Fakulta informačních technologií, Vysoké učení technické v Brně

Využití řízených gramatik v programovacích jazycích

Projekt do předmětu Teorie programovacích jazyků
doktorského studijního programu

15. ledna 2006

Zbyněk Křivka

Abstrakt

Následující text je součástí projektu do předmětu Teorie programovacích jazyků doktorského studijního programu. Zaměřuje se na využití různých typů řízených gramatik, známých z teorie formálních jazyků pro potřeby popisu programovacích jazyků. Práce se snaží vytvořit pro čtenáře náhled na tuto problematiku a nastínit případná řešení některých probíraných aspektů.

Klíčová slova

Chomského hierarchie, řízená gramatika, programovaná gramatika, maticová gramatika, konečný index, syntaktická analýza, sémantická analýza, překladač, sémantické akce, atributová gramatika, kontextový jazyk, programovací jazyk, deklarace proměnné, bloky

Obsah

1 Úvod

Jednou z oblastí, kde se teoretičtí informatici snaží nacházet praktické aplikace teorie formálních jazyků, je oblast překladačů a popisování programovacích jazyků. V případě popisu syntaxe a aplikace bezkontextových gramatik či obecně modelů bezkontextových jazyků se jedná o velmi dobře prozkoumanou a zdokumentovanou oblast. Pokud se ovšem budeme zabývat využitím řízených gramatik a popisu sémantických vlastností jazyků, narazíme na relativně malé množství dostupné literatury ke studiu.

Tato práce si v žádném případě neklade za cíl být všeobsahujícím přehledem této tématiky, ale jde spíše o úvod do této oblasti a nastínění dalšího možného studia či výzkumu.

Od čtenáře se očekává základní znalost teorie formálních jazyků a přehled v oblasti používání a tvorby překladačů programovacích jazyků. I přesto budou základní a nejdůležitější pojmy zopakovány v následující úvodní podkapitole.

1.1 Základní pojmy

V rámci úvodu k této práci bych rád stručně definoval a popsal základní pojmy dále používané v textu, případně upozornil na některá zjednodušení, která si dovolím učinit.

Většina definic je upravována pro potřeby tohoto textu, věty a teorémy jsou většinou uváděny bez důkazu, který lze nalézt v literatuře, ze které jsem čerpal.

Definice 1. Gramatika, pravidla

Obecná *gramatika* je čtveřice $G = (V, T, P, S)$, kde V je konečná *totální abeceda* (všechny symboly), T je konečná *abeceda terminálů* (vstupních symbolů), P je *konečná relace*, $S \in (V - T)$ je *startující axiom*. Podle definice konečné relace P gramatiky G rozlišujeme mnoho typů gramatik (např. různé gramatiky Chomského hierarchie či řízené gramatiky atp.). Prvky relace P nazýváme *přepisovací pravidla* či stručněji *pravidla*.

Každé pravidlo má unikátní *návěští* $p_1, p_2, \ldots, p_{|P|}$. Dále si označme množinu $N = V - T$ jako množinu *neterminálů*. *Lab*(*P*) případně *lab*(*p*) budeme značit množinu návěští pravidel z P, respektive návěští pravidla p.

Definice 2. Relace přímé derivace, jazyk

Nechť G je libovolná gramatika. Dva řetězce x, y nad abecedou V jsou v binární *relaci přímé derivace* D, pokud aplikací nějakého pravidla z G na x dostaneme řetězec y. Způsob aplikace je dán konkrétním typem gramatiky. Pro čitelnější zápis pro $(x, y) \in D$ píšeme $x \Rightarrow y$. Tranzitivní a reflexivně-tranzitivní uzávěr relace přímé derivace značíme $\Rightarrow^+$ a $\Rightarrow^*$. Pro odlišení různých derivačních kroků doplňujeme znak $\Rightarrow$ o různé levé a pravé dolní indexy (např. $\Rightarrow_{ac}$). *Jazyk* daný gramatikou G si pak definujme jako množinu řetězců nad abecedou terminálů L(G), pro kterou platí $L(G) = \{ w \in T^* \mid S \Rightarrow^* w \}$. Občas může být pro speciální modely definován jejich jazyk jiným způsobem a v takovém případě na to bude čtenář upozorněn.

Definice 3. Programovací jazyk

Programovací jazyk je jazyk zadaný svou syntaxí a sémantikou a slouží pro psaní zdrojových textů programů pro počítač. *Správný zdrojový text* (*program*) je věta tohoto jazyka.

Definice 4. Syntaktická a sémantická analýza

Syntaktická analýza zdrojového textu programu je zkontrolování syntaktické správnosti věty programovacího jazyka. Sémantická analýza kontroluje sémantickou správnost zdrojového textu programu. Pokud zdrojový text splňuje obě kontroly, je větou daného programovacího jazyka a obecný překladač může přistoupit k překladu do jazyka procesoru.

Syntaktický popis programovacího jazyka se nejčastěji provádí pomocí formální gramatiky či méně formální (E)BNF; sémantika bývá popisována strukturovaným přirozeným jazykem. Implementace syntaktické analýzy je lehce automatizovatelná na rozdíl od sémantické kontroly, která zachycuje mnohem náročnější požadavky na zdrojový text (často tzv. kontextové vlastnosti).

Definice 5. Překlad

Překlad je binární relace T: $V_I^* \times V_O^*$. Slovně řečeno, překlad je transformace T řetězce nad vstupní abecedou V_I na řetězec výstupní abecedy V_O. V praxi je často T funkcí.

Definice 6. Značení tříd jazyků

Třídy jazyků generované nebo přijímané různými formálními modely (různými typy gramatik či automatů) budeme značit tučným velkým písmenem L s parametry onoho formálního modelu, $\boldsymbol{L}$([*Typ*,] *Základ* [– ε] [, *Kontrola_na_výskyt*]), kde

Typ ∈ {*M*, *P*, *RC*, …} a lze vynechat,

Základ ∈ {*FIN*, *LIN*, *REG*, *CF*, *CS*, *RE*} popisuje model, na kterém stavíme (nejčastěji vypovídá o základním tvaru gramatických pravidel, ze kterých vycházíme). V případě dodatku „– ε“ neuvažujeme v modelu tzv. vymazávací pravidla, tj. pravidla s prázdným řetězcem na pravé straně.

Kontrola_na_výskyt ∈ {*ac*} a lze vynechat.

U jazyků s konečným indexem *n* toto bude řečeno dolním pravým indexem u $\boldsymbol{L}$, tj. $\boldsymbol{L}_n$(…).

Například $\boldsymbol{L}_2$(*M*, *CF* – ε, *ac*) označuje třídu jazyků generovaných maticovými gramatikami s kontrolou na výskyt bez vymazávacích pravidel konečného indexu 2.

Definice 7. Chomského klasifikace jazyků a gramatik

Relace přímé derivace D v gramatikách Chomského klasifikace je definována jako binární relace nad V^*, kdy x přímo derivuje y (neboli $(x, y) \in D$) právě, když $x = uaz$, $y = ubz$ a bylo použito pravidlo r_i: $a \to b$, kde $x, y \in V^*$. Zkráceně píšeme $x \Rightarrow y$ [r_i: $a \to b$].

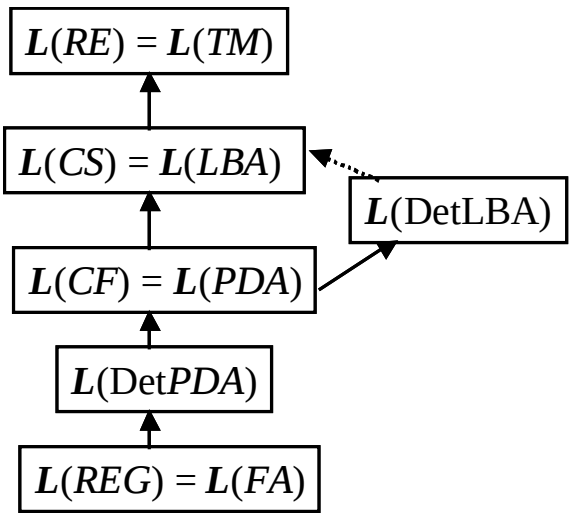
Chomského klasifikace gramatik obsahuje čtyři typy gramatik, které se liší množnými tvary jejich pravidel následovně (v závorce uvádím anglický název gramatiky a označení třídy jazyků generovaných daným typem gramatiky):

Neomezené gramatiky (angl. *unbounded*, $\boldsymbol{L}(RE)$): $a \rightarrow b$, kde $a, b \in V^*$.
Kontextové gramatiky (angl. *context-sensitive*, $\boldsymbol{L}(CS)$): $a \rightarrow b$, kde $a \in V^+$, $b \in V^*$.
Bezkontextové gramatiky (angl. *context-free*, $\boldsymbol{L}(CF)$): $a \rightarrow b$, kde $a \in N$, $b \in V^*$.
Lineární gramatiky (angl. linear, $\boldsymbol{L}(LIN)$: $a \rightarrow b$, kde $\alpha \in N$, $b \in T^*NT^* \cup T$.
Regulární gramatiky (angl. *regular*, $\boldsymbol{L}(REG)$): $a \rightarrow b$, kde $\alpha \in N$, $b \in TN \cup T$.

Daným třídám, jež jsou popsány gramatikami, odpovídají i třídy jazyků definované automaty (pouze uvedu jejich označení):

Turingův stroj (TM), Lineárně omezený automat (LBA), Zásobníkový automat (PDA) a konečný automat (FA). V případě deterministické varianty automatu doplňuji předponu „Det“.

