Test
Übersetzer
2. Halbzeit
WS 97/98

Bitte benutzen Sie die Rückseite der Blätter für Ihre Notizen. Bearbeiten Sie jede Aufgabe nur auf den Zetteln, die für diese Aufgabe vorgesehen sind.

Bitte vergessen Sie nicht, Ihren Namen, Ihren Vornamen und Ihre Matrikel-Nummer einzutragen!

Viel Erfolg !

Name:

Vorname:

Matrikel-Nummer:

e-Mail:

Inhaltsverzeichnis

Aufgabe	1	2	3	4	Σ
Punkte	10	10	10	10	40
Erreichte Punkte					

 Name:

Aufgabe 1: (10 Punkte) **Kurzfragen** **erreicht: ( )**

Beantworten Sie die folgenden Fragen nur in Stichworten. Es werden keine ausführlichen Erklärungen erwartet.

Semantische Analyse

a) Warum können die in der Sprache *C* benutzten *Gültigkeitsregeln* für *Variablenbezeichner* nicht auch für *Sprungmarken* benutzt werden? → 1P

```
Dann wären keine Vorwärtssprünge möglich oder Sprungmarken
müssten vorab deklariert werden können. Sprünge in andere Blöcke
hinein wären ebenfalls unmöglich.
```

b) Der *Umgebungsmodul* legt für jeden Bezeichner einen *Keller* an. Geben Sie einen Ausschnitt aus einem C-Programm an, bei dessen Analyse (zeitweilig) 3 Einträge auf dem Keller für den Bezeichner `a` liegen. → 1P

```
{ int a; { int a; { int a; } } }
```

c) Aus welchen Informationen wird bei der *Operatoridentifikation* der *Zieloperator* ermittelt? → 1P

```
aus dem Quelloperator und den Typen der Operanden (sowie den
zulässigen Coercions)
```

d) Ergänzen Sie die folgende Grammatikproduktion um eine *Attributberechnung*, die die LAG(1)-Eigenschaft der attributierten Grammatik zerstört. → 1P

```
RULE: A ::= B C    COMPUTE

B.a = f(C.b);
/* oder */
B.a = f(B.b);

END;
```

Optimierung

e) Warum ist es sinnvoll, nach dem *Entfernen nicht erreichbaren Codes* das Programm (erneut) auf *überflüssige Zuweisungen* zu untersuchen? → 1P

```
Falls der entfernte Code die letzte Verwendungsstelle einer
Variablen v enthielt, ist jetzt jede Zuweisung an v überflüssig
geworden und kann entfernt werden.
```

Aufgabe 1: (Fortsetzung)

f) Welches *Datenflußproblem* muß gelöst werden, bevor *Konstantenweitergabe* stattfinden kann? → 1P

```
erreichende Definitionen
```

g) Kreuzen Sie die zutreffenden Charakterisierungen der folgenden beiden Probleme aus der Datenflußanalyse an. → 1P

	verfügbare Ausdrücke	lebendige Variablen
Vorwärts-Problem	×	
Rückwärts-Problem		×
Existenz-Problem		×
All-Problem	×	

h) Was sagt ein *Zyklus* in einem *Aufrufgraphen* über das untersuchte Programm aus? → 1P

```
Es existiert eine direkt oder indirekt rekursive Prozedur bzw.
Funktion.
```

Code-Erzeugung

i) Warum enthält eine *Prozedurschachtel* für C keinen Verweis auf den *statischen Vorgänger*? → 1P

```
Weil es in C keine geschachtelten Prozeduren/Funktionen gibt.
```

j) Geben Sie in der Notation von Folie U-62 eine *Code-Sequenz* für folgende *Ablaufstruktur* an. → 1P

```
do { S } while (B);

M1:
    Code (S)
    Code (B, true, M1)
```

 Name:

Aufgabe 2: (10 Punkte) **Attributierte Grammatiken**

erreicht: ( )

