Formální jazyky a překladače

Přednášky:

-Typy překladačů, základní struktura překladače
-Regulární gramatiky, konečné automaty a jejich využití v lexikální analýze
-Úvod do syntaktické analýzy, metoda rekurzivního sestupu
-Překlad příkazů
-Zpracování deklarací
-Přidělování paměti
-Interpretační zpracování
-Generování cílového kódu
-Vlastnosti bezkontextových gramatik
-Deterministická analýza shora dolů
-LL(1) transformace
-Deterministická analýza zdola nahoru
-Formální překlady

Formální jazyky a překladače

Cvičení:

-Opakování teoretického základu
-Generátor lexikální analýzy - LEX
-Jazyk PL0 a jeho překladač
-Rozšíření konstrukcí jazyka PL0, zadání individuálních úloh
-Příklady LL gramatik
-Příklady LR gramatik, ukázky práce s generátorem YACC

Formální jazyky a překladače

Zápočet:
Udělen na základě referátu a předvedení zadané modifikace překladače PL0
Zkouška:
Písemná forma - 2 příklady (s použitím vlastní literatury) a otázky (bez použití literatury). Výsledky budou spolu s termíny pro reklamaci, zápis do indexu či ústní přezkoušení zveřejněny na webu a nástěnce. Po uplynutí termínu bude výsledek zapsán do databáze známek i v případě, že student nepředložil index k zápisu.

Formální jazyky a překladače

Literatura základní: web stránky **(Nejsou určeny k samostudiu)**
https://courseware.zcu.cz/portal/studium/courseware/kiv/fjp/prednasky.html
Portál ZČU > Courseware > Předměty po fakultách > Fakulta aplikovaných věd > Katedra informatiky a výpočetní techniky > FJP > O předmětu

Další zdroje:

Melichar, Češka, Ježek, Rychta: Konstrukce překladačů (ČVUT)
Melichar: Jazyky a překlady (ČVUT)
Molnár a kol.: Gramatiky a jazyky (Alfa)

Doporučená

Aho, Sethi, Ullman: Compilers Principles Technics and Tools

(v knihovně):

Appel A.W.: Modern Compiler Implementation in Java
http://www.cs.princeton.edu/~appel/modern/java/

On line knihy:

http://link.springer.com/book/10.1007%2F978-3-642-17540-4
https://link.springer.com/book/10.1007%2F978-3-642-14909-2
http://www.diku.dk/~torbenm/Basics/basics_lulu2.pdf

Formální jazyky a překladače - organizace

FJP je šestikreditový předmět doporučený pro 1. ročník navazující *Informatika a výpočetní technika*. K absolvování zkoušky je zapotřebí splnit požadavky cvičení a napsat zkouškový test. Celkové hodnocení se snadno zjistí z bodového zisku a následující převodní tabulky:

Více než 82 b.	**výborně**
65 – 82 b.	**velmi dobře**
51 – 64 b.	**dobře**
0 – 50 b.	**nevyhověl**

Formální jazyky a překladače -organizace

Na některých cvičeních se zadává samostatné vyřešení příkladu do příštího cvičení; správné řešení je bodově honorováno.
V rámci cvičení vypracovávají studenti semestrální práci.
Hodnocení semestrální práce je až 35 bodů. Je určen mezní termín odevzdání práce. Za každý den prodlení se automaticky strhává 1 bod. Z cvičení je zapotřebí získat alespoň 20 bodů.

Zkouška probíhá písemně. Maximální bodové hodnocení je 55 bodů. Pro absolvování zkoušky musí student získat alespoň 30 bodů.
Po probrání ucelených částí bude krátkým testem v rámci přednášky ověřována aktivita studentů. Získané bodové ohodnocení (max. 5 bodů) bude zohledněno při zkoušce.

Využití teorie překladačů a formálních jazyků

Znalost principů překladače patří k základním disciplínám informatiky. Programátorům umožní vytváření lepších programů v konkrétních jazycích, pochopit výhody a úskalí konkrétních programových struktur a lépe porozumět příčinám chyb, které překladač hlásí.

Základní typy:
Assemblery překlad z JSI. Hlavní problém = adresace symbolických jmen, makra

Kompilátory generují kód (strojový / symbolický / jiný jazyk)

Interprety provádí překlad i exekuci programu převedeného do vhodné formy

Využití teorie překladačů a formálních jazyků

Ostatní:

Strukturní editory	napovídají možné tvary konstrukcí programů, či strukturovaných textů
Pretty printers	provádí úpravu struktury výpisů
Statické odlaďovače	vyhledávání chyb bez exekuce programu
Reverzní překladače jazyka	převádí strojový kód do JSI / vyššího
Formátory textu	překladače pro sazbu textu (Tex→DVI)
Silikonové překladače	pro návrh integrovaných obvodů. Proměnné nereprezentují místo v paměti, ale log. proměnnou obvodu. Výstupem je návrh obvodu.
Příkazové interprety	pro administraci OS / sítí (viz shell Unixu)
Dotazovací interprety	analýza a překlad příkazů a podmínek dotazů a příkazů DB jazyků
Preprocesory	realizují vnořování částí programu do hostitelského jazyka (expandují makra, přidají include <něco.h> soubory apod.)
Analyzátory textu	kontroly pravopisu, práce s dokumenty, vyhledávání, indexace, zpracování XML.