Pravidlo $a \rightarrow b$ budeme nazývat *vymazávací pravidlo* (angl. *erasing production*), pokud $b = \varepsilon$.



Obr. 1: *Chomského hierarchie tříd jazyků pro gramatiky a automaty*

Poznámka k obrázku tříd jazyků: Šipka spojuje vždy třídu a nadtřídu a vyjadřuje tak vztah vlastní inkluze mezi třídami jazyků (Např. spojuje-li šipka třídu A s třídou B a šipka je u třídy B, tak to znamená $A \subset B$). V případě čárkované šipky se jedná o pouhou inkluzi (tj. je otevřeným problémem, zda jde o vlastní inkluzi nebo o ekvivalenci tříd).

Ještě doplňme, že $\boldsymbol{L}(REG) \subset \boldsymbol{L}(LIN) \subset \boldsymbol{L}(CF)$.

1.2 Překlad

Nyní se podívejme na techniku nejčastěji používanou při analýze programovacích jazyků a překladu zdrojových textů programů. Vše je popsáno především pro zopakování základního současného přístupu k překladu programovacích jazyků (i těch s kontextovými

vlastnostmi). Po čtenáři se vyžaduje alespoň zevrubná znalost metod syntaktické analýzy shora dolů a zdola nahoru.

Nejdříve se zaměříme na definici několika formálních pojmů (viz [5]), ze kterých bude částečně vycházet následující méně formální a více praktická část této kapitoly (viz kapitola 5 z [1]).

Rozšířením bezkontextové gramatiky o popis nejen syntaktické, ale i sémantické části vznikly tzv. atributové gramatiky, které jsou v hojné míře využívány jako formalismus při konstrukci překladačů programovacích jazyků. Konkrétněji bylo každé pravidlo doplněno o posloupnost sémantických akcí. Tyto akce jsou popisovány v pseudo-programovacím jazyce, jehož mocnost se většinou pohybuje mezi třídou kontextových jazyků a třídou jazyků přijímaných Turingovými stroji.

Definice 8. Atribut, atributový symbol

Atribut je veličina s množinou hodnot (potenciálně i nekonečnou) neboli tzv. oborem hodnot. Atribut lze přirovnat k proměnné určitého typu (např. celé číslo, výčtový typ).

Atributovaný symbol je symbol nějaké abecedy s konečnou množinou atributů.

Atributovaný řetězec nad abecedou V je řetězec atributovaných symbolů z V.

Atribut a přiřazený symbolu x značíme $x.a$. Atributovaný symbol x s přiřazenými atributy $a_1, a_2, \ldots, a_n$ označíme $x[x.a_1, x.a_2, \ldots, x.a_n]$, kde $x.a_i$ mohou být nahrazeny hodnotami z oboru hodnot atributu a_i, $1 \le i \le n$.

Následující definice popisuje atributovou překladovou gramatiku, která je jedním z formálních modelů pro popis atributovaného překladu.

Definice 9. Atributová překladová gramatika

Atributovaný překlad je relace $Z_A \subset T^* \times D^*$, kde T^* je množina vstupních atributovaných řetězců, D^* je množina výstupních atributovaných řetězců.

Překladová gramatika je pětice ${}_PG = (N, T, D, R, S)$, kde N, T, D jsou konečné množiny neterminálů, vstupních symbolů, výstupních symbolů a S je startující neterminální symbol a každé pravidlo v množině pravidel R je tvaru: $r: A \to x$, kde $A \in N$, $x \in (N \cup T \cup D)^*$.

Atributová překladová gramatika je čtveřice ${}_{AP}G = ({}_PG, A, V, F)$, kde

${}_PG$ je překladová gramatika ${}_PG = (N, T, D, R, S)$ a nazývá se základní překladová gramatika;

A je konečná množina *syntetizovaných* (S) a *dědičných* (I) atributů, kde $S \cap I = \varnothing$;

V: $N \cup T \cup D \to 2^A$ je zobrazení, kde $S(X)$ označuje množinu syntetizovaných atributů neterminálu X a $I(X)$ dědičných atributů neterminálu X tak, že $V(X \in N) = S(X) \cup I(X)$ a $S(X) \cap I(X) = \varnothing$, $V(X \in T) = S(X)$, $V(X \in D) = I(X)$;

F je konečná množina sémantických pravidel. Pro každý symbol X_k z pravé strany pravidla $r \in R$ a dědičný atribut $d \in I(X_k)$ je dáno sémantické pravidlo d ß $f_{r,d,k}(a_1, a_2, \ldots, a_m)$, kde $a_1, a_2, \ldots, a_m$ jsou atributy symbolů z pravidla r; pro každý syntetizovaný atribut s symbolu A na levé straně $r \in R$ je definováno sémantické pravidlo s ß $f_{r,s,A}(a_1, a_2, \ldots, a_m)$, kde $a_1, a_2, \ldots, a_m$ jsou atributy symbolů v pravidle r.

Poznámka: Kvůli možnosti určení všech atributů předpokládejme, že jsou zadány: (a) hodnoty dědičných atributů počátečního symbolu gramatiky a (b) hodnoty syntetizovaných atributů všech vstupních symbolů.

V případě atributové gramatiky je základní gramatikou bezkontextová gramatika, která neobsahuje výstupní symboly. Z definice nejsou patrny všechny okolnosti, na kterých může hodnota atributu výstupního symbolu záležet:

- na hodnotách atributů vstupních symbolů,
- na struktuře vstupního řetězce (resp. na vstupním řetězci samotném),
- na zadaných parametrech (konstanty parametrizující překlad).

Myšlenka – Atributová řízená gramatika

Atributová maticová (programovaná) gramatika nebyla v žádné studované literatuře popsána a mohlo by být zajímavé gramatiku podobného typu vytvořit a zkombinovat možnosti samotné řízené gramatiky pro popis některých formálnějších nebezkontextových vlastností (viz následující dvě hlavní kapitoly). Základním nedostatkem takovéto invence je nedostatečně efektivní algoritmus obecné syntaktické analýzy jazyků založených na těchto řízených gramatikách. Dále by bylo možné řízené gramatiky patřičným způsobem omezit, aby více odpovídaly účelům atributové gramatiky, např. aby reprezentovaly graf závislosti nebo již přímo jeho topologické uspořádání.

Následuje několik odstavců o syntaxí řízeném překladu (popsáno v knize [1]), který je podobný formálnějšímu modelu atributovaných překladových gramatik.

Základní kroky překladu:

1. Analýza vstupu (lexikálního charakteru)
2. Vytvoření rozkladového stromu (angl. *parse tree*)
3. Zkonstruování grafu závislostí atributů
4. Převod na strom pro vyhodnocení sémantických pravidel (včetně pořadí)
5. Výsledkem je překlad vstupu

Pro výpočet sémantických pravidel existuje několik přístupů:

- jednoprůchodový (např. L-atributové překlady bez nutnosti explicitní konstrukce rozkladového stromu)
- víceprůchodový

1.2.1 Syntaxí řízený překlad

Definice 10. Syntaxí řízená definice

Syntaxí řízená definice je atributová gramatika se sémantickými pravidly s povolenými vedlejšími účinky (angl. *side effect*).

Syntaxí řízená definice je v praxi pro překlad používána právě pro svou větší univerzálnost než atributové překladové gramatiky, které jsou omezeny formálnějším popisem sémantických akcí.
Sémantické akce syntaxí řízené definice povolují:

- generování kódu (výstupu)
- uložení informací do tabulky symbolů
- zahlášení chyby na chybový výstup
- kód s vedlejšími účinky (např. změna globální proměnné, výpis na výstupní zařízení)
- a jiné (sémantické akce v syntaxí řízeném překladu mohou obsahovat prakticky libovolný programový kód)

Hodnota atributu uzlu rozkladového stromu je dána sémantickým pravidlem daného gramatického pravidla použitého v tomto uzlu. Hodnota syntetizovaného atributu (dále S-atribut, angl. *synthetized atribute*) se vyhodnocuje s využitím atributů z podstromu daného uzlu. Hodnota dědičného atributu (dále I-atribut, angl. *inherited atribute*) se vyhodnocuje s využitím atributů rodiče a přímých potomků tohoto rodiče. Tyto vzájemné závislosti atributů nám určují tzv. graf závislosti (viz níže), ze kterého lze potom vypočítat možné validní pořadí vyhodnocování sémantických pravidel.

S-atributovaná syntaxí řízená definice pracuje vždy jen se syntetizovanými atributy a tedy stylem zdola nahoru. Potom lze například generátory LR analyzátorů přizpůsobit na mechanickou implementaci S-atributované definice založené na LR gramatice (tj. základní gramatika je LR gramatika).

Hodnota I-atributů je závislá na rodiči (angl. *parent*) a/nebo jeho přímých potomcích (angl. *siblings*). Tento typ atributů se používá pro vyjádření závislosti programové konstrukce (tj. nějakého podstromu rozkladového stromu) na kontextu, ve kterém se nachází. Jistě jste si již povšimli, že právě toto je důvod, proč jsme nuceni bezkontextovou gramatiku různými způsoby rozšiřovat (ať už praktickými způsoby nebo formálně). S trochou nadsázky můžeme říci, že právě řízenými gramatikami se v této práci snažíme nutnost používání I-atributů eliminovat nebo alespoň omezit. Vyhodnocení I-atributů probíhá shora dolů.

