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.
- 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.

Bedingung für diese beiden Operationen ist zum einen, dass sich eine entsprechende Regel in der Grammatik zu finden ist, und dass die Single-Head Eigenschaft nicht verletzt wird (dazu unten mehr).

- 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:

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

shift:

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

Left-Arc:

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

shift:

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

shift:

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

Left-Arc:

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

Right-Arc:

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

3

Wenn dann man den vorgestellten Algorithmus als Suchproblem auffassen sollte, kann man dies wie folgt modellieren: Jeder Zusand im Suchraum ist der aktuelle Zustand des Algorithmus, d.h. der Stack, der verbliebende Input und die Menge der gefundenen Kanten. Der Startzustand wäre somit der Zustand in dem der Stack leer ist, der komplette Input noch in der Liste steht und die Menge der Kanten die leer ist.
Die Endzustände sind dann die Zustände in denen die Inputliste leer ist und der enstandene Graph ein der Definition nach wohlgeformter dependency Tree ist (was beispielsweise nicht der Fall ist, wenn nur die Shift-Operation ausgeführt wird).
Die Nachbarn eines Zustands entsprechen, dann den Zuständen, die durch die Ausführung einer der in dem Algorithmus angegebenen Schritte (Shift, Reduce, Left Arc, Right Arc) erreicht werden können. Es gibt natürlich nicht von jedem Zustand immer vier Kanten, da nicht immer jede der Aktionen ausgeführt werden kann. Beispielsweise kann bei einem leeren Stack in keinem Fall noch ein Reduce ausgeführt werden.
Der Suchraum steht vor dem Parsen noch nicht fest, da er nur sukzessive aus den Zuständen errechnet werden kann (sofern man sich auf den Algorithmus beschränkt). Es kann erst lokal geprüft werden, welche Schritte möglich sind. Das durch den Algorithmus beschriebene Suchproblem ist besser als die Erstellung aller möglichen Bäume, da wie im Paper bewiesen nur ein linearen Zeitaufwand in Abhängigkeit der Inputlänge benötigt wird, während allein für das Berechnen aller möglichen Kombinationen einen weitaus höheren Aufwand von nöten wird (Alle Kombinationen und davon die Potenzmenge bilden) und für das finden des Besten noch einmal ein Aufwand $O(n)$ zu veranschlagen ist.
Von den bisher vorgestellten Verfahren (wir betrachten hier A*, BFS und DFS) bietet sich wohl DFS an. A* ist unmöglich, da der Suchraum unbekannt ist. BFS ist im Gegensatz zu DFS wesentlich speicherplatzintensiver und da in diesem Fall auch nicht das Auffinden eines kürzesten Pfades im Vordergrund steht, ist DFS zu bevorzugen.