Assignment 4
Subtyping

15-836: Substructural Logics
Frank Pfenning

Due Friday, October 6, 2023
90 points

In this assignment we explore subtyping on message-passing programs.
You should hand in two files:

- `hw4.pdf` with the solutions to Problems 1 and 2.
- `hw4.mps` with the programs written in the MPASS language as specified in Problem 3. This file is autograded.

The handout has the following structure:

- `src/`, the implementation of MPass.
- `starter/`, the starter code for this assignment, including a template for `hw4.mps`
- `hw4.pdf`, this assignment spec
- `*.{tex,sty}`, LaTeX source, macros, and style files

1 Admissibility of Subsumption (30 pts)

Here are the properties of left and right subsumption, formulated as an admissible inference rule.

$$\frac{\Delta \vdash P :: (x : A) \quad A \leq B}{\Delta \vdash P :: (x : B)}\ \mathsf{SubR} \qquad \frac{A \leq B \quad \Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \vdash Q :: (z : C)}\ \mathsf{SubL}$$

Task 1 Prove that left and right subsumption are admissible. You only need to show the cases for $\multimap R$, $\multimap L^*$ (as revised in Lecture 8), and id. Clearly state the overall structure of your proof. You may use reflexivity and transitivity of subtyping as you see fit.

2 Generating Counterexamples (30 pts)

Task 2 Define a judgment for $A \not\leq B$ by inference rules. Your judgment should be defined *inductively* and every proof should correspond to a counterexample. For this task it is important that we do not allow $\oplus\{\,\}$ and $\&\{\,\}$ (a restriction also enforced by MPASS). Channels of these types would not allow any messages to be sent along them.

We defined message prefixes (inductively) by

$$\begin{array}{lrcl} \text{Message prefixes} & p & ::= & !(\,) \mid !k.p \mid !(p).\_ \mid !(\_).p \\ & & \mid & ?k.p \mid ?(p).\_ \mid ?(\_).p \\ & & \mid & \_ \end{array}$$

Here, ! means a message is sent by the provider, ? means a message is received by the provider. The underscore '_' is the end of the prefix. Message prefixes they have the following interpretation:

- $(\,)$ is the unit message, without a continuation
- $k.p$ is a label k followed by a prefix p
- $(p).\_$ represents (sending or receiving) of a channel b with continued messages p on b
- $(\_).p$ represents the (sending or receiving) of a channel b, with continued messages p on the original channel a
- _ represents the end of the message prefix.

We use message prefixes to express counterexamples to subtyping. For example, the message prefix !succ.!zero._ is a counterexample to nat $\leq$ even as shown in the Lecture 8 notes. The prefix ?ins._ is a counterexample to $\mathsf{store}^2 \leq \mathsf{store}^1$. If we define lists of binary numbers (list, as defined in lecture) and lists of binary numbers with no leading zeros (std_list), then the prefix !cons.!(!b0.!e._)._ would be a counterexample to list $\leq$ std_list.

Task 3 Extend the judgment $A \not\leq B$ to $p : A \not\leq B$ where p is a message prefix that is a counterexample to $A \leq B$. Show the instrumented inference rules.

Task 4 Define a family of type store_A, parameterized over a type A as

$$\begin{array}{l} \mathsf{store}_A = \&\{\ \mathsf{ins} : A \multimap \mathsf{store}_A, \\ \qquad\qquad \mathsf{del} : \oplus\{\ \mathsf{none} : \mathbf{1}, \mathsf{some} : A \otimes \mathsf{store}_A\ \}\ \} \end{array}$$

Determine whether $\mathsf{store}_{\mathsf{pos}} \leq \mathsf{store}_{\mathsf{std}}$ and $\mathsf{store}_{\mathsf{std}} \leq \mathsf{store}_{\mathsf{pos}}$, where pos and std are defined as in lecture. For those that are false, give counterexamples. You do not need to show any derivations in your system, but you should ensure they exists.

3 Programming with Subtyping (30 pts)

In this problem you will explore the extent to which you can successfully use subtyping in MPASS. These problems will be autograded. Remember that you have to invoke MPASS with the `--subtyping` flag.

In all cases you may define auxiliary types and processes as you see fit. Not all of the types of processes may be implementable with the proposed typing, so be on your toes.

Task 5 Write, with the given more precise types:

```
proc zero (x : std) =
proc succ (x : pos) (y : std) =
proc pred (x : std) (y : pos) =
proc dbl (x : std) (y : std) =
```

Here, *dbl* is supposed to double the number received on y and send the bits along x. The predecessor *pred* is like the process from the previous assignment, except that is it guaranteed to get a positive number as input.

Task 6 On binary numbers (this time leading zeros are allowed), define mult3 $\leq$ bin such that mult3 contains exactly the multiples of three. Then define a function times3 that multiplies its input by 3.

```
type mult3 =
proc times3 (x : mult3) (y : bin) =
```

In the next task we deal with infinite streams of zeros and ones.

```
type stream = +{'b0 : stream, 'b1 : stream}
```

Task 7 Define a subtype `stream0x0` of `stream` where there are no two consecutive zeros, and a processes that takes an arbitrary stream and compresses consecutive zeros into a single zero and passes the ones through unchanged.

```
type stream0x0 =
proc compress (x : stream0x0) (y : stream) =
```