Technische Hochschule Darmstadt

Praktische Informatik

Dr. G. Snelting

Klausur "Syntax und Semantik von Programmiersprachen"

26.2.91

Name, Vorname, Matrikelnummer:_____

Fachbereich, Fachsemester:_____

Die Klausur soll Prüfungsleistung sein im Bereich

Informatik I (Theoretische Informatik) / Informatik II (Praktische Informatik)
(Nichtzutreffendes streichen)

Falls "Informatik I" gewünscht ist, ist Aufgabe 5a zu bearbeiten. Falls "Informatik II" gewünscht ist, ist stattdessen Aufgabe 5b zu bearbeiten.

Zugelassene Hilfsmittel: Skript, eigene Unterlagen.

Die Klausur umfaßt 11 Seiten. Bitte überprüfen Sie Ihr Exemplar auf Vollständigkeit.

Aufgabe	max. Punkte	Punkte
1	20	
2	15	
3	25	
4	20	
5a	20	
5b	20	
Σ	100	

Aufgabe 1. Beim in Algol–60 verwendeten Parameterübergabeverfahren "call by name" wird bei jeder Verwendung eines formalen Parameters im Rumpf einer Prozedur der entsprechende aktuelle Parameter neu ausgewertet, und zwar in der Umgebung des Rufers. Dazu betrachten wir das Beispielprogramm (nach D. Knuth)

```
BEGIN
  INTEGER i, s;
  INTEGER ARRAY a[1:10, 1:10];
  PROCEDURE p(x,y,z); INTEGER x,y,z;
    COMMENT x,y,z sind Parameter, keine lokalen Variablen;
    BEGIN
      z:=0;
      FOR x:=1 STEP 1 UNTIL 10 DO
        z:=z+y
    END;
  ...
  p(i,a[i,i],s);
  p(i,a[1,i],s);
END;
```

a) Was berechnen die beiden Prozeduraufrufe?

b) Realisiert wird call-by-name, indem beim Aufruf ein sog. *thunk* übergeben wird, das ist eine Funktion, die bei jedem Zugriff auf den formalen Parameter dessen Adresse berechnet. Prozeduren sollen der Einfachheit halber nur einen Parameter haben und nicht rekursiv sein; ferner soll natürlich statische Bindung verwendet werden. Der aktuelle Parameter sei eine Variable. Geben Sie entsprechende Bedeutungsfunktionen der Standardsemantik für Prozedurdeklaration, Prozeduraufruf und Parameterzugriff an! Ergänzen Sie dazu die Vorgaben auf der nächsten Seite.

$env = Id \xrightarrow[m]{} (descr \times entry)$

$descr = ... \mid$ 'name-parm'

$entry = ... \mid$ {Bitte Einträge für Prozeduren und name-parameter ergänzen:}

$V : decl \rightarrow env \rightarrow env$

$V[\![$PROCEDURE $\mathrm{Id}_1(\mathrm{Id}_2)$;stmt;$]\!] =$

$S : stmt \rightarrow env \rightarrow store \rightarrow store$

$S[\![$CALL Id(var)$]\!] =$

$L : var \rightarrow env \rightarrow store \rightarrow loc$

$L[\![\mathrm{Id}]\!] = \lambda env.\lambda store.$ let $(descr, entry) = env(\mathrm{Id})$ in
if $descr \neq$'name-parm' then ... {interessiert hier nicht}
else {bitte ergänzen}

Aufgabe 2. Es soll ein Codeerzeugungsschema für die WITH-Anweisung hergeleitet werden. Dieses hat die Syntax *WITH var DO stmt* und es sei vorausgesetzt, daß *var* eine Record-Variable ist. Die WITH-Anweisung eröffnet bekanntlich einen neuen Gültigkeitsbereich, in dem die Record-Komponenten wie normale Variable direkt angesprochen werden können. Dazu wird die Adresse von *var* in ein freies Display-Register geladen. Dieses zeigt dann auf so etwas wie einen "activation record", nämlich die Record-Variable. Die Adressierung der Record-Komponenten innerhalb des WITH kann genauso erfolgen wie normale Variablenadressierung in Prozeduren, wobei als Relativadressen für die unmittelbar ansprechbaren Komponenten die im Environment vermerkten Offsets dieser Komponenten verwendet werden.

a) Geben Sie das Codeerzeugungsschema an. Beachten Sie, daß nach dem oben gesagten der Rumpf in einem geänderten Environment ausgewertet werden muß.

