NICHT JEDER BLICK IST NAH. KEIN DORF IST SPÄT.
EIN SCHLOSS IST FREI UND JEDER BAUER IST FERN.
JEDER FREMDE IST FERN. EIN TAG IST SPÄT.
JEDES HAUS IST DUNKEL. EIN AUGE IST TIEF.
NICHT JEDES SCHLOSS IST ALT. JEDER TAG IST ALT.
NICHT JEDER GAST IST WÜTEND. EINE KIRCHE IST SCHMAL.
KEIN HAUS IST OFFEN UND NICHT JEDE KIRCHE IST STILL.
NICHT JEDES AUGE IST WÜTEND. KEIN BLICK IST NEU.
JEDER WEG IST NAH. NICHT JEDES SCHLOSS IST LEISE.
KEIN TISCH IST SCHMAL UND JEDER TURM IST NEU.
JEDER BAUER IST FREI. JEDER BAUER IST NAH.
KEIN WEG IST GUT ODER NICHT JEDER GRAF IST OFFEN.
NICHT JEDER TAG IST GROSS. JEDES HAUS IST STILL.
EIN WEG IST GUT. NICHT JEDER GRAF IST DUNKEL.
JEDER FREMDE IST FREI. JEDES DORF IST NEU.
JEDES SCHLOSS IST FREI. NICHT JEDER BAUER IST GROSS.
NICHT JEDER TURM IST GROSS ODER NICHT JEDER BLICK IST FREI.
EINE KIRCHE IST STARK ODER NICHT JEDES DORF IST FERN.
JEDER FREMDE IST NAH, SO GILT KEIN FREMDER IST ALT.
EIN HAUS IST OFFEN. KEIN WEG IST OFFEN.

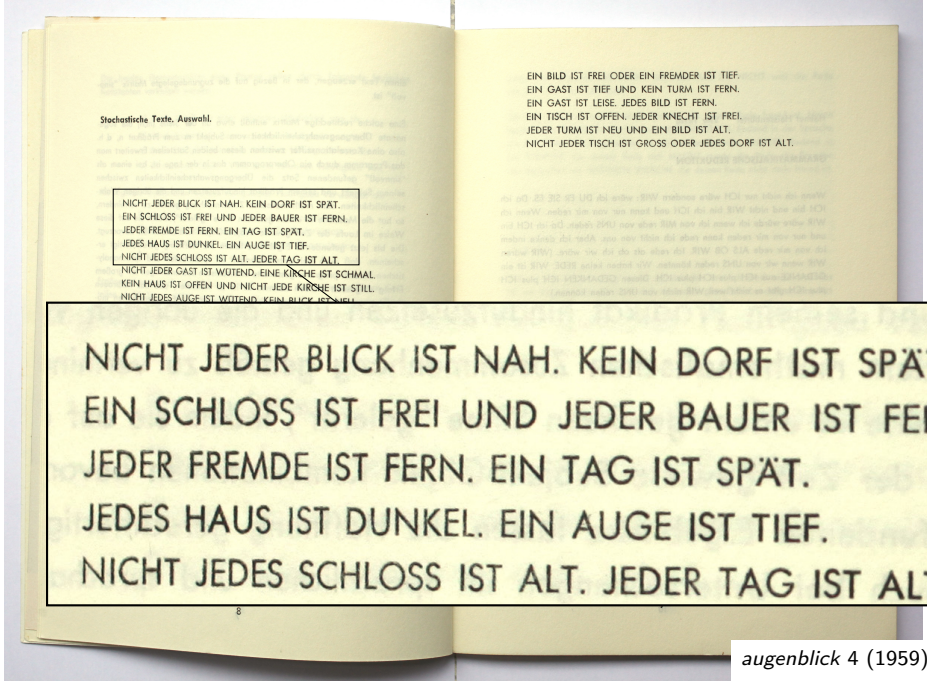
# Generierung von Sprache 1959 und heute: Alte und neue Sprachmodelle

## Einführung in die Informationsverarbeitung

Nils Reiter
nils.reiter@uni-koeln.de

December 22, 2022

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN


Stochastische Texte. Auswahl.

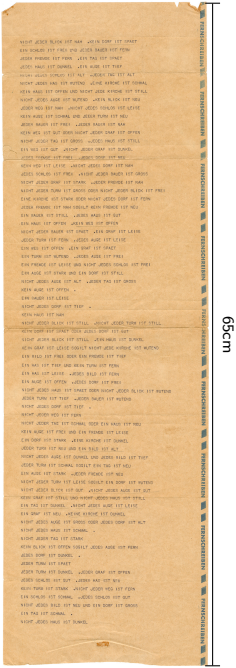
NICHT JEDER BLICK IST NAH. KEIN DORF IST SPÄT.
EIN SCHLOSS IST FREI UND JEDER BAUER IST FERN.
JEDER FREMDE IST FERN. EIN TAG IST SPÄT.
JEDES HAUS IST DUNKEL. EIN AUGE IST TIEF.
NICHT JEDES SCHLOSS IST ALT. JEDER TAG IST ALT.
NICHT JEDER GAST IST WÜTEND. EINE KIRCHE IST SCHMAL.
KEIN HAUS IST OFFEN UND NICHT JEDE KIRCHE IST STILL.
NICHT JEDES AUGE IST WITTEND. KEIN BLICK IST [illegible]

EIN BILD IST FREI ODER EIN FREMDER IST TIEF.
EIN GAST IST TIEF UND KEIN TURM IST FERN.
EIN GAST IST LEISE. JEDES BILD IST FERN.
EIN TISCH IST OFFEN. JEDER KNECHT IST FREI.
JEDER TURM IST NEU UND EIN BILD IST ALT.
NICHT JEDER TISCH IST GROSS ODER JEDES DORF IST ALT.

NICHT JEDER BLICK IST NAH. KEIN DORF IST SPÄT.
EIN SCHLOSS IST FREI UND JEDER BAUER IST FERN.
JEDER FREMDE IST FERN. EIN TAG IST SPÄT.
JEDES HAUS IST DUNKEL. EIN AUGE IST TIEF.
NICHT JEDES SCHLOSS IST ALT. JEDER TAG IST ALT.

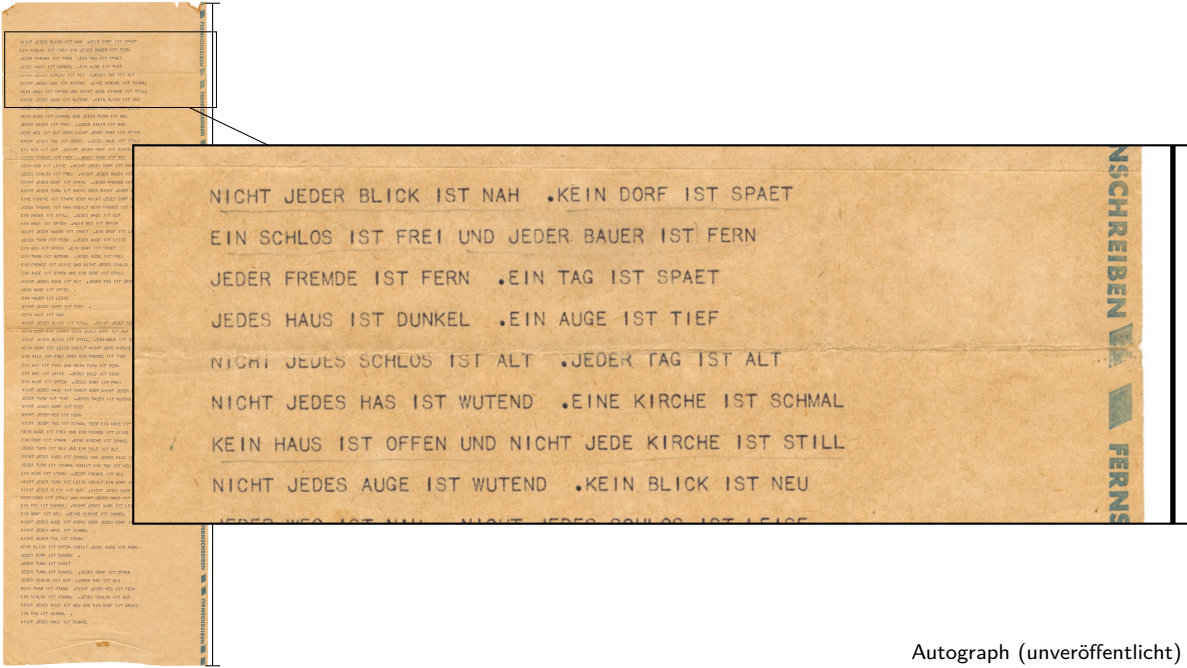
8

*augenblick* 4 (1959), H. 1, S. 8–9



65cm

Autograph (unveröffentlicht)




Autograph (unveröffentlicht)

# Die Autoren

Abbildung: Theo Lutz

Abbildung: Zuse Z22

# Theo Lutz

- *1932, Tübingen
- Ab 1953: Mathematik an Technischer Hochschule Stuttgart
- Besucht Lehrveranstaltungen bei Max Bense (‚Stuttgarter Schule')
- Diplomarbeit 1959, Promotion 1976 (bei Rul Gunzenhäuser)
- †2010, Esslingen
  - Nachlass seit 2019 im Deutschen Literaturarchiv, Marbach am Neckar

Toni Bernhart (2020). „Beiwerk als Werk. *Stochastische Texte* von Theo Lutz". In: *editio* 34. DOI: 10.1515/editio-2020-0010

# Zuse Z22

- Siebtes Computer-Modell entwickelt von Konrad Zuse bis 1957
  - Leitender Ingenieur: Lorenz Hanewinkel
  - Letztes funktionsfähiges Exemplar: Zentrum für Kunst und Medien (ZKM), Karlsruhe
- Neupreis: ca. $250\,000$ DM
- Ab 1960: Zweite Version Z22/R
  - Änderungen, „die die Rechengeschwindigkeit der Anlage erhöhen und die Programmierung an einigen Stellen vereinfachen“ (*Z22/R* 1960)

Z22 im ZKM Karlsruhe, Foto: Lemming81/Wikipedia

Z22 im Deutschen Museum München, Foto: Deutsches Museum

Produktfoto Zuse Z 22

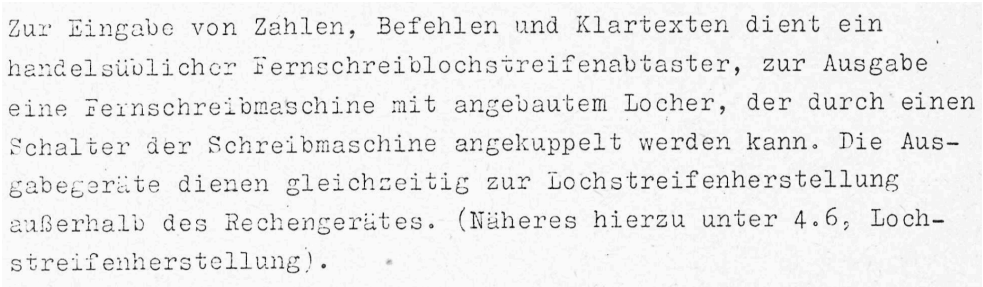
# Aufbau der Maschine

- Von-Neumann-Architektur: Daten und Befehle im gleichen Speicher
- Speicher: 38-stellige Binärzahlen (= 1 ‚Wort')
  - Trommelspeicher: 8192 Speicherzellen für je 1 Wort (= 38.9 kB)
    - Mittlere Zugriffszeit: 5 msec (*Programmierungsanleitung* 1960, 2)
  - Schnellspeicher: 14 Zellen (Z22R: 25) „Ohne Zugriffszeit" (*Programmierungsanleitung* 1960, 2)
    - Adressen 0–31: Schnellspeicheradressen; 4: Akkumulator

