

# Theory exercises

Program verification with types and logic (NWI-IMC060)

## Week 6

1. In this exercise, you will extend the lambda calculus with mutable state that we have seen during the lectures and the Coq exercises with different forms of loops.
  - (a) Extend the big- and small-step operational semantics with a **do**  $e_1$  **while**  $e_2$  construct as found in programming languages like C and Java. Explain how your semantics corresponds to the intuitive semantics. Make sure that you define the semantics in such a way that the big- and the small-step semantics coincide. Motivate why this is the case.
  - (b) Extend the big- and small-step operational semantics with a **for**  $n = \{e_1 \dots e_2\}$  **do**  $e_3$  construct as found in programming languages like Bash and Basic. Explain how your semantics corresponds to the intuitive semantics. Make sure that you define the semantics in such a way that the big- and the small-step semantics coincide. Motivate why this is so.
2. Consider the syntax of the programming language TL:

$$\begin{aligned} A &::= \mathbf{unit} \mid \mathbf{nat} \mid \mathbf{ref}(A) \\ v &::= () \mid n \mid l \\ e &::= x \mid v \mid \mathbf{let} \ x = e_1 \ \mathbf{in} \ e_2 \mid \mathbf{alloc} \ e \mid !e \mid e_1 \leftarrow e_2 \mid \mathbf{covfefe} \end{aligned}$$

TL has three types: the unit type **unit** (whose only value is the unit value  $()$ ), natural numbers **nat** (whose values are the numerals  $n$ ), and references **ref**( $A$ ) to type  $A$  (whose values are the locations  $l$ ). Apart from the conventional constructs for let-bindings and the heap operations (**alloc**  $e$  to allocate  $e$ , and  $!e$  to dereference  $e$ , and  $e_1 \leftarrow e_2$  to assign  $e_2$  to  $e_1$ ), TL has a new construct called **covfefe**, whose typing rules and small-step operational semantics are as follows:

$$\frac{}{\Gamma \mid \Sigma \vdash \mathbf{covfefe} : \mathbf{unit}} \quad \frac{}{(\mathbf{covfefe}, h) \Rightarrow ((), h)} \quad \frac{h(l) \neq \mathbf{None}}{(\mathbf{covfefe}, h) \Rightarrow ((), h[l := 13])}$$

Recall that heaps  $h$  are maps from locations to values, and heap typings  $\Sigma$  are maps from locations to types. The typing rules and rules of the operational semantics for the other constructs are standard. The notation  $h[l := 13]$  sets the value of location  $l$  to 13 in heap  $h$ .

- (a) Does the language TL enjoy the *progress* property? You should carefully explain what progress means, and why it holds or not. You **do not** need to give a formal proof.
- (b) Does the language TL enjoy the *preservation* property? You should carefully explain what preservation means, and why it holds or not. You **do not** need to give a formal proof.
- (c) Is the language TL *type-safe*? If you think TL is type-safe, you should carefully explain your answer, but you **do not** have to give a formal proof. If you think TL is not type-safe, you should give an example program that demonstrates that type safety is violated.