1 Sprache

Wir betrachten eine einfache Sprache, auf die wir im Folgenden unsere Analysen anwenden. Die Grammatik der Sprache ist in Backus-Naur-Form (BNF) angegeben.

```
Stmt ::= Var = Expr ;
Stmt ::= if ( Expr ) Stmt else Stmt | if ( Expr ) Stmt
Stmt ::= while ( Expr ) Stmt
Stmt ::= return Expr ;
Stmt ::= Stmt Stmt
Expr ::= Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr |
         Expr == Expr | Expr != Expr | Expr < Expr | Expr > Expr |
         Const | Var
```

Die Sprachelemente umfassen nur Zuweisungen, Fallunterscheidungen mittels if, while-Schleifen und Hintereinanderausführung von Anweisungen. Das Endergebnis des Programms wird mit einer return-Anweisung zurückgegeben. Insbesondere existieren keine Funktionsaufrufe. Variablen (Var) sind beliebige Namen, Konstanten (Const) sind beliebige ganze Zahlen. Ist ein Vergleich wahr, so liefert er als Ergebnis den Wert 1, sonst 0. Eine Bedingung ist wahr, wenn der Wert des dazugehörigen Ausdrucks nicht 0 ist. Die Zugehörigkeit einer Anweisung zu einer umgebenden Anweisung wird mit Einrückung verdeutlicht.
Hinweis: Für die Durchführung der Analysen ist es hilfreich, Programme als Steuerflussgraphen darzustellen.

2 Konstantenfaltung

Konstantenfaltung bestimmt Ausdrücke, die immer zum selben Wert ausgewertet werden und ersetzt diese Ausdrücke durch Konstanten. Dies vermeidet wiederholtes Berechnen dieser Ausdrücke. Ein einfaches Beispiel für Konstantenfaltung ist die Ersetzung von $5+3$ durch 8. Wir wollen Konstanten nicht nur innerhalb einzelner Ausdrücke falten, sondern auch über mehrere Anweisungen hinweg. Hierfür müssen wir zunächst bestimmen, welche Variablen an welchen Programmstellen immer denselben Wert haben. Dazu merken wir uns, welchen Wert jede Variable an jeder Programmstelle hat. Anfangs nehmen wir die Werte an jeder Stelle als nicht analysiert an, was wir durch $\bot$ (bottom) darstellen.

Um die Werte an einer Anweisung zu berechnen, übernehmen wir zunächst die Werte der Vorgängeranweisung und werten den Effekt der aktuellen Anweisung aus. Wird eine Programmstelle von mehreren Vorgängeranweisungen erreicht (der Kopf einer Schleife, die Anweisung nach einem if), so müssen wir deren Informationen kombinieren. Ist eine Variable an einem Vorgänger noch nicht analysiert ($\bot$), so wird dieser ignoriert. Falls jedoch alle Vorgänger $\bot$ als Wert für diese Variable haben, so verwenden wir dies. Stimmen die Werte einer Variable an allen übrigen Vorgängern überein, so ist dies der Ausgangswert der Variable an dieser Stelle. Unterscheiden sich die Werte, so vermerken wir, dass die Variable an dieser Stelle keinen konstanten Wert hat mit dem Zeichen $\top$ (top).

Nur Zuweisungen ändern die Belegung von Variablen. Wir werten daher die rechte Seite einer Zuweisung aus und vermerken diesen Wert als neue Belegung für die Variable auf der linken Seite. Wir brauchen nur die Auswertung einzelner Operationen zu beschreiben. Der Wert eines aus mehreren Operationen zusammengesetzten Ausdrucks ergibt sich durch Einsetzen der Ergebnisse von Operationen in die sie verwendende Operation. Wir werten Operationen folgendermaßen aus:

- Ist ein Operand $\bot$, so ist das Ergebnis $\bot$.
- Ist ein Operand $\top$, so ist das Ergebnis $\top$.
- Ansonsten sind alle Operanden Konstanten oder Variablen mit konstantem Wert. Wir können den Wert dieser Operation somit direkt berechnen.

Wir wenden diese Regeln so lange an, bis sich an keiner Programmstelle die berechneten Werte mehr ändern. Zur Verdeutlichung folgt ein kleines Beispiel:

Programm	Initialisierung	1. Schritt	2. Schritt	3. Schritt
a = 19;	$\{a = \bot, b = \bot\}$	$\{a = 19, b = \bot\}$	$\{a = 19, b = \bot\}$	$\{a = 19, b = \bot\}$
b = a + 23;	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = 19, b = 42\}$	$\{a = 19, b = 42\}$
if (...)	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = 19, b = 42\}$
b = 0;	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$
return b;	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$	$\{a = \bot, b = \bot\}$

