Projektová dokumentace

Překladač imperativního jazyka IFJ22 do IFJcode22

Tým: xpoliv07

Varianta: TRP

Michael Polívka	**xpoliv07**	**25%**
Prokop Schield	xschie03	25%
Jakub Lukáš	xlukas18	25%
Martin Medřík	xmedri01	25%

5. 12. 2022

studovna FIT VUT

1. Úvod

Cílem projektu byla implementace funkčního překladače v jazyce C, který bude překládat kód vstupního jazyka IFJ22 do tříadresného kódu IFJcode22. IFJ22 je upravená (nekompatibilní) podmnožina jazyka PHP8. Při implementaci jsme dbali a dodržování zásad objektově orientovaného programování.

Rozšíření: BOOLTHEN. CYCLES, FUNEXP

2. Návrh implementace

2.1. Lexikální analyzátor

Soubor: scanner.c

Lexikální analýza je implementovaná podle FSM na *obr. 1.*
Jeho hlavním úkolem je analýza kódu ze vstupu a hledání lexémů, které využívá syntaktická analýza. Vrací vždy token, jehož struktura je definovaná v *singleton.h.* Hodnoty, které obsahuje, jsou výsledky po FSM. Ten je implementovaný funkcí *lexer_get_token.*

2.2 Syntaktický analyzátor

Soubor: parser.c

Syntaktický analyzátor přijímá utf-8 stream a pomocí lexikálního analyzátoru jej přeměňuje na tokeny, ze kterých postupně skládá syntaktický strom. Interně využívá precedenční analýzu a rekurzivní sestup. Ze zadání vybočuje zejména tak, že přiřazení pokládá za jinak standardní operaci, která však musí mít v L-hodnotě proměnnou.

2.3 Sémantický analyzátor

Soubor: semantic.c

Většina sémantické kontroly probíhá při samotném generování. Před generováním prvotní kontroly v souboru *semantic.c* prochází strom generovaný parserem a hledá v něm sémantické chyby, které lze odhalit před generováním. Jako jsou třeba špatně zadané parametry při volání funkcí. Určuje navíc typ konstant s neurčitým datovým typem. Ostatní kontroly probíhaní až za běhu cílového kódu.

2.4 Generátor kódu

Soubory: generator.c, buildins.c

Vstupem generátoru je syntaktický strom, výstupem je cílový jazyk IFJcode22.

Generátor kódu se stará o správnost datových typů, a to implicitním vygenerováním převodníku. Tento vygenerovaný kód slouží k přetypování, aby byl možný například součet booleovského True a hodnoty null. Také obsahuje funkce pro aritmetické operace a operace porovnání.

Generátor také myslí na vestavěné funkce, u kterých byl kladen důraz na využití zásobníku a funkcí s ním spojených.

3. Práce v týmu

Na projektu jsme začali pracovat v říjnu. Práce probíhala v následujícím pořadí. Lexikální analyzátor, syntaktický analyzátor a sémantický analyzátor na kterých se pracovalo paralelně. Jako poslední byl implantovaný generátor.

Schůzky nejprve probíhali prostřednictvím online služeb, ale postupně jsme všichni začali upřednostňovat schůzky osobní, nejčastěji v knihovně či studovně FIT .

Ke sdílení souborů jsme používali GitHubový repositář.

