Technische Universität Wien
Fakultät für Informatik
Assist. Prof. Florian Zuleger
Assist. Prof. Georg Weissenbacher
Yulia Demyanova, Mag.
Moritz Sinn, M.Sc.; Ivan Radiček, M.Sc.

SS 2015
Exercise 5
23rd May 2015

Exercises on Semantics of Programming Languages

Solutions are to be handed in at the lecture on June 8. Later submissions will not be accepted.

<u>Exercise 1</u> Length of a List **(6 Points)**

We encode lists in the untyped λ-calculus (without numbers and booleans) as follows:

- We model the list H::T with *head* H and *tail* T by the lambda term λxy.xHT.
- The empty list *nil* is defined as λxy.y.

Write a λ-term that calculates the length of a list, i.e., give a λ term for the function

$$\text{len} = \lambda\ell.\ \text{IF ISNIL } \ell \text{ THEN } \underline{0} \text{ ELSE SUCC (len (TAIL } \ell))).$$

Hints:

- Use the lambda-terms from the lecture for the numbers (the Church numerals $\underline{0}$, $\underline{1}$, $\underline{2}$, ...).
- Define appropriate lambda-terms for the functions IF-THEN-ELSE, ISNIL, SUCC and TAIL.
- You will need to use the fixed-point λ-term $\Theta := (\lambda xy.y(xxy))(\lambda xy.y(xxy)))$.
- Execute your lambda term for len on an empty and on a list with length two.

<u>Exercise 2</u> Pair and list in simply-typed λ-calculus **(7 Points)**

Extend the syntax, the evaluation and typing rules of the simply-typed λ-calculus for the following constructs.

a) **Pair**. Introduce:

- a new term *pair*, written $\langle t_1, t_2\rangle$
- two new *projection* terms, written $t.1$ for the projection of t to its first argument and $t.2$ for the projection of t to its second argument
- a new type constructor $\tau_1 \times \tau_2$, called the *product* of τ_1 and τ_2.

The semantics of the i-th projection term is to return the i-th element of the pair.

Example. The following program evaluates to 8: $\langle 1, 0\rangle.1 + \langle 2, 3\rangle.2 + \langle 4, 1\rangle.1$

a) **List**. Introduce:

– The following terms:
  * The *empty* list $nil[\tau]$.
  * The list *constructor* $cons[\tau]\ t_1\ t_2$. The list is formed by adding a new element t_1 of type τ to the front of a list t_2 of type $List\ \tau$.
  * The *head* of a list $head[\tau]\ t$, which returns the first element of the list t.
  * The *tail* of a list $tail[\tau]\ t$, which returns the list containing the elements of the list t starting from the second one.
  * The *emptiness* test for lists $isnil[\tau]\ t$ - a boolean predicate, which yields `tt` if the list t is the empty list.

– The type constructor $List\ \tau$. For every type τ, the type $List\ \tau$ describes finite-length lists whose elements are drawn from τ.

Example. Let k be the list $cons[int]\ 1\ (cons[int]\ 2\ (cons[int]\ 3\ nil[int]))$. The following program calculates the sum of the elements of k, i.e., evaluates to 6:

$$(\mathit{fix}\ \lambda f : (List\ \tau \to int) \to (List\ \tau \to int).\lambda \ell : (List\ \tau \to int). \\ \text{if isnil[int] } \ell \text{ then 0 else head[int] } \ell + \text{f (tail[int] } \ell))\ k$$

Exercise 3 Progress and preservation properties **(7 Points)**

For the simply-typed λ-calculus prove the following properties. You can choose which function application semantics to use (call-by-name, call-by-value, or you can prove for both):

- **Progress property**: Suppose M is a closed term and $M : \sigma$. Then either:
  – M is a value
  – or $M \to N$ for some N
- **Preservation property**: If $\Gamma \vdash M : \sigma$ and $M \to N$ then $\Gamma \vdash N : \sigma$.

Hints:

- The progress property is proved by induction on the structure of the typing judgment $\Gamma \vdash M : \sigma$.
- The preservation property is proved by induction on the structure of the term M. You are allowed to use the substitution property without proof:

  Substitution property: $\Gamma, x : \sigma_1 \vdash M : \sigma_2$ and $\Gamma \vdash N : \sigma_1$ implies $\Gamma \vdash M[\![N/x]\!] : \sigma_2$.