UNIVERSITÄT
BAYREUTH

University of Bayreuth

Institute for Computer Science

Bachelor Thesis

in Applied Computer Science

Topic:	The CYK-Algorithm, a CLI-Tool and different ways of filling the Parsing Table
Author:	Andreas Braun <www.github.com/AndreasBraun5> Matrikel-Nr. 1200197
Version date:	March 3, 2017
1. Supervisor:	Prof. Dr. Wim Martens
2. Supervisor:	Tina Trautner

To my parents.

Abstract

The abstract of this thesis will be found here.

Zusammenfassung

Hier steht die Zusammenfassung dieser Bachelorarbeit.

Contents

1 Introduction

Let there be a grammar $G = (V,\ \Sigma,\ S,\ P)$ in chomsky normal form (CNF).
V is a finite set of variables.
Σ is an alphabet.
S is the starting symbol and $S \in V$.
P is a finite set of rules: $P \subseteq V \times (V \cup \Sigma)^*$. G is in CNF and therefore, more specifically, it holds: $P\ \subseteq\ V \times (V^2 \cup \Sigma)$.

For simplification the default definitions hold:

- $V = \{A, B, ...\}$
- $(V^2 \cup\ \Sigma)^* = \{a, b, ...\} \cup \{AB, BS, AC, ...\}$

Let there be a word $w \in \Sigma^*$, a language $L(G)$ and a grammar G in CNF.

1.1 Forward Problem vs. Backward Problem

Forward Problem ($G \xrightarrow{derivation} w$):
Informal definition: "Forming a derivation from a root node to a final sentence." [Duda 8.6.3 page 426]
Input: Grammar G in CNF.
Output: Derivation d that shows implicitly $w \subseteq L$.

Backward Problem = Parsing ($w \overset{?}{\subseteq} L(G)$):
Informal definition: "Given a particular w, find a derivation in G that leads to w. This process, called parsing, is virtually always much more difficult than forming a derivation." [Duda 8.6.3 page 426]
Input: w and a grammar G in CNF.
Output: $w \subseteq L(G) \Longrightarrow$ derivation d.

1.2 Parsing: Bottom-Up vs Top-Down

Bottom-Up: Bottom-Up parsing is "the general method used in the Cocke-Younger-Kasami(CYK) algorithm, which fills a parse table from the "bottom up"." (Bottom up means starting from the leaves.) [Duda 8.6.3 page 426]

Top-Down: "Top-Down parsing starts with the root node and successively applies productions from P, with the goal of finding a derivation of the test sentence w. Because it is rare indeed that the sentence is derived in the first production attempted,

it is necessary to specify some criteria to guide the choice of which rewrite rule to apply. Such criteria could include beginning the parse at the first (left) character in the sentence (i.e., finding a small set of rewrite rules that yield the first character), then iteratively expanding the production to derive subsequent characters, or instead starting at the last (right) character in the sentence." [Duda 8.6.3 page 428]

1.3 Scope of this thesis

The starting point of this thesis was to get a command line interface (CLI) tool to automatically generate $exercises = (grammar,\ word,\ parse\ table,\ derivation\ tree)$, which are used to test if the students have understood the way of working of the CYK algorithm. A scoring model is used to evaluate the generated exercises regarding their usability in an exam.

This alone doesn't meet the requirements for being an adequate topic for a bachelor thesis. Therefore the task of finding a clever algorithm to get exercises with a high chance of being usable as an exam exercise was added.

2 Simple Scoring Model

Short preface to the rationale about the scoring model.

2.1 Elimination Criteria and Selection Criteria

Success rates: Producibility: $w \subseteq L(G)$
Grammar restricitons: $n = |w|$, maxNumberOfVarsPerCell; **Delete SuccessRates-GrammarRestrictions class. Move maxNumberOfVarsPerCell to exam restrictions class and use n only as parameter.**
Exam restrictions: rightCellCombinationsForcedCount, maxSumOfProductions, maxSumOfVarsInPyramid

Picture of used scoring model without weights here. Maybe one picture together with the next subsection.

2.2 Weighting of the criteria

Picture of the final used scoring model with weights here.

