15.0 Role of Knowledge in Language Understanding
15.1 Deconstructing Language: A Symbolic Analysis
15.2 Syntax
15.3 Syntax and Knowledge with ATN

15.4 Stochastic Tools for Language Analysis
15.5 Natural Language Applications
15.6 Epilogue and References
15.7 Exercises

Parsers
Fig 15.1 A blocks world, adapted from Winograd (1972).
To manage this complexity, linguists have defined different levels of analysis for natural language:

1. **Prosody** deals with the rhythm and intonation of language. This level of analysis is difficult to formalize and often neglected; however, its importance is evident in the powerful effect of poetry or religious chants, as well as the role played by rhythm in children’s wordplay and the babbling of infants.

2. **Phonology** examines the sounds that are combined to form language. This branch of linguistics is important for computerized speech recognition and generation.

3. **Morphology** is concerned with the components (morphemes) that make up words. These include the rules governing the formation of words, such as the effect of prefixes (un-, non-, anti-, etc.) and suffixes (-ing, -ly, etc.) that modify the meaning of root words. Morphological analysis is important in determining the role of a word in a sentence, including its tense, number, and part of speech.

4. **Syntax** studies the rules for combining words into legal phrases and sentences, and the use of those rules to parse and generate sentences. This is the best formalized and thus the most successfully automated component of linguistic analysis.

5. **Semantics** considers the meaning of words, phrases, and sentences and the ways in which meaning is conveyed in natural language expressions.

6. **Pragmatics** is the study of the ways in which language is used and its effects on the listener. For example, pragmatics would address the reason why “Yes” is usually an inappropriate answer to the question “Do you know what time it is?”

7. **World knowledge** includes knowledge of the physical world, the world of human social interaction, and the role of goals and intentions in communication. This general background knowledge is essential to understand the full meaning of a text or conversation.
Fig 15.2 Stages in producing an internal representation of a sentence.
Fig 15.3 Parse tree for the sentence “The man bites the dog.”
Sentence →  . Noun Verb
Noun →  . mary
Noun --> mary .
Sentence --> Noun . Verb

Verb -->  . runs
Verb --> runs .
Sentence --> Noun Verb .

predict: Noun followed by a Verb
Predict: mary
Scanned: mary
Completed: Noun; predict: Verb

Predict: runs
Scanned: runs
Completed: Verb,
Completed: sentence
Figure 15.4 The relationship of dotted rules to the generation of a parse tree.
The chart for mary runs, with three state lists, is:

S0:  
\[
[(\$ \rightarrow . \text{S}), \text{start}]
\]
\[
(S \rightarrow . \text{Noun Verb}), \text{predictor}
\]

S1:  
\[
[(\text{Noun} \rightarrow \text{mary} .), \text{scanner}]
\]
\[
(S \rightarrow \text{Noun} . \text{Verb}), \text{completer}
\]

S2:  
\[
[(\text{Verb} \rightarrow \text{runs} .)] \text{scanner}
\]
\[
(S \rightarrow \text{Noun Verb} .), \text{completer}
\]
\[
(\$ \rightarrow \text{S} .), \text{completer}
\]

function EARLEY-PARSE(words, grammar) returns chart
begin
    chart := empty
    ADDTOCHART(($Æ . S, [0, 0]), chart[0]) % dummy start state
    for i from 0 to LENGTH(words) do
        for each state in chart[i] do
            if rule_rhs(state) = … . A … and A is not a part of speech
                then PREDICTOR(state)
            else if rule_rhs(state) = … . L … % L is part of speech
                then SCANNER(state)
            else COMPLETER(state) % rule_rhs = RHS
        end
    end
procedure PREDICTOR((A Æ … . B ..., [i, j]))
begin
    for each (B Æ RHS) in grammar do
        ADDTOCHART((B Æ . RHS, [j, j]), chart[j])
end

procedure SCANNER((A Æ … . L ..., [i, j]))
begin
    if (L Æ word[j]) is_in grammar
        then ADDTOCHART((L Æ word[j] ., [j, j + 1]), chart[j + 1])
end

procedure COMPLETER((B Æ ... ., [j, k]))
begin
    for each (A Æ ... . B ..., [i, j]) in chart[j] do
        ADDTOCHART((A Æ ... B . ..., [i, k]), chart[k])
end

procedure ADDTOCHART(state, state-list)
begin
    if state is not in state-list
        then ADDTOEND(state, state-list)
end
Fig 15.5 Transition network definition of a simple English grammar.
Pseudo-code for a transition network parser appears on the following two slides. It is defined using two mutually recursive functions, `parse` and `transition`.

```plaintext
begin
    save pointer to current location in input stream;
    case

    grammar_symbol is a terminal:
        if grammar_symbol matches the next word in the input stream
        then return (success)
        else begin
            reset input stream;
            return (failure)
        end;

    grammar_symbol is a nonterminal:
        begin
            retrieve the transition network labeled by grammar symbol;
            state := start state of network;
            if transition(state) returns success
            then return (success)
            else begin
                reset input stream;
                return (failure)
            end
        end

    end
end.
```

continued...
function transition (current_state);

begin
  case

    current_state is a final state:
    return (success)

    current_state is not a final state:
    while there are unexamined transitions out of current_state
    do begin
      grammar_symbol := the label on the next unexamined transition;
      if parse(grammar_symbol) returns (success)
      then begin
        next_state := state at end of the transition;
        if transition(next_state) returns success;
        then return (success)
      end
    end

  end

return (failure)

end
end.
Fig 15.6 Trace of a transition network parse of the sentence “Dog bites.”
Fig 15.7 Structures representing the sentence, noun phrase, and verb phrase nonterminals of the grammar.
## Fig 15.8 Dictionary entries for a simple ATN

