Machine Learning

Lecture 18

Learning in Natural Language Processing (NLP).

Methods of simulation of understanding of NL

Two approach to develop NLP-systems

- Syntactic-oriented
  - Sequential analyzing with all stages:
    - Morphologic analyzing
    - Syntactic analyzing
    - Semantic analyzing
    - Pragmatic analyzing
- Semantic-oriented
  - Recognition of semantics and pragmatics of sentences
  - Syntactic analyzing is little importance

Technologies of processing of NL

- Syntactic grammars
- Semantic grammars
- Extended nets of transitions
- Templates
- Case frames
- Neural networks

Syntactic analyzing

- A natural language grammar specifies allowable sentence structures in terms of basic syntactic categories such as nouns and verbs, and allows us to determine the *structure* of the sentence. It is defined in a similar way to a grammar for a programming language, though tends to be more complex, and the notations used are somewhat different. Because of the complexity of natural language a given grammar is unlikely to cover all possible syntactically acceptable sentences.
- To parse correct sentences:
  - John ate the biscuit.
  - The lion ate the schizophrenic.
  - The lion kissed John.
- To exclude incorrect sentences:
  - Ate John biscuit the.
  - Schizophrenic the lion the ate.
  - Biscuit lion kissed.

Simple context free grammar for previous examples

- sentence --> noun_phrase, verb_phrase.
- noun_phrase --> proper_name.
- noun_phrase --> determiner, noun.
- verb_phrase --> verb, noun_phrase.
- proper_name --> [Mary].
- proper_name --> [John].
- noun --> [schizophrenic].
- noun --> [biscuit].
- verb --> [ate].
- verb --> [kissed].
- determiner --> [the].

To enforce subject-verb agreement the simplest method is to add arguments to our grammar rules.

- sentence --> noun_phrase(Num), verb_phrase(Num).
- noun_phrase(Num) --> proper_name(Num).
- noun_phrase(Num) --> determiner(Num), noun(Num).
- verb_phrase(Num) --> verb(Num), noun_phrase(_).
- proper_name(sing) --> [mary].
- noun(sing) --> [lion].
- noun(plur) --> [lions].
- det(sing) --> [the].
- det(plur) --> [the].
- verb(sing) --> [eats].
- verb(plur) --> [eat].

To extend the grammar to allow adjectives we need to add an extra rule or two, e.g.,

- noun_phrase(Num) --> determiner(Num), adjectives, noun(Num).
- adjectives --> adjective, adjectives.
- adjectives --> adjective.
- adjective --> [ferocious].
- adjective --> [ugly].
- etc.

That is, noun phrases can consist of a determiner, some adjectives and a noun. Adjectives consist of an adjective and some more adjectives, OR just of an adjective. We can now parse the sentence ``the ferocious ugly lion eats Mary''.

Also about context free grammars

- Another thing we may need to do to our grammar is extend it so we can distinguish between *transitive* verbs that take an object (e.g., likes) and *intransitive* verbs that don't (e.g., talks). (``Mary likes the lion'' is OK while ``Mary likes'' is not. ``Mary talks'' is OK while ``Mary talks the lion'' is not).
- Our grammar so far (if we put all the bits together) still only parses sentences of a very simple form. It certainly wouldn't parse the sentences I'm currently writing! We can try adding more and more rules to account for more and more of English - for example, we need rules that deal with prepositional phrases (e.g., ``Mary likes the lion *with the long mane*''), and relative clauses (e.g., ``The lion *that ate Mary* kissed John'').
- As we add more and more rules to allow more bits of English to be parsed then we may find that our basic grammar formalism becomes inadequate, and we need a more powerful one to allow us to concisely capture the rules of syntax. There are lots of different grammar formalisms that have been developed (e.g., unification grammar, categorial grammar), but we won't go into them.

