Prüfung aus Übersetzerbau 19.6.1998 Musterlösung

1. 25 % Quadrupel-Code

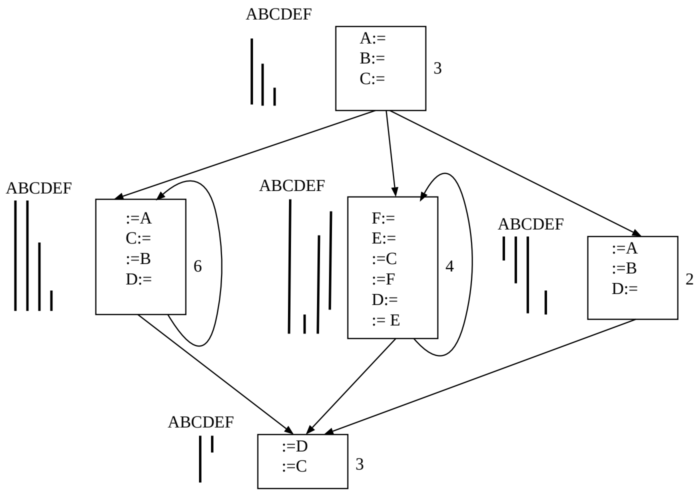
Erzeugen Sie für das rechte Programmstück Quadrupel-Code nach der Kontrollflußmethode. Ein `INTEGER` ist 4 Byte groß. Die Untergrenze aller Indexbereiche ist 0.

```
VAR
  a: ARRAY[30,10] OF INTEGER;
  b: ARRAY[20] OF INTEGER;
  i,j,s: INTEGER;
⋮
WHILE J<20 DO
  IF (i>=15) AND (j<10) THEN
    s := s + a[i,j] * b[j];
  ELSE
    s := s + b[j+10];
  END
j := j + 1;
END
```

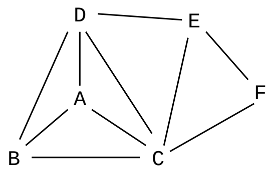
```
Wbegin:
    if j<20 goto Wtrue
    goto Wfalse
Wtrue:
    if i>=15 goto ANDtrue
    goto IFfalse
ANDtrue:
    if j<10 goto IFtrue
    goto IFfalse
IFtrue:
    t1 := i * 10
    t2 := t1 + j
    t3 := t2 * 4
    t4 := t3 + adr(a)
    t5 := @t4
    t6 := j * 4
    t7 := t6 + adr(b)
    t8 := @t7
    t9 := t5 * t8
    t10 := t9 + s
    s := t10
    goto IFend
IFfalse:
    t11 := j + 10
    t12 := t11 * 4
    t13 := t12 + adr(b)
    t14 := @t13
    t15 := t14 + s
    s := t15
IFend:
    t16 := j + 1
    j := t16
    goto Wbegin
Wfalse:
```

2. 25 %

Gegeben sei der folgende Kontrollflußgraph. Links von den Blöcken sind die Aktivitätsbereiche der Pseudoregister angegeben, rechts davon die erwarteten Ausführungshäufigkeiten.



a) (15 %) Geben Sie den Konfliktgraphen und die **Auslagerungskosten** für alle Pseudoregister an - dabei soll angenommen werden, daß ein Speicherbefehl einen Zyklus und ein Ladebefehl zwei Zyklen kosten soll.



	R:=	:=R	Kosten
A	3	8	19
B	3	8	19
C	9	7	23
D	12	3	18
E	4	8	12
F	4	4	12

$$Kosten = (\mathtt{R} :=) + (:= \mathtt{R}) * 2$$

b) (10 %) Bestimmen Sie eine Reihenfolge der **Registerbelegung**, belegen Sie die Pseudoregister mit realen Registern und kennzeichnen Sie die auszulagernden Pseudoregister. Nehmen Sie an, daß 3 reale Register zur Verfügung stehen.

