# Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

Języki formalne i techniki translacji - Wykład 8

Maciek Gębala

27 listopada 2018



Notatki

## Zależności kontekstowe

- Zgodność programu źródłowego z regułami składniowymi i semantycznymi języka programowania.
- Kontrola zależności kontekstowych może być wykonywana
  - statycznie – podczas kompilacji programu
  - dynamicznie – podczas wykonywania programu wynikowego.



Notatki

## Kontrola statyczna

- Kontrola zgodności typów.
- Sprawdzenie przepływu sterowania (czy instrukcje zostały użyte w dopuszczalny sposób).
- Sprawdzenie unikalności nazw deklaracji (czy użyte obiekty są poprawnie zadeklarowane).
- Sprawdzenie powtórzeń nazw (czy nazwy są poprawnie powtórzone).



Notatki

## System typów

- System typów jest określany w definicji języka.
- System typów języka może być również określony przez system reguł wiązania typów z odpowiednimi regułami języka (patrz SML).
- Zgodność programu z systemem typów języka jest sprawdzana za pomocą kontrolera typów.
- System typów jest nazywany silnym jeśli nie ma potrzeby sprawdzania typów w trakcie działania programu wynikowego.
- Język jest nazywany silnie utypowionym jeśli ma silny system typów.



Notatki

## Wyrażenia określające typy

- Typy podstawowe wbudowane w język – wyrażenia w gramatyce języka.
- Konstruktory typów – reguły pozwalające na tworzenie nowych typów na podstawie już istniejących.
  - Zestaw konstruktorów zależy od języka źródłowego.
  - Typowe konstruktory typów konstruują typy takie jak tablice, rekordy, wskaźniki, funkcje, procedury.



Notatki

## Wyrażenia określające typy – funkcje

- Funkcje (procedury, podprogramy) są podstawową metodą strukturalizacji programów.
- Funkcja przekształca elementy dziedziny (iloczyn kartezjański argumentów) w elementy zbioru wartości funkcji.
- Typ funkcji jest wyrażeniem zawierającym iloczyn kartezjański dziedziny i typ wartości funkcji.



Notatki

## Kontroler typów – tablica symboli

- Tablica symboli zawiera informacje o każdym użytym symbolu.
- Informacje przechowujemy w postaci atrybutów takich jak: typ, zasięg występowania, wartość, itp.
- Reprezentacja tablicy musi być efektywna czasowo i pamięciowo.
- Musi zostać zapewniona efektywna obsługa blokowych konstrukcji języka (konteksty lokalne).



Notatki

## Kontroler typów - operacje

- Deklaracja zmiennej – dodajemy nowy symbol do tablicy z typem czy zasięgiem występowania.
  (Błędy: brak miejsca w tablicy, redeklaracja zmiennej.)
- Wyrażenia stałe – rekonstrukcja typu.
- Identyfikator – sprawdzenie atrybutów w tablicy typów.
- Operator – sprawdzenie dla operatora czy typy jego argumentów odpowiadają typowi operatora.
- Tablica – sprawdzenie czy indeks jest odpowiedniego typu.
- Wskaźnik – czy prawidłowo użyto wskaźnika, rekonstrukcja jego typu.



Notatki

## Kontroler typów - instrukcje

- Sprawdzenie czy warunki logiczne są rzeczywiście typu logicznego.
- Sprawdzenie czy przypisania są możliwe ze względu na system typów.
- Uwzględnienie niejawnych konwersji typów.



Notatki

## Problemy

- Przeciążanie funkcji i operatorów.
- Funkcje polimorficzne.
- Rekonstrukcje typów.



Notatki

## Faza syntezy

- Generowanie kodu pośredniego.
- Optymalizacja kodu.
- Generowanie kodu wynikowego.



Notatki

## Po co język pośredni?

- Załóżmy że mamy $n$ języków wysokiego poziomu i $k$ rodzajów maszyn docelowych.
- Pisząc kompilatory bezpośrednie musimy stworzyć $nk$ różnych implementacji.
- Korzystając z kodu pośredniego piszemy $n$ przodów kompilatora i osobno $k$ tyłów kompilatora.
- Łatwiej też wprowadzać nowe języki i nowe rodzaje maszyn.