Věta 1: Každá syntaxí řízená definice (i využívající I-atributy) lze transformovat na ekvivalentní syntaxí řízenou definici obsahující pouze S-atributy.

Poznámka: Předchozí věta je podle mého názoru možná pouze díky tomu, že i sémantická pravidla pracující syntetizovaným způsobem mají možnosti vkládání prakticky libovolného programového kódu (či odkazu na libovolnou proceduru) do sémantických akcí (včetně vedlejších efektů).

Definice 11. Graf závislosti

Graf závislosti je orientovaný acyklický graf reprezentující vzájemné závislosti atributů v rozkladovém stromě.

Algoritmus: Vytvoření grafu závislosti

Pro každé sémantické pravidlo tvaru b ß $f(c1, c2, \ldots, c_n)$ a pro každý uzel u rozkladového stromu využívající toto sémantické pravidlo se vytvoří hrana z atributu uzlu c_i do atributu uzlu b v daném uzlu u, pro všechna $i \in \{1, 2, \ldots, n\}$.

Definice 12. Topologická uspořádanost

Topologicky uspořádaný orientovaný acyklický graf je seznam uzlů $m_1, m_2, \ldots, m_k$, kde pro každé m_i, m_j takové, že m_i à m_j, platí $i < j$. Problémem při výpočtu topologického uspořádání jsou tedy cykly v grafu.

Věta 2: Každé topologické uspořádání grafu závislosti je validní pořadí (seznam, posloupnost) pro vyhodnocení sémantických pravidel v uzlech rozkladového stromu pro daný vstup.

Syntaxí řízený překlad se skládá z následujících kroků:

1. základní gramatika vytvoří rozkladový strom na základě vstupu
2. z atributů anotovaného rozkladového stromu se zkonstruuje graf závislosti
3. z topologického uspořádání tohoto grafu nezávislosti dostaneme pořadí vyhodnocení sémantických pravidel
4. vyhodnocením sémantických pravidel dostaneme překlad vstupu

1.3 Nedostatky bezkontextových gramatik

Teorie okolo bezkontextových gramatik a jazyků je asi (spolu s regulárními jazyky) nejpropracovanější model formálních jazyků, včetně jeho mnohých aplikací a praktických algoritmů. Bohužel svět není zdaleka bezkontextový. Kontext je potřeba uvažovat v mnoha reálných situací, ať už v oblasti programovacích jazyků, přirozeného jazyka či v biologii.

V následujících několika bodech krátce nastíním několik příkladů, kdy nám bezkontextový popis nedostačuje (převzato a inspirováno knihou [3], kapitola 0.4).

1.3.1 Přirozený jazyk

Většina nebo snad i všechny přirozené jazyky nejsou ani regulární ani bezkontextové. Jako příklad si vezměme českou větu:

Jan, Marie a David jsou Čech, Slovenka a Angličan.

Tato věta je správně pouze, pokud každému jménu, respektive pohlaví, kterému jméno odpovídá, správně přísluší národnost a to buď v rodě mužském nebo ženském. Uvažujme, že obecně může být tento jmenný seznam nekonečný, takže je potřeba tuto kontextovost zachytit nějakým modelem. Pokud bychom vynechali nepodstatné části věty, tak lze demonstrovanou vlastnost popsat jazykem

$$L = \{xx \mid x \in \{a, b\}^*, |x| \geq 2\},$$

který není bezkontextový.

1.3.2 Programovací jazyky

Sice bylo dokázáno (Ginsburg, Rice 1962), že každý programovací jazyk popsaný Back-Naurusovou formou (dále BNF, případně rozšířená EBNF) tvoří bezkontextový jazyk, což bylo demonstrováno na jazyce ALGOL. Ovšem ještě ten samý rok Floyd ukázal, že kvůli sémantickým podmínkám kladeným na správnou větu jazyka ALGOL, tento jazyk není bezkontextový. Zobecnění tohoto faktu má pro praxi zásadní význam až do dneška.

Vše lze vidět na jednoduchém příkladě věty jazyka ALGOL:

```
begin
      integer x;
      y := 1;
end,
```

která je úplně správně (tedy splňuje i sémantické kontroly) pouze v případě, že řetězce *x* a *y* jsou totožné. Tato vlastnost je velmi podobná jazyku *L* z předchozí podkapitoly 1.3.1.

1.3.3 Biologické a ekonomické problémy

Dále se využitím formálních jazyků zabývali Herman a Rozenberg (1975), kteří ukázali, že tzv. Lindenmayerovy systémy jsou schopné popsat jednoduchý růst červené řasy díky svému paralelismu v derivačním kroku, čehož bezkontextová gramatika není schopna.

Také v modelování ekonomických problémů, jako je problém kurýra nebo problém plánování (sestavování rozvrhů), jsou bezkontextové gramatiky jako model nedostačující a je třeba využít modely mocnější.

Většinu těchto problémů lze řešit použitím mocnějších kontextových nebo dokonce neomezených gramatik, které ale mají podstatně složitější tvar pravidel než bezkontextové gramatiky. Z tohoto důvodu se zavedly řízené gramatiky, které mají pravidla většinou bezkontextového tvaru a zvýšení mocnosti zajišťují modifikací samotného derivačního kroku (někdy je nutná i dodatečná informace, tedy rozšíření gramatiky o další komponentu). Tímto typem gramatik se budeme zabývat ve zbytku této práce.

2 Řízené gramatiky

Většinou ovšem při popisu dříve nastíněných nebezkontextových problémů vystačíme i s modely jednoduššími než jsou kontextové gramatiky nebo dokonce Turingovy stroje. Proto byly v 70. letech 20. století hojně zaváděny různé modifikace bezkontextových gramatik, které si právě kladly za cíl zvýšit mocnost modelu, a přesto zbytečně nezkomplikovat jeho pravidla. Několik takových modifikací a rozšíření, které většinou označujeme jako *řízené gramatiky* si nyní popíšeme.

Není-li řečeno jinak, tak jako základ pravidel všech dále definovaných řízených gramatik uvažuji pravidla bezkontextového tvaru (viz Definice 7).

Definice 13. Výpočet výskytů symbolů v řetězci

Nechť $occur(x, a)$ je funkce $\Sigma^* \times \Sigma \to N \cup \{0\}$ vracející počet výskytů symbolu $a \in \Sigma$ v řetězci $x \in \Sigma^*$, kde Σ je libovolná konečná množina.

Nechť $|x|_A$ je funkce $\Sigma^* \times \Gamma \to N \cup \{0\}$, $\Gamma \subseteq \Sigma$ vracející součet jednotlivých $occur(x, a)$ pro všechna $a \in A$ v řetězci $x \in \Sigma^*$, kde Σ je libovolná konečná množina. Poznámka: $occur(x, a) = |x|_{\{a\}}$.

Definice 14. Problémy rozhodnutelnosti (angl. *decision problems*)

Nechť Ω je libovolná třída gramatik nebo systémů a G, G_1, $G_2 \in \Omega$ jsou gramatiky této třídy.

Problém členství (angl. *membership problem*) je dán otázkou, zda daný řetězec $w \in V^*$ patří do jazyka generovaného gramatikou G určitého typu Ω, tj. $w \in L(G)$?

Problém ekvivalence (angl. *equivalence problem*) je dán otázkou, zda jsou gramatiky G_1 a G_2 ekvivalentní, tj. $L(G_1) = L(G_2)$?

Problém prázdnosti jazyka (angl. *emptiness problem*) je dán otázkou, zda jazyk $L(G)$ generovaný gramatikou G je prázdný, tj. $L(G) = \emptyset$?

Poznámka: Rozhodnutelnost značíme značkou „+“ a nerozhodnutelnost „–“.

2.1 Gramatiky konečného indexu

Neformálně řečeno, index gramatiky je maximální počet neterminálů, které se objeví během derivace věty jazyka (ve všech větných formách) při použití nejúspornější derivace („nejúspornější“ zde znamená „za využití nejmenšího možného počtu neterminálů“). Konečnost indexu je sice přirozenou, ale velmi silně omezující podmínkou se zásadním vlivem na výslednou sílu.

Definice 15. Konečný index

Nechť G je gramatika libovolného typu a nechť N je její abeceda neterminálů, T je abeceda terminálů a S startující axiom. Pak pro derivaci

$$D: S = w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_r = w \in T^* \text{ podle } G \text{ mějme}$$
$$Ind(D, G) = \max\{|w_i|_N \mid 1 \le i \le r\}, \text{ a pro } w \in T^* \text{ definujeme}$$

$Ind(w, G) = \min\{Ind(D, G) \mid D$ je derivace řetězce w podle $G\}$.

Index gramatiky G je tedy definován jako

$Ind(G) = \sup\{Ind(w, G) \mid w \in L(G)\}$.

Pro jazyk generovaný gramatikou typu X definujeme $Ind_X(L) = \inf\{Ind(G) \mid L(G) = L, G$ je gramatika typu $X\}$, kde v kontextu této práce bude X vybraný typ řízené gramatiky (viz dále). Nakonec si označme některé třídy jazyků omezených konečným indexem:

Třída jazyků s indexem k: $\boldsymbol{L}_k(X) = \{L \mid L \in \boldsymbol{L}(X), G$ je typu X a $Ind_X(L) \leq k\}$, $k \geq 1$,

Třída jazyků s konečným indexem: $L_{fin}(X) = \bigcup_{k \geq 1} L_k(X)$.