Registerbelegung
Reihenfolge von links nach rechts

A	B	C	D	E	F
1	2	3	*	2	1

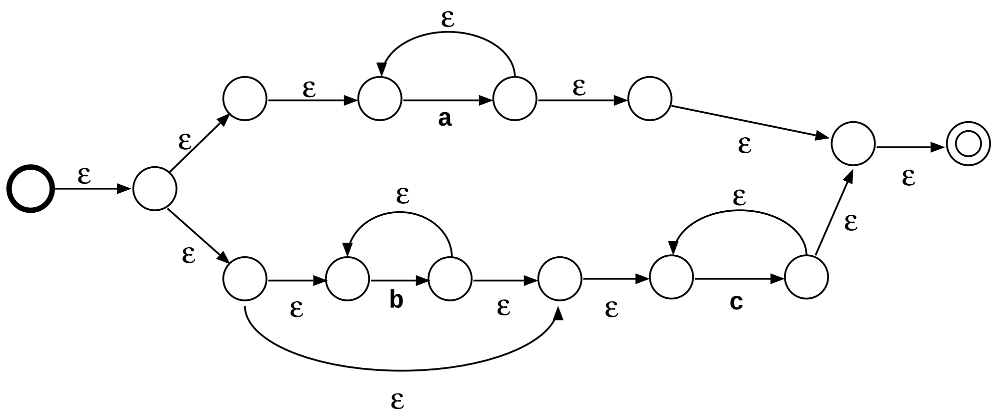
Das Register `D` muß ausgelagert werden weil seine Priorität mit $\frac{18}{3}$ am niedrigsten ist (`A`: $\frac{19}{3}$, `B`: $\frac{19}{3}$, `C`: $\frac{23}{3}$).

Anmerkung: Die obige Reihenfolge der Registerbelegung ist nur eine von mehreren möglichen Lösungen.

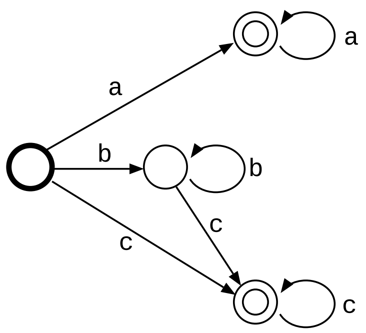
3. 25 % NFA und DFA

Gegeben sei der reguläre Ausdruck `a+|(b*c+)`

a) (10 %) Formen Sie den regulären Ausdruck in einen äquivalenten **NFA** mit ε -Kanten um. Verwenden Sie dazu die Umformungsregeln aus Kapitel 3 des Skriptums. (Eine Lösung ohne Verwendung des Algorithmus ist ungültig.)



b) (15 %) Erzeugen Sie aus dem NFA einen entsprechenden **DFA** und **Minimieren** Sie den entstandenen **DFA**, falls dies möglich ist.



4. 25 % Attributierte Grammatik

Ein Filesystem `FS` besteht aus einer Liste von Harddisks. Es gibt mindestens eine Harddisk. Nach dem Namen `name` einer Harddisk `HD` steht die Clustergröße `num` in Bytes, ein `":"` und danach die Directories. Jede Harddisk hat mindestens ein Directory. Ein Directory `D` hat einen Namen `name` und danach steht die Anzahl der Cluster `num` die es belegt. Die Directories sind durch `","` und die Harddisks durch `";"` getrennt, z.B.:

```
alpha 2048: bin 10, etc 7; beta 4096: ftp 20, pub 12; gamma 8192: www 35
```

a) (7 %) Erstellen Sie eine Grammatik die ein ganzes Filesystems einlesen kann. (Verwenden Sie dazu die Terminalsymbole `"name"`, `"num"`, `":"`, `","` und `";"` und ignorieren Sie etwaige Leerzeichen.)

FS	→	HDL
HDL	→	HDL `";"` `name` `num` `":"` DL
HDL	→	`name` `num` `":"` DL
DL	→	DL `","` `name` `num`
DL	→	`name` `num`

b) (18 %) Erweitern Sie die Grammatik aus a) um Attribute zur Berechnung des verbrauchten Platzes. Die Ausgabe soll so wie das Eingabeformat sein und im synthetisierten Attribut `"FS.x"` erfolgen. Alle Größen sind in Byte anzugeben. Im Attribut `"name.x"` steht der Name und in `"num.val"` der Zahlenwert des entsprechenden Terminalsymbols. Der Operator `"|"` hängt Zeichen- und Zahlen intelligent aneinander (`"gnu"|123` ergibt `"gnu 123"`). (Ignorieren Sie auch hier etwaige Leerzeichen.) Das obige Beispiel ergibt:

```
alpha 34816: bin 20480, etc 14336; beta 131072: ftp 81920, pub 49152; gamma
286720: www 286720
```

FS	→	HDL	FS.x := HDL.x
HDL	→	HDL `";"` `name` `num` `":"` DL	HDL0.x := HDL1.x | `";"` | `name`.x | DL.size | `":"` | DL.x DL.cg := `num`.val
HDL	→	`name` `num` `":"` DL	HDL.x := `name`.x | DL.size | `":"` | DL.x DL.cg := `num`.val
DL	→	DL `","` `name` `num`	DL0.x := DL1.x | `","` | `name`.x | `num`.val ∗ DL0.cg DL0.size := DL1.size + `num`.val ∗ DL0.cg
DL	→	`name` `num`	DL.x := `name`.x | `num`.val ∗ DL.cg DL.size := `num`.val ∗ DL.cg