Kompilatory
(1) Wstęp

Paweł J. Matuszyk

AGH Kraków

Plan wykładu

Literatura

- Hopcroft J. E., Ullman J. D., *Wprowadzenie do teorii automatów, języków i obliczeń*, PWN,
- Blikle A., *Automaty i gramatyki*, PWN, 1971.
- Kowalski St., Mostowski A. Wł., *Teoria automatów i lingwistyka matematyczna*, PWN, 1979. 1994.
- Blikle A., *Wybrane zagadnienia lingwistyki matematycznej*, w: *Problemy przetwarzania informacji*, tom 2, praca zbiorowa pod red. A. Mazurkiewicza, WNT 1974.
- Aho A.V., Ullman J.D., *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, 1972.
- Foster J.M., *Automatyczna analiza składniowa*, 1976.
- Gries D., *Konstrukcja translatorów dla maszyn cyfrowych*, 1984.
- Hopgood F.R.A., *Metody kompilacji*, 1982.
- Waite W.M., Goos G., *Konstrukcja kompilatorów*, WNT, 1989.
- Aho A.V., Sethi R., Ullman J.D., *Compiler: Principles, Techniques and Tools*, Addison-Wesley, 1986.

Translatory

Translator

program, który umożliwia wykonanie programów napisanych w języku różnym od języka komputera.

Wyróżniamy dwa rodzaje translatorów:

Kompilatory

to programy, które tłumaczą program źródłowy na równoważny program wynikowy.

Interpretery

które można nazwać dynamicznymi translatorami — tłumaczą na bieżąco i wykonują program źródłowy.

Kompilator
Podstawowe cechy

- Program źródłowy jest napisany w języku źródłowym, a program wynikowy należy do języka wynikowego.
- Wykonanie programu kompilatora następuje w czasie tłumaczenia.
- Program po przetłumaczeniu nie da się zmienić, jest statyczny.
- Kompilator jako dane wejściowe otrzymuje „cały” program źródłowy i przekształca go na postać wynikową.

Interpreter
Podstawowe cechy

Działanie interpretera polega na:

- wyodrębnieniu niewielkich jednostek programu źródłowego,
- tłumaczeniu ich na pewną postać wynikową oraz
- natychmiastowym ich wykonywaniu.

Cechy:

- Proces interpretacji jest cykliczny.
- W czasie interpretacji przechowywany jest program źródłowy.

Zalety

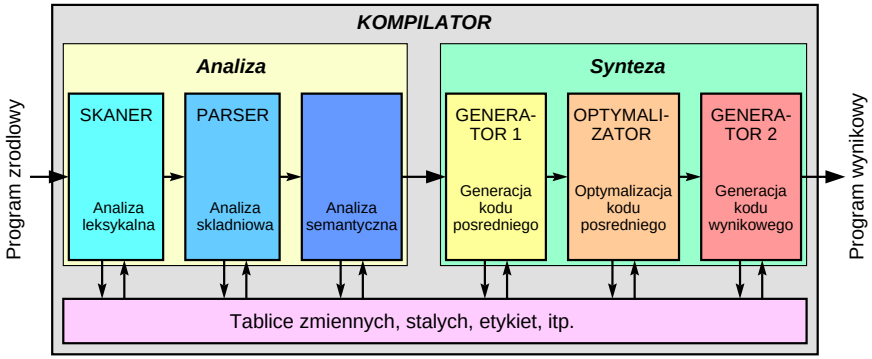
Kompilatory

- Program wynikowy wykonuje się szybciej.
- Do wykonania programu wynikowego nie jest potrzebny kompilator.
- Lepsza optymalizacja kodu (optymalizacja dedykowa- na pod daną architekturę).

Interpretery

- Łatwość zmian programu.
- Mniejsza zajętość pamięci zewnętrznej (tylko tekst źródłowy).
- Możliwość pracy konwersacyjnej (zatrzymanie wyk., zmiana wartości zmiennych, kontynuacja wyk.).
- Przenośność, wykorzystanie w zastosowaniach sieciowych.

Konstrukcja kompilatora



Etap analizy

Rozdzielenie etapu analizy na dwie odrębne funkcje:

- analizę leksykalną oraz
- analizę syntaktyczną

sprawia, że jedna i druga mogą być wykonywane przy użyciu bardziej efektywnych algorytmów, gdyż algorytmy te istotnie się różnią, wykorzystując inne pryncypia formalne i realizacyjne.