2.3 Direct Ranking vs. Preference Analysis vs. ...

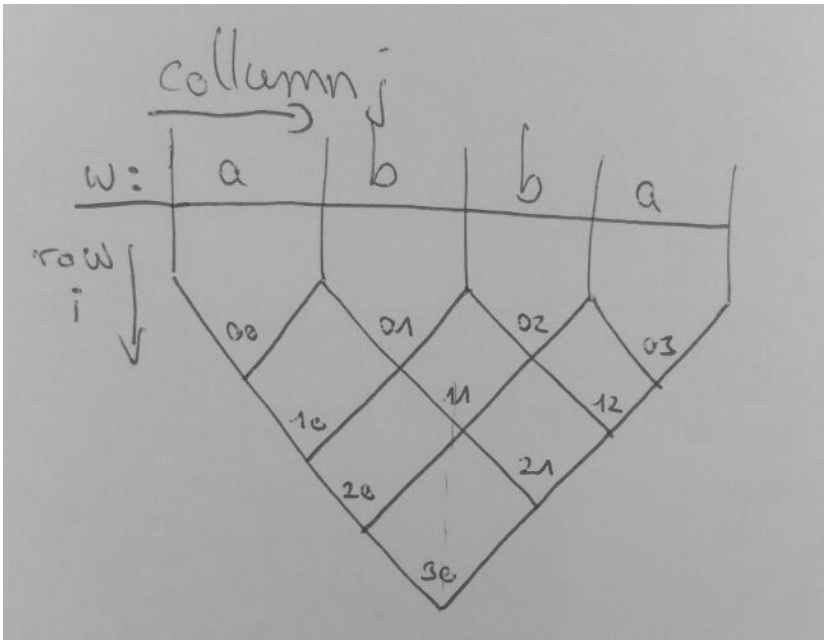
What method is used to compare the results out of the scoring model.
Direct Ranking is the simplest way.

3 Algorithms

3.1 Help Data Structure Pyramid and Others

Define $[i,\ j]$:
$[i,\ j] := \{i,\ i+1, ..., j-1,\ j\} \subseteq \mathbb{N}_{\geq 0}$.

Define *Pyramid*:
$Pyramid := \{cell_{i,j} \mid i \in \mathbb{N}_{\geq 0},\ j \in [0,\ j_{max} - i],\ i_{max} = j_{max} = |word| - 1\}$.
$cell_{i,j} = \{c \mid c \subseteq V\}$.
$EmptyPyramid \Leftrightarrow \forall i\ \forall j\ cell_{i,j} = \emptyset$.
Regarding one $cell_{i,j}$: $cell_{i,j} = cellDown$, $cell_{i-1,j} = cellUpperLeft$ and $cell_{i-1,j+1} = cellUpperRight$



3.2 Exam Exercise Generating Algorithms

3.2.1 Algorithm: AlgorithmName

Things like the $G = (V, \Sigma, S, P)$ can be assumed as known.
$P = P \cup \{distribute\ \{\sigma \mid \sigma \in w\}\ uniform\ randomly\ over\ \{v \mid v \in V\}\}$ which equals the $distributeRhse$ module.
Bias is only allowed top vs down regarding the pyramid. No left or right bias intended yet.
(A), (B), ... represent exchangeable algorithm modules.

3.2.1.1 Basic Idea

3.2.1.2 Tweak Idea 1 for Algorithm

3.2.1.3 Tweak Idea 2 for Algorithm

3.2.1.4 Finished Algorithm

3.2.2 Algorithm: GeneratorGrammarDiceRollOnly

Very naive way of generating grammars. This is intended to be the starting point for our algorithms we find. Each found algorithm must have a higher score than this algorithm or otherwise it would be worse than simple dice rolling and then removing the useless productions.

Algorithm 1: GeneratorGrammarDiceRollOnly

Input: $Word\ w \in \Sigma^*$

Output: $P \subseteq V \times (V^2 \cup \Sigma)$

1 $P = \{distribute\ \sigma \in \Sigma\ over\ V\}$ (A) ;

2 $P = P \cup \{distribute\ vc \in V^2\ over\ V\}$ (B) ;

3 $P = P \setminus \{p \mid p \subseteq P,\ vc\ is\ right\ in\ p,\ \forall i\ \forall j\ vc \notin cell_{i,j}\ of\ the\ pyramid\}$;

4 **return** P;

Line 3: Removes all useless production.

Does the output $P \subseteq V \times (V^2 \cup \Sigma)$ imply that G is in CNF? CNF does only have useful variables [TI script Def. 8.3 page 210] vs. $P \subseteq V \times (V^2 \cup \Sigma)$

Algorithm 2: GeneratorGrammarDiceRollOnlyBias

Input: $Word\ w \in \Sigma^*,\ P \subseteq V \times (V^2 \cup \Sigma) = \emptyset$,

Output: $Grammar\ G\ in\ CNF$

1 NOT FINISHED, MAYBE LATER.

2 $pick\ uniform\ randomly\ \{v \mid v \in V\} with...$;

3 ...;

4 $P = P \cup \{distribute\ \{\sigma \mid \sigma \in w\}\ over\ \{v \mid v \in V\}\}$;

5 $P = P \cup \{distribute\ \{vc \mid vc \in V^2\}\ over\ \{v \mid v \in V\}\}$;

6 $P = P \setminus \{p \mid p \subseteq P,\ vc\ is\ right\ in\ p,\ \forall i\ \forall j\ vc \notin cell_{i,j}\ of\ the\ pyramid\}$;

7 **return** G;

3.2.3 Algorithm: BottomUp GeneratorGrammarDiceRollMartens

Algorithm 3: GeneratorGrammarDiceRollMartens

Input: $Word\ w \in \Sigma^*$

Output: $P \subseteq V \times (V^2 \cup \Sigma)$

1 $P = \{distribute\ \sigma \in \Sigma\ over\ V\}$ (A) ;
2 $pyramid = CYK(G,\ w)$;
3 **for** $i := 1$ ***to*** i_{max} **do**
4 $J \subseteq \mathbb{N}$;
5 $cellSet \subseteq V^2$;
6 **while** $|J| < j_{max} - i$ **do**
7 $choose\ one\ j \notin J\ uniform\ randomly\ in\ [0, j_{max} - i]$;
8 $J = J \cup j$;
9 $cellSet = calculateSubsetForCell(pyramid,\ i,\ j)$;
10 $P = P \cup \{distribute\ vc \in cellSet\ over\ V\}$ (B) ;
11 $pyramid = CYK(G,\ w)$;
12 $evaluate\ stopping\ criteria\ regarding\ the\ pyramid$ (C) ;
13 **if** $stopping\ criteria = true$ **then**
14 **return** P;
15 **end**
16 **end**
17 **end**
18 **return** P;

Line 2: Fills the i=0 row of the pyramid.
Line 7: Instead of going from left to right, choose j uniform randomly with the restrictions that one cell is only visited one time.
Note: Maybe modify algorithm to also work with the threshold.

Algorithm 4: GeneratorGrammarDiceRollMartens2

Input: $Word\ w \in \Sigma^*$

Output: $P \subseteq V \times (V^2 \cup \Sigma)$

```
1  P = {distribute σ ∈ Σ over V} (A) ;
2  pyramid = CYK(G, word) ;
3  for i := 1 to i_max do
4      for j := 0 to j_max − i do
5          rowSet = rowSet ∪ {(XY, i) | X, Y ∈ V,
           XY ∈ calculateSubsetForCell(Pyramid, i, j)};
6      end
7      while threshold_i = false do
8          choose one vc ∈ rowSet with priority, depending on i,
           uniform randomly (D) ;
9          P = P ∪ {distribute vc ∈ rowSet over V} (B) ;
10         pyramid = CYK(G, w);
11         evaluate and update threshold_i, regarding line i;
12         evaluate stopping criteria, regarding the pyramid (C) ;
13         if stopping criteria = true then
14             return P;
15         end
16     end
17 end
18 return P;
```

Line 2: Fills the i=0 row of the pyramid.

Line 5: $(AB,1),(AB,2),(BC,3)... \in sub \rightarrow$ multiple occurrences of AB are allowed. This considers "more important" compound variables.

Line 8: One vc can be chosen several times.

Note Line 8: Priority mechanism: In line $i+1$ the $k = \{(A,l) \mid (A,l) \in sub,\ l = i\}$ are preferred over the
$m = \{(A,n) \mid (A,n) \in sub,\ n < i\}$. In what way are they preferred? Using some kind of factor to weight the i of (A,i).

Note Line 11: Threshold, Linear or log function $f(i)$?

3.2.4 Algorithm: Idea 1, TopDown From node to leaves

Algorithm 5: Idea1

Input: $Word\ w \in \Sigma^*,\ i, j \in \mathbb{N},\ \forall j\ cell_{0,j} \neq \emptyset$

Output: $P \subseteq V \times (V^2 \cup \Sigma$

1 **if** $i = 1$ **then**

2 **return** $cell_{i,j}$;

3 **end**

4 *choose one m uniform randomly in* $[m_{min},\ m_{max}] \longrightarrow cellLeft\ and\ cellRight$;

5 $A = Idea1(w, P, cellLeft)$;

6 $B = Idea1(w, P, cellRight)$;

7 $VC = uniform\ random\ subset\ of\ size\ [min,\ max]\ from\ \{vc \mid v \in A \wedge c \in B\}$;

8 $P = P \cup \{distribute\ vc \in VC\ over\ V\}\}$ (A) ;

9 **return** G;

Line 4: If one would be at $cell_{2,2}$ you choose from $[2, 1]$, which isn't allowed.

Line 5 $\longrightarrow$ *new cells :* $cell_{m,j}$ *and* $cell_{i-m,m+j+1}$ *See Alg. subSetCalc with* $k \in [i-1; 0]$ $[Y \in V_{k,j},\ Z \in V_{i-k-1,k+j+1}]$

3.2.5 Algorithm: Idea 2, How often cells are used for subset calculations

3.2.6 Tweaking Sub Procedures in more detail

Maybe don't keep this so that the Algorithms can be read without flipping pages.

Algorithm 6: distributeRhse2

Input: $Rhse \subseteq (V^2 \cup \Sigma),\ i \in \mathbb{N},\ j \in \mathbb{N}$

Output: *Grammar G in CNF with uniform randomly distributed Rhse's.*

1 *choose n uniform randomly in* $[i, j]$;
2 *choose* $V_{add} := uniform\ random\ subset\ of\ size\ n\ from\ V$;
3 $P = P \cup \{\text{"}v \longrightarrow rhse\text{"} \mid v \in V_{add},\ rhse \in Rhse\}$;
4 **return** G;

Algorithm 6 isn't needed anymore for the descriptions of the basic idea of the algorithm. It will be a module later on while tweaking the algorithms.

Algorithm 7: calculateSubsetForCell

Input: $cell_{i,j} \in pyramid$

Output: $V_{i,j} \subseteq V^2$

1 $V_{i,j} = \emptyset$;
2 **for** $k := i - 1 \to 0$ **do**
3 $\quad V_{i,j} = V_{i,j} \cup \{X \mid X \longrightarrow YZ,\ Y \in V_{k,j},\ Z \in V_{i-k-1,k+j+1}\}$;
4 **end**
5 **return** $V_{i,j}$;

Algorithm 7 describes the magic of the CKY-algorithm. It shows what cells are taken into account while filling one cell of the parse table.

3.3 Criteria Checking Procedures

Description of the checks here.
All test of the GrammarValidityChecker class are based on the simple setV matrix.

isValid = isWordProducible && isExamConstraints && isGrammarRestrictions

isWordProducible = CYK.algorithmAdvanced()

isExamConstraints = isRightCellCombinationsForced && isMaxSumOfProductionsCount && isMaxSumOfVarsInPyramidCount && countRightCellCombinationsForced

isGrammarRestrictions = isSizeOfWordCount && isMaxNumberOfVarsPerCellCount

Algorithm 8: checkForceCombinationPerCell

Input: $cell_{i,j} \subseteq V,\ cell_{i-1,j} \subseteq V,\ cell_{i-1,j+1} \subseteq V,\ P \subseteq V \times (V^2 \cup \Sigma)$
Output: $varsForcing \subseteq V$

$varsForcing \subseteq V$;
$varComp = \{XY \mid X \in cell_{i-1,j}\ \wedge\ Y \in cell_{i-1,j+1}\}$;
foreach $v \in cell_{i,j}$ **do**
  $prods = \{p \mid p \subseteq P,\ v\ is\ left\ in\ p\}$;
  $rhses = \{rhse \mid rhse\ is\ right\ in\ p \in prods\}$;
  if $varComp \nsubseteq rhses$ **then**
    $varsForcing = varsForcing \cup v$;
  end
end
return $varsForcing$;

Input: $cell_{i,j} = cellDown$, $cell_{i-1,j} = cellUpperLeft$ and $cell_{i-1,j+1} = cellUpperRight$

Algorithm 8 is a check that needs to be explained.

Algorithm 9: checksumOfProductions

Input: $max \in \mathbb{N}_{\geq 0}$
Output: $true \iff sum \leq max$

if $|P| > max$ **then**
  return $fales$;
end
return $true$;

Algorithm 9 can be explained via the Output of the algorithm alone.

Algorithm 10: checkMaxNumberOfVarsPerCell

Input: $max \in \mathbb{N}_{\geq 0}$

Output: $true \iff \forall cell_{i,j} \in pyramid,\ |cell_{i,j}| \leq max$

```
1 for i := 1 to i_max do
2     for j := 0 to j_max − i do
3         if |cell_{i,j}| > max then
4             return false;
5         end
6     end
7 end
8 return true;
```

Algorithm 10 can be explained via the Output of the algorithm alone.

Algorithm 11: checkMaxSumOfVarsInPyramid

Input: $max \in \mathbb{N}_{\geq 0}$

Output: $true \iff sum \leq max$

```
1  sum = 0;
2  for i := 1 to i_max do
3      for j := 0 to j_max − i do
4          sum = sum + |cell_{i,j}|;
5          if sum > max then
6              return false;
7          end
8      end
9  end
10 return true;
```

Algorithm 11 could possible be explained via a simple mathematical statement like the algorithms 9 and 10.

4 CLI Tool

4.1 Short Requirements Specification

Use Cases i $\longrightarrow$ "Lastenheft".
Input and Output parameter identification.
Here is described what the finished tool must and can do.

4.1.1 Exam Exercises

An exam exercise consists out of a grammar, a word, a parsing table and a derivation tree. Creating a exam exercises must be possible. Therefore it is needed:

- Selection of a possible exam exercise out of high scoring samples $\longrightarrow$ calculateSamples.jar [Input parameter: countOfNewSamples (better scoring samples in exchange for longer computation time)], which upon execution fills samples.txt with new high scoring samples, together with its actual scoring model parameters. Out of this samples one can be selected manually that is used for an exam exercise.

- Modifying of a exam exercise candidate: Changing the grammar and changing the word. [?changing the pyramid (I think no, because of the strong interconnection between the grammar and the parsing table it is already covered through being able to change the grammar)?] $\longrightarrow$ calculateExamExercise.jar [Input parameter: examExercise.txt], that updates pre defined information for one sample upon execution.

- Predefined Information: It is a printable version of the finished exam exercise like grammar.png, parsingTable.png and derivationTable.png together with its latex code, that was used for its creation - modification later one possible. Also it is examExerciseInfo.txt, that has the information about its actual scoring model parameters.

4.1.2 Fun With CNF's and CYK

Trying out stuff freestyle:

- Se

-

4.2 Overview - UML

UML-Diagramm showing the general idea of the implementation.
List noteworthy used libraries here, too.
Maybe some information out of the statistics tool of IntelliJ.

4.2.1 UML: More Detail 1

4.2.2 UML: More Detail 2

4.3 User Interaction

Here the specific must can do's are explained with short examples.

4.3.1 Use Case 1

4.3.2 Use Case i

5 Notes

The informal goal is to find a suitable combination of a grammar and a word that meets the demands of an exam exercise. Also the CYK pyramid and one derivation tree of the word must be generated automatically as a "solution picture".
Firstly the exam exercise must have an upper limit of variables per cell while computing the CYK-pyramid.
Secondly the exam exercise must have one or more "special properties" so that it can be checked if the students have clearly understood the algorithm, e.g. "Excluding the possibility of luck."

The more formal goal is identify and determine parameters that in general can be used to define the properties of a grammar, so that the demanded restrictions are met. Also parameters could be identified for words, but "which is less likely to contribute, than the parameters of the grammar." [Second appointment with Martens]

Some introductory stuff:

Possible basic approaches for getting these parameters are the Rejection Sampling method and the "Tina+Wim" method.
Also backtracking plays some role, but right now I don't know where to put it. Backtracking is underapproach to Rejection Sampling.
Note: Starting with one half of a word and one half of a grammar.

Identify restrictions (=parameters) regarding the grammar.
Maybe find restrictions regarding the words, too.

Procedures for automated generation. Each generation procedure considers different restrictions and restriction combinations. The restrictions within one generation procedure can be optimized on its own. Up till now:
Generating grammars: DiceRolling, ...
Generating words: DiceRolling, ...

Parameter optimisation via theoretical and/or benchmarking approach.
Benchmarking = generate N grammars and test them, (N=100000).
Define a success rate and try to increase it.

The overall strategy is as following:

1.) Identify theoretically a restriction/parameter for the grammar. Think about the influence it will have. Think also about correlations between the restrictions.
2.) Validate the theoretical conclusion with the benchmark. Test out the influence of this parameter upon the success rate. Try different parameter settings.

The ordering of step 1 and step 2 can be changed.

Used software

Github
IntelliJ IDEA

Algorithms

This section contains all algorithms referenced in this thesis.

References

[1] JSR 220: Enterprise Java Beans 3.0 https://jcp.org/en/jsr/detail?id=220, 09/09/2015