4.6.1 Muunnos Chomskyn normaalimuotoon

- Idea:
  1. Poistetaan lähtösymboli produktioiden oikealta puolelta
  2. Poistetaan ϵ-produktiot
  3. Poistetaan yksikköproduktiot $A \to B$
  4. Poistetaan produktiot $A \to X_1X_2...X_k$, $k \geq 2$
- Produktioiden oikealla puolella olevien lähtösymbolien poistaminen
  - Jos lähtösymboli S on jonkin produktion oikealla puolella lisätään uusi lähtösymboli S' ja sille produktio $S' \to S$

ϵ-produktioiden poistaminen

Olkoon $G = (V, \Sigma, P, S)$ kontekstiton kielioppi. Välike $A \in V - \Sigma$ on *tyhjentyvä* (engl. nullable), jos $A \underset{G}{\Rightarrow}^* \epsilon$

Lemma:
Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa enintään lähtösymboli on tyhjentyvä.
(Huom! Lähtösymbolin tyhjentymistä ei voida välttää, jos $\epsilon \in L(G)$.)
Todistus: Olkoon $G = (V, \Sigma, P, S)$.

1. Selvitetään ensin G:n tyhjentyvät välikkeet (NULL-joukko) seuraavasti:
   (i) asetetaan aluksi
   $$\text{NULL} := \{A \in V - \Sigma \mid A \to \epsilon \text{ on } G\text{:n produktio}\};$$
   (ii) toistetaan sitten seuraavaa NULL-joukon laajennusoperaatiota, kunnes joukko ei enää kasva:
   $$\begin{aligned} \text{NULL} \;:=\; & \text{NULL} \cup \\ & \{A \in V - \Sigma \mid A \to B_1 \dots B_k \text{ on } G\text{:n produktio}, \\ & B_i \in \text{NULL kaikilla } i = 1, \dots, k\}. \end{aligned}$$
2. Tämän jälkeen korvataan kukin G:n produktio $A \to X_1 \dots X_k$ kaikkien sellaisten produktioiden joukolla, jotka ovat muotoa
   $$A \to \alpha_1 \dots \alpha_k, \qquad \text{missä} \quad \alpha_i = \begin{cases} X_i, & \text{jos } X_i \notin \text{NULL}; \\ X_i \text{ tai } \epsilon, & \text{jos } X_i \in \text{NULL}. \end{cases}$$

3. Lopuksi poistetaan kaikki muotoa $A \to \epsilon$ olevat produktiot. Jos poistettavana on myös produktio $S \to \epsilon$, otetaan muodostettavaan kielioppiin G' uusi lähtösymboli S' ja sille produktiot $S' \to S$ ja $S' \to \epsilon$. $\square$

Esim. Poistetaan ϵ-produktiot seuraavasta kieliopista:

$$\begin{array}{lcll} S & \to & A \mid B & \\ A & \to & aBa \mid \epsilon & \Rightarrow \quad (\text{NULL} = \{A, B, S\}) \\ B & \to & bAb \mid \epsilon & \end{array}$$

$$\begin{array}{lcll} S & \to & A \mid B \mid \epsilon & \\ A & \to & aBa \mid aa \mid \epsilon & \Rightarrow \\ B & \to & bAb \mid bb \mid \epsilon & \end{array}$$

$$\begin{array}{lcl} S' & \to & S \mid \epsilon \\ S & \to & A \mid B \\ A & \to & aBa \mid aa \\ B & \to & bAb \mid bb \end{array}$$

Yksikköproduktioiden poistaminen

- Produktio muotoa $A \to B$, missä A ja B ovat välikkeitä, on *yksikköproduktio* (engl. unit production)

  Lemma:
  Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti kielioppi G', jossa ei ole yksikköproduktioita.
  Todistus: Olkoon $G = (V, \Sigma, P, S)$.

