <i>NF-scheme</i> dalam Program . . . . .	149
9.1.5	Representasi <i>Suitable Replacement</i> dalam Program . . . . .	149
9.1.6	Representasi Pasangan <i>NF-Scheme</i> dan <i>Meta-Variabel</i> -nya dalam Program . . . . .	150
9.2	Antarmuka Program . . . . .	150
9.3	Alur Kerja Program . . . . .	150
9.3.1	Bagian <i>Input</i> . . . . .	152
9.3.2	Bagian Pemrosesan . . . . .	152
9.3.3	Bagian <i>Output</i> . . . . .	153
9.4	Implementasi <i>Parser</i> untuk Tipe . . . . .	156
9.5	Predikat Pendukung untuk Implementasi Algoritma Pencarian <i>Type Inhabitant</i> . . . . .	159
9.6	Implementasi Algoritma Pencarian <i>Type Inhabitant</i> . . . . .	165
9.7	Uji Coba Hasil Implementasi Algoritma Pencarian <i>Type Inhabitant</i> . .	177
<b>10</b>	<b>Kesimpulan dan Saran</b>	<b>179</b>
10.1	Kesimpulan . . . . .	179
10.2	Saran . . . . .	180
	<b>Daftar Pustaka</b>	<b>183</b>
	<b>A Tampilan Program</b>	<b>185</b>
	<b>B Code Implementasi <i>Parser</i> <math>\lambda</math>-term</b>	<b>188</b>

C	<i>Code Implementasi Parser Untuk Tipe dari <math>\lambda</math>-term</i>	193
D	Rangkuman Uji Coba Implementasi Algoritma <i>Principal Type</i>	197
E	Rangkuman Ujicoba Implementasi Algoritma Pencarian <i>Type Inhabitant</i>	200
F	<i>Flow Chart</i> Algoritma <i>Principal Type</i>	214
G	<i>Flow Chart</i> Algoritma Pencarian <i>Type Inhabitant</i>	225
H	<i>Code</i> Main.java	229
I	<i>Code</i> Implementasi PAlgorithmPanel.java	239
J	<i>Code</i> Implementasi TypeInhabitantFinderPanel.java	247
K	<i>Code</i> Implementasi TypeTextField.java	258
L	<i>Code</i> Implementasi LambdaTermTextField.java	262
M	<i>Code</i> Implementasi PDFViewer.java	266
N	<i>Code</i> Implementasi SplashScreen	275
O	<i>Code</i> Implementasi TEX2PDFConverter.java	280
P	<i>Code</i> Implementasi JavaPrologInterface.java	282

# Daftar Gambar

2.1	<i>Parse tree</i> dari kalimat "the cat scares the mouse" . . . . .	18
2.2	<i>Parse tree</i> dari rule <i>grammar</i> "s $\rightarrow$ p, q, r" [Bra01] . . . . .	19
3.1	Pembentukan pohon konstruksi untuk sebuah <i>term-variable</i> . . . . .	28
3.2	Pembentukan pohon konstruksi untuk sebuah aplikasi. . . . .	28
3.3	Pembentukan pohon konstruksi untuk sebuah abstraksi. . . . .	29
3.4	<i>Construction-Tree</i> dari $(\lambda x.yx)(\lambda z.x(yx))$ . . . . .	29
4.1	<i>Construction-Tree</i> dari $\lambda xyz.x(yz)$ . . . . .	44
5.1	<i>Parse tree</i> untuk $\lambda yuv.(\lambda x.x)(vu)$ . . . . .	67
6.1	Alur kerja program implementasi algoritma PT. . . . .	80
6.2	Gambar $\lambda$ - <i>term textfield</i> . . . . .	81
9.1	Alur kerja program implementasi algoritma pencarian <i>Type Inhabitant</i> . . . . .	151
9.2	Gambar <i>type textfield</i> . . . . .	152
A.1	Tampilan panel algoritma Principal Type. . . . .	185
A.2	Tampilan panel algoritma pencarian Type Inhabitant. . . . .	186
A.3	<i>Splash screen</i> dari aplikasi. . . . .	186
A.4	Tampilan menu dalam aplikasi. . . . .	187
F.1	Gambaran umum algoritma PT . . . . .	214
F.2	Kasus I algoritma PT . . . . .	215
F.3	Kasus II algoritma PT . . . . .	216
F.4	Kasus III algoritma PT . . . . .	217
F.5	Kasus IV algoritma PT . . . . .	218

