DD2552 semteo23: Homework Problem Set 1

Karl Palmskog

Please hand in your individually written solutions by 18:00, September 22, 2023 on Canvas or by email to palmskog@kth.se.

Solutions will be graded A-F. Problems are marked either E or C. All E problems must be solved with only minor errors to receive grade E or higher. C problems also have a number of points, and if you have solved all E problems with only minor errors, the total number of points will determine a grade A-E, as follows:

- 0 - 4 points: E
- 5 - 9 points: D
- 10 - 14 points: C
- 15 - 19 points: B
- 20 - 25 points: A

1. Grammars and inductive relations

a) [E] Define a grammar for basic regular expressions. Follow Harper's book (PFPL) in giving both abstract syntax and concrete syntax for your regular expressions, which must include the following:

   - the void regular expression, matching nothing
   - the unit regular expression, matching the empty string
   - the regular expression matching a single character
   - the regular expression concatenating (the languages of) two other regular expressions
   - the regular expression alternating (the languages of) two other regular expressions

b) [E] Define an inductive relation $s \in lang(r)$ using judgment rules between strings s and regular expressions r that describes when a string matches a regular expression. Define the relation using judgment rules.

c) [C, 5 points] Extend the regular expression grammar with the Kleene star operator representing zero or more concatenations of (the language of) a regular expression. Extend your inductive relation with rules for the Kleene star. Briefly argue why your rules capture the intended meaning of the Kleene star.

2. Combinatory logic and beta reduction

Recall from Harper's lambda calculus note that the **K** combinator is defined as $\lambda x.\lambda y.x$, and the **I** combinator is defined as $\lambda x.x$. We also define the **L** combinator as $\lambda x.\lambda y.x(yy)$.

a) [E] Show using the rules for β-equivalence given by Harper ($\equiv_\beta$) that $(\mathbf{LK})\mathbf{K} \equiv_\beta \mathbf{K}(\mathbf{KK})$. Carefully name each rule you use.

b) [E] The β-reduction judgment $t \prec_\beta t'$ is defined by the same rules as for β-equivalence, but without the rules for reflexivity or symmetry. Fully write out and name the rules for $\prec_\beta$ and prove that $\mathbf{K\,I}, \mathbf{I} \prec_\beta \mathbf{I}$ by providing a derivation tree using your rules. Indicate the name of the rule used in each rule application.

3. Lambda calculus multiplication

Recall from Harper's lambda calculus note the **Y** combinator and definition of **add**.

a) [E] Use the following recursion equations to write a definition of **mulproto**, a "prototype" of a multiplication function **mul** that takes an additional argument–the function to be called in lieu of calling itself. You can use **case**, **succ**, and **add** from Harper's note without defining them.

```
x * 0 = x
x * (y + 1) = x * y + x
```

b) [C, 5 points] Define **mul** as **Y mulproto** and show step-by-step that **mul** $\equiv_\beta$ **mulproto mul**. Explain briefly why this equivalence is useful.

4. Binary tree extension

Consider a binary tree as a data type that doesn't store any values. In CakeML, this can be defined as follows:

```
datatype btree = Leaf | Branch btree btree
```

a) [E] Follow Harper's approach with natural numbers from PFPL Chapter 9 and define the syntax for the T language extended with binary trees, including syntax for trees themselves (and tree types) and a recursor for trees.

b) [E] Provide statics for your extended T language, i.e., provide a typing judgment/relation.

c) [E] Provide dynamics for your extended T language, i.e., provide a reduction judgment/relation.

d) [C, 8 points] Specify and sketch a proof of safety for your extended language, similar to Harper's Theorem 9.3.

5. Recursive data types and functions

Consider Harper's natural numbers from PFPL Chapter 9.

a) [E] Encode natural numbers as an ML-style datatype using CakeML's datatype declaration syntax.

b) [E] Define addition as a recursive function `add` on the natural number datatype defined in a) using Harper's recursion equations from the note about Lambda caculus as a guide. Use CakeML's function definition syntax.

c) [E] Define multiplication as a recursive function `mul` on the natural number datatype using the recursion equations above as a guide, calling the addition function defined before. Use CakeML's function definition syntax.

d) [C, 2 points] Define a "truncated subtraction" function taking two natural numbers n and m, returning 0 if n is less than or equal to m, and the usual $n - m$ otherwise. Use CakeML's function definition syntax.

6. General recursive function

Consider a function `f` on lists defined by the following equations:

```
f nil l2 = l2
f (h1::t1) l2 = h1::(f l2 t1)
```

a) [E] Describe briefly in words what the function computes.

b) [E] Provide a version of the function that is structurally recursive. Sketch an argument that the original function and your function are extensionally equal (return the same result for the same input).

c) [E] Define a well-founded relation on 2-tuples of lists (a "measure" on the input) to demonstrate that the original function terminates.

d) [C, 5 points] Prove carefully that the well-founded relation on input you defined is indeed well-founded, using the conventional definition of well-foundedness as absence of infinitely descending chains.