FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Výukový nástroj pro LL a LR syntaktickou analýzu
Student:	Jiří Folprecht
Vedoucí:	Ing. Tomáš Pecka
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2025/2026

Pokyny pro vypracování

Seznamte se s LL syntaktickou analýzou [1,3] a LR syntaktickou analýzou [1,2,4].
Seznamte se s aktuálním nástrojem pro výpočet LL(1) rozkladové tabulky [5,6].
Rozšiřte tento nástroj o výpočet rozkladové tabulky pro LL(1) gramatiky a LR(1) gramatiky.
Nástroj musí podporovat zadávání gramatik, výpočet rozkladové tabulky a pomocných funkcí potřebných pro její sestrojení, včetně upozornění na kolize v gramatice.
Dále pak musí podporovat výpočet rozkladu slova v gramatice, včetně výpisu posloupnosti konfigurací analyzátoru.
Kód vhodně otestujte.

[1] AHO, Alfred V.; LAM, Monica S.; SETHI, Ravi a ULLMAN, Jeffrey D. Compilers: principles, techniques & tools. Second edition. Boston: Addison Wesley, 2007. ISBN 0-321-48681-1.
[2] MELICHAR, Bořivoj; JANOUŠEK, Jan a VAGNER, Ladislav. Parsing and translation. Praha: Česká technika - nakladatelství ČVUT, 2013. ISBN 978-80-01-05192-4.
[3] https://courses.fit.cvut.cz/BI-PJP/
[4] https://courses.fit.cvut.cz/NI-SYP/
[5] https://pages.fit.cvut.cz/peckato1/parsingtbl/
[6] https://gitlab.fit.cvut.cz/peckato1/parsingtbl

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 28. února 2024 v Praze.

Bakalářská práce

VÝUKOVÝ NÁSTROJ PRO LL A LR SYNTAKTICKOU ANALÝZU

Jiří Folprecht

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Tomáš Pecka
16. května 2024

Odkaz na tuto práci: Folprecht Jiří. *Výukový nástroj pro LL a LR syntaktickou analýzu.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Seznam obrázků

Seznam tabulek

List of Listings

Rád bych poděkoval panu Ing. Tomáši Peckovi za jeho vedení a rady během psaní této práce. Děkuji také své rodině a přátelům za jejich podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16. května 2024

Abstrakt

Páce se zabývá implementací algoritmů pro výpočet rozkladových tabulek a následně vývojem přívětivého rozhraní v internetovém prohlížeči pro zadávání gramatik a zobrazování výsledných tabulek.

Analytická část popisuje potřebnou teorii, zejména struktury které používáme k popisu principů syntaktické analýzy. Dále nabízí podrobný popis algoritmů používaných k výpočtu rozkladových tabulek a srovnání jednotlivých algoritmů podle typu gramatik. Ukazuje se, že algoritmy určené pro složitější třídy gramatik jsou značně složitější.

Praktická část se zabývá samotnou implementací datových struktur a algoritmů potřebných pro výpočet rozkladových tabulek a tvorbou webového rozhraní. Webové rozhraní je vytvořeno pomocí javascriptové knihovny React a umožňuje uživateli zadávat gramatiky a prezentuje mu výsledky výpočtů.

Klíčová slova Syntaktická analýza, LL, LR, SLR, typescript, vite, npm, frontend, React

Abstract

The thesis deals with the implementation of algorithms for computing parsing tables and subsequently with the development of a user-friendly interface in a web browser for entering grammars and displaying the resulting tables.

The analytical part describes the necessary theory, particularly the structures used to describe the principles of syntactic analysis. It also offers a detailed description of the algorithms used to compute parsing tables and compares the various algorithms according to the type of grammars. It is shown that algorithms designed for more complex classes of grammars are considerably more complex.

The practical part deals with the implementation of data structures and algorithms needed for computing parsing tables and the creation of a web interface. The web interface is created using the React JavaScript library and allows the user to enter grammars and presents the results of computations.

Keywords parsing, LL, LR, SLR, typescript, vite, npm, frontend, React

Seznam zkratek

JS	Javascipt
TS	Typescirpt
LL	Left Left (deterministická syntaktická analýza shora dolů)
LR	Left Right (deterministická syntaktická analýza zdola nahoru)
SLR	Simple LR
G	Gramatika
HTML	Hypertext Markup Language

Úvod

Tato práce se zabývá tvorbou nástroje na výuku syntaktické analýzy. Teorie syntaktické analýzy je důležitá pro zpracovávání a překlad zejména textu. Jedná se o poměrně novou teorii, byla vytvořena převážně v druhé polovině 20. století. Její pochopení může být pro začátečníka poměrně náročné.

Nástroj vytvářený v této práci bude pomáhat zejména studentům a lidem kteří chtějí pochopit syntaktickou analýzu. Ukazuje na příkladech její průběh a jaké datové struktury jsou použity.

Práce bude stavět na základech existují aplikace, která zatím nabízí jen ukázku syntaktické analýzy pro LL(1) gramatiky. Kromě LL(1) analýzy jsou v praxi velmi často užívané i analýzy, SLR(1) a LR(1), ty v současné aplikaci chybí.

Teoretická část se zabývá definováním potřebných pojmů a popisem jednotlivých analýz (LL(1), SLR(1) a LR(1)) a klíčových rozdílů mezi nimi. Praktická část se zabývá implementací výukového nástroje.

Cíle

Cílem práce je, seznámit se s LL syntaktickou analýzou a LR syntaktickou analýzou, seznámit se se stávající aplikací pro výpočet rozkladové tabulky pro LL(1) jazyky. Následně původní aplikaci rozšířit o výpočet LL(1) a LR(1) rozkladových tabulek. Nástroj bude podporovat zadávání gramatik, bude generovat tabulky a bude poskytovat ukázku výpočtu pravého respektive levého rozkladu slova z jazyka na základě vygenerovaných tabulek.

