Kompilátory
Úvodná prednáška

Ján Šturc
zima 2010

Literatúra

- A.V. Aho, M.S. Lam, R. Sethi, J. D. Ullman: Compilers - Principles, techniques and tools. Addison-Wesley 2006
- Aho, Sethi and Ullman: Compilers - Principles, techniques and tools. Addison-Wesley 1986
- Aho, Ullman: Principles of Compiler Design. Addison-Wesley 1977
- Steven Muchnick: Advanced Compiler Design and Implementation. Morgan Kaufman Publishers 1997
- A. W. Appel: Modern Compiler Implementation in Java. Cambridge University Press 1998
- C.N. Fischer, R.J. LeBlanc: Crafting a Compiler. The Benjamin/Cummings Publishing Company, Inc. 1988

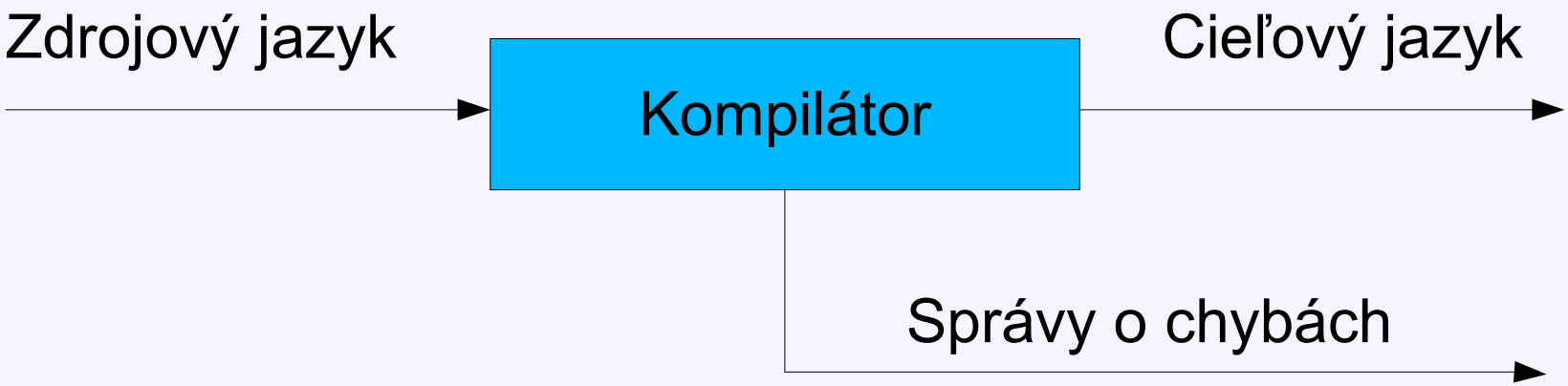
Časopisy a konferencie

- ACM Transactions on Programming Languages and Systems (TOPLAS)
- Software practice and experience

Teória kompilátorov sa považuje za už uzavretú vednú disciplínu. Znamená to, že „koryfeji“ si myslia, že sa v nej nedá už nič objaviť.
Neplatí pre optimalizáciu programov a kódu.

Bakalárske práce sa dajú robiť.
Magisterské a PhD. len v prípade novej prieraznej myšlienky.
Väčšinou nie.

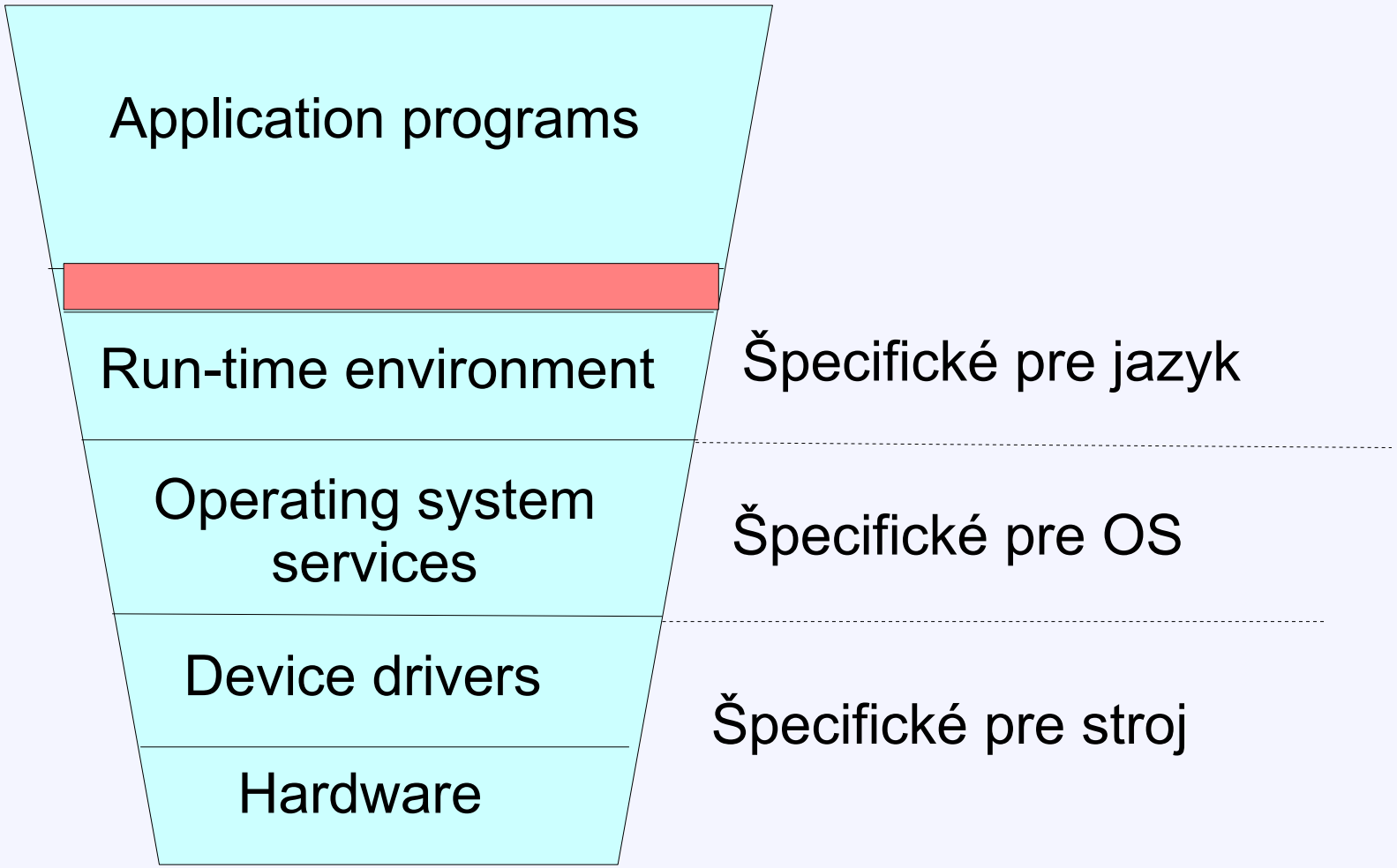
Kompilátor – čo to je ?



V prednáške:
Zdrojový jazyk je nejaký programovací jazyk (Pascal, C, Java, …)
Cieľový jazyk kód abstraktného alebo skutočného počítača.

Vrstvová (cibuľová) predstava hardware a software.
Obrátená pyramída.

Obrátená pyramída



Základné úlohy

- Rozpoznanie vstupu
  - Lexikálna analýza
  - Syntaktická analýza
  - Vytvorenie syntaktického stromu
- Vytvorenie cieľového abstraktného stroja (medzijazyk a podpora počas behu)
  - správa pamäti
  - zásobník, heap (halda), tabuľky – **dátové štruktúry**
  - makrá a podprogramy
- Optimalizácia
  - analýza toku dát
  - dynamické programovanie
  - kukátková (peephole) optimalizácia
- Generovanie „kódu“

Gramatiky a jazyky

- Gramatika $G = < N, T, R, S >$, kde
  - N je množina neterminálnych symbolov (premenných)
  - T je množina terminálnych symbolov
  - S je počiatočný symbol
  - R je množina pravidiel
- Jazyk $\mathcal{L}(G) = \{\omega: (S \underset{*}{\rightarrow} \omega) \wedge (\omega \in T^*)\}$.
- Chomského hierachia obmedzenie na pravidlá