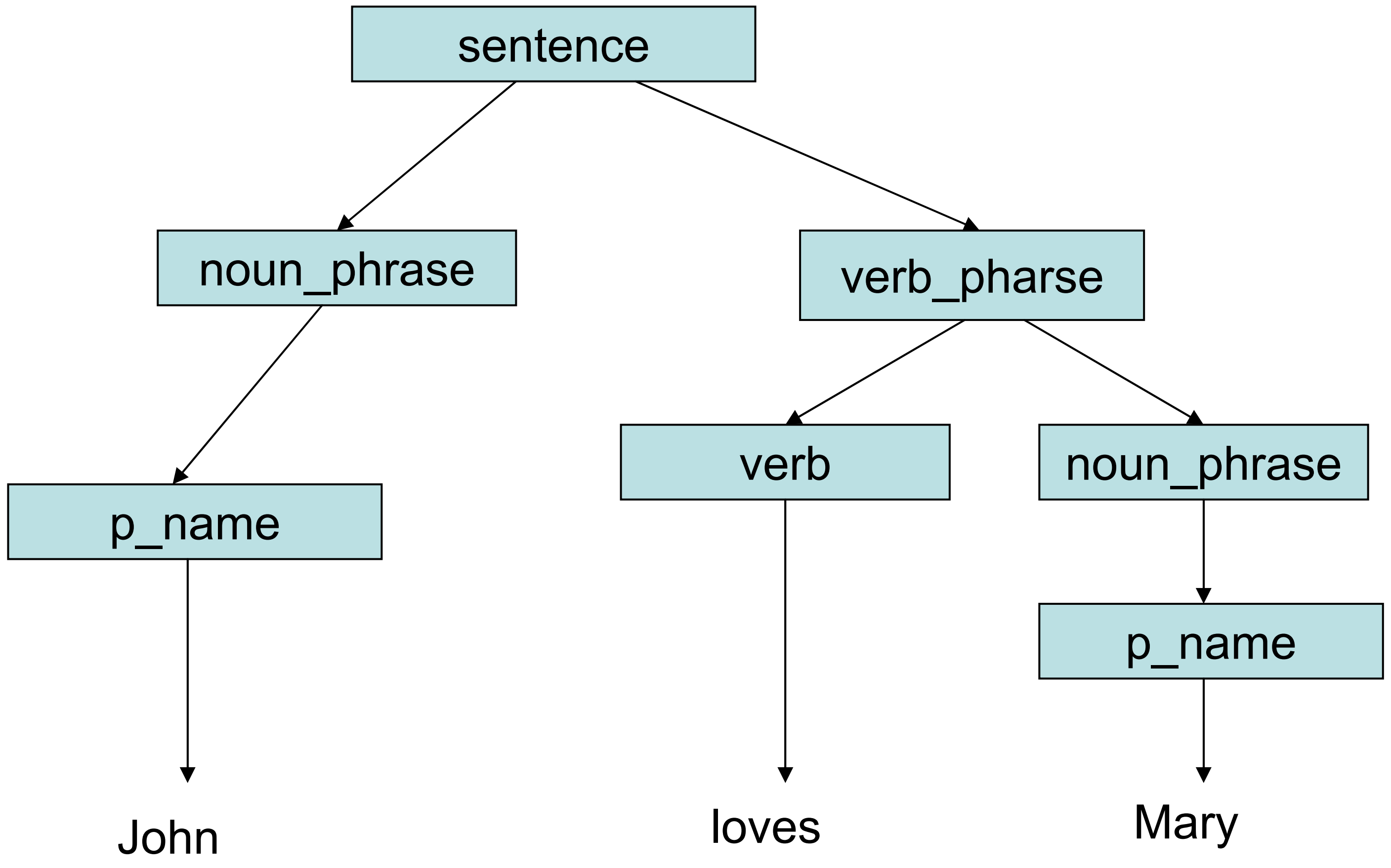
Grammars in Prolog

- The general formalism we have used above is based on Prolog's built in grammar formalism, known as *direct clause grammars*. Prolog allows you to write grammars directly using the notation above, but internally in translates it into ordinary Prolog rules. A rule such as:
  - sentence --> noun_phrase, verb_phrase.
- is translated internally into a form like:
  - sentence(Words, Remainder) :- noun_phrase(Words, NPRemainder), verb_phrase(NPRemainder, Remainder).
- A grammar rule takes a list of words (or other items) as its first argument and instantiates its second argument to the remaining words once some of the list has been parsed using the rule. So
  - ?- noun_phrase([john, hit, mary], X)
- would return X=[hit, mary].
- You shouldn't need to fully understand this internal representation when using Prolog's grammar facilities - However, when checking whether a sentence parses (given a grammar like the ones in the last section) you have to give a query like the following:
  - ?- phrase(sentence, [john, likes, mary]).

