1 Fragen

1. Lösung des Shift/Reduce Konflikt. Was steht auf dem Keller? Was ist das nächste Zeichen?
2. Wie funktioniert der Keller im Bezug auf die Elimation der linksrekursion? Script Seite 174
3. Wie geh ich mit Mehrdeutigkeit um im Shift Reduce Kontext?
4. Nicht-deterministischer Übergang?
5. Teilkonstruktion Mengen?
6. Wie löse ich den Reduce/Reduce Konflikt?
7. Wie geh ich bei Lark vor?
8. Wie gehe ich mit dem Ast um?
9. Wie wird Parser/rex umgesetzt?

   Ich antworte ihr auf die Frage

2 Aufgaben

Auf Seite 7 im Script sind die Übungsaufgaben verzeichnet
Laut den Alt-Klausuren

- Strukturierung von einem Übersetzer
- Fragen zur Grammatik!!!

  rechts links regulär definition

  Pumping Lemma (Siehe Inkodo Fragen)

  Kontextfrei/Kontextsensitive
- Chomsky-Hierarchie

  Arten von Grammatik 0 bis 3
- Lark+Ast oder Rex

- Lark-Spezifikation
- Top-Down-Parser/Rekursiver Abstiegs-Parser
- Abstrakter Syntaxbaum
  - Grammatik
  - Automat
  - Ableitung

2.1 Laut Vollmer

- Scanner
- Baum
- rekursiver Abstiegsparser (ist ein LL-Automat)
- Makefile nicht
- LL-Automat Mengen schneiden
- Lark-Spezifikationen
- Rex
- Insert Rules in der Reihenfolge da sonst First-Rule-Match
- Abstrakter Syntaxbaum
- Parser (ist ein LR-Automat) -Spezifikation
- Algorithmus: AST-Aufbau Button-Up
- Attibut Grammatik
- Funktionale Spezifikation einer Attributberechnung (Baum gegeben)

Andere Aufgaben

- Quiz File
- Teilmengenkonstruktion NEA in DEA Automat zeichnen
-

3 LL-Eigenschaften

Seite 166 im Script stehen die Eigenschaften
Wie andere Grammatik transformiert findet man im Script auf Seite 169 Eine Grammatik kann nicht die LL-Eigenschaften erfüllen wenn sie linksrekursion bzw. linksgleiche Produktionen enthält (Was sind Produktionen?) Allgemeine Elimination von linksrekursion auf Seite 171 im Script

$$\begin{aligned} A ::= A\alpha \\ \Longrightarrow \\ A ::= \beta A' \\ A' ::= \alpha A' | \epsilon \end{aligned}$$

Definition (Linksfaktorisierung). Problem $FIRST(..FOLLOW(..))$ nicht disjunkt:

$$\begin{aligned} A ::= \alpha\beta_1 | \alpha\beta_2 \\ \Longrightarrow \\ A ::= \alpha A' \\ A' ::= \beta_1 | \beta_2 \end{aligned}$$

$FF_1 = FIRST(TE'FOLLOW(E)) = i*$ das in den geschweiften Klammern ist das $First(T)$ wenn ϵ nicht in der First-Menge ist. Falls doch ist es das $First$ vom nächssten nicht Terminal. Falls alle ein ϵ in ihrer First_Menge haben ist es das $Follow(E')$.

3.1 LL-Bedingungen

Die Grammatik erfüllt die LL-Bedingungen wenn die gleichen Follows in den First-Follow-Mengen einen unterschiedlichen Inhalt haben.

$$\begin{aligned} FF_1 = FIRST(TE'FOLLOW(E)) = i* \\ FF_2 = FIRST(\epsilon FOLLOW(E)) = \# \end{aligned}$$

3.2 LL-Automaten

Seite 177 findet man die LL-Automaten

- Der LL-Automat erzeugt eine Linksableitung des Eingabewortes
- Erfüllt G die LL-Bedingungen, dann kann ein deterministischer Automat konstruiert werden.

1. Transfomieren Sie die Grammatik, so dass die Grammatik die LL(1) Bedingung erfüllt
2. Erstellen Sie den nichtdeterministischen LL(1)-Automaten für diese Grammatik
3. Erstellen Sie hieraus den deterministischen LL(1)-Automaten
   (nun ja er ist nicht ganz deterministisch, da die Produktionen eines Nichtterminals die LL(1) Bedingung nicht erfüllt, erstellen Sie den Automaten trotzdem!)
   Markieren Sie die nichtdeterministischen Automatenregeln.
4. Akzeptieren Sie mit diesem Automaten das "Programmï + i[i+i]

Beispiel Aufgabe auf Seite 179 im Script

- Grammatik linksrekursion rausbekommen
- First-Follow-Menge berechnen
- LL1 Eigenschaften herausfinden
- Automat
- Automat mit First-Follow

Die Regel schreib man einfach umgekehrt zur Grammatik.

