UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i :	INF5110
Eksamensdag :	Tirsdag 6. juni 2006
Tid for eksamen :	09.00 - 12.00
Oppgavesettet er på :	5 sider
Vedlegg :	Intet
Tillatte hjelpemidler :	Alle trykte og skrevne

Les gjennom ***hele*** oppgavesettet før du begynner å løse den første oppgaven. Dersom du savner opplysninger i oppgaven, kan du selv legge dine egne forutsetninger til grunn og gjøre rimelige antagelser, så lenge de ikke bryter med oppgavens "ånd". Gjør i så tilfelle rede for disse forutsetningene og antagelsene.

Oppgave 1

Det følgende er et fragment (dvs ikke-interessante deler av grammatikken er ikke tatt med) av en grammatikk for et språk med prosedyrer. Alle prosedyrer har én parameter, og den er enten ’by value’, ’by reference’ (keyword **`ref`**), eller ’by-value-result’ (keyword **`result`**).

```
procedure → proc id (param) stmt
param → type id | ref type id | result type id
call → id(exp)
exp → id
exp → id[exp]
exp → exp aritop exp
```

Følgende programmer erklærer begge en integer variabel `i` og en prosedyre `change`, og programmene assigner `1` til `i`, kaller prosedyren `change(i)` og skriver ut verdien av `i`. Forskjellen er at Program 1 har prosedyren med ’call by reference’, mens Program 2 har prosedyren med ’call by value result’. Språket følger vanlige statiske skopregler.

Program 1:

```
{
  int i;
  proc change(ref int p) {
    p=2; i=0;
  };
  i=1;
  change(i);
  write(i)
}
```

Program 2:

```
{
  int i;
  proc change(result int p) {
    p=2; i=0;
  };
  i=1;
  change(i);
  write(i)
}
```

1a

Anta at semantikken for ' by-value-result' er slik at adressen (location) til den aktuelle parameter beregnes ved prosedyrekallet (procedure entry).
Hva skriver Program 1 og Program 2 ut når de utføres?

1b

Anta at semantikken for ' by-value-result' er slik at adressen (location) til den aktuelle beregnes på nytt ved *avslutning* av prosedyrekallet (prosedyre exit).
Hva skriver Program 1 og Program 2 ut når de utføres?

1c

Den enkle reglen i dette språket er at prosedyrer med en parameter 'by reference' eller 'by-value-result' bare kan kalles med et uttrykk som enten er en enkel variabel (***id***) eller en indisert variabel (***id*****[***exp***]**).
Fyll ut de tomme felter i følgende attributtgrammatikk slik at attributtet *ok* for *call* er *true* hvis kallet er gjort ifølge denne regel, ellers *false*.
Symboltabellen er innrettet på akkurat denne reglen, slik at navn på prosedyrer er assosiert med en verdi som sier om denne prosedyre har en parameter 'by reference' (verdien *ref*), 'by value-result' (verdien *result*) eller 'by value' (*value*). *lookupkind(**id**.name)* gir verdien som svarer til hvordan prosedyren med navn ***id***.*name* er definert. Det er ikke behov for å sjekke om prosedyrenavnet (***id***) i en *call*-setning faktisk er deklarert.

Grammar Rule	Semantic Rule
procedure → **proc** ***id*** **(***param***)** *stmt*	*insert(**id**.name, param.kind)*
param → *type* ***id***	
param → **ref** *type* ***id***	
param → **result** *type* ***id***	
call → ***id*** **(***exp***)**	*call.ok =*
exp_1 → ***id*** **[**exp_2**]**	
exp → ***id***	
exp_1 → exp_2 *aritop* exp_3	

Oppgave 2

Betrakt følgende grammatikk G, hvor S og T er ikketerminal-symboler, # og a er terminalsymboler, og S er startsymbolet.

$S \rightarrow TS$
$S \rightarrow T$
$T \rightarrow \#T$
$T \rightarrow a$

a) Finn First og Follow-mengdene til T og S (og la $ betegne 'end-of-file' som i boka).

b) Formulér med dine egne ord hvilke sekvenser av terminalsymboler du kan lage ut fra S'.

c) Avgjør om du kan lage et regulært uttrykk som uttrykker disse sekvensene av # og a som du kan utlede fra S, og hvis svaret er 'ja', gi et slikt regulært uttrykk.

d) Innfør et nytt start-symbol $S' \rightarrow S$ og lag LR(0)-DFA-en for G rett fra denne grammatikken. Nummerér tilstandene.

e) Gi et kort argument som bestemmer hvike(n) av følgende fem grupper G hører med i:
   a. LR(1)
   b. LALR(1)
   c. SLR(1)
   d. LR(0)
   e. Ingen av de overstående.