Parsing

- As an example, suppose you were trying to parse ``John loves Mary'' given the following grammar:
- sentence --> noun_phrase, verb_pharse.
- verb_phrase --> verb, noun_phrase.
- noun_pharse --> det, noun.
- noun_phrase --> p_name.
- verb --> [loves].
- p_name --> [john].
- p_name --> [mary].

Parsing (2)


sentence
noun_phrase
verb_pharse
p_name
verb
noun_phrase
p_name
John
loves
Mary

Parsing (3)

- You might start off expanding “sentence" to a verb phrase and a noun phrase. Then the noun phrase would be expanded to give a determiner and a noun, using the third rule. A determiner is a primitive syntactic category (a terminal node in the grammar) so we check whether the first word (John) belongs to that category. It doesn't - John is a proper noun - so we backtrack and find another way of expanding “noun_phrase" and try the fourth rule. Now, as John is a proper name this will work OK, so we continue the parse with the rest of the sentence (“loves Mary"). We haven't yet expanded verb phrase, so we try to parse “loves Mary" as a verb phrase. This will eventually succeed, so the whole thing succeeds.
- It may be clear by now that in Prolog the parsing mechanism is really just Prolog's built in search procedure. Prolog will just backtrack to explore the different possible syntactic structures.

Parsing (4)

- Simple parsers are often inefficient, because they don't keep a record of all the bits of the sentence that have been parsed. Using simple depth first search with backtracking can result in useful bits of parsing being undone on backtracking: if you have a rule ``a -> b, c, d'' and a rule ``a -> b, c, e'' then you may find the first part of the sentence consists of ``b'' and ``c'' constituents, go on to check if the rest is a ``d'' constituent, but when that fails a Prolog-like system will through away its conclusions about the first half when backtracking to try the second rule. It will then duplicate the parsing of the ``b'' and ``c'' bits. Anyway, better parsers try to avoid this. Two important kinds of parsers used in natural language are *transition network* parsers and *chart* parsers.

Parsing (5)

- For a parse to be useful we often want to return the parse tree, once we've parsed the sentence. This may then be used by a semantic component, to help determine the meaning of the sentence. [We sometimes do the semantic processing at the same time as the parsing, making this unecessary. Both approaches are sometimes used.]
- Anyway, in Prolog we return the parse tree by adding extra arguments to our grammar rules, like the following:
- 3 noun_phrase(np(DetTree, NounTree)) --> determiner(DetTree), noun(Num, NounTree). noun(noun(banana)) --> [banana].

Multiple parses

- In general, as discussed earlier, there may be many different parses for a complex sentence, as the grammar rules and dictionary allow the same list of words to be parsed in several different ways. A commonly cited example is the pair of sentences:
- Time flies like an arrow.
- Fruit flies like a banana.
- “Flies'' can be either a verb and a noun, while ``likes'' can be either a verb and a preposition (or whatever). So, in the first sentence ``Time'' should be the noun phrase and ``flies like an arrow'' to be the verb phrase (with ``like an arrow'' *modifying* flies). In the second sentence ``Fruit flies'' should be the noun phrase and “like a banana'' to be the verb phrase. Now, WE know which is the ``right'' parse because we know that there is no such thing as a “time fly'' and it would be a bit strange to “fly like a banana''. But without such general knowledge about word meanings we coudn't tell which parse is correct, so a parser, with no semantic component, should return both parses, and leave it up to the semantic stage of analysis to throw out the bogus one.

Disadvantages of syntactic analyzing

- For real natural language grammar became very large
- Difficulties to implement of analyzing of ambiguities and using of context
- Often correct recognized structure hasn’t sense and vise versa

Conclusion:

Syntactic-oriented analyzing is not perspective approach

Semantic grammars

- Deal with words-parts (categories) of language but with parts (concepts) of domain

Example of grammar:

S → <present> the <attribute> of <ship>
<present> → what is | [can you] tell me
<ship> → the <shipname> | <classname> class ship

Can you tell me the class of the Enterprise? (*Enterprise* - name of ship).

