Institut für Informatik
Heinrich-Heine-Universität Düsseldorf
Prof. Dr. Alice McHardy
Robin Küppers
Sebastian Konietzny
Aaron Weimann

WS 2013/2014
05.11.2013
Übungsblatt 2

Einführung in die Informatik I
Übungen zur Vorlesung
Musterlösung

2.1 Determinismus / Determiniertheit (5 Punkte)

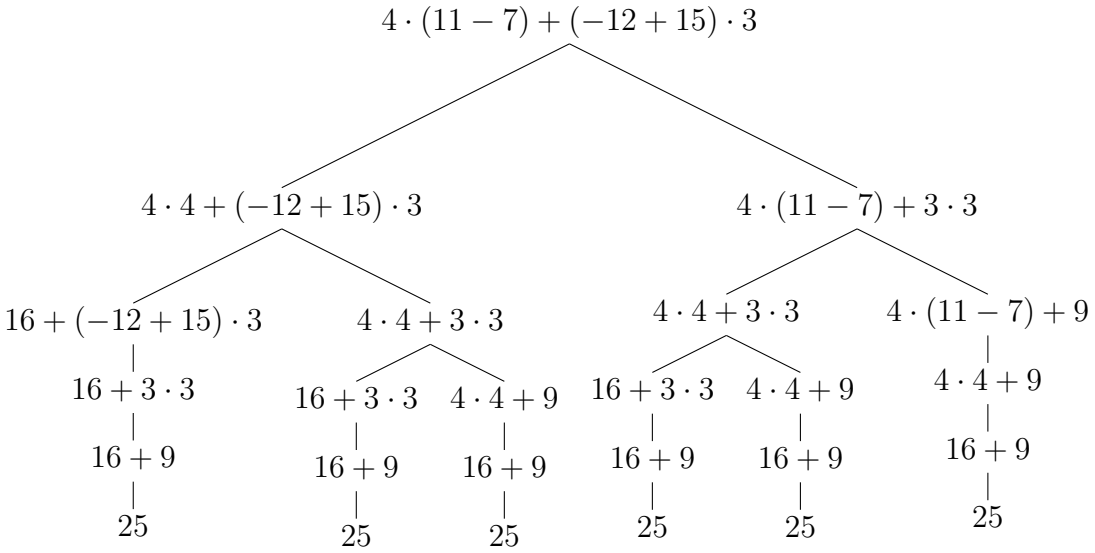
Gegeben sei folgende Formel:

$$4 \cdot (11 - 7) + (-12 + 15) \cdot 3$$

1. Nennen Sie die typischen Eigenschaften eines Algorithmus. (1 Punkt)
2. Werten Sie die gegebene Formel schrittweise unter Beachtung der üblichen Vorrangregeln (Multiplikation/Division vor Addition/Subtraktion,...) aus. Ignorieren Sie für die Auswertung das Distributivgesetz. Führen Sie pro Schritt immer nur eine Rechenoperation aus. Untersuchen Sie die Auswertung bezüglich Determinismus und Determiniertheit und geben Sie sämtliche Alternativen bei der Auswertung an. (3 Punkte)
3. Wie kann eine deterministische Auswertung der Formel erreicht werden? (1 Punkt)

Musterlösung

Ein Algorithmus ist eine genau festgelegte Berechnungsvorschrift, um eine spezifizierte Eingabe in eine spezifizierte Ausgabe umzuwandeln. In der Vorlesung wurden Eigenschaften von Algorithmen genannt, die aber nicht alle zwingend bei jedem Algorithmus realisiert sein müssen. Im Folgenden werden Beispiele genannt:

1. Terminierend, Korrekt, Determiniert, Deterministisch (es gibt aber z.B. auch nicht-deterministische determinierte Algorithmen)
2. 

3. Eine deterministische Auswertung der Formel kann dadurch erreicht werden, dass eine zusätzliche Regel eingeführt wird, die z.B. festlegt, dass immer zuerst die linke oder rechte aller Möglichkeiten bearbeitet wird.

2.2 Reguläre Ausdrücke (6 Punkte)

Zwei reguläre Ausdrücke x, y sind gleich, wenn die von ihnen beschriebenen Sprachen $L(x)$ und $L(y)$ identisch sind. Beispielsweise erzeugt der Ausdruck $z = 1(0+1)^*$ die Sprache der binären Zahlen ohne führende Nullen: $L(z) = \{1, 10, 11, 100, 101, \ldots\}$. Seien a, b, c beliebige reguläre Ausdrücke. Beweisen Sie die folgenden Gleichheiten[1] bzw. zeigen Sie an einem Gegenbeispiel ihre Ungültigkeit:

1. $ab + ba = ba + ab$
2. $(aa + ab)^* = aa^* + ab^*$
3. $(aa + ab) + ac = aa + (ab + ac)$
4. $(ab + a)^* = a(b)^*$
5. $ab + bc + ba = b(a + c) + ab$
6. $(abc^*ab(c + d))^*a = (a(b + c)^*abc + aba)^*c$

[1] Die Gleichheit zweier Sprachen kann auch durch Aufzählen aller möglichen Sätze gezeigt werden (sofern zeitlich machbar und sinnvoll).