# Aufbau der Maschine

- Von-Neumann-Architektur: Daten und Befehle im gleichen Speicher
- Speicher: 38-stellige Binärzahlen (= 1 ‚Wort')
  - Trommelspeicher: 8192 Speicherzellen für je 1 Wort (= 38.9 kB)
    - Mittlere Zugriffszeit: 5 msec (*Programmierungsanleitung* 1960, 2)
  - Schnellspeicher: 14 Zellen (Z22R: 25) „Ohne Zugriffszeit" (*Programmierungsanleitung* 1960, 2)
    - Adressen 0–31: Schnellspeicheradressen; 4: Akkumulator
- Rechen- und Leitwerk
  - Rechenoperationen: Addition, Subtraktion, logische Intersektion, Verschiebungen
  - Optional: Verwendung von zwei Zellen für eine Zahl (doppelte Länge!)

# Aufbau der Maschine



Zur Eingabe von Zahlen, Befehlen und Klartexten dient ein handelsüblicher Fernschreiblochstreifenabtaster, zur Ausgabe eine Fernschreibmaschine mit angebautem Locher, der durch einen Schalter der Schreibmaschine angekuppelt werden kann. Die Ausgabegeräte dienen gleichzeitig zur Lochstreifenherstellung außerhalb des Rechengerätes. (Näheres hierzu unter 4.6, Lochstreifenherstellung).

Abbildung: Ein- und Ausgabe bei einer Z22 (*Programmierungsanleitung* 1960, 3)

# Z22 Programmieren

- Was es nicht gibt: Variablen, Funktionen, Klassen/Objekte, Datentypen

# Z22 Programmieren

- Was es nicht gibt: Variablen, Funktionen, Klassen/Objekte, Datentypen
- Speicheradressen sind durchnummeriert
- In Speicheradressen stehen Daten und Befehle

# Z22 Programmieren

- Was es nicht gibt: Variablen, Funktionen, Klassen/Objekte, Datentypen
- Speicheradressen sind durchnummeriert
- In Speicheradressen stehen Daten und Befehle
- Default: Wenn Du Befehl in Zelle $X$ abgearbeitet hast, mache bei $X + 1$ weiter
  - Ausnahme Sprung: Nimm als nächstes den Befehl aus Speicherzelle $Y$ (danach $Y + 1$ etc.)

# Z22 Programmieren

- Was es nicht gibt: Variablen, Funktionen, Klassen/Objekte, Datentypen
- Speicheradressen sind durchnummeriert
- In Speicheradressen stehen Daten und Befehle
- Default: Wenn Du Befehl in Zelle $X$ abgearbeitet hast, mache bei $X + 1$ weiter
  - Ausnahme Sprung: Nimm als nächstes den Befehl aus Speicherzelle $Y$ (danach $Y + 1$ etc.)
- Befehle: Mach etwas mit Speicherzelle $X$

# Programmieren

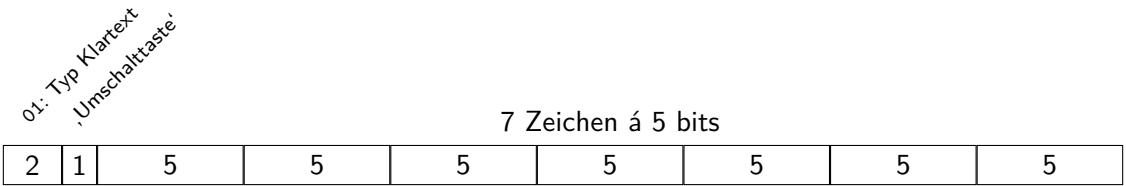
## Daten

- 38-bit-Zellen („Worte“)
  - Zahlen: 1 bit Vorzeichen, Rest Ganzzahlwert $-2^{37} < n < 2^{37} - 1$
    - Negativzahlen durch 2er-Komplement
    - Gleitkommazahlen durch Software

# Programmieren

## Daten

- 38-bit-Zellen („Worte“)
  - Zahlen: 1 bit Vorzeichen, Rest Ganzzahlwert
    $-2^{37} < n < 2^{37} - 1$
    - Negativzahlen durch 2er-Komplement
    - Gleitkommazahlen durch Software
  - Klartext: „International Telegraph Alphabet No. 1“ (= „Baudot code“, 1870s)
    - 5 Bit pro Zeichen, eine Zelle kann also max. 7 Zeichen speichern
    - $2^5 = 32$ Zeichen, doppelt belegt durch ‚Umschalttaste‘



# Programmieren

## Daten

- 38-bit-Zellen („Worte“)
  - Zahlen: 1 bit Vorzeichen, Rest Ganzzahlwert
    $-2^{37} < n < 2^{37} - 1$
    - Negativzahlen durch 2er-Komplement
    - Gleitkommazahlen durch Software
  - Klartext: „International Telegraph Alphabet No. 1“ (= „Baudot code“, 1870s)
    - 5 Bit pro Zeichen, eine Zelle kann also max. 7 Zeichen speichern
    - $2^5 = 32$ Zeichen, doppelt belegt durch ‚Umschalttaste‘

01: Typ Klartext
‚Umschalttaste‘

7 Zeichen á 5 bits

| 2 | 1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
|---|---|---|---|---|---|---|---|---|
| 01 | 0 | 00101 | 01110 | 10100 | 10010 | 11000 | 00101 | 00100 |
| | | S | C | H | L | O | S | ␣ |

