

Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

Języki formalne i techniki translacji - Wykład 8

Maciek Gębala

26 listopada 2019

Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

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.

Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

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).

Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

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.

Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

Notatki

Notatki

Notatki

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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).

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Kontroler typów - operacje

- Deklaracja zmiennej – dodajemy nowy symbol do tablicy z typem czy zasięgiem występowania.
(**Błędy**: brak miejsca w tablicy, redeclaracja 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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

Notatki

Notatki

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Problemy

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

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Faza syntezy

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

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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 przodków kompilatora i osobno k tyłów kompilatora.
- Łatwiej też wprowadzać nowe języki i nowe rodzaje maszyn.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

Notatki

Notatki

Notatki

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).

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 $*x \leftarrow y$;

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.

Przykład przekładu

Język źródłowy

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

Trójadresowy kod pośredni

```
i ← n
loop:
if i < 1 goto end
t2 ← 2 * i
a[t2] ← i
i ← i - 1
goto loop
end:
```

Generowanie kodu wynikowego

- Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

Graf przepływu

- Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

Przydział rejestrów

- Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

Optymalizacja kodu

- Maciek Gębala Zależności kontekstowe. Synteza kodu i środowisko czasu wykonywania

This image shows a full page of white paper with ten horizontal dashed lines, typical of primary school handwriting practice paper. The lines are evenly spaced and extend across the entire width of the page. There is no text or other markings on the paper.

[illegible][illegible]

This image shows a full page of primary-ruled paper. It features ten sets of horizontal lines, each consisting of a solid top line, a dashed middle line, and a solid bottom line, providing a guide for letter height and placement. The paper is otherwise blank, with no margins or additional markings.

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).

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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: $2 \cdot \pi$ zastępujemy przez `6.28`;
- Eliminacja wspólnych podwyrażeń;
- Usuwanie martwego kodu.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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 (pętla o stałym niewielkim przebiegu).
- Funkcje wykonujące niewielkie obliczenia można zamiast wywoływać wkleić bezpośrednio do kodu (rozwijanie funkcji).

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Środowisko czasu wykonania

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

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

Notatki

Notatki

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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).

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

Notatki

Notatki

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

Przekazywanie parametru przez kopiowanie i odtwarzanie

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

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

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.

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

Organizacja pamięci

Pamięć jest w czasie wykonywania programu dzielona na cztery obszary:

- 1 obszar kodu
- 2 obszar danych statycznych
- 3 stos
- 4 sarta

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

Maciek Gębala

Zależności kontekstowe, Synteza kodu i środowisko czasu wykonywania

Notatki

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

Notatki

[illegible]

Notatki

[illegible]

Notatki

[illegible]

Notatki

[illegible]