Prüfung Informatik D-MATH/D-PHYS 30. 9. 2005

Dr. Bernd Gärtner

Lösung.

Aufgabe 1. (15 Punkte)

(a) 5 Punkte

```
  3+2/5*2-1.0/2
→ 3+0*2-1.0/2 (int-Division)
→ 3+0-1.0/2 (int-Multiplikation)
→ 3-1.0/2 (int-Addition)
→ 3-1.0/2.0 (Konversion nach double)
→ 3-0.5 (double-Division)
→ 2.5 (double-Subtraktion)
```

(b) 5 Punkte

```
  3/2.0 <= 1 || 3%2 > 188
→ 3.0/2.0 <= 1 || 3%2 > 188 (Konversion nach double)
→ 1.5 <= 1 || 3%2 > 188 (double-Division)
→ 1.5 <= 1.0 || 3%2 > 188 (Konversion nach double)
→ false || 3%2 > 188 (int Vergleich)
→ false || 1 > 188 (int-modulo)
→ false || false (int-Vergleich)
→ false (bool-Evaluation)
```

(c) 5 Punkte

```
  4<3 || 3<4 && true || 18.6/3.1f>6.2
→ false || 3<4 && true || 18.6/3.1f>6.2 (int Vergleich)
→ false || true && true || 18.6/3.1f>6.2 (int Vergleich)
→ false || true || 18.6/3.1f>6.2 (bool-Evaluation)
→ true (Short-circuit-Evaluation)
```

Aufgabe 2. (25 Punkte) Weil die Funktion 2^y monoton steigend ist, ist $\lfloor \log x \rfloor$ die *grösste* ganze Zahl y mit $2^y \leq x$. Diese finden wir zum Beispiel mit folgender Funktion.

```
int log (unsigned int x) {
  // PRE: x > 0
```

```
  int y = 0;
  int zweihochy = 1;
  while (2*zweihochy <= x) {
    // y noch zu klein
    ++y;
    zweihochy *= 2;
  }
  return y;
}
```

Die Precondition ist $x > 0$, weil der Logarithmus für $x = 0$ undefiniert ist. Alternativ könnte man $\lfloor \log 0 \rfloor := 0$ definieren und keine Precondition fordern. Dies entspricht dann allerdings nicht mehr der mathematischen Funktion $\lfloor \log x \rfloor$, und das abweichende Verhalten muss in der Postcondition dokumentiert werden.
Für die Funktion selbst gibt es 20 Punkte, für die Precondition 5.

Aufgabe 3. (20 Punkte)

(a) (10 Punkte) Beide Funktionen geben die Anzahl der Primteiler von n zurück. Für $n = 24$ z.B. ergibt sich 4, wegen $24 = 2 \cdot 2 \cdot 2 \cdot 3$. Für $n \in \{0, 1\}$ wird 1 zurückgegeben.

Die erste Funktion teilt n zunächst durch den kleinsten echten Teiler i (dieser muss dann ein Primteiler sein) und ruft sich dann rekursiv für den Rest n/i auf. Die zweite Funktion findet zunächst den grössten echten Teiler i von n. Es folgt, dass n/i dann der kleinste echte Teiler von n ist, und damit ein Primteiler. Der Rest $n/(n/i) = i$ wird dann wie vorher rekursiv bearbeitet. Beide Funktionen rufen sich also für festes n rekursiv mit dem *gleichen* Argument (n geteilt durch kleinsten Primteiler) auf.

(b) (10 Punkte) Falls n eine Primzahl ist, sind beide Varianten gleich "effizient". Für alle anderen Zahlen aber ist die erste klar besser. Wir wissen, dass der grösste echte Teiler von n höchstens die Grösse $\sqrt{n}$ hat. In Variante zwei werden deshalb *mindestens* ungefähr $n - \sqrt{n}$ Zahlen i vergeblich getestet, bevor überhaupt ein Teiler-Kandidat auftaucht. In Variante eins ist die Suchzeit dagegen proportional zur Grösse des kleinsten Primteilers, also *höchstens* $\sqrt{n}$. Für grosses n ist $\sqrt{n}$ wesentlich kleiner als $n - \sqrt{n}$.

Aufgabe 4. (20 Punkte) Wir wählen das Startsymbol S und die Regeln

$$S \to S0 \mid S1S1 \mid \varepsilon.$$

Offenbar stellen wir damit sicher, dass die Anzahl der Einsen in jedem erzeugten Wort gerade ist. Um zu sehen, dass jedes Wort mit gerade vielen Einsen erzeugt wird, kann man formal mit Induktion argumentieren, dies war allerdings nicht gefordert. Wir machen es hier trotzdem, wobei die Induktion über die Länge des Wortes läuft.

