Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Dr. A. Lochbihler and Prof. Dr. P. Müller

# Formal Methods and Functional Programming

## Exercise Sheet 9: IMP States and Expressions

Submission deadline: Monday, April 28, 2014, 11:00 am

## Assignment 1 (Simplifying State Updates)

(i) Prove that (for all states $\sigma$, variables $x$ and values $v_1, v_2$), $\sigma[x\mapsto v_1][x\mapsto v_2] = \sigma[x\mapsto v_2]$.

(ii) Prove that (for all states $\sigma$, variables $x, y$ and values $v_1, v_2$), if $x \not\equiv y$, then $\sigma[x \mapsto v_1][y \mapsto v_2] = \sigma[y \mapsto v_2][x \mapsto v_1]$. Is the condition $x \not\equiv y$ necessary?

(iii) Prove that for all variables $x$, values $v_1, v_2$, for all natural numbers $m$, for all sequences of length $m$ of variables $\vec{y} \equiv y_1, y_2, \ldots, y_m$ and corresponding values $\vec{v'} \equiv v'_1, v'_2, \ldots, v'_m$, and for all states $\sigma$:
$\sigma[x\mapsto v_1][\vec{y}\mapsto\vec{v'}][x\mapsto v_2] = \sigma[\vec{y}\mapsto\vec{v'}][x\mapsto v_2]$.
**Note**: This exercise is a bit more involved.

Note: these results tell you that you can "clean up" states as you apply additional state updates to them: if many state updates to the same variable have been applied, you can always leave out all except the last one, and you'll still define exactly the same state (part (iii)), and reordering state updates applied to different variables never changes the state (part (ii)).

## Assignment 2 (Substitution Properties)

Consider the substitution operations $a[x \mapsto e]$ on arithmetic expressions and $b[x \mapsto e]$ on boolean expressions, as described in the lecture notes. These substitution operations replace all occurrences of the variable $x$ in the expression $a$ (or $b$) with the arithmetic expression $e$.

In the exercise sessions, we proved the following property for substitution on arithmetic expressions (in which the second $\mapsto$ indicates a *state update* as defined in the lecture notes):

**Substitution lemma for arithmetic expressions** For all arithmetic expressions $e, e'$, for all variables $x$, and all states $\sigma$,

$$\mathcal{A}[\![e[x \mapsto e']]\!]\sigma = \mathcal{A}[\![e]\!]\big(\sigma[x \mapsto \mathcal{A}[\![e']\!]\sigma]\big)\,.$$

Prove the following corresponding substitution property for boolean expressions (which was also mentioned in the lecture), i.e., prove :

For all boolean expressions $b$, all arithmetic expressions $e$, all variables $x$, and all states $\sigma$,

$$\mathcal{B}[\![b[x \mapsto e]]\!]\sigma = \mathcal{B}[\![b]\!]\big(\sigma[x \mapsto \mathcal{A}[\![e]\!]\sigma]\big)\ ,$$

## Assignment 3 (Similar States)

Consider the definition of the *free variables* of an expression, as defined in the lecture. In this question, we show that an expression will always be evaluated the same way in two different states, provided that the states agree on the values assigned to the free variables of the expression.

Formally, prove that:

$$\forall \sigma, \sigma', e.((\forall x.x \in FV(e) \Rightarrow \sigma(x) = \sigma'(x)) \Rightarrow \mathcal{A}[\![e]\!]\sigma = \mathcal{A}[\![e]\!]\sigma')$$

## Assignment 4 (Implementing IMP Expressions)

Implement, using the programming language Haskell, the syntax of the **IMP** expression language as algebraic (Haskell) data types (where `Aexp` and `Bexp` are the datatypes representing arithmetic and boolean expressions respectively). Implement also the semantics of boolean and arithmetic expressions (note that you do not have to implement a parser for **IMP**, but only the data types representing its expression syntax, and the semantics has to deal with these data types). You do not need to implement statements yet; extending these definitions to build an interpreter for IMP will be the topic of later exercises. The signature of the semantic functions for expressions should be

```
evalBexp ::  Bexp -> State -> Bool
```

and

```
evalAexp ::  Aexp -> State -> Integer
```

respectively (where `State` is the datatype representing states). Please email your solution for this assignment to your tutor. The email addresses of the tutors are:

| | |
|---|---|
| Alex Summers | alexander.summers@inf.ethz.ch |
| Milos Novacek | milos.novacek@inf.ethz.ch |
| Uri Juhasz | uri.juhasz@inf.ethz.ch |
| Alex Viand | vianda@student.ethz.ch |
| Cyril Steimer | csteimer@student.ethz.ch |

## Assignment 5 (Substitutions On Expressions)

Substitutions on expressions do not necessarily *commute*; that is, it is not always the case that $e[x \mapsto e_1][y \mapsto e_2] \equiv e[y \mapsto e_2][x \mapsto e_1]$. In general, three extra conditions need to be imposed for substitutions to commute in this way: we need to know that $x \not\equiv y$, that $y \notin FV(e_1)$ and that $x \notin FV(e_2)$. For example, if $x \equiv y$ were allowed, then we could choose $e$ to be $x$, $e_1$ to be the numeral $1$ and $e_2$ to be the numeral $2$, and then $e[x \mapsto e_1][y \mapsto e_2] \equiv 1 \not\equiv 2 \equiv e[y \mapsto e_2][x \mapsto e_1]$.

(i) Show that the condition $y \notin FV(e_1)$ is also necessary.

(ii) Prove that:

$$\begin{aligned}\forall x, y, e_1, e_2.\quad (x \not\equiv y \land y \notin FV(e_1) \land x \notin FV(e_2) \Rightarrow \\ \forall e.\ e[x \mapsto e_1][y \mapsto e_2] \equiv e[y \mapsto e_2][x \mapsto e_1])\end{aligned}$$

You may assume the following lemma (which was proved in your exercise session):

$$\forall e, e', x.\ x \notin FV(e) \Rightarrow e[x \mapsto e'] \equiv e$$

(iii) Consider the following, closely-related result (which states that the *interpretations* of the two expressions will always be the same):

$$\begin{aligned}\forall x, y, e_1, e_2, \sigma \quad (x \not\equiv y \land y \notin FV(e_1) \land x \notin FV(e_2) \Rightarrow \\ \forall e.\ \mathcal{A}[\![e[x \mapsto e_1][y \mapsto e_2]]\!]\sigma = \mathcal{A}[\![e[y \mapsto e_2][x \mapsto e_1]]\!]\sigma)\end{aligned}$$

A simple way to prove this result is to use the result from part (ii) of this question. Find an alternative proof of this result, which does not use part (ii), and which does not require a further induction argument (Hint: consider using the other results mentioned in this exercise sheet).
**Note**: This exercise is a bit more involved.