$compile(\text{"WITH var DO stmt"}, env) =$

b) Nun sei folgendes Programmfragment gegeben:

```
VAR a: RECORD a,b: INTEGER END; ...
WITH a DO a:=b+1;
```

Welcher Code (IBM 360) wird für die WITH Anweisung erzeugt? Die Deklaration sei auf statischer Tiefe drei, und das Record habe Relativadresse 136. Ein Integer belege 4 Byte, und Display-Register seien von 1 aufsteigend verwendet. Freie Register können zur Expression-Auswertung verwendet werden.

Aufgabe 3. In Modula-2 gibt es Prozedurvariablen, die etwa wie folgt deklariert werden:

```
VAR p: PROCEDURE (INTEGER, REAL);
```

Man kann solchen Prozedurvariablen eine Prozedurkonstante zuweisen, und sie dann wie eine Prozedurkonstante aufrufen.

a) Geben Sie den Code (IBM-Assembler) an, der als Übersetzung der Zuweisung "p:=q" erzeugt wird. p sei eine Prozedurvariable der statischen Tiefe 2, q sei eine Prozedurkonstante der statischen Tiefe 3. Aufbau von Display und Activation Records sei wie in der Vorlesung beschrieben.

b) Warum verlangt Wirth im Modula-2 Report, daß jene Prozedurkonstanten, die einer Prozedurvariablen zugewiesen werden können, nicht lokal zu anderen Prozeduren deklariert sein dürfen?

c) In manchen Programmiersprachen gibt es sog. *Exceptions* zum Behandeln von Ausnahmesituation, z.B. Division durch Null. In dieser Aufgabe soll die Realisierung eines einfachen Exception-Mechanismus hergeleitet werden. Dazu führen wir neue syntaktische Konstrukte ein:

decl = ... | EXCEPTION Id | ON Id DO stmt stmt = ... | RAISE Id

"EXCEPTION Id" deklariert eine Exception, dies muß geschehen, bevor sie in "ON ..." oder "RAISE Id" verwendet werden kann. "ON Id DO stmt" aktiviert einen sog. Exception-Handler, dessen Rumpf ausgeführt wird, wenn die Ausnahmesituation "Id" eintritt. Exceptions werden normalerweise vom Betriebssystem erzeugt (z.B. als Folge eines Hardware-Interrupts), können aber auch manuell ausgelöst werden. Dazu dient die Anweisung "RAISE Id", die den Ausnahmezustand herbeiführt und zur Aktivierung des Handlers führt. Zu einer Exception kann es beliebig viele Handler geben. Beim Auslösen einer Exception wird der Rumpf des zuletzt für diese Exception aktivierten Handlers ausgeführt, danach wird nicht etwa hinter die RAISE-Anweisung zurückgekehrt, vielmehr wird die Prozedur, in der der Handler deklariert war, sofort verlassen.

Beispiel:

```
MODULE m;
EXCEPTION overflow;
...
PROCEDURE push(s: stack; x: stackitem);
  BEGIN
    ...
    IF s.size > max THEN RAISE overflow FI;
    ...
  END push;
...
PROCEDURE test;
  VAR
    x: stack; y: stackitem;
  ON overflow DO
    writeln(y, ' passt nicht mehr in den Keller');
  BEGIN
    ...
    push(x,y);
    ...
  END test;
BEGIN
  ...
  test;
  ...
END m.
```

Falls jemals "RAISE overflow" ausgeführt wird, wird die Abarbeitung nach Ausgabe der Fehlermeldung im Hauptprogramm hinter dem Aufruf von "test" fortgesetzt.