2.2 Maticová gramatika

Definice 16. Maticová gramatika

Maticová gramatika je čtveřice $G = (V, T, M, S)$, kde V, T, S mají stejný význam jako u bezkontextové gramatiky (viz Definice 7) a konečná množina M sekvencí tvaru: m: $(r_1, \ldots, r_n)$, $n \geq 1$, kde $r_1, \ldots, r_n$ jsou přepisovací pravidla tvaru r_i: $A_i \rightarrow \beta_i$, kde $A_i \in (V - T)$, $\beta_i \in V^*$ pro všechna $1 \leq i \leq n$. Sekvenci m nazýváme *matice pravidel*, nebo jednoduše *matice*. V případě, že žádná matice z M neobsahuje vymazávací pravidla, mluvíme o *maticové gramatice bez vymazávacích pravidel*. $P(M) = \{ r_i \mid m$: $(r_1, \ldots, r_n)$, $1 \leq i \leq n$, $m \in M\}$ označuje množinu všech přepisovacích pravidel ze všech matic z M.

Definice 17. Relace přímé derivace maticové gramatiky

Nechť $G = (V, T, M, S)$ je maticová gramatika (viz předchozí definice). Pro dva řetězce x, $y \in V^*$ píšeme $x \Rightarrow_G y$ $[m]$ (v případě jednoznačnosti lze $_G$ nebo $[m]$ ze zápisu vypustit) právě, když existuje posloupnost řetězců $x_0, x_1, \ldots, x_n \in V^*$ a matice m: $(r_1, \ldots, r_n) \in M$, r_i: $A_i \rightarrow \beta_i$, $1 \leq i \leq n$, takové že $x_0 = x$, $x_n = y$ a pro všechna $0 \leq i \leq n - 1$ platí:

$$x_{i-1} = u_{i-1} A_i v_{i-1} \text{ a } x_i = u_{i-1} \beta_i v_{i-1}, \text{ kde } u_{i-1}, v_{i-1} \in V^*.$$

Standardním způsobem je definován tranzitivní a reflexivně-tranzitivní uzávěr *relace přímé derivace* $\Rightarrow$, a značíme jej $\Rightarrow^+$ a $\Rightarrow^*$. Jazyk generovaný maticovou gramatikou G je definován jako $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$.

Definice 18. Maticová gramatika s kontrolou na výskyt

Maticová gramatika s kontrolou na výskyt (angl. *appearance checking*) je pětice $G = (V, T, M, S, F)$, kde první čtyři komponenty mají stejný význam jako u obyčejné maticové gramatiky a poslední komponenta F je konečná množina pravidel, $F \subseteq P(M)$. *Relace přímé derivace* $x \,{}_{ac}\Rightarrow_G y$ $[m]$ je definována tak, že je možno při aplikaci posloupnosti pravidel matice m přeskočit aplikaci těch pravidel matice m, která nejsou v dané konfiguraci aplikovatelná a jsou z množiny F. Uzávěry ${}_{ac}\Rightarrow^+$, ${}_{ac}\Rightarrow^*$ a jazyk $L(G)$ jsou definovány analogicky.

Definice 19. Zjednodušená maticová gramatika

Zjednodušená maticová gramatika stupně n, $n \geq 1$, je $(n + 3)$-tice $G = (V_1, V_2, \ldots, V_n, T, M, S)$, kde $V_1, V_2, \ldots, V_n, T$ jsou navzájem po dvojicích disjunktní množiny neobsahující S, M je konečná množina matic následujících možných tvarů:

1. $(S \rightarrow w)$, $w \in T^*$,
2. $(S \rightarrow A_1A_2\ldots A_n)$, $A_i \in V_i$, $1 \leq i \leq n$,
3. $(A_1 \rightarrow x_1, A_2 \rightarrow x_2, \ldots, A_n \rightarrow x_n)$, $A_i \in V_i$, $x_i \in (T \cup V_i)^*$, a $|x_i|_{Vi} = |x_j|_{Vj}$ pro všechna $i, j \in (1, 2, \ldots n)$.

Nechť x, y jsou větné formy, pak $x \Rightarrow y$ píšeme právě, když

a) $x = S$, $(S \rightarrow y) \in M$

nebo

b) $x = y_1A_1z_1y_2A_2z_2\ldots y_nA_nz_n$, $y = y_1w_1z_1y_2w_2z_2\ldots y_nw_nz_n$, $y_i \in T^*$, $A_i \in V_i$, z_i, $w_i \in (T \cup V_i)^*$ pro všechna $i \in (1, 2, \ldots n)$, $(A_1 \rightarrow w_1, A_2 \rightarrow w_2, \ldots, A_n \rightarrow w_n) \in M$.

Slovně řečeno, derivace se aplikuje nejlevějším možným způsobem na každý z n podřetězců z $(T \cup V_i)^*$ v aktuální větné formě. Jazyk takovéto gramatiky je definován klasickým způsobem.

Nechť $X \in \{CF, CF - \varepsilon, LIN, LIN - \varepsilon, REG, REG - \varepsilon\}$, pak $\boldsymbol{L}(SM, X, n)$ označuje třídu jazyků generovanou zjednodušenými maticovými gramatikami stupně n typu X a $\boldsymbol{L}(SM, X) = \cup_{n \geq 1} \boldsymbol{L}(SM, X, n)$.

Příklad zjednodušené maticové gramatiky pro jazyk: $L = gsm(V(n, c))$, kde $V(n, c) = \{(wc)^n \mid w \in V^+, c \notin V^*\}$, n je přirozené číslo, pak $L \in \boldsymbol{L}(SM, REG)$: $V_1 = \{A_1\}$, $V_2 = \{A_2\}$, …, $V_n = \{A_n\}$, $M = \{(S \rightarrow A_1A_2\ldots A_n), (A_1 \rightarrow aA_1, A_2 \rightarrow aA_2, \ldots, A_n \rightarrow aA_n), (A_1 \rightarrow bA_1, A_2 \rightarrow bA_2, \ldots, A_n \rightarrow bA_n), (A_1 \rightarrow ac, A_2 \rightarrow ac, \ldots, A_n \rightarrow ac), (A_1 \rightarrow bc, A_2 \rightarrow bc, \ldots, A_n \rightarrow bc)\}$.

Teorém 3: $\boldsymbol{L}(SM, CF - \varepsilon) \subset \boldsymbol{L}(SM, CF) \subset \boldsymbol{L}(CS)$ (z [3], teorémy 1.5.1 a 1.5.5, str. 71 a 79).
Poznamenejme, že problém prázdnosti jazyka je pro zjednodušené maticové gramatiky rozhodnutelný (podle důsledku 3 ze strany 71 z [3]).

2.3 Programovaná gramatika

Definice 20. Programovaná gramatika

Programovaná gramatika je čtveřice $G = (V, T, P, S)$, kde všechny komponenty mají stejný význam jako u bezkontextové gramatiky. Všechna pravidla jsou tvaru:

$$(r: A \rightarrow \beta, R, F), \text{ kde } A \in V^*, \beta \in V^*, R \subseteq Lab(P), F \subseteq Lab(P) \text{ a}$$

R nazýváme množinou následných pravidel (angl. *success field*) a F zotavovací množinou pravidla r (angl. *failure field*).

Pokud je alespoň u jednoho pravidla zotavovací množina neprázdná, mluvíme o tzv. *programované gramatice s kontrolou na výskyt*.

2.3.1 Grafová reprezentace programovaných gramatik

V [4] lze nalézt velmi praktický a intuitivní způsob konstrukce grafu (diagramu) pro danou programovanou gramatiku. Pro konstrukci grafu jsou použita pravidla programované gramatiky a jejich vzájemné vztahy přes množiny následných pravidel a zotavovacích pravidel přirazené každému pravidlu. Právě přes tento vztah je možno v programovaných gramatikách modelovat kontext ve větě nad rámec bezkontextových jazyků. Připomeňme si, že programovanými gramatikami ale nelze dosáhnout schopností kontextových jazyků.

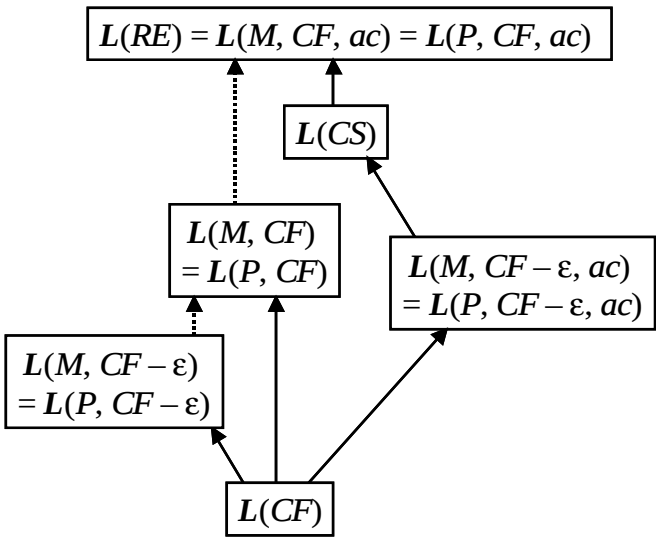
Definice 21. Graf programované gramatiky

Graf programované gramatiky je orientovaný graf s ohodnocenými hranami, jehož vrcholy (uzly) jsou pravidla gramatiky a orientované hrany jsou definovány prostřednictvím vztahů mezi pravidly dané množinou následných pravidel a množinou zotavovacích pravidel přiřazených každému pravidlu programované gramatiky (tyto množiny mohou být i prázdné) následovně:

Pro každá dvě pravidla r_{i-1}: $B \rightarrow b$, R_{i-1}, F_{i-1} a r_i: $A \rightarrow a$, R_i, F_i a $X \in \{R_{i-1}, F_{i-1}\}$, pro která platí, že $r_i \in X$, pak vrcholy r_{i-1} a r_i spojíme hranou orientovanou od r_{i-1} k r_i a ohodnocenou X.

2.4 Hierarchie řízených gramatik

Po definicích několika zajímavých typů řízených gramatik je vhodné si uvést mocnosti jednotlivých tříd jazyků daných těmito gramatikami. Největší vypovídající hodnotu má také porovnání s Chomského hierarchií tříd jazyků, které tvoří jeden ze základních pilířů teorie formálních jazyků i teorie překladačů (kde má smysl uvažovat třídy definované různými modely automatů; viz předchozí Obr. 1:).



Obr. 2: *Hierarchie tříd jazyků definovaných řízenými gramatikami*

Poznámka: Obrázek 2 je vytvořen z teorémů v [3] na stránkách 31-43, kde je možno nalézt i další věty a důkazy včetně několika významných otevřených problémů (např. Jaký je vztah mezi třídou $\boldsymbol{L}(M, CF)$ a třídami $\boldsymbol{L}(RE)$, $\boldsymbol{L}(CS)$ a $\boldsymbol{L}(M, CF - \varepsilon, ac)$.

2.5 Vlastnosti řízených gramatik

Pro praktické použití různých modelů řízených gramatik má značný význam také rozhodnutelnost některých základních problémů (viz Definice 14). Pro nás je asi nejzajímavější problém členství, který je rozhodnutelný ve všech třídách jazyků kromě třídy rekurzivně spočetných jazyků (nejmocnější třída gramaticky/algoritmicky popsatelných jazyků).

Problém:	$\boldsymbol{L}(REG)$	$\boldsymbol{L}(CF)$	$\boldsymbol{L}(M, CF - \varepsilon)$	$\boldsymbol{L}(M, CF)$	$\boldsymbol{L}(M, CF - \varepsilon, ac)$	$\boldsymbol{L}(CS)$	$\boldsymbol{L}(RE)$
Členství	+	+	+	+	+ (NP)	+	-
Prázdnosti	+	+	+	+ (NP)	-	-	-
Ekvivalence	+	-	-	-	-	-	-

Tab. 1: *Vlastnosti gramatik Chomského hierarchie a řízených gramatik* ([3], [6])

Poznámka (NP) u některých tříd jazyků znamená, že rozhodnutelnost dané otázky je NP problém, tj. rozhodnutelný nedeterministickým Turingovým strojem v polynomiálním čase. Vzhledem k dosavadní neexistenci takového stroje se z praktického hlediska jedná obecně o exponenciální časovou složitost.

U zjednodušených maticových gramatik (pro nedostatek prostoru nejsou uvedeny v Tab. 1:; všechny uvedené problémy jsou rozhodnutelné v polynomiálním čase (kubickém resp. kvadratickém)) je zajímavé, že mají kubickou časovou složitost pro řešení otázky členství věty v jazyce popsaného takovouto gramatikou stejně jako u gramatik bezkontextových.

V literatuře o řízených gramatikách je také často studována takzvaná složitost popisu jazyka daným typem řízené gramatiky (angl. *descriptional complexity*). Po prostudování tohoto tématu jsem nenašel žádné klíčové vlastnosti pro řízené gramatiky. Při porovnání v jednotlivých metrikách (např. počet nutných neterminálů, počet pravidel) většinou převyšují bezkontextové gramatiky, ovšem často díky navýšení jiné neměřené metriky. Zde vidím jistý prostor pro další zkoumání, kdy bychom se snažili snižovat různé metriky současně (např. snížit počet neterminálů při zachování počtu pravidel apod.).

Teorém 4: ([3], str. 51, teorém 1.3.6):
Je nerozhodnutelné, zda maticová gramatika bez vymazávacích pravidel a bez kontroly na výskyt generuje bezkontextový jazyk.

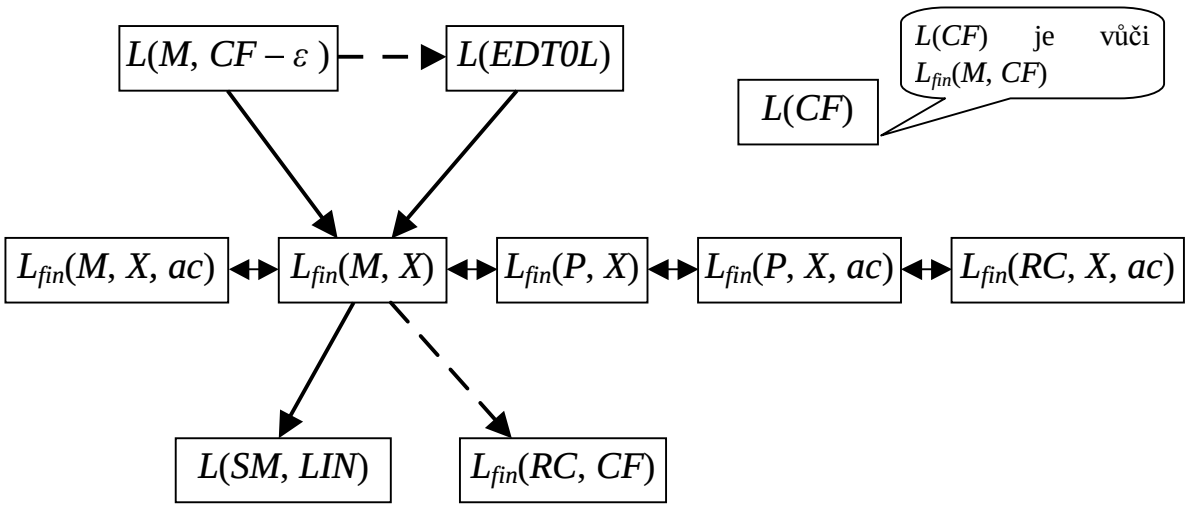
2.6 Generativní síla řízených gramatik s konečným indexem

Následuje několik zajímavých výsledků vybraných z kapitoly 3 v [3].

Věta 5: $\boldsymbol{L}_{fin}(M, CF) = \boldsymbol{L}_{fin}(M, CF - \varepsilon) = \boldsymbol{L}_{fin}(M, CF, ac) = \boldsymbol{L}_{fin}(M, CF - \varepsilon, ac)$.

Tedy slovně shrnuto, při zavedení omezení gramatiky konečným indexem nemá smysl se příliš zabývat povolením nebo zakázáním vymazávacích pravidel nebo kontrol výskytu. Dále existuje velké množství různých typů řízených gramatik se shodnou mocností. Pro usnadnění zápisu $X \in \{CF, CF - \varepsilon\}$.

Věta 6: $\boldsymbol{L_{fin}}(M, X) = \boldsymbol{L_{fin}}(P, X) = \boldsymbol{L_{fin}}(P, X, ac) = \boldsymbol{L_{fin}}(RC, X, ac)$, kde *RC* je gramatika s nahodilým kontextem (viz [3], str. 30 a [6] str. 115-119).



Obrázek 1: Zařazení gramatik konečného indexu do zkrácené hierarchie jazyků

Poznamenejme, že $\boldsymbol{L}(EDT0L)$ označuje třídu jazyků generovanou rozšířenými deterministickými *L*-systémy s tabulkami, kterými se v této práci nezabýváme.

Věta 7: $\boldsymbol{L}(CF) - \boldsymbol{L_{fin}}(M, CF) \neq \emptyset$.

Věta 8: $\boldsymbol{L_{fin}}(M, X, \textit{left-2}) = \boldsymbol{L_{fin}}(M, X)$, kde *left-2* je typ nejlevější derivace.

Věta 9: Problémy inkluze věty do jazyka, prázdnosti jazyka a konečnosti jazyka jsou pro maticové gramatiky konečného indexu rozhodnutelné.

Věta 10: Pro dvě libovolné maticové gramatiky konečného indexu G_1, G_2 jsou následující problémy nerozhodnutelné:

Patří daná věta do jazyka $L(G_1)$?

Je $L(G_1)$ regulární jazyk nebo roven jazyku V^*?

Je $L(G_1) \cap L(G_2)$ prázdný, konečný nebo regulární jazyk?

Je $L(G_1) \subseteq L(G_2)$ nebo $L(G_1) = L(G_2)$?

Je $L(G_1)$ bezkontextový jazyk?

Je $L(G_1) \cap L(G_2)$ bezkontextový nebo patří do $\boldsymbol{L_{fin}}(M, CF)$?

Poznámka: Gruska (1969) dokázal, že neexistuje algoritmus pro výpočet *Ind(G)* pro libovolnou bezkontextovou gramatiku *G*. Tvrzení platí i pro maticové gramatiky.

Věta 11: Komplement maticového jazyka konečného indexu je kontextový jazyk.

Vybrané otevřené problémy:

Existují bezkontextové jazyky v $\boldsymbol{L_{fin}}(M, CF)$ mající nekonečný bezkontextový index?

Věta 12: Platí vlastní inkluze $\boldsymbol{L_n}(M, CF) \subset \boldsymbol{L_{n+1}}(M, CF)$ pro všechna $n \geq 1$. Tuto vlastnost označujeme jako *nekonečná hierarchie* jazyků konečného indexu.