F.6	Kasus IVa algoritma PT . . . . .	219
F.7	Contoh kasus IVa algoritma PT . . . . .	220
F.8	Kasus IVa2 algoritma PT . . . . .	221
F.9	Kasus IVb algoritma PT . . . . .	222
F.10	Contoh kasus IVb algoritma PT . . . . .	223
F.11	Kasus IVb2 algoritma PT . . . . .	224
G.1	Gambaran umum algoritma pencarian <i>Type Inhabitant</i> . . . . .	226
G.2	Sublangkah II dari algoritma pencarian <i>Type Inhabitant</i> . . . . .	227
G.3	Bagian IIa1 dari algoritma pencarian <i>Type Inhabitant</i> . . . . .	228
G.4	Bagian IIa2 dari algoritma pencarian <i>Type Inhabitant</i> . . . . .	228

# Daftar Tabel

D.1 Tabel Rangkuman Hasil Uji Coba Implementasi Algoritma *Principal Type* 197

# Bab 1

## Pendahuluan

Bab ini memaparkan latar belakang, tujuan, ruang lingkup, metodologi penelitian, dan sistematika penulisan laporan tugas akhir ini.

### 1.1 Latar Belakang

Misalkan  $A$  adalah sebuah himpunan yang mengandung semua himpunan yang bukan merupakan elemen dari dirinya sendiri. Apakah  $A$  merupakan elemen dari dirinya sendiri (himpunan  $A$  tersebut)? Atau mungkin untuk lebih mudah memahaminya, kita tinjau pernyataan berikut: "Seorang tukang cukur mencukur semua orang yang tidak mencukur dirinya sendiri, apakah tukang cukur tersebut mencukur dirinya sendiri?". Hal tersebut merupakan paradoks Russel yang paling terkenal, yang muncul sekitar tahun 1900an [Mun07] [BR27]. Baik jawaban positif atau negatif pada pertanyaan tersebut akan menghasilkan kontradiksi.

Dengan tujuan untuk mengatasi paradoks Russel yang telah mengacaukan fondasi matematika pada masa tersebut, diajukanlah *type theory*. Kemudian seiring berkembangnya waktu, penggunaan *type theory* diperluas hingga menjadi alat standar yang digunakan para ahli logika (*logician*), terutama dalam bidang *proof-theory*. Pada tahun 1970an, kebutuhan akan bahasa pemrograman yang lebih tangguh membuat *type theory* menjadi perhatian orang-orang dari dunia ilmu komputer (*computer scientist*). Kemudian beberapa bahasa pemrograman baru dikembangkan pada tahun 1970an dan 80an dengan didasarkan pada *type theory*. Contoh utamanya adalah ML, dikembangkan oleh universitas Edinburgh, melalui grup yang dipimpin oleh Robin Milner. Contoh

yang lainnya adalah HOL (Universitas Cambridge), Miranda (*Research Software Ltd.*) dan Nuprl (Universitas Cornell)[Hin97].

Studi tentang *type system* dan aplikasinya telah memberikan kontribusi yang penting dalam dunia ilmu komputer, seperti dalam rekayasa perangkat lunak, basis data, *computational linguistics*, desain bahasa pemrograman, *automated theorem proving*, *high performance compiler* dan keamanan jaringan komputer. Dalam ilmu komputer terdapat dua cabang utama dalam mempelajari *type system*, yaitu:

1. Aliran yang lebih memfokuskan pada sisi praktis dari *type system* yang menyangkut aplikasinya pada bahasa pemrograman.
2. Aliran yang memfokuskan pada hal yang lebih abstrak, yaitu mempelajari hubungan antar berbagai *type system* dan beraneka ragam logika melalui *Curry Howard correspondence*.