Folgende abstrakte Grammatik beschreibt eine blockstrukturierte Minisprache. In dieser Sprache gelten für die Bezeichneridentifikation die Algol-Regeln.

```
   Program ::= Block
     Block ::= Def Stmt
       Def ::= Def Def
       Def ::= DefinedId
      Stmt ::= Stmt Stmt
      Stmt ::= Block
      Stmt ::= AppliedId ':=' AppliedId
 DefinedId ::= ident
 AppliedId ::= ident
```

a) Es soll eine Warnung ausgegeben werden, falls ein definierend auftretender Bezeichner eine andere Definition gleichen Namens verdeckt (obwohl das natürlich keinen Fehler darstellt). Dazu wird im Kontext der Produktion `DefinedId ::= ident` die Funktion `KeyInEnv` des Umgebungsmoduls benutzt und gegebenenfalls eine Meldung produziert. Geben Sie (in Worten) die genaue Vorbedingung für diese Überprüfung an. → 2P

`Alle Definitionen aus umgebenden Blöcken müssen bereits eingetragen worden sein. Definitionen aus dem aktuellen Block dürfen noch nicht eingetragen sein.`

b) Nun soll eine Warnung ausgegeben werden, falls ein definierter Bezeichner *nicht* benutzt wird. Welche Vorbedingung muß im Kontext der Regel `DefinedId ::= ident` gelten, bevor eine Warnung bezüglich dieses Bezeichners ausgegeben werden kann? → 1P

`Alle Benutzungen im aktuellen Block und den darin eingeschachtelten Blöcken müssen identifiziert worden sein (, ohne daß der hier definierte Bezeichner dabei benutzt wurde).`

Welche Eigenschaften und Attribute sind nötig, um die Vorbedingung zu gewährleisten, und die eigentliche Überprüfung vorzunehmen? → 2P

Eine Eigenschaft *Usage* mit den Werten *NotUsed* und *IsUsed* speichert für jeden definierten Bezeichner die Information, ob er benutzt wurde. Das Attribut *GotUsage* signalisiert die Bearbeitung einer einzelnen Bezeichneranwendung. Die Attribute *LocalDone* und *Done* registrieren die vollständige Abarbeitung des aktuellen Blocks (ohne bzw. einschließlich der eingeschachtelten Blöcke).

Blatt 4 Name:

Aufgabe 2: (Fortsetzung)

c) Vervollständigen Sie den vorgegebenen AG-Rahmen um Berechnungen, die Ihren Lösungsansatz aus Teil (b) realisieren. Gehen Sie davon aus, daß das Attribut `Key` für alle Bezeichnerauftreten gemäß Algol-Regeln berechnet wurde. Die Ausgabe der Meldung kann durch `message(WARNING, "Meldung")` erfolgen. → 5P

Erinnerung: Für jede Definition eines Bezeichners, der nicht benutzt wird, soll eine Warnung ausgegeben werden.