Na czym polega analiza leksykalna?

Skaner dostając na wejściu ciąg znaków przetwarza je i produkuje ciąg tokenów.

Przykład

we: `koszt := (cena + VAT) * 0.99;`
wy: `id1 := ( id2 + id3 ) * num4`

Nazwa	Typ	Informacje
`koszt`	variable	zmienna_pozycyjna
`cena`	variable	zmienna_pozycyjna
`VAT`	variable	zmienna_pozycyjna
`0.99`	constant	zmienna_pozycyjna

Tablica: Tablica symboli

Zadania skanera

- wyodrębnianie tokenów,
- ignorowanie komentarzy,
- ignorowanie białych znaków,
- korelowanie błędów zgłaszanych przez kompilator z numerami linii kodu,
- tworzenie kopii zbioru wejściowego łącznie z komunikatami o błędach,
- preprocessing.

Tokeny, leksemy, wzorce

```
const double PI=3.1415;
```

leksemy:	const	double	PI	=	3.1415
tokeny:	*const*	*double*	*id*	*relop*	*number*

Wzorce:

- są opisami sposobu wyodrębniania tokenów,
- pełnią role produkcji w gramatykach leksykalnych.

Tokeny:

- są symbolami nieterminalnymi w gramatykach leksykalnych,
- są symbolami terminalnymi w gramatykach syntaktycznych.

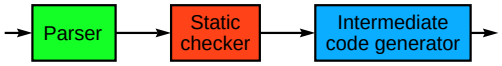
Zadania parsera

- Ma za zadanie dokonać rozbioru syntaktycznego tekstu źródłowego analizowanego programu.
- Pracuje na podstawie gramatyki syntaktycznej, która na ogół jest gramatyką bezkontekstową.
- Od parsera oczekujemy odtworzenia wywodu i jakiegoś zanotowania tego wywodu — najczęściej stosowanym sposobem notowania wyprowadzenia jest podanie ciągu numerów produkcji.
- Niekiedy równolegle z analizą syntaktyczną parser wykonuje pewne akcje semantyczne jak np. generowanie kodu pośredniego w postaci ONP, tetrad (n-tek), itd.

Analizator semantyczny

analiza semantyczna = kontrola statyczna

Kontrola statyczna, a w szczególności także kontrola typów może być przeprowadzana (przynajmniej częściowo) równolegle z parsingiem i/lub generacją kodu pośredniego albo niezależnie, jak na schemacie poniżej.



Zadania analizatora semantycznego

- Kontrola typów (*type checking*) — np. czy operatory nie są zastosowane do nieodpowiednich operandów;
- Kontrola przebiegu sterowania (*flow-of-control-checking*) — kompilator musi sprawdzić, czy sterowanie w programie jest przekazywane w odpowiednie miejsce (np. czy nie jest wykonywany skok spoza pętli do jej wnętrza);
- Kontrola unikalności (*uniqueness checking*) — kompilator powinien sprawdzić, czy pewne obiekty są definiowane w programie tylko jeden raz;
- Kontrola powtarzalności nazw (*name-related checking*) — kompilator powinien sprawdzić, czy w pewnych konstrukcjach określona nazwa pojawia się więcej niż jeden raz.

Generacja kodu pośredniego

- Kod pośredni jest z reguły niezależny sprzętowo.
- Przyczyny jego stosowania:
  - Łatwość generowania kompilatorów tego samego języka dla różnych maszyn.
  - Łatwość przeprowadzania optymalizacji na bazie kodu pośredniego niezależnie od sprzętu.
- Języki kodu pośredniego są językami dla pewnej maszyny abstrakcyjnej:
  - Odwrotna notacja polska (notacja postfiksowa) $\Rightarrow$ maszyna stosowa.
  - Drzewa syntaktyczne lub grafy skierowane acykliczne.
  - Kod trójadresowy.

Generacja kodu wynikowego

- Problem generacji optymalnego kodu zależnego sprzętowo jest matematycznie nierozwiązywalny bądź też praktycznie niemożliwy do rozwiązania. $\Rightarrow$
- Dlatego też w praktyce zadowalamy się technikami heurystycznymi dającymi dobry kod, ale nie zawsze optymalny.