Musterlösung

1. $ab + ba = ba + ab$

   $L(x) = \{ab, ba\}$
   $L(y) = \{ba, ab\}$
   $\Rightarrow L(x) = L(y)$

2. $(aa + ab)^* = aa^* + ab^*$

   $L(x) = \{\epsilon, aa, ab, aaaa, aaab, abaa, abab, \ldots\}$
   $L(y) = \{a, aa, aaa, ab, abb, abbb, \ldots\}$also ein a gefolgt von beliebig vielen a oder b
   z.B.: $aaab \in L(x) \wedge aaab \notin L(y)$
   $\Rightarrow L(x) \neq L(y)$

3. $(aa + ab) + ac = aa + (ab + ac)$

   $L(x) = \{aa, ab, ac\}$
   $L(y) = \{aa, ab, ac\}$
   $\Rightarrow L(x) = L(y)$

4. $(ab + a)^* = a(b)^*$

   $L(x) = \{\epsilon, ab, a, abab, aba, aab, aa, \ldots\}$
   $L(y) = \{a, ab, abb, abbb, \ldots\}$
   z.B.: $abb \notin L(x) \wedge abb \in L(y)$
   $\Rightarrow L(x) \neq L(y)$

5. $ab + bc + ba = b(a + c) + ab$

   $L(x) = \{ab, bc, ba\}$
   $L(y) = \{ba, bc, ab\}$
   $\Rightarrow L(x) = L(y)$

6. $(abc^*ab(c + d))^*a = (a(b + c)^*abc + aba)^*c$

   $\forall x \in L(x)$ gilt: x endet immer auf a
   $\forall y \in L(y)$ gilt: y endet immer auf c
   $\Rightarrow L(x) \neq L(y)$

2.3 BNF (6 Punkte)

Gegeben ist die folgende Grammatik G_1 in BNF:

```
<S>  ::=  a <G> a | b <G> b | <G>
<G>  ::=  <F> | c <F> c | c <S> c
<F>  ::=  a | b | c
```

1. Sind folgende Wörter in der von G_1 erzeugten Sprache enthalten, d.h. lassen sie sich ausgehend von $< S >$ durch Anwenden der Produktionsregeln ableiten? `abc aaa babcb bcacb acbbca acabaca bb ccccccc` (4 Punkte)

2. Können Wörter beliebiger Länge erzeugt werden? Begründung? (2 Punkte)

Musterlösung

1. `abc`: nicht enthalten, da nicht am mittleren Buchstaben gespiegelt

   `aaa`: `<S> -> a <G> a -> a <F> a -> aaa`

   `babcb`: nicht enthalten, da nicht am mittleren Buchstaben gespiegelt

   `bcacb`: `<S> -> b <G> b -> bc <F> cb -> bcacb`

   `acbbca`: nicht enthalten, da nicht am mittleren Buchstaben gespiegelt

   `acabaca`: `<S> -> a <G> a -> ac <S> ca -> aca <G> aca`
   `-> aca <F> aca -> acabaca`

   `bb`: nicht enthalten, da nicht am mittleren Buchstaben gespiegelt

   `ccccccc`: `<S> -> <G> -> c <S> c -> c <G> c -> cc <S> cc`
   `-> cc <G> cc -> ccc <F> ccc -> ccccccc`

2. Alle Wörter der Sprache G_1 sind um einen Buchstaben in der Mitte gespiegelt und genügen somit der Form xyx. Es können somit nur Wörter mit ungerader Länge erzeugt werden:

   Formale Lösung[2]:

   $|xyx| = |x| + |y| + |x| = 2 \cdot |x| + |y|$
   Da $|y| = 1$, gilt: $|xyx| = 2 \cdot |x| + 1$
   $\Rightarrow 2 \cdot |x| \bmod 2 = 0$
   $\Rightarrow 2 \cdot |x| + 1 \bmod 2 = 1$
   $\Rightarrow |xyx|$ ist immer ungerade.

[2] Für die Lösung dieser Aufgabe ist KEINE formale Lösung notwendig.

