TYPE SOUNDNESS PROOF OF SIMPLE LANGUAGE

We consider the following simple language with types `Pos, Int`. Programs consist of a number of variable declarations

```
var x: Pos
var y: Int
...
```

followed by statements of one of 3 forms:

```
z = x
z = x + y
z = x / y
```

In particular, we do not have complex expressions, if-then-else statements etc. We also make the assumption that the default initialization value for variables of type `Pos` is 1 and for `Int` is 0.
We have `Pos <: Int` and the following type rules:

$$\frac{}{\text{k: Pos}} \qquad \frac{}{\text{-k: Int}}$$

$$\frac{\Gamma \vdash x : T \qquad T <: T'}{\Gamma \vdash x : T'} \qquad \frac{(x, T) \in \Gamma}{\Gamma \vdash x : T}$$

$$\frac{(x, T) \in \Gamma \qquad \Gamma \vdash e : T}{\Gamma \vdash (x = e) : void}$$

$$\frac{e_1 : Int \qquad e_2 : Int}{e_1 + e_2 : Int} \qquad \frac{e_1 : Pos \qquad e_2 : Pos}{e_1 + e_2 : Pos}$$

$$\frac{e_1 : Int \qquad e_2 : Pos}{e_1 / e_2 : Int}$$

The **soundness property** we want to prove is that division-by-zero does not happen in any execution of a program that type checks according to the above rules.
We will prove this by induction on the program execution. For this we need to

(1) define a formal description of program execution (operational semantics)
(2) find an invariant
(3) prove this invariant is preserved for each execution step and initially
(4) show the invariant implies no division by zero

1) Operational semantics. We give the small-step semantics of the programs where we use the following:

V: set of variables in the program
pc: integer variable denoting the program counter
g: V → Int: function giving values to program variables
(g, pc): program state

We denote by g(x) the *value* mapped to x by g. We denote by g(z := e) the function update that returns the function that has the same values for all variables, except for z, which now maps to e.
Then, for the three possible statements in the program we get:

z = x: (g, pc) → (g', pc + 1) such that g' = g(z := g(x))
z = x + y: (g, pc) → (g', pc + 1) such that g' = g(z := g(x) + g(y))
z = x / y: (g, pc) → (g', pc + 1) such that g' = g(z := g(x) / g(y))

2) Invariant. The invariant is the following:
"A state is very good, if each variable belongs to the domain determined by its type."
This expresses in particular, that if a variable has type `Pos`, then we can deduce that it will be strictly greater than 0. That is, we assign a numerical interpretation to the types. Formally, we can write

$$\forall x \in V.\ (x, Pos) \in \Gamma \to g(x) > 0$$

We could equally write $\forall x \in V.\ (x, Pos) \in \Gamma \to g(x) \in \mathbb{N}_+$. The condition $\forall x \in V.\ (x, Int) \in \Gamma \to g(x) \in \mathbb{Z}$ trivially holds for all variables, thus we do not write it here.

3) Invariant is inductive. Now we show that the invariant is inductive. That is, first we argue that it holds in the initial state, which we can set to be after all the variable declarations have occurred. Since we assign 1 to all variables of type `Pos`, and $1 > 0$, the invariant holds.
Now we assume that the invariant holds in the current program state (g, pc) and that the program type checks and we show that for each possible statement the invariant still holds in the next state, i.e. we need to show that $\forall x \in V.\ (x, Pos) \in \Gamma \to g'(x) > 0$

z = x:

(1) $\forall a \in V.\ (a, Pos) \in \Gamma \to g'(a) > 0$

(2) $\forall a \in V.\ (a, Pos) \in \Gamma \to g(z := g(x))(a) > 0$

(3) $\forall a \in V.\ (a, Pos) \in \Gamma \to (a = z \wedge g(x) > 0) \vee (a \neq z \wedge g'(a) = g(a) > 0)$

where the second step follows by our operational semantics. What this expresses is that in the next step, the mapping of g is unchanged for all variables, except for z. For all the variables that are unchanged, the property clearly continues to holds, so we need to only show that it also holds for the new value of z.
Given that the program statement type checks, we have one of the following cases:

$$\frac{(z, Int) \in \Gamma \qquad \Gamma \vdash x =: Int}{\Gamma \vdash (z = x) : void} \qquad \frac{(z, Pos) \in \Gamma \qquad \Gamma \vdash x : Pos}{\Gamma \vdash (z = x) : void}$$

If $(z, Int) \in \Gamma$ (first case/type rule), then the condition in the invariant trivially holds, since the left-hand side of the implication: $(z, Pos) \in \Gamma$ evaluates to false. If $(z, Pos) \in \Gamma$, then also $(x, Pos) \in \Gamma$ and thus $g(x) > 0$ by inductive hypothesis, which makes the first conjunct true when $a = z$.

z = x + y: Similarly,

(4) $\forall a \in V.\ (a, Pos) \in \Gamma \to g'(a) > 0$

(5) $\forall a \in V.\ (a, Pos) \in \Gamma \to g(z := g(x) + g(y))(a) > 0$

(6) $\forall a \in V.\ (a, Pos) \in \Gamma \to (a = z \wedge g(x) + g(y) > 0) \vee (a \neq z \wedge g'(a) = g(a) > 0)$

If the program type checks, we have the following two possibilities:

$$\dfrac{(z, Int) \in \Gamma \qquad \dfrac{\Gamma \vdash x : Int \qquad \Gamma \vdash y : Int}{\Gamma \vdash x + y : Int}}{\Gamma \vdash (z = x + y) : void} \qquad \dfrac{(z, Pos) \in \Gamma \qquad \dfrac{\Gamma \vdash x : Pos \qquad \Gamma \vdash y : Pos}{\Gamma \vdash x + y : Pos}}{\Gamma \vdash (z = x + y) : void}$$

For the left case, the left-hand side of the implication is again false, hence the invariant holds. For the right case, since $x : Pos$ and $y : Pos$, we necessarily must have $(x, Pos) \in \Gamma$ and $(y, Pos) \in \Gamma$ and thus $g(x) > 0$ and $g(y) > 0$ by inductive hypothesis. $g(x) + g(y) > 0$ then follows.

z = x / y:

(7) $\forall a \in V.\ (a, Pos) \in \Gamma \to g'(a) > 0$

(8) $\forall a \in V.\ (a, Pos) \in \Gamma \to g(z := g(x)/g(y))(a) > 0$

(9) $\forall a \in V.\ (a, Pos) \in \Gamma \to (a = z \wedge g(x)/g(y) > 0) \vee (a \neq z \wedge g'(a) = g(a) > 0)$

If the program type checks, we have only the following possibility:

$$\dfrac{(z, Int) \in \Gamma \qquad \dfrac{\Gamma \vdash x : Int \qquad \Gamma \vdash y : Pos}{\Gamma \vdash x/y : Int}}{\Gamma \vdash (z = x/y) : void}$$

Again, since $(z, Int) \in \Gamma$, the left hand side of the implication is false for z, the invariant holds.

Invariant implies no crash. We have shown that for a program that type checks $\forall x \in V.\ (x, Pos) \in \Gamma \to g(x) > 0$ holds for all program states. The only place were the program could divide by zero is the division instruction, and we have only the following type rule that could apply:

$$\dfrac{x : Int \qquad y : Pos}{x/y : Int}$$

It follows from the invariant that $g(y) > 0$ and thus a division by zero is not possible.