<table>
<thead>
<tr>
<th>Word</th>
<th>Definition</th>
<th>Word</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>PART_OF_SPEECH: article</td>
<td>like</td>
<td>PART_OF_SPEECH: verb</td>
</tr>
<tr>
<td></td>
<td>ROOT: a</td>
<td></td>
<td>ROOT: like</td>
</tr>
<tr>
<td></td>
<td>NUMBER: singular</td>
<td></td>
<td>NUMBER: plural</td>
</tr>
<tr>
<td>bite</td>
<td>PART_OF_SPEECH: verb</td>
<td>likes</td>
<td>PART_OF_SPEECH: verb</td>
</tr>
<tr>
<td></td>
<td>ROOT: bite</td>
<td></td>
<td>ROOT: like</td>
</tr>
<tr>
<td></td>
<td>NUMBER: plural</td>
<td></td>
<td>NUMBER: singular</td>
</tr>
<tr>
<td>bites</td>
<td>PART_OF_SPEECH: verb</td>
<td>man</td>
<td>PART_OF_SPEECH: noun</td>
</tr>
<tr>
<td></td>
<td>ROOT: bite</td>
<td></td>
<td>ROOT: man</td>
</tr>
<tr>
<td></td>
<td>NUMBER: singular</td>
<td></td>
<td>NUMBER: singular</td>
</tr>
<tr>
<td>dog</td>
<td>PART_OF_SPEECH: noun</td>
<td>men</td>
<td>PART_OF_SPEECH: noun</td>
</tr>
<tr>
<td></td>
<td>ROOT: dog</td>
<td></td>
<td>ROOT: man</td>
</tr>
<tr>
<td></td>
<td>NUMBER: singular</td>
<td></td>
<td>NUMBER: plural</td>
</tr>
<tr>
<td>dogs</td>
<td>PART_OF_SPEECH: noun</td>
<td>the</td>
<td>PART_OF_SPEECH: article</td>
</tr>
<tr>
<td></td>
<td>ROOT: dog</td>
<td></td>
<td>ROOT: the</td>
</tr>
<tr>
<td></td>
<td>NUMBER: plural</td>
<td></td>
<td>NUMBER: plural or singular</td>
</tr>
</tbody>
</table>
Fig 15.9 An ATN grammar that checks number agreement and builds a parse tree.

```
sentence:

function sentence-1;
begin
  NOUNPhrase := structure returned by noun_phrase network;
  SENTENCE.SUBJECT := NOUNPhrase;
end.

function sentence-2;
begin
  VERBPhrase := structure returned by verb_phrase network;
  if NOUNPhrase.NUMBER = VERBPhrase.NUMBER
    then begin
      SENTENCE.VERB.PHRASE := VERBPhrase;
      return SENTENCE
    end
  else fail
end.
```

continued...
Fig 15.9 continued from previous slide.

```
function noun_phrase-1;
  begin
    ARTICLE := definition frame for next word of input;
    if ARTICLE.PART_OF_SPEECH=article
      then NOUN PHRASE.DETERMINER := ARTICLE
    else fail
  end.

function noun_phrase-2;
  begin
    NOUN := definition frame for next word of input;
    if NOUN.PART_OF_SPEECH=noun and
      NOUN.NUMBER agrees with
      NOUN PHRASE.DETERMINER.NUMBER
      then begin
        NOUN PHRASE.NOUN := NOUN;
        NOUN PHRASE.NUMBER := NOUN.NUMBER
        return NOUN_PHASE
      end
    else fail
  end.
```

continued...
Fig 15.9 continued from previous slide.

function noun_phrase-3
begin
    NOUN := definition frame for next word of input;
end
if NOUN.PART_OF_SPEECH=noun then begin
    NOUN PHRASE.DETERMINER := unspecified;
    NOUN PHRASE.NOUN := NOUN
    NOUN PHRASE.NUMBER := NOUN.NUMBER
end
else fail
end.

function verb_phrase-1
begin
    VERB := definition frame for next word of input;
end
if VERB.PART_OF_SPEECH=verb then begin
    VERB PHRASE.VERB := VERB;
    VERB PHRASE.NUMBER := VERB.NUMBER;
end;
end.

continued...
Fig 15.8 continued from previous slide.

