Übersicht zur IUPAC-Nomenklatur

Deutung der Namen organischer Stoffe gemäß der IUPAC-Nomenklatur von 1993

Ein systematischer Name besteht im allgemeinen aus 3 bis 4 Hauptteilen:
1) Präfix(e): korrespondierend zu den Substituenten (table 5,9)
2) Stammform: (Rule A-1..)
3) Infix(e): Ungesättigtheit anzeigend (
4) Suffix: eine funktionelle Gruppe, die die Natur der Stammform charakterisiert. (table 5,R-C)

Zahlen geben die Position in der Stammform an. Die Nummerierung ...
Auswertungsreihenfolge:
Stammform, Suffix (dementsprechend die Nummerierung), Infix, Präfix (evtl. rekursiv auswerten)

Konstruktion der Namen organischer Stoffe gemäß der IUPAC-Nomenklatur von 1993

Die Konstruktion eines Namens zu einer Strukturformel führt derzeit nicht notwendigerweise zu ein und demselben Namen, aber der erhaltene Name beschreibt eindeutig die chemische Strukturformel.
a) Festlegung der Nomenklaturoperation (substitutive und funktionale? Beschreibung)
b) Bestimmung der funktionellen Gruppe → Suffix
c) Bestimmung der Hauptmolekülkette inklusive aller "nicht-ablösbaren" Präfixe → Stammform
d) (Bennennung der Stammform mit der funktionellen Gruppe)
e) Bestimmung der Infixe/Präfixe
f) Bennennung der detachable substituierbaren Präfixe und Vervollständigung der Nummerierung der Struktur, falls erforderlich.
g) Zusammenfassen der Komponenten in alphabetischer Reihenfolge vor allen Präfixen der Stammform

Sonderregeln:

Tabellen:
Table 5: Suffixe und Präfixe zu funktionellen Gruppen
Table 9: Präfixe zu funktionellen Gruppen
Table 10/[Vollmer]: Rangfolgen funktioneller Gruppen
Table 11: Basic numerical terms (mono, di ...)

Arbeitsanleitung zur Realisierung von Ziel 2 (Parser-Konstruktion):

Nützliche Begriffe:
konextfreie Grammatik, Strukturbaum, (Ableitungsbaum), BNF, Terminal, Nichtterminal, Eingabefolge, Symbolfolge, Operationsfolge, Elementfolge
Der Ableitungsbaum gibt den Wörtern einer Sprache eine Struktur.

Wir werden nun einen zielbezogenen Zerteiler (Parser) konstruieren, indem wir einen rekursiven Algorithmus erstellen, der auf der kontextfreien Grammatik unserer Quellsprache basiert. Die Symbolfolge wird von links nach rechts abgearbeitet (Linksableitung). Obwohl es nicht unbedingt die effizienteste Methode ist, werden wir die ermittelte Struktur in einer Datenstruktur explizit speichern, um eine Schnittstelle zur semantischen Analyse zu erzeugen.

Zu jedem Nichtterminal X (...) der Grammatik wird eine Prozedur erstellt, die alle aus X ableitbaren Elementfolgen akzeptiert. Im Rumpf der Prozedur steht für jede Produktion X ::= $x_1...x_n$ eine Operationsfolge. Die Operationsfolge enthält für jedes Element der rechten Seite der Produktion eine der folgenden Operationen:

- Handelt es sich bei x_i um ein Terminal und x_i ist gleich dem nächsten Symbol der Symbolfolge, wird es akzeptiert.
- Ist x_i ein Nichtterminal, wird die entsprechende Prozedur aufgerufen.

In der Prozedur muss ggf. entschieden werden, welche der möglichen Alternativen auszuwählen ist, wenn für das Nichterminal X mehrere Produktion existieren.

- Dies geschieht meist, indem eine Vorschau auf das nächste Symbol der Symbolfolge durchgeführt wird.
- man kann auch die sogenannte Greibach-Normalform für die Grammatik fordern, d. h. alle Produktionen haben die Form A::=aa', wobei a ein Terminal und a' eine beliebige Folge von Nichtterminalen ist.

Zur Reduzierung des Umfangs des Parsers können die folgenden R-Maßnahmen auf der kf-Grammatik ausgeführt werden.
Wesentlicher ist die Auflösung von Entscheidungskonflikten mit E-Maßnahmen in den Prozeduren. Dies werden wir z. Teil mit systematischen Verfahren, z. Teil intuitiv durchführen, da manche Verfahren sehr komplex werden können.

Maßnahmen:

R: Entfernung (aller) nutzloser Symbole aus der Grammatik (Hopcroft 95ff)
Entfernung von Mehrdeutigkeiten
R: Semantisch bedeutungslose Zeichen werden weggelassen Bsp. "-")
[E: Beseitigung von Linksrekursionen der Form A::= * Av (auch wenn gilt: A::=Bu und B::=*Av)]
E: Produktionen der Form A::= vu|vw werden durch Linksfaktorisierung beseitigt.
E: Eliminierung von Produktionen der Form A::= e (e-Produktion; Hopcroft 97f)
R: Entfernung von Kettenproduktionen A::=B (Hopcroft 98f)
E: Chomsky/Greibach-Normalform (Hopcroft 99ff)