Formální jazyky a překladače

Formálně je překladač zobrazením:

Překladač: zdrojový jazyk → cílový jazyk

Činnost assembleru: **JSI → strojový kód**

Činnost interpretu: **vyšší progr. jazyk ⟹ výsledky**

data ⟹

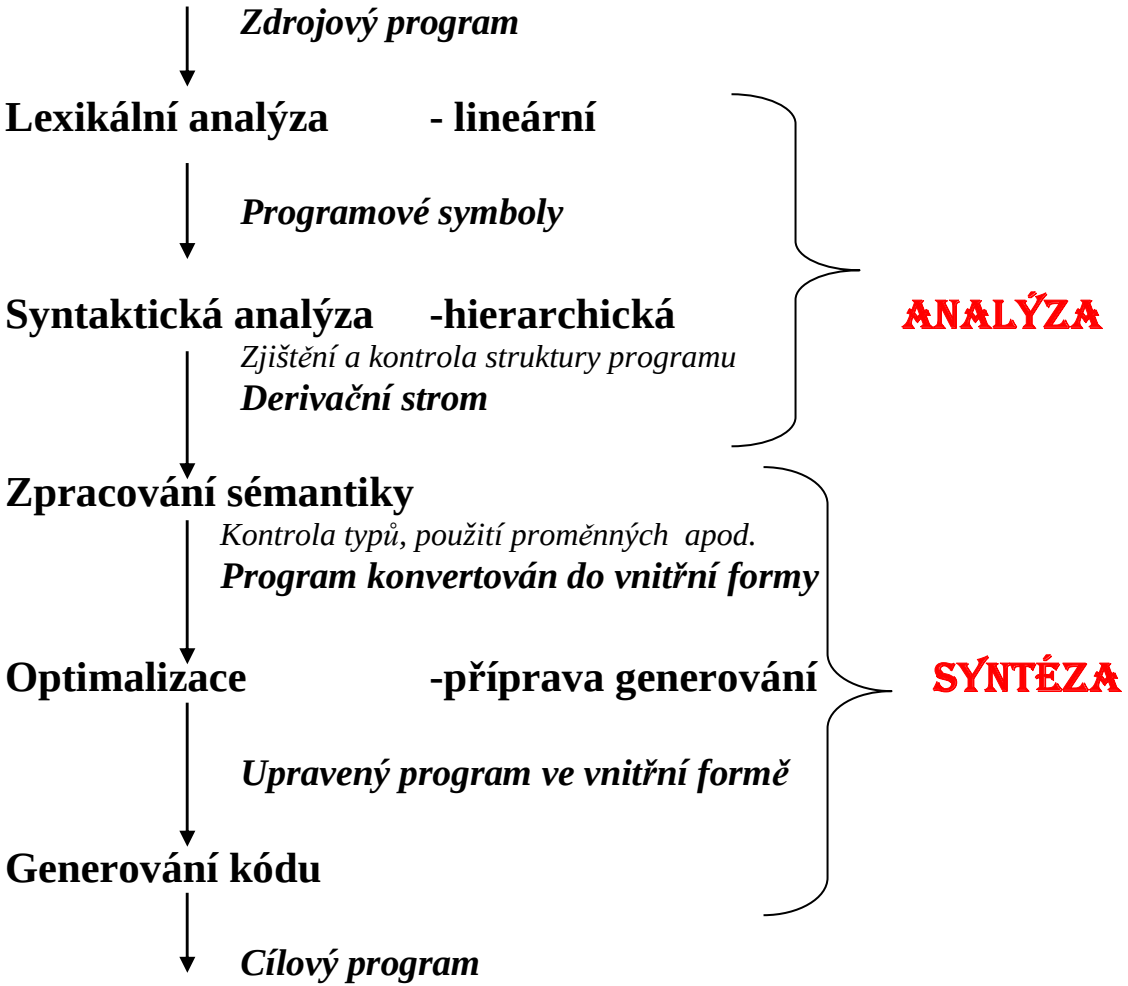
Činnost kompilátoru: **vyšší progr. Jazyk → strojový kód**

Pozn.: První překladač – Fortran IBM (Backus 1957)
pracnost 18 člověkoroků -ad hoc technologie

Využití teorie překladačů a formálních jazyků

Dávkový překladač	batchové zpracování
Konverzační překladač	interaktivní
Inkrementální překladač	interaktivní + překládá po úsecích (př. Basic překlad po řádcích)
Křížový překladač	překlad na jiném procesoru než exekuce (viz zabudované systémy)
Kaskádní překladač	máme již A → B, ale chceme A→C, uděláme B → C. Kdy se to vyplatí? Komplikace - chybová hlášení výpočtu jsou pomíchaná
Optimalizující překladač	**(**možnost ovlivnění optimalizace času / paměti programátorem)
Paralelizující překladač	zjišťuje nezávislost úseků programu

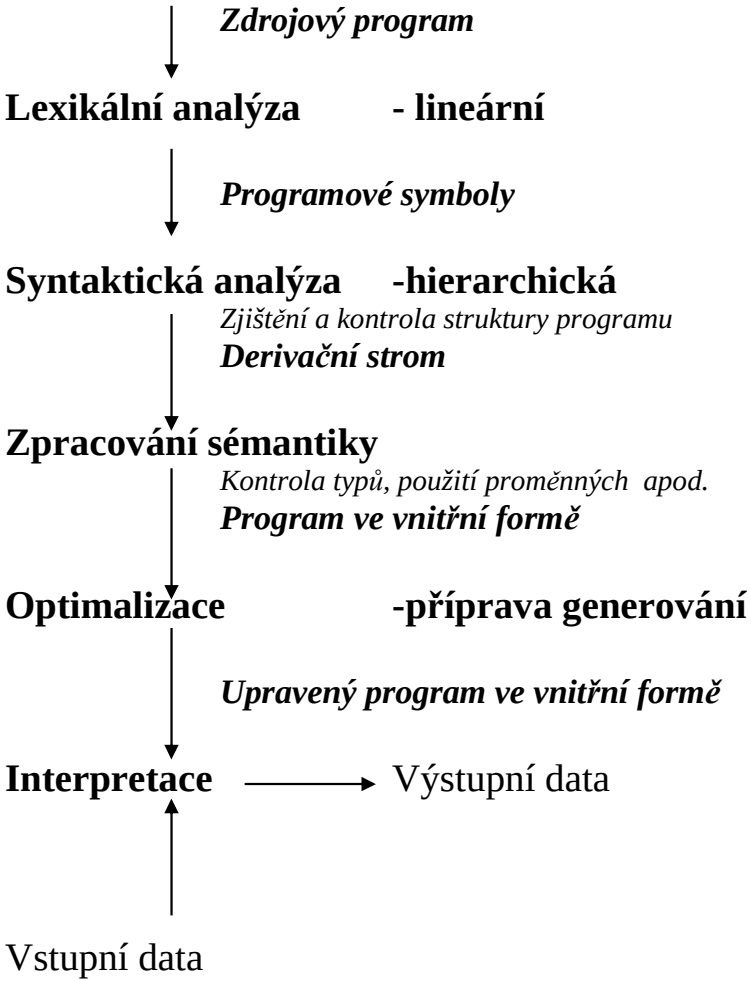
Hlavní části překladače - Kompilátor:



Všechny části spolupracují s pracovními tabulkami překladače.
Základní tabulkou kompilátoru i interpretu je Tabulka symbolů

Výhodou kompilátoru je rychlá exekuce programu

Interpret



Výhodou interpretu je:
eliminace kroků cyklu (Editace→překlad→sestavení→exekuce)

Snazší realizace ladících mechanismů (zachování původních jmen symbolů)

Vícefázový / víceprůchodový překladač

Fáze = logicky dekomponovaná část,
(může obsahovat více průchodů, např. optimalizace).

Průchod = čtení vstupního řetězce,
zpracování,
zápis výstupního řetězce
(může obsahovat více fází).

Jednoprůchodový překladač , vlastnosti:
všechny fáze probíhají v rámci jediného čtení zdrojového textu programu,
omezená možnost kontextových kontrol,
omezená možnost optimalizace,
lepší možnost zpracování chyb a ladění (pro výuku)

Co má vliv na strukturu překladače:

Vlastnosti zdrojového a cílového jazyka,

Vlastnosti hostitelského počítače,

Rychlost/velikost překladače,

Rychlost/velikost cílového kódu,

Ladicí schopnosti (detekce chyb, zotavení),

Velikost projektu, prostředky, termíny.

Testování a údržba překladače:

Formální specifikace jazyka ⇒ možnost automatického generování testů,

Systematické testování ⇒ regresní testy = sada testů, doplňovaná o testy na odhalené chyby. Po každé změně v překladači se provedou všechny testy a porovnají se výstupy s předešlými.

Vnitřní jazyky překladače

Postfixová notace:

Operátory bezprostředně následují za svými operandy, pořadí operandů je zachováno

Vyjadřuje precedenci operátorů

Př.1 a + b → a b +

Př.2 (a + b) * (c + d) → a b + c d + *

Postfixový zápis nepotřebuje (a nemá) závorky

Postfix je elegantně vyhodnotitelný zásobníkem:

1) **Čti symbol postfixového řetězce,**
2) **Je-li symbolem operand, ulož jej do zásobníku.**
3) **Je-li symbolem operátor, proveď jeho operaci nad vrcholem zásobníku a výsledek vlož do zásobníku**
4) **Jdi na 1).**
5) **Po přečtení a vyhodnocení celého řetězce je výsledek uložen v zásobníku (princip interpretace).**

Pro př.2 **(a + b) * (c + d)**

→ a b + c d + *

čte a **čte b** **čte +** **čte c** **čte d** **čte +** **čte ***

?

Jak vypadají obsahy zásobníku nakreslíme na tabuli

!! Pozn. Musíme umět vyjádřit i jiné než konstrukce pro výrazy.

Vnitřní jazyky překladače

Prefixová notace:

Operátory bezprostředně předchází operandy, pořadí operandů je zachováno

Vyjadřuje precedenci operátorů, nepotřebuje závorky

a + b → + a b

(a + b) * (c + d) → * + a b + c d

Prefixový zápis nemá závorky

!Pozor, není to zrcadlový obraz operátorů z postfixu

!!! pořadí operandů u postfixu i prefixu zůstává zachováno, mění se pořadí operátorů!!!

Zkusme na tabuli př. A = - B * C + D

Vnitřní jazyky překladače

Víceadresové instrukce (čtveřice / trojice)

Čtveřice operátor, operand, operand, výsledek
Např. +, a, b, Vysledek

Potřeba přidělovat paměť pomocným prom.

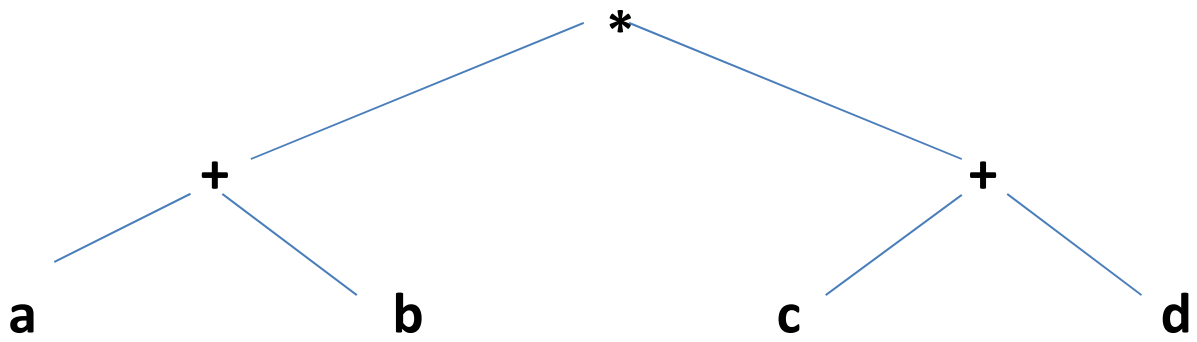
Př 2) (a + b) * (c + d)

tvar	význam
+, a, b, Pom1	Pom1 = a + b
+, c. d. Pom2	Pom2 = c + d
*, Pom1, Pom2, Vysl	Vysl = Pom1* Pom2

Trojice odkládají potřebu přidělovat paměť pomocným proměnným v době generování víceadresových instrukcí. Vztahují výsledek operace k číslu trojice

Př 2)	**1)**	**+,**	**a,**	**b**
	2)	**+,**	**c,**	**d**
	3)	***,**	**(1),**	**(2)**

Vyjadřují abstraktní syntaktický strom



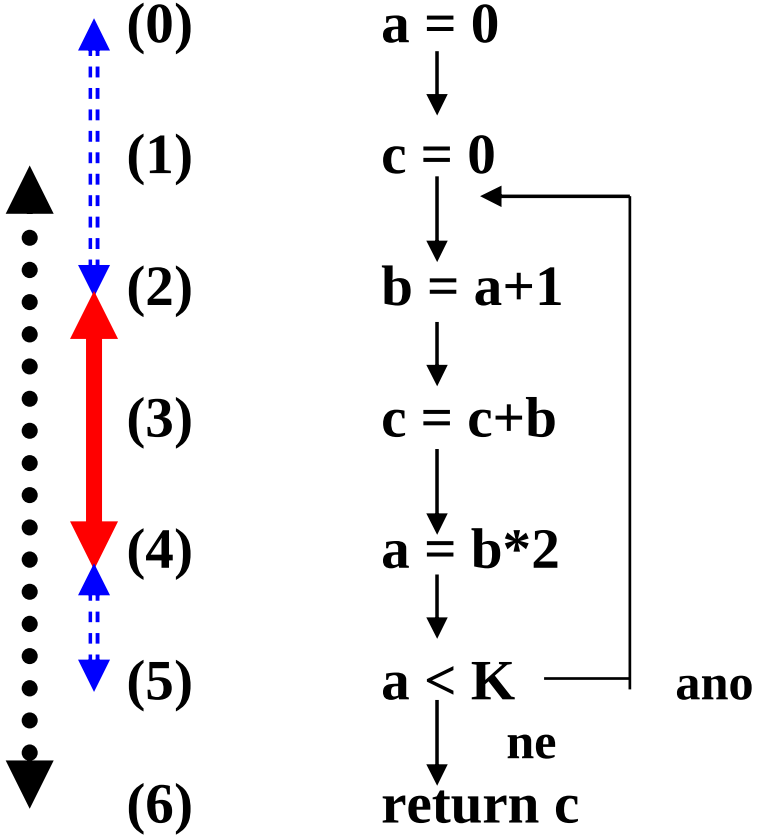
Odlišovat:
Abstraktní syntaktický strom = syntaktický strom
Jeho uzly jsou symboly zdrojového jazyka

Derivační strom = konkrétní synt. strom = (parse tree)
Jeho uzly jsou symboly gramatiky

Teď něco o optimalizaci

Optimalizace (př. redukce počtu registrů)

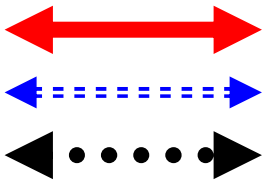
Př.



vývojový diag.

program

```
    a = 0; c = 0;
L : b = a + 1;
    c = c + b;
    a = b * 2;
    if a < K goto L;
    return c;
```



živé b (2) → (3) → (4)

živé a (0) → (1) → (2), (4) → (5) → (2)

živé c (1) → (2) → (3) → (4) → (5) → (2)

(5) → (6)

?Kolik potřebujeme registrů pro proměnné a, b, c, když K je konstanta?

