STUDI ANALISIS METODE-METODE PARSING DAN INTERPRETASI SEMANTIK PADA NATURAL *LANGUAGE PROCESSING*

James Suciadi
Fakultas Teknologi Industri, Jurusan Teknik Informatika - Universistas Kristen Petra
e-mail: jsuciadi@peter.petra.ac.id

ABSTRAK: Tiga proses utama yang dilakukan dalam pengolahan bahasa alami ialah analisa sintaksis, interpretasi semantik dan interpretasi kontekstual. Makalah ini akan membahas mengenai proses yang pertama dan kedua. Analisa sintaksis atau *parsing* ialah proses penentuan struktur sebuah kalimat berdasarkan *grammar* dan *lexicon* tertentu.

Parsing dapat dilakukan secara *top-down* maupun *bottom-up*, masing-masing memiliki kelebihan dan kekurangannya sendiri. Top-down parsing tidak dapat menangani grammar dengan *left-recursion*, sedangkan bottom-up parsing tidak dapat menangani grammar dengan *empty production*. Karena itu metode parsing yang terbaik ialah yang dapat menggabungkan kedua cara ini.

Interpretasi semantik ialah proses penerjemahan sebuah kalimat menjadi bentuk representasi artinya yang umum disebut *logical form* tanpa memperhatikan konteks. Dua proses utama yang diperlukan dalam membentuk logical form ialah penentuan peran tiap kata dan frase dalam kalimat, serta pemilihan arti kata yang tepat untuk membentuk kalimat yang masuk akal. Peranan kata-kata dan frase dalam kalimat dapat direpresentasikan dalam bentuk predikat-argumen biasa ataupun menggunakan *thematic roles*. Sedangkan proses pemilihan arti kata yang tepat dapat dilakukan dengan *selectional restrictions* ataupun *context activation*.

Kata kunci: pengolahan bahasa alami, analisa sintaksis, interpretasi semantik, *grammar*, *lexicon*.

ABSTRACT: *Three main processes in Natural Language Processing are syntax analysis or parsing, semantic interpretation and contextual interpretation. This paper discuss about the first and the second of these processes. Parsing is the recognition of the sentence structure based on a grammar and a lexicon.*

Parsing can be done in either top-down or bottom-up methods, each has its own advantages and disadvantages. Top-down parsers can not handle grammar with left-recursion, where bottom-up parsers can not handle grammar with empty production. The best parsers combine these two approaches.

Semantic interpretation is the process of mapping a sentence into its context-independent meaning representation called logical form. There are two processes needed in building logical form, the first is to identify the semantic roles that each word and phrase plays in the sentence, the second is to choose the correct sense of each word to build a plausible sentence, which called word-sense disambiguation. The semantic roles may be represented using predicate-argument relations or using the thematic roles, word-sense disambiguation can be done by selectional restrictions or by context-activation.

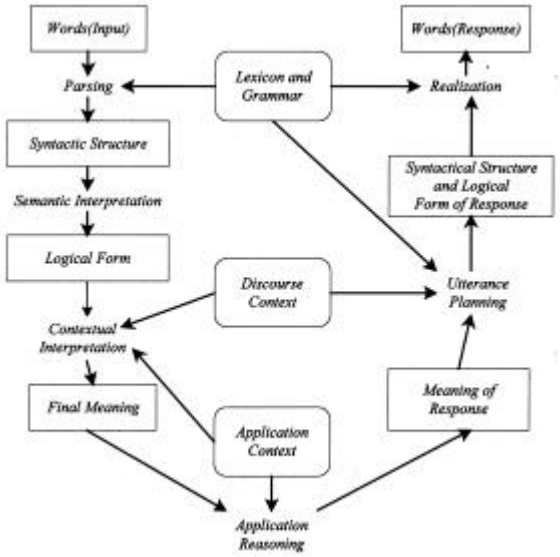
Keywords: natural language processing, parsing, semantik interpretation, grammar, lexicon.

1. PENDAHULUAN

Natual Language Processing (NLP) atau pengolahan bahasa alami merupakan salah satu bidang ilmu *Artificial Intelligence* (Kecerdasan Buatan) yang mempelajari komunikasi antara manusia dengan komputer melalui bahasa alami. NLP tidak bertujuan untuk mentransformasikan bahasa yang diterima dalam bentuk suara menjadi data digital dan/atau sebaliknya pula; melainkan bertujuan untuk memahami arti dari ucapan yang diberikan dalam bahasa alami dan memberikan respon yang sesuai, misalnya dengan melakukan suatu aksi tertentu atau menampilkan data tertentu.

Untuk mencapai tujuan ini dibutuhkan tiga tahap proses. Proses yang pertama ialah *parsing* atau analisa sintaksis yang memeriksa kebenaran struktur kalimat

berdasarkan suatu *grammar* (tata bahasa) dan *lexicon* (kosa kata) tertentu. Proses kedua ialah *semantic interpretation* atau interpretasi semantik yang bertujuan untuk merepresentasikan arti dari kalimat secara *context-independent* untuk keperluan lebih lanjut. Sedangkan proses ketiga ialah *contextual interpretation* atau interpretasi kontekstual yang bertujuan untuk merepresentasikan arti secara *context-dependent* dan menentukan maksud dari penggunaan kalimat. Gambaran organisasi dari sebuah sistem NLP yang lengkap ditunjukkan pada gambar 1.



Gambar 1. Organisasi Sebuah Sistem NLP

2. GRAMMAR DAN LEXICON

Grammar dan lexicon memegang peranan yang sangat penting dalam NLP. Perkembangan NLP mengalami kemajuan yang sangat pesat sejalan dengan berkembangnya grammar dan lexicon.