# Programmieren

## Befehle

| Typ | PP | P | QQ | Q | Y | C | N | LL | R | U | A | S | F | K | H | Z | G | V | Schnellspeicher | Trommelspeicher |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 21–25 | 26–38 |

Abbildung: Repräsentation von Befehlen in der Z22

# Programmieren

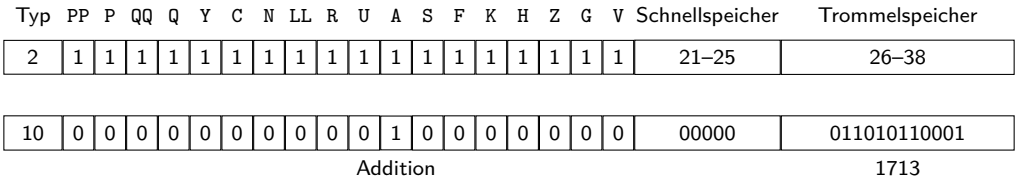
## Befehle

| Typ | PP | P | QQ | Q | Y | C | N | LL | R | U | A | S | F | K | H | Z | G | V | Schnellspeicher | Trommelspeicher |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 21–25 | 26–38 |
| 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 00000 | 011010110001 |
| | | | | | | | | | | | Addition | | | | | | | | | 1713 |

Abbildung: Repräsentation von Befehlen in der Z22

# Programmieren

## Befehle




= ›Addiere Inhalt von Speicherstelle 1713 und Akkumulator, schreibe das Ergebnis in Akkumulator‹

= ‚A1713‘ (Freiburger Code) = $\langle a \rangle + \langle 1713 \rangle \rightarrow a$

Abbildung: Repräsentation von Befehlen in der Z22

# Freiburger Code

*In Arbeitsgemeinschaft des Mathematischen Instituts der Universität Freiburg und der wissenschaftlichen Abteilung der Firma ZUSE K.-G. wurde ein Code entwickelt, der den Gebrauch der Maschine für den Mathematiker äußerst einfach gestaltet.*

# Freiburger Code

*In Arbeitsgemeinschaft des Mathematischen Instituts der Universität Freiburg und der wissenschaftlichen Abteilung der Firma ZUSE K.-G. wurde ein Code entwickelt, der den Gebrauch der Maschine für den Mathematiker äußerst einfach gestaltet.*

| | | |
|---|---|---|
| Am | ⟨a⟩ + ⟨m⟩ → a | ⟨c⟩ → b |
| Sm | ⟨a⟩ − ⟨m⟩ → a | |
| NSm | − ⟨m⟩ → a | |
| CAn | ⟨a⟩ + n' → a | |
| CBn | n' → a | |
| CSn | ⟨a⟩ − n' → a | |
| CNSn | − n' → a | |
| Im | ⟨a⟩ ∧ ⟨m⟩ → a | |
| CIn | ⟨a⟩ ∧ n' → a | |

Abbildung: Befehlsliste im Freiburger Code (*Programmierungsanleitung* 1960, 96)

# Freiburger Code

*In Arbeitsgemeinschaft des Mathematischen Instituts der Universität Freiburg und der wissenschaftlichen Abteilung der Firma ZUSE K.-G. wurde ein Code entwickelt, der den Gebrauch der Maschine für den Mathematiker äußerst einfach gestaltet.*

| | | |
|---|---|---|
| Am | ⟨a⟩ + ⟨m⟩ → a | ⟨c⟩ → b |
| Sm | ⟨a⟩ − ⟨m⟩ → a | |
| NSm | − ⟨m⟩ → a | |
| CAn | ⟨a⟩ + n' → a | |
| CBn | n' → a | |
| CSn | ⟨a⟩ − n' → a | |
| CNSn | − n' → a | |
| Im | ⟨a⟩ ∧ ⟨m⟩ → a | |
| CIn | ⟨a⟩ ∧ n' → a | |

Abbildung: Befehlsliste im Freiburger Code (*Programmierungsanleitung* 1960, 96)

‚Interpretation' des Freiburger Codes erfolgt beim Einlesen durch das Leseprogramm (in Software!)

Section 3

# Lutz' Code

# Überblick

- Drei Teile
  - (Pseudo-)Zufallsgenerator (15 Zeilen)
  - Satzbau (221 Zeilen)
  - Lexikon (77 Zeilen)

# Überblick

- Drei Teile
  - (Pseudo-)Zufallsgenerator (15 Zeilen)
  - Satzbau (221 Zeilen)
  - Lexikon (77 Zeilen)

*Mit der Existenz eines solchen Zufallsgenerators ist das Problem der stochastischen Texte im wesentlichen gelöst. In ihrem Speicher enthält die Maschine eine gewisse Anzahl von Subjekten, Prädikaten, logischen Operatoren, logischen Konstanten und das Wort „ist“, verschlüsselt als Dualzahlen.* *(Lutz, 1959)*

## Listing 1: Lutz' Zufallsgenerator

```
1699 T1700T // Was folgt ab Position 1700 in Speicher schreiben
1700 B05    // Inhalt von Zelle 5 in Akku (Rücksprungbefehl)
1701 T1712  // Akkuinhalt → 1712
1702 B1713  // Inhalt 1713 → Akku
1703 LLA0   // Akkuinhalt um 2 bits nach links schieben
1704 LLA0   // = Multiplikation mit 4
1705 A1713  // Akkuinhalt + Inhalt 1713
1706 RA0    // Akkuinhalt um 1 bit nach rechts verschieben
1707 RA0    // = Division durch 2, abrunden
1708 RA0    // nochmal
1709 RA0    // und nochmal
1710 U1713  // Akkuinhalt → 1713
1711 CI15   // Akkuinhalt ∧ 15 (nur vier letzte Stellen)
1712 0
1713 12345678' // Startwert für Zufallszahl
1714 B0+1900   // Inhalt 1900 in Akku
1715 ...
```

# Zufallsgenerator

$$
\begin{aligned}
r_0 &= 12345678 \\
r_{i+1} &= \lfloor \lfloor \lfloor \lfloor (((r_i \times 4) \times 4) + r_i)/2 \rfloor /2 \rfloor /2 \rfloor /2 \rfloor
\end{aligned}
$$

# Zufallsgenerator

$$
\begin{aligned}
r_0 &= 12345678 \\
r_{i+1} &= \lfloor\lfloor\lfloor\lfloor(((r_i \times 4) \times 4) + r_i)/2\rfloor/2\rfloor/2\rfloor/2\rfloor
\end{aligned}
$$

12 345 678
13 117 282
13 937 112
14 808 181
15 733 692
16 717 047
17 761 862
18 871 978
20 051 476
21 304 693
22 636 236
24 051 000
25 554 187
27 151 323
28 848 280
30 651 297
32 567 003
34 602 440
36 765 092
39 062 910
41 504 341

# Zufallsgenerator

$$
\begin{aligned}
r_0 &= 12345678 \\
r_{i+1} &= \lfloor\lfloor\lfloor\lfloor(((r_i \times 4) \times 4) + r_i)/2\rfloor/2\rfloor/2\rfloor/2\rfloor
\end{aligned}
$$

- Pseudozufall, wie heute
- Steigende Werte durch Abrundung nach Division
  - → Variablenüberlauf nach 153 Iterationen
  - Generierte Zahlen werden dann negativ (zweier-Komplement!), vorderstes Bit fällt weg

12 345 678
13 117 282
13 937 112
14 808 181
15 733 692
16 717 047
17 761 862
18 871 978
20 051 476
21 304 693
22 636 236
24 051 000
25 554 187
27 151 323
28 848 280
30 651 297
32 567 003
34 602 440
36 765 092
39 062 910
41 504 341

# Code-Exegese

- Im Prinzip: Programm lesen, ab Position 1700 in Speicher schreiben und ausführen

# Code-Exegese

- Im Prinzip: Programm lesen, ab Position 1700 in Speicher schreiben und ausführen
- Aber: Bandbefehle
  - Bandbefehle werden direkt *beim Einlesen* ausgeführt

# Code-Exegese

- Im Prinzip: Programm lesen, ab Position 1700 in Speicher schreiben und ausführen
- Aber: Bandbefehle
  - Bandbefehle werden direkt *beim Einlesen* ausgeführt
  - `TmT`: Speichere das folgende Wort in die Speicherzelle $m$
  - `EmE`: Höre auf zu lesen und fange an Position $m$ an auszuführen

# Code-Exegese

- Im Prinzip: Programm lesen, ab Position 1700 in Speicher schreiben und ausführen
- Aber: Bandbefehle
  - Bandbefehle werden direkt *beim Einlesen* ausgeführt
  - `TmT`: Speichere das folgende Wort in die Speicherzelle $m$
  - `EmE`: Höre auf zu lesen und fange an Position $m$ an auszuführen

## Lutz

```
1 T1870T // Nächstes Wort: 1870
2 E1809  // Springe nach 1809
3 T1774T // Nächstes Wort: 1774
4 A0     // ⟨a⟩ + ⟨0⟩ → a
5 T1723T // Nächstes Wort: 1723
6 A0     // ⟨a⟩ + ⟨0⟩ → a
```

- 23 `TmT`-Befehle, die das gelesene Programm nach dem Einlesen verändern
- Bug fixes?

```
1720 F1700   // springe nach 1700 (Zufallsgenerator) und zurück
1721 U14     // Akkuinhalt (Zufallszahl) in Zelle 14
1722 A14     // ⟨a⟩ + ⟨14⟩ → a
1723 A14     // ⟨a⟩ + ⟨14⟩ → a
1724 A1714   // ⟨a⟩ + ⟨1714⟩ → a
1725 U1728   // ⟨a⟩ → 1728
1726 CA1     // ⟨a⟩ + 1 → a
1727 T1730   // ⟨a⟩ → 1730; 0 → a
1728 0
1729 T13     // ⟨a⟩ → 13; 0 → a
1730 0
1731 T14     // ⟨a⟩ → 14; ⟨0⟩ → a
```

```
1900 GRAF
1901 0'
1902 FREMDE
1903 0'
1904 BLICK
1905 0'
1906 KIRCHE
1907 1'
1908 SCHLOS
1909 2'
```

```
1720 F1700   // springe nach 1700 (Zufallsgenerator) und zurück
1721 U14     // Akkuinhalt (Zufallszahl) in Zelle 14
1722 A14     // ⟨a⟩ + ⟨14⟩ → a
1723 A0      // ⟨a⟩ + ⟨0⟩ → a
1724 A1714   // ⟨a⟩ + ⟨1714⟩ → a
1725 U1728   // ⟨a⟩ → 1728
1726 CA0     // ⟨a⟩ + 0 → a
1727 T1730   // ⟨a⟩ → 1730; 0 → a
1728 0
1729 T13     // ⟨a⟩ → 13; 0 → a
1730 0
1731 T14     // ⟨a⟩ → 14; ⟨0⟩
```

```
1900 GRAF
1901 0'
1902 FREMDE
1903 0'
1904 BLICK
1905 0'
1906 KIRCHE
1907 1'
1908 SCHLOS
1909 2'
```

# Ein Hack

- Zelle 1724: `A1714` – addiere Inhalt von Zelle 1714 zum Akkumulator
- Zelle 1714: `B0+1900` – Befehl zum laden von Speicherstelle 1900

