Annotated Programs

Engelbert Hubbers

September 6, 2011

Introduction

In chapter 6 of their book [1], Nielson and Nielson introduce the system of axiomatic program verification. The rules for this system are given as axioms and derivation rules in Table 6.1 on page 178. In particular this means that proofs in this system are always given as derivation trees. Unfortunately these trees can get very wide and the general proof is a bit hard to follow: you basically start with a precondition at the bottom of the tree, work through all smaller trees in a depth first scenario to finally end with the desired post-condition again at the bottom of the tree. In order to improve the readability of the proofs, we introduce a different notation that allows us to write the correctness proofs in a short and comprehensible way.

1 Intuitive annotations

The idea is that we write the main precondition above the first line of code, all intermediate predicates between the code, and the final post-condition below the last line of code. Let's give an example.

Consider the program `while` true `do skip`. We would like to prove here[1] that the assertion

$$\{\text{true}\}\ \texttt{while}\ \text{true}\ \texttt{do skip}\ \{\text{false}\}$$

holds Figure 1 shows the proof in terms of trees as defined in [1]. Note that the numbers attached

$$\dfrac{\dfrac{\dfrac{}{\{\text{true}\}^4\ \texttt{skip}\ \{\text{true}\}^5}\,[\text{ass}_\text{p}]}{\{\text{true}\wedge\text{true}\}^3\ \texttt{skip}\ \{\text{true}\}^6}\,[\text{cons}_\text{p}]}{\{\text{true}\}^2\ \texttt{while}\ \text{true}\ \texttt{do skip}\ \{\neg\text{true}\wedge\text{true}\}^7}\,[\text{while}_\text{p}]$$
$$\dfrac{\{\text{true}\}^2\ \texttt{while}\ \text{true}\ \texttt{do skip}\ \{\neg\text{true}\wedge\text{true}\}^7}{\{\text{true}\}^1\ \texttt{while}\ \text{true}\ \texttt{do skip}\ \{\text{false}\}^8}\,[\text{cons}_\text{p}]$$

Figure 1: Proof in normal tree style

to the predicates are officially not part of this tree style and therefore they are displayed in a different color. They are here only to match a specific predicate in this tree to a specific predicate in the annotated program style.

Figure 2 shows two versions of the proof in our annotated program style.

[1] Note that in [1] a different assertion is proved: {true} `while` true `do skip` {true}. Convince yourself that both assertions are indeed correct.

Figure 2: Proofs in annotated program style

But how should we read this? Let's have a look at the proof on the left. From top to bottom, each time we encounter another predicate, it should follow from the one directly above. If there are no statements between these predicates, the lower predicate should be directly implied by the one above as is for instance the case for predicates 7 and 8. And if there are statements between the predicates as is the case with predicates 4 and 5, it means that the one above the statement acts as a precondition to this statement, and the one below acts as a post-condition to it. As a consequence, the numbering of the predicates is now in a normal order. At the first line the initial precondition is written and at the last line the desired post-condition is written.

Note that this numbering is not part of the official style. As stated before, it is only used to indicate which predicate in the program corresponds to which predicate in the tree. In particular, the numbers help us to recognize the rules being used. Predicates 1, 2, 7 and 8 correspond with the lower [cons_p] rule in the tree. Predicates 2, 3, 6 and 7 correspond with the [while_p] rule. Predicates 3, 4, 5 and 6 correspond with the upper [cons_p] rule. Predicates 4 and 5 correspond with the [skip_p] axiom.

Now if we take a look at the proof on the right, we see that the predicates 2 and 6 have disappeared. This is because they are exactly the same as the predicates 1 and 5 directly above them. In general a [cons_p] rule deals with four predicates. However, in case we are only weakening the postcondition, but keep the same precondition like we did in the lower [cons_p] rule, we do not copy the precondition, but simply write it down once. And likewise if we are only weakening the precondition as we did with the upper [cons_p] rule. The result is the proof on the right.

2 Formalization

Now that we have explained the idea behind the notation, we can introduce it formally by giving the translation for all axioms and rules. See Table 1 for the details.

In addition to this table we have to make two remarks.

1. As we have seen before the general [cons_p] rule deals with four predicates, but in case $P = P'$ or $Q = Q'$, the predicate will be listed only once.

2. Since there is no [comp_p] rule for a composition of three or more statements, the derivation tree style automatically treats $S_1; S_2; S_3$ as a nested composition of $S_1; (S_2; S_3)$ or $(S_1; S_2); S_3$. Table 2 shows how we deal with this nesting in the annotated program style. As you can see, again we opt for eliminating duplicates as much as possible.

Table 1: Transformation between tree style presentation and annotated program style

Rule	Tree style	Program style
$[\text{ass}_\text{p}]$	$\{P\,[x \mapsto \mathcal{A}[\![a]\!]]\}\ \ x := a\ \ \{P\}$	$\{P\,[x \mapsto \mathcal{A}[\![a]\!]]\}$ `x:=a` $\{P\}$
$[\text{skip}_\text{p}]$	$\{P\}\ \texttt{skip}\ \{P\}$	$\{P\}$ `skip` $\{P\}$
$[\text{comp}_\text{p}]$	$\dfrac{\{P\}\ \ S_1\ \ \{Q\} \quad \{Q\}\ \ S_2\ \ \{R\}}{\{P\}\ \ S_1; S_2\ \ \{R\}}$	$\{P\}$ S_1 $\{Q\}$ S_2 $\{R\}$

$[\text{if}_\text{p}]$	$\dfrac{\{\mathcal{B}[\![b]\!] \wedge P\}\ \ S_1\ \ \{Q\} \quad \{\neg\mathcal{B}[\![b]\!] \wedge P\}\ \ S_2\ \ \{Q\}}{\{P\}\ \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \ \{Q\}}$	$\{P\}$
		`if` b
		`then`
		$\quad\{\mathcal{B}[\![b]\!] \wedge P\}$
		$\quad S_1$
		$\quad\{Q\}$
		`else`
		$\quad\{\neg\mathcal{B}[\![b]\!] \wedge P\}$
		$\quad S_2$
		$\quad\{Q\}$
		$\{Q\}$

$[\text{while}_\text{p}]$	$\dfrac{\{\mathcal{B}[\![b]\!] \wedge P\}\ \ S\ \ \{P\}}{\{P\}\ \texttt{while}\ b\ \texttt{do}\ S\ \ \{\neg\mathcal{B}[\![b]\!] \wedge P\}}$	$\{P\}$ `while` b `do` $\quad\{\mathcal{B}[\![b]\!] \wedge P\}$ $\quad S$ $\quad\{P\}$ $\{\neg\mathcal{B}[\![b]\!] \wedge P\}$
$[\text{cons}_\text{p}]$	$\dfrac{\{P'\}\ \ S\ \ \{Q'\}}{\{P\}\ \ S\ \ \{Q\}}$ if $P \Rightarrow P'$ and $Q' \Rightarrow Q$	$\{P\}$ $\{P'\}$ S $\{Q'\}$ $\{Q\}$

Table 2: Nested compositions

Tree style	Program style
$\dfrac{\{P\}\ \ S_1\ \ \{Q\} \qquad \dfrac{\{Q\}\ \ S_2\ \ \{R\} \quad \{R\}\ \ S_3\ \ \{T\}}{\{Q\}\ \ S_2; S_3\ \ \{T\}}\,[\mathrm{comp_p}]}{\{P\}\ \ S_1; S_2; S_3\ \ \{T\}}\,[\mathrm{comp_p}]$	$\{P\}$ S_1 $\{Q\}$ S_2 $\{R\}$ S_3 $\{T\}$

3 Example

Figure 3 shows a program which is known as Euclid's algorithm. We want to prove that this

```
x:=m;
y:=n;
while x≠y
do
  if x>y
  then
    x:=x-y;
  else
    y:=y-x;
g:=x;
```

Figure 3: Euclid's algorithm

algorithm computes the greatest common divisor of the values m and n. Or in other words, we want to prove

$$\{m > 0 \wedge n > 0\}\ \ S\ \ \{g = \gcd(m, n)\} \tag{1}$$

where S is the whole program. In order to prove this we may assume that

$$u > v > 0 \text{ implies } \gcd(u, v) = \gcd(u - v, v) \tag{2}$$

Figure 4 gives a proof in tree style. It automatically shows the problem with this type of proofs: even if we abbreviate all predicates, it is still difficult to print the whole tree on one sheet of A4-paper! Before we will go into the details of the proof itself, we present the same proof in annotated program style in Figure 5. The first thing that strikes is that the representation of this proof is short enough to allow us to include some extra information:

- Line numbers that we can use later on to explain the individual steps in the proof
- The abbreviations used in Figure 4.

As before, we use a different color for this extra information, which is not officially part of the style.

We complete this example by explaining all the steps in the proof of Figure 5, which implicitly also explains all the steps in the proof of Figure 4.

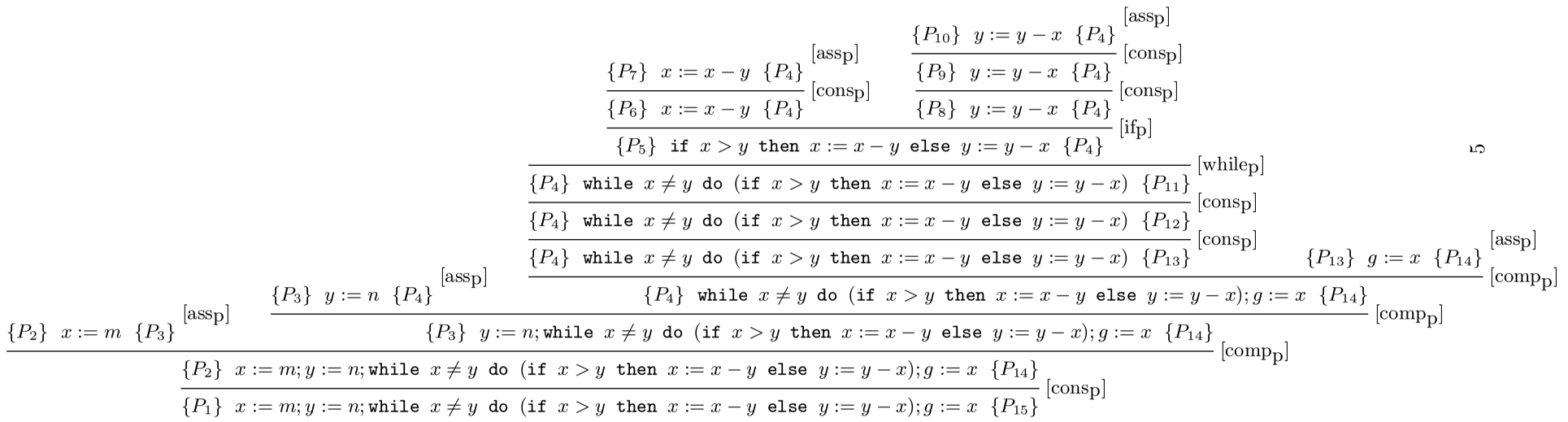

Figure 4: Euclid's algorithm proved by a tree

01. $\{m > 0 \wedge n > 0\}$ = $\{P_1\}$
02. $\{m > 0 \wedge n > 0 \wedge m > 0 \wedge n > 0 \wedge \gcd(m,n) = \gcd(m,n)\}$ = $\{P_2\}$
`x:=m;`
03. $\{m > 0 \wedge n > 0 \wedge x > 0 \wedge n > 0 \wedge \gcd(x,n) = \gcd(m,n)\}$ = $\{P_3\}$
`y:=n;`
04. $\{m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_4\}$
`while x≠y`
`do`
05. $\{x \neq y \wedge m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_5\}$
`if x>y`
`then`
06. $\{x > y \wedge x \neq y \wedge m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_6\}$
07. $\{m > 0 \wedge n > 0 \wedge x - y > 0 \wedge y > 0 \wedge \gcd(x-y,y) = \gcd(m,n)\}$ = $\{P_7\}$
`x:=x-y;`
08. $\{m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_4\}$
`else`
09. $\{\neg(x > y) \wedge x \neq y \wedge m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_8\}$
10. $\{y > x \wedge m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_9\}$
11. $\{m > 0 \wedge n > 0 \wedge x > 0 \wedge y - x > 0 \wedge \gcd(x,y-x) = \gcd(m,n)\}$ = $\{P_{10}\}$
`y:=y-x;`
12. $\{m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_4\}$
13. $\{m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_4\}$
14. $\{\neg(x \neq y) \wedge m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_{11}\}$
15. $\{x = y \wedge m > 0 \wedge n > 0 \wedge x > 0 \wedge y > 0 \wedge \gcd(x,y) = \gcd(m,n)\}$ = $\{P_{12}\}$
16. $\{m > 0 \wedge n > 0 \wedge x = \gcd(m,n)\}$ = $\{P_{13}\}$
`g:=x;`
17. $\{m > 0 \wedge n > 0 \wedge g = \gcd(m,n)\}$ = $\{P_{14}\}$
18. $\{g = \gcd(m,n)\}$ = $\{P_{15}\}$

Figure 5: Euclid's algorithm proved by an annotated program

1. `01-->02`: $[\text{cons}_p]$, elementary mathematics (EM).
2. `02-->03`: $[\text{ass}_p]$, $P_2 = P_3\,[x \mapsto \mathcal{A}[\![m]\!]]$.
3. `03-->04`: $[\text{ass}_p]$, $P_3 = P_4\,[y \mapsto \mathcal{A}[\![n]\!]]$.
4. `04-->14`: $[\text{while}_p]$, based upon `04`, `05`, `13` and `14`. Note that P_4 is the loop invariant, $P_5 = x \neq y \wedge P_4$ and $P_{11} = \neg(x \neq y) \wedge P_4$.
5. `05-->13`: $[\text{if}_p]$, based upon `05`, `06`, `08`, `09`, `12` and `13`. Note that $P_6 = x > y \wedge P_5$, and $P_8 = \neg(x > y) \wedge P_5$.
6. `06-->07`: $[\text{cons}_p]$, based upon (2).
7. `07-->08`: $[\text{ass}_p]$, $P_7 = P_4\,[x \mapsto \mathcal{A}[\![x - y]\!]]$.
8. `09-->10`: $[\text{cons}_p]$, EM.
9. `10-->11`: $[\text{cons}_p]$, based upon (2).
10. `11-->12`: $[\text{ass}_p]$, $P_{10} = P_4\,[y \mapsto \mathcal{A}[\![y - x]\!]]$.

11. `14-->15`: $[\text{cons}_p]$, EM.
12. `15-->16`: $[\text{cons}_p]$, $x = y \wedge x > 0 \wedge y > 0$ implies $\gcd(x, y) = x$.
13. `16-->17`: $[\text{ass}_p]$, $P_{13} = P_{14}\,[g \mapsto \mathcal{A}[\![x]\!]]$.
14. `17-->18`: $[\text{cons}_p]$, EM.

References

[1] Hanne Riis Nielson and Flemming Nielson. *Semantics with Applications: A Formal Introduction.* Wiley, 1999 revised edition, 1992.