Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Fürst

Sintaksna analiza in indukcija grafnih gramatik

Doktorska disertacija

Mentor: prof. dr. Viljan Mahnič

Somentor: prof. dr. Marjan Mernik
(Univerza v Mariboru)

Ljubljana, 2013

Izjava o avtorstvu doktorske disertacije

Spodaj podpisani **Luka Fürst** z vpisno številko **63990376** sem avtor doktorske disertacije z naslovom **Sintaksna analiza in indukcija grafnih gramatik**. S svojim podpisom zagotavljam, da:

- sem doktorsko disertacijo izdelal samostojno pod vodstvom mentorja **prof. dr. Viljana Mahniča** in somentorja **prof. dr. Marjana Mernika**;
- so elektronska oblika doktorske disertacije, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko doktorske disertacije;
- soglašam z javno objavo elektronske oblike doktorske disertacije v zbirki *Dela FRI*.

V Ljubljani, dne 17. septembra 2013 Podpis avtorja: _____

Povzetek

Grafne gramatike so grafni zamenjevalni sistemi in jih zato lahko obravnavamo kot posplošitev širše znanih besedilnih gramatik. Grafna gramatika je, če nekoliko poenostavimo, sestavljena iz množice začetnih grafov (aksiomov) in iz množice zamenjevalnih pravil (produkcij). Vsaka produkcija podaja pravilo za zamenjavo grafa ali njegovega dela z nekim drugim grafom ali njegovim delom. Produkcijo na danem grafu uporabimo tako, da na njem izvršimo zamenjavo, ki jo produkcija predpisuje. Podobno kot je besedilna gramatika sredstvo za opis sintakse besedilnega formalnega (npr. programskega) jezika, lahko z grafno gramatiko opišemo sintakso grafnega jezika. Jezik, ki ga podana grafna gramatika določa, je množica, sestavljena iz množice aksiomov gramatike in iz grafov, ki jih je mogoče pridobiti z uporabami produkcij na aksiomih in drugih grafih iz jezika gramatike.

Prvi prispevek. Sintaksni analizator je algoritem, ki ugotovi, ali podani graf G pripada jeziku podane grafne gramatike GG. Če to drži, potem sintaksni analizator proizvede še izpeljavo grafa G v gramatiki GG — zaporedje uporab produkcij, ki enega od aksiomov gramatike GG po korakih pretvori v graf G. Medtem ko je sintaksna analiza besedilnih gramatik razmeroma znan in raziskan problem, je pri grafnih gramatikah drugače. V okviru prvega prispevka k znanosti, ki ga predstavimo v doktorski disertaciji, smo na več načinov izboljšali sintaksni analizator, ki sta ga leta 1997 v 1. številki revije *Journal of Visual Languages and Computing* opisala Rekers in Schürr. Ena od naših izboljšav povečuje razred grafnih gramatik, ki jih sintaksni analizator lahko sprejema na vhodu, preostale štiri pa so namenjene povečanju njegove časovne in prostorske učinkovitosti. Izboljšani sintaksni analizator uporabljamo v metodi za indukcijo grafnih gramatik, ki jo prav tako predstavimo v doktorski disertaciji.

Drugi prispevek. Grafno gramatiko lahko obravnavamo kot strnjen opis množice grafov, ki pripadajo njenemu jeziku. V okviru drugega prispevka k znanosti se ukvarjamo s problemom indukcije grafne gramatike na podlagi množice grafov, opremljenih z oznakami »pozitiven« in »negativen«. V doktorski disertaciji predstavimo izviren algoritem za izgradnjo grafne gramatike, ki smiselno posplošuje podano množico grafov. Indukcijski algoritem prične z gramatiko, katere jezik vsebuje natanko vse pozitivne vhodne grafe in nobenega negativnega, nato pa postopoma gradi množice čedalje splošnejših gramatik, katerih jeziki še vedno ne vsebujejo nobenega negativnega vhodnega grafa. Algoritem po načelu Ockhamove britve vrne najmanjšo gramatiko, ki jo ustvari v opisanem postopku. Konsistentnost posameznih gramatik z vhodno množico preverja s pomočjo sintaksnega analizatorja, ki ga predstavimo v doktorski disertaciji.

Tretji prispevek. Tretji prispevek k znanosti posega na področje domensko specifičnega modeliranja. Osnovna koncepta domensko specifičnega modeliranja sta *model*, ki predstavlja množico entitet in njihovih medsebojnih razmerij v modelirani domeni, in *metamodel*, ki podaja množico veljavnih modelov v dani domeni. Metamodele in modele pogosto predstavljamo z razrednimi oz. objektnimi diagrami v jeziku UML. Ker so objektni diagrami UML običajni grafi, lahko rečemo, da metamodel podobno kot grafna gramatika opisuje množico grafov. Za razliko od grafne gramatike pa metamodel svojo množico grafov določa deklarativno, ne generativno,

saj predpisuje lastnosti, ki jih morajo veljavni modeli posedovati, ne podaja pa pravil za sistematično tvorbo veljavnih modelov. Vendar pa lahko tovrstna generativna pravila pridobimo s pretvorbo podanega metamodela v grafno gramatiko, ki opisuje isto množico grafov kot metamodel. V doktorski disertaciji predstavimo izvirni postopek za takšno pretvorbo, poleg tega pa prikažemo, kako lahko izhodno grafno gramatiko uporabimo za semantično analizo ali transformacijo veljavnih modelov.

Ključne besede: grafna gramatika; sintaksna analiza; sintaksni analizator; indukcija; strojno učenje; domensko specifično modeliranje; metamodel

Abstract

Graph Grammar Parsing and Induction

Graph grammars are graph replacement systems and can be therefore regarded as a generalization of well-known string grammars. In slightly simplified terms, a graph grammar is composed of a set of initial graphs (axioms) and a set of replacement rules (productions). Every production specifies a possible replacement of a graph or its part with another graph or its part. To apply a production to a given graph, we transform it by applying a replacement specified by the production. In a similar manner as a string grammar can be used to define the syntax of a formal string language (e.g., a programming language), a graph grammar can be employed to define the syntax of a graph language. The language determined by a given graph grammar is a set comprising the grammar's axioms and all graphs that can be obtained by applying the grammar's productions to the axioms and to other graphs from the language.

First contribution. A parser is an algorithm that determines whether a given graph G belongs to the language of a given graph grammar GG. If this is the case, then the parser also produces a derivation of the graph G in the grammar GG, i.e., a sequence of production applications that gradually transforms one of the grammar's axioms to the graph G. In contrast to string grammar parsing, graph grammar parsing is a relatively unknown and under-researched problem. Our first scientific contribution presented in this thesis is an improved version of the graph grammar parser proposed by J. Rekers and A. Schürr in issue 1 of the 1997 *Journal of Visual Languages and Computing*. We improved the parser in several ways. One of the improvements expands the class of grammars accepted by the parser, whereas the other four increase the parser's temporal and spatial efficiency. The improved parser is employed in our graph grammar induction method, which is also presented in this thesis.

Second contribution. A graph grammar may be viewed as a compact description of the set of graphs belonging to its language. Our second scientific contribution deals with the problem of graph grammar induction based on a given set of graphs, each of which may be labeled 'positive' or 'negative'. We propose an original algorithm for the construction of a graph grammar that meaningfully generalizes the given graph set. Our induction algorithm starts with a grammar whose language comprises exactly all positive input graphs and none of the negative ones. After that, it gradually builds sets of progressively more general grammars whose languages still do not contain any negative input graph. Guided by Ockham's razor, the algorithm outputs the smallest grammar created in the process. The consistency of individual grammars with the input graph set is verified using the graph grammar parser described in this thesis.

Third contribution. Our third scientific contribution pertains to the field of domain specific modeling. The fundamental concepts in this field are the *model*, which represents a set of entities and their relations in a given modeling domain, and the *metamodel*, which specifies a set of valid models in the domain. Metamodels and models are often represented by UML class and object diagrams, respectively. Since UML object diagrams take the form of ordinary graphs, it can be said that a

metamodel, like a graph grammar, describes a set of graphs. In contrast to a graph grammar, a metamodel describes its graph set declaratively rather than generatively, since it defines the properties to be possessed by all valid models, but not the rules for the automatic generation of valid models. However, such generative rules can be obtained by converting a given metamodel into a graph grammar that defines the same graph set as the metamodel. In the thesis, we present an original metamodel-to-graph-grammar procedure and show how the output graph grammar can be used for semantic analysis or transformation of valid models.

Keywords: graph grammar; parsing; parser; induction; machine learning; domain-specific modeling; metamodel

Zahvala

Doktorska disertacija je obsežen in dolgotrajen projekt, sestavljen iz mnogih dni zbiranja in prebiranja literature, premišljevanja, iskanja izvirnih zamisli, programiranja, eksperimentiranja, analiziranja in seveda izdelave samega besedila. V času, ko sem se ukvarjal s tem projektom, sem bil v stiku s številnimi ljudmi. Nekateri so bili v projektu neposredno udeleženi, nekateri so pomagali s kakšno zamislijo ali vzpodbudo, nekateri pa so zgolj prijetno zapolnjevali trenutke, ko je projekt počival. V obdobju od novembra 2007 do septembra 2013, ko je zlagoma, ob številnih drugih obveznih in manj obveznih dejavnostih, nastajalo pričujoče delo, se je zvrstilo mnogo posameznikov s takšnimi in drugačnimi vlogami, zato naj mi bralec ne zameri, če se bom zahvalil nekoliko širšemu krogu, kot je to morda v navadi. Gotovo pa bom marsikoga tudi nehote izpustil, zato naj se že sedaj zahvalim vsem, ki se v tem sestavku ne bodo našli, pa bi se morali.

Zahvaljujem se mentorju prof. Viljanu Mahniču in somentorju prof. Marjanu Merniku, ki sta me vodila skozi celoten doktorski študij, me vzpodbujala in mi pomagala pri pisanju znanstvenih člankov in prispevkov ter mi lajšala (na srečo ne prehuda) krizna obdobja. Prof. Mahniču bi se rad — kot njegov asistent — zahvalil tudi za pomoč in podporo pri oblikovanju moje pedagoške plati, prof. Merniku pa za izjemno odzivnost pri odgovorih in komentarjih, za prijeten teden v ZDA in za njegov angažma pri članku za revijo *Software and Systems Modeling*.

Poleg prof. Mahniča in prof. Mernika mojo doktorsko komisijo sestavljata še prof. Borut Robič in prof. Matjaž Gams, ki se jima zahvaljujem za koristne pripombe pri predstavitvi doktorske teme in pri pripravi zaključne različice disertacije.

H kakovosti doktorske disertacije so neposredno prispevali neimenovani recenzenti člankov za reviji *IET Software* [37] in *Software and Systems Modeling* [36] ter prispevka za konferenco AGTIVE 2011 v Budimpešti [38], na katerih temeljijo osrednja poglavja doktorske disertacije. Za koristne pripombe se zahvaljujem tudi udeležencem konference AGTIVE 2011 in Juretu Leskovcu.

V okviru doktorskega študija sem teden dni gostoval na univerzi v Birminghamu v Alabami (ZDA). Moji gostitelji so bili prof. Mernik in njegovi tamkajšnji sodelavci, za kar se jim zahvaljujem.

Preblisk, ki je privedel do članka za konferenco AGTIVE 2011 in do četrtega poglavja doktorske disertacije, sem dobil ob branju odličnega doktorata Douglasa Lenata [62]. Nanj me je opozoril Jože Veber, konceptualni izumitelj dialektičnega *mislečega stroja* [96]. Ta še vedno obstaja zgolj na papirju, saj je zahtevne filozofske koncepte težko prevesti v zobata kolesa oziroma v delujočo programsko kodo ...

Kot član Laboratorija za tehnologijo programske opreme (LTPO) se zahvaljujem Marku Poženelu, mojemu nepogrešljivemu sodelavcu pri skoraj vseh vajah, in Igorju Rožancu, mojemu nekdanjemu »šefu« v Sežani, za številne prijetne trenutke in zanimive »življenjske« debate, v katerih s svojimi inteligentnimi in duhovitimi poantami, ki dajo misliti, pogosto sodeluje tudi Boštjan Slivnik. Za krajše obdobje sta bila moja sodelavca v LTPO in pri pedagoškem delu tudi Rok Štebe in Anže Časar.

Med moje pedagoške sodelavce v času doktorskega študija se poleg že omenjenih uvrščajo še (po abecedi) Uroš Čibej, Tomaž Dobravec, Roman Dorn, Luka Haupt-

man, Alenka Kavčič, Bojan Klemenc, Rajko Mahkovic, Aleš Smrdel in Iztok Starc. Zahvaljujem se tudi demonstratorjem, ki so pomagali pri izvajanju vaj, predvsem pa študentom, ki so me potrpežljivo prenašali ...

Zahvaljujem se Urošu Čibeju in Juriju Miheliču za povabilo v krog LALGinarjevcev, za številne prijetne pogovore ob (odličnih!) kosilih v menzi, pa še za kaj. Izvrstno intelektualno vzdušje na LALGinarjih redno soustvarjajo tudi nekateri drugi posamezniki iz naše fakultete. Poimensko jih raje ne bom navajal, saj bom gotovo koga izpustil.

Urošu Čibeju bi se rad zahvalil tudi za povabilo k »Zotkini« fakultetni izpostavi. V tem kontekstu naj omenim še Mijo Kordež, Boruta Jurčiča Zlobca in Darka Pevca. Slednjemu gre zahvala tudi za krasen teden v Avstraliji. In seveda, skoraj bi pozabil na legendarnega Romana Dorna!

Zahvaljujem se doc. Andreju Brodniku – Andyju, prof. Borutu Robiču, Gašperju Feletu Žoržu – Polžu in še komu za povabilo k sodelovanju pri pripravi konference ALGO 2012 v Ljubljani in za vse prijetne reči, ki so iz tega sledile.

Zahvaljujem se tudi drugim sodelavcem na FRI, ki so tako ali drugače prispevali k prijetnemu in ustvarjalnemu vzdušju.

Zahvaljujem se staršem za ljubezen, podporo in nedeljska kosila. Ko smo že pri prehrani, ne morem mimo *keksov* moje sestre in *štrudljev* tete Vesne, ki me je skupaj s svojo številno družino kar pogosto gostila. Zahvala gre tudi preostalemu sorodstvu. Julija, Irma, Ivan & Co., vi ste seveda všteti!

Janko, hvala ti za šah, tarok in hribolazenje!

Doktorsko disertacijo posvečam babici in dedu, ki sta se njenega nastanka iskreno veselila, a ga žal nista dočakala. Hvala vama za vse lepe trenutke!

Zadnji, a najpomembnejši odstavek zahvale namenjam osebi, ki v mojem življenju zavzema edinstveno mesto. Ne bom je imenoval, a se bo gotovo prepoznala, če bo to brala. Hvala ti za vse, kar mi daješ! Ob tebi sem odkril čudovite razsežnosti življenja, pred katerimi sem se leta in leta skrival ...

KAZALO

SLIKE

TABELE

POGLAVJE

1

UVOD

V uvodnem poglavju doktorske disertacije naše delo motiviramo (podpoglavje 1.1) in ga umestimo v ožje področje računalništva in informatike (podpoglavje 1.2). V podpoglavju 1.3 opredelimo prispevke k znanosti, ki jih predstavljamo v doktorski disertaciji. V podpoglavju 1.4 navajamo metode dela, ki smo se jih posluževali pri raziskovalnem delu v okviru priprave disertacije. Podpoglavje 1.5 služi kot kažipot po nadaljnjih poglavjih disertacije.

1.1 Motivacija

V pričujoči doktorski disertaciji se ukvarjamo s področjem *grafnih gramatik*, v okviru katerega predstavljamo tri medsebojno razmeroma neodvisne prispevke k znanosti. V tem podpoglavju najprej uvedemo in motiviramo pojem grafnih gramatik (razdelek 1.1.1), nato pa predstavimo koncepte, povezane s posameznimi prispevki doktorske disertacije (razdelki 1.1.2–1.1.4). Pričujoče podpoglavje je razmeroma neformalno, saj je njegov namen zgolj seznanitev bralca z vsebino doktorske disertacije. Na tem mestu tudi ne navajamo sorodnih del, ki bi utemeljevala izvirnost našega dela. To bomo storili v poglavjih 3, 4 in 5.

1.1.1 Grafne gramatike

Grafi — množice vozlišč in povezav med njimi — zavzemajo pomembno mesto na številnih strokovnih in znanstvenih področjih, pa tudi v vsakdanjem življenju. Navedimo nekaj primerov podatkov, ki jih lahko na naraven način predstavimo z grafi:

- Računalniška omrežja: vozlišča so usmerjevalniki ali druge naprave, povezave pa so kabli ali brezžične povezave.
- Transportna omrežja: vozlišča so kraji, povezave pa cestne, železniške, letalske ipd. povezave.
- Kemijske spojine: vozlišča so atomi, povezave pa vezi med njimi.
- Socialna omrežja (npr. Facebook): vozlišča so uporabniki, povezave pa »prijateljstva« (med uporabnikoma A in B obstaja povezava, če sta A in B »prijatelja«).
- Hipertekst: vozlišča so spletne strani, povezave pa hiperpovezave.
- Članki in citati: vozlišča so članki, povezave pa citati.
- Diagrami poteka: vozlišča so ukazi ali stavki, povezave pa podajajo potek programa.
- Razredni diagrami v jeziku UML: vozlišča so razredi, povezave pa asociacije med razredi.
- Objektni diagrami v jeziku UML: vozlišča so objekti, povezave pa vezi med objekti.

Grafne gramatike so v poznih 1960-ih letih [76] definirali kot posplošitev besedilnih gramatik (formalnih gramatik nad nizi). Grafna gramatika je strnjen opis množice grafov. V tem poglavju bomo grafne gramatike in sorodne pojme predstavili na nekoliko poenostavljen in posplošen način. Podrobnosti, ki niso bistvene za razumevanje ključnih konceptov, si bomo prihranili za poglavje 2.

V splošnem si grafno gramatiko lahko predstavljamo kot dvojico $(\mathcal{A}, \mathcal{P})$, kjer je $\mathcal{A}$ množica *začetnih grafov* (*aksiomov*), $\mathcal{P}$ pa množica *zamenjevalnih pravil* (*produkcij*).[1] Vsaka produkcija podaja pravilo za zamenjavo grafa ali njegovega dela z nekim drugim grafom. Produkcijo na danem grafu uporabimo tako, da izvršimo zamenjavo, ki jo produkcija opisuje. Produkcijo lahko zapišemo kot pravilo oblike $L ::= R$, kjer je leva stran L graf ali grafni element (vozlišče ali povezava), desna stran R pa je graf. Produkcijo $L ::= R$ na danem grafu uporabimo tako, da v njem poiščemo podgraf ali grafni element oblike L in ga zamenjamo z grafom R. (Ta opis je nekoliko poenostavljen, a nam bo v tem poglavju dovolj dobro služil.)

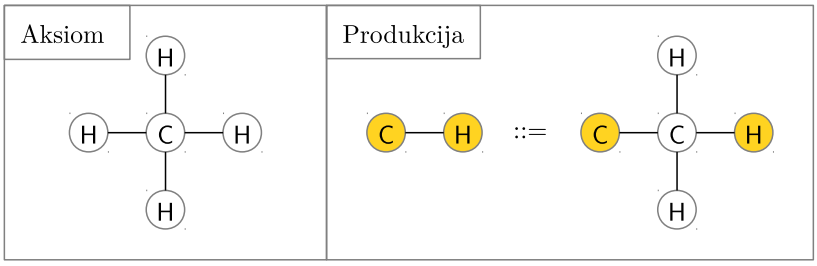
Množico grafov, ki jo grafna gramatika opisuje, imenujemo *jezik*. Jezik grafne gramatike je sestavljen iz njenih aksiomov in iz vseh grafov, ki jih lahko pridobimo z uporabami produkcij gramatike na aksiomih in drugih grafih, ki pripadajo jeziku gramatike. Natančneje:

(1) Jezik grafne gramatike vsebuje vse njene aksiome.

(2) Če v grafu, ki pripada jeziku grafne gramatike, izvršimo zamenjavo, kot jo določa ena od produkcij gramatike, potem dobljeni graf tudi pripada jeziku grafne gramatike.

(3) Jezik grafne gramatike ne vsebuje nobenih drugih grafov razen tistih, ki jih določata pravili (1) in (2).

Grafna gramatika na sliki 1.1 je sestavljena iz enega aksioma in ene produkcije. Aksiom predstavlja kemijsko strukturno formulo metana (CH_4), produkcija pa pove, da je poljubno povezavo med vozliščema z oznakama C in H (vez med ogljikovim in vodikovim atomom) možno nadomestiti s podgrafom oblike H−C−H, pri čemer je vozlišče C v dodanem podgrafu treba povezati še s krajiščema povezave, ki jo nadomestimo. Krajišči povezave na levi strani produkcije, ki smo ju označili z rumeno barvo, ostaneta nedotaknjeni, saj predstavljata le okvir (*kontekst*) zamenjave.



Slika 1.1: *Grafna gramatika z enim aksiomom in eno produkcijo.*

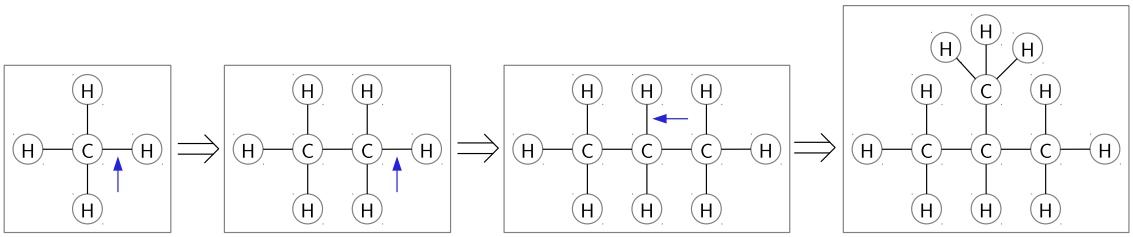
Jezik gramatike s slike 1.1 torej vsebuje aksiom (strukturno formulo metana) in vse grafe, ki jih je možno iz aksioma neposredno ali posredno pridobiti s pomočjo uporab produkcije gramatike. Produkcijo je mogoče uporabiti na poljubni povezavi C−H. Ugotovimo lahko, da imajo grafi v jeziku gramatike sledeče lastnosti:

[1] V literaturi se namesto izrazov »zamenjevalno pravilo« (angl. *substitution rule*) in »produkcija« (angl. *production*) uporablja tudi izraz »prepisovalno pravilo« (angl. *rewrite rule*).

- Vsako vozlišče v grafu je označeno bodisi z oznako C ali z oznako H. Vse povezave so neoznačene in enojne.
- Graf vsebuje najmanj eno vozlišče z oznako C in najmanj štiri vozlišča z oznako H.
- Vsako vozlišče C je povezano z natanko štirimi drugimi vozlišči.
- Vsako vozlišče H je povezano z natanko enim vozliščem C.
- Graf ne vsebuje nobenih ciklov.

Veljavnost vsake od navedenih lastnosti lahko pokažemo tako, da (1) preverimo, ali lastnost velja za aksiom gramatike, in (2) preverimo, ali produkcija lastnost ohranja: če lastnost velja za nek graf G, moramo preveriti, ali velja tudi za graf, pridobljen z uporabo produkcije na neki povezavi C−H v grafu G. Na osnovi navedenih lastnosti lahko zaključimo, da jezik gramatike s slike 1.1 vsebuje natanko vse veljavne strukturne formule acikličnih alkanov (ogljikovodikov s samimi enojnimi vezmi).

Zaporedje uporab produkcij, ki enega od aksiomov grafne gramatike po korakih pretvori v podani graf G, imenujemo *izpeljava* (angl. *derivation*) grafa G v dani grafni gramatiki. Izpeljava služi kot dokaz pripadnosti grafa jeziku grafne gramatike. Slika 1.2 prikazuje izpeljavo strukturne formule metilpropana v gramatiki s slike 1.1. Z modrimi puščicami so označene povezave, na katerih v tekočem koraku izpeljave uporabimo produkcijo gramatike.



Slika 1.2: *Izpeljava strukturne formule metilpropana v grafni gramatiki s slike 1.1.*

Naj opozorimo, da gramatika s slike 1.1 ne predstavlja sheme za dejansko kemijsko sintezo acikličnih alkanov. V doktorski disertaciji bomo strukturne formule kemijskih spojin obravnavali zgolj kot matematične strukture (grafe); s kemijskimi lastnostmi spojin se ne bomo ukvarjali. Kljub temu pa je nekatere organske kemijske reakcije možno predstaviti z grafnimi gramatikami [8].

Kot bomo videli v poglavju 2, je grafne gramatike možno definirati na več različnih načinov. Podobno kot pri besedilnih gramatikah lahko vpeljemo določene omejitve (npr. kontekstno neodvisnost), končne in nekončne simbole itd. Gramatika s slike 1.1 je tako primer kontekstno odvisne grafne gramatike, saj vozlišči C in H, ki pri uporabi produkcije ostaneta nedotaknjeni, predstavljata kontekst zamenjave. Osnovna zamisel — gradnja grafov s pomočjo zamenjav — pa je pri vseh definicijah grafnih gramatik enaka.

Grafne gramatike so uporabne na številnih področjih [10, 28], med drugim na področjih vizualnih programskih in modelirnih jezikov [42, 80, 103], modeliranja

programske opreme [13, 30, 36, 78], razpoznavanja vzorcev [34, 61, 63], računalniške grafike [3, 52, 85], funkcijskih programskih jezikov [77] itd.

1.1.2 Grafne gramatike in sintaksna analiza

Sintaksa mnogih besedilnih programskih jezikov (npr. pascal, C ali java) je definirana s pomočjo besedilnih gramatik. Besedilna gramatika določa množico vseh sintaktično pravilnih programov v danem programskem jeziku. Sintaktično pravilnost programa (glede na gramatiko programskega jezika) preverimo s pomočjo *sintaksne analize* (angl. *parsing*) oziroma s pomočjo algoritma, imenovanega *sintaksni analizator* (angl. *parser*). Če je program sintaktično pravilen, sintaksni analizator zgradi njegovo izpeljavo (npr. v obliki drevesa izpeljav, angl. *parse tree*), ki lahko služi kot osnova za prevod programa v strojni jezik [1]. Sintaksna analiza je torej temeljni postopek pri obdelavi programov v besedilnih programskih jezikih.

Na področju grafnih gramatik je sintaksni analizator algoritem, ki kot vhod sprejme graf in grafno gramatiko, njegov izhod pa je izpeljava grafa v grafni gramatiki (če graf pripada jeziku gramatike) oziroma odgovor »ne« (v nasprotnem primeru). Po analogiji z besedilnimi gramatikami lahko grafni sintaksni analizator uporabljamo za sintaksno analizo programov v grafnih programskih jezikih. Primer preprostega grafnega programskega jezika je jezik diagramov poteka. Sintaksni analizator bi na vhodu sprejel graf in grafno gramatiko diagramov poteka, nato pa bi preveril, ali podani graf pripada jeziku podane gramatike (ali je podani graf veljaven diagram poteka), in v primeru pritrdilnega odgovora zgradil izpeljavo podanega grafa. Na podlagi izpeljave bi lahko zgradili prevod diagrama poteka v, denimo, enakovreden besedilni program. Sintaksni analizator torej nastopa kot sestavni del programerskih orodij, kot so prevajalniki, razhroščevalniki ipd. Zaradi tega se je s problemom sintaksne analize — tako pri besedilnih kot pri grafnih gramatikah — smiselno ukvarjati.

V doktorski disertaciji bomo predstavili drugačno uporabo grafnega sintaksnega analizatorja. Kot bomo videli v razdelku 1.1.3 in poglavju 4, igra sintaksni analizator pomembno vlogo v metodi za indukcijo grafnih gramatik. Indukcijski postopek si pri gradnji grafne gramatike na podlagi dane množice »pozitivnih« in »negativnih« učnih grafov pogosto pomaga s sintaksnim analizatorjem, saj ga uporabi za vsak par izdelane kandidatne gramatike in negativnega učnega grafa. Zaradi tega potrebujemo karseda učinkovit sintaksni analizator.

Na začetku našega raziskovalnega dela na področju sintaksne analize grafnih gramatik smo realizirali Rekers-Schürrov sintaksni analizator [80], v nadaljevanju pa smo ga izboljšali po izvirnih zamislih, ki jih bomo predstavili v poglavju 3. Rezultat naših izboljšav je širši razred gramatik, ki jih sintaksni analizator lahko sprejema na vhodu, in boljša računska učinkovitost za mnoge primere vhodnih gramatik. Za gramatike kemijskih strukturnih formul, kot je npr. tista s slike 1.1, je sintaksna analiza postala časovno in prostorsko izvedljiva šele z našimi izboljšavami.

1.1.3 Indukcija grafnih gramatik

Kot smo videli v razdelku 1.1.1, lahko s pomočjo grafnih gramatik opišemo nekatere množice kemijskih spojin. Grafna gramatika s slike 1.1, denimo, opisuje množico acikličnih alkanov. Problem indukcije grafne gramatike se ukvarja z *učenjem* grafne gramatike na podlagi množice znanih grafov iz njenega jezika. Na primer, sistemu za indukcijo grafnih gramatik bi lahko na vhodu podali množico strukturnih formul izbranih alkanov; izbrali bi, denimo, nekaj nerazvejanih, nekaj šibko razvejanih in nekaj močno razvejanih alkanov, da bi čim bolje zajeli strukturne razlike znotraj množice alkanov. Pri takšnem vhodu bi od sistema pričakovali, da bo zgradil gramatiko, enakovredno tisti s slike 1.1.

Na področju kemije si lahko zamislimo še drugačen učni scenarij: sistemu podamo množico strukturnih formul kemijskih spojin, ki imajo določeno lastnost (npr. rakotvornost), in množico strukturnih formul, ki te lastnosti nimajo. Pri takšnem vhodu od indukcijskega sistema pričakujemo, da bo zgradil grafno gramatiko, s pomočjo katere bo mogoče napovedovati prisotnost oz. odsotnost dane lastnosti (v našem primeru rakotvornosti) pri kemijskih spojinah, za katere še ni znano, ali to lastnost izkazujejo.

Beseda *indukcija* pomeni sklepanje od posameznega k splošnemu, torej gradnja splošnega pravila (v našem primeru grafne gramatike) na podlagi posameznih primerov (v našem primeru množice vhodnih grafov). Ukvarjali se bomo z dvema oblikama indukcije grafnih gramatik:

Indukcija na podlagi enakovrednih grafov. Pri tej obliki indukcije so vsi vhodni grafi enakovredni. To obliko uporabljamo, kadar domnevamo, da vsi grafi pripadajo isti gramatiki, in bi želeli to gramatiko poiskati ali pa kadar nas zanimajo morebitne skupne lastnosti vhodnih grafov in bi jih želeli izraziti z grafno gramatiko. Primer te oblike je indukcija grafne gramatike na podlagi množice strukturnih formul alkanov.

Indukcija na podlagi grafov z znano razredno pripadnostjo. Pri tej obliki indukcije sta podana dva *razreda* grafov, ki ju običajno imenujemo »pozitivni« in »negativni« razred. Vsak vhodni graf pripada bodisi pozitivnemu ali negativnemu razredu. Razredne pripadnosti vhodnih grafov so podane. Naloga indukcijskega algoritma je zgraditi grafno gramatiko, katere jezik vsebuje vse pozitivne in nobenega negativnega vhodnega grafa. Primer te oblike je indukcija grafne gramatike na podlagi množice strukturnih formul kemijskih spojin, pri katerih je znana prisotnost oz. odsotnost lastnosti, ki bi jo želeli napovedovati pri še nerazvrščenih kemijskih spojinah. Opisani indukcijski scenarij je mogoče posplošiti na več razredov, saj lahko za vsak razred posebej zgradimo gramatiko, ki razlikuje med vhodnimi grafi iz tega razreda in vsemi ostalimi vhodnimi grafi.

Indukcija na podlagi enakovrednih grafov približno ustreza problemu nenadzorovanega strojnega učenja (angl. *unsupervised machine learning*), čeprav pod tem pojmom običajno razumemo iskanje skupin učnih podatkov s podobnimi lastnostmi (angl. *clustering*), s katerim se v doktorski disertaciji ne bomo ukvarjali. To obliko indukcije si lahko predstavljamo tudi kot problem s področja odkrivanja znanja v

podatkih (angl. *data mining*). Indukcijo na podlagi grafov z znano razredno pripadnostjo pa lahko nedvoumno označimo kot problem nadzorovanega strojnega učenja (angl. *supervised machine learning*). Grafna gramatika, ki jo gradimo pri tej obliki indukcije, predstavlja *klasifikator* (angl. *classifier*) — orodje za razvrščanje bodočih (še neuvrščenih) grafov v enega od podanih razredov. Če graf pripada jeziku inducirane gramatike, ga uvrstimo v pozitivni, sicer pa v negativni razred.

Pogoj iz druge oblike indukcije je možno trivialno izpolniti, saj gramatika z množico aksiomov $\{G^+ \mid G^+ \text{ je pozitiven vhodni graf}\}$ in brez produkcij pokriva natanko vse pozitivne vhodne grafe in nobenega negativnega. (Gramatika *pokriva* graf natanko tedaj, ko ta pripada njenemu jeziku.) Vendar pa s takšno gramatiko nismo zadovoljni, saj se na bodočih (testnih) grafih najverjetneje ne bo dobro obnašala. Zato bomo pri indukciji na podlagi grafov z znano razredno pripadnostjo iskali *najmanjšo* grafno gramatiko, ki izpolnjuje zahtevani pogoj, saj lahko po načelu Ockhamove britve upamo, da bo takšna gramatika dobro klasificirala testne grafe. Kakovost inducirane gramatike lahko preverimo po običajnem receptu v strojnem učenju: gramatiko induciramo na podlagi množice učnih grafov, preverimo pa jo na (povsem ločeni) množici testnih grafov. Pri podani množici učnih in testnih grafov lahko inducirano gramatiko torej ocenimo tudi kvantitativno, ne zgolj kvalitativno, saj lahko izračunamo, kolikšen delež testnih grafov gramatika pravilno razvrsti.

Indukcijski algoritem izhaja iz gramatike, ki pokriva natanko vse pozitivne učne grafe. Nato v vsakem naslednjem koraku zgradi množico, sestavljeno iz nekoliko posplošenih različic grafnih gramatik iz predhodnega koraka, pri čemer obdrži samo tiste gramatike, ki ne pokrivajo nobenega negativnega učnega grafa. Na ta način algoritem — skladno s pomenom pojma *indukcija* — postopoma gradi čedalje splošnejše gramatike, ki so konsistentne s podano množico učnih grafov. Algoritem v prostoru, ki ga preiskuje, išče najmanjšo gramatiko.

Problem indukcije na podlagi enakovrednih grafov je slabše definiran, saj pogoju, da mora inducirana gramatika pokrivati vse učne grafe, vselej zadoščata najbolj specifična gramatika (gramatika, ki pokriva zgolj učne grafe in nič drugega) in najbolj splošna gramatika (gramatika, ki pokriva vse možne grafe iz neke domene). Zato bomo pri tej obliki indukcije gramatike gradili na enak način kot pri indukciji na podlagi grafov z znano razredno pripadnostjo, vendar pa bomo nastale gramatike ocenjevali zgolj kvalitativno. Zanimalo nas bo, ali inducirana gramatika zajema skupne lastnosti grafov iz podane množice.

Omenili smo, da si indukcijski postopek pomaga s sintaksnim analizatorjem za grafne gramatike. Sintaksni analizator se uporablja pri preverjanju gramatik, ki nastajajo pri postopnem posploševanju izhodiščne gramatike. Z njegovo pomočjo za vsako nastalo gramatiko preverimo, ali pokriva katerega od negativnih učnih grafov. To pomeni, da sintaksni analizator uporabimo za vsak par gramatike in negativnega učnega grafa. Zato je učinkovit sintaksni analizator potreben pogoj za časovno in prostorsko izvedljivost indukcijskega postopka.

Sintaksni analizator uporabljamo tudi za preverjanje kakovosti induciranih gramatik. Z njegovo pomočjo ugotovimo, katere testne grafe inducirana gramatika pravilno klasificira.

Indukcijski postopek, ki ga bomo predstavili v poglavju 4, uporablja izviren pristop k posploševanju grafnih gramatik. Na področju indukcije *grafnih* grama-

tik lahko kot izviren pristop opredelimo tudi vključitev sintaksnega analizatorja v indukcijski postopek. Izvirnost našega indukcijskega postopka bomo utemeljili v poglavju 4.

1.1.4 Pretvorba metamodela v grafno gramatiko

V tretjem prispevku k znanosti, ki ga bomo predstavili v doktorski disertaciji, bomo pokazali uporabo grafnih gramatik na področju domensko specifičnega modeliranja [39]. Domensko specifično modeliranje se najpogosteje uporablja pri razvoju programske opreme, lahko pa z njegovo pomočjo razvijamo sisteme tudi na drugih inženirskih področjih. Osnovna zamisel domensko specifičnega modeliranja je predstavitev sistema, ki ga želimo razviti, z množico *modelov* in *metamodelov*. Vsak model predstavlja neko množico entitet v sistemu in razmerja med njimi. Metamodel pa opisuje množico veljavnih modelov; vsak model, ki izpolnjuje pravila, predpisana z metamodelom, predstavlja neko veljavno stanje sistema.

Za opis modelov se pogosto uporabljajo *objektni diagrami*, za opis metamodelov pa *razredni diagrami* v jeziku UML [35]. Objektni diagram je graf, v katerem vozlišča predstavljajo objekte (entitete v modelu), povezave pa vezi med objekti (entitetami). Razredni diagram pa je graf, v katerem vozlišča predstavljajo posamezne razrede (entitetne tipe), povezave pa tipe vezi med objekti (entitetami) in omejitve, ki veljajo zanje. Razredni diagram je strnjen opis množice vseh objektnih diagramov, ki se skladajo s pravili, določenimi z razrednim diagramom — torej vseh objektnih diagramov, ki so veljavni glede na razredni diagram.

Glede na to, da so modeli (kot objektni diagrami UML) dejansko grafi, lahko rečemo, da je metamodel (kot razredni diagram UML) strnjen opis množice vseh grafov, ki spoštujejo njegova pravila. Kot smo videli, je tudi grafna gramatika strnjen opis množice grafov. Vendar pa se metamodeli bistveno razlikujejo od grafnih gramatik, saj grafna gramatika opisuje svojo množico grafov *generativno*, metamodel pa *deklarativno*. Medtem ko grafna gramatika opredeljuje pravila za *tvorbo* (generiranje) grafov, ki pripadajo njenemu jeziku, metamodel podaja (deklarira) *lastnosti*, ki določajo veljavne grafe.

Oba načina opisovanja množice grafov imata svoje prednosti in slabosti. Dobra stran deklarativnega opisa množice grafov je enostavno preverjanje pripadnosti grafa tej množici; zlahka ugotovimo, ali podani model poseduje lastnosti, ki jih predpisuje dani metamodel. Pri grafnih gramatikah pa preverjanje pripadnosti jeziku še zdaleč ni enostavno, saj v ta namen, kot smo zapisali v razdelku 1.1.2, potrebujemo sintaksni analizator. Po drugi strani pa nam grafna gramatika omogoča samodejno in sistematično tvorbo grafov, ki pripadajo njenemu jeziku, medtem ko metamodel te možnosti neposredno ne ponuja. Samodejna gradnja veljavnih modelov (glede na dani metamodel) pa je smiselna, saj za sistematično testiranje modelskih transformacij [64, 88] in drugih operacij oz. algoritmov na (meta)modelih praviloma potrebujemo veliko število veljavnih modelov. Možnost za samodejno tvorbo veljavnih modelov lahko torej pridobimo tako, da metamodel pretvorimo v grafno gramatiko, katere jezik je enak množici veljavnih modelov za dani metamodel. Poleg samodejne gradnje veljavnih modelov nam grafna gramatika, ki je enakovredna metamodelu, omogoča tudi semantično analizo ali transformacijo veljavnih modelov, saj lahko

produkcije grafnih gramatik opremimo s semantičnimi pravili na podoben način kot produkcije besedilnih gramatik [74].

V poglavju 5 bomo predstavili izviren postopek za pretvorbo metamodela v kontekstno odvisno grafno gramatiko. Izhodna gramatika je definirana tako, da jo je možno podati kot vhod izboljšanemu Rekers-Schürrovemu sintaksnemu analizatorju, ki ga bomo predstavili v poglavju 3. Poleg tega bomo pokazali, kako lahko izhodno gramatiko opremimo s semantičnimi pravili in tako pridobimo možnost semantične analize modelov.

1.2 Umestitev doktorske disertacije

Doktorsko disertacijo lahko umestimo v štiri ožja področja računalništva in informatike: grafne gramatike (razdelek 1.2.1), algoritmi in podatkovne strukture (razdelek 1.2.2), tehnologija programske opreme (razdelek 1.2.3) in strojno učenje (razdelek 1.2.4).

1.2.1 Grafne gramatike

V doktorski disertaciji se ukvarjamo s področjem grafnih gramatik. Glavni rezultati s tega področja so zbrani v obsežni trilogiji *Handbook of Graph Grammars and Computing by Graph Transformation* [28, 29, 81], ki jo bomo v nadaljevanju imenovali kar *Priročnik*. Medtem ko je bilo v preteklosti opravljenega veliko teoretičnega in praktičnega dela na temo grafnih gramatik, je v novejšem času to področje vsaj v primerjavi z nekaterimi drugimi vejami računalništva slabo poznano. Po našem védenju obstajata dve mednarodni konferenci na temo grafnih gramatik in sorodnega področja grafnih transformacij (*International Conference on Graph Transformation* na vsaki dve leti in *Applications of Graph Transformations with Industrial Relevance* na vsaka štiri leta), mednarodnih revij, prvenstveno namenjenih grafnim gramatikam in transformacijam, pa ne poznamo. Poleg tega se izmed že tako ali tako skromnega števila člankov na to temo le redki ukvarjajo s problemom sintaksne analize in indukcije grafnih gramatik. Večina sodobnih raziskav se ubada z uporabo grafnih transformacij na področju modeliranja programske opreme, ki je bistveno bolje podprto s konferencami in revijami. Prispevkov na temo grafnih gramatik samih po sebi (brez povezave z bolj priljubljenimi področji) je v novejšem času bore malo.

Zakaj področje grafnih gramatik ni bolj dejavno? Po mnenju strokovnjakov, izraženem v pogovorih na konferenci AGTIVE 2011 v Budimpešti, sta glavna razloga sledeča:

- Področje grafnih gramatik nikoli ni bilo posebej znano in priljubljeno. Od vsega začetka je omejeno na peščico nemških in nizozemskih raziskovalnih skupin in se z izjemo redkih »otokov« skorajda ni prebilo onstran meja omenjenih držav.

- Algoritmi, povezani z grafnimi gramatikami, so praviloma računsko zahtevni. Problem sintaksne analize, denimo, je že za zelo omejene razrede grafnih gra-

matik NP-težak [82]. To neprijetno dejstvo je gotovo marsikoga odvrnilo od raziskav na tem področju.

V novejšem času so ameriški raziskovalci, zbrani okrog profesorjev Diane J. Cook in Lawrencea B. Holderja, objavili nekaj člankov in doktorat na temo indukcije grafnih gramatik [4, 49, 55, 56], a se, kot je videti, od leta 2009 s tem področjem ne ukvarjajo več.

Medtem ko se bomo v poglavju 5 z uporabo grafnih gramatik na področju modeliranja programske opreme tudi mi pridružili novejšim raziskovalnim smernicam, bomo v poglavjih 3 in 4 bredli po razmeroma deviškem terenu v redko obiskanih zakotjih računalniške znanosti. Vendar pa poglavji 3 in 4 posegata tudi na področji algoritmov in podatkovnih struktur ter strojnega učenja, ki sta bistveno živahnejši od področja grafnih gramatik.

1.2.2 Algoritmi in podatkovne strukture

Doktorska disertacija posega tudi na področje algoritmov in podatkovnih struktur, saj v vseh treh prispevkih raziskujemo nove algoritme ali izboljšujemo obstoječe. To področje je bistveno dejavnejše od področja grafnih gramatik. Poleg številnih znanstvenih dogodkov in publikacij, namenjenih algoritmom in podatkovnim strukturam nasploh, obstajajo tudi konference in revije na temo grafnih algoritmov, denimo konferenca *European Conference on Combinatorics, Graph Theory, and Applications* (EUROCOMB), revija *Journal Of Graph Algorithms and Applications* itd.

1.2.3 Tehnologija programske opreme

S področjem tehnologije programske opreme se ukvarjamo predvsem v poglavju 5, do neke mere pa tudi v poglavjih 3 in 4, saj bi bilo naše rezultate možno uporabiti v orodju za specifikacijo sintakse grafnega programskega jezika in za razvoj programov v takšnih jezikih. V poglavju 5 pa posegamo na področje modeliranja programske opreme, ki se je v zadnjih letih močno razmahnilo. Obstaja precej konferenc in revij z omenjenega področja, denimo konferenca *International Conference on Model-Driven Engineering Languages and Systems* (MoDELS), revija *Software and Systems Modeling* itd.

1.2.4 Strojno učenje

Področja strojnega učenja, ki se ukvarja z odkrivanjem pravil ali vzorcev iz množice podatkov, se dotikamo v poglavju 4. Ker si grafno gramatiko lahko predstavljamo kot množico pravil, ki opisujejo njen jezik, je indukcija grafne gramatike oblika učenja pravil na podlagi množice primerov. Na področju strojnega učenja obstaja vrsta mednarodnih konferenc in revij, denimo konferenca *International Conference on Machine Learning* (ICML), revija *Machine Learning* itd.

1.3 Prispevki k znanosti

Doktorska disertacija predstavlja tri neodvisne prispevke k znanosti:

Izboljšava obstoječe metode za sintaksno analizo pri grafnih gramatikah. Izboljšali smo Rekers-Schürrov sintaksni analizator za kontekstno odvisne grafne gramatike [80] in članek o izboljšani metodi objavili v reviji s faktorjem vpliva [37]. Povečali smo razred gramatik, ki jih sprejema Rekers-Schürrov sintaksni analizator, in za mnoge primere vhodnih gramatik izboljšali časovno in prostorsko učinkovitost sintaksnega analizatorja.

Izvirna metoda za indukcijo grafnih gramatik. Metoda, ki smo jo objavili na mednarodnem simpoziju AGTIVE 2011 [38], uporablja izviren pristop k indukciji grafnih gramatik, poleg tega pa indukcijski postopek prepleta s sintaksno analizo, kar omogoča indukcijo na podlagi pozitivnih *in* negativnih vhodnih grafov.

Izvirna metoda za pretvorbo metamodela v grafno gramatiko. Razvili smo izvirno metodo za pretvorbo metamodela v obliki razrednega diagrama UML v kontekstno odvisno grafno gramatiko. Metodo smo opisali v članku za revijo *Software and Systems Modeling* [36].

1.4 Metode dela

Pri raziskovalnem delu, ki ga opisujemo v doktorski disertaciji, smo se posluževali metod, ki so običajne na področjih, na katera posega naše delo:

- *Razvoj novih algoritmov.* Pri vseh treh prispevkih, ki jih predstavljamo v doktorski disertaciji, smo razvijali nove algoritme ali izboljševali obstoječe. Algoritme smo implementirali v programskem jeziku java. Pri vizualizaciji grafov in grafnih gramatik smo si pomagali z orodjema yEd[2] in knjižnico JUNG.[3]

- *Eksperimentalna analiza.* Eksperimentalne analize smo se posluževali pri algoritmih v poglavjih 3 in 4, saj sintaksnega analizatorja in metode za indukcijo grafnih gramatik teoretično ni mogoče dovolj dobro ovrednotiti. Eksperimentalna analiza je pri obeh prispevkih nujna za pridobitev občutka o delovanju algoritmov na različnih vhodnih podatkih.

- *Teoretična analiza.* Pri metodi za pretvorbo metamodela v grafno gramatiko nam že teoretična slika da dovolj dobro sliko o delovanju pretvornega algoritma. Pri sintaksnem analizatorju pa bomo s teoretično analizo pokazali, da naše izboljšave ohranjajo njegovo pravilnost.

- *Kritično vrednotenje rezultatov.* V doktorski disertaciji bomo rezultate poskušali objektivno ovrednotiti in prikazati tako prednosti kot pomanjkljivosti predstavljenih algoritmov.

1.5 Pregled doktorske disertacije

Ta razdelek služi kot kažipot za nadaljnje besedilo doktorske disertacije.

[2] `http://www.yworks.com/en/products_yed_about.html`
[3] `http://jung.sourceforge.net/`

1.5.1 Pregled posameznih poglavij

Sledi pregled vsebine posameznih poglavij:

Poglavje 2: To poglavje je namenjeno stični točki vseh treh prispevkov: grafnim gramatikam. V poglavju 2 formalno predstavimo grafne gramatike in vzpostavimo notacijo in terminologijo, ki jo uporabljamo v vseh nadaljnjih poglavjih.

Poglavje 3: V tem poglavju predstavimo Rekers-Schürrov sintaksni analizator in njegove izvirne izboljšave.

Poglavje 4: V tem poglavju predstavimo našo metodo za indukcijo grafnih gramatik.

Poglavje 5: V tem poglavju prikažemo izviren postopek za pretvorbo metamodela v enakovredno grafno gramatiko.

Poglavje 6: S tem poglavjem se doktorska disertacija zaključi.

1.5.2 Prednostna razmerja med poglavji

Poglavje 2 je predpogoj za razumevanje vseh nadaljnjih poglavij. Poglavja 3, 4 in 5 pa je načeloma mogoče brati v poljubnem vrstnem redu. V poglavjih 4 in 5 se sicer sklicujemo na sintaksni analizator za grafne gramatike, vendar pa bo za razumevanje njegove vloge pri indukciji grafnih gramatik in pretvorbi metamodela v grafno gramatiko povsem zadoščal abstraktni pogled, ki smo ga predstavili že v razdelku 1.1.2. Na tehnične podrobnosti sintaksne analize, predstavljene v poglavju 3, se v poglavjih 4 in 5 ne sklicujemo.

POGLAVJE

2

GRAFNE GRAMATIKE

V tem poglavju najprej (v podpoglavju 2.1) definiramo nekatere splošne pojme in uvedemo notacijo, ki jo uporabljamo v nadaljevanju doktorske disertacije. V podpoglavju 2.2 predstavimo koncepte iz teorije grafov, ki so ključni za razumevanje grafnih gramatik. V podpoglavju 2.3 predstavimo grafne gramatike na splošno, poleg tega pa na kratko opišemo nekatere oblike grafnih gramatik. V podpoglavju 2.4 pa podrobno predstavimo kontekstno odvisne grafne gramatike, kot sta jih definirala Rekers in Schürr [80], saj tovrstne gramatike uporabljamo v vseh nadaljnjih poglavjih.

2.1 Nekatere splošne definicije in oznake

V tem podpoglavju bomo vzpostavili nekaj splošnih definicij in dogovorov, ki se jih bomo držali v celotni doktorski disertaciji. Omejili se bomo na tiste pojme, ki jih bomo v nadaljnjem besedilu dejansko potrebovali. V razdelku 2.1.1 bomo pregledali nekaj konceptov iz teorije množic, v razdelku 2.1.2 pa ključne pojme na temo preslikav in relacij.

2.1.1 Zbirke elementov

Množica je zbirka elementov brez podvajanja. V skladu z uveljavljenimi dogovori bomo vsebino množice podali kot seznam elementov znotraj zavitih oklepajev. Na primer, zapis $\{a, b, c\}$ predstavlja množico z elementi a, b in c. Ker bomo pogosto uporabljali množice, sestavljene iz strnjenega zaporedja celih števil, bomo uvedli okrajšavo $a\,..\,b$ za končno množico $\{a, a+1, \ldots, b\}$ $(a, b \in \mathbb{Z})$ in okrajšavo $a\,..\,*$ za števno neskončno množico $\{a, a+1, a+2, \ldots, \}$ $(a \in \mathbb{Z})$. Velja $a\,..\,a = \{a\}$. V primeru $a > b$ velja $a\,..\,b = \emptyset$ (prazna množica). Prikazani zapis intervalskih množic izvira iz načina zapisa multiplikativnosti asociacij v jeziku UML [35]. Koncepte, kot so podmnožica, nadmnožica, presek, unija, razlika, kartezični produkt ipd., bomo uporabljali in zapisovali na uveljavljen način. Na primer, zapis $\mathcal{A} \subset \mathcal{B}$ pove, da je $\mathcal{A}$ stroga podmnožica množica $\mathcal{B}$, zapis $\mathcal{A} \subseteq \mathcal{B}$ pa dopušča možnost, da sta množici enaki. Razliko množic $\mathcal{A}$ in $\mathcal{B}$ bomo zapisovali kot $\mathcal{A} \setminus \mathcal{B}$, ne kot $\mathcal{A} - \mathcal{B}$, kot lahko vidimo ponekod v literaturi. Kartezični produkt množic $\mathcal{A}$ in $\mathcal{B}$ označujemo z $\mathcal{A} \times \mathcal{B}$. Kartezični produkt množice s samo seboj zapisujemo kot potence: $\mathcal{A}^1 = \mathcal{A}$, $\mathcal{A}^2 = \mathcal{A} \times \mathcal{A}$, $\mathcal{A}^k = \mathcal{A} \times \mathcal{A}^{k-1}$ (pri $k \geq 2$).

Zbirko elementov, v kateri se vsak element lahko poljubnokrat ponovi, bomo imenovali *vreča* (angl. *bag* ali *multiset*). Vreče bomo zapisovali na enak način kot običajne množice. Z zapisom $num[\mathcal{A}](x)$ bomo označili število kopij elementa x v vreči $\mathcal{A}$. Na primer, za vrečo $\mathcal{A} = \{a, c, c\}$ velja $num[\mathcal{A}](a) = 1$, $num[\mathcal{A}](b) = 0$, $num[\mathcal{A}](c) = 2$ itd.

Vreča $\mathcal{A}$ je *podvreča* vreče $\mathcal{B}$, če za vsak element x iz vreče $\mathcal{A}$ velja $num[\mathcal{A}](x) \leq num[\mathcal{B}](x)$. Na primer, vreča $\mathcal{A} = \{a, c, c\}$ je podvreča vreče $\mathcal{B} = \{a, a, b, b, c, c\}$, ne pa vreče $\mathcal{C} = \{a, a, b, b, c\}$, saj v vreči $\mathcal{C}$ element c nastopa le enkrat. Vreči $\mathcal{A}$ in $\mathcal{B}$ sta *enaki*, če je $\mathcal{A}$ podvreča $\mathcal{B}$ in hkrati $\mathcal{B}$ podvreča $\mathcal{A}$.

Če je $\mathcal{A}$ množica, bomo z zapisom $|\mathcal{A}|$ označevali njeno *velikost* ali *kardinalnost*, tj. število njenih elementov. Če je $\mathcal{A}$ vreča, potem $|\mathcal{A}|$ predstavlja skupno število vseh pojavitev elementov. Na primer, $|\{a, c, c\}| = 3$. Oznaka $\mathcal{P}(\mathcal{B})$ bo predstavljala potenčno množico (množico vseh podmnožic) množice $\mathcal{B}$. »Potenčne vreče« ne bomo potrebovali nikjer, zato je ne bomo definirali.

Pri množicah in vrečah vrstni red elementov ni pomemben (dejansko sploh ni določen), pri *zaporedjih* pa je. Podobno kot pri vrečah se elementi tudi v zaporedjih lahko ponavljajo. Elemente zaporedij bomo pisali v okroglih oklepajih, npr. (b, a, b, c). Zaporedju k elementov pravimo tudi *k-terica* (dvojica/par, trojica, četverica itd.). Zaporedji sta *enaki*, če vsebujeta enako število elementov in če so vsi elementi na istoležnih pozicijah v obeh zaporedjih enaki.

Zaporedja znakov se imenujejo *nizi*. Nize običajno pišemo kar kot besede; de-

nimo, zapis `kolo` je okrajšava za zaporedje znakov (`k`, `o`, `l`, `o`). Zapis ϵ predstavlja prazen niz.

Množico nizov imenujemo tudi *jezik*. Kartezični produkt jezikov imenujemo *stik*. Stik jezikov $\mathcal{A}$ in $\mathcal{B}$ ponavadi označujemo kot $\mathcal{AB}$ namesto kot $\mathcal{A} \times \mathcal{B}$. Definiran je kot $\mathcal{AB} = \{ab \mid a \in \mathcal{A} \wedge b \in \mathcal{B}\}$. Na primer, če velja $\mathcal{A} = \{$`ac`, `b`$\}$ in $\mathcal{B} = \{$`p`, ϵ, `q`$\}$, potem je $\mathcal{AB} = \{$`acp`, `ac`, `acq`, `bp`, `b`, `bq`$\}$. Velja $|\mathcal{AB}| = |\mathcal{A}| \cdot |\mathcal{B}|$. Zapis $\mathcal{A}^2$ predstavlja stik jezika $\mathcal{A}$ s samim seboj ($\mathcal{AA}$). Stik množice s samo seboj lahko posplošimo na poljuben ne-negativen »eksponent«: $\mathcal{A}^0 = \{\epsilon\}$, $\mathcal{A}^1 = \mathcal{A}$, $\mathcal{A}^k = \mathcal{A}\mathcal{A}^{k-1}$ (pri $k \geq 1$). Zapis $\mathcal{A}^*$ predstavlja *Kleenovo ovojnico* jezika $\mathcal{A}$, ki je definirana kot $\mathcal{A}^* = (\mathcal{A}^0 \cup \mathcal{A}^1 \cup \mathcal{A}^2 \cup \ldots)$. Na primer, če je $\mathcal{A} = \{$`a`, `b`$\}$, potem je $\mathcal{A}^* = \{\epsilon$, `a`, `b`, `aa`, `ab`, `ba`, `bb`, `aaa`, `aab`, `aba`, `abb`, `baa`, `bab`, `bba`, `bbb`, $\ldots\}$, skratka množica vseh možnih nizov, sestavljenih iz znakov `a` in `b`.

2.1.2 Preslikave in relacije

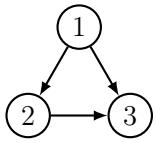
Delna (parcialna) preslikava $f\colon \mathcal{A} \to \mathcal{B}$ je predpis, ki vsakemu elementu množice $\mathcal{A}$ priredi največ po en element množice $\mathcal{B}$. *Popolna (totalna) preslikava* ali preprosto *preslikava* $f\colon \mathcal{A} \to \mathcal{B}$ je predpis, ki vsakemu elementu množice $\mathcal{A}$ priredi *natanko* po en element množice $\mathcal{B}$. Besedo *funkcija* bomo v doktorski disertaciji uporabljali kot sopomenko besede *preslikava*, vendar pa bomo ponekod raje govorili o preslikavah (npr. pri predpisih nad grafi), ponekod pa o funkcijah (npr. pri predpisih nad množicami števil).

Preslikava $f\colon \mathcal{A} \to \mathcal{B}$ je definirana na posameznih *elementih* množice $\mathcal{A}$, vendar pa jo bomo, kadar bo to smiselno, na naraven način razširili na *podmnožice* množice $\mathcal{A}$. Če velja $\mathcal{A}' \subseteq \mathcal{A}$, potem definiramo $f(\mathcal{A}') = \{f(a) \mid a \in \mathcal{A}'\}$. Na primer, če je $f(x) = x^2$, potem velja $f(\{-5, -2, 2\}) = \{4, 25\}$.

Pri fiksni *kratnosti* $k \geq 1$ je *relacija* $\mathbf{R}$ nad množico $\mathcal{A}$ definirana kot množica k-teric elementov iz $\mathcal{A}$ (torej podmnožica množice $\mathcal{A}^k$), za katere velja določena lastnost. Na primer, relacija $<$ (»manjši od«) nad množico $\{1, 2, 3\}$ je množica parov $\{(1,2), (1,3), (2,3)\}$, saj velja $1 < 2$, $1 < 3$ in $2 < 3$. Relacija $\mathbf{R}$ *velja* za neko k-terico $(a_1, \ldots, a_k)$ natanko tedaj, ko je $(a_1, \ldots, a_k) \in \mathbf{R}$. Rekli bomo tudi, da so elementi $a_1, \ldots, a_k$ *v relaciji* $\mathbf{R}$ oziroma da ima relacija $\mathbf{R}$ pri tej k-terici *resnično vrednost*. Če neka k-terica ne pripada relaciji $\mathbf{R}$, pa bomo zapisali, da relacija za to k-terico *ne velja* oziroma da ima *neresnično vrednost*. Relacijo $\mathbf{R}$ nad množico $\mathcal{A}$ *ovrednotimo* ali *izračunamo*, če določimo njene vrednosti pri vseh elementih množice $\mathcal{A}^k$. Zapis $\mathbf{R}(a_1, \ldots, a_k)$ je okrajšava zapisa $(a_1, \ldots, a_k) \in \mathbf{R}$. Pri *dvojiških* relacijah (relacijah s kratnostjo 2) bomo poleg $(a, b) \in \mathbf{R}$ in $\mathbf{R}(a, b)$ pisali tudi $a\,\mathbf{R}\,b$. Takšen način uporabljamo pri običajnih matematičnih relacijah, saj, denimo, pišemo $2 < 3$, ne $<(2,3)$ ali $(2,3) \in <$.

Če elementom množice $\mathcal{A}$ dodelimo zaporedne številke, potem lahko dvojiško relacijo $\mathbf{R}$ nad množico $\mathcal{A}$ zapišemo kot kvadratno dvojiško matriko z $|\mathcal{A}|$ vrsticami in stolpci. Element na mestu (i, j) v matriki ima vrednost 1, če je i-ti element množice $\mathcal{A}$ v relaciji z njenim j-tim elementom; sicer ima element vrednost 0. Dvojiško relacijo lahko predstavimo tudi z enostavnim usmerjenim grafom: vozlišča grafa pripadajo posameznim elementom množice $\mathcal{A}$, povezave pa dvojicam, ki so v relaciji. Na primer, relaciji $<$ nad množico $\{1, 2, 3\}$ ustrezata sledeča matrika in graf:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$



Dvojiška relacija $\mathbf{R}$ nad množico $\mathcal{A}$ je *refleksivna*, če velja $a\,\mathbf{R}\,a$ za vsak element $a \in \mathcal{A}$. Relacija $\mathbf{R}$ je *simetrična*, če za vsako dvojico a, $b \in \mathcal{A}$ velja $(a\,\mathbf{R}\,b \implies b\,\mathbf{R}\,a)$. Relacija $\mathbf{R}$ je *tranzitivna*, če za vsako trojico a, b, $c \in \mathcal{A}$ velja $(a\,\mathbf{R}\,b \wedge b\,\mathbf{R}\,c \implies a\,\mathbf{R}\,c)$.

Če je $\mathbf{R}$ dvojiška relacija nad množico $\mathcal{A}$, potem je njena *tranzitivna ovojnica* $\mathbf{R}^+$ definirana kot najmanjša tranzitivna relacija nad množico $\mathcal{A}$, ki v celoti vsebuje relacijo $\mathbf{R}$. (Ker so relacije množice, lahko tudi zanje definiramo velikost in vsebovanost. Velikost relacije je število k-teric, pri katerih ima relacija resnično vrednost.) Tranzitivno ovojnico relacije $\mathbf{R}$ nad množico $\mathcal{A}$ lahko zapišemo kot $\mathbf{R}^+ = \{(a,b) \in \mathcal{A}^2 \mid a\,\mathbf{R}\,b \vee (\exists c \in \mathcal{A}\colon a\,\mathbf{R}\,c \wedge c\,\mathbf{R}^+\,b)\}$. Če relacijo zapišemo kot matriko ali graf, lahko njeno tranzitivno ovojnico v času $O(|\mathcal{A}|^3)$ izračunamo s Floyd-Warshallovim algoritmom [19]. *Refleksivno-tranzitivna* ovojnica relacije $\mathbf{R}$ je relacija $\mathbf{R}^* = \mathbf{R}^+ \cup \{(a,a) \mid a \in \mathcal{A}\}$.

2.2 Grafi

V tem podpoglavju bomo formalno opredelili pojem grafa in tiste sorodne koncepte, ki jih bomo potrebovali v nadaljevanju doktorske disertacije. Matematiki graf pogosto definirajo kot dvojico $(\mathcal{V}, \mathcal{E})$, kjer je $\mathcal{V}$ množica *vozlišč*, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ pa množica *povezav*. Ta definicija je primerna za enostavne grafe brez zank in vzporednih povezav, mi pa se bomo ukvarjali tudi z grafi, ki lahko vsebujejo takšne povezave. Poleg tega bi želeli v definicijo grafa zajeti tudi oznake vozlišč in povezav. Sledeča definicija nam bo zato bolje služila:

Definicija 2.1 (graf)**.** *Graf* G je sedmerica $(\mathcal{V}[G], \mathcal{E}[G], \mathit{source}[G], \mathit{target}[G], \mathit{conn}[G], \mathit{Labels}[G], \mathit{label}[G])$, pri čemer:

- $\mathcal{V}[G]$ je množica *vozlišč* grafa G.
- $\mathcal{E}[G]$ je množica *povezav* grafa G.
- $\mathit{source}[G]\colon \mathcal{E}[G] \to \mathcal{V}[G]$ je funkcija, ki vsaki povezavi priredi njeno *izvorno vozlišče*.
- $\mathit{target}[G]\colon \mathcal{E}[G] \to \mathcal{V}[G]$ je funkcija, ki vsaki povezavi priredi njeno *ciljno vozlišče*.
- $\mathit{conn}[G]\colon \mathcal{E}[G] \to \mathcal{P}(\mathcal{V}[G])$ je funkcija, ki vsaki povezavi priredi množico vozlišč, ki jih povezuje. Elementa množice $\mathit{conn}[G](e)$ bomo označili s $\mathit{conn}_1[G](e)$ in $\mathit{conn}_2[G](e)$. Če je povezava e zanka, potem velja $\mathit{conn}_1[G](e) = \mathit{conn}_2[G](e)$, množica $\mathit{conn}[G](e)$ pa vsebuje en sam element.
- $\mathit{Labels}[G]$ je množica *oznak*.

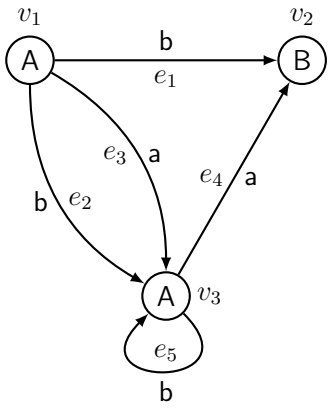
- $label[G]\colon \mathcal{V}[G] \cup \mathcal{E}[G] \to Labels[G]$ je funkcija, ki vsakemu vozlišču in vsaki povezavi priredi oznako.

Predpostavili bomo, da množica $Labels[G]$ vsebuje samo tiste oznake, ki v grafu dejansko nastopajo:

$$Labels[G] = \{l \mid (\exists v \in \mathcal{V}[G]\colon l = label(v)) \vee (\exists e \in \mathcal{E}[G]\colon l = label(e))\} \qquad (2.1)$$

Zapis $\mathcal{V}[G]$, $\mathcal{E}[G]$ itd. bomo okrajšali v $\mathcal{V}$, $\mathcal{E}$ itd., kadar nas graf, na katerega se množica ali funkcija nanaša, ne bo zanimal ali pa bo znan iz konteksta.[1] Funkciji *source* in *target* sta smiselni samo pri usmerjenih grafih, tj. grafih, kjer razlikujemo med izvornimi in ciljnimi vozlišči posameznih povezav. Pri neusmerjenih grafih pa pojma izvornega in ciljnega vozlišča nista določena, saj sta za vsako povezavo pomembni le identiteti vozlišč, ki ju povezuje. Zato bomo pri neusmerjenih grafih predpostavili, da funkciji *source* in *target* nista definirani. Funkcija *conn* pa je smiselna za obe vrsti grafov. Pri usmerjenih grafih je definirana kot $conn(e) = \{source(e), target(e)\}$.

Slika 2.1 prikazuje graf, za katerega velja $\mathcal{V} = \{v_1, v_2, v_3\}$, $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5\}$, $source = \{e_1 \mapsto v_1, e_2 \mapsto v_1, e_3 \mapsto v_1, e_4 \mapsto v_3, e_5 \mapsto v_3\}$, $target = \{e_1 \mapsto v_2, e_2 \mapsto v_3, e_3 \mapsto v_3, e_4 \mapsto v_2, e_5 \mapsto v_3\}$, $conn = \{e_1 \mapsto \{v_1, v_2\}, e_2 \mapsto \{v_1, v_3\}, e_3 \mapsto \{v_1, v_3\}, e_4 \mapsto \{v_2, v_3\}, e_5 \mapsto \{v_3\}\}$, $Labels = \{\mathsf{A}, \mathsf{B}, \mathsf{a}, \mathsf{b}\}$ in $label = \{v_1 \mapsto \mathsf{A}, v_2 \mapsto \mathsf{B}, v_3 \mapsto \mathsf{A}, e_1 \mapsto \mathsf{b}, e_2 \mapsto \mathsf{b}, e_3 \mapsto \mathsf{a}, e_4 \mapsto \mathsf{a}, e_5 \mapsto \mathsf{b}\}$.



Slika 2.1: *Primer grafa.*

Definicija 2.2 (prazen graf)**.** *Prazen graf* je graf λ z lastnostjo $\mathcal{V}[\lambda] = \mathcal{E}[\lambda] = \emptyset$.

Definicija 2.3 (grafni elementi)**.** *Množica elementov* grafa G je definirana kot $Elements[G] \equiv \mathcal{V}[G] \cup \mathcal{E}[G]$. *Grafni element* je element množice *Elements*, torej vozlišče ali povezava.

[1] Zakaj smo se pri zapisu sestavnih delov grafa ($\mathcal{V}[G]$, $\mathcal{E}[G]$ itd.) odločili za uporabo oglatih oklepajev namesto običajnejših okroglih oklepajev ($\mathcal{V}(G)$) ali indeksov ($\mathcal{V}_G$)? Okrogli oklepaji se nam ne zdijo najprimernejši, saj je množica $\mathcal{V}[G]$ sestavni del grafa G, ne funkcija ali preslikava nad grafom G, na kar bi namigoval zapis $\mathcal{V}(G)$. Indeksi pa kaj kmalu postanejo nepregledni, še zlasti, če jih kopičimo. Tudi v nadaljnjem besedilu bomo namesto indeksov pogosto uporabljali zapis z oglatimi oklepaji.

Namesto $x \in \mathit{Elements}[G]$ bomo v prid berljivosti pisali kar $x \in G$. Ta zapis ni povsem natančen, saj graf strogo vzeto ni množica grafnih elementov, pač pa sedmerica iz definicije 2.1, ki *vsebuje* množici vozlišč in povezav. Kljub temu pa zapis $x \in G$ ne bo nikoli povzročal dvoumnosti.

Elementi grafa so lahko označeni ali neoznačeni. Kljub temu pa je koristno, da funkcijo *label* definiramo za vse elemente. Dogovorimo se, da za vse neoznačene elemente velja $\mathit{label}(x) = \#$. Oznaka # ne bo igrala nobene druge vloge.

Vsak grafni element ima enolično določen *identifikator*. Identifikatorji elementov v grafu na sliki 2.1 so v_1, v_2, v_3 in e_1, …, e_5. Oznake elementov pa niso nujno enolične; lahko imajo tudi vsi grafni elementi isto oznako.

Besedna zveza »element x« predstavlja element z *identifikatorjem* x, ne *oznako* x. Za graf na sliki 2.1 bi torej lahko zapisali, da vsebuje »povezavo e_1« in »dve povezavi *z oznakama* a«. Vendar pa nas bo pogosto zanimala le oznaka elementa, njegov identifikator v grafu pa ne bo pomemben. V takšnih primerih bi ponavljanje prilastka »z oznako« kaj kmalu postalo utrujajoče, zato si bomo pogosto privoščili majhno malomarnost. Kadar bo povsem jasno, da govorimo o oznaki, ne identiteti elementa, bomo besedno zvezo »element z oznako x« okrajšali v »element x«. Dobesedne oznake bomo tako v besedilu kot na slikah zapisovali s pokončno neserifno pisavo. *Poševno serifno* pisavo pa bomo uporabljali za identifikatorje in za spremenljivke, katerih vrednosti so oznake; na primer, trditev »vozlišče v ima oznako x« pove, da je oznaka vozlišča z identifikatorjem v zapisana v spremenljivki x. Oznaka vozlišča bo na slikah prikazana kot napis znotraj geometrijskega lika, ki bo predstavljal vozlišče, identifikator (kadar ga bomo prikazali) pa bo prikazan ob robu lika.

Definicija 2.4 (sosednost)**.** Vozlišče v je *sosed* vozlišča u natanko tedaj, ko obstaja povezava e, tako da velja $u = \mathit{source}(e)$ in $v = \mathit{target}(e)$ (v usmerjenem grafu) oziroma $\mathit{conn}(e) = \{u, v\}$ (v neusmerjenem grafu).

Na primer, v grafu s slike 2.1 je vozlišče v_3 sosed vozlišča v_1, obratno pa ne drži.

Definicija 2.5 (odnos med povezavo in vozliščem)**.** Povezava e *vstopa* v vozlišče v, če je $v = \mathit{target}(e)$. Povezava e *izstopa* iz vozlišča v (ali *izvira* v vozlišču v), če je $v = \mathit{source}(e)$. Povezava e je *pripeta* na vozlišče v, če je $v \in \mathit{conn}(e)$.

Definicija 2.6 (sprehod)**.** *Sprehod* v grafu G je zaporedje (v_1, e_1, v_2, e_2, v_3, …, e_{n-1}, v_n), tako da velja $v_i = \mathit{source}(e_i)$ in $v_{i+1} = \mathit{target}(e_i)$ za vse $i \in 1\,..\,n-1$ (v primeru usmerjenega grafa) oziroma $\mathit{conn}(e_i) = \{v_i, v_{i+1}\}$ za vse $i \in 1\,..\,n-1$ (v primeru neusmerjenega grafa).

Sprehod (v_1, e_1, v_2, e_2, v_3, …, e_{n-1}, v_n) bomo pregledneje zapisali kot $v_1 \xrightarrow{e_1} v_2 \xrightarrow{e_2} v_3 \ldots \xrightarrow{e_{n-1}} v_n$. Na primer, v grafu s slike 2.1 lahko opravimo sprehod $v_1 \xrightarrow{e_3} v_3 \xrightarrow{e_5} v_3 \xrightarrow{e_4} v_2$.

Definicija 2.7 (podgraf)**.** Graf G je *podgraf* grafa H ($G \sqsubseteq H$) natanko tedaj, ko velja $\mathcal{V}[G] \subseteq \mathcal{V}[H]$, $\mathcal{E}[G] \subseteq \mathcal{E}[H]$ in $\mathit{Labels}[G] \subseteq \mathit{Labels}[H]$ ter $\mathit{source}[G](e) = \mathit{source}[H](e)$, $\mathit{target}[G](e) = \mathit{target}[H](e)$, $\mathit{conn}[G](e) = \mathit{conn}[H](e)$ in $\mathit{label}[G](x) = \mathit{label}[H](x)$ za vse $e \in \mathcal{E}[G]$ in $x \in G$.

Primer podgrafa grafa s slike 2.1 je graf, sestavljen iz njegovih vozlišč v_1 in v_3 ter povezav e_3 in e_5.

Definicija 2.8 (presek grafov)**.** *Presek* grafov G in H je graf $P = G \cap H$, ki je definiran tako:

- $\mathcal{V}[P] = \mathcal{V}[G] \cap \mathcal{V}[H]$;
- $\mathcal{E}[P] = \mathcal{E}[G] \cap \mathcal{E}[H]$;
- $source[P](e) = source[G](e)$ za vsak $e \in \mathcal{E}[P]$;
- $target[P](e) = target[G](e)$ za vsak $e \in \mathcal{E}[P]$;
- $conn[P](e) = conn[G](e)$ za vsak $e \in \mathcal{E}[P]$;
- $Labels[P] = Labels[G] \cap Labels[H]$;
- $label[P](x) = label[G](x)$ za vsak $x \in P$.

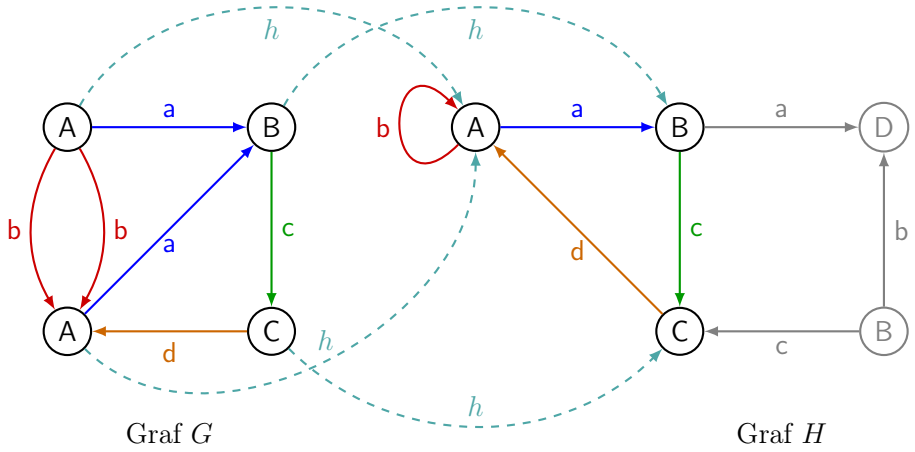
Presek grafov G in H je torej sestavljen iz vseh elementov, ki so skupni obema grafoma. Funkcije *source*, *target*, *conn* so v grafu $G \cap H$ definirane enako kot v grafih G in H, le da so omejene na skupne elemente obeh grafov. Na podoben način lahko definiramo tudi *unijo* in *razliko* grafov. V nasprotju s presekom in unijo pa razlika grafov ni nujno veljaven graf. Na primer, naj bosta grafa G in H podgrafa grafa s slike 2.1. Graf G naj bo sestavljen iz njegovih elementov v_1, v_2 in e_1, graf H pa naj vsebuje samo vozlišče v_1. Razlika grafov G in H potemtakem vsebuje elementa v_2 in e_1, ki pa ne tvorita veljavnega grafa, saj izvorno krajišče povezave e_1 (vozlišče v_1) ne pripada razliki. Z drugimi besedami: v razliki $G \setminus H$ vrednost $source(e_1)$ ni definirana. Pravimo, da je povezava e_1 *viseča povezava* (angl. *dangling edge*). Veljaven graf ne sme vsebovati nobene viseče povezave.

Množice grafnih elementov, ki vsebujejo viseče povezave, bomo — kjer bo to smiselno — obravnavali na enak način kot množice, ki tvorijo veljavne grafe. Na primer, za množico elementov $\mathcal{A} = \{v_2, e_1\}$ iz grafa s slike 2.1 lahko definiramo množici $\mathcal{V}[\mathcal{A}] = \{v_2\}$ in $\mathcal{E}[\mathcal{A}] = \{e_1\}$. Prav tako lahko določimo $target[\mathcal{A}](e_1) = v_2$ in $label[\mathcal{A}](e_1) = \mathsf{b}$. Definirati je mogoče celo $source[\mathcal{A}](e_1) = v_1$ in $conn[\mathcal{A}](e_1) = \{v_1, v_2\}$, čeprav vozlišče v_1 ne pripada množici $\mathcal{A}$.

Definicija 2.9 (homomorfizem)**.** *Homomorfizem* $h\colon G \to H$ je popolna preslikava vozlišč grafa G v vozlišča grafa H in povezav grafa G v povezave grafa H, tako da za vse $x \in G$ in $e \in \mathcal{E}[G]$ velja $label[H](h(x)) = label[G](x)$ in $conn[H](h(e)) = h(conn[G](e))$, v primeru usmerjenih grafov pa poleg tega velja še $source[H](h(e)) = h(source[G](e))$ in $target[H](h(e)) = h(target[G](e))$.

Homomorfizem je torej preslikava, ki ohranja sosednosti in oznake. Če je vozlišče izvor oz. cilj neke povezave v grafu G, bo to veljalo tudi za njuni sliki v grafu H. Če je vozlišče v v grafu G sosed vozlišča u, bo tudi vozlišče $h(v)$ v grafu H sosed vozlišča $h(u)$. Obratno pa ni nujno res: nesosednji vozlišči se lahko preslikata v sosednji. Prav tako lahko homomorfizem več enako označenih vozlišč oz. povezav preslika v eno samo vozlišče oz. povezavo. Slika 2.2 prikazuje primer homomorfizma med

dvema grafoma. Preslikave vozlišč so označene s črtkanimi krivuljami, preslikave povezav pa z različnimi barvami. Na primer, obe vozlišči z oznako A v grafu G se preslikata v isto vozlišče v grafu H. Podobno velja tudi za povezavi z oznako a in povezavi z oznako b.



Slika 2.2: *Primer homomorfizma.*

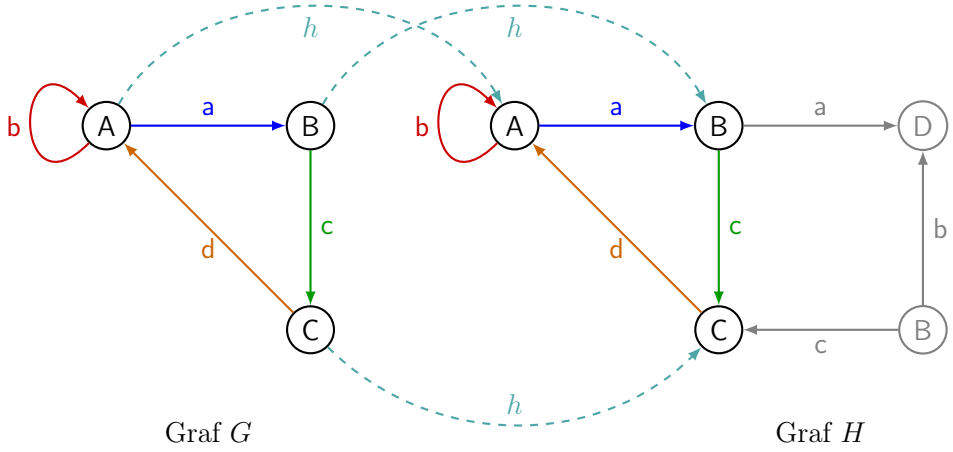
Homomorfizem bomo na naraven način razširili na množice grafnih elementov in na podgrafe. Naj bo $h\colon G \to H$ homomorfizem. Če je $\mathcal{A} \subseteq \mathit{Elements}[G]$ neka podmnožica elementov grafa G, potem velja $h(\mathcal{A}) = \{h(x) \mid x \in \mathcal{A}\}$. Homomorfizem $h\colon G \to H$ preslika graf G v graf $H' \sqsubseteq H$ $(h(G) = H')$, za katerega velja $\mathcal{V}[H'] = h(\mathcal{V}[G])$, $\mathcal{E}[H'] = h(\mathcal{E}[G])$ in $\mathit{Labels}[H'] = \mathit{Labels}[G]$, za vsak $x \in H'$ in $e \in \mathcal{E}[H']$ pa velja $\mathit{source}[H'](e) = \mathit{source}[H](e)$, $\mathit{target}[H'](e) = \mathit{target}[H](e)$, $\mathit{conn}[H'](e) = \mathit{conn}[H](e)$ in $\mathit{label}[H'](x) = \mathit{label}[H](x)$.

Ker je homomorfizem preslikava, lahko poleg »običajnega« (popolnega) homomorfizma definiramo tudi *delni homomorfizem*. Delni homomorfizem $h\colon G \to H$ je definiran na enak način kot popolni, le da ne zahtevamo, da imajo vsi elementi grafa G svojo sliko v grafu H. Za elemente, ki jih delni homomorfizem preslika, pa morajo veljati vse lastnosti, ki definirajo homomorfizem. V nadaljevanju bomo z besedo *homomorfizem* dosledno označevali popoln homomorfizem; kadar bomo govorili o delnem homomorfizmu, bomo to izrecno poudarili.

Definicija 2.10 (monomorfizem)**.** *Monomorfizem* je injektiven homomorfizem. Homomorfizem $h\colon G \to H$ je monomorfizem natanko tedaj, ko za vsako dvojico *različnih* elementov $x \in G$ in $y \in G$ velja $h(x) \neq h(y)$.

Pri monomorfizmu se torej ne more zgoditi, da bi se več vozlišč ali povezav preslikalo v eno samo vozlišče ali povezavo. Primer monomorfizma med dvema grafoma je prikazan na sliki 2.3.

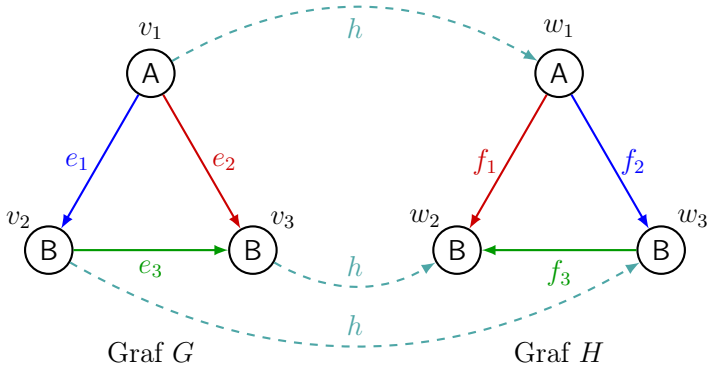
Definicija 2.11 (pojavitev)**.** Naj bosta G in H grafa, $h\colon G \to H$ pa monomorfizem. Potem je graf $h(G) \sqsubseteq H$ *pojavitev* grafa G v grafu H.



Slika 2.3: *Primer monomorfizma.*

Definicija 2.12 (izomorfizem)**.** *Izomorfizem* je homomorfizem, ki je tako injektiven kot surjektiven.

Izomorfizem je torej povratno enolična preslikava, ki ohranja sosednosti in oznake. Izomorfizem $h\colon G \to H$ torej vsakemu elementu grafa G priredi enolično določen element grafa H, prav tako pa za vsak element y v grafu H obstaja enolično določen element x v grafu G, tako da velja $h(x) = y$. Izomorfizem je možen le med grafoma, ki imata enako število vozlišč in povezav. Primer izomorfizma med dvema grafoma je prikazan na sliki 2.4. Naj omenimo, da preslikava $h' = \{v_1 \mapsto w_1,\ v_2 \mapsto w_2,\ v_3 \mapsto w_3,\ e_1 \mapsto f_1,\ e_2 \mapsto f_2,\ e_3 \mapsto f_3\}$ ne bi bila izomorfizem (niti homomorfizem), saj bi zanjo veljalo $w_3 = source(h'(e_3)) \neq h'(source(e_3)) = w_2$, kar je v nasprotju z definicijo homomorfizma.



Slika 2.4: *Primer izomorfizma.*

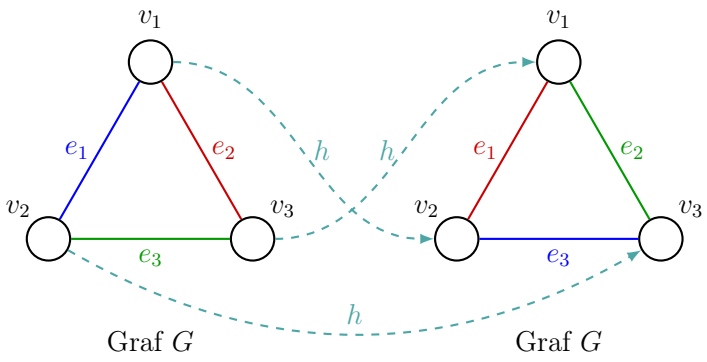
Ker je monomorfizem $h\colon G \to H$ izomorfizem med grafom G in nekim podgrafom grafa H, pravimo monomorfizmu tudi *podgrafni izomorfizem* (angl. *subgraph isomorphism*) [94].

Definicija 2.13 (izomorfna grafa)**.** Grafa G in H sta *izomorfna* natanko tedaj, ko med njima obstaja izomorfizem.

Pojavitev grafa G v grafu H (definicija 2.11) je potemtakem podgraf grafa H, ki je izomorfen grafu G.

Definicija 2.14 (avtomorfizem)**.** *Avtomorfizem* je izomorfizem v okviru istega grafa. Avtomorfizem v grafu G je izomorfizem $h\colon G \to G$.

Avtomorfizem si lahko predstavljamo tudi kot izomorfizem med dvema kopijama istega grafa. Primer avtomorfizma je prikazan na sliki 2.5. Z avtomorfizmi se bomo ukvarjali v poglavju 3, saj ena od izboljšav Rekers-Schürrovega sintaksnega analizatorja temelji na odpravljanju podvajanja, ki je posledica avtomorfizmov v produkcijah.



Slika 2.5: *Primer avtomorfizma.*

2.3 Grafne gramatike

V tem podpoglavju bomo na splošno opredelili grafne gramatike in pregledali nekaj njihovih oblik. Kontekstno odvisne grafne gramatike bomo predstavili v ločenem podpoglavju (2.4), saj jih bomo uporabljali v vseh nadaljnjih poglavjih doktorske disertacije.

Besedilnim gramatikam bomo kljub njihovi splošni poznanosti naklonili krajši sestavek (razdelek 2.3.1), saj mnogi koncepti iz njihovega sveta nastopajo tudi v svetu grafnih gramatik. V razdelku 2.3.2 bomo navedli nekaj skupnih značilnosti grafnih gramatik. V razdelkih 2.3.3, 2.3.4 in 2.3.5 bomo predstavili tri različne oblike grafnih gramatik, ki jih lahko obravnavamo kot »čiste« grafne posplošitve besedilnih gramatik. V razdelku 2.3.6 bomo navedli še nekaj oblik grafnih gramatik in formalizmov, sorodnih grafnim gramatikam.

2.3.1 Besedilne gramatike

Ta razdelek predstavlja osnovne koncepte s področja besedilnih gramatik. Ker so se besedilne gramatike tako v teoriji kot v praksi že zdavnaj uveljavile, se ne bomo

sklicevali na izvirne vire. Bralec jih lahko poišče, denimo, v knjigi Hopcrofta in sod. [45].

Besedilna gramatika je sistem pravil za zamenjevanje nizov. Formalno jo lahko zapišemo kot četverico $(\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$, kjer je $\mathcal{N}$ množica *nekončnih simbolov*, $\mathcal{T}$ množica *končnih simbolov*, $\mathcal{P}$ množica zamenjevalnih pravil (*produkcij*), $S \in \mathcal{N}$ pa je *začetni simbol* gramatike. Vsaka produkcija zavzema obliko $\alpha ::= \beta$, kjer velja $\alpha \in (\mathcal{N} \cup \mathcal{T})^* \mathcal{N} (\mathcal{N} \cup \mathcal{T})^*$ in $\beta \in (\mathcal{N} \cup \mathcal{T})^*$. (V teh zapisih nastopata stik množic in Kleenova ovojnica, ki smo ju predstavili v razdelku 2.1.1.) Obe strani produkcije sta torej sestavljeni iz poljubnega zaporedja končnih in nekončnih simbolov, vendar pa se mora na levi strani nahajati vsaj en nekončni simbol.

Produkcijo $\alpha ::= \beta$ lahko *uporabimo* na nizu γ, če ta vsebuje vsaj en podniz, ki je enak nizu α. To storimo tako, da v nizu γ izberemo nek podniz, ki je enak nizu α, odstranimo izbrani podniz iz niza γ, nato pa na njegovo mesto vstavimo niz β. Na primer, če produkcijo $\mathsf{aB} ::= \mathsf{Cd}$ uporabimo na nizu aBBaaB, dobimo bodisi niz CdBaaB ali niz aBBaCd. Zapis $\gamma \overset{p}{\Rightarrow} \delta$ označuje, da z uporabo produkcije p preoblikujemo niz γ v niz δ. Če identiteta produkcije ni pomembna, pišemo $\gamma \Rightarrow \delta$. Z zapisom $\gamma \Rightarrow^* \delta$ pa označujemo, da obstaja (morebiti prazno) zaporedje uporab produkcij iz dane gramatike, ki niz γ preoblikuje v niz δ. Na primer, pri gramatiki s produkcijama $\mathsf{S} ::= \mathsf{ab}$ in $\mathsf{S} ::= \mathsf{aSb}$, kjer je nekončni simbol S začetni simbol gramatike, a in b pa sta končna simbola, lahko zapišemo $\mathsf{S} \Rightarrow \mathsf{aSb} \Rightarrow \mathsf{aaSbb} \Rightarrow \mathsf{aaaSbbb} \Rightarrow \mathsf{aaaabbbb}$ oziroma $\mathsf{S} \Rightarrow^* \mathsf{aaaabbbb}$.

Zaporedje uporab produkcij, ki iz začetnega simbola gramatike privede do niza δ, imenujemo *izpeljava* niza δ. Niz δ je *izpeljiv* v dani gramatiki, če obstaja izpeljava zanj, torej če velja $S \Rightarrow^* \delta$. *Jezik* gramatike G (oznaka: $L(G)$) je množica vseh izpeljivih nizov, ki so sestavljeni zgolj iz končnih simbolov gramatike. Formalno:

$$L(G) = \{w \in \mathcal{T}^* \mid S \Rightarrow^* w\} \tag{2.2}$$

Rekli bomo, da gramatika *generira* svoj jezik. Na primer, zgoraj navedena gramatika s produkcijama $\mathsf{S} ::= \mathsf{ab}$ in $\mathsf{S} ::= \mathsf{aSb}$ generira jezik $\{\mathsf{a}^n\mathsf{b}^n \mid n \geq 1\}$. Pojem jezika lahko uporabljamo tudi brez povezave z gramatikami. V tem primeru ta pojem predstavlja množico nizov, sestavljenih zgolj iz končnih simbolov.

Besedilne gramatike so glede na moč njihovih jezikov urejene v *hierarhijo Chomskega*. Če razred gramatik $\mathcal{G}$ v hierarhiji Chomskega leži nižje od razreda gramatik $\mathcal{G}'$, potem to pomeni dvoje:

- Za vsak jezik L, ki ga generira neka gramatika v razredu $\mathcal{G}$, obstaja gramatika $G' \in \mathcal{G}'$ z lastnostjo $L(G') = L$.
- Obstaja vsaj en jezik L', ki ga generira neka gramatika v razredu $\mathcal{G}'$, vendar pa ga ne generira nobena gramatika v razredu $\mathcal{G}$.

Z drugimi besedami: če z $\mathcal{L}(\mathcal{G})$ označimo razred vseh jezikov, ki jih generirajo gramatike iz razreda $\mathcal{G}$, in če razred $\mathcal{G}_1$ v hierarhiji Chomskega leži nižje od razreda $\mathcal{G}_2$, potem velja $\mathcal{L}(\mathcal{G}_1) \subset \mathcal{L}(\mathcal{G}_2)$.

Hierarhija Chomskega vsebuje štiri razrede gramatik:

Regularne gramatike. Ta razred vsebuje gramatike, ki so sestavljene zgolj iz produkcij oblike $A ::= a$ in $A ::= aB$, pri čemer je a končni simbol, A in B pa sta nekončna simbola. Regularne gramatike generirajo *regularne jezike*.