Kapitola 1
Teoretická část

1.1 Definice základních pojmů

Mezi hlavní pojmy teorie syntaktické analýzy patří symbol a abeceda. Definovat co je to symbol je obtížné, proto se v literatuře často jeho definice vynechává, nebo se jen konstatuje že symbol je prvek abecedy a abeceda je konečná množina symbolů. Takto ale nelze symbol a abecedu formálně definovat, jednalo by se o definici kruhem.

Cílem této práce není vyřešit filozofický problém definice symbolu. Pro potřeby této práce postačí, když si čtenář představí symbol jako unikátní token na vstupu algoritmu.

Následující definice jsou převzaté ze zdrojů [1] a [2]

▶ **Příklad 1.1.** příklady symbolů:

1. písmena české abecedy a, b, c, d, ...
2. slova přirozeného jazyka: pes, kočka, love (aj), Shadenfreude (nj), kultakala (fj)
3. klíčová slova programovacího jazyka: if, while, for, then, function, void, null

▶ **Definice 1.2** (Abeceda)**.** *Abeceda je neprázdná konečná množina množina symbolů.*

▶ **Příklad 1.3.** příklady abeced:

1. Binární abeceda {0, 1}
2. Latinka {a, b, c, ...}
3. Množina všech slov spisovné češtiny {být, les, jestliže, ...}

▶ **Definice 1.4** (Věta)**.** *Věta (někdy se označuje jako slovo nebo řetězec) Věta je konečná posloupnost řetězců abecedy*

1. ε *značí prázdnou větu, prázdná posloupnost symbolů*
2. Σ^* *množina všech vět (včetně prázdné) nad abecedou* Σ
3. Σ^+ *množina všech neprázdných řetězců nad* Σ

▶ **Definice 1.5** (Zřetězení)**.** *Buďte x a y dvě věty o větě z řekneme, že je zřetězením x a y pokud platí*

▶ **Definice 1.6** (Délka věty)**.** *Buď a věta, kladné celé číslo d nazveme délkou věty pokud platí* $a = a1.a2.a3. \ldots ad.$ *Délku věty značíme* $|a|$*. Pro* ε *platí* $|\varepsilon| = 0$*.*

▶ **Příklad 1.7.** Mějme abecedu Σ = {a, aa, aaa} a věty x, y, z

1. Pokud x = aaa potom |x| = 1
2. Pokud y = a.aa potom |y| = 2
3. Pokud z = a.a.a.a potom |z| = 4

▶ **Definice 1.8** (Formální jazyk nad abecedou Σ)**.** *Formální jazyk nad abecedou Σ je libovolná podmnožina všech možných řetězců*

1.2 Definice gramatik

Jedním z formalizmů používaných pro popis formálních jazyků je gramatika.

▶ **Definice 1.9** (Gramatika)**.** *Buď G uspořádaná čtveřice (Σ, N, P, S) kde:*

1. *Σ je konečná neprázdná množina terminálních symbolů*
2. *T je konečná neprázdná množina neterminálních symbolů*
3. *P je neprázdná konečná množina přepisovacích pravidel*
4. *S je počáteční neterminální symbol*

▶ **Příklad 1.10.** Mějme abecedu Σ = {a, aa, aaa} a věty x, y, z

1. Pokud x = aaa potom |x| = 1
2. Pokud y = a.aa potom |y| = 2
3. Pokud z = a.a.a.a potom |z| = 4

▶ **Definice 1.11** (Neterminální symbol gramatiky)**.** *Neterminální symbol gramatiky, zkráceně neterminál, sám o sobě nenese žádný význam. Používá se k popisu vztahů mezi terminálními symboly. (Něco jako proměnná v matematice nebo v programování.)*

▶ **Definice 1.12** (Terminální symbol gramatiky)**.** *Terminální symbol gramatiky, zkráceně terminál, je prvkem abecedy jazyka generovaného gramatikou.*

▶ **Definice 1.13** (Přepisovací pravidlo)**.** *Pravidlo je zobrazení* $(\Sigma \cup N)^*.A.(\Sigma \cup N)^* \times (\Sigma \cup N)^*$, *kde* $A \epsilon N$. *Pravidlo zapisujeme jako* $\alpha A \beta \rightarrow \gamma$ *kde* α, β, $\gamma \epsilon (\Sigma \cup N)^*$.

▶ **Definice 1.14** (Bezkontextová gramatika)**.** *Gramatika je bezkontextová právě tehdy když všechna její pravidla jsou tvaru* $A \rightarrow \gamma$, *kde* $A \epsilon N$ *a* $\gamma \epsilon (\Sigma \cup N)^*$

▶ **Definice 1.15** (Rozšířená bezkontextová gramatika)**.** *Buď G bezkontextová gramatika o gramatice G' řekneme, že je rozšířenou gramatikou G pokud platí, že G' vznikla z G přidáním pravidla* $S' \rightarrow S$, *kde S je počátečním neterminálem G a* $S' \notin S$.

1.3 Syntaktická analýza

▶ **Definice 1.16** (Levá a pravá derivace)**.** *Levá derivace je taková derivace, která v každém kroku nahrazuje vždy ten neterminál, který je v dané větné formě nejvíce nalevo. Podobně pravá derivace vždy nahrazuje neterminál nejvíce napravo.*

▶ **Definice 1.17** (Levá a pravá derivace)**.** *Levá derivace je taková derivace, která v každém kroku nahrazuje vždy ten neterminál, který je v dané větné formě nejvíce nalevo. Podobně pravá derivace vždy nahrazuje neterminál nejvíce napravo.*

▶ **Definice 1.18** (Levý rozklad). *Levý rozklad věty w v bezkontextové gramatice je G je posloupnost pravidel použitých v levé derivaci.*

▶ **Definice 1.19** (Pravý rozklad). *Pravý rozklad věty w v bezkontextové gramatice G je obrácená posloupnost pravidel použitých v pravé derivaci.*

▶ **Definice 1.20** (Syntaktická analýza). *Syntaktická analýza je proces, který pro danou bezkontextovou gramatiku G a řetězec w určí, zdali w ϵ L(G). V kladném případě též získáme syntaktickou strukturu řetězce v podobě levého či pravého rozkladu.*

Syntaktická analýza je proces hledání rozkladu slova. To znamená, že hledá posloupnost pravidel která vede k vygenerování dané věty z počátečního symbolu gramatiky. Jedna věta z jazyka generovaného gramatikou může vzniknout pomocí vícero takovýchto posloupností (díky zkracujícím pravidlům až nekonečně mnoho).

LL a LR jsou metody syntaktické analýzy které jsou deterministické. Při čtení ze vstupu vždy ví, jak pokračovat a pokud se podaří vstup přijmout vrátí vždy stejný pravý respektive levý rozklad.

1.4 LL

Ve zkratce LL první L znamená „Left“, tedy že se vstup čte zleva a druhé znamená „Leftmost“, tedy že při analýze se vždy používá levá derivace anglicky „Leftmost derivation“.

Číslo v závorkách znamená na kolik symbolů dopředu se algoritmus konstrukce tabulky dívá. Například algoritmus konstrukce tabulky LL(1) se dívá o jeden symbol dopředu.

1.4.1 Postup sestavení LL rozkladové tabulky

Rozkladová tabulka má pro každý neterminál dané gramatiky jeden řádek a má sloupce pro všechny možné věty které mohou být ve výhledu.

K stavení tabulky je potřeba vypočítat funkce first() a follow(). Nejprve si představíme funkce first a follow a potom si ukážeme jak je rozšířit na jejich k varianty.

Pro gramatiku G = (Σ, N, P, S) vypadají funkce FIRST a FOLLOW následovně.

1.4.2 FIRST1

Funkce FIRST1(α) je definovaná pro všechny větné formy, tedy pro neterminály, terminály a zřetězení terminálu a neterminálů. A jejím výsledkem je množina do, které patří všechny neterminály, které se mohou nacházet na začátku některé z vět vygenerovaných z dané větné formy. Pokud je možné přepsat větnou formu na ϵ patří do množiny FIRST1(α) i ε.

Formálně:

$$FIRST(\alpha) = \{\, a \in (\Sigma \cup \varepsilon) \mid \alpha \rightarrow^* a\beta \} \; kde \; \alpha, \beta \in (\Sigma \cup N)^*$$

1.4.3 FOLLOW1

Funkce FOLLOW1(A) je definovaná jen pro neterminály. Jejím výsledkem je množina do které patří všechny terminály které se mohou nacházet přímo za neterminálem A v libovolné větné formě vzniklé derivováním započatým z počátečního symbolu. Pokud můžeme derivacemi získat větnou formu ve které se nachází A jako poslední symbol patří do množiny FOLLOW1(A) i ε.

Formálně:

$$FOLLOW(A) = \{ a \in (\Sigma \cup \varepsilon) \mid S \rightarrow^* \alpha A \beta \wedge a \in FIRST(\beta) \}$$
$$kde\ \alpha, \beta \cup (\Sigma \cup N)^* \text{ a S je počáteční neterminál gramatiky}$$

Abychom sestavili rozkladovou tabulku, budeme potřebovat spočítat FISRT pro všechny pravé strany pravidel v gramatice G.

1.4.4 Algoritmus výpočtu funkce FIRST

Definice FIRST je poměrně přímočará, nemusí z ní ale být na první pohled jasné jak počítá.

1. $FIRST(\varepsilon) = \{\varepsilon\}$
2. $FIRST(a) = \{a\}\ kde\ a \in \Sigma$
3. $FIRST(A) = first(A)\ kde\ A \in N$
4. $FIRST(A\alpha) = (first(A) - \{\varepsilon\}) \cup FIRST(\alpha)\ kde\ A \in N,\ \varepsilon \in first(A)$

Algoritmus výpočtu first(A) pro všechna A ∈ N

1. pro všechny A ∈ N first(A) = ∅
2. pro všechna pravidla A → α first(A) := first(A) ∪ FIRTS(α)
3. opakuj krok 2, pokud se alespoň jedna množina first(A) změnila

[3]

1.4.5 Algoritmus výpočtu funkce FOLLOW

Z definice FOLLOW plyne, že nejprve potřebujeme spočítat FIRST. FIRST již máme, můžeme se tedy podívat na výpočet funkce FOLLOW.

1. $FOLLOW(S) = \{\varepsilon\}$
2. $FOLLOW(A) = \emptyset$ pro všechna $A \in (N - \{S\})$
3. Pro všechna pravidla:
   a. Má-li pravidlo tvar X → $\alpha Y \beta$, pak FOLLOW(Y) := FOLLOW(Y) ∪ $FIRST(\beta) - \{\varepsilon\}$
   b. Má-li pravidlo tvar X → $\alpha Y \beta$ a $\varepsilon \in FIRST(\beta)$ *pak* $FOLLOW(Y) := FOLLOW(Y) \cup FOLLOW(X)$
4. Opakuj krok 3, pokud se alespoň jedna množina FOLLOW(A) změnila

[3]

