NLP with Scala

Christoph Lüth and Martin Ring

28.09.2018

Agenda

- Introduction to NLP
- Introduction to Scala
- Tokenization and Splitting
- Word-level Analysis
- Named Entity Recognition
- Grammar Tree Analysis
- Dependency Analysis

NLP: Short History

- Early Resarch: Rule Based Parsing
- 1980s - 1990s: Statistical Revolution
- Modern Days: Deep Learning and Statistics

Word Databases

- e.g. WordNet

S: (n) head, **chief**, top dog (a person who is in charge) *"the head of the whole operation"*
- *direct hyponym* / *full hyponym*
- *direct hypernym* / ***inherited hypernym*** / *sister term*
  - S: (n) leader (a person who rules or guides or inspires others)
    - S: (n) person, individual, someone, somebody, mortal, soul (a human being) *"there was too much for one person to do"*
      - S: (n) organism, being (a living thing that has (or can develop) the ability to act or function independently)
        - S: (n) living thing, animate thing (a living (or once living) entity)
          - S: (n) whole, unit (an assemblage of parts that is regarded as a single entity) *"how big is that part compared to the whole?"; "the team is a unit"*
            - S: (n) object, physical object (a tangible and visible entity; an entity that can cast a shadow) *"it was full of rackets, balls and other objects"*
              - S: (n) physical entity (an entity that has physical existence)
                - S: (n) entity (that which is perceived or known or inferred to have its own distinct existence (living or

NLP: Syntactical Concepts

- Sentence Splitting / Tokenization
- Lemmatization / Stemming
- Morphological Segmentation
- POS Tagging
- Parse Tree Generation

Splitting / Tokenization

- Word and sentence boundary recognition
- Often relatively simple:
  - Word Boundaries: `[^\w,]+`
  - Sentence Boundaries: `[;.?!]+`

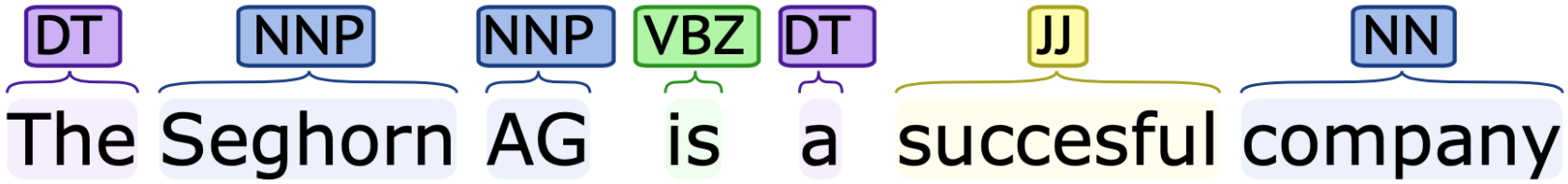
Lemmatization / Stemming

- **Stemming** - Reduce words to their stem:
  - e.g. relational $\rightarrow$ relate, tanned $\rightarrow$ tan
  - Porter Stemming Algorithm:
    - `(m>0)EED` $\rightarrow$ `EE`
    - `(*v*)ED` $\rightarrow$ `()`
    - `(*v*)ING` $\rightarrow$ `()`
- **Lemmatization** - Reduce words into their uninflected form:
  - e.g. [went, goes] $\rightarrow$ go
  - e.g. People $\rightarrow$ Person
  - e.g. [are, is, been] $\rightarrow$s be

