Theoretische Informatik III (T3INF2002)

Formale Sprachen und Automaten | Einführung Compilerbau

Vorlesung im Wintersemester 2022/23

Formale Sprachen und Automaten

- Kontextsensitive Sprachen
- Typ-0-Grammatiken
- Turing-Maschine
- Automatentheorie

Kontextsensitive Sprachen

Kontextsensitive Grammatiken

- Kontextsensitive Grammatiken sind eine Erweiterung der kontextfreien Grammatiken.
- Diese Zeichnen sich dadurch aus, dass auf der linken Seite einer Produktion eine beliebige Kombination aus Terminal- und Nonterminalzeichen stehen darf.
- Dadurch wird es möglich, die Ersetzbarkeit eines Nonterminals an die Beschaffenheit seiner Umgebung –Kontext – zu binden.
- Produktionen haben Einschränkung:
  - Anwendung einer Produktion darf nicht zu einer Verkürzung der Zeichenkette führen
  - Ausnahme: das leere Wort ε wird direkt aus dem Startsymbol ableitet

Kontextsensitive Grammatiken

Durch kontextsensitiver Grammatiken kann die Sprache

$L_{C1} = \{a^i b^i c^i \mid i \in N+\}$ erzeugen werden.

— Konstruktion einer Grammatik lässt sich in drei Schritte unterteilen:

1. Neben dem Startsymbol S werden drei Nonterminale A, B und C eingeführt

   -> diese stehen stellvertretend für eines der Terminalzeichen a, b und c

   Durch die entsprechende Produktionen stellt man sicher, dass neben einem a, b und c die Nonterminale A, B und C beliebig oft, aber in gleicher Anzahl erzeugt werden können

Kontextsensitive Grammatiken

2. Die neuen Produktionen erlauben es, die benötigte Anzahl A's, B's und C's zu erzeugen. Diese sind ungeordnet und müssen vor der weiteren Bearbeitung in die richtige Reihenfolge gebracht werden.

-> Um die notwendige Sortierung zu gewährleisten, erweitert man die Grammatik um weitere Produktionen

3. Zum Schluss benötigt man Produktionen, die aus den Nonterminalen A, B und C die Terminalzeichen a, b und c erzeugen. - Ersetzung darf nicht beliebig erfolgen, sondern nur dann, wenn die Nonterminale in der richtigen Reihenfolge angeordnet sind.

-> Um dies zu erreichen, werden die Eigenschaften der kontextsensitiven Grammatiken genutzt, die es ermöglichen, dass Terminalzeichen auch auf der linken Seite einer Produktion stehen dürfen

Kontextsensitive Grammatiken: LC1 = {$a^i b^i c^i$ | i ∈ N+}

Erzeugen der Grammatik

Schritt 1:

S → SABC

S → abc

Kontextsensitive Grammatiken: LC1 = {$a^i b^i c^i$ | i ∈ N+}

Erzeugen der Grammatik

Schritt 1:

S → SABC

S → abc

Schritt 2:

CA → AC	cA → Ac
CB → BC	cB → Bc
BA → AB	bA → Ab

Kontextsensitive Grammatiken: LC1 = {$a^i b^i c^i$ | i ∈ N+}

Erzeugen der Grammatik

Schritt 1:

S → SABC

S → abc

Schritt 2:

CA → AC	cA → Ac
CB → BC	cB → Bc
BA → AB	bA → Ab

Schritt 3:

aA → aa

bB → bb

cC → cc

Übung: Ableitung des Worts aaabbbccc

$S \rightarrow SABC$

$S \rightarrow abc$

$CA \rightarrow AC$

$CB \rightarrow BC$

$BA \rightarrow AB$

$aA \rightarrow aa$

$bB \rightarrow bb$

$cC \rightarrow cc$

$cA \rightarrow Ac$

$cB \rightarrow Bc$

$bA \rightarrow Ab$

Entscheidungsprobleme

- Für alle Produktionen $l \rightarrow r$ einer kontextsensitiven Grammatik gilt $|l| \leq |r|$, d. h., ein Wort kann in einem Ableitungsschritt nicht kürzer werden.
- Eigenschaften können genutzt werden, um das Wortproblem zu entscheiden.
  - Um die Frage $\omega \in L$ für ein Wort mit $|\omega| = n$ zu beantworten, beginnt man mit einer Menge, die nur das Startsymbol enthält.
  - Anschließend reiht man in einem iterativen Prozess alle Wörter an, die durch die Anwendung einer Schlussregel entstehen und eine Länge $\leq n$ besitzen.
- Da nur endlich viele Wörter existieren, deren Länge $\leq n$ ist, wird nach endlich vielen Schritten entweder das gesuchte Wort ω erzeugt oder ein Fixpunkt erreicht.
- Im ersten Fall ist das Wortproblem positiv, im zweiten Fall negativ entschieden.

Entscheidungsprobleme

- Anders als beim Wortproblem, sind das Leerheitsproblem, das Endlichkeitsproblem und das Äquivalenzproblem für kontextsensitive Sprachen nicht entscheidbar.
- Es gibt kein Verfahren, das die Fragestellung für eine beliebige Grammatik immer korrekt beantwortet.

Typ-0-Grammatiken

Eine Grammatik G = (V, T, P, S) heißt Typ-0-Grammatik, wenn alle Produktionen die Form

u -> v mit u ∈ A+, v ∈ A* haben.

- Produktionen einer Typ-0-Grammatik unterliegen keinen Einschränkungen.
- Auf der linken Seite einer Produktion steht ein beliebiges, nichtleeres Wort über dem Gesamtalphabet A = V vereinigt mit T, auf der rechten Seite ebenfalls ein beliebiges, aber möglicherweise auch leeres Wort über A.

Typ-0-Grammatiken

- Ausdrucksstärke von Typ-0-Grammatiken führt dazu, dass man nur noch begrenzt Aussagen über die erzeugten Sprachen machen kann.
- Das Wortproblem, das Leerheitsproblem, das Endlichkeitsproblem und das Äquivalenzproblem sind daher unentscheidbar.

Turing-Maschine

Berechenbarkeitstheorie und Turing-Maschine

- Turing-Maschine ist das älteste und gleichzeitig am häufigsten genutzte Modell, um den Berechenbarkeitsbegriff formal zu erfassen
- Die Turing- Maschine erfüllt in jeder Hinsicht die Anforderungen eines formalen Modells, so dass sie mathematisch präzise Aussagen über den Berechenbarkeitsbegriff ermöglicht.
- Mit den Berechenbarkeitstheorien wird untersucht, welche Probleme durch welche Maschinen lösbar sind
- Rechnermodell und Programmiersprachen sind Turing-vollständig, wenn mit ihnen eine universelle Turingmaschine simuliert werden kann

Berechenbarkeitstheorie und Turing-Maschine

Alan Turing hat die Turing-Maschine entwickelt, die ein mathematisch einfaches Modell für Algorithmen darstellt

- Definition der Turing-Maschine geht darauf zurück, wie Menschen eine systematische Berechnung (z. B. eine Addition) durchführen
- Die jeweilige Aktion hängt nur von endlich vielen Zeichen ab
- Berechnung selbst wird gesteuert durch eine endliche Berechnungsvorschrift

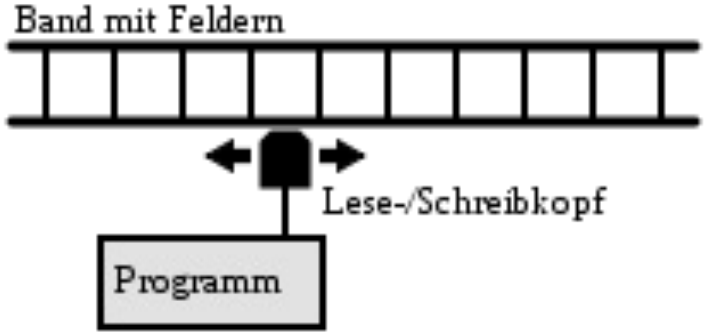
Turing-Maschine

Eine Turing-Maschine ist eine einfache Maschine und besteht aus:

- einem potentiell unendlichen Band, welches in Felder eingeteilt ist und pro Feld genau ein Zeichenaufnehmen kann,
- einem Schreib-Lesekopf und
- einem internen Zustand.