Algorithm 1 Výpočet množin FIRST pro pravé strany pravidel a neterminály

```
1: procedure vypočtiMnožinyFIRST(G)
2:     for all neterminál ∈ G.neterminály do        ▷ Nastav first pro všechny neterminály jako prázdnou množinu
3:         neterminál.first ← {}
4:     end for
5:     for all terminál ∈ G.terminál do        ▷ Nastav first pro každý terminál jeho first jako jednoprvkovou množinu obsahující právě tento terminál
6:         terminál.first ← {terminál}
7:     end for
8:     for all pravidlo ∈ G.pravidla do        ▷ Nastav first pro všechna pravidla jako prázdnou množinu
9:         pravidlo.first ← {}
10:    end for
11:    změna ← True                    ▷ Flag sledující, jestli se nějaká množina first změnila
12:    while změna do
13:        změna ← False                                                    ▷ Resetuj flag
14:        for all pravidlo ∈ G.pravidla do
15:            if vypočtiFIRSTProPravidlo(pravidlo) then
16:                změna ← True
17:            end if
18:        end for
19:    end while
20: end procedure

21: procedure vypočtiFIRSTProPravidlo(pravidlo)
22:    změna ← False
23:    for all symbol ∈ rule.praváStrana do
24:        pravidlo.first = pravidlo.first ∪ symbol.first
25:        if Přibylo něco nového do pravidlo.first then
26:            změna ← True
27:            pravidlo.leváStrana.first = pravidlo.leváStrana.first ∪ symbol.first  ▷ Gramatika je bezkontextová, levou stranu pravidel tvoří jeden neterminál
28:        end if
29:        if symbol.first neobsahuje ε then
30:            break
31:        end if
32:    end for
33:    return změna
34: end procedure
```

1.4.6 FIRSTk

Funkce FIRSTk(α) je rozšířením funkce FIRST1(α). Funkce FIRSTk(α) je rovněž definovaná pro všechny větné formy, tedy pro neterminály, terminály a zřetězení terminálů a neterminálů. Ale jejím výsledkem je množina vět délky k případně kratší, pokud se nachází na konci vstupu. Jedná se o věty, které mnohou být předponou věty vzniklé derivací z větné formy α.

Formálně:

$$\begin{aligned} FIRST_k(\alpha) = &\{ a_1 a_2 \dots a_k \mid a_i \in T, i \in \{1, \dots, k\}, \alpha \to^* a_1 a_2 \dots a_k \beta\} \cup \\ &\{ a_1 a_2 \dots a_{k-1} \varepsilon \mid a_i \in T, i \in \{1, \dots, k-1\}, \alpha \to^* a_1 a_2 \dots a_{k-1}\} \cup \dots \cup \\ &\{ \varepsilon_1 \varepsilon_2 \dots \varepsilon_k \mid \alpha \to^* \varepsilon\}, \ kde\ \alpha, \beta \in (\Sigma \cup N)^* \end{aligned}$$

[4]

1.4.7 Sestaveni tabulky

Po sestrojení množin FIRST a FOLLOW lze sestavit rozkladovou tabulku.

Rozkladová tabulka pro LL(1) má za každý neterminál a ϵ jeden sloupec a za každý terminál jeden řádek. Buňky v tabulce obsahují pravidla, která budou použita pro expanzi pro danou kombinaci terminálu a neterminálu. Buďte E neterminál a a terminál, potom v buňce T[E, a] bude pravidlo P $A \to w$ právě tehdy, když a je ve FIRST(α) nebo ε je ve FIRST(α) a a je ve FOLLOW(A).

1.5 LR

LR je zkratka, kde L znamená, že čteme vstup zleva doprava a R znamená, že v každé derivace nahrazujeme neterminál nacházející se nejvíce vpravo ve větné formě. Číslo v závorkách znamená, na kolik symbolů dopředu se algoritmus konstrukce tabulky dívá.

Existuje více typů LR analýzy. Tato bakalářská práce se zaobírá třemi typy. Konkrétně jsou to LR(0), SLR(1) a LR(1).

1.5.1 LR(0)

Metoda LR(0) je nejjednodušší metodou z uvedených. Během konstrukce tabulek tabulek se nepoužívá výhled dopředu. Výsledný syntaktický analyzátor se skládá ze dvou tabulek. Z tabulky přechodů (goto table) a z tabulky akcí (action table).

K sestrojení tabulek je zapotřebí kanonická množina prvků LR(0).

▶ **Definice 1.21** (Položka LR(0)). *LR(0) položka je uspořádaná dvojice přepisovacího pravidla a čísla x. Kde x značí pozici v pravé straně pravidla, platí tedy, že* $0 \leq x \leq$ *délka pravé strany.*

[4]

Položka se obvykle značí jako pravidlo uprostřed jehož pravé strany je tečka, např $A \to \beta.B\gamma$, $A \to .\alpha$ nebo $A \to \alpha$.

1.5.1.1 Pozice

Pozice říká jaká část pravé strany byla již zpracována. Všechno nalevo od pozice je na zásobníku. O symbolu napravo se předpokládá, že bude další na vstupu. Pokud pozice je na konci pravidla, znamená to, že máme na zásobníku celou pravou stranu pravidla a můžeme jí zredukovat.

1.5.1.2 Uzávěr

Když pozice je před neterminálem E v LR(0) prvku, znamená to, že očekáváme na vstupu symbol na který se může neterminál E přepsat. Do uzávěru se vloží všechna možná pravidla, jejichž levá strana je rovna E.

1.5.1.3 Výpočet uzávěru (closure) množiny M

Pokud M je množina položek pro gramatiku G, potom uzávěr M je množina položek konstruovaná pomocí dvou následujících pravidel.

1. Přidáme všechny položky z množiny M do uzávěru M
2. Pokud $A \to \alpha B \beta$ je v uzávěru M a $B \to \gamma$ je pravidlo, potom přidáme položku $B \to .\gamma$ do M (pokud se již v množině nenachází). Aplikujeme toto pravidlo dokud nemohou být žádné položky přidány do uzávěru množiny M.

Closure se používá k rozšíření množiny LR(0) položek. Zajišťuje, že jsou přidána všechna možná pravidla, která lze v dané fázi výpočtu využít. Za každý prvek LR(0) v množině, ve kterém je tečka bezprostředně před neterminálem se přidají všechna pravidla pro daný neterminál.