-zjištění živých a neživých proměnných v data flow diagramu,

-vytvoření interferenčního grafu,

-barvení grafu.

počet potřebných barev = počet potřebných registrů

Matice interferencí

	a	b	c
a			x
b			x
c	x	x	

Výsledek= ?

Optimalizace cyklů (vysazení nezávislé části před cyklus).

```
while (j < k) {
sum = sum + a[i][j];  q = fce(k);
j++;
}

if (j < k) {
sum = sum + a[i][j]; q = fce(k);
j++;
while (j < k) {
sum = sum + a[i][j];
j++;
}
}
```

Optimalizace v základním bloku (je lokální optimalizací)
Základní blok = sekvence příkazů bez skoků mezi sebou
Často analýza metodou posuvného okénka (peephole)

-Eliminace společných subvýrazů

Př. a[i] := a[i]+2;

-Propagace konstant

```
if (a=100) {
        x = x + a;
}
```

-Eliminace kontroly indexu

```
i = 0;
a[i] = a[i] + 5;
```

-Odstranění mrtvého kódu

-Strenght reduction

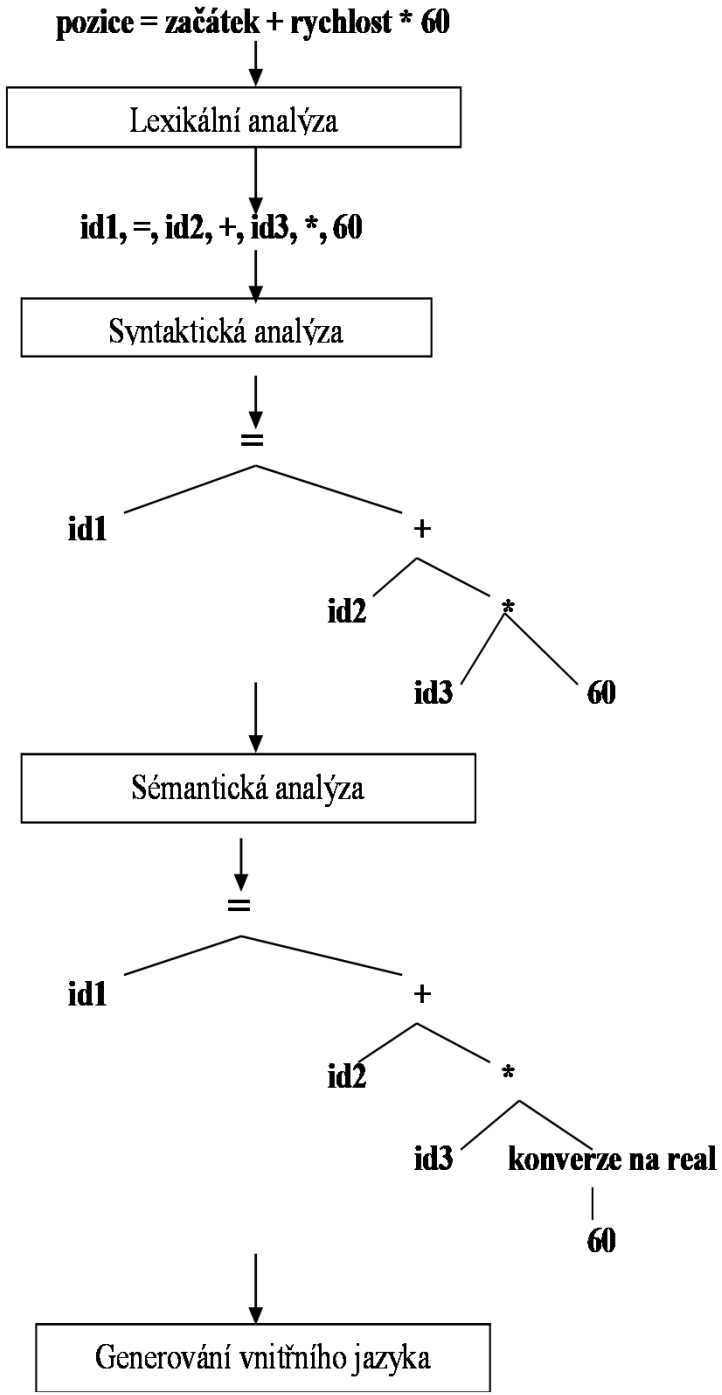
-Přeuspořádání kódu

-Algebraické optimalizace

Globální optimalizace pracuje vně základních bloků

Př. překladu příkazu

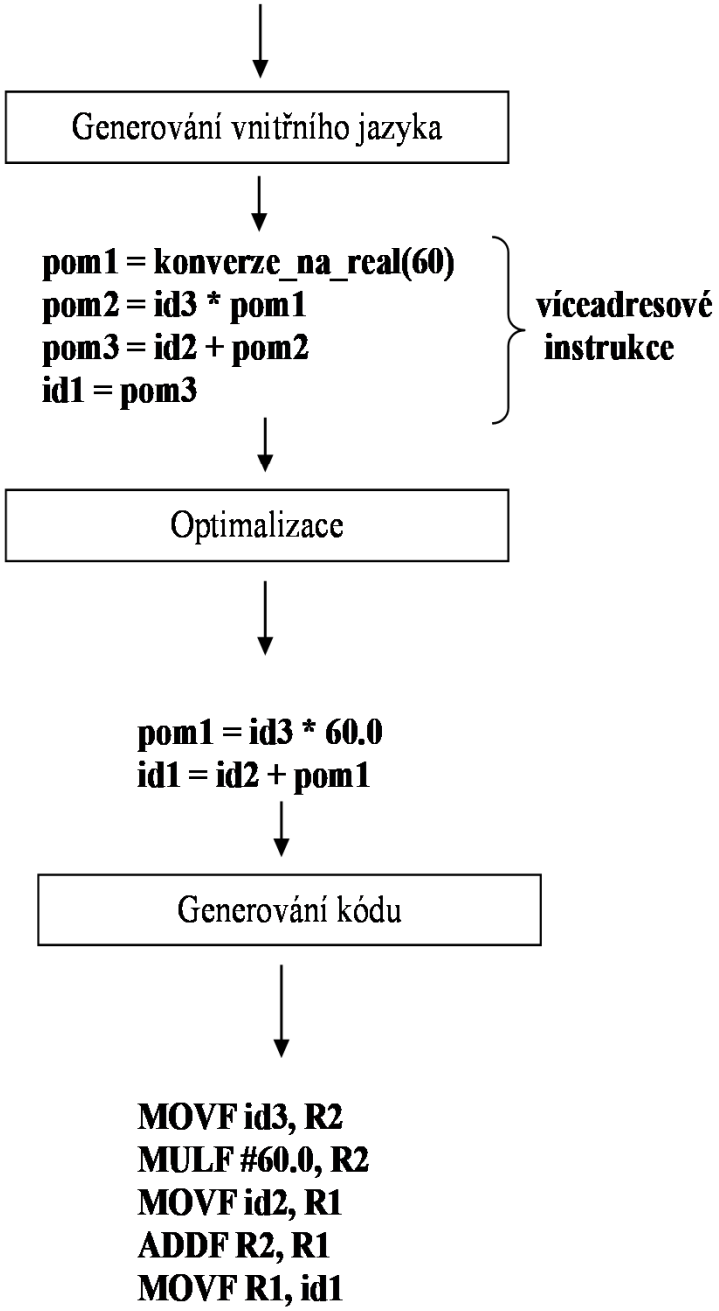
pozice = začátek + rychlost * 60



Část 1

1	pozice	...
2	zacatek	...
3	rychlost	...
4		
5		

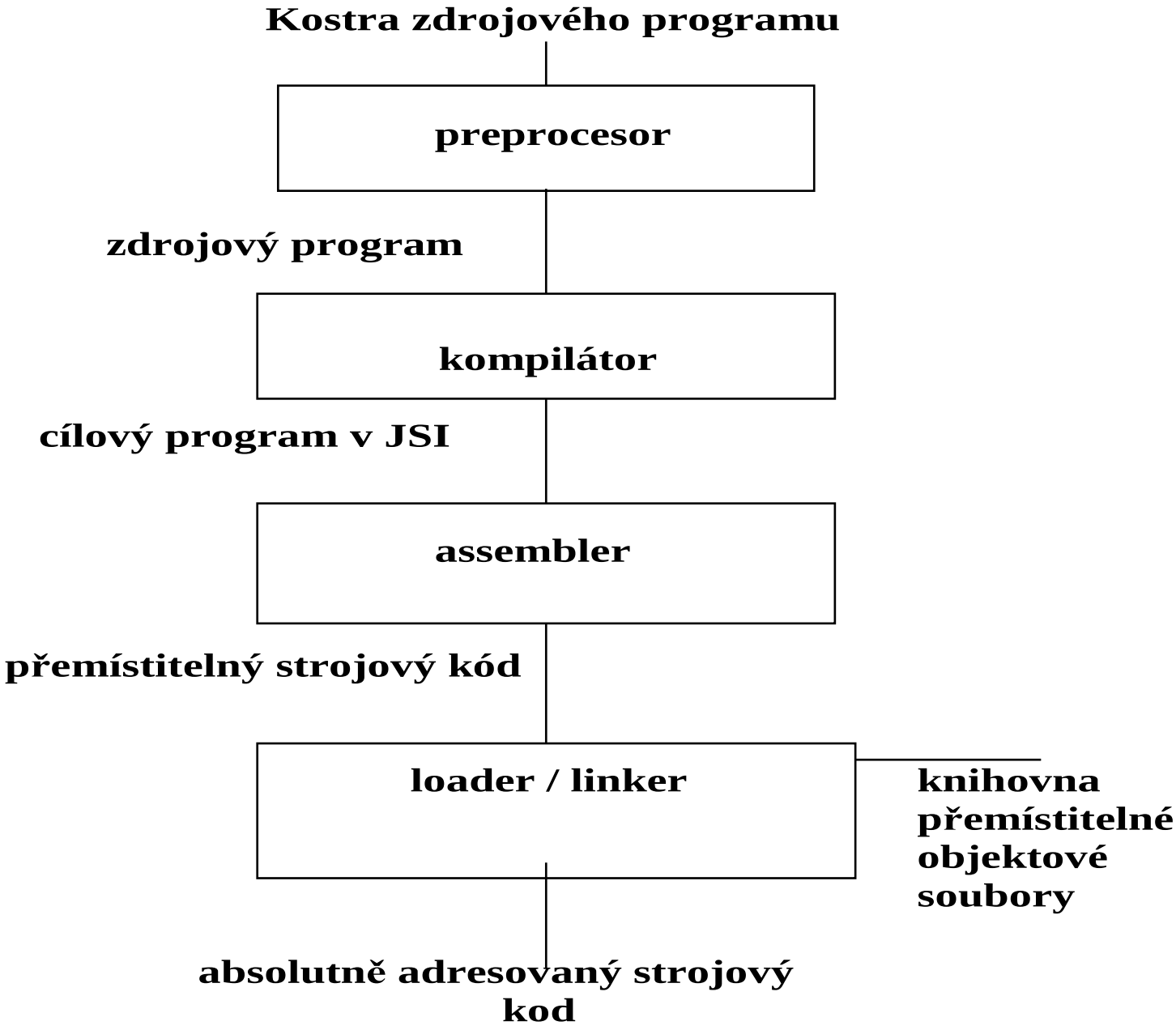
Část 2



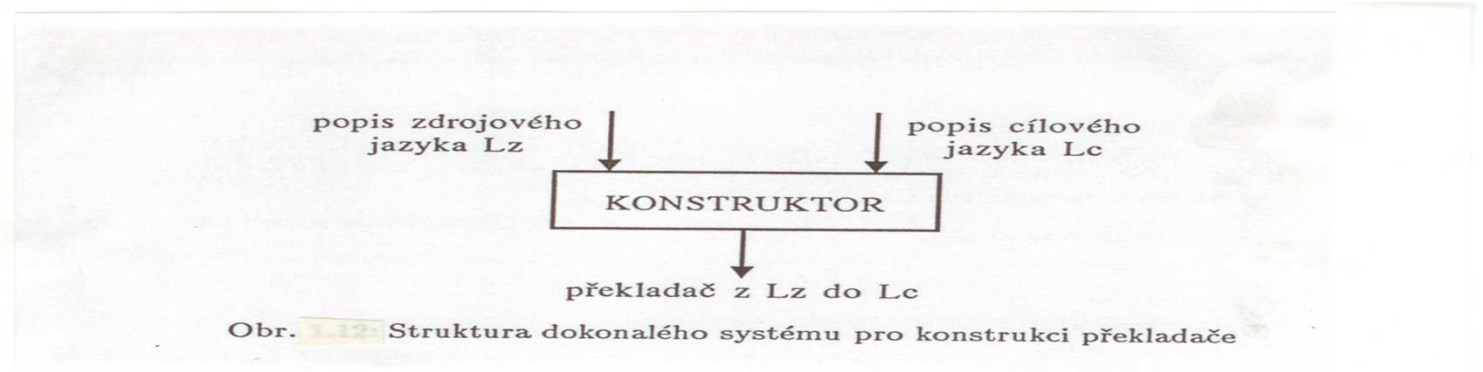
Tabulka Symbolů

1	pozice	...
2	zacatek	...
3	rychlost	...
4		
5		

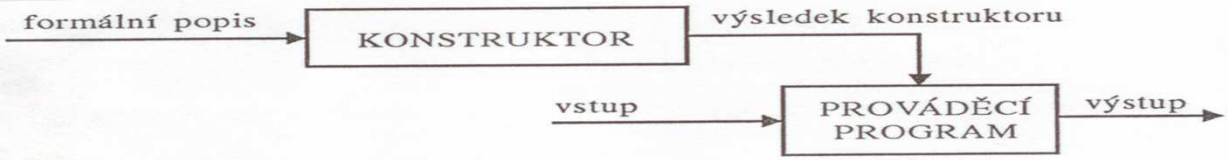
Systém zpracování jazyka



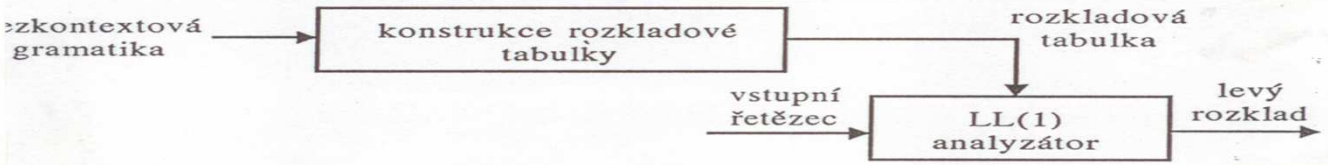
Principální možnost automatizace kstrukce překladače