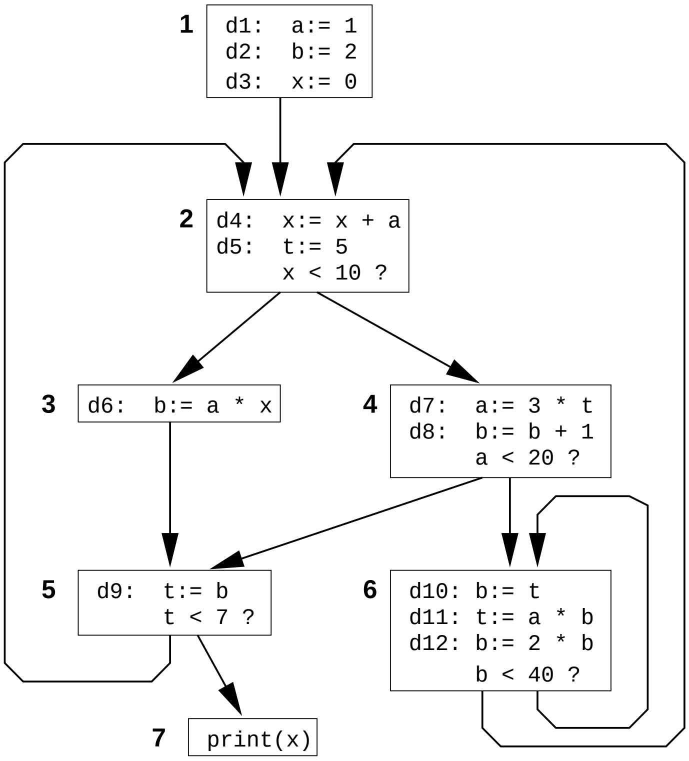
```
ATTR      Key: DefTableKey;  /* wurde bereits berechnet */
          GotUsage: VOID;    /* einzelne Bezeichneranwendung wurde registriert */
          LocalDone: VOID;   /* alle Bez.anwendungen im akt. Block bearbeitet */
          Done: VOID;        /* wie oben, aber inkl. eingeschachtelter Blöcke */


RULE p1 : AppliedId  ::=     ident      COMPUTE

          AppliedId.GotUsage = ResetUsage(AppliedId.Key, IsUsed);
END;


RULE p2 : DefinedId  ::=     ident      COMPUTE

          IF( EQ(GetUsage(DefinedId.Key, NotUsed), NotUsed),
              message(WARNING, "identifier is not used"))
                  <- INCLUDING Block.Done;
END;


RULE p3 : Block  ::=         Def   Stmt      COMPUTE

          Block.LocalDone = CONSTITUENTS AppliedId.GotUsage;
          Block.Done = CONSTITUENTS Block.Done <- Block.LocalDone;
END;
```

Blatt 5 Name:

Aufgabe 3: (10 Punkte) Optimierung erreicht: ( )

Iterative Datenflußanalyse

In dem folgenden Ablaufgraphen einer Prozedur sollen die *erreichenden Definitionen* bestimmt werden.



a) Geben Sie die für das Problem der *erreichenden Definitionen* passenden Datenflußgleichungen und die optimale Durchlaufreihenfolge der Grundblöcke im Ablaufgrafen an: → 1P

`Out(B) = Gen(B) ∪ (In(B) − Kill(B))`

`In(B) =` $\bigcup_{B' \in Vorg(B)}$ `Out(B')`

Durchlaufreihenfolge: `DFS-Ordnung (topologisch vorwärts)`

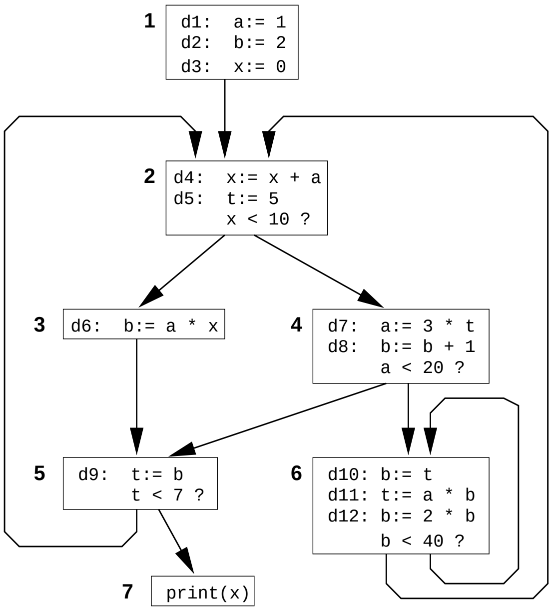
b) Bestimmen Sie die `Gen`- und `Kill`-Mengen sowie die initiale Belegung der `In`- und `Out`-Mengen. Führen Sie außerdem die *erste Iteration* des Lösungsverfahrens gemäß obiger Reihenfolge durch und berechnen Sie die resultierenden `In`- und `Out`-Mengen. → 4P
Hinweis: Geben Sie nur die Nummern der Definitionen an.