$$E' ::= +TE'$$
$$E'qt \longrightarrow E'T+$$

4 LR-Automaten

Beispiel 75 *(Ausdrucksgrammatik)*

```
Grammatik G = (N, T, P, E),
T = {+, *, (, ), i},
N = {E, T, F}
P = {1) E ::= T       2) E ::= E + T
     3) T ::= F       4) T ::= T * F
     5) F ::= i       6) F ::= ( E )}


LR-Automat A = (T, {q}, R, q, {q}, N ∪ T, ε)
R = {1)   Tq → Eq    2) E+Tq → Eq
     3)   Fq → Tq    4) T*Fq → Tq
     5)   iq → Fq    6) (E)q → Fq
     q+ → +q, q* → *q, q) → )q, q( → (q, qi → iq
     Eq# → q#
    }
```

Beispiel 76 *(Ausdrucksgrammatik, LR-Ableitung)*

Regel	*Keller*	*Eingabe*	*umgekehrte Rechtssableitung*
shift	q	1 + 2 * 3 #	
reduce 5	i	q + 2 * 3 #	1 + 2 * 3
reduce 3	F	q + 2 * 3 #	F + 2 * 3
reduce 1	T	q + 2 * 3 #	T + 2 * 3
shift	E	q + 2 * 3 #	
shift	E +	q 2 * 3 #	
reduce 5	E + i	q * 3 #	E + 2 * 3
reduce 3	E + F	q * 3 #	E + F * 3
shift	E + T	q * 3 #	
shift	E + T *	q 3 #	
reduce 5	E + T * i	q #	E + T * 3
reduce 4	E + T * F	q #	E + T * F
reduce 2	E + T	q #	E + T
reduce	E	q #	E
		q #	

4.1 Konflikte

Shift/Reduce

Mehrdeutigkeit wird unterbunden in dem bevorzugt gesschiftet wird. $\Longrightarrow$ Lösung Dangeling-Else Problem. Nicht immer shiften Operatorvorrang (Skript

Seite 198f)

Reduce/Reduce

Macht es einen Unterschied?
Wenn nicht eine Regel wegschmeisen

5 Struktur vom Compiler

1. Wozu kann ein Übersetzerbenutzt werden?

   Erzeugen von Maschinencode

   Programmanalyse und das Füllen einer Datenbank mit Informationen über das Programm

   Programmtransformation in eine andere Programmiersprache

2. Wie ist ein Übersetzer strukturiert? Beschreiben Sie **kurz** die Aufgaben und Ergebnisse der einzelen Phasen!

   1. Lexikalische Analyse, 2.syntaktische Analyse, 3. Transformation(nicht ausreichend)

3. Welche Rolle spielt die Grammatik in einer Sprache?

   a Eine Grammatik dient zur Spezifikation der Programmiersprachr für den Benutzer der Sprache, damit dieser weiß, wie ein korrektes Programm geschriebenwerden kann(ANleitung zum Generieren eines Satzes der Sprache).

   b Eine Grammatik dient zur Spezifikation des Übersetzters für diese Programmiersprache(Anleitung zur Konstruktion eines Akzeptors für diese Sprache).

6 Fragen zur Grammatik

1. Geben sie die Definition einer regulären Grammatik an.

$G = (N, T, P, Z)$, Nichtterminale(z.B. E,T,D die Dinge mit denen man die Regeln macht), Terminale = Symbole(z.B. +,-,*),Produktion (Regel), Startsymbol

2. Geben sie die Definition des Begriffes der von einer Grammatik erzeugten Sprache an.

$$L = \{w \in T* | Z \Longrightarrow *w\}$$

   - Kein Plan was das heißen soll steht auch auf Seite 71 im Script

3. Geben Sie die Definition eines endlichen Automaten an.

4. Geben sie die Definition der von diesem Automaten akzeptierten Sprache an.(Welche Art von Sprachen wird vom Automat akzeptiert)

5. Was ist der Zusammenhang zwischen einem endlichen Automaten A, der die von G erzeugte Sprache akzeptiert.

   - Zu jeder regulären Grammatik G gibt es einen endlichen Automaten A, der die von G erzeugte Sprache akzeptiert(Gibt einen Beweis aber kein bock den hinzuschreiben oder zu lernen)

6. Was ist der Zusammenhang zwischen deterministischen und nichtdeterministischen endlichen Automaten?

   - Zu jedem nichtdeterministischen endlichen Automaten gibt es einen deterministischen endlichen Automaten, der die gleiche Sprache akzeptiert.

7. Es wird eine Sprache beschrieben an dieser sollen folgende Aufgaben durch geführt werden

   a Die rechts reguläre Grammatik soll für die Sprache angegeben werden.

   b Für die Grammatik soll der Endliche Automat angegeben werden. (eigentlich dann mit Shift und Reduce)

   c Stellen sie ihren Automaten graphisch dar. Kennzeichnen Sie Start- und Finalzustände.

   d Ist Ihr Automat deterministisch? Falls nein,kennzeichnen Sie nichtdeterministischen Übergänge.

