Seminar 8 - Analiză semantică - Exerciții rezolvate

Matei Barbu

11 decembrie 2022

Cuprins

Exercițiul 1

Analiză statică:

```
Class Bazz inherits Object { };
Class Bar inherits Bazz { };
Class Foo inherits Bar { };

-- Ex1. Care sunt tipurile celor 3 expresii,
-- daca avem ierarhia de clase definita mai sus
{
  case 3 of y: Int => y; z: Bool => z; esac; -- Tip = lub(Int, Bool) = Object

  if 0 = 1 then new Bar else new Foo fi; -- Tip = Bar

  let x: Bazz new Bar in x; -- Tip = Bazz (chiar dacă x e un Bar, el rămâne ce era la
  ↪  început)
}
```

Ierarhia de clase:

$$Foo \leq Barr \leq Bazz \leq Object$$

Exercițiul 2-3

```
-- Ex2. Ce va afisa urmatorul program daca este "statically scoped"?  -- Raspuns
-- Ex3. Ce va afisa urmatorul program daca este "dynamically scoped"?  -- Raspuns

Class Main inherits IO {
  x : Int 5;
```

```
  foo(z:Int) : Int {
    x+z
  };

  bar(y:Int) : Int {
    {
      let x : Int 1 in
        let z : Int 2 in
          foo(y);
    }
  };

  main(): Object {
    {
      let x : Int 7 in
        out_int(foo(bar(3)));
    }
  };
}
```

Statically scoped

Cine e `foo(bar(3))`?

- `bar(3) = foo(3) = x (5) + z (3) = 8`
- `foo(bar(3)) = 5 + 8 = 13`

Dynamically scoped

> Tot ce întâlnești pe parcurs e pus într-o stivă imaginară.

Cine e `foo(bar(3))`?

```
{x : 5} -- stiva de simboluri, în care căutăm de la vârf (dreapta) la fund (stânga) un
↪  simbol
main()
    {x : 5, x : 7}
    out_int(foo(bar(3)))
      foo(bar(3)) => x (7) + bar(3)
        {x : 5, x : 7, y : 3}
        bar(3)
          {x : 5, x : 7, y : 3, x : 1, z : 2, z : 3}
          foo(3) => x (1) + z (3) = 4
      foo(bar(3)) => x (7) + bar(3) = 11
```

Exercițiul 4-5

```
-- Ex5. Ce va afisa programul? Mai intai rezolvam ex4.
class Main {
  main() : Object {
    (new Bar).bar()
  };
};

class Foo inherits IO {
  foo() : SELF_TYPE {
    {
      out_string("Foo.foo()\n");
```

```
      foo();
      self;
    }
  };
};

class Bar inherits Foo {
  foo() : SELF_TYPE {
    {
      out_string("Bar.foo()\n");
      new SELF_TYPE;
    }
  };

  -- Ex4. Ce tip s-ar potrivi pentru valoarea returnata de
  -- aceasta functie daca avem "static type checking"?
  bar() : (*???????????????*) {
    case foo() of
      f : Foo => f@Foo.foo();
      b : Bar => (new Bazz).foo();
      o : Object => foo();
    esac
  };
};

class Bazz inherits Bar {
  foo() : SELF_TYPE {
    {
      out_string("Bazz.foo()\n");
      (new Bar)@Foo.foo();
      self;
    }
  };
};
```

Tipul returnat de funcție?

Ierarhia de clase:

$$Bazz \leq Barr \leq Foo \leq IO \leq Object$$

$$\texttt{foo()}_{\texttt{bar()}_{Bar}} = lub \left(\begin{cases} Foo & \text{cazul } \texttt{Foo} \\ Bazz & \text{cazul } \texttt{Bar} \\ \underbrace{SELF\_TYPE_{Bar}}_{\text{tipul din analiză statică}} & \text{cazul } \texttt{Object} \end{cases} \right) = \underbrace{Foo}_{\text{cel mai general tip}}$$

Ce afișează programul?

```
=> main()
(new Bar).bar()
  => bar() :: Bar
  case foo() of
    => foo() :: Bar
      out_string("Bar.foo()\n");
      new SELF_TYPE; -- Tip dinamic : Bar
    (new Bazz).foo()
      => foo() :: Bazz
        out_string("Bazz.foo()\n");
```

```
        (new Bar)@Foo.foo();
          => foo :: Foo
            out_string("Foo.foo()\n");
            foo(); -- tipul dinamic a lui self: Bar
            => foo() :: Bar
              out_string("Bar.foo()\n");
... -- nu mai urmează afișări ulterioare
```

Exercițiul 9

Pentru expresia $(7 + 5) * (3 + 2)$

Care dintre următoarele pot fi stări ale mașinii cu stivă și acumulator în timpul evaluării?

- ☒ `acc`: 5, `stack`: 7, init
- ☐ `acc`: 3, `stack`: 2, 7, 5, init
- ☐ `acc`: 5, `stack`: 7, 5, init
- ☒ `acc`: 3, `stack`: 12, init
- ☐ `acc`: 5, `stack`: init
- ☐ `acc`: 12, `stack`: 5, init

Intrucțiuni acceptate:

```
acc <- value
push acc
pop
acc <- acc + top
```

> Preferabil cât mai puțină memorie folosită și instrucțiuni. Întotdeauna când o operație se termină o și reducem.

Cod

```
acc <- "7 + 5"  | acc <- 7
                | push acc
                | acc <- 5
                | acc <- acc + top
                | pop

push acc

acc <- "3 + 2"  | acc <- 3
                | push acc
                | acc <- 2
                | acc <- acc + top
                | pop

acc <- acc + top
pop
```

Exercițiul 10

În cazul în care am avea o mașină cu stivă și un registru, care dintre următoarele afirmații sunt adevărate.

- ☒ Computing expressions $1 + 2 + 3 + ... + n$ and $1 + (2 + (3 + (... + n)))$ require the same number of `acc <- acc + top_of_stack` actions.
- ☒ Computing the expression $1 + 2 + 3 + ... + n$ requires $n - 1$ `push` actions.
- ☐ Computing the expression $1 + (2 + (3 + (... + n)))$ requires n `pop` actions.
- ☐ Computing the expression $1 + 2 + 3 + ... + n$ requires 1 `push` action.

☒ Computing expressions $1+2+3+...+n$ and $1+(2+(3+(...+n)))$ require dierent amounts of stack space.

- The left associative form requires n-1 `push`es. This number is equal to the number of `+`es in the expression, for `x + y` requires a `push` regardless of the form `x` and `y` have.
- The right expression requires the same number of `push`s. It only differs what it pushes. The left expression pushed partial sums on the stack, while the right one pushes the terms themselves.
- Because of the invariant condition (the stack must hold the same contents when the evaluation of a expression ended as it held when the evalution began) the number of `push`es = the number of `pop`s.