Teorém 13: ([3], str. 51, teorém 1.3.5):

Problém členství pro maticovou gramatiku bez vymazávacích pravidel s kontrolou na výskyt je NP-úplný.

Na využitelnosti řízených gramatik s kontrolou na výskyt se značně podepisuje jejich složitost při obecném rozhodování, zda věta patří do jazyka generovaného danou řízenou gramatikou s kontrolou na výskyt. V praxi samozřejmě budou tyto lidsky konstruované gramatiky pravděpodobně vykazovat lepší vlastnosti a možnost rychlejšího zpracování, ale nemůžeme si tím být jisti v současném stavu rozpracovanosti teorie.

3 Využití řízených gramatik

Tato hlavní kapitola se zabývá využitím řízených gramatik pro potřeby popisu syntaxe a především i sémantiky programovacích jazyků.

Nebudeme se zaměřovat na žádný konkrétní jazyk, ale spíše na problém popisu některých sémantických charakteristik společných pro větší množinu programovacích jazyků.

3.1 Sémantické vlastnosti jazyků

Nejprve si popišme a pojmenujme základní problémy popisu sémantiky programovacích jazyků.

Většina nástrojů pro tvorbu překladačů rozšiřuje zápis klasické bezkontextové gramatiky.

3.1.1 Deklarace a definice proměnných

Již v úvodu jsme si ukázali jeden z kořenů nebezkontextovosti většiny programovacích jazyků. Nyní se na tento *deklarační problém* podíváme podrobněji.

Definice: *Deklarace* specifikuje velikost proměnné (u pole počet položek, u záznamu seznam položek apod.), identifikátor (pro pojmenování a následně snadnější přístup v textovém zápisu programu), typ (a z toho plynoucí množinu aplikovatelných operací nad touto proměnnou) a mnoho dalších aspektů.
Definice: *Definice proměnné* je rezervování paměťového místa pro danou proměnnou a případně inicializace (naplnění) této proměnné nějakou hodnotou.

Většina programovacích jazyků (např. ALGOL, Pascal, C, C++, částečně Smalltalk a mnoho dalších) vyžaduje, aby každé definici proměnné předcházela její deklarace. Některé jazyky umožňují také obě části spojit do jednoho příkazu (např. C++), ale touto situací se nebudeme nyní zabývat, jelikož v případě povolení pouze tohoto způsobu deklarace a definice proměnné se celý problém značně zjednodušuje.

Aby se tento požadavek zaručil již během překladu zdrojového textu programu, je třeba provádět tzv. dodatečné sémantické kontroly, zda byla každá použitá proměnná v předchozím kódu (případně kdekoli v kódu) deklarována. Pro implementaci takovéto kontroly je potřeba dodatečná informační struktura, kterou si během zpracovávání zdrojového textu překladač tvoří, a nazývá se *tabulka symbolů*. Tato kontrola značně komplikuje možnost vytvářet překladače pomocí tzv. generátorů překladačů (např. programy Bison, YACC, ad.), a proto se zkoumají možnosti popsat i tuto sémantickou kontrolu pomocí gramatiky a zjednodušit tak vytváření překladačů.

Toto je v případě použití kontextových jazyků problematické kvůli značném nárůstu obtížnosti gramatiky (především počtu potřebných gramatických pravidel). V tento moment vstupují na scénu řízené gramatiky, kde je v [3] ukázáno na mnoha případových studiích, že

využití řízených gramatik může značně snížit komplexitu (počty potřebných neterminálů a/nebo pravidel) gramatik pro popis jazyků bezkontextových a jazyků silnějších než bezkontextových.

3.1.2 Statická typová kontrola

Statická kontrola typů je jedna z mnoha sémantických kontrol (viz [1], str. 343), které může překladač provádět. Naopak dynamická typová kontrola je prováděna až za běhu programu.

Kontrola typu verifikuje typy výrazů, aby odpovídaly jejich kontextům např. parametr funkce či požadavek na typ operandu určitého operátoru apod.

Pokud bychom se dívali na všechny konstrukce jako na funkce, jak to dělají například funkcionální jazyky (případně formální programovací jazyky na bázi lambda-kalkulu), tak bychom si všimli velké míry analogie s deklaračním problémem u identifikátorů a u operací/funkcí, kde je však kromě názvu a typu proměnné zároveň požadována ještě posloupnost (může být prázdná) argumentů. Proměnná je vlastně z tohoto pohledu funkce bez parametrů. V takovémto zobecněném přístupu však musíme mít možnost i funkci přiřazovat různé definice (implementace).

V této práci se touto kontrolou nebudu podrobněji zabývat. Pouze v posledním příkladu převzatém z [4] v kapitole 3.3.2 je modelována mimo jiné i jednoduchá statická typová kontrola atomických datových typů (celočíselný a logický datový typ).

3.2 Vztah maticových gramatik a programovacích jazyků

Podívejme se na návrh řešení problému s deklaracemi proměnných před jejich použitím z kapitoly 3.1.1, které prezentují Dassow a Paun ve své knize (viz [3], str. 263-266) z roku 1989. Tento příklad je zaměřen na jazyk ALGOL 60, ale lze jej zobecnit na většinu dnes používaných jazyků, které vyžadují deklaraci proměnných.

Mějme podmnožinu jazyku ALGOL, jazyk

$$L = \{\ \texttt{begin integer } x\texttt{; } x_1 \texttt{ = 1; } x_2 \texttt{ = 1; } \ldots x_n \texttt{ = 1; end} \mid n \geq 1, x \in \{a, b\}^+, x_i = x, 1 \leq i \leq n\ \}.$$

Nyní si ukážeme, že $L \in \boldsymbol{L}(M, CF - \varepsilon, ac)$ tím, že sestrojíme maticovou gramatiku s kontrolou na výskyt

$G = (V, T, M, S, F)$, kde

$V = \{S, A, B, C, C', D, D_a, D_b, D_a', D_b', E, Z\}$,

$T = \{\texttt{a}, \texttt{b}, \texttt{;}, \texttt{begin}, \texttt{end}, \texttt{integer}, \texttt{=}, \texttt{1}\}$,

$M = \{$

$[m_1]$: $(S \rightarrow \texttt{begin integer } C\texttt{; } A\ B)$,

$[m_2]$: $(A \rightarrow C\ \texttt{= 1;} A, B \rightarrow B)$,

$[m_3]$: $(A \rightarrow C\ \texttt{= 1;}, B \rightarrow D)$,

$[m_4]$: $(D \rightarrow D_a)$, $[m_5]$: $(D \rightarrow D_b)$, $[m_6]$: $(D \rightarrow D_a')$, $[m_7]$: $(D \rightarrow D_b')$,

$[m_8]$: $(C \rightarrow aC', D_a \rightarrow D_a)$, $[m_9]$: $(C \rightarrow bC', D_b \rightarrow D_b)$,

$[m_{10}]$: $(C \rightarrow Z, D_a \rightarrow E)$, $[m_{11}]$: $(C \rightarrow Z, D_b \rightarrow E)$,

$[m_{12}]$: $(C' \rightarrow C, E \rightarrow E)$, $[m_{13}]$: $(C' \rightarrow Z, E \rightarrow D)$,

$[m_{14}]$: $(C \to a, D_a' \to D_a')$, $[m_{15}]$: $(C \to b, D_b' \to D_b')$,
$[m_{16}]$: $(C \to Z, D_a' \to \texttt{end})$, $[m_{17}]$: $(C \to Z, D_b' \to \texttt{end})$,
},
$F = \{C \to Z, C' \to Z\}$.

Při hlubším zamyšlení zjistíme, že bez kontroly na výskyt je problematické tento jazyk maticovou gramatikou popsat. Nemůžeme ovšem tvrdit, že to nelze, protože to by muselo být nejdříve formálně dokázáno. Jiná situace nastane v případě, že máme o něco jednodušší gramatiky než jsou maticové, a to konkrétně zjednodušené maticové gramatiky (viz Definice 19), a maticové gramatiky s konečným indexem. V těchto případech je v [3] dokázáno, že těmito gramatikami výše popsaný jazyk modelovat nelze. Respektive budeme vždy nuceni omezit se pouze na konečný počet různých identifikátorů (tj. muselo by platit, že $x \in \{a, b\}^+$, $|x| \leq m$, m je kladná přirozená konstanta známá již při tvorbě gramatiky (viz příklad u definice zjednodušené maticové gramatiky). Navíc by nám tímto stylem značně narostlo, v závislosti na m, množství pravidel.

Dále vidíme, že počet pravidel je také lineárně závislý na velikosti abecedy, z jejíchž symbolů tvoříme identifikátor.