1.5.2 Operace goto

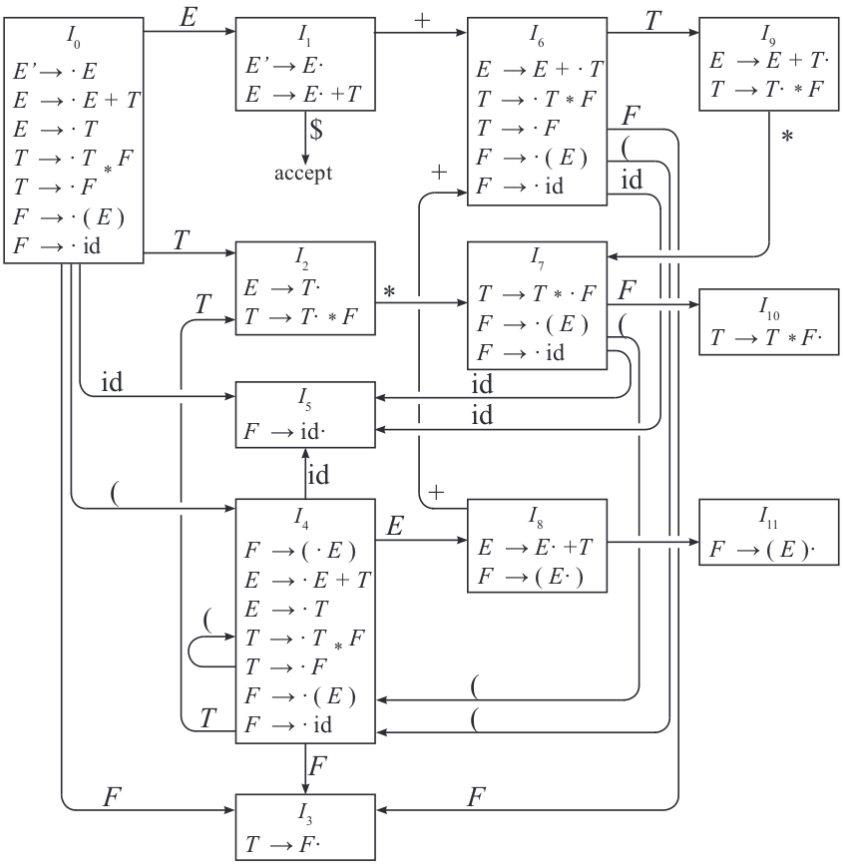
Closure symbolizuje stav ve kterém se syntaktický analyzátor může nacházet. Pro přechod mezi jednotlivými stavy slouží operace goto.

▶ **Definice 1.22** (goto)**.** *Buď M množina prvků reprezentující stav syntaktického analyzátoru a x symbol (neterminál nebo terminál), goto vytvoří množiny reprezentující nové stavy pomocí následujících dvou kroků*

1. *Přidá všechny položky, ve kterých se nachází tečka před x*
2. *Aplikuje closure*

1.5.3 Sestavení grafu přechodů syntaktického analyzátoru

Pomocí operací closure a goto se dá sestavit graf reprezentující stavy syntaktického analyzátoru LR(0) a přechody mezi nimi.



(a) Množiny položek LR(0), obrázek z dragonbook strana 244

[5]

1.5.4 Konstrukce tabulek pro LR(0)

Syntaktický analyzátor LR(0) používá dvě tabulky, tabulku přechodů (goto) a tabulky akcí (action).

1.5.4.1 Tabulka přechodů

Tabulka přechodů bude mít za každý stav (unikátní množinu LR(0)) položek jeden řádek a za každý symbol z gramatiky jeden sloupec. Tabulku lze vytvořit jednoduše průchodem grafu.

1.5.4.2 Tabulka akcí

Syntaktický analyzátor LR(0) provádí tři typy akcí posun, redukci a přijetí, kde přijetí je redukce podle pravidla, jehož přidáním vznikla z původní gramatiky, rozšířená gramatika. Tabulka akcí pro LR(0) má pouze jeden sloupec a má jeden řádek za každý stav analyzátoru, má tedy stejný počet řádků jako tabulka přechodů. Do sloupce se přidají akce za každý stav následovně:

1. Za každou LR(0) položku ve tvaru $A \to \alpha.$ se přidá redukce podle pravidla $A \to \alpha$, kde $A \in N a\ \alpha \in (\Sigma \cup N)^*$

2. Za každou LR(0) položku ve tvaru $A \to \alpha.b\gamma$ kde $\alpha, \gamma \in (\Sigma \cup N)^*$ a $b \in \Sigma$, přidáme akci posun

Pokud se v tabulce octnou dvě a více akcí, nastává tzv. konflikt. Když se analyzátor dostane do bodu, kdy má vyžít buňku tabulky kde je více akcí, tak neví kterou z nich má použít. Z popsaného postupu konstrukce tabulky plyne, že může nastat konflikt typu posun–redukce nebo redukce–redukce.

1.5.5 SLR(1)

LR(0) syntaktická analýza je poměrně „slabá“, dá se použít je pro některé jednoduché gramatiky. Pro většinu gramatik budou u LR(0) vznikat konflikty v tabulce akcí. SLR(1) syntaktická analýza je „silnější“, množina gramatik pro které je použitelná je větší.

1.5.6 Konstrukce tabulek pro SLR(1)

Tak jako LR(0) i syntaktický analyzátor SLR(1) používá dvě tabulky, tabulku přechodů (goto) a tabulky akcí (action).

1.5.6.1 Tabulka přechodů

Tabulka přechodů vypadá stejně jako u LR(0) a tvoří se úplně stejně.

1.5.6.2 Tabulka akcí

Tak jako tabulka akcí pro LR(0) i tabulka akcí pro SLR(1) má jeden řádek za každý stav, liší se však počtem svých sloupců. Tabulka pro SLR(1) má sloupec za každý neterminální symbol gramatiky a jeden další za ε. Buňky tabulky se vyplňují podobně jako v LR(0), s tím rozdílem, že musí být rozhodnuto, do kterého sloupce má být akce zařazena.