  1. Selvitetään ensin G:n kunkin välikkeen "yksikköseuraajat" seuraavasti:

     (i) asetetaan aluksi kullekin $A \in V - \Sigma$:

     $$F(A) := \{B \in V - \Sigma \mid A \to B \text{ on } G\text{:n produktio}\};$$

     (ii) toistetaan sitten seuraavia F-joukkojen laajennusoperaatiota, kunnes joukot eivät enää kasva:

     $$F(A) := F(A) \cup \bigcup \{F(B) \mid A \to B \text{ on } G\text{:n produktio}\}.$$

2. Tämän jälkeen poistetaan G:stä kaikki yksikköproduktiot ja lisätään niiden sijaan kaikki mahdolliset produktiot muotoa $A \to \omega$, missä $B \to \omega$ on G:n ei-yksikköproduktio jollakin $B \in F(A)$. $\square$

Esim. Poistetaan yksikköproduktiot edellä saadusta kieliopista

$$\begin{aligned} S' &\to S \mid \epsilon \\ S &\to A \mid B \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb. \end{aligned}$$

Välikkeiden yksikköseuraajat ovat: $F(S') = \{S, A, B\}$, $F(S) = \{A, B\}$, $F(A) = F(B) = \emptyset$. Korvaamalla yksikköproduktiot edellä esitetyllä tavalla saadaan kielioppi

$$\begin{aligned} S' &\to aBa \mid aa \mid bAb \mid bb \mid \epsilon \\ S &\to aBa \mid aa \mid bAb \mid bb \\ A &\to aBa \mid aa \\ B &\to bAb \mid bb \end{aligned}$$

Produktioiden $A \to X_1 \dots X_k$, $k > 2$ poisto

- halutaan poistaa oikealta puolelta merkkijonot, joissa on enemmän kuin kaksi välikesymbolia, ja samalla muutetaan joukossa olevat päätesymbolit sopiviksi välikkeiksi (siis esim. $A \to BbC$, $A \to abcd$ ovat tällaisia "laittomia" sääntöjä)

- Lisätään kielioppiin kutakin päätemerkkiä a varten uusi välike C_a ja sille produktio $C_a \to a$

- Korvataan kussakin muotoa $A \to X_1 \dots X_k$, $k \geq 2$, olevassa produktiossa ensin kaikki päätemerkit em. uusilla välikkeillä, ja sitten koko produktio produktiojoukolla

$$\begin{aligned} A &\to X_1 A_1 \\ A_1 &\to X_2 A_2 \\ &\vdots \\ A_{k-2} &\to X_{k-1} X_k, \end{aligned}$$

  missä $A_1, \dots, A_{k-2}$ ovat jälleen uusia välikkeitä.

  (Tarkkaan ottaen uusi produktiojoukko on siis oikeastaan

$$\begin{aligned} A &\to X_1' A_1 \\ A_1 &\to X_2' A_2 \\ &\vdots \\ A_{k-2} &\to X_{k-1}' X_k', \end{aligned}$$

  missä

$$X_i' = \begin{cases} X_i, & \text{jos } X_i \in V - \Sigma; \\ C_a, & \text{jos } X_i = a \in \Sigma) \end{cases}$$

Lause: Mistä tahansa kontekstittomasta kieliopista G voidaan muodostaa ekvivalentti Chomskyn normaalimuotoinen kielioppi G'.
Todistus: Olkoon $G = (V, \Sigma, P, S)$. Poistetaan ensin G:stä lähtösymboli produktioiden oikealta puolelta, ϵ-produktiot ja yksikköproduktiot em. lemmojen konstruktioilla. Tämän jälkeen kaikki G:n produktiot ovat muotoa $A \to a$ tai $A \to X_1 \dots X_k$, $k \geq 2$ (tai $S \to \epsilon$). Viimeksi mainitut poistetaan ed. konstruktion mukaisesti. □

Huom! Turhat välikkeet ja produktiot voidaan aina poistaa.