Disadvantages of semantic grammars

- For real world grammar became very large

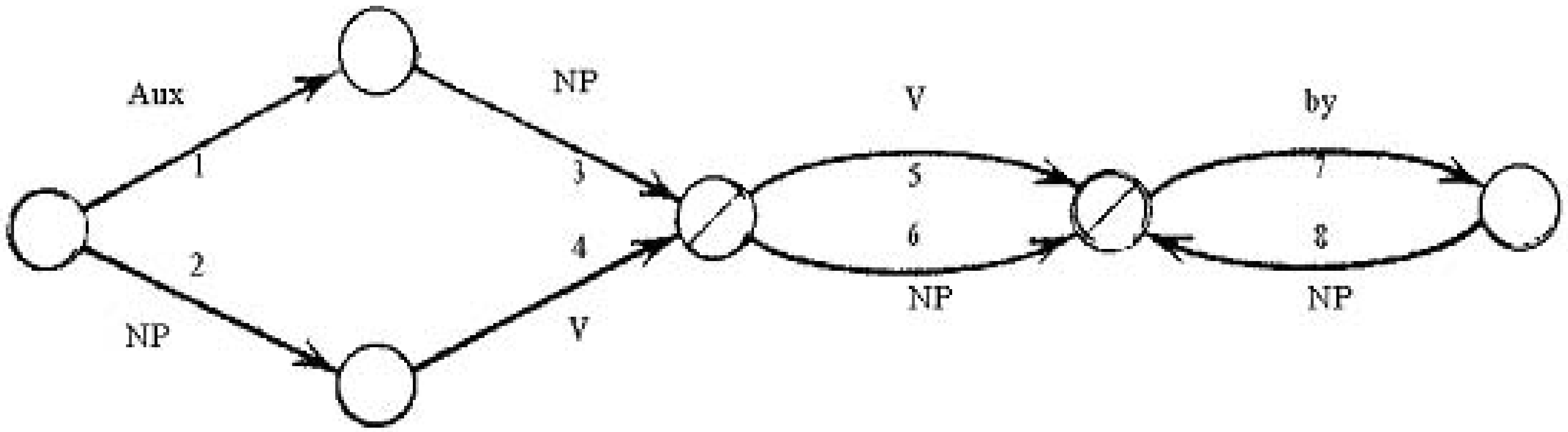
Conclusion:

Semantic grammar may be used in enough special domain (for example, for different referral services)

Transition networks

- Based on states and transitions between them
- Transitions are executed when determined class of words is read in sentence (in nodes any test is executed, result of it is selection of transition)
- Results of test stores as any information about recognized words for later analyzing
  - Subject
  - Object
  - Verb
  - Kind of sentence (declarative, question, order and so on)
  - And so on

Example of analyzing



- *"The rabbit nibbles the carrot"* is recognized by this graph
  - Result of testing of "*The rabbit*" is transition 2 and storing of subject=the rabbit, kind of sentence=declarative
  - result of testing of "*nibbles*" is transition 4 and storing verb=nibbles
  - Result of testing of "*the carrot*" is transition 6 and storing of object=the carrot

Disadvantages

- Non modularity
- Difficulties at modifications
- Too sensitivity to incorrect sentences (mistake in sentence does impossible continue analyzing)
- Sometimes result of analyzing of correct sentence has not sense

