Tutorial 6: Search and Parsing

Finn-Lasse Jörgensen, Frederik Wille, Tronje Krabbe

November 28, 2016

Exercise 6.1 (Search and Parsing)

1. a) 1. Left-Arc: Pop ein Element vom Stack und erzeuge eine neue Kante auf dieses Element vom nächsten Input-Token.
   2. Right-Arc: Push ein Input-Token auf den Stack und erzeuge eine neue Kante von dem Token auf das erste Element des Stacks.
   3. Reduce: Pop ein Element vom Stack.
   4. Shift: Push das nächste Input Token auf den Stack.

   b) Der Algorithmus terminiert, sobald die Liste an Input-Tokens leer ist.

   c) * **azyklisch und zusammenhängend** Der Graph darf weder Zyklen noch Partitionen enthalten, da es sich sonst nicht um einen Baum handeln würde.
   * **projective** Aus dem Paper:

     > A dependency graph is projective iff every dependent node is graph adjacent to its head.

2. – Die search states sind Tripel der Form $\langle S, I, A \rangle$, wobei S der Stack ist, I der Input-Stack und A die Liste der Arc-Relationen.
   – Der start state ist ein state in dem S leer ist, I voll mit der Input-List, und A ebenfalls leer, da noch keine Relationen gefunden wurden.
   – Ein goal state wurde erreicht, wenn I leer ist und A 'well formed'.
   – Der search state könnte vor dem Parsen erstellt werden, was allerdings nicht effektiv wäre, da das Erstellen des search state mehr Aufwand ist, als das Suchen selbst.

– Der Algorithmus terminiert in linearer Zeit! Angenommen, n ist die Kardinalität von I, so terminiert er nach $2n$ Transitionen.
– Wir wissen, dass jeder Pfad höchstens $2n$ states lang sein kann. Im worst case müssten wir sowohl mit BFS als auch mit DFS alle Pfade ablaufen. Für A* eine Heuristik zu finden, ist ziemlich schwer. Insgesamt ist der Algorithmus wohl sehr viel besser, als die Verfahren, die wir bereits kennen.