Kontekstno neodvisne gramatike. Ta razred vsebuje gramatike s produkcijami oblike $A ::= \alpha$, kjer je A nek nekončni simbol, $\alpha \in (\mathcal{N} \cup \mathcal{T})^*$ pa je poljubno zaporedje končnih in nekončnih simbolov. Kontekstno neodvisne gramatike generirajo *kontekstno neodvisne jezike*.

Kontekstno odvisne gramatike. Ta razred vsebuje gramatike s produkcijami oblike $\alpha A \beta ::= \alpha \gamma \beta$, kjer je A nekončni simbol, α, β in γ pa so zaporedja končnih in nekončnih simbolov. Zaporedji α in β lahko prazni, zaporedje γ pa mora biti neprazno. Dovoljene so tudi produkcije oblike $A ::= \epsilon$, vendar le pod pogojem, da nekončni simbol A ne nastopa na desni strani nobene produkcije. Kontekstno odvisne gramatike generirajo *kontekstno odvisne jezike*.

Neomejene gramatike (gramatike tipa 0). Ta razred vsebuje vse besedilne gramatike. Neomejene gramatike generirajo *rekurzivno naštevne* (ali *Turingove*) *jezike*.

V zvezi z naštetimi razredi gramatik lahko definiramo številne odločljivostne probleme. Za nas bo najzanimivejši *problem pripadnosti* (angl. *membership problem*). Problem pripadnosti je definiran takole:

> Podani sta gramatika G in beseda w. Ali beseda w pripada jeziku gramatike G?

Problem pripadnosti je tesno povezan s pojmom sintaksne analize, ki smo ga neformalno predstavili v razdelku 1.1.2. Razlika je le v tem, da nas pri problemu pripadnosti zanima le odgovor na vprašanje $w \in L(G)$, pri sintaksni analizi pa nas v primeru pritrdilnega odgovora na to vprašanje zanima tudi izpeljava besede w v gramatiki G.

Če gramatika G pripada razredu regularnih gramatik ali razredu kontekstno neodvisnih gramatik, potem je problem pripadnosti mogoče rešiti v času, ki polinomsko narašča z dolžino besede w in velikostjo gramatike G. Problema pripadnosti (in obenem tudi sintaksne analize) za tovrstne gramatike se lahko lotimo, denimo, z algoritmom CYK [101] ali z Earleyjevim sintaksnim analizatorjem [27]. Pri konstantni velikosti gramatike oba algoritma dosegata asimptotično časovno zahtevnost $O(|w|^3)$, kjer je $|w|$ dolžina vhodne besede w. Pri kontekstno odvisnih gramatikah oz. jezikih je problem pripadnosti *PSPACE*-poln, kar pomeni, da ga najbrž ni mogoče reševati v polinomskem času; če bi to bilo mogoče, bi namreč veljalo $P = NP$. Pri rekurzivno naštevnih jezikih pa problem pripadnosti v splošnem sploh ni odločljiv. Za vsak rekurzivno naštevni jezik sicer obstaja Turingov stroj, ki se bo z odgovorom »da« ustavil pri vseh vhodnih besedah, ki pripadajo jeziku. Ni pa nujno, da obstaja Turingov stroj, ki se bo poleg tega z odgovorom »ne« ustavil pri vseh nepripadajočih besedah.

2.3.2 Splošno o grafnih gramatikah

Grafne gramatike [81] so posplošitev besedilnih. Lahko jih definiramo na enak način kot besedilne, le da v njih namesto nizov in podnizov nastopajo grafi in podgrafi, namesto simbolov pa oznake grafnih elementov — vozlišč in povezav. Grafno gramatiko lahko potemtakem definiramo kot četverico $(\mathcal{N}, \mathcal{T}, \mathcal{P}, S)$, kjer je $\mathcal{N}$ množica

nekončnih oznak grafnih elementov, $\mathcal{T}$ množica *končnih oznak* grafnih elementov, $\mathcal{P}$ množica *produkcij*, S pa je oznaka *začetnega grafnega elementa*. Vse produkcije imajo obliko $L ::= R$, pri čemer sta L in R v splošnem poljubna grafa. Za razliko od produkcij besedilnih gramatik pri produkcijah grafnih gramatik ne zahtevamo, da morajo njihove leve strani vsebovati najmanj en grafni element z nekončno oznako. (Če smo natančnejši: to zahtevamo pri kontekstno neodvisnih grafnih gramatikah, pri kontekstno odvisnih pa dovoljujemo tudi produkcije, katerih leve strani vsebujejo zgolj elemente s končnimi oznakami.) Namesto množic $\mathcal{N}$ in $\mathcal{T}$ lahko definiramo ločene množice oznak za vozlišča in povezave. Prav tako lahko namesto začetnega grafnega elementa (S) definiramo tudi začetni graf (aksiom) gramatike ali pa množico aksiomov, kot smo to storili v uvodni predstavitvi grafnih gramatik v razdelku 1.1.1.

Glavna razlika med besedilnimi in grafnimi gramatikami se nahaja v samem postopku zamenjevanja podnizov oz. podgrafov. Če želimo v nizu $\alpha\beta\gamma$ zamenjati podniz β z nizom δ, je postopek enolično določen: iz izvornega niza odstranimo podniz β, s čimer dobimo niz $\alpha\gamma$, nato pa na mesto odstranjenega niza (torej med niza α in γ) vstavimo niz δ, da dobimo končni niz $\alpha\delta\gamma$. Pri grafih pa navodilo »v grafu H zamenjaj podgraf G z grafom G'« v splošnem ni dovolj dobro definirano. Denimo, da je podgraf G sestavljen zgolj iz enega samega vozlišča (npr. v). Kako naj v grafu H zamenjamo vozlišče v z grafom G'? Kaj naj storimo s povezavami vozlišča v, ko to vozlišče odstranimo iz grafa H? Kako naj graf G' vpnemo v graf H? Mar v ta namen ustvarimo nove povezave ali zgolj uporabimo povezave, ki so pripadale vozlišču v? Ker na tovrstna vprašanja ni enoznačnih odgovorov, obstaja veliko različnih definicij grafnih gramatik.

Izbrane oblike grafnih gramatik so v smiselno hierarhijo, podobno hierarhiji Chomskega, poskusili urediti Blostein in sod. [9]. *Priročnik* [81] se s splošno hierarhijo ne ukvarja, predstavlja pa hierarhijo posameznih oblik grafnih gramatik, zlasti vozliščnih gramatik. Za nas bosta zanimivi le sledeči skupini grafnih gramatik:

Kontekstno neodvisne grafne gramatike (angl. *context-free graph grammars*): Po analogiji s kontekstno neodvisnimi besedilnimi gramatikami vsaka produkcija tovrstnih gramatik zamenja en sam grafni element (vozlišče ali povezavo) z nekim grafom. Glede na to, ali produkcije podajajo pravila za zamenjevanje vozlišč ali za zamenjevanje povezav, ločimo *vozliščne gramatike* ali natančneje *gramatike z zamenjavami vozlišč* (angl. *node replacement grammars*) [71] in *(hiper)povezavne gramatike* ali natančneje *gramatike z zamenjavami (hiper)povezav* (angl. *(hyper)edge replacement grammars*) [40]. V razdelkih 2.3.3 in 2.3.4 bomo predstavili obe vrsti gramatik.

Kontekstno neodvisne grafne gramatike so v nekaterih vidikih podobne svojim besedilnim soimenjakinjam. Značilna skupna točka je možnost predstavitve izpeljave niza oz. (hiper)grafa z *drevesom izpeljav* (angl. *parse tree*). Drevo izpeljav je drevo, ki ima v korenu začetni simbol (oz. začetni (hiper)grafni element) gramatike, njegovi otroci pa so končni in nekončni simboli (oz. (hiper)grafni elementi), ki se razvijejo iz začetnega simbola v enem koraku izpeljave. Otroci vsakega nekončno označenega otroka korena so definirani na enak način kot

otroci korena (in tako naprej za nekončno označene otroke teh otrok itd.). Listi drevesa vsebujejo množico končnih simbolov (oz. (hiper)grafnih elementov s končnimi oznakami) celotnega izpeljanega niza (oz. (hiper)grafa).

Kontekstno odvisne grafne gramatike (angl. *context-sensitive graph grammars*) [80]: Tovrstne gramatike vsebujejo produkcije za zamenjavo podgrafov. Na kratko jih bomo predstavili v razdelku 2.3.5, obširneje pa se jim bomo posvetili v podpoglavju 2.4.

2.3.3 Vozliščne gramatike

Vsaka produkcija vozliščne gramatike podaja pravilo za zamenjavo nekega vozlišča z grafom. Ta zamenjava je smiselna le v primeru, če dodani graf (tj. graf na desni strani produkcije) na nek način vpnemo v gostiteljski graf (tj. graf, na katerem uporabimo produkcijo). To storimo tako, da vozlišča dodanega grafa v skladu s posebnimi pravili povežemo z vozlišči, s katerimi je bilo povezano vozlišče, ki smo ga pri zamenjavi odstranili. Pravila, ki določajo povezave dodanega grafa s sosedi odstranjenega vozlišča, se imenujejo *vložitvena pravila* (angl. *embedding rules*).

Vsaka produkcija vozliščne gramatike zavzema obliko $X ::= (D, \mathcal{C})$, kjer je X nekončna oznaka vozlišča, D graf, $\mathcal{C}$ pa množica vložitvenih pravil za povezavo vozlišč grafa G s soseščino odstranjenega vozlišča z oznako A. Produkcijo $X ::= (D, \mathcal{C})$ na gostiteljskem grafu H uporabimo na sledeči način:

1. V grafu H poiščemo vozlišče z oznako X. Če takšnega vozlišča ni, produkcije ne moremo uporabiti. Če je takšnih vozlišč več, lahko izberemo katerokoli od njih.

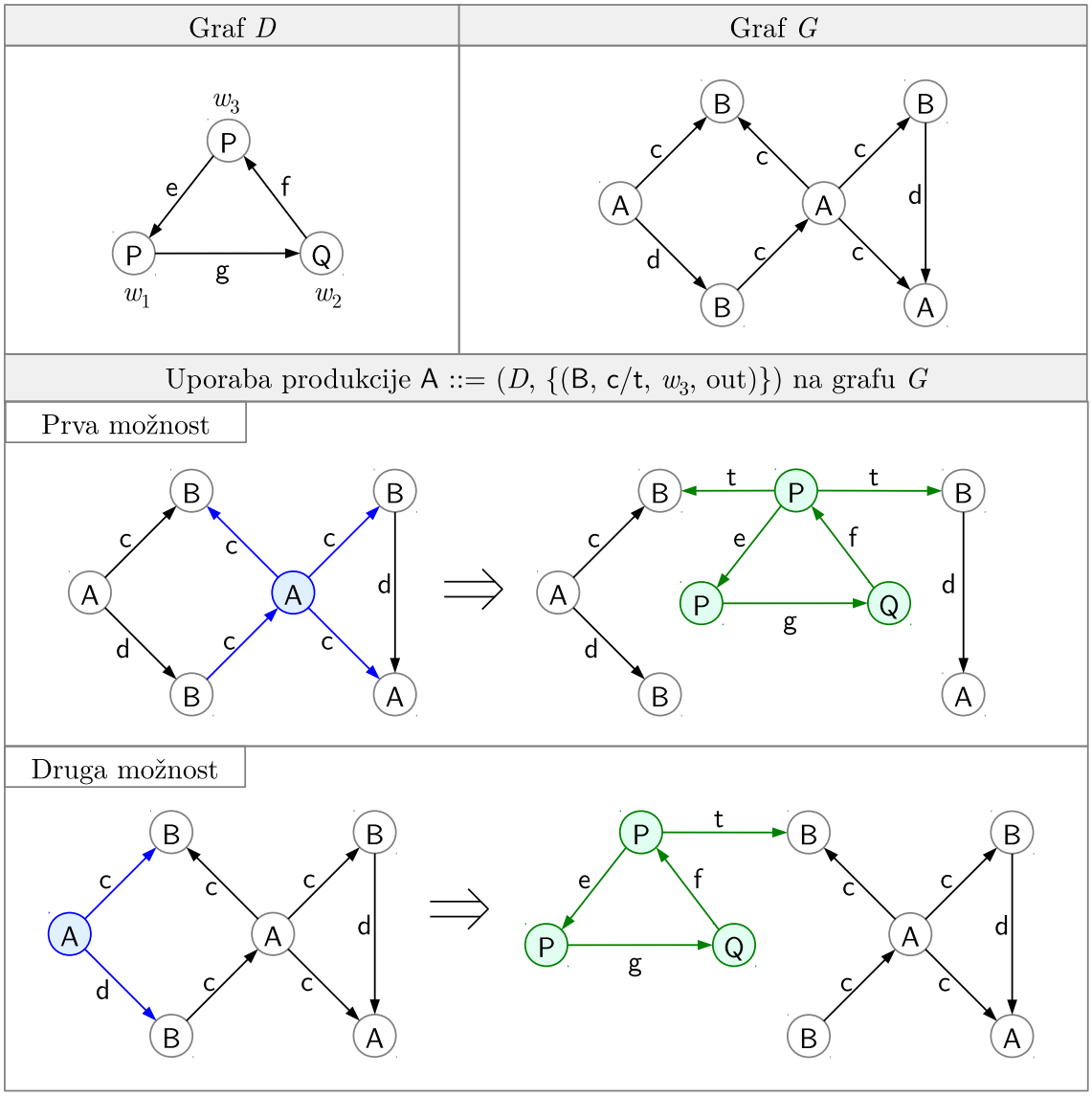
2. Iz grafa H odstranimo izbrano vozlišče skupaj z vsemi povezavami, ki so nanj pripete.

3. V graf H disjunktno dodamo graf D.

4. S pomočjo množice vložitvenih pravil $\mathcal{C}$ povežemo vozlišča dodanega grafa D z vozlišči, ki so bila v začetnem grafu H povezana z odstranjenim vozliščem.

Vložitvena pravila je možno definirati na mnogo različnih načinov. V *Priročniku* so podrobno predstavljene gramatike tipa *edNCE* (podvrsta vozliščnih gramatik), pri katerih je vsako posamezno vložitveno pravilo definirano kot četverica $(\mu, p/q, x, d)$, kjer je μ oznaka vozlišča, p in q sta oznaki povezav, x je konkretno vozlišče dodanega grafa D ($x \in \mathcal{V}[D]$), d pa je element množice $\{\text{in}, \text{out}\}$. Naj bo m vozlišče v gostiteljskem grafu, ki ga zamenjujemo z grafom D. Če je $d = \text{in}$, potem vložitveno pravilo izpolnimo tako, da vzpostavimo po eno povezavo z oznako q od vsakega vozlišča m' z oznako μ, ki je bilo z vozliščem m povezano s povezavo z oznako p v smeri $m' \to m$, do vozlišča x v dodanem grafu D. V primeru $d = \text{out}$ pa vzpostavimo po eno povezavo z oznako q od vozlišča x do vsakega vozlišča m' z oznako μ, ki je bilo z vozliščem m povezano s povezavo z oznako p v smeri $m \to m'$. Vložitveni postopek torej poveže vozlišče x v dodanem grafu z vsemi sosedi odstranjenega vozlišča m, ki izpolnjujejo pogoje glede njihovih oznak ter smeri in oznak njihovih povezav z

vozliščem m. Vse dodane povezave dobijo enake oznake in smeri. Primer uporabe produkcije je prikazan na sliki 2.6.



Slika 2.6: *Primer uporabe produkcije vozliščne gramatike.*

Ni zelo verjetno, da za gramatike edNCE obstaja učinkovit algoritem za reševanje problema pripadnosti, saj je ta problem *NP*-težak že za razred gramatik LIN·A-edNCE, tj. podrazred gramatik edNCE, pri katerih ima graf D v vsaki produkciji $X ::= (D, \mathcal{C})$ največ eno nekončno označeno vozlišče, v vsakem vložitvenem pravilu $(\mu, p/q, x, d)$ pa sta oznaka μ in oznaka vozlišča x končni.

Zanimiv podrazred gramatik edNCE (oziroma grafnih gramatik nasploh) so gramatike, ki generirajo linearne grafe z neoznačenimi vozlišči in označenimi povezavami, torej verige oblike $\bigcirc \xrightarrow{a_1} \bigcirc \xrightarrow{a_2} \bigcirc \xrightarrow{a_3} \ldots \xrightarrow{a_n} \bigcirc$. Graf takšne oblike lahko tolmačimo kot niz $a_1a_2a_3 \ldots a_n$. Grafne gramatike, katerih jeziki vsebujejo zgolj grafe, ki predstavljajo nize, bomo v splošnem označevali s pridevnikom *nizotvorne* (angl. *string-generating graph grammars*). Izkaže se, da lahko nizotvorne gramatike edNCE

generirajo vse množice verig, ki predstavljajo kontekstno neodvisne besedilne jezike, poleg tega pa obstaja gramatika, ki tvori verige oblike $a^n b^n c^n$ pri $n \geq 1$. Ker jezik, sestavljen iz takšnih nizov, ni kontekstno neodvisen, lahko rečemo, da imajo nizotvorne gramatike edNCE večjo izrazno moč od kontekstno neodvisnih besedilnih gramatik.

2.3.4 (Hiper)povezavne gramatike

Pri vozliščnih gramatikah moramo tako ali drugače rešiti problem vložitve desne strani produkcije v gostiteljski graf. Problemu vložitve pa se lahko povsem izognemo, če namesto vozlišč zamenjujemo *povezave*. Žal pa imajo povezavne gramatike (gramatike z zamenjavami povezav), ki jih bomo na kratko predstavili v podrazdelku 2.3.4.1, manjšo izrazno moč od vozliščnih. Zato se namesto povezavnih gramatik pogosto uporabljajo *hiperpovezavne* gramatike (gramatike z zamenjavami *hiperpovezav*). Hiperpovezave so posplošene povezave, ki lahko med seboj povezujejo poljubno (toda fiksno) število vozlišč, ne zgolj dve, kot to velja za običajne povezave. Hiperpovezavne gramatike delujejo nad *hipergrafi*, posplošenimi grafi, sestavljenimi iz vozlišč in hiperpovezav. Pojem hipergrafa bomo predstavili v podrazdelku 2.3.4.2, hiperpovezavnim gramatikam pa bomo namenili podrazdelek 2.3.4.3.

Pri (hiper)povezavnih gramatikah običajno predpostavljamo, da so vse (hiper)povezave usmerjene in označene, vozlišča pa neoznačena. To predpostavko bomo zaradi enostavnosti uporabljali tudi mi. Vendar pa je tovrstne gramatike možno povsem naravno posplošiti na primer neusmerjenih povezav in/ali označenih vozlišč.

2.3.4.1 Povezavne gramatike

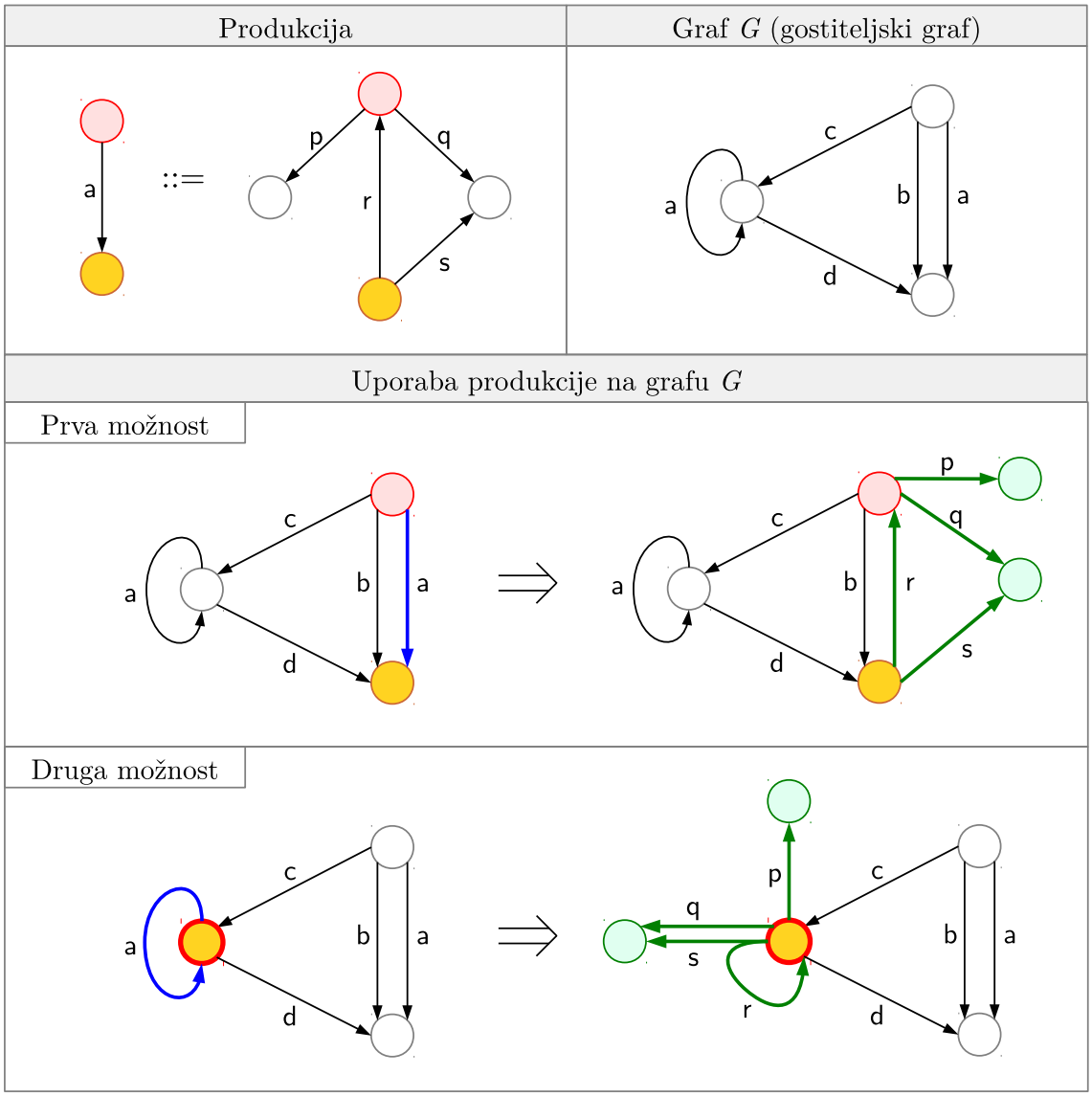
Povezavne gramatike vsebujejo produkcije oblike $Y ::= D$, kjer je Y nekončna oznaka usmerjene povezave, D pa je usmerjen graf z označenimi povezavami in neoznačenimi vozlišči, med katerimi natanko dve igrata posebno vlogo. To sta t.i. *zunanji vozlišči* (angl. *external vertices/nodes*). Zunanji vozlišči med seboj razlikujemo — eno od njiju je »prvo«, drugo pa »drugo« zunanje vozlišče. Produkcijo $Y ::= D$ na usmerjenem gostiteljskem grafu H z označenimi povezavami in neoznačenimi vozlišči uporabimo na sledeči način:

1. V grafu H poiščemo povezavo z oznako Y. Če takšne povezave ni, produkcije ne moremo uporabiti. Če je takšnih povezav več, lahko izberemo katerokoli od njih.

2. Iz grafa H odstranimo izbrano povezavo.

3. V graf H dodamo graf D, pri čemer prvo zunanje vozlišče grafa D poistovetimo z izvornim krajiščem odstranjene povezave, drugo zunanje vozlišče pa s ciljnim krajiščem odstranjene povezave.

Primer produkcije povezavne gramatike in njene uporabe na gostiteljskem grafu je prikazan na sliki 2.7. Prvo zunanje vozlišče je označeno s svetlordečo, drugo pa z rumeno barvo. Pri drugi možni uporabi produkcije sta obe zunanji vozlišči poistoveteni z istim vozliščem gostiteljskega grafa.



Slika 2.7: *Primer povezavne produkcije in njene uporabe na gostiteljskem grafu.*

V primeru neusmerjenih povezav in grafov ne razlikujemo med zunanjima vozliščema grafa D. Pri uporabi produkcije $Y ::= D$ na izbrani povezavi z oznako Y lahko zunanji vozlišči grafa D na oba možna načina poistovetimo s krajiščema izbrane povezave. Povezavo lahko zato nadomestimo z grafom D na dva možna načina.

Povezavne gramatike so enostavnejše od gramatik edNCE, žal pa so tudi šibkejše, saj generirajo strogo podmnožico jezikov, ki jih generirajo gramatike edNCE. Ta odnos velja tudi v primeru, če se omejimo na tvorbo grafov, ki predstavljajo nize. Medtem ko so nizotvorne gramatike edNCE močnejše od kontekstno neodvisnih besedilnih, so jim nizotvorne povezavne gramatike povsem enakovredne [31].

Kljub relativni šibkosti povezavnih gramatik je problem pripadnosti zanje še vedno *NP*-težak. Ker *NP*-težkosti očitno ne moremo ubežati, se je bolj smiselno ukvarjati s posplošitvijo, ki ima večjo izrazno moč. To so t.i. hiperpovezavne gramatike.

2.3.4.2 Hipergrafi

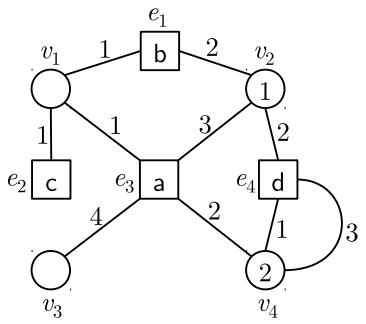
Povezave v običajnih grafih povezujejo po dve vozlišči. Lahko se sicer zgodi, da isto vozlišče služi kot izvorno in hkrati ciljno krajišče povezave, vendar pa še vedno lahko rečemo, da ima vsaka povezava dve priključni mesti oziroma »lovki« (angl. *tentacles*). *Hiperpovezava* (angl. *hyperedge*) pa je povezava s poljubnim (toda fiksnim) številom lovk, zato lahko med seboj povezuje tudi več kot dve vozlišči. Navadna povezava je potemtakem hiperpovezava z dvema lovkama. *Hipergraf* (angl. *hypergraph*) pa je po zgledu običajnih grafov struktura, ki vsebuje (običajna) vozlišča in hiperpovezave.

Podobno kot običajna povezava je lahko tudi hiperpovezava usmerjena ali neusmerjena. Pri neusmerjeni hiperpovezavi so vse lovke med seboj enakovredne, podobno kot sta pri običajni neusmerjeni povezavi medsebojno enakovredni obe njuni krajišči. Pri usmerjeni hiperpovezavi pa lovke oštevilčimo z zaporednimi številkami in jih tako razločujemo. (Pri običajni usmerjeni povezavi bi njeno izvorno krajišče oz. »lovko« oštevilčili z 1, ciljno pa z 2.)

Po običajni definiciji hipergrafa, kakršno najdemo tudi v *Priročniku*, so vozlišča neoznačena, hiperpovezave pa označene in usmerjene. Tako je hipergraf mogoče opredeliti kot šesterico $H = (\mathcal{V}, \mathcal{E}, att, \Sigma, label, Ext)$ (ali $\mathcal{V}[H]$, $\mathcal{E}[H]$, ...), kjer je $\mathcal{V}$ množica vozlišč, $\mathcal{E}$ množica hiperpovezav, $att\colon \mathcal{E} \to \mathcal{V}^*$ funkcija, ki vsaki hiperpovezavi priredi zaporedje vozlišč, ki jih hiperpovezava povezuje, Σ množica oznak povezav, $label\colon \mathcal{E} \to \Sigma$ funkcija, ki vsaki hiperpovezavi dodeli oznako, *Ext* pa je zaporedje zunanjih vozlišč hipergrafa. Funkcija *att* je posplošitev funkcij *source* in *target* pri običajnih grafih. Število lovk hiperpovezave (vrednost $|att(e)|$ pri $e \in \mathcal{E}$) imenujemo *tip* hiperpovezave (oznaka: $type(e)$). Zaradi enostavnosti se običajno predpostavi, da imajo vse hiperpovezave z isto oznako isti tip, torej da oznaka hiperpovezave enolično določa število njenih lovk. Zato lahko pišemo tudi $type(Y)$, kjer je $Y \in \Sigma$ neka oznaka hiperpovezave.

Slika 2.8 prikazuje primer hipergrafa. Vozlišča so prikazana s krogi, hiperpovezave s kvadrati, lovke hiperpovezav pa z oštevilčenimi daljicami ali krivuljami. Zunanji vozlišči sta prikazani kot vozlišči z napisoma 1 in 2. V tem hipergrafu velja $\mathcal{V} = \{v_1, v_2, v_3, v_4\}$, $\mathcal{E} = \{e_1, e_2, e_3, e_4\}$, $att(e_1) = (v_1, v_2)$, $att(e_2) = (v_1)$,

$att(e_3) = (v_1, v_4, v_2, v_3)$, $att(e_4) = (v_4, v_2, v_4)$, $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\}$, $label = \{e_1 \mapsto \mathsf{b}, e_2 \mapsto \mathsf{c}, e_3 \mapsto \mathsf{a}, e_4 \mapsto \mathsf{d}\}$ in $Ext = (v_2, v_4)$. S slike je razvidno, da lahko hipergraf predstavimo kot bipartiten običajni graf, saj lahko vozlišča hipergrafa predstavimo kot vozlišča ene vrste, hiperpovezave kot vozlišča druge vrste, lovke hiperpovezav pa kot običajne povezave.



Slika 2.8: *Primer hipergrafa.*

2.3.4.3 Hiperpovezavne gramatike

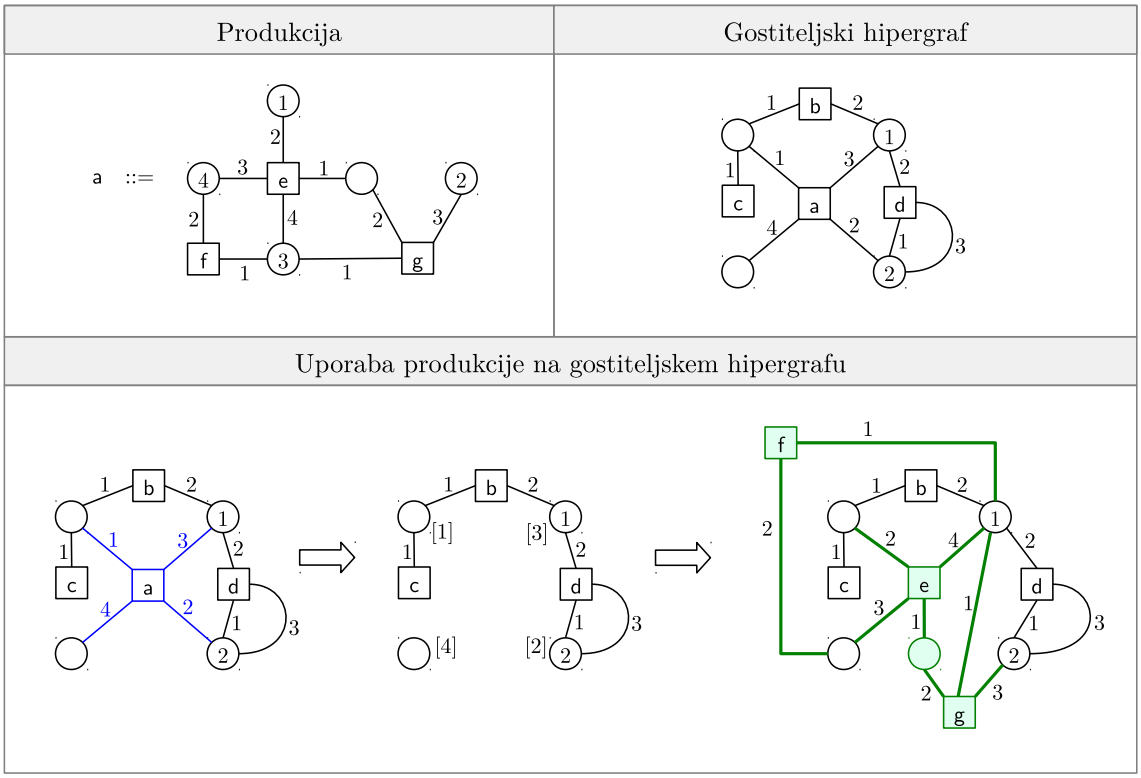
Hiperpovezavne gramatike vsebujejo produkcije oblike $Y ::= D$, kjer je Y oznaka hiperpovezave, D pa hipergraf. Produkcija je veljavna samo v primeru, če je tip oznake Y enak številu zunanjih vozlišč hipergrafa D. Produkcijo $Y ::= D$ na gostiteljskem hipergrafu H uporabimo na sledeči način:

1. V hipergrafu H poiščemo hiperpovezavo z oznako Y. Če takšne hiperpovezave ni, produkcije ne moremo uporabiti. Če je takšnih hiperpovezav več, lahko izberemo katerokoli od njih.

2. Iz hipergrafa H odstranimo izbrano hiperpovezavo.

3. V hipergraf H dodamo hipergraf D, pri čemer i-to zunanje vozlišče hipergrafa D poistovetimo z vozliščem, vezanim na i-to lovko odstranjene hiperpovezave (za vsak $i \in 1 \,..\, type(Y)$).

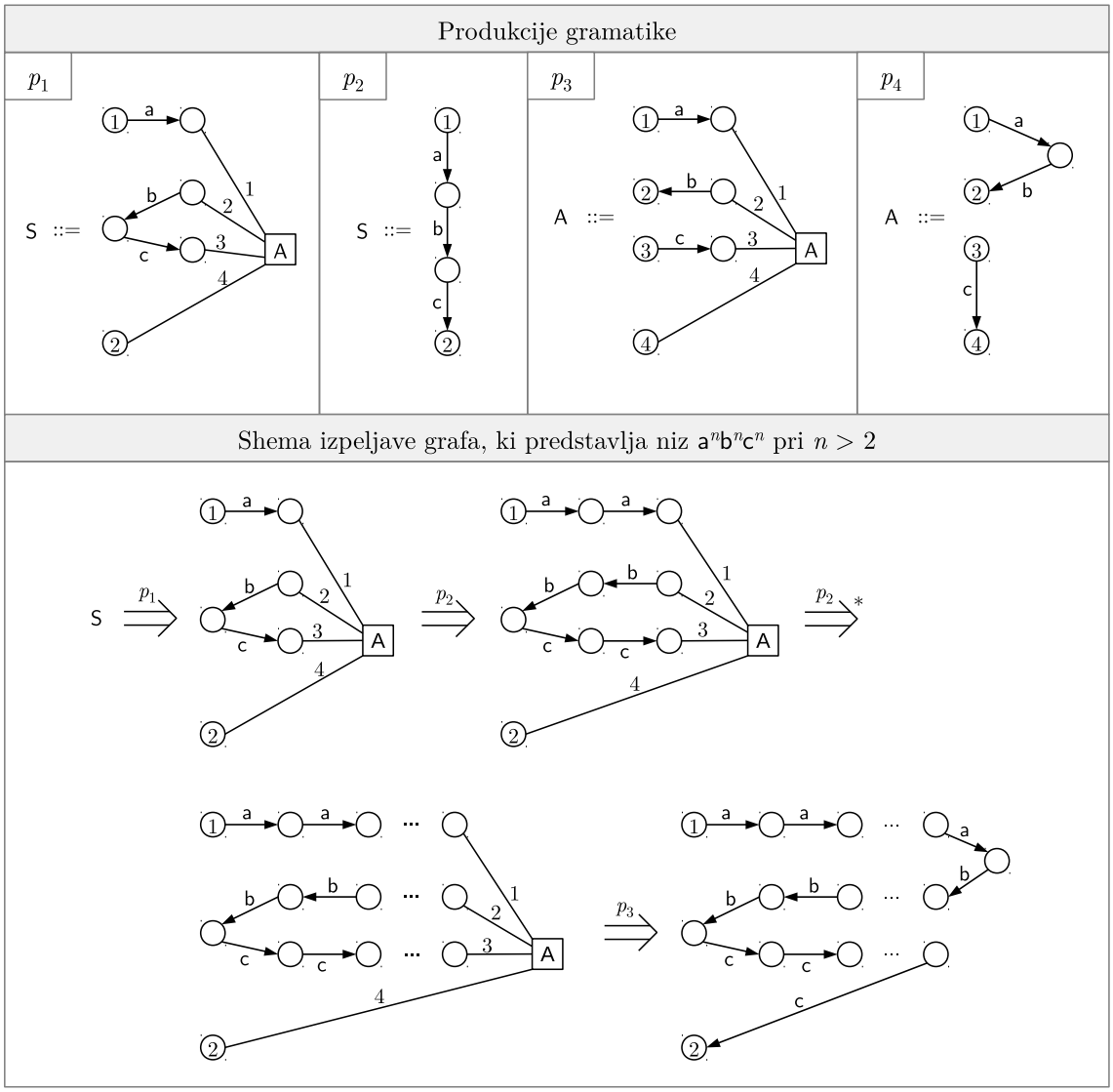
Primer uporabe produkcije je prikazan na sliki 2.9. Za lažje razumevanje si oglejmo, kako se v gostiteljski hipergraf vpne hiperpovezava z oznako e. Na prvo lovko hiperpovezave se veže novo vozlišče, ki ga uporaba produkcije doda v hipergraf. Na drugo lovko hiperpovezave e se veže vozlišče, ki se je poistovetilo s prvim zunanjim vozliščem produkcije. To je vozlišče, ki je bilo vezano na prvo lovko odstranjene hiperpovezave a, torej zgornje levo vozlišče hipergrafa. Na tretjo lovko hiperpovezave e se veže vozlišče, ki se je poistovetilo s četrtim zunanjim vozliščem produkcije, torej z vozliščem, ki je bilo vezano na četrto lovko odstranjene hiperpovezave a. To je spodnje levo vozlišče hipergrafa. Na četrto lovko hiperpovezave e pa se veže vozlišče, ki se je poistovetilo s tretjim zunanjim vozliščem produkcije, tj. zgornje desno vozlišče hipergrafa.



Slika 2.9: *Primer hiperpovezavne produkcije in njene uporabe na gostiteljskem hipergrafu.*

Zanimiva podrazreda hiperpovezavnih gramatik sta razreda grafotvornih in nizotvornih hiperpovezavnih gramatik. Grafotvorne gramatike generirajo zgolj običajne grafe namesto splošnih hipergrafov, nizotvorne pa grafe, ki predstavljajo nize. V nasprotju z nizotvornimi povezavnimi gramatikami so njihove hiperpovezavne inačice zmožne generirati tudi jezike, ki niso kontekstno neodvisni, kot je npr. jezik $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$. Gramatika, ki generira ta jezik, je sestavljena iz končnih oznak a, b in c, iz nekončnih oznak S (oznaka začetne hiperpovezave) in A z lastnostma $\mathit{type}(\mathsf{S}) = 0$ in $\mathit{type}(\mathsf{A}) = 4$ ter iz produkcij, prikazanih na sliki 2.10. Na isti sliki je prikazana tudi shema izpeljave grafa, ki predstavlja niz $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$ pri $n > 2$.

Nizotvorne hiperpovezavne gramatike je možno razporediti v hierarhijo glede na njihovo izrazno moč. Naj bo *red* gramatike takšno celo število r, da za vse hiperpovezave e, ki nastopajo v produkcijah gramatike, velja $\mathit{type}(e) \leq r$. Naj $\mathcal{SL}_r$ označuje razred vseh jezikov, ki so sestavljeni iz grafov za predstavitev nizov in ki jih generirajo nizotvorne hiperpovezavne gramatike reda r. Razred $\mathcal{SL}_2$ torej vsebuje vse jezike, ki jih generirajo nizotvorne povezavne gramatike, in je potemtakem enakovreden razredu kontekstno neodvisnih besedilnih jezikov. Za razrede, ki jih generirajo gramatike višjih redov, pa velja $\mathcal{SL}_{2k} = \mathcal{SL}_{2k+1}$ in $\mathcal{SL}_{2k} \subset \mathcal{SL}_{2(k+1)}$ pri vseh $k \in \mathbb{N}$ in $k \geq 1$. Jezik $\{a_1^n a_2^n \ldots a_{2k}^n \mid n \geq 1\}$ je namreč možno generirati z nizotvorno hiperpovezavno gramatiko reda $2k$, ne pa z istovrstno gramatiko reda $2k - 1$.



Slika 2.10: *Produkcije hiperpovezavne gramatike za jezik* $\{\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n \mid n \geq 1\}$ *in shema izpeljave grafa, ki predstavlja niz* $\mathsf{a}^n\mathsf{b}^n\mathsf{c}^n$ *pri* $n > 2$.

Problem pripadnosti je za splošne hiperpovezavne gramatike seveda *NP*-težak, saj to velja že za splošne povezavne gramatike. Polinomski algoritem za reševanje tega problema obstaja samo za omejen podrazred hiperpovezavnih gramatik [60].

2.3.5 Kontekstno odvisne grafne gramatike

Kontekstno odvisnim grafnim gramatikam bomo namenili lastno podpoglavje (2.4), zato bomo na tem mestu podali samo njihove osnovne značilnosti. Kontekstno odvisne grafne gramatike vsebujejo produkcije za zamenjavo podgrafov s poljubnimi grafi. Podobno kot zamenjava vozlišča tudi zamenjava podgrafa ni enolično določena. V nasprotju z vozliščnimi gramatikami pa pri produkcijah kontekstno odvisnih grafnih gramatik ne uporabljamo vložitvenih pravil, ampak t.i. *kontekst*. Kontekst je sestavljen iz elementov, ki pri zamenjavi ostanejo nedotaknjeni, vendar pa določajo način, kako se elementi desne strani produkcije pripnejo na obstoječe elemente gostiteljskega grafa. Uporabo produkcije s kontekstom bomo podrobneje predstavili v podpoglavju 2.4, na tem mestu pa se spomnimo na sliko 1.1 (stran 4).

Pri kontekstno odvisnih grafnih gramatikah, ki ne izpolnjujejo nobenih dodatnih pogojev, problem pripadnosti v splošnem ni odločljiv. Vendar pa sta Rekers in Schürr [80] pokazala, da je odločljivost problema pripadnosti možno zagotoviti z razmeroma blagim dodatnim pogojem. Če grafna gramatika izpolnjuje ta pogoj, potem se bo sintaksni analizator, ki sta ga iznašla Rekers in Schürr [79, 80], s pravilnim odgovorom ustavil pri vseh vhodnih grafih. Problem pripadnosti je tudi pri kontekstno odvisnih gramatikah *NP*-težak, zato Rekers-Schürrov sintaksni analizator v najslabšem primeru potrebuje eksponentno mnogo časa in prostora, da zgradi izpeljavo vhodnega grafa oziroma sporoči, da vhodni graf ne pripada dani gramatiki.

Problem pripadnosti je *NP*-težak tako pri (splošnih) kontekstno neodvisnih kot pri kontekstno odvisnih grafnih gramatikah. Ker se *NP*-težkosti skoraj v nobenem primeru ne moremo izogniti, se je smiselno ukvarjati s čim močnejšim grafnogramatičnim formalizmom, pri katerem je problem pripadnosti oz. sintaksne analize še odločljiv. Zato smo se pri našem raziskovalnem delu osredotočili na sintaksno analizo kontekstno odvisnih grafnih gramatik.

2.3.6 Sorodni formalizmi

Poleg predstavljenih osnovnih oblik grafnih gramatik najdemo v literaturi še številne druge oblike in formalizme, sorodne grafnim gramatikam. Navedimo jih zgolj nekaj:

Relacijske gramatike (angl. *relation grammars*) [20] so definirane na podoben način kot kontekstno neodvisne besedilne gramatike, le da so produkcije lahko opremljene z dodatnimi predikati, ki opisujejo odnose med posameznimi simboli. Ferrucci in sod. [33] so pokazali, da so relacijske gramatike enakovredne vozliščnim gramatikam edNCE, ki smo si jih ogledali v razdelku 2.3.3.

Adaptivne zvezdne gramatike (angl. *adaptive star grammars*) [25, 67] so razširitev neke oblike hiperpovezavnih gramatik. V produkcijah adaptivnih zvezdnih gramatik je mogoče poljubno število vozlišč predstaviti z enim samim *večkratnim vozliščem* (angl. *multiple node*). V tovrstnih gramatikah je zato mogoče

podajati splošnejša zamenjevalna pravila kot v običajnih vozliščnih in hiperpovezavnih gramatikah. Razred nizotvornih adaptivnih zvezdnih gramatik je enakovreden razredu besedilnih gramatik tipa 0. Zaradi tega ni presenetljivo, da problem pripadnosti za splošne adaptivne zvezdne gramatike ni odločljiv.

Rezervirane grafne gramatike (angl. *reserved graph grammars*) [103] so podobne kontekstno odvisnim grafnim gramatikam, le da je kontekst pri posameznih produkcijah podan s pomočjo »podvozlišč«, tj. posebnih elementov znotraj vozlišč, ki predstavljajo priključke za povezave. Ta formalizem omogoča nekoliko večjo fleksibilnost pri podajanju zamenjevalnih pravil, a za ceno kompleksnejše definicije vozlišča grafa.

Prostorske grafne gramatike (angl. *spatial graph grammars*) [53] so definirane kot posplošitev rezerviranih grafnih gramatik. V prostorskih grafnih gramatikah lahko med posameznimi vozlišči v produkcijah nastopajo tudi topološke in smerne relacije, kot so npr. »se dotika«, »je levo od« ipd., s čimer je mogoče na enostavnejši način podati sintakso nekaterih vizualnih programskih jezikov.

Trojne grafne gramatike (angl. *triple graph grammars*) [54, 86] so kontekstno odvisne grafne gramatike, katerih produkcije lahko grafne elemente zgolj dodajajo; odvzemanje ni možno. Tovrstne gramatike so namenjene podajanju modelskih transformacij. Vsaka produkcija je razdeljena na tri dele. Prvi del vsebuje elemente modela v izvorni domeni (metamodelu), drugi pripadajoče elemente modela v ciljni domeni (metamodelu), tretji del produkcije pa vsebuje grafne elemente, ki povezujejo pripadajoče elemente obeh modelov med seboj. S pomočjo trojnih grafnih gramatik lahko marsikateri problem s področja modeliranja prevedemo v enakovreden problem v grafnogramatičnem okolju.

2.4 Kontekstno odvisne grafne gramatike

V tem podpoglavju bomo predstavili kontekstno odvisne grafne gramatike v skladu z definicijo Rekersa in Schürra [80]. V razdelku 2.4.1 si bomo ogledali zgradbo tovrstnih gramatik in njihovih produkcij. V razdelku 2.4.2 bomo definirali uporabo produkcije na danem grafu. V razdelku 2.4.3 bomo pojma izpeljave in jezika, ki smo ju predstavili že v podpoglavju 2.3, formalno definirali. V razdelku 2.4.4 bomo govorili o t.i. sintaksni odločljivosti kontekstno odvisnih grafnih gramatik.

2.4.1 Produkcije in gramatike

Kontekstno odvisne grafne gramatike, kot sta jih definirala Rekers in Schürr [80], lahko obravnavamo kot grafno razširitev kontekstno odvisnih besedilnih gramatik. Definirali jih bomo nekoliko drugače kot Rekers in Schürr. Rekers in Schürr sta uvedla pojem *plasti*, ki naj bi nadomestil tradicionalno dvoplastno delitev oznak gramatike na končne in nekončne, in sta zato svoje gramatike imenovala tudi *plastne grafne gramatike* (angl. *layered graph grammars*). V doktorski disertaciji se bomo raje skušali držati analogije z besedilnimi gramatikami, zato bomo uporabljali

pojma končnih in nekončnih oznak v običajnem pomenu. O »plasteh« bomo govorili zgolj kot o konceptu, s pomočjo katerega je definirana sintaksna odločljivost gramatik (razdelek 2.4.4), sicer pa plastem ne bomo pripisovali nikakršnega pomena. Zaradi tega bomo namesto poimenovanja »plastne grafne gramatike« uporabljali ime »kontekstno odvisne grafne gramatike«.

Definicija 2.15 (kontekstno odvisna grafna produkcija)**.** *Kontekstno odvisna grafna produkcija* p je pravilo oblike $Lhs[p] ::= Rhs[p]$, pri čemer velja:

- $Lhs[p]$ (*leva stran* produkcije, angl. *left-hand side*) je poljuben graf.
- $Rhs[p]$ (*desna stran*, angl. *right-hand side*) je poljuben graf, ki se lahko delno (ali celo povsem) prekriva z grafom $Lhs[p]$.

Definicija 2.16 (kontekstni graf)**.** *Kontekstni graf* (angl. *context graph*) produkcije p je graf $Common[p] = Lhs[p] \cap Rhs[p]$.

Graf *Common* je torej sestavljen iz elementov, ki so skupni grafoma *Lhs* in *Rhs*. Velja opozoriti, da govorimo o elementih, ki so prisotni v obeh grafih, ne o različnih elementih z istimi oznakami.

Definicija 2.17 (kontekstno odvisna grafna gramatika)**.** *Kontekstno odvisna grafna gramatika* je trojica $GG = (\mathcal{N}, \mathcal{T}, \mathcal{P})$, pri čemer velja:

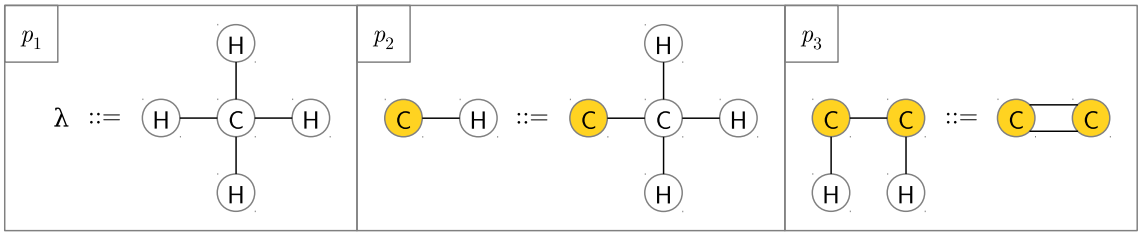
- $\mathcal{N}$ je množica *nekončnih oznak*;
- $\mathcal{T}$ je množica *končnih oznak*;
- $\mathcal{N} \cap \mathcal{T} = \emptyset$;
- $\mathcal{P}$ je množica kontekstno odvisnih grafnih *produkcij*, v katerih so grafni elementi označeni z oznakami iz množice $\mathcal{N} \cup \mathcal{T}$.

Vsaj ena produkcija gramatike mora biti oblike $\lambda ::= G$, kjer λ označuje prazen, G pa nek neprazen graf. Produkcije takšne oblike tvorijo množico *začetnih produkcij* gramatike. Množico *začetnih grafov* gramatike sestavljajo desne strani začetnih produkcij.

Začetne produkcije igrajo posebno vlogo, saj jih je mogoče uporabiti samo v prvem koraku izpeljave. Zaradi tega v nekem smislu niso »prave« produkcije. Kljub temu jih bomo zaradi enostavnosti obravnavali na enak način kot vse ostale produkcije.

V nadaljevanju razdelka bomo kot tekoči primer uporabljali kontekstno odvisno grafno gramatiko $GG_{\mathrm{AHCt}} = (\mathcal{N}, \mathcal{T}, \mathcal{P})$, kjer je $\mathcal{N} = \emptyset$, $\mathcal{T} = \{\mathsf{C}, \mathsf{H}, \#\}$, množica produkcij $\mathcal{P}$ pa je prikazana v na sliki 2.11. (Spomnimo se, da neoznačene grafne elemente obravnavamo kot elemente z oznako #.) Elementi, ki so na sliki prikazani z rumeno barvo, pripadajo kontekstnim grafom posameznih produkcij. Tega dogovora se bomo držali tudi v prihodnje.

Definicija 2.18 (končno/nekončno označen grafni element)**.** Grafni element (vozlišče ali povezava) x je *končno* (oziroma *nekončno*) *označen* natanko v primeru, če velja $label(x) \in \mathcal{T}$ (oziroma $label(x) \in \mathcal{N}$).



Slika 2.11: *Produkcije gramatike* GG_{AHCt}.

Definicija 2.19 (končno/nekončno označen graf)**.** Graf je *končno označen* natanko tedaj, ko so *vsi* njegovi elementi končno označeni. Graf je *nekončno označen* natanko tedaj, ko je *vsaj eden* od njegovih elementov nekončno označen.

Definicija 2.20 (izključna leva/desna stran)**.** *Izključna leva stran* (angl. *exclusive left-hand side*) produkcije p je množica grafnih elementov $Xlhs[p] = Lhs[p] \setminus Common[p]$. *Izključna desna stran* (angl. *exclusive right-hand side*) produkcije p je množica grafnih elementov $Xrhs[p] = Rhs[p] \setminus Common[p]$.

Definicija 2.21 (unijski graf)**.** *Unijski graf* (angl. *union graph*) produkcije p je graf $Union[p] = Lhs[p] \cup Rhs[p]$.

Na sliki 2.12 so za vsako produkcijo gramatike GG_{AHCt} prikazane množice *Lhs*, *Rhs*, *Common*, *Xlhs*, *Xrhs* in *Union*. Graf $Common[p_1]$ je prazen. Množice $Xlhs[p_2]$, $Xrhs[p_2]$, $Xlhs[p_3]$ in $Xrhs[p_3]$ ne tvorijo veljavnih grafov, saj jim vozlišča, ki so na sliki 2.12 označena s svetlejšim črtkanim robom in svetlejšim napisom, ne pripadajo. Zato vsaka od navedenih množic vsebuje vsaj po eno visečo povezavo.

Pomembno je razumeti, da so pri poljubni produkciji p grafi $Lhs[p]$, $Rhs[p]$ in $Common[p]$ podgrafi grafa $Union[p]$, množici $Xlhs[p]$ in $Xrhs[p]$ pa sta podmnožici elementov grafov $Lhs[p]$ oz. $Rhs[p]$. Grafi *Lhs*, *Rhs*, *Common* ter množici *Xlhs* in *Xrhs* torej niso kopije posameznih delov grafa *Union*, temveč samo *pogledi* nanje. Da bi ta koncept bolje ponazorili, smo vsa vozlišča na sliki 2.12 (razen fiktivnih, ki ponazarjajo viseče povezave) opremili z identifikatorji. Denimo, identifikator v_1, ki nastopa v grafih $Lhs[p_2]$, $Rhs[p_2]$, $Common[p_2]$ in $Union[p_2]$, označuje eno in isto vozlišče.

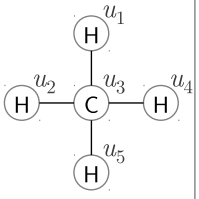
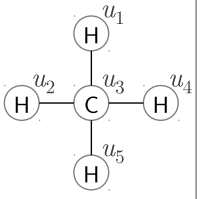
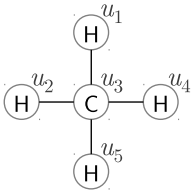
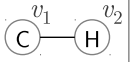
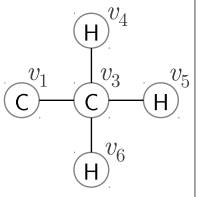
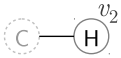
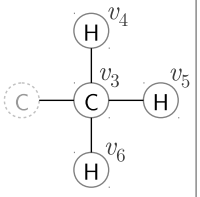
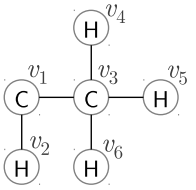
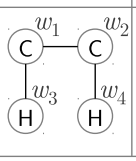
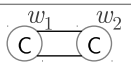
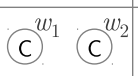
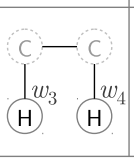
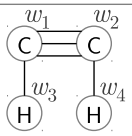
Sledeči definiciji nam bosta prišli prav predvsem v poglavju 4, ko se bomo ukvarjali z indukcijo grafnih gramatik:

Definicija 2.22 (velikost produkcije)**.** *Velikost* produkcije je število elementov njenega unijskega grafa:

$$size(p) = |Elements[Union[p]]|.$$

Definicija 2.23 (velikost gramatike)**.** Velikost gramatike je vsota velikosti vseh njenih produkcij:

$$size(GG) = \sum_{p \in \mathcal{P}[GG]} size(p)$$

p	$Lhs[p]$	$Rhs[p]$	$Common[p]$	$Xlhs[p]$	$Xrhs[p]$	$Union[p]$
p_1	λ		λ	λ		
p_2						
p_3						

Slika 2.12: *Množice Lhs, Rhs, Common, Xlhs, Xrhs in Union za posamezne produkcije gramatike* GG_{AHCt}.

2.4.2 Uporaba produkcije

Produkcija je pravilo za zamenjavo podgrafa z drugim podgrafom v nekem gostiteljskem grafu. V definiciji 2.28 bomo opisali, kako to pravilo dejansko uporabimo. Še prej pa moramo definirati posebno obliko homomorfizma, ki je tesno povezana z uporabami produkcij in, kot bomo videli v poglavju 3, s sintaksno analizo.

Definicija 2.24 (p-homomorfizem)**.** Naj bo p kontekstno odvisna produkcija, P nek podgraf grafa $Union[p]$, G graf, $h\colon P \to G$ pa homomorfizem. Potem je homomorfizem h *p-homomorfizem* natanko tedaj, ko je njegova omejitev na elemente množice $P \setminus Common[p]$ monomorfizem.

Z drugimi besedami: p-homomorfizem med elementi produkcije p in grafom G je homomorfizem, pri katerem morajo biti vsi elementi produkcije, ki *ne* pripadajo grafu $Common[p]$, na pripadajoče elemente grafa G preslikani injektivno, za elemente grafa $Common[p]$ pa injektivnost ni zahtevana; dopustno je, da se več različnih elementov grafa $Common[p]$ preslika v isti element grafa G, če so izpolnjeni pogoji za homomorfizem iz definicije 2.9. V povezavi s p-homomorfizmom bomo definirali še koncepte, podobne pojmu pojavitve, ki smo ga uvedli v definiciji 2.11:

Definicija 2.25 (l-slika produkcije)**.** Naj bo p kontekstno odvisna produkcija, G graf, $h\colon Lhs[p] \to G$ pa p-homomorfizem. Potem je graf $h(Lhs[p])$ *l-slika* produkcije p v grafu G.

Definicija 2.26 (c-slika produkcije)**.** Naj bo p kontekstno odvisna produkcija, G graf, $h\colon Common[p] \to G$ pa homomorfizem. Potem je graf $h(Common[p])$ *c-slika* produkcije p v grafu G.

Definicija 2.27 (r-slika produkcije)**.** Naj bo p kontekstno odvisna produkcija, G graf, $h\colon Rhs[p] \to G$ pa p-homomorfizem. Potem je graf $h(Rhs[p])$ *r-slika* produkcije p v grafu G.

Sedaj so izpolnjeni vsi pogoji za definicijo uporabo produkcije:

Definicija 2.28 (uporaba produkcije)**.** Produkcijo p na gostiteljskem grafu G *uporabimo* tako, da izvršimo sledeče tri korake:

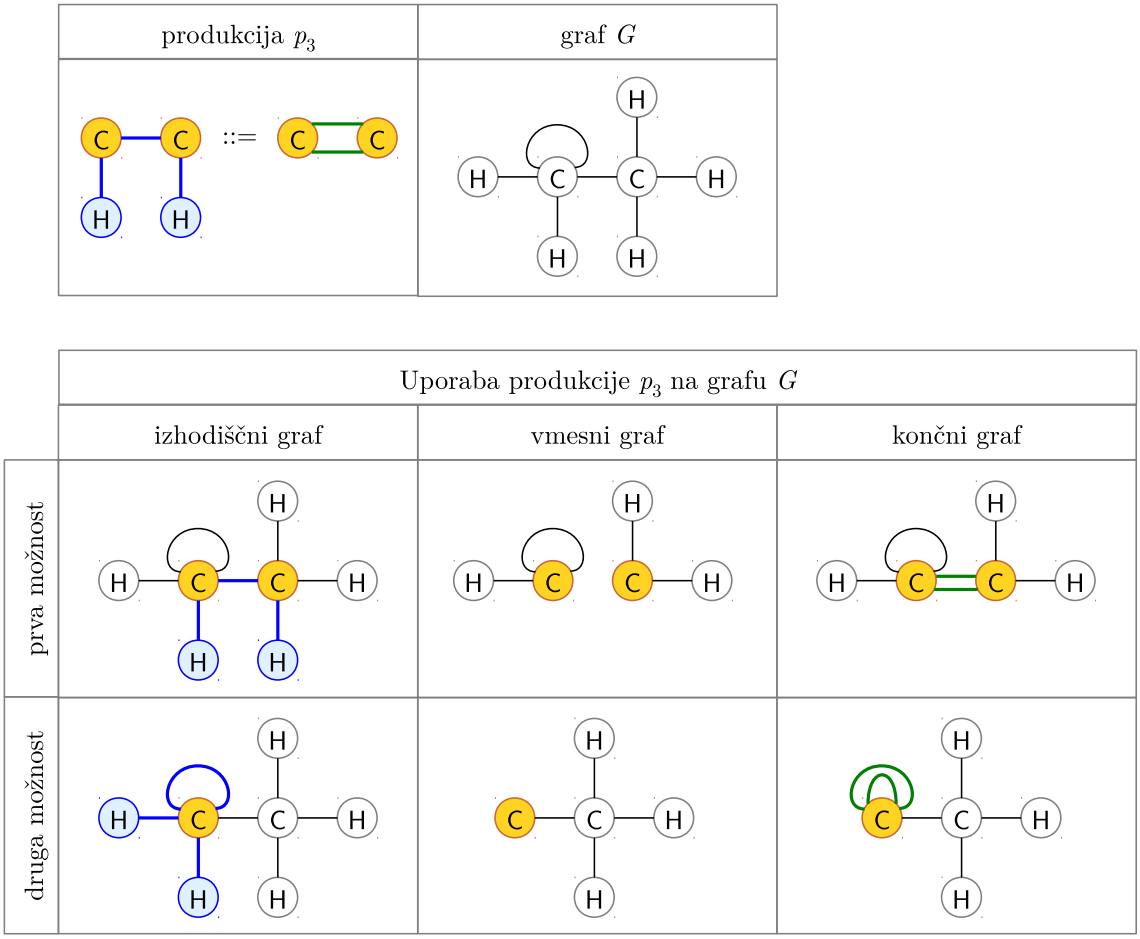
1. V grafu G poiščemo l-sliko produkcije p, torej nek graf $L \sqsubseteq G$ z lastnostjo $L = h_L(Lhs[p])$ za nek p-homomorfizem $h_L\colon Lhs[p] \to G$. Graf $h_L(Lhs[p])$ je sestavljen iz elementov množice $h_L(Xlhs[p])$ in elementov grafa $h_L(Common[p])$, ki je c-slika produkcije p v grafu G.

2. Iz grafa G odstranimo vse elemente množice $h_L(Xlhs[p])$ in dobimo graf G'. To lahko storimo le v primeru, če je izpolnjen *pogoj visečih povezav* (angl. *dangling edge condition*). Ta pogoj je izpolnjen, kadar za vsako vozlišče v množici $h_L(Xlhs[p])$ velja, da tudi vse nanj pripete povezave pripadajo množici $h_L(Xlhs[p])$. Če to ne drži, potem bi po odstranitvi nekega vozlišča v grafu G ostale viseče povezave, kar pa ni dovoljeno.

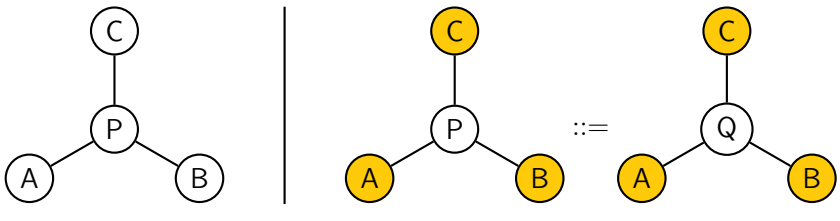
3. Graf H z lastnostjo $G \overset{p}{\Rightarrow} H$ dobimo tako, da v graf G' dodamo kopije elementov $Xrhs[p]$ in jih na elemente grafa $h_L(Common[p])$ vežemo na enak način, kot so elementi množice $Xrhs[p]$ vezani na elemente podgrafa $Common[p]$ znotraj grafa $Rhs[p]$. Če je izvor (oz. cilj) povezave $e \in Xrhs[p]$ v grafu $Rhs[p]$ vozlišče $u \in Xrhs[p]$, potem mora biti izvor (oz. cilj) njene kopije e' v grafu H kopija vozlišča u. Če pa je izvor (oz. cilj) povezave $e \in Xrhs[p]$ v grafu $Rhs[p]$ vozlišče $u \in Common[p]$, potem mora biti izvor (oz. cilj) njene kopije e' v grafu H vozlišče $h_L(u)$. Dodane kopije elementov $Xrhs[p]$ tvorijo skupaj z elementi grafa $h_L(Common[p])$ r-sliko produkcije p v grafu H.

Slika 2.13 prikazuje dve različni možnosti za uporabo produkcije p_3 gramatike GG_{AHCt} na nekem grafu. Obarvani elementi v prvem, drugem in tretjem stolpcu po vrsti tvorijo l-sliko, c-sliko in r-sliko produkcije v grafu. Pri prvi možni uporabi produkcije so vsi elementi njene leve strani na elemente njene l-slike preslikani injektivno; p-homomorfizem leve strani je v tem primeru torej monomorfizem. Zato je tudi r-slika v končnem grafu injektivna slika desne strani produkcije. Pri drugi možnosti pa se obe vozlišči grafa $Common[p_3]$ preslikata v isto vozlišče grafa G. To vozlišče zato v končnem grafu pridobi še eno zanko. Naj omenimo, da izhodiščni in končni graf ne predstavljata veljavne kemijske strukturne formule, prav tako pa nista vsebovana v jeziku gramatike GG_{AHCt}.

Produkcijo p lahko na grafu G uporabimo v primeru, če graf G vsebuje l-sliko produkcije p in če je izpolnjen pogoj visečih povezav. Zaradi tega pogoja je včasih težko ali celo nemogoče definirati produkcije za navidez preproste zamenjave. Denimo, da bi v grafu na levem delu slike 2.14 želeli zamenjati vozlišče z oznako P z vozliščem z oznako Q. Prva zamisel bi bila uporaba produkcije $\mathsf{P} ::= \mathsf{Q}$. Žal pa te produkcije na prikazanem grafu ne moremo uporabiti, saj bi po odstranitvi vozlišča P dobili tri viseče povezave. Zamišljeno zamenjavo lahko izvršimo le s produkcijo na desnem delu slike 2.14.



Slika 2.13: *Dve možni uporabi produkcije na grafu.*



Slika 2.14: *Graf (levo) in produkcija (desno).*

Če torej želimo izdelati produkcijo za zamenjavo podgrafa L s podgrafom R, moramo upoštevati vse možne soseščine podgrafa L v gostiteljskem grafu G. Za vsako tako soseščino izdelamo svojo produkcijo. Če število možnih soseščin ni omejeno, ustreznih produkcij ni mogoče definirati. Ni mogoče, denimo, definirati množice produkcij za zamenjavo vozlišča z oznako P z vozliščem z oznako Q v *poljubnem* grafu.

V nadaljevanju bomo občasno potrebovali tudi pojem *obratne uporabe* produkcije (uporabe produkcije v obratni smeri) na nekem grafu. Obratna uporaba produkcije $L ::= R$ je enakovredna uporabi produkcije $R ::= L$. Formalneje:

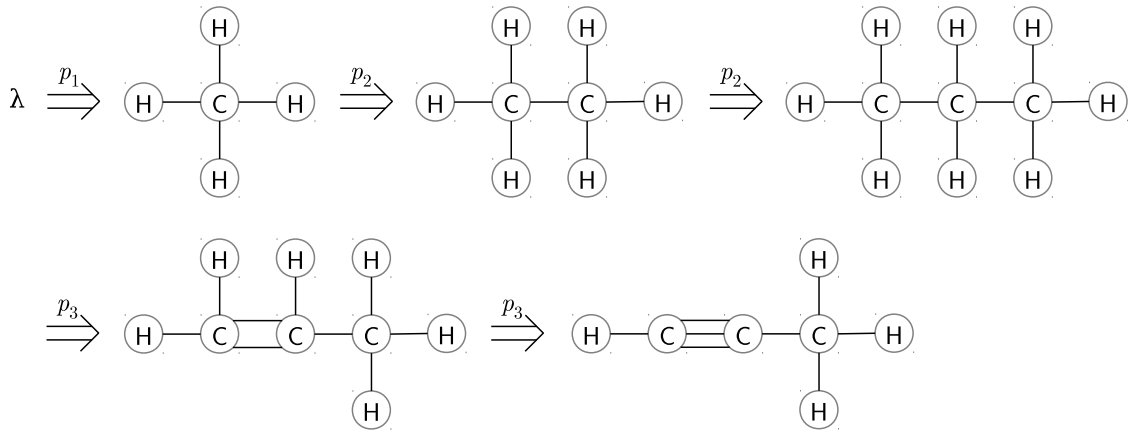
Definicija 2.29 (obratna uporaba produkcije)**.** Produkcijo p na danem grafu *obratno uporabimo* tako, da na njem uporabimo produkcijo p', za katero velja $Union[p'] = Union[p]$, $Common[p'] = Common[p]$, $Lhs[p'] = Rhs[p]$, $Xlhs[p'] = Xrhs[p]$, $Rhs[p'] = Lhs[p]$ in $Xrhs[p'] = Xlhs[p]$.

Produkcijo torej obratno uporabimo tako, da zamenjamo vlogi leve in desne strani ter množic *Xlhs* in *Xrhs*.

2.4.3 Izpeljava in jezik

Definicija 2.30 (izpeljava)**.** *Izpeljava* grafa G v podani grafni gramatiki je zaporedje uporab produkcij $p_{i[1]}, \ldots, p_{i[k]}$, ki prazen graf po korakih pretvori v graf G ($\lambda \overset{p_{i[1]}}{\Longrightarrow} \ldots \overset{p_{i[k]}}{\Longrightarrow} G$). Prvi korak izpeljave se vedno prične z eno od začetnih produkcij, v kasnejših korakih pa teh produkcij ni več dovoljeno uporabljati.

Na sliki 2.15 je prikazana izpeljava strukturne formule propina ($HC \equiv C - CH_3$) v gramatiki GG_{AHCt}.



Slika 2.15: *Izpeljava strukturne formule propina v gramatiki* GG_{AHCt}.

Definicija 2.31 (izpeljivost)**.** Graf G je *izpeljiv* v gramatiki GG natanko v primeru, če obstaja izpeljava grafa G v gramatiki GG.

Definicija 2.32 (generiranje)**.** Gramatika GG *generira* graf G natanko tedaj, ko je graf G izpeljiv v gramatiki GG.

Definicija 2.33 (jezik)**.** *Jezik* grafne gramatike GG (oznaka: $L(GG)$) je množica vseh *končno označenih* grafov, izpeljivih v gramatiki GG.

Definicija 2.34 (pokrivanje)**.** Grafna gramatika GG *pokriva* graf G natanko tedaj, ko velja $G \in L(GG)$.

Definicija 2.35 (sintaksni analizator)**.** *Sintaksni analizator* je algoritem, ki za podani graf G in grafno gramatiko GG ugotovi, ali velja $G \in L(GG)$, in — če to drži — poišče izpeljavo grafa G v gramatiki GG.

Jezik gramatike GG_{AHCt} je množica vseh grafov, ki predstavljajo legalne strukturne formule acikličnih ogljikovodikov. Ogljikovodik je kemijska spojina ogljika (C) in vodika (H), v kateri vsak ogljikov atom tvori po štiri vezi z drugimi atomi, vsak vodikov atom pa po eno vez. Ogljikovi atomi so lahko med seboj povezani z enojnimi, dvojnimi ali trojnimi vezmi. Vsaka dvojna (oz. trojna) vez šteje kot dve (oz. tri) enojne. V strukturni formuli so atomi predstavljeni z vozlišči, vezi pa s povezavami. Večkratne vezi so predstavljene z vzporednimi povezavami. Aciklični ogljikovodiki so ogljikovodiki, katerih strukturne formule so aciklični grafi.

Če želimo preveriti, ali jezik gramatike GG dejansko sovpada z množico veljavnih strukturnih formul acikličnih ogljikovodikov, moramo pokazati sledeči trditvi:

(1) Vsak graf, ki pripada jeziku gramatike, predstavlja veljavno strukturno formulo nekega ogljikovodika.

(2) Vsaka veljavna strukturna formula ogljikovodika pripada jeziku gramatike GG.

Preverimo najprej trditev (1). Produkcija p_1 ustvari graf, ki predstavlja veljavno strukturno formulo, saj ogljikov atom v središču spojine tvori štiri vezi, vsi vodikovi atomi pa po eno vez. Produkcija p_2 ohranja veljavnost strukturne formule: če to produkcijo uporabimo na veljavni strukturni formuli, bo nastala strukturna formula še vedno veljavna, saj ogljikov atom, na katerem uporabimo produkcijo p_2, izgubljeno vez z vodikovim atomom nadomesti z vezjo z novonastalim ogljikovim atomom. Novonastali ogljikov atom tvori štiri vezi, vsi novonastali vodikovi atomi pa po eno. Veljavnost strukturne formule ohranja tudi produkcija p_3. Oba ogljikova atoma, ki sta udeležena pri uporabi produkcije, izgubita po eno vez z vodikovima atomoma, ki ju uporaba produkcije odstrani, vendar pridobita novo medsebojno vez, kar pomeni, da oba ohranita svoje prvotno število vezi. Ker produkcija p_1 ustvari veljavno strukturno formulo, ostale produkcije pa to veljavnost ohranjajo, gramatika GG_{AHCt} ne more generirati neveljavnih strukturnih formul.

Če bi gramatika GG_{AHCt} vsebovala tudi nekončne oznake, bi se dokaza trditve (1) lahko lotili tako, da bi najprej pokazali, da za vse grafe, izpeljive v gramatiki GG_{AHCt}, velja določena invarianta — lastnost, ki jo začetna(-e) produkcija(-e) vzpostavi(jo), vse ostale produkcije pa ohranjajo. V drugem koraku pa bi morali pokazati, da se pri izpeljivih grafih, ki vsebujejo zgolj končno označene elemente (torej pri grafih iz jezika gramatike), ta invarianta preoblikuje v ciljno lastnost jezika — v našem primeru v lastnost, ki jo spoštujejo vse veljavne strukturne formule.

Trditev (2) pa pokažemo tako, da za nek graf G, ki predstavlja veljavno strukturno formulo, preverimo, ali ga je mogoče izpeljati v gramatiki GG_{AHCt}. To lahko storimo tako, da graf G poskušamo z *obratnimi* uporabami produkcij pretvoriti v

prazen graf (oziroma v enega od začetnih grafov). Predpostavimo torej, da graf G predstavlja veljavno strukturno formulo nekega acikličnega ogljikovodika. Najprej pretvorimo vse večkratne vezi v enojne z obratnimi uporabami produkcije p_3. Sedaj vsako vozlišče C tvori po štiri enojne vezi. Če je nastali graf enak desni strani produkcije p_1, smo končali, saj nas obratna uporaba produkcije p_1 pripelje do želenega praznega grafa. V nasprotnem primeru pa mora biti v nastalem grafu vsaj eno vozlišče C povezano s tremi vozlišči H, saj je graf acikličen. Na takšnem vozlišču C in njegovih štirih sosednjih vozliščih obratno uporabimo produkcijo p_2 in dobimo graf, ki še vedno predstavlja veljavno strukturno formulo nekega acikličnega ogljikovodika, le da je manjši od izhodiščnega grafa. Postopek ponavljamo, dokler grafa ne pretvorimo v prazen graf.

2.4.4 Sintaksna odločljivost

Rekers in Schürr [79, 80] sta pojem *sintaksne odločljivosti* (angl. *parsability*) kontekstno odvisne grafne gramatike definirala takole:

Definicija 2.36 (sintaksna odločljivost)**.** Gramatika $GG = (\mathcal{N}, \mathcal{T}, \mathcal{P})$ je *sintaksno odločljiva* (angl. *parsable*) natanko tedaj, ko so za vsako produkcijo $p \in \mathcal{P}$ izpolnjeni sledeči pogoji:

(1) Graf $Rhs[p]$ je povezan.

(2) Pri vseh nezačetnih produkcijah je graf $Lhs[p]$ neprazen.

(3) Množica $Xrhs[p]$ je neprazna.

(4) Produkcija p izpolnjuje t.i. *plastni pogoj* (angl. *layering condition*), ki ga bomo opredelili v definiciji 2.40.

Rekers in Schürr sta pokazala sledeče: če gramatika GG izpolnjuje pogoje iz definicije 2.36, potem je za *poljuben* graf G mogoče v končnem času in prostoru ugotoviti, ali velja $G \in L(GG)$. Pri sintaksno odločljivih gramatikah bo Rekers-Schürrov sintaksni analizator [79, 80] pravilno deloval za poljuben vhodni graf. V razdelku 3.4.1 bomo videli, da naša izboljšana različica Rekers-Schürrovega sintaksnega analizatorja [37] ne potrebuje prve zahteve, saj deluje tudi za gramatike, ki vsebujejo produkcije z nepovezanimi desnimi stranmi.

Pred predstavitvijo plastnega pogoja moramo opredeliti nekaj pomožnih konceptov:

Definicija 2.37 (leksikografska urejenost)**.** Celoštevilski vektor $v = (v_1, \ldots, v_n)$ je *leksikografsko manjši* od celoštevilskega vektorja $w = (w_1, \ldots, w_n)$ (oznaka: $v <_{\text{lex}} w$) natanko v primeru, če obstaja indeks $j \in 1\,..\,n$, tako da veljata sledeča pogoja:

1. $v_j < w_j$;

2. $v_i = w_i$ pri vseh $i < j$.

Definicija 2.38 (plastna funkcija, plast)**.** Pri podani gramatiki $(\mathcal{N}, \mathcal{T}, \mathcal{P})$ in celem številu $M \geq 0$ imenujemo preslikavo $layer\colon \mathcal{N} \cup \mathcal{T} \to 0\,..\,M$ *plastna funkcija* (angl. *layering function*). Vrednost $layer(A)$ za oznako $A \in \mathcal{N} \cup \mathcal{T}$ imenujemo *plast* (angl. *layer*) oznake A. Rekli bomo, da oznaka A *pripada* plasti k, če je $layer(A) = k$.

Definicija 2.39 (plastni vektor)**.** *Plastni vektor* (angl. *layering vector*) grafa G je vektor

$$lvec(G) = (\langle G\rangle_0,\ \langle G\rangle_1,\ \ldots,\ \langle G\rangle_M), \tag{2.3}$$

pri čemer

$$\langle G\rangle_i = |\{x \in G \mid layer(label[G](x)) = i\}| \tag{2.4}$$

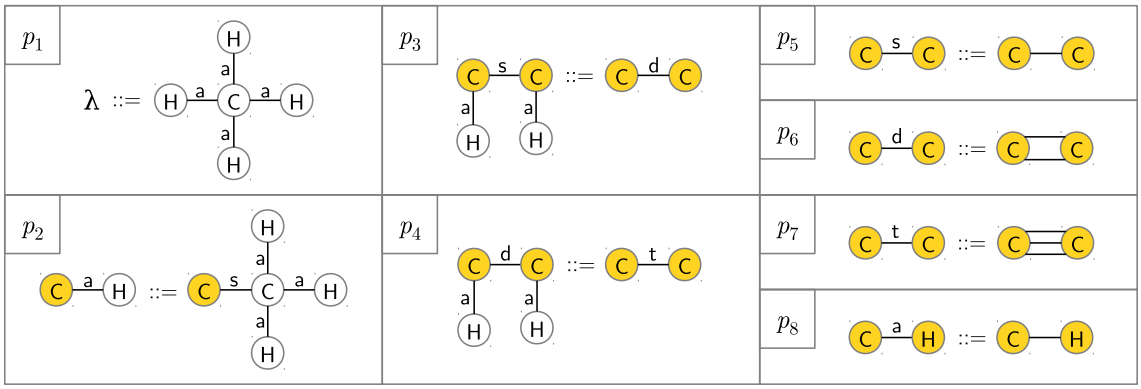
označuje število elementov grafa G, katerih oznake pripadajo plasti i.

S pomočjo navedenih pomožnih definicij sta Rekers in Schürr definirala plastni pogoj takole:

Definicija 2.40 (plastni pogoj)**.** Gramatika $GG = (\mathcal{N}, \mathcal{T}, \mathcal{P})$ izpolnjuje plastni pogoj natanko tedaj, ko obstaja celo število $M \geq 0$ in plastna funkcija $layer\colon \mathcal{N} \cup \mathcal{T} \to 0\,..\,M$, tako da za vsako produkcijo $p \in \mathcal{P}$ velja $lvec(Lhs[p]) <_{\text{lex}} lvec(Rhs[p])$.

Plastni pogoj preprečuje cikle v množici produkcij (npr. $G_1 ::= G_2$, $G_2 ::= G_3$, $G_3 ::= G_1$), saj zahteva, da mora biti pri vsaki produkciji plastni vektor desne strani leksikografsko večji od plastnega vektorja leve strani. Plastni pogoj zagotavlja, da se bo tudi naiven, na sestopanju temelječ sintaksni analizator (algoritem, ki poskuša podani graf G z obratnimi uporabami produkcij pretvoriti v prazen graf, ob neuspehih pa sestopa) prej ali slej ustavil, saj vsaka obratna uporaba produkcije leksikografsko zmanjša plastni vektor grafa, ki ga sintaksno analiziramo. Plastni pogoj lahko razumemo kot analogijo pogoju, ki pri kontekstno odvisnih besedilnih gramatikah določa, da mora biti leva stran produkcije krajša od desne. Vendar pa je plastni pogoj manj omejujoč. Če v vsaki produkciji leva stran vsebuje manj elementov kot desna, potem lahko plastnemu pogoju zadostimo enostavno tako, da plasti vseh oznak postavimo na isto vrednost (recimo na 0). Po drugi strani pa lahko plastni pogoj izpolnimo tudi v primerih, ko je pri nekaterih (ali celo pri vseh) produkcijah leva stran večja od desne.

Gramatika GG_{AHCt} ne izpolnjuje plastnega pogoja. Ne glede na to, koliko plasti vzamemo in kako po njih razporedimo oznake C, H in $\#$, bo za produkcijo p_3 vedno veljalo $lvec(Lhs[p_3]) >_{\text{lex}} lvec(Rhs[p_3])$, saj za vsako od oznak C, H in $\#$ velja, da leva stran vsebuje več (ali kvečjemu enako število) elementov te oznake kot desna stran. Plastnemu pogoju pa je mogoče zadostiti, če uvedemo dodatne (nekončne) oznake. Gramatika $GG_{\text{AHC}} = (\{\mathsf{a}, \mathsf{s}, \mathsf{d}, \mathsf{t}\}, \{\mathsf{C}, \mathsf{H}, \#\}, \mathcal{P})$, kjer je množica produkcij $\mathcal{P}$ prikazana na sliki 2.16, generira enako množico grafov kot gramatika GG_{AHCt}, vendar pa izpolnjuje plastni pogoj. Če postavimo $M = 3$, plastno funkcijo pa definiramo v skladu s predpisom v tabeli 2.1, potem za leve in desne strani posameznih produkcij dobimo plastne vektorje, kot jih prikazuje tabela 2.2. Kot vidimo, je pri vsaki produkciji plastni vektor leve strani leksikografsko manjši od plastnega vektorja desne strani, zato gramatika GG_{AHC} izpolnjuje plastni pogoj. Ker so izpolnjeni tudi drugi pogoji iz definicije 2.36, lahko zaključimo, da je gramatika GG_{AHC} sintaksno odločljiva.



Slika 2.16: *Gramatika* GG_{AHC}.

Tabela 2.1: *Primer plastne funkcije za gramatiko* GG_{AHC}.

oznaka	C	#	t	d	s	a	H
plast	0	0	1	2	3	3	3

Tabela 2.2: *Plastni vektorji za gramatiko* GG_{AHC} *pri plastni funkciji iz tabele 2.1.*

p	$lvec(Lhs[p])$	$lvec(Rhs[p])$	$lvec(Lhs[p]) <_{\text{lex}} lvec(Rhs[p])$?
p_1	$(0,0,0,0)$	$(1,0,0,8)$	da
p_2	$(1,0,0,2)$	$(2,0,0,7)$	da
p_3	$(2,0,0,5)$	$(2,0,1,0)$	da
p_4	$(2,0,1,4)$	$(2,1,0,0)$	da
p_5	$(2,0,0,1)$	$(3,0,0,0)$	da
p_6	$(2,0,1,0)$	$(4,0,0,0)$	da
p_7	$(2,1,0,0)$	$(5,0,0,0)$	da
p_8	$(1,0,0,2)$	$(2,0,0,1)$	da

POGLAVJE

3

SINTAKSNA ANALIZA PRI GRAFNIH GRAMATIKAH

V tem poglavju predstavljamo izvirne izboljšave Rekers-Schürrovega sintaksnega analizatorja za grafne gramatike. Ena od izboljšav razširja razred gramatik, ki jih sintaksni analizator lahko sprejema na vhodu, saj odpravlja zahtevo po povezanih desnih straneh produkcij. Ostale izboljšave pa so namenjene povečanju časovne in prostorske učinkovitosti sintaksnega analizatorja. Izboljšave smo ovrednotili s testiranjem na različnih kombinacijah vhodnih grafnih gramatik in zaporedij grafov. Osnovo za pričujoče poglavje predstavlja naš članek za revijo IET Software [37], vendar pa na tem mestu predstavljamo še dve dodatni izboljšavi, ostale izboljšave pa so opisane podrobneje in natančneje kot v članku. Tudi poglavje o eksperimentalnih rezultatih je v primerjavi s člankom bistveno širše.

Podpoglavje 3.1 služi kot uvod in predstavitev izbranih sorodnih del. V podpoglavju 3.2 prikažemo grafne gramatike, na katerih smo izboljšave testirali. Na testne grafne gramatike se namreč sklicujemo tudi v drugih podpoglavjih, ne zgolj v podpoglavju o eksperimentalnem vrednotenju. Podpoglavje 3.3 predstavi izhodiščni Rekers-Schürrov sintaksni analizator. Podpoglavje 3.4 je namenjeno izvirnim izboljšavam. V podpoglavju 3.5 eksperimentalno ovrednotimo vse izboljšave, podpoglavje 3.6 pa zaključi poglavje o sintaksni analizi.

3.1 Uvod in sorodna dela

Problem sintaksne analize pri grafnih gramatikah smo neformalno predstavili že v razdelku 1.1.2, v podpoglavju 2.3 pa smo govorili o sorodnem problemu pripadnosti. Pri obeh problemih je vhod sestavljen iz grafne gramatike in grafa, vendar pa nas pri problemu pripadnosti zanima samo odgovor na vprašanje glede pripadnosti grafa jeziku gramatike, pri problemu sintaksne analize pa bi v primeru pozitivnega odgovora na to vprašanje želeli pridobiti tudi izpeljavo podanega grafa v podani grafni gramatiki. Sintaksni analizator je algoritem, ki rešuje problem sintaksne analize. Kot smo napovedali, se bomo ukvarjali s sintaksno analizo kontekstno odvisnih grafnih gramatik. Sintaksni analizator za tovrstne gramatike sta iznašla Rekers in Schürr [79, 80].

Po implementaciji Rekers-Schürrovega sintaksnega analizatorja se je njegova neučinkovitost pri nekaterih primerih vhodnih gramatik hitro pokazala. Po analizi vzrokov za neučinkovitost smo se lotili izboljšav, ki jih opisujemo v tem poglavju. Izboljšani sintaksni analizator v najslabšem primeru še vedno porabi eksponentno mnogo časa v odvisnosti od velikosti vhodnega grafa (pri fiksni vhodni gramatiki), saj je zaradi NP-težkosti malo verjetno, da obstaja polinomski algoritem za sintaksno analizo splošnih kontekstno odvisnih gramatik. Kljub temu pa smo zmanjšali porabo časa in prostora za mnoge praktične primere gramatik. V nekaterih primerih vhodnih gramatik smo računsko zahtevnost dejansko zmanjšali z eksponentne na polinomsko. Poleg tega smo odpravili omejitev (1) iz definicije 2.36, ki jo je predpisoval izhodiščni sintaksni analizator. Izboljšani algoritem sprejema tudi gramatike, ki vsebujejo produkcije z nepovezanimi desnimi stranmi.

V svetu »običajnih« programskih jezikov, sintaktično opredeljenih s kontekstno neodvisnimi besedilnimi gramatikami, sintaksni analizatorji nastopajo kot sestavni deli prevajalnikov, analizatorjev kode, razhroščevalnikov itd. Sintaksni analizator za grafne gramatike bi prišel v poštev pri grafnih programskih jezikih. Kot bomo pokazali v tem poglavju, smo izboljšani Rekers-Schürrov sintaksni analizator med drugim preizkušali tudi na gramatiki diagramov poteka, ki definira sintakso preprostega grafnega programskega jezika. Sintaksni analizator smo uporabili tudi v naši metodi za indukcijo grafnih gramatik, ki jo bomo opisali v poglavju 4. V poglavju 5 pa bomo pokazali, da je sintaksni analizator mogoče smiselno uporabiti tudi na področju domensko specifičnega modeliranja.

Sintaksni analizatorji obstajajo za večino tipov grafnih gramatik, ki smo jih navedli v podpoglavju 2.3. Nekateri od njih se zgledujejo po algoritmu CYK (Cocke-Younger-Kasami) [60, 67] ali po Earleyjevem sintaksnem analizatorju [87]. Kot smo že omenili, pa je sintaksna analiza celo pri splošnih povezavnih gramatikah NP-težak problem, zato je polinomsko časovno zahtevnost mogoče doseči samo pri gramatikah z določenimi omejitvami [15, 24, 50, 60, 98]. Sintaksni analizatorji obstajajo tudi za relacijske gramatike [92], podrazred adaptivnih zvezdnih gramatik [67] in podrazred rezerviranih grafnih gramatik [103].

Kot smo že omenili, lahko pri sintaksni analizi sintaksno odločljivih kontekstno odvisnih grafnih gramatik uberemo t.i. naivni pristop. V tem pristopu poskušamo graf, ki ga želimo sintaksno analizirati, z obratnimi uporabami produkcij pretvoriti v enega od začetnih grafov gramatike, pri čemer sistematično (s pomočjo sestopanja)

preizkušamo vse možnosti, dokler nam ne uspe. Takšen algoritem je sicer enostaven, vendar pa lahko hitro podleže kombinatorični eksploziji — še zlasti v primeru vhodnih grafov, ki ne pripadajo jeziku vhodne gramatike. Kljub temu so se Bottoni in sod. [11] lotili izboljšave tega pristopa. Njihov algoritem ugotavlja, kdaj je sestopanje sploh potrebno, kadar pa se mu ni mogoče izogniti, poskuša z njim vsaj karseda dolgo odlašati. Avtorji svojega algoritma žal niso eksperimentalno ovrednotili, prav tako pa nam ni znano, ali obstaja eksperimentalna primerjava med algoritmom Bottonija in sod. in Rekers-Schürrovim pristopom.

3.2 Testne grafne gramatike

V tem podpoglavju bomo predstavili grafne gramatike, ki smo jih uporabljali pri eksperimentalnem delu (podpoglavje 3.5). Na testne gramatike se bomo sklicevali tudi v drugih delih tega poglavja.

Produkcije testnih gramatik so prikazane na slikah 3.1 in 3.2. Spomnimo se, da so z rumeno barvo označeni elementi kontekstnih grafov (*Common*) posameznih produkcij. Opišimo še druge lastnosti posameznih gramatik:

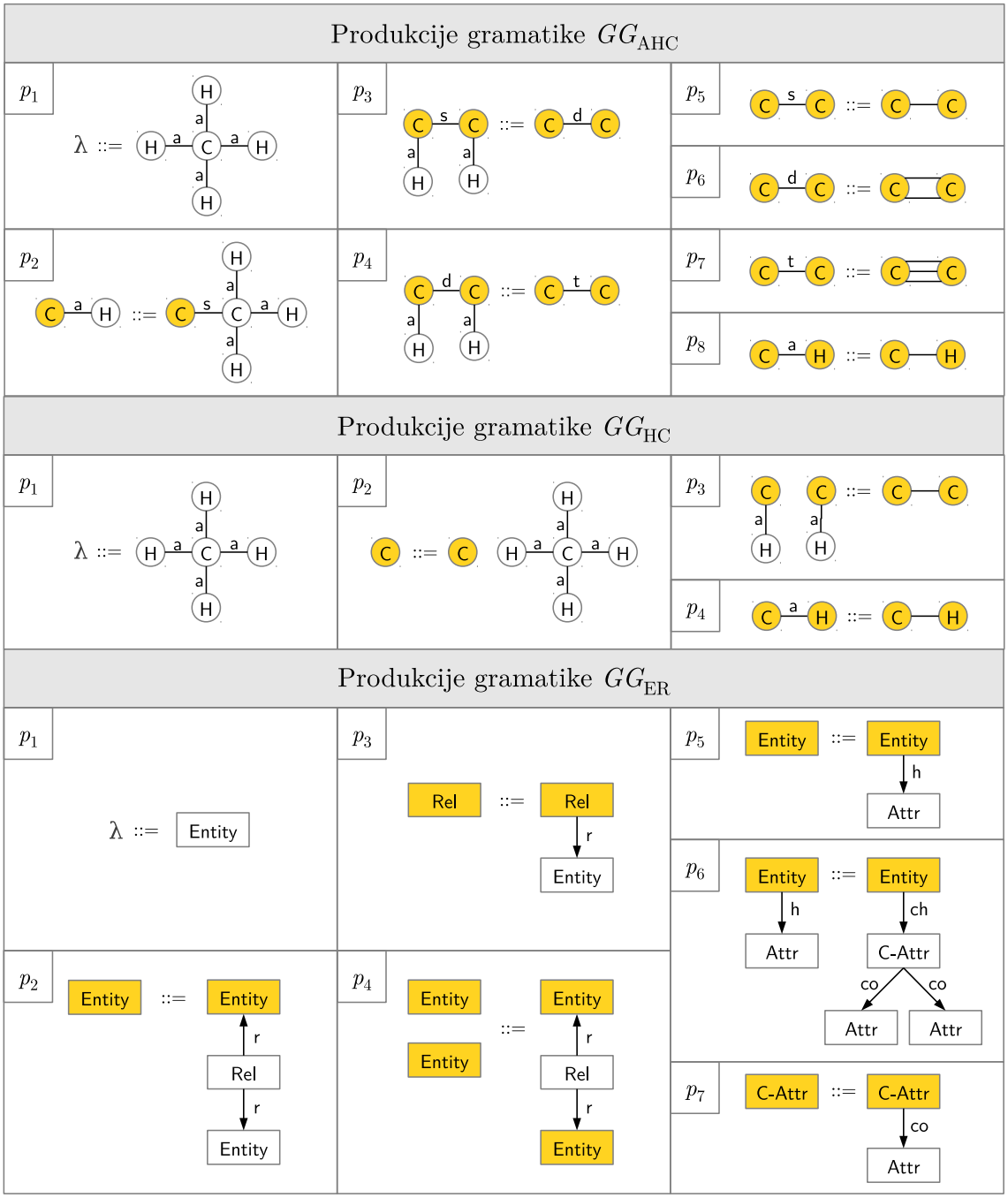
Gramatika GG_{AHC}**:** Za to gramatiko velja $\mathcal{N} = \{\mathsf{a}, \mathsf{s}, \mathsf{d}, \mathsf{t}\}$ in $\mathcal{T} = \{\mathsf{C}, \mathsf{H}, \#\}$. Kot smo videli v razdelku 2.4.4, gramatika GG_{AHC} generira jezik strukturnih formul acikličnih ogljikovodikov (angl. ***a**cyclic **h**ydro**c**arbons*).