→ ‚type error' – Z22 bricht ab, wenn z.B. ein Wert ins Befehlsregister kommt

# Ein Hack

- Zelle 1724: `A1714` – addiere Inhalt von Zelle 1714 zum Akkumulator
- Zelle 1714: `B0+1900` – Befehl zum laden von Speicherstelle 1900

→ ‚type error' – Z22 bricht ab, wenn z.B. ein Wert ins Befehlsregister kommt

# Das Lexikon

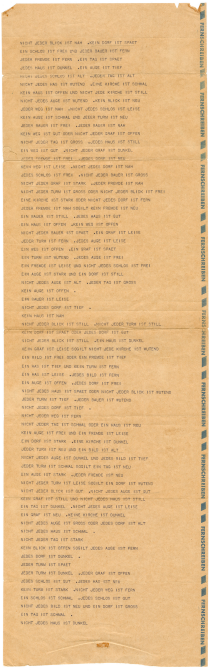
```
1981 T1982T
1982 UND
1983 UND
1984 ODER
1985 ODER
1986 SOGILT
1987 .
```

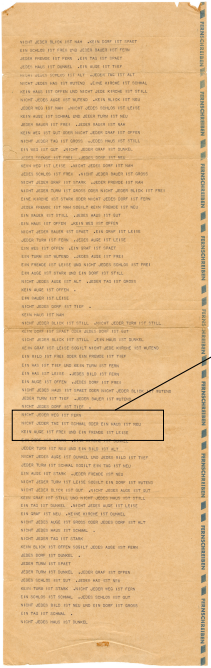
```
1989 T1990T
1990 EIN
1991 EINE
1992 EIN
1993 KEIN
1994 KEINE
1995 KEIN
1996 JEDER
1997 JEDE
1998 JEDES
1999 NICHT
2000 IST
```

```
1899 T1900T
1900 GRAF
1901 O'
1902 FREMDE
1903 O'
1904 BLICK
1905 O'
1906 KIRCHE
1907 1'
1908 SCHLOS
1909 2'
1910 BILD
1911 2'
1912 AUGE
1913 2'
1914 DORF
1915 2'
1916 TURM
1917 O'
```

```
1918 BAUER
1919 O'
1920 WEG
1921 O'
1922 GAST
1923 O'
1924 TAG
1925 O'
1926 HAUS
1927 2'
1928 TISCH
1929 O'
1930 KNECHR
1931 O
```

```
1949 T1950T
1950 OFFEN
1951 STILL
1952 STARK
1953 GUT
1954 SCHMAL
1955 NAH
1956 NEU
1957 LEISE
1958 FERN
1959 TIEF
1960 SPAET
1961 DUNKEL
1962 FREI
1963 GROSS
1964 ALT
1965 WUTEND
```



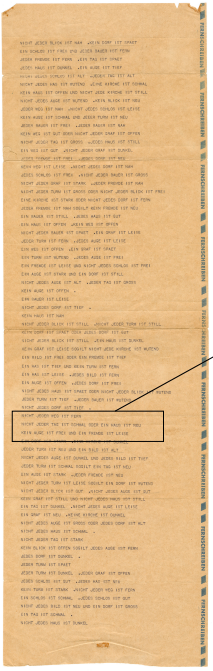


NICHT JEDER WEG IST FERN

NICHT JEDER TAG IST SCHMAL ODER EIN HAUS IST NEU

KEIN AUGE IST FREI UND EIN FREMDE IST LEISE

EIN DORF IST STARK . EINE KIRCHE IST DUNKEL



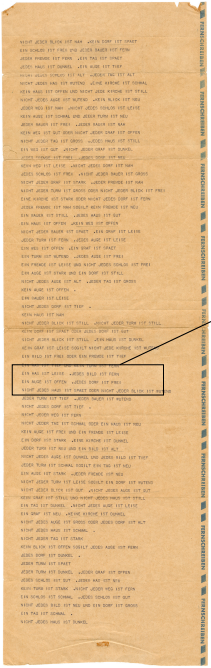
NICHT JEDER WEG IST FERN

NICHT JEDER TAG IST SCHMAL ODER EIN HAUS IST NEU

KEIN AUGE IST FREI UND EIN FREMDE IST LEISE

EIN DORF IST STARK . EINE KIRCHE IST DUNKEL

- Determinationsflexion ‚Der Fremde' vs. ‚Ein Fremder'
- Agreement nicht implementiert

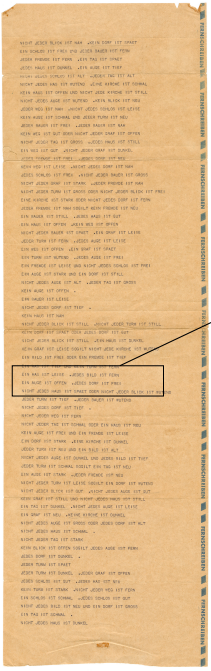


EIN HAS IST TIEF UND KEIN TURM IST FERN

EIN HAS IST LEISE •JEDES BILD IST FERN

EIN AUGE IST OFFEN •JEDES DORF IST FREI

NICHT JEDES HAUS IST SPAET ODER NICHT JEDER BLICK IST WUT

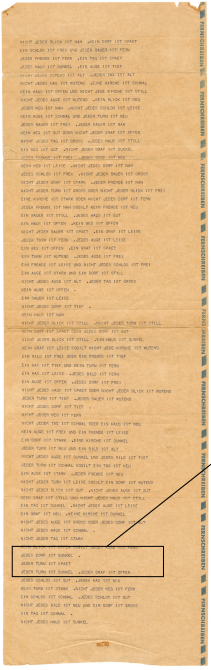


EIN HAS IST LEISE .JEDES BILD IST FERN

EIN AUGE IST OFFEN .JEDES DORF IST FREI

NICHT JEDES HAUS IST SPAET ODER NICHT JEDER BLICK IST WUT

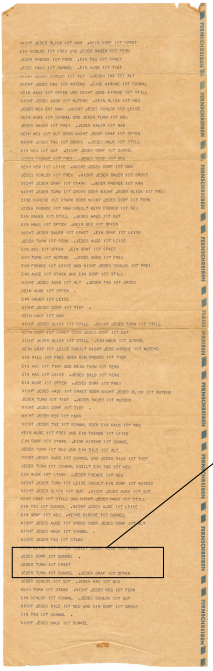
- ‚HAUS‘ im Lexikon enthalten, ‚HAS‘ nicht
- An anderer Stelle wird ‚HAUS‘ generiert
- Systematischer Fehler: Nur bei ‚HAUS‘ $\rightarrow$ ‚HAS‘ fehlt ein Buchstabe



JEDES DORF IST DUNKEL .

JEDER TURM IST SPAET

JEDER TURM IST DUNKEL .JEDER GRAF IST OFFEN



JEDES DORF IST DUNKEL .

JEDER TURM IST SPAET

JEDER TURM IST DUNKEL .JEDER GRAF IST OFFEN

/Das Dorf ist still./

Diese Elementarsätze sollen entweder miteinander verknüpft werden durch eine der drei folgenden Verknüpfungen

a) Konjunktion "und" mit einer Häufigkeit von 1/8
b) Disjunktion "oder" mit einer Häufigkeit von 1/8
c) Implikation "so gilt" mit einer Häufigk.von 1/8

oder sie sollen voneinander getrennt werden durch

d) einen Punkt "." mit einer Häufigkeit von 4/8 oder
e) einen Punkt mit nachfolgendem "Absatz", Häufigk. 1/8.

R. Gunzenhäuser an Lutz, 21.07.1959

# Limitierungen und weiterführende Überlegungen

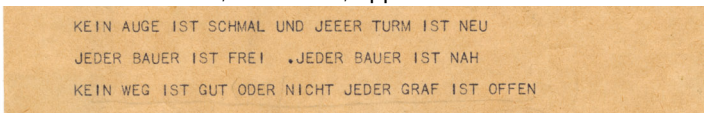
- Kein *stack*, keine Rekursion
  - Unterfunktion Zufallsgenerator schreibt Rücksprungbefehl immer an gleiche Stelle

# Limitierungen und weiterführende Überlegungen

- Kein *stack*, keine Rekursion
  - Unterfunktion Zufallsgenerator schreibt Rücksprungbefehl immer an gleiche Stelle
- Komplexere Regeln scheitern am Agreement-Handling

# Limitierungen und weiterführende Überlegungen

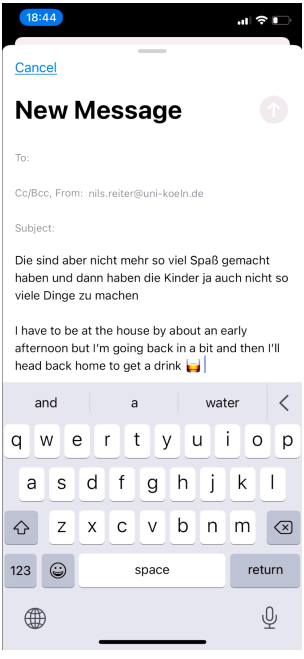
- Kein *stack*, keine Rekursion
  - Unterfunktion Zufallsgenerator schreibt Rücksprungbefehl immer an gleiche Stelle
- Komplexere Regeln scheitern am Agreement-Handling
- Kann man weitere Texte erzeugen? Ja, aber ...
  - An sich deterministisch, aber bits ‚kippen um'




  - Nicht reproduzierbar

Section 4

# Language Modeling


18:44
Cancel
New Message
To:
Cc/Bcc, From: nils.reiter@uni-koeln.de
Subject:
Die sind aber nicht mehr so viel Spaß gemacht haben und dann haben die Kinder ja auch nicht so viele Dinge zu machen
I have to be at the house by about an early afternoon but I'm going back in a bit and then I'll head back home to get a drink 🍻
and
a
water
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
return

# Introduction

- One of the oldest NLP tasks
  - Long before predictive typing on smart phones became a thing
- Language model (LM) predicts the next word, given previous words (history)
- Formally: $p(\text{word}|\text{history})$

# Introduction

- One of the oldest NLP tasks
  - Long before predictive typing on smart phones became a thing
- Language model (LM) predicts the next word, given previous words (history)
- Formally: $p(\text{word}|\text{history})$

**Example**

Maria stolperte über den großen schwarzen ____.

# Introduction

- One of the oldest NLP tasks
  - Long before predictive typing on smart phones became a thing
- Language model (LM) predicts the next word, given previous words (history)
- Formally: $p(\text{word}|\text{history})$

## Example

Maria stolperte über den großen schwarzen ____.

## Reading

Christopher D. Manning/Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts und London, England: MIT Press, Ch. 6.1–6.2. Ilias

# History

- Not all textual histories can be treated individually
  - We couldn't predict anything on completely new histories
  - Chance of a text re-appearing is astronomically slim
- Predicting the next word on unseen sentences requires *generalization*

# History

- Not all textual histories can be treated individually
  - We couldn't predict anything on completely new histories
  - Chance of a text re-appearing is astronomically slim
- Predicting the next word on unseen sentences requires *generalization*
- Instances of textual histories need to be grouped together
  - Manning/Schütze (MS99, 192): „Equivalence Classes“

# History

- Not all textual histories can be treated individually
  - We couldn't predict anything on completely new histories
  - Chance of a text re-appearing is astronomically slim
- Predicting the next word on unseen sentences requires *generalization*
- Instances of textual histories need to be grouped together
  - Manning/Schütze (MS99, 192): „Equivalence Classes“

More generalization $\xleftarrow{\text{less}}$ Equivalence classes $\xrightarrow{\text{more}}$ More discrimination

Abbildung: Compromise between generalization and discrimination

# Forming Equivalence Classes

## Different strategies

- Stemming/lemmatization: Don't look at word forms, look at lemmas or stems
  - E.e.: $p(\text{bark}|\text{the dog})$ instead of $p(\text{barks}|\text{The dog})$

# Forming Equivalence Classes

## Different strategies

- Stemming/lemmatization: Don't look at word forms, look at lemmas or stems
  - E.e.: $p(\text{bark}|\text{the dog})$ instead of $p(\text{barks}|\text{The dog})$
- Selected history: Only look at selected word classes
  - Content words like nouns, verbs, adjectives and adverbs
  - E.g., $p(\text{barks}|\text{dog})$ instead of $p(\text{barks}|\text{The dog})$

# Forming Equivalence Classes

## Different strategies

- Stemming/lemmatization: Don't look at word forms, look at lemmas or stems
  - E.e.: $p(\text{bark}|\text{the dog})$ instead of $p(\text{barks}|\text{The dog})$
- Selected history: Only look at selected word classes
  - Content words like nouns, verbs, adjectives and adverbs
  - E.g., $p(\text{barks}|\text{dog})$ instead of $p(\text{barks}|\text{The dog})$
- Both require linguistic pre-analysis of the text
  - Time-consuming and error-prone (on a large scale)

# Forming Equivalence Classes

## Different strategies

- Stemming/lemmatization: Don't look at word forms, look at lemmas or stems
  - E.e.: $p(\text{bark}|\text{the dog})$ instead of $p(\text{barks}|\text{The dog})$
- Selected history: Only look at selected word classes
  - Content words like nouns, verbs, adjectives and adverbs
  - E.g., $p(\text{barks}|\text{dog})$ instead of $p(\text{barks}|\text{The dog})$
- Both require linguistic pre-analysis of the text
  - Time-consuming and error-prone (on a large scale)
- Limit history: Only look at the last $n$ words

# Markov Assumption

- Assumption: Only the local context influences the next word

WMarkov property

# Markov Assumption

- Assumption: Only the local context influences the next word — WMarkov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

# Markov Assumption

- Assumption: Only the local context influences the next word — WMarkov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

## Example

Bigram model: „schwarzen _____“

# Markov Assumption

- Assumption: Only the local context influences the next word — WMarkov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

**Example**

Trigram model: „großen schwarzen ____“

# Markov Assumption

- Assumption: Only the local context influences the next word
  Wikipedia: Markov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

## Example

4-gram model: „den großen schwarzen _____“

# Markov Assumption

- Assumption: Only the local context influences the next word — WMarkov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

**Example**

5-gram model: „über den großen schwarzen ____“

# Markov Assumption

- Assumption: Only the local context influences the next word — WMarkov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

## Example

6-gram model: „stolperte über den großen schwarzen _____“

# Markov Assumption

- Assumption: Only the local context influences the next word — WMarkov property
- $n$-gram model: Only the last $n-1$ words are looked at to predict the $n$th word
  - Bigram model: $p(w_2|\langle w_1 \rangle)$
  - Trigram model: $p(w_3|\langle w_1, w_2 \rangle)$
  - 4-gram model: $p(w_4|\langle w_1, w_2, w_3 \rangle)$

**Example**

7-gram model: „Maria stolperte über den großen schwarzen ____“

# ChatGPT

- A chatbot powered by a large language model
- Released by the company OpenAI
  - It's just a name, the relevant things are not open
- Research release: `https://chat.openai.com/chat`
- No communicative intent: Produced words are likely continuations of previous words

demo

# Large Language Models

The good

- Texts grammatically correct
- Mostly coherent and stylistically ok

# Large Language Models

The good

- Texts grammatically correct
- Mostly coherent and stylistically ok

The bad

- Some facts are just wrong
- No one guarantees factual correctness
- Results/answers not reproducible

# Large Language Models

The good

- Texts grammatically correct
- Mostly coherent and stylistically ok

The bad

- Some facts are just wrong
- No one guarantees factual correctness
- Results/answers not reproducible

The ugly

- No one can guarantee factual correctness
- No control over training data
  - Just because many people claim something doesn't make it true
  - The internet is full of mysoginy, racism etc.
- Different groups are differently represented

# References I


- Bernhart, Toni (2020). „Beiwerk als Werk. *Stochastische Texte* von Theo Lutz“. In: *editio* 34. DOI: 10.1515/editio-2020-0010.
- Lutz, Theo (1959). In: *augenblick* 4.1, S. 3–9.
- Manning, Christopher D./Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts und London, England: MIT Press.
- *Programmierungsanleitung* (1960). *Programmgesteuerte Elektronische Rechenanlage Zuse Z 22 und Z 22/R. Programmierungsanleitung*. Zuse KG. Bad Hersfeld, Western Germany.
- *Z22/R* (1960). Zuse KG. Bad Hersfeld, Western Germany.

## und kein engel ist schön

I

jeder schnee ist kalt
und nicht jeder engel ist weiß
und nicht jeder schnee ist still
und kein friede ist kalt
und kein engel ist hell
und jeder friede ist still

II

kein friede ist weiß
oder der engel ist weiß
oder ein christbaum ist kalt
und nicht jeder friede ist schön
und ein christkind ist leise

III

jeder nikolaus ist still
oder nicht jedes kind ist still
oder der wald ist kalt
und kein wald ist schön
oder kein engel ist schön
oder der schlitten ist still
oder das kind ist weiß

IV

der schnee ist kalt
und jeder friede ist tief
und kein christbaum ist leise
oder jede kerze ist weiß
oder ein friede ist kalt
oder nicht jede kerze ist rein
und ein engel ist rein
und jeder friede ist still
oder jeder friede ist weiß
oder das kind ist still

V

ein engel ist überall

*electronus*