2.4 Struktogramm, Das Sieb des Eratosthenes (8 Punkte)

Eine Primzahl ist eine natürliche Zahl, die genau zwei natürliche Zahlen als Teiler hat. Für manche Zahlen ist es schwierig, festzustellen, ob sie prim sind oder nicht. Der folgende Algorithmus[3] bekommt als Eingabe eine Zahl $N \geq 2$ und bestimmt alle Primzahlen zwischen 2 und N. Dafür markiert der Algorithmus alle Zahlen, die nicht prim sein können, da sie sich als Produkt von Primzahlen darstellen lassen.

```
Objekte: N, i, j               // N, i und j sind positive, ganze Zahlen
         markiert              // markiert ist eine Liste von booleschen Werten
Eingabe: N
Ausgabe: markiert

markiert[1] := true; // 1 ist keine Primzahl
i := 2;
WHILE i <= N DO
  markiert[i] := false;
  i: = i + 1;
OD
// Untersuchung aller Zahlen zwischen 2 und Wurzel(N)
i := 2;
WHILE i*i <= N DO
  IF not markiert[i] THEN
    // i ist prim, streiche seine Vielfache mit i*i beginnend
    j := i * i;
    WHILE j <= N DO
      markiert[j] = true;
      j := j + i;
    OD
  FI
  i := i+1;
OD
```

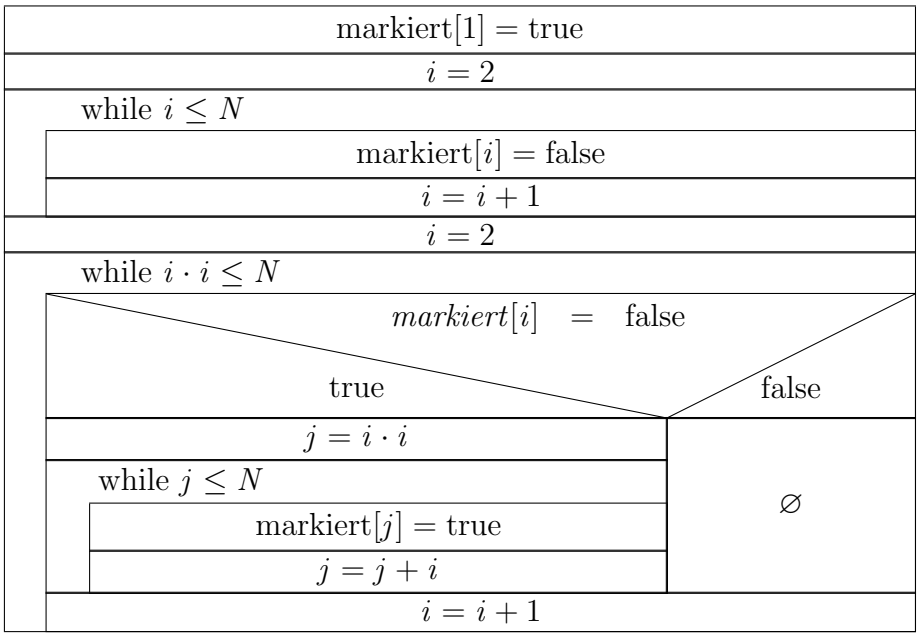
Nach Durchlaufen des Algorithmus sind alle nicht markierten Zahlen prim.

1. Stellen Sie den Algorithmus als Struktogramm (nach Nassi-Schneiderman dar). (6 Punkte)
2. Um zu bestimmen, ob eine gegebene Zahl N prim ist, kann man den oben angegebenen Algorithmus mit der Eingabe N durchführen und schließlich überprüfen, ob die Zahl N selbst markiert worden ist. Wie könnte man die Laufzeit dieses Primzahltests verkürzen, ohne den grundlegenden Algorithmus zu ändern? (2 Punkte)

[3]Beachten Sie, dass *markiert* eine Liste ist, die N Werte enthält. Auf einzelne Elemente kann mittels eines Index i zugegriffen werden, d.h. markiert[i] entspricht dem i-ten Listenelement. Die Zeichenkombination // leitet einen Kommentartext ein, der bei der Ausführung nicht berücksichtigt wird.

Musterlösung

1.



2.

- Sobald N markiert wird, kann der Algorithmus abbrechen, denn N ist dann sicher keine Primzahl.
- Wenn N eine gerade Zahl ist, kann es keine Primzahl sein. Analog gibt es weitere einfache Teilbarkeitstests, z.B. für die Teilbarkeit durch 3 (Quersumme ohne Rest durch 3 teilbar) oder 5 (letzte Ziffer in Dezimaldarstellung 0 oder 5).