2. Kapitel

Symbolentschlüsselung

Kapitel 2: Symbolentschlüsselung

1. Eingliederung in den Übersetzer / Zielvorgaben
   - Eingabe und Ergebnis der Symbolentschlüsselung?
   - Welche Randbedingungen müssen beachtet werden?
   - (Informationskompression, Geschwindigkeit)
2. Theoretische Grundlage: Endliche Automaten
3. Implementierung
   - 3.1 Implementierung endlicher Automaten, Tabellenkompression
   - 3.2 die Symbolfolge
   - 3.3 Implementierung der Symboltabelle
4. Einsatz von Generatoren

2.1 Symbolentschlüsselung: Aufgabe

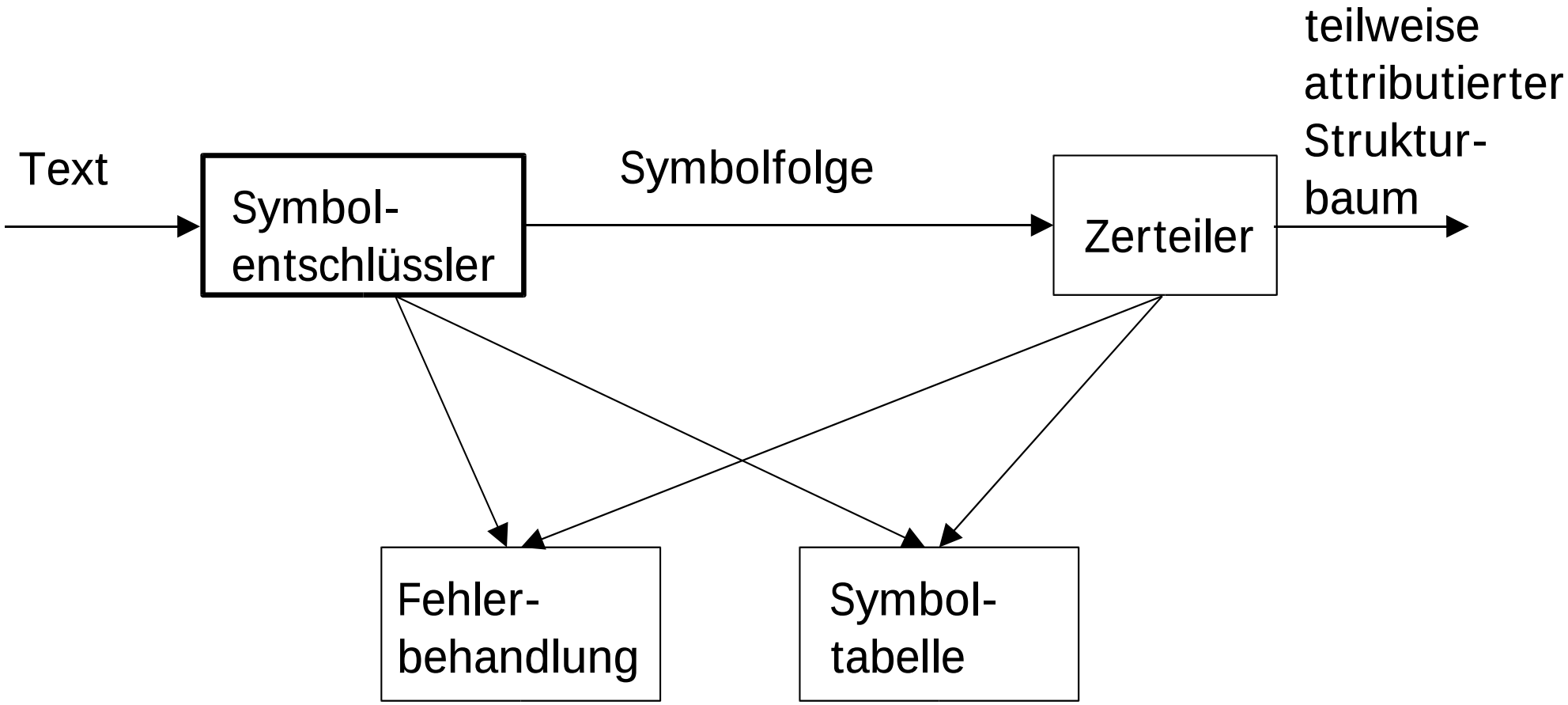
zerlegt Quellprogramm (`Text`) in Sequenz bedeutungstragender Einheiten (`Symbolfolge`)

beseitigt überflüssige Zeichen(folgen) wie

- Kommentare,
- Leerzeichen, Tabulatoren usw.

Modell: endlicher Automat aus programmiertechnischen Gründen: Geschwindigkeit höher

2.1 Eingliederung in den Übersetzer



2.1 Warum getrennte Symbolentschlüsselung?

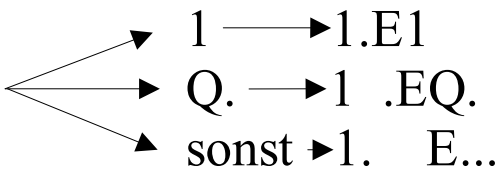
- Durchschnittliche Komplexität einer Anweisung (Knuth und andere):
  `var := var + const;`
  const $\in \{ -1, 0, 1 \}$
- 6 Symbole, aber ca. 40-60 Zeichen,
  viele (10-40) Leerzeichen pro Zeile wegen Einrückungen und Kommentaren
  - Informationskompression größer als in allen anderen Teilen des Übersetzers
  - deshalb Abtrennung
  - Aber: Kompression nur in Anzahl Symbolen, nicht unbedingt in Anzahl Bytes!

D.E. Knuth: An Empirical Study of FORTRAN Programs, Software P&E, 1(1971), 105-134

2.1 Weitere Gründe

- Beobachtung zeigt, daß bei modernen Programmiersprachen endliche Automaten ausreichen
- Umfangreiche Eingabe, daher effiziente Hilfsmittel
  (endl. Automat schneller als Kellerautomat)
- Sich selbst erfüllende Prophezeiung:
  Weil endliche Automaten ausreichen, sind moderne Programmiersprachen so formuliert, daß endliche Automaten ausreichen.

- Ausnahmen:
  - endl. Automat mit Rücksetzen am Ende: 1.E
    (Fortran, andere Sprachen: +:=,=:=,...)
  - mehrere Automaten, gesteuert von Zerteilung:
    Fortran Formate (Text ohne Anführungszeichen!)



