Simple Text Parser
Assignment in Computational Linguistics
Module, University of Derby in Austria

Dimitri PISSARENKO
dimitri.pissarenko@gmx.net

May 16, 2002

Contents

Table of contents **1**

List of figures **2**

1 Assignment specification **3**
1.1 General description . . . 3
1.2 Chosen text . . . 3

2 Linguistic side of parsing **3**
2.1 Elements of language . . . 3
2.2 Phrase structure representation . . . 5
2.3 Representation of grammatical rules in Prolog . . . 5
2.4 Definition of the grammar for the chosen text . . . 6
2.4.1 Sentence 1 . . . 8
2.4.2 Sentence 2 . . . 8
2.4.3 Sentence 3 . . . 13
2.4.4 Sentence 4 . . . 19
2.4.5 Sentence 5 . . . 20
2.4.6 Sentence 6 . . . 24
2.4.7 Sentence 7 . . . 28
2.4.8 Sentence 8 . . . 31

3 Computational side of parsing **35**
3.1 User's manual for the parser . . . 35
3.1.1 Starting the parser . . . 35
3.1.2 User interface . . . 35
3.2 Failure scenarios . . . 37
3.2.1 Unknown grammar . . . 38
3.2.2 Missing period . . . 38
3.3 Technical structure of the parser . . . 39
3.4 Foreign source code . . . 39

3.5 Configuration information 40

References **40**

A Source code **40**
A.1 nlp.pl 40
A.2 gui.pl 41
A.3 ppt.pl 45
A.4 input.pl 48
A.5 grammar.pl 50
A.6 lexicon.pl 51

B Parse trees **53**

List of Figures

1 Grammar of the sentence *I love Ann.* 6
2 Parse tree of the first sentence generated by the parser. 7
3 Parser window after parsing the second sentence. 9
4 Parse tree of the second sentence generated by the parser. 10
5 Parser window after parsing the third sentence. 14
6 Parse tree of the third sentence generated by the parser. 15
7 Parse tree of the fourth sentence generated by the parser. 18
8 Parser window after parsing the fifth sentence. 20
9 Parse tree of the fifth sentence generated by the parser. 21
10 Parser window after parsing the sixth sentence. 25
11 Parse tree of the sixth sentence generated by the parser. 26
12 Parser window after parsing the seventh sentence. 28
13 Parse tree of the seventh sentence generated by the parser. . . . 29
14 Parser window after parsing the eighth sentence. 32
15 Parse tree of the eighth sentence generated by the parser. 33
16 User interface of the parser. 36
17 Reaction of the parser to entering a sentence of unknown structure (the entered sentence is German). 37
18 Reaction of the parser to entering a sentence without a concluding period. 38
19 Structure of the program 39
20 Parse tree of the first phrase 54
21 First part of the parse tree of the second phrase ("In this sense war is politics and war itself is a political action...") 54
22 Second part of the parse tree of the second phrase ("...since ancient times there has never been a war that did not have a political character.") 55
23 Parse tree of the third phrase 56
24 Parse tree of the fourth phrase 57
25 Parse tree of the fifth phrase 58
26 Parse tree of the sixth phrase 59
27 Parse tree of the seventh phrase 60
28 Parse tree of the eighth phrase 61

1 Assignment specification

1.1 General description

The software produced in scope of this assignment is a simple text parser, capable of generating a parse tree for a pre-defined story with a fixed grammar and lexicon.

1.2 Chosen text

The parser described in this document was developed for parsing the sentences of the following text:

```
War is the continuation of politics.  In this sense war is politics
and war itself is a political action; since ancient times there has
never been a war that did not have a political character.  But war
has its own particular characteristics and in this sense it cannot
be equated with politics in general.  War is the continuation of politics
by other means.  When politics develops to a certain stage beyond which
it cannot proceed by the usual means, war breaks out to sweep the obstacles
from the way.  When the obstacle is removed and our political aim attained,
the war will stop.  But if the obstacle is not completely swept away,
the war will have to continue till the aim is fully accomplished.  It
can therefore be said that politics is war without bloodshed while
war is politics with bloodshed.
```

This is a simplified fragment of [Mao, 1938, pp. 152–153, 180].

2 Linguistic side of parsing

2.1 Elements of language

In this section the individual elements (words) of the phrasal structure will be presented and briefly explained where necessary. The character sequences in parentheses denote the appropriate abbreviations, which are used later (see section 2.2) for the definition of the grammar of the story to be processed by the parser[1].
The phrasal elements used in this work are:

Sentence (S)

Adverb (Adv) Adverbs are words that usually denote a circumstantiality of some kind, like degree (*more*, *almost*), manner (*fast*), time (*now*, *always*, *never*, *often*), place (*here*), logical relation (*also*), negation (*not*), modality or speaker's attitude (*maybe*; Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Noun (N) Nouns are words that refer to human beings, animals, inanimate objects, matter, actions, properties, times, measure units etc., like *boy*, *horse*,

[1] Same abbreviations are used in the source code files *grammar.pl* and *lexicon.pl* for the definition of grammar rules and lexicon.

mountain, *stone*, *milk*, *song*, *hate*, *redness*, *hour*, *mile* (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Noun phrase (NP) A phrasal unit which consists of a noun and some other phrase structure element. For instance, *the king* is a noun phrase, because it consists of a noun (*king*) and a determiner *the* ([Matthews, 1998, p. 15]).

Determiner (Det) Class of words occurring with nouns often expressing notions of number or quantity, e.g. *a*, *the* ([Matthews, 1998, p. 287]).

Verb (V) Verbs are words that denote various types of actions, like events (*give*, *throw*, *injure*, *disappear*, *put*), processes (*go*, *fall*, *swim*, *float*) or states (*sit*, *own*, *be*). Usually the verb describes what somebody or something is doing (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Verb phrase (VP) A phrasal unit, which consists of a verb and another phrasal element, which may provide additional information about the action described by the verb. In the sentence *The knight challenged the king*, the verb phrase *challenged the king* consists of a verb (*challenged*) and a noun phrase *the king* ([Matthews, 1998, p. 15]).

Conjunction (C) Conjunctions are words that conjoin two or more elements of the same kind, like *and*, *but*, *or* (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Pronoun (PN) Pronouns are words that indicate how a referent can be identified (or not identified) in its context, or that characterize the referent with respect to amount or number: *I*, *me* - *my*; *you* - *your*, *he*, *him*, *his*; *this*, *that*, *these*, *those*; *all*, *both*, *everybody*, *some*, *any*, *few*, *many* etc. (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso)

Adjective (Adj) Adjectives are words that usually denote permanent or temporary qualities of different kinds: *red*, *blue*, *tall*, *short*, *fat*, *happy*, *like*, *delicious*, *angry*, *dead* (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Subjunction (Subj) Subjunctions are words that embed a clause within another clause, like *that*, *since*, *if*, *than*, *although* (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Particle (Part) Examples of phrasal particles are *to* and *out*.

Preposition (P) Prepositions are words that denote a relation between two referents, either alone (*the wheels of the car*) or together with a verb, an adjective or a participle (*the wheels*, *sitting on the car*). Some frequently used prepositions are: *at*, *between*, *by*, *for*, *from*, *in*, *on*, *to*, *with* (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Sentential adverbial (SA) Sentential adverbials modify the content of the clause or convey the speaker's comment on the content of what he is saying, e.g. *of course* in *Of course, nobody will listen to him* (Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso).

Prepositional phrase (PP)

Adverbial phrase (AP)

Pronoun phrase (PNP)

2.2 Phrase structure representation

A *grammar* is a systematic description of a language. It usually includes statements about the vocabulary, *phonology* (sound system of the language), *morphology* (internal structure of words), *syntax* (information about possible combinations of words) and *semantics* (linguistic meaning) of the language ([Matthews, 1998, pp. 288, 15]).
In this work, grammar usually refers to the syntactic structure of the language. In this section, the means for representation of syntactic structure of sentences, which is used in this work, will be outlined. *Lexicon* refers to a data set containing the vocabulary and the basic classification of words (into nouns, verbs etc.).
In this work, the structure of individual sentences of the text is described by means of the so-called *context-free* rules. Consider the sentence, *I love Ann.* Using the context-free rules, the structure of this sentence (grammar part) can be written as

$$\begin{aligned} &\mathrm{S} \rightarrow \mathrm{PN\,VP}\,. \\ &\mathrm{VP} \rightarrow \mathrm{V\,NP} \end{aligned}$$

and the lexicon part as

$$\begin{aligned} &\mathrm{PN} \rightarrow \mathrm{i} \\ &\mathrm{V} \rightarrow \mathrm{love} \\ &\mathrm{NP} \rightarrow \mathrm{ann} \end{aligned}$$

because the sentence consists of a pronoun *I* and a verb phrase *love Ann* (see figure 1). These rules are called context-free because, they apply always. So, $\mathrm{VP} \rightarrow \mathrm{V\,NP}$ states that any VP everywhere can consist of a verb followed by a noun phrase ([Matthews, 1998, p. 15]).
In contrast, *context-sensitive* rules are valid only under certain conditions. For instance, the following rule (for some imaginary language) states that a NP can consist of a determiner followed by a noun if it is preceded by a PP and followed by an AP ([Matthews, 1998, p. 15]):

$$\mathrm{PP\,NP\,AP} \rightarrow \mathrm{PP\,Det\,N\,AP}$$

2.3 Representation of grammatical rules in Prolog

Many Prolog systems, including the one used for this work (LPA Win-Prolog 3.3), have built-in support for natural language processing and relieve the developer from implementing the actual parsing routines. Usually, only the definition of grammar and lexicon must be performed manually.
The Prolog system used in scope of this assignment requires the programmer to define the rules using the so-called *definite clause grammar* (DCG) notation. For instance, the sentence structure given in section 2.2 is defined in Prolog as:

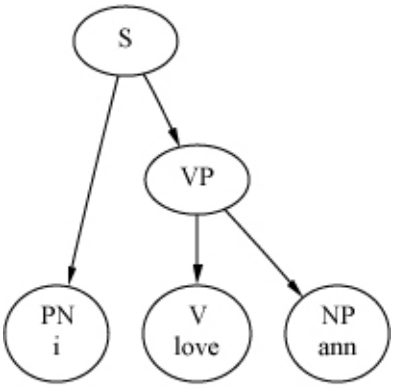

Figure 1: Grammar of the sentence *I love Ann.*

```
s(s(Pn,Vp)) --> pn(Pn),vp(Vp).
vp(vp(V,Np)) --> v(V),np(Np).
```

for the grammar and

```
pn(pn(i)) --> [i].
v(v(love)) --> [love].
np(np(ann)) --> [ann].
```

for the lexicon. For further details on the DCG notation in general and LPA Prolog built-in natural language processing support the reader should refer to [Matthews, 1998, pp185–214] and [Shalfield, 2001, pp56–70] respectively.

2.4 Definition of the grammar for the chosen text

In this section, the grammar and lexicon definitions for the chosen text will be given sentence by sentence.

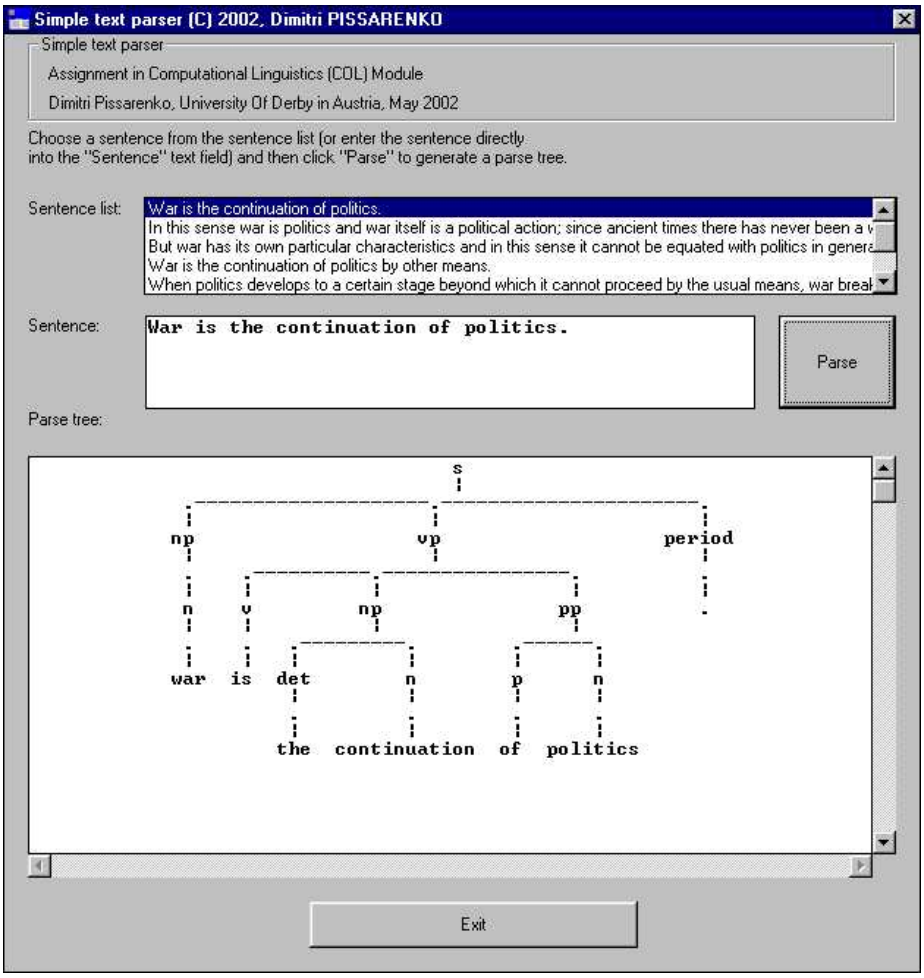

Figure 2: Parse tree of the first sentence generated by the parser.

2.4.1 Sentence 1

Sentence: *War is the continuation of politics.*
Context-free grammar:

$$\begin{aligned}
&S \rightarrow NP\,VP\,.\\
&NP \rightarrow N\\
&VP \rightarrow V\,NP\,PP\\
&NP \rightarrow Det\,N\\
&PP \rightarrow P\,N\\
&N \rightarrow \text{war}\\
&N \rightarrow \text{continuation}\\
&N \rightarrow \text{politics}\\
&V \rightarrow \text{is}\\
&P \rightarrow \text{of}\\
&Det \rightarrow \text{the}
\end{aligned}$$

Expected (correct) parse tree: See figure 20.
Parse tree generated by the parser: See figure 2.
Grammar (DCG):

```
s(s(Np,Vp,Period))-->np(Np), vp(Vp),period(Period).

np(np(N)) --> n(N).

vp(vp(V,Np,Pp)) --> v(V), np(Np), pp(Pp).

np(np(Det,N)) --> det(Det), n(N).

pp(pp(P, N)) --> p(P), n(N).
```

Lexicon (DCG):

```
n(n(war)) --> [war].

n(n(continuation)) --> [continuation].

n(n(politics)) --> [politics].

v(v(is)) --> [is].

p(p(of)) -->[of].

det(det(the)) --> [the].
```

2.4.2 Sentence 2

Sentence: *In this sense war is politics and war itself is a political action; since ancient times there has never been a war that did not have a political character.*

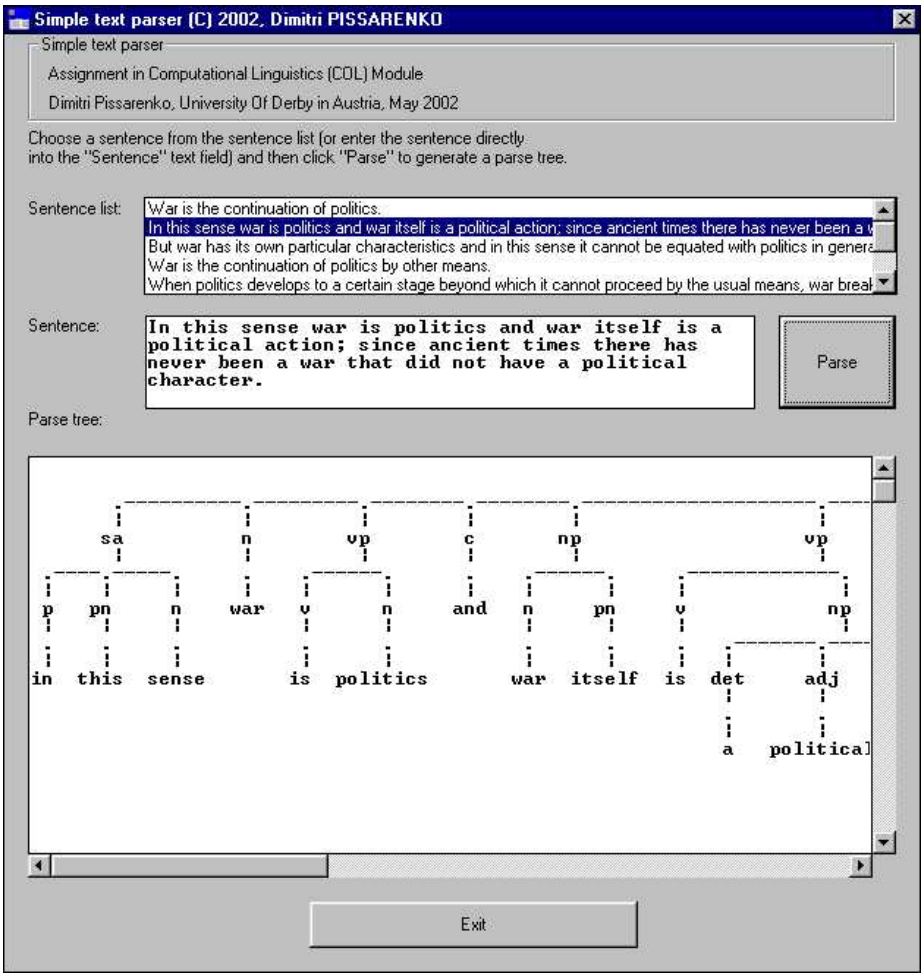

Figure 3: Parser window after parsing the second sentence.

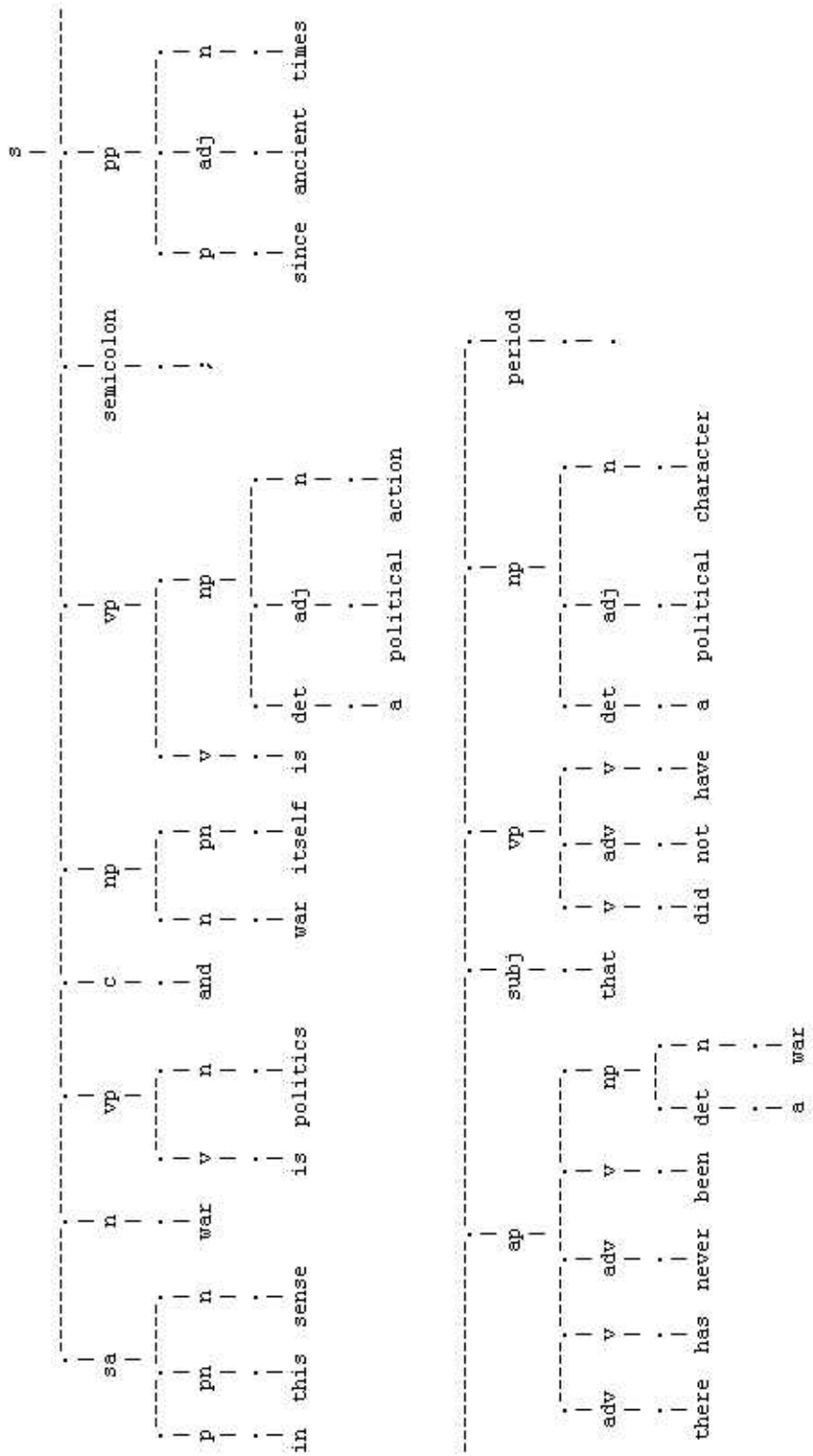

Figure 4: Parse tree of the second sentence generated by the parser.

Context-free grammar:

S → SA N VP C NP VP; PP AP Subj VP NP .
SA → P PN N
VP → V N
NP → N PN
VP → V NP
NP → Det Adj N
PP → P Adj N
AP → Adv V Adv V NP
VP → V Adv V
NP → Det Adj N

P → in
PN → this
N → sense
N → war
V → is
N → politics
C → and
PN → itself
Det → a
Adj → political
N → action
P → since
Adj → ancient
N → times
Adv → there
V → has
Adv → never
V → been
Subj → that
V → did
Adv → not
V → have
Adj → political
N → character

Expected (correct) parse tree: See figures 21 and 22.
Parse tree generated by the parser: See figures 3 and 4.
Grammar (DCG):

```
s(s(Sa,N,Vp0,C,Np0,Vp1,Semicolon,Pp,Ap,Subj,Vp2,Np1,Period)) -->
    sa(Sa),n(N),vp(Vp0),c(C),np(Np0),vp(Vp1),semicolon(Semicolon),
    pp(Pp),ap(Ap),subj(Subj),vp(Vp2),np(Np1),period(Period).

sa(sa(P,Pn,N)) --> p(P), pn(Pn), n(N).

vp(vp(V, N)) --> v(V), n(N).

np(np(N, Pn)) --> n(N), pn(Pn).

vp(vp(V, Np)) --> v(V), np(Np).

np(np(Det,Adj,N)) --> det(Det),adj(Adj),n(N).

pp(pp(P,Adj,N)) --> p(P), adj(Adj), n(N).

ap(ap(Adv0,V0,Adv1,V1,Np)) --> adv(Adv0), v(V0), adv(Adv1), v(V1),
np(Np).

vp(vp(V0,Adv,V1)) --> v(V0), adv(Adv), v(V1).

np(np(Det,Adj,N)) --> det(Det), adj(Adj), n(N).
```

Lexicon (DCG):

```
p(p(in)) --> [in].

pn(pn(this)) --> [this].

n(n(sense)) --> [sense].

n(n(war)) --> [war].

v(v(is)) --> [is].

n(n(politics)) --> [politics].

c(c(and)) --> [and].

pn(pn(itself)) --> [itself].

det(det(a)) --> [a].

adj(adj(political)) --> [political].

n(n(action)) --> [action].

p(p(since)) --> [since].

adj(adj(ancient)) --> [ancient].
```

```
n(n(times)) --> [times].

adv(adv(there)) --> [there].

v(v(has)) --> [has].

adv(adv(never)) --> [never].

v(v(been)) --> [been].

subj(subj(that)) --> [that].

v(v(did)) --> [did].

adv(adv(not)) --> [not].

v(v(have)) --> [have].

adj(adj(political)) --> [political].

n(n(character)) --> [character].
```

2.4.3 Sentence 3

Sentence: *But war has its own particular characteristics and in this sense it cannot be equated with politics in general.*
Context-free grammar:

$$
\begin{aligned}
&\mathrm{S} \rightarrow \mathrm{C\,N\,VP\,C\,AP\,PN\,VP\,AP\,.}\\
&\mathrm{VP} \rightarrow \mathrm{V\,PNP\,NP}\\
&\mathrm{PNP} \rightarrow \mathrm{PN\,PN}\\
&\mathrm{NP} \rightarrow \mathrm{Adj\,N}\\
&\mathrm{AP} \rightarrow \mathrm{P\,PN\,N}\\
&\mathrm{VP} \rightarrow \mathrm{V\,V\,V}\\
&\mathrm{AP} \rightarrow \mathrm{P\,NP}\\
&\mathrm{NP} \rightarrow \mathrm{N\,AP}\\
&\mathrm{AP} \rightarrow \mathrm{P\,Adj}
\end{aligned}
$$

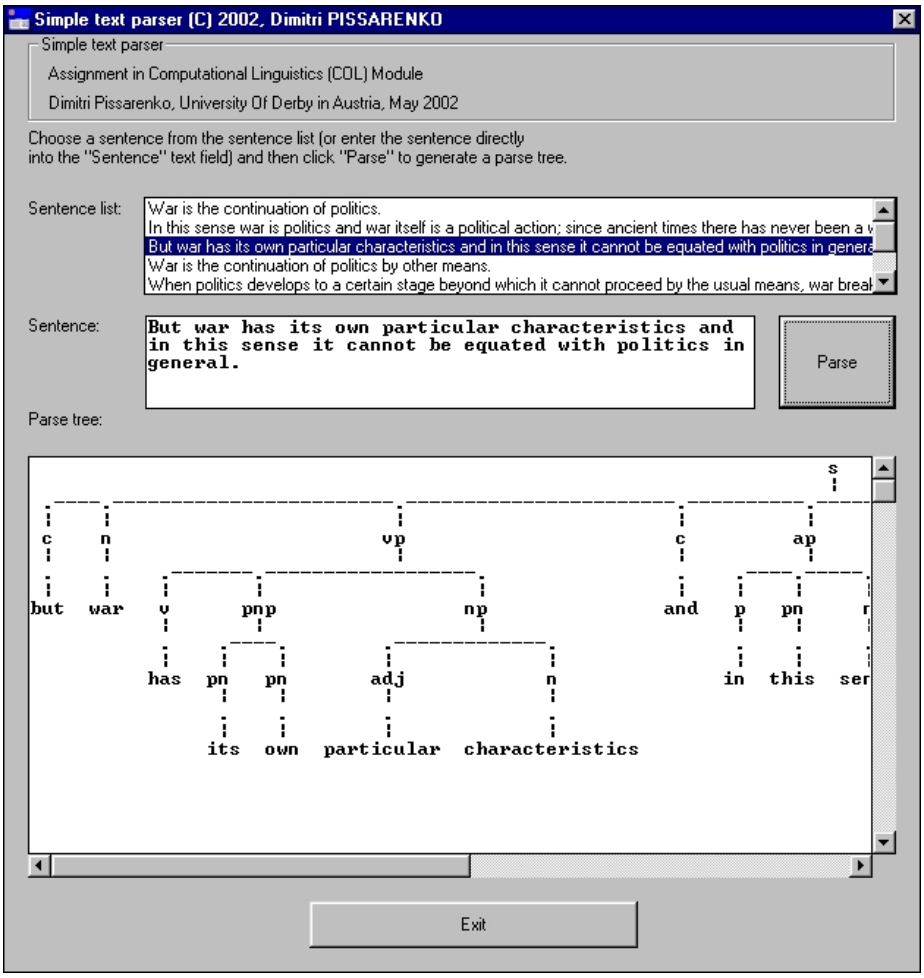

Figure 5: Parser window after parsing the third sentence.

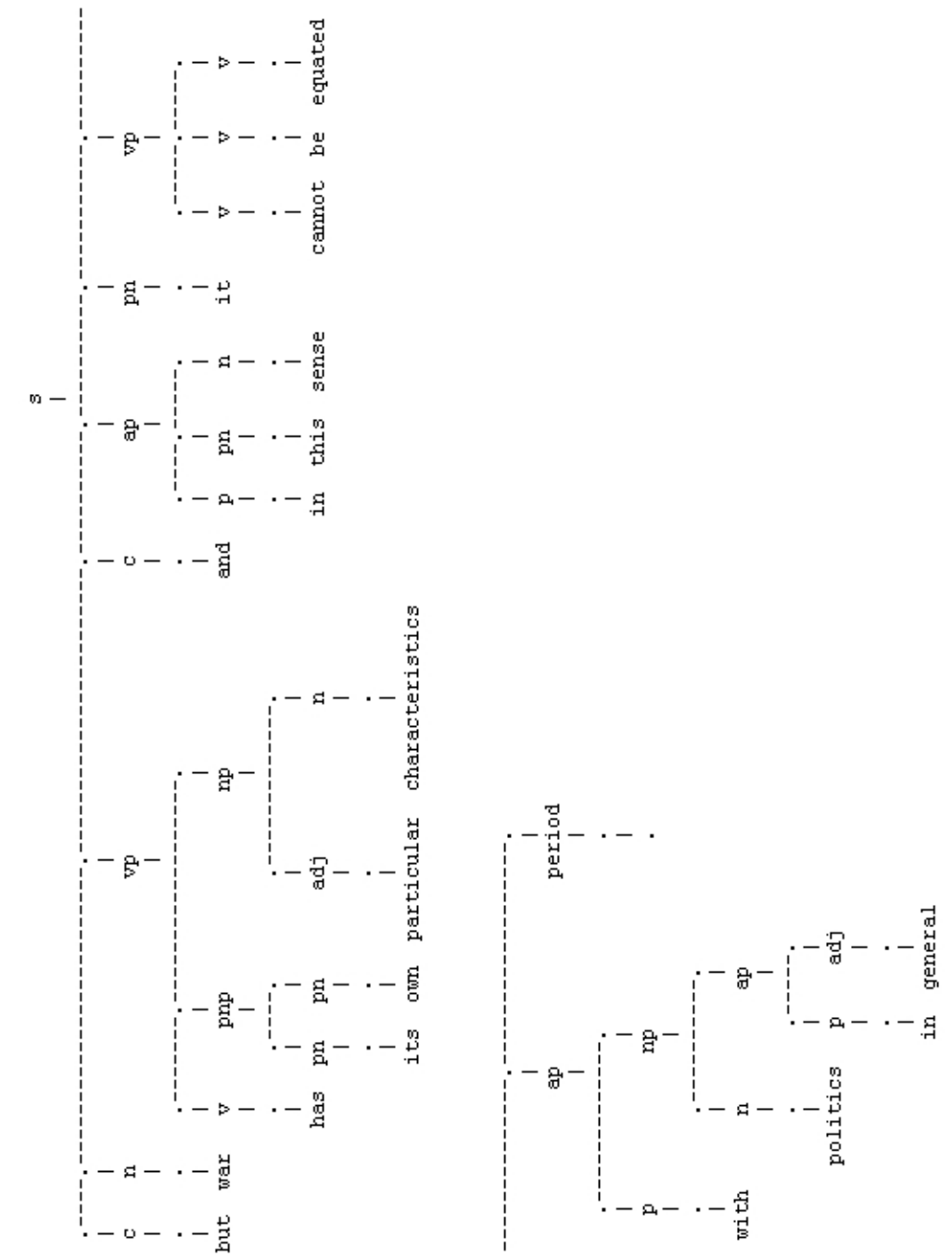

Figure 6: Parse tree of the third sentence generated by the parser.

C → but
N → war
V → has
PN → its
PN → own
Adj → particular
N → characteristics
C → and
P → in
PN → this
N → sense
PN → it
V → cannot
V → be
V → equated
P → with
N → politics
P → in
Adj → general

Expected (correct) parse tree: See figure 23.
Parse tree generated by the parser: See figures 5 and 6.
Grammar (DCG):

```
s(s(C0,N,Vp0,C1,Ap0,Pn,Vp1,Ap1,Period)) --> c(C0), n(N), vp(Vp0),
c(C1),ap(Ap0), pn(Pn), vp(Vp1), ap(Ap1), period(Period).

vp(vp(V,Pnp,Np)) --> v(V), pnp(Pnp), np(Np).

pnp(pnp(Pn0,Pn1)) --> pn(Pn0), pn(Pn1).

np(np(Adj,N)) --> adj(Adj), n(N).

ap(ap(P,Pn,N)) --> p(P), pn(Pn), n(N).

vp(vp(V0,V1,V2)) --> v(V0), v(V1), v(V2).

ap(ap(P, Np)) --> p(P), np(Np).

np(np(N, Ap)) --> n(N), ap(Ap).

ap(ap(P, Adj)) --> p(P), adj(Adj).
```

Lexicon (DCG):

```
c(c(but)) --> [but].
```

```
n(n(war)) --> [war].

v(v(has)) --> [has].

pn(pn(its)) --> [its].

pn(pn(own)) --> [own].

adj(adj(particular)) --> [particular].

n(n(characteristics)) --> [characteristics].

c(c(and)) --> [and].

p(p(in)) --> [in].

pn(pn(this)) --> [this].

n(n(sense)) --> [sense].

pn(pn(it)) --> [it].

v(v(cannot)) --> [cannot].

v(v(be)) --> [be].

v(v(equated)) --> [equated].

p(p(with)) --> [with].

n(n(politics)) --> [politics].

p(p(in)) --> [in].

adj(adj(general)) --> [general].
```

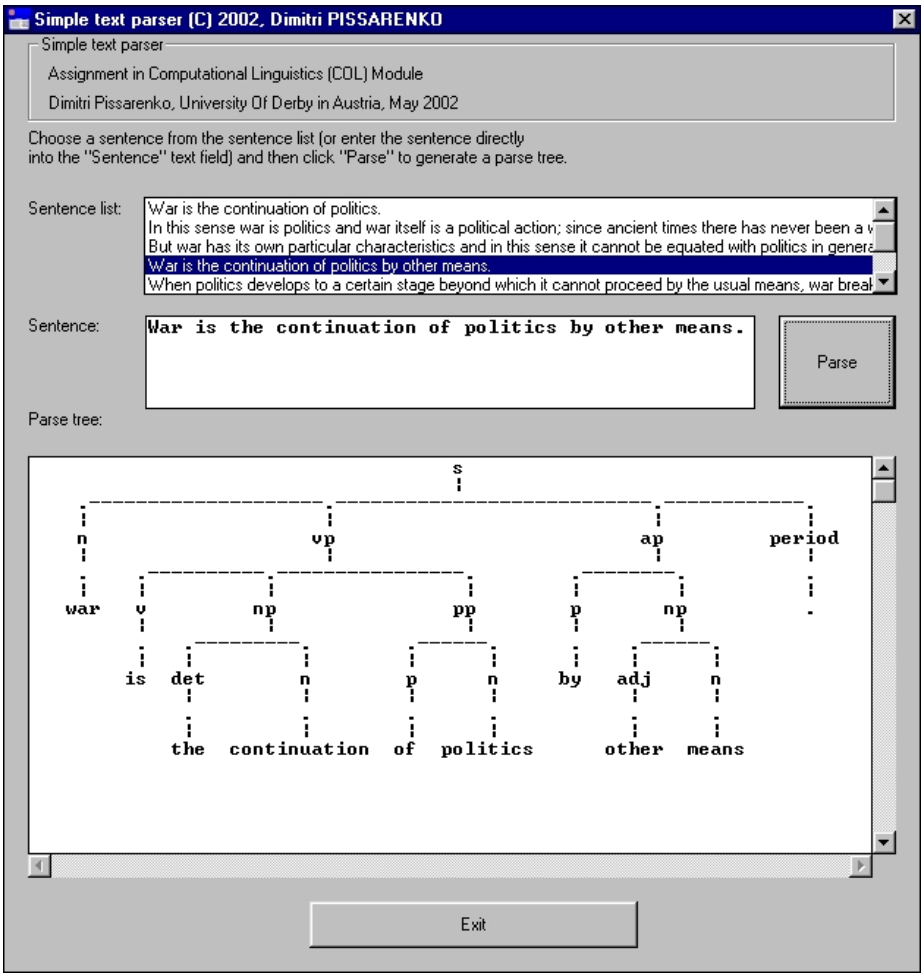

Figure 7: Parse tree of the fourth sentence generated by the parser.

2.4.4 Sentence 4

Sentence: *War is the continuation of politics by other means.*
Context-free grammar:

$$
\begin{aligned}
&S \rightarrow N\,VP\,AP\,.\\
&VP \rightarrow V\,NP\\
&NP \rightarrow N\,PP\\
&PP \rightarrow P\,N\\
&AP \rightarrow P\,Adj\,N\\
&N \rightarrow \text{war}\\
&V \rightarrow \text{is}\\
&N \rightarrow \text{continuation}\\
&P \rightarrow \text{of}\\
&N \rightarrow \text{politics}\\
&P \rightarrow \text{by}\\
&Adj \rightarrow \text{other}\\
&N \rightarrow \text{means}
\end{aligned}
$$

Expected (correct) parse tree: See figure 24.
Parse tree generated by the parser: See figure 7.
Grammar (DCG):

```
s(s(N,Vp,Ap,Period)) --> n(N), vp(Vp), ap(Ap), period(Period).

vp(vp(V,Np)) --> v(V), np(Np).

np(np(N,Pp)) --> n(N), pp(Pp).

pp(pp(P, N)) --> p(P), n(N).

ap(ap(P,Adj,N)) --> p(P), adj(Adj), n(N).
```

Lexicon (DCG):

```
n(n(war)) --> [war].

v(v(is)) --> [is].

n(n(continuation)) --> [continuation].

p(p(of)) --> [of].

n(n(politics)) --> [politics].

p(p(by)) --> [by].

adj(adj(other)) --> [other].
```

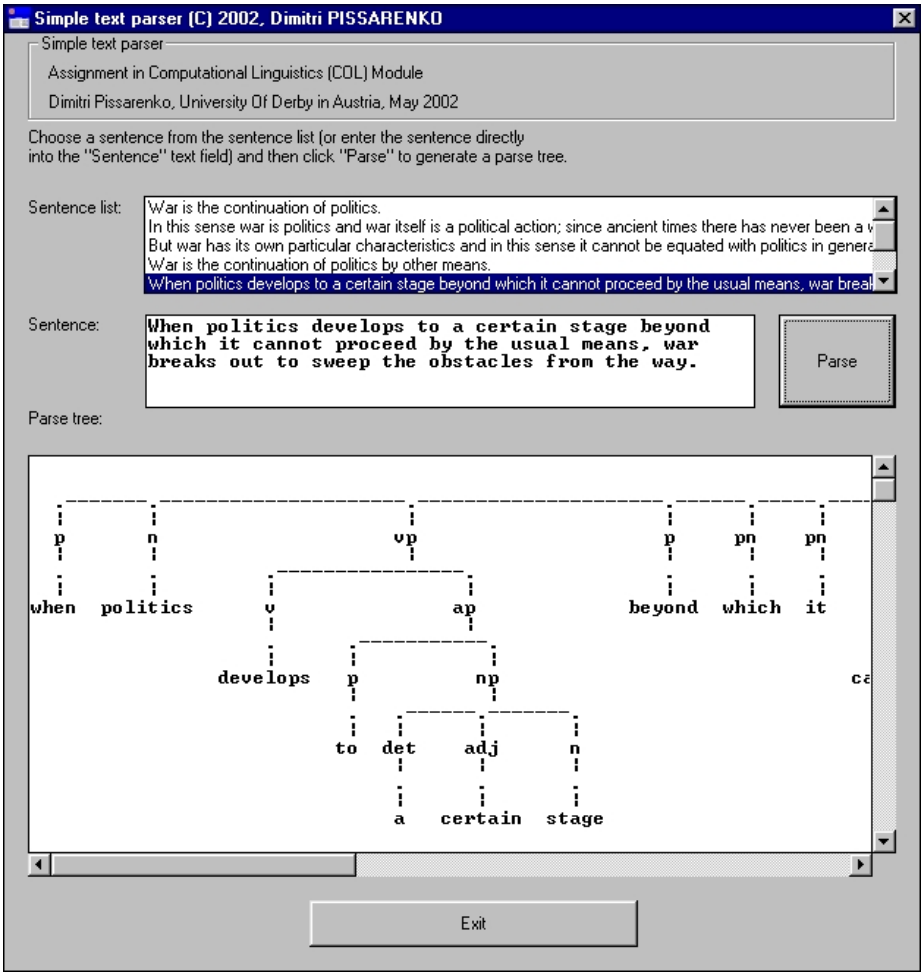

Figure 8: Parser window after parsing the fifth sentence.

```
n(n(means)) --> [means].
```

2.4.5 Sentence 5

Sentence: *When politics develops to a certain stage beyond which it cannot proceed by the usual means, war breaks out to sweep the obstacles from the way.*

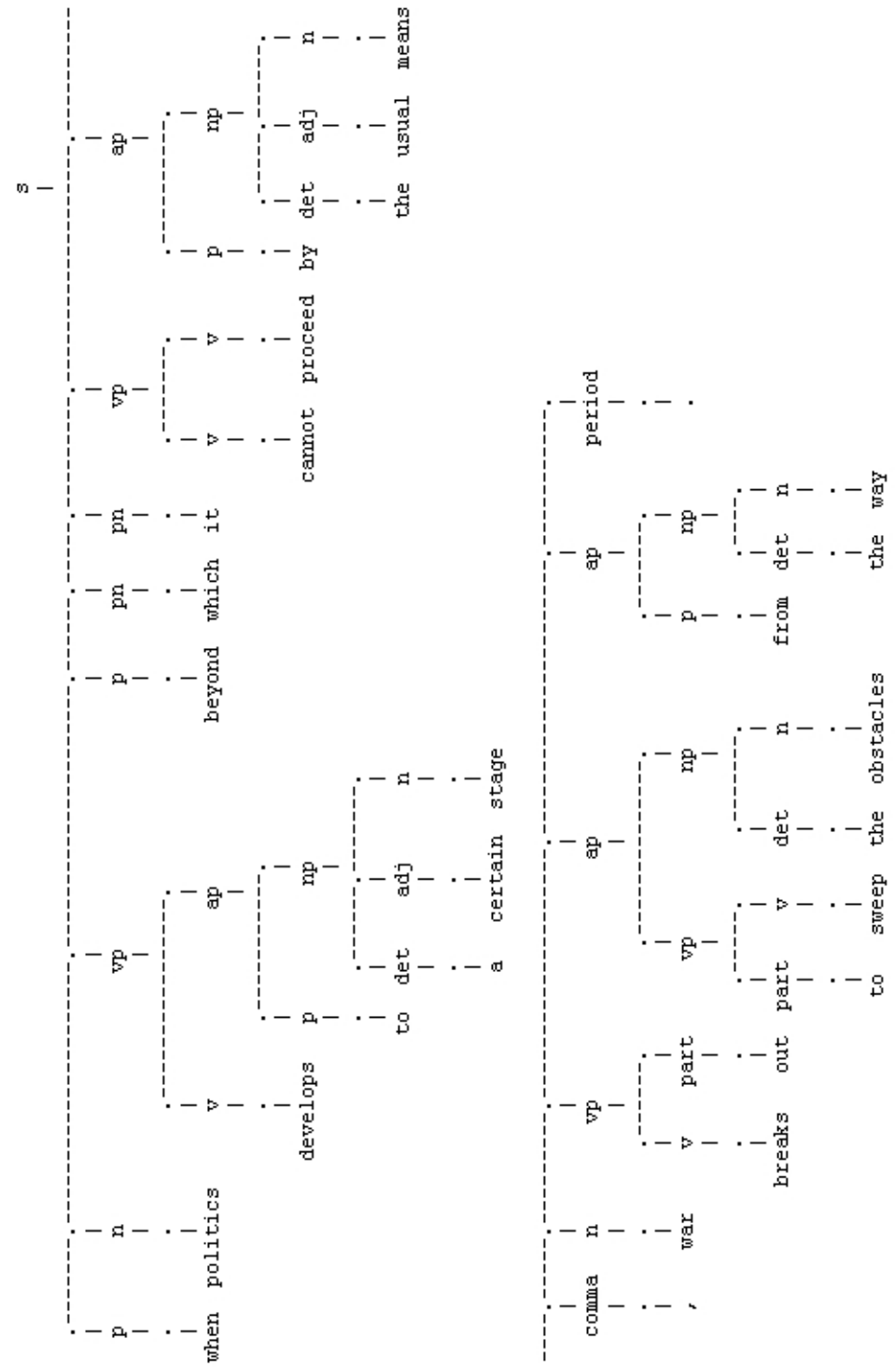

Figure 9: Parse tree of the fifth sentence generated by the parser.

Context-free grammar:

S → P N VP P PN PN VP AP, N VP AP AP .
VP → V AP
AP → Part NP
NP → Det Adj N
VP → V V
AP → P NP
NP → Det Adj NP
VP → V Part
AP → VP NP
VP → Part V
NP → Det N
AP → P NP
NP → Det N

P → when
N → politics
V → develops
P → to
Det → a
Adj → certain
N → stage
P → beyond
PN → which
PN → it
V → cannot
V → proceed
P → by
Adj → usual
N → means
V → breaks
Part → out
Part → to
V → sweep
N → osbtacles
P → from
N → way

Expected (correct) parse tree: See figure 25.

Parse tree generated by the parser: See figures 8 and 9.
Grammar (DCG):

```
s(s(P0,N0,Vp0,P1,Pn0,Pn1,Vp1,Ap0,Comma,N1,Vp2,Ap1,Ap2, Period))
-->
p(P0),n(N0),vp(Vp0),p(P1),pn(Pn0),pn(Pn1),vp(Vp1),ap(Ap0),
comma(Comma),n(N1),vp(Vp2),ap(Ap1),ap(Ap2),period(Period).

vp(vp(V,Ap)) --> v(V), ap(Ap).

ap(ap(Part,Np)) --> part(Part),np(Np).

np(np(Det,Adj,N)) --> det(Det), adj(Adj), n(N).

vp(vp(V0, V1)) --> v(V0), v(V1).

ap(ap(P,Np)) --> p(P), np(Np).

np(np(Det,Adj,N)) --> det(Det),adj(Adj),n(N).

vp(vp(V,Part)) --> v(V), part(Part).

ap(ap(Vp,Np)) --> vp(Vp), np(Np).

vp(vp(Part, V)) --> part(Part), v(V).

np(np(Det, N)) --> det(Det), n(N).

ap(ap(P,Np)) --> p(P), np(Np).

np(np(Det, N)) --> det(Det), n(N).
```

Lexicon (DCG):

```
p(p(when)) --> [when].

n(n(politics)) --> [politics].

v(v(develops)) --> [develops].

p(p(to)) --> [to].

det(det(a)) --> [a].

adj(adj(certain)) --> [certain].

n(n(stage)) --> [stage].

p(p(beyond)) --> [beyond].

pn(pn(which)) --> [which].
```

```
pn(pn(it)) --> [it].

v(v(cannot)) --> [cannot].

v(v(proceed)) --> [proceed].

p(p(by)) --> [by].

adj(adj(usual)) --> [usual].

n(n(means)) --> [means].

v(v(breaks)) --> [breaks].

part(part(out)) --> [out].

part(part(to)) --> [to].

v(v(sweep)) --> [sweep].

n(n(obstacles)) --> [obstacles].

p(p(from)) --> [from].

n(n(way)) --> [way].
```

2.4.6 Sentence 6

Sentence: *When the obstacle is removed and our political aim attained, the war will stop.*
Context-free grammar:

$$
\begin{aligned}
&\mathrm{S} \rightarrow \mathrm{AP}, \mathrm{NP}\,\mathrm{VP}\,. \\
&\mathrm{AP} \rightarrow \mathrm{Adv}\,\mathrm{NP}\,\mathrm{VP}\,\mathrm{C}\,\mathrm{VP} \\
&\mathrm{VP} \rightarrow \mathrm{V}\,\mathrm{V} \\
&\mathrm{VP} \rightarrow \mathrm{NP}\,\mathrm{V} \\
&\mathrm{NP} \rightarrow \mathrm{PN}\,\mathrm{Adj}\,\mathrm{N}
\end{aligned}
$$

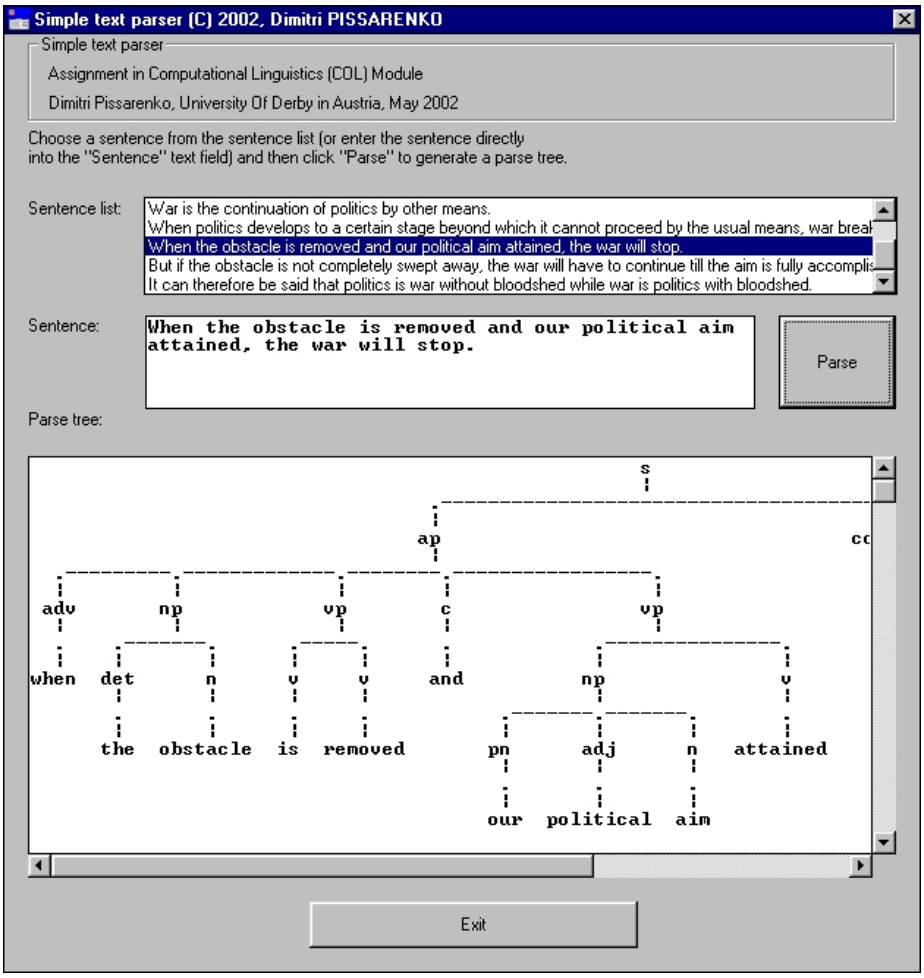

Figure 10: Parser window after parsing the sixth sentence.

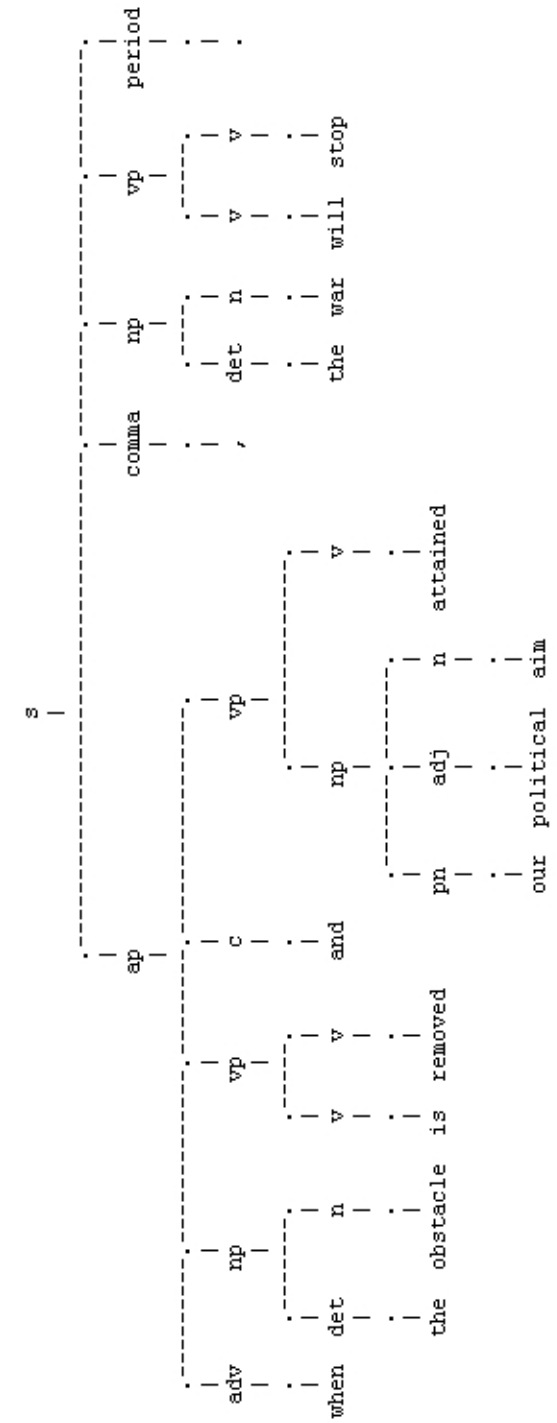

Figure 11: Parse tree of the sixth sentence generated by the parser.

Adv → when
Det → the
N → obstacle
V → is
V → removed
C → and
PN → our
Adj → political
N → aim
V → attained
N → war
V → will
V → stop

Expected (correct) parse tree: See figure 26.
Parse tree generated by the parser: See figures 10 and 11.
Grammar (DCG):

```
s(s(Ap,Comma,Np,Vp,Period)) --> ap(Ap), comma(Comma), np(Np),
vp(Vp),period(Period).

ap(ap(Adv,Np,Vp0,C,Vp1)) --> adv(Adv),np(Np),vp(Vp0),c(C),vp(Vp1).

vp(vp(V0,V1)) --> v(V0), v(V1).

vp(vp(Np,V)) --> np(Np), v(V).

np(np(Pn,Adj,N)) --> pn(Pn), adj(Adj), n(N).
```

Lexicon (DCG):

```
adv(adv(when)) --> [when].

det(det(the)) --> [the].

n(n(obstacle)) --> [obstacle].

v(v(is)) --> [is].

v(v(removed)) --> [removed].

c(c(and)) --> [and].

pn(pn(our)) --> [our].

adj(adj(political)) --> [political].
```

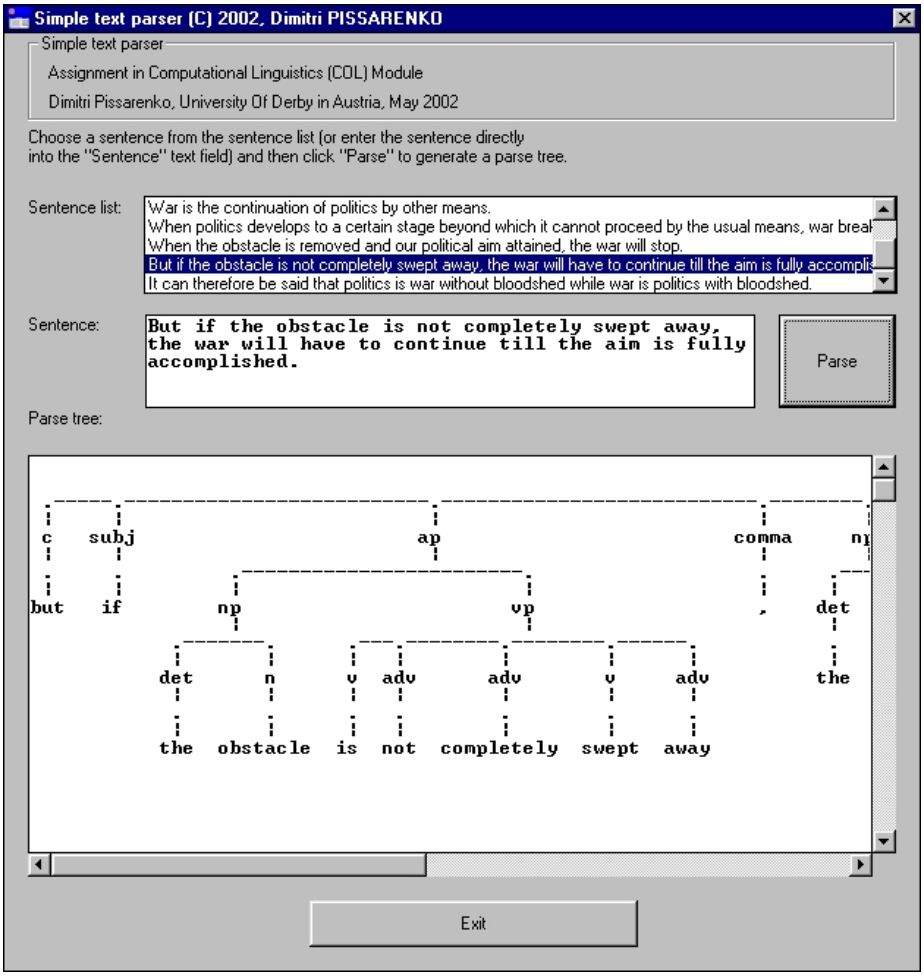

Figure 12: Parser window after parsing the seventh sentence.

```
n(n(aim)) --> [aim].

v(v(attained)) --> [attained].

n(n(war)) --> [war].

v(v(will)) --> [will].

v(v(stop)) --> [stop].
```

2.4.7 Sentence 7

Sentence: *But if the obstacle is not completely swept away, the war will have to continue till the aim is fully accomplished.*

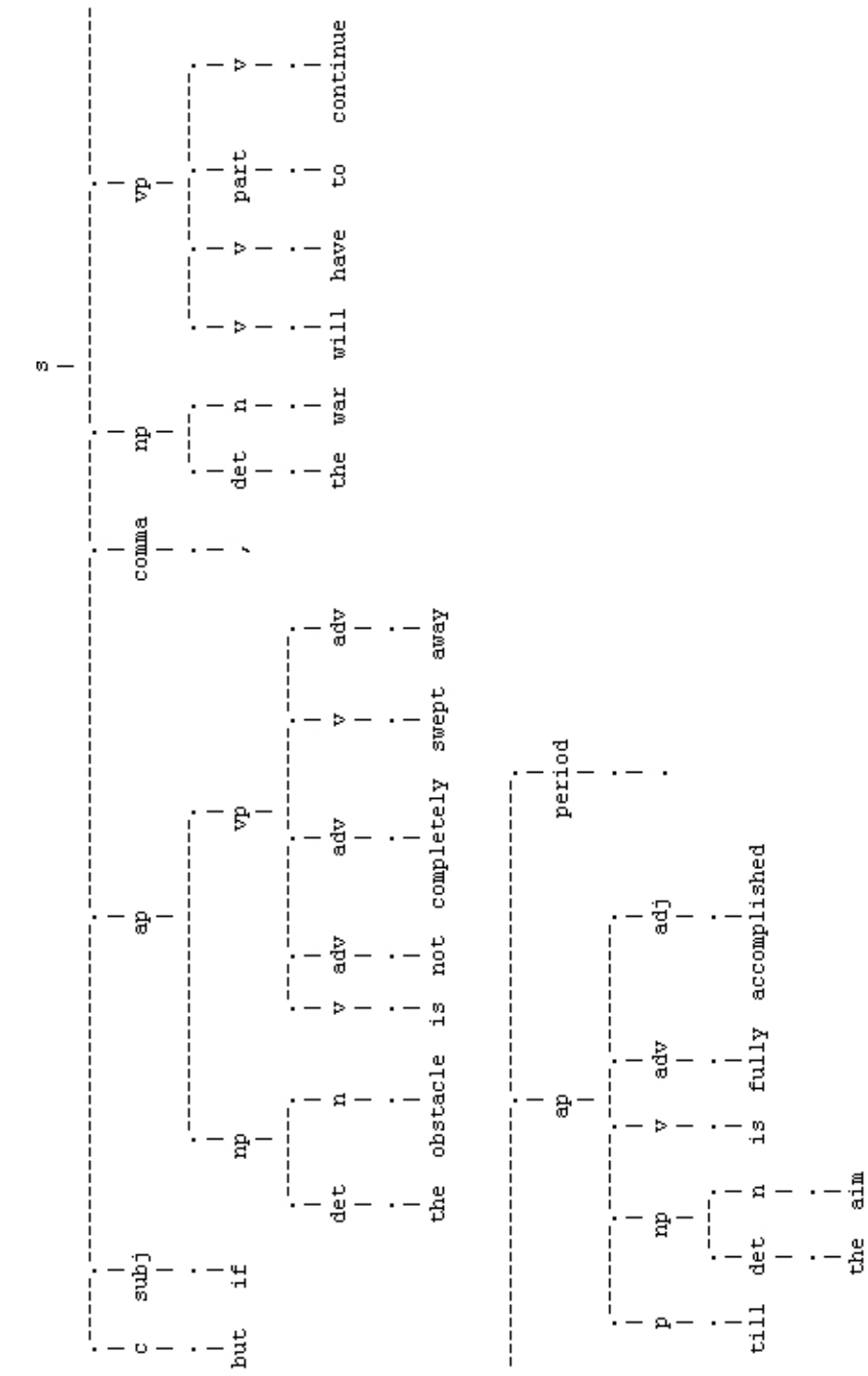

Figure 13: Parse tree of the seventh sentence generated by the parser.

Context-free grammar:

$S \rightarrow C\,Subj\,AP, NP\,VP\,AP\,.$
$AP \rightarrow NP\,VP$
$NP \rightarrow Det\,N$
$VP \rightarrow V\,Adv\,Adv\,V\,Adv$
$VP \rightarrow V\,V\,Part\,V$
$AP \rightarrow P\,NP\,V\,Adv\,Adj$
$NP \rightarrow Det\,N$
$C \rightarrow but$
$Subj \rightarrow if$
$N \rightarrow obstacle$
$Adv \rightarrow not$
$Adv \rightarrow completely$
$V \rightarrow swept$
$Adv \rightarrow away$
$V \rightarrow will$
$V \rightarrow have$
$Part \rightarrow to$
$V \rightarrow continue$
$P \rightarrow till$
$N \rightarrow aim$
$Adv \rightarrow fully$
$Adj \rightarrow accomplished$

Expected (correct) parse tree: See figure 27.
Parse tree generated by the parser: See figures 12 and 13.
Grammar (DCG):

```
s(s(C,Subj,Ap0,Comma,Np,Vp,Ap1,Period)) --> c(C), subj(Subj),
ap(Ap0),comma(Comma), np(Np),vp(Vp), ap(Ap1),period(Period).

ap(ap(Np,Vp)) --> np(Np), vp(Vp).

np(np(Det,N)) --> det(Det), n(N).

vp(vp(V0,Adv0,Adv1,V1,Adv2)) --> v(V0),
adv(Adv0),adv(Adv1),v(V1),adv(Adv2).

vp(vp(V0,V1,Part,V2)) --> v(V0), v(V1), part(Part), v(V2).

ap(ap(P,Np,V,Adv,Adj)) --> p(P), np(Np), v(V), adv(Adv), adj(Adj).

np(np(Det,N)) --> det(Det), n(N).
```

Lexicon (DCG):

```
c(c(but)) --> [but].

subj(subj(if)) --> [if].

n(n(obstacle)) --> [obstacle].

adv(adv(not)) --> [not].

adv(adv(completely)) --> [completely].

v(v(swept)) --> [swept].

adv(adv(away)) --> [away].

v(v(will)) --> [will].

v(v(have)) --> [have].

part(part(to)) --> [to].

v(v(continue)) --> [continue].

p(p(till)) --> [till].

n(n(aim)) --> [aim].

adv(adv(fully)) --> [fully].

adj(adj(accomplished)) --> [accomplished].
```

2.4.8 Sentence 8

Sentence: *It can therefore be said that politics is war without bloodshed while war is politics with bloodshed.*

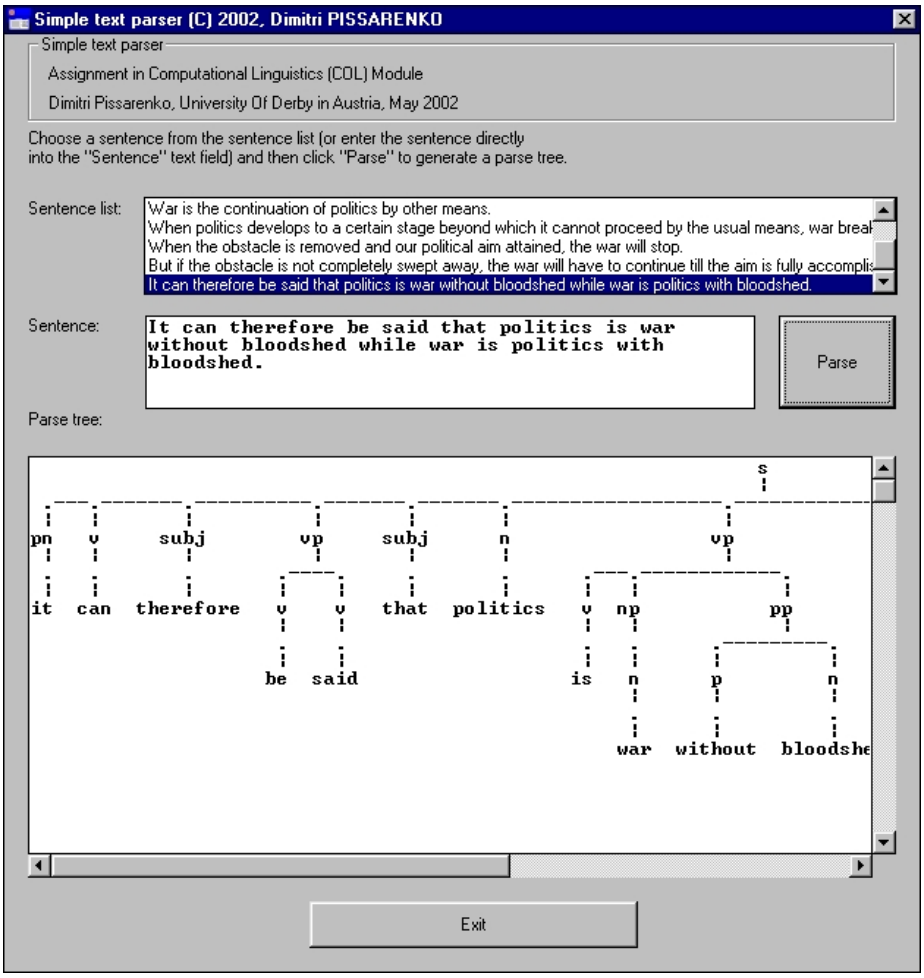

Figure 14: Parser window after parsing the eighth sentence.

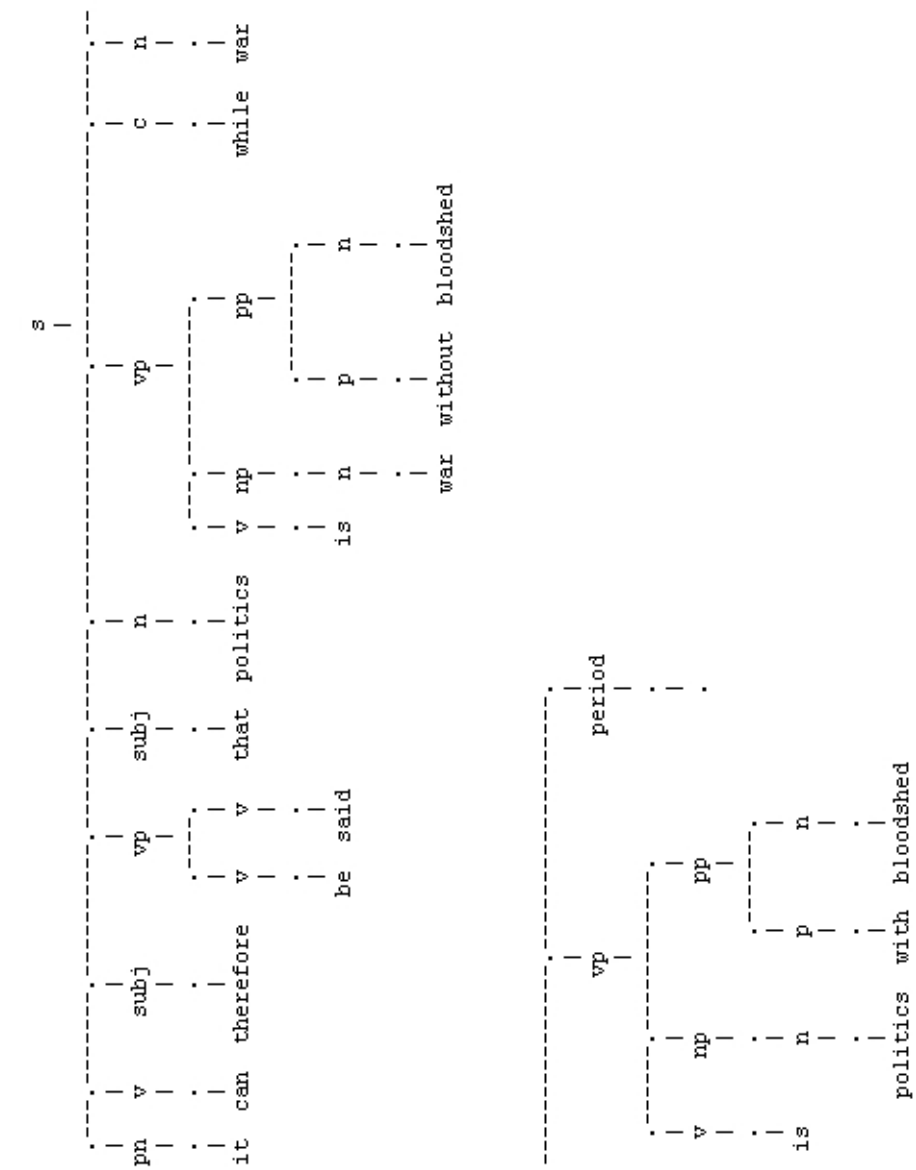

Figure 15: Parse tree of the eighth sentence generated by the parser.

Context-free grammar:

S → PN V Subj VP Subj N VP C N VP .
VP → V V
VP → V N PP
VP → V NP
NP → N PP
PP → P N
PN → it
V → can
Subj → therefore
V → be
V → said
Subj → that
N → politics
V → is
N → war
P → without
N → bloodshed
C → while
N → politics
P → with

Expected (correct) parse tree: See figure 28.
Parse tree generated by the parser: See figures 14 and 15.
Grammar (DCG):

```
s(s(Pn,V,Subj0,Vp0,Subj1,N0,Vp1,C,N1,Vp2,Period)) --> pn(Pn),v(V),
subj(Subj0),vp(Vp0),subj(Subj1),n(N0),vp(Vp1),c(C),n(N1),vp(Vp2),
period(Period).

vp(vp(V0,V1)) --> v(V0), v(V1).

vp(vp(V,N,Pp)) --> v(V),n(N),pp(Pp).

vp(vp(V,Np)) --> v(V), np(Np).

np(np(N,Pp)) --> n(N), pp(Pp).
```

Lexicon (DCG):

```
pn(pn(it)) --> [it].

v(v(can)) --> [can].

subj(subj(therefore)) --> [therefore].
```

```
v(v(be)) --> [be].

v(v(said)) --> [said].

subj(subj(that)) --> [that].

n(n(politics)) --> [politics].

v(v(is)) --> [is].

n(n(war)) --> [war].

p(p(without)) --> [without].

n(n(bloodshed)) --> [bloodshed].

c(c(while)) --> [while].

n(n(politics)) --> [politics].

p(p(with)) --> [with].

n(n(bloodshed)) --> [bloodshed].
```

3 Computational side of parsing

3.1 User's manual for the parser

3.1.1 Starting the parser

In order to launch the parser under LPA Prolog 3.3, the following steps should be performed:

1. Start the LPA Prolog 3.3 development environment.
2. Select the menu item *Open...* from the menu *File*.
3. In the following dialog box, select the file *nlp.pl* and click the *OK* button.
4. Select the menu item *Compile* from the menu *Run*.

Upon completing these steps, the parser window is shown.

3.1.2 User interface

The structure of the user interface is given in figure 16. The sentence to be parsed must be entered into the *sentence text field*. By pressing the *"Parse" button* the sentence is parsed and the parse tree is printed in the *parse tree output area*. If an error occurs during parsing, the error message is output in this area, either.
For convenience reasons, a *sentence list* is provided, from which the user can

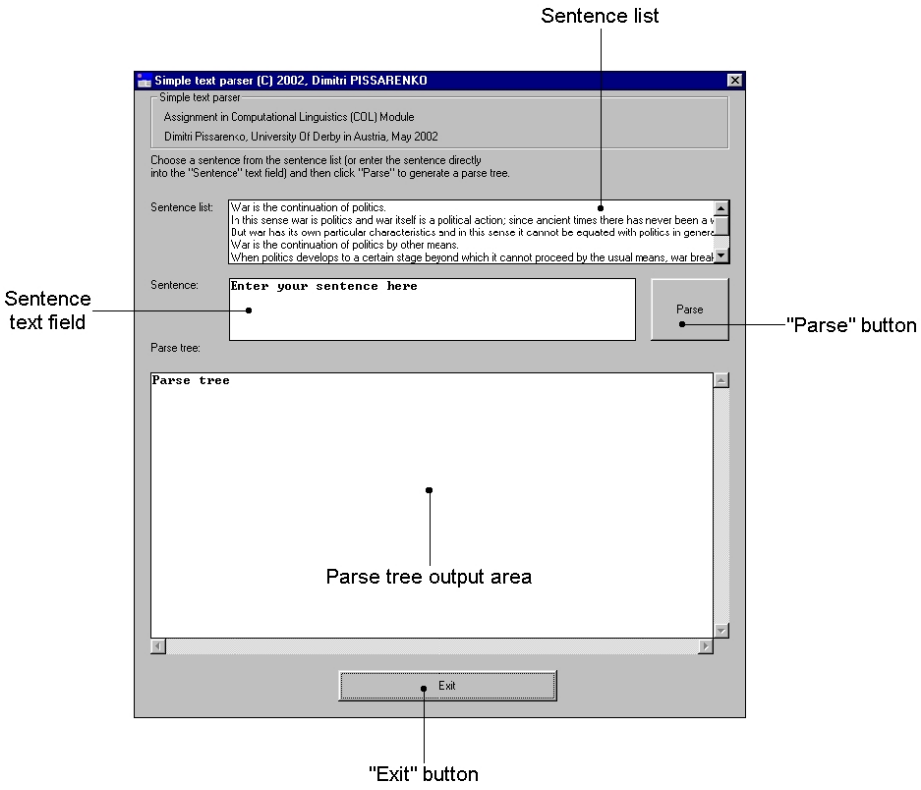

Figure 16: User interface of the parser.

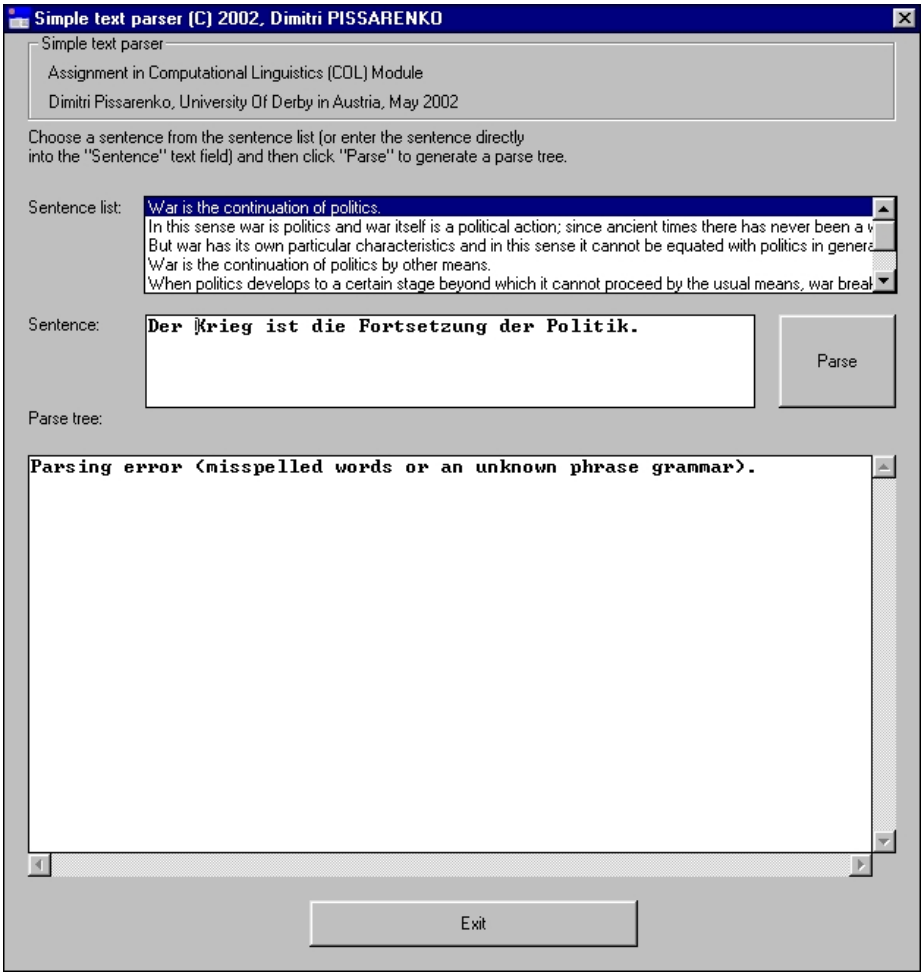

Figure 17: Reaction of the parser to entering a sentence of unknown structure (the entered sentence is German).

choose the sentence he wants to parse. The selected sentence is immediately "transferred" to the *sentence text field*. As in the previous case, the user has to press the *"Parse" button* in order to start parsing.
The program can be terminated by pressing the *"Exit" button* or the *close* menu item from the system menu of the parser.

3.2 Failure scenarios

Two types of error may occur during parsing. Firstly, the sentence entered by the user may be of unknown grammar and secondly, the user might forget to place a period (.) after the last word of the sentence.

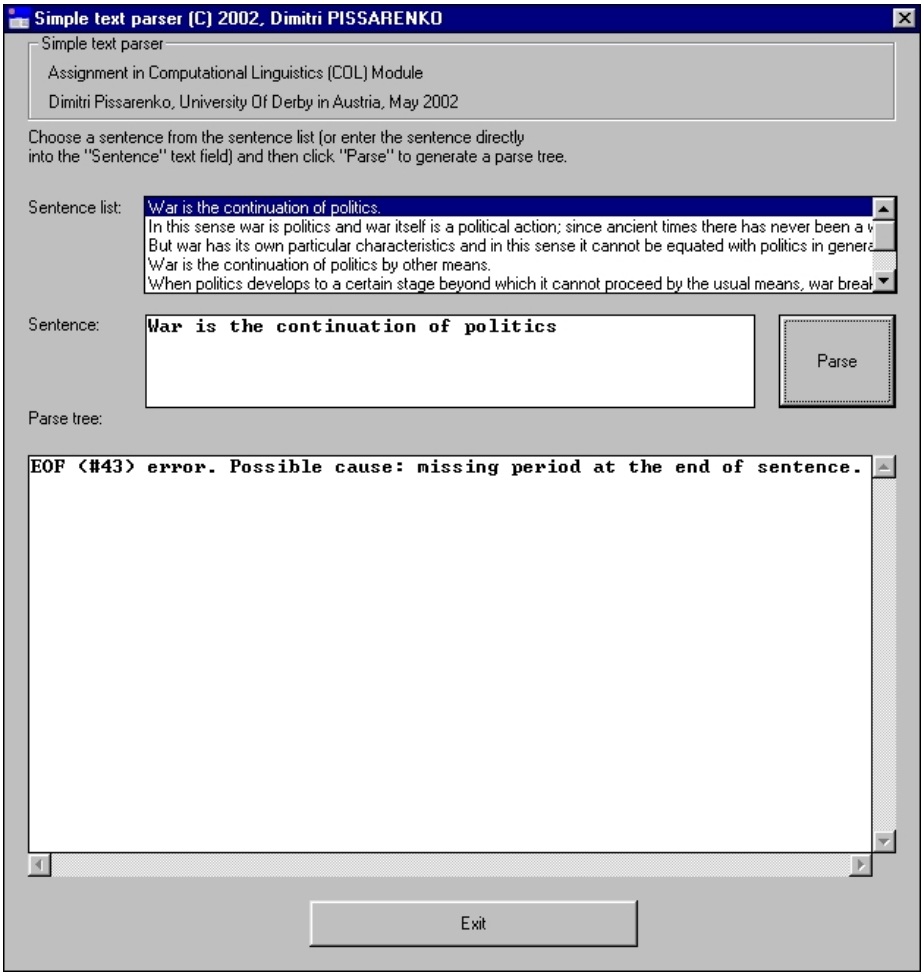

Figure 18: Reaction of the parser to entering a sentence without a concluding period.

3.2.1 Unknown grammar

If the user tries to parse a sentence, whose structure is not defined in the grammar file of the parser, or misspells a certain word, the parse tree cannot be generated.
In this case, an error message is put out in the parse tree output area (see figure 17).

3.2.2 Missing period

According to the grammar used in this work, each sentence has to end with a period. Otherwise, the parser can not parse the text due to technical specialities of Prolog.
A missing period in the text to be parsed causes an error which is "caught" by

the parser and shown in the parse tree output area (figure 18).

3.3 Technical structure of the parser

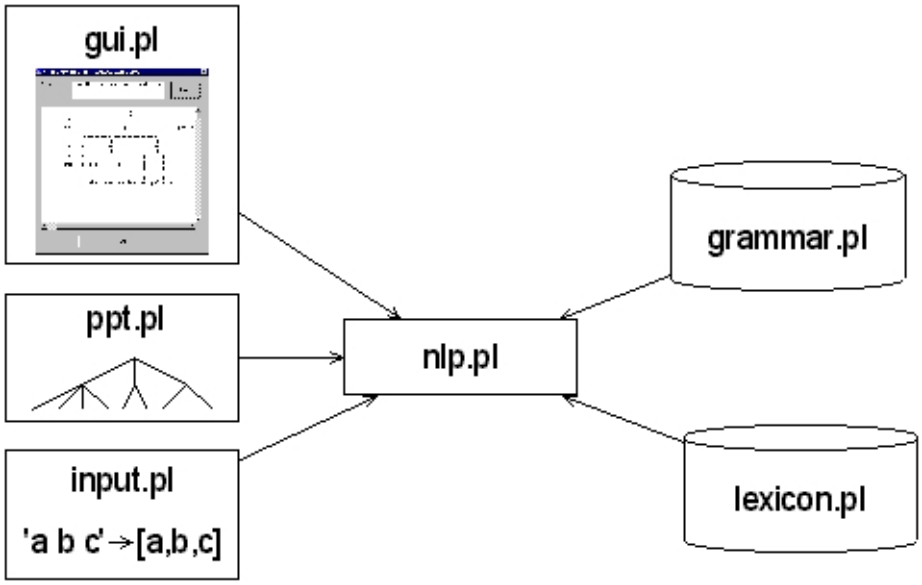

Figure 19: Structure of the program

The program consists of several files (see figure 19):

- *nlp.pl* is the "central" file, that loads (consults) all other parts and starts the application.
- *gui.pl* contains predicates for handling the graphical user interface.
- *ppt.pl* is the tree printing predicate by Gustaf Neumann.
- *input.pl* contains the predicates for transforming strings to Prolog lists that can be processed by means of LPA Prolog natural language processing support.
- *grammar.pl* is the grammar file (syntax of all phrases of the text).
- *lexicon.pl* is the lexicon file.

3.4 Foreign source code

Part of the source code of the parser has been taken from other sources, and not created by the author of this work.
This applies to the following parts:

- The *ppt/1* predicate (file *ppt.pl*) is the product of Gustaf Neumann's mind.

- The *read_in/1* predicate (file *input.pl*) is David Mitchell's intellectual property and was downloaded from `http://www.cs.sfu.ca/~mitchell/teaching/384/a6files/microml`.
- The predicates *get_lbx_selection/2*, *get_lbx_selection/5*, *get_selected/5* (file *gui.pl*) were all taken from the example file *eg_lbx.pl* which is bundled with LPA Prolog 3.3.

3.5 Configuration information

The parser presented in scope of this work was developed and tested using LPA Win-Prolog 3.3. It can *not* be guaranteed that the parser will work with other Prolog dialects or with LPA Win-Prolog of other versions.

References

Det humanistiske fakultet, Universitetet i Tromso, 9037 Tromso. Intercomprehension in germanic languages online, 2002. URL `http://www.hum.uit.no/a/svenonius/lingua/flow/co/gram/rfgrsv/svrtoc.html`. (URL accessed on February 20, 2002).

T. Mao. *On Protracted War, Selected Works*, volume II. May 1938.

C. Matthews. *An Introduction to Natural Language Processing Through Prolog.* Addison-Wesley Longman, 1998. ISBN 0-582-066220.

R. Shalfield. *WIN-PROLOG Programming Guide.* Logic Programming Associates Ltd, 2001.

A Source code

A.1 nlp.pl

```
1  /**
2   * Simple PROLOG text parser
3   *
4   * Dimitri PISSARENKO, University of Derby in Austria
5   * May 15, 2002
6   *
7   * Main file. Loads all necessary predicates and opens the window
8   * of the application
9   **/
10
11 ?- abolish_files(['ppt.pl', 'grammar.pl', 'lexicon.pl', 'gui.pl', 'input.pl']).
12 ?- consult('gui.pl').
13 ?- consult('ppt.pl').
14 ?- consult('grammar.pl').
15 ?- consult('lexicon.pl').
16 ?- consult('input.pl').
```

```

?- start.
```

A.2 gui.pl

```
/**
 * Simple PROLOG text parser
 *
 * Dimitri PISSARENKO, University of Derby in Austria
 * May 15, 2002
 *
 * In this file, the graphical user interface (GUI) of the parser is defined.
 **/
/* The predicate start creates and opens the window of the application */
start :-
  /* Create a dialog box */
  wdcreate(user_dialog,‘Simple text parser (C) 2002, Dimitri PISSARENKO‘,
  235,32,626,661,[ws_sysmenu,ws_caption,dlg_ownedbyprolog]),
  /* Create the "Exit" button */
  wccreate((user_dialog,102),button,‘Exit‘,208,592,224,32,
   [ws_child,ws_visible,ws_tabstop,bs_pushbutton]),
  wccreate((user_dialog,1005),static,
   ‘Dimitri Pissarenko, University Of Derby in Austria, May 2002‘,
   30,40,560,16,[ws_child,ws_visible,ss_left]),
  wccreate((user_dialog,1004),static,
   ‘Assignment in Computational Linguistics (COL) Module‘,
   30,20,560,16,[ws_child,ws_visible,ss_left]),
  wccreate((user_dialog,1002),static,‘Sentence:‘,16,192,64,16,
   [ws_child,ws_visible,ss_left]),
  /* Create a text field for entering the sentence directly */
  wccreate((user_dialog,800),editor,‘Enter your sentence here‘,
   96,192,416,64,
   [ws_child,ws_visible,ws_tabstop,ws_border,es_left,es_multiline]),
  /* Create the "Parse" button */
  wccreate((user_dialog,100),button,‘Parse‘,528,192,80,64,
   [ws_child,ws_visible,ws_tabstop,bs_pushbutton]),
  /* Create a list box with all the sentences */
  wccreate((user_dialog,400),listbox,‘sentenceList‘,96,112,512,64,
   [ws_child,ws_visible,ws_tabstop,ws_border,
   ws_vscroll,lbs_notify,lbs_hasstrings]),
  wccreate((user_dialog,1000),static,‘Sentence list:‘,
   16,112,64,16,[ws_child,ws_visible,ss_left]),
  wccreate((user_dialog,1001),static,
   ‘Choose a sentence from the sentence list (or enter the sentence directly
into the "Sentence" text field) and then click "Parse" to generate a parse tree.‘,
   16,64,592,32,[ws_child,ws_visible,ss_left]),
  wccreate((user_dialog,1100),button,‘Simple text parser‘,
   16,0,592,60,[ws_child,bs_groupbox,ws_visible]),
  /* Create the parse tree text field */
  wccreate((user_dialog,801),editor,
```

```
46    'Parse tree',16,288,592,288,[ws_child,ws_visible,ws_tabstop,ws_border,
47    es_left,es_multiline,ws_hscroll,ws_vscroll,es_autohscroll,
48    es_autovscroll,es_readonly]),
49   wccreate((user_dialog,1003),static,'Parse tree:',16,256,80,16,
50    [ws_child,ws_visible,ss_left]),
51   /* The following commands (wlbxadd) fill the sentence list with entries, */
52   /* ie sentences of the story. */
53   wlbxadd((user_dialog,400), 0, 'War is the continuation of politics.'),
54   wlbxadd((user_dialog,400), 1,
55   'In this sense war is politics and war itself is a political action; since
56   ancient times there has never been a war that did not have
57   a political character.'),
58   wlbxadd((user_dialog,400), 2,
59   'But war has its own particular characteristics and in this sense it
60   cannot be equated with politics in general.'),
61   wlbxadd((user_dialog,400), 3,
62   'War is the continuation of politics by other means.'),
63   wlbxadd((user_dialog,400), 4,
64   'When politics develops to a certain stage beyond which it cannot proceed
65   by the usual means, war breaks out to sweep the obstacles from the way.'),
66   wlbxadd((user_dialog,400), 5,
67   'When the obstacle is removed and our political aim attained, the war will stop.'),
68   wlbxadd((user_dialog,400), 6,
69   'But if the obstacle is not completely swept away, the war will have to
70   continue till the aim is fully accomplished.'),
71   wlbxadd((user_dialog,400), 7,
72   'It can therefore be said that politics is war without
73   bloodshed while war is politics with bloodshed.'),
74   /* Attach an event handler to the dialog box */
75   window_handler('user_dialog', windowHandler),
76   /* Open the dialog box */
77   wshow('user_dialog',1) .
78
79 /* The following predicate is called when the user presses the "Parse" */
80 /* button and the entered sentence is parsed properly (no errors occur). */
81 windowHandler((user_dialog,100), msg_button, _, _) :-
82    /* Save the phrase entered by the user in variable EnteredString */
83    wtext((user_dialog,800), EnteredString),
84    /* Convert string EnteredString into a PROLOG list Sentence */
85    r(Sentence) <~ EnteredString,
86    /* Parse Sentence and save the results in ParseResult  */
87    phrase(s(ParseResult),Sentence),
88    /* Generate a "graphical" tree from ParseResult and save it */
89    /* in ParseTree                                        */
90    ppt(ParseResult) ~> ParseTree,
91    /* Display ParseTree in the parse tree text field      */
92    wtext((user_dialog,801), ParseTree).
93
94 /* The following predicate is called when the user presses the "Parse" */
95 /* button and the entered sentence can not be parsed properly because the  */
```

```
96 /* either some word is misspelled or the entered sentence has an unknown */
97 /* grammar.                                                               */
98 windowHandler((user_dialog,100), msg_button, _, _) :-
99     /* Print an error message in the parse tree text field. */
100    wtext((user_dialog,801),
101    `Parsing error (misspelled words or an unknown phrase grammar).`).
102
103 /* The following predicate is called when the user presses the "Parse" */
104 /* button and the entered sentence cannot be parsed because a period (.) at */
105 /* the end of the sentence is missing.                                    */
106 '?ERROR?'( 43, Goal ) :-
107     /* Print an error message in the parse tree text field. */
108    wtext((user_dialog,801),
109    `EOF (#43) error. Possible cause: missing period at the end of sentence.`),
110    abort.
111
112 /* The following predicate is called when the user presses the "Exit" */
113 /* button.                                                            */
114 windowHandler((user_dialog,102), msg_button, _, _) :-
115    /* Close the window */
116    wclose(user_dialog).
117
118 /* The following predicate is called when the user calls the "Close" */
119 /* menu item in the system menu or the cross (X) in the title bar.  */
120 windowHandler(user_dialog, msg_close, _, _) :-
121    /* Close the window */
122    wclose(user_dialog).
123
124 /* The following predicate is called when the user selects a sentence from */
125 /* the sentence list by clicking on the desired item                */
126 windowHandler((user_dialog,400), msg_select, _, _) :-
127   /* Get the list of all the selected items and store this list in the */
128   /* variable SelectedItems.                                          */
129   get_lbx_selection((user_dialog,400), SelectedItems),
130   /* Take the first item from the list of selected items and store it in */
131   /* the variable FirstSelectedItem.                                  */
132   head(SelectedItems, FirstSelectedItem),
133   /* Print the selected item (sentence) in the text field for entering the */
134   /* sentence to parse.                                               */
135   wtext((user_dialog,800), FirstSelectedItem).
136
137 /* This predicate determines the first element of the list (first argument)*/
138 /* and stores this first element in the second argument             */
139 head([H|T], H).
140
141 /* The predicates get_lbx_selection/2, get_lbx_selection/5, get_selected/5 */
142 /* were all taken from the example files bundled with LPA WIN-PROLOG 3.300 */
143 /* These predicates and the documentation can be found in the file  */
144 /* eg_lbx.pl in the directory examples                              */
145
```

```
146 /************************************************************************
147 ** given a listbox return a list of its currently selected items
148 ************************************************************************/
149 % get the next item after 0 (the first item) in the given listbox,
150 % start checking the items in the listbox starting at the first item,
151 % giving the next item and the empty list as the list of selections so
152 % far and return the result
153
154 get_lbx_selection( Lbx, Selections ) :-
155   wlbxfnd( Lbx, 0, ``, NextItem ),
156   get_lbx_selection( Lbx, 0, NextItem, [], Selections ).
157
158 /************************************************************************
159 ** build a list of the current selections in the listbox
160 ************************************************************************/
161
162 % check the selection state of the given item in the given listbox
163 % if it is selected add it to the list of selections found so far.
164 % depending on the value of the next item either finish or find the item
165 % following and continue checking from the next item.
166
167 % if the next item is 0, we are back at the start of the listbox
168 % so check the selection of the current item and finish
169
170 get_lbx_selection( Lbx, Item, 0, SoFar, Sels ) :-
171   wlbxsel( Lbx, Item, Sel ),
172   get_selected( Sel, Lbx, Item, SoFar, Sels ).
173
174 % if the next item is not 0 check the selection of the current item
175 % find the item after next and continue checking from the next item
176
177 get_lbx_selection( Lbx, Item, NextItem, SoFar, Sels ) :-
178   wlbxsel( Lbx, Item, Sel ),
179   get_selected( Sel, Lbx, Item, SoFar, NS ),
180   wlbxfnd( Lbx, NextItem, ``, AfterNextItem ),
181   get_lbx_selection( Lbx, NextItem, AfterNextItem, NS, Sels ).
182
183 /************************************************************************
184 ** add a selected menu item to a list
185 ************************************************************************/
186
187 % if the selection state is 0, the item is not selected
188 % so the list of selected items remains unchanged.
189
190 get_selected( 0, _,    _,   SoFar, SoFar ).
191
192 % if the selection state is 1, the item is selected
193 % so add the item's string to the list of selected items
194
195 get_selected( 1, Lbx, Item, SoFar, [ItemStr|SoFar] ) :-
```

```
196  wlbxget( Lbx, Item, ItemStr ).
```

A.3 ppt.pl

```
1  /**
2   * Simple PROLOG text parser
3   *
4   * Dimitri PISSARENKO, University of Derby in Austria
5   * May 15, 2002
6   *
7   * Definition of the ppt predicate for displaying tree structures.
8   * This predicate was developed by Gustaf Neumann (see below).
9   **/
10 /*
11   ppt/1,
12   printing Prolog terms in a tree layout for typewriter output.
13
14   Written in Spring 1985 -- Gustaf Neumann
15
16   (c)1985 Gustaf Neumann, Wirtschaftsuniversitaet Wien,
17      Augasse 2-6, 1090 Vienna, Austria *222/31 336-4533.
18      email: neumann@wu-wien.ac.at or neumann@awiwuw11.bitnet
19
20   Permission is granted to use, copy and distribute this program as long
21   (1) this note remains intact,
22   (2) no fees are charged and
23   (3) no further restrictions are imposed.
24
25   The following predicates are not defined within this program:
26   -   length(List,Length),
27   -   tab(Exp),
28   -   append(A,B,AB).
29   Do not try to print infinite trees :-)
30
31   To show, what this program does, issue the goal: examples.
32 */
33 ?-op(100,xfy,:).
34
35 examples:- example(X), ppt(X), nl, nl, write(X), nl,
36   wait_for_input, fail.
37 examples.
38
39 example(sin(alpha)/cos(beta-phi)+cos(beta)*tan(360-alpha)).
40 example(buch(titel(wirschaftsinformatik1),autor(hans_robert, hansen))).
41 example((a:-b,c,d)).
42 %example((ppt(X,Y):-Body)):- clause(ppt(X,Y),Body).
43 example(sentence(np(proper(gustaf)),vp(v(likes),np(proper(prolog))))).
44 example(sentence(np(det(that),n(man),rel(that,vp(iv(whistle)))),
45           vp(tv(tunes),np(det(nil),n(pianos),rel(nil))))).
46 example(wirtschaftsinformatik(leitung(hans_robert),
```

```
47          sekretariat(virly,anita),
48          assistenten(lore,rony,goeha,gu,margret,andy,stessi))).
49
50
51 /*************************************************************************
52 *                    top level predicate ppt/1                          *
53 *************************************************************************/
54 ppt(Term):- ppt(Term,arc).
55 ppt(Term,Arc) :-
56      number_vars(Term,0,_),       /* ground all variables in Term  */
57     {pos(Term,Pos,C,0-Right)},    /* compute hor. positions of nodes */
58     {inv([Pos],[]:_,H:T,s)},      /* invert structure for printing */
59      posdiff(-(72-Right)//2,0,Tab), /* compute hor. tab for centering */
60     {print_tree(H:T,[C],Tab,Arc)}.
61                                   /* print tree in line printer mode */
62
63 /*************************************************************************
64 *                    Compute Positions of Nodes                         *
65 *************************************************************************/
66 pos(Head,t(Head,Rel,L,[],0)-[], Nc, N0-Nn):- /* leaf node      */
67     atomic(Head), !,
68     string_length(Head,L), Nn is N0+L,
69     Rel is L//2,                            /* middle of the node */
70     Nc  is (N0+Nn)//2.                      /* center over node */
71 pos(X,t(Head,Rel,L,Centers,Adj)-A, Nc, N0-N2):- /* non-leaf node */
72     X =.. [Head|Args],
73     pos_list(Args,A,Centers,N0-N1),
74     string_length(Head,L), posdiff(N1-N0,L,Error),
75     Adj is (Error+((N1-N0) mod 2))//2,
76     N2 is N1+Error,
77     Rel is L//2,                            /* middle of the node */
78     Nc  is (N0+N2)//2.
79
80 pos_list([], [],     [],        N-N).
81 pos_list([H], [A],    [Center],  N-N1) :- !, pos(H,A,Center,N-N1).
82 pos_list([H|T],[A|Args],[C|Centers],N0-Nn):-
83     pos( H,   A,      C,        N0-N1),
84     N2 is N1+2, pos_list(T,Args,Centers,N2-Nn).
85
86 string_length(X,L):- atomic(X), name(X,S), length(S,L).
87
88 posdiff(Expr,L,Adj):- Adj is L-Expr, Adj > 0, !.
89 posdiff(_,_,0).
90
91 /*************************************************************************
92
93 *                           invert tree                                 *
94 *************************************************************************/
95 inv([Node-Sons|Brothers],List:Deep,[Node|List1]:Deep2,_):-
96     inv(Brothers,List:Deep,List1:Deep1,b),
```

```
97      inv(Sons,Deep1,Deep2,s).
98  inv([],[]:[],[],s).
99  inv([],[]:[],[]:_,b).
100 inv([],E,E,_).
101
102 /*************************************************************************
103 *                            print tree                                 *
104 *************************************************************************/
105 print_tree(Node:Deep, Centers, Tab, Arc) :-
106     tab(Tab), print_list(Node,0,Centers,Cd), nl,
107     {(   Arc  == noarc
108      ;    Deep == []
109      ;    tab(Tab), marks(Centers,Node,0),
110           tab(Tab), horarc(Node,0,_), nl,
111           tab(Tab), marks(Cd,0)
112     )},
113     print_tree(Deep,Cd,Tab,Arc).
114 print_tree([],[],_,_).
115
116 print_list([t(H,Rel,L,Cd,Adj)|R], P0, [C|Centers], Ca) :-
117     P is C-Rel, tab(P-P0), write(H), Pn is P+L,
118     print_list(R,Pn,Centers,Cr),
119     add_to(Cd,Adj,Cda), {append(Cda,Cr,Ca)}.
120 print_list([],_,[],[]).
121
122 /*************************************************************************
123 *                            draw arcs                                  *
124 *************************************************************************/
125 marks([],[],_) :- nl.
126 marks([H|T],[t(_,_,_,[],_)|R],E) :- !, tab(H-E), write(' '),marks(T,R,H+1).
127 marks([H|T],[_|R],         E) :-   tab(H-E), write('|'),marks(T,R,H+1).
128
129 marks([],_) :- nl.
130 marks([H|T],E) :- tab(H-E), write('|'), marks(T,H+1).
131
132 horarc([], A,A).
133 horarc([t([],_,_,_,_ )|R],P,P2) :- !, horarc(R,P,P2).
134 horarc([t(_,_,_,Cd,Adj)|R],P,P2) :- line(Cd,Adj,P,P1), horarc(R,P1,P2).
135
136 line([],  _,E,P) :- P is E.
137 line([H], A,E,P) :- !, tab(H+A-E), write('.'), P is H+A+1.
138 line([H|T],A,E,P) :-  tab(H+A-E), write('.'), line_([H|T],A,H+A+1,P).
139
140 line_([],     _,E,P) :- P is E.
141 line_([H],    A,E,P) :- line_to(H+A-E), P is H+A+1.
142 line_([_,T|Tt],A,E,P) :- line_to(T+A-E), write('.'), line_([T|Tt],A,T+A+1,P).
143
144 line_to(Exp) :- L is Exp, line_to_(L,'-').
145 line_to_(L,_) :- L < 1.
146 line_to_(L,C) :- L >= 1, write(C), L1 is L-1, line_to_(L1,C).
```

```
147
148 add_to([],_,[]).
149 add_to([H|T],A,[Ha|Ta]) :- Ha is H+A, add_to(T,A,Ta).
150
151 /*************************************************************************
152 *                     misc utility predicates                    *
153 *************************************************************************/
154 {G} :- G,!.
155
156 wait_for_input :- get0(_).
157 %wait_for_input :- system([clrscrn,more]).
158
159 number_vars(Term,N0,N1) :-
160    var(Term), !,
161    name(N0,Digits), name('V',[C]),
162    name(Term,[C|Digits]),
163    N1 is N0+1.
164 number_vars(Term,N0,N0) :-
165    atomic(Term), !.
166
167 number_vars(Term,N0,N1) :-
168    Term =.. [_|Args],
169    number_list(Args,N0,N1).
170
171 number_list([],N0,N0).
172 number_list([H|T],N0,N2) :- number_vars(H,N0,N1), number_list(T,N1,N2).
```

A.4 input.pl

```
1  /**
2   * Simple PROLOG text parser
3   *
4   * Dimitri PISSARENKO, University of Derby in Austria
5   * May 15, 2002
6   *
7   * Definition of the read_in predicate for converting strings
8   * to PROLOG lists that can be used in connection with NLP facilities
9   * of LPA PROLOG.
10  *
11  * This predicate was taken from:
12  * David Mitchell, Inital Code for Assignment 6,
13  * http://www.cs.sfu.ca/~mitchell/teaching/384/a6files/microml, March 2001
14  *
15  **/
16
17 /* Read in a Sentence */
18 r(S) :- read_in(S).
19 read_in([W|Ws]):- get0(C), readword(C,W,C1), restsent(W,C1,Ws).
20 /* note that you cannot call this with W,Ws partially instantiated */
21
```

```
/* Given a word and the folling character, read rest of sentence */
restsent(W,_,[]):- lastword(W), !.
restsent(_,C,[W1|Ws]):- readword(C,W1,C1),restsent(W1,C1,Ws).

/* Given the first character, read the rest of a word */
readword(C,W,C1):- single_character(C),!,name(W,[C]),get0(C1).
readword(C,W,C2):-
     in_word(C,NewC), !,
     get0(C1),
     restword(C1,Cs,C2),
     name(W,[NewC|Cs]).
readword(_,W,C2):- get0(C1),readword(C1,W,C2).
restword(C,[NewC|Cs],C2):-
    in_word(C,NewC), !,
    get0(C1),
    restword(C1,Cs,C2).
restword(C,[],C).

/* these characters end sentences */
lastword('.').
lastword('!').
lastword('?').

/* functions of ascii characters                  */
/*  we define each as being:                      */
/*    - a character which may occur in a word.    */
/*    - a single-char word.                       */
/*    - something we ingnore.                     */
/* below 32 we ignore */
%%%%in_word(39,39).     /* ' */
/* 48-57 are numerals */
in_word(C,C):- C>47, C<58. /* 0..9 */
%%%%in_word(34,34).     /* " */
/* 97-122 are lower case characters */
in_word(C,C):- C>96, C<123. /* a..z */
/* 65-90 are upper case characters */
in_word(C,L):- C>64, C<91, L is C+32.  /* A..Z */
%% single_character(32). /* */
single_character(33). /* ! */
single_character(34). /* " */
single_character(35). /* # */
single_character(36). /* $ */
single_character(37). /* % */
single_character(38). /* & */
single_character(39). /* ' */
single_character(40). /* ( */
single_character(41). /* ) */
single_character(42). /* * */
single_character(43). /* + */
single_character(44). /* , */
```

```
72 single_character(45). /* - */ %%in_word(45,45).     /* - */
73 single_character(46). /* . */
74 single_character(47). /* / */
75 single_character(58). /* : */
76 single_character(59). /* ; */
77 single_character(60). /* < */
78 single_character(61). /* = */
79 single_character(62). /* > */
80 single_character(63). /* ? */
81 single_character(64). /* @ */
82 single_character(91). /* [ */
83 single_character(92). /* \ */
84 single_character(93). /* ] */
85 single_character(94). /* ^ */
86 single_character(95). /* _ */
87 single_character(96). /* ` */
88 single_character(123). /* { */
89 single_character(124). /* | */
90 single_character(125). /* } */
91 single_character(126). /* ~ */
```

A.5 grammar.pl

```
1  /**
2   * Simple PROLOG text parser
3   *
4   * Dimitri PISSARENKO, University of Derby in Austria
5   * May 15, 2002
6   *
7   * In this file, the grammar supported by the parser is defined.
8   **/
9
10 s(s(Np,Vp,Period))-->np(Np),vp(Vp),period(Period).
11 np(np(N)) --> n(N).
12 vp(vp(V,Np,Pp)) --> v(V),np(Np),pp(Pp).
13 np(np(Det,N)) --> det(Det),n(N).
14 pp(pp(P,N)) --> p(P),n(N).
15 s(s(Sa,N,Vp0,C,Np0,Vp1,Semicolon,Pp,Ap,Subj,Vp2,Np1,Period)) -->
16    sa(Sa),n(N),vp(Vp0),c(C),np(Np0),vp(Vp1),semicolon(Semicolon),
17    pp(Pp),ap(Ap),subj(Subj),vp(Vp2),np(Np1),period(Period).
18 sa(sa(P,Pn,N)) --> p(P),pn(Pn),n(N).
19 vp(vp(V,N)) --> v(V),n(N).
20 np(np(N,Pn)) --> n(N),pn(Pn).
21 vp(vp(V,Np)) --> v(V),np(Np).
22 np(np(Det,Adj,N)) --> det(Det),adj(Adj),n(N).
23 pp(pp(P,Adj,N)) --> p(P),adj(Adj),n(N).
24 ap(ap(Adv0,V0,Adv1,V1,Np)) --> adv(Adv0),v(V0),adv(Adv1),v(V1),np(Np).
25 vp(vp(V0,Adv,V1)) --> v(V0),adv(Adv),v(V1).
26 s(s(C0,N,Vp0,C1,Ap0,Pn,Vp1,Ap1,Period)) --> c(C0),n(N),vp(Vp0),c(C1),
27    ap(Ap0),pn(Pn),vp(Vp1),ap(Ap1),period(Period).
```

```
vp(vp(V,Pnp,Np)) --> v(V),pnp(Pnp),np(Np).
pnp(pnp(Pn0,Pn1)) --> pn(Pn0),pn(Pn1).
np(np(Adj,N)) --> adj(Adj),n(N).
ap(ap(P,Pn,N)) --> p(P),pn(Pn),n(N).
vp(vp(V0,V1,V2)) --> v(V0),v(V1),v(V2).
ap(ap(P,Np)) --> p(P),np(Np).
np(np(N,Ap)) --> n(N),ap(Ap).
ap(ap(P,Adj)) --> p(P),adj(Adj).
s(s(N,Vp,Ap,Period)) --> n(N),vp(Vp),ap(Ap),period(Period).
np(np(N,Pp)) --> n(N),pp(Pp).
ap(ap(P,Adj,N)) --> p(P),adj(Adj),n(N).
s(s(P0,N0,Vp0,P1,Pn0,Pn1,Vp1,Ap0,Comma,N1,Vp2,Ap1,Ap2,Period)) -->
   p(P0),n(N0),vp(Vp0),p(P1),pn(Pn0),pn(Pn1),vp(Vp1),ap(Ap0),
   comma(Comma),n(N1),vp(Vp2),ap(Ap1),ap(Ap2),period(Period).
vp(vp(V,Ap)) --> v(V),ap(Ap).
ap(ap(Part,Np)) --> part(Part),np(Np).
vp(vp(V0,V1)) --> v(V0),v(V1).
vp(vp(V,Part)) --> v(V),part(Part).
ap(ap(Vp,Np)) --> vp(Vp),np(Np).
vp(vp(Part,V)) --> part(Part),v(V).
s(s(Ap,Comma,Np,Vp,Period)) --> ap(Ap),comma(Comma),np(Np),vp(Vp),
   period(Period).
ap(ap(Adv,Np,Vp0,C,Vp1)) --> adv(Adv),np(Np),vp(Vp0),c(C),vp(Vp1).
vp(vp(Np,V)) --> np(Np),v(V).
np(np(Pn,Adj,N)) --> pn(Pn),adj(Adj),n(N).
s(s(C,Subj,Ap0,Comma,Np,Vp,Ap1,Period)) --> c(C),subj(Subj),ap(Ap0),
   comma(Comma),np(Np),vp(Vp),ap(Ap1),period(Period).
ap(ap(Np,Vp)) --> np(Np),vp(Vp).
vp(vp(V0,Adv0,Adv1,V1,Adv2)) --> v(V0),adv(Adv0),adv(Adv1),
   v(V1),adv(Adv2).
vp(vp(V0,V1,Part,V2)) --> v(V0),v(V1),part(Part),v(V2).
ap(ap(P,Np,V,Adv,Adj)) --> p(P),np(Np),v(V),adv(Adv),adj(Adj).
s(s(Pn,V,Subj0,Vp0,Subj1,N0,Vp1,C,N1,Vp2,Period)) --> pn(Pn),v(V),
   subj(Subj0),vp(Vp0),subj(Subj1),n(N0),vp(Vp1),c(C),n(N1),vp(Vp2),
   period(Period).
vp(vp(V,N,Pp)) --> v(V),n(N),pp(Pp).
```

A.6 lexicon.pl

```
/**
 * Simple PROLOG text parser
 *
 * Dimitri PISSARENKO, University of Derby in Austria
 * May 15, 2002
 *
 * In this file, the lexicon of the parser is defined.
 **/

period(period('.')) --> ['.'].
semicolon(semicolon(';')) --> [';'].
```

```
12 comma(comma(',')) --> [','].
13
14 n(n(war)) --> [war].
15 n(n(continuation)) --> [continuation].
16 n(n(politics)) --> [politics].
17 v(v(is)) --> [is].
18 p(p(of)) --> [of].
19 det(det(the)) --> [the].
20 p(p(in)) --> [in].
21 pn(pn(this)) --> [this].
22 n(n(sense)) --> [sense].
23 c(c(and)) --> [and].
24 pn(pn(itself)) --> [itself].
25 det(det(a)) --> [a].
26 adj(adj(political)) --> [political].
27 n(n(action)) --> [action].
28 p(p(since)) --> [since].
29 adj(adj(ancient)) --> [ancient].
30 n(n(times)) --> [times].
31 adv(adv(there)) --> [there].
32 v(v(has)) --> [has].
33 adv(adv(never)) --> [never].
34 v(v(been)) --> [been].
35 subj(subj(that)) --> [that].
36 v(v(did)) --> [did].
37 adv(adv(not)) --> [not].
38 v(v(have)) --> [have].
39 n(n(character)) --> [character].
40 c(c(but)) --> [but].
41 pn(pn(its)) --> [its].
42 pn(pn(own)) --> [own].
43 adj(adj(particular)) --> [particular].
44 n(n(characteristics)) --> [characteristics].
45 pn(pn(it)) --> [it].
46 v(v(cannot)) --> [cannot].
47 v(v(be)) --> [be].
48 v(v(equated)) --> [equated].
49 p(p(with)) --> [with].
50 adj(adj(general)) --> [general].
51 p(p(by)) --> [by].
52 adj(adj(other)) --> [other].
53 n(n(means)) --> [means].
54 p(p(when)) --> [when].
55 v(v(develops)) --> [develops].
56 p(p(to)) --> [to].
57 adj(adj(certain)) --> [certain].
58 n(n(stage)) --> [stage].
59 p(p(beyond)) --> [beyond].
60 pn(pn(which)) --> [which].
61 v(v(proceed)) --> [proceed].
```

```
62 adj(adj(usual)) --> [usual].
63 v(v(breaks)) --> [breaks].
64 part(part(out)) --> [out].
65 part(part(to)) --> [to].
66 v(v(sweep)) --> [sweep].
67 n(n(obstacles)) --> [obstacles].
68 p(p(from)) --> [from].
69 n(n(way)) --> [way].
70 adv(adv(when)) --> [when].
71 n(n(obstacle)) --> [obstacle].
72 v(v(removed)) --> [removed].
73 pn(pn(our)) --> [our].
74 n(n(aim)) --> [aim].
75 v(v(attained)) --> [attained].
76 v(v(will)) --> [will].
77 v(v(stop)) --> [stop].
78 subj(subj(if)) --> [if].
79 adv(adv(completely)) --> [completely].
80 v(v(swept)) --> [swept].
81 adv(adv(away)) --> [away].
82 v(v(continue)) --> [continue].
83 p(p(till)) --> [till].
84 adv(adv(fully)) --> [fully].
85 adj(adj(accomplished)) --> [accomplished].
86 v(v(can)) --> [can].
87 subj(subj(therefore)) --> [therefore].
88 v(v(said)) --> [said].
89 p(p(without)) --> [without].
90 n(n(bloodshed)) --> [bloodshed].
91 c(c(while)) --> [while].
```

B Parse trees

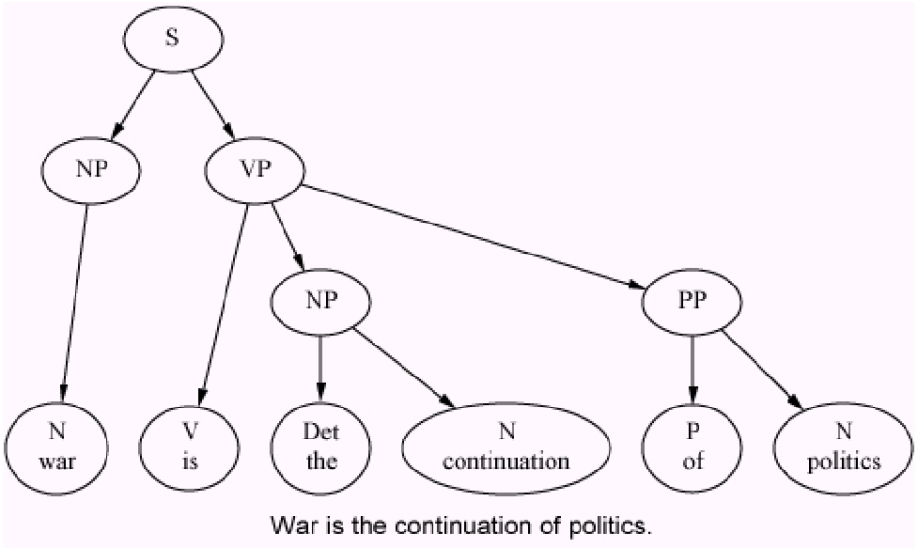

Figure 20: Parse tree of the first phrase

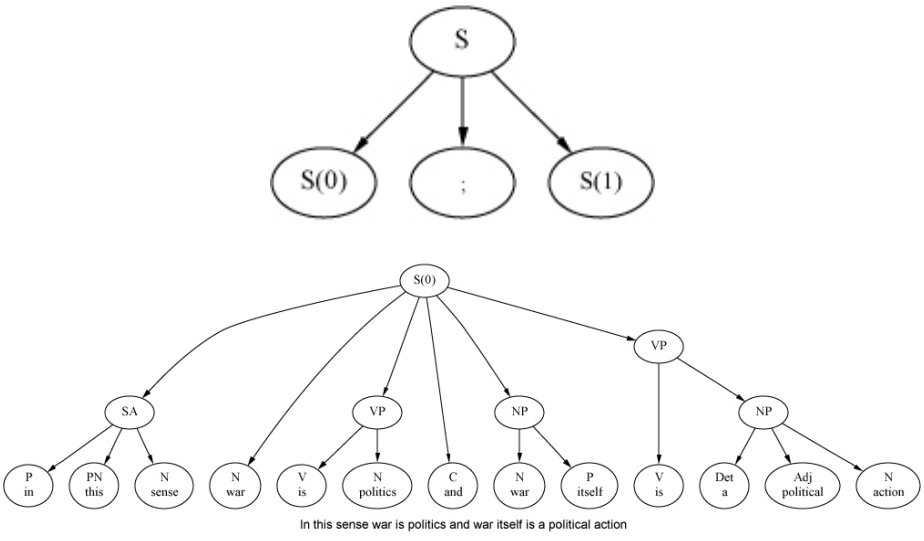

Figure 21: First part of the parse tree of the second phrase ("In this sense war is politics and war itself is a political action...")

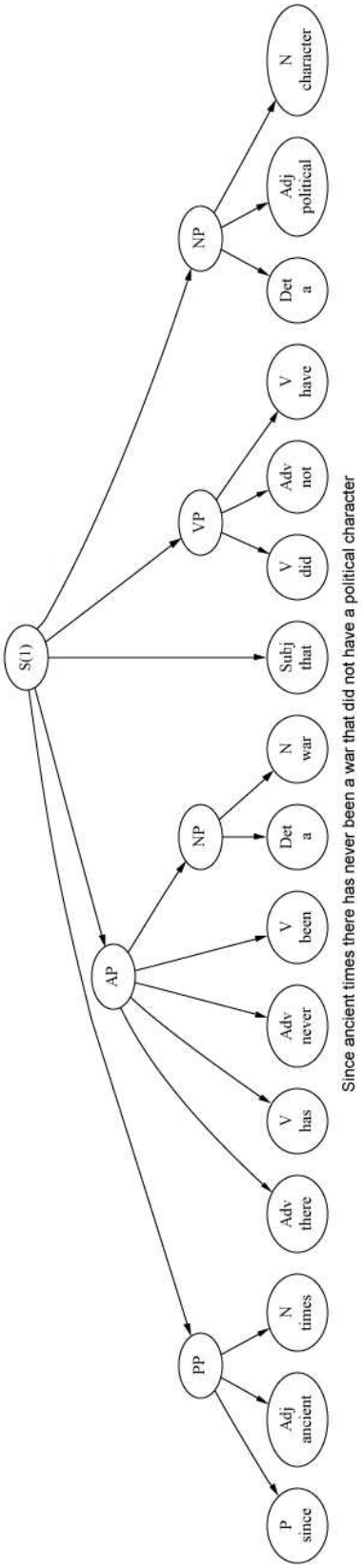

Figure 22: Second part of the parse tree of the second phrase ("...since ancient times there has never been a war that did not have a political character.")

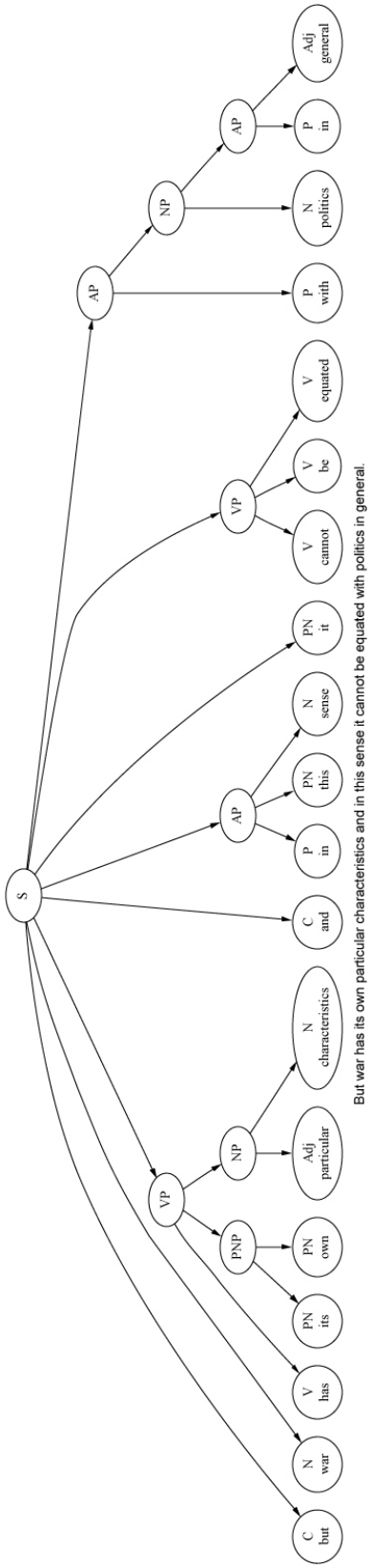

Figure 23: Parse tree of the third phrase

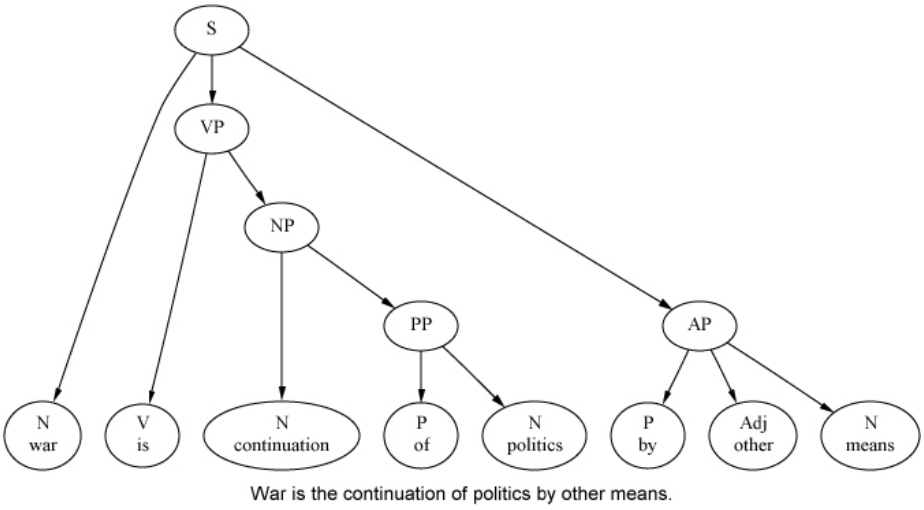

Figure 24: Parse tree of the fourth phrase

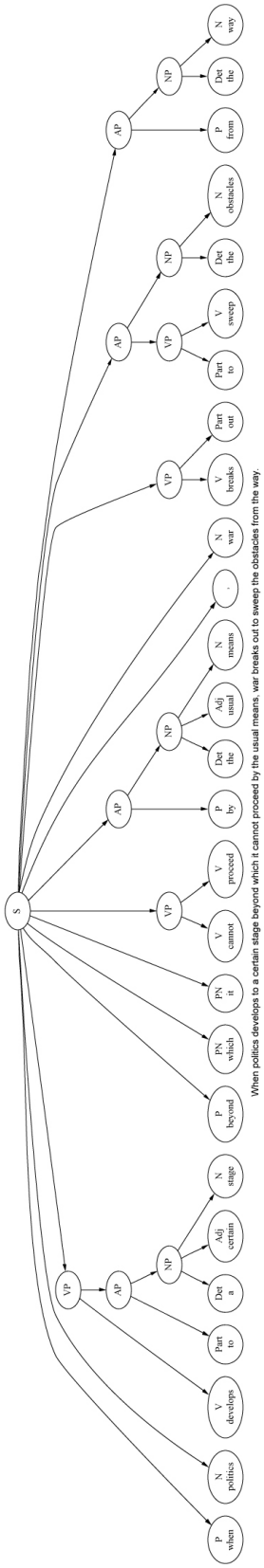

Figure 25: Parse tree of the fifth phrase

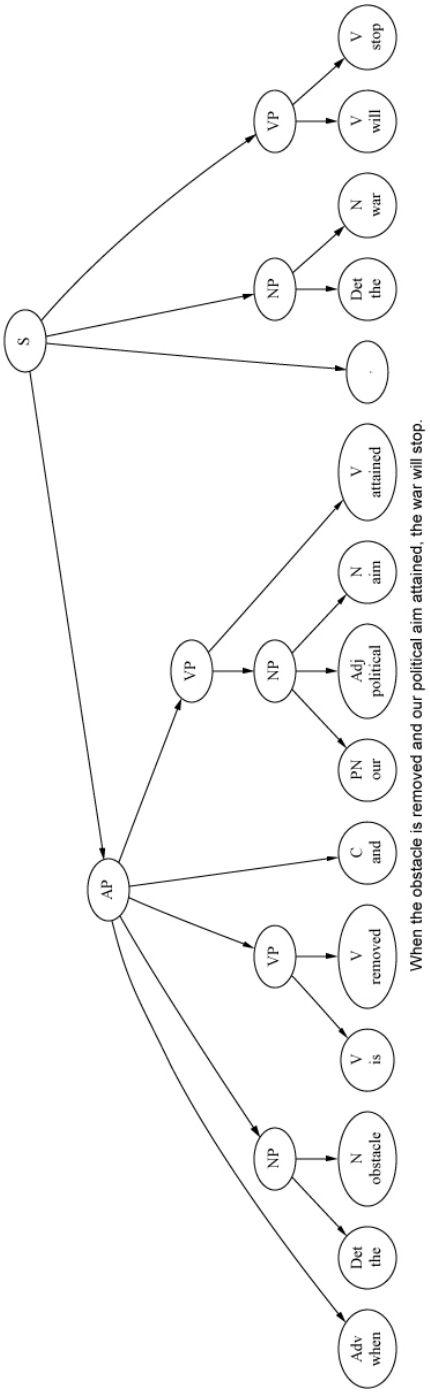

Figure 26: Parse tree of the sixth phrase

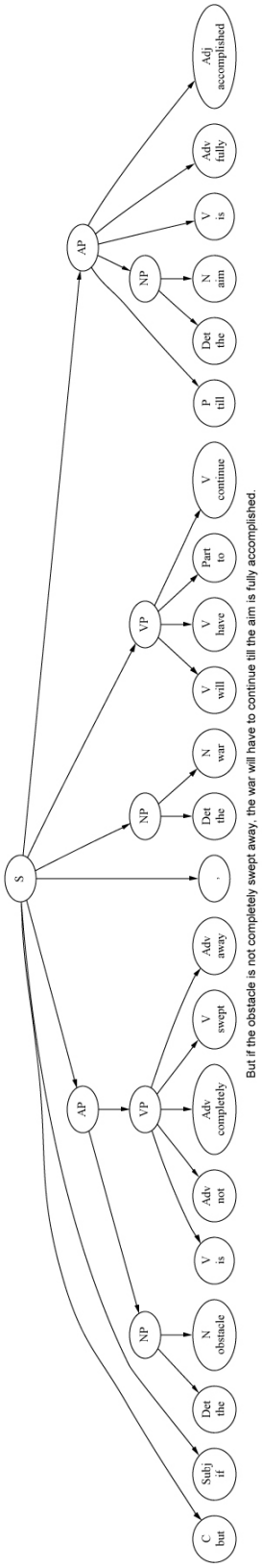

Figure 27: Parse tree of the seventh phrase

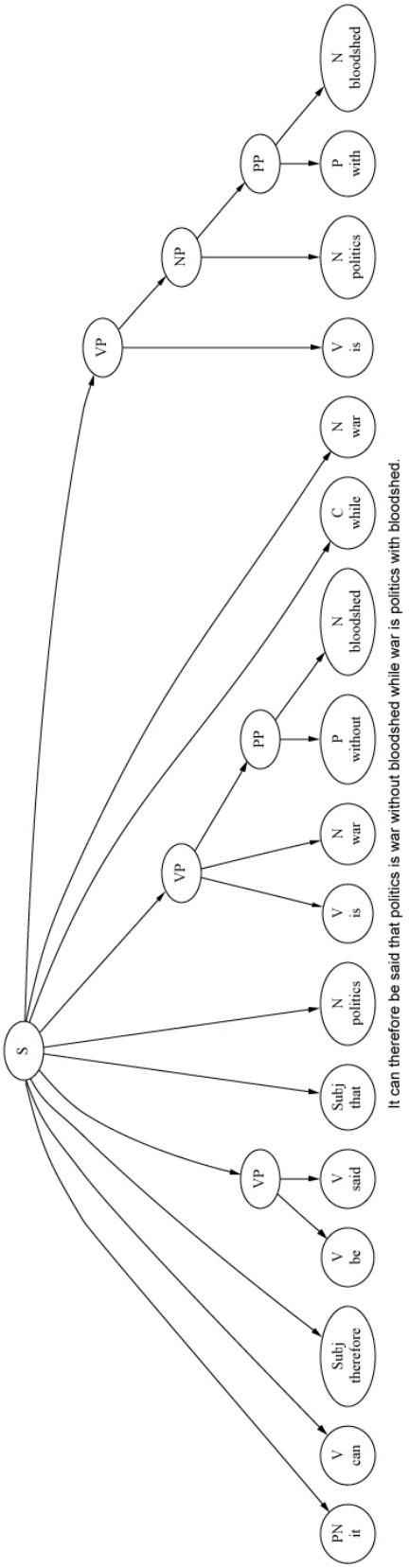

Figure 28: Parse tree of the eighth phrase