1[9] Let \( a \), \( b \), and \( c \) be integer variables. Express the following as simply as possible without using quantifiers, assignments, or dependent compositions. Prove.
\[
b := a - b. \quad b := a - b
\]
\[
\begin{align*}
& b := a - b. \quad b := a - b \\
& \quad \quad \quad \quad \quad \text{expand last assignment} \\
\Rightarrow & b := a - b. \quad a' = a \land b' = a - b \land c' = c \\
\quad \quad \quad \quad \quad \text{Substitution Law} \\
\Rightarrow & a' = a \land b' = a - (a - b) \land c' = c \\
\Rightarrow & a' = a \land b' = b \land c' = c \\
\Rightarrow & \text{ok}
\end{align*}
\]

2 Let \( x \) be an integer variable. Let \( P \) be a specification refined as follows.
\[
P \iff \begin{cases} 
\text{if } x > 0 \text{ then } x := x - 1. \ P \\
\text{else if } x < 0 \text{ then } x := x + 1. \ P \\
\text{else } \text{ok fi fi}
\end{cases}
\]

(a)[15] Prove the refinement when \( P \iff x' = 0 \).
\[
\begin{align*}
& x' = 0 \iff x > 0 \land (x := x - 1. \ x' = 0) \\
& x' = 0 \iff x < 0 \land (x := x + 1. \ x' = 0) \\
& x' = 0 \iff x = 0 \land \text{ok}
\end{align*}
\]

Let's start with the first.
\[
\begin{align*}
& x > 0 \land (x := x - 1. \ x' = 0) \\
\Rightarrow & x > 0 \land x' = 0 \\
\Rightarrow & x' = 0
\end{align*}
\]

Now the middle one.
\[
\begin{align*}
& x < 0 \land (x := x + 1. \ x' = 0) \\
\Rightarrow & x < 0 \land x' = 0 \\
\Rightarrow & x' = 0
\end{align*}
\]

And the last one.
\[
\begin{align*}
& x = 0 \land \text{ok} \\
\Rightarrow & x = 0 \land x' = x \\
\Rightarrow & x' = 0
\end{align*}
\]

(b)[15] Add recursive time and find and prove an upper bound for the execution time.
\[
P \iff \begin{cases} 
\text{if } x > 0 \text{ then } x := x - 1. \ t := t + 1. \ P \\
\text{else if } x < 0 \text{ then } x := x + 1. \ t := t + 1. \ P \\
\text{else } \text{ok fi fi}
\end{cases}
\]

The exact execution timing specification is
\[
P \iff t' = t + \text{abs} \ x
\]

Using refinement by cases, I must prove three things:
\[
\begin{align*}
& t' = t + \text{abs} \ x \iff x > 0 \land (x := x - 1. \ t := t + 1. \ t' = t + \text{abs} \ x) \\
& t' = t + \text{abs} \ x \iff x < 0 \land (x := x + 1. \ t := t + 1. \ t' = t + \text{abs} \ x) \\
& t' = t + \text{abs} \ x \iff x = 0 \land \text{ok}
\end{align*}
\]

Let's start with the first.
\[
\begin{align*}
& x > 0 \land (x := x - 1. \ t := t + 1. \ t' = t + \text{abs} \ x) \\
\Rightarrow & x > 0 \land t' = t + 1 + \text{abs} (x - 1) \\
\Rightarrow & x > 0 \land t' = t + \text{abs} \ x \\
\Rightarrow & t' = t + \text{abs} \ x
\end{align*}
\]

Now the middle one.
$x < 0 \land (x := x + 1. \ t := t + 1. \ t' = t + \text{abs} \ x)$

use substitution law twice

$\Rightarrow x < 0 \land t' = t + 1 + \text{abs}(x+1)$
when $x < 0, 1 + \text{abs}(x+1) = \text{abs} \ x$

specialization

$\Rightarrow t' = t + \text{abs} \ x$

And the last one.

$x = 0 \land \text{ok}$

replace ok

when $x = 0, \text{abs} \ x = 0$

specialization

$\Rightarrow t' = t + \text{abs} \ x$

3[15] Let do P od be a loop notation, and within P let exit n when b mean that if b is T
then execution exits n enclosing loops, and if b is ⊥ then execution continues with
whatever follows exit n when b. Here is a nest of loops. All exits are shown. What
refinements need to be proven in order to prove that this nest of loops refines
specification S? Hint: further specifications will be needed.

§ Each loop needs a specification. Using P, Q, and R for the inner loops,

$S \Leftarrow A. \ P$

$P \Leftarrow B. \ if \ u \ then \ ok$

else C. if v then E. if w then ok

else F. Q fi

else D. P fi fi

$Q \Leftarrow G. \ R$

$R \Leftarrow H. \ if \ x \ then \ K. \ if \ z \ then \ M. \ S$

else L. Q fi

else I. if y then M. S

else J. R fi fi
4[9] Using recursive data definition, without using \textit{nat}, define \textit{dec} as the bunch of all and only the decimal-point numbers. These are the rationals that can be expressed as a finite string of decimal digits containing a decimal point. Note: you are defining a bunch of numbers, not a bunch of texts.

§ If we're just allowed digits and a decimal point, they're non-negative.
\begin{equation*}
\begin{aligned}
0, \text{dec+1}, \text{dec}/10 : \text{dec} & \quad \text{construction} \\
0, B+1, B/10 : B \Rightarrow \text{dec: B} & \quad \text{induction}
\end{aligned}
\end{equation*}
\text{or}
\begin{equation*}
\begin{aligned}
de\text{c} &= 0, B, B/10 : \text{dec} & \quad \text{fixed-point construction} \\
B &= 0, B+1, B/10 \Rightarrow \text{dec: B} & \quad \text{fixed-point induction}
\end{aligned}
\end{equation*}

5 Here is a theory.
\begin{itemize}
\item[] \textit{start: chain}
\item[] \textit{link: chain}→chain
\item[] \textit{isStart: chain}→bin
\item[] \textit{isStart start}
\item[] \forall c : \textit{chain} \Rightarrow \textit{isStart (link c)}
\end{itemize}
Here is a possible implementation of that theory.
\begin{itemize}
\item[] \textit{chain} = \textit{nat}
\item[] \textit{start} = 0
\item[] \textit{link} = \langle n : \textit{nat} \rightarrow n+1 \rangle
\item[] \textit{isStart} = \langle n : \textit{nat} \rightarrow n=0 \rangle
\end{itemize}

(a)[6] What must be proven to prove that this really is an implementation of the theory? You are not being asked to prove; just state what must be proven.

§ Let the conjunction of the five theory axioms be \textit{T}. Let the conjunction of the four implementation equations be \textit{I}. We must prove \textit{I}⇒\textit{T}.

(b)[3] How can we know whether the theory is consistent?

§ Theory \textit{T} is implemented using Number Theory and Function Theory. If those two theories are consistent, then the proof that \textit{I} implements \textit{T} is a proof of the consistency of \textit{T}.

6 Let \textit{a}, \textit{b} and \textit{x} be integer variables. Variables \textit{a} and \textit{b} are implementer's variables, and \textit{x} is a user's variable for the operations
\begin{itemize}
\item[] \textit{start} \equiv a:=0. \ b:=0
\item[] \textit{step} \equiv a:=a+1. \ b:=b–1
\item[] \textit{ask} \equiv x:=a–b
\end{itemize}
Reimplement this theory replacing the two old implementer's variables \textit{a} and \textit{b} with one new integer implementer's variable \textit{c}.

(a)[6] What is the data transformer?

§ \textit{c} = a–b
Using your data transformer, transform \(\textit{ask}\).

\[\forall a, b \cdot c = a-b \Rightarrow \exists a', b' \cdot c' = a'-b' \land (x:= a-b)\]

- replace assignment in old variables
- \(\forall a, b \cdot c = a-b \Rightarrow \exists a', b' \cdot c' = a'-b' \land a' = a \land b' = b \land x' = a-b\) one-point \(a', b'\)
- \(\forall a, b \cdot c = a-b \Rightarrow c' = a-b \land x' = a-b\) one-point \(a\) using \(a = b+c\)
- \(\forall b \cdot c' = b+c-b \land x' = b+c-b\) simplify
- \(\forall b \cdot c' = c \land x' = c\) This is an assignment in the new variable
- \(x := c\)

The program

\[(\text{chan } c: \text{int} \cdot c! 6)\]

declares fresh channel \(c\), then writes \(6\) on the channel, and that is the end of the scope of the channel. With read and write cursors \(r\) and \(w\), and additional state variable \(x\), and time variable \(t\), what are the final values of \(x\) and \(t\)? Prove.

\[\text{Expand chan, then var}\]

\[\exists \mathcal{M}: \infty \times \text{int} \cdot \exists \mathcal{T}: \infty \times \text{xnat} \cdot \exists r', w, w': \text{xnat} \]
\[r:= 0 \land w:= 0 \land \mathcal{M}_0 = 6 \land \mathcal{T}_0 = t \land (w:= w+1)\]

Replace final assignment, then substitution Law 2 times

\[\exists \mathcal{M}: \infty \times \text{int} \cdot \exists \mathcal{T}: \infty \times \text{xnat} \cdot \exists r', w, w': \text{xnat} \]
\[\mathcal{M}_0 = 6 \land \mathcal{T}_0 = t \land r' = 0 \land w' = 1 \land x' = x \land t' = t\]

One-point laws

\[x' = x \land t' = t\]

The theory says that \(x\) and \(t\) are unchanged.