B	Gen(B)	Kill(B)	In(B)	Out(B)	In(B)	Out(B)
1	1,2,3	4,6,7,8,10,12	∅	1,2,3	∅	1,2,3
2	4,5	3,9,11	∅	4,5	1,2,3,9,11,12	1,2,4,5,12
3	6	2,8,10,12	∅	6	1,2,4,5,12	1,4,5,6
4	7,8	1,2,6,10,12	∅	7,8	1,2,4,5,12	4,5,7,8
5	9	5,11	∅	9	1,4,5,6,7,8	1,4,6,7,8,9
6	11,12	2,5,6,8,9	∅	11,12	4,5,7,8,11,12	4,7,11,12
7	∅	∅	∅	∅	1,4,6,7,8,9	1,4,6,7,8,9

Blatt 6 Name:

Aufgabe 3: (Fortsetzung)

Schleifenerkennung



c) Bestimmen Sie in dem bereits bekannten Ablaufgraphen die Rückwärtskanten. Geben Sie die in der Prozedur enthaltenen Schleifen als Mengen von Grundblöcken an. → 2P

Rückwärtskanten: $5 \rightarrow 2$, $6 \rightarrow 2$, $6 \rightarrow 6$
Schleifen:
$S_1 = \{2,5,3,4\} \cup \{2,6,4\} = \{2,3,4,5,6\}$
$S_2 = \{6\}$
Dabei ist S_2 in S_1 geschachtelt.

d) Skizzieren Sie ein C-Programm, das dem obigen Ablaufgraphen entspricht, und (möglichst) ohne `goto` auskommt. → 3P
Hinweis: Kürzen Sie die Anweisungen aus den Grundblöcken mit B_1, B_2, ..., B_7 ab.

```
{
   B1
   do {   B2
          if (x < 10) B3
          else {B4
                if (a < 20) {
                      do { B6 } while (b < 40);
                      continue;
                }
          }
          B5
   } while (t < 7);
   B7
}
```

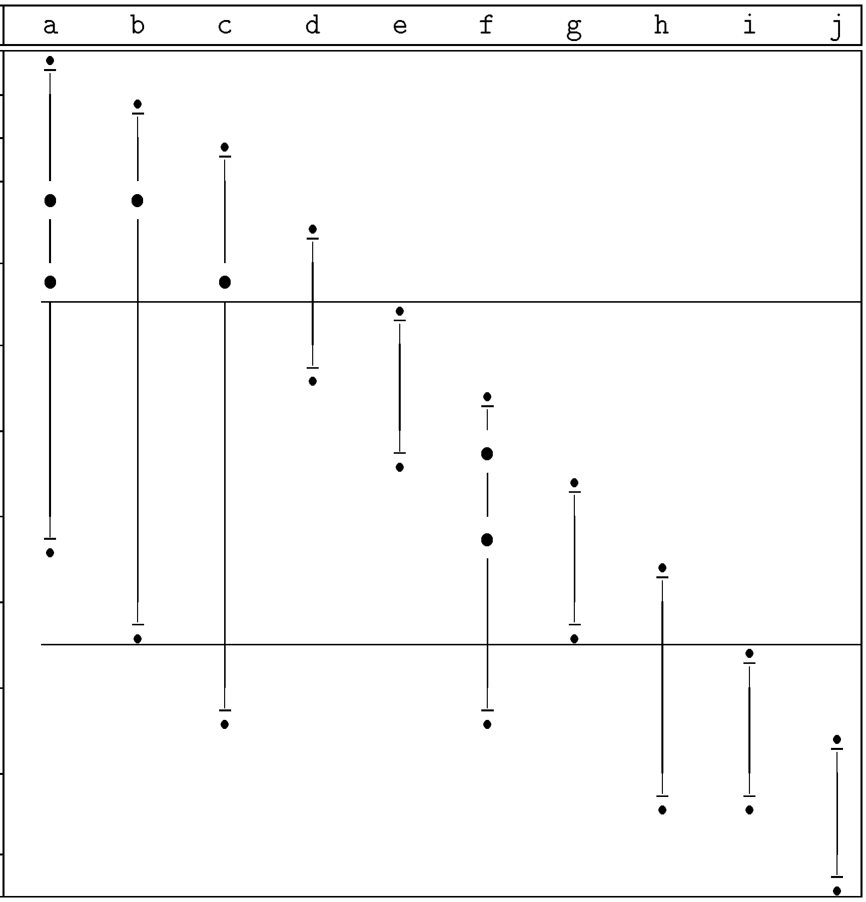
Blatt 7 Name:

Aufgabe 4: (10 Punkte) Registerzuteilung erreicht: ( )

a) Registerzuteilung im Grundblock:

Betrachten Sie den unten vorgegebenen Grundblock. Stellen Sie die Lebensdauer der Werte `a`, `b`, ..., `j` als Intervallgraph dar und bestimmen Sie den Registerbedarf. Bei `v`, `w`, ..., `z` handelt es sich um Variablen des Programms, die im Speicher gehalten werden. → 3P

```
a := v
b := w
c := x
d := a + b
e := a − c
f := d / 2
g := e * f
h := f − a
i := g + b
j := c − f
y := h * i
z := j
```



Registerbedarf: `5 Register (zwischen e := a − c und i := g + b)`

Nehmen Sie an, daß genau die benötigte Anzahl Register verfügbar ist, und ordnen Sie die Register den Werten `a`, `b`, ..., `j` zu. → 1P

Wert	a	b	c	d	e	f	g	h	i	j
Register	r1	r2	r3	r4	r5	r4	r5	r1	r2	r3

Welchen Wert würden Sie zwischenspeichern, falls ein Register zu wenig verfügbar wäre? Begründen Sie kurz Ihre Entscheidung. → 1P

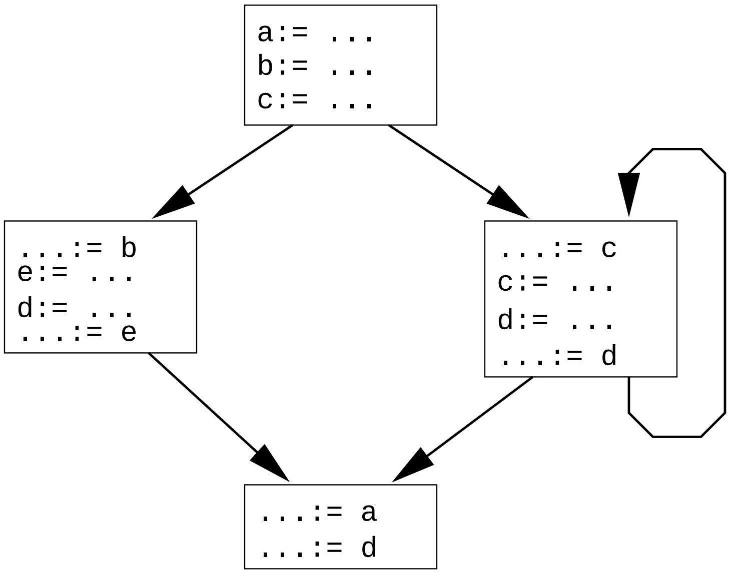
`Der Wert c sollte zum Zwischenspeichern gewählt werden, da er von allen geeigneten Werten am spätesten wiederverwendet wird, schon im Speicher steht und im Bereich des maximalen Schnitts nicht verwendet wird.`

Blatt 8 Name:

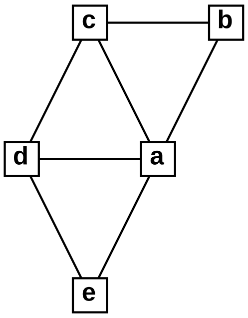
Aufgabe 4: (Fortsetzung)

b) Registerzuteilung durch Graphfärbung:

In dem folgenden Ablaufgraphen bedeutet `x:=...` eine Definition des symbolischen Registers `x` und `...:= x` eine Benutzung von `x`.



Erstellen Sie den Unverträglichkeitsgraphen und bestimmen Sie den minimalen Registerbedarf k. → 4P



Registerbedarf k**:** `3 Register`

Ordnen Sie die k physikalischen Register den symbolischen Registern `a`, `b`, …, `e` zu. → 1P

symb. Register	a	b	c	d	e
phys. Register	r1	r2	r3	r2	r3