Notatki

## Języki pośrednie

Języki pośrednie mogą być:

- zorientowane na maszynę docelową (uwzględniające fizyczne właściwości maszyn: liczba rejestrów, organizacja pamięci, współbieżność);
- zorientowane na język programowania (np. bytecode Java-y);
- uniwersalne (np. kod trójadresowy – liniowa reprezentacja drzewa wyprowadzenia, prosty język podobny do języka maszynowego).



Notatki

## Kod trójadresowy - instrukcje

- Instrukcje przypisania z wykorzystaniem binarnego operatora $x \leftarrow y \circ z$;
- Instrukcje przypisania z wykorzystaniem unarnego operatora $x \leftarrow \circ y$;
- Instrukcje kopiowania $x \leftarrow y$;
- Skoki bezwarunkowe goto L;
- Skoki warunkowe if $x \sim y$ goto L;
- Przypisania indeksowane $x \leftarrow y[i]$ lub $x[i] \leftarrow y$;
- Przypisania z użyciem adresów i wskaźników $x \leftarrow \&y$, $x \leftarrow *y$ lub $*x \leftarrow y$;



Notatki

## Kod trójadresowy - instrukcje

Instrukcje związane z podprogramami:

- Przekazanie parametru param $x$;
- Wywołanie podprogramu call P, $n$;
- Powrót z procedury return;
- Powrót z funkcji ze zwróceniem wartości return $y$.

Podprogramy wymagają implementacji stosu na gromadzenie argumentów i wywołań.



Notatki

## Przykład przekładu

**Język źródłowy**

```
for i:=n downto 1 do a[i]:=i;
```

**Trójadresowy kod pośredni**

$i \leftarrow n$
loop:
if $i < 1$ goto end
$t2 \leftarrow 2 * i$
$a[t2] \leftarrow i$
$i \leftarrow i - 1$
goto loop
end:



Notatki

## Generowanie kodu wynikowego

- Ostatnią fazą kompilatora jest generator kodu wynikowego.
- Z kodu pośredniego wytwarzany jest równoważny kod wynikowy.
- Kod wynikowy musi być poprawny i efektywny.
- W analizie algorytmów generowania i optymalizacji kodu wynikowego pomocne są grafy przepływu.
- Najważniejsze problemy przy generowaniu kodu wynikowego to wybór rozkazów i przydział rejestrów.



Notatki

## Graf przepływu

- Blok podstawowy – fragment kodu wykonywany zawsze od pierwszej do ostatniej instrukcji.
- Graf przepływu – wierzchołkami są bloki podstawowe, krawędzie skierowane odzwierciedlają przepływ sterowania.

**Przykład**

$x \leftarrow a$
$y \leftarrow b$
W: if $x = y$ goto F
if $x < y$ goto E
$x \leftarrow x - y$
goto W
E: $y \leftarrow y - x$
goto W
F: stop



Notatki

## Przydział rejestrów

- Właściwy przydział rejestrów ma ogromne znaczenie dla efektywności działania programu wynikowego.
- Operacje na rejestrach są zawsze szybsze ale rejestrów jest niewiele.
- Rejestry mogą być różnego rodzaju – dedykowane do określonych operacji.



Notatki

## Optymalizacja kodu

- Optymalizacja kodu to przekształcenie mające na celu poprawę efektywności kodu wynikowego.
- Celem optymalizacji jest uzyskanie szybszego lub mniejszego kodu.
- Optymalizacja opiera się na analizie przepływu sterowania i przepływu danych.
- Optymalizacja musi zachować semantykę programu.
- Optymalizacja powinna być opłacalna – zauważalny efekt powinien odpowiadać nakładowi pracy.



Notatki

## Optymalizacja kodu

**Ulepszenia mogą być wykonane na poziomie:**

- kodu źródłowego – profilowanie programu, przekształcenia pętli;
- reprezentacji pośredniej – ulepszanie pętli i wywoływania procedur, obliczanie adresów;
- kodu wynikowego – przydział rejestrów, idiomy maszynowe, szeregowanie instrukcji.

Optymalizacja może być lokalna (na poziomie bloków podstawowych) lub globalna (ponad blokami podstawowymi).