Příklad úspěšné derivace programu `begin integer ba; ba = 1; ba = 1; end`, kdy byla spolu se syntaktickou správností provedena i jednoduchá sémantická kontrola:

$S \underset{ac}{\Rightarrow}$ `begin integer` C`;` A B $[m_1]$
$\underset{ac}{\Rightarrow}$ `begin integer` C`;` C `= 1;` A B $[m_2]$
$\underset{ac}{\Rightarrow}$ `begin integer` C`;` C `= 1;` C `= 1;` D $[m_3]$
$\underset{ac}{\Rightarrow}$ `begin integer` C`;` C `= 1;` C `= 1;` D_b $[m_5]$
$\underset{ac}{\Rightarrow}$ `begin integer` bC'`;` bC' `= 1;` bC' `= 1;` D_b $[3 \times m_9]$
$\underset{ac}{\Rightarrow}$ `begin integer` bC'`;` bC' `= 1;` bC' `= 1;` E [m_{11} s vynecháním $C \to Z$]
$\underset{ac}{\Rightarrow}$ `begin integer` bC`;` bC `= 1;` bC `= 1;` E $[3 \times m_{12}]$
$\underset{ac}{\Rightarrow}$ `begin integer` bC`;` bC `= 1;` bC `= 1;` D [m_{13} s vynecháním $C' \to Z$]
$\underset{ac}{\Rightarrow}$ `begin integer` bC`;` bC `= 1;` bC `= 1;` D_a' $[m_6]$
$\underset{ac}{\Rightarrow}$ `begin integer` ba`;` ba `= 1;` ba `= 1;` D_a' $[3 \times m_{14}]$
$\underset{ac}{\Rightarrow}$ `begin integer` ba`;` ba `= 1;` ba `= 1;` `end` [m_{16} s vynecháním $C \to Z$]

Následuje příklad selhání syntaktické analýzy, která kontroluje i deklaraci proměnné před jejím použitím, pro větu `begin integer ba; ba = 1; a = 1; end`:

$S \underset{ac}{\Rightarrow}^3$ `begin integer` C`;` C `= 1;` C `= 1;` D $[m_1, m_2, m_3]$,

tyto tři kroky šlo provést deterministicky; nyní se pokusíme generovat dále dvěma cestami:

a) $\underset{ac}{\Rightarrow}$ `begin integer` C`;` C `= 1;` C `= 1;` D_b $[m_5]$
$\underset{ac}{\Rightarrow}$ `begin integer` bC'`;` bC' `= 1;` C `= 1;` D_b $[2 \times m_9]$,

nyní lze provést pouze matici m_{11} (m_9 nelze použít, protože potřebujeme vygenerovat identifikátor a), která způsobí vygenerování symbolu.

Další možnost, jak po prvních třech derivačních krocích pokračovat, je:

b) $_{ac}\Rightarrow$ `begin integer` C`;` C `=` `1;` C `=` `1;` $D_a'[m_4]$

$_{ac}\Rightarrow$ `begin integer` C`;` C `=` `1;` a `=` `1;` $D_a'[m_{14}]$,

nyní lze provést pouze matici m_{16}, která opět zablokuje symbolem Z možnost dokončení derivace. Existují ještě další analogické možnosti, které ovšem také končí zablokováním.

Poznámka k formálním důkazům vlastností programovacích jazyků:

Programovací jazyky mají často velmi složitou gramatiku a jakékoli matematické důkazy nad takto velkými gramatikami by byly extrémně náročné až neproveditelné, nemluvě o jejich pochopitelnosti. Z tohoto důvodu se takovýto jazyk omezuje pomocí nějaké funkce nebo homomorfismu na jazyk podstatně jednodušší, který reprezentuje pouze základní jádro vlastnosti programovacího jazyka, kterou se snažíme popsat a něco o ní dokázat.

Například uvažujme pro předchozí jazyk homomorfismus h, který vymaže z každé věty všechny symboly kromě těch, které tvoří identifikátory (a, b), a těch které oddělují příkazy (středník). Pak dostáváme $h(L) = \{(w;)^n \mid n \geq 2, w \in \{a, b\}^+\}$, kde jsou kulaté závorky zamýšleny pouze jako metaznaky a nepatří do abecedy jazyka. A jak již bylo zmíněno dříve, $h(L) \in \boldsymbol{L}(SM, CF) \cup \boldsymbol{L}_{fin}(M, CF)$, což je dokázáno v [3] na stranách 264 a 265. Dále je v této knize uvedena hypotéza o tom, že pravděpodobně $h(L) \in \boldsymbol{L}(M, CF)$, což je ale poměrně zajímavý odhad, protože ani neznáme maximální sílu tohoto typu gramatik, protože $\boldsymbol{L}(M, CF) \subseteq \boldsymbol{L}(RE)$ je stále otevřený problém (tj. Je tato inkluze vlastní?).

Dalšími sémantickými vlastnostmi programovacích jazyků se v duchu této studie žádná literatura nezabývá. Jenom v [3] je doplněno, že u dalších vlastností zatím není k dispozici patřičné formální zdokumentování (případně zjednodušení na základní jádro jako je tomu u $h(L)$).

3.3 Programované gramatiky a popis deklaračního problému

V této podkapitole se budeme zabývat stejným sémantickým problémem, ale z pohledu programovaných gramatik s využitím jejich grafové reprezentace. Následuje zevrubný popis a neúplná ukázka řešení z kapitoly 4 v [4].

Definice 22. Podprogram programované gramatiky G s množinou pravidel P

Podprogram je množina pravidel $P_i \subseteq P$, které tvoří logický celek. Pravidla podprogramu mají v množinách následných pravidel a zotavovacích pravidel pouze odkazy na pravidla právě z tohoto podprogramu nebo pro vyskočení z podprogramu klíčové slovo `out` s parametrem X určujícím, ze které množiny se budou vybírat pravidla po ukončení podprogramu právě tímto klíčovým slovem `out`, $X \in \{s, j\}$. Parametr s znamená, že se bude po ukončení programu vybírat následující pravidlo z množiny následujících pravidel; u j ze zotavovacích pravidel. Dále je třeba definovat nový tvar pravidel pro volání podprogramů: (*jménoPodprogramu*, R, F). Každému podprogramu je navíc nutno určit počáteční neterminál podprogramu (existuje i modifikace určující startující pravidla podprogramu).

Je-li programovaná gramatika tvořena *k* podprogramy, pak platí:

1. všechny množiny pravidel podprogramů jsou navzájem disjunktní;
2. sjednocení všech pravidel všech podprogramů dává množinu pravidel této gramatiky.

Tato zajímavá modifikace programovaných gramatik již značně připomíná princip strukturovaného programování, avšak na úrovni nižší než mají kontextové jazyky. Bylo by jistě zajímavé hlouběji prozkoumat možnosti modulárně skládat programované gramatiky pro složitější jazyky z již předdefinovaných modulů (podprogramů).

Věta 14: Každá programovaná gramatika tvořená z podprogramů lze transformovat na ekvivalentní klasickou programovanou gramatiku (podle Definice 20) a naopak.

3.3.1 Stručný popis ukázkového programovacího jazyka

Volitelná sekvence deklarací proměnných je následována povinnou sekvencí příkazů. Všechny deklarace i příkazy jsou ukončeny středníkem.

Proměnné jsou možné dvou typů: `bool` a `int`. Typová informace se ukládá jako první písmeno identifikátoru proměnné (`i` nebo `b`). Deklarovaná proměnná nemusí být použita v žádném příkazu.

Celočíselné výrazy povolují pouze načtení hodnoty `readint()` a binární operátory různých priorit. Logické výrazy vrací hodnoty `true` nebo `false` a mohou obsahovat binární logické operace nebo relace. Výrazy lze použít pouze v příkazech, které jsou trojího typu: (a) přiřazení `ivar = 1;` (b) vytisknutí výsledku celočíselného výrazu `printint(ivar)` a (c) vytisknutí výsledku logického výrazu `printbool(true)`.

3.3.2 Příklad programované gramatiky pro ukázkový jazyk

Startující neterminál je *S* (určuje startující podprogram(y)). Následuje komentovaný výpis nejdůležitějších podprogramů řešení. Všechny podprogramy včetně přehledné grafové reprezentace naleznete ve 4. kapitole v [4].

Programovaná gramatika s kontrolou na výskyt a bez vymazávacích pravidel je tvořena pěti podprogramy (v závorce je uveden počáteční neterminál podprogramu):

- STRUKTURA (*S*) – startující podprogram vygeneruje základní skeleton zdrojového textu
- PŘÍKAZ (*STMT*) – přepisuje *STMT* a reflektuje možné formy příkazů v jazyce
- DEKLARACE (*DECL*) – přepisuje *DECL*; současně umožňuje „shodně“ přepisovat jeden nebo více výskytů neterminálů (*I*|*B*)*VAR*
- IVÝRAZ (*IE*) – přepisuje neterminál *IE* a reprezentuje celočíselný výraz
- BVÝRAZ (*BE*) – přepisuje neterminál *BE* a reprezentuje logický výraz ve větě

Dále uvedu pouze pravidla modulů STRUKTURA, DEKLARACE a PŘÍKAZ, které jsou dostatečné pro demonstraci řešení deklaračního problému. Ostatní moduly demonstrují

ještě typovou kontrolu operandů binárních (případně unárních) operátorů a podobným způsobem by bylo možné předvést kontrolu argumentů funkcí mezi deklarací a použitím apod.