8. Konstruieren Sie (mittels des aus der Vorlesung bekannten Verfahrens) den deterministischen endlichen Automaten, der die von Ihnen definierten C-Bezeichner akzeptiert. Zeichen Sie den resultierenden Automaten(Es ist die Teilmengen Konstruktion)

6.1 Chomsky-Hierarchie

Typ-0, rekursiv aufzählbar

Typ-1, kontextsensitiv

$xAy ::= xwy$

Ist $Z \longrightarrow \epsilon$, dann darf Z nicht auf der rechten Seite einer Regel vorkommen.

Typ-2, kontextfrei

$A \longrightarrow w$

Keller TM

Nichtterminal durch Wort ersetzen entweder rechts oder links regulär

Typ-3, regulär

$A \longrightarrow a$ oder $A \longrightarrow \epsilon$

Nichtterminal weder durch eine Folge an Zeichen ersetzt

rechts-regulär $A \longrightarrow aB$

links-regulär $A \longrightarrow Ba$

6.2 Pumping Lemma

$a^n b^n$ ist das Kammagebirge ..(()).. ist nicht regulär und verletzt das Pumping Lemma
xabz
xababz
xaaabbbz das geht nicht deswegen nicht regulär

Lexikalische Analyse / Scanner-Generator: Rex

Eingabe

- Quelltext = Folge (ASCII) Zeichen

Ausgabe

- Folge von Symbole
- Attribute für Token

Aufgaben

- Zusammenfassung von (ASCII) Zeichen zu Token
- Überlesen von Leerzeichen und Kommentaren
- Konvertierung / Normalisierung
- Berechnung der Token-Attribute

Spezifikationsmethode

- Reguläre Ausdrücke
- Regeln zur Auflösung von Mehrdeutigkeiten???
- Semantische Aktionen (C-Anweisungen)

In der Klausur sollen die Spezifikationen definiert werden.

1. Vordefinerte Regeln
2. Benannte reguläre Ausdrücke
3. Schlüsselwörter sind case-inte siehe Skript Seite 145
4. Ganze Zahl oder Festkommazahl?
5. Bezeichner oder Schlüsselwört?
6. Kommentare Überlesen
7. DEA: Und erkann doch zählen!
8. String-Literale akzeptieren und transformieren

6.3 Benannte reguläre Ausdrücke

Häufig benötigt man einen (komplexen) Ausdruck mehrfach in einer Scannerspezifikation, z. B. **{ 0 -9 }** oder **{a-zA-Z_}**. Anstatt diese nun immer wieder in den Regeln zu notieren, kann man auch sogenannte benannte reguläre Ausdrücke benutzen.
Der Ausdruck wird in **DEFINE** definiert und in **Rule** wird die Benutzung definert. In der Klausur vllt bool

Pattern Matches-Regel

Regel: First Rule Matches

Gibt es mehrere regulären Ausdrücke in einer Scannerspezifikation, welche alle die aktuellen Zeichen $z_0 z_1 z_2 ... z_n$ der Eingabe akzeptieren würden, wird die Regel zur Akzeption ausgewählt, welche textuell zuerst in der Scannerspezifikation steht.

6.4 Teilmengenkonsstruktion (Arbeitslistenalgorithmus)

1. Zeichne eine Tabelle(DEA, Zustände, NEA)
2. Starte mit dem Startsymbol und führe einen neuen Zustand ein.
3. Folge den Produktionen des Startzutands und erzeuge für die erreichbare Zustände durch ein Terminalsymbol im DEA, einen neuen Zustand.
4. Wiederhole die Suche nach erreichbaren Zuständen für alle anderen Zustände des erzeugten DEAs, solange bis sich nichts mehr ändert

Beispiel 54 *(vereinfachte Floatingpoint Konstanten)*

Gegeben ist die reguläre Grammatik G:

$$
\begin{aligned}
G &= (N,T,P,C)\\
T &= \{n,\bullet,+,-,e\}\\
N &= \{C,F,I,X,S,U\}
\end{aligned}
$$

$$
P = \left\{\begin{array}{ll}
C ::= n, & C ::= nF,\\
C ::= \bullet I, & \\
F ::= \bullet I, & F ::= eS,\\
I ::= n, & I ::= nX,\\
X ::= eS, & \\
S ::= n, & S ::= +U,\\
S ::= -U, & \\
U ::= n &
\end{array}\right\}
$$

Gesucht wird der endliche Automat A:

$$
\begin{aligned}
A &= (T,Q,R,q_C,F)\\
T &= \{n,.,+,-,e\}\\
Q &= \{q_C,q_F,q_I,q_X,q_S,q_U,f\}\\
F &= \{f\}
\end{aligned}
$$

$$
R = \left\{\begin{array}{ll}
q_C n \to f, & q_C n \to q_F,\\
q_C . \to q_I, & \\
q_F . \to q_I, & q_F e \to q_S,\\
q_I n \to f, & q_I n \to q_X,\\
q_X e \to q_S & \\
q_S n \to f, & q_S + \to q_U,\\
q_S - \to q_U, & \\
q_U n \to f &
\end{array}\right\}
$$