Pokud se jedná o posun zapíše se tato akce za každý přechod z daného stavu na do sloupce pro příslušný neterminál. Analyzátor se dívá o jeden symbol dopředu a posun se provádí pokud je tečka před neterminálem proto je logicky ve výhledu neterminál na který se provádí posun. Redukce se provádí, pokud je tečka na konci položky, redukce podle daného pravidla se tak doplní do sloupců které odpovídají neterminálům, které se mohou nacházet ve výpočtu těsně za neterminálem na který se redukuje. To od povídá přesně funkci FOLLOW. Je-li v množině položek položka $A \to \alpha.$, přidá se do akce redukce podle pravidla $A \to \alpha.$ do sloupců odpovídajících neterminálům z FOLLOW(A).

1.5.7 LR(1)

Syntaktická analýza LR(1) je nejsilnější ze všech tří zde uvedených způsobů. Tvorba tabulek probíhá identicky jako u SLR(1) s tím, že místo LR(0) položek (které se požívají v LR(0) a SLR(1)) požívají položky LR(1).

▶ **Definice 1.23** (Položka LR(1)). *LR(1) položka je uspořádaná trojice přepisovacího pravidla, čísla x a výhledu. Kde x značí pozici v pravé straně pravidla, platí tedy, že $0 \leq x \leq$ délka pravé strany. Výhledem je terminální symbol který bude následovat dále za položkou ve výpočtu, nebo ε.*

[4]

Obvykle se značí jako $[A \to \alpha.\beta, a]$ kde $A \in N, \alpha, \beta \in (\Sigma \cup N)^*$ a $a \in (\Sigma \cup \epsilon)$.

Tím že každá položka obsahuje oproti LR(0) navíc i výhled, vzniká mnohem více unikátních položek a to vede k daleko většímu počtu unikátních stavů ve kterých se může syntaktický

analyzátor nacházet. Díky tomu LR(1) funguje pro větší množinu gramatik než LR(0) a SLR(1), na druhou stranu, ale výsledné tabulky (jak tabulka přechodů tak i tabulka akcí) budou větší, což může být pro některé rozsáhlejší gramatiky problém.

Kapitola 2

Existující řešení

2.1 Současná aplikace pro výuku předmětu BI-PJP na ČVUT FIT

Existující aplikace která zatím nabízí jen ukázku syntaktické analýzy pro LL(1) gramatiky. Kromě LL(1) existují i LR(0), SLR(1) a LR(1) analýzy ty v současné aplikaci chybí a jejich přidáním ze zabývá tato práce.

1. Odkaz na nasazenou aplikaci: https://pages.fit.cvut.cz/peckato1/parsingtbl/
2. Odkaz na zdrojový kód (gitlab): https://gitlab.fit.cvut.cz/peckato1/parsingtbl

2.2 Aplikace na generování rozkladových tabulek pro LL(k) z VUT FIT

Zajímavá ukázka trochu jiného přístupu k LL(k) analýze. Požívá inovativní algoritmus, který umožňuje aby byl vstup čten po jednom symbolu a odstraňuje některé nevýhody LL(k) pro k větší než 1.

1. Odkaz na nasazenou aplikaci: http://www.fit.vutbr.cz/ ikocman/llkptg/
2. Odkaz na zdrojový kód: https://github.com/rkocman/LLk-Parsing-Table-Generator

   Použitá teorie je popsána například v [4]

Kapitola 3
Implementace

3.1 Návrh datových struktur

Nejprve jsem začal návrhem potřebných datových struktur.

Vytvořil jsem abstraktní třídu GrammarSymbol, která má dvě konkrétní implementace. Terminal a NonTerminal.

Listing 3.1 třída GrammarSymbol

```
export abstract class GrammarSymbol {
    constructor(public readonly name: string) {}

    equals(other: GrammarSymbol) {
        return this.name === other.name;
    }

    abstract getFirst(): SetOfWords;
    abstract getFollow(): SetOfWords;
    abstract canChangeToEpsilon(): boolean;
    abstract getRules(): Rule[];
    abstract getValue(): Terminal | undefined;
}
```

GrammarSymbol má jeden atribut *name* unikátní jméno, řetězec ze kterého symbol vznikl. Funkce getFirst() a getFollow() vracejí množiny slov obecně délky k. CanChangeToEpsilon() vrátí true, pokud je daný symbol možné přepsat na epsilon. getRules() vrací pole pravidel které mají daný symbol na levé straně, Protože gramatika je bezkontextová je na straně právě vždy jeden neterminál. Value vrátí hodnotu symbolu, vizte dále.

Listing 3.2 třída Terminal

```
export class Terminal extends GrammarSymbol {

    constructor(public name: string) {
        super(name);
        this.firstSet.add(new Word([this]));
    }

    getFirst(): SetOfWords { return this.firstSet; }
    getFollow(): SetOfWords { return this.firstSet; }
    canChangeToEpsilon(): boolean { return false; }
    getRules(): Rule[] { return []; }
    getValue(): Terminal | undefined { return this; }
    private firstSet: SetOfWords = new SetOfWords();
}
```

Terminál je konkrétní implementací třídy GrammarSymbol. Každá instance třídy terminal má jako atribut množinu slov s jedním prvkem, kterým je právě onen neterminál, kterou vrací ve svojí implementaci funkce getFirst(). FOLLOW by pro neterminál nemělo být definováno, zde je getFollow() pouze pro dodržení rozhraní GrammarSymbol. Neterminál se nikdy nemůže přepsat na ε proto canChangeToEpsilon() vždy vrací false. Neterminál se nikdy nevyskytuje na pravé straně pravidla, proto getRules() vrací prázdné pole.

