Compiler Construction

Escaping variables

Variable escape analysis phase

Variable escape analysis phase

The objective of this semantic phase is to find out if each variable escapes or not.

This phase is:

- required for the translation into intermediate code
- (one of) the last time we can compute this information that is source related

Escaping Variable

Technically escaping means “cannot be stored in a register”.

In C
- Large values (arrays, structs).
- Variables whose address is taken.
- Variable arguments.

In Tiger
- Variables/Variables accessed from a nested function
- Arguments accessed from a nested scope
  $\Rightarrow$ Non-local variable

Example

```
let
  var one := 1
  var two := 2
  function incr (x: int) : int =
      x + one
in
  incr (two)
end
```

Annotating the AST

- The translation to intermediate representation needs to know which variables are non local from their definitions
- Therefore a preleminary pass should flag non local variables

Conservative approach

Consider all variables as escaping, since it is safe to put a non escaping variable onto the stack, while the converse is unsafe!

```
let
  var /*escaping*/ one := 1
  var /*escaping*/ two := 2
  function incr (/*escaping*/ x: int) : int =
      x + one
in
  incr (two)
end
```

Computing escapes

1. Tag all variables as non-escaping
2. Detect and tag all escaping variables
3. Only need to know the depth of the declaration's scope

```
let
  var /*escaping*/ one := 1
  var              two := 2
  function incr (x: int) : int =
      x + one
in
  incr (two)
end
```

The Escapes & Recursion

```
let
  function one(input : int) =
    let
      function two() =
        (print("two: "); print_int(input);
         print("\n");
         one(input))
    in
      if input > 0 then
        (input := input - 1;
         two(); print("one: ");
         print_int(input); print("\n"))
    end
in
  one (3)
end
```

The Escapes and Functional Programming

This IS valid Tiger!

```
let
  function add(/*nonlocal*/ a: int, b: int) : int =
    let
      function add_a(x: int) : int = a + x
    in
      add_a(b)
    end
in
  print_int(add(1, 2));
  print("\n")
end
```

The Escapes and Functional Programming

This is NOT valid Tiger!

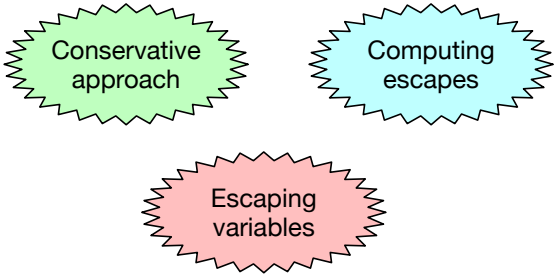
```
let
  function add(/*nonlocal*/ a: int) : int =
    let
      function add_a(x: int) : int = a + x
    in
      add_a
    end
in
  let var a1 := add(1)
  in
    print_int( a1(2) ); print("\n")
  end
end
```

The Escapes and Functional Programming

This is NOT valid Tiger!

```
let
  function add(/*nonlocal*/ a: int) : int =
    let
      function add_a(x: int) : int = a + x
    in
      add_a
    end
in
  let var a1 := add(1)
      var a2 := add(2)
  in
    print_int( a1(2) ); print("\n")
    print_int( a2(2) ); print("\n")
  end
end
```

Summary


Conservative approach
Computing escapes
Escaping variables