Beim Sprachentwurf: Unabhängigkeit Symbolentschlüsselung/Zerteilung als Ziel

2.1 Beispiel: Ausnahmen

- Fortran 77
  `READ 5,ggg,...`
  ggg ist eine Formatanweisung, kein Bezeichner
- C
  `#pragma ...`
  Pragma in C: `...` kann beliebiges enthalten; ist also insbesondere nicht im Sprachstandard definiert
- Pascal
  `(*D ...*)`
  Pragma in Pascal
- Zeichenketten in vielen Programmiersprachen

2.1 ADT Symbolentschlüsselung

gelieferte Operationen:

```
initialisiere
beende
nächstes_Symbol
```

benötigte Operationen:

- Eingabe:

  ```
  öffne/schließe (Eingabedatei)
  nächstes_Zeichen oder lade_Datei
  ```

- Symboltabelle:

  ```
  initialisiere
  trage_ein(Text, Schlüssel)
  suche_oder_trage_ein(Text): Schlüssel
  gib_text(Schlüssel): Text
  ```

- Fehlerbehandlung:

  ```
  Fehlereintrag(Nr,Text)
  ```

2.1 Symbolidentifikation

Symbole können identifiziert werden durch:

- Endzustand im Automaten
  (für jedes Symbol ein eigener Endzustand)
- Symboltabellen
  (durch Vergleich mit deren Einträgen)
- Hybrider Ansatz
  - Wortsymbole und Bezeichner erst in Symboltabelle unterschieden
  - andere mit unterschiedlichen Endzuständen

2.1 Zielvorgaben

soll höchstens 6%-10% der Gesamtlaufzeit eines nicht-optimierenden Übersetzers benötigen

15% einschl. syntaktischer Analyse

Hauptaufwand: Einlesen der Quelle

2.1 Einlesen der Quelldatei

Implementierungsmöglichkeiten:

- zeichenweise: zu langsam wegen Prozeduraufruf/Systemaufruf für jedes gelesene Zeichen, nur bei Lesen von Tastatur
- zeilenweise: Zeilen unbeschränkter Länge bei generiertem Code!? doppeltes Lesen wegen Suche nach Zeilenwechsel, mehrfaches Kopieren von Puffern
- gepufferte Eingabe
- Heutzutage: komplette Datei in virtuellen Hauptspeicher: hoher Speicherbedarf bei vielen offenen Dateien

Kapitel 2: Symbolentschlüsselung

1. Eingliederung in den Übersetzer / Zielvorgaben
   - Eingabe und Ergebnis der Symbolentschlüsselung?
   - Welche Randbedingungen müssen beachtet werden?
   - (Informationskompression, Geschwindigkeit)
2. Theoretische Grundlage: Endliche Automaten
3. Implementierung
   - 3.1 Implementierung endlicher Automaten, Tabellenkompression
   - 3.2 die Symbolfolge
   - 3.3 Implementierung der Symboltabelle
4. Einsatz von Generatoren

2.2 Endliche Automaten

- Definition endliche Automaten, reguläre Grammatiken
- Prinzip des längsten Musters, Beispiel Fortran
- Reguläre Ausdrücke, Konstruktion äquivalenter endl. Automaten
  - deterministische Automaten durch Teilmengenkonstruktion
  - Minimierung durch Äquivalenzklassenbildung
  - Behandlung von End- und Fehlerzuständen
- Beispiele endlicher Automaten

Definition endlicher Automat

Ein endlicher Automat A ist ein Quintupel (T, Q, R, q_0, F), so daß:

- $Q \neq \emptyset$ ist eine endliche Menge von Zuständen
- Anfangszustand $q_0 \in Q$
- $F \subseteq Q$ sind die Endzustände
- $(T \cup Q, R)$ ist ein allgemeines Ersetzungssystem, $T \cap Q = \emptyset$
- Elemente in R: $qt \to q'$, $q, q' \in Q$, $t \in T$

A akzeptiert die Zeichenketten $L(A) = \{\tau \in T^* \mid q_0\tau \to^* q, q \in F\}$.

Die Automaten A, A' sind äquivalent gdw. $L(A) = L(A')$.

A ist deterministisch, wenn jede Ableitung mit höchstens einer weiteren Möglichkeit fortgesetzt werden kann.

Reguläre Grammatiken

Definition:

Eine Grammatik $G = (T, N, P, Z)$ ist regulär, wenn jede Produktion in P die Form $A \to a$, $A \to \epsilon$ oder $A \to aB$ hat, $A, B \in N$, $a \in T$.

Satz:

Zu jeder regulären Grammatik G existiert ein endlicher Automat A, so daß $L(A) = L(G)$ gilt.

2.2 Prinzip des längsten Musters

Wann hört der Automat auf? z.B. bei abc

- Prinzip: der Automat liest immer so weit,
  bis das gelesene Zeichen nicht mehr zum Symbol gehören kann
  - bei Bezeichnern: bis ein Zeichen erreicht ist,
    das kein Buchstabe oder Ziffer (oder Unterstrich,...) ist
- Konsequenz: der Automat startet mit dem Zeichen,
  das er beim Vorgängersymbol als letztes las
  - Grundzustand: ein Zeichen im Puffer

2.2 Symbolentschlüsselung Fortran 77

Problem:

- Zwischenräume können auch in Symbolen vorkommen,
- Symboleinteilung abhängig davon, ob Anweisung mit Wortsymbol beginnt
- Dies kann erst später entschieden werden:
  - `DO 10 I = 1.5` ist Zuweisung an Variable `DO10I`
  - `DO10I = 1,5` ist Schleifensteuerung (Zähler `I`, Endmarke `10`)

Verfahren:

- Lies gesamte Anweisung. Anweisung beginnt mit Wortsymbol, wenn
  - Anweisung kein Gleichheitszeichen (außerhalb von Klammern) enthält
  - nach einem Gleichheitszeichen ein Komma (außerhalb von Klammern) folgt
  - ...
- Andernfalls ist die Anweisung eine Zuweisung, beginnend mit einem Bezeichner

2.2 Reguläre Ausdrücke

Gegeben: Vokabular V sowie die Symbole ε, +, *, (,), [,], die nicht in V enthalten sind. Eine Zeichenkette R über V ist ein regulärer Ausdruck über V, wenn:

- R ist ein einziges Zeichen aus V oder Symbol ε, oder
- R hat die Form $(X+Y)$, (XY), $(X)^*$, X^+, $[X]_m^n$, wobei X und Y reguläre Ausdrücke sind.
- Klammern können weggelassen werden:
  * hat höchste Priorität, + eine niedrigere als Konkatenation.
- Satz:
  Für jeden regulären Ausdruck R existiert ein endlicher Automat A, so daß $L(A)=L(R)$.

2.2 Beispiel: Konstruktion eines endlichen Automaten

Regulärer Ausdruck:

$(bu\ ((bu + zi))^*)$

Einfügen von Zuständen:

$${}_0({}_1bu_2({}_3({}_4bu_5 + {}_6zi_7)_8)^*_9)_{10}$$

Übergangsregeln zwischen allen benachbarten Zuständen:

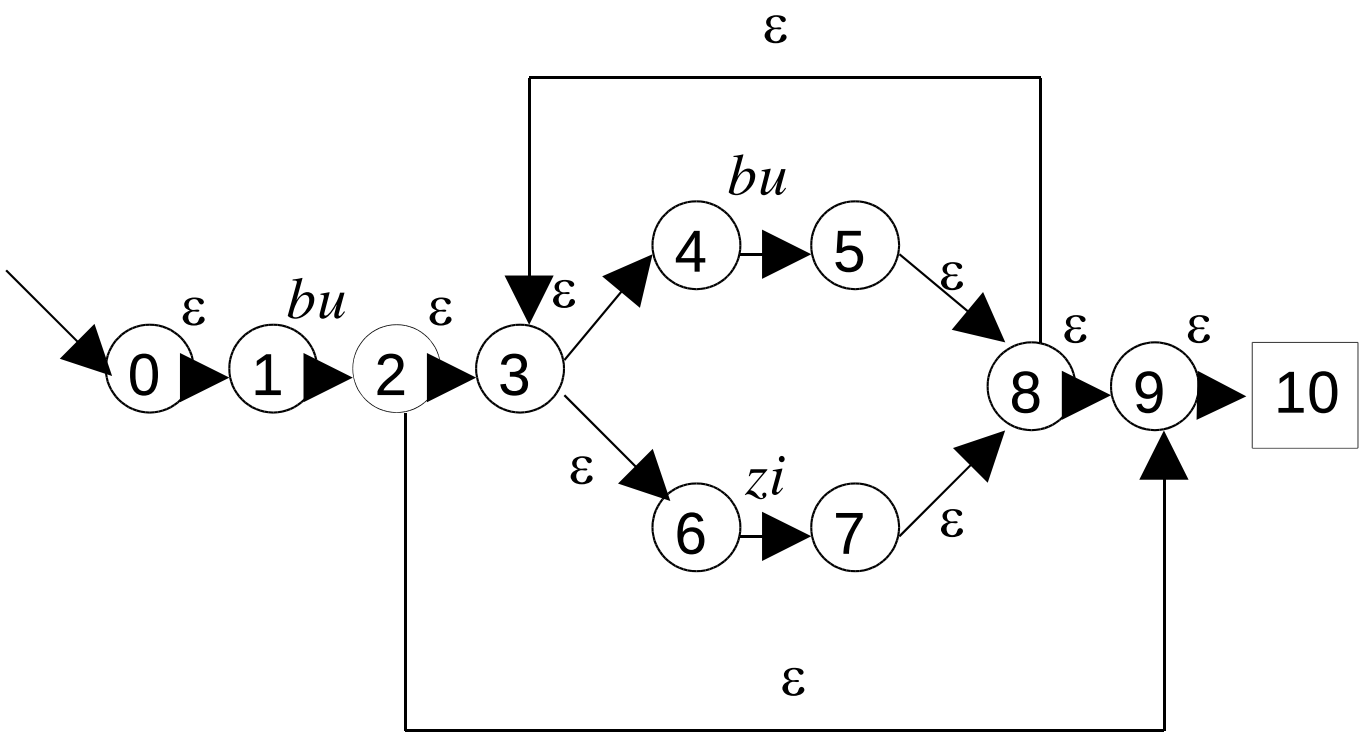
$$0\ \varepsilon \rightarrow 1,\ 1\ bu \rightarrow 2,\ 2\ \varepsilon \rightarrow 3,\ 3\ \varepsilon \rightarrow 4,\ 3\ \varepsilon \rightarrow 6,$$
$$4\ bu \rightarrow 5,\ 6\ zi \rightarrow 7,\ 5\ \varepsilon \rightarrow 8,\ 7\ \varepsilon \rightarrow 8,\ 8\ \varepsilon \rightarrow 9,$$
$$9\ \varepsilon \rightarrow 2,\ 2\ \varepsilon \rightarrow 9,\ 9\ \varepsilon \rightarrow 10$$

Konstruktion des endlichen Automaten

- Füge zu Beginn von R sowie nach jedem Terminalzeichen eine eindeutige Zahl ein (Zustände des zu konstruierenden Automaten)
- Ein einzelnes Zeichen bedeutet einen Zustandsübergang.
- $S + T$: führt zu Zustandsübergängen der zu Beginn gegebenen Zustände in alle durch Einzelzeichen erreichbaren Folgezustände.
- ST: führt zu Zustandsübergängen von allen Endzuständen von S in den Anfangszustand von T.
- S^*: führt zu Zustandsübergängen aus sämtlichen Endzuständen von S in die aus den Anfangszuständen von S mit einem Zeichen erreichbaren Zustände. Markiere Anfangszustände von S als mögliche Endzustände.

(Band 1, "Vorlesungen über Informatik" (Goos), S.93)

2.2 Beispiel: Endlicher Automat



Endzustand

Durchgangszustand

Startzustand

Deterministische Automaten

Satz: Sei A ein nichtdeterministischer endlicher Automat. Dann existiert ein deterministischer endlicher Automat A', so daß $L(A) = L(A')$ gilt.

Beweis: Teilmengenkonstruktion.

(Band 1, "Vorlesungen über Informatik" (Goos), S.94)

2.2 Teilmengenkonstruktion

Zustände in A' sind Mengen von Zuständen in A

- Initial: $q'_0 = \{q_0\}$
- ε-Übergänge: $q_i \in q' \wedge q_i \; \varepsilon \rightarrow_A q_j \Leftrightarrow q_j \in q'$
- Sonstige Übergänge:
  $\{q_1, \ldots, q_k\} \; a \rightarrow_{A'} \{p_1, \ldots, p_l\} \Leftrightarrow q_i \; a \rightarrow_A p_j$
- Endzustände:
  q'_E Endzustand in $A' \Leftrightarrow q_i \in q'_E$ Endzustand in A

Komplexität praktisch linear,
theoretisch und in pathologischen Fällen
exponentiell.

pathologisch z.B.: $(a+b)^* a (a+b)^{n-1}$

2.2 Teilmengenkonstruktion: Beweis

Induktion über Länge von Eingaben x

- Anfang $|x|=0$

  alle Zustände $q_i \in A$, die mit ε-Übergang aus q_0 erreichbar sind, sind initial in q'_0

- Schritt $|x|=n$
  - Annahme: $q'_0\, x \rightarrow^*_{A'} q'_n \Leftrightarrow q_0\, x \rightarrow_{A^*} q_n \wedge q_n \in q'_n$
  - Schluß $|x'|=n+1,\ x' = xa$:

    nach Konstruktion gilt

    $q'_n = \{q_1, \ldots, q_k\}\ a \rightarrow_{A'} q'_{n+1} = \{p_1, \ldots, p_l\} \Leftrightarrow q_i\, a \rightarrow_A p_j$

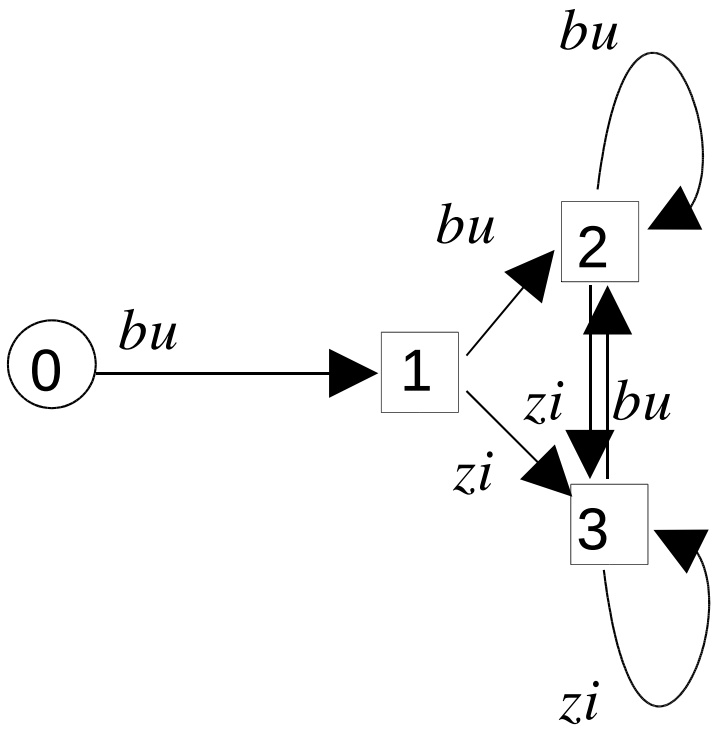
    also auch

    $q'_0\, x\, a \rightarrow^*_{A'} q'_{n+1} \Leftrightarrow q_0\, x\, a \rightarrow^*_A q \wedge q \in q'_{n+1}$