Listing 3.3 třída NonTerminal

```
export class NonTerminal extends GrammarSymbol {

    constructor(public name: string) {
        super(name);
    }

    getFirst(): SetOfWords { return this.first; }
    getFollow(): SetOfWords { return this.follow; }
    canChangeToEpsilon(): boolean { return this.epsilon }
    setCanChangeToEpsilon(epsilon: boolean): void { this.epsilon = epsilon; }
    getRules(): Rule[] { return this.rules; }
    getValue(): Terminal | undefined { return undefined; }

    public rules: Rule[] = [];
    public rightSidesOfRules: RightSideOfRule[] = [];
    public first: SetOfWords = new SetOfWords();
    public follow: SetOfWords = new SetOfWords();
    public followedByEOF: boolean = false;
    private epsilon: boolean = false;
}
```

Třída NonTerminal je taktéž instancí třídy GrammarSymbol. Má atributy first a follow které slouží k akumulování množin vět reprezentujících FIRST a FOLLOW pro daný neterminál, vizte dále výpočet first a follow. Atribut followedByEOF je nastaven na true, pokud se daný neterminál

může vyskytnout na konci nějaké větné formy vygenerované z počátečního neterminálu.

Listing 3.4 třída Grammar

```
export class Grammar {
    constructor(
        public nonTerminals: Map<string, NonTerminal>,
        public terminals:    Map<string, Terminal>,
        public rules: Rule[],
        public startingNonTerminal: NonTerminal,
    ) {}
}
```

Grammar je třída která reprezentuje celou gramatiku. Atribut *startingNonTerminal* je

3.2 Implementace FIRST a FOLLOW

Z předchozích ukázek datových struktur vyplývá, že pro třídu NonTerminal je nutné dopočítat její atributy. K tomu slouží funkce first a follow, které berou jako argument instaci třídy Grammar a dopočítají atributy first a follow pro neterminály a pravidla.

Součástí implementace jsou i varianty rozšířené varianty funkcí first a follow, firstK a followK které by bylo do budoucna možné použít pro

3.3 Immutable JS

Při vyváření syntaktických analyzátorů LL je potřeba použít strukturu která implementuje množinu slov a v LR je potřeba množina LR(0) respektive LR(1) prvků. Základní šablona pro množiny v javasciptu i typescriptu používá na porovnání prvků *SameValueZero* algoritmus, vlastní algoritmus pro porovnání prvků si uživatel nemůže nastavit.

SameValueZero funguje podobně jako operátor ===, některé základní datové typy jako number, string, atd. porovnává na základě hodny. Uživatelské objekty vytvořené na základě tříd porovnává jen na základě reference. V současné době existuje návrh [6] na to, aby vestavěné typy *Record* a *Tuple* (které jsou immutable) byly porovnávány na základě jejich jejich hodnoty. Návrh však nebyl stále implementován a není ani jasné, jestli v budoucnu implementován bude.

Standardní javasciptovou množinu tedy nešlo použít, kvůli tomu jak jsou v aplikaci navrženy datové struktury. V praxi se tento problém obvykle řeší následujícími způsoby.

1. Použití vlastní datové struktury
2. Serializace objektů do typu *string*
3. Použití knihovny třetí strany

Během vývoje aplikace byly vytvořeny třídy SetOfWords a SetOfItems. SetOfWords jako abstrakce pro množiny slov (zejména pro first a follow) a SetOfItems jako abstrakce pro množiny položek (LR(0) i LR(1)). Jejich účelem je skrýt konkrétní implementaci množin, tak aby, pokud se jí někdo rozhodne změnit, ji mohl změnit pouze na dvou místech v programu.

Ze začátku vývoje byla využívána vlastní implementace AVL stromu, ale později byla nahrazena za použití knihovny immutable.js. Immutable.js je v praxi velmi často používaná ověřená knihovna, poskytuje immutable datová struktury a byla i inspirací pro výše zmíněný návrh. Díky immutablilitě, která pro tuto aplikaci není potřeba, je o trochu pomalejší, ale protože tento bude počítat spíše menší tabulky, nemělo by pomalejší implementace množin působit problémy.

3.4 LR0Graph, LR1Graph

Množiny stavů syntaktických analyzátorů LR(0) a LR(1) jsou společně s přechody mezi nimi jsou modelovány jako orientované grafy a které lze získat voláním funkcí getLR0Graph a getLR1Graph. Průchodem těchto grafů se potom vytváří přechodové a tabulky a tabulky akcí.

3.5 Tabulky

V aplikaci jsou reprezentované třídami LL1ParsingTable, LR0ActionTable, LR1ActionTable, LRGoToTable které jsou vnitřně implementovány jako klasická typscriptová mapa a navenek prezentují srozumitelné rozhraní, tak aby se daly snadno použít k syntaktické analýze řetězců, které budou uživatelé zadávat na vstupu. Vnitřně je potom používají react komponenty, které je celé projdou a zobrazí uživateli výslednou tabulku.

3.6 React komponenty

Výsledné tabulky, které jsou zobrazovány uživateli jsou react komponenty, které jsou vytvářeny pomocí výše zmíněný struktur a algoritmů.

V této kapitole jsou ukázky výsledných vygenerovaných tabulek.

```
S -> a A a
S -> b A b
S -> a B b
S -> b B a
A -> a
B -> a
```

(a) Textový vstup uživatele

Uživatel zadá textový vstup ve stejné formě jako v původní aplikaci a potom si zvolí, jaké tabulky si zobrazí. Na výběr má LL(1), LR(0), SLR(1) a LR(1). Do dalšího textového vstupu pak muže zadat slovo, pro které analyzátor pokusí, za pomoci vygenerovaných tabulek najít odpovídající rozklad.

LL(1) first and follow sets

	Rule (A ⟶ α)	First(α)	first[A]	Follow(A)
1	S' ⟶ S	a b	a b	ε
2	S ⟶ a A a	a	a b	ε
3	S ⟶ b A b	b		
4	S ⟶ a B b	a		
5	S ⟶ b B a	b		
6	A ⟶ a	a	a	a b
7	B ⟶ a	a	a	a b

(b) Tabulka first follow

Pokud se podaří správně načíst zadanou gramatiku zobrazí komponenta ukazující množiny first a follow pro neterminály a pravidla.