  0. Žiadne obmedzenie $\alpha \rightarrow \omega$, $\alpha \in (N \cup T)^*$ a $\omega \in (N \cup T)^*$.
  1. Nepredlžujúce $\alpha \rightarrow \omega$, $\alpha \in (N \cup T)^*$ a $\omega \in (N \cup T)^*$ a $|\alpha| \leq |\omega|$.
  1'. Kontext senzitívne $\alpha x \beta \rightarrow \alpha \omega \beta$, $x \in N$ a α, β, ω sú z $(N \cup T)^*$.
  2. Bezkontextové $x \rightarrow \omega$, $x \in N$ a $\omega \in (N \cup T)^*$
  3. Regulárne $x \rightarrow \omega$ alebo $x \rightarrow \omega y$, x a y sú z N a $\omega \in T^*$.
- **Pre nás sú zaujimavé** podtriedy bezkontextových (kontext senzitívnych) jazykov, ktoré sa dajú rozpoznať v lineárnom čase.

Syntax a sémantika

- Syntax je to čo je popísané gramatikou.
- Sémantika to ostatné.

  Sú to relatívne pojmy: V logike je niečo syntaktická vlastnosť, ak sa to dá rozpoznať algoritmicky.

Príklad: (Syntaxou riadený preklad)

- $S \rightarrow aSbS \mid \varepsilon$ Dyckov jazyk (správne spárované zátvorky)
- $S \rightarrow a\{\ count{+}{+};\}S$

  $S \rightarrow b\{\ count{-}{-};\ \textbf{if}\ count < 0\ \textbf{then}\ error;\}S$

  $S \rightarrow \varepsilon \quad \{\ \textbf{if}\ count > 0\ \textbf{then}\ error;\}$

Poznámka: V kompilátoroch ε-pravidlá používame len, keď sa žiadne iné pravidlo nedá použiť. Kontext senzitívne pravidlo tento problém rieši: $aSb \rightarrow ab$ (až na prázdne slovo).

Aby to fungovalo, musíme pridať pravidlo $S' \rightarrow \{\ count{:=}0;\}S$, kde S' je nový počiatočný symbol.

Mikrokompilátor

- Rozpoznáva Dyckov jazyk pomocou regulárnej gramatiky

```
procedure S';
begin count:=0; S end;
procedure S;
begin if lookahead = 'a' then { match('a'); count++; S;}
                else lookahead = 'b': { match('b'); count--;
 if count = 0 then {if lookahead = '$' then accept
                elseif  lookahead = 'a' then S else error }}
end;
```

• Po optimalizácii:

```
count:= 0;
Loop:   c:= readchar;
            if c = 'a' then { count++; goto Loop;}
            if c = 'b' then { count--;
                if count = 0 then case lookahead of
                  $: accept; a: goto Loop; other: Error }
```

Lexika a syntax

- Iné relatívne rozdelenie. Teoreticky zbytočné. Prakticky užitočné.
- V prirodzenom jazyku lexika je slovná zásoba, rozpoznávanie slov, gramatika či syntax sa týka štruktúry viet.
- V kompilátoroch je lexika časť, ktorá sa dá popísať regulárnym jazykom.
- Lexika zahrnuje:
  - rezervované slová, identifikátory
  - konštanty
  - operátory
  - oddelovače (punktuáciu)
- Lexiku oddelujeme, lebo je to zdlhavá, ale triviálna časť.
  - letter → 'A' | 'B' | … | 'Z' | 'a' | … | 'z' veľa pravidiel a znakov
  - begin → b e g i n dlhé pravidlá
    - identifier → letter (letter | digit)*

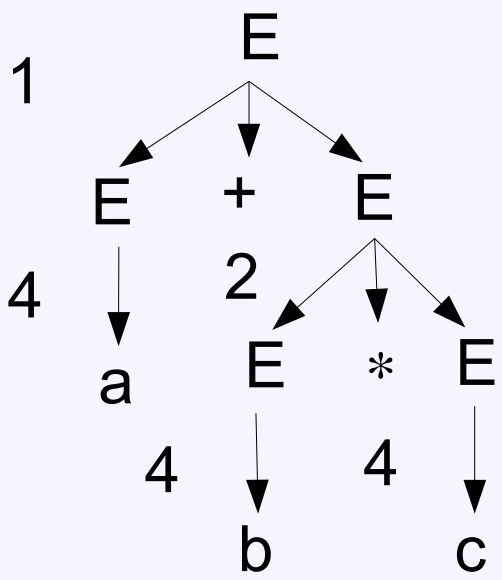
      identifier → letter | identifier letter | identifier digit