Gramatika GG_{HC}**:** Za to gramatiko velja $\mathcal{N} = \{\mathsf{a}\}$ in $\mathcal{T} = \{\mathsf{C}, \mathsf{H}, \#\}$. Gramatika GG_{HC} generira grafe, sestavljene iz ene ali več povezanih komponent, od katerih vsaka predstavlja neko veljavno strukturno formulo ogljikovodika (angl. ***h**ydro**c**arbon*). To trditev lahko dokažemo na podoben način kot pri gramatiki GG_{AHC} oziroma GG_{AHCt} (razdelek 2.4.3).

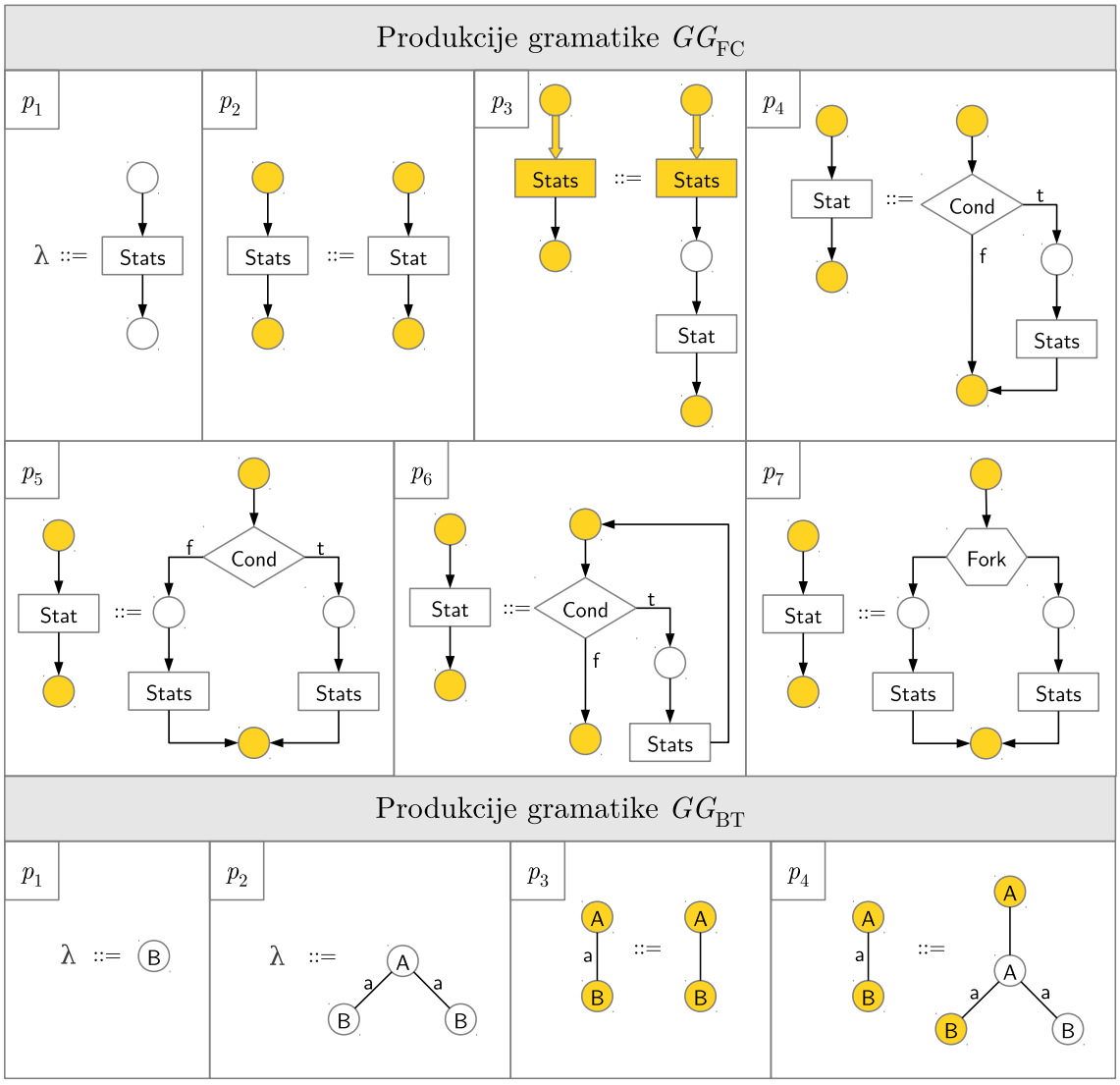
Z gramatiko GG_{HC} je možno generirati strukturno formulo *poljubnega* ogljikovodika, acikličnega ali cikličnega. Gradnjo ciklov nam omogoča produkcija p_3, saj je z njeno uporabo možno vzpostaviti vez med poljubnima vozliščema C. Slika 3.3 prikazuje izpeljavo strukturne formule ciklopropena v gramatiki GG_{HC}.

Gramatika GG_{ER}**:** V tej gramatiki so vse oznake končne. Gramatiko GG_{ER} smo povzeli po članku Rekersa in Schürra [80]. Gramatika generira diagrame ER (entiteta-razmerje, angl. ***e**ntity-**r**elationship*). Produkciji p_2 in p_4 zagotavljata, da vsaka relacija (vozlišče z oznako Rel) povezuje najmanj dve entiteti (vozlišči z oznako Entity), produkcija p_3 pa omogoča vezavo dodatnih entitet na posamezno relacijo. Produkciji p_5 in p_6 omogočata dodajanje enostavnih (Attr) in sestavljenih ($\mathsf{C\text{-}Attr}$) atributov k entitetam. Zaradi produkcije p_7 je vsakemu sestavljenemu atributu možno dodati poljubno število navadnih atributov.

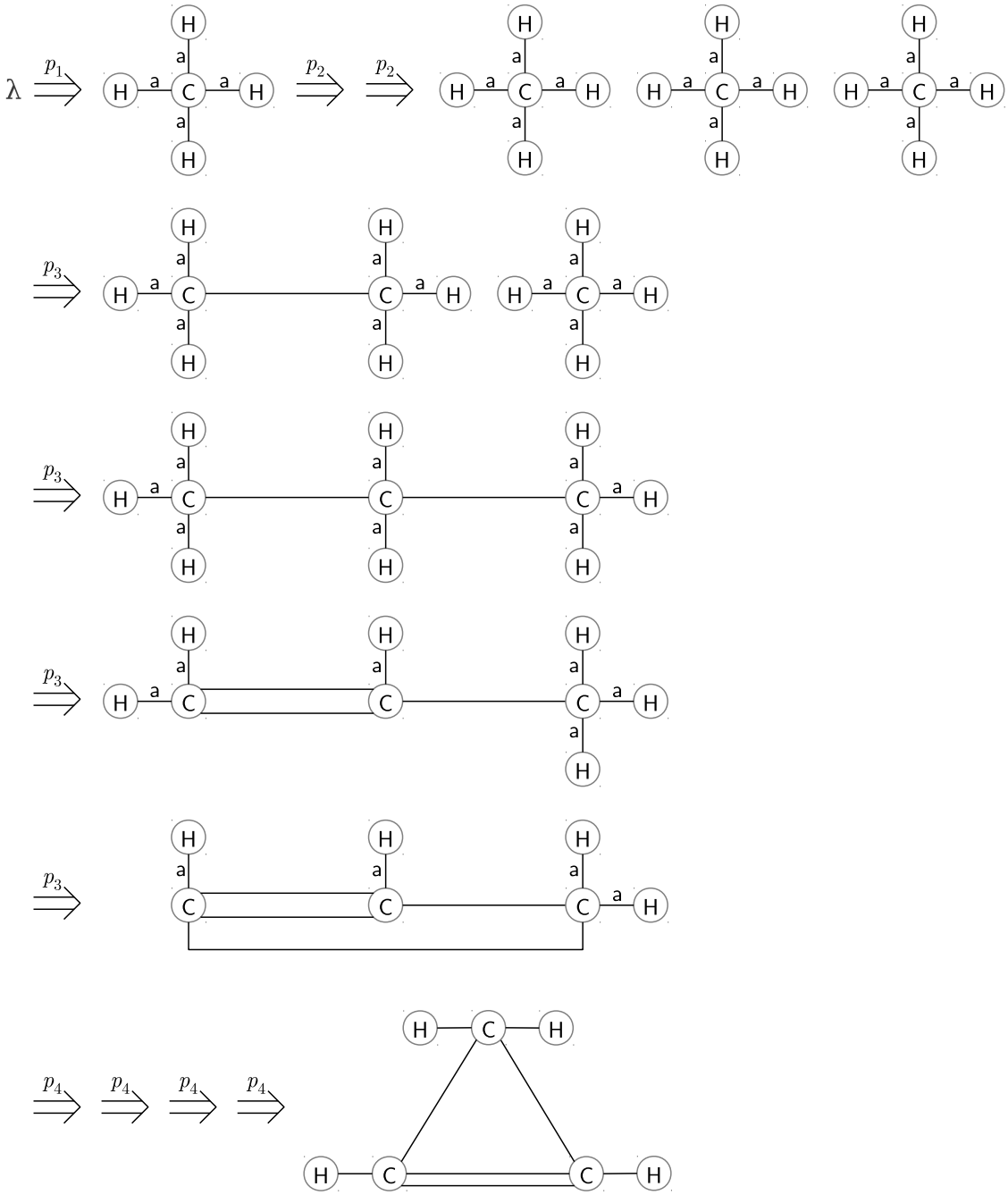
Gramatika GG_{FC}**:** V tej gramatiki je oznaka Stats nekončna, ostale pa so končne. Gramatika GG_{FC} generira jezik neke oblike diagramov poteka (angl. ***flowc**harts*). Produkciji p_2 in p_3 omogočata tvorbo poljubno dolgega zaporedja stavkov. Produkcije od p_4 do p_7 določajo možne vrste stavkov: stavek je lahko pogojnik tipa *if-then* (produkcija p_4), pogojnik tipa *if-then-else* (produkcija p_5), zanka



Slika 3.1: *Produkcije gramatik GG_{AHC}, GG_{HC} in GG_{ER}.*



Slika 3.2: *Produkcije gramatik GG_{FC} in GG_{BT}.*



Slika 3.3: *Izpeljava strukturne formule ciklopropena v gramatiki* GG_{HC}.

while (produkcija p_6) ali vejitev na dva vzporedna tokova izvajanja (produkcija p_7).

Gramatika GG_{BT}: V tej gramatiki je oznaka a nekončna, oznake A, B in $\#$ pa so končne. Gramatika GG_{BT} generira jezik dvojiških dreves (angl. ***b**inary **t**rees*), v katerih imajo notranja vozlišča oznako A, listi pa oznako B. Produkcija p_1 tvori drevo z enim samim listom. Kot izhodišče za gradnjo netrivialnih dreves služi produkcija p_2. Produkcija p_3 je namenjena odstranjevanju nekončnih oznak a, produkcija p_4 pa povečevanju drevesa. Uporaba produkcije p_4 vstavi novo notranje vozlišče in nov list v trenutno drevo.

3.3 Izvirni Rekers-Schürrov sintaksni analizator

V tem podpoglavju predstavljamo izvirni Rekers-Schürrov sintaksni analizator. Sintaksni analizator bomo opisali do te mere, da bo mogoče razumeti osnove njegovega delovanja in izboljšave, ki jih bomo predstavili v podpoglavju 3.4. Podrobnosti, ki temu namenu ne služijo, bomo izpustili, saj je podroben opis sintaksnega analizatorja podan v članku [80] in tehničnem poročilu [79] Rekersa in Schürra, mnogi implementacijski vidiki pa so predstavljeni v magistrski nalogi Vermeulena [97].

V razdelku 3.3.1 bomo sintaksni analizator na kratko pregledali, v razdelkih 3.3.2–3.3.4 pa bomo njegovo delovanje podrobneje predstavili.

3.3.1 Pregled delovanja

Rekers-Schürrov sintaksni analizator na vhodu sprejme kontekstno odvisno grafno gramatiko GG in graf G. Algoritem predpostavlja, da gramatika GG izpolnjuje pogoje sintaksne odločljivosti (definicija 2.36), sicer ustavljivost ni zagotovljena. Če velja $G \in L(GG)$, potem sintaksni analizator proizvede neko izpeljavo grafa G v gramatiki GG, sicer pa sporoči, da graf ne pripada jeziku gramatike. Spomnimo se, da je izpeljava grafa G v gramatiki GG neko zaporedje uporab produkcij, ki se prične s praznim grafom, zaključi pa z grafom G.

Sintaksni analizator deluje v dveh ločenih fazah, ki ju lahko v grobem opredelimo takole:

Faza 1 prečeše vhodni graf G in proizvede množico $\mathcal{D}$, ki vsebuje potencialne sestavne dele izpeljave grafa G v vhodni gramatiki GG. Če graf G pripada jeziku gramatike GG, potem množica $\mathcal{D}$ zanesljivo vsebuje sestavne dele, iz katerih je mogoče zgraditi poljubno izpeljavo grafa G. Množica $\mathcal{D}$ je torej popolna, a pogosto redundantna. Poleg izgradnje množice $\mathcal{D}$ faza 1 tudi vzpostavi določene relacije med njenimi elementi, ki bodo prišle prav v fazi 2.

Faza 2 poskuša zgraditi izpeljavo grafa G iz njenih potencialnih sestavnih delov, ki jih vsebuje množica $\mathcal{D}$. V ta namen mora ugotoviti, kateri elementi množice $\mathcal{D}$ lahko sodelujejo v izpeljavi in v kakšnem vrstnem redu morajo nastopati. Faza 2 išče zaporedje, ki tvori izpeljavo, po načelu sestopanja, vendar si izdatno pomaga z relacijami, ki jih je vzpostavila faza 1.

Pred opisom obeh faz sintaksnega analizatorja se spomnimo, da je l-slika (oz. r-slika) produkcije p v grafu G homomorfna slika leve (oz. desne) strani produkcije p v tem grafu, pri čemer so elementi $Xlhs[p]$ (oz. $Xrhs[p]$) na elemente grafa G preslikani injektivno. Preslikava med elementi produkcije in elementi grafa se imenuje p-homomorfizem. Definirajmo še sledeči sorodni pojem:

Definicija 3.1 (produkcijski primerek)**.** Naj bo p kontekstno odvisna produkcija, G graf, $h\colon Union[p] \to G$ pa p-homomorfizem. Potem je četverica $pi = (p, h(Xlhs[p]), h(Common[p]), h(Xrhs[p]))$ *primerek* produkcije p v grafu G. Sestavne dele produkcijskega primerka bomo označevali na enak način kot sestavne dele produkcije:

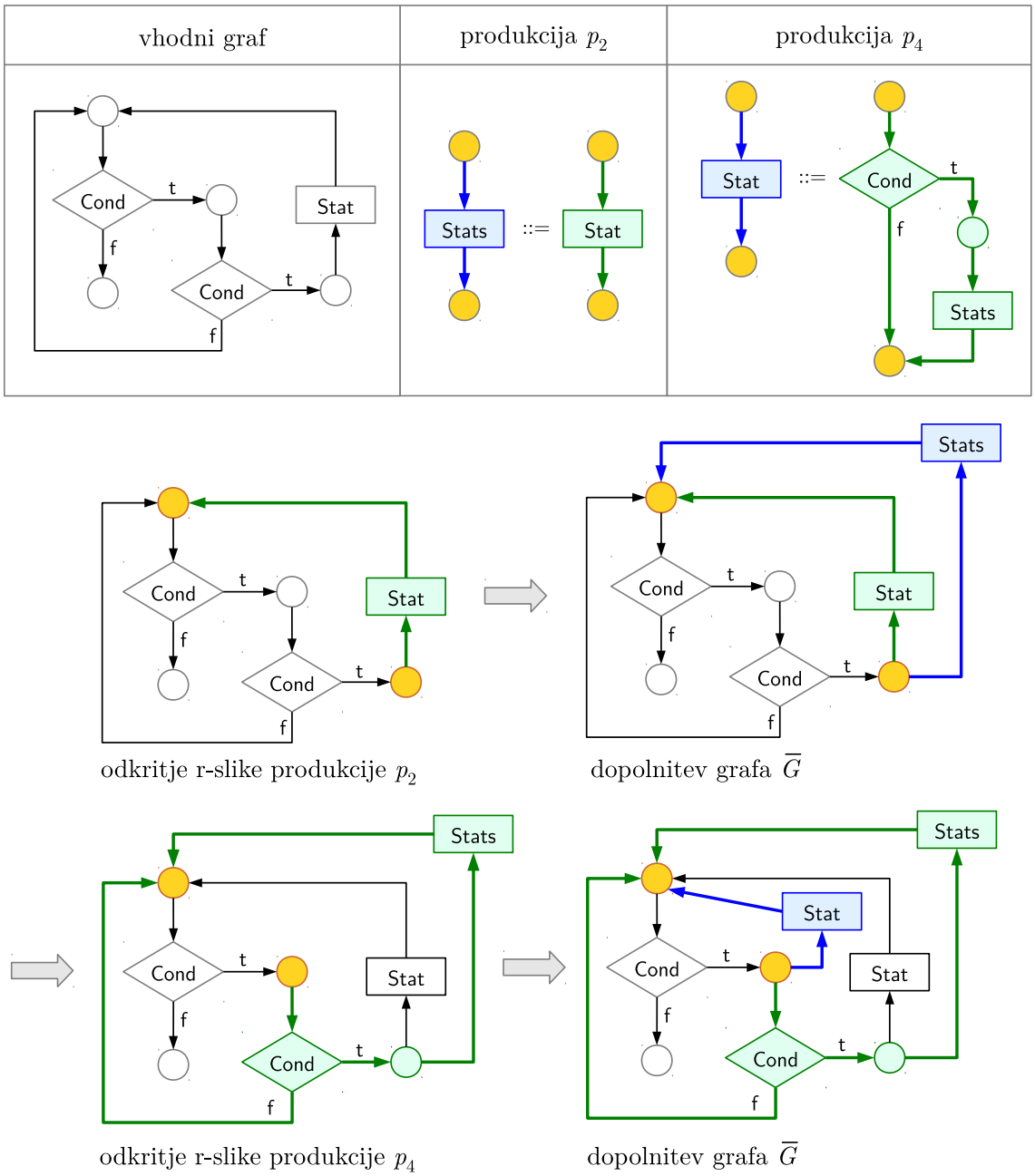
- $Lhs[pi] = h(Lhs[p])$.
- $Rhs[pi] = h(Rhs[p])$.
- $Common[pi] = h(Common[p])$.
- $Xlhs[pi] = h(Xlhs[p])$.
- $Xrhs[pi] = h(Xrhs[p])$.
- $Union[pi] = h(Union[p])$.

Primerek produkcije p v gostiteljskem grafu G je torej p-homomorfna slika celotne produkcije p v grafu G. Tako kot pri produkciji p tudi za primerek pi velja, da množice grafnih elementov $Lhs[pi]$, $Rhs[pi]$, $Common[pi]$ in $Union[pi]$ vedno tvorijo veljavne podgrafe grafa G, pri množicah $Xlhs[pi]$ in $Xrhs[pi]$ pa to v splošnem ne drži.

3.3.2 Faza 1

Prva faza sintaksnega analizatorja na začetku ustvari graf $\overline{G}$ kot kopijo vhodnega grafa G. Nato v grafu $\overline{G}$ sistematično išče r-slike posameznih produkcij. Kadarkoli odkrije graf $R \sqsubseteq \overline{G}$ z lastnostjo $R = h_R(Rhs[p])$ za neko produkcijo p in nek p-homomorfizem h_R, dopolni graf $\overline{G}$ tako, da na elemente c-slike produkcije p znotraj grafa R (tj. $h_R(Common[p])$) pripne kopije elementov $Xlhs[p]$ na enak način, kot so elementi $Xlhs[p]$ vezani na elemente $Common[p]$ na levi strani produkcije p. Elementi grafa R skupaj z dodanimi kopijami elementov $Xlhs[p]$ tvorijo *primerek* produkcije p.

Slika 3.4 prikazuje začetni del sintaksne analize nekega grafa pri podani gramatiki GG_{FC} (slika 3.2). Faza 1 najprej v grafu odkrije r-sliko produkcije p_2 in graf $\overline{G}$ ustrezno dopolni. Po dopolnitvi grafa $\overline{G}$ vsi obarvani elementi *skupaj* tvorijo primerek produkcije p_2. Zaradi dodanih elementov bo faza 1 kasneje odkrila še r-sliko produkcije p_4 in ponovno dopolnila graf $\overline{G}$ ter tako ustvarila še primerek produkcije p_4. Kot vidimo, faza 1 pri iskanju r-slik produkcij ne upošteva samo izhodiščnih elementov grafa $\overline{G}$, ampak tudi elemente, dodane kot posledice predhodnih odkritij r-slik. Pomembno je razumeti, da se elementi v graf $\overline{G}$ zgolj dodajajo; obstoječi elementi se nikoli ne brišejo ali spreminjajo.



Slika 3.4: *Začetni del sintaksne analize v fazi 1.*

Faza 1 ponavlja postopek odkrivanja r-slik produkcij in posledičnega dopolnjevanja grafa $\overline{G}$, dokler ne najde vseh možnih r-slik vseh produkcij vhodne gramatike GG. Podrobnosti algoritma so podane v Rekers-Schürrovem tehničnem poročilu [79].

Opisani postopek bi lahko zašel v neskončno zanko, saj obstaja možnost, da kopije elementov *Xlhs*, ki jih postopek pripne na r-sliko produkcije v grafu $\overline{G}$, skupaj z nekaterimi elementi r-slike tvorijo novo r-sliko *iste* produkcije. Če bi se to primerilo, bi nastal neskončen cikel odkritij in dopolnitev. Vendar pa je faza 1 zasnovana tako, da produkcijske primerke, ki bi lahko povzročili neskončno zanko, sproti odstranjuje. Za tovrstne produkcijske primerke velja relacija *nekonsistentnosti*, ki jo faza 1 sproti vrednoti. O relacijah med produkcijskimi primerki bomo govorili v razdelku 3.3.3.

Pri iskanju r-slik produkcij v grafu $\overline{G}$ si faza 1 pomaga z *iskalnimi načrti* desnih strani produkcij. Iskalni načrt (angl. *search plan*) povezanega grafa G je seznam navodil za zaporedno obiskovanje njegovih elementov. Prvo navodilo iskalnega načrta ima sledečo obliko (povzeto po [79]):

$\langle head(v\colon A)\rangle$: Poišči vozlišče z oznako A in mu dodeli identifikator v.

Pri ostalih navodilih so možne tri oblike:

(1) $\langle e\colon v \xrightarrow{a} (w\colon B)\rangle$: Prični v že obiskanem vozlišču z identifikatorjem v, potuj po povezavi z oznako a do ciljnega vozlišča z oznako B in dodeli povezavi identifikator e, ciljnemu vozlišču pa w.

(2) $\langle e\colon v \xleftarrow{a} (w\colon B)\rangle$: Prični v že obiskanem vozlišču z identifikatorjem v, potuj po povezavi z oznako a v *obratni smeri* do izvornega vozlišča z oznako B in dodeli povezavi identifikator e, izvornemu vozlišču pa w.

(3) $\langle e\colon v \xrightarrow{a} w\rangle$: Preveri, ali obstaja povezava z oznako a, ki vodi od že obiskanega vozlišča z identifikatorjem v do že obiskanega vozlišča z identifikatorjem w, in ji dodeli identifikator e.

Iskalni načrt mora popisati celoten graf. Tabela 3.1 prikazuje enega od številnih veljavnih iskalnih načrtov za desno stran produkcije p_4 gramatike GG_{FC}.

Tabela 3.1: *Iskalni načrt za desno stran produkcije p_4 gramatike GG_{FC}.*

korak	navodilo
1	$\langle head(v_1\colon \#)\rangle$
2	$\langle e_1\colon v_1 \xleftarrow{\#} (v_2\colon \mathsf{Stats})\rangle$
3	$\langle e_2\colon v_1 \xleftarrow{\mathsf{f}} (v_3\colon \mathsf{Cond})\rangle$
4	$\langle e_3\colon v_3 \xrightarrow{\mathsf{t}} (v_4\colon \#)\rangle$
5	$\langle e_4\colon v_4 \xrightarrow{\#} v_2\rangle$
6	$\langle e_5\colon v_3 \xleftarrow{\#} (v_5\colon \#)\rangle$

Faza 1 poišče r-sliko produkcije p v grafu $\overline{G}$ tako, da sledi iskalnemu načrtu za graf $Rhs[p]$, pri čemer upošteva, da morajo biti elementi množice $Xrhs[p]$ injektivno

preslikani na elemente r-slike. Za delovanje faze 1 moramo torej pripraviti iskalni načrt za vsako desno stran. Iskalne načrte lahko zgradimo s preprostim samodejnim postopkom, saj so s stališča ustavljivosti in pravilnosti delovanja faze 1 vsi veljavni iskalni načrti enakovredni. Poskrbeti moramo zgolj za to, da z navodili popišemo vsa vozlišča in povezave.

Ker mora faza 1 v sproti dopolnjujočem se grafu $\overline{G}$ poiskati vse r-slike produkcij, poskuša vsako navodilo vsakega iskalnega načrta izpolniti na vse možne načine. Če nekemu navodilu ustreza več elementov grafa $\overline{G}$, bo faza 1 prej ali slej obiskala vsakega od njih. Ta strategija zagotavlja pravilnost delovanja za poljuben iskalni načrt.

V nadaljevanju bomo potrebovali pojem *uporabe* produkcijskega primerka:

Definicija 3.2 (uporaba produkcijskega primerka)**.** Naj bo pi primerek produkcije p v grafu $\overline{G}$. Naj bo $G' \sqsubseteq \overline{G}$ graf, ki vsebuje vse elemente množice $Lhs[pi]$ (torej tisti del primerka, ki tvori l-sliko produkcije p), ne vsebuje pa nobenega elementa množice $Xrhs[pi]$. Potem primerek pi na grafu $G' \sqsubseteq \overline{G}$ *uporabimo* tako, da iz grafa G' odstranimo elemente množice $Xlhs[pi]$, nato pa na elemente množice $Common[pi]$ (ki je v grafu G' še vedno prisotna) vežemo elemente množice $Xrhs[pi]$.

Naj opozorimo na razliko med uporabo *produkcije* in uporabo *produkcijskega primerka*. Uporaba produkcije p doda v graf *kopije* elementov $Xrhs[p]$, uporaba produkcijskega primerka pi pa v graf doda elemente $Xrhs[pi] \subseteq Elements[\overline{G}]$ same po sebi. Vsak produkcijski primerek je sestavljen iz neke podmnožice elementov grafa $\overline{G}$, ki ga je faza 1 zgradila kot nadgraf grafa G.

Glavni izhod faze 1 je množica vseh produkcijskih primerkov $\mathcal{D}$, ki jih je iskalno-dopolnjevalni postopek ustvaril v grafu $\overline{G}$. Rekers in Schürr sta pokazala, da obstaja tesna povezava med produkcijskimi primerki in izpeljavo grafa G. Izpeljavo grafa G namreč lahko sestavimo iz uporab produkcijskih primerkov. Natančneje: če je graf G izpeljiv v gramatiki GG, potem obstaja množica produkcijskih primerkov $\mathcal{D}' \subseteq \mathcal{D}$, tako da neko zaporedje uporab primerkov iz množice $\mathcal{D}'$ tvori izpeljavo grafa G. Ker izpeljavo sestavimo iz produkcijskih primerkov, pridobljenih iz elementov grafa $\overline{G}$, je vsak graf, ki nastopa v izpeljavi, podgraf grafa $\overline{G}$.

Faza 2 bi množico $\mathcal{D}' \subseteq \mathcal{D}$ in ustrezno zaporedje produkcijskih primerkov v množici $\mathcal{D}'$ lahko poiskala z izčrpnim preiskovanjem množice $\mathcal{D}$. Vendar pa bi bil tak pristop računsko neizvedljiv že za majhne množice $\mathcal{D}$. Faza 1 zato med produkcijskimi primerki v množici $\mathcal{D}$ vzpostavi določene *relacije* (razdelek 3.3.3), ki v fazi 2 močno omejijo prostor iskanja, saj podajajo prednostna in izključevalna razmerja med produkcijskimi primerki v veljavni izpeljavi. Na primer, če sta primerka v relaciji **excludes***, potem lahko v poljubni veljavni izpeljavi grafa G nastopa kvečjemu eden od njiju. Poleg množice $\mathcal{D}$ potemtakem izhod faze 1 sestavljajo tudi vrednosti posameznih relacij.

Slika 3.5 prikazuje delovanje faze 1 na primeru gramatike GG_{FC} in grafa G, ki pripada jeziku te gramatike. Del (a) prikazuje rast grafa $\overline{G}$ v iskalno-dopolnjevalnem postopku. Na začetku faza 1 odkrije r-sliko produkcije p_2, sestavljeno iz elementov v_3, v_4, v_5, e_4 in e_5. Po odkritju te r-slike na c-sliko (vozlišči v_3 in v_4) veže kopije elementov $Xlhs[p_2]$ in tako ustvari produkcijski primerek pi_1. Nato faza 1 odkrije r-sliko produkcije p_4 in ustvari produkcijski primerek pi_2. In tako naprej. V delu

(b) so navedeni vsi produkcijski primerki, odkriti v fazi 1. Produkcijski primerek pi, ki pripada produkciji p, je podan kot četverica (p, $Xlhs[pi]$, $Common[pi]$, $Xrhs[pi]$). Del (c) prikazuje nekatere relacije med produkcijskimi primerki, ki jih bomo spoznali v razdelku 3.3.3. Del (d) prikazuje izpeljavo grafa G v gramatiki GG_{FC}, kot jo izdela faza 2. Vidimo, da je izpeljava sestavljena iz zaporedja uporab produkcijskih primerkov, da so vsi grafi v izpeljavi podgrafi grafa $\overline{G}$ in da je produkcijski primerek pi_3 odveč.

3.3.3 Relacije med produkcijskimi primerki

V tem razdelku bomo opisali posamezne relacije med produkcijskimi primerki, ki jih sproti vrednoti faza 1. Faza 1 neposredno uporablja le relacijo **inconsistent**, faza 2 pa relaciji **above** in **excludes***. Ostale relacije so pomožne; definirane so zgolj s ciljem lažjega in učinkovitejšega vrednotenja glavnih relacij.

Spomnimo se, da uporaba produkcijskega primerka pi na nekem grafu $G' \sqsubseteq \overline{G}$, ki nastopa v (nastajajoči) izpeljavi vhodnega grafa G, iz grafa G' najprej odstrani elemente $Xlhs[pi]$, nato pa vanj doda elemente $Xrhs[pi]$ in jih ustrezno veže na elemente $Common[pi]$. To je možno storiti le v primeru, če graf G' vsebuje vse elemente iz množice $Lhs[pi]$, a nobenega iz množice $Xrhs[pi]$.

Rekers in Schürr sta relacije med produkcijskimi primerki definirala takole:

Definicija 3.3 (relacija **conseqOf**)**.** Produkcijski primerek pi_2 je *posledica* (angl. *consequence of*) produkcijskega primerka pi_1 (pi_2 **conseqOf** pi_1) natanko tedaj, ko velja $pi_1 \neq pi_2$ in

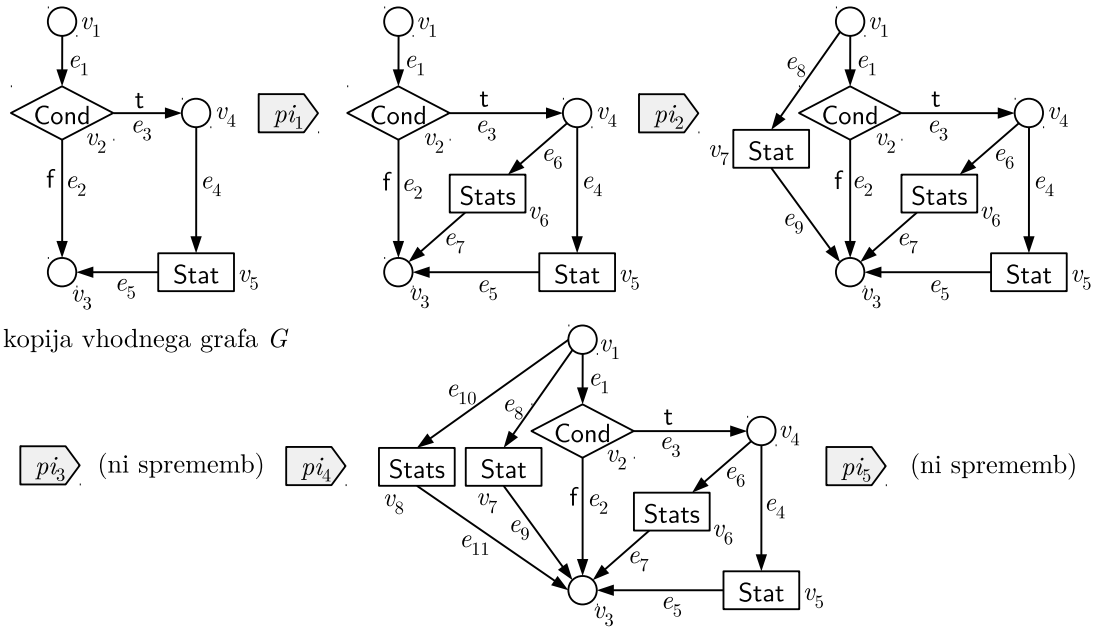
$$Rhs[pi_1] \cap Xlhs[pi_2] \neq \emptyset.$$

Rekers in Schürr sta uporabila oznako **consequence**, vendar pa po našem mnenju poimenovanje **conseqOf** bolj jasno izraža smer relacije. Če je produkcijski primerek pi_2 posledica produkcijskega primerka pi_1, potem mora poljubna izpeljava grafa G, ki vsebuje primerek pi_1, vsebovati tudi primerek pi_2. Primerek pi_2 se mora v izpeljavi nahajati kasneje kot primerek pi_1. To lahko ugotovimo na sledeči način: Elementi $Xlhs[pi_2]$ so elementi grafa $\overline{G}$, ki so nastali po odkritju r-slike produkcije, ki ji pripada primerek pi_2. Ti elementi v vhodnem grafu G ne obstajajo, saj jih je faza 1 *ustvarila*. Če izpeljava grafa G vsebuje primerek pi_1 in če velja $Rhs[pi_1] \cap Xlhs[pi_2] \neq \emptyset$, potem uporaba primerka pi_1 v izpeljavi doda (ali uporablja kot kontekst) določene elemente, ki jih uporaba primerka pi_2 odstrani. Primerek pi_2 je *edini*, ki lahko te elemente odstrani, saj so nastali ob nastanku primerka pi_2, kasneje pa se niso več spreminjali. Te elemente pa v izpeljavi grafa G prej ali slej *moramo* odstraniti, saj jih v grafu G ni. Zato mora poljubna izpeljava grafa G, ki vsebuje uporabo primerka pi_1, na neki točki med uporabo primerka pi_1 in zaključkom izpeljave vsebovati tudi uporabo primerka pi_2.

Definicija 3.4 (relacija **conseqOf**$^+$)**.** Relacija **conseqOf**$^+$ je tranzitivna ovojnica relacije **conseqOf**.

Definicija 3.5 (relacija **above**)**.** Produkcijski primerek pi_1 je *nad* (angl. *above*) produkcijskim primerkom pi_2 (pi_1 **above** pi_2) natanko tedaj, ko velja $pi_1 \neq pi_2$ in ko je izpolnjen vsaj eden od sledečih pogojev:

(a) Rast grafa $\overline{G}$ v fazi 1



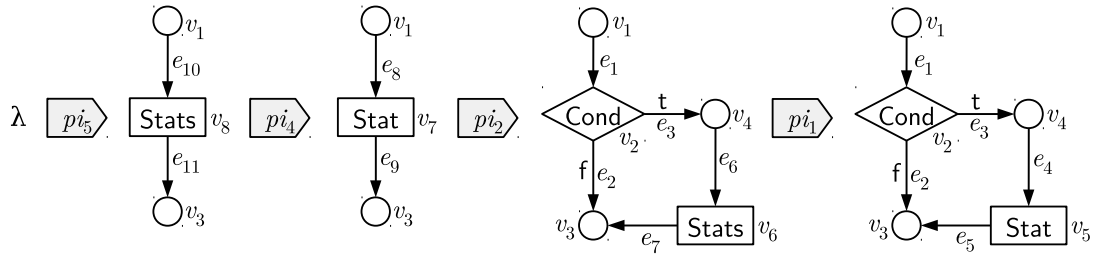
(b) Produkcijski primerki, ki jih ustvari faza 1

$pi_1 = (p_2, \{v_6, e_6, e_7\}, \{v_3, v_4\}, \{v_5, e_4, e_5\})$
$pi_2 = (p_4, \{v_7, e_8, e_9\}, \{v_1, v_3\}, \{v_2, v_4, v_6, e_1, e_2, e_3, e_6, e_7\})$
$pi_3 = (p_1, \emptyset, \emptyset, \{v_3, v_4, v_6, e_6, e_7\})$
$pi_4 = (p_2, \{v_8, e_{10}, e_{11}\}, \{v_1, v_3\}, \{v_7, e_8, e_9\})$
$pi_5 = (p_1, \emptyset, \emptyset, \{v_1, v_3, v_8, e_{10}, e_{11}\})$

(c) Relacije med ustvarjenimi produkcijskimi primerki

pi_1 **conseqOf** pi_2	pi_1 **conseqOf**$^+$ pi_2	pi_2 **above** pi_1	pi_2 **excludes** pi_3	pi_2 **excludes*** pi_3
pi_1 **conseqOf** pi_3	pi_1 **conseqOf**$^+$ pi_3	pi_3 **above** pi_1	pi_3 **excludes** pi_2	pi_3 **excludes*** pi_2
pi_2 **conseqOf** pi_4	pi_1 **conseqOf**$^+$ pi_4	pi_3 **above** pi_2	pi_3 **excludes** pi_5	pi_3 **excludes*** pi_4
pi_4 **conseqOf** pi_5	pi_1 **conseqOf**$^+$ pi_5	pi_3 **above** pi_4	pi_5 **excludes** pi_3	pi_3 **excludes*** pi_5
	pi_2 **conseqOf**$^+$ pi_4	pi_4 **above** pi_2		pi_4 **excludes*** pi_3
	pi_2 **conseqOf**$^+$ pi_5	pi_5 **above** pi_1		pi_5 **excludes*** pi_3
	pi_4 **conseqOf**$^+$ pi_5	pi_5 **above** pi_2		
		pi_5 **above** pi_4		

(d) Izpeljava vhodnega grafa, kot jo izdela faza 2



Slika 3.5: *Izvajanje faze 1 na grafu iz jezika* $L(GG_{\mathrm{FC}})$.

- pi_2 **conseqOf** pi_1.
- $Xrhs[pi_1] \cap Common[pi_2] \neq \emptyset$.
- $\exists e \in \mathcal{E}[Xlhs[pi_1]], \exists v \in \mathcal{V}[Xlhs[pi_2]] : v \in conn(e)$.

Če je produkcijski primerek pi_1 nad produkcijskim primerkom pi_2, potem mora v vsaki izpeljavi, ki vsebuje oba primerka, primerek pi_1 nastopati pred primerkom pi_2. Relacija **above** je šibkejša od relacije **conseqOf**, saj dopušča možnost, da v izpeljavi nastopa samo primerek pi_1. Če je primerek pi_2 posledica primerka pi_1, potem se mora primerek pi_1 v izpeljavi prav gotovo nahajati pred primerkom pi_2. To velja tudi tedaj, ko primerek pi_1 v tekoči graf v izpeljavi doda ($Xrhs[pi_1]$) vsaj enega od elementov, ki jih primerek pi_2 uporablja kot kontekst ($Common(pi_2)$). Tretji pogoj pa določa, da mora primerek pi_1 v izpeljavi nastopati pred primerkom pi_2 tudi tedaj, ko primerek pi_1 odstrani vsaj eno povezavo, pripeto na neko vozlišče, ki ga odstrani primerek pi_2. Če bi v tem primeru primerek pi_2 v izpeljavi uporabili pred primerkom pi_1, bi namreč dobili vsaj eno visečo povezavo.

Trojiška relacija $\mathbf{above}_{pi}$ je definirana kot omejitev relacije **above** na posledice nekega produkcijskega primerka:

Definicija 3.6 (relacija $\mathbf{above}_{pi}$)**.** Produkcijski primerki pi_1, pi_2 in pi so v relaciji pi_1 $\mathbf{above}_{pi}$ pi_2 natanko tedaj, ko velja pi_1 **above** pi_2, poleg tega pa je izpolnjen še vsaj eden od sledečih pogojev:

- $pi_1 = pi \wedge pi_2$ $\mathbf{conseqOf}^+$ pi.
- $pi_2 = pi \wedge pi_1$ $\mathbf{conseqOf}^+$ pi.
- pi_1 $\mathbf{conseqOf}^+$ $pi \wedge pi_2$ $\mathbf{conseqOf}^+$ pi.

Definicija 3.7 (relacija $\mathbf{above}^+_{pi}$)**.** Trojiška relacija $\mathbf{above}^+_{pi}$ je tranzitivna ovojnica relacije $\mathbf{above}_{pi}$.

Če velja relacija pi_1 $\mathbf{above}^+_{pi}$ pi_2, potem se mora v vsaki izpeljavi, ki vsebuje produkcijski primerek pi ter produkcijska primerka pi_1 in pi_2 kot njegovi neposredni ali posredni posledici, primerek pi_1 nahajati pred primerkom pi_2.

Definicija 3.8 (relacija **excludes**)**.** Produkcijski primerek pi_1 se *izključuje* (angl. *excludes*) s produkcijskim primerkom pi_2 (pi_1 **excludes** pi_2) natanko tedaj, ko velja $pi_1 \neq pi_2$ in ko je izpolnjen vsaj eden od sledečih pogojev:

- pi_1 **above** $pi_2 \wedge pi_2$ **above** pi_1.
- $Xrhs[pi_1] \cap Xrhs[pi_2] \neq \emptyset$.

Če se produkcijska primerka izključujeta, potem ne moreta oba hkrati nastopati v isti izpeljavi grafa G. Pri prvem pogoju za relacijo **excludes** medsebojno izključevanje sledi iz pomena relacije **above**. Drugi pogoj je izpolnjen, če produkcijska primerka v izpeljavi ustvarita vsaj en skupni element grafa $\overline{G}$. Takšna primerka se izključujeta, ker je v izpeljavi možno vsak element grafa $\overline{G}$ ustvariti kvečjemu enkrat (spomnimo se, da faza 1 elemente v graf $\overline{G}$ zgolj dodaja; nikoli jih ne odstranjuje ali spreminja).

Definicija 3.9 (relacija **excludes***)**.** Produkcijska primerka pi_1 in pi_2 sta v simetrični relaciji **excludes*** (pi_1 **excludes*** pi_2) natanko tedaj, ko je izpolnjen vsaj eden od sledečih pogojev:

- pi_1 **excludes** pi_2.
- $\exists pi'$: pi' **conseqOf**$^+$ $pi_1 \wedge pi'$ **excludes** pi_2.
- $\exists pi''$: pi'' **conseqOf**$^+$ $pi_2 \wedge pi_1$ **excludes** pi''.
- $\exists pi', pi''$: pi' **conseqOf**$^+$ $pi_1 \wedge pi''$ **conseqOf**$^+$ $pi_2 \wedge pi'$ **excludes** pi''.

Produkcijska primerka pi_1 in pi_2 sta torej v relaciji **excludes*** natanko tedaj, ko se primerek pi_1 ali ena od njegovih neposrednih ali posrednih posledic izključuje s primerkom pi_2 ali z eno od njegovih neposrednih ali posrednih posledic. Takšna primerka ne moreta nastopati skupaj v nobeni veljavni relaciji. Če bi izpeljava vsebovala tako primerek pi_1 kot tudi primerek pi_2, bi v skladu z definicijo relacije **conseqOf**$^+$ morala vsebovati tudi vse njune neposredne in posredne posledice. Vendar pa bi se v nastali množici produkcijskih primerkov po definiciji relacije **excludes*** vsaj dva med seboj izključevala, kar pomeni, da primerka pi_1 in pi_2 ne moreta biti sestavni del iste izpeljave.

Rekers in Schürr sledeče relacije nista definirala, nam pa bo koristila pri izboljšavi 3 (razdelek 3.4.3):

Definicija 3.10 (relacija **excludes**$^\star_{\text{self}}$)**.** Produkcijski primerek pi se *samoizključuje* (**excludes**$^\star_{\text{self}}(pi)$), če se izključuje z vsaj eno od svojih posledic:

$$\textbf{excludes}^\star_{\text{self}}(pi) \iff pi \textbf{ excludes}^\star pi$$

Definicija 3.11 (relacija **inconsistent**)**.** Produkcijski primerek pi je nekonsistenten (**inconsistent**(pi)), če izpolnjuje vsaj enega od sledečih pogojev:

- **excludes**$^\star_{\text{self}}(pi)$.
- $\exists pi', pi''$: pi' **above**$^+_{pi}$ $pi'' \wedge pi''$ **above**$^+_{pi}$ pi'.

Nekonsistentni produkcijski primerki ne morejo nastopati v nobeni izpeljavi. Poleg tega lahko takšen primerek v fazi 1 povzroči neskončno zanko dopolnjevanj grafa $\overline{G}$ in odkrivanj novih r-slik iste produkcije. Da takšne pojave preprečimo, moramo nekonsistentne produkcijske primerke odkrivati in odstranjevati sproti. Zato je treba relacijo **inconsistent** ovrednotiti za vsak novonastali produkcijski primerek v fazi 1. Ker pa je relacija **inconsistent** odvisna od vseh ostalih relacij, je treba v fazi 1 vse relacije vrednotiti sproti. Vermeulen [97] je pokazal, da lahko večino relacij ovrednotimo učinkoviteje, kot če bi jih vrednotili neposredno po formulah. V naši implementaciji Rekers-Schürrovega sintaksnega analizatorja smo Vermeulenova priporočila v celoti upoštevali.

V delu (c) na sliki 3.5 so podane vse resnične vrednosti relacij **conseqOf**, **conseqOf**$^+$, **above**, **excludes** in **excludes*** za množico produkcijskih primerkov, ki jih ustvari faza 1, prikazana v delu (a) iste slike. Noben produkcijski primerek ni nekonsistenten.

3.3.4 Faza 2

Ker se vse naše izboljšave nanašajo na fazo 1, bomo drugo fazo sintaksnega analizatorja zgolj na kratko opisali. Faza 2 na vhodu sprejme množico produkcijskih primerkov in vrednosti relacij med njimi, na izhodu pa bodisi vrne izpeljavo grafa G v gramatiki GG (če velja $G \in L(GG)$) ali pa sporoči, da graf ne pripada gramatiki.

Faza 2 poskuša v množici produkcijskih primerkov, ki jih zgradi faza 1, poiskati zaporedje primerkov, ki tvori izpeljavo grafa G. Izpeljavo gradi po posameznih korakih od začetka proti koncu, vendar pa lahko tudi sestopa. Ker se vsaka izpeljava prične z uporabo neke začetne produkcije, faza 2 na začetku izbere in uporabi enega od primerkov takšnih produkcij. Nato v vsakem koraku glede na množico že uporabljenih produkcijskih primerkov zgradi množico kandidatnih primerkov. Ta množica vsebuje vsak primerek pi, ki izpolnjuje sledeče pogoje:

- Primerek pi ni v relaciji **excludes*** z nobenim od že uporabljenih produkcijskih primerkov.
- Med uporabljenimi produkcijskimi primerki obstaja vsaj en primerek pi', tako da velja pi' **above** pi.
- Med primerki, ki jih je v izpeljavi potencialno še mogoče uporabiti, ni nobenega primerka pi'' z lastnostjo pi'' **above** pi.

Drugi in tretji pogoj zagotavljata, da se produkcijski primerki obravnavajo po njihovem vrstnem redu glede na relacijo **above**.

Če množica kandidatnih produkcijskih primerkov vsebuje primerek pi, ki se ne izključuje (**excludes***) z nobenim od še ne uporabljenih primerkov, potem faza 2 primerek pi izbere in uporabi na tekočem grafu izpeljave. Če pa se vsak kandidatni primerek izključuje z vsaj enim še ne uporabljenim primerkom, potem faza 2 izbere enega od kandidatnih primerkov in ustvari razvejišče: v prvi veji izbrani primerek uporabi, v drugi pa ga izloči iz nadaljnje obravnave. Faza 2 najprej izvrši prvo vejo, če pa na poti po njej zaide v slepo ulico, sestopi do razvejišča in izbere drugo vejo.

Če je množica kandidatnih produkcijskih primerkov prazna, imamo dve možnosti: (1) izpeljava je končana (to se zgodi v primeru, če je tekoči graf izpeljave istoveten z grafom G) ali (2) izpeljave ni mogoče nadaljevati. V primeru (2) je faza 2 zašla v slepo ulico, zato mora sestopiti do zadnjega razvejišča.

3.4 Izvirne izboljšave

V tem podpoglavju bomo predstavili izvirne izboljšave Rekers-Schürrovega sintaksnega analizatorja. Prva izboljšava (razdelek 3.4.1) nekoliko poveča razred grafnih gramatik, ki jih sprejema sintaksni analizator, ostale štiri izboljšave (razdelki 3.4.2–3.4.5) pa so namenjene povečanju računske učinkovitosti. Ker v nadaljevanju tega razdelka pogosto posegamo v kompleksno drobovje sintaksnega analizatorja, je smiselno podati osnovne zamisli posameznih izboljšav:

- Izboljšava 1 odpravlja potrebo po povezanih desnih straneh produkcij vhodne gramatike.

- Izboljšava 2 izkorišča simetrijo v produkcijah in na ta način doseže potencialno ogromne časovno-prostorske prihranke.

- Izboljšavi 3 in 4 poskušata zmanjšati potrebo po vrednotenju računsko zahtevne relacije **inconsistent**. V okviru izboljšave 3 poskušamo ugotoviti, katere produkcije lahko tvorijo nekonsistentne primerke, v okviru izboljšave 4 pa poskušamo (bodoče) nekonsistentne produkcijske primerke odkriti še pred njihovim nastankom.

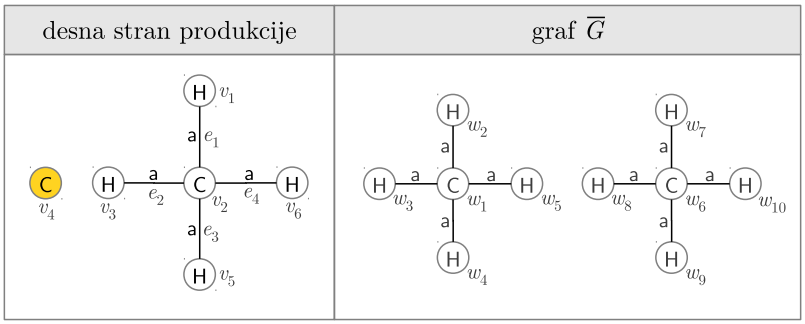
- Izboljšava 5 poskuša predčasno odkriti izpeljavo vhodnega grafa. To stori tako, da špekulativno izvede fazo 2, brž ko so izpolnjeni določeni pogoji, potrebni za obstoj izpeljave.

3.4.1 Izboljšava 1: Dopuščanje produkcij z nepovezanimi desnimi stranmi

Izvirni Rekers-Schürrov sintaksni analizator predpostavlja, da so desne strani vseh produkcij povezani grafi, zato ne deluje za gramatike, kot je GG_{HC} (slika 3.1). Ta zahteva je povezana z iskalnimi načrti, ki jih pri iskanju r-slik produkcij uporablja faza 1. Vsako nezačetno navodilo iskalnega načrta naroči fazi 1, naj bodisi poišče neko vozlišče s sledenjem določeni povezavi iz že odkritega vozlišča ali pa preveri obstoj določene povezave med dvema že odkritima vozliščema. Iskalni načrt, ki je sestavljen zgolj iz tovrstnih navodil, ne more opisati postopka obiskovanja elementov nepovezanega grafa.

Da bi sintaksni analizator lahko sprejemal gramatike z nepovezanimi desnimi stranmi, smo uvedli nov tip navodil iskalnega načrta — t.i. *skočna* navodila. Skočno navodilo zavzema obliko $\langle jump(v\colon A)\rangle$, kar pomeni »skoči na vozlišče z oznako A in mu dodeli enolični identifikator v«. Uporabimo ga lahko kjerkoli v iskalnem načrtu. Če faza 1 pri izvrševanju iskalnega načrta na grafu $\overline{G}$ naleti na skočno navodilo, poskuša poiskati vozlišče s predpisano oznako, ki ga v trenutnem procesu izvrševanja iskalnega načrta še ni odkrila. Drugih kriterijev za izbiro vozlišča ni.

Slika 3.6 prikazuje desno stran produkcije p_2 gramatike GG_{HC} in graf $\overline{G}$, v katerem želimo poiskati r-sliko te produkcije. Tabela 3.2 predstavlja enega od številnih možnih iskalnih načrtov za omenjeno desno stran. Oglejmo si, kako bi se prikazani iskalni načrt izvajal na prikazanem grafu $\overline{G}$. Prvemu navodilu iskalnega načrta ustreza katerokoli vozlišče z oznako H v grafu $\overline{G}$. Če iskalni postopek vozlišču v_1 v navodilu iskalnega načrta priredi vozlišče w_2, w_3, w_4 ali w_5 v grafu $\overline{G}$, potem mora v koraku 2 vozlišču v_2 prirediti vozlišče w_1. Če pa vozlišču v_1 priredi vozlišče w_7, w_8, w_9 ali w_{10}, potem mora vozlišču v_2 prirediti vozlišče w_6. V prvem primeru lahko postopek v koraku 4 vozlišču v_4 priredi zgolj vozlišče w_6, v drugem primeru pa zgolj vozlišče w_1, saj je vozlišče w_1 v prvem oz. w_6 v drugem primeru že prirejeno vozlišču v_1 v iskalnem načrtu. Ker faza 1 izvrši iskalni načrt na vse možne načine, bo polovica odkritih r-slik v grafu $\overline{G}$ pokrivala vozlišča w_1, w_2, w_3, w_4, w_5 in w_6, druga polovica pa vozlišča w_1, w_6, w_7, w_8, w_9 in w_{10}.



Slika 3.6: *Desna stran produkcije p_2 gramatike GG_{HC} in primer grafa $\overline{G}$*

Tabela 3.2: *Primer iskalnega načrta za desno stran produkcije p_2 gramatike GG_{HC}.*

korak	navodilo
1	$\langle head(v_1\colon \mathsf{H})\rangle$
2	$\langle e_1\colon v_1 \overset{\mathsf{a}}{\leftarrow} (v_2\colon \mathsf{C})\rangle$
3	$\langle e_2\colon v_2 \overset{\mathsf{a}}{\leftarrow} (v_3\colon \mathsf{H})\rangle$
4	$\langle jump(v_4\colon \mathsf{C})\rangle$
5	$\langle e_3\colon v_2 \overset{\mathsf{a}}{\leftarrow} (v_5\colon \mathsf{H})\rangle$
6	$\langle e_4\colon v_2 \overset{\mathsf{a}}{\leftarrow} (v_6\colon \mathsf{H})\rangle$

3.4.2 Izboljšava 2: Preprečevanje nepotrebnih večkratnih odkritij istih r-slik

Ta razdelek bo obsežnejši, zato ga bomo razbili na več podrazdelkov. V podrazdelku 3.4.2.1 bomo problem večkratnih odkritij istih r-slik predstavili s pomočjo primera. V podrazdelku 3.4.2.2 bomo uvedli pojem *medsebojne zamenljivosti* elementov produkcij, ki je tesno povezan z navedenim problemom. V podrazdelku 3.4.2.3 bomo pokazali, kako lahko medsebojno zamenljivost uporabimo za odpravo problema večkratnih odkritij, v podrazdelku 3.4.2.4 pa bomo predstavili algoritem za iskanje medsebojno zamenljivih elementov. V podrazdelku 3.4.2.5 bomo odgovorili na vprašanje, kdaj se izboljšava 2 izplača.

3.4.2.1 Opis problema s primerom

Spomnimo se, da faza 1 za vsako produkcijo vhodne gramatike poišče vse njene r-slike v grafu $\overline{G}$. Vsaka r-slika je določena s p-homomorfizmom med desno stranjo produkcije in grafom $\overline{G}$. Elementi $Xrhs[p]$ morajo biti na pripadajoče elemente r-slike preslikani injektivno, torej z monomorfizmom, za elemente $Common[p]$ pa injektivnost ni zahtevana.

Vsak p-homomorfizem med elementi $Rhs[p]$ in grafom $\overline{G}$ določa neko r-sliko produkcije p. Vendar pa se lahko zgodi, da je *ista* r-slika (isti podgraf grafa $\overline{G}$) določena z več različnimi p-homomorfizmi med desno stranjo produkcije p in grafom $\overline{G}$. V

takih primerih bo faza 1 isto r-sliko produkcije p odkrila večkrat.

Večkratno odkritje istih r-slik produkcij je sicer v nekaterih primerih nujno za pravilno delovanje sintaksnega analizatorja, v mnogih pa predstavlja zgolj odvečno delo. Izboljšava sintaksnega analizatorja, ki jo bomo opisali v nadaljevanju, se ukvarja s preprečevanjem tovrstne redundance. Opisani problem je odkril Vermeulen [97], vendar ga ni formalno opisal, kot rešitev pa je predlagal ročne posege v gramatiko. Naša rešitev pa je povsem avtomatizirana in temelji na formalnih konceptih preslikav med grafi.

Ilustrirajmo problem večkratnih odkritij s primerom. Slika 3.7 prikazuje dve produkciji in njuni r-sliki v nekem grafu $\overline{G}$. Ker velja $Xrhs[p_1] = Rhs[p_1]$ in $Xrhs[p_2] = Rhs[p_2]$, morajo biti vsi elementi desnih strani obeh produkcij injektivno preslikani na elemente grafa $\overline{G}$. Tabela 3.3 prikazuje vse p-homomorfizme med desnima stranema produkcij in njunima r-slikama.[1] V prikazanem primeru bi faza 1 r-sliko produkcije p_1 odkrila štirikrat, r-sliko produkcije p_2 pa šestkrat. Kot bomo videli kasneje, bi pri produkcijah p_1 in p_2 zadoščalo, če bi faza 1 sintaksnega analizatorja vsako njuno r-sliko v grafu $\overline{G}$ odkrila samo enkrat, in to ne glede na graf $\overline{G}$. Pri iskanju r-slik produkcij, ki vsebujejo nekakšno (zaenkrat še neznano) simetrijo, so torej možni veliki prihranki. Naša naslednja naloga bo torej identificirati takšno simetrijo in jo izkoristiti za izboljšanje učinkovitosti sintaksnega analizatorja.

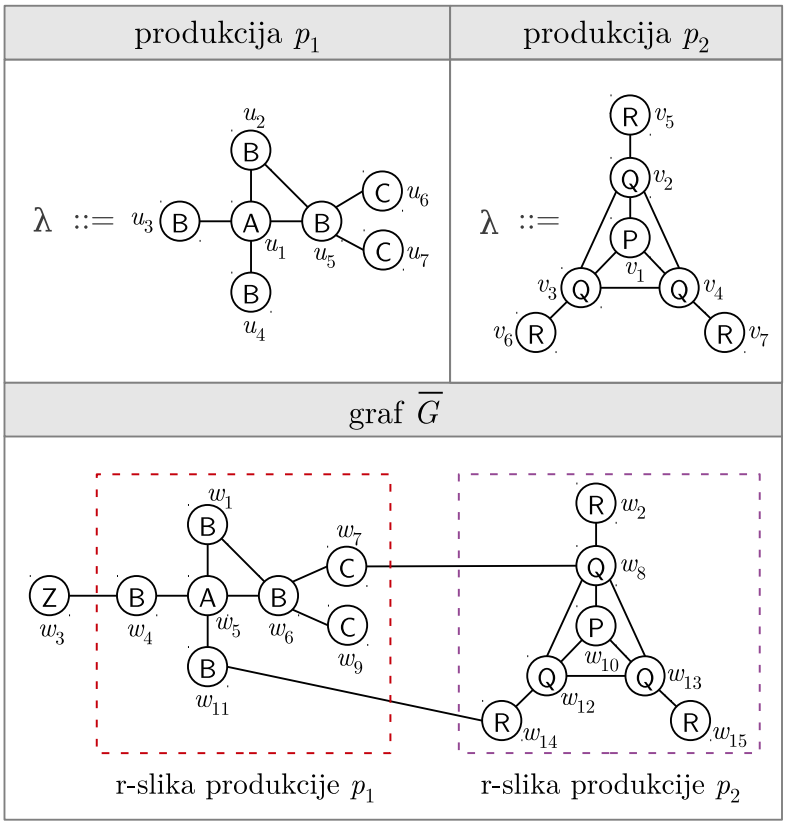
Tabela 3.3: *Vsi p-homomorfizmi* $Rhs[p_1] \to \overline{G}$ *in* $Rhs[p_2] \to \overline{G}$ *za primer produkcij* p_1 *in* p_2 *ter grafa* $\overline{G}$ *s slike 3.7.*

p	p-homomorfizmi $Rhs[p] \to \overline{G}$
p_1	$\{u_1 \mapsto w_5,\ u_2 \mapsto w_1,\ u_3 \mapsto w_4,\ u_4 \mapsto w_{11},\ u_5 \mapsto w_6,\ u_6 \mapsto w_7,\ u_7 \mapsto w_9\}$,
	$\{u_1 \mapsto w_5,\ u_2 \mapsto w_1,\ u_3 \mapsto w_4,\ u_4 \mapsto w_{11},\ u_5 \mapsto w_6,\ u_6 \mapsto w_9,\ u_7 \mapsto w_7\}$,
	$\{u_1 \mapsto w_5,\ u_2 \mapsto w_1,\ u_3 \mapsto w_{11},\ u_4 \mapsto w_4,\ u_5 \mapsto w_6,\ u_6 \mapsto w_7,\ u_7 \mapsto w_9\}$,
	$\{u_1 \mapsto w_5,\ u_2 \mapsto w_1,\ u_3 \mapsto w_{11},\ u_4 \mapsto w_4,\ u_5 \mapsto w_6,\ u_6 \mapsto w_9,\ u_7 \mapsto w_7\}$.
p_2	$\{v_1 \mapsto w_{10},\ v_2 \mapsto w_8,\ v_3 \mapsto w_{12},\ v_4 \mapsto w_{13},\ v_5 \mapsto w_2,\ v_6 \mapsto w_{14},\ v_7 \mapsto w_{15}\}$,
	$\{v_1 \mapsto w_{10},\ v_2 \mapsto w_8,\ v_3 \mapsto w_{13},\ v_4 \mapsto w_{12},\ v_5 \mapsto w_2,\ v_6 \mapsto w_{15},\ v_7 \mapsto w_{14}\}$,
	$\{v_1 \mapsto w_{10},\ v_2 \mapsto w_{12},\ v_3 \mapsto w_8,\ v_4 \mapsto w_{13},\ v_5 \mapsto w_{14},\ v_6 \mapsto w_2,\ v_7 \mapsto w_{15}\}$,
	$\{v_1 \mapsto w_{10},\ v_2 \mapsto w_{12},\ v_3 \mapsto w_{13},\ v_4 \mapsto w_8,\ v_5 \mapsto w_{14},\ v_6 \mapsto w_{15},\ v_7 \mapsto w_2\}$,
	$\{v_1 \mapsto w_{10},\ v_2 \mapsto w_{13},\ v_3 \mapsto w_8,\ v_4 \mapsto w_{12},\ v_5 \mapsto w_{15},\ v_6 \mapsto w_2,\ v_7 \mapsto w_{14}\}$,
	$\{v_1 \mapsto w_{10},\ v_2 \mapsto w_{13},\ v_3 \mapsto w_{12},\ v_4 \mapsto w_8,\ v_5 \mapsto w_{15},\ v_6 \mapsto w_{14},\ v_7 \mapsto w_2\}$.

3.4.2.2 Avtomorfizmi in medsebojno zamenljivi elementi v desnih straneh

Večkratni p-homomorfizmi med desnimi stranmi in njihovimi r-slikami, ki so vzrok za večkratna odkritja istih r-slik, so posledica *avtomorfizmov* desnih strani (definicija

[1] Zaradi preglednosti smo v tabeli prikazali le tiste dele posameznih p-homomorfizmov, ki se nanašajo na vozlišča. Tako bomo ravnali tudi v nadaljevanju tega razdelka. Seveda pa (popoln) p-homomorfizem vsebuje podatke o preslikavah vseh elementov, torej tako vozlišč kot povezav.



Slika 3.7: *Produkciji in njuni r-sliki v grafu* $\overline{G}$.

2.14). V primeru s slike 3.7 in iz tabele 3.3, denimo, so štirje p-homomorfizmi med desno stranjo produkcije p_1 in *isto* r-sliko produkcije p_1 v grafu $\overline{G}$ posledica sledečih štirih avtomorfizmov v grafu $Rhs[p_1]$ (prim. s tabelo 3.3):

$$\begin{array}{l}
\{u_1 \mapsto u_1,\ u_2 \mapsto u_2,\ u_3 \mapsto u_3,\ u_4 \mapsto u_4,\ u_5 \mapsto u_5,\ u_6 \mapsto u_6,\ u_7 \mapsto u_7\},\\
\{u_1 \mapsto u_1,\ u_2 \mapsto u_2,\ u_3 \mapsto u_3,\ u_4 \mapsto u_4,\ u_5 \mapsto u_5,\ u_6 \mapsto u_7,\ u_7 \mapsto u_6\},\\
\{u_1 \mapsto u_1,\ u_2 \mapsto u_2,\ u_3 \mapsto u_4,\ u_4 \mapsto u_3,\ u_5 \mapsto u_5,\ u_6 \mapsto u_6,\ u_7 \mapsto u_7\},\\
\{u_1 \mapsto u_1,\ u_2 \mapsto u_2,\ u_3 \mapsto u_4,\ u_4 \mapsto u_3,\ u_5 \mapsto u_5,\ u_6 \mapsto u_7,\ u_7 \mapsto u_6\}.
\end{array}$$

Šesterica p-homomorfizmov $Rhs[p_2] \to \overline{G}$ je posledica sledečih avtomorfizmov v grafu $Rhs[p_2]$:

$$\begin{array}{l}
\{v_1 \mapsto v_1,\ v_2 \mapsto v_2,\ v_3 \mapsto v_3,\ v_4 \mapsto v_4,\ v_5 \mapsto v_5,\ v_6 \mapsto v_6,\ v_7 \mapsto v_7\},\\
\{v_1 \mapsto v_1,\ v_2 \mapsto v_2,\ v_3 \mapsto v_4,\ v_4 \mapsto v_3,\ v_5 \mapsto v_5,\ v_6 \mapsto v_7,\ v_7 \mapsto v_6\},\\
\{v_1 \mapsto v_1,\ v_2 \mapsto v_3,\ v_3 \mapsto v_2,\ v_4 \mapsto v_4,\ v_5 \mapsto v_6,\ v_6 \mapsto v_5,\ v_7 \mapsto v_7\},\\
\{v_1 \mapsto v_1,\ v_2 \mapsto v_3,\ v_3 \mapsto v_4,\ v_4 \mapsto v_2,\ v_5 \mapsto v_6,\ v_6 \mapsto v_7,\ v_7 \mapsto v_5\},\\
\{v_1 \mapsto v_1,\ v_2 \mapsto v_4,\ v_3 \mapsto v_2,\ v_4 \mapsto v_3,\ v_5 \mapsto v_7,\ v_6 \mapsto v_5,\ v_7 \mapsto v_6\},\\
\{v_1 \mapsto v_1,\ v_2 \mapsto v_4,\ v_3 \mapsto v_3,\ v_4 \mapsto v_2,\ v_5 \mapsto v_7,\ v_6 \mapsto v_6,\ v_7 \mapsto v_5\}.
\end{array}$$

Sedaj bomo definirali pojem *medsebojne zamenljivosti* (angl. *interchangeability*), ki je tesno povezan s pojmom avtomorfizma. Sledečih definicij ne bomo takoj utemeljili. Njihov pomen se bo razkril šele v podrazdelku 3.4.2.3, kjer bomo opisali,

kako lahko s pomočjo poznavanja medsebojne zamenljivosti elementov produkcij preprečujemo nastanek nepotrebnih večkratnih odkritij istih r-slik.

Definicija 3.12 (medsebojna zamenljivost na ravni grafa)**.** Vozlišča $v_1, \ldots, v_k$ v grafu G so *medsebojno zamenljiva glede na graf* G (oznaka: $\mathbf{inter}[G](v_1, \ldots, v_k)$) natanko v primeru, če za vsako permutacijo σ množice $1\,..\,k$ obstaja avtomorfizem $h\colon G \to G$, tako da velja $h(v_1) = v_{\sigma(1)}, \ldots, h(v_k) = v_{\sigma(k)}$.

Definicija 3.13 (zamenljivostni razred grafa)**.** *Zamenljivostni razred* (angl. *interchangeability class*) grafa G je množica vozlišč, ki so medsebojno zamenljiva glede na graf G.

Na primer, desni strani produkcij p_1 in p_2 na sliki 3.7 vsebujeta po dva zamenljivostna razreda:

- $Rhs[p_1]$: $\{3, 4\}$ in $\{6, 7\}$.
- $Rhs[p_2]$: $\{2, 3, 4\}$ in $\{5, 6, 7\}$.

Definicija 3.14 (medsebojna neodvisnost zamenljivostnih razredov)**.** Naj v grafu G obstaja s zamenljivostnih razredov, pri čemer je i-ti zamenljivostni razred (za $i \in 1\,..\,s$) sestavljen iz k_i vozlišč. Navedeni zamenljivostni razredi so *medsebojno neodvisni* natanko v primeru, če graf G vsebuje avtomorfizem za vsako od $k_1!\,k_2!\,\ldots\,k_s!$ permutacij vozlišč, ki tvorijo zamenljivostne razrede.

Zamenljivostna razreda v grafu $Rhs[p_1]$ sta medsebojno neodvisna, ker v grafu $Rhs[p_1]$ obstaja avtomorfizem za vsako od štirih ($= 2!\,2!$) permutacij vozlišč $\{u_3, u_4\}$ in $\{u_6, u_7\}$. Te permutacije so (u_3, u_4, u_6, u_7), (u_3, u_4, u_7, u_6), (u_4, u_3, u_6, u_7) in (u_4, u_3, u_7, u_6). Zamenljivostna razreda grafa $Rhs[p_2]$ pa nista medsebojno neodvisna, saj vsaka permutacija vozlišč $\{v_2, v_3, v_4\}$ ustreza enolično določeni permutaciji $\{v_5, v_6, v_7\}$, in obratno. Z drugimi besedami: množice vozlišč $\{v_2, v_3, v_4\}$ ni mogoče permutirati neodvisno od množice $\{v_5, v_6, v_7\}$, saj vsak avtomorfizem, ki permutira množico $\{v_2, v_3, v_4\}$, na povsem enak način permutira tudi množico $\{v_5, v_6, v_7\}$. Graf $Rhs[p_2]$ potemtakem vsebuje zgolj en neodvisni zamenljivostni razred: bodisi $\{v_2, v_3, v_4\}$ ali $\{v_5, v_6, v_7\}$.

Skupina medsebojno zamenljivih vozlišč na ravni desne strani produkcije je sicer zagotovilo za večkratna odkritja istih r-slik, vendar pa ni nujno, da so ta odkritja tudi dejansko nepotrebna; lahko bi se namreč zgodilo, da po njihovi preprečitvi sintaksni analizator ne bi več deloval pravilno. Kot bomo videli kasneje, je pogoj za varno preprečitev večkratnih odkritij medsebojna zamenljivost na ravni celotne produkcije.

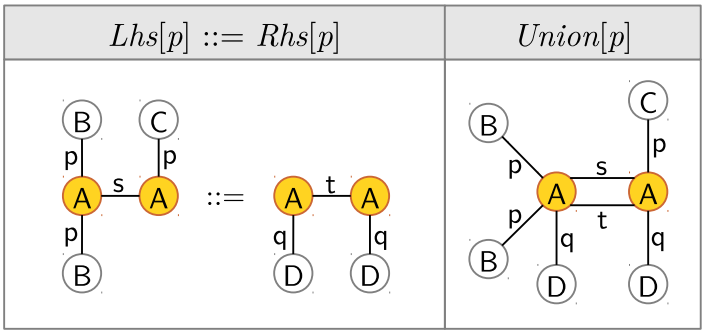
Definicija 3.15 (medsebojna zamenljivost na ravni produkcije)**.** Vozlišča $v_1, \ldots, v_k \in Rhs[p]$ so *medsebojno zamenljiva glede na produkcijo* p (oznaka: $\mathbf{inter}[p](v_1, \ldots, v_k)$) natanko tedaj, ko so izpolnjeni sledeči trije pogoji:

1. Velja bodisi $v_1, \ldots, v_k \in Xrhs[p]$ ali $v_1, \ldots, v_k \in Common[p]$.
2. Vozlišča $v_1, \ldots, v_k$ so medsebojno zamenljiva glede na graf $Rhs[p]$. Velja torej $\mathbf{inter}[Rhs[p]](v_1, \ldots, v_k)$.

3. Vozlišča $v_1, \ldots, v_k$ so medsebojno zamenljiva glede na graf $Union[p]$, torej **inter**$[Union[p]](v_1, \ldots, v_k)$.

Definicija 3.16 (zamenljivostni razred produkcije)**.** Zamenljivostni razred produkcije p je množica vozlišč grafa $Rhs[p]$, ki so medsebojno zamenljiva glede na produkcijo p.

V produkciji na sliki 3.8 sta vozlišči z oznako A medsebojno zamenljivi glede na desno stran produkcije, ne pa glede na celotno produkcijo (gl. graf *Union* za to produkcijo). Enako velja za vozlišči z oznako D. Zato v prikazanem primeru ne smemo preprečiti večkratnih odkritij istih r-slik.



Slika 3.8: *Produkcija in njen graf Union.*

Definicija 3.17 (medsebojna neodvisnost zamenljivostnih razredov na ravni produkcije)**.** Skupina zamenljivostnih razredov na ravni produkcije p je *medsebojno neodvisna* natanko tedaj, ko so zamenljivostni razredi v tej skupini medsebojno neodvisni v okviru grafa $Rhs[p]$.

V nadaljevanju bomo z besedno zvezo »zamenljivostni razred« označevali zamenljivostni razred na ravni produkcije. Če produkcija vsebuje s medsebojno neodvisnih zamenljivostnih razredov, ki vsebujejo po k_1, k_2, ... oziroma k_s vozlišč, bo faza 1 vsako r-sliko te produkcije odkrila $(k_1!\, k_2! \ldots k_s!)$-krat, saj vsak avtomorfizem desne strani produkcije določa po en p-homomorfizem med desno stranjo in vsako posamezno r-sliko produkcije v grafu $\overline{G}$. (Če smo natančni, je število odkritij lahko tudi manjše od $k_1!\, k_2! \ldots k_s!$, a le tedaj, ko zamenljivostni razred vsebuje vsaj dve vozlišči iz grafa $Common[p]$. V takšnih primerih se namreč lahko več vozlišč iz grafa $Common[p]$ preslika v isto vozlišče grafa $\overline{G}$.)

Pojem medsebojne zamenljivosti bomo definirali tudi za povezave:

Definicija 3.18 (medsebojna zamenljivost povezav)**.** Povezave $e_1, \ldots, e_k$ so *medsebojno zamenljive* glede na produkcijo p natanko tedaj, ko so izpolnjeni sledeči trije pogoji:

1. Velja bodisi $e_1, \ldots, e_k \in Xrhs[p]$ ali $e_1, \ldots, e_k \in Common[p]$.

2. Vse povezave so enako označene, tj. $label(e_1) = \ldots = label(e_k)$.

3. Vse povezave povezujejo isto dvojico vozlišč v isti smeri, tj. $source(e_1) = \ldots = source(e_k)$ in $target(e_1) = \ldots = target(e_k)$ (v usmerjenih grafih) oz. $conn(e_1) = \ldots = conn(e_k)$ (v neusmerjenih grafih).

Zamenljivostni razred za povezave je množica medsebojno zamenljivih povezav v dani produkciji.

Na primer, obe povezavi na desni strani produkcije p_6 v gramatiki GG_{AHC} (slika 3.1) sta medsebojno zamenljivi. Enako velja za vse tri povezave na desni strani produkcije p_7.

3.4.2.3 Preprečevanje nepotrebnih večkratnih odkritij istih r-slik produkcij

Za potrebe tega podrazdelka predpostavimo, da ima vsako vozlišče v grafu svoj enolično določen celoštevilski *indeks*. Indeks vozlišča v bomo označili z $index(v)$.

Kot smo videli, so večkratna odkritja istih r-slik povezana z zamenljivostnimi razredi. Sledeča trditev[2] pravi, da je grafnim elementom (vozliščem oz. povezavam) v zamenljivostnem razredu produkcije vedno mogoče na enoličen način prirediti grafne elemente v r-sliki produkcije:

Trditev 3.19. Denimo, da desna stran produkcije p vsebuje en sam zamenljivostni razred, ta pa sestoji iz elementov $x_1, \ldots, x_k \in Rhs[p]$, pri čemer velja $index(x_1) < \ldots < index(x_k)$. Potem med grafom $Rhs[p]$ in vsako r-sliko produkcije p v grafu $\overline{G}$ obstaja natanko en p-homomorfizem h z lastnostjo $index(h(x_1)) \leq \ldots \leq index(h(x_k))$.

Dokaz. Predpostavimo najprej, da velja $x_1, \ldots, x_k \in Xrhs[p]$. Naj za p-homomorfizem $h\colon Rhs[p] \to \overline{G}$ velja $h(x_1) = y_{i[1]}, \ldots, h(x_k) = y_{i[k]}$, pri čemer je zaporedje $(i[1], \ldots, i[k])$ neka permutacija množice $1\,..\,k$. Denimo, da velja $index(y_1) < \ldots < index(y_k)$. Ker so elementi $x_1, \ldots, x_k$ medsebojno zamenljivi, obstaja $k!$ avtomorfizmov $Rhs[p] \to Rhs[p]$, za natanko enega od njih (npr. g) pa velja $g(x_{i[1]}) = x_1, \ldots, g(x_{i[k]}) = x_k$. Definirajmo novo preslikavo kot $h' = h \circ g$. Ker sta grafa $Rhs[p]$ in $g(Rhs[p])$ po definiciji avtomorfizma izomorfna, je preslikava h' prav tako p-homomorfizem med grafoma $Rhs[p]$ in $\overline{G}$. Zanjo velja $h'(x_{i[1]}) = h(g(x_{i[1]})) = h(x_1) = y_{i[1]}, \ldots, h'(x_{i[k]}) = h(g(x_{i[k]})) = h(x_k) = y_{i[k]}$ oziroma $h'(x_i) = y_i$ za vsak $i \in 1\,..\,k$. Za p-homomorfizem torej velja $index(h'(x_1)) < \ldots < index(h'(x_k))$, poleg tega pa je edini s to lastnostjo, saj izvira iz unikatnega avtomorfizma g. Ker se zaradi predpostavke $x_1, \ldots, x_k \in Xrhs[p]$ vsi elementi x_i preslikajo v različne elemente y_i, je p-homomorfizem h' tudi edini, ki izpolnjuje pogoj $index(h'(x_1)) \leq \ldots \leq index(h'(x_k))$.

Če medsebojno zamenljivi elementi $x_1, \ldots, x_k$ pripadajo grafu $Common[p]$ (druga in zadnja možnost!), potem se lahko več različnih elementov preslika v isti element grafa $\overline{G}$. Vendar pa tudi v tem primeru med grafom $Rhs[p]$ in vsako r-sliko produkcije p obstaja en sam p-homomorfizem h z lastnostjo $index(h(x_1)) \leq \ldots \leq index(h(x_k))$, saj gre, denimo, pri p-homomorfizmih $\{u \mapsto w, v \mapsto w\}$ in $\{v \mapsto w, u \mapsto w\}$ dejansko za eno in isto preslikavo. □

[2]Za označevanje glavnih trditev (izrekov) bomo uporabljali izraz »trditev«, za označevanje pomožnih trditev pa izraz »lema«.

Trditev 3.19 lahko neposredno izkoristimo za preprečevanje nepotrebnih večkratnih odkritij istih r-slik. Naj produkcija p zadošča pogoju iz trditve; njen edini zamenljivostni razred naj bo sestavljen iz elementov $x_1, \ldots, x_k \in Xrhs[p]$ z lastnostjo $index(x_1) < \ldots < index(x_k)$. Faza 1 v izvirnem Rekers-Schürrovem sintaksnem analizatorju izčrpno preišče prostor možnih delnih in popolnih p-homomorfizmov med grafom $Rhs[p]$ in grafom $\overline{G}$ in tako vzpostavi popoln p-homomorfizem $Rhs[p] \to \overline{G}$ za vsako permutacijo elementov $x_1, \ldots, x_k$, posledica česar je $k!$ odkritij vsake r-slike produkcije p. Opisano redundanco lahko odpravimo tako, da v fazi 1 vzpostavimo samo tiste popolne p-homomorfizme $h\colon Rhs[p] \to \overline{G}$, za katere velja $index(h(x_1)) \leq \ldots \leq index(h(x_k))$, pri tem pa isti pogoj upoštevamo tudi za vse njihove delne pod-p-homomorfizme. Ker po trditvi 3.19 med desno stranjo produkcije p in vsako njeno r-sliko v grafu $\overline{G}$ obstaja *natanko en* popoln p-homomorfizem h, ki izpolnjuje pogoj $index(h(x_1)) \leq \ldots \leq index(h(x_k))$, bo faza 1 vsako r-sliko produkcije p odkrila natanko enkrat.

Opisano shemo brez težav posplošimo na več medsebojno neodvisnih zamenljivostnih razredov:

Trditev 3.20. Denimo, da desna stran produkcije p vsebuje s medsebojno neodvisnih zamenljivostnih razredov in da i-ti ($i \in 1\,..\,s$) zamenljivostni razred vsebuje elemente $x_1^{(i)}, \ldots, x_{k[i]}^{(i)}$ z lastnostjo $index(x_1^{(i)}) < \ldots < index(x_{k[i]}^{(i)})$. Če pri iskanju p-homomorfizmov $h\colon Rhs[p] \to \overline{G}$ v fazi 1 upoštevamo pogoje $index(h(x_1^{(i)})) \leq \ldots \leq index(h(x_{k[i]}^{(i)}))$ za vse $i \in 1\,..\,s$, potem bo faza 1 vsako r-sliko produkcije odkrila natanko enkrat.

Dokaz. Ker so po predpostavki iz trditve zamenljivostni razredi med seboj neodvisni, lahko v grafu $Rhs[p]$ elemente vsakega zamenljivostnega razreda med seboj poljubno permutiramo (neodvisno od elementov ostalih $s-1$ zamenljivostnih razredov), pa bo nastali graf vedno izomorfen grafu $Rhs[p]$. Podobno kot pri dokazu trditve 3.19 lahko ugotovimo, da obstaja natanko en p-homomorfizem, ki izpolnjuje pogoje iz trditve, saj obstaja natanko ena permutacija grafnih elementov iz zamenljivostnih razredov, ki določa takšen p-homomorfizem. Ker faza 1 prečeše vse delne in popolne p-homomorfizme med desno stranjo produkcije in vsako njeno r-sliko, bo iskani p-homomorfizem zanesljivo odkrila, in to zgolj enkrat. □

Prikažimo to zamisel na primeru s slike 3.7. Osredotočimo se na produkcijo p_2. Predpostavimo, da v produkciji velja $index(v_i) = i$, v grafu pa $index(w_i) = i$ za vse indekse i. Produkcija p_2 vsebuje en sam *neodvisen* zamenljivostni razred — bodisi $\{v_2, v_3, v_4\}$ ali $\{v_5, v_6, v_7\}$. Razreda sta si povsem enakovredna, zato izberimo prvega. Denimo, da faza 1 pravkar išče r-slike produkcije p_2 v grafu $\overline{G}$ in da je že vzpostavila delni p-homomorfizem $h\colon \{v_1 \mapsto w_{10},\ v_2 \mapsto w_{12},\ v_3 \mapsto w_{13}\}$. Ker velja $index(h(v_2)) < index(h(v_3))$, faza 1 delnega p-homomorfizma h ne bo zavrgla. Vendar pa p-homomorfizma h ni mogoče dopolniti do popolnega p-homomorfizma, saj lahko vozlišču v_4 v grafu $Rhs[p_2]$ priredimo le še vozlišče w_8 v grafu $\overline{G}$, nakar dobimo prepovedan p-homomorfizem $\{v_1 \mapsto w_{10},\ v_2 \mapsto w_{12},\ v_3 \mapsto w_{13},\ v_4 \mapsto w_8\}$. Zato bo faza 1 poskušala poiskati drugačen p-homomorfizem med elementi grafa $Rhs[p_2]$ in r-sliko produkcije p_2 v grafu $\overline{G}$. Do konca svojega delovanja bo natanko

enkrat odkrila edini p-homomorfizem, ki izpolnjuje pogoj iz trditve 3.20. To je p-homomorfizem $\{v_1 \mapsto w_{10},\ v_2 \mapsto w_8,\ v_3 \mapsto w_{12},\ v_4 \mapsto w_{13},\ v_5 \mapsto w_2,\ v_6 \mapsto w_{14},\ v_7 \mapsto w_{15}\}$. Zato bo faza 1 r-sliko produkcije p_2 v grafu $\overline{G}$ odkrila natanko enkrat.

V predstavljenem primeru pogoj iz trditve 3.20 velja tudi za zamenljivostni razred $\{v_5, v_6, v_7\}$. Vendar pa tega ne bi mogli zagotoviti, če bi, denimo, v grafu $\overline{G}$ medsebojno zamenjali indeksa vozlišč w_{14} in w_{15}. Ker pa zamenljivostna razreda nista medsebojno neodvisna, zadošča, če pogoj iz trditve 3.20 upoštevamo le za enega od njiju. V nasprotju s produkcijo p_2 pa sta pri produkciji p_1 s slike 3.7 zamenljivostna razreda med seboj neodvisna, zato bi morali pogoj iz trditve 3.20 zagotoviti za oba razreda.

Predstavili smo metodo za preprečevanje nepotrebnih večkratnih odkritij istih r-slik, sedaj pa moramo še pokazati, da predstavljena metoda ohranja pravilnost delovanja sintaksnega analizatorja. Sledeča lema in trditev nista bistveni za razumevanje izboljšave 2, zato lahko bralec takoj preide na podrazdelek 3.4.2.4.

Lema 3.21. Podan je graf G. Naj $v_0 \colon b_0 \xrightarrow{e_1 \colon a_1} v_1 \colon b_1 \xrightarrow{e_2 \colon a_2} \ldots$ označuje sprehod, ki se prične v vozlišču $v_0 \in G$ z oznako b_0, nato pa po vrsti obišče povezavo e_1 z oznako a_1, vozlišče v_1 z oznako b_1, povezavo e_2 z oznako a_2 itd. Denimo, da sta vozlišči $v_0 \in G$ in $w_0 \in G$ medsebojno zamenljivi glede na graf G. Potem velja sledeče: Če obstaja sprehod $v_0 \colon b_0 \xrightarrow{e_1 \colon a_1} v_1 \colon b_1 \xrightarrow{e_2 \colon a_2} \ldots$, potem sprehod $w_0 \colon b_0 \xrightarrow{f_1 \colon a_1} w_1 \colon b_1 \xrightarrow{f_2 \colon a_2} \ldots$ prav tako obstaja.

Dokaz. Če sta vozlišči v_0 in w_0 medsebojno zamenljivi, potem po definiciji 3.12 v grafu G obstaja avtomorfizem h, tako da velja $h(v_0) = w_0$ in $h(w_0) = v_0$. Predpostavimo, da je vozlišče v_0 preko povezave e_1 z oznako a_1 povezano z vozliščem v_1 z oznako b_1. Avtomorfizem h že po definiciji homomorfizma ohranja sosednosti in oznake, zato povezavo e_1 in vozlišče v_1 preslika v povezavo f_1 z oznako a_1 in v vozlišče w_1 z oznako b_1, tako da je vozlišče w_1 sosed vozlišča w_0 preko povezave f_1. Takšno vozlišče w_1 in takšna povezava f_1 morata obstajati, saj sicer preslikava h ne bi bila avtomorfizem. Avtomorfizem h potemtakem preslika sprehod $v_0 \colon b_0 \xrightarrow{e_1 \colon a_1} v_1 \colon b_1$ v sprehod $w_0 \colon b_0 \xrightarrow{f_1 \colon a_1} w_1 \colon b_1$. Z enakim razmislekom kot pri dosedanjem sklepanju lahko ugotovimo, da avtomorfizem h vozlišče v_2, ki je preko povezave e_2 povezano z vozliščem v_1, preslika v vozlišče w_2, ki je preko povezave f_2 povezano z vozliščem w_1, tako da velja $label(w_2) = label(v_2)$ in $label(f_2) = label(e_2)$. Sprehod $v_0 \colon b_0 \xrightarrow{e_1 \colon a_1} v_1 \colon b_1 \xrightarrow{e_2 \colon a_2} v_2 \colon b_2$ se tako preslika v sprehod $w_0 \colon b_0 \xrightarrow{f_1 \colon a_1} w_1 \colon b_1 \xrightarrow{f_2 \colon a_2} w_2 \colon b_2$. Na opisani način lahko oba sprehoda poljubno podaljšujemo. □

Trditev 3.22. Predlagana shema za preprečevanje nepotrebnih večkratnih odkrivanj istih r-slik ohranja pravilnost delovanja sintaksnega analizatorja.

Dokaz. Pokazati moramo, da predlagana shema zgolj odpravi odvečno delo in da nima nikakršnega učinka na sintaksni analizator z izjemo (potencialno zelo velike) pohitritve. Osredotočimo se na produkcijo p v vhodni gramatiki. Predpostavimo, da produkcija p vsebuje en sam zamenljivostni razred, ta pa sestoji iz k vozlišč iz grafa $Rhs(p)$. Dokaz je možno premočrtno prilagoditi za primer, ko produkcija p vsebuje več medsebojno neodvisnih zamenljivostnih razredov, saj lahko zamenljivostne razrede obravnavamo neodvisno drug od drugega.

Izberimo dvoje vozlišč iz zamenljivostnega razreda produkcije p, denimo v_0 in w_0. Naj bo $v_0 \colon b_0 \xrightarrow{e_1 \colon a_1} v_1 \colon b_1 \ldots \xrightarrow{e_r \colon a_r} v_r \colon b_r \xrightarrow{e_{r+1} \colon a_{r+1}} v_{r+1} \colon b_{r+1} \ldots \xrightarrow{e_s \colon a_s} v_s \colon b_s$ sprehod z lastnostma $v_0, e_1, v_1, \ldots, e_r, v_r \in \mathit{Rhs}[p]$ in $e_{r+1}, v_{r+1}, \ldots, e_s, v_s \in \mathit{Union}[p] \setminus \mathit{Rhs}[p]$ $(= \mathit{Xlhs}[p])$. Ker sta po definiciji 3.15 vozlišči v_0 in w_0 medsebojno zamenljivi tako v grafu $\mathit{Rhs}[p]$ kot v grafu $\mathit{Union}[p]$, mora po lemi 3.21 obstajati tudi sprehod $w_0 \colon b_0 \xrightarrow{f_1 \colon a_1} w_1 \colon b_1 \ldots \xrightarrow{f_r \colon a_r} w_r \colon b_r \xrightarrow{f_{r+1} \colon a_{r+1}} w_{r+1} \colon b_{r+1} \ldots \xrightarrow{f_s \colon a_s} w_s \colon b_s$, tako da velja $w_0, f_1, w_1, \ldots, f_r, w_r \in \mathit{Rhs}[p]$ in $f_{r+1}, w_{r+1}, \ldots, f_s, w_s \in \mathit{Xlhs}[p]$.

Ko faza 1 odkrije p-homomorfizem $h \colon \mathit{Rhs}[p] \to \overline{G}$ in s tem r-sliko produkcije p v grafu $\overline{G}$, preslika sprehod $v_0 \colon b_0 \xrightarrow{e_1 \colon a_1} v_1 \colon b_1 \ldots \xrightarrow{e_r \colon a_r} v_r \colon b_r$ v grafu $\mathit{Rhs}[p]$ v pripadajoči sprehod $v'_0 \colon b_0 \xrightarrow{e'_1 \colon a_1} v'_1 \colon b_1 \ldots \xrightarrow{e'_r \colon a_r} v'_r \colon b_r$ (kjer je $v'_i = h(v_i)$ pri vseh $i \in 0 \,..\, r$ in $e'_i = h(e_i)$ pri vseh $i \in 1 \,..\, r$) v r-sliki produkcije p. Na podoben način se preslika sprehod $w_0 \colon b_0 \xrightarrow{f_1 \colon a_1} w_1 \colon b_1 \ldots \xrightarrow{f_r \colon a_r} w_r \colon b_r$. Faza 1 nato dopolni pravkar odkrito r-sliko do produkcijskega primerka, kar stori tako, da nanjo pripne kopijo grafa $\mathit{Xlhs}[p]$. Sprehod $(v_r \colon b_r) \xrightarrow{e_{r+1} \colon a_{r+1}} v_{r+1} \colon b_{r+1} \ldots \xrightarrow{e_s \colon a_s} v_s \colon b_s$ v grafu $\mathit{Xlhs}[p]$ predstavlja verigo elementov z oznakami $a_{r+1}, b_{r+1}, \ldots, a_s, b_s$, pripeto na vozlišče $v'_r = h(v_r)$, sprehod $(w_r \colon b_r) \xrightarrow{f_{r+1} \colon a_{r+1}} w_{r+1} \colon b_{r+1} \ldots \xrightarrow{f_s \colon a_s} w_s \colon b_s$ pa verigo elementov z oznakami $a_{r+1}, b_{r+1}, \ldots, a_s, b_s$, pripeto na vozlišče $w'_r = h(w_r)$. Verigi sta sestavljeni iz enako označenih elementov na istoležnih pozicijah (lahko pa celo sovpadata), poleg tega pa ju faza 1 na r-sliko pripne na isti relativni poziciji glede na vozlišče $h(v_0)$ oziroma $h(w_0)$, saj ima sprehod od vozlišča $h(w_0)$ do vozlišča $h(w_r)$ (in naprej do vozlišča $h(w_s)$) isto dolžino in isto zaporedje oznak obiskanih grafnih elementov kot sprehod od vozlišča $h(v_0)$ do vozlišča $h(v_r)$ (in naprej do vozlišča $h(v_s)$). Ker ta lastnost velja za poljubno dvojico sprehodov po enako označenih elementih, ki se pričenjata v vozlišču v_0 oziroma w_0, lahko trdimo, da dopolnitev r-slike z vidika vozlišča $h(v_0)$ »izgleda« povsem enako kot z vidika vozlišča $h(w_0)$. (To pomeni, da za vsak sprehod S iz vozlišča $h(v_0)$ obstaja sprehod iz vozlišča $h(w_0)$, ki obišče enako zaporedje oznak kot sprehod S.) To opažanje lahko posplošimo na vseh k vozlišč zamenljivostnega razreda, saj smo vozlišči v_0 in w_0 iz zamenljivostnega razreda izbrali naključno.