LR goto

Symbols	S	A	B	a	b
# -1	S 1			a 1	b 2
A 1				a 2	
A 2					b 3
B 1					b 1
B 2				a 4	
S 1					
a 1		A 1	B 1	a 3	
a 2					
a 3					
a 4					
a 5					
b 1					
b 2		A 2	B 2	a 5	
b 3					

LR(1) Action

Symbols	a	b	ε
# -1	S	S	
A 1	S		
A 2		S	
B 1		S	
B 2	S		
S 1			R: 0
a 1	S		
a 2			R: 1
a 3	R: 5	R: 6	
a 4			R: 4
a 5	R: 6	R: 5	
b 1			R: 3
b 2	S		
b 3			R: 2

(c) Tabulky goto a action

Tabulka goto ukazuje přechody syntaktického analyzátoru a vedle tabulka action ukazuje jaké akce se mají v dané fázi analýzy použít.

LR(1) Parser

Step	Input	Stack	Right parse	Next operation
1	b a b	# -1		shift
2	a b	# -1 b 2		shift
3	b	# -1 b 2 a 5		reduce 6
4	b	# -1 b 2 A 2	6	shift
5		# -1 b 2 A 2 b 3	6	reduce 3
6		# -1 S 1	3, 6	Accept
7		# -1	1, 3, 6	

(d) Ukázka běhu LR analyzátoru

Synktaktický analyzátor, v tomto případě LR, je taktéž implementován jako react komponenta. Je to tabulka, která ve svých jednotlivých řádcích ukazuje analýzy a v tomto případě slovo přijal a v posledním řádku je tak pravý rozklad slova.

3.7 Testování

K testování byl využit framework jest. Jest v praxi hojně využívaný framework. Umožňuje spouštět testy paralelně a nabízí jednoduché a přehledné API. Testovány jsou především výše zmíněné algoritmy a některé pomocné třídy jako setOfItems a setOfWords.

3.8 Vite

V projektu je požit typescipt a react obě tyto technologie umožňují snazší vývoj a dělají výsledný kód čitelnější. Aby mohl klient (prohlížeč) spouště výslednou aplikaci je potřeba přeložit (traspilovat) výsledný kód do javascriptu, který už nebude obsahovat konstrukty s typescriptu a reactu. Toho ze docílit více způsoby, ze byl zvolen právě vite.

Mezi hlavní výhody Vite patří jednoduchá konfigurace, používá princip *convention over configuration*, většina nastavení už je od začátku nastaven správně a není je třeba měnit. Speciálně pro menší aplikace jako je tato není většinou třeba téměř nic nastavovat. Dále vite poskytuje vlastní vestavný server s podporou HMR (Hot Module Replacement). HMR je technologie která umožňuje provádět změny v aplikaci za běhu bez nutnosti zastavit server a kompilovat celý projekt, tím umožnuje plinulý vývoj aplikace na localhostu.

3.9 Možná rozšíření

Nástroj by bylo možné rozšířit o LALR(1) analýzu která je v praxi často používaná. Dále by bylo užitečné, přidat vizualizaci množin LR položek, aby si studenti mohli lépe představit proces tvorby tabulek.

Kapitola 4
Závěr

V analytické části práce byla rozebrána teorie a struktury, které jsou klíčové pro porozumění principů syntaktické analýzy. Dále byly popsány algoritmy pro výpočet rozkladových tabulek a provedeno jejich srovnání podle typu gramatik. Z tohoto srovnání vyplývá, že algoritmy určené pro složitější třídy gramatik jsou značně náročnější.

V praktické části práce byly implementovány datové struktury a algoritmy nezbytné pro výpočet rozkladových tabulek a vytvořeno webové rozhraní. Webové rozhraní, postavené pomocí javascriptové knihovny React, umožňuje uživatelům zadávat gramatiky a prezentuje výsledky výpočtů.

Algoritmy ze zadání LL(1) a LR(1) se podařilo naimplementovat a otestovat. Webové rozhraní funguje, zobrazuje rozkladové tabulky a ukazuje průběh syntaktické analýzy pro slova z jazyka generovaného zadanou gramatikou. Zadání páce se tedy podařilo splnit.

Bibliografie

1. HOLUB, Jan. *Přednáška BI-AAG čílo 1, Základní pojmy* [online]. 2023. [cit. 2024-04-29]. Dostupné z: `https://courses.fit.cvut.cz/BI-AAG/lectures/bi-aag-01-zakladni_pojmy.pdf`. Additional Information.
2. ŠESTÁKOVÁ, Eliška. *Automaty a gramatiky: sbírka řešených příkladů*. Thákurova 1, 160 41 Praha 6: Česká technika - nakladatelství ČVUT, 2017. ISBN 978-80-01-06306-4.
3. JANOUŠEK, Jan. *Přednáška BI-PJP čílo 4* [online]. 2020. [cit. 2024-04-29]. Dostupné z: `https://courses.fit.cvut.cz/BI-PJP/lectures/files/pjplecture2_2020.pdf`. Additional Information.
4. KOLÁŘ, Dušan. *Zásobníkové automaty: Další rozšíření a transformace*. Brno, CZ: Faculty of Information Technology VUT, 2005.
5. *Compilers: principles, techniques and tools. Second edition.* Boston: Addison Wesley, 2007. ISBN 0-321-48681-1.
6. ROBIN RICARD Rick Button, Nicolò Ribaudo. *JavaScript Records and Tuples Proposal* [online]. 2023. [cit. 2024-05-02]. Dostupné z: `https://github.com/tc39/proposal-record-tuple#usage-in-mapsetweakmapweakset`. It is possible to use a Record or Tuple as a key in a Map, and as a value in a Set. When using a Record or Tuple here, they are compared by value.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF