Bachelor thesis
-
Pseudocode

Edser Apperloo

April 25, 2018

1 Grammar to Graph

A context-free Grammar G is defined as (V, Σ, P, S) with the following properties:

- V is an alphabet of non-terminals: $\{A, B, C, ...\}$
- Σ is an alphabet of terminals: $\{a, b, c, ...\}$
- $V \cap \Sigma = \emptyset$
- P is a set of production rules: $\{A \rightarrow \alpha | A \in V,\ \alpha \in (V \cup \Sigma)^*\}$
- if $(A \rightarrow \alpha) \in P$ then A is the left-hand-side (*lhs*) and α is the right-hand-side (*rhs*)
- S is the starting non-terminal of G: $S \in V$

A Graph G is defined as (V, E) with the following properties:

- V is a set of vertices in the graph
- E is a set of two-element tuples of vertices: $(A, B) \in E \implies A, B \in V$
- if $(A, B) \in E$ then A is the source and B is the target

For our graph the non-terminals are not important so they will be omitted. This results in the following pseudo-code:

Algorithm 1 grammar_to_graph

```
procedure GRAMMAR_TO_GRAPH(grammar)
    V ← grammar.V
    E ← {}
    Graph graph ← (V, E)
    for all nt ∈ grammar.N do
        reachable ← reachable_non_terminals(nt, grammar)
        for r ∈ reachable do
            graph.E.add((nt, r))
    return graph

/*get all non terminals reachable in 1 derivation step*/
procedure REACHABLE_NON_TERMINALS(nt, grammar)
    reachable ← {}
    /*If properly implemented this for-loop will sum up to O(|P|) instead of O(|V|∗|P|)*/
    for all {p|p ∈ grammar.P ∧ p.lhs == nt} do
        for all {token|token ∈ p.rhs ∧ token ∈ grammar.V} do
            reachable.add(symbol)
    return reachable
```