   Hint: Finn ut hvilke mulige konflikter du har i DFA-en og/eller om grammatikken er entydig..

f) Lag parseringstabellen for G ut fra den typen grammatikk den er.

g) Vis hvordan setningen: "a#a" vil bli parsert ved å skrive opp, som i boka, stakk-innholdet og input for hver av skift- eller reduser-operasjonene du gjør under parseringen. Få også med numrene til tilstandene på stakken (som i boka).

Oppgave 3

3a

Anta ta vi har et språk med klasser og subklasser. Alle metoder er virtuelle, slik at de kan redefineres i subklasser.

Klassen `Graph` definerer (sammen med klassene `Node`, `Edge`) grafer som består av `Node`-objekter som er forbundet med `Edge`-objekter. Et objekt av klassen `Graph` representerer en graf. Alle noder i en graf antas å være forbundet via attributen `startNode`, som er en referanse til ett `Node`-objekt.

Deler av klassedefinisjoner som ikke er signifikante for oppgaven er antydet med `'...'`

```
class Node{ ... }
class Edge{ ... }

class Graph {
  Node startNode;
  void connect(Node n1,n2) {
     ... connects two Nodes by creating an Edge-object ...};
}
```

De følgende klasser definerer subklasser (`City` og `Road`) til henholdsvis `Node` og `Edge`, en subklasse (`RoadAndCityGraph`) til `Graph`, og en subklasse (`TravelingSalesmanGraph`) til `RoadAndCityGraph`. Metoden `display` vil tegne grafen med utgangspunkt i `startNode`.

```
class City extends Node {
  String name;
  ...
}

class Road extends Edge {
  String name;
  int distance;
  ...
}


class RoadAndCityGraph extends Graph {
  String country;
  void connect(Node n1,n2) {
    ... connects two City objects (treated as Nodes),
        by creating a Road object ... };
  void display() {
    ... displays Roads and Cities with names...};
}


class TravelingSalesmanGraph extends RoadAndCityGraph {
  void display() {
    ... displays Cities with names,
        and Roads with name and distance ... };
}
```

Vis hvordan objekter av klassene `Graph`, `RoadAndCityGraph` og `TravelingSalesmanGraph` vil være strukturert (layout) og tegn virtuell-tabellen for hvert av objektene. Bruk navn av formen <klassenavn>::<metodenavn> til å angi hvilken definisjon som gjelder for hvert objekt.

3b

Anta at klassene `Node` og `Edge` er definert som indre klasser til klassen `Graph`, og at slike indre klasser kan redefineres i subklasser, på samme måte som virtuelle metoder. Vi kan altså snakke om at indre klasser er virtuelle klasser. Redefinerte klasser blir automatisk subklasser av de tilsvarende virtuelle klasser; f.eks. vil klassen `Node` i `RoadAndCityGraph` være en subklasse til klassen `Node` i `Graph`.

```
class Graph {
  class Node{ ... }
  class Edge{ ... }

  Node startNode;
  void connect(Node n1,n2) {
     ... connects two Nodes by creating an Edge-object ...};
}

class RoadAndCityGraph extends Graph {
  class Node {
    String name;
    ...
  }
  class Edge {
    String name;
    int distance;
    ...
  }

  String country;

  void connect(Node n1,n2) {
    ... connects two Node objects by creating an Edge-object ... };
  void display() {
    ... displays Edges and Nodes with names...};
}

class TravelingSalesmanGraph extends RoadAndCityGraph {
  void display() {
    ... displays Edges and Nodes with names, and Edges with distance ... };
}
```

På samme måte som virtuell-tabellen for virtuelle metoder brukes ved kall på virtuelle metoder, så vil man nå også trenge en annen virtuell-tabell ved generering av objekter av virtuelle klasser. F.eks. innholder metoden `connect` i klassen `Graph` en generering av et `Edge`-objekt. Hvis denne metoden kalles i et `RoadAndCityGraph`-objekt, skal man generere et `Edge`-objekt slik det er definert i klassen `RoadAndCityGraph`.

Vis hvordan en slik virtuell-tabell for virtuelle klasser kan se ut. Ikke ta med virtuell-tabellen fra spørsmål 3a.

Forklar hvordan man bruker virtuell-tabellen ved utførelse av `new Edge()` i metoden `connect` i klassen `Graph`.

---------- o 0 o ----------

Slutt på oppgavesettet – lykke til!

Arne Maus og Birger Møller-Pedersen