CS 496 – Quiz 2
Deadline: 23 February, 11:59PM

21 February 2020

Exercise 1

This exercise proposes LET+TUPLES, an extension of the LET-language with tuples.

Concrete Syntax. Its concrete syntax is given below:

<Program>	::=	<Expression>
<Expression>	::=	<Number> |<Identifier> |<Expression> - <Expression>
<Expression>	::=	zero? (<Expression>)
<Expression>	::=	if <Expression> then <Expression> else <Expression>
<Expression>	::=	let <Identifier> = <Expression> in <Expression>
<Expression>	::=	< <Expression>$^{+/,}$ >
<Expression>	::=	untuple < <Identifier>$^{+/,}$ > = <Expression> in <Expression>

Only the last two productions in the grammar are new, the others are part of LET.

- Tuples are constructed by listing its components as comma-separated expressions enclosed in angle brackets. There must be at least one component.
- The expression `untuple <x1,...,xn>=e1 in e2` evaluates `e1`, makes sure it is a tuple of n values, say v1 to vn, and then evaluates `e2` in the extended environment where each `xi` is bound to `vi`. Variables `x1`,...,`xn` must be distinct.

Abstract Syntax. Only the last two variants are new the others are part of LET:

```
type expr =
  | Var of string
  | Int of int
  | Sub of expr*expr
  | Let of string*expr*expr
  | IsZero of expr
  | ITE of expr*expr*expr
  | Tuple of expr list
  | Untuple of string list * expr*expr
```

The Interpreter. Examples of programs in LET+TUPLES and the result of evaluating them are:

1. `<2,3,4>` evaluates to `Ok (TupleVal [NumVal 2; NumVal 3; NumVal 4])`.
2. `<2,3,zero?(0)>` evaluates to `Ok (TupleVal [NumVal 2; NumVal 3; BoolVal true])`.

3. `<<7,9>,3>` evaluates to `Ok (TupleVal [TupleVal [NumVal 7; NumVal 9]; NumVal 3])`.

4. `<zero?(4),11-2>` evaluates to `Ok (TupleVal [BoolVal false; NumVal 9])`.

5. `untuple <x,y,z> = <3, <5 , 12>,4> in x` evaluates to `Ok (NumVal 3)`.

6. `let x = 34 in untuple <y,z>=<3,x> in y+z` evaluates to `Ok (NumVal 37)`.

7. `untuple <x,y,z> = <3,4> in x` evaluates to `Error "extend_env_list: No.of args does not match no. of params!"`.

8. `untuple <x> = <3,4> in x` evaluates to `Error "extend_env_list: No.of args does not match no. of params! ''`

Implementation of the Interpreter. The set of expressed values has already been extended for you (file `ds.ml`):

```
type exp_val =
  | NumVal of int
  | BoolVal of bool
  | UnitVal
  | TupleVal of exp_val list (* new *)
```

Extend the interpreter for LET to LET+TUPLES, so `eval_expr` is capable of executing expressions involving `tuple` and `untuple`. You may define helper functions. These will most likely have to be added to file `ds.ml`.

```
let rec eval_expr : expr -> exp_val ea_result = fun e ->
  match e with
  | Int(n) -> return @@ NumVal n
  | Var(id) -> apply_env id
  ...
  | Tuple(es) -> error "implement"
  | Untuple(ids,e1,e2) -> error "implement"
```

Hints.

1. For the untuple case you will need to extend the environment with a list of variable/-value bindings. You can use the following helper function in file `ds.ml` (the `lookup_env` function is also defined in `ds.ml`):

```
let rec extend_env_list : string list -> exp_val list -> env ea_result =
    fun ids evs ->
  match ids,evs  with
  | [],[] -> lookup_env
  | id::ids, ev::evs when List.mem id ids -> error "duplicate identifiers
      "
  | id::ids, ev::evs -> extend_env id ev >>+ extend_env_list ids evs
  | _,_ -> error "extend_env_list: Arguments do not match parameters!"
```

Submission. At most two members per group. Via canvas. Submit an archive with all the files in the stub. Name the archive `q2.zip`. Place name of other team member as canvas comment.