Je nach Zustand und Inhalt des Bandes kann die Turing-Maschine folgende Aktionen ausführen:

- ein neues Zeichen Schreiben,
- den Schreib-Lesekopf um eine Position nach rechts oder links bewegen und
- den internen Zustand verändern.



https://upload.wikimedia.org/wikipedia/commons/thumb/c/c4/Turingmaschine.svg/301px-Turingmaschine.svg.png

Turing-Maschine

Ein Turing-Maschine besteht aus:

- einer endlichen Zustandsmenge Z,
- einer endlichen Menge von Eingabezeichen Σ,
- einer Menge von zulässigen Bandzeichen $\Gamma \supset \Sigma$,
- Startzustand $z_0 \in \mathrm{Z}$,
- Leerzeichen[ ] $\in \Gamma - \Sigma$,
- Endzustand $z_e \in \mathrm{Z}$,
- Transitionstabelle $\Delta \subseteq \mathrm{Z} \times \Gamma \times \Gamma \times \{l, r, n\} \times \mathrm{Z}$

Transitionstabelle besteht aus 5-Tupeln $(z, \gamma, \gamma', a, z')$

Turing-Maschine

Transitionstabelle besteht aus 5-Tupeln $(z, \gamma, \gamma', a, z')$

z - ist der aktuelle Zustand der Maschine

γ - ist das Zeichen unter dem Schreib-Lesekopf

a - ist die auszuführende Aktion (l, r oder n für links, rechts oder noop)

γ' - ist das vorher zu druckende Zeichen

z' - ist der Nachfolgezustand

Beispiel einer Transitionstabelle:

z_0	a	b	r	z_0
z_0	b	a	r	z_0
z_0	[ ]	[ ]	n	z_0

Turing-Maschine

Church'sche These

Sie fasst zusammen, dass alles, was berechenbar ist, durch Turing-Maschinen beschreibbar ist.

Jede im intuitiven Sinn berechenbare Funktion ist Turing-Maschinen berechenbar.

— Da Turing-Maschinen Algorithmen beschreiben, können alle berechenbaren Funktionen genau durch den Begriff Algorithmus charakterisiert werden.

Komplexitätstheorie

- In der Komplexitätstheorie wird der Ressourcenbedarf algorithmisch bearbeitbarer Probleme auf mathematisch definierten formalen Rechnermodellen untersucht.
- Dabei spielen die Laufzeit und die Speicherkapazität eine besondere Rolle, die mit der Landau-Notation dargestellt werden.
- Es werden die Laufzeit und der Speicherplatzbedarf in Abhängigkeit der Länge der Eingabe notiert.
- Algorithmen, die sich durch einen konstanten Faktor in ihrer Laufzeit bzw. ihrem Speicherbedarf unterscheiden, werden durch die Landau-Notation derselben Klasse, d. h. einer Menge von Problemen mit äquivalenter vom Algorithmus für die Lösung benötigter Laufzeit, zugeordnet.

Automatentheorie

Grundlagen der Automatentheorie und formale Sprachen

Automaten (in der Informatik) sind einfache Modelle von zustandsorientierten Maschinen, die sich nach bestimmten Regeln (dem Programm) verhalten.

Alphabet einer Sprache:

- Programme werden in einer Programmiersprache geschrieben.
- Die Menge aller Zeichen die sie zulässt, wird als Alphabet der jeweiligen Programmiersprache bezeichnet.
- Zulässige Zeichen sind z.B. <, =, +, - sowie Buchstaben und Ziffern

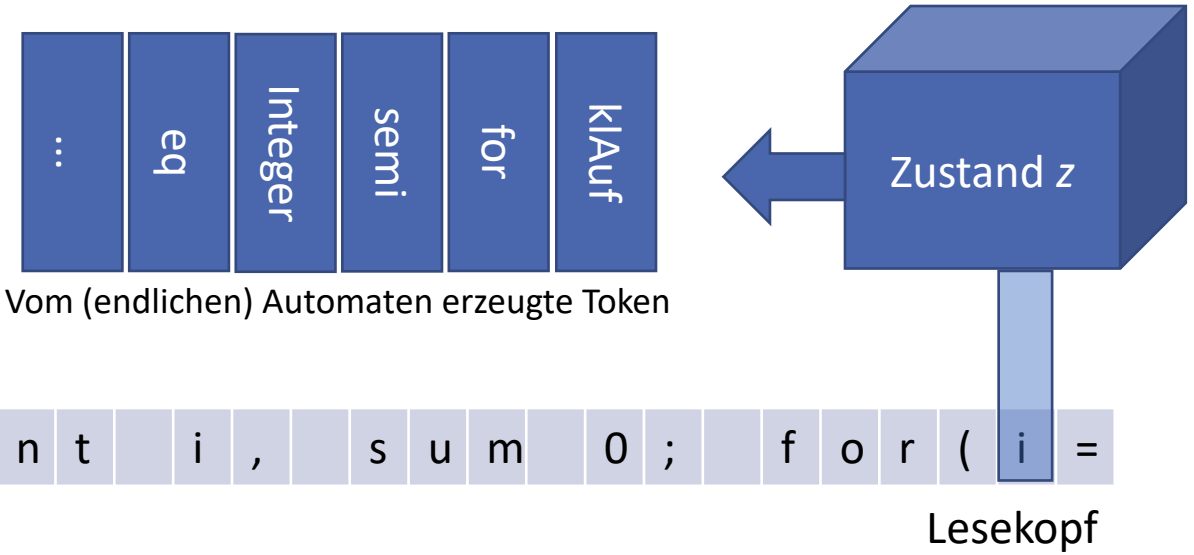
Wörter einer Sprache:

- Wörter werden aus dem Alphabet definiert. Sie unterteilen sich in *Schlüsselwörter* (while, for, if), *Sonderzeichen* (+, -, =), *Benutzerdefinierte Bezeichner* (name, wohnort) und *Konstanten* (123, 5.19, ‚r', ‚Hallo')

1. Phase: Lexikalische Analyse

Das **Programm wird in** seine einzelnen Wörter sogenannte **Token zerlegt** und alle *white spaces* (Leer-, Tabulator-, Neuezeilezeichen) entfernt. Das **Token** gibt an, zu welcher **Klasse von Wörtern** das Wort gehört. Jedem Schlüsselwort ist ein eigener Token zugeordnet, die Zeichenkette wird in eine effizientere Codierung (Token) überführt. Zuordnung von Token wird auch als Scannen bezeichnet.

```
main() {
 int i, sum=0;
 for (i=1; i<100; i = i+2)
         sum = sum + i;
}
```



Vom (endlichen) Automaten erzeugte Token

2. Phase: Syntaktische Analyse

Die Syntaxanalyse prüft, ob die Reihenfolge der einzelnen Wörter (Tokens) in der zugrundeliegenden Programmiersprache erlaubt sind. Jede Programmiersprache besitzt eine Grammatik, an die sich die Quellprogramme (bzw. die Programmierer) halten müssen.

Die Überprüfung wird auch als Parsen und entsprechende Analyseprogramme als Parser bezeichnet.

Richtige Grammatik in C

```
main() {
 int i, sum=0;
 for (i=1; i<100; i = i+2)
         sum = sum + i;
}
```

Falsche Grammatik in C

```
main() {
 int i, sum=0;
 for (i=1; i<100; i = i+2)
         sum = sum + * i;
}
```

Reguläre Ausdrücke

Reguläre Ausdrücke beschreiben eine Familie von formalen Sprachen.

Ein Regulärer Ausdruck ist eine Zeichenkette, die der Beschreibung von Mengen von Zeichenketten mit Hilfe bestimmter syntaktischer Regeln dient.

Zu jedem regulären Ausdruck existiert ein endlicher Automat, der die vom Ausdruck spezifizierte Sprache akzeptiert.

