Integer Division in Logarithmic Time

Course on Formal Methods in Computer Science

Gernot Salzer

May 9, 2011

The example is inspired by those in [2]. The programming language and the specific form of the Hoare calculus used here are described in [3]. The following presentation is rather lengthy, giving all intermediate steps and additional explanations. At the exam it is sufficient to describe the main steps.

1 Statement of the problem

The following program computes the integer quotient n/m for positive integers m and n, with the final result in a. It takes time logarithmic in the quotient a, if we assign unit cost to the integer operations. Show that the program is totally correct.

```
c := 1;
while c * m ≤ n do
  c := 2 * c
od;
a := 0;
while c ≠ 1 do
  c := c/2;
  if (a + c) * m > n then
    skip
  else
    a := a + c
  fi
od
```

(Note that in a real implementation, division by 2 can be implemented as a shift operation, i.e., the program does not need division to compute division.)

2 Solution

We divide the correctness proof into two parts, which correspond to the two loops, i.e., we prove the correctness of the two assertions

Initialisation loop

$\{\,1\colon \mathit{Pre}\,\}$
$c := 1;$
while $c * m \leq n$ do
  $c := 2 * c$
od;
$\{\,2\colon \mathit{Inv}_1 \wedge c * m > n\,\}$

Division by adding powers of two

$\{\,2\colon \mathit{Inv}_1 \wedge c * m > n\,\}$
$a := 0;$
while $c \neq 1$ do
  $c := c/2;$
  if $(a + c) * m > n$ then
    skip
  else
    $a := a + c$
  fi
od
$\{\,3\colon \mathit{Post}\,\}$

Pre and *Post* denote the pre- and postcondition of the original program, Inv_1 and Inv_2 are the invariants of the two loops. We will determine these four formulas later on.

Note that the postcondition of the first correctness assertion is equal to the precondition of the second one. By the rule for sequential composition, the correctness of these two assertions implies the correctness of the original problem.

2.1 Initialisation loop

The first loop initialises the variable c to the smallest power of two that is greater than the quotient n/m.

Partial correctness

We start by annotating the program with conditions (assertions) according to the rules of the Hoare calculus for partial correctness; termination will be discussed separately. The formulas are numbered in the order in which they are added.

$\{\,1\colon \mathit{Pre}\,\}$
$\{\,7\colon \mathit{Inv}_1[c/1]\,\}$
$c := 1;$
$\{\,4\colon \mathit{Inv}_1\,\}$
while $c * m \leq n$ do
  $\{\,5\colon \mathit{Inv}_1 \wedge c * m \leq n\,\}$
  $\{\,8\colon \mathit{Inv}_1[c/2{*}c]\,\}$
  $c := 2 * c$
  $\{\,6\colon \mathit{Inv}_1\,\}$
od;
$\{\,2\colon \mathit{Inv}_1 \wedge c * m > n\,\}$

To complete the proof we have to find appropriate formulas *Pre* and Inv_1 such that the implications $1 \Rightarrow 7$ and $5 \Rightarrow 8$ are valid.

The precondition has to constrain the values of the input variables to those, for which the algorithm works. The condition $c * m \leq n$ properly terminates the loop only for

non-negative values of m and n. Moreover, since the program computes integer division, the divisor m may not be null. So our guess for an appropriate precondition is the formula $Pre \equiv n \geq 0 \wedge m > 0$.

The loop initialises the variable c to a power of 2 that is greater than the quotient. The property of being greater than the quotient ($c > n/m$ or equivalently $c * m > n$) holds after the loop due to the negated while-condition, hence the invariant has to specify only the fact that c is a power of 2. We choose the formula $power2(c) \wedge Pre$ as invariant Inv_1, where $power2(c)$ is an abbreviation for the formula $\exists x(x \geq 0 \wedge c = 2^x)$.[1] Pre is part of the invariant to propagate this information also to the second loop.

It remains to prove the validity of the implications derived above.

1 ⇒ 7:

$$\begin{aligned} Pre &\Rightarrow Inv_1[c/1] \\ Pre &\Rightarrow (power2(c) \wedge Pre)[c/1] \\ Pre &\Rightarrow \boldsymbol{power2(1)} \wedge Pre \end{aligned}$$

This implication is valid since $1 = 2^0$ is indeed a power of 2.

5 ⇒ 8:

$$\begin{aligned} Inv_1 \wedge c * m \leq n &\Rightarrow Inv_1[c/2 * c] \\ power2(c) \wedge Pre \wedge c * m \leq n &\Rightarrow (power2(c) \wedge Pre)[c/2 * c] \\ \boldsymbol{power2(c)} \wedge Pre \wedge c * m \leq n &\Rightarrow \boldsymbol{power2(2 * c)} \wedge Pre \end{aligned}$$

This implication is valid since $2 * c$ is indeed a power of 2 provided that c itself is a power of 2.