Esim. Muunnetaan Chomskyn normaalimuotoon kielioppi

$$\begin{aligned} S &\to aBCd \mid bbb \\ B &\to b \\ C &\to c \end{aligned}$$

Tuloksena saadaan kielioppi

$$\begin{aligned} S &\to C_a S_1^1 \\ S_1^1 &\to B S_2^1 \\ S_2^1 &\to C C_d \\ S &\to C_b S_1^2 \\ S_1^2 &\to C_b C_b \\ B &\to b \\ C &\to c \\ C_a &\to a \\ C_b &\to b \\ (C_c &\to c) \\ C_d &\to d \end{aligned}$$

4.7 *Attribuuttikieliopit

Attribuuttikieliopit ovat kätevä tapa ilmaista, mitä toimintoja jäsennyksen yhteydessä suoritetaan. Ne siis lisäävät puhtaasti syntaktiseen kielioppiin semantiikan (merkityksen). Attribuuttikieliopit ovat tarpeellisia esimerkiksi kääntäjissä, joissa lähdekoodin jäsennys (syntaktisen oikeellisuuden tarkistus) ei riitä, vaan koodi pitäisi myös kääntää suoritettavaksi ohjelmaksi. Ajan puutteen takia emme kuitenkaan käsittele attribuuttikielioppeja kurssillamme, mutta seuraavssa on esitlty niiden idea lyhyesti.

- liitetään kontekstittomiin kielioppeihin kielen semantiikan kuvausta
- esim. lauseke $3 + 2 \times 1$ noudattaa kieliopin G_{expr} syntaksia (ja kuuluu siis G_{expr}:in tuottamaan kieleen) – mutta mitä lauseke merkitsee?
- lausekkeen arvon evaluoiminen edellyttää sopivien attribuuttien ja niiden evaluointisääntöjen liittämistä kielioppiin
- Idea:
  - kieliopin mukaisen jäsennyspuun solmu, jonka nimi on symboli $X \sim$ tietue tyyppiä X.
  - tietuetyyppiin X kuuluvat kentät $\sim X$:n *attribuutit*, merk. $X.s$, $X.t$ jne.
  - kussakin X-tyyppisessä jäsennyspuun solmussa X:n attribuuteista eri *ilmentymät*
  - produktioihin $A \to X_1 \ldots X_k$ liitetään attribuuttien *evaluointisääntöjä*, jotka ilmaisevat miten annetun jäsennyspuun solmun attribuutti-ilmentymien arvot määräytyvät sen vanhempi- ja lapsisolmujen attribuutti-ilmentymien arvoista.
- Huom! Säännöt voivat olla periaatteessa minkälaisia funktioita tahansa, kunhan niiden argumentteina esiintyy vain paikallisesti saatavissa olevaa tietoa.
- ts. produktioon $A \to X_1 \ldots X_k$ liitettävissä säännöissä saa mainita vain symbolien $A, X_1, \ldots, X_k$ attribuutteja

Esim. Liitetään etumerkillisiä kokonaislukuja tuottavaan kontekstittomaan kielioppiin attribuutit ja niiden evaluointisäännöt kieliopin tuottamien lukujen arvojen määrittämiseen.

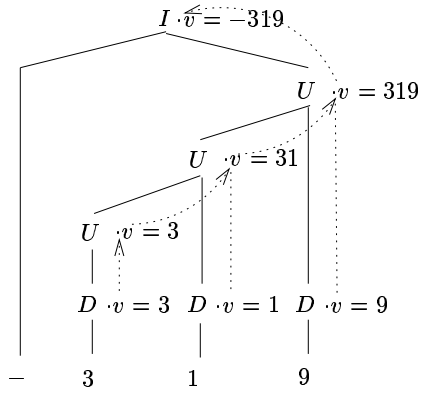
- kuhunkin jäsennyspuun X-tyyppiseen välikesolmuun liitetään attribuutti-ilmentymä $X.v$, jonka arvoksi tulee X:stä tuotetun numerojonon lukuarvo
- juurisolmun v-ilmentymän arvoksi tulee koko puun tuotoksena olevan numerojonon lukuarvo

Produktiot:			*Evaluointisäännöt:*		
I	$\rightarrow$	$+U$	$I.v$	$:=$	$U.v$
I	$\rightarrow$	$-U$	$I.v$	$:=$	$-U.v$
I	$\rightarrow$	U	$I.v$	$:=$	$U.v$
U	$\rightarrow$	D	$U.v$	$:=$	$D.v$
U	$\rightarrow$	UD	$U_1.v$	$:=$	$10 * U_2.v + D.v$
D	$\rightarrow$	0	$D.v$	$:=$	0
D	$\rightarrow$	1	$D.v$	$:=$	1
$\vdots$					
D	$\rightarrow$	9	$D.v$	$:=$	9

- produktion evaluointisäännössä saman välikkeen eri esiintymät voidaan erottaa alaindekseillä
- yllä U_1 vastaa ensimmäistä produktiossa $U \rightarrow UD$ esiintyvää U:ta ja U_2 toista

Evaluointisääntojen mukainen *attributoitu jäsennyspuu* kieliopin tuottamalle lauseelle "-319" (katkoviivoilla on esitetty attribuutti-ilmentymien väliset evaluointiriippuvuudet):

- Attribuuttikieliopin attribuutti t on *synteettinen*, jos sen kuhunkin produktioon $A \rightarrow X_1 \ldots X_k$ liittyvä evaluointisääntö on muotoa $A.t := f(A, X_1, \ldots, X_k)$.
- Tällöin jäsennyspuussa kunkin solmun mahdollisen t-ilmentymän arvo riippuu vain solmun omien ja sen jälkeläisten attribuutti-ilmentymien arvoista
- Muunlaiset attribuutit ovat *periytyviä*
- yllä esim. attribuutti v on synteettinen
- pyritään käyttämään pääasiassa synteettisiä attribuutteja, koska ne voidaan evaluoida helposti yhdellä jäsennyspuun lehdistä juureen suuntautuvalla läpikäynnillä.



Kuva 4.10: Attributoitu jäsennyspuu.

- perittyjä attribuutteja voidaan käyttää, kunhan attribuutti-ilmentymien riippuvuusverkkoihin ei tule syklejä

- Attribuutti-ilmentymien arvot voidaan usein laskea suoraan jäsennysrutiineissa, tarvitsematta muodostaa jäsennyspuuta eksplisiittisesti.

- Esim. ohjelma, joka muuntaa syötteenä annettuja aritmeettisia lausekkeita ns. *postfix*-esitykseen.

  - postfix-esitys: ensin operandit, sitten operaattori; esim. lauseke "$(a + b) * c$" on postfix-muodossa "$ab + c*$" $\Rightarrow$ ei tarvita sulkuja operaatioiden suoritusjärjestyksen ilmaisemiseen
  - kuhunkin välikkeeseen X liittyvän attribuutti-ilmentymän $X.pf$ arvo on X:stä tuotetun alilausekkeen postfix-esitys

Produktiot:			*Evaluointisäännöt:*		
E	$\rightarrow$	$T + E$	$E_1.pf$	$:=$	$(T.pf)\hat{}(E_2.pf)\hat{}('+')$
E	$\rightarrow$	T	$E.pf$	$:=$	$T.pf$
T	$\rightarrow$	$F * T$	$T_1.pf$	$:=$	$(F.pf)\hat{}(T_2.pf)\hat{}('*')$
T	$\rightarrow$	F	$T.pf$	$:=$	$F.pf$
F	$\rightarrow$	a	$F.pf$	$:=$	$'a'$
F	$\rightarrow$	(E)	$F.pf$	$:=$	$E.pf$

Pseudokoodina:

```
char next;
textpf;     /* Tuloksena saatava postfix-esitys */
function E(text pf)
text pf1, pf2;
begin /*E → T + E | T*/
    T(pf1);
    if (next == '+') then
    begin
        next = getnext;
        E(pf2);
        pf = pf1^pf2^'+'
    end;
    else pf = pf1;
end;

function T(text pf)
text pf1, pf2;
begin  T → F * T | F *
    F(pf1);
    if (next == '*') then
    begin
        next = getnext;
        T(pf2);
        pf = pf1^pf2^'*'
    end;
    else pf = pf1;
end;

function F(text pf)
text pf1;
begin /*F → a | (E)*/
    if (next == 'a') then
    begin
        pf = 'a';
        next = getnext
    end;
    else
    if (next == '(') then
    begin
        next = getnext;
        E(pf1);
```

```
        if (next ≠ ')') then ERROR;
        next = getnext;
        pf = pf1
    end;
    elseERROR;
end;

begin /* Pääohjelma */
    next = getnext;
    E(pf);
    printf(pf)
end;
```

4.8 Kontekstittomien kielten ominaisuuksia

4.8.1 Kontekstittomien kielten sulkeumaominaisuudet

Kontekstittomille kielille pätee joitain samantapaisia sulkeumaominaisuuksia kuin säännöllisille kielille.

Lause Olkoon L_1 ja L_2 kontekstittomia kieliä. Tällöin myös

1. $L_1 \bigcup L_2$ (kielten yhdiste)
2. $L_1 L_2$ (kielten katenaatio)
3. $(L_1)^*$ (kielen sulkeuma)
4. $(L_1)^R$ (kielen käänteiskieli)

ovat kontekstittomia.

Huom! Kontekstittomat kielet eivät kuitenkaan ole suljettuja leikkauksen ja komplementin suhteen! Jos L_1 ja L_2 ovat kontekstittomia, ei $L_1 \bigcap L_2$ silti ole välttämättä kontekstiton! Samoin $\overline{L_1} = \Sigma^* \setminus L_1$ (kielen komplementti) ei välttämättä ole kontekstiton.

Toisaalta, jos L on kontekstiton kieli ja R on säännöllinen kieli, niin $L \bigcap R$ on kontekstiton. (Ks. esim. Hopcroft, Motwani, Ullman, teoreema 7.27.)

4.8.2 Ratkeavat ja ratkeamattomat ongelmat

Edellä olemme jo oppineet joitain kontekstittomien kielten ominaisuuksia, joita voidaan ratkaista tietokoneella. Tärkein tällainen ominaisuus koskee jäsentämistä: mille tahansa merkkijonolle x ja annetulle kontekstittomalle kieliopille G voimme ratkaista, kuuluuko x kieleen $L(G)$. Ts. ongelma $x \in L(G)$ on algoritmisesti ratkeava. Mikä jäsennysmenetelmä on kulloinkin paras, riippuu annetusta kielestä. Jos kielioppi voidaan esittää LL(1)-muodossa, tarjoaa tämä tehokkaan (ajassa $O(n)$ toimivan) ja helpon jäsennysmenetelmän. Ohjelmointikielten kääntäjät puolestaan käyttävät LR(k)-jäsentäjiä, kuten Unixin yacc- ja bison-ohjelmien tuottamia jäsentäjiä. Mielivaltaisen kieliopin taas voimme aina muuntaa Chomskyn normaalimuotoon ja jäsentää CYK-algoritmilla ajassa $O(n^3)$.

Eräät tärkeistä kontekstittomien kielten ominaisuuksista ovat kuitenkin ratkeamattomia ongelmia. Tällaisia ongelmia ovat:

- Kahden kieliopin ekvivalenssi ts. $L(G_1) = L(G_2)$?
- Annetun kieliopin moniselitteisyys $Ambiguous(G)$?
- Onko annetun mv. kieliopin kuvaama kieli kontekstiton $Context{-}free(L(G))$?

Viimeiseen mäistä ongelmista liittyy kontekstittomien kielten pumppauslemma, jolla ihminen voi osoittaa joitain kieliä ei-kontekstittomiksi samaan tapaan kuin säännöllisten kielten pumppauslemmalla. Tämäkään pumppauslemma ei kuitenkaan anna kontekstittomuuden välttämättömiä vaan ainoastaan riittävät ehdot, eikä pumppauslemmaa voida soveltaa algoritmisesti. Ongelma pysyy siis yleisessä tapauksessa ratkeamattomana.

4.9 *Kontekstittomien kielten rajoituksista

- Kontekstittomille kielille voidaan todistaa samantapainen pumppauslemma kuin säännöllisillekin kielille
- Erona on vain se, että nyt merkkijonoa (osat uvwxy) on pumpattava samanaikaisesti kahdesta paikasta (osista v ja x)
- *Lemma* [Kontekstittomien kielten pumppauslemma eli uvwxy-lemma]: Olk. L kontekstiton kieli. Tällöin on olemassa sellainen $n \geq 1$, että mikä tahansa $z \in L$, $|z| \geq n$, voidaan jakaa osiin $z = uvwxy$ siten, että

  (i) $|vx| \geq 1$,

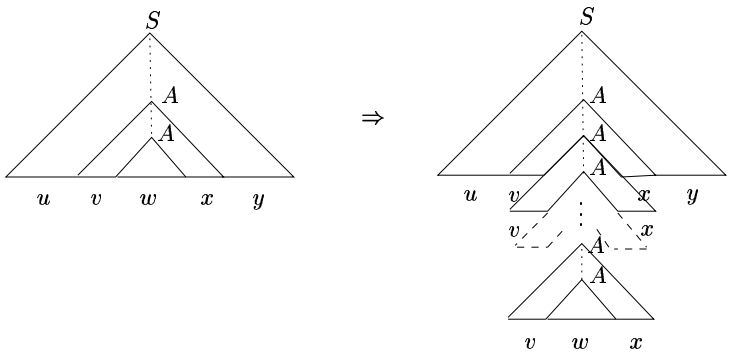
  (ii) $|vwx| \leq n$,

  (iii) $uv^iwx^iy \in L$ kaikilla $i = 0, 1, 2, \ldots$

  Huom: Taas vain äärettömät kielet ovat kiinnostavia

  Todistus:

  - Olk. $G = (V, \Sigma, P, S)$ Chomskyn normaalimuotoinen kielioppi L:lle. Tällöin missä tahansa G:n jäsennyspuussa, jonka korkeus (= pisimmän juuresta lehteen kulkevan polun pituus) on h, on enintään 2^h lehteä. Ts. minkä tahansa merkkijonon $z \in L$ jokaisessa jäsennyspuussa on polku, jonka pituus on vähintään $\log_2 |z|$
  - Olk. $k = |V - \Sigma|$ kieliopin G välikkeiden määrä. Asetetaan $n = 2^{k+1}$. Tarkastellaan jotakin $z \in L$, $|z| \geq n$, ja sen jotakin jäsennyspuuta



Kuva 4.11: Kontekstittoman kielen merkkijonon pumppaus.

- $\Rightarrow$ puussa on polku, jonka pituus on $\geq k+1$. Tarkastellaan tämän polun "alinta" $k+1$:n pituista osaa. Tällä polulla, jossa on $k+1$ välikettä vastaavaa solmua ja välikkeitä on k kappaletta, on siis jonkin välikkeen toistuttava. Olkoon A joku polun toistuva välike (ks. kuva 4.11).
- Merkkijono z voidaan nyt osittaa $z = uvwxy$, missä w on A:n alimmasta ilmentymästä tuotettu osajono ja vwx seuraavaksi ylemmästä A:n ilmentymästä tuotettu osajono; osajonot saadaan johdosta