Morphological Segmentation

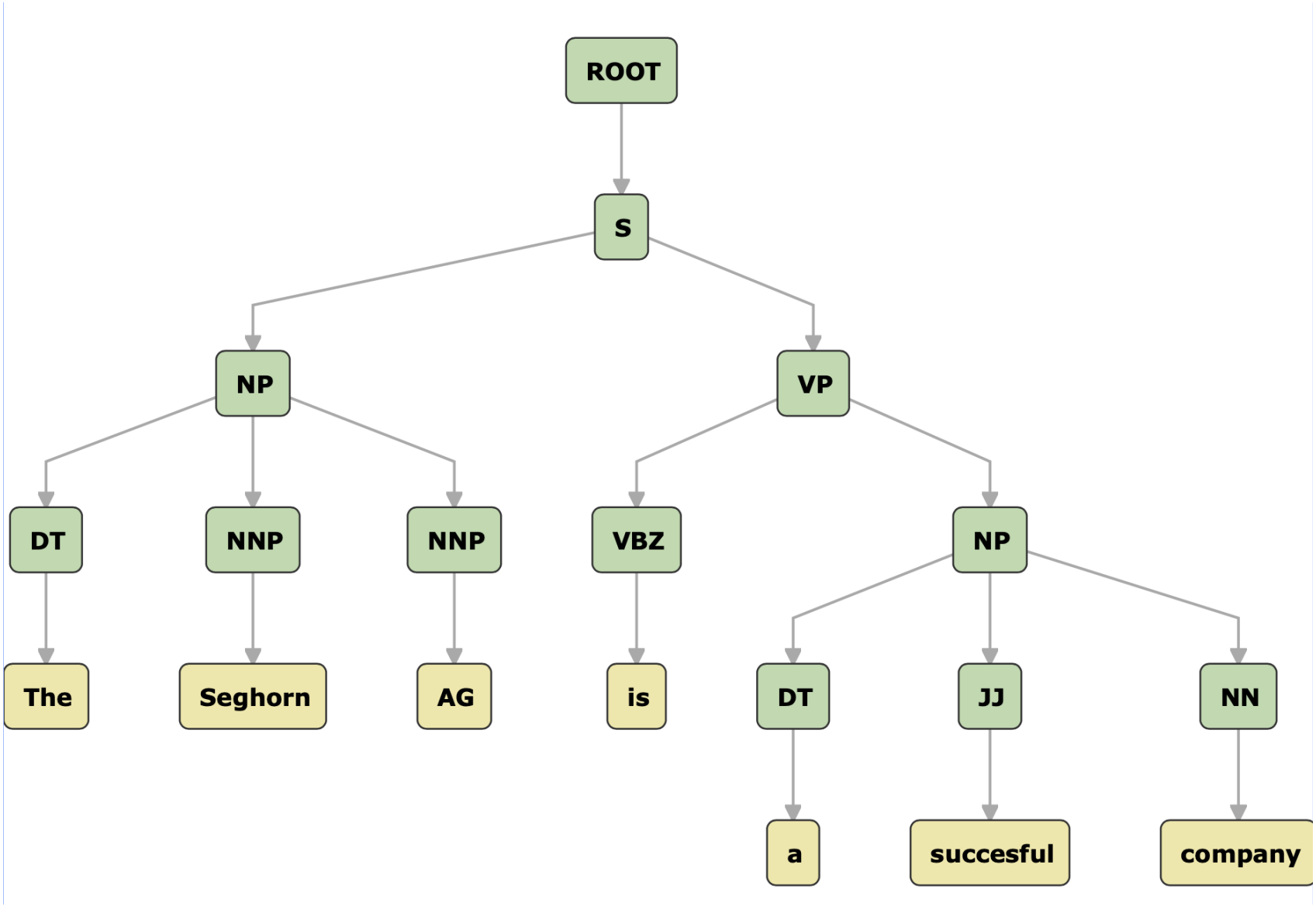
- In-word morpheme segmentation
- Importance depends on language
- English: irrelevant
- German: important

POS Tagging



DT	Determiner
NNP	Proper noun, singular
VBZ	Verb, 3rd person singular present
JJ	Adjective
NN	Noun, singular or mass