STRUKTURA:

a_1: ($S \rightarrow$ *STMTS* , $\{a_3, a_4\}$, {out φ})
a_2: ($S \rightarrow$ *DECLS STMTS* , $\{a_3, a_4\}$, $\{\emptyset\}$)
a_3: (*STMTS* $\rightarrow$ *STMT STMTS* , $\{a_7\}$, $\{\emptyset\}$)
a_4: (*STMTS* $\rightarrow$ *STMT* , $\{a_8\}$, $\{\emptyset\}$)
a_5: (*DECLS* $\rightarrow$ *DECLS DECL* , $\{a_9\}$, $\{\emptyset\}$)
a_6: (*DECLS* $\rightarrow$ *DECL* , $\{a_{10}\}$, {out σ})
a_7: (**PŘÍKAZ** , $\{a_3, a_4\}$, $\{\emptyset\}$)
a_8: (**PŘÍKAZ** , $\{a_5, a_6\}$, $\{\emptyset\}$)
a_9: (**DEKLARACE** , $\{a_5, a_6\}$, $\{\emptyset\}$)
a_{10}: (**DEKLARACE** , {out σ}, $\{\emptyset\}$)

PŘÍKAZ:

b_1: (*STMT* $\rightarrow$ *VAR* `=` *EXPR* `;` , $\{b_4, b_5\}$, {out φ})
b_2: (*STMT* $\rightarrow$ `printint` `(` *IE* `)` `;` , $\{b_8\}$, $\{\emptyset\}$)
b_3: (*STMT* $\rightarrow$ `printbool` `(` *BE* `)` `;` , $\{b_9\}$, $\{\emptyset\}$)
b_4: (*VAR* $\rightarrow$ *BVAR* , $\{b_6\}$, $\{\emptyset\}$)
b_5: (*VAR* $\rightarrow$ *IVAR* , $\{b_7\}$, $\{\emptyset\}$)
b_6: (*EXPR* $\rightarrow$ *BE* , $\{b_9\}$, $\{\emptyset\}$)
b_7: (*EXPR* $\rightarrow$ *IE* , $\{b_8\}$, $\{\emptyset\}$)
b_8: (**IVÝRAZ** , {out σ}, $\{\emptyset\}$)
b_9: (**BVÝRAZ** , {out σ}, $\{\emptyset\}$)

DEKLARACE:

c_1: (*DECL* $\rightarrow$ `int i` *NAME* `;` , $\{c_3, c_5–c_8\}$, {out φ})
c_2: (*DECL* $\rightarrow$ `boolean b` *NAME* `;` , $\{c_4–c_8\}$, $\{\emptyset\}$)
c_3: (*IVAR* $\rightarrow$ `i` *NAME* , $\{c_3, c_5–c_8\}$, $\{\emptyset\}$)
c_4: (*BVAR* $\rightarrow$ `b` *NAME* , $\{c_4–c_8\}$, $\{\emptyset\}$)
c_5: (*NAME* $\rightarrow$ `a` *CONT* , $\{c_5\}$, $\{c_9\}$)
c_6: (*NAME* $\rightarrow$ `b` *CONT* , $\{c_6\}$, $\{c_9\}$)
c_7: (*NAME* $\rightarrow$ `a` , $\{c_7\}$, {out σ})
c_8: (*NAME* $\rightarrow$ `b` , $\{c_8\}$, {out σ})
c_9: (*CONT* $\rightarrow$ *NAME* , $\{c_9\}$, $\{c_5–c_8\}$)

V podkapitole 4.3 v [4] je navíc také ukázka modelování stejného jednoduchého programovacího jazyka pomocí neomezené generativní gramatiky. Teoreticky by stačila kontextová gramatika, ale z didaktických a praktických důvodů byla pro pohodlnější popis zvolena mocnější gramatika.

V rámci vlastního studia tohoto příkladu jsem provedl srovnání popisné složitosti neomezené gramatiky a programované gramatiky s kontrolou na výskyt a bez vymazávacích

pravidel pro popis daného jednoduchého programovacího jazyka. Výsledky jsou shrnuty v následující tabulce (tučně jsou zvýrazněny lepší hodnoty).

Vlastnost gramatiky / Gramatika	**Programovaná**	**Neomezená**
Počet pravidel (celkem)	**43**	90
Počet bezkontextových jader pravidel (cfp)	**43**	43
Počet cfp s prázdnou zotavovací množinou	28	**43**
Počet neterminálů	**23**	30
Počet vymazávacích pravidel	**0**	1

Tab. 2: *Porovnání popisné složitosti dvou typů gramatik pro popis jednoduchého programovacího jazyka včetně několika sémantických kontrol*

Všimněme si, že pro popis na syntaktické úrovni některých sémantických vlastností postačují jak programované, tak maticové gramatiky s kontrolou na výskyt a bez nutnosti vymazávacích pravidel.

Popis programovanou gramatikou (a případně i jejím grafem) je však pohodlnější a přehlednější, především díky možnosti intuitivně dekomponovat složitou gramatiku na téměř samostatné moduly (podprogram = podgraf grafu celé programované gramatiky) propojené malým počtem vazeb (orientované hrany v grafu).

3.4 Shrnutí

V kapitolách 3.2 a 3.3 jsme došli k velmi zajímavým závěrům, že minimálně některé sémantické vlastnosti programovacích jazyků lze popisovat jazyky, které tvoří vlastní podtřídu kontextových jazyků, a tudíž by měly mít některé praktické vlastnosti o něco lepší.

Bohužel jsem ve studované literatuře nenarazil na žádnou podstatnou vlastnost (např. různé problémy nebo věty), ve které by byly maticové (potažmo programované) gramatiky s kontrolou na výskyt a bez vymazávacích pravidel výhodnější. O podpoření tohoto přístupu se budu případně pokoušet i v praktické kapitole mé disertační práce.

4 Závěr

Obecně lze říci, že sice existují gramatické modely pro popis jak syntaxe tak sémantiky programovacích jazyků, ale prostředky a algoritmy pro jejich praktické zpracování (analýza, překlad) jsou velmi náročné na časovou a často i prostorovou složitost. Složitost je dána jak velkým množstvím pravidel, které jsou ovlivňovány například velikostí abecedy pro tvorbu identifikátorů, tak přílišnou obecností použitelných formálních gramatických modelů. Toto je také asi nejpádnější důvod, proč v praxi nevídáme využívání těchto nastíněných gramatických modelů právě k účelu popisu programovacích jazyků a raději jsou sémantické vlastnosti implementovány ručně či za využití popisu sémantických akcí, jak je to zabudováno do nejznámějších generátorů překladačů (např. Bizon, YACC).

Rozhodně zajímavou myšlenkou, ke které mě inspirovala tato studie, je kombinace řízených gramatik a atributovaného překladu. V některých ohledech by se mohlo zdát, že využití prakticky všemocných sémantických pravidel je mnohem pohodlnější, ale vyžaduje to často dodatečné programování místo využívání formálně striktně definovaných obratů (např. nutnost využívat vedlejší účinky apod.). Každopádně nebylo možno v časovém rozsahu ani v zadání této práce se touto myšlenkou hlouběji zabývat, což ovšem nevylučuje její prozkoumání v rámci tvořené disertační práce.

4.1 Trendy v syntaktické analýze programovacích jazyků

Nejčastěji zmiňovaným novým trendem v syntaktické analýze formálních jazyků je paralelizace – ať již za účelem dekompozice problému a zrychlení analýzy, nebo za účelem zvýšení generativní síly takovýchto modelů. V praxi zde opět vidíme problém s rozdělováním kontextu, tj. opět je vše komplikováno především sémantickou částí jazyka, což se většinou obchází komunikací a synchronizací jednotlivých paralelních komponent takového systému.

4.2 Trendy v sémantické analýze programovacích jazyků

Trendy zaznamenané v oblasti sémantického vývoje jazyků jsou lehce odlišné. Z mého pohledu se zdá, že se jedná o boj na dvou frontách:

1. snaha o zjednodušení sémantiky na nutné minimum a značné zprůhlednění zdrojových textů takovýchto programovacích jazyků, aniž by se nutně snížila jejich použitelnost (např. Smalltalk, SELF, LISP a další)
2. snaha o rozšíření starých, ale populárních jazyků za účelem vyhovění nejčastějších současných požadavků bez výrazného držení se nějaké koncepce (většina objektových rozšíření rodiny odvozené od jazyka C, např. C++, C#, PHP, atd.)

Přestože se může zdát druhý přístup jako zavrženíhodný, není tomu tak. Je potřeba si uvědomit, že řada programátorů se raději naučí více vlastností jazyka a zjednoduší si tak tvorbu komplikovanějších konstrukcí kódu (ačkoli nutně nemusí jít o přehlednější vyjádření).

Dalším trendem je zušlechťování vývojových prostředí, které se snaží kromě kontrol syntaxe již během samotného psaní kódu, kontrolovat i některé sémantické vlastnosti, jako například deklaraci proměnných nebo statickou typovou kontrolu.

4.3 Další využití řízených gramatik

Řízené gramatiky však lze uplatnit i mimo oblast překladačů a popis programovacích jazyků. Jisté aplikace můžeme nalézt v těchto oblastech (více viz [3], kapitola 9):

- Petriho sítě
- Popis uměleckých děl a folklóru
- Ekonomické procesy

Literatura

[1] Aho, A. V., Sethi, R., Ullman, J. D.: *Compilers - Principles, Techniques, and Tools*, Addison-Wesley, 1986, ISBN 0-201-10194-7, str. 279-386

[2] Dassow, J.: *Grammars with Regulated Rewriting*, [Poznámky k tématu PhD studia na Otto-von-Guericke-Universität Magdeburg], 2004
(Dokument je dostupný na URL http://theo.cs.uni-magdeburg.de/dassow/tarraphd.pdf)

[3] Dassow, J., Paun, G.: *Regulated Rewriting in Formal Language Theory*. Springer, New York, 1989, 308 p., ISBN 0-38751-414-7

[4] Kot, M.: *Řízené gramatiky*, VŠB – Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, Katedra informatiky, [Diplomová práce], 2002

[5] Melichar, B.: Jazyky a překlady, [Skripta], ČVUT, Praha, 2003, str. 203-235

[6] Rozenberg, G., Salomma, A. (Editors): *Handbook of Formal Languages, Volume* 2*: Linear Modelling: Background and Application*, Springer, 1997, ISBN 3-540-60648-3, str. 101-150