*Type system* pertama kali digunakan dalam dunia ilmu komputer sekitar awal tahun 1950 pada bahasa pemrograman seperti FORTRAN. Penggunaan konsep ini berperan untuk meningkatkan efisiensi proses perhitungan, dengan cara membedakan antara penghitungan bilangan bulat (integer) dan bilangan real. Perbedaan ini memungkinkan *compiler* untuk menggunakan representasi yang berbeda dan membuat instruksi mesin yang sesuai untuk operasi primitif. Selain itu, *type system* juga telah berperan banyak dalam desain bahasa pemrograman. Hal yang paling dapat terlihat jelas adalah memungkinkannya deteksi dini terhadap beberapa *programming error* [Pie02].

*Type theory* memiliki banyak *varian*, salah satunya adalah *Type Assignment* (TA). TA merupakan salah satu *type theory* yang paling sederhana. Dalam sistem TA, sebuah tipe hanya tersusun dari *type-variable* dan tanda panah. *Term* dalam sistem TA ini mengikuti  $\lambda$ -calculus yang *term*-nya hanya tersusun dari  $\lambda$ -abstraction, *application* dan *term-variable*. Kemampuan TA dalam mengekspresikan sesuatu (*expressive power*) mendekati sistem yang disebut *simple type theory*, suatu sistem yang dikembangkan oleh Alonzo Church [Hin97].

Pemikiran dan algoritma di dalam TA tidaklah *trivial*, dan banyak teknik kompleks yang digunakan untuk menganalisa struktur dalam *type theory* yang lebih kompleks hadir di TA dalam bentuk yang mudah untuk dipahami. Hal ini menjadikan TA sebagai suatu sarana pembelajaran yang baik untuk mempelajari teknik dalam *type*

*theory* secara keseluruhan. Dengan mempelajari teknik dasar dari *type theory* melalui TA, kita juga akan mendapatkan fondasi yang baik untuk mempelajari *type system* yang lain [Hin97].

Melihat pentingnya TA dan juga kaitannya dengan *type theory*, dalam tugas akhir ini penulis mempelajari TA dengan berfokus pada dua algoritma dalam TA, yaitu algoritma *Principal Type* dan algoritma pencarian *type inhabitant*. Algoritma *Principal Type* merupakan algoritma untuk memberikan tipe pada sebuah  $\lambda$ -term. Algoritma ini menerima *input* sebuah  $\lambda$ -term. Jika  $\lambda$ -term yang diberikan dapat memiliki tipe (*typable*), maka algoritma ini akan mengeluarkan tipe paling umum dari  $\lambda$ -term tersebut. Namun jika  $\lambda$ -term yang diberikan tidak dapat memiliki tipe (*untypable*), maka algoritma ini akan mengeluarkan pernyataan yang benar bahwa  $\lambda$ -term tersebut tidak dapat memiliki tipe. Kemudian algoritma pencarian *type inhabitant* merupakan algoritma untuk mencari *inhabitant* dari sebuah tipe. Algoritma ini merupakan bagian inti dari algoritma penghitungan *type inhabitant* Ben-Yelles, yang berfungsi untuk menentukan apakah suatu tipe  $\tau$  memiliki sejumlah tak hingga atau berhingga *inhabitant*, atau mungkin tidak memiliki *inhabitant* sama sekali.

Dalam tugas akhir ini kedua algoritma tersebut diimplementasikan menggunakan PROLOG, dan hasilnya dijelaskan pada laporan ini. Dalam mengimplementasikan kedua algoritma tersebut, penulis juga mengembangkan *parser* untuk  $\lambda$ -term dan tipe menggunakan *Definite Clause Grammar* (DCG) dalam PROLOG.

## 1.2 Tujuan

Tujuan dari tugas akhir ini adalah sebagai berikut:

1. Mempelajari algoritma *Principal Type* (PT *algorithm*) dan juga algoritma pencarian *type-inhabitant* pada sistem *Type-Assignment* untuk  $\lambda$ -calculus (sistem  $TA_\lambda$ ), serta memberikan pemaparan yang diharapkan dapat lebih mudah dimengerti dibandingkan dengan literatur yang sudah ada.
2. Membuat *parser*  $\lambda$ -term dengan menggunakan DCG dalam PROLOG.
3. Mengimplementasikan algoritma *Principal Type*.

