

GWV – Grundlagen der Wissensverarbeitung

Tutorial 5 : Searching

*Enrico Milutzki (6671784), Love Kumar (7195374),
Nikolai Poets (6911432), Jan Willruth (6768273)*

Exercise 6.1 : (Search and Parsing)



1.

a)

Left-Arc:

Fügt die Relation $n' \rightarrow n$, wobei n' das nächste Zeichen der Eingabe ist und n das oberste Element des Stacks, den Relationen (A) des dependency graphs hinzu und entfernt n vom Stack (S).

Right-Arc:

Fügt die Relation $n \rightarrow n'$ der Menge der Relationen (A) des dependency graphs hinzu und schiebt n' auf den Stack (S).

Reduce:



Entfernt das oberste Element n vom Stack.

Shift:

Schiebt das nächste Zeichen n der Eingabe (I) auf den Stack.

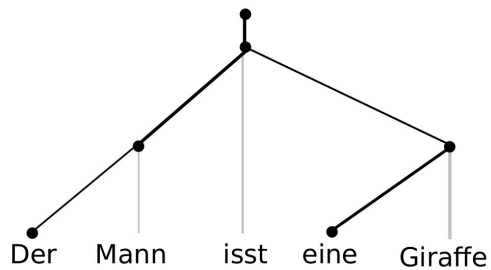
b)

$(S, \text{nil}, A) \rightarrow$ Wenn es keine verbleibenden input tokens mehr gibt.

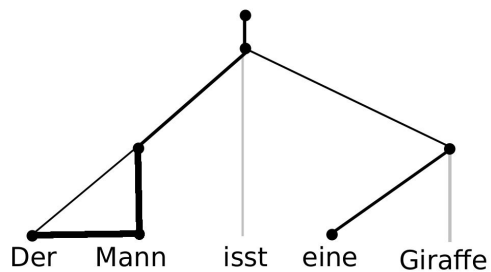
c&d)

(Beschreibung der Eigenschaft, darunter ein Graph, der die jeweilige Eigenschaft verletzt. Alle nachfolgenden Graphen sind nicht gerichtet)

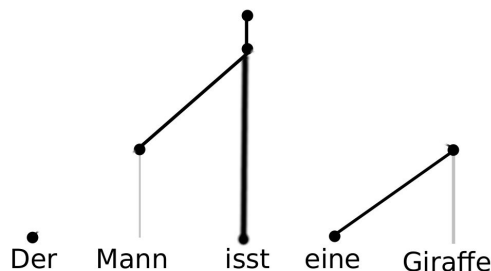
Gerichtet: Kanten gehen nur in eine Richtung.



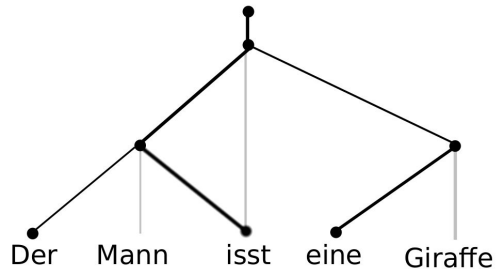
Azyklisch: Es gibt keine Kreise/Zyklen.



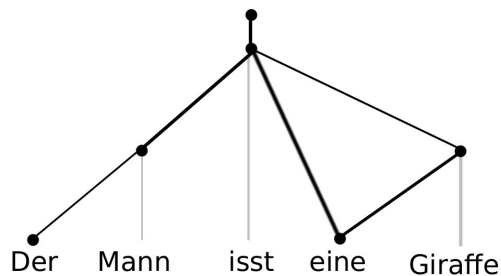
Zusammenhängend: Alle Teile des Graphen sind miteinander verbunden, d.h. es gibt von jedem Knoten einen Pfad zu einem anderen Knoten.



Projektiv: Ein dependency graph ist projektiv genau dann wenn jeder dependent node graph adjacent zu seinem head ist. Zwei verschiedene Knoten n und n' sind graph adjacent genau dann wenn ein weiterer Knoten n'' , der im Eingabe-String zwischen n und n' steht, im Graphen unter n und n' steht.



(Ein Kopf: Jeder Knoten hat nur einen Vorgänger; nur für well-formed dependency graphs)



2.

	<nil, Der Mann isst eine Giraffe, \emptyset >
Shift	<Der, Mann isst eine Giraffe, \emptyset >
Left-Arc	<nil, Mann isst eine Giraffe, {(Mann, Der)}>
Shift	<Mann, isst eine Giraffe, {(Mann, Der)}>
Left-Arc	<nil, isst eine Giraffe, {(isst, Mann), (Mann, Der)}>
Shift	<isst, eine Giraffe, {(isst, Mann), (Mann, Der)}>
Shift	<eine isst, Giraffe, {(isst, Mann), (Mann, Der)}>
Left-Arc	<isst, Giraffe, {(Giraffe, eine), (isst, Mann), (Mann, Der)}>
Right-Arc	<Giraffe isst, nil, {(isst, Giraffe), (Giraffe, eine), (isst, Mann), (Mann, Der)}>



3.




- Die Suchzustände lassen sich darstellen:

<S, W, A> mit A für die Menge der dependency-Relationen, W der zu parsende Satz und S für einen Stack als Zwischenspeicher

- Der Startzustand: <nil, W, \emptyset >

- Form: <S, nil, A>

- Die Zustandsübergänge sind Left-Arc, Right-Arc, Reduce und Shift

- Ja 

- Es lassen sich während der Suche schon viele sinnlose Bäume ausschließen ohne die kompletten Bäume zu erstellen, statt erst alle möglichen dependency trees zu erstellen und dann den besten zu suchen



- Tiefensuche, wenn man annimmt, dass die Suche erst terminiert, wenn der Baum wohlgeformt ist