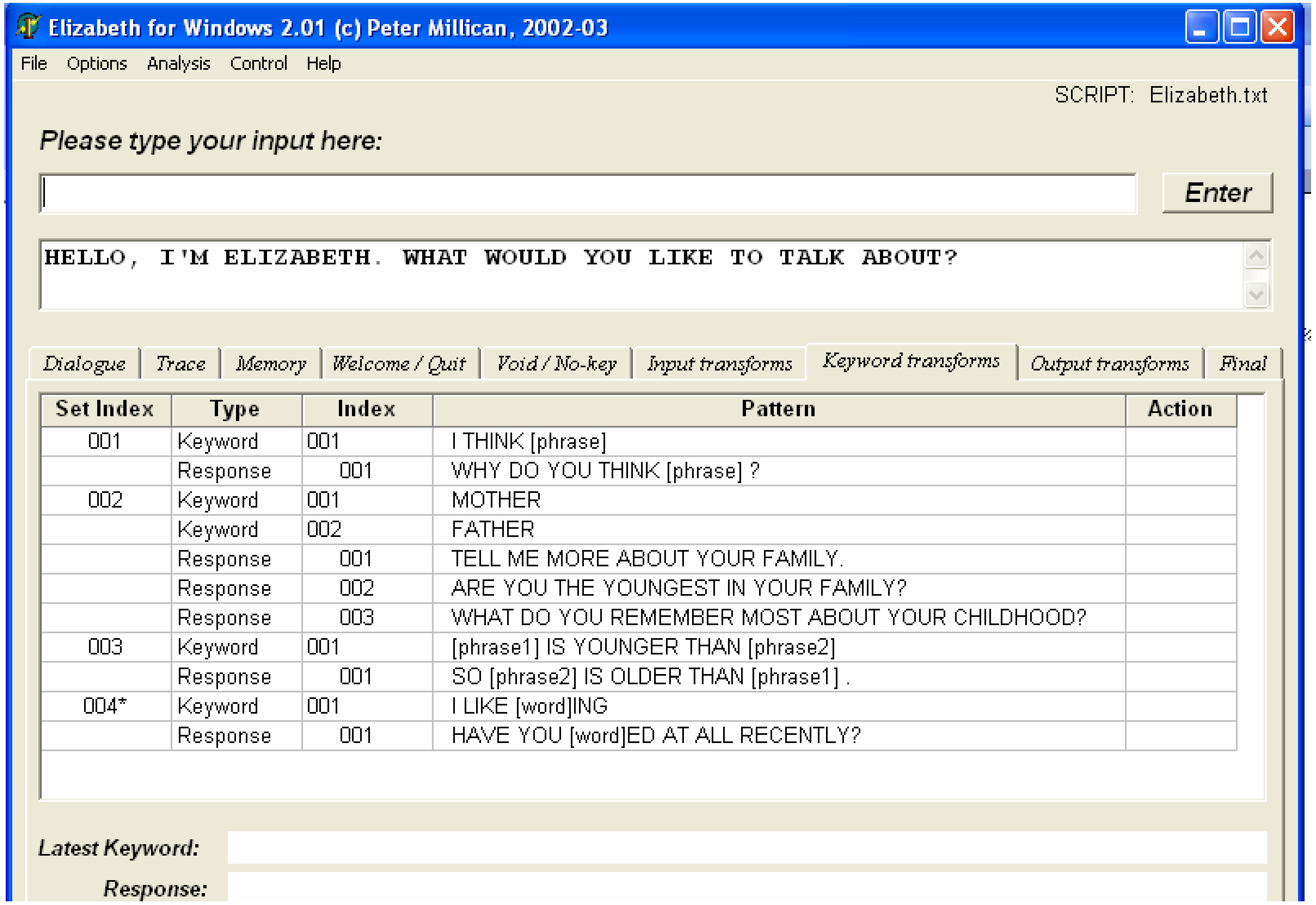
Templates

- This is first method for dialog programs (implemented in Eliza)
- Based on matching of sentence (pattern) with templates
- Template includes answer – response on sentence
- Template can to include words of NL (key words) and variables (during matching any words from sentence may be assigned to variables)
- Values of variables may be used in responses

A typical conversation with ELIZA is the following:

- C: Well, my boyfriend made me come here.
- R: Your boyfriend made you come here.
- C: I am unhappy.
- R: Do you think coming here will help you not to be unhappy?
- C: I need some help, that much seems certain.
- R: What would it mean to you if you got some help?
- C: Perhaps I could learn to get along with my mother.
- R: Tell me more about your family.
- C: You are like my father in some ways.
- R: What resemblance do you see?
- C: My father is afraid of everybody.
- R: What else comes to mind when you think of your father.
- C: Bullies.
- R: Does that have anything to with the fact that your boyfriend made you come here?