2.1 Grammar

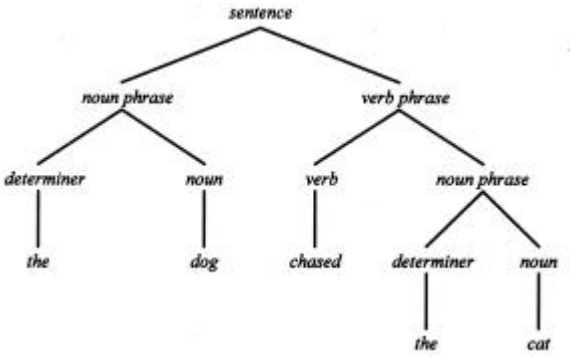
Grammar ialah aturan-aturan pembentukan suatu kalimat dalam sebuah bahasa, atau yang biasa disebut tata bahasa. Dengan adanya grammar, parsing dapat dilakukan secara cepat dengan hanya melakukan proses *searching*. Parser akan mencari aturan-aturan yang tepat untuk membentuk struktur suatu kalimat. Salah satu contoh grammar yang sederhana untuk bahasa Inggris ialah sebagai berikut:

```
sentence     ® noun phrase + verb phrase
noun phrase  ® determiner + noun
verb phrase  ® verb + noun phrase
determiner   ® the, a
noun         ® dog, dogs, cat
verb         ® chased, see, sees
```

Grammar ini akan mampu menghasilkan sekumpulan kalimat bahasa Inggris yang sederhana, seperti :

- *The dog chased a cat*
- *The dog sees a cat*
- *The dogs sees a cat (nonvalid)*
- *A dogs chased a cat (nonvalid)*

Proses pembentukan kalimat dari grammar disebut *derivation*. Contoh derivation untuk menghasilkan kalimat "*The dog chased the cat*" dari grammar di atas ditunjukkan pada gambar 2.



Gambar 2. Parse Tree untuk Kalimat *"The dog chased the cat"*

2.2 Lexicon

Untuk membuat sebuah aplikasi NLP yang handal, dibutuhkan kamus atau kosa kata yang handal pula. Seperti halnya seorang manusia, semakin lengkap kosa kata dalam sebuah sistem NLP, semakin baik sistem tersebut dapat berkomunikasi. Kosa kata dalam komputer disebut lexicon.

Faktor yang penting menyangkut lexicon ialah penyimpanannya, karena umumnya lexicon memiliki ukuran yang sangat besar. Karena itu umumnya lexicon hanya menyimpan bentuk dasar dari kata-kata yang ada, sedangkan untuk bentuk-bentuk turunannya (misalnya *kicks* dari *kick*) dapat diperoleh dengan menerapkan analisa *morphology* (proses perubahan bentuk kata).

3. PARSING

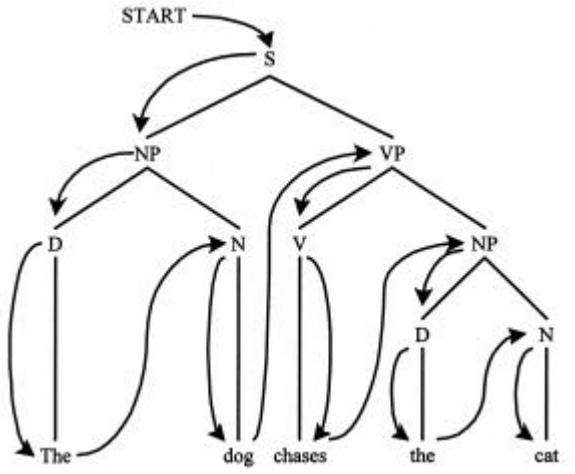
Proses parsing tidak hanya dapat dilakukan dalam NLP, melainkan juga pada bidang lain seperti pada pembuatan sebuah *compiler*. Metode-metode parsing yang dibahas berikut khusus digunakan dalam NLP. Sebelumnya perlu diketahui arti dari istilah *constituent*, yaitu unsur-unsur pembentuk kalimat yang dapat berdiri sendiri, contohnya *noun phrase*, *verb phrase* dan sebagainya; dan istilah *parser* yaitu program yang melakukan proses parsing.

3.1 Top-down Parsing

Top-down parser bekerja dengan cara menguraikan sebuah kalimat mulai dari constituent yang terbesar yaitu sampai menjadi constituent yang terkecil. Hal ini dilakukan terus-menerus sampai semua komponen yang dihasilkan ialah constituent terkecil dalam kalimat, yaitu kata.

Sebagai contoh, dengan menggunakan contoh grammar di atas, dapat dilakukan proses top-down parsing untuk kalimat *"The dog chased the cat"* yang ditunjukkan pada gambar 3. Dari gambar ini terlihat bahwa top-down parser menelusuri setiap node pada parse tree secara *pre-order*. Beberapa metode parsing yang bekerja secara top-down ialah:

- Top-down parser biasa
- Recursive-descent parser
- Transition-network parser
- Chart parser



Gambar 3. Cara Kerja Top-down Parser

Top-down parser dapat diimplementasikan dengan berbagai bahasa pemrograman, namun akan lebih baik jika digunakan *declarative language* seperti Prolog atau LISP. Hal ini disebabkan oleh karena pada dasarnya proses parsing ialah proses searching yang dilakukan secara rekursif dan backtracking, dimana proses ini sudah tersedia secara otomatis dalam bahasa Prolog.

Dengan demikian parser yang ditulis dalam Prolog atau bahasa deklaratif lainnya akan menjadi jauh lebih sederhana daripada parser yang dibuat dalam bahasa prosedural biasanya seperti Pascal, C dan sebagainya. Program 1 menunjukkan implementasi top-down parser biasa dalam bahasa Prolog.

Program 1 Simple Top-Down Parser

```
1.  DOMAINS
2.      list = symbol*
3.  PREDICATES
4.      rule(symbol,list)
5.      word(symbol,symbol)
6.      parse(symbol,list,list)
7.      parse_list(list,list,list)
8.
9.  CLAUSES
10.     % rules
11.     rule(s,[np,vp]).
12.     rule(np,[d,n]).
13.     rule(vp,[v]).
14.     rule(vp,[v,np]).
15.     rule(d,[]).
16.
17.     % lexicon
18.     word(d,the).
19.     word(d,a).
20.     word(n,dog).
21.     word(n,dogs).
22.     word(n,cat).
23.     word(n,cats).
24.     word(v,chase).
25.     word(v,chases).
26.     word(v,barked).
27.
28.     % parse(C,S1,S)
29.     % periksa apakah S1 dimulai dengan
        constituent C
30.     % S ialah sisa kalimat setelah C
        dikeluarkan dari S1
31.     parse(C,[Word|S],S) :-
32.        word(C,Word).      % jika C ialah
           terminal (kata), stop
33.     parse(C,S1,S) :-
34.        rule(C,Cs), % jika C ialah nonter-
           minal, expand
35.        parse_list(Cs,S1,S).
36.
37.     % parse_list([C|Cs],S1,S)
38.     % periksa apakah S1 dimulai dengan
        constituent berkategori
39.     %    C diikuti oleh Cs
40.     % S ialah sisa kalimat setelah C dan
        Cs dikeluarkan
41.     %    dari S1
42.     parse_list([C|Cs],S1,S) :-
43.        parse(C,S1,S2),
44.        parse_list(Cs,S2,S).
45.     parse_list([],S,S).
```

Predikat *parse* berfungsi untuk melakukan proses parsing terhadap sebuah *constituent* tunggal (misalnya *s*, *np* dan sebagainya), sedangkan *parse_list* berfungsi untuk melakukan proses parsing terhadap sekumpulan constituent, misalnya [*np,vp*], [*d,n*] dan sebagainya.

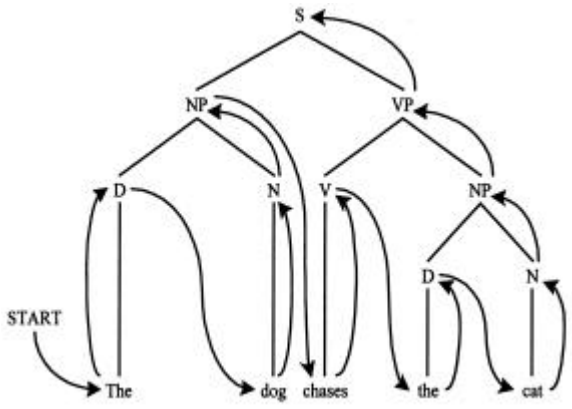
Parser akan dipanggil dengan query *parse(s,X,[]),* dimana *s* berarti kalimat dan *X* ialah input kalimat yang berbentuk list dari symbol. Sebagai contoh, untuk melakukan parsing terhadap kalimat *"The dog chases the cat"*, maka query yang diberikan pada external goal Turbo Prolog ialah sebagai berikut :

```
?- parse(s,[the,dog,chases,the,cat],[])
```

dan Prolog akan memberikan jawaban YES karena kalimat tersebut sesuai dengan grammar yang ditunjukkan pada baris 11-15 oleh predikat *rule*.

3.2. Bottom-Up Parsing

Bottom-up parser bekerja dengan cara mengambil satu demi satu kata dari kalimat yang diberikan, untuk dirangkaikan menjadi constituent yang lebih besar. Hal ini dilakukan terus-menerus sampai constituent yang terbentuk ialah sentence atau kalimat. Dengan demikian metode bottom-up bekerja dengan cara yang terbalik dari top-down. Cara kerja bottom-up parser ditunjukkan pada gambar 4.



Gambar 4. Cara Kerja Bottom-up Parser

Metode parsing yang bekerja secara bottom-up antara lain ialah bottom-up parser biasa dan shift-reduce parser. Program 2 menunjukkan contoh implementasi bottom-up parser biasa dalam bahasa Turbo Prolog. Perhatikan bahwa parser ini tidak membedakan antara *rule* (grammar) dan *word* (lexicon) sehingga cara kerjanya sangat sederhana namun sangat "bodoh" karena akan terus mengulang-ulang kesalahan yang sama.

Kesederhanaan metode ini terletak pada predikat untuk parsing, yaitu *parse* yang hanya memiliki sebuah argumen. Argumen ini berisi kalimat yang akan diparse dalam bentuk list dari symbol. Kata-kata dari input kalimat akan dirangkaikan sambil mencari aturan yang lebih luas, sampai tinggal sebuah simbol saja dalam list, yaitu *s*.

Program 2 Simple Bottom-Up Parser

```
1:  DOMAINS
2:     list = symbol*
3:
4:  PREDICATES
5:     rule(symbol,list)
6:     parse(list)
7:     append(list,list,list)
8:
9:  CLAUSES
10:    % rules
11:    rule(s,[np,vp]).
12:    rule(np,[d,n]).
13:    rule(vp,[v]).
14:    rule(vp,[v,np]).
15:
16:    % lexicon
17:    rule(d,[the]).
18:    rule(d,[a]).
19:    rule(n,[dog]).
20:    rule(n,[cat]).
21:    rule(v,[chased]).
22:    rule(v,[saw]).
23:    rule(v,[barked]).
24:    rule(v,[runs]).
25:
26:    % parse list dengan cara bottom-up
27:    % parser akan mencari semua kemung-
       kinan kombinasi
28:    % kata yang dapat membentuk consti-
       tuent
29:    parse([s]).
30:    parse(List) :-
31:         append(Front,Back,List),
32:         append(RHS,Rest,Back),
33:         rule(LHS,RHS),
34:         append(Front,[LHS|Rest],NewList),
35:         parse(NewList).
36:
37:    % append untuk menambah isi list 1 ke
       list 2
38:    % hasilnya pada list 3
39:         append([],X,X).
40:         append([H1|T1],L2,[H1|L3]) :-
41:              append(T1,L2,L3).
```

Jika diberikan input kalimat *"The dog chased the cat"* maka query pada external goal Turbo Prolog berbentuk :

```
?-parse([the,dog,chased,the,cat])
```

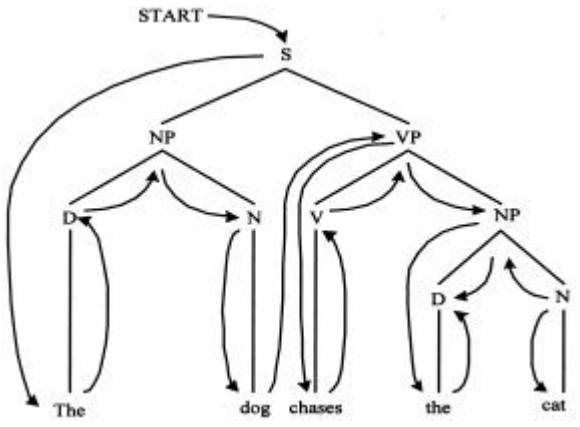
Prolog kemudian akan memebrikan jawaban YES karena kalimat tersebut sesuai dengan grammar pada baris 11-24.

3.3 Gabungan Top-Down dan Bottom-Up Parsing

Baik top-down parsing mapun bottom-up parsing memiliki kekurangan dan kelebihannya masing-masing. Metode top-down mampu menangani grammar dengan *empty production* (misalnya *d ® 0*) namun tidak dapat menangani grammar dengan *left recursion* (misalnya *np ® np conj np*). Sedangkan metode bottom-up dapat menangani left recursion namun tidak dapat menangani empty production.

Dengan demikian metode parsing yang terbaik ialah metode yang dapat menggabungkan top-down dan *bottom-up* parsing. Ada beberapa metode yang dikembangkan yang menggabungkan kedua metode ini, di antaranya ialah *left-corner parsing* serta *Earley's parsing*.

Cara kerja left-corner parsing ialah dengan mula-mula menerima sebuah kata, menentukan jenis constituent apa yang dimulai dengan jenis kata tersebut, kemudian melakukan proses parsing terhadap sisa dari constituent tersebut secara top-down. Dengan demikian proses parsing dimulai secara bottom-up dan diakhiri secara top-down. Program 3 menunjukkan implementasi left-corner parser dalam Turbo Prolog, sedangkan alur kerjanya ditunjukkan pada gambar 5.



Gambar 5. Cara Kerja Left-Corner Parser

Program 3 Left-Corner Parser

```
1:  DOMAINS
2:     list = symbol*
3:
4:  PREDICATES
5:     rule(symbol,list)
6:     word(symbol,symbol)
7:     link(symbol,symbol)
8:     parse(symbol,list,list)
9:     parse_list(list,list,list)
10:    complete(symbol,symbol,list,list)
11:
12: CLAUSES
13:    % rules
14:    rule(s,[np,vp]).
15:    rule(np,[d,n]).
16:    rule(vp,[v]).
17:    rule(vp,[v,np]).
18:    rule(d,[]).
19:
20:    % lexicon
21:    word(d,the).
22:    word(d,a).
23:    word(n,dog).
24:    word(n,dogs).
25:    word(n,cat).
26:    word(n,cats).
27:    word(v,chases).
28:    word(v,chased).
29:    word(v,barked).
30:
31:    %links
32:    link(np,s).
33:    link(d,np).
34:    link(d,s).
35:    link(v,vp).
36:    link(X,X).
37:
38:    % parse(C,S1,S)
39:    %  parse constituent dengan kategori
       C yang dimulai dengan
40:    %    input  string  S1  dan  berakhir
       dengan input string S
41:    parse(C,[Word|S2],S) :-
42:       word(W,Word),
43:       link(W,C),
44:       complete(W,C,S2,S).
45:    parse(C,S2,S) :-
46:    rule(W,[]),% untuk null constituent
47:       link(W,C),
48:       complete(W,C,S2,S).
49:
50:    % parse(C1,S1,S)
51:    %  parsing list C1 dan hasilnya di S
52:    parse_list([C|Cs],S1,S) :-
53:       parse(C,S1,S2),
54:       parse_list(Cs,S2,S).
55:    parse_list([],S,S).
56:
57:    % complete(W,C,S1,S)
58:    % Memastikan bahwa W ialah subconsti-
       tuent pertama dari C,
59:    % kemudian melakukan left-corner par-
       sing terhadap sisa dari C
60:    complete(C,C,S,S).%  jika  C=W  jangan
       lakukan apa-apa
61:    complete(W,C,S1,S) :-
62:      rule(P,[W|Rest]),
63:      parse_list(Rest,S1,S2),
64:      complete(P,C,S2,S).
```

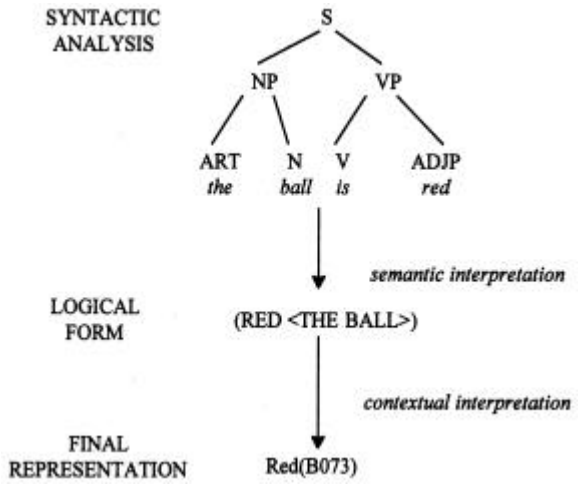
Misalnya diberikan query

```
?- parse(s,[the,dog,chased,cat],[])
```

maka parser akan memberikan jawaban YES karena kalimat tersebut sesuai dengan grammar pada baris 14-18.

4. INTERPRETASI SEMANTIK PADA NLP

Interpretasi semantik bertujuan untuk menerjemahkan kalimat menjadi bentuk representasi yang disebut logical form, sedangkan contextual interpretation akan menerjemahkan logical form ke dalam bentuk representasi lain yang disebut *knowledge representation* atau *final meaning representation*. Gambar 6 menunjukkan contoh sederhana mengenai apa yang dilakukan oleh proses-proses dalam NLP.



Gambar 6. Tahap-tahap Interpretasi Kalimat

Input untuk proses interpretasi semantik ialah parse tree hasil parsing, sedangkan outputnya ialah representasi yang sesuai dalam arti dapat diproses dan dipahami oleh komputer. Representasi yang umum dipakai ialah logical form. Sebenarnya teks asalpun merupakan suatu bentuk representasi, hanya saja masih dalam bentuk yang belum dapat dipahami komputer.

4.1 The Logical Form

Logical form merupakan bentuk representasi arti secara context-independent dengan menggunakan hukum-hukum logika. Logical form dipandang sesuai untuk representasi arti dari bahasa alami karena bahasa alami memiliki persamaan-persamaan dengan pernyataan-pernyataan logika. Sebagai contoh, jika sebuah kalimat bernilai benar, maka negasi kalimat tersebut bernilai salah. Selain itu, kalimat-kalimat dalam bahasa alami bisa pula dihubungkan dengan operator *'and'*, *'or'* dan sebagainya seperti pada pernyataan-pernyataan logika.

Komputer menyimpan arti dari kata-kata bukan dalam bentuk definisinya, melainkan dalam suatu bentuk hubungan antara predikat dan argumennya. Sebagai contoh, arti dari kata kerja *'eat'* dapat disimpan sebagai berikut :

eat(<ANIMATE>,<FOOD>)

Ini berarti, predikat *'eat'* membutuhkan dua buah argumen di mana argumen pertama berjenis ANIMATE (mahluk hidup yang bergerak), sedangkan argumen kedua berjenis FOOD (makanan).

Sebuah kalimat dibentuk dari sebuah *predicate* diikuti oleh satu atau lebih *terms* sebagai argumennya. Predicate ialah komponen logical form yang menjelaskan hubungan antar obyek atau sifat dari suatu obyek, sedangkan terms ialah komponen logical form yang menyatakan tentang suatu obyek di dunia, termasuk kejadian (*event*) dan situasi. Predicate yang hanya membutuhkan sebuah argumen disebut unary predicate, dan disebut pula properties atau sifat. Predicate yang membutuhkan dua argumen disebut binary predicate; dan predicate yang membutuhkan n argumen disebut n-*ary predicate.*

Jenis representasi logical form seperti ini disebut bentuk predicate-argument. Sebagai contoh, kalimat *"John broke the window with the hammer"* dapat direpresentasikan sebagai berikut :

((PAST BREAKS) JOHN (THE WINDOW) (WITH THE HAMMER))

Artinya, *John* melakukan kegiatan *break* pada waktu yang lampau, terhadap *the window* dengan alat *with the hammer*.

4.2 Thematic roles

Dengan bentuk predicate-argument, kata kerja yang sama tetapi memiliki jumlah

argumen yang berbeda akan disimpan sebagai arti yang berbeda. Pendekatan semacam ini lebih mudah diimplementasikan, namun kurang menjelaskan peranan tiap frase secara tepat. Perhatikan contoh berikut ini :

John broke the window with the hammer.
The hammer broke the window.
The window broke.

Bentuk-bentuk logical form dari contoh-contoh di atas ialah:

((PAST BREAKS) JOHN (THE WINDOW) (WITH THE HAMMER))
((PAST BREAKS) (THE HAMMER) (THE WINDOW))
((PAST BREAKS) (THE WINDOW))

Pada kalimat kedua *the hammer* dianggap sebagai subyek, sedangkan pada kalimat ketiga *the window* yang dianggap sebagai subyek.

Untuk mengatasi hal ini, Davidson (1967) memperkenalkan *event*, dan lebih lanjut konsep ini dikembangkan menjadi *case grammar* pada tahun 1970. Konsep ini kemudian dikembangkan lagi menjadi apa yang disebut *thematic roles* atau *case roles.* Prinsipnya ialah dengan memandang suatu kalimat sebagai suatu event atau kejadian, dan menentukan peranan tiap kata atau frase dalam kejadian tersebut. Peran-peran inilah yang disebut thematic roles.

Sejumlah thematic roles yang umum digunakan ialah sebagai berikut :

- *AGENT*
- *THEME*
- *INSTRUMENT*
- *BENEFICIARY*
- *EXPERIENCER*
- *CO-AGENT, CO-THEME*
- *AT-LOC, FROM-LOC, TO-LOC, PATH-LOC*
- *AT-TIME, FROM-TIME, TO-TIME*
- *AT-VALUE, FROM-VALUE, TO-VALUE*
- *AT-POSS, FROM-POSS, TO-POSS*

Dengan thematic roles, logical form dari contoh kalimat kedua di atas menjadi sebagai berikut :

(EVENT (BREAK)) (INSTRUMENT (THE HAMMER)) (THEME (THE WINDOW)) (AT-TIME (PAST))

Terlihat bahwa bentuk ini lebih memperjelas peranan tiap kata dibandingkan dengan bentuk predicate-argument biasa. Program 4 menunjukkan implementasi *thematic roles* dalam Turbo Prolog.

Program 4 Contoh Implementasi Thematic Roles Sederhana

```
DOMAINS
id  = integer
tuple = t(symbol,slist)
list  = tuple*
lf    = logical_form(list)
slist = symbol*

PREDICATES
lex(symbol,slist,slist)
rule(symbol,slist)
word(symbol,symbol)
parse(symbol,slist,slist)
parse_list(slist,slist,slist)
intsem(slist,lf)
semantik(slist,slist,list,list,slist)
semantik2(slist,slist,list,list,slist)
carievent(slist,symbol,slist,slist)
insertt(tuple,list,list)
insert(symbol,slist,slist)
carinp(slist,slist,slist)
process(slist)
cetak(list)
writelist(slist)
gabung(slist,slist,slist)

CLAUSES
lex(reads,[agent,theme],[]).
lex(breaks,[agent,theme],[instrument]).
lex(puts,[agent,theme,at_loc],[]).
lex(gives,[agent,theme,to_poss],[]).
lex(runs,[theme],[to_loc]).

rule(s,[np,vp]).
rule(s,[np,vp,pp]).
rule(np,[d,n]).
rule(d,[]).
rule(vp,[v]).
rule(vp,[v,np]).
rule(pp,[p,np]).

word(d,the).
word(n,boy).
word(n,cat).
word(n,window).
word(n,hammer).
word(n,house).
word(n,book).
word(n,table).
word(p,with).
word(p,to).
word(p,on).
word(v,X) :-
        lex(X,_,_).

parse(C,[Word|S],S) :-
        word(C,Word).
parse(C,S1,S) :-
        rule(C,Cs),
        parse_list(Cs,S1,S).

parse_list([C|Cs],S1,S) :-
        parse(C,S1,S2),
        parse_list(Cs,S2,S).
        parse_list([],S,S).

intsem(L,logical_form(T2)) :-
        carievent(L,E,[],L1),
        lex(E,S1,S2),
        insertt(t(event,[E]),[],T0),
        semantik(L1,S1,T0,T1,L2),
```

```
71:         semantik2(L2,S2,T1,T2,_).
72:
73: semantik(L,[],X,X,L) :- !.
74: semantik(L,[agent|T],I,O,NL) :-
75:         carinp(L,NP,L2),
76:         insertt(t(agent,NP),I,I2),
77:         semantik(L2,T,I2,O,NL).
78: semantik(L,[theme|T],I,O,NL) :-
79:         carinp(L,NP,L2),
80:         insertt(t(theme,NP),I,I2),
81:         semantik(L2,T,I2,O,NL).
82: semantik(L,[instrument|T],I,O,NL) :-
83:         L = [with|L1],
84:         carinp(L1,NP,L2),
85: insertt(t(instrument,[with|NP]),I,I2),
86:         semantik(L2,T,I2,O,NL).
87: semantik(L,[at_loc|T],I,O,NL) :-
88:         L = [on|L1],
89:         carinp(L1,NP,L2),
90:         insertt(t(at_loc,[on|NP]),I,I2),
91:         semantik(L2,T,I2,O,NL).
92: semantik(L,[to_loc|T],I,O,NL) :-
93:         L = [to|L1],
94:         carinp(L1,NP,L2),
95:         insertt(t(to_loc,[to|NP]),I,I2),
96:         semantik(L2,T,I2,O,NL).
97: semantik(L,[to_poss|T],I,O,NL) :-
98:         L = [to|L1],
99:         carinp(L1,NP,L2),
100:        insertt(t(to_poss,[to|NP]),I,I2),
101:        semantik(L2,T,I2,O,NL).
102: semantik2(L,[instrument|T],I,O,NL) :-
103:        L = [with|L1],
104:        carinp(L1,NP,L2), !,
105:        insertt(t(instrument,[with|NP]),I
           ,I2),
106:        semantik2(L2,T,I2,O,NL).
107: semantik2(L,[to_loc|T],I,O,NL) :-
108:        L = [to|L1],
109:        carinp(L1,NP,L2), !,
110:        insertt(t(to_loc,[to|NP]),I,I2),
111:        semantik(L2,T,I2,O,NL).
112: semantik2(L,_,I,I,L).
113:
114: carinp([A,B|T],[A,B],T) :-
115:        word(d,A),
116:        word(n,B), !.
117: carinp([A|T],[A],T) :-
118:        word(n,A).
119:
120: insertt(X,[],[X]) :- !.
121: insertt(X,[H|T],[H|T1]) :-
122:         insertt(X,T,T1).
123: insert(X,[],[X]) :- !.
124: insert(X,[H|T],[H|T1]) :-
125:         insert(X,T,T1).
126:
127: carievent([H|T],H,L1,L2) :-
128:         word(v,H), !,
129:         gabung(L1,T,L2).
130: carievent([H|T],E,L1,L2) :-
131:         insert(H,L1,L3),
132:         carievent(T,E,L3,L2).
133:
134: gabung(L,[],L) :- !.
135: gabung(L1,[H|T],L2) :-
136:         insert(H,L1,L3),
137:         gabung(L3,T,L2).
138:
139: process(L) :-
140:        parse(s,L,[]),
141:        intsem(L,X),
142:        write(X),
143:        X = logical_form(LF),
144:        cetak(LF).
145:
146: cetak([]) :- !.
147: cetak([H|T]) :-
148:        H = t(A,B),
149:        write("\n",A," : "),
150:        writelist(B),
151:        cetak(T).
152:
153: writelist([]) :- !.
154: writelist([H|T]) :-
155:        write(H," "),
156:        writelist(T).
```

Jika program ini dijalankan dan diberikan query pada external goal Turbo Prolog sebagai berikut :

```
?-process([the,boy,breaks,the,window,with,
the, hammer])
```

maka program akan menghasilkan output sebagai berikut :

```
logical_form([t(event,[breaks]),t(agent,[t
              he,boy]),
              t(theme,[the,window]),
              t(instrument,[with,the,hamme
              r])
event : breaks
agent : the boy
theme : the window
instrument : with the hammer
```

4.3 Word-sense Disambiguation

Terdapat tiga jenis *lexical ambiguity* (kerancuan arti yang berhubungan dengan kata), yaitu *polysemy*, *homonymy* dan *categorial ambiguity.* Polysemy ialah kata-kata yang memiliki arti yang sama atau berhubungan satu dengan yang lainnya, misalnya *'open'*, *'unfolding'*, *'expanding'*, dan sebagainya. Homonymy ialah arti-arti yang berbeda dari sebuah kata, misalnya kata *'bark'* yang bisa berarti 'gonggongan' ataupun 'kulit kayu'. Sedangkan categorial ambiguity ialah kata yang bisa berfungsi sebagai jenis kata yang berbeda, contohnya ialah kata *'dog'* yang bisa berfungsi sebagai kata benda ataupun kata kerja.

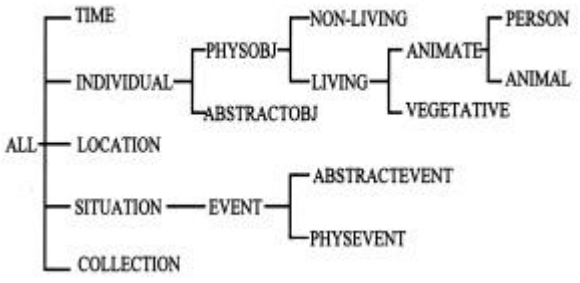
Yang dimaksud word-sense ambiguity pada interpretasi semantik ialah homonimy. Categorial ambiguity merupakan masalah yang ditangani dalam proses parsing, sedangkan polysemy merupakan masalah yang ditangani contextual interpretation, karena komputer harus mengetahui ungkapan-ungkapan mana yang memiliki arti yang sama. Untuk mengatasi homonimy akan digunakan teknik selectional restriction dan *context activation*.

a. Selectional Restriction

Arti dari kata-kata dapat saling berelasi sesuai dengan kelas obyek yang dijelaskan. Sebagai contoh, DOG1 menjelaskan kelas obyek *'dog'*, CAT1 menjelaskan kelas obyek *'cat'*, dan sebagainya. DOG1 dan CAT1 disebut hubungan yang *disjoint*, karena menjelaskan kelas obyek yang berbeda. Namun keduanya bisa pula berada pada kelas yang sama, jika menjelaskan kelas obyek *'animal'*. Selain itu terdapat pula relasi yang berupa *subclass*, misalnya antara DOG dan MAMMAL; dan relasi yang *overlap*, misalnya antara PET dan MAMMAL.

Relasi-relasi ini penting artinya dalam word-sense disambiguation. Sebagai contoh, predikat *'purple'* yang menunjukkan warna harus memiliki argumen berupa *physical object*, karena sebuah obyek yang mempunyai warna haruslah obyek yang konkrit. Physical object sendiri dapat dibagi menjadi kelas *living* dan *non-living*, sehingga baik *'table'* maupun *'horse'* dapat menjadi argumen untuk predikat *'purple'*.

Untuk menjelaskan hubungan-hubungan ini dibutuhkan suatu hirarki yang disebut *word-sense hierarchy* atau *semantic type hierarchy*. Hirarki ini menunjukkan pembagian jenis-jenis obyek yang ada di dunia, baik yang konkrit maupun yang abstrak. Hirarki yang lengkap untuk bahasa Inggris sangatlah luas dan kompleks, namun beberapa ahli telah memperkenalkan hirarki yang cukup sederhana yang dapat digunakan dalam NLP. Salah satu contoh word-sense hierarchy yang digunakan dalam NLP ditunjukkan oleh gambar 7 [2].



Gambar 7. Word-Sense Hierarchy

Selectional restrictions memilih arti kata yang tepat berdasarkan hubungan yang sesuai antara sebuah predikat dan argumennya. Tiap argumen merupakan salah satu kelas obyek dalam word-sense hierarchy. Sebagai contoh, dengan mengasumsikan bahwa kelas obyek *'non-living'* masih dapat dibagi menjadi *'textobj'* dan *'non-textobj'*, maka predikat *read* dapat memiliki bentuk selectional restriction berikut :

(EVENT READ) (AGENT PERSON) (THEME TEXTOBJ)

yang berarti *agent* untuk event *read* harus berupa *person*, sedangkan *theme* untuk event *read* harus berupa *textobj*.

b. Context Activation

Teknik ini dapat memilih arti kata yang tepat dalam ruang lingkup yang terbatas, dengan cara mengikutsertakan konteks dalam lexicon. Sebagai contoh, perhatikan arti dari kata 'pen' yang ambiguous pada dua kalimat berikut :

There are pigs in the pen (pen = kandang hewan)
There is ink in the pen (pen = alat untuk menulis)

Arti dari kata 'pen' ditentukan oleh konteks yang ada sebelumnya, yaitu 'pigs' yang menunjukkan konteks pertanian, ataupun 'ink' yang menunjukkan konteks penulisan. Posisi kata-kata yang menunjukkan ciri konteks ini tidak harus selalu mendahului kata 'pen'.

Untuk memilih arti kata yang tepat, digunakan dua macam lexicon yaitu kata-kata yang tidak ambiguous dan kata-kata yang bersifat ambiguous. Kedua macam lexicon ini menentukan konteks apa yang sedang aktif. Kemudian input kalimat akan discan untuk mencari konteks dari kata-kata yang tidak ambiguous, dan menerapkan konteks ini kepada kata-kata yang ambiguous untuk memilih arti yang tepat. Program 5 menunjukkan implementasi word-sense disambiguation dengan context activation dalam Turbo Prolog.

Program 5 Word-sense Disambiguation dengan Context-Activation

```
1: DOMAINS
2:       list = symbol*
3: PREDICATES
4:       cue(symbol,symbol) % word has single
         meaning for context
5:       mic(symbol,symbol,symbol) % word is
         ambiguous in context
6:       collect_cues(list,list,list)
7:       disambiguate(list,list,list)
8:       member(symbol,list)
9:       process(list,list,list)
10: CLAUSES
11:      cue(farmer,farm).
12:      cue(pigs,farm).
13:      cue(ink,writing).
14:      cue(airport,aircraft).
15:      cue(carpenter,woodworking).
16:
17:      mic(pen,pen_for_animals,farm).
18:      mic(pen,writing_pen,writing).
19:      mic(plane,plane_tool,woodworking).
20:      mic(plane,airplane,aircraft).
21:      mic(terminal,airport_terminal,aircra
         ft).
22:      mic(terminal,computer_terminal,compu
         ter).
23:
24:      member(X,[X|_]) :- !.
25:      member(X,[_|T]) :-
26:            member(X,T).
27:
28:      collect_cues([],C,C).
29:      collect_cues([H|T],C,NC) :-
30:            cue(H,C1),
31: not(member(C1,C)), !,
32:            collect_cues(T,[C1|C],NC).
33:      collect_cues([_|T],C,NC) :-
34:            collect_cues(T,C,NC).
35:
36:      disambiguate([],[],_).
37:      disambiguate([H|T],[H2|T2],C) :-
38:            mic(H,H2,C1),% H is ambiguous
              and means H2 in context C1
39:            member(C1,C),
40:            disambiguate(T,T2,C).
41:      disambiguate([H|T],[H|T2],C) :-
42:            not(mic(H,_,_)),
43:            disambiguate(T,T2,C).
44:
45:      process(W,D,C) :-
46:            collect_cues(W,[],C),
47:            disambiguate(W,D,C).
```

Program dipanggil dengan query pada external goal Prolog dengan predikat process seperti berikut :

```
?-process([there,are,pigs,in,the,pen],NewSen-
tence,Context)
```

Hasil output query ini ialah :

```
NewSentence=  [there,are,pigs,in,the,pen_for_
animals], Context = [farm]

YES
```

Kekurangan metode ini ialah ruang lingkupnya sangatlah kecil, karena untuk konteks yang sempit sekalipun terdapat cukup banyak kata yang bersifat ambiguous.

5. KESIMPULAN

Dari hasil penulisan makalah ini, beberapa kesimpulan yang dapat diambil ialah sebagai berikut :

- Tidak ada metode parsing yang ideal untuk segala masalah dalam NLP. Pemilihan metode parsing yang digunakan harus dilakukan secara jeli, dengan memperhatikan kompleksitas grammar dan kebutuhan dari aplikasi.
- Jika aplikasi NLP yang dibuat menggunakan grammar yang kompleks, maka parser yang menggabungkan metode top-down dan bottom-up merupakan pilihan yang terbaik karena mengatasi kekurangan pada masing-masing metode.
- Thematic roles lebih menguntungkan dalam pembentukan logical form dibandingkan dengan hubungan predicate-argument biasa, karena lebih memperjelas peran tiap elemen kalimat dibandingkan dengan predicate-argument biasa.
- Word-sense hierarchy yang digunakan oleh selectional restrictions sangat membantu dalam melakukan proses word-sense disambiguation, sehingga lebih baik dibandingkan context activation.

DAFTAR PUSTAKA

1. James Allen, *Natural Language Understanding*, Benjamin-Cummings Publishing Company, 1991

2. Michael A. Covington, *Natural Language Processing for Prolog Programmers*, Prentice Hall, 1994

3. Gerald Gazdar and Chris Mellish, *Natural Language Processing in Prolog*, Addison-Wesley, 1989

4. Graeme Hirst, *Semantic Interpretation and the Resolution of Ambiguity*, 1987