4. Membuat *parser* untuk tipe dengan menggunakan DCG dalam PROLOG.
5. Mengimplementasikan algoritma pencarian *type-inhabitant*.
6. Membuat antarmuka yang memadai untuk memudahkan *user* dalam menggunakan (mencoba) kedua algoritma tersebut.

## 1.3 Ruang Lingkup

Cakupan tugas akhir ini meliputi pembelajaran algoritma *principal type* dan juga algoritma pencarian *type inhabitant* pada sistem  $TA_\lambda$ . Dalam tugas akhir ini, dua algoritma tersebut diimplementasikan. Selain itu, tugas akhir ini juga meliputi pembuatan *parser* untuk  $\lambda$ -term dan tipe, dengan menggunakan *Definite Clause Grammar* (DCG) dalam PROLOG, serta pembuatan antarmuka yang memadai untuk memudahkan *user* dalam menggunakan (mencoba) kedua algoritma tersebut.

## 1.4 Metodologi Penelitian

Penelitian yang dilakukan terdiri dari beberapa tahapan sebagai berikut:

1. Studi literatur. Pembelajaran literatur melalui buku dan artikel yang berhubungan dengan subyek yang sedang dibahas.
2. Implementasi. Implementasi algoritma *Principal Type* dan algoritma pencarian *type inhabitant* yang dilakukan dengan PROLOG. Implementasi *parser* untuk  $\lambda$ -term dan tipe dari  $\lambda$ -term dilakukan dengan DCG dalam PROLOG, dan pembuatan antarmuka menggunakan Java.
3. Uji coba hasil implementasi. Melakukan uji coba penerapan algoritma *Principal Type* pada sejumlah  $\lambda$ -term dan algoritma pencarian *type inhabitant* pada sejumlah tipe.
4. Analisa terhadap semua hal yang telah dilakukan dalam proses penelitian. Analisa ini menghasilkan sejumlah kesimpulan dan saran.

## 1.5 Sistematika Penulisan Laporan

Berikut adalah sistematika penyajian laporan ini:

Bab 1 Pendahuluan. Bab ini memaparkan latar belakang, tujuan, ruang lingkup, metodologi penelitian, dan sistematika penulisan laporan tugas akhir.

Bab 2 *Definite Clause Grammar* (DCG). Bab ini menjelaskan secara sekilas tentang *Definite Clause Grammar* (DCG) dalam PROLOG. Dalam bab ini juga disertakan beberapa contoh penggunaan DCG.

Bab 3 *Type-Free  $\lambda$ -calculus*. Bab ini menjelaskan pengertian dan beberapa konsep terkait *type-free  $\lambda$ -calculus*.

Bab 4 *Type-Assignment* pada  $\lambda$ -term ( $TA_\lambda$ ). Bab ini memberikan pengenalan kepada sistem  $TA_\lambda$ .

Bab 5 *Principal-Type (PT) Algorithm: Algoritma Pemberian Tipe pada  $\lambda$ -term*. Bab ini menjelaskan algoritma *principal type*. Dalam bab ini juga dijelaskan beberapa konsep pendukung yang dibutuhkan untuk memahami algoritma tersebut.

Bab 6 Implementasi Algoritma *Principal Type*. Bab ini menjelaskan implementasi algoritma *principal type*, yang juga disertai implementasi algoritma unifikasi Robinson. Selain itu bab ini juga menjelaskan implementasi *parser* untuk  $\lambda$ -term yang diimplementasikan dengan DCG dalam PROLOG.

Bab 7 *Typed Term*. Bab ini menjelaskan *typed term*. Alih-alih menjelaskan *typed term* sebagai pendekatan  $TA_\lambda$  yang dilakukan Church secara keseluruhan, bab ini hanya menjelaskan *typed term* sebagai notasi alternatif dalam deduksi  $TA_\lambda$ .

Bab 8 Algoritma Pencarian *Type Inhabitant*. Bab ini memaparkan algoritma pencarian *type inhabitant*.

Bab 9 Implementasi Algoritma Pencarian *Type Inhabitant*. Bab ini menjelaskan implementasi algoritma pencarian *type inhabitant*. Selain itu bab ini juga menjelaskan implementasi *parser* untuk tipe yang diimplementasikan dengan DCG dalam PROLOG.

Bab 10 Kesimpulan dan Saran.