Das eindeutige Wort ε der Länge 0 in $L(S)$ wird erzeugt, wegen Regel 3 (dritte "oder"-Klausel). Nehmen wir nun an, wir können bereits alle Wörter mit gerade vielen Einsen und Länge bis zu $n-1$ erzeugen. Sei nun ein Wort w mit gerade vielen Einsen und Länge $n > 0$ gegeben. Wir unterscheiden zwei Fälle.

(i) w ist von der Form $w'0$. Dann beginnen wir mit der Ableitung $S \to S0$ (Regel 1) und ersetzen S dann gemäss der Ableitung von w', von der wir mit Induktion annehmen, dass es sie gibt. Die gesamte Ableitung erzeugt dann das Wort $w'0 = w$.

(ii) w ist von der Form $w'1$. Weil w eine gerade Anzahl von Einsen besitzt, muss es in w' mindestens eine weitere 1 geben, also können wir w in der Form

$$w = u1v1$$

schreiben, wobei v keine Einsen enthält, u also folglich wieder eine gerade Anzahl von Einsen. Es gilt $|u|, |v| \leq n-1$, wir können also Induktion anwenden. Wir beginnen mit der Ableitung $S \to S1S1$ (Regel 2), dann ersetzen wir das erste S gemäss der Ableitung von u und das zweite S gemäss der Ableitung von v. Wir erhalten schliesslich w.

Die gesuchten Ableitungen für die zwei konkreten Wörter sind

$$\underline{S} \overset{2}{\to} \underline{S}1S1 \overset{1}{\to} S01\underline{S}1 \overset{1}{\to} \underline{S}01S01 \overset{3}{\to} 01\underline{S}01 \overset{3}{\to} 0101$$

und

$$\underline{S} \overset{2}{\to} \underline{S}1S1 \overset{2}{\to} \underline{S}1S11S1 \overset{3}{\to} 1\underline{S}11S1 \overset{3}{\to} 111\underline{S}1 \overset{3}{\to} 1111,$$

wobei das jeweils abgeleitete Symbol unterstrichen und die verwendete Regel über dem Ableitungspfeil angegeben ist.

Für die Grammatik gibt es 12 Punkte, für jede der Ableitungen 4 Punkte. Falls ein Wort richtig aus der falschen Grammtik abgeleitet wird, werden die 4 Punkte gutgeschrieben.

Aufgabe 5. (16 Punkte)

n	f1(n)	n	f2(n)	n	f3(n)	n	f4(n)	n
5	6	5	6	6	6	6	6	7

Aus der Bewertung werden Folgefehler herausgerechnet: für jeden Wert, der korrekt aus den vorhergehenden Werten folgt, gibt es 2 Punkte. Im Fall von `fi(n)` ist der vorhergehende Wert von `n` relevant, im Fall eines `n` sind die beiden vorhergehenden Werte wichtig.

Aufgabe 6. (25 Punkte) Wir können ein `modulo7`-Objekt zum Beispiel durch einen `unsigned int` Wert zwischen 0 und 6 repräsentieren.

```
private:
  // Repraesentation
  unsigned int m; // Invariante: m ist aus {0,...,6}
};
```

Die Konstruktoren sind dann einfach; wir verwenden den Modulo-Operator des Typs `unsigned int`.

```
Modulo7::Modulo7 ()
  : m(0)
{}

Modulo7::Modulo7 (unsigned int n)
  : m (n % 7)
{}
```

Die Addition ist ebenfalls einfach: wir addieren die beiden Repräsentationen über dem Typ `unsigned int` und nehmen das Ergebnis modulo 7 (man prüfe anhand der Verknüpfungstabelle, das dies das Richtige liefert). Aus diesem Ergebnis konstruieren wir dann den Rückgabewert vom Typ `Modulo7` mittels unserer benutzerdefinierten Konversion.

```
Modulo7 operator+ (Modulo7 x, Modulo7 y) {
  return Modulo7(x.m+y.m);
}
```

Für die Subtraktion würden wir gerne die beiden Repräsentationen vom Typ `unsigned int` subtrahieren und dann das Ergebnis modulo 7 nehmen. Dabei müssen wir uns aber im Klaren sein, dass das Ergebnis der Subtraktion negativ sein kann, was bedeutet, dass der Wert undefiniert ist. Um das zu vermeiden, addieren wir vor der Subtraktion einfach den Wert 7, was modulo 7 nichts ändert. Man prüfe anhand der Verknüpfungstabelle und der Definition der Subtraktion modulo 7, das dies wiederum das Richtige liefert.

```
Modulo7 operator- (Modulo7 x, Modulo7 y) {
  return Modulo7(7+x.m-y.m);
}
```

Hier ist es wichtig, dass `7+x.m-y.m` implizit als `(7+x.m)-y.m` geklammert ist, andernfalls würde das Problem mit dem negativen Zwischenergebnis erhalten bleiben.

Für Teil (a) gibt es 3 Punkte, für den Default-Konstruktor 3, für die Konversion 5 und für die zwei arithemtischen Operatoren jeweils 7 Punkte.