Ker zamenljivostni razred sestoji iz k vozlišč, bi faza 1 brez naše izboljšave vsako r-sliko produkcije p odkrila in dopolnila do $k!$-krat. Vsaka dopolnitev bi z vidika k vozlišč, ki tvorijo zamenljivostni razred, »izgledala« povsem enako. (To pomeni, da bi za vsak sprehod S iz enega od teh k vozlišč obstajali sprehodi iz vsakega od ostalih $k - 1$ vozlišč, ki bi obiskali enako zaporedje oznak kot sprehod S.) Ker se pri odkritju r-slike skupina medsebojno zamenljivih vozlišč v grafu $\mathit{Rhs}[p]$ vedno preslika na skupino medsebojno zamenljivih vozlišč znotraj r-slike, bi vsako odkritje r-sliko dopolnilo na enak način z vidika skupine vozlišč iz zamenljivostnega razreda. Neizboljšana faza 1 bi svojo nalogo (odkritje r-slike in njeno dopolnitev) potemtakem do $k!$-krat ponovila. Ker bi imele vse ponovitve povsem enak učinek, zadošča, da vsako r-sliko odkrijemo in dopolnimo le enkrat. Torej lahko v primeru medsebojno zamenljivih vozlišč *na ravni produkcije* povsem varno uporabimo postopek preprečevanja večkratnih odkrivanj istih r-slik, ki smo ga opisali v trditvi 3.20. □

Trditev 3.22 smo dokazali zgolj za primer skupine medsebojno zamenljivih vozlišč. Dokaz za skupino medsebojno zamenljivih povezav bi zgradili na podoben

način.

3.4.2.4 Iskanje zamenljivostnih razredov

V tem odseku bomo predstavili algoritem za iskanje medsebojno neodvisnih zamenljivostnih razredov na ravni dane produkcije. Osredotočili se bomo na zamenljivostne razrede, sestavljene iz vozlišč, saj zamenljivostnih razredov povezav ni težko poiskati neposredno po definiciji 3.18. Uporaba definicij 3.12 in 3.15 za iskanje zamenljivostnih razredov vozlišč pa ni najprimernejša, saj pogoja 2 in 3 v definiciji 3.15 zahtevata preverjanje velikega števila izomorfizmov: za množico k vozlišč bi morali preveriti $\Theta(k!)$ izomorfizmov.

V nadaljevanju bomo uporabljali sledeči notacijski dogovor:

Dogovor 3.23. Naj $copy(G,\ x_{i[1]}\colon L_1,\ \ldots,\ x_{i[k]}\colon L_k)$ označuje kopijo grafa G, v kateri so kopije elementov $x_{i[1]}, \ldots, x_{i[k]} \in G$ označene z oznakami $L_1, \ldots, L_k$. Na primer, za graf $Rhs[p_2]$ s slike 3.7 je graf H_1 na sliki 3.9 (zgornji levi graf na desni strani te slike) definiran kot $copy(Rhs[p_2],\ v_2\colon \$_1,\ v_3\colon \$_2,\ v_4\colon \$_3)$.

Algoritem za iskanje zamenljivostnih razredov temelji na sledečih trditvah:

Trditev 3.24. Naj bosta u in v vozlišči grafa G in naj bosta $\$_1$ in $\$_2$ oznaki, ki ne nastopata v množici oznak grafa G ($\{\$_1, \$_2\} \cap Labels[G] = \emptyset$). Naj bosta grafa H_1 in H_2 definirana kot $H_1 = copy(G,\ u\colon \$_1,\ v\colon \$_2)$ in $H_2 = copy(G,\ u\colon \$_2,\ v\colon \$_1)$. Potem velja sledeča ekvivalenca:

$$\mathbf{inter}[G](u, v) \iff label[G](u) = label[G](v) \wedge H_1 \simeq H_2. \tag{3.1}$$

Dokaz. ($\Longrightarrow$) Če velja $\mathbf{inter}[G](u, v)$, potem v grafu G obstaja avtomorfizem h, tako da velja $h(u) = v$ in $h(v) = u$. To je mogoče samo tedaj, ko imata vozlišči u in v isto oznako. Ker sta grafa H_1 in H_2 kopiji grafa G z izjemo vozlišč z oznakama $\$_1$ in $\$_2$ (ki sta kopiji medsebojno zamenljivih vozlišč u in v), avtomorfizem h neposredno določa izomorfizem h' med grafoma H_1 in H_2. Predpisa $h(u) = v$ in $h(v) = u$ določata predpisa $h'(u'_1) = v'_2$ in $h'(v'_1) = u'_2$, kjer je u'_1 (oz. u'_2) kopija vozlišča u v grafu H_1 (oz. H_2), v'_1 (oz. v'_2) pa kopija vozlišča v v grafu H_1 (oz. H_2). Grafa H_1 in H_2 sta potemtakem izomorfna.

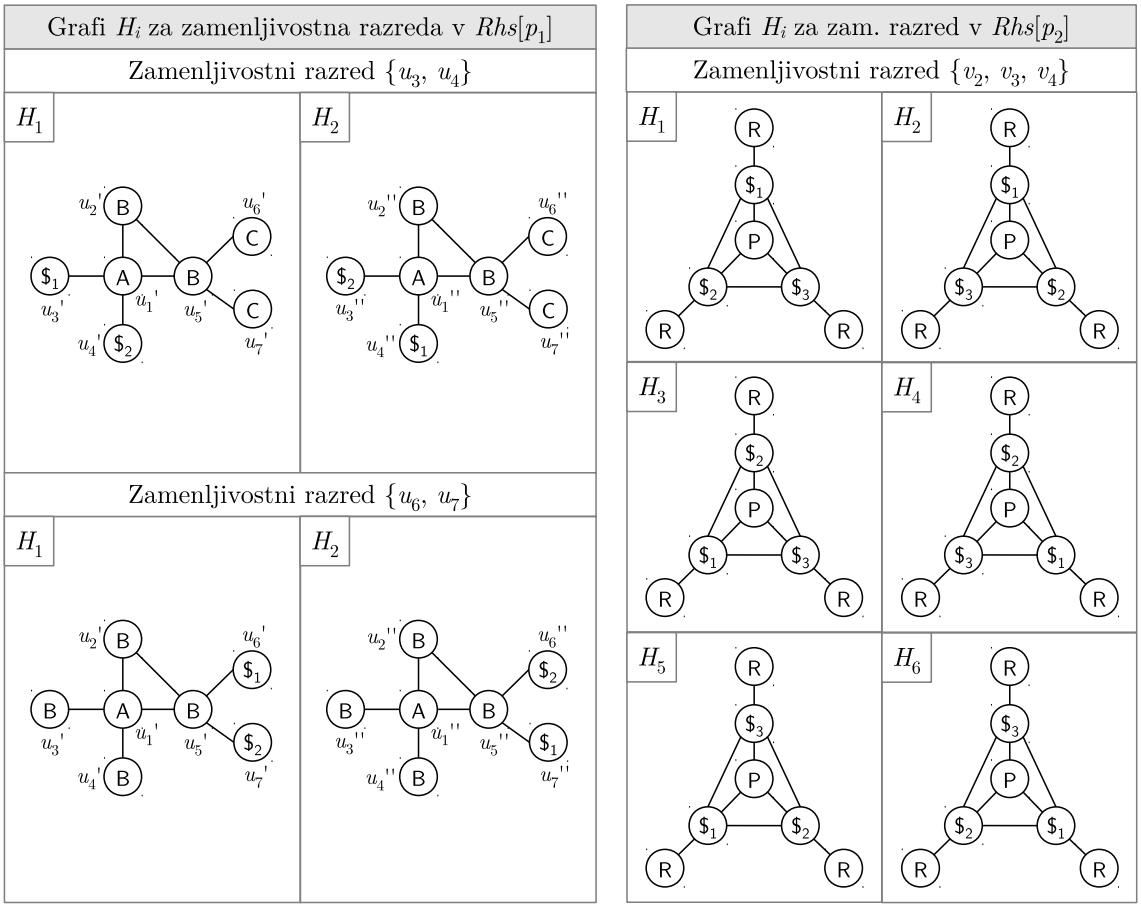
($\Longleftarrow$) Če velja $H_1 \simeq H_2$, potem poljuben izomorfizem med grafoma H_1 in H_2 preslika vozlišči z oznakama $\$_1$ in $\$_2$ v grafu H_1 v vozlišči z oznakama $\$_1$ in $\$_2$ v grafu H_2, saj se oznaki $\$_1$ in $\$_2$ po predpostavki ne pojavljata nikjer drugje v grafih H_1 in H_2. Ti dve preslikavi ustrezata preslikavama $u \mapsto v$ in $v \mapsto u$ v grafu G. Še več: če velja $label(u) = label(v)$, potem izomorfizem med grafoma H_1 in H_2 ustreza avtomorfizmu v grafu G, saj sta grafa H_1 in H_2 z izjemo oznak kopij vozlišč u in v izomorfna grafu G. □

Trditev 3.25. Naj bodo $v_1, \ldots, v_k$ vozlišča v grafu G, oznake $\$_1, \ldots, \$_k$ pa naj ne bodo prisotne nikjer v grafu G. Pri vsakem $i \in 1\,..\,k!$ naj σ_i označuje i-to permutacijo množice $\{\$_1, \ldots, \$_k\}$, graf H_i pa naj bo definiran kot $H_i = copy(G,\ v_1\colon \sigma_i(\$_1), \ldots, v_k\colon \sigma_i(\$_k))$. Potem velja sledeča ekvivalenca:

$$\mathbf{inter}[G](v_1, \ldots, v_k) \iff (label(v_1) = \ldots = label(v_k)) \wedge (H_1 \simeq \ldots \simeq H_{k!}). \tag{3.2}$$

Dokaz. Ta trditev je posplošitev trditve 3.24. Dokažemo jo lahko na podoben način. □

Slika 3.9 prikazuje grafe H_i za grafa $Rhs[p_1]$ in $Rhs[p_2]$ s slike 3.7. Lahko preverimo, da so vsi grafi H_i za isti zamenljivostni razred med seboj izomorfni. Na primer, med grafoma H_1 in H_2 za zamenljivostni razred $\{u_6, u_7\}$ grafa $Rhs[p_1]$ obstaja izomorfizem $\{u'_1 \mapsto u''_1, u'_2 \mapsto u''_2, u'_3 \mapsto u''_3, u'_4 \mapsto u''_4, u'_5 \mapsto u''_5, u'_6 \mapsto u''_7, u'_7 \mapsto u''_6\}$. Ta izomorfizem ustreza avtomorfizmu $\{u_1 \mapsto u_1, u_2 \mapsto u_2, u_3 \mapsto u_3, u_4 \mapsto u_4, u_5 \mapsto u_5, u_6 \mapsto u_7, u_7 \mapsto u_6\}$ v grafu $Rhs[p_1]$.



Slika 3.9: *Grafi H_i iz trditve 3.25 za posamezne zamenljivostne razrede v produkcijah p_1 in p_2 s slike 3.7.*

Algoritem za iskanje zamenljivostnih razredov bomo izdelali na podlagi sledeče trditve:

Trditev 3.26. Naj bo $\{v_1, \ldots, v_k\}$ pri $k > 2$ podmnožica množice vozlišč v grafu G. Denimo, da graf G ne vsebuje nobene izmed oznak $\$_1, \ldots, \$_k, \pounds_1, \ldots, \pounds_k$. Pri danem indeksu $i \in 1\,..\,k-1$ definirajmo graf $K_i = copy(G, v_1\colon \pounds_1, \ldots, v_{i-1}\colon \pounds_{i-1}, v_{i+1}\colon \pounds_{i+1}, \ldots, v_{k-1}\colon \pounds_{k-1})$. Potem velja sledeče:

$$\mathbf{inter}[G](v_1, \ldots, v_k) \iff \mathbf{inter}[G](v_1, \ldots, v_{k-1}) \wedge \mathbf{inter}[K_1](v_1, v_k) \wedge \mathbf{inter}[K_2](v_2, v_k) \wedge \ldots \wedge \mathbf{inter}[K_{k-1}](v_{k-1}, v_k). \quad (3.3)$$

Dokaz. ($\Longrightarrow$) Denimo, da so vozlišča $v_1, \ldots, v_k$ v grafu G medsebojno zamenljiva. Potem je po trditvi 3.25 graf $H_1 = \mathit{copy}(G, v_1\colon \$_1, \ldots, v_k\colon \$_k)$ izomorfen poljubnemu grafu $\mathit{copy}(G, v_1\colon \sigma(\$_1), \ldots, v_k\colon \sigma(\$_k))$, kjer je σ neka permutacija množice $\{\$_1, \ldots, \$_k\}$. Z drugimi besedami: v grafu H_1 lahko oznake $\$_1, \ldots, \$_k$, ki jih nosijo kopije vozlišč $v_1, \ldots, v_k$, med seboj poljubno zamenjujemo, pa bomo vedno dobili graf, izomorfen grafu H. Če lahko med seboj poljubno zamenjujemo oznake *vseh* kopij vozlišč $v_1, \ldots, v_k$, lahko tudi oznako kopije vozlišča v_k fiksiramo in medsebojno zamenjujemo zgolj oznake kopij vozlišč $v_1, \ldots, v_{k-1}$; tudi v tem primeru bodo vsi nastali grafi izomorfni grafu H_1. Torej velja $\mathbf{inter}[G](v_1, \ldots, v_{k-1})$. Veljavnost lastnosti $\mathbf{inter}[K_i](v_i, v_k)$ pri $i \in 1\,..\,k-1$ pa pokažemo tako, da preverimo, ali sta grafa $L_1 = \mathit{copy}(K_i, v_i\colon \$_i, v_k\colon \$_k)$ in $L_2 = \mathit{copy}(K_i, v_i\colon \$_k, v_k\colon \$_i)$ izomorfna. To drži, saj lahko graf L_1 zapišemo kot $\mathit{copy}(G, v_1\colon \pounds_1, \ldots, v_{i-1}\colon \pounds_{i-1}, v_i\colon \$_i, v_{i+1}\colon \pounds_{i+1}, \ldots, v_{k-1}\colon \pounds_{k-1}, v_k\colon \$_k)$, graf L_2 pa kot $\mathit{copy}(G, v_1\colon \pounds_1, \ldots, v_{i-1}\colon \pounds_{i-1}, v_i\colon \$_k, v_{i+1}\colon \pounds_{i+1}, \ldots, v_{k-1}\colon \pounds_{k-1}, v_k\colon \$_i)$. Graf L_2 je torej kopija grafa L_1, le da sta oznaki kopij vozlišč v_i in v_k med seboj zamenjani. Ker so vozlišča $v_1, \ldots, v_k$ v grafu G medsebojno zamenljiva, sta po trditvi 3.25 grafa L_1 in L_2 izomorfna. Torej velja $\mathbf{inter}[K_i](v_i, v_k)$ za vsak $i \in 1\,..\,k-1$.

($\Longleftarrow$) Naj bo $H_1 = \mathit{copy}(G, v_1\colon \$_1, \ldots, v_k\colon \$_k)$. Preveriti moramo, ali je graf H_1 izomorfen poljubnemu grafu $H = \mathit{copy}(G, v_1\colon \sigma(\$_1), \ldots, v_k\colon \sigma(\$_k))$, kjer je σ permutacija množice $\{\$_1, \ldots, \$_k\}$. Če velja $\sigma(\$_k) = \$_k$, potem to gotovo drži, saj so po predpostavki vozlišča $v_1, \ldots, v_{k-1}$ medsebojno zamenljiva, zaradi česar je v grafu H možno oznake $\$_1, \ldots, \$_{k-1}$ poljubno zamenjevati med seboj, ne da bi izgubili izomorfnost. Če velja $\sigma(\$_k) = \$_i$, kjer je $i \neq k$, pa postopajmo takole: Definirajmo graf $H' = \mathit{copy}(H_1, v_i\colon \$_k, v_k\colon \$_i)$. Predpostavljena lastnost $\mathbf{inter}[K_i](v_i, v_k)$ pomeni, da lahko v grafu z unikatnimi oznakami vozlišč $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}$ (to namreč velja za graf K_i) zamenjamo oznaki vozlišč v_i in v_k, pa bo nastali graf izomorfen prvotnemu. Torej je graf H' izomorfen grafu H_1. Zaradi lastnosti $\mathbf{inter}[G]\{v_1, \ldots, v_{k-1}\}$ lahko sedaj oznake vozlišč $v_1, \ldots, v_{k-1}$ v grafu H' med seboj poljubno zamenjujemo, pa bomo vedno dobili graf, izomorfen grafu H'. Če oznake teh vozlišč medsebojno zamenjamo tako, da velja $\mathit{label}(v_1) = \sigma(\$_1), \ldots, \mathit{label}(v_{k-1}) = \sigma(\$_{k-1})$, dobimo (ob upoštevanju dejstva $\mathit{label}(v_k) = \sigma(\$_k)$) kopijo grafa H. Torej lahko zaključimo, da je graf H_1 izomorfen grafu H ne glede na permutacijo σ. Potemtakem so vozlišča $v_1, \ldots, v_k$ v grafu G res medsebojno zamenljiva. □

Ekvivalenca (3.3) nam omogoča, da problem preverjanja medsebojne zamenljivosti k vozlišč učinkovito prevedemo na manjši problem. Medsebojno zamenljivost vozlišč $v_1, \ldots, v_k$ (pri $k > 2$) v grafu G preverimo tako, da najprej (rekurzivno) preverimo medsebojno zamenljivost vozlišč $v_1, \ldots, v_{k-1}$, nato pa preverimo še medsebojne zamenljivosti za vse pare $(v_1, v_k), \ldots, (v_{k-1}, v_k)$, pri čemer pred obravnavo para (v_i, v_k) (za $i \in 1\,..\,k-1$) vsem vozliščem $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{k-1}$ dodelimo unikatne oznake.

Če z $N(k)$ označimo število izomorfizmov, ki jih moramo preveriti za ugotavljanje medsebojne zamenljivosti k vozlišč, potem na podlagi ekvivalence (3.3) dobimo relaciji $N(k) = 1$ pri $k = 2$ in $N(k) = N(k-1) + k - 1$ pri $k > 2$. Od tod sledi $N(k) = \Theta(k^2)$, kar je bistveno bolje od izhodiščnega števila $\Theta(k!)$.

Procedura poišči**N**eodvisne**R**azrede na sliki 3.10 sprejme graf G, ki ne sme vsebovati vozlišč z oznakami $\$_i$ pri $i \geq 1$, in vrne množico medsebojno neodvisnih

zamenljivostnih razredov vozlišč $\mathcal{I}$ v tem grafu. Procedura prične s prazno množico $\mathcal{I}$ in nato vanjo postopoma dodaja zamenljivostne razrede. Kadar najde dvojico medsebojno zamenljivih vozlišč u in v, poskuša s pomočjo procedure povečajRazred poiskati vsa vozlišča, ki sodijo v isti zamenljivostni razred kot vozlišči u in v. Pogoj v vrstici 5 (**inter**$[G](u, v)$) preverjamo s pomočjo ekvivalence (3.3). Množica Q v proceduri poišči NeodvisneRazrede vsebuje vozlišča, ki še niso del nobenega odkritega zamenljivostnega razreda; na ta način preprečimo večkratno odkrivanje istih zamenljivostnih razredov.

```
1  procedura poišči NeodvisneRazrede(G)
2      I := ∅;
3      Q := V[G];
4      i := 1;
5      dokler ∃u, v ∈ Q: inter[G](u, v) izvajaj
6          C := povečajRazred(u, v, Q);
7          I := I ∪ {C};
8          za vsak z ∈ C izvedi label(z) := $_i;  i := i + 1 konec;
9          Q := Q \ C
10     konec;
11     vrni I
12 konec;
13
14 procedura povečajRazred(u, v, Q)
15     C := {u, v};
16     R := Q \ {u, v};
17     sprememba := 1;
18     dokler (sprememba = 1) izvajaj      // dokler se razred C povečuje ...
19         sprememba := 0;
20         za vsak z ∈ R izvedi      // Lahko razred C povečamo z vozliščem z?
21             C := C ∪ {z};
22             če je C zamenljivostni razred v grafu G potem
23                 R := R \ {z};
24                 sprememba := 1;
25                 prekini notranjo zanko   // pojdi v naslednji obhod zanke dokler
26             konec;
27             C := C \ {z}
28         konec
29     konec;
30     vrni C
31 konec
```

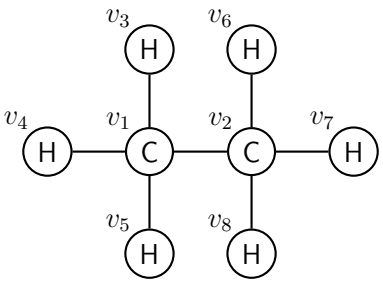
Slika 3.10: *Algoritem za iskanje neodvisnih zamenljivostnih razredov v podanem grafu.*

Po odkritju in širitvi zamenljivostnega razreda procedura poišči NeodvisneRazrede spremeni oznake vozlišč, ki tvorijo zamenljivostni razred, v enolične oznake oblike

$\$_i$. Na ta način zagotovimo, da bo procedura poišči Neodvisne Razrede v nadaljevanju odkrivala zgolj tiste zamenljivostne razrede, ki so neodvisni od doslej odkritih. Brez vrstice 8 bi procedura v grafu $Rhs[p_2]$ s slike 3.7 odkrila tako razred $\{v_2, v_3, v_4\}$ kot razred $\{v_5, v_6, v_7\}$, kar bi bilo napačno.

Procedura poišči Neodvisne Razrede poišče neko množico medsebojno neodvisnih zamenljivostnih razredov, za katero pa ni nujno, da pokriva največje možno število vozlišč. Na primer, v grafu na sliki 3.11 bi lahko algoritem našel katerokoli od sledečih množic:

- $\mathcal{I}_1 = \{\{v_1, v_2\}, \{v_3, v_4, v_5\}, \{v_6, v_7, v_8\}\}$.
- $\mathcal{I}_2 = \{\{v_3, v_4, v_5\}, \{v_6, v_7, v_8\}\}$.
- $\mathcal{I}_3 = \{\{v_3, v_6\}, \{v_4, v_5\}, \{v_7, v_8\}\}$.



Slika 3.11: *Strukturna formula etana.*

Rezultat procedure poišči Neodvisne Razrede je odvisen od tega, kateri zamenljivostni razred bo najprej odkrila. Množico $\mathcal{I}_1$, ki je s stališča pokritosti grafa optimalna, lahko zgradi samo tedaj, če najprej odkrije razred $\{v_1, v_2\}$; če procedura na začetku odkrije, denimo, razred $\{v_3, v_4, v_5\}$, potem razreda $\{v_1, v_2\}$ ne bo več odkrila, saj po spremembi oznak vozlišč v_3, v_4 in v_5 v unikatne oznake vozlišči v_1 in v_2 ne tvorita več zamenljivostnega razreda. Če bi želeli zgraditi optimalno množico zamenljivostnih razredov, bi morali poskusiti z več različnimi indeksiranji vozlišč, saj procedura poišči Neodvisne Razrede obravnava vozlišča po nekem fiksnem vrstnem redu. Lahko pa bi uporabili katerega od številnih optimizacijskih pristopov, saj je iskanje optimalne množice medsebojno neodvisnih zamenljivostnih razredov mogoče obravnavati kot problem iskanja v prostoru tovrstnih množic.

Slika 3.12 prikazuje zamenljivostne razrede na ravni posameznih produkcij, ki jih za naše testne gramatike (sliki 3.1 in 3.2 na straneh 53 in 54) odkrije procedura poišči Neodvisne Razrede. Prikazali smo samo produkcije, ki vsebujejo medsebojno zamenljive elemente. Nobena produkcija ne vsebuje več kot dveh medsebojno neodvisnih zamenljivostnih razredov. Elementi, ki pripadajo prvemu zamenljivostnemu razredu v produkciji, so označeni z modrimi krogci, za elemente drugega zamenljivostnega razreda (kjer ta obstaja) pa smo uporabili zelene krogce. Naj spomnimo, da za zamenljivost na ravni *produkcije* ni dovolj, če so grafni elementi medsebojno zamenljivi zgolj na ravni desne strani produkcije; izpolnjeni morajo biti vsi pogoji iz definicije 3.15. Tako na primer vozlišči z oznako Entity v produkciji p_2 gramatike

GG_{ER} (slika 3.2) nista medsebojno zamenljivi na ravni produkcije, čeprav sta medsebojno zamenljivi na ravni njene desne strani. To pomeni, da večkratnih odkritij istih r-slik produkcije p_2 gramatike GG_{ER} ni mogoče odpraviti; če bi jih, bi tvegali napačno delovanje sintaksnega analizatorja.

3.4.2.5 Kdaj se izboljšava 2 obrestuje?

Odgovor na to vprašanje je na dlani: izboljšava 2 se pri sintaksni analizi grafa v dani grafni gramatiki obrestuje, ko gramatika vsebuje vsaj en par medsebojno zamenljivih elementov. Kot bomo videli v podpoglavju 3.5, izboljšava 2 prinese ogromne časovno-prostorske prihranke pri gramatikah GG_{AHC} in GG_{HC} ter celo pri gramatiki GG_{FC}, ki vsebuje en sam par medsebojno zamenljivih vozlišč v celotni množici produkcij. Pri gramatikah GG_{AHC} in GG_{FC} smo pri sintaksni analizi zaporedja testnih grafov z vključitvijo izboljšave 2 dosegli zmanjšanje velikostnega reda časovne zahtevnosti z eksponentnega na polinomskega.

3.4.3 Izboljšava 3: Zmanjševanje potrebe po vrednotenju relacije nekonsistentnosti

Spomnimo se, da je *produkcijski primerek* (definicija 3.1) p-homomorfna slika celotne produkcije v grafu $\overline{G}$, zapisana v obliki trojice l-slike, c-slike in r-slike produkcije. V nadaljevanju se bomo držali sledečega dogovora glede notacije: oznake p, p', p_1 ipd. bodo predstavljale produkcije, oznake pi, pi', pi_1 ipd. pa produkcijske primerke.

Faza 1, kot vemo, v grafu $\overline{G}$ išče r-slike produkcij in graf dopolnjuje s kopijami njihovih množic *Xlhs*, pri tem pa ustvarja produkcijske primerke. Da bi se izognili morebitni neskončni zanki odkrivanj in dopolnjevanj grafa $\overline{G}$, moramo v fazi 1 ovrednotiti relacijo **inconsistent** po vsakem nastanku novega produkcijskega primerka. Žal pa je ta postopek računsko zahteven, saj relacija **inconsistent** temelji na trojiški relaciji $\mathbf{above}^{+}_{pi}$, ki pri n produkcijskih primerkih potrebuje $\Theta(n^3)$ časa. Če pa bi za produkcijo p vnaprej vedeli, da za noben njen primerek pi ne more veljati $\mathbf{inconsistent}(pi)$, bi se lahko izognili vrednotenju relacije **inconsistent** (ter tudi $\mathbf{excludes}^{\star}_{\mathrm{self}}$ in $\mathbf{above}^{+}_{pi}$) za vse primerke produkcije p. V tem razdelku bomo opisali pogoje, pod katerimi so takšni prihranki mogoči.

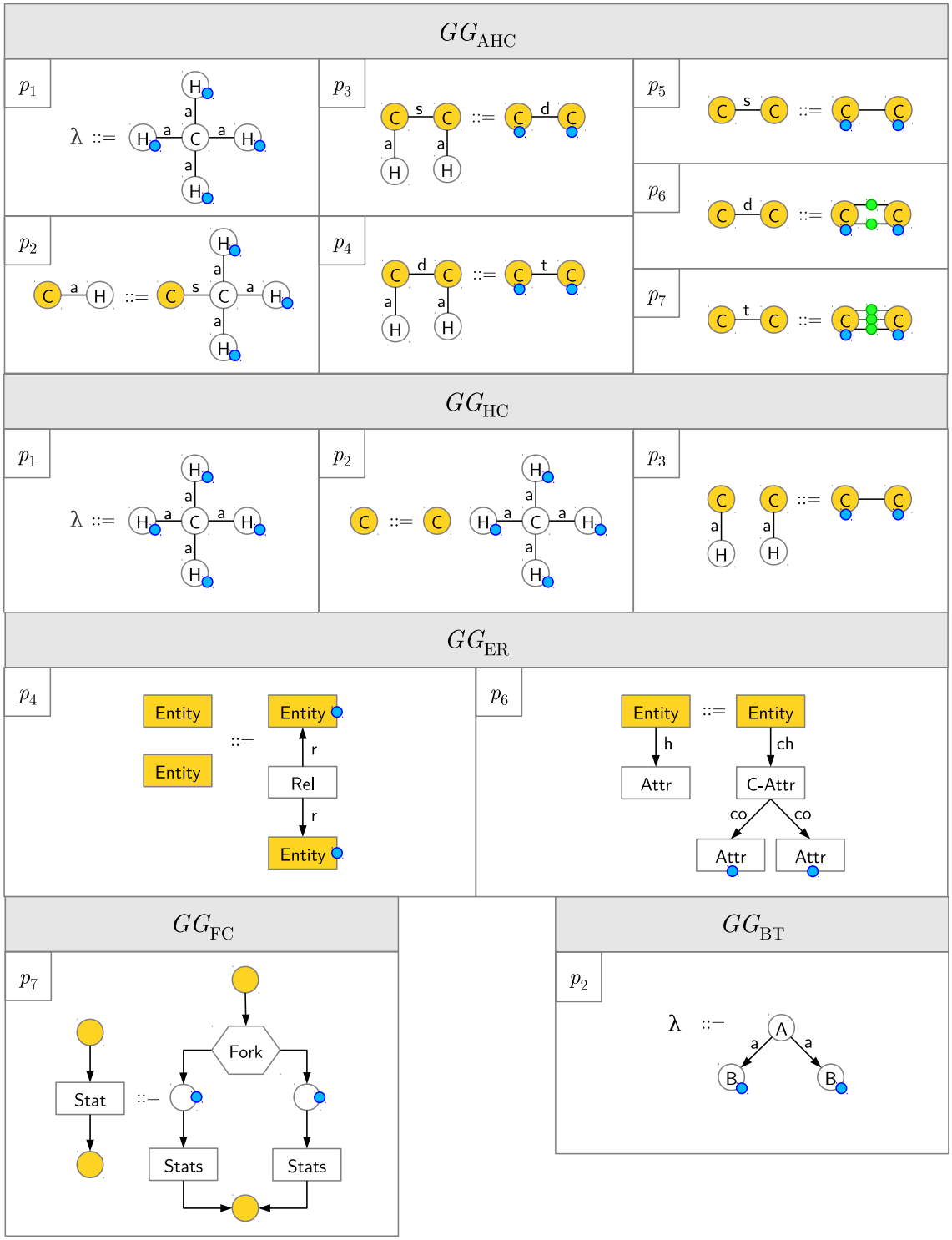
Naj zapis $Inst[p]$ označuje množico vseh mogočih primerkov produkcije p, ki jih lahko ustvari faza 1, upoštevajoč vse možne vhodne grafe. V splošnem o množici $Inst[p]$ ne moremo reči ničesar, v nekaterih primerih pa zanjo, kot bomo videli, veljajo določena pravila. Naš cilj bo ugotoviti, ali množica $Inst[p]$ pri neki produkciji p vsebuje tudi nekonsistentne primerke. Formalizirajmo to možnost s sledečo definicijo:

Definicija 3.27 (relacija **pInconsistent**)**.** Naj bo p produkcija. Potem je vrednost relacije $\mathbf{pInconsistent}(p)$ definirana z logičnim izrazom

$$\mathbf{pInconsistent}(p) \iff \exists pi \in Inst[p] \colon \mathbf{inconsistent}(pi) \tag{3.4}$$

oziroma z enakovrednim izrazom

$$\neg\mathbf{pInconsistent}(p) \iff \forall pi \in Inst[p] \colon \neg\mathbf{inconsistent}(pi). \tag{3.5}$$



Slika 3.12: *Zamenljivostni razredi v naših testnih gramatikah.*

Če torej velja **pInconsistent**(p), potem ne moremo vnaprej reči ničesar o veljavnosti relacije **inconsistent**(pi) za posamezne primerke produkcije pi. Če velja $\neg$**pInconsistent**(p), pa lahko z gotovostjo trdimo, da bo za *vse mogoče* primerke produkcije p (torej za vse $pi \in Inst[p]$) veljalo $\neg$**inconsistent**(pi).

Ker v splošnem ne moremo reči ničesar o možnosti nastopa nekonsistentnega primerka neke produkcije, bomo za vsako produkcijo p iz vhodne gramatike privzeli **pInconsistent**(p). Tudi če to za katero od produkcij ne bo veljalo, ne bomo z (napačno) predpostavko **pInconsistent**(p) izgubili ničesar, saj bomo pri sintaksni analizi grafa zgolj po nepotrebnem vrednotili relacijo **inconsistent** za primerke takšne produkcije. V nadaljevanju bomo poiskali pogoje, pri katerih zanesljivo velja $\neg$**pInconsistent**(p). Če so ti pogoji izpolnjeni, potem po definiciji 3.27 vemo, da noben primerek produkcije p ne bo nekonsistenten (ne glede na vhodni graf), zato nam za primerke produkcije p ne bo nikoli treba vrednotiti računsko zahtevne relacije **inconsistent**. Tako lahko v ugodnih primerih dosežemo precejšnje prihranke.

Po zgledu relacije **pInconsistent** definirajmo še relacije **pConseqOf**, **pAbove**, **pExcludes** itd., saj jih bomo potrebovali pri določanju pogojev za veljavnost izraza $\neg$**pInconsistent**(p):

Definicija 3.28 (relacije **pConseqOf**, **pAbove** itd.)**.** Naj bodo p, p_1 in p_2 produkcije vhodne gramatike. Relacije **pConseqOf**, **pAbove** itd. so definirane takole:

- $\exists pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **conseqOf** $pi_2 \iff p_1$ **pConseqOf** p_2
- $\exists pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **conseqOf**$^+$ $pi_2 \iff p_1$ **pConseqOf**$^+$ p_2
- $\exists pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **above** $pi_2 \iff p_1$ **pAbove** p_2
- $\exists pi \in Inst[p],\ pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **above**$_{pi}$ $pi_2 \iff$ p_1 **pAbove**$_p$ p_2
- $\exists pi \in Inst[p],\ pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **above**$^+_{pi}$ $pi_2 \iff$ p_1 **pAbove**^+_p p_2
- $\exists pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **excludes** $pi_2 \iff p_1$ **pExcludes** p_2
- $\exists pi_1 \in Inst[p_1],\ pi_2 \in Inst[p_2]\colon pi_1$ **excludes*** $pi_2 \iff p_1$ **pExcludes*** p_2
- $\exists pi \in Inst[p]\colon$ **excludes**$^\star_{\text{self}}(pi) \iff$ **pExcludes**$^\star_{\text{self}}(p)$
- $\exists pi \in Inst[p]\colon$ **inconsistent**$(pi) \iff$ **pInconsistent**(p)

Medtem ko relacija pi **conseqOf** pi ne velja za noben produkcijski primerek pi, je veljavnost relacije p **pConseqOf** p možna, saj se lahko zgodi, da sta različna primerka iste produkcije p v relaciji **conseqOf**. Še več, privzeli bomo, da je relacija p **pConseqOf** p veljavna, razen če bodo pogoji, ki jih bomo predstavili v nadaljevanju, narekovali nasprotno. Enako bomo privzeli za vse ostale relacije (p_1 **pConseqOf** p_2, p_1 **pConseqOf**$^+$ p_2, p_1 **pAbove** p_2 itd.).

V nadaljevanju tega podrazdelka bomo potrebovali sledečo pomožno definicijo:

Definicija 3.29 (množica *CLabels*)**.** Naj bo G usmerjen graf ali množica grafnih elementov z morebitnimi visečimi povezavami. Potem je množica $CLabels[G]$ definirana tako:

$$\begin{aligned} CLabels[G] = {} & \{label(v) \mid v \in \mathcal{V}[G]\} \cup \\ & \{(label(v), label(e), label(w)) \mid e \in \mathcal{E}[G] \wedge v = source(e) \wedge w = target(e)\}. \end{aligned} \quad (3.6)$$

V primeru neusmerjenega grafa ali množice grafnih elementov se definicija glasi

$$\begin{aligned} & CLabels[G] = \{label(v) \mid v \in \mathcal{V}[G]\} \cup \\ & \{(label(v), label(e), label(w)) \mid e \in \mathcal{E}[G] \wedge conn(e) = \{v, w\} \wedge label(v) \preceq label(w)\}, \end{aligned} \quad (3.7)$$

pri čemer zapis $\preceq$ predstavlja poljubno relacijo, ki popolnoma ureja množico oznak.

Na primer, če relacija $\preceq$ ureja množico oznak po abecedi in če je p produkcija na sliki 3.8, potem velja $CLabels[Rhs[p]] = \{\mathsf{A},\ \mathsf{D},\ (\mathsf{A},\ \mathsf{t},\ \mathsf{A}),\ (\mathsf{A},\ \mathsf{q},\ \mathsf{D})\}$ in $CLabels[Xrhs[p]] = \{\mathsf{D},\ (\mathsf{A},\ \mathsf{t},\ \mathsf{A}),\ (\mathsf{A},\ \mathsf{q},\ \mathsf{D})\}$. V primeru gramatike GG_{HC} (slika 3.1) velja $Labels[Xrhs[p_3]] \cap Labels[Xrhs[p_4]] = \{\#\}$ in $CLabels[Xrhs[p_3]] \cap CLabels[Xrhs[p_4]] = \emptyset$.

Sedaj bomo predstavili pogoj, pri katerem lahko za podani produkciji p_1 in p_2 trdimo, da velja $\neg(p_1\ \mathbf{pConseqOf}\ p_2)$:

Trditev 3.30. Za poljubni produkciji p_1 in p_2 iz vhodne gramatike velja

$$CLabels[Xlhs[p_1]] \cap CLabels[Rhs[p_2]] = \emptyset \implies \neg(p_1\ \mathbf{pConseqOf}\ p_2). \quad (3.8)$$

Dokaz. Če množici $Xlhs[p_1]$ in $Rhs[p_2]$ nimata nobenih skupnih oznak, potem se pri nobeni dvojici produkcijskih primerkov $pi_1 \in Inst[p_1]$ in $pi_2 \in Inst[p_2]$ ne more zgoditi, da bi se njuni sliki v grafu $\overline{G}$ (torej množici $Xlhs[pi_1]$ in $Rhs[pi_2]$) prekrivali. Za poljubno dvojico produkcijskih primerkov $pi_1 \in Inst[p_1]$ in $pi_2 \in Inst[p_2]$ torej velja $Xlhs[pi_1] \cap Rhs[pi_2] = \emptyset$ in potemtakem $\neg(pi_1\ \mathbf{conseqOf}\ pi_2)$. Od tod sledi $\neg(p_1\ \mathbf{pConseqOf}\ p_2)$. □

Podobne pogoje lahko postavimo tudi za druge relacije:

Trditev 3.31. Naj bodo p, p_1 in p_2 produkcije vhodne gramatike GG. Potem veljajo sledeči stavki:

- Relacija $\mathbf{pConseqOf}^+$ je tranzitivna ovojnica relacije **pConseqOf**.
- $\neg(p_2\ \mathbf{pConseqOf}\ p_1) \wedge$
  $CLabels[Xrhs[p_1]] \cap CLabels[Common[p_2]] = \emptyset \wedge$
  $\forall e \in \mathcal{E}[Xlhs[p_1]],\ v \in \mathcal{V}[Xlhs[p_2]]\colon label(v) \notin Labels[conn(e)]$
  $\implies \neg(p_1\ \mathbf{pAbove}\ p_2)$
- $\neg(p_1\ \mathbf{pAbove}\ p_2) \vee \neg(p_1\ \mathbf{pConseqOf}^+\ p) \vee \neg(p_2\ \mathbf{pConseqOf}^+\ p)$
  $\implies \neg(p_1\ \mathbf{pAbove}_p\ p_2)$

- Relacija $\mathbf{pAbove}_p^+$ je tranzitivna ovojnica relacije $\mathbf{pAbove}_p$.

- $(\neg(p_1\ \mathbf{pAbove}\ p_2) \vee \neg(p_2\ \mathbf{pAbove}\ p_1)) \wedge$
  $\quad CLabels[Xrhs[p_1]] \cap CLabels[Xrhs[p_2]] = \emptyset$
  $\implies \neg(p_1\ \mathbf{pExcludes}\ p_2)$

- $\neg(p_1\ \mathbf{pExcludes}\ p_2) \wedge$
  $\quad (\ \forall p', p'' \in \mathcal{P}[GG]\colon \neg(p'\ \mathbf{pConseqOf}^+\ p_1) \vee$
  $\qquad\qquad \neg(p''\ \mathbf{pConseqOf}^+\ p_2) \vee$
  $\qquad\qquad \neg(p'\ \mathbf{pExcludes}\ p'')\ )$
  $\implies \neg(p_1\ \mathbf{pExcludes}^\star\ p_2)$

- $\forall p', p'' \in \mathcal{P}[GG]\colon \neg(p'\ \mathbf{pConseqOf}^+\ p) \vee$
  $\qquad\qquad \neg(p''\ \mathbf{pConseqOf}^+\ p) \vee$
  $\qquad\qquad \neg(p'\ \mathbf{pExcludes}\ p'')$
  $\implies \neg\mathbf{pExcludes}^\star_{\text{self}}(p)$

- $\neg\mathbf{pExcludes}^\star_{\text{self}}(p) \wedge$
  $\quad (\forall p', p'' \in \mathcal{P}[GG]\colon \neg(p'\ \mathbf{pAbove}_p^+\ p'') \vee \neg(p''\ \mathbf{pAbove}_p^+\ p'))$
  $\implies \neg\mathbf{pInconsistent}(p)$

Dokaz. Stavki iz gornje trditve so pridobljeni iz definicij 3.3–3.11 na povsem enakovreden način kot trditev 3.30, zato jih lahko dokažemo na podoben način. □

Na podlagi stavkov iz trditve 3.31 lahko določimo, za katere produkcije p iz gramatike velja $\neg\mathbf{pInconsistent}(p)$. Kot smo povedali, lahko pri produkciji p z lastnostjo $\neg\mathbf{pInconsistent}(p)$ varno predpostavimo $\neg\mathbf{inconsistent}(pi)$ za vse njene primerke pi, ne glede na vhodni graf.

Relacija $\mathbf{pExcludes}^\star_{\text{self}}(p)$ ni enakovredna relaciji $p\ \mathbf{pExcludes}^\star\ p$, čeprav velja $\mathbf{excludes}^\star_{\text{self}}(pi) \iff pi\ \mathbf{excludes}^\star\ pi$. Če velja $\neg\mathbf{pExcludes}^\star_{\text{self}}(p)$, potem ne obstaja primerek pi produkcije p, da bi veljalo $pi\ \mathbf{excludes}^\star\ pi$, lahko pa obstajata *različna* primerka produkcije p (denimo pi_1 in pi_2), za katera velja $pi_1\ \mathbf{excludes}^\star\ pi_2$. Veljavnost izraza $\neg(p\ \mathbf{pExcludes}^\star\ p)$ pa pomeni, da v množici $Inst[p]$ ne obstajata niti enaka niti različna primerka pi_1 in pi_2 z lastnostjo $pi_1\ \mathbf{excludes}^\star\ pi_2$. To je razlog, zakaj smo v razdelku 3.3.3 uvedli relacijo $\mathbf{excludes}^\star_{\text{self}}$ in zakaj smo nekatere relacije definirali nekoliko drugače (četudi enakovredno) kot Rekers in Schürr.

Poleg stavkov iz trditve 3.31 lahko za relacijo **pExcludes** zapišemo še sledečo logično zvezo:

Trditev 3.32. Za poljubno produkcijo p in povezavo $e \in Xrhs[p]$ velja:

$$\begin{aligned}&\neg(p\ \mathbf{pAbove}\ p) \wedge Xrhs[p] = \{e\} \wedge Common[p] = conn(e) \wedge \\ &\quad label(conn_1(e)) \neq label(conn_2(e)) \\ &\implies \neg(p\ \mathbf{pExcludes}\ p).\end{aligned} \tag{3.9}$$

Če je v veljavi izboljšava 2, potem velja tudi sledeče:

$$\begin{aligned}\neg(p\,\mathbf{pAbove}\,p) \wedge \mathit{Xrhs}[p] = \{e\} \wedge \mathit{Common}[p] = \mathit{conn}(e) \wedge \\ \mathbf{inter}[p](\mathit{conn}_1(e), \mathit{conn}_2(e)) \\ \Longrightarrow\ \neg(p\,\mathbf{pExcludes}\,p).\end{aligned} \tag{3.10}$$

Dokaz. Če množica $\mathit{Xrhs}[p]$ vsebuje zgolj povezavo e (in nič drugega), potem lahko množica $\mathit{Xrhs}[pi_1] \cap \mathit{Xrhs}[pi_2]$ pri poljubni dvojici produkcijskih primerkov pi_1, $pi_2 \in \mathit{Inst}[p]$ vsebuje kvečjemu povezavo e' v grafu $\overline{G}$, ki je v primerkih pi_1 in pi_2 prirejena povezavi e. Ker množica $\mathit{Rhs}[p] = \mathit{Common}[p] \cup \mathit{Xrhs}[p]$ poleg povezave e vsebuje le še njuni krajišči $\mathit{conn}_1(e)$ in $\mathit{conn}_2(e)$, množica $\mathit{Xrhs}[pi_1] \cap \mathit{Xrhs}[pi_2]$ vsebuje povezavo e' samo tedaj, ko množici $\mathit{Rhs}[pi_1]$ in $\mathit{Rhs}[pi_2]$ sovpadata; sicer velja $\mathit{Xrhs}[pi_1] \cap \mathit{Xrhs}[pi_2] = \emptyset$. Če sta oznaki krajišč povezave e različni, potem množici $\mathit{Rhs}[pi_1]$ in $\mathit{Rhs}[pi_2]$ ne moreta nikoli sovpadati, saj bi to pomenilo, da bi faza 1 dvakrat odkrila ne samo isto r-sliko produkcije p, pač pa celo isti p-homomorfizem med grafoma $\mathit{Rhs}[p]$ in $\overline{G}$, kar se ne more zgoditi. To velja tudi v primeru, kadar sta oznaki krajišč medsebojno zamenljivi in je v veljavi izboljšava 2, ki prepreči dvakratno odkritje iste r-slike z dvema različnima p-homomorfizmoma. Zaradi tega velja $\mathit{Xrhs}[pi_1] \cap \mathit{Xrhs}[pi_2] = \emptyset$, odkoder ob predpostavki $\neg(p\,\mathbf{pAbove}\,p)$ sledi $\neg(pi_1\,\mathbf{excludes}\,pi_2)$. Ker ta lastnost velja za poljubna primerka pi_1 in pi_2, ki pripadata produkciji p, lahko zaključimo, da velja $\neg(p\,\mathbf{pExcludes}\,p)$. □

Tabela 3.4 prikazuje veljavnosti relacij **pConseqOf**, **pConseqOf**$^+$ itd. za posamezne pare produkcij gramatike GG_{HC} (slika 3.1), pridobljene na podlagi trditev 3.30–3.32. Oznaka p_{vr} predstavlja produkcijo v vrstici, oznaka p_{st} pa produkcijo v stolpcu tabele. Element z vrednostjo 0 pove, da obravnavani par produkcij ni v obravnavani relaciji, ker to sledi iz trditev 3.30–3.32. Elementi z vrednostjo 1 pa predstavljajo pare produkcij, za katere tega ne moremo trditi, zato zanje predpostavimo veljavnost obravnavane relacije. Tabela 3.5 na enak način prikazuje veljavnosti relacije **pInconsistent** za posamezne produkcije naših testnih gramatik. Vidimo, da nobena produkcija gramatike GG_{HC} ne more tvoriti nekonsistentnih produkcijskih primerkov, kar pomeni, da lahko pri sintaksni analizi grafov v tej gramatiki povsem odpravimo vrednotenje računsko zahtevne relacije **inconsistent**. Kot bomo videli v podpoglavju 3.5, vključitev izboljšave 3 pri gramatiki GG_{HC} močno zmanjša porabo časa za sintaksno analizo. Drugo skrajnost pa predstavlja gramatika GG_{FC}, pri kateri nam izboljšava 3 nič ne koristi, saj lahko — glede na pogoje iz trditev 3.30–3.32 — vse produkcije tvorijo nekonsistentne primerke.

Izboljšava 3 se obrestuje, kadar velja $\neg\mathbf{pInconsistent}(p)$ za veliko produkcij oziroma za produkcije, katerih primerki se v grafih pogosto pojavljajo. Vendar pa tega za podano gramatiko ne moremo napovedati vnaprej. Dobre možnosti imamo tedaj, kadar vhodna gramatika vsebuje veliko produkcij s praznimi množicami Xlhs. Pri takšni gramatiki za mnoge dvojice produkcij velja $\neg(p_1\,\mathbf{pConseqOf}\,p_2)$. Ker relacija **pInconsistent** temelji na relacijah $\mathbf{pAbove}_p$ in $\mathbf{pExcludes}^{\star}_{\mathrm{self}}$, ti dve pa na relaciji **conseqOf**, lahko upamo, da bo veliko število primerov $\neg(p_1\,\mathbf{pConseqOf}\,p_2)$ vodilo vsaj do nekaterih primerov $\neg\mathbf{pInconsistent}(p)$.

Tabela 3.4: *Vrednosti relacij* **pConseqOf**, **pConseqOf**$^+$ *itd. za produkcije gramatike* GG_{HC}.

p_{st} **pConseqOf** p_{vr}

	p_1	p_2	p_3	p_4
p_1	0	0	1	1
p_2	0	0	1	1
p_3	0	0	0	0
p_4	0	0	1	0

p_{st} **pConseqOf**$^+$ p_{vr}

	p_1	p_2	p_3	p_4
p_1	0	0	1	1
p_2	0	0	1	1
p_3	0	0	0	0
p_4	0	0	1	0

p_{vr} **pAbove** p_{st}

	p_1	p_2	p_3	p_4
p_1	0	1	1	1
p_2	0	1	1	1
p_3	0	0	0	0
p_4	0	0	1	0

p_{vr} **pAbove**$^+_{p_1}$ p_{st}

	p_1	p_2	p_3	p_4
p_1	0	0	1	1
p_2	0	0	0	0
p_3	0	0	0	0
p_4	0	0	1	0

p_{vr} **pAbove**$^+_{p_2}$ p_{st}

	p_1	p_2	p_3	p_4
p_1	0	0	0	0
p_2	0	0	1	1
p_3	0	0	0	0
p_4	0	0	1	0

p_{vr} **pAbove**$^+_{p_3}$ p_{st}

	p_1	p_2	p_3	p_4
p_1	0	0	0	0
p_2	0	0	0	0
p_3	0	0	0	0
p_4	0	0	0	0

p_{vr} **pAbove**$^+_{p_4}$ p_{st}

	p_1	p_2	p_3	p_4
p_1	0	0	0	0
p_2	0	0	0	0
p_3	0	0	0	0
p_4	0	0	1	0

p_{vr} **pExcludes** p_{st}

	p_1	p_2	p_3	p_4
p_1	1	1	0	0
p_2	1	1	0	0
p_3	0	0	0	0
p_4	0	0	0	0

p_{vr} **pExcludes*** p_{st}

	p_1	p_2	p_3	p_4
p_1	1	1	0	0
p_2	1	1	0	0
p_3	0	0	0	0
p_4	0	0	0	0

Tabela 3.5: *Vrednosti relacije* **pInconsistent** *za produkcije testnih gramatik.*

gramatika	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
GG_{AHC}	1	1	1	1	0	0	0	1
GG_{HC}	0	0	0	0				
GG_{ER}	0	0	0	0	1	1	1	
GG_{FC}	1	1	1	1	1	1	1	
GG_{BT}	0	1	0	1				

3.4.4 Izboljšava 4: Zgodnje odkrivanje nekonsistentnih produkcijskih primerkov

V zvezi z relacijo **inconsistent** se nam splača poiskati še kakšen prihranek, saj je njeno sprotno vrednotenje računsko zahtevno. Izboljšava, ki jo bomo predstavili v tem razdelku, izkorišča način iskanja r-slik posameznih produkcij v grafu $\overline{G}$. Spomnimo se, da faza 1 sledi iskalnim načrtom desnih strani; v vsakem koraku poskuša na vse mogoče načine izpolniti trenutno navodilo v iskalnem načrtu neke desne strani. Poleg tega iskalne načrte različnih desnih strani izpolnjuje — po terminologiji Rekersa in Schürra — *psevdoparalelno*. Namesto da bi najprej izpolnila iskalni načrt za desno stran prve produkcije ($Rhs[p_1]$), nato iskalni načrt za desno stran druge produkcije ($Rhs[p_2]$) itd., faza 1 najprej (na vse možne načine) izpolni prvo pravilo iskalnega načrta za graf $Rhs[p_1]$, nato prvo pravilo iskalnega načrta za graf $Rhs[p_2]$ itd. S takšnim načinom izvajanja iskalnih načrtov je namreč mogoče lažje zagotoviti sistematični prebor sproti dopolnjujočega se grafa $\overline{G}$.

Izboljšava 4 poskuša zaznati in odstraniti nekonsistentni produkcijski primerek že v njegovem nastajanju, torej tedaj, ko p-homomorfizem med elementi desne strani neke produkcije in elementi grafa $\overline{G}$ še ni v celoti vzpostavljen. Če ugotovimo, da se bo nek delni p-homomorfizem med desno stranjo in grafom $\overline{G}$ (torej p-homomorfna slika nekega podgrafa desne strani v grafu $\overline{G}$) zanesljivo razvil v nekonsistenten produkcijski primerek, potem lahko takšen delni p-homomorfizem predčasno zavržemo in s tem morda preprečimo nastanek še kakšnega nekonsistentnega produkcijskega primerka.

Izboljšava 4 temelji na sledeči trditvi:

Trditev 3.33. Naj bosta pi in qi dva različna produkcijska primerka. Potem velja sledeče:

$$(Rhs[pi] \cap Xlhs[qi] \neq \emptyset) \wedge (Rhs[pi] \cap Xrhs[qi] \neq \emptyset) \implies \mathbf{inconsistent}(pi). \quad (3.11)$$

Dokaz. Veljavnost implikacije pokažemo s pomočjo definicij 3.3–3.11 ter pravil izjavnega računa in teorije množic:

$$\begin{aligned}
&(Rhs[pi] \cap Xlhs[qi] \neq \emptyset) \wedge (Rhs[pi] \cap Xrhs[qi] \neq \emptyset) \\
&\quad \implies (qi\ \mathbf{conseqOf}\ pi) \wedge ((Xrhs[pi] \cup Common[pi]) \cap Xrhs[qi] \neq \emptyset) \\
&\quad \implies (qi\ \mathbf{conseqOf}\ pi \wedge pi\ \mathbf{above}\ qi) \wedge \\
&\qquad\quad ((Xrhs[pi] \cap Xrhs[qi] \neq \emptyset) \vee (Common[pi] \cap Xrhs[qi] \neq \emptyset)) \\
&\quad \implies (qi\ \mathbf{conseqOf}\ pi \wedge pi\ \mathbf{above}\ qi) \wedge (pi\ \mathbf{excludes}\ qi \vee qi\ \mathbf{above}\ pi) \\
&\quad \implies (qi\ \mathbf{conseqOf}\ pi \wedge pi\ \mathbf{excludes}\ qi) \vee \\
&\qquad\quad (qi\ \mathbf{conseqOf}\ pi \wedge pi\ \mathbf{above}\ qi \wedge qi\ \mathbf{above}\ pi) \\
&\quad \implies (qi\ \mathbf{conseqOf}^+\ pi \wedge pi\ \mathbf{excludes}\ qi) \vee (qi\ \mathbf{conseqOf}^+\ pi \wedge pi\ \mathbf{excludes}\ qi) \\
&\quad \implies qi\ \mathbf{conseqOf}^+\ pi \wedge pi\ \mathbf{excludes}\ qi \\
&\quad \implies pi\ \mathbf{excludes}^\star\ pi \\
&\quad \implies \mathbf{inconsistent}(pi).
\end{aligned}$$

□

S pomočjo trditve 3.33 lahko nekatere nekonsistentne produkcijske primerke zatremo že »v kali«, saj je pogoj (3.11) možno preverjati že tedaj, ko p-homomorfizem med desno stranjo produkcije in grafom $\overline{G}$ šele nastaja. Produkcijski primerek qi mora v celoti obstajati, saj se njegova izključna leva stran ($Xlhs[qi]$) zgradi šele ob dokončnem odkritju r-slike pripadajoče produkcije. Pri produkcijskem primerku pi pa to ni potrebno, saj se množica $Rhs[pi]$ vzpostavlja že med iskanjem slik elementov desne strani produkcije v grafu $\overline{G}$. Čim je pogoj iz trditve 3.33 izpolnjen, trenutni delni p-homomorfizem med desno stranjo in grafom $\overline{G}$ zavržemo; če bi se namreč delni p-homomorfizem razvil do popolnega p-homomorfizma, bi bil nastali produkcijski primerek zanesljivo nekonsistenten.

Opozoriti velja, da s pogojem (3.11) ne moremo zajeti vseh nekonsistentnih produkcijskih primerkov. Primerek pi je nekonsistenten tudi v primeru, če obstajata produkcijska primerka pi' in pi'', tako da velja pi' $\mathbf{above}^+_{pi}$ pi'' in pi'' $\mathbf{above}^+_{pi}$ pi'. Takšnih primerov trditev 3.33 ne zajema.

3.4.5 Izboljšava 5: Predčasen poskus izvedbe faze 2

Rekers-Schürrov sintaksni analizator je sestavljen iz dveh ločenih faz. Faza 2 se prične šele potem, ko faza 1 poišče vse r-slike produkcij v grafu $\overline{G}$ in tako zgradi popolno množico produkcijskih primerkov. Vendar pa se lahko zgodi, da že (majhen) del te množice zadošča za uspešno sintaksno analizo, torej da je izpeljavo vhodnega grafa mogoče zgraditi že z neko podmnožico množice produkcijskih primerkov, ki jih ustvari faza 1.

Izboljšava 5 temelji na sledeči hevristiki: Denimo, da je faza 1 pri sintaksni analizi s podanim grafom G in gramatiko GG pravkar ustvarila produkcijski primerek pi_0, ki pripada eni od začetnih produkcij gramatike. Če velja

$$Elements[G] \subseteq Xrhs[pi_0] \cup \bigcup_{pi\colon pi\ \mathbf{conseqOf}^+\ pi_0} Xrhs[pi], \tag{3.12}$$

potem je smiselno nemudoma poskusiti poiskati izpeljavo s pomočjo faze 2, saj obstaja možnost, da trenutna množica produkcijskih primerkov, ki jo je dotlej ustvarila faza 1, vsebuje podmnožico primerkov, ki tvorijo izpeljavo grafa G. Ta možnost obstaja iz sledečih razlogov:

- Če se izpeljava vhodnega grafa G resnično prične s produkcijskim primerkom pi_0, potem izpeljava po definiciji relacije $\mathbf{conseqOf}^+$ vsebuje tudi vse primerke pi z lastnostjo pi $\mathbf{conseqOf}^+$ pi_0.

- Uporaba produkcijskega primerka pi v izpeljavi doda elemente $Xrhs[pi]$ v trenutni graf. Graf G, ki je cilj izpeljave, potemtakem vse svoje elemente pridobi iz množic $Xrhs$ produkcijskih primerkov, ki nastopajo v izpeljavi. Če množica $Xrhs[pi_0]$ in množice $Xrhs[pi]$ za primerke pi z lastnostjo pi $\mathbf{conseqOf}^+$ pi_0 skupaj pokrivajo vse elemente grafa G, potem obstaja možnost, da neko zaporedje teh primerkov tvori izpeljavo grafa G.

Izboljšavo 5 realiziramo na sledeči način: Kadarkoli faza 1 najde primerek ene od začetnih produkcij, preverimo, ali ta primerek izpolnjuje pogoj (3.12). Če to drži,

takoj poskusimo izvršiti fazo 2 na trenutni množici zgrajenih produkcijskih primerkov. Če faza 2 odkrije izpeljavo, je postopek sintaksne analize uspešno končan, sicer pa nadaljujemo s fazo 1.

Opisana izboljšava nam lahko koristi le v primeru, če vhodni graf pripada jeziku gramatike. V nasprotnem primeru pa se zaradi sprotnega preverjanja pogoja (3.12) in morebitnih neuspešnih poskusov izpeljav čas sintaksne analize celo poveča.

3.5 Eksperimentalni rezultati

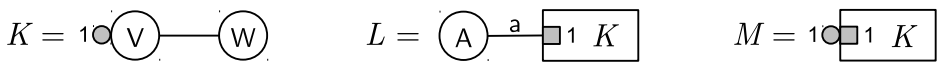
V tem podpoglavju bomo eksperimentalno ovrednotili izhodiščni Rekers-Schürrov sintaksni analizator in naše izboljšave. Ker so vse izboljšave razen izboljšave 1 namenjene povečanju učinkovitosti sintaksnega analizatorja, se bomo osredotočili na količino dela, ki ga sintaksni analizator opravi pri različnih vhodnih podatkih in različnih izboljšavah. Količino dela bomo merili s porabo časa in s številom produkcijskih primerkov, ki jih sintaksni analizator ustvari tekom faze 1.

V razdelku 3.5.1 bomo opisali poskuse, ki smo jih izvedli. V razdelku 3.5.2 bomo rezultate poskusov predstavili, v razdelku 3.5.3 pa jih bomo komentirali.

3.5.1 Izvedba poskusov

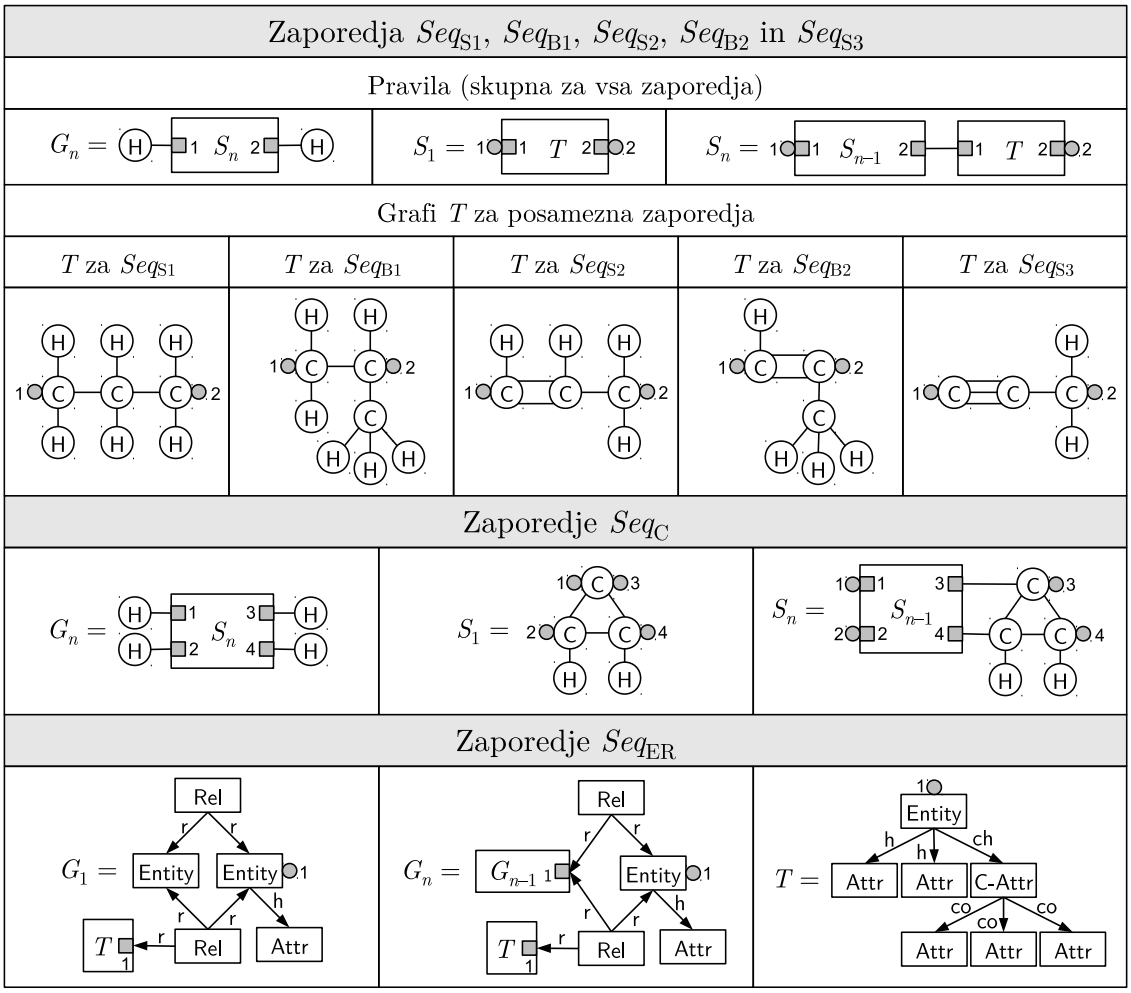
Sintaksnemu analizatorju moramo na vhodu podati tako grafno gramatiko kot graf. Preizkusili smo vseh pet testnih gramatik, ki smo jih predstavili v podpoglavju 3.2 in prikazali na slikah 3.1 in 3.2. Za vsako gramatiko smo izdelali najmanj po eno zaporedje linearno povečujočih se grafov, ki pripadajo njenemu jeziku. Skupno smo izdelali 9 takšnih zaporedij.

Testna zaporedja grafov so formalno definirana na slikah 3.13 in 3.14. Vsako zaporedje je sestavljeno iz grafov G_1, G_2, ...; oznaka G_n tako predstavlja n-ti graf zaporedja. Pri definiciji posameznih zaporedij si pomagamo s pomožnimi grafi (ti so povsod na slikah 3.13 in 3.14 označeni s *poševno pisavo*) ter s simboli oblike ◯ in □. Simbol oblike ◯ predstavlja definicijo *vtičnice*. Vtičnica je oštevilčeno mesto v grafu, na katero lahko preko ene ali več povezav priključimo nek drug graf ali del grafa. Vtičnico vedno priredimo nekemu vozlišču, saj lahko le nanj pripnemo povezavo. Simbol oblike □ pa predstavlja uporabo vtičnice. Za lažje razumevanje vtičnic si oglejmo sledeči primer:



Na gornji sliki je graf K sestavljen iz vozlišč z oznakama V in W ter njune medsebojne povezave. Vozlišču V smo priredili vtičnico s številko 1. Graf L je sestavljen iz vozlišča z oznako A, povezave z oznako a ter elementov grafa K, pri čemer je povezava a vezana na vozlišče z oznako V, ki mu je v grafu K prirejena vtičnica s številko 1. Graf M je kopija grafa K. Vtičnica s številko 1 v grafu M sovpada z vtičnico s številko 1 v grafu K.

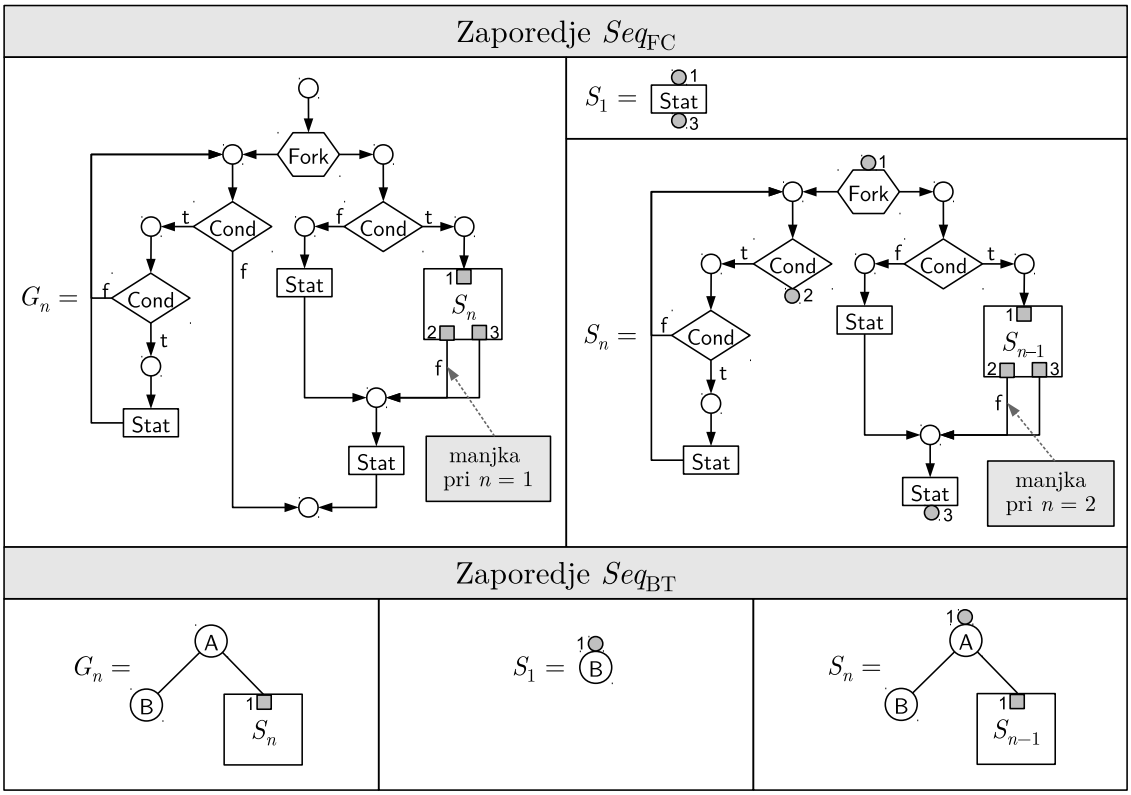
Za lažjo predstavo so na slikah 3.15, 3.16 in 3.17 prikazani prvi trije členi (grafi G_1, G_2 in G_3) vsakega testnega zaporedja. Vidimo, da se število elementov v vseh



Slika 3.13: *Testna zaporedja grafov za gramatike GG_{AHC}, GG_{HC} in GG_{ER}.*

zaporedjih linearno povečuje. Zaporedja Seq_{S1}, Seq_{B1}, Seq_{S2}, Seq_{B2} in Seq_{S3} so sestavljena iz grafov, ki pripadajo jezikoma gramatik GG_{AHC} in GG_{HC}, saj gre za strukturne formule acikličnih ogljikovodikov. V oznakah zaporedij se številke 1, 2 in 3 nanašajo na maksimalno kratnost vezi v spojinah (enojne, dvojne ali trojne vezi). S črko S so označena zaporedja linearnih (nerazvejanih) ogljikovodikov (angl. *linear* ali *straight-chain hydrocarbons*), torej ogljikovodikov z ravno verigo ogljikovih atomov, s črko B pa smo označili zaporedja ogljikovodikov z razvejano verigo ogljikovih atomov (angl. *branched hydrocarbons*). Zaporedje Seq_C je sestavljeno iz strukturnih formul cikličnih ogljikovodikov, zato njegovi grafi pripadajo zgolj jeziku gramatike GG_{HC}. Grafi iz zaporedij Seq_{ER}, Seq_{FC} in Seq_{BT} po vrsti pripadajo jezikom gramatik GG_{ER}, GG_{FC} in GG_{BT}.

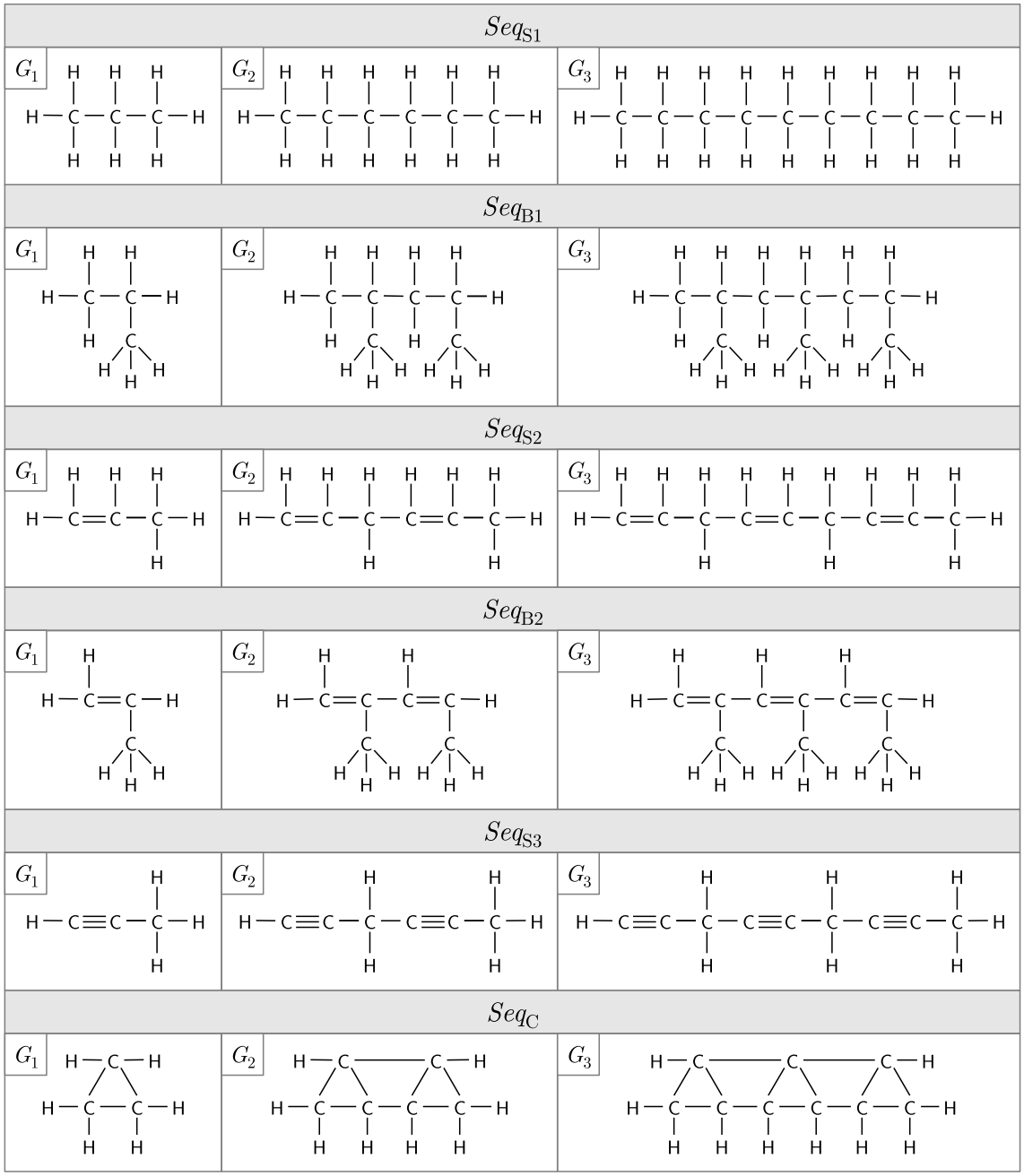
Sintaksni analizator in njegove izboljšave smo preizkušali s skupno 14 pari grafnih gramatik in zaporedij grafov. Vsako izmed zaporedij Seq_{S1}, Seq_{B1}, Seq_{S2}, Seq_{B2} in Seq_{S3} smo sintaksno analizirali v kombinaciji z gramatikama GG_{AHC} in GG_{HC}. Zaporedje Seq_C smo sintaksno analizirali z gramatiko GG_{HC}, zaporedje Seq_{ER} z gramatiko GG_{ER}, zaporedje Seq_{FC} z gramatiko GG_{FC}, zaporedje Seq_{BT} pa v kombinaciji z gramatiko GG_{BT}. Pri vsaki dvojici gramatike in zaporedja smo sintaksno



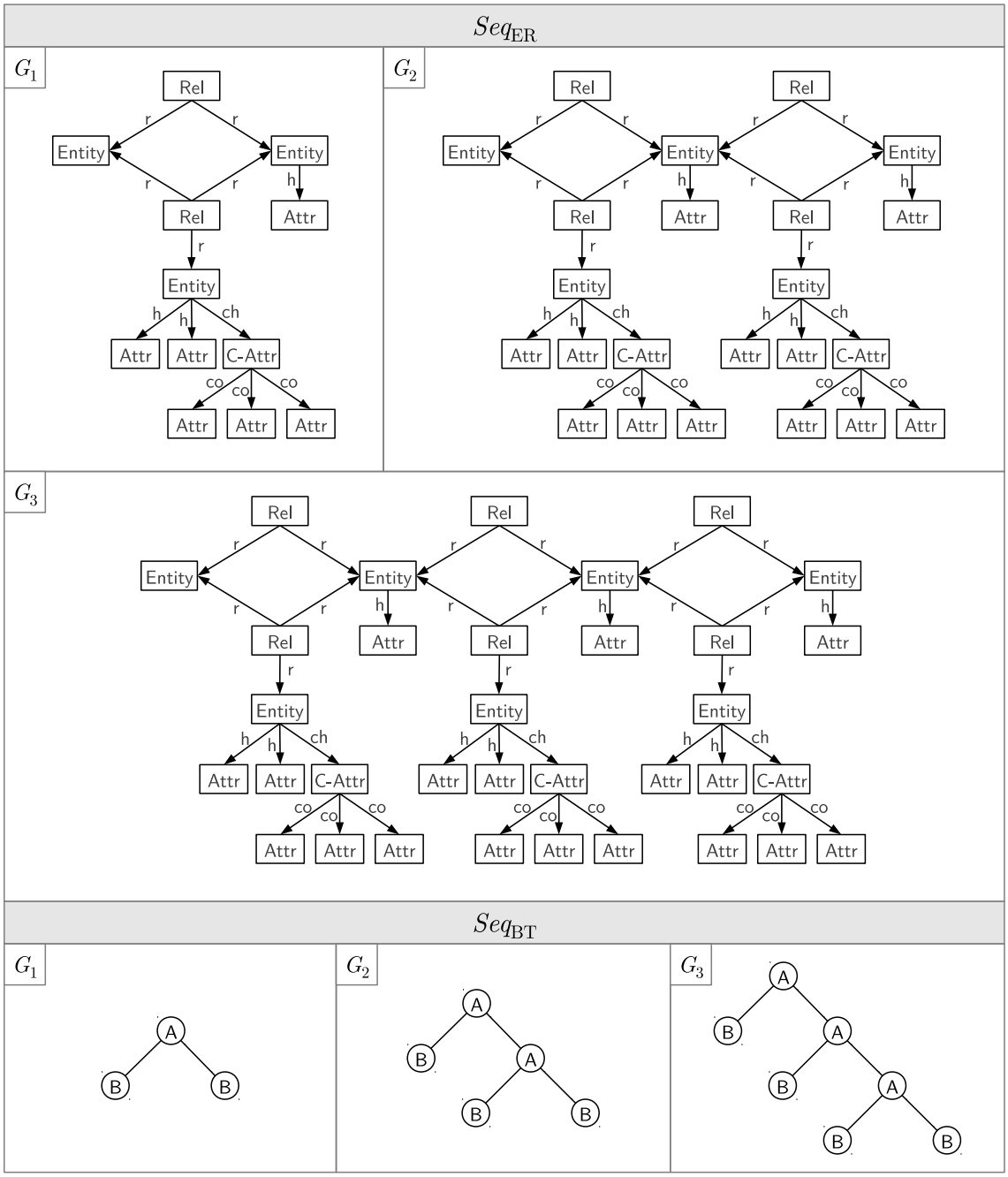
Slika 3.14: *Testna zaporedja grafov za gramatike GG_{FC} in GG_{BT}.*

analizirali grafe od G_1 do G_{10}, nato pa vsak deseti graf do vključno G_{100} (skupaj 19 grafov). Za vsako dvojico grafne gramatike in grafa smo sintaksno analizo izvedli po 5-krat; v i-ti izvedbi (za $i \in 1\,..\,5$) smo uporabili Rekers-Schürrov sintaksni analizator z vključenimi izboljšavami od 1 do vključno i. Izboljšava 1, ki ne vpliva na učinkovitost sintaksne analize, pač pa zgolj omogoča obravnavo produkcij z nepovezanimi desnimi stranmi, je bila torej vključena v vseh izvajanjih sintaksnega analizatorja.

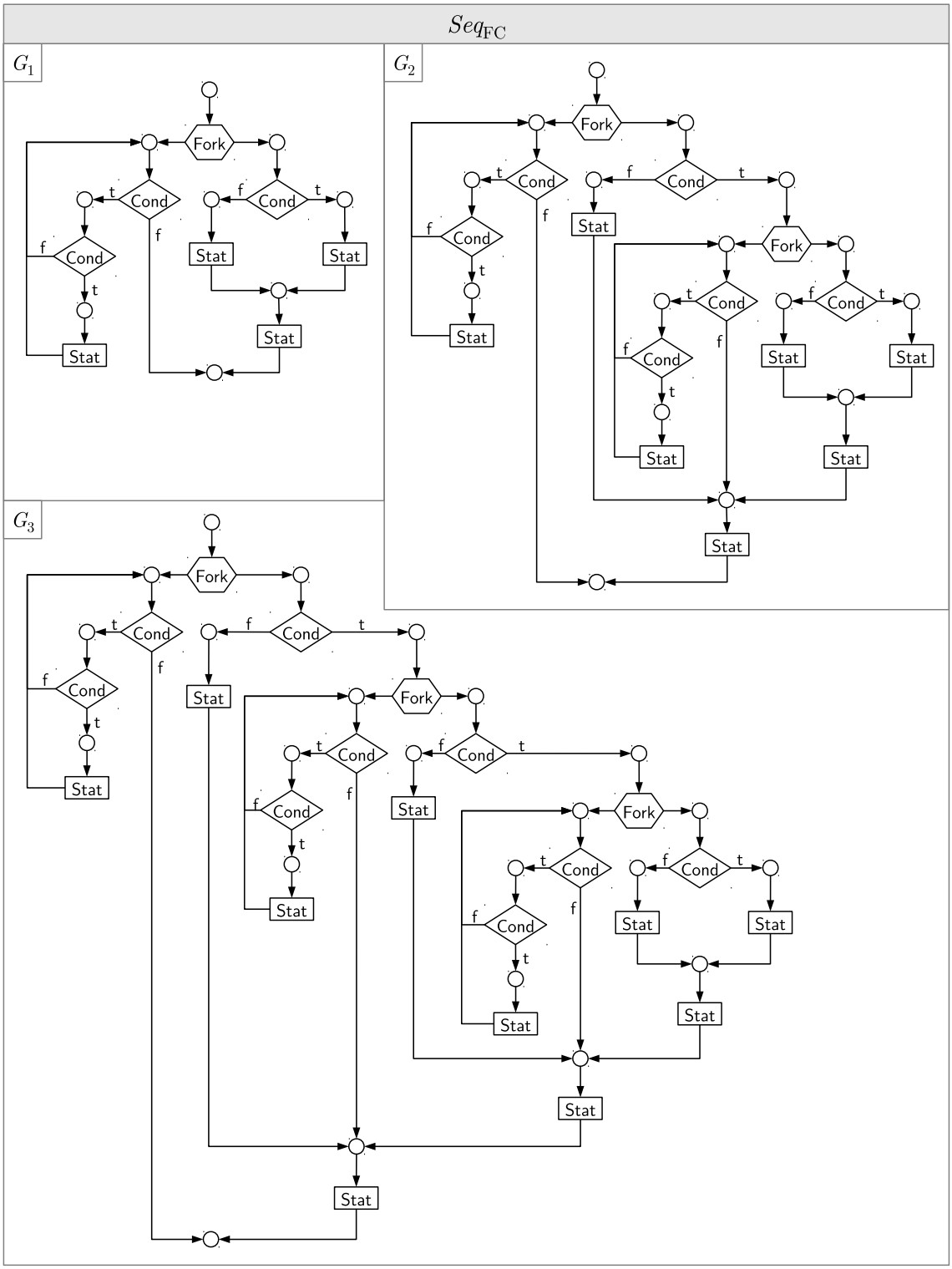
Naj beseda »poskus« označuje enkratno sintaksno analizo določenega grafa iz določenega zaporedja pri podani gramatiki in pri podani množici izboljšav. Na primer, v enem od poskusov smo sintaksnemu analizatorju na vhodu podali graf G_{30} iz zaporedja Seq_{B2} in gramatiko GG_{HC}, pri tem pa smo uporabili izboljšave 1, 2 in 3. Pri vsakem poskusu sta nas zanimala celoten čas izvajanja sintaksne analize (faza 1 in faza 2 skupaj) in število ustvarjenih produkcijskih primerkov v fazi 1. To število je dokaj dober pokazatelj količine dela, ki ga opravi sintaksni analizator. Poskuse smo izvajali na računalniku s 3,40-gigaherčnim osemjedrnim procesorjem Intel Core i7. Vsak poskus smo časovno omejili na 10^4 sekund (približno 2h 47min), pomnilniško pa na 4 GiB. Čim je sintaksni analizator presegel časovno ali prostorsko omejitev, smo ga prekinili. Če je sintaksni analizator pri neki kombinaciji množice izboljšav, grafne gramatike in zaporedja grafov presegel časovno ali prostorsko omejitev pri sintaksni analizi grafa G_i v tem zaporedju, potem poskusov za grafe G_j za $j > i$ pri isti kombinaciji množice izboljšav, gramatike in zaporedja sploh nismo izvajali, saj bi gotovo presegli časovno-prostorske omejitve.



Slika 3.15: *Prvi trije grafi v zaporedjih* Seq_{S1}, Seq_{B1}, Seq_{S2}, Seq_{B2}, Seq_{S3} *in* Seq_{C}.



Slika 3.16: *Prvi trije grafi v zaporedjih* Seq_{ER} *in* Seq_{BT}.



Slika 3.17: *Prvi trije grafi v zaporedju* Seq_{FC}.

Da bi lahko čas karseda verodostojno ocenili, smo krajše poskuse večkrat ponovili in izračunali povprečno trajanje poskusa. Vsak poskus, ki je za svoje izvajanje porabil $t < 20$ sekund časa, smo ponovili po $\lceil 20/t \rceil$-krat, kjer je $\lceil x \rceil$ najmanjše celo število m z lastnostjo $m \geq x$. Poskuse, ki so trajali od 20 do 200 sekund časa, smo ponovili po dvakrat, poskusov, ki so potrebovali več kot 200 sekund časa, pa nismo ponavljali.

3.5.2 Rezultati poskusov

Tabele 3.6–3.12 prikazujejo podatke o povprečnem trajanju posameznih poskusov v sekundah. Zaradi prihranka prostora smo v isti tabeli združili po dva para grafne gramatike in zaporedja grafov. Posamezne vrstice tabel se nanašajo na posamezne grafe v zaporedjih, stolpci pa na vključene izboljšave. Stolpec $1\,..\,i$ (pri $i \in 1\,..\,5$) tako predstavlja poskus, pri katerem smo vključili izboljšave od 1 do vključno i. Na primer, iz spodnje polovice tabele 3.9 lahko razberemo, da je sintaksni analizator z vključenimi izboljšavami 1, 2 in 3 za sintaksno analizo 6. člena zaporedja Seq_{S2} pri podani gramatiki GG_{HC} potreboval 0,384 sekunde, z vključenimi izboljšavami 1, 2, 3 in 4 pa 0,401 sekunde. Znak $\times$ v tabeli predstavlja poskuse, ki so presegli časovno ali prostorsko omejitev, in poskuse, ki jih nismo izvedli, ker je omejitve presegel že nek poskus za manjši graf pri isti kombinaciji množice izboljšav, gramatike in zaporedja grafov.

Ker čas ni edini možni pokazatelj količine dela, ki ga pri posameznih poskusih opravi sintaksni analizator, smo beležili tudi število vseh produkcijskih primerkov, ustvarjenih v fazi 1, pri čemer smo upoštevali tudi nekonsistentne primerke, ki jih faza 1 sproti odstranjuje. Rezultati za posamezne kombinacije množice izboljšav, gramatike in zaporedja grafov so prikazani v tabeli 3.13 kot funkcije številke člena. Stolpec $1\,..\,i$ tudi tokrat predstavlja sintaksno analizo z vključenimi izboljšavami od 1 do vključno i. Na primer, pri kombinaciji množice izboljšav 1, 2, 3 in 4, gramatike GG_{AHC} in zaporedja Seq_{S3} je sintaksni analizator pri obdelavi grafa G_n ustvaril $(409n - 20)$ produkcijskih primerkov. Ker izboljšava 3 ne vpliva na število odkritih produkcijskih primerkov, smo stolpec $1\,..\,3$ izpustili, saj bi bila njegova vsebina povsod enaka vsebini stolpca $1\,..\,2$. Prvi stolpec podaja dolžino izpeljave (število produkcijskih primerkov, ki tvorijo izpeljavo). Razlika med številom ustvarjenih produkcijskih primerkov in dolžino izpeljave je mera za odvečno delo, ki ga v fazi 1 opravi sintaksni analizator, saj bi v idealnem primeru sintaksni analizator tvoril natanko toliko produkcijskih primerkov, kot znaša dolžina izpeljave.

Pri kombinaciji izhodiščnega Rekers-Schürrovega sintaksnega analizatorja (oziroma sintaksnega analizatorja z izboljšavo 1, ki ne vpliva na učinkovitost delovanja), gramatike GG_{AHC} in zaporedij grafov, ki pripadajo jeziku te gramatike, je težko oceniti rast števila produkcijskih primerkov glede na število elementov vhodnega grafa, saj se nobeden od poskusov ni izvršil do konca. Občutek o rasti pa lahko pridobimo z razmislekom, ki ga predstavljamo v nadaljevanju. Zaradi enostavnosti se osredotočimo na graf linearnega alkana z m ogljikovimi atomi, torej grafa, sestavljenega iz verige m vozlišč C, povezanih s samimi enojnimi povezavami, in $2m + 2$ vozlišč H. Zaradi lažjega sklicevanja naj bodo vozlišča C v tej verigi oštevilčena kot C_1, ..., C_m. Vozlišči C_1 in C_m sta povezani s po enim vozliščem C in tremi vozlišči H, vsako

Tabela 3.6: *Trajanje sintaksne analize (v sekundah) [1/7]. Stolpec* $1 \ldots i$ *predstavlja vključene izboljšave od 1 do vključno* i.

AHC + S1	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	×	0,193	0,192	0,178	0,177
	2	×	0,210	0,210	0,203	0,209
	3	×	0,245	0,246	0,235	0,240
	4	×	0,273	0,274	0,260	0,260
	5	×	0,285	0,281	0,282	0,279
	6	×	0,294	0,294	0,302	0,286
	7	×	0,322	0,323	0,314	0,306
	8	×	0,338	0,338	0,328	0,336
	9	×	0,361	0,361	0,339	0,355
	10	×	0,378	0,375	0,357	0,368
	20	×	0,636	0,636	0,552	0,492
	30	×	0,988	0,988	0,801	0,685
	40	×	1,60	1,58	1,20	1,01
	50	×	2,48	2,45	1,76	1,48
	60	×	3,79	3,70	2,52	2,18
	70	×	5,37	5,26	3,59	3,18
	80	×	7,56	7,52	4,89	4,40
	90	×	10,7	10,6	6,85	6,38
	100	×	15,0	14,8	9,75	9,20
AHC + B1	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	×	0,183	0,182	0,175	0,180
	2	×	0,218	0,210	0,198	0,207
	3	×	0,256	0,248	0,235	0,225
	4	×	0,284	0,279	0,272	0,228
	5	×	0,290	0,294	0,290	0,245
	6	×	0,309	0,313	0,311	0,272
	7	×	0,331	0,328	0,321	0,283
	8	×	0,348	0,346	0,337	0,302
	9	×	0,370	0,374	0,355	0,308
	10	×	0,387	0,385	0,376	0,322
	20	×	0,655	0,658	0,595	0,415
	30	×	1,05	1,00	0,885	0,498
	40	×	1,65	1,67	1,31	0,587
	50	×	2,64	2,63	1,91	0,738
	60	×	4,01	3,95	2,78	0,924
	70	×	5,82	5,67	3,86	1,13
	80	×	8,08	8,10	5,27	1,43
	90	×	11,5	11,4	7,54	1,71
	100	×	16,7	16,5	10,5	2,15

Tabela 3.7: *Trajanje sintaksne analize (v sekundah) [2/7].*

AHC + S2	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	×	0,192	0,191	0,193	0,183
	2	×	0,236	0,236	0,212	0,224
	3	×	0,273	0,272	0,253	0,252
	4	×	0,293	0,292	0,283	0,266
	5	×	0,305	0,304	0,294	0,295
	6	×	0,330	0,332	0,300	0,317
	7	×	0,348	0,349	0,334	0,329
	8	×	0,361	0,361	0,343	0,339
	9	×	0,385	0,384	0,368	0,365
	10	×	0,409	0,410	0,375	0,398
	20	×	0,737	0,748	0,576	0,512
	30	×	1,12	1,12	0,864	0,732
	40	×	1,97	1,97	1,30	1,09
	50	×	2,91	2,91	1,93	1,62
	60	×	4,33	4,40	2,81	2,42
	70	×	6,37	6,41	3,95	3,55
	80	×	9,13	9,22	5,54	5,04
	90	×	13,2	13,3	8,03	7,53
	100	×	18,7	18,7	11,5	10,9
AHC + B2	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	×	0,192	0,192	0,193	0,185
	2	×	0,236	0,235	0,212	0,219
	3	×	0,272	0,272	0,252	0,228
	4	×	0,291	0,291	0,283	0,256
	5	×	0,308	0,308	0,294	0,274
	6	×	0,331	0,329	0,299	0,285
	7	×	0,347	0,349	0,333	0,291
	8	×	0,365	0,363	0,341	0,312
	9	×	0,383	0,381	0,363	0,318
	10	×	0,406	0,406	0,371	0,341
	20	×	0,753	0,749	0,567	0,414
	30	×	1,12	1,12	0,860	0,502
	40	×	1,97	1,94	1,29	0,618
	50	×	3,04	2,98	1,96	0,764
	60	×	4,52	4,41	2,84	0,943
	70	×	6,54	6,40	4,03	1,19
	80	×	9,42	9,20	5,65	1,66
	90	×	13,8	13,3	8,15	2,01
	100	×	18,9	18,8	11,8	2,51

Tabela 3.8: *Trajanje sintaksne analize (v sekundah) [3/7].*

AHC + S3	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	×	0,314	0,315	0,301	0,300
	2	×	0,382	0,381	0,366	0,369
	3	×	0,446	0,440	0,416	0,434
	4	×	0,495	0,493	0,456	0,462
	5	×	0,520	0,528	0,500	0,500
	6	×	0,552	0,555	0,538	0,541
	7	×	0,616	0,616	0,604	0,613
	8	×	0,660	0,654	0,682	0,631
	9	×	0,692	0,697	0,648	0,644
	10	×	0,751	0,760	0,684	0,683
	20	×	1,72	1,75	1,30	1,24
	30	×	3,71	3,84	2,40	2,24
	40	×	6,84	7,16	4,19	3,98
	50	×	11,9	12,5	6,80	6,60
	60	×	19,0	20,0	10,7	10,4
	70	×	30,1	31,6	17,0	16,6
	80	×	45,1	47,2	25,5	25,3
	90	×	65,1	67,9	37,2	36,4
	100	×	90,6	94,1	51,8	50,9
HC + S1	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	0,870	0,179	0,179	0,179	0,181
	2	68,1	0,235	0,234	0,241	0,241
	3	1600	0,286	0,286	0,297	0,298
	4	×	0,332	0,324	0,345	0,346
	5	×	0,378	0,364	0,384	0,383
	6	×	0,426	0,394	0,408	0,411
	7	×	0,505	0,416	0,442	0,443
	8	×	0,630	0,449	0,468	0,475
	9	×	1,02	0,655	0,700	0,675
	10	×	1,17	0,788	0,854	0,797
	20	×	102	2,71	2,77	2,80
	30	×	1990	23,3	23,5	23,5
	40	×	×	144	145	145
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×

Tabela 3.9: *Trajanje sintaksne analize (v sekundah) [4/7].*

HC + B1	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	0,865	0,179	0,180	0,180	0,180
	2	109	0,236	0,235	0,240	0,241
	3	2640	0,288	0,285	0,297	0,297
	4	×	0,335	0,322	0,345	0,345
	5	×	0,377	0,363	0,381	0,382
	6	×	0,421	0,388	0,403	0,408
	7	×	0,501	0,410	0,438	0,439
	8	×	0,625	0,442	0,463	0,470
	9	×	1,01	0,657	0,696	0,672
	10	×	1,17	0,781	0,851	0,792
	20	×	102	2,70	2,77	2,79
	30	×	1990	23,5	23,2	23,5
	40	×	×	142	144	145
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×
HC + S2	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	2,95	0,178	0,177	0,178	0,180
	2	621	0,233	0,233	0,238	0,237
	3	×	0,286	0,282	0,295	0,295
	4	×	0,329	0,321	0,338	0,343
	5	×	0,368	0,363	0,371	0,372
	6	×	0,419	0,384	0,401	0,405
	7	×	0,501	0,414	0,435	0,438
	8	×	0,614	0,441	0,459	0,461
	9	×	1,00	0,655	0,693	0,669
	10	×	1,17	0,777	0,850	0,790
	20	×	99,8	2,68	2,75	2,76
	30	×	1960	23,4	23,5	23,4
	40	×	×	143	144	144
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×

Tabela 3.10: *Trajanje sintaksne analize (v sekundah) [5/7].*

HC + B2	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	2,95	0,178	0,179	0,180	0,179
	2	1200	0,234	0,233	0,237	0,238
	3	×	0,285	0,285	0,294	0,294
	4	×	0,330	0,323	0,341	0,342
	5	×	0,366	0,359	0,373	0,375
	6	×	0,416	0,385	0,402	0,403
	7	×	0,500	0,414	0,433	0,440
	8	×	0,612	0,438	0,460	0,465
	9	×	1,00	0,650	0,694	0,674
	10	×	1,16	0,775	0,840	0,808
	20	×	99,8	2,68	2,76	2,78
	30	×	1970	23,1	23,2	23,6
	40	×	×	144	144	144
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×
HC + S3	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	25,8	0,177	0,177	0,179	0,178
	2	5740	0,230	0,230	0,235	0,235
	3	×	0,280	0,280	0,291	0,292
	4	×	0,324	0,320	0,337	0,339
	5	×	0,368	0,362	0,372	0,372
	6	×	0,409	0,383	0,398	0,399
	7	×	0,492	0,412	0,435	0,436
	8	×	0,601	0,435	0,454	0,459
	9	×	0,994	0,645	0,689	0,668
	10	×	1,12	0,770	0,834	0,800
	20	×	97,8	2,65	2,73	2,73
	30	×	1940	23,2	23,2	23,1
	40	×	×	142	142	142
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×

Tabela 3.11: *Trajanje sintaksne analize (v sekundah) [6/7].*

HC + C	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	1,80	0,177	0,177	0,178	0,179
	2	1040	0,232	0,232	0,236	0,236
	3	×	0,283	0,280	0,293	0,293
	4	×	0,326	0,323	0,335	0,335
	5	×	0,367	0,364	0,371	0,375
	6	×	0,413	0,386	0,401	0,403
	7	×	0,496	0,417	0,437	0,439
	8	×	0,606	0,446	0,469	0,472
	9	×	0,988	0,642	0,692	0,711
	10	×	1,12	0,776	0,837	0,802
	20	×	97,9	2,66	2,74	2,74
	30	×	1940	23,1	23,1	23,3
	40	×	×	143	143	144
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×
ER + ER	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	0,174	0,166	0,166	0,166	0,166
	2	0,189	0,182	0,181	0,182	0,181
	3	0,210	0,201	0,211	0,213	0,211
	4	0,240	0,219	0,216	0,220	0,224
	5	0,254	0,241	0,243	0,244	0,244
	6	0,269	0,248	0,255	0,256	0,257
	7	0,286	0,264	0,264	0,267	0,264
	8	0,307	0,273	0,267	0,279	0,279
	9	0,330	0,292	0,286	0,289	0,289
	10	0,354	0,308	0,299	0,304	0,302
	20	0,739	0,534	0,448	0,451	0,453
	30	1,67	0,943	0,647	0,664	0,664
	40	3,92	1,75	1,02	1,04	1,05
	50	9,44	3,18	1,64	1,66	1,67
	60	19,1	6,50	3,04	3,06	3,05
	70	34,9	11,8	5,25	5,24	5,26
	80	58,5	19,8	8,42	8,43	8,46
	90	92,2	31,2	13,0	13,0	13,0
	100	139	47,0	19,4	19,5	19,4

Tabela 3.12: *Trajanje sintaksne analize (v sekundah) [7/7].*

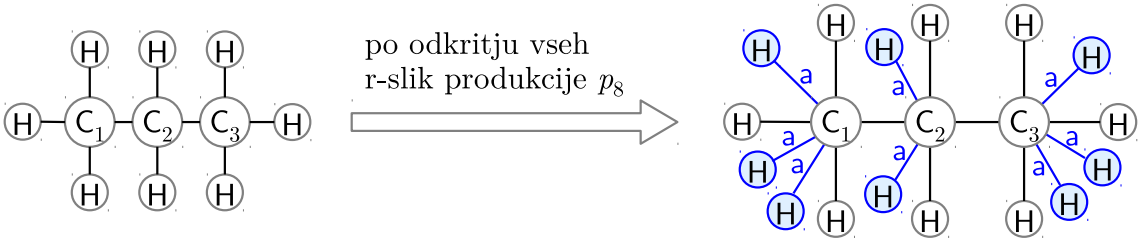
FC + FC	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	0,190	0,190	0,190	0,191	0,199
	2	0,266	0,236	0,236	0,247	0,233
	3	0,383	0,290	0,289	0,293	0,295
	4	0,741	0,377	0,377	0,383	0,345
	5	1,58	0,463	0,462	0,463	0,445
	6	3,86	0,585	0,585	0,584	0,556
	7	27,7	0,776	0,759	0,728	0,709
	8	303	0,997	1,01	0,945	0,910
	9	3290	1,22	1,25	1,22	1,10
	10	×	1,45	1,45	1,32	1,30
	20	×	25,8	25,6	25,6	22,9
	30	×	388	388	391	357
	40	×	2660	2660	2660	2540
	50	×	×	×	×	×
	60	×	×	×	×	×
	70	×	×	×	×	×
	80	×	×	×	×	×
	90	×	×	×	×	×
	100	×	×	×	×	×
BT + BT	člen	1 . . 1	1 . . 2	1 . . 3	1 . . 4	1 . . 5
	1	0,150	0,149	0,150	0,151	0,149
	2	0,157	0,156	0,156	0,155	0,153
	3	0,190	0,178	0,176	0,166	0,157
	4	0,223	0,209	0,209	0,187	0,161
	5	0,320	0,272	0,273	0,240	0,166
	6	0,647	0,471	0,470	0,375	0,170
	7	1,88	1,11	1,10	0,596	0,188
	8	17,0	8,11	8,05	2,02	0,190
	9	345	142	145	17,8	0,186
	10	×	4580	4290	502	0,202
	20	×	×	×	×	0,252
	30	×	×	×	×	0,307
	40	×	×	×	×	0,384
	50	×	×	×	×	0,415
	60	×	×	×	×	0,480
	70	×	×	×	×	0,561
	80	×	×	×	×	0,643
	90	×	×	×	×	0,728
	100	×	×	×	×	0,827

Tabela 3.13: *Dolžina izpeljave in število produkcijskih primerkov v odvisnosti od dolžine zaporedja (n).*

GG + *Seq*	dolžina izpeljave	1..1	1..2	1..4	1..5
AHC + S1	$12n+1$	(gl. besedilo)	$36n-7$	$18n-1$	$18n-2$
AHC + B1	$12n+1$	(gl. besedilo)	$36n-7$	$18n-1$	$12n+9$
AHC + S2	$11n+1$	(gl. besedilo)	$53n-7$	$23n-1$	$23n-2$
AHC + B2	$11n+1$	(gl. besedilo)	$53n-7$	$23n-1$	$15n+13$
AHC + S3	$10n+1$	(gl. besedilo)	$655n-38$	$409n-20$	$409n-21$
HC + S1	$12n+1$	$3240n^2-1428n$	$9n^2+9n+1$	$9n^2+9n+1$	$9n^2+9n+1$
HC + B1	$12n+1$	$3960n^2-2148n$	$9n^2+9n+1$	$9n^2+9n+1$	$9n^2+9n+1$
HC + S2	$11n+1$	$6120n^2-2148n$	$9n^2+8n+1$	$9n^2+8n+1$	$9n^2+8n+1$
HC + B2	$11n+1$	$7920n^2-3948n$	$9n^2+8n+1$	$9n^2+8n+1$	$9n^2+8n+1$
HC + S3	$10n+1$	$11160n^2-3228n$	$9n^2+7n+1$	$9n^2+7n+1$	$9n^2+7n+1$
HC + C	$10n+2$	$10080n^2-7908n$	$9n^2+7n+2$	$9n^2+7n+2$	$9n^2+7n+2$
ER + ER	$9n+1$	$41n+1$	$31n+1$	$31n+1$	$31n+1$
FC + FC	$11n+2$	$21\cdot 2^n+\Theta(n^2)$	$6n^2+21n+2$	$6n^2+20n+2$	$6n^2+15n+5$
BT + BT	$2n+1$	(tabela 3.14)	(tabela 3.14)	(tabela 3.14)	$5n-1$

izmed vozlišč C_2, ..., C_{m-1} pa je povezano z dvema vozliščema C in dvema vozliščema H. Vse povezave C—H so izomorfne z desno stranjo produkcije p_8 (slika 3.1 na strani 53), zato bo sintaksni analizator prej ali slej na vsako vozlišče C s povezavami a pripel po tri (v primeru vozlišč 1 in m) oziroma po dve (v primeru vseh ostalih vozlišč C) vozlišči H (gl. sliko 3.18 za primer strukturne formule propana). Po tej operaciji je podgraf, sestavljen iz vozlišča C_2 ter vozlišča C_1 in njegovih treh sosedov $\overset{a}{—}\mathsf{H}$, izomorfen desni strani produkcije p_2. Ker je možnih 6 (= 3!) različnih preslikav med vozlišči H na desni strani produkcije p_2 in vozlišči v podgrafu in ker izboljšava 2 ni vključena, bo r-slika produkcije p_2 odkrita 6-krat. Po vsakem odkritju bo faza 1 s povezavo a pripela po eno novo vozlišče H na vozlišče C_2. Vozlišče C_2 je sedaj preko povezav a povezano z osmimi vozlišči H. Sedaj se osredotočimo na podgraf, sestavljen iz vozlišča C_3 ter vozlišča C_2 in njegovih osmih povezav $\overset{a}{—}\mathsf{H}$. V tem podgrafu je desno stran produkcije p_2 možno odkriti kar $\binom{8}{3}3! = 336$-krat (po enkrat za vsako podmnožico treh vozlišč H izmed osmih in za vsako permutacijo izbranih treh vozlišč). Kot rezultat teh odkritij bo sintaksni analizator na vozlišče C_3 preko povezav a pripel 336 vozlišč H. Če nadaljujemo s potovanjem po verigi vozlišč C, ugotovimo, da bo postopek na opisani način na vozlišče C_4 pripel $\binom{336+2}{3}3! \approx 38\cdot 10^6$ vozlišč H, na vozlišče C_5 že $5{,}6\cdot 10^{22}$ vozlišč H itd. K opisanemu vrtoglavemu naraščanju števila odkritih produkcijskih primerkov pa moramo prišteti še primerke produkcije p_1 in produkcijske primerke, ki nastajajo v smeri C_m, ..., C_1. Oboji se množijo eksponentno. Vidimo torej, da izhodiščni Rekers-Schürrov sintaksni analizator zaradi kombinatorične eksplozije odpove že pri zelo majhnih grafih ogljikovodikov. Iz tabel 3.6–3.8 je razvidno, da je izboljšava 2 nujna za dosego obvladljivosti, vendar

pa bomo več o tem povedali v razdelku 3.5.3.



Slika 3.18: *Pričetek sintaksne analize strukturne formule propana pri podani gramatiki* GG_{AHC}.

Tudi pri gramatiki GG_{HC} je izhodiščni sintaksni analizator že pri tretjem ali četrtem členu presegel dodeljeni časovno-prostorski okvir, vendar pa je naraščanje števila produkcijskih primerkov tokrat kvadratno (četudi z veliko konstanto), ne pa eksponentno kot pri gramatiki GG_{AHC}. Funkcijo naraščanja števila primerkov z dolžino zaporedja lahko izračunamo in zapišemo v eksplicitni obliki. Prikazali bomo izračun funkcije za kombinacijo $GG_{\mathrm{HC}} + Seq_{\mathrm{S1}}$ $(3240n^2 - 1428n)$; za ostala zaporedja grafov je izračun podoben. Predstavljajmo si graf linearnega alkana z $m = 3n$ vozlišči C in $2m + 2$ vozlišči H, pri čemer naj bodo vozlišča C tako kot pri razlagi v prejšnjem odstavku oštevilčena kot $\mathsf{C}_1, \ldots, \mathsf{C}_m$. Sintaksni analizator bo odkril $2m + 2$ r-slik produkcije p_4, po eno za vsako vozlišče H. Ker bo vsako r-sliko odkril le po enkrat, bo ustvaril m primerkov produkcije p_4, na vsako vozlišče C pa bo preko povezave a pripel po eno vozlišče H. Nastali graf vsebuje $m - 1$ r-slik produkcije p_3 (po eno za vsako povezavo C_i–C_{i+1}), vendar pa bo vsaka r-slika odkrita dvakrat, saj je desno stran produkcije p_3 na podgraf C_i–C_{i+1} možno homomorfno preslikati na dva načina (prvo vozlišče produkcije na vozlišče C_i in drugo na vozlišče C_{i+1} ali pa obratno). Kot posledica teh odkritij nastane $2(m - 1)$ produkcijskih primerkov. Na vozlišči C_1 in C_m je sedaj preko povezav a pripetih po 5 vozlišč H (tri ob izdelavi primerkov produkcij p_4 in dve po dvakratnem odkritju r-slike produkcije p_3 v podgrafih C_1–C_2 oz. C_{m-1}–C_m), na ostala vozlišča pa po 6 vozlišč H (dve po izdelavi primerkov produkcij p_4, dve za dvakratni nastanek primerka produkcije p_3 v podgrafu C_{i-1}–C_i in dve za dvakratni nastanek takšnega primerka v podgrafu C_i–C_{i+1}). Pri odkrivanju r-slik produkcije p_2 bo faza 1 obravnavala vseh $m(m - 1)$ dvojic vozlišč C_i in C_j pri $i \neq j$ (dvojici $(\mathsf{C}_i, \mathsf{C}_j)$ in $(\mathsf{C}_j, \mathsf{C}_i)$ bosta obravnavani kot različni). Pri vsaki od $2(m - 1)$ dvojic $(\mathsf{C}_2, \mathsf{C}_1)$, $(\mathsf{C}_3, \mathsf{C}_1)$, $\ldots$, $(\mathsf{C}_m, \mathsf{C}_1)$ in $(\mathsf{C}_1, \mathsf{C}_m)$, $(\mathsf{C}_2, \mathsf{C}_m)$, $\ldots$, $(\mathsf{C}_{m-1}, \mathsf{C}_m)$ (denimo, da prvi element vsake dvojice predstavlja vozlišče grafa, v katero se preslika vozlišče C iz množice $Common[p_2]$) bo desna stran produkcije p_2 odkrita po $\binom{5}{4}4! = 120$-krat, pri vsaki od preostalih $m(m-1)-2(m-1)$ dvojic pa po $\binom{6}{4}4! = 360$-krat. V skupnem seštevku bo torej faza 1 ustvarila $(360m^2 - 840m + 480)$ primerkov produkcije p_2. Ker velja $Xlhs[p_2] = \emptyset$, se graf pri nobenem nastanku primerka produkcije p_2 ne bo spremenil. Desna stran produkcije p_1 bo odkrita $2\binom{5}{4}4! + (m-2)\binom{6}{4}4! = (360m - 480)$-krat. Skupno število ustvarjenih primerkov vseh produkcij potemtakem znaša $360m^2 - 476m$ oziroma (ob upoštevanju predpostavke $m = 3n$) $3240n^2 - 1428n$.

Oznaka $\Theta(n^2)$ v tabeli 3.13 pri dvojici $GG_{\mathrm{FC}} + Seq_{\mathrm{FC}}$ in izhodiščnem sintaksnem

analizatorju predstavlja funkcijo $3n^2 + 6n - 19$. Za par $GG_{\mathrm{BT}} + Seq_{\mathrm{BT}}$ je rast števila produkcijskih primerkov v odvisnosti od številke člena prikazana v tabeli 3.14. Stolpec $1\,..\,i$ ponovno predstavlja vključene izboljšave od 1 do vključno i.

Tabela 3.14: *Dolžina izpeljave in število produkcijskih primerkov v odvisnosti od dolžine zaporedja (n) za dvojico* $GG_{\mathrm{BT}} + Seq_{\mathrm{BT}}$.

n	dolžina izpeljave	$1\,..\,1$	$1\,..\,2$	$1\,..\,4$	$1\,..\,5$
1	3	6	5	5	4
2	5	22	18	11	9
3	7	66	53	22	14
4	9	202	160	47	19
5	11	666	523	112	24
6	13	2362	1846	305	29
7	15	8826	6881	946	34
8	17	34042	26508	3251	39
9	19	133626	103991	11956	44
10	21	×	411874	45749	49

Poraba časa v fazi 1 asimptotično narašča kot kubična funkcija v odvisnosti od števila izdelanih produkcijskih primerkov. Največ časa nam vzame računanje relacije $\mathbf{above}^{+}_{pi}$, ki jo je treba ovrednotiti za vse produkcijske primerke razen za tiste, ki pripadajo produkcijam, za katere s pomočjo izboljšave 3 ugotovimo, da ne morejo tvoriti nekonsistentnih produkcijskih primerkov. Ker je relacija $\mathbf{above}^{+}_{pi}$ trojiška, moramo pri k produkcijskih primerkih njeno vrednost izračunati za k^3 trojic. Porabo časa v fazi 2 je v splošnem težje oceniti, saj ta faza temelji na sestopanju, katerega učinkovitost je lahko močno odvisna od vrstnega reda obravnave produkcijskih primerkov. Vendar pa je pri vseh preizkušenih dvojicah gramatik in zaporedij faza 1 v splošnem porabila bistveno več časa kot faza 2, zato časovna ocena $O(k^3)$ pri k produkcijskih primerkih vsaj pri naših poskusih ustreza resničnosti.

3.5.3 Komentar rezultatov

Rezultate bomo najprej pokomentirali na splošno (podrazdelek 3.5.3.1), nato pa se bomo lotili posameznih gramatik (podrazdelki 3.5.3.2–3.5.3.4).

3.5.3.1 Splošen pregled

Pogled na tabele 3.6–3.14 nam razkrije, da izboljšava 2 pri vseh dvojicah gramatik in zaporedij znatno izboljša učinkovitost sintaksne analize. Razlog je v tem, da vse gramatike vsebujejo vsaj po eno desno stran z medsebojno zamenljivimi vozlišči ali povezavami. Izboljšava 3 sicer ne vpliva na število ustvarjenih produkcijskih primerkov, vendar pa z njo pri gramatikah GG_{HC} in GG_{ER} vseeno dosežemo občutne časovne prihranke, saj je pri teh gramatikah za več produkcij možno ugotoviti, da

pri sintaksni analizi ne morejo tvoriti nekonsistentnih produkcijskih primerkov. Izboljšava 4 se nam obrestuje predvsem pri gramatiki GG_{BT}, do določene mere pa tudi pri gramatiki GG_{AHC}. Pri vseh poskusih z omenjenima gramatikama namreč nastane veliko takšnih nekonsistentnih produkcijskih primerkov, ki jih je po receptu iz izboljšave 4 možno predčasno odkriti in zatreti. Izboljšava 5 pa doseže ogromne prihranke pri gramatiki GG_{BT}, manjše, a še vedno otipljive, pa pri dvojicah $GG_{\mathrm{AHC}} + Seq_{\mathrm{B1}}$ in $GG_{\mathrm{AHC}} + Seq_{\mathrm{B2}}$.

3.5.3.2 Gramatika GG_{AHC}

Pri poskusih z gramatiko GG_{AHC} je vrednost izboljšave 2 najbolj očitna. Medtem ko neizboljšan sintaksni analizator že pri prvih členih posameznih zaporedij grafov ogljikovodikov podleže kombinatorični eksploziji, smo z izboljšavo 2 dosegli celo *linearno* odvisnost števila ustvarjenih produkcijskih primerkov od velikosti grafa v zaporedju. Na osnovi zadnjega odstavka v razdelku 3.5.2 lahko sklepamo, da poraba časa narašča kot $O(n^3)$, kjer je n številka grafa v zaporedju.

Zakaj izboljšava 2 pri gramatiki GG_{AHC} tako močno poveča učinkovitost sintaksne analize? Oglejmo si primer sintaksne analize grafa linearnega alkana z vozlišči $\mathsf{C}_1, \ldots, \mathsf{C}_m$, kot smo jo obravnavali v razdelku 3.5.2. Denimo, da je faza 1 že odkrila vse r-slike produkcije p_8 in ustvarila pripadajoče produkcijske primerke. Na vozlišče C_1 so torej preko povezav a pripeta tri vozlišča H, na vozlišče C_2 pa dve. Brez izboljšave 2 bi faza 1 r-sliko produkcije p_2, ki jo vsebuje opisani podgraf, odkrila $3! = 6$-krat, izboljšava 2 pa število odkritij zmanjša na 1. Tako bo vozlišče C_2 pridobilo samo eno vozlišče H preko povezave a. Sedaj podgraf, sestavljen iz vozlišč C_2 in C_3 ter nanju vezanih vozlišč H vsebuje samo eno r-sliko produkcije p_2. Ta bo zaradi izboljšave 2 odkrita samo enkrat, zato bo tudi vozlišče C_3 pridobilo samo eno novo vozlišče H. Odkritje r-slike produkcije p_2 v okolici povezave $\mathsf{C}_{i-1}{-}\mathsf{C}_i$ torej doda samo eno vozlišče H k vozlišču C_i. Ker pa zaradi dodanega vozlišča H nastane nova r-slika iste produkcije v okolici povezave $\mathsf{C}_{i-1}{-}\mathsf{C}_i$, bi lahko domnevali, da bo faza 1 kljub izboljšavi zapadla v začaran krog odkrivanj in dodajanj, saj bi po odkritju r-slike dodali novo vozlišče H k vozlišču C_{i-1}, kar bi povzročilo novo odkritje r-slike iste produkcije v okolici povezave $\mathsf{C}_{i-1}{-}\mathsf{C}_i$ itd. Vendar pa se to ne zgodi, saj je produkcijski primerek, ki nastane kot posledica opisanega odkritja, nekonsistenten, zato ga takoj odstranimo in razveljavimo njegove učinke (dodajanje vozlišča H).

Vidimo, da se pri analizi grafa linearnega alkana število primerkov produkcije p_2 povečuje linearno s številom vozlišč. Ker enako velja tudi za primerke drugih produkcij, je število odkritih produkcijskih primerkov linearna funkcija velikosti grafa. Pri grafih s trojnimi vezmi (zaporedje Seq_{S3}) je število produkcijskih primerkov znatno večje kot pri grafih z enojnimi in dvojnimi vezmi, vendar pa je še vedno linearno. Slabša učinkovitost sintaksne analize grafov s trojnimi vezmi je posledica dejstva, da podgraf $\mathsf{C} \equiv \mathsf{C}$ poleg ene r-slike produkcije p_7 vsebuje tudi tri r-slike produkcije p_5 in tri r-slike produkcije p_6, saj je trojno vez mogoče interpretirati kot tri pare dvojnih.

Izboljšava 3 se pri gramatiki GG_{AHC} ne obnese, saj — sodeč po tabeli 3.5 na strani 87 — le produkcije p_5, p_6 in p_7 ne morejo tvoriti nekonsistentnih produkcijskih primerkov. Primerkov teh produkcij pa ni veliko, zato ne moremo kaj dosti prihraniti. Kot kaže primer $GG_{\mathrm{AHC}} + Seq_{\mathrm{S3}}$, se lahko zaradi dodatnega preverjanja

trajanje sintaksne analize celo malo podaljša.

Sintaksni analizator je pri naših poskusih z gramatiko GG_{AHC} ustvarjal veliko nekonsistentnih produkcijskih primerkov. Delež nekonsistentnih produkcijskih primerkov med vsemi odkritimi produkcijskimi primerki je znašal malo manj kot 50% pri sintaksni analizi zaporedij Seq_{S1} in Seq_{B1}, okrog 64% pri zaporedjih Seq_{S2} in Seq_{B2} in kar 96% pri zaporedju Seq_{S3}. S pomočjo izboljšave 4 je bilo možno preprečiti nastanek vseh nekonsistentnih produkcijskih primerkov pri zaporedjih Seq_{S1} in Seq_{B1}, 88% takšnih primerkov pri zaporedjih Seq_{S2} in Seq_{B2} ter 39% takšnih primerkov pri zaporedju Seq_{S3}. Ostali nekonsistentni produkcijski primerki niso ustrezali pogoju v trditvi 3.33 (stran 88), zato jih z izboljšavo 4 nismo mogli preprečiti. Glede na celotno število produkcijskih primerkov (nekonsistentnih in vseh ostalih) je izboljšava 4 pri zaporedjih Seq_{S1} in Seq_{B1} preprečila nastanek 50% primerkov. Pri zaporedjih Seq_{S2} in Seq_{B2} je ta delež znašal 56%, pri zaporedju Seq_{S3} pa 37%. Izboljšava 4 pri vseh petih zaporedjih v grobem doseže podobne relativne prihranke, ki pa se zaradi asimptotično kubične odvisnosti porabe časa od številke člena zaporedja povečujejo skladno z naraščanjem velikosti grafa.

Izboljšava 5 pri zaporedjih grafov linearnih ogljikovodikov (Seq_{S1}, Seq_{S2} in Seq_{S3}) ne prinese bistvenih prihrankov. V tabeli 3.13 vidimo, da se število odkritih produkcijskih primerkov ne glede na velikost grafa zmanjša samo za 1. Zanimivo pa je, da je pri grafih razvejanih ogljikovodikov (zaporedji Seq_{B1} in Seq_{B2}) z uporabo izboljšave 5 možno doseči skoraj optimalno število produkcijskih primerkov. Pri zaporedju Seq_{B1} je razlika med številom izdelanih produkcijskih primerkov pri vključenih vseh petih izboljšavah in optimalnim številom produkcijskih primerkov (tj. dolžina izpeljave) enaka 8 ne glede na velikost grafa, pri zaporedju Seq_{B2} pa optimalno število asimptotično presegamo le za 36 odstotkov, medtem ko ta presežek pri zaporedju Seq_{S2} znaša 109 odstotkov. Tudi razlike v trajanju so očitne. Sintaksni analizator pri obdelavi razvejanega ogljikovodika ustvari povsem enako število produkcijskih primerkov kot pri njegovem linearnem ekvivalentu. S stališča števila in zgradbe produkcijskih primerkov so linearni in razvejani ogljikovodiki enakovredni. Zaradi razlik v zgradbi teh dveh oblik ogljikovodikov (slika 3.15) pa nastane razlika v vrstnem redu odkrivanja produkcijskih primerkov. Pri razvejanih ogljikovodikih je vrstni red s stališča izboljšave 5 ugodnejši, saj — kot so pokazali poskusi — sintaksni analizator bistveno prej ustvari vse produkcijske primerke, ki so potrebni za izpeljavo ogljikovodika.

3.5.3.3 Gramatika GG_{HC}

Pri gramatiki GG_{HC} izboljšava 2 ne zmanjša velikostnega reda rasti števila produkcijskih primerkov oziroma rasti porabe časa in prostora, močno pa zmanjša konstantni faktor. Poskušajmo ugotoviti, zakaj je, denimo, število ustvarjenih produkcijskih primerkov pri grafu linearnega alkana z $m = 3n$ vozlišči enako $9n^2 + 9n + 1$ oziroma $m^2 + 3m + 1$. Enako kot pri izhodiščni različici sintaksnega analizatorja bo faza 1 ustvarila $2m + 2$ primerkov produkcije p_4. Ker sta vozlišči C na desni strani produkcije p_3 medsebojno zamenljivi, bo faza 1 pri iskanju r-slik produkcije p_3 vsako povezavo $\mathsf{C}-\mathsf{C}$ obravnavala samo enkrat. Po izdelavi vseh $m - 1$ primerkov produkcije p_3 bodo na vsako vozlišče C preko povezav a pripeta po 4 vozlišča H. Desna stran produkcije p_2 vsebuje dve medsebojno nezamenljivi in nepovezani vozlišči C,

kar pomeni, da bo faza 1 obravnavala vse pare različnih vozlišč C, pri čemer bo razlikovala med parom (C_i, C_j) (pri $i \neq j$) in parom (C_j, C_i). Ker sedaj vsak par vozlišč C skupaj s pripadajočimi sosedi $\overset{\mathsf{a}}{—}\mathsf{H}$ vsebuje po eno r-sliko produkcije p_2, bo faza 1 ustvarila $m(m-1)$ primerkov produkcije p_2, pri tem pa zaradi praznosti množice $Xlhs[p_2]$ v obravnavani graf ne bo dodala ničesar. Ker so vozlišča H v produkciji p_1 medsebojno zamenljiva, bo faza 1 ustvarila m njenih primerkov, za vsako vozlišče C po enega. Skupno število ustvarjenih produkcijskih primerkov je tako res enako $m^2 + 3m + 1$ oziroma $9n^2 + 9n + 1$.

Če je kvadratno naraščanje števila produkcijskih primerkov slabost gramatike GG_{HC} (v primerjavi z gramatiko GG_{AHC}), pa je njena dobra stran enakovredna obravnava enojnih, dvojnih in trojnih vezi. Zaradi manjšega števila r-slik produkcije p_4 se število produkcijskih primerkov s povečevanjem kratnosti vezi celo rahlo zmanjšuje. Tudi pri porabi časa ni bistvenih razlik med šestimi zaporedji, ki jih uporabljamo skupaj z gramatiko GG_{HC}.

Pri gramatiki GG_{HC} izboljšava 3 precej poveča učinkovitost sintaksne analize, saj smo po formulah iz razdelka 3.4.3 ugotovili, da nobena produkcija ne more tvoriti nekonsistentnih primerkov (tabela 3.5). Zato lahko vrednotenje časovno potratne trojiške relacije $\mathbf{above}^{+}_{pi}$ povsem opustimo. Seveda pa nam zaradi odsotnosti nekonsistentnih produkcijskih primerkov izboljšava 4 ne more prinesti nikakršnih prihrankov.

Poskusimo pojasniti, zakaj nam izboljšava 5 pri sintaksni analizi z vhodno gramatiko GG_{HC} ne koristi. Vsaka izpeljava strukturne formule ogljikovodika, ki vsebuje $m \geq 2$ vozlišč C, se prične s primerkom produkcije p_1, nadaljuje z $m-1$ primerki produkcije p_2, nato pa sledi ustrezno število primerkov produkcij p_3 in p_4. Vendar pa zaradi lastnosti $Xlhs[p_2] = \emptyset$ noben primerek produkcije p_2 ne more biti posledica (**conseqOf**) nobenega drugega primerka. Zato množica produkcijskih primerkov pi, ki so s poljubnim primerkom začetne produkcije (p_1) v relaciji $\mathbf{conseqOf}^{+}$, ne vsebuje nobenega primerka produkcije p_2. Potemtakem je pogoj (3.12) iz razdelka 3.4.5 lahko izpolnjen samo pri sintaksni analizi strukturne formule metana (CH_4), saj zgolj s primerki produkcij p_1, p_3 in p_4 ni mogoče zgraditi strukturnih formul ogljikovodikov z več kot enim ogljikovim atomom. Izboljšave 5 torej pri grafih z več kot enim vozliščem C sploh ne moremo uporabiti.

3.5.3.4 Gramatike GG_{ER}, GG_{FC} in GG_{BT}

Pri dvojici $GG_{\mathrm{ER}} + Seq_{\mathrm{ER}}$ zadovoljivo deluje že izhodiščni sintaksni analizator, saj doseže linearno odvisnost števila produkcijskih primerkov od velikosti grafa. Izboljšava 2 se izplača zaradi medsebojno zamenljivih vozlišč Entity v produkciji p_4 in Attr v produkciji p_6. Porabo časa zmanjša tudi izboljšava 3, saj — sodeč po tabeli 3.5 — produkcije p_1, p_2, p_3 in p_4 ne morejo tvoriti nekonsistentnih produkcijskih primerkov. Pri testnem zaporedju dejansko nobena produkcija ne tvori nekonsistentnega primerka, vendar pa tega za produkcije p_5, p_6 in p_7 ne smemo predpostaviti, saj formule v razdelku 3.4.3 dopuščajo možnost nastanka nekonsistentnih primerkov teh produkcij. Zaradi odsotnosti nekonsistentnih produkcijskih primerkov nam izboljšava 4 ne more koristiti. Izboljšava 5 se prav tako ne obrestuje, in sicer iz podobnih razlogov kot pri gramatiki GG_{HC}: ker imajo vse produkcije razen p_6 prazne množice $Xlhs$, noben primerek teh produkcij ne more biti posledica (**conseqOf**) nobenega

drugega primerka, zato je pogoj (3.12) iz razdelka 3.4.5 lahko izpolnjen zgolj pri grafu, ki ne vsebuje nič drugega kot eno samo vozlišče Entity.

Pri uporabi izhodiščnega sintaksnega analizatorja za par $GG_{\mathrm{FC}} + Seq_{\mathrm{FC}}$ število produkcijskih primerkov narašča kot eksponentna funkcija velikosti grafa, z izboljšavo 2 pa zaradi zamenljivosti vozlišč Stats v produkciji p_7 dosežemo kvadratno rast. Izboljšava 3 nam to pot ne pomaga, saj za nobeno produkcijo ne moremo vnaprej napovedati, da ne bo tvorila nekonsistentnih produkcijskih primerkov. Žal velika večina (npr. 95% pri $n = 20$) nekonsistentnih produkcijskih primerkov ne ustreza pogoju iz trditve 3.33, zato tudi izboljšava 4 ne more doseči otipljivih prihrankov. Majhne prihranke dosežemo z izboljšavo 5, ki še nekoliko zmanjša število ustvarjenih produkcijskih primerkov (npr. za 3,5% pri $n = 20$).

Sodeč po tabeli 3.14 se pri dvojici $GG_{\mathrm{BT}} + Seq_{\mathrm{BT}}$ izhodiščni sintaksni analizator prav tako obnaša eksponentno. V nasprotju z dvojicami $GG_{\mathrm{AHC}} + \ldots$ in $GG_{\mathrm{FC}} + Seq_{\mathrm{FC}}$ pa nas izboljšava 2 tokrat ne reši pred eksponentno rastjo števila produkcijskih primerkov in časovno-prostorske porabe. Medsebojno zamenljivi sta samo vozlišči z oznako B v produkciji p_2, ne pa tudi vozlišči B v produkciji p_4, saj eno od njiju pripada množici $Common[p_4]$, drugo pa množici $Xrhs[p_4]$. Žal pa je prav produkcija p_4 kriva za eksponentno rast. Sodeč po tabeli 3.5 produkciji p_1 in p_3 ne moreta tvoriti nekonsistentnih primerkov, zato se izboljšava 3 nekoliko pozna vsaj pri grafu G_{10}. Ker je velika večina produkcijskih primerkov nekonsistentnih, od teh pa jih precejšen delež izpolnjuje pogoj (3.11) iz trditve 3.33, se izboljšava 4 tokrat izkaže najbolje doslej. Izjemno povečanje učinkovitosti — z eksponentne rasti na linearno — pa doseže izboljšava 5. Sintaksni analizator namreč v primeru para $GG_{\mathrm{BT}} + Seq_{\mathrm{BT}}$ že zelo hitro poišče produkcijske primerke, s katerimi je mogoče sestaviti izpeljavo, vendar pa nato zaradi simetrije v produkciji p_4, ki je izboljšava 2 ne more odpraviti, še dolgo časa ustvarja (po večini nekonsistentne) produkcijske primerke.

3.6 Zaključek

Predstavili smo pet izvirnih izboljšav Rekers-Schürrovega sintaksnega analizatorja za kontekstno odvisne grafne gramatike in vsako od njih ovrednotili na 14 parih grafnih gramatik in zaporedij grafov. Prva izboljšava je povečala razred gramatik, ki jih sintaksni analizator sprejema na vhodu, saj je odpravila zahtevo po produkcijah s povezanimi desnimi stranmi. Ostale izboljšave so povečale računsko učinkovitost sintaksnega analizatorja.

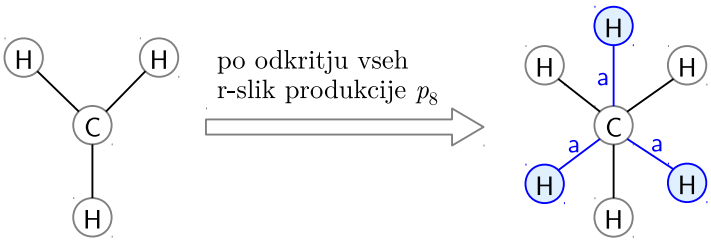
Izboljšava 2 odpravlja redundanco, ki je posledica avtomorfizmov znotraj desnih strani produkcij. S to izboljšavo smo na testnih gramatikah dosegli ogromne prihranke. Brez izboljšave 2 je bil sintaksni analizator pri obeh »kemijskih« gramatikah zaradi kombinatorične eksplozije domala neuporaben, izboljšava 2 pa je porabo časa pri poskusih z navedenima gramatikama spravila v polinomske okvire. Pri poskusih z gramatiko diagramov poteka (GG_{FC}) smo videli, da lahko že ena sama produkcija z enim samim avtomorfizmom povzroči eksponentno rast porabe časa. Z vključitvijo izboljšave 2 smo tudi v tem primeru računsko zahtevnost zmanjšali na polinomsko.

Izboljšavi 3 in 4 sta namenjeni zmanjševanju potrebe po vrednotenju računsko zahtevne relacije **inconsistent**. Izboljšava 3 sestoji iz množice pravil, s pomočjo katerih je mogoče ugotoviti, ali posamezne produkcije lahko tvorijo nekonsisten-

tne produkcijske primerke. Na ta način smo dodatne časovne prihranke dosegli pri eni od »kemijskih« gramatik (GG_{HC}) in pri gramatiki diagramov entiteta-razmerje (GG_{ER}). Z uporabo izboljšave 4 lahko za določene produkcijske primerke še pred njihovim nastankom ugotovimo, da bo zanje veljala relacija **inconsistent** in da jih je zato smiselno takoj izločiti iz nadaljnje obravnave. Ta izboljšava se je najbolj obnesla pri gramatiki dvojiških dreves (GG_{BT}), v manjši meri pa tudi pri »kemijski« gramatiki GG_{AHC}, saj je sintaksni analizator pri poskusih s tema gramatikama tvoril veliko število nekonsistentnih produkcijskih primerkov. Izboljšava 5 poskuša ob izpolnjenosti določenih pogojev predčasno izvesti fazo 2. Ta izboljšava je močno pospešila izvajanje poskusov z gramatiko GG_{BT}, saj je faza 1 pri sintaksni analizi grafov iz zaporedja Seq_{BT} že na začetku ustvarila vse produkcijske primerke, potrebne za izgradnjo izpeljave.

Sintaksni analizator bi bilo možno še dodatno izboljšati. Glede na poskuse z gramatiko GG_{ER} domnevamo, da nobena njena produkcija ne more tvoriti nekonsistentnih produkcijskih primerkov, vendar pa smo s pravili v okviru izboljšave 3 to lahko potrdili le za štiri produkcije od sedmih. Napovedovanje možnosti nastopa nekonsistentnih produkcijskih primerkov je skoraj zanesljivo mogoče še izboljšati.

Kljub izboljšavam je še vedno možno najti zaporedja grafov, ki so za naše testne gramatike z vidika računske zahtevnosti neugodna. Predstavljajmo si zaporedje grafov $Seq_{\mathrm{CH}n}$, v katerem je n-ti člen (graf G_n) sestavljen iz enega vozlišča C, n vozlišč H in n enojnih neoznačenih in neusmerjenih povezav med vozliščem C in vsakim od vozlišč H. Četrti člen zaporedja je torej strukturna formula metana (CH_4), ostali členi pa ne predstavljajo veljavnih ogljikovodikov. Koliko produkcijskih primerkov nastane pri sintaksni analizi grafa G_n v gramatiki GG_{AHC} ali GG_{HC}? Če je $n < 4$, je odgovor enak n, saj v grafu obstaja n r-slik produkcije p_8 (v primeru gramatike GG_{AHC}) oziroma p_4 (v primeru gramatike GG_{HC}), po dodajanju n vozlišč H in enakega števila povezav a (slika 3.19) pa se faza 1 zaključi, saj nastali graf ne vsebuje več nobene še ne odkrite r-slike katere od produkcij gramatike. Pri $n \geq 4$ pa nastali graf vsebuje $\binom{n}{4}$ r-slik produkcije p_1, po eno za vsako četverico vozlišč H. Z izboljšavo 2 sicer dosežemo, da sintaksni analizator vsako r-sliko odkrije le enkrat, še vedno pa odkrije *vse* r-slike, kar pomeni, da bo faza 1 skupno ustvarila $\binom{n}{4} + n = \Theta(n^4)$ produkcijskih primerkov. Ta rast je sicer še vedno polinomska, vendar pa je občutno hitrejša od linearne oz. kvadratne rasti pri ostalih poskusih z gramatiko GG_{AHC} oz. GG_{HC} (gl. tabelo 3.13).



Slika 3.19: *Graf G_3 iz zaporedja $Seq_{\mathrm{CH}n}$ in pripadajoči graf $\overline{G}$ po odkritju vseh r-slik produkcije p_8 iz gramatike GG_{AHC} (ali produkcije p_4 iz gramatike GG_{HC}) v fazi 1.*

Opisani problem lahko rešimo na podoben način kot problem večkratnih odkritij

istih r-slik produkcij. V grafu G_n je vseh n vozlišč H medsebojno zamenljivih (v smislu definicije 3.12), kar je možno ugotoviti s pomočjo procedure poišči**N**eodvisne**R**azrede (slika 3.10). Vsi primerki produkcije p_1 v grafu G_n so med seboj enakovredni, saj vsi pokrivajo isto vozlišče C in enako veliko podmnožico medsebojno zamenljivih vozlišč H. Ker se vsi takšni primerki zaradi prekrivajočih se množic *Xrhs* med seboj izključujejo (**excludes**, gl. definicijo 3.8), zadošča, če faza 1 ustvari samo en primerek produkcije p_1 izmed $\binom{n}{4}$ možnih. Na ta način lahko dosežemo bistvene časovne in prostorske prihranke, res pa je, da z naraščajočim številom n narašča tudi cena ugotavljanja medsebojno neodvisnih vozlišč. Kljub temu pa preliminarni poskusi kažejo, da se opisano izboljšavo splača uvesti. Pri $n = 15$ in gramatiki GG_{AHC} sintaksni analizator brez opisane izboljšave (a z vključeno izboljšavo 2) porabi približno 8.0 sekunde, z obema izboljšavama pa 0.58 sekunde. Pri $n = 50$ bi sintaksni analizator brez opisane izboljšave ustvaril več kot 230 000 produkcijskih primerkov, kar pomeni, da bi nanj čakali najmanj nekaj ur, ob vključeni izboljšavi pa ustvari 51 primerkov in potrebuje skupno 3,6 sekunde. Podobne rezultate dobimo tudi pri gramatiki GG_{HC}.

Nekaj primerov uporabe sintaksnega analizatorja smo navedli že v uvodu. V naslednjem poglavju pa si bomo ogledali njegovo uporabo pri indukciji grafnih gramatik.

POGLAVJE

4

INDUKCIJA GRAFNIH GRAMATIK

V tem poglavju predstavimo izviren pristop k indukciji grafnih gramatik na podlagi množic grafov, ki smo ga motivirali in na kratko predstavili v razdelku 1.1.3. Besedilo tega poglavja v grobem temelji na našem prispevku za konferenco AGTIVE 2011 [38], vendar pa bomo indukcijsko metodo predstavili podrobneje kot v prispevku, prikazali pa bomo tudi več eksperimentalnih rezultatov.

Uvodnemu podpoglavju 4.1 sledi podpoglavje 4.2, ki je namenjeno predstavitvi izbranih sorodnih del. Podpoglavje 4.3 za potrebe predstavitve indukcijske metode uvede nekaj definicij. Indukcijski algoritem gradi gramatike, ki pripadajo podrazredu razreda kontekstno odvisnih grafnih gramatik. Ta podrazred je v podpoglavju 4.4 natančno opredeljen. V podpoglavju 4.5 predstavimo indukcijski algoritem. V podpoglavju 4.6 prikažemo in komentiramo rezultate poskusov, ki smo jih izvedli z indukcijskim algoritmom. V tem podpoglavju se posvetimo tudi problemu računske zahtevnosti. Podpoglavje 4.7 zaključi to poglavje.

4.1 Uvod

Grafne gramatike so kljub svoji relativni nepoznanosti uporabne na različnih področjih [28]. Vendar pa so jih raziskovalci razmeroma redko obravnavali (zgolj) kot formalizme za opisovanje *množic grafov*, čeprav je prav to njihov prvenstveni pomen. V poglavju 3 nas je zanimalo, ali podani graf pripada množici, ki jo podana grafna gramatika opisuje. Govorili smo o problemu pripadnosti in problemu sintaksne analize. V tem poglavju pa si bomo zastavili drugačno vprašanje: kakšna grafna gramatika učinkovito opisuje podano množico grafov? Ukvarjali se bomo s problemom *indukcije* grafne gramatike oziroma s problemom iskanja grafne gramatike, ki pokriva podano množico grafov.

Problem iskanja gramatike, ki pokriva vse vhodne grafe, je trivialno rešljiv, saj množico grafov $\mathcal{G}$ pokriva že gramatika s produkcijami $\lambda ::= G$ za vsak graf $G \in \mathcal{G}$. Vendar pa nas bo zanimala gramatika, ki podano vhodno množico grafov smiselno posplošuje. Kot smo povedali že v uvodnem poglavju (razdelek 1.1.3), si lahko zamislimo dva scenarija indukcije grafne gramatike:

(1) indukcija zgolj na podlagi množice »pozitivnih« grafov;

(2) indukcija na podlagi množice »pozitivnih« in množice »negativnih« grafov.

Pri prvem indukcijskem scenariju predpostavljamo, da vsi vhodni grafi pripadajo jeziku ciljne grafne gramatike. Na izhodu indukcijskega algoritma potemtakem pričakujemo gramatiko, ki vhodno množico smiselno posplošuje. Na primer, če bi indukcijskemu algoritmu podali množico različnih primerov diagramov poteka, bi na izhodu pričakovali splošno gramatiko diagramov poteka. Pri drugem indukcijskem scenariju pa »pozitivni« vhodni grafi po predpostavki pripadajo jeziku ciljne grafne gramatike, »negativni« pa ne. V tem primeru nas zanima grafna gramatika, ki smiselno posplošuje pozitivno vhodno množico, vendar pa ne zajema nobenega grafa iz (posplošene) negativne vhodne množice.

Negativna vhodna množica lahko predstavlja razred grafov z nekim določenim pomenom, lahko pa služi le kot varovalka pred pretiranim posploševanjem pozitivne množice. V primeru indukcije gramatike diagramov poteka bi negativna množica lahko vsebovala primere grafov, ki ne predstavljajo veljavnih diagramov poteka. Negativna množica bi v takšnem primeru zgolj preprečevala pretirano posploševanje, ne bi pa predstavljala razreda grafov z določenim pomenom, saj bi lahko vsebovala raznovrstne grafe, ki jim je skupno le to, da ne predstavljajo veljavnih diagramov poteka. (Seveda ima tudi takšna množica določen pomen, vendar pa je definiran negativno, kar ni običajno.) Negativna množica s pozitivno definiranim pomenom nastopi tedaj, ko se ukvarjamo samo z grafi, ki pripadajo bodisi enemu ali drugemu razredu. Za potrebe primera predpostavimo, da se omejimo zgolj na diagrame poteka, ki vsebujejo natanko en pogojni stavek. Denimo, da bi želeli inducirati gramatiko za ločevanje diagramov s pogojnim stavkom tipa »če-potem« od diagramov s pogojnim stavkom tipa »če-potem-sicer«. V tem primeru bi obe množici, pozitivna in negativna, vsebovali grafe z določenim skupnim pomenom, saj bi (denimo) pozitivna množica vsebovala primere diagramov s pogojnim stavkom tipa »če-potem«, negativna pa primere diagramov s pogojnim stavkom tipa »če-potem-sicer«.

Gramatiko, ki je rezultat indukcije na podlagi pozitivne in negativne vhodne množice, lahko kvantitativno ocenimo na način, ki je običajen v strojnem učenju [70]. Inducirano gramatiko lahko namreč obravnavamo kot klasifikator: če graf pripada njenemu jeziku, lahko rečemo, da ga gramatika »klasificira« kot pozitiven graf, in obratno, zato lahko gramatiko ovrednotimo na enak način kot klasifikator. V ta namen poleg *učnih množic* — množic pozitivnih in negativnih grafov, na podlagi katerih induciramo grafno gramatiko — pripravimo še *testni množici*, množici pozitivnih in negativnih grafov, ki sta namenjeni ocenjevanju inducirane gramatike. Testni množici morata biti strogo ločeni od učnih. Gramatiko induciramo na učnih množicah, nato pa s pomočjo sintaksnega analizatorja za vsak graf iz testnih množic preverimo, ali pripada jeziku inducirane gramatike. Na podlagi dobljenih podatkov lahko izračunamo klasifikacijsko točnost inducirane gramatike na testni množici.

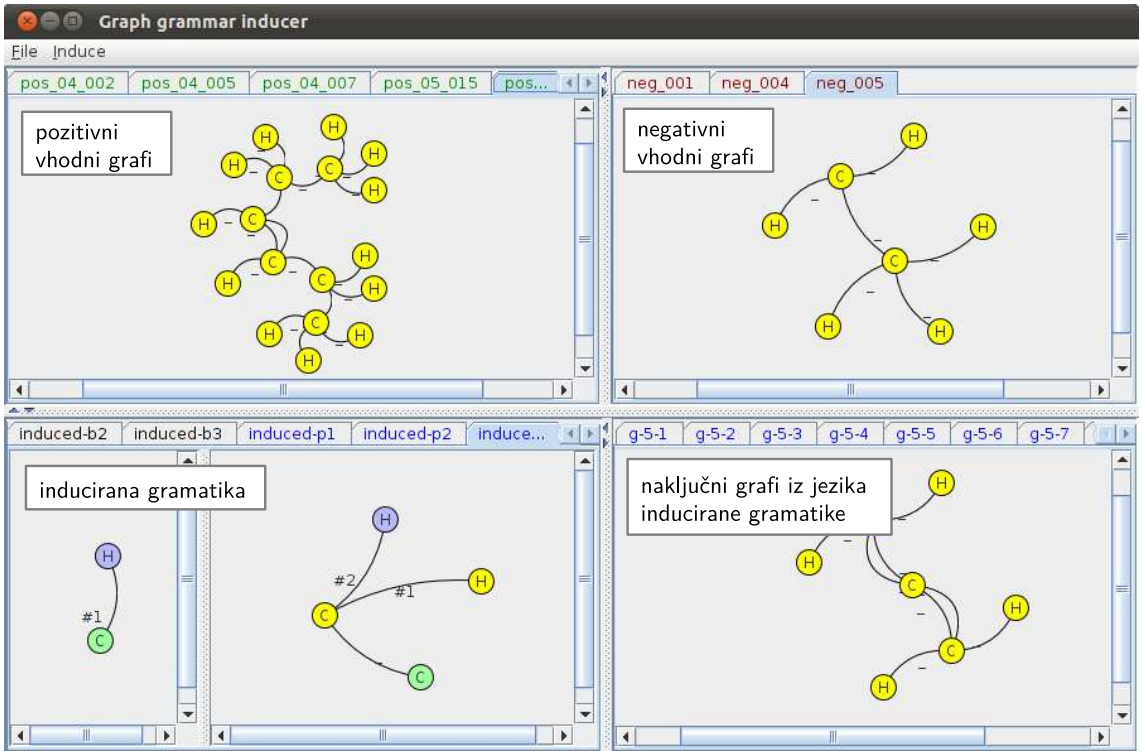
Metodo za indukcijo grafnih gramatik, ki jo predstavljamo v tem poglavju, bi lahko uporabili v orodju za specifikacijo sintakse grafnega jezika z grafno gramatiko. Ker ročna izdelava gramatike za ciljni jezik praviloma ni enostaven problem, bi lahko uporabniku, ki se ga loteva, pomagali z interaktivnim orodjem. Zaslonski posnetek prototipa takšnega orodja je prikazan na sliki 4.1. Uporabnik orodja poda primere grafov, ki pripadajo ciljnemu jeziku, in primere grafov, ki ciljnemu jeziku ne pripadajo, orodje pa nato samodejno zgradi gramatiko, konsistentno s podanimi grafi. Uporabnik lahko zatem po potrebi spremeni množici vhodnih grafov in ponovi indukcijski postopek. Orodje omogoča tudi samodejno tvorbo naključnih grafov, ki pripadajo jeziku inducirane gramatike. Ta možnost je namenjena posrednemu preverjanju inducirane gramatike.

Ker lahko grafna gramatika služi kot klasifikator grafov, je indukcijo grafne gramatike možno obravnavati kot učenje grafnega klasifikatorja. V razdelku 1.1.3 smo omenili uporabo indukcijske metode za gradnjo klasifikatorja kemijskih spojin, dejansko pa bi se lahko indukcijske metode poslužili povsod, kjer se pojavlja potreba po klasifikaciji grafov.

Indukcijski algoritem, ki ga predstavljamo v doktorski disertaciji, hevristično preiskuje prostor grafnih gramatik, konsistentnih s podanima vhodnima množicama grafov. Algoritem prične s trivialno gramatiko (gramatiko, ki pokriva natanko vse pozitivne vhodne grafe), nato pa gradi čedalje splošnejše gramatike, ki so še vedno konsistentne z vhodom. Ker je razred grafnih gramatik, ki jih tvori indukcijski algoritem, podrazred razreda kontekstno odvisnih grafnih gramatik, njihovo konsistentnost z vhodnima množicama preverjamo s pomočjo izboljšanega Rekers-Schürrovega sintaksnega analizatorja, ki smo ga predstavili v poglavju 3. Če sintaksni analizator ugotovi, da dana gramatika pokriva katerega od negativnih vhodnih grafov, potem indukcijski algoritem takšno gramatiko takoj zavrže.

Medtem ko je več sorodnih del problem indukcije prevedlo na problem iskanja v smeri od specifičnega proti splošnemu, pa lahko postopek posploševanja gramatik opredelimo kot izviren. Do določene mere je izvirna tudi vključitev sintaksnega analizatorja v postopek indukcije. Kot bomo videli v podpoglavju 4.2, preplet indukcije in sintaksne analize sicer ni novost na področju indukcije *besedilnih* gramatik, je pa (po našem védenju) novost pri indukciji *grafnih* gramatik.

Ciljni formalizem naše indukcijske metode — razred grafnih gramatik, ki jih induciramo — je podoben povezavnim gramatikam in je zato šibkejši od splošnih



Slika 4.1: *Zaslonski posnetek prototipa orodja za interaktivno indukcijo grafne gramatike.*

vozliščnih, hiperpovezavnih ali kontekstno odvisnih grafnih gramatik. Kljub temu pa lahko, kot bomo videli, s ciljnim formalizmom opišemo številne zanimive grafne jezike.

4.2 Sorodna dela

Na področju indukcije grafnih gramatik doslej ni bilo opravljenega zelo veliko dela. To lahko pripišemo relativni neznanosti samih grafnih gramatik in zahtevnosti problema indukcije, najbrž pa tudi odsotnosti učinkovitih sintaksnih analizatorjev za grafne gramatike, kar izvira iz NP-težkosti problema pripadnosti za številne splošne razrede grafnih gramatik.

Enega od prvih pristopov k indukciji grafnih gramatik sta predlagala Jeltsch in Kreowski [48]. Njun algoritem inducira hiperpovezavno grafno gramatiko na podlagi množice pozitivnih vhodnih grafov. Algoritem prične s trivialno hiperpovezavno gramatiko, nato pa jo zaporedoma posplošuje. Naš algoritem v osnovi temelji na podobni ideji (čeprav uporabljamo drugačen način posploševanja), vendar pa je posplošitveni operator v našem pristopu vpet v iskalno shemo; za razliko od Jeltscha in Kreowskega preiskujemo prostor možnih gramatik. Poleg tega lahko naš algoritem sprejema tako pozitivne kot negativne vhodne grafe.

Pristopi, ki so jih predstavili Jonyer in sod. [49], Kukluk in sod. [56] ter Ates in sod. [4], temeljijo na algoritmu *Subdue* [44], ki si ga velja nekoliko pobliže ogledati.

Algoritem Subdue išče ponovljive podgrafe v danem vhodnem grafu G. Grafi, ki imajo v danem vhodnem grafu veliko različnih pojavitev, lahko predstavljajo nek zanimiv samostojen koncept (npr. funkcionalno skupino v kemijski formuli), zato se jih splača poiskati. Algoritem Subdue išče podgrafe od spodaj navzgor z uporabo snopovnega iskanja. Začetni nabor podgrafov sestavljajo posamezna vozlišča, nato pa se podgrafi iterativno povečujejo. Vsak podgraf se ovrednoti z uporabo *načela najkrajšega opisa* (angl. *minumum description length (MDL) principle*). Algoritem za vsak podgraf $S \subseteq G$ izračuna dve vrednosti: $D(S)$ predstavlja število bitov, potrebnih za kodiranje grafa S, $D(G|S)$ pa predstavlja število bitov, ki bi bili potrebni za kodiranje grafa G, če bi vse pojavitve grafa S v grafu G nadomestili z ustrezno kodo. Algoritem išče podgrafe S z najmanjšo vsoto $D(S) + D(G|S)$. To so podgrafi, ki vodijo do največjih prihrankov pri opisu grafa G.

Jonyer in sod. [49] gradijo grafno gramatiko na podlagi pozitivnih vhodnih grafov. Tudi njihov algoritem deluje v smeri od specifičnega proti splošnemu. V vsakem posplošitvenem koraku s pomočjo algoritma Subdue poišče tisti podgraf S v množici vhodnih grafov, ki ima v smislu načela najkrajšega opisa najboljšo oceno. Algoritem takšen graf S zamenja z vozliščem z oznako A in v grafno gramatiko, ki jo gradi, doda produkcijo $A ::= S$. Ker produkcije v nasprotju s produkcijami običajnih vozliščnih gramatik (gl. razdelek 2.3.3) niso opremljene z vložitvenimi pravili, lahko gramatike, ki jih gradi algoritem Jonyerja in sod., učinkovito predstavljajo zgolj verige podobnih grafov, ki so med seboj na istoležnih mestih povezani s po eno povezavo. Primer takšnega grafa predstavlja graf para-terfenila[1], ki je sestavljen iz treh aromatskih obročev, pri čemer sta prvi in drugi ter drugi in tretji med seboj na istoležnih mestih povezana s po eno povezavo. Razred grafov, ki jih je možno opisati z induciranimi grafnimi gramatikami, je torej precej omejen.

Izboljšano različico tega pristopa Jonyerja in sod. so predlagali Kukluk in sod. [56]. Njihov pristop inducira gramatike, ki lahko predstavljajo tudi zaporedja grafov s skupnimi povezavami, denimo graf tetracena[2], ki je sestavljen iz štirih aromatskih obročev, pri čemer si po dva sosednja obroča delita skupno povezavo.

Nedavno sta Brijder in Blockeel [12] predstavila metodo za indukcijo vozliščnih gramatik na podlagi enega vhodnega grafa, ki vsebuje množico izomorfnih podgrafov. Oates in sod. [73] so predstavili metodo za učenje verjetnostnih parametrov t.i. stohastičnih grafnih gramatik z znano strukturo. Pri tovrstnih gramatikah so produkcije opremljene z verjetnostmi uporabe, zato posamezni grafi pripadajo jeziku stohastične gramatike le z določeno verjetnostjo.

Nobeden od omenjenih pristopov ne uporablja sintaksnega analizatorja. Zaradi tega ne morejo sprejemati negativnih vhodnih grafov, ki bi lahko preprečevali morebitne pretirane posplošitve grafnih gramatik. Pristop Atesa in sod. [4] uporablja sintaksni analizator, vendar le za preverjanje grafne gramatike, ki jo indukcijski algoritem vrne kot svoj rezultat; uporaba sintaksnega analizatorja ni vtkana v sam postopek indukcije, kot to velja za našo metodo. Ates in sod. inducirajo gramatike iz podrazreda t.i. prostorskih grafnih gramatik [53].

Navdih za problem indukcije grafnih gramatik predstavlja problem indukcije *besedilnih* gramatik, kjer je bilo doslej opravljenega nekoliko več raziskovalnega dela

[1] `http://en.wikipedia.org/wiki/Terphenyl`
[2] `http://en.wikipedia.org/wiki/Tetracene`

[75, 90]. Mnogi pristopi temeljijo na podobnih idejah kot naša metoda, tj. iskanje v smeri od specifičnega proti splošnemu, preverjanje kandidatnih gramatik s pomočjo sintaksnega analizatorja ipd. [26, 72].

Črepinšek in sod. [21] so se problema indukcije kontekstno neodvisnih besedilnih gramatik lotili z genetskimi pristopi. Njihova metoda gradi kandidatne gramatike s pomočjo genetskih operatorjev križanja in mutacije, mera za primernost (angl. *fitness*) posameznih kandidatnih gramatik pa je stopnja konsistentnosti s podano vhodno množico.

Zanimiv je tudi pristop Vanlehna in Balla [95], ki inducira kontekstno neodvisne besedilne gramatike z uporabo t.i. *prostora različic* (angl. *version space*) [69]. Metoda vzdržuje dve množici kandidatnih gramatik, $\mathcal{G}$ in $\mathcal{S}$, ki sta podmnožici množice vseh možnih gramatik, ki jih algoritem lahko tvori med svojim delovanjem. Množico $\mathcal{G}$ sestavljajo najbolj splošne gramatike, ki ne pokrivajo nobenega od (do danega trenutka obravnavanih) negativnih vhodnih nizov, množico $\mathcal{S}$ pa tvorijo najbolj specifične gramatike, ki pokrivajo vse (do danega trenutka obravnavane) pozitivne vhodne nize. Vsaka gramatika, ki je najmanj tako splošna kot gramatike iz množice $\mathcal{S}$ in kvečjemu tako splošna kot gramatike iz množice $\mathcal{G}$, je torej konsistentna z množico (doslej obravnavanih) vhodnih nizov. Metoda zaporedoma obravnava vhodne nize. Z naraščanjem števila obravnavanih negativnih nizov postajajo gramatike iz množice $\mathcal{G}$ čedalje bolj specifične, z naraščanjem števila obravnavanih pozitivnih nizov pa postajajo gramatike iz množice $\mathcal{S}$ čedalje splošnejše. Rezultat izvajanja algoritma je prostor gramatik, omejen z gramatikami iz množic $\mathcal{G}$ in $\mathcal{S}$. Vse gramatike iz tega prostora so namreč konsistentne z vsemi vhodnimi nizi. Opisani postopek bi bilo mogoče prilagoditi tudi za indukcijo grafnih gramatik, žal pa je že pri indukciji besedilnih gramatik računsko precej zahteven.

Problem indukcije grafnih gramatik je soroden tudi problemu indukcije metamodelov [46] in problemu učenja modelskih transformacijskih pravil na podlagi primerov transformacij med modeli [6]. Ker lahko modelska transformacijska pravila predstavimo z grafnimi gramatikami v formalizmu trojnih grafnih gramatik [86], lahko problem učenja transformacijskih pravil formuliramo kot problem indukcije tovrstnih gramatik.

Področje indukcije grafnih gramatik je povezano s področjem grafnega podatkovnega rudarjenja (angl. *graph data mining*) [18], še zlasti s problemom iskanja pogostih podgrafov v dani množici grafov. Cilj teh algoritmov je poiskati vse grafe, ki imajo v dani množici grafov dovolj veliko število pojavitev (npr. večje od določene spodnje meje). Poleg že opisanega algoritma Subdue [44] velja omeniti še sledeče:

- Algoritem *FSG* [57] je primeren za iskanje pogostih podgrafov v množici manjših neoznačenih neusmerjenih grafov. Algoritem sistematično tvori t.i. kandidatne grafe z naraščajočim številom povezav; najprej tvori vse možne grafe brez povezav, nato grafe z eno povezavo itd. Za vsak kandidatni graf določi število njegovih pojavitev v vhodni množici. Večje kandidatne grafe tvori z združevanjem manjših. Ker je določanje števila pojavitev grafa v množici gostiteljskih grafov NP-poln problem [94], algoritem FSG število pojavitev večjih grafov zgolj ocenjuje na podlagi (ocene) števila pojavitev manjših grafov.

- Algoritem *HSiGram* [59] je prilagoditev algoritma FSG za primer enega sa-

mega velikega vhodnega grafa. Od algoritma FSG se razlikuje predvsem po načinu štetja pojavitev posameznih kandidatnih grafov, saj upošteva tudi primere, ko se različne pojavitve istega grafa med seboj prekrivajo.

- Medtem ko algoritma FSG in HSiGram tvorita kandidatne grafe po izčrpnem sistematičnem postopku, ki je v grobem neodvisen od vhodne množice grafov, pa algoritem *VSiGram* [59] uporablja zgolj informacijo, ki je na voljo v vhodni množici. Algoritem na začetku v vhodnih grafih poišče vse podgrafe z eno samo povezavo, nato pa večje podgrafe gradi tako, da manjše podgrafe razširja s povezavami, katerih slike dejansko obstajajo v vhodni množici. V nasprotju z algoritmoma FSG in HSiGram tako gradi samo podgrafe, ki imajo v vhodni množici najmanj eno pojavitev. Podoben način iskanja podgrafov uporabljamo tudi pri naši indukcijski metodi.

- Algoritem *Grew* [58] hierarhično združuje posamezne pogoste podgrafe v t.i. supervozlišča. Posamezno supervozlišče predstavlja celoten podgraf. Algoritem je preprostejši in časovno učinkovitejši od algoritmov FSG, HSiGram in VSiGram, vendar pa je tudi manj natančen, saj med drugim zaradi krčenja podgrafov v supervozlišča sploh ne more obravnavati prekrivajočih se pojavitev istih grafov.

4.3 Definicije

Poleg definicij, ki smo jih uvedli v poglavju 2, nam bo v nadaljevanju koristilo še nekaj konceptov, povezanih z grafi in grafnimi gramatikami. Nekateri med njimi (npr. induciran podgraf) so splošno uveljavljeni, nekatere (npr. soseščina podgrafa) pa bomo definirali za potrebe tega poglavja.

Definicija 4.1 (soseščina podgrafa)**.** Naj bo graf G povezan podgraf grafa H ($G \sqsubseteq H$). Potem je *soseščina* grafa G v grafu H (oznaka: $Nh_H(G)$) množica vseh vozlišč v množici $\mathcal{V}[H] \setminus \mathcal{V}[G]$, ki so povezana z najmanj enim vozliščem v množici $\mathcal{V}[G]$:

$$Nh_H(G) = \{v \in \mathcal{V}[H] \setminus \mathcal{V}[G] \mid \exists w \in \mathcal{V}[G],\, e \in \mathcal{E}[H] \colon conn(e) = \{v, w\}\}. \quad (4.1)$$

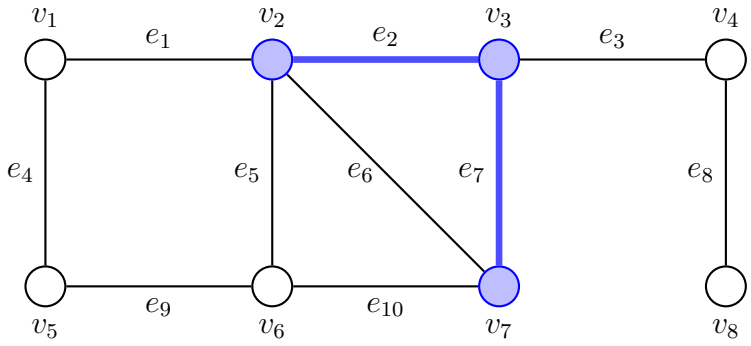
Slika 4.2 prikazuje primer grafa in podgrafa. Graf je sestavljen iz vseh prikazanih elementov, podgraf pa samo iz modro označenih. V tem primeru je soseščina podgrafa v grafu množica vozlišč $\{v_1, v_4, v_6\}$.

Definicija 4.2 (induciran podgraf)**.** Graf G je *induciran podgraf* (angl. *induced subgraph*) grafa H, če velja $G \sqsubseteq H$, poleg tega pa za vsako povezavo $e \in \mathcal{E}[H]$ velja sledeče:

$$conn(e) \subseteq \mathcal{V}[G] \implies e \in \mathcal{E}[G] \quad (4.2)$$

Z drugimi besedami: induciran podgraf grafa H je sestavljen iz neke podmnožice vozlišč grafa H in iz vseh povezav grafa H, ki povezujejo vozlišča v tej podmnožici.

Podgraf na sliki 4.2 ni induciran, saj vsebuje vozlišči v_2 in v_7, ne vsebuje pa povezave med njima. Podgraf bi postal induciran, če bi mu dodali še povezavo e_6.

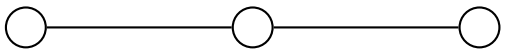


Slika 4.2: *Graf in (neinduciran) podgraf.*

Besedna zveza »induciran podgraf« nima nikakršne povezave z indukcijo gramatik, ki je predmet pričujočega poglavja. Da bi preprečili morebitne nesporazume zaradi dveh različnih pomenov pridevnika »induciran«, bomo besedno zvezo »induciran podgraf« nadomestili s tvorjenko »i-podgraf«. Ker se bomo ukvarjali samo s povezanimi podgrafi, se dogovorimo, da beseda »i-podgraf« predstavlja *povezan* induciran podgraf.

Definicija 4.3 (inducirana pojavitev (i-pojavitev))**.** Naj bosta G in H grafa, $h\colon G \to H$ pa monomorfizem. Potem je graf $h(G) \sqsubseteq H$ *inducirana pojavitev* (*i-pojavitev*) grafa G v grafu H natanko tedaj, ko je graf $h(G)$ i-podgraf grafa H.

Vzemimo za primer sledeči graf:



Ta graf ima v grafu s slike 4.2 več i-pojavitev, npr. podgraf, ki ga tvori množica elementov $\{v_1, e_1, v_2, e_5, v_6\}$, podgraf $\{v_7, e_7, v_3, e_3, v_4\}$ itd. Podgraf $\{v_2, e_2, v_3, e_7, v_7\}$, ki je na sliki 4.2 označen z modro barvo, pa *ni* i-pojavitev gornjega grafa, saj ni induciran. Modro označeni podgraf je zgolj navadna (neinducirana) pojavitev gornjega grafa.

Definicija 4.4 (splošnost in specifičnost gramatike)**.** Gramatika GG je *vsaj tako splošna* kot gramatika GG' natanko tedaj, ko velja $L(GG) \supseteq L(GG')$. Gramatika GG je *vsaj tako specifična* kot gramatika GG' natanko tedaj, ko velja $L(GG) \subseteq L(GG')$.

V nadaljevanju bomo uporabljali sledečo notacijo za zapis nekaterih grafov:

- Zapis $\langle u\colon A \xrightarrow{e\colon t} v\colon B\rangle$ bo predstavljal neusmerjen graf z lastnostmi $\mathcal{V} = \{u, v\}$, $\mathcal{E} = \{e\}$, $conn(e) = \{u, v\}$, $label(u) = A$, $label(v) = B$ in $label(e) = t$. Kadar nas oznake povezave in njenih krajišč ne bodo zanimale, bomo zapis okrajšali v $\langle u \xrightarrow{e} v\rangle$.

- Zapis $\langle u\colon A \xrightarrow{e\colon t} v\colon B\rangle$ bo predstavljal usmerjen graf z lastnostmi $\mathcal{V} = \{u, v\}$, $\mathcal{E} = \{e\}$, $source(e) = u$, $target(e) = v$, $label(u) = A$, $label(v) = B$ in $label(e) = t$. Kadar nas oznake povezave in njenih krajišč ne bodo zanimale, bomo zapis okrajšali v $\langle u \xrightarrow{e} v\rangle$.

- Zapis $\langle u(S)v \rangle$ bo predstavljal usmerjen ali neusmerjen povezan graf, sestavljen iz vozlišč u in v, podgrafa S z lastnostjo $Nh_{\langle u(S)v \rangle}(S) = \{u, v\}$ in množice povezav med vozlišči podgrafa S in vozliščema u in v. Pri takšnem zapisu nas zanima samo dejstvo, da vozlišči u in v tvorita soseščino podgrafa S, povezave med soseščino in podgrafom pa nas ne zanimajo.

Spomnimo se na pojem *velikosti* grafne gramatike (definicija 2.23 na strani 40). Rekli bomo, da je gramatika GG_1 *večja* (oz. *manjša*) od gramatike GG_2, če velja $size(GG_1) > size(GG_2)$ (oz. $size(GG_1) < size(GG_2)$).

4.4 Ciljni formalizem

Z izrazom *ciljni formalizem* bomo označevali razred grafnih gramatik, ki jih je (potencialno) zmožen inducirati indukcijski algoritem. Algoritem tudi med svojim delovanjem tvori izključno gramatike iz ciljnega formalizma. Ciljni formalizem naše indukcijske metode je podrazred kontekstno odvisnih grafnih gramatik, podoben razredu povezavnih gramatik (podrazdelek 2.3.4.1). Formalizem, ki smo ga izbrali, je sicer šibkejši od razreda splošnih kontekstno odvisnih grafnih gramatik, opisanih v podpoglavju 2.4, kljub temu pa z njim lahko izrazimo nekatere zanimive grafne jezike, kot so npr. jeziki, ki jih generirajo gramatike GG_{AHC}, GG_{FC} in GG_{BT} s slik 3.1 (stran 53) in 3.2 (stran 54). Gramatika GG_{BT} že sama po sebi pripada ciljnemu formalizmu indukcijskega algoritma, za gramatiko GG_{FC} pa to skoraj velja, saj bi bilo za dosego skladnosti s ciljnim formalizmom treba zgolj nebistveno spremeniti produkcijo p_3 (vozlišče Stats in povezavo do njega bi morali odstraniti iz množice *Common*). Kasneje bomo videli, da je tudi gramatiko GG_{AHC} možno prepisati v enakovredno gramatiko, ki pripada ciljnemu formalizmu.

Gramatike iz ciljnega formalizma so definirane na enak način kot kontekstno odvisne grafne gramatike (definicija 2.17 na strani 39), vendar pa je, kot bomo videli v sledeči definiciji, oblika produkcij omejena na tri tipe.

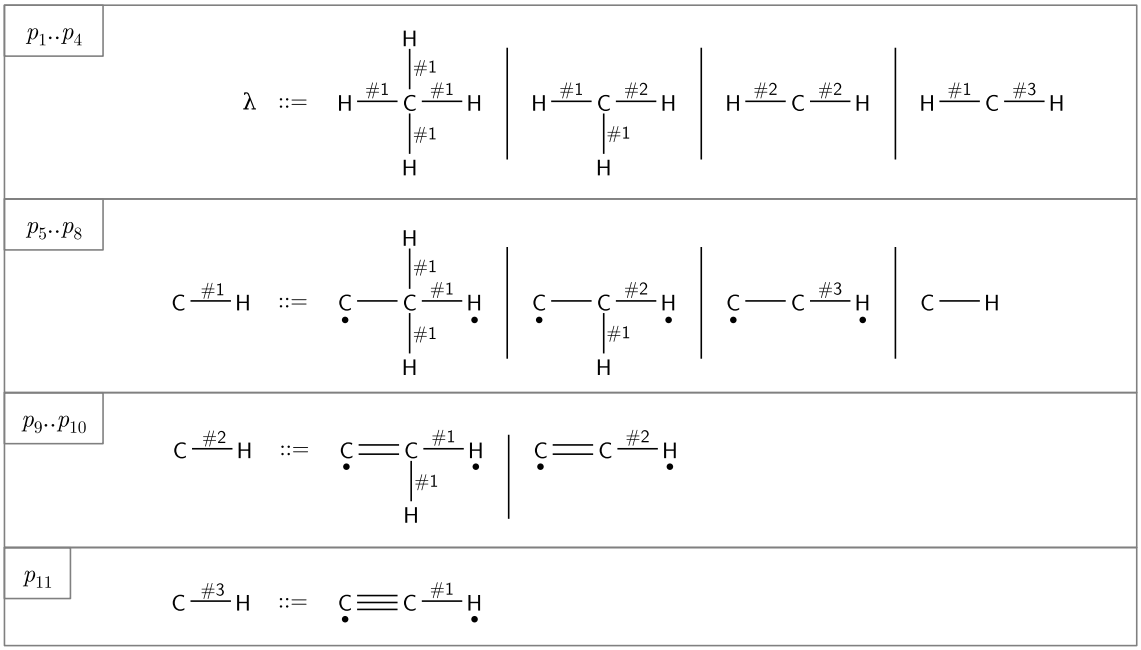
Definicija 4.5 (ciljni formalizem indukcijskega algoritma)**.** Kontekstno odvisna grafna gramatika $GG = (\mathcal{N}, \mathcal{T}, \mathcal{P})$ pripada ciljnemu formalizmu indukcijskega algoritma natanko tedaj, ko je množica nekončnih oznak $\mathcal{N}$ sestavljena iz oznak oblike $\#i$ pri $i \in \mathbb{N}$ (torej $\mathcal{N} = \{\#1, \#2, \ldots\}$), vsaka produkcija pa pripada enemu od sledečih treh tipov:

Tip I: Produkcije tega tipa zavzemajo obliko $\lambda ::= R$, kjer je R poljuben graf.

Tip II: Produkcije tega tipa zavzemajo obliko $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B \rangle ::= \langle u(S)v \rangle$ ali $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B \rangle ::= \langle u(S)v \rangle$, kjer velja $\{A, B\} \subseteq \mathcal{T}$, $\#i \in \mathcal{N}$, S pa je neprazen povezan graf, ki mora biti pri prvi obliki produkcije neusmerjen, pri drugi pa usmerjen. Graf S bomo imenovali *jedro* produkcije, vozlišči u in v pa *stražarja*. Jedro je pri obeh oblikah produkcije lahko s stražarjema povezano s poljubnim številom poljubno označenih neusmerjenih povezav (pri prvi obliki) oziroma usmerjenih povezav (pri drugi obliki), le pogoj $Nh_{\langle u(S)v \rangle}(S) = \{u, v\}$ mora biti pri obeh oblikah izpolnjen.

Tip III: Produkcije tega tipa zavzemajo obliko $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u \xrightarrow{f\colon r} v\rangle$ ali $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u \xrightarrow{f\colon r} v\rangle$, kjer velja $\{A, B\} \subseteq \mathcal{T}$, $\#i \in \mathcal{N}$ in $r \in \mathcal{T} \cup \mathcal{N}$.

Slika 4.3 prikazuje gramatiko, ki pripada ciljnemu formalizmu in ki generira isti jezik kot gramatika GG_{AHC} s slike 3.1, torej jezik acikličnih ogljikovodikov. Zapis oblike $L ::= R_1 \mid R_2 \mid \ldots \mid R_n$ predstavlja množico produkcij oblike $L ::= R_1$, $L ::= R_2$, $\ldots$, $L ::= R_n$. S črnimi krogci ob oznakah vozlišč so označeni stražarji (elementi množic *Common*) na desnih straneh produkcij tipa II. Prikazana gramatika vsebuje vse tipe produkcij: produkcije p_1, p_2, p_3 in p_4 so tipa I, produkcija p_8 je tipa III, ostale produkcije pa so tipa II.



Slika 4.3: *Gramatika, ki generira isti jezik kot gramatika GG_{AHC} s slike 3.1.*

Za gramatike, ki pripadajo ciljnemu formalizmu indukcijskega algoritma, je množica začetnih grafov definirana povsem enako kot za splošne kontekstno odvisne grafne gramatike (definicija 2.17):

Definicija 4.6 (množica začetnih grafov)**.** Naj bo GG kontekstno odvisna grafna gramatika, ki pripada ciljnemu formalizmu indukcijskega algoritma. *Množico začetnih grafov* gramatike GG (oznaka: $\mathcal{B}(GG)$) tvorijo vse desne strani produkcij tipa I:

$$\mathcal{B}(GG) = \{G \mid (\lambda ::= G) \in \mathcal{P}[GG]\} \tag{4.3}$$

So gramatike iz ciljnega formalizma indukcijskega algoritma sintaksno odločljive? Pogoja (2) in (3) iz definicije 2.36 (stran 46) sta očitno izpolnjena. Izpolnjen je tudi prvi pogoj, čeprav to s stališča izboljšane različice Rekers-Schürrovega sintaksnega analizatorja ne bi bilo potrebno. Pokazati pa je mogoče tudi izpolnjenost plastnega

pogoja. Pri produkcijah tipa I je plastni pogoj vedno izpolnjen, ne glede na razporeditev oznak po plasteh. Če oznakam vozlišč dodelimo nižje številke plasti kot oznakam povezav, potem je plastni pogoj izpolnjen tudi pri produkcijah tipa II, saj sta stražarja skupna obema stranema produkcije, desna stran pa zaradi nepraznosti jedra S vsebuje vsaj še eno dodatno vozlišče. Produkcije tipa III pa izpolnjujejo plastni pogoj natanko tedaj, ko v množici tovrstnih produkcij ni nobenih cikličnih podmnožic, kot je npr. $\{A \overset{p}{-} B ::= A \overset{q}{-} B,\ A \overset{q}{-} B ::= A \overset{r}{-} B,\ A \overset{r}{-} B ::= A \overset{p}{-} B\}$. V primeru odsotnosti takih podmnožic je namreč mogoče oznake povezav po plasteh razporediti tako, da bodo plastni vektorji levih strani pri vseh produkcijah leksikografsko manjši od plastnih vektorjev desnih strani.

Sledeča lastnost nam bo prišla prav pri posploševanju gramatik v okviru indukcijskega algoritma (podpoglavje 4.5):

Trditev 4.7. Naj bo p produkcija tipa II z desno stranjo oblike $\langle u(S)v \rangle$ in z različno označenima stražarjema u in v. Potreben (ne pa tudi zadosten!) pogoj za možnost *obratne* uporabe produkcije p na grafu G je obstoj i-podgrafa $S' \sqsubseteq G$, ki je izomorfen grafu S in ki ima v grafu G natanko dve sosednji vozlišči ($|Nh_G(S')| = 2$).

Dokaz. Pri obratni uporabi produkcije p na grafu G moramo v grafu G najprej poiskati r-sliko produkcije p. Ker je graf S v celoti vsebovan v množici $Xrhs[p]$, se morajo elementi grafa S na ustrezne elemente r-slike v grafu G preslikati z monomorfizmom (injektivno), ne zgolj s homomorfizmom. Zato mora graf G vsebovati podgraf S', ki je izomorfen grafu S.

Ker sta oznaki stražarjev u in v po predpostavki različni, se morata stražarja preslikati v dve različni vozlišči v grafu G, čeprav za elemente množice $Common[p]$, ki ji stražarja pripadata, injektivna preslikava v splošnem ni zahtevana. Zato mora podgraf S' v grafu G imeti najmanj dve sosednji vozlišči.

Pokažimo sedaj, da mora podgraf S' v grafu G imeti *natanko* dva soseda. Če je število sosedov večje od 2, potem to pomeni, da podgraf S' ni povezan samo s slikama stražarjev u in v, ampak je vsaj eno od njegovih vozlišč povezano še z nekim vozliščem $w \in \mathcal{V}[G] \setminus \mathcal{V}[S']$. Če to drži, potem produkcije p na podgrafu S' ne moremo obratno uporabiti, saj bi po odstranitvi podgrafa S' in njegovih povezav s slikama stražarjev u in v (kar bi bil prvi korak obratne uporabe produkcije p) v grafu G ostala vsaj ena viseča povezava — povezava do vozlišča w. Zato mora veljati $|Nh_G(S')| = 2$.

Pokažimo še, da mora biti podgraf S' i-pojavitev grafa S v grafu G, ne zgolj navadna pojavitev. Če graf S' ni induciran podgraf grafa G, potem v grafu G obstaja vsaj ena povezava med dvema vozliščema grafa S' (ali vsaj ena zanka na nekem vozlišču grafa S'), ki ni del grafa S'. To ponovno pomeni, da bi po odstranitvi grafa S' v grafu G nastala vsaj ena viseča povezava, zato produkcije p na grafu S' ni mogoče obratno uporabiti. Zaradi tega mora biti graf S' i-podgraf grafa G oziroma i-pojavitev grafa S. □

4.5 Indukcijski algoritem

V tem podpoglavju bomo podrobno opisali indukcijsko metodo. Algoritem bomo opisali po načelu »od zgoraj navzdol«. V razdelku 4.5.1 bomo predstavili ogrodje in-

dukcijskega algoritma. V razdelku 4.5.2 bomo opisali prvi, v razdelku 4.5.3 pa drugi način posploševanja gramatik, ki ga algoritem uporablja pri gradnji novih gramatik. V razdelku 4.5.4 bomo na kratko predstavili algoritem za iskanje i-podgrafov, ki predstavlja osnovo za prvi način posploševanja gramatik.

V nadaljevanju razdelka se bomo pogosto sklicevali na sliko 4.4, ki prikazuje izsek postopka indukcije grafne gramatike na podlagi strukturne formule butana. Algoritem najprej izdela gramatiko GG_1, nadaljnje gramatike pa gradi s pomočjo posploševanj obstoječih. Puščice na sliki predstavljajo posamezne posplošitvene korake. Oznaka oblike $\mathrm{A}/p_{j,k}$ na puščici od gramatike GG_i do gramatike GG_j pove, da je algoritem gramatiko GG_j zgradil s posplošitvijo gramatike GG_i na prvi način (»a-posplošitev«), pri čemer je ustvaril produkcijo $p_{j,k}$ in jo dodal v gramatiko GG_j. Oznaka B na puščici $GG_i \rightarrow GG_j$ pa pove, da je algoritem gramatiko GG_j zgradil s posplošitvijo gramatike GG_i na drugi način (»b-posplošitev«).

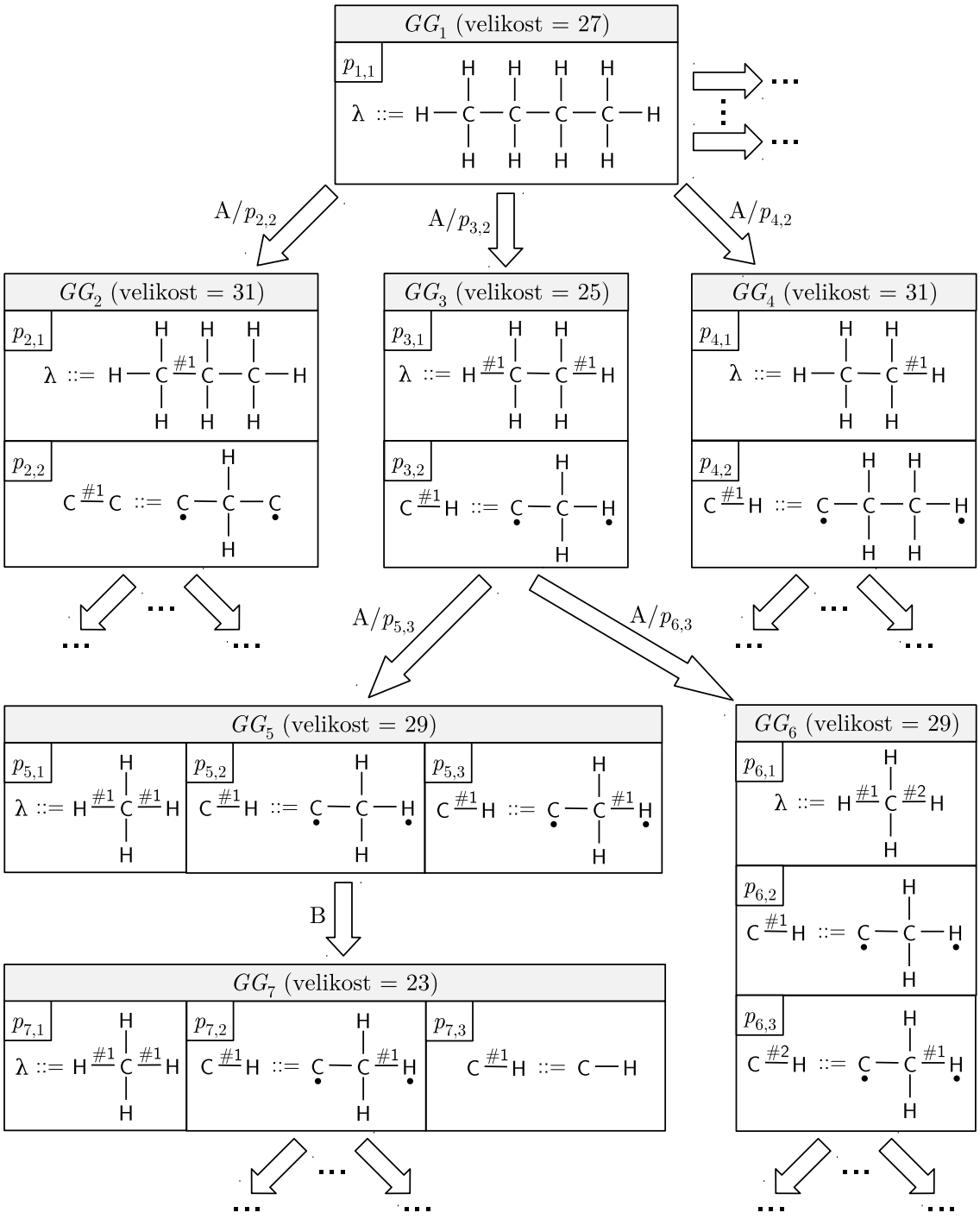
4.5.1 Pregled indukcijskega algoritma

Psevdokoda indukcijskega algoritma je prikazana na sliki 4.5. Algoritem inducira grafno gramatiko na podlagi neprazne množice pozitivnih grafov ($\mathcal{G}^+$) in (po želji prazne) množice negativnih grafov ($\mathcal{G}^-$), pri čemer morata imeti podani množici prazen presek. Pomen dodatnih parametrov *širinaSnopa* in *maksŠtJedrnihVozlišč* bomo pojasnili kasneje.

Indukcijski algoritem deluje po načelu snopovnega iskanja (angl. *beam search*). Iskalni prostor algoritma si lahko predstavljamo kot usmerjen acikličen graf, v katerem vozlišča predstavljajo posamezne *kandidatne gramatike*. Kandidatne gramatike so grafne gramatike v ciljnem formalizmu (podpoglavje 4.4), ki pokrivajo vse pozitivne vhodne grafe in nobenega negativnega in so tako konsistentne z vhodnima množicama $\mathcal{G}^+$ in $\mathcal{G}^-$. V grafu iskalnega prostora obstaja povezava $u \rightarrow v$ natanko tedaj, ko je kandidatna gramatika, ki pripada vozlišču v, *osnovna posplošitev* kandidatne gramatike, ki pripada vozlišču u. O osnovnih posplošitvah bomo več povedali v nadaljevanju, v danem trenutku pa se zadovoljimo s pojasnilom, da je osnovna posplošitev dane gramatike GG gramatika, ki je vsaj tako splošna kot gramatika GG, torej gramatika, ki pokriva vsaj toliko pozitivnih vhodnih grafov kot gramatika GG.

Indukcijski algoritem prične z iskanjem v izhodiščnem vozlišču konceptualnega grafa iskalnega prostora, torej v vozlišču brez vstopnih povezav. Izhodiščno vozlišče predstavlja t.i. izhodiščno gramatiko — gramatiko, ki pokriva natanko množico pozitivnih vhodnih grafov. Ta gramatika je najbolj specifična gramatika v iskalnem prostoru; vse ostale gramatike so pridobljene kot neposredne ali posredne posplošitve izhodiščne gramatike. Zato lahko v grobem trdimo, da indukcijski algoritem preiskuje prostor kandidatnih gramatik v smeri od specifičnega proti splošnemu, čeprav to ne drži v vseh korakih izvajanja, saj algoritem ne obravnava vozlišč iskalnega prostora strogo v skladu z relacijo osnovne posplošitve. Indukcijski algoritem v svojem iskalnem prostoru išče *najmanjšo* gramatiko, saj lahko za takšno gramatiko po hevrističnem načelu Ockhamove britve pričakujemo, da bo smiselno posploševala množico pozitivnih vhodnih grafov.

V vrstici 2 na sliki 4.5 indukcijski algoritem zgradi izhodiščno gramatiko (GG_1).



Slika 4.4: *Indukcija grafne gramatike na podlagi strukturne formule butana.*

```
1  procedura inducirajGramatiko(G+, G−, širinaSnopa, maksŠtJedrnihVozlišč)
2      GG1 := gramatika s produkcijami λ ::= G (za vsak graf G ∈ G+);
3      GGmin := GG1;        // GGmin: do danega trenutka najmanjša gramatika
4      Q := {GG1};
5      dokler Q ≠ ∅ izvajaj
6          GG := najmanjša gramatika v vrsti Q;
7          če |GG| < |GGmin| potem
8              GGmin := GG
9          konec;
10         Q := Q \ {GG};
11         NoveProdukcije := poišči_nove_produkcije(GG, maksŠtJedrnihVozlišč);
12         za vsako produkcijo p ∈ NoveProdukcije izvedi
13             GG' := A-posplošitev gramatike GG z dodatkom produkcije p;
14             če GG' ne pokriva nobenega grafa iz množice G− potem
15                 GG'' := B-posplošitev gramatike GG' (če obstaja);
16                 če GG'' obstaja in ne pokriva nobenega grafa iz množice G− potem
17                     Q := Q ∪ {GG''}
18                 sicer
19                     Q := Q ∪ {GG'}
20                 konec
21             konec
22         konec;
23         v vrsti Q ohrani širinaSnopa najmanjših gramatik, ostale pa zavrzi
24     konec;
25     vrni GGmin
26 konec
```

Slika 4.5: *Psevdokoda indukcijskega algoritma.*

Algoritem vzdržuje do danega trenutka najmanjšo gramatiko (GG_{min}) in prioritetno vrsto Q, v kateri je prioriteta določena z velikostjo gramatike (najmanjša gramatika ima največjo prioriteto). Vrsta na začetku vsebuje zgolj izhodiščno gramatiko. V svoji glavni zanki (vrstice 5–24) algoritem najprej pridobi najmanjšo gramatiko (GG) iz vrste Q, nato pa po potrebi posodobi podatek o doslej najmanjši gramatiki. Sledi odstranitev gramatike GG iz vrste Q. V vrsticah 11–22 algoritem tvori vse osnovne posplošitve gramatike GG in vsako nastalo gramatiko, ki je konsistentna z negativno vhodno množico, doda v vrsto Q. V skladu z delovanjem snopovnega iskanja je velikost vrste omejena s *širino snopa* (angl. *beam width*), ki jo določa parameter *širinaSnopa*. Algoritem zato v prioritetni vrsti ohrani zgolj *širinaSnopa* najmanjših gramatik, ostale pa zavrže. Ko se vrsta izprazni, algoritem vrne najmanjšo izdelano kandidatno gramatiko (vrstica 25) in se izteče. Algoritem lahko prekinemo kadarkoli med izvajanjem in za njegov izhod proglasimo do danega trenutka najmanjšo gramatiko (GG_{min}). Ker so vse gramatike, ki jih algoritem postavi v vrsto, konsistentne z vhodnima množicama, bo gramatika GG_{min} v vsakem trenutku izvajanja algoritma pokrivala vse pozitivne vhodne grafe in nobenega negativnega. Proceduro

poišči**NoveProdukcije**, ki jo indukcijski algoritem kliče v vrstici 11, bomo predstavili v razdelku 4.5.2.

Pri preverjanju konsistentnosti posameznih gramatik z vhodnima množicama zadošča, če preverimo zgolj konsistentnost z negativno množico, saj je konsistentnost s pozitivno množico zagotovljena že z dejstvom, da pričnemo z gramatiko, ki pokriva vse pozitivne vhodne grafe, nato pa gramatike kvečjemu posplošujemo, s čimer njihove jezike kvečjemu povečujemo. Konsistentnost posameznih gramatik z negativno vhodno množico preverjamo s pomočjo izboljšanega Rekers-Schürrovega sintaksnega analizatorja, ki smo ga predstavili v poglavju 3. Če sintaksni analizator najde izpeljavo katerega od negativnih grafov, potem indukcijski algoritem trenutno obravnavano gramatiko nemudoma zavrže, v nasprotnem primeru pa jo postavi v prioritetno vrsto.

Ker smo indukcijski algoritem prevedli na iskanje po prostoru kandidatnih gramatik, bi lahko namesto snopovnega iskanja uporabili tudi kakšen drugačen iskalni postopek. Za snopovno iskanje smo se odločili zato, ker se nam zdi dober kompromis med popolnimi, a računsko zahtevnimi izčrpnimi preiskovalnimi pristopi, kot sta iskanje v širino in iskanje v globino (angl. *breadth-first search* in *depth-first search*), ter med računsko učinkovitejšimi hevrističnimi pristopi, ki pa ne jamčijo optimalne rešitve. S parametrom *širinaSnopa* lahko nastavljamo stopnjo »izčrpnosti« iskalnega postopka. Če širino snopa nastavimo na neskončno, dobimo iskanje v širino, če jo nastavimo na 1, pa dobimo iskanje tipa »samo najboljši« (angl. *best-only search*), ki v vsakem koraku obravnava samo trenutno najmanjšo gramatiko, ostale pa zavrže.

Hevrističnost indukcijskega algoritma se izkazuje v dveh vidikih: v osredotočenosti na iskanje *najmanjše* gramatike in v omejitvi na gramatike, konsistentne z vhodnima množicama. Čeprav je obe hevristiki mogoče smiselno utemeljiti, nikakor ne želimo trditi, da sta »optimalni« v kakršnemkoli pomenu te besede. Inducirano gramatiko lahko kvantitativno ocenimo le s pomočjo posebne testne množice grafov, ki je povsem ločena od vhodnih (učnih) množic $\mathcal{G}^+$ in $\mathcal{G}^-$ in je ne poznamo v času indukcije.

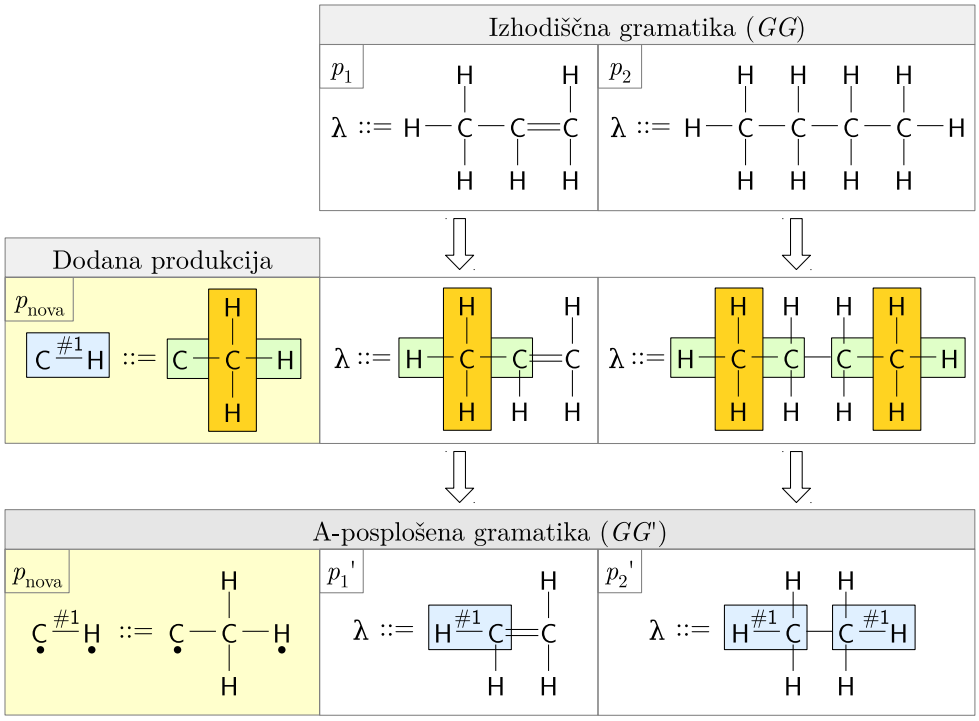
Kot smo že omenili, algoritem gramatike posplošuje na dva načina. Osnovno posplošitev prvega tipa (»a-posplošitev«) bomo opisali v razdelku 4.5.2, osnovno posplošitev drugega tipa (»b-posplošitev«) pa v razdelku 4.5.3. Pri podani grafni gramatiki je rezultat obeh tipov osnovnih posplošitev grafna gramatika, ki pokriva najmanj toliko pozitivnih vhodnih grafov kot podana gramatika. Ni nujno, da je nastala gramatika *dejanska* posplošitev podane gramatike; lahko je zgolj njena preoblikovana različica. V vsakem primeru pa po a- ali b-posplošitvi dobimo gramatiko, ki je najmanj tako splošna kot podana gramatika.

4.5.2 A-posplošitev

Začasno predpostavimo, da obe vhodni množici sestavljajo neusmerjeni grafi. Z usmerjenimi grafi se bomo ukvarjali ob koncu razdelka.

Naj bo GG gramatika, ki pokriva vse pozitivne vhodne grafe in nobenega negativnega. *A-posplošitev* gramatike GG je gramatika, ki jo pridobimo tako, da gramatiki GG dodamo produkcijo p_{nova} tipa II, nato pa dodano produkcijo p_{nova} v *obratni smeri* uporabimo na vseh začetnih grafih gramatike (grafih iz množice

$\mathcal{B}(GG)$), na katerih je to možno vsaj enkrat storiti. Postopek obratnih uporab ponavljamo, dokler je mogoče. Vrstni red izvajanja obratnih uporab ni določen. V postopku a-posplošitve se produkcije oblike $\lambda ::= R$ (produkcije tipa I) pretvorijo v produkcije $\lambda ::= R'$, kjer je graf R' bodisi enak grafu R ali pa je pridobljen z eno ali več obratnimi uporabami produkcije p_{nova} na grafu R. Primer a-posplošitve je prikazan na sliki 4.6. Na sliki 4.4 so skoraj vse prikazane osnovne posplošitve a-posplošitve; izjema je le posplošitev $GG_5 \rightarrow GG_7$. Na primer, gramatika GG_3 je a-posplošitev gramatike GG_1 z dodano produkcijo $p_{3,2}$.



Slika 4.6: *Gramatika in njena a-posplošitev.*

Pokažimo, da a-posplošitev gramatike GG pokriva vsaj toliko grafov kot gramatika GG:

Trditev 4.8. Naj bo gramatika GG' a-posplošitev gramatike GG z dodano produkcijo p_{nova}. Potem je gramatika GG' vsaj tako splošna kot gramatika GG.

Dokaz. Po definiciji a-posplošitve so vse produkcije tipov II in III, ki jih vsebuje gramatika GG, vsebovane tudi v gramatiki GG'. Morebitne razlike so zgolj pri produkcijah tipa I, torej pri produkcijah oblike $\lambda ::= R$. Lahko se zgodi, da gramatika GG vsebuje produkcijo $\lambda ::= R$, gramatika GG' pa produkcijo $\lambda ::= R'$, kjer je R' graf, pridobljen iz grafa R z obratnimi uporabami produkcije p_{nova}. Vendar pa lahko vsako izpeljavo v gramatiki GG, ki se prične z zaporedjem $\lambda \Rightarrow R$, nadomestimo z enakovredno izpeljavo $\lambda \Rightarrow R' \overset{p_{\text{nova}}}{\Longrightarrow}{}^{*} R$ v gramatiki GG', saj je graf R (po definiciji) mogoče pridobiti z uporabami produkcije p_{nova} na grafu R'. Ker je produkcije tipa I mogoče uporabiti samo na začetku izpeljave, lahko zaključimo, da za vsako izpeljavo v gramatiki GG obstaja enakovredna izpeljava v gramatiki GG'. Zaradi tega je vsak graf, ki pripada jeziku gramatike GG, tudi pripadnik jezika gramatike GG'. □

A-posplošitev gramatike GG lahko načeloma pridobimo z dodatkom poljubne produkcije tipa II. Vendar pa je prvenstveni cilj a-posplošitve zmanjšanje oziroma vsaj preoblikovanje gramatike. Če ima nek graf oblike $\langle u(S)v\rangle$ veliko pojavitev v množici $\mathcal{B}(GG)$, potem se nam v gramatiko splača dodati produkcijo oblike $\langle u \overset{e}{—} v\rangle ::= \langle u(S)v\rangle$, saj bodo obratne uporabe te produkcije pojavitve grafa $\langle u(S)v\rangle$ nadomestile z manjšimi grafi $\langle u \overset{e}{—} v\rangle$. Produkcij, ki nimajo nobene r-slike v množici $\mathcal{B}(GG)$, pa sploh nima smisla obravnavati, saj z njihovo uvedbo ni mogoče doseči nikakršnih prihrankov — vsaj ne kratkoročno.

Pri iskanju kandidatnih produkcij p_{nova} se hevristično omejimo na produkcije, ki imajo vsaj po eno r-sliko v množici začetnih grafov gramatike GG. Zato med vsemi možnimi produkcijami oblike $\langle u\colon A \overset{e\colon \#i}{—} v\colon B\rangle ::= \langle u(S)v\rangle$ upoštevamo samo takšne, pri katerih ima graf $\langle u(S)v\rangle$ vsaj eno pojavitev v vsaj enem izhodiščnem grafu gramatike GG. Ker vnaprej ne moremo vedeti, katera a-posplošitev gramatike GG bo (neposredno ali posredno) vodila do najmanjše gramatike, moramo obravnavati čimveč kandidatov za dodano produkcijo p_{nova}.

Postopek iskanja kandidatnih produkcij tipa II, ki lahko nastopajo kot produkcije p_{nova} pri gradnji a-posplošitev dane gramatike, je prikazan na sliki 4.7. Procedura poišči<u></u>NoveProdukcije sprejme gramatiko GG, ki jo želimo a-posplošiti, in parameter *maksŠtJedrnihVozlišč*, ki ga bomo pojasnili v kratkem. Kot smo povedali, iščemo samo takšne produkcije p_{nova} oblike $\langle u\colon A \overset{e\colon \#i}{—} v\colon B\rangle ::= \langle u(S)v\rangle$, ki jih je mogoče obratno uporabiti vsaj na enem grafu iz množice $\mathcal{B}(GG)$. Produkcija ima to lastnost samo tedaj, kadar ima njena desna stran (graf $\langle u(S)v\rangle$) v množici $\mathcal{B}(GG)$ vsaj eno pojavitev, njeno jedro (graf S) pa ima v okviru te pojavitve svojo i-pojavitev z natanko dvema sosedama (gl. trditev 4.7).

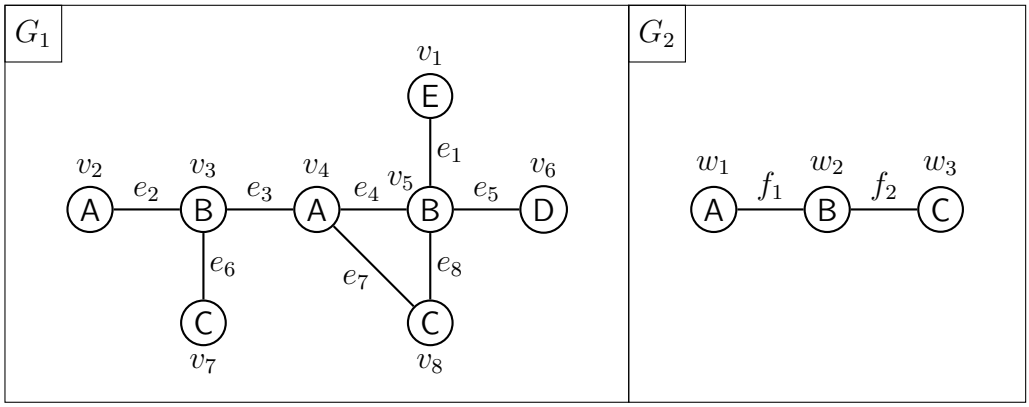
Za iskanje potencialnih jeder bodočih produkcij p_{nova} sta zadolžena procedura poišči<u></u>Podgrafe v vrstici 2 in pogoj v vrstici 6. Procedura poišči<u></u>Podgrafe izdela množico vseh grafov z največ *maksŠtJedrnihVozlišč* vozlišči, ki imajo v množici $\mathcal{B}(GG)$ vsaj po eno i-pojavitev. Množica *VzGoPo*, ki jo vrne procedura poišči<u></u>Podgrafe, je sestavljena iz parov oblike (*Vzorec*, *GoPo*), kjer je *Vzorec* graf, ki ima vsaj eno i-pojavitev v množici $\mathcal{B}(GG)$, *GoPo* pa je množica parov (*Gostitelj*, *Pojavitev*), kjer je *Pojavitev* neka i-pojavitev grafa *Vzorec* v grafu *Gostitelj* $\in \mathcal{B}(GG)$. Lahko se seveda zgodi, da ima isti graf *Vzorec* več i-pojavitev v istem grafu *Gostitelj*; v tem primeru množica *GoPo* vsebuje več parov (*Gostitelj*, *Pojavitev*) za isti graf *Gostitelj*. Proceduro poišči<u></u>Podgrafe bomo predstavili v razdelku 4.5.4.

Pogojni stavek v vrstici 6 preveri, ali ima trenutno obravnavana i-pojavitev trenutno obravnavanega grafa *Vzorec* v svojem gostiteljskem grafu natanko dve sosednji vozlišči. Če to drži, potem lahko na osnovi grafa *Vzorec* zgradimo vsaj eno produkcijo, ki jo bo mogoče vsaj enkrat obratno uporabiti na vsaj enem grafu iz množice $\mathcal{B}(GG)$. Algoritem v vrsticah 16–18 ustvari najmanj eno kandidatno produkcijo p_{nova}. Vsaka ustvarjena produkcija zavzema obliko $\langle u\colon A \overset{e\colon \#i}{—} v\colon B\rangle ::= \langle u(\mathit{Vzorec})v\rangle$, kjer sta A in B oznaki vozlišč, ki v gostiteljskem grafu trenutno obravnavane i-pojavitve grafa *Vzorec* nastopata kot soseda te i-pojavitve, i pa je neko naravno število. Ustvarjene produkcije se med seboj razlikujejo samo po vrednostih spremenljivke i. O tem bomo več povedali v nadaljevanju, sedaj pa predstavimo postopek iskanja kandidatnih produkcij s primerom.

Denimo, da gramatiko induciramo na podlagi grafov G_1 in G_2 s slike 4.8. Množica

```
1  procedura poišči_Nove_Produkcije(GG, maksŠtJedrnihVozlišč)
2      VzGoPo := poišči_Podgrafe(B(GG), maksŠtJedrnihVozlišč);
3      Produkcije := ∅;
4      za vsak par (Vzorec, GoPo) ∈ VzGoPo izvedi
5          za vsak par (Gostitelj, Pojavitev) ∈ GoPo izvedi
6              če |Nh_Gostitelj(Pojavitev)| = 2 potem
7                  naj bosta u in v vozlišči v množici Nh_Gostitelj(Pojavitev);
8                  A := label(u);
9                  B := label(v);
10                 I := {i | ⟨u: A --e:#i-- v: B⟩ ima i-pojavitev v GG};
11                 če I = ∅ potem
12                     k := 1
13                 sicer
14                     k := max(I) + 1
15                 konec;
16                 za vsak i ∈ I ∪ {k} izvedi
17                     Produkcije := Produkcije ∪ {⟨u: A --e:#i-- v: B⟩ ::= ⟨u(Vzorec)v⟩}
18                 konec
19             konec
20         konec
21     konec;
22     vrni Produkcije
23 konec
```

Slika 4.7: *Iskanje kandidatnih produkcij* p_{nova}.



Slika 4.8: *Dva primera grafov.*

VzGoPo bi v tem primeru vsebovala številne pare (*Vzorec*, *GoPo*). Navedimo tri med njimi:

- (A, { $(G_1, \{v_2\})$, $(G_1, \{v_4\})$, $(G_2, \{w_1\})$ }).

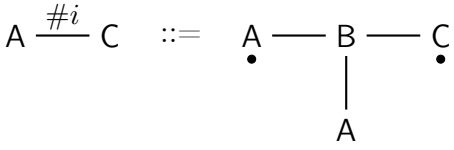
  Ta par predstavlja dejstvo, da ima graf, sestavljen zgolj iz vozlišča z oznako A, dve i-pojavitvi v grafu G_1 (i-podgraf, ki ga tvori vozlišče v_2, in i-podgraf, ki ga tvori vozlišče v_4) in eno i-pojavitev v grafu G_2 (i-podgraf, ki ga tvori vozlišče w_1).

- (A−B, { $(G_1, \{v_2, e_2, v_3\})$, $(G_1, \{v_4, e_3, v_3\})$, $(G_1, \{v_4, e_4, v_5\})$, $(G_2, \{w_1, f_1, w_2\})$ }).

- (A−B−C, { $(G_1, \{v_2, e_2, v_3, e_6, v_7\})$, $(G_1, \{v_4, e_3, v_3, e_6, v_7\})$, $(G_2, \{w_1, f_1, w_2, f_2, w_3\})$ }).

  Podgraf $\{v_4, e_4, v_5, e_8, v_8\}$ v grafu G_1 *ni* i-pojavitev grafa A−B−C, saj ni induciran.

Med navedenimi i-pojavitvami ima le ena v svojem gostiteljskem grafu *natanko* dva soseda. Gre za i-pojavitev grafa A−B, ki v grafu G_1 obsega množico elementov $\{v_2, e_2, v_3\}$; njena soseda sta vozlišči v_4 (A) in v_7 (C). Ker sta oba soseda povezana z vozliščem v_3 (B), bi metoda poišči Nove Produkcije zgradila najmanj eno kandidatno produkcijo sledeče oblike:



Vrnimo se k splošni obravnavi procedure poišči Nove Produkcije. Videli smo, da podana dvojica grafa *Vzorec* in njegove i-pojavitve (skupaj z gostiteljskim grafom, ki vsebuje to i-pojavitev) natančno določa vse sestavne dele kandidatne produkcije p_{nova} razen nekončne oznake $\#i$. Spremenljivki i bi lahko dodelili poljubno naravno število in bi tako teoretično dobili neskončno mnogo produkcij p_{nova}. Vendar pa sta produkciji $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u(S)v\rangle$ in $\langle u\colon A \xrightarrow{e\colon \#j} v\colon B\rangle ::= \langle u(S)v\rangle$ s stališča posploševanja gramatike GG povsem enakovredni, če se grafa $\langle A \xrightarrow{\#i} B\rangle$ in $\langle A \xrightarrow{\#j} B\rangle$ ne pojavljata nikjer v množici desnih strani produkcij gramatike GG. Če bi takšni produkciji v skladu s postopkom a-posploševanja posamično dodali v gramatiko GG, bi gramatiko GG obakrat zgolj preoblikovali brez dejanske posplošitve. Če pa graf $\langle A \xrightarrow{\#i} B\rangle$ nastopa kot podgraf v vsaj eni desni strani v gramatiki GG, bi produkcija $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u(S)v\rangle$ gramatiko (praviloma) dejansko posplošila, saj bi nastala gramatika (praviloma) pokrivala več grafov kot gramatika GG. Zato je število različnih oznak povezave na levi strani produkcije (in s tem število različnih kandidatnih produkcij) pogojeno s številom pojavitev grafov $\langle A \xrightarrow{\#i} B\rangle$ v desnih straneh produkcij gramatike GG. Kandidatno produkcijo $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u(S)v\rangle$ ustvarimo za vsako naravno število i, pri katerem graf $\langle A \xrightarrow{\#i} B\rangle$ nastopa kot podgraf v vsaj eni desni strani v gramatiki GG, in za

število k, ki je za 1 večje od največjega števila i z opisano lastnostjo. Na ta način upoštevamo različne možnosti za posplošitev gramatike, poleg tega pa upoštevamo tudi možnost preoblikovanja gramatike brez dejanske posplošitve.

Za zgled vzemimo gramatiko GG_3 s slike 4.5. Pri a-posploševanju te gramatike bi procedura poišči*NoveProdukcije* izdelala kandidatni produkciji $p_{5,3}$ in $p_{6,3}$, ki se razlikujeta zgolj v oznaki povezave na levi strani. Pri a-posplošitvi gramatike GG_3 s produkcijo $p_{5,3}$ dobimo gramatiko GG_5, ki pokriva več grafov od gramatike GG_3 in je zato njena dejanska posplošitev, po dodatku produkcije $p_{6,3}$ pa dobimo gramatiko GG_6, ki je zgolj preoblikovana različica gramatike GG_3.

Po vsaki izvedbi a-posplošitve nastalo gramatiko še dodatno preoblikujemo. To storimo tako, da vsako njeno produkcijo p tipa II obratno uporabimo na vsakem začetnem grafu, na katerem je takšna obratna uporaba možna. Postopek ponavljamo, dokler je mogoče. Dobljena gramatika je poenostavljena, a povsem enakovredna različica izhodiščne gramatike. Opisani poenostavitveni postopek ni bistven za delovanje indukcijskega algoritma, vendar pa lahko znatno izboljša njegovo računsko učinkovitost.

Doslej smo se omejili na neusmerjene grafe. V primeru usmerjenih grafov postopek a-posploševanja in iskanja kandidatnih produkcij p_{nova} poteka enako, le da lahko graf $\langle u(S)v\rangle$ nastopa kot desna stran dveh različnih družin produkcij:

- $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u(S)v\rangle$
- $\langle v\colon B \xrightarrow{e\colon \#i} u\colon A\rangle ::= \langle u(S)v\rangle$

Pri delu z usmerjenimi grafi imamo torej pri določanju leve strani produkcije dve prostostni stopnji: oznako in smer povezave. Ker nobene smeri ne moremo vnaprej proglasiti za obetavnejšo, moramo upoštevati obe možnosti. Podobno kot pri neusmerjenih grafih družina $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u(S)v\rangle$ vsebuje produkcijo za vsako naravno število i, pri katerem graf $\langle A \xrightarrow{\#i} B\rangle$ nastopa kot podgraf v vsaj eni desni strani v gramatiki GG, in za število k, ki je za 1 večje od največjega števila i z opisano lastnostjo. Družina produkcij $\langle v\colon B \xrightarrow{e\colon \#i} u\colon A\rangle ::= \langle u(S)v\rangle$ je definirana na enak način.

4.5.3 B-posplošitev

Osnovna zamisel b-posplošitve je poenostavitev gramatike; morebitna posplošitev je zgolj stranski učinek. Podobno kot pri a-posplošitvi je b-posplošitev gramatike GG gramatika, ki je vsaj tako splošna kot gramatika GG.

B-posplošitev dane kandidatne gramatike nadomesti dve »podobni« produkciji tipa I ali II (»podobnost« bomo definirali kasneje) s produkcijo istega tipa, ki zajema skupne sestavne dele obeh produkcij, ter z množico produkcij tipa III, ki opisujejo razlike med njima. Na sliki 4.4 je gramatika GG_7 b-posplošitev gramatike GG_5: »podobni« produkciji $p_{5,2}$ in $p_{5,3}$ v gramatiki GG_5 sta nadomeščeni s produkcijo $p_{7,2}$, ki zajema skupne sestavne dele produkcij $p_{5,2}$ in $p_{5,3}$ (v tem primeru je ta produkcija kar enaka produkciji $p_{5,3}$), in s produkcijo $p_{7,3}$, ki zajema edino razliko med produkcijama $p_{5,2}$ in $p_{5,3}$. Ni težko pokazati, da je gramatika GG_7 vsaj tako splošna kot gramatika GG_5.

Pojem »podobnih« produkcij temelji na konceptu *poenotenja*:

Definicija 4.9 (poenotenje oznak povezav)**.** Naj bo $GG = (\mathcal{N}, \mathcal{T}, \mathcal{P})$ gramatika, ki pripada ciljnemu formalizmu. Oznaki povezave l in m je *možno poenotiti* ($l \cong m$) natanko tedaj, ko velja $(l = m) \vee (l \in \mathcal{N}) \vee (m \in \mathcal{N})$. Če je oznaki l in m možno poenotiti, potem je njuno *poenotenje* (angl. *unification*) definirano takole:

$$unif(l,\, m) = \begin{cases} l, & \text{če velja } l \in \mathcal{N}; \\ m & \text{sicer.} \end{cases} \tag{4.4}$$

Definicija 4.10 (poenotenje grafov)**.** Grafa G in H je *možno poenotiti* natanko tedaj, ko obstaja *unifikacijska preslikava* $h\colon G \to H$ s sledečimi lastnostmi:

- Preslikava h bijektivno preslika vozlišča grafa G v vozlišča grafa H in povezave grafa G v povezave grafa H.
- Za vsako povezavo $e \in \mathcal{E}[G]$ velja $source[H](h(e)) = h(source[G](e))$ in $target[H](h(e)) = h(target[G](e))$ (v primeru usmerjenih grafov) oziroma $conn[H](h(e)) = h(conn[G](e))$ (v primeru neusmerjenih grafov).
- Za vsako vozlišče $v \in \mathcal{V}[G]$ velja $label(h(v)) = label(v)$.
- Za vsako povezavo $e \in \mathcal{E}[G]$ velja $label(h(e)) \cong label(e)$.

Če je grafa G in H možno poenotiti z unifikacijsko preslikavo $h\colon G \to H$, potem je *poenotenje* grafov G in H graf $G' = unif(G, H)$, ki ga iz grafa G pridobimo tako, da vsaki njegovi povezavi $e \in \mathcal{E}[G]$ nastavimo oznako na poenotenje trenutne oznake povezave e in oznake povezave $h(e)$ v grafu H:

$$label[G'](e) = unif(label[G](e),\, label[H](h(e))). \tag{4.5}$$

Unifikacijska preslikava je torej preslikava, ki je podobna izomorfizmu, le da je zahteva $label(e) = label(h(e))$ nadomeščena z milejšo zahtevo $label(e) \cong label(h(e))$. Vse ostale lastnosti unifikacijske preslikave pa so enake kot pri izomorfizmu.

Ponovno bomo začasno predpostavili, da delamo z neusmerjenimi grafi. Definirajmo b-posplošitev produkcij tipa I in II:

Definicija 4.11 (b-posplošitev dvojice produkcij tipa I)**.** Naj bosta p in q produkciji gramatike GG s sledečimi lastnostmi:

- Produkcija p zavzema obliko $\lambda ::= Rhs[p]$.
- Produkcija q zavzema obliko $\lambda ::= Rhs[q]$.
- Grafa $Rhs[p]$ in $Rhs[q]$ je možno poenotiti.

Potem je *b-posplošitev* produkcij p in q množica, ki vsebuje sledeče:

- produkcijo $r_0\colon \lambda ::= unif(Rhs[p],\, Rhs[q])$;

- po eno produkcijo $r_e\colon \langle w\colon P \xrightarrow{f\colon \#j} w'\colon Q\rangle ::= \langle w \xrightarrow{f'\colon l} w'\rangle$ za vsako povezavo $e \in \mathcal{E}[Rhs[r_0]]$ z lastnostmi $label(conn(e)) = \{P, Q\}$, $label(e) = \#j$ in $label(g(e)) = l \vee label(g'(e)) = l$ (pri $l \neq \#j$), kjer sta $g\colon Rhs[r_0] \to Rhs[p]$ in $g'\colon Rhs[r_0] \to Rhs[q]$ unifikacijski preslikavi (pri podani povezavi $e \in \mathcal{E}[Rhs[r_0]]$ sta $g(e)$ in $g'(e)$ povezavi v grafih $Rhs[p]$ in $Rhs[q]$, iz katerih je nastala povezava e v poenotenem grafu $Rhs[r_0]$).

Definicija 4.12 (b-posplošitev dvojice produkcij tipa II)**.** Naj bosta p in q produkciji gramatike GG s sledečimi lastnostmi:

- Produkcija p zavzema obliko $\langle u\colon A \xrightarrow{e\colon \#i} v\colon B\rangle ::= \langle u(S)v\rangle$.
- Produkcija q zavzema obliko $\langle u'\colon A \xrightarrow{e'\colon \#i} v'\colon B\rangle ::= \langle u'(S')v'\rangle$ (leva stran produkcije q je torej izomorfna levi strani produkcije p).
- Grafa S in S' je možno poenotiti.
- Obstaja unifikacijska preslikava $h\colon Rhs[p] \to Rhs[q]$, tako da velja $h(u) = u'$ in $h(v) = v'$.

Potem je *b-posplošitev* produkcij p in q množica, ki vsebuje sledeče:

- produkcijo $r_0\colon \langle u''\colon A \xrightarrow{e''\colon \#i} v''\colon B\rangle ::= \langle u''(S'')v''\rangle$, kjer je $S'' = unif(S, S')$ in $Rhs[r_0] = \langle u''(S'')v''\rangle = unif(Rhs[p], Rhs[q]) = unif(\langle u(S)v\rangle, \langle u'(S')v'\rangle)$;
- po eno produkcijo $r_e\colon \langle w\colon P \xrightarrow{f\colon \#j} w'\colon Q\rangle ::= \langle w \xrightarrow{f'\colon l} w'\rangle$ za vsako povezavo $e \in \mathcal{E}[Rhs[r_0]]$ z lastnostmi $label(conn(e)) = \{P, Q\}$, $label(e) = \#j$ in $label(g(e)) = l \vee label(g'(e)) = l$ (pri $l \neq \#j$), kjer sta $g\colon Rhs[r_0] \to Rhs[p]$ in $g'\colon Rhs[r_0] \to Rhs[q]$ unifikacijski preslikavi (pri podani povezavi $e \in \mathcal{E}[Rhs[r_0]]$ sta $g(e)$ in $g'(e)$ povezavi v grafih $Rhs[p]$ in $Rhs[q]$, iz katerih je nastala povezava e v poenotenem grafu $Rhs[r_0]$).

Slika 4.9 prikazuje primer dvojice produkcij in njuno b-posplošitev. Produkcije z usmerjenimi povezavami obravnavamo na enak način kot produkcije z neusmerjenimi povezavami, le da namesto funkcije *conn* uporabljamo funkciji *source* in *target*.

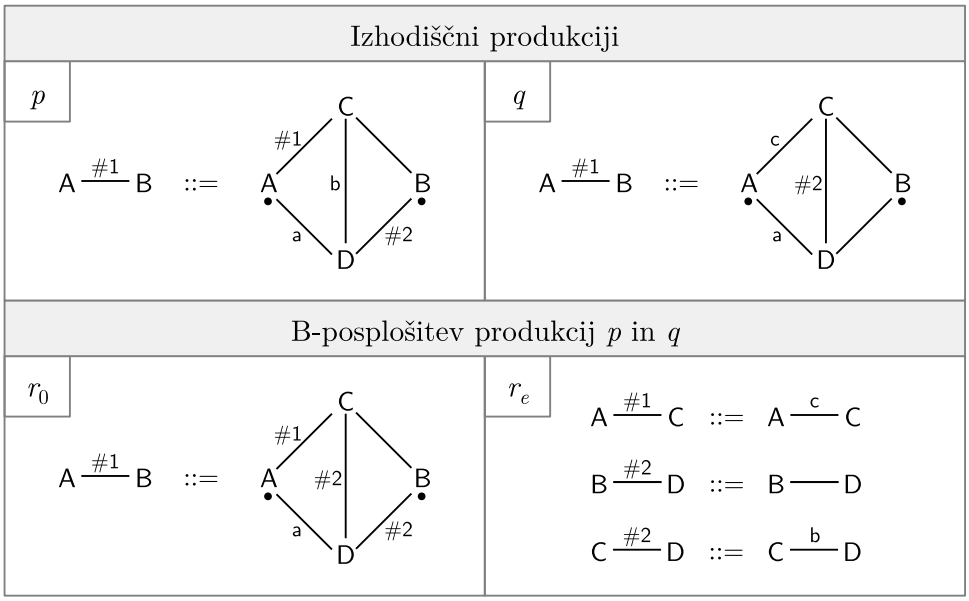
Gramatiko GG lahko b-posplošimo samo v primeru, če vsebuje najmanj eno dvojico produkcij, ki ustreza pogojem iz definicij 4.11 in 4.12. B-posplošitev gramatike GG pridobimo tako, da izberemo par produkcij, ki ju je mogoče b-posplošiti, nato pa izbrani produkciji nadomestimo z množico produkcij, ki je rezultat njune b-posplošitve.

Pokažimo, da b-posplošitev ne zmanjša splošnosti gramatike:

Trditev 4.13. Naj bo gramatika GG' b-posplošitev gramatike GG. Potem je gramatika GG' vsaj tako splošna kot gramatika GG.

Dokaz. Denimo, da je gramatika GG' nastala kot rezultat b-posplošitve produkcij p in q v gramatiki GG. Gramatika GG' torej namesto produkcij p in q vsebuje produkcijo r_0 in produkcije r_e, opredeljene z definicijama 4.11 in 4.12.

Trditev lahko dokažemo tako, da pokažemo, da je vsako uporabo produkcije p ali q (na nekem grafu) mogoče simulirati z uporabami produkcij r_0 in r_e. Od tod



Slika 4.9: *Produkciji in njuna b-posplošitev.*

namreč sledi, da je vsako izpeljavo v gramatiki GG mogoče simulirati z enakovredno izpeljavo v gramatiki GG', kar pomeni, da vsak graf, ki pripada jeziku gramatike GG, pripada tudi jeziku gramatike GG'.

Uporaba produkcije p na danem grafu nadomesti pojavitev grafa $Lhs[p]$ s kopijo grafa $Rhs[p]$. Ker sta levi strani produkcij p in r_0 izomorfni, lahko na pojavitvi grafa $Lhs[p]$ namesto produkcije p uporabimo produkcijo r_0. Ta korak zamenja pojavitev grafa $Lhs[p]$ s kopijo grafa $Rhs[r_0]$. Nastalo kopijo grafa $Rhs[r_0]$ pa lahko vedno preoblikujemo v graf, izomorfen grafu $Rhs[p]$, saj je graf $Rhs[r_0]$ definiran kot poenotenje grafov $Rhs[p]$ in $Rhs[q]$, množica produkcij r_e pa vsebuje produkcijo za vsako povezavo, ki je v grafu $Rhs[r_0]$ označena drugače kot pripadajoči povezavi v grafih $Rhs[p]$ in $Rhs[q]$. Torej lahko uporabo produkcije p simuliramo tako, da najprej uporabimo produkcijo r_0, nato pa na dodani kopiji grafa $Rhs[r_0]$ uporabljamo ustrezne produkcije r_e tako dolgo, dokler oznak povezav v dodani kopiji ne nastavimo na oznake pripadajočih povezav v grafu $Rhs[p]$. Na enak način bi simulirali tudi uporabo produkcije q. □

4.5.4 Iskanje i-podgrafov

Procedura poišči*NoveProdukcije* na sliki 4.7 za svoje delovanje potrebuje množico grafov, ki imajo vsaj po eno i-pojavitev v množici začetnih grafov trenutno obravnavane gramatike, in podatke o njihovih i-pojavitvah v tej množici. To nalogo opravlja procedura poišči*Podgrafe* na sliki 4.10. Procedura se omejuje na grafe z največ *maksŠtVozlišč* vozlišči.

Procedura poišči*Podgrafe* na začetku izdela množico vseh neizomorfnih grafov z enim samim vozliščem (skupaj z morebitnimi zankami na edinem vozlišču), ki v množici $\mathcal{G}$ vsebujejo vsaj po eno i-pojavitev. Za vsak izdelani graf S_1 si v množici $GoPo[S_1]$ zabeleži vse njegove i-pojavitve v množici $\mathcal{G}$. Procedura nato v vsakem

```
1  procedura poišči Podgrafe(𝒢, maksŠtVozlišč)
2      𝒮₁ := {S₁ | |𝒱[S₁]| = 1 ∧ graf S₁ ima vsaj eno i-pojavitev v 𝒢};
3      iz množice 𝒮₁ odstrani vse izomorfne dvojnike;
4      VzGoPo := ∅;
5      za vsak graf S₁ ∈ 𝒮₁ izvedi
6          GoPo[S₁] := {(G, P) | G ∈ 𝒢 ∧ graf P je i-pojavitev grafa S₁ v grafu G};
7          VzGoPo := VzGoPo ∪ {(S₁, GoPo[S₁])}
8      konec;
9      za vsak i ∈ 2 .. maksŠtVozlišč izvedi
10         𝒮ᵢ := ∅;
11         za vsak graf Sᵢ₋₁ ∈ 𝒮ᵢ₋₁ izvedi
12             za vsak par (G, P) ∈ GoPo[Sᵢ₋₁] izvedi
13                 za vsako vozlišče v ∈ Nh_G(P) izvedi
14                     P′ := P ∪ {v};
15                     P′ := P′ ∪ {e ∈ ℰ[G] \ ℰ[P′] | ∃w ∈ 𝒱[P′]: conn(e) = {v, w}};
16                     če ¬∃(G, Q) ∈ GoPo[Sᵢ₋₁]: Q = P′ potem
17                         če ∃S ∈ 𝒮ᵢ: grafa S in P′ sta izomorfna potem
18                             GoPo[S] := GoPo[S] ∪ {(G, P′)}
19                         sicer
20                             S := kopija grafa P′;
21                             𝒮ᵢ := 𝒮ᵢ ∪ {S};
22                             GoPo[S] := {(G, P′)}
23                         konec
24                     konec
25                 konec
26             konec
27         konec;
28         VzGoPo := VzGoPo ∪ {(Sᵢ, GoPo[Sᵢ]) | Sᵢ ∈ 𝒮ᵢ}
29     konec;
30     vrni VzGoPo
31 konec
```

Slika 4.10: *Iskanje grafov z vsaj eno i-pojavitvijo v dani množici grafov.*

obhodu glavne zanke (zanke po spremenljivki i) na podlagi grafov z $i-1$ vozlišči, ki imajo v vhodni množici $\mathcal{G}$ vsaj po eno i-pojavitev, zgradi množico takšnih grafov z i vozlišči in za vsak dobljeni graf S_i shrani vse njegove i-pojavitve v množico $GoPo[S_i]$. To doseže s širjenjem posameznih i-pojavitev posameznih grafov z $i-1$ vozlišči. Vsako i-pojavitev P razširi z vsakim posameznim sosednjim vozliščem v v njenem gostiteljskem grafu G (vrstica 14). Da ohrani induciranost i-pojavitve, mora procedura poleg vozlišča v vanjo dodati še vse povezave med vozliščem v in obstoječimi vozlišči i-pojavitve P, vključno z morebitnimi zankami na vozlišču v. Na ta način procedura izdela i-podgraf $P' \sqsubseteq P$ z i vozlišči (vrstica 15). Lahko se nam zgodi, da smo *isti* i-podgraf že nekoč ustvarili, le vrstni red obiskovanja njegovih elementov je bil drugačen. V tem primeru i-podgraf P' zavržemo (vrstica 16), saj podvajanja v množicah $GoPo$ povečujejo porabo časa in prostora. V vrstici 17 preverimo, ali je i-podgraf P' izomorfen kateremu od že izdelanih grafov z i vozlišči. Če takšen graf S obstaja, procedura zabeleži i-podgraf P' kot njegovo novo i-pojavitev, sicer pa ustvari nov graf kot kopijo i-podgrafa P' in podgraf P' zabeleži kot njegovo (do sedaj edino) i-pojavitev. Podatki o posameznih grafih z i vozlišči in njihovih i-pojavitvah v množici $\mathcal{G}$ se ob koncu vsakega obhoda glavne zanke shranijo v množico $VzGoPo$. Ta množica na koncu predstavlja izhod procedure.

Zaradi pogostega preverjanja izomorfnosti mora biti postopek za to opravilo računsko karseda učinkovit. Če vsak graf zapišemo s t.i. *kanoničnim nizom* [5], potem lahko preverjanje izomorfnosti prevedemo na preverjanje enakosti nizov. Niz, ki predstavlja graf, mora biti neobčutljiv na spremembe v oštevilčenju elementov grafa, saj lahko le tako zagotovimo, da bosta niza, ki pripadata izomorfnima grafoma, v vseh primerih enaka. To lahko dosežemo tako, da vsakemu grafnemu elementu priredimo kanonično zaporedno številko (številko, ki je odvisna samo od strukture grafa in oznak v njem, ne pa od izhodiščnega oštevilčenja elementov), nato pa oznake elementov skupaj z enoličnimi ločili zapišemo v niz v takšnem vrstnem redu, kot ga določajo kanonične številke. Žal pa izdelava kanoničnih nizov ne terja nič manj časa kot preverjanje izomorfizmov. K sreči pa v proceduri poišči Podgrafe ni nujno, da so nizi povsem neobčutljivi na oštevilčenje grafnih elementov. Zagotoviti moramo, da imajo neizomorfni grafi različne nize, v obratni smeri pa to ni obvezno: če izomorfna grafa proglasimo za neizomorfna, bomo s tem zgolj povečali porabo časa in prostora, druge škode pa ne bo. Namesto kanoničnih nizov se tako zadovoljimo s psevdokanoničnimi, seveda pa si tudi zanje želimo čimvečje neobčutljivosti na izhodiščno oštevilčenje grafnih elementov. Izkaže se, da lahko precejšnjo neobčutljivost dosežemo že s psevdokanoničnim oštevilčenjem, ki temelji na kombinaciji preprostih lokalnih lastnosti posameznih elementov, kot so oznaka elementa, število njegovih sosedov, oznake sosedov in že dodeljene psevdokanonične številke sosedov.

4.6 Eksperimentalni rezultati

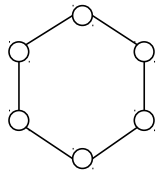
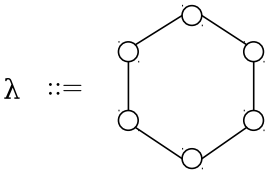
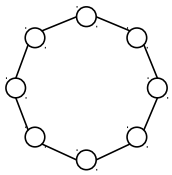
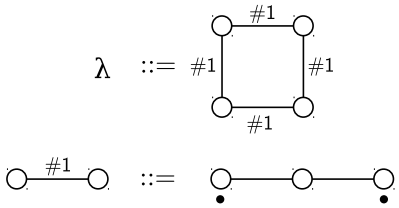
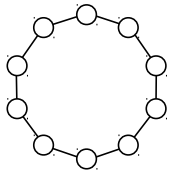
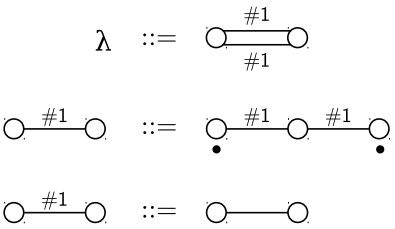
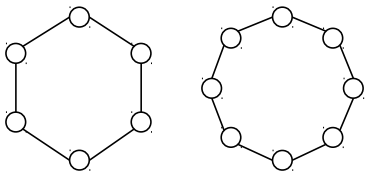
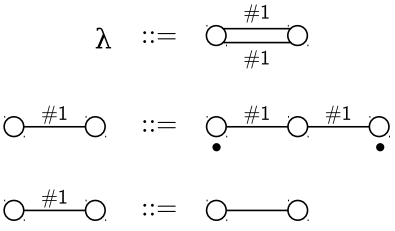
V tem podpoglavju predstavimo nekaj rezultatov poskusov z indukcijskim algoritmom. Grafne gramatike smo inducirali na podlagi preprostih ciklov (razdelek 4.6.1), na podlagi označenih linearnih grafov (razdelek 4.6.2), na podlagi dvojiških dreves (razdelek 4.6.3) in na podlagi strukturnih formul linearnih ogljikovodikov z enojnimi in dvojnimi vezmi (razdelek 4.6.4). Razdelek 4.6.5 je posvečen vprašanju časovne

zahtevnosti indukcijskega algoritma.

V vseh poskusih z indukcijskim algoritmom — razen v tistih, kjer je izrecno navedeno drugače — smo vrednosti parametrov procedure inducirajGramatiko (slika 4.5) določili kot $širinaSnopa = 10$ in $maksŠtJedrnihVozlišč = 5$.

4.6.1 Indukcija gramatike na podlagi preprostih ciklov

V prvem sklopu poskusov smo grafno gramatiko inducirali na podlagi preprostih n-ciklov. Preprost n-cikel je neoznačen neusmerjen graf z vozlišči v_1, v_2, ..., v_n in s povezavami v_1—v_2, v_2—v_3, ..., v_{n-1}—v_n, v_n—v_1. Pri vseh poskusih je bila negativna vhodna množica prazna; cikli so nastopali zgolj kot pozitivni vhodni grafi. Slika 4.11 prikazuje rezultate štirih izvedb indukcijskega algoritma. V prvem poskusu je pozitivna vhodna množica ($\mathcal{G}^+$) vsebovala zgolj 6-cikel, v drugem poskusu 8-cikel, v tretjem 10-cikel, v četrtem pa 6- in 8-cikel.

Vhodna množica $\mathcal{G}^+$	Inducirana gramatika	Velikost
		12
		14
		14
		14

Indukcijski algoritem je pri podani vhodni množici, ki je vsebovala zgolj 6-cikel, kot svoj rezultat vrnil trivialno gramatiko. Kot smo videli v podpoglavju 4.5, algoritem prične s trivialno gramatiko in jo začasno proglasi za trenutno najmanjšo ($GG_{\min}$). Če med svojim delovanjem ne tvori nobene manjše gramatike, potem trivialna gramatika ostane najmanjša. Natanko to se zgodi v primeru indukcije na podlagi 6-cikla. Trivialno gramatiko dobimo tudi pri indukciji na podlagi 7-cikla.

Pri indukciji na podlagi 8-cikla pa so razmere nekoliko drugačne. Velikost trivialne gramatike v tem primeru znaša 16 (8 vozlišč, 8 povezav), indukcijski algoritem pa je odkril (oz. tvoril) gramatiko velikosti 14. (Velikost prve produkcije znaša 8, velikost druge pa 6, saj sta stražarja skupna obema stranema produkcije, zato ju je treba šteti samo enkrat.) Žal pa je inducirana gramatika povsem enakovredna trivialni, saj ne pokriva ničesar drugega kot 8-cikel. Gre zgolj za kompaktnejši zapis trivialne gramatike. Podoben rezultat dobimo tudi pri indukciji na podlagi 9-cikla. Začetni graf inducirane gramatike je v tem primeru 3-cikel, pri katerem so oznake vseh povezav enake #1, produkcija tipa II pa povezavo z oznako #1 zamenja z dvema vozliščema in tremi povezavami. Tudi ta gramatika je zgolj prepis trivialne gramatike.

Če indukcijskemu algoritmu na vhodu podamo n-cikel z $n \geq 10$, pa na izhodu dobimo gramatiko, ki pokriva *vse* n-cikle od $n = 2$ naprej. Nedvomno lahko trdimo, da jezik, ki ga generira izhodna gramatika, smiselno posplošuje vhodni graf. Prva produkcija gramatike ustvari nekončno označen 2-cikel, druga omogoča poljubno povečevanje cikla, tretja produkcija pa odstrani nekončno oznako posamezne povezave. Enako gramatiko dobimo tudi v primeru, če vhodna množica vsebuje (denimo) 6-cikel in 8-cikel, oziroma nasploh tedaj, ko je vhodna množica dovolj velika, da splošna gramatika ciklov postane najmanjša kandidatna gramatika. Splošna gramatika dejansko nastane tudi pri indukciji samo na podlagi 6-cikla in pri indukciji samo na podlagi 8-cikla, vendar je s svojo velikostjo 14 prevelika, da bi lahko »izpodrinila« trivialno gramatiko oziroma njeno kompaktnejšo različico. Pri večjih vhodih pa splošna gramatika ciklov postane najmanjša v indukcijskem prostoru, zato jo algoritem vrne kot svoj rezultat.

4.6.2 Indukcija gramatike na podlagi označenih linearnih grafov

Rezultate drugega sklopa poskusov prikazuje slika 4.12. Tokrat smo grafne gramatike inducirali na podlagi linearnih grafov z označenimi vozlišči in neoznačenimi povezavami. Najprej smo gramatiko inducirali zgolj na osnovi grafa G_1; negativna vhodna množica je bila prazna. Graf G_1 je dovolj velik, da indukcijski algoritem kot svoj rezultat ni vrnil trivialne gramatike, ampak bistveno splošnejšo gramatiko. Izhodna gramatika GG_1 namreč generira neskončen jezik {A−B−A, A−B−A−B−A, A−B−A−B−A−B−A, ...}, kar je nedvomno smiselna posplošitev vhodnega grafa.

Slika 4.13 prikazuje korake pri tvorbi gramatike GG_1, ki nastane pri indukciji na podlagi grafa G_1. Upoštevajmo, da slika 4.13 prikazuje samo korake, ki neposredno vodijo od grafa G_1 do gramatike GG_1; stranske veje poteka algoritma niso prikazane. Indukcijski algoritem kot vedno prične s trivialno gramatiko ($GG_1^{(0)}$). Ena od možnih a-posplošitev gramatike $GG_1^{(0)}$ je gramatika $GG_1^{(1)}$, ki nastane z

Vhodni grafi	
G_1	A — B — A — B — A — B — A — B — A — B — A
G_2	A — B — B — A — B — B — A — B — B — A
G_3	A — B — B — B — A

Rezultati indukcije		
Množica $\mathcal{G}^+$	Množica $\mathcal{G}^-$	Inducirana gramatika
$\{G_1\}$	(prazna)	GG_1 $\lambda ::= \mathsf{A} \overset{\#1}{—} \mathsf{A}$ $\mathsf{A} \overset{\#1}{—} \mathsf{A} ::= \underset{\bullet}{\mathsf{A}} \overset{\#1}{—} \mathsf{A} \overset{\#1}{—} \underset{\bullet}{\mathsf{A}}$ $\mathsf{A} \overset{\#1}{—} \mathsf{A} ::= \underset{\bullet}{\mathsf{A}} — \mathsf{B} — \underset{\bullet}{\mathsf{A}}$
$\{G_1,\ G_2\}$	(prazna)	GG_{12} $\lambda ::= \mathsf{A} — \mathsf{B} \overset{\#1}{—} \mathsf{B} — \mathsf{A}$ $\mathsf{B} \overset{\#1}{—} \mathsf{B} ::= \underset{\bullet}{\mathsf{B}} \overset{\#1}{—} \mathsf{B} \overset{\#1}{—} \underset{\bullet}{\mathsf{B}}$ $\mathsf{B} \overset{\#1}{—} \mathsf{B} ::= \underset{\bullet}{\mathsf{B}} — \mathsf{A} — \underset{\bullet}{\mathsf{B}}$ $\mathsf{B} \overset{\#1}{—} \mathsf{B} ::= \mathsf{B} — \mathsf{B}$
$\{G_1,\ G_2\}$	$\{G_3\}$	$GG_{12/3}$ $\lambda ::= \mathsf{A} \overset{\#1}{—} \mathsf{B} \overset{\#1}{—} \mathsf{A}$ $\mathsf{A} \overset{\#1}{—} \mathsf{B} ::= \underset{\bullet}{\mathsf{A}} \overset{\#1}{—} \mathsf{B} \overset{\#1}{—} \underset{\bullet}{\mathsf{B}}$ $\mathsf{B} \overset{\#1}{—} \mathsf{B} ::= \underset{\bullet}{\mathsf{B}} — \mathsf{B} \overset{\#1}{—} \underset{\bullet}{\mathsf{B}}$ $\mathsf{B} \overset{\#1}{—} \mathsf{B} ::= \underset{\bullet}{\mathsf{B}} — \mathsf{A} — \underset{\bullet}{\mathsf{B}}$ $\mathsf{A} \overset{\#1}{—} \mathsf{B} ::= \mathsf{A} — \mathsf{B}$

Slika 4.12: *Indukcija grafnih gramatik na podlagi označenih linearnih grafov.*

dodatkom produkcije p_2. Desna stran te produkcije se večkrat pojavi v začetnem grafu gramatike $GG_1^{(0)}$ (v grafu $Rhs[p_1^{(0)}]$), zato je produkcija $p_1^{(1)}$, ki je posledica obratnih uporab dodane produkcije p_2 na grafu $Rhs[p_1^{(0)}]$, precej manjša od produkcije $p_1^{(0)}$. Gramatika GG_1 nastane kot ena od možnih a-posplošitev gramatike $GG_1^{(1)}$, vendar pa slika 4.13 zaradi boljše razumljivosti prikazuje še vmesni korak pri a-posploševanju gramatike $GG_1^{(1)}$. Gramatika $GG_1^{(2)}$ je pridobljena z dodatkom produkcije p_3 in z enkratno obratno uporabo te produkcije na grafu $Rhs[p_1^{(1)}]$, pri čemer se produkcija $p_1^{(1)}$ pretvori v produkcijo $p_1^{(2)}$. Ko algoritem dodano produkcijo p_3 na grafu $Rhs[p_1^{(1)}]$ obratno uporabi tolikokrat, kolikor je mogoče, nastane gramatika GG_1, ki je končni rezultat a-posplošitve gramatike $GG_1^{(1)}$. Te gramatike ni več mogoče niti a-posplošiti niti b-posplošiti. Ker je najmanjša med vsemi gramatikami, ki jih je indukcijski algoritem tvoril med svojim delovanjem (tudi med tistimi, ki na sliki 4.13 niso prikazane), jo algoritem vrne kot svoj rezultat.

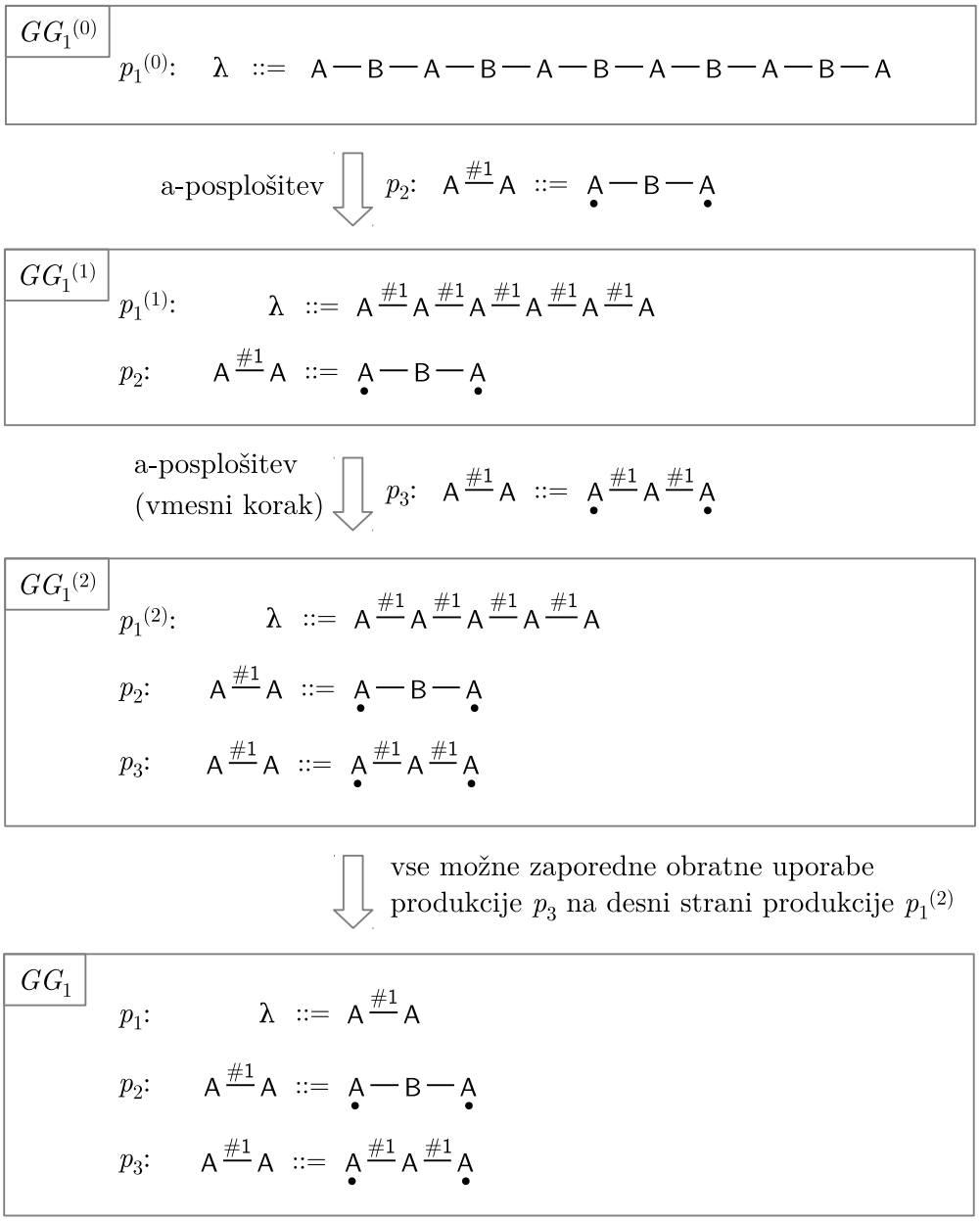
Vrnimo se k sliki 4.12. V drugem indukcijskem poskusu je bila množica negativnih vhodnih grafov ($\mathcal{G}^-$) še vedno prazna, množica pozitivnih vhodnih grafov ($\mathcal{G}^+$) pa je poleg grafa G_1 vsebovala še graf G_2. Ta graf ne pripada jeziku gramatike GG_1, saj vsebuje podzaporedja dveh vozlišč B, ki v grafih iz jezika $L(GG_1)$ niso možna. Gramatika GG_{12}, ki jo je na podlagi množice $\mathcal{G}^+ = \{G_1, G_2\}$ zgradil algoritem, posplošuje oba vhodna grafa, saj je njen jezik sestavljen iz vseh linearnih grafov s sledečimi lastnostmi:

- graf je sestavljen iz vozlišč z oznakami A in B in iz neoznačenih povezav;
- graf vsebuje najmanj 4 vozlišča;
- na obeh koncih verige vozlišč nastopata vozlišči A;
- veriga ne vsebuje nobenega podzaporedja dveh ali več vozlišč A;
- veriga lahko vsebuje poljubno dolga podzaporedja vozlišč B.

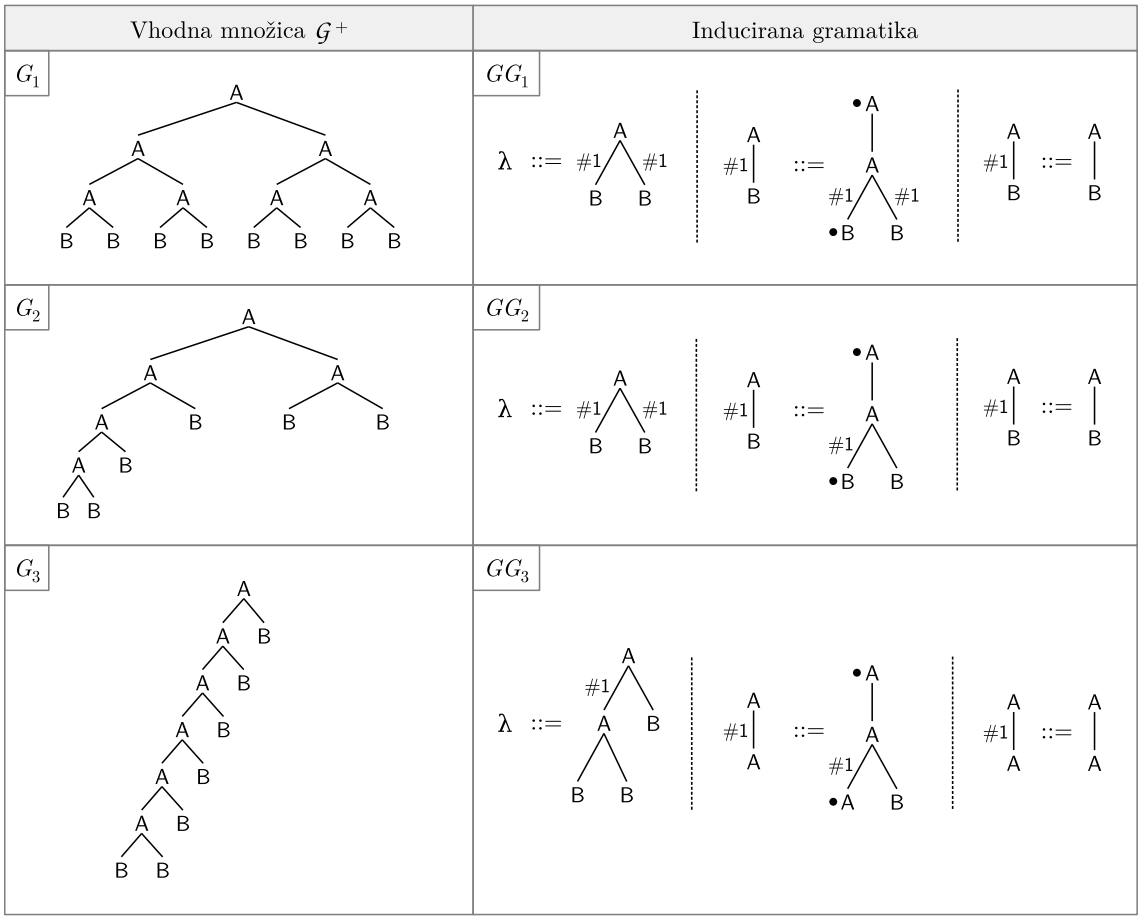
V tretjem poskusu smo ohranili množico pozitivnih vhodnih grafov, v množico negativnih grafov pa smo postavili zgolj graf G_3. Graf G_3 pripada jeziku gramatike GG_{12}, zato bo ta gramatika v indukcijskem postopku sicer nastala, vendar pa jo bo algoritem zaradi nekonsistentnosti z množico $\mathcal{G}^-$ takoj zavrgel. Rezultat indukcije je v tem primeru gramatika $GG_{12/3}$, ki generira podjezik jezika gramatike GG_{12}. Lahko preverimo, da jezik $L(GG_{12/3})$ ne vsebuje grafa G_3, sicer pa zanj veljajo iste splošne lastnosti kot za jezik $L(GG_{12})$.

4.6.3 Indukcija gramatike na podlagi dvojiških dreves

V tretjem sklopu poskusov smo gramatiko inducirali na podlagi dvojiških dreves, pri katerih so vsa notranja vozlišča nosila oznako A, vsi listi oznako B, vsako notranje vozlišče pa je imelo *natanko* dva otroka. V vseh primerih smo indukcijskemu algoritmu na vhodu podali eno samo dvojiško drevo, negativna vhodna množica pa je bila vsakokrat prazna. Nekaj rezultatov je prikazanih na sliki 4.14.



Slika 4.13: *Potek indukcije gramatike GG_1 s slike 4.12.*



Slika 4.14: *Indukcija grafnih gramatik na podlagi dvojiških dreves.*

Dobljene rezultate lahko označimo kot uspeh indukcijske metode. Na podlagi drevesa G_1 je algoritem induciral povsem splošno gramatiko dvojiških dreves. Gramatika GG_1 namreč pokriva vsa dvojiška drevesa razen drevesa brez notranjih vozlišč, poleg tega pa ne pokriva ničesar drugega. Od gramatike GG_{BT} na sliki 3.2 (stran 54) se razlikuje samo po odsotnosti produkcije $\lambda ::= \mathsf{B}$. Če bi želeli zajeti še to produkcijo, bi morali drevo brez notranjih vozlišč podati kot del pozitivne vhodne množice.

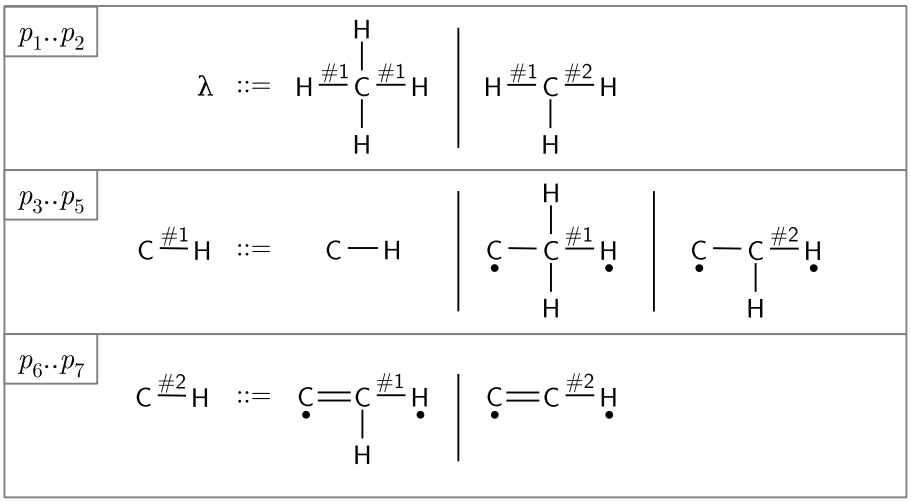
Indukcijski algoritem je pri večini podanih dreves kot svoj rezultat vrnil splošno gramatiko dvojiških dreves. Izjemo so seveda predstavljala drevesa, pri katerih skupno število vozlišč in povezav ni presegalo velikosti splošne gramatike, saj so bile v takšnih primerih trivialne gramatike vsaj tako majhne kot splošna gramatika. Drugi primer izjeme predstavlja gramatika GG_2 na sliki 4.14. Gramatika GG_2 ne pokriva celotne množice dvojiških dreves, saj v produkciji tipa II manjka ena nekončna oznaka. Vhodno drevo G_2 je namreč v nekaterih vejah preplitvo, zato indukcijski algoritem ni mogel izvesti vseh a-posplošitvenih korakov, potrebnih za indukcijo splošne gramatike.

Indukcijski algoritem se je izkazal kot odporen na različne primere neizrojenih vhodnih dreves in na različne vrednosti parametrov *širinaSnopa* in *maksŠtJedrnihVozlišč* (pri pogoju *maksŠtJedrnihVozlišč* ≥ 2).

Gramatika GG_3 s slike 4.14 je rezultat indukcije na podlagi izrojenega (linearnega) dvojiškega drevesa. Kot bi lahko pričakovali, algoritem v tem primeru inducira splošno gramatiko izrojenih dreves.

4.6.4 Indukcija gramatike na podlagi kemijskih strukturnih formul

Indukcijski algoritem smo preizkusili tudi nad grafi kemijskih spojin. Poskušali smo inducirati gramatiko za jezik, ki ga bomo v nadaljevanju krajše označevali z zapisom LHC12. Gre za jezik strukturnih formul linearnih ogljikovodikov z enojnimi in dvojnimi vezmi. To so spojine ogljikovih in vodikovih atomov, v katerih vsi ogljikovi atomi (vozlišča C) tvorijo povezano verigo. Ogljikovi atomi so med seboj lahko povezani samo z enojnimi ali dvojnimi vezmi. Vsak ogljikov atom tvori natanko štiri vezi (pri čemer dvojna vez šteje kot dve vezi), vsak vodikov atom (vozlišče H) pa natanko eno. Referenčna gramatika za ta jezik je predstavljena na sliki 4.15.



Slika 4.15: *Referenčna gramatika za jezik strukturnih formul ogljikovodikov z enojnimi in dvojnimi vezmi.*

Opisani indukcijski problem se je izkazal za bistveno trši oreh kot problemi iz predhodnih sklopov poskusov. Zahtevo po indukciji gramatike, ki generira *izključno* jezik LHC12, smo morali nekoliko omiliti. Zadovoljili smo se z gramatiko, ki sicer generira izključno veljavne strukturne formule ogljikovodikov, omejitev na *linearne* ogljikovodike pa smo umaknili. Poskusi so pokazali, da ciljne gramatike najverjetneje ni mogoče inducirati samo na podlagi pozitivnih vhodnih grafov. Poleg tega — v nasprotju s poskusi iz predhodnih razdelkov tega podpoglavja — želene gramatike ni bilo mogoče inducirati na podlagi skoraj poljubnih vhodnih grafov. V iskanje dvojice množic, na podlagi katerih bi bilo mogoče inducirati gramatiko z želenimi lastnostmi, smo zato morali vložiti nekoliko več truda.

Za potrebe tega razdelka naj zapis $\mathcal{H}_i$ označuje množico vseh grafov iz jezika LHC12, ki vsebujejo natanko i vozlišč C. Množica $\mathcal{H}_1$ tako vsebuje zgolj graf metana, množica $\mathcal{H}_2$ graf etana in etena itd.

Da bi ugotovili, ali je gramatiko jezika LHC12 sploh mogoče inducirati, in da bi (v primeru pozitivnega odgovora na to vprašanje) odkrili nabor grafov, ki vodijo do indukcije takšne gramatike, smo najprej pripravili množico 42 pozitivnih grafov ($\mathcal{G}_0^+$) in množico 200 negativnih grafov ($\mathcal{G}_0^-$). Množico $\mathcal{G}_0^+$ smo sestavili kot unijo $\mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{H}_3 \cup \mathcal{H}_4 \cup \mathcal{H}_5 \cup \mathcal{H}_6$, množico $\mathcal{G}_0^-$ pa smo pridobili tako, da smo 200-krat ponovili sledeči postopek:

(1) naključno izberi graf iz množice $\mathcal{H}_1 \cup \mathcal{H}_2 \cup \mathcal{H}_3 \cup \mathcal{H}_4$;

(2) izbranemu grafu naključno odstrani eno ali dve vozlišči H;

(3) dobljeni graf dodaj v množico $\mathcal{G}_0^-$.

Grafi, pridobljeni po opisanem postopku, zanesljivo ne pripadajo jeziku LHC12, saj v vsakem od njih vsaj eno vozlišče C tvori manj kot štiri vezi. Izbrani velikosti množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$ nimata nobenega posebnega pomena. Za potrebe našega poskusa smo v izhodišču potrebovali zgolj dovolj veliki množici pozitivnih in negativnih grafov.

Zanimalo nas je, ali obstajata (po možnosti čim manjši) podmnožici $\mathcal{S}^+ \subseteq \mathcal{G}_0^+$ in $\mathcal{S}^- \subseteq \mathcal{G}_0^-$, tako da bi grafna gramatika, inducirana na njuni podlagi, pokrivala vse grafe iz množice $\mathcal{G}_0^+$ in nobenega iz množice $\mathcal{G}_0^-$. Množici $\mathcal{S}^+$ in $\mathcal{S}^-$ smo pridobili s preprosto hevristično proceduro poiščiPopolniPodmnožici, ki jo prikazuje slika 4.16. Procedura prične z množico $\mathcal{S}^+$, ki vsebuje izključno najmanjši graf iz množice $\mathcal{G}_0^+$ (to je graf metana), in s prazno množico $\mathcal{S}^-$, nato pa vsakokrat inducira gramatiko GG na podlagi trenutnih množic $\mathcal{S}^+$ in $\mathcal{S}^-$. Če inducirana gramatika GG pravilno klasificira vse grafe iz množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$ (če torej velja $G \in L(GG)$ za vse grafe $G \in \mathcal{G}_0^+$ in $G \notin L(GG)$ za vse grafe $G \in \mathcal{G}_0^-$), potem množici $\mathcal{S}^+$ in $\mathcal{S}^-$ zadoščata podani zahtevi, zato ju procedura vrne kot rezultat. V nasprotnem primeru pa procedura poveča bodisi množico $\mathcal{S}^-$ ali množico $\mathcal{S}^+$, ponovno inducira gramatiko na podlagi posodobljenih množic $\mathcal{S}^+$ in $\mathcal{S}^-$ ter inducirano gramatiko preveri na množicah $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$. Pri povečevanju množic $\mathcal{S}^+$ in $\mathcal{S}^-$ uporabljamo preprosto hevristiko: če trenutna gramatika GG napačno klasificira vsaj en graf iz množice $\mathcal{G}_0^-$, potem v množico $\mathcal{S}^-$ dodamo najmanjši napačno klasificiran graf iz množice $\mathcal{G}_0^-$, sicer pa v množico $\mathcal{S}^+$ dodamo najmanjši napačno klasificiran graf iz množice $\mathcal{G}_0^+$. Na ta način lahko upamo, da bo inducirana gramatika GG čedalje bolje klasificirala grafe iz množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$. Razlog za postopno dodajanje napačno klasificiranih grafov in za dodajanje najmanjših tovrstnih grafov je v težnji po odkritju čim manjših množic $\mathcal{S}^+$ in $\mathcal{S}^-$ z iskano lastnostjo.

Izkazalo se je, da obstajata podmnožici $\mathcal{S}^+ \subseteq \mathcal{G}_0^+$ in $\mathcal{S}^- \subseteq \mathcal{G}_0^-$, ki vodita do inducirane gramatike, konsistentne z množicama $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$. Množici $\mathcal{S}^+$ in $\mathcal{S}^-$, ki ju je kot rezultat vrnila procedura poiščiPopolniPodmnožici, sta prikazani na sliki 4.17. Gramatika, inducirana na podlagi teh množic $\mathcal{S}^+$ in $\mathcal{S}^-$, je prikazana na sliki 4.18. To gramatiko bomo v nadaljevanju označevali kot GG_{ind}.

Inducirana gramatika GG_{ind} ne klasificira pravilno zgolj grafov iz množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$, ampak izpolnjuje tudi zahteve, ki smo jih predpisali na začetku tega razdelka. Gramatika GG_{ind} namreč pokriva vse grafe jezika LHC12, poleg tega pa vsi grafi, ki jih gramatika pokriva, predstavljajo veljavne (čeprav ne nujno linearne) ogljikovodike z enojnimi in dvojnimi vezmi. Pokritost celotnega jezika LHC12 lahko dokažemo z matematično indukcijo po dolžini verige vozlišč C: najprej preverimo,

```
1  procedura poišči_Popolni_Podmnožici(G_0^+, G_0^-, širinaSnopa, maksŠtJedrnihVozlišč)
2      S^+ := {najmanjši graf iz množice G_0^+};
3      S^- := ∅;
4      GG := inducirajGramatiko(S^+, S^-, širinaSnopa, maksŠtJedrnihVozlišč);
5      Z^+ := {G ∈ G_0^+ | GG ne pokriva grafa G};
6      Z^- := {G ∈ G_0^- | GG pokriva graf G};
7      dokler (Z^+ ≠ ∅) ∨ (Z^- ≠ ∅) izvajaj
8          če Z^- ≠ ∅ potem
9              S^- := S^- ∪ {najmanjši graf iz množice Z^-}
10         sicer
11             S^+ := S^+ ∪ {najmanjši graf iz množice Z^+}
12         konec;
13         GG := inducirajGramatiko(S^+, S^-, širinaSnopa, maksŠtJedrnihVozlišč);
14         Z^+ := {G ∈ G_0^+ | GG ne pokriva grafa G};
15         Z^- := {G ∈ G_0^- | GG pokriva graf G}
16     konec;
17     vrni (S^+, S^-)
18 konec
```

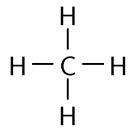
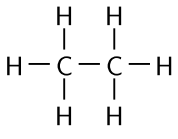
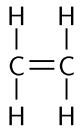
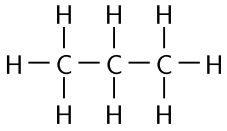
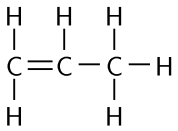
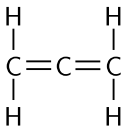
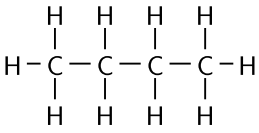
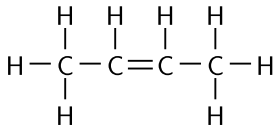
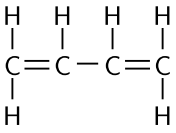
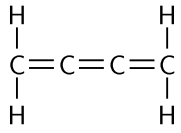
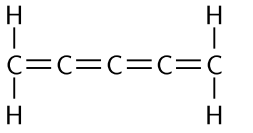
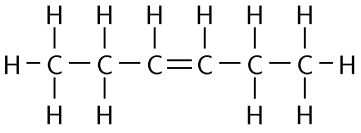
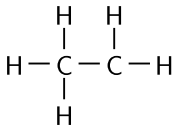
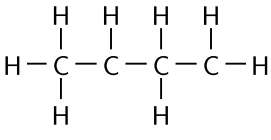
Slika 4.16: *Postopek iskanja podmnožic $\mathcal{S}^+ \subseteq \mathcal{G}_0^+$ in $\mathcal{S}^- \subseteq \mathcal{G}_0^-$, pri katerih je inducirana gramatika konsistentna z množicama $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$.*

ali jeziku gramatike GG_{ind} pripada edini graf z enim samim vozliščem C (tj. graf metana), nato pa ob predpostavki, da jeziku pripadajo vsi grafi z n vozlišči C, pokažemo, da to velja tudi za grafe z $n+1$ vozlišči. Dokaza trditve, da gramatika GG_{ind} pokriva samo veljavne grafe ogljikovodikov, pa se lahko lotimo tako, da pokažemo, da vsako vozlišče vsakega grafa iz jezika gramatike GG_{ind} tvori predpisano skupno število povezav s sosednjimi vozlišči (štiri v primeru vozlišč C in eno v primeru vozlišč H). To sledi iz enostavno preverljivega dejstva, da se vsak podgraf $\mathsf{C}\overset{\#1}{-}\mathsf{H}$ z uporabo produkcij gramatike GG_{ind} razširi v podgraf $\mathsf{C}-(\ldots)-\mathsf{H}$ in da se vsak podgraf $\mathsf{C}\overset{\#2}{-}\mathsf{H}$ razširi v podgraf $\mathsf{C}=(\ldots)-\mathsf{H}$. Gramatika GG_{ind} je tudi dokaj majhna, saj njena velikost znaša 59, medtem ko velikost referenčne gramatike za jezik LHC12 (slika 4.15) znaša 54.

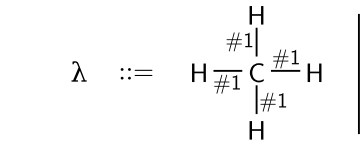
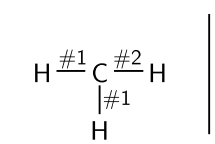
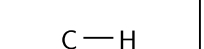
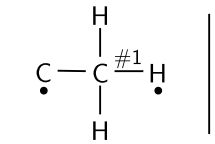
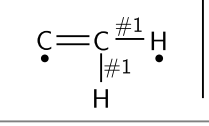
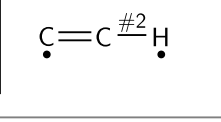
Indukcijski algoritem se je pri indukciji gramatike na podlagi množic $\mathcal{S}^+$ in $\mathcal{S}^-$ s slike 4.17 izkazal kot odporen na različne vrednosti parametrov *širinaSnopa* in *maksŠtJedrnihVozlišč* (pri pogoju *maksŠtJedrnihVozlišč* ≥ 3). Vrednosti 1 in 2 za parameter *maksŠtJedrnihVozlišč* ne moreta privesti do veljavne gramatike jezika LHC12, saj, denimo, v grafu linearnega alkana (ogljikovodika s samimi enojnimi vezmi) ne obstaja noben i-podgraf z manj kot tremi vozlišči, ki bi imel natanko dva soseda.

4.6.5 Računska zahtevnost

Količina časa in prostora, ki ju pri svojem delovanju porabi indukcijski algoritem, je v najslabšem primeru eksponentno odvisna od velikosti vhodnih množic. Razlog je

Množica $\mathcal{S}^+$		
		
		
		
		
Množica $\mathcal{S}^-$		
		

Slika 4.17: *Množici $\mathcal{S}^+$ in $\mathcal{S}^-$, pridobljeni kot podmnožici izhodiščnih množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$ s pomočjo procedure* poišči*Popolni*Podmnožici*.*

$p_1..p_3$	λ ::=		
$p_4..p_6$	 ::=		
$p_7..p_8$	 ::=		

Slika 4.18: *Gramatika, inducirana na podlagi množic $\mathcal{S}^+$ in $\mathcal{S}^-$ s slike 4.17.*

tako v izčrpnem postopku iskanja i-podgrafov, ki je osnova za a-posploševanja kandidatnih gramatik, kot tudi v Rekers-Schürrovem sintaksnem analizatorju, ki kljub izboljšavam, predstavljenim v poglavju 3, v najslabšem primeru še vedno porabi eksponentno količino računskih virov. Računsko zahtevnost postopka za iskanje podgrafov lahko zmanjšamo za ceno neodkritja nekaterih podgrafov. Na primer, pristopi Jonyerja in sod. [49], Kukluka in sod. [56] ter Atesa in sod. [4] omejujejo število odkritih podgrafov in tako delujejo v polinomskem času in prostoru za ceno suboptimalnih rezultatov.

Tabela 4.1 prikazuje skupno število izdelanih gramatik v času delovanja indukcijskega algoritma (vključno z gramatikami, nekonsistentnimi z negativno vhodno množico) in skupno porabo časa pri indukciji gramatike na podlagi množic $\mathcal{S}^+$ in $\mathcal{S}^-$ s slike 4.17. Vse poskuse smo izvršili na računalniku z 1,86-gigaherčnim procesorjem Intel Core 2 Duo. Levi del tabele prikazuje rezultate pri različnih širinah snopa in pri fiksni zgornji meji števila jedrnih vozlišč ($maksŠtJedrnihVozlišč = 5$), desni del tabele pa prikazuje rezultate pri različnih zgornjih mejah števila jedrnih vozlišč in pri fiksni širini snopa ($širinaSnopa = 10$).

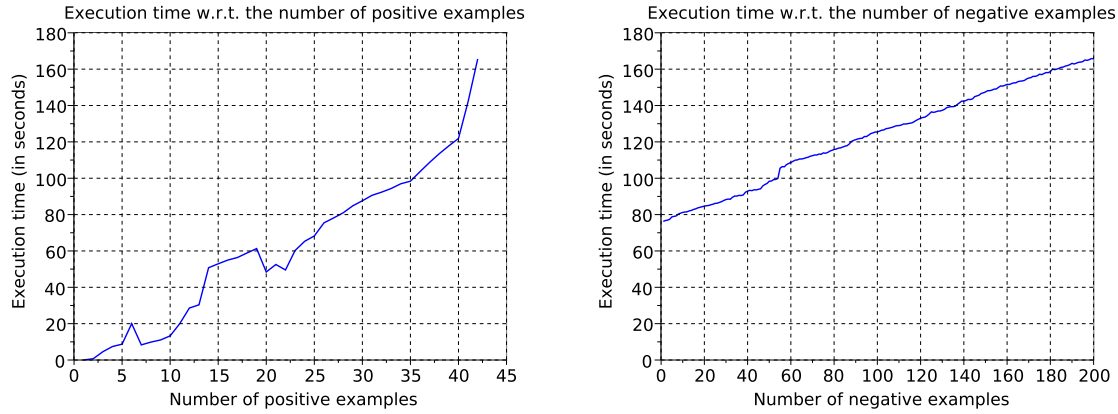
Tabela 4.1: *Število vseh izdelanih gramatik in skupna poraba časa pri indukciji gramatike na podlagi množic $\mathcal{S}^+$ in $\mathcal{S}^-$ s slike 4.17.*

	širinaSnopa				*maksŠtJedrnihVozlišč*			
	1	10	100	1000	3	5	7	9
število izdelanih gramatik	116	148	590	24 435	95	148	195	224
poraba časa (v sekundah)	6,8	7,2	12,2	370	3,4	7,2	12,4	17,5

Diagrama na sliki 4.19 prikazujeta odvisnost porabe časa indukcijskega algoritma (v sekundah) od števila vhodnih primerov. Tokrat smo grafne gramatike inducirali na podlagi množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$, ne na podlagi njunih podmnožic $\mathcal{S}^+$ in $\mathcal{S}^-$. Spomnimo se, da množica $\mathcal{G}_0^+$ vsebuje 42 pozitivnih, množica $\mathcal{G}^-$ pa 200 negativnih grafov.

Diagram na levi strani slike 4.19 smo pridobili tako, da smo število negativnih vhodnih grafov fiksirali na 200, število pozitivnih vhodnih grafov pa smo povečevali od 1 do 42, pri čemer smo grafe iz množice $\mathcal{G}_0^+$ v tekočo pozitivno vhodno množico dodajali po naraščajoči velikosti. Parameter *širinaSnopa* smo fiksirali na vrednost 10, parameter *maksŠtJedrnihVozlišč* pa na vrednost 5. Skupno smo tako izvedli 42 indukcijskih poskusov (najprej z najmanjšim grafom iz množice $\mathcal{G}_0^+$ in s celotno množico $\mathcal{G}_0^-$, nato z dvema najmanjšima grafoma iz množice $\mathcal{G}_0^+$ in s celotno množico $\mathcal{G}_0^-$ itd.). Vidimo, da poraba časa ne narašča povsem monotono s številom povečevanja pozitivnih vhodnih grafov, saj trajanje indukcije ni odvisno samo od števila ustvarjenih gramatik (ki se povečuje s številom pozitivnih vhodnih grafov), ampak tudi od »ugodnosti« izdelanih grafnih gramatik s stališča sintaksnega analizatorja, ki pa je vnaprej ni mogoče predvideti. Diagram na desni strani slike 4.19 smo pridobili na podoben način kot diagram na levi, le da smo število pozitivnih vhodnih grafov fiksirali na 42, število negativnih pa smo povečevali od 1 do 200.

Indukcijski algoritem je za indukcijo gramatike na podlagi celotnih množic $\mathcal{G}_0^+$ in $\mathcal{G}_0^-$ porabil slabe tri minute časa. Vendar pa lahko indukcijski algoritem kadarkoli



Slika 4.19: *Trajanje izvajanja indukcijskega algoritma v odvisnosti od števila vhodnih grafov.*

prekinemo in kot njegov rezultat uporabimo doslej najmanjšo kandidatno gramatiko (gramatiko $GG_{\min}$), saj bo zanesljivo konsistentna z vhodnima množicama. Algoritem sicer gradi čedalje manjše in praviloma tudi čedalje splošnejše gramatike $GG_{\min}$, vendar pa so prav vse konsistentne z vhodnima množicama.

4.7 Zaključek

V tem poglavju smo predstavili izviren pristop k indukciji grafnih gramatik. Pri podani neprazni množici pozitivnih grafov $\mathcal{G}^+$ in (po želji prazni) množici negativnih grafov $\mathcal{G}^-$ poskuša algoritem poiskati čim manjšo grafno gramatiko, ki pokriva vse grafe iz množice $\mathcal{G}^+$ in nobenega iz množice $\mathcal{G}^-$. Indukcijski algoritem preiskuje prostor gramatik, konsistentnih z vhodnima množicama, v smeri od najbolj specifične gramatike proti splošnejšim gramatikam. Algoritem posplošuje gramatike na dva izvirna načina. Cilj prvega načina je poiskati ponavljajoče se podgrafe v začetnih produkcijah gramatike in jih »izpostaviti« v obliki novih produkcij, cilj drugega načina posploševanja pa je varčnejši zapis parov podobnih produkcij. Algoritem si pri preverjanju konsistentnosti posameznih gramatik z vhodnima množicama pomaga z izboljšanim Rekers-Schürrovim sintaksnim analizatorjem.

Indukcijski algoritem je v nekaterih primerih pokazal presenetljivo vzpodbudne indukcijske sposobnosti. To lahko v precejšnji meri trdimo tudi za primer gramatike linearnih ogljikovodikov z enojnimi in dvojnimi vezmi. V prispevku za konferenco AGTIVE [38] smo predstavili tudi ugodne, čeprav ne idealne rezultate pri indukciji gramatike diagramov poteka. Kljub nekaterim nespornim uspehom indukcijske metode pa pri večjih grafnih množicah njena časovna zahtevnost kmalu postane problematična. Preliminarni preizkus algoritma na neki *realni* podatkovni bazi strukturnih formul kemijskih spojin (190 pozitivnih in 98 negativnih učnih grafov, od 50 do 100 vozlišč in od 50 do 100 povezav na graf) zaradi prevelike časovne zahtevnosti žal ni obrodil sadov.

Uporabnost algoritma bi potemtakem lahko povečali z zmanjšanjem časovne zah-

tevnosti. Tukaj se ponujata vsaj dve možnosti:

- V proceduri poišči Podgrafe bi se lahko omejili le na podgrafe, ki v množici začetnih grafov dane gramatike nastopajo dovolj pogosto. Na ta način bi bilo mogoče bistveno zmanjšati število izdelanih gramatik, tvegali pa bi izpustitev katere od gramatik, ki se kratkoročno ne zdi obetavna, dolgoročno pa bi lahko vodila do velikih prihrankov.

- Glede na to, da je ciljni formalizem indukcijskega algoritma znatno enostavnejši od splošnih kontekstno odvisnih grafnih gramatik, bi lahko poskusili Rekers-Schürrov sintaksni analizator obogatiti s kakšnimi *ad hoc* tehnikami, ki bi povečale računsko učinkovitost za gramatike iz ciljnega formalizma.

Indukcijski algoritem bi bilo možno izboljšati tudi kako drugače. V trenutni različici uporabljamo snopovni iskalni algoritem, ki je eden od najenostavnejših optimizacijskih pristopov. Žal pa se je pri poskusih pokazalo, da se snopovni iskalni postopek pogosto ujame v lokalnih minimumih. Indukcijski algoritem med svojim delovanjem praviloma zgradi veliko število medsebojno zelo podobnih in približno enako velikih grafnih gramatik. Vrsta, ki jo vzdržuje snopovni algoritem, se — tudi če je razmeroma velika — lahko hitro napolni z množico skoraj enakih gramatik, kar pomeni, da se snopovno iskanje dejansko prevede na (počasno) različico iskanja tipa »samo najboljši«. Zato bi bilo snopovno iskanje smiselno nadomestiti s kakšnim naprednejšim iskalnim pristopom, lahko pa bi vsaj, denimo, s postopkom simuliranega ohlajanja poskušali preprečevati prezgodnje nastajanje »monokulture« podobnih gramatik.

Ciljni formalizem indukcijskega algoritma je podoben povezavnim gramatikam, ki smo jih predstavili v podrazdelku 2.3.4.1. Desna stran produkcije tipa II je sestavljena iz dveh stražarjev, ki tvorita soseščino jedra produkcije. Če bi vsaka produkcija tipa II lahko vsebovala do vključno n stražarjev, ki bi tvorili soseščino jedra, bi dobili formalizem, podoben hiperpovezavnim gramatikam reda n (podrazdelek 2.3.4.3). Pri $n \geq 3$ so tovrstne gramatike močnejše od povezavnih. Hiperpovezave bi lahko predstavili z običajnimi vozlišči, njihove lovke pa z običajnimi povezavami, kot je to prikazano na sliki 2.8 (stran 34). Ogrodje indukcijskega algoritma bi lahko ostalo nedotaknjeno, spremeniti pa bi morali predstavitev grafov in produkcij ter določene podrobnosti, kot je npr. omejitev na obravnavo podgrafov z natanko dvema sosednjima vozliščema.

POGLAVJE

5

PRETVORBA METAMODELA V GRAFNO GRAMATIKO

V tem poglavju se ukvarjamo z uporabo grafnih gramatik na področju domensko specifičnega modeliranja. Besedilo temelji na članku za revijo *Software and Systems Modeling* [36].

Predstavljamo metodo za pretvorbo metamodela v obliki razrednega diagrama UML (Unified Modeling Language) v grafno gramatiko, katere jezik vsebuje natanko tiste modelske grafe (objektne diagrame UML), ki izpolnjujejo pravila vhodnega metamodela. Poglavitni smisel omenjene pretvorbe je pridobitev generativnega opisa vhodnega metamodela. Metamodeli namreč podajajo zgolj deklarativni opis modelske domene, saj opisujejo lastnosti, ki jim morajo veljavni modeli v dani domeni zadoščati, ne vsebujejo pa pravil, po katerih bi množico veljavnih modelov bilo mogoče sistematično graditi. Grafne gramatike podajajo prav takšna pravila, zato s pretvorbo metamodela v enakovredno grafno gramatiko pridobimo možnost sistematične tvorbe veljavnih modelov, kar nam lahko koristi, denimo, pri gradnji testnih primerov za modelske transformacije.

Od sorodnih pristopov k pretvorbi metamodelov v grafne gramatike se naša metoda razlikuje predvsem po izbiri izhodnega formalizma. Medtem ko si sorodni pristopi pomagajo z naprednimi, pogosto *ad hoc* tehnikami za opis grafnih gramatik, naša metoda uporablja preprost in jedrnat Rekers-Schürrov formalizem, ki ga lahko, kot smo opisali v podpoglavju 2.4, obravnavamo kot čisto grafno posplošitev standardnih kontekstno odvisnih tekstovnih gramatik. Pokazali bomo, da je metamodele (razredne diagrame UML) s poljubnimi multiplikativnostmi in dedovanjem mogoče pretvoriti v gramatike, ki uporabljajo zgolj gramatične simbole in kontekstno odvisne grafne produkcije. To je izvirni prispevek naše metode.

Poleg metode za pretvorbo metamodelov v enakovredne grafne gramatike bomo

v pričujočem poglavju predstavili tudi izvirno zamisel o semantični analizi modelov. Grafno gramatiko $GG(MM)$, ki jo pridobimo na podlagi danega metamodela MM, lahko opremimo s semantičnimi pravili na podoben način kot tekstovne gramatike pri gradnji prevajalnikov. Osnovo za semantično analizo nekega modela (objektnega diagrama UML), ki izpolnjuje pravila danega metamodela, predstavlja izpeljava modela v gramatiki $GG(MM)$. Če na izpeljavi uporabimo semantična pravila, s katerimi smo opremili gramatiko $GG(MM)$, pridobimo semantiko (»pomen«) danega modela.

V podpoglavju 5.1 problem pretvorbe motiviramo in neformalno predstavimo, poleg tega pa opredelimo naš prispevek. V podpoglavju 5.2 sledi pregled izbranih sorodnih del. Podpoglavje 5.3 formalno opredeli problem pretvorbe. Zaradi enostavnejše razlage je obravnava dedovanja ločena od obravnave multiplikativnosti. Tako je v podpoglavju 5.4 opisana pretvorba metamodelov s poljubnimi multiplikativnostmi, a brez dedovanja, metamodele z dedovanjem pa obravnavamo v podpoglavju 5.5. V podpoglavju 5.6 se ukvarjamo z računsko zahtevnostjo pretvornega postopka. Podpoglavje 5.7 je namenjeno povezavi med sintaksno analizo modelov in njihovo semantiko. V podpoglavju 5.8 obravnavamo nekatere elemente vhodnih metamodelov, ki smo jih v predhodnih podpoglavjih zaradi enostavnosti zanemarili. S podpoglavjem 5.9 bomo poglavje zaključili.

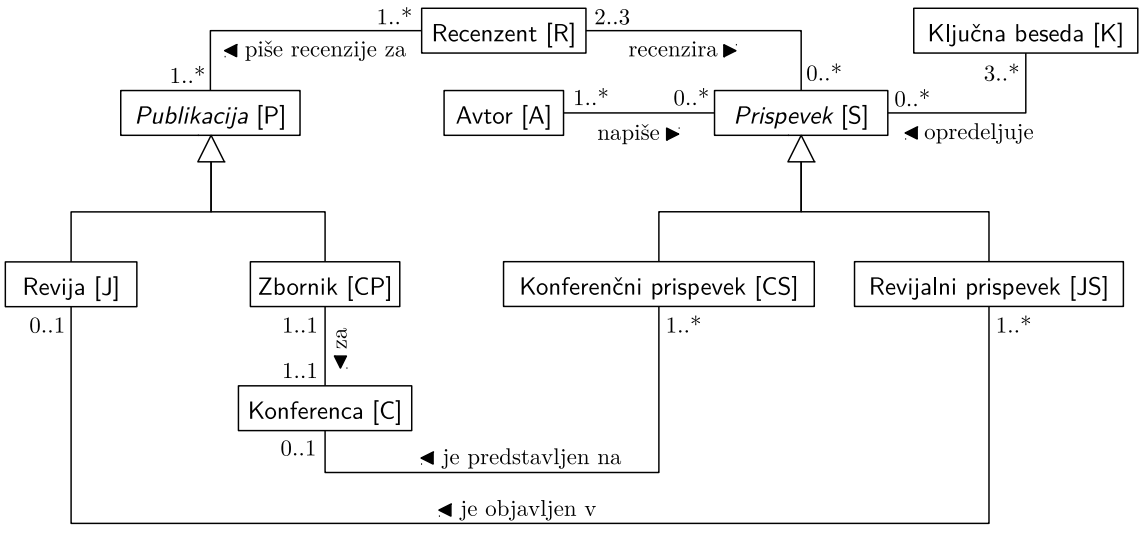
5.1 Uvod

V tem podpoglavju bomo najprej predstavili pojma *modela* in *metamodela* (razdelek 5.1.1), nato pa se bomo posvetili povezavi med metamodeli in grafnimi gramatikami (razdelek 5.1.2). Problem in naš pristop bomo predstavili v razmeroma neformalnem slogu; formalne definicije bodo sledile v kasnejših podpoglavjih. V razdelku 5.1.3 bomo opredelili naš prispevek k znanosti v okviru pričujočega poglavja.

5.1.1 Modeli in metamodeli

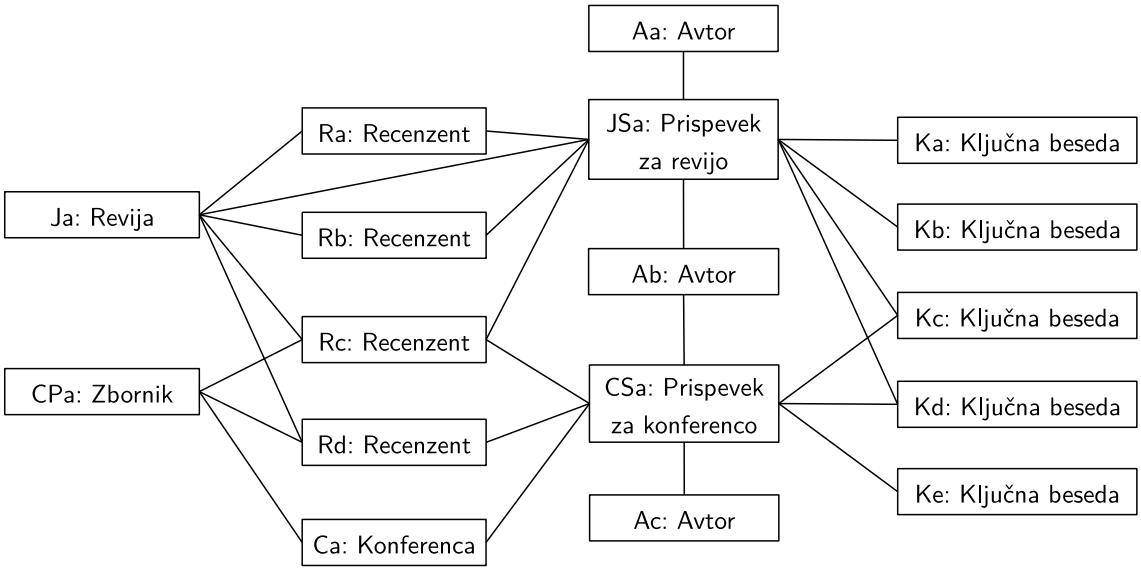
Pojem *model* na področju domensko specifičnega modeliranja [39, 51, 66, 89] predstavlja opis nekega stanja v neki domeni. Model v dani domeni je *veljaven*, če predstavlja opis veljavnega oziroma mogočega stanja v dani domeni; v nasprotnem primeru je model *neveljaven*. V nekaterih primerih lahko potencialno neskončno množico veljavnih modelov predstavimo s končnim opisom. Takšnemu opisu pravimo *metamodel*. Metamodel podaja množico pravil, ki jih morajo izpolnjevati vsi veljavni modeli v dani domeni. Za modele, ki so glede na podani metamodel veljavni, bomo rekli, da so *skladni* z metamodelom. Množico modelov, skladnih s podanim metamodelov, bomo imenovali *jezik* metamodela.

Priljubljen način za opis modelov in metamodelov je diagramski jezik UML [35]. V tem jeziku so modeli predstavljeni z *objektnimi diagrami*, metamodeli pa z *razrednimi diagrami*. Razredni diagram na sliki 5.1 predstavlja primer metamodela za domeno znanstvenega založništva, objektni diagram na sliki 5.2 pa predstavlja nek model, ki je skladen s tem metamodelom. Metamodel, ki ga predstavlja razredni diagram na sliki 5.1, bo služil kot tekoči primer. Za lažje sklicevanje ga bomo v nadaljevanju označevali z MM_{pub}.



Slika 5.1: MM_{pub}*: metamodel iz domene znanstvenega založništva.*

V nadaljevanju bomo predpostavili, da so modeli predstavljeni z objektnimi, metamodeli pa z razrednimi diagrami UML. Model je potemtakem graf, v katerem vozlišča predstavljajo posamezne *objekte* (angl. *objects*), povezave pa *vezi* (angl. *links*)



Slika 5.2: *Model, skladen z metamodelom MM_{pub} s slike 5.1.*

med posameznimi objekti. Objekti so primerki *razredov* (angl. *classes*), vezi pa primerki *asociacij* (angl. *associations*). Razredi in asociacije so določeni v metamodelu oziroma razrednem diagramu UML, ki ga bomo opisali v naslednjih odstavkih.

Metamodel je graf, v katerem vozlišča predstavljajo posamezne *razrede*. Razred predstavlja množico objektov istega tipa v veljavnem modelu. Metamodel MM_{pub} vsebuje 10 razredov. Njihova imena bomo zaradi jedrnatosti zapisovali z okrajšavami, ki so na sliki 5.1 prikazane v oglatih oklepajih ob polnih imenih razredov. Okrajšave izvirajo iz angleških poimenovanj razredov (prispevek — submission, zbornik — conference proceedings, revija — journal itd.).

Povezave v metamodelu predstavljajo razmerja med razredi. Povezave, ki so prikazane z neusmerjenimi daljicami, predstavljajo *asociacije*. Asociacija predstavlja množico vezi v veljavnem modelu. Napisi ob asociacijah opredeljujejo njihov pomen, smeri, nakazane s polnimi trikotniki, pa nam povedo, kako naj beremo napise. Metamodel MM_{pub} nam med drugim pove, da avtor napiše prispevek (asociacija med razredoma A in S), da je konferenčni prispevek predstavljen na konferenci (asociacija med razredoma CS in C) itd. Številske oznake ob krajiščih asociacij, imenovane *multiplikativnosti* (angl. *multiplicities*), določajo številostna razmerja med objekti posameznih razredov. Vsaka multiplikativnost je zapisana v obliki celoštevilskega intervala. Multiplikativnost na asociaciji med P in Q, ki je zapisana bližje okvirčku za razred Q, pove, s koliko objekti razreda Q mora v veljavnem modelu biti povezan vsak objekt razreda P. Na primer, multiplikativnost $2\,..\,3$ na asociaciji med razredoma S in R pove, da mora v veljavnem modelu biti vsak objekt razreda S povezan z najmanj dvema in največ tremi objekti razreda R. Z drugimi besedami: vsak prispevek morata recenzirati najmanj dva in največ trije recenzenti. Multiplikativnost oblike $a\,..\,*$ predstavlja navzgor neomejen interval. Na primer, multiplikativnost $0\,..\,*$ na asociaciji $\mathsf{A}{-}\mathsf{S}$ pove, da lahko vsak avtor napiše poljubno število prispevkov (vštevši nič).

Poleg povezav, opremljenih z multiplikativnostmi, lahko v metamodelih obsta-

jajo tudi povezave, ki se v enem od krajišč zaključijo s praznim trikotnikom. Takšne povezave opredeljujejo *dedna razmerja* (angl. *inheritance*) med razredi. V metamodelu MM_{pub} sta razreda CS in JS *podrazreda* (angl. *subclass*) razreda S, enako pa velja tudi za razreda J in CP v razmerju do razreda P. Podrazredi podedujejo vse lastnosti nadrazredov. Na primer, razreda CS in JS imata oba po štiri asociacije, in sicer tri podedovane (z razredi A, K in R) in po eno dodatno (z razredom C oziroma J).

Razredi, katerih imena so prikazana s poševno pisavo, so *abstraktni*, kar pomeni, da noben veljavni model ne more vsebovati njihovih objektov. Metamodel MM_{pub} vsebuje dva abstraktna razreda, P in S. Noben model, ki je skladen z metamodelom MM_{pub}, ne more vsebovati objektov razredov P in S, lahko pa vsebuje objekte njihovih podrazredov, saj ti niso abstraktni.

Ni težko preveriti, da je model na sliki 5.2 skladen z metamodelom MM_{pub}. Na primer, objekt JSa tipa JS je povezan z enim objektom tipa J (spoštovanje multiplikativnosti $0\,..\,1$ v smeri $\mathsf{JS} \to \mathsf{J}$), s tremi objekti tipa R (spoštovanje podedovane multiplikativnosti $2\,..\,3$ v smeri $\mathsf{S} \to \mathsf{R}$), z dvema objektoma tipa A (skladnost s podedovano multiplikativnostjo $1\,..\,*$ v smeri $\mathsf{S} \to \mathsf{A}$) in s štirimi objekti tipa K (skladnost s podedovano multiplikativnostjo $3\,..\,*$ v smeri $\mathsf{S} \to \mathsf{A}$).

Poleg asociacij, multiplikativnosti, dedovanja in abstraktnih razredov lahko pri metamodelih (oziroma razrednih diagramih UML) uporabimo še mnogo drugih elementov. Razrede lahko opremimo z atributi in metodami, ločimo več vrst povezav med razredi (poleg asociacije poznamo tudi kompozicijo in agregacijo), kompleksnejše omejitve lahko podamo s pravili v jeziku OCL (Object Constraint Language) [99] itd. Vendar pa se bomo v doktorski disertaciji osredotočili na bistvene elemente metamodelov, ki smo jih predstavili v tem razdelku. Pomen posameznih elementov bomo formalno opredelili v razdelku 5.3.2.

5.1.2 Metamodeli in grafne gramatike

Modeli so po naši predpostavki predstavljeni z objektnimi diagrami UML, torej z grafi, katerih vozlišča predstavljajo objekte, povezave pa vezi v modelu. Ker bi se radi izognili pretiranemu enačenju med abstraktnim konceptom (modelom) in njegovo konkretno predstavitvijo (grafom), kljub temu pa bi želeli poudarjati tesno povezavo med obema pojmoma, bomo graf, ki predstavlja nek model (veljaven ali neveljaven), imenovali *modelski graf*. Modeli so sestavljeni iz objektov in vezi, modelski grafi pa iz vozlišč in povezav, ki objekte in vezi predstavljajo.

Metamodel učinkovito in pregledno opisuje množico veljavnih modelov oziroma — ob predpostavki grafne predstavitve modelov — množico veljavnih modelskih grafov. Kljub temu pa mu lahko pripišemo pomanjkljivost: metamodel je zgolj *deklarativni*, ne pa tudi *generativni* formalizem, saj množico veljavnih modelskih grafov (tj. svoj jezik) le *opisuje*, ne ponuja pa nobenega neposrednega načina za samodejno *tvorbo* elementov množice. Možnost generiranja modelskih grafov, ki so skladni z danim metamodelom, bi nam, denimo, koristila pri testiranju modelskih transformacij ali različnih algoritmov na (meta)modelih [23, 30, 102]. Zato se nam s problemom samodejne tvorbe veljavnih modelov na podlagi danega metamodela splača ukvarjati.

Omejitve, ki so posledica deklarativnega značaja metamodelov, lahko presežemo z uporabo grafnih gramatik. Grafna gramatika je generativni formalizem, saj podaja pravila za gradnjo grafov, ki pripadajo njenemu jeziku. Metamodel in grafno gramatiko lahko obravnavamo kot dva različna formalizma za opis množic grafov: metamodel podaja pravila, ki jim mora množica veljavnih (modelskih) grafov zadoščati, grafna gramatika pa določa pravila za gradnjo veljavnih grafov. Možnost samodejne tvorbe modelskih grafov, skladnih s podanim metamodelom, lahko potemtakem pridobimo tako, da metamodel pretvorimo v grafno gramatiko, katere jezik je enak jeziku metamodela. Kot bomo videli, lahko grafna gramatika, pridobljena na podlagi metamodela, služi tudi kot osnova za semantično analizo ali transformacijo modelov [17, 32, 41, 83, 84], saj je produkcije grafnih gramatik možno opremiti s semantičnimi pravili na podoben način kot besedilne gramatike [1, 74].

S stališča domensko-specifičnega modeliranja je problem pretvorbe metamodela v grafno gramatiko torej zanimiv zaradi možnosti samodejne tvorbe modelov in vpeljave semantičnih pravil. Problem pa je zanimiv tudi s čisto teoretičnega vidika, saj se ukvarjamo z dvema bistveno različnima načinoma za opis množic grafov.

Po našem védenju so prvo metodo za pretvorbo metamodela v enakovredno grafno gramatiko predstavili Ehrig in sod. [30]. Njihov postopek pretvori metamodel v kontekstno odvisno grafno gramatiko, ki poleg kontekstno odvisnih produkcij temelji na *pogojih uporabe* (angl. *application conditions*) in *prednostnih razredih* (angl. *precedence classes*). Pogoji uporabe so mehanizem za omejevanje območja uporabe posameznih produkcij. Medtem ko je brez pogojev uporabe produkcijo mogoče uporabiti na poljubnem podgrafu, ki se p-homomorfno ujema z njeno levo stranjo (če ni kršen pogoj visečih povezav, gl. razdelek 2.4.2), lahko s pogoji uporabe uporabnost produkcij dodatno omejimo. Prednostni razredi podajajo vrstni red uporabe produkcij, in sicer v smislu, da je pri gradnji grafa na podlagi gramatike treba na začetku (do izpolnitve določenega pogoja) uporabljati zgolj produkcije prvega prednostnega razreda, nato zgolj produkcije drugega prednostnega razreda itd. Metoda Ehriga in sod. predpostavlja, da vhodni metamodel vsebuje le multiplikativnosti $0..1$, $1..1$, $0..*$ in $1..*$, razširitev, ki so jo predlagali Taentzer in sod. [91], pa deluje za poljubne multiplikativnosti.

Hoffmann in Minas [43] sta predstavila metodo za pretvorbo metamodela v enakovredno grafno gramatiko iz formalizma ASG (angl. *Adaptive Star Grammars*) [25]. Za razliko od kontekstno odvisnih grafnih gramatik tovrstne gramatike omogočajo obravnavo poljubnega števila vozlišč in pripadajočih povezav kot enega samega »večkratnega vozlišča«, zato lahko z eno samo produkcijo gramatike ASG predstavimo poljubno mnogo običajnih (kontekstno odvisnih) grafnih produkcij.

V nasprotju s ciljnimi formalizmi gramatik, ki jih uporabljajo doslej predlagani pristopi za pretvorbo metamodela v grafno gramatiko, naš pristop ne uporablja nobenih zahtevnejših ali *ad hoc* elementov grafnih gramatik, kot so pogoji uporabe, prednostni razredi, večkratna vozlišča ipd. Metamodel s poljubnimi multiplikativnostmi in dedovanjem bomo pretvorili v kontekstno odvisno grafno gramatiko (podpoglavje 2.4) brez dodatnih elementov.

Ehrig in sod. so metamodel v grafno gramatiko pretvorili s ciljem samodejne tvorbe veljavnih modelov. Naša metoda omogoča tvorbo modelskih grafov na podoben način kot metoda Ehriga in sod., vendar pa se bomo osredotočili na sintaksno

analizo veljavnih modelskih grafov, saj lahko izpeljava takšnega grafa v izhodni gramatiki služi kot osnova za uporabo semantičnih pravil.

Da si zagotovimo možnost sintaksne analize modelskih grafov, bodo vse izhodne gramatike izpolnjevale pogoje (2)–(4) iz definicije sintaksne odločljivosti (definicija 2.36, stran 46). Na ta način bomo lahko izpeljave modelskih grafov v izhodni gramatiki gradili s pomočjo izboljšanega Rekers-Schürrovega sintaksnega analizatorja, ki smo ga predstavili v poglavju 3. (Kot smo videli v razdelku 3.4.1, izboljšava 1 odpravlja potrebo po povezanih desnih straneh produkcij gramatike, ki jih predpisuje pogoj (1) v definiciji 2.36.) Pokazali pa bomo, da bomo izboljšani Rekers-Schürrov sintaksni analizator dejansko potrebovali samo v primerih, ko vhodni metamodel ne bo na voljo, saj bomo sicer lahko (na osnovi podatkov v vhodnem metamodelu) sintaksno analizo izvajali kar s pomočjo učinkovitega *ad hoc* postopka.

5.1.3 Opredelitev prispevka

Osrednji prispevek tega poglavja je predlog metode za pretvorbo metamodelov v enakovredne kontekstno odvisne grafne gramatike, kot sta jih definirala Rekers in Schürr [80]. Omejen nabor elementov, ki jih ponuja Rekers-Schürrov formalizem, vodi do metode, ki se bistveno razlikuje od obstoječih metod; te namreč poleg osnovnih gramatičnih elementov uporabljajo tudi različne zmogljivejše tehnike. Pokazali bomo, da je metamodele s poljubnimi multiplikativnostmi in dedovanjem mogoče pretvoriti v gramatike, ki uporabljajo zgolj gramatične oznake in kontekstno odvisne grafne produkcije.

Drugi prispevek tega poglavja je prikaz povezave med sintaksno in semantično analizo modelskih grafov po zgledu besedilnih gramatik s prilastki (angl. *attribute grammars*).

5.2 Sorodna dela

Odnos med metamodeli in grafnimi gramatikami je proučevalo več raziskovalcev, še zlasti na področju vizualnih jezikov. Sintakso vizualnega jezika (množico vseh veljavnih vizualnih stavkov) je mogoče definirati z metamodelom ali z grafno gramatiko, torej deklarativno ali generativno. Tveit [93] je primerjala metamodelski pristop k definiciji vizualne sintakse, kot ga ponuja orodje EMF [14], in grafnogramatični pristop, kot ga ponuja orodje DiaGen [68]. Bardohl in sod. [7] so razpravljali o združitvi lastnosti obeh pristopov in so v ta namen predlagali vključitev pravil dedovanja v grafnogramatični formalizem.

Če modele in metamodele predstavimo z grafi, lahko mnoge probleme s področja (meta)modeliranja predstavimo v grafnem okolju. Trojne grafne gramatike (angl. *Triple Graph Grammars*) [54, 86] so primer grafnogramatičnega formalizma, v katerem je mogoče tovrstne probleme elegantno predstaviti. Pomemben razred metamodelskih problemov je povezan z (meta)modelskimi transformacijami [64], ki jih je pogosto mogoče formulirati v obliki množice grafnih produkcij [16, 65]. V (meta)modelirnem orodju AToM^3 [22] so modelske transformacije prav tako predstavljene z grafnimi produkcijami.

Nekateri raziskovalci so se ukvarjali s povezavo med metamodeli in besedilnimi gramatikami. Alanen in Porres [2] sta razvila metodo za pretvorbo metamodela v kontekstno neodvisno besedilno gramatiko in obratno. Zaradi omejitev tovrstnih gramatik njun postopek deluje le za precej omejen razred metamodelov. Javed in sod. [47] so razvili postopek za dvosmerno prevedbo med metamodeli in kontekstno neodvisnimi besedilnimi gramatikami, da bi se lahko problema indukcije metamodela na podlagi množice modelov lotili s tehnikami za indukcijo tovrstnih gramatik [46].

5.3 Formalna opredelitev problema

V tem podpoglavju bomo problem pretvorbe metamodela v enakovredno grafno gramatiko formalno opredelili. V razdelku 5.3.1 bomo predstavili nekaj dogovorov glede notacije, ki se jih bomo držali v nadaljnjem besedilu. V razdelkih 5.3.2 in 5.3.3 bomo definirali vhodni metamodel in izhodno gramatiko. V razdelku 5.3.4 bomo definirali še problem pretvorbe.

5.3.1 Dogovori glede notacije

V tem poglavju bomo uporabljali notacijo, ki smo jo vzpostavili v poglavju 2. Spomnimo se na zapis celoštevilskega intervala $\{a,\ a+1,\ \ldots,\ b\}$ v obliki $a\,..\,b$ in neskončne celoštevilske množice $\{a,\ a+1,\ \ldots\}$ v obliki $a\,..\,*$. Ta notacija je skladna s pomenom multiplikativnosti v metamodelih.

Pri zapisovanju vsebine vreč (množic, v katerih se elementi lahko ponavljajo) bomo za zapis n kopij elementa A uporabljali okrajšavo $n\mathsf{A}$. Z uporabo tega dogovora lahko vsebino vreče $\{\mathsf{B},\ \mathsf{B},\ \mathsf{B},\ \mathsf{C},\ \mathsf{C},\ \mathsf{C},\ \mathsf{C},\ \mathsf{D}\}$ krajše zapišemo kot $\{3\mathsf{B},\ 4\mathsf{C},\ \mathsf{D}\}$. Spomnimo se, da oznaka $|\mathcal{A}|$ pri podani vreči $\mathcal{A}$ predstavlja skupno število pojavitev elementov v vreči. Na primer, $|\{3\mathsf{B},\ 4\mathsf{C},\ \mathsf{D}\}| = 8$.

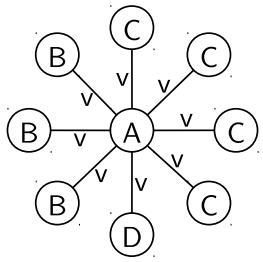
Pri zapisovanju oznak grafnih elementov ali razredov v metamodelih bomo pokončno neserifno pisavo uporabljali za dobesedne oznake, *poševno serifno pisavo* pa za spremenljivke, katerih vrednosti so oznake. Na primer, zapis A predstavlja dobesedno oznako A, zapis A pa predstavlja oznako, določeno z vrednostjo spremenljivke A.

V tem poglavju bomo pogosto uporabljali posebne oblike grafov, ki se imenujejo *zvezde*:

Definicija 5.1 (zvezda)**.** *Zvezda* je graf, sestavljen iz vozlišč v, w_1, ..., w_n (kjer je $n \geq 0$) in neusmerjenih povezav $v{-}w_1$, $v{-}w_2$, ..., $v{-}w_n$. Vozlišče v je *središče* zvezde, ostala vozlišča pa so *listi*.

Nekoliko nenavadno poimenovanje »list« izvira iz dejstva, da so zvezde posebni primeri dreves oziroma acikličnih povezanih grafov. Zvezde, v katerih so vse povezave enako označene, bomo v besedilu zapisovali na sledeči način: zapis $C \overset{a}{-} \mathcal{L}$ predstavlja zvezdo, v kateri so oznake vseh povezav enake a, oznaka središča je enaka C, oznake listov pa so podane v vreči $\mathcal{L}$. Na primer, zvezdo na sliki 5.3 lahko predstavimo kot $\mathsf{A} \overset{\mathsf{v}}{-} \{\mathsf{B},\ \mathsf{B},\ \mathsf{B},\ \mathsf{C},\ \mathsf{C},\ \mathsf{C},\ \mathsf{C},\ \mathsf{D}\}$ ali kot $\mathsf{A} \overset{\mathsf{v}}{-} \{3\mathsf{B},\ 4\mathsf{C},\ \mathsf{D}\}$. Zvezdo z neoznačenimi povezavami bomo predstavili kot $C{-}\mathcal{L}$. Zvezdo z enim samim listom

bomo pogosto zapisali kot $C \overset{a}{—} A$ namesto kot $C \overset{a}{—} \{A\}$. Če sta G_1 in G_2 zvezdi s praznim presekom, bomo z zapisom G_1; G_2 predstavili unijo obeh zvezd. Na primer, zapis $\mathsf{A}—\mathsf{B}$; $\mathsf{C} \overset{\mathsf{a}}{—} \{\mathsf{D}, \mathsf{E}\}$ predstavlja graf, sestavljen iz vozlišč A, B, C, D in E ter povezav $\mathsf{A}—\mathsf{B}$, $\mathsf{C} \overset{\mathsf{a}}{—} \mathsf{D}$ in $\mathsf{C} \overset{\mathsf{a}}{—} \mathsf{E}$.



Slika 5.3: *Primer zvezde.*

5.3.2 Vhod: metamodel

Razred metamodelov, ki jih na vhodu sprejema naš pretvorni algoritem, lahko formalno opredelimo na sledeči način:

Definicija 5.2 (metamodel, elementi metamodela)**.** Metamodel MM je peterica ($\mathcal{C}$, $\mathcal{C}_\mathrm{A}$, **Assoc**, *mult*, **Inh**), pri čemer:

- $\mathcal{C}$ označuje množico oznak *razredov* (predpostavili bomo, da imajo vsi razredi medsebojno različne oznake);
- $\mathcal{C}_\mathrm{A} \subseteq \mathcal{C}$ označuje množico oznak *abstraktnih razredov*;
- **Assoc** $\subseteq \mathcal{C} \times \mathcal{C}$ označuje relacijo *asociacije*;
- $mult \colon \mathcal{C} \times \mathcal{C} \to \mathbb{N}_0 \times (\mathbb{N}_0 \cup \{*\})$ označuje (delno) funkcijo *multiplikativnosti*;
- **Inh** $\subseteq \mathcal{C} \times \mathcal{C}$ označuje aciklično relacijo *dedovanja*.

Definirajmo še nekaj pojmov, tesno povezanih s pravkar opredeljenimi:

Definicija 5.3 (asociacija)**.** *Asociacija* je dvojica razredov P in Q, tako da velja P **Assoc** Q (razred P je v relaciji asociacije z razredom Q).

Definicija 5.4 (multiplikativnost)**.** *Multiplikativnost* je vrednost funkcije multiplikativnosti za neko asociacijo v neki smeri. Vrednost funkcije multiplikativnosti je definirana natanko za tiste pare razredov P in Q, za katere velja P **Assoc** Q. V takih primerih je vrednost funkcije urejena dvojica (a, b) ali $(a, *)$, kjer sta a in b celi števili z lastnostjo $0 \leq a \leq b$.

Ker multiplikativnosti predstavljajo celoštevilske intervale ali neskončne celoštevilske množice, bomo uporabljali zapis $a \mathinner{..} b$ oziroma $a \mathinner{..} *$ namesto (a, b) oziroma $(a, *)$.

Definicija 5.5 (podrazred, nadrazred)**.** Razred P je (neposredni) *podrazred* razreda Q natanko tedaj, ko velja $P\,\mathbf{Inh}\,Q$. Razred P je (neposredni) *nadrazred* razreda R natanko tedaj, ko velja $R\,\mathbf{Inh}\,P$.

Predpostavka 5.6. Predpostavili bomo, da v vhodnem metamodelu veljajo sledeče omejitve:

(1) Relacija asociacije je simetrična: iz $P\,\mathbf{Assoc}\,Q$ sledi $Q\,\mathbf{Assoc}\,P$ za vse P, $Q \in \mathcal{C}$.

(2) Asociacije niso označene.

(3) Vsaka dvojica razredov je povezana s po največ eno asociacijo.

Navedene omejitve niso ključnega pomena za našo pretvorno metodo. Kot bomo videli v podpoglavju 5.8, je gornje omejitve možno odstraniti brez posebno zahtevnih prilagoditev metode. Vendar pa nam opisane omejitve omogočajo uvedbo določenih poenostavitev, kot je na primer zapis asociacij z neurejenimi dvojicami. Asociacijo med razredoma P in Q bomo zapisali kot $P{-}Q$. Namesto »razreda, povezana z asociacijo« pa bomo pisali enostavno »povezana razreda«.

V razrednem diagramu UML, ki predstavlja metamodel, so razredi prikazani kot vozlišča, asociacije pa kot povezave. Abstraktni razredi so prikazani z oznakami v poševni pisavi. Multiplikativnost za dvojico razredov P in Q v smeri $P \to Q$ (torej vrednost funkcije $mult(P, Q)$) je prikazana kot oznaka, ki je na povezavi $P{-}Q$ zapisana bližje vozlišču Q. Relacija dedovanja je prikazana s praznimi trikotniki, usmerjenimi proti nadrazredom. Na primer, metamodel MM_{pub} (slika 5.1) bi formalno definirali kot peterico ($\mathcal{C}$, $\mathcal{C}_{\text{A}}$, **Assoc**, *mult*, **Inh**), pri čemer velja:

- $\mathcal{C} = \{\mathsf{R}, \mathsf{K}, \mathsf{P}, \mathsf{A}, \mathsf{S}, \mathsf{J}, \mathsf{CP}, \mathsf{CS}, \mathsf{JS}, \mathsf{C}\}$.
- $\mathcal{C}_{\text{A}} = \{\mathsf{P}, \mathsf{S}\}$.
- $\mathbf{Assoc} = \{\mathsf{P}{-}\mathsf{R}, \mathsf{S}{-}\mathsf{R}, \mathsf{S}{-}\mathsf{A}, \mathsf{S}{-}\mathsf{K}, \mathsf{J}{-}\mathsf{JS}, \mathsf{C}{-}\mathsf{CP}, \mathsf{C}{-}\mathsf{CS}\}$.
- $mult(\mathsf{S}, \mathsf{R}) = 2\,..\,3$, $mult(\mathsf{R}, \mathsf{S}) = 0\,..\,*$, vrednosti $mult(\mathsf{CS}, \mathsf{R})$ in $mult(\mathsf{R}, \mathsf{CS})$ sta nedefinirani itd.
- $\mathbf{Inh} = \{(\mathsf{J}, \mathsf{P}), (\mathsf{CP}, \mathsf{P}), (\mathsf{CS}, \mathsf{S}), (\mathsf{JS}, \mathsf{S})\}$.

Definirajmo nekaj pojmov, ki so tesno povezani z relacijama asociacije in dedovanja:

Definicija 5.7 (funkcija asociacije)**.** Za metamodel $MM = (\mathcal{C}, \mathcal{C}_{\text{A}}, \mathbf{Assoc}, mult, \mathbf{Inh})$ naj bo funkcija $Assoc\colon \mathcal{C} \to \mathcal{P}(\mathcal{C})$ definirana kot $Assoc(P) = \{Q \in \mathcal{C} \mid P\,\mathbf{Assoc}\,Q\}$.

Definicija 5.8 (ovojnici relacije **Inh** in pripadajoči funkciji)**.** Naj $\mathbf{Inh}^+$ označuje tranzitivno ovojnico relacije dedovanja, tj. $Q\,\mathbf{Inh}^+\,P \iff (Q\,\mathbf{Inh}\,P) \vee (\exists R \in \mathcal{C}\colon Q\mathbf{Inh}R \wedge R\mathbf{Inh}^+P)$. Definirajmo funkcijo $Inh^+\colon \mathcal{C} \to \mathcal{P}(\mathcal{C})$ kot $Inh^+(P) = \{Q \in \mathcal{C} \mid Q\,\mathbf{Inh}^+\,P\}$. Naj $\mathbf{Inh}^*$ označuje refleksivno-tranzitivno ovojnico relacije dedovanja, tj. $\mathbf{Inh}^* = \mathcal{I} \cup \mathbf{Inh}^+$, kjer $\mathcal{I}$ označuje relacijo identitete. Funkcija $Inh^*\colon \mathcal{C} \to \mathcal{P}(\mathcal{C})$ naj bo definirana kot $Inh^*(P) = \{P\} \cup Inh^+(P)$.

Predpostavka 5.9. Relacija dedovanja in njeni ovojnici morajo poleg acikličnosti izpolnjevati še naslednji zahtevi:

(1) Enkratno dedovanje:

$$Q \,\mathbf{Inh}\, P' \wedge Q \,\mathbf{Inh}\, P'' \implies P' = P''. \tag{5.1}$$

(2) Asociacij ni mogoče redefinirati:

$$\begin{aligned} &(P' \in \mathit{Inh}^+(P) \wedge Q' \in \mathit{Inh}^*(Q) \vee P' \in \mathit{Inh}^*(P) \wedge Q' \in \mathit{Inh}^+(Q)) \\ &\quad \wedge P \,\mathbf{Assoc}\, Q \\ &\implies \neg(P' \,\mathbf{Assoc}\, Q') \end{aligned} \tag{5.2}$$

Če sta razreda P in Q povezana, potem zahteva (2) preprečuje asociacije med razredom P in neposrednimi ali posrednimi podrazredi razreda Q, med razredom Q in neposrednimi ali posrednimi podrazredi razreda P ter med neposrednimi ali posrednimi podrazredi razredov P in Q. Metamodel MM_{pub}, denimo, ne bi mogel vsebovati asociacije med razredoma A in CS, saj je razred A že povezan z razredom S, ki je nadrazred razreda CS. V podpoglavju 5.8 bomo pokazali, da je opisano zahtevo možno omiliti.

Definicija 5.10 (skladnost z metamodelom)**.** Modelski graf M je *skladen* z metamodelom $MM = (\mathcal{C}, \mathcal{C}_{\text{A}}, \mathbf{Assoc}, \mathit{mult}, \mathbf{Inh})$ natanko v primeru, če so izpolnjeni sledeči pogoji:

(1) Vsako vozlišče modelskega grafa M predstavlja objekt nekega neabstraktnega razreda metamodela. Vozlišče z oznako P predstavlja objekt razreda P. Povezave v grafu M so neoznačene.

(2) Za vsako dvojico razredov $P, Q \in \mathcal{C}$ z lastnostjo $\mathit{mult}(P, Q) = a\,..\,b$ velja, da je vsako vozlišče grafa M, ki predstavlja objekt nekega razreda iz množice $\mathit{Inh}^*(P)$, povezano z najmanj a in največ b vozlišči, ki predstavljajo objekte razredov iz množice $\mathit{Inh}^*(Q)$.

(3) Za vsako dvojico razredov $P, Q \in \mathcal{C}$ z lastnostjo $\mathit{mult}(P, Q) = a\,..\,*$ velja, da je vsako vozlišče grafa M, ki predstavlja objekt nekega razreda iz množice $\mathit{Inh}^*(P)$, povezano z najmanj a vozlišči, ki predstavljajo objekte razredov iz množice $\mathit{Inh}^*(Q)$.

Na primer, v poljubnem modelskem grafu, skladnem z metamodelom MM_{pub}, mora biti vsako vozlišče z oznako R povezano s po najmanj enim vozliščem z oznako P, J ali CP (ker velja $\mathit{mult}(\mathsf{R}, \mathsf{P}) = 1\,..\,*$ in $\mathit{Inh}^*(\mathsf{P}) = \{\mathsf{P}, \mathsf{J}, \mathsf{CP}\}$) in s poljubnim številom (vštevši 0) vozlišč z oznakami S, CS ali JS (ker velja $\mathit{mult}(\mathsf{R}, \mathsf{S}) = 0\,..\,*$ in $\mathit{Inh}^*(\mathsf{S}) = \{\mathsf{S}, \mathsf{CS}, \mathsf{JS}\}$). Ker pa sta razreda P in S abstraktna, vozlišč s takšnima oznakama ne more vsebovati noben veljaven modelski graf.

Definicija 5.11 (jezik metamodela)**.** *Jezik* metamodela je množica vseh modelskih grafov, ki so skladni z njim. Jezik metamodela MM bomo označili kot $L(MM)$.

Ker so po predpostavki oznake razredov med seboj različne, lahko množico razredov enolično uredimo. Urejenost je lahko določena kar z abecednim vrstnim redom oznak, v splošnem pa jo lahko podaja poljubna urejevalna funkcija:

Definicija 5.12 (urejenost razredov)**.** Naj $order\colon \mathcal{C} \to 1\,..\,|\mathcal{C}|$ označuje funkcijo, ki vsakemu razredu $P \in \mathcal{C}$ priredi enolično celo število z intervala $1\,..\,|\mathcal{C}|$.

5.3.3 Izhod: grafna gramatika

Pretvorni algoritem, ki ga bomo predstavili v nadaljevanju, bo kot svoj izhod izdelal kontekstno odvisno grafno gramatiko v (prilagojenem) Rekers-Schürrovem formalizmu, ki smo ga opisali v podpoglavju 2.4. Zahtevali bomo, da izhodna gramatika izpolnjuje pogoje (2)–(4) iz definicije 2.36 (stran 46), ki zagotavljajo pravilno delovanje in ustavljivost izboljšanega Rekers-Schürrovega sintaksnega analizatorja. Na ta način bo mogoče ugotavljati veljavnost modelskih grafov in graditi izpeljave veljavnih modelskih grafov tudi v primeru, če vhodni metamodel po pretvorbi v enakovredno grafno gramatiko ne bo več na voljo.

Zaradi prihranka prostora bomo produkcije zapisovali kar v besedilu. Pri vsaki tako prikazani produkciji bo kontekstni graf (*Common*) vseboval natanko tista vozlišča, ki imajo na levi in desni strani enake oznake. Kontekstni grafi v tem poglavju ne bodo vsebovali nobenih povezav. Na primer, pri produkciji $\mathsf{A} \overset{\mathsf{v}}{—} \mathsf{B}^1$; $\mathsf{B} \overset{\mathsf{v}}{—} \mathsf{A}^1 ::= \mathsf{A}—\mathsf{B}$ je leva stran graf z vozlišči z oznakami A, B, A^1 in B^1 in s povezavama $\mathsf{A} \overset{\mathsf{v}}{—} \mathsf{B}^1$ in $\mathsf{B} \overset{\mathsf{v}}{—} \mathsf{A}^1$, desna stran je graf z vozliščema z oznakama A in B in z neoznačeno povezavo med njima, kontekstni graf pa je sestavljen iz vozlišč z oznakama A in B, ki nastopata na obeh straneh produkcije. Množico produkcij z enakimi levimi stranmi ($L ::= R_1$, $L ::= R_2$, …, $L ::= R_k$) bomo krajše zapisali kot $L ::= R_1 \mid R_2 \mid \ldots \mid R_k$.

5.3.4 Problem: pretvorba metamodela v grafno gramatiko

Problem, s katerim se ukvarjamo v tem poglavju, bi lahko formalno opredelili takole:

> Za podani metamodel MM, ki ustreza pogojem iz razdelka 5.3.2, izdelaj kontekstno odvisno grafno gramatiko $GG(MM)$, za katero velja sledeče:
>
> - Jezik gramatike $GG(MM)$ vsebuje natanko tiste modelske grafe, ki so skladni z metamodelom MM:
>
> $$L(GG(MM)) = L(MM) \tag{5.3}$$
>
> - Gramatika $GG(MM)$ izpolnjuje pogoje (2)–(4) iz definicije 2.36.

5.4 Pretvorba metamodelov s poljubnimi multiplikativnostmi

Pričenjamo z opisom metode za pretvorbo metamodela v enakovredno grafno gramatiko. Zavoljo lažje razlage se bomo v tem podpoglavju omejili na metamodele, ki

ne vsebujejo dedovanja, lahko pa vsebujejo poljubne multiplikativnosti. Dedovanje bomo obravnavali v podpoglavju 5.5.

V razdelku 5.4.1 bomo pristop na kratko pregledali in uvedli oznake, ki jih bomo uporabljali pri opisu pretvornega postopka v razdelku 5.4.2. V razdelku 5.4.3 bomo postopek ponazorili s konkretnim primerom pretvorbe, v razdelku 5.4.4 pa bomo dokazali, da postopek za poljuben vhodni metamodel, ki izpolnjuje pogoje iz razdelka 5.3.2, zgradi sintaksno odločljivo kontekstno odvisno grafno gramatiko, katere jezik je enak jeziku vhodnega metamodela.

5.4.1 Pregled pristopa

Predpostavimo, da je vhodni metamodel MM definiran kot $MM = (\mathcal{C}, \emptyset, \textbf{Assoc}, mult, \emptyset)$, pri čemer je $\mathcal{C} = \{C_1, \ldots, C_n\}$. Naša pretvorna metoda na podlagi metamodela MM izdela grafno gramatiko $GG(MM)$, ki je enakovredna metamodelu MM. Vsak graf $G \in L(GG(MM))$ predstavlja model v obliki objektnega diagrama UML, ki je skladen z metamodelom MM. Poleg tega za vsak model, ki je skladen z metamodelom MM, velja, da njegova predstavitev z objektnim diagramom UML pripada jeziku gramatike $GG(MM)$.

Množica končnih oznak gramatike $GG(MM)$, ki jo za dani metamodel MM zgradi pretvorna metoda, je sestavljena iz oznak $C_1, \ldots, C_n$ in #. V vsakem grafu G, izpeljivem v gramatiki $GG(MM)$, vozlišče z oznako C_i predstavlja objekt razreda C_i, neoznačena povezava (ki jo obravnavamo kot povezavo z oznako #) pa predstavlja vez med dvema objektoma.

Izhodno gramatiko $GG(MM)$ bomo zgradili tako, da bo z uporabo njenih produkcij mogoče zgraditi poljuben modelski graf, skladen z metamodelom MM. Izpeljava modelskega grafa s pomočjo produkcij izhodne gramatike bo praviloma potekala preko več nekončno označenih in zato neveljavnih vmesnih grafov. Čeprav vmesni grafi ne bodo predstavljali veljavnih modelov, bodo vsebovali podatke o tem, kako jih je mogoče pretvoriti v nek končno označen in veljaven modelski graf. Ti podatki bodo vsebovani v vozliščih z nekončnimi oznakami. Vsako tako vozlišče bo predstavljalo eno ali več povezav, ki jih bo v grafu še treba vzpostaviti, da bo zadoščeno eni od multiplikativnosti v vhodnem metamodelu MM. V izpeljavi končno označenega in veljavnega modelskega grafa se bodo vozlišča z nekončnimi oznakami postopoma pretvarjala v povezave s končnimi oznakami.

Nekončne oznake gramatike $GG(MM)$ zavzemajo obliko C_i^d, kjer je d pozitivno celo število in $i \in 1..n$, ali obliko $\overline{C_i}$, kjer je $i \in 1..n$. Namesto oznake C_i za splošen indeks i bomo raje uporabljali spremenljivko P ali Q. Tako bosta zapisa P^d in Q^d predstavljala nekončno oznako C_i^d, zapisa $\overline{P}$ in $\overline{Q}$ pa nekončno oznako $\overline{C_i}$, obakrat za splošen indeks i.

Kaj predstavljajo posamezne nekončne oznake izhodne gramatike $GG(MM)$? V nekončno označenem grafu, izpeljivem v gramatiki $GG(MM)$, vsako vozlišče z nekončno oznako Q^1, ki je povezano z vozliščem s končno oznako P, predstavlja povezavo, ki jo bo treba vzpostaviti med vozliščem P in (morebiti še ne obstoječim) vozliščem s končno oznako Q. Vozlišče z oznako Q^d (kjer je $d \geq 2$), povezano z vozliščem z oznako P, predstavlja skupino najmanj ene in največ d povezav, ki jih bo treba vzpostaviti med vozliščem P in (morebiti še ne obstoječimi) vozlišči z

oznako Q. Zvezda $P \overset{\mathsf{v}}{—} \{mQ^1, Q^d\}$ (v je nekončna oznaka povezave) potemtakem predstavlja zahtevo, da bo vozlišče z oznako P treba povezati z najmanj $m+1$ in največ $m+d$ (morebiti še ne obstoječimi) vozlišči z oznako Q, kar ustreza pomenu multiplikativnosti $mult(P, Q) = m+1\,..\,m+d$. Multiplikativnosti lahko torej predstavimo z zvezdami s končno označenimi središči in nekončno označenimi listi. Ta ugotovitev predstavlja osnovo za razumevanje pretvorne metode, ki jo bomo predstavili v naslednjem razdelku.

Kot bomo pokazali ob koncu razdelka 5.4.2, multiplikativnosti oblike $0\,..\,b$ ni mogoče obravnavati kot poseben primer multiplikativnosti $a\,..\,b$. Za delo z multiplikativnostmi $0\,..\,b$ smo uvedli nekončne oznake $\overline{C_i}$. Njihov pomen je opredeljen na sledeči način: Denimo, da je razred P povezan z razredi $Q_1, \ldots, Q_s$, pri čemer velja $order(Q_1) < \ldots < order(Q_s)$ (definicija 5.12), in da je multiplikativnost za dvojico razredov P in Q v smeri $P \to Q_i$ enaka $0\,..\,b_i$ za vsak $i \in 1\,..\,s$. Potem vozlišče z nekončno oznako $\overline{Q_j}$ pri nekem $j \in 1\,..\,s$, ki je povezano z vozliščem s končno oznako P, predstavlja zahtevo, da bomo za zadostitev posameznim multiplikativnostim $0\,..\,b_j$, $0\,..\,b_{j+1}$, $\ldots$, $0\,..\,b_s$ morali vozlišče P povezati z neko neprazno podvrečo vreče vozlišč $\{b_jQ_j, \ldots, b_sQ_s\}$ (to vrečo sestavlja b_j vozlišč z oznako Q_j, b_{j+1} vozlišč z oznako Q_{j+1} itd.)

Glede na vlogo, ki jo igrajo v postopku gradnje nekega veljavnega modelskega grafa, bi lahko produkcije gramatike $GG(MM)$ razdelili v tri skupine. Namen prve skupine produkcij je ustvariti poljubno število disjunktnih zvezd, ki predstavljajo posamezne multiplikativnosti. Druga skupina produkcij se ukvarja s posebnostmi, ki jih pri svoji obravnavi zahtevajo multiplikativnosti $a\,..\,*$, $a\,..\,b$ pri $a \geq 1$ in $0\,..\,b$. Ta skupina med drugim vsebuje produkcije za pretvorbo posameznega vozlišča z oznako Q^d v skupino najmanj enega in največ d vozlišč z oznako Q^1. Produkcije tretje skupine so namenjene za pretvorbo vozlišč z oznakami Q^1 v ustrezno število povezav s končnimi oznakami, s čimer se izdelava veljavnega modelskega grafa zaključi.

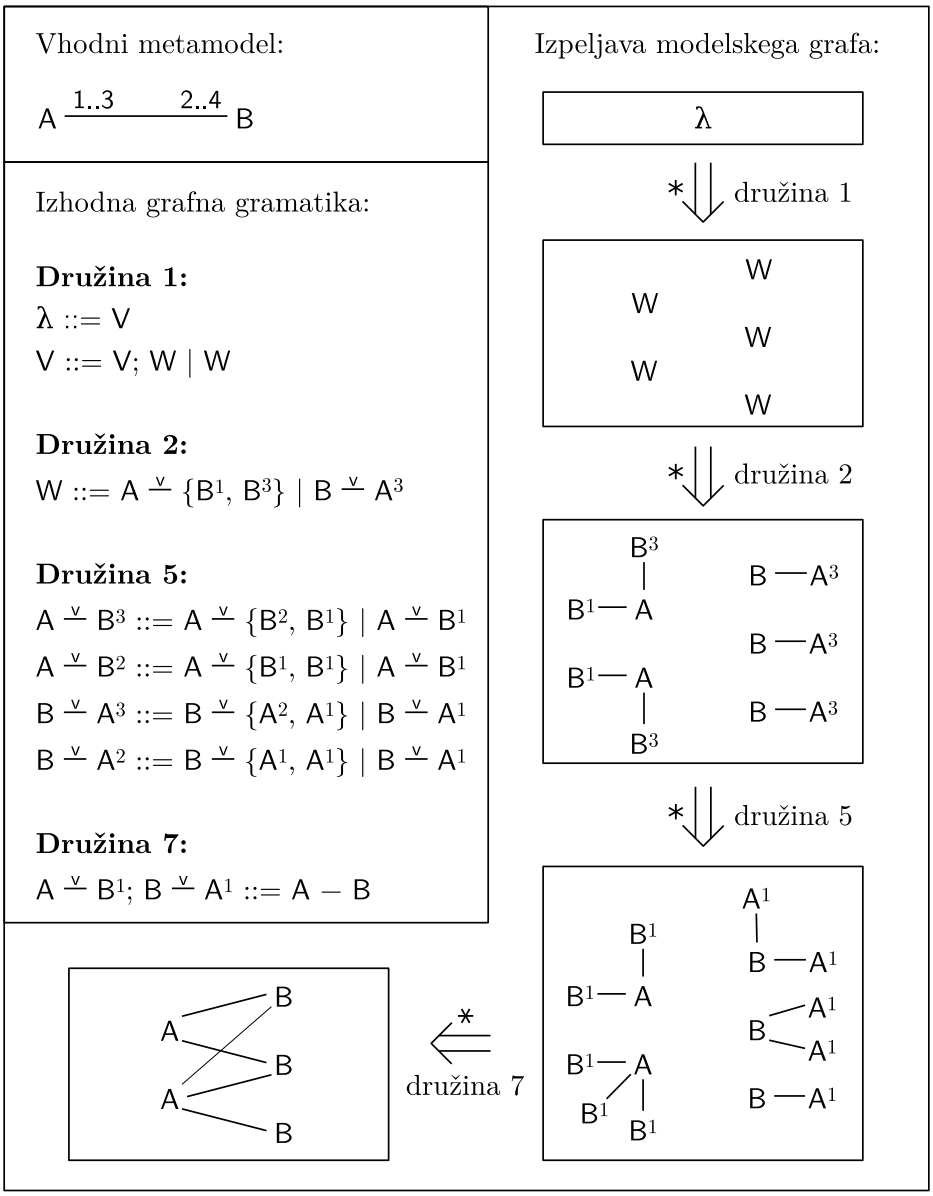
Slika 5.4 prikazuje vhodni metamodel MM, izhodno gramatiko $GG(MM)$, ki jo zgradi pretvorni postopek, in izpeljavo veljavnega modelskega grafa v gramatiki $GG(MM)$. Namen slike 5.4 je razumeti postopek izpeljave oziroma gradnje veljavnega modelskega grafa. Postopek za izdelavo gramatike $GG(MM)$ bomo opisali v naslednjem razdelku.

5.4.2 Izdelava izhodne gramatike

Postopek za izdelavo grafne gramatike $GG(MM)$ na podlagi metamodela MM je skiciran v psevdokodi na sliki 5.5. V sledečem besedilu ga bomo formalno predstavili.

Pri podanem vhodnem metamodelu $MM = (\mathcal{C}, \emptyset, \textbf{Assoc}, mult, \emptyset)$, kjer je $\mathcal{C} = \{C_1, \ldots, C_n\}$ in $\mathcal{C} \cap \{\mathsf{V}, \mathsf{W}, \mathsf{v}\} = \emptyset$, je izhodna grafna gramatika $GG(MM)$ definirana kot trojica $(\mathcal{N}, \mathcal{T}, \mathcal{P})$, pri čemer sta množici končnih in nekončnih oznak opredeljeni takole:

- $\mathcal{N} = \{\mathsf{V}, \mathsf{W}, \mathsf{v}\} \cup \mathcal{C}_i^j \cup \overline{\mathcal{C}_i}$, pri čemer zapisa $\mathcal{C}_i^j$ in $\overline{\mathcal{C}_i}$ po vrsti označujeta množici vseh oznak oblike C_i^j in vseh oznak oblike $\overline{C_i}$, ki nastopajo v produkcijah gramatike $GG(MM)$.
- $\mathcal{T} = \{C_1, \ldots, C_n, \#\}$,



Slika 5.4: *Vhodni metamodel, izhodna gramatika in izpeljava veljavnega modelskega grafa v izhodni gramatiki.*

```
1  procedura pretvori(MM = (C, ∅, Assoc, mult, ∅))
2      N := ...;        // gl. besedilo v razdelku 5.4.2
3      T := C ∪ {#};
4      P := {λ ::= V} ∪ {V ::= V;W} ∪ {V ::= W};        // družina 1
5      za vsak razred P ∈ C izvedi
6          P := P ∪ množica produkcij družine 2 za razred P;
7          za vsak razred Q ∈ C, tako da velja mult(P, Q) = a..*, izvedi
8              P := P ∪ {P ::= P —v— Q^1}        // družina 3
9          konec;
10         P := P ∪ množica produkcij družine 4 za razred P;
11         P := P ∪ množica produkcij družine 5 za razred P
12     konec;
13     za vsako asociacijo P—Q ∈ Assoc izvedi
14         P := P ∪ {P —v— Q^1; Q —v— P^1 ::= P—Q}        // družina 7
15     konec;
16     vrni GG(MM) := (N, T, P)
17 konec
```

Slika 5.5: *Shema pretvorne metode za metamodel brez dedovanja. Podrobnosti so podane v razdelku 5.4.2.*

Množica produkcij $\mathcal{P}$ sestoji iz šestih družin, ki jih bomo označili s številkami 1, 2, 3, 4, 5 in 7 (številko 6 si bomo prihranili za pozneje). Glede na vlogo, ki jo produkcije igrajo pri gradnji veljavnega modelskega grafa (predzadnji odstavek razdelka 5.4.1), družini 1 in 2 tvorita prvo skupino, družine 3, 4 in 5 drugo, družina 7 pa tretjo skupino produkcij. Posamezne družine produkcij definiramo na sledeči način:

Družina 1: Vsebuje produkcije, namenjene za tvorbo poljubnega števila nepovezanih vozlišč z nekončno oznako W:

$\lambda ::= \mathsf{V}$	(produkcija 1.1)
$\mathsf{V} ::= \mathsf{V};\ \mathsf{W}$	(produkcija 1.2)
$\mathsf{V} ::= \mathsf{W}$	(produkcija 1.3)

Produkcija $\lambda ::= \mathsf{V}$ je začetna produkcija gramatike $GG(MM)$. Glede na notacijske dogovore v podpoglavju 5.3 je graf na levi strani produkcije $\mathsf{V} ::= \mathsf{V};\ \mathsf{W}$ sestavljen zgolj iz vozlišča z oznako V, graf na desni strani je sestavljen iz dveh nepovezanih vozlišč z oznakama V in W, kontekstni graf pa vsebuje vozlišče V.

Družina 2: Produkcije iz te družine so namenjene za pretvorbo posameznih vozlišč z oznako W v zvezde, ki predstavljajo posamezna vozlišča končno označenega modelskega grafa (objekte modela) skupaj z njihovimi zahtevami po povezavah, kot jih narekujejo multiplikativnosti vhodnega metamodela MM. Za vsak razred $P \in \mathcal{C}$ vsebuje družina 2 najmanj po eno in največ po dve produkciji.

Naj za razred $P \in \mathcal{C}$ velja $\mathbf{Assoc}(P) = \{Q_1, \ldots, Q_p\}$ in

$$mult(P, Q_i) = \begin{cases} a_i \,..\, a_i \ (a_i \geq 1) & \text{pri } i \in 1 \,..\, k, \\ a_i \,..\, * \ (a_i \geq 0) & \text{pri } i \in k+1 \,..\, l, \\ a_i \,..\, b_i \ (1 \leq a_i < b_i) & \text{pri } i \in l+1 \,..\, m, \\ 0 \,..\, b_i \ (b_i \geq 1) & \text{pri } i \in m+1 \,..\, p, \end{cases} \tag{5.4}$$

kjer velja $0 \leq k \leq l \leq m \leq p$ in $order(Q_{m+1}) < \ldots < order(Q_p)$. (Razrede, povezane z danim razredom, lahko vedno razvrstimo v skladu z enačbo (5.4), saj je katerikoli od intervalov $1 \,..\, k$, $k+1 \,..\, l$, $l+1 \,..\, m$ in $m+1 \,..\, p$ lahko prazen.) Če je $m = p$, potem razredu P pripada natanko ena produkcija družine 2:

$$\mathsf{W} ::= P \overset{\mathsf{v}}{—} \mathcal{Q} \qquad \text{(družina 2.1)}$$

V nasprotnem primeru pa množico produkcij družine 2 za razred P sestavljata dve produkciji:

$$\mathsf{W} ::= P \overset{\mathsf{v}}{—} \mathcal{Q} \qquad \text{(družina 2.1)}$$
$$\mathsf{W} ::= P \overset{\mathsf{v}}{—} (\mathcal{Q} \cup \{\overline{Q_{m+1}}\}) \qquad \text{(družina 2.2)}$$

V obeh primerih je $\mathcal{Q}$ vreča, definirana kot

$$\begin{aligned} \mathcal{Q} \ = & \{a_1 \, Q_1^1, \ \ldots, \ a_l \, Q_l^1\} \cup \\ & \{(a_{l+1} - 1) \, Q_{l+1}^1, \ \ldots, \ (a_m - 1) \, Q_m^1\} \cup \\ & \{Q_{l+1}^{b_{l+1} - a_{l+1} + 1}, \ \ldots, \ Q_m^{b_m - a_m + 1}\}. \end{aligned} \tag{5.5}$$

Če je vreča $\mathcal{Q}$ prazna, produkciji družin 2.1 in 2.2 postaneta $\mathsf{W} ::= P$ oziroma $\mathsf{W} ::= P \overset{\mathsf{v}}{—} \overline{Q_{m+1}}$. Upoštevati moramo, da je razred Q_{m+1} »najmanjši« (glede na funkcijo *order*) med vsemi razredi Q, pri katerih je multiplikativnost $mult(P, Q)$ oblike $0 \,..\, b$.

Vsaka zvezda, ki jo ustvari produkcija družine 2, predstavlja zahtevo, da bo vozlišče z oznako P (središče zvezde) treba povezati z a_i vozlišči z oznako Q_i (za vsak $i \in 1 \,..\, l$) ter z najmanj a_i in kvečjemu b_i $(= (a_i - 1) + (b_i - a_i + 1))$ vozlišči z oznako Q_i (za vsak $i \in l+1 \,..\, m$), kar ustreza pomenu multiplikativnosti $a_i \,..\, a_i$, $a_i \,..\, *$ in $a_i \,..\, b_i$. Vozlišče z oznako $\overline{Q_{m+1}}$, ki ga ustvari produkcija družine 2.2, predstavlja dejstvo, da bo vozlišče P treba povezati z neko neprazno podvrečo vreče vozlišč $\{b_{m+1} Q_{m+1}, \ \ldots, \ b_p Q_p\}$.

Družina 2 ne razlikuje med multiplikativnostmi $a_i \,..\, a_i$ in $a_i \,..\, *$, saj lahko obema vrstama multiplikativnosti zadostimo z a_i povezavami. Z neobstoječimi zgornjimi mejami multiplikativnosti $a_i \,..\, *$ se bo ukvarjala družina 3.

Metamodel MM_{pub} (slika 5.1) bo služil kot tekoči primer v predstavitvi pretvornega postopka. Na primer, razredu S v tem metamodelu pripada sledeča produkcija družine 2:

$$\mathsf{W} ::= \mathsf{S} \overset{\mathsf{v}}{—} \{\mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{R}^1, \mathsf{R}^2\}$$

Družina 3: Ta družina vsebuje produkcije za obravnavo multiplikativnosti oblike $a\,..\,*$. Vsaki dvojici razredov P in Q z lastnostjo $mult(P, Q) = a\,..\,*$ pripada sledeča produkcija družine 3:

$$P ::= P \overset{\vee}{—} Q^1$$

Ta produkcija omogoča vezavo poljubnega števila dodatnih vozlišč z oznako Q^1 na vozlišče z oznako P. Ker vsako vozlišče z oznako Q^1 predstavlja bodočo povezavo z vozliščem z oznako Q, lahko z večkratno uporabo produkcije družine 3 na istem vozlišču P ustvarimo zametke poljubnega števila bodočih povezav z različnimi vozlišči Q. Produkcija družine 3 torej zajema pomen neobstoječe zgornje meje multiplikativnosti oblike $a\,..\,*$.

Razredu S v metamodelu MM_{pub} pripadata dve produkciji družine 3:

$$\mathsf{S} ::= \mathsf{S} \overset{\vee}{—} \mathsf{A}^1 \mid \mathsf{S} \overset{\vee}{—} \mathsf{K}^1$$

Družina 4: Razredu P, ki je z razredi $Q_1, \ldots, Q_p$ povezan na način, kot ga določa enačba (5.4), pripadajo sledeče produkcije družine 4:

- Za vsak $i \in m+1\,..\,p$:

$$P \overset{\vee}{—} \overline{Q_i} ::= P \overset{\vee}{—} Q_i^{b_i} \quad \text{(družina 4.1)}$$

- Za vsak $i \in m+1\,..\,p-1$:

$$P \overset{\vee}{—} \overline{Q_i} ::= P \overset{\vee}{—} \{Q_i^{b_i},\ \overline{Q_{i+1}}\} \quad \text{(družina 4.2)}$$
$$P \overset{\vee}{—} \overline{Q_i} ::= P \overset{\vee}{—} \overline{Q_{i+1}} \quad \text{(družina 4.3)}$$

Z uporabo produkcij družine 4 je vozlišče z oznako $\overline{Q_{m+1}}$, ki je povezano z vozliščem z oznako P, mogoče pretvoriti v poljubno neprazno podmnožico $\mathcal{F}'$ množice vozlišč $\mathcal{F} \equiv \{Q_{m+1}^{b_{m+1}}, \ldots, Q_p^{b_p}\}$ (skupaj s pripadajočimi povezavami z vozliščem P), in sicer po postopku, ki ga lahko obravnavamo kot zaporedje največ $(p-m)$ neodvisnih dvojiških odločitvenih korakov. Pričnemo s prazno množico $\mathcal{F}'$. V j-tem koraku se odločamo, ali bi v množico $\mathcal{F}'$ vključili vozlišče z oznako $Q_{m+j}^{b_{m+j}}$. Odločitev sprejmemo z uporabo ustrezne produkcije: produkcija družine 4.2 vključi to vozlišče v množico $\mathcal{F}'$, produkcija družine 4.3 pa ga izpusti. Obe produkciji spremenita oznako vozlišča $\overline{Q_{m+j}}$ v $\overline{Q_{m+j+1}}$ in tako omogočita nadaljevanje postopka; v naslednjem koraku se bomo odločali, ali naj v nastajajočo množico $\mathcal{F}'$ vključimo vozlišče z oznako $Q_{m+j+1}^{b_{m+j+1}}$. Uporaba produkcije družine 4.1 pa odstrani vozlišče $\overline{Q_{m+j}}$ in tako zaključi postopek gradnje množice $\mathcal{F}'$. Produkcije družine 4 potemtakem ustrezno obravnavajo skupino razredov, ki so z razredom P povezani preko multiplikativnosti oblike $0\,..\,b$.

V primeru metamodela MM_{pub} razredu JS, denimo, pripada ena sama produkcija družine 4:

$$\mathsf{JS} \overset{\vee}{—} \overline{\mathsf{J}} ::= \mathsf{JS} \overset{\vee}{—} \mathsf{J}^1$$

Ta produkcija spada v družino 4.1. Razredu JS ne pripada nobena produkcija iz družin 4.2 in 4.3.

Družina 5: Produkcije iz te družine so namenjene pretvorbi vozlišča z nekončno oznako Q^d ($d \geq 2$), povezanega z vozliščem s končno oznako P, v najmanj eno in največ d vozlišč z nekončno oznako Q^1 (skupaj s pripadajočimi povezavami z vozliščem P). Za vsak podgraf $P \overset{\vee}{-} Q^d$ ($d \geq 2$), ki nastopa na desni strani neke produkcije (vključujoč produkcije iz družine 5), vsebuje družina 5 sledeči produkciji:

$$P \overset{\vee}{-} Q^d ::= P \overset{\vee}{-} \{Q^{d-1},\ Q^1\} \qquad \text{(družina 5.1)}$$
$$P \overset{\vee}{-} Q^d ::= P \overset{\vee}{-} Q^1 \qquad \text{(družina 5.2)}$$

V edini produkciji družine 2 za razred S iz metamodela MM_{pub}, tj.

$$\mathsf{W} ::= \mathsf{S} \overset{\vee}{-} \{\mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{R}^1, \mathsf{R}^2\},$$

nastopa podgraf $\mathsf{S} \overset{\vee}{-} \mathsf{R}^2$, ki mu pripadata dve produkciji družine 5:

$$\mathsf{S} \overset{\vee}{-} \mathsf{R}^2 ::= \mathsf{S} \overset{\vee}{-} \{\mathsf{R}^1, \mathsf{R}^1\} \mid \mathsf{S} \overset{\vee}{-} \mathsf{R}^1$$

Družina 7: Družina 7 vsebuje po eno produkcijo za vsako asociacijo $P{-}Q \in$ **Assoc**:

$$P \overset{\vee}{-} Q^1;\ Q \overset{\vee}{-} P^1 ::= P{-}Q$$

Ta produkcija pretvori vzajemno potrebo po povezavi med vozliščema P in Q, ki jo predstavljata vozlišči Q^1 in P^1, v dejansko (končno označeno) povezavo. (Spomnimo se, da neoznačeno povezavo obravnavamo kot povezavo z oznako $\#$, ta oznaka pa pripada množici končnih oznak izhodne gramatike.) Z zaporednimi uporabami produkcij iz družine 7 lahko zahteve po povezavah, ki jih predstavljajo posamezna vozlišča P^1, postopoma preoblikujemo v dejanske povezave. V razdelku 5.4.4 bomo pokazali, da je modelski graf, ki nastane po pretvorbi vseh vozlišč z nekončnimi oznakami v neoznačene povezave, skladen z metamodelom MM.

V primeru metamodela MM_{pub} je asociacija med razredoma S in R vzrok za naslednjo produkcijo družine 7:

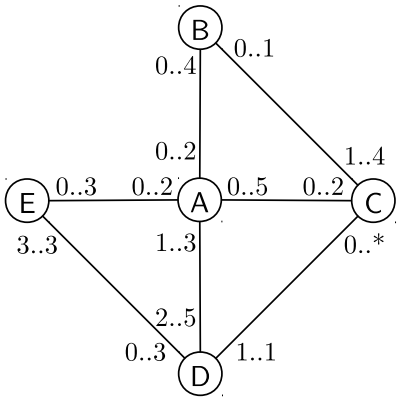
$$\mathsf{S} \overset{\vee}{-} \mathsf{R}^1;\ \mathsf{R} \overset{\vee}{-} \mathsf{S}^1 ::= \mathsf{S}{-}\mathsf{R}$$

Zakaj smo multiplikativnosti oblike $0\,..\,b$ ločili od multiplikativnosti oblike $a\,..\,b$ pri $a \geq 1$? Nenazadnje bi se lahko pretvarjali, kot da je multiplikativnost $0\,..\,b$ za dvojico razredov P in Q v smeri $P \to Q$ dejansko multiplikativnost $1\,..\,b$, možnost neobstoja povezave med vozliščema P in Q v modelskem grafu pa bi upoštevali s produkcijo $P \overset{\vee}{-} Q^1 ::= P$. Žal pa je ta zamisel neizvedljiva, saj produkcija $P \overset{\vee}{-} Q^1 ::= P$ ne izpolnjuje zahteve (3) iz definicije 2.36 (stran 46) in zato ne more biti sestavni del nobene sintaksno odločljive gramatike. Multiplikativnosti $0\,..\,b$ so torej zahtevale uvedbo dodatnih nekončnih oznak in pripadajočih produkcij.

5.4.3 Primer pretvorbe

Za metamodel MM s slike 5.6 bi pretvorni postopek zgradil izhodno gramatiko $GG(MM) = \{\mathcal{N}, \mathcal{T}, \mathcal{P}\}$ s sledečimi lastnostmi:

- $\mathcal{N} = \{\mathsf{V}, \mathsf{W}, \mathsf{v}, \mathsf{A}^1, \mathsf{A}^2, \mathsf{A}^3, \mathsf{A}^4, \mathsf{A}^5, \mathsf{B}^1, \mathsf{B}^2, \mathsf{B}^3, \mathsf{B}^4, \mathsf{C}^1, \mathsf{C}^2, \mathsf{C}^3, \mathsf{C}^4, \mathsf{D}^1, \mathsf{D}^2, \mathsf{D}^3, \mathsf{D}^4, \mathsf{E}^1, \mathsf{E}^2, \mathsf{E}^3, \overline{\mathsf{A}}, \overline{\mathsf{B}}, \overline{\mathsf{C}}, \overline{\mathsf{D}}, \overline{\mathsf{E}}\}$.
- $\mathcal{T} = \{\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{D}, \mathsf{E}, \#\}$.
- Produkcije gramatike so nanizane v tabeli 5.1. Pri gradnji množice produkcij smo predpostavili abecedno urejenost razredov metamodela ($order(\mathsf{A}) < order(\mathsf{B}) < order(\mathsf{C}) < order(\mathsf{D}) < order(\mathsf{E})$). Zaradi prihranka prostora smo pri družini 5 izpustili produkcije s sledečimi levimi stranmi: $\mathsf{A}\overset{\mathsf{v}}{-}\mathsf{D}^4$, $\mathsf{A}\overset{\mathsf{v}}{-}\mathsf{D}^3$, $\mathsf{A}\overset{\mathsf{v}}{-}\mathsf{D}^2$, $\mathsf{A}\overset{\mathsf{v}}{-}\mathsf{E}^3$, $\mathsf{A}\overset{\mathsf{v}}{-}\mathsf{E}^2$, $\mathsf{B}\overset{\mathsf{v}}{-}\mathsf{A}^2$, $\mathsf{B}\overset{\mathsf{v}}{-}\mathsf{C}^4$, $\mathsf{B}\overset{\mathsf{v}}{-}\mathsf{C}^3$, $\mathsf{B}\overset{\mathsf{v}}{-}\mathsf{C}^2$, $\mathsf{C}\overset{\mathsf{v}}{-}\mathsf{A}^5$, $\mathsf{C}\overset{\mathsf{v}}{-}\mathsf{A}^4$, $\mathsf{C}\overset{\mathsf{v}}{-}\mathsf{A}^3$, $\mathsf{C}\overset{\mathsf{v}}{-}\mathsf{A}^2$, $\mathsf{D}\overset{\mathsf{v}}{-}\mathsf{A}^3$, $\mathsf{D}\overset{\mathsf{v}}{-}\mathsf{A}^2$ in $\mathsf{E}\overset{\mathsf{v}}{-}\mathsf{A}^2$.



Slika 5.6: *Primer metamodela brez dedovanja.*

5.4.4 Dokaz sintaksne odločljivosti gramatike $GG(MM)$ in enakosti jezikov $L(GG(MM))$ in $L(MM)$

V tem razdelku bomo dokazali, da je pri poljubnem vhodnem metamodelu MM, ki izpolnjuje pogoje iz razdelka 5.3.2, izhodna gramatika $GG(MM)$ sintaksno odločljiva in da je njen jezik istoveten z jezikom metamodela MM, tj. $L(GG(MM)) = L(MM)$. Istovetnost jezikov bomo dokazali s pomočjo pravila $L_1 = L_2 \iff (L_1 \subseteq L_2) \wedge (L_2 \subseteq L_1)$. Dokaz lastnosti $L(GG(MM)) = L(MM)$ bomo torej razbili na dva dela:

- $L(GG(MM)) \subseteq L(MM)$: Vsak končno označen graf, ki je izpeljiv v gramatiki $GG(MM)$, predstavlja model, ki je skladen z metamodelom MM.
- $L(MM) \subseteq L(GG(MM))$: Vsak modelski graf, ki je skladen z metamodelom MM, je končno označen in izpeljiv v gramatiki $GG(MM)$.

Bralec, ki ga razmeroma dolgovezni dokazi ne zanimajo, lahko brez škode za razumevanje besedila preide na podpoglavje 5.5 (stran 180).

Tabela 5.1: *Produkcije izhodne grafne gramatike za metamodel s slike 5.6.*

družina	produkcije
1	$\lambda ::= \mathsf{V}$ $\mathsf{V} ::= \mathsf{V};\ \mathsf{W} \mid \mathsf{W}$
2	$\mathsf{W} ::= \mathsf{A} \overset{\vee}{-} \{\mathsf{D}^1, \mathsf{D}^4\} \mid \mathsf{A} \overset{\vee}{-} \{\mathsf{D}^1, \mathsf{D}^4, \overline{\mathsf{B}}\} \mid$ $\mathsf{B} \overset{\vee}{-} \mathsf{C}^4 \mid \mathsf{B} \overset{\vee}{-} \{\mathsf{C}^4, \overline{\mathsf{A}}\} \mid$ $\mathsf{C} \overset{\vee}{-} \mathsf{D}^1 \mid \mathsf{C} \overset{\vee}{-} \{\mathsf{D}^1, \overline{\mathsf{A}}\} \mid$ $\mathsf{D} \overset{\vee}{-} \{\mathsf{A}^3, \mathsf{E}^1, \mathsf{E}^1, \mathsf{E}^1\} \mid$ $\mathsf{E} \mid \mathsf{E} \overset{\vee}{-} \overline{\mathsf{A}}$
3	$\mathsf{D} ::= \mathsf{D} \overset{\vee}{-} \mathsf{C}^1$
4	$\mathsf{A} \overset{\vee}{-} \overline{\mathsf{B}} ::= \mathsf{A} \overset{\vee}{-} \mathsf{B}^4 \mid \mathsf{A} \overset{\vee}{-} \{\mathsf{B}^4, \overline{\mathsf{C}}\} \mid \mathsf{A} \overset{\vee}{-} \overline{\mathsf{C}}$ $\mathsf{A} \overset{\vee}{-} \overline{\mathsf{C}} ::= \mathsf{A} \overset{\vee}{-} \mathsf{C}^2 \mid \mathsf{A} \overset{\vee}{-} \{\mathsf{C}^2, \overline{\mathsf{E}}\} \mid \mathsf{A} \overset{\vee}{-} \overline{\mathsf{E}}$ $\mathsf{A} \overset{\vee}{-} \overline{\mathsf{E}} ::= \mathsf{A} \overset{\vee}{-} \mathsf{E}^3$ $\mathsf{B} \overset{\vee}{-} \overline{\mathsf{A}} ::= \mathsf{B} \overset{\vee}{-} \mathsf{A}^2$ $\mathsf{C} \overset{\vee}{-} \overline{\mathsf{A}} ::= \mathsf{C} \overset{\vee}{-} \mathsf{A}^5 \mid \mathsf{C} \overset{\vee}{-} \{\mathsf{A}^5, \overline{\mathsf{B}}\} \mid \mathsf{C} \overset{\vee}{-} \overline{\mathsf{B}}$ $\mathsf{C} \overset{\vee}{-} \overline{\mathsf{B}} ::= \mathsf{C} \overset{\vee}{-} \mathsf{B}^1$ $\mathsf{E} \overset{\vee}{-} \overline{\mathsf{A}} ::= \mathsf{E} \overset{\vee}{-} \mathsf{A}^2 \mid \mathsf{E} \overset{\vee}{-} \{\mathsf{A}^2, \overline{\mathsf{D}}\} \mid \mathsf{E} \overset{\vee}{-} \overline{\mathsf{D}}$ $\mathsf{E} \overset{\vee}{-} \overline{\mathsf{D}} ::= \mathsf{E} \overset{\vee}{-} \mathsf{D}^3$
5	$\mathsf{A} \overset{\vee}{-} \mathsf{B}^4 ::= \mathsf{A} \overset{\vee}{-} \{\mathsf{B}^3, \mathsf{B}^1\} \mid \mathsf{A} \overset{\vee}{-} \mathsf{B}^1$ $\mathsf{A} \overset{\vee}{-} \mathsf{B}^3 ::= \mathsf{A} \overset{\vee}{-} \{\mathsf{B}^2, \mathsf{B}^1\} \mid \mathsf{A} \overset{\vee}{-} \mathsf{B}^1$ $\mathsf{A} \overset{\vee}{-} \mathsf{B}^2 ::= \mathsf{A} \overset{\vee}{-} \{\mathsf{B}^1, \mathsf{B}^1\} \mid \mathsf{A} \overset{\vee}{-} \mathsf{B}^1$ $\mathsf{A} \overset{\vee}{-} \mathsf{C}^2 ::= \mathsf{A} \overset{\vee}{-} \{\mathsf{C}^1, \mathsf{C}^1\} \mid \mathsf{A} \overset{\vee}{-} \mathsf{C}^1$ $\ldots$ $\mathsf{E} \overset{\vee}{-} \mathsf{D}^3 ::= \mathsf{E} \overset{\vee}{-} \{\mathsf{D}^2, \mathsf{D}^1\} \mid \mathsf{E} \overset{\vee}{-} \mathsf{D}^1$ $\mathsf{E} \overset{\vee}{-} \mathsf{D}^2 ::= \mathsf{E} \overset{\vee}{-} \{\mathsf{D}^1, \mathsf{D}^1\} \mid \mathsf{E} \overset{\vee}{-} \mathsf{D}^1$
7	$\mathsf{A} \overset{\vee}{-} \mathsf{B}^1;\ \mathsf{B} \overset{\vee}{-} \mathsf{A}^1 ::= \mathsf{A}{-}\mathsf{B}$ $\mathsf{A} \overset{\vee}{-} \mathsf{C}^1;\ \mathsf{C} \overset{\vee}{-} \mathsf{A}^1 ::= \mathsf{A}{-}\mathsf{C}$ $\mathsf{A} \overset{\vee}{-} \mathsf{D}^1;\ \mathsf{D} \overset{\vee}{-} \mathsf{A}^1 ::= \mathsf{A}{-}\mathsf{D}$ $\mathsf{A} \overset{\vee}{-} \mathsf{E}^1;\ \mathsf{E} \overset{\vee}{-} \mathsf{A}^1 ::= \mathsf{A}{-}\mathsf{E}$ $\mathsf{B} \overset{\vee}{-} \mathsf{C}^1;\ \mathsf{C} \overset{\vee}{-} \mathsf{B}^1 ::= \mathsf{B}{-}\mathsf{C}$ $\mathsf{C} \overset{\vee}{-} \mathsf{D}^1;\ \mathsf{D} \overset{\vee}{-} \mathsf{C}^1 ::= \mathsf{C}{-}\mathsf{D}$ $\mathsf{D} \overset{\vee}{-} \mathsf{E}^1;\ \mathsf{E} \overset{\vee}{-} \mathsf{D}^1 ::= \mathsf{D}{-}\mathsf{E}$

5.4.4.1 Sintaksna odločljivost gramatike $GG(MM)$

Trditev 5.13. Gramatika $GG(MM)$ je sintaksno odločljiva.

Dokaz. Sintaksno odločljivost najlažje pokažemo tako, da za vsako produkcijo gramatike $GG(MM)$ preverimo zahteve (2), (3) in (4) iz definicije 2.36 (stran 46). Vse nezačetne produkcije imajo neprazne leve strani, torej je zahteva (2) izpolnjena. Gramatika ustreza tudi zahtevi (3), saj je pri vseh produkcijah množica *Xrhs* neprazna. Na primer, pri vsaki produkciji iz družine 7 množica *Xrhs* vsebuje neoznačeno povezavo. Če želimo preveriti skladnost z zahtevo (4) (plastni pogoj), moramo poiskati ne-negativno celo število M in ustrezno plastno funkcijo. Če število M postavimo na vrednost $|\mathcal{C}| + 3$, plastno funkcijo pa definiramo tako, kot prikazuje tabela 5.2, potem vsaka produkcija $p \in \mathcal{P}[GG(MM)]$ izpolnjuje plastni pogoj $lvec(Lhs[p]) <_{\text{lex}} lvec(Rhs[p])$:

- Družine 1, 3, 5 in 7: Tabela 5.3 prikazuje vektorje $lvec(Lhs[p])$ in $lvec(Rhs[p])$ za te družine produkcij. Zlahka preverimo, da velja $lvec(Lhs[p]) <_{\text{lex}} lvec(Rhs[p])$ za vse štiri družine.

- Družina 2: Za to družino produkcij velja $lvec(Lhs[p]) = (0, 1, 0, \ldots, 0)$, vektor $lvec(Rhs[p])$ pa je leksikografsko večji ali enak kot vektor $(1, 0, 0, \ldots, 0)$, ker po definiciji plastne funkcije za vsak razred P velja $layer(P) = 0$. Zato v vsakem primeru velja $lvec(Lhs[p]) <_{\text{lex}} lvec(Rhs[p])$.

- Družina 4.1: V tem primeru imamo $lvec(Lhs[p]) = (1, 1, 0, \ldots, 0, 1, 0, \ldots, 0)$ in $lvec(Rhs[p]) = (1, 1, 1, 0, \ldots, 0)$ (oziroma $(1, 2, 0, \ldots, 0)$, če velja $b_i = 1$). V obeh vektorjih sta prvi dve števili 1 posledici vozlišča P in povezave v, ki sta prisotni na obeh straneh produkcij družine 4.1. Tretje število 1 je posledica vozlišča $\overline{Q_i}$ v $Lhs[p]$ oziroma vozlišča $Q_i^{b_i}$ v $Rhs[p]$. Ker za vse $i \in 1\,..\,|\mathcal{C}|$ velja $layer(\overline{Q_i}) \geq 3$, je vektor $lvec(Lhs[p])$ vedno leksikografsko manjši ali enak kot vektor $(1, 1, 0, 1, 0, \ldots, 0)$, ta pa je leksikografsko manjši od vektorja $lvec(Rhs[p])$.

- Družina 4.2: Ta primer je podoben družini 4.1.

- Družina 4.3: V tem primeru vektorja $lvec(Lhs[p])$ in $lvec(Rhs[p])$ oba zavzemata obliko $(1, 1, 0, \ldots, 0, 1, 0, \ldots, 0)$. Tretje število 1 je posledica vozlišča $\overline{Q_i}$ v $Lhs[p]$ in vozlišča $\overline{Q_{i+1}}$ v $Rhs[p]$. Vendar pa se tretje število 1 v vektorju $lvec(Lhs[p])$ nahaja bolj desno kot v vektorju $lvec(Rhs[p])$, ker iz predpostavke $order(Q_i) < order(Q_{i+1})$ sledi $layer(Q_i) > layer(Q_{i+1})$. Zaradi tega velja $lvec(Lhs[p]) <_{\text{lex}} lvec(Rhs[p])$.

Ker smo našli število M in plastno funkcijo, tako da je plastni pogoj izpolnjen za vse produkcije, lahko zaključimo, da je gramatika sintaksno odločljiva. □

Tabela 5.2: *Plastna funkcija, ki dokazuje sintaksno odločljivost grafne gramatike $GG(MM)$ pri poljubnem metamodelu $MM = (\mathcal{C}, \emptyset,$ **Assoc**$, mult, \emptyset)$.*

oznake	plast
#	0
W, v	1
V	2
C_i pri vseh $i \in 1\,..\,\lvert\mathcal{C}\rvert$	0
C_i^1 pri vseh $i \in 1\,..\,\lvert\mathcal{C}\rvert$	1
C_i^d pri vseh $i \in 1\,..\,\lvert\mathcal{C}\rvert$, $d \geq 2$	2
$\overline{C_i}$ pri vseh $i \in 1\,..\,\lvert\mathcal{C}\rvert$	$\lvert\mathcal{C}\rvert + 3 - order(C_i)$

Tabela 5.3: *Vektorji $lvec(Lhs[p])$ in $lvec(Rhs[p])$ za produkcije družin 1, 3, 5 in 7. Pripadajoča plastna funkcija je prikazana v tabeli 5.2.*

družina	$lvec(Lhs[p])$	$lvec(Rhs[p])$
1.1	$(0,0,0,0,\ldots,0)$	$(0,0,1,0,\ldots,0)$
1.2	$(0,0,1,0,\ldots,0)$	$(0,1,1,0,\ldots,0)$
1.3	$(0,0,1,0,\ldots,0)$	$(0,1,0,0,\ldots,0)$
3	$(1,0,0,0,\ldots,0)$	$(1,2,0,0,\ldots,0)$
5.1	$(1,1,1,0,\ldots,0)$	$(1,2,1,0,\ldots,0)^a$
5.2	$(1,1,1,0,\ldots,0)$	$(1,2,0,0,\ldots,0)$
7	$(2,4,0,0,\ldots,0)$	$(3,0,0,0,\ldots,0)$

[a] $(1,3,0,0,\ldots,0)$ za primer $d = 2$

5.4.4.2 Dokaz lastnosti $L(GG(MM)) \subseteq L(MM)$

Dokaz lastnosti, da je vsak končno označen graf, ki je izpeljiv v gramatiki $GG(MM)$, skladen z metamodelom MM, bomo pričeli z dvema lemama. V lemi 5.14 bomo pokazali, da vsak končno ali nekončno označen graf, ki je izpeljiv v gramatiki $GG(MM)$, poseduje določene lastnosti. Lema 5.15 bo zgolj poenostavljena in oslabljena različica leme 5.14. Lastnost $L(GG(MM)) \subseteq L(MM)$ bomo v okviru trditve 5.16 dokazali z omejitvijo leme 5.15 na *končno označene* grafe, izpeljive v gramatiki $GG(MM)$, torej na grafe, ki pripadajo jeziku $L(GG(MM))$.

Lema 5.14. Naj bo nek razred P iz metamodela MM povezan z razredi $Q_1, \ldots, Q_p$ na način, kot ga določa enačba (5.4) na strani 169. Naj bo G nek graf, izpeljiv v gramatiki $GG(MM)$. Potem za vsako vozlišče s končno oznako P v grafu G veljajo sledeče lastnosti:

(1) Pri vsakem $i \in 1\,..\,k$ je vozlišče P povezano z u_i vozlišči s končno oznako Q_i in z v_i vozlišči z nekončno oznako Q_i^1, tako da velja $u_i \geq 0$, $v_i \geq 0$ in $u_i + v_i = a_i$.

(2) Pri vsakem $i \in k+1\,..\,l$ je vozlišče P povezano z u_i vozlišči Q_i in v_i vozlišči Q_i^1, tako da velja $u_i \geq 0$, $v_i \geq 0$ in $u_i + v_i \geq a_i$.

(3) Pri vsakem $i \in l+1\,..\,m$ je vozlišče P povezano z u_i vozlišči Q_i, v_i vozlišči Q_i^1 in največ enim vozliščem $Q_i^{w_i}$, tako da velja $u_i \geq 0$, $v_i \geq 0$ in $2 \leq w_i \leq b_i - a_i + 1$. Če je vozlišče P dejansko povezano z vozliščem $Q_i^{w_i}$, potem velja tudi $u_i + v_i + w_i = b_i$, sicer (če takšne povezave ni) pa velja $a_i \leq u_i + v_i \leq b_i$.

(4) Pri vsakem $i \in m+1\,..\,p$ je vozlišče P povezano z u_i vozlišči Q_i, v_i vozlišči Q_i^1, kvečjemu enim vozliščem $\overline{Q_i}$ in kvečjemu enim vozliščem $Q_i^{w_i}$, tako da velja $u_i \geq 0$, $v_i \geq 0$ in $2 \leq w_i \leq b_i$. Če je vozlišče P dejansko povezano z vozliščem $Q_i^{w_i}$, potem velja tudi $u_i + v_i + w_i = b_i$, sicer pa velja $0 \leq u_i + v_i \leq b_i$.

Dokaz. Da bi lastnosti iz leme dokazali za vse grafe, izpeljive v gramatiki $GG(MM)$, moramo preveriti dvoje:

1. Ali lastnosti veljajo za začetni graf gramatike? (Začetni graf gramatike je desna stran začetne produkcije, torej graf z enim samim vozliščem in brez povezav, pri čemer je oznaka vozlišča enaka V.)

2. Ali se lastnosti ohranijo po uporabi poljubne produkcije gramatike $GG(MM)$? Z drugimi besedami: če na nekem grafu, za katerega lastnosti iz leme veljajo, uporabimo poljubno produkcijo gramatike $GG(MM)$, ali bodo v nastalem grafu te lastnosti še vedno veljale?

Lastnosti iz leme za začetni graf gramatike trivialno veljajo, saj se nanašajo samo na vozlišča s končnimi oznakami. Preostane nam še, da preverimo, ali uporaba produkcij iz posameznih družin ohranja podane lastnosti:

- Družina 1: Uporaba produkcij iz te družine ohranja lastnosti iz leme, saj nobena od obeh članic družine ne uvede nobenega končno označenega vozlišča.

- Družina 2: Osredotočimo se na družino 2.2; dokaz za družino 2.1 je podoben. Produkcija družine 2.2, ki pripada razredu P, ustvari sledečo zvezdo:

$$\begin{aligned} P \xrightarrow{\vee} \{ & a_1 Q_1^1, \ldots, a_l Q_l^1, \\ & (a_{l+1}-1)\, Q_{l+1}^1, \ldots, (a_m - 1)\, Q_m^1, \\ & Q_{l+1}^{b_{l+1}-a_{l+1}+1}, \ldots, Q_m^{b_m - a_m + 1}, \overline{Q_{m+1}} \}. \end{aligned}$$

  Vozlišče s končno oznako P v središču zvezde poseduje lastnosti (1) in (2), saj pri vseh $i \in 1\,..\,l$ velja $u_i + v_i = 0 + a_i = a_i$. Lastnost (3) za vozlišče P prav tako velja, saj pri vseh $i \in l+1\,..\,m$ velja $u_i + v_i + w_i = 0 + (a_i - 1) + (b_i - a_i + 1) = b_i$. Med vozlišči z indeksi na intervalu $m+1\,..\,p$ je vozlišče P povezano samo z vozliščem $\overline{Q_{m+1}}$. Zato pri vseh $i \in m+1\,..\,p$ velja $u_i + v_i = 0 \in 0\,..\,b_i$, kar pomeni, da lastnost (4) tudi velja. Uporaba produkcij družine 2 potemtakem ohranja lastnost iz leme.

- Družina 3: Uporaba poljubne produkcije iz družine 3 poveča člen v_i v vsoti $u_i + v_i$ za nek $i \in k+1\,..\,l$. Ta sprememba ohranja neenakost $u_i + v_i \geq a_i$ iz lastnosti (2). Na ostale lastnosti iz leme družina 3 ne vpliva. Zato lahko trdimo, da uporaba družine 3 ohranja lastnosti iz leme.

- Družina 4: Dokaz, da družina 4 ohranja lastnosti iz leme, bomo zasnovali na podlagi sledeče pomožne trditve: *Ne obstaja indeks i, pri katerem bi bilo vozlišče P hkrati povezano tako z vozliščem $\overline{Q_i}$ kot z vozliščem Q_i^d za nek $d \geq 1$.* To sledi iz dejstva, da produkcije družin 2.2 in 4, ki rokujejo z vozlišči oblike $\overline{Q}$, zagotavljajo, da je vozlišče P povezano z največ enim takim vozliščem (recimo z vozliščem $\overline{Q_r}$ za nek $r \in m+1\,..\,p$) in da je indeks tega vozlišča (r), če obstaja, vedno večji od indeksov vseh vozlišč oblike Q_j^d.

  Vrnimo se k lemi 5.14. Produkcije družine 4 rokujejo z vozlišči $\overline{Q_i}$ pri $i \in m+1\,..\,p$ in lahko tako vplivajo zgolj na lastnost (4). Družina 4.3 ohranja vsoti $u_i + v_i$ in $u_i + v_i + w_i$ ter z njima lastnost (4), saj niti ne doda niti ne odvzame nobenega vozlišča oblike Q_j^d pri $d \geq 1$. Produkcije družin 4.1 in 4.2 pripnejo novo vozlišče $Q_i^{b_i}$ (za nek indeks $i \in m+1\,..\,p$) k vozlišču P. Ker — glede na pomožno trditev iz prejšnjega odstavka — vozlišče P ne more biti hkrati povezano tako z vozliščem $\overline{Q_i}$ kot z vozliščem Q_i^d, je po uporabi produkcije družine 4.1 ali 4.2 vozlišče $Q_i^{b_i}$ edino nekončno označeno vozlišče z indeksom i, ki je povezano z vozliščem P. Vsota $u_i + v_i + w_i$ potemtakem znaša $0 + 0 + b_i = b_i$, kar se sklada z lastnostjo (4) iz leme.

- Družina 5: Produkcije družin 5.1 in 5.2 lahko uporabimo samo na podgrafu $P \overset{\vee}{-} Q_i^{w_i}$, kjer velja $2 \leq w_i \leq b_i - a_i + 1$ pri nekem $i \in l+1\,..\,m$ ali pa $2 \leq w_i \leq b_i$ pri nekem $i \in m+1\,..\,p$. Predpostavimo, da velja $i \in l+1\,..\,m$; dokaz za $i \in m+1\,..\,p$ je podoben. Če lastnosti iz leme za dani podgraf veljajo, potem na podlagi lastnosti (3) za vozlišče P in indeks i velja enačba $u_i + v_i + w_i = b_i$. Preveriti moramo, ali lastnost (3) za isto vozlišče in isti indeks velja tudi po uporabi produkcije družine 5.1 ali 5.2 na danem podgrafu. (Ostale lastnosti se na indeks $i \in l+1\,..\,m$ ne nanašajo.)

  Produkcija družine 5.1 zmanjša člen w_i za 1 in hkrati poveča člen v_i (število povezav vozlišča P z vozlišči Q_i^1) za 1. Vsota $u_i + v_i + w_i$ se tako ohrani, kar pomeni, da smo lemo za družino 5.1 dokazali. Produkcija družine 5.2 zamenja vozlišče $Q_i^{w_i}$ v podgrafu $P \overset{\vee}{-} Q_i^{w_i}$ z vozliščem Q_i^1. To dejanje ne vpliva na člen u_i ($u_i' = u_i$), vendar pa poveča člen v_i ($v_i' = v_i + 1$). Ker vozlišče P po uporabi produkcije družine 5.2 ni več povezano z nobenim vozliščem oblike $Q_i^{w_i}$ pri $w_i \geq 2$, moramo preveriti, ali velja $a_i \leq u_i' + v_i' \leq b_i$. Ta lastnost drži, saj velja $u_i' + v_i' = u_i + (v_i + 1) = (u_i + v_i + w_i) + (1 - w_i) = b_i + 1 - w_i$, vrednost tega izraza pa glede na lastnost $2 \leq w_i \leq b_i - a_i + 1$ leži med a_i in $b_i - 1$. Ker torej velja $a_i \leq u_i' + v_i' \leq b_i$, lahko zaključimo, da družina 5.2 prav tako ohranja lastnost iz leme.

- Družina 7: Uporaba produkcije družine 7 na dvojici vozlišč P in Q ne spremeni vsot $u_i + v_i$ in $u_i + v_i + w_i$, saj vozlišče P izgubi povezavo z vozliščem Q^1, obenem pa pridobi povezavo z vozliščem Q, iz česar sledi $u' + v' = (u+1) + (v-1) = u + v$. Enako velja za vozlišče Q. Družina 7 potemtakem ohranja lastnost iz leme.

□

Lema 5.15. Naj bo razred P v metamodelu MM povezan z razredi $Q_1, \ldots, Q_p$. Potem v vsakem grafu, izpeljivem v gramatiki $GG(MM)$, velja sledeča lastnost za vsako vozlišče P:

Pri vsakem $i \in 1\,..\,p$ je vozlišče P povezano z u_i vozlišči Q_i, v_i vozlišči Q_i^1, y_i vozlišči $Q_i^{w_i}$ (kjer je $w_i \geq 2$) in največ enim vozliščem $\overline{Q_i}$, tako da velja $u_i \geq 0$, $v_i \geq 0$, $y_i \in \{0, 1\}$ in $u_i + v_i + y_i w_i \in \mathit{mult}(P, Q_i)$.

Dokaz. Lema 5.15 neposredno sledi iz močnejše leme 5.14, saj lahko vsako enačbo ali neenačbo, ki vsebuje vsoto $u_i + v_i$ ali $u_i + v_i + w_i$, prepišemo ali posplošimo kot $u_i + v_i + y_i w_i \in \mathit{mult}(P, Q_i)$. Na primer, enačbo $u_i + v_i + w_i = b_i$ in neenačbo $a_i \leq u_i + v_i \leq b_i$ iz lastnosti (3) v lemi 5.14 lahko združimo in posplošimo (oslabimo) v neenačbo $u_i + v_i + y_i w_i \in a_i\,..\,b_i$. V primeru $y_i = 0$ dobimo neenačbo, v primeru $y_i = 1$ pa posplošitev enačbe. □

Trditev 5.16. Če graf pripada jeziku $L(GG(MM))$, potem je tudi skladen z metamodelom MM. Z drugimi besedami: $L(GG(MM)) \subseteq L(MM)$.

Dokaz. Če velja $G \in L(GG(MM))$, potem je graf G končno označen in izpeljiv v gramatiki $GG(MM)$. Ker je graf G izpeljiv, zanj velja lastnost iz leme 5.15. Torej za vsako dvojico povezanih razredov P in Q_i velja, da je vsako vozlišče P v grafu G povezano z u_i vozlišči Q_i, v_i vozlišči Q_i^1, y_i vozlišči $Q_i^{w_i}$ (kjer je $w_i \geq 2$) in največ enim vozliščem $\overline{Q_i}$, tako da velja $u_i \geq 0$, $v_i \geq 0$, $y_i \in \{0, 1\}$ in $u_i + v_i + y_i w_i \in \mathit{mult}(P, Q_i)$. Ker pa so vsa vozlišča grafa G končno označena, sta vrednosti spremenljivk v_i in y_i enaki 0 pri vseh i. Zaradi tega se pri vseh indeksih i neenačbe $u_i + v_i + y_i w_i \in \mathit{mult}(P, Q_i)$ poenostavijo v $u_i \in \mathit{mult}(P, Q_i)$. Ker u_i predstavlja število vozlišč Q_i, povezanih z vozliščem P, relacija $u_i \in \mathit{mult}(P, Q_i)$ pomeni, da je to število skladno z multiplikativnostjo za smer $P \to Q_i$, ki jo predpisuje metamodel MM. Ker takšno sklepanje velja za poljuben par povezanih vozlišč P in Q_i, lahko zaključimo, da je graf G v celoti skladen z metamodelom MM. Torej velja $G \in L(MM)$. □

5.4.4.3 Dokaz lastnosti $L(MM) \subseteq L(GG(MM))$

Če želimo dokazati lastnost $L(MM) \subseteq L(GG(MM))$, moramo za vsak modelski graf G, ki je skladen z metamodelom MM, pokazati, da ga je mogoče z uporabo produkcij gramatike $GG(MM)$ izpeljati iz začetnega grafa gramatike ali pa (drugi način) da ga je mogoče pretvoriti v začetni graf z uporabo produkcij v obratni smeri. Lastnost bomo dokazali na drugi način, še pred tem pa bomo postavili in dokazali pomožno trditev, ki nam bo koristila pri glavnem dokazu.

Lema 5.17. Naj bo razred P v metamodelu povezan z razredi $Q_1, \ldots, Q_p$ tako, kot narekuje enačba (5.4). Potem je z obratno uporabo produkcij družine 4 mogoče vsak graf oblike $P \overset{\vee}{-} \{Q_{i[1]}^{b_{i[1]}}, \ldots, Q_{i[r]}^{b_{i[r]}}\}$, kjer velja $1 \leq r \leq p - m$ in $m + 1 \leq i[1] < \ldots < i[r] \leq p$, pretvoriti v graf $G_I \equiv P \overset{\vee}{-} \overline{Q_{m+1}}$.

Dokaz. Naj bo graf G oblike $P \overset{\vee}{-} \{Q_{i[1]}^{b_{i[1]}}, \ldots, Q_{i[r]}^{b_{i[r]}}\}$, pri čemer velja $1 \leq r \leq p - m$ in $m + 1 \leq i[1] < \ldots < i[r] \leq p$. Enkratna obratna uporaba produkcije družine 4.1 pretvori graf G v graf $H_r \equiv P \overset{\vee}{-} \{Q_{i[1]}^{b_{i[1]}}, \ldots, Q_{i[r-1]}^{b_{i[r-1]}}, \overline{Q_{i[r]}}\}$. Sedaj moramo pokazati, da je graf H_r pri poljubnem $r \geq 1$ možno pretvoriti v graf $G_I \equiv P \overset{\vee}{-} \overline{Q_{m+1}}$:

- (Primer $r = 1$) Graf $H_1 = P \overset{\vee}{-} \overline{Q_{i[1]}}$ pretvorimo v graf $G_I = P \overset{\vee}{-} \overline{Q_{m+1}}$ z $(i[1] - m - 1)$ zaporednimi obratnimi uporabami produkcije družine 4.3.

- (Splošen primer) Pri splošnem r lahko graf H_r pretvorimo v graf H_{r-1} na sledeči način: Če velja $i[r] > i[r-1]+1$, potem najprej z $(i[r]-i[r-1]-1)$ obratnimi uporabami produkcije družine 4.3 pretvorimo graf H_r v graf $P \overset{\vee}{-} \{Q_{i[1]}^{b_{i[1]}}, \ldots, Q_{i[r-2]}^{b_{i[r-2]}}, Q_{i[r-1]}^{b_{i[r-1]}}, \overline{Q_{i[r-1]+1}}\}$. Sedaj pa nastali graf (oziroma graf H_r, če je na začetku veljalo $i[r] = i[r-1]+1$) z enkratno obratno uporabo produkcije družine 4.2 pretvorimo v graf $H_{r-1} = P \overset{\vee}{-} \{Q_{i[1]}^{b_{i[1]}}, \ldots, Q_{i[r-2]}^{b_{i[r-2]}}, \overline{Q_{i[r-1]}}\}$. Ker opisana pretvorba $H_r \to H_{r-1}$ deluje pri vseh $r > 1$, lahko graf H_r postopoma pretvorimo v graf H_1 in od tam v graf G_I. □

Trditev 5.18. Če graf pripada jeziku $L(MM)$, potem ga je mogoče izpeljati v gramatiki $GG(MM)$. Z drugimi besedami: $L(MM) \subseteq L(GG(MM))$.

Dokaz. Pokazali bomo, da je poljuben graf $G \in L(MM)$ možno z obratnimi uporabami produkcij gramatike $GG(MM)$ pretvoriti v začetni graf gramatike, tj. graf, sestavljen zgolj iz vozlišča z oznako V. Naj bo razred P v metamodelu MM povezan z razredi $Q_1, \ldots, Q_p$ v skladu z enačbo (5.4). Osredotočimo se na neko vozlišče z oznako P v grafu G. Ker to vozlišče predstavlja objekt razreda P in ker je G po predpostavki veljaven modelski graf, je vozlišče P povezano s t_1 vozlišči Q_1, t_2 vozlišči Q_2, ... in s t_p vozlišči Q_p, tako da velja $t_i \in mult(P, Q_i)$ pri vseh $i \in 1\,..\,p$. Z obratnimi uporabami produkcij družine 7 sedaj vozlišče P in povezave, pripete nanj, pretvorimo v zvezdo $P \overset{\vee}{-} \{t_1 Q_1^1, \ldots, t_p Q_p^1\}$. (Ta pretvorba vpliva tudi na vozlišča $Q_1, \ldots, Q_p$, saj izgubijo svoje povezave z vozliščem P, toda ta vozlišča bomo kasneje tako ali tako pretvorili v zvezde na enak način kot vozlišče P.) Zvezdo $P \overset{\vee}{-} \{t_1 Q_1^1, \ldots, t_p Q_p^1\}$ nato obdelamo po sledečih korakih:

1. Za vsak $i \in k+1\,..\,l$ zmanjšamo število povezav med vozliščem P in vozlišči z oznako Q_i^1 s t_i na a_i. To dosežemo s $(t_i - a_i)$ obratnimi uporabami ustrezne produkcije družine 3.

2. Za vsak $i \in l+1\,..\,m$ pretvorimo enega od sosedov vozlišča P z oznako Q_i^1 v vozlišče z oznako $Q_i^{b_i - t_i + 1}$. To dosežemo z obratno uporabo ustrezne produkcije družine 5.2. Nastalo vozlišče in $(t_i - a_i)$ vozlišč Q_i^1 nato skupaj pretvorimo v vozlišče $Q_i^{b_i - a_i + 1}$, kar dosežemo s $(t_i - a_i)$ obratnimi uporabami ustrezne produkcije družine 5.1. Vozlišče P je sedaj povezano s po $(a_i - 1)$ vozlišči Q_i^1 in s po enim vozliščem $Q_i^{b_i - a_i + 1}$ za vsak $i \in l+1\,..\,m$.

3. Med števili $t_{m+1}, \ldots, t_p$ označimo s $t_{i[1]}, \ldots, t_{i[r]}$ (kjer velja $0 \le r \le p-m$ in $i[1] < \ldots < i[r]$) tista, ki niso enaka 0. Če je $r = 0$, ne naredimo ničesar. Sicer pa za vsak $j \in 1\,..\,r$ pretvorimo $t_{i[j]}$ vozlišč $Q_{i[j]}^1$ v vozlišče $Q_{i[j]}^{b_{i[j]}}$. To dosežemo z obratnimi uporabami produkcij družine 5, in sicer na podoben način kot v koraku 2. Vozlišče P je sedaj povezano z vozliščem $Q_{i[1]}^{b_{i[1]}}$, vozliščem $Q_{i[2]}^{b_{i[2]}}$, ... in z vozliščem $Q_{i[r]}^{b_{i[r]}}$, kar pomeni, da ustreza predpostavki v lemi 5.17. Z uporabo navodil iz dokaza leme 5.17 pretvorimo vsa vozlišča z indeksi z intervala $m+1\,..\,p$, povezana z vozliščem P, v eno samo vozlišče z oznako $\overline{Q_{m+1}}$.

Po opisanih pretvorbah ima zvezda za vozlišče P obliko $P \overset{\vee}{-\!\!-} \mathcal{Q}$ (če je v koraku 3 veljalo $r = 0$) ali obliko $P \overset{\vee}{-\!\!-} \mathcal{Q} \cup \{\overline{Q_{m+1}}\}$ (v nasprotnem primeru), pri čemer je $\mathcal{Q} = \{a_1 Q_1^1, \ldots, a_l Q_l^1, (a_{l+1}-1)Q_{l+1}^1, \ldots, (a_m-1)Q_m^1, Q_{l+1}^{b_{l+1}-a_{l+1}+1}, \ldots, Q_m^{b_m-a_m+1}\}$. Zvezda se v obeh primerih ujema z desno stranjo produkcije družine 2, ki pripada razredu P. Obratna uporaba te produkcije pretvori zvezdo v eno samo vozlišče W. Če opisani pretvorni postopek ponovimo za vsa vozlišča izhodiščnega grafa G, pretvorimo celoten graf v $|\mathcal{V}[G]|$ nepovezanih vozlišč z oznako W. Po obratni uporabi produkcije 1.3 in po $(|\mathcal{V}[G]|-1)$ obratnih uporabah produkcije 1.2 dobimo eno samo vozlišče z oznako V. Graf G smo torej uspešno pretvorili v začetni graf gramatike $GG(MM)$. Trditev je tako dokazana. □

V pravkar predstavljenem dokazu je opisan učinkovit polinomski algoritem za sintaksno analizo modelskih grafov v izhodni gramatiki $GG(MM)$. Kot bomo pokazali v razdelku 5.7, lahko izpeljava modelskega grafa v izhodni gramatiki služi kot osnova za uporabo semantičnih pravil.

5.5 Obravnava dedovanja

V tem podpoglavju bomo v našo pretvorno metodo vključili obravnavo dedovanja in abstraktnih razredov (razdelek 5.5.1). Pokazali bomo, da opisana razširitev metode ohranja sintaksno odločljivost izhodne gramatike in enakovrednost jezikov (razdelek 5.5.2).

5.5.1 Razširitev pretvorne metode

Na prvi pogled bi lahko domnevali, da je metamodel z dedovanjem mogoče enostavno prepisati v enakovreden metamodel brez dedovanja. Žal pa to v splošnem ni tako. Oglejmo si metamodel MM_{pub} (slika 5.1 na strani 155). Morda se zdi, da lahko, denimo, dedni odnos $\mathsf{J} \to \mathsf{P}$ odpravimo tako, da razreda J in R povežemo s kopijo asociacije med razredoma P in R. Ta pristop bi deloval v smeri $\mathsf{J} \to \mathsf{R}$, ne pa v obratni smeri. Razlog je v tem, da multiplikativnost $1..*$ v smeri $\mathsf{R} \to \mathsf{P}$ zahteva, da mora biti *skupno* število povezav $\mathsf{R} \to \mathsf{J}$ in $\mathsf{R} \to \mathsf{CP}$ v veljavnem modelskem grafu enako najmanj 1 (gl. definicijo 5.10 na strani 163). Takšnih zahtev brez uporabe dedovanja ne moremo enostavno predstaviti.

Definirajmo tri koncepte, tesno povezane z dedovanjem:

Definicija 5.19 (razširjena relacija asociacije)**.** Za metamodel ($\mathcal{C}$, $\mathcal{C}_{\text{A}}$, **Assoc**, *mult*, **Inh**) je *razširjena relacija asociacije* $\mathbf{Assoc}_{\text{ext}} \subseteq \mathcal{C} \times \mathcal{C}$ definirana kot

$$\mathbf{Assoc}_{\text{ext}} = \mathbf{Assoc} \cup \{(P,\, Q) \in \mathcal{C} \times \mathcal{C} \mid \exists R \in \mathcal{C} \colon P\, \mathbf{Inh}^+\, R \wedge R\, \mathbf{Assoc}\, Q\}. \quad (5.6)$$

Razširjena relacija asociacije ni nujno simetrična. Denimo, v metamodelu MM_{pub} velja $\mathsf{J}\ \mathbf{Assoc}_{\text{ext}}\ \mathsf{R}$, toda $\neg(\mathsf{R}\ \mathbf{Assoc}_{\text{ext}}\ \mathsf{J})$.

Definicija 5.20 (razširjena funkcija asociacije)**.** *Razširjena funkcija asociacije* $Assoc_{\text{ext}} \colon \mathcal{C} \to \mathcal{P}(\mathcal{C})$ je definirana kot $Assoc_{\text{ext}}(P) = \{Q \in \mathcal{C} \mid P\ \mathbf{Assoc}_{\text{ext}}\ Q\}$.

Definicija 5.21 (razširjena funkcija multiplikativnosti)**.** *Razširjena funkcija multiplikativnosti* $mult_{\text{ext}} \colon \mathcal{C} \times \mathcal{C} \to \mathbb{N}_0 \times (\mathbb{N}_0 \cup \{*\})$ je definirana kot

$$mult_{\text{ext}}(P, Q) = \begin{cases} mult(P, Q), & \text{če } P \,\mathbf{Assoc}\, Q; \\ mult(R, Q), & \text{če } \exists R\colon P \,\mathbf{Inh}^+\, R \wedge R \,\mathbf{Assoc}\, Q; \\ \text{nedefinirano} & \text{sicer.} \end{cases}$$

Pri vključitvi dedovanja v pretvorno metodo upoštevamo sledeči pravili, ki ju lahko izpeljemo iz definicije 5.10:

1. Če velja $R \,\mathbf{Inh}^+\, P$, potem razred R podeduje vse zahteve glede multiplikativnosti, ki so v metamodelu določene za razred P v smeri od razreda P proti povezanim razredom, saj po naši predpostavki asociacij ni mogoče redefinirati (zahteva (2) v predpostavki 5.9 na strani 163). Z drugimi besedami: pri vsakem razredu $Q \in \mathbf{Assoc}(P)$ je omejitev glede števila vezi med objektom R in objekti Q v veljavnem modelu enaka omejitvi glede števila vezi med objektom P in objekti Q, tj. $mult(P, Q)$. Multiplikativnost za smer $R \to Q$ je torej določena kot $mult_{\text{ext}}(R, Q)$.

2. Denimo, da za razrede P, Q in S velja $P \,\mathbf{Assoc}\, Q$ in $S \,\mathbf{Inh}^+\, Q$. Multiplikativnost $mult(P, Q)$ po definiciji 5.10 podaja omejitev glede skupnega števila vezi med posameznim objektom razreda P in objekti razredov iz množice $Inh^*(Q)$ v veljavnem modelu.

   Denimo, da gradimo modelski graf z uporabo produkcij gramatike $GG(MM)$. V zvezdi, ki jo za vozlišče P ustvari produkcija družine 2, listi z oznako Q^1 predstavljajo bodoče povezave med vozliščem P in posameznimi (morebiti še ne obstoječimi) vozlišči Q. Ker po predpostavki velja $S \,\mathbf{Inh}^+\, Q$, lahko v tej zvezdi poljubno mnogo vozlišč Q^1 zamenjamo z enakim številom vozlišč S^1, saj na ta način ohranimo skupno število bodočih povezav med vozliščem P in vozlišči z oznakami iz množice $Inh^*(Q)$. Pravila, ki ga določa multiplikativnost $mult(P, Q)$, tako ne kršimo.

   Opisano pravilo lahko še posplošimo: Če velja (1) $S \,\mathbf{Inh}^+\, Q$, (2) $P \,\mathbf{Assoc}\, Q$ in (3) $R \,\mathbf{Inh}^*\, P$, potem lahko v grafu, izpeljivem v gramatiki $GG(MM)$, poljubno vozlišče Q^1, ki je povezano z vozliščem R, zamenjamo z vozliščem S^1. Pogoja (2) in (3) lahko združimo v obliko $R \in Inh^*(Assoc(Q))$.

Gornji pravili bomo najprej prevedli v dodatna navodila za gradnjo gramatike $GG(MM)$, v razdelku 5.5.2 pa bomo pokazali, da posodobljena pretvorna metoda pravilno deluje tudi za metamodele z dedovanjem.

Pravilo 1 vključimo v pretvorno metodo tako, da namesto relacije **Assoc** ter funkcij *Assoc* in *mult* uporabimo relacijo $\mathbf{Assoc}_{\text{ext}}$ ter funkciji $Assoc_{\text{ext}}$ in $mult_{\text{ext}}$. Z izjemo te spremembe je pretvorni postopek povsem enak kot v podpoglavju 5.4. Na primer, v izhodni gramatiki za metamodel MM_{pub} imamo zaradi dejstva $Assoc_{\text{ext}}(\mathsf{CS}) = \{\mathsf{C}, \mathsf{R}, \mathsf{A}, \mathsf{K}\}$ sledeči produkciji družine 2 za razred CS:

$$\mathsf{W} ::= \mathsf{CS} \xrightarrow{\mathsf{v}} \{\mathsf{A}^1, \mathsf{R}^1, \mathsf{R}^2, 3\mathsf{K}^1\} \mid \mathsf{CS} \xrightarrow{\mathsf{v}} \{\mathsf{A}^1, \mathsf{R}^1, \mathsf{R}^2, 3\mathsf{K}^1, \overline{\mathsf{C}}\}$$

Ker podedovani multiplikativnosti za dvojici $\mathsf{CS} \to \mathsf{A}$ in $\mathsf{CS} \to \mathsf{K}$ nimata zgornje meje, razredu CS pripadata dve produkciji družine 3:

$$\mathsf{CS} ::= \mathsf{CS} \overset{\mathsf{v}}{\text{—}} \mathsf{A}^1 \mid \mathsf{CS} \overset{\mathsf{v}}{\text{—}} \mathsf{K}^1$$

Pravilo 2 lahko neposredno prepišemo v novo družino produkcij:

Družina 6: Družina 6 vsebuje sledečo produkcijo za vsako trojico razredov (Q, R, S), za katero velja $S\,\mathbf{Inh}^+\,Q$ in $R \in Inh^*(Assoc(Q))$:

$$R \overset{\mathsf{v}}{\text{—}} Q^1 ::= R \overset{\mathsf{v}}{\text{—}} S^1$$

V primeru metamodela MM_{pub} imamo zaradi lastnosti $Inh^+(\mathsf{S}) = \{\mathsf{CS}, \mathsf{JS}\}$ in $Inh^*(Assoc(\mathsf{S})) = \{\mathsf{R}, \mathsf{A}, \mathsf{K}\}$ sledeče produkcije družine 6:

$$\mathsf{R} \overset{\mathsf{v}}{\text{—}} \mathsf{S}^1 ::= \mathsf{R} \overset{\mathsf{v}}{\text{—}} \mathsf{CS}^1 \mid \mathsf{R} \overset{\mathsf{v}}{\text{—}} \mathsf{JS}^1$$
$$\mathsf{A} \overset{\mathsf{v}}{\text{—}} \mathsf{S}^1 ::= \mathsf{A} \overset{\mathsf{v}}{\text{—}} \mathsf{CS}^1 \mid \mathsf{A} \overset{\mathsf{v}}{\text{—}} \mathsf{JS}^1$$
$$\mathsf{K} \overset{\mathsf{v}}{\text{—}} \mathsf{S}^1 ::= \mathsf{K} \overset{\mathsf{v}}{\text{—}} \mathsf{CS}^1 \mid \mathsf{K} \overset{\mathsf{v}}{\text{—}} \mathsf{JS}^1$$

Pojasniti moramo še obravnavo abstraktnih razredov. Spomnimo se, da noben veljaven model ne more vsebovati objektov abstraktnih razredov (zahteva (1) v definiciji 5.10). To pravilo lahko uveljavimo enostavno tako, da dodamo oznake vseh abstraktnih razredov v množico nekončnih oznak izhodne gramatike. Na primer, v izhodni gramatiki za metamodel MM_{pub} moramo oznaki P in S obravnavati kot nekončni. Zato so vsi grafi, ki vsebujejo vozlišča s takšnimi oznakami, nekončno označeni in zaradi tega ne pripadajo jeziku izhodne gramatike.

Sedaj smo opremljeni z vsem potrebnim za zapis izhodne gramatike za metamodel MM_{pub}. Njene produkcije so navedene v tabeli 5.4.

5.5.2 Sintaksna odločljivost in enakovrednost jezikov

Za dokaz sintaksne odločljivosti in trditve $L(GG(MM)) \subseteq L(MM)$ bi načeloma morali — z ustreznimi prilagoditvami — ponoviti dolgovezne dokaze iz podpoglavja 5.4, saj dedovanja ne moremo obravnavati ločeno od multiplikativnosti. Vendar pa je edina netrivialna sprememba pretvornega postopka nova družina 6. Na ostale družine produkcij vpeljava dedovanja ni vplivala, razen da sedaj uporabljamo razširjene asociacije in multiplikativnosti namesto "navadnih". Zaradi tega se bomo osredotočili na družino 6.

Pri plastni funkciji, ki smo jo podali v tabeli 5.2 na strani 175, produkcije družine 6 ne izpolnjujejo plastnega pogoja, saj velja $layer(Q^1) = layer(S^1) = 1$. Ta problem rešimo s spremembo plastne funkcije v skladu s sledečim pravilom: če velja $S\,\mathbf{Inh}\,Q$, potem določimo $layer(S^1) < layer(Q^1)$. Razmerja med plastmi, ki zagotavljajo skladnost s plastnim pogojem, ohranimo tako, da oznake W, V in v ter oznake oblike P^d (pri $d \geq 2$) in $\overline{P}$ prestavimo za $\max_i\{layer(P_i^1)\}$ plasti višje. Pri tako spremenjeni plastni funkciji vse družine produkcij izpolnjujejo plastni pogoj, zato lahko trdimo, da je izhodna grafna gramatika sintaksno odločljiva.

Tabela 5.4: *Produkcije izhodne grafne gramatike za metamodel MM_{pub} (slika 5.1).*

družina	produkcije
1	$\lambda ::= \mathsf{V}$ $\mathsf{V} ::= \mathsf{V};\ \mathsf{W} \mid \mathsf{W}$
2	$\mathsf{W} ::= \mathsf{P} \overset{\mathsf{v}}{-} \mathsf{R}^1 \mid$ $\mathsf{J} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{JS}^1\} \mid$ $\mathsf{CP} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{C}^1\} \mid$ $\mathsf{C} \overset{\mathsf{v}}{-} \{\mathsf{CP}^1, \mathsf{CS}^1\} \mid$ $\mathsf{S} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^2, \mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1\} \mid$ $\mathsf{CS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^2, \mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1\} \mid$ $\mathsf{CS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^2, \mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1, \overline{\mathsf{C}}\} \mid$ $\mathsf{JS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^2, \mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1\} \mid$ $\mathsf{JS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^2, \mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1, \overline{\mathsf{J}}\} \mid$ $\mathsf{R} \overset{\mathsf{v}}{-} \mathsf{P}^1 \mid$ $\mathsf{A} \mid$ K
3	$\mathsf{P} ::= \mathsf{P} \overset{\mathsf{v}}{-} \mathsf{R}^1$ $\mathsf{J} ::= \mathsf{J} \overset{\mathsf{v}}{-} \mathsf{R}^1 \mid \mathsf{J} \overset{\mathsf{v}}{-} \mathsf{JS}^1$ $\mathsf{CP} ::= \mathsf{CP} \overset{\mathsf{v}}{-} \mathsf{R}^1$ $\mathsf{C} ::= \mathsf{C} \overset{\mathsf{v}}{-} \mathsf{CS}^1$ $\mathsf{S} ::= \mathsf{S} \overset{\mathsf{v}}{-} \mathsf{A}^1 \mid \mathsf{S} \overset{\mathsf{v}}{-} \mathsf{K}^1$ $\mathsf{CS} ::= \mathsf{CS} \overset{\mathsf{v}}{-} \mathsf{A}^1 \mid \mathsf{CS} \overset{\mathsf{v}}{-} \mathsf{K}^1$ $\mathsf{JS} ::= \mathsf{JS} \overset{\mathsf{v}}{-} \mathsf{A}^1 \mid \mathsf{JS} \overset{\mathsf{v}}{-} \mathsf{K}^1$ $\mathsf{R} ::= \mathsf{R} \overset{\mathsf{v}}{-} \mathsf{P}^1 \mid \mathsf{R} \overset{\mathsf{v}}{-} \mathsf{S}^1$ $\mathsf{A} ::= \mathsf{A} \overset{\mathsf{v}}{-} \mathsf{S}^1$ $\mathsf{K} ::= \mathsf{K} \overset{\mathsf{v}}{-} \mathsf{S}^1$
4	$\mathsf{CS} \overset{\mathsf{v}}{-} \overline{\mathsf{C}} ::= \mathsf{CS} \overset{\mathsf{v}}{-} \mathsf{C}^1$ $\mathsf{JS} \overset{\mathsf{v}}{-} \overline{\mathsf{J}} ::= \mathsf{JS} \overset{\mathsf{v}}{-} \mathsf{J}^1$
5	$\mathsf{S} \overset{\mathsf{v}}{-} \mathsf{R}^2 ::= \mathsf{S} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^1\} \mid \mathsf{S} \overset{\mathsf{v}}{-} \mathsf{R}^1$ $\mathsf{CS} \overset{\mathsf{v}}{-} \mathsf{R}^2 ::= \mathsf{CS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^1\} \mid \mathsf{CS} \overset{\mathsf{v}}{-} \mathsf{R}^1$ $\mathsf{JS} \overset{\mathsf{v}}{-} \mathsf{R}^2 ::= \mathsf{JS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^1\} \mid \mathsf{JS} \overset{\mathsf{v}}{-} \mathsf{R}^1$
6	$\mathsf{R} \overset{\mathsf{v}}{-} \mathsf{P}^1 ::= \mathsf{R} \overset{\mathsf{v}}{-} \mathsf{J}^1 \mid \mathsf{R} \overset{\mathsf{v}}{-} \mathsf{CP}^1$ $\mathsf{R} \overset{\mathsf{v}}{-} \mathsf{S}^1 ::= \mathsf{R} \overset{\mathsf{v}}{-} \mathsf{CS}^1 \mid \mathsf{R} \overset{\mathsf{v}}{-} \mathsf{JS}^1$ $\mathsf{A} \overset{\mathsf{v}}{-} \mathsf{S}^1 ::= \mathsf{A} \overset{\mathsf{v}}{-} \mathsf{CS}^1 \mid \mathsf{A} \overset{\mathsf{v}}{-} \mathsf{JS}^1$ $\mathsf{K} \overset{\mathsf{v}}{-} \mathsf{S}^1 ::= \mathsf{K} \overset{\mathsf{v}}{-} \mathsf{CS}^1 \mid \mathsf{K} \overset{\mathsf{v}}{-} \mathsf{JS}^1$
7	$P \overset{\mathsf{v}}{-} Q^1;\ Q \overset{\mathsf{v}}{-} P^1 ::= P - Q$ za vse pare $(P, Q) \in \{(\mathsf{P}, \mathsf{R}),\ (\mathsf{J}, \mathsf{R}),\ (\mathsf{CP}, \mathsf{R}),\ (\mathsf{J}, \mathsf{JS}),$ $(\mathsf{CP}, \mathsf{C}),\ (\mathsf{C}, \mathsf{CS}),\ (\mathsf{S}, \mathsf{A}),\ (\mathsf{S}, \mathsf{K}),\ (\mathsf{S}, \mathsf{R}),\ (\mathsf{CS}, \mathsf{A}),\ (\mathsf{CS}, \mathsf{K}),$ $(\mathsf{CS}, \mathsf{R}),\ (\mathsf{JS}, \mathsf{A}),\ (\mathsf{JS}, \mathsf{K}),\ (\mathsf{JS}, \mathsf{R})\}$

Preden utemeljimo enakovrednost jezikov, uvedimo pomožno definicijo. Graf, izpeljiv v gramatiki $GG(MM)$, je *potencialno veljaven*, če se bo po zaključku izpeljave v gramatiki $GG(MM)$, torej po izginotju vseh nekončno označenih elementov, razvil v veljaven modelski graf. To ne pomeni nujno, da je tak graf sploh *mogoče* razviti v veljaven graf; če je jezik $L(MM)$ prazen, to prav gotovo ni mogoče. Vendar pa mora veljati sledeče: če je graf mogoče pretvoriti v končno označen graf, potem je nastali graf pripadnik jezika $L(MM)$. V dokazu trditve 5.16 smo dejansko pokazali, da so vsi grafi, izpeljivi v gramatiki $GG(MM)$, potencialno veljavni, iz česar sledi $L(GG(MM)) \subseteq L(MM)$. V naslednjem odstavku bomo utemeljili, da gramatika ohranja to lastnost tudi po uvedbi družine 6.

Veljavnost lastnosti $L(GG(MM)) = L(MM)$ ponovno pokažemo s prevedbo na lastnosti $L(GG(MM)) \subseteq L(MM)$ in $L(MM) \subseteq L(GG(MM))$. Na vprašanje, ali vpeljava družine 6 ohranja prvo lastnost, lahko odgovorimo tako, da preverimo, ali uporaba produkcije družine 6 na potencialno veljavnem grafu privede do nekega drugega potencialno veljavnega grafa. Upoštevajmo, da produkcija družine 6 pretvori podgraf oblike $R \overset{\vee}{—} Q^1$ v podgraf $R \overset{\vee}{—} S^1$, kjer je $S \,\mathbf{Inh}^+\, Q$. Produkcija torej zamenja bodočo povezavo vozlišča R z vozliščem Q (to bodočo povezavo predstavlja vozlišče Q^1) z bodočo povezavo z vozliščem z oznako iz množice $Inh^+(Q)$ (to bodočo povezavo predstavlja vozlišče S^1). Opisana zamenjava ne spremeni skupnega števila bodočih povezav med vozliščem R in vozlišči iz množice $Inh^*(Q)$, zato se skladnost z multiplikativnostjo $mult(R, Q)$ (če velja $R \in Assoc(Q)$) oziroma $mult_{\text{ext}}(R, Q)$ (če velja $R \in Inh^+(Assoc(Q))$) ohrani. Torej je graf, ki ga dobimo z uporabo produkcije 6 na potencialno veljavnem grafu, tudi potencialno veljaven.

Da bi dokazali lastnost $L(MM) \subseteq L(GG(MM))$, izberimo graf G iz jezika $L(MM)$ in ga poskusimo z obratnimi uporabami produkcij gramatike pretvoriti v začetni graf gramatike (graf, ki ga tvori vozlišče z oznako V). Tako kot v dokazu trditve 5.18 bomo najprej z obratnimi uporabami produkcij družine 7 pretvorili posamezna vozlišča grafa G v zvezde. Vsaka zvezda zavzema obliko $P \overset{\vee}{—} \{t_1 Q_1^1, \ldots, t_p Q_p^1\}$, kjer je $p \geq 0$ in $t_i \geq 0$ za vsak $i \in 1\,..\,p$. Osredotočimo se na list $Q^1 \equiv Q_i^1$ za nek $i \in 1\,..\,p$. V splošnem ni nujno, da sta razreda P in Q neposredno povezana, vsekakor pa morata obstajati razreda P' in Q', tako da velja $P \,\mathbf{Inh}^*\, P'$, $Q \,\mathbf{Inh}^*\, Q'$ in $P' \,\mathbf{Assoc}\, Q'$. Podgraf $P \overset{\vee}{—} Q^1$ lahko sedaj pretvorimo v $P \overset{\vee}{—} Q'^1$ s pomočjo obratne uporabe produkcije $P \overset{\vee}{—} Q'^1 ::= P \overset{\vee}{—} Q^1$, ki po definiciji družine 6 zanesljivo obstaja (če je seveda $Q' \neq Q$). Zato lahko zvezdo $P \overset{\vee}{—} \{t_1 Q_1^1, \ldots, t_p Q_p^1\}$ pretvorimo v zvezdo $P \overset{\vee}{—} \{t_1 Q_1'^1, \ldots, t_p Q_p'^1\}$, pri čemer so razredi $Q_1', \ldots, Q_p'$ določeni na enak način kot razred Q'. Pri vsakem $i \in 1\,..\,p$ velja $P \,\mathbf{Assoc}\, Q_i'$ ali $P' \,\mathbf{Assoc}\, Q_i'$, kjer je $P \,\mathbf{Inh}^*\, P'$. V obeh primerih lahko nastalo zvezdo pretvorimo v vozlišče W na enak način kot v dokazu trditve 5.18, saj se dedni odnos $P \,\mathbf{Inh}^*\, P'$ zrcali v gramatiki $GG(MM)$ (vsaki produkciji za vozlišče P' pripada produkcija za vozlišče P, ki odseva dedni odnos).

5.6 Računska zahtevnost

V tem podpoglavju bomo ocenili računsko zahtevnost pretvorne metode. Računska zahtevnost je popolnoma določena z velikostjo izhodne gramatike (v smislu definicije 2.23 na strani 40) za podani metamodel $MM = (\mathcal{C}, \mathcal{C}_{\text{A}}, \mathbf{Assoc}, mult, \mathbf{Inh})$, saj

razen zapisa izhodne gramatike pretvorna metoda ne potrebuje nobenega upoštevanja vrednega dodatnega časa ali prostora. V nadaljevanju bomo z besedno zvezo »kompleksnost družine d« označevali skupno velikost vseh produkcij družine d. Naj opozorimo, da nas zanima vsota velikosti posameznih produkcij v posameznih družinah, ne število produkcij po družinah. Računsko zahtevnost pretvorne metode bomo ocenili kot funkcijo števila razredov v vhodnem metamodelu ($|\mathcal{C}|$) in morebitnih drugih parametrov metamodela. Funkcije ne bomo podali natančno, ampak bomo, kot je to običajno pri zapisovanju računske zahtevnosti, ocenili zgolj njeno asimptotično zgornjo mejo.

Ocenimo kompleksnosti posameznih družin produkcij. Kompleksnost družine 1 ni odvisna od metamodela, zato jo lahko zapišemo kar kot $O(1)$. Kompleksnost družine 2 znaša $O(|\mathcal{C}|^2 \log A)$, kjer je A največja spodnja meja med multiplikativnostmi v metamodelu:

$$A = \max\{a \in \mathbb{N}_0 \mid \exists P,\, Q \in \mathcal{C}\colon\, mult(P,\, Q) = a\,..\,b \vee mult(P,\, Q) = a\,..\,*\} \quad (5.7)$$

To sledi iz dejstev, da produkcije družine 2 na desnih straneh vsebujejo največ po dve zvezdi za vsakega od $|\mathcal{C}|$ razredov, da vsaka zvezda vsebuje največ po $|\mathcal{C}|$ različno označenih listov in da med listi posamezne zvezde vsaka oznaka nastopa v največ A kopijah. Na primer, razredu D v metamodelu na sliki 5.6 (stran 172) pripada zvezda $\mathsf{D} \overset{\vee}{—} \{\mathsf{A}^3,\, 3\mathsf{E}^1\}$ (tabela 5.1 na strani 173), ki zaradi multiplikativnosti $3\,..\,3$ vsebuje tri liste z oznako E. Po strogi interpretaciji definicije 2.23 bi morali kompleksnost družine 2 zapisati kot $O(|\mathcal{C}|^2 A)$. Ker pa lahko A listov z isto oznako učinkovito zapišemo z $O(\log A)$ biti (namesto da vsako od A kopij zapišemo posebej, lahko zapišemo le eno kopijo in sámo število A), je kompleksnost družine 2 dejansko enaka $O(|\mathcal{C}|^2 \log A)$.

Družine 3, 4 in 7 imajo kompleksnost $O(|\mathcal{C}|^2)$, ker vsaka od njih vsebuje konstantno število produkcij na asociacijo (skupno število asociacij je kvečjemu reda $O(|\mathcal{C}|^2)$), vsaka produkcija pa ima konstantno velikost. Kompleksnost družine 6 je prav tako $O(|\mathcal{C}|^2)$. Preden to trditev pojasnimo, dokažimo sledečo lemo:

Lema 5.22. V množici produkcij družine 6 se vsaka desna stran pojavi največ po enkrat.

Dokaz. Predpostavimo nasprotno: da obstaja graf $R \overset{\vee}{—} Q^1$, ki služi kot desna stran dveh različnih produkcij družine 6. Levi strani teh dveh produkcij se morata potemtakem glasiti $R \overset{\vee}{—} P'^1$ in $R \overset{\vee}{—} P''^1$, kjer je $Q\,\mathbf{Inh}^+\,P'$ in $Q\,\mathbf{Inh}^+\,P''$, iz česar sledi bodisi $P'\,\mathbf{Inh}^*\,P''$ ali $P''\,\mathbf{Inh}^+\,P'$ (zaradi predpostavke o enkratnem dedovanju, gl. zahtevo (1) v predpostavki 5.9 na strani 163). Če takšna dvojica levih strani obstaja, potem morata biti razreda P' in P'' oba neposredno ali posredno povezana z razredom R. Vendar pa je zaradi predpostavljene prepovedi redefinicije asociacije (zahteva (2) v predpostavki 5.9) to mogoče samo v primeru, če velja $P' = P''$. Zaradi tega sta obe levi strani — in zato tudi obe produkciji — enaki, kar nasprotuje naši predpostavki z začetka dokaza. Lemo smo tako dokazali s protislovjem. □

Ker vse desne strani produkcij družine 6 zavzemajo obliko $R \overset{\vee}{—} Q$, družina 6 vsebuje največ $|\mathcal{C}|^2$ različnih desnih strani in zaradi leme 5.22 tudi največ $|\mathcal{C}|^2$ produkcij. Kompleksnost družine 6 tako res znaša $O(|\mathcal{C}|^2)$.

Kompleksnost družine 5 je premosorazmerna tako s številom asociacij kot tudi z največjo razliko med zgornjo in spodnjo mejo med multiplikativnostmi tipa $a\,..\,b$. Za vsako dvojico razredov $(P,\,Q)$ z lastnostjo $\mathit{mult}(P,\,Q) = a\,..\,b$ $(0 \leq a < b)$ velja, da družina 5 vsebuje po dve produkciji za vsako od levih strani iz množice $\{P \overset{\vee}{\text{—}} Q^2,$ $\ldots,\ P \overset{\vee}{\text{—}} Q^t\}$, kjer je $t = b - a + 1$ v primeru $a \geq 1$ oziroma $t = b$ v primeru $a = 0$. Zaradi tega je kompleksnost družine 5 enaka $O(|\mathcal{C}|^2 D)$, pri čemer

$$D = \max\{b - a \mid \exists P,\, Q \in \mathcal{C} \colon \mathit{mult}(P,\, Q) = a\,..\,b\}. \tag{5.8}$$

Računsko zahtevnost celotnega pretvornega postopka lahko torej podamo kot $O(|\mathcal{C}|^2 \max\{D,\, \log A\})$. Postopek je s stališča računske zahtevnosti občutljiv na razlike med zgornjimi in spodnjimi mejami pri omejenih multiplikativnostih in (v veliko manjši meri) na velikost spodnjih mej multiplikativnosti.

5.7 Sintaksna analiza in semantika modelov

Če graf G pripada jeziku gramatike GG, potem lahko izpeljavo grafa G v gramatiki GG pridobimo s pomočjo sintaksnega analizatorja. Pri podanem metamodelu MM lahko izpeljavo modelskega grafa $M \in L(MM)$ v gramatiki $GG(MM)$ pridobimo bodisi s pomočjo prilagojenega Rekers-Schürrovega sintaksnega analizatorja (poglavje 3) ali pa s pomočjo učinkovitega algoritma, opisanega v dokazu trditve 5.18 na strani 179. Izpeljava modelskega grafa v gramatiki $GG(MM)$ lahko služi kot osnova za semantično analizo ali transformacijo modelskega grafa. Če produkcije gramatike $GG(MM)$ opremimo s semantičnimi pravili, potem lahko »semantiko« (»pomen«) vhodnega modelskega grafa (tj. rezultat semantične analize ali transformacije modelskega grafa) pridobimo z uporabo semantičnih pravil na izpeljavi modelskega grafa v gramatiki $GG(MM)$.

Zamisel za povezavo med sintaksno in semantično analizo bomo prikazali na primeru. Denimo, da bi za podani modelski graf M, ki je skladen z metamodelom MM_{pub} (slika 5.1), želeli pridobiti niz, v katerem so našteti vsi revijalni prispevki in njihovi recenzenti. V primeru modelskega grafa na sliki 5.2 želeni izhodni niz izgleda takole:

```
JSa => Ra, Rb, Rc
```

Ta primer bo služil zgolj za ilustracijo podajanja in uporabe gramatičnih semantičnih pravil. Želeni izhodni niz bi lahko pridobili tudi na način, ki ne bi vključeval izdelave enakovredne grafne gramatike.

Da bi problem iz opisanega primera rešili s pomočjo semantičnih pravil, sledimo načrtu, ki je tipičen za kontekstno neodvisne besedilne gramatike [74]. Semantična pravila opredelimo kot na posamezne produkcije vezane izraze za vrednotenje *semantičnih prilastkov* (angl. *semantic attributes*), tj. spremenljivk, prirejenih posameznim vozliščem in povezavam v izpeljavi. Semantična pravila za produkcijo p imajo obliko $S_{i[1]}.A_{j[1]} := f(S_{i[2]}.A_{j[2]},\, \ldots,\, S_{i[k]}.A_{j[k]})$, kjer so $S_{i[1]},\, \ldots,\, S_{i[k]}$ oznake, ki nastopajo v produkciji p, $A_{j[1]},\, \ldots,\, A_{j[k]}$ pa so prilastki. Tako kot pri kontekstno neodvisnih besedilnih gramatikah tudi pri grafnih gramatikah razlikujemo med *podedovanimi* (angl. *inherited*) in *pridobljenimi* (angl. *synthesized*) prilastki. Podedovane prilastke

vrednotimo v smeri izpeljave, tj. v vrstnem redu, ki ga določa zaporedje vmesnih grafov pri izpeljavi modelskega grafa. Pridobljene prilastke vrednotimo v obratni smeri izpeljave. Če izpeljava vsebuje pretvorbo grafa G v graf G' z uporabo produkcije p, potem podedovane prilastke elementov grafa G' izračunamo na podlagi (podedovanih) prilastkov grafa G, pridobljene prilastke grafa G pa izračunamo na podlagi prilastkov grafa G'. (V primeru kontekstno neodvisnih gramatik se vrstni red vrednotenja prilastkov nanaša na *drevo*, ne na *zaporedje* izpeljave. V primeru kontekstno odvisnih gramatik pa dreves izpeljav v splošnem ni mogoče definirati, saj lahko leva stran produkcije vsebuje več kot en element.)

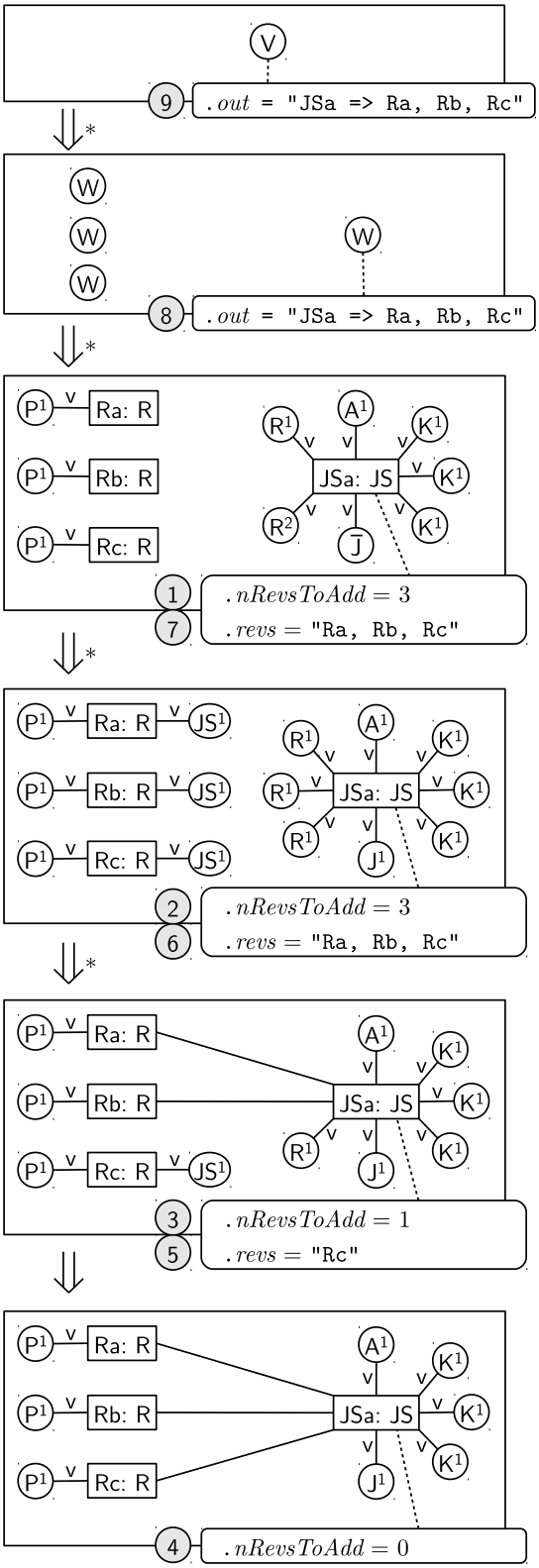
Vrnimo se k problemu naštevanja revijalnih prispevkov in njihovih recenzentov za podani modelski graf M. Predpostavimo, da vsakemu vozlišču grafa M pripada prilastek *id*, ki hrani ime objekta, predstavljenega z vozliščem. Na primer, JSa.*id* = `"JSa"`. Definirali bomo štiri semantične prilastke. Podedovani prilastek *nRevsToAdd*, ki v izpeljavi modelskega grafa pripada posameznim vozliščem z oznako JS, pove, koliko povezav med pripadajočim vozliščem JS in vozlišči R bo še treba vzpostaviti od trenutnega stanja izpeljave do njenega zaključka. Pridobljeni prilastek *revs* uporabljamo za zbiranje imen recenzentov za posamezne revijalne prispevke. Vozliščem z oznako V in W pripadata pridobljena prilastka *out*. Prilastek *out* za posamezno vozlišče W na koncu postopka semantičnega vrednotenja hrani izhodni niz za revijalni prispevek, ki ga predstavlja iz pripadajočega vozlišča W izvirajoče vozlišče JS. Pridobljeni prilastek *out* za vozlišče V pa na koncu semantičnega vrednotenja hrani izhodni niz, ki vsebuje želene podatke za vse revijalne prispevke.

Tabela 5.5 prikazuje semantična pravila za izbrane produkcije gramatike $GG(MM_{\text{pub}})$ (gl. tabelo 5.4). Da bi lahko razlikovali med različnimi vozlišči z isto oznako znotraj iste produkcije, uporabljamo indekse $\langle 1 \rangle$, $\langle 2 \rangle$ itd. Operator $\odot$ označuje stikanje nizov. Slika 5.7 ponazarja del izpeljave modelskega grafa s slike 5.2 na strani 156, pri čemer smo nekaj vmesnih korakov izpustili. V zaobljenih okvirjih so prikazane vrednosti semantičnih prilastkov za posamezna vozlišča. Številke levo od okvirjev označujejo vrstni red vrednotenja prilastkov. Podedovane prilastke ovrednotimo v prvem sprehodu po izpeljavi, pridobljene pa v drugem (obratnem) sprehodu. V zadnjem koraku izračunamo vrednost prilastka *out* za edino vozlišče začetnega grafa izpeljave (V). Ta vrednost je niz, ki smo ga želeli pridobiti.

5.8 Razširitve vhodnega formalizma

Pretvorno metodo smo zgradili na podlagi določenih predpostavk o vhodnih metamodelih. V tem podpoglavju bomo nakazali, kako bi morali metodo prilagoditi v primeru asimetričnih asociacij, označenih in vzporednih asociacij ter redefinicij asociacij.

Predpostavili smo, da je relacija **Assoc** simetrična (predpostavka 5.6), vendar pa lahko v metamodelih uporabljamo tudi asimetrične (usmerjene) asociacije. S takšnimi asociacijami predstavljamo usmerjene odnose med razredi. Na primer, asociacija $P \xrightarrow{1..1} Q$ predstavlja dejstvo, da mora biti vsak objekt razreda P povezan s po natanko enim objektom razreda Q preko usmerjene vezi $P \to Q$. Jezik metamodela $P \xrightarrow{1..1} Q$ vsebuje natanko dva povezana modelska grafa: Q in $P \to Q$.



Slika 5.7: *Del izpeljave modelskega grafa s slike 5.2 in koraki vrednotenja semantičnih prilastkov.*

Tabela 5.5: *Izbrane produkcije izhodne gramatike za metamodel MM_{pub} in pripadajoča semantična pravila.*

produkcija	semantična pravila
$\mathsf{V}_{\langle 1\rangle} ::= \mathsf{V}_{\langle 2\rangle};\ \mathsf{W}$	$\mathsf{V}_{\langle 1\rangle}.out := \mathsf{V}_{\langle 2\rangle}.out \odot \mathsf{W}.out$
$\mathsf{V} ::= \mathsf{W}$	$\mathsf{V}.out := \mathsf{W}.out$
$\mathsf{W} ::= \mathsf{JS} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^2, \mathsf{A}^1, \mathsf{K}^1, \mathsf{K}^1, \mathsf{K}^1, \overline{\mathsf{J}}\}$	$\mathsf{W}.out := \mathsf{JS}.id \odot \texttt{"=>"} \odot \mathsf{JS}.revs;$ $\mathsf{JS}.nRevsToAdd := 3$
$\mathsf{JS}_{\langle 1\rangle} \overset{\mathsf{v}}{-} \mathsf{R}^2 ::= \mathsf{JS}_{\langle 2\rangle} \overset{\mathsf{v}}{-} \{\mathsf{R}^1, \mathsf{R}^1\}$	$\mathsf{JS}_{\langle 2\rangle}.nRevsToAdd := \mathsf{JS}_{\langle 1\rangle}.nRevsToAdd;$ $\mathsf{JS}_{\langle 1\rangle}.revs := \mathsf{JS}_{\langle 2\rangle}.revs$
$\mathsf{JS}_{\langle 1\rangle} \overset{\mathsf{v}}{-} \mathsf{R}^2 ::= \mathsf{JS}_{\langle 2\rangle} \overset{\mathsf{v}}{-} \mathsf{R}^1$	$\mathsf{JS}_{\langle 2\rangle}.nRevsToAdd := \mathsf{JS}_{\langle 1\rangle}.nRevsToAdd - 1;$ $\mathsf{JS}_{\langle 1\rangle}.revs := \mathsf{JS}_{\langle 2\rangle}.revs$
$\mathsf{JS}_{\langle 1\rangle} \overset{\mathsf{v}}{-} \mathsf{R}^1;\ \mathsf{R}_{\langle 1\rangle} \overset{\mathsf{v}}{-} \mathsf{JS}^1 ::= \mathsf{JS}_{\langle 2\rangle} - \mathsf{R}_{\langle 2\rangle}$	$\mathsf{JS}_{\langle 2\rangle}.nRevsToAdd := \mathsf{JS}_{\langle 1\rangle}.nRevsToAdd - 1;$ $\mathsf{JS}_{\langle 1\rangle}.revs :=$ **če** $\mathsf{JS}_{\langle 2\rangle}.nRevsToAdd = 0,$ **potem** $\mathsf{R}_{\langle 2\rangle}.id,$ **sicer** $\mathsf{R}_{\langle 2\rangle}.id \odot \texttt{", "} \odot \mathsf{JS}_{\langle 2\rangle}.revs$

Vsi ostali grafi, ki pripadajo jeziku tega metamodela, so nepovezani; pridobimo jih s kopiranjem in disjunktnimi unijami grafov Q in $P \to Q$.

V pretvorni metodi bi lahko asociacije $P \xrightarrow{a\,..\,b} Q$ in $P \xrightarrow{a\,..\,*} Q$ obravnavali na podoben način kot asociacije $P \xrightarrow{0\,..\,1\ a\,..\,b} Q$ in $P \xrightarrow{0\,..\,1\ a\,..\,*} Q$, le povezave med vozlišči P in Q ter med vozlišči P in Q^d bi morali predstaviti kot usmerjene namesto kot neusmerjene. Na primer, metamodel $\mathsf{A} \xrightarrow{1\,..\,1} \mathsf{B}$ lahko pretvorimo v gramatiko s produkcijami $\lambda ::= \mathsf{V}$ in $\mathsf{V} ::= \mathsf{V}; \mathsf{W} \mid \mathsf{W}$ (družina 1), $\mathsf{W} ::= \mathsf{A} \to \mathsf{B}^1 \mid \mathsf{B}^1 \mid \mathsf{B} \overset{\mathsf{v}}{-} \mathsf{A}^1$ (družina 2) ter $\mathsf{A} \to \mathsf{B}^1;\ \mathsf{B} \overset{\mathsf{v}}{-} \mathsf{A}^1 ::= \mathsf{A} \to \mathsf{B}$ (družina 7).

Predpostavili smo, da so asociacije neoznačene in nevzporedne. Označene asociacije lahko enostavno pretvorimo v označene povezave. Vzporedne asociacije lahko obravnavamo, kot če bi bile sestavljene iz več ločenih asociacij. Recimo, da imamo metamodel z razredoma N in E (angl. *nodes* in *edges*, vozlišča in povezave) in dvema vzporednima asociacijama $\mathsf{N} - \mathsf{E}$ z oznakama s (angl. *source*, izvor) in t (angl. *target*, cilj). Obe asociaciji sta opremljeni z multiplikativnostjo $1\,..\,1$ v smeri $\mathsf{E} \to \mathsf{N}$ in z multiplikativnostjo $0\,..\,*$ v smeri $\mathsf{N} \to \mathsf{E}$. Opisani metamodel tako predstavlja razred usmerjenih grafov. Njemu enakovredna grafna gramatika sestoji iz produkcij $\lambda ::= \mathsf{V}$ in $\mathsf{V} ::= \mathsf{V}; \mathsf{W} \mid \mathsf{W}$ (družina 1), $\mathsf{W} ::= \mathsf{N}^1 \overset{\mathsf{vs}}{-} \mathsf{E} \overset{\mathsf{vt}}{-} \mathsf{N}^1 \mid \mathsf{N}$ (družina 2), $\mathsf{N} ::= \mathsf{N} \overset{\mathsf{vs}}{-} \mathsf{E}^1 \mid \mathsf{N} \overset{\mathsf{vt}}{-} \mathsf{E}^1$ (družina 3) ter $\mathsf{E} \overset{\mathsf{vs}}{-} \mathsf{N}^1;\ \mathsf{N} \overset{\mathsf{vs}}{-} \mathsf{E}^1 ::= \mathsf{E} \overset{\mathsf{s}}{-} \mathsf{N}$ in $\mathsf{E} \overset{\mathsf{vt}}{-} \mathsf{N}^1; \mathsf{N} \overset{\mathsf{vt}}{-} \mathsf{E}^1 ::= \mathsf{E} \overset{\mathsf{t}}{-} \mathsf{N}$ (družina 7), pri čemer sta oznaki vs in vt nekončni, oznaki s in t pa končni.

Predpostavili smo, da podrazredi ne morejo redefinirati asociacij (predpostavka 5.9). Vendar pa bi to omejitev lahko delno sprostili. Pri podanih razredih P in Q, kjer ne velja niti $P\ \mathbf{Inh}^+\ Q$ niti $Q\ \mathbf{Inh}^+\ P$, bi namreč asociacijo v smeri od razreda P proti razredu Q lahko redefinirali v podrazredih razreda P. Na primer, predstavljajmo si metamodel z razredi P (angl. *parent*, oče/mati), C (angl. *child*, otrok) in

F (angl. *friend*, prijatelj), pri čemer velja $\mathsf{C}\,\mathbf{Inh}\,\mathsf{P}$, $\mathbf{Assoc} = \{\mathsf{P}{-}\mathsf{F},\ \mathsf{C} \rightsquigarrow \mathsf{F}\}$, $mult(\mathsf{P},\ \mathsf{F}) = 1\,..\,1$, $mult(\mathsf{F},\ \mathsf{P}) = 2\,..\,2$ in $mult(\mathsf{C},\ \mathsf{F}) = 2\,..\,2$. Zapis $\mathsf{C} \rightsquigarrow \mathsf{F}$ označuje, da je asociacija $\mathsf{P}{-}\mathsf{F}$ v razredu C redefinirana samo v smeri $\mathsf{P} \to \mathsf{F}$. Metamodel predpisuje, da mora biti vsak objekt razreda P povezan z najmanj enim objektom razreda F, vsak objekt razreda F z najmanj dvema objektoma razredov iz množice $\{\mathsf{P},\ \mathsf{C}\}$, vsak objekt razreda C pa z natanko dvema objektoma F. Primer modelskega grafa, ki zadošča tem omejitvam, je $\mathsf{P}{-}\mathsf{F}{-}\mathsf{C}{-}\mathsf{F}{-}\mathsf{P}$. Gramatika, enakovredna opisanemu metamodelu, sestoji iz produkcij $\lambda ::= \mathsf{V}$ in $\mathsf{V} ::= \mathsf{V};\mathsf{W} \mid \mathsf{W}$ (družina 1), $\mathsf{W} ::= \mathsf{P}\overset{\mathsf{v}}{-}\mathsf{F}^1 \mid \mathsf{C}\overset{\mathsf{v}}{-}\{\mathsf{F}^1,\ \mathsf{F}^1\} \mid \mathsf{F}\overset{\mathsf{v}}{-}\{\mathsf{P}^1,\ \mathsf{P}^1\}$ (družina 2), $\mathsf{F}\overset{\mathsf{v}}{-}\mathsf{P}^1 ::= \mathsf{F}\overset{\mathsf{v}}{-}\mathsf{C}^1$ (družina 6) ter $\mathsf{P}\overset{\mathsf{v}}{-}\mathsf{F}^1;\mathsf{F}\overset{\mathsf{v}}{-}\mathsf{P}^1 ::= \mathsf{P}{-}\mathsf{F}$ in $\mathsf{C}\overset{\mathsf{v}}{-}\mathsf{F}^1;\mathsf{F}\overset{\mathsf{v}}{-}\mathsf{C}^1 ::= \mathsf{C}{-}\mathsf{F}$ (družina 7).

5.9 Zaključek

V tem poglavju smo predstavili izvirno metodo za pretvorbo metamodela v enakovredno grafno gramatiko. Pokazali smo, da izhodna gramatika opredeljuje enak jezik kot vhodni metamodel in da lahko pretvorbo opravimo v času in prostoru, ki je polinomsko odvisen od parametrov metamodela. Poleg tega smo si ogledali, kako lahko sintaksno odločljivost izhodne gramatike uporabimo za definicijo semantičnih pravil na podoben način kot pri kontekstno neodvisnih besedilnih gramatikah s prilastki.

V nasprotju z izhodnimi gramatikami, ki jih gradijo obstoječe pretvorne metode, izhodne gramatike našega pristopa pripadajo formalizmu, ki ga lahko obravnavamo kot čisto grafno razširitev standardnih kontekstno odvisnih besedilnih gramatik. Z uporabo takšnega ciljnega formalizma smo pokazali, da pri pretvorbi ne potrebujemo zahtevnejših gramatičnih elementov, kot so npr. pogoji uporabe in večkratna vozlišča.

Da bi poenostavili pretvorni postopek, smo uvedli določene omejitve glede vhodnih metamodelov. Kot smo nakazali v podpoglavju 5.8, lahko nekatere omejitve brez težav odpravimo. Nadaljnja možnost za izboljšavo naše metode je uvedba (omejene) podpore omejitvam OCL, formalizmu za podajanje zahtevnejših razmerij med razredi metamodela [99, 100]. Metoda Ehriga in sod. [30] je zmožna rokovanja z omejeno množico omejitev OCL; obravnava jih tako, da jih pretvori v pogoje uporabe znotraj izhodne grafne gramatike. V našem primeru bi lahko ubrali podoben pristop, saj je formalizem LGG možno brez posebnih težav razširiti s pogoji uporabe. Z raziskovalnega vidika pa se je zanimiveje vprašati, do kakšne mere bi lahko omejitve OCL podprli zgolj z uporabo standardnih elementov formalizma LGG.

Druga zanimiva možnost za nadaljnji razvoj metode bi bila poglobljena raziskava povezav med sintaksno in semantično analizo. Primer, ki smo ga pokazali v podpoglavju 5.7, je namenjen zgolj ilustraciji semantične analize, saj bi lahko podatke, ki smo jih iz podanega veljavnega modelskega grafa želeli izluščiti s pomočjo semantičnih pravil, lahko pridobili tudi kako drugače. Smiselno bi bilo poiskati primere uporabe, kjer se prednosti semantične analize pred *ad hoc* pristopi resnično izkažejo, podobno kot se, denimo, njene prednosti pokažejo v svetu prevajanja besedilnih programskih jezikov.

POGLAVJE

6

ZAKLJUČEK

Zaključno poglavje je sestavljeno iz dveh podpoglavij. V podpoglavju 6.1 se še enkrat ozremo na opravljeno delo, v podpoglavju 6.2 pa povzamemo zamisli za prihodnost, ki smo jih predstavili v zaključkih poglavij 3, 4 in 5.

6.1 Pregled doktorske disertacije

V doktorski disertaciji smo se ukvarjali s tremi različnimi problemi s področja grafnih gramatik:

Sintaksna analiza: Cilj problema sintaksne analize je zgraditi izpeljavo podanega grafa v podani grafni gramatiki, če ta obstaja. Posvetili smo se sintaksni analizi za kontekstno odvisne grafne gramatike. Izhajali smo iz pristopa, ki sta ga iznašla Rekers in Schürr [80], ter predlagali in uspešno preizkusili pet različnih izboljšav. Z eno od izboljšav smo odpravili zahtevo po povezanih desnih straneh produkcij vhodne grafne gramatike, z ostalimi štirimi pa smo povečali računsko učinkovitost sintaksnega analizatorja za mnoge primere grafnih gramatik. Izboljšave smo preizkusili na petih grafnih gramatikah s smiselnim pomenom in pri vseh dosegli bistveno povečanje učinkovitosti. Ponekod smo porabo časa zmanjšali celo z eksponentne na polinomsko. Z izboljšavami smo torej povečali uporabnost Rekers-Schürrovega sintaksnega analizatorja. Izboljšani sintaksni analizator uporabljamo v metodi za indukcijo grafnih gramatik.

Indukcija grafnih gramatik: Cilj tega problema je izgradnja grafne gramatike, ki smiselno posplošuje dano množico »pozitivnih« grafov, hkrati pa ne pokriva nobenega od podanih »negativnih« grafov. Indukcijski algoritem, ki smo ga predlagali in preizkusili, deluje po načelu hevrističnega preiskovanja prostora grafnih gramatik v smeri od specifičnega proti splošnemu. Algoritem prične z gramatiko, ki pokriva natanko množico pozitivnih grafov, nato pa po izvirnem postopku sistematično gradi posplošitve gramatik. S pomočjo sintaksnega analizatorja za vsako gramatiko preveri, ali pokriva katerega od negativnih vhodnih grafov; če to drži, jo izloči iz nadaljnje obravnave. Algoritem v svojem iskalnem prostoru išče najmanjšo gramatiko, ki je konsistentna z vhodnima množicama, saj lahko za majhne gramatike po načelu Ockhamove britve upamo, da bodo smiselno in karseda jedrnato posploševale vhodno množico. Indukcijsko metodo smo preizkusili na različnih grafnih množicah in v vseh primerih dobili smiselne rezultate, vendar pa bo za uporabnost metode na večjih vhodnih množicah grafov treba vložiti še nekaj truda.

Pretvorba metamodela v enakovredno grafno gramatiko: Predstavili smo izvirno metodo za pretvorbo metamodela v obliki razrednega diagrama UML v kontekstno odvisno grafno gramatiko, katere jezik vsebuje natanko tiste grafe, ki predstavljajo veljavne modele glede na vhodni metamodel. Izhodno grafno gramatiko lahko uporabimo za sistematično gradnjo veljavnih modelskih grafov, ki je metamodel kot deklarativni formalizem neposredno ne omogoča. Pokazali smo, da je izhodno gramatiko mogoče obogatiti s semantičnimi pravili in tako pridobiti možnost za semantično analizo veljavnih modelskih grafov.

Doktorska disertacija prinaša sledeče prispevke k znanosti:

- Izboljšava metode za sintaksno analizo pri kontekstno odvisnih grafnih gramatikah.

- Predlog nove metode za indukcijo grafnih gramatik. Kot izvirna prispevka lahko opredelimo oba načina posploševanja grafnih gramatik in uporabo sintaksnega analizatorja za preverjanje konsistentnosti posameznih grafnih gramatik z množico negativnih vhodnih grafov.

- Predlog nove metode za pretvorbo metamodela v enakovredno grafno gramatiko. Metoda uporablja drugačen ciljni formalizem in drugačen način pretvorbe kot sorodne metode, poleg tega pa smo predlagali pristop k semantični analizi modelskih grafov, ki temelji na definiciji in uporabi prilastkov, vezanih na posamezne elemente izhodne grafne gramatike.

Predstavljeni prispevki posegajo na različna širša področja računalniške znanosti. S prvim prispevkom smo posegli na področje algoritmov in podatkovnih struktur, z drugim na področje strojnega učenja, s tretjim pa na področje tehnologije programske opreme. Na ta način smo pokazali raznovrstno uporabnost grafnih gramatik.

6.2 Nadaljnje delo

Povzemimo nekatere predloge za nadaljnje delo, ki smo jih predstavili v zaključkih poglavij 3, 4 in 5. Pri sintaksnem analizatorju smo predlagali izboljšavo na temo preprečevanja odvečnega dela v primeru avtomorfizmov v vhodnih grafih. V povezavi z indukcijsko metodo bi se bilo smiselno osredotočiti na morebitno povečanje učinkovitosti sintaksnega analizatorja za grafne gramatike, ki pripadajo ciljnemu formalizmu indukcijskega algoritma, saj je razred gramatik iz ciljnega formalizma podrazred splošnega razreda kontekstno odvisnih grafnih gramatik.

Indukcijska metoda v danem trenutku zaradi precejšnje računske zahtevnosti žal še ni uporabna za reševanje klasifikacijskih problemov nad večjimi množicami grafov. Računsko zahtevnost indukcijske metode bi bilo mogoče povečati na različne načine, npr. z že omenjeno prilagoditvijo sintaksnega analizatorja na razred gramatik iz ciljnega formalizma ali pa z uporabo učinkovitejšega, četudi morda suboptimalnega pristopa k iskanju kandidatnih i-podgrafov, ki lahko nastopajo kot jedra bodočih produkcij. Tudi iskalni postopek bi bilo mogoče na različne načine izboljšati. Problem indukcije grafnih gramatik še zdaleč ni rešen, zato možnosti za nadaljnje delo ne bo zmanjkalo.

Pri metodi za pretvorbo metamodela v enakovredno grafno gramatiko v danem trenutku obravnavamo le osnovni nabor elementov metamodelov oz. razrednih diagramov UML. Pretvorno metodo bi lahko obogatili z obravnavo naprednejših metamodelskih elementov, npr. omejitev OCL. Pri semantični analizi modelskih grafov bi bilo smiselno poiskati konkretne primere uporabe, kjer se sistematičen pristop, prikazan v podpoglavju 5.7, obnese bolje od različnih *ad hoc* pristopov.

LITERATURA

[1] A.V. Aho, M.S. Lam, R. Sethi in J.D. Ullman. *Compilers: Principles, Techniques, and Tools*, 2. izdaja. Addison-Wesley, Boston, Massachusetts, ZDA, 2006.

[2] M. Alanen in I. Porres. A relation between context-free grammars and Meta Object Facility metamodels. Tehnično poročilo 606, Turku Centre for Computer Science, marec 2003.

[3] N. Aschenbrenner in L. Geiger. Transforming scene graphs using triple graph grammars — a practice report. V: A. Schürr, M. Nagl in A. Zündorf, uredniki, *3rd International Symposium on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007)*, Kassel, Nemčija, oktober 2007, zbirka *Lecture Notes in Computer Science*, zvezek 5088, strani 32–43. Springer, 2008.

[4] K. Ates, J. P. Kukluk, L. B. Holder, D. J. Cook in K. Zhang. Graph grammar induction on structural data for visual programming. V: *18th International Conference on Tools with Artificial Intelligence (ICTAI 2006)*, Washington, DC, ZDA, november 2006, strani 232–242. IEEE Computer Society, 2006.

[5] L. Babai in E. M. Luks. Canonical labeling of graphs. V: D. S. Johnson, R. Fagin, M. L. Fredman, D. Harel, R. M. Karp, N. A. Lynch, C. H. Papadimitriou, R. L. Rivest, W. L. Ruzzo in J. I. Seiferas, uredniki, *15th Annual ACM Symposium on Theory of Computing*, Boston, Massachusetts, ZDA, april 1983, strani 171–183. ACM, 1983.

[6] Z. Balogh in D. Varró. Model transformation by example using inductive logic programming. *Software and Systems Modeling*, 8(3):347–364, 2009.

[7] R. Bardohl, H. Ehrig, J. de Lara in G. Taentzer. Integrating meta-modelling aspects with graph transformation for efficient visual language definition and

model manipulation. V: *7th International Conference on Fundamental Approaches to Software Engineering*, Barcelona, Španija, marec–april 2004, zbirka *Lecture Notes in Computer Science*, zvezek 2984, strani 214–228. Springer, 2004.

[8] G. Benkö, C. Flamm in P. F. Stadler. A graph-based toy model of chemistry. *Journal of Chemical Information and Computer Sciences*, 43(4):1085–1093, 2003.

[9] D. Blostein, H. Fahmy in A. Grbavec. Practical use of graph rewriting. Tehnično poročilo 95-373, Queen's University, Kingston, Ontario, Kanada, januar 1995.

[10] D. Blostein in A. Schürr. Computing with graphs and graph transformations. *Software – Practice and Experience*, 29(3):197–217, 1999.

[11] P. Bottoni, G. Taentzer in A. Schürr. Efficient parsing of visual languages based on critical pair analysis and contextual layered graph transformation. V: *16th IEEE International Symposium on Visual Languages (VL 2000)*, Seattle, Washington, ZDA, september 2000, strani 59–60. IEEE Computer Society, 2000.

[12] R. Brijder in H. Blockeel. On the inference of non-confluent NLC graph grammars. *Journal of Logic and Computation*, 23(4):799–814, 2013.

[13] T. Buchmann, A. Dotor, S. Uhrig in B. Westfechtel. Model-driven software development with graph transformations: A comparative case study. V: A. Schürr, M. Nagl in A. Zündorf, uredniki, *3rd International Symposium on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2007)*, Kassel, Nemčija, oktober 2007, zbirka *Lecture Notes in Computer Science*, zvezek 5088, strani 345–360. Springer, 2008.

[14] F. Budinsky, D. Steinberg in R. Ellersick. *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley Professional, 2003.

[15] H. Bunke in B. Haller. A parser for context free plex grammars. V: M. Nagl, urednik, *15th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1989)*, Castel Rolduc, Nizozemska, junij 1989, zbirka *Lecture Notes in Computer Science*, zvezek 411, strani 136–150. Springer, 1990.

[16] F. Büttner in M. Gogolla. Realizing UML metamodel transformations with AGG. *Electronic Notes in Theoretical Computer Science*, 109:31–42, 2004.

[17] K. Chen, J. Sztipanovits in S. Neema. Compositional specification of behavioral semantics. V: R. Lauwereins, J. Madsen, urednika, *Design, Automation, and Test in Europe (DATE 2007)*, Nica, Francija, april 2007, strani 906–911. ACM, 2007.

[18] D. J. Cook in L. B. Holder. *Mining Graph Data*. John Wiley & Sons, 2006.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest in C. Stein. *Introduction to Algorithms*, 3. izdaja. MIT Press, 2009.

[20] C. Crimi, A. Guercio, G. Nota, G. Pacini, G. Tortora in M. Tucci. Relation grammars and their application to multi-dimensional languages. *Visual Languages and Computing*, 2(4):333–346, 1991.

[21] M. Črepinšek, M. Mernik, F. Javed, B.R. Bryant in A.P. Sprague. Extracting grammar from programs: evolutionary approach. *ACM SIGPLAN Notices*, 40(4):39–46, 2005.

[22] J. de Lara, H. Vangheluwe in M. Alfonseca. Metamodelling and graph grammars for multi-paradigm modelling in AToM^3. *Software and Systems Modeling*, 3(3):194–209, 2004.

[23] T.T. Dinh-Trong, S. Ghosh in R.B. France. A systematic approach to generate inputs to test UML design models. V: *17th International Symposium on Software Reliability Engineering (ISSRE 2006)*, Raleigh, Severna Karolina, ZDA, november 2006, strani 95–104. IEEE Computer Society, 2006.

[24] F. Drewes. Recognising k-connected hypergraphs in cubic time. *Theoretical Computer Science*, 109(1&2):83–122, 1993.

[25] F. Drewes, B. Hoffmann, D. Janssens in M. Minas. Adaptive star grammars and their languages. *Theoretical Computer Science*, 411(34-36):3090–3109, 2010.

[26] A. Dubey, P. Jalote in S.K. Aggarwal. Learning context-free grammar rules from a set of programs. *IET Software*, 2(3):223–240, 2008.

[27] J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

[28] H. Ehrig., G. Engels in G. Rozenberg, uredniki. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 2 (Applications, Languages, and Tools)*. World Scientific, 1999.

[29] H. Ehrig, H.J. Kreowski, U. Montanari in G. Rozenberg, uredniki. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 3 (Concurrency, Parallelism, and Distribution)*. World Scientific, 1999.

[30] K. Ehrig, J.M. Küster in G. Taentzer. Generating instance models from meta models. *Software and System Modeling*, 8(4):479–500, 2009.

[31] J. Engelfriet in L. Heyker. The string generating power of context-free hypergraph grammars. *Journal of Computer and System Sciences*, 43(2):328–360, 1991.

[32] G. Engels, J.H. Hausmann, R. Heckel in S. Sauer. Dynamic meta modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. V: A. Evans, S. Kent in B. Selic, uredniki, *3rd International Conference*

on the Unified Modeling Language: Advancing the Standard (UML 2000), York, Združeno kraljestvo, oktober 2000, zbirka *Lecture Notes in Computer Science*, zvezek 1939, strani 323–337. Springer, 2000.

[33] F. Ferrucci, G. Pacini, G. Satta, M.I. Sessa, G. Tortora, M. Tucci in G. Vitiello. Symbol-relation grammars: A formalism for graphical languages. *Information and Computation*, 131(1):1–46, 1996.

[34] M. Flasiński in S. Myśliński. On the use of graph parsing for recognition of isolated hand postures of Polish Sign Language. *Pattern Recognition*, 43(6):2249–2264, 2010.

[35] M. Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 3. izdaja. Addison-Wesley Longman Publishing Co., Boston, Massachusetts, ZDA, 2003.

[36] L. Fürst, M. Mernik in V. Mahnič. Converting metamodels to graph grammars: Doing without advanced graph grammar features. *Journal of Software and Systems Modeling*. (Članek sprejet v objavo.)

[37] L. Fürst, M. Mernik in V. Mahnič. Improving the graph grammar parser of Rekers and Schürr. *IET Software*, 5(2):246–261, 2011.

[38] L. Fürst, M. Mernik in V. Mahnič. Graph grammar induction as a parser-controlled heuristic search process. V: A. Schürr, D. Varró in G. Varró, uredniki, *4th International Symposium on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2011)*, Budimpešta, Madžarska, oktober 2011, zbirka *Lecture Notes in Computer Science*, zvezek 7233, strani 121–136. Springer, 2012.

[39] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema in J. Sprinkle. Domain-specific modeling. V: P.A. Fishwick, urednik, *Handbook of Dynamic System Modeling*. CRC Press, Florida, ZDA, 2007.

[40] A. Habel. *Hyperedge Replacement: Grammars and Languages*, zbirka *Lecture Notes in Computer Science*, zvezek 643, Springer, 1992.

[41] D. Harel in B. Rumpe. Meaningful modeling: what's the semantics of "semantics"? *Computer*, 37(10):64–72, 2004.

[42] F. Hermann, H. Ehrig in G. Taentzer. A typed attributed graph grammar with inheritance for the abstract syntax of UML class and sequence diagrams. *Electronic Notes in Theoretical Computer Science*, 211:261–269, 2008.

[43] B. Hoffmann in M. Minas. Generating instance graphs from class diagrams with adaptive star grammars. V: *3rd International Workshop on Graph Computation Models (GCM 2010)*, zbirka *Electronic Communications of the European Association of Software Science and Technology*, zvezek 39, 2011.

[44] L. B. Holder, D. J. Cook in S. Djoko. Substucture discovery in the SUBDUE system. V: U. M. Fayyad in R. Uthurusamy, urednika, *Knowledge Discovery in Databases Workshop*, Seattle, Washington, ZDA, julij 1994, strani 169–180. AAAI Press, 1994.

[45] J.E. Hopcroft, R. Motwani in J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, 3. izdaja. Addison-Wesley, Boston, Massachusetts, ZDA, 2006.

[46] F. Javed, M. Mernik, B.R. Bryant in A. Sprague. An unsupervised incremental learning algorithm for domain-specific language development. *Applied Artificial Intelligence*, 22(7-8):707–729, 2008.

[47] F. Javed, M. Mernik, J. Gray in B.R. Bryant. MARS: A metamodel recovery system using grammar inference. *Information and Software Technology*, 50(9–10):948–968, 2008.

[48] E. Jeltsch in H.-J. Kreowski. Grammatical inference based on hyperedge replacement. V: H. Ehrig, H.-J. Kreowski in G. Rozenberg, uredniki, *4th International Workshop on Graph-Grammars and Their Application to Computer Science*, Bremen, Nemčija, marec 1990, zbirka *Lecture Notes in Computer Science*, zvezek 532. strani 461–474. Springer, 1991.

[49] I. Jonyer, L. B. Holder in D. J. Cook. MDL-based context-free graph grammar induction and applications. *International Journal of Artificial Intelligence Tools*, 13(1):65–79, 2004.

[50] M. Kaul. Parsing of graphs in linear time. V: H. Ehrig, M. Nagl in G. Rozenberg, uredniki, *2nd International Workshop on Graph Grammars and Their Application to Computer Science*, Osnabrück, Nemčija, oktober 1982, zbirka *Lecture Notes in Computer Science*, zvezek 153, strani 206–218. Springer, 1983.

[51] S. Kelly in J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full-Code Generation*. John Wiley and Sons, 2008.

[52] R. Klempien-Hinrichs in C. von Totth. Generation of Celtic key patterns with tree-based collage grammars. *Electronic Communication of the European Association of Software Science and Technology*, 26, 2010.

[53] J. Kong, K. Zhang in X. Zeng. Spatial graph grammars for graphical user interfaces. *ACM Transactions on Computer-Human Interaction*, 13(2):268–307, 2006.

[54] A. Königs in A. Schürr. Tool integration with Triple Graph Grammars – a survey. *Electronic Notes in Theoretical Computer Science*, 148(1):113–150, 2006.

[55] J. Kukluk. *Inference of Node and Edge Replacement Graph Grammars*. Doktorska disertacija, University of Texas at Arlington, 2007.

[56] J. Kukluk, L. Holder in D. Cook. Inferring graph grammars by detecting overlap in frequent subgraphs. *International Journal of Applied Mathematics and Computer Science*, 18(2):241–250, 2008.

[57] M. Kuramochi in G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1038–1051, 2004.

[58] M. Kuramochi in G. Karypis. GREW – a scalable frequent subgraph discovery algorithm. V: *4th International Conference on Data Mining (ICDM 2004)*, Brighton, Združeno kraljestvo, november 2004, strani 439–442. IEEE Computer Society, 2004.

[59] M. Kuramochi in G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.

[60] C. Lautemann. The complexity of graph languages generated by hyperedge replacement. *Acta Informatica*, 27(5):399–421, 1989.

[61] S. Lavirotte in L. Pottier. Optical formula recognition. V: *4th International Conference Document Analysis and Recognition (ICDAR '97)*, Ulm, Nemčija, avgust 1997, strani 357–361. IEEE Computer Society, 1997.

[62] D.B. Lenat. *AM: An artificial intelligence approach to discovery in mathematics as heuristic search*. Doktorska disertacija, Stanford University, 1976.

[63] L. Lin, T. Wu, J. Porway in Z. Xu. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition*, 42(7):1297–1307, 2009.

[64] T. Mens in P. van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, 2006.

[65] T. Mens, P. van Gorp, D. Varró in G. Karsai. Applying a model transformation taxonomy to graph transformation technology. *Electronic Notes in Theoretical Computer Science*, 152:143–159, 2006.

[66] M. Mernik, J. Heering in A.M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, 2005.

[67] M. Minas. Parsing of adaptive star grammars. V: G. Karsai in G. Taentzer, urednika, *2nd International Workshop on Graph and Model Transformation (GraMoT 2006)*, Brighton, Združeno kraljestvo, september 2006, zbirka *Electronic Communications of the European Association of Software Science and Technology*, zvezek 4, 2006.

[68] M. Minas in O. Köth. Generating diagram editors with DiaGen. V: M. Nagl, A. Schürr in M. Münch, uredniki, *International Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE'99)*, Kerkrade, Nizozemska, september 1999, zbirka *Lecture Notes in Computer Science*, zvezek 1779, strani 433–440. Springer, 2000.

[69] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[70] T. M. Mitchell. *Machine Learning*, 1. izdaja. McGraw-Hill, New York, ZDA, 1997.

[71] M. Nagl. Formal languages of labelled graphs. *Computing*, 16(1-2):113–137, 1976.

[72] K. Nakamura in M. Matsumoto. Incremental learning of context free grammars based on bottom-up parsing and search. *Pattern Recognition*, 38(9):1384–1392, 2005.

[73] T. Oates, S. Doshi in F. Huang. Estimating maximum likelihood parameters for stochastic context-free graph grammars. V: T. Horváth, urednik, *13th International Conference on Inductive Logic Programming*, Szeged, Madžarska, september–oktober 2003, zbirka *Lecture Notes in Computer Science*, zvezek 2835, strani 281–298. Springer, 2003.

[74] J. Paakki. Attribute grammar paradigms — a high-level methodology in language implementation. *ACM Computing Surveys*, 27(2):196–255, 1995.

[75] R. Parekh in V. Honavar. Grammar inference, automata induction, and language acquisition. V: R. Dale, H. L. Somers in H. Moisl, uredniki, *Handbook of Natural Language Processing*, strani 727–764. Marcel Dekker, New York, ZDA, 2000.

[76] J.L. Pfaltz in A. Rosenfeld. Web grammars. V: D.E. Walker in L.M. Norton, urednika, *1st International Joint Conference on Artificial Intelligence (IJCAI)*, Washington, DC, ZDA, maj 1969, strani 609–620. William Kaufmann, 1969.

[77] M. J. Plasmeijer in M. C. J. D. van Eekelen. Term graph rewriting and mobile expressions in functional languages. V: M. Nagl, A. Schürr in M. Münch, uredniki, *International Workshop on Applications of Graph Transformations with Industrial Relevance (AGTIVE'99)*, Kerkrade, Nizozemska, september 1999, zbirka *Lecture Notes in Computer Science*, zvezek 1779, strani 1–13. Springer, 2000.

[78] V. Rafe, A. T. Rahmani, L. Baresi in P. Spoletini. Towards automated verification of layered graph transformation specifications. *IET Software*, 3(4):276–291, 2009.

[79] J. Rekers in A. Schürr. A parsing algorithm for context-sensitive graph grammars. Tehnično poročilo 95-05, Leiden University, 1995.

[80] J. Rekers in A. Schürr. Defining and parsing visual languages with Layered Graph Grammars. *Journal of Visual Languages and Computing*, 8(1):27–55, 1997.

[81] G. Rozenberg, urednik. *Handbook of Graph Grammars and Computing by Graph Transformation: Volume 1 (Foundations)*. World Scientific, 1997.

[82] G. Rozenberg in E. Welzl. Boundary NLC graph grammars – basic definitions, normal forms, and complexity. *Information and Control*, 69(1–3):136–167, 1986.

[83] D. Di Ruscio, F. Jouault, I. Kurtev, J. Bézivin in A. Pierantonio. Extending AMMA for supporting dynamic semantics specifications of DSLs. Tehnično poročilo 06.02, INRIA/LINA, april 2006.

[84] D.A. Sadilek in G. Wachsmuth. Using grammarware languages to define operational semantics of modelled languages. V: M. Oriol in B. Meyer, urednika, *47th International Conference on Objects, Models, Components, and Patterns (TOOLS-EUROPE'09)*, Zürich, Švica, junij–julij 2009, zbirka *Lecture Notes in Business Information Processing*, zvezek 33, strani 348–356. Springer, 2009.

[85] F. Schöler in V. Steinhage. Towards an automated 3D reconstruction of plant architecture. V: A. Schürr, D. Varró in G. Varró, uredniki, *4th International Symposium on Applications of Graph Transformations with Industrial Relevance (AGTIVE 2011)*, Budimpešta, Madžarska, oktober 2011, zbirka *Lecture Notes in Computer Science*, zvezek 7233, strani 51–64. Springer, 2012.

[86] A. Schürr. Specification of graph translators with Triple Graph Grammars. V: E.W. Mayr, G. Schmidt in G. Tinhofer, uredniki, *20th International Workshop on Graph-Theoretic Concepts in Computer Science*, Herrsching, Nemčija, junij 1994, zbirka *Lecture Notes in Computer Science (WG'94)*, zvezek 903, strani 151–163. Springer, 1995.

[87] S. Seifert in I. Fischer. Parsing string generating hypergraph grammars. V: H. Ehrig, G. Engels, F. Parisi-Presicce in G. Rozenberg, uredniki, *2nd International Conference on Graph Transformations (ICGT 2004)*, Rim, Italija, september–oktober 2004, zbirka *Lecture Notes in Computer Science*, zvezek 3256, strani 352–367. Springer, 2004.

[88] S. Sen, B. Baudry in J.-M. Mottu. On combining multi-formalism knowledge to select models for model transformation testing. V: *1st International Conference on Software Testing, Verification, and Validation (ICST 2008)*, Lillehammer, Norveška, april 2008, strani 328–337. IEEE Computer Society, 2008.

[89] J. Sprinkle, M. Mernik, J.-P. Tolvanen in D. Spinellis. Guest editors' introduction: What kinds of nails need a domain-specific hammer? *IEEE Software*, 26(4):15–18, 2009.

[90] A. Stevenson in J. R. Cordy. Grammatical inference in software engineering: An overview of the state of the art. V: K. Czarnecki in G. Hedin, urednika, *International Conference on Software Language Engineering (SLE 2012)*, Dresden, Nemčija, september 2012, zbirka *Lecture Notes in Computer Science*, zvezek 7745, strani 204–223. Springer, 2013.

[91] G. Taentzer. Instance generation from type graphs with arbitrary multiplicities. V: *11th International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2012)*, zbirka *Electronic Communications of the European Association of Software Science and Technology*, zvezek 47, 2012.

[92] M. Tucci, G. Vitiello in G. Costagliola. Parsing nonlinear languages. *IEEE Trans. on Software Engineering*, 20(9):720–739, 1994.

[93] M.S. Tveit. Specification of graphical representations – using hypergraphs or meta-models? V: *Norsk Informatikkonferanse*, strani 39–50. Tapir Akademisk Forlag, 2008.

[94] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[95] K. VanLehn in W. Ball. A version space approach to learning context-free grammars. *Machine Learning*, 2(1):39–74, 1987.

[96] J. Veber. Koncepcija delovanja mislečega stroja. *Organizacija in kadri*, 3(5):367–378, 1974.

[97] J.T. Vermeulen. Viability of a parsing algorithm for context-sensitive graph grammars. Magistrska naloga, Leiden University, 1996.

[98] W. Vogler. Recognizing edge replacement graph languages in cubic time. V: H. Ehrig, H.-J. Kreowski in G. Rozenberg, uredniki, *4th International Workshop on Graph-Grammars and Their Application to Computer Science*, Bremen, Nemčija, marec 1990, zbirka *Lecture Notes in Computer Science*, zvezek 532, strani 676–687. Springer, 1991.

[99] J. Warmer in A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*, 2. izdaja. Addison-Wesley Longman Publishing Co., Boston, Massachusetts, ZDA, 2003.

[100] J. Winkelmann, G. Taentzer, K. Ehrig in J.M. Küster. Translation of restricted OCL constraints into graph constraints for generating meta model instances by graph grammars. *Electronic Notes in Theoretical Computer Science*, 211:159–170, 2008.

[101] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

[102] L. Yu, R.B. France in I. Ray. Scenario-based static analysis of UML class models. V: *Model-Driven Engineering Languages and Systems (MoDELS 2008)*, Toulouse, Francija, september–oktober 2008, zbirka *Lecture Notes in Computer Science*, zvezek 5301, strani 234–248. Springer, 2008.

[103] D.-Q. Zhang, K. Zhang in J. Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Journal*, 44(3):186–200, 2001.