

Tentamen i kurser

TDDA 37/TDDB 44 Kompilatorkonstruktion

TDDA 28/TDDB 29 Kompilatorer och interpretatorer

2001-08-15 kl 14.00 - 18.00

Hjälpmedel: Inga.

Poänggränser: Max = 40 poäng, för godkänd krävs 20 poäng.

Jourhavande lärare: Jonas Wallgren, endast telefonjour: 08-6644638

Uppgift 1 (4p) Formella språk och automater

OBS! Denna uppgift är avsedd endast för TDDA 28/TDDB 29 (Kompilatorer och interpretatorer)!

Ge en DFA och ett reguljärt uttryck för språket över $\{0,1\}$ sådant att för varje sträng gäller att om den innehåller 01 (noll och ett i direkt följd) så måste den innehålla 10.

Uppgift 2 (4p) Symboltabell

Antag att vi har en hashkodad symboltabell, där namn lagras i en speciell strängarea. Givet nedanstående program:

```
PROGRAM Foo1;  
  VAR Ida, Ceasar: INTEGER;  
    PROCEDURE Foo2;  
      VAR Kristina, Ida: BOOLEAN;  
        PROCEDURE F003;  
          VAR Ida, Hejsan: REAL;  
(* L1 *)  
          BEGIN .... END;  
(* L2 *)  
          BEGIN ... END;  
        BEGIN ... END.  
    END  
END.
```

Visa hur länkad hashtabell, symboltabell (även innehållande typinformation) och blocktabell ser ut vid positionerna L1 och L2 under kompileringprocessen. Antag att alla namn har unika hashvärden.

Uppgift 3 (4p) Top-downparsning

När man ska anpassa en grammatik för recursive-descentparsning finns det två speciellt viktiga transformationer som utförs.

- Namnge och definiera transformationerna. Definitionerna skall vara generella - inte bara illustrerade genom exempel.
- Vilka problem är det man vill undvika genom att göra dessa transformationer? Förklara vad som händer i parsern om man inte gör dem, gärna med konkreta exempel.

Uppgift 4 (5p) SLR(1)-parsning

Givet nedanstående grammatik:

```
1. <S> ::= a <S> a  
2.      | <S> c  
3.      | c
```

- Konstruera den kanoniska samlingen av LR(0) items för ovanstående grammatik.
- Använd de tillstånd och tillståndsövergångar som fås mha a) och konstruera *action*- och *goto*-tabeller.
- Visa hur strängen *acca* parsas.

Uppgift 5 (6p): Mellankodsgenerering

```
IF a + b = 0
  THEN WHILE x < y DO
    BEGIN
      x := x + c;
      y := y - c;
    END
  ELSE a := - b;
```

- a) Översätt ovanstående sats till omvänd polsk notation.
- b) Översätt ovanstående sats till kvadrupler.
- c) Översätt ovanstående sats till abstrakt syntaxträd.

Uppgift 6 (5p) Syntaxstyrd översättning

Ge en grammatikregel för en if-sats i ett högnivåspråk.

Skriv de semantiska reglerna, ett syntaxstyrt översättningsschema, för översättning av if-satsen till kvadrupler. Antag att översättningsschemat ska implementeras i bottom-upparsningmiljö med en semantisk stack. Utgå från din grammatikregel ovan, som dock kanske måste modifieras.

Det är ej tillåtet att definiera och använda symboliska label, dvs alla hopp ska ske till en absolut kvadrupel adress. Förklara införda attribut och eventuella funktioner och instruktioner som används. Ange alla dina antaganden.

Uppgift 7 (4p): Kodoptimering

- a) Vad är en loop?
- b) Beskriv loopoptimeringsmetoderna som presenterades i kursen. Använd kodexempel.

Uppgift 8 (4p) Minneshantering

Vad innehåller en aktiveringspost? När används aktiveringsposter och hur används de?

Uppgift 9 (4p) Bootstrapping och kompilatorgenerering

Antag att vi har ett nytt, häftigt språk X som vi vill ha en kompilator för. Vi kan skriva en kompilator i X själv som genererar bra kod. Vad vi har tillgång till är en kompilator för språket P på maskinen M. P är inte tillräckligt kraftfullt för att kunna skriva en kompilator för X som genererar bra kod. Visa hur man kan göra för att konstruera en kompilator som körs på maskinen M och som genererar bra kod som kan köras på en annan maskin N. Beskriv vilka olika moment som måste utföras och i vilken ordning. Använd T-diagram.

Uppgift 10 (4p): Kodgenerering för RISC

OBS! Denna uppgift är avsedd endast för TDDA 37/TDDB 44 (Kompilatorkonstruktion)!

- a) Vad är branchprediktion och när används det? Ge exempel! Varför är det viktigt för pipelineade processorer?
- b) Förklara software pipelining. Visa med ett enkelt exempel