Parse Trees

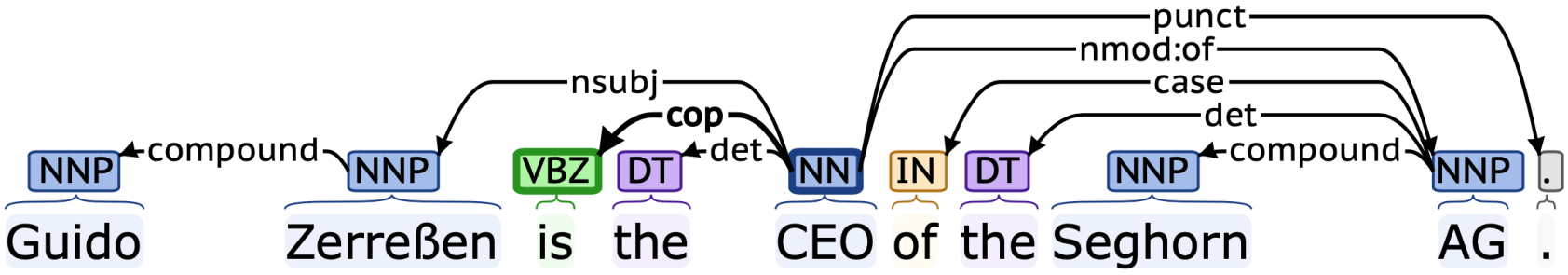

ROOT
S
NP
VP
DT
NNP
NNP
VBZ
NP
The
Seghorn
AG
is
DT
JJ
NN
a
succesful
company

NLP: Semantic Concepts

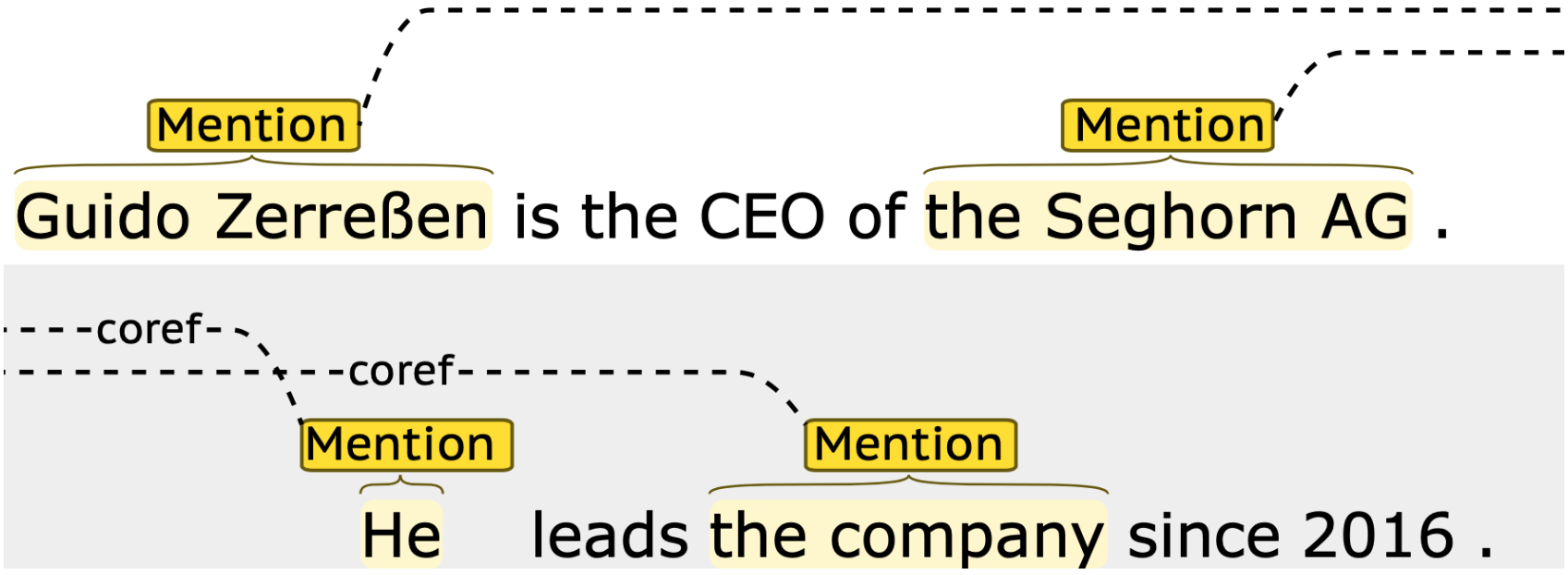
- Named Entity Recognition
- Dependency Graph
- Coreference
- Sentiment Analysis

Named Entity Recogition

Dependency Graphs



Coreferences



Scala

- A scalable language
- Rein objektorientiert
- Funktional
- Basiert auf der JVM
- Seit 2004 http://www.scala-lang.org
- Seit 2011 kommerziell

Scala am Beispiel (01)

- Veränderliche Variablen (var)
- Unveränderliche Variablen (val)
- while-Schleifen
- Rekursion
- Typinferenz

Jetzt Sie: schreiben Sie eine Funktion, welche eine Zeichenkette n-mal wiederholt:

- Rekursiv
- Iterativ

Scala am Beispiel (02)

- Klassenparameter
- Klassenvorbedingungen
- private Werte und Methoden
- `override` nicht optional
- Overloading
- Operatoren
- Companion objects

Jetzt Sie: erweitern Sie die Klasse 'Rational'

- so dass ganze Zahlen ohne den Nenner 1 ausgegeben werden, und
- um Multiplikation und Division (`*` und `/`)
- um Gleichheit (Methode `equals`)

Scala am Beispiel (03)

- case classes:
  - Factory-Methode für Konstruktoren
  - Parameter als `val`
  - abgeleitete Implementierungen
  - pattern matching
- `sealed` verhindert Erweiterung
- *Algebraische Datentypen*

Jetzt Sie:

- Erweitern Sie die Klasse `Expr` um eine Funktion, welche einen Ausdruck **schön** ausgibt (Bonuspunkte für Berücksichtigung der Operatorpräzendenz)
- Ersetzen Sie das Argument von `BinOp` und `UnOp` durch einen geeigneten Aufzählungstyp

