Fundamentals of MCS: CS - Part 2

NGUYEN T. Hoang - SID: 15M54097 | Fall 2015, W832 Mon. Period 7-8

Due date: 2016/01/18

Problem

For this Part II of Fall 2015 Fundamentals of Mathematical and Computing Sciences: Computer Science class, I choose **Assignment 3** for submission.

Q3.1. We would like to port the compiler to another stack machine whose behavior is slightly different from the original one. Although the representation of its structure remains the same (`Definition` *prog* := *list instr* and `Definition` *stack* := *list nat*), the new stack machine's interpretation of instructions is slightly different:

```
Definition instrDenote' (i : instr) (s : stack) : option stack :=
  match i with
  | iConst n ⇒ Some (n :: s)
  | iBinop b ⇒ match s with
                | arg2 :: arg1 :: s' ⇒ Some ((binopDenote b) arg1 arg2 :: s')
                | _ ⇒ None
                end
  end.
```

The instrDenote' function assumes that the second operand at the stack top while instrDenote assumes the first one at the top. Given this modified instrDenote' function, try to modifiy the implementation of the compiler so that it suits the new definition and prove its correctness.

Q3.2. Extend your implementation of **Q3.1** to add Minus operator to binop and adjust definitions of denotations, the compiler, appropriately and complete the proof.

Answer:

Q3.1 - Modified Stack Machine.

Since we are given new *instrDenote'* function, I am going to change the *compile* and *progDenote* function into *compile'* and *progDenote'* function that accept the new definition of *instrDenote'*. The new functions are defined as follow:

```
Fixpoint progDenote' (p : prog) (s : stack) : option stack :=
  match p with
  | nil ⇒ Some s
  | i :: p' ⇒ match instrDenote' i s with
                | None ⇒ None
                | Some s' ⇒ progDenote' p' s'
                end
  end.
```

```
Fixpoint compile' (e : exp) : prog :=
  match e with
  | Const n ⇒ iConst n :: nil
  | Binop b e1 e2 ⇒ (compile e1) ++ (compile e2) ++ (iBinop b :: nil)
  end.
```

Before going to the proof, I would like to test out the new Stack Machine with few examples of program evaluation and compiler evaluation:

```
Eval simpl in progDenote' (compile' (Const 3)) nil.
  = Some (3 :: nil) : option stack
```

```
Eval simpl in progDenote' (compile' (Binop Plus (Const 3) (Const 4))) nil.
  = Some (7 :: nil) : option stack
```

```
Eval simpl in progDenote' (compile' (Binop Times
                (Binop Plus (Const 3) (Const 4))
                (Binop Plus (Const 5) (Const 6)))) nil.
  = Some (77 :: nil) : option stack
```

```
Eval simpl in compile' (Binop Times (Binop Plus (Const 2) (Const 3)) (Const 7)).
  = iConst 3 :: iConst 2 :: iBinopPlus :: iConst 7 :: iBinop Times :: nil : prog
```

Our modified compiler should work with *all* input, therefore we have the compile'_correct theorem as follow:

```
Theorem compile'_correct : ∀ e, progDenote' (compile' e) nil = Some (expDenote e :: nil).
```

To prove this theorem, as in [?], I will use the standard trick of *strengthening the induction hypothesis*. By proving the fact that, given *any* expression, program list state, and stack state, the modified compiler will correctly compile the program to run with progDenote'.

Lemma compile'_correct' : $\forall$ e p s,
progDenote' (compile' e ++ p) s = progDenote' p (expDenote e :: s).

Firstly I use `intros` tactic to handle the "$\forall$" condition. We have: `intros`.

1 `subgoal`

e : *exp*
p : *list instr*
s : *stack*
============================
progDenote' (compile' e ++ p) s = progDenote' p (expDenote e :: s)
]] *)

Using induction on e, we will have 2 subgoals corresponding to 2 cases of e: *Const n* and *Binop b e1 e2*. `induction` e.

2 `subgoals`

n : *nat*
p : *list instr*
s : *stack*
============================
progDenote' (compile' (Const n) ++ p) s =
progDenote' p (expDenote (Const n) :: s)

`subgoal` 2 *is*:
progDenote' (compile' (Binop b e1 e2) ++ p) s =
progDenote' p (expDenote (Binop b e1 e2) :: s)

`Abort`.

Q1.1 - Regular language definition with $\approx_L$.

My proof contains two parts: