UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miha Drole

Sintaksna analiza rahlo kontekstno odvisnih jezikov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Igor Kononenko

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01762/2011
Datum: 05.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MIHA DROLE**

Naslov: **SINTAKSNA ANALIZA RAHLO KONTEKSTNO ODVISNIH JEZIKOV**
PARSING MILDLY CONTEXT-SENSITIVE LANGUAGES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Sintaksna analiza kontekstno odvisnih jezikov je PSPACE-poln problem, zato v praksi ni izvedljiva. Namesto nje se v lingvistiki raziskuje sintaksno analizo rahlo kontekstno odvisnih jezikov (angl. Mildly Context-Sensitive Languages) . Diplomsko delo naj opiše pojem rahlo kontekstno odvisnih jezikov in poda pregled najpomembnejših formalizmov za opis posameznih razredov rahlo kontekstno odvisnih jezikov. Za vsak opisani formalizem naj bo predstavljen tudi pripadajoči algoritem za sintaksno analizo.

Mentor:

prof. dr. Igor Kononenko

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Miha Drole,

z vpisno številko 63050028,

sem avtor diplomskega dela z naslovom:

Sintaksna analiza rahlo kontekstno odvisnih jezikov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Igorja Kononenka
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 05. 07. 2011 Podpis avtorja:

Zahvala

Zahvaljujem se svojemu mentorju, prof. dr. Igorju Kononenku, za pomoč in nasvete pri izdelavi diplomske naloge. Zahvaljujem se tudi pred. dr. Boštjanu Slivniku, ki me je navdušil nad formalnimi jeziki, mi pomagal pri izbiri teme in me usmerjal pri delu.

Kazalo

Povzetek **1**

Abstract **2**

1 Uvod **3**

2 Kontekstno odvisne gramatike **5**

3 Rahlo kontekstno odvisne gramatike **9**

4 Gramatike z vgrajevanjem dreves **11**
4.1 Umestitev v Chomskyjevo hierarhijo 13
4.2 Algoritem 15

5 Gramatike z označbo **21**
5.1 Dokazi ekvivalence 23

6 Linearne indeksirane gramatike **25**
6.1 Dokazi ekvivalence 28
6.2 Algoritem 30

7 Kombinatorične kategorične gramatike **36**
7.1 Dokazi ekvivalence 38
7.2 Algoritem 42

8 Zaključek **50**

A Slovarček pojmov **52**

Literatura **54**

Seznam uporabljenih kratic in simbolov

EPDA zloženi skladovni avtomati (embedded push-down automata)

HG gramatike z označbo (head grammars)

HL jeziki gramatik z označbo (head languages)

LIG linearne indeksirane gramatike (linear indexed grammars)

LIJ linearni indeksirani jeziki (linear indexed languages)

KKG kombinatorične kategorične gramatike (combinatory categorial grammars)

KKJ kombinatorični kategorični jeziki (combinatory categorial languages)

KNG kontekstno neodvisne gramatike (context-free grammars)

KNJ kontekstno neodvisni jeziki (context-free languages)

KOG kontekstno odvisne gramatike (context-sensitive grammars)

KOG kontekstno odvisni jeziki (context-sensitive languages)

RKOG rahlo kontekstno odvisne gramatike (mildly context-sensitive grammars)

TAG gramatike z vgrajevanjem dreves (tree-adjoining grammars)

TAL jeziki gramatik z vgrajevanjem dreves (tree-adjoining languages)

Povzetek

V diplomskem delu obravnavamo štiri formalizme, ki v Chomskyjevi hierarhiji ležijo med kontekstno neodvisnimi gramatikami (KNG) in kontekstno odvisnimi gramatikami (KOG), poimensko gramatike z vgrajevanjem dreves (TAG), gramatike z oznako (HG), linearne indeksirane gramatike (LIG) in kombinatorične kategorične gramatike (KKG). Za vsako obravnavano gramatiko vključimo njen opis z osnovnimi definicijami ter navedemo dokaze njihove umeščenosti. Za TAG opišemo dokaz, da lahko generirajo vse jezike iz množice KNJ in še nekatere, ki jih v KNJ ni. V poglavjih, namenjenih HG, LIG in KKG, povzamemo dokaze za njihovo medsebojno ekvivalenco in ekvivalenco s TAG s pomočjo tranzitivnosti ekvivalence. Vsak dokaz ekvivalence sestoji iz dveh korakov. Za izbrani formalizem najprej povzamemo dokaz, da je mogoče vse jezike, ki jih generira, opisati z nekim drugim formalizmom iz skupine. V drugem koraku dokaza pokažemo, da izbrani formalizem lahko generira vse jezike, ki jih je mogoče opisati z nekim drugim formalizmom iz skupine. Za TAG, LIG in KKG predstavimo tudi algoritme za razpoznavanje besed, ki jim pripadajo. Vsi predstavljeni algoritmi dosegajo časovno zahtevnost reda $O(n^6)$. Za KKG predstavimo tudi algoritem za izgradnjo opisa vseh dreves izpeljav neke vhodne besede.

Ključne besede:

formalni jeziki, kontekstno odvisne gramatike, gramatike z vgrajevanjem dreves, gramatike z oznako, linearne indeksirane gramatike, kombinatorične kategorične gramatike, Chomskyjeva hierarhija

Abstract

Parsing Mildly Context-Sensitive Languages

In this thesis we discuss four formalisms, that lie between context-free grammars (KNG) and context-sensitive grammars (KOG) in the Chomsky hierarchy, namely tree-adjoining grammars (TAG), head grammars (HG), linear indexed grammars (LIG) and combinatory categorial grammars (KKG). For each formalism discussed we include its description with basic definitions and descriptions of proofs of their placement in the hierarchy. For TAG we give the proof that they can generate all the languages that can be generated by KNG and some that cannot. In chapters dedicated to HG, LIG and KKG we describe proofs for their mutual equivalence and equivalence with TAG using the transitivity of equivalence. Each proof of equivalence consists of two steps. For the chosen formalism we first give the proof that the set of languages generated by it can also be generated by another formalism in the group. In the second part of the proof we provide a description of the proof that the chosen formalism can generate all the languages that another formalism in the group can generate. For TAG, LIG and KKG we also present recognition algorithms. All algorithms presented have $O(n^6)$ time complexity. In case of KKG we also present an algorithm that builds a description of all parse trees for an input.

Keywords:

formal languages, context-sensitive grammars, tree-adjoining grammars, head grammars, linear indexed grammars, combinatory categorial grammars

Poglavje 1

Uvod

Osnovni način človekovega komuniciranja poteka s pomočjo jezika. Zaradi svoje pomembne funkcije je pogost predmet raziskav. Znani lingvist Noam Chomsky je v svojih raziskovanjih jezikov definiral hierarhijo gramatik, danes znano pod imenom Chomskyjeva hierarhija, ki sestoji iz regularnih gramatik, kontekstno neodvisnih gramatik (KNG), kontekstno odvisnih gramatik (KOG) ter gramatik tipa 0 ([4]).

Regularne in kontekstno neodvisne gramatike so se izkazale za prešibke, da bi opisovale nekatere lastnosti naravnega jezika. Logična izbira za opisovanje so tako postale kontekstno odvisne gramatike. Zanje velja, da je z njimi mogoče opisati večino lastnosti naravnih jezikov, vendar se je izkazalo, da so zaradi svoje zahtevnosti v praksi neuporabne, saj je odločitveni problem pripadnosti neke besede jeziku kontekstno odvisne gramatike PSPACE poln ([4]).

Raziskave so se usmerile v iskanje drugih formalizmov, ki bi bili močnejši od kontekstno neodvisnih gramatik (in torej primernejši za opisovanje kompleksnejših jezikov) in hkrati še vedno obvladljivi v praksi.

V pričujočem delu so obravnavani štirje izmed formalizmov, ki so plod teh prizadevanj: gramatike z vgrajevanjem dreves (tree-adjoining grammars, TAG, [6]), gramatike z oznako (head grammars, HG, [5]), linearne indeksirane gramatike (linear indexed grammars, LIG, [1, 5]) in kombinatorične kategorične gramatike (combinatory categorial grammars, KKG, [2, 10]). Kljub temu, da se kot formalizmi med seboj bistveno razlikujejo, so raziskave pokazale, da so si po množici jezikov, ki jih lahko generirajo, enakovredne. Za vsak obravnavani formalizem je podan njegov opis z osnovnimi definicijami, dokaz ekvivalence z ostalimi predstavljenimi vrstami gramatik in pri treh tudi algoritem za njegovo obravnavo.

V poglavju 2 na kratko predstavimo KOG in KNG ter prikažemo razmerje

med jeziki, ki jih lahko opišeta. Poglavje 3 je namenjeno osnovni definiciji rahlo kontekstno odvisnih gramatik (RKOG). Poglavje 4 obravnava TAG in zanje prirejen Earleyjev algoritem. Tema poglavja 5 so HG in njihova ekvivalenca s TAG. V poglavju 6 obravnavamo LIG, jih primerjamo s HG in TAG ter predstavimo algoritem zanje. Poglavje 7 je namenjeno KKG, povzetkom dokazov za njihovo ekvivalenco z ostalimi podrobneje obravnavanimi gramatikami in opisu algoritma CYK, prilagojenega za KKG. V poglavju 8 na kratko ponovimo najvažnejše ugotovitve in navedemo smernice za nadaljnje delo.

Poglavje 2

Kontekstno odvisne gramatike

KOG so posplošitev KNG. Njuni definiciji se razlikujeta zgolj v definiciji produkcije - če produkcije KNG na svoji levi strani dovoljujejo zgolj posamezen vmesni simbol, KOG dovoljujejo poljuben izraz z vsaj enim vmesnim simbolom. Od tod tudi izhaja njihovo ime. Takšna definicija produkcij namreč omogoča omejevanje uporabe produkcij glede na umeščenost vmesnega simbola (kontekst), ki ga produkcija slika.

Za lažjo primerjavo kontekstno odvisnih jezikov (KOJ) s kontekstno neodvisnimi jeziki (KNJ) najprej definiramo obe vrsti gramatik in razložimo, kaj jezik ustrezne gramatike sploh je.

Definicija 1. *KNG* je četverka $\langle N, T, P, S \rangle$, kjer je

- N končna množica vmesnih simbolov,
- T končna množica končnih simbolov,
- P končna množica produkcij,
- S začetni simbol, $S \in N$.

Vse produkcije v P so oblike $X \longrightarrow \upsilon$, kjer $X \in N$, $\upsilon \in (N \cup T)^*$.

Definicija 2. *KOG* je četverka $\langle N, T, P, S \rangle$, kjer je

- N končna množica vmesnih simbolov,
- T končna množica končnih simbolov,
- P končna množica produkcij,

- S začetni simbol, $S \in N$.

Vse produkcije v P so oblike $\alpha \longrightarrow \beta$, kjer je $|\beta| \geq |\alpha|$ in $\alpha \neq \epsilon$.

Definicija 3. *Korak izpeljave* v KOG ali KNG $G = \langle N, T, P, S \rangle$ je definiran kot $\alpha\beta\gamma \Longrightarrow \alpha\delta\gamma$, kjer obstaja produkcija $\beta \longrightarrow \delta$ v P. Z $\alpha \Longrightarrow^* \beta$ označimo, da obstaja neko zaporedje korakov izpeljave, ki se prične z α in privede v β.

Definicija 4. *Jezik* KOG ali KNG $L(G)$, $G = \langle N, T, P, S \rangle$, je definiran kot množica vseh besed $w \in T^*$, za katere obstaja takšno zaporedje korakov izpeljave v G, da velja $S \Longrightarrow^* w$.

Produkcije pri KOG so torej precej bolj splošno definirane kot pri KNG. Na svoji levi strani dovoljujejo poljuben niz z vsaj enim vmesnim simbolom, desna stran produkcije pa je vedno vsaj tako dolga kot leva.

Iz definicije KOG sledi tudi, da takšna gramatika ne more opisati prazne besede oz. besede $w = \epsilon$. V kolikor ϵ potrebujemo, med produkcije dodamo novo produkcijo $S \longrightarrow \epsilon$, pri čemer se S nikoli ne pojavi na desni strani produkcije. To preprosto dosežemo tako, da vzamemo gramatiko $G = \langle N, T, P, S \rangle$, ki ji želimo dodati besedo ϵ, nadomestimo začetni simbol S z novim simbolom S' in v P dodamo naslednji produkciji:

- $S' \longrightarrow S$,
- $S' \longrightarrow \epsilon$.

Rezultat je nova gramatika $G' = \langle N \cup \{S'\}, T, P', S' \rangle$, $P' = P \cup \{S' \longrightarrow S, S' \longrightarrow \epsilon\}$.

Za prikaz odnosa med razredoma KNJ in KOJ si najprej pripravimo orodje, ki ga bomo v nadaljevanju uporabljali za dokazovanje, da nekega jezika ni mogoče opisati s pomočjo KNG. Dokaz sledeče leme je dostopen v [4].

Lema 1. *Lema o napihovanju za KNG.* Naj bo L KNJ. Obstaja takšna konstanta n, odvisna zgolj od L, da vsako besedo z v L, $|z| \geq n$ lahko zapišemo kot $z = uvwxy$, pri čemer:

- $|vx| \geq 1$,
- $|vwx| \leq n$,
- za vsak $i \geq 0$ je uv^iwx^iy v L.

Sedaj imamo na voljo vsa potrebna orodja, da dokažemo, da so KNJ strogo vsebovani v KOJ. Dokaz izvedemo v dveh korakih. Najprej dokažemo, da za vsak kontekstno neodvisen jezik velja, da ga je mogoče opisati tudi s KOG, nato pa še, da obstajajo jeziki, ki jih ni mogoče zajeti s KNG, vendar jih lahko opišemo s KOG.

Izrek 1. Razred KNJ je vsebovan v razredu KOJ.

Dokaz. Vsebovanost sledi neposredno iz definicij gramatik, ki te jezike opisujejo. Iz definicije produkcij KOG in KNG sledi, da so KNG le poseben primer KOG. Produkcija oblike $X \longrightarrow Y$ (KNG) je namreč enaka produkciji $\alpha \longrightarrow \beta$ pri $\alpha = X$ in $\beta = Y$. Problematične so zgolj produkcije v KNG, ki imajo na svoji desni strani ϵ, vendar te težave lahko rešimo s pretvorbo v katero izmed normalnih oblik za KNG, ki takšne produkcije odstranijo. □

Izrek 2. Razred KNJ je strogo vsebovan v razredu KOJ.

Dokaz. Za dokaz izreka zadostuje, da poiščemo jezik, ki ni kontekstno neodvisen in ga lahko opišemo s KOG.

Primer takšnega jezika je $L = \{a^n b^n c^n \mid n \geq 1\}$.

Za L najprej pokažemo, da ni kontekstno neodvisen, nato pa še, da ga je mogoče opisati s KOG.

Izrek 3. Jezik $L = \{a^n b^n c^n \mid n \geq 1\}$ ni kontekstno neodvisen.

Dokaz. Izrek dokažemo s pomočjo leme o napihovanju za KNJ, definirane pod 1.

Predpostavimo, da je jezik L kontekstno neodvisen. Po lemi o napihovanju velja, da lahko vsako besedo $w \in L, w \geq n$ zapišemo kot $w = uvwyx$. Sedaj pokažemo, da ne obstaja takšna izbira v in y, da za vsak $i \geq 1$ velja $uv^i wy^i x \in L$.

Očitno je, da i in j ne moreta vsebovati vsak zgolj enega simbola – če želimo, da bo rezultat napihovanja še v jeziku, se mora enako povečati število vseh znakov, na ta način pa lahko povečujemo le dva.

Iz gornje ugotovitve sledi, da mora (vsaj) en izmed v in y vsebovati dva simbola. Vendar napihovanje dveh simbolov ne da pričakovanega rezultata. Tako $(ab)^3 \neq a^3 b^3$, temveč $(ab)^3 = ababab$. Vendar beseda, ki vsebuje del takšne oblike, ne more biti v L.

Za jezik L lema torej ne velja, s čimer smo prišli v protislovje – jezik $L = \{a^n b^n c^n \mid n \geq 1\}$ torej ne more biti kontekstno neodvisen. □

Izrek 4. Jezik $L = \{a^n b^n c^n \mid n \geq 1\}$ je kontekstno odvisen.

Dokaz. Dokaz je konstrukcija KOG, ki jezik L opisuje.
$S \longrightarrow \epsilon$
$S \longrightarrow S_0$
$S_0 \longrightarrow aS_0BC$
$S_0 \longrightarrow aBC$
$CB \longrightarrow HB$
$HB \longrightarrow HC$
$HC \longrightarrow BC$
$aB \longrightarrow ab$
$bB \longrightarrow bb$
$bC \longrightarrow bc$
$cC \longrightarrow cc$

□

Iz dokazanih izrekov 3 in 4 sledi, da je razred KOJ močnejši od razreda KNJ. S tem je ob upoštevanju izreka 1 izrek 2 dokazan. □

Kot zanimivost naj na tem mestu navedemo še nekatere lastnosti KOG. Kot vsakemu razredu v Chomskyjevi hierarhiji tudi temu formalizmu pripada lastna vrsta avtomata, ki lahko generira enak nabor jezikov. Ta avtomat je linearno omejeni avtomat.

Definicija 5. *Linearno omejeni avtomat* je Turingov stroj s prostorsko kompleksnostjo $S(n) = c * n$ oz. (z uporabo teorema o kompresiji, opisanega v [4]) $S(n) = n$.

Linearno omejen avtomat je torej Turingov stroj, ki med svojim delovanjem porabi največ toliko prostora, kolikor ga zaseda vhodna beseda.

Prav tako naj definiramo normalno obliko kontekstno odvisnih gramatik.

Definicija 6. KOG je v *Kurodovi normalni obliki*, če so vse njene produkcije v eni izmed naslednjih oblik:

- $A \longrightarrow a$,
- $A \longrightarrow B$,
- $A \longrightarrow BC$,
- $AB \longrightarrow CD$.

Poglavje 3

Rahlo kontekstno odvisne gramatike

Rahlo kontekstno odvisne gramatike so množica formalizmov, ki po svoji izrazni moči ležijo med kontekstno neodvisnimi in kontekstno odvisnimi gramatikami. Vse jezike, ki jih generira nek formalizem iz rahlo kontekstno odvisnih gramatik, je mogoče opisati tudi z uporabo KOG. Med temi jeziki pa so tudi takšni, ki niso kontekstno neodvisni in jih torej ni mogoče opisati s KNG. Z drugimi besedami so KNJ prava podmnožica rahlo kontekstno odvisnih jezikov, ti pa so prava podmnožica KOJ.

Kdaj nek formalizem spada med rahlo kontekstno odvisne formalizme, določa naslednja definicija:

Definicija 7. *RKOG* so gramatike, za katere velja, da

- so KNJ prava podmnožica jezikov, ki jih opisujejo,
- jezike, ki jih opisujejo, lahko razčlenimo v polinomskem času,
- so monotone,
- imajo omejeno sposobnost upoštevanja odvisnosti (so delno kontekstno odvisne).

V razred RKOG spada več do sedaj odkritih gramatik. V nadaljevanju so opisane štiri izmed njih, ki so si med seboj ekvivalentne po izrazni moči.

Pri presojanju njihove ekvivalence uporabljamo šibko ekvivalenco (8), ki v nasprotju z močno ekvivalenco (9) jemlje v poštev zgolj jezike, ki jih je z gramatiko moč opisati, ne pa tudi izpeljave same.

Definicija 8. Gramatiki G_1 in G_2 sta *šibko ekvivalentni*, če generirata enak jezik ($L(G_1) = L(G_2)$).

Definicija 9. Gramatiki G_1 in G_2 sta *močno ekvivalentni*, če sta šibko ekvivalentni in imata za vsako besedo ekvivalentno tudi drevo izpeljav.

Poglavje 4

Gramatike z vgrajevanjem dreves

Definicija 10. *Meja drevesa* α, $Frontier(\alpha)$ je niz x, ki ga sestavljajo oznake listov drevesa v takšnem vrstnem redu, kot bi jih obiskalo preiskovanje v globino.

Definicija 11. *Začetno drevo* $\alpha \in I$ je drevo, katerega meja sestoji le iz končnih in praznih simbolov ($Frontier(\alpha) \in (\Sigma \cup \{\epsilon\})^+$).

Definicija 12. *Pomožno drevo* $\beta \in A$ je drevo, katerega meja je oblike uBw, kjer sta $u, w \in (\Sigma \cup \{\epsilon\})^+$ (sestojita le iz končnih in praznih simbolov) in je B enak oznaki korena drevesa β. List z oznako B imenujemo *stopalo (foot node)*.

Definicija 13. TAG je peterka $\langle \Sigma, N, I, A, S \rangle$, kjer je

- Σ končna množica končnih simbolov,
- N končna množica vmesnih simbolov,
- I končna množica začetnih dreves (initial trees),
- A končna množica pomožnih dreves (auxiliary trees),
- S začetni simbol, $S \in N$.

Nad vmesnimi simboli v drevesih je definirana operacija vgrajevanja (adjoining). Pomožno drevo $\beta \in A$ lahko vgradimo v drevo γ na vozlišče n, katerega oznaka je enaka oznaki korena pomožnega drevesa. Rezultat vgrajevanja je drevo, ki ga dobimo, če:

1. Poddrevo drevesa γ s korenom n – imenujmo ga δ – odstranimo iz γ in ga shranimo.

2. Na mesto vozlišča n pripnemo pomožno drevo β (vozlišče n je sedaj koren β).

3. V prvem koraku odstranjeno drevo δ pripnemo na stopalo v koraku 2 pripetega drevesa β.

Nad vgrajevanjem dreves so definirane tudi 3 vrste omejitev:

- omejeno vgrajevanje (selective adjunction, $SA\{K\}$) – na to vozlišče lahko vgradimo le drevesa iz množice $K \subseteq A$,

- obvezno vgrajevanje (obligatory adjunction, OA) – nad vozliščem, ki je označeno s to omejitvijo, moramo vedno vršiti vgrajevanje,

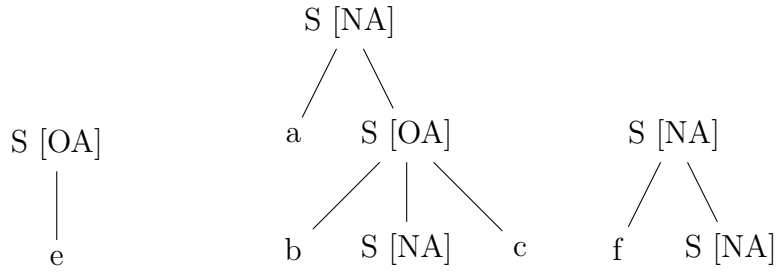
- prepovedano vgrajevanje (null adjunction, NA) je pravzaprav le poseben primer omejenega vgrajevanja $SA\{\}$..

V izvirni definiciji TAG teh omejitev ni, vendar uporaba omejitev OA in NA poveča izrazno moč gramatik ([7]), zato na tem mestu obravnavamo razširjeno obliko TAG. Primer jezika, ki ga ni mogoče opisati brez uporabe omejitev nad vgrajevanjem je $L = a^n f b^n e c^n$. Drevesa TAG, ki ta jezik generira, so prikazana na sliki 4.1.

Sedaj, ko so podane vse potrebne definicije TAG, lahko definiramo tudi izpeljavo in jezik, ki ga neka TAG generira.

Definicija 14. *Korak izpeljave v TAG* $G = \langle \Sigma, N, I, A, S \rangle$ je definiran kot $\alpha \Longrightarrow \beta$, kjer sta α in β drevesi in obstajta neko pomožno drevo $\gamma \in A$ in nek naslov n v drevesu α, da velja, da je γ mogoče vgraditi na lokacijo n drevesa α in je rezultat vgraditve drevesa γ v drevo α na mesto n enak drevesu β. Z $\alpha \Longrightarrow^* \beta$ (α in β sta drevesi) označimo, da obstaja takšno zaporedje korakov izpeljave, ki se prične z drevesom α in privede v β.

Definicija 15. *Jezik TAG* $L(G)$, $G = \langle \Sigma, N, I, A, S \rangle$, je definiran kot množica vseh besed $w \in \Sigma^*$, ki so enake meji nekega drevesa β, ki nad nobenim svojim vozliščem nima omejitve obveznega vgrajevanja in za katerega velja $\alpha \Longrightarrow^* \beta$, kjer je $\alpha \in I$ in je koren α označen z S.



Slika 4.1: TAG, katere jezik je $a^n f b^n e c^n$, ki ga ni mogoče generirati brez uporabe omejitve obveznega vgrajevanja.

4.1 Umestitev v Chomskyjevo hierarhijo

Izrek 5. Kontekstno neodvisni jeziki so prava podmnožica jezikov, ki jih opisujejo TAG ($KNJ \subsetneq TAL$).

Dokaz sestoji iz dveh delov.

Izrek 6. $KNJ \subseteq TAL$.

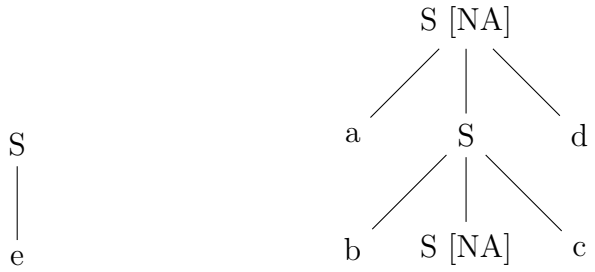
Dokaz. Dokaz je konstrukcija TAG $G' = \langle \Sigma, N, I, A, S \rangle$, ki je ekvivalentna KNG $G = \langle N, T, P, S \rangle$. G mora biti končno dvoumna (torej ne obstaja takšen vmesni simbol B, da zanj velja $B \Longrightarrow^+ B$ in $\epsilon \notin L(G)$.

Vmesne simbole v N ločimo na rekurzivne in nerekurzivne. Simbol X naj bo rekurziven, kadar $\exists \alpha, \beta : X \Longrightarrow^* \alpha X \beta$. Simboli, ki niso rekurzivni, so nerekurzivni.

Rekurzivne simbole (in pravila) identificiramo s pomočjo usmerjenega grafa $\mathscr{G} = \langle V, E \rangle$. Vozlišča grafa naj bodo vmesni simboli ($V = N$), vozlišče A pa naj ima povezavo, usmerjeno v vozlišče B, kadar v P obstaja produkcija oblike $A \longrightarrow \alpha B \beta$. Povezave označimo s produkcijo, iz katere izhajajo.

Problem iskanja rekurzivnih simbolov s tem postane problem iskanja ciklov v usmerjenem grafu (denimo s pomočjo Tarjanovega algoritma) – nek simbol je rekurziven, kadar je njegovo vozlišče del cikla. Podobno je neka produkcija rekurzivna, če obstaja povezava v ciklu, ki nosi njeno oznako. Vse nerekurzivne produkcije naj bodo vključene v množici $NRP \subseteq P$.

S takšno ločitvijo smo dosegli delitev, podobno delitvi TAG na začetna in pomožna. Preostane izgradnja teh dreves. Začetna drevesa ustrezajo vsem drevesnim strukturam, ki jih lahko izpeljemo iz G zgolj z uporabo nerekurzivnih pravil. Pomožna drevesa izpeljemo iz ciklov prej grajenega grafa, ki jih moramo poprej razbiti na osnovne cikle (osnovni cikli ne vsebujejo drugih



Slika 4.2: TAG, katere jezik je $a^n b^n e c^n d^n$.

ciklov) $\mathscr{C} = \{c_0, c_1, ...c_m\}$. Postopek pridobivanja pomožnih dreves iz minimalnih ciklov je sledeč: za vsak cikel $c_j \in \mathscr{C}$, za vsako vozlišče $n_i \in c_j$ velja, da obstajata takšna α_i in β_i, da velja $N_i \Longrightarrow^* \alpha_i N_i \beta_i (\alpha_i \beta_i \neq \epsilon)$, kjer je N_i oznaka vozlišča n_i. Za vse tako dobljene α_i in β_i določimo vsa zaporedja končnih simbolov $u_i, v_i \in \Sigma^*$, v katera se lahko razvijeta z uporabo nerekurzivnih produkcij ($\alpha_i \underset{NRP}{\Longrightarrow^*} u_i, \beta_i \underset{NRP}{\Longrightarrow^*} v_i$). Za vsako izpeljavo $N_i \Longrightarrow \alpha_i N_i \beta_i \underset{NRP}{\Longrightarrow^*} u_i N_i v_i$ sestavimo drevo izpeljav in ga dodamo v množico pomožnih dreves A.

Rezultat tega postopka je TAG G', ki je ekvivalentna KNG G, iz česar sledi $KNJ \subseteq TAL$. □

Izrek 7. $KNJ \subsetneq TAL$

Dokaz. Zadostuje poiskati jezik, ki ni kontekstno neodvisen in ga je mogoče opisati z uporabo TAG. Primer takšnega jezika je $L = \{a^n b^n e c^n d^n\}$.

Kontekstno odvisnost jezika L pokažemo z uporabo leme o napihovanju za KNJ. Podobno kot za jezik $a^n b^n c^n$ ugotovimo, da za L ne obstaja takšna razdelitev $w = uvwyx$, da bi za vsak $i \geq 1$ veljalo $uv^i wy^i x \in L$ (jezik zahteva napihovanje štirih simbolov, vendar lahko napihujemo le dva).

Na 4.2 je prikazana TAG G, katere jezik je L.

□

Jezike, ki jih je mogoče opisati s TAG (in v nadaljevanju opisane po moči ekvivalentne gramatike) sprejemajo zloženi skladovni avtomati (embedded pushdown automata, EPDA). EPDA se vedejo podobno kot navadni skladovni avtomati, le da na vsakem koraku (lahko) ustvarijo nove sklade pod in nad trenutnim skladom. Obnašanje GSA je opisano s prehodi $\langle a, q, s\rangle \longrightarrow \langle q', sb_1, \ldots, sb_m, POP/PUSH(x), sa_1, ..., sa_n, premik\rangle$, kjer je a vhodni simbol, ki je trenutno pod glavo avtomata, q trenutno notranje stanje avtomata, s simbol na trenutnem skladu, q' novo stanje avtomata, $sb_{1..m}$ novoustvarjeni

skladi pod trenutnim skladom, $sa_{1..n}$ novoustvarjeni skladi nad trenutnim skladom, $POP/PUSH(x)$ operacija, ki naj se izvede nad njim, in *premik* logična spremenljivka, ki pove, ali naj se glava nad vhodom premakne en simbol desno. Če $n > 0$ (v prehodu so bili ustvarjeni skladi nad trenutnim), se kot novi trenutni sklad vzame sklad sa_n. Če dosežemo dno trenutnega sklada, preidemo na sklad naravnost pod njim (če ta obstaja). Ti pravili zagotavljata, da je trenutni skladovni simbol vedno vrhnji simbol najvišjega sklada. EPDA sprejme niz, če se sprejemanje zaključi na mestu za zadnjim simbolom s praznim trenutnim skladom, ki pod seboj nima nobenega sklada več.

4.2 Algoritem

Za razpoznavanje in analizo nizov, opisanih z neko TAG, je bilo razvitih več algoritmov, ki izhajajo iz algoritmov nad KNG (Earleyjev algoritem, algoritem CYK). Algoritmi dosegajo časovno kompleksnost reda $O(n^6)$ (nadgradnjo CYK sta razvila Vijay-Shanker in Joshi leta 1985, izpeljava Earleyjevega algoritma je (med drugim) opisana v [6]). Zaradi splošnosti je na tem mestu podrobneje predstavljena razširitev Earleyjevega algoritma, ki za obravnavani TAG postavlja le eno omejitev – vsako drevo mora na svoji meji imeti vsaj en končni simbol, kar pa za večino TAG velja samo po sebi. Če izbrana TAG G ne zadošča temu pogoju, obstaja neka druga TAG G', ki je ekvivalentna G (dobimo jo s konstrukcijo, podobno tisti v dokazu $KNJ \subseteq TAL$) in zadošča temu pogoju.

Algoritem gre čez vhod z leve proti desni in uporablja izgradnjo drevesa od spodaj navzgor (bottom-up) s predikcijo od zgoraj navzdol (top-down).

Najprej določimo način prehoda čez drevo. Drevo prehodimo s pomočjo označevanja – označeno drevo ima v nekem trenutku le eno oznako, ki je določena z vozliščem in pozicijo glede na vozlišče. Vozlišče je lahko označeno na štirih mestih: zgoraj levo (la, left above), spodaj levo (lb, left left), zgoraj desno (ra, right above) in spodaj desno (rb, right below). Preiskovanje drevesa obravnavamo kot premikanje oznake po drevesu z začetno oznako levo nad korenom in zaključkom desno nad njim. Zaporedje prehajanja določajo naslednja pravila:

- la notranjega vozlišča $\longrightarrow$ lb istega vozlišča,
- lb notranjega vozlišča $\longrightarrow$ la najbolj levega potomca,
- la lista $\longrightarrow$ ra istega lista,

- rb vozlišča A $\longrightarrow$ ra vozlišča A,
- ra vozlišča A $\longrightarrow$
  - la desnega brata, če ga ima,
  - sicer rb starša.

Definicija 16. *Gornovo naslavljanje* je način, kako vsakemu vozlišču v drevesu dodeliti naslov. Gornov naslov je enolično določen s pomočjo dveh pravil:

- Gornov naslov korena drevesa je 0,
- Gornov naslov y-tega potomca vozlišča z Gornovim naslovom x je $x \cdot y$.

Gornov naslov vozlišča c na sliki 4.2 bi tako bil $0 \cdot 2 \cdot 3$.

Trenutna oznaka je torej določena z drevesom, vozliščem in lokacijo – trenutno oznako opisuje trojka $\langle \alpha, VOZ, POZ \rangle$, kjer je VOZ Gornov naslov vozlišča (kot je definiran v 16), $\alpha \in A \cup I$, $POZ \in \{la, lb, ra, rb\}$.

Algoritem se izvaja nad množico stanj. Vsako stanje je osmerka

$$\langle \alpha, VOZ, POZ, i, j, k, l, sat \rangle.$$

Prvi trije členi so določeni enako kot prej, i, j, k in l so indeksi v vhodnem nizu (zavzemajo vrednosti do dolžine vhoda), $sat \in \{true, false\}$ pa označuje prisotnost vgrajevanja na trenutnem vozlišču VOZ.

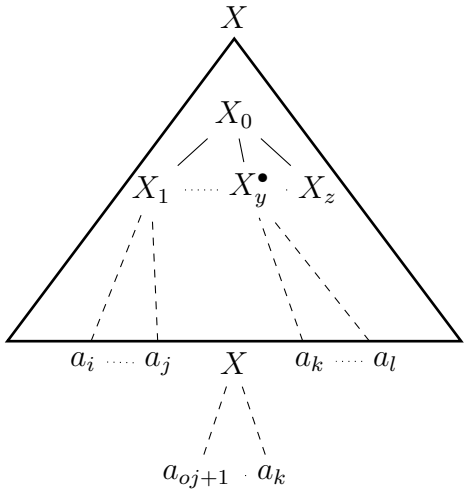
Pomen nekega stanja je prikazan na sliki 4.3.

V grobem je mogoče algoritem razdeliti na 5 korakov:

- inicializacija
- iskanje
- zaključek
- predikcija
- vgrajevanje

Inicializacija: nastavi se začetno stanje algoritma. Inicializira se množico $C = \{\langle \alpha, 0, la, 0, 0, -, -, F \rangle\}$.

Iskanje: je operacija tipa od spodaj navzgor (*bottom-up*), ki preiskuje vhodni niz. Aktivira se, kadar se oznaka nahaja levo nad končnim simbolom. Njeno obnašanje zajema dve možnosti:



Slika 4.3: Primer stanja drevesa α med izvajanjem algoritma. • označuje trenutno označeno mesto.

- označeni simbol ni prazen: oznaka se premakne desno in v S se doda stanje z razširjenim področjem (če je bilo prej področje določeno z i in l, je v novem stanju i, $l + 1$)
- označeni simbol je prazen: oznaka se le premakne v desno

Predikcija: je operacija tipa od vrha navzdol (*top-down*), ki na podlagi levega konteksta predvidi naslednji simbol. Postopek predvidevanja je razdeljen na tri korake:

1. Če je oznaka levo nad vmesnim sibolom, algoritem presodi vsa pomožna drevesa, ki jih je mogoče vgraditi na trenutno označeno vozlišče.
2. Če je oznaka levo nad vmesnim simbolom in nad vozliščem ni omejitve obveznega vgrajevanja, algoritem poskuša razpoznati drevo brez vgrajevanja na tem vozlišču in le premakne oznako pod označeno vozlišče.
3. Ta korak se izvede, če je oznaka desno pod stopalom pomožnega drevesa (A). Algoritem preveri vsa vozlišča, na katera bi bilo mogoče A vgraditi in poskuša prepoznati njihova poddrevesa.

Zaključek: združi dva elementa v C v enega, ki zajema večji del vhoda. Sestoji iz dveh korakov, ki se izvedeta, če je oznaka desno pod vozliščem.

1. Upošteva možnost, da bo naslednji simbol prišel z desne strani stopala pomožnega drevesa, vgrajenega na trenutno označeno vozlišče.

2. Poskuša nadaljevati s prepoznavanjem istega drevesa in združiti meje dveh elementov znotraj njega. Glede na položaj stopala loči dva primera:

   (a) označeno vozlišče vsebuje stopalo x,

   (b) označeno vozlišče ne vsebuje stopala.

Vgrajevanje: korak vgrajevanja združuje elemente z uporabo vgrajevanja. Sestavljeni element zajema večji del vhoda. Rezultatu tega koraka mora algoritem postaviti zastavico *sat* (označiti mora izvedbo vgrajevanja), saj večkratno izvajanje vgrajevanja nad istim vozliščem ni mogoče.

Algoritem 4.1 Earleyjev algoritem, prilagojen za TAG.

Require: $G = \langle \Sigma, N, I, A, S \rangle$ **and** $a = a_1 \dots a_n$
  // Inicializacija
  $C = \{\}$
  $C' = C$
  for all $\alpha \in I$ **do**
    if $\alpha(0) == S$ **then**
      $C = C \cup \{\langle \alpha, 0, la, 0, 0, -, -, \textbf{false} \rangle\}$
    end if
  end for
  while $C \neq C'$ **do**
    $C' = C$
    // Iskanje
    for all $\langle \alpha, voz, la, i, j, k, l, \textbf{false} \rangle \in C$ **do**
      if $\alpha(voz) \in \Sigma$ **and** $\alpha(voz) == a_{l+1}$ **then**
        $C = C \cup \{\langle \alpha, voz, ra, i, j, k, l+1, \textbf{false} \rangle\}$
      else if $\alpha(voz) == \epsilon$ **then**
        $C = C \cup \{\langle \alpha, voz, ra, i, j, k, l, \textbf{false} \rangle\}$
      end if
    end for
    // Predikcija
    for all $\langle \alpha, voz, la, i, j, k, l, \textbf{false} \rangle \in C$ **do**
      if $\alpha(voz) \in N$ **then**
        for all $\beta \in mozne\_vgraditve(\alpha, voz)$ **do**

$C = C \cup \{\langle \beta, 0, la, l, -, l, l, \textbf{false}\rangle\}$
end for
if not $obvezno\_vgrajevanje(\alpha, voz)$ **then**
$C = C \cup \{\langle \alpha, voz, lb, i, j, k, l, \textbf{false}\rangle\}$
end if
end if
end for
for all $\langle \alpha, voz, la, i, j, k, l, \textbf{false}\rangle \in C$ **do**
if $stopalo(\alpha(voz))$ **then**
for all $\delta, voz' : \alpha \in mozne\_vgraditve(\delta, voz')$ **do**
$C = C \cup \{\langle \delta, voz', lb, l, -, -, l, \textbf{false}\rangle\}$
end for
end if
end for
// Complete
for all $\langle \alpha, voz, rb, i, j, k, l, \textbf{false}\rangle \in C$ **do**
for all $\langle \beta, voz', lb, i, -, -, i, \textbf{false}\rangle \in C$ **do**
if $stopalo(\beta(voz'))$ **and** $\beta \in mogoca\_vgrajevanja(\alpha, voz)$ **then**
$C = C \cup \{\beta, voz', rb, i, i, l, l, \textbf{false}\}$
end if
end for
end for
for all $\langle \alpha, voz, rb, i, j, k, l, \textbf{true}\rangle \in C$ **do**
for all $\langle \alpha, voz', la, h, j', k', i, vgr?\rangle \in C$ **do**
if $\alpha(voz) \in N$ **then**
$C = C \cup \{\langle \alpha, voz', ra, h, j/j', k/k', l, vgr?\rangle\}$ // V teh primerih bo vedno določen le en indeks v parih j, j' in k, k' – vrednost j/j' oziroma k/k' nadomestimo z neprazno vrednostjo.
end if
end for
end for
// Vgrajevanje
for all $\langle \beta, 0, ra, i, j, k, l, \textbf{false}\rangle \in C$ **do**
for all $\langle \alpha, voz, rb, j, p, q, k, \textbf{false}\rangle \in C$ **do**
if $\beta \in mogoca\_vgrajevanja(\alpha, voz)$ **then**
$C = C \cup \{\langle \alpha, voz, rb, i, p, q, l, \textbf{true}\rangle\}$
end if
end for
end for

end while

Časovna kompleksnost algoritma izhaja iz koraka vgrajevanja. V njem nastopa 6 spremenljivk, katerih zaloga vrednosti je odvisna od dolžine vhoda (to so spremenljivke i, j, k, l, p, q). Zaloga vrednosti ostalih spremenljivk je odvisna od obravnavene gramatike. Skupna časovna kompleksnost tako znaša $O(|A| * |A \cup I| * max\_vozlisc * n^6)$, v primeru nedvoumne gramatike pa $O(|A| * |A \cup I| * max\_vozlisc * n^4)$.

Poglavje 5

Gramatike z označbo

Gramatike z označbo (Head Grammars, HG) so gramatike, definirane nad zaporedji simbolov (nizi), izmed katerih je en simbol (ali mesto) *označen*, imenujemo ga *glava (head)*. Nad nizi sta definirani dve operaciji: *združevanje (concatenation)* in *ovijanje (wrapping)*.

Definicija 17. *HG* so četvorka $\langle N, T, P, S\rangle$, kjer je

- N končna množica vmesnih simbolov,
- T končna množica končnih simbolov,
- P končna množica produkcij,
- S začetni simbol, $S \in N$.

Produkcije v HG so dveh oblik:

- $A \longrightarrow f(\alpha_1, ..., \alpha_n)$,
- $A \longrightarrow \alpha_1$,

kjer je $A \in N$ in α_i bodisi vmesni simbol bodisi niz elementov množice $T \cup \{\epsilon\}$, v katerem je en simbol (ali mesto) označen.

V uporabi sta dve obliki zapisa HG.

Prva pozna 6 tipov operacij, oblika uporablja označevanje simbolov – simbol a je označen, če je zapisan kot $\bar{a}$:

- $LC1(u_1\bar{a_1}v_1, u_2\bar{a_2}v_2) = u_1\bar{a_1}v_1u_2a_2v_2$,
- $LC2(u_1\bar{a_1}v_1, u_2\bar{a_2}v_2) = u_1a_1v_1u_2\bar{a_2}v_2$,

- $LL1(u_1\bar{a_1}v_1, u_2\bar{a_2}v_2) = u_1\bar{a_1}u_2a_2v_2v_1$,
- $LL2(u_1\bar{a_1}v_1, u_2\bar{a_2}v_2) = u_1a_1u_2\bar{a_2}v_2v_1$,
- $LR1(u_1\bar{a_1}v_1, u_2\bar{a_2}v_2) = u_1u_2a_2v_2\bar{a_1}v_1$,
- $LR2(u_1\bar{a_1}v_1, u_2\bar{a_2}v_2) = u_1u_2\bar{a_2}v_2a_1v_1$.

Številka 1 oz. 2 na koncu imena operacije pove, katera izmed oznak se bo ohranila (vsak niz ima namreč lahko zgolj eno oznako). Operacija LC izvaja združevanje nizov, LL in LR pa ovijanje. Ovijanje ima dve različici, glede na to, ali se drugi niz vrine za (operacija LL) ali pred (operacija LR) označeni simbol prvega niza.

Druga oblika ima le tri različne vrste funkcij (v primerjavi s šestimi v prvi obliki), oznaka pa se vedno nahaja med dvema simboloma. V nizih je označena z ($\uparrow$).

- $C1(w_1 \uparrow u_1, w_2 \uparrow u_2) = w_1 \uparrow u_1w_2u_2$,
- $C2(w_1 \uparrow u_1, w_2 \uparrow u_2) = w_1u_1w_2 \uparrow u_2$,
- $W(w_1 \uparrow u_1, w_2 \uparrow u_2) = w_1w_2 \uparrow u_2u_1$.

C predstavlja združevanje nizov (pomen 1 in 2 je enak kot pri prvi obliki zapisa), W pa ovijanje. Za opis ovijanja zadostuje ena sama operacija, saj je označeno mesto med simboloma in ne na simbolu (tako ne potrebujemo več inačic za vstavljanje pred/po oznaki – označeni prazen prostor po ovijanju ni več prazen, zatorej je v rezultatu vedno označeno mesto drugega niza). Zaradi lažjega zapisa združevanja nizov uvedemo posplošeno obliko zapisa operacije združevanja $C_{i,n}(w_1 \uparrow u_1, \ldots, w_i \uparrow u_i, \ldots, w_n \uparrow u_n) = w_1u_1 \ldots w_i \uparrow u_i \ldots w_nu_n$, ki združi n nizov in ohrani oznako $i$tega.

Za HG definiramo še izpeljavo in jezik, ki ga generira.

Definicija 18. *Izpeljava v HG* $G = \langle N, T, P, S\rangle$ sovpada z zaporedjem uporab produkcij in izvedb operacij. Če je nek vmesni simbol A s končnim številom uporab produkcij iz P mogoče pretvoriti v niz $u \uparrow v$, kjer u, $v \in \{T \cup \{\epsilon\}\}^*$, to zapišemo kot $A \Longrightarrow^* u \uparrow v$.

Definicija 19. *Jezik HG* $L(G)$, $G = \langle N, T, P, S\rangle$, je množica besed $w \in T^*$, za katere obstaja takšna razdelitev $w = uv$, da v G velja $S \Longrightarrow^* u \uparrow v$.

Primer HG, ki generira (rahlo) kontekstno odvisni jezik $L = a^n b^n c^n d^n$ (povzeto po [12]). $G = \langle \{S, T\}, \{a, b, c, d\}, P, S \rangle$, kjer je P:

$$S \longrightarrow \epsilon \uparrow \epsilon$$
$$S \longrightarrow c_{3,1}(a \uparrow \epsilon, T, \epsilon \uparrow d)$$
$$T \longrightarrow w(S, b \uparrow c).$$

5.1 Dokazi ekvivalence

Zaradi temeljnih razlik med TAG in HG je presojanje močne ekvivalence nesmiselno. Sta pa množici TAL in HL šibko ekvivalentni. Šibka ekvivalenca se intuitivno kaže v podobnosti operacij vgrajevanja v TAG in ovijanja v HG – kot pri vgrajevanju gre tudi pri ovijanju za vstavljanje niza v niz z možnostjo rekurzije.

Dokaz šibke ekvivalence s TAG sestoji iz dveh korakov, ki jih na tem mestu zgolj orišemo.

Izrek 8. $HL \supseteq TAL$

Dokaz. Dokaz je konstrukcija, ki za poljubno TAG sestavi ekvivalentno HG.

- Za vsako pomožno drevo β v TAG ustvarimo množico produkcij tako, da vsakemu možnemu drevesu, dobljenemu iz β, z mejo $w_1 X w_2$ (X je stopalo drevesa β), ustreza izpeljava v HG, katere produkt je $w_1 \uparrow w_2$. Vgrajevanje drevesa β oponašamo z uporabo ovijanja.

- Za vsako osnovno drevo (pomožna in začetna drevesa) dodamo produkcije, dobljene po naslednjih korakih:

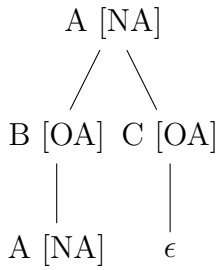
  - za vsako vozlišče v drevesu α z naslovom μ ustvarimo dva vmesna simbola X_μ in Y_μ
  - s pomočjo X_μ izpeljemo nize na meji dreves, izpeljanih iz drevesa s korenom v μ
  - Y_μ združuje nize, izpeljane iz otrok vozlišča (naslovljenih $\mu{\cdot}1, ...\mu{\cdot}n$) μ ($Y_\mu \longrightarrow C_{i,n}(X_{\mu\cdot 1}, ...X_{\mu\cdot n})$)
  - vgrajevanje simuliramo s produkcijama
    $X_\mu \longrightarrow W(X_\eta, Y_\mu)$ in
    $X_\mu \longrightarrow Y_\mu$,
    kjer je η koren nekega pomožnega drevesa (v primerih omejitev ustrezno produkcijo izpustimo – prvo pri prepovedi vgrajevanja, drugo pri obveznem vgrajevanju)

□

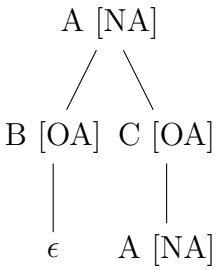
Izrek 9. $TAL \supseteq HL$

Dokaz. Za dokaz vsebovanosti zadostuje, da vsaki operaciji $(C1, C2, W)$ v HG najdemo ekvivalentno strukturo v TAG.

- Združevanje nizov je predstavljeno na slikah 5.1 in 5.2.
- Ovijanje je predstavljeno na sliki 5.3.



Slika 5.1: Drevo, ki simulira operacijo C1(B,C).



Slika 5.2: Drevo, ki simulira operacijo C2(B,C).

Slika 5.3: Drevo, ki simulira operacijo W(B,C).

□

Poglavje 6

Linearne indeksirane gramatike

Linearne indeksirane gramatike so poseben primer indeksiranih gramatik.

Razširitev LIG glede na KNG so skladi indeksov, ki jih lahko pripišemo vmesnim simbolom. Na skladu so lahko zgolj indeksi – simboli iz množice I. Pri LIG vsaka produkcija vpliva na natanko en sklad.

Definicija 20. *LIG* je peterka $\langle N, T, I, S, P \rangle$, kjer je

- N množica vmesnih simbolov,
- T množica končnih simbolov,
- I množica indeksov,
- S začetni simbol ($S \in N$),
- P množica produkcij.

Notacija produkcij se med viri razlikuje v usmerjenosti sklada. V [1] je vrh sklada na desni, v [5] pa na levi. V nadaljevanju se uporablja notacija iz [5].

Produkcije v S so v eni izmed štirih oblik:

1. $A[\cdots] \longrightarrow \alpha_1 B[\cdots]\alpha_2$ (uporaba nad nizom: $\beta A[\delta]\gamma \longrightarrow \beta\alpha_1 B[\delta]\alpha_2\gamma$),
2. $A[\cdots] \longrightarrow \alpha_1 B[i\cdots]\alpha_2$ (uporaba nad nizom: $\beta A[\delta]\gamma \longrightarrow \beta\alpha_1 B[i\delta]\alpha_2\gamma$),
3. $A[i\cdots] \longrightarrow \alpha_1 B[\cdots]\alpha_2$ (uporaba nad nizom: $\beta A[i\delta]\gamma \longrightarrow \beta\alpha_1 B[\delta]\alpha_2\gamma$),
4. $A[] \longrightarrow a$,

kjer:

- $a \in T$,
- $A, B \in N$,
- $i \in I$ simbol na vrhu sklada,
- $\alpha_{1,2} \in (N \cup T)^*$,
- $\delta \in I^*$, predstavlja sklad (ali preostanek sklada, če sledi i),
- $\beta, \gamma \in (N[I^*] \cup T)^*$.

Produkcija tipa 1 en vmesni simbol (A) pretvori v niz končnih simbolov in vmesnih simbolovb, pri čemer enemu izmed vmesnih simbolov (B) dodeli sklad simbola A (δ). Produkcije tipa 2 in 3 so razširitve produkcij tipa 1 – poleg pretvorbe v nov niz in prenos sklada se izvede še ena operacija nad skladom – PUSH pri tipu 2, kjer se poleg prenosa sklada nanj potisne dodaten simbol i, in POP pri tipu 3, kjer se sklad prenese brez svojega vrhnjega indeksa. Potomca, ki prevzame sklad starša, imenujemo *odvisni potomec*.

Produkcijo tipa 1 je mogoče simulirati z uporabo produkcij 2 in 3. Splošno produkcijo te oblike $A[\cdots] \longrightarrow \alpha_1 B[\cdots]\alpha_2$ bi nadomestili z uvedbo posebnega simbola $\pi \in I$ in produkcijama

- $A[\cdots] \longrightarrow \alpha_1 B[\pi \cdots]\alpha_2$,
- $B[\pi \cdots] \longrightarrow B[\cdots]$.

Za lažjo uporabo LIG nekoliko razširimo njihovo notacijo, s čimer pa ne povečamo njihove izrazne moči.

Izrek 10. Z LIG je mogoče poleg prenosa sklada tudi napolniti sklade neodvisnih potomcev z vnaprej določenimi indeksi. Notacija: sklade neodvisnih potomcev produkcije napolnimo z želenimi indeksi, primer: $A[\cdots] \longrightarrow B[\cdots]C[i_0 i_1 \ldots i_n]$.

Dokaz. V linearno indeksiranih gramatikah je mogoče simulirati produkcije, ki poleg prenosa sklada na odvisnega potomca dodajo na sklade drugih potomcev vnaprej določene indekse. Sledi konstrukcija zaporedja produkcij, ki simulirajo produkcijo $A[\cdots] \longrightarrow B[\cdots]C[i_0 i_1 \ldots i_n]$. Za vsak $i_j, 0 \leq j \leq n$ ustvarimo nov vmesni simbol P_j in za vse $1 \leq k \leq n$ produkcijo $P_k[\cdots] \longrightarrow P_{k-1}[i_k \cdots]$. Na koncu med produkcije dodamo še produkcijo $P_0[\cdots] \longrightarrow C[i_0 \cdots]$, s čimer je konstrukcija zaključena. Konstrukcije za primere, kjer poleg prenosa sklada pride tudi do njegove spremembe (prek operacije push ali pop) ali kjer je odvisni potomec na desni, se od gornje razlikujejo zgolj v prvem koraku. □

Izrek 11. S sklada vmesnega simbola na levi strani produkcije LIG je mogoče v enem koraku odstraniti zaporedje simbolov. Notacija: sklad vmesnega simbola na levi strani produkcije napolnimo z želenim zaporedjem indeksov, denimo $A[i_1 i_2 i_3 \cdots] \longrightarrow B[\cdots]$.

Dokaz. Veljavnost razširitve 2 dokažemo s sledečo konstrukcijo. Dana naj bo splošna produkcija v novi notaciji: $A[i_1 i_2 \ldots i_n] \longrightarrow \alpha$, kjer je α kakršnakoli veljavna desna stran produkcije v LIG. Takšno produkcijo je mogoče simulirati z zaporedjem produkcij, dobljenih po sledečem postopku:

1. v množico vmesnih simbolov dodamo simbole P_i; $i = 1 \ldots n-1$
2. v množico produkcij dodamo sledeče produkcije
   - $A[i_1 \cdots] \longrightarrow P_1[\cdots]$,
   - $P_j[i_{j+1} \cdots] \longrightarrow P_{j+1}[\cdots]$; $1 < j < n-1$,
   - $P_{n-1}[i_n \cdots] \longrightarrow \alpha$,

Na ta način dobljena LIG izvede produkcijo le, če je zaporedje simbolov na vrhu sklada enako $i_1 i_2 \ldots i_n$. □

Pred dokazovanjem sorodnosti z drugimi formalizmi definirajmo še izpeljavo in jezik LIG.

Definicija 21. *Korak izpeljave v LIG* $G = \langle N, T, I, S, P \rangle$ je definiran kot

- $\Upsilon A[i\delta]\Upsilon' \Longrightarrow \Upsilon \alpha A[j\delta]\beta\Upsilon$, pri produkcijah iz P oblike

$$A[i \cdots] \longrightarrow \alpha A[j \cdots]\beta;$$

- $\Upsilon A[]\Upsilon' \Longrightarrow \Upsilon a \Upsilon'$, za produkcije iz P oblike

$$A[] \longrightarrow a;$$

kjer $i, j \in \{I \cup \{\epsilon\}\}$, $\delta \in I^*$, $A \in N$, $a \in T$. Z $\alpha \Longrightarrow^* \beta$ zapišemo, da obstaja zaporedje korakov izpeljave, ki prične z α in privede v β.

Definicija 22. *Jezik LIG* $L(G)$, $G = \langle N, T, I, S, P \rangle$, je množica besed $w \in T^*$, za katere velja $S \Longrightarrow^* w$.

6.1 Dokazi ekvivalence

Izrek 12. LIG so šibko ekvivalentne TAG in HG.

Izrek 12 dokažemo prek dokazov dveh drugih izrekov. Najprej pokažemo vsebovanost TAG v LIG, nato pa še vsebovanost LIG v HG. Ker so dokazano TAG in HG šibko ekvivalentne, velja šibka ekvivalenca tudi v odnosu z LIG.

Izrek 13. $TAG \subseteq LIG$

Dokaz. Dokaz s konstrukcijo: Za vsako TAG G obstaja LIG G', ki generira enak jezik ($L(G) = L(G')$).

Podana naj bo TAG $G = \langle \Sigma, N, I, A, S \rangle$. Naj bo Υ množica vseh vozlišč v vseh osnovnih drevesih. Posamezno vozlišče v Υ naj bo enolično označeno z oznako α^g, kjer je α ime drevesa, kateremu pripada vozlišče, in g Gornov naslov vozlišča v drevesu. Zanjo sestavimo LIG $G' = \langle N', T, Id, S', P \rangle$, kjer $N' = \{\alpha^{gt} | \alpha^g \in \Upsilon\} \cup \{\alpha^{gb} | \alpha^g \in \Upsilon\}$ in $Id = \Upsilon$, produkcije pa dobimo s pomočjo pravil:

1. $S'[\cdots] \longrightarrow \alpha^{0t}[\cdots]$ za vsak $\alpha \in I$, katerega koren je označen z S.
2. $\alpha^{gb}[\cdots] \longrightarrow \alpha^{g_1 t}[\cdots] \ldots \alpha^{g_m t}$, kjer je $\alpha \in I \cup A$, vozlišče α^g ni na hrbtenici drevesa α in so vozlišča z naslovi $\alpha^{g_1} \ldots \alpha^{g_m}$ potomci vozlišča α^g.
3. $\beta^{gb}[\cdots] \longrightarrow \beta^{g_1 t} \ldots \beta^{g_k t}[\cdots] \ldots \beta^{g_m t}$, kjer je $\beta \in A$, vozlišči β^g in β^k ležita na hrbtenici drevesa β in so vozlišča $\beta^{g_1} \ldots \beta^{g_m}$ potomci vozlišča β^g.
4. $\alpha^{gt}[\cdots] \longrightarrow \alpha^{gb}[\cdots]$ za vsako vozlišče α^{gt}, ki ni omejeno z obveznim vgrajevanjem.
5. $\alpha^{gt}[\cdots] \longrightarrow \beta^{0t}[\alpha^g \cdots]$ za vsak par α^g, β, kjer $\alpha \in A \cup I, \beta \in A$ in je drevo β mogoče vgraditi na vozlišče α^g.
6. $\beta^{fb}[\alpha^g \cdots] \longrightarrow \alpha^{gb}[\cdots]$, kjer je β^f stopalo drevesa $\beta \in A$ in je β mogoče vgraditi na vozlišče α^g.
7. $\alpha^{gt}[] \longrightarrow a$ za vsako vozlišče $\alpha^g, \alpha \in A \cup I$, ki je označeno z $a \in T \cup \{\epsilon\}$.

Produkcije, dobljene s prvim pravilom, pričnejo izpeljavo in ustrezajo izbiri začetnega drevesa. Drugo pravilo skrbi za prenos vozlišč, ki ne ležijo na hrbtenici. Po dogovoru je odvisni potomec prvi potomec (pravzaprav je v primeru vozlišč, ki ne ležijo na hrbtenici, sklad vedno prazen, vendar LIG ne

dovoljuje produkcij brez odvisnega potomca). Tretje pravilo sestavi prehod po hrbtenici pomožnega drevesa. Četrto pravilo za vsako vozlišče nadaljuje z razpoznavanjem trenutnega drevesa brez vgrajevanja nad tem vozliščem (če to ni obvezno). Peto pravilo nasprotno prične s prepoznavanjem na to vozlišče vgrajenega drevesa (razen, če je vgrajevanje nad tem vozliščem prepovedano). Pri tem na sklad potisne trenutno vozlišče, s čimer si omogoči povratek, ki ga izvede z uporabo pravila 6. Zadnje pravilo skrbi za prepoznavanje vhoda.

Vsakemu vozlišču α^g v N ustrezata dva vmesna simbola v N' – α^{gt} in α^{gb}. Z uporabo dveh simbolov preprečimo večkratno izvajanje vgrajevanja nad enim vozliščem (pravilo, ki ustreza operaciji vgrajevanja, zahteva na levi strani produkcije simbol z oznako t, simboli, nad katerimi je bila ta operacija bodisi že izvedena bodisi ne bo izvedena, pa nosijo oznako b). □

Izrek 14. $HG \supseteq LIG$

Dokaz. Dokaz s konstrukcijo. Za vsako LIG $G = \langle N, T, I, S, P\rangle$ obstaja HG $G' = \langle N', T, S, P'\rangle$, ki je šibko ekvivalentna G. Takšno HG dobimo po sledečem postopku:

Dodatni vmesni simboli v G' so označeni s trojicami (A, B, i), $A, B \in N, i \in I$, kjer

- A predstavlja začetni simbol nadaljnje izpeljave,
- B predstavlja cilj nadaljnje izpeljave (simbol, kjer se bo ta veja izpeljave zaključila),
- i predstavlja simbol na vrhu sklada.

Produkcije v G' dobimo po sledečem postopku:

1. Iz produkcij v G:
   (a) Za vsako produkcijo oblike $A[\cdots] \longrightarrow \alpha_1 A_j[i \cdots]\alpha_2$ v G dodamo v G' produkcijo
   $$(A, C, \epsilon) \longrightarrow C_{j,n}(A_1, \ldots, A_{j-1}, (A_j, C, l), A_{j+1}, \ldots A_n)$$
   za $\forall C \in N$.
   (b) Za vsako produkcijo oblike $A[i \cdots] \longrightarrow \alpha_1 A_j[\cdots]\alpha_2$ v G dodamo v G' produkcijo
   $$(A, C, i) \longrightarrow C_{j,n}(A_1, \ldots, A_{j-1}, (A_j, C, \epsilon), A_{j+1}, \ldots A_n)$$
   za $\forall C \in N$.

(c) Za vsako produkcijo oblike $A[] \longrightarrow a$ v G dodamo v G' produkcijo

$$A \longrightarrow \epsilon \uparrow a.$$

2. Za vsako četvorko A, B, C, i, kjer $A, B, C \in N$ in $i \in I$

   (a) $(A, B, i) \longrightarrow W((A, C, \epsilon), (C, B, i))$.

   (b) $(A, A, \epsilon) \longrightarrow \epsilon \uparrow \epsilon$.

   (c) $A \longrightarrow W((A, B, \epsilon), B)$.

Produkcije v prvi skupini so neposredne preslikave produkcij iz G, drugo skupino pa sestavljajo pomožne produkcije. G' ima poleg pomožnih produkcij še dodatne, dobljene iz pravil 1.a in 1.b (na vsako uporabo pravila 1.a ali 1.b dobimo $|N|$ novih produkcij v G'). Te produkcije služijo različnim možnim potekom nadaljnje izpeljave (za vsak simbol v N pričnemo novo vejo, katere cilj je ta simbol).

V drugi skupini je prva produkcija namenjena simuliranju delovanja sklada – predvidi, da se bo trenutni simbol na vrhu sklada porabil šele kasneje. Druga produkcija predstavlja robni primer trenutne veje izpeljave, ko je njen začetni simbol enak njenemu cilju. Tretja produkcija pa služi pričetku razvijanja simbolov, ki v G niso odvisni potomci. □

6.2 Algoritem

Za izvajanje prilagojenega algoritma CYK za LIG je treba gramatiko pretvoriti v ustrezno normalno obliko, podobno Chomskyjevi pri KNG – to normalno obliko imenujemo Chomskyjeva normalna oblika za LIG. V tej obliki ima lahko vsaka produkcija na svoji desni strani največ dva vmesna simbola ali enega končnega, torej:

Definicija 23. Neka LIG $G = \langle N, T, I, S, P \rangle$ je v Chomskyjevi normalni obliki za LIG, kadar je vsaka produkcija iz P v eni izmed naslednjih oblik:

- $A[] \longrightarrow a$,
- $A[\eta \cdots] \longrightarrow BC[\xi \cdots]$,
- $A[\eta \cdots] \longrightarrow B[\xi \cdots]C$,
- $A[\eta \cdots] \longrightarrow B[\xi \cdots]$,

kjer $\eta, \xi \in I \cup \{\epsilon\}$ (pri vsaki izmed zadnjih treh oblik so možne vse tri operacije – push, pop in navaden prenos sklada).

Izrek 15. Za vsako LIG G obstaja ekvivalentna LIG G' v Chomskyjevi normalni obliki za LIG.

Dokaz. Brez izgube splošnosti lahko predvidevamo, da gramatika G vsebuje le produkcije naslednjih oblik:

- $A[i\cdots] \longrightarrow \beta_1 B[\cdots]\beta_2$,
- $A[\cdots] \longrightarrow \beta_1 B[i\cdots]\beta_2$,
- $A[] \longrightarrow a$,

kjer $\beta = N^*$ (že dokazano je mogoče produkcije, ki zgolj prenesejo sklad, simulirati s produkcijami push in pop, vmesne simbole, ki nastopajo v produkcijah push/pop, pa nadomestimo z vmesnimi simboli $N_a, a \in T$ in produkcijami $N_a[] \longrightarrow a$).

Produkcija POP, oblike $A[i\cdots] \longrightarrow A_1A_2\ldots A_nB[\cdots]C_1C_2\ldots C_m$, se pretvori v zaporedje produkcij:

- $A[\cdots] \longrightarrow A_1[]X_1[\cdots]$,
- $X_1[\cdots] \longrightarrow A_2[]X_2[\cdots]$
  $\cdots$,
- $X_{n-2}[\cdots] \longrightarrow A_{n-1}[]X_{n-1}[\cdots]$,
- $X_{n-1}[\cdots] \longrightarrow A_n[]X_n[\cdots]$,
- $X_n[i\cdots] \longrightarrow B[\cdots]Y_1[]$
- $Y_1[\cdots] \longrightarrow C_1[\cdots]Y_2[]$
  $\cdots$,
- $Y_{m-1}[\cdots] \longrightarrow C_{m-1}[\cdots]C_m[]$.

Produkcija PUSH, oblike $A[\cdots] \longrightarrow A_1A_2\ldots A_nB[i\cdots]C_1C_2\ldots C_m$, se pretvori v zaporedje produkcij:

- $A[\cdots] \longrightarrow A_1[]X_1[\cdots]$,
- $X_1[\cdots] \longrightarrow A_2[]X_2[\cdots]$
  $\cdots$,

- $X_{n-2}[\cdots] \longrightarrow A_{n-1}[]X_{n-1}[\cdots]$,
- $X_{n-1}[\cdots] \longrightarrow A_n[]X_n[\cdots]$,
- $X_n[\cdots] \longrightarrow B[i\cdots]Y_1[]$,
- $Y_1[\cdots] \longrightarrow C_1[\cdots]Y_2[]$
  $\cdots$,
- $Y_{m-1}[\cdots] \longrightarrow C_{m-1}[\cdots]C_m[]$.

□

Vnosi v tabeli CYK so oblike $[A, \eta, i, j|B, p, q]$, kjer

- $A, B \in N$,
- $\eta \in I \cup \{\epsilon\}$,
- $0 \leq i \leq p \leq q \leq j$,

in so lahko mesta B, p, q prazna (označeno z $-$). η predstavlja simbol na vrhu sklada indeksov (če $\eta = \epsilon$, je ta prazen), trojka B, p, q pa služi kot logični kazalec na elemente oblike $[B, \eta', p, q|B', p', q']$, s čimer lahko sestavimo trenutni sklad.

Vsak vnos predstavlja eno izmed dveh možnosti

- $A[\eta] \longrightarrow^* a_{i+1} \ldots a_p B[]a_{q+1} \ldots a_j$, kadar B, p in q niso prazni,
- $A[] \longrightarrow^* a_{i+1} \ldots a_j$, kadar je $\eta = \epsilon$ in $B = p = q = -$.

Vhodna beseda $a_1 \ldots a_n$ pripada gramatiki, če se med delovanjem algoritma zgenerira vnos $[S, -, 0, n|-, -, -]$.

Algoritem 6.1 Razpoznavalnik za LIG.

Require: $G = \langle N, T, I, P, S\rangle$ **and** $a = a_1 \ldots a_n$
  // Inicializacija
  $\mathcal{D} = \{\}$
  $\mathcal{D}' = \mathcal{D}$
  for all $a_j \in a$ **do**
    for all $A; A[] \longrightarrow a_j \in P$ **do**
      $\mathcal{D} = \mathcal{D} \cup \{[A, -, j, j+1|-, -, -]\}$

end for
end for
while $\mathcal{D}' \neq \mathcal{D}$ **do**
$\mathcal{D}' = \mathcal{D}$
// Prenos sklada na desnega / levega potomca
for all $[B, \epsilon, i, k|-, -, -] \in \mathcal{D}$ **do**
for all $[C, \eta, k, j|D, p, q] \in \mathcal{D}$ **do**
for all $A[\cdots] \longrightarrow B[]C[\cdots] \in P$ **do**
$\mathcal{D} = \mathcal{D} \cup \{[A, \eta, i, j|D, p, q]\}$
end for
end for
end for
for all $[B, \epsilon, i, k|-, -, -] \in \mathcal{D}$ **do**
for all $[C, \epsilon, k, j|-, -, -] \in \mathcal{D}$ **do**
for all $A[\cdots] \longrightarrow B[\cdots]C[] \in P$ **do**
$\mathcal{D} = \mathcal{D} \cup \{[A, \eta, i, j|D, p, q]\}$
end for
end for
end for
// Produkcije z enim samim potomcem
for all $[B, \eta, i, j|D, p, q] \in \mathcal{D}$ **do**
for all $A[\cdots] \longrightarrow B[\cdots] \in P$ **do**
$\mathcal{D} = \mathcal{D} \cup \{[A, \eta, i, j|D, p, q]\}$
end for
end for
// Operacija POP z vsemi možnostmi
for all $[B, \epsilon, i, k|-, -, -] \in \mathcal{D}$ **do**
for all $[C, \eta', k, j|D, p, q] \in \mathcal{D}$ **do**
for all $A[\eta \cdots] \longrightarrow B[]C[\cdots] \in P$ **do**
$\mathcal{D} = \mathcal{D} \cup \{[A, \eta, i, j|C, k, j]\}$
end for
end for
end for
for all $[B, \eta', i, k|D, p, q] \in \mathcal{D}$ **do**
for all $[C, \epsilon, k, j|-, -, -] \in \mathcal{D}$ **do**
for all $A[\eta \cdots] \longrightarrow B[\cdots]C[] \in P$ **do**
$\mathcal{D} = \mathcal{D} \cup \{[A, \eta, i, j|B, i, k]\}$
end for
end for

```
    for all A[η···] ⟶ B[···] ∈ P do
      D = D ∪ {[A, η, i, j|B, i, j]}
    end for
  end for
  // Operacija PUSH z vsemi možnostmi // PUSH1
  for all [B, ϵ, i, k|−, −, −] ∈ D do
    for all [C, η, k, j|D, p, q] ∈ D do
      for all [D, η′, p, q|E, r, s] ∈ D do
        for all A[···] ⟶ B[]C[η···] ∈ P do
          D = D ∪ {[A, η′, i, j|E, r, s]}
        end for
      end for
    end for
  end for// PUSH2
  for all [B, η, i, k|D, p, q] ∈ D do
    for all [C, ϵ, k, j|−, −, −] ∈ D do
      for all [D, η′, p, q|E, r, s] ∈ D do
        for all A[···] ⟶ B[η···]C[] ∈ P do
          D = D ∪ {[A, η′, i, j|E, r, s]}
        end for
      end for
    end for
  end for
  for all [B, η, i, j|D, p, q] ∈ D do
    for all [D, η′, p, q|E, r, s] ∈ D do
      for all A[···] ⟶ B[η···] do
        D = D ∪ {[A, η′, i, j|E, r, s]}
      end for
    end for
  end for
end while
if [S, −, 0, n|−, −, −] ∈ D then
  return true
else
  return false
end if
```

Časovna kompleksnost gornjega algoritma je $O(n^7)$, izhaja pa iz predzadnjih dveh korakov (PUSH z dvema potomcema) – od vhoda odvisnih parametrov je v teh dveh korakih namreč sedem (i, k, p, q, j, r, s), od katerih lahko

vsak zavzame n vrednosti.

Vendar je mogoče algoritem pohitriti na časovno kompleksnost $O(n^6)$ z razbitjem najkompleksnejših korakov na dva podkoraka, saj sta notranji zanki neodvisni od indeksa i.

Odsek psevdokode, označen s PUSH1, tako postane:

for all $[C, \eta, k, j | D, p, q] \in \mathcal{D}$ **do**
  for all $[D, \eta', p, q | E, r, s] \in \mathcal{D}$ **do**
    $\mathcal{D} = \mathcal{D} \cup \{[C, \eta', k, j | E, r, s]\}$
  end for
end for
for all $[C, \eta', k, j | E, r, s] \in \mathcal{D}$ **do**
  for all $[B, \epsilon, i, k | -, -, -] \in \mathcal{D}$ **do**
    for all $A[\cdots] \longrightarrow B[]C[\eta \cdots] \in P$ **do**
      $\mathcal{D} = \mathcal{D} \cup \{[A, \eta', i, j | E, r, s]\}$
    end for
  end for
end for

Odsek PUSH2 pa:

for all $[B, \eta, i, k | D, p, q] \in \mathcal{D}$ **do**
  for all $[D, \eta', p, q | E, r, s] \in \mathcal{D}$ **do**
    $\mathcal{D} = \mathcal{D} \cup \{[B, \eta', i, k | E, r, s]\}$
  end for
end for
for all $[B, \eta', i, k | E, r, s] \in \mathcal{D}$ **do**
  for all $[C, \epsilon, k, j | -, -, -] \in \mathcal{D}$ **do**
    for all $A[\cdots] \longrightarrow B[\eta \cdots]C[] \in P$ **do**
      $\mathcal{D} = \mathcal{D} \cup \{[A, \eta', i, j | E, r, s]\}$
    end for
  end for
end for

Z uporabo gornjih zamenjav imamo v vsakem koraku največ šest od dolžine vhoda odvisnih spremenljivk (v prvem in tretjem izmed novo dodanih – i, k, p, q, r, s), s čimer se je časovna kompleksnost algoritma zmanjšala z $O(n^7)$ na $O(n^6)$.

Poglavje 7

Kombinatorične kategorične gramatike

Definicija 24. *KKG* je peterka $\langle N, T, S, f, R\rangle$, kjer je

- N končna množica vmesnih simbolov (*osnovnih kategorij*),
- T končna množica končnih simbolov,
- S začetni simbol, $S \in N$,
- f funkcija, ki slika elemente množice končnih simbolov v končne podmnožice množice *sestavljenih kategorij* $C(N)$ $C(N)$,
- R končna množica *kombinatoričnih pravil*, kjer $C(N)$ sestoji iz elementov, pridobljenih s pomočjo izrazov:
  - $c \in N \Longrightarrow c \in C(N)$,
  - $c_1, c_2 \in N \Longrightarrow c_1/c_2 \in C(N), c_1\backslash c_2 \in C(N)$.

Iz definicije izhaja, da je, če je N neprazna, množica $C(N)$ neskončna, zajema pa vse mogoče sestave atomičnih kategorij, povezanih z operatorjema $\backslash$ in $/$.

Skrajno levo kategorijo sestavljene kategorije imenujemo *ciljna* kategorija.

Izpeljava v KKG poteka z uporabo štirih vrst kombinatoričnih pravil:

1. aplikacija naprej
$$(x/y)y \longrightarrow x,$$

2. aplikacija nazaj
$$y(x\backslash y) \longrightarrow x,$$

3. posplošena kompozicija naprej
$$(x/y)(\ldots(y|_1z_1)|_2\ldots|_nz_n) \longrightarrow (\ldots(x|_1z_1)|_2\ldots|_nz_n); n \geq 0,$$

4. posplošena kompozicija nazaj
$$(\ldots(y|_1z_1)|_2\ldots|_nz_n)(x\backslash y) \longrightarrow (\ldots(x|_1z_1)|_2\ldots|_nz_n); n \geq 0.$$

Intuitivno lahko operatorja \ in / razložimo na primerih $c_1\backslash c_2$ in c_1/c_2 kot funkciji, ki se lahko združita s kategorijo c_2 (argumentom) na svoji desni v primeru operatorja / oz. levi pri \ in vrneta vrednost c_1. Kategorijo c_2 imenujemo *sekundarna* kategorija, $c_1|c_2$ pa *primarna* kategorija.

Posamezno kategorično pravilo je mogoče omejiti na dva načina:

- omejimo začetno (skrajno levo) osnovno kategorijo spremenljivke x – omejitev označimo z nadpisano osnovno kategorijo – primer: x^A označuje vse kategorije, ki jih lahko zapišemo kot $A|z$, $| \in /, \backslash, z \in C(N)$,
- omejimo kategorijo, ki jo lahko zavzame y (namesto y v pravilo vstavimo poljubno kategorijo).

Za definiranje jezika, ki ga generira KKG moramo najprej definirati izpeljavo.

Definicija 25. *Korak izpeljave KKG* $G = \langle N, T, S, f, R\rangle$ je definiran kot:

- $\tau_1 c \tau_2 \Longrightarrow \tau_1 c_1 c_2 \tau_2$, kadar je $c_1c_2 \longrightarrow c$ v R;
- $\tau_1 c \tau_2 \Longrightarrow \tau_1 a \tau_2$, kadar je $c \in f(a)$.

Z $\alpha \Longrightarrow^* \beta$ označimo, da je niz α mogoče po nekem zaporedju korakov izpeljave pretvoriti v β.

Definicija 26. *Jezik KKG* $L(G)$, $G = \langle N, T, S, f, R\rangle$, je množica vseh besed $w \in T^*$, za katere velja $S \Longrightarrow^* w$.

Primer KKG G_1, ki generira (rahlo) kontekstno odvisen jezik $L = \{a^nb^nc^nd^n | n \geq 0\}$:

$$N = \{S, T, A, B, D\}$$
$$T = \{a, b, c, d\}$$

$$f(a) = \{(A/D)\} \qquad f(b) = \{B\} \qquad f(d) = \{D\}$$

$$f(\epsilon) = \{(S/T), T\} \qquad f(c) = \{(T\backslash A/T\backslash B)\}$$

$$R = \left\{ \begin{array}{c} (x^S/T)\,(T\backslash A/T\backslash B) \longrightarrow (x^S\backslash A/T\backslash B) \text{ pravilo 1} \\ (A/D)\,(x^S\backslash A) \longrightarrow (x^S/D) \text{ pravilo 2} \\ (x^S/y)\,y \longrightarrow x^S \text{ pravilo 3} \\ y\,(x^S\backslash y) \longrightarrow x^S \text{ pravilo 4} \end{array} \right\}$$

Izpeljava vhodnega niza *aabbccdd* z uporabo KKG G_1:

$S \Longrightarrow (S/D)\,D$ po pravilu 3
$\Longrightarrow (A/D)\,(S\backslash A)\,D$ po pravilu 2
$\Longrightarrow (A/D)\,(S\backslash A/D)\,D\,D$ po pravilu 3
$\Longrightarrow (A/D)\,(A/D)\,(S\backslash A\backslash A)\,D\,D$ po pravilu 2
$\Longrightarrow (A/D)\,(A/D)\,(S\backslash A\backslash A/T)\,T\,D\,D$ po pravilu 3
$\Longrightarrow (A/D)\,(A/D)\,B\,(S\backslash A\backslash A/T\backslash B)\,T\,D\,D$ po pravilu 4
$\Longrightarrow (A/D)\,(A/D)\,B\,(S\backslash A/T)\,(T\backslash A/T\backslash B)\,T\,D\,D$ po pravilu 1
$\Longrightarrow (A/D)\,(A/D)\,B\,B\,(S\backslash A/T\backslash B)\,(T\backslash A/T\backslash B)\,T\,D\,D$ po pravilu 4
$\Longrightarrow (A/D)\,(A/D)\,B\,B\,(S/T)\,(T\backslash A/T\backslash B)\,(T\backslash A/T\backslash B)\,T\,D\,D$ po pravilu 4

V zadnjem koraku uporabimo inverz funkcije f in namesto kategorij vstavimo ustrezne končne simbole (ali ϵ). Povsod namesto A/D vstavimo a, B se zamenja z b, S/T in T z ϵ, D z d in $(T\backslash A/T\backslash B)$ nadomestimo z c. Rezultat zamenjave je niz *aabbϵccϵdd* oziroma *aabbccdd*.

Izpeljavo je najlažje razumeti v obratni smeri, kjer pričnemo z vhodnim nizem, ga s pomočjo funkcije f pretvorimo v kategorije, ki jih nato preoblikujemo z uporabo pravil iz R.

7.1 Dokazi ekvivalence

KKG so šibko ekvivalentne TAG, LIG in HG. Trditev dokažemo v dveh korakih – najprej pokažemo, da so KKJ vsebovani v LIJ, nato pa še, da so TAL vsebovani v KKJ.

Izrek 16. $KKJ \subseteq LIJ$

Dokaz. Za poljubno KKG $G = \langle N, T, S, f, R\rangle$ je mogoče najti takšno LIG $G' = \langle N, T, I, S, P\rangle$, da velja $L(G) = L(G')$. Za konstrukcijo je bistvena

podobnost med obnašanjem sklada in uporabo kombinatoričnih pravil – obe namreč krajšata le z ene strani (sklad s svojega vrha, kombinatorično pravilo z desne).

Indeksi gramatike G' so: $I = N \cup \{\backslash, /, (,)\}$.

Produkcije P dobimo iz množice kombinatoričnih pravil R in funkcije f.

V vsakem kombinatoričnem pravilu sekundarno kategorijo $A|_1 c_1|_2 c_2 \ldots |_n c_n$ pretvorimo v $A[c_n|_n \ldots c_1|_1]$ (zaradi notacije LIG, kjer je vrh sklada na levi, je vrstni red obrnjen). Primarna kategorija pravila ostane nedefinirana z izjemo svojega skrajno levega in desnega simbola – kategorija $B|_1 d_1 \ldots |_m d_m$ tako postane $B[d_m \cdots]$.

Produkcije, ki slikajo iz vmesnih v končne simbole, pridobimo iz funkcije f po sledečih preslikavah:

- $A \in f(a), a \in T \cup \{\epsilon\} : A[] \longrightarrow a \in P$,
- $A|_1 c_1|_2 \ldots |_n c_n \in f(a), a \in T \cup \{\epsilon\} : A[c_n|_n \ldots |_2 c_1|_1] \longrightarrow a \in P$.

□

Izrek 17. $KKJ \subseteq TAL$

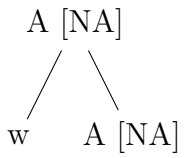
Dokaz. Za vsako TAG $G = \langle \Sigma, N, I, A, S \rangle$ obstaja KKG $G' = \langle N', \Sigma, S, f, R \rangle$, ki generira enak jezik kot G.

Brez dokaza (dokaz je dostopen v [9]) navedimo, da za vsako TAG obstaja šibko ekvivalentna TAG v obliki $G'' = \langle \Sigma, N, \{\alpha\}, \mathcal{A}_1 \cup \mathcal{A}_2, S \rangle$, kjer je drevo α takšno, kot je prikazano na sliki 7.1, množica $\mathcal{A}_1$ sestoji iz dreves oblike na sliki 7.2 in so vsa drevesa v množici $\mathcal{A}_2$ *porezana*. Neko drevo je porezano (prunned), kadar zanj veljajo naslednje trditve:

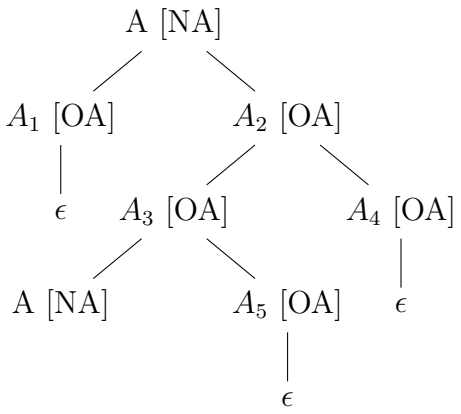
1. je binarno (vsako vozlišče ima največ dva potomca),
2. za vsa vozlišča velja, da je na njih vgrajevanje bodisi prepovedano bodisi obvezno brez dodatnih omejitev (množica dreves, ki jih lahko vgradimo, je enaka množici vseh pomožnih dreves),
3. edini vozlišči s prepovedjo vgrajevanja sta koren in stopalo drevesa (posledično tudi velja, da je vgrajevanje na ostalih vozliščih hrbtenice obvezno),
4. za vsako notranje vozlišče, ki ni na hrbtenici drevesa (ne leži na poti med korenom in stopalom), velja, da:
   - je nad njim zahtevano vgrajevanje brez dodatnih omejitev,

- ima enega samega potomca,
- ta potomec ima oznako ϵ.

Slika 7.1: Drevo α, začetno drevo TAG G'.



Slika 7.2: Pomožno drevo $\beta \in \mathcal{A}_1$.



Slika 7.3: Primer porezanega drevesa (kakršna so v $\mathcal{A}_2$).

Elemente, ki definirajo gramatiko G', dobimo po sledečih postopkih.

Množica osnovnih kategorij N' je prava nadmnožnica množice N, definirana pa je z izrazom $N' = N \cup \{\hat{A} | A \in N\}$.

Za definiranje funkcije f in kombinatoričnih pravil najprej potrebujemo način zapisa dreves iz G kot (sestavljenih) kategorij. V ta namen definiramo

funkcijo $enc(\beta, n)$, ki v ustrezen zapis pretvori vozlišče drevesa β z naslovom n. Naj velja $enc(\beta) = enc(\beta, d_f)$, kjer je d_f naslov stopala drevesa β.

Kodirna funkcija enc je določena na sledeč način: robni primer je koren, kjer ločimo dve možnosti:

- $enc(\beta, 0) = A$, če je nad korenom vgrajevanje prepovedano
- $enc(\beta, 0) = A/\hat{A}$, če je nad korenom vgrajevanje obvezno

Splošni primer, $enc(\beta, d)$, pa obravnavamo s pomočjo sledečih izrazov:

- če $d = d_p1$ in vozlišče d_p2 obstaja (torej, kadar je d levi potomec vozlišča z dvema potomcema) je vrednost $enc(\beta, d)$ enaka
  - $(enc(\beta, d_p)/B)$, če je vozlišče d stopalo drevesa β,
  - sicer $((enc(\beta, d_p)/B)/\hat{A})$,

  kjer je B oznaka vozlišča d_p2;
- če $d = d_p2$ (d je desni izmed potomcev vozlišča), je $enc(\beta, d)$
  - $(enc(\beta, d_p)\backslash B)$, če je d stopalo drevesa β,
  - sicer $((enc(\beta, d_p)\backslash B)/\hat{A}$,

  kjer je B oznaka vozlišča d_p1;
- če pa je d edini potomec vozlišča d_p je
  - $enc(\beta, d) = enc(\beta, d_p)$, če je d stopalo,
  - sicer $enc(\beta, d) = (enc(\beta, d_p)/\hat{a})$;

kjer d leži na hrbtenici drevesa β, d_p je starš vozlišča d, tako da velja bodisi $d = d_p1$ (kadar je levi ali edini potomec) bodisi $d = d_p2$ (kadar je desni potomec), A pa je oznaka vozlišča d v drevesu β.

Funkcijo f, ki slika končne simbole v kategorije, zgradimo s pomočjo dveh pravil:

1. Za vsako drevo β_1 v $\mathcal{A}_1$ vključimo v f preslikavo $f(w) = A$, kjer je A oznaka korena drevesa β in w oznaka njegovega levega potomca ($w \in T \cup \{\epsilon\}$).

2. Za vsako drevo β_2 v $\mathcal{A}_2$ vključimo v f preslikavi $f(\epsilon) = enc(\beta, d_f)$ in $f(\epsilon) = enc(\hat{\beta}, d_f)$, kjer je d_f stopalo drevesa β, $enc(\hat{\beta}, d_f)$ pa je enak $enc(\beta, d_f)$ z izjemo ciljne (skrajno leve) kategorije – če je ciljna kategorija $enc(\beta, d_f)$ kategorija A, je ciljna kategorija $enc(\beta, d_f)$ kategorija $\hat{A}$.

Množica kombinatoričnih pravil R vključuje

- $(x/A)\ A \longrightarrow x$,
- $A\ (x\backslash A) \longrightarrow x$,
- $(x/\hat{A})\ (\ldots(\hat{A}|_1 z_1)|_2 \ldots|_i z_i) \longrightarrow (\ldots(x|_1 z_1)\ldots|_i z_i)$

za vsak $A \in N$ in $0 \leq i \leq k$, kjer k označuje število kategorij, ki nastopajo v najdaljši kategoriji iz zaloge vrednosti funkcije f, zmanjšane za 1 (ciljne kategorije pri štetju ne upoštevamo).

Iz konstrukcije kombinatoričnih pravil je razvidna vloga dodatnih vmesnih simbolov, označenih z ˆ. Ti vmesni simboli označujejo mesta, kjer je mogoča izvedba kompozicije, s pomočjo katere simuliramo vgrajevanje. □

7.2 Algoritem

Algoritem za kombinatorične kategorične gramatike je izpeljan iz algoritma CYK za KNG. Sestoji iz dveh glavnih korakov:

- prepoznavanje vhodnega niza – odloči, ali vhodna beseda $a_1 \ldots a_n$ pripada jeziku $L(G)$ gramatike G,
- v kolikor $a_1 \ldots a_n \in L(G)$, se izvede analiza, ki zgenerira množico dreves izpeljav za dani vhodni niz.

Ker je mogoče, da je različnih izpeljav vhodnega niza eksponentno mnogo, je potreben učinkovit način zapisa vseh možnosti. Eksponentna rast števila različnih izpeljav predstavlja izziv tudi pri prvem delu algoritma.

Analiza pokaže, da je uporaba določenega kombinatoričnega pravila odvisna od

- ciljne kategorije primarne kategorije pravila,
- skrajno desne kategorije primarne kategorije pravila,
- sekundarne kategorije.

Edina izmed gornjih postavk, ki predstavlja težavo pri hranjenju, je sekundarna kategorija. Pri tem nam pomaga zapis s pomočjo *deljenih kategorij*. Osnovna ideja deljenih kategorij izhaja iz same definicije kombinatoričnih pravil. Velja namreč, da se v novonastali kategoriji večji del sekundarne kategorije ponovi. Sekundarno kategorijo tako razstavimo na dva dela – del, ki je zapisan ob kategoriji in ima omejeno velikost (v nadaljevanju označena z b), ter deljeni del, ki je zgolj referenciran.

Treba je torej najti način, kako na učinkovit način hraniti podatke, potrebne za izvedbo pravila. Predstavljeni algoritem, povzet po [2], hrani vnose oblike $((A, \alpha), T)$ v štiridimenzionalni tabeli $L[i, j][p, q]$, kjer je pomen simbolov določen sledeče:

- A je ciljna kategorija,
- α je bodisi celotna pripona kategorije (kadar je dolžina pripone manjša od b) bodisi vrh pripone (zadnji (desni) del pripone, če je pripona daljša od b),
- T je rep kategorije,
- i, j sta indeksa v vhodni besedi,
- p, q sta indeksa deljene kategorije.

Vnos $((A, \alpha), T)$, zapisan v $L[i, j][p, q]$, pomeni, da je iz kategorije $A\beta\alpha$ mogoče izpeljati niz $a_i \dots a_j$, kjer je β pripona vnosa $L[p, q][r, s]$ brez repa T (T je oblike $\backslash, /N$ in predstavlja operator in osnovno kategorijo, ki sta bila uporabljena pri izvedbi operacije in ju torej v končni kategoriji ni).

Pred zapisom algoritma ostaja odprto še vprašanje vrednosti b. Vrednost je neposredno odvisna od gramatike, izračunamo pa jo iz funkcije f in množice kombinatoričnih pravil R. Naj bo

$$b_1 = max(\{n; A|_1 z_1 \dots |_n z_n\}),$$

kjer je $A|_1 z_1 \dots |_n z_n$ sekundarna kategorija nekega pravila iz R in

$$b_2 = max(\{n; f(a) = A|_1 a_1 \dots |_n a_n\}$$

za nek $a \in T \cup \{\epsilon\}$. b je sedaj določen kot

$$b = b_1 + max(b_1, b_2).$$

Algoritem se prične z inicializacijo, kjer za vsako kategorijo $c \in f(a_i)$, $c = A\alpha$ v tabelo L dodamo $L[i,i][0,0] = ((A,\alpha),-)$. Nato iteriramo in dodajamo v tabelo L vnose po sledečih treh pravilih (zapisana pravila veljajo za uporabo kompozicije naprej, vendar je mogoče ustrezna pravila za kompozicijo nazaj iz teh dobiti preprosto z zamenjavo indeksov, zato jih ne obravnavamo posebej).

1. Če $((A,\alpha/B),T) \in L[i,k][p,q]$ in $((B,\beta),-) \in L[k+1,j][0,0]$ velja bodisi, da je T enak $-$ in $|\alpha\beta| < b$ bodisi $T \neq -$ in ena izmed naslednjih možnosti

   - $|\beta| > 1$ in je α prazen,
   - $|\beta| = 1$,
   - β prazen in α neprazen,

   vnesemo v $L[i,j][p,q]$ vnos $((A,\alpha\beta),T)$. V tem primeru je namreč dolžina dobljene kategorije manjša od največje velikosti in ni potrebna uporaba deljenih kategorij.

2. Če $((A,\alpha/B),T) \in L[i,k][p,q]$ in $((B,\beta),-) \in L[k+1,j][0,0]$ in je bodisi T prazen in $|\alpha\beta| \geq b$ bodisi T neprazen in drži, da je število kategorij, ki nastopajo v β ,večje od 1 in α neprazen, vnesemo v $L[i,j][i,k]$ vnos $((A,\beta),/B)$. Dobljena kategorija v tem primeru presega dovoljeno največjo velikost in jo je treba razdeliti. V ta namen ustvarimo povezavo na preostanek kategorije, kot vrh pripone določimo pripono sekundarne kategorije in si zapomnimo rep (pripona nove kategorije je namreč sestavljena iz pripon obeh vnosov, vendar je zadnji del pripone primarne kategorije odveč – pri izvedbi operacije namreč odpade oziroma se nadomesti s pripono sekundarne kategorije).

3. Če $((A,/B),T) \in L[i,k][p,q]$, $((B,\epsilon),-) \in L[k+1,j][0,0]$ in se v tabeli $L[p,q][r,s]$ nahaja $((A,\beta T),T')$ pri čemer je T neprazen, dodamo v $L[i,j][r,s]$ vnos $((A,\beta),T')$. To pravilo se uporabi, kadar se pripona končne kategorije skrajša do te mere, da je za njen zapis potreben dostop do deljene kategorije.

Izvajanje teh pravil ponavljamo, dokler se L ne preneha spreminjati.

Vhodna beseda $a_1 \ldots a_n$ pripada jeziku gramatike G, če velja $((S,\epsilon),-) \in L[1,n][0,0]$.

Zapisani algoritem ima časovno zahtevnost reda $O(n^7)$, kar je razvidno iz tretjega pravila, ki ima sedem prostih indeksov (i,j,k,p,q,r,s), ki lahko zavzamejo n različnih vrednosti.

Podobno kot pri algoritmu za LIG je mogoče tudi na tem mestu tretje pravilo razbiti na dve pravili, kot je to predlagano v [10] – iskanje vnosa $((A, \beta T), T')$ v $L[p, q][r, s]$ je namreč neodvisno od indeksa k, zato ga lahko izvedemo posebej. Tretje pravilo se torej razdeli na

3.a Če $((A, /B), T) \in L[i, k][p, q]$ in $((B, \epsilon), -) \in L[k + 1, j][0, 0]$, je

$$L[i, j][p, q] = ((A, \epsilon), T).$$

3.b Če $((A, \epsilon), T) \in L[i, j][p, q]$ in $((A, \beta T), T') \in L[p, q][r, s]$, je

$$L[i, j][r, s] = ((A, \beta), T').$$

S tem se časovna kompleksnost razpoznavanja zmanjša z $O(n^7)$ na $O(n^6)$ – pravilo z največ od vhoda odvisnimi indeksi je sedaj pravilo 3b s šestimi mesti, ki lahko zavzamejo n različnih vrednosti (i, j, p, q, r, s).

Če je algoritem vhodno besedo prepoznal (vhodna beseda je v jeziku $L(G)$), je treba za ta vhod še izdelati drevo izpeljav. Za učinkovito izvedbo je spet treba najti način, kako zapisati potencialno eksponentno mnogo dreves izpeljav. Vijay-Shanker in Weir v svojem algoritmu za opis izpeljav izdelata linearno indeksirano gramatiko $G_p = \langle \{P\}, T_p, I_p, S_p, P_p \rangle$, katere jezik opisuje vse mogoče izpeljave vhodnega niza z gramatiko G.

Osnovni korak grajenja posameznega drevesa je ugotavljanje, zakaj je posamezen vnos $((a, \alpha), t)$ v tabeli L. Cilj je poiskati kategoriji v L, ki ob združitvi z uporabo ene izmed operacij vrneta kategorijo trenutno izbranega vnosa. Indeksi gramatike I_p so torej četvorka (A, α, i, j). Množico končnih simbolov T_p sestavljajo oznake za korake izpeljave dveh oblik

- $\langle c, a \rangle$ za uporabo funkcije f, kjer je $a \in T$ in $c \in f(a)$,
- F_m označuje kompozicijo naprej, katere sekundarno kategorijo sestavlja m kategorij.

Generiranje P_p je pravzaprav prehod čez vnose tabele L, ki nastopajo v izpeljavah, in invertiranje pravil, ki so bila uporabljena v fazi prepoznavanja vhoda.

Za izbor naslednjega vnosa, ki ga bo algoritem obravnaval, se uporablja sistem označevanja vnosov. Označimo le tista vozlišča, za katera vemo, da nastopajo v izpeljavi vhodnega niza.

V fazi inicializacije označimo vnos $((S, \epsilon), -) \in L[1, n][0, 0]$. Jedro algoritma predstavlja prehod čez vse označene vnose, kjer za vsakega poiščemo

dve kategoriji iz L, ki se ob uporabi nekega kombinatoričnega pravila združita v trenutno izbrano kategorijo. Iz na ta način povezanih kategorij izpeljemo uporabljeno kombinatorično pravilo. Ko je povezava znana, sestavimo produkcijo in jo vstavimo v P_p.

Produkcijo sestavimo s pomočjo naslednjih pravil:

- Če je trenutni vnos $((A, \alpha), T) \in L[i, i][0, 0]$, poiščemo a_i, za katerega je $f(a_i) = A\alpha$, in dodamo v P_p produkcijo

$$P[(A, \alpha, i, i)] \longrightarrow \langle A\alpha, a_i \rangle.$$

- Če je trenutni vnos $((A, \alpha\beta), T)$ iz $L[i, j][p, q]$, poiščemo ustrezna vnosa $((A, \alpha/B), T) \in L[i, k][p, q]$ in $((B, \beta), -) \in L[k + 1, j][0, 0]$ in v P_p vstavimo produkcijo

$$P[(A, \alpha\beta, i, j) \cdots] \rightarrow F_m P[(A, \alpha/B, i, k) \cdots] P[(B, \beta, k + 1, j)].$$

- Če je trenutno vozlišče $((A, \beta), T)$ iz $L[i, j][i, k]$, v L poiščemo vnosa $((A, \alpha/B), T') \in L[i, k][p, q]$ in $((B, \beta), -) \in L[k + 1, j][0, 0]$ in v P_p dodamo

$$P[(A, \beta, i, k)(A, \alpha/B, i, j)] \cdots] \rightarrow F_m P[(A, \alpha/B, i, k) \cdots] P[(B, \beta, k{+}1, j)].$$

- Če je trenutno vozlišče $((A, \alpha), T) \in L[i, j][p, q]$ in najdemo z njim povezana vnosa $((A, /B), T') \in L[i, k][r, s]$, $T' \neq T$ in $((B, \epsilon), -) \in L[k + 1, j][0, 0]$, poiščemo še tretji vnos $((A, \alpha T'), T) \in L[r, s][p, q]$ in dodamo med produkcije

$$P[(A, \alpha, i, j) \cdots] \rightarrow F_0 P[(A, /B, i, k)(A, \alpha T', r, s) \cdots] P[(B, \epsilon, k + 1, j)].$$

Po končanem vstavljanju nove produkcije označimo vse vnose, ki so bili uporabljeni pri njenem določanju in še niso označeni. Izvajanje algoritma se zaključi, ko ni več še neobravnavanih označenih vnosov.

Časovno najzahtevnejše pravilo pri gradnji gramatike za opisovanje dreves izpeljav je zadnje. Ima namreč sedem vrednosti, ki lahko zavzamejo n vrednosti, kar navrže časovno zahtevnost reda $O(n^7)$.

Ker algoritem za izgradnjo dreves izpeljav zgolj v vzvratni smeri išče pravila, uporabljena v fazi prepoznavanja vhoda, je mogoče obe fazi združiti in graditi gramatiko G_p vzporedno s prepoznavanjem vhoda, ne da bi povečali časovno kompleksnost. Je pa vredno omeniti, da na ta način dobljena gramatika lahko vsebuje nepotrebne produkcije, saj vključuje vse vnose iz L –

tudi slepe veje izpeljave, ki ne nastopajo v nobenem izmed dreves izpeljav za vhodni niz.

Algoritem 7.1 Razpoznavalnik vhoda za KKG.

Require: $a = a_1 \dots a_n$, b
  // Inicializacija
  $L = \{\}$
  $L' = L$
  for $i = 1$ **to** n **do**
    for $c \in f(a_i)$ **do**
      $L[i,i][0,0] \leftarrow ((A, \alpha), -)$
    end for
  end for
  while $L \neq L'$ **do**
    $L' = L$
    for all $i, j, k, p, q \in [0, n]$ **do**
      for all $((A, \alpha/B), T) \in L[i,k][p,q]$ **do**
        for all $((B, \beta), -) \in L[k+1, j][0,0]$ **do**
          // Pravilo 1
          if $(T = NULL)$ **or** $(len(\alpha\beta) < b$ **and** $((len(\beta) > 1$ **and** $\alpha = \epsilon)$ **or** $len(\beta) = 1$ **or** $(\beta = \epsilon$ **and** $\alpha \neq \epsilon)))$ **then**
            $L[i,j][p,q] \leftarrow ((A, \alpha\beta), T)$
          end if
          // Pravilo 2
          if $(T = NULL$ **and** $len(\alpha\beta) \geq b)$ **or** $(T \neq NULL$ **and** $len(b) > 1$ **and** $\alpha \neq \epsilon)$ **then**
            $L[i,j][i,k] \leftarrow ((A, \beta)m/B)$
          end if
        end for
      end for
      // Pravilo 3.a
      for all $((A, /B), T) \in L[i,k][p,q]$ **do**
        for all $((B, \epsilon), -) \in L[k+1, j][0,0]$ **do**
          $L[i,j][p,q] \leftarrow ((A, \epsilon), T)$
        end for
      end for
    end for
    // Pravilo 3.b

```
    for all i, j, p, q, r, s ∈ [0, n] do
      for all ((A, ε), T) ∈ L[i, j][p, q] do
        for all ((A, βT), T′) ∈ L[p, q][r, s] do
          L[i, j][r, s] ← ((A, β), T′)
        end for
      end for
    end for
  end while
  if ((S, ε), −) ∈ L[1, n][0, 0] then
    return L
  else
    return NULL
  end if
```

Algoritem 7.2 Algoritem za izgradnjo množice dreves izpeljav KKG.

```
Require: L, a = a_1 ... a_n
  // Inicializacija
  mark(((S, ε), −) ∈ L[1, n][0, 0])
  N_p = {P}
  T_p = init_terminals()
  I_p = init_indices()
  S_p = {S}
  P_p = {}
  while marked\processed ≠ ∅ do
    for all i ∈ [0, n] do
      if marked(((A, α), T) ∈ L[i, i][0, 0]) then
        P_p ← (P[(A, α, i, i)] ⟶ ⟨Aα, a_i⟩)
        I_p ← (A, α, i, i)
        T_p ← ⟨Aα, a_i⟩
      end if
    end for
    for all i, j, p, q ∈ [0, n] do
      if marked(((A, αβ), T) ∈ L[i, j][p, q]) then
        if α ≠ ε and β ≠ ε then
          for all k do
            find_and_mark((A, α/B), T) ∈ L[i, k][p, q]
            find_and_mark((B, β), −) ∈ L[k + 1, j][0, 0]
            P_p ← (P[(A, αβ, i, j) ⋯] ⟶
```

$F_m P[(A, \alpha/B, i, k) \cdots] P[(B, \beta, k+1, j)])$
end for
end if
if $\alpha = \epsilon$ **then**
for all k **do**
$find\_and\_mark(((A, \alpha/B), T') \in L[i, k][p, q])$
$find\_and\_mark(((B, \beta), -) \in L[k+1, j][0, 0])$
$P_p \leftarrow (P[(A, \beta, i, k)(A, \alpha/B, i, j) \cdots] \longrightarrow$
$F_m P[(A, \alpha/B, i, k) \cdots] P[(B, \beta, k+1, j)])$
end for
end if
if $\beta = \epsilon$ **then**
for all k **do**
$find\_and\_mark(((A, /B), T') \in L[i, k][r, s])$
$find\_and\_mark(((B, \epsilon), -) \in L[k+1, j][0, 0])$
$find\_and\_mark(((A, \alpha T'), T) \in L[r, s][p, q])$
$P_p \leftarrow (P[(A, \alpha, i, j) \cdots] \longrightarrow$
$F_0 P[(A, /B, i, k)(A, \alpha T', r, s) \cdots]) P[(B, \epsilon, k+1, j)])$
end for
end if
end if
$mark\_as\_processed(((A, \alpha\beta), T) \in L[i, j][p, q])$
end for
end while
return $\langle N_p, T_p, I_p, S_p, P_p \rangle$

Poglavje 8

Zaključek

S področja rahlo kontekstno odvisnih formalizmov smo obravnavali štiri formalizme, za katere smo pokazali, da so si med seboj ekvivalentni tako po izrazni moči kot po kompleksnosti – vsi predstavljeni algoritmi imajo časovno zahtevnost $O(n^6)$. Izbor ustrezne gramatike je tako odvisen predvsem od zastavljenega problema, zato uporabimo tisti formalizem, v katerega ga najlažje prevedemo.

V praksi je najpogosteje zaslediti gramatike TAG in KKG. Prvenstveno se oboje pojavlja predvsem na področju obravnave struktur v naravnem jeziku (Xtag project[1], ccgbank). KKG so bile denimo uporabljene pri izločanju informacij iz biomedicinske literature ([8]), za TAG pa najdemo primere uporabe tudi izven področja lingvistike – na primer pri modeliranju in napovedovanju sekundarne strukture RNK ([11]).

Vendar kljub temu, da je problem razpoznavanja vhodnih besed pri RKOG časovno bistveno manj zahteven kot pri KOG, je zahtevnost reda $O(n^6)$ za prakso še vedno pogosto previsoka.

Na področju formalizmov, ki po svoji izrazni moči ležijo med kontekstno odvisnimi in kontekstno neodvisnimi gramatikami, bi lahko obravnavali še mnoge bolj ali manj podobne formalizme.

Zgolj med formalizmi, ki so tesno povezani z rahlo kontekstno odvisnimi gramatikami, bi lahko poleg obravnavanih omenili tudi nekatere razširitve gramatik z vgrajevanjem dreves – denimo feature structure based TAG ([5]) in gramatike z vgrajevanjem večkomponentnih dreves (multi-component tree adjoining grammars, [5]). Med formalizme, povezane z rahlo kontekstno odvisnimi gramatikami, spadajo tudi linearni kontekstno neodvisni prepisovalni sistemi (Linear Context-Free Rewriting Systems, [5]).

[1] http://www.cis.upenn.edu/~xtag/

Z razširitvijo področja obravnave tudi izven rahlo kontekstnih gramatik k močnejšim formalizmom naletimo na indeksirane gramatike ([4]) in njihove izpeljanke (sekvenčno indeksirane gramatike ([3]) ...) ter druge.

Prav tako bi se lahko tudi pri drugih gramatikah dotaknili poskusov zapisa vseh mogočih dreves izpeljav, kot smo to storili v primeru KKG. Še nedotaknjeno je ostalo tudi vprašanje parelelizacije obravnavanih algoritmov.

Dodatek A

Slovarček pojmov

adjoining vgrajevanje

auxiliary tree pomožno drevo

combinatory categorial grammar kombinatorična kategorična gramatika

concatenation združevanje

dependant descendant odvisni potomec

embedded push-down automaton zloženi skladovni avtomat

foot stopalo

head grammar gramatika z oznako

initial tree začetno drevo

linear indexed grammar linearna indeksirana gramatika

mildly context-sensitive grammar rahlo kontekstno odvisna gramatika

parse tree drevo izpeljav

prunned tree porezano drevo

root koren

sequentially indexed grammars sekvenčno indeksirane gramatike

spine hrbtenica

target category ciljna kategorija

tree-adjoining grammar gramatika z vgrajevanjem dreves

wrapping ovijanje

Literatura

[1] M. A. Alonso in sod., Relating tabular parsing algorithms for LIG and TAG, V: Bunt in sod. (ur.) *New developments in parsing technology*, Berlin: Springer, 2004, str. 157-185.

[2] F. S. Eigner (2007), Combinatory Categorial Grammar. Predstavljeno na seminarju Formal Grammars. Dostopno na: http://www.ps.uni-saarland.de/courses/seminar-ws06/papers/03_fabienne_eigner.pdf

[3] J. Eijck, Sequentially Indexed Grammars, 2005. Dostopno na: http://www.cwi.nl/~jve/papers/05/sig/SIG.pdf

[4] J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Boston: Addison-Wesley, 1979.

[5] A. K. Joshi, K. Vijay-Shanker, D. Weir, The Convergence of Mildly Context-Sensitive Grammar Formalisms, *Foundational Issues in Natural Language Processing*, Cambridge: MIT Press, 1991, str. 31-81.

[6] A. K. Joshi, Y. Schabes, Tree-adjoining grammars, V: Rozenberg in Salomaa (ur.) *Handbook of Formal Languages: Beyond words*, Berlin: Springer, 1997, str. 69-124.

[7] A. Kroch, A. Joshi, The Linguistic Relevance of Tree Adjoining Grammar, 1985. Dostopno na: http://www.cs.sfu.ca/~anoop/courses/ReadingGroup-Summer-2006/KrochJoshi85.pdf

[8] J. C. Park, Using combinatory categorial grammar to extract biomedical information, *IEEE Intelligent Systems*, št.6, zv.16, 2001, str. 62-67.

[9] K. Vijay-Shanker, D. Weir, The Equivalence of Four Extensions of Context-Free Grammars, *Mathematical Systems Theory*, št. 6, zv. 27, 1994, str. 511-546.

[10] K. Vijay-Shanker, D. Weir, Polynomial time parsing of combinatory categorial grammars, *Proceedings of the 28th annual meeting on Association for Computational Linguistics*, Stroudsburg: Association for Computational Linguistics, 1990, str. 1-8.

[11] Y. Uemura in sod., Tree adjoining grammars for RNA structure prediction, *Theoretical Computer Science*, št.2, zv.210, 1999, str. 277-303.

[12] (2011) Head grammars. Dostopno na: http://en.wikipedia.org/wiki/Head_grammar