2.2 Beispiel: Teilmengenkonstruktion

- Nichtdeterministischer Automat:
  $0\ \varepsilon \rightarrow 1,\ 1\ bu \rightarrow 2,\ 2\ \varepsilon \rightarrow 3,\ 3\ \varepsilon \rightarrow 4,\ 3\ \varepsilon \rightarrow 6,$
  $4\ bu \rightarrow 5,\ 6\ zi \rightarrow 7,\ 5\ \varepsilon \rightarrow 8,\ 7\ \varepsilon \rightarrow 8,$
  $8\ \varepsilon \rightarrow 3,\ 2\ \varepsilon \rightarrow 9,\ 9\ \varepsilon \rightarrow 10$
- Teilmengen:
  $0' = \{0,1\},\ 1' = \{2,3,4,6,9,10\},$
  $2' = \{5,8,9,10,3,4,6\},\ 3' = \{7,8,9,10,3,4,6\}$
  Anmerkung: Alter Zustand kann in mehreren neuen vorkommen (z.B. 3)
- Übergänge:
  $0'\ bu \rightarrow 1',\ 1'\ bu \rightarrow 2',\ 1'\ zi \rightarrow 3',$
  $2'\ bu \rightarrow 2',\ 2'\ zi \rightarrow 3',\ 3'\ zi \rightarrow 3',\ 3'\ bu \rightarrow 2$
- Endzustände: $1',\ 2',\ 3'$

2.2 Beispiel: Teilmengenkonstruktion (resultierender Automat)



Anmerkung: Dies ist nicht der einfachste mögliche Automat, sondern nur ein deterministischer

Minimierung von Automaten

Definition: Ein deterministischer endlicher Automat heißt minimal, wenn es keine äquivalenten Automaten mit weniger Zuständen gibt.

Satz: Für jeden vollständig spezifizierten deterministischen endlichen Automat A existiert ein minimaler Automat A' mit $L(A) = L(A')$.

(Band 1, "Vorlesungen über Informatik" (Goos), S.95)

2.2 Äquivalenzklassenbildung

- Initial:
  - $\{q_1, \ldots, q_k\} = q'$ und $\{p_1, \ldots, p_l\} = q_E$
  - $p_1, \ldots, p_l$ Endzustände, $q_1, \ldots, q_k$ keine Endzustände
  - wir setzen $q_1 \equiv_0 \ldots \equiv_0 q_k$ und $p_1 \equiv_0 \ldots \equiv_0 p_l$
- Rekursion:
  - $q_i \equiv_{k+1} q_j \Leftrightarrow q_i \equiv_k q_j \wedge \forall a: q_i\, a \rightarrow p_i \wedge q_j\, a \rightarrow p_j \wedge p_i \equiv_k p_j$
- Abschluß:
  - $q_i \equiv q_j \Leftrightarrow q_i \equiv_k q_j \wedge q_i \equiv_{k+1} q_j$
  - Äquivalenzklasse je ein Zustand, Übergänge und Endzustände wie bei Teilmengenkonstruktion

2.2 Beispiel: Äquivalenzklassenbildung

Deterministischer Automat (nicht minimal):

$0\ bu \rightarrow 1,\ 1\ bu \rightarrow 2,\ 1\ zi \rightarrow 3,$

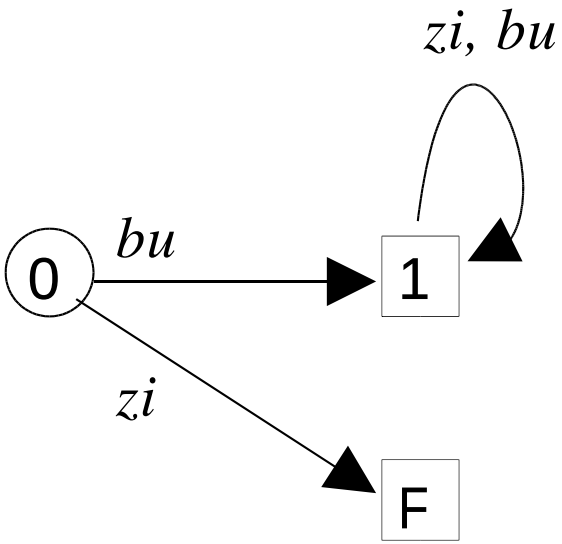
$2\ bu \rightarrow 2,\ 2\ zi \rightarrow 3,\ 3\ zi \rightarrow 3,\ 3\ bu \rightarrow 2$

- Zustände $\{0\}$ Endzustände $\{1,2,3\}$ Fehler $\{F\}$
- Partitionierung?

  $\{0\}\ bu \rightarrow \{1,2,3\},\ \{0\}\ zi \rightarrow \{F\}$

  $\{1,2,3\}\ bu \rightarrow \{1,2,3\}\ ,\ \{1,2,3\}\ zi \rightarrow \{1,2,3\}$
- Keine weitere Partitionierung

2.2 Beispiel: Äquivalenzklassenbildung (resultierender Automat)



2.2 Aufwand

- Deterministisch machen:
  - exponentiell bei pathologischen Beispielen
  - in der Praxis weniger als quadratisch
- Minimierung:
  - quadratisch
  - Es gibt auch $O(n \log n)$-Algorithmen, praktisch nicht bewährt, Programmlogik zu kompliziert, die meisten Automaten sind klein, daher nicht notwendig
- Automat unvollständig: minimaler Automat nicht eindeutig