Obr. 1.12: Struktura dokonalého systému pro konstrukci překladače

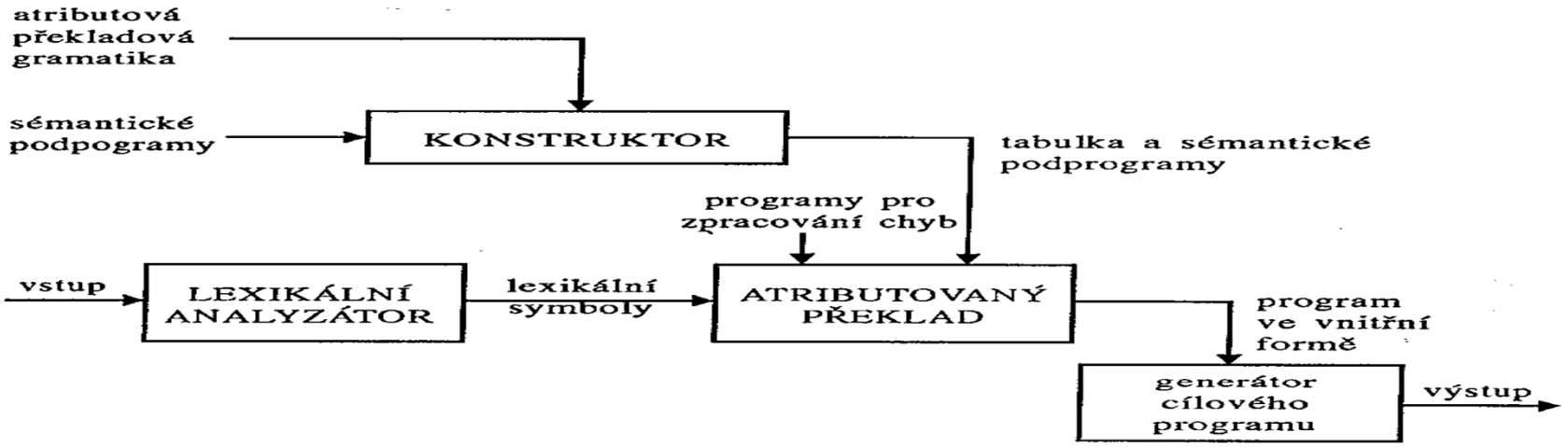


Obr. 1.13: Dvojice *konstruktor—prováděcí program*

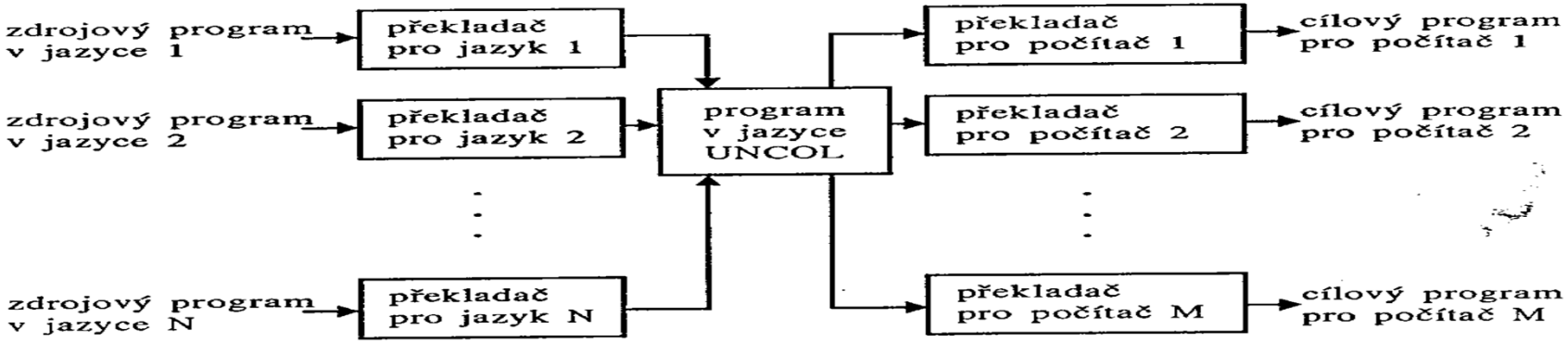


Obr. 1.14: Systém pro konstrukci $LL(1)$ analyzátoru

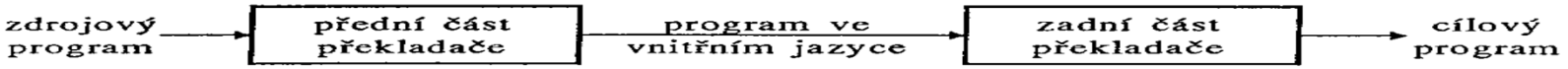
Jaké prostředky k tomu máme a co bychom chtěli



Obr. Systém automatické konstrukce syntaktického analyzátoru s připojeným zpracováním sémantiky



Obr. Základní myšlenka univerzálního vnitřního jazyka UNCOL



Obr. Přední a zadní část překladače

Stránky softwarových prostředků pro zpracování formálních jazyků

Lex, Flex, Yacc, Bizon stránky **http://dinosaur.compilertools.net/**

Compiler Construction Kits http://catalog.compilertools.net/kits.html

Lexer and Parser Generators http://catalog.compilertools.net/lexparse.html

Attribute Grammar Systems http://catalog.compilertools.net/attribute.html

Transformation Tools http://catalog.compilertools.net/trafo.html

Backend Generators http://catalog.compilertools.net/backend.html

Program Analysis and Optimisation ttp://catalog.compilertools.net/optim.html

Environment Generators http://catalog.compilertools.net/env.html

Infrastructure, Components, Tools http://catalog.compilertools.net/infra.html

Compiler Construction with Java http://catalog.compilertools.net/java.html