Notatki

## Optymalizacja kodu

- Idiomy maszynowe:
  `mov eax 0` zastępujemy `xor eax eax`;
- Tożsamości algebraiczne: $0 \cdot x \iff 0$, $1 \cdot x \iff x$;
- Redukcja siły operatora: $2 \cdot x \iff x + x$, $8 \cdot x \iff x >> 3$;
- Zwijanie stałych: `2*PI` zastępujemy przez `6.28`;
- Eliminacja wspólnych podwyrażeń;
- Usuwanie martwego kodu.



Notatki

## Optymalizacja kodu

- Jeżeli jakieś wyrażenie w pętli nie zmienia wartości w trakcie przebiegu pętli to kod obliczający wartość tego wyrażenia można wyciągnąć przed pętlę.
- Pętlę można rozwinąć jeżeli wykonanie jej bez skoków zajmuje mniej miejsca (pętle o stałym niewielkim przebiegu).
- Funkcje wykonujące niewielkie obliczenia można zamiast wywoływać wkleić bezpośrednio do kodu (rozwijanie funkcji).



Notatki

## Środowisko czasu wykonania

- Dostęp do nazw nielokalnych.
- Przekazywanie parametrów.
- Dynamiczny przydział pamięci.



Notatki

## Zmienne nielokalne

- Problem dostępu do zmiennych nielokalnych występuje w językach dopuszczających wielokrotne zagnieżdżanie podprogramów i deklarowanie zmiennych lokalnych.
- Problemem jest określenie która zmienna o tej samej nazwie jest właściwa: trzeba określić zakresy widzialności.



Notatki

## Parametry podprogramów

**Tryby przekazywania parametrów**

- wejściowy (IN) – parametr zawiera wartość stałej, zmiennej lub wyrażenia;
- wyjściowy (OUT) – program zwraca za pośrednictwem wyrażenia wartość;
- wejściowo-wyjściowy (IN-OUT) – program zarówno otrzymuje jak i zwraca wartość za pośrednictwem parametru.

**Metody przekazywania parametrów:**

przez wartość, przez referencję, przez rezultat, przez kopiowanie i odtwarzanie, przez nazwę.



Notatki

## Przekazywanie parametru przez wartość

- Najprostsza metoda przekazywania parametrów.
- Wykorzystywana do przekazywania parametrów tylko w trybie wejściowym.
- Domyślna w większości języków.
- Podprogram ma dostęp tylko do kopii zmiennych (brak efektów ubocznych).
- W przypadku dużych obiektów metoda bardzo kosztowna.



Notatki

## Przekazywanie parametru przez referencję

- Wykorzystywana do przekazywania parametrów tylko w trybie wejściowo-wyjściowym.
- Parametr jest przekazywany jako wskaźnik (referencja) do miejsca przechowywania aktualnego parametru.
- Nie ma problemu z dużymi obiektami ale dostęp do parametrów jest wolniejszy (adresowanie pośrednie).



Notatki

## Przekazywanie parametru przez rezultat

- Wykorzystywana do przekazywania parametrów tylko w trybie wyjściowym.
- Dostępna w niewielu współczesnych językach (Ada), w innych zastępowana funkcjami lub referencjami.
- W wywołaniu parametr nie ma określonej wartości.
- Parametr musi być zmienną.



Notatki

## Przekazywanie parametru przez kopiowanie i odtwarzanie

- Hybryda metod przekazywania przez wartość i referencję.
- Po powrocie z podprogramu wartości są przepisywane.



Notatki

## Przekazywanie parametru przez nazwę

- Podprogram jest traktowany jako makro.
- Podprogram jest rozwijany tak jakby był w miejscu wywołania.
- Parametry są tekstowo wstawiane w miejsce parametrów formalnych.



Notatki

## Organizacja pamięci

Pamięć jest w czasie wykonywania programu dzielona na cztery obszary:

1. obszar kodu
2. obszar danych statycznych
3. stos
4. sterta

Obszary stosu i sterty zmieniają swój rozmiar w trakcie działania programu.



Notatki

# Zarządzanie pamięcią – sterta

- Dynamiczne przydzielanie pamięci.
- Zwalnianie pamięci
  - jawne – uruchamiane przez programistę (problemy z wiszącymi referencjami i niezwolnioną pamięcią (Memory Leak)).
  - niejawne – uruchamiane automatycznie (Garbage Collector).
- Fragmentacja i scalanie pamięci.
- Metody przydziału pamięci.



Notatki

Notatki

Notatki

Notatki