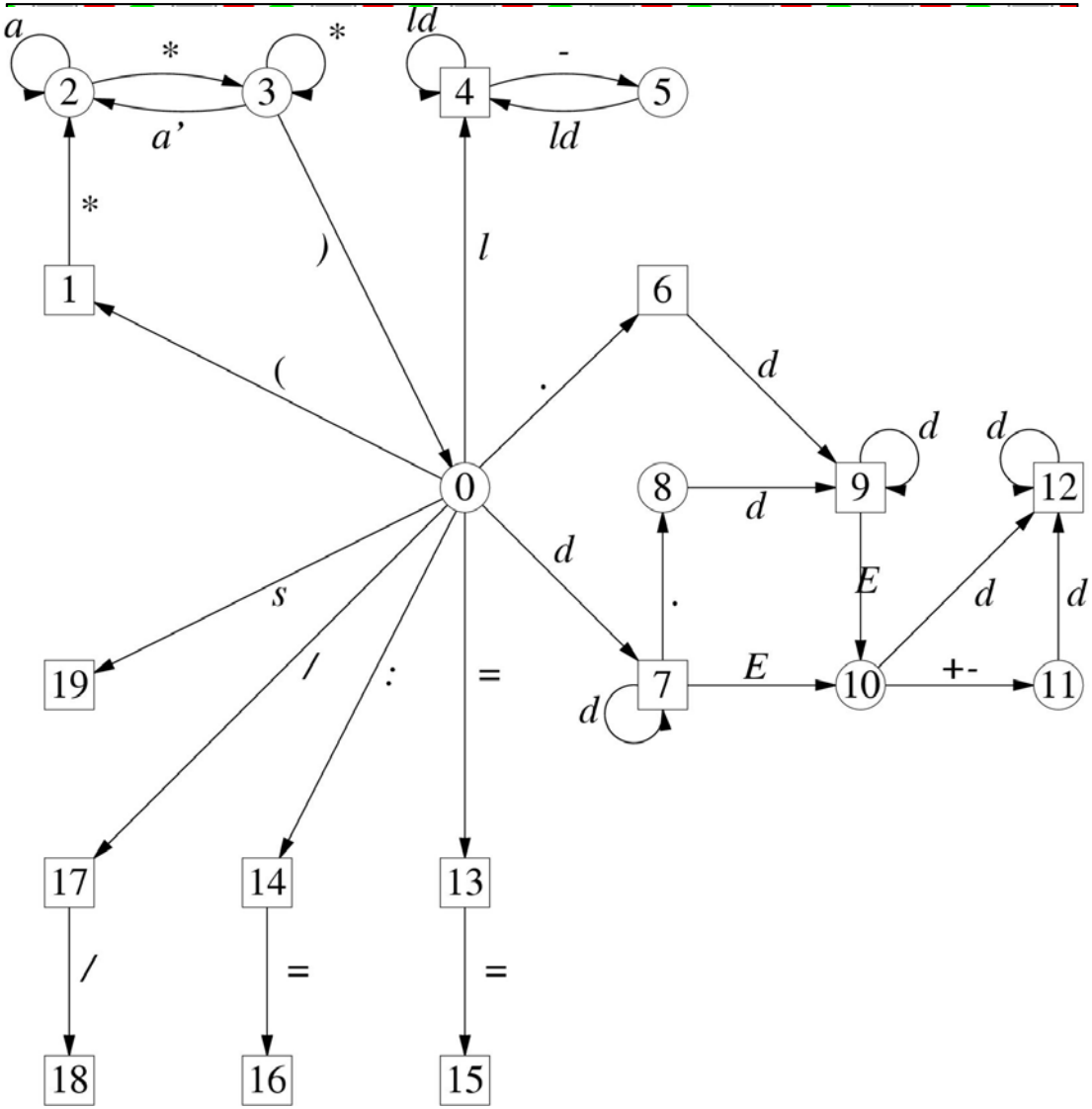
- Beispiel exponentieller Aufwand der Teilmengenkonstruktion:
  $(a+b)^*a(a+b)^{n-1}$
  Anzahl Zustände des deterministischen Automaten $\geq 2^n$

2.2 End- und Fehlerzustände

- Endzustand entsteht durch die Regel
  $A \rightarrow a$
- Jeder Fehlerzustand ist Endzustand
- Bei Minimierung müssen die End- und Fehlerzustände erhalten bleiben
  - Äquivalenzklassenbildung beginnt mit n+2 Klassen
    - Anzahl der Endzustände: n
    - Ein Fehlerzustand
    - Eine Klasse aller andern Zustände

- Beachte: Eigentlich sind alle Automaten auf den Folien und in Übersetzerbaubüchern falsch, wenn der Fehlerzustand nicht enthalten ist. Dies ist Konvention.

2.2 Symbolentschlüssler als Automat



Kapitel 2: Symbolentschlüsselung

1. Eingliederung in den Übersetzer / Zielvorgaben
   Was soll das Ergebnis der Symbolentschlüsselung sein?
   Welche Randbedingungen müssen beachtet werden?
   (Informationskompression, Geschwindigkeit)
2. Theoretische Grundlage: Endliche Automaten
3. Implementierung
   3.1 Implementierung endlicher Automaten, Tabellenkompression
   3.2 die Symbolfolge
   3.3 Implementierung der Symboltabelle
4. Einsatz von Generatoren

2.3.1 Tabellendarstellung endlicher Automaten

- Ziel: Effiziente Ausführung eines endlichen Automaten
  - Ermitteln von Übergängen in O(1)

- Alternativen
  - Adjazenzliste: O(log(k)), k maximaler Ausgangsgrad
  - Adjazenzmatrix: O(1) mit kleiner Konstante
  - Ausprogrammieren (mit indiziertem Sprung):
    O(1), aber keine Sprungvorhersage möglich
- Größe der Adjazenzmatrix = $|Q|*|T|$
  - Für klassische Alphabete ist $|T| = 256$
    ~ 40 echte Zeichen, alle andern führen in den Fehlerzustand
  - $|Q| \sim 100$
  - Problem Speicherbedarf

2.3.1 Beispiel für Tabelle

Tabelle:

Zustand	*bu*	*zi*	*trennzeichen*
0	1	*fehler*	0
1	1	1	Ende

```
Symboltabelle st =
new Symboltabelle(); ...
zustand = 0; cc=next();
while (zustand <> endzustand)
loop
 zustand=tabelle[zustand,cc];
 cc=next(); -- oder
            -- cc = zklasse[next()]
end;
if zustand == FEHLER
 return ERROR;
else
 return st.suchenOderEintragen
   (text);
```

alternativ: mehrere Endzustände, Ergebnis durch Fallunterscheidung ermittelt

2.3.1 Beispiel für Programm

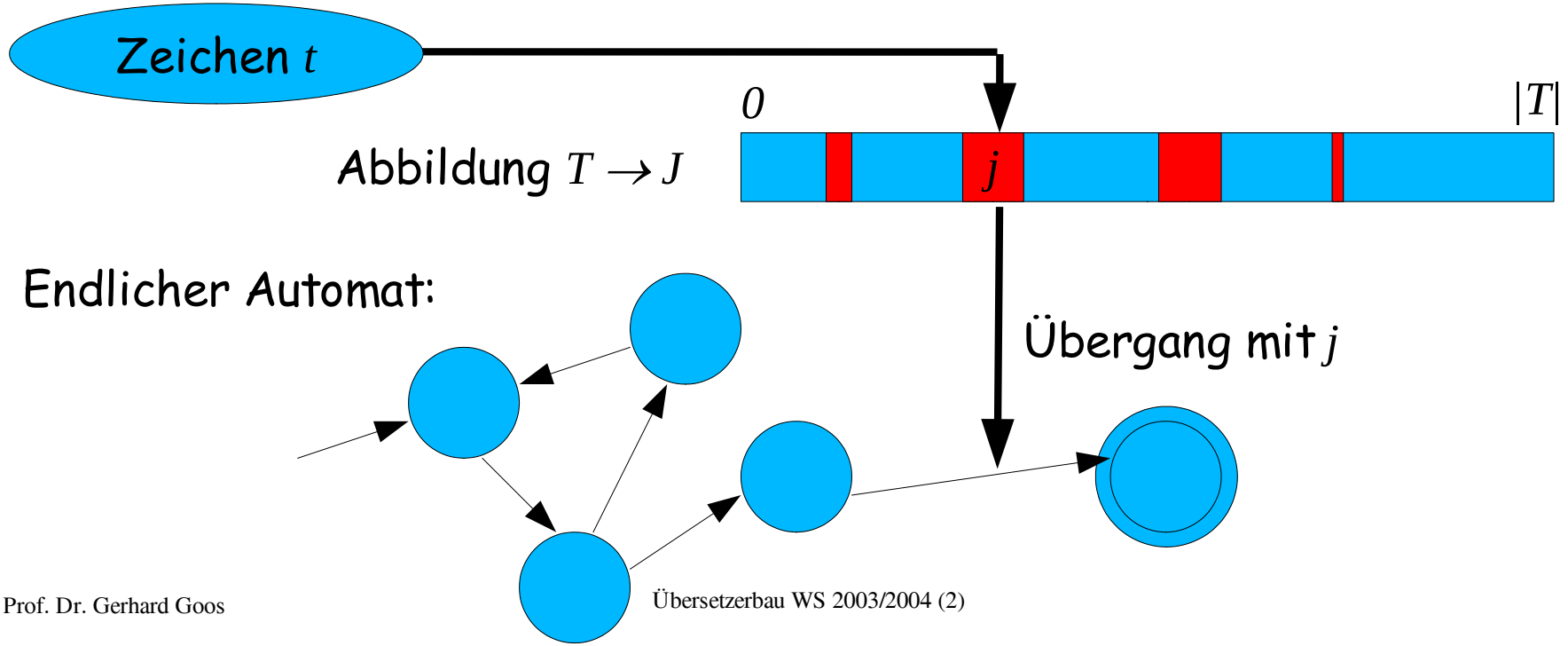
```
Symboltabelle st = new Symboltabelle();
...
zustand = 0; cc=next();
while (cc == leerzeichen) loop cc=next() end;
while (zustand <> FEHLER) loop
  case zustand of
  0: case cc of
       bu: zustand = 1;
       zi: return FEHLER;
     end;
  1: case cc of
       bu, zi: zustand = 1;
        default: return st.suchenOderEintragen(text);
     end;
  end;
  cc=next();
end;
```

2.3.1 Tabelle vs. Programm: Bewertung

- Pragmatisch: Generatoren können Tabellen besser verwenden
- Programmierte Version schneller und kleiner
- Tabelle übersichtlicher, systematischer, änderungsfreundlicher, aber langsamer
  Begründung: Tabelle ist implementiert durch Schleife mit Abfrage nach allen Eventualitäten, führt zu nicht vermeidbaren Leerprozeduren
- Erfahrung: Programmcode in beiden Fällen gleich groß, Tabelle kommt extra dazu

2.3.1 Tabellenkomprimierung

- lege „ähnliche“ Spalten zusammen: partitioniere T
- benutze „neue“ Zeichen J für Übergänge im endlichen Automaten
- Optimierung nach deterministisch Machen und Minimieren
- erfordert zusätzliche Indirektion zur Laufzeit
- Tabellenkomprimierung reduziert auf 5 bis 10% der ursprünglichen Größe
- Synergetische Effekte durch Prozessorcache



2.3.1 Besonderheiten

- Pragma: Kommentar zur Steuerung der Übersetzung
- kann eigene, nicht reguläre Syntax enthalten
- Behandlung nicht allein durch Symbolentschlüsselung möglich
- Vorgehen bei Pragmas:
  - Kommentaranfänge als Eintrag in Symboltabelle
  - Sonderbehandlung
  - abhängig von Implementierung der Symbolfolge
- schwieriges Problem, hier nicht weiter behandelt

2.3.1 Entschlüsseln von UNICODE

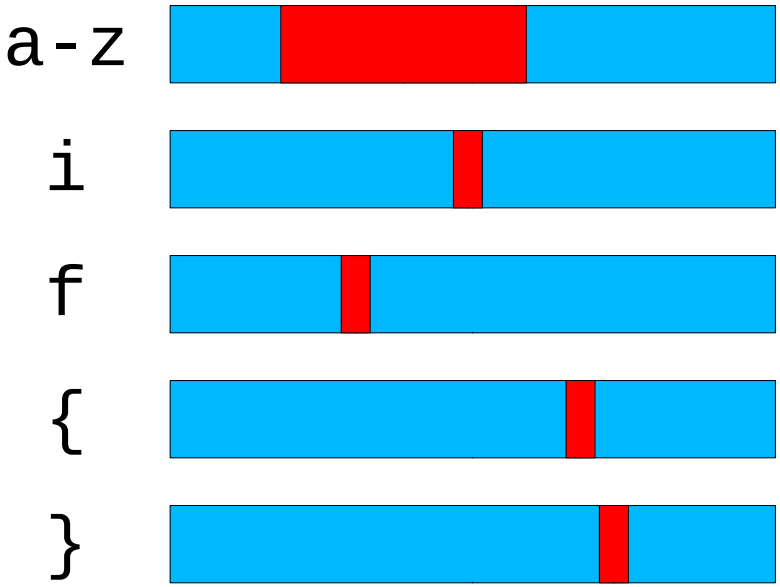
Wozu UNICODE?

- zur Internationalisierung (z.B. mit Ressource-Dateien)
- Standards wie XML, Java usw. erfordern es

- Probleme: $|T| = 2^{16}$ bei UTF-16, bei UTF-8 variable Zeichenlänge
  - zu hoher Speicher und Laufzeitbedarf für deterministisch Machen und Minimieren
- Lösung
  - Partitioniere T vorher
  - Grundidee:
    reguläre Ausdrücke $(X+Y)$, (XY), $(X)*$ mit Zeichenmengen $X, Y \subset T$
- Beobachtung: bedeutungstragend weiterhin nur ca. 40 Zeichenmengen

2.3.1 Vorgehen

Zeichenmengen:



Durch Ψ induzierte Partition:

Partitionsnummern:

0 1 2 1 3 1 0 4 5 0

Jede Zeichenmenge Z zerlegt T in Z und $T \backslash Z$.
Für eine Menge Ψ von Zeichenmengen Z bestimme die induzierte Partition von T.

Algorithmus:

- Stelle jedes `Z` als Intervall `[b,e)` dar
- sortiere nach Anfangswert `b`
- Arbeite sortierte Liste ab und erstelle Partitionierung
- Ersetze Zeichenmengen durch Partitionsnummern
- Erzeuge und benutze den Automaten wie bekannt (mit Abbildung $\Psi \rightarrow J$)

Kapitel 2: Symbolentschlüsselung

1. Eingliederung in den Übersetzer / Zielvorgaben
   Was soll das Ergebnis der Symbolentschlüsselung sein?
   Welche Randbedingungen müssen beachtet werden?
   (Informationskompression, Geschwindigkeit)
2. Theoretische Grundlage: Endliche Automaten
3. Implementierung
   3.1 Implementierung endlicher Automaten, Tabellenkompression
   3.2 die Symbolfolge
   3.3 Implementierung der Symboltabelle
4. Einsatz von Generatoren

2.3.2 ADT Symbolfolge

```
abstract class Symbolfolge is
   initialisiere();
   beende();
   nächstes_Symbol: Symbol;
end Symbolfolge;
```

2.3.2 Symbolfolge

Symbolfolge: Strom(Symbol)

Symbol: (Schlüssel, Merkmal, Position)

Schlüssel:

- das syntaktische Terminalsymbol für den Zerteiler, z.B. Code des Endzustandes im Symbolentschlüsseler oder vordefiniert für reservierte Bezeichner, z.B. Wortsymbole

Merkmal:

- Für Bezeichner und Konstanten: Verweis, der zumindest Text und Textlänge zu unterscheiden und zu rekonstruieren gestattet
  - zweckmäßig: Verweis auf ein Objekt, das zusätzliche Attribute erlaubt
- sonst: nil (Merkmal überflüssig, nicht definiert)

2.3.2 Umfang der Symbole

- Symbole (inkl. Bezeichner) in 32 bit codierbar
- Position der Symbole benötigt für Fehlerausgabe
  - Datei, Zeilen- und Spalteninformation (alternativ Zeile und Relativadresse)
  - 4 Bytes für Zeile
  - mind. 2 Bytes für Spalte
  - Positionierung bei Wysiwyg Editoren, wenn Tabulatoren beliebig definiert werden können?
  - Achtung: Zeilenzählung bei generiertem Code problematisch
- Merkmal: Maschinenbreite (oft 32 bit)
- Summe: etwa 10 Bytes

2.3.2 Symbolfolge: Implementierung

- Anfrage (Funktionsaufruf)
  - Zerteiler ruft Symbolentschlüsseler
  - Symbolentschlüsseler ruft Zerteiler
- Strom (Pipeline)
- Reihung (Symbole sind bereits abgelegt)
  - 20% schneller auf heutigen Architekturen wegen Befehls-Cache
  - das kann sich ändern

Kapitel 2: Symbolentschlüsselung

1. Eingliederung in den Übersetzer / Zielvorgaben
   Was soll das Ergebnis der Symbolentschlüsselung sein?
   Welche Randbedingungen müssen beachtet werden?
   (Informationskompression, Geschwindigkeit)
2. Theoretische Grundlage: Endliche Automaten
3. Implementierung
   3.1 Implementierung endlicher Automaten, Tabellenkompression
   3.2 die Symbolfolge
   3.3 Implementierung der Symboltabelle
4. Einsatz von Generatoren

2.3.3 Ziele und Kriterien

Ziele:

- die Merkmale bzw. syntaktischen Schlüssel aller Symbole festlegen, die nicht durch Endzustände des Automaten bestimmt werden können
- Bezeichner und Konstante durch Merkmal einheitlicher Länge codieren
- Aufbewahrung der Bezeichner- und Konstantentexte für die weitere Bearbeitung und für Fehlermeldungen
- Ankerpunkt, von dem aus verschiedene Vereinbarungen eines Bezeichners erreichbar sind (vereinfacht die semantische Analyse)

Kriterien

- anfangs unbekannte Anzahl von Bezeichnern und Konstanten unbeschränkter (!) Länge aufnehmen (Faustregel: pro 10 Zeilen 1 Bezeichner)
- Suche nach Bezeichnertexten, wenn möglich mit Aufwand O(1)

2.3.3 ADT Symboltabelle

```
abstract class Symboltabelle is
      init;
      final;
      trage_ein(Text, Schlüssel): Schlüssel;
      suche_oder_trage_ein(Text): Schlüssel;
      gib_text(Schlüssel): Text;
end Symboltabelle;
```

trage_ein dient für Voreinträge (z.B. println)

2.3.3 Suchverfahren in Symboltabelle

Sequentielle Suche (nie sinnvoll)

Suchbaum (in Fortran nötig)

Haschen

- Perfektes Haschen (nur bei vorher bekannter Bezeichnermenge)
- Verkettetes Haschen (Aufwand?)
- Haschen mit quadratischem Sondieren o.ä. (Aufwand?)
  $index_i(x) := (h(x) + i^2 * g(x)) \bmod |Tabelle|$
  i: i-te Kollision, h: Haschfunktion, g: optionale Haschfunktion (ggf. g := 1)
- Einträge: Verweise auf Symbole, gleichzeitig deren Merkmal

Achtung: Bestimmte Implementierung der Symboltabelle kann von der Quellsprache erzwungen werden, siehe Fortran

2.3.3 Umstiegspunkt für Symboltabelle

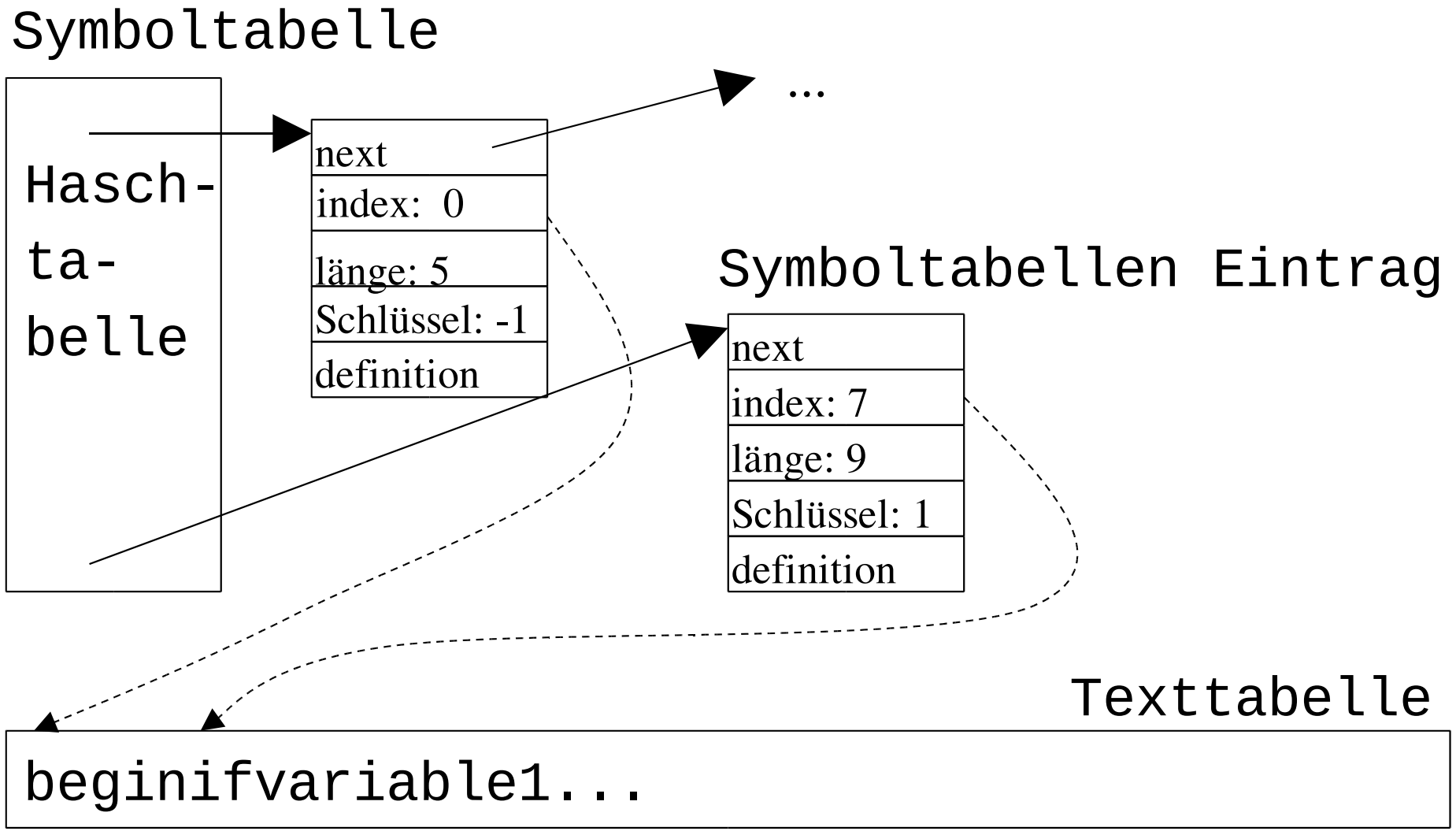
Alternativen:

- Perfektes Haschen: bestes Verfahren, wenn anwendbar
- Sondieren (z.B. quadratisch): Aufwand: Haschfunktion + # Kollisionen
- verkettetes Haschen: Aufwand: Haschfunktion + Kettenlänge
- Suchbaum: Aufwand Pfadlänge im Baum (O(log(Anzahl Einträge))?)
  in formatiertem FORTRAN sind Leerzeichen erlaubt, ständiges Berechnen von Haschschlüssel und Suchen in Tabelle ist zu teuer, im Baum ist binäre Suche möglich

Wann welche Technik?

- perfektes Haschen, wenn möglich
- quadratisches Sondieren mit Aufwand O(1), wenn Haschfunktion gleichverteilt und Haschtabelle nur halbvoll, Abhilfe Tabellenverdopplung
- verkettetes Haschen: O(1), wenn Haschfunktion gleichverteilt und Haschtabelle nur halbvoll, keine Abhilfe, wenn letzteres nicht geht
- Suchbaum: sonst

2.3.3 Organisation der Symboltabelle



2.3.3 ADT Symboltabelleneintrag

```
abstract class SymboltabellenEintrag is
      next: SymboltabellenEintrag;  //Verkettung
      index: Integer;          // Index in Texttabelle
      länge: Integer;          // Länge in Texttabelle
      schlüssel: Schlüssel;  // < 0 Schlüsselwort
      definition: Definitionstabelle;
end SymboltabellenEintrag;
```

Verweise auf Objekte dieses Typs als Merkmal für Bezeichner usw.

2.3.3 Haschen

- Tabellenlänge: Abstand zu n×256!
  - Primzahl
  - 2^p (Vermeiden von ganzz. Division), p kein Vielfaches von 8!
- Haschverfahren:
  - Haschen mit quadratischem Sondieren, ...
    - verlangt eventuell Tabellenverlängerung
  - verkettetes Haschen
- Faustregel: ca. ein Neueintrag für 10 Zeilen Quelltext

- Haschtabelle nach Ende der Symbolentschlüsselung überflüssig, nur Symboleinträge und Texttabelle werden noch benötigt

2.3.3 Kosten einer Tabellenverdopplung

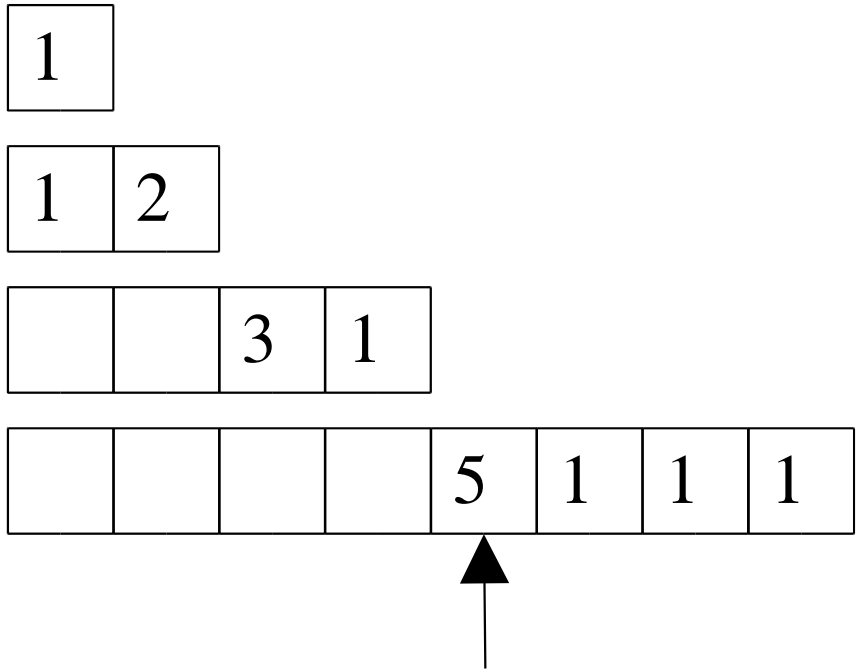
Sei

$$n = 2^k$$

Zugriffszeit für n Elemente:

$$n+n/2+n/4+n/8+...<2n \in O(n)$$

amortisiert für ein Element: $O(1)$



4 Schreibzugriffe für Kopieren + eigentlicher Schreibzugriff

Achtung: Der Index kann deshalb nicht das Merkmal eines Symbols sein, da er sich offenbar ändern kann.

2.3.3 Haschfunktionen

- Haschfunktion muß schnell berechnet werden können!
- Anfang/Ende vieler Bezeichner gleich (a1,a2,...)
- id = $id_1 id_2 ... id_k$
  - $h(id) = abs(id_1) + abs(id_k) + abs(id_{(k+1)/2})$
  - $h(id) = c_1 \times abs(id_1) + c_2 \times abs(id_k) + c_3 \times abs(id_{(k+1)/2})$
    - c_i=1,4,8? Spreizung über 0...255, abhängig vom Zeichensatz
  - $h(id) = \sum_{i=1}^{k} c_i \times abs(id_i)$
  - ...
  - alle Berechnungen modulo Tabellenlänge
- bei Haschfunktionen kommt es auf Gleichverteilung an, nicht auf die aktuelle Rechenmethode
  - z.B. spart wortweises Addieren Zeit (verletzt aber die Typregeln)

Kapitel 2: Symbolentschlüsselung

1. Eingliederung in den Übersetzer / Zielvorgaben
   Eingabe und Ergebnis der Symbolentschlüsselung?
   Welche Randbedingungen müssen beachtet werden?
   (Informationskompression, Geschwindigkeit)
2. Einlesen der Quelldatei
3. Implementierung
   3.1 Implementierung endlicher Automaten, Tabellenkompression
   3.2 Implementierung der Symboltabelle
   3.3 die Symbolfolge
4. Einsatz von Generatoren

2.4 Einsatz von Generatoren

- Symbolentschlüssler automatisch generierbar
- dazu notwendig: Beschreibung des endlichen Automaten
- Ergebnis entweder Tabellenversion oder Programmversion der Implementierung
- Beispiele für Generatoren zur Symbolentschlüsselung: Flex, Rex, Lex, ...
- Generatoren für Symbolentschlsselung und Zerteilung arbeiten zusammen, z.B. Lex und Yacc, Flex und Bison, ...

In Übersetzerbau II werden diese Systeme näher betrachtet.