Alp2 - Übungsblatt 5 (Aufgabe 4)

Bearbeitet von: Jasmine Cavael & Alexander Chmielus

Tutor: Fabian Halama

Tutorium 10 (Do. 16-18)

Mit diesem Schreibprogramm ist es leider etwas umständlich, mathematische Zeichen einzufügen in den Formeln. Wir werden daher Python-Syntax verwenden:
&& als logisches UND
|| als logisches ODER
! als logische Negation
--> Implikation
== ist gleich (wir benutzen = als Zuweisung, daher der Unterschied.)
!= ist nicht gleich

4 a)

{hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)}

4 b)

Das Programm sieht so aus:

```
{P} = {Zahl >= 0}
hilf = Zahl
bits = 1
{INV} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)}
while hilf > 1:
{INV && B} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1) && hilf > 1}
        hilf = hilf // 2
        bits = bits + 1
{INV && !B} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1) && hilf <= 1}
{Q} = {bits == BinaryDigits(Zahl)}
```

Um die Invarianten-Eigenschaft zu zeigen, müssen wir laut **While-Regel** zeigen, dass {INV && B} S {INV} gültig ist. Das ganze sieht dann also so aus:

```
{INV && B} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1) && hilf > 1}
        hilf = hilf // 2
        bits = bits + 1
{INV} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)}
```

Zuweisungsaxiom (bits = bits + 1):

{INV1} = {hilf >= 0 && (BinaryDigits(hilf) + bits + 1 == BinaryDigits(Zahl) + 1)}

Zuweisungsaxiom (hilf = hilf // 2):

{INV2 = {(hilf // 2) >= 0 && (BinaryDigits(hilf // 2) + bits + 1 == BinaryDigits(Zahl) + 1)}

Jetzt müssen wir für die **Konsequenz-Regel (der stärkeren Vorbedingung)** zeigen, dass
{INV && B} --> {INV2}.
Zuerst zeigen wir, dass (**hilf // 2 >= 0.)**
(hilf >= 0) --> ((hilf // 2) >= 0) ist offensichtlich, da wenn hilf = 0 ist auch hilf // 2 = 0 ist und wenn hilf > 0 ist, ist es auch (hilf // 2).

Als nächstes (**BinaryDigits(hilf // 2) + bits + 1 == BinaryDigits(Zahl) + 1).**
Wir issen aus {INV}, dass (BinaryDigits(hilf) + bits == BinaryDigits(Zahl + 1). Eigentlich müssten wir jetzt eine Fallunterscheidung für BinaryDigits(hilf) machen. Da wir aber außerdem aus {INV} wissen, dass hilf > 1, entfällt der erste Fall.
Dadurch wissen wir, dass BinaryDigits(hilf) = BinaryDigits(hilf // 2) + 1. Setzen wir das nun in {INV} ein ergibt sich: (BinaryDigits(hilf // 2) + 1 + bits == BinaryDigits(Zahl + 1), was genau das ist, was wir zeigen wollten.
(BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1) && hilf > 1) --> (BinaryDigits(hilf // 2) + bits + 1 == BinaryDigits(Zahl) + 1)

Die Schleife sieht nun so aus:
{INV && B} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1) && hilf > 1}
    {INV4}
    hilf = hilf // 2
    {INV3}
    bits = bits + 1
{INV} = {hilf >= 0 && (BinaryDigits(hilf) + bits == BinaryDigits(Zahl) + 1)}

Nach den Zuweisungsaxiomen, sowie der Konsequenz- und der Sequenzregel sind sämtliche Programmformeln und damit auch das gesamte Programm nach der While-Regel gültig.