function verb_phrase-2
begin
  NOUN_PHRASE := structure returned by noun_phrase network;
  VERB_PHRASE.OBJECT := NOUN_PHRASE;
  return VERB_PHRASE
end.

function verb_phrase-3
begin
  VERB := definition frame for next word of input;

  if VERB.PART_OF_SPEECH=verb
    then begin
      VERB_PHRASE.VERB := VERB;
      VERB_PHRASE.NUMBER := VERB.NUMBER;
      VERB_PHRASE.OBJECT := unspecified;
      return VERB_PHRASE;
    end;
end.
Fig 10 Parse tree for the sentence “The dog likes a man” returned by an ATN parser.
Fig 15.11 Type hierarchy used in “dogs world” example.
Fig 15.12 Case frames for the verbs “like” and “bite.”
Rules for our example are described as pseudo-code procedures. In each procedure, if a specified join or other test fails, that interpretation is rejected as semantically incorrect.

procedure sentence;
    begin
        call noun_phrase to get a representation of the subject;
        call verb_phrase to get a representation of the verb_phrase;
        using join and restrict, bind the noun concept returned for the subject to
        the agent of the graph for the verb_phrase
    end.

procedure noun_phrase;
    begin
        call noun to get a representation of the noun;
        case
            the article is indefinite and number singular: the noun concept is generic;
            the article is definite and number singular: bind marker to noun concept;
            number is plural: indicate that the noun concept is plural
        end case
    end.
procedure verb_phrase;
    begin
        call verb to get a representation of the verb;
        if the verb has an object
            then begin
                call noun_phrase to get a representation of the object;
                using join and restrict, bind concept for object to object of the verb
            end
        end
    end.

procedure verb;
    begin
        begin
            retrieve the case frame for the verb
        end.
    end.

procedure noun;
    begin
        begin
            retrieve the concept for the noun
        end.
Fig 15.13 Construction of a semantic representation from the parse tree of Figure 15.10.
Fig 15.14 Two different parses of “Print the file on the printer.”
Fig 15.15 Conceptual graph for the question “Who loves Jane?”
Fig 15.16 Two relations in an employee database.

<table>
<thead>
<tr>
<th>employee</th>
<th>manager</th>
<th>employee</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>Jane Martinez</td>
<td>John Smith</td>
<td>$35,000.00</td>
</tr>
<tr>
<td>Alex Barrero</td>
<td>Ed Angel</td>
<td>Alex Barrero</td>
<td>$42,000.00</td>
</tr>
<tr>
<td>Don Morrison</td>
<td>Jane Martinez</td>
<td>Don Morrison</td>
<td>$50,000.00</td>
</tr>
<tr>
<td>Jan Claus</td>
<td>Ed Angel</td>
<td>Jan Claus</td>
<td>$40,000.00</td>
</tr>
<tr>
<td>Anne Cable</td>
<td>Bob Veroff</td>
<td>Anne Cable</td>
<td>$45,000.00</td>
</tr>
</tbody>
</table>
Fig 15.17 Entity-relationship diagrams of the manager_of_hire and employee_salary relations.
Fig 15.18 Knowledge base entry for “hire” queries.
Fig 15.19 Development of a database query from the graph of a natural language input.

Semantic interpretation of natural language query:

```
Semantic interpretation of natural language query:

hire → agent → person: ?

object

person: "john smith"
```

Expanded graph for query:

```
Expanded graph for query:

hire → agent → manager: ?

object

manager_of_hire:

employee: "john smith"
```

Query in SQL database language:

```
SELECT MANAGER
FROM MANAGER_OF_HIRE
WHERE EMPLOYEE = "john smith"
```
Fig 15.20 Sample text, template summary, and information extraction for computer science advertisement.

Sample Computer Science Job Ad (an excerpt):

The Department of Computer Science of the University of New Mexico . . . is conducting a search to fill two tenure-track positions. We are interested in hiring people with research interests in:

- Software, including analysis, design, and development tools. . .
- Systems, including architecture, compilers, networks. . .

. . .

Candidates must have completed a doctorate in. . .

The department has internationally recognized research programs in adaptive computation, artificial intelligence, . . . and enjoys strong research collaborations with the Santa Fe Institute and several national laboratories. . .

Sample Partially Filled Template:

Employer: Department of Computer Science, University of New Mexico
Location City: Albuquerque
Location State: NM 87131
Job Description: Tenure track faculty
Job Qualifications: PhD in . . .
Skills Required: software, systems, . . .
Platform Experience: . . .
About the Employer: (text attached)
Fig 15.21 An architecture for information extraction, from Cardie (1997).

1. Text: The Department of Computer Science of the University of New Mexico is conducting a search to fill two track track positions. We are interested in hiring . . .

2. Tokenization and Tagging: The/The det Department/noun of/prep . . .

3. Sentence Analysis: Department/subj is conducting/verb search/obj . . .

4. Extraction: Employer: Department of Computer Science
Job Description: Tenure track position . . .

5. Merging: tenure track position = faculty
New Mexico = NM . . .

6. Template Generation: As in Figure 13.19