Sei Σ = { a, b } ein Alphabet.

Reguläre Ausdruck	Reguläre Sprache	Worte
a|b	{ a, b }	a, b
ab*a	{a}{b}*{a}	aa, aba, abba, abbba, …
(ab)*	{ ab }*	ε, ab, abab, …
abba	{a}{b}{b}{a}	abba

Endliche Automaten

- Ein **endlicher Automat** ist ein sehr einfaches Modell einer zustandsorientierten Maschine, die eine **endliche Menge** innerer **Zustände** hat.
- Ein Eingabewort wird **Zeichenweise** eingelesen und führt dem Programm entsprechend bei jedem Zeichen eine **Zustandsänderung** durch.
- Endliche Automaten haben eine **gekennzeichneten Startzustand** und eine definierte **Menge von Endzuständen**. Wird ein **Endzustand** beim Lesen (Zustandsübergange) eines Wortes **erreicht**, so ist dieses in der Sprache **vorhanden und wird akzeptiert**.

Die Menge alle von einem Automaten akzeptierten Wörter werden als die akzeptierte Sprache bezeichnet.

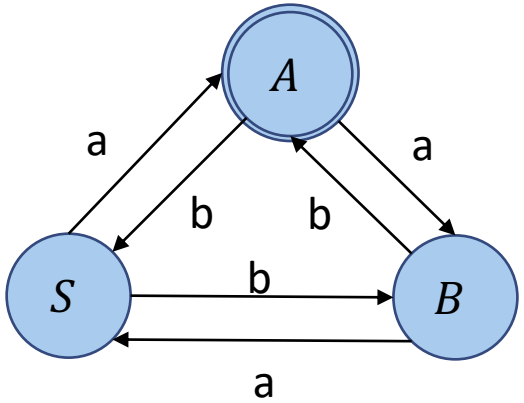
Endliche Automaten - Bestandteile

Zustände:

Zustandsübergänge: ⟶

gerichtete Kanten werden beschriftet
mit a ∈ Σ, Σ Alphabet

Beispiel



Vom Automaten akzeptierte Sprache ist die Menge aller Wörter w ∈ Σ^* , die vom Start- in den Endzustand führen

Endliche Automaten

Automat der alle Wörter, die durch den regulären Ausdruck 01(001)*01 abgebildet werden können, erzeugen kann.

Übung: Endlicher Automat

Erstellen Sie einen endlichen Automaten, der den Ausdruck $[+\ \ -]?\,[1-9][0-9]^* + 0$ abbilden kann!

? – steht für optionales Zeichen

* - mindestens einmal

Endliche Automaten und Kontextfreie Sprachen

Endliche Automaten werden genutzt, um in einer Programmiersprache erlaubte Wörter zu beschreiben.

Scanner, die **endliche Automaten** realisieren, erstellen eine Folge von Token. In der Syntaxanalyse muss bestimmt werden, ob die gelieferte Folge an Token in der jeweiligen Sprache erlaubt ist oder nicht.

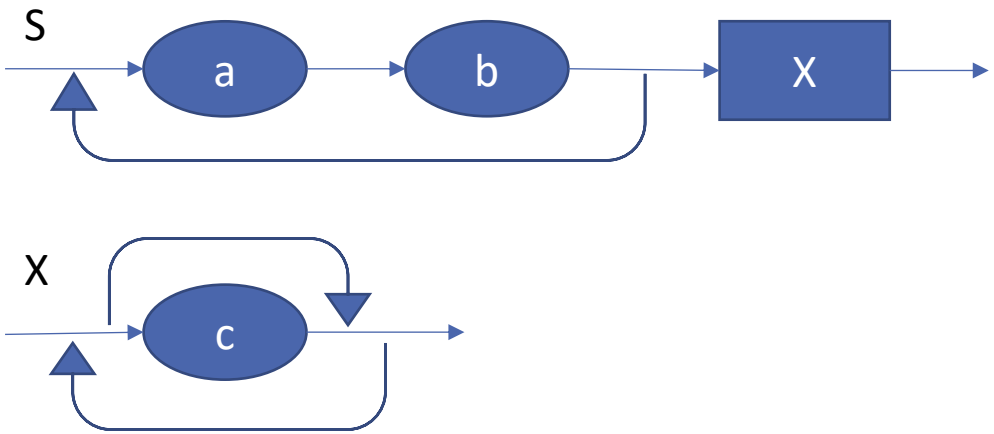
Endliche Automaten reichen nicht aus, um alle erlaubten bzw. nicht erlaubten Token zu erkennen.