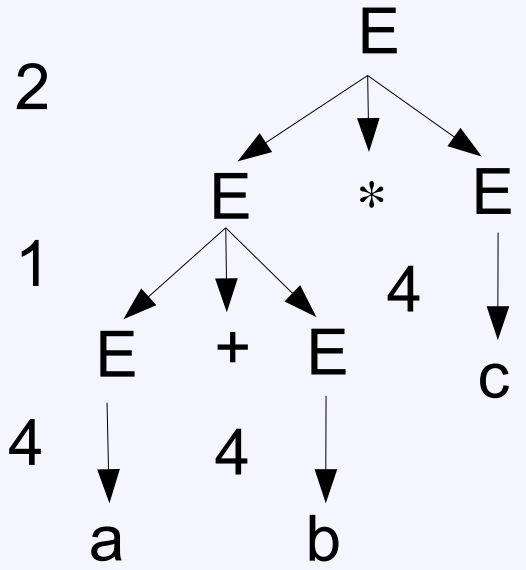
Syntaktický strom, parse

- Aritmetický výraz (expression)

1. $E \rightarrow E + E$
2. $E \rightarrow E * E$
3. $E \rightarrow (E)$
4. $E \rightarrow$ **id**

$a + b * c$





Ľavé (leftmost) odvodenie: 14244, 21444
Pravé (rightmost): 12444, 24144

Gramatika je jednoznačná (nonambigous), ak každé slovo jazyka $\mathscr{L}(G)$ má jediný syntaktický strom.

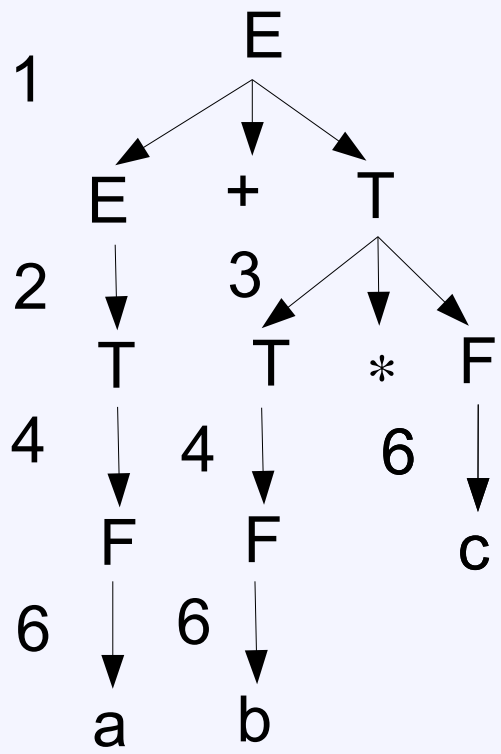
Programovací jazyk má mať jednoznačnú gramatiku.

Úskalie jednoznačnosti

- Jednoznačná gramatika pre aritmetický výraz:

1. $E \rightarrow E + T$
2. $E \rightarrow T$
3. $T \rightarrow T * F$
4. $T \rightarrow F$
5. $F \rightarrow (E)$
6. $F \rightarrow \mathbf{id}$

$a + b * c$



- Viac neterminálnych symbolov
- Viac pravidiel
- Väčší syntaktický strom

- Keď vynecháme prepisovacie pravidlá (2, 4) a premenujeme naspäť T a F na E (homomorfizmus) dostaneme prvý strom z predošlého slidu.

Kontext senzitívne zjednoznačnenie

1. E → E + E (+ | ')' | $) Pravidlo sa použije len, keď nasleduje jeden zo symbolov v zátvorke. Tento symbol zostane na vstupe. $ je pravá zarážka EOF. Alternatívne, pravidlo sa nepoužije, keď nasleduje ∗ .
2. E → E ∗ E
3. E→ (E)
4. E→ **id**

Pravé odvodenie a + b∗ c je 12444 a to je prvý strom.

Množiny First a Follow

Daná je gramatika $G=<N,T,R,S>$.

$First(x) \subseteq T \cup \{\varepsilon\}$, ak $x \in T$, potom $First(x) = \{x\}$.