Program Elizabeth



SCRIPT: Elizabeth.txt

Please type your input here:

```
HELLO, I'M ELIZABETH. WHAT WOULD YOU LIKE TO TALK ABOUT?
```

Set Index	Type	Index	Pattern	Action
001	Keyword	001	I THINK [phrase]	
	Response	001	WHY DO YOU THINK [phrase] ?	
002	Keyword	001	MOTHER	
	Keyword	002	FATHER	
	Response	001	TELL ME MORE ABOUT YOUR FAMILY.	
	Response	002	ARE YOU THE YOUNGEST IN YOUR FAMILY?	
	Response	003	WHAT DO YOU REMEMBER MOST ABOUT YOUR CHILDHOOD?	
003	Keyword	001	[phrase1] IS YOUNGER THAN [phrase2]	
	Response	001	SO [phrase2] IS OLDER THAN [phrase1] .	
004*	Keyword	001	I LIKE [word]ING	
	Response	001	HAVE YOU [word]ED AT ALL RECENTLY?	

Latest Keyword:

Response:

Examples of templates for program ALICE (language AIML)

- <category>
- <pattern>YES</pattern>
- <template>Ok. Next question? </template>
- </category>
- <category>
- <pattern>WHY *</pattern>
- <template>
- <random>
- <li>I wonder myself. </li>
- <li>Yes, why? </li>
- </random>
- </template>
- </category>

Case frames

- Structure of case frame:
  - Verb – name of frame
  - Roles:
    - Agent
    - Object
    - Tool
    - Recipient
    - Direction
    - Place
    - Coagent

Example:

- Give
  - Role:
    - » Agent: John
    - » Object: ball
    - » Recipient: Mary
  - Gram:
    - » Time: past
    - » Voice: active

Roles may be obligatory and not obligatory

Analyzing with case frames

1. Select main verb from sentence or context
2. Select for this verb corresponding frame
3. Matching of words in sentence and obligatory roles in frame. If not all obligatory roles are found in sentence or context, then try find another frame and repeat step 3. If another frame is not found, analyzing of sentence is impossible
4. Matching of words in sentence and not obligatory roles in frame.
5. If after that there are not analyzed words in sentence, then may be one of different strategies:
    1. Ignore of these words and successful finish analyzing
    2. To try continue analyzing by other ways
    3. To finish analyzing as fail

Pragmatics

- Pragmatics is the last stage of analysis, where the meaning is elaborated based on contextual and world knowledge. Contextual knowledge includes knowledge of the previous sentences (spoken or written), general knowledge about the world, and knowledge of the speaker.
- One important task at this stage are to work out *referents* of expressions. For example, in the sentence ``he kicked the brown dog'' the expression ``the brown dog'' refers to a particular brown dog (say, Fido). The pronoun ``he'' refers to the particular guy we are talking about (Fred). A full representation of the meaning of the sentence should mention Fido and Fred.

Pragmatics (2)

- We can often find this out by looking at the previous sentence, e.g.:
  - Fred went to the park.
  - He kicked the brown dog.
- We can work out from this that ``he'' refers to Fred. We might also guess that the brown dog is in the park, but to work out that we mean Fido we'd need some extra general or contextual knowledge - that the only brown dog that generally frequents the park is Fido. In general this kind of inference is pretty difficult, though quite alot can be done using simple strategies, like looking at who's mentioned in the previous sentence to work out who ``he'' refers to. Of course, sometimes there may be two people (or two dogs) that the speaker might be referring to, e.g.,
  - There was a brown dog and a black dog in the park. Fred went to the park with Jim. He kicked the dog.
- In cases like this we have *referential ambiguity*. It is seldom quite as explicit as this, but in general can be a big problem. When the intended referent is unclear a natural language dialogue system may have to initiate a *clarification subdialogue*, asking for example ``Do you mean the black one or the brown one.''.

Pragmatics (3)

- Anyway, another thing that is often done at this stage of analysis (pragmatics) is to try and guess at the *goals* underlying utterances. For example, if someone asks how much something is you generally assume that they have the goal of (probably) buying it. If you can guess at people's goals you can be a bit more helpful in responding to their questions. So, an automatic airline information service, when asked when the next flight to Paris is, shouldn`t just say ``6pm'' if it knows the flight is full. It should guess that the questioner wants to travel on it, check that this is possible, and say ``6pm, but it's full. The next flight with an empty seat is at 8pm.''