GWV – Grundlagen der Wissensverarbeitung

Blatt 06:

Felix Degenhardt / Benjamin Cordt / Paul Hölzen

1.)

S is the stack.

I is the list of (remaining) input tokens.

A is the(current) arc relation for the dependency graph.

a.)

Right-Arc:

Hier wird eine Kante von dem Obersten Objekt aus **S** zu dem vordersten Objekt aus **I** geschaffen. Dies ist jedoch nur erlaubt wenn die existierende

Regeln die Verbindung erlauben und es noch keine Kante zum vordersten

Objekt von **I** in **A** vorhanden ist.

Des weiteren wird das vorderste Objekt von **I** auf den Stack gepackt.

Left-Arc:

Hier wird eine Kante von dem ersten Wort aus der Queue **I** zu dem obersten

Objekt aus dem Stack **S** geschaffen.

Dies ist jedoch nur erlaubt wenn die existierende Regeln die Verbindung erlauben und es noch keine Kante zum obersten Objekt des Stacks in der Menge der Kanten **A** gibt.

Reduce:

Hier wird das oberste Objekt des Stacks S entfernt falls dieses schon einen head hat.

Shift:

Packt das vorderste Objekt von **I** auf den Stack **S**.

b.)

Der Parsing-Algorithmus wird terminiert wenn die Queue **I** leer ist.

c. & d.)

Single head:

c.) Hier darf jeder Knoten des Graphen maximal einen anderen Knoten als head besitzen.

Soll heißen jeder Knoten darf nur von einem anderen Knoten abhängig sein.

d.)

$$!\{(a,n),(b,n)\} \quad (1)$$

Acyclic:

c.) Hier darf der Graph keine Zyklen enthalten.

Somit ist es nicht erlaubt das sich zwei Knoten, weder direkt noch indirekt gegenseitig als head haben.

d.)

$$!\{(a,b),(b,c),(c,a)\} \quad (2)$$

Connected:

c.) Hier muss der Graph zusammenhängend sein.

Somit ist also ein Knoten der transitive Kopf aller anderen Knoten

b.)

$$!\{(a,c),(b,d)\} \quad (3)$$

Projective:

c.) Hier müssen bei dem Graph alle abhängigen Knoten benachbart zu ihrem head sein.

Somit gibt es also keine Knoten zwischen einem Head und einem von ihm abhängigen Knoten, welcher nicht vom head direkt abhängt oder einem vom head anhängigen Knoten abhängt.

b)

$$!\{(a,n),(b,n)\} \quad (4)$$

2.)

Initialisierung

W= Der Mann isst eine Giraffe.

<, Der Mann isst eine Giraffe, >

Shift

<Der, Mann isst eine Giraffe, >

Left-Arc

<, Mann isst eine Giraffe, {(Mann, Der)}>

Shift

<Mann, isst eine Giraffe, {(Mann, Der)}>

Left-Arc

<, isst eine Giraffe, {(Mann, Der) (isst, Mann)}>

Shift

<isst, eine Giraffe, {(Mann, Der)(isst, Mann)}>

Shift

<eine isst, Giraffe, {(Mann, Der)(isst, Mann)}>

Left-Arc

<isst, Giraffe, {(Mann, Der)(isst, Mann),(Giraffe, eine)}>

Right-Arc

< Giraffe isst, ,{(Mann, Der)(isst, Mann),(Giraffe, eine)(isst, Giraffe)}>

Terminierung

3.)

- Bäume
- Ein leerer Baum
- Die fertigen Abhängigkeits- Bäume
- Die mit dem Algorithmus möglichen Aktionen
- Ja es ist möglich wenn man den zu Parsenden Satz kennt
- Die Vorteile sind das man viele Möglichkeiten evtl. nicht betrachten muss und somit schneller zu einem Ziel kommt
- Die Breitensuche wäre nicht geeignet da mit dieser fast alle möglichen Bäume angeschaut werden da bei dieser die Elemente nach und nach abgearbeitet und in eine Liste geschrieben werden. Und da der ganze Satz geparsed werden muss

Besser wäre die Tiefensuche da diese mit einem Stack arbeitet und so schneller zu einem Ziel gelangt da die Ziele immer an einem Blatt zu findne sind.

- Eine Standard Implementation der Tiefensuche welche an jedem Knoten Kanten für die im Algorithmus möglichen Aktionen entlang gehen kann um das Ziel zu suchen.