Geben Sie nun eine Source-to-Source Transformation an, die Programme mit Exceptions in Programme ohne Exceptions übersetzt! Hinweis: Verwenden Sie Prozedurvariablen. Nehmen Sie ferner an, es gäbe in Modula-2 eine GOTO-Anweisung.

transform("EXCEPTION Id") =

transform("RAISE Id") =

transform("PROCEDURE Id_1; declarations; ON Id_2 DO stmt; BEGIN statements END") =

Aufgabe 4. Gegeben sei die kontextfreie Grammatik

S ::= L "=" R | R

L ::= "*" R | id

R ::= L

a) Geben Sie für jedes Nichtterminal die First- und Follow-Mengen an.

b) Konstruieren Sie den LR(0)-Automaten.

c) Ist die Grammatik SLR(1)? Untersuchen Sie für jeden Konflikt im LR(0)-Diagramm seine Auflösbarkeit.

Aufgabe 5a.

Es gibt einen interessanten Zusammenhang zwischen axiomatischer Semantik und Continuation-Semantik, der in dieser Aufgabe bewiesen werden soll.

a) Geben Sie zunächst die Continuation-Semantik für Zuweisung, zusammengesetzte Anweisung und bedingte Anweisung an (für Expressions soll Standardsemantik verwendet werden, keine Expression Continuations). Geben Sie ferner die "weakest precondition" $wp(stmt, P)$ für diese Anweisungen an. Vor- und Nachbedingungen seien Prädikate, also Ausdrücke $\in expr$, von denen unterstellt wird, daß ihre Auswertung einen booleschen Wert ergibt.

$$env = Id \xrightarrow[m]{} loc, \quad cont = store \to store, \quad store = loc \xrightarrow[m]{} val$$

$$C : stmt \to env \to cont \to cont$$

$$E : expr \to env \to store \to val$$

$$wp : stmt \to expr \to expr$$

$C[\![Id := expr]\!] =$

$C[\![stmt_1; stmt_2]\!] =$

$C[\![\text{IF } expr \text{ THEN } stmt_1 \text{ ELSE } stmt_2]\!] =$

$wp(Id := expr, P) =$

$wp(stmt_1; stmt_2, P) =$

$wp(\text{IF } expr \text{ THEN } stmt_1 \text{ ELSE } stmt_2, P) =$

b) Wir führen nun in Abänderung von a) spezielle Continuations ein: $cont = store \to val$, lassen aber sonst die Bedeutungsfunktion C unverändert. Überzeugen Sie sich, daß C nach wie vor wohldefiniert ist, und beweisen Sie, daß folgende Gleichung gilt:

$$C[\![stmt]\!](e)(E[\![P]\!](e)) = E[\![wp(stmt, P)]\!](e)$$

für alle Prädikate p und alle Environments e. Hinweis: Sie benötigen eine triviale Verallgemeinerung eines in der Vorlesung bewiesenen Lemmas, das eine Aussage über $E[\![expr_1(Id \leftarrow expr_2)]\!]$ macht. Geben Sie zuerst das Lemma an.

Aufgabe 5b. Wir betrachten noch einmal die Grammatik aus Aufgabe 4.

a) Geben Sie eine äquivalente LL(1)-Grammatik an.

b) Geben Sie eine abstrakte Syntax sowie die Transformation konkrete Syntax→abstrakte Syntax an.

c) Das strukturierte Terminal "id" sei wie üblich durch folgenden regulären Ausdruck beschrieben:

id = letter (letter|digit)*

wobei "letter" und "digit" wie gewohnt definiert sind. Ferner soll als zusätzliche Einschränkung gelten: ein Bezeichner darf nicht die Buchstaben "HJH" enthalten (in dieser Reihenfolge; also z.B. "AHBJCH" ist verboten, aber "JHHX" ist erlaubt). Geben Sie einen minimalen deterministischen endlichen Automaten an, der die so eingeschränkten Bezeichner erkennt!