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy$$

- Koska $S \Rightarrow^* uAy$, $A \Rightarrow^* vAx$ ja $A \Rightarrow^* w$, osajonoja v ja x voidaan "pumpata" w:n ympärillä:

$$S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uv^2Ax^2y \Rightarrow^* \ldots \Rightarrow^*$$

$$uv^iAx^iy \Rightarrow^* uv^iwx^iy$$

- $\Rightarrow uv^iwx^iy \in L$ kaikilla $i = 0, 1, 2, \ldots$
- Koska kielioppi G on Chomskyn normaalimuodossa ja $A \Rightarrow^* vAx$, on oltava $|vx| \geq 1$
- Ylemmästä A:n ilmentymästä alkavan jäsennyspuun polun pituus on enintään $k+1 \Rightarrow$ vastaavan alipuun tuotokselle $|vwx| \leq 2^{k+1} = n$. □

- Esim. Väite: kieli $L = \{a^kb^kc^k \mid k \geq 0\}$ ei ole kontekstiton. Tod. Vastaväite: L on kontekstiton. Valitaan parametri n lemman mukaisesti ja tarkastellaan merkkijonoa $z = a^nb^nc^n \in L$. Tällöin z voidaan jakaa pumpattavaksi osiin

$$z = uvwxy, \qquad |vx| \geq 1, \qquad |vwx| \leq n.$$

Tällöin merkkijono vx ei voi sisältää sekä a:ta, b:tä että c:tä. Merkkijonossa u$v^0wx^0y = uwy$ on siten ylijäämä jotakin merkkiä muihin merkkeihin nähden eli $uwy \notin L$. Ristiriita. □

4.10 Kontekstittomien kielten sovelluksia

Jäsentäjät

Ohjelmointikieli voidaan kuvata kontekstittomana kielioppina, jota voidaan käsitellä jäsentäjällä (*parser*). Jäsentäjä on kääntäjän osa, joka tutkii ohjelman rakenteen ja esittää sen jäsennyspuuna Esim. UNIX tarjoaa komennon `yacc`, joka generoi jäsentäjän annetusta kontekstittomasta kieliopista `yacc`:ille annetaan syötteenä kontekstiton kielioppi (`yacc`-notaatiossa säännön nuoli on korvattu kaksoispisteellä), jonka jokaiseen sääntöön voidaan liittää C-koodina määritelty toiminto. Toiminto suoritetaan aina, kun luodaan vastaava jäsennyspuun solmu. Tyypillisesti toiminto vain luo jäsennyspuun solmun, mutta joissain sovelluksissa jäsennyspuuta ei eksplisiittisesti luoda, vaan toiminto tekee jotain muuta, esim. sisällyttää palan objektikoodia kyseiseen paikkaan.

Esimerkkiksi seuraava kielioppi:

$E \to I|E+E|E*E|(E)$
$I \to a|b|Ia|Ib|I0|I1$

annettaisiin `yacc`:ille muodossa:

Exp	: Id	{...}
	| $Exp'+'Exp$	{...}
	| $Exp'*'Exp$	{...}
	| $'('Exp')'$	{...}
	;	
Id	: 'a'	{...}
	| 'b'	{...}
	| Id 'a'	{...}
	| Id 'b'	{...}
	| Id '0'	{...}
	| Id '1'	{...}
	;	

missä {...} sisältää jonkin toiminnon C-koodina.

Dokumentin rakenteen kuvaus

- ns. "mark-up"-kielet kuten HTML ja XML

- kielen merkkijonot dokumentteja, joissa tiettyjä kielen semantiikkaa kuvaavia merkkejä (tags) — esim. <OL> ja </OL> ∼ järjestetty lista
- HTML: laillisen HTML-dokumentin kuvaus ja dokumentin prosessoinnin ohjaus
- XML: kuvaa tekstin semantiikan — esim.
  <ADDR>Perhospolu 17 C 3, 33000 Kukkamäki<ADDR> kertoo, että osajono on osoite.