Ak $x \in N$, potom

$First(x) = \{ y: x \xrightarrow{*} y\omega \wedge (y \in T) \vee (y = \varepsilon) \wedge (x \rightarrow \varepsilon) \in R)\}$.

$Follow(x) \subseteq T \cup \{\$\}$ ($ označuje pravú zarážku - EOF)

$Follow(x) = \{y: S \xrightarrow{*} \alpha x y \omega \wedge (y \in T) \vee (y = \$) \wedge S \rightarrow^* \beta x \}$.

Zovšeobecnenie First pre reťazce:

Nech $\xi = x_1 x_2 ... x_n$, potom $First(\xi)$ vypočítame:

$First(\xi) := First(x_1); i := 1;$

while $i \leq n \wedge \varepsilon \in First(x_i)$ **do** $\{ i++; First(\xi) := First(\xi) \cup First(x_i); \}$

! Výpočet First a Follow.

Metódy syntaktickej analýzy

- Všeobecné metódy
  - Cocke-Younger- Kasami (CYK) $O(n^3)$
    - Earley $O(n^3)$, ale ak jazyk je jednoznačný $O(n^2)$
- Zhora dolu (top-down)
  - Na základe množín First háda pravidlá v ľavom odvodení. Aby nebol potrebný backtrack, musí gramatika spĺňovať veľa obmedzení.
  - Algoritmy: rekurzívny zostup (recursive descent), LL(1) zásobníkový automat **O(n)**
- Zdola nahor (bottom-up)
  - Hľadá pravé strany pravidiel vo vstupe a redukuje ich na základe množín Follow. Konštruje pravé odvodenie v opačnom poradí. Menej obmedzení.
  - Algoritmus: posunovo redukčná (shift reduce) schéma. Precedenčná, ...LR(1) metódy.

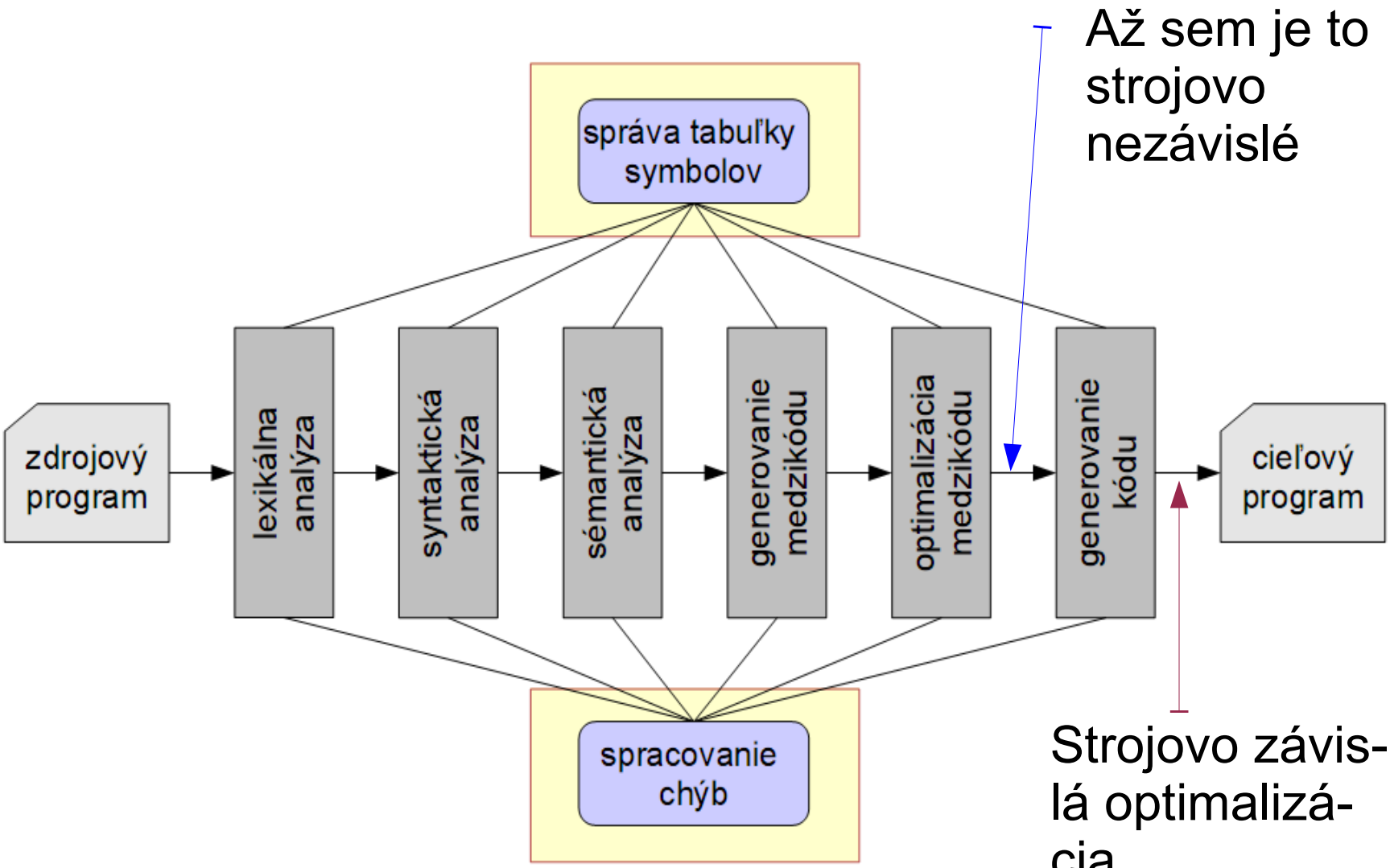
Postavenie syntaktickej analýzy

- V kompilátoroch je obvykle synaktická analýza hlavný program. Hovoríme o syntaxou riadenom preklade.
- Syntax driven programming je programovacia paradigma. Bola v móde začiatkom 70-tých rokov.
  - Osobne si myslím, že poskytuje väčšiu flexibilitu ako objektové programovanie, postráda ale grafický interface.
- Realizuje sa to vložením „sémantických akcií“ do pravidiel gramatiky.
  - Pri analýze zhora dolu, vieme vždy, v ktorom pravidle sa nachádzame, teda môžu byť tieto akcie kdekoľvek v pravidle.
  - Pri analýze zdola nahor pravidlo rozpoznáme až na konci. Sémantické akcie môžu byť len na konci pravidla.
  - Modifikácia gramatiky vložením umelých neterminálov s ε-pravidlami.

Sémantika

- Sémantika sa v kompilátoroch obmedzuje na kontrolu podmienok, ktoré sa nedajú popísať bezkontextovým jazykom.
  - deklarácia, použitie
  - násobné deklarácie toho istého objektu
  - kontrola typov
- Systém typov
  - základné typy (boolean, integer, char, real, …)
  - typové konštruktory (array, record, function, pointer, …)
  - pomocné typy (void, error, …)
- Ekvivalencia typov
  - menná zhoda
  - štrukturálna ekvivalencia
- Polymorfizmus – unifikácia

Štruktúra kompilátoru

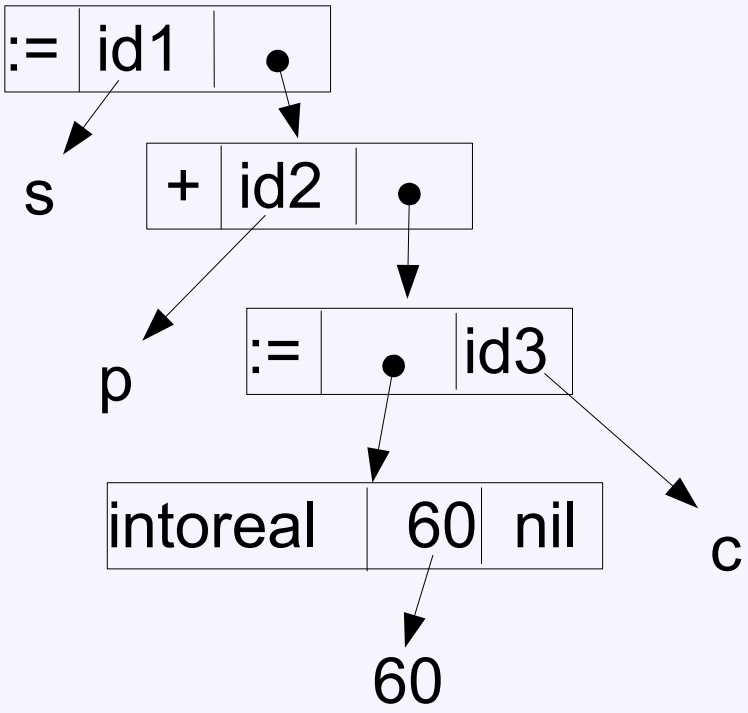


Medzijazyk (medzikód)

- Kód abstraktného trojadresového počítača pozostáva zo štvoríc <operácia, 1. operand, 2. operand, výsledok>.
  Často používame infixovú notáciu:
  výsledok:= 1. operand operácia 2. operand
  Medzikód je postupnosť takýchto operácii.
- Trojicová forma <operácia, 1. operand, 2. operand >.
  V tomto prípade výsledok je samotná trojica, operandy sú reprezentované smerníkmi na trojice.
- Niekedy sa ako medzijazyk používa polská suffixová forma. Nedá sa moc optimalizovať. Je však vhodná pre okamžitý výpočet na zásobníku.

Príklad

real s, p, c;
s := p + 60 ∗ c;



štvorice:

t_1:= ref s
t_2:= p
t_3:= 60
t_4:= intoreal t_3
t_5:= c
t_6:= t_4 ∗ t_5
t_7:= t_2 + t_6
s:= t_7 (store t_7,t_1)

Po optimalizácii:

t_2:= p
t_5:= c
t_5:= #60.0 ∗ c
t_2:= t_2 + t_5
s:= t_2

Generovaný kód:

```
MOVF   p, R1
MOVF   c, R2
MULF #60.0, R2
ADDF   R2, R1
MOVF   R1, s
```

V skutočnosti kompilátor negeneruje asembler, ale kód mu zodpovedajúci s absolútnymi adresami. (BRC)

Podpora počas behu

- Silne závisí od jazyka
  - prológ, WAM
  - SQL, relačná algebra
  - Java, JRE
- Pre klasické imperatívne jazyky
  - správa pamäti (zásobník)
  - volanie rekurzívnych podprogramov
  - tabuľka symbolov
- Objektové jazyky
  - štandardné preddefinované objekty
  - aplikačný interface

Tabuľka symbolov

- Potrebné informácie o objektoch programu: identifikátoroch, konštantách, funkciach
  - názov
  - typ
  - alokovaná pamäť (adresa, veľkosť)
  - pri funkciach aj počet a typ parametrov a návratovej hodnoty
- Informácie potrebné pre optimalizáciu a generovanie kódu
  - adres descriptor (kde všade sa nachádza, registre, pamäť)
- V jazykoch z blokovou štruktúrou musí rešpektovať rozsah platnosti (scope).
- Primeraná veľkosť a rýchlosť operácii (find, insert, modify)
- Kvôli ladeniu (debugging) žije aj počas behu programu.

Nástroje: compiler-compiler, TWS

- Lexikálna analýza (generátory scannerov)
  - vstup popis regulárneho jazyka
  - Lex, Flex
  - existuje aj veľa iných menej rozšírených
- Syntaxou riadený preklad (generátory parserov)
  - vstup obvykle CF – gramatika
  - Yacc, Bison LALR(1)
  - CoCo/R LL(1)
  - ANTLR
- Generátory kódu
  - prekladajú medzijazyk do strojového kódu
  - sú to obvykle šablóny alebo makrá
- Analyzátory toku dát
  - riešia úlohy toku dát

Prečo sa tým máme zaoberať

- Proti
  - Softwarové firmy to už majú zvládnuté, všetky kompilátory sú urobené.
  - Ako teoretická disciplína je to prakticky uzavreté. Nedajú sa očakávať významnejšie objavy.
- Za
  - koncom 50-tých rokov bol kompilátor veľký výskumny projekt (napr. Fortran 1957). Dnes sa kompilátor zvládne aj ako študentský ročníkový projekt.
  - je to poučná ukázka zvládnutia ťažkej softwarovej úlohy a využitia teórie formálnych jazykov v praxi.
  - Techniky majú širšie využitie
    - editovanie
    - tlačové a sádzacie programy (napr. LaTeX)
    - statická kontrola dát, programov a dokumentov
    - spracovanie štruktúrovaných dokumentov (XML)