Termination

As variant (bound function) t we need an integer expression that decreases in each iteration and that is bounded by zero. Since termination is tied to the loop condition, the latter is a good starting point. It can be rewritten as $0 \leq n - cm$, hence $n - cm$ might be a suitable variant: The expression decreases in each iteration and its value is obviously bounded by zero as long as the loop continues. To prove this in a formal way we have to show that the assertion

$$\{\, Inv_1 \wedge c * m \leq n \wedge t = t_0 \,\}\, c := 2 * c \,\{\, c * m \leq n \Rightarrow 0 \leq t < t_0 \,\}$$

is correct (see rule wht‴ on the slides).

$\{\, 9\colon Inv_1 \wedge c * m \leq n \wedge 0 \leq t = t_0 \,\}$
$\{\, 11\colon (c * m \leq n \Rightarrow 0 \leq t < t_0)[c/2 * c] \,\}$
$c := 2 * c$
$\{\, 10\colon c * m \leq n \Rightarrow 0 \leq t < t_0 \,\}$

[1] You may object that 2^x is no proper expression of our toy language, since it does not contain an exponentiation operator. Of course we can define the function recursively, but our type of first-order formulas does not admit recursive definitions. However, it can be shown that every recursive function can be defined as a first-order formula [1, Chapter 16].

9 ⇒ 11:

$$
\begin{aligned}
&Inv_1 \wedge cm \le n \wedge \qquad t = t_0 \Rightarrow (cm \le n \Rightarrow 0 \le t < t_0)[c/2c] \\
&Inv_1 \wedge cm \le n \wedge n - cm = t_0 \Rightarrow (cm \le n \Rightarrow 0 \le n - cm < t_0)[c/2c] \\
&Inv_1 \wedge cm \le n \wedge n - cm = t_0 \Rightarrow (2cm \le n \Rightarrow 0 \le n - 2cm < t_0) \\
&Inv_1 \wedge cm \le n \wedge n - cm = t_0 \wedge 2cm \le n \Rightarrow 0 \le n - 2cm < n - cm \\
power2(c) \wedge n \ge 0 \wedge m > 0 \wedge cm \le n \wedge n - cm = t_0 \wedge 2cm \le n \Rightarrow n \ge 2cm > cm
\end{aligned}
$$

The conclusion consists of two inequalities. The first one, $n \ge 2cm$, holds because it also occurs in the premise. The second one, $2cm > cm$, is true since m is greater than zero and c being a power of two implies $2c > c$.

2.2 Division by adding powers of two

We now turn to the second loop, which computes the result of the division.

Partial correctness

We start again by annotating the program with further assertions following the rules of the Hoare calculus. We use B as an abbreviation for the if-condition $(a + c) * m > n$.

$\{\, 2\colon Inv_1 \wedge c * m > n \,\}$
$\{\, 16\colon Inv_2[a/0] \,\}$
$a := 0;$
$\{\, 12\colon Inv_2 \,\}$
while $c \neq 1$ do
  $\{\, 13\colon Inv_2 \wedge c \neq 1 \,\}$
  $\{\, 22\colon \big((B \Rightarrow Inv_2) \wedge (\neg B \Rightarrow Inv_2[a/a + c])\big)[c/(c/2)] \,\}$
  $c := c/2;$
  $\{\, 21\colon \ (B \Rightarrow Inv_2) \wedge (\neg B \Rightarrow Inv_2[a/a + c]) \,\}$
  if $(a + c) * m > n$ then
    $\{\, 19\colon Inv_2 \,\}$
    skip
    $\{\, 17\colon Inv_2 \,\}$
  else
    $\{\, 20\colon Inv_2[a/a + c] \,\}$
    $a := a + c$
    $\{\, 18\colon Inv_2 \,\}$
  fi
  $\{\, 14\colon Inv_2 \,\}$
od
$\{\, 15\colon Inv_2 \wedge c = 1 \,\}$
$\{\, 3\colon Post \,\}$

As postcondition we could use the formula $a = n/m$, which uses the integer division operator of our toy language. Because of truncation, integer division is not the inverse of integer multiplication, and simplifying formulas using this operator may be tricky. Therefore we use the formula $\mathit{Post} \equiv a*m \le n < (a+1)*m$ instead. (Convince yourself that for positive integers, this condition is indeed equivalent to $a = n/m$!)

