Type Soundness & Subtyping

Exercise 1

Consider a simple programming language with integer arithmetic, boolean expressions and user-defined functions.

```
T := Int | Bool | (T₁, …, Tₙ) => T
t := true | false | c₁
   | t₁ == t₂ | t₁ + t₂ | t₁ && t₂
   | if (t₁) t₂ else t₃
   | f(t₁, … , tₙ) | x
```

Where c_1 represents integer literals, == represents equality (between integers, as well as between booleans), + represents the usual integer addition and && represents conjunction. The meta-variable f refers to names of user-defined function and x refers to names of variables.

You may assume that you have a fixed environment e which contains information about user-defined functions (i.e. the function arguments, their types, the function body and the result type).

Part 1

Write down the *"usual"* typing rules for this language.

Solution:

$$\frac{}{\Gamma \vdash \texttt{true} : \texttt{Bool}} \qquad \frac{}{\Gamma \vdash \texttt{false} : \texttt{Bool}} \qquad \frac{\texttt{c} \text{ is a literal integer}}{\Gamma \vdash \texttt{c} : \texttt{Int}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Int} \quad \Gamma \vdash t_2 : \texttt{Int}}{\Gamma \vdash t_1 \texttt{ == } t_2 : \texttt{Bool}} \qquad \frac{\Gamma \vdash t_1 : \texttt{Bool} \quad \Gamma \vdash t_2 : \texttt{Bool}}{\Gamma \vdash t_1 \texttt{ == } t_2 : \texttt{Bool}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Int} \quad \Gamma \vdash t_2 : \texttt{Int}}{\Gamma \vdash t_1 \texttt{ + } t_2 : \texttt{Int}} \qquad \frac{\Gamma \vdash t_1 : \texttt{Bool} \quad \Gamma \vdash t_2 : \texttt{Bool}}{\Gamma \vdash t_1 \texttt{ \&\& } t_2 : \texttt{Bool}}$$

$$\frac{\Gamma \vdash t_1 : \texttt{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \texttt{if (}t_1\texttt{) } t_2 \texttt{ else } t_3 : T}$$

$$\frac{\Gamma(\texttt{x}) = T}{\Gamma \vdash \texttt{x} : T} \qquad \frac{\Gamma \vdash f : (T_1, \ldots, T_n) \texttt{ => } T \quad \Gamma \vdash t_1 : T_1 \quad \ldots \quad \Gamma \vdash t_n : T_n}{\Gamma \vdash f(t_1, \ldots, t_n) : T}$$

Part 2

Inductively define the substitution operation for your terms, which replaces every free occurence of a variable in an expression by an expression *without free variables*.

Solution:

$$\frac{}{\texttt{true}[x := e] \to \texttt{true}} \qquad \frac{}{\texttt{false}[x := e] \to \texttt{false}} \qquad \frac{\texttt{c is a literal integer}}{\texttt{c}[x := e] \to \texttt{c}}$$

$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{(t_1 \texttt{ == } t_2)[x := e] \to (t_1' \texttt{ == } t_2')}$$

$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{(t_1 \texttt{ + } t_2)[x := e] \to (t_1' \texttt{ + } t_2')} \qquad \frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_2'}{(t_1 \texttt{ \&\& } t_2)[x := e] \to (t_1' \texttt{ \&\& } t_2')}$$

$$\frac{t_1[x := e] \to t_1' \qquad t_2[x := e] \to t_3' \qquad t_1[x := e] \to t_3'}{(\texttt{if } (t_1) \texttt{ } t_2 \texttt{ else } t_3)[x := e] \to (\texttt{if } (t_1') \texttt{ } t_2' \texttt{ else } t_3')}$$

$$\frac{x = y}{\texttt{y}[x := e] \to e} \qquad \frac{x \neq y}{\texttt{y}[x := e] \to \texttt{y}} \qquad \frac{t_1[x := e] \to t_1' \qquad \dots \qquad t_n[x := e] \to t_n'}{f(t_1, \dots, t_n)[x := e] \to f(t_1', \dots, t_n')}$$

Prove that substitution preserves the type of an expression, given that the variable and the expression have the same type.

Solution:

We are given an expression t, an identifier x and a replacement expression e.
We assume that t type checks to type T in an environment E where x is defined at type Tx (if x is not defined in the environment, the proof is trivial).
We also assume that the expression e type checks to Tx in the empty environment (e doesn't contain free variables).

The idea is simple. If t type check in environment E, then we must have a derivation tree for E |- t : T. We must also have a derivation for ø |- e : Tx. We can get a derivation tree for E |- t[x := e] : Tx by replacing, in the original derivation tree, all subtrees rooted at E' |- x : Tx by a derivation tree for E' |- e : Tx, which is possible due to ø |- e : Tx. Qed.

Part 3

Write the operational semantics rules for the language, assuming ***call-by-name*** semantics for function calls. In call-by-name semantics, the arguments of a function are not evaluated before the call. In your operational semantics, parameters in the function body are to merely be substituted by the corresponding unevaluated argument expression.

$$\frac{\begin{array}{c} b_0 \texttt{ is the body of } f\texttt{, and } x_i \texttt{ are the } n \texttt{ parameters of } f \\ b_0[x_1 := t_1] \to b_1 \qquad \ldots \qquad b_{n-1}[x_n := t_n] \to b_n \end{array}}{f(t_1, \ldots, t_n) \rightsquigarrow b_n}$$

Part 4

Adapt the soundness proof seen in the last lecture to account for the new semantics. Prove only the cases related to function application.

Solution:

Progress is straightforward, as the call-by-name rule always applied. Preservation follows from the fact that substitution preserves types.

Exercise 2

In this second exercise, we will have a look at a simple programming language with the following types and terms:

```
T := Integer | Pos | Neg
t := c₁ | t₁ + t₂ | t₁ * t₂ | t₁ / t₂
```

`Integer` is the type of all integer numbers, while `Pos` is the type of all *strictly* positive integer numbers and `Neg` the type of all *strictly* negative numbers. Note that, interestingly, some terms will accept multiple types.

For instance, `14` will have the types `Integer` and `Pos`, while `-2` will have the types `Integer` and `Neg`. The constant `0` on the other hand will only have the type `Integer`.

Part 1

Write down typing rules for the terms of the language. Try to preserve information about positivity and negativity. Also, make sure that your type system prohibits division by zero.

Solution:

$$\frac{c \text{ is a literal}}{c : \texttt{Integer}} \qquad \frac{c \text{ is a positive literal}}{c : \texttt{Pos}} \qquad \frac{c \text{ is a negative literal}}{c : \texttt{Neg}}$$

$$\frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Integer}}{t_1 + t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Pos}}{t_1 + t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Neg}}{t_1 + t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Integer}}{t_1 + t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Pos}}{t_1 + t_2 : \texttt{Pos}} \qquad \frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Neg}}{t_1 + t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Integer}}{t_1 + t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Pos}}{t_1 + t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Neg}}{t_1 + t_2 : \texttt{Neg}}$$

$$\frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Integer}}{t_1 * t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Pos}}{t_1 * t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Neg}}{t_1 * t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Integer}}{t_1 * t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Pos}}{t_1 * t_2 : \texttt{Pos}} \qquad \frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Neg}}{t_1 * t_2 : \texttt{Neg}}$$

$$\frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Integer}}{t_1 * t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Pos}}{t_1 * t_2 : \texttt{Neg}} \qquad \frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Neg}}{t_1 * t_2 : \texttt{Pos}}$$

$$\frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Pos}}{t_1 / t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Neg}}{t_1 / t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Pos}}{t_1 / t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Neg}}{t_1 / t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Pos}}{t_1 / t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Neg}}{t_1 / t_2 : \texttt{Integer}}$$

Part 2

Under your type system, what are the types, if any, of the following terms? Write down a derivation for each possible type.

Solution: (Derivations not shown)

```
1 + 1  // Integer, Pos

-2 * 4  // Integer, Neg

-1 * (2 + -1) // Integer

7 / (18 + -1) // No types.
```

Part 3

We now introduce a new relation, T_1 `<:` T_2, which we call the *subtyping* relation.

T_1 `<:` T_2 can be read as "T_1 is a subtype of T_2". When T_1 `<:` T_2, terms of type T_1 can safely be used in the context where terms of type T_2 are expected. In this exercise, what pairs of types can be made part of this subtyping relation? List all such possible pairs.

Solution:

```
Integer <: Integer
Pos <: Pos
Neg <: Neg
Pos <: Integer
Neg <: Integer
```

Part 4

Write down the *subsumption* rule, which bridges the gap between the subtyping relation and the typing relation. The rule should state that if a term has a type T_1 and T_1 is a subtype of T_2, then the term has also type T_2.

Solution:

$$\frac{t : T_1 \qquad T_1 <: T_2}{t : T_2}$$

Now that you have defined this rule, can you remove some of the typing rules you had previously defined for the various constructs of the language ?

Solution: *(Remaing rules shown)*

$$\frac{}{0 : \texttt{Integer}} \qquad \frac{c \text{ is a positive literal}}{c : \texttt{Pos}} \qquad \frac{c \text{ is a negative literal}}{c : \texttt{Neg}}$$

$$\frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Integer}}{t_1 + t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Pos}}{t_1 + t_2 : \texttt{Pos}}$$

$$\frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Neg}}{t_1 + t_2 : \texttt{Neg}}$$

$$\frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Integer}}{t_1 * t_2 : \texttt{Integer}}$$

$$\frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Pos}}{t_1 * t_2 : \texttt{Pos}} \qquad \frac{t_1 : \texttt{Pos} \quad t_2 : \texttt{Neg}}{t_1 * t_2 : \texttt{Neg}}$$

$$\frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Pos}}{t_1 * t_2 : \texttt{Neg}} \qquad \frac{t_1 : \texttt{Neg} \quad t_2 : \texttt{Neg}}{t_1 * t_2 : \texttt{Pos}}$$

$$\frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Pos}}{t_1 / t_2 : \texttt{Integer}} \qquad \frac{t_1 : \texttt{Integer} \quad t_2 : \texttt{Neg}}{t_1 / t_2 : \texttt{Integer}}$$

Part 5

Let's now expand our language and add a primitive "power" function to it:

$t := \ldots \mid \texttt{power}(t_1, t_2)$

With the following typing rule:

$$\frac{\Gamma \vdash t_1 : \texttt{Integer} \qquad \Gamma \vdash t_2 : \texttt{Integer}}{\Gamma \vdash \texttt{power}(t_1, t_2) : \texttt{Integer}}$$

Typecheck the following expression under the empty environment. Show a type derivation.

```
power(7 / 2, 3)
```

Solution:

$$\dfrac{\dfrac{\dfrac{}{7 : \texttt{Pos}} \quad \texttt{Pos} <: \texttt{Integer}}{7 : \texttt{Integer}} \qquad \dfrac{}{2 : \texttt{Pos}}}{7/2 : \texttt{Integer}} \qquad \dfrac{\dfrac{}{3 : \texttt{Pos}} \quad \texttt{Pos} <: \texttt{Integer}}{3 : \texttt{Integer}}$$

$$\texttt{power}(7/2, 3) : \texttt{Integer}$$

Does there exist multiple valid type derivations that assign the same type to the above expression?

Solution: With the modified typing rules of part 4, there are multiples possible derivation trees. Indeed, it is possible to apply the subsumption rule an arbitrarily number of times by having the two types be equal. Ignoring that trivial transformation, then there is only one possible derivation.