- So gilt ein arithmetischer Ausdruck wie $\big(a * \big((b + c) * 7\big) + y\big)$nur dann als korrekt, wenn die Anzahl öffnender und schließender Klammern identisch ist.
- Endliche Automaten können dies nicht erkennen, da sie nur Zustandsübergange besitzen, ohne Gedächtnis.

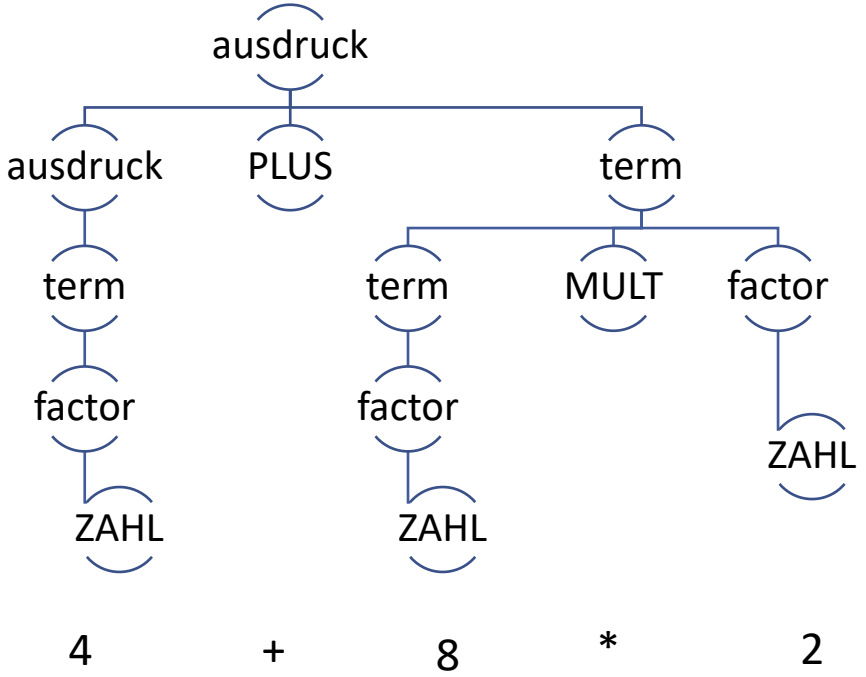
Um das vorgenannte Problem zu lösen, setzt man *Kontextfreie Grammatiken* ein.

Kontextfreie Grammatik

Syntaxdiagramm



Backus-Naur-Form (parse tree)



Im parse tree wird zunächst nach allen Kindknoten geschaut, weshalb der Ausdruck 8*2 vor dem Wert 4 addiert wird

Kellerautomaten

Zur Erzeugung **kontextfreier Sprachen** können keine endlichen Automaten verwendet werden, hierfür nutzt man die sogenannten **Kellerautomaten** oder Stackautomaten.

Ein Kellerautomat liest die Eingabe zeichenweise von links nach rechts ein, wobei das Eingabezeichen möglich sofort verarbeitet wird.

- Handelt es sich um Zeichen, die nicht unmittelbar ausgeführt werden können z.B. eine Klammer (, wird das Zeichen in einem eigenen Stack abgelegt und die Verarbeitung aufgeschoben bis die schließende Klammer) gelesen wird.
- Bei der Verarbeitung der Zeichen wird der Inhalt des Stacks berücksichtigt (wichtig ist das oberste Zeichen).

Die Eingabe ist erfolgreich abgearbeitet, wenn am Ende der Stack leer ist. Sind noch Zeichen darin enthalten hat ein auflösendes Zeichen z.B. schließende Klammer gefehlt.