For the invariant we choose $\mathit{Inv}_2 \equiv a*m \le n < (a+c)*m \wedge \mathit{power2}(c)$. This invariant is obtained by weakening the postcondition: the constant 1 is replaced by the variable c, which is required to be a power of 2.[2]

To complete the proof of partial correctness we have to prove the validity of three implications.

2 ⇒ 16:

$$\mathit{Inv}_1 \wedge c*m > n \Rightarrow \mathit{Inv}_2[a/0]$$
$$\mathit{power2}(c) \wedge n \ge 0 \wedge m > 0 \wedge c*m > n \Rightarrow (a*m \le n < (a+c)*m \wedge \mathit{power2}(c))[a/0]$$
$$\mathit{power2}(c) \wedge n \ge 0 \wedge m > 0 \wedge c*m > n \Rightarrow 0*m \le n < (0+c)*m \wedge \mathit{power2}(c)$$
$$\mathit{power2}(c) \wedge n \ge 0 \wedge m > 0 \wedge c*m > n \Rightarrow 0 \le n < c*m \wedge \mathit{power2}(c)$$
$$\boldsymbol{\mathit{power2}(c) \wedge 0 \le n < c*m} \wedge m > 0 \Rightarrow \boldsymbol{\mathit{power2}(c) \wedge 0 \le n < c*m}$$

13 ⇒ 22:

$$\mathit{Inv}_2 \wedge c \ne 1 \Rightarrow \big((B \Rightarrow \mathit{Inv}_2) \wedge (\neg B \Rightarrow \mathit{Inv}_2[a/a+c])\big)[c/(c/2)]$$
$$\mathit{Inv}_2 \wedge c \ne 1 \Rightarrow (B[c/(c/2)] \Rightarrow \mathit{Inv}_2[c/(c/2)]) \wedge (\neg B[c/(c/2)] \Rightarrow \mathit{Inv}_2[a/a+c][c/(c/2)]$$

The formula is of the form $\alpha \Rightarrow ((\beta \Rightarrow \gamma) \wedge (\delta \Rightarrow \varepsilon))$. It is valid if and only if the two implications $(\alpha \wedge \beta) \Rightarrow \gamma$ and $(\alpha \wedge \delta) \Rightarrow \varepsilon$ are valid.[3] Hence we may split the formula and prove the two implications separately; they correspond to the two branches of the if-statement.

13 ⇒ 22, then-branch:

$$\mathit{Inv}_2 \wedge c \ne 1 \wedge (a + c/2)*m > n \Rightarrow \mathit{Inv}_2[c/(c/2)]$$
$$\mathit{Inv}_2 \wedge c \ne 1 \wedge (a + c/2)*m > n \Rightarrow (a*m \le n < (a+c)*m \wedge \mathit{power2}(c))[c/(c/2)]$$
$$\mathit{Inv}_2 \wedge c \ne 1 \wedge (a + c/2)*m > n \Rightarrow a*m \le n < (a+c/2)*m \wedge \mathit{power2}(c/2)$$
$$a*m \le n < (a+c)*m \wedge \mathit{power2}(c) \wedge c \ne 1 \wedge n < (a+c/2)*m \Rightarrow a*m \le n < (a+c/2)*m \wedge \mathit{power2}(c/2)$$
$$a*m \le n < (a+c/2)*m \wedge \boldsymbol{\mathit{power2}(c) \wedge c \ne 1} \Rightarrow a*m \le n < (a+c/2)*m \wedge \boldsymbol{\mathit{power2}(c/2)}$$

[2] This explanation is by no means sufficient to understand how to obtain invariants in general or how to come up with this particular one. Finding invariants needs experience and is beyond the scope of this course.

[3] Verify this fact using a truth table or algebraic laws!

13 ⇒ 22, else-branch:

$Inv_2 \wedge c \neq 1 \wedge (a + c/2) * m \leq n \Rightarrow Inv_2[a/a + c][c/(c/2)]$
$Inv_2 \wedge c \neq 1 \wedge (a + c/2) * m \leq n \Rightarrow (a * m \leq n < (a + c) * m \wedge power2(c))[a/a + c][c/(c/2)]$
$Inv_2 \wedge c \neq 1 \wedge (a + c/2) * m \leq n \Rightarrow ((a + c) * m \leq n < (a + c + c) * m \wedge power2(c))[c/(c/2)]$
$Inv_2 \wedge c \neq 1 \wedge (a + c/2) * m \leq n \Rightarrow (a + c/2) * m \leq n < (a + 2(c/2)) * m \wedge power2(c/2)$
$a * m \leq n < (a + c) * m \wedge power2(c) \wedge c \neq 1 \wedge (a + c/2) * m \leq n$
$\Rightarrow (a + c/2) * m \leq n < (a + c) * m \wedge power2(c/2)$
$(a + c/2) * m \leq n < (a + c) * m \wedge \boldsymbol{power2(c) \wedge c \neq 1}$
$\Rightarrow (a + c/2) * m \leq n < (a + c) * m \wedge \boldsymbol{power2(c/2)}$

In both cases we observe that $c/2$ is a power of 2, since according to the premise c is a power of 2 different from 1, i.e., c is a power of 2 that is at least 2. The rest of the conclusion occurs identically in the premise of the implication.

15 ⇒ 3:

$$Inv_2 \wedge c = 1 \Rightarrow Post$$
$$a * m \leq n < (a + c) * m \wedge power2(c) \wedge c = 1 \Rightarrow a * m \leq n < (a + 1) * m$$
$$\boldsymbol{a * m \leq n < (a + 1) * m} \wedge power2(1) \wedge c = 1 \Rightarrow \boldsymbol{a * m \leq n < (a + 1) * m}$$

Termination

We choose c as variant for the second loop and thus have to show the correctness of the assertion $\{\, Inv_2 \wedge c \neq 1 \wedge c = t_0 \,\}$ loop body $\{\, 0 \leq c < t_0 \,\}$.

$\{\, 23\colon Inv_2 \wedge c \neq 1 \wedge c = t_0 \,\}$
$\{\, 30\colon 0 \leq c/2 < t_0 \,\}$
$c := c/2;$
$\{\, 29\colon \big((B \Rightarrow 0 \leq c < t_0) \wedge (\neg B \Rightarrow 0 \leq c < t_0)\big) \equiv 0 \leq c < t_0 \,\}$
if $(a + c) * m > n$ then
  $\{\, 27\colon 0 \leq c < t_0 \,\}$
  skip
  $\{\, 25\colon 0 \leq c < t_0 \,\}$
else
  $\{\, 28\colon 0 \leq c < t_0 \,\}$
  $a := a + c$
  $\{\, 26\colon 0 \leq c < t_0 \,\}$
fi
$\{\, 24\colon 0 \leq c < t_0 \,\}$

23 ⇒ 30:

$$Inv_2 \wedge c \neq 1 \wedge c = t_0 \Rightarrow 0 \leq c/2 < t_0$$
$$a * m \leq n < (a + c) * m \wedge \boldsymbol{power2(c)} \wedge c \neq 1 \wedge c = t_0 \Rightarrow \mathbf{0 \leq c/2 < c}$$

According to the premise c is a power of two, hence $c/2$ is non-negative as well as less than c.

3 Automatising the Hoare calculus

As we see above, a small program gives already rise to numerous formulas. In fact, real-world programming languages with their type systems generate even more verification conditions, since type compatibility has to be checked at each step. Most validity proofs are straight-forward, but require careful thinking to make sure that all conditions are indeed met. Doing the proofs by hand is tedious and error-prone, therefore automatisation is an important issue to make formal verification of software work in practice.

One particular system for verifying software is *Perfect Developer*[4], which uses its own object-oriented language for specifying algorithms (called *Perfect*) and which generates Java, Ada, or C++ code from the verified program. As an example, the program above takes the following form in Perfect (the keywords `keep` and `decreases` introduce the invariant and variant, respectively).

```
ghost function power2(c: int): bool
decrease max(c,0)
^= c>0
 & ([c=1]:   true,
    [c%2=0]: power2(c/2),
    [c%2=1]: false
   );

function div(m,n: int): int
pre n>=0, m>0
satisfy result*m <= n < (result+1)*m
  var a,c: int;
  c! = 1;
  loop
    change c
    keep power2(c'), (c'/2)*m < c'*m
    until c'*m > n
    decrease n-c'*m;
    c! = 2*c
  end;
  a! = 0;
```

[4]Escher Technologies, `www.eschertech.com`

```
  loop
    change a,c
    keep power2(c'), a'*m <= n < (a'+c')*m
    until c' = 1
    decrease c';
    c! = c/2;
    if [ (a+c)*m > n ]:
      pass;
    []:
      a! = a+c;
    fi;
  end;
  value a;
end;
```

Perfect Developer generates 26 verification conditions and (in this case) proves all of them fully automatically.

References

[1] George S. Boolos, John P. Burgess, and Richard C. Jeffrey. *Computability and Logic.* Cambridge University Press, 2003.

[2] David Gries. *The Science of Programming.* Springer, 1981.

[3] Gernot Salzer. Formal verification of software. Lecture notes, Technische Universität Wien, `http://www.logic.at/lvas/fminf/`, April 2010.