Programm	4. Schritt	5. Schritt	Transformation
a = 19;	$\{a = 19, b = \bot\}$	$\{a = 19, b = \bot\}$	a = 19;
b = a + 23;	$\{a = 19, b = 42\}$	$\{a = 19, b = 42\}$	b = 42;
if (...)	$\{a = 19, b = 42\}$	$\{a = 19, b = 42\}$	if (...)
b = 0;	$\{a = 19, b = 0\}$	$\{a = 19, b = 0\}$	b = 0;
return b;	$\{a = \bot, b = \bot\}$	$\{a = 19, b = \top\}$	return b;

1. Bestimmen Sie anhand des beschriebenen Verfahrens, welche Berechnungen in den folgenden Programmen durch Konstanten ersetzt werden können.
   Hinweis: Das Verfahren konvergiert schneller, wenn man – anders als im Beispiel oben – Ergebnisse der aktuellen Iteration mit einbezieht und so mehrere Schritte in einem Durchlauf vollzieht.

2. Verfeinern Sie das Verfahren, so dass noch mehr konstante Ausdrücke erkannt werden.

3. Darf das Verfahren zu einem beliebigen Zeitpunkt abgebrochen werden und können die bis dahin berechneten Ergebnisse verwendet werden? Begründen Sie Ihre Antwort.

1.
```
a = 1;
b = 5;
c = a + b;
a = 9;
d = a + b;
if (c < d)
  a = 2 * b - 1;
  c = d;
e = 3 * c;
b = 3 * d;
i = 1;
while (i < e)
  a = 3 * 3;
  i = i * 2;
return a;
```

2.
```
if (...)
  x = 23;
else
  x = 42;
return x * 0;
```

3.
```
x = 1;
i = 0;
while (i != 10)
  x = 2 - x;
  i = i + 1;
return x;
```

4.
```
if (...)
  a = 3;
  b = 4;
else
  b = 3;
  a = 4;
return a + b;
```

3 Lebendige Variablen

Eine Variable ist an einer Programmstelle lebendig, wenn deren Wert später noch benötigt wird. Ist eine Zuweisung tot, d.h. nicht lebendig, so können diese Zuweisung und alle damit verbundenen Berechnungen entfernt werden. So ist im Beispiel zur Konstantenfaltung nach der Transformation die Variable a tot.

Genau definieren wir Lebendigkeit folgendermaßen: Eine Variable ist an einem Programmpunkt lebendig, wenn von diesem aus ein Programmpunkt erreichbar ist, an dem die Variable verwendet wird und die Variable auf dem Weg dorthin nicht überschrieben wird. Ist die Verwendung eine Zuweisung, so muss die zugewiesene Variable dort lebendig sein;[1] andernfalls ist die Variable an der betrachteten Stelle tot.

Zur systematischen Bestimmung der lebendigen Variablen nehmen wir zunächst an, dass alle Variablen an allen Programmstellen tot sind. Die Berechnung der lebendigen Variablen an einer Programmstelle beginnt zunächst mit der Vereinigung der lebendigen Variablen aller nachfolgenden Programmstellen (z.B. Konsequenz und Alternative einer if-Anweisung). Ist die betrachtete Anweisung eine Zuweisung, so prüfen wir zunächst, ob die zugewiesene Variable lebendig ist. Falls dies nicht der Fall ist, so ist die Betrachtung dieser Programmstelle beendet. Andernfalls entfernen wir die Variable aus der Menge der lebendigen Variablen, da sie hier überschrieben wird und somit Zuweisungen weiter oben auf diesem Pfad für die Betrachtung der Lebendigkeit blockiert. Weiter – oder falls es sich nicht um eine Zuweisung handelt – fügen wir alle im Ausdruck dieser Anweisung verwendeten Variablen zu den lebendigen Variablen hinzu. Abermals wiederholen wir dieses Verfahren an allen Programmpunkten, bis sich keine Menge mehr ändert.

1. Berechnen Sie für jede Programmstelle der nachfolgenden Progamme die lebendigen Variablen.
2. Während die Konstantenfaltung das Programm entlang des Programmflusses analysiert, wird bei der Bestimmung der lebendigen Variablen das Programm entgegen dessen verarbeitet. Beschreiben Sie einen weiteren strukturellen Unterschied zwischen den Analysen.

1.
```
x = 2;
y = 4;
x = 1;
if (y > x)
  z = y;
else
  z = y * y;
x = z;
return z;
```

2.
```
x = 0;
y = 0;
while (x != 10)
  y = y + x;
  x = x + 1;
return 0;
```

[1] Dies wird auch als „echte Lebendigkeit“ bezeichnet. Einfache Lebendigkeit lässt diesen Punkt aus.

3.
```
a = 1;
c = a;
i = 10;
k = 42;
while (i != 0)
  b = 1 - i;
  if (a != 0)
    a = b + c;
  n = 0;
  while (n < i * i)
    n = n + i;
    k = k - 1;
  i = i - 1;
return a;
```