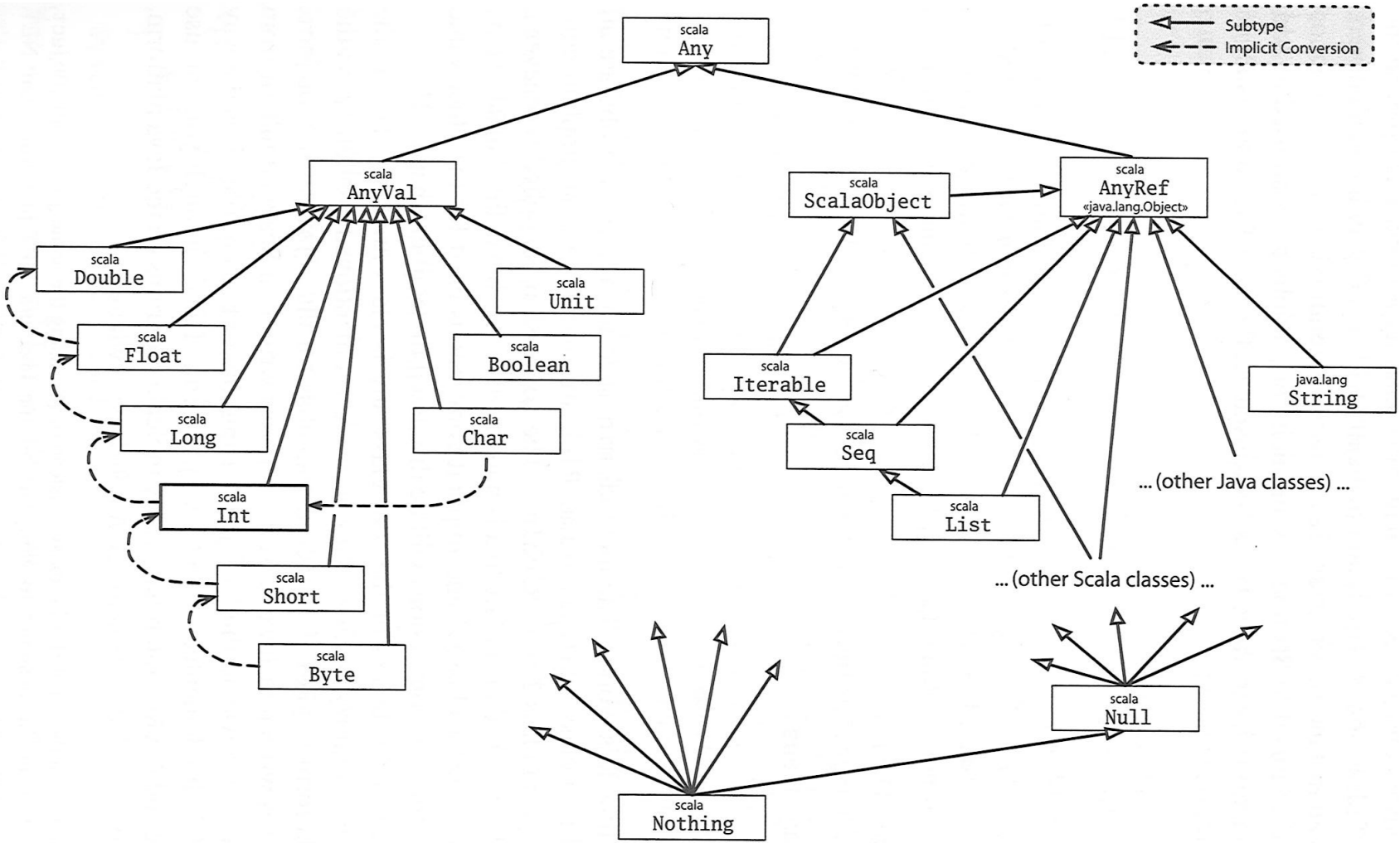
Nützliche Typen:

- Listen
- Tupel
- Maps und Mengen
- Arrays
- Options

Typsystem

- Das Typsystem behebt mehrere Probleme von Java:
  - Werte vs. Objekte
  - Scala vs. Java
  - `NULL` Referenzen
- Parametrische Polymorphie (Generics)
- Funktionen als Objekte
- Typvarianz (ask us later)

Typhierarchie



Traits

- Abstrakte Klassen ohne Oberklasse und Konstruktor
- Mehrfachvererbung von traits möglich
- Nützlich zur Strukturierung (Aspektorientierung)

Jetzt Sie: Definieren Sie eine Funktion, welche die Lösung des n-Damen-Problems ansprechend darstellt.

Mehr Scala:

- Nebenläufigkeit und reaktive Prorgammierung mit Aktoren (acca)
- `scala.js`: Übersetzer Scala $\rightarrow$ Javascript
- `sbt`: simple build tool