Grundlagen der Wissensverarbeitung
Blatt 5

Daniel Speck, Lena Niermeyer

14.11.2015

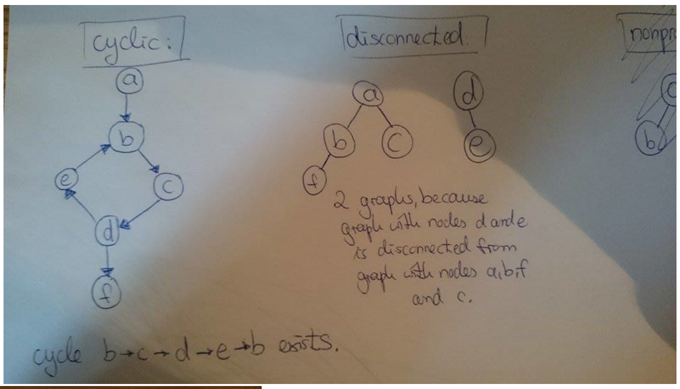
Exercise 1.2: (Search and Parsing)

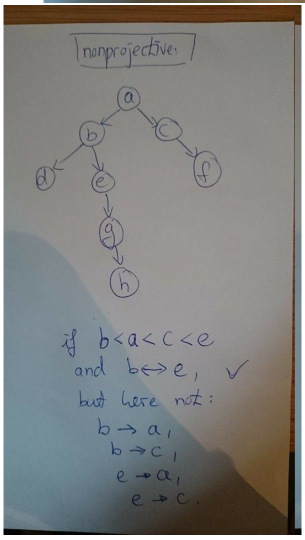
1.a) By what operations is the input transformed into the output? What do these operations do?
The input is a sentence of length n, which should be grammatically correct. Output is the dependency tree. The following operations transform the sentence into the tree: Left-Arc, Right-Arc, Reduce and Shift.
Shift pushes the next input token n onto the stack. Reduce pops the node n on top of the stack. Left-Arc pops the node n on top of the stack and adds the arc relation (n',n), where n' is the next input token. Right-Arc pushes the next input token n' on top of the stack (before that, n was top of the stack) and adds the arc relation (n,n').

b) When does the parsing algorithm terminate?
The parsing algorithm terminates, if the triple is of the form ⟨S,nil,A⟩. S stands for the list in the stack, nil means, that the sentence W was accepted and no input tokens remain in the list I. A is the set of arc relations.

c) Describe the formal properties of a dependency tree as defined in the paper.
Dependency trees / graphs should be acyclic, connected and projective.
Acyclic (no loops) means, that two nodes, who are neighbours, can only be connected via one directed arc. Otherwise they would create a cycle.
Connected means, that every node needs minimum one neighbour. If one node has no neighbour it would be a single node, like a word alone, not part of the sentence, which has to be transformed into a dependency tree.
Projective means, that, if the sentence has a relation from n to n', when n appears before n" and n" before n' in the sentence, then there must be an arc from n to n" or an arc from n' to n". For exapmle: The sentence "John saw a dog": saw is n, a is n", dog is n'. saw has an arc to dog and dog has an arc to a. So it's projective.

d) For each property: Give an example dependency tree that violates the property. Note: We ask for a tree, not a sentence! Do not try to find a matching sentence to your trees, it will only distract you.


cyclic:
a
b
c
d
e
f
cycle b→c→d→e→b exists.
disconnected
a
b
c
d
e
f
2 graphs, because graph with nodes d and e is disconnected from graph with nodes a,b,f and c.
nonpr



2. Try to use the proposed parser actions to produce the tree depicted in Figure 1. Write down the steps and the intermediate states.
W = Der Mann isst eine Giraffe.
R ⊇ {Der ← Mann,
Mann ← isst,
isst → Giraffe,
eine ← Giraffe }
⟨ nil, Der Mann isst eine Giraffe, ∅ ⟩ [Initialization]
→ w/ S ⟨Der, Mann isst eine Giraffe, ∅ ⟩
→ w/ LA ⟨nil, Mann isst eine Giraffe, {(Mann, Der)}⟩
→ w/ S ⟨Mann, isst eine Giraffe, {(Mann, Der)}⟩
→ w/ LA ⟨nil, isst eine Giraffe, {(Mann, Der), (isst, Mann)}⟩
→ w/ S ⟨isst, eine Giraffe, {(Mann, Der), (isst, Mann)}⟩
→ w/ S ⟨eine isst, Giraffe, {(Mann, Der), (isst, Mann)}⟩
→ w/ LA ⟨isst, Giraffe, {(Mann, Der), (isst, Mann), (Giraffe, eine)}⟩
→ w/ RA ⟨Giraffe isst, nil, {(Mann, Der), (isst, Mann), (Giraffe, eine), (isst, Giraffe)}⟩ [Termination]

3. If you view parsing using the proposed parsing algorithm as a search problem.

What are the search states?
The search states are represented by the triples ⟨S,I,A⟩, where S is the stack, I the input tokens and A the arc relations. This triple changes with the Parser

transitions.

What is the start state?
The start state is the Initialization ⟨nil,W,∅⟩, where nil is an empty stack, W is the sentence which has to be transformed into a dependency tree and the empty set is placeholder for the arc relations to come.

What are the end states?
The parsing algorithm terminates, if the triple is of the form ⟨S,nil,A⟩. S stands for the list in the stack, nil means, that the sentence W was accepted and no input tokens remain in the list I. A is the set of arc relations. Termination has no condition, except that the input token list has to be empty. So it's possible to have more then one end state, because for example
⟨Giraffe isst, nil, {(Mann, Der), (isst, Mann), (Giraffe, eine), (isst, Giraffe)}⟩
is an end state, but it is possible to reduce it to
⟨isst, nil, {(Mann, Der), (isst, Mann), (Giraffe, eine), (isst, Giraffe)}⟩ or
⟨nil, nil, {(Mann, Der), (isst, Mann), (Giraffe, eine), (isst, Giraffe)}⟩, which are also end states.

What are the state transitions?
The state transitions are the parser transitions Left-Arc, Right-Arc, Reduce and Shift. They transform the states to different states, by changing the stack, input token list or arc relations list.

Can the search space be created before parsing starts?
It has to. We need sentence W for the Initialization.

What is the advantage of the proposed algorithm in contrast to simply trying to find a good dependency tree by enumerating all possible trees and selecting the best one from them?
Because this would take more time. For example: The sentence "Der Mann isst eine Giraffe". We needed 8 parser transitions to get the dependency tree. But we could find 5! = 120 trees for this sentence (5 faculty, because the sentence has 5 words and we could find 120 combinations to draw a tree).

For the search strategies discussed so far: are they a good fit for this search problem and why (not)?
No they aren't. BFS marks the nodes, which appaear in the same generation of a tree. But it doesn't create an arc relation of neighboured nodes to save the information about grammatical relations. Similar problem with DFS. A* may have grammar information, but also does not save arc relations.

How would you design a parser using the parser actions together with an appropriate search procedure?
Create consistency parse tree. Search sentence for root node (within a sentence, there is only ever one root node. It is the word with no arc directed on it, just from it.). Then find branch nodes, which connect to child nodes. We get this information from the grammar (relations).