Ziel: LL(1)-Grammatik (enthält keine ε-Produktionen)
Die Implementierung von LL(1)-Zerteilern erfolgt meist durch die Technik des rekursiven Abstiegs oder mit direkt programmierten Kellerautomaten. Tabellengesteuerte LL(1)-Zerteiler sind speichereffizient, aber meist langsamer.

zum Beispiel:
<Infix>::= ε ist eine ε-Produktion. Da keine weitere Produktion mit <Infix> ::= ... existiert, wird die Produktion <Infix>::= ε eliminiert und die Produktion <Formula>::= <Prefix><MainChain><Infix> ersetzt durch <Formula>::=<Prefix><MainChain>

<Prefix>::=... enthält drei mögliche Ableitungen, davon eine ε-Produktion.
Zunächst wird die ε-Produktion beseitigt, indem die Produktion <Prefix>::=ε eliminiert wird und die Produktion <Formula>::=<MainChain> der Grammatik hinzugefügt wird.
Für die verbleibenden Produktionen <Prefix>::= ... gilt es nun sogenannte Entscheidungsmengen zu bestimmen, anhand derer eindeutig eine der möglichen Ableitungen ausgewählt werden kann. Unglücklicherweise beginnen jedoch die Ableitungen mit einem Nichtterminal. Daher ermittelt man die Anfangsmengen Anf(<...>) rekursiv. Also Anf(<Pos>..)=Anf(<Number>|<Number>,<Pos>)=Anf(1|2|...|6)= {1,2,3,4,5,6}
und analog Anf(<Substituent>)= {mono, di, tri, tetra, penta, hexa}

Die obigen Operationen bewirkten, dass für Produktionen der Form **<Formula>::=...** zwei mögliche Ableitungen existieren, so dass hier wiederum die Entscheidungsmengen zu bestimmen sind: Anf(<Prefix>...)=Anf(<Pos>)+Anf(<Substituent>) = {1,2,3,4,5,6, mono, di, tri, tetra, penta, hexa} bzw. Anf(<MainChain>)= {meth, eth, prop, but, pent, hex}

Durch sogenanntes Linksfaktorisieren erhält man aus den Produktionen **<Pos>::= ...** Produktionen, die keine gemeinsamen Anfänge haben.
<Pos>::=<Number><X> und <X>::=,<Number>|e

Bei einer Beseitigung der ε-Produktion folgt:
<Pos>::=<Number><X>|<Number> und <X>::=,<Pos>
Doch dies führt uns zum alten Problem zurück!

Also müssen wir versuchen Entscheidungsmengen für die Produktionen <X>::=... zu gewinnen. Es folgt:
E_1=Anf(,<Number>)={,} und E_2=Anf(ε)=Folge(ε)=Folge(<Pos>)={-}.

Damit sind nun alle Konflikte gelöst und es ergibt sich die folgende LL(1)-Grammatik:

	::=	Entscheidungsmenge
<Formula>	<Prefix><MainChain> | <MainChain>	{1,2, ...,6, mono, di, ..., hexa} {meth, eth, prop, but, pent, hex}
<MainChain>	<SUAC>ane	
<SUAC>	meth | eth | prop | but | pent | hex	
<SUAC-Radical>	<SUAC>yl	
<Prefix>	<Pos> - <Substituent> | <Substituent>	{1,2,3,4,5,6} {mono, di, tri, tetra, penta, hexa}
<Substituent>	<BNT><SUAC-Radical>	
<BNT>	mono, di, tri, tetra, penta, hexa	
<Pos>	<Number><X>	
<X>	,<Number> | ε	{ , } { - }
<Number>	1 | 2 | 3 | 4 | 5 | 6	

SUAC = saturated unbranched-chain compound = gesättigte unverzweigte Kette, BNT = basic numerical term

Diese lässt sich nun leicht in einen rekursiven Algorithmus transferieren.

An jeder beliebigen Stelle der Operationenfolge in einer Prozedur können Aktionen eingefügt werden, die einen Strukturbaum zu der Eingabefolge aufbauen (Postfix- und Präfixaufbau).

Die Überpüfung auf semantische Fehler schieben wir vorerst mal beiseite.

Wie bekomme ich nun den gewünschten Code, sprich: wie durchlaufe ich geschickt den Strukturbaum?
Dazu kann ich mir zunächst grob überlegen, wie meine Ausgabe aufgebaut wird und dies in einem Algorithmus beschreiben.

```
Ermittle Kohlenstoffanzahl der Kette          ← <SUAC> im MainChain-Ast oder
for p:=1 to Kohlenstoffanzahl do                im rechten Sohn von <Substituent>
begin
        Schreibe C
        Überprüfe, ob ein Substituent für p existiert   ← <Pos> im <Prefix>-Ast
                Ja => Schreibe [
                      Ermittle Kohlenstoffanzahl der Kette
                      Schreibe ]
end
```

Letzlich erhält man einen einfachen, die Struktur beschreibenden Code.

Das entscheidende Teilstück eines Programmcodes in VRML könnte dann wie folgt aussehen:

```
....
# Die eigentliche Formel ########
Hydrogen { }
Carbon {
 translation 0 0 -1.145     # Hydrogen + Carbon Radius
 drei Carbon {
  eins Carbon { }
  drei Carbon {
   eins Carbon { }
   drei Carbon {
    drei Carbon {
     eins Carbon { }
     drei Carbon {
      drei Carbon { }
 }}}}}}
 ...
```