GWV-Abgabe zum 7.11.2014

Arne Struck, Knut Götz

14. November 2014

1

a)

Der Algorithmus erhält einen Satz und eine eine Grammtik, in der neben Terminalen (hier die einzelnen Wörter des Satzes) auch eine Menge R an Regeln gegegeben ist. Der Output ist ein gerichteter Graph (ein Baum). Da die Wörter des Satzes die Knoten des Graphens bilden muss diese Menge im Algorithmus nicht verändert werden. Der Algorithmus erzeugt sukzessive die Menge der Kanten, anhand der gegebenen Regeln der Grammatik. Der aktuelle Status des Parses ist gegeben durch eine Stack S, einer Liste von Wörtern des Satzes (in der Reihenfolge wie im Satz) und die Menge der Kanten. Folgende Operationen stehen dabei zur Verfügung:

- Left-Arc:
  Fügt eine Kante zwischen dem ersten Wort in der Wortliste zu dem Top of Stack Wort hinzu und popped den Stack. (Bedingungen auch angeben?)
- Right-Arc:
  Fügt eine Kante zwischen dem Top of Stack Wort und ersten Wort der Inputliste hinzu. Dabei wird das erste Wort aus der Inputliste entfernt und auf den Stack gepusht.
- Reduce:
  Popt Top of Stack, falls es bereits einen Kante zu diesem Wort in der Kantenmenge gibt.
- Shift:
  Entfernt das erste Wort der Inputliste und pusht es auf den Stack.

b)

Der Algorithmus terminiert, wenn die Liste mit dem Input Wörtern leer ist. Dann kann der entstandenen Dependency Graph zurückgeben werden.

c)

Ein dependency Graph muss folgende vier Bedingungen erfüllen:

- Single Head:
  Diese Eigenschaft fordert, dass jedes Wort nur einem übergeordneten Wort zugeordnet ist, d.h. es gibt maximal eine eingehende Kante pro Wort.
- Acyclic:
  Diese Eigenschaft fordert einfach, dass der dependency Graph keine Zyklen aufweist.

- Connected:
  Diese Eigenschaft fordert, dass der Graph zusammenhängend ist, d.h. wandelt man alle gerichetete Kanten in ungerichtete Kanten um, dann gibt es von jedem Knoten einen Pfad zu einem anderen Knoten.

- Projective:
  Gibt es eine Verbindung zwischen zwei Wörtern und es gibt ein Wort zwischen diesen Wörtern. Dann muss es einen gerichteten Pfad von einem dieser beiden Wörtern zu dem dazwischenliegenden geben.

d)

Man kann als Beispielsatz "Das ist ein Satz" verwenden, dann gilt:

$$N_w = \{Das, ist, ein, Satz\}$$

Hier jeweils ein Graph die eine Eigenschaft nicht erfüllen.

- Single Head:

$$A = \{(Das, ist), (ein, Satz), (Das, Satz)\}$$

- Acyclic:

$$A = \{(Das, ist), (ein, Satz), (Satz, ein)\}$$

- Connected:

$$A = \{(ist, Das), (ist, Satz)\}$$

- Projective:

$$A = \{(Das, Satz)\}$$

2

Input:

$$\begin{aligned} \text{List} &= \{\text{Der, Mann, isst, eine, Giraffe}\} \\ \text{Stack} &= \{\} \\ A &= \{\} \end{aligned}$$

Shift:

$$\begin{aligned} \text{List} &= \{\text{Mann, isst, eine, Giraffe}\} \\ \text{Stack} &= \{\text{Der}\} \\ A &= \{\} \end{aligned}$$

Left-Arc:

List = {Mann, isst, eine, Giraffe}
Stack = {}
A = {(Mann, Der)}

shift:

List = {Mann, isst, eine, Giraffe}
Stack = {Mann}
A = {(Mann, Der)}

Left-Arc:

List = {isst, eine, Giraffe}
Stack = {}
A = {(isst, Mann), (Mann, Der)}

shift:

List = {eine, Giraffe}
Stack = {isst}
A = {(isst, Mann), (Mann, Der)}

shift:

List = {Giraffe}
Stack = {eine, isst}
A = {(isst, Mann), (Mann, Der)}

Left-Arc:

List = {Giraffe}
Stack = {isst}
A = {(Giraffe, eine), (isst, Mann), (Mann, Der)}

Right-Arc:

List = {}
Stack = {Giraffe, isst}
A = {(isst, Girafffe), (Giraffe, eine), (isst, Mann), (Mann, Der)}

3

TODO