3.1. Rozdělení práce v týmu

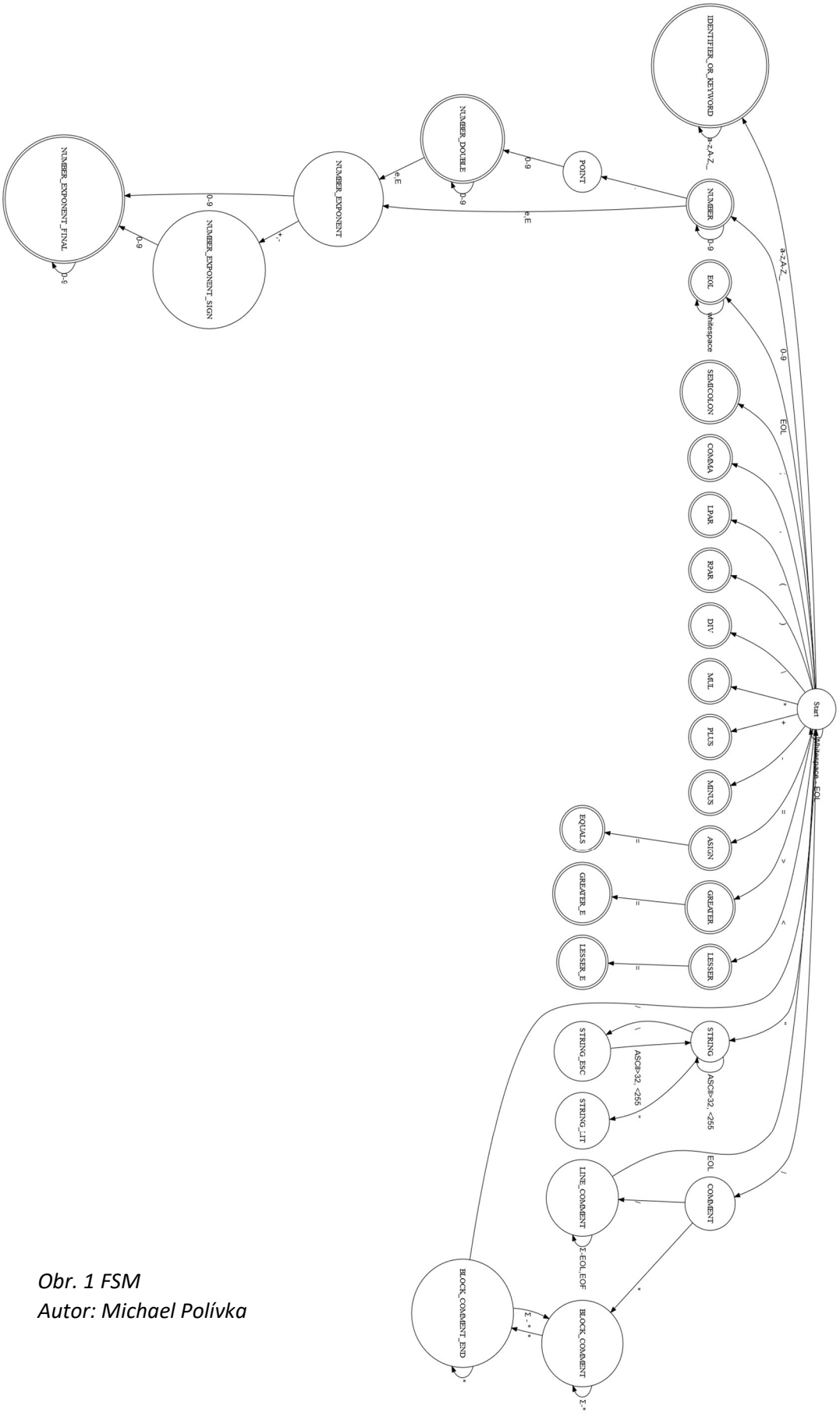
Michael Polívka: Organizace práce, lexikální analyzátor, generátor
Prokop Schield: syntaktický analyzátor, boilerplate, tabulka symbolů
Jakub Lukáš: sémantický analyzátor, návrh lexikálního analyzátoru, návrh gramatiky
Martin Medřík: buildin a podpůrné funkce v IFJcode22, dokumentace

4. LL Gramatika

ll grammar for IFJ22 compiler

1. <prog> -> <head> <main> <end>
2. <main> -> <fnc_dec> <main>
3. <main> -> <code> <main>
4. <main> -> epsilon
5. <head> -> <?php declare(strict_types=1);
6. <end> -> ?>
7. <end> -> EOF
8. <assignment> -> $ id = <expression>;
9. <expression> -> $ id
10. <expression> -> const
11. <expression> -> id(<fnc_arguments>)
12. <expression> -> <expression> operator <expression>
13. <fnc_dec> -> function id(<fnc_params>) type { <code> <return> <code> }
14. <fnc_params> -> epsilon
15. <fnc_params> -> type $ id <fnc_params2>
16. <fnc_params2> -> epsilon
17. <fnc_params2> -> , type $ id <fnc_params2>
18. <return> -> return <expression>;
19. <void_fnc_dec> -> function id(<fnc_params>) void {<code> <void_return> <code>}
20. <void_return> -> epsilon
21. <void_return> -> return; 22. <fnc_arguments> -> epsilon
23. <fnc_arguments> -> <expression> <fnc_arguments2>
24. <fnc_arguments2> -> epsilon
25. <fnc_arguments2> -> , <expression> <fnc_arguments2>
26. <if_statement> -> if (<expression>) { <code> } <else>
27. <else> -> "epsolon"
28. <else> -> else { <code> }
29. <code> -> while (<expression>) { <code> } <code>
30. <code> -> for (; ;) { <code> } <code>
31. <code> -> epsilon
32. <code> -> <assignment> <code>
33. <code> -> <fnc_call> <code>
34. <code> -> <if_statement> <code>

5. Automat konečných stavů



Obr. 1 FSM
Autor: Michael Polívka

6. Precedenční tabulka

	* /	+ - .	< > <= >=	=== !==	(	)	i	$
* /	>	>	>	>	<	>	<	>
+ - .	<	>	>	>	<	>	<	>
< > <= >=	<	<	>	>	<	>	<	>
=== !===	<	<	<	>	<	>	<	>
(	>	>	<	<	<		<	>
)	<	<	>	>		>